

Rafael A. Irizarry

Introduction to Data Science

Contents

Preface	23
Acknowledgments	25
Introduction	27
1 Getting started with R and RStudio	29
1.1 Why R?	29
1.2 The R console	29
1.3 Scripts	30
1.4 RStudio	31
1.4.1 The panes	31
1.4.2 Key bindings	33
1.4.3 Running commands while editing scripts	34
1.4.4 Changing global options	36
1.5 Installing R packages	36
I R	39
2 R basics	41
2.1 Case study: US Gun Murders	41
2.2 The very basics	43
2.2.1 Objects	43
2.2.2 The workspace	43
2.2.3 Functions	44
2.2.4 Other prebuilt objects	46
2.2.5 Variable names	47
2.2.6 Saving your workspace	47
2.2.7 Motivating scripts	47
2.2.8 Commenting your code	48

2.3	Exercises	48
2.4	Data types	49
2.4.1	Data frames	49
2.4.2	Examining an object	49
2.4.3	The accessor: <code>\$</code>	50
2.4.4	Vectors: numerics, characters, and logical	51
2.4.5	Factors	52
2.4.6	Lists	52
2.4.7	Matrices	54
2.5	Exercises	55
2.6	Vectors	56
2.6.1	Creating vectors	56
2.6.2	Names	57
2.6.3	Sequences	58
2.6.4	Subsetting	58
2.7	Coercion	59
2.7.1	Not availables (NA)	60
2.8	Exercises	60
2.9	Sorting	61
2.9.1	<code>sort</code>	61
2.9.2	<code>order</code>	61
2.9.3	<code>max</code> and <code>which.max</code>	62
2.9.4	<code>rank</code>	63
2.9.5	Beware of recycling	63
2.10	Exercises	64
2.11	Vector arithmetics	65
2.11.1	Rescaling a vector	65
2.11.2	Two vectors	65
2.12	Exercises	66
2.13	Indexing	66
2.13.1	Subsetting with logicals	67
2.13.2	Logical operators	67
2.13.3	<code>which</code>	68
2.13.4	<code>match</code>	68

0.0 Contents	5
2.13.5 <code>%in%</code>	69
2.14 Exercises	69
2.15 Basic plots	70
2.15.1 <code>plot</code>	70
2.15.2 <code>hist</code>	70
2.15.3 <code>boxplot</code>	71
2.15.4 <code>image</code>	72
2.16 Exercises	72
3 Programming basics	73
3.1 Conditional expressions	73
3.2 Defining functions	75
3.3 Namespaces	76
3.4 For-loops	77
3.5 Vectorization and functionals	78
3.6 Exercises	79
4 The tidyverse	81
4.1 Tidy data	81
4.2 Exercises	82
4.3 Manipulating data frames	83
4.3.1 Adding a column with <code>mutate</code>	83
4.3.2 Subsetting with <code>filter</code>	84
4.3.3 Selecting columns with <code>select</code>	84
4.4 Exercises	85
4.5 The pipe: <code>%>%</code>	86
4.6 Exercises	87
4.7 Summarizing data	88
4.7.1 <code>summarize</code>	88
4.7.2 <code>pull</code>	90
4.7.3 Group then summarize with <code>group_by</code>	91
4.8 Sorting data frames	92
4.8.1 Nested sorting	93
4.8.2 The top n	93
4.9 Exercises	93

4.10	Tibbles	95
4.10.1	Tibbles display better	95
4.10.2	Subsets of tibbles are tibbles	96
4.10.3	Tibbles can have complex entries	96
4.10.4	Tibbles can be grouped	97
4.10.5	Create a tibble using <code>tibble</code> instead of <code>data.frame</code>	97
4.11	The dot operator	97
4.12	<code>do</code>	98
4.13	The <code>purrr</code> package	100
4.14	Tidyverse conditionals	101
4.14.1	<code>case_when</code>	101
4.14.2	<code>between</code>	102
4.15	Exercises	102
5	Importing data	103
5.1	Paths and the working directory	104
5.1.1	The filesystem	104
5.1.2	Relative and full paths	105
5.1.3	The working directory	105
5.1.4	Generating path names	106
5.1.5	Copying files using paths	106
5.2	The <code>readr</code> and <code>readxl</code> packages	107
5.2.1	<code>readr</code>	107
5.2.2	<code>readxl</code>	108
5.3	Exercises	108
5.4	Downloading files	109
5.5	R-base importing functions	110
5.5.1	<code>scan</code>	110
5.6	Text versus binary files	111
5.7	Unicode versus ASCII	111
5.8	Organizing data with spreadsheets	112
5.9	Exercises	112
II	Data Visualization	113
6	Introduction to data visualization	115

0.0 Contents	7
7 ggplot2	119
7.1 The components of a graph	120
7.2 ggplot objects	121
7.3 Geometries	122
7.4 Aesthetic mappings	123
7.5 Layers	124
7.5.1 Tinkering with arguments	125
7.6 Global versus local aesthetic mappings	126
7.7 Scales	127
7.8 Labels and titles	128
7.9 Categories as colors	129
7.10 Annotation, shapes, and adjustments	130
7.11 Add-on packages	131
7.12 Putting it all together	132
7.13 Quick plots with <code>qplot</code>	133
7.14 Grids of plots	134
7.15 Exercises	134
8 Visualizing data distributions	137
8.1 Variable types	137
8.2 Case study: describing student heights	138
8.3 Distribution function	138
8.4 Cumulative distribution functions	139
8.5 Histograms	140
8.6 Smoothed density	141
8.6.1 Interpreting the y-axis	145
8.6.2 Densities permit stratification	146
8.7 Exercises	146
8.8 The normal distribution	150
8.9 Standard units	152
8.10 Quantile-quantile plots	153
8.11 Percentiles	155
8.12 Boxplots	155
8.13 Stratification	157
8.14 Case study: describing student heights (continued)	157

8.15	Exercises	159
8.16	ggplot2 geometries	160
8.16.1	Barplots	161
8.16.2	Histograms	162
8.16.3	Density plots	163
8.16.4	Boxplots	164
8.16.5	QQ-plots	164
8.16.6	Images	165
8.16.7	Quick plots	166
8.17	Exercises	168
9	Data visualization in practice	169
9.1	Case study: new insights on poverty	169
9.1.1	Hans Rosling's quiz	170
9.2	Scatterplots	171
9.3	Faceting	172
9.3.1	<code>facet_wrap</code>	174
9.3.2	Fixed scales for better comparisons	175
9.4	Time series plots	175
9.4.1	Labels instead of legends	178
9.5	Data transformations	179
9.5.1	Log transformation	179
9.5.2	Which base?	181
9.5.3	Transform the values or the scale?	182
9.6	Visualizing multimodal distributions	183
9.7	Comparing multiple distributions with boxplots and ridge plots	183
9.7.1	Boxplots	184
9.7.2	Ridge plots	185
9.7.3	Example: 1970 versus 2010 income distributions	187
9.7.4	Accessing computed variables	193
9.7.5	Weighted densities	196
9.8	The ecological fallacy and importance of showing the data	196
9.8.1	Logistic transformation	197
9.8.2	Show the data	197

<i>0.0 Contents</i>	9
10 Data visualization principles	199
10.1 Encoding data using visual cues	199
10.2 Know when to include 0	202
10.3 Do not distort quantities	205
10.4 Order categories by a meaningful value	207
10.5 Show the data	208
10.6 Ease comparisons	211
10.6.1 Use common axes	211
10.6.2 Align plots vertically to see horizontal changes and horizontally to see vertical changes	212
10.6.3 Consider transformations	213
10.6.4 Visual cues to be compared should be adjacent	215
10.6.5 Use color	216
10.7 Think of the color blind	216
10.8 Plots for two variables	217
10.8.1 Slope charts	217
10.8.2 Bland-Altman plot	219
10.9 Encoding a third variable	219
10.10 Avoid pseudo-three-dimensional plots	221
10.11 Avoid too many significant digits	223
10.12 Know your audience	224
10.13 Exercises	224
10.14 Case study: vaccines and infectious diseases	229
10.15 Exercises	232
11 Robust summaries	233
11.1 Outliers	233
11.2 Median	234
11.3 The inter quartile range (IQR)	234
11.4 Tukey's definition of an outlier	235
11.5 Median absolute deviation	236
11.6 Exercises	236
11.7 Case study: self-reported student heights	237
III Statistics with R	241

12 Introduction to statistics with R	243
13 Probability	245
13.1 Discrete probability	245
13.1.1 Relative frequency	245
13.1.2 Notation	246
13.1.3 Probability distributions	246
13.2 Monte Carlo simulations for categorical data	246
13.2.1 Setting the random seed	248
13.2.2 With and without replacement	248
13.3 Independence	249
13.4 Conditional probabilities	249
13.5 Addition and multiplication rules	250
13.5.1 Multiplication rule	250
13.5.2 Multiplication rule under independence	250
13.5.3 Addition rule	251
13.6 Combinations and permutations	251
13.6.1 Monte Carlo example	255
13.7 Examples	255
13.7.1 Monty Hall problem	256
13.7.2 Birthday problem	257
13.8 Infinity in practice	259
13.9 Exercises	260
13.10 Continuous probability	262
13.11 Theoretical continuous distributions	263
13.11.1 Theoretical distributions as approximations	263
13.11.2 The probability density	265
13.12 Monte Carlo simulations for continuous variables	266
13.13 Continuous distributions	267
13.14 Exercises	267
14 Random variables	269
14.1 Random variables	269
14.2 Sampling models	270
14.3 The probability distribution of a random variable	271

<i>0.0 Contents</i>	11
14.4 Distributions versus probability distributions	273
14.5 Notation for random variables	273
14.6 The expected value and standard error	274
14.6.1 Population SD versus the sample SD	276
14.7 Central Limit Theorem	277
14.7.1 How large is large in the Central Limit Theorem?	278
14.8 Statistical properties of averages	278
14.9 Law of large numbers	280
14.9.1 Misinterpreting law of averages	280
14.10 Exercises	280
14.11 Case study: The Big Short	282
14.11.1 Interest rates explained with chance model	282
14.11.2 The Big Short	285
14.12 Exercises	288
15 Statistical inference	289
15.1 Polls	289
15.1.1 The sampling model for polls	290
15.2 Populations, samples, parameters, and estimates	292
15.2.1 The sample average	292
15.2.2 Parameters	293
15.2.3 Polling versus forecasting	293
15.2.4 Properties of our estimate: expected value and standard error	294
15.3 Exercises	295
15.4 Central Limit Theorem in practice	296
15.4.1 A Monte Carlo simulation	297
15.4.2 The spread	299
15.4.3 Bias: why not run a very large poll?	299
15.5 Exercises	300
15.6 Confidence intervals	302
15.6.1 A Monte Carlo simulation	304
15.6.2 The correct language	305
15.7 Exercises	305
15.8 Power	306
15.9 p-values	307

15.10 Association tests	308
15.10.1 Lady Tasting Tea	309
15.10.2 Two-by-two tables	310
15.10.3 Chi-square Test	310
15.10.4 The odds ratio	311
15.10.5 Confidence intervals for the odds ratio	312
15.10.6 Small count correction	313
15.10.7 Large samples, small p-values	313
15.11 Exercises	314
16 Statistical models	315
16.1 Poll aggregators	316
16.1.1 Poll data	318
16.1.2 Pollster bias	320
16.2 Data-driven models	321
16.3 Exercises	323
16.4 Bayesian statistics	326
16.4.1 Bayes theorem	326
16.5 Bayes theorem simulation	327
16.5.1 Bayes in practice	328
16.6 Hierarchical models	329
16.7 Exercises	331
16.8 Case study: election forecasting	333
16.8.1 Bayesian approach	334
16.8.2 The general bias	335
16.8.3 Mathematical representations of models	335
16.8.4 Predicting the electoral college	338
16.8.5 Forecasting	342
16.9 Exercises	345
16.10 The t-distribution	346
17 Regression	349
17.1 Case study: is height hereditary?	349
17.2 The correlation coefficient	350
17.2.1 Sample correlation is a random variable	352

0.0 Contents	13
17.2.2 Correlation is not always a useful summary	354
17.3 Conditional expectations	354
17.4 The regression line	357
17.4.1 Regression improves precision	358
17.4.2 Bivariate normal distribution (advanced)	359
17.4.3 Variance explained	361
17.4.4 Warning: there are two regression lines	361
17.5 Exercises	362
18 Linear models	363
18.1 Case study: Moneyball	363
18.1.1 Sabermetrics	364
18.1.2 Baseball basics	365
18.1.3 No awards for BB	366
18.1.4 Base on balls or stolen bases?	367
18.1.5 Regression applied to baseball statistics	369
18.2 Confounding	372
18.2.1 Understanding confounding through stratification	373
18.2.2 Multivariate regression	376
18.3 Least squares estimates	376
18.3.1 Interpreting linear models	377
18.3.2 Least Squares Estimates (LSE)	377
18.3.3 The <code>lm</code> function	379
18.3.4 LSE are random variables	380
18.3.5 Predicted values are random variables	381
18.4 Exercises	382
18.5 Linear regression in the tidyverse	383
18.5.1 The broom package	386
18.6 Exercises	387
18.7 Case study: Moneyball (continued)	388
18.7.1 Adding salary and position information	392
18.7.2 Picking nine players	393
18.8 The regression fallacy	395
18.9 Measurement error models	396
18.10 Exercises	399

19 Association is not causation	401
19.1 Spurious correlation	401
19.2 Outliers	404
19.3 Reversing cause and effect	406
19.4 Confounders	407
19.4.1 Example: UC Berkeley admissions	407
19.4.2 Confounding explained graphically	408
19.4.3 Average after stratifying	409
19.5 Simpson's paradox	410
19.6 Exercises	411
IV Data Wrangling	413
20 Introduction to data wrangling	415
21 Reshaping data	417
21.1 gather	417
21.2 spread	419
21.3 separate	419
21.4 unite	422
21.5 Exercises	423
22 Joining tables	425
22.1 Joins	426
22.1.1 Left join	427
22.1.2 Right join	428
22.1.3 Inner join	428
22.1.4 Full join	428
22.1.5 Semi join	429
22.1.6 Anti join	429
22.2 Binding	430
22.2.1 Binding columns	430
22.2.2 Binding by rows	430
22.3 Set operators	431
22.3.1 Intersect	431
22.3.2 Union	432

<i>0.0 Contents</i>	15
22.3.3 <code>setdiff</code>	432
22.3.4 <code>setequal</code>	432
22.4 Exercises	433
23 Web scraping	435
23.1 HTML	436
23.2 The <code>rvest</code> package	437
23.3 CSS selectors	439
23.4 JSON	440
23.5 Exercises	441
24 String processing	443
24.1 The <code>stringr</code> package	443
24.2 Case study 1: US murders data	445
24.3 Case study 2: self-reported heights	447
24.4 How to <i>escape</i> when defining strings	449
24.5 Regular expressions	451
24.5.1 Strings are a regexp	451
24.5.2 Special characters	451
24.5.3 Character classes	453
24.5.4 Anchors	454
24.5.5 Quantifiers	454
24.5.6 White space <code>\s</code>	455
24.5.7 Quantifiers: <code>*</code> , <code>?</code> , <code>+</code>	456
24.5.8 Not	456
24.5.9 Groups	457
24.6 Search and replace with regex	458
24.6.1 Search and replace using groups	460
24.7 Testing and improving	461
24.8 Trimming	463
24.9 Changing lettercase	464
24.10 Case study 2: self-reported heights (continued)	464
24.10.1 The <code>extract</code> function	465
24.10.2 Putting it all together	466
24.11 String splitting	467

24.12	Case study 3: extracting tables from a PDF	470
24.13	Recoding	473
24.14	Exercises	474
25	Parsing dates and times	477
25.1	The date data type	477
25.2	The lubridate package	478
25.3	Exercises	481
26	Text mining	483
26.1	Case study: Trump tweets	483
26.2	Text as data	485
26.3	Sentiment analysis	490
26.4	Exercises	495
V	Machine Learning	497
27	Introduction to machine learning	499
27.1	Notation	499
27.2	An example	500
27.3	Exercises	502
27.4	Evaluation metrics	502
27.4.1	Training and test sets	503
27.4.2	Overall accuracy	504
27.4.3	The confusion matrix	506
27.4.4	Sensitivity and specificity	507
27.4.5	Balanced accuracy and F_1 score	509
27.4.6	Prevalence matters in practice	510
27.4.7	ROC and precision-recall curves	511
27.4.8	The loss function	512
27.5	Exercises	514
27.6	Conditional probabilities and expectations	514
27.6.1	Conditional probabilities	515
27.6.2	Conditional expectations	516
27.6.3	Conditional expectation minimizes squared loss function	516
27.7	Exercises	517
27.8	Case study: is it a 2 or a 7?	517

<i>0.0 Contents</i>	17
28 Smoothing	521
28.1 Bin smoothing	523
28.2 Kernels	525
28.3 Local weighted regression (loess)	526
28.3.1 Fitting parabolas	530
28.3.2 Beware of default smoothing parameters	531
28.4 Connecting smoothing to machine learning	532
28.5 Exercises	532
29 Cross validation	535
29.1 Motivation with k-nearest neighbors	535
29.1.1 Over-training	537
29.1.2 Over-smoothing	538
29.1.3 Picking the k in kNN	539
29.2 Mathematical description of cross validation	541
29.3 K-fold cross validation	542
29.4 Exercises	545
29.5 Bootstrap	546
29.6 Exercises	549
30 The caret package	551
30.1 The caret <code>train</code> function	551
30.2 Cross validation	552
30.3 Example: fitting with loess	554
31 Examples of algorithms	557
31.1 Linear regression	557
31.1.1 The <code>predict</code> function	558
31.2 Exercises	559
31.3 Logistic regression	561
31.3.1 Generalized linear models	562
31.3.2 Logistic regression with more than one predictor	566
31.4 Exercises	567
31.5 k-nearest neighbors	568
31.6 Exercises	569
31.7 Generative models	569

31.7.1 Naive Bayes	570
31.7.2 Controlling prevalence	571
31.7.3 Quadratic discriminant analysis	573
31.7.4 Linear discriminant analysis	575
31.7.5 Connection to distance	577
31.8 Case study: more than three classes	577
31.9 Exercises	581
31.10 Classification and regression trees (CART)	582
31.10.1 The curse of dimensionality	582
31.10.2 CART motivation	583
31.10.3 Regression trees	586
31.10.4 Classification (decision) trees	592
31.11 Random forests	594
31.12 Exercises	599
32 Machine learning in practice	601
32.1 Preprocessing	602
32.2 k-nearest neighbor and random forest	603
32.3 Variable importance	606
32.4 Visual assessments	607
32.5 Ensembles	607
32.6 Exercises	608
33 Large datasets	609
33.1 Matrix algebra	609
33.1.1 Notation	610
33.1.2 Converting a vector to a matrix	612
33.1.3 Row and column summaries	613
33.1.4 <code>apply</code>	614
33.1.5 Filtering columns based on summaries	614
33.1.6 Indexing with matrices	616
33.1.7 Binarizing the data	618
33.1.8 Vectorization for matrices	618
33.1.9 Matrix algebra operations	619
33.2 Exercises	619

0.0 Contents	19
33.3 Distance	619
33.3.1 Euclidean distance	620
33.3.2 Distance in higher dimensions	620
33.3.3 Euclidean distance example	621
33.3.4 Predictor space	623
33.3.5 Distance between predictors	623
33.4 Exercises	623
33.5 Dimension reduction	624
33.5.1 Preserving distance	624
33.5.2 Linear transformations (advanced)	627
33.5.3 Orthogonal transformations (advanced)	628
33.5.4 Principal component analysis	630
33.5.5 Iris example	632
33.5.6 MNIST example	635
33.6 Exercises	637
33.7 Recommendation systems	638
33.7.1 Movielens data	638
33.7.2 Recommendation systems as a machine learning challenge	640
33.7.3 Loss function	640
33.7.4 A first model	641
33.7.5 Modeling movie effects	642
33.7.6 User effects	643
33.8 Exercises	644
33.9 Regularization	645
33.9.1 Motivation	645
33.9.2 Penalized least squares	647
33.9.3 Choosing the penalty terms	650
33.10 Exercises	652
33.11 Matrix factorization	653
33.11.1 Factors analysis	656
33.11.2 Connection to SVD and PCA	658
33.12 Exercises	661

34 Clustering	667
34.1 Hierarchical clustering	668
34.2 k-means	670
34.3 Heatmaps	670
34.4 Filtering features	671
34.5 Exercises	672
 VI Productivity Tools	 673
 35 Introduction to productivity tools	 675
 36 Organizing with Unix	 677
36.1 Naming convention	677
36.2 The terminal	678
36.3 The filesystem	678
36.3.1 Directories and subdirectories	679
36.3.2 The home directory	679
36.3.3 Working directory	680
36.3.4 Paths	681
36.4 Unix commands	681
36.4.1 <code>ls</code> : Listing directory content	682
36.4.2 <code>mkdir</code> and <code>rmdir</code> : make and remove a directory	682
36.4.3 <code>cd</code> : navigating the filesystem by changing directories	683
36.5 Some examples	685
36.6 More Unix commands	686
36.6.1 <code>mv</code> : moving files	686
36.6.2 <code>cp</code> : copying files	687
36.6.3 <code>rm</code> : removing files	687
36.6.4 <code>less</code> : looking at a file	688
36.7 Preparing for a data science project	688
36.8 Advanced Unix	689
36.8.1 Arguments	689
36.8.2 Getting help	690
36.8.3 Pipes	691
36.8.4 Wild cards	691
36.8.5 Environment variables	692

<i>0.0 Contents</i>	21
36.8.6 Shells	692
36.8.7 Executables	693
36.8.8 Permissions and file types	693
36.8.9 Commands you should learn	694
36.8.10 File manipulation in R	694
37 Git and GitHub	695
37.1 Why use Git and GitHub?	695
37.2 GitHub accounts	695
37.3 GitHub repositories	698
37.4 Overview of Git	699
37.4.1 Clone	700
37.5 Initializing a Git directory	704
37.6 Using Git and GitHub in RStudio	706
38 Reproducible projects with RStudio and R markdown	711
38.1 RStudio projects	711
38.2 R markdown	714
38.2.1 The header	716
38.2.2 R code chunks	716
38.2.3 Global options	717
38.2.4 knitR	717
38.2.5 More on R markdown	718
38.3 Organizing a data science project	718
38.3.1 Create directories in Unix	718
38.3.2 Create an RStudio project	719
38.3.3 Edit some R scripts	720
38.3.4 Create some more directories using Unix	721
38.3.5 Add a README file	721
38.3.6 Initializing a Git directory	721



Preface

This book started out as the class notes used in the HarvardX Data Science Series¹.

The link for the online version of the book is <https://rafalab.github.io/dsbook/>

The R markdown code used to generate the book is available on GitHub². Note that, the graphical theme used for plots throughout the book can be recreated using the `ds_theme_set()` function from **dslabs** package.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)³.

We make announcements related to the book on Twitter. For updates follow @rafalab⁴

¹<https://www.edx.org/professional-certificate/harvardx-data-science>

²<https://github.com/rafalab/dsbook>

³<https://creativecommons.org/licenses/by-nc-sa/4.0>

⁴<https://twitter.com/rafalab>



Acknowledgments

This book is dedicated to all the people involved in building and maintaining R and the R packages we use in this book. A special thanks to the developers and maintainers of R base, the tidyverse, and the caret package.

A special thanks to my tidyverse guru David Robinson and Amy Gill for dozens of comments, edits, and suggestions. Also, many thanks to Stephanie Hicks who twice served as a co-instructor in my data science classes and Yihui Xie who patiently put up with my many questions about bookdown. Thanks also to Karl Broman, from whom I borrowed ideas for the Data Visualization and Productivity Tools parts, and to Hector Corrada-Bravo, for advice on how to best teach machine learning. Thanks to Peter Aldhous from whom I borrowed ideas for the principles of data visualization section and Jenny Bryan for writing *Happy Git and GitHub for the useR*, which influenced our Git chapters. Thanks to Alyssa Frazee for helping create the homework problem that became the Recommendation Systems chapter and to Amanda Cox for providing the New York Regents exams data. Also, many thanks to Jeff Leek, Roger Peng, and Brian Caffo, whose class inspired the way this book is divided and to Garrett Golemund and Hadley Wickham for making the bookdown code for their R for Data Science book open. Finally, thanks to Alex Nones for proofreading the manuscript during its various stages.

This book was conceived during the teaching of several applied statistics courses, starting over fifteen years ago. The teaching assistants working with me throughout the years made important indirect contributions to this book. The latest iteration of this course is a HarvardX series coordinated by Heather Sternshein and Zzofia Gajdos. We thank them for their contributions. We are also grateful to all the students whose questions and comments helped us improve the book. The courses were partially funded by NIH grant R25GM114818. We are very grateful to the National Institutes of Health for its support.

A special thanks goes to all those who edited the book via GitHub pull requests or made suggestions by creating an *issue*: `nickyfoto` (Huang Qiang), `desautm` (Marc-André Désautels), `michaschwab` (Michail Schwab), `alvarolarreategui` (Alvaro Larreategui), `jakevc` (Jake VanCampen), `omerta` (Guillermo Lengemann), `espinielli` (Enrico Spinielli), `asimumba` (Aaron Simumba), `braunschweig` (Maldewar), `gwierzchowski` (Grzegorz Wierzchowski), `technocrat` (Richard Careaga), `atzakas`, `defeit` (David Emerson Feit), `shiraamitchell` (Shira Mitchell), `Nathalie-S`, `andreashandel` (Andreas Handel), `berkowitze` (Elias Berkowitz), `Dean-Webb` (Dean Webber), `mohayusuf`, `jimrothstein`, `mPloenzke` (Matthew Ploenzke), and David D. Kane.



Introduction

The demand for skilled data science practitioners in industry, academia, and government is rapidly growing. This book introduces concepts and skills that can help you tackle real-world data analysis challenges. It covers concepts from probability, statistical inference, linear regression, and machine learning. It also helps you develop skills such as R programming, data wrangling with **dplyr**, data visualization with **ggplot2**, algorithm building with **caret**, file organization with UNIX/Linux shell, version control with Git and GitHub, and reproducible document preparation with **knitr** and R markdown. The book is divided into six parts: **R**, **Data Visualization**, **Data Wrangling**, **Statistics with R**, **Machine Learning**, and **Productivity Tools**. Each part has several chapters meant to be presented as one lecture and includes dozens of exercises distributed across chapters.

Case studies

Throughout the book, we use motivating case studies. In each case study, we try to realistically mimic a data scientist's experience. For each of the concepts covered, we start by asking specific questions and answer these through data analysis. We learn the concepts as a means to answer the questions. Examples of the case studies included in the book are:

Case Study	Concept
US murder rates by state	R Basics
Student heights	Statistical Summaries
Trends in world health and economics	Data Visualization
The impact of vaccines on infectious disease rates	Data Visualization
The financial crisis of 2007-2008	Probability
Election forecasting	Statistical Inference
Reported student heights	Data Wrangling
Money Ball: Building a baseball team	Linear Regression
MNIST: Image processing hand-written digits	Machine Learning
Movie recommendation systems	Machine Learning

Who will find this book useful?

This book is meant to be a textbook for a first course in Data Science. No previous knowledge of R is necessary, although some experience with programming may be helpful. The statistical concepts used to answer the case study questions are only briefly introduced, so a Probability and Statistics textbook is highly recommended for in-depth understanding of these concepts. If you read and understand all the chapters and complete all the exercises, you will be well-positioned to perform basic data analysis tasks and you will be prepared to learn the more advanced concepts and skills needed to become an expert.

What does this book cover?

We start by going over the **basics of R** and the **tidyverse**. You learn R throughout the book, but in the first part we go over the building blocks needed to keep learning.

The growing availability of informative datasets and software tools has led to increased reliance on **data visualizations** in many fields. In the second part we demonstrate how to use **ggplot2** to generate graphs and describe important data visualization principles.

In the third part we demonstrate the importance of statistics in data analysis by answering case study questions using **probability, inference, and regression** with R.

The fourth part uses several examples to familiarize the reader with **data wrangling**. Among the specific skills we learn are web scrapping, using regular expressions, and joining and reshaping data tables. We do this using **tidyverse** tools.

In the fifth part we present several challenges that lead us to introduce **machine learning**. We learn to use the **caret** package to build prediction algorithms including K-nearest neighbors and random forests.

In the final part, we provide a brief introduction to the **productivity tools** we use on a day-to-day basis in data science projects. These are RStudio, UNIX/Linux shell, Git and GitHub, and **knitr** and R Markdown.

What is not covered by this book?

This book focuses on the data analysis aspects of data science. We therefore do not cover aspects related to data management or engineering. Although R programming is an essential part of the book, we do not teach more advanced computer science topics such as data structures, optimization, and algorithm theory. Similarly, we do not cover topics such as web services, interactive graphics, parallel computing, and data streaming processing. The statistical concepts are presented mainly as tools to solve problems and in-depth theoretical descriptions are not included in this book.

1

Getting started with R and RStudio

1.1 Why R?

R is not a programming language like C or Java. It was not created by software engineers for software development. Instead, it was developed by statisticians as an interactive environment for data analysis. You can read the full history in the paper A Brief History of S¹. The interactivity is an indispensable feature in data science because, as you will soon learn, the ability to quickly explore data is a necessity for success in this field. However, like in other programming languages, you can save your work as scripts that can be easily executed at any moment. These scripts serve as a record of the analysis you performed, a key feature that facilitates reproducible work. If you are an expert programmer, you should not expect R to follow the conventions you are used to since you will be disappointed. If you are patient, you will come to appreciate the unequal power of R when it comes to data analysis and, specifically, data visualization.

Other attractive features of R are:

1. R is free and open source².
2. It runs on all major platforms: Windows, Mac Os, UNIX/Linux.
3. Scripts and data objects can be shared seamlessly across platforms.
4. There is a large, growing, and active community of R users and, as a result, there are numerous resources for learning and asking questions^{3 4 5}.
5. It is easy for others to contribute add-ons which enables developers to share software implementations of new data science methodologies. This gives R users early access to the latest methods and to tools which are developed for a wide variety of disciplines, including ecology, molecular biology, social sciences, and geography, just to name a few examples.

1.2 The R console

Interactive data analysis usually occurs on the *R console* that executes commands as you type them. There are several ways to gain access to an R console. One way is to simply start R on your computer. The console looks something like this:

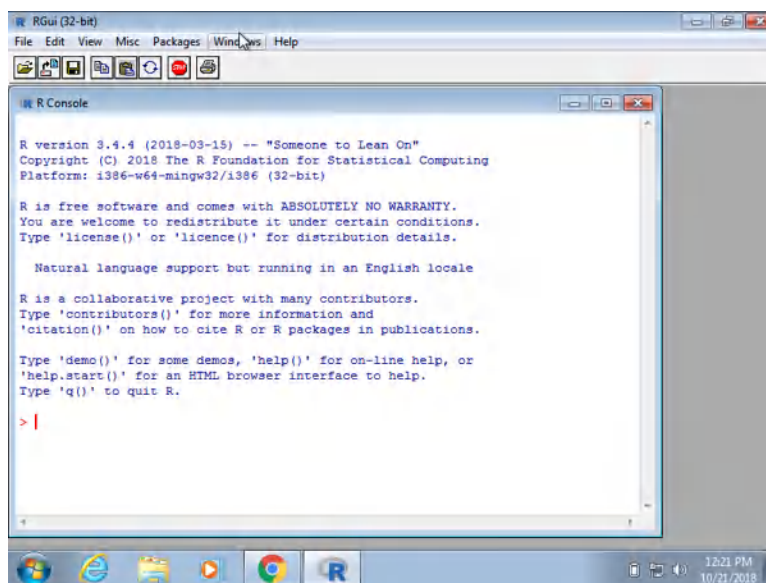
¹<https://pdfs.semanticscholar.org/9b48/46f192aa37ca122cfabb1ed1b59866d8bfda.pdf>

²<https://opensource.org/history>

³<https://stats.stackexchange.com/questions/138/free-resources-for-learning-r>

⁴<https://www.r-project.org/help.html>

⁵<https://stackoverflow.com/documentation/r/topics>



As a quick example, try using the console to calculate a 15% tip on a meal that cost \$19.71:

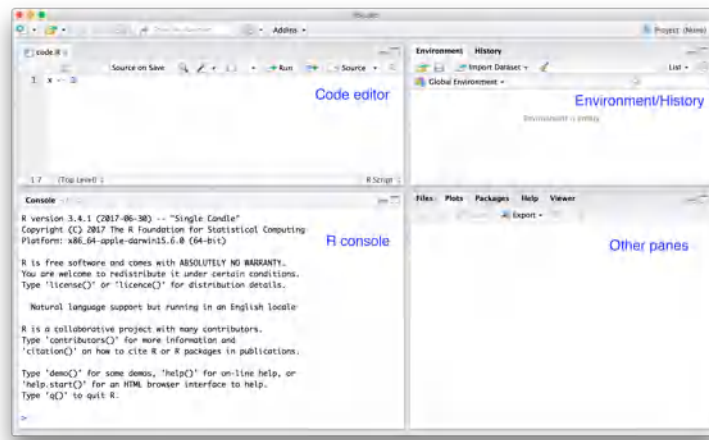
```
0.15 * 19.71  
#> [1] 2.96
```

Note that in this book, grey boxes are used to show R code typed into the R console. The symbol `#>` is used to denote what the R console outputs.

1.3 Scripts

One of the great advantages of R over point-and-click analysis software is that you can save your work as scripts. You can edit and save these scripts using a text editor. The material in this book was developed using the interactive *integrated development environment* (IDE) RStudio⁶. RStudio includes an editor with many R specific features, a console to execute your code, and other useful panes, including one to show figures.

⁶<https://www.rstudio.com/>



Most web-based R consoles also provide a pane to edit scripts, but not all permit you to save the scripts for later use.

All the R scripts used to generate this book can be found on GitHub⁷.

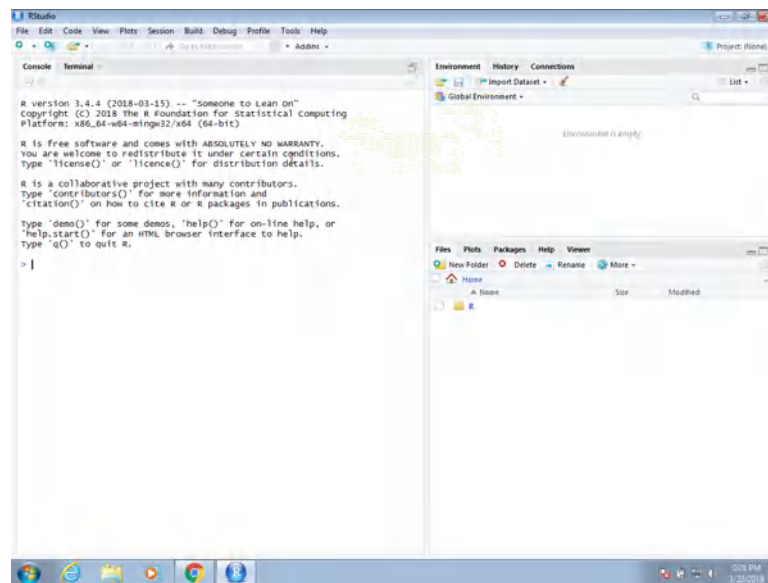
1.4 RStudio

RStudio will be our launching pad for data science projects. It not only provides an editor for us to create and edit our scripts but also provides many other useful tools. In this section, we go over some of the basics.

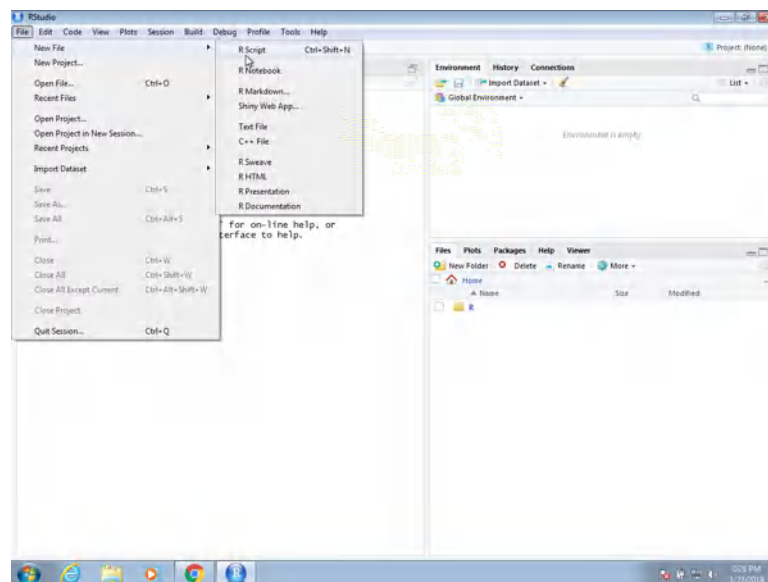
1.4.1 The panes

When you start RStudio for the first time, you will see three panes. The left pane shows the R console. On the right, the top pane includes tabs such as *Environment* and *History*, while the bottom pane shows five tabs: *File*, *Plots*, *Packages*, *Help*, and *Viewer* (these tabs may change in new versions). You can click on each tab to move across the different features.

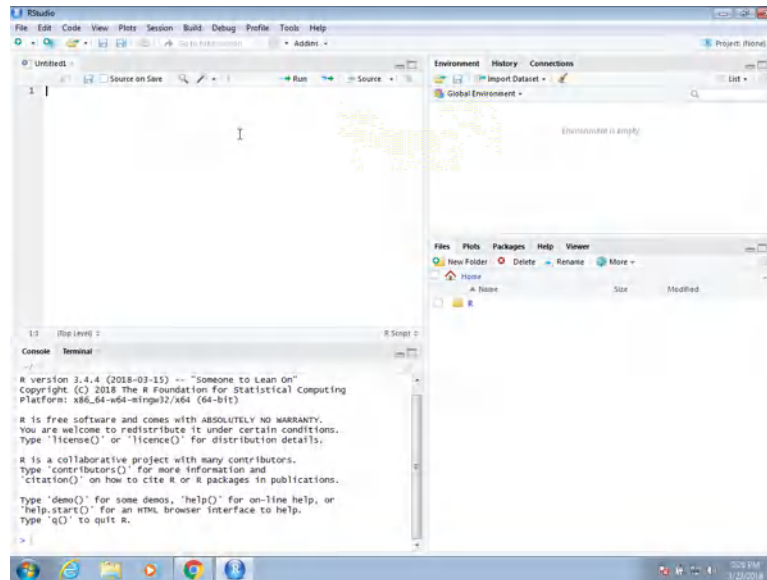
⁷<https://github.com/rafalab/dsbook>



To start a new script, you can click on File, the New File, then R Script.



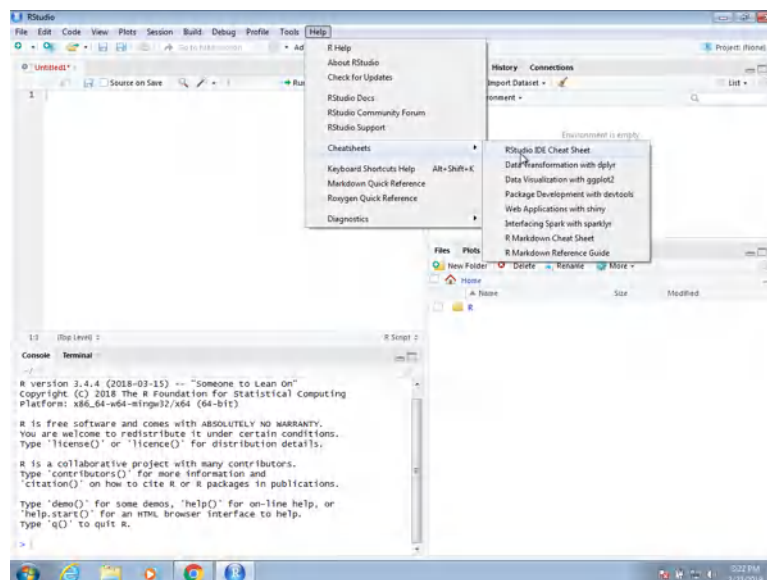
This starts a new pane on the left and it is here where you can start writing your script.



1.4.2 Key bindings

Many tasks we perform with the mouse can be achieved with a combination of key strokes instead. These keyboard versions for performing tasks are referred to as *key bindings*. For example, we just showed how to use the mouse to start a new script, but you can also use a key binding: Ctrl+Shift+N on Windows and command+shift+N on the Mac.

Although in this tutorial we often show how to use the mouse, **we highly recommend that you memorize key bindings for the operations you use most**. RStudio provides a useful cheat sheet with the most widely used commands. You can get it from RStudio directly:



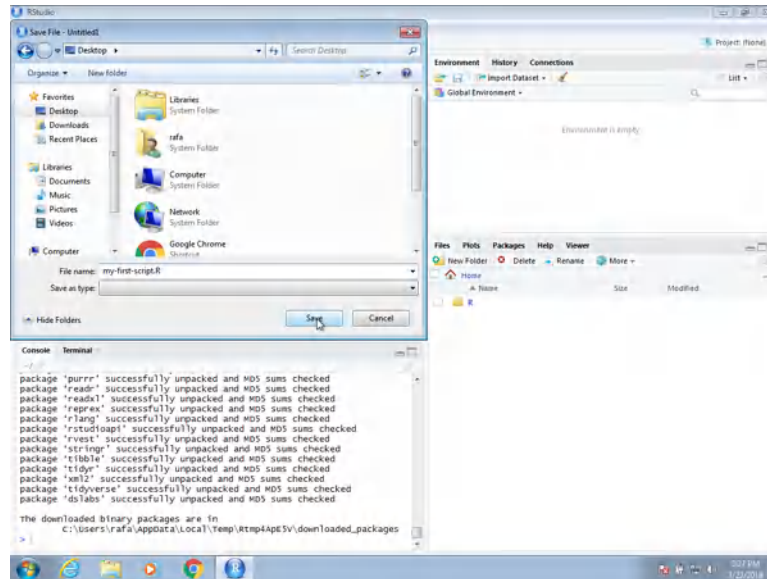
You might want to keep this handy so you can look up key-bindings when you find yourself performing repetitive point-and-clicking.

1.4.3 Running commands while editing scripts

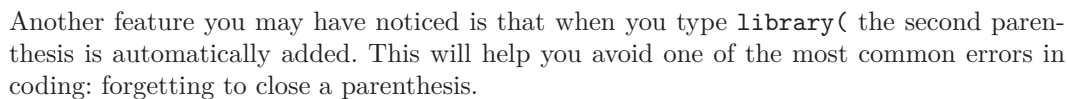
There are many editors specifically made for coding. These are useful because color and indentation are automatically added to make code more readable. RStudio is one of these editors, and it was specifically developed for R. One of the main advantages provided by RStudio over other editors is that we can test our code easily as we edit our scripts. Below we show an example.

Let's start by opening a new script as we did before. A next step is to give the script a name. We can do this through the editor by saving the current new unnamed script. To do this, click on the save icon or use the key binding Ctrl+S on Windows and command+S on the Mac.

When you ask for the document to be saved for the first time, RStudio will prompt you for a name. A good convention is to use a descriptive name, with lower case letters, no spaces, only hyphens to separate words, and then followed by the suffix *.R*. We will call this script *my-first-script.R*.



Now we are ready to start editing our first script. The first lines of code in an R script are dedicated to loading the libraries we will use. Another useful RStudio feature is that once we type `library()` it starts auto-completing with libraries that we have installed. Note what happens when we type `library(tidyverse)`:



Once you run the code, you will see it appear in the R console and, in this case, the generated plot appears in the plots console. Note that the plot console has a useful interface that permits you to click back and forward across different plots, zoom in to the plot, or save the plots as files.



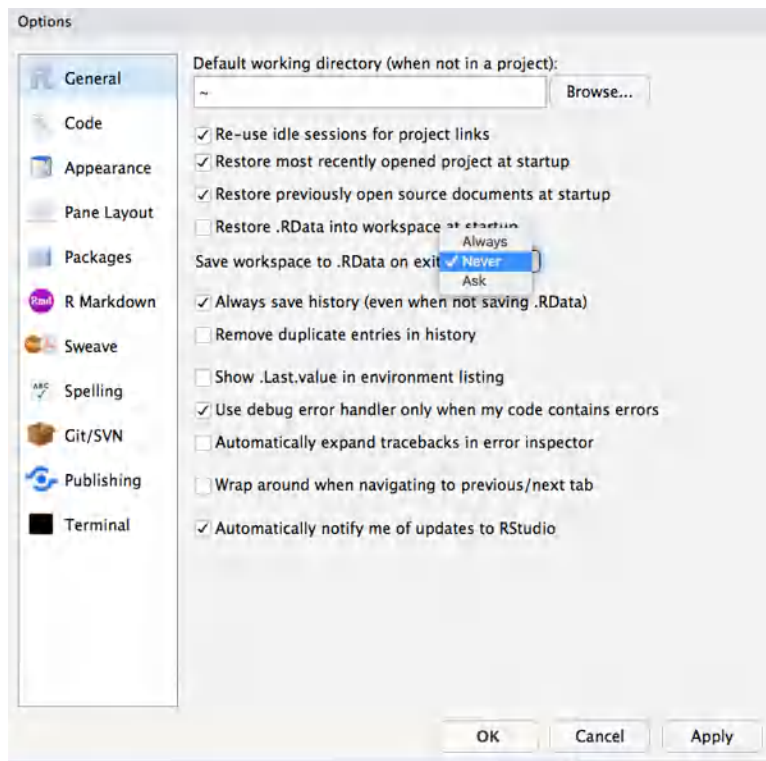
To run one line at a time instead of the entire script, you can use Control-Enter on Windows and command-return on the Mac.

1.4.4 Changing global options

You can change the look and functionality of RStudio quite a bit.

To change the global options you click on *Tools* then *Global Options...*.

As an example we show how to make a change that we **highly recommend**. This is to change the *Save workspace to .RData on exit* to *Never* and uncheck the *Restore .RData into workspace at start*. By default, when you exit R saves all the objects you have created into a file called .RData. This is done so that when you restart the session in the same folder, it will load these objects. We find that this causes confusion especially when we share code with colleagues and assume they have this .RData file. To change these options, make your *General* settings look like this:



1.5 Installing R packages

The functionality provided by a fresh install of R is only a small fraction of what is possible. In fact, we refer to what you get after your first install as *base R*. The extra functionality

comes from add-ons available from developers. There are currently hundreds of these available from CRAN and many others shared via other repositories such as GitHub. However, because not everybody needs all available functionality, R instead makes different components available via *packages*. R makes it very easy to install packages from within R. For example, to install the **dslabs** package, which we use to share datasets and code related to this book, you would type:

```
install.packages("dslabs")
```

In RStudio, you can navigate to the *Tools* tab and select install packages. We can then load the package into our R sessions using the `library` function:

```
library(dslabs)
```

As you go through this book, you will see that we load packages without installing them. This is because once you install a package, it remains installed and only needs to be loaded with `library`. The package remains loaded until we quit the R session. If you try to load a package and get an error, it probably means you need to install it first.

We can install more than one package at once by feeding a character vector to this function:

```
install.packages(c("tidyverse", "dslabs"))
```

Note that installing **tidyverse** actually installs several packages. This commonly occurs when a package has *dependencies*, or uses functions from other packages. When you load a package using `library`, you also load its dependencies.

Once packages are installed, you can load them into R and you do not need to install them again, unless you install a fresh version of R. Remember packages are installed in R not RStudio.

It is helpful to keep a list of all the packages you need for your work in a script because if you need to perform a fresh install of R, you can re-install all your packages by simply running a script.

You can see all the packages you have installed using the following function:

```
installed.packages()
```



Part I

R



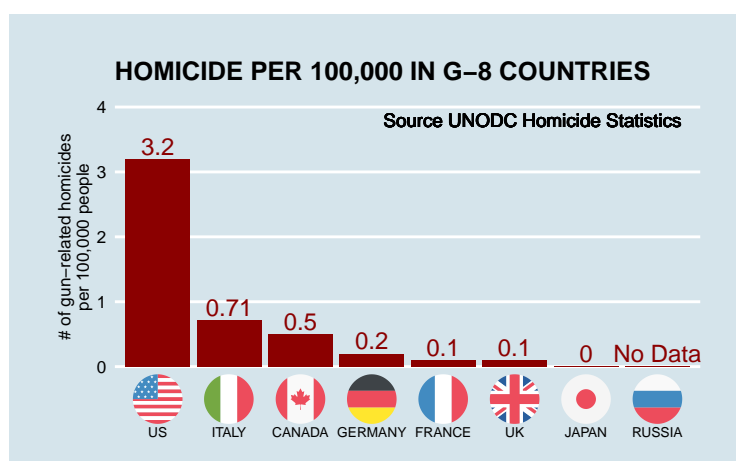
2

R basics

In this book, we will be using the R software environment for all our analysis. You will learn R and data analysis techniques simultaneously. To follow along you will therefore need access to R. We also recommend the use of an *integrated development environment* (IDE), such as RStudio, to save your work. Note that it is common for a course or workshop to offer access to an R environment and an IDE through your web browser, as done by RStudio cloud¹. If you have access to such a resource, you don't need to install R and RStudio. However, if you intend on becoming an advanced data analyst, we highly recommend installing these tools on your computer². Both R and RStudio are free and available online.

2.1 Case study: US Gun Murders

Imagine you live in Europe and are offered a job in a US company with many locations across all states. It is a great job, but news with headlines such as **US Gun Homicide Rate Higher Than Other Developed Countries**³ have you worried. Charts like this may concern you even more:

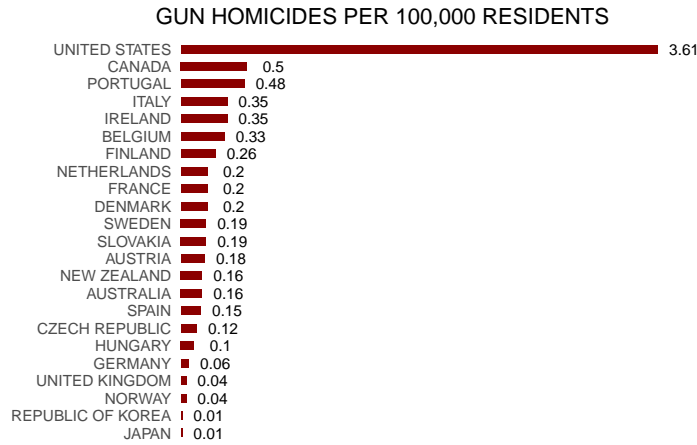


Or even worse, this version from everytown.org:

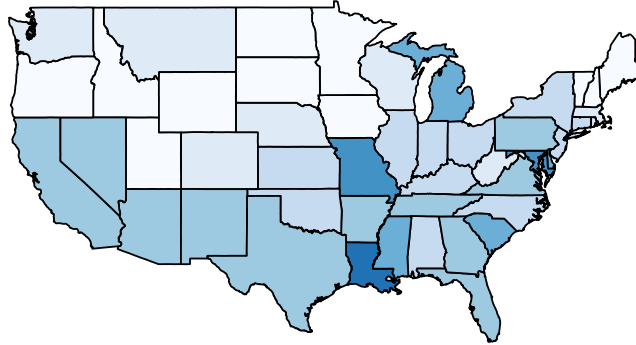
¹<https://rstudio.cloud>

²<https://rafalab.github.io/dsbook/installing-r-rstudio.html>

³<http://abcnews.go.com/blogs/headlines/2012/12/us-gun-ownership-homicide-rate-higher-than-other-developed-countries/>



But then you remember that the US is a large and diverse country with 50 very different states as well as the District of Columbia (DC).



California, for example, has a larger population than Canada, and 20 US states have populations larger than that of Norway. In some respects, the variability across states in the US is akin to the variability across countries in Europe. Furthermore, although not included in the charts above, the murder rates in Lithuania, Ukraine, and Russia are higher than 4 per 100,000. So perhaps the news reports that worried you are too superficial. You have options of where to live and want to determine the safety of each particular state. We will gain some insights by examining data related to gun homicides in the US during 2010 using R.

Before we get started with our example, we need to cover logistics as well as some of the very basic building blocks that are required to gain more advanced R skills. Be aware that the usefulness of some of these building blocks may not be immediately obvious, but later in the book you will appreciate having mastered these skills.

2.2 The very basics

Before we get started with the motivating dataset, we need to cover the very basics of R.

2.2.1 Objects

Suppose a high school student asks us for help solving several quadratic equations of the form $ax^2 + bx + c = 0$. The quadratic formula gives us the solutions:

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

which of course change depending on the values of a , b , and c . One advantage of programming languages is that we can define variables and write expressions with these variables, similar to how we do so in math, but obtain a numeric solution. We will write out general code for the quadratic equation below, but if we are asked to solve $x^2 + x - 1 = 0$, then we define:

```
a <- 1
b <- 1
c <- -1
```

which stores the values for later use. We use `<-` to assign values to the variables.

We can also assign values using `=` instead of `<-`, but we recommend against using `=` to avoid confusion.

Copy and paste the code above into your console to define the three variables. Note that R does not print anything when we make this assignment. This means the objects were defined successfully. Had you made a mistake, you would have received an error message.

To see the value stored in a variable, we simply ask R to evaluate `a` and it shows the stored value:

```
a
#> [1] 1
```

A more explicit way to ask R to show us the value stored in `a` is using `print` like this:

```
print(a)
#> [1] 1
```

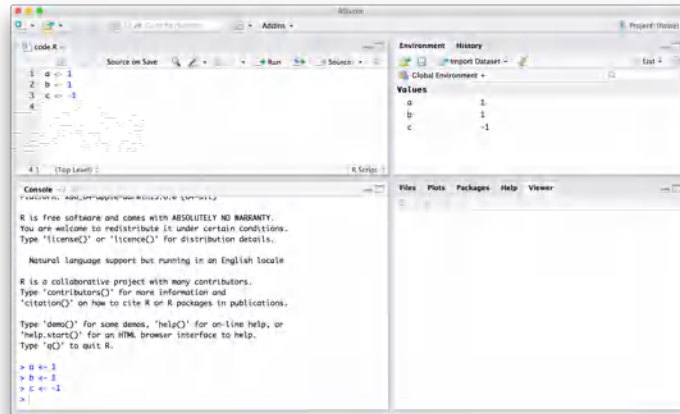
We use the term *object* to describe stuff that is stored in R. Variables are examples, but objects can also be more complicated entities such as functions, which are described later.

2.2.2 The workspace

As we define objects in the console, we are actually changing the *workspace*. You can see all the variables saved in your workspace by typing:

```
ls()
#> [1] "a"      "b"      "c"      "dat"     "img_path" "murders"
```

In RStudio, the *Environment* tab shows the values:



We should see `a`, `b`, and `c`. If you try to recover the value of a variable that is not in your workspace, you receive an error. For example, if you type `x` you will receive the following message: `Error: object 'x' not found`.

Now since these values are saved in variables, to obtain a solution to our equation, we use the quadratic formula:

```
(-b + sqrt(b^2 - 4*a*c) ) / ( 2*a )
#> [1] 0.618
(-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
#> [1] -1.62
```

2.2.3 Functions

Once you define variables, the data analysis process can usually be described as a series of *functions* applied to the data. R includes several predefined functions and most of the analysis pipelines we construct make extensive use of these.

We already used the `install.packages`, `library`, and `ls` functions. We also used the function `sqrt` to solve the quadratic equation above. There are many more prebuilt functions and even more can be added through packages. These functions do not appear in the workspace because you did not define them, but they are available for immediate use.

In general, we need to use parentheses to evaluate a function. If you type `ls`, the function is not evaluated and instead R shows you the code that defines the function. If you type `ls()` the function is evaluated and, as seen above, we see objects in the workspace.

Unlike `ls`, most functions require one or more *arguments*. Below is an example of how we assign an object to the argument of the function `log`. Remember that we earlier defined `a` to be 1:

```
log(8)
#> [1] 2.08
log(a)
#> [1] 0
```

You can find out what the function expects and what it does by reviewing the very useful manuals included in R. You can get help by using the `help` function like this:

```
help("log")
```

For most functions, we can also use this shorthand:

```
?log
```

The help page will show you what arguments the function is expecting. For example, `log` needs `x` and `base` to run. However, some arguments are required and others are optional. You can determine which arguments are optional by noting in the help document that a default value is assigned with `=`. Defining these is optional. For example, the base of the function `log` defaults to `base = exp(1)` making `log` the natural log by default.

If you want a quick look at the arguments without opening the help system, you can type:

```
args(log)
#> function (x, base = exp(1))
#> NULL
```

You can change the default values by simply assigning another object:

```
log(8, base = 2)
#> [1] 3
```

Note that we have not been specifying the argument `x` as such:

```
log(x = 8, base = 2)
#> [1] 3
```

The above code works, but we can save ourselves some typing: if no argument name is used, R assumes you are entering arguments in the order shown in the help file or by `args`. So by not using the names, it assumes the arguments are `x` followed by `base`:

```
log(8,2)
#> [1] 3
```

If using the arguments' names, then we can include them in whatever order we want:

```
log(base = 2, x = 8)
#> [1] 3
```

To specify arguments, we must use `=`, and cannot use `<-`.

There are some exceptions to the rule that functions need the parentheses to be evaluated. Among these, the most commonly used are the arithmetic and relational operators. For example:

```
2 ^ 3
#> [1] 8
```

You can see the arithmetic operators by typing:

```
help("+")
```

or

```
?"+"
```

and the relational operators by typing:

```
help(">")
```

or

```
?">"
```

2.2.4 Other prebuilt objects

There are several datasets that are included for users to practice and test out functions. You can see all the available datasets by typing:

```
data()
```

This shows you the object name for these datasets. These datasets are objects that can be used by simply typing the name. For example, if you type:

```
co2
```

R will show you Mauna Loa atmospheric CO2 concentration data.

Other prebuilt objects are mathematical quantities, such as the constant π and ∞ :

```
pi
#> [1] 3.14
Inf+1
#> [1] Inf
```

2.2.5 Variable names

We have used the letters `a`, `b`, and `c` as variable names, but variable names can be almost anything. Some basic rules in R are that variable names have to start with a letter, can't contain spaces, and should not be variables that are predefined in R. For example, don't name one of your variables `install.packages` by typing something like `install.packages <- 2`.

A nice convention to follow is to use meaningful words that describe what is stored, use only lower case, and use underscores as a substitute for spaces. For the quadratic equations, we could use something like this:

```
solution_1 <- (-b + sqrt(b^2 - 4*a*c)) / (2*a)
solution_2 <- (-b - sqrt(b^2 - 4*a*c)) / (2*a)
```

For more advice, we highly recommend studying Hadley Wickham's style guide⁴.

2.2.6 Saving your workspace

Values remain in the workspace until you end your session or erase them with the function `rm`. But workspaces also can be saved for later use. In fact, when you quit R, the program asks you if you want to save your workspace. If you do save it, the next time you start R, the program will restore the workspace.

We actually recommend against saving the workspace this way because, as you start working on different projects, it will become harder to keep track of what is saved. Instead, we recommend you assign the workspace a specific name. You can do this by using the function `save` or `save.image`. To load, use the function `load`. When saving a workspace, we recommend the suffix `rda` or `RData`. In RStudio, you can also do this by navigating to the *Session* tab and choosing *Save Workspace as*. You can later load it using the *Load Workspace* options in the same tab. You can read the help pages on `save`, `save.image`, and `load` to learn more.

2.2.7 Motivating scripts

To solve another equation such as $3x^2 + 2x - 1$, we can copy and paste the code above and then redefine the variables and recompute the solution:

```
a <- 3
b <- 2
c <- -1
(-b + sqrt(b^2 - 4*a*c)) / (2*a)
(-b - sqrt(b^2 - 4*a*c)) / (2*a)
```

By creating and saving a script with the code above, we would not need to retype everything each time and, instead, simply change the variable names. Try writing the script above into an editor and notice how easy it is to change the variables and receive an answer.

⁴<http://adv-r.had.co.nz/Style.html>

2.2.8 Commenting your code

If a line of R code starts with the symbol #, it is not evaluated. We can use this to write reminders of why we wrote particular code. For example, in the script above we could add:

```
## Code to compute solution to quadratic equation of the form ax^2 + bx + c
## define the variables
a <- 3
b <- 2
c <- -1

## now compute the solution
(-b + sqrt(b^2 - 4*a*c)) / (2*a)
(-b - sqrt(b^2 - 4*a*c)) / (2*a)
```

2.3 Exercises

1. What is the sum of the first 100 positive integers? The formula for the sum of integers 1 through n is $n(n+1)/2$. Define $n = 100$ and then use R to compute the sum of 1 through 100 using the formula. What is the sum?
2. Now use the same formula to compute the sum of the integers from 1 through 1,000.
3. Look at the result of typing the following code into R:

```
n <- 1000
x <- seq(1, n)
sum(x)
```

Based on the result, what do you think the functions `seq` and `sum` do? You can use `help`.

- a. `sum` creates a list of numbers and `seq` adds them up.
 - b. `seq` creates a list of numbers and `sum` adds them up.
 - c. `seq` creates a random list and `sum` computes the sum of 1 through 1,000.
 - d. `sum` always returns the same number.
4. In math and programming, we say that we evaluate a function when we replace the argument with a given number. So if we type `sqrt(4)`, we evaluate the `sqrt` function. In R, you can evaluate a function inside another function. The evaluations happen from the inside out. Use one line of code to compute the log, in base 10, of the square root of 100.
 5. Which of the following will always return the numeric value stored in `x`? You can try out examples and use the help system if you want.
 - a. `log(10^x)`
 - b. `log10(x^10)`
 - c. `log(exp(x))`
 - d. `exp(log(x, base = 2))`

2.4 Data types

Variables in R can be of different types. For example, we need to distinguish numbers from character strings and tables from simple lists of numbers. The function `class` helps us determine what type of object we have:

```
a <- 2
class(a)
#> [1] "numeric"
```

To work efficiently in R, it is important to learn the different types of variables and what we can do with these.

2.4.1 Data frames

Up to now, the variables we have defined are just one number. This is not very useful for storing data. The most common way of storing a dataset in R is in a *data frame*. Conceptually, we can think of a data frame as a table with rows representing observations and the different variables reported for each observation defining the columns. Data frames are particularly useful for datasets because we can combine different data types into one object.

A large proportion of data analysis challenges start with data stored in a data frame. For example, we stored the data for our motivating example in a data frame. You can access this dataset by loading the **dslabs** library and loading the **murders** dataset using the `data` function:

```
library(dslabs)
data(murders)
```

To see that this is in fact a data frame, we type:

```
class(murders)
#> [1] "data.frame"
```

2.4.2 Examining an object

The function `str` is useful for finding out more about the structure of an object:

```
str(murders)
#> 'data.frame':   51 obs. of  5 variables:
#> $ state : chr "Alabama" "Alaska" "Arizona" "Arkansas" ...
#> $ abb  : chr "AL" "AK" "AZ" "AR" ...
#> $ region : Factor w/ 4 levels "Northeast","South",...: 2 4 4 2 4 4 1 2 2
#>      2 ...
#> $ population: num 4779736 710231 6392017 2915918 37253956 ...
#> $ total : num 135 19 232 93 1257 ...
```


This tells us much more about the object. We see that the table has 51 rows (50 states plus DC) and five variables. We can show the first six lines using the function `head`:

```
head(murders)
#>      state abb region population total
#> 1  Alabama AL  South    4779736    135
#> 2  Alaska  AK   West     710231     19
#> 3  Arizona AZ   West    6392017    232
#> 4  Arkansas AR  South    2915918     93
#> 5 California CA   West   37253956   1257
#> 6  Colorado CO   West    5029196     65
```

In this dataset, each state is considered an observation and five variables are reported for each state.

Before we go any further in answering our original question about different states, let's learn more about the components of this object.

2.4.3 The accessor: `$`

For our analysis, we will need to access the different variables represented by columns included in this data frame. To do this, we use the accessor operator `$` in the following way:

```
murders$population
#> [1] 4779736 710231 6392017 2915918 37253956 5029196 3574097
#> [8] 897934 601723 19687653 9920000 1360301 1567582 12830632
#> [15] 6483802 3046355 2853118 4339367 4533372 1328361 5773552
#> [22] 6547629 9883640 5303925 2967297 5988927 989415 1826341
#> [29] 2700551 1316470 8791894 2059179 19378102 9535483 672591
#> [36] 11536504 3751351 3831074 12702379 1052567 4625364 814180
#> [43] 6346105 25145561 2763885 625741 8001024 6724540 1852994
#> [50] 5686986 563626
```

But how did we know to use `population`? Previously, by applying the function `str` to the object `murders`, we revealed the names for each of the five variables stored in this table. We can quickly access the variable names using:

```
names(murders)
#> [1] "state"      "abb"        "region"     "population" "total"
```

It is important to know that the order of the entries in `murders$population` preserves the order of the rows in our data table. This will later permit us to manipulate one variable based on the results of another. For example, we will be able to order the state names by the number of murders.

Tip: R comes with a very nice auto-complete functionality that saves us the trouble of typing out all the names. Try typing `murders$p` then hitting the `tab` key on your keyboard. This functionality and many other useful auto-complete features are available when working in RStudio.

2.4.4 Vectors: numerics, characters, and logical

The object `murders$population` is not one number but several. We call these types of objects *vectors*. A single number is technically a vector of length 1, but in general we use the term vectors to refer to objects with several entries. The function `length` tells you how many entries are in the vector:

```
pop <- murders$population
length(pop)
#> [1] 51
```

This particular vector is *numeric* since population sizes are numbers:

```
class(pop)
#> [1] "numeric"
```

In a numeric vector, every entry must be a number.

To store character strings, vectors can also be of class *character*. For example, the state names are characters:

```
class(murders$state)
#> [1] "character"
```

As with numeric vectors, all entries in a character vector need to be a character.

Another important type of vectors are *logical vectors*. These must be either `TRUE` or `FALSE`.

```
z <- 3 == 2
z
#> [1] FALSE
class(z)
#> [1] "logical"
```

Here the `==` is a relational operator asking if 3 is equal to 2. In R, if you just use one `=`, you actually assign a variable, but if you use two `==` you test for equality.

You can see the other *relational operators* by typing:

```
?Comparison
```

In future sections, you will see how useful relational operators can be.

We discuss more important features of vectors after the next set of exercises.

Advanced: Mathematically, the values in `pop` are integers and there is an integer class in R. However, by default, numbers are assigned class *numeric* even when they are round integers. For example, `class(1)` returns *numeric*. You can turn them into class *integer* with the `as.integer()` function or by adding an `L` like this: `1L`. Note the class by typing: `class(1L)`

2.4.5 Factors

In the `murders` dataset, we might expect the region to also be a character vector. However, it is not:

```
class(murders$region)
#> [1] "factor"
```

It is a *factor*. Factors are useful for storing categorical data. We can see that there are only 4 regions by using the `levels` function:

```
levels(murders$region)
#> [1] "Northeast" "South" "North Central" "West"
```

In the background, R stores these *levels* as integers and keeps a map to keep track of the labels. This is more memory efficient than storing all the characters.

Note that the levels have an order that is different from the order of appearance in the factor object. The default is for the levels to follow alphabetical order. However, often we want the levels to follow a different order. We will see several examples of this in the Data Visualization part of the book. The function `reorder` lets us change the order of the levels of a factor variable based on a summary computed on a numeric vector. We will demonstrate this with a simple example.

Suppose we want the levels of the region by the total number of murders rather than alphabetical order. If there are values associated with each level, we can use the `reorder` and specify a data summary to determine the order. The following code takes the sum of the total murders in each region, and reorders the factor following these sums.

```
region <- murders$region
value <- murders$total
region <- reorder(region, value, FUN = sum)
levels(region)
#> [1] "Northeast" "North Central" "West" "South"
```

The new order is in agreement with the fact that the Northeast has the least murders and the South has the most.

Warning: Factors can be a source of confusion since sometimes they behave like characters and sometimes they do not. As a result, confusing factors and characters are a common source of bugs.

2.4.6 Lists

Data frames are a special case of *lists*. We will cover lists in more detail later, but know that they are useful because you can store any combination of different types. Below is an example of a list we created for you:

```
record
#> $name
#> [1] "John Doe"
#>
#> $student_id
#> [1] 1234
#>
#> $grades
#> [1] 95 82 91 97 93
#>
#> $final_grade
#> [1] "A"
class(record)
#> [1] "list"
```

As with data frames, you can extract the components of a list with the accessor `$`. In fact, data frames are a type of list.

```
record$student_id
#> [1] 1234
```

We can also use double square brackets (`[[`) like this:

```
record[["student_id"]]
#> [1] 1234
```

You should get used to the fact that in R, there are often several ways to do the same thing, such as accessing entries.

You might also encounter lists without variable names.

```
record2
#> [[1]]
#> [1] "John Doe"
#>
#> [[2]]
#> [1] 1234
```

If a list does not have names, you cannot extract the elements with `$`, but you can still use the brackets method and instead of providing the variable name, you provide the list index, like this:

```
record2[[1]]
#> [1] "John Doe"
```

We won't be using lists until later, but you might encounter one in your own exploration of R. For this reason, we show you some basics here.

2.4.7 Matrices

Matrices are another type of object that are common in R. Matrices are similar to data frames in that they are two-dimensional: they have rows and columns. However, like numeric, character and logical vectors, entries in matrices have to be all the same type. For this reason data frames are much more useful for storing data, since we can have characters, factors, and numbers in them.

Yet matrices have a major advantage over data frames: we can perform matrix algebra operations, a powerful type of mathematical technique. We do not describe these operations in this book, but much of what happens in the background when you perform a data analysis involves matrices. We cover matrices in more detail in Chapter 33.1 but describe them briefly here since some of the functions we will learn return matrices.

We can define a matrix using the `matrix` function. We need to specify the number of rows and columns.

```
mat <- matrix(1:12, 4, 3)
mat
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12
```

You can access specific entries in a matrix using square brackets (`[]`). If you want the second row, third column, you use:

```
mat[2, 3]
#> [1] 10
```

If you want the entire second row, you leave the column spot empty:

```
mat[2, ]
#> [1]  2  6 10
```

Notice that this returns a vector, not a matrix.

Similarly, if you want the entire third column, you leave the row spot empty:

```
mat[, 3]
#> [1]  9 10 11 12
```

This is also a vector, not a matrix.

You can access more than one column or more than one row if you like. This will give you a new matrix.

```
mat[, 2:3]
#>      [,1] [,2]
#> [1,]    5    9
```

```
#> [2,]    6    10
#> [3,]    7    11
#> [4,]    8    12
```

You can subset both rows and columns:

```
mat[1:2, 2:3]
#>      [,1] [,2]
#> [1,]    5    9
#> [2,]    6   10
```

We can convert matrices into data frames using the function `as.data.frame`:

```
as.data.frame(mat)
#>   V1 V2 V3
#> 1  1  5  9
#> 2  2  6 10
#> 3  3  7 11
#> 4  4  8 12
```

You can also use single square brackets (`[]`) to access rows and columns of a data frame:

```
data("murders")
murders[25, 1]
#> [1] "Mississippi"
murders[2:3, ]
#>   state abb region population total
#> 2 Alaska AK  West    710231     19
#> 3 Arizona AZ  West   6392017    232
```

2.5 Exercises

1. Load the US murders dataset.

```
library(dslabs)
data(murders)
```

Use the function `str` to examine the structure of the `murders` object. Which of the following best describes the variables represented in this data frame?

- The 51 states.
- The murder rates for all 50 states and DC.
- The state name, the abbreviation of the state name, the state's region, and the state's population and total number of murders for 2010.
- `str` shows no relevant information.

2. What are the column names used by the data frame for these five variables?
3. Use the accessor `$` to extract the state abbreviations and assign them to the object `a`. What is the class of this object?
4. Now use the square brackets to extract the state abbreviations and assign them to the object `b`. Use the `identical` function to determine if `a` and `b` are the same.
5. We saw that the `region` column stores a factor. You can corroborate this by typing:

```
class(murders$region)
```

With one line of code, use the function `levels` and `length` to determine the number of regions defined by this dataset.

6. The function `table` takes a vector and returns the frequency of each element. You can quickly see how many states are in each region by applying this function. Use this function in one line of code to create a table of states per region.

2.6 Vectors

In R, the most basic objects available to store data are *vectors*. As we have seen, complex datasets can usually be broken down into components that are vectors. For example, in a data frame, each column is a vector. Here we learn more about this important class.

2.6.1 Creating vectors

We can create vectors using the function `c`, which stands for *concatenate*. We use `c` to concatenate entries in the following way:

```
codes <- c(380, 124, 818)
codes
#> [1] 380 124 818
```

We can also create character vectors. We use the quotes to denote that the entries are characters rather than variable names.

```
country <- c("italy", "canada", "egypt")
```

In R you can also use single quotes:

```
country <- c('italy', 'canada', 'egypt')
```

But be careful not to confuse the single quote `'` with the *back quote* ```.

By now you should know that if you type:

```
country <- c(italy, canada, egypt)
```

you receive an error because the variables `italy`, `canada`, and `egypt` are not defined. If we do not use the quotes, R looks for variables with those names and returns an error.

2.6.2 Names

Sometimes it is useful to name the entries of a vector. For example, when defining a vector of country codes, we can use the names to connect the two:

```
codes <- c(italy = 380, canada = 124, egypt = 818)
codes
#>   italy canada  egypt
#>    380    124    818
```

The object `codes` continues to be a numeric vector:

```
class(codes)
#> [1] "numeric"
```

but with names:

```
names(codes)
#> [1] "italy" "canada" "egypt"
```

If the use of strings without quotes looks confusing, know that you can use the quotes as well:

```
codes <- c("italy" = 380, "canada" = 124, "egypt" = 818)
codes
#>   italy canada  egypt
#>    380    124    818
```

There is no difference between this function call and the previous one. This is one of the many ways in which R is quirky compared to other languages.

We can also assign names using the `names` functions:

```
codes <- c(380, 124, 818)
country <- c("italy", "canada", "egypt")
names(codes) <- country
codes
#>   italy canada  egypt
#>    380    124    818
```


2.6.3 Sequences

Another useful function for creating vectors generates sequences:

```
seq(1, 10)
#> [1] 1 2 3 4 5 6 7 8 9 10
```

The first argument defines the start, and the second defines the end which is included. The default is to go up in increments of 1, but a third argument lets us tell it how much to jump by:

```
seq(1, 10, 2)
#> [1] 1 3 5 7 9
```

If we want consecutive integers, we can use the following shorthand:

```
1:10
#> [1] 1 2 3 4 5 6 7 8 9 10
```

When we use these functions, R produces integers, not numerics, because they are typically used to index something:

```
class(1:10)
#> [1] "integer"
```

However, if we create a sequence including non-integers, the class changes:

```
class(seq(1, 10, 0.5))
#> [1] "numeric"
```

2.6.4 Subsetting

We use square brackets to access specific elements of a vector. For the vector `codes` we defined above, we can access the second element using:

```
codes[2]
#> canada
#> 124
```

You can get more than one entry by using a multi-entry vector as an index:

```
codes[c(1,3)]
#> italy egypt
#> 380 818
```

The sequences defined above are particularly useful if we want to access, say, the first two elements:

```
codes[1:2]
#>  italy canada
#>   380   124
```

If the elements have names, we can also access the entries using these names. Below are two examples.

```
codes["canada"]
#> canada
#>   124
codes[c("egypt", "italy")]
#> egypt italy
#>  818  380
```

2.7 Coercion

In general, *coercion* is an attempt by R to be flexible with data types. When an entry does not match the expected, some of the prebuilt R functions try to guess what was meant before throwing an error. This can also lead to confusion. Failing to understand *coercion* can drive programmers crazy when attempting to code in R since it behaves quite differently from most other languages in this regard. Let's learn about it with some examples.

We said that vectors must be all of the same type. So if we try to combine, say, numbers and characters, you might expect an error:

```
x <- c(1, "canada", 3)
```

But we don't get one, not even a warning! What happened? Look at `x` and its class:

```
x
#> [1] "1"      "canada" "3"
class(x)
#> [1] "character"
```

R *coerced* the data into characters. It guessed that because you put a character string in the vector, you meant the 1 and 3 to actually be character strings "1" and "3". The fact that not even a warning is issued is an example of how coercion can cause many unnoticed errors in R.

R also offers functions to change from one type to another. For example, you can turn numbers into characters with:

```
x <- 1:5
y <- as.character(x)
y
#> [1] "1" "2" "3" "4" "5"
```

You can turn it back with `as.numeric`:

```
as.numeric(y)
#> [1] 1 2 3 4 5
```

This function is actually quite useful since datasets that include numbers as character strings are common.

2.7.1 Not availables (NA)

When a function tries to coerce one type to another and encounters an impossible case, it usually gives us a warning and turns the entry into a special value called an NA for “not available”. For example:

```
x <- c("1", "b", "3")
as.numeric(x)
#> Warning: NAs introduced by coercion
#> [1] 1 NA 3
```

R does not have any guesses for what number you want when you type `b`, so it does not try.

As a data scientist you will encounter the NAs often as they are generally used for missing data, a common problem in real-world datasets.

2.8 Exercises

1. Use the function `c` to create a vector with the average high temperatures in January for Beijing, Lagos, Paris, Rio de Janeiro, San Juan, and Toronto, which are 35, 88, 42, 84, 81, and 30 degrees Fahrenheit. Call the object `temp`.
2. Now create a vector with the city names and call the object `city`.
3. Use the `names` function and the objects defined in the previous exercises to associate the temperature data with its corresponding city.
4. Use the `[` and `:` operators to access the temperature of the first three cities on the list.
5. Use the `[` operator to access the temperature of Paris and San Juan.
6. Use the `:` operator to create a sequence of numbers 12, 13, 14, ..., 73.
7. Create a vector containing all the positive odd numbers smaller than 100.
8. Create a vector of numbers that starts at 6, does not pass 55, and adds numbers in increments of $4/7$: 6, $6 + 4/7$, $6 + 8/7$, and so on. How many numbers does the list have? Hint: use `seq` and `length`.
9. What is the class of the following object `a <- seq(1, 10, 0.5)`?
10. What is the class of the following object `a <- seq(1, 10)`?

11. The class of `class(a<-1)` is numeric, not integer. R defaults to numeric and to force an integer, you need to add the letter L. Confirm that the class of 1L is integer.

12. Define the following vector:

```
x <- c("1", "3", "5")
```

and coerce it to get integers.

2.9 Sorting

Now that we have mastered some basic R knowledge, let's try to gain some insights into the safety of different states in the context of gun murders.

2.9.1 sort

Say we want to rank the states from least to most gun murders. The function `sort` sorts a vector in increasing order. We can therefore see the largest number of gun murders by typing:

```
library(dslabs)
data(murders)
sort(murders$total)
#> [1] 2 4 5 5 7 8 11 12 12 16 19 21 22
#> [14] 27 32 36 38 53 63 65 67 84 93 93 97 97
#> [27] 99 111 116 118 120 135 142 207 219 232 246 250 286
#> [40] 293 310 321 351 364 376 413 457 517 669 805 1257
```

However, this does not give us information about which states have which murder totals. For example, we don't know which state had 1257.

2.9.2 order

The function `order` is closer to what we want. It takes a vector as input and returns the vector of indexes that sorts the input vector. This may sound confusing so let's look at a simple example. We can create a vector and sort it:

```
x <- c(31, 4, 15, 92, 65)
sort(x)
#> [1] 4 15 31 65 92
```

Rather than sort the input vector, the function `order` returns the index that sorts input vector:

```
index <- order(x)
x[index]
#> [1] 4 15 31 65 92
```

This is the same output as that returned by `sort(x)`. If we look at this index, we see why it works:

```
x
#> [1] 31 4 15 92 65
order(x)
#> [1] 2 3 1 5 4
```

The second entry of `x` is the smallest, so `order(x)` starts with 2. The next smallest is the third entry, so the second entry is 3 and so on.

How does this help us order the states by murders? First, remember that the entries of vectors you access with `$` follow the same order as the rows in the table. For example, these two vectors containing state names and abbreviations, respectively, are matched by their order:

```
murders$state[1:6]
#> [1] "Alabama" "Alaska" "Arizona" "Arkansas" "California"
#> [6] "Colorado"
murders$abb[1:6]
#> [1] "AL" "AK" "AZ" "AR" "CA" "CO"
```

This means we can order the state names by their total murders. We first obtain the index that orders the vectors according to murder totals and then index the state names vector:

```
ind <- order(murders$total)
murders$abb[ind]
#> [1] "VT" "ND" "NH" "WY" "HI" "SD" "ME" "ID" "MT" "RI" "AK" "IA" "UT"
#> [14] "WV" "NE" "OR" "DE" "MN" "KS" "CO" "NM" "NV" "AR" "WA" "CT" "WI"
#> [27] "DC" "OK" "KY" "MA" "MS" "AL" "IN" "SC" "TN" "AZ" "NJ" "VA" "NC"
#> [40] "MD" "OH" "MO" "LA" "IL" "GA" "MI" "PA" "NY" "FL" "TX" "CA"
```

According to the above, California had the most murders.

2.9.3 max and which.max

If we are only interested in the entry with the largest value, we can use `max` for the value:

```
max(murders$total)
#> [1] 1257
```

and `which.max` for the index of the largest value:

```
i_max <- which.max(murders$total)
murders$state[i_max]
#> [1] "California"
```

For the minimum, we can use `min` and `which.min` in the same way.

Does this mean California is the most dangerous state? In an upcoming section, we argue that we should be considering rates instead of totals. Before doing that, we introduce one last order-related function: `rank`.

2.9.4 rank

Although not as frequently used as `order` and `sort`, the function `rank` is also related to order and can be useful. For any given vector it returns a vector with the rank of the first entry, second entry, etc., of the input vector. Here is a simple example:

```
x <- c(31, 4, 15, 92, 65)
rank(x)
#> [1] 3 1 2 5 4
```

To summarize, let's look at the results of the three functions we have introduced:

original	sort	order	rank
31	4	2	3
4	15	3	1
15	31	1	2
92	65	5	5
65	92	4	4

2.9.5 Beware of recycling

Another common source of unnoticed errors in R is the use of *recycling*. We saw that vectors are added elementwise. So if the vectors don't match in length, it is natural to assume that we should get an error. But we don't. Notice what happens:

```
x <- c(1,2,3)
y <- c(10, 20, 30, 40, 50, 60, 70)
x+y
#> Warning in x + y: longer object length is not a multiple of shorter
#> object length
#> [1] 11 22 33 41 52 63 71
```

We do get a warning, but no error. For the output, R has recycled the numbers in `x`. Notice the last digit of numbers in the output.

2.10 Exercises

For these exercises we will use the US murders dataset. Make sure you load it prior to starting.

```
library(dslabs)
data("murders")
```

1. Use the `$` operator to access the population size data and store it as the object `pop`. Then use the `sort` function to redefine `pop` so that it is sorted. Finally, use the `[]` operator to report the smallest population size.
2. Now instead of the smallest population size, find the index of the entry with the smallest population size. Hint: use `order` instead of `sort`.
3. We can actually perform the same operation as in the previous exercise using the function `which.min`. Write one line of code that does this.
4. Now we know how small the smallest state is and we know which row represents it. Which state is it? Define a variable `states` to be the state names from the `murders` data frame. Report the name of the state with the smallest population.
5. You can create a data frame using the `data.frame` function. Here is a quick example:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
         "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Use the `rank` function to determine the population rank of each state from smallest population size to biggest. Save these ranks in an object called `ranks`, then create a data frame with the state name and its rank. Call the data frame `my_df`.

6. Repeat the previous exercise, but this time order `my_df` so that the states are ordered from least populous to most populous. Hint: create an object `ind` that stores the indexes needed to order the population values. Then use the bracket operator `[]` to re-order each column in the data frame.
7. The `na_example` vector represents a series of counts. You can quickly examine the object using:

```
data("na_example")
str(na_example)
#> int [1:1000] 2 1 3 2 1 3 1 4 3 2 ...
```

However, when we compute the average with the function `mean`, we obtain an NA:

```
mean(na_example)
#> [1] NA
```

The `is.na` function returns a logical vector that tells us which entries are NA. Assign this logical vector to an object called `ind` and determine how many NAs does `na_example` have.

8. Now compute the average again, but only for the entries that are not NA. Hint: remember the `!` operator.

2.11 Vector arithmetics

California had the most murders, but does this mean it is the most dangerous state? What if it just has many more people than any other state? We can quickly confirm that California indeed has the largest population:

```
library(dslabs)
data("murders")
murders$state[which.max(murders$population)]
#> [1] "California"
```

with over 37 million inhabitants. It is therefore unfair to compare the totals if we are interested in learning how safe the state is. What we really should be computing is the murders per capita. The reports we describe in the motivating section used murders per 100,000 as the unit. To compute this quantity, the powerful vector arithmetic capabilities of R come in handy.

2.11.1 Rescaling a vector

In R, arithmetic operations on vectors occur *element-wise*. For a quick example, suppose we have height in inches:

```
inches <- c(69, 62, 66, 70, 70, 73, 67, 73, 67, 70)
```

and want to convert to centimeters. Notice what happens when we multiply `inches` by 2.54:

```
inches * 2.54
#> [1] 175 157 168 178 178 185 170 185 170 178
```

In the line above, we multiplied each element by 2.54. Similarly, if for each entry we want to compute how many inches taller or shorter than 69 inches, the average height for males, we can subtract it from every entry like this:

```
inches - 69
#> [1] 0 -7 -3 1 1 4 -2 4 -2 1
```

2.11.2 Two vectors

If we have two vectors of the same length, and we sum them in R, they will be added entry by entry as follows:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} a + e \\ b + f \\ c + g \\ d + h \end{pmatrix}$$

The same holds for other mathematical operations, such as $-$, $*$ and $/$.

This implies that to compute the murder rates we can simply type:

```
murder_rate <- murders$total / murders$population * 100000
```

Once we do this, we notice that California is no longer near the top of the list. In fact, we can use what we have learned to order the states by murder rate:

```
murders$abb[order(murder_rate)]
#> [1] "VT" "NH" "HI" "ND" "IA" "ID" "UT" "ME" "WY" "OR" "SD" "MN" "MT"
#> [14] "CO" "WA" "WV" "RI" "WI" "NE" "MA" "IN" "KS" "NY" "KY" "AK" "OH"
#> [27] "CT" "NJ" "AL" "IL" "OK" "NC" "NV" "VA" "AR" "TX" "NM" "CA" "FL"
#> [40] "TN" "PA" "AZ" "GA" "MS" "MI" "DE" "SC" "MD" "MO" "LA" "DC"
```

2.12 Exercises

1. Previously we created this data frame:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
         "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Remake the data frame using the code above, but add a line that converts the temperature from Fahrenheit to Celsius. The conversion is $C = \frac{5}{9} \times (F - 32)$.

