

Introduction to Data Science - Introduction	3
Introduction to Data Science	5
Introduction to Data Science - Summary statistics	7
Introduction to Data Science - 1 Distributions	8
Introduction to Data Science - 2 Robust summaries	30
Introduction to Data Science - Probability	35
Introduction to Data Science - 3 Discrete probability	36
Introduction to Data Science - 4 Continuous probability	50
Introduction to Data Science - 5 Random variables	57
Introduction to Data Science - Statistical inference	68
Introduction to Data Science - 6 Parameters and Estimates	69
Introduction to Data Science - 7 Central Limit Theorem	76
Introduction to Data Science - 8 Confidence intervals	81
Introduction to Data Science - 9 Hypothesis testing	86
Introduction to Data Science - 10 Bootstrap	89
Introduction to Data Science - 11 Data-driven models	92
Introduction to Data Science - 12 Bayesian statistics	102
Introduction to Data Science - 13 Hierarchical Models	107
Introduction to Data Science - Linear Models	121
Introduction to Data Science - 14 Regression	122
Introduction to Data Science - 15 Multivariate Regression	146
Introduction to Data Science - 16 Measurement error models	167
Introduction to Data Science - 17 Treatment effect models	171
Introduction to Data Science - 18 Association tests	183
Introduction to Data Science - 19 Association is not causation	190
Introduction to Data Science - High dimensional data	202
Introduction to Data Science - 20 Matrices in R	203
Introduction to Data Science - 21 Applied Linear Algebra	216
Introduction to Data Science - 22 Dimension reduction	223
Introduction to Data Science - 23 Regularization	244

Introduction to Data Science - 24 Matrix Factorization	256
Introduction to Data Science - Machine Learning	268
Introduction to Data Science - 25 Notation and terminology	269
Introduction to Data Science - 26 Evaluation metrics	271
Introduction to Data Science - 27 Conditional probabilities and expectation- s	282
Introduction to Data Science - 28 Smoothing	285
Introduction to Data Science - 29 Resampling methods	304
Introduction to Data Science - 30 Examples of algorithms	318
Introduction to Data Science - 31 Machine learning in practice	346
Introduction to Data Science - 32 Clustering	361

Introduction to Data Science - Introduction

Rafael A. Irizarry

The phrase *data science* began gaining significant popularity around 2012, thanks in part to the publication titled “*Data Scientist: The Most Alluring Profession of the 21st Century*”¹. This aligns with the rise of a new kind of endeavor in the technology sector and some academic projects during the 2000s: the extraction of insights from messy, complex, and large datasets, which had become increasingly prevalent, all made possible with the advent of digital storage of data.

Some examples include using data from various political pollsters to improve election predictions, extracting information from athletic department websites to evaluate baseball prospects, analyzing movie ratings from all streaming service users to make personalized recommendations, developing software to read zip codes by digitizing written digits, and using advanced measurement technologies to understand the molecular causes of diseases. This book is centered around these, and other practical examples.

Achieving success in these instances involves a collaborative effort by a team of experts with different but complementary skills. In this book, our primary focus is on data analysis. To grasp the best ways to analyze data effectively in the mentioned examples, we will cover key mathematical concepts. Some of these concepts are not new and were originally developed for different purposes, but they have proven to be adaptable and useful in various contexts.

Over the past several decades, data analysts have developed ideas, concepts, and methodologies applicable across a broad range of projects. They’ve also identified common ways to get fooled by apparent patterns in the data and important mathematical realities that are not immediately obvious. This collective wisdom has evolved into the field of Statistics, a discipline offering a mathematical framework to simplify the articulation and rigorous assessment of these concepts. For a data analyst, it’s crucial to have a comprehensive understanding of this field to prevent repeated errors and unnecessary reinvention of methodologies.

There is no shortage of exceptional Statistics textbooks detailing this mathematical framework. In this book, we emphasize bridging theory and practice, applying these concepts to actual real-world challenges using data examples and in-depth case studies. We provide representative case studies that mirror what a practicing data analyst experiences. In each case study, we present and break down the R code applied to solve the problem. We also use R code to elucidate key statistical concepts often discussed in a mathematical context.

The book is divided into six sections: **Summary Statistics, Probability, Statistical Inference, Linear Models, High Dimensional Data and Machine Learning**. Although the first two parts use data examples to illustrate concepts, the real-world case studies don’t appear until the third part. Each part comprises several chapters, each roughly designed for a single lecture and including a variety of exercises. All data referenced in the book is included in the `dslabs` package with all the Quarto code used to generate the book available on GitHub².

Who will find this book useful?

This book is meant to be a textbook for a second course in Data Science with a focus on data analysis. Previous knowledge of R, such as that covered in Introduction to Data Science, is necessary. If you read and understand all the chapters and complete all the exercises in this book, you will be well-positioned to perform advanced data analysis tasks and you will be prepared to learn the more advanced concepts and skills needed to become an expert.

What is not covered by this book?

This book focuses on the application of statistical and machine learning methods in data analysis. We do not go in depth into the theoretical aspects of the methods, and highly recommend complementing this book with probability and statistics textbooks. We also 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.

1. <https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century>
2. <https://github.com/rafalab/dsbook-part-2>

Introduction to Data Science

Rafael A. Irizarry

Statistics and Prediction Algorithms Through Case Studies

Preface

This is the website for the **Statistics and Prediction Algorithms Through Case Studies** part of **Introduction to Data Science**.

The website for the **Data Wrangling and Visualization with R** is here.

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

A hardcopy version of the first edition of the book, which combined both parts, is available from CRC Press².

A free PDF of the October 24, 2019 version of the book, which combined both parts, is available from Leanpub³.

The Quarto code used to generate the book is available on GitHub⁴. 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.

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

Acknowledgments

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 Héctor Corrada-Bravo, for advice on how to best teach machine learning. Thanks to Alyssa Frazee for helping create the homework problem that became the Recommendation Systems case study. Also, many thanks to Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund for making the Quarto 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 Zofia 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* or sending an email: `nickyfoto` (Huang Qiang), `desautm` (Marc-André Désautels), `michaschwab` (Michail Schwab), `alvarolarreategui` (Alvaro Larreategui), `jakevc` (Jake VanCampen), `omerta` (Guillermo Lengemann), `espinelli` (Enrico Spinielli), `asimumba` (Aaron Simumba), `braunschweig` (Maledewar), `gwierzchowski` (Grzegorz Wierzchowski), `technocrat` (Richard Careaga), `atzakas`, `defeit` (David Emerson Feit), `shiraamitchell` (Shira Mitchell), `Nathalie-S`, `andreashandel` (Andreas Handel), `berkowitz` (Elias Berkowitz), `Dean-Webb` (Dean Webber), `mohayusuf`, `jimrothstein`, `mPloenzke` (Matthew Ploenzke), `NicholasDowand` (Nicholas Dow), `kant` (Dario

Hereñú), **debbieyuster** (Debbie Yuster), **tuanchauict** (Tuan Chau), **phzeller**, **BTJ01** (BradJ), **glsnow** (Greg Snow), **mberlanda** (Mauro Berlanda), **wfan9**, **larswestvang** (Lars Westvang), **jj999** (Jan Andrejkovic), **Kriegslustig** (Luca Nils Schmid), **odahhani**, **aidanhorn** (Aidan Horn), **atraxler** (Adrienne Traxler), **alvegorova**, **wycheong** (Won Young Cheong), **med-hat** (Medhat Khalil), **biscotty666** (Brian Carey), **kengustafson**, **Yowza63**, **ryan-heslin** (Ryan Heslin), **raffaem**, **tim8west**, David D. Kane, El Mustapha El Abbassi, Vadim Zipunnikov, Anna Quaglieri, Chris Dong, Rick Schoenberg, Isabella Grabski, and Doug Snyder.

1. <https://www.edx.org/professional-certificate/harvardx-data-science>
2. https://www.routledge.com/Introduction-to-Data-Science-Data-Analysis-and-Prediction-Algorithms-with/Irizarry/p/book/9780367357986?utm_source=author&utm_medium=shared_link&utm_campaign=B043135_jm1_5ll_6rm_t081_1al_introductiontodatascienceauthorshare
3. <https://leanpub.com/datasciencebook>
4. <https://github.com/rafalab/dsbook-part-2>

Introduction to Data Science - Summary statistics

Rafael A. Irizarry

We start by describing a simple yet powerful data analysis technique: constructing data summaries. Although the approach does not require mathematical models or probability, the motivation for the summaries we describe will later help us understand both these topics.

You have likely 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, with 50 the standard deviation. The report has summarized the entirety 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? In this section, we answer these questions and motivate several useful summary statistics and plots, including the average, standard deviation, median, quartiles, histograms, and density plots.

Introduction to Data Science - 1 Distributions

Rafael A. Irizarry

To illustrate the concepts needed to understand distribution and how they relate to summary statistics, we will 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 included in the `dslabs` package.

```
library(dslabs)
```

One way to convey the heights to ET is to simply send him this list of 1050 heights. However, 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 1.10.

It turns out that, in some cases, the average and the standard deviation are 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.

1.1 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 US regions (Northeast, South, North Central, West). Some categorical data can be ordered even if they are not numbers, such as spiciness (mild, medium, hot). In statistics textbooks, these 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 number of gun murders per year, are discrete because they must 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. However, there are indeed examples that can be considered both numerical and ordinal.

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. Here is an example with US state regions:

```
prop.table(table(state.region))
#> state.region
#>      Northeast          South       North Central           West
#>      0.18              0.32        0.24            0.26
```

When the data is numerical, the task of constructing a summary based on the distribution is more challenging. We introduce an artificial, yet illustrative, motivating problem that will help us introduce the concepts needed to understand distributions.

1.2 Empirical 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, as 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 entries $\{x\}$ that are below a , for all possible values of a . This function is called the empirical cumulative distribution function (eCDF) and often denoted with $F(a)$:

$$F(a) = \text{Proportion of data points that are less than or equal to } a$$

Here is a plot of $F(a)$ for the male height data:

```
library(tidyverse)
heights |> filter(sex == "Male") |>
  ggplot(aes(height)) +
  stat_ecdf() +
  labs(x = "a", y = "F(a)")
```

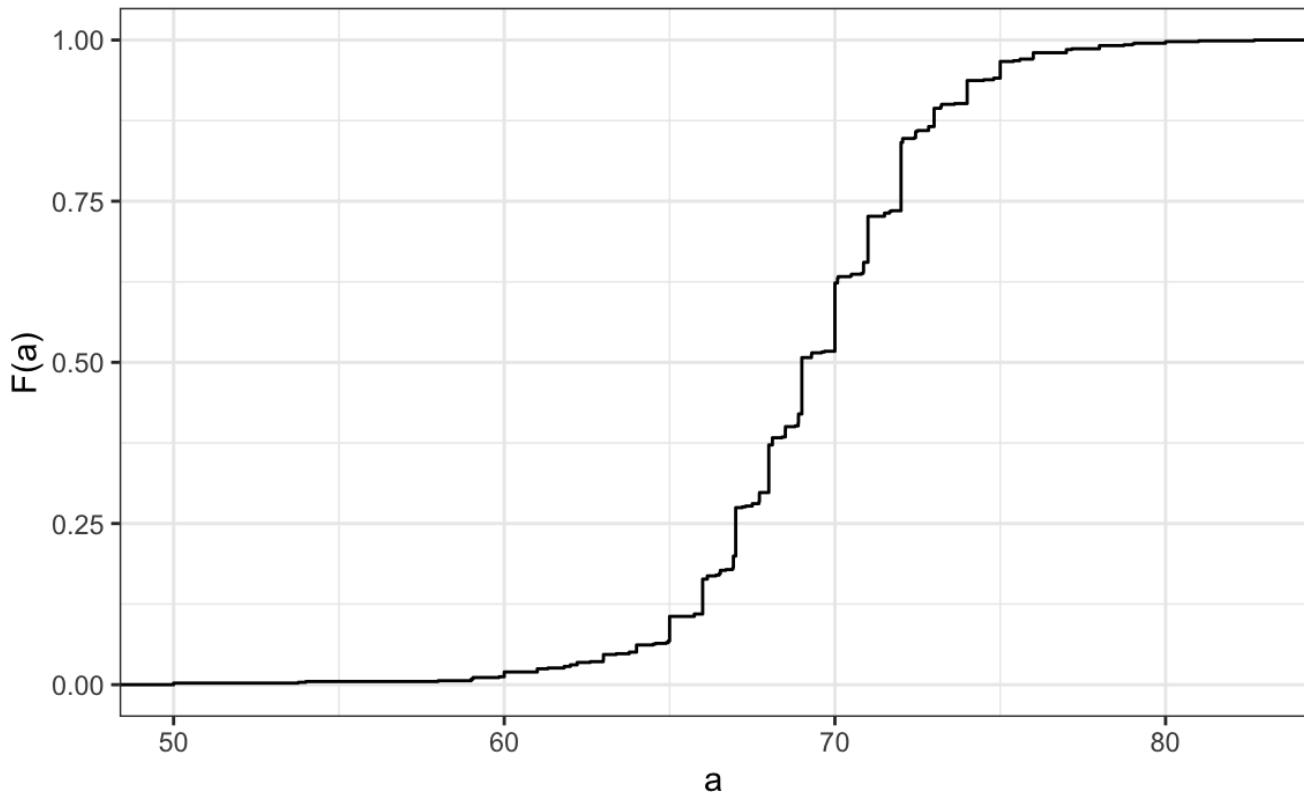


Figure 1:

Similar to what the frequency table does for categorical data, the eCDF defines the distribution for numerical data. From the plot, we can see that 16% of the values are below 65, since $F(66)=0.1637931$, or that 84% of the values are below 72, since $F(72)=0.841133$, 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.

Note: the reason we add the word *empirical* is because, as we will see in Section 4.1), the cumulative distribution function (CDF) can be defined mathematically, meaning without any data.

1.3 Histograms

Although the eCDF concept is widely discussed in statistics textbooks, the summary 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, whether the distribution symmetric, or which ranges contain 95% of the values. Histograms, on the other hand, are much preferred because they greatly facilitate answering such questions. Histograms sacrifice just a bit of information to produce summaries 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: $\{(49.5, 50.5], (50.5, 51.5], (51.5, 52.5], (52.5, 53.5], \dots, (82.5, 83.5]\}$

```
heights |> filter(sex == "Male") |>
  ggplot(aes(height)) +
  geom_histogram(binwidth = 1, color = "black")
```

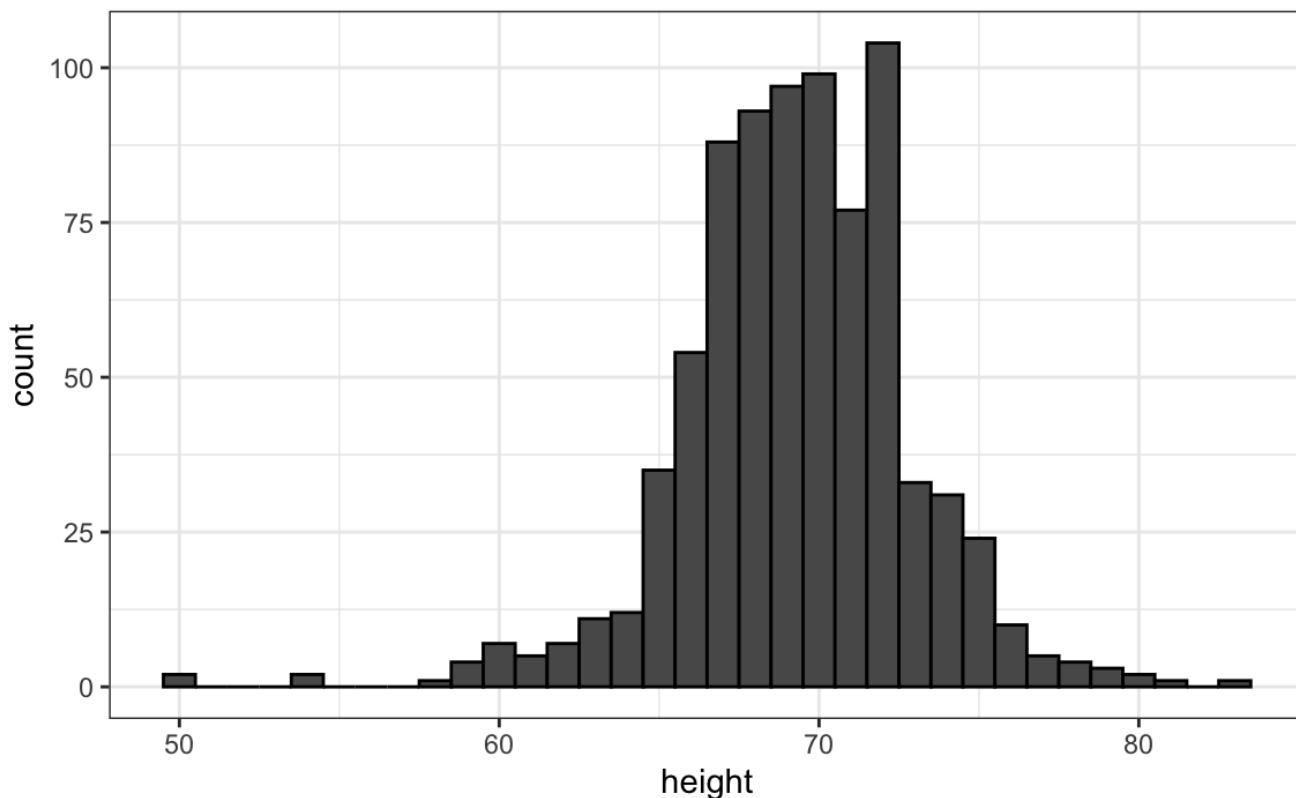


Figure 2:

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? Notice 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.

1.4 Smoothed density

Smooth density plots are similar to histograms, but the data is not divided into bins. Here is what a smooth density plot looks like for our heights data:

```
heights |>
  filter(sex == "Male") |>
  ggplot(aes(height)) +
  geom_density(alpha = 0.2, fill = "#00BFC4")
```

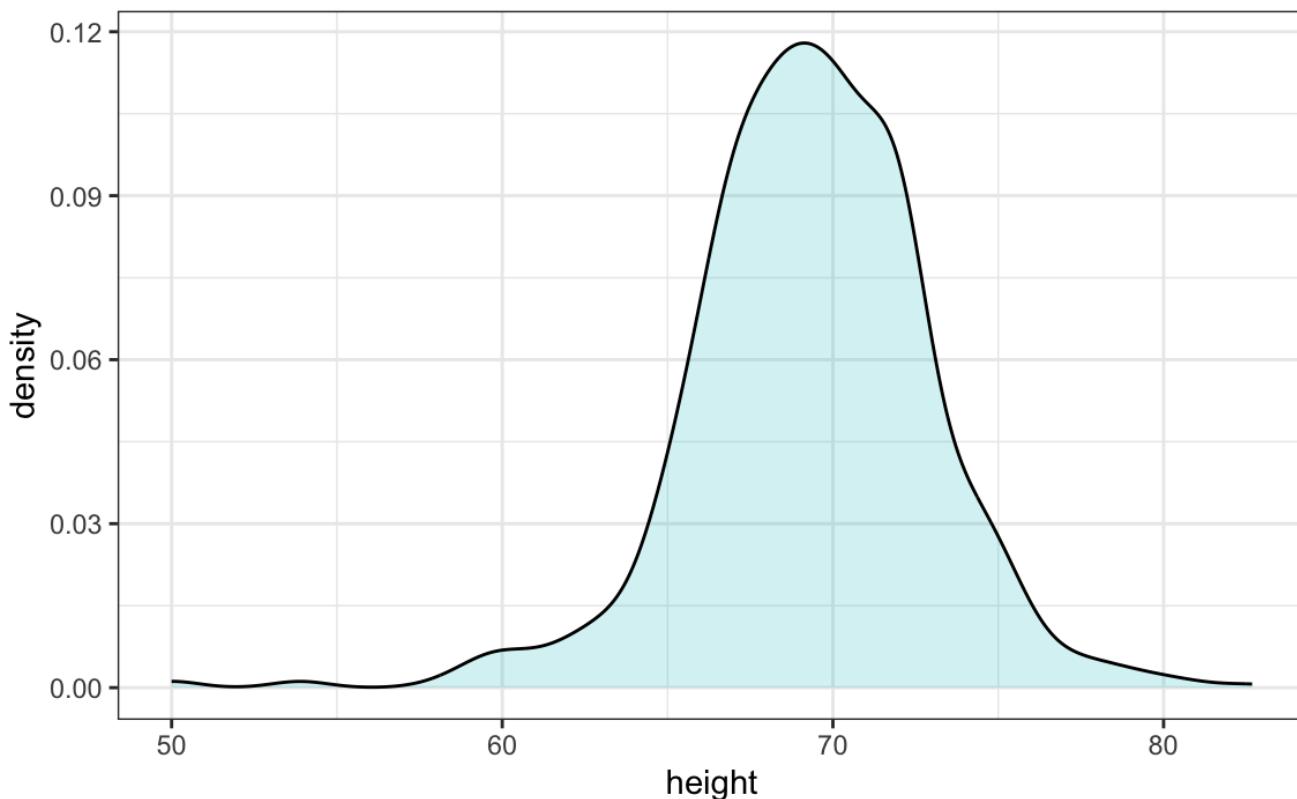


Figure 3:

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.

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 other list of values, and what we truly want to report to ET is this larger distribution, as 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 present a hypothetical histogram with bins of size 1:

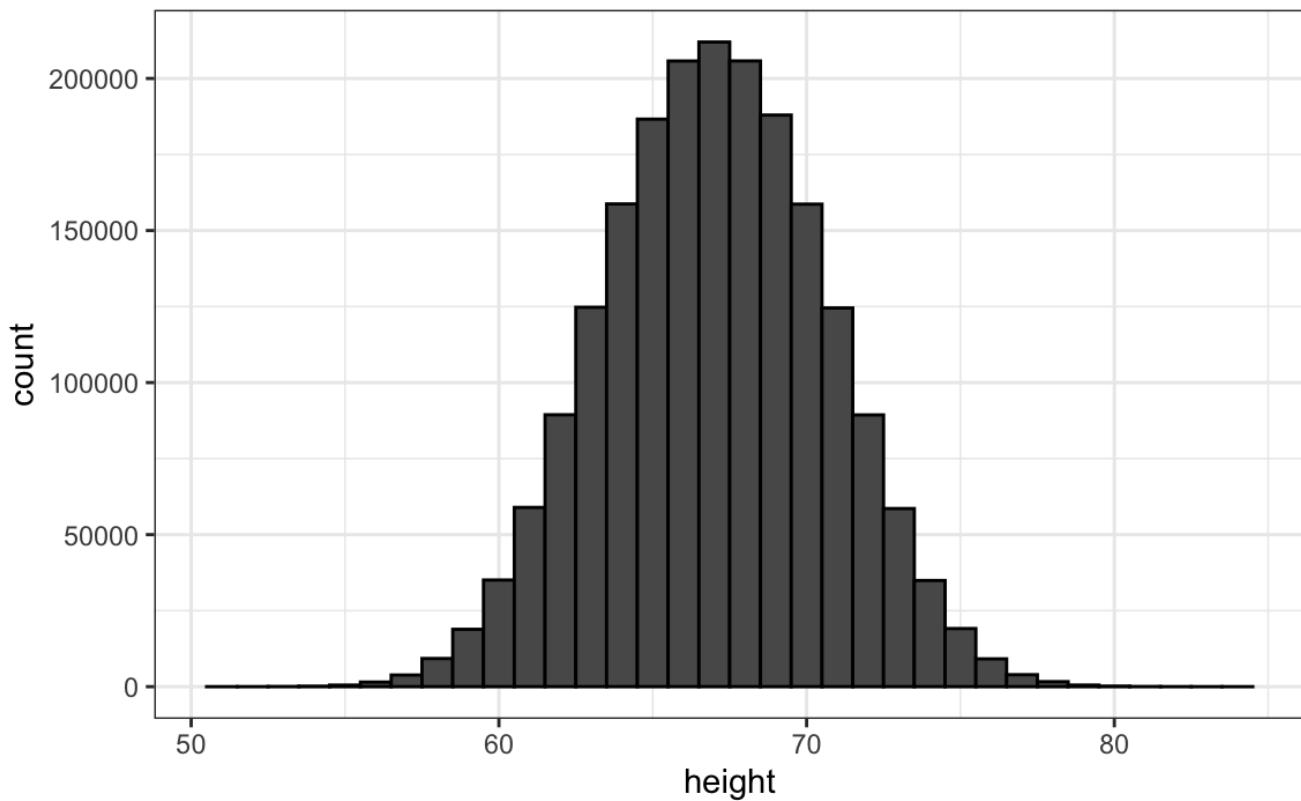


Figure 4:

The smaller we make the bins, the smoother the histogram becomes. Below 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.

Therefore, we make a histogram using bin sizes appropriate for our data, computing frequencies rather than counts. Additionally, we draw a smooth curve that passes 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:

```
p <- heights |> filter(sex == "Male") |>
  ggplot(aes(height)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 1, alpha = 0.5)

p1 <- p + geom_line(stat = 'density', adjust = 0.5)
p2 <- p + geom_line(stat = 'density', adjust = 2)
```

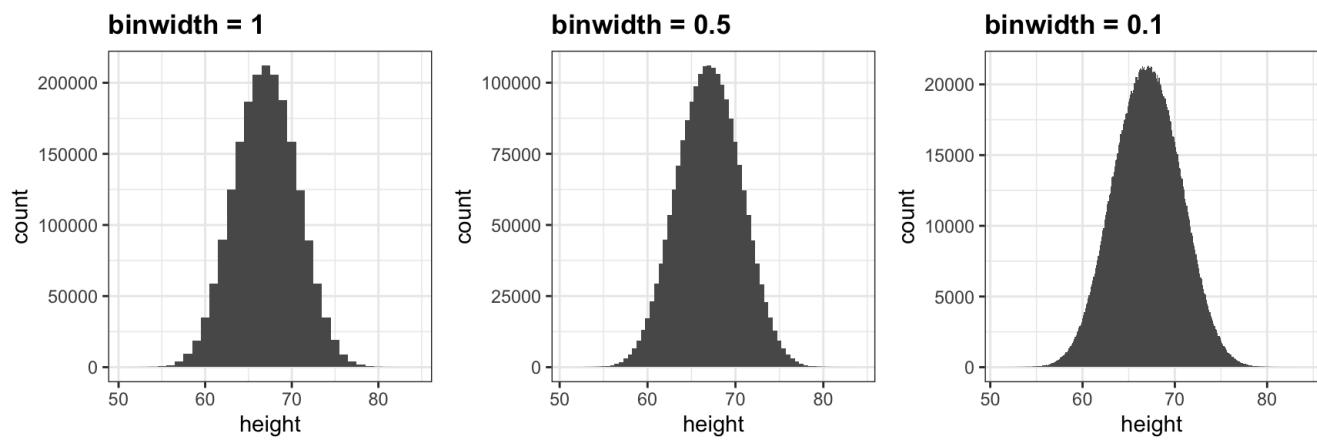


Figure 5:

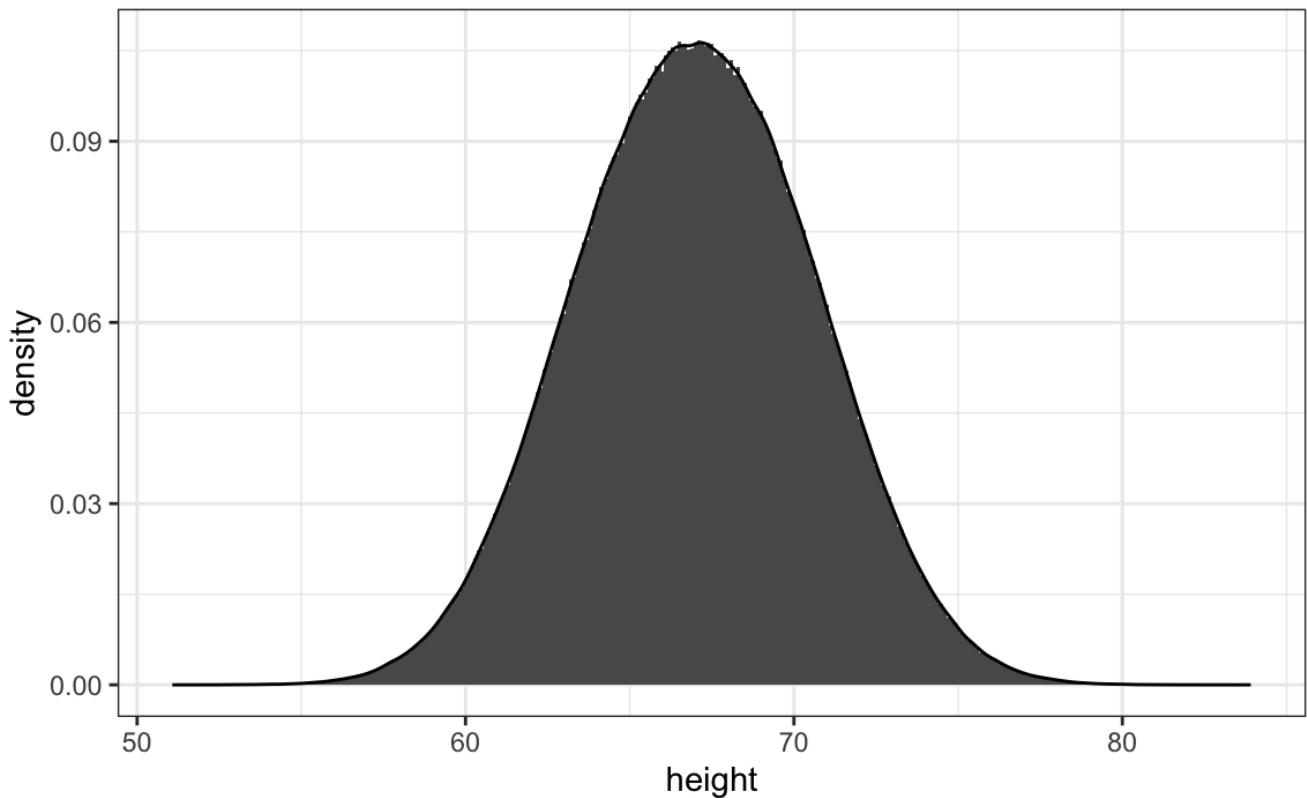


Figure 6:

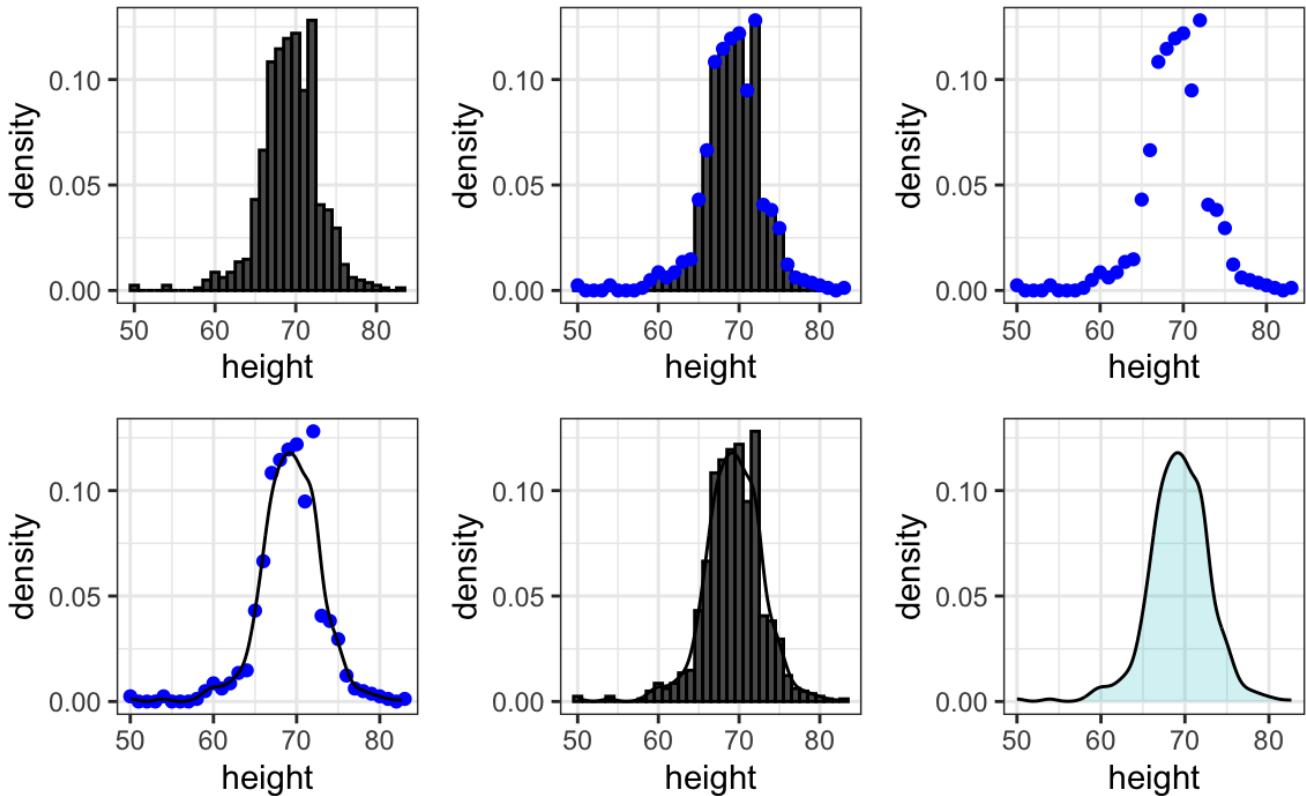


Figure 7:

```
library(gridExtra)
#>
#> Attaching package: 'gridExtra'
#> The following object is masked from 'package:dplyr':
#>
#>   combine
grid.arrange(p1,p2, ncol = 2)
```

We need to make this choice with care as the resulting summary 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 relatively smooth, resembling the example on the right more than the one on the left.

While the histogram is an assumption-free summary, the smoothed density is based on some assumptions.