2. What is the following sum $1 + 1/2^2 + 1/3^2 + \dots 1/100^2$? Hint: thanks to Euler, we know it should be close to $\pi^2/6$.
3. Compute the per 100,000 murder rate for each state and store it in the object `murder_rate`. Then compute the average murder rate for the US using the function `mean`. What is the average?

2.13 Indexing

R provides a powerful and convenient way of indexing vectors. We can, for example, subset a vector based on properties of another vector. In this section, we continue working with our US murders example, which we can load like this:

```
library(dslabs)
data("murders")
```

2.13.1 Subsetting with logicals

We have now calculated the murder rate using:

```
murder_rate <- murders$total / murders$population * 100000
```

Imagine you are moving from Italy where, according to an ABC news report, the murder rate is only 0.71 per 100,000. You would prefer to move to a state with a similar murder rate. Another powerful feature of R is that we can use logicals to index vectors. If we compare a vector to a single number, it actually performs the test for each entry. The following is an example related to the question above:

```
ind <- murder_rate < 0.71
```

If we instead want to know if a value is less or equal, we can use:

```
ind <- murder_rate <= 0.71
```

Note that we get back a logical vector with `TRUE` for each entry smaller than or equal to 0.71. To see which states these are, we can leverage the fact that vectors can be indexed with logicals.

```
murders$state[ind]
#> [1] "Hawaii"      "Iowa"        "New Hampshire" "North Dakota"
#> [5] "Vermont"
```

In order to count how many are `TRUE`, the function `sum` returns the sum of the entries of a vector and logical vectors get *coerced* to numeric with `TRUE` coded as 1 and `FALSE` as 0. Thus we can count the states using:

```
sum(ind)
#> [1] 5
```

2.13.2 Logical operators

Suppose we like the mountains and we want to move to a safe state in the western region of the country. We want the murder rate to be at most 1. In this case, we want two different things to be true. Here we can use the logical operator *and*, which in R is represented with `&`. This operation results in `TRUE` only when both logicals are `TRUE`. To see this, consider this example:

```
TRUE & TRUE
#> [1] TRUE
TRUE & FALSE
#> [1] FALSE
FALSE & FALSE
#> [1] FALSE
```

For our example, we can form two logicals:

```
west <- murders$region == "West"
safe <- murder_rate <= 1
```

and we can use the `&` to get a vector of logicals that tells us which states satisfy both conditions:

```
ind <- safe & west
murders$state[ind]
#> [1] "Hawaii" "Idaho" "Oregon" "Utah" "Wyoming"
```

2.13.3 which

Suppose we want to look up California's murder rate. For this type of operation, it is convenient to convert vectors of logicals into indexes instead of keeping long vectors of logicals. The function `which` tells us which entries of a logical vector are TRUE. So we can type:

```
ind <- which(murders$state == "California")
murder_rate[ind]
#> [1] 3.37
```

2.13.4 match

If instead of just one state we want to find out the murder rates for several states, say New York, Florida, and Texas, we can use the function `match`. This function tells us which indexes of a second vector match each of the entries of a first vector:

```
ind <- match(c("New York", "Florida", "Texas"), murders$state)
ind
#> [1] 33 10 44
```

Now we can look at the murder rates:

```
murder_rate[ind]
#> [1] 2.67 3.40 3.20
```

2.13.5 %in%

If rather than an index we want a logical that tells us whether or not each element of a first vector is in a second, we can use the function `%in%`. Let's imagine you are not sure if Boston, Dakota, and Washington are states. You can find out like this:

```
c("Boston", "Dakota", "Washington") %in% murders$state
#> [1] FALSE FALSE TRUE
```

Note that we will be using `%in%` often throughout the book.

Advanced: There is a connection between `match` and `%in%` through `which`. To see this, notice that the following two lines produce the same index (although in different order):

```
match(c("New York", "Florida", "Texas"), murders$state)
#> [1] 33 10 44
which(murders$state%in%c("New York", "Florida", "Texas"))
#> [1] 10 33 44
```

2.14 Exercises

Start by loading the library and data.

```
library(dslabs)
data(murders)
```

1. Compute the per 100,000 murder rate for each state and store it in an object called `murder_rate`. Then use logical operators to create a logical vector named `low` that tells us which entries of `murder_rate` are lower than 1.
2. Now use the results from the previous exercise and the function `which` to determine the indices of `murder_rate` associated with values lower than 1.
3. Use the results from the previous exercise to report the names of the states with murder rates lower than 1.
4. Now extend the code from exercises 2 and 3 to report the states in the Northeast with murder rates lower than 1. Hint: use the previously defined logical vector `low` and the logical operator `&`.
5. In a previous exercise we computed the murder rate for each state and the average of these numbers. How many states are below the average?
6. Use the `match` function to identify the states with abbreviations AK, MI, and IA. Hint: start by defining an index of the entries of `murders$abb` that match the three abbreviations, then use the `[]` operator to extract the states.
7. Use the `%in%` operator to create a logical vector that answers the question: which of the following are actual abbreviations: MA, ME, MI, MO, MU?

8. Extend the code you used in exercise 7 to report the one entry that is **not** an actual abbreviation. Hint: use the `!` operator, which turns `FALSE` into `TRUE` and vice versa, then `which` to obtain an index.

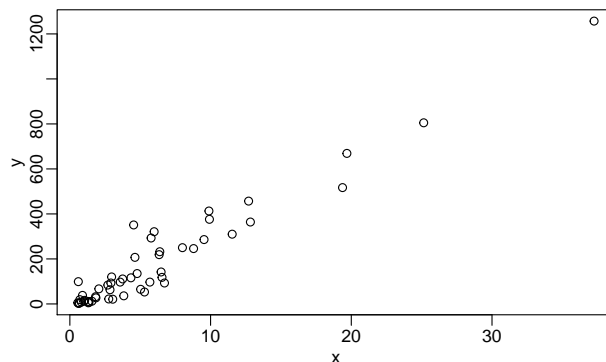
2.15 Basic plots

In Chapter 7 we describe an add-on package that provides a powerful approach to producing plots in R. We then have an entire part on Data Visualization in which we provide many examples. Here we briefly describe some of the functions that are available in a basic R installation.

2.15.1 plot

The `plot` function can be used to make scatterplots. Here is a plot of total murders versus population.

```
x <- murders$population / 10^6
y <- murders$total
plot(x, y)
```



For a quick plot that avoids accessing variables twice, we can use the `with` function:

```
with(murders, plot(population, total))
```

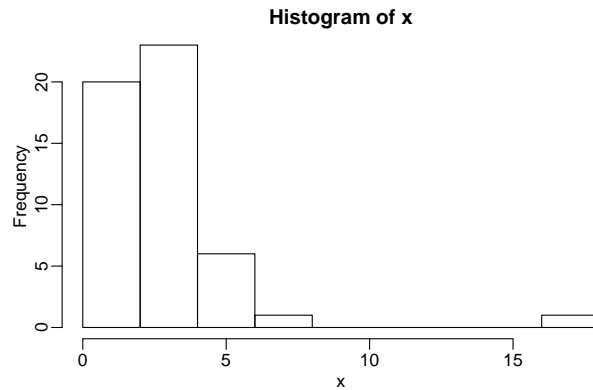
The function `with` lets us use the `murders` column names in the `plot` function. It also works with any data frames and any function.

2.15.2 hist

We will describe histograms as they relate to distributions in the Data Visualization part of the book. Here we will simply note that histograms are a powerful graphical summary of

a list of numbers that gives you a general overview of the types of values you have. We can make a histogram of our murder rates by simply typing:

```
x <- with(murders, total / population * 100000)
hist(x)
```



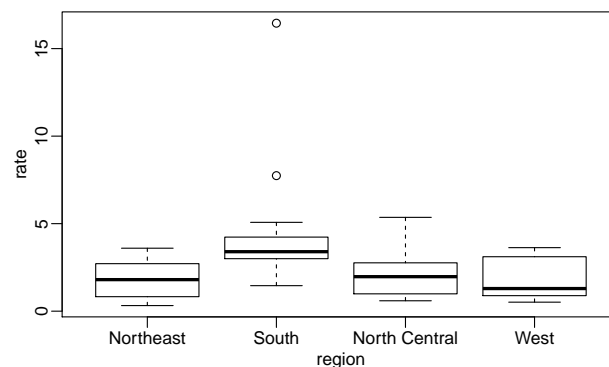
We can see that there is a wide range of values with most of them between 2 and 3 and one very extreme case with a murder rate of more than 15:

```
murders$state[which.max(x)]
#> [1] "District of Columbia"
```

2.15.3 boxplot

Boxplots will also be described in the Data Visualization part of the book. They provide a more terse summary than histograms, but they are easier to stack with other boxplots. For example, here we can use them to compare the different regions:

```
murders$rate <- with(murders, total / population * 100000)
boxplot(rate~region, data = murders)
```

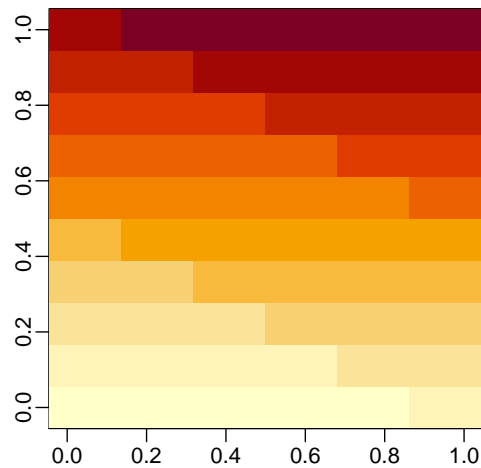


We can see that the South has higher murder rates than the other three regions.

2.15.4 image

The `image` function displays the values in a matrix using color. Here is a quick example:

```
x <- matrix(1:120, 12, 10)
image(x)
```



2.16 Exercises

1. We made a plot of total murders versus population and noted a strong relationship. Not surprisingly, states with larger populations had more murders.

```
library(dslabs)
data(murders)
population_in_millions <- murders$population/10^6
total_gun_murders <- murders$total
plot(population_in_millions, total_gun_murders)
```

Keep in mind that many states have populations below 5 million and are bunched up. We may gain further insights from making this plot in the log scale. Transform the variables using the `log10` transformation and then plot them.

2. Create a histogram of the state populations.
3. Generate boxplots of the state populations by region.

3

Programming basics

We teach R because it greatly facilitates data analysis, the main topic of this book. By coding in R, we can efficiently perform exploratory data analysis, build data analysis pipelines, and prepare data visualization to communicate results. However, R is not just a data analysis environment but a programming language. Advanced R programmers can develop complex packages and even improve R itself, but we do not cover advanced programming in this book. Nonetheless, in this section, we introduce three key programming concepts: conditional expressions, for-loops, and functions. These are not just key building blocks for advanced programming, but are sometimes useful during data analysis. We also note that there are several functions that are widely used to program in R but that we will not cover in this book. These include `split`, `cut`, `do.call`, and `Reduce`, as well as the `data.table` package. These are worth learning if you plan to become an expert R programmer.

3.1 Conditional expressions

Conditional expressions are one of the basic features of programming. They are used for what is called *flow control*. The most common conditional expression is the if-else statement. In R, we can actually perform quite a bit of data analysis without conditionals. However, they do come up occasionally, and you will need them once you start writing your own functions and packages.

Here is a very simple example showing the general structure of an if-else statement. The basic idea is to print the reciprocal of `a` unless `a` is 0:

```
a <- 0

if(a!=0){
  print(1/a)
} else{
  print("No reciprocal for 0.")
}
#> [1] "No reciprocal for 0."
```

Let's look at one more example using the US murders data frame:

```
library(dslabs)
data(murders)
murder_rate <- murders$total / murders$population*100000
```


Here is a very simple example that tells us which states, if any, have a murder rate lower than 0.5 per 100,000. The `if` statement protects us from the case in which no state satisfies the condition.

```
ind <- which.min(murder_rate)

if(murder_rate[ind] < 0.5){
  print(murders$state[ind])
} else{
  print("No state has murder rate that low")
}
#> [1] "Vermont"
```

If we try it again with a rate of 0.25, we get a different answer:

```
if(murder_rate[ind] < 0.25){
  print(murders$state[ind])
} else{
  print("No state has a murder rate that low.")
}
#> [1] "No state has a murder rate that low."
```

A related function that is very useful is `ifelse`. This function takes three arguments: a logical and two possible answers. If the logical is `TRUE`, the value in the second argument is returned and if `FALSE`, the value in the third argument is returned. Here is an example:

```
a <- 0
ifelse(a > 0, 1/a, NA)
#> [1] NA
```

The function is particularly useful because it works on vectors. It examines each entry of the logical vector and returns elements from the vector provided in the second argument, if the entry is `TRUE`, or elements from the vector provided in the third argument, if the entry is `FALSE`.

```
a <- c(0, 1, 2, -4, 5)
result <- ifelse(a > 0, 1/a, NA)
```

This table helps us see what happened:

a	is_a_positive	answer1	answer2	result
0	FALSE	Inf	NA	NA
1	TRUE	1.00	NA	1.0
2	TRUE	0.50	NA	0.5
-4	FALSE	-0.25	NA	NA
5	TRUE	0.20	NA	0.2

Here is an example of how this function can be readily used to replace all the missing values in a vector with zeros:

```
data(na_example)
no_nas <- ifelse(is.na(na_example), 0, na_example)
sum(is.na(no_nas))
#> [1] 0
```

Two other useful functions are `any` and `all`. The `any` function takes a vector of logicals and returns `TRUE` if any of the entries is `TRUE`. The `all` function takes a vector of logicals and returns `TRUE` if all of the entries are `TRUE`. Here is an example:

```
z <- c(TRUE, TRUE, FALSE)
any(z)
#> [1] TRUE
all(z)
#> [1] FALSE
```

3.2 Defining functions

As you become more experienced, you will find yourself needing to perform the same operations over and over. A simple example is computing averages. We can compute the average of a vector `x` using the `sum` and `length` functions: `sum(x)/length(x)`. Because we do this repeatedly, it is much more efficient to write a function that performs this operation. This particular operation is so common that someone already wrote the `mean` function and it is included in base R. However, you will encounter situations in which the function does not already exist, so R permits you to write your own. A simple version of a function that computes the average can be defined like this:

```
avg <- function(x){
  s <- sum(x)
  n <- length(x)
  s/n
}
```

Now `avg` is a function that computes the mean:

```
x <- 1:100
identical(mean(x), avg(x))
#> [1] TRUE
```

Notice that variables defined inside a function are not saved in the workspace. So while we use `s` and `n` when we call `avg`, the values are created and changed only during the call. Here is an illustrative example:

```
s <- 3
avg(1:10)
#> [1] 5.5
s
#> [1] 3
```

Note how `s` is still 3 after we call `avg`.

In general, functions are objects, so we assign them to variable names with `<-`. The function `function` tells R you are about to define a function. The general form of a function definition looks like this:

```
my_function <- function(VARIABLE_NAME){
  perform operations on VARIABLE_NAME and calculate VALUE
  VALUE
}
```

The functions you define can have multiple arguments as well as default values. For example, we can define a function that computes either the arithmetic or geometric average depending on a user defined variable like this:

```
avg <- function(x, arithmetic = TRUE){
  n <- length(x)
  ifelse(arithmetic, sum(x)/n, prod(x)^(1/n))
}
```

We will learn more about how to create functions through experience as we face more complex tasks.

3.3 Namespaces

Once you start becoming more of an R expert user, you will likely need to load several add-on packages for some of your analysis. Once you start doing this, it is likely that two packages use the same name for two different functions. And often these functions do completely different things. In fact, you have already encountered this because both **dplyr** and the R-base **stats** package define a `filter` function. There are five other examples in **dplyr**. We know this because when we first load **dplyr** we see the following message:

The following objects are masked from ‘package:stats’:

```
filter, lag
```

The following objects are masked from ‘package:base’:

```
intersect, setdiff, setequal, union
```

So what does R do when we type `filter`? Does it use the **dplyr** function or the **stats** function? From our previous work we know it uses the **dplyr** one. But what if we want to use the **stats** version?

These functions live in different *namespaces*. R will follow a certain order when searching for a function in these *namespaces*. You can see the order by typing:

```
search()
```

The first entry in this list is the global environment which includes all the objects you define.

So what if we want to use the **stats filter** instead of the **dplyr filter** but **dplyr** appears first in the search list? You can force the use of a specific name space by using double colons (**::**) like this:

```
stats::filter
```

If we want to be absolutely sure we use the **dplyr filter** we can use

```
dplyr::filter
```

Also note that if we want to use a function in a package without loading the entire package, we can use the double colon as well.

For more on this more advanced topic we recommend the R packages book¹.

3.4 For-loops

The formula for the sum of the series $1 + 2 + \dots + n$ is $n(n+1)/2$. What if we weren't sure that was the right function? How could we check? Using what we learned about functions we can create one that computes the S_n :

```
compute_s_n <- function(n){
  x <- 1:n
  sum(x)
}
```

How can we compute S_n for various values of n , say $n = 1, \dots, 25$? Do we write 25 lines of code calling **compute_s_n**? No, that is what for-loops are for in programming. In this case, we are performing exactly the same task over and over, and the only thing that is changing is the value of n . For-loops let us define the range that our variable takes (in our example $n = 1, \dots, 10$), then change the value and evaluate expression as you *loop*.

Perhaps the simplest example of a for-loop is this useless piece of code:

```
for(i in 1:5){
  print(i)
}
#> [1] 1
#> [1] 2
#> [1] 3
#> [1] 4
#> [1] 5
```

¹<http://r-pkgs.had.co.nz/namespaces.html>

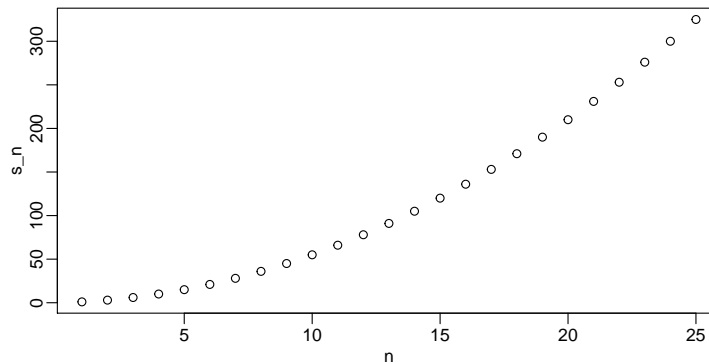
Here is the for-loop we would write for our S_n example:

```
m <- 25
s_n <- vector(length = m) # create an empty vector
for(n in 1:m){
  s_n[n] <- compute_s_n(n)
}
```

In each iteration $n = 1, n = 2$, etc..., we compute S_n and store it in the n th entry of `s_n`.

Now we can create a plot to search for a pattern:

```
n <- 1:m
plot(n, s_n)
```



If you noticed that it appears to be a quadratic, you are on the right track because the formula is $n(n+1)/2$.

3.5 Vectorization and functionals

Although for-loops are an important concept to understand, in R we rarely use them. As you learn more R, you will realize that *vectorization* is preferred over for-loops since it results in shorter and clearer code. We already saw examples in the Vector Arithmetic section. A *vectorized* function is a function that will apply the same operation on each of the vectors.

```
x <- 1:10
sqrt(x)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
y <- 1:10
x*y
#> [1] 1 4 9 16 25 36 49 64 81 100
```

To make this calculation, there is no need for for-loops. However, not all functions work this

way. For instance, the function we just wrote, `compute_s_n`, does not work element-wise since it is expecting a scalar. This piece of code does not run the function on each entry of `n`:

```
n <- 1:25
compute_s_n(n)
```

Functionals are functions that help us apply the same function to each entry in a vector, matrix, data frame, or list. Here we cover the functional that operates on numeric, logical, and character vectors: `sapply`.

The function `sapply` permits us to perform element-wise operations on any function. Here is how it works:

```
x <- 1:10
sapply(x, sqrt)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

Each element of `x` is passed on to the function `sqrt` and the result is returned. These results are concatenated. In this case, the result is a vector of the same length as the original `x`. This implies that the for-loop above can be written as follows:

```
n <- 1:25
s_n <- sapply(n, compute_s_n)
```

Other functionals are `apply`, `lapply`, `tapply`, `mapply`, `vapply`, and `replicate`. We mostly use `sapply`, `apply`, and `replicate` in this book, but we recommend familiarizing yourselves with the others as they can be very useful.

3.6 Exercises

1. What will this conditional expression return?

```
x <- c(1,2,-3,4)

if(all(x>0)){
  print("All Postives")
} else{
  print("Not all positives")
}
```

2. Which of the following expressions is always **FALSE** when at least one entry of a logical vector `x` is **TRUE**?

- a. `all(x)`
- b. `any(x)`

- c. `any(!x)`
- d. `all(!x)`

3. The function `nchar` tells you how many characters long a character vector is. Write a line of code that assigns to the object `new_names` the state abbreviation when the state name is longer than 8 characters.

4. Create a function `sum_n` that for any given value, say n , computes the sum of the integers from 1 to n (inclusive). Use the function to determine the sum of integers from 1 to 5,000.

5. Create a function `altman_plot` that takes two arguments, x and y , and plots the difference against the sum.

6. After running the code below, what is the value of `x`?

```
x <- 3
my_func <- function(y){
  x <- 5
  y+5
}
```

7. Write a function `compute_s_n` that for any given n computes the sum $S_n = 1^2 + 2^2 + 3^2 + \dots n^2$. Report the value of the sum when $n = 10$.

8. Define an empty numerical vector `s_n` of size 25 using `s_n <- vector("numeric", 25)` and store in the results of S_1, S_2, \dots, S_{25} using a for-loop.

9. Repeat exercise 8, but this time use `sapply`.

10. Repeat exercise 8, but this time use `map_dbl`.

11. Plot S_n versus n . Use points defined by $n = 1, \dots, 25$.

12. Confirm that the formula for this sum is $S_n = n(n+1)(2n+1)/6$.

4

The tidyverse

Up to now we have been manipulating vectors by reordering and subsetting them through indexing. However, once we start more advanced analyses, the preferred unit for data storage is not the vector but the data frame. In this chapter we learn to work directly with data frames, which greatly facilitate the organization of information. We will be using data frames for the majority of this book. We will focus on a specific data format referred to as *tidy* and on specific collection of packages that are particularly helpful for working with *tidy* data referred to as the *tidyverse*.

We can load all the tidyverse packages at once by installing and loading the **tidyverse** package:

```
library(tidyverse)
```

We will learn how to implement the tidyverse approach throughout the book, but before delving into the details, in this chapter we introduce some of the most widely used tidyverse functionality, starting with the **dplyr** package for manipulating data frames and the **purrr** package for working with functions. Note that the tidyverse also includes a graphing package, **ggplot2**, which we introduce later in Chapter 7 in the Data Visualization part of the book; the **readr** package discussed in Chapter 5; and many others. In this chapter, we first introduce the concept of *tidy data* and then demonstrate how we use the tidyverse to work with data frames in this format.

4.1 Tidy data

We say that a data table is in *tidy* format if each row represents one observation and columns represent the different variables available for each of these observations. The **murders** dataset is an example of a tidy data frame.

```
#>      state abb region population total
#> 1  Alabama AL  South    4779736   135
#> 2   Alaska AK   West     710231    19
#> 3  Arizona AZ   West    6392017   232
#> 4  Arkansas AR  South    2915918    93
#> 5 California CA   West   37253956  1257
#> 6  Colorado CO   West    5029196    65
```

Each row represent a state with each of the five columns providing a different variable related to these states: name, abbreviation, region, population, and total murders.

To see how the same information can be provided in different formats, consider the following example:

```
#>      country year fertility
#> 1    Germany 1960      2.41
#> 2 South Korea 1960      6.16
#> 3    Germany 1961      2.44
#> 4 South Korea 1961      5.99
#> 5    Germany 1962      2.47
#> 6 South Korea 1962      5.79
```

This tidy dataset provides fertility rates for two countries across the years. This is a tidy dataset because each row presents one observation with the three variables being country, year, and fertility rate. However, this dataset originally came in another format and was reshaped for the **dslabs** package. Originally, the data was in the following format:

```
#>      country 1960 1961 1962
#> 1    Germany 2.41 2.44 2.47
#> 2 South Korea 6.16 5.99 5.79
```

The same information is provided, but there are two important differences in the format: 1) each row includes several observations and 2) one of the variables, year, is stored in the header. For the tidyverse packages to be optimally used, data need to be reshaped into **tidy** format, which you will learn to do in the Data Wrangling part of the book. Until then, we will use example datasets that are already in tidy format.

Although not immediately obvious, as you go through the book you will start to appreciate the advantages of working in a framework in which functions use tidy formats for both inputs and outputs. You will see how this permits the data analyst to focus on more important aspects of the analysis rather than the format of the data.

4.2 Exercises

1. Examine the built-in dataset `co2`. Which of the following is true:

- `co2` is tidy data: it has one year for each row.
- `co2` is not tidy: we need at least one column with a character vector.
- `co2` is not tidy: it is a matrix instead of a data frame.
- `co2` is not tidy: to be tidy we would have to wrangle it to have three columns (year, month and value), then each `co2` observation would have a row.

2. Examine the built-in dataset `ChickWeight`. Which of the following is true:

- `ChickWeight` is not tidy: each chick has more than one row.
- `ChickWeight` is tidy: each observation (a weight) is represented by one row. The chick from which this measurement came is one of the variables.

- c. `ChickWeight` is not tidy: we are missing the year column.
 - d. `ChickWeight` is tidy: it is stored in a data frame.
3. Examine the built-in dataset `BOD`. Which of the following is true:
- a. `BOD` is not tidy: it only has six rows.
 - b. `BOD` is not tidy: the first column is just an index.
 - c. `BOD` is tidy: each row is an observation with two values (time and demand)
 - d. `BOD` is tidy: all small datasets are tidy by definition.
4. Which of the following built-in datasets is tidy (you can pick more than one):
- a. `BJsales`
 - b. `EuStockMarkets`
 - c. `DNase`
 - d. `Formaldehyde`
 - e. `Orange`
 - f. `UCBAdmissions`

4.3 Manipulating data frames

The **dplyr** package from the **tidyverse** introduces functions that perform some of the most common operations when working with data frames and uses names for these functions that are relatively easy to remember. For instance, to change the data table by adding a new column, we use `mutate`. To filter the data table to a subset of rows, we use `filter`. Finally, to subset the data by selecting specific columns, we use `select`.

4.3.1 Adding a column with `mutate`

We want all the necessary information for our analysis to be included in the data table. So the first task is to add the murder rates to our murders data frame. The function `mutate` takes the data frame as a first argument and the name and values of the variable as a second argument using the convention `name = values`. So, to add murder rates, we use:

```
library(dslabs)
data("murders")
murders <- mutate(murders, rate = total / population * 100000)
```

Notice that here we used `total` and `population` inside the function, which are objects that are **not** defined in our workspace. But why don't we get an error?

This is one of **dplyr**'s main features. Functions in this package, such as `mutate`, know to look for variables in the data frame provided in the first argument. In the call to `mutate` above, `total` will have the values in `murders$total`. This approach makes the code much more readable.

We can see that the new column is added:

```
head(murders)
#>      state abb region population total rate
#> 1  Alabama AL  South    4779736    135 2.82
#> 2  Alaska  AK   West     710231     19 2.68
#> 3  Arizona AZ   West    6392017    232 3.63
#> 4  Arkansas AR  South    2915918     93 3.19
#> 5 California CA   West    37253956  1257 3.37
#> 6  Colorado CO   West     5029196     65 1.29
```

Although we have overwritten the original `murders` object, this does not change the object that loaded with `data(murders)`. If we load the `murders` data again, the original will overwrite our mutated version.

4.3.2 Subsetting with filter

Now suppose that we want to filter the data table to only show the entries for which the murder rate is lower than 0.71. To do this we use the `filter` function, which takes the data table as the first argument and then the conditional statement as the second. Like `mutate`, we can use the unquoted variable names from `murders` inside the function and it will know we mean the columns and not objects in the workspace.

```
filter(murders, rate <= 0.71)
#>      state abb      region population total  rate
#> 1  Hawaii  HI      West    1360301      7 0.515
#> 2   Iowa  IA North Central    3046355     21 0.689
#> 3 New Hampshire NH    Northeast    1316470      5 0.380
#> 4 North Dakota ND North Central     672591      4 0.595
#> 5  Vermont VT    Northeast     625741      2 0.320
```

4.3.3 Selecting columns with select

Although our data table only has six columns, some data tables include hundreds. If we want to view just a few, we can use the `dplyr` `select` function. In the code below we select three columns, assign this to a new object and then filter the new object:

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
#>      state      region  rate
#> 1  Hawaii      West 0.515
#> 2   Iowa North Central 0.689
#> 3 New Hampshire Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5  Vermont Northeast 0.320
```

In the call to `select`, the first argument `murders` is an object, but `state`, `region`, and `rate` are variable names.

4.4 Exercises

1. Load the **dplyr** package and the murders dataset.

```
library(dplyr)
library(dslabs)
data(murders)
```

You can add columns using the **dplyr** function `mutate`. This function is aware of the column names and inside the function you can call them unquoted:

```
murders <- mutate(murders, population_in_millions = population / 10^6)
```

We can write `population` rather than `murders$population`. The function `mutate` knows we are grabbing columns from `murders`.

Use the function `mutate` to add a murders column named `rate` with the per 100,000 murder rate as in the example code above. Make sure you redefine `murders` as done in the example code above (`murders <- [your code]`) so we can keep using this variable.

2. If `rank(x)` gives you the ranks of `x` from lowest to highest, `rank(-x)` gives you the ranks from highest to lowest. Use the function `mutate` to add a column `rank` containing the rank, from highest to lowest murder rate. Make sure you redefine `murders` so we can keep using this variable.

3. With **dplyr**, we can use `select` to show only certain columns. For example, with this code we would only show the states and population sizes:

```
select(murders, state, population) %>% head()
```

Use `select` to show the state names and abbreviations in `murders`. Do not redefine `murders`, just show the results.

4. The **dplyr** function `filter` is used to choose specific rows of the data frame to keep. Unlike `select` which is for columns, `filter` is for rows. For example, you can show just the New York row like this:

```
filter(murders, state == "New York")
```

You can use other logical vectors to filter rows.

Use `filter` to show the top 5 states with the highest murder rates. After we add murder rate and rank, do not change the murders dataset, just show the result. Remember that you can filter based on the `rank` column.

5. We can remove rows using the `!=` operator. For example, to remove Florida, we would do this:

```
no_florida <- filter(murders, state != "Florida")
```

Create a new data frame called `no_south` that removes states from the South region. How many states are in this category? You can use the function `nrow` for this.

6. We can also use `%in%` to filter with **dplyr**. You can therefore see the data from New York and Texas like this:

```
filter(murders, state %in% c("New York", "Texas"))
```

Create a new data frame called `murders_nw` with only the states from the Northeast and the West. How many states are in this category?

7. Suppose you want to live in the Northeast or West **and** want the murder rate to be less than 1. We want to see the data for the states satisfying these options. Note that you can use logical operators with `filter`. Here is an example in which we filter to keep only small states in the Northeast region.

```
filter(murders, population < 5000000 & region == "Northeast")
```

Make sure `murders` has been defined with `rate` and `rank` and still has all states. Create a table called `my_states` that contains rows for states satisfying both the conditions: it is in the Northeast or West and the murder rate is less than 1. Use `select` to show only the state name, the rate, and the rank.

4.5 The pipe: `%>%`

With **dplyr** we can perform a series of operations, for example `select` and then `filter`, by sending the results of one function to another using what is called the *pipe operator*: `%>%`. Some details are included below.

We wrote code above to show three variables (state, region, rate) for states that have murder rates below 0.71. To do this, we defined the intermediate object `new_table`. In **dplyr** we can write code that looks more like a description of what we want to do without intermediate objects:

original data → select → filter

For such an operation, we can use the pipe `%>%`. The code looks like this:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
#>      state      region rate
#> 1   Hawaii         West 0.515
#> 2    Iowa North Central 0.689
#> 3 New Hampshire Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5   Vermont      Northeast 0.320
```

This line of code is equivalent to the two lines of code above. What is going on here?

In general, the pipe *sends* the result of the left side of the pipe to be the first argument of the function on the right side of the pipe. Here is a very simple example:

```
16 %>% sqrt()
#> [1] 4
```

We can continue to pipe values along:

```
16 %>% sqrt() %>% log2()
#> [1] 2
```

The above statement is equivalent to `log2(sqrt(16))`.

Remember that the pipe sends values to the first argument, so we can define other arguments as if the first argument is already defined:

```
16 %>% sqrt() %>% log(base = 2)
#> [1] 2
```

Therefore, when using the pipe with data frames and **dplyr**, we no longer need to specify the required first argument since the **dplyr** functions we have described all take the data as the first argument. In the code we wrote:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
```

`murders` is the first argument of the `select` function, and the new data frame (formerly `new_table`) is the first argument of the `filter` function.

Note that the pipe works well with functions where the first argument is the input data. Functions in **tidyverse** packages like **dplyr** have this format and can be used easily with the pipe.

4.6 Exercises

1. The pipe `%>%` can be used to perform operations sequentially without having to define intermediate objects. Start by redefining `murder` to include `rate` and `rank`.

```
murders <- mutate(murders, rate = total / population * 100000,
                  rank = rank(-rate))
```

In the solution to the previous exercise, we did the following:

```
my_states <- filter(murders, region %in% c("Northeast", "West") &
                  rate < 1)

select(my_states, state, rate, rank)
```

The pipe `%>%` permits us to perform both operations sequentially without having to define an intermediate variable `my_states`. We therefore could have mutated and selected in the same line like this:

```
mutate(murders, rate = total / population * 100000,
       rank = rank(-rate)) %>%
  select(state, rate, rank)
```

Notice that `select` no longer has a data frame as the first argument. The first argument is assumed to be the result of the operation conducted right before the `%>%`.

Repeat the previous exercise, but now instead of creating a new object, show the result and only include the state, rate, and rank columns. Use a pipe `%>%` to do this in just one line.

2. Reset `murders` to the original table by using `data(murders)`. Use a pipe to create a new data frame called `my_states` that considers only states in the Northeast or West which have a murder rate lower than 1, and contains only the state, rate and rank columns. The pipe should also have four components separated by three `%>%`. The code should look something like this:

```
my_states <- murders %>%
  mutate SOMETHING %>%
  filter SOMETHING %>%
  select SOMETHING
```

4.7 Summarizing data

An important part of exploratory data analysis is summarizing data. The average and standard deviation are two examples of widely used summary statistics. More informative summaries can often be achieved by first splitting data into groups. In this section, we cover two new **dplyr** verbs that make these computations easier: `summarize` and `group_by`. We learn to access resulting values using the `pull` function.

4.7.1 summarize

The `summarize` function in **dplyr** provides a way to compute summary statistics with intuitive and readable code. We start with a simple example based on heights. The `heights` dataset includes heights and sex reported by students in an in-class survey.

```
library(dplyr)
library(dslabs)
data(heights)
```

The following code computes the average and standard deviation for females:

```
s <- heights %>%
  filter(sex == "Female") %>%
  summarize(average = mean(height), standard_deviation = sd(height))
s
#>   average standard_deviation
#> 1    64.9             3.76
```

This takes our original data table as input, filters it to keep only females, and then produces a new summarized table with just the average and the standard deviation of heights. We get to choose the names of the columns of the resulting table. For example, above we decided to use `average` and `standard_deviation`, but we could have used other names just the same.

Because the resulting table stored in `s` is a data frame, we can access the components with the accessor `$`:

```
s$average
#> [1] 64.9
s$standard_deviation
#> [1] 3.76
```

As with most other **dplyr** functions, `summarize` is aware of the variable names and we can use them directly. So when inside the call to the `summarize` function we write `mean(height)`, the function is accessing the column with the name “height” and then computing the average of the resulting numeric vector. We can compute any other summary that operates on vectors and returns a single value. For example, we can add the median, minimum, and maximum heights like this:

```
heights %>%
  filter(sex == "Female") %>%
  summarize(median = median(height), minimum = min(height),
            maximum = max(height))
#>   median minimum maximum
#> 1     65      51      79
```

We can obtain these three values with just one line using the `quantile` function: for example, `quantile(x, c(0,0.5,1))` returns the min (0th percentile), median (50th percentile), and max (100th percentile) of the vector `x`. However, if we attempt to use a function like this that returns two or more values inside `summarize`:

```
heights %>%
  filter(sex == "Female") %>%
  summarize(range = quantile(height, c(0, 0.5, 1)))
```

we will receive an error: `Error: expecting result of length one, got : 2`. With the function `summarize`, we can only call functions that return a single value. In Section 4.12, we will learn how to deal with functions that return more than one value.

For another example of how we can use the `summarize` function, let’s compute the average murder rate for the United States. Remember our data table includes total murders and population size for each state and we have already used **dplyr** to add a murder rate column:


```
murders <- murders %>% mutate(rate = total/population*100000)
```

Remember that the US murder rate is **not** the average of the state murder rates:

```
summarize(murders, mean(rate))
#>   mean(rate)
#> 1      2.78
```

This is because in the computation above the small states are given the same weight as the large ones. The US murder rate is the total number of murders in the US divided by the total US population. So the correct computation is:

```
us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population) * 100000)
us_murder_rate
#>   rate
#> 1 3.03
```

This computation counts larger states proportionally to their size which results in a larger value.

4.7.2 pull

The `us_murder_rate` object defined above represents just one number. Yet we are storing it in a data frame:

```
class(us_murder_rate)
#> [1] "data.frame"
```

since, as most **dplyr** functions, `summarize` always returns a data frame.

This might be problematic if we want to use this result with functions that require a numeric value. Here we show a useful trick for accessing values stored in data when using pipes: when a data object is piped that object and its columns can be accessed using the `pull` function. To understand what we mean take a look at this line of code:

```
us_murder_rate %>% pull(rate)
#> [1] 3.03
```

This returns the value in the `rate` column of `us_murder_rate` making it equivalent to `us_murder_rate$rate`.

To get a number from the original data table with one line of code we can type:

```
us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population) * 100000) %>%
  pull(rate)

us_murder_rate
#> [1] 3.03
```

which is now a numeric:

```
class(us_murder_rate)
#> [1] "numeric"
```

4.7.3 Group then summarize with `group_by`

A common operation in data exploration is to first split data into groups and then compute summaries for each group. For example, we may want to compute the average and standard deviation for men's and women's heights separately. The `group_by` function helps us do this.

If we type this:

```
heights %>% group_by(sex)
#> # A tibble: 1,050 x 2
#> # Groups:   sex [2]
#>   sex    height
#>   <fct>   <dbl>
#> 1 Male      75
#> 2 Male      70
#> 3 Male      68
#> 4 Male      74
#> 5 Male      61
#> # ... with 1,045 more rows
```

The result does not look very different from `heights`, except we see `Groups: sex [2]` when we print the object. Although not immediately obvious from its appearance, this is now a special data frame called a *grouped data frame* and `dplyr` functions, in particular `summarize`, will behave differently when acting on this object. Conceptually, you can think of this table as many tables, with the same columns but not necessarily the same number of rows, stacked together in one object. When we summarize the data after grouping, this is what happens:

```
heights %>%
  group_by(sex) %>%
  summarize(average = mean(height), standard_deviation = sd(height))
#> # A tibble: 2 x 3
#>   sex    average standard_deviation
#>   <fct>   <dbl>             <dbl>
#> 1 Female    64.9              3.76
#> 2 Male     69.3              3.61
```

The `summarize` function applies the summarization to each group separately.

For another example, let's compute the median murder rate in the four regions of the country:

```
murders %>%
  group_by(region) %>%
  summarize(median_rate = median(rate))
```

```
#> # A tibble: 4 x 2
#>   region      median_rate
#>   <fct>          <dbl>
#> 1 Northeast      1.80
#> 2 South          3.40
#> 3 North Central  1.97
#> 4 West           1.29
```

4.8 Sorting data frames

When examining a dataset, it is often convenient to sort the table by the different columns. We know about the `order` and `sort` function, but for ordering entire tables, the **dplyr** function `arrange` is useful. For example, here we order the states by population size:

```
murders %>%
  arrange(population) %>%
  head()
#>       state abb      region population total  rate
#> 1    Wyoming WY        West    563626     5 0.887
#> 2 District of Columbia DC      South    601723    99 16.453
#> 3    Vermont VT      Northeast    625741     2 0.320
#> 4 North Dakota ND North Central    672591     4 0.595
#> 5    Alaska AK        West    710231    19 2.675
#> 6 South Dakota SD North Central    814180     8 0.983
```

With `arrange` we get to decide which column to sort by. To see the states by population, from smallest to largest, we arrange by `rate` instead:

```
murders %>%
  arrange(rate) %>%
  head()
#>       state abb      region population total  rate
#> 1    Vermont VT      Northeast    625741     2 0.320
#> 2 New Hampshire NH      Northeast    1316470     5 0.380
#> 3    Hawaii HI        West    1360301     7 0.515
#> 4 North Dakota ND North Central    672591     4 0.595
#> 5    Iowa IA North Central    3046355    21 0.689
#> 6    Idaho ID        West    1567582    12 0.766
```

Note that the default behavior is to order in ascending order. In **dplyr**, the function `desc` transforms a vector so that it is in descending order. To sort the table in descending order, we can type:

```
murders %>%
  arrange(desc(rate))
```

4.8.1 Nested sorting

If we are ordering by a column with ties, we can use a second column to break the tie. Similarly, a third column can be used to break ties between first and second and so on. Here we order by `region`, then within region we order by murder rate:

```
murders %>%
  arrange(region, rate) %>%
  head()
#>      state abb   region population total  rate
#> 1  Vermont  VT Northeast    625741     2 0.320
#> 2 New Hampshire NH Northeast    1316470     5 0.380
#> 3    Maine   ME Northeast    1328361    11 0.828
#> 4 Rhode Island RI Northeast    1052567    16 1.520
#> 5 Massachusetts MA Northeast    6547629   118 1.802
#> 6    New York  NY Northeast    19378102   517 2.668
```

4.8.2 The top *n*

In the code above, we have used the function `head` to avoid having the page fill up with the entire dataset. If we want to see a larger proportion, we can use the `top_n` function. This function takes a data frame as its first argument, the number of rows to show in the second, and the variable to filter by in the third. Here is an example of how to see the top 5 rows:

```
murders %>% top_n(5, rate)
#>      state abb   region population total  rate
#> 1 District of Columbia DC      South    601723     99 16.45
#> 2    Louisiana  LA      South    4533372    351  7.74
#> 3    Maryland  MD      South    5773552    293  5.07
#> 4    Missouri  MO North Central    5988927    321  5.36
#> 5  South Carolina SC      South    4625364    207  4.48
```

Note that rows are not sorted by `rate`, only filtered. If we want to sort, we need to use `arrange`. Note that if the third argument is left blank, `top_n`, filters by the last column.

4.9 Exercises

For these exercises, we will be using the data from the survey collected by the United States National Center for Health Statistics (NCHS). This center has conducted a series of health and nutrition surveys since the 1960's. Starting in 1999, about 5,000 individuals of all ages have been interviewed every year and they complete the health examination component of the survey. Part of the data is made available via the **NHANES** package. Once you install the **NHANES** package, you can load the data like this:

```
library(NHANES)
data(NHANES)
```

The **NHANES** data has many missing values. Remember that the main summarization function in R will return **NA** if any of the entries of the input vector is an **NA**. Here is an example:

```
library(dslabs)
data(na_example)
mean(na_example)
#> [1] NA
sd(na_example)
#> [1] NA
```

To ignore the NAs we can use the **na.rm** argument:

```
mean(na_example, na.rm = TRUE)
#> [1] 2.3
sd(na_example, na.rm = TRUE)
#> [1] 1.22
```

Let's now explore the NHANES data.

1. We will provide some basic facts about blood pressure. First let's select a group to set the standard. We will use 20-to-29-year-old females. **AgeDecade** is a categorical variable with these ages. Note that the category is coded like " 20-29", with a space in front! What is the average and standard deviation of systolic blood pressure as saved in the **BPSysAve** variable? Save it to a variable called **ref**.

Hint: Use **filter** and **summarize** and use the **na.rm = TRUE** argument when computing the average and standard deviation. You can also filter the NA values using **filter**.

2. Using a pipe, assign the average to a numeric variable **ref_avg**. Hint: Use the code similar to above and then **pull**.

3. Now report the min and max values for the same group.

4. Compute the average and standard deviation for females, but for each age group separately rather than a selected decade as in question 1. Note that the age groups are defined by **AgeDecade**. Hint: rather than filtering by age and gender, filter by **Gender** and then use **group_by**.

5. Repeat exercise 4 for males.

6. We can actually combine both summaries for exercises 4 and 5 into one line of code. This is because **group_by** permits us to group by more than one variable. Obtain one big summary table using **group_by(AgeDecade, Gender)**.

7. For males between the ages of 40-49, compare systolic blood pressure across race as reported in the **Race1** variable. Order the resulting table from lowest to highest average systolic blood pressure.

4.10 Tibbles

Tidy data must be stored in data frames. We introduced the data frame in Section 2.4.1 and have been using the `murders` data frame throughout the book. In Section 4.7.3 we introduced the `group_by` function, which permits stratifying data before computing summary statistics. But where is the group information stored in the data frame?

```
murders %>% group_by(region)
#> # A tibble: 51 x 6
#> # Groups:   region [4]
#>   state     abb region population total   rate
#>   <chr>    <chr> <fct>      <dbl> <dbl> <dbl>
#> 1 Alabama AL    South    4779736  135  2.82
#> 2 Alaska  AK    West     710231  19  2.68
#> 3 Arizona AZ    West    6392017  232  3.63
#> 4 Arkansas AR    South   2915918  93  3.19
#> 5 California CA    West   37253956 1257  3.37
#> # ... with 46 more rows
```

Notice that there are no columns with this information. But, if you look closely at the output above, you see the line `A tibble` followed by dimensions. We can learn the class of the returned object using:

```
murders %>% group_by(region) %>% class()
#> [1] "grouped_df" "tbl_df"      "tbl"         "data.frame"
```

The `tbl`, pronounced tibble, is a special kind of data frame. The functions `group_by` and `summarize` always return this type of data frame. The `group_by` function returns a special kind of `tbl`, the `grouped_df`. We will say more about these later. For consistency, the `dplyr` manipulation verbs (`select`, `filter`, `mutate`, and `arrange`) preserve the class of the input: if they receive a regular data frame they return a regular data frame, while if they receive a tibble they return a tibble. But tibbles are the preferred format in the tidyverse and as a result tidyverse functions that produce a data frame from scratch return a tibble. For example, in Chapter 5 we will see that tidyverse functions used to import data create tibbles.

Tibbles are very similar to data frames. In fact, you can think of them as a modern version of data frames. Nonetheless there are three important differences which we describe in the next.

4.10.1 Tibbles display better

The print method for tibbles is more readable than that of a data frame. To see this, compare the outputs of typing `murders` and the output of `murders` if we convert it to a tibble. We can do this using `as_tibble(murders)`. If using RStudio, output for a tibble adjusts to your window size. To see this, change the width of your R console and notice how more/less columns are shown.

4.10.2 Subsets of tibbles are tibbles

If you subset the columns of a data frame, you may get back an object that is not a data frame, such as a vector or scalar. For example:

```
class(murders[,4])
#> [1] "numeric"
```

is not a data frame. With tibbles this does not happen:

```
class(as_tibble(murders)[,4])
#> [1] "tbl_df"      "tbl"          "data.frame"
```

This is useful in the tidyverse since functions require data frames as input.

With tibbles, if you want to access the vector that defines a column, and not get back a data frame, you need to use the accessor `$`:

```
class(as_tibble(murders)$population)
#> [1] "numeric"
```

A related feature is that tibbles will give you a warning if you try to access a column that does not exist. If we accidentally write `Population` instead of `population` this:

```
murders$Population
#> NULL
```

returns a `NULL` with no warning, which can make it harder to debug. In contrast, if we try this with a tibble we get an informative warning:

```
as_tibble(murders)$Population
#> Warning: Unknown or uninitialised column: 'Population'.
#> NULL
```

4.10.3 Tibbles can have complex entries

While data frame columns need to be vectors of numbers, strings, or logical values, tibbles can have more complex objects, such as lists or functions. Also, we can create tibbles with functions:

```
tibble(id = c(1, 2, 3), func = c(mean, median, sd))
#> # A tibble: 3 x 2
#>       id func
#>   <dbl> <list>
#> 1     1  1 <fn>
#> 2     2  2 <fn>
#> 3     3  3 <fn>
```

4.10.4 Tibbles can be grouped

The function `group_by` returns a special kind of tibble: a grouped tibble. This class stores information that lets you know which rows are in which groups. The tidyverse functions, in particular the `summarize` function, are aware of the group information.

4.10.5 Create a tibble using `tibble` instead of `data.frame`

It is sometimes useful for us to create our own data frames. To create a data frame in the tibble format, you can do this by using the `tibble` function.

```
grades <- tibble(names = c("John", "Juan", "Jean", "Yao"),
                 exam_1 = c(95, 80, 90, 85),
                 exam_2 = c(90, 85, 85, 90))
```

Note that base R (without packages loaded) has a function with a very similar name, `data.frame`, that can be used to create a regular data frame rather than a tibble. One other important difference is that by default `data.frame` coerces characters into factors without providing a warning or message:

```
grades <- data.frame(names = c("John", "Juan", "Jean", "Yao"),
                    exam_1 = c(95, 80, 90, 85),
                    exam_2 = c(90, 85, 85, 90))

class(grades$names)
#> [1] "factor"
```

To avoid this, we use the rather cumbersome argument `stringsAsFactors`:

```
grades <- data.frame(names = c("John", "Juan", "Jean", "Yao"),
                    exam_1 = c(95, 80, 90, 85),
                    exam_2 = c(90, 85, 85, 90),
                    stringsAsFactors = FALSE)

class(grades$names)
#> [1] "character"
```

To convert a regular data frame to a tibble, you can use the `as_tibble` function.

```
as_tibble(grades) %>% class()
#> [1] "tbl_df"      "tbl"        "data.frame"
```

4.11 The dot operator

One of the advantages of using the pipe `%>%` is that we do not have to keep naming new objects as we manipulate the data frame. As a quick reminder, if we want to compute the median murder rate for states in the southern states, instead of typing:


```
tab_1 <- filter(murders, region == "South")
tab_2 <- mutate(tab_1, rate = total / population * 10^5)
rates <- tab_2$rate
median(rates)
#> [1] 3.4
```

We can avoid defining any new intermediate objects by instead typing:

```
filter(murders, region == "South") %>%
  mutate(rate = total / population * 10^5) %>%
  summarize(median = median(rate)) %>%
  pull(median)
#> [1] 3.4
```

We can do this because each of these functions takes a data frame as the first argument. But what if we want to access a component of the data frame. For example, what if the `pull` function was not available and we wanted to access `tab_2$rate`? What data frame name would we use? The answer is the dot operator.

For example to access the rate vector without the `pull` function we could use

```
rates <- filter(murders, region == "South") %>%
  mutate(rate = total / population * 10^5) %>%
  .$rate
median(rates)
#> [1] 3.4
```

In the next section, we will see other instances in which using the `.` is useful.

4.12 do

The tidyverse functions know how to interpret grouped tibbles. Furthermore, to facilitate stringing commands through the pipe `%>%`, tidyverse functions consistently return data frames, since this assures that the output of a function is accepted as the input of another. But most R functions do not recognize grouped tibbles nor do they return data frames. The `quantile` function is an example we described in Section 4.7.1. The `do` function serves as a bridge between R functions such as `quantile` and the tidyverse. The `do` function understands grouped tibbles and always returns a data frame.

In Section 4.7.1, we noted that if we attempt to use `quantile` to obtain the min, median and max in one call, we will receive an error: `Error: expecting result of length one, got : 2.`

```
data(heights)
heights %>%
  filter(sex == "Female") %>%
  summarize(range = quantile(height, c(0, 0.5, 1)))
```

We can use the `do` function to fix this.

First we have to write a function that fits into the tidyverse approach: that is, it receives a data frame and returns a data frame.

```
my_summary <- function(dat){
  x <- quantile(dat$height, c(0, 0.5, 1))
  tibble(min = x[1], median = x[2], max = x[3])
}
```

We can now apply the function to the heights dataset to obtain the summaries:

```
heights %>%
  group_by(sex) %>%
  my_summary
#> # A tibble: 1 x 3
#>   min median max
#>   <dbl>   <dbl> <dbl>
#> 1    50   68.5  82.7
```

But this is not what we want. We want a summary for each sex and the code returned just one summary. This is because `my_summary` is not part of the tidyverse and does not know how to handle grouped tibbles. `do` makes this connection:

```
heights %>%
  group_by(sex) %>%
  do(my_summary())
#> # A tibble: 2 x 4
#> # Groups:   sex [2]
#>   sex      min median max
#>   <fct>   <dbl>   <dbl> <dbl>
#> 1 Female    51   65.0   79
#> 2 Male     50    69   82.7
```

Note that here we need to use the dot operator. The tibble created by `group_by` is piped to `do`. Within the call to `do`, the name of this tibble is `.` and we want to send it to `my_summary`. If you do not use the dot, then `my_summary` has ___no argument and returns an error telling us that argument "dat" is missing. You can see the error by typing:

```
heights %>%
  group_by(sex) %>%
  do(my_summary())
```

If you do not use the parenthesis, then the function is not executed and instead `do` tries to return the function. This gives an error because `do` must always return a data frame. You can see the error by typing:

```
heights %>%
  group_by(sex) %>%
  do(my_summary)
```

4.13 The purrr package

In Section 3.5 we learned about the `sapply` function, which permitted us to apply the same function to each element of a vector. We constructed a function and used `sapply` to compute the sum of the first `n` integers for several values of `n` like this:

```
compute_s_n <- function(n){  
  x <- 1:n  
  sum(x)  
}  
n <- 1:25  
s_n <- sapply(n, compute_s_n)
```

This type of operation, applying the same function or procedure to elements of an object, is quite common in data analysis. The **purrr** package includes functions similar to `sapply` but that better interact with other tidyverse functions. The main advantage is that we can better control the output type of functions. In contrast, `sapply` can return several different object types; for example, we might expect a numeric result from a line of code, but `sapply` might convert our result to character under some circumstances. **purrr** functions will never do this: they will return objects of a specified type or return an error if this is not possible.

The first **purrr** function we will learn is `map`, which works very similar to `sapply` but always, without exception, returns a list:

```
library(purrr)  
s_n <- map(n, compute_s_n)  
class(s_n)  
#> [1] "list"
```

If we want a numeric vector, we can instead use `map_dbl` which always returns a vector of numeric values.

```
s_n <- map_dbl(n, compute_s_n)  
class(s_n)  
#> [1] "numeric"
```

This produces the same results as the `sapply` call shown above.

A particularly useful **purrr** function for interacting with the rest of the tidyverse is `map_df`, which always returns a tibble data frame. However, the function being called needs to return a vector or a list with names. For this reason, the following code would result in a `Argument 1 must have names` error:

```
s_n <- map_df(n, compute_s_n)
```

We need to change the function to make this work:

```
compute_s_n <- function(n){
  x <- 1:n
  tibble(sum = sum(x))
}
s_n <- map_df(n, compute_s_n)
```

The **purrr** package provides much more functionality not covered here. For more details you can consult [this online resource](#).

4.14 Tidyverse conditionals

A typical data analysis will often involve one or more conditional operations. In Section 3.1 we described the **ifelse** function, which we will use extensively in this book. In this section we present two **dplyr** functions that provide further functionality for performing conditional operations.

4.14.1 case_when

The **case_when** function is useful for vectorizing conditional statements. It is similar to **ifelse** but can output any number of values, as opposed to just **TRUE** or **FALSE**. Here is an example splitting numbers into negative, positive, and 0:

```
x <- c(-2, -1, 0, 1, 2)
case_when(x < 0 ~ "Negative", x > 0 ~ "Positive", TRUE ~ "Zero")
#> [1] "Negative" "Negative" "Zero"      "Positive" "Positive"
```

A common use for this function is to define categorical variables based on existing variables. For example, suppose we want to compare the murder rates in three groups of states: *New England*, *West Coast*, *South*, and *other*. For each state, we need to ask if it is in New England, if it is not we ask if it is in the West Coast, if not we ask if it is in the South, and if not we assign *other*. Here is how we use **case_when** to do this:

```
murders %>%
  mutate(group = case_when(
    abb %in% c("ME", "NH", "VT", "MA", "RI", "CT") ~ "New England",
    abb %in% c("WA", "OR", "CA") ~ "West Coast",
    region == "South" ~ "South",
    TRUE ~ "Other")) %>%
  group_by(group) %>%
  summarize(rate = sum(total) / sum(population) * 10^5)
#> # A tibble: 4 x 2
#>   group      rate
#>   <chr>    <dbl>
#> 1 New England 1.72
#> 2 Other      2.71
```

```
#> 3 South      3.63
#> 4 West Coast  2.90
```

4.14.2 between

A common operation in data analysis is to determine if a value falls inside an interval. We can check this using conditionals. For example to check if the elements of a vector `x` are between `a` and `b` we can type

```
x >= a & x <= b
```

However, this can become cumbersome, especially within the tidyverse approach. The `between` function performs the same operation.

```
between(x, a, b)
```

4.15 Exercises

1. Load the `murders` dataset. Which of the following is true?
 - a. `murders` is in tidy format and is stored in a tibble.
 - b. `murders` is in tidy format and is stored in a data frame.
 - c. `murders` is not in tidy format and is stored in a tibble.
 - d. `murders` is not in tidy format and is stored in a data frame.
2. Use `as_tibble` to convert the `murders` data table into a tibble and save it in an object called `murders_tibble`.
3. Use the `group_by` function to convert `murders` into a tibble that is grouped by region.
4. Write tidyverse code that is equivalent to this code:

```
exp(mean(log(murders$population)))
```

Write it using the pipe so that each function is called without arguments. Use the dot operator to access the population. Hint: The code should start with `murders %>%`.

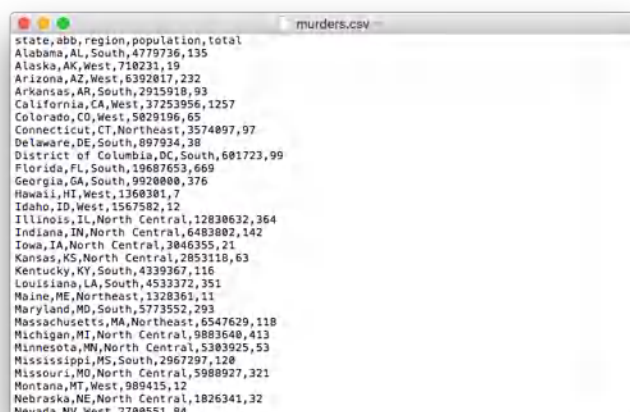
5. Use the `map_df` to create a data frame with three columns named `n`, `s_n`, and `s_n_2`. The first column should contain the numbers 1 through 100. The second and third columns should each contain the sum of 1 through `n` with `n` the row number.

5

Importing data

We have been using data sets already stored as R objects. A data scientist will rarely have such luck and will have to import data into R from either a file, a database, or other sources. Currently, one of the most common ways of storing and sharing data for analysis is through electronic spreadsheets. A spreadsheet stores data in rows and columns. It is basically a file version of a data frame. When saving such a table to a computer file, one needs a way to define when a new row or column ends and the other begins. This in turn defines the cells in which single values are stored.

When creating spreadsheets with text files, like the ones created with a simple text editor, a new row is defined with return and columns are separated with some predefined special character. The most common characters are comma (,), semicolon (;), space (), and tab (a preset number of spaces or \t). Here is an example of what a comma separated file looks like if we open it with a basic text editor:



```
state,abb,region,population,total
Alabama,AL,South,4779736,135
Alaska,AK,West,718231,19
Arizona,AZ,West,6392817,232
Arkansas,AR,South,2915918,93
California,CA,West,37253956,1257
Colorado,CO,West,5829196,65
Connecticut,CT,Northeast,3574897,97
Delaware,DE,South,897924,38
District of Columbia,DC,South,681723,99
Florida,FL,South,19687653,669
Georgia,GA,South,9928888,376
Hawaii,HI,West,1368381,7
Idaho,ID,West,1567582,12
Illinois,IL,North Central,12838632,364
Indiana,IN,North Central,6483882,142
Iowa,IA,North Central,3046355,21
Kansas,KS,North Central,2853118,63
Kentucky,KY,South,4399267,116
Louisiana,LA,South,4533372,351
Maine,ME,Northeast,1328361,11
Maryland,MD,South,5773552,293
Massachusetts,MA,Northeast,6547629,118
Michigan,MI,North Central,9883648,413
Minnesota,MN,North Central,5383925,53
Mississippi,MS,South,2967297,128
Missouri,MO,North Central,5988927,321
Montana,MT,West,989415,12
Nebraska,NE,North Central,1826341,32
Nevada,NV,West,2708551,84
```

The first row contains column names rather than data. We call this a *header*, and when we read-in data from a spreadsheet it is important to know if the file has a header or not. Most reading functions assume there is a header. To know if the file has a header, it helps to look at the file before trying to read it. This can be done with a text editor or with RStudio. In RStudio, we can do this by either opening the file in the editor or navigating to the file location, double clicking on the file, and hitting *View File*.

However, not all spreadsheet files are in a text format. Google Sheets, which are rendered on a browser, are an example. Another example is the proprietary format used by Microsoft Excel. These can't be viewed with a text editor. Despite this, due to the widespread use of Microsoft Excel software, this format is widely used.

We start this chapter by describing the difference between text (ASCII), Unicode, and binary files and how this affects how we import them. We then explain the concepts of file paths and working directories, which are essential to understand how to import data effectively. We then introduce the **readr** and **readxl** package and the functions that are available to import spreadsheets into R. Finally, we provide some recommendations on how to store and organize data in files. More complex challenges such as extracting data from web pages or PDF documents are left for the Data Wrangling part of the book.

5.1 Paths and the working directory

The first step when importing data from a spreadsheet is to locate the file containing the data. Although we do not recommend it, you can use an approach similar to what you do to open files in Microsoft Excel by clicking on the RStudio “File” menu, clicking “Import Dataset”, then clicking through folders until you find the file. We want to be able to write code rather than use the point-and-click approach. The keys and concepts we need to learn to do this are described in detail in the Productivity Tools part of this book. Here we provide an overview of the very basics.

The main challenge in this first step is that we need to let the R functions doing the importing know where to look for the file containing the data. The simplest way to do this is to have a copy of the file in the folder in which the importing functions look by default. Once we do this, all we have to supply to the importing function is the filename.

A spreadsheet containing the US murders data is included as part of the **dslabs** package. Finding this file is not straightforward, but the following lines of code copy the file to the folder in which R looks in by default. We explain how these lines work below.

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders.csv")
```

This code does not read the data into R, it just copies a file. But once the file is copied, we can import the data with a simple line of code. Here we use the **read_csv** function from the **readr** package, which is part of the tidyverse.

```
library(tidyverse)
dat <- read_csv(filename)
```

The data is imported and stored in **dat**. The rest of this section defines some important concepts and provides an overview of how we write code that tells R how to find the files we want to import. Chapter 36 provides more details on this topic.

5.1.1 The filesystem

You can think of your computer’s filesystem as a series of nested folders, each containing other folders and files. Data scientists refer to folders as *directories*. We refer to the folder

that contains all other folders as the *root directory*. We refer to the directory in which we are currently located as the *working directory*. The working directory therefore changes as you move through folders: think of it as your current location.

5.1.2 Relative and full paths

The *path* of a file is a list of directory names that can be thought of as instructions on what folders to click on, and in what order, to find the file. If these instructions are for finding the file from the root directory we refer to it as the *full path*. If the instructions are for finding the file starting in the working directory we refer to it as a *relative path*. Section 36.3 provides more details on this topic.

To see an example of a full path on your system type the following:

```
system.file(package = "dslabs")
```

The strings separated by slashes are the directory names. The first slash represents the root directory and we know this is a full path because it starts with a slash. If the first directory name appears without a slash in front, then the path is assumed to be relative. We can use the function `list.files` to see examples of relative paths.

```
dir <- system.file(package = "dslabs")
list.files(path = dir)
#> [1] "data"      "DESCRIPTION" "extdata"    "help"
#> [5] "html"     "INDEX"      "Meta"      "NAMESPACE"
#> [9] "R"        "script"
```

These relative paths give us the location of the files or directories if we start in the directory with the full path. For example, the full path to the `help` directory in the example above is `/Library/Frameworks/R.framework/Versions/3.5/Resources/library/dslabs/help`.

Note: You will probably not make much use of the `system.file` function in your day-to-day data analysis work. We introduce it in this section because it facilitates the sharing of spreadsheets by including them in the **dslabs** package. You will rarely have the luxury of data being included in packages you already have installed. However, you will frequently need to navigate full and relative paths and import spreadsheet formatted data.

5.1.3 The working directory

We highly recommend only writing relative paths in your code. The reason is that full paths are unique to your computer and you want your code to be portable. You can get the full path of your working directory without writing out explicitly by using the `getwd` function.

```
wd <- getwd()
```

If you need to change your working directory, you can use the function `setwd` or you can change it through RStudio by clicking on “Session”.

5.1.4 Generating path names

Another example of obtaining a full path without writing out explicitly was given above when we created the object `fullpath` like this:

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
```

The function `system.file` provides the full path of the folder containing all the files and directories relevant to the package specified by the `package` argument. By exploring the directories in `dir` we find that the `extdata` contains the file we want:

```
dir <- system.file(package = "dslabs")
filename %in% list.files(file.path(dir, "extdata"))
#> [1] TRUE
```

The `system.file` function permits us to provide a subdirectory as a first argument, so we can obtain the fullpath of the `extdata` directory like this:

```
dir <- system.file("extdata", package = "dslabs")
```

The function `file.path` is used to combine directory names to produce the full path of the file we want to import.

```
fullpath <- file.path(dir, filename)
```

5.1.5 Copying files using paths

The final line of code we used to copy the file into our home directory used the function `file.copy`. This function takes two arguments: the file to copy and the name to give it in the new directory.

```
file.copy(fullpath, "murders.csv")
#> [1] TRUE
```

If a file is copied successfully, the `file.copy` function returns `TRUE`. Note that we are giving the file the same name, `murders.csv`, but we could have named it anything. Also note that by not starting the string with a slash, R assumes this is a relative path and copies the file to the working directory.

You should be able to see the file in your working directory and can check by using:

```
list.files()
```

5.2 The readr and readxl packages

In this section we introduce the main tidyverse data importing functions. We will use the `murders.csv` file provided by the **dslabs** package as an example. To simplify the illustration we will copy the file to our working directory using the following code:

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders.csv")
```

5.2.1 readr

The **readr** library includes functions for reading data stored in text file spreadsheets into R. **readr** is part of the **tidyverse** package, or you can load it directly:

```
library(readr)
```

The following functions are available to read-in spreadsheets:

Function	Format	Typical suffix
<code>read_table</code>	white space separated values	txt
<code>read_csv</code>	comma separated values	csv
<code>read_csv2</code>	semicolon separated values	csv
<code>read_tsv</code>	tab delimited separated values	tsv
<code>read_delim</code>	general text file format, must define delimiter	txt

Although the suffix usually tells us what type of file it is, there is no guarantee that these always match. We can open the file to take a look or use the function `read_lines` to look at a few lines:

```
read_lines("murders.csv", n_max = 3)
#> [1] "state,abb,region,population,total"
#> [2] "Alabama,AL,South,4779736,135"
#> [3] "Alaska,AK,West,710231,19"
```

This also shows that there is a header. Now we are ready to read-in the data into R. From the `.csv` suffix and the peek at the file, we know to use `read_csv`:

```
dat <- read_csv(filename)
#> Parsed with column specification:
#> cols(
#>   state = col_character(),
#>   abb = col_character(),
```

```
#>   region = col_character(),
#>   population = col_double(),
#>   total = col_double()
#> )
```

Note that we receive a message letting us know what data types were used for each column. Also note that `dat` is a **tibble**, not just a data frame. This is because `read_csv` is a **tidyverse** parser. We can confirm that the data has in fact been read-in with:

```
View(dat)
```

Finally, note that we can also use the full path for the file:

```
dat <- read_csv(fullpath)
```

5.2.2 readxl

You can load the **readxl** package using

```
library(readxl)
```

The package provides functions to read-in Microsoft Excel formats:

Function	Format	Typical suffix
<code>read_excel</code>	auto detect the format	xls, xlsx
<code>read_xls</code>	original format	xls
<code>read_xlsx</code>	new format	xlsx

The Microsoft Excel formats permit you to have more than one spreadsheet in one file. These are referred to as *sheets*. The functions listed above read the first sheet by default, but we can also read the others. The `excel_sheets` function gives us the names of all the sheets in an Excel file. These names can then be passed to the `sheet` argument in the three functions above to read sheets other than the first.

5.3 Exercises

1. Use the `read_csv` function to read each of the files that the following code saves in the `files` object:

```
path <- system.file("extdata", package = "dslabs")
files <- list.files(path)
files
```

2. Note that the last one, the `olive` file, gives us a warning. This is because the first line of the file is missing the header for the first column.

Read the help file for `read_csv` to figure out how to read in the file without reading this header. If you skip the header, you should not get this warning. Save the result to an object called `dat`.

3. A problem with the previous approach is that we don't know what the columns represent. Type:

```
names(dat)
```

to see that the names are not informative.

Use the `readLines` function to read in just the first line (we later learn how to extract values from the output).

5.4 Downloading files

Another common place for data to reside is on the internet. When these data are in files, we can download them and then import them or even read them directly from the web. For example, we note that because our `dslabs` package is on GitHub, the file we downloaded with the package has a url:

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/
extdata/murders.csv"
```

The `read_csv` file can read these files directly:

```
dat <- read_csv(url)
```

If you want to have a local copy of the file, you can use the `download.file` function:

```
download.file(url, "murders.csv")
```

This will download the file and save it on your system with the name `murders.csv`. You can use any name here, not necessarily `murders.csv`. Note that when using `download.file` you should be careful as it will overwrite existing files without warning.

Two functions that are sometimes useful when downloading data from the internet are `tempdir` and `tempfile`. The first creates a directory with a random name that is very likely to be unique. Similarly, `tempfile` creates a character string, not a file, that is likely to be a unique filename. So you can run a command like this which erases the temporary file once it imports the data:

```
tmp_filename <- tempfile()
download.file(url, tmp_filename)
dat <- read_csv(tmp_filename)
file.remove(tmp_filename)
```

5.5 R-base importing functions

R-base also provides import functions. These have similar names to those in the **tidyverse**, for example `read.table`, `read.csv` and `read.delim`. However, there are a couple of important differences. To show this we read-in the data with an R-base function:

```
dat2 <- read.csv(filename)
```

An important difference is that the characters are converted to factors:

```
class(dat2$abb)
#> [1] "factor"
class(dat2$region)
#> [1] "factor"
```

This can be avoided by setting the argument `stringsAsFactors` to `FALSE`.

```
dat <- read.csv("murders.csv", stringsAsFactors = FALSE)
class(dat$state)
#> [1] "character"
```

In our experience this can be a cause for confusion since a variable that was saved as characters in file is converted to factors regardless of what the variable represents. In fact, we **highly** recommend setting `stringsAsFactors=FALSE` to be your default approach when using the R-base parsers. You can easily convert the desired columns to factors after importing data.

5.5.1 scan

When reading in spreadsheets many things can go wrong. The file might have a multiline header, be missing cells, or it might use an unexpected encoding¹. We recommend you read this post about common issues found here: <https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/>.

With experience you will learn how to deal with different challenges. Carefully reading the help files for the functions discussed here will be useful. Two other functions that are helpful are `scan`. With `scan` you can read-in each cell of a file. Here is an example:

```
path <- system.file("extdata", package = "dslabs")
filename <- "murders.csv"
x <- scan(file.path(path, filename), sep=",", what = "c")
x[1:10]
#> [1] "state"      "abb"        "region"     "population" "total"
#> [6] "Alabama"   "AL"         "South"      "4779736"    "135"
```

Note that the tidyverse provides `read_lines`, a similarly useful function.

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

5.6 Text versus binary files

For data science purposes, files can generally be classified into two categories: text files (also known as ASCII files) and binary files. You have already worked with text files. All your R scripts are text files and so are the R markdown files used to create this book. The csv tables you have read are also text files. One big advantage of these files is that we can easily “look” at them without having to purchase any kind of special software or follow complicated instructions. Any text editor can be used to examine a text file, including freely available editors such as RStudio, Notepad, textEdit, vi, emacs, nano, and pico. To see this, try opening a csv file using the “Open file” RStudio tool. You should be able to see the content right on your editor. However, if you try to open, say, an Excel xls file, jpg or png file, you will not be able to see anything immediately useful. These are binary files. Excel files are actually compressed folders with several text files inside. But the main distinction here is that text files can be easily examined.

Although R includes tools for reading widely used binary files, such as xls files, in general you will want to find data sets stored in text files. Similarly, when sharing data you want to make it available as text files as long as storage is not an issue (binary files are much more efficient at saving space on your disk). In general, plain-text formats make it easier to share data since commercial software is not required for working with the data.

Extracting data from a spreadsheet stored as a text file is perhaps the easiest way to bring data from a file to an R session. Unfortunately, spreadsheets are not always available and the fact that you can look at text files does not necessarily imply that extracting data from them will be straightforward. In the Data Wrangling part of the book we learn to extract data from more complex text files such as html files.

5.7 Unicode versus ASCII

A pitfall in data science is assuming a file is an ASCII text file when, in fact, it is something else that can look a lot like an ASCII text file: a Unicode text file.

To understand the difference between these, remember that everything on a computer needs to eventually be converted to 0s and 1s. ASCII is an *encoding* that maps characters to numbers. ASCII uses 7 bits (0s and 1s) which results in $2^7 = 128$ unique items, enough to encode all the characters on an English language keyboard. However, other languages use characters not included in this encoding. For example, the é in México is not encoded by ASCII. For this reason, a new encoding, using more than 7 bits, was defined: Unicode. When using Unicode, one can choose between 8, 16, and 32 bits abbreviated UTF-8, UTF-16, and UTF-32 respectively. RStudio actually defaults to UTF-8 encoding.

Although we do not go into the details of how to deal with the different encodings here, it is important that you know these different encodings exist so that you can better diagnose a problem if you encounter it. One way problems manifest themselves is when you see “weird looking” characters you were not expecting. This StackOverflow discussion is an example: <https://stackoverflow.com/questions/18789330/r-on-windows-character-encoding-hell>.

5.8 Organizing data with spreadsheets

Although there are R packages designed to read this format, if you are choosing a file format to save your own data, you generally want to avoid Microsoft Excel. We recommend Google Sheets as a free software tool for organizing data. We provide more recommendations in the section Data Organization with Spreadsheets. This book focuses on data analysis. Yet often a data scientist needs to collect data or work with others collecting data. Filling out a spreadsheet by hand is a practice we highly discourage and instead recommend that the process be automatized as much as possible. But sometimes you just have to do it. In this section, we provide recommendations on how to store data in a spreadsheet. We summarize a paper by Karl Broman and Kara Woo². Below are their general recommendations. Please read the paper for important details.

- **Be Consistent** - Before you commence entering data, have a plan. Once you have a plan, be consistent and stick to it.
- **Choose Good Names for Things** - You want the names you pick for objects, files, and directories to be memorable, easy to spell, and descriptive. This is actually a hard balance to achieve and it does require time and thought. One important rule to follow is **do not use spaces**, use underscores `_` or dashes instead `-`. Also, avoid symbols; stick to letters and numbers.
- **Write Dates as YYYY-MM-DD** - To avoid confusion, we strongly recommend using this global ISO 8601 standard.
- **No Empty Cells** - Fill in all cells and use some common code for missing data.
- **Put Just One Thing in a Cell** - It is better to add columns to store the extra information rather than having more than one piece of information in one cell.
- **Make It a Rectangle** - The spreadsheet should be a rectangle.
- **Create a Data Dictionary** - If you need to explain things, such as what the columns are or what the labels used for categorical variables are, do this in a separate file.
- **No Calculations in the Raw Data Files** - Excel permits you to perform calculations. Do not make this part of your spreadsheet. Code for calculations should be in a script.
- **Do Not Use Font Color or Highlighting as Data** - Most import functions are not able to import this information. Encode this information as a variable instead.
- **Make Backups** - Make regular backups of your data.
- **Use Data Validation to Avoid Errors** - Leverage the tools in your spreadsheet software so that the process is as error-free and repetitive-stress-injury-free as possible.
- **Save the Data as Text Files** - Save files for sharing in comma or tab delimited format.

5.9 Exercises

1. Pick a measurement you can take on a regular basis. For example, your daily weight or how long it takes you to run 5 miles. Keep a spreadsheet that includes the date, the hour, the measurement, and any other informative variable you think is worth keeping. Do this for 2 weeks. Then make a plot.

²<https://www.tandfonline.com/doi/abs/10.1080/00031305.2017.1375989>

Part II

Data Visualization



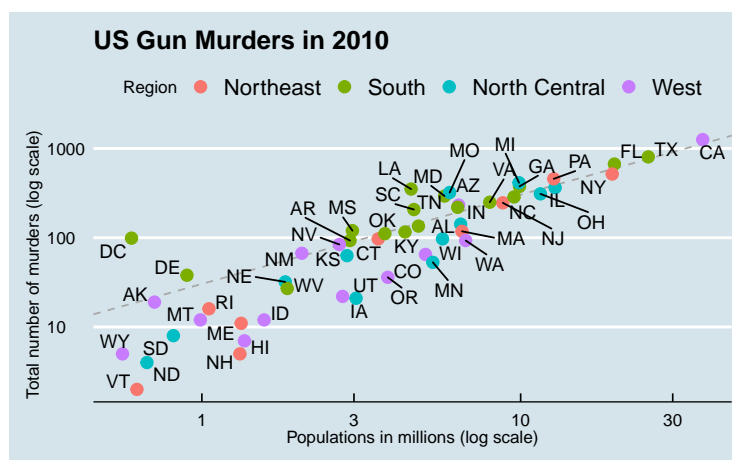
6

Introduction to data visualization

Looking at the numbers and character strings that define a dataset is rarely useful. To convince yourself, print and stare at the US murders data table:

```
library(dslabs)
data(murders)
head(murders)
#>      state abb region population total
#> 1  Alabama AL  South   4779736   135
#> 2  Alaska AK   West    710231    19
#> 3  Arizona AZ   West   6392017   232
#> 4  Arkansas AR  South   2915918    93
#> 5 California CA  West  37253956  1257
#> 6  Colorado CO  West   5029196    65
```

What do you learn from staring at this table? How quickly can you determine which states have the largest populations? Which states have the smallest? How large is a typical state? Is there a relationship between population size and total murders? How do murder rates vary across regions of the country? For most human brains, it is quite difficult to extract this information just by looking at the numbers. In contrast, the answer to all the questions above are readily available from examining this plot:

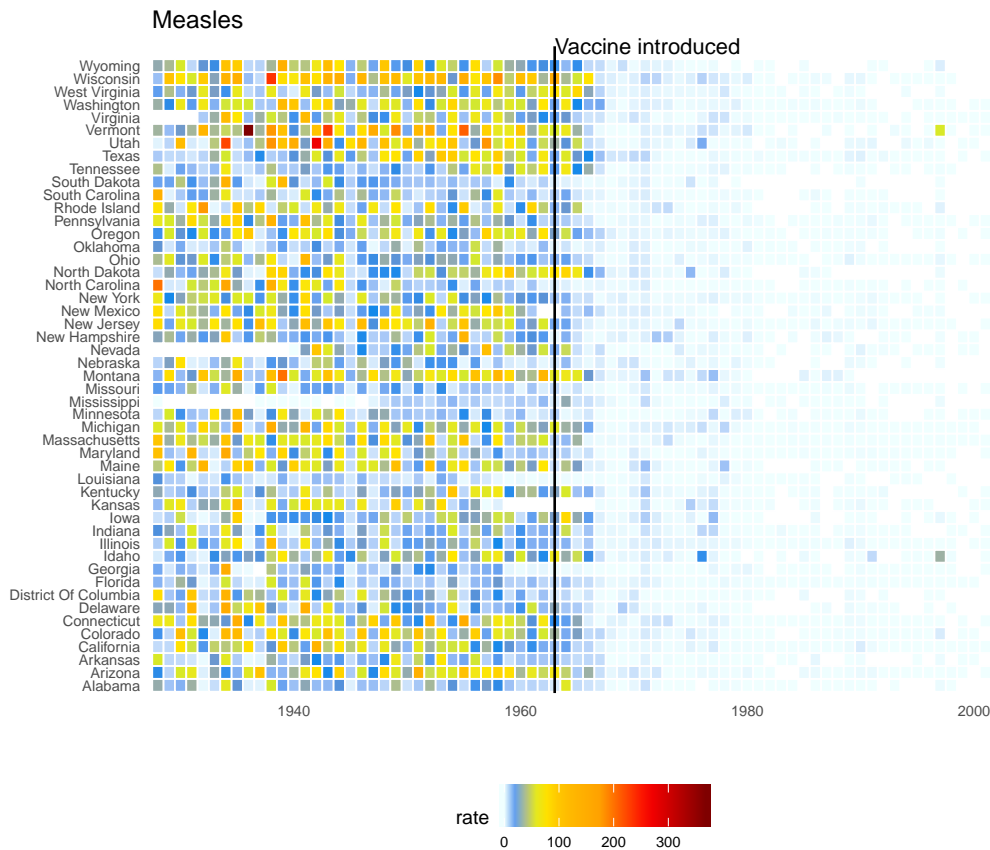


We are reminded of the saying “a picture is worth a thousand words”. Data visualization provides a powerful way to communicate a data-driven finding. In some cases, the visualization is so convincing that no follow-up analysis is required.

The growing availability of informative datasets and software tools has led to increased reliance on data visualizations across many industries, academia, and government. A salient

example is news organizations, which are increasingly embracing *data journalism* and including effective *infographics* as part of their reporting.

A particularly effective example is a Wall Street Journal article¹ showing data related to the impact of vaccines on battling infectious diseases. One of the graphs shows measles cases by US state through the years with a vertical line demonstrating when the vaccine was introduced.



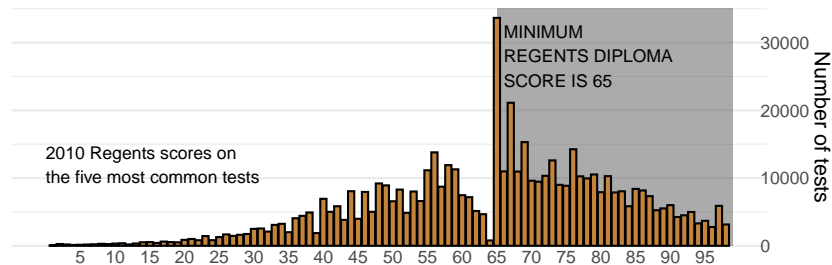
Another striking example comes from a New York Times chart², which summarizes scores from the NYC Regents Exams. As described in the article³, these scores are collected for several reasons, including to determine if a student graduates from high school. In New York City you need a 65 to pass. The distribution of the test scores forces us to notice something somewhat problematic:

¹http://graphics.wsj.com/infectious-diseases-and-vaccines/?mc_cid=711ddeb86e

²<http://graphics8.nytimes.com/images/2011/02/19/nyregion/19schools/19schools-sch-popup.gif>

³<https://www.nytimes.com/2011/02/19/nyregion/19schools.html>

Scraping by



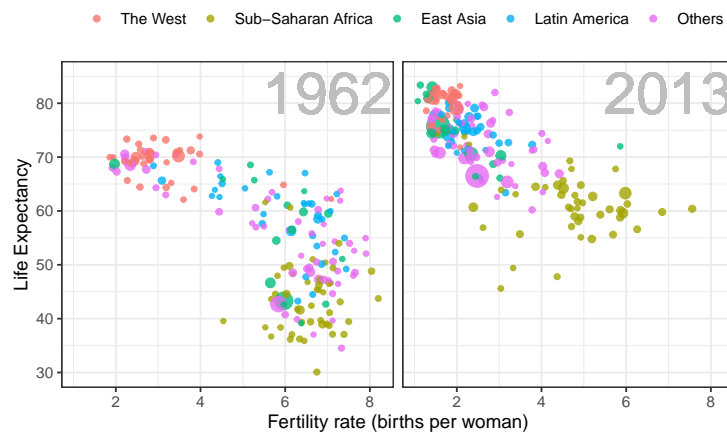
The most common test score is the minimum passing grade, with very few scores just below the threshold. This unexpected result is consistent with students close to passing having their scores bumped up.

This is an example of how data visualization can lead to discoveries which would otherwise be missed if we simply subjected the data to a battery of data analysis tools or procedures. Data visualization is the strongest tool of what we call *exploratory data analysis* (EDA). John W. Tukey⁴, considered the father of EDA, once said,

“The greatest value of a picture is when it forces us to notice what we never expected to see.”

Many widely used data analysis tools were initiated by discoveries made via EDA. EDA is perhaps the most important part of data analysis, yet it is one that is often overlooked.

Data visualization is also now pervasive in philanthropic and educational organizations. In the talks *New Insights on Poverty*⁵ and *The Best Stats You’ve Ever Seen*⁶, Hans Rosling forces us to notice the unexpected with a series of plots related to world health and economics. In his videos, he uses animated graphs to show us how the world is changing and how old narratives are no longer true.



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

⁵https://www.ted.com/talks/hans_rosling_reveals_new_insights_on_poverty?language=en

⁶https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen

It is also important to note that mistakes, biases, systematic errors and other unexpected problems often lead to data that should be handled with care. Failure to discover these problems can give rise to flawed analyses and false discoveries. As an example, consider that measurement devices sometimes fail and that most data analysis procedures are not designed to detect these. Yet these data analysis procedures will still give you an answer. The fact that it can be difficult or impossible to notice an error just from the reported results makes data visualization particularly important.

In this part of the book, we will learn the basics of data visualization and exploratory data analysis by using three motivating examples. We will use the **ggplot2** package to code. To learn the very basics, we will start with a somewhat artificial example: heights reported by students. Then we will cover the two examples mentioned above: 1) world health and economics and 2) infectious disease trends in the United States.

Of course, there is much more to data visualization than what we cover here. The following are references for those who wish to learn more:

- ER Tufte (1983) The visual display of quantitative information. Graphics Press.
- ER Tufte (1990) Envisioning information. Graphics Press.
- ER Tufte (1997) Visual explanations. Graphics Press.
- WS Cleveland (1993) Visualizing data. Hobart Press.
- WS Cleveland (1994) The elements of graphing data. CRC Press.
- A Gelman, C Pasarica, R Dodhia (2002) Let's practice what we preach: Turning tables into graphs. The American Statistician 56:121-130.
- NB Robbins (2004) Creating more effective graphs. Wiley.
- A Cairo (2013) The functional art: An introduction to information graphics and visualization. New Riders.
- N Yau (2013) Data points: Visualization that means something. Wiley.

We also do not cover interactive graphics, a topic that is too advanced for this book. Some useful resources for those interested in learning more can be found below:

- <https://shiny.rstudio.com/>
- <https://d3js.org/>

7

ggplot2

Exploratory data visualization is perhaps the greatest strength of R. One can quickly go from idea to data to plot with a unique balance of flexibility and ease. For example, Excel may be easier than R for some plots, but it is nowhere near as flexible. D3.js may be more flexible and powerful than R, but it takes much longer to generate a plot.

Throughout the book, we will be creating plots using the **ggplot2**¹ package.

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

Many other approaches are available for creating plots in R. In fact, the plotting capabilities that come with a basic installation of R are already quite powerful. There are also other packages for creating graphics such as **grid** and **lattice**. We chose to use **ggplot2** in this book because it breaks plots into components in a way that permits beginners to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and comparatively easy to remember.

One reason **ggplot2** is generally more intuitive for beginners is that it uses a grammar of graphics², the *gg* in **ggplot2**. This is analogous to the way learning grammar can help a beginner construct hundreds of different sentences by learning just a handful of verbs, nouns and adjectives without having to memorize each specific sentence. Similarly, by learning a handful of **ggplot2** building blocks and its grammar, you will be able to create hundreds of different plots.

Another reason **ggplot2** is easy for beginners is that its default behavior is carefully chosen to satisfy the great majority of cases and is visually pleasing. As a result, it is possible to create informative and elegant graphs with relatively simple and readable code.

One limitation is that **ggplot2** is designed to work exclusively with data tables in tidy format (where rows are observations and columns are variables). However, a substantial percentage of datasets that beginners work with are in, or can be converted into, this format. An advantage of this approach is that, assuming that our data is tidy, **ggplot2** simplifies plotting code and the learning of grammar for a variety of plots.

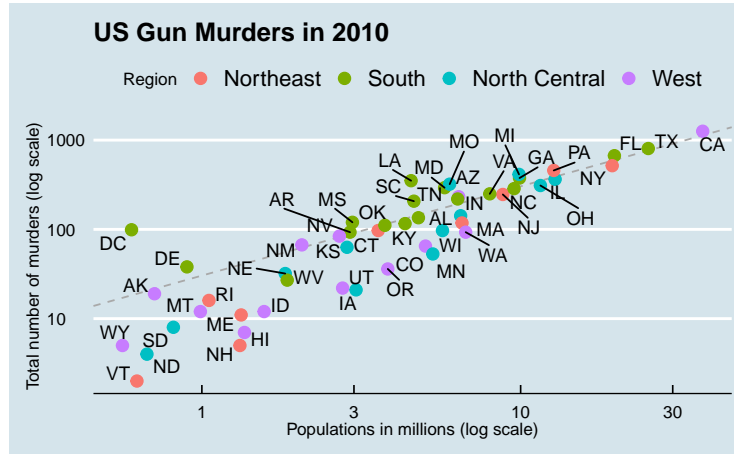
To use **ggplot2** you will have to learn several functions and arguments. These are hard to memorize, so we highly recommend you have the ggplot2 cheat sheet handy. You can get a copy here: <https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf> or simply perform an internet search for “ggplot2 cheat sheet”.

¹<https://ggplot2.tidyverse.org/>

²<http://www.springer.com/us/book/9780387245447>

7.1 The components of a graph

We will construct a graph that summarizes the US murders dataset that looks like this:



We can clearly see how much states vary across population size and the total number of murders. Not surprisingly, we also see a clear relationship between murder totals and population size. A state falling on the dashed grey line has the same murder rate as the US average. The four geographic regions are denoted with color, which depicts how most southern states have murder rates above the average.

This data visualization shows us pretty much all the information in the data table. The code needed to make this plot is relatively simple. We will learn to create the plot part by part.

The first step in learning **ggplot2** is to be able to break a graph apart into components. Let's break down the plot above and introduce some of the **ggplot2** terminology. The main three components to note are:

- **Data:** The US murders data table is being summarized. We refer to this as the **data** component.
- **Geometry:** The plot above is a scatterplot. This is referred to as the **geometry** component. Other possible geometries are barplot, histogram, smooth densities, qqplot, and boxplot. We will learn more about these in the Data Visualization part of the book.
- **Aesthetic mapping:** The plot uses several visual cues to represent the information provided by the dataset. The two most important cues in this plot are the point positions on the x-axis and y-axis, which represent population size and the total number of murders, respectively. Each point represents a different observation, and we *map* data about these observations to visual cues like x- and y-scale. Color is another visual cue that we map to region. We refer to this as the **aesthetic mapping** component. How we define the mapping depends on what **geometry** we are using.

We also note that:

- The points are labeled with the state abbreviations.

- The range of the x-axis and y-axis appears to be defined by the range of the data. They are both on log-scales.
- There are labels, a title, a legend, and we use the style of The Economist magazine.

We will now construct the plot piece by piece.

We start by loading the dataset:

```
library(dslabs)
data(murders)
```

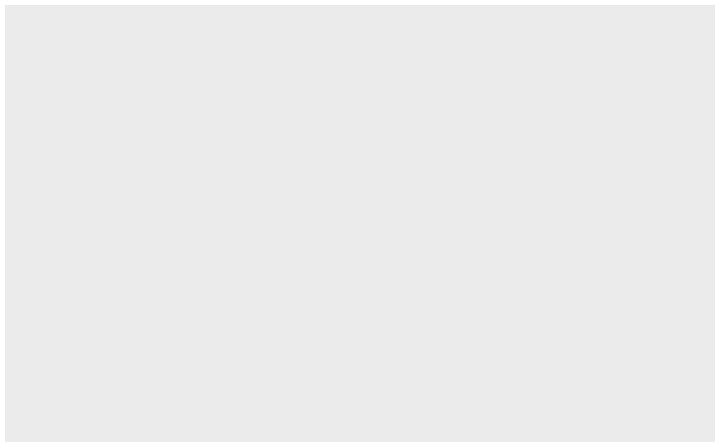
7.2 *ggplot* objects

The first step in creating a **ggplot2** graph is to define a **ggplot** object. We do this with the function **ggplot**, which initializes the graph. If we read the help file for this function, we see that the first argument is used to specify what data is associated with this object:

```
ggplot(data = murders)
```

We can also pipe the data in as the first argument. So this line of code is equivalent to the one above:

```
murders %>% ggplot()
```



It renders a plot, in this case a blank slate since no geometry has been defined. The only style choice we see is a grey background.

What has happened above is that the object was created and, because it was not assigned, it was automatically evaluated. But we can assign our plot to an object, for example like this:


```
p <- ggplot(data = murders)
class(p)
#> [1] "gg"      "ggplot"
```

To render the plot associated with this object, we simply print the object `p`. The following two lines of code each produce the same plot we see above:

```
print(p)
p
```

7.3 Geometries

In `ggplot2` we create graphs by adding *layers*. Layers can define geometries, compute summary statistics, define what scales to use, or even change styles. To add layers, we use the symbol `+`. In general, a line of code will look like this:

DATA %>% `ggplot()` + LAYER 1 + LAYER 2 + ... + LAYER N

Usually, the first added layer defines the geometry. We want to make a scatterplot. What geometry do we use?

Taking a quick look at the cheat sheet, we see that the function used to create plots with this geometry is `geom_point`.



(Image courtesy of RStudio³. CC-BY-4.0 license⁴.)

Geometry function names follow the pattern: `geom_X` where X is the name of the geometry. Some examples include `geom_point`, `geom_bar`, and `geom_histogram`.

For `geom_point` to run properly we need to provide data and a mapping. We have already connected the object `p` with the `murders` data table, and if we add the layer `geom_point` it defaults to using this data. To find out what mappings are expected, we read the **Aesthetics** section of the help file `geom_point` help file:

³<https://github.com/rstudio/cheatsheets>

⁴<https://github.com/rstudio/cheatsheets/blob/master/LICENSE>

```

> Aesthetics
>
> geom_point understands the following aesthetics (required aesthetics are in bold):
>
> x
>
> y
>
> alpha
>
> colour

```

and, as expected, we see that at least two arguments are required **x** and **y**.

7.4 Aesthetic mappings

Aesthetic mappings describe how properties of the data connect with features of the graph, such as distance along an axis, size, or color. The `aes` function connects data with what we see on the graph by defining aesthetic mappings and will be one of the functions you use most often when plotting. The outcome of the `aes` function is often used as the argument of a geometry function. This example produces a scatterplot of total murders versus population in millions:

```

murders %>% ggplot() +
  geom_point(aes(x = population/10^6, y = total))

```

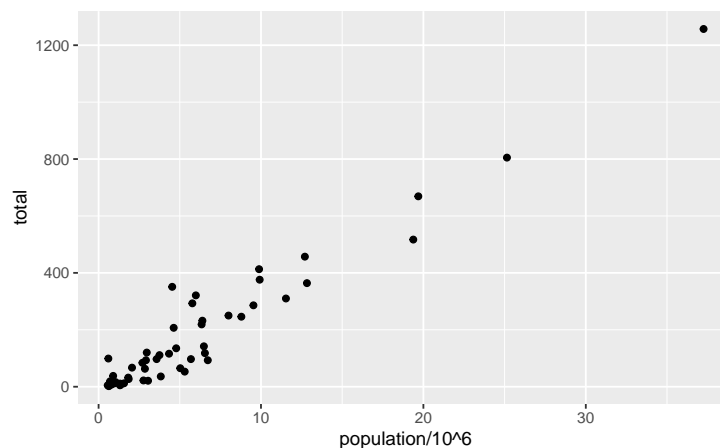
We can drop the `x =` and `y =` if we wanted to since these are the first and second expected arguments, as seen in the help page.

Instead of defining our plot from scratch, we can also add a layer to the `p` object that was defined above as `p <- ggplot(data = murders):`

```

p + geom_point(aes(population/10^6, total))

```



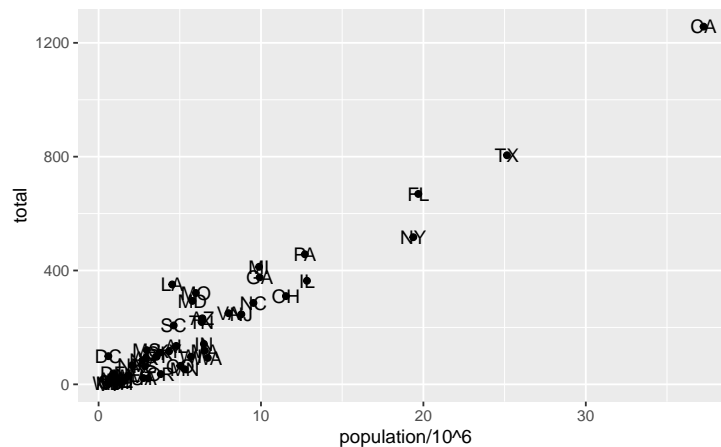
The scale and labels are defined by default when adding this layer. Like **dplyr** functions, **aes** also uses the variable names from the object component: we can use **population** and **total** without having to call them as **murders\$population** and **murders\$total**. The behavior of recognizing the variables from the data component is quite specific to **aes**. With most functions, if you try to access the values of **population** or **total** outside of **aes** you receive an error.

7.5 Layers

A second layer in the plot we wish to make involves adding a label to each point to identify the state. The **geom_label** and **geom_text** functions permit us to add text to the plot with and without a rectangle behind the text, respectively.

Because each point (each state in this case) has a label, we need an aesthetic mapping to make the connection between points and labels. By reading the help file, we learn that we supply the mapping between point and label through the **label** argument of **aes**. So the code looks like this:

```
p + geom_point(aes(population/10^6, total)) +  
  geom_text(aes(population/10^6, total, label = abb))
```



We have successfully added a second layer to the plot.

As an example of the unique behavior of **aes** mentioned above, note that this call:

```
p_test <- p + geom_text(aes(population/10^6, total, label = abb))
```

is fine, whereas this call:

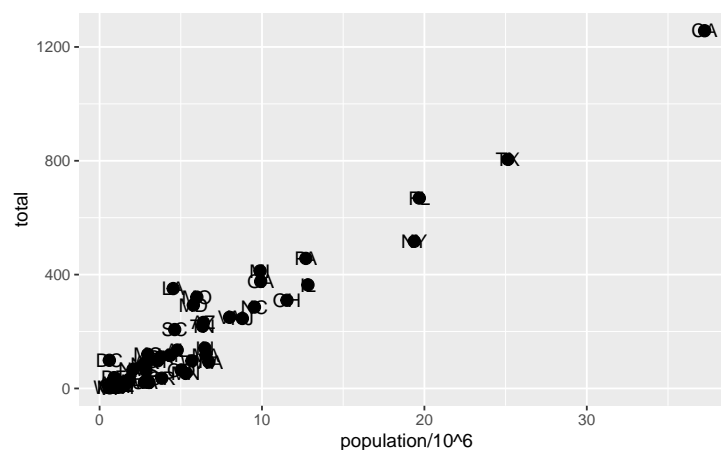
```
p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

will give you an error since `abb` is not found because it is outside of the `aes` function. The layer `geom_text` does not know where to find `abb` since it is a column name and not a global variable.

7.5.1 Tinkering with arguments

Each geometry function has many arguments other than `aes` and `data`. They tend to be specific to the function. For example, in the plot we wish to make, the points are larger than the default size. In the help file we see that `size` is an aesthetic and we can change it like this:

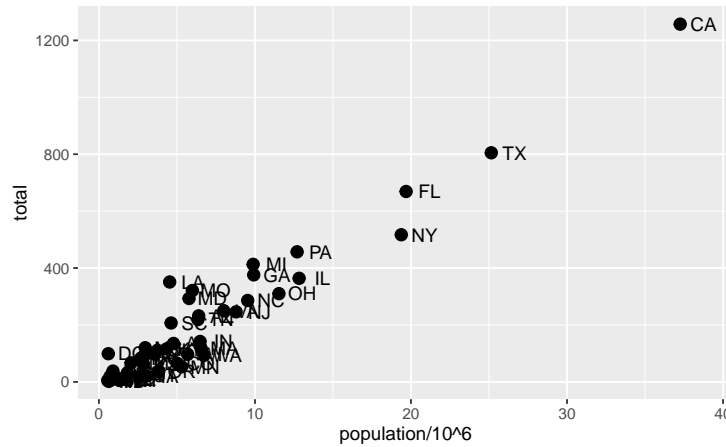
```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))
```



`size` is **not** a mapping: whereas mappings use data from specific observations and need to be inside `aes()`, operations we want to affect all the points the same way do not need to be included inside `aes`.

Now because the points are larger it is hard to see the labels. If we read the help file for `geom_text`, we see the `nudge_x` argument, which moves the text slightly to the right or to the left:

```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb), nudge_x = 1.5)
```



This is preferred as it makes it easier to read the text. In Section 7.11 we learn a better way of assuring we can see the points and the labels.

7.6 Global versus local aesthetic mappings

In the previous line of code, we define the mapping `aes(population/106, total)` twice, once in each geometry. We can avoid this by using a *global* aesthetic mapping. We can do this when we define the blank slate `ggplot` object. Remember that the function `ggplot` contains an argument that permits us to define aesthetic mappings:

```
args(ggplot)
#> function (data = NULL, mapping = aes(), ..., environment = parent.frame())
#> NULL
```

If we define a mapping in `ggplot`, all the geometries that are added as layers will default to this mapping. We redefine `p`:

```
p <- murders %>% ggplot(aes(population/106, total, label = abb))
```

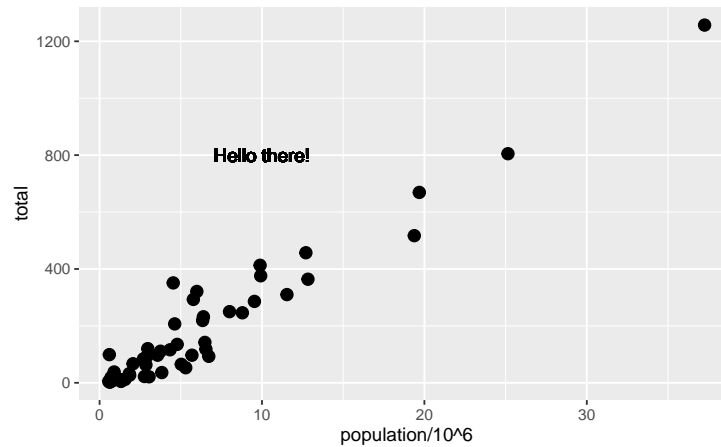
and then we can simply write the following code to produce the previous plot:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 1.5)
```

We keep the `size` and `nudge_x` arguments in `geom_point` and `geom_text`, respectively, because we want to only increase the size of points and only nudge the labels. If we put those arguments in `aes` then they would apply to both plots. Also note that the `geom_point` function does not need a `label` argument and therefore ignores that aesthetic.

If necessary, we can override the global mapping by defining a new mapping within each layer. These *local* definitions override the *global*. Here is an example:

```
p + geom_point(size = 3) +
  geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```

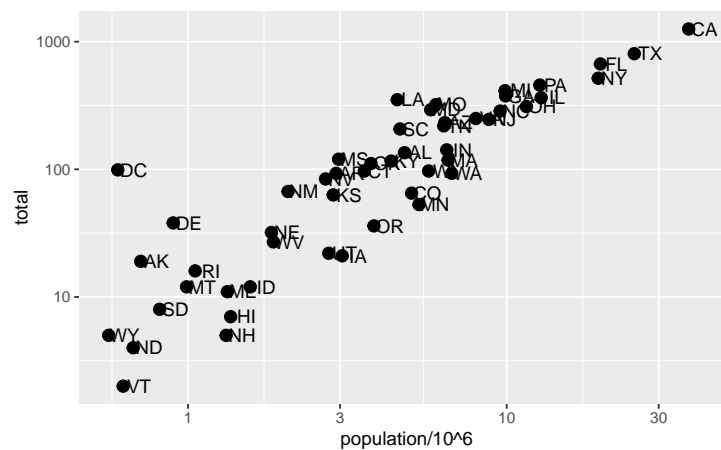


Clearly, the second call to `geom_text` does not use `population` and `total`.

7.7 Scales

First, our desired scales are in log-scale. This is not the default, so this change needs to be added through a *scales* layer. A quick look at the cheat sheet reveals the `scale_x_continuous` function lets us control the behavior of scales. We use them like this:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```



Because we are in the log-scale now, the *nudge* must be made smaller.

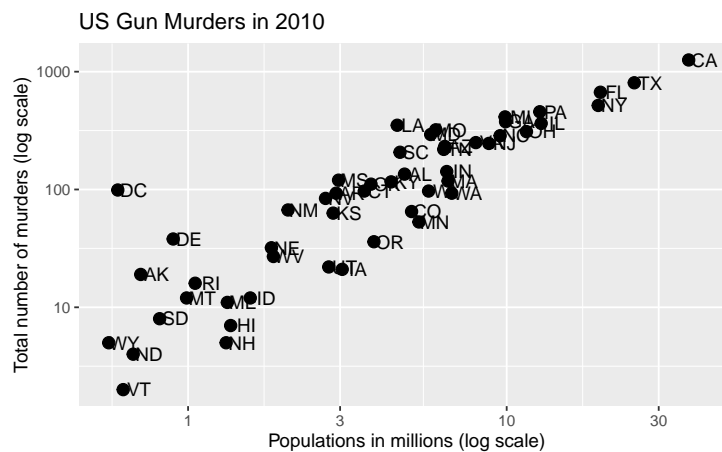
This particular transformation is so common that **ggplot2** provides the specialized functions `scale_x_log10` and `scale_y_log10`, which we can use to rewrite the code like this:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10()
```

7.8 Labels and titles

Similarly, the cheat sheet quickly reveals that to change labels and add a title, we use the following functions:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")
```



We are almost there! All we have left to do is add color, a legend, and optional changes to the style.

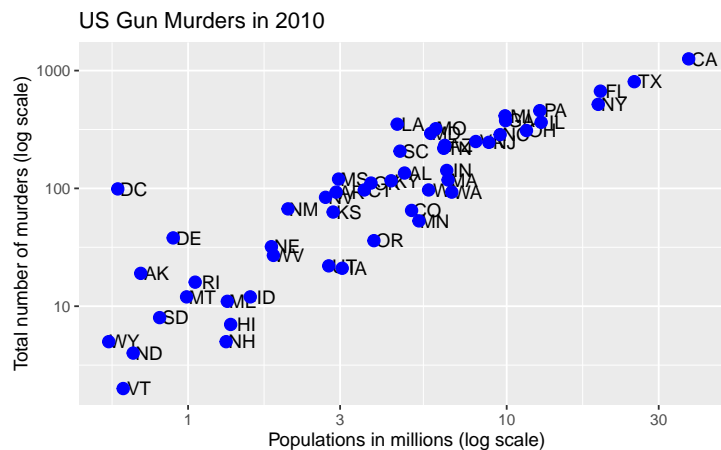
7.9 Categories as colors

We can change the color of the points using the `col` argument in the `geom_point` function. To facilitate demonstration of new features, we will redefine `p` to be everything except the points layer:

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")
```

and then test out what happens by adding different calls to `geom_point`. We can make all the points blue by adding the `color` argument:

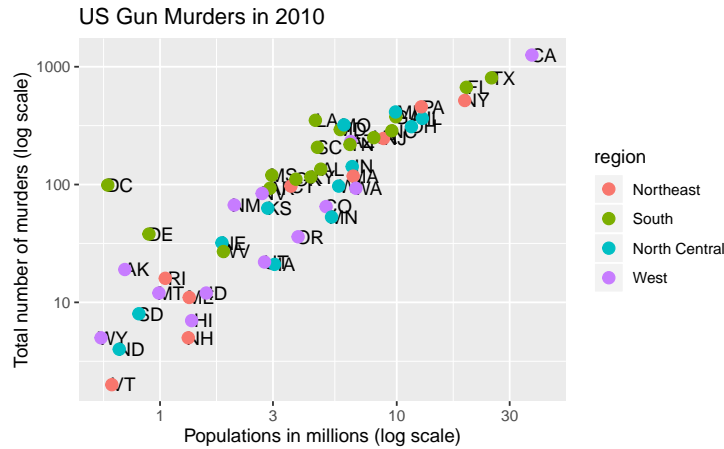
```
p + geom_point(size = 3, color = "blue")
```



This, of course, is not what we want. We want to assign color depending on the geographical region. A nice default behavior of **ggplot2** is that if we assign a categorical variable to color, it automatically assigns a different color to each category and also adds a legend.

Since the choice of color is determined by a feature of each observation, this is an aesthetic mapping. To map each point to a color, we need to use `aes`. We use the following code:

```
p + geom_point(aes(col=region), size = 3)
```

The `x` and `y` mappings are inherited from those already defined in `p`, so we do not redefine them. We also move `aes` to the first argument since that is where mappings are expected in this function call.

Here we see yet another useful default behavior: **ggplot2** automatically adds a legend that maps color to region. To avoid adding this legend we set the `geom_point` argument `show.legend = FALSE`.

7.10 Annotation, shapes, and adjustments

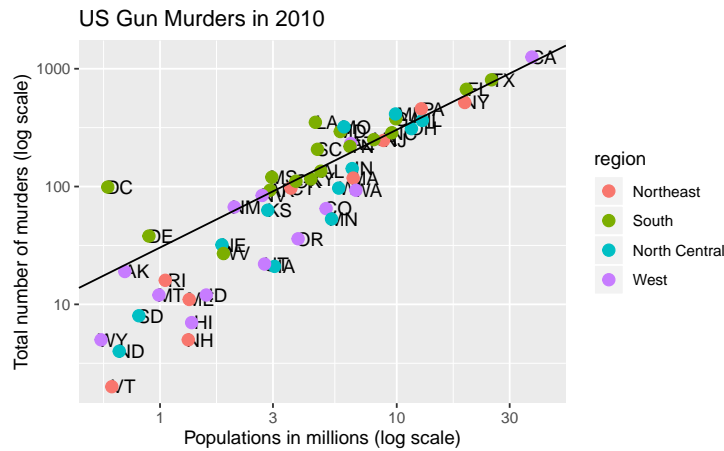
We often want to add shapes or annotation to figures that are not derived directly from the aesthetic mapping; examples include labels, boxes, shaded areas, and lines.

Here we want to add a line that represents the average murder rate for the entire country. Once we determine the per million rate to be r , this line is defined by the formula: $y = rx$, with y and x our axes: total murders and population in millions, respectively. In the log-scale this line turns into: $\log(y) = \log(r) + \log(x)$. So in our plot it's a line with slope 1 and intercept $\log(r)$. To compute this value, we use our **dplyr** skills:

```
r <- murders %>%
  summarize(rate = sum(total) / sum(population) * 10^6) %>%
  pull(rate)
```

To add a line we use the `geom_abline` function. **ggplot2** uses `ab` in the name to remind us we are supplying the intercept (`a`) and slope (`b`). The default line has slope 1 and intercept 0 so we only have to define the intercept:

```
p + geom_point(aes(col=region), size = 3) +
  geom_abline(intercept = log10(r))
```



Here `geom_abline` does not use any information from the data object.

We can change the line type and color of the lines using arguments. Also, we draw it first so it doesn't go over our points.

```
p <- p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3)
```

Note that we have redefined `p` and used this new `p` below and in the next section.

The default plots created by **ggplot2** are already very useful. However, we frequently need to make minor tweaks to the default behavior. Although it is not always obvious how to make these even with the cheat sheet, **ggplot2** is very flexible.

For example, we can make changes to the legend via the `scale_color_discrete` function. In our plot the word *region* is capitalized and we can change it like this:

```
p <- p + scale_color_discrete(name = "Region")
```

7.11 Add-on packages

The power of **ggplot2** is augmented further due to the availability of add-on packages. The remaining changes needed to put the finishing touches on our plot require the **ggthemes** and **ggrepel** packages.

The style of a **ggplot2** graph can be changed using the **theme** functions. Several themes are included as part of the **ggplot2** package. In fact, for most of the plots in this book, we use a function in the **dslabs** package that automatically sets a default theme:

```
ds_theme_set()
```

Many other themes are added by the package **ggthemes**. Among those are the

`theme_economist` theme that we used. After installing the package, you can change the style by adding a layer like this:

```
library(ggthemes)
p + theme_economist()
```

You can see how some of the other themes look by simply changing the function. For instance, you might try the `theme_fivethirtyeight()` theme instead.

The final difference has to do with the position of the labels. In our plot, some of the labels fall on top of each other. The add-on package **ggrepel** includes a geometry that adds labels while ensuring that they don't fall on top of each other. We simply change `geom_text` with `geom_text_repel`.

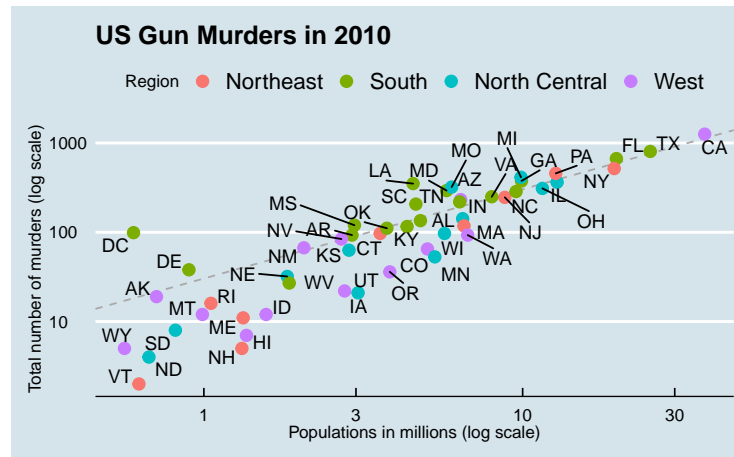
7.12 Putting it all together

Now that we are done testing, we can write one piece of code that produces our desired plot from scratch.

```
library(ggthemes)
library(ggrepel)

r <- murders %>%
  summarize(rate = sum(total) / sum(population) * 10^6) %>%
  pull(rate)

murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  scale_color_discrete(name = "Region") +
  theme_economist()
```



7.13 Quick plots with *qplot*

We have learned the powerful approach to generating visualization with *ggplot*. However, there are instances in which all we want is to make a quick plot of, for example, a histogram of the values in a vector, a scatterplot of the values in two vectors, or a boxplot using categorical and numeric vectors. We demonstrated how to generate these plots with *hist*, *plot*, and *boxplot*. However, if we want to keep consistent with the *ggplot* style, we can use the function *qplot*.

If we have values in two vectors, say:

```
data(murders)
x <- log10(murders$population)
y <- murders$total
```

and we want to make a scatterplot with *ggplot*, we would have to type something like:

```
data.frame(x = x, y = y) %>%
  ggplot(aes(x, y)) +
  geom_point()
```

This seems like too much code for such a simple plot. The *qplot* function sacrifices the flexibility provided by the *ggplot* approach, but allows us to generate a plot quickly.

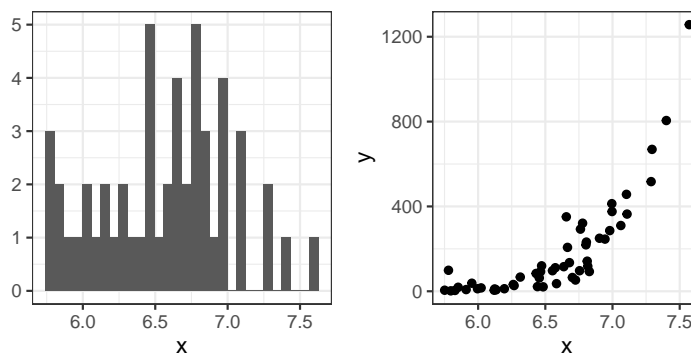
```
qplot(x, y)
```

We will learn more about *qplot* in Section 8.16

7.14 Grids of plots

There are often reasons to graph plots next to each other. The **gridExtra** package permits us to do that:

```
library(gridExtra)
p1 <- qplot(x)
p2 <- qplot(x,y)
grid.arrange(p1, p2, ncol = 2)
```



7.15 Exercises

Start by loading the **dplyr** and **ggplot2** library as well as the **murders** and **heights** data.

```
library(dplyr)
library(ggplot2)
library(dslabs)
data(heights)
data(murders)
```

1. With **ggplot2** plots can be saved as objects. For example we can associate a dataset with a plot object like this

```
p <- ggplot(data = murders)
```

Because **data** is the first argument we don't need to spell it out

```
p <- ggplot(murders)
```

and we can also use the pipe:

```
p <- murders %>% ggplot()
```

What is class of the object `p`?

2. Remember that to print an object you can use the command `print` or simply type the object. Print the object `p` defined in exercise one and describe what you see.

- a. Nothing happens.
- b. A blank slate plot.
- c. A scatterplot.
- d. A histogram.

3. Using the pipe `%>%`, create an object `p` but this time associated with the `heights` dataset instead of the `murders` dataset.

4. What is the class of the object `p` you have just created?

5. Now we are going to add a layer and the corresponding aesthetic mappings. For the `murders` data we plotted total murders versus population sizes. Explore the `murders` data frame to remind yourself what are the names for these two variables and select the correct answer. **Hint:** Look at `?murders`.

- a. `state` and `abb`.
- b. `total_murders` and `population_size`.
- c. `total` and `population`.
- d. `murders` and `size`.

6. To create the scatterplot we add a layer with `geom_point`. The aesthetic mappings require us to define the x-axis and y-axis variables, respectively. So the code looks like this:

```
murders %>% ggplot(aes(x = , y = )) +  
  geom_point()
```

except we have to define the two variables `x` and `y`. Fill this out with the correct variable names.

7. Note that if we don't use argument names, we can obtain the same plot by making sure we enter the variable names in the right order like this:

```
murders %>% ggplot(aes(population, total)) +  
  geom_point()
```

Remake the plot but now with `total` in the x-axis and `population` in the y-axis.

8. If instead of points we want to add text, we can use the `geom_text()` or `geom_label()` geometries. The following code

```
murders %>% ggplot(aes(population, total)) + geom_label()
```

will give us the error message: `Error: geom_label requires the following missing aesthetics: label`

Why is this?

- a. We need to map a character to each point through the label argument in `aes`.
 - b. We need to let `geom_label` know what character to use in the plot.
 - c. The `geom_label` geometry does not require x-axis and y-axis values.
 - d. `geom_label` is not a `ggplot2` command.
9. Rewrite the code above to abbreviation as the label through `aes`
10. Change the color of the labels through blue. How will we do this?
- a. Adding a column called `blue` to `murders`.
 - b. Because each label needs a different color we map the colors through `aes`.
 - c. Use the `color` argument in `ggplot`.
 - d. Because we want all colors to be blue, we do not need to map colors, just use the color argument in `geom_label`.
11. Rewrite the code above to make the labels blue.
12. Now suppose we want to use color to represent the different regions. In this case which of the following is most appropriate:
- a. Adding a column called `color` to `murders` with the color we want to use.
 - b. Because each label needs a different color we map the colors through the color argument of `aes`.
 - c. Use the `color` argument in `ggplot`.
 - d. Because we want all colors to be blue, we do not need to map colors, just use the color argument in `geom_label`.
13. Rewrite the code above to make the labels' color be determined by the state's region.
14. Now we are going to change the x-axis to a log scale to account for the fact the distribution of population is skewed. Let's start by defining an object `p` holding the plot we have made up to now
- ```
p <- murders %>%
 ggplot(aes(population, total, label = abb, color = region)) +
 geom_label()
```
- To change the y-axis to a log scale we learned about the `scale_y_log10()` function. Add this layer to the object `p` to change the scale and render the plot.
15. Repeat the previous exercise but now change both axes to be in the log scale.
16. Now edit the code above to add the title "Gun murder data" to the plot. Hint: use the `ggtitle` function.

# 8

---

## Visualizing data distributions

---

You may have noticed that numerical data is often summarized with the *average* value. For example, the quality of a high school is sometimes summarized with one number: the average score on a standardized test. Occasionally, a second number is reported: the *standard deviation*. For example, you might read a report stating that scores were 680 plus or minus 50 (the standard deviation). The report has summarized an entire vector of scores with just two numbers. Is this appropriate? Is there any important piece of information that we are missing by only looking at this summary rather than the entire list?

Our first data visualization building block is learning to summarize lists of factors or numeric vectors. More often than not, the best way to share or explore this summary is through data visualization. The most basic statistical summary of a list of objects or numbers is its distribution. Once a vector has been summarized as a distribution, there are several data visualization techniques to effectively relay this information.

In this chapter, we first discuss properties of a variety of distributions and how to visualize distributions using a motivating example of student heights. We then discuss the **ggplot2** geometries for these visualizations in Section 8.16.

---

### 8.1 Variable types

We will be working with two types of variables: categorical and numeric. Each can be divided into two other groups: categorical can be ordinal or not, whereas numerical variables can be discrete or continuous.

When each entry in a vector comes from one of a small number of groups, we refer to the data as *categorical data*. Two simple examples are sex (male or female) and regions (Northeast, South, North Central, West). Some categorical data can be ordered even if they are not numbers per se, such as spiciness (mild, medium, hot). In statistics textbooks, ordered categorical data are referred to as *ordinal* data.

Examples of numerical data are population sizes, murder rates, and heights. Some numerical data can be treated as ordered categorical. We can further divide numerical data into continuous and discrete. Continuous variables are those that can take any value, such as heights, if measured with enough precision. For example, a pair of twins may be 68.12 and 68.11 inches, respectively. Counts, such as population sizes, are discrete because they have to be round numbers.

Keep in mind that discrete numeric data can be considered ordinal. Although this is technically true, we usually reserve the term ordinal data for variables belonging to a small number of different groups, with each group having many members. In contrast, when we



have many groups with few cases in each group, we typically refer to them as discrete numerical variables. So, for example, the number of packs of cigarettes a person smokes a day, rounded to the closest pack, would be considered ordinal, while the actual number of cigarettes would be considered a numerical variable. But, indeed, there are examples that can be considered both numerical and ordinal when it comes to visualizing data.

---

## 8.2 Case study: describing student heights

Here we introduce a new motivating problem. It is an artificial one, but it will help us illustrate the concepts needed to understand distributions.

Pretend that we have to describe the heights of our classmates to ET, an extraterrestrial that has never seen humans. As a first step, we need to collect data. To do this, we ask students to report their heights in inches. We ask them to provide sex information because we know there are two different distributions by sex. We collect the data and save it in the `heights` data frame:

```
library(tidyverse)
library(dslabs)
data(heights)
```

One way to convey the heights to ET is to simply send him this list of 1050 heights. But there are much more effective ways to convey this information, and understanding the concept of a distribution will help. To simplify the explanation, we first focus on male heights. We examine the female height data in [Section 8.14](#).

---

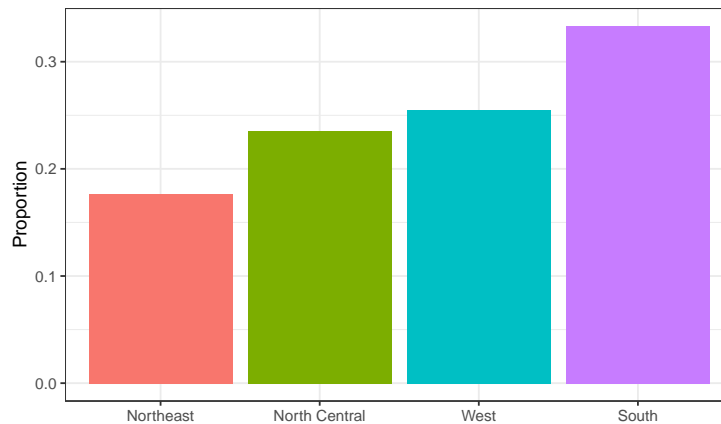
## 8.3 Distribution function

It turns out that, in some cases, the average and the standard deviation are pretty much all we need to understand the data. We will learn data visualization techniques that will help us determine when this two number summary is appropriate. These same techniques will serve as an alternative for when two numbers are not enough.

The most basic statistical summary of a list of objects or numbers is its distribution. The simplest way to think of a distribution is as a compact description of a list with many entries. This concept should not be new for readers of this book. For example, with categorical data, the distribution simply describes the proportion of each unique category. The sex represented in the heights dataset is:

```
#>
#> Female Male
#> 0.227 0.773
```

This two-category *frequency table* is the simplest form of a distribution. We don't really need to visualize it since one number describes everything we need to know: 23% are females and the rest are males. When there are more categories, then a simple barplot describes the distribution. Here is an example with US state regions:



This particular plot simply shows us four numbers, one for each category. We usually use barplots to display a few numbers. Although this particular plot does not provide much more insight than a frequency table itself, it is a first example of how we convert a vector into a plot that succinctly summarizes all the information in the vector. When the data is numerical, the task of displaying distributions is more challenging.

---

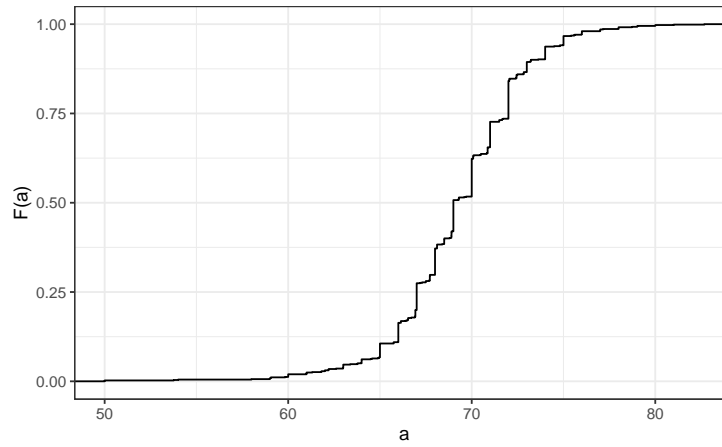
## 8.4 Cumulative distribution functions

Numerical data that are not categorical also have distributions. In general, when data is not categorical, reporting the frequency of each entry is not an effective summary since most entries are unique. In our case study, while several students reported a height of 68 inches, only one student reported a height of 68.503937007874 inches and only one student reported a height 68.8976377952756 inches. We assume that they converted from 174 and 175 centimeters, respectively.

Statistics textbooks teach us that a more useful way to define a distribution for numeric data is to define a function that reports the proportion of the data below  $a$  for all possible values of  $a$ . This function is called the cumulative distribution function (CDF). In statistics, the following notation is used:

$$F(a) = \Pr(x \leq a)$$

Here is a plot of  $F$  for the male height data:



Similar to what the frequency table does for categorical data, the CDF defines the distribution for numerical data. From the plot, we can see that 16% of the values are below 65, since  $F(66) = 0.164$ , or that 84% of the values are below 72, since  $F(72) = 0.841$ , and so on. In fact, we can report the proportion of values between any two heights, say  $a$  and  $b$ , by computing  $F(b) - F(a)$ . This means that if we send this plot above to ET, he will have all the information needed to reconstruct the entire list. Paraphrasing the expression “a picture is worth a thousand words”, in this case, a picture is as informative as 812 numbers.

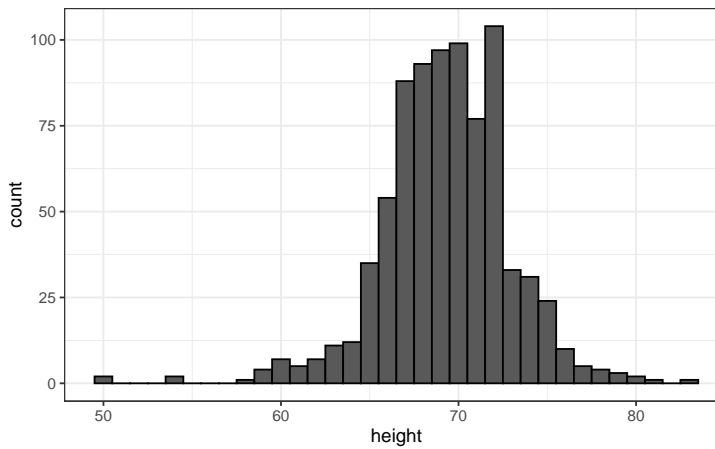
A final note: because CDFs can be defined mathematically the word *empirical* is added to make the distinction when data is used. We therefore use the term empirical CDF (eCDF).

---

## 8.5 Histograms

Although the CDF concept is widely discussed in statistics textbooks, the plot is actually not very popular in practice. The main reason is that it does not easily convey characteristics of interest such as: at what value is the distribution centered? Is the distribution symmetric? What ranges contain 95% of the values? Histograms are much preferred because they greatly facilitate answering such questions. Histograms sacrifice just a bit of information to produce plots that are much easier to interpret.

The simplest way to make a histogram is to divide the span of our data into non-overlapping bins of the same size. Then, for each bin, we count the number of values that fall in that interval. The histogram plots these counts as bars with the base of the bar defined by the intervals. Here is the histogram for the height data splitting the range of values into one inch intervals:  $[49.5, 50.5]$ ,  $[51.5, 52.5]$ ,  $(53.5, 54.5]$ , ...,  $(82.5, 83.5]$



As you can see in the figure above, a histogram is similar to a barplot, but it differs in that the x-axis is numerical, not categorical.

If we send this plot to ET, he will immediately learn some important properties about our data. First, the range of the data is from 50 to 84 with the majority (more than 95%) between 63 and 75 inches. Second, the heights are close to symmetric around 69 inches. Also, by adding up counts, ET could obtain a very good approximation of the proportion of the data in any interval. Therefore, the histogram above is not only easy to interpret, but also provides almost all the information contained in the raw list of 812 heights with about 30 bin counts.

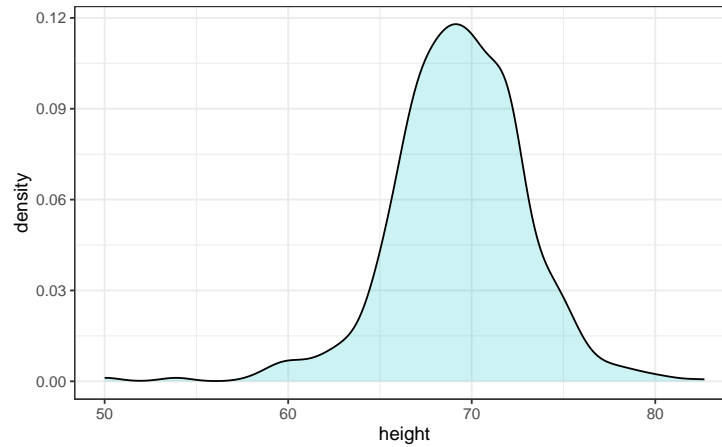
What information do we lose? Note that all values in each interval are treated the same when computing bin heights. So, for example, the histogram does not distinguish between 64, 64.1, and 64.2 inches. Given that these differences are almost unnoticeable to the eye, the practical implications are negligible and we were able to summarize the data to just 23 numbers.

We discuss how to code histograms in Section 8.16.

---

## 8.6 Smoothed density

Smooth density plots are aesthetically more appealing than histograms. Here is what a smooth density plot looks like for our heights data:

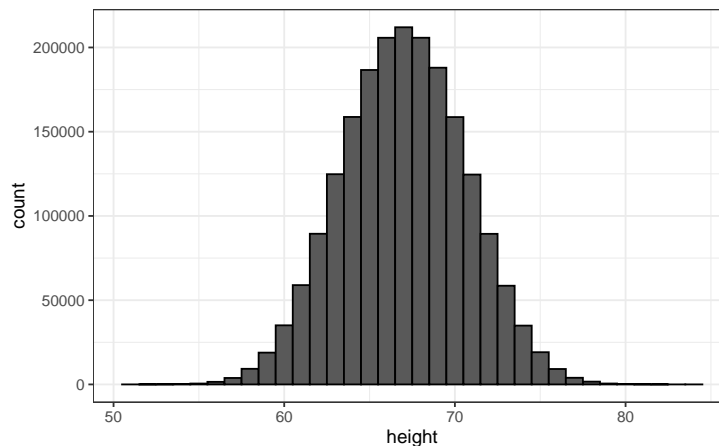


In this plot, we no longer have sharp edges at the interval boundaries and many of the local peaks have been removed. Also, the scale of the y-axis changed from counts to *density*.

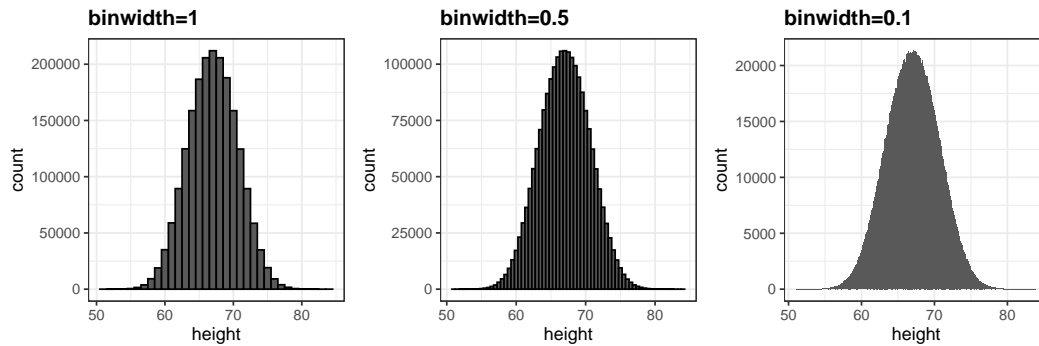
To understand the smooth densities, we have to understand *estimates*, a topic we don't cover until later. However, we provide a heuristic explanation to help you understand the basics so you can use this useful data visualization tool.

The main new concept you must understand is that we assume that our list of observed values is a subset of a much larger list of unobserved values. In the case of heights, you can imagine that our list of 812 male students comes from a hypothetical list containing all the heights of all the male students in all the world measured very precisely. Let's say there are 1,000,000 of these measurements. This list of values has a distribution, like any list of values, and this larger distribution is really what we want to report to ET since it is much more general. Unfortunately, we don't get to see it.

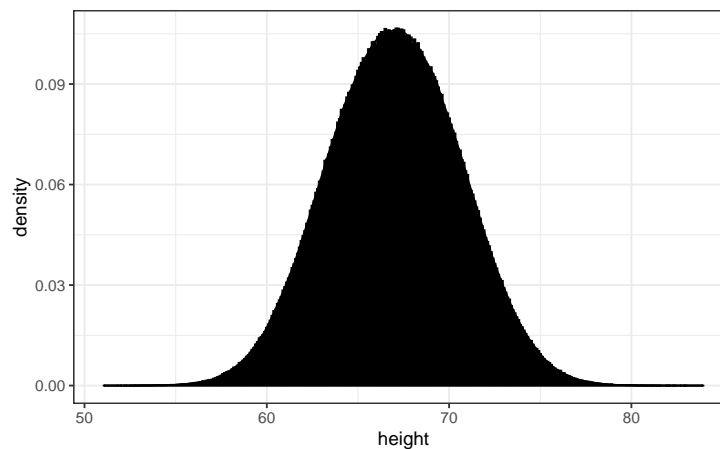
However, we make an assumption that helps us perhaps approximate it. If we had 1,000,000 values, measured very precisely, we could make a histogram with very, very small bins. The assumption is that if we show this, the height of consecutive bins will be similar. This is what we mean by smooth: we don't have big jumps in the heights of consecutive bins. Below we have a hypothetical histogram with bins of size 1:



The smaller we make the bins, the smoother the histogram gets. Here are the histograms with bin width of 1, 0.5, and 0.1:

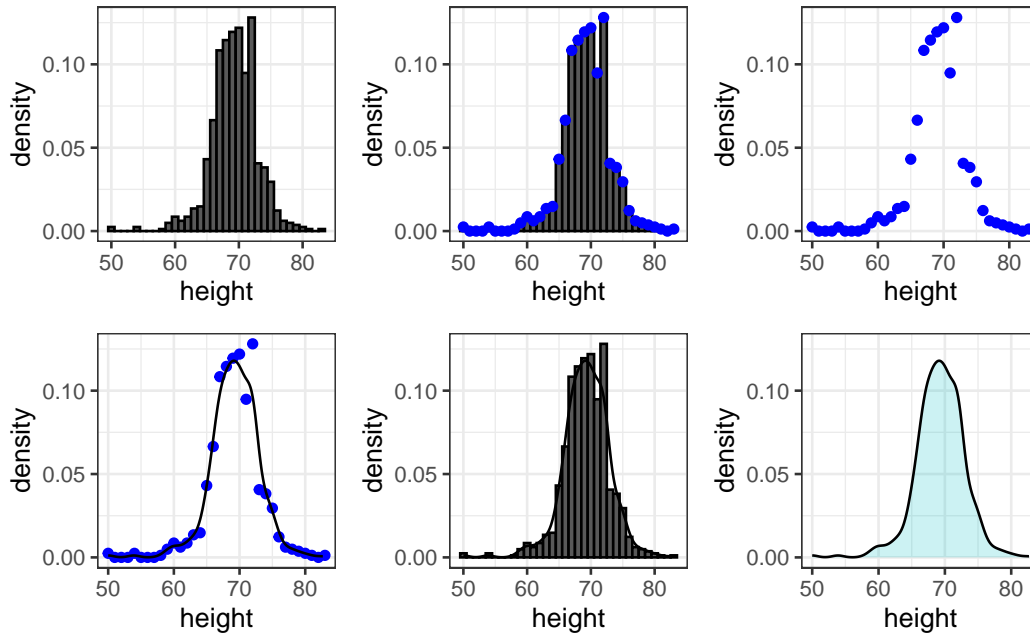


The smooth density is basically the curve that goes through the top of the histogram bars when the bins are very, very small. To make the curve not depend on the hypothetical size of the hypothetical list, we compute the curve on frequencies rather than counts:

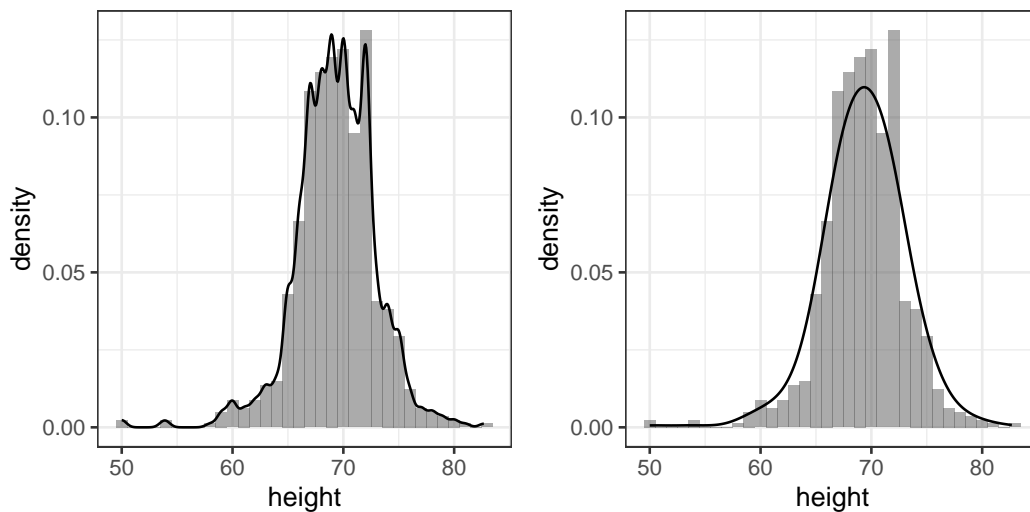


Now, back to reality. We don't have millions of measurements. Instead, we have 812 and we can't make a histogram with very small bins.

We therefore make a histogram, using bin sizes appropriate for our data and computing frequencies rather than counts, and we draw a smooth curve that goes through the tops of the histogram bars. The following plots demonstrate the steps that lead to a smooth density:



However, remember that *smooth* is a relative term. We can actually control the *smoothness* of the curve that defines the smooth density through an option in the function that computes the smooth density curve. Here are two examples using different degrees of smoothness on the same histogram:



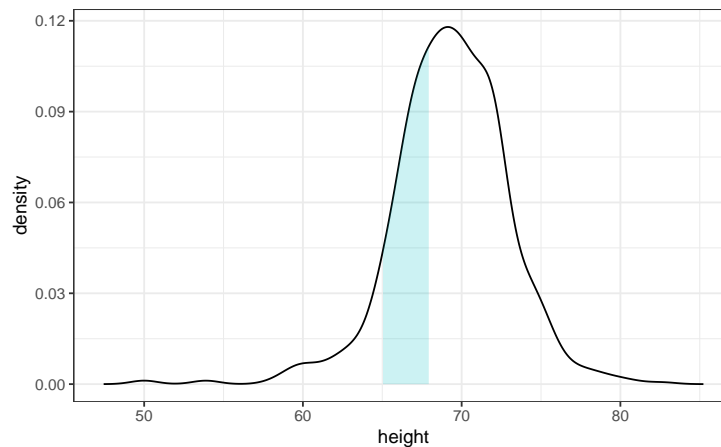
We need to make this choice with care as the resulting visualizations can change our interpretation of the data. We should select a degree of smoothness that we can defend as being representative of the underlying data. In the case of height, we really do have reason to believe that the proportion of people with similar heights should be the same. For example,

the proportion that is 72 inches should be more similar to the proportion that is 71 than to the proportion that is 78 or 65. This implies that the curve should be pretty smooth; that is, the curve should look more like the example on the right than on the left.

While the histogram is an assumption-free summary, the smoothed density is based on some assumptions.

### 8.6.1 Interpreting the y-axis

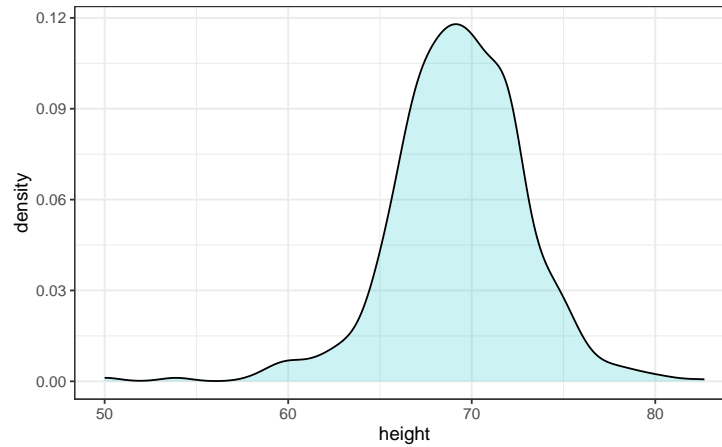
Note that interpreting the y-axis of a smooth density plot is not straightforward. It is scaled so that the area under the density curve adds up to 1. If you imagine we form a bin with a base 1 unit in length, the y-axis value tells us the proportion of values in that bin. However, this is only true for bins of size 1. For other size intervals, the best way to determine the proportion of data in that interval is by computing the proportion of the total area contained in that interval. For example, here are the proportion of values between 65 and 68:



The proportion of this area is about 0.3, meaning that about that proportion is between 65 and 68 inches.

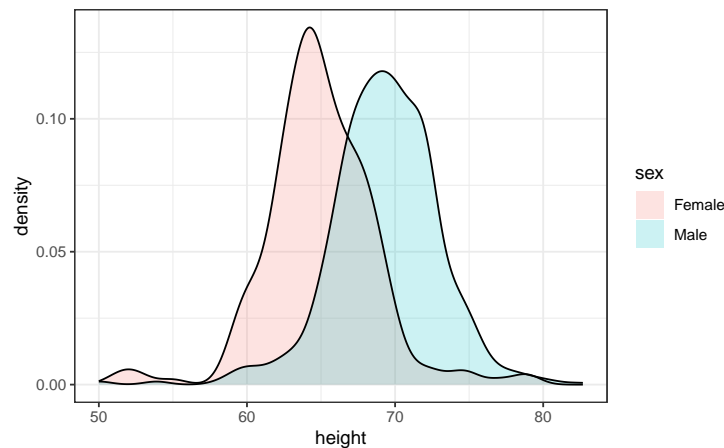
By understanding this, we are ready to use the smooth density as a summary. For this dataset, we would feel quite comfortable with the smoothness assumption, and therefore with sharing this aesthetically pleasing figure with ET, which he could use to understand our male heights data:





### 8.6.2 Densities permit stratification

As a final note, we point out that an advantage of smooth densities over histograms for visualization purposes is that densities make it easier to compare two distributions. This is in large part because the jagged edges of the histogram add clutter. Here is an example comparing male and female heights:

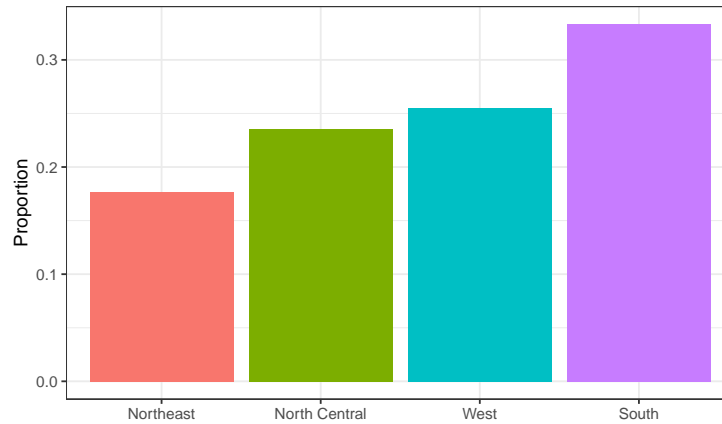


With the right argument, `ggplot` automatically shades the intersecting region with a different color. We will show examples of `ggplot2` code for densities in Section 9 as well as Section 8.16.

---

## 8.7 Exercises

1. In the `murders` dataset, the `region` is a categorical variable and the following is its distribution:

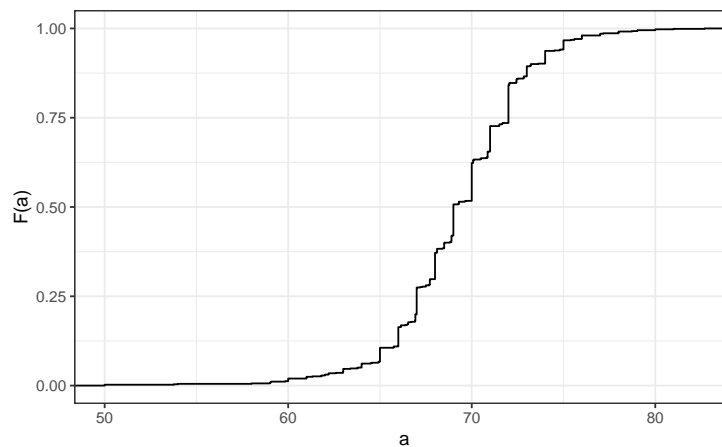


To the closet 5%, what proportion of the states are in the North Central region?

2. Which of the following is true:

- The graph above is a histogram.
- The graph above shows only four numbers with a bar plot.
- Categories are not numbers, so it does not make sense to graph the distribution.
- The colors, not the height of the bars, describe the distribution.

3. The plot below shows the eCDF for male heights:



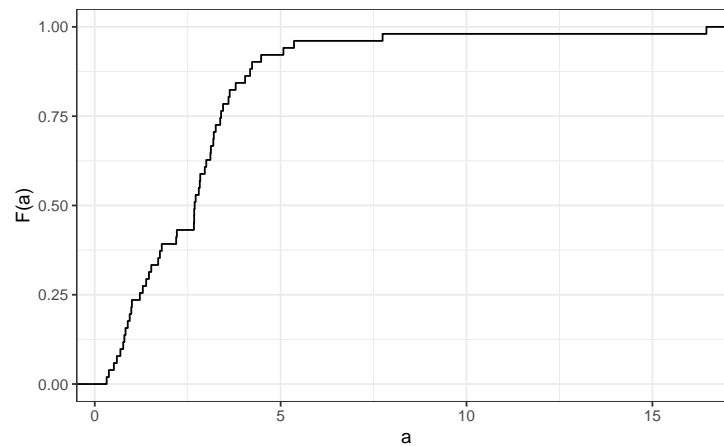
Based on the plot, what percentage of males are shorter than 75 inches?

- 100%
- 95%
- 80%
- 72 inches

4. To the closest inch, what height  $m$  has the property that  $1/2$  of the male students are taller than  $m$  and  $1/2$  are shorter?

- a. 61 inches
- b. 64 inches
- c. 69 inches
- d. 74 inches

5. Here is an eCDF of the murder rates across states:



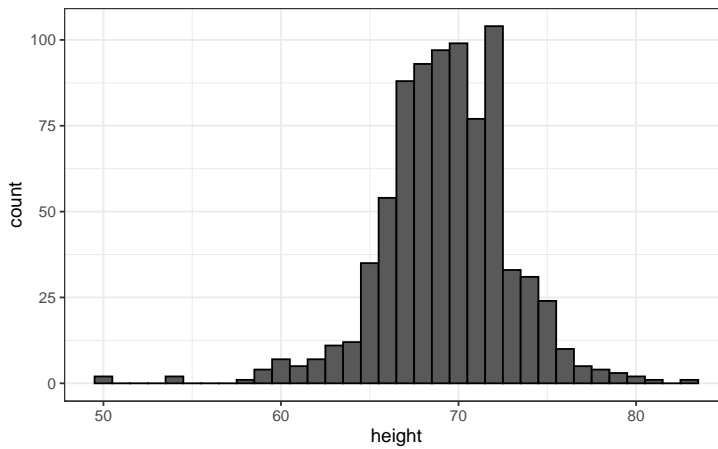
Knowing that there are 51 states (counting DC) and based on this plot, how many states have murder rates larger than 10 per 100,000 people?

- a. 1
- b. 5
- c. 10
- d. 50

6. Based on the eCDF above, which of the following statements are true:

- a. About half the states have murder rates above 7 per 100,000 and the other half below.
- b. Most states have murder rates below 2 per 100,000.
- c. All the states have murder rates above 2 per 100,000.
- d. With the exception of 4 states, the murder rates are below 5 per 100,000.

7. Below is a histogram of male heights in our `heights` dataset:



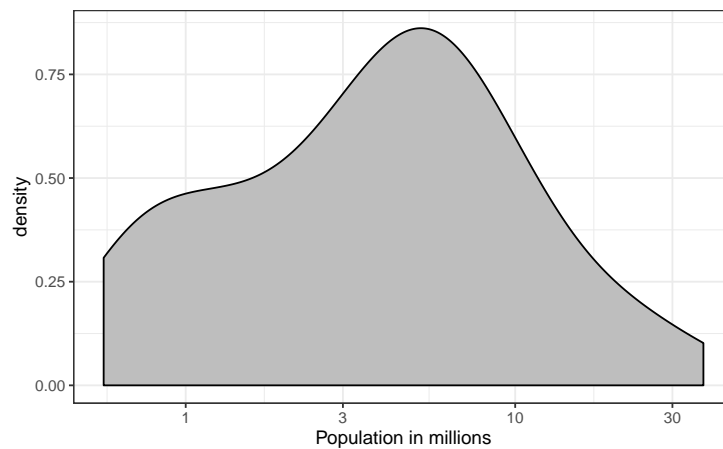
Based on this plot, how many males are between 63.5 and 65.5?

- a. 10
- b. 24
- c. 34
- d. 100

8. About what **percentage** are shorter than 60 inches?

- a. 1%
- b. 10%
- c. 25%
- d. 50%

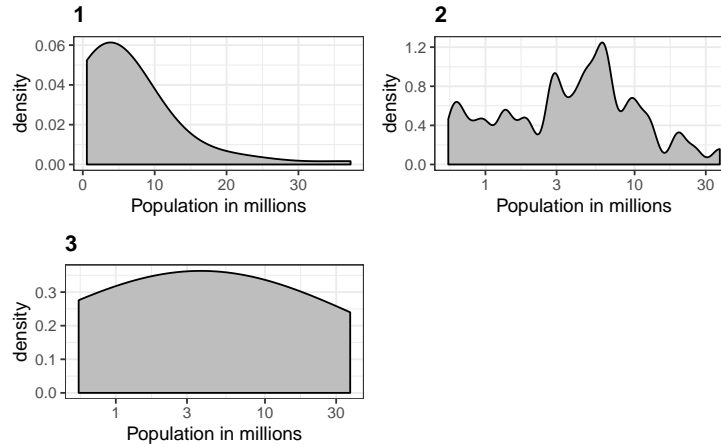
9. Based on the density plot below, about what proportion of US states have populations larger than 10 million?



- a. 0.02
- b. 0.15

- c. 0.50
- d. 0.55

10. Below are three density plots. Is it possible that they are from the same dataset?



Which of the following statements is true:

- a. It is impossible that they are from the same dataset.
- b. They are from the same dataset, but the plots are different due to code errors.
- c. They are the same dataset, but the first and second plot undersmooth and the third oversmooths.
- d. They are the same dataset, but the first is not in the log scale, the second undersmooths, and the third oversmooths.

## 8.8 The normal distribution

Histograms and density plots provide excellent summaries of a distribution. But can we summarize even further? We often see the average and standard deviation used as summary statistics: a two-number summary! To understand what these summaries are and why they are so widely used, we need to understand the normal distribution.

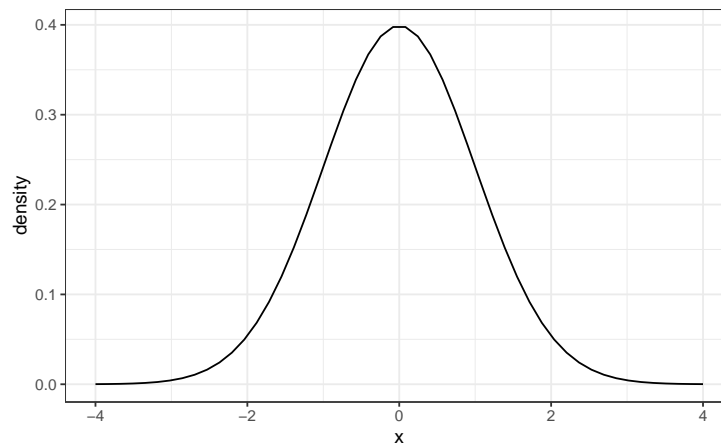
The normal distribution, also known as the bell curve and as the Gaussian distribution, is one of the most famous mathematical concepts in history. A reason for this is that approximately normal distributions occur in many situations, including gambling winnings, heights, weights, blood pressure, standardized test scores, and experimental measurement errors. There are explanations for this, but we describe these later. Here we focus on how the normal distribution helps us summarize data.

Rather than using data, the normal distribution is defined with a mathematical formula. For any interval  $(a, b)$ , the proportion of values in that interval can be computed using this formula:

$$\Pr(a < x < b) = \int_a^b \frac{1}{\sqrt{2\pi}s} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2} dx$$

You don't need to memorize or understand the details of the formula. But note that it is completely defined by just two parameters:  $m$  and  $s$ . The rest of the symbols in the formula represent the interval ends that we determine,  $a$  and  $b$ , and known mathematical constants  $\pi$  and  $e$ . These two parameters,  $m$  and  $s$ , are referred to as the *average* (also called the *mean*) and the *standard deviation* (SD) of the distribution, respectively.

The distribution is symmetric, centered at the average, and most values (about 95%) are within 2 SDs from the average. Here is what the normal distribution looks like when the average is 0 and the SD is 1:



The fact that the distribution is defined by just two parameters implies that if a dataset is approximated by a normal distribution, all the information needed to describe the distribution can be encoded in just two numbers: the average and the standard deviation. We now define these values for an arbitrary list of numbers.

For a list of numbers contained in a vector  $\mathbf{x}$ , the average is defined as:

```
m <- sum(x) / length(x)
```

and the SD is defined as:

```
s <- sqrt(sum((x-mu)^2) / length(x))
```

which can be interpreted as the average distance between values and their average.

Let's compute the values for the height for males which we will store in the object  $x$ :

```
index <- heights$sex == "Male"
x <- heights$height[index]
```

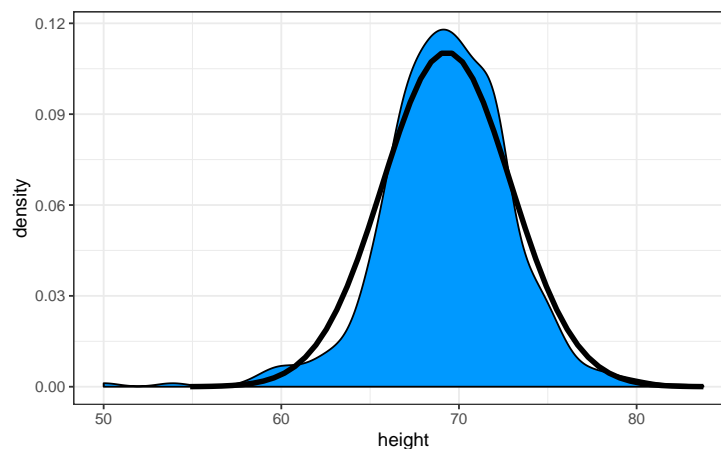
The pre-built functions `mean` and `sd` (note that for reasons explained in Section 16.2, `sd` divides by `length(x)-1` rather than `length(x)`) can be used here:

```

m <- mean(x)
s <- sd(x)
c(average = m, sd = s)
#> average sd
#> 69.31 3.61

```

Here is a plot of the smooth density and the normal distribution with mean = 69.3 and SD = 3.6 plotted as a black line with our student height smooth density in blue:



The normal distribution does appear to be quite a good approximation here. We now will see how well this approximation works at predicting the proportion of values within intervals.

---

## 8.9 Standard units

For data that is approximately normally distributed, it is convenient to think in terms of *standard units*. The standard unit of a value tells us how many standard deviations away from the average it is. Specifically, for a value  $x$  from a vector  $\mathbf{X}$ , we define the value of  $x$  in standard units as  $z = (x - m)/s$  with  $m$  and  $s$  the average and standard deviation of  $\mathbf{X}$ , respectively. Why is this convenient?

First look back at the formula for the normal distribution and note that what is being exponentiated is  $-z^2/2$  with  $z$  equivalent to  $x$  in standard units. Because the maximum of  $e^{-z^2/2}$  is when  $z = 0$ , this explains why the maximum of the distribution occurs at the average. It also explains the symmetry since  $-z^2/2$  is symmetric around 0. Second, note that if we convert the normally distributed data to standard units, we can quickly know if, for example, a person is about average ( $z = 0$ ), one of the largest ( $z \approx 2$ ), one of the smallest ( $z \approx -2$ ), or an extremely rare occurrence ( $z > 3$  or  $z < -3$ ). Remember that it does not matter what the original units are, these rules apply to any data that is approximately normal.

In R, we can obtain standard units using the function `scale`:

```
z <- scale(x)
```

Now to see how many men are within 2 SDs from the average, we simply type:

```
mean(abs(z) < 2)
#> [1] 0.95
```

The proportion is about 95%, which is what the normal distribution predicts! To further confirm that, in fact, the approximation is a good one, we can use quantile-quantile plots.

---

## 8.10 Quantile-quantile plots

A systematic way to assess how well the normal distribution fits the data is to check if the observed and predicted proportions match. In general, this is the approach of the quantile-quantile plot (QQ-plot).

First let's define the theoretical quantiles for the normal distribution. In statistics books we use the symbol  $\Phi(x)$  to define the function that gives us the probability of a standard normal distribution being smaller than  $x$ . So, for example,  $\Phi(-1.96) = 0.025$  and  $\Phi(1.96) = 0.975$ . In R, we can evaluate  $\Phi$  using the `pnorm` function:

```
pnorm(-1.96)
#> [1] 0.025
```

The inverse function  $\Phi^{-1}(x)$  gives us the *theoretical quantiles* for the normal distribution. So, for example,  $\Phi^{-1}(0.975) = 1.96$ . In R, we can evaluate the inverse of  $\Phi$  using the `qnorm` function.

```
qnorm(0.975)
#> [1] 1.96
```

Note that these calculations are for the standard normal distribution by default (mean = 0, standard deviation = 1), but we can also define these for any normal distribution. We can do this using the `mean` and `sd` arguments in the `pnorm` and `qnorm` function. For example, we can use `qnorm` to determine quantiles of a distribution with a specific average and standard deviation

```
qnorm(0.975, mean = 5, sd = 2)
#> [1] 8.92
```

For the normal distribution, all the calculations related to quantiles are done without data, thus the name *theoretical quantiles*. But quantiles can be defined for any distribution, including an empirical one. So if we have data in a vector  $x$ , we can define the quantile associated with any proportion  $p$  as the  $q$  for which the proportion of values below  $q$  is  $p$ . Using R code, we can define  $q$  as the value for which `mean(x <= q) = p`. Notice that not all  $p$  have



a  $q$  for which the proportion is exactly  $p$ . There are several ways of defining the best  $q$  as discussed in the help for the `quantile` function.

To give a quick example, for the male heights data, we have that:

```
mean(x <= 69.5)
#> [1] 0.515
```

So about 50% are shorter or equal to 69 inches. This implies that if  $p = 0.50$  then  $q = 69.5$ .

The idea of a QQ-plot is that if your data is well approximated by normal distribution then the quantiles of your data should be similar to the quantiles of a normal distribution. To construct a QQ-plot, we do the following:

1. Define a vector of  $m$  proportions  $p_1, p_2, \dots, p_m$ .
2. Define a vector of quantiles  $q_1, \dots, q_m$  for your data for the proportions  $p_1, \dots, p_m$ . We refer to these as the *sample quantiles*.
3. Define a vector of theoretical quantiles for the proportions  $p_1, \dots, p_m$  for a normal distribution with the same average and standard deviation as the data.
4. Plot the sample quantiles versus the theoretical quantiles.

Let's construct a QQ-plot using R code. Start by defining the vector of proportions.

```
p <- seq(0.05, 0.95, 0.05)
```

To obtain the quantiles from the data, we can use the `quantile` function like this:

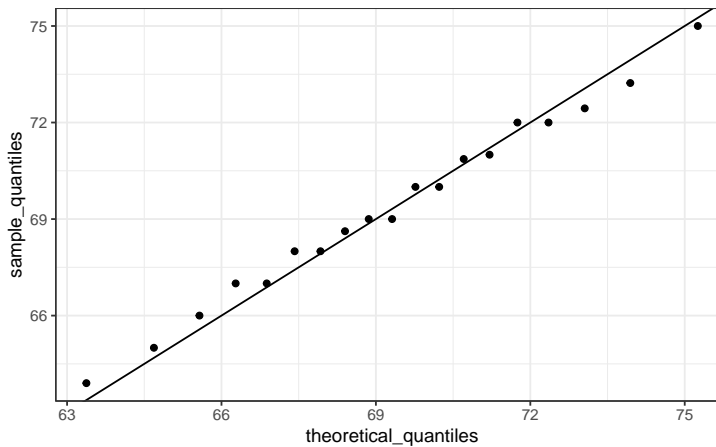
```
sample_quantiles <- quantile(x, p)
```

To obtain the theoretical normal distribution quantiles with the corresponding average and SD, we use the `qnorm` function:

```
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x))
```

To see if they match or not, we plot them against each other and draw the identity line:

```
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```



Notice that this code becomes much cleaner if we use standard units:

```
sample_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```

The above code is included to help describe QQ-plots. However, in practice it is easier to use the **ggplot2** code described in Section 8.16:

```
heights %>% filter(sex == "Male") %>%
 ggplot(aes(sample = scale(height))) +
 geom_qq() +
 geom_abline()
```

While for the illustration above we used 20 quantiles, the default from the `geom_qq` function is to use as many quantiles as data points.

---

## 8.11 Percentiles

Before we move on, let's define some terms that are commonly used in exploratory data analysis.

*Percentiles* are special cases of *quantiles* that are commonly used. The percentiles are the quantiles you obtain when setting the  $p$  at 0.01, 0.02, ..., 0.99. We call, for example, the case of  $p = 0.25$  the 25th percentile, which gives us a number for which 25% of the data is below. The most famous percentile is the 50th, also known as the *median*.

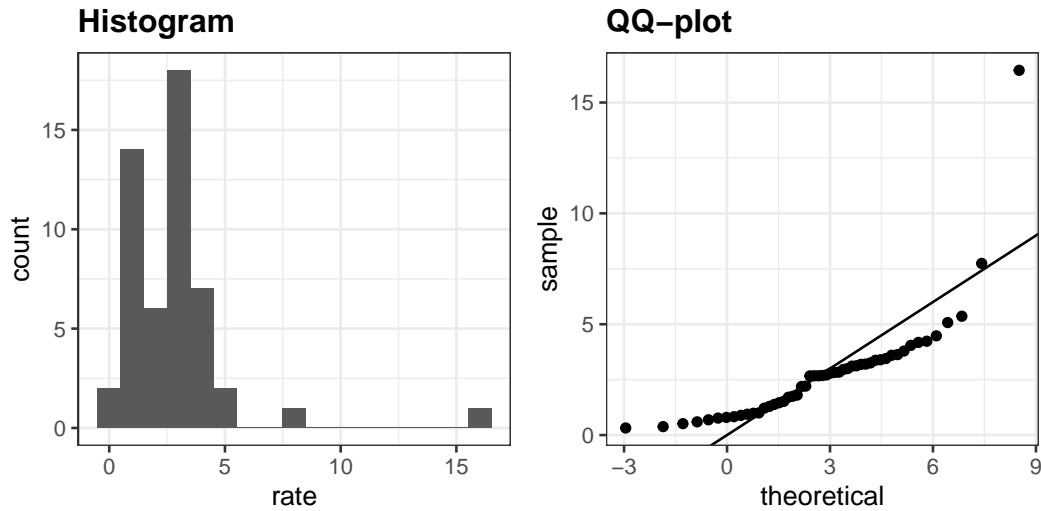
For the normal distribution the *median* and average are the same, but this is generally not the case.

Another special case that receives a name are the *quartiles*, which are obtained when setting  $p = 0.25, 0.50$ , and  $0.75$ .

---

## 8.12 Boxplots

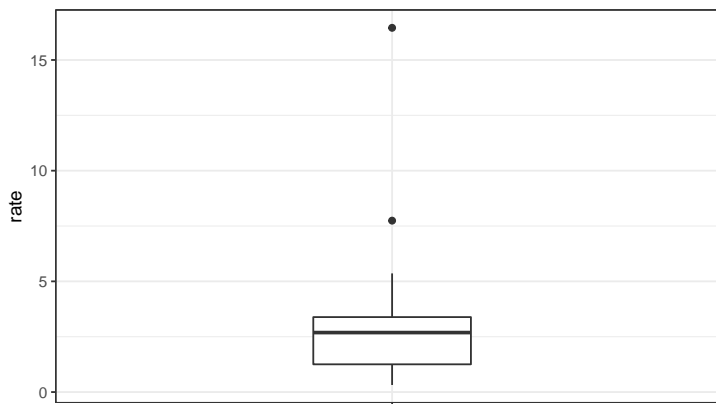
To introduce boxplots we will go back to the US murder data. Suppose we want to summarize the murder rate distribution. Using the data visualization technique we have learned, we can quickly see that the normal approximation does not apply here:



In this case, the histogram above or a smooth density plot would serve as a relatively succinct summary.

Now suppose those used to receiving just two numbers as summaries ask us for a more compact numerical summary.

Here Tukey offered some advice. Provide a five-number summary composed of the range along with the quartiles (the 25th, 50th, and 75th percentiles). Tukey further suggested that we ignore *outliers* when computing the range and instead plot these as independent points. We provide a detailed explanation of outliers later. Finally, he suggested we plot these numbers as a “box” with “whiskers” like this:



with the box defined by the 25% and 75% percentile and the whiskers showing the range. The distance between these two is called the *interquartile* range. The two points are outliers according to Tukey’s definition. The median is shown with a horizontal line. Today, we call these *boxplots*.

From just this simple plot, we know that the median is about 2.5, that the distribution

is not symmetric, and that the range is 0 to 5 for the great majority of states with two exceptions.

We discuss how to make boxplots in Section 8.16.

---

## 8.13 Stratification

In data analysis we often divide observations into groups based on the values of one or more variables associated with those observations. For example in the next section we divide the height values into groups based on a sex variable: females and males. We call this procedure *stratification* and refer to the resulting groups as *strata*.

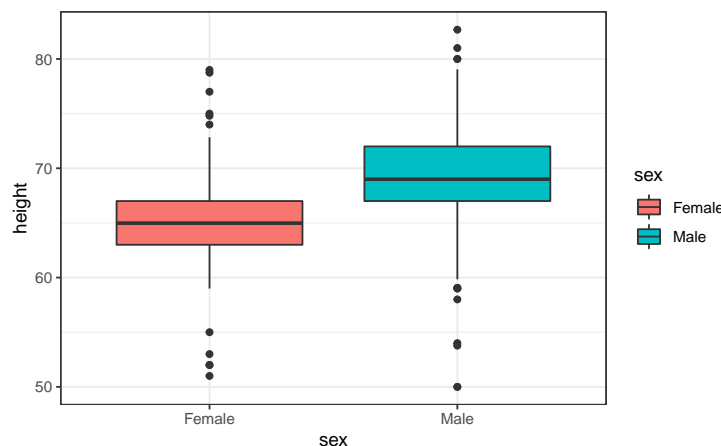
Stratification is common in data visualization because we are often interested in how the distribution of variables differs across different subgroups. We will see several examples throughout this part of the book. We will revisit the concept of stratification when we learn regression in Chapter 17 and in the Machine Learning part of the book.

---

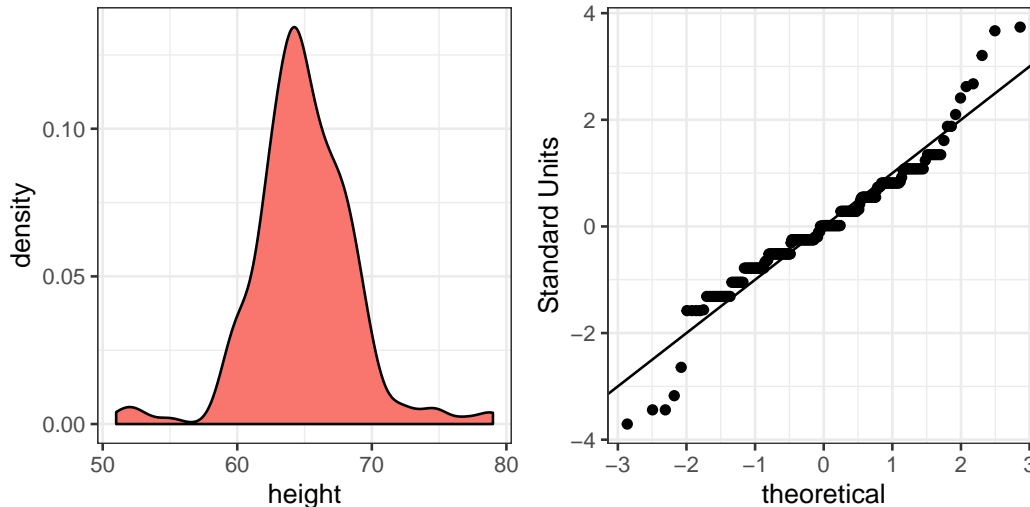
## 8.14 Case study: describing student heights (continued)

Using the histogram, density plots, and QQ-plots, we have become convinced that the male height data is well approximated with a normal distribution. In this case, we report back to ET a very succinct summary: male heights follow a normal distribution with an average of 69.3 inches and a SD of 3.6 inches. With this information, ET will have a good idea of what to expect when he meets our male students. However, to provide a complete picture we need to also provide a summary of the female heights.

We learned that boxplots are useful when we want to quickly compare two or more distributions. Here are the heights for men and women:



The plot immediately reveals that males are, on average, taller than females. The standard deviations appear to be similar. But does the normal approximation also work for the female height data collected by the survey? We expect that they will follow a normal distribution, just like males. However, exploratory plots reveal that the approximation is not as useful:



We see something we did not see for the males: the density plot has a second “bump”. Also, the Q-Q-plot shows that the highest points tend to be taller than expected by the normal distribution. Finally, we also see five points in the Q-Q-plot that suggest shorter than expected heights for a normal distribution. When reporting back to ET, we might need to provide a histogram rather than just the average and standard deviation for the female heights.

However, go back and read Tukey’s quote. We have noticed what we didn’t expect to see. If we look at other female height distributions, we do find that they are well approximated with a normal distribution. So why are our female students different? Is our class a requirement for the female basketball team? Are small proportions of females claiming to be taller than they are? Another, perhaps more likely, explanation is that in the form students used to enter their heights, **FEMALE** was the default sex and some males entered their heights, but forgot to change the sex variable. In any case, data visualization has helped discover a potential flaw in our data.

Regarding the five smallest values, note that these values are:

```
heights %>% filter(sex == "Female") %>%
 top_n(5, desc(height)) %>%
 pull(height)
#> [1] 51 53 55 52 52
```

Because these are reported heights, a possibility is that the student meant to enter 5'1", 5'2", 5'3" or 5'5".