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 that 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 30% of male heights are 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:

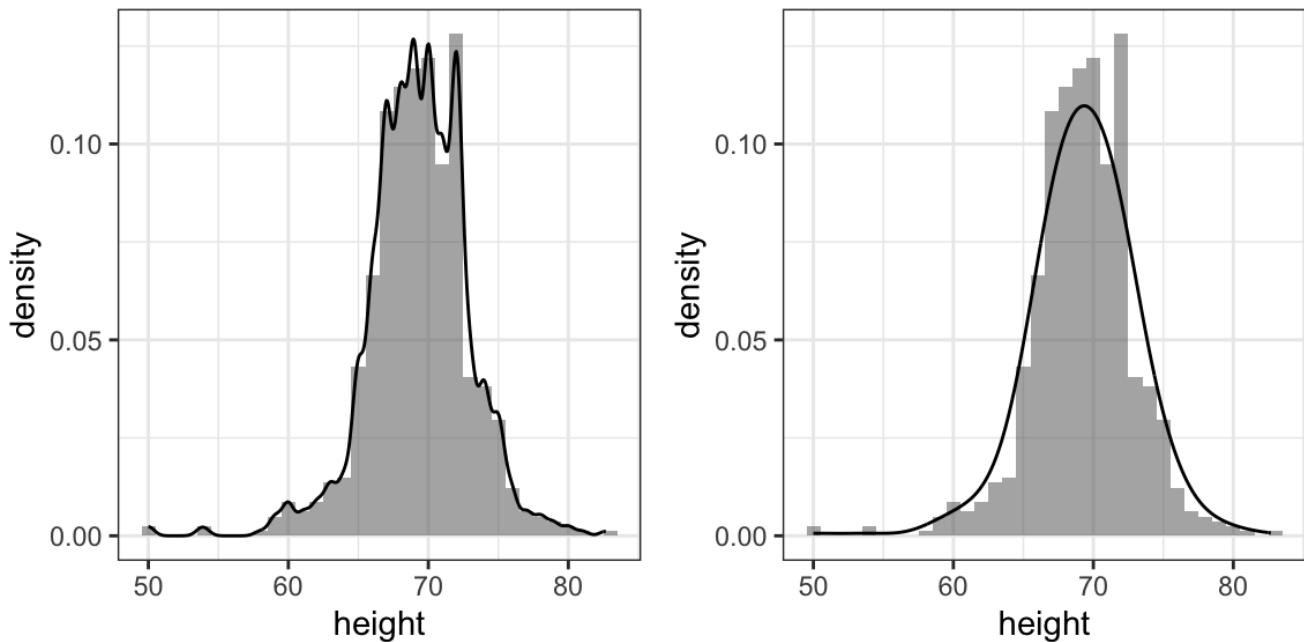


Figure 8:

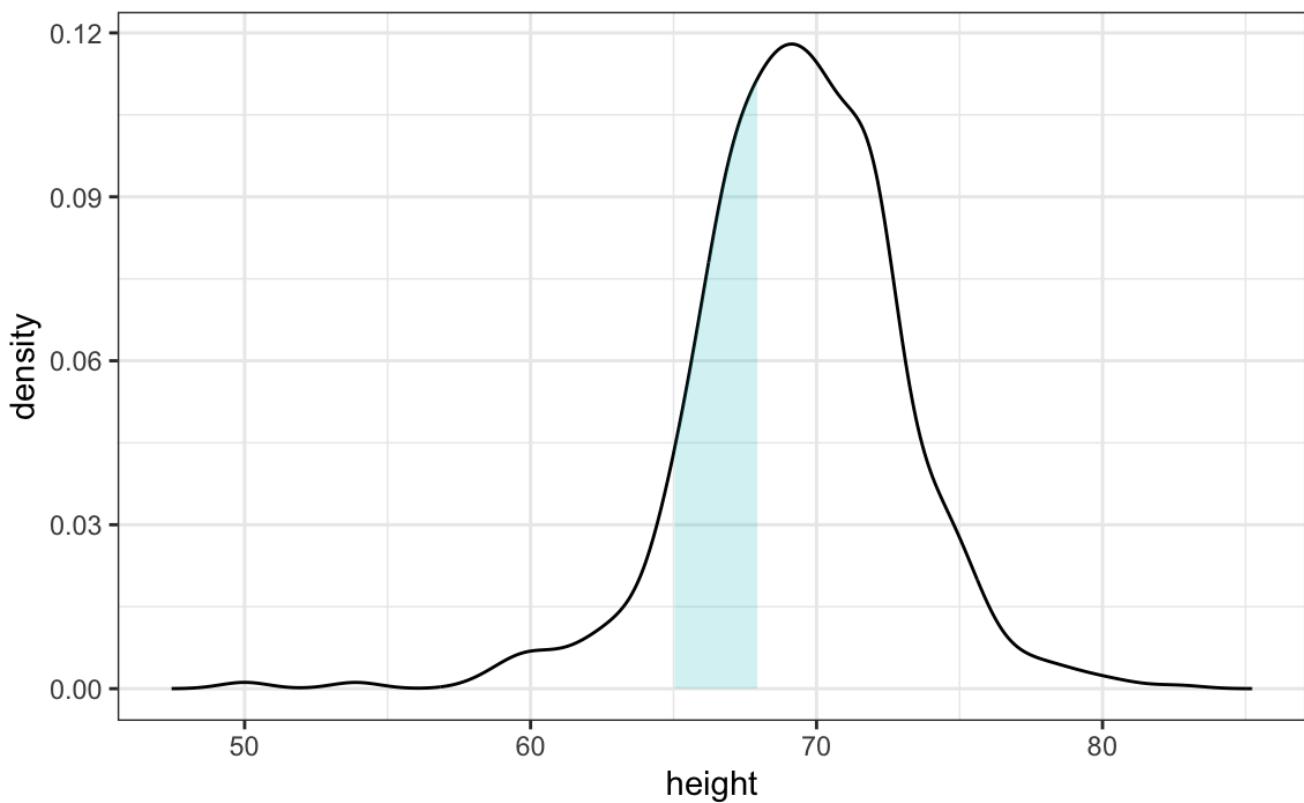


Figure 9:

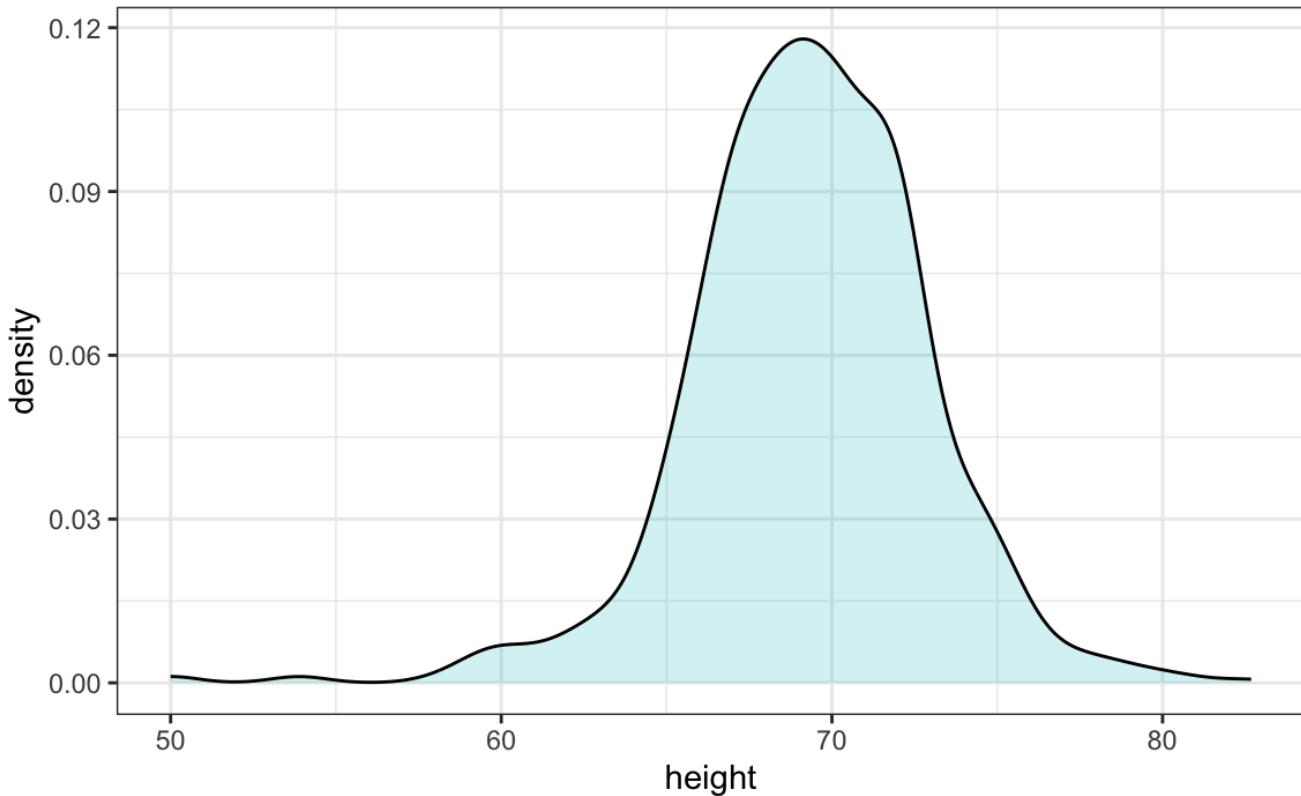


Figure 10:

With the material covered up to this point, you can complete exercises 1 through 10.

1.5 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. One 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 these occurrences, which we will describe 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:

$$\Pr(a < x \leq b) = \int_a^b \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx$$

You don't need to memorize or understand the details of the formula. However, it is important to note that it is completely defined by just two parameters: (μ) and (σ) . The rest of the symbols in the formula represent the interval ends, (a) and (b) , and known mathematical constants (π) and (e) . These two parameters, (μ) and (σ) , are referred to as the *average* (also called the *mean*) and the *standard deviation* (SD) of the distribution, respectively (and are the Greek letters for (m) and (s)).

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

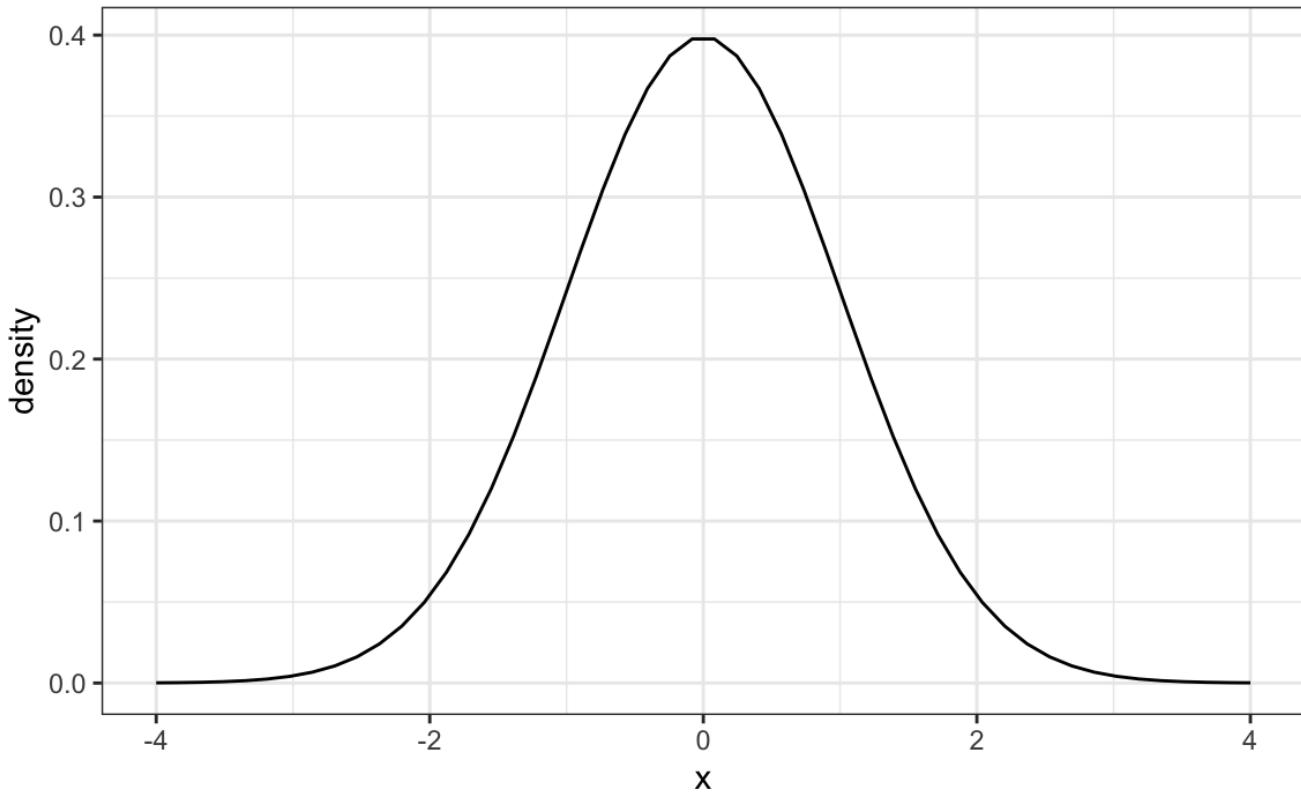


Figure 11:

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 - m)^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 \mathbf{x} :

```
index <- heights$sex == "Male"
x <- heights$height[index]
```

The pre-built functions `mean` and `sd` can be used here:

```
m <- mean(x)
s <- sd(x)
```

For reasons explained in Section 11.2.1, `sd(x)` divides by `length(x)-1` rather than `length(x)`. But note that when `length(x)` is large, `sd(x)` and `sqrt(sum((x-mu)^2) / length(x))` are practically equal.

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 will now see how well this approximation works at predicting the proportion of values within intervals.

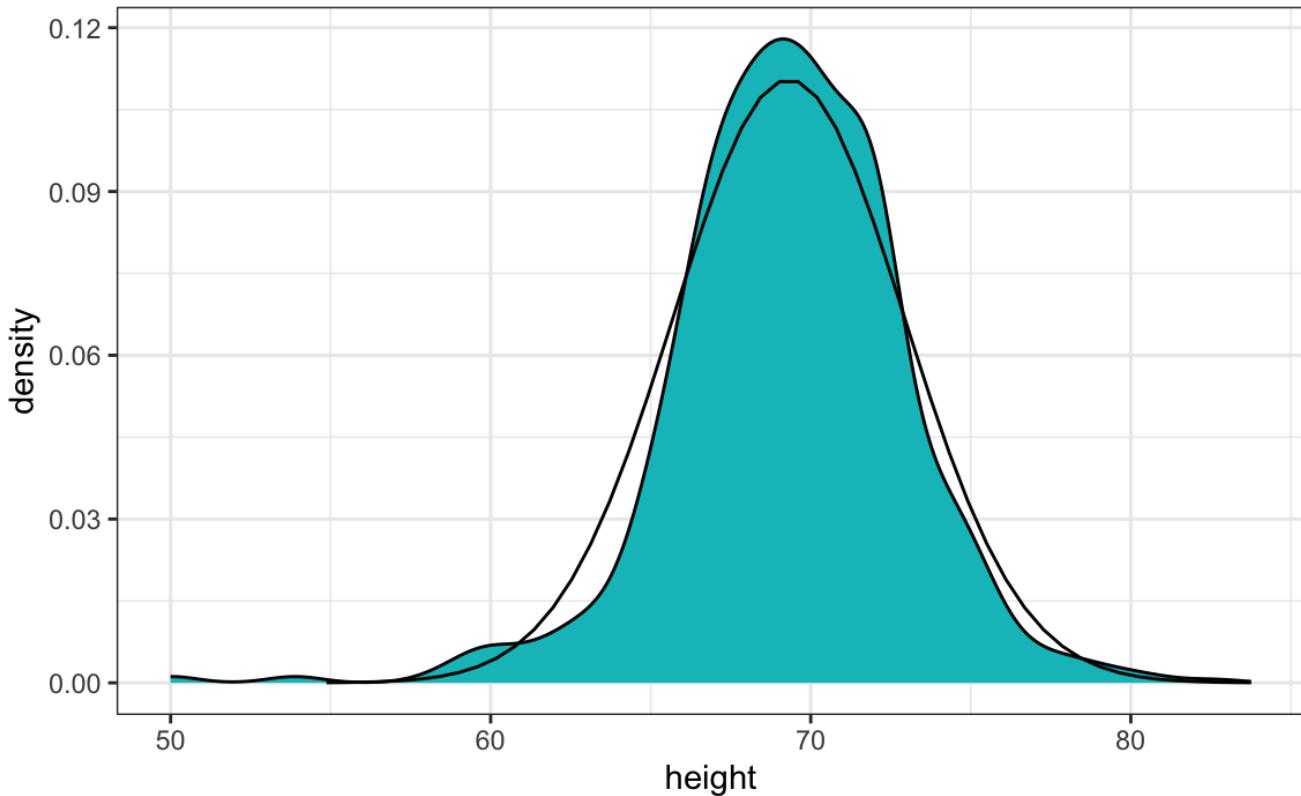


Figure 12:

1.6 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 X , we define the value of x in standard units as $z = (x - \bar{m})/s$ with \bar{m} and s the average and standard deviation of X , respectively. Why is this convenient?

First, revisit the formula for the normal distribution and observe 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. Secondly, note that by converting the normally distributed data to standard units, we can quickly ascertain whether, 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)
```

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 validate this approximation, we can use quantile-quantile plots.

1.7 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 (qqplot).

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 proportion of a standard normal distributed data that are smaller than x . So, for example, $\Phi(-1.96) = 0.025$ and $\Phi(1.96) = 0.975$. In R, we can evaluate Φ 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. Thus, for instance, $\Phi^{-1}(0.975) = 1.96$. In R, we can evaluate the inverse of Φ 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, hence the name *theoretical quantiles*. But quantiles can be defined for any distribution, including an empirical one. If we therefore 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
```

Therefore about 50% are shorter or equal to 69 inches. This implies that if $(p = 0.50)$, then $(q = 69.5)$.

The idea of a qqplot 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 qqplot, 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 qqplot 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()
#> Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```

Notice that this code becomes much cleaner if we use standard units:

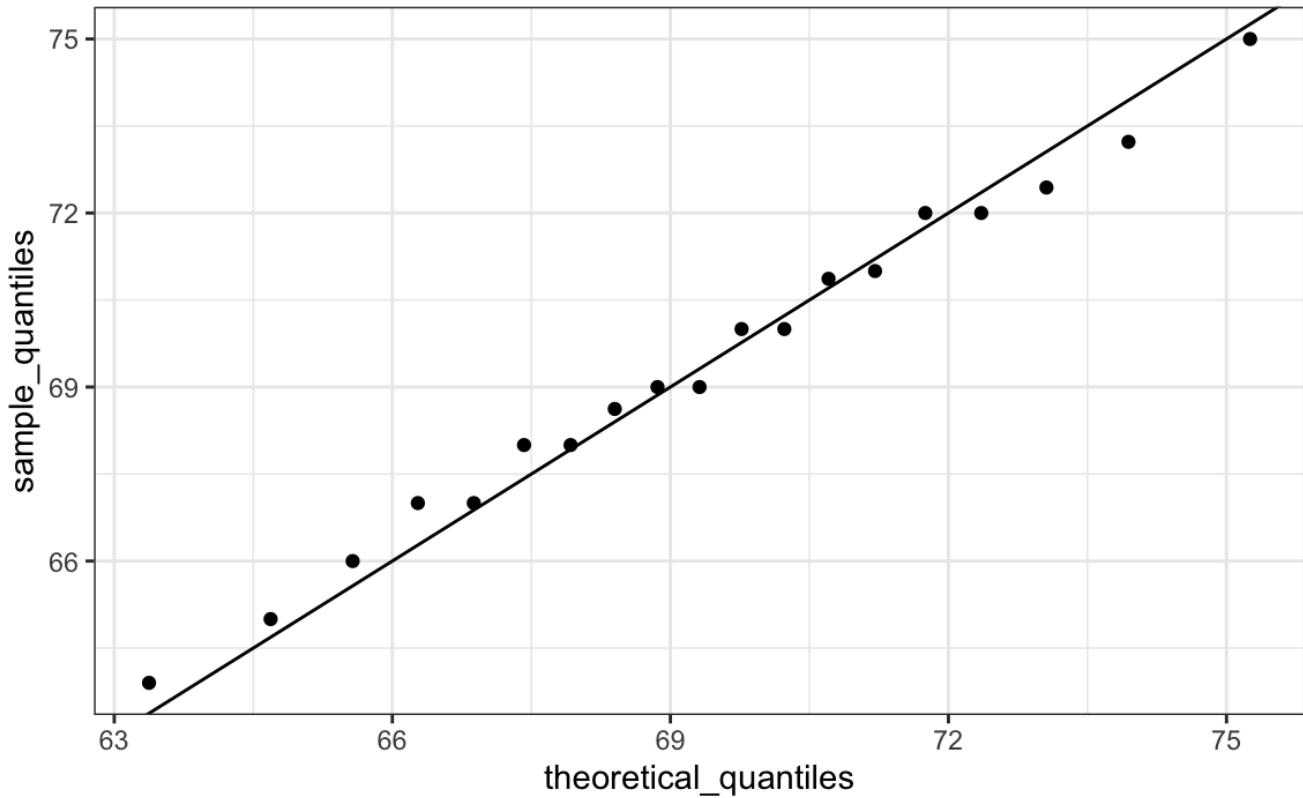


Figure 13:

```
sample_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```

The above code is included to help describe qqplots. However, in practice it is easier to use `ggplot2` code:

```
heights |> filter(sex == "Male") |>
  ggplot(aes(sample = scale(height))) +
  geom_qq() +
  geom_abline()
```

In the illustration above we used 20 quantiles; however, the default for the `geom_qq` function is to use as many quantiles as data points.

Although here we used qqplots to compare an observed distribution to the mathematically defined normal distribution, qqplots can be used to compare any two distributions.

1.8 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 $\backslash(p\backslash)$ at $\backslash(0.01, 0.02, \dots, 0.99\backslash)$. For example, we refer to the case of $\backslash(p = 0.25\backslash)$ as the 25th percentile, representing a value below which 25% of the data falls. 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 $\backslash(p = 0.25, 0.50\backslash)$, and $\backslash(0.75\backslash)$.

1.9 Boxplots

To introduce boxplots, we will use a dataset of US murders by state. Suppose we want to summarize the murder rate distribution. Using the techniques we have learned, we can quickly see that the normal approximation does not apply in this case:

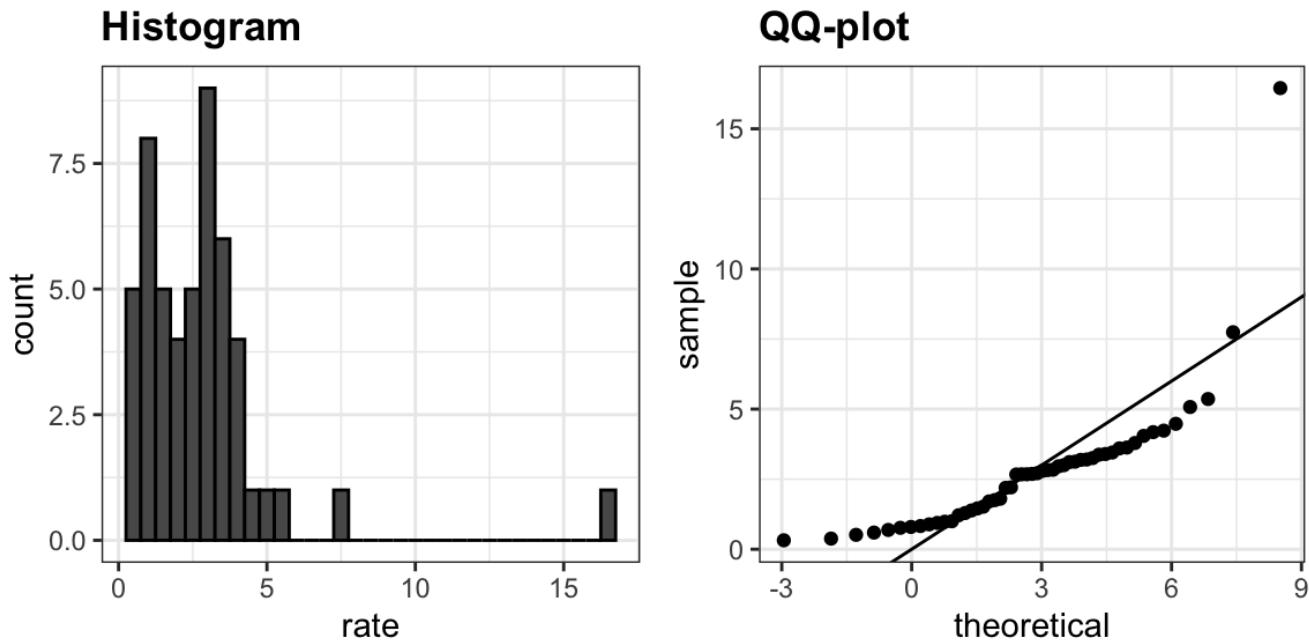


Figure 14:

In this instance, 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.

The boxplot provides a five-number summary composed of the range along with the quartiles (the 25th, 50th, and 75th percentiles). The boxplot often ignores *outliers* when computing the range and instead plots these as independent points. We will provide a detailed explanation of outliers later. Finally, 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 considered outliers by the default R function we used. The median represented by 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.

1.10 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.

Using the histogram, density plots, and qqplots, 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.

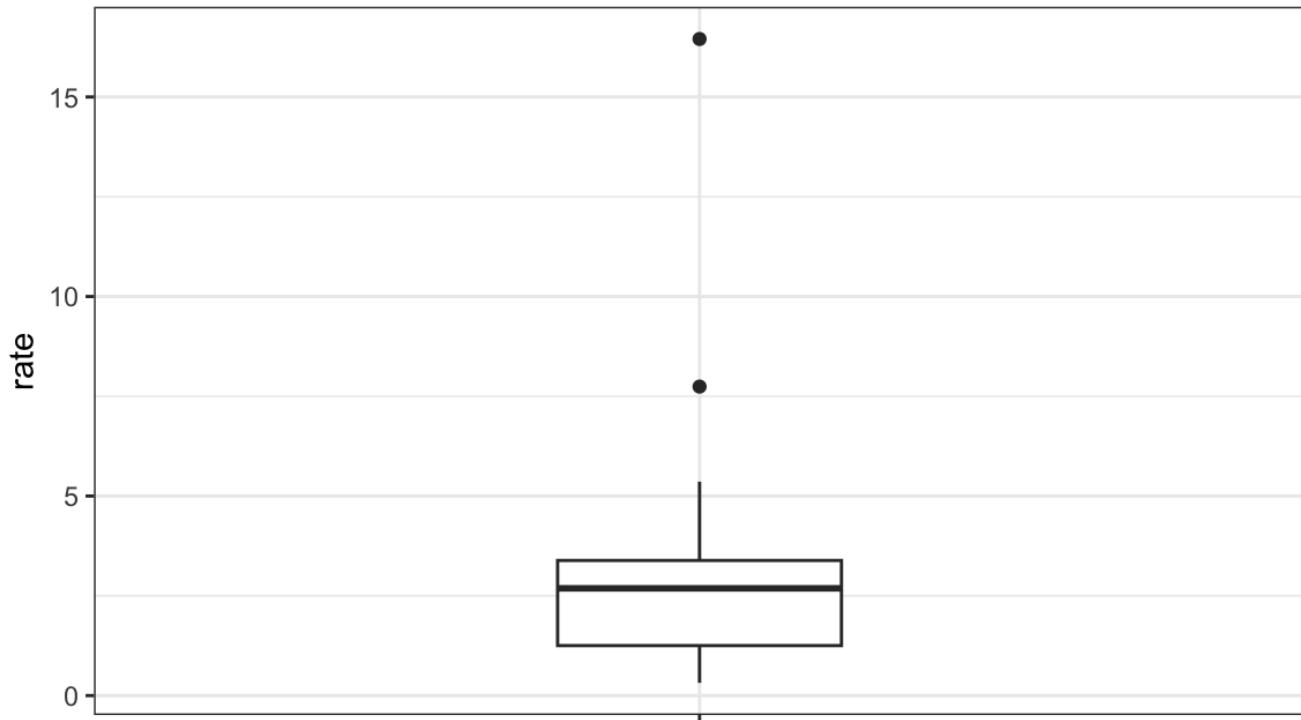


Figure 15:

We learned that boxplots are useful when we want to quickly compare two or more distributions. Here are the heights for men and women:

```
heights |> ggplot(aes(sex, height, fill = sex)) + geom_boxplot()
```

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 qqplot shows that the highest points tend to be taller than expected by the normal distribution. Finally, we also see five points in the qqplot 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.

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 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".

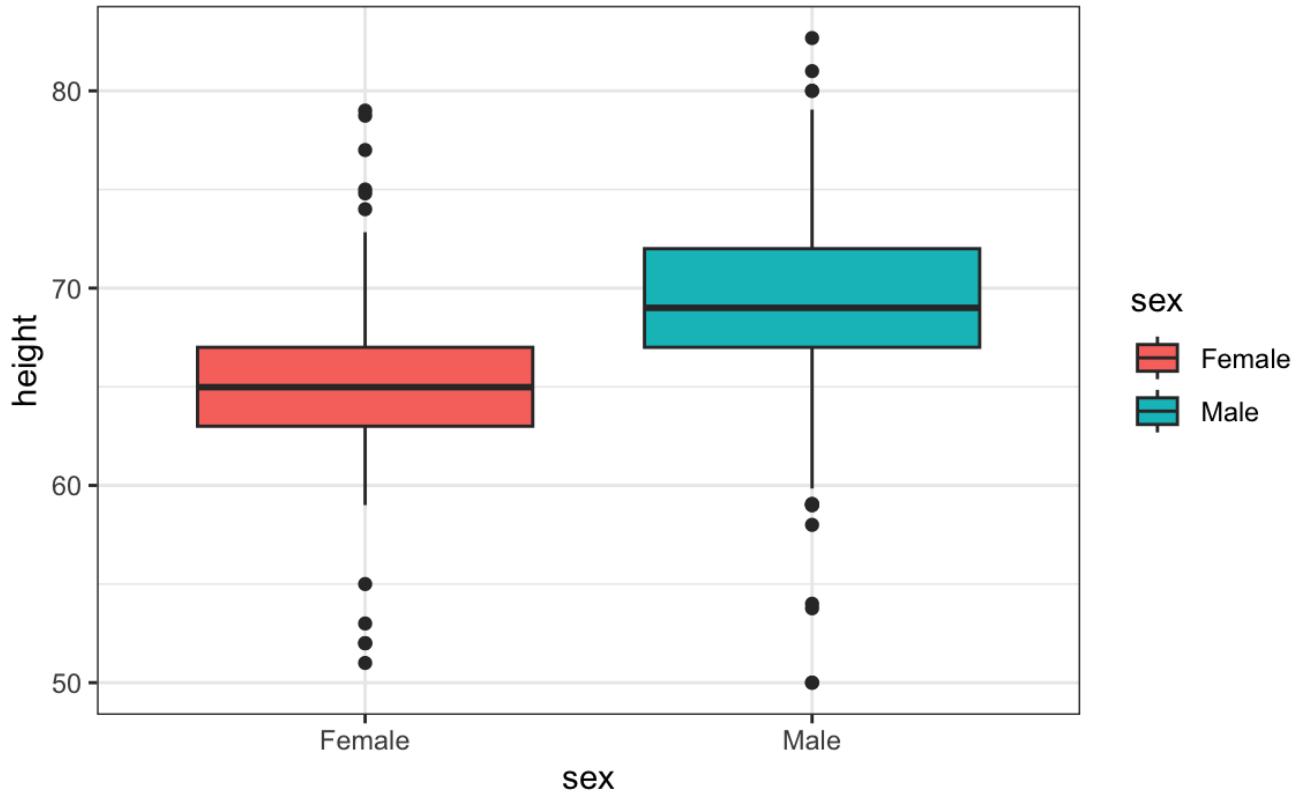


Figure 16:

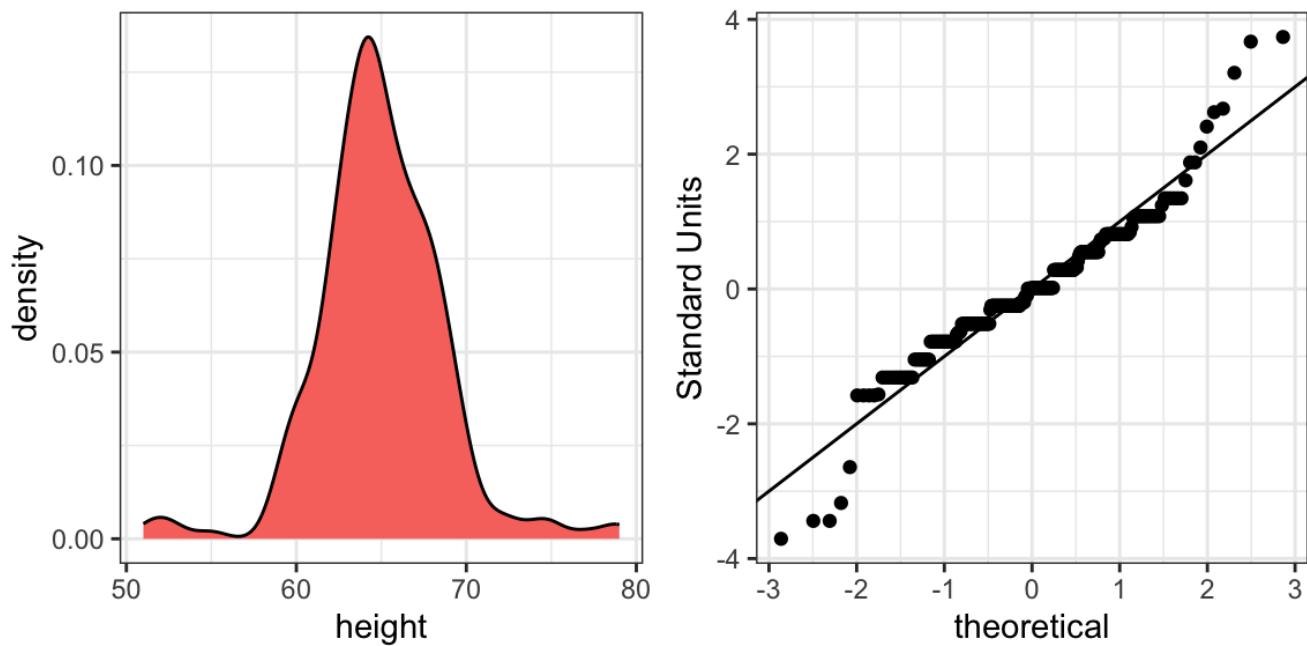


Figure 17:

1.11 Exercises

1. In the `murders` dataset, the region is a categorical variable and the following is its distribution:

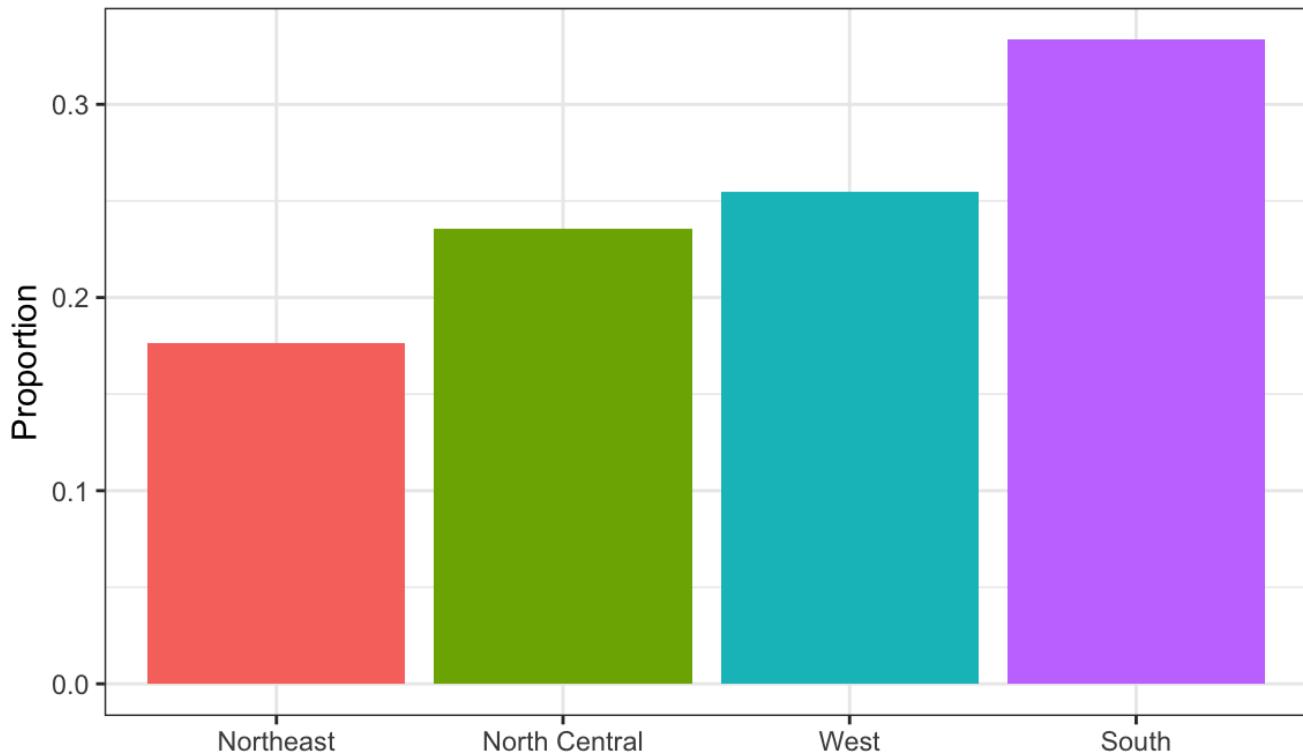


Figure 18:

To the closest 5%, what proportion of the states are in the North Central region?

2. Which of the following is true:

- a. The graph above is a histogram.
- b. The graph above shows only four numbers with a bar plot.
- c. Categories are not numbers, so it does not make sense to graph the distribution.
- d. 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?

- a. 100%
- b. 95%
- c. 80%
- d. 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:

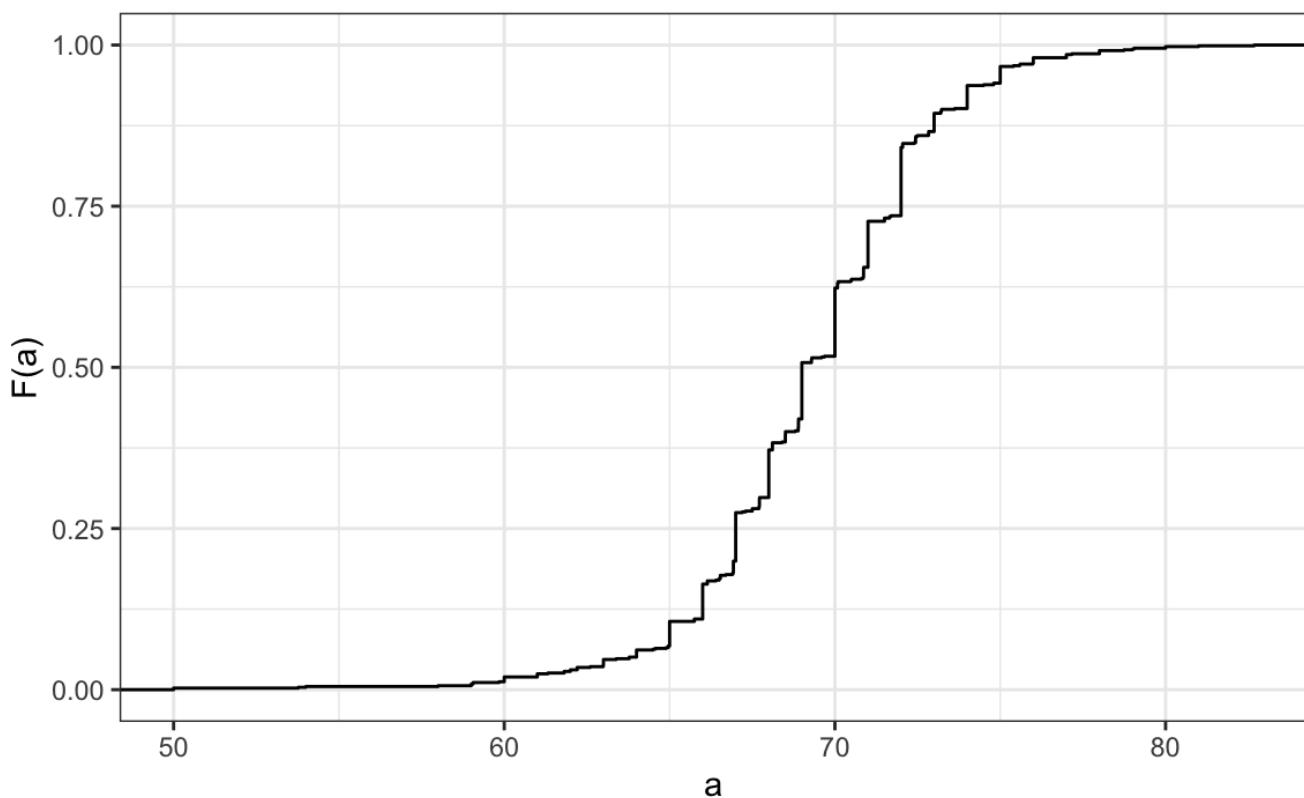


Figure 19:

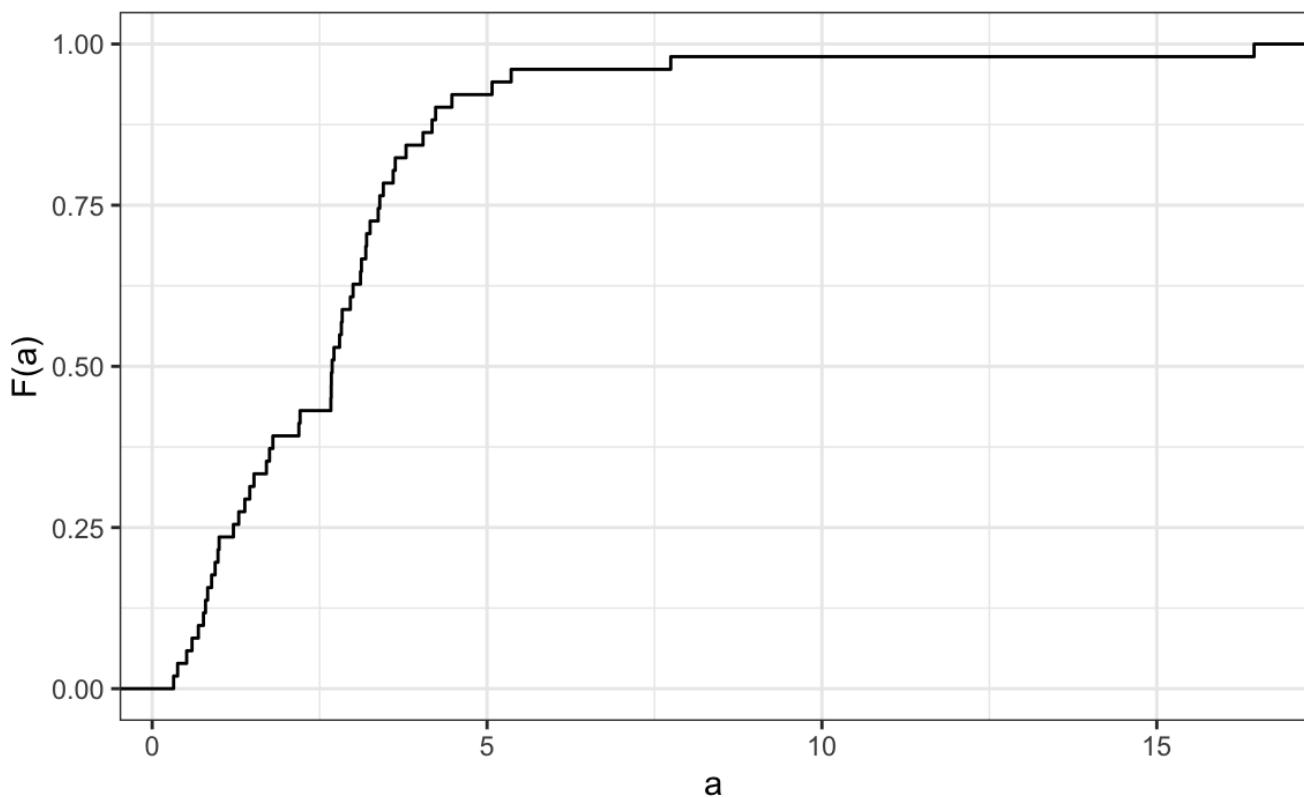


Figure 20:

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:

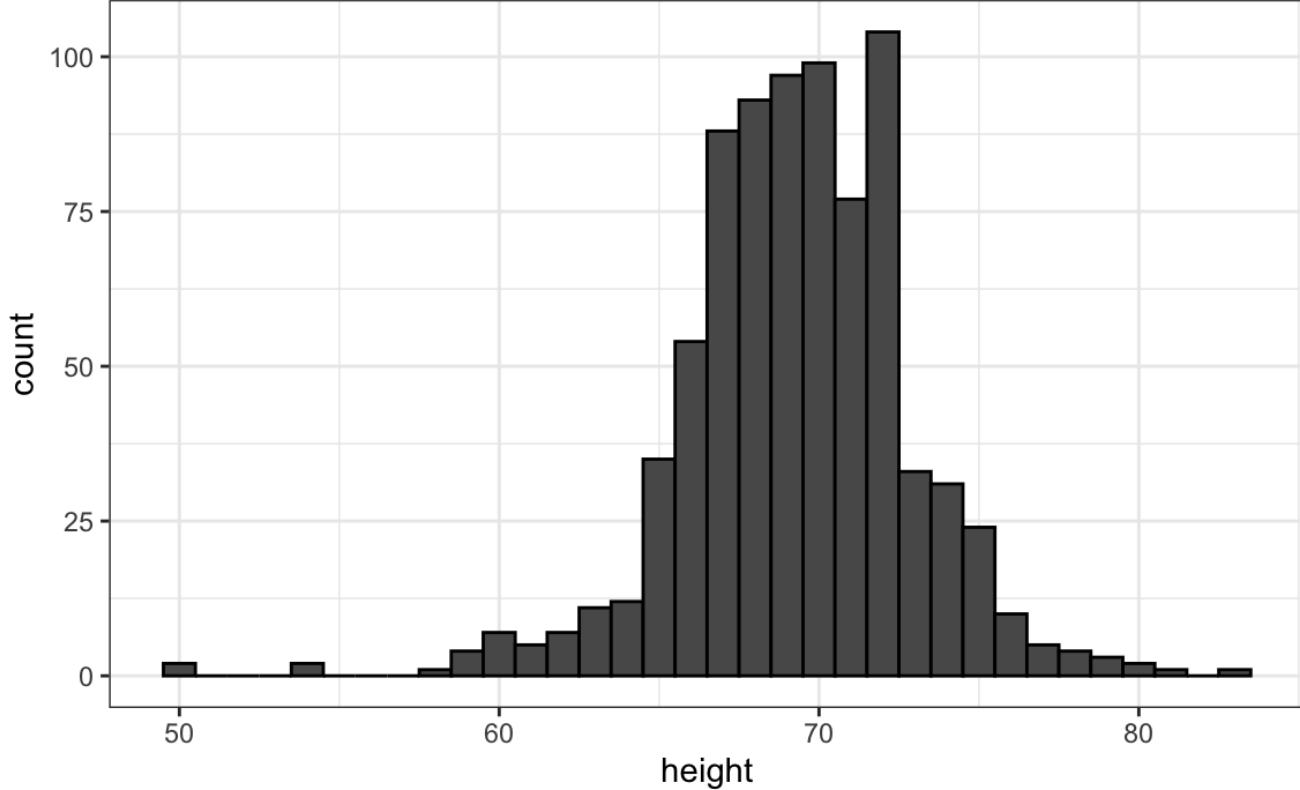


Figure 21:

Based on this plot, how many males are between 63.5 and 65.5?

- a. 10
 - b. 24
 - c. 47
 - 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?

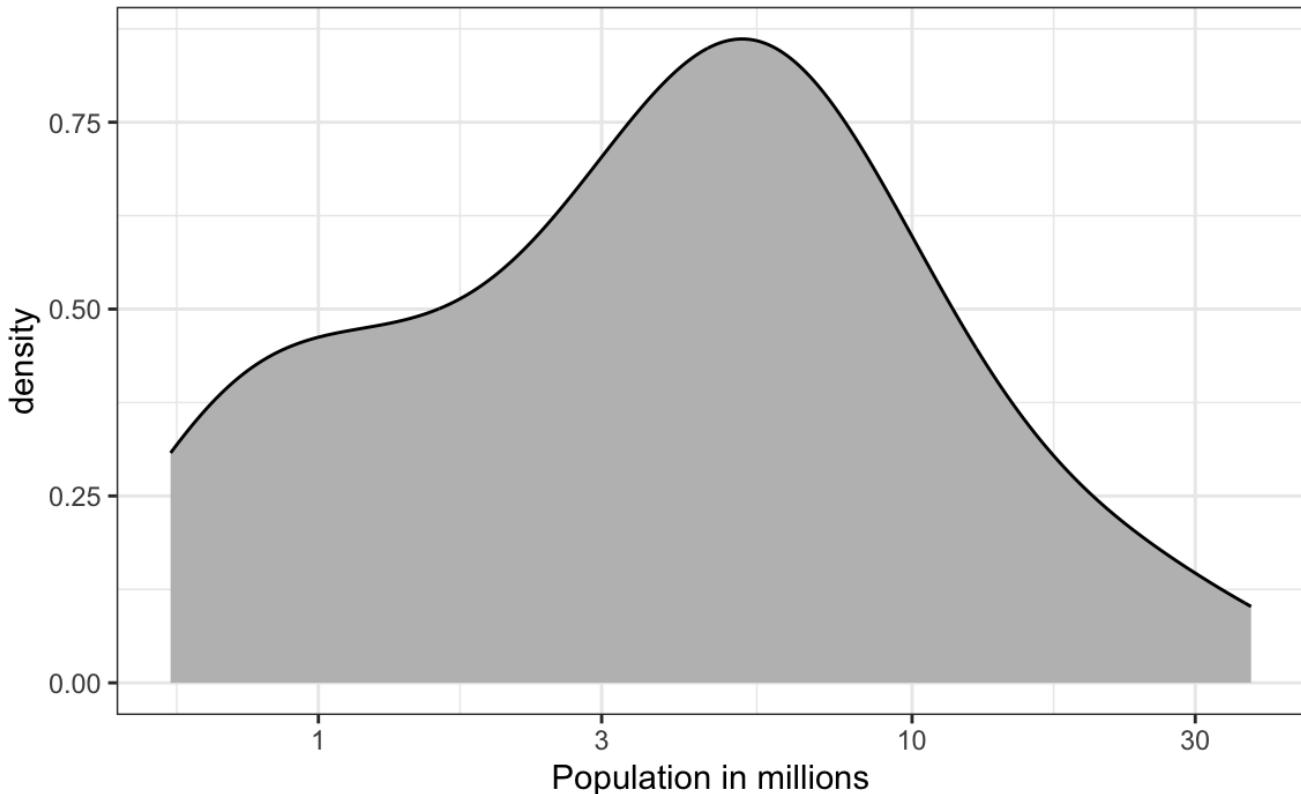


Figure 22:

- 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.
11. Define variables containing the heights of males and females as follows:
- ```
library(dslabs)
male <- heights$height[heights$sex == "Male"]
female <- heights$height[heights$sex == "Female"]
```
- How many measurements do we have for each?
12. 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, 70th, & 90th percentiles for each sex. Then create a data frame with these two as columns.
13. Study the following boxplots showing population sizes by country:
- Which continent has the country with the biggest population size?

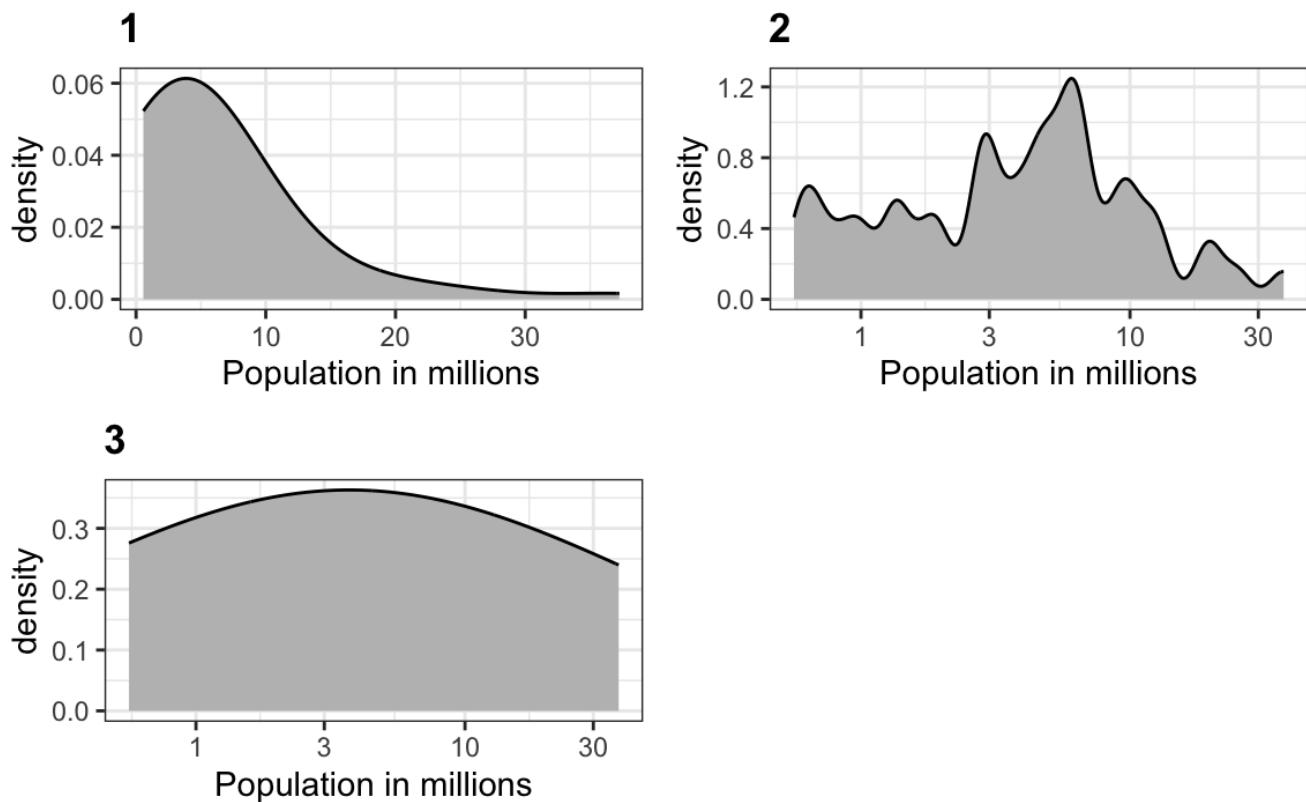


Figure 23:

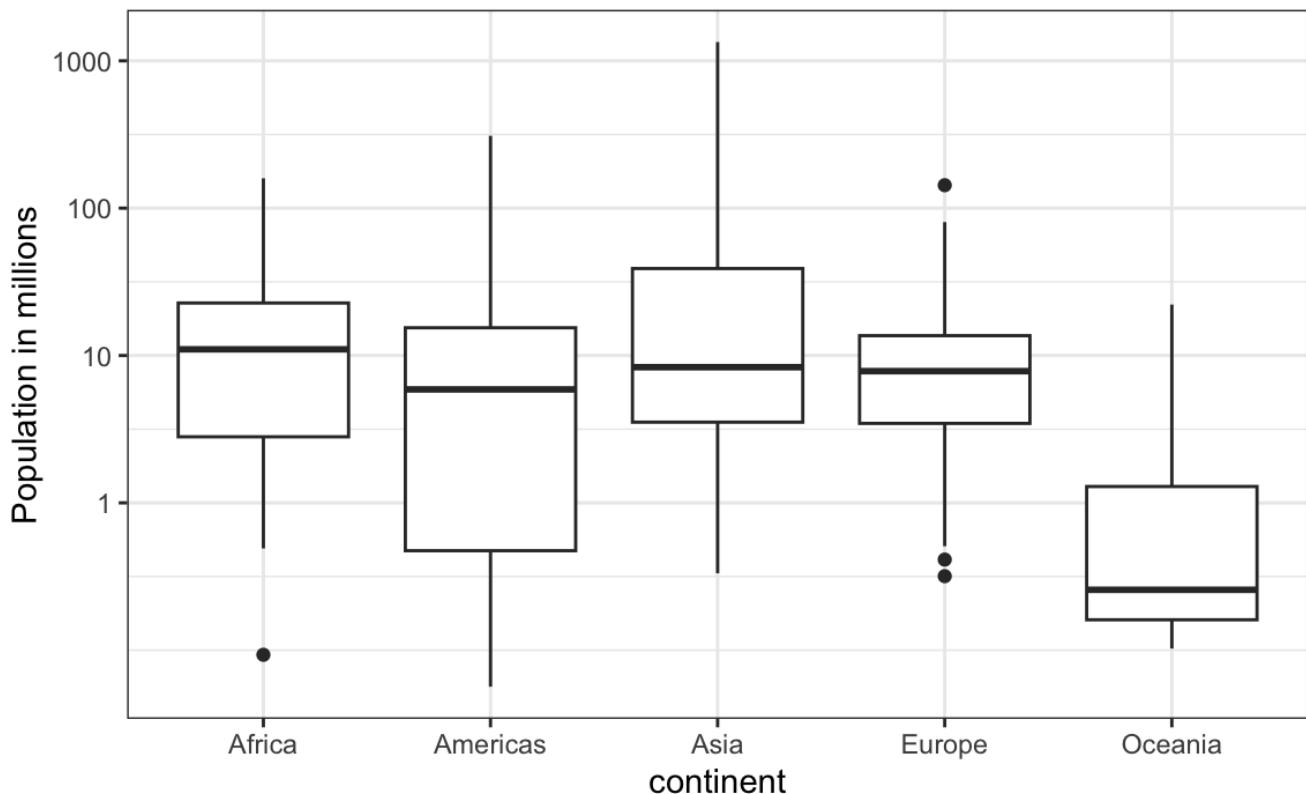


Figure 24:

14. Which continent has the largest median population size?
15. What is median population size for Africa to the nearest million?
16. What proportion of countries in Europe have populations below 14 million?
  - a. 0.99
  - b. 0.75
  - c. 0.50
  - d. 0.25

17. If we use a log transformation, which continent shown above has the largest interquartile range?

18. Load the height dataset and create a vector `x` with just the male heights:

```
library(dslabs)
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`.

19. 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.
20. Notice that the approximation calculated in question nine 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 larger is the actual proportion than the approximation?
21. 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.
22. 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?
23. 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.
25. In answering the previous questions, we found that it is not uncommon 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 10, 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 10, the normal approximation tends to overestimate the extreme values. It's possible that there are fewer seven-footers than we predicted.

# Introduction to Data Science - 2 Robust summaries

Rafael A. Irizarry

Note that the heights we explored in the Chapter 1 are not the original heights reported by students. A second challenge involves exploring the *original* reported heights, which are also included in the **dslabs** package in the **reported\_heights** object. We will see that due to errors in reporting, using *robust summaries* are necessary to produce useful summaries.

## 2.1 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 real-world data analysis. 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. For example, an individual 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(dslabs)
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 analysis 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 -9.4892954, 21.6969354 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.

## 2.2 The 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

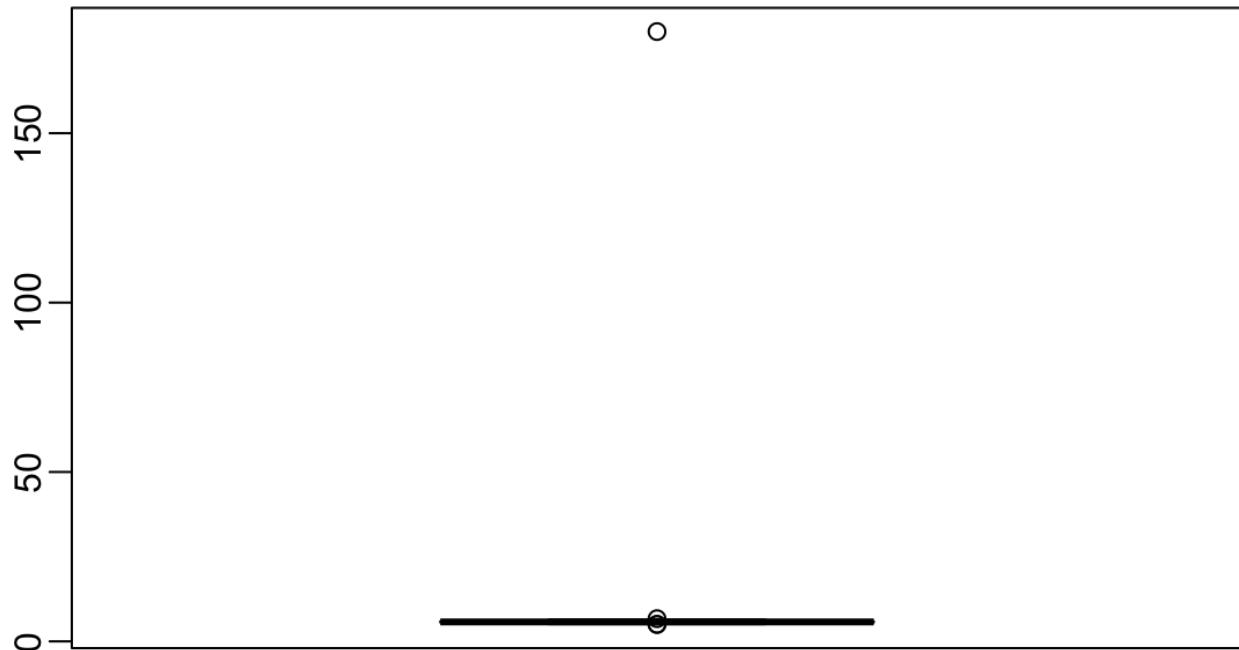


Figure 1:

$\backslash(\Delta)$  by adding  $\backslash(\Delta \times)$  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.

### 2.3 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.

## 2.4 A data-driven definition of outliers

In R, points falling outside the whiskers of the boxplot are referred to as *outliers*. This definition of outlier was introduced by John 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 1,000 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, 99.9998% 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)
```

## 2.5 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.

## 2.6 Exercises

We are going to use the **HistData** package. Load the height data set and create a vector `x`, consisting solely of the male heights used in Galton's data on the heights of parents and their children, part of his historic research on heredity.

```
library(HistData)
x <- Galton$child
```

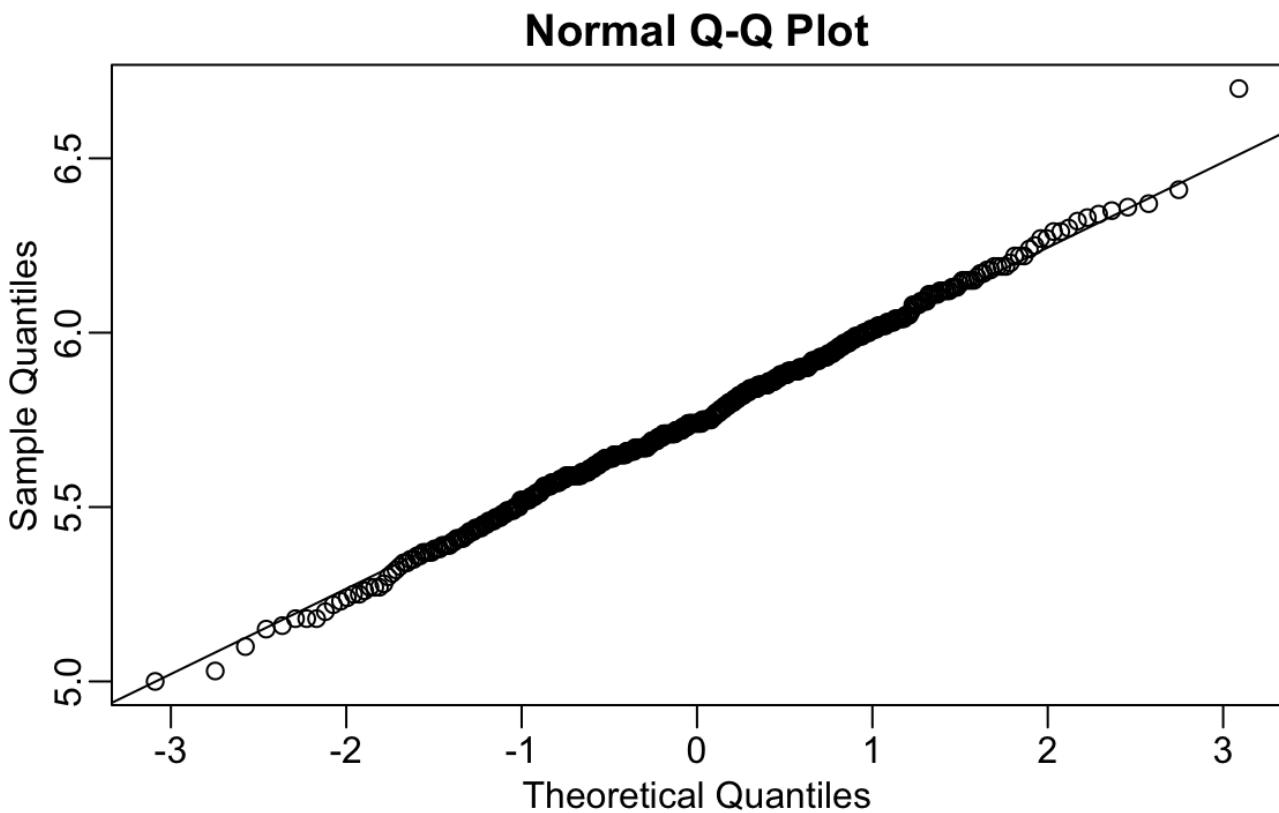


Figure 2:

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 qqplot 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`.
9. Using the `murders` dataset in the `dslabs` package. Compute the murder rate for each state. Make a boxplot comparing the murder rates for each region of the United States.
10. For the same dataset, compute the median and IQR murder rate for each region.

11. Add a column to the `reported_heights` with the year the height was entered. You can use the `year` function in the `lubridate` package to extract the year from `reported_heights$time_stamp`. Change the `height` column from characters to numbers using `parse_number` from the `readr` package. Some of the heights will be converted to `NA` because they were incorrectly entered and include characters, for example `165cm`. These heights were supposed to be reported in inches, but many clearly did not. Convert any entry below 54 or above 72 to `NA` using the `na_if` function from `dplyr`. Once you do this, stratify by sex and year and report the percentage of incorrectly entered heights, represented by the `NA`.

12. 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 in the object `reported_heights`. Note that the `height` column in this data frame is a character, and if we try to create a new column with the numeric version:

```
library(tidyverse)
reported_heights <- reported_heights |>
 mutate(original_heights = height, height = as.numeric(height))
```

we get a warnings about NAs. Examine the rows that result in NAs and describe why this is happening.

13. Remove the entries that result in NAs when attempting to convert heights to numbers. Compute the mean, standard deviation, median, and MAD by sex. What do you notice?

14. Generate boxplots summarizing the heights for males and females and describe what you see.

15. Look at the largest 10 heights and provide a hypothesis for what you think is happening.

16. Review all the nonsensical answers by looking at the data considered to be *far out* by Tukey and comment on the type of errors you see.

# Introduction to Data Science - Probability

Rafael A. Irizarry

In games of chance, probability has a very intuitive definition. 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 section of the book is to help us in comprehending how probability is useful for understanding and describing real-world events when performing data analysis. Probability theory is useful whenever our data is affected by chance in some manner. All the other sections in this book build upon probability theory. A knowledge of probability is therefore indispensable for addressing most data analysis challenges.

Given that knowing how to compute probabilities gives strategic advantage in games of chance, many smart individuals throughout history, 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. Additionally, casinos depend on probability theory to develop games that almost certainly guarantee a profit. We will use casino games to illustrate the fundamental concepts.

This part of the book discusses concepts that can be found in many comprehensive books on probability theory. These books delve into the mathematical theories and formulas behind probability.

This book, however, takes a different approach. Instead of diving into the mathematical theories, it uses R to demonstrate these concepts. This helps readers visualize and better understand the principles of probability in practical terms, as they can see the results and outcomes by running code.

Despite this practical approach, the book does not immediately apply these probability concepts to real-world data. This connection between probability theory and real data will be made in a subsequent section or part of the book.

In other words, while you're learning about probability now, it'll be a bit longer before you see how these concepts relate directly to real datasets.

# Introduction to Data Science - 3 Discrete probability

Rafael A. Irizarry

We begin by covering some basic principles related to categorical data. The specific area of probability which deals with categorical data is referred to as *discrete probability*. Understanding this topic, will help us comprehend the probability theory we will later introduce for numeric and continuous data, which is much more common in data analysis. Since discrete probability is invaluable in card games, we will use these as illustrative examples.

## 3.1 Relative frequency

The term *probability* is used in everyday language. Yet answering questions about probability is often hard, if not impossible. In this section, we discuss a mathematical definition of *probability* that allows 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 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 an equal 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.

## 3.2 Notation

We use the notation  $\Pr(A)$  to denote the probability of event  $(A)$  occurring. 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, where we randomly phone 100 likely voters at random, an example of an event is “calling 48 Democrats and 52 Republicans.”

In data science applications, we often encounter continuous variables. These events will often be questions, such as “Is this person taller than 6 feet?” In these cases, we represent events in a more mathematical form:  $(X \geq 6)$ . We will see more of these examples later, but for now, we will focus on categorical data.

## 3.3 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 analogous to 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 |

## 3.4 Monte Carlo

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 a single random outcome. We want to repeat this experiment an infinite number of times, but it is impossible to repeat indefinitely. Instead, we repeat the experiment a large enough number of times to make the results practically equivalent to repeating indefinitely. **This is an example of a *Monte Carlo* simulation.**

Much of what mathematical and theoretical statisticians study, topics that we do not cover in this book, relates to providing rigorous definitions of “practically equivalent”. Additionally, they explore how close a large number of experiments brings us to what happens in the limit. Later in this section, we provide a practical approach to determining what is “large enough”.

To perform our first Monte Carlo simulation, we use the `replicate` function, which allows us to repeat the same task any number of times. Here, we repeat the random event  $\backslash(B = \backslash)$  10,000 times:

```
B <- 10000
events <- replicate(B, sample(beads, 1))
```

We can now verify if our definition actually is in agreement with this Monte Carlo simulation approximation. We use `table` to see the distribution:

```
tab <- table(events)
tab
#> events
#> blue red
#> 6027 3973
```

and `prop.table` gives us the proportions:

```
prop.table(tab)
#> events
#> blue red
#> 0.603 0.397
```

The numbers above represent the estimated probabilities obtained by this Monte Carlo simulation. Statistical theory, not covered here, tells us that as  $\backslash(B \backslash)$  gets larger, the estimates get closer to  $3/5=.6$  and  $2/5=.4$ .

This is a simple and not very useful example, since we can easily compute the probabilities mathematically. Monte Carlo simulations are useful when it is hard, or impossible, to compute the exact probabilities mathematically. Before delving into more complex examples, we use simple ones to demonstrate the computing tools available in R.

### 3.4.1 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 potentially change by chance, indicating that a static version of the book may show a different result than what you obtain when following the code as presented. This is actually fine, given that the results are random and change

over time. However, if you want to ensure that results are consistent with each run, 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 every time. A popular way to pick the seed is the year - month - day. For example, we chose 1986 on December 20, 2018:  $\backslash(2018 - 12 - 20 = 1986\backslash)$ .

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.

### 3.4.2 With and without replacement

The function `sample` has an argument that allows 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 returned to 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 consistently comprise 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`.

### 3.5 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,  $\backslash(\backslashdots\backslash)$ , Ten, Jack, Queen, King, and Ace. Now, if we deal a King for the first card and do not replace it in the deck, the probability of a second card being a King decreases 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. However, 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, as the the only bead left is red. The events are not independent, so the probabilities change.

## 3.6 Conditional probabilities

When events are not independent, *conditional probabilities* are useful. We have already demonstrated 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.

## 3.7 Addition and multiplication rules

### 3.7.1 Multiplication rule

If we want to determine 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).

In a Blackjack game, to calculate the chances of obtaining a 21 by drawing an Ace and then a face card, we compute the probability of the first card being an Ace and multiply it by the probability of drawing a face card or a 10, given that the first card 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:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B \mid A)\Pr(C \mid A \text{ and } B)$$

### 3.7.2 Multiplication rule under independence

When dealing with independent events, the multiplication rule becomes simpler:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B)\Pr(C)$$

However, we have to be very careful before using this version of the multiplication rule, since assuming independence can result in very different and incorrect probability calculations when events are not actually independent.

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. Using the multiplication rule, we therefore conclude that only  $(1/10 \times 1/5) = 0.02$  have both.

But, to multiply like this, we need to assume independence! Let's say the conditional probability of a man having a mustache, conditional on him having a beard, is .95. Then, 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.

### 3.7.3 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; consider a Venn diagram. If we simply add the probabilities, we count the intersection twice, so we need to subtract one instance.

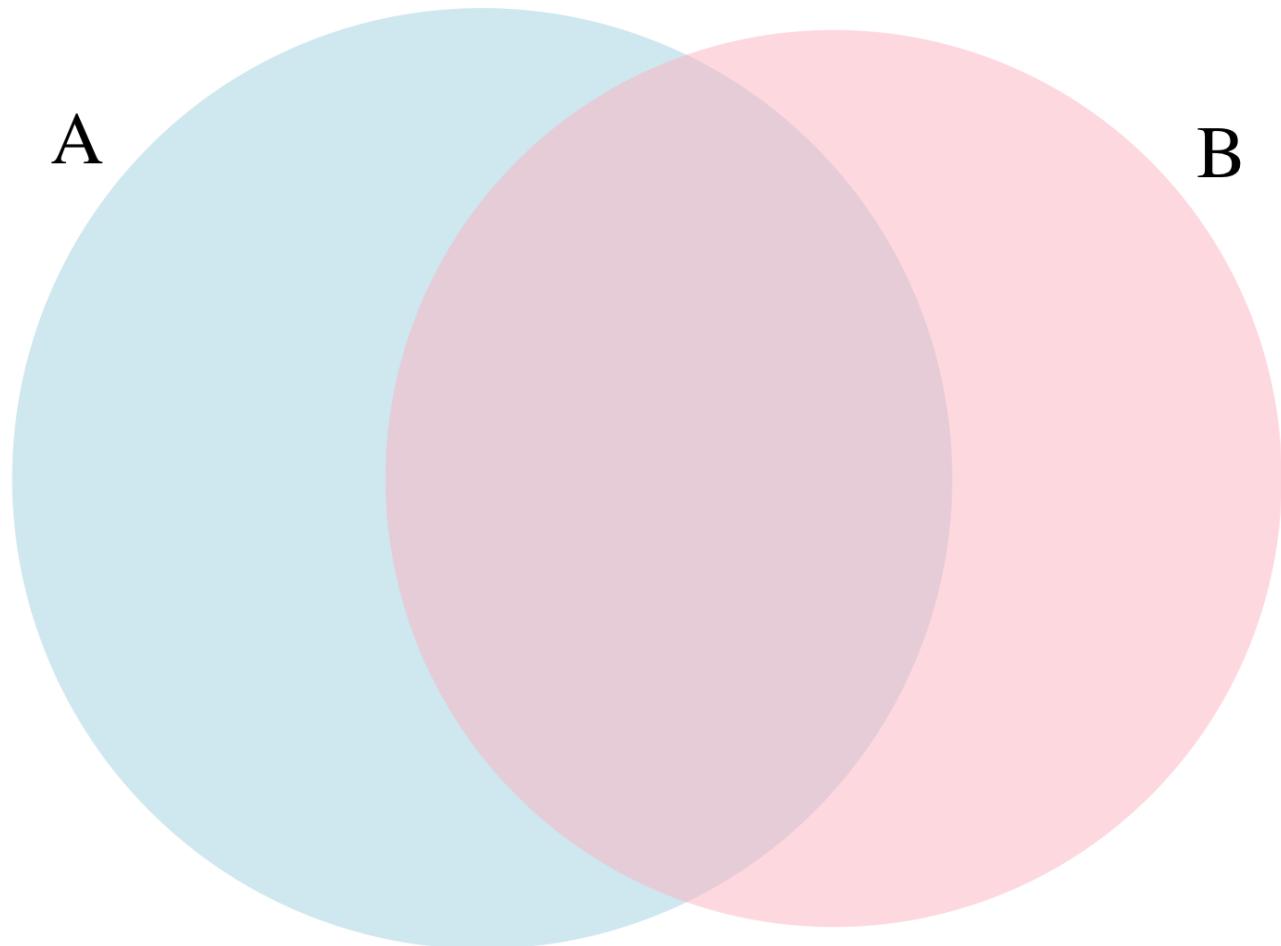


Figure 1:

### 3.8 Combinations and permutations

In our very first example, we imagined an urn with five beads. As a reminder, to compute the probability distribution of a single draw, we simply listed out all the possibilities, which amounted to 5. Subsequently, for each event, we counted how many of these possibilities were associated with the event. The resulting probability of choosing a blue bead is  $3/5$ , as three of the five possible outcomes 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, which 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 as 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, what about the conditional probability of the second card being a King, given that the first card was a King? Earlier, we deduced that if one King is already drawn from the deck, leaving 51 cards, 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 obtain 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
```

Observe 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 obtain 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 as follows:

```
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 as 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:

$$\left[ \frac{\Pr(A \text{ and } B)}{\Pr(A)} \right]$$

What about if the order does not matter? For example, in Blackjack, if you obtain 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 assumption is made based on our knowledge of how `combinations` enumerates possibilities, and it will list this case first. However, 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
```

### 3.8.1 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 obtain. We can use the function `sample` to draw two cards without replacement:

```
hand <- sample(deck, 2)
hand
#> [1] "Queen Clubs" "Seven Spades"
```

and then check if one card is an Ace and the other is a face card or a 10. Going forward, we include 10 when we refer to a *face card*. Now, we need to check both possibilities:

```
(hand[1] %in% aces & hand[2] %in% facecard) |
 (hand[2] %in% aces & hand[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
```

## 3.9 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.

### 3.9.1 Monty Hall problem

In the 1970s, there was a game show called “Let’s Make a Deal,” with Monty Hall as the host. At some point in the game, contestants were asked to pick one of three doors. Behind one door, there was a prize, while the other doors had a goat behind them to show the contestant 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 reveal to the contestant that 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 demonstrate 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 might seem 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(doors, 1)
 show <- sample(doors[!doors %in% c(my_pick, prize_door)], 1)
 stick <- my_pick
 stick == prize_door
 switch <- doors[!doors %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 see that the lines starting with `my_pick` and `show` have no influence on the last logical operation, when we stick to our original choice. From this, we should realize that the chance is 1 in 3, as we initially considered. When we switch, the Monte Carlo estimate confirms the 2/3 calculation. This helps us gain some insight by demonstrating 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, we win  $1 - 1/3 = 2/3$  of the times.

### 3.9.2 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 that later. Here, we use a Monte Carlo simulation. For simplicity, we assume nobody was born on February 29, which doesn’t significantly change the answer.

First, note that birthdays can be represented as numbers between 1 and 365, so a sample of 50 birthdays can be obtained as follows:

```
n <- 50
bdays <- sample(1:365, n, replace = TRUE)
```

To check if there are at least two people with the same birthday in this particular set of 50 people, we can use the `duplicated` function, 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; there were at least two people who had the same birthday.

To estimate the probability of a shared birthday in the group, we repeat this experiment by repeatedly sampling sets of 50 birthdays:

```
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.

Let's say we want to use this knowledge to make bet with friends about the likelihood of two people sharing the same birthday in a group. At what group size do the chances become greater than 50%? Greater 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  $\backslash(n\backslash)$ :

```
library(tidyverse)
prob <- sapply(n, compute_prob)
qplot(n, prob)
```

Now, let's compute the exact probabilities instead of relying on Monte Carlo approximations. Not only do we obtain the exact answer using math, but the computations are much faster since we did not 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.

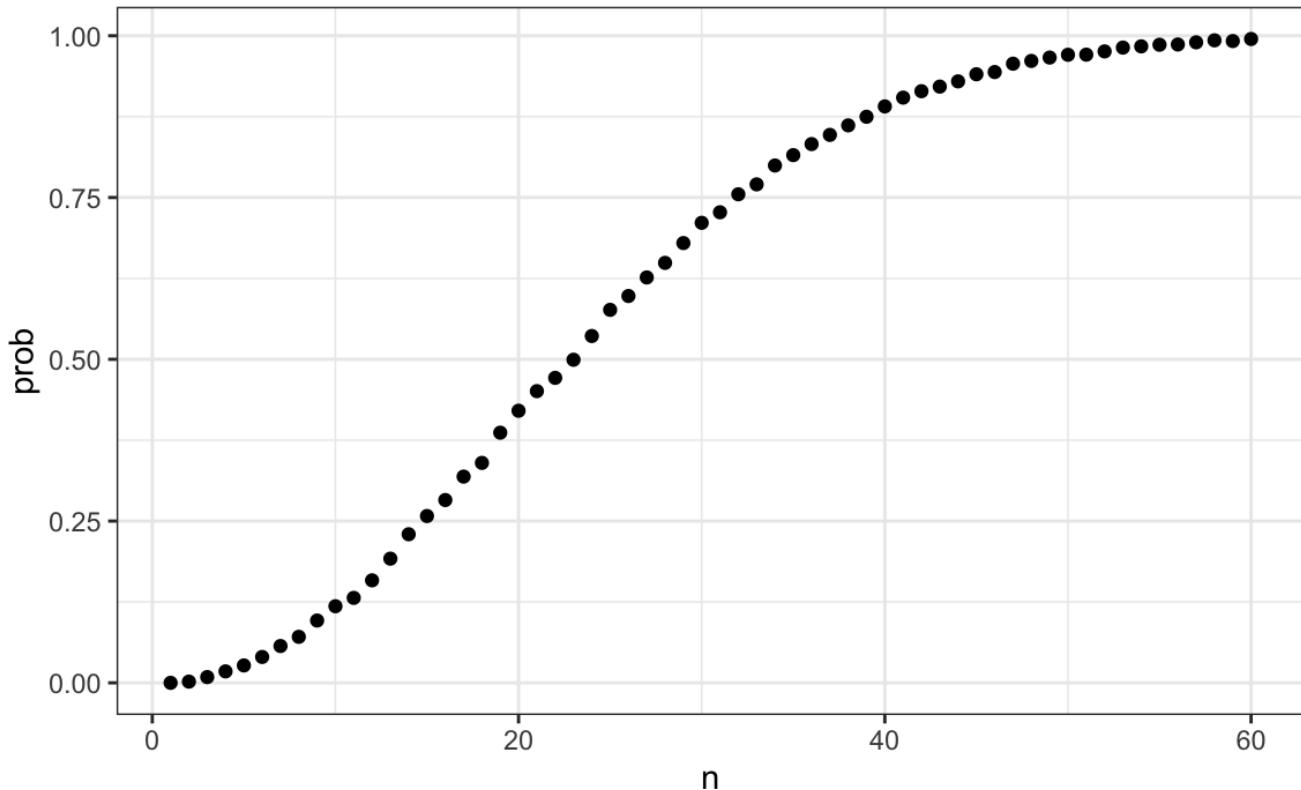


Figure 2:

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 day, is  $364/365$ . Subsequently, given that the first two people have unique birthdays, person 3 is left with 363 days to choose from. This pattern continues, and we find that the chances of all 50 people having a unique birthday is:

$$[ 1 \times \frac{364}{365} \times \frac{363}{365} \dots \times \frac{365-n+1}{365} ]$$

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 we not been able to compute the exact probabilities, we could still accurately estimate them.

### 3.10 Infinity in practice

The theory described here requires repeating experiments over and over indefinitely. In practice, we can't do this. In the examples above, we used  $(B=10,000)$  Monte Carlo experiments, yielding 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. However, in more complex calculations, 10,000 may not be nearly enough. Moreover, for some calculations, 10,000 experiments might not be computationally feasible.

In practical scenarios, we won't know what the answer is beforehand, so we won't know if our Monte Carlo estimate is accurate. We know that the larger the  $(B)$ , the better the approximation. But how large do we need it to be? This is actually a challenging question, and answering it often requires advanced theoretical statistics training.

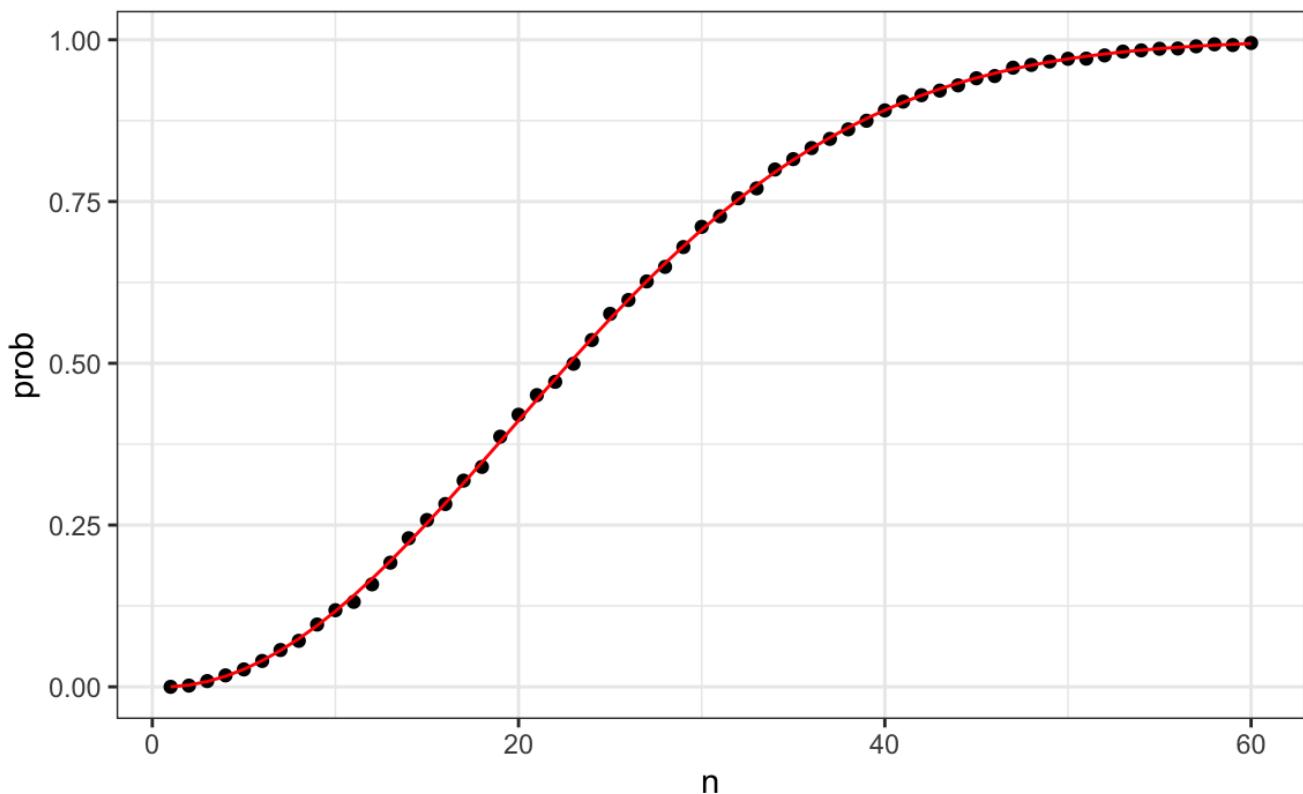


Figure 3:

One practical approach is to check for the stability of the estimate. The following example illustrates 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)
plot(log10(B), prob)
```

In this plot, we can see that the values start to stabilize at around 1000. Note that the exact probability, which is known in this case, is 0.5686997.

### 3.11 Exercises

1. 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?
2. What is the probability that the ball will not be cyan?
3. 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 of the first draw being cyan and the second draw not being cyan?
4. 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 of the first draw being cyan and the second draw not being cyan?
5. Two events  $\{A\}$  and  $\{B\}$  are independent if  $\Pr(A \cap B) = \Pr(A) \Pr(B)$ .

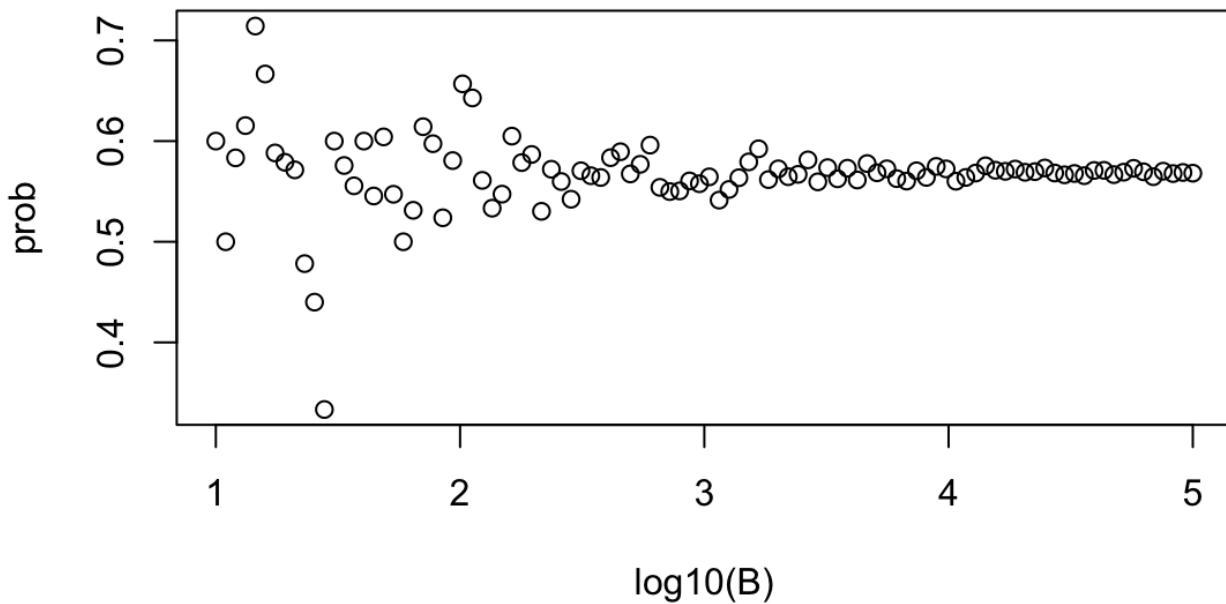


Figure 4:

Under which situation are the draws independent?

- a. You don't replace the draw.
  - b. You replace the draw.
  - c. Neither.
  - d. Both.
6. Let's say you've drawn 5 balls from the box, with replacement, and all have been yellow. What is the probability that the next one will be yellow?
7. If you roll a 6-sided die six times, what is the probability of not seeing a 6?
8. Two teams, let's 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 will win **at least** one game?
9. Create a Monte Carlo simulation to confirm your answer to the previous problem. Use  $B \leftarrow 10000$  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,  $\backslash(A\backslash)$  and  $\backslash(B\backslash)$ , are playing a seven game series. Team  $\backslash(A\backslash)$  is better than team  $\backslash(B\backslash)$  and has a  $\backslash(p>0.5\backslash)$  chance of winning each game. Given a value  $\backslash(p\backslash)$ , the probability of winning the series for the underdog team  $\backslash(B\backslash)$  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, best of 3 games, best of 5 games,... Specifically, `N <- seq(1, 25, 2)`. Hint: use the function below.

```

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)
}

```

---

1. [https://en.wikipedia.org/wiki/Urn\\_problem](https://en.wikipedia.org/wiki/Urn_problem)
2. <https://www.khanacademy.org/math/precalculus/prob-comb/dependent-events-precalc/v/monty-hall-problem>
3. [https://en.wikipedia.org/wiki/Monty\\_Hall\\_problem](https://en.wikipedia.org/wiki/Monty_Hall_problem)

# Introduction to Data Science - 4 Continuous probability

Rafael A. Irizarry

In Section 1.2, 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. Similarly, for a random variable that can take any value in a continuous set, it is impossible to assign a positive probabilities to the infinite number of possible values. Here, we outline the mathematical definitions of distributions for continuous random variables and useful approximations frequently employed in data analysis.

## 4.1 Cumulative distribution functions

We used the heights of adult male students as an example:

```
library(tidyverse)
library(dslabs)
x <- heights %>% filter(sex == "Male") %>% pull(height)
```

and defined the empirical cumulative distribution function (eCDF) 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  $a$ .

Let's connect the eCDF to probability by asking: if I randomly pick one of the male students, what is the chance that he is taller than 70.5 inches? Since 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 eCDF we obtain an answer by typing:

```
1 - F(70)
#> [1] 0.377
```

The CDF is a version of the eCDF that assigns theoretical probabilities for each  $\Pr(X \leq a)$  rather than proportions computed from data. Although, as we just demonstrated, proportions computed from data can be used to define probabilities for a random variable. Specifically, the CDF for a random outcome  $(X)$  defines, for any number  $(a)$ , the probability of observing a value larger than  $(a)$ .

$$\Pr(X \leq a)$$

Once a CDF is defined, we can use it to compute the probability of any subset of values. For instance, the probability of a student being between height  $a$  and height  $b$  is:

$$\Pr(a < X \leq b) = F(b) - F(a)$$

Since we can compute the probability for any possible event using this approach, the CDF defines the probability distribution.

## 4.2 Probability density function

A mathematical result that is very useful in practice is that, for most CDFs, we can define a function, call it  $(f(x))$ , that permits us to construct the CDF using Calculus, like this:

$$F(b) - F(a) = \int_a^b f(x) dx$$

$f(x)$  is referred to as the *probability density function*. The intuition is that even for continuous outcomes we can define tiny intervals, that are almost as small as points, that have positive probabilities. If we think of the size of these intervals as the base of a rectangle, the probability density function  $f$  determines the height of the rectangle so that the summing up of the area of these rectangles approximate the probability  $(F(b) - F(a))$ . This sum can be written as Riemann sum that is approximated by an integral:

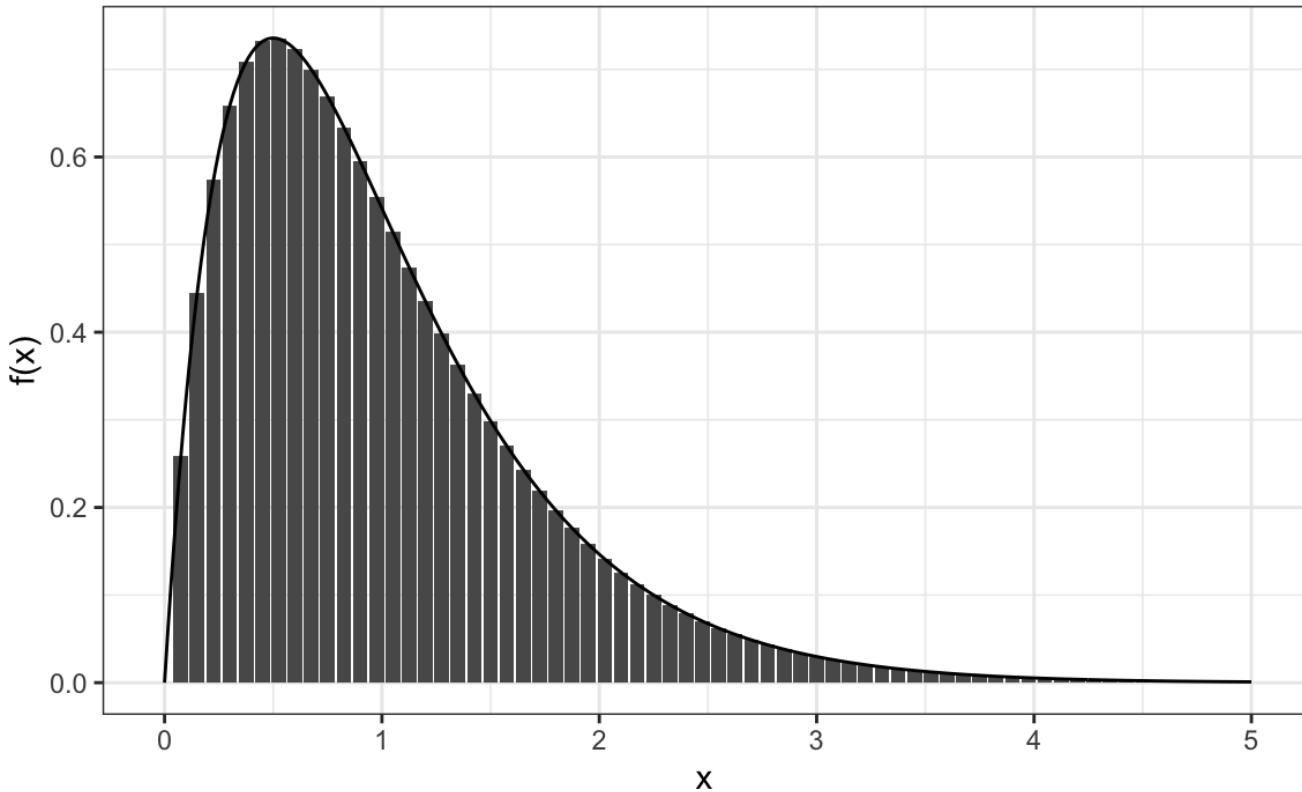


Figure 1:

An example of such a continuous distribution is the normal distribution. As we saw in Section 1.5, the probability density function is given by:

$$f(x) = e^{-\frac{1}{2} \left( \frac{x-\mu}{\sigma} \right)^2}$$

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  $\mu$  and standard deviation  $\sigma$  if its probability distribution is defined by:

$$F(a) = pnorm(a, \mu, \sigma)$$

This is useful because, if we are willing to use the normal approximation for, let's 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
```

### 4.3 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. Below is a plot of that probability distribution:

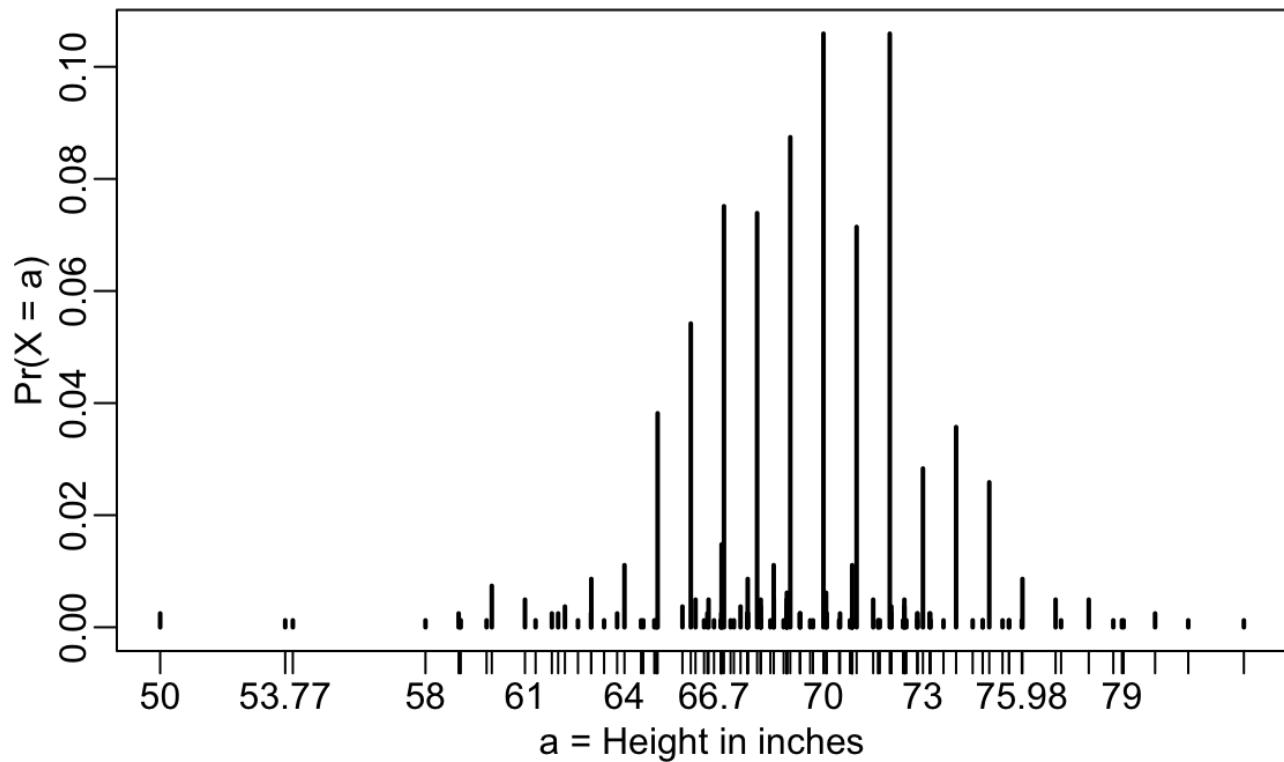


Figure 2:

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 0.0012315 or 1 in 812. The probability for 70 inches is much higher at 0.1059113, 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 instance, 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 therefore 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.

## 4.4 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  $\backslash(X\backslash)$ , can be 1, 2, 3, 4, 5 or 6. The probability of 4 is defined as:

$$\backslash[\ \backslash mbox{Pr}\{X=4\} = 1/6 \ \backslash]$$

The CDF can then easily be defined:

$$\backslash[ F(4) = \backslash mbox{Pr}\{X \leq 4\} = \backslash mbox{Pr}\{X = 4\} + \backslash mbox{Pr}\{X = 3\} + \backslash mbox{Pr}\{X = 2\} + \backslash mbox{Pr}\{X = 1\} \ \backslash]$$

Although for continuous distributions the probability of a single value  $\backslash(\backslash mbox{Pr}\{X=x\}\backslash)$  is not defined, there is a theoretical definition that has a similar interpretation. The probability density at  $\backslash(x\backslash)$  is defined as the function  $\backslash(f(a)\backslash)$  such that:

$$\backslash[ F(a) = \backslash mbox{Pr}\{X \leq a\} = \backslash int_{-\infty}^a f(x), dx \ \backslash]$$

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  $\backslash(f(x)\backslash)$  as a curve for which the area under that curve, up to the value  $\backslash(a\backslash)$ , gives you the probability  $\backslash(\backslash mbox{Pr}\{X \leq a\}\backslash)$ .

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`.

While it may not be immediately apparent why knowing about probability densities is useful, understanding this concept is essential for individuals aiming to fit models to data for which predefined functions are not available.

## 4.5 Monte Carlo

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)

```

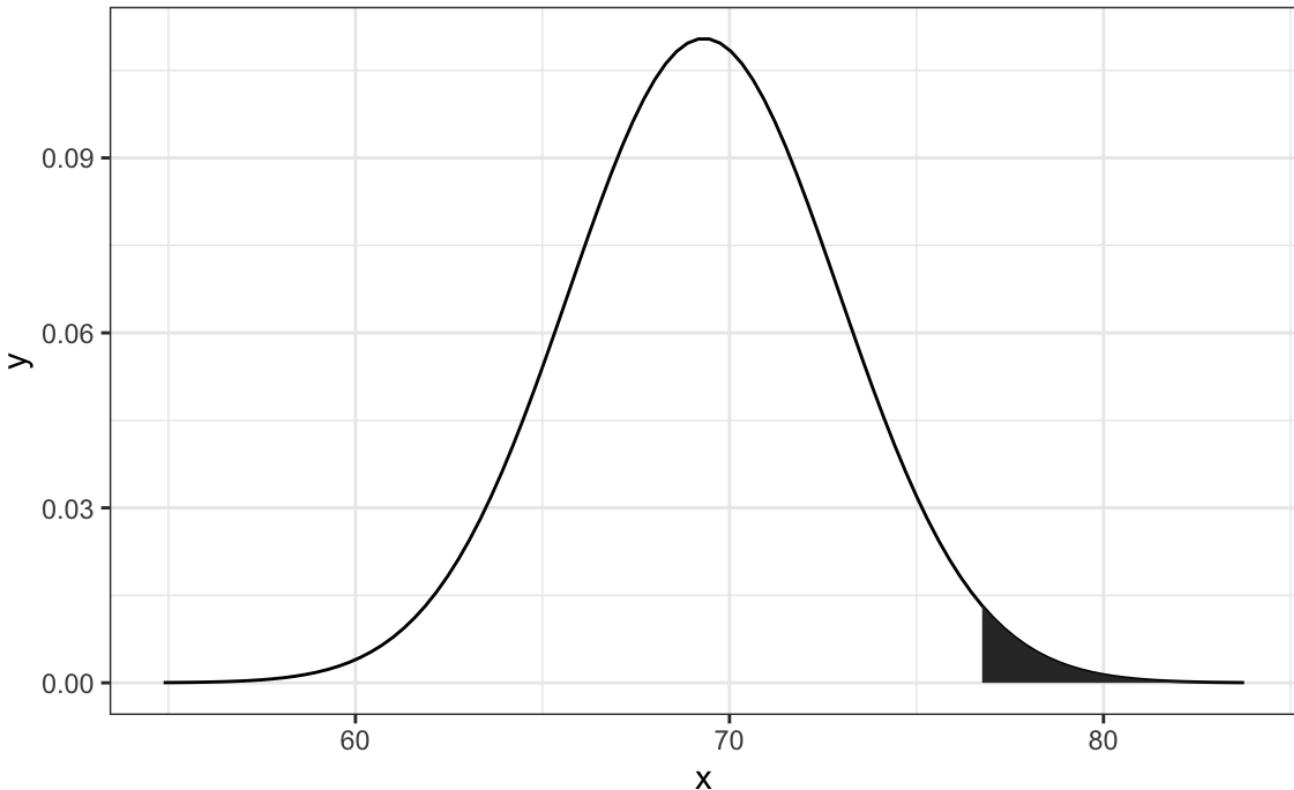


Figure 3:

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] 0.0191
```

Here is the resulting distribution:

Note that it does not look normal.

## 4.6 Continuous distributions

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:

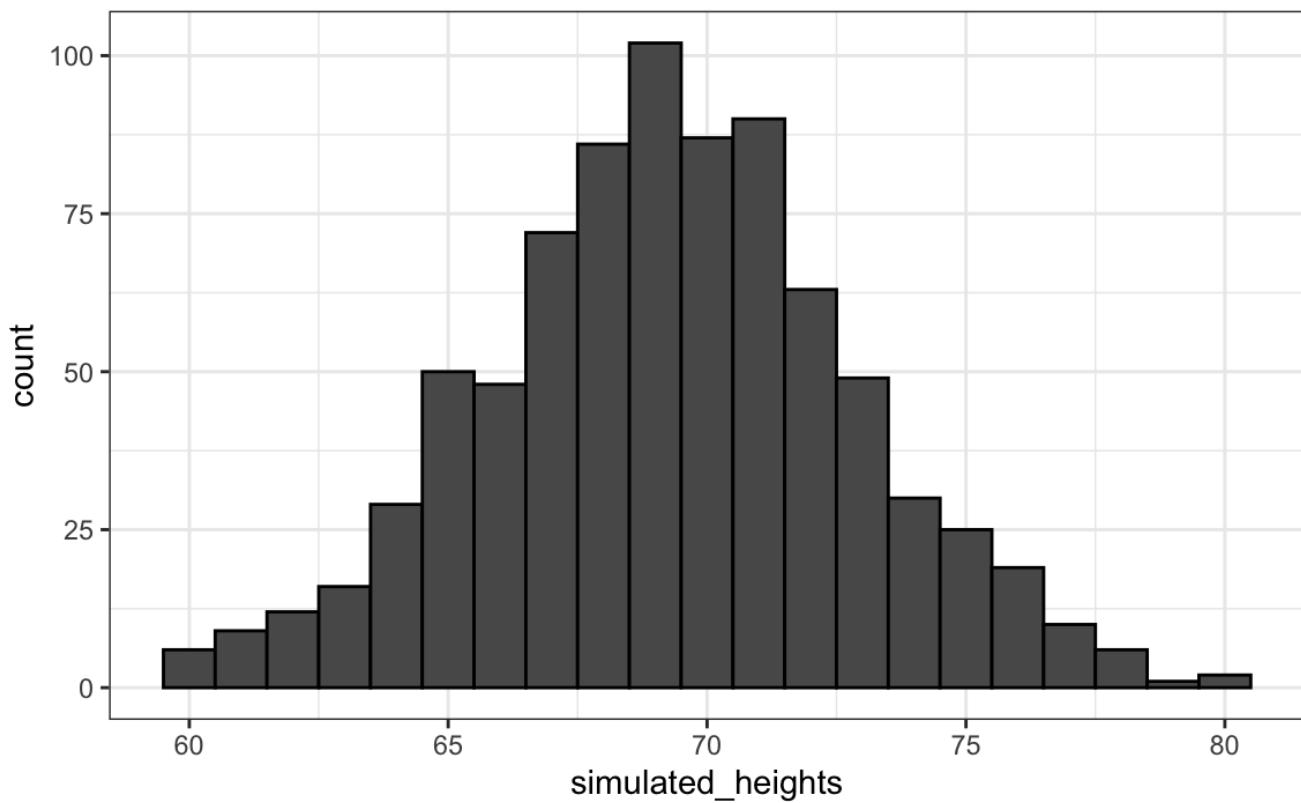


Figure 4:

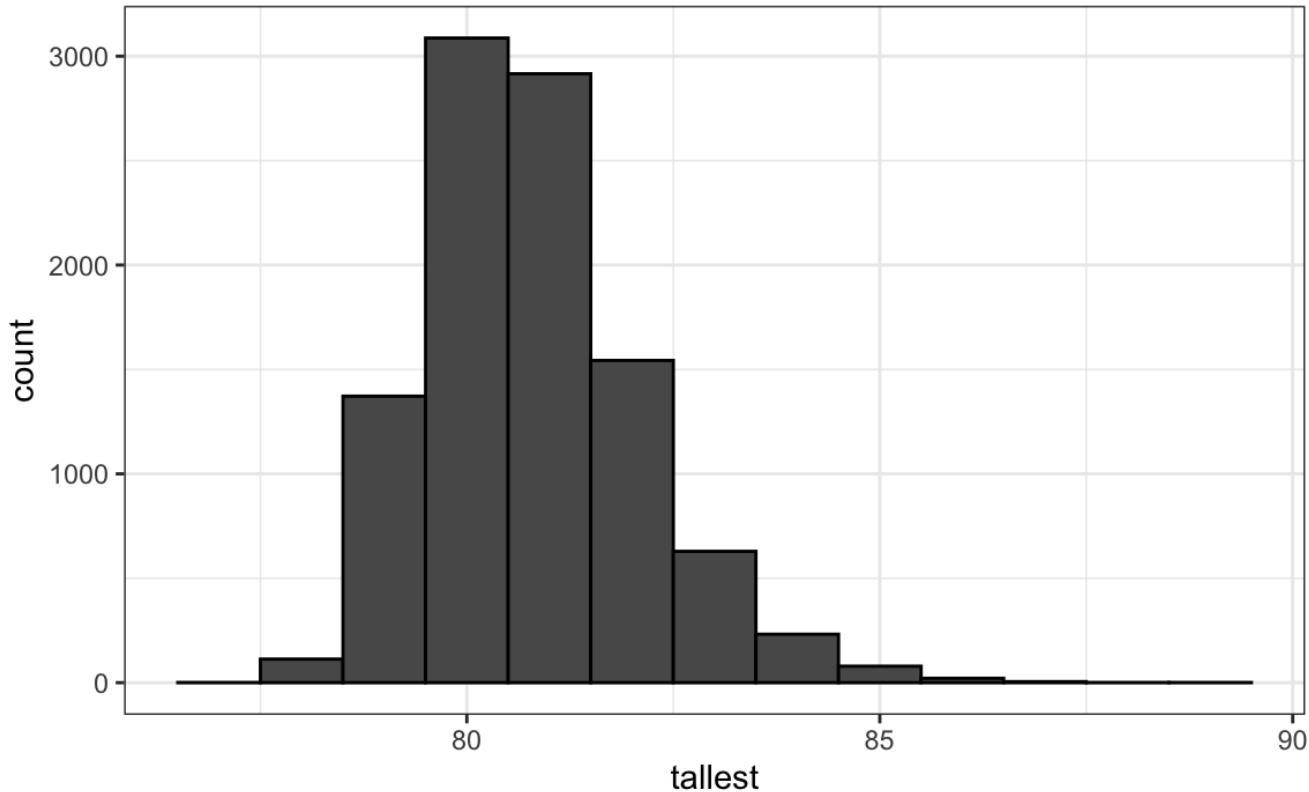


Figure 5:

```

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 11.2.3, 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.

## 4.7 Exercises

- 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?
- 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?
- 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?
- 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?
- Notice that the answer to the question does not change when you change units. This makes sense since the standard deviations from the average for an entry in a list are not affected by what units we use. In fact, if you look closely, you notice that 61 and 67 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.
- To understand the mathematical rationale that explains why the answers to exercises 3, 4, and 5 are the same, suppose we have a random variable with average  $\langle m \rangle$  and standard error  $\langle s \rangle$ . Suppose we ask the probability of  $\langle X \rangle$  being smaller or equal to  $\langle a \rangle$ . Remember that, by definition,  $\langle a \rangle$  is  $\langle (a - m)/s \rangle$  standard deviations  $\langle s \rangle$  away from the average  $\langle m \rangle$ . The probability is:

$$\Pr\{X \leq a\}$$

Now we subtract  $\langle m \rangle$  to both sides and then divide both sides by  $\langle s \rangle$ :

$$\Pr\left\{\frac{X - \mu}{\sigma} \leq \frac{a - \mu}{\sigma}\right\}$$

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  $\langle Z \rangle$ :

$$\Pr\{Z \leq \frac{a - \mu}{\sigma}\}$$

So, no matter the units, the probability of  $\Pr\{X \leq a\}$  is the same as the probability of a standard normal variable being less than  $\Pr\{Z \leq \frac{a - \mu}{\sigma}\}$ . 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 - mu)/sigma)`
  - c. `pnorm((a - mu)/sigma, mu, sigma)`
  - d. `pnorm(a)`
- 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`.
  - 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.

# Introduction to Data Science - 5 Random variables

Rafael A. Irizarry

In data science, we often work with data that is affected by chance, whether it comes from a random sample, is subject to measurement error, or 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 accomplishing this. The first step is learning 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 required government bailouts to avoid complete closure.

## 5.1 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, changing randomly each time we select a new bead. 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
```

Sometimes it's 1 and sometimes it's 0.

## 5.2 Sampling models

Many data generation procedures, those that produce the data we study, can be effectively modeled 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 codes 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 outcomes 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 be modeled by draws from an urn, reflecting the way individuals are assigned into group; when getting assigned, individuals draw their group at random. Sampling models are therefore ubiquitous in data science. Casino games offer a plethora of

real-world cases in which sampling models are used to answer specific questions. We will therefore start with these 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 available 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 decide against installing roulette wheels.

We are going to define a random variable  $\langle S \rangle$  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, resulting in a draw a -\$1. Otherwise, the casino wins a dollar, and we draw a \$1. To construct our random variable  $\langle S \rangle$ , 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
```

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**, as it involves modeling the random behavior of roulette through the sampling of draws from an urn. The total winnings  $\langle S \rangle$  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
```

### 5.3 The probability distribution of a random variable

If you run the code above, you see that  $\langle S \rangle$  changes every time. This is, of course, because  $\langle S \rangle$  is a **random variable**. The probability distribution of a random variable informs us about the probability of the observed value falling at any given interval. For example, if we want to know the probability that we lose money, we are asking the probability that  $\langle S \rangle$  is in the interval  $\langle S < 0 \rangle$ .

Keep in mind that if we can define a cumulative distribution function  $\langle F(a) \rangle = \Pr\{\langle S \rangle \leq a\}$ , then we will be able to answer any question related to the probability of events defined by our random variable  $\langle S \rangle$ , including the event  $\langle S < 0 \rangle$ . We call this  $\langle F \rangle$  the random variable's *distribution function*.

We can estimate the distribution function for the random variable  $\langle S \rangle$  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 repeatedly play roulette, specifically  $\langle B = 10,000 \rangle$  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  $\Pr(S < 0)$ , allowing us to 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  $\Pr(S)$  by creating a histogram showing the probability  $\Pr(F(b) - F(a))$  for several intervals  $((a,b])$ :

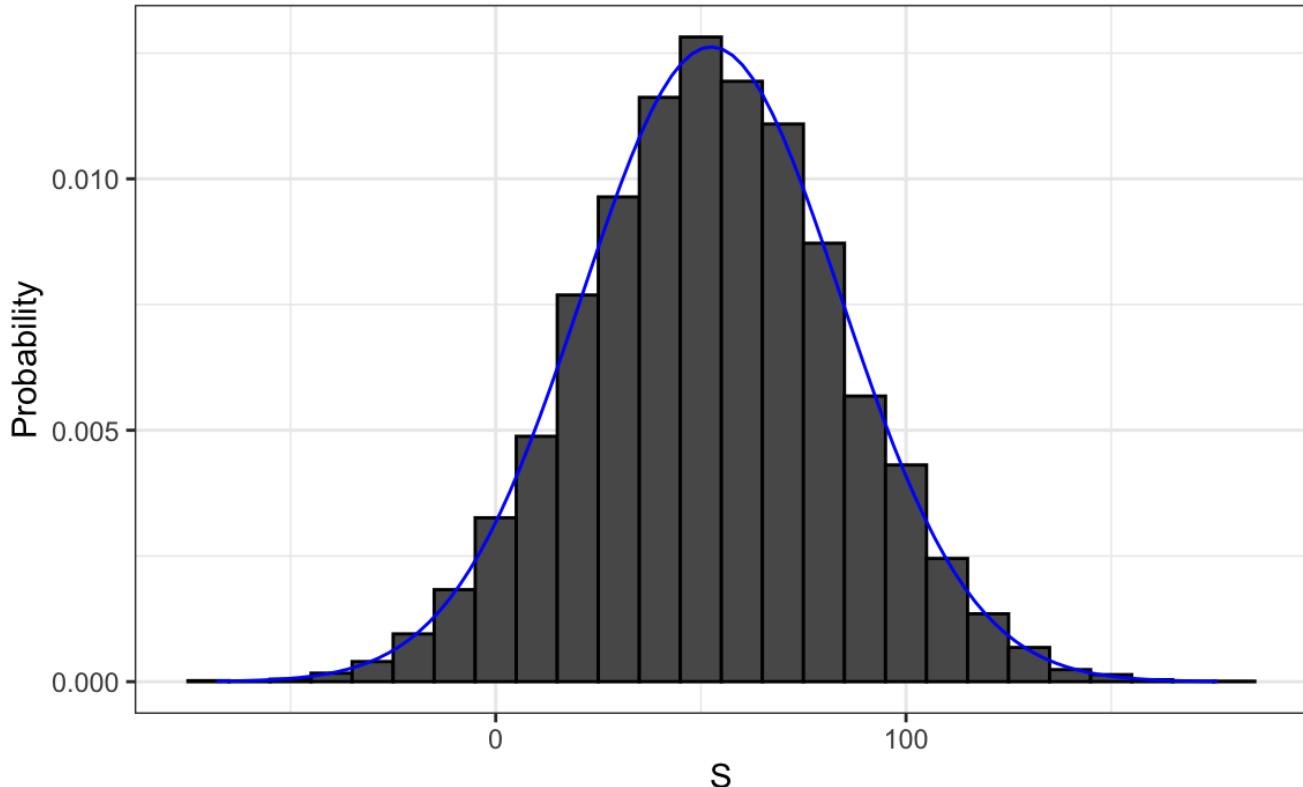


Figure 1:

We see that the distribution appears to be approximately normal. A qqplot will confirm that the normal approximation is close to a perfect approximation for this distribution. In fact, if the distribution is normal, all we need to define it are the average and the standard deviation. Since 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 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  $\Pr(S)$ . More details on these concepts will be provided in the next section.

Statistical theory offers a method to derive the distribution of random variables defined as the sum of independent random draw of numbers from an urn. Specifically, in our example above, we can demonstrate that  $\Pr((S+n)/2)$  follows a binomial distribution. We therefore do not need to run Monte Carlo simulations to determine the probability distribution of  $\Pr(S)$ . The simulations were conducted 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
```

Since this is a discrete probability function, to obtain  $\Pr(S < 0)$  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>.

We do not delve into these details here. Instead, we will explore an incredibly useful approximation provided by mathematical theory, which generally applies to sums and averages of draws from any urn: the Central Limit Theorem (CLT).

## 5.4 Distributions versus probability distributions

Before we continue, let's establish an important distinction and connection between the distribution of a list of numbers and a probability distribution. 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 indicates what proportion of the list is less than or equal to  $a$ . Given their usefulness as summaries when the distribution is approximately normal, we also define the average and standard deviation. These are determined with a straightforward operation involving the vector containing the list of numbers, denoted as  $x$ :

```
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 containing numbers, then there exists 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 without involving an urn is by running a Monte Carlo simulation and generating a very large list of outcomes of  $(X)$ . These outcomes form a list of numbers, and 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.

## 5.5 Notation for random variables

In statistical textbooks, upper case letters denote random variables, and we will adhere to this convention. Lower case letters are used for observed values. You will see some notation that include both. For example, you will see events defined as  $(X \leq x)$ . Here  $(X)$  is a random variable and  $(x)$  is an arbitrary value and not 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. In this case, the probability of  $(X=x)$  is  $1/6$  regardless of the observed value  $(x)$ .

This notation may seem a bit strange because when we inquire about probability,  $(X)$  is not an observed quantity; it's a random quantity that we will encounter in the future. We can discuss what we expect  $(X)$  to be, what values are probable, but we can't discuss what value  $(X)$  is. Once we have data, we do see a realization of  $(X)$ . Therefore, data analysts often speak of what could have been after observing what actually happened.

## 5.6 The expected value and standard error

We have described sampling models for draws. We will now review the mathematical theory that allows us to 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 proportions, 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 represent it with the letter  $(E)$  like this:

$\|\mbox{E}[X]\|$

to denote the expected value of the random variable  $(X)$ .

A random variable will vary around its expected value in a manner that if you take the average of many, many draws, the average will approximate the expected value. This approximation improves as you take more draws, making the expected value a useful quantity to compute.

For discrete random variable with possible outcomes  $(x_1, \dots, x_n)$ , the expected value is defined as:

$\|\mbox{E}[X] = \sum_{i=1}^n x_i \Pr(X = x_i)\|$  If  $(X)$  is a continuous random variable with a range of values  $(a)$  to  $(b)$  and a probability density function  $(f(x))$ , this sum transforms into an integral:

$\|\int_a^b x f(x) dx\|$

Note that in the case that we are picking values from an urn, and each value  $(x_i)$  has an equal chance  $(1/n)$  of being selected, the above equation is simply the average of the  $(x_i)$ s.

$\|\mbox{E}[X] = \frac{1}{n} \sum_{i=1}^n x_i\|$

In the urn used to model betting on red in roulette, we have 20 one-dollar bills and 18 negative one-dollar bills, so the expected value is:

$\|\mbox{E}[X] = (20 + -18)/38\|$

which is about 5 cents. You might consider it a bit counterintuitive to say that  $(X)$  varies around 0.05 when it only takes the values 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:

$\|\mbox{E}[X] = ap + b(1-p)\|$

To confirm this, observe 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,  $(n \times p + n \times b \times (1-p))$ , 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. This, in turn, 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 the number of draws  $(\times)$  the average of the numbers in the urn.

Therefore, if 1,000 people play roulette, the casino expects to win, on average, about 1,000  $(\times)$  \$0.05 = \$50. However, 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:

$\|\mbox{SE}[X]\|$

to denote the standard error of a random variable.

For discrete random variable with possible outcomes  $(x_1, \dots, x_n)$ , the standard error is defined as:

$\|\mbox{SE}[X] = \sqrt{\sum_{i=1}^n (x_i - \mbox{E}[X])^2 \Pr(X = x_i)}\|$  which you can think of as the expected *average distance* of  $(X)$  from the expected value.

If  $(X)$  is a continuous random variable, with range of values  $(a)$  to  $(b)$  and probability density function  $(f(x))$ , this sum turns into an integral:

$\|\sqrt{\int_a^b (x - \mbox{E}[X])^2 f(x) dx}\|$

Note that in the case that we are picking values from an urn where each value  $(x_i)$  has an equal chance  $(1/n)$  of being selected, the above equation is simply the standard deviation of the  $(x_i)$ s.

$\sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$  with  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ . 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:

$$\sqrt{p(1-p)}$$

So in our roulette example, the standard deviation of the values inside the urn is  $\sqrt{10/19 \times 9/19}$  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 obtain either 1 or -1, with 1 slightly favored over -1.

A widely used mathematical result is that **if our draws are independent**, then the *standard error of the sum* is given by the equation:

$$\sqrt{n} \times \text{standard deviation of the numbers in the urn}$$

Using this formula, 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 to install more roulette wheels. But we still haven't answered the question: How likely is the casino to lose money? The CLT will help in this regard.

The exact probability for the casino winnings can be computed precisely, rather than approximately, using the binomial distribution. However, here we focus on the CLT, which can be applied more broadly to sums of random variables in a way that the binomial distribution cannot.

## 5.7 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. Given that 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  $\sum(S)$  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
```

### 5.7.1 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, much larger sample sizes are needed.

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 explore the properties of the Poisson distribution using `dpois` and `ppois`. You can generate random variables following this distribution with `rpois`. However, we won’t cover the theory here. You can learn about the Poisson distribution in any probability textbook and even Wikipedia<sup>4</sup>.

## 5.8 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:

$$\mathbb{E}[X_1 + X_2 + \dots + X_n] = \mathbb{E}[X_1] + \mathbb{E}[X_2] + \dots + \mathbb{E}[X_n]$$

If  $(X)$  represents independent draws from the urn, then they all have the same expected value. Let’s denote the expected value with  $(\mu)$  and rewrite the equation as:

$$\mathbb{E}[X_1 + X_2 + \dots + X_n] = n\mu$$

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:

$$\mathbb{E}[aX] = a\mathbb{E}[X]$$

To understand why this is intuitive, consider changing units. If we change the units of a random variable, such as 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, denoted as  $\mu$  again:

$$\mathbb{E}[(X_1 + X_2 + \dots + X_n)/n] = \mathbb{E}[X_1 + X_2 + \dots + X_n]/n = n\mu/n = \mu$$

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:

$$\mathbb{E}[\text{SE}(X_1 + X_2 + \dots + X_n)] = \sqrt{\mathbb{E}[X_1]^2 + \mathbb{E}[X_2]^2 + \dots + \mathbb{E}[X_n]^2}$$

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:

$$\mathbb{E}[\text{SE}(aX)] = a \mathbb{E}[\text{SE}(X)]$$

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  $\sigma$ :

$$\mathbb{E}[\text{SE}((X_1 + X_2 + \dots + X_n)/n)] = \mathbb{E}[\text{SE}(X_1 + X_2 + \dots + X_n)/n] = \sqrt{\mathbb{E}[X_1]^2 + \mathbb{E}[X_2]^2 + \dots + \mathbb{E}[X_n]^2}/n = \sqrt{\sigma^2 + \sigma^2 + \dots + \sigma^2}/n = \sqrt{n\sigma^2}/n = \sigma/\sqrt{n}$$

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.

The assumption of independence is important

Let's make the explanation more concise and clear:

The given equation reveals crucial insights for practical scenarios. Specifically, it suggests that the standard error can be minimized by increasing the sample size,  $n$ , and we can quantify this reduction. However, this principle holds true only when the variables  $(X_1, X_2, \dots, X_n)$  are independent. If they are not, the estimated standard error can be significantly off.

In Section 14.2, we introduce the concept of correlation, which quantifies the degree to which variables are interdependent. If the correlation coefficient among the  $(X)$  variables is  $\rho$ , the standard error of their average is:

$$\text{SE}(\bar{X}) = \sigma \sqrt{\frac{1 + (n-1)\rho}{n}}$$

The key observation here is that as  $\rho$  approaches its upper limit of 1, the standard error increases. Notably, in the situation where  $\rho = 1$ , the standard error,  $\text{SE}(\bar{X})$ , equals  $\sigma$ , and it becomes unaffected by the sample size  $n$ .

### 5.8.1 Law of large numbers

An important implication of result 5 above 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.

### Misinterpretation of the 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. Yet these events are independent so the chance of a coin landing heads is 50%, regardless of the previous 5. The same principle applies to 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.

## 5.9 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  $\backslash(X\backslash)$  for your winnings. Hint: Refer to 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  $\backslash(X\backslash)$ .

4. Compute the standard error of  $\backslash(X\backslash)$ .

5. Now create a random variable  $\backslash(S\backslash)$  that is the sum of your winnings after betting on green 1000 times. Hint: change the argument `size` and `replace` in your answer to exercise 2. Start your code by setting the seed to 1 with `set.seed(1)`.

6. What is the expected value of  $\backslash(S\backslash)$ ?

7. What is the standard error of  $\backslash(S\backslash)$ ?

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  $\backslash(S\backslash)$ . 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?

- 1,000 simulations is not enough. If we do more, they match.
- 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.
- The difference is within rounding error.
- The CLT only works for averages.

12. Now create a random variable  $\backslash(Y\backslash)$  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  $\backslash(Y\backslash)$ ?

14. What is the standard error of  $\backslash(Y\backslash)$ ?

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  $\backslash(Y\backslash)$ . Compute the average and standard deviation of the resulting list to confirm the results of 13 and 14. Start your code by setting the seed to 1 with `set.seed(1)`.

17. Now compare your answer to 15 using the Monte Carlo result.
18. The Monte Carlo result and the CLT approximation are now much closer. What could account for this?
- We are now computing averages instead of sums.
  - 2,500 Monte Carlo simulations is not better than 1,000.
  - The CLT works better when the sample size is larger. We increased from 1,000 to 2,500.
  - It is not closer. The difference is within rounding error.
19. More complex versions of the sampling models we have discussed are also used by banks to determine interest rates and insurance companies to determine premiums. To understand this, suppose you run a small bank that has a history of identifying potential homeowners that can be trusted to make payments. In fact, historically, only 2% of your customers default in a given year, meaning that they don't pay back the money that you lent them. Suppose your bank will give out  $n = \$1,000$  loans for \$180,000 this year. Also, after adding up all costs, suppose your bank loses  $(l) = \$200,000$  per foreclosure. For simplicity, we assume this includes all operational costs. What is the expected profit  $(S)$  for your bank under this scenario?
20. Note that the total loss defined by the final sum in the previous exercise is a random variable. Every time you run the sampling model code, you obtain a different number of people defaulting which results in a different loss. Code a sampling model for the random variable representing your bank's profit  $(S)$  under scenario described in 19.
21. The previous exercise demonstrates that if you simply loan money to everybody without interest, you will end up losing money due to the 2% that defaults. Although you know 2% of your clients will probably default, you don't know which ones, so you can't remove them. 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. What quantity  $(x)$  would you have to charge each borrower so that your bank's expected profit is 0? Assume that you don't get  $(x)$  from the borrowers that default. Also, note  $(x)$  is not the interest rate, but the total you add meaning  $(x/180000)$  is the *interest rate*.
22. Rewrite the sample model from exercise 20 and run a Monte Carlo simulation to get an idea of the distribution of your profit when you charge interest rates.
23. We don't actually need a Monte Carlo simulation. Based on what we have learned, the CLT informs us that, since our losses are a sum of independent draws, its distribution is approximately normal. What are the expected value and standard errors of the profit  $(S)$ ? Write these as functions of the probability of foreclosure  $(p)$ , the number of loans  $(n)$ , the loss per foreclosure  $(l)$ , and the quantity you charge each borrower  $(x)$ .
24. If you set  $(x)$  to assure your bank breaks even (expected profit is 0), what is the probability that your bank loses money?
25. Suppose that if your bank has negative profit, it has to close. Therefore, you need to increase  $(x)$  to minimize this risk. However, setting the interest rates too high may lead your clients to choose another bank. 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? Hint: We want  $(\Pr(S < 0) = 0.01)$ . Note that you can add subtract constants to both side of an inequality, and the probability does not change:  $(\Pr(S < 0) = \Pr(S+k < 0+k))$ . Similarly, with division of positive constants:  $(\Pr(S+k < 0+k) = \Pr((S+k)/m < k/m))$ . Use this fact and the CLT to transform the left side of the inequality in  $(\Pr(S < 0))$  into a standard normal.
26. Our interest rate now increases. But it is still a very competitive interest rate. For the  $(x)$  you obtained in 25, what is expected profit per loan and the expected total profit?
27. Run a Monte Carlo simulation to double check the theoretical approximation used in 25 and 26.
28. One of your employees points out that, since the bank is making a profit per loan, the bank should give out more loans! Why limit it to 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. The employee 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. Suppose the default probability is twice as high, or 4%, and you set the interest rate to 5%, or  $(x = \$9,000)$ , what is your expected profit per loan?
29. How much do we have to increase  $(n)$  by to assure the probability of losing money is still less than 0.01?
30. Confirm the result in exercise 29 with a Monte Carlo simulation.

31. According to this equation, giving out more loans increases your expected profit and lowers the chances of losing money! Giving out more loans 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, the movies "The Big Short" and "Margin Call" are made, recounting the mistake your friend, and many others, made. What happened?

Your colleague's scheme was mainly based on this mathematical formula  $(\text{SE}) = \sigma / \sqrt{n}$ . 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.

To construct a more realistic simulation than the original one your colleague ran, let's assume there is a global event affecting everybody with high-risk mortgages and altering their probability simultaneously. We will assume that with a 50-50 chance all the default probabilities slightly increase or decrease to somewhere between 0.03 and 0.05. However, this change occurs universally, impacting everybody at once, not just one person. As these draws are no longer independent, our equation for the standard error of the sum of random variables does not apply. Write a Monte Carlo simulation for your total profit with this model.

32. Use the simulation results to report the expected profit, the probability of losing money, and the probability of losing more than \$10,000,000. Study the distribution of profit and discuss how making the wrong assumption lead to a catastrophic result.

- 
1. [https://en.wikipedia.org/w/index.php?title=Financial\\_crisis\\_of\\_2007&oldid=932008](https://en.wikipedia.org/w/index.php?title=Financial_crisis_of_2007&oldid=932008)
  2. [https://en.wikipedia.org/w/index.php?title=Security\\_\(finance\)](https://en.wikipedia.org/w/index.php?title=Security_(finance))
  3. [https://en.wikipedia.org/w/index.php?title=Binomial\\_distribution](https://en.wikipedia.org/w/index.php?title=Binomial_distribution)
  4. [https://en.wikipedia.org/w/index.php?title=Poisson\\_distribution](https://en.wikipedia.org/w/index.php?title=Poisson_distribution)

# Introduction to Data Science - Statistical inference

Rafael A. Irizarry

Statistical Inference is the branch of statistics dedicated to distinguishing patterns arising from signal versus those arising from chance. It is a broad topic and, in this section, we review the basics using polls as a motivating example. To illustrate the concepts, we supplement mathematical formulas with Monte Carlo simulations and R code. We motivate the concepts with election forecasting as a case study.

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, many 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 or cards being dealt somewhere?

In this part of the book, we will demonstrate how the probability concepts covered in the previous part can be applied to develop statistical approaches that render polls effective tools. 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 straightforward example to introduce the main concepts of statistical inference. Forecasting an election is a more complex process that involves combining results from 50 states and DC. We will delve into this subject in the last chapter, after we cover all the basic concepts. Specifically, we will learn the statistical concepts necessary to define *estimates* and *margins of errors* for the popular vote, and show how these are used to construct *confidence intervals*. Once we grasp these ideas, we will be able to understand *statistical power* and *p-values*, concepts that are ubiquitous in, for example, the academic literature. We will then aggregate data from different pollsters to highlight the shortcomings of the models used by traditional pollsters and present a method for improving these models. To understand probabilistic statements about the chances of a candidate winning, we will introduce *Bayesian modeling*. Finally, we put it all together using *hierarchical models* to recreate the simplified version of the FiveThirtyEight model and apply it to the 2016 election.

---

1. <https://www.youtube.com/watch?v=TbKkjm-gheY>

# Introduction to Data Science - 6 Parameters and Estimates

Rafael A. Irizarry

Opinion polling has been conducted since the 19th century. The general aim is to describe the opinions held by a specific population on a given set of topics. In recent times, these polls have been pervasive in the US during presidential elections. Polls are useful when interviewing every member of a specific population is logically impossible. The general strategy involves interviewing a smaller, randomly chosen group and then inferring the opinions of the entire population from those of this subset. Statistical theory, known as *inference*, is used to justify the process and is the primary focus of this part of the book.

Perhaps the best known opinion polls are those conducted to determine which candidate is preferred by voters in a given election. Political strategists extensively use polls to decide, among other things, where to allocate resources, such as determining the geographical locations to focus their “get out the vote” efforts.

Elections are a particularly interesting instances of opinion polls because reveal the actual opinion of the entire population on election day. Of course, it costs millions of dollars to run an real election, which makes polling a cost-effective strategy for those seeking to forecast the results. In addition to strategist, news organizations are also interested in forecasting elections due to the apparent demand for what they reveal.

## 6.1 The sampling model for polls

We start by connecting probability theory to the task of using polls to learn about a population.

Although typically the results of polls run by political candidates are kept private, polls are also conducted by news organizations because results tend to be of interest to the general public and made public. We will eventually be looking at these public datasets.

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>:

| Poll         | Date          | Sample   | MoE | Clinton | Trump | Spread       |
|--------------|---------------|----------|-----|---------|-------|--------------|
| RCP Average  | 10/31 - 11/7  | --       | --  | 47.2    | 44.3  | Clinton +2.9 |
| Bloomberg    | 11/4 - 11/6   | 799 LV   | 3.5 | 46.0    | 43.0  | Clinton +3   |
| Economist    | 11/4 - 11/7   | 3669 LV  | --  | 49.0    | 45.0  | Clinton +4   |
| IBD          | 11/3 - 11/6   | 1026 LV  | 3.1 | 43.0    | 42.0  | Clinton +1   |
| 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   |
| CBS News     | 11/2 - 11/6   | 1426 LV  | 3.0 | 47.0    | 43.0  | Clinton +4   |
| LA Times     | 10/31 - 11/6  | 2935 LV  | 4.5 | 43.0    | 48.0  | Trump +5     |
| NBC News     | 11/3 - 11/5   | 1282 LV  | 2.7 | 48.0    | 43.0  | Clinton +5   |
| NBC News     | 10/31 - 11/6  | 30145 LV | 1.0 | 51.0    | 44.0  | Clinton +7   |
| McClatchy    | 11/1 - 11/3   | 940 LV   | 3.2 | 46.0    | 44.0  | Clinton +2   |
| Reuters      | 10/31 - 11/4  | 2244 LV  | 2.2 | 44.0    | 40.0  | Clinton +4   |
| GravisGravis | 10/31 - 10/31 | 5360 RV  | 1.3 | 50.0    | 50.0  | Tie          |

Let’s make some observations about the table above. First, observe that different polls, all conducted days before the election, report different *spreads*: the estimated difference between support for the two candidates. Notice that

the reported spreads hover around what eventually became the actual result: Clinton won the popular vote by 2.1%. Additionally, we see a column titled **MoE** which stands for *margin of error*.

To help us understand the connection between polls and what we have learned, let's construct a situation similar to what pollsters face. To simulate the challenge pollsters encounter in terms of competing with other pollsters for media attention, we will use an urn filled with 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 reflect the fact that running polls is expensive, it costs you \$0.10 for each bead you sample. Therefore, if your sample size is 250, and you win, you will break even since you would have paid \$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 receive half what you paid and proceed 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. In this model, the beads inside the urn represent individuals who will vote on election day. The red beads represent those voting for the Republican candidate, while the blue beads represent the Democrats. For simplicity, let's assume there are no other colors; that is, that there are just two parties: Republican and Democratic.

## 6.2 Populations, samples, parameters, and estimates

We want to predict the proportion of blue beads in the urn. Let's call this quantity  $\hat{p}$ , which then tells us the proportion of red beads  $1 - \hat{p}$ , and the spread  $\hat{p} - (1 - \hat{p})$ , which simplifies to  $2\hat{p} - 1$ .

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 goal of statistical inference is to predict the parameter  $p$  based on 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  $\hat{p}$  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  $\hat{p}$  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.

Observe 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 improve it.

### 6.2.1 The sample average

Conducting an opinion poll is being modeled as taking a random sample from an urn. We propose using 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  $2\hat{p} - 1$ . However, for simplicity, we will illustrate the concepts for estimating  $p$ . We will use our knowledge of probability to justify our use of the sample proportion and to quantify its proximity to the population proportion  $p$ .



Figure 1:

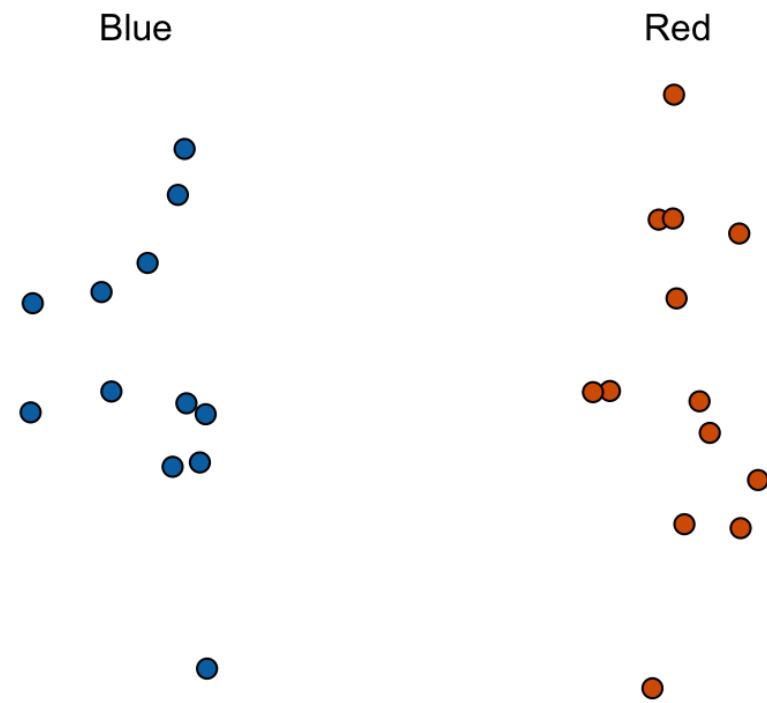


Figure 2:



Figure 3:

We start by defining the random variable  $\langle X \rangle$  as  $\langle X=1 \rangle$ , if we pick a blue bead at random, and  $\langle X=0 \rangle$  if it is red. This implies that the population is a list of 0s and 1s. If we sample  $\langle N \rangle$  beads, then the average of the draws  $\langle X_1, \dots, X_N \rangle$  is equivalent to the proportion of blue beads in our sample. This is because adding the  $\langle X \rangle$ s is equivalent to counting the blue beads, and dividing this count by the total  $\langle N \rangle$  is equivalent to computing a proportion. We use the symbol  $\langle \bar{X} \rangle$  to represent this average. In statistics textbooks, a bar on top of a symbol typically denotes the average. The theory we just covered about the sum of draws becomes useful because the average is a sum of draws multiplied by the constant  $\langle 1/N \rangle$ :

$$\langle \bar{X} \rangle = \frac{1}{N} \sum_{i=1}^N X_i$$

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? Firstly, we know that the expected value of the sum of draws is  $\langle N \rangle$  times the average of the values in the urn. We know that the average of the 0s and 1s in the urn must be  $\langle p \rangle$ , the proportion of blue beads.