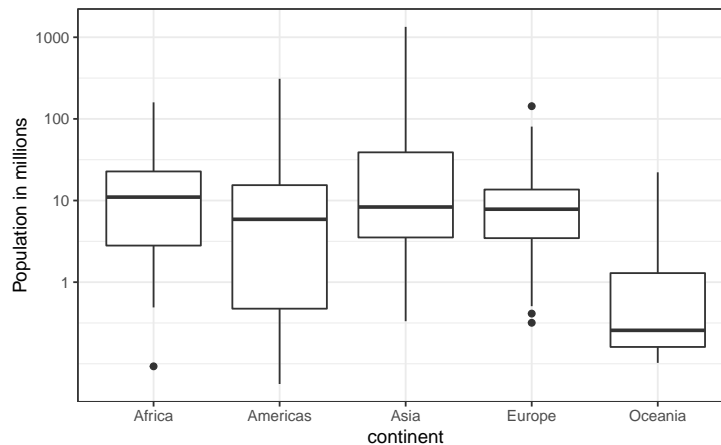
## 8.15 Exercises

1. Define variables containing the heights of males and females like this:

```
library(dslabs)
data(heights)
male <- heights$height[heights$sex == "Male"]
female <- heights$height[heights$sex == "Female"]
```

How many measurements do we have for each?

2. Suppose we can't make a plot and want to compare the distributions side by side. We can't just list all the numbers. Instead, we will look at the percentiles. Create a five row table showing `female_percentiles` and `male_percentiles` with the 10th, 30th, 50th, ..., 90th percentiles for each sex. Then create a data frame with these two as columns.
3. Study the following boxplots showing population sizes by country:



Which continent has the country with the biggest population size?

4. What continent has the largest median population size?
5. What is median population size for Africa to the nearest million?
6. What proportion of countries in Europe have populations below 14 million?
  - a. 0.99
  - b. 0.75
  - c. 0.50
  - d. 0.25
7. If we use a log transformation, which continent shown above has the largest interquartile range?
8. Load the height data set and create a vector `x` with just the male heights:

```
library(dslabs)
data(heights)
x <- heights$height[heights$sex=="Male"]
```

What proportion of the data is between 69 and 72 inches (taller than 69, but shorter or equal to 72)? Hint: use a logical operator and `mean`.

9. Suppose all you know about the data is the average and the standard deviation. Use the normal approximation to estimate the proportion you just calculated. Hint: start by computing the average and standard deviation. Then use the `pnorm` function to predict the proportions.

10. Notice that the approximation calculated in question two is very close to the exact calculation in the first question. Now perform the same task for more extreme values. Compare the exact calculation and the normal approximation for the interval (79,81]. How many times bigger is the actual proportion than the approximation?

11. Approximate the distribution of adult men in the world as normally distributed with an average of 69 inches and a standard deviation of 3 inches. Using this approximation, estimate the proportion of adult men that are 7 feet tall or taller, referred to as *seven footers*. Hint: use the `pnorm` function.

12. There are about 1 billion men between the ages of 18 and 40 in the world. Use your answer to the previous question to estimate how many of these men (18-40 year olds) are seven feet tall or taller in the world?

13. There are about 10 National Basketball Association (NBA) players that are 7 feet tall or higher. Using the answer to the previous two questions, what proportion of the world's 18-to-40-year-old *seven footers* are in the NBA?

14. Repeat the calculations performed in the previous question for LeBron James' height: 6 feet 8 inches. There are about 150 players that are at least that tall.

15. In answering the previous questions, we found that it is not at all rare for a seven footer to become an NBA player. What would be a fair critique of our calculations:

- a. Practice and talent are what make a great basketball player, not height.
- b. The normal approximation is not appropriate for heights.
- c. As seen in question 3, the normal approximation tends to underestimate the extreme values. It's possible that there are more seven footers than we predicted.
- d. As seen in question 3, the normal approximation tends to overestimate the extreme values. It's possible that there are fewer seven footers than we predicted.

---

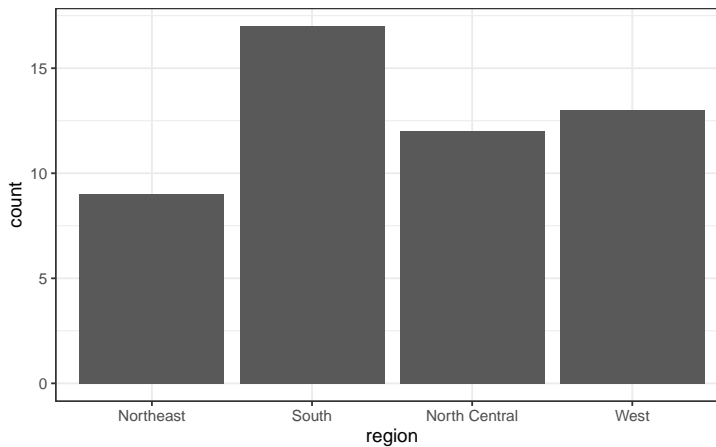
## 8.16 ggplot2 geometries

In Chapter 7, we introduced the **ggplot2** package for data visualization. Here we demonstrate how to generate plots related to distributions, specifically the plots shown earlier in this chapter.

### 8.16.1 Barplots

To generate a barplot we can use the `geom_bar` geometry. The default is to count the number of each category and draw a bar. Here is the plot for the regions of the US.

```
murders %>% ggplot(aes(region)) + geom_bar()
```



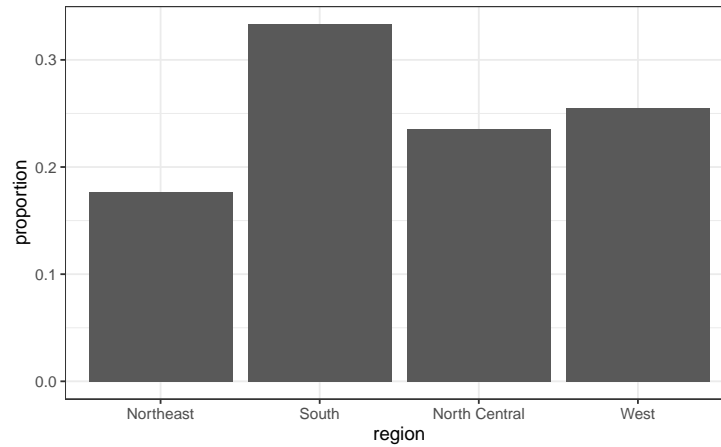
We often already have a table with a distribution that we want to present as a barplot. Here is an example of such a table:

```
data(murders)
tab <- murders %>%
 count(region) %>%
 mutate(proportion = n/sum(n))
tab
#> # A tibble: 4 x 3
#> region n proportion
#> <fct> <int> <dbl>
#> 1 Northeast 9 0.176
#> 2 South 17 0.333
#> 3 North Central 12 0.235
#> 4 West 13 0.255
```

We no longer want `geom_bar` to count, but rather just plot a bar to the height provided by the `proportion` variable. For this we need to provide `x` (the categories) and `y` (the values) and use the `stat="identity"` option.

```
tab %>% ggplot(aes(region, proportion)) + geom_bar(stat = "identity")
```





### 8.16.2 Histograms

To generate histograms we use `geom_histogram`. By looking at the help file for this function, we learn that the only required argument is `x`, the variable for which we will construct a histogram. We dropped the `x` because we know it is the first argument. The code looks like this:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_histogram()
```

If we run the code above, it gives us a message:

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

We previously used a bin size of 1 inch, so the code looks like this:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_histogram(binwidth = 1)
```

Finally, if for aesthetic reasons we want to add color, we use the arguments described in the help file. We also add labels and a title:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_histogram(binwidth = 1, fill = "blue", col = "black") +
 xlab("Male heights in inches") +
 ggtitle("Histogram")
```



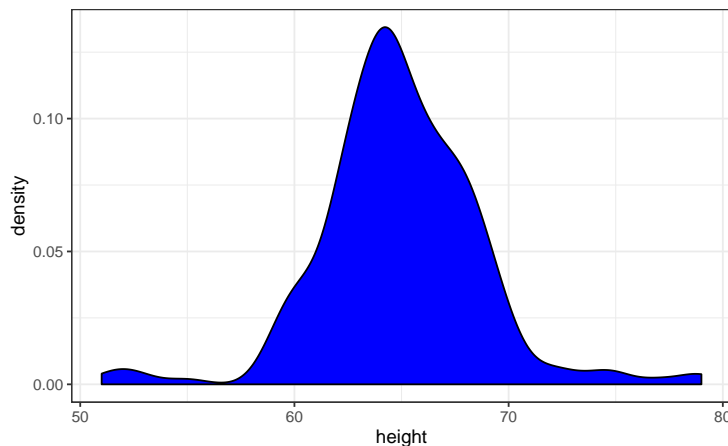
### 8.16.3 Density plots

To create a smooth density, we use the `geom_density`. To make a smooth density plot with the data previously shown as a histogram we can use this code:

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_density()
```

To fill in with color, we can use the `fill` argument.

```
heights %>%
 filter(sex == "Female") %>%
 ggplot(aes(height)) +
 geom_density(fill="blue")
```



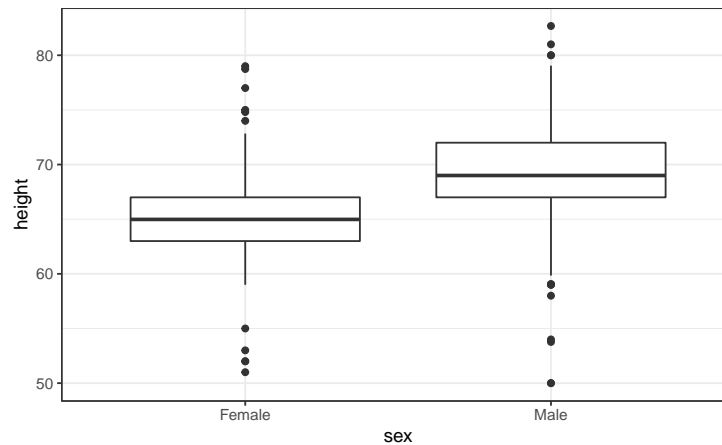
To change the smoothness of the density, we use the `adjust` argument to multiply the

default value by that `adjust`. For example, if we want the bandwidth to be twice as big we use:

```
heights %>%
 filter(sex == "Female") +
 geom_density(fill="blue", adjust = 2)
```

### 8.16.4 Boxplots

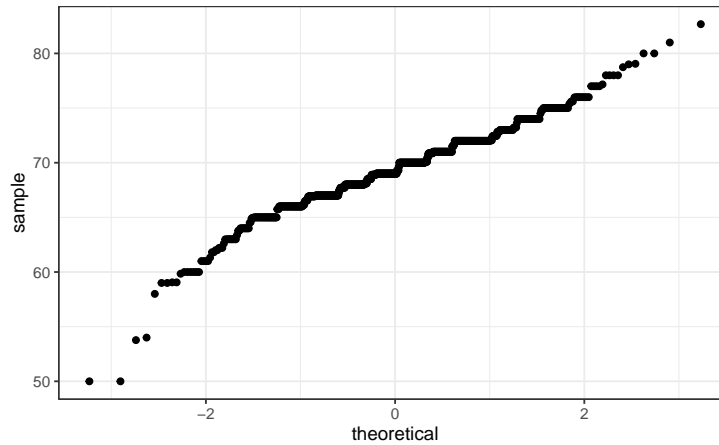
The geometry for boxplot is `geom_boxplot`. As discussed, boxplots are useful for comparing distributions. For example, below are the previously shown heights for women, but compared to men. For this geometry, we need arguments `x` as the categories, and `y` as the values.



### 8.16.5 QQ-plots

For qq-plots we use the `geom_qq` geometry. From the help file, we learn that we need to specify the `sample` (we will learn about samples in a later chapter). Here is the qqplot for men heights.

```
heights %>% filter(sex=="Male") %>%
 ggplot(aes(sample = height)) +
 geom_qq()
```



By default, the sample variable is compared to a normal distribution with average 0 and standard deviation 1. To change this, we use the `dparams` arguments based on the help file. Adding an identity line is as simple as assigning another layer. For straight lines, we use the `geom_abline` function. The default line is the identity line (slope = 1, intercept = 0).

```
params <- heights %>% filter(sex=="Male") %>%
 summarize(mean = mean(height), sd = sd(height))

heights %>% filter(sex=="Male") %>%
 ggplot(aes(sample = height)) +
 geom_qq(dparams = params) +
 geom_abline()
```

Another option here is to scale the data first and then make a qqplot against the standard normal.

```
heights %>%
 filter(sex=="Male") %>%
 ggplot(aes(sample = scale(height))) +
 geom_qq() +
 geom_abline()
```

### 8.16.6 Images

Images were not needed for the concepts described in this chapter, but we will use images in Section 10.14, so we introduce the two geometries used to create images: `geom_tile` and `geom_raster`. They behave similarly; to see how they differ, please consult the help file. To create an image in **ggplot2** we need a data frame with the x and y coordinates as well as the values associated with each of these. Here is a data frame.

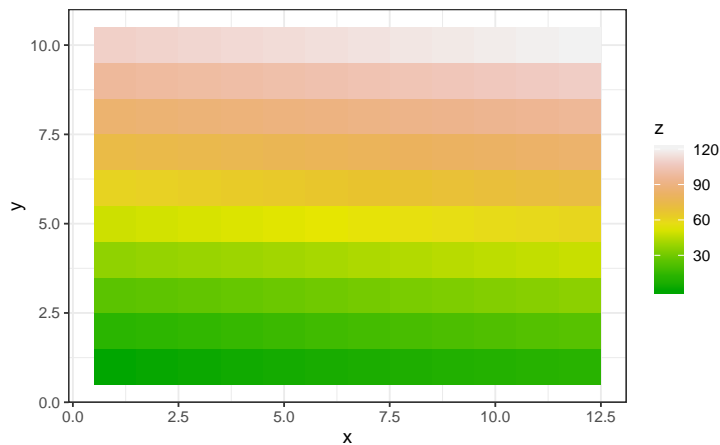
```
x <- expand.grid(x = 1:12, y = 1:10) %>%
 mutate(z = 1:120)
```

Note that this is the tidy version of a matrix, `matrix(1:120, 12, 10)`. To plot the image we use the following code:

```
x %>% ggplot(aes(x, y, fill = z)) +
 geom_raster()
```

With these images you will often want to change the color scale. This can be done through the `scale_fill_gradientn` layer.

```
x %>% ggplot(aes(x, y, fill = z)) +
 geom_raster() +
 scale_fill_gradientn(colors = terrain.colors(10))
```



### 8.16.7 Quick plots

In Section 7.13 we introduced `qplot` as a useful function when we need to make a quick scatterplot. We can also use `qplot` to make histograms, density plots, boxplot, qqplots and more. Although it does not provide the level of control of `ggplot`, `qplot` is definitely useful as it permits us to make a plot with a short snippet of code.

Suppose we have the female heights in an object `x`:

```
x <- heights %>%
 filter(sex=="Male") %>%
 pull(height)
```

To make a quick histogram we can use:

```
qplot(x)
```

The function guesses that we want to make a histogram because we only supplied one variable. In Section 7.13 we saw that if we supply `qplot` two variables, it automatically makes a scatterplot.

To make a quick `qplot` you have to use the `sample` argument. Note that we can add layers just as we do with `ggplot`.

```
qplot(sample = scale(x)) + geom_abline()
```

If we supply a factor and a numeric vector, we obtain a plot like the one below. Note that in the code below we are using the `data` argument. Because the data frame is not the first argument in `qplot`, we have to use the dot operator.

```
heights %>% qplot(sex, height, data = .)
```

We can also select a specific geometry by using the `geom` argument. So to convert the plot above to a boxplot, we use the following code:

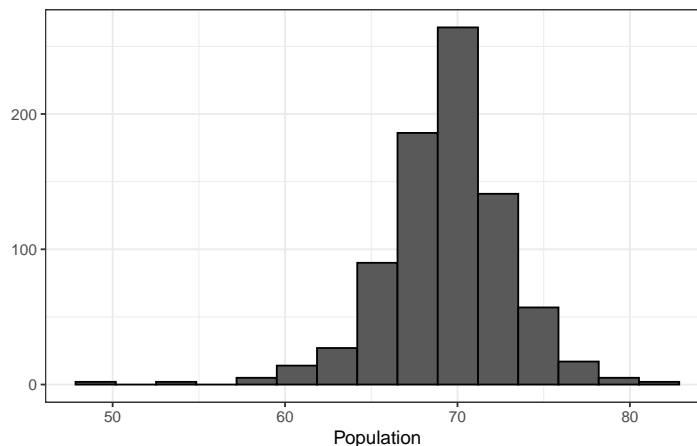
```
heights %>% qplot(sex, height, data = ., geom = "boxplot")
```

We can also use the `geom` argument to generate a density plot instead of a histogram:

```
qplot(x, geom = "density")
```

Although not as much as with `ggplot`, we do have some flexibility to improve the results of `qplot`. Looking at the help file we see several ways in which we can improve the look of the histogram above. Here is an example:

```
qplot(x, bins=15, color = I("black"), xlab = "Population")
```



**Technical note:** The reason we use `I("black")` is because we want `qplot` to treat "black" as a character rather than convert it to a factor, which is the default behavior within `aes`, which is internally called here. In general, the function `I` is used in R to say "keep it as it is".

---

## 8.17 Exercises

1. Now we are going to use the `geom_histogram` function to make a histogram of the heights in the `height` data frame. When reading the documentation for this function we see that it requires just one mapping, the values to be used for the histogram. Make a histogram of all the plots.

What is the variable containing the heights?

- a. `sex`
- b. `heights`
- c. `height`
- d. `heights$height`

2. Now create a ggplot object using the pipe to assign the heights data to a ggplot object. Assign `height` to the x values through the `aes` function.

3. Now we are ready to add a layer to actually make the histogram. Use the object created in the previous exercise and the `geom_histogram` function to make the histogram.

4. Note that when we run the code in the previous exercise we get the warning: `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

Use the `binwidth` argument to change the histogram made in the previous exercise to use bins of size 1 inch.

5. Instead of a histogram, we are going to make a smooth density plot. In this case we will not make an object, but instead render the plot with one line of code. Change the geometry in the code previously used to make a smooth density instead of a histogram.

6. Now we are going to make a density plot for males and females separately. We can do this using the `group` argument. We assign groups via the aesthetic mapping as each point needs to a group before making the calculations needed to estimate a density.

7. We can also assign groups through the `color` argument. This has the added benefit that it uses color to distinguish the groups. Change the code above to use color.

8. We can also assign groups through the `fill` argument. This has the added benefit that it uses colors to distinguish the groups, like this:

```
heights %>%
 ggplot(aes(height, fill = sex)) +
 geom_density()
```

However, here the second density is drawn over the other. We can make the curves more visible by using alpha blending to add transparency. Set the alpha parameter to 0.2 in the `geom_density` function to make this change.

# 9

---

## *Data visualization in practice*

---

In this chapter, we will demonstrate how relatively simple **ggplot2** code can create insightful and aesthetically pleasing plots. As motivation we will create plots that help us better understand trends in world health and economics. We will implement what we learned in Chapters 7 and 8.16 and learn how to augment the code to perfect the plots. As we go through our case study, we will describe relevant general data visualization principles and learn concepts such as *faceting*, *time series plots*, *transformations*, and *ridge plots*.

---

### 9.1 Case study: new insights on poverty

Hans Rosling<sup>1</sup> was the co-founder of the Gapminder Foundation<sup>2</sup>, an organization dedicated to educating the public by using data to dispel common myths about the so-called developing world. The organization uses data to show how actual trends in health and economics contradict the narratives that emanate from sensationalist media coverage of catastrophes, tragedies, and other unfortunate events. As stated in the Gapminder Foundation’s website:

Journalists and lobbyists tell dramatic stories. That’s their job. They tell stories about extraordinary events and unusual people. The piles of dramatic stories pile up in peoples’ minds into an over-dramatic worldview and strong negative stress feelings: “The world is getting worse!”, “It’s we vs. them!”, “Other people are strange!”, “The population just keeps growing!” and “Nobody cares!”

Hans Rosling conveyed actual data-based trends in a dramatic way of his own, using effective data visualization. This section is based on two talks that exemplify this approach to education: [New Insights on Poverty]<sup>3</sup> and The Best Stats You’ve Ever Seen<sup>4</sup>. Specifically, in this section, we use data to attempt to answer the following two questions:

1. Is it a fair characterization of today’s world to say it is divided into western rich nations and the developing world in Africa, Asia, and Latin America?
2. Has income inequality across countries worsened during the last 40 years?

To answer these questions, we will be using the **gapminder** dataset provided in **dslabs**. This dataset was created using a number of spreadsheets available from the Gapminder Foundation. You can access the table like this:

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Hans\\_Rosling](https://en.wikipedia.org/wiki/Hans_Rosling)

<sup>2</sup><http://www.gapminder.org/>

<sup>3</sup>[https://www.ted.com/talks/hans\\_rosling\\_reveals\\_new\\_insights\\_on\\_poverty?language=en](https://www.ted.com/talks/hans_rosling_reveals_new_insights_on_poverty?language=en)

<sup>4</sup>[https://www.ted.com/talks/hans\\_rosling\\_shows\\_the\\_best\\_stats\\_you\\_ve\\_ever\\_seen](https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen)



```
library(tidyverse)
library(dslabs)
data(gapminder)
gapminder %>% as_tibble()
#> # A tibble: 10,545 x 9
#> country year infant_mortality life_expectancy fertility population
#> <fct> <int> <dbl> <dbl> <dbl> <dbl>
#> 1 Albania 1960 115. 62.9 6.19 1636054
#> 2 Algeria 1960 148. 47.5 7.65 11124892
#> 3 Angola 1960 208 36.0 7.32 5270844
#> 4 Antigu~ 1960 NA 63.0 4.43 54681
#> 5 Argent~ 1960 59.9 65.4 3.11 20619075
#> # ... with 1.054e+04 more rows, and 3 more variables: gdp <dbl>,
#> # continent <fct>, region <fct>
```

### 9.1.1 Hans Rosling's quiz

As done in the *New Insights on Poverty* video, we start by testing our knowledge regarding differences in child mortality across different countries. For each of the six pairs of countries below, which country do you think had the highest child mortality rates in 2015? Which pairs do you think are most similar?

1. Sri Lanka or Turkey
2. Poland or South Korea
3. Malaysia or Russia
4. Pakistan or Vietnam
5. Thailand or South Africa

When answering these questions without data, the non-European countries are typically picked as having higher child mortality rates: Sri Lanka over Turkey, South Korea over Poland, and Malaysia over Russia. It is also common to assume that countries considered to be part of the developing world: Pakistan, Vietnam, Thailand, and South Africa, have similarly high mortality rates.

To answer these questions **with data**, we can use **dplyr**. For example, for the first comparison we see that:

```
gapminder %>%
 filter(year == 2015 & country %in% c("Sri Lanka", "Turkey")) %>%
 select(country, infant_mortality)
#> country infant_mortality
#> 1 Sri Lanka 8.4
#> 2 Turkey 11.6
```

Turkey has the higher infant mortality rate.

We can use this code on all comparisons and find the following:

| country   | infant mortality | country      | infant mortality |
|-----------|------------------|--------------|------------------|
| Sri Lanka | 8.4              | Turkey       | 11.6             |
| Poland    | 4.5              | South Korea  | 2.9              |
| Malaysia  | 6.0              | Russia       | 8.2              |
| Pakistan  | 65.8             | Vietnam      | 17.3             |
| Thailand  | 10.5             | South Africa | 33.6             |

We see that the European countries on this list have higher child mortality rates: Poland has a higher rate than South Korea, and Russia has a higher rate than Malaysia. We also see that Pakistan has a much higher rate than Vietnam, and South Africa has a much higher rate than Thailand. It turns out that when Hans Rosling gave this quiz to educated groups of people, the average score was less than 2.5 out of 5, worse than what they would have obtained had they guessed randomly. This implies that more than ignorant, we are misinformed. In this chapter we see how data visualization helps inform us.

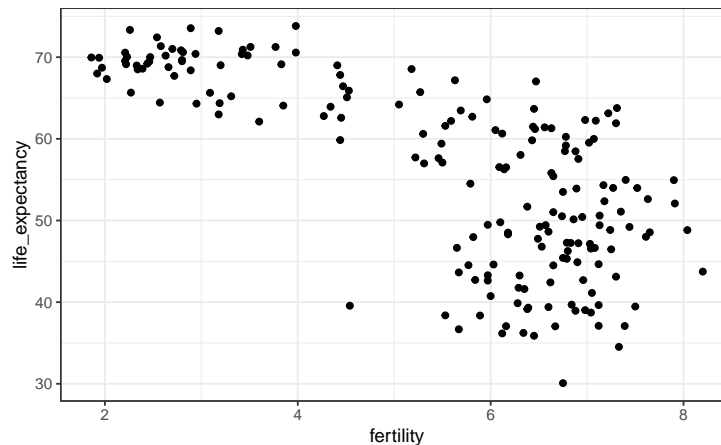
## 9.2 Scatterplots

The reason for this stems from the preconceived notion that the world is divided into two groups: the western world (Western Europe and North America), characterized by long life spans and small families, versus the developing world (Africa, Asia, and Latin America) characterized by short life spans and large families. But do the data support this dichotomous view?

The necessary data to answer this question is also available in our `gapminder` table. Using our newly learned data visualization skills, we will be able to tackle this challenge.

In order to analyze this world view, our first plot is a scatterplot of life expectancy versus fertility rates (average number of children per woman). We start by looking at data from about 50 years ago, when perhaps this view was first cemented in our minds.

```
filter(gapminder, year == 1962) %>%
 ggplot(aes(fertility, life_expectancy)) +
 geom_point()
```

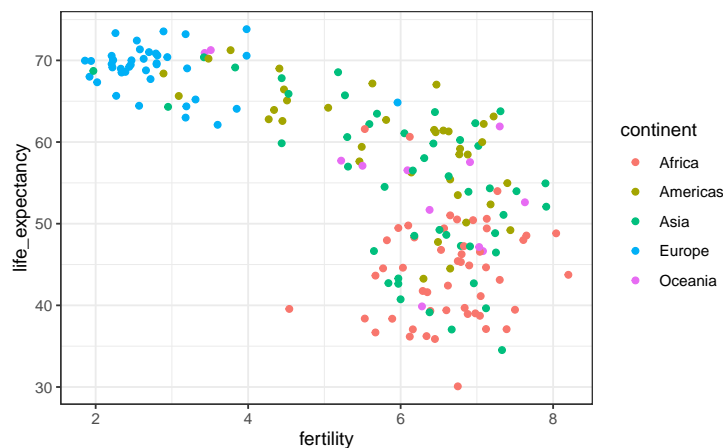


Most points fall into two distinct categories:

1. Life expectancy around 70 years and 3 or fewer children per family.
2. Life expectancy lower than 65 years and more than 5 children per family.

To confirm that indeed these countries are from the regions we expect, we can use color to represent continent.

```
filter(gapminder, year == 1962) %>%
 ggplot(aes(fertility, life_expectancy, color = continent)) +
 geom_point()
```



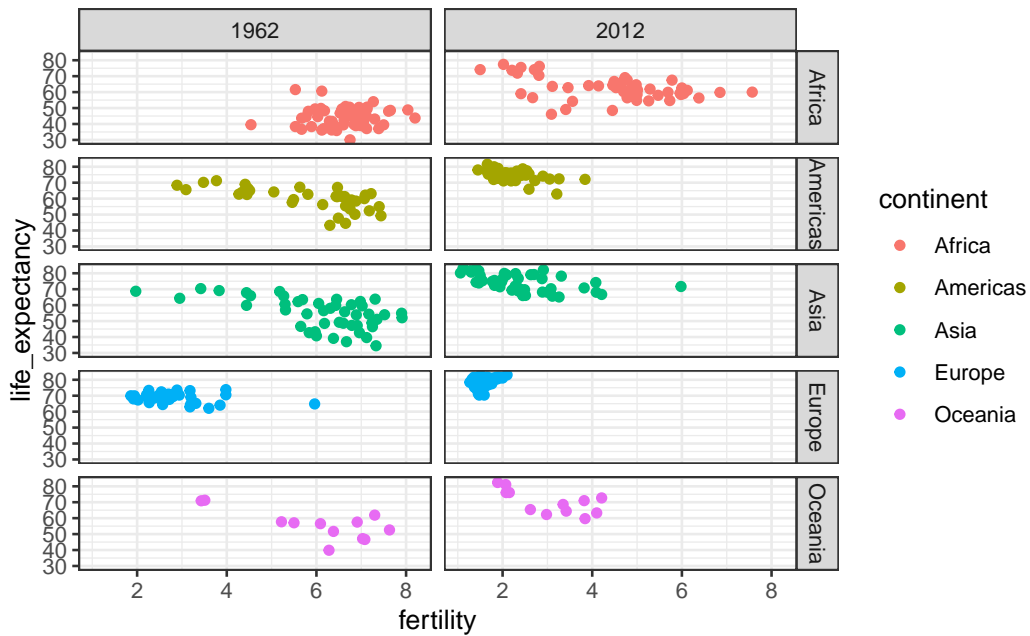
In 1962, “the West versus developing world” view was grounded in some reality. Is this still the case 50 years later?

### 9.3 Faceting

We could easily plot the 2012 data in the same way we did for 1962. To make comparisons, however, side by side plots are preferable. In **ggplot2**, we can achieve this by *faceting* variables: we stratify the data by some variable and make the same plot for each strata.

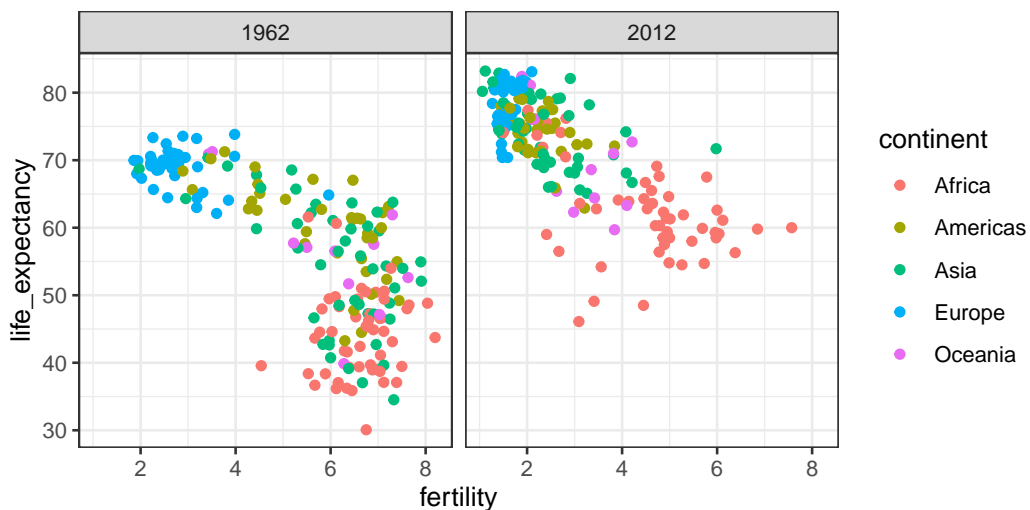
To achieve faceting, we add a layer with the function `facet_grid`, which automatically separates the plots. This function lets you facet by up to two variables using columns to represent one variable and rows to represent the other. The function expects the row and column variables to be separated by a `~`. Here is an example of a scatterplot with `facet_grid` added as the last layer:

```
filter(gapminder, year%in%c(1962, 2012)) %>%
 ggplot(aes(fertility, life_expectancy, col = continent)) +
 geom_point() +
 facet_grid(continent~year)
```



We see a plot for each continent/year pair. However, this is just an example and more than what we want, which is simply to compare 1962 and 2012. In this case, there is just one variable and we use `.` to let facet know that we are not using one of the variables:

```
filter(gapminder, year%in%c(1962, 2012)) %>%
 ggplot(aes(fertility, life_expectancy, col = continent)) +
 geom_point() +
 facet_grid(. ~ year)
```



This plot clearly shows that the majority of countries have moved from the *developing world*

cluster to the *western world* one. In 2012, the western versus developing world view no longer makes sense. This is particularly clear when comparing Europe to Asia, the latter of which includes several countries that have made great improvements.

### 9.3.1 facet\_wrap

To explore how this transformation happened through the years, we can make the plot for several years. For example, we can add 1970, 1980, 1990, and 2000. If we do this, we will not want all the plots on the same row, the default behavior of `facet_grid`, since they will become too thin to show the data. Instead, we will want to use multiple rows and columns. The function `facet_wrap` permits us to do this by automatically wrapping the series of plots so that each display has viewable dimensions:

```
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
 filter(year %in% years & continent %in% continents) %>%
 ggplot(aes(fertility, life_expectancy, col = continent)) +
 geom_point() +
 facet_wrap(~year)
```

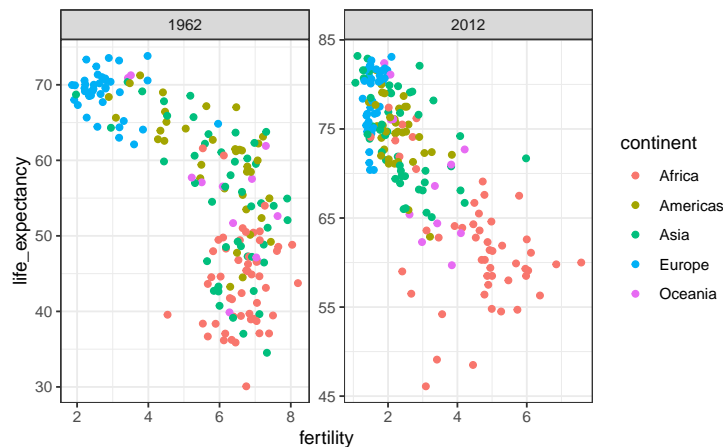


This plot clearly shows how most Asian countries have improved at a much faster rate than European ones.

### 9.3.2 Fixed scales for better comparisons

The default choice of the range of the axes is important. When not using `facet`, this range is determined by the data shown in the plot. When using `facet`, this range is determined by the data shown in all plots and therefore kept fixed across plots. This makes comparisons across plots much easier. For example, in the above plot, we can see that life expectancy has increased and the fertility has decreased across most countries. We see this because the cloud of points moves. This is not the case if we adjust the scales:

```
filter(gapminder, year%in%c(1962, 2012)) %>%
 ggplot(aes(fertility, life_expectancy, col = continent)) +
 geom_point() +
 facet_wrap(. ~ year, scales = "free")
```



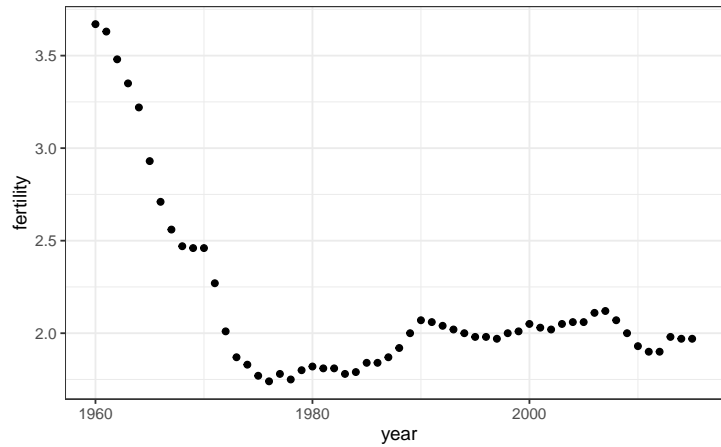
In the plot above, we have to pay special attention to the range to notice that the plot on the right has a larger life expectancy.

## 9.4 Time series plots

The visualizations above effectively illustrate that data no longer supports the western versus developing world view. Once we see these plots, new questions emerge. For example, which countries are improving more and which ones less? Was the improvement constant during the last 50 years or was it more accelerated during certain periods? For a closer look that may help answer these questions, we introduce *time series plots*.

Time series plots have time in the x-axis and an outcome or measurement of interest on the y-axis. For example, here is a trend plot of United States fertility rates:

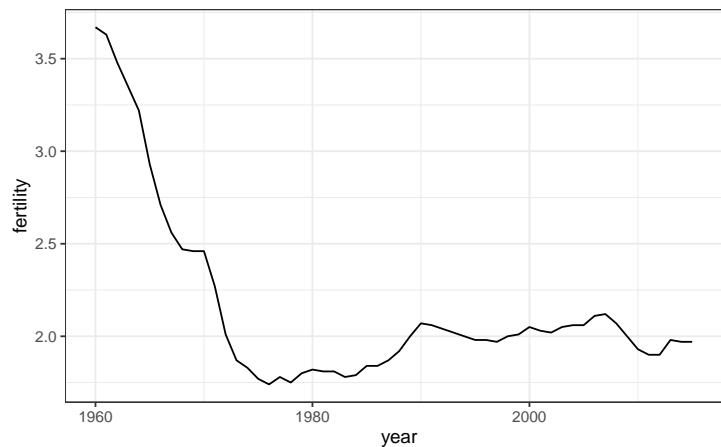
```
gapminder %>%
 filter(country == "United States") %>%
 ggplot(aes(year, fertility)) +
 geom_point()
```



We see that the trend is not linear at all. Instead there is sharp drop during the 1960s and 1970s to below 2. Then the trend comes back to 2 and stabilizes during the 1990s.

When the points are regularly and densely spaced, as they are here, we create curves by joining the points with lines, to convey that these data are from a single series, here a country. To do this, we use the `geom_line` function instead of `geom_point`.

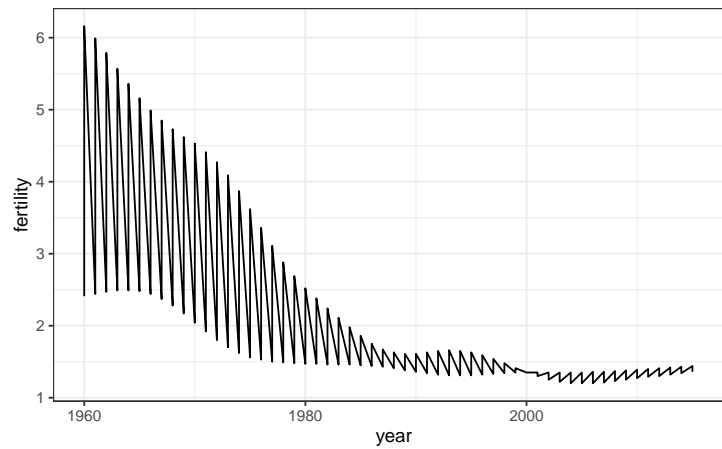
```
gapminder %>%
 filter(country == "United States") %>%
 ggplot(aes(year, fertility)) +
 geom_line()
```



This is particularly helpful when we look at two countries. If we subset the data to include two countries, one from Europe and one from Asia, then adapt the code above:

```
countries <- c("South Korea", "Germany")

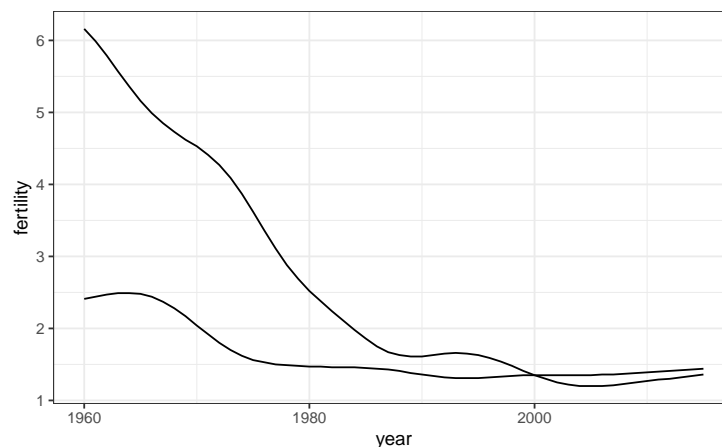
gapminder %>% filter(country %in% countries) %>%
 ggplot(aes(year, fertility)) +
 geom_line()
```



Unfortunately, this is **not** the plot that we want. Rather than a line for each country, the points for both countries are joined. This is actually expected since we have not told `ggplot` anything about wanting two separate lines. To let `ggplot` know that there are two curves that need to be made separately, we assign each point to a `group`, one for each country:

```
countries <- c("South Korea", "Germany")

gapminder %>% filter(country %in% countries & !is.na(fertility)) %>%
 ggplot(aes(year, fertility, group = country)) +
 geom_line()
```

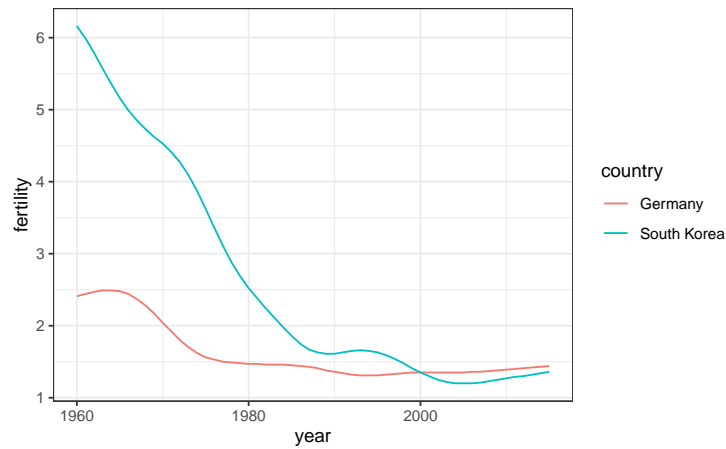


But which line goes with which country? We can assign colors to make this distinction. A useful side-effect of using the `color` argument to assign different colors to the different countries is that the data is automatically grouped:

```
countries <- c("South Korea", "Germany")

gapminder %>% filter(country %in% countries & !is.na(fertility)) %>%
 ggplot(aes(year, fertility, col = country)) +
 geom_line()
```





The plot clearly shows how South Korea's fertility rate dropped drastically during the 1960s and 1970s, and by 1990 had a similar rate to that of Germany.

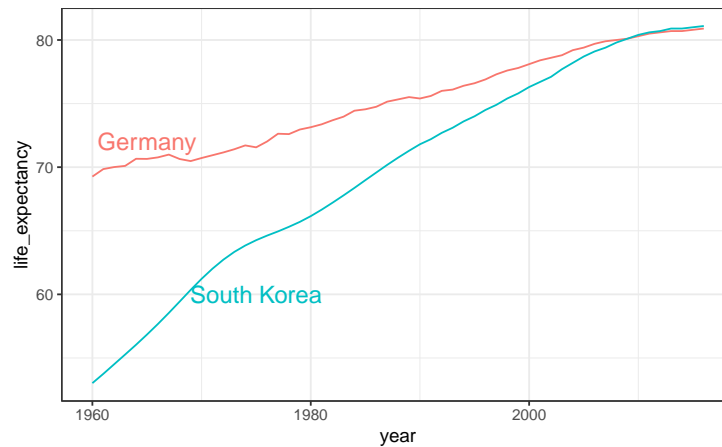
#### 9.4.1 Labels instead of legends

For trend plots we recommend labeling the lines rather than using legends since the viewer can quickly see which line is which country. This suggestion actually applies to most plots: labeling is usually preferred over legends.

We demonstrate how we can do this using the life expectancy data. We define a data table with the label locations and then use a second mapping just for these labels:

```
labels <- data.frame(country = countries, x = c(1975,1965), y = c(60,72))
```

```
gapminder %>%
 filter(country %in% countries) %>%
 ggplot(aes(year, life_expectancy, col = country)) +
 geom_line() +
 geom_text(data = labels, aes(x, y, label = country), size = 5) +
 theme(legend.position = "none")
```



The plot clearly shows how an improvement in life expectancy followed the drops in fertility rates. In 1960, Germans lived 15 years longer than South Koreans, although by 2010 the gap is completely closed. It exemplifies the improvement that many non-western countries have achieved in the last 40 years.

---

## 9.5 Data transformations

We now shift our attention to the second question related to the commonly held notion that wealth distribution across the world has become worse during the last decades. When general audiences are asked if poor countries have become poorer and rich countries become richer, the majority answers yes. By using stratification, histograms, smooth densities, and boxplots, we will be able to understand if this is in fact the case. First we learn how transformations can sometimes help provide more informative summaries and plots.

The `gapminder` data table includes a column with the countries' gross domestic product (GDP). GDP measures the market value of goods and services produced by a country in a year. The GDP per person is often used as a rough summary of a country's wealth. Here we divide this quantity by 365 to obtain the more interpretable measure *dollars per day*. Using current US dollars as a unit, a person surviving on an income of less than \$2 a day is defined to be living in *absolute poverty*. We add this variable to the data table:

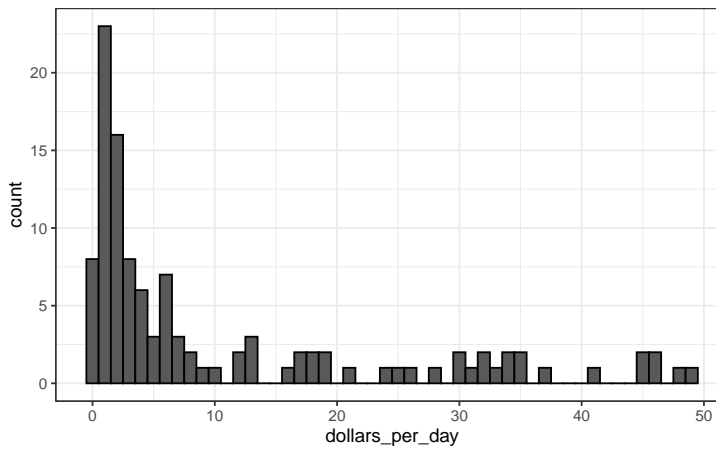
```
gapminder <- gapminder %>% mutate(dollars_per_day = gdp/population/365)
```

The GDP values are adjusted for inflation and represent current US dollars, so these values are meant to be comparable across the years. Of course, these are country averages and within each country there is much variability. All the graphs and insights described below relate to country averages and not to individuals.

### 9.5.1 Log transformation

Here is a histogram of per day incomes from 1970:

```
past_year <- 1970
gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 ggplot(aes(dollars_per_day)) +
 geom_histogram(binwidth = 1, color = "black")
```



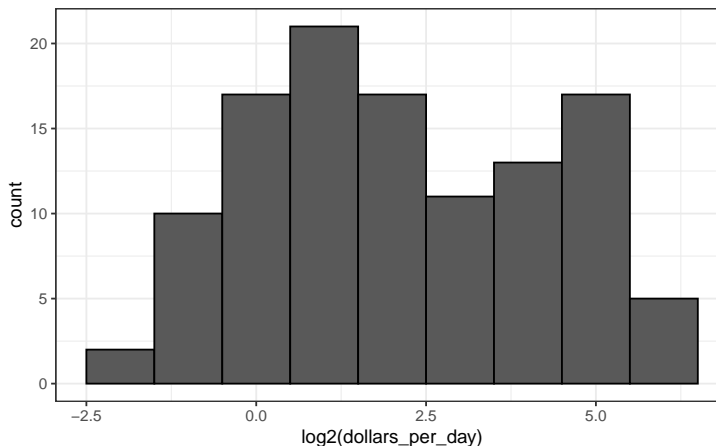
We use the `color = "black"` argument to draw a boundary and clearly distinguish the bins.

In this plot, we see that for the majority of countries, averages are below \$10 a day. However, the majority of the x-axis is dedicated to the 35 countries with averages above \$10. So the plot is not very informative about countries with values below \$10 a day.

It might be more informative to quickly be able to see how many countries have average daily incomes of about \$1 (extremely poor), \$2 (very poor), \$4 (poor), \$8 (middle), \$16 (well off), \$32 (rich), \$64 (very rich) per day. These changes are multiplicative and log transformations convert multiplicative changes into additive ones: when using base 2, a doubling of a value turns into an increase by 1.

Here is the distribution if we apply a log base 2 transform:

```
gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 ggplot(aes(log2(dollars_per_day))) +
 geom_histogram(binwidth = 1, color = "black")
```



In a way this provides a *close-up* of the mid to lower income countries.

### 9.5.2 Which base?

In the case above, we used base 2 in the log transformations. Other common choices are base  $e$  (the natural log) and base 10.

In general, we do not recommend using the natural log for data exploration and visualization. This is because while  $2^2, 2^3, 2^4, \dots$  or  $10^2, 10^3, \dots$  are easy to compute in our heads, the same is not true for  $e^2, e^3, \dots$ , so the scale is not intuitive or easy to interpret.

In the dollars per day example, we used base 2 instead of base 10 because the resulting range is easier to interpret. The range of the values being plotted is 0.327, 48.885.

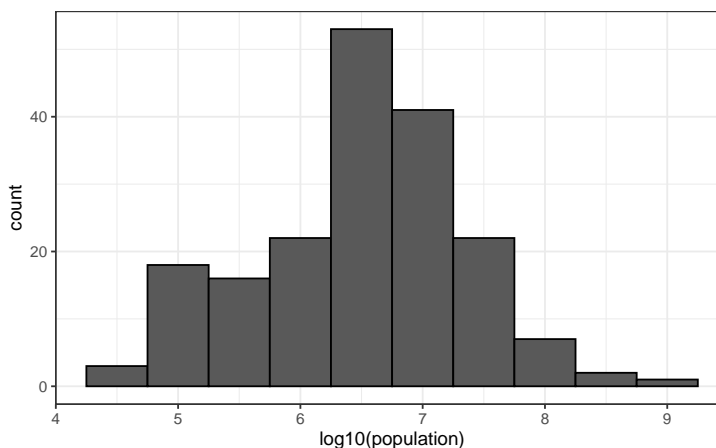
In base 10, this turns into a range that includes very few integers: just 0 and 1. With base two, our range includes -2, -1, 0, 1, 2, 3, 4, and 5. It is easier to compute  $2^x$  and  $10^x$  when  $x$  is an integer and between -10 and 10, so we prefer to have smaller integers in the scale. Another consequence of a limited range is that choosing the binwidth is more challenging. With log base 2, we know that a binwidth of 1 will translate to a bin with range  $x$  to  $2x$ .

For an example in which base 10 makes more sense, consider population sizes. A log base 10 is preferable since the range for these is:

```
filter(gapminder, year == past_year) %>%
 summarize(min = min(population), max = max(population))
#> min max
#> 1 46075 8.09e+08
```

Here is the histogram of the transformed values:

```
gapminder %>%
 filter(year == past_year) %>%
 ggplot(aes(log10(population))) +
 geom_histogram(binwidth = 0.5, color = "black")
```



In the above, we quickly see that country populations range between ten thousand and ten billion.

### 9.5.3 Transform the values or the scale?

There are two ways we can use log transformations in plots. We can log the values before plotting them or use log scales in the axes. Both approaches are useful and have different strengths. If we log the data, we can more easily interpret intermediate values in the scale. For example, if we see:

----1----x----2-----3----

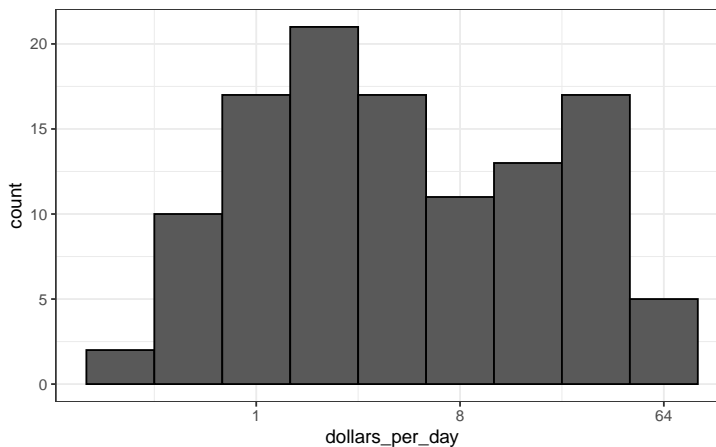
for log transformed data, we know that the value of  $x$  is about 1.5. If the scales are logged:

----1----x----10-----100---

then, to determine  $x$ , we need to compute  $10^{1.5}$ , which is not easy to do in our heads. The advantage of using logged scales is that we see the original values on the axes. However, the advantage of showing logged scales is that the original values are displayed in the plot, which are easier to interpret. For example, we would see “32 dollars a day” instead of “5 log base 2 dollars a day”.

As we learned earlier, if we want to scale the axis with logs, we can use the `scale_x_continuous` function. Instead of logging the values first, we apply this layer:

```
gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 ggplot(aes(dollars_per_day)) +
 geom_histogram(binwidth = 1, color = "black") +
 scale_x_continuous(trans = "log2")
```



Note that the log base 10 transformation has its own function: `scale_x_log10()`, but currently base 2 does not, although we could easily define our own.

There are other transformations available through the `trans` argument. As we learn later on, the square root (`sqrt`) transformation is useful when considering counts. The logistic transformation (`logit`) is useful when plotting proportions between 0 and 1. The `reverse` transformation is useful when we want smaller values to be on the right or on top.

## 9.6 Visualizing multimodal distributions

In the histogram above we see two *bumps*: one at about 4 and another at about 32. In statistics these bumps are sometimes referred to as *modes*. The mode of a distribution is the value with the highest frequency. The mode of the normal distribution is the average. When a distribution, like the one above, doesn't monotonically decrease from the mode, we call the locations where it goes up and down again *local modes* and say that the distribution has *multiple modes*.

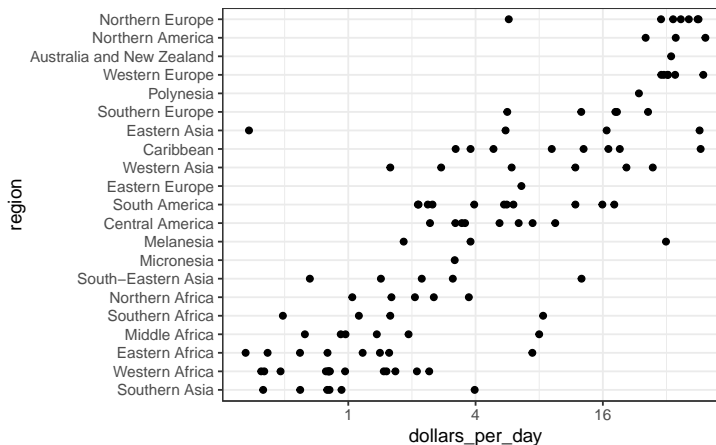
The histogram above suggests that the 1970 country income distribution has two modes: one at about 2 dollars per day (1 in the log 2 scale) and another at about 32 dollars per day (5 in the log 2 scale). This *bimodality* is consistent with a dichotomous world made up of countries with average incomes less than \$8 (3 in the log 2 scale) a day and countries above that.

## 9.7 Comparing multiple distributions with boxplots and ridge plots

A histogram showed us that the 1970 income distribution values show a dichotomy. However, the histogram does not show us if the two groups of countries are *west* versus the *developing* world.

Let's start by quickly examining the data by region. We reorder the regions by the median value and use a log scale.

```
gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
 ggplot(aes(dollars_per_day, region)) +
 geom_point() +
 scale_x_continuous(trans = "log2")
```



We can already see that there is indeed a “west versus the rest” dichotomy: we see two clear groups, with the rich group composed of North America, Northern and Western Europe, New Zealand and Australia. We define groups based on this observation:

```
gapminder <- gapminder %>%
 mutate(group = case_when(
 region %in% c("Western Europe", "Northern Europe", "Southern Europe",
 "Northern America",
 "Australia and New Zealand") ~ "West",
 region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
 region %in% c("Caribbean", "Central America",
 "South America") ~ "Latin America",
 continent == "Africa" &
 region != "Northern Africa" ~ "Sub-Saharan",
 TRUE ~ "Others"))
```

We turn this group variable into a factor to control the order of the levels:

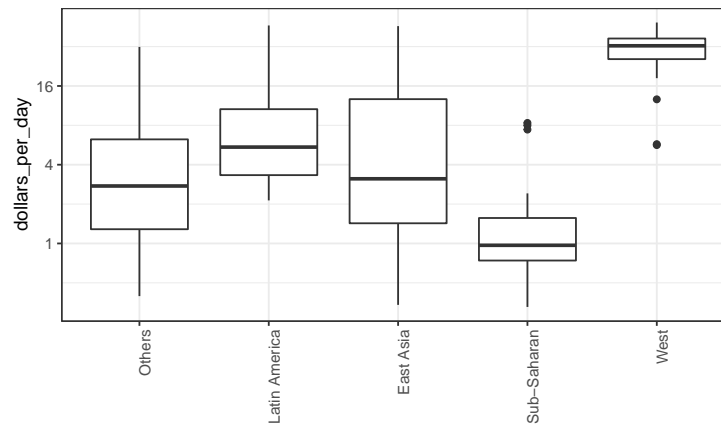
```
gapminder <- gapminder %>%
 mutate(group = factor(group, levels = c("Others", "Latin America",
 "East Asia", "Sub-Saharan",
 "West")))
```

In the next section we demonstrate how to visualize and compare distributions across groups.

### 9.7.1 Boxplots

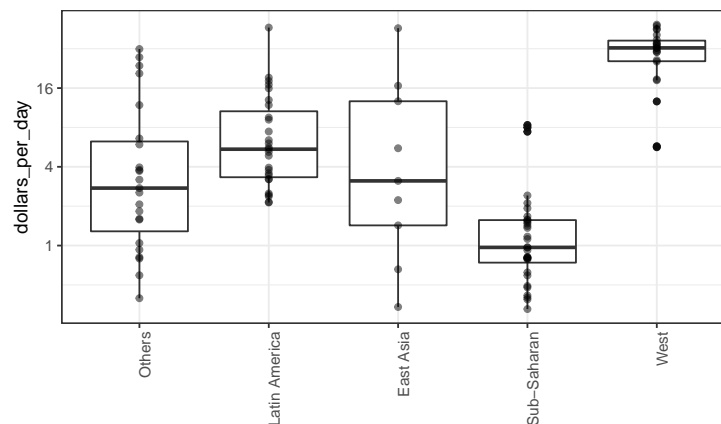
The exploratory data analysis above has revealed two characteristics about average income distribution in 1970. Using a histogram, we found a bimodal distribution with the modes relating to poor and rich countries. We now want to compare the distribution across these five groups to confirm the “west versus the rest” dichotomy. The number of points in each category is large enough that a summary plot may be useful. We could generate five histograms or five density plots, but it may be more practical to have all the visual summaries in one plot. We therefore start by stacking boxplots next to each other. Note that we add the layer `theme(axis.text.x = element_text(angle = 90, hjust = 1))` to turn the group labels vertical, since they do not fit if we show them horizontally, and remove the axis label to make space.

```
p <- gapminder %>%
 filter(year == past_year & !is.na(gdp)) %>%
 ggplot(aes(group, dollars_per_day)) +
 geom_boxplot() +
 scale_y_continuous(trans = "log2") +
 xlab("") +
 theme(axis.text.x = element_text(angle = 90, hjust = 1))
p
```



Boxplots have the limitation that by summarizing the data into five numbers, we might miss important characteristics of the data. One way to avoid this is by showing the data.

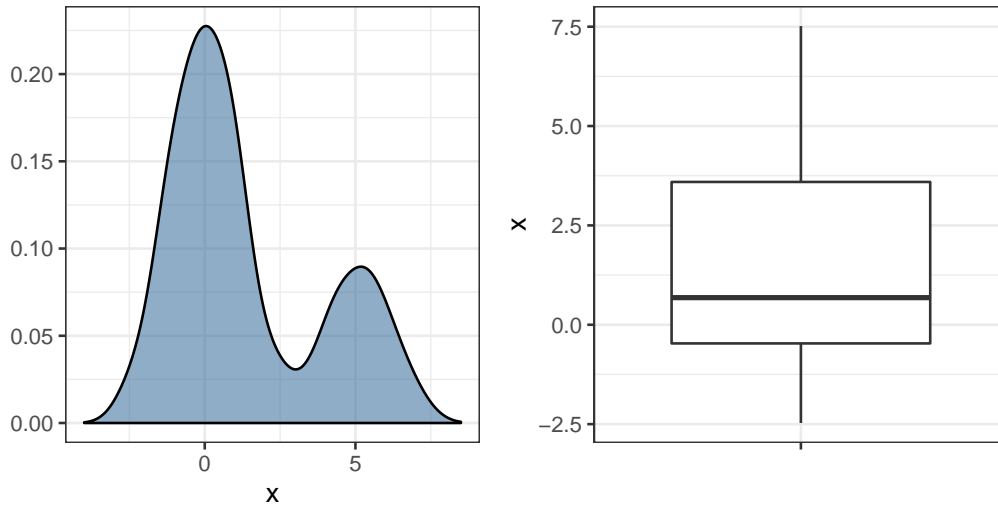
```
p + geom_point(alpha = 0.5)
```



### 9.7.2 Ridge plots

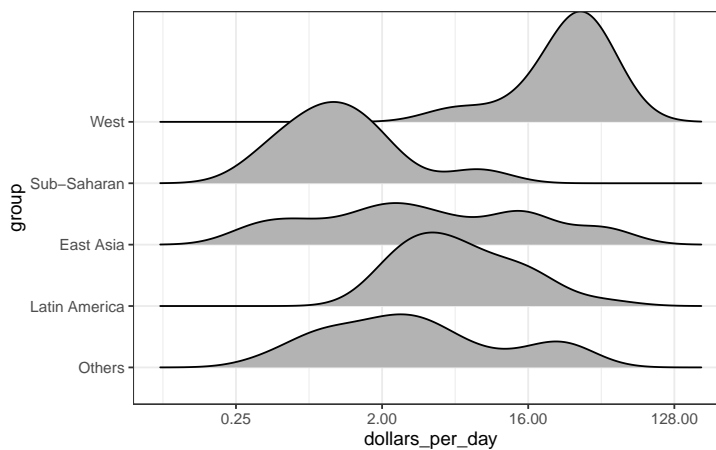
Showing each individual point does not always reveal important characteristics of the distribution. Although not the case here, when the number of data points is so large that there is over-plotting, showing the data can be counterproductive. Boxplots help with this by providing a five-number summary, but this has limitations too. For example, boxplots will not permit us to discover bimodal distributions. To see this, note that the two plots below are summarizing the same dataset:





In cases in which we are concerned that the boxplot summary is too simplistic, we can show stacked smooth densities or histograms. We refer to these as *ridge plots*. Because we are used to visualizing densities with values in the x-axis, we stack them vertically. Also, because more space is needed in this approach, it is convenient to overlay them. The package **ggrridges** provides a convenient function for doing this. Here is the income data shown above with boxplots but with a *ridge plot*.

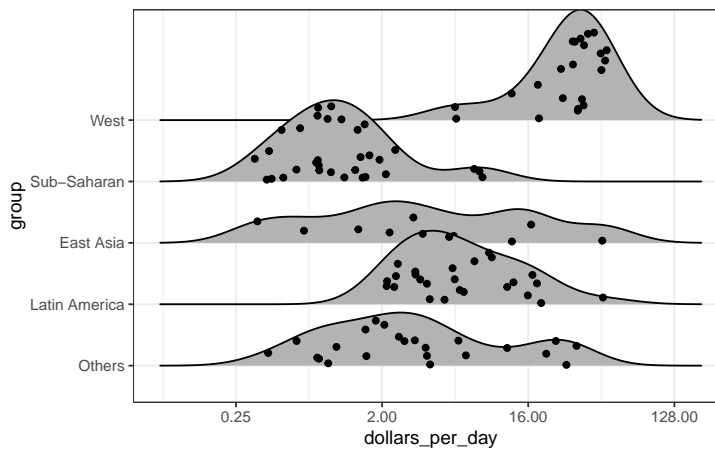
```
library(ggrridges)
p <- gapminder %>%
 filter(year == past_year & !is.na(dollars_per_day)) %>%
 ggplot(aes(dollars_per_day, group)) +
 scale_x_continuous(trans = "log2")
p + geom_density_ridges()
```



Note that we have to invert the **x** and **y** used for the boxplot. A useful **geom\_density\_ridges** parameter is **scale**, which lets you determine the amount of overlap, with **scale = 1** meaning no overlap and larger values resulting in more overlap.

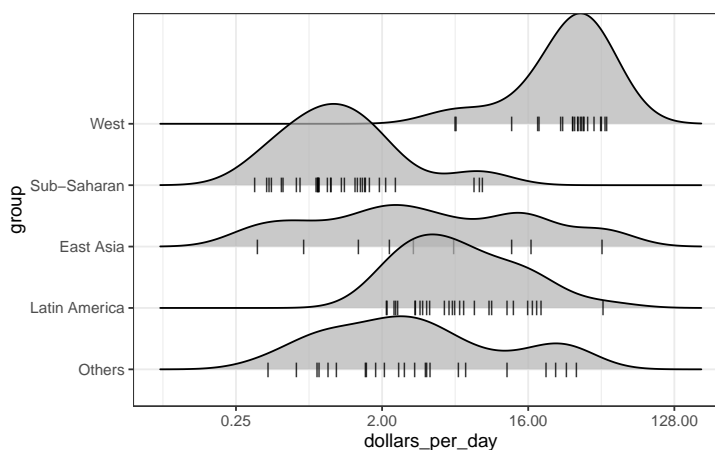
If the number of data points is small enough, we can add them to the ridge plot using the following code:

```
p + geom_density_ridges(jittered_points = TRUE)
```



By default, the height of the points is jittered and should not be interpreted in any way. To show data points, but without using jitter we can use the following code to add what is referred to as a *rug representation* of the data.

```
p + geom_density_ridges(jittered_points = TRUE,
 position = position_points_jitter(height = 0),
 point_shape = '|', point_size = 3,
 point_alpha = 1, alpha = 0.7)
```



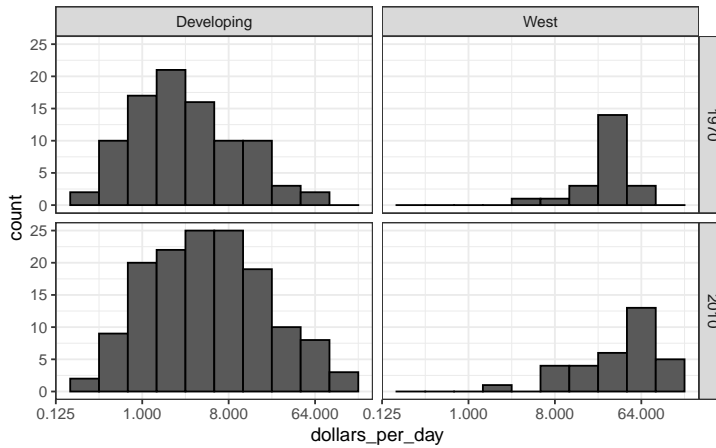
### 9.7.3 Example: 1970 versus 2010 income distributions

Data exploration clearly shows that in 1970 there was a “west versus the rest” dichotomy. But does this dichotomy persist? Let’s use `facet_grid` see how the distributions have changed. To start, we will focus on two groups: the west and the rest. We make four histograms.

```

past_year <- 1970
present_year <- 2010
years <- c(past_year, present_year)
gapminder %>%
 filter(year %in% years & !is.na(gdp)) %>%
 mutate(west = ifelse(group == "West", "West", "Developing")) %>%
 ggplot(aes(dollars_per_day)) +
 geom_histogram(binwidth = 1, color = "black") +
 scale_x_continuous(trans = "log2") +
 facet_grid(year ~ west)

```



Before we interpret the findings of this plot, we notice that there are more countries represented in the 2010 histograms than in 1970: the total counts are larger. One reason for this is that several countries were founded after 1970. For example, the Soviet Union divided into several countries during the 1990s. Another reason is that data was available for more countries in 2010.

We remake the plots using only countries with data available for both years. In the data wrangling part of this book, we will learn **tidyverse** tools that permit us to write efficient code for this, but here we can use simple code using the **intersect** function:

```

country_list_1 <- gapminder %>%
 filter(year == past_year & !is.na(dollars_per_day)) %>%
 pull(country)

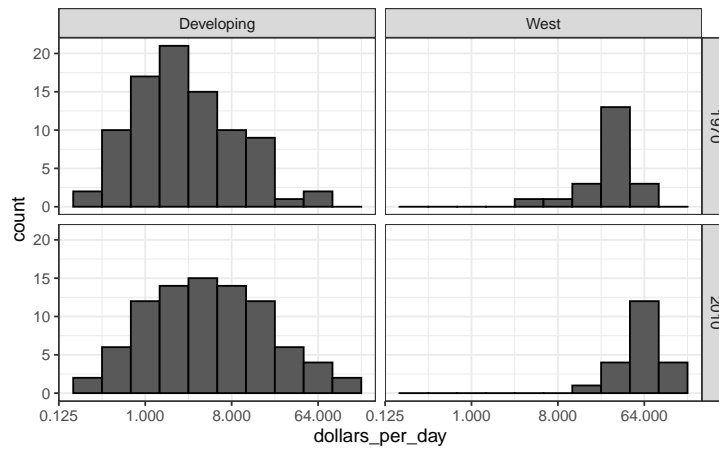
country_list_2 <- gapminder %>%
 filter(year == present_year & !is.na(dollars_per_day)) %>%
 pull(country)

country_list <- intersect(country_list_1, country_list_2)

```

These 108 account for 86% of the world population, so this subset should be representative.

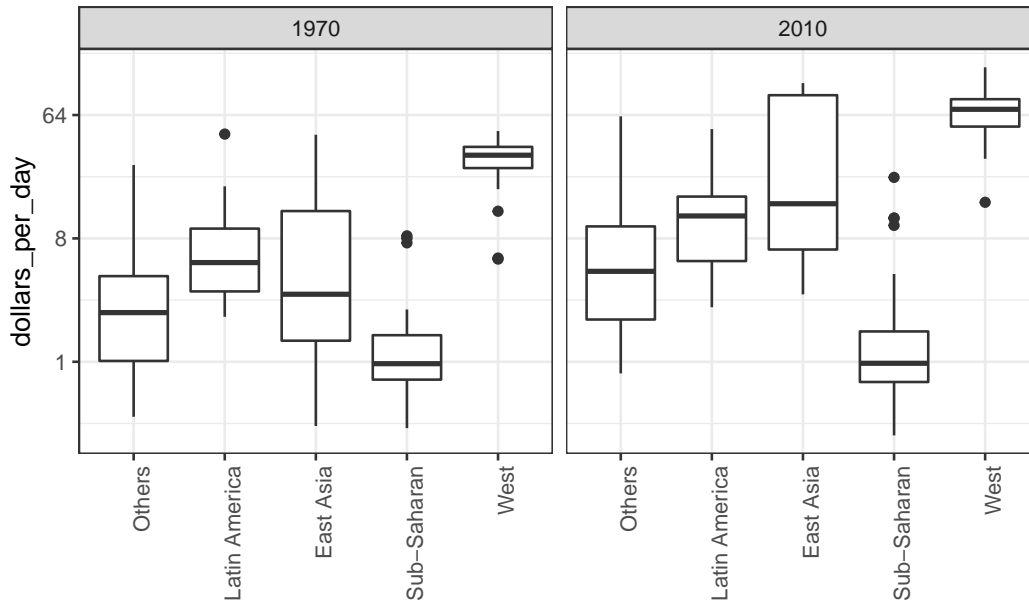
Let's remake the plot, but only for this subset by simply adding `country %in% country_list` to the filter function:



We now see that the rich countries have become a bit richer, but percentage-wise, the poor countries appear to have improved more. In particular, we see that the proportion of *developing* countries earning more than \$16 a day increased substantially.

To see which specific regions improved the most, we can remake the boxplots we made above, but now adding the year 2010 and then using facet to compare the two years.

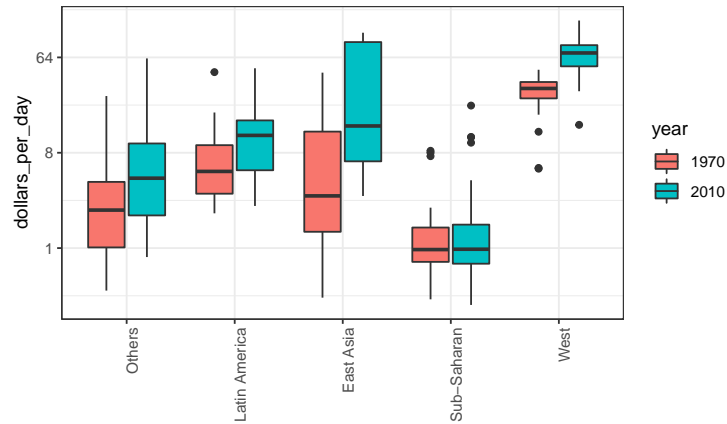
```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 ggplot(aes(group, dollars_per_day)) +
 geom_boxplot() +
 theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
 scale_y_continuous(trans = "log2") +
 xlab("") +
 facet_grid(. ~ year)
```



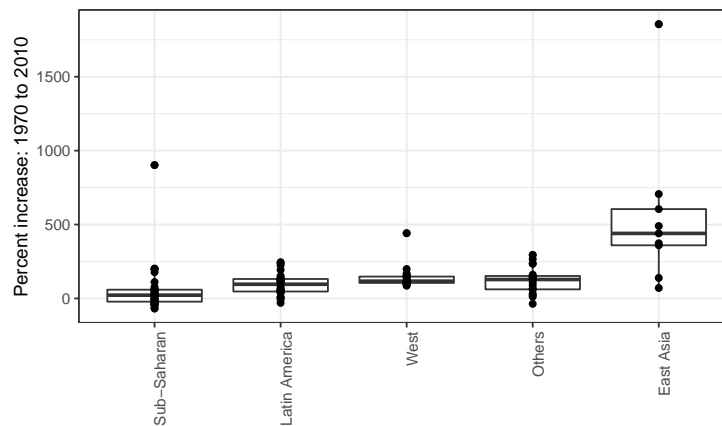
Here, we pause to introduce another powerful **ggplot2** feature. Because we want to compare each region before and after, it would be convenient to have the 1970 boxplot next to the 2010 boxplot for each region. In general, comparisons are easier when data are plotted next to each other.

So instead of faceting, we keep the data from each year together and ask to color (or fill) them depending on the year. Note that groups are automatically separated by year and each pair of boxplots drawn next to each other. Because year is a number, we turn it into a factor since **ggplot2** automatically assigns a color to each category of a factor. Note that we have to convert the year columns from numeric to factor.

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 mutate(year = factor(year)) %>%
 ggplot(aes(group, dollars_per_day, fill = year)) +
 geom_boxplot() +
 theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
 scale_y_continuous(trans = "log2") +
 xlab("")
```



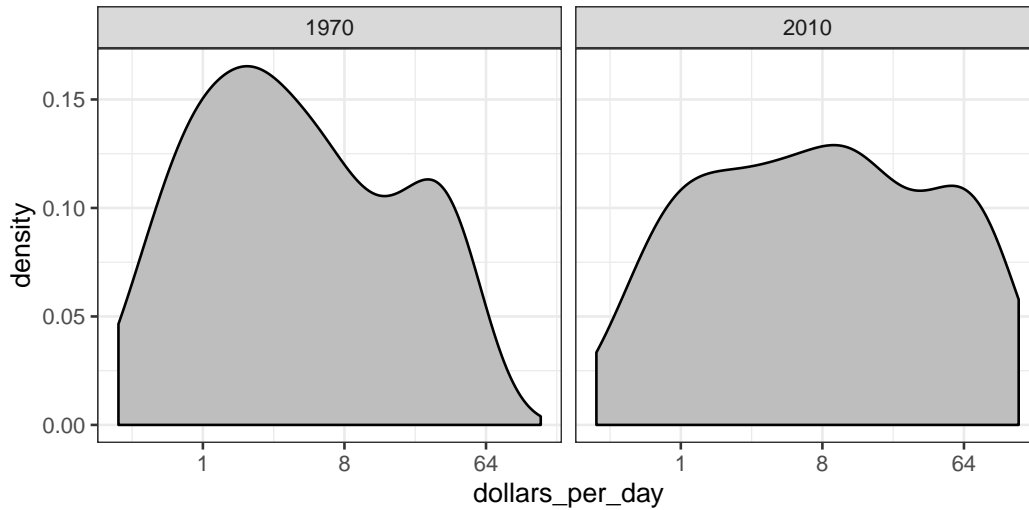
Finally, we point out that if what we are most interested in is comparing before and after values, it might make more sense to plot the percentage increases. We are still not ready to learn to code this, but here is what the plot would look like:



The previous data exploration suggested that the income gap between rich and poor countries has narrowed considerably during the last 40 years. We used a series of histograms and boxplots to see this. We suggest a succinct way to convey this message with just one plot.

Let's start by noting that density plots for income distribution in 1970 and 2010 deliver the message that the gap is closing:

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 ggplot(aes(dollars_per_day)) +
 geom_density(fill = "grey") +
 scale_x_continuous(trans = "log2") +
 facet_grid(. ~ year)
```



In the 1970 plot, we see two clear modes: poor and rich countries. In 2010, it appears that some of the poor countries have shifted towards the right, closing the gap.

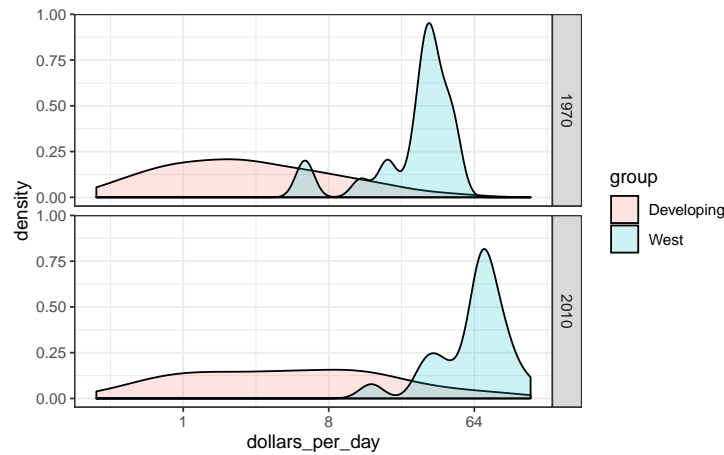
The next message we need to convey is that the reason for this change in distribution is that several poor countries became richer, rather than some rich countries becoming poorer. To do this, we can assign a color to the groups we identified during data exploration.

However, we first need to learn how to make these smooth densities in a way that preserves information on the number of countries in each group. To understand why we need this, note the discrepancy in the size of each group:

| Developing | West |
|------------|------|
| 87         | 21   |

But when we overlay two densities, the default is to have the area represented by each distribution add up to 1, regardless of the size of each group:

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 mutate(group = ifelse(group == "West", "West", "Developing")) %>%
 ggplot(aes(dollars_per_day, fill = group)) +
 scale_x_continuous(trans = "log2") +
 geom_density(alpha = 0.2) +
 facet_grid(year ~ .)
```



This makes it appear as if there are the same number of countries in each group. To change this, we will need to learn to access computed variables with `geom_density` function.

#### 9.7.4 Accessing computed variables

To have the areas of these densities be proportional to the size of the groups, we can simply multiply the y-axis values by the size of the group. From the `geom_density` help file, we see that the functions compute a variable called `count` that does exactly this. We want this variable to be on the y-axis rather than the density.

In **ggplot2**, we access these variables by surrounding the name with two dots. We will therefore use the following mapping:

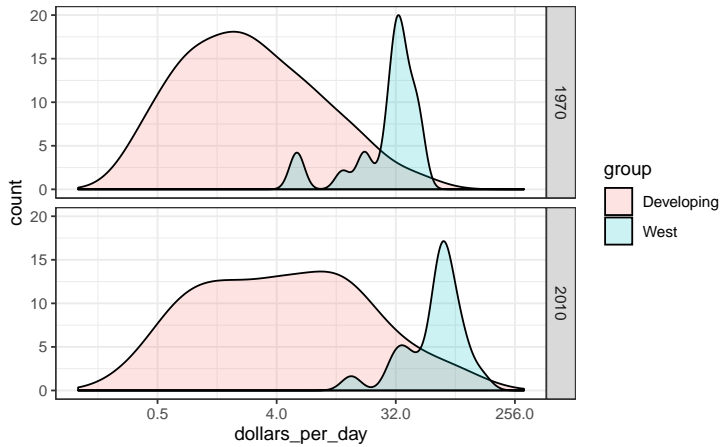
```
aes(x = dollars_per_day, y = ..count..)
```

We can now create the desired plot by simply changing the mapping in the previous code chunk. We will also expand the limits of the x-axis.

```
p <- gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 mutate(group = ifelse(group == "West", "West", "Developing")) %>%
 ggplot(aes(dollars_per_day, y = ..count.., fill = group)) +
 scale_x_continuous(trans = "log2", limit = c(0.125, 300))

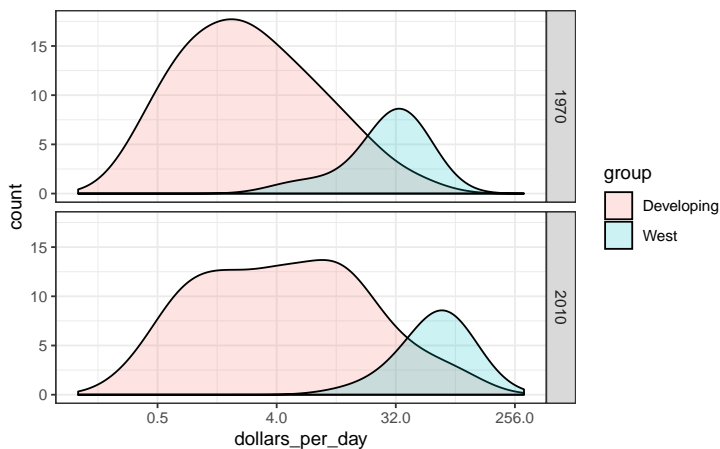
p + geom_density(alpha = 0.2) +
 facet_grid(year ~ .)
```





If we want the densities to be smoother, we use the `bw` argument so that the same bandwidth is used in each density. We selected 0.75 after trying out several values.

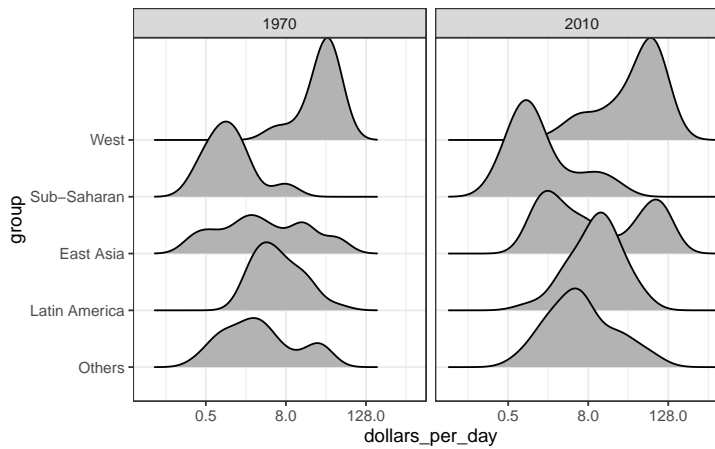
```
p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~ .)
```



This plot now shows what is happening very clearly. The developing world distribution is changing. A third mode appears consisting of the countries that most narrowed the gap.

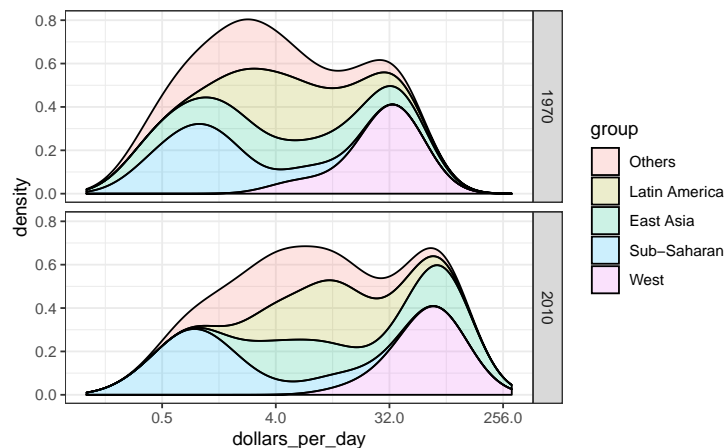
To visualize if any of the groups defined above are driving this we can quickly make a ridge plot:

```
gapminder %>%
 filter(year %in% years & !is.na(dollars_per_day)) %>%
 ggplot(aes(dollars_per_day, group)) +
 scale_x_continuous(trans = "log2") +
 geom_density_ridges(adjust = 1.5) +
 facet_grid(. ~ year)
```



Another way to achieve this is by stacking the densities on top of each other:

```
gapminder %>%
 filter(year %in% years & country %in% country_list) %>%
 group_by(year) %>%
 mutate(weight = population/sum(population)*2) %>%
 ungroup() %>%
 ggplot(aes(dollars_per_day, fill = group)) +
 scale_x_continuous(trans = "log2", limit = c(0.125, 300)) +
 geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
 facet_grid(year ~ .)
```

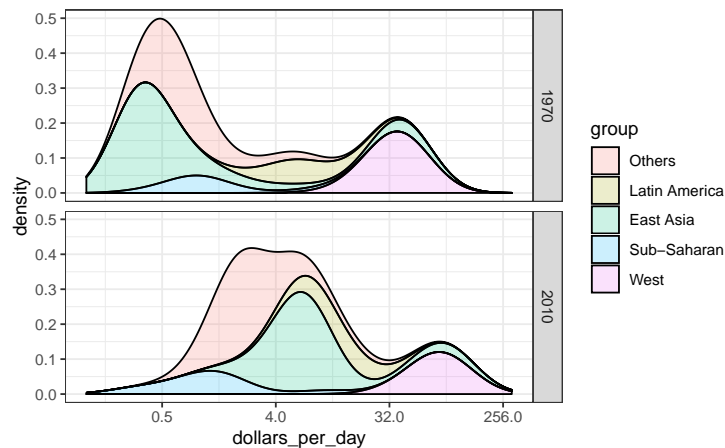


Here we can clearly see how the distributions for East Asia, Latin America, and others shift markedly to the right. While Sub-Saharan Africa remains stagnant.

Notice that we order the levels of the group so that the West's density is plotted first, then Sub-Saharan Africa. Having the two extremes plotted first allows us to see the remaining bimodality better.

### 9.7.5 Weighted densities

As a final point, we note that these distributions weigh every country the same. So if most of the population is improving, but living in a very large country, such as China, we might not appreciate this. We can actually weight the smooth densities using the `weight` mapping argument. The plot then looks like this:



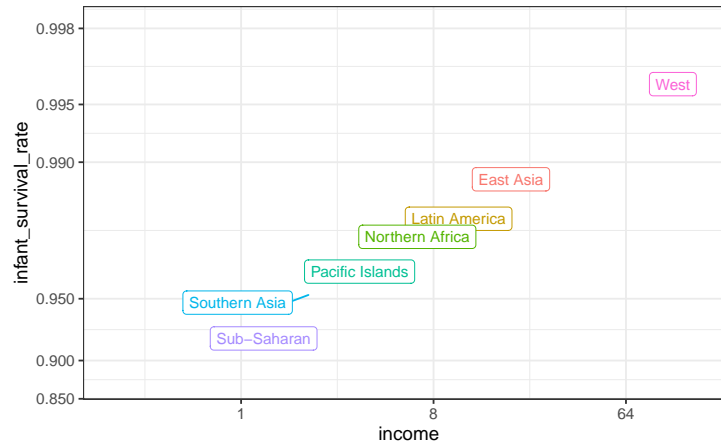
This particular figure shows very clearly how the income distribution gap is closing with most of the poor remaining in Sub-Saharan Africa.

---

## 9.8 The ecological fallacy and importance of showing the data

Throughout this section, we have been comparing regions of the world. We have seen that, on average, some regions do better than others. In this section, we focus on describing the importance of variability within the groups when examining the relationship between a country's infant mortality rates and average income.

We define a few more regions and compare the averages across regions:



The relationship between these two variables is almost perfectly linear and the graph shows a dramatic difference. While in the West less than 0.5% of infants die, in Sub-Saharan Africa the rate is higher than 6%!

Note that the plot uses a new transformation, the logistic transformation.

### 9.8.1 Logistic transformation

The logistic or logit transformation for a proportion or rate  $p$  is defined as:

$$f(p) = \log\left(\frac{p}{1-p}\right)$$

When  $p$  is a proportion or probability, the quantity that is being logged,  $p/(1-p)$ , is called the *odds*. In this case  $p$  is the proportion of infants that survived. The odds tell us how many more infants are expected to survive than to die. The log transformation makes this symmetric. If the rates are the same, then the log odds is 0. Fold increases or decreases turn into positive and negative increments, respectively.

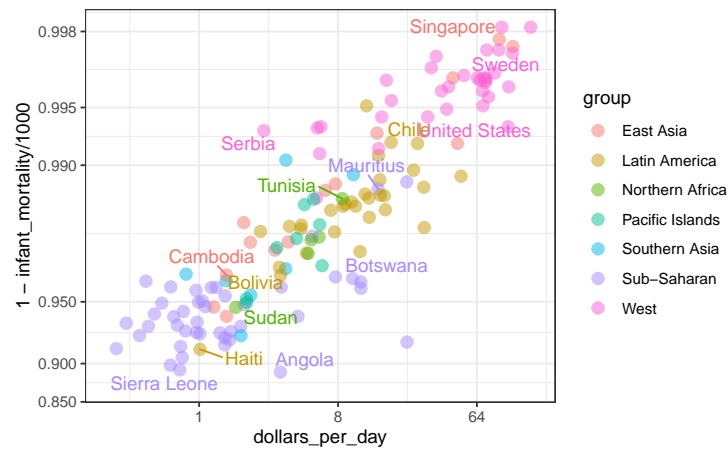
This scale is useful when we want to highlight differences near 0 or 1. For survival rates this is important because a survival rate of 90% is unacceptable, while a survival of 99% is relatively good. We would much prefer a survival rate closer to 99.9%. We want our scale to highlight these difference and the logit does this. Note that  $99.9/0.1$  is about 10 times bigger than  $99/1$  which is about 10 times larger than  $90/10$ . By using the log, these fold changes turn into constant increases.

### 9.8.2 Show the data

Now, back to our plot. Based on the plot above, do we conclude that a country with a low income is destined to have low survival rate? Do we conclude that survival rates in Sub-Saharan Africa are all lower than in Southern Asia, which in turn are lower than in the Pacific Islands, and so on?

Jumping to this conclusion based on a plot showing averages is referred to as the *ecological fallacy*. The almost perfect relationship between survival rates and income is only observed

for the averages at the region level. Once we show all the data, we see a somewhat more complicated story:



Specifically, we see that there is a large amount of variability. We see that countries from the same regions can be quite different and that countries with the same income can have different survival rates. For example, while on average Sub-Saharan Africa had the worse health and economic outcomes, there is wide variability within that group. Mauritius and Botswana are doing better than Angola and Sierra Leone, with Mauritius comparable to Western countries.

# 10

---

## *Data visualization principles*

---

We have already provided some rules to follow as we created plots for our examples. Here, we aim to provide some general principles we can use as a guide for effective data visualization. Much of this section is based on a talk by Karl Broman<sup>1</sup> titled “Creating Effective Figures and Tables”<sup>2</sup> and includes some of the figures which were made with code that Karl makes available on his GitHub repository<sup>3</sup>, as well as class notes from Peter Aldhous’ Introduction to Data Visualization course<sup>4</sup>. Following Karl’s approach, we show some examples of plot styles we should avoid, explain how to improve them, and use these as motivation for a list of principles. We compare and contrast plots that follow these principles to those that don’t.

The principles are mostly based on research related to how humans detect patterns and make visual comparisons. The preferred approaches are those that best fit the way our brains process visual information. When deciding on a visualization approach, it is also important to keep our goal in mind. We may be comparing a viewable number of quantities, describing distributions for categories or numeric values, comparing the data from two groups, or describing the relationship between two variables. As a final note, we want to emphasize that for a data scientist it is important to adapt and optimize graphs to the audience. For example, an exploratory plot made for ourselves will be different than a chart intended to communicate a finding to a general audience.

We will be using these libraries:

```
library(tidyverse)
library(dslabs)
library(gridExtra)
```

---

### 10.1 Encoding data using visual cues

We start by describing some principles for encoding data. There are several approaches at our disposal including position, aligned lengths, angles, area, brightness, and color hue.

To illustrate how some of these strategies compare, let’s suppose we want to report the results from two hypothetical polls regarding browser preference taken in 2000 and then 2015. For each year, we are simply comparing four quantities – the four percentages. A widely used graphical representation of percentages, popularized by Microsoft Excel, is the pie chart:

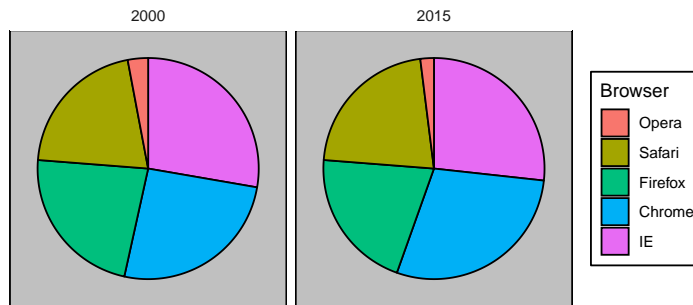
---

<sup>1</sup><http://kbroman.org/>

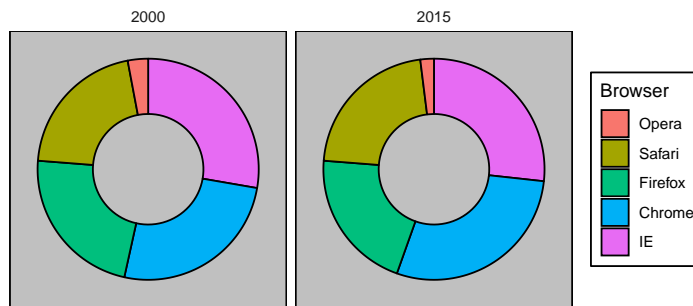
<sup>2</sup><https://www.biostat.wisc.edu/~kbroman/presentations/graphs2017.pdf>

<sup>3</sup>[https://github.com/kbroman/Talk\\_Graphs](https://github.com/kbroman/Talk_Graphs)

<sup>4</sup><http://paldhous.github.io/ucb/2016/dataviz/index.html>



Here we are representing quantities with both areas and angles, since both the angle and area of each pie slice are proportional to the quantity the slice represents. This turns out to be a sub-optimal choice since, as demonstrated by perception studies, humans are not good at precisely quantifying angles and are even worse when area is the only available visual cue. The donut chart is an example of a plot that uses only area:



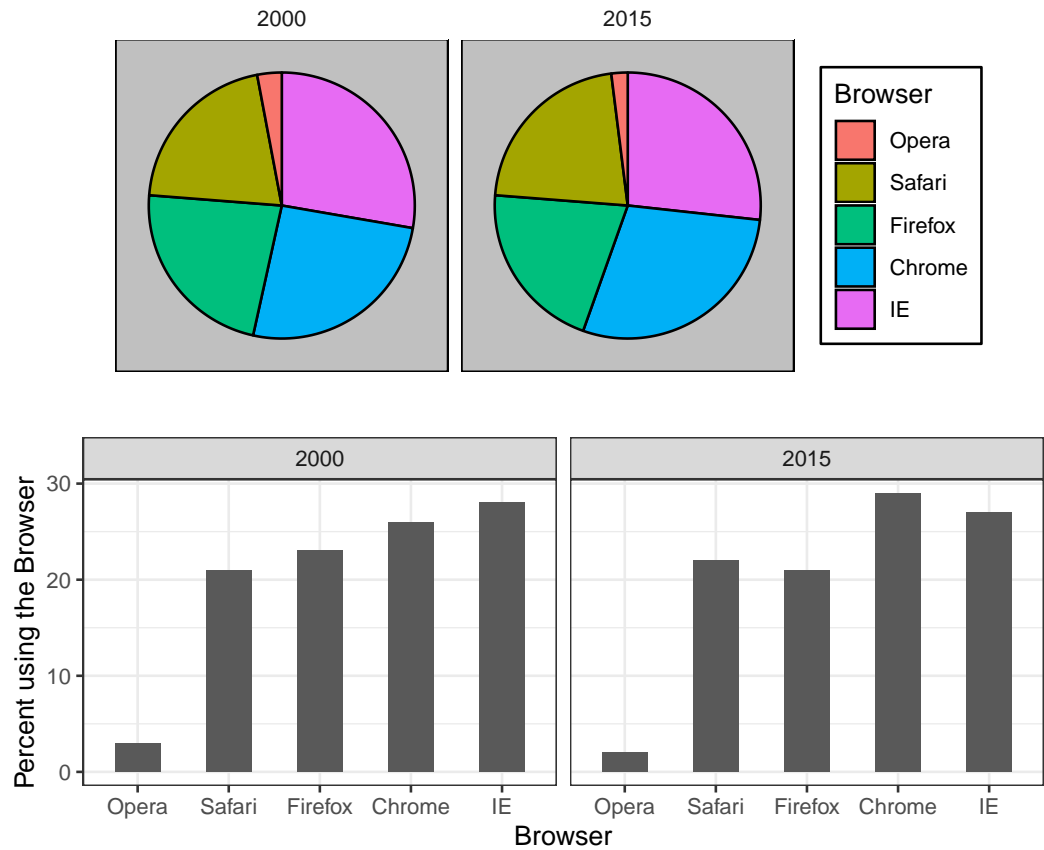
To see how hard it is to quantify angles and area, note that the rankings and all the percentages in the plots above changed from 2000 to 2015. Can you determine the actual percentages and rank the browsers' popularity? Can you see how the percentages changed from 2000 to 2015? It is not easy to tell from the plot. In fact, the `pie` R function help file states that:

Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.

In this case, simply showing the numbers is not only clearer, but would also save on printing costs if printing a paper copy:

| Browser | 2000 | 2015 |
|---------|------|------|
| Opera   | 3    | 2    |
| Safari  | 21   | 22   |
| Firefox | 23   | 21   |
| Chrome  | 26   | 29   |
| IE      | 28   | 27   |

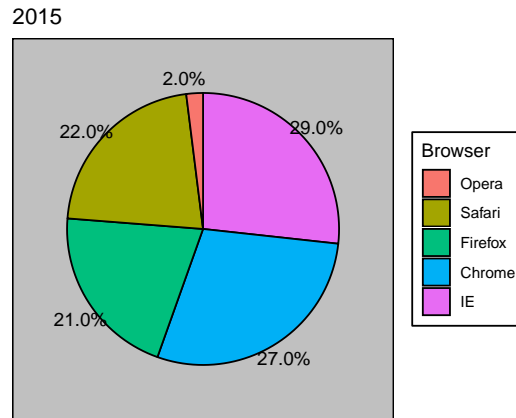
The preferred way to plot these quantities is to use length and position as visual cues, since humans are much better at judging linear measures. The barplot uses this approach by using bars of length proportional to the quantities of interest. By adding horizontal lines at strategically chosen values, in this case at every multiple of 10, we ease the visual burden of quantifying through the position of the top of the bars. Compare and contrast the information we can extract from the two figures.



Notice how much easier it is to see the differences in the barplot. In fact, we can now determine the actual percentages by following a horizontal line to the x-axis.

If for some reason you need to make a pie chart, label each pie slice with its respective percentage so viewers do not have to infer them from the angles or area:

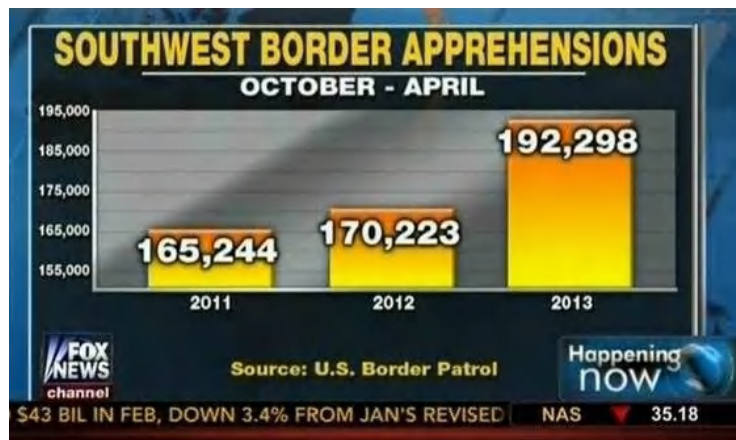




In general, when displaying quantities, position and length are preferred over angles and/or area. Brightness and color are even harder to quantify than angles. But, as we will see later, they are sometimes useful when more than two dimensions must be displayed at once.

## 10.2 Know when to include 0

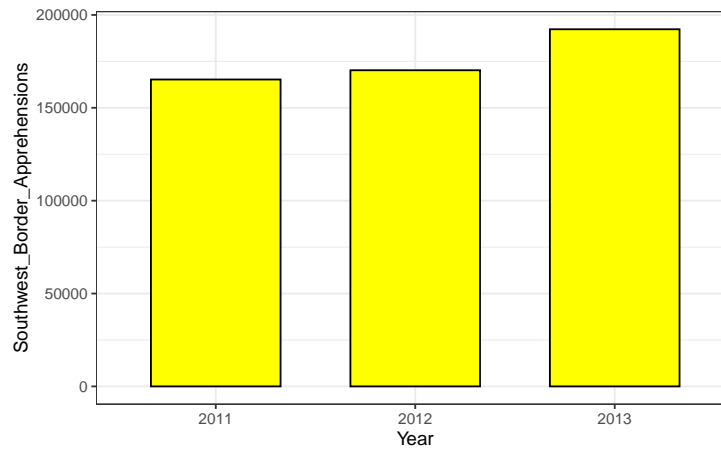
When using barplots, it is misinformative not to start the bars at 0. This is because, by using a barplot, we are implying the length is proportional to the quantities being displayed. By avoiding 0, relatively small differences can be made to look much bigger than they actually are. This approach is often used by politicians or media organizations trying to exaggerate a difference. Below is an illustrative example used by Peter Aldhous in this lecture: <http://paldhous.github.io/ucb/2016/dataviz/week2.html>.



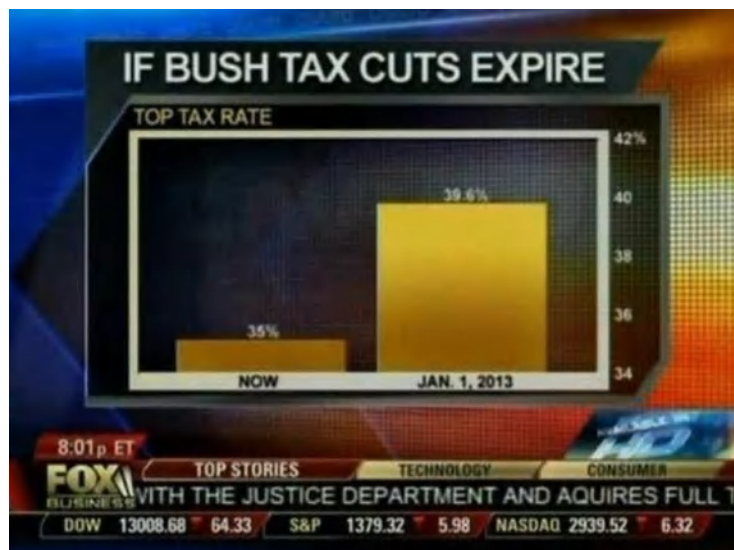
(Source: Fox News, via Media Matters<sup>5</sup>.)

<sup>5</sup><http://mediamatters.org/blog/2013/04/05/fox-news-newest-dishonest-chart-immigration-enf/193507>

From the plot above, it appears that apprehensions have almost tripled when, in fact, they have only increased by about 16%. Starting the graph at 0 illustrates this clearly:



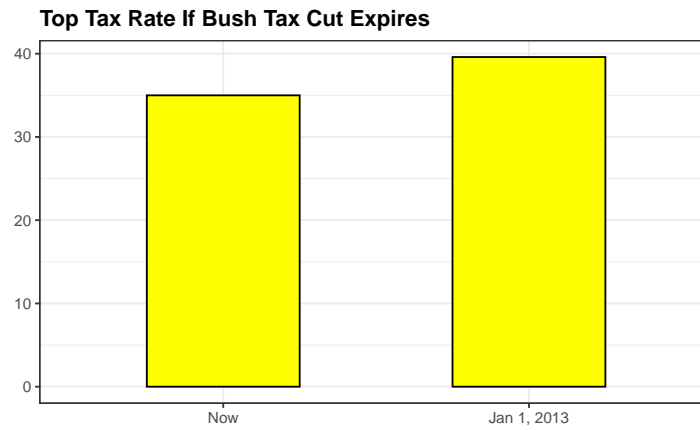
Here is another example, described in detail in a Flowing Data blog post:



(Source: Fox News, via Flowing Data<sup>6</sup>.)

This plot makes a 13% increase look like a five fold change. Here is the appropriate plot:

<sup>6</sup><http://flowingdata.com/2012/08/06/fox-news-continues-charting-excellence/>



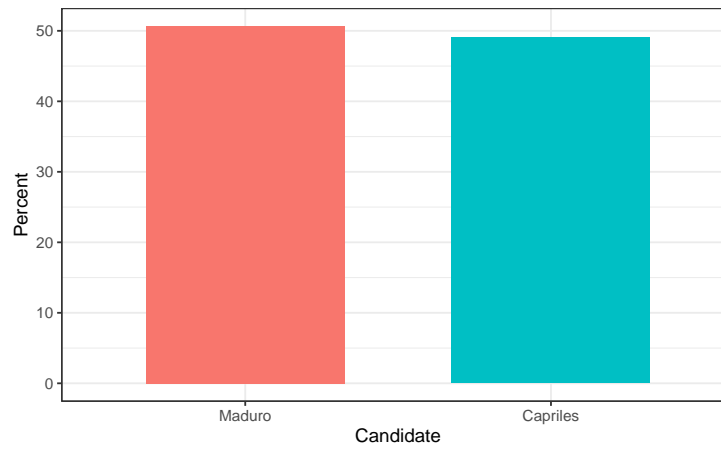
Finally, here is an extreme example that makes a very small difference of under 2% look like a 10-100 fold change:



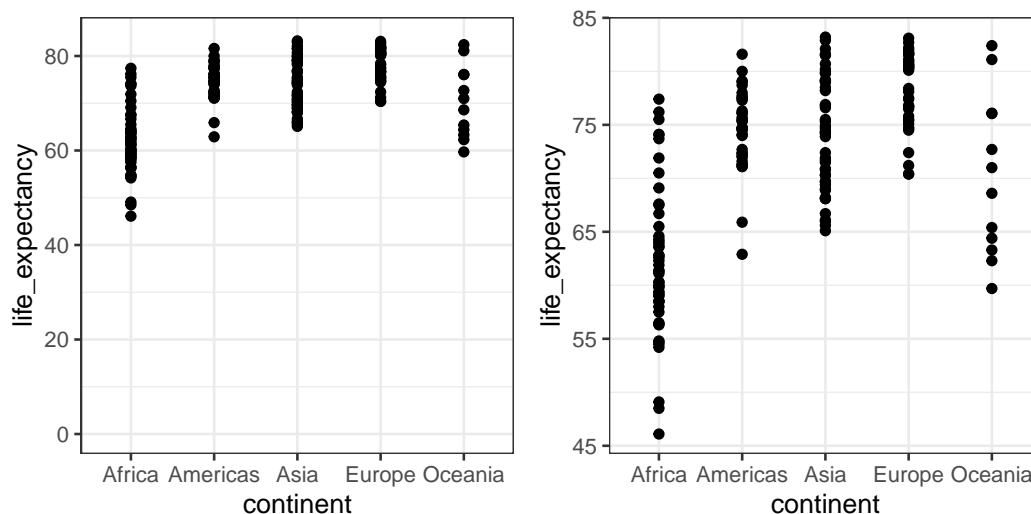
(Source: Venezolana de Televisión via Pakistan Today<sup>7</sup> and Diego Mariano.)

Here is the appropriate plot:

<sup>7</sup><https://www.pakistantoday.com.pk/2018/05/18/whats-at-stake-in-venezuelan-presidential-vote>



When using position rather than length, it is then not necessary to include 0. This is particularly the case when we want to compare differences between groups relative to the within-group variability. Here is an illustrative example showing country average life expectancy stratified across continents in 2012:



Note that in the plot on the left, which includes 0, the space between 0 and 43 adds no information and makes it harder to compare the between and within group variability.

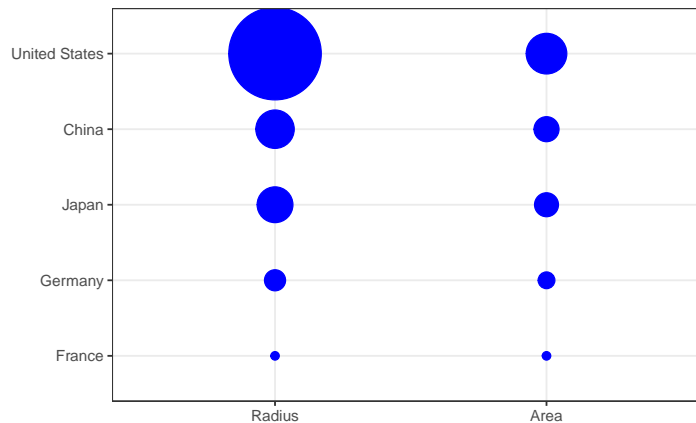
### 10.3 Do not distort quantities

During President Barack Obama's 2011 State of the Union Address, the following chart was used to compare the US GDP to the GDP of four competing nations:



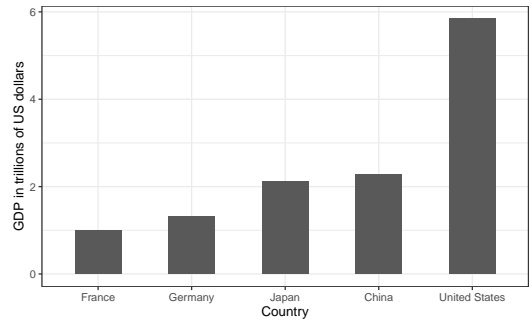
(Source: The 2011 State of the Union Address<sup>8</sup>)

Judging by the area of the circles, the US appears to have an economy over five times larger than China's and over 30 times larger than France's. However, if we look at the actual numbers, we see that this is not the case. The actual ratios are 2.6 and 5.8 times bigger than China and France, respectively. The reason for this distortion is that the radius, rather than the area, was made to be proportional to the quantity, which implies that the proportion between the areas is squared: 2.6 turns into 6.5 and 5.8 turns into 34.1. Here is a comparison of the circles we get if we make the value proportional to the radius and to the area:



Not surprisingly, **ggplot2** defaults to using area rather than radius. Of course, in this case, we really should not be using area at all since we can use position and length:

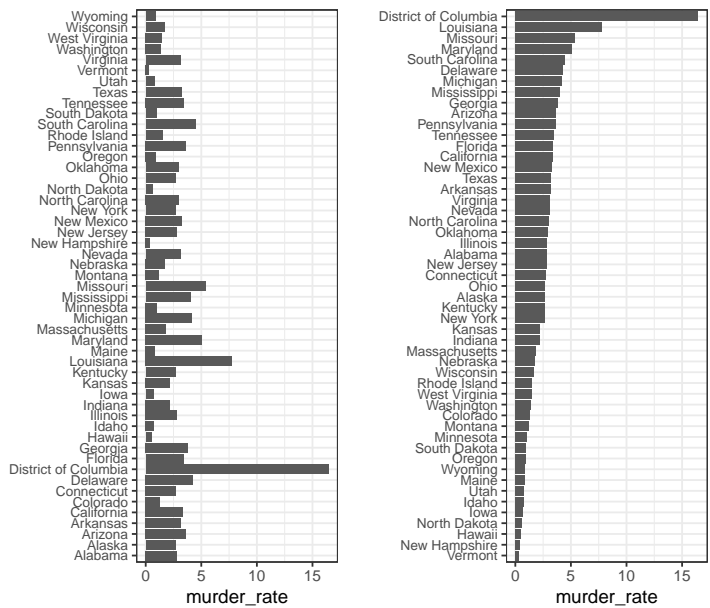
<sup>8</sup><https://www.youtube.com/watch?v=kl2g40GoRxg>



## 10.4 Order categories by a meaningful value

When one of the axes is used to show categories, as is done in barplots, the default **ggplot2** behavior is to order the categories alphabetically when they are defined by character strings. If they are defined by factors, they are ordered by the factor levels. We rarely want to use alphabetical order. Instead, we should order by a meaningful quantity. In all the cases above, the barplots were ordered by the values being displayed. The exception was the graph showing barplots comparing browsers. In this case, we kept the order the same across the barplots to ease the comparison. Specifically, instead of ordering the browsers separately in the two years, we ordered both years by the average value of 2000 and 2015.

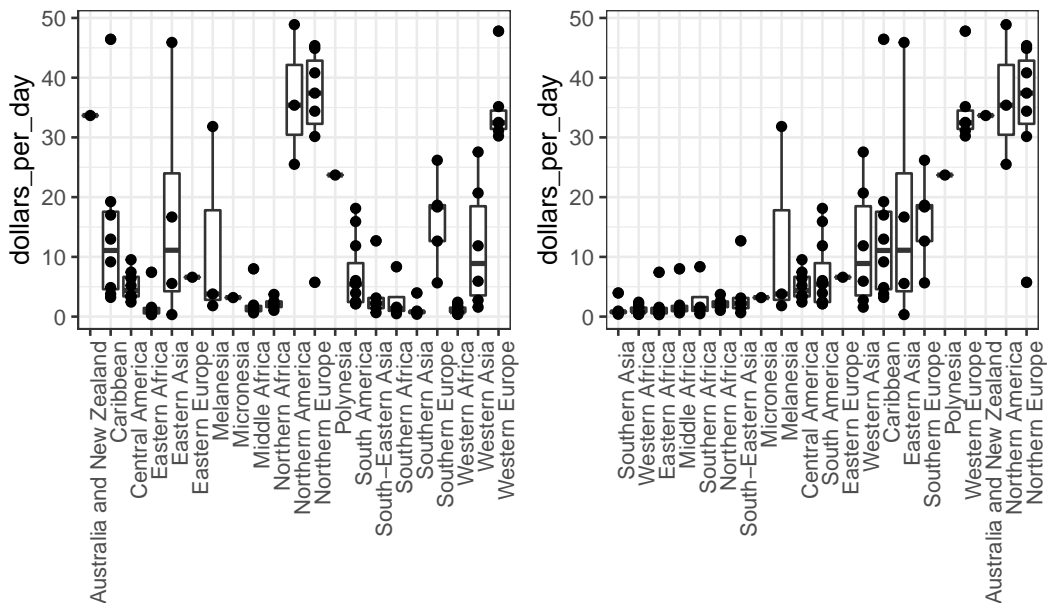
We previously learned how to use the **reorder** function, which helps us achieve this goal. To appreciate how the right order can help convey a message, suppose we want to create a plot to compare the murder rate across states. We are particularly interested in the most dangerous and safest states. Note the difference when we order alphabetically (the default) versus when we order by the actual rate:



We can make the second plot like this:

```
data(murders)
murders %>% mutate(murder_rate = total / population * 100000) %>%
 mutate(state = reorder(state, murder_rate)) %>%
 ggplot(aes(state, murder_rate)) +
 geom_bar(stat="identity") +
 coord_flip() +
 theme(axis.text.y = element_text(size = 6)) +
 xlab("")
```

The `reorder` function lets us reorder groups as well. Earlier we saw an example related to income distributions across regions. Here are the two versions plotted against each other:



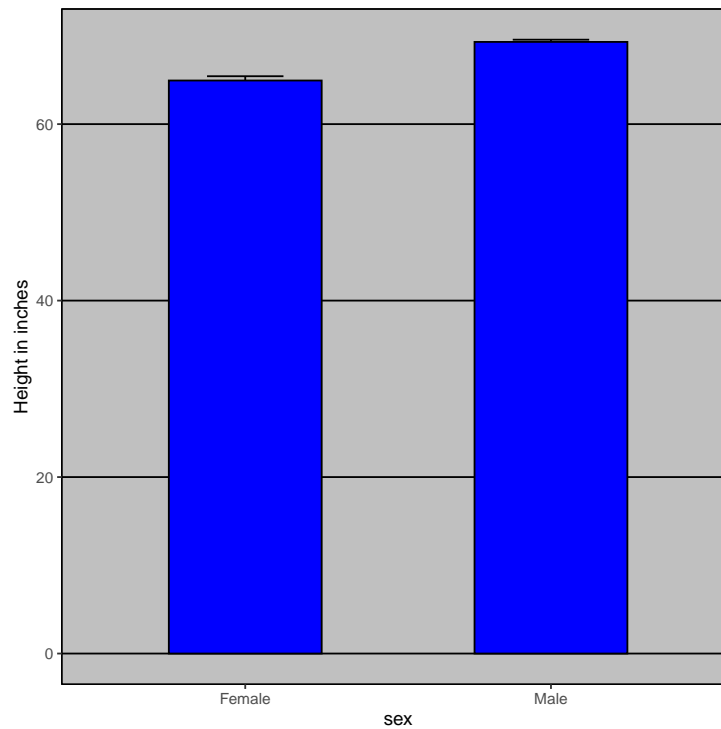
The first orders the regions alphabetically, while the second orders them by the group's median.

## 10.5 Show the data

We have focused on displaying single quantities across categories. We now shift our attention to displaying data, with a focus on comparing groups.

To motivate our first principle, “show the data”, we go back to our artificial example of describing heights to ET, an extraterrestrial. This time let's assume ET is interested in the

difference in heights between males and females. A commonly seen plot used for comparisons between groups, popularized by software such as Microsoft Excel, is the dynamite plot, which shows the average and standard errors (standard errors are defined in a later chapter, but do not confuse them with the standard deviation of the data). The plot looks like this:

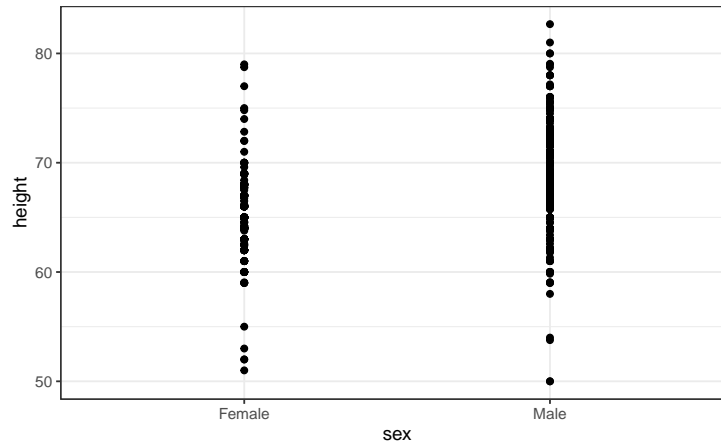


The average of each group is represented by the top of each bar and the antennae extend out from the average to the average plus two standard errors. If all ET receives is this plot, he will have little information on what to expect if he meets a group of human males and females. The bars go to 0: does this mean there are tiny humans measuring less than one foot? Are all males taller than the tallest females? Is there a range of heights? ET can't answer these questions since we have provided almost no information on the height distribution.

This brings us to our first principle: show the data. This simple **ggplot2** code already generates a more informative plot than the barplot by simply showing all the data points:

```
heights %>%
 ggplot(aes(sex, height)) +
 geom_point()
```

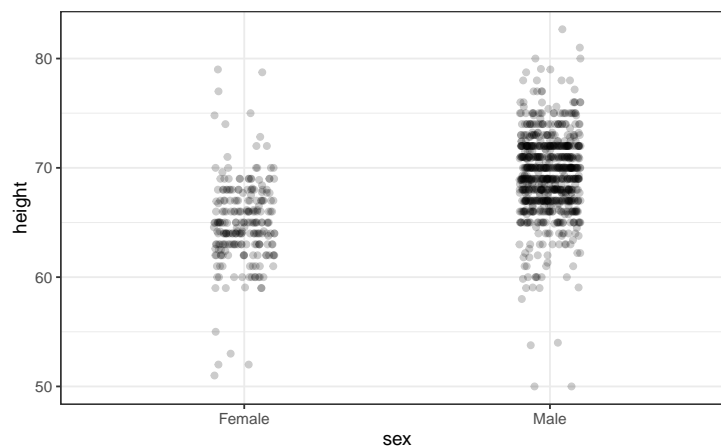




For example, this plot gives us an idea of the range of the data. However, this plot has limitations as well, since we can't really see all the 238 and 812 points plotted for females and males, respectively, and many points are plotted on top of each other. As we have previously described, visualizing the distribution is much more informative. But before doing this, we point out two ways we can improve a plot showing all the points.

The first is to add *jitter*, which adds a small random shift to each point. In this case, adding horizontal jitter does not alter the interpretation, since the point heights do not change, but we minimize the number of points that fall on top of each other and, therefore, get a better visual sense of how the data is distributed. A second improvement comes from using *alpha blending*: making the points somewhat transparent. The more points fall on top of each other, the darker the plot, which also helps us get a sense of how the points are distributed. Here is the same plot with jitter and alpha blending:

```
heights %>%
 ggplot(aes(sex, height)) +
 geom_jitter(width = 0.1, alpha = 0.2)
```

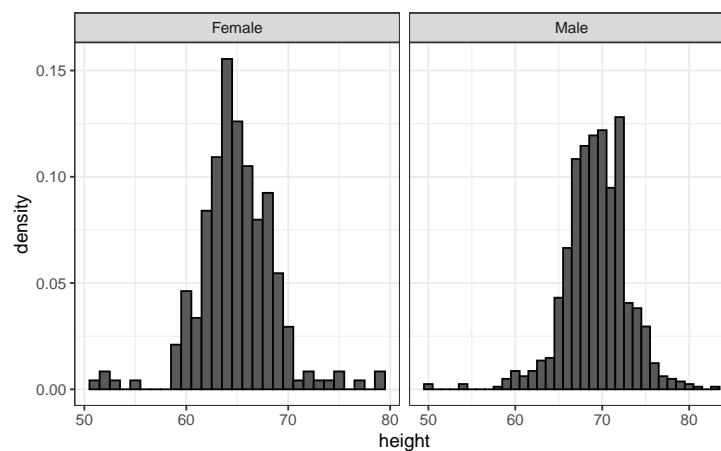


Now we start getting a sense that, on average, males are taller than females. We also note dark horizontal bands of points, demonstrating that many report values that are rounded to the nearest integer.

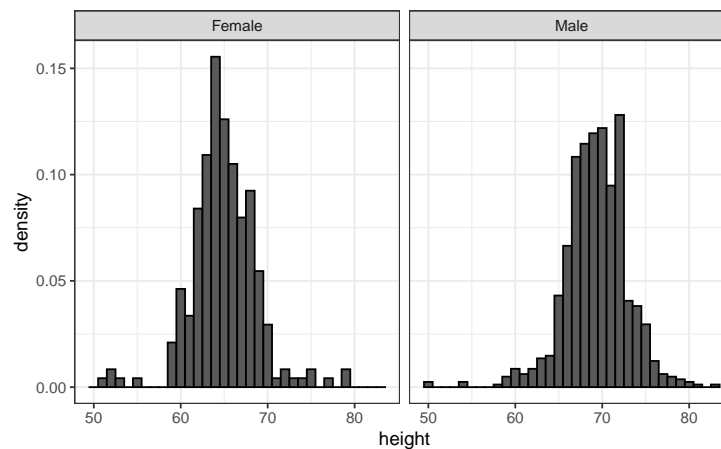
## 10.6 Ease comparisons

### 10.6.1 Use common axes

Since there are so many points, it is more effective to show distributions rather than individual points. We therefore show histograms for each group:

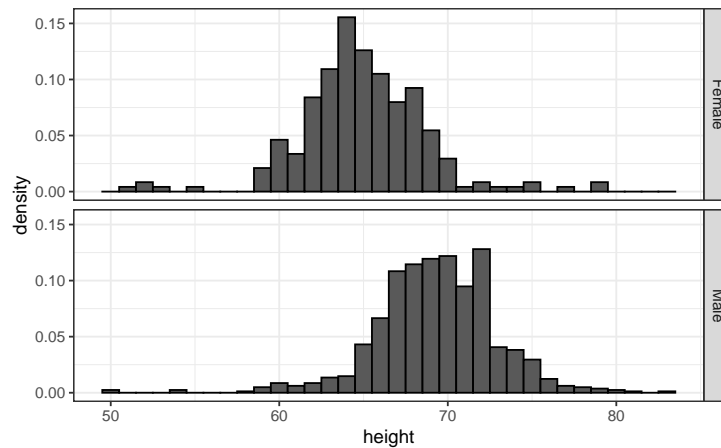


However, from this plot it is not immediately obvious that males are, on average, taller than females. We have to look carefully to notice that the x-axis has a higher range of values in the male histogram. An important principle here is to **keep the axes the same** when comparing data across two plots. Below we see how the comparison becomes easier:



### 10.6.2 Align plots vertically to see horizontal changes and horizontally to see vertical changes

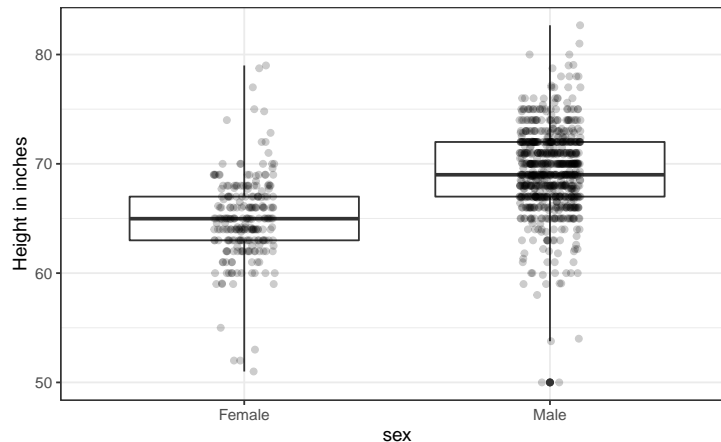
In these histograms, the visual cue related to decreases or increases in height are shifts to the left or right, respectively: horizontal changes. Aligning the plots vertically helps us see this change when the axes are fixed:



```
heights %>%
 ggplot(aes(height, ..density..)) +
 geom_histogram(binwidth = 1, color="black") +
 facet_grid(sex~.)
```

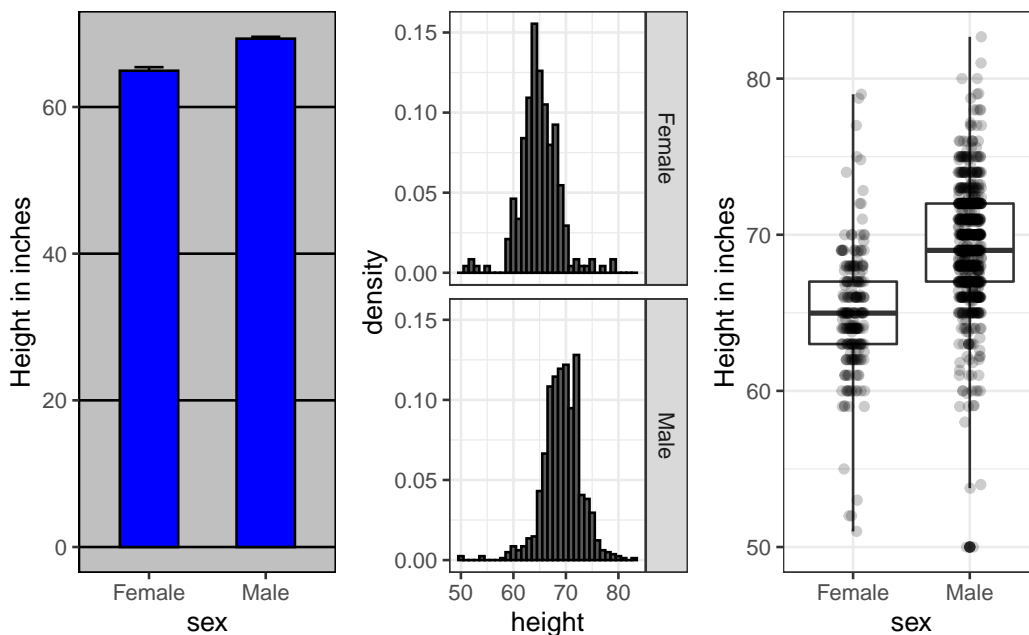
This plot makes it much easier to notice that men are, on average, taller.

If, we want the more compact summary provided by boxplots, we then align them horizontally since, by default, boxplots move up and down with changes in height. Following our *show the data* principle, we then overlay all the data points:



```
heights %>%
 ggplot(aes(sex, height)) +
 geom_boxplot(coef=3) +
 geom_jitter(width = 0.1, alpha = 0.2) +
 ylab("Height in inches")
```

Now contrast and compare these three plots, based on exactly the same data:

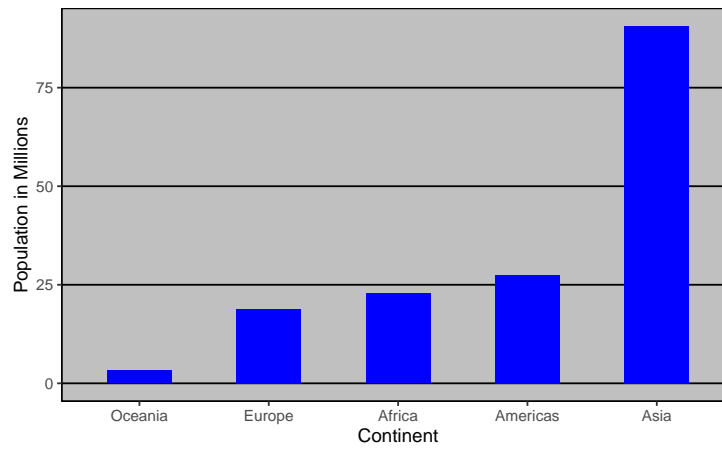


Notice how much more we learn from the two plots on the right. Barplots are useful for showing one number, but not very useful when we want to describe distributions.

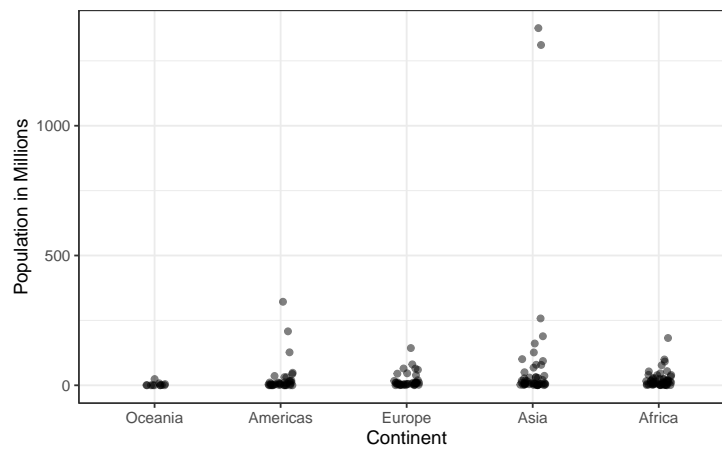
### 10.6.3 Consider transformations

We have motivated the use of the log transformation in cases where the changes are multiplicative. Population size was an example in which we found a log transformation to yield a more informative transformation.

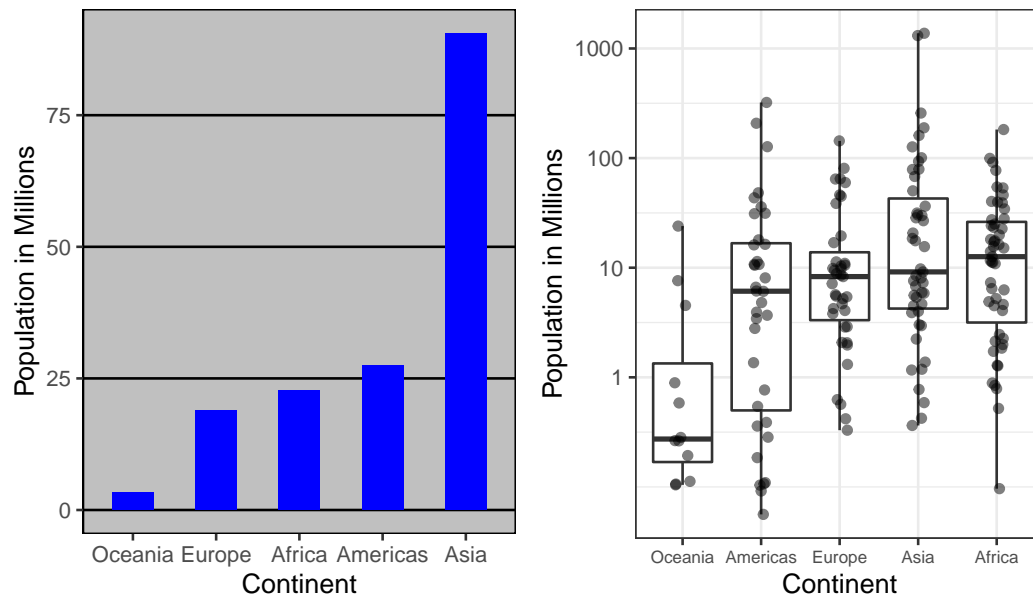
The combination of an incorrectly chosen barplot and a failure to use a log transformation when one is merited can be particularly distorting. As an example, consider this barplot showing the average population sizes for each continent in 2015:



From this plot, one would conclude that countries in Asia are much more populous than in other continents. Following the *show the data* principle, we quickly notice that this is due to two very large countries, which we assume are India and China:



Using a log transformation here provides a much more informative plot. We compare the original barplot to a boxplot using the log scale transformation for the y-axis:

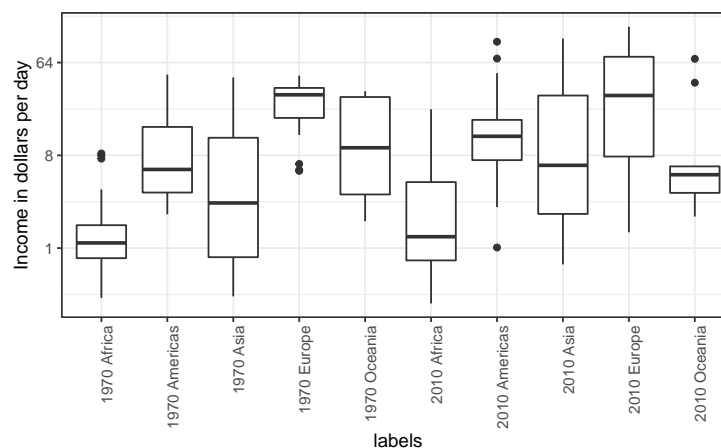


With the new plot, we realize that countries in Africa actually have a larger median population size than those in Asia.

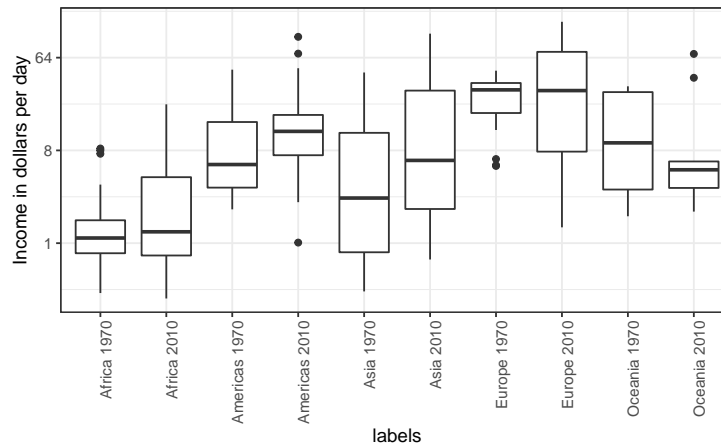
Other transformations you should consider are the logistic transformation (`logit`), useful to better see fold changes in odds, and the square root transformation (`sqrt`), useful for count data.

#### 10.6.4 Visual cues to be compared should be adjacent

For each continent, let's compare income in 1970 versus 2010. When comparing income data across regions between 1970 and 2010, we made a figure similar to the one below, but this time we investigate continents rather than regions.

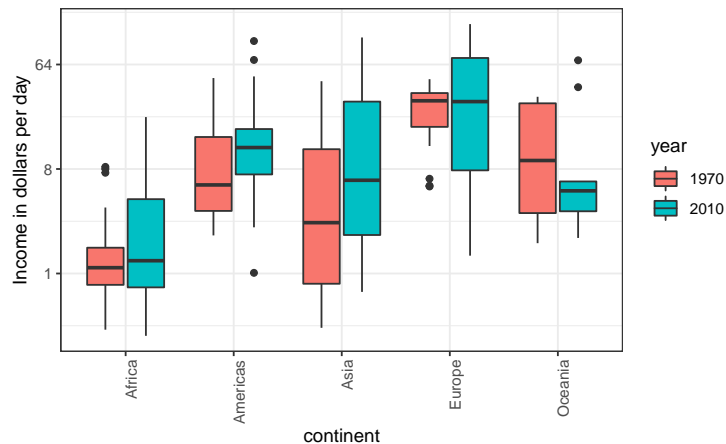


The default in **ggplot2** is to order labels alphabetically so the labels with 1970 come before the labels with 2010, making the comparisons challenging because a continent's distribution in 1970 is visually far from its distribution in 2010. It is much easier to make the comparison between 1970 and 2010 for each continent when the boxplots for that continent are next to each other:



### 10.6.5 Use color

The comparison becomes even easier to make if we use color to denote the two things we want to compare:



## 10.7 Think of the color blind

About 10% of the population is color blind. Unfortunately, the default colors used in **ggplot2** are not optimal for this group. However, **ggplot2** does make it easy to change the

color palette used in the plots. An example of how we can use a color blind friendly palette is described here: [http://www.cookbook-r.com/Graphs/Colors\\_\(ggplot2\)/#a-colorblind-friendly-palette](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/#a-colorblind-friendly-palette):

```
color_blind_friendly_cols <-
 c("#999999", "#E69F00", "#56B4E9", "#009E73",
 "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```

Here are the colors



There are several resources that can help you select colors, for example this one: <http://bconnelly.net/2013/10/creating-colorblind-friendly-figures/>.

## 10.8 Plots for two variables

In general, you should use scatterplots to visualize the relationship between two variables. In every single instance in which we have examined the relationship between two variables, including total murders versus population size, life expectancy versus fertility rates, and infant mortality versus income, we have used scatterplots. This is the plot we generally recommend. However, there are some exceptions and we describe two alternative plots here: the *slope chart* and the *Bland-Altman plot*.

### 10.8.1 Slope charts

One exception where another type of plot may be more informative is when you are comparing variables of the same type, but at different time points and for a relatively small number of comparisons. For example, comparing life expectancy between 2010 and 2015. In this case, we might recommend a *slope chart*.

There is no geometry for slope charts in **ggplot2**, but we can construct one using **geom\_line**. We need to do some tinkering to add labels. Below is an example comparing 2010 to 2015 for large western countries:

```
west <- c("Western Europe", "Northern Europe", "Southern Europe",
 "Northern America", "Australia and New Zealand")

dat <- gapminder %>%
 filter(year%in% c(2010, 2015) & region %in% west &
 !is.na(life_expectancy) & population > 10^7)

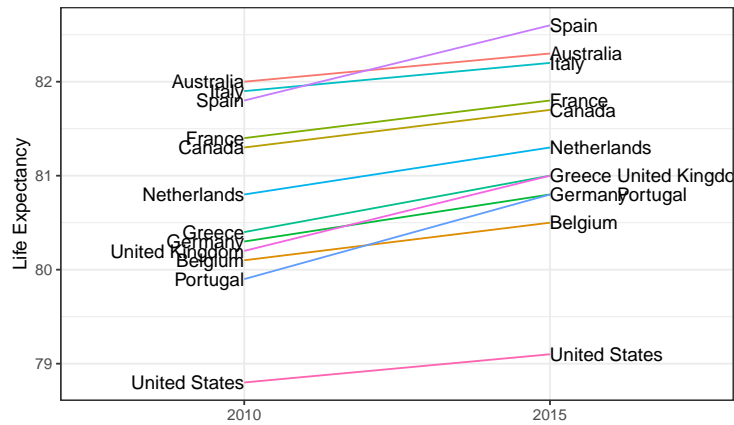
dat %>%
 mutate(location = ifelse(year == 2010, 1, 2),
```



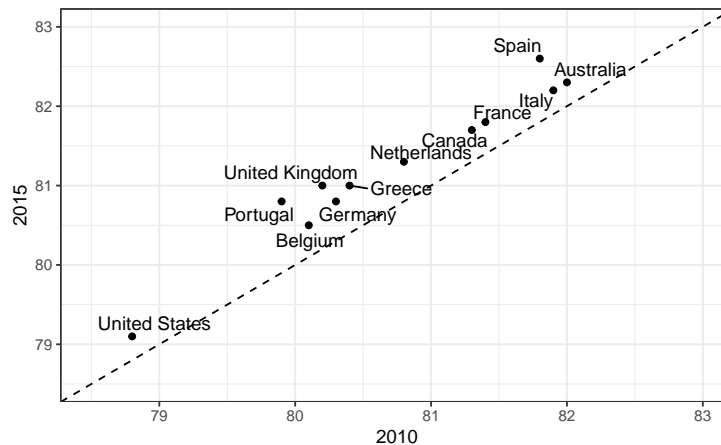
```

location = ifelse(year == 2015 &
 country %in% c("United Kingdom", "Portugal"),
 location+0.22, location),
hjust = ifelse(year == 2010, 1, 0)) %>%
mutate(year = as.factor(year)) %>%
ggplot(aes(year, life_expectancy, group = country)) +
geom_line(aes(color = country), show.legend = FALSE) +
geom_text(aes(x = location, label = country, hjust = hjust),
 show.legend = FALSE) +
xlab("") + ylab("Life Expectancy")

```



An advantage of the slope chart is that it permits us to quickly get an idea of changes based on the slope of the lines. Although we are using angle as the visual cue, we also have position to determine the exact values. Comparing the improvements is a bit harder with a scatterplot:

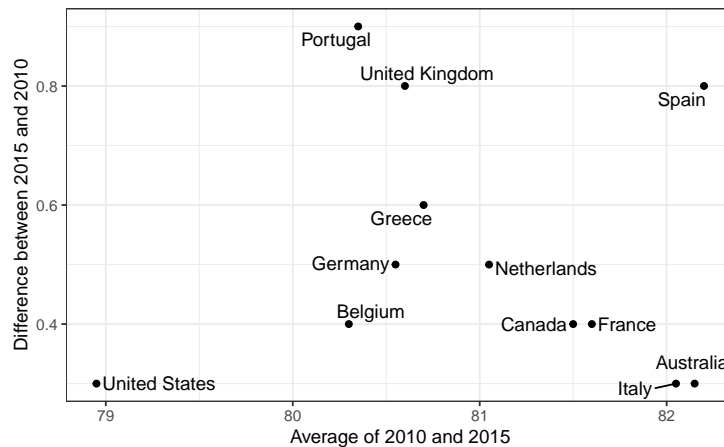


In the scatterplot, we have followed the principle *use common axes* since we are comparing these before and after. However, if we have many points, slope charts stop being useful as it becomes hard to see all the lines.

### 10.8.2 Bland-Altman plot

Since we are primarily interested in the difference, it makes sense to dedicate one of our axes to it. The Bland-Altman plot, also known as the Tukey mean-difference plot and the MA-plot, shows the difference versus the average:

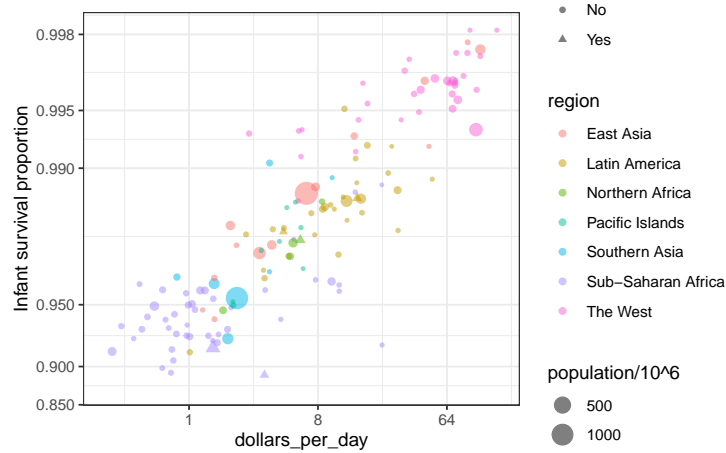
```
library(ggrepel)
dat %>%
 mutate(year = paste0("life_expectancy_", year)) %>%
 select(country, year, life_expectancy) %>%
 spread(year, life_expectancy) %>%
 mutate(average = (life_expectancy_2015 + life_expectancy_2010)/2,
 difference = life_expectancy_2015 - life_expectancy_2010) %>%
 ggplot(aes(average, difference, label = country)) +
 geom_point() +
 geom_text_repel() +
 geom_abline(lty = 2) +
 xlab("Average of 2010 and 2015") +
 ylab("Difference between 2015 and 2010")
```



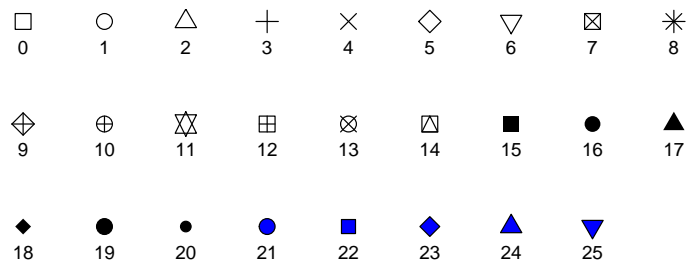
Here, by simply looking at the y-axis, we quickly see which countries have shown the most improvement. We also get an idea of the overall value from the x-axis.

## 10.9 Encoding a third variable

An earlier scatterplot showed the relationship between infant survival and average income. Below is a version of this plot that encodes three variables: OPEC membership, region, and population.



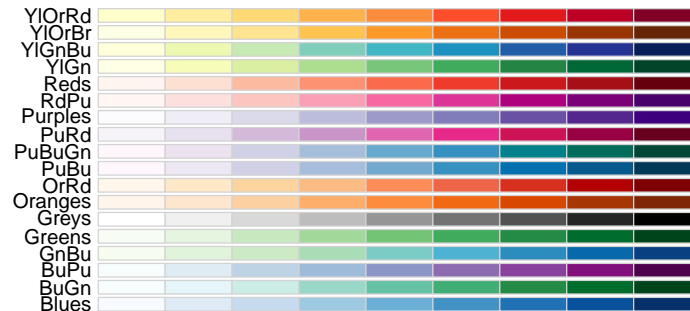
We encode categorical variables with color and shape. These shapes can be controlled with `shape` argument. Below are the shapes available for use in R. For the last five, the color goes inside.



For continuous variables, we can use color, intensity, or size. We now show an example of how we do this with a case study.

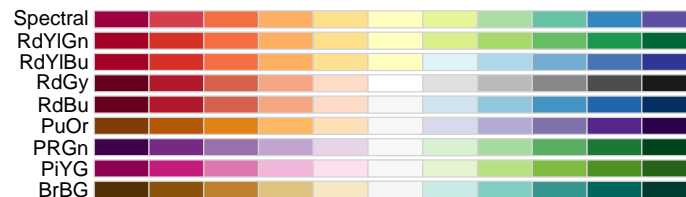
When selecting colors to quantify a numeric variable, we choose between two options: sequential and diverging. Sequential colors are suited for data that goes from high to low. High values are clearly distinguished from low values. Here are some examples offered by the package `RColorBrewer`:

```
library(RColorBrewer)
display.brewer.all(type="seq")
```



Diverging colors are used to represent values that diverge from a center. We put equal emphasis on both ends of the data range: higher than the center and lower than the center. An example of when we would use a divergent pattern would be if we were to show height in standard deviations away from the average. Here are some examples of divergent patterns:

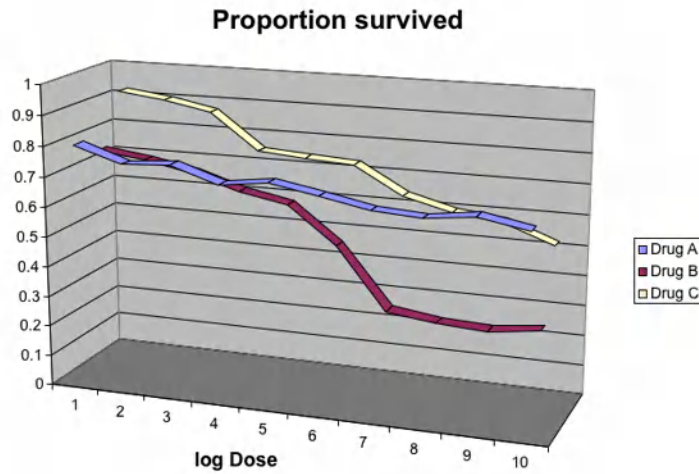
```
library(RColorBrewer)
display.brewer.all(type="div")
```



## 10.10 Avoid pseudo-three-dimensional plots

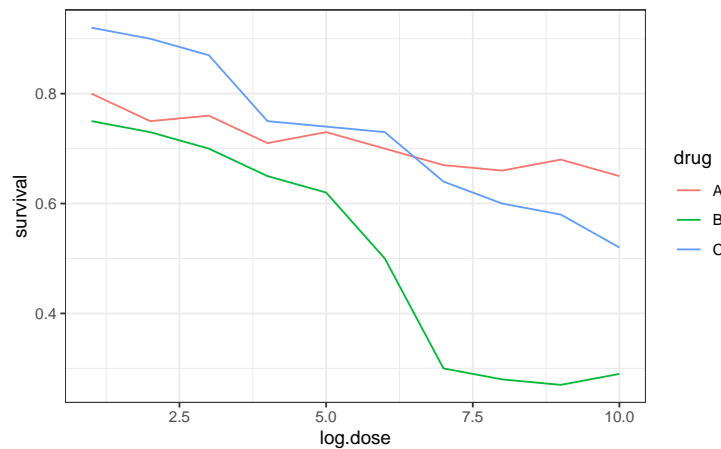
The figure below, taken from the scientific literature<sup>9</sup>, shows three variables: dose, drug type and survival. Although your screen/book page is flat and two-dimensional, the plot tries to imitate three dimensions and assigned a dimension to each variable.

<sup>9</sup>[https://projecteuclid.org/download/pdf\\_1/euclid.ss/1177010488](https://projecteuclid.org/download/pdf_1/euclid.ss/1177010488)



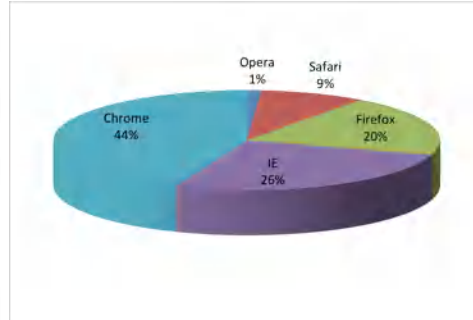
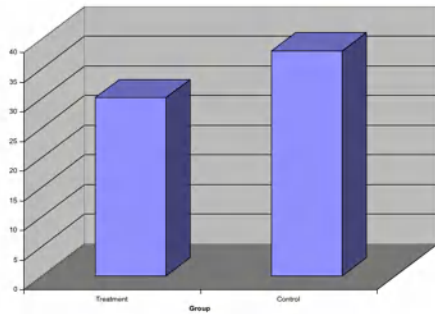
(Image courtesy of Karl Broman)

Humans are not good at seeing in three dimensions (which explains why it is hard to parallel park) and our limitation is even worse with regard to pseudo-three-dimensions. To see this, try to determine the values of the survival variable in the plot above. Can you tell when the purple ribbon intersects the red one? This is an example in which we can easily use color to represent the categorical variable instead of using a pseudo-3D:



Notice how much easier it is to determine the survival values.

Pseudo-3D is sometimes used completely gratuitously: plots are made to look 3D even when the 3rd dimension does not represent a quantity. This only adds confusion and makes it harder to relay your message. Here are two examples:



(Images courtesy of Karl Broman)

## 10.11 Avoid too many significant digits

By default, statistical software like R returns many significant digits. The default behavior in R is to show 7 significant digits. That many digits often adds no information and the added visual clutter can make it hard for the viewer to understand the message. As an example, here are the per 10,000 disease rates, computed from totals and population in R, for California across the five decades:

| state      | year | Measles    | Pertussis  | Polio      |
|------------|------|------------|------------|------------|
| California | 1940 | 37.8826320 | 18.3397861 | 18.3397861 |
| California | 1950 | 13.9124205 | 4.7467350  | 4.7467350  |
| California | 1960 | 14.1386471 | 0.0000000  | 0.0000000  |
| California | 1970 | 0.9767889  | 0.0000000  | 0.0000000  |
| California | 1980 | 0.3743467  | 0.0515466  | 0.0515466  |

We are reporting precision up to 0.00001 cases per 10,000, a very small value in the context of the changes that are occurring across the dates. In this case, two significant figures is more than enough and clearly makes the point that rates are decreasing:

| state      | year | Measles | Pertussis | Polio |
|------------|------|---------|-----------|-------|
| California | 1940 | 37.9    | 18.3      | 18.3  |
| California | 1950 | 13.9    | 4.7       | 4.7   |
| California | 1960 | 14.1    | 0.0       | 0.0   |
| California | 1970 | 1.0     | 0.0       | 0.0   |
| California | 1980 | 0.4     | 0.1       | 0.1   |

Useful ways to change the number of significant digits or to round numbers are `signif` and `round`. You can define the number of significant digits globally by setting options like this: `options(digits = 3)`.

Another principle related to displaying tables is to place values being compared on columns rather than rows. Note that our table above is easier to read than this one:

| state      | disease   | 1940 | 1950 | 1960 | 1970 | 1980 |
|------------|-----------|------|------|------|------|------|
| California | Measles   | 37.9 | 13.9 | 14.1 | 1    | 0.4  |
| California | Pertussis | 18.3 | 4.7  | 0.0  | 0    | 0.1  |
| California | Polio     | 18.3 | 4.7  | 0.0  | 0    | 0.1  |

## 10.12 Know your audience

Graphs can be used for 1) our own exploratory data analysis, 2) to convey a message to experts, or 3) to help tell a story to a general audience. Make sure that the intended audience understands each element of the plot.

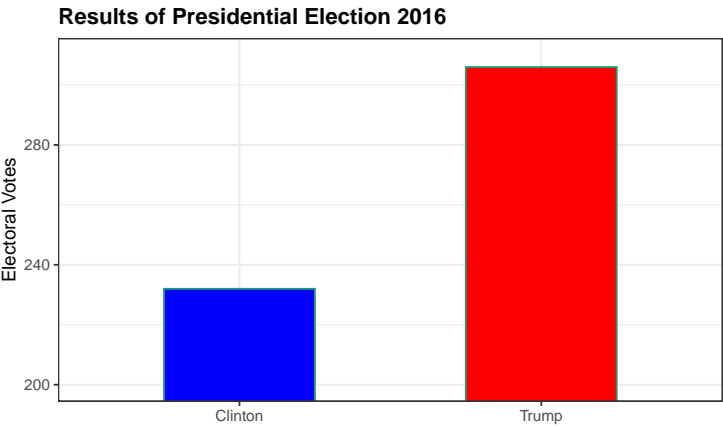
As a simple example, consider that for your own exploration it may be more useful to log-transform data and then plot it. However, for a general audience that is unfamiliar with converting logged values back to the original measurements, using a log-scale for the axis instead of log-transformed values will be much easier to digest.

## 10.13 Exercises

For these exercises, we will be using the vaccines data in the **dslabs** package:

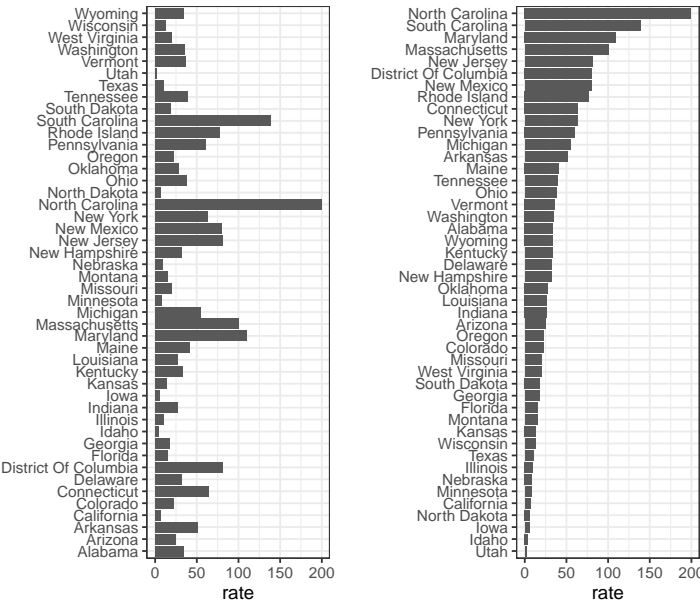
```
library(dslabs)
data(us_contagious_diseases)
```

1. Pie charts are appropriate:
  - a. When we want to display percentages.
  - b. When **ggplot2** is not available.
  - c. When I am in a bakery.
  - d. Never. Barplots and tables are always better.
2. What is the problem with the plot below:



- a. The values are wrong. The final vote was 306 to 232.
- b. The axis does not start at 0. Judging by the length, it appears Trump received 3 times as many votes when, in fact, it was about 30% more.
- c. The colors should be the same.
- d. Percentages should be shown as a pie chart.

3. Take a look at the following two plots. They show the same information: 1928 rates of measles across the 50 states.



Which plot is easier to read if you are interested in determining which are the best and worst states in terms of rates, and why?

- a. They provide the same information, so they are both equally as good.
- b. The plot on the right is better because it orders the states alphabetically.



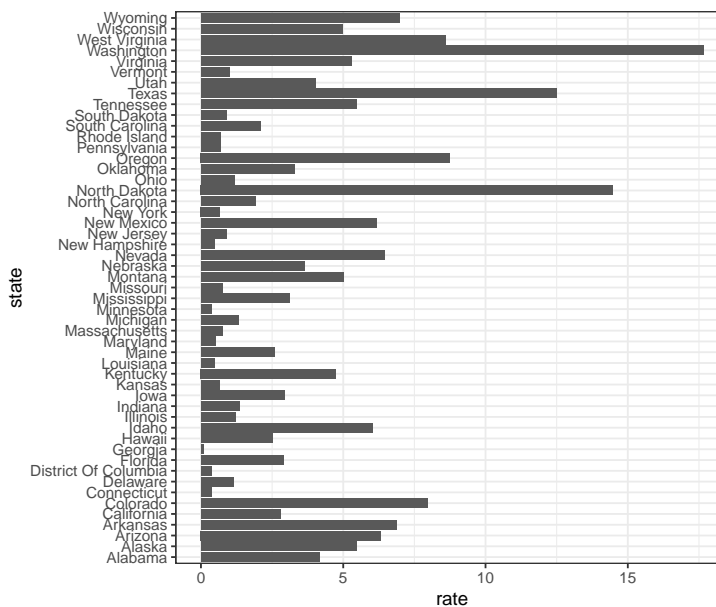
- c. The plot on the right is better because alphabetical order has nothing to do with the disease and by ordering according to actual rate, we quickly see the states with most and least rates.
- d. Both plots should be a pie chart.

4. To make the plot on the left, we have to reorder the levels of the states' variables.

```
dat <- us_contagious_diseases %>%
 filter(year == 1967 & disease=="Measles" & !is.na(population)) %>%
 mutate(rate = count / population * 10000 * 52 / weeks_reporting)
```

Note what happens when we make a barplot:

```
dat %>% ggplot(aes(state, rate)) +
 geom_bar(stat="identity") +
 coord_flip()
```



Define these objects:

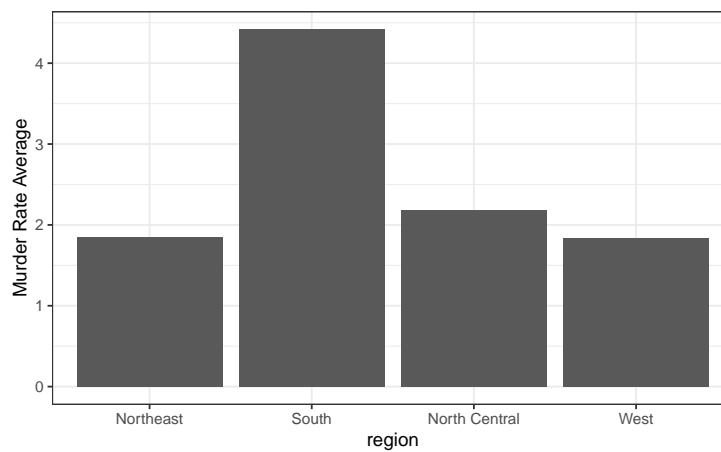
```
state <- dat$state
rate <- dat$count/dat$population*10000*52/dat$weeks_reporting
```

Redefine the `state` object so that the levels are re-ordered. Print the new object `state` and its levels so you can see that the vector is not re-ordered by the levels.

5. Now with one line of code, define the `dat` table as done above, but change the use `mutate` to create a `rate` variable and re-order the `state` variable so that the levels are re-ordered by this variable. Then make a barplot using the code above, but for this new `dat`.

6. Say we are interested in comparing gun homicide rates across regions of the US. We see this plot:

```
library(dslabs)
data("murders")
murders %>% mutate(rate = total/population*100000) %>%
group_by(region) %>%
summarize(avg = mean(rate)) %>%
mutate(region = factor(region)) %>%
ggplot(aes(region, avg)) +
geom_bar(stat="identity") +
ylab("Murder Rate Average")
```



and decide to move to a state in the western region. What is the main problem with this interpretation?

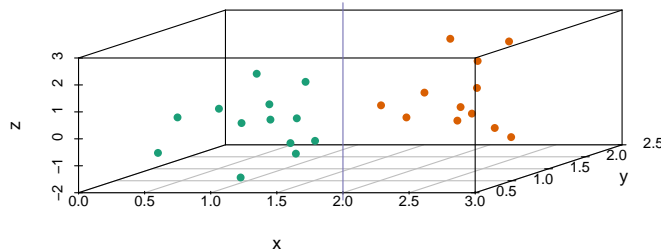
- The categories are ordered alphabetically.
- The graph does not show standard errors.
- It does not show all the data. We do not see the variability within a region and it's possible that the safest states are not in the West.
- The Northeast has the lowest average.

7. Make a boxplot of the murder rates defined as

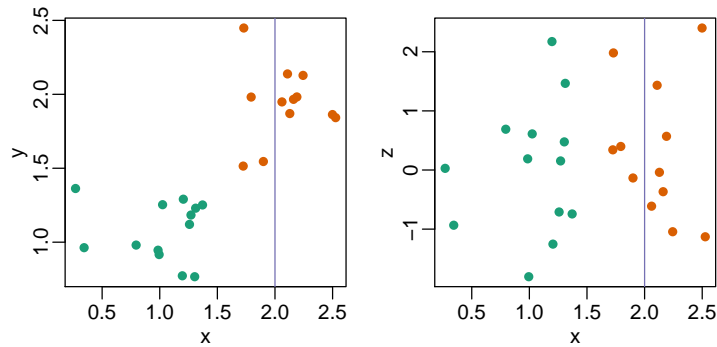
```
data("murders")
murders %>% mutate(rate = total/population*100000)
```

by region, showing all the points and ordering the regions by their median rate.

8. The plots below show three continuous variables.



The line  $x = 2$  appears to separate the points. But it is actually not the case, which we can see by plotting the data in a couple of two-dimensional plots.



Why is this happening?

- Humans are not good at reading pseudo-3D plots.
- There must be an error in the code.
- The colors confuse us.
- Scatterplots should not be used to compare two variables when we have access to 3.

9. Reproduce the image plot we previously made but for smallpox. For this plot, do not include years in which cases were not reported in 10 or more weeks.

10. Now reproduce the time series plot we previously made, but this time following the instructions of the previous question.

11. For the state of California, make time series plots showing rates for all diseases. Include only years with 10 or more weeks reporting. Use a different color for each disease.

12. Now do the same for the rates for the US. Hint: compute the US rate by using `summarize`, the total divided by total population.

## 10.14 Case study: vaccines and infectious diseases

Vaccines have helped save millions of lives. In the 19th century, before herd immunization was achieved through vaccination programs, deaths from infectious diseases, such as smallpox and polio, were common. However, today vaccination programs have become somewhat controversial despite all the scientific evidence for their importance.

The controversy started with a paper<sup>10</sup> published in 1988 and led by Andrew Wakefield claiming there was a link between the administration of the measles, mumps, and rubella (MMR) vaccine and the appearance of autism and bowel disease. Despite much scientific evidence contradicting this finding, sensationalist media reports and fear-mongering from conspiracy theorists led parts of the public into believing that vaccines were harmful. As a result, many parents ceased to vaccinate their children. This dangerous practice can be potentially disastrous given that the Centers for Disease Control (CDC) estimates that vaccinations will prevent more than 21 million hospitalizations and 732,000 deaths among children born in the last 20 years (see Benefits from Immunization during the Vaccines for Children Program Era — United States, 1994-2013, MMWR<sup>11</sup>). The 1988 paper has since been retracted and Andrew Wakefield was eventually “struck off the UK medical register, with a statement identifying deliberate falsification in the research published in *The Lancet*, and was thereby barred from practicing medicine in the UK.” (source: Wikipedia<sup>12</sup>). Yet misconceptions persist, in part due to self-proclaimed activists who continue to disseminate misinformation about vaccines.

Effective communication of data is a strong antidote to misinformation and fear-mongering. Earlier we used an example provided by a Wall Street Journal article<sup>13</sup> showing data related to the impact of vaccines on battling infectious diseases. Here we reconstruct that example.

The data used for these plots were collected, organized, and distributed by the Tycho Project<sup>14</sup>. They include weekly reported counts for seven diseases from 1928 to 2011, from all fifty states. We include the yearly totals in the **dslabs** package:

```
library(tidyverse)
library(RColorBrewer)
library(dslabs)
data(us_contagious_diseases)
names(us_contagious_diseases)
#> [1] "disease" "state" "year"
#> [4] "weeks_reporting" "count" "population"
```

We create a temporary object **dat** that stores only the measles data, includes a per 100,000 rate, orders states by average value of disease and removes Alaska and Hawaii since they only became states in the late 1950s. Note that there is a **weeks\_reporting** column that tells us for how many weeks of the year data was reported. We have to adjust for that value when computing the rate.

<sup>10</sup>[http://www.thelancet.com/journals/lancet/article/PIIS0140-6736\(97\)11096-0/abstract](http://www.thelancet.com/journals/lancet/article/PIIS0140-6736(97)11096-0/abstract)

<sup>11</sup><https://www.cdc.gov/mmwr/preview/mmwrhtml/mm6316a4.htm>

<sup>12</sup>[https://en.wikipedia.org/wiki/Andrew\\_Wakefield](https://en.wikipedia.org/wiki/Andrew_Wakefield)

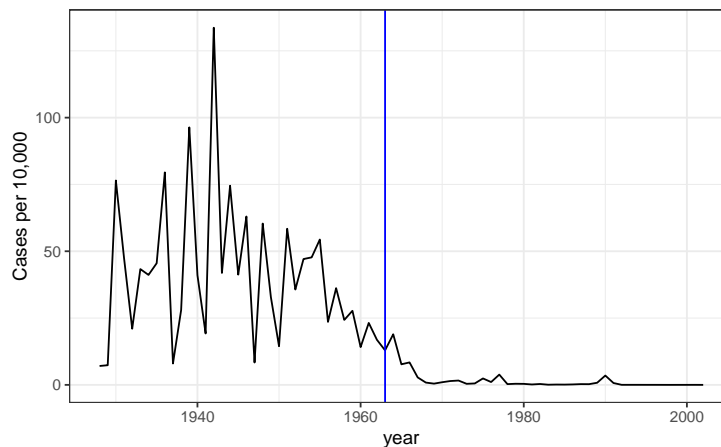
<sup>13</sup><http://graphics.wsj.com/infectious-diseases-and-vaccines/>

<sup>14</sup><http://www.tycho.pitt.edu/>

```
the_disease <- "Measles"
dat <- us_contagious_diseases %>%
 filter(!state%in%c("Hawaii","Alaska") & disease == the_disease) %>%
 mutate(rate = count / population * 10000 * 52 / weeks_reporting) %>%
 mutate(state = reorder(state, rate))
```

We can now easily plot disease rates per year. Here are the measles data from California:

```
dat %>% filter(state == "California" & !is.na(rate)) %>%
 ggplot(aes(year, rate)) +
 geom_line() +
 ylab("Cases per 10,000") +
 geom_vline(xintercept=1963, col = "blue")
```



We add a vertical line at 1963 since this is when the vaccine was introduced [Control, Centers for Disease; Prevention (2014). CDC health information for international travel 2014 (the yellow book). p. 250. ISBN 9780199948505].

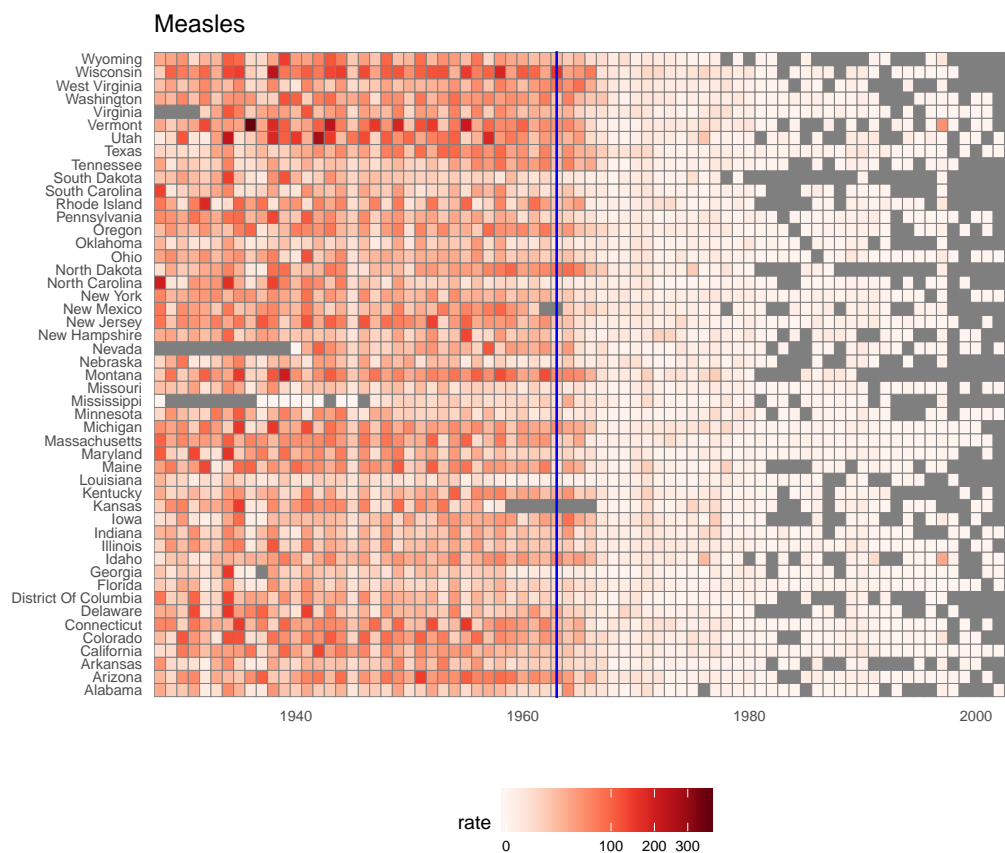
Now can we show data for all states in one plot? We have three variables to show: year, state, and rate. In the WSJ figure, they use the x-axis for year, the y-axis for state, and color hue to represent rates. However, the color scale they use, which goes from yellow to blue to green to orange to red, can be improved.

In our example, we want to use a sequential palette since there is no meaningful center, just low and high rates.

We use the geometry `geom_tile` to tile the region with colors representing disease rates. We use a square root transformation to avoid having the really high counts dominate the plot. Notice that missing values are shown in grey. Note that once a disease was pretty much eradicated, some states stopped reporting cases all together. This is why we see so much grey after 1980.

```
dat %>% ggplot(aes(year, state, fill = rate)) +
 geom_tile(color = "grey50") +
 scale_x_continuous(expand=c(0,0)) +
```

```
scale_fill_gradientn(colors = brewer.pal(9, "Reds"), trans = "sqrt") +
geom_vline(xintercept=1963, col = "blue") +
theme_minimal() +
theme(panel.grid = element_blank(),
 legend.position="bottom",
 text = element_text(size = 8)) +
ggtitle(the_disease) +
ylab("") + xlab("")
```

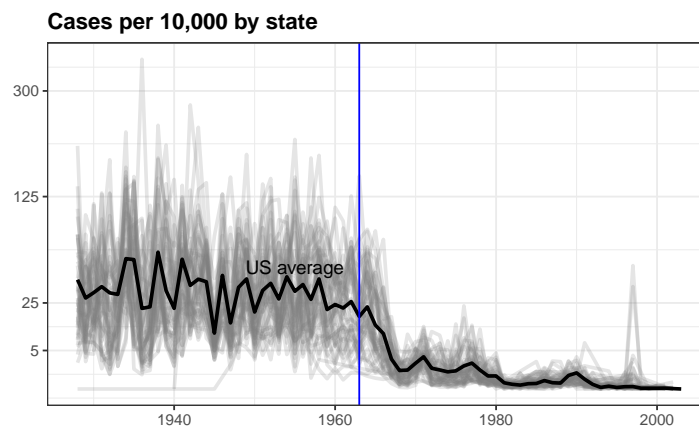


This plot makes a very striking argument for the contribution of vaccines. However, one limitation of this plot is that it uses color to represent quantity, which we earlier explained makes it harder to know exactly how high values are going. Position and lengths are better cues. If we are willing to lose state information, we can make a version of the plot that shows the values with position. We can also show the average for the US, which we compute like this:

```
avg <- us_contagious_diseases %>%
 filter(disease==the_disease) %>% group_by(year) %>%
 summarize(us_rate = sum(count, na.rm = TRUE) /
 sum(population, na.rm = TRUE) * 10000)
```

Now to make the plot we simply use the `geom_line` geometry:

```
dat %>%
 filter(!is.na(rate)) %>%
 ggplot() +
 geom_line(aes(year, rate, group = state), color = "grey50",
 show.legend = FALSE, alpha = 0.2, size = 1) +
 geom_line(mapping = aes(year, us_rate), data = avg, size = 1) +
 scale_y_continuous(trans = "sqrt", breaks = c(5, 25, 125, 300)) +
 ggtitle("Cases per 10,000 by state") +
 xlab("") + ylab("") +
 geom_text(data = data.frame(x = 1955, y = 50),
 mapping = aes(x, y, label="US average"),
 color="black") +
 geom_vline(xintercept=1963, col = "blue")
```



In theory, we could use color to represent the categorical value `state`, but it is hard to pick 50 distinct colors.