Here, we encounter an important difference compared to what we did in the section on probability: we don't know the composition of the urn. While we know there are blue and red beads, we don't know how many of each. This is what we want to find out: we are trying to **estimate**  $\langle p \rangle$ .

### 6.2.2 Parameters

Just as we use variables to define unknowns in systems of equations, in statistical inference, we define *parameters* to represent unknown parts of our models. In the urn model, which we are using to simulate an opinion poll, we do not know the proportion of blue beads in the urn. We define the parameters  $\langle p \rangle$  to represent this quantity. Since our main goal is determining  $\langle p \rangle$ , we are going to *estimate this parameter*.

Introductory statistics textbooks usually use the population average as the first example of a parameter. Note that in our example the parameter of interest  $\langle p \rangle$  is defined by the proportion of 1s (blue) and 0s (red) in the urn, which is also the average of the numbers in the urn. Our parameter of interest can therefore be thought of as a population average.

The concepts presented here on how we estimate parameters, and provide insights into how good these estimates are, extend to many data analysis tasks. For example, we may want to determine the difference in health improvement between patients receiving treatment and a control group, investigate the health effects of smoking on a population, analyze the differences in racial groups of fatal shootings by police, or assess 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.

## 6.3 Polling versus forecasting

Before we continue, it's important to clarify a practical issue related to forecasting an election. If a poll is conducted four months before the election, it is estimating the  $\langle p \rangle$  for that moment, and not for election day. The  $\langle p \rangle$  for election night might be different, as people's opinions tend to fluctuate through time. Generally, the polls conducted the night before the election tend to be the most accurate, since opinions do not change significantly in a day. However, forecasters try to develop tools that model how opinions vary over time and aim to predict the election night results by taking into consideration these fluctuations. We will explore some approaches for doing this in a later section.

## 6.4 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  $\langle \bar{X} \rangle$ . Remember that  $\langle \bar{X} \rangle$  is the sum of independent draws so the rules we covered in the probability chapter apply.

Applying the concepts we have learned, the expected value of the sum  $\langle N \bar{X} \rangle$  is  $\langle N \times p \rangle$  the average of the urn, denoted as  $\langle p \rangle$ . Dividing by the non-random constant  $\langle N \rangle$  yields the expected value of the average  $\langle \bar{X} \rangle$  as  $\langle p \rangle$ . We can write it using our mathematical notation:

$$\langle \bar{X} \rangle = p$$

We can also use what we learned to determine 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 it is  $\sqrt{N} \times \sqrt{p(1-p)} = \sqrt{N} \times \sqrt{p(1-p)}$ . Because we are dividing the sum by  $N$ , we arrive at the following formula for the standard error of the average:

$$\text{SE}(\bar{X}) = \sqrt{p(1-p)/N}$$

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  $N$ :

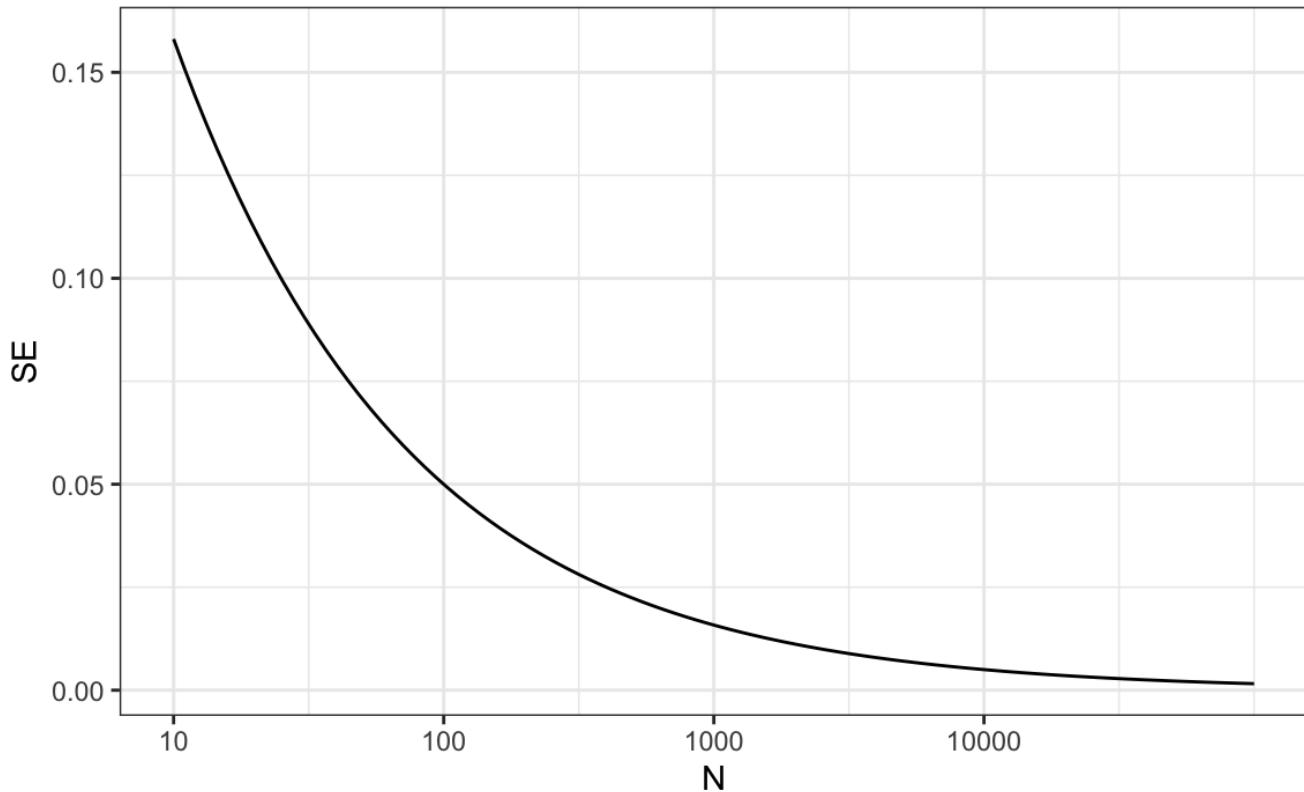


Figure 4:

The plot shows that we would need a poll of over 10,000 people to achieve a standard error that low. We rarely see polls of this size due in part to the associated costs. According to the Real Clear Politics table, 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 Chapter 7.

## 6.5 Exercises

1. Suppose you poll a population in which a proportion  $\langle(p)\rangle$  of voters are Democrats and  $\langle(1-p)\rangle$  are Republicans. Your sample size is  $\langle(N=25)\rangle$ . Consider the random variable  $\langle(S)\rangle$ , 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  $\langle(p)\rangle$ .
2. What is the standard error of  $\langle(S)\rangle$ ? Hint: It's a function of  $\langle(p)\rangle$ .
3. Consider the random variable  $\langle(S/N)\rangle$ . This is equivalent to the sample average, which we have been denoting as  $\langle(\bar{X})\rangle$ . What is the expected value of the  $\langle(\bar{X})\rangle$ ? Hint: It's a function of  $\langle(p)\rangle$ .
4. What is the standard error of  $\langle(\bar{X})\rangle$ ? Hint: It's a function of  $\langle(p)\rangle$ .
5. Write a line of code that gives you the standard error `se` for the problem above for several values of  $\langle(p)\rangle$ , 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  $\langle(N=25)\rangle$ ,  $\langle(N=100)\rangle$ , and  $\langle(N=1000)\rangle$ .
7. If we are interested in the difference in proportions,  $\langle(\mu = p - (1-p))\rangle$ , our estimate is  $\langle(\hat{\mu})\rangle = \bar{X} - (1-\bar{X})$ . Use the rules we learned about sums of random variables and scaled random variables to derive the expected value of  $\langle(\hat{\mu})\rangle$ .
8. What is the standard error of  $\langle(\hat{\mu})\rangle$ ?
9. If the actual  $\langle(p=.45)\rangle$ , it means the Republicans are winning by a relatively large margin, since  $\langle(\mu = -.1)\rangle$ , which is a 10% margin of victory. In this case, what is the standard error of  $\langle(2\hat{\mu}-1)\rangle$  if we take a sample of  $\langle(N=25)\rangle$ ?
10. Given the answer to exercise 9, which of the following best describes your strategy of using a sample size of  $\langle(N=25)\rangle$ ?
  - a. The expected value of our estimate  $\langle(2\bar{X}-1)\rangle$  is  $\langle(\mu)\rangle$ , so our prediction will be accurate.
  - b. Our standard error is larger than the difference, so the chances of  $\langle(2\bar{X}-1)\rangle$  representing a large margin are not 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  $\langle(p)\rangle$ , we have no way of knowing that making  $\langle(N)\rangle$  larger would actually improve our standard error.

- 
1. <http://www.realclearpolitics.com>
  2. [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)

# Introduction to Data Science - 7 Central Limit Theorem

Rafael A. Irizarry

The CLT tells us that the distribution function for a sum of draws is approximately normal. Additionally, we have learned that dividing a normally distributed random variable by a constant results in another 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 section: subtract the expected value and divide by the standard error to obtain a standard normal random variable, which we'll denote as  $Z$ , on the left. Since  $p$  is the expected value and  $\sqrt{p(1-p)/N}$  is the standard error, we get:

$\Pr(Z \leq \frac{.01}{\sqrt{p(1-p)/N}}) - \Pr(Z \leq -\frac{.01}{\sqrt{p(1-p)/N}})$

One problem we have is that since we don't know  $p$ , we don't know  $\sqrt{p(1-p)/N}$ . However, 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
```

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 simply as 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  $\bar{p}$ , we get:

$\Pr(Z \leq \bar{p} + 1.96\hat{SE}(\bar{p}) / \hat{SE}(\bar{p})) - \Pr(Z \leq \bar{p} - 1.96\hat{SE}(\bar{p}) / \hat{SE}(\bar{p}))$  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  $\hat{p}$  will be within  $1.96\hat{SE}(\bar{p})$ , in our case within about 0.2, of  $\bar{p}$ . Observe 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, consider that if we had obtained a  $\hat{p}=0.48$  with a sample size of 2,000, our standard error  $\hat{SE}(\bar{p})$  would have been 0.0111714. 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 lead us to believe 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.

## 7.1 A Monte Carlo simulation

Suppose we want to use a Monte Carlo simulation to corroborate the tools we have developed 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, that we don't know  $p$ . We could construct an urn, similar to the one pictured above, and conduct an analog simulation (without a computer). While time-consuming, we could take 10,000 samples, count the beads, and track 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.

Therefore, one approach we can use to corroborate theoretical results is to pick one or several values of  $p$  and run 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  $(\sqrt{p(1-p)/N}) = 0.0157321$ . The simulation confirms this:

```
mean(x_hat)
#> [1] 0.45
sd(x_hat)
#> [1] 0.0157321
```

A histogram and qqplot confirm that the normal approximation is also accurate:

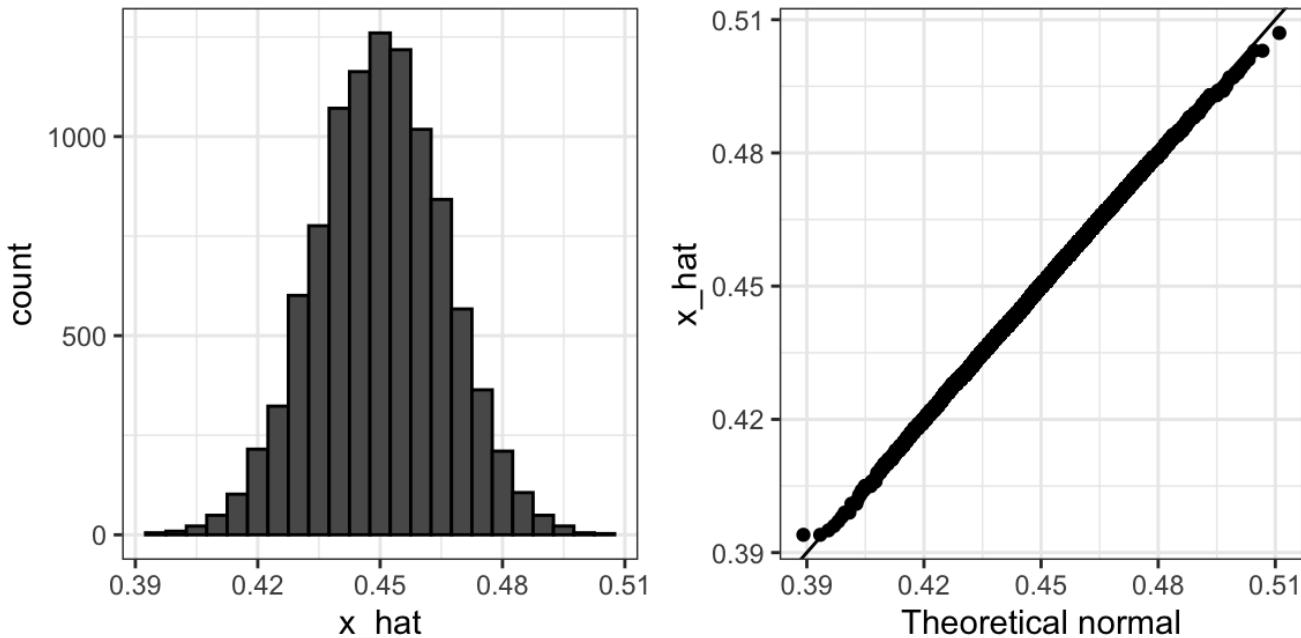


Figure 1:

Of course, in real life, we would never be able to run such an experiment because we don't know  $(p)$ . However, we can 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 rerunning the code above after changing the values of  $p$  and  $N$ .

## 7.2 The spread

The objective of the competition is to predict the spread, not the proportion  $(p)$ . However, since we are assuming there are only two parties, we know that the spread is  $(\mu = p - (1-p) = 2p - 1)$ . As a result, everything we have done can easily be adapted to an estimate of  $(\mu)$ . Once we have our estimate  $(\bar{X})$  and  $(\hat{\text{SE}}(\bar{X}))$ , we estimate the spread with  $(2\bar{X} - 1)$  and, since we are multiplying by 2, the standard error is  $(2\hat{\text{SE}}(\bar{X}))$ . 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, this is not a very useful sample size. Nevertheless, the point is that, once we have an estimate and standard error for  $(p)$ , we have it for the spread  $(\mu)$ .

We use  $(\mu)$  to denote the spread here and in the next sections because this is the typical notation used in statistical textbooks for the parameter of interest. The reason we use  $(\mu)$  is that a population mean is often the parameter of interest, and  $(\mu)$  is the Greek letter for  $m$ .

### 7.3 Bias: Why not run a very large poll?

For realistic values of  $\langle p \rangle$ , let's say ranging from 0.35 to 0.65, if we conduct a very large poll with 100,000 people, theory tells us that we would predict the election perfectly, as the largest possible margin of error is around 0.3%:

```
#> Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```

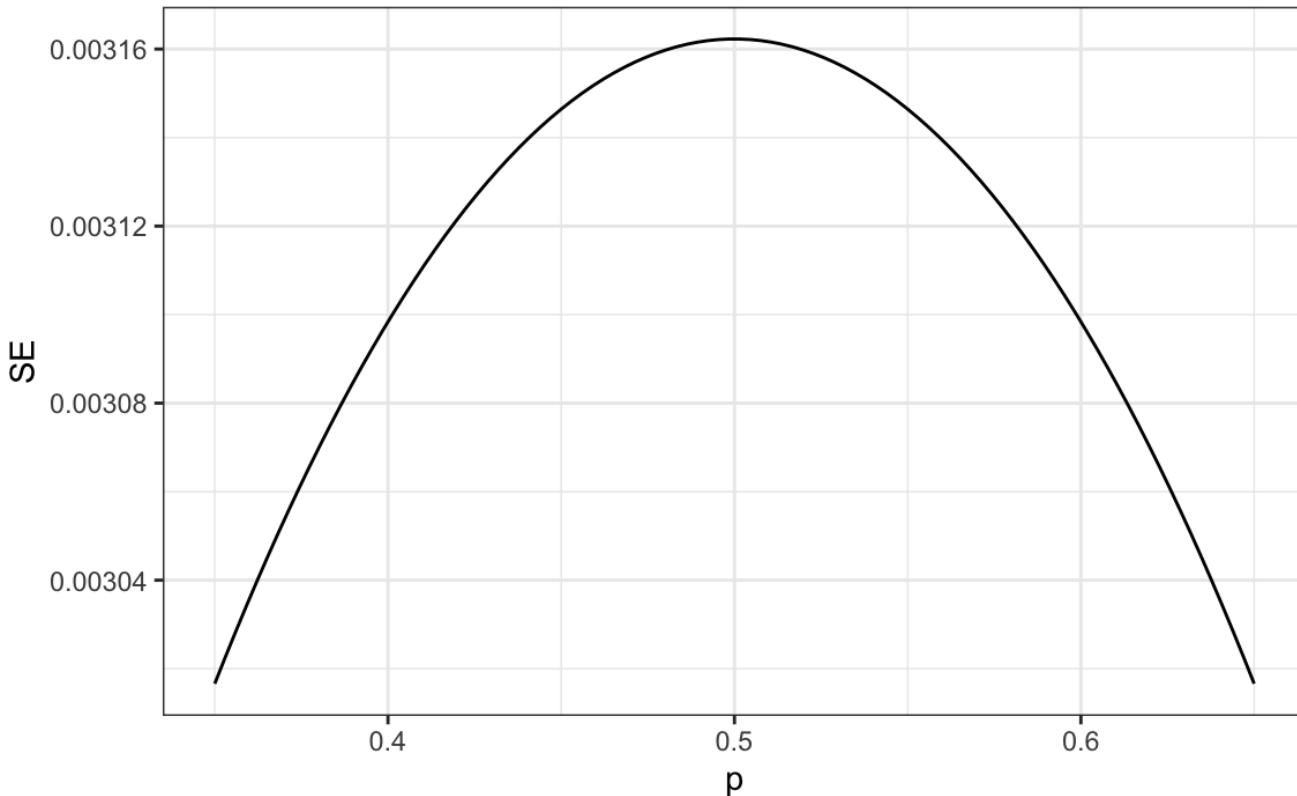


Figure 2:

One reason is that conducting such a poll is very expensive. Another, and possibly more important reason, is that theory has its limitations. Polling is much more complicated than simply picking beads from an urn. Some people might lie to pollsters, and others might not have phones. However, perhaps the most important way an actual poll differs from an urn model is that we don't actually 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  $\langle p \rangle$ . We call this bias. Historically, we observe that polls are indeed biased, although not by a substantial amount. The typical bias appears to be about 1-2%. This makes election forecasting a bit more interesting, and we will explore how to model this in a later section.

### 7.4 Exercises

1. Write an *urn model* function that takes the proportion of Democrats  $\langle p \rangle$  and the sample size  $\langle N \rangle$  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  $\langle N=100 \rangle$ . 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  $\langle p \rangle$  and our estimate  $\langle \bar{X} \rangle$ . 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  $\lvert(\bar{X} - p)\rvert$  is a random variable. In practice, the error is not observed because we do not know  $\lvert(p)\rvert$ . Here, we observe it since we constructed the simulation. What is the average size of the error if we define the size by taking the absolute value  $\lvert(\mid \bar{X} - p \mid)\rvert$ ?
5. The standard error is related to the typical **size** of the error we make when predicting. 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  $\lvert(\bar{X})\rvert$ . What does theory tell us is the standard error of  $\lvert(\bar{X})\rvert$  for a sample size of 100?
7. In practice, we don't know  $\lvert(p)\rvert$ , so we construct an estimate of the theoretical prediction based by plugging in  $\lvert(\bar{X})\rvert$  for  $\lvert(p)\rvert$ . 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  $\lvert(p)\rvert$  with  $\lvert(\bar{X})\rvert$ . 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  $\lvert(p=0.5)\rvert$ . Create a plot of the largest standard error for  $\lvert(N)\rvert$  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%?
- a. 100
  - b. 500
  - c. 2,500
  - d. 4,000
9. For sample size  $\lvert(N=100)\rvert$ , the Central Limit Theorem tells us that the distribution of  $\lvert(\bar{X})\rvert$  is:
- a. practically equal to  $\lvert(p)\rvert$ .
  - b. approximately normal with expected value  $\lvert(p)\rvert$  and standard error  $\lvert(\sqrt{p(1-p)/N})\rvert$ .
  - c. approximately normal with expected value  $\lvert(\bar{X})\rvert$  and standard error  $\lvert(\sqrt{\lvert(\bar{X})(1-\bar{X})/N})\rvert$ .
  - d. not a random variable.
10. Based on the answer from exercise 8, the error  $\lvert(\bar{X} - p)\rvert$  is:
- a. practically equal to 0.
  - b. approximately normal with expected value  $\lvert(0)\rvert$  and standard error  $\lvert(\sqrt{p(1-p)/N})\rvert$ .
  - c. approximately normal with expected value  $\lvert(p)\rvert$  and standard error  $\lvert(\sqrt{p(1-p)/N})\rvert$ .
  - d. 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  $\lvert(p=0.45)\rvert$  and  $\lvert(N=100)\rvert$  as in exercise 2, use the CLT to estimate the probability that  $\lvert(\bar{X}) > 0.5\rvert$ . Assume you know  $\lvert(p=0.45)\rvert$  for this calculation.
13. Assume you are in a practical situation and you don't know  $\lvert(p)\rvert$ . Take a sample of size  $\lvert(N=100)\rvert$  and obtain a sample average of  $\lvert(\bar{X}) = 0.51\rvert$ . What is the CLT approximation for the probability that your error is equal to or larger than 0.01?

# Introduction to Data Science - 8 Confidence intervals

Rafael A. Irizarry

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`. Below is an example using a temperature dataset available in R:

**Average Yearly Temperatures in New Haven**

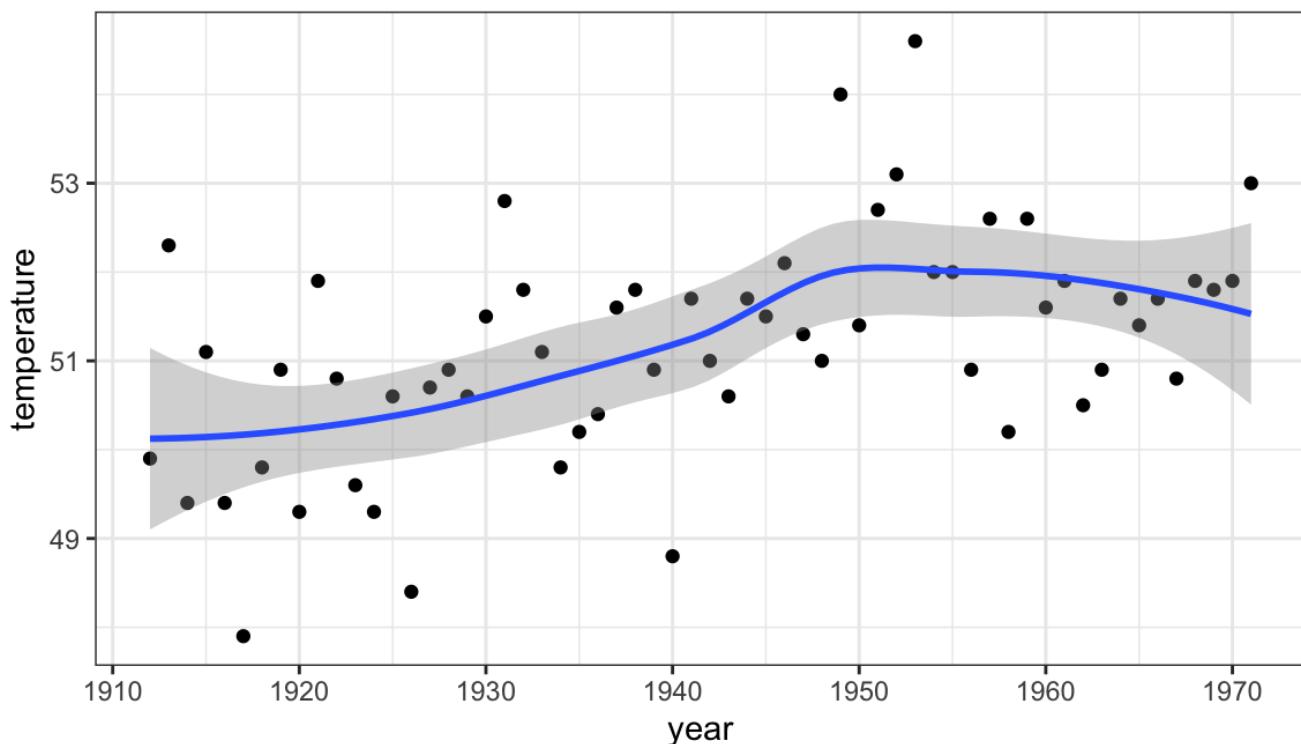


Figure 1:

In the Machine Learning section, 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  $\langle p \rangle$ , you receive half the money you spent on your “poll” back and proceed 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  $\langle [0,1] \rangle$  is guaranteed to include  $\langle p \rangle$ . 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  $\bar{p}$ . If we are asked to create an interval with, say, a 95% chance of including  $\bar{p}$ , 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 now see how this works mathematically.

We want to know the probability that the interval  $(\bar{X} - 2\hat{SE}, \bar{X} + 2\hat{SE})$  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] 0.427 0.489
```

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] 0.467 0.529
```

Keep sampling and creating intervals, and you will see the random variation.

To determine the probability that the interval includes  $\bar{p}$ , we need to compute the following:

$$\Pr(\bar{X} - 1.96\hat{SE} \leq p \leq \bar{X} + 1.96\hat{SE})$$

By subtracting and dividing the same quantities in all parts of the equation, we find that the above is equivalent to:

$$\Pr(-1.96 \leq \frac{\bar{X} - p}{\hat{SE}} \leq 1.96)$$

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 probability, not just 0.95 and 0.99. In statistics textbooks, confidence interval formulas are given for arbitrary probabilities written as  $\sqrt{1-\alpha}$ . We can obtain the  $\sqrt{z}$  for the equation above using `z = qnorm(1 - alpha / 2)` because  $\sqrt{(1 - \alpha/2)^2} = 1 - \sqrt{\alpha}$ . So, for example, for  $\sqrt{\alpha=0.05}$ ,  $\sqrt{1 - \alpha/2} = 0.975$  and we get the  $\sqrt{z=1.96}$  we used above:

```
qnorm(0.975)
#> [1] 1.96
```

## 8.1 A Monte Carlo simulation

We can run a Monte Carlo simulation to confirm that, in fact, a 95% confidence interval includes  $\sqrt{p}$  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:

When applying the theory we described above, it's important to remember that it's the intervals that are random, not  $\sqrt{p}$ . In the plot above, we can see the random intervals moving around, while the proportion of blue beads in the urn  $\sqrt{p}$ , represented with the vertical line, remains in the same place. So the 95% relates to the probability that the random interval falls on top of  $\sqrt{p}$ . Stating that  $\sqrt{p}$  has a 95% chance of being between this or that is technically incorrect because  $\sqrt{p}$  is not random.

## 8.2 Exercises

For these exercises, we will use actual polls from the 2016 election. You can load the data from the `dslabs` package.

```
library(dslabs)
```

Specifically, we will use all the national polls that ended within one week prior to 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$sample_size[1]
x_hat <- polls$rawpoll_clinton[1]/100
```

Assume there are only two candidates and construct a 95% confidence interval for the election night proportion  $\sqrt{p}$ .

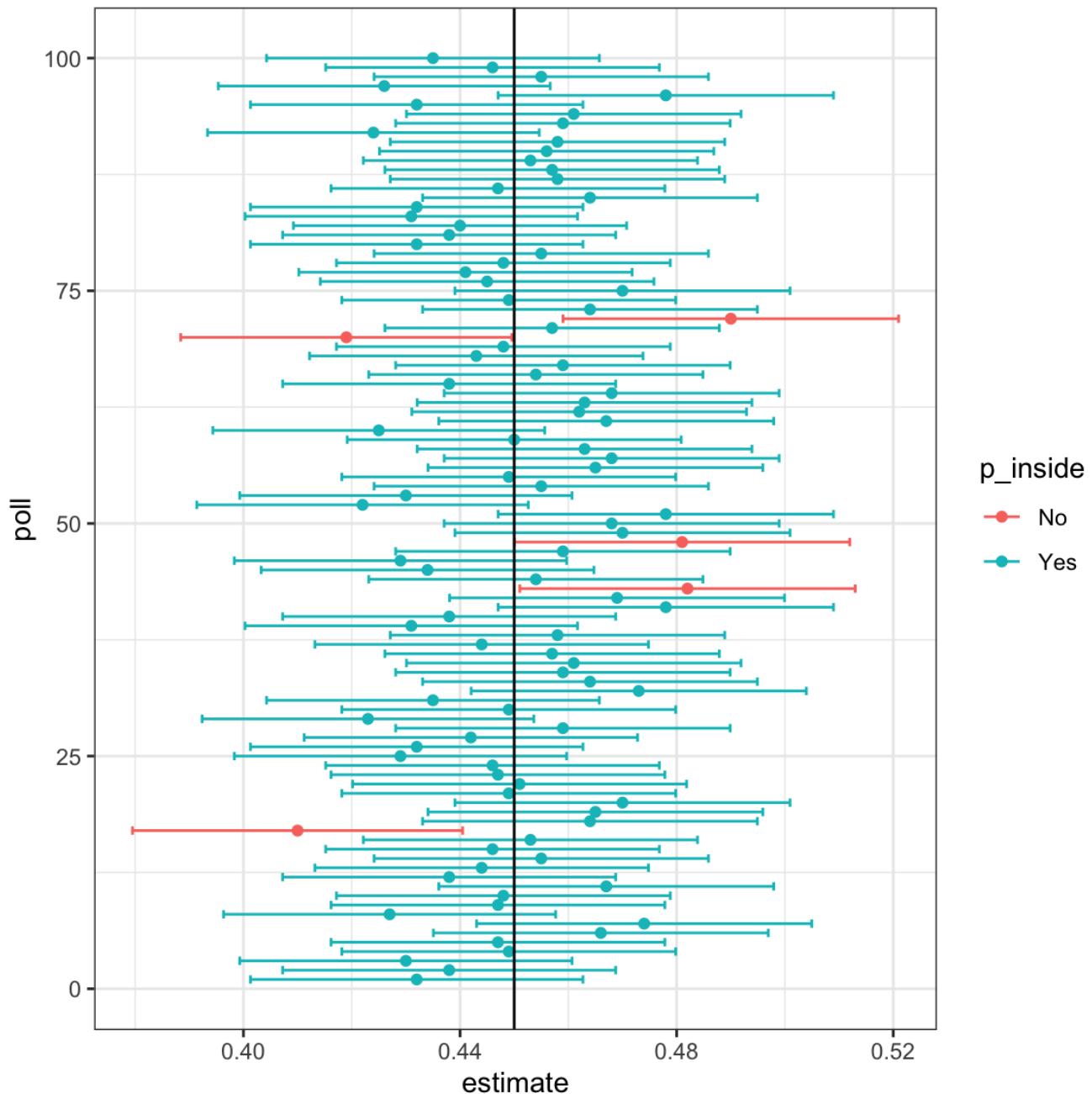


Figure 2:

2. Now use `dplyr` to add a confidence interval as two columns, call them `lower` and `upper`, to the object `poll1`. 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  $\backslash(p=0.482\backslash)$  or not.
4. For the table you just created, what proportion of confidence intervals included  $\backslash(p\backslash)$ ?
5. If these confidence intervals are constructed correctly, and the theory holds up, what proportion should include  $\backslash(p\backslash)$ ?
6. A much smaller proportion of the polls than expected produce confidence intervals containing  $\backslash(p\backslash)$ . If you look closely at the table, you will see that most polls that fail to include  $\backslash(p\backslash)$  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,  $\backslash(\mu\backslash)$ , which in this election was  $\backslash(0.482 - 0.461 = 0.021\backslash)$ . Assume that there are only two parties and that  $\backslash(\mu = 2p - 1\backslash)$ . 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(mu_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 increases 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  $\backslash(mu=0.021\backslash)$ . Stratify by pollster.
10. Redo the plot that you made for exercise 9, but only for pollsters that took five or more polls.

# Introduction to Data Science - 9 Hypothesis testing

Rafael A. Irizarry

In scientific studies, you'll often see phrases like "the results are statistically significant". This points to a technique called hypothesis testing, where we use p-values, a type of probability, to test our initial assumption or hypothesis.

In hypothesis testing, rather than providing an estimate of the parameter we're studying, we provide a probability that serves as evidence supporting or contradicting a specific hypothesis. The hypothesis usually involves whether a parameter is different from a predetermined value (often 0).

Hypothesis testing is used when you can phrase your research question in terms of whether a parameter differs from this predetermined value. It's applied in various fields, asking questions such as: Does a medication extend the lives of cancer patients? Does an increase in gun sales correlate with more gun violence? Does class size affect test scores?

Take, for instance, the previously used example with colored beads. We might not be concerned about the exact proportion of blue beads, but instead ask: Are there more blue beads than red ones? This could be rephrased as asking if the proportion of blue beads is more than 0.5.

The initial hypothesis that the parameter equals the predetermined value is called the "null hypothesis". It's popular because it allows us to focus on the data's properties under this null scenario. Once data is collected, we estimate the parameter and calculate the p-value, which is the probability of the estimate being as extreme as observed if the null hypothesis is true. If the p-value is small, it indicates the null hypothesis is unlikely, providing evidence against it.

We will see more examples of hypothesis testing in Chapter 17.

## 9.1 p-values

Suppose we take a random sample of  $(N=100)$  and we observe  $(52)$  blue beads, which gives us  $(\bar{X} = 0.52)$ . This seems to be pointing to the existence of more blue than red beads since 0.52 is larger than 0.5. However, we know there is chance involved in this process and we could get a 52 even when the actual  $(p=0.5)$ . We call the assumption that  $(p = 0.5)$  a *null hypothesis*. The null hypothesis is the skeptic's hypothesis.

We have observed a random variable  $(\bar{X} = 0.52)$ , 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? If the p-value is small enough, we *reject the null hypothesis* and say that the results are *statistically significant*.

The p-value of 0.05 as a threshold for statistical significance is conventionally used in many areas of research. A cutoff of 0.01 is also used to define *highly significance*. The choice of 0.05 is somewhat arbitrary and was popularized by the British statistician Ronald Fisher in the 1920s. We do not recommend using these cutoff without justification and recommend avoiding the phrase *statistically significant*.

To obtain a p-value for our example, we write:

$$\Pr(|\bar{X} - 0.5| > 0.02)$$

assuming the  $(p=0.5)$ . Under the null hypothesis we know that:

$$|\sqrt{N}(\bar{X} - 0.5)| \sim N(0, 1)$$

is standard normal. We, therefore, can compute the probability above, which is the p-value.

$$\Pr\left(|\sqrt{N}(\bar{X} - 0.5)| > \sqrt{0.02}\right) = \Pr\left(Z > \sqrt{0.02}\right)$$

```

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 because 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.

We can show mathematically that if a  $((1-\alpha)\times 100)\%$  confidence interval does not contain the null hypothesis value, the null hypothesis is rejected with a p-value as smaller or equal than  $\alpha$ . So *statistical significance* can be determined from confidence intervals. However, unlike the confidence interval, the p-value does not provide an estimate of the magnitude of the effect. For this reason, we recommend avoiding p-values whenever you can compute a confidence interval.