## 10.15 Exercises

1. Reproduce the image plot we previously made but for smallpox. For this plot, do not include years in which cases were not reported in 10 or more weeks.
2. Now reproduce the time series plot we previously made, but this time following the instructions of the previous question for smallpox.
3. For the state of California, make a time series plot showing rates for all diseases. Include only years with 10 or more weeks reporting. Use a different color for each disease.
4. Now do the same for the rates for the US. Hint: compute the US rate by using `summarize`: the total divided by total population.

# 11

## *Robust summaries*

### 11.1 Outliers

We previously described how boxplots show *outliers*, but we did not provide a precise definition. Here we discuss outliers, approaches that can help detect them, and summaries that take into account their presence.

Outliers are very common in data science. Data recording can be complex and it is common to observe data points generated in error. For example, an old monitoring device may read out nonsensical measurements before completely failing. Human error is also a source of outliers, in particular when data entry is done manually. An individual, for instance, may mistakenly enter their height in centimeters instead of inches or put the decimal in the wrong place.

How do we distinguish an outlier from measurements that were too big or too small simply due to expected variability? This is not always an easy question to answer, but we try to provide some guidance. Let's begin with a simple case.

Suppose a colleague is charged with collecting demography data for a group of males. The data report height in feet and are stored in the object:

```
library(tidyverse)
library(dslabs)
data(outlier_example)
str(outlier_example)
#> num [1:500] 5.59 5.8 5.54 6.15 5.83 5.54 5.87 5.93 5.89 5.67 ...
```

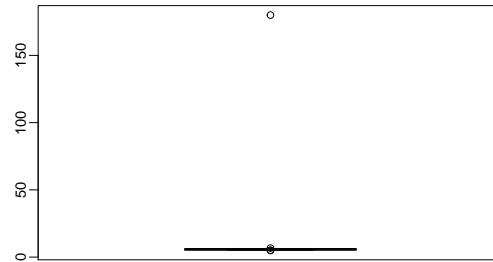
Our colleague uses the fact that heights are usually well approximated by a normal distribution and summarizes the data with average and standard deviation:

```
mean(outlier_example)
#> [1] 6.1
sd(outlier_example)
#> [1] 7.8
```

and writes a report on the interesting fact that this group of males is much taller than usual. The average height is over six feet tall! Using your data science skills, however, you notice something else that is unexpected: the standard deviation is over 7 feet. Adding and subtracting two standard deviations, you note that 95% of this population will have heights between -9.489, 21.697 feet, which does not make sense. A quick plot reveals the problem:



```
boxplot(outlier_example)
```



There appears to be at least one value that is nonsensical, since we know that a height of 180 feet is impossible. The boxplot detects this point as an outlier.

---

## 11.2 Median

When we have an outlier like this, the average can become very large. Mathematically, we can make the average as large as we want by simply changing one number: with 500 data points, we can increase the average by any amount  $\Delta$  by adding  $\Delta \times 500$  to a single number. The median, defined as the value for which half the values are smaller and the other half are bigger, is robust to such outliers. No matter how large we make the largest point, the median remains the same.

With this data the median is:

```
median(outlier_example)
#> [1] 5.74
```

which is about 5 feet and 9 inches.

The median is what boxplots display as a horizontal line.

---

## 11.3 The inter quartile range (IQR)

The box in boxplots is defined by the first and third quartile. These are meant to provide an idea of the variability in the data: 50% of the data is within this range. The difference between the 3rd and 1st quartile (or 75th and 25th percentiles) is referred to as the inter quartile range (IQR). As is the case with the median, this quantity will be robust to outliers as large values do not affect it. We can do some math to see that for normally distributed data, the IQR / 1.349 approximates the standard deviation of the data had an outlier not

been present. We can see that this works well in our example since we get a standard deviation estimate of:

```
IQR(outlier_example) / 1.349
#> [1] 0.245
```

which is about 3 inches.

---

## 11.4 Tukey's definition of an outlier

In R, points falling outside the whiskers of the boxplot are referred to as *outliers*. This definition of outlier was introduced by Tukey. The top whisker ends at the 75th percentile plus  $1.5 \times \text{IQR}$ . Similarly the bottom whisker ends at the 25th percentile minus  $1.5 \times \text{IQR}$ . If we define the first and third quartiles as  $Q_1$  and  $Q_3$ , respectively, then an outlier is anything outside the range:

$$[Q_1 - 1.5 \times (Q_3 - Q_1), Q_3 + 1.5 \times (Q_3 - Q_1)].$$

When the data is normally distributed, the standard units of these values are:

```
q3 <- qnorm(0.75)
q1 <- qnorm(0.25)
iqr <- q3 - q1
r <- c(q1 - 1.5*iqr, q3 + 1.5*iqr)
r
#> [1] -2.7 2.7
```

Using the `pnorm` function, we see that 99.3% of the data falls in this interval.

Keep in mind that this is not such an extreme event: if we have 1000 data points that are normally distributed, we expect to see about 7 outside of this range. But these would not be outliers since we expect to see them under the typical variation.

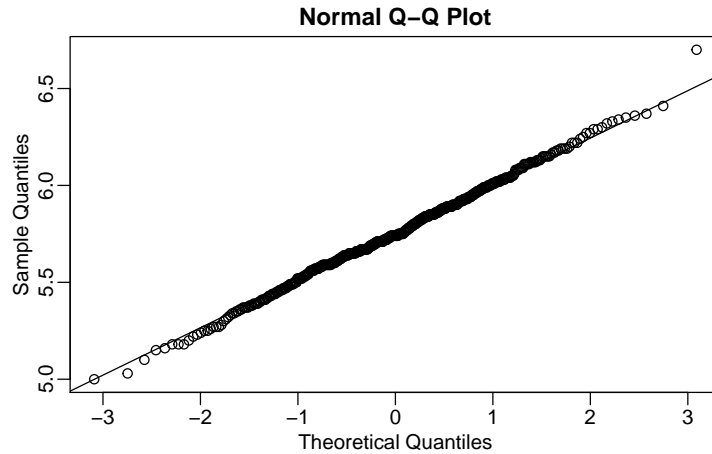
If we want an outlier to be rarer, we can increase the 1.5 to a larger number. Tukey also used 3 and called these *far out* outliers. With a normal distribution, 100% of the data falls in this interval. This translates into about 2 in a million chance of being outside the range. In the `geom_boxplot` function, this can be controlled by the `outlier.size` argument, which defaults to 1.5.

The 180 inches measurement is well beyond the range of the height data:

```
max_height <- quantile(outlier_example, 0.75) + 3*IQR(outlier_example)
max_height
#> 75%
#> 6.91
```

If we take this value out, we can see that the data is in fact normally distributed as expected:

```
x <- outlier_example[outlier_example < max_height]
qqnorm(x)
qqline(x)
```



## 11.5 Median absolute deviation

Another way to robustly estimate the standard deviation in the presence of outliers is to use the median absolute deviation (MAD). To compute the MAD, we first compute the median, and then for each value we compute the distance between that value and the median. The MAD is defined as the median of these distances. For technical reasons not discussed here, this quantity needs to be multiplied by 1.4826 to assure it approximates the actual standard deviation. The `mad` function already incorporates this correction. For the height data, we get a MAD of:

```
mad(outlier_example)
#> [1] 0.237
```

which is about 3 inches.

## 11.6 Exercises

We are going to use the **HistData** package. If it is not installed you can install it like this:

```
install.packages("HistData")
```

Load the height data set and create a vector `x` with just the male heights used in Galton's data on the heights of parents and their children from his historic research on heredity.

```
library(HistData)
data(Galton)
x <- Galton$child
```

1. Compute the average and median of these data.
2. Compute the median and median absolute deviation of these data.
3. Now suppose Galton made a mistake when entering the first value and forgot to use the decimal point. You can imitate this error by typing:

```
x_with_error <- x
x_with_error[1] <- x_with_error[1]*10
```

How many inches does the average grow after this mistake?

4. How many inches does the SD grow after this mistake?
5. How many inches does the median grow after this mistake?
6. How many inches does the MAD grow after this mistake?
7. How could you use exploratory data analysis to detect that an error was made?
  - a. Since it is only one value out of many, we will not be able to detect this.
  - b. We would see an obvious shift in the distribution.
  - c. A boxplot, histogram, or qq-plot would reveal a clear outlier.
  - d. A scatterplot would show high levels of measurement error.
8. How much can the average accidentally grow with mistakes like this? Write a function called `error_avg` that takes a value `k` and returns the average of the vector `x` after the first entry changed to `k`. Show the results for `k=10000` and `k=-10000`.

---

## 11.7 Case study: self-reported student heights

The heights we have been looking at are not the original heights reported by students. The original reported heights are also included in the **dslabs** package and can be loaded like this:

```
library(dslabs)
data("reported_heights")
```

Height is a character vector so we create a new column with the numeric version:

```
reported_heights <- reported_heights %>%
 mutate(original_heights = height, height = as.numeric(height))
#> Warning: NAs introduced by coercion
```

Note that we get a warning about NAs. This is because some of the self reported heights were not numbers. We can see why we get these:

```
reported_heights %>% filter(is.na(height)) %>% head()
#> time_stamp sex height original_heights
#> 1 2014-09-02 15:16:28 Male NA 5' 4"
#> 2 2014-09-02 15:16:37 Female NA 165cm
#> 3 2014-09-02 15:16:52 Male NA 5'7
#> 4 2014-09-02 15:16:56 Male NA >9000
#> 5 2014-09-02 15:16:56 Male NA 5'7"
#> 6 2014-09-02 15:17:09 Female NA 5'3"
```

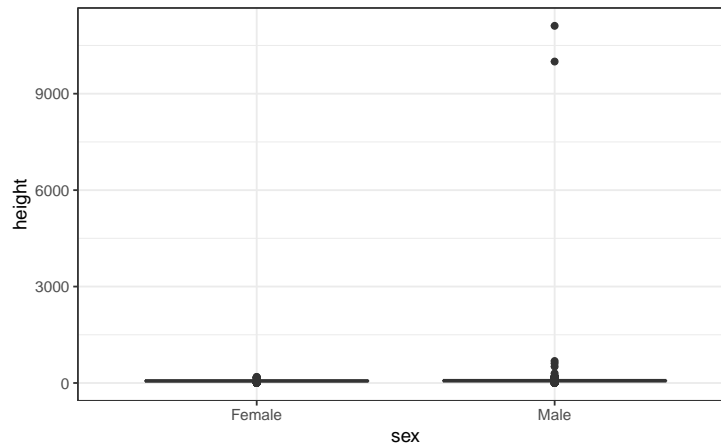
Some students self-reported their heights using feet and inches rather than just inches. Others used centimeters and others were just trolling. For now we will remove these entries:

```
reported_heights <- filter(reported_heights, !is.na(height))
```

If we compute the average and standard deviation, we notice that we obtain strange results. The average and standard deviation are different from the median and MAD:

```
reported_heights %>%
 group_by(sex) %>%
 summarize(average = mean(height), sd = sd(height),
 median = median(height), MAD = mad(height))
#> # A tibble: 2 x 5
#> sex average sd median MAD
#> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 Female 63.4 27.9 64.2 4.05
#> 2 Male 103. 530. 70 4.45
```

This suggests that we have outliers, which is confirmed by creating a boxplot:



We can see some rather extreme values. To see what these values are, we can quickly look at the largest values using the `arrange` function:

```
reported_heights %>% arrange(desc(height)) %>% top_n(10, height)
#> time_stamp sex height original_heights
#> 1 2014-09-03 23:55:37 Male 11111 11111
#> 2 2016-04-10 22:45:49 Male 10000 10000
#> 3 2015-08-10 03:10:01 Male 684 684
#> 4 2015-02-27 18:05:06 Male 612 612
#> 5 2014-09-02 15:16:41 Male 511 511
#> 6 2014-09-07 20:53:43 Male 300 300
#> 7 2014-11-28 12:18:40 Male 214 214
#> 8 2017-04-03 16:16:57 Male 210 210
#> 9 2015-11-24 10:39:45 Male 192 192
#> 10 2014-12-26 10:00:12 Male 190 190
#> 11 2016-11-06 10:21:02 Female 190 190
```

The first seven entries look like strange errors. However, the next few look like they were entered as centimeters instead of inches. Since 184 cm is equivalent to six feet tall, we suspect that 184 was actually meant to be 72 inches.

We can review all the nonsensical answers by looking at the data considered to be *far out* by Tukey:

```
whisker <- 3*IQR(reported_heights$height)
max_height <- quantile(reported_heights$height, .75) + whisker
min_height <- quantile(reported_heights$height, .25) - whisker
reported_heights %>%
 filter(!between(height, min_height, max_height)) %>%
 select(original_heights) %>%
 head(n=10) %>% pull(original_heights)
#> [1] "6" "5.3" "511" "6" "2" "5.25" "5.5" "11111"
#> [9] "6" "6.5"
```

Examining these heights carefully, we see two common mistakes: entries in centimeters, which turn out to be too large, and entries of the form *x.y* with *x* and *y* representing feet and inches, respectively, which turn out to be too small. Some of the even smaller values, such as 1.6, could be entries in meters.

In the Data Wrangling part of this book we will learn techniques for correcting these values and converting them into inches. Here we were able to detect this problem using careful data exploration to uncover issues with the data: the first step in the great majority of data science projects.



Part III

# Statistics with R





# 12

---

## *Introduction to statistics with R*

---

Data analysis is one of the main focuses of this book. While the computing tools we have introduced are relatively recent developments, data analysis has been around for over a century. Throughout the years, data analysts working on specific projects have come up with ideas and concepts that generalize across many applications. They have also identified common ways to get fooled by apparent patterns in the data and important mathematical realities that are not immediately obvious. The accumulation of these ideas and insights has given rise to the discipline of statistics, which provides a mathematical framework that greatly facilitates the description and formal evaluation of these ideas.

To avoid repeating common mistakes and wasting time reinventing the wheel, it is important for a data analyst to have an in-depth understanding of statistics. However, due to the maturity of the discipline, there are dozens of excellent books already published on this topic and we therefore do not focus on describing the mathematical framework here. Instead, we introduce concepts briefly and then provide detailed case studies demonstrating how statistics is used in data analysis along with R code implementing these ideas. We also use R code to help elucidate some of the main statistical concepts that are usually described using mathematics. We highly recommend complementing this part of the book with a basic statistics textbook. Two examples are *Statistics* by Freedman, Pisani, and Purves and *Statistical Inference* by Casella and Berger. The specific concepts covered in this part are Probability, Statistical Inference, Statistical Models, Regression, and Linear Models, which are major topics covered in a statistics course. The case studies we present relate to the financial crisis, forecasting election results, understanding heredity, and building a baseball team.



# 13

## *Probability*

---

In games of chance, probability has a very intuitive definition. For instance, we know what it means that the chance of a pair of dice coming up seven is 1 in 6. However, this is not the case in other contexts. Today probability theory is being used much more broadly with the word *probability* commonly used in everyday language. Google’s auto-complete of “What are the chances of” give us: “having twins”, “rain today”, “getting struck by lightning”, and “getting cancer”. One of the goals of this part of the book is to help us understand how probability is useful to understand and describe real-world events when performing data analysis.

Because knowing how to compute probabilities gives you an edge in games of chance, throughout history many smart individuals, including famous mathematicians such as Cardano, Fermat, and Pascal, spent time and energy thinking through the math of these games. As a result, Probability Theory was born. Probability continues to be highly useful in modern games of chance. For example, in poker, we can compute the probability of winning a hand based on the cards on the table. Also, casinos rely on probability theory to develop games that almost certainly guarantee a profit.

Probability theory is useful in many other contexts and, in particular, in areas that depend on data affected by chance in some way. All of the other chapters in this part build upon probability theory. Knowledge of probability is therefore indispensable for data science.

---

### 13.1 Discrete probability

We start by covering some basic principles related to categorical data. The subset of probability is referred to as *discrete probability*. It will help us understand the probability theory we will later introduce for numeric and continuous data, which is much more common in data science applications. Discrete probability is more useful in card games and therefore we use these as examples.

#### 13.1.1 Relative frequency

The word probability is used in everyday language. Answering questions about probability is often hard, if not impossible. Here we discuss a mathematical definition of *probability* that does permit us to give precise answers to certain questions.

For example, if I have 2 red beads and 3 blue beads inside an urn<sup>1</sup> (most probability books

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Urn\\_problem](https://en.wikipedia.org/wiki/Urn_problem)

use this archaic term, so we do too) and I pick one at random, what is the probability of picking a red one? Our intuition tells us that the answer is  $2/5$  or 40%. A precise definition can be given by noting that there are five possible outcomes of which two satisfy the condition necessary for the event “pick a red bead”. Since each of the five outcomes has the same chance of occurring, we conclude that the probability is .4 for red and .6 for blue.

A more tangible way to think about the probability of an event is as the proportion of times the event occurs when we repeat the experiment an infinite number of times, independently, and under the same conditions.

### 13.1.2 Notation

We use the notation  $\Pr(A)$  to denote the probability of event  $A$  happening. We use the very general term *event* to refer to things that can happen when something occurs by chance. In our previous example, the event was “picking a red bead”. In a political poll in which we call 100 likely voters at random, an example of an event is “calling 48 Democrats and 52 Republicans”.

In data science applications, we will often deal with continuous variables. These events will often be things like “is this person taller than 6 feet”. In this case, we write events in a more mathematical form:  $X \geq 6$ . We will see more of these examples later. Here we focus on categorical data.

### 13.1.3 Probability distributions

If we know the relative frequency of the different categories, defining a distribution for categorical outcomes is relatively straightforward. We simply assign a probability to each category. In cases that can be thought of as beads in an urn, for each bead type, their proportion defines the distribution.

If we are randomly calling likely voters from a population that is 44% Democrat, 44% Republican, 10% undecided, and 2% Green Party, these proportions define the probability for each group. The probability distribution is:

|                                    |   |      |
|------------------------------------|---|------|
| $\Pr(\text{picking a Republican})$ | = | 0.44 |
| $\Pr(\text{picking a Democrat})$   | = | 0.44 |
| $\Pr(\text{picking an undecided})$ | = | 0.10 |
| $\Pr(\text{picking a Green})$      | = | 0.02 |

---

## 13.2 Monte Carlo simulations for categorical data

Computers provide a way to actually perform the simple random experiment described above: pick a bead at random from a bag that contains three blue beads and two red ones. Random number generators permit us to mimic the process of picking at random.

An example is the `sample` function in R. We demonstrate its use in the code below. First, we use the function `rep` to generate the urn:

```
beads <- rep(c("red", "blue"), times = c(2,3))
beads
#> [1] "red" "red" "blue" "blue" "blue"
```

and then use `sample` to pick a bead at random:

```
sample(beads, 1)
#> [1] "red"
```

This line of code produces one random outcome. We want to repeat this experiment an infinite number of times, but it is impossible to repeat forever. Instead, we repeat the experiment a large enough number of times to make the results practically equivalent to repeating forever. **This is an example of a *Monte Carlo* simulation.**

Much of what mathematical and theoretical statisticians study, which we do not cover in this book, relates to providing rigorous definitions of “practically equivalent” as well as studying how close a large number of experiments gets us to what happens in the limit. Later in this section, we provide a practical approach to deciding what is “large enough”.

To perform our first Monte Carlo simulation, we use the `replicate` function, which permits us to repeat the same task any number of times. Here, we repeat the random event  $B = 10,000$  times:

```
B <- 10000
events <- replicate(B, sample(beads, 1))
```

We can now see if our definition actually is in agreement with this Monte Carlo simulation approximation. We can use `table` to see the distribution:

```
tab <- table(events)
tab
#> events
#> blue red
#> 6059 3941
```

and `prop.table` gives us the proportions:

```
prop.table(tab)
#> events
#> blue red
#> 0.606 0.394
```

The numbers above are the estimated probabilities provided by this Monte Carlo simulation. Statistical theory, not covered here, tells us that as  $B$  gets larger, the estimates get closer to  $3/5=0.6$  and  $2/5=0.4$ .

Although this is a simple and not very useful example, we will use Monte Carlo simulations to estimate probabilities in cases in which it is harder to compute the exact ones. Before delving into more complex examples, we use simple ones to demonstrate the computing tools available in R.

### 13.2.1 Setting the random seed

Before we continue, we will briefly explain the following important line of code:

```
set.seed(1986)
```

Throughout this book, we use random number generators. This implies that many of the results presented can actually change by chance, which then suggests that a frozen version of the book may show a different result than what you obtain when you try to code as shown in the book. This is actually fine since the results are random and change from time to time. However, if you want to ensure that results are exactly the same every time you run them, you can set R's random number generation seed to a specific number. Above we set it to 1986. We want to avoid using the same seed everytime. A popular way to pick the seed is the year - month - day. For example, we picked 1986 on December 20, 2018:  $2018 - 12 - 20 = 1986$ .

You can learn more about setting the seed by looking at the documentation:

```
?set.seed
```

In the exercises, we may ask you to set the seed to assure that the results you obtain are exactly what we expect them to be.

### 13.2.2 With and without replacement

The function `sample` has an argument that permits us to pick more than one element from the urn. However, by default, this selection occurs *without replacement*: after a bead is selected, it is not put back in the bag. Notice what happens when we ask to randomly select five beads:

```
sample(beads, 5)
#> [1] "red" "blue" "blue" "blue" "red"
sample(beads, 5)
#> [1] "red" "red" "blue" "blue" "blue"
sample(beads, 5)
#> [1] "blue" "red" "blue" "red" "blue"
```

This results in rearrangements that always have three blue and two red beads. If we ask that six beads be selected, we get an error:

```
sample(beads, 6)
```

```
Error in sample.int(length(x), size, replace, prob) : cannot take a
sample larger than the population when 'replace = FALSE'
```

However, the `sample` function can be used directly, without the use of `replicate`, to repeat the same experiment of picking 1 out of the 5 beads, continually, under the same conditions. To do this, we sample *with replacement*: return the bead back to the urn after selecting it. We can tell `sample` to do this by changing the `replace` argument, which defaults to `FALSE`, to `replace = TRUE`:

```
events <- sample(beads, B, replace = TRUE)
prop.table(table(events))
#> events
#> blue red
#> 0.602 0.398
```

Not surprisingly, we get results very similar to those previously obtained with `replicate`.

---

### 13.3 Independence

We say two events are independent if the outcome of one does not affect the other. The classic example is coin tosses. Every time we toss a fair coin, the probability of seeing heads is  $1/2$  regardless of what previous tosses have revealed. The same is true when we pick beads from an urn with replacement. In the example above, the probability of red is 0.40 regardless of previous draws.

Many examples of events that are not independent come from card games. When we deal the first card, the probability of getting a King is  $1/13$  since there are thirteen possibilities: Ace, Deuce, Three, ..., Ten, Jack, Queen, King, and Ace. Now if we deal a King for the first card, and don't replace it into the deck, the probabilities of a second card being a King is less because there are only three Kings left: the probability is 3 out of 51. These events are therefore **not independent**: the first outcome affected the next one.

To see an extreme case of non-independent events, consider our example of drawing five beads at random **without** replacement:

```
x <- sample(beads, 5)
```

If you have to guess the color of the first bead, you will predict blue since blue has a 60% chance. But if I show you the result of the last four outcomes:

```
x[2:5]
#> [1] "blue" "blue" "blue" "red"
```

would you still guess blue? Of course not. Now you know that the probability of red is 1 since the only bead left is red. The events are not independent, so the probabilities change.

---

### 13.4 Conditional probabilities

When events are not independent, *conditional probabilities* are useful. We already saw an example of a conditional probability: we computed the probability that a second dealt card is a King given that the first was a King. In probability, we use the following notation:



$$\Pr(\text{Card 2 is a king} \mid \text{Card 1 is a king}) = 3/51$$

We use the  $\mid$  as shorthand for “given that” or “conditional on”.

When two events, say  $A$  and  $B$ , are independent, we have:

$$\Pr(A \mid B) = \Pr(A)$$

This is the mathematical way of saying: the fact that  $B$  happened does not affect the probability of  $A$  happening. In fact, this can be considered the mathematical definition of independence.

## 13.5 Addition and multiplication rules

### 13.5.1 Multiplication rule

If we want to know the probability of two events, say  $A$  and  $B$ , occurring, we can use the multiplication rule:

$$\Pr(A \text{ and } B) = \Pr(A)\Pr(B \mid A)$$

Let’s use Blackjack as an example. In Blackjack, you are assigned two random cards. After you see what you have, you can ask for more. The goal is to get closer to 21 than the dealer, without going over. Face cards are worth 10 points and Aces are worth 11 or 1 (you choose).

So, in a Blackjack game, to calculate the chances of getting a 21 by drawing an Ace and then a face card, we compute the probability of the first being an Ace and multiply by the probability of drawing a face card or a 10 given that the first was an Ace:  $1/13 \times 16/51 \approx 0.025$

The multiplication rule also applies to more than two events. We can use induction to expand for more events:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B \mid A)\Pr(C \mid A \text{ and } B)$$

### 13.5.2 Multiplication rule under independence

When we have independent events, then the multiplication rule becomes simpler:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B)\Pr(C)$$

But we have to be very careful before using this since assuming independence can result in very different and incorrect probability calculations when we don’t actually have independence.

As an example, imagine a court case in which the suspect was described as having a mustache and a beard. The defendant has a mustache and a beard and the prosecution brings in

an “expert” to testify that  $1/10$  men have beards and  $1/5$  have mustaches, so using the multiplication rule we conclude that only  $1/10 \times 1/5$  or  $0.02$  have both.

But to multiply like this we need to assume independence! Say the conditional probability of a man having a mustache conditional on him having a beard is  $.95$ . So the correct calculation probability is much higher:  $1/10 \times 95/100 = 0.095$ .

The multiplication rule also gives us a general formula for computing conditional probabilities:

$$\Pr(B \mid A) = \frac{\Pr(A \text{ and } B)}{\Pr(A)}$$

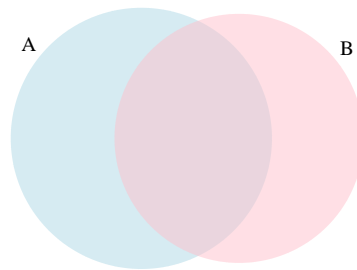
To illustrate how we use these formulas and concepts in practice, we will use several examples related to card games.

### 13.5.3 Addition rule

The addition rule tells us that:

$$\Pr(A \text{ or } B) = \Pr(A) + \Pr(B) - \Pr(A \text{ and } B)$$

This rule is intuitive: think of a Venn diagram. If we simply add the probabilities, we count the intersection twice so we need to subtract one instance.




---

## 13.6 Combinations and permutations

In our very first example, we imagined an urn with five beads. As a reminder, to compute the probability distribution of one draw, we simply listed out all the possibilities. There were 5 and so then, for each event, we counted how many of these possibilities were associated with the event. The resulting probability of choosing a blue bead is  $3/5$  because out of the five possible outcomes, three were blue.

For more complicated cases, the computations are not as straightforward. For instance, what

is the probability that if I draw five cards without replacement, I get all cards of the same suit, what is known as a “flush” in poker? In a discrete probability course you learn theory on how to make these computations. Here we focus on how to use R code to compute the answers.

First, let’s construct a deck of cards. For this, we will use the `expand.grid` and `paste` functions. We use `paste` to create strings by joining smaller strings. To do this, we take the number and suit of a card and create the card name like this:

```
number <- "Three"
suit <- "Hearts"
paste(number, suit)
#> [1] "Three Hearts"
```

`paste` also works on pairs of vectors performing the operation element-wise:

```
paste(letters[1:5], as.character(1:5))
#> [1] "a 1" "b 2" "c 3" "d 4" "e 5"
```

The function `expand.grid` gives us all the combinations of entries of two vectors. For example, if you have blue and black pants and white, grey, and plaid shirts, all your combinations are:

```
expand.grid(pants = c("blue", "black"), shirt = c("white", "grey", "plaid"))
#> pants shirt
#> 1 blue white
#> 2 black white
#> 3 blue grey
#> 4 black grey
#> 5 blue plaid
#> 6 black plaid
```

Here is how we generate a deck of cards:

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
numbers <- c("Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
 "Eight", "Nine", "Ten", "Jack", "Queen", "King")
deck <- expand.grid(number=numbers, suit=suits)
deck <- paste(deck$number, deck$suit)
```

With the deck constructed, we can double check that the probability of a King in the first card is  $1/13$  by computing the proportion of possible outcomes that satisfy our condition:

```
kings <- paste("King", suits)
mean(deck %in% kings)
#> [1] 0.0769
```

Now, how about the conditional probability of the second card being a King given that the first was a King? Earlier, we deduced that if one King is already out of the deck and there are 51 left, then this probability is  $3/51$ . Let’s confirm by listing out all possible outcomes.

To do this, we can use the `permutations` function from the **gtools** package. For any list of size `n`, this function computes all the different combinations we can get when we select `r` items. Here are all the ways we can choose two numbers from a list consisting of 1,2,3:

```
library(gtools)
permutations(3, 2)
#> [,1] [,2]
#> [1,] 1 2
#> [2,] 1 3
#> [3,] 2 1
#> [4,] 2 3
#> [5,] 3 1
#> [6,] 3 2
```

Notice that the order matters here: 3,1 is different than 1,3. Also, note that (1,1), (2,2), and (3,3) do not appear because once we pick a number, it can't appear again.

Optionally, we can add a vector. If you want to see five random seven digit phone numbers out of all possible phone numbers (without repeats), you can type:

```
all_phone_numbers <- permutations(10, 7, v = 0:9)
n <- nrow(all_phone_numbers)
index <- sample(n, 5)
all_phone_numbers[index,]
#> [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,] 1 3 8 0 6 7 5
#> [2,] 2 9 1 6 4 8 0
#> [3,] 5 1 6 0 9 8 2
#> [4,] 7 4 6 0 2 8 1
#> [5,] 4 6 5 9 2 8 0
```

Instead of using the numbers 1 through 10, the default, it uses what we provided through `v`: the digits 0 through 9.

To compute all possible ways we can choose two cards when the order matters, we type:

```
hands <- permutations(52, 2, v = deck)
```

This is a matrix with two columns and 2652 rows. With a matrix we can get the first and second cards like this:

```
first_card <- hands[,1]
second_card <- hands[,2]
```

Now the cases for which the first hand was a King can be computed like this:

```
kings <- paste("King", suits)
sum(first_card %in% kings)
#> [1] 204
```

To get the conditional probability, we compute what fraction of these have a King in the second card:

```
sum(first_card%in%kings & second_card%in%kings) / sum(first_card%in%kings)
#> [1] 0.0588
```

which is exactly 3/51, as we had already deduced. Notice that the code above is equivalent to:

```
mean(first_card%in%kings & second_card%in%kings) / mean(first_card%in%kings)
#> [1] 0.0588
```

which uses `mean` instead of `sum` and is an R version of:

$$\frac{\Pr(A \text{ and } B)}{\Pr(A)}$$

How about if the order doesn't matter? For example, in Blackjack if you get an Ace and a face card in the first draw, it is called a *Natural 21* and you win automatically. If we wanted to compute the probability of this happening, we would enumerate the *combinations*, not the permutations, since the order does not matter.

```
combinations(3,2)
#> [,1] [,2]
#> [1,] 1 2
#> [2,] 1 3
#> [3,] 2 3
```

In the second line, the outcome does not include (2,1) because (1,2) already was enumerated. The same applies to (3,1) and (3,2).

So to compute the probability of a *Natural 21* in Blackjack, we can do this:

```
aces <- paste("Ace", suits)

facecard <- c("King", "Queen", "Jack", "Ten")
facecard <- expand.grid(number = facecard, suit = suits)
facecard <- paste(facecard$number, facecard$suit)

hands <- combinations(52, 2, v = deck)
mean(hands[,1] %in% aces & hands[,2] %in% facecard)
#> [1] 0.0483
```

In the last line, we assume the Ace comes first. This is only because we know the way `combination` enumerates possibilities and it will list this case first. But to be safe, we could have written this and produced the same answer:

```
mean((hands[,1] %in% aces & hands[,2] %in% facecard) |
 (hands[,2] %in% aces & hands[,1] %in% facecard))
#> [1] 0.0483
```

### 13.6.1 Monte Carlo example

Instead of using `combinations` to deduce the exact probability of a Natural 21, we can use a Monte Carlo to estimate this probability. In this case, we draw two cards over and over and keep track of how many 21s we get. We can use the function `sample` to draw two cards without replacements:

```
hand <- sample(deck, 2)
hand
#> [1] "Queen Clubs" "Seven Spades"
```

And then check if one card is an Ace and the other a face card or a 10. Going forward, we include 10 when we say *face card*. Now we need to check both possibilities:

```
(hands[1] %in% aces & hands[2] %in% facecard) |
 (hands[2] %in% aces & hands[1] %in% facecard)
#> [1] FALSE
```

If we repeat this 10,000 times, we get a very good approximation of the probability of a Natural 21.

Let's start by writing a function that draws a hand and returns TRUE if we get a 21. The function does not need any arguments because it uses objects defined in the global environment.

```
blackjack <- function(){
 hand <- sample(deck, 2)
 (hand[1] %in% aces & hand[2] %in% facecard) |
 (hand[2] %in% aces & hand[1] %in% facecard)
}
```

Here we do have to check both possibilities: Ace first or Ace second because we are not using the `combinations` function. The function returns TRUE if we get a 21 and FALSE otherwise:

```
blackjack()
#> [1] FALSE
```

Now we can play this game, say, 10,000 times:

```
B <- 10000
results <- replicate(B, blackjack())
mean(results)
#> [1] 0.0475
```

---

## 13.7 Examples

In this section, we describe two discrete probability popular examples: the Monty Hall problem and the birthday problem. We use R to help illustrate the mathematical concepts.

### 13.7.1 Monty Hall problem

In the 1970s, there was a game show called “Let’s Make a Deal” and Monty Hall was the host. At some point in the game, contestants were asked to pick one of three doors. Behind one door there was a prize. The other doors had a goat behind them to show the contestant they had lost. After the contestant picked a door, before revealing whether the chosen door contained a prize, Monty Hall would open one of the two remaining doors and show the contestant there was no prize behind that door. Then he would ask “Do you want to switch doors?” What would you do?

We can use probability to show that if you stick with the original door choice, your chances of winning a prize remain 1 in 3. However, if you switch to the other door, your chances of winning double to 2 in 3! This seems counterintuitive. Many people incorrectly think both chances are 1 in 2 since you are choosing between 2 options. You can watch a detailed mathematical explanation on Khan Academy<sup>2</sup> or read one on Wikipedia<sup>3</sup>. Below we use a Monte Carlo simulation to see which strategy is better. Note that this code is written longer than it should be for pedagogical purposes.

Let’s start with the stick strategy:

```
B <- 10000
monty_hall <- function(strategy){
 doors <- as.character(1:3)
 prize <- sample(c("car", "goat", "goat"))
 prize_door <- doors[prize == "car"]
 my_pick <- sample(door, 1)
 show <- sample(door[!door %in% c(my_pick, prize_door)], 1)
 stick <- my_pick
 stick == prize_door
 switch <- door[!door %in% c(my_pick, show)]
 choice <- ifelse(strategy == "stick", stick, switch)
 choice == prize_door
}
stick <- replicate(B, monty_hall("stick"))
mean(stick)
#> [1] 0.342
switch <- replicate(B, monty_hall("switch"))
mean(switch)
#> [1] 0.668
```

As we write the code, we note that the lines starting with `my_pick` and `show` have no influence on the last logical operation when we stick to our original choice anyway. From this we should realize that the chance is 1 in 3, what we began with. When we switch, the Monte Carlo estimate confirms the 2/3 calculation. This helps us gain some insight by showing that we are removing a door, `show`, that is definitely not a winner from our choices. We also see that unless we get it right when we first pick, you win:  $1 - 1/3 = 2/3$ .

<sup>2</sup><https://www.khanacademy.org/math/prec calculus/prob-comb/dependent-events-prec calc/v/monty-hall-problem>

<sup>3</sup>[https://en.wikipedia.org/wiki/Monty\\_Hall\\_problem](https://en.wikipedia.org/wiki/Monty_Hall_problem)

### 13.7.2 Birthday problem

Suppose you are in a classroom with 50 people. If we assume this is a randomly selected group of 50 people, what is the chance that at least two people have the same birthday? Although it is somewhat advanced, we can deduce this mathematically. We will do this later. Here we use a Monte Carlo simulation. For simplicity, we assume nobody was born on February 29. This actually doesn't change the answer much.

First, note that birthdays can be represented as numbers between 1 and 365, so a sample of 50 birthdays can be obtained like this:

```
n <- 50
bdays <- sample(1:365, n, replace = TRUE)
```

To check if in this particular set of 50 people we have at least two with the same birthday, we can use the function `duplicated`, which returns `TRUE` whenever an element of a vector is a duplicate. Here is an example:

```
duplicated(c(1,2,3,1,4,3,5))
#> [1] FALSE FALSE FALSE TRUE FALSE TRUE FALSE
```

The second time 1 and 3 appear, we get a `TRUE`. So to check if two birthdays were the same, we simply use the `any` and `duplicated` functions like this:

```
any(duplicated(bdays))
#> [1] TRUE
```

In this case, we see that it did happen. At least two people had the same birthday.

To estimate the probability of a shared birthday in the group, we repeat this experiment by sampling sets of 50 birthdays over and over:

```
B <- 10000
same_birthday <- function(n){
 bdays <- sample(1:365, n, replace=TRUE)
 any(duplicated(bdays))
}
results <- replicate(B, same_birthday(50))
mean(results)
#> [1] 0.969
```

Were you expecting the probability to be this high?

People tend to underestimate these probabilities. To get an intuition as to why it is so high, think about what happens when the group size is close to 365. At this stage, we run out of days and the probability is one.

Say we want to use this knowledge to bet with friends about two people having the same birthday in a group of people. When are the chances larger than 50%? Larger than 75%?

Let's create a look-up table. We can quickly create a function to compute this for any group size:



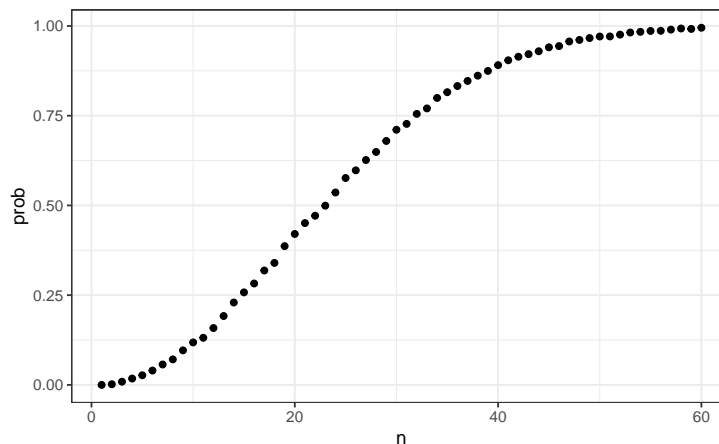
```
compute_prob <- function(n, B=10000){
 results <- replicate(B, same_birthday(n))
 mean(results)
}
```

Using the function `sapply`, we can perform element-wise operations on any function:

```
n <- seq(1,60)
prob <- sapply(n, compute_prob)
```

We can now make a plot of the estimated probabilities of two people having the same birthday in a group of size  $n$ :

```
library(tidyverse)
prob <- sapply(n, compute_prob)
qplot(n, prob)
```



Now let's compute the exact probabilities rather than use Monte Carlo approximations. Not only do we get the exact answer using math, but the computations are much faster since we don't have to generate experiments.

To make the math simpler, instead of computing the probability of it happening, we will compute the probability of it not happening. For this, we use the multiplication rule.

Let's start with the first person. The probability that person 1 has a unique birthday is 1. The probability that person 2 has a unique birthday, given that person 1 already took one, is  $364/365$ . Then, given that the first two people have unique birthdays, person 3 is left with 363 days to choose from. We continue this way and find the chances of all 50 people having a unique birthday is:

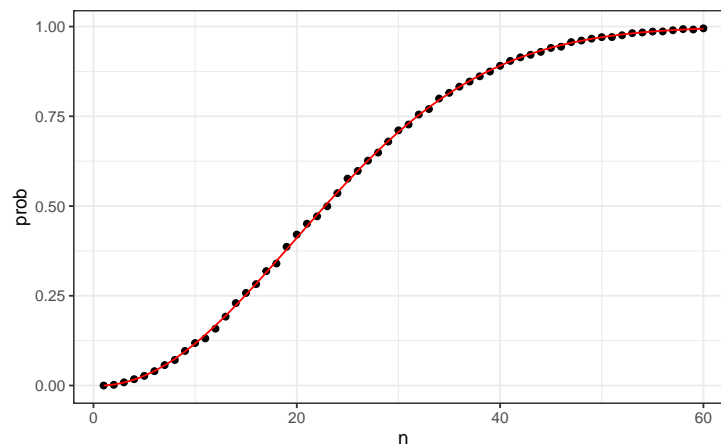
$$1 \times \frac{364}{365} \times \frac{363}{365} \cdots \frac{365 - n + 1}{365}$$

We can write a function that does this for any number:

```

exact_prob <- function(n){
 prob_unique <- seq(365, 365-n+1)/365
 1 - prod(prob_unique)
}
eprob <- sapply(n, exact_prob)
qplot(n, prob) + geom_line(aes(n, eprob), col = "red")

```



This plot shows that the Monte Carlo simulation provided a very good estimate of the exact probability. Had it not been possible to compute the exact probabilities, we would have still been able to accurately estimate the probabilities.

## 13.8 Infinity in practice

The theory described here requires repeating experiments over and over forever. In practice we can't do this. In the examples above, we used  $B = 10,000$  Monte Carlo experiments and it turned out that this provided accurate estimates. The larger this number, the more accurate the estimate becomes until the approximation is so good that your computer can't tell the difference. But in more complex calculations, 10,000 may not be nearly enough. Also, for some calculations, 10,000 experiments might not be computationally feasible. In practice, we won't know what the answer is, so we won't know if our Monte Carlo estimate is accurate. We know that the larger  $B$ , the better the approximation. But how big do we need it to be? This is actually a challenging question and answering it often requires advanced theoretical statistics training.

One practical approach we will describe here is to check for the stability of the estimate. The following is an example with the birthday problem for a group of 25 people.

```

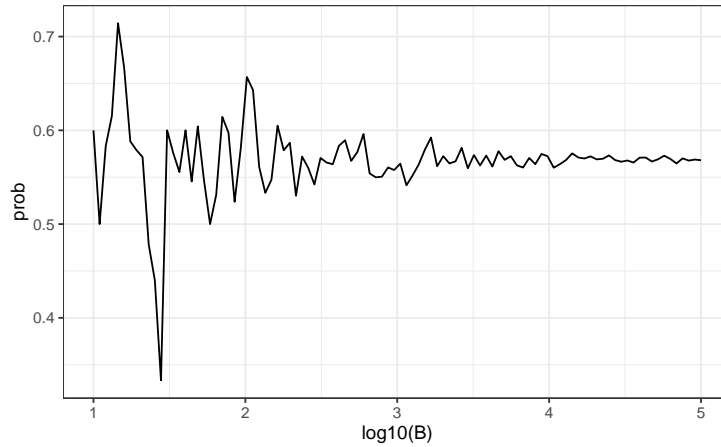
B <- 10^seq(1, 5, len = 100)
compute_prob <- function(B, n=25){
 same_day <- replicate(B, same_birthday(n))
 mean(same_day)
}

```

```

}
prob <- sapply(B, compute_prob)
qplot(log10(B), prob, geom = "line")

```



In this plot, we can see that the values start to stabilize (that is, they vary less than .01) around 1000. Note that the exact probability, which we know in this case, is 0.569.

---

### 13.9 Exercises

- One ball will be drawn at random from a box containing: 3 cyan balls, 5 magenta balls, and 7 yellow balls. What is the probability that the ball will be cyan?
- What is the probability that the ball will not be cyan?
- Instead of taking just one draw, consider taking two draws. You take the second draw without returning the first draw to the box. We call this sampling **without** replacement. What is the probability that the first draw is cyan and that the second draw is not cyan?
- Now repeat the experiment, but this time, after taking the first draw and recording the color, return it to the box and shake the box. We call this sampling **with** replacement. What is the probability that the first draw is cyan and that the second draw is not cyan?
- Two events  $A$  and  $B$  are independent if  $\Pr(A \text{ and } B) = \Pr(A)P(B)$ . Under which situation are the draws independent?
  - You don't replace the draw.
  - You replace the draw.
  - Neither
  - Both
- Say you've drawn 5 balls from the box, with replacement, and all have been yellow. What is the probability that the next one is yellow?

7. If you roll a 6-sided die six times, what is the probability of not seeing a 6?
8. Two teams, say the Celtics and the Cavs, are playing a seven game series. The Cavs are a better team and have a 60% chance of winning each game. What is the probability that the Celtics win **at least** one game?
9. Create a Monte Carlo simulation to confirm your answer to the previous problem. Use `B <- 10000` simulations. Hint: use the following code to generate the results of the first four games:

```
celtic_wins <- sample(c(0,1), 4, replace = TRUE, prob = c(0.6, 0.4))
```

The Celtics must win one of these 4 games.

10. Two teams, say the Cavs and the Warriors, are playing a seven game championship series. The first to win four games, therefore, wins the series. The teams are equally good so they each have a 50-50 chance of winning each game. If the Cavs lose the first game, what is the probability that they win the series?
11. Confirm the results of the previous question with a Monte Carlo simulation.
12. Two teams, *A* and *B*, are playing a seven game series. Team *A* is better than team *B* and has a  $p > 0.5$  chance of winning each game. Given a value  $p$ , the probability of winning the series for the underdog team *B* can be computed with the following function based on a Monte Carlo simulation:

```
prob_win <- function(p){
 B <- 10000
 result <- replicate(B, {
 b_win <- sample(c(1,0), 7, replace = TRUE, prob = c(1-p, p))
 sum(b_win)>=4
 })
 mean(result)
}
```

Use the function `sapply` to compute the probability, call it `Pr`, of winning for `p <- seq(0.5, 0.95, 0.025)`. Then plot the result.

13. Repeat the exercise above, but now keep the probability fixed at `p <- 0.75` and compute the probability for different series lengths: best of 1 game, 3 games, 5 games,... Specifically, `N <- seq(1, 25, 2)`. Hint: use this function:

```
prob_win <- function(N, p=0.75){
 B <- 10000
 result <- replicate(B, {
 b_win <- sample(c(1,0), N, replace = TRUE, prob = c(1-p, p))
 sum(b_win)>=(N+1)/2
 })
 mean(result)
}
```

### 13.10 Continuous probability

In Section 8.4, we explained why when summarizing a list of numeric values, such as heights, it is not useful to construct a distribution that defines a proportion to each possible outcome. For example, if we measure every single person in a very large population of size  $n$  with extremely high precision, since no two people are exactly the same height, we need to assign the proportion  $1/n$  to each observed value and attain no useful summary at all. Similarly, when defining probability distributions, it is not useful to assign a very small probability to every single height.

Just as when using distributions to summarize numeric data, it is much more practical to define a function that operates on intervals rather than single values. The standard way of doing this is using the *cumulative distribution function* (CDF).

We described empirical cumulative distribution function (eCDF) in Section 8.4 as a basic summary of a list of numeric values. As an example, we earlier defined the height distribution for adult male students. Here, we define the vector  $x$  to contain these heights:

```
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)
```

We defined the empirical distribution function as:

```
F <- function(a) mean(x<=a)
```

which, for any value  $a$ , gives the proportion of values in the list  $x$  that are smaller or equal than  $a$ .

Keep in mind that we have not yet introduced probability in the context of CDFs. Let's do this by asking the following: if I pick one of the male students at random, what is the chance that he is taller than 70.5 inches? Because every student has the same chance of being picked, the answer to this is equivalent to the proportion of students that are taller than 70.5 inches. Using the CDF we obtain an answer by typing:

```
1 - F(70)
#> [1] 0.377
```

Once a CDF is defined, we can use this to compute the probability of any subset. For instance, the probability of a student being between height  $a$  and height  $b$  is:

```
F(b)-F(a)
```

Because we can compute the probability for any possible event this way, the cumulative probability function defines the probability distribution for picking a height at random from our vector of heights  $x$ .

## 13.11 Theoretical continuous distributions

In Section 8.8 we introduced the normal distribution as a useful approximation to many naturally occurring distributions, including that of height. The cumulative distribution for the normal distribution is defined by a mathematical formula which in R can be obtained with the function `pnorm`. We say that a random quantity is normally distributed with average `m` and standard deviation `s` if its probability distribution is defined by:

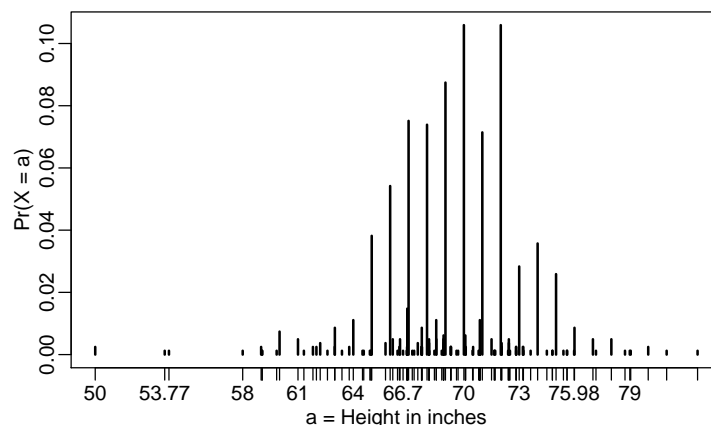
```
F(a) = pnorm(a, m, s)
```

This is useful because if we are willing to use the normal approximation for, say, height, we don't need the entire dataset to answer questions such as: what is the probability that a randomly selected student is taller than 70 inches? We just need the average height and standard deviation:

```
m <- mean(x)
s <- sd(x)
1 - pnorm(70.5, m, s)
#> [1] 0.371
```

### 13.11.1 Theoretical distributions as approximations

The normal distribution is derived mathematically: we do not need data to define it. For practicing data scientists, almost everything we do involves data. Data is always, technically speaking, discrete. For example, we could consider our height data categorical with each specific height a unique category. The probability distribution is defined by the proportion of students reporting each height. Here is a plot of that probability distribution:



While most students rounded up their heights to the nearest inch, others reported values with more precision. One student reported his height to be 69.6850393700787, which is 177 centimeters. The probability assigned to this height is 0.001 or 1 in 812. The probability for

70 inches is much higher at 0.106, but does it really make sense to think of the probability of being exactly 70 inches as being different than 69.6850393700787? Clearly it is much more useful for data analytic purposes to treat this outcome as a continuous numeric variable, keeping in mind that very few people, or perhaps none, are exactly 70 inches, and that the reason we get more values at 70 is because people round to the nearest inch.

With continuous distributions, the probability of a singular value is not even defined. For example, it does not make sense to ask what is the probability that a normally distributed value is 70. Instead, we define probabilities for intervals. We thus could ask what is the probability that someone is between 69.5 and 70.5.

In cases like height, in which the data is rounded, the normal approximation is particularly useful if we deal with intervals that include exactly one round number. For example, the normal distribution is useful for approximating the proportion of students reporting values in intervals like the following three:

```
mean(x <= 68.5) - mean(x <= 67.5)
#> [1] 0.115
mean(x <= 69.5) - mean(x <= 68.5)
#> [1] 0.119
mean(x <= 70.5) - mean(x <= 69.5)
#> [1] 0.122
```

Note how close we get with the normal approximation:

```
pnorm(68.5, m, s) - pnorm(67.5, m, s)
#> [1] 0.103
pnorm(69.5, m, s) - pnorm(68.5, m, s)
#> [1] 0.11
pnorm(70.5, m, s) - pnorm(69.5, m, s)
#> [1] 0.108
```

However, the approximation is not as useful for other intervals. For instance, notice how the approximation breaks down when we try to estimate:

```
mean(x <= 70.9) - mean(x <= 70.1)
#> [1] 0.0222
```

with

```
pnorm(70.9, m, s) - pnorm(70.1, m, s)
#> [1] 0.0836
```

In general, we call this situation *discretization*. Although the true height distribution is continuous, the reported heights tend to be more common at discrete values, in this case, due to rounding. As long as we are aware of how to deal with this reality, the normal approximation can still be a very useful tool.

### 13.11.2 The probability density

For categorical distributions, we can define the probability of a category. For example, a roll of a die, let's call it  $X$ , can be 1,2,3,4,5 or 6. The probability of 4 is defined as:

$$\Pr(X = 4) = 1/6$$

The CDF can then easily be defined:

$$F(4) = \Pr(X \leq 4) = \Pr(X = 4) + \Pr(X = 3) + \Pr(X = 2) + \Pr(X = 1)$$

Although for continuous distributions the probability of a single value  $\Pr(X = x)$  is not defined, there is a theoretical definition that has a similar interpretation. The probability density at  $x$  is defined as the function  $f(x)$  such that:

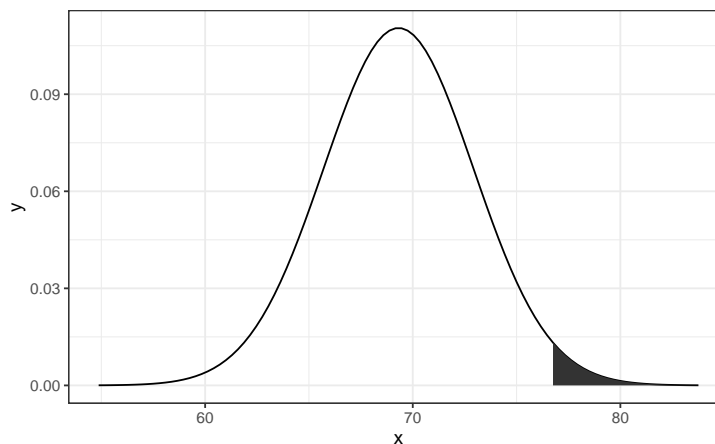
$$F(a) = \Pr(X \leq a) = \int_{-\infty}^a f(x) dx$$

For those that know calculus, remember that the integral is related to a sum: it is the sum of bars with widths approximating 0. If you don't know calculus, you can think of  $f(x)$  as a curve for which the area under that curve up to the value  $a$ , gives you the probability  $\Pr(X \leq a)$ .

For example, to use the normal approximation to estimate the probability of someone being taller than 76 inches, we use:

```
1 - pnorm(76, m, s)
#> [1] 0.0321
```

which mathematically is the grey area below:



The curve you see is the probability density for the normal distribution. In R, we get this using the function `dnorm`.

Although it may not be immediately obvious why knowing about probability densities is useful, understanding this concept will be essential to those wanting to fit models to data for which predefined functions are not available.

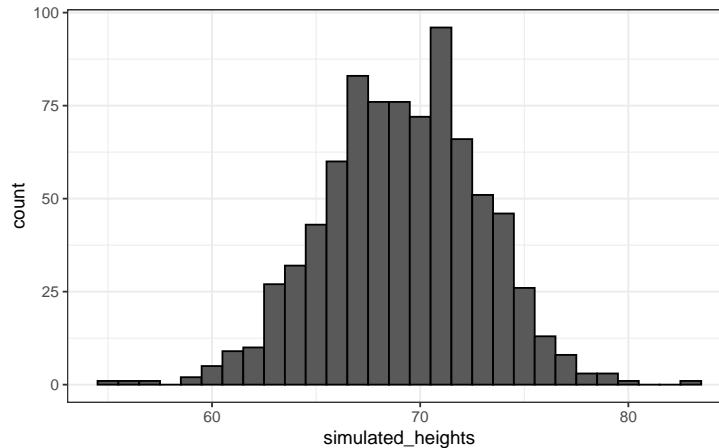


### 13.12 Monte Carlo simulations for continuous variables

R provides functions to generate normally distributed outcomes. Specifically, the `rnorm` function takes three arguments: size, average (defaults to 0), and standard deviation (defaults to 1) and produces random numbers. Here is an example of how we could generate data that looks like our reported heights:

```
n <- length(x)
m <- mean(x)
s <- sd(x)
simulated_heights <- rnorm(n, m, s)
```

Not surprisingly, the distribution looks normal:



This is one of the most useful functions in R as it will permit us to generate data that mimics natural events and answers questions related to what could happen by chance by running Monte Carlo simulations.

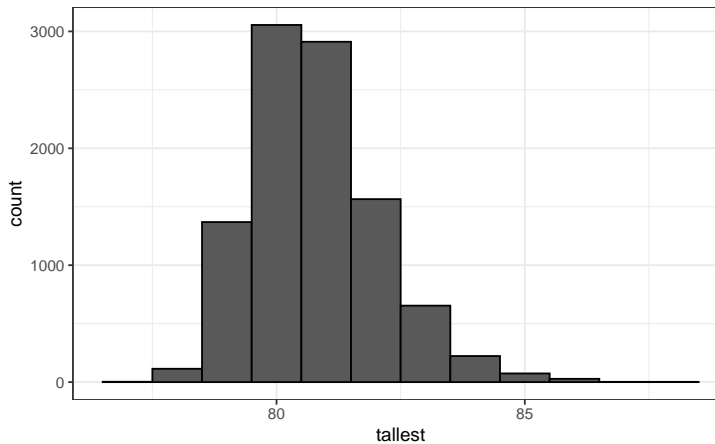
If, for example, we pick 800 males at random, what is the distribution of the tallest person? How rare is a seven footer in a group of 800 males? The following Monte Carlo simulation helps us answer that question:

```
B <- 10000
tallest <- replicate(B, {
 simulated_data <- rnorm(800, m, s)
 max(simulated_data)
})
```

Having a seven footer is quite rare:

```
mean(tallest >= 7*12)
#> [1] 0.0189
```

Here is the resulting distribution:



Note that it does not look normal.

---

### 13.13 Continuous distributions

We introduced the normal distribution in Section 8.8 and used it as an example above. The normal distribution is not the only useful theoretical distribution. Other continuous distributions that we may encounter are the student-t, Chi-square, exponential, gamma, beta, and beta-binomial. R provides functions to compute the density, the quantiles, the cumulative distribution functions and to generate Monte Carlo simulations. R uses a convention that lets us remember the names, namely using the letters **d**, **q**, **p**, and **r** in front of a shorthand for the distribution. We have already seen the functions **dnorm**, **pnorm**, and **rnorm** for the normal distribution. The functions **qnorm** gives us the quantiles. We can therefore draw a distribution like this:

```
x <- seq(-4, 4, length.out = 100)
qplot(x, f, geom = "line", data = data.frame(x, f = dnorm(x)))
```

For the student-t, described later in Section 16.10, the shorthand **t** is used so the functions are **dt** for the density, **qt** for the quantiles, **pt** for the cumulative distribution function, and **rt** for Monte Carlo simulation.

---

### 13.14 Exercises

1. Assume the distribution of female heights is approximated by a normal distribution with a mean of 64 inches and a standard deviation of 3 inches. If we pick a female at random, what is the probability that she is 5 feet or shorter?

2. Assume the distribution of female heights is approximated by a normal distribution with a mean of 64 inches and a standard deviation of 3 inches. If we pick a female at random, what is the probability that she is 6 feet or taller?
3. Assume the distribution of female heights is approximated by a normal distribution with a mean of 64 inches and a standard deviation of 3 inches. If we pick a female at random, what is the probability that she is between 61 and 67 inches?
4. Repeat the exercise above, but convert everything to centimeters. That is, multiply every height, including the standard deviation, by 2.54. What is the answer now?
5. Notice that the answer to the question does not change when you change units. This makes sense since the answer to the question should not be affected by what units we use. In fact, if you look closely, you notice that 61 and 64 are both 1 SD away from the average. Compute the probability that a randomly picked, normally distributed random variable is within 1 SD from the average.
6. To see the math that explains why the answers to questions 3, 4, and 5 are the same, suppose we have a random variable with average  $m$  and standard error  $s$ . Suppose we ask the probability of  $X$  being smaller or equal to  $a$ . Remember that, by definition,  $a$  is  $(a - m)/s$  standard deviations  $s$  away from the average  $m$ . The probability is:

$$\Pr(X \leq a)$$

Now we subtract  $\mu$  to both sides and then divide both sides by  $\sigma$ :

$$\Pr\left(\frac{X - m}{s} \leq \frac{a - m}{s}\right)$$

The quantity on the left is a standard normal random variable. It has an average of 0 and a standard error of 1. We will call it  $Z$ :

$$\Pr\left(Z \leq \frac{a - m}{s}\right)$$

So, no matter the units, the probability of  $X \leq a$  is the same as the probability of a standard normal variable being less than  $(a - m)/s$ . If  $\mu$  is the average and  $\sigma$  the standard error, which of the following R code would give us the right answer in every situation:

- a. `mean(X<=a)`
- b. `pnorm((a - m)/s)`
- c. `pnorm((a - m)/s, m, s)`
- d. `pnorm(a)`

7. Imagine the distribution of male adults is approximately normal with an expected value of 69 and a standard deviation of 3. How tall is the male in the 99th percentile? Hint: use `qnorm`.
8. The distribution of IQ scores is approximately normally distributed. The average is 100 and the standard deviation is 15. Suppose you want to know the distribution of the highest IQ across all graduating classes if 10,000 people are born each in your school district. Run a Monte Carlo simulation with `B=1000` generating 10,000 IQ scores and keeping the highest. Make a histogram.

# 14

## *Random variables*

In data science, we often deal with data that is affected by chance in some way: the data comes from a random sample, the data is affected by measurement error, or the data measures some outcome that is random in nature. Being able to quantify the uncertainty introduced by randomness is one of the most important jobs of a data analyst. Statistical inference offers a framework, as well as several practical tools, for doing this. The first step is to learn how to mathematically describe random variables.

In this chapter, we introduce random variables and their properties starting with their application to games of chance. We then describe some of the events surrounding the financial crisis of 2007-2008<sup>1</sup> using probability theory. This financial crisis was in part caused by underestimating the risk of certain securities<sup>2</sup> sold by financial institutions. Specifically, the risks of mortgage-backed securities (MBS) and collateralized debt obligations (CDO) were grossly underestimated. These assets were sold at prices that assumed most homeowners would make their monthly payments, and the probability of this not occurring was calculated as being low. A combination of factors resulted in many more defaults than were expected, which led to a price crash of these securities. As a consequence, banks lost so much money that they needed government bailouts to avoid closing down completely.

### 14.1 Random variables

Random variables are numeric outcomes resulting from random processes. We can easily generate random variables using some of the simple examples we have shown. For example, define  $X$  to be 1 if a bead is blue and red otherwise:

```
beads <- rep(c("red", "blue"), times = c(2,3))
X <- ifelse(sample(beads, 1) == "blue", 1, 0)
```

Here  $X$  is a random variable: every time we select a new bead the outcome changes randomly. See below:

```
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 1
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 0
ifelse(sample(beads, 1) == "blue", 1, 0)
#> [1] 0
```

<sup>1</sup>[https://en.wikipedia.org/w/index.php?title=Financial\\_crisis\\_of\\_2007%E2%80%932008](https://en.wikipedia.org/w/index.php?title=Financial_crisis_of_2007%E2%80%932008)

<sup>2</sup>[https://en.wikipedia.org/w/index.php?title=Security\\_\(finance\)](https://en.wikipedia.org/w/index.php?title=Security_(finance))

Sometimes it's 1 and sometimes it's 0.

## 14.2 Sampling models

Many data generation procedures, those that produce the data we study, can be modeled quite well as draws from an urn. For instance, we can model the process of polling likely voters as drawing 0s (Republicans) and 1s (Democrats) from an urn containing the 0 and 1 code for all likely voters. In epidemiological studies, we often assume that the subjects in our study are a random sample from the population of interest. The data related to a specific outcome can be modeled as a random sample from an urn containing the outcome for the entire population of interest. Similarly, in experimental research, we often assume that the individual organisms we are studying, for example worms, flies, or mice, are a random sample from a larger population. Randomized experiments can also be modeled by draws from an urn given the way individuals are assigned into groups: when getting assigned, you draw your group at random. Sampling models are therefore ubiquitous in data science. Casino games offer a plethora of examples of real-world situations in which sampling models are used to answer specific questions. We will therefore start with such examples.

Suppose a very small casino hires you to consult on whether they should set up roulette wheels. To keep the example simple, we will assume that 1,000 people will play and that the only game you can play on the roulette wheel is to bet on red or black. The casino wants you to predict how much money they will make or lose. They want a range of values and, in particular, they want to know what's the chance of losing money. If this probability is too high, they will pass on installing roulette wheels.

We are going to define a random variable  $S$  that will represent the casino's total winnings. Let's start by constructing the urn. A roulette wheel has 18 red pockets, 18 black pockets and 2 green ones. So playing a color in one game of roulette is equivalent to drawing from this urn:

```
color <- rep(c("Black", "Red", "Green"), c(18, 18, 2))
```

The 1,000 outcomes from 1,000 people playing are independent draws from this urn. If red comes up, the gambler wins and the casino loses a dollar, so we draw a -\$1. Otherwise, the casino wins a dollar and we draw a \$1. To construct our random variable  $S$ , we can use this code:

```
n <- 1000
X <- sample(ifelse(color == "Red", -1, 1), n, replace = TRUE)
X[1:10]
#> [1] -1 1 1 1 -1 -1 -1 1 1 1
```

Because we know the proportions of 1s and -1s, we can generate the draws with one line of code, without defining `color`:

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
```

We call this a **sampling model** since we are modeling the random behavior of roulette with the sampling of draws from an urn. The total winnings  $S$  is simply the sum of these 1,000 independent draws:

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
S <- sum(X)
S
#> [1] 22
```

---

### 14.3 The probability distribution of a random variable

If you run the code above, you see that  $S$  changes every time. This is, of course, because  $S$  is a **random variable**. The probability distribution of a random variable tells us the probability of the observed value falling at any given interval. So, for example, if we want to know the probability that we lose money, we are asking the probability that  $S$  is in the interval  $S < 0$ .

Note that if we can define a cumulative distribution function  $F(a) = \Pr(S \leq a)$ , then we will be able to answer any question related to the probability of events defined by our random variable  $S$ , including the event  $S < 0$ . We call this  $F$  the random variable's *distribution function*.

We can estimate the distribution function for the random variable  $S$  by using a Monte Carlo simulation to generate many realizations of the random variable. With this code, we run the experiment of having 1,000 people play roulette, over and over, specifically  $B = 10,000$  times:

```
n <- 1000
B <- 10000
roulette_winnings <- function(n){
 X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
 sum(X)
}
S <- replicate(B, roulette_winnings(n))
```

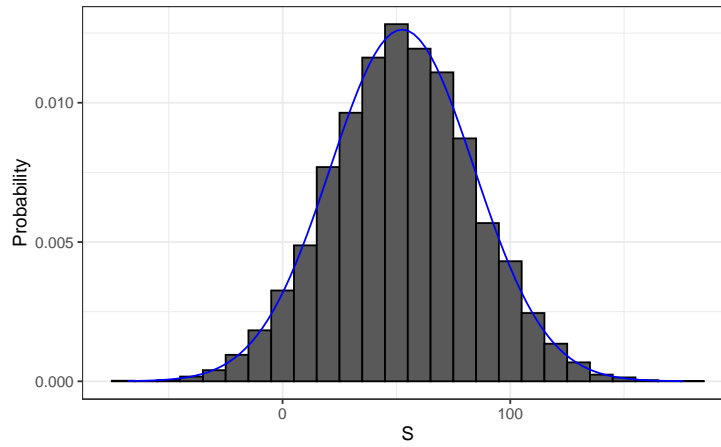
Now we can ask the following: in our simulations, how often did we get sums less than or equal to  $a$ ?

```
mean(S <= a)
```

This will be a very good approximation of  $F(a)$  and we can easily answer the casino's question: how likely is it that we will lose money? We can see it is quite low:

```
mean(S<0)
#> [1] 0.0456
```

We can visualize the distribution of  $S$  by creating a histogram showing the probability  $F(b) - F(a)$  for several intervals  $(a, b]$ :



We see that the distribution appears to be approximately normal. A qq-plot will confirm that the normal approximation is close to a perfect approximation for this distribution. If, in fact, the distribution is normal, then all we need to define the distribution is the average and the standard deviation. Because we have the original values from which the distribution is created, we can easily compute these with `mean(S)` and `sd(S)`. The blue curve you see added to the histogram above is a normal density with this average and standard deviation.

This average and this standard deviation have special names. They are referred to as the *expected value* and *standard error* of the random variable  $S$ . We will say more about these in the next section.

Statistical theory provides a way to derive the distribution of random variables defined as independent random draws from an urn. Specifically, in our example above, we can show that  $(S + n)/2$  follows a binomial distribution. We therefore do not need to run for Monte Carlo simulations to know the probability distribution of  $S$ . We did this for illustrative purposes.

We can use the function `dbinom` and `pbinom` to compute the probabilities exactly. For example, to compute  $\Pr(S < 0)$  we note that:

$$\Pr(S < 0) = \Pr((S + n)/2 < (0 + n)/2)$$

and we can use the `pbinom` to compute

$$\Pr(S \leq 0)$$

```
n <- 1000
pbinom(n/2, size = n, prob = 10/19)
#> [1] 0.0511
```

Because this is a discrete probability function, to get  $\Pr(S < 0)$  rather than  $\Pr(S \leq 0)$ , we write:

```
pbinom(n/2-1, size = n, prob = 10/19)
#> [1] 0.0448
```

For the details of the binomial distribution, you can consult any basic probability book or even Wikipedia<sup>3</sup>.

Here we do not cover these details. Instead, we will discuss an incredibly useful approximation provided by mathematical theory that applies generally to sums and averages of draws from any urn: the Central Limit Theorem (CLT).

---

## 14.4 Distributions versus probability distributions

Before we continue, let's make an important distinction and connection between the distribution of a list of numbers and a probability distribution. In the visualization chapter, we described how any list of numbers  $x_1, \dots, x_n$  has a distribution. The definition is quite straightforward. We define  $F(a)$  as the function that tells us what proportion of the list is less than or equal to  $a$ . Because they are useful summaries when the distribution is approximately normal, we define the average and standard deviation. These are defined with a straightforward operation of the vector containing the list of numbers  $\mathbf{x}$ :

```
m <- sum(x)/length(x)
s <- sqrt(sum((x - m)^2) / length(x))
```

A random variable  $X$  has a distribution function. To define this, we do not need a list of numbers. It is a theoretical concept. In this case, we define the distribution as the  $F(a)$  that answers the question: what is the probability that  $X$  is less than or equal to  $a$ ? There is no list of numbers.

However, if  $X$  is defined by drawing from an urn with numbers in it, then there is a list: the list of numbers inside the urn. In this case, the distribution of that list is the probability distribution of  $X$  and the average and standard deviation of that list are the expected value and standard error of the random variable.

Another way to think about it that does not involve an urn is to run a Monte Carlo simulation and generate a very large list of outcomes of  $X$ . These outcomes are a list of numbers. The distribution of this list will be a very good approximation of the probability distribution of  $X$ . The longer the list, the better the approximation. The average and standard deviation of this list will approximate the expected value and standard error of the random variable.

---

## 14.5 Notation for random variables

In statistical textbooks, upper case letters are used to denote random variables and we follow this convention here. Lower case letters are used for observed values. You will see some notation that includes both. For example, you will see events defined as  $X \leq x$ . Here  $X$  is a random variable, making it a random event, and  $x$  is an arbitrary value and not

---

<sup>3</sup>[https://en.wikipedia.org/w/index.php?title=Binomial\\_distribution](https://en.wikipedia.org/w/index.php?title=Binomial_distribution)



random. So, for example,  $X$  might represent the number on a die roll and  $x$  will represent an actual value we see 1, 2, 3, 4, 5, or 6. So in this case, the probability of  $X = x$  is  $1/6$  regardless of the observed value  $x$ . This notation is a bit strange because, when we ask questions about probability,  $X$  is not an observed quantity. Instead, it's a random quantity that we will see in the future. We can talk about what we expect it to be, what values are probable, but not what it is. But once we have data, we do see a realization of  $X$ . So data scientists talk of what could have been after we see what actually happened.

---

## 14.6 The expected value and standard error

We have described sampling models for draws. We will now go over the mathematical theory that lets us approximate the probability distributions for the sum of draws. Once we do this, we will be able to help the casino predict how much money they will make. The same approach we use for the sum of draws will be useful for describing the distribution of averages and proportion which we will need to understand how polls work.

The first important concept to learn is the *expected value*. In statistics books, it is common to use letter E like this:

$$E[X]$$

to denote the expected value of the random variable  $X$ .

A random variable will vary around its expected value in a way that if you take the average of many, many draws, the average of the draws will approximate the expected value, getting closer and closer the more draws you take.

Theoretical statistics provides techniques that facilitate the calculation of expected values in different circumstances. For example, a useful formula tells us that the *expected value of a random variable defined by one draw is the average of the numbers in the urn*. In the urn used to model betting on red in roulette, we have 20 one dollars and 18 negative one dollars. The expected value is thus:

$$E[X] = (20 + -18)/38$$

which is about 5 cents. It is a bit counterintuitive to say that  $X$  varies around 0.05, when the only values it takes is 1 and -1. One way to make sense of the expected value in this context is by realizing that if we play the game over and over, the casino wins, on average, 5 cents per game. A Monte Carlo simulation confirms this:

```
B <- 10^6
x <- sample(c(-1,1), B, replace = TRUE, prob=c(9/19, 10/19))
mean(x)
#> [1] 0.0517
```

In general, if the urn has two possible outcomes, say  $a$  and  $b$ , with proportions  $p$  and  $1 - p$  respectively, the average is:

$$E[X] = ap + b(1 - p)$$

To see this, notice that if there are  $n$  beads in the urn, then we have  $np$   $a$ s and  $n(1 - p)$   $b$ s and because the average is the sum,  $n \times a \times p + n \times b \times (1 - p)$ , divided by the total  $n$ , we get that the average is  $ap + b(1 - p)$ .

Now the reason we define the expected value is because this mathematical definition turns out to be useful for approximating the probability distributions of sum, which then is useful for describing the distribution of averages and proportions. The first useful fact is that the *expected value of the sum of the draws* is:

$$\text{number of draws} \times \text{average of the numbers in the urn}$$

So if 1,000 people play roulette, the casino expects to win, on average, about  $1,000 \times \$0.05 = \$50$ . But this is an expected value. How different can one observation be from the expected value? The casino really needs to know this. What is the range of possibilities? If negative numbers are too likely, they will not install roulette wheels. Statistical theory once again answers this question. The *standard error* (SE) gives us an idea of the size of the variation around the expected value. In statistics books, it's common to use:

$$SE[X]$$

to denote the standard error of a random variable.

**If our draws are independent**, then the *standard error of the sum* is given by the equation:

$$\sqrt{\text{number of draws}} \times \text{standard deviation of the numbers in the urn}$$

Using the definition of standard deviation, we can derive, with a bit of math, that if an urn contains two values  $a$  and  $b$  with proportions  $p$  and  $(1 - p)$ , respectively, the standard deviation is:

$$|b - a| \sqrt{p(1 - p)}.$$

So in our roulette example, the standard deviation of the values inside the urn is:  $|1 - (-1)| \sqrt{10/19 \times 9/19}$  or:

```
2 * sqrt(90)/19
#> [1] 0.999
```

The standard error tells us the typical difference between a random variable and its expectation. Since one draw is obviously the sum of just one draw, we can use the formula above to calculate that the random variable defined by one draw has an expected value of 0.05 and a standard error of about 1. This makes sense since we either get 1 or -1, with 1 slightly favored over -1.

Using the formula above, the sum of 1,000 people playing has standard error of about \$32:

```
n <- 1000
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

As a result, when 1,000 people bet on red, the casino is expected to win \$50 with a standard error of \$32. It therefore seems like a safe bet. But we still haven't answered the question: how likely is it to lose money? Here the CLT will help.

**Advanced note:** Before continuing we should point out that exact probability calculations for the casino winnings can be performed with the binomial distribution. However, here we focus on the CLT, which can be generally applied to sums of random variables in a way that the binomial distribution can't.

### 14.6.1 Population SD versus the sample SD

The standard deviation of a list `x` (below we use heights as an example) is defined as the square root of the average of the squared differences:

```
library(dslabs)
x <- heights$height
m <- mean(x)
s <- sqrt(mean((x-m)^2))
```

Using mathematical notation we write:

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

However, be aware that the `sd` function returns a slightly different result:

```
identical(s, sd(x))
#> [1] FALSE
s-sd(x)
#> [1] -0.00194
```

This is because the `sd` function R does not return the `sd` of the list, but rather uses a formula that estimates standard deviations of a population from a random sample  $X_1, \dots, X_N$  which, for reasons not discussed here, divide the sum of squares by the  $N - 1$ .

$$\bar{X} = \frac{1}{N} \sum_{i=1}^N X_i, \quad s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_i - \bar{X})^2}$$

You can see that this is the case by typing:

```
n <- length(x)
s-sd(x)*sqrt((n-1) / n)
#> [1] 0
```

For all the theory discussed here, you need to compute the actual standard deviation as defined:

```
sqrt(mean((x-m)^2))
```

So be careful when using the `sd` function in R. However, keep in mind that throughout the book we sometimes use the `sd` function when we really want the actual SD. This is because when the list size is big, these two are practically equivalent since  $\sqrt{(N-1)/N} \approx 1$ .

---

## 14.7 Central Limit Theorem

The Central Limit Theorem (CLT) tells us that when the number of draws, also called the *sample size*, is large, the probability distribution of the sum of the independent draws is approximately normal. Because sampling models are used for so many data generation processes, the CLT is considered one of the most important mathematical insights in history.

Previously, we discussed that if we know that the distribution of a list of numbers is approximated by the normal distribution, all we need to describe the list are the average and standard deviation. We also know that the same applies to probability distributions. If a random variable has a probability distribution that is approximated with the normal distribution, then all we need to describe the probability distribution are the average and standard deviation, referred to as the expected value and standard error.

We previously ran this Monte Carlo simulation:

```
n <- 1000
B <- 10000
roulette_winnings <- function(n){
 X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
 sum(X)
}
S <- replicate(B, roulette_winnings(n))
```

The Central Limit Theorem (CLT) tells us that the sum  $S$  is approximated by a normal distribution. Using the formulas above, we know that the expected value and standard error are:

```
n * (20-18)/38
#> [1] 52.6
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

The theoretical values above match those obtained with the Monte Carlo simulation:

```
mean(S)
#> [1] 52.2
sd(S)
#> [1] 31.7
```

Using the CLT, we can skip the Monte Carlo simulation and instead compute the probability of the casino losing money using this approximation:

```
mu <- n * (20-18)/38
se <- sqrt(n) * 2 * sqrt(90)/19
pnorm(0, mu, se)
#> [1] 0.0478
```

which is also in very good agreement with our Monte Carlo result:

```
mean(S < 0)
#> [1] 0.0458
```

### 14.7.1 How large is large in the Central Limit Theorem?

The CLT works when the number of draws is large. But large is a relative term. In many circumstances as few as 30 draws is enough to make the CLT useful. In some specific instances, as few as 10 is enough. However, these should not be considered general rules. Note, for example, that when the probability of success is very small, we need much larger sample sizes.

By way of illustration, let's consider the lottery. In the lottery, the chances of winning are less than 1 in a million. Thousands of people play so the number of draws is very large. Yet the number of winners, the sum of the draws, range between 0 and 4. This sum is certainly not well approximated by a normal distribution, so the CLT does not apply, even with the very large sample size. This is generally true when the probability of a success is very low. In these cases, the Poisson distribution is more appropriate.

You can examine the properties of the Poisson distribution using `dpois` and `ppois`. You can generate random variables following this distribution with `rpois`. However, we do not cover the theory here. You can learn about the Poisson distribution in any probability textbook and even Wikipedia<sup>4</sup>

---

## 14.8 Statistical properties of averages

There are several useful mathematical results that we used above and often employ when working with data. We list them below.

1. The expected value of the sum of random variables is the sum of each random variable's expected value. We can write it like this:

$$E[X_1 + X_2 + \cdots + X_n] = E[X_1] + E[X_2] + \cdots + E[X_n]$$

If the  $X$  are independent draws from the urn, then they all have the same expected value. Let's call it  $\mu$  and thus:

---

<sup>4</sup>[https://en.wikipedia.org/w/index.php?title=Poisson\\_distribution](https://en.wikipedia.org/w/index.php?title=Poisson_distribution)

$$E[X_1 + X_2 + \cdots + X_n] = n\mu$$

which is another way of writing the result we show above for the sum of draws.

2. The expected value of a non-random constant times a random variable is the non-random constant times the expected value of a random variable. This is easier to explain with symbols:

$$E[aX] = a \times E[X]$$

To see why this is intuitive, consider change of units. If we change the units of a random variable, say from dollars to cents, the expectation should change in the same way. A consequence of the above two facts is that the expected value of the average of independent draws from the same urn is the expected value of the urn, call it  $\mu$  again:

$$E[(X_1 + X_2 + \cdots + X_n)/n] = E[X_1 + X_2 + \cdots + X_n]/n = n\mu/n = \mu$$

3. The square of the standard error of the sum of **independent** random variables is the sum of the square of the standard error of each random variable. This one is easier to understand in math form:

$$SE[X_1 + X_2 + \cdots + X_n] = \sqrt{SE[X_1]^2 + SE[X_2]^2 + \cdots + SE[X_n]^2}$$

The square of the standard error is referred to as the *variance* in statistical textbooks. Note that this particular property is not as intuitive as the previous three and more in depth explanations can be found in statistics textbooks.

4. The standard error of a non-random constant times a random variable is the non-random constant times the random variable's standard error. As with the expectation:

$$SE[aX] = a \times SE[X]$$

To see why this is intuitive, again think of units.

A consequence of 3 and 4 is that the standard error of the average of independent draws from the same urn is the standard deviation of the urn divided by the square root of  $n$  (the number of draws), call it  $\sigma$ :

$$\begin{aligned} SE[(X_1 + X_2 + \cdots + X_n)/n] &= SE[X_1 + X_2 + \cdots + X_n]/n \\ &= \sqrt{SE[X_1]^2 + SE[X_2]^2 + \cdots + SE[X_n]^2}/n \\ &= \sqrt{\sigma^2 + \sigma^2 + \cdots + \sigma^2}/n \\ &= \sqrt{n\sigma^2}/n \\ &= \sigma/\sqrt{n} \end{aligned}$$

5. If  $X$  is a normally distributed random variable, then if  $a$  and  $b$  are non-random constants,  $aX + b$  is also a normally distributed random variable. All we are doing is changing the units of the random variable by multiplying by  $a$ , then shifting the center by  $b$ .

Note that statistical textbooks use the Greek letters  $\mu$  and  $\sigma$  to denote the expected value

and standard error, respectively. This is because  $\mu$  is the Greek letter for  $m$ , the first letter of *mean*, which is another term used for expected value. Similarly,  $\sigma$  is the Greek letter for  $s$ , the first letter of standard error.

---

## 14.9 Law of large numbers

An important implication of the final result is that the standard error of the average becomes smaller and smaller as  $n$  grows larger. When  $n$  is very large, then the standard error is practically 0 and the average of the draws converges to the average of the urn. This is known in statistical textbooks as the law of large numbers or the law of averages.

### 14.9.1 Misinterpreting law of averages

The law of averages is sometimes misinterpreted. For example, if you toss a coin 5 times and see a head each time, you might hear someone argue that the next toss is probably a tail because of the law of averages: on average we should see 50% heads and 50% tails. A similar argument would be to say that red “is due” on the roulette wheel after seeing black come up five times in a row. These events are independent so the chance of a coin landing heads is 50% regardless of the previous 5. This is also the case for the roulette outcome. The law of averages applies only when the number of draws is very large and not in small samples. After a million tosses, you will definitely see about 50% heads regardless of the outcome of the first five tosses.

Another funny misuse of the law of averages is in sports when TV sportscasters predict a player is about to succeed because they have failed a few times in a row.

---

## 14.10 Exercises

1. In American Roulette you can also bet on green. There are 18 reds, 18 blacks and 2 greens (0 and 00). What are the chances the green comes out?
2. The payout for winning on green is \$17 dollars. This means that if you bet a dollar and it lands on green, you get \$17. Create a sampling model using sample to simulate the random variable  $X$  for your winnings. Hint: see the example below for how it should look like when betting on red.

```
x <- sample(c(1,-1), 1, prob = c(9/19, 10/19))
```

3. Compute the expected value of  $X$ .
4. Compute the standard error of  $X$ .
5. Now create a random variable  $S$  that is the sum of your winnings after betting on green

1000 times. Hint: change the argument `size` and `replace` in your answer to question 2. Start your code by setting the seed to 1 with `set.seed(1)`.

6. What is the expected value of  $S$ ?

7. What is the standard error of  $S$ ?

8. What is the probability that you end up winning money? Hint: use the CLT.

9. Create a Monte Carlo simulation that generates 1,000 outcomes of  $S$ . Compute the average and standard deviation of the resulting list to confirm the results of 6 and 7. Start your code by setting the seed to 1 with `set.seed(1)`.

10. Now check your answer to 8 using the Monte Carlo result.

11. The Monte Carlo result and the CLT approximation are close, but not that close. What could account for this?

- a. 1,000 simulations is not enough. If we do more, they match.
- b. The CLT does not work as well when the probability of success is small. In this case, it was  $1/19$ . If we make the number of roulette plays bigger, they will match better.
- c. The difference is within rounding error.
- d. The CLT only works for averages.

12. Now create a random variable  $Y$  that is your average winnings per bet after playing off your winnings after betting on green 1,000 times.

13. What is the expected value of  $Y$ ?

14. What is the standard error of  $Y$ ?

15. What is the probability that you end up with winnings per game that are positive? Hint: use the CLT.

16. Create a Monte Carlo simulation that generates 2,500 outcomes of  $Y$ . Compute the average and standard deviation of the resulting list to confirm the results of 6 and 7. Start your code by setting the seed to 1 with `set.seed(1)`.

17. Now check your answer to 8 using the Monte Carlo result.

18. The Monte Carlo result and the CLT approximation are now much closer. What could account for this?

- a. We are now computing averages instead of sums.
- b. 2,500 Monte Carlo simulations is not better than 1,000.
- c. The CLT works better when the sample size is larger. We increased from 1,000 to 2,500.
- d. It is not closer. The difference is within rounding error.



## 14.11 Case study: The Big Short

### 14.11.1 Interest rates explained with chance model

More complex versions of the sampling models we have discussed are also used by banks to decide interest rates. Suppose you run a small bank that has a history of identifying potential homeowners that can be trusted to make payments. In fact, historically, in a given year, only 2% of your customers default, meaning that they don't pay back the money that you lent them. However, you are aware that if you simply loan money to everybody without interest, you will end up losing money due to this 2%. Although you know 2% of your clients will probably default, you don't know which ones. Yet by charging everybody just a bit extra in interest, you can make up the losses incurred due to that 2% and also cover your operating costs. You can also make a profit, but if you set the interest rates too high, your clients will go to another bank. We use all these facts and some probability theory to decide what interest rate you should charge.

Suppose your bank will give out 1,000 loans for \$180,000 this year. Also, after adding up all costs, suppose your bank loses \$200,000 per foreclosure. For simplicity, we assume this includes all operational costs. A sampling model for this scenario can be coded like this:

```
n <- 1000
loss_per_foreclosure <- -200000
p <- 0.02
defaults <- sample(c(0,1), n, prob=c(1-p, p), replace = TRUE)
sum(defaults * loss_per_foreclosure)
#> [1] -3600000
```

Note that the total loss defined by the final sum is a random variable. Every time you run the above code, you get a different answer. We can easily construct a Monte Carlo simulation to get an idea of the distribution of this random variable.

```
B <- 10000
losses <- replicate(B, {
 defaults <- sample(c(0,1), n, prob=c(1-p, p), replace = TRUE)
 sum(defaults * loss_per_foreclosure)
})
```

We don't really need a Monte Carlo simulation though. Using what we have learned, the CLT tells us that because our losses are a sum of independent draws, its distribution is approximately normal with expected value and standard errors given by:

```
n*(p*loss_per_foreclosure + (1-p)*0)
#> [1] -4e+06
sqrt(n)*abs(loss_per_foreclosure)*sqrt(p*(1-p))
#> [1] 885438
```

We can now set an interest rate to guarantee that, on average, we break even. Basically, we

need to add a quantity  $x$  to each loan, which in this case are represented by draws, so that the expected value is 0. If we define  $l$  to be the loss per foreclosure, we need:

$$lp + x(1 - p) = 0$$

which implies  $x$  is

```
- loss_per_foreclosure*p/(1-p)
#> [1] 4082
```

or an interest rate of 0.023.

However, we still have a problem. Although this interest rate guarantees that on average we break even, there is a 50% chance that we lose money. If our bank loses money, we have to close it down. We therefore need to pick an interest rate that makes it unlikely for this to happen. At the same time, if the interest rate is too high, our clients will go to another bank so we must be willing to take some risks. So let's say that we want our chances of losing money to be 1 in 100, what does the  $x$  quantity need to be now? This one is a bit harder. We want the sum  $S$  to have:

$$\Pr(S < 0) = 0.01$$

We know that  $S$  is approximately normal. The expected value of  $S$  is

$$E[S] = \{lp + x(1 - p)\}n$$

with  $n$  the number of draws, which in this case represents loans. The standard error is

$$SD[S] = |x - l|\sqrt{np(1 - p)}.$$

Because  $x$  is positive and  $l$  negative  $|x - l| = x - l$ . Note that these are just an application of the formulas shown earlier, but using more compact symbols.

Now we are going to use a mathematical “trick” that is very common in statistics. We add and subtract the same quantities to both sides of the event  $S < 0$  so that the probability does not change and we end up with a standard normal random variable on the left, which will then permit us to write down an equation with only  $x$  as an unknown. This “trick” is as follows:

If  $\Pr(S < 0) = 0.01$  then

$$\Pr\left(\frac{S - E[S]}{SE[S]} < \frac{-E[S]}{SE[S]}\right)$$

And remember  $E[S]$  and  $SE[S]$  are the expected value and standard error of  $S$ , respectively. All we did above was add and divide by the same quantity on both sides. We did this because now the term on the left is a standard normal random variable, which we will rename  $Z$ . Now we fill in the blanks with the actual formula for expected value and standard error:

$$\Pr\left(Z < \frac{-\{lp + x(1 - p)\}n}{(x - l)\sqrt{np(1 - p)}}\right) = 0.01$$

It may look complicated, but remember that  $l$ ,  $p$  and  $n$  are all known amounts, so eventually we will replace them with numbers.

Now because the  $Z$  is a normal random with expected value 0 and standard error 1, it means that the quantity on the right side of the  $<$  sign must be equal to:

```
qnorm(0.01)
#> [1] -2.33
```

for the equation to hold true. Remember that  $z = \text{qnorm}(0.01)$  gives us the value of  $z$  for which:

$$\Pr(Z \leq z) = 0.01$$

So this means that the right side of the complicated equation must be  $z = \text{qnorm}(0.01)$ .

$$\frac{-\{lp + x(1-p)\}n}{(x-l)\sqrt{np(1-p)}} = z$$

The trick works because we end up with an expression containing  $x$  that we know has to be equal to a known quantity  $z$ . Solving for  $x$  is now simply algebra:

$$x = -l \frac{np - z\sqrt{np(1-p)}}{n(1-p) + z\sqrt{np(1-p)}}$$

which is:

```
l <- loss_per_foreclosure
z <- qnorm(0.01)
x <- -l*(np - z*sqrt(np*(1-p)))/ (n*(1-p) + z*sqrt(np*(1-p)))
x
#> [1] 6249
```

Our interest rate now goes up to 0.035. This is still a very competitive interest rate. By choosing this interest rate, we now have an expected profit per loan of:

```
loss_per_foreclosure*p + x*(1-p)
#> [1] 2124
```

which is a total expected profit of about:

```
n*(loss_per_foreclosure*p + x*(1-p))
#> [1] 2124198
```

dollars!

We can run a Monte Carlo simulation to double check our theoretical approximations:

```

B <- 100000
profit <- replicate(B, {
 draws <- sample(c(x, loss_per_foreclosure), n,
 prob=c(1-p, p), replace = TRUE)
 sum(draws)
})
mean(profit)
#> [1] 2118604
mean(profit<0)
#> [1] 0.013

```

### 14.11.2 The Big Short

One of your employees points out that since the bank is making 2,124 dollars per loan, the bank should give out more loans! Why just  $n$ ? You explain that finding those  $n$  clients was hard. You need a group that is predictable and that keeps the chances of defaults low. He then points out that even if the probability of default is higher, as long as our expected value is positive, you can minimize your chances of losses by increasing  $n$  and relying on the law of large numbers.

He claims that even if the default rate is twice as high, say 4%, if we set the rate just a bit higher than this value:

```

p <- 0.04
r <- (- loss_per_foreclosure*p/(1-p)) / 180000
r
#> [1] 0.0463

```

we will profit. At 5%, we are guaranteed a positive expected value of:

```

r <- 0.05
x <- r*180000
loss_per_foreclosure*p + x * (1-p)
#> [1] 640

```

and can minimize our chances of losing money by simply increasing  $n$  since:

$$\Pr(S < 0) = \Pr\left(Z < -\frac{E[S]}{SE[S]}\right)$$

with  $Z$  a standard normal random variable as shown earlier. If we define  $\mu$  and  $\sigma$  to be the expected value and standard deviation of the urn, respectively (that is of a single loan), using the formulas above we have:  $E[S] = n\mu$  and  $SE[S] = \sqrt{n}\sigma$ . So if we define  $z = \text{qnorm}(0.01)$ , we have:

$$-\frac{n\mu}{\sqrt{n}\sigma} = -\frac{\sqrt{n}\mu}{\sigma} = z$$

which implies that if we let:

$$n \geq z^2 \sigma^2 / \mu^2$$

we are guaranteed to have a probability of less than 0.01. The implication is that, as long as  $\mu$  is positive, we can find an  $n$  that minimizes the probability of a loss. This is a form of the law of large numbers: when  $n$  is large, our average earnings per loan converges to the expected earning  $\mu$ .

With  $x$  fixed, now we can ask what  $n$  do we need for the probability to be 0.01? In our example, if we give out:

```
z <- qnorm(0.01)
n <- ceiling((z^2*(x-1)^2*p*(1-p))/(1*p + x*(1-p))^2)
n
#> [1] 22163
```

loans, the probability of losing is about 0.01 and we are expected to earn a total of

```
n*(loss_per_foreclosure*p + x * (1-p))
#> [1] 14184320
```

dollars! We can confirm this with a Monte Carlo simulation:

```
p <- 0.04
x <- 0.05*180000
profit <- replicate(B, {
 draws <- sample(c(x, loss_per_foreclosure), n,
 prob=c(1-p, p), replace = TRUE)
 sum(draws)
})
mean(profit)
#> [1] 14185724
```

This seems like a no brainer. As a result, your colleague decides to leave your bank and start his own high-risk mortgage company. A few months later, your colleague's bank has gone bankrupt. A book is written and eventually a movie is made relating the mistake your friend, and many others, made. What happened?

Your colleague's scheme was mainly based on this mathematical formula:

$$\text{SE}[(X_1 + X_2 + \cdots + X_n)/n] = \sigma/\sqrt{n}$$

By making  $n$  large, we minimize the standard error of our per-loan profit. However, for this rule to hold, the  $X$ s must be independent draws: one person defaulting must be independent of others defaulting. Note that in the case of averaging the **same** event over and over, an extreme example of events that are not independent, we get a standard error that is  $\sqrt{n}$  times bigger:

$$\text{SE}[(X_1 + X_1 + \cdots + X_1)/n] = \text{SE}[nX_1/n] = \sigma > \sigma/\sqrt{n}$$

To construct a more realistic simulation than the original one your colleague ran, let's assume there is a global event that affects everybody with high-risk mortgages and changes their probability. We will assume that with 50-50 chance, all the probabilities go up or down slightly to somewhere between 0.03 and 0.05. But it happens to everybody at once, not just one person. These draws are no longer independent.

```
p <- 0.04
x <- 0.05*180000
profit <- replicate(B, {
 new_p <- 0.04 + sample(seq(-0.01, 0.01, length = 100), 1)
 draws <- sample(c(x, loss_per_foreclosure), n,
 prob=c(1-new_p, new_p), replace = TRUE)
 sum(draws)
})
```

Note that our expected profit is still large:

```
mean(profit)
#> [1] 14082671
```

However, the probability of the bank having negative earnings shoots up to:

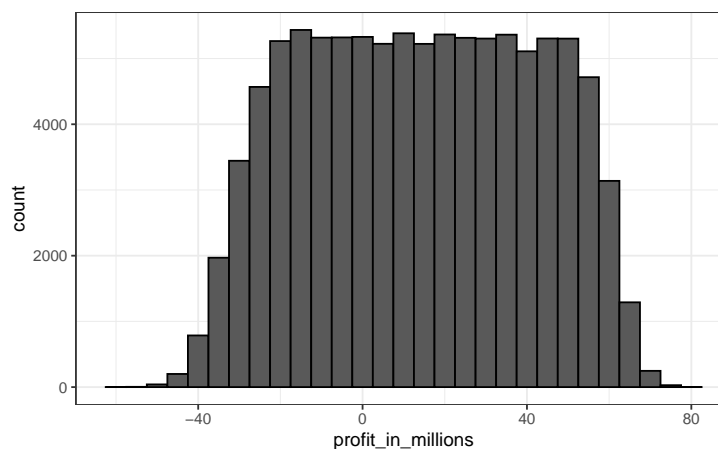
```
mean(profit<0)
#> [1] 0.35
```

Even scarier is that the probability of losing more than 10 million dollars is:

```
mean(profit < -10000000)
#> [1] 0.244
```

To understand how this happens look at the distribution:

```
data.frame(profit_in_millions=profit/10^6) %>%
 ggplot(aes(profit_in_millions)) +
 geom_histogram(color="black", binwidth = 5)
```



The theory completely breaks down and the random variable has much more variability than expected. The financial meltdown of 2007 was due, among other things, to financial “experts” assuming independence when there was none.

---

### 14.12 Exercises

1. Create a random variable  $S$  with the earnings of your bank if you give out 10,000 loans, the default rate is 0.3, and you lose \$200,000 in each foreclosure. Hint: use the code we showed in the previous section, but change the parameters.
2. Run a Monte Carlo simulation with 10,000 outcomes for  $S$ . Make a histogram of the results.
3. What is the expected value of  $S$ ?
4. What is the standard error of  $S$ ?
5. Suppose we give out loans for \$180,000. What should the interest rate be so that our expected value is 0?
6. (Harder) What should the interest rate be so that the chance of losing money is 1 in 20? In math notation, what should the interest rate be so that  $\Pr(S < 0) = 0.05$  ?
7. If the bank wants to minimize the probabilities of losing money, which of the following does **not** make interest rates go up?
  - a. A smaller pool of loans.
  - b. A larger probability of default.
  - c. A smaller required probability of losing money.
  - d. The number of Monte Carlo simulations.

# 15

---

## *Statistical inference*

---

In Chapter 16 we will describe, in some detail, how poll aggregators such as FiveThirtyEight use data to predict election outcomes. To understand how they do this, we first need to learn the basics of *Statistical Inference*, the part of statistics that helps distinguish patterns arising from signal from those arising from chance. Statistical inference is a broad topic and here we go over the very basics using polls as a motivating example. To describe the concepts, we complement the mathematical formulas with Monte Carlo simulations and R code.

---

### 15.1 Polls

Opinion polling has been conducted since the 19th century. The general goal is to describe the opinions held by a specific population on a given set of topics. In recent times, these polls have been pervasive during presidential elections. Polls are useful when interviewing every member of a particular population is logistically impossible. The general strategy is to interview a smaller group, chosen at random, and then infer the opinions of the entire population from the opinions of the smaller group. Statistical theory is used to justify the process. This theory is referred to as *inference* and it is the main topic of this chapter.

Perhaps the best known opinion polls are those conducted to determine which candidate is preferred by voters in a given election. Political strategists make extensive use of polls to decide, among other things, how to invest resources. For example, they may want to know in which geographical locations to focus their “get out the vote” efforts.

Elections are a particularly interesting case of opinion polls because the actual opinion of the entire population is revealed on election day. Of course, it costs millions of dollars to run an actual election which makes polling a cost effective strategy for those that want to forecast the results.

Although typically the results of these polls are kept private, similar polls are conducted by news organizations because results tend to be of interest to the general public and made public. We will eventually be looking at such data.

Real Clear Politics<sup>1</sup> is an example of a news aggregator that organizes and publishes poll results. For example, they present the following poll results reporting estimates of the popular vote for the 2016 presidential election<sup>2</sup>:

---

<sup>1</sup><http://www.realclearpolitics.com>

<sup>2</sup>[http://www.realclearpolitics.com/epolls/2016/president/us/general\\_election\\_trump\\_vs\\_clinton-5491.html](http://www.realclearpolitics.com/epolls/2016/president/us/general_election_trump_vs_clinton-5491.html)



| Poll          | Date        | Sample  | MoE | Clinton | Trump | Spread       |
|---------------|-------------|---------|-----|---------|-------|--------------|
| Final Results | —           | —       | —   | 48.2    | 46.1  | Clinton +2.1 |
| RCP Average   | 11/1 - 11/7 | —       | —   | 46.8    | 43.6  | Clinton +3.2 |
| Bloomberg     | 11/4 - 11/6 | 799 LV  | 3.5 | 46.0    | 43.0  | Clinton +3   |
| IBD           | 11/4 - 11/7 | 1107 LV | 3.1 | 43.0    | 42.0  | Clinton +1   |
| Economist     | 11/4 - 11/7 | 3669 LV | —   | 49.0    | 45.0  | Clinton +4   |
| LA Times      | 11/1 - 11/7 | 2935 LV | 4.5 | 44.0    | 47.0  | Trump +3     |
| ABC           | 11/3 - 11/6 | 2220 LV | 2.5 | 49.0    | 46.0  | Clinton +3   |
| FOX News      | 11/3 - 11/6 | 1295 LV | 2.5 | 48.0    | 44.0  | Clinton +4   |
| Monmouth      | 11/3 - 11/6 | 748 LV  | 3.6 | 50.0    | 44.0  | Clinton +6   |
| NBC News      | 11/3 - 11/5 | 1282 LV | 2.7 | 48.0    | 43.0  | Clinton +5   |
| CBS News      | 11/2 - 11/6 | 1426 LV | 3.0 | 47.0    | 43.0  | Clinton +4   |
| Reuters       | 11/2 - 11/6 | 2196 LV | 2.3 | 44.0    | 39.0  | Clinton +5   |

Although in the United States the popular vote does not determine the result of the presidential election, we will use it as an illustrative and simple example of how well polls work. Forecasting the election is a more complex process since it involves combining results from 50 states and DC and we describe it in Section 16.8.

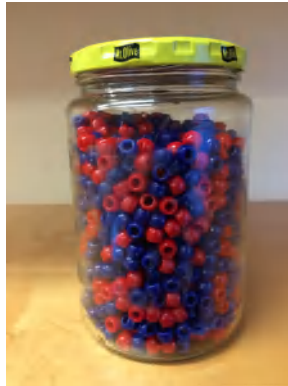
Let's make some observations about the table above. First, note that different polls, all taken days before the election, report a different *spread*: the estimated difference between support for the two candidates. Notice also that the reported spreads hover around what ended up being the actual result: Clinton won the popular vote by 2.1%. We also see a column titled **MoE** which stands for *margin of error*.

In this section, we will show how the probability concepts we learned in the previous chapter can be applied to develop the statistical approaches that make polls an effective tool. We will learn the statistical concepts necessary to define *estimates* and *margins of errors*, and show how we can use these to forecast final results relatively well and also provide an estimate of the precision of our forecast. Once we learn this, we will be able to understand two concepts that are ubiquitous in data science: *confidence intervals* and *p-values*. Finally, to understand probabilistic statements about the probability of a candidate winning, we will have to learn about Bayesian modeling. In the final sections, we put it all together to recreate the simplified version of the FiveThirtyEight model and apply it to the 2016 election.

We start by connecting probability theory to the task of using polls to learn about a population.

### 15.1.1 The sampling model for polls

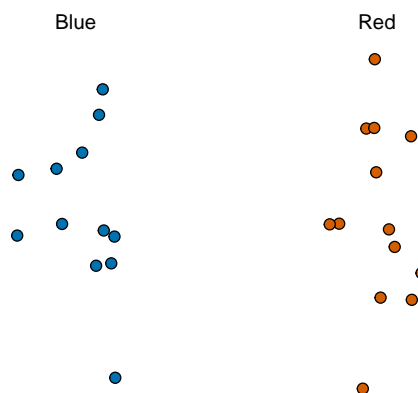
To help us understand the connection between polls and what we have learned, let's construct a similar situation to the one pollsters face. To mimic the challenge real pollsters face in terms of competing with other pollsters for media attention, we will use an urn full of beads to represent voters and pretend we are competing for a \$25 dollar prize. The challenge is to guess the spread between the proportion of blue and red beads in this urn (in this case, a pickle jar):



Before making a prediction, you can take a sample (with replacement) from the urn. To mimic the fact that running polls is expensive, it costs you \$0.10 per each bead you sample. Therefore, if your sample size is 250, and you win, you will break even since you will pay \$25 to collect your \$25 prize. Your entry into the competition can be an interval. If the interval you submit contains the true proportion, you get half what you paid and pass to the second phase of the competition. In the second phase, the entry with the smallest interval is selected as the winner.

The **dslabs** package includes a function that shows a random draw from this urn:

```
library(tidyverse)
library(dslabs)
take_poll(25)
```



Think about how you would construct your interval based on the data shown above.

We have just described a simple sampling model for opinion polls. The beads inside the urn represent the individuals that will vote on election day. Those that will vote for the Republican candidate are represented with red beads and the Democrats with the blue beads. For simplicity, assume there are no other colors. That is, that there are just two parties: Republican and Democratic.

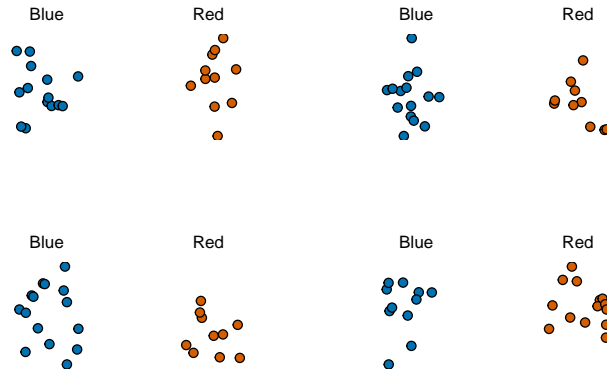
## 15.2 Populations, samples, parameters, and estimates

We want to predict the proportion of blue beads in the urn. Let's call this quantity  $p$ , which then tells us the proportion of red beads  $1 - p$ , and the spread  $p - (1 - p)$ , which simplifies to  $2p - 1$ .

In statistical textbooks, the beads in the urn are called the *population*. The proportion of blue beads in the population  $p$  is called a *parameter*. The 25 beads we see in the previous plot are called a *sample*. The task of statistical inference is to predict the parameter  $p$  using the observed data in the sample.

Can we do this with the 25 observations above? It is certainly informative. For example, given that we see 13 red and 12 blue beads, it is unlikely that  $p > .9$  or  $p < .1$ . But are we ready to predict with certainty that there are more red beads than blue in the jar?

We want to construct an estimate of  $p$  using only the information we observe. An estimate should be thought of as a summary of the observed data that we think is informative about the parameter of interest. It seems intuitive to think that the proportion of blue beads in the sample 0.48 must be at least related to the actual proportion  $p$ . But do we simply predict  $p$  to be 0.48? First, remember that the sample proportion is a random variable. If we run the command `take_poll(25)` four times, we get a different answer each time, since the sample proportion is a random variable.



Note that in the four random samples shown above, the sample proportions range from 0.44 to 0.60. By describing the distribution of this random variable, we will be able to gain insights into how good this estimate is and how we can make it better.

### 15.2.1 The sample average

Conducting an opinion poll is being modeled as taking a random sample from an urn. We are proposing the use of the proportion of blue beads in our sample as an *estimate* of the parameter  $p$ . Once we have this estimate, we can easily report an estimate for the spread  $2p - 1$ , but for simplicity we will illustrate the concepts for estimating  $p$ . We will use our

knowledge of probability to defend our use of the sample proportion and quantify how close we think it is from the population proportion  $p$ .

We start by defining the random variable  $X$  as:  $X = 1$  if we pick a blue bead at random and  $X = 0$  if it is red. This implies that the population is a list of 0s and 1s. If we sample  $N$  beads, then the average of the draws  $X_1, \dots, X_N$  is equivalent to the proportion of blue beads in our sample. This is because adding the  $X$ s is equivalent to counting the blue beads and dividing this count by the total  $N$  is equivalent to computing a proportion. We use the symbol  $\bar{X}$  to represent this average. In general, in statistics textbooks a bar on top of a symbol means the average. The theory we just learned about the sum of draws becomes useful because the average is a sum of draws multiplied by the constant  $1/N$ :

$$\bar{X} = 1/N \times \sum_{i=1}^N X_i$$

For simplicity, let's assume that the draws are independent: after we see each sampled bead, we return it to the urn. In this case, what do we know about the distribution of the sum of draws? First, we know that the expected value of the sum of draws is  $N$  times the average of the values in the urn. We know that the average of the 0s and 1s in the urn must be  $p$ , the proportion of blue beads.

Here we encounter an important difference with what we did in the Probability chapter: we don't know what is in the urn. We know there are blue and red beads, but we don't know how many of each. This is what we want to find out: we are trying to **estimate**  $p$ .

### 15.2.2 Parameters

Just like we use variables to define unknowns in systems of equations, in statistical inference we define *parameters* to define unknown parts of our models. In the urn model which we are using to mimic an opinion poll, we do not know the proportion of blue beads in the urn. We define the parameters  $p$  to represent this quantity.  $p$  is the average of the urn because if we take the average of the 1s (blue) and 0s (red), we get the proportion of blue beads. Since our main goal is figuring out what is  $p$ , we are going to *estimate this parameter*.

The ideas presented here on how we estimate parameters, and provide insights into how good these estimates are, extrapolate to many data science tasks. For example, we may want to determine the difference in health improvement between patients receiving treatment and a control group. We may ask, what are the health effects of smoking on a population? What are the differences in racial groups of fatal shootings by police? What is the rate of change in life expectancy in the US during the last 10 years? All these questions can be framed as a task of estimating a parameter from a sample.

### 15.2.3 Polling versus forecasting

Before we continue, let's make an important clarification related to the practical problem of forecasting the election. If a poll is conducted four months before the election, it is estimating the  $p$  for that moment and not for election day. The  $p$  for election night might be different since people's opinions fluctuate through time. The polls provided the night before the election tend to be the most accurate since opinions don't change that much in a day. However, forecasters try to build tools that model how opinions vary across time and try to

predict the election night results taking into consideration the fact that opinions fluctuate. We will describe some approaches for doing this in a later section.

#### 15.2.4 Properties of our estimate: expected value and standard error

To understand how good our estimate is, we will describe the statistical properties of the random variable defined above: the sample proportion  $\bar{X}$ . Remember that  $\bar{X}$  is the sum of independent draws so the rules we covered in the probability chapter apply.

Using what we have learned, the expected value of the sum  $N\bar{X}$  is  $N \times$  the average of the urn,  $p$ . So dividing by the non-random constant  $N$  gives us that the expected value of the average  $\bar{X}$  is  $p$ . We can write it using our mathematical notation:

$$E(\bar{X}) = p$$

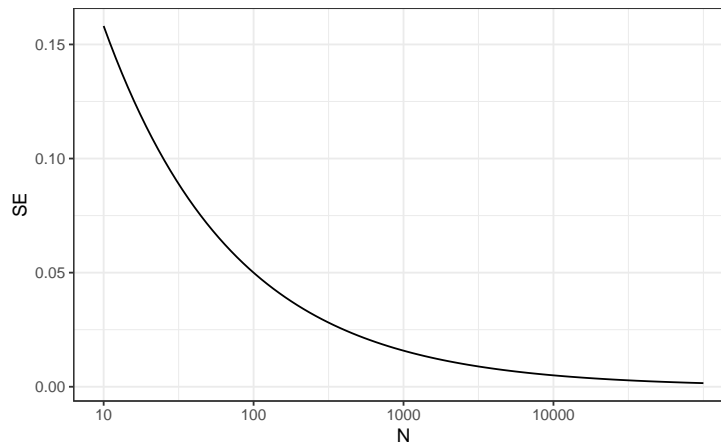
We can also use what we learned to figure out the standard error: the standard error of the sum is  $\sqrt{N} \times$  the standard deviation of the urn. Can we compute the standard error of the urn? We learned a formula that tells us that it is  $(1 - 0)\sqrt{p(1 - p)} = \sqrt{p(1 - p)}$ . Because we are dividing the sum by  $N$ , we arrive at the following formula for the standard error of the average:

$$SE(\bar{X}) = \sqrt{p(1 - p)/N}$$

This result reveals the power of polls. The expected value of the sample proportion  $\bar{X}$  is the parameter of interest  $p$  and we can make the standard error as small as we want by increasing  $N$ . The law of large numbers tells us that with a large enough poll, our estimate converges to  $p$ .

If we take a large enough poll to make our standard error about 1%, we will be quite certain about who will win. But how large does the poll have to be for the standard error to be this small?

One problem is that we do not know  $p$ , so we can't compute the standard error. However, for illustrative purposes, let's assume that  $p = 0.51$  and make a plot of the standard error versus the sample size  $N$ :



From the plot we see that we would need a poll of over 10,000 people to get the standard error that low. We rarely see polls of this size due in part to costs. From the Real Clear Politics table, we learn that the sample sizes in opinion polls range from 500-3,500 people. For a sample size of 1,000 and  $p = 0.51$ , the standard error is:

```
sqrt(p*(1-p))/sqrt(1000)
#> [1] 0.0158
```

or 1.5 percentage points. So even with large polls, for close elections,  $\bar{X}$  can lead us astray if we don't realize it is a random variable. Nonetheless, we can actually say more about how close we get the  $p$  and we do that in Section 15.4.

---

### 15.3 Exercises

1. Suppose you poll a population in which a proportion  $p$  of voters are Democrats and  $1 - p$  are Republicans. Your sample size is  $N = 25$ . Consider the random variable  $S$  which is the **total** number of Democrats in your sample. What is the expected value of this random variable? Hint: it's a function of  $p$ .
2. What is the standard error of  $S$ ? Hint: it's a function of  $p$ .
3. Consider the random variable  $S/N$ . This is equivalent to the sample average, which we have been denoting as  $\bar{X}$ . What is the expected value of the  $\bar{X}$ ? Hint: it's a function of  $p$ .
4. What is the standard error of  $\bar{X}$ ? Hint: it's a function of  $p$ .
5. Write a line of code that gives you the standard error `se` for the problem above for several values of  $p$ , specifically for `p <- seq(0, 1, length = 100)`. Make a plot of `se` versus `p`.
6. Copy the code above and put it inside a for-loop to make the plot for  $N = 25$ ,  $N = 100$ , and  $N = 1000$ .
7. If we are interested in the difference in proportions,  $p - (1 - p)$ , our estimate is  $d = \bar{X} - (1 - \bar{X})$ . Use the rules we learned about sums of random variables and scaled random variables to derive the expected value of  $d$ .
8. What is the standard error of  $d$ ?
9. If the actual  $p = .45$ , it means the Republicans are winning by a relatively large margin since  $d = -.1$ , which is a 10% margin of victory. In this case, what is the standard error of  $2\hat{X} - 1$  if we take a sample of  $N = 25$ ?
10. Given the answer to 9, which of the following best describes your strategy of using a sample size of  $N = 25$ ?
  - a. The expected value of our estimate  $2\bar{X} - 1$  is  $d$ , so our prediction will be right on.
  - b. Our standard error is larger than the difference, so the chances of  $2\bar{X} - 1$  being positive and throwing us off were not that small. We should pick a larger sample size.
  - c. The difference is 10% and the standard error is about 0.2, therefore much smaller than the difference.
  - d. Because we don't know  $p$ , we have no way of knowing that making  $N$  larger would actually improve our standard error.

## 15.4 Central Limit Theorem in practice

The CLT tells us that the distribution function for a sum of draws is approximately normal. We also learned that dividing a normally distributed random variable by a constant is also a normally distributed variable. This implies that the distribution of  $\bar{X}$  is approximately normal.

In summary, we have that  $\bar{X}$  has an approximately normal distribution with expected value  $p$  and standard error  $\sqrt{p(1-p)/N}$ .

Now how does this help us? Suppose we want to know what is the probability that we are within 1% from  $p$ . We are basically asking what is

$$\Pr(|\bar{X} - p| \leq .01)$$

which is the same as:

$$\Pr(\bar{X} \leq p + .01) - \Pr(\bar{X} \leq p - .01)$$

Can we answer this question? We can use the mathematical trick we learned in the previous chapter. Subtract the expected value and divide by the standard error to get a standard normal random variable, call it  $Z$ , on the left. Since  $p$  is the expected value and  $\text{SE}(\bar{X}) = \sqrt{p(1-p)/N}$  is the standard error we get:

$$\Pr\left(Z \leq \frac{.01}{\text{SE}(\bar{X})}\right) - \Pr\left(Z \leq -\frac{.01}{\text{SE}(\bar{X})}\right)$$

One problem we have is that since we don't know  $p$ , we don't know  $\text{SE}(\bar{X})$ . But it turns out that the CLT still works if we estimate the standard error by using  $\bar{X}$  in place of  $p$ . We say that we *plug-in* the estimate. Our estimate of the standard error is therefore:

$$\hat{\text{SE}}(\bar{X}) = \sqrt{\bar{X}(1 - \bar{X})/N}$$

In statistics textbooks, we use a little hat to denote estimates. The estimate can be constructed using the observed data and  $N$ .

Now we continue with our calculation, but dividing by  $\hat{\text{SE}}(\bar{X}) = \sqrt{\bar{X}(1 - \bar{X})/N}$  instead. In our first sample we had 12 blue and 13 red so  $\bar{X} = 0.48$  and our estimate of standard error is:

```
x_hat <- 0.48
se <- sqrt(x_hat*(1-x_hat)/25)
se
#> [1] 0.0999
```

And now we can answer the question of the probability of being close to  $p$ . The answer is:

```
pnorm(0.01/se) - pnorm(-0.01/se)
#> [1] 0.0797
```

Therefore, there is a small chance that we will be close. A poll of only  $N = 25$  people is not really very useful, at least not for a close election.

Earlier we mentioned the *margin of error*. Now we can define it because it is simply two times the standard error, which we can now estimate. In our case it is:

```
1.96*se
#> [1] 0.196
```

Why do we multiply by 1.96? Because if you ask what is the probability that we are within 1.96 standard errors from  $p$ , we get:

$$\Pr(Z \leq 1.96 \text{SE}(\bar{X})/\text{SE}(\bar{X})) - \Pr(Z \leq -1.96 \text{SE}(\bar{X})/\text{SE}(\bar{X}))$$

which is:

$$\Pr(Z \leq 1.96) - \Pr(Z \leq -1.96)$$

which we know is about 95%:

```
pnorm(1.96)-pnorm(-1.96)
#> [1] 0.95
```

Hence, there is a 95% probability that  $\bar{X}$  will be within  $1.96 \times \hat{SE}(\bar{X})$ , in our case within about 0.2, of  $p$ . Note that 95% is somewhat of an arbitrary choice and sometimes other percentages are used, but it is the most commonly used value to define margin of error. We often round 1.96 up to 2 for simplicity of presentation.

In summary, the CLT tells us that our poll based on a sample size of 25 is not very useful. We don't really learn much when the margin of error is this large. All we can really say is that the popular vote will not be won by a large margin. This is why pollsters tend to use larger sample sizes.

From the table above, we see that typical sample sizes range from 700 to 3500. To see how this gives us a much more practical result, notice that if we had obtained a  $\bar{X}=0.48$  with a sample size of 2,000, our standard error  $\hat{SE}(\bar{X})$  would have been 0.011. So our result is an estimate of 48% with a margin of error of 2%. In this case, the result is much more informative and would make us think that there are more red balls than blue. Keep in mind, however, that this is hypothetical. We did not take a poll of 2,000 since we don't want to ruin the competition.

### 15.4.1 A Monte Carlo simulation

Suppose we want to use a Monte Carlo simulation to corroborate the tools we have built using probability theory. To create the simulation, we would write code like this:



```

B <- 10000
N <- 1000
x_hat <- replicate(B, {
 x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
 mean(x)
})

```

The problem is, of course, we don't know  $p$ . We could construct an urn like the one pictured above and run an analog (without a computer) simulation. It would take a long time, but you could take 10,000 samples, count the beads and keep track of the proportions of blue. We can use the function `take_poll(n=1000)` instead of drawing from an actual urn, but it would still take time to count the beads and enter the results.

One thing we therefore do to corroborate theoretical results is to pick one or several values of  $p$  and run the simulations. Let's set  $p=0.45$ . We can then simulate a poll:

```

p <- 0.45
N <- 1000

x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
x_hat <- mean(x)

```

In this particular sample, our estimate is `x_hat`. We can use that code to do a Monte Carlo simulation:

```

B <- 10000
x_hat <- replicate(B, {
 x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
 mean(x)
})

```

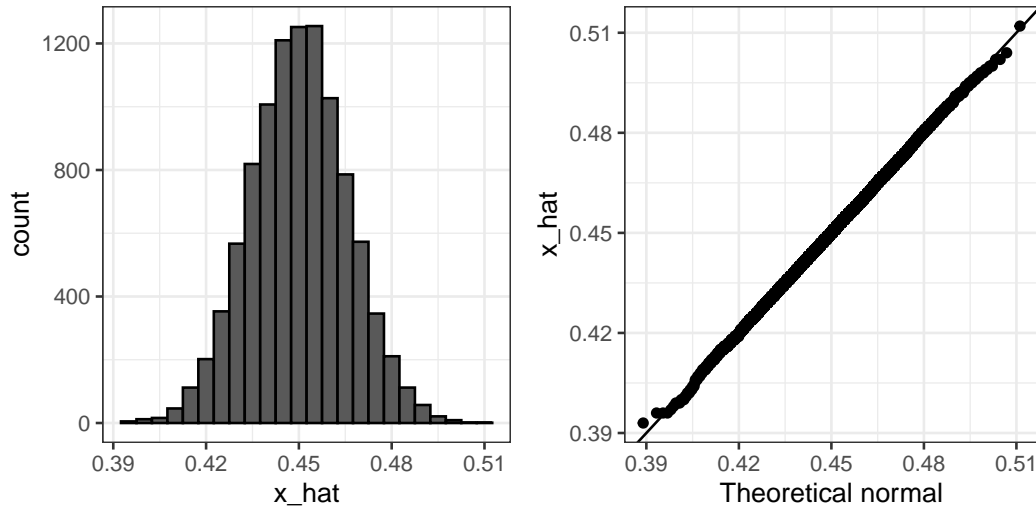
To review, the theory tells us that  $\bar{X}$  is approximately normally distributed, has expected value  $p = 0.45$  and standard error  $\sqrt{p(1-p)/N} = 0.016$ . The simulation confirms this:

```

mean(x_hat)
#> [1] 0.45
sd(x_hat)
#> [1] 0.0157

```

A histogram and qq-plot confirm that the normal approximation is accurate as well:



Of course, in real life we would never be able to run such an experiment because we don't know  $p$ . But we could run it for various values of  $p$  and  $N$  and see that the theory does indeed work well for most values. You can easily do this by re-running the code above after changing  $p$  and  $N$ .

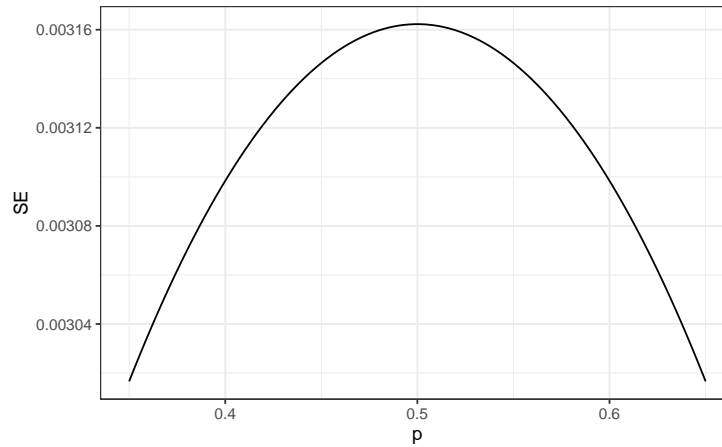
### 15.4.2 The spread

The competition is to predict the spread, not the proportion  $p$ . However, because we are assuming there are only two parties, we know that the spread is  $p - (1 - p) = 2p - 1$ . As a result, everything we have done can easily be adapted to an estimate of  $2p - 1$ . Once we have our estimate  $\bar{X}$  and  $\text{SE}(\bar{X})$ , we estimate the spread with  $2\bar{X} - 1$  and, since we are multiplying by 2, the standard error is  $2\text{SE}(\bar{X})$ . Note that subtracting 1 does not add any variability so it does not affect the standard error.

For our 25 item sample above, our estimate  $p$  is .48 with margin of error .20 and our estimate of the spread is 0.04 with margin of error .40. Again, not a very useful sample size. However, the point is that once we have an estimate and standard error for  $p$ , we have it for the spread  $2p - 1$ .

### 15.4.3 Bias: why not run a very large poll?

For realistic values of  $p$ , say from 0.35 to 0.65, if we run a very large poll with 100,000 people, theory tells us that we would predict the election perfectly since the largest possible margin of error is around 0.3%:



One reason is that running such a poll is very expensive. Another possibly more important reason is that theory has its limitations. Polling is much more complicated than picking beads from an urn. Some people might lie to pollsters and others might not have phones. But perhaps the most important way an actual poll differs from an urn model is that we actually don't know for sure who is in our population and who is not. How do we know who is going to vote? Are we reaching all possible voters? Hence, even if our margin of error is very small, it might not be exactly right that our expected value is  $p$ . We call this bias. Historically, we observe that polls are indeed biased, although not by that much. The typical bias appears to be about 1-2%. This makes election forecasting a bit more interesting and we will talk about how to model this in a later chapter.

## 15.5 Exercises

1. Write an *urn model* function that takes the proportion of Democrats  $p$  and the sample size  $N$  as arguments and returns the sample average if Democrats are 1s and Republicans are 0s. Call the function `take_sample`.
2. Now assume `p <- 0.45` and that your sample size is  $N = 100$ . Take a sample 10,000 times and save the vector of `mean(X) - p` into an object called `errors`. Hint: use the function you wrote for exercise 1 to write this in one line of code.
3. The vector `errors` contains, for each simulated sample, the difference between the actual  $p$  and our estimate  $\bar{X}$ . We refer to this difference as the *error*. Compute the average and make a histogram of the errors generated in the Monte Carlo simulation and select which of the following best describes their distributions:

```
mean(errors)
hist(errors)
```

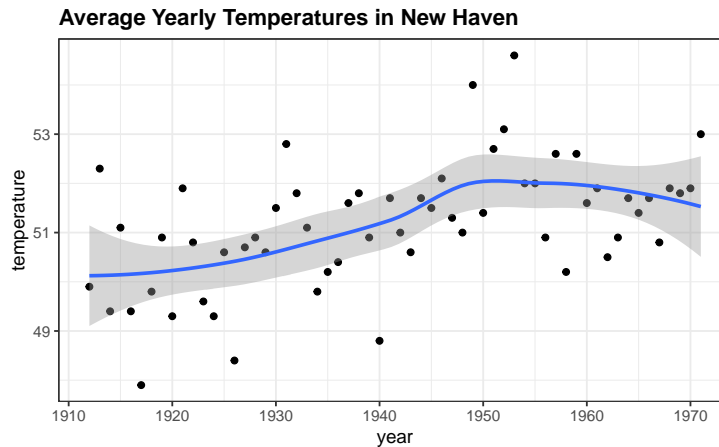
- a. The errors are all about 0.05.
- b. The errors are all about -0.05.
- c. The errors are symmetrically distributed around 0.

- d. The errors range from -1 to 1.
4. The error  $\bar{X} - p$  is a random variable. In practice, the error is not observed because we do not know  $p$ . Here we observe it because we constructed the simulation. What is the average size of the error if we define the size by taking the absolute value  $|\bar{X} - p|$ ?
5. The standard error is related to the typical **size** of the error we make when predicting. We say **size** because we just saw that the errors are centered around 0, so thus the average error value is 0. For mathematical reasons related to the Central Limit Theorem, we actually use the standard deviation of **errors** rather than the average of the absolute values to quantify the typical size. What is this standard deviation of the errors?
6. The theory we just learned tells us what this standard deviation is going to be because it is the standard error of  $\bar{X}$ . What does theory tell us is the standard error of  $\bar{X}$  for a sample size of 100?
7. In practice, we don't know  $p$ , so we construct an estimate of the theoretical prediction based by plugging in  $\bar{X}$  for  $p$ . Compute this estimate. Set the seed at 1 with `set.seed(1)`.
8. Note how close the standard error estimates obtained from the Monte Carlo simulation (exercise 5), the theoretical prediction (exercise 6), and the estimate of the theoretical prediction (exercise 7) are. The theory is working and it gives us a practical approach to knowing the typical error we will make if we predict  $p$  with  $\bar{X}$ . Another advantage that the theoretical result provides is that it gives an idea of how large a sample size is required to obtain the precision we need. Earlier we learned that the largest standard errors occur for  $p = 0.5$ . Create a plot of the largest standard error for  $N$  ranging from 100 to 5,000. Based on this plot, how large does the sample size have to be to have a standard error of about 1%?
- 100
  - 500
  - 2,500
  - 4,000
9. For sample size  $N = 100$ , the central limit theorem tells us that the distribution of  $\bar{X}$  is:
- practically equal to  $p$ .
  - approximately normal with expected value  $p$  and standard error  $\sqrt{p(1-p)/N}$ .
  - approximately normal with expected value  $\bar{X}$  and standard error  $\sqrt{\bar{X}(1-\bar{X})/N}$ .
  - not a random variable.
10. Based on the answer from exercise 8, the error  $\bar{X} - p$  is:
- practically equal to 0.
  - approximately normal with expected value 0 and standard error  $\sqrt{p(1-p)/N}$ .
  - approximately normal with expected value  $p$  and standard error  $\sqrt{p(1-p)/N}$ .
  - not a random variable.
11. To corroborate your answer to exercise 9, make a qq-plot of the **errors** you generated in exercise 2 to see if they follow a normal distribution.
12. If  $p = 0.45$  and  $N = 100$  as in exercise 2, use the CLT to estimate the probability that  $\bar{X} > 0.5$ . You can assume you know  $p = 0.45$  for this calculation.

13. Assume you are in a practical situation and you don't know  $p$ . Take a sample of size  $N = 100$  and obtain a sample average of  $\bar{X} = 0.51$ . What is the CLT approximation for the probability that your error is equal to or larger than 0.01?

## 15.6 Confidence intervals

Confidence intervals are a very useful concept widely employed by data analysts. A version of these that are commonly seen come from the `ggplot` geometry `geom_smooth`. Here is an example using a temperature dataset available in R:



In the Machine Learning part we will learn how the curve is formed, but for now consider the shaded area around the curve. This is created using the concept of confidence intervals.

In our earlier competition, you were asked to give an interval. If the interval you submitted includes the  $p$ , you get half the money you spent on your “poll” back and pass to the next stage of the competition. One way to pass to the second round is to report a very large interval. For example, the interval  $[0, 1]$  is guaranteed to include  $p$ . However, with an interval this big, we have no chance of winning the competition. Similarly, if you are an election forecaster and predict the spread will be between -100% and 100%, you will be ridiculed for stating the obvious. Even a smaller interval, such as saying the spread will be between -10 and 10%, will not be considered serious.

On the other hand, the smaller the interval we report, the smaller our chances are of winning the prize. Likewise, a bold pollster that reports very small intervals and misses the mark most of the time will not be considered a good pollster. We want to be somewhere in between.

We can use the statistical theory we have learned to compute the probability of any given interval including  $p$ . If we are asked to create an interval with, say, a 95% chance of including  $p$ , we can do that as well. These are called 95% confidence intervals.

When a pollster reports an estimate and a margin of error, they are, in a way, reporting a 95% confidence interval. Let's show how this works mathematically.

We want to know the probability that the interval  $[\bar{X} - 2\hat{\text{SE}}(\bar{X}), \bar{X} + 2\hat{\text{SE}}(\bar{X})]$  contains

the true proportion  $p$ . First, consider that the start and end of these intervals are random variables: every time we take a sample, they change. To illustrate this, run the Monte Carlo simulation above twice. We use the same parameters as above:

```
p <- 0.45
N <- 1000
```

And notice that the interval here:

```
x <- sample(c(0, 1), size = N, replace = TRUE, prob = c(1-p, p))
x_hat <- mean(x)
se_hat <- sqrt(x_hat * (1 - x_hat) / N)
c(x_hat - 1.96 * se_hat, x_hat + 1.96 * se_hat)
#> [1] 0.418 0.480
```

is different from this one:

```
x <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
x_hat <- mean(x)
se_hat <- sqrt(x_hat * (1 - x_hat) / N)
c(x_hat - 1.96 * se_hat, x_hat + 1.96 * se_hat)
#> [1] 0.422 0.484
```

Keep sampling and creating intervals and you will see the random variation.

To determine the probability that the interval includes  $p$ , we need to compute this:

$$\Pr\left(\bar{X} - 1.96\hat{SE}(\bar{X}) \leq p \leq \bar{X} + 1.96\hat{SE}(\bar{X})\right)$$

By subtracting and dividing the same quantities in all parts of the equation, we get that the above is equivalent to:

$$\Pr\left(-1.96 \leq \frac{\bar{X} - p}{\hat{SE}(\bar{X})} \leq 1.96\right)$$

The term in the middle is an approximately normal random variable with expected value 0 and standard error 1, which we have been denoting with  $Z$ , so we have:

$$\Pr(-1.96 \leq Z \leq 1.96)$$

which we can quickly compute using :

```
pnorm(1.96) - pnorm(-1.96)
#> [1] 0.95
```

proving that we have a 95% probability.

If we want to have a larger probability, say 99%, we need to multiply by whatever  $z$  satisfies the following:

$$\Pr(-z \leq Z \leq z) = 0.99$$

Using:

```
z <- qnorm(0.995)
z
#> [1] 2.58
```

will achieve this because by definition `pnorm(qnorm(0.995))` is 0.995 and by symmetry `pnorm(1-qnorm(0.995))` is  $1 - 0.995$ . As a consequence, we have that:

```
pnorm(z) - pnorm(-z)
#> [1] 0.99
```

is  $0.995 - 0.005 = 0.99$ . We can use this approach for any proportion  $p$ : we set  $z = \text{qnorm}(1 - (1 - p)/2)$  because  $1 - (1 - p)/2 + (1 - p)/2 = p$ .

So, for example, for  $p = 0.95$ ,  $1 - (1 - p)/2 = 0.975$  and we get the 1.96 we have been using:

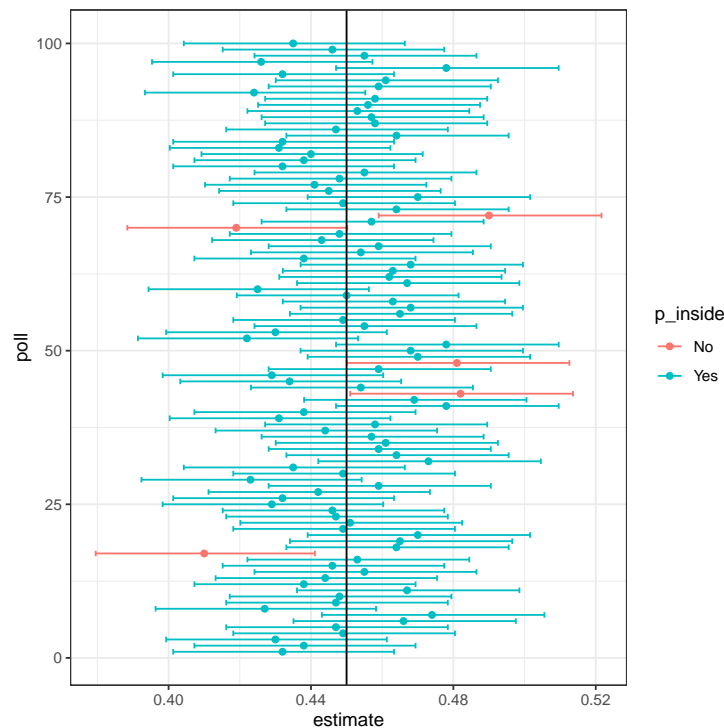
```
qnorm(0.975)
#> [1] 1.96
```

### 15.6.1 A Monte Carlo simulation

We can run a Monte Carlo simulation to confirm that, in fact, a 95% confidence interval includes  $p$  95% of the time.

```
N <- 1000
B <- 10000
inside <- replicate(B, {
 x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
 x_hat <- mean(x)
 se_hat <- sqrt(x_hat * (1 - x_hat) / N)
 between(p, x_hat - 1.96 * se_hat, x_hat + 1.96 * se_hat)
})
mean(inside)
#> [1] 0.948
```

The following plot shows the first 100 confidence intervals. In this case, we created the simulation so the black line denotes the parameter we are trying to estimate:



### 15.6.2 The correct language

When using the theory we described above, it is important to remember that it is the intervals that are random, not  $p$ . In the plot above, we can see the random intervals moving around and  $p$ , represented with the vertical line, staying in the same place. The proportion of blue in the urn  $p$  is not. So the 95% relates to the probability that this random interval falls on top of  $p$ . Saying the  $p$  has a 95% chance of being between this and that is technically an incorrect statement because  $p$  is not random.

## 15.7 Exercises

For these exercises, we will use actual polls from the 2016 election. You can load the data from the **dslabs** package.

```
library(dslabs)
data("polls_us_election_2016")
```

Specifically, we will use all the national polls that ended within one week before the election.

```
library(tidyverse)
polls <- polls_us_election_2016 %>%
 filter(enddate >= "2016-10-31" & state == "U.S.")
```



1. For the first poll, you can obtain the samples size and estimated Clinton percentage with:

```
N <- polls$samplesize[1]
x_hat <- polls$rawpoll_clinton[1]/100
```

Assume there are only two candidates and construct a 95% confidence interval for the election night proportion  $p$ .

2. Now use `dplyr` to add a confidence interval as two columns, call them `lower` and `upper`, to the object `poll`. Then use `select` to show the `pollster`, `enddate`, `x_hat`, `lower`, `upper` variables. Hint: define temporary columns `x_hat` and `se_hat`.

3. The final tally for the popular vote was Clinton 48.2% and Trump 46.1%. Add a column, call it `hit`, to the previous table stating if the confidence interval included the true proportion  $p = 0.482$  or not.

4. For the table you just created, what proportion of confidence intervals included  $p$ ?

5. If these confidence intervals are constructed correctly, and the theory holds up, what proportion should include  $p$ ?

6. A much smaller proportion of the polls than expected produce confidence intervals containing  $p$ . If you look closely at the table, you will see that most polls that fail to include  $p$  are underestimating. The reason for this is undecided voters, individuals polled that do not yet know who they will vote for or do not want to say. Because, historically, undecideds divide evenly between the two main candidates on election day, it is more informative to estimate the spread or the difference between the proportion of two candidates  $d$ , which in this election was  $0.482 - 0.461 = 0.021$ . Assume that there are only two parties and that  $d = 2p - 1$ , redefine `polls` as below and re-do exercise 1, but for the difference.

```
polls <- polls_us_election_2016 %>%
 filter(enddate >= "2016-10-31" & state == "U.S.") %>%
 mutate(d_hat = rawpoll_clinton / 100 - rawpoll_trump / 100)
```

7. Now repeat exercise 3, but for the difference.
8. Now repeat exercise 4, but for the difference.
9. Although the proportion of confidence intervals goes up substantially, it is still lower than 0.95. In the next chapter, we learn the reason for this. To motivate this, make a plot of the error, the difference between each poll's estimate and the actual  $d = 0.021$ . Stratify by `pollster`.
10. Redo the plot that you made for exercise 9, but only for pollsters that took five or more polls.

---

## 15.8 Power

Pollsters are not successful at providing correct confidence intervals, but rather at predicting who will win. When we took a 25 bead sample size, the confidence interval for the spread:

```

N <- 25
x_hat <- 0.48
(2 * x_hat - 1) + c(-1.96, 1.96) * 2 * sqrt(x_hat * (1 - x_hat) / N)
#> [1] -0.432 0.352

```

includes 0. If this were a poll and we were forced to make a declaration, we would have to say it was a “toss-up”.

A problem with our poll results is that given the sample size and the value of  $p$ , we would have to sacrifice on the probability of an incorrect call to create an interval that does not include 0.

This does not mean that the election is close. It only means that we have a small sample size. In statistical textbooks this is called lack of *power*. In the context of polls, *power* is the probability of detecting spreads different from 0.

By increasing our sample size, we lower our standard error and therefore have a much better chance of detecting the direction of the spread.

---

## 15.9 p-values

p-values are ubiquitous in the scientific literature. They are related to confidence intervals so we introduce the concept here.

Let’s consider the blue and red beads. Suppose that rather than wanting an estimate of the spread or the proportion of blue, I am interested only in the question: are there more blue beads or red beads? I want to know if the spread  $2p - 1 > 0$ .

Say we take a random sample of  $N = 100$  and we observe 52 blue beads, which gives us  $2\bar{X} - 1 = 0.04$ . This seems to be pointing to the existence of more blue than red beads since 0.04 is larger than 0. However, as data scientists we need to be skeptical. We know there is chance involved in this process and we could get a 52 even when the actual spread is 0. We call the assumption that the spread is  $2p - 1 = 0$  a *null hypothesis*. The null hypothesis is the skeptic’s hypothesis. We have observed a random variable  $2 * \bar{X} - 1 = 0.04$  and the p-value is the answer to the question: how likely is it to see a value this large, when the null hypothesis is true? So we write:

$$\Pr(|\bar{X} - 0.5| > 0.02)$$

assuming the  $2p - 1 = 0$  or  $p = 0.5$ . Under the null hypothesis we know that:

$$\sqrt{N} \frac{\bar{X} - 0.5}{\sqrt{0.5(1 - 0.5)}}$$

is standard normal. We therefore can compute the probability above, which is the p-value.

$$\Pr\left(\sqrt{N} \frac{|\bar{X} - 0.5|}{\sqrt{0.5(1 - 0.5)}} > \sqrt{N} \frac{0.02}{\sqrt{0.5(1 - 0.5)}}\right)$$

```
N <- 100
z <- sqrt(N)*0.02/0.5
1 - (pnorm(z) - pnorm(-z))
#> [1] 0.689
```

In this case, there is actually a large chance of seeing 52 or larger under the null hypothesis.

Keep in mind that there is a close connection between p-values and confidence intervals. If a 95% confidence interval of the spread does not include 0, we know that the p-value must be smaller than 0.05.

To learn more about p-values, you can consult any statistics textbook. However, in general, we prefer reporting confidence intervals over p-values since it gives us an idea of the size of the estimate. If we just report the p-value we provide no information about the significance of the finding in the context of the problem.

---

## 15.10 Association tests

The statistical tests we have studied up to now leave out a substantial portion of data types. Specifically, we have not discussed inference for binary, categorical, and ordinal data. To give a very specific example, consider the following case study.

A 2014 PNAS paper<sup>3</sup> analyzed success rates from funding agencies in the Netherlands and concluded that their:

results reveal gender bias favoring male applicants over female applicants in the prioritization of their “quality of researcher” (but not “quality of proposal”) evaluations and success rates, as well as in the language use in instructional and evaluation materials.

The main evidence for this conclusion comes down to a comparison of the percentages. Table S1 in the paper includes the information we need. Here are the three columns showing the overall outcomes:

```
library(tidyverse)
library(dslabs)
data("research_funding_rates")
research_funding_rates %>% select(discipline, applications_total,
 success_rates_total) %>% head()
#> discipline applications_total success_rates_total
#> 1 Chemical sciences 122 26.2
#> 2 Physical sciences 174 20.1
#> 3 Physics 76 26.3
#> 4 Humanities 396 16.4
#> 5 Technical sciences 251 17.1
#> 6 Interdisciplinary 183 15.8
```

---

<sup>3</sup><http://www.pnas.org/content/112/40/12349.abstract>

We have these values for each gender:

```
names(research_funding_rates)
#> [1] "discipline" "applications_total" "applications_men"
#> [4] "applications_women" "awards_total" "awards_men"
#> [7] "awards_women" "success_rates_total" "success_rates_men"
#> [10] "success_rates_women"
```

We can compute the totals that were successful and the totals that were not as follows:

```
totals <- research_funding_rates %>%
 select(-discipline) %>%
 summarize_all(sum) %>%
 summarize(yes_men = awards_men,
 no_men = applications_men - awards_men,
 yes_women = awards_women,
 no_women = applications_women - awards_women)
```

So we see that a larger percent of men than women received awards:

```
totals %>% summarize(percent_men = yes_men/(yes_men+no_men),
 percent_women = yes_women/(yes_women+no_women))
#> percent_men percent_women
#> 1 0.177 0.149
```

But could this be due just to random variability? Here we learn how to perform inference for this type of data.

### 15.10.1 Lady Tasting Tea

R.A. Fisher<sup>4</sup> was one of the first to formalize hypothesis testing. The “Lady Tasting Tea” is one of the most famous examples.

The story is as follows: an acquaintance of Fisher’s claimed that she could tell if milk was added before or after tea was poured. Fisher was skeptical. He designed an experiment to test this claim. He gave her four pairs of cups of tea: one with milk poured first, the other after. The order was randomized. The null hypothesis here is that she is guessing. Fisher derived the distribution for the number of correct picks on the assumption that the choices were random and independent.

As an example, suppose she picked 3 out of 4 correctly. Do we believe she has a special ability? The basic question we ask is: if the tester is actually guessing, what are the chances that she gets 3 or more correct? Just as we have done before, we can compute a probability under the null hypothesis that she is guessing 4 of each. Under this null hypothesis, we can think of this particular example as picking 4 balls out of an urn with 4 blue (correct answer) and 4 red (incorrect answer) balls. Remember, she knows that there are four before tea and four after.

Under the null hypothesis that she is simply guessing, each ball has the same chance of

<sup>4</sup>[https://en.wikipedia.org/wiki/Ronald\\_Fisher](https://en.wikipedia.org/wiki/Ronald_Fisher)

being picked. We can then use combinations to figure out each probability. The probability of picking 3 is  $\binom{4}{3}\binom{4}{1}/\binom{8}{4} = 16/70$ . The probability of picking all 4 correct is  $\binom{4}{4}\binom{4}{0}/\binom{8}{4} = 1/70$ . Thus, the chance of observing a 3 or something more extreme, under the null hypothesis, is  $\approx 0.24$ . This is the p-value. The procedure that produced this p-value is called *Fisher's exact test* and it uses the *hypergeometric distribution*.

### 15.10.2 Two-by-two tables

The data from the experiment is usually summarized by a table like this:

```
tab <- matrix(c(3,1,1,3),2,2)
rownames(tab)<-c("Poured Before","Poured After")
colnames(tab)<-c("Guessed before","Guessed after")
tab
#> Guessed before Guessed after
#> Poured Before 3 1
#> Poured After 1 3
```

These are referred to as a two-by-two table. For each of the four combinations one can get with a pair of binary variables, they show the observed counts for each occurrence.

The function `fisher.test` performs the inference calculations above:

```
fisher.test(tab, alternative="greater")$p.value
#> [1] 0.243
```

### 15.10.3 Chi-square Test

Notice that, in a way, our funding rates example is similar to the Lady Tasting Tea. However, in the Lady Tasting Tea example, the number of blue and red beads is experimentally fixed and the number of answers given for each category is also fixed. This is because Fisher made sure there were four cups with milk poured before tea and four cups with milk poured after and the lady knew this, so the answers would also have to include four before and four after. If this is the case, the sum of the rows and the sum of the columns are fixed. This defines constraints on the possible ways we can fill the two by two table and also permits us to use the hypergeometric distribution. In general, this is not the case. Nonetheless, there is another approach, the Chi-squared test, which is described below.

Imagine we have 290, 1,345, 177, 1,011 applicants, some are men and some are women and some get funded, whereas others don't. We saw that the success rates for men and woman were:

```
totals %>% summarize(percent_men = yes_men/(yes_men+no_men),
 percent_women = yes_women/(yes_women+no_women))
#> percent_men percent_women
#> 1 0.177 0.149
```

respectively. Would we see this again if we randomly assign funding at the overall rate:

```
rate <- totals %>%
 summarize(percent_total =
 (yes_men + yes_women)/
 (yes_men + no_men + yes_women + no_women)) %>%
 pull(percent_total)
rate
#> [1] 0.165
```

The Chi-square test answers this question. The first step is to create the two-by-two data table:

```
two_by_two <- data.frame(awarded = c("no", "yes"),
 men = c(totals$no_men, totals$yes_men),
 women = c(totals$no_women, totals$yes_women))

two_by_two
#> awarded men women
#> 1 no 1345 1011
#> 2 yes 290 177
```

The general idea of the Chi-square test is to compare this two-by-two table to what you expect to see, which would be:

```
data.frame(awarded = c("no", "yes"),
 men = (totals$no_men + totals$yes_men) * c(1 - rate, rate),
 women = (totals$no_women + totals$yes_women) * c(1 - rate, rate))
#> awarded men women
#> 1 no 1365 991
#> 2 yes 270 197
```

We can see that more men than expected and fewer women than expected received funding. However, under the null hypothesis these observations are random variables. The Chi-square test tells us how likely it is to see a deviation this large or larger. This test uses an asymptotic result, similar to the CLT, related to the sums of independent binary outcomes. The R function `chisq.test` takes a two-by-two table and returns the results from the test:

```
chisq_test <- two_by_two %>% select(-awarded) %>% chisq.test()
```

We see that the p-value is 0.0509:

```
chisq_test$p.value
#> [1] 0.0509
```

#### 15.10.4 The odds ratio

An informative summary statistic associated with two-by-two tables is the odds ratio. Define the two variables as  $X = 1$  if you are a male and 0 otherwise, and  $Y = 1$  if you are funded and 0 otherwise. The odds of getting funded if you are a man is defined:

$$\Pr(Y = 1 \mid X = 1) / \Pr(Y = 0 \mid X = 1)$$

and can be computed like this:

```
odds_men <- with(two_by_two, (men[2]/sum(men)) / (men[1]/sum(men)))
odds_men
#> [1] 0.216
```

And the odds of being funded if you are a woman is:

$$\Pr(Y = 1 \mid X = 0) / \Pr(Y = 0 \mid X = 0)$$

and can be computed like this:

```
odds_women <- with(two_by_two, (women[2]/sum(women)) / (women[1]/sum(women)))
odds_women
#> [1] 0.175
```

The odds ratio is the ratio for these two odds: how many times larger are the odds for men than for women?

```
odds_men / odds_women
#> [1] 1.23
```

We often see two-by-two tables written out as

|             | Men | Women |
|-------------|-----|-------|
| Awarded     | a   | b     |
| Not Awarded | c   | d     |

In this case, the odds ratio is  $\frac{a/c}{b/d}$  which is equivalent to  $(ad)/(bc)$

### 15.10.5 Confidence intervals for the odds ratio

Computing confidence intervals for the odds ratio is not mathematically straightforward. Unlike other statistics, for which we can derive useful approximations of their distributions, the odds ratio is not only a ratio, but a ratio of ratios. Therefore, there is no simple way of using, for example, the CLT.

However, statistical theory tells us that when all four entries of the two-by-two table are large enough, then the log of the odds ratio is approximately normal with standard error

$$\sqrt{1/a + 1/b + 1/c + 1/d}$$

This implies that a 95% confidence interval for the log odds ratio can be formed by:

$$\log\left(\frac{ad}{bc}\right) \pm 1.96\sqrt{1/a + 1/b + 1/c + 1/d}$$

By exponentiating these two numbers we can construct a confidence interval of the odds ratio.

Using R we can compute this confidence interval as follows:

```
log_or <- log(odds_men / odds_women)
se <- two_by_two %>% select(-awarded) %>%
 summarize(se = sqrt(sum(1/men) + sum(1/women))) %>%
 pull(se)
ci <- log_or + c(-1,1) * qnorm(0.975) * se
```

If we want to convert it back to the odds ratio scale, we can exponentiate:

```
exp(ci)
#> [1] 1.00 1.51
```

Note that 1 is not included in the confidence interval which must mean that the p-value is smaller than 0.05. We can confirm this using:

```
2*(1 - pnorm(log_or, 0, se))
#> [1] 0.0454
```

This is a slightly different p-value than that with the Chi-square test. This is because we are using a different asymptotic approximation to the null distribution. To learn more about inference and asymptotic theory for odds ratio, consult the *Generalized Linear Models* book by McCullagh and Nelder.

### 15.10.6 Small count correction

Note that the log odds ratio is not defined if any of the cells of the two-by-two table is 0. This is because if  $a$ ,  $b$ ,  $c$ , or  $d$  is 0, the  $\log(\frac{ad}{bc})$  is either the log of 0 or has a 0 in the denominator. For this situation, it is common practice to avoid 0s by adding 0.5 to each cell. This is referred to as the *Haldane-Anscombe correction* and has been shown, both in practice and theory, to work well.

### 15.10.7 Large samples, small p-values

As mentioned earlier, reporting only p-values is not an appropriate way to report the results of data analysis. In scientific journals, for example, some studies seem to overemphasize p-values. Some of these studies have large sample sizes and report impressively small p-values. Yet when one looks closely at the results, we realize odds ratios are quite modest: barely bigger than 1. In this case the difference may not be *practically significant* or *scientifically significant*.

Note that the relationship between odds ratio and p-value is not one-to-one. It depends on



the sample size. So a very small p-value does not necessarily mean a very large odds ratio. Notice what happens to the p-value if we multiply our two-by-two table by 10, which does not change the odds ratio:

```
two_by_two %>% select(-awarded) %>%
 mutate(men = men*10, women = women*10) %>%
 chisq.test() %>% .$.p.value
#> [1] 2.63e-10
```

---

### 15.11 Exercises

1. A famous athlete has an impressive career, winning 70% of her 500 career matches. However, this athlete gets criticized because in important events, such as the Olympics, she has a losing record of 8 wins and 9 losses. Perform a Chi-square test to determine if this losing record can be simply due to chance as opposed to not performing well under pressure.
2. Why did we use the Chi-square test instead of Fisher's exact test in the previous exercise?
  - a. It actually does not matter, since they give the exact same p-value.
  - b. Fisher's exact and the Chi-square are different names for the same test.
  - c. Because the sum of the rows and columns of the two-by-two table are not fixed so the hypergeometric distribution is not an appropriate assumption for the null hypothesis. For this reason, Fisher's exact test is rarely applicable with observational data.
  - d. Because the Chi-square test runs faster.
3. Compute the odds ratio of "losing under pressure" along with a confidence interval.
4. Notice that the p-value is larger than 0.05 but the 95% confidence interval does not include 1. What explains this?
  - a. We made a mistake in our code.
  - b. These are not t-tests so the connection between p-value and confidence intervals does not apply.
  - c. Different approximations are used for the p-value and the confidence interval calculation. If we had a larger sample size the match would be better.
  - d. We should use the Fisher exact test to get confidence intervals.
5. Multiply the two-by-two table by 2 and see if the p-value and confidence retrieval are a better match.

# 16

## *Statistical models*

---

“All models are wrong, but some are useful.” –George E. P. Box

The day before the 2008 presidential election, Nate Silver’s FiveThirtyEight stated that “Barack Obama appears poised for a decisive electoral victory”. They went further and predicted that Obama would win the election with 349 electoral votes to 189, and the popular vote by a margin of 6.1%. FiveThirtyEight also attached a probabilistic statement to their prediction claiming that Obama had a 91% chance of winning the election. The predictions were quite accurate since, in the final results, Obama won the electoral college 365 to 173 and the popular vote by a 7.2% difference. Their performance in the 2008 election brought FiveThirtyEight to the attention of political pundits and TV personalities. Four years later, the week before the 2012 presidential election, FiveThirtyEight’s Nate Silver was giving Obama a 90% chance of winning despite many of the experts thinking the final results would be closer. Political commentator Joe Scarborough said during his show<sup>1</sup>:

Anybody that thinks that this race is anything but a toss-up right now is such an ideologue ... they’re jokes.

To which Nate Silver responded via Twitter:

If you think it’s a toss-up, let’s bet. If Obama wins, you donate \$1,000 to the American Red Cross. If Romney wins, I do. Deal?

In 2016, Silver was not as certain and gave Hillary Clinton only a 71% of winning. In contrast, most other forecasters were almost certain she would win. She lost. But 71% is still more than 50%, so was Mr. Silver wrong? And what does probability mean in this context anyway? Are dice being tossed somewhere?

In this chapter we will demonstrate how *poll aggregators*, such as FiveThirtyEight, collected and combined data reported by different experts to produce improved predictions. We will introduce ideas behind the *statistical models*, also known as *probability models*, that were used by poll aggregators to improve election forecasts beyond the power of individual polls. In this chapter, we motivate the models, building on the statistical inference concepts we learned in Chapter 15. We start with relatively simple models, realizing that the actual data science exercise of forecasting elections involves rather complex ones, which we introduce towards the end of the chapter in Section 16.8.

---

<sup>1</sup><https://www.youtube.com/watch?v=TbKkjm-gheY>

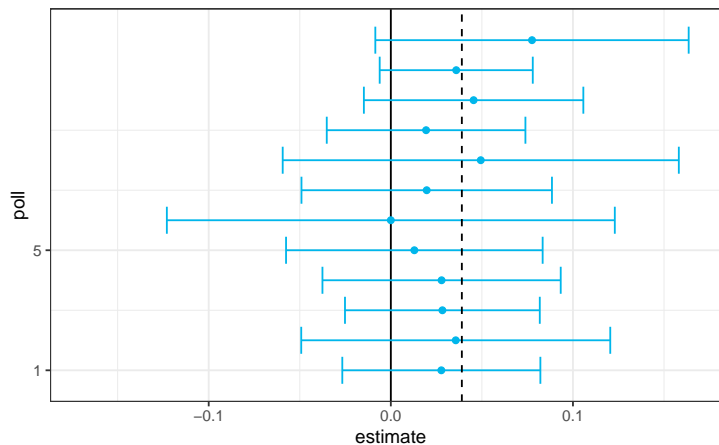
## 16.1 Poll aggregators

As we described earlier, a few weeks before the 2012 election Nate Silver was giving Obama a 90% chance of winning. How was Mr. Silver so confident? We will use a Monte Carlo simulation to illustrate the insight Mr. Silver had and others missed. To do this, we generate results for 12 polls taken the week before the election. We mimic sample sizes from actual polls and construct and report 95% confidence intervals for each of the 12 polls. We save the results from this simulation in a data frame and add a poll ID column.

```
library(tidyverse)
library(dslabs)
d <- 0.039
Ns <- c(1298, 533, 1342, 897, 774, 254, 812, 324, 1291, 1056, 2172, 516)
p <- (d + 1) / 2

polls <- map_df(Ns, function(N) {
 x <- sample(c(0,1), size=N, replace=TRUE, prob=c(1-p, p))
 x_hat <- mean(x)
 se_hat <- sqrt(x_hat * (1 - x_hat) / N)
 list(estimate = 2 * x_hat - 1,
 low = 2*(x_hat - 1.96*se_hat) - 1,
 high = 2*(x_hat + 1.96*se_hat) - 1,
 sample_size = N)
}) %>% mutate(poll = seq_along(Ns))
```

Here is a visualization showing the intervals the pollsters would have reported for the difference between Obama and Romney:



Not surprisingly, all 12 polls report confidence intervals that include the election night result (dashed line). However, all 12 polls also include 0 (solid black line) as well. Therefore, if asked individually for a prediction, the pollsters would have to say: it's a toss-up. Below we describe a key insight they are missing.

Poll aggregators, such as Nate Silver, realized that by combining the results of different polls you could greatly improve precision. By doing this, we are effectively conducting a poll with a huge sample size. We can therefore report a smaller 95% confidence interval and a more precise prediction.

Although as aggregators we do not have access to the raw poll data, we can use mathematics to reconstruct what we would have obtained had we made one large poll with:

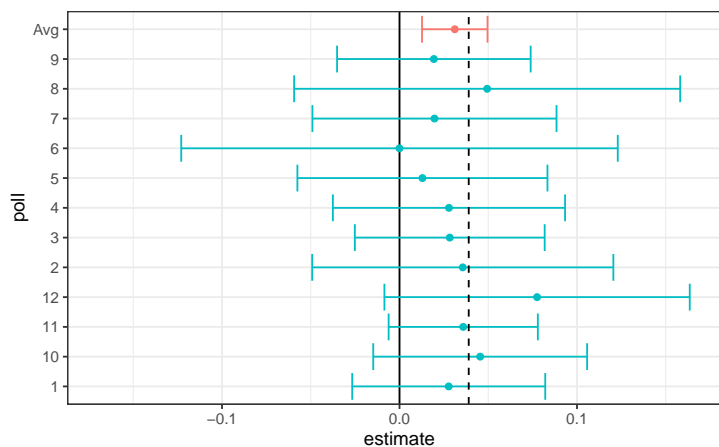
```
sum(polls$sample_size)
#> [1] 11269
```

participants. Basically, we construct an estimate of the spread, let's call it  $d$ , with a weighted average in the following way:

```
d_hat <- polls %>%
 summarize(avg = sum(estimate*sample_size) / sum(sample_size)) %>%
 pull(avg)
```

Once we have an estimate of  $d$ , we can construct an estimate for the proportion voting for Obama, which we can then use to estimate the standard error. Once we do this, we see that our margin of error is 0.018.

Thus, we can predict that the spread will be 3.1 plus or minus 1.8, which not only includes the actual result we eventually observed on election night, but is quite far from including 0. Once we combine the 12 polls, we become quite certain that Obama will win the popular vote.



Of course, this was just a simulation to illustrate the idea. The actual data science exercise of forecasting elections is much more complicated and it involves modeling. Below we explain how pollsters fit multilevel models to the data and use this to forecast election results. In the 2008 and 2012 US presidential elections, Nate Silver used this approach to make an almost perfect prediction and silence the pundits.

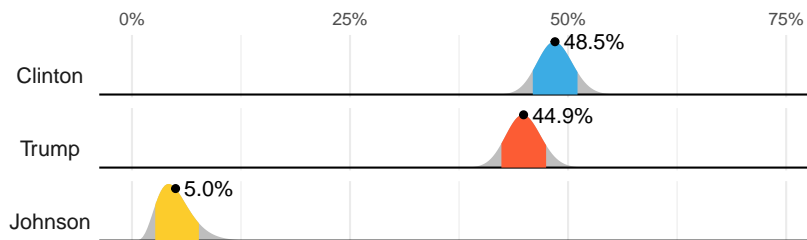
Since the 2008 elections, other organizations have started their own election forecasting group that, like Nate Silver's, aggregates polling data and uses statistical models to make predictions. In 2016, forecasters underestimated Trump's chances of winning greatly. The

day before the election the *New York Times* reported<sup>2</sup> the following probabilities for Hillary Clinton winning the presidency:

|          | NYT | 538 | HuffPost | PW  | PEC  | DK  | Cook     | Roth     |
|----------|-----|-----|----------|-----|------|-----|----------|----------|
| Win Prob | 85% | 71% | 98%      | 89% | >99% | 92% | Lean Dem | Lean Dem |

For example, the Princeton Election Consortium (PEC) gave Trump less than 1% chance of winning, while the Huffington Post gave him a 2% chance. In contrast, FiveThirtyEight had Trump’s probability of winning at 29%, higher than tossing two coins and getting two heads. In fact, four days before the election FiveThirtyEight published an article titled *Trump Is Just A Normal Polling Error Behind Clinton*<sup>3</sup>. By understanding statistical models and how these forecasters use them, we will start to understand how this happened.

Although not nearly as interesting as predicting the electoral college, for illustrative purposes we will start by looking at predictions for the popular vote. FiveThirtyEight predicted a 3.6% advantage for Clinton<sup>4</sup>, included the actual result of 2.1% (48.2% to 46.1%) in their interval, and was much more confident about Clinton winning the election, giving her an 81.4% chance. Their prediction was summarized with a chart like this:



The colored areas represent values with an 80% chance of including the actual result, according to the FiveThirtyEight model.

We introduce actual data from the 2016 US presidential election to show how models are motivated and built to produce these predictions. To understand the “81.4% chance” statement we need to describe Bayesian statistics, which we do in Sections 16.4 and 16.8.1.

### 16.1.1 Poll data

We use public polling data organized by FiveThirtyEight for the 2016 presidential election. The data is included as part of the **dslabs** package:

```
data(polls_us_election_2016)
```

The table includes results for national polls, as well as state polls, taken during the year prior to the election. For this first example, we will filter the data to include national polls

<sup>2</sup><https://www.nytimes.com/interactive/2016/upshot/presidential-polls-forecast.html>

<sup>3</sup><https://fivethirtyeight.com/features/trump-is-just-a-normal-polling-error-behind-clinton/>

<sup>4</sup><https://projects.fivethirtyeight.com/2016-election-forecast/>

conducted during the week before the election. We also remove polls that FiveThirtyEight has determined not to be reliable and graded with a “B” or less. Some polls have not been graded and we include those:

```
polls <- polls_us_election_2016 %>%
 filter(state == "U.S." & enddate >= "2016-10-31" &
 (grade %in% c("A+", "A", "A-", "B+") | is.na(grade)))
```

We add a spread estimate:

```
polls <- polls %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

For this example, we will assume that there are only two parties and call  $p$  the proportion voting for Clinton and  $1 - p$  the proportion voting for Trump. We are interested in the spread  $2p - 1$ . Let’s call the spread  $d$  (for difference).

We have 49 estimates of the spread. The theory we learned tells us that these estimates are a random variable with a probability distribution that is approximately normal. The expected value is the election night spread  $d$  and the standard error is  $2\sqrt{p(1-p)/N}$ . Assuming the urn model we described earlier is a good one, we can use this information to construct a confidence interval based on the aggregated data. The estimated spread is:

```
d_hat <- polls %>%
 summarize(d_hat = sum(spread * samplesize) / sum(samplesize)) %>%
 pull(d_hat)
```

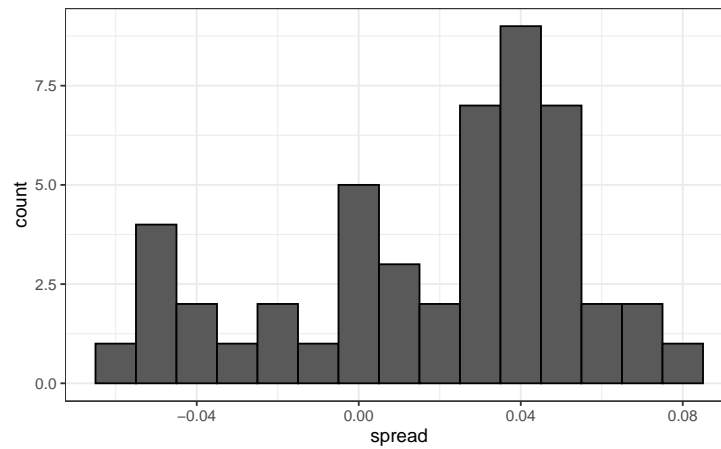
and the standard error is:

```
p_hat <- (d_hat+1)/2
moe <- 1.96 * 2 * sqrt(p_hat * (1 - p_hat) / sum(polls$samplesize))
moe
#> [1] 0.00662
```

So we report a spread of 1.43% with a margin of error of 0.66%. On election night, we discover that the actual percentage was 2.1%, which is outside a 95% confidence interval. What happened?

A histogram of the reported spreads shows a problem:

```
polls %>%
 ggplot(aes(spread)) +
 geom_histogram(color="black", binwidth = .01)
```



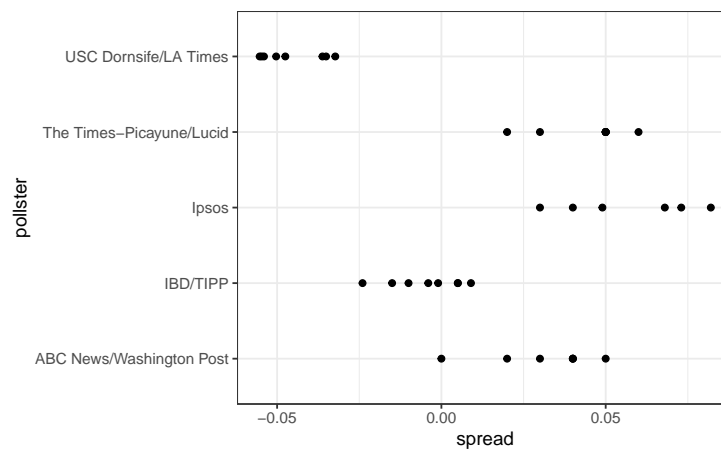
The data does not appear to be normally distributed and the standard error appears to be larger than 0.007. The theory is not quite working here.

### 16.1.2 Pollster bias

Notice that various pollsters are involved and some are taking several polls a week:

```
polls %>% group_by(pollster) %>% summarize(n())
#> # A tibble: 15 x 2
#> pollster `n()`
#> <fct> <int>
#> 1 ABC News/Washington Post 7
#> 2 Angus Reid Global 1
#> 3 CBS News/New York Times 2
#> 4 Fox News/Anderson Robbins Research/Shaw & Company Research 2
#> 5 IBD/TIPP 8
#> # ... with 10 more rows
```

Let's visualize the data for the pollsters that are regularly polling:



This plot reveals an unexpected result. First, consider that the standard error predicted by theory for each poll:

```
polls %>% group_by(pollster) %>%
 filter(n() >= 6) %>%
 summarize(se = 2 * sqrt(p_hat * (1-p_hat) / median(samplesize)))
#> # A tibble: 5 x 2
#> pollster se
#> <fct> <dbl>
#> 1 ABC News/Washington Post 0.0265
#> 2 IBD/TIPP 0.0333
#> 3 Ipsos 0.0225
#> 4 The Times-Picayune/Lucid 0.0196
#> 5 USC Dornsife/LA Times 0.0183
```

is between 0.018 and 0.033, which agrees with the within poll variation we see. However, there appears to be differences *across the polls*. Note, for example, how the USC Dornsife/LA Times pollster is predicting a 4% win for Trump, while Ipsos is predicting a win larger than 5% for Clinton. The theory we learned says nothing about different pollsters producing polls with different expected values. All the polls should have the same expected value. FiveThirtyEight refers to these differences as “house effects”. We also call them *pollster bias*.

In the following section, rather than use the urn model theory, we are instead going to develop a data-driven model.

---

## 16.2 Data-driven models

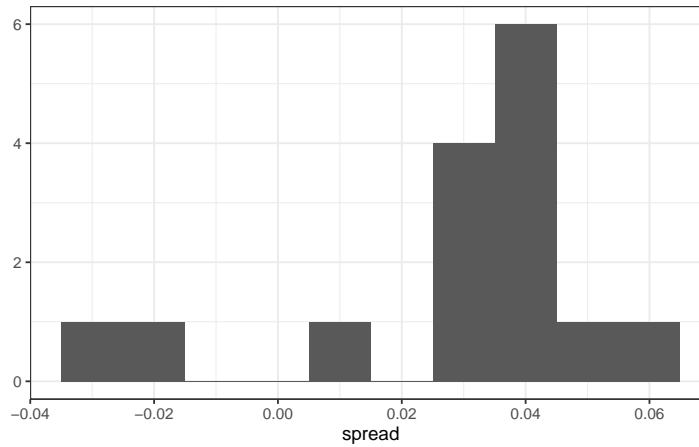
For each pollster, let’s collect their last reported result before the election:

```
one_poll_per_pollster <- polls %>% group_by(pollster) %>%
 filter(enddate == max(enddate)) %>%
 ungroup()
```

Here is a histogram of the data for these 15 pollsters:

```
qplot(spread, data = one_poll_per_pollster, binwidth = 0.01)
```





In the previous section, we saw that using the urn model theory to combine these results might not be appropriate due to the pollster effect. Instead, we will model this spread data directly.

The new model can also be thought of as an urn model, although the connection is not as direct. Rather than 0s (Republicans) and 1s (Democrats), our urn now contains poll results from all possible pollsters. We *assume* that the expected value of our urn is the actual spread  $d = 2p - 1$ .