## 9.2 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

```

included 0. If this were a poll and we were forced to make a declaration, we would have to say it was a “toss-up”.

One problem with our poll results is that, given the sample size and the value of  $p$ , we would have to sacrifice 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 thus, have a much better chance of detecting the direction of the spread.

## 9.3 Exercises

- Generate a sample of size  $(N=1000)$  from an urn model with 50% blue beads:

```

N <- 1000
p <- 0.5
x <- rbinom(N, 1, 0.5)

```

then, compute a p-value to test if  $(p=0.5)$ . Repeat this 10,000 times and report how often the p-value is lower than 0.05? How often is it lower than 0.01?

- Make a histogram of the p-values you generated in exercise 1. Which of the following seems to be true?
  - The p-values are all 0.05.
  - The p-values are normally distributed; CLT seems to hold.
  - The p-values are uniformly distributed.
  - The p-values are all less than 0.05.
- Demonstrate, mathematically, why see the histogram we see in exercise 2.
- Generate a sample of size  $(N=1000)$  from an urn model with 52% blue beads:

```
N <- 1000
p <- 0.52
x <- rbinom(N, 1, 0.5)
```

Compute a p-value to test if  $\text{p}=0.5$ . Repeat this 10,000 times and report how often the p-value is larger than 0.05? Note that you are computing  $1 - \text{power}$ .

5. Repeat exercise for but for the following values:

```
values <- expand.grid(N = c(25, 50, 100, 500, 1000), p = seq(0.51, 0.75, 0.01))
```

Plot power as a function of  $N$  with a different color curve for each value of  $p$ .

# Introduction to Data Science - 10 Bootstrap

Rafael A. Irizarry

CLT provides an useful approach to building confidence intervals and performing hypothesis testing. However, it does not always apply. Here we provide a short introduction to an alternative approach to estimating the distribution of an estimate that does not rely on CLT.

## 10.1 Example: median income

Suppose the income distribution of your population is as follows:

```
set.seed(1995)
n <- 10^6
income <- 10^(rnorm(n, log10(45000), log10(3)))
hist(income/10^3, nclass = 1000)
```

**Histogram of income/10<sup>3</sup>**

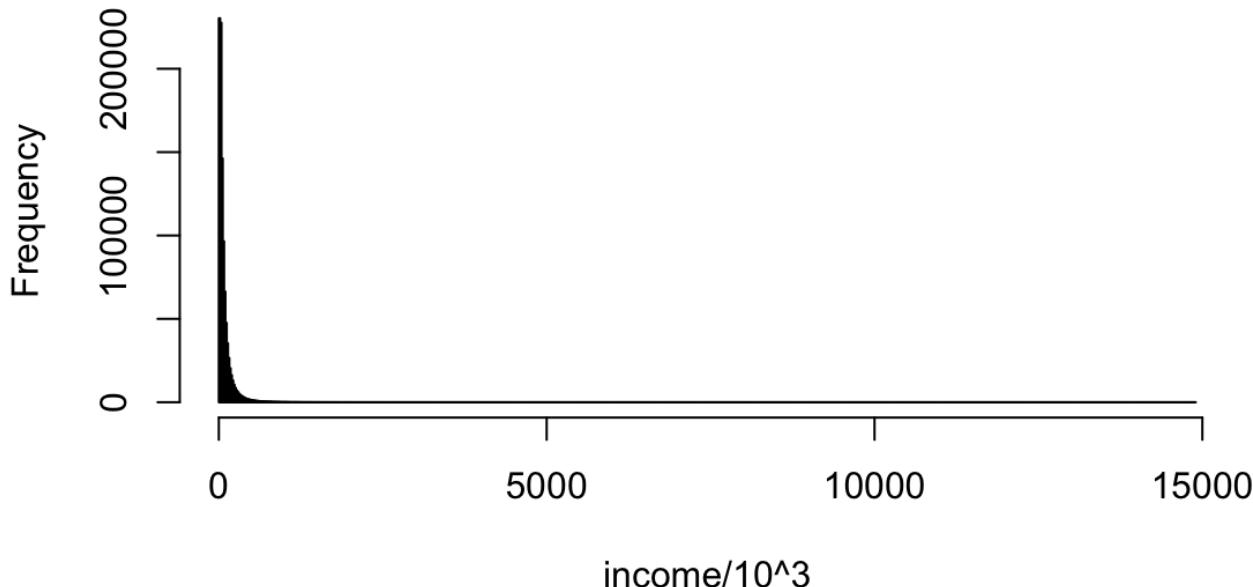


Figure 1:

The population median is:

```

m <- median(income)
m
#> [1] 44939

```

Suppose we don't have access to the entire population, but want to estimate the median  $\langle m \rangle$ . We take a sample of 100 and estimate the population median  $\langle m \rangle$  with the sample median  $\langle M \rangle$ :

```

N <- 100
x <- sample(income, N)
median(x)
#> [1] 38461

```

## 10.2 Confidence intervals for the median

Can we construct a confidence interval? What is the distribution of  $\langle M \rangle$ ?

Because we are simulating the data, we can use a Monte Carlo simulation to learn the distribution of  $\langle M \rangle$ .

```

library(gridExtra)
B <- 10^4
m <- replicate(B, {
 x <- sample(income, N)
 median(x)
})
hist(m, nclass = 30)
qqnorm(scale(m)); abline(0,1)

```

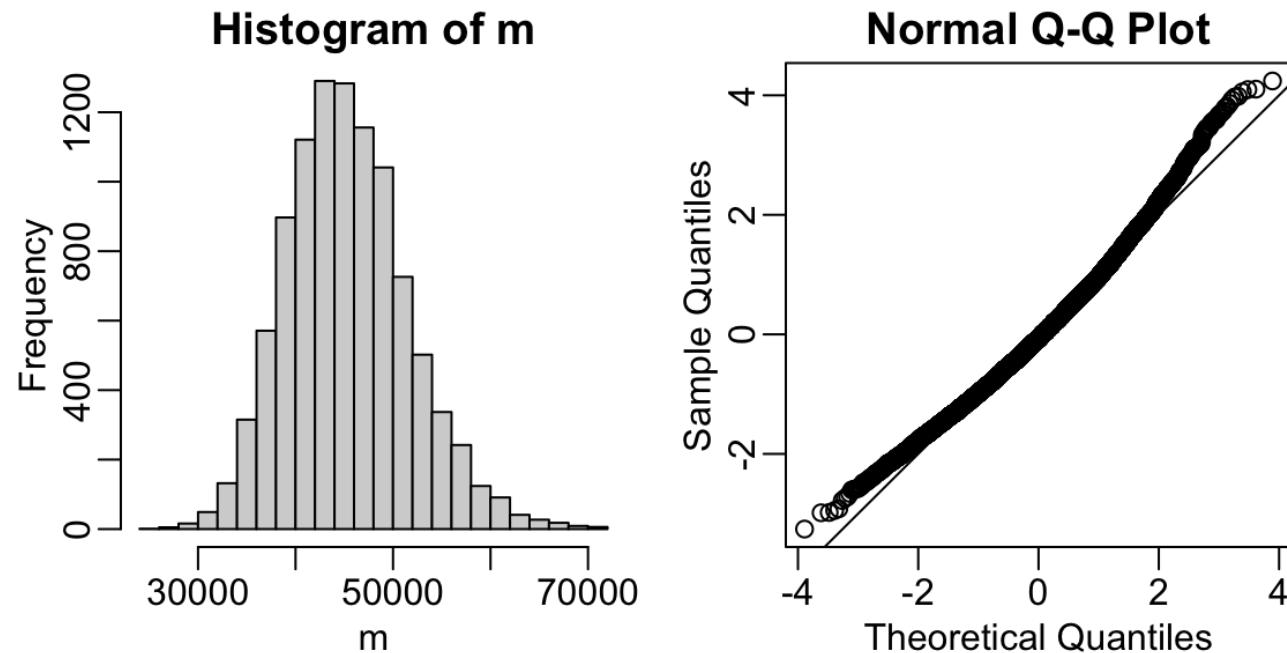


Figure 2:

If we know this distribution, we can construct a confidence interval. The problem here is that, as we have already described, in practice we do not have access to the distribution. In the past, we have used the Central Limit Theorem, but the CLT we studied applies to averages and here we are interested in the median. We can see that the 95% confidence interval based on CLT

```

median(x) + 1.96*sd(x)/sqrt(N)*c(-1, 1)
#> [1] 21018 55905

```

is quite different from the confidence interval we would generate if we know the actual distribution of  $\langle M \rangle$ :

```
quantile(m, c(0.025, 0.975))
#> 2.5% 97.5%
#> 34438 59050
```

The bootstrap permits us to approximate a Monte Carlo simulation without access to the entire distribution. The general idea is relatively simple. We act as if the observed sample is the population. We then sample (with replacement) datasets, of the same sample size as the original dataset. Then we compute the summary statistic, in this case the median, on these *bootstrap samples*.

Theory tells us that, in many situations, the distribution of the statistics obtained with bootstrap samples approximate the distribution of our actual statistic. This is how we construct bootstrap samples and an approximate distribution:

```
B <- 10^4
m_star <- replicate(B, {
 x_star <- sample(x, N, replace = TRUE)
 median(x_star)
})
```

Note a confidence interval constructed with the bootstrap is much closer to one constructed with the theoretical distribution:

```
quantile(m_star, c(0.025, 0.975))
#> 2.5% 97.5%
#> 30253 56909
```

For more on the Bootstrap, including corrections one can apply to improve these confidence intervals, please consult the book *An introduction to the bootstrap* by Efron, B., & Tibshirani, R. J.

### 10.3 Exercises

1. Generate a random dataset like this:

```
y <- rnorm(100, 0, 1)
```

Estimate the 75th quantile, which we know is:

```
qnorm(0.75)
```

with the sample quantile:

```
quantile(y, 0.75)
```

Run a Monte Carlo simulation to learn the expected value and standard error of this random variable.

2. In practice, we can't run a Monte Carlo simulation because we don't know if `rnorm` is being used to simulate the data. Use the bootstrap to estimate the standard error using just the initial sample `y`. Use 10 bootstrap samples.
3. Redo exercise 12, but with 10,000 bootstrap samples.

# Introduction to Data Science - 11 Data-driven models

Rafael A. Irizarry

“All models are wrong, but some are useful.” —George E. P. Box

So far, our analysis of poll-related results has been based on a simple sampling model. This model assumes that each voter has an equal chance of being selected for the poll, similar to picking beads from an urn with two colors. However, in this section, we explore real-world data and discover that this model is incorrect. Instead, we propose a more effective approach in which we directly model the outcomes of pollsters rather than the polls themselves.

A more recent development since the original invention of opinion polls, is the use of computers to aggregate publicly available data from different sources and develop data-driven forecasting models. Here, we explore how poll aggregators collected and combined data reported by different experts to produce improved predictions. We will introduce ideas behind the statistical models used to improve election forecasts beyond the power of individual polls. Specifically, we introduce a useful model for constructing a confidence interval for the difference in popular vote.

It’s important to note that this chapter only provides a glimpse into the vast realm of statistical models. For instance, the model we describe here does not allow us to assign a probability to a particular candidate winning the popular vote, as done by popular poll aggregators such as FiveThirtyEight. In the next section, we delve into Bayesian models, which provide the mathematical framework to make such probabilistic statements. Furthermore, in #sec-linear-models, we discuss linear models, which are widely used in statistical modeling. However, these introductions are just scratching the surface, and readers interested in statistical modeling should supplement the material presented in this book with additional references.

## 11.1 Case study: 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 which others missed. To do this, we generate results for 12 polls taken the week before the election. We mimic sample sizes from actual polls, 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)
mu <- 0.039
Ns <- c(1298, 533, 1342, 897, 774, 254, 812, 324, 1291, 1056, 2172, 516)
p <- (mu + 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 that the pollsters would have reported for the difference between Obama and Romney:

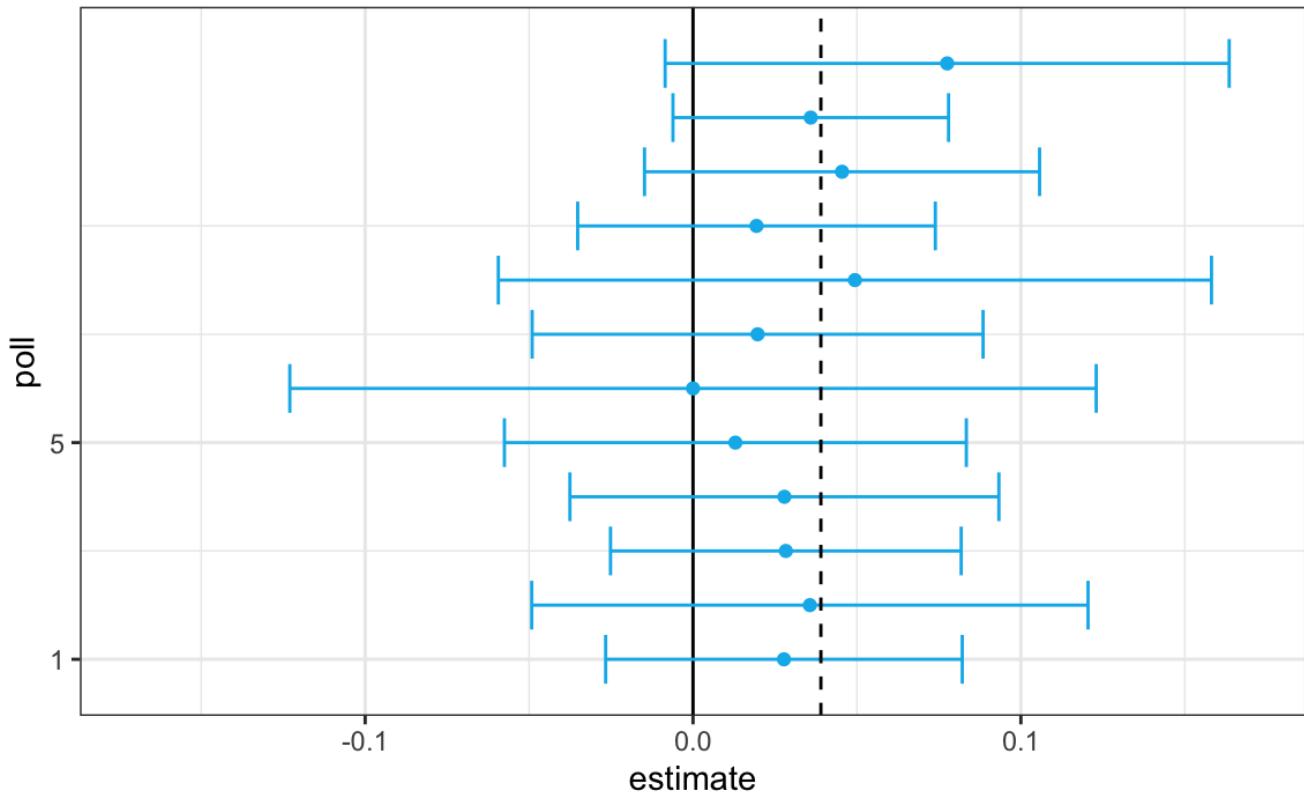


Figure 1:

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  $\langle\langle \mu \rangle\rangle$ , with a weighted average in the following way:

```
mu_hat <- polls |>
 summarize(avg = sum(estimate*sample_size) / sum(sample_size)) |>
 pull(avg)
```

Once we have an estimate of  $\langle\langle \mu \rangle\rangle$ , 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 0.0184545.

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.

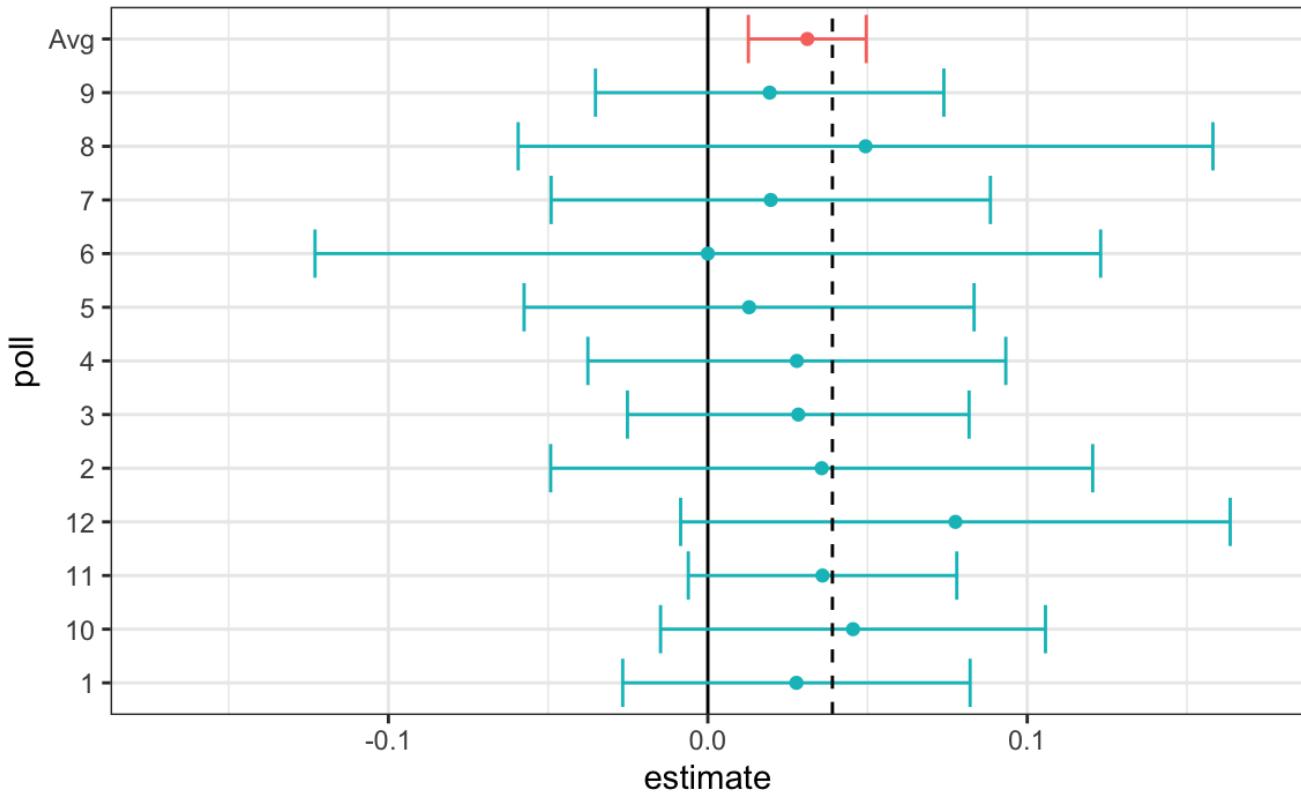


Figure 2:

However, this was just a simulation to illustrate the idea. Let's look at real data from the 2016 presidential election. Specifically, the following subset of the `polls_us_election_2016` data in `dslabs` includes results for national polls as well as state polls taken during the year prior to the election and organized by FiveThirtyEight. For this first example, we will filter the data to include national polls conducted during the week before the election. We also remove polls that FiveThirtyEight has determined to be unreliable and graded with a "B" or less. Some polls have not been graded, and we include those:

```
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)))
```

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  $(\mu)$  (for difference).

We have 49 estimates of the spread. The theory we learned from sampling models 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  $(\mu)$  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:

```
mu_hat <- polls |>
 summarize(mu_hat = sum(spread*samplesize)/sum(samplesize)) |>
 pull(mu_hat)
```

and the standard error is:

```
p_hat <- (mu_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 reveals a problem:

```
polls |> ggplot(aes(spread)) + geom_histogram(color = "black", binwidth = .01)
```

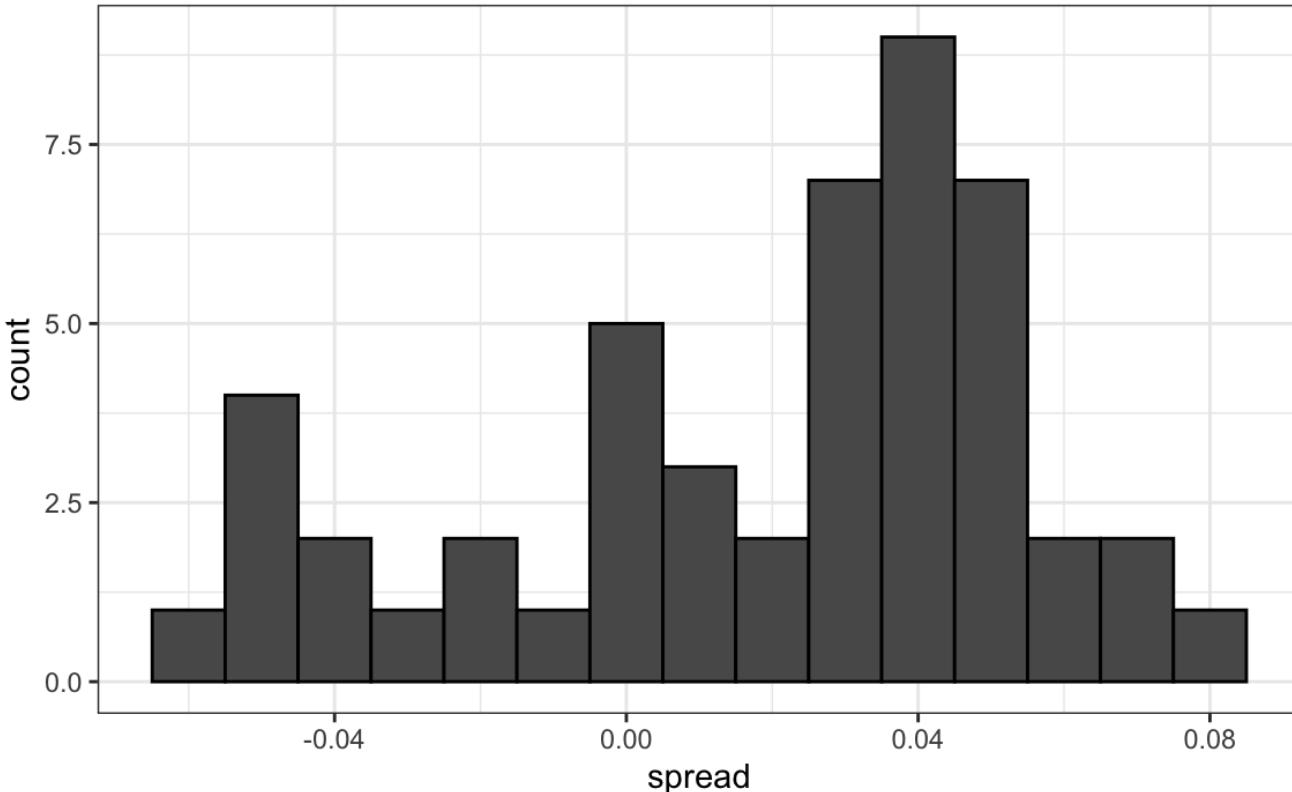


Figure 3:

The data does not appear to be normally distributed, and the standard error appears to be larger than 0.0066232. The theory is not working here, and in the next section, we describe a useful data-driven model.

## 11.2 Beyond the simple sampling model

Notice that data come from various pollsters, and some are taking several polls a week:

```
polls |> group_by(pollster) |> summarize(n())
#> # A tibble: 15 × 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
#> # 10 more rows
```

Let's visualize the data for the pollsters that are regularly polling:

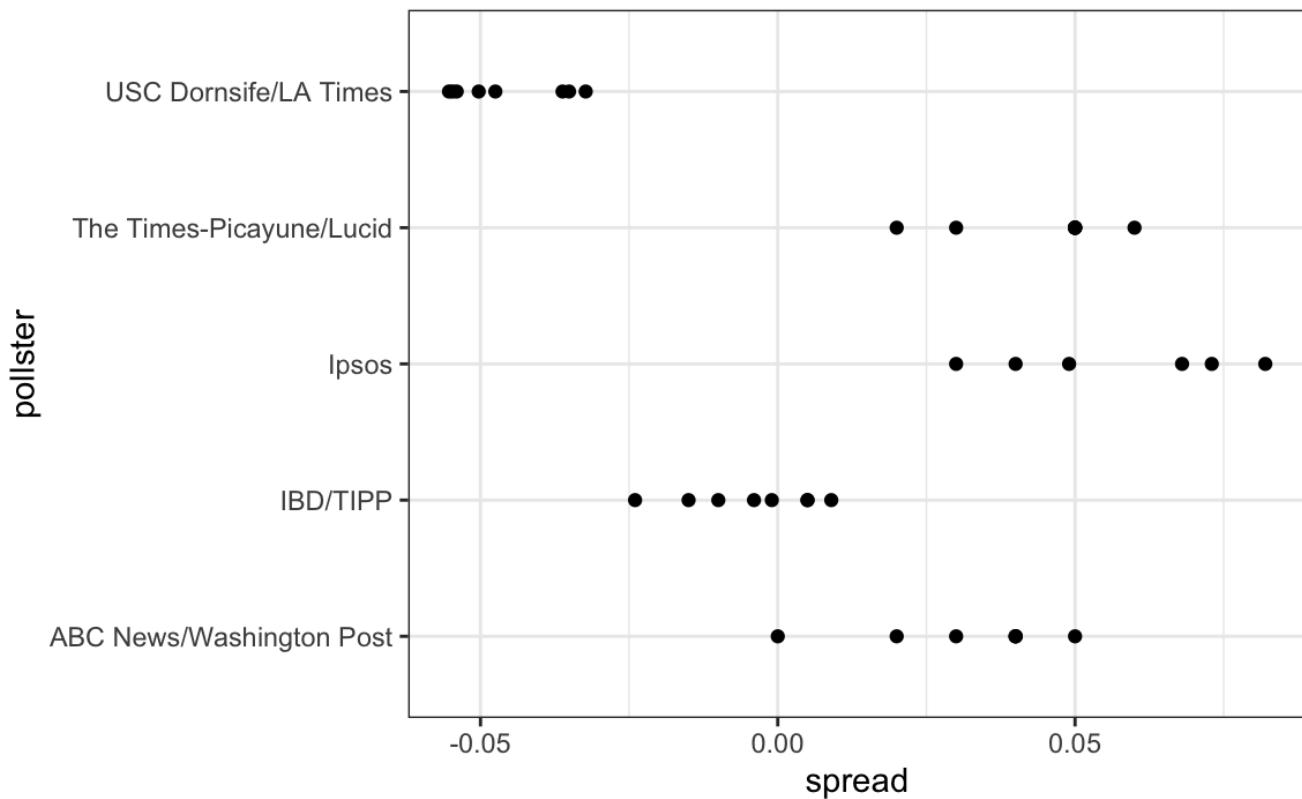


Figure 4:

This plot reveals an unexpected result. First, consider that the standard error predicted by theory for each poll is between 0.018 and 0.033:

```
polls |> group_by(pollster) |>
 filter(n() >= 6) |>
 summarize(se = 2*sqrt(p_hat*(1 - p_hat)/median(samplesize)))
#> # A tibble: 5 × 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
```

This agrees with the within poll variation we see. However, there appears to be differences *across the polls*. Observe, for example, how the USC Dornsife/LA Times pollster is predicting a 4% lead for Trump, while Ipsos is predicting a lead larger than 5% for Clinton. The theory we learned says nothing about different pollsters producing polls with different expected values, instead it assumes all the polls have the same expected value. FiveThirtyEight refers to these differences as *house effects*. We also call them *pollster bias*. Nothing in our simple urn model provides an explanation for these pollster-to-pollster differences.

This model misspecification led to an overconfident interval that ended up not including the election night result. So, rather than modeling the process generating these values with an urn model, we instead model the pollster results directly. To do this, we start by collecting some data. Specifically, for each pollster, we look at the 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)
#> Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```

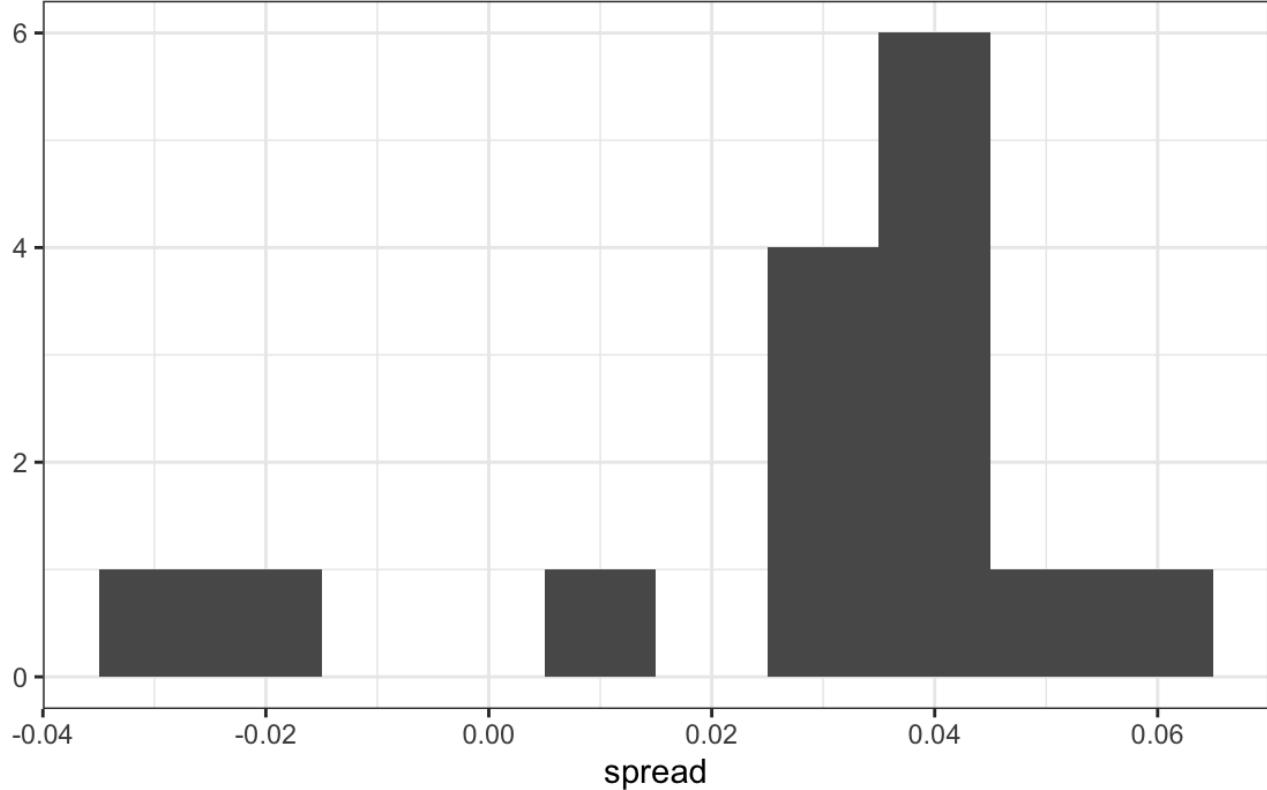


Figure 5:

Although we are no longer using a model with red (Republicans) and blue (Democrats) beads in an urn, our new model can also be thought of as an urn model, but containing poll results from all possible pollsters. Think of our  $N=15$  data points  $(X_1, \dots, X_N)$  as a random sample from this urn. To develop a useful model, we *assume* that the expected value of our urn is the actual spread  $(\mu=2p-1)$ , which implies that the sample average has expected value  $(\mu)$ .

Now, because instead of 0s and 1s, our urn contains continuous numbers, 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.

So our new statistical model is that  $(X_1, \dots, X_N)$  are a random sample with expected  $(\mu)$  and standard deviation  $(\sigma)$ . The distribution, for now, is unspecified. But we consider  $(N)$  to be large enough to assume that the sample average  $(\bar{X}) = (\sum_{i=1}^N X_i)$  follows a normal distribution with expected value  $(\mu)$  and standard error  $(\sigma / \sqrt{N})$ . We write:

$(\bar{X} \sim N(\mu, \sigma / \sqrt{N}))$  Here the  $(\sim)$  symbol tells us that the random variable on the left of the symbol follows the distribution on the right. We use the notation  $(N(a,b))$  to represent the normal distribution with mean  $(a)$  and standard deviation  $(b)$ .

This model for the sample average will be used again the next chapter.

### 11.2.1 Estimating the standard deviation

The model we have specified has two unknown parameters: the expected value  $(\mu)$  and the standard deviation  $(\sigma)$ . We know that the sample average  $(\bar{X})$  will be our estimate of  $(\mu)$ . But what about  $(\sigma)$ ?

Our task is to estimate  $(\mu)$ . Given that we model the observed values  $(X_1, \dots, X_N)$  as a random sample from the urn, 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.

Theory tells us that we can estimate the urn model  $(\sigma)$  with the *sample standard deviation* defined as:

$$[ s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_i - \bar{X})^2 } ]$$

Keep in mind that, 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 do not cover it here.

The `sd` function in R computes the sample standard deviation:

```
sd(one_poll_per_pollster$spread)
#> [1] 0.0242
```

### 11.2.2 Computing a confidence interval

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.

### 11.2.3 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  $(\sigma)$ . Applying 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.

Note that 30 is a very general rule of thumb based on the case when the data come from a normal distribution. There are cases when a large sample size is needed as well as cases when smaller sample sizes are good enough.