Because instead of 0s and 1s, our urn contains continuous numbers between -1 and 1, the standard deviation of the urn is no longer  $\sqrt{p(1-p)}$ . Rather than voter sampling variability, the standard error now includes the pollster-to-pollster variability. Our new urn also includes the sampling variability from the polling. Regardless, this standard deviation is now an unknown parameter. In statistics textbooks, the Greek symbol  $\sigma$  is used to represent this parameter.

In summary, we have two unknown parameters: the expected value  $d$  and the standard deviation  $\sigma$ .

Our task is to estimate  $d$ . Because we model the observed values  $X_1, \dots, X_N$  as a random sample from the urn, the CLT might still work in this situation because it is an average of independent random variables. For a large enough sample size  $N$ , the probability distribution of the sample average  $\bar{X}$  is approximately normal with expected value  $\mu$  and standard error  $\sigma/\sqrt{N}$ . If we are willing to consider  $N = 15$  large enough, we can use this to construct confidence intervals.

A problem is that we don't know  $\sigma$ . But theory tells us that we can estimate the urn model  $\sigma$  with the *sample standard deviation* defined as  $s = \sqrt{\sum_{i=1}^N (X_i - \bar{X})^2 / (N - 1)}$ .

Unlike for the population standard deviation definition, we now divide by  $N - 1$ . This makes  $s$  a better estimate of  $\sigma$ . There is a mathematical explanation for this, which is explained in most statistics textbooks, but we don't cover it here.

The `sd` function in R computes the sample standard deviation:

```
sd(one_poll_per_pollster$spread)
#> [1] 0.0242
```

We are now ready to form a new confidence interval based on our new data-driven model:

```

results <- one_poll_per_pollster %>%
 summarize(avg = mean(spread),
 se = sd(spread) / sqrt(length(spread))) %>%
 mutate(start = avg - 1.96 * se,
 end = avg + 1.96 * se)
round(results * 100, 1)
#> avg se start end
#> 1 2.9 0.6 1.7 4.1

```

Our confidence interval is wider now since it incorporates the pollster variability. It does include the election night result of 2.1%. Also, note that it was small enough not to include 0, which means we were confident Clinton would win the popular vote.

Are we now ready to declare a probability of Clinton winning the popular vote? Not yet. In our model  $d$  is a fixed parameter so we can't talk about probabilities. To provide probabilities, we will need to learn about Bayesian statistics.

---

## 16.3 Exercises

We have been using urn models to motivate the use of probability models. Most data science applications are not related to data obtained from urns. More common are data that come from individuals. The reason probability plays a role here is because the data come from a random sample. The random sample is taken from a population and the urn serves as an analogy for the population.

Let's revisit the heights dataset. Suppose we consider the males in our course the population.

```

library(dslabs)
data(heights)
x <- heights %>% filter(sex == "Male") %>%
 pull(height)

```

1. Mathematically speaking,  $\mathbf{x}$  is our population. Using the urn analogy, we have an urn with the values of  $\mathbf{x}$  in it. What are the average and standard deviation of our population?
2. Call the population average computed above  $\mu$  and the standard deviation  $\sigma$ . Now take a sample of size 50, with replacement, and construct an estimate for  $\mu$  and  $\sigma$ .
3. What does the theory tell us about the sample average  $\bar{X}$  and how it is related to  $\mu$ ?
  - a. It is practically identical to  $\mu$ .
  - b. It is a random variable with expected value  $\mu$  and standard error  $\sigma/\sqrt{N}$ .
  - c. It is a random variable with expected value  $\mu$  and standard error  $\sigma$ .
  - d. Contains no information.

4. So how is this useful? We are going to use an oversimplified yet illustrative example. Suppose we want to know the average height of our male students, but we only get to measure 50 of the 708. We will use  $\bar{X}$  as our estimate. We know from the answer to exercise

3 that the standard estimate of our error  $\bar{X} - \mu$  is  $\sigma/\sqrt{N}$ . We want to compute this, but we don't know  $\sigma$ . Based on what is described in this section, show your estimate of  $\sigma$ .

5. Now that we have an estimate of  $\sigma$ , let's call our estimate  $s$ . Construct a 95% confidence interval for  $\mu$ .

6. Now run a Monte Carlo simulation in which you compute 10,000 confidence intervals as you have just done. What proportion of these intervals include  $\mu$ ?

7. In this section, we talked about pollster bias. We used visualization to motivate the presence of such bias. Here we will give it a more rigorous treatment. We will consider two pollsters that conducted daily polls. We will look at national polls for the month before the election.

```
data(polls_us_election_2016)
polls <- polls_us_election_2016 %>%
 filter(pollster %in% c("Rasmussen Reports/Pulse Opinion Research",
 "The Times-Picayune/Lucid") &
 enddate >= "2016-10-15" &
 state == "U.S.") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

We want to answer the question: is there a poll bias? Make a plot showing the spreads for each poll.

8. The data does seem to suggest there is a difference. However, these data are subject to variability. Perhaps the differences we observe are due to chance.

The urn model theory says nothing about pollster effect. Under the urn model, both pollsters have the same expected value: the election day difference, that we call  $d$ .

To answer the question “is there an urn model?”, we will model the observed data  $Y_{i,j}$  in the following way:

$$Y_{i,j} = d + b_i + \varepsilon_{i,j}$$

with  $i = 1, 2$  indexing the two pollsters,  $b_i$  the bias for pollster  $i$  and  $\varepsilon_{i,j}$  poll to poll chance variability. We assume the  $\varepsilon$  are independent from each other, have expected value 0 and standard deviation  $\sigma_i$  regardless of  $j$ .

Which of the following best represents our question?

- a. Is  $\varepsilon_{i,j} = 0$ ?
- b. How close are the  $Y_{i,j}$  to  $d$ ?
- c. Is  $b_1 \neq b_2$ ?
- d. Are  $b_1 = 0$  and  $b_2 = 0$ ?

9. In the right side of this model only  $\varepsilon_{i,j}$  is a random variable. The other two are constants. What is the expected value of  $Y_{1,j}$ ?

10. Suppose we define  $\bar{Y}_1$  as the average of poll results from the first poll,  $Y_{1,1}, \dots, Y_{1,N_1}$  with  $N_1$  the number of polls conducted by the first pollster:

```
polls %>%
 filter(pollster=="Rasmussen Reports/Pulse Opinion Research") %>%
 summarize(N_1 = n())
```

What is the expected values  $\bar{Y}_1$ ?

11. What is the standard error of  $\bar{Y}_1$ ?
12. Suppose we define  $\bar{Y}_2$  as the average of poll results from the first poll,  $Y_{2,1}, \dots, Y_{2,N_2}$  with  $N_2$  the number of polls conducted by the first pollster. What is the expected value  $\bar{Y}_2$ ?
13. What is the standard error of  $\bar{Y}_2$ ?
14. Using what we learned by answering the questions above, what is the expected value of  $\bar{Y}_2 - \bar{Y}_1$ ?
15. Using what we learned by answering the questions above, what is the standard error of  $\bar{Y}_2 - \bar{Y}_1$ ?
16. The answer to the question above depends on  $\sigma_1$  and  $\sigma_2$ , which we don't know. We learned that we can estimate these with the sample standard deviation. Write code that computes these two estimates.
17. What does the CLT tell us about the distribution of  $\bar{Y}_2 - \bar{Y}_1$ ?
  - a. Nothing because this is not the average of a sample.
  - b. Because the  $Y_{ij}$  are approximately normal, so are the averages.
  - c. Note that  $\bar{Y}_2$  and  $\bar{Y}_1$  are sample averages, so if we assume  $N_2$  and  $N_1$  are large enough, each is approximately normal. The difference of normals is also normal.
  - d. The data are not 0 or 1, so CLT does not apply.
18. We have constructed a random variable that has expected value  $b_2 - b_1$ , the pollster bias difference. If our model holds, then this random variable has an approximately normal distribution and we know its standard error. The standard error depends on  $\sigma_1$  and  $\sigma_2$ , but we can plug the sample standard deviations we computed above. We started off by asking: is  $b_2 - b_1$  different from 0? Use all the information we have learned above to construct a 95% confidence interval for the difference  $b_2$  and  $b_1$ .
19. The confidence interval tells us there is relatively strong pollster effect resulting in a difference of about 5%. Random variability does not seem to explain it. We can compute a p-value to relay the fact that chance does not explain it. What is the p-value?
20. The statistic formed by dividing our estimate of  $b_2 - b_1$  by its estimated standard error:

$$\frac{\bar{Y}_2 - \bar{Y}_1}{\sqrt{s_2^2/N_2 + s_1^2/N_1}}$$

is called the t-statistic. Now notice that we have more than two pollsters. We can also test for pollster effect using all pollsters, not just two. The idea is to compare the variability across polls to variability within polls. We can actually construct statistics to test for effects and approximate their distribution. The area of statistics that does this is called Analysis of Variance or ANOVA. We do not cover it here, but ANOVA provides a very useful set of tools to answer questions such as: is there a pollster effect?

For this exercise, create a new table:

```
polls <- polls_us_election_2016 %>%
 filter(enddate >= "2016-10-15" &
 state == "U.S.") %>%
 group_by(pollster) %>%
 filter(n() >= 5) %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100) %>%
 ungroup()
```

Compute the average and standard deviation for each pollster and examine the variability across the averages and how it compares to the variability within the pollsters, summarized by the standard deviation.

---

## 16.4 Bayesian statistics

What does it mean when an election forecaster tells us that a given candidate has a 90% chance of winning? In the context of the urn model, this would be equivalent to stating that the probability  $p > 0.5$  is 90%. However, as we discussed earlier, in the urn model  $p$  is a fixed parameter and it does not make sense to talk about probability. With Bayesian statistics, we model  $p$  as random variable and thus a statement such as “90% chance of winning” is consistent with the approach.

Forecasters also use models to describe variability at different levels. For example, sampling variability, pollster to pollster variability, day to day variability, and election to election variability. One of the most successful approaches used for this are hierarchical models, which can be explained in the context of Bayesian statistics.

In this chapter we briefly describe Bayesian statistics. For an in-depth treatment of this topic we recommend one of the following textbooks:

- Berger JO (1985). Statistical Decision Theory and Bayesian Analysis, 2nd edition. Springer-Verlag.
- Lee PM (1989). Bayesian Statistics: An Introduction. Oxford.

### 16.4.1 Bayes theorem

We start by describing Bayes theorem. We do this using a hypothetical cystic fibrosis test as an example. Suppose a test for cystic fibrosis has an accuracy of 99%. We will use the following notation:

$$\text{Prob}(+ \mid D = 1) = 0.99, \text{Prob}(- \mid D = 0) = 0.99$$

with  $+$  meaning a positive test and  $D$  representing if you actually have the disease (1) or not (0).

Suppose we select a random person and they test positive. What is the probability that

they have the disease? We write this as  $\text{Prob}(D = 1 \mid +)$ ? The cystic fibrosis rate is 1 in 3,900 which implies that  $\text{Prob}(D = 1) = 0.00025$ . To answer this question, we will use Bayes theorem, which in general tells us that:

$$\Pr(A \mid B) = \frac{\Pr(B \mid A)\Pr(A)}{\Pr(B)}$$

This equation applied to our problem becomes:

$$\begin{aligned}\Pr(D = 1 \mid +) &= \frac{P(+ \mid D = 1) \cdot P(D = 1)}{\Pr(+)} \\ &= \frac{\Pr(+ \mid D = 1) \cdot P(D = 1)}{\Pr(+ \mid D = 1) \cdot P(D = 1) + \Pr(+ \mid D = 0)\Pr(D = 0)}\end{aligned}$$

Plugging in the numbers we get:

$$\frac{0.99 \cdot 0.00025}{0.99 \cdot 0.00025 + 0.01 \cdot (.99975)} = 0.02$$

This says that despite the test having 0.99 accuracy, the probability of having the disease given a positive test is only 0.02. This may appear counter-intuitive to some, but the reason this is the case is because we have to factor in the very rare probability that a person, chosen at random, has the disease. To illustrate this, we run a Monte Carlo simulation.

---

## 16.5 Bayes theorem simulation

The following simulation is meant to help you visualize Bayes theorem. We start by randomly selecting 100,000 people from a population in which the disease in question has a 1 in 4,000 prevalence.

```
prev <- 0.00025
N <- 100000
outcome <- sample(c("Disease", "Healthy"), N, replace = TRUE,
 prob = c(prev, 1 - prev))
```

Note that there are very few people with the disease:

```
N_D <- sum(outcome == "Disease")
N_D
#> [1] 23
N_H <- sum(outcome == "Healthy")
N_H
#> [1] 99977
```

Also, there are many without the disease, which makes it more probable that we will see some false positives given that the test is not perfect. Now each person gets the test, which is correct 99% of the time:

```

accuracy <- 0.99
test <- vector("character", N)
test[outcome == "Disease"] <- sample(c("+", "-"), N_D, replace = TRUE,
 prob = c(accuracy, 1 - accuracy))
test[outcome == "Healthy"] <- sample(c("-", "+"), N_H, replace = TRUE,
 prob = c(accuracy, 1 - accuracy))

```

Because there are so many more controls than cases, even with a low false positive rate we get more controls than cases in the group that tested positive:

```

table(outcome, test)
#> test
#> outcome - +
#> Disease 0 23
#> Healthy 99012 965

```

From this table, we see that the proportion of positive tests that have the disease is 23 out of 988. We can run this over and over again to see that, in fact, the probability converges to about 0.022.

### 16.5.1 Bayes in practice

José Iglesias is a professional baseball player. In April 2013, when he was starting his career, he was performing rather well:

| Month | At Bats | H | AVG  |
|-------|---------|---|------|
| April | 20      | 9 | .450 |

The batting average (**AVG**) statistic is one way of measuring success. Roughly speaking, it tells us the success rate when batting. An **AVG** of .450 means José has been successful 45% of the times he has batted (**At Bats**) which is rather high, historically speaking. Keep in mind that no one has finished a season with an **AVG** of .400 or more since Ted Williams did it in 1941! To illustrate the way hierarchical models are powerful, we will try to predict José's batting average at the end of the season. Note that in a typical season, players have about 500 at bats.

With the techniques we have learned up to now, referred to as *frequentist techniques*, the best we can do is provide a confidence interval. We can think of outcomes from hitting as a binomial with a success rate of  $p$ . So if the success rate is indeed .450, the standard error of just 20 at bats is:

$$\sqrt{\frac{.450(1 - .450)}{20}} = .111$$

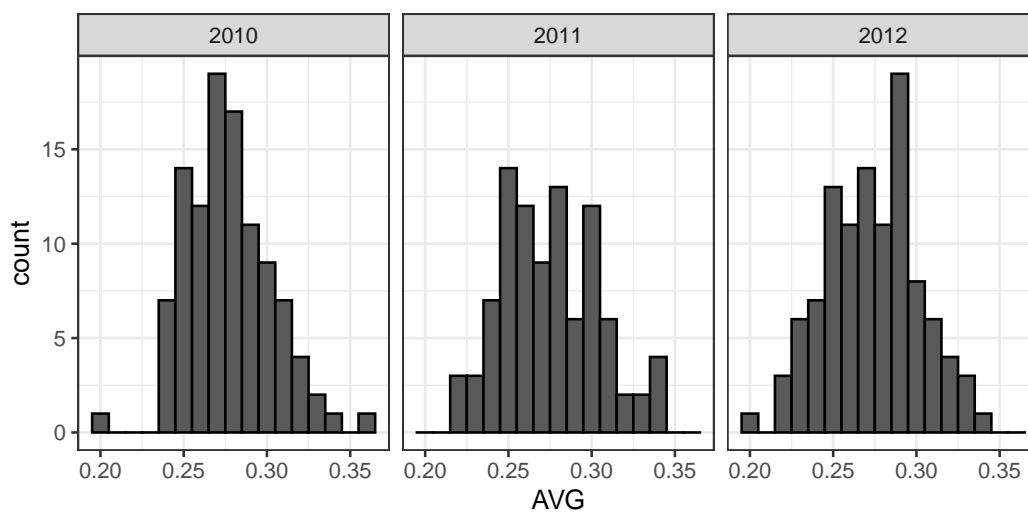
This means that our confidence interval is  $.450 - .222$  to  $.450 + .222$  or .228 to .672.

This prediction has two problems. First, it is very large, so not very useful. Second, it is

centered at .450, which implies that our best guess is that this new player will break Ted Williams' record.

If you follow baseball, this last statement will seem wrong and this is because you are implicitly using a hierarchical model that factors in information from years of following baseball. Here we show how we can quantify this intuition.

First, let's explore the distribution of batting averages for all players with more than 500 at bats during the previous three seasons:



The average player had an *AVG* of .275 and the standard deviation of the population of players was 0.027. So we can see already that .450 would be quite an anomaly since it is over six standard deviations away from the mean.

So is José lucky or is he the best batter seen in the last 50 years? Perhaps it's a combination of both luck and talent. But how much of each? If we become convinced that he is lucky, we should trade him to a team that trusts the .450 observation and is maybe overestimating his potential.

## 16.6 Hierarchical models

The hierarchical model provides a mathematical description of how we came to see the observation of .450. First, we pick a player at random with an intrinsic ability summarized by, for example,  $p$ . Then we see 20 random outcomes with success probability  $p$ .

We use a model to represent two levels of variability in our data. First, each player is assigned a natural ability to hit. We will use the symbol  $p$  to represent this ability. You can think of  $p$  as the batting average you would converge to if this particular player batted over and over again.



Based on the plots we showed earlier, we assume that  $p$  has a normal distribution. With expected value .270 and standard error 0.027.

Now the second level of variability has to do with luck when batting. Regardless of how good the player is, sometimes you have bad luck and sometimes you have good luck. At each at bat, this player has a probability of success  $p$ . If we add up these successes and failures, then the CLT tells us that the observed average, call it  $Y$ , has a normal distribution with expected value  $p$  and standard error  $\sqrt{p(1-p)/N}$  with  $N$  the number of at bats.

Statistical textbooks will write the model like this:

$$\begin{aligned} p &\sim N(\mu, \tau^2) \\ Y | p &\sim N(p, \sigma^2) \end{aligned}$$

Here the  $\sim$  symbol tells us the random variable on the left of the symbol follows the distribution on the right and  $N(a, b^2)$  represents the normal distribution with mean  $a$  and standard deviation  $b$ . The  $|$  is read as *conditioned on*, and it means that we are treating the random variable to the right of the symbol as known. We refer to the model as hierarchical because we need to know  $p$ , the first level, in order to model  $Y$ , the second level. In our example the first level describes randomness in assigning talent to a player and the second describes randomness in this particular player's performance once we have fixed the talent parameter. In a Bayesian framework, the first level is called a *prior distribution* and the second the *sampling distribution*. The data analysis we have conducted here suggests that we set  $\mu = .270$ ,  $\tau = 0.027$ , and  $\sigma^2 = p(1-p)/N$ .

Now, let's use this model for José's data. Suppose we want to predict his innate ability in the form of his *true* batting average  $p$ . This would be the hierarchical model for our data:

$$\begin{aligned} p &\sim N(.275, .027^2) \\ Y | p &\sim N(p, .111^2) \end{aligned}$$

We now are ready to compute a posterior distribution to summarize our prediction of  $p$ . The continuous version of Bayes' rule can be used here to derive the *posterior probability function*, which is the distribution of  $p$  assuming we observe  $Y = y$ . In our case, we can show that when we fix  $Y = y$ ,  $p$  follows a normal distribution with expected value:

$$\begin{aligned} E(p | Y = y) &= B\mu + (1 - B)y \\ &= \mu + (1 - B)(y - \mu) \\ \text{with } B &= \frac{\sigma^2}{\sigma^2 + \tau^2} \end{aligned}$$

This is a weighted average of the population average  $\mu$  and the observed data  $y$ . The weight depends on the SD of the population  $\tau$  and the SD of our observed data  $\sigma$ . This weighted average is sometimes referred to as *shrinking* because it *shrinks* estimates towards a prior mean. In the case of José Iglesias, we have:

$$\begin{aligned} E(p | Y = .450) &= B \times .275 + (1 - B) \times .450 \\ &= .275 + (1 - B)(.450 - .275) \\ B &= \frac{.111^2}{.111^2 + .027^2} = 0.944 \\ E(p | Y = .450) &\approx .285 \end{aligned}$$

We do not show the derivation here, but the standard error can be shown to be:

$$\text{SE}(p | y)^2 = \frac{1}{1/\sigma^2 + 1/\tau^2} = \frac{1}{1/.111^2 + 1/.027^2} = 0.00069$$

and the standard deviation is therefore 0.026. So we started with a frequentist 95% confidence interval that ignored data from other players and summarized just José's data:  $.450 \pm 0.220$ . Then we used a Bayesian approach that incorporated data from other players and other years to obtain a posterior probability. This is actually referred to as an empirical Bayes approach because we used data to construct the prior. From the posterior, we can report what is called a 95% *credible interval* by reporting a region, centered at the mean, with a 95% chance of occurring. In our case, this turns out to be:  $.285 \pm 0.052$ .

The Bayesian credible interval suggests that if another team is impressed by the .450 observation, we should consider trading José as we are predicting he will be just slightly above average. Interestingly, the Red Sox traded José to the Detroit Tigers in July. Here are the José Iglesias batting averages for the next five months:

| Month           | At Bat | Hits | AVG  |
|-----------------|--------|------|------|
| April           | 20     | 9    | .450 |
| May             | 26     | 11   | .423 |
| June            | 86     | 34   | .395 |
| July            | 83     | 17   | .205 |
| August          | 85     | 25   | .294 |
| September       | 50     | 10   | .200 |
| Total w/o April | 330    | 97   | .293 |

Although both intervals included the final batting average, the Bayesian credible interval provided a much more precise prediction. In particular, it predicted that he would not be as good during the remainder of the season.

---

## 16.7 Exercises

1. In 1999, in England, Sally Clark<sup>5</sup> was found guilty of the murder of two of her sons. Both infants were found dead in the morning, one in 1996 and another in 1998. In both cases, she claimed the cause of death was sudden infant death syndrome (SIDS). No evidence of physical harm was found on the two infants so the main piece of evidence against her was the testimony of Professor Sir Roy Meadow, who testified that the chances of two infants dying of SIDS was 1 in 73 million. He arrived at this figure by finding that the rate of SIDS was 1 in 8,500 and then calculating that the chance of two SIDS cases was  $8,500 \times 8,500 \approx 73$  million. Which of the following do you agree with?

- a. Sir Meadow assumed that the probability of the second son being affected by SIDS was independent of the first son being affected, thereby ignoring possible genetic

---

<sup>5</sup>[https://en.wikipedia.org/wiki/Sally\\_Clark](https://en.wikipedia.org/wiki/Sally_Clark)

- causes. If genetics plays a role then:  $\Pr(\text{second case of SIDS} \mid \text{first case of SIDS}) < \Pr(\text{first case of SIDS})$ .
- Nothing. The multiplication rule always applies in this way:  $\Pr(A \text{ and } B) = \Pr(A)\Pr(B)$
  - Sir Meadow is an expert and we should trust his calculations.
  - Numbers don't lie.
- Let's assume that there is in fact a genetic component to SIDS and the probability of  $\Pr(\text{second case of SIDS} \mid \text{first case of SIDS}) = 1/100$ , is much higher than 1 in 8,500. What is the probability of both of her sons dying of SIDS?
  - Many press reports stated that the expert claimed the probability of Sally Clark being innocent as 1 in 73 million. Perhaps the jury and judge also interpreted the testimony this way. This probability can be written as the probability of *a mother is a son-murdering psychopath* given that *two of her children are found dead with no evidence of physical harm*. According to Bayes' rule, what is this?
  - Assume that the chance of a son-murdering psychopath finding a way to kill her children, without leaving evidence of physical harm, is:

$$\Pr(A \mid B) = 0.50$$

with  $A$  = two of her children are found dead with no evidence of physical harm and  $B$  = a mother is a son-murdering psychopath = 0.50. Assume that the rate of son-murdering psychopaths mothers is 1 in 1,000,000. According to Bayes' theorem, what is the probability of  $\Pr(B \mid A)$  ?

5/. After Sally Clark was found guilty, the Royal Statistical Society issued a statement saying that there was "no statistical basis" for the expert's claim. They expressed concern at the "misuse of statistics in the courts". Eventually, Sally Clark was acquitted in June 2003. What did the expert miss?

- He made an arithmetic error.
  - He made two mistakes. First, he misused the multiplication rule and did not take into account how rare it is for a mother to murder her children. After using Bayes' rule, we found a probability closer to 0.5 than 1 in 73 million.
  - He mixed up the numerator and denominator of Bayes' rule.
  - He did not use R.
- Florida is one of the most closely watched states in the U.S. election because it has many electoral votes, and the election is generally close, and Florida tends to be a swing state that can vote either way. Create the following table with the polls taken during the last two weeks:

```
library(tidyverse)
library(dslabs)
data(polls_us_election_2016)
polls <- polls_us_election_2016 %>%
 filter(state == "Florida" & enddate >= "2016-11-04") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Take the average spread of these polls. The CLT tells us this average is approximately

normal. Calculate an average and provide an estimate of the standard error. Save your results in an object called `results`.

7. Now assume a Bayesian model that sets the prior distribution for Florida's election night spread  $d$  to be Normal with expected value  $\mu$  and standard deviation  $\tau$ . What are the interpretations of  $\mu$  and  $\tau$ ?

- a.  $\mu$  and  $\tau$  are arbitrary numbers that let us make probability statements about  $d$ .
- b.  $\mu$  and  $\tau$  summarize what we would predict for Florida before seeing any polls. Based on past elections, we would set  $\mu$  close to 0 because both Republicans and Democrats have won, and  $\tau$  at about 0.02, because these elections tend to be close.
- c.  $\mu$  and  $\tau$  summarize what we want to be true. We therefore set  $\mu$  at 0.10 and  $\tau$  at 0.01.
- d. The choice of prior has no effect on Bayesian analysis.

8. The CLT tells us that our estimate of the spread  $\hat{d}$  has normal distribution with expected value  $d$  and standard deviation  $\sigma$  calculated in problem 6. Use the formulas we showed for the posterior distribution to calculate the expected value of the posterior distribution if we set  $\mu = 0$  and  $\tau = 0.01$ .

9. Now compute the standard deviation of the posterior distribution.

10. Using the fact that the posterior distribution is normal, create an interval that has a 95% probability of occurring centered at the posterior expected value. Note that we call these credible intervals.

11. According to this analysis, what was the probability that Trump wins Florida?

12. Now use `supply` function to change the prior variance from `seq(0.05, 0.05, len = 100)` and observe how the probability changes by making a plot.

---

## 16.8 Case study: election forecasting

In a previous section, we generated these data tables:

```
library(tidyverse)
library(dslabs)
polls <- polls_us_election_2016 %>%
 filter(state == "U.S." & enddate >= "2016-10-31" &
 (grade %in% c("A+", "A", "A-", "B+") | is.na(grade))) %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)

one_poll_per_pollster <- polls %>% group_by(pollster) %>%
 filter(enddate == max(enddate)) %>%
 ungroup()

results <- one_poll_per_pollster %>%
 summarize(avg = mean(spread), se = sd(spread)/sqrt(length(spread))) %>%
 mutate(start = avg - 1.96*se, end = avg + 1.96*se)
```

Below, we will use these for our forecasting.

### 16.8.1 Bayesian approach

Pollsters tend to make probabilistic statements about the results of the election. For example, “The chance that Obama wins the electoral college is 91%” is a probabilistic statement about a parameter which in previous sections we have denoted with  $d$ . We showed that for the 2016 election, FiveThirtyEight gave Clinton an 81.4% chance of winning the popular vote. To do this, they used the Bayesian approach we described.

We assume a hierarchical model similar to what we did to predict the performance of a baseball player. Statistical textbooks will write the model like this:

$$d \sim N(\mu, \tau^2) \text{ describes our best guess had we not seen any polling data}$$

$$\bar{X} \mid d \sim N(d, \sigma^2) \text{ describes randomness due to sampling and the pollster effect}$$

For our best guess, we note that before any poll data is available, we can use data sources other than polling data. A popular approach is to use what pollsters call *fundamentals*, which are based on properties about the current economy that historically appear to have an effect in favor or against the incumbent party. We won’t use these here. Instead, we will use  $\mu = 0$ , which is interpreted as a model that simply does not provide any information on who will win. For the standard deviation, we will use recent historical data that shows the winner of the popular vote has an average spread of about 3.5%. Therefore, we set  $\tau = 0.035$ .

Now we can use the formulas for the posterior distribution for the parameter  $d$ : the probability of  $d > 0$  given the observed poll data:

```
mu <- 0
tau <- 0.035
sigma <- results$se
Y <- results$avg
B <- sigma^2 / (sigma^2 + tau^2)

posterior_mean <- B*mu + (1-B)*Y
posterior_se <- sqrt(1/ (1/sigma^2 + 1/tau^2))

posterior_mean
#> [1] 0.0281
posterior_se
#> [1] 0.00615
```

To make a probability statement, we use the fact that the posterior distribution is also normal. And we have a credible interval of:

```
posterior_mean + c(-1.96, 1.96)*posterior_se
#> [1] 0.0160 0.0401
```

The posterior probability  $\Pr(d > 0 \mid \bar{X})$  can be computed like this:

```
1 - pnorm(0, posterior_mean, posterior_se)
#> [1] 1
```

This says we are 100% sure Clinton will win the popular vote, which seems too overconfident. Also, it is not in agreement with FiveThirtyEight's 81.4%. What explains this difference?

### 16.8.2 The general bias

After elections are over, one can look at the difference between pollster predictions and actual result. An important observation that our model does not take into account is that it is common to see a general bias that affects many pollsters in the same way making the observed data correlated. There is no good explanation for this, but we do observe it in historical data: in one election, the average of polls favors Democrats by 2%, then in the following election they favor Republicans by 1%, then in the next election there is no bias, then in the following one Republicans are favored by 3%, and so on. In 2016, the polls were biased in favor of the Democrats by 1-2%.

Although we know this bias term affects our polls, we have no way of knowing what this bias is until election night. So we can't correct our polls accordingly. What we can do is include a term in our model that accounts for this variability.

### 16.8.3 Mathematical representations of models

Suppose we are collecting data from one pollster and we assume there is no general bias. The pollster collects several polls with a sample size of  $N$ , so we observe several measurements of the spread  $X_1, \dots, X_J$ . The theory tells us that these random variables have expected value  $d$  and standard error  $2\sqrt{p(1-p)/N}$ . Let's start by using the following model to describe the observed variability:

$$X_j = d + \varepsilon_j.$$

We use the index  $j$  to represent the different polls and we define  $\varepsilon_j$  to be a random variable that explains the poll-to-poll variability introduced by sampling error. To do this, we assume its average is 0 and standard error is  $2\sqrt{p(1-p)/N}$ . If  $d$  is 2.1 and the sample size for these polls is 2,000, we can simulate  $J = 6$  data points from this model like this:

```
set.seed(3)
J <- 6
N <- 2000
d <- .021
p <- (d + 1)/2
X <- d + rnorm(J, 0, 2 * sqrt(p * (1 - p) / N))
```

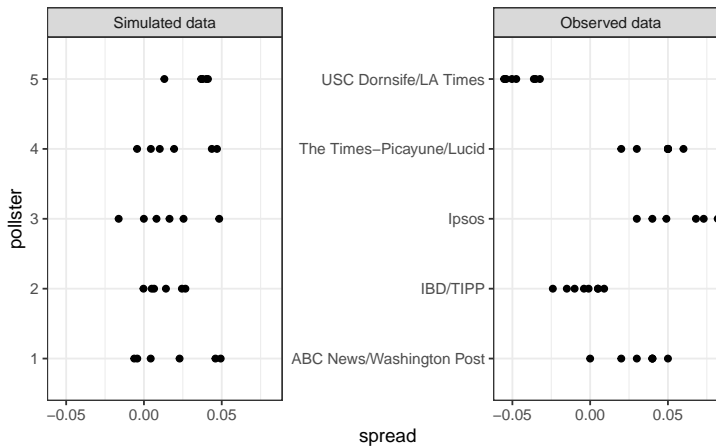
Now suppose we have  $J = 6$  data points from  $I = 5$  different pollsters. To represent this we now need two indexes, one for pollster and one for the polls each pollster takes. We use  $X_{ij}$  with  $i$  representing the pollster and  $j$  representing the  $j$ -th poll from that pollster. If we apply the same model, we write:

$$X_{i,j} = d + \varepsilon_{i,j}$$

To simulate data, we now have to loop through the pollsters:

```
I <- 5
J <- 6
N <- 2000
X <- sapply(1:I, function(i){
 d + rnorm(J, 0, 2 * sqrt(p * (1 - p) / N))
})
```

The simulated data does not really seem to capture the features of the actual data:



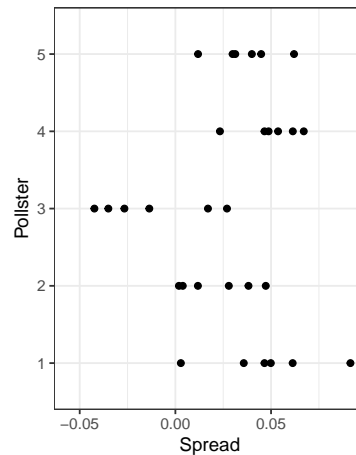
The model above does not account for pollster-to-pollster variability. To fix this, we add a new term for the pollster effect. We will use  $h_i$  to represent the house effect of the  $i$ -th pollster. The model is now augmented to:

$$X_{i,j} = d + h_i + \varepsilon_{i,j}$$

To simulate data from a specific pollster, we now need to draw an  $h_i$  and then add the  $\varepsilon$ s. Here is how we would do it for one specific pollster. We assume  $\sigma_h$  is 0.025:

```
I <- 5
J <- 6
N <- 2000
d <- .021
p <- (d + 1) / 2
h <- rnorm(I, 0, 0.025)
X <- sapply(1:I, function(i){
 d + h[i] + rnorm(J, 0, 2 * sqrt(p * (1 - p) / N))
})
```

The simulated data now looks more like the actual data:



Note that  $h_i$  is common to all the observed spreads from a specific pollster. Different pollsters have a different  $h_i$ , which explains why we can see the groups of points shift up and down from pollster to pollster.

Now, in the model above, we assume the average house effect is 0. We think that for every pollster biased in favor of our party, there is another one in favor of the other and assume the standard deviation is  $\sigma_h$ . But historically we see that every election has a general bias affecting all polls. We can observe this with the 2016 data, but if we collect historical data, we see that the average of polls misses by more than models like the one above predict. To see this, we would take the average of polls for each election year and compare it to the actual value. If we did this, we would see a difference with a standard deviation of between 2-3%. To incorporate this into the model, we can add another term to account for this variability:

$$X_{i,j} = d + b + h_i + \varepsilon_{i,j}.$$

Here  $b$  is a random variable that accounts for the election-to-election variability. This random variable changes from election to election, but for any given election, it is the same for all pollsters and polls within on election. This is why it does not have indexes. This implies that all the random variables  $X_{i,j}$  for an election year are correlated since they all have  $b$  in common.

One way to interpret  $b$  is as the difference between the average of all polls from all pollsters and the actual result of the election. Because we don't know the actual result until after the election, we can't estimate  $b$  until after the election. However, we can estimate  $b$  from previous elections and study the distribution of these values. Based on this approach we assume that, across election years,  $b$  has expected value 0 and the standard error is about  $\sigma_b = 0.025$ .

An implication of adding this term to the model is that the standard deviation for  $X_{i,j}$  is actually higher than what we earlier called  $\sigma$ , which combines the pollster variability and the sample in variability, and was estimated with:

```
sd(one_poll_per_pollster$spread)
#> [1] 0.0242
```

This estimate does not include the variability introduced by  $b$ . Note that because



$$\bar{X} = d + b + \frac{1}{N} \sum_{i=1}^N X_i,$$

the standard deviation of  $\bar{X}$  is:

$$\sqrt{\sigma^2/N + \sigma_b^2}.$$

Since the same  $b$  is in every measurement, the average does not reduce the variability introduced by the  $b$  term. This is an important point: it does not matter how many polls you take, this bias does not get reduced.

If we redo the Bayesian calculation taking this variability into account, we get a result much closer to FiveThirtyEight's:

```
mu <- 0
tau <- 0.035
sigma <- sqrt(results$se^2 + .025^2)
Y <- results$avg
B <- sigma^2 / (sigma^2 + tau^2)

posterior_mean <- B*mu + (1-B)*Y
posterior_se <- sqrt(1/ (1/sigma^2 + 1/tau^2))

1 - pnorm(0, posterior_mean, posterior_se)
#> [1] 0.817
```

#### 16.8.4 Predicting the electoral college

Up to now we have focused on the popular vote. But in the United States, elections are not decided by the popular vote but rather by what is known as the electoral college. Each state gets a number of electoral votes that depends, in a somewhat complex way, on the population size of the state. Here are the top 5 states ranked by electoral votes in 2016.

```
results_us_election_2016 %>% top_n(5, electoral_votes)
#> state electoral_votes clinton trump others
#> 1 California 55 61.7 31.6 6.7
#> 2 Texas 38 43.2 52.2 4.5
#> 3 Florida 29 47.8 49.0 3.2
#> 4 New York 29 59.0 36.5 4.5
#> 5 Illinois 20 55.8 38.8 5.4
#> 6 Pennsylvania 20 47.9 48.6 3.6
```

With some minor exceptions we don't discuss, the electoral votes are won all or nothing. For example, if you win California by just 1 vote, you still get all 55 of its electoral votes. This means that by winning a few big states by a large margin, but losing many small states by small margins, you can win the popular vote and yet lose the electoral college. This happened in 1876, 1888, 2000, and 2016. The idea behind this is to avoid a few large states having the power to dominate the presidential election. Nonetheless, many people in the US consider the electoral college unfair and would like to see it abolished.

We are now ready to predict the electoral college result for 2016. We start by aggregating results from a poll taken during the last week before the election. We use the `str_detect`, a function we introduce later in Section 24.1, to remove polls that are not for entire states.

```
results <- polls_us_election_2016 %>%
 filter(state!="U.S." &
 !str_detect(state, "CD") &
 enddate >="2016-10-31" &
 (grade %in% c("A+", "A", "A-", "B+") | is.na(grade))) %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100) %>%
 group_by(state) %>%
 summarize(avg = mean(spread), sd = sd(spread), n = n()) %>%
 mutate(state = as.character(state))
```

Here are the five closest races according to the polls:

```
results %>% arrange(abs(avg))
#> # A tibble: 47 x 4
#> state avg sd n
#> <chr> <dbl> <dbl> <int>
#> 1 Florida 0.00356 0.0163 7
#> 2 North Carolina -0.00730 0.0306 9
#> 3 Ohio -0.0104 0.0252 6
#> 4 Nevada 0.0169 0.0441 7
#> 5 Iowa -0.0197 0.0437 3
#> # ... with 42 more rows
```

We now introduce the command `left_join` that will let us easily add the number of electoral votes for each state from the dataset `us_electoral_votes_2016`. We will describe this function in detail in the Wrangling chapter. Here, we simply say that the function combines the two datasets so that the information from the second argument is added to the information in the first:

```
results <- left_join(results, results_us_election_2016, by = "state")
```

Notice that some states have no polls because the winner is pretty much known:

```
results_us_election_2016 %>% filter(!state %in% results$state) %>%
 pull(state)
#> [1] "Rhode Island" "Alaska" "Wyoming"
#> [4] "District of Columbia"
```

No polls were conducted in DC, Rhode Island, Alaska, and Wyoming because Democrats are sure to win in the first two and Republicans in the last two.

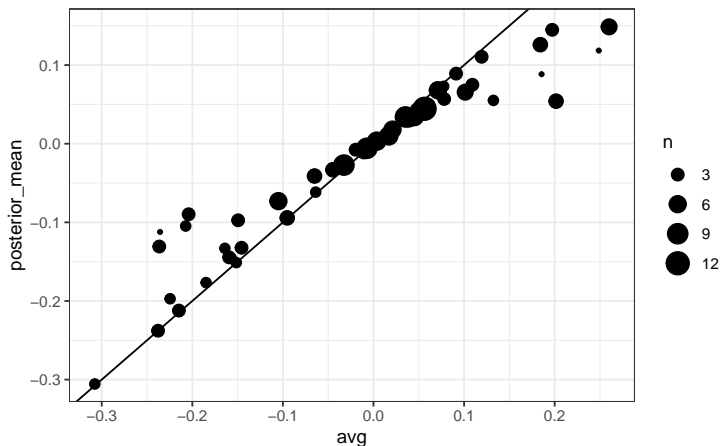
Because we can't estimate the standard deviation for states with just one poll, we will estimate it as the median of the standard deviations estimated for states with more than one poll:

```
results <- results %>%
 mutate(sd = ifelse(is.na(sd), median(results$sd, na.rm = TRUE), sd))
```

To make probabilistic arguments, we will use a Monte Carlo simulation. For each state, we apply the Bayesian approach to generate an election day  $d$ . We could construct the priors for each state based on recent history. However, to keep it simple, we assign a prior to each state that assumes we know nothing about what will happen. Since from election year to election year the results from a specific state don't change that much, we will assign a standard deviation of 2% or  $\tau = 0.02$ . For now, we will assume, incorrectly, that the poll results from each state are independent. The code for the Bayesian calculation under these assumptions looks like this:

```
#> # A tibble: 47 x 12
#> state avg sd n electoral_votes clinton trump others
#> <chr> <dbl> <dbl> <int> <int> <dbl> <dbl> <dbl>
#> 1 Alab~ -0.149 2.53e-2 3 9 34.4 62.1 3.6
#> 2 Ariz~ -0.0326 2.70e-2 9 11 45.1 48.7 6.2
#> 3 Arka~ -0.151 9.90e-4 2 6 33.7 60.6 5.8
#> 4 Cali~ 0.260 3.87e-2 5 55 61.7 31.6 6.7
#> 5 Colo~ 0.0452 2.95e-2 7 9 48.2 43.3 8.6
#> # ... with 42 more rows, and 4 more variables: sigma <dbl>, B <dbl>,
#> # posterior_mean <dbl>, posterior_se <dbl>
```

The estimates based on posterior do move the estimates towards 0, although the states with many polls are influenced less. This is expected as the more poll data we collect, the more we trust those results:



Now we repeat this 10,000 times and generate an outcome from the posterior. In each iteration, we keep track of the total number of electoral votes for Clinton. Remember that Trump gets 270 minus the votes for Clinton. Also note that the reason we add 7 in the code is to account for Rhode Island and D.C.:

```

B <- 10000
mu <- 0
tau <- 0.02
clinton_EV <- replicate(B, {
 results %>% mutate(sigma = sd/sqrt(n),
 B = sigma^2 / (sigma^2 + tau^2),
 posterior_mean = B * mu + (1 - B) * avg,
 posterior_se = sqrt(1 / (1/sigma^2 + 1/tau^2)),
 result = rnorm(length(posterior_mean),
 posterior_mean, posterior_se),
 clinton = ifelse(result > 0, electoral_votes, 0)) %>%
 summarize(clinton = sum(clinton)) %>%
 pull(clinton) + 7
})

mean(clinton_EV > 269)
#> [1] 0.998

```

This model gives Clinton over 99% chance of winning. A similar prediction was made by the Princeton Election Consortium. We now know it was quite off. What happened?

The model above ignores the general bias and assumes the results from different states are independent. After the election, we realized that the general bias in 2016 was not that big: it was between 1 and 2%. But because the election was close in several big states and these states had a large number of polls, pollsters that ignored the general bias greatly underestimated the standard error. Using the notation we introduce, they assumed the standard error was  $\sqrt{\sigma^2/N}$  which with large  $N$  is quite smaller than the more accurate estimate  $\sqrt{\sigma^2/N + \sigma_b^2}$ . FiveThirtyEight, which models the general bias in a rather sophisticated way, reported a closer result. We can simulate the results now with a bias term. For the state level, the general bias can be larger so we set it at  $\sigma_b = 0.03$ :

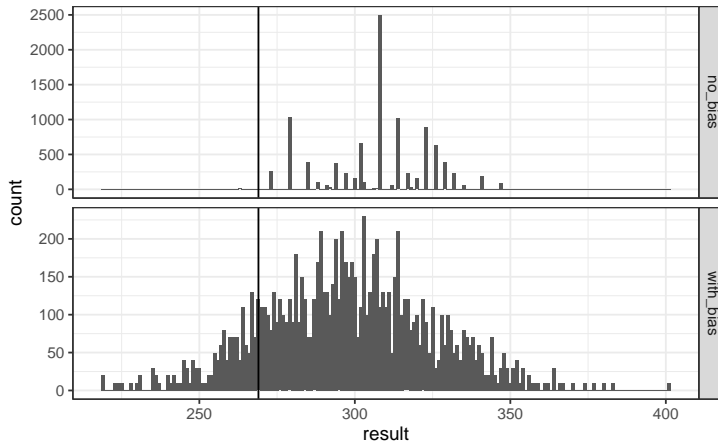
```

tau <- 0.02
bias_sd <- 0.03
clinton_EV_2 <- replicate(1000, {
 results %>% mutate(sigma = sqrt(sd^2/n + bias_sd^2),
 B = sigma^2 / (sigma^2 + tau^2),
 posterior_mean = B*mu + (1-B)*avg,
 posterior_se = sqrt(1/ (1/sigma^2 + 1/tau^2)),
 result = rnorm(length(posterior_mean),
 posterior_mean, posterior_se),
 clinton = ifelse(result>0, electoral_votes, 0)) %>%
 summarize(clinton = sum(clinton) + 7) %>%
 pull(clinton)
})

mean(clinton_EV_2 > 269)
#> [1] 0.848

```

This gives us a much more sensible estimate. Looking at the outcomes of the simulation, we see how the bias term adds variability to the final results.



FiveThirtyEight includes many other features we do not include here. One is that they model variability with distributions that have high probabilities for extreme events compared to the normal. One way we could do this is by changing the distribution used in the simulation from a normal distribution to a t-distribution. FiveThirtyEight predicted a probability of 71%.

### 16.8.5 Forecasting

Forecasters like to make predictions well before the election. The predictions are adapted as new polls come out. However, an important question forecasters must ask is: how informative are polls taken several weeks before the election about the actual election? Here we study the variability of poll results across time.

To make sure the variability we observe is not due to pollster effects, let's study data from one pollster:

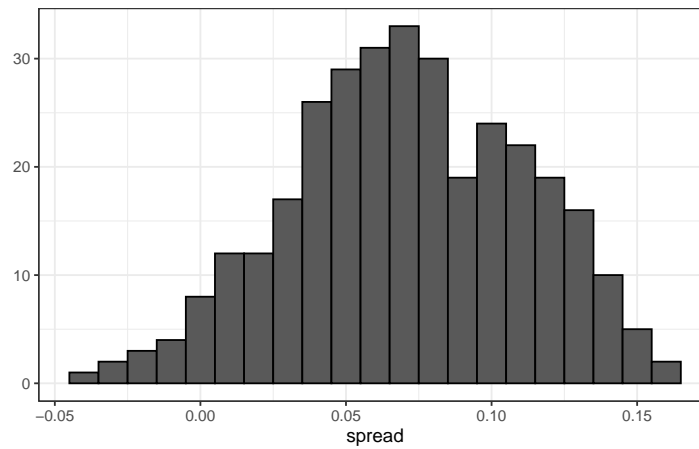
```
one_pollster <- polls_us_election_2016 %>%
 filter(pollster == "Ipsos" & state == "U.S.") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Since there is no pollster effect, then perhaps the theoretical standard error matches the data-derived standard deviation. We compute both here:

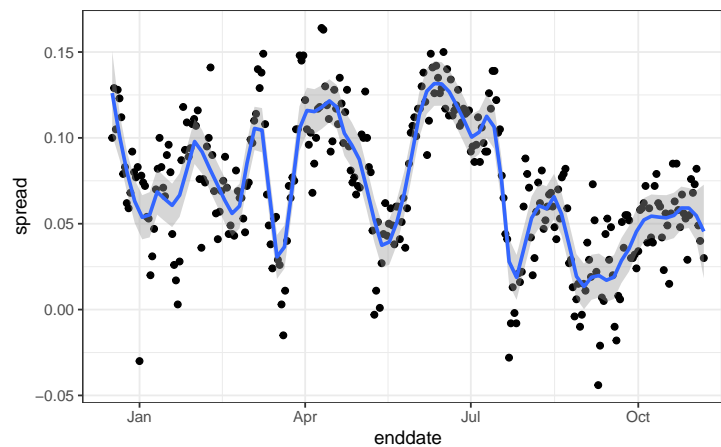
```
se <- one_pollster %>%
 summarize(empirical = sd(spread),
 theoretical = 2 * sqrt(mean(spread) * (1 - mean(spread)) /
 min(samplesize)))

se
#> empirical theoretical
#> 1 0.0403 0.0326
```

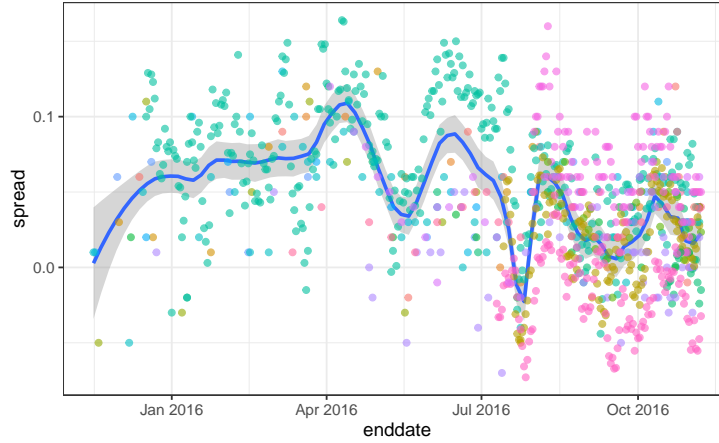
But the empirical standard deviation is higher than the highest possible theoretical estimate. Furthermore, the spread data does not look normal as the theory would predict:



The models we have described include pollster-to-pollster variability and sampling error. But this plot is for one pollster and the variability we see is certainly not explained by sampling error. Where is the extra variability coming from? The following plots make a strong case that it comes from time fluctuations not accounted for by the theory that assumes  $p$  is fixed:



Some of the peaks and valleys we see coincide with events such as the party conventions, which tend to give the candidate a boost. We can see the peaks and valleys are consistent across several pollsters:



This implies that, if we are going to forecast, our model must include a term to accounts for the time effect. We need to write a model including a bias term for time:

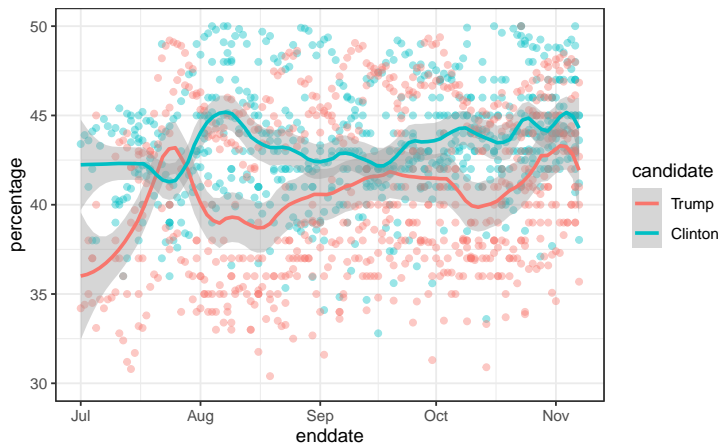
$$Y_{i,j,t} = d + b + h_j + b_t + \varepsilon_{i,j,t}$$

The standard deviation of  $b_t$  would depend on  $t$  since the closer we get to election day, the closer to 0 this bias term should be.

Pollsters also try to estimate trends from these data and incorporate these into their predictions. We can model the time trend with a function  $f(t)$  and rewrite the model like this: The blue lines in the plots above:

$$Y_{i,j,t} = d + b + h_j + b_t + f(t) + \varepsilon_{i,j,t},$$

We usually see the estimated  $f(t)$  not for the difference, but for the actual percentages for each candidate like this:



Once a model like the one above is selected, we can use historical and present data to estimate all the necessary parameters to make predictions. There is a variety of methods for estimating trends  $f(t)$  which we discuss in the Machine Learning part.

## 16.9 Exercises

1. Create this table:

```
library(tidyverse)
library(dslabs)
data("polls_us_election_2016")
polls <- polls_us_election_2016 %>%
 filter(state != "U.S." & enddate >= "2016-10-31") %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100)
```

Now for each poll use the CLT to create a 95% confidence interval for the spread reported by each poll. Call the resulting object `cis` with columns `lower` and `upper` for the limits of the confidence intervals. Use the `select` function to keep the columns `state`, `startdate`, `end date`, `pollster`, `grade`, `spread`, `lower`, `upper`.

2. You can add the final result to the `cis` table you just created using the `right_join` function like this:

```
add <- results_us_election_2016 %>%
 mutate(actual_spread = clinton/100 - trump/100) %>%
 select(state, actual_spread)
cis <- cis %>%
 mutate(state = as.character(state)) %>%
 left_join(add, by = "state")
```

Now determine how often the 95% confidence interval includes the actual result.

3. Repeat this, but show the proportion of hits for each pollster. Show only pollsters with more than 5 polls and order them from best to worst. Show the number of polls conducted by each pollster and the FiveThirtyEight grade of each pollster. Hint: use `n=n()`, `grade = grade[1]` in the call to `summarize`.
4. Repeat exercise 3, but instead of `pollster`, stratify by `state`. Note that here we can't show grades.
5. Make a barplot based on the result of exercise 4. Use `coord_flip`.
6. Add two columns to the `cis` table by computing, for each poll, the difference between the predicted spread and the actual spread, and define a column `hit` that is true if the signs are the same. Hint: use the function `sign`. Call the object `resids`.
7. Create a plot like in exercise 5, but for the proportion of times the sign of the spread agreed.
8. In exercise 7, we see that for most states the polls had it right 100% of the time. For only 9 states did the polls miss more than 25% of the time. In particular, notice that in Wisconsin every single poll got it wrong. In Pennsylvania and Michigan more than 90% of the polls had the signs wrong. Make a histogram of the errors. What is the median of these errors?
9. We see that at the state level, the median error was 3% in favor of Clinton. The distribution



is not centered at 0, but at 0.03. This is the general bias we described in the section above. Create a boxplot to see if the bias was general to all states or it affected some states differently. Use `filter(grade %in% c("A+", "A", "A-", "B+") | is.na(grade))` to only include pollsters with high grades.

10. Some of these states only have a few polls. Repeat exercise 9, but only include states with 5 good polls or more. Hint: use `group_by`, `filter` then `ungroup`. You will see that the West (Washington, New Mexico, California) underestimated Hillary's performance, while the Midwest (Michigan, Pennsylvania, Wisconsin, Ohio, Missouri) overestimated it. In our simulation, we did not model this behavior since we added general bias, rather than a regional bias. Note that some pollsters may now be modeling correlation between similar states and estimating this correlation from historical data. To learn more about this, you can learn about random effects and mixed models.

---

## 16.10 The t-distribution

Above we made use of the CLT with a sample size of 15. Because we are estimating a second parameters  $\sigma$ , further variability is introduced into our confidence interval which results in intervals that are too small. For very large sample sizes this extra variability is negligible, but, in general, for values smaller than 30 we need to be cautious about using the CLT.

However, if the data in the urn is known to follow a normal distribution, then we actually have mathematical theory that tells us how much bigger we need to make the intervals to account for the estimation of  $\sigma$ . Using this theory, we can construct confidence intervals for any  $N$ . But again, this works only if **the data in the urn is known to follow a normal distribution**. So for the 0, 1 data of our previous urn model, this theory definitely does not apply.

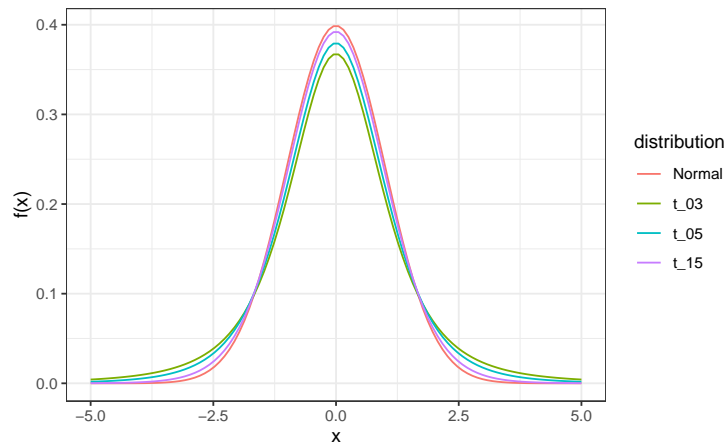
The statistic on which confidence intervals for  $d$  are based is

$$Z = \frac{\bar{X} - d}{\sigma/\sqrt{N}}$$

CLT tells us that  $Z$  is approximately normally distributed with expected value 0 and standard error 1. But in practice we don't know  $\sigma$  so we use:

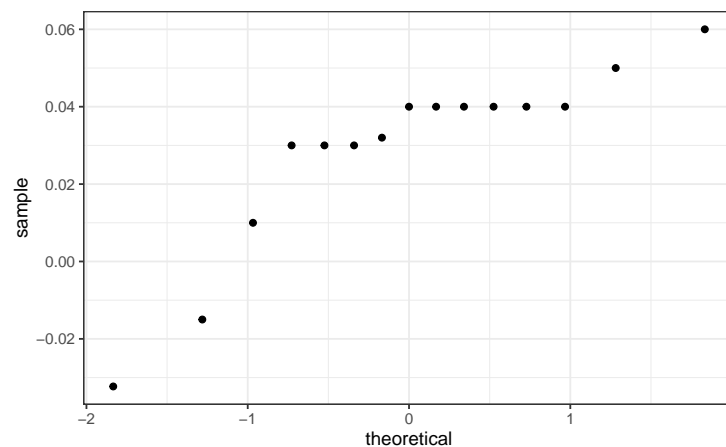
$$Z = \frac{\bar{X} - d}{s/\sqrt{N}}$$

By substituting  $\sigma$  with  $s$  we introduce some variability. The theory tells us that  $Z$  follows a t-distribution with  $N - 1$  *degrees of freedom*. The degrees of freedom is a parameter that controls the variability via fatter tails:



If we are willing to assume the pollster effect data is normally distributed, based on the sample data  $X_1, \dots, X_N$ ,

```
one_poll_per_pollster %>%
 ggplot(aes(sample=spread)) + stat_qq()
```



then  $Z$  follows a *t*-distribution with  $N - 1$  degrees of freedom. So perhaps a better confidence interval for  $d$  is:

```
z <- qt(0.975, nrow(one_poll_per_pollster)-1)
one_poll_per_pollster %>%
 summarize(avg = mean(spread), moe = z*sd(spread)/sqrt(length(spread))) %>%
 mutate(start = avg - moe, end = avg + moe)
#> # A tibble: 1 x 4
#> avg moe start end
#> <dbl> <dbl> <dbl> <dbl>
#> 1 0.0290 0.0134 0.0156 0.0424
```

A bit larger than the one using normal is

```
qt(0.975, 14)
#> [1] 2.14
```

is bigger than

```
qnorm(0.975)
#> [1] 1.96
```

The t-distribution can also be used to model errors in bigger deviations that are more likely than with the normal distribution, as seen in the densities we previously saw. Fivethirtyeight uses the t-distribution to generate errors that better model the deviations we see in election data. For example, in Wisconsin the average of six polls was 7% in favor of Clinton with a standard deviation of 1%, but Trump won by 0.7%. Even after taking into account the overall bias, this 7.7% residual is more in line with t-distributed data than the normal distribution.

```
data("polls_us_election_2016")
polls_us_election_2016 %>%
 filter(state == "Wisconsin" &
 enddate >="2016-10-31" &
 (grade %in% c("A+", "A", "A-", "B+") | is.na(grade))) %>%
 mutate(spread = rawpoll_clinton/100 - rawpoll_trump/100) %>%
 mutate(state = as.character(state)) %>%
 left_join(results_us_election_2016, by = "state") %>%
 mutate(actual = clinton/100 - trump/100) %>%
 summarize(actual = first(actual), avg = mean(spread),
 sd = sd(spread), n = n()) %>%
 select(actual, avg, sd, n)
#> actual avg sd n
#> 1 -0.007 0.0711 0.0104 6
```

# 17

## *Regression*

---

Up to this point, this book has focused mainly on single variables. However, in data science applications, it is very common to be interested in the relationship between two or more variables. For instance, in Chapter 18 we will use a data-driven approach that examines the relationship between player statistics and success to guide the building of a baseball team with a limited budget. Before delving into this more complex example, we introduce necessary concepts needed to understand regression using a simpler illustration. We actually use the dataset from which regression was born.

The example is from genetics. Francis Galton<sup>1</sup> studied the variation and heredity of human traits. Among many other traits, Galton collected and studied height data from families to try to understand heredity. While doing this, he developed the concepts of correlation and regression, as well as a connection to pairs of data that follow a normal distribution. Of course, at the time this data was collected our knowledge of genetics was quite limited compared to what we know today. A very specific question Galton tried to answer was: how well can we predict a child's height based on the parents' height? The technique he developed to answer this question, regression, can also be applied to our baseball question. Regression can be applied in many other circumstances as well.

**Historical note:** Galton made important contributions to statistics and genetics, but he was also one of the first proponents of eugenics, a scientifically flawed philosophical movement favored by many biologists of Galton's time but with horrific historical consequences. You can read more about it here: <https://pged.org/history-eugenics-and-genetics/>.

---

### 17.1 Case study: is height hereditary?

We have access to Galton's family height data through the **HistData** package. This data contains heights on several dozen families: mothers, fathers, daughters, and sons. To imitate Galton's analysis, we will create a dataset with the heights of fathers and a randomly selected son of each family:

```
library(tidyverse)
library(HistData)
data("GaltonFamilies")

set.seed(1983)
galton_heights <- GaltonFamilies %>%
 filter(gender == "male") %>%
```

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Francis\\_Galton](https://en.wikipedia.org/wiki/Francis_Galton)

```
group_by(family) %>%
 sample_n(1) %>%
 ungroup() %>%
 select(father, childHeight) %>%
 rename(son = childHeight)
```

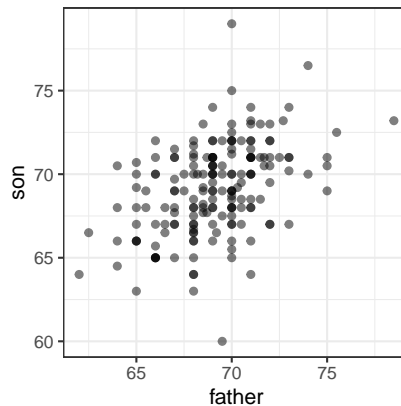
In the exercises, we will look at other relationships including mothers and daughters.

Suppose we were asked to summarize the father and son data. Since both distributions are well approximated by the normal distribution, we could use the two averages and two standard deviations as summaries:

```
galton_heights %>%
 summarize(mean(father), sd(father), mean(son), sd(son))
#> # A tibble: 1 x 4
#> `mean(father)` `sd(father)` `mean(son)` `sd(son)`
#> <dbl> <dbl> <dbl> <dbl>
#> 1 69.1 2.55 69.2 2.71
```

However, this summary fails to describe an important characteristic of the data: the trend that the taller the father, the taller the son.

```
galton_heights %>% ggplot(aes(father, son)) +
 geom_point(alpha = 0.5)
```



We will learn that the correlation coefficient is an informative summary of how two variables move together and then see how this can be used to predict one variable using the other.

---

## 17.2 The correlation coefficient

The correlation coefficient is defined for a list of pairs  $(x_1, y_1), \dots, (x_n, y_n)$  as the average of the product of the standardized values:

$$\rho = \frac{1}{n} \sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right) \left( \frac{y_i - \mu_y}{\sigma_y} \right)$$

with  $\mu_x, \mu_y$  the averages of  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$ , respectively, and  $\sigma_x, \sigma_y$  the standard deviations. The Greek letter  $\rho$  is commonly used in statistics books to denote the correlation. The Greek letter for  $r$ ,  $\rho$ , because it is the first letter of regression. Soon we learn about the connection between correlation and regression. We can represent the formula above with R code using:

```
rho <- mean(scale(x) * scale(y))
```

To understand why this equation does in fact summarize how two variables move together, consider the  $i$ -th entry of  $x$  is  $\left( \frac{x_i - \mu_x}{\sigma_x} \right)$  SDs away from the average. Similarly, the  $y_i$  that is paired with  $x_i$ , is  $\left( \frac{y_i - \mu_y}{\sigma_y} \right)$  SDs away from the average  $y$ . If  $x$  and  $y$  are unrelated, the product  $\left( \frac{x_i - \mu_x}{\sigma_x} \right) \left( \frac{y_i - \mu_y}{\sigma_y} \right)$  will be positive (  $+\times+$  and  $-\times-$  ) as often as negative (  $+\times-$  and  $-\times+$  ) and will average out to about 0. This correlation is the average and therefore unrelated variables will have 0 correlation. If instead the quantities vary together, then we are averaging mostly positive products (  $+\times+$  and  $-\times-$  ) and we get a positive correlation. If they vary in opposite directions, we get a negative correlation.

The correlation coefficient is always between -1 and 1. We can show this mathematically: consider that we can't have higher correlation than when we compare a list to itself (perfect correlation) and in this case the correlation is:

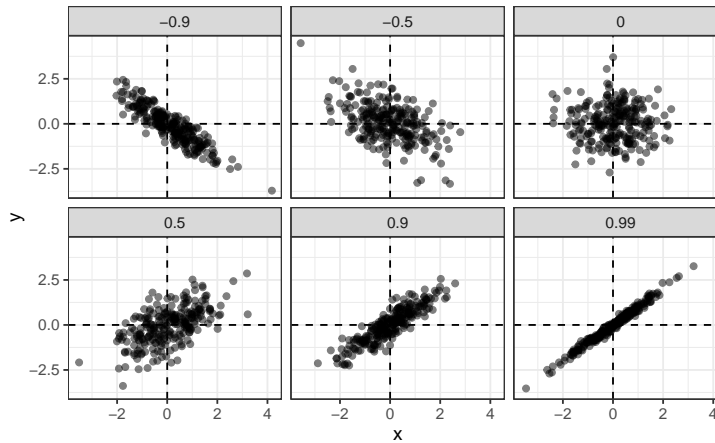
$$\rho = \frac{1}{n} \sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right)^2 = \frac{1}{\sigma_x^2} \frac{1}{n} \sum_{i=1}^n (x_i - \mu_x)^2 = \frac{1}{\sigma_x^2} \sigma_x^2 = 1$$

A similar derivation, but with  $x$  and its exact opposite, proves the correlation has to be bigger or equal to -1.

For other pairs, the correlation is in between -1 and 1. The correlation between father and son's heights is about 0.5:

```
galton_heights %>% summarize(r = cor(father, son)) %>% pull(r)
#> [1] 0.433
```

To see what data looks like for different values of  $\rho$ , here are six examples of pairs with correlations ranging from -0.9 to 0.99:



### 17.2.1 Sample correlation is a random variable

Before we continue connecting correlation to regression, let's remind ourselves about random variability.

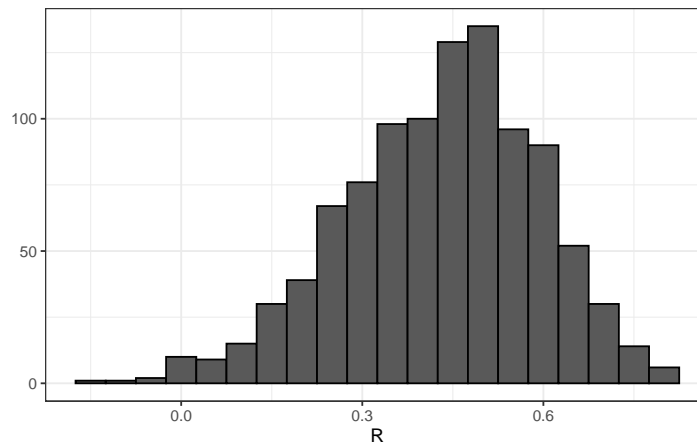
In most data science applications, we observe data that includes random variation. For example, in many cases, we do not observe data for the entire population of interest but rather for a random sample. As with the average and standard deviation, the *sample correlation* is the most commonly used estimate of the population correlation. This implies that the correlation we compute and use as a summary is a random variable.

By way of illustration, let's assume that the 179 pairs of fathers and sons is our entire population. A less fortunate geneticist can only afford measurements from a random sample of 25 pairs. The sample correlation can be computed with:

```
R <- sample_n(galton_heights, 25, replace = TRUE) %>%
 summarize(r = cor(father, son)) %>% pull(r)
```

R is a random variable. We can run a Monte Carlo simulation to see its distribution:

```
B <- 1000
N <- 25
R <- replicate(B, {
 sample_n(galton_heights, N, replace = TRUE) %>%
 summarize(r=cor(father, son)) %>%
 pull(r)
})
qplot(R, geom = "histogram", binwidth = 0.05, color = I("black"))
```



We see that the expected value of  $R$  is the population correlation:

```
mean(R)
#> [1] 0.431
```

and that it has a relatively high standard error relative to the range of values  $R$  can take:

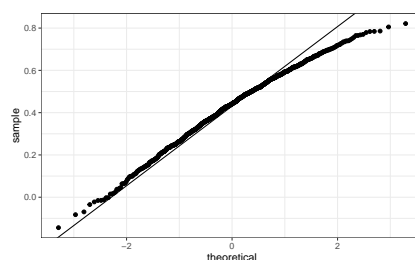
```
sd(R)
#> [1] 0.161
```

So, when interpreting correlations, remember that correlations derived from samples are estimates containing uncertainty.

Also, note that because the sample correlation is an average of independent draws, the central limit actually applies. Therefore, for large enough  $N$ , the distribution of  $R$  is approximately normal with expected value  $\rho$ . The standard deviation, which is somewhat complex to derive, is  $\sqrt{\frac{1-\rho^2}{N-2}}$ .

In our example,  $N = 25$  does not seem to be large enough to make the approximation a good one:

```
ggplot(aes(sample=R), data = data.frame(R)) +
 stat_qq() +
 geom_abline(intercept = mean(R), slope = sqrt((1-mean(R)^2)/(N-2)))
```

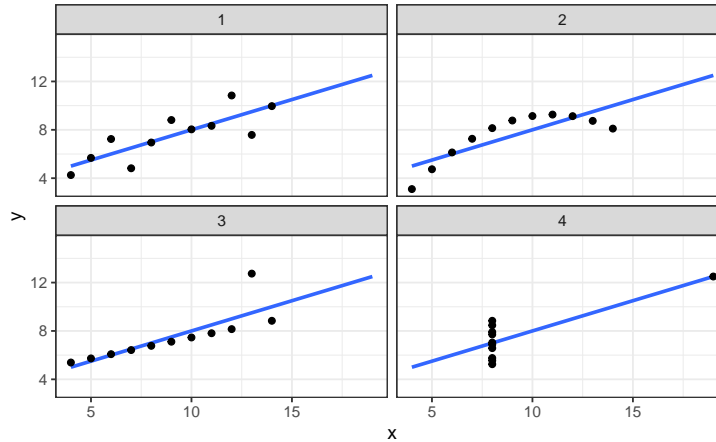


If you increase  $N$ , you will see the distribution converging to normal.



### 17.2.2 Correlation is not always a useful summary

Correlation is not always a good summary of the relationship between two variables. The following four artificial datasets, referred to as Anscombe's quartet, famously illustrate this point. All these pairs have a correlation of 0.82:



Correlation is only meaningful in a particular context. To help us understand when it is that correlation is meaningful as a summary statistic, we will return to the example of predicting a son's height using his father's height. This will help motivate and define linear regression. We start by demonstrating how correlation can be useful for prediction.

## 17.3 Conditional expectations

Suppose we are asked to guess the height of a randomly selected son and we don't know his father's height. Because the distribution of sons' heights is approximately normal, we know the average height, 69.2, is the value with the highest proportion and would be the prediction with the highest chance of minimizing the error. But what if we are told that the father is taller than average, say 72 inches tall, do we still guess 69.2 for the son?

It turns out that if we were able to collect data from a very large number of fathers that are 72 inches, the distribution of their sons' heights would be normally distributed. This implies that the average of the distribution computed on this subset would be our best prediction.

In general, we call this approach *conditioning*. The general idea is that we stratify a population into groups and compute summaries in each group. Conditioning is therefore related to the concept of stratification described in Section 8.13. To provide a mathematical description of conditioning, consider we have a population of pairs of values  $(x_1, y_1), \dots, (x_n, y_n)$ , for example all father and son heights in England. In the previous chapter we learned that if you take a random pair  $(X, Y)$ , the expected value and best predictor of  $Y$  is  $E(Y) = \mu_y$ , the population average  $1/n \sum_{i=1}^n y_i$ . However, we are no longer interested in the general population, instead we are interested in only the subset of a population with a specific  $x_i$  value, 72 inches in our example. This subset of the population, is also a population and

thus the same principles and properties we have learned apply. The  $y_i$  in the subpopulation have a distribution, referred to as the *conditional distribution*, and this distribution has an expected value referred to as the *conditional expectation*. In our example, the conditional expectation is the average height of all sons in England with fathers that are 72 inches. The statistical notation for the conditional expectation is

$$E(Y \mid X = x)$$

with  $x$  representing the fixed value that defines that subset, for example 72 inches. Similarly, we denote the standard deviation of the strata with

$$SD(Y \mid X = x) = \sqrt{\text{Var}(Y \mid X = x)}$$

Because the conditional expectation  $E(Y \mid X = x)$  is the best predictor for the random variable  $Y$  for an individual in the strata defined by  $X = x$ , many data science challenges reduce to estimating this quantity. The conditional standard deviation quantifies the precision of the prediction.

In the example we have been considering, we are interested in computing the average son height *conditioned* on the father being 72 inches tall. We want to estimate  $E(Y \mid X = 72)$  using the sample collected by Galton. We previously learned that the sample average is the preferred approach to estimating the population average. However, a challenge when using this approach to estimating conditional expectations is that for continuous data we don't have many data points matching exactly one value in our sample. For example, we have only:

```
sum(galton_heights$father == 72)
#> [1] 8
```

fathers that are exactly 72-inches. If we change the number to 72.5, we get even fewer data points:

```
sum(galton_heights$father == 72.5)
#> [1] 1
```

A practical way to improve these estimates of the conditional expectations, is to define strata of with similar values of  $x$ . In our example, we can round father heights to the nearest inch and assume that they are all 72 inches. If we do this, we end up with the following prediction for the son of a father that is 72 inches tall:

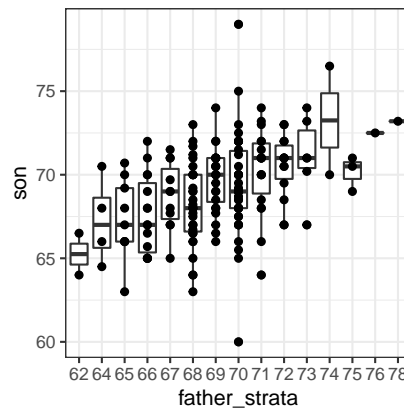
```
conditional_avg <- galton_heights %>%
 filter(round(father) == 72) %>%
 summarize(avg = mean(son)) %>%
 pull(avg)
conditional_avg
#> [1] 70.5
```

Note that a 72-inch father is taller than average – specifically,  $72 - 69.1/2.5 = 1.1$  standard deviations taller than the average father. Our prediction 70.5 is also taller than average, but

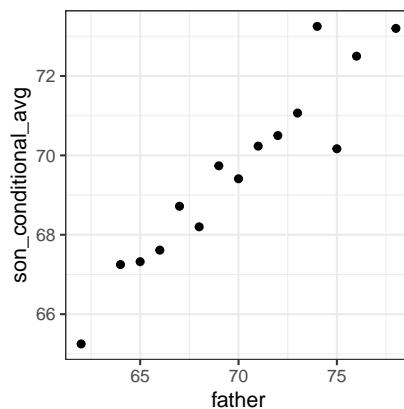
only 0.49 standard deviations larger than the average son. The sons of 72-inch fathers have *regressed* some to the average height. We notice that the reduction in how many SDs taller is about 0.5, which happens to be the correlation. As we will see in a later section, this is not a coincidence.

If we want to make a prediction of any height, not just 72, we could apply the same approach to each strata. Stratification followed by boxplots lets us see the distribution of each group:

```
galton_heights %>% mutate(father_strata = factor(round(father))) %>%
 ggplot(aes(father_strata, son)) +
 geom_boxplot() +
 geom_point()
```



Not surprisingly, the centers of the groups are increasing with height. Furthermore, these centers appear to follow a linear relationship. Below we plot the averages of each group. If we take into account that these averages are random variables with standard errors, the data is consistent with these points following a straight line:



The fact that these conditional averages follow a line is not a coincidence. In the next section, we explain that the line these averages follow is what we call the *regression line*, which improves the precision of our estimates. However, it is not always appropriate to estimate conditional expectations with the regression line so we also describe Galton's theoretical justification for using the regression line.

## 17.4 The regression line

If we are predicting a random variable  $Y$  knowing the value of another  $X = x$  using a regression line, then we predict that for every standard deviation,  $\sigma_X$ , that  $x$  increases above the average  $\mu_X$ ,  $Y$  increase  $\rho$  standard deviations  $\sigma_Y$  above the average  $\mu_Y$  with  $\rho$  the correlation between  $X$  and  $Y$ . The formula for the regression is therefore:

$$\left( \frac{Y - \mu_Y}{\sigma_Y} \right) = \rho \left( \frac{x - \mu_X}{\sigma_X} \right)$$

We can rewrite it like this:

$$Y = \mu_Y + \rho \left( \frac{x - \mu_X}{\sigma_X} \right) \sigma_Y$$

If there is perfect correlation, the regression line predicts an increase that is the same number of SDs. If there is 0 correlation, then we don't use  $x$  at all for the prediction and simply predict the average  $\mu_Y$ . For values between 0 and 1, the prediction is somewhere in between. If the correlation is negative, we predict a reduction instead of an increase.

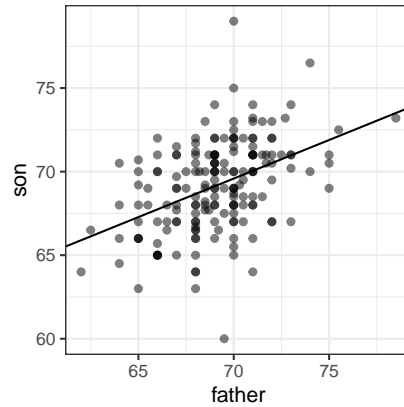
Note that if the correlation is positive and lower than 1, our prediction is closer, in standard units, to the average height than the value used to predict,  $x$ , is to the average of the  $x$ s. This is why we call it *regression*: the son regresses to the average height. In fact, the title of Galton's paper was: *Regression toward mediocrity in hereditary stature*. To add regression lines to plots, we will need the above formula in the form:

$$y = b + mx \text{ with slope } m = \rho \frac{\sigma_y}{\sigma_x} \text{ and intercept } b = \mu_y - m\mu_x$$

Here we add the regression line to the original data:

```
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)

galton_heights %>%
 ggplot(aes(father, son)) +
 geom_point(alpha = 0.5) +
 geom_abline(slope = r * s_y/s_x, intercept = mu_y - r * s_y/s_x * mu_x)
```



The regression formula implies that if we first standardize the variables, that is subtract the average and divide by the standard deviation, then the regression line has intercept 0 and slope equal to the correlation  $\rho$ . You can make same plot, but using standard units like this:

```
galton_heights %>%
 ggplot(aes(scale(father), scale(son))) +
 geom_point(alpha = 0.5) +
 geom_abline(intercept = 0, slope = r)
```

#### 17.4.1 Regression improves precision

Let's compare the two approaches to prediction that we have presented:

1. Round fathers' heights to closest inch, stratify, and then take the average.
2. Compute the regression line and use it to predict.

We use a Monte Carlo simulation sampling  $N = 50$  families:

```
B <- 1000
N <- 50

set.seed(1983)
conditional_avg <- replicate(B, {
 dat <- sample_n(galton_heights, N)
 dat %>% filter(round(father) == 72) %>%
 summarize(avg = mean(son)) %>%
 pull(avg)
})

regression_prediction <- replicate(B, {
 dat <- sample_n(galton_heights, N)
 mu_x <- mean(dat$father)
 mu_y <- mean(dat$son)
 s_x <- sd(dat$father)
 s_y <- sd(dat$son)
```

```
r <- cor(dat$father, dat$son)
mu_y + r*(72 - mu_x)/s_x*s_y
})
```

Although the expected value of these two random variables is about the same:

```
mean(conditional_avg, na.rm = TRUE)
#> [1] 70.5
mean(regression_prediction)
#> [1] 70.5
```

The standard error for the regression prediction is substantially smaller:

```
sd(conditional_avg, na.rm = TRUE)
#> [1] 0.964
sd(regression_prediction)
#> [1] 0.452
```

The regression line is therefore much more stable than the conditional mean. There is an intuitive reason for this. The conditional average is computed on a relatively small subset: the fathers that are about 72 inches tall. In fact, in some of the permutations we have no data, which is why we use `na.rm=TRUE`. The regression always uses all the data.

So why not always use the regression for prediction? Because it is not always appropriate. For example, Anscombe provided cases for which the data does not have a linear relationship. So are we justified in using the regression line to predict? Galton answered this in the positive for height data. The justification, which we include in the next section, is somewhat more advanced than the rest of the chapter.

### 17.4.2 Bivariate normal distribution (advanced)

Correlation and the regression slope are a widely used summary statistic, but they are often misused or misinterpreted. Anscombe's examples provide over-simplified cases of dataset in which summarizing with correlation would be a mistake. But there are many more real-life examples.

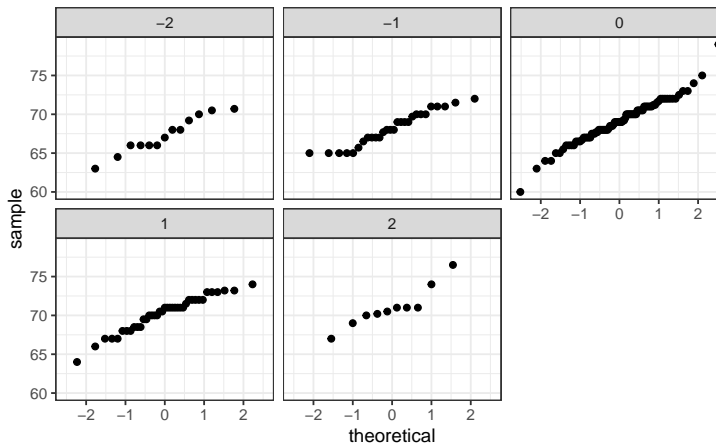
The main way we motivate the use of correlation involves what is called the *bivariate normal distribution*.

When a pair of random variables is approximated by the bivariate normal distribution, scatterplots look like ovals. As we saw in Section 17.2, they can be thin (high correlation) or circle-shaped (no correlation).

A more technical way to define the bivariate normal distribution is the following: if  $X$  is a normally distributed random variable,  $Y$  is also a normally distributed random variable, and the conditional distribution of  $Y$  for any  $X = x$  is approximately normal, then the pair is approximately bivariate normal.

If we think the height data is well approximated by the bivariate normal distribution, then we should see the normal approximation hold for each strata. Here we stratify the son heights by the standardized father heights and see that the assumption appears to hold:

```
galton_heights %>%
 mutate(z_father = round((father - mean(father)) / sd(father))) %>%
 filter(z_father %in% -2:2) %>%
 ggplot() +
 stat_qq(aes(sample = son)) +
 facet_wrap(~ z_father)
```



Now we come back to defining correlation. Galton used mathematical statistics to demonstrate that, when two variables follow a bivariate normal distribution, computing the regression line is equivalent to computing conditional expectations. We don't show the derivation here, but we can show that under this assumption, for any given value of  $x$ , the expected value of the  $Y$  in pairs for which  $X = x$  is:

$$E(Y|X = x) = \mu_Y + \rho \frac{X - \mu_X}{\sigma_X} \sigma_Y$$

This is the regression line, with slope

$$\rho \frac{\sigma_Y}{\sigma_X}$$

and intercept  $\mu_Y - \rho \mu_X$ . It is equivalent to the regression equation we showed earlier which can be written like this:

$$\frac{E(Y | X = x) - \mu_Y}{\sigma_Y} = \rho \frac{x - \mu_X}{\sigma_X}$$

This implies that, if our data is approximately bivariate, the regression line gives the conditional probability. Therefore, we can obtain a much more stable estimate of the conditional expectation by finding the regression line and using it to predict.

In summary, if our data is approximately bivariate, then the conditional expectation, the best prediction of  $Y$  given we know the value of  $X$ , is given by the regression line.

### 17.4.3 Variance explained

The bivariate normal theory also tells us that the standard deviation of the *conditional* distribution described above is:

$$\text{SD}(Y \mid X = x) = \sigma_Y \sqrt{1 - \rho^2}$$

To see why this is intuitive, notice that without conditioning,  $\text{SD}(Y) = \sigma_Y$ , we are looking at the variability of all the sons. But once we condition, we are only looking at the variability of the sons with a tall, 72-inch, father. This group will all tend to be somewhat tall so the standard deviation is reduced.

Specifically, it is reduced to  $\sqrt{1 - \rho^2} = \sqrt{1 - 0.25} = 0.87$  of what it was originally. We could say that father heights “explain” 13% of the variability observed in son heights.

The statement “ $X$  explains such and such percent of the variability” is commonly used in academic papers. In this case, this percent actually refers to the variance (the SD squared). So if the data is bivariate normal, the variance is reduced by  $1 - \rho^2$ , so we say that  $X$  explains  $1 - (1 - \rho^2) = \rho^2$  (the correlation squared) of the variance.

But it is important to remember that the “variance explained” statement only makes sense when the data is approximated by a bivariate normal distribution.

### 17.4.4 Warning: there are two regression lines

We computed a regression line to predict the son’s height from father’s height. We used these calculations:

```
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)
m_1 <- r * s_y / s_x
b_1 <- mu_y - m_1*mu_x
```

which gives us the function  $E(Y \mid X = x) = 37.3 + 0.46 x$ .

What if we want to predict the father’s height based on the son’s? It is important to know that this is not determined by computing the inverse function:  $x = \{E(Y \mid X = x) - 37.3\} / 0.5$ .

We need to compute  $E(X \mid Y = y)$ . Since the data is approximately bivariate normal, the theory described above tells us that this conditional expectation will follow a line with slope and intercept:

```
m_2 <- r * s_x / s_y
b_2 <- mu_x - m_2 * mu_y
```

So we get  $E(X \mid Y = y) = 40.9 + 0.41y$ . Again we see regression to the average: the