The statistic on which confidence intervals for  $(\mu)$  are based is:

$$[ Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{N}} ]$$

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:

$$[ t = \frac{\bar{X} - \mu}{s/\sqrt{N}} ]$$

This is referred to as a *t-statistic*. By substituting  $\sigma$  with  $s$ , we introduce some variability. The theory tells us that  $t$  follows a *student t-distribution* with  $(N-1)$  degrees of freedom. The degrees of freedom is a parameter that controls the variability via fatter tails:

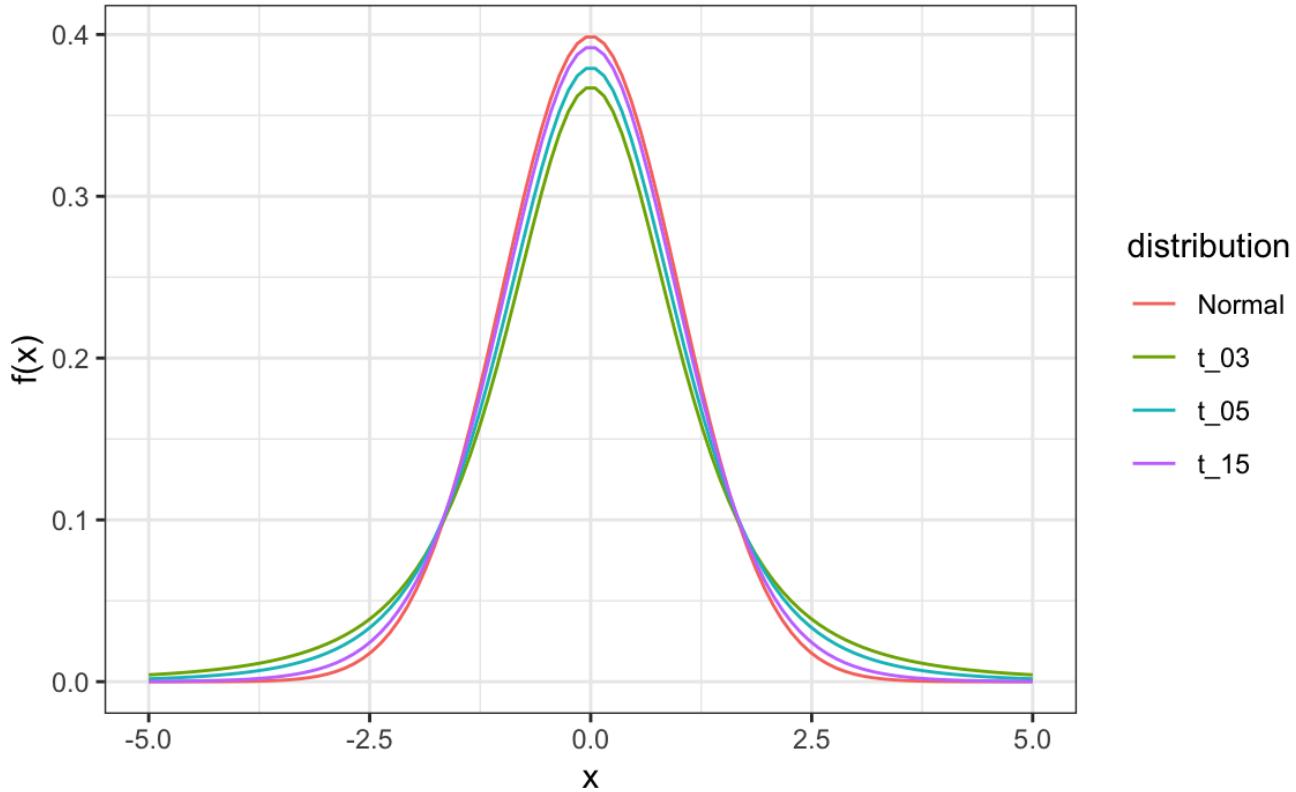


Figure 6:

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  $t$  follows a t-distribution with  $(N-1)$  degrees of freedom. So perhaps a better confidence interval for  $\mu$  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 × 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
```

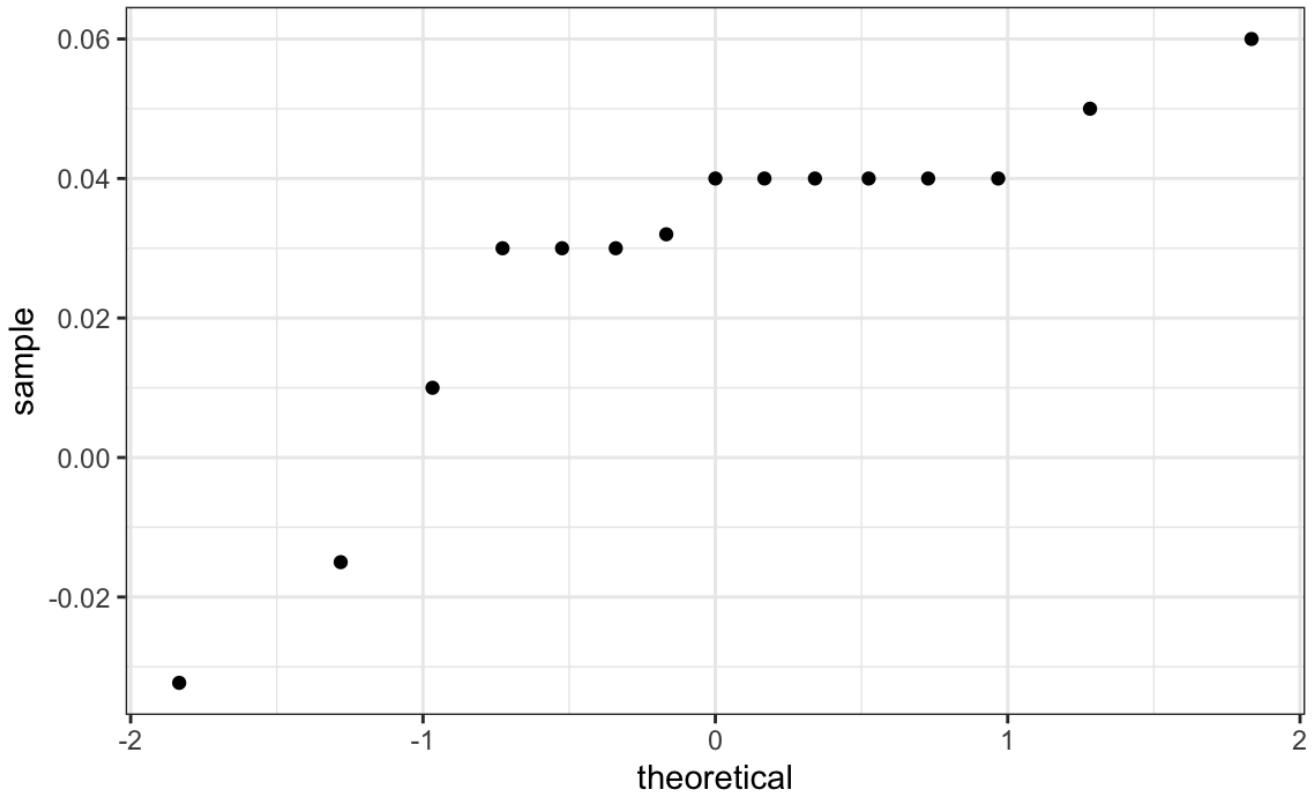


Figure 7:

This results in a slightly larger confidence interval than we obtained before:

```
#> start end
#> 1 1.6 4.2
```

Note that using the t-distribution and the t-statistic is the basis for *t*-tests, a widely used approach for computing p-values. To learn more about t-tests, you can consult any statistics textbook.

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 observed. 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.

```
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
```

### 11.3 Exercises

We have been using urn models to motivate the use of probability models. Yet, 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.

Define the males that replied to the height survey as the population

```
library(dslabs)
x <- heights |> filter(sex == "Male") |>
 pull(height)
```

to answer the following questions.

1. Mathematically speaking,  $x$  is our population. Using the urn analogy, we have an urn with the values of  $x$  in it. What are the average and standard deviation of our population?
2. Call the population average computed above  $\langle(\mu)\rangle$  and the standard deviation  $\langle(\sigma)\rangle$ . Now take a sample of size 50, with replacement, and construct an estimate for  $\langle(\mu)\rangle$  and  $\langle(\sigma)\rangle$ .
3. What does the theory tell us about the sample average  $\langle(\bar{X})\rangle$  and how it is related to  $\langle(\mu)\rangle$ ?
  - a. It is practically identical to  $\langle(\mu)\rangle$ .
  - b. It is a random variable with expected value  $\langle(\mu)\rangle$  and standard error  $\langle(\sigma/\sqrt{N})\rangle$ .
  - c. It is a random variable with expected value  $\langle(\mu)\rangle$  and standard error  $\langle(\sigma)\rangle$ .
  - 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 can only measure 50 of the 708. We will use  $\langle(\bar{X})\rangle$  as our estimate. We know from the answer to exercise 3 that the standard estimate of our error  $\langle(\bar{X}-\mu)\rangle$  is  $\langle(\sigma/\sqrt{N})\rangle$ . We want to compute this, but we don't know  $\langle(\sigma)\rangle$ . Based on what is described in this section, show your estimate of  $\langle(\sigma)\rangle$ .
5. Now that we have an estimate of  $\langle(\sigma)\rangle$ , let's call our estimate  $\langle(s)\rangle$ . Construct a 95% confidence interval for  $\langle(\mu)\rangle$ .
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  $\langle(\mu)\rangle$ ?
7. Use the `qnorm` and `qt` functions to generate quantiles. Compare these quantiles for different degrees of freedom for the t-distribution. Use this to motivate the sample size of 30 rule of thumb.

# Introduction to Data Science - 12 Bayesian statistics

Rafael A. Irizarry

In 2016, FiveThirtyEight showed this chart depicting distributions for the percent of the popular vote for each candidate:

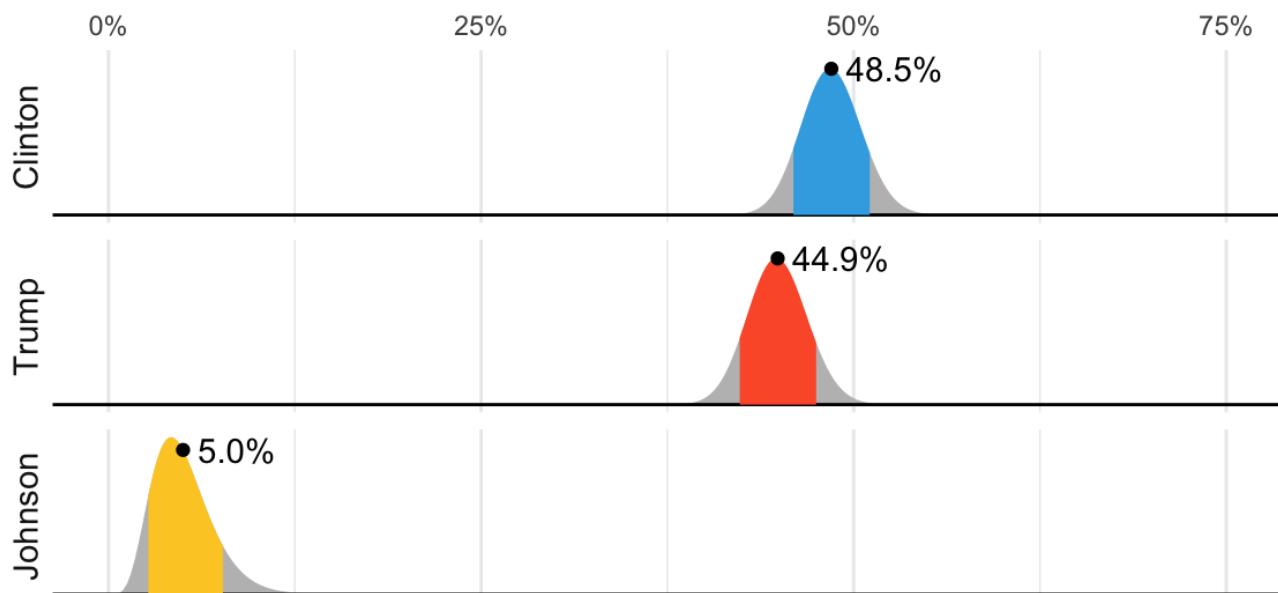


Figure 1: The colored areas represent values with an 80% chance of including the actual result, according to the FiveThirtyEight model.

But what does this mean in the context of the theory we have previously covered, in which these percentages are considered fixed? Furthermore, election forecasters make probabilistic statements such “Obama has a 90% chance of winning the election.” Note that in the context of an urn model, this would be equivalent to stating that the probability  $(p>0.5)$  is 90%. However, the urn model  $(p)$  is a fixed parameter and it does not make sense to talk about probability. With Bayesian statistics, we assume  $(p)$  is random variable, and thus, a statement such as “90% chance of winning” is consistent with the mathematical 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.

The approach described in the previous chapters, where the parameters are considered fixed, is often referred to as **frequentist**.

In this chapter, we will briefly describe Bayesian statistics. We use three cases studies: 1) interpreting diagnostic tests for a rare disease, 2) predicting the performance of an athlete, and 3) estimating the probability of Hillary

Clinton winning in 2016 using pre-election poll data. 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.

## 12.1 Bayes theorem

We start by describing Bayes theorem, 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:

$$\Pr(\text{Test Positive} \mid D=1) = 0.99, \Pr(\text{Test Negative} \mid D=0) = 0.99$$

with  $\text{(+)}$  meaning a positive test and  $\text{(D)}$  representing if you actually have the disease (1) or not (0).

Imagine we select a random person and they test positive. What is the probability that they have the disease? We write this as  $\Pr(D=1 \mid +)$ . The cystic fibrosis rate is 1 in 3,900, which implies that  $\Pr(D=1) = 0.00025$ . 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, when applied to our problem, becomes:

$$\Pr(D=1 \mid +) = \frac{\Pr(+ \mid D=1) \cdot \Pr(D=1)}{\Pr(+ \mid D=1) \cdot \Pr(D=1) + \Pr(+ \mid D=0) \cdot \Pr(D=0)}$$

Plugging in the numbers, we get:

$$\Pr(D=1 \mid +) = \frac{0.99 \cdot 0.00025}{0.99 \cdot 0.00025 + 0.01 \cdot (0.9975)} = 0.02$$

According to the above, despite the test having 0.99 accuracy, the probability of having the disease given a positive test is only 0.02. This might seem counter-intuitive to some, but it is because we must factor in the very rare probability that a randomly chosen person has the disease. To illustrate this, we run a Monte Carlo simulation.

### 12.1.1 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 people 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))
```

Given that there are so many more controls than cases, even with a low false positive rate, we end up with 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.

## 12.2 Priors, posteriors and and credible intervals

In the previous chapter, we computed an estimate and margin of error for the difference in popular votes between Hillary Clinton and Donald Trump. We denoted the parameter, the the difference in popular votes, with  $(\mu)$ . The estimate was between 2 and 3 percent, and the confidence interval did not include 0. A forecaster would use this to predict Hillary Clinton would win the popular vote. But to make a probabilistic statement about winning the election, we need to use a Bayesian approach.

We start the Bayesian approach by quantifying our knowledge *before* seeing any data. This is done using a probability distribution referred to as a *prior*. For our example, we could write:

$$[\mu \sim N(\theta, \tau)]$$

We can think of  $(\theta)$  as our best guess for the popular vote difference had we not seen any polling data, and we can think of  $(\tau)$  as quantifying how certain we feel about this guess. Generally, if we have *expert knowledge* related to  $(\mu)$ , we can try to quantify it with the prior distribution. In the case of election polls, experts use *fundamentals*, which include, for example, the state of the economy, to develop prior distributions.

The data is used to update our initial guess or *prior belief*. This can be done mathematically if we define the distribution for the observed data for any given  $(\mu)$ . In our particular example, we would write down a model the average of our polls, which is the same as before:

$$[\bar{X} \mid \mu \sim N(\mu, \sigma/\sqrt{N})]$$

As before,  $(\sigma)$  describes randomness due to sampling and the pollster effects. In the Bayesian contexts, this is referred to as the sampling distribution. Note that we write the conditional  $(\bar{X} \mid \mu)$  because  $(\mu)$  is now considered a random variable.

We do not show the derivations here, but we can now use calculus and a version of Bayes' Theorem to derive the distribution of  $(\mu)$  conditional of the data, referred to as the posterior distribution. Specifically, we can show  $(\mu \mid \bar{X})$  follows a normal distribution with expected value:

$$\begin{aligned} E(\mu \mid \bar{X}) &= B\theta + (1-B)\bar{X} \\ B &= \frac{\sigma^2/N}{\sigma^2/N + \tau^2} \end{aligned}$$

$$SE(\mu \mid \bar{X}) = \sqrt{\frac{1}{B(1-B)}} \cdot \sqrt{\frac{\sigma^2}{N} + \tau^2}$$

Note that the expected value is a weighted average of our prior guess  $(\theta)$  and the observed data  $(\bar{X})$ . The weight depends on how certain we are about our prior belief, quantified by  $(\tau)$ , and the precision  $(\sigma^2/N)$  of the summary of our observed data. This weighted average is sometimes referred to as *shrinking* because it *shrinks* estimates towards a prior value.

These quantities are useful for updating our beliefs. Specifically, we use the posterior distribution not only to compute the expected value of  $(\mu)$  given the observed data, but also, for any probability  $(\alpha)$ , we can construct intervals centered at our estimate and with  $(\alpha)$  chance of occurring.

To compute a posterior distribution and construct a credible interval, we define a prior distribution with mean 0% and standard error 3.5%, which can be interpreted as follows: before seeing polling data, we don't think any candidate has the advantage, and a difference of up to 7% either way is possible. We compute the posterior distribution using the equations above:

```

theta <- 0
tau <- 0.035
sigma <- results$se
x_bar <- results$avg
B <- sigma^2 / (sigma^2 + tau^2)

posterior_mean <- B*theta + (1 - B)*x_bar
posterior_se <- sqrt(1/(1/sigma^2 + 1/tau^2))

posterior_mean
#> [1] 0.0281
posterior_se
#> [1] 0.00615

```

Since we know the posterior distribution is normal, we can construct a credible interval like this:

```

posterior_mean + c(-1, 1) * qnorm(0.975) * posterior_se
#> [1] 0.0160 0.0401

```

Furthermore, we can now make the probabilistic statement that we could not make with the frequentists approach by computing the posterior probability of Hillary winning the popular vote. Specifically,  $\Pr(\mu > 0 \mid \bar{X})$  can be computed as follows:

```

1 - pnorm(0, posterior_mean, posterior_se)
#> [1] 1

```

According to the above, we are 100% sure Clinton will win the popular vote, which seems too overconfident. Additionally, it is not in agreement with FiveThirtyEight's 81.4%. What explains this difference? There is a level of uncertainty that we are not yet describing, and we will return to that in Chapter 13.

## 12.3 Exercises

1. In 1999, in England, Sally Clark<sup>1</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 causes. If genetics plays a role then:  $\Pr(\text{second case of SIDS} \mid \text{first case of SIDS}) > \Pr(\text{first case of SIDS})$ .
  - b. Nothing. The multiplication rule always applies in this way:  $\Pr(A \text{ and } B) = \Pr(A)\Pr(B)$
  - c. Sir Meadow is an expert and we should trust his calculations.
  - d. Numbers don't lie.
2. 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?
3. Many press reports stated that the expert claimed the probability of Sally Clark being innocent was 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?
4. 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?

- a. He made an arithmetic error.
- b. 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.
- c. He mixed up the numerator and denominator of Bayes’ rule.
- d. He did not use R.

6. Florida is one of the most closely watched states in U.S. elections because it has many electoral votes. In past elections, Florida was a swing state where both Republicans and Democrats won implying it could affect a close election.

Create the following table with the polls taken during the last two weeks:

```
library(tidyverse)
library(dslabs)
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  $\mu$  to follow a normal distribution with expected value  $\theta$  and standard deviation  $\tau$ . What are the interpretations of  $\theta$  and  $\tau$ ?

- a.  $\theta$  and  $\tau$  are arbitrary numbers that let us make probability statements about  $\mu$ .
- b.  $\theta$  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.  $\theta$  and  $\tau$  summarize what we want to be true. We therefore set  $\theta$  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  $\hat{\mu}$  has normal distribution with expected value  $\mu$  and standard deviation  $\sigma$  calculated in exercise 6. Use the formulas we provided for the posterior distribution to calculate the expected value of the posterior distribution if we set  $\theta = 0$  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 **sapply** function to change the prior variance from `seq(0.005, 0.05, len = 100)` and observe how the probability changes by making a plot.

1. [https://en.wikipedia.org/wiki/Sally\\_Clark](https://en.wikipedia.org/wiki/Sally_Clark)

# Introduction to Data Science - 13 Hierarchical Models

Rafael A. Irizarry

Hierarchical models are useful for quantifying different levels of variability or uncertainty. One can use them using a Bayesian or Frequentist framework. However, because in the Frequentist framework they often extend a model with a fixed parameter by assuming the parameter is actually random, the model description includes two distributions that look like the prior and a sampling distribution used in the Bayesian framework. This makes the resulting summaries very similar or even equal to what is obtained with a Bayesian context. A key difference between the Bayesian and the Frequentist hierarchical model approach is that, in the latter, we use data to construct priors rather than treat priors as a quantification of prior expert knowledge. In this section, we illustrate the use of hierarchical models with an example from sports, in which dedicated fans intuitively apply the ideas of hierarchical models to manage expectations when a new player is off to an exceptionally good start.

## 13.1 Case study: election forecasting

Since the 2008 elections, organizations other than FiveThirtyEight have started their own election forecasting groups that also aggregate polling data and uses statistical models to make predictions. However, in 2016, forecasters underestimated Trump's chances of winning greatly. The day before the election, the *New York Times* reported<sup>1</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 |

Meanwhile, 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%, substantially higher than the others. In fact, four days before the election, FiveThirtyEight published an article titled *Trump Is Just A Normal Polling Error Behind Clinton*<sup>2</sup>.

So why did FiveThirtyEight's model fair so much better than others? How could PEC and Huffington Post get it so wrong if they were using the same data? In this chapter, we describe how FiveThirtyEight used a hierarchical model to correctly account for key sources of variability and outperform all other forecasters. For illustrative purposes, we will continue examining our popular vote example. In the final section, we will describe the more complex approach used to forecast the electoral college result.

## 13.2 The general bias

In the previous chapter, we computed the posterior probability of Hillary Clinton winning the popular vote with a standard Bayesian analysis and found it to be very close to 100%. However, FiveThirtyEight gave her a 81.4% chance<sup>3</sup>. What explains this difference? Below, we describe the *general bias*, another source of variability, included in the FiveThirtyEight model, that accounts for the difference.

After elections are over, one can look at the difference between the pollster predictions and the actual result. An important observation, that our initial models did not take into account, is that it is common to see a general bias that affects most pollsters in the same way, making the observed data correlated. There is no agreed upon 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 the variability.

### 13.3 Mathematical representations of the hierarchical model

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  $\langle N \rangle$ , so we observe several measurements of the spread  $\langle X_1, \dots, X_J \rangle$ . Suppose the real proportion for Hillary is  $\langle p \rangle$  and the difference is  $\langle \mu \rangle$ . The urn model theory tells us that these random variables are normally distributed, with expected value  $\langle \mu \rangle$  and standard error  $\langle 2 \sqrt{p(1-p)/N} \rangle$ :

$$\langle X_j \sim \text{mbox}\{N\}(\mu, \sqrt{p(1-p)/N}) \rangle$$

We use the index  $\langle j \rangle$  to represent the different polls conducted by this pollster. Below is a simulation for six polls assuming the spread is 2.1 and  $\langle N \rangle$  is 2,000:

```
set.seed(3)
J <- 6
N <- 2000
mu <- .021
p <- (mu + 1)/2
X <- rnorm(J, mu, 2*sqrt(p*(1 - p)/N))
```

Now, suppose we have  $\langle J=6 \rangle$  polls from each of  $\langle I=5 \rangle$  different pollsters. For simplicity, let's say all polls had the same sample size  $\langle N \rangle$ . The urn model tell us the distribution is the same for all pollsters, so to simulate data, we use the same model for each:

```
I <- 5
J <- 6
N <- 2000
X <- sapply(1:I, function(i){
 rnorm(J, mu, 2*sqrt(p*(1 - p)/N))
})
```

As expected, the simulated data does not really seem to capture the features of the actual data because it does not account for pollster-to-pollster variability:

To fix this, we need to represent the two levels of variability and we need two indexes, one for pollster and one for the polls each pollster takes. We use  $\langle X_{ij} \rangle$  with  $\langle i \rangle$  representing the pollster and  $\langle j \rangle$  representing the  $\langle j \rangle$ -th poll from that pollster. The model is now augmented to include pollster effects  $\langle h_i \rangle$ , referred to as "house effects" by FiveThirtyEight, with standard deviation  $\langle \sigma_h \rangle$ :

$$\langle \begin{aligned} h_i &\sim \text{mbox}\{N\}(0, \sigma_h) \\ X_{ij} &\mid h_i \sim \text{mbox}\{N\}(\mu + h_i, \sqrt{p(1-p)/N}) \end{aligned} \rangle$$

To simulate data from a specific pollster, we first need to draw an  $\langle h_i \rangle$ , and then generate individual poll data after adding this effect. Here is how we would do it for one specific pollster. We assume  $\langle \sigma_h \rangle$  is 0.025:

```
I <- 5
J <- 6
N <- 2000
mu <- .021
p <- (mu + 1)/2
h <- rnorm(I, 0, 0.025)
X <- sapply(1:I, function(i){
 mu + h[i] + rnorm(J, 0, 2*sqrt(p*(1 - p)/N))
})
```

The simulated data now looks more like the actual data:

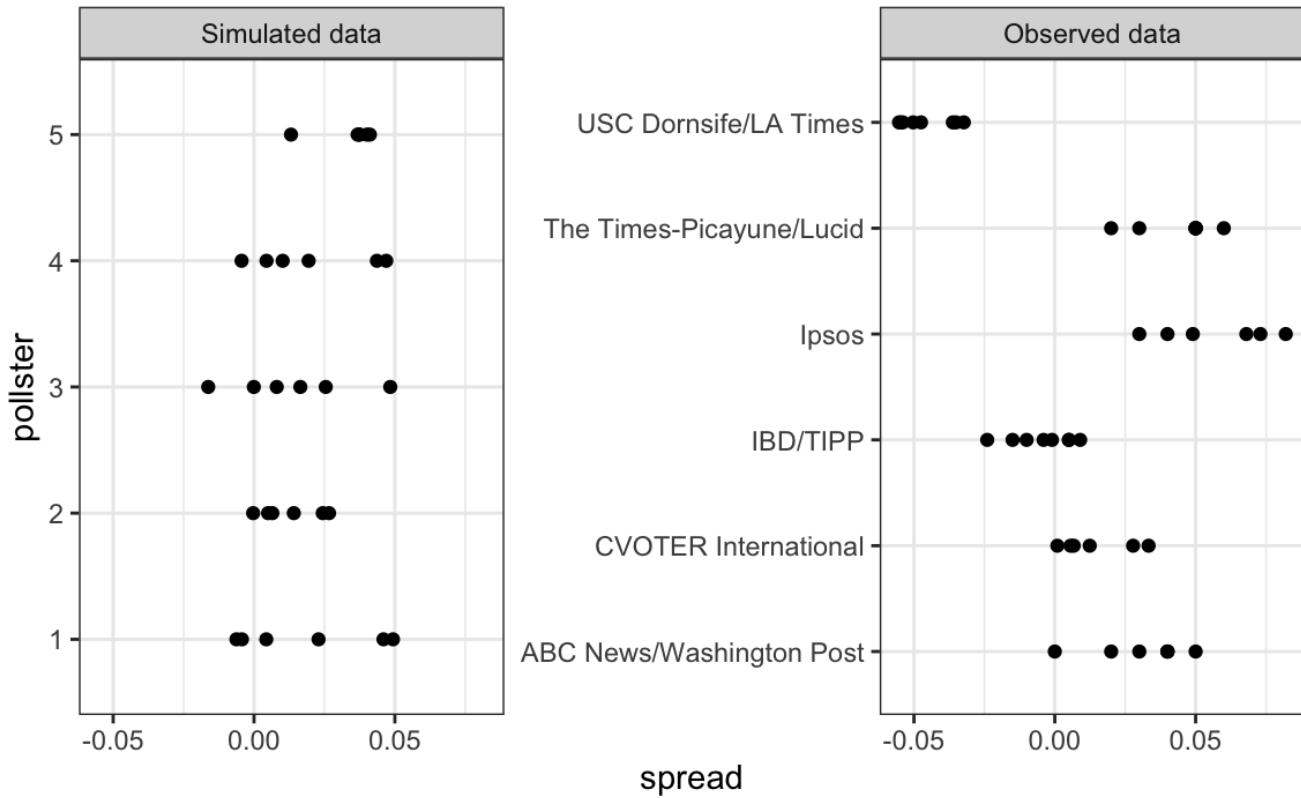


Figure 1:

Note that  $\langle h_i \rangle$  is common to all the observed spreads from a specific pollster. Different pollsters have a different  $\langle h_i \rangle$ , 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  $\langle \sigma_h \rangle$ . 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 account for this variability we can add another level to the model as follows:

$$\begin{aligned} b &\sim \text{mbox}\{N\}(\mu, \sigma_b) \\ h_j &\mid b \sim \text{mbox}\{N\}(b, \sigma_h) \\ X_{i,j} &\mid b, h_j \sim \text{mbox}\{N\}(\mu + h_j, \sqrt{p(1-p)/N}) \end{aligned}$$

This model accounts for three levels of variability: 1) variability in the bias observed from election to election, quantified by  $\langle \sigma_b \rangle$ , 2) pollster-to-pollster or house effect variability, quantified by  $\langle \sigma_h \rangle$ , and 3) poll sampling variability, which we can derive to be  $\langle \sqrt{p(1-p)/N} \rangle$ .

Note that not including a term like  $\langle b \rangle$  in the models is what led many forecasters to be overconfident. This random variable changes from election to election, but for any given election, it is the same for all pollsters and polls within one election (note it does not have an index). This implies that we can't estimate  $\langle \sigma_h \rangle$  with data from just one election. It also implies that the random variables  $\langle X_{i,j} \rangle$  for a fixed election year share the same  $\langle b \rangle$  and are therefore correlated.

One way to interpret  $\langle b \rangle$  is as the difference between the average of all polls from all pollsters and the actual result of the election. Since we don't know the actual result until after the election, we can't estimate  $\langle b \rangle$  until then.

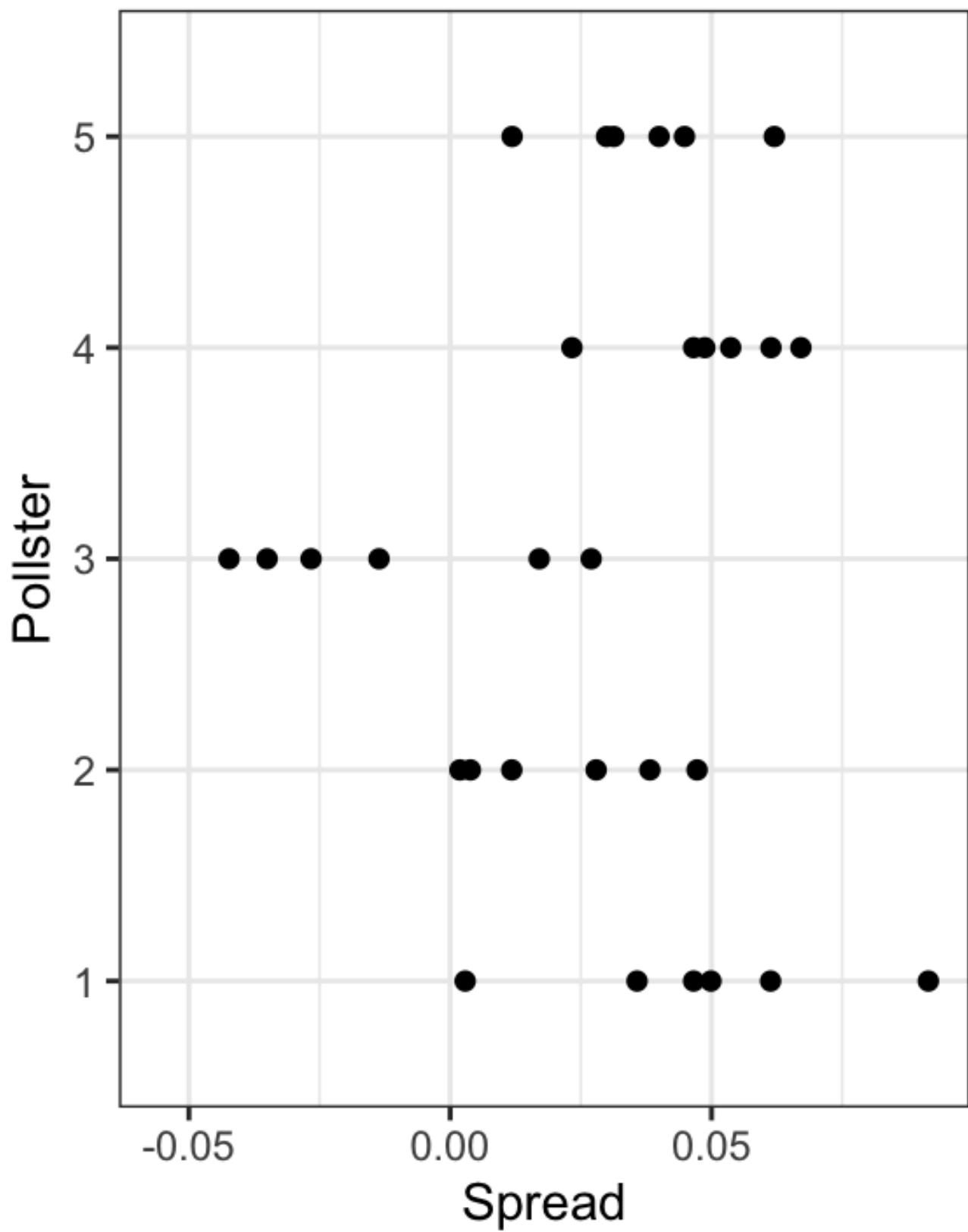


Figure 2:

## 13.4 Computing a posterior probability

Some of the results presented in this section rely on calculations of the statistical properties of summaries based on correlated random variables. To learn about the related mathematical details we skip in this book, please consult a textbook on hierarchical models.

Now, let's fit the model above to data. We will use the same data used in the previous chapters and saved in `one_poll_per_pollster`.

```
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()
```

Here, we have just one poll per pollster, so we will drop the  $\langle j \rangle$  index and represent the data as before with  $\langle X_{-1}, \dots, X_I \rangle$ . As a reminder, we have data from  $I=15$  pollsters. Based on the model assumptions described above, we can mathematically show that the average  $\langle \bar{X} \rangle$ :

```
x_bar <- mean(one_poll_per_pollster$spread)
```

has expected value  $\langle \mu \rangle$ ; thus, it provides an unbiased estimate of the outcome of interest. But how precise is this estimate? Can we use the observed sample standard deviation to construct an estimate of the standard error of  $\langle \bar{X} \rangle$ ?

It turns out that, because the  $\langle X_i \rangle$  are correlated, estimating the standard error is more complex than what we have described up to now. Specifically, using advanced statistical calculations not shown here, we can show that the typical variance (standard error squared) estimate:

```
s2 <- with(one_poll_per_pollster, sd(spread)^2/length(spread))
```

will consistently underestimate the true standard error by about  $\langle \sigma_b^2 \rangle$ . And, as mentioned earlier, to estimate  $\langle \sigma_b \rangle$ , we need data from several elections. By collecting and analyzing polling data from several elections, FiveThirtyEight estimates this variability and finds that  $\langle \sigma_b \rangle \approx 0.025$ . We can therefore greatly improve our standard error estimate by adding this quantity:

```
sigma_b <- 0.025
se <- sqrt(s2 + sigma_b^2)
```

If we redo the Bayesian calculation taking this variability into account, we obtain a result much closer to FiveThirtyEight's:

```
mu <- 0
tau <- 0.035
B <- se^2/(se^2 + tau^2)
posterior_mean <- B*mu + (1 - B)*x_bar
posterior_se <- sqrt(1/(1/se^2 + 1/tau^2))

1 - pnorm(0, posterior_mean, posterior_se)
#> [1] 0.817
```

Notice that by accounting for the general bias term, our Bayesian analysis now produces a posterior probability similar to that reported by FiveThirtyEight.

Keep in mind that we are simplifying FiveThirtyEight's calculations related to the general bias  $\langle b \rangle$ . For example, one of the many ways their analysis is more complex than the one presented here is that FiveThirtyEight permits  $\langle b \rangle$  to vary across regions of the country. This helps because, historically, we have observed geographical patterns in voting behaviors.

## 13.5 Predicting the electoral college

Up to now, we have focused on the popular vote. However, 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 won't discuss, the electoral votes are won on an all-or-nothing basis. For example, if you won California in 2016 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 prevent a few large states from having the power to dominate the presidential election.

Many people in the US consider the electoral college unfair and would like to see it abolished in favor of the popular vote.

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 `grepl`, which finds strings in character vectors, to remove polls that are not for entire states.

```
results <- polls_us_election_2016 |>
 filter(state != "U.S." &
 !grepl("CD", state) &
 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 × 4
#> state avg sd n
#> <chr> <dbl> <dbl> <int>
#> 1 Florida 0.00356 0.0163 7
#> 2 North Carolina -0.0073 0.0306 9
#> 3 Ohio -0.0104 0.0252 6
#> 4 Nevada 0.0169 0.0441 7
#> 5 Iowa -0.0197 0.0437 3
#> # 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`. 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  $(\mu)$ . 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. Given that results from a specific state don't vary significantly from election year to election year, 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:

```
mu <- 0
tau <- 0.02
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)))
#> # A tibble: 47 × 12
#> state avg sd n electoral_votes clinton trump others
#> <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
#> 1 Alabama -0.149 0.0253 3 9 34.4 62.1 3.6
#> 2 Arizona -0.0326 0.0270 9 11 45.1 48.7 6.2
#> 3 Arkansas -0.151 0.000990 2 6 33.7 60.6 5.8
#> 4 California 0.260 0.0387 5 55 61.7 31.6 6.7
#> 5 Colorado 0.0452 0.0295 7 9 48.2 43.3 8.6
#> # 42 more rows
#> # 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 track the total number of electoral votes for Clinton. Remember that Trump gets 270 votes minus the ones 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)
```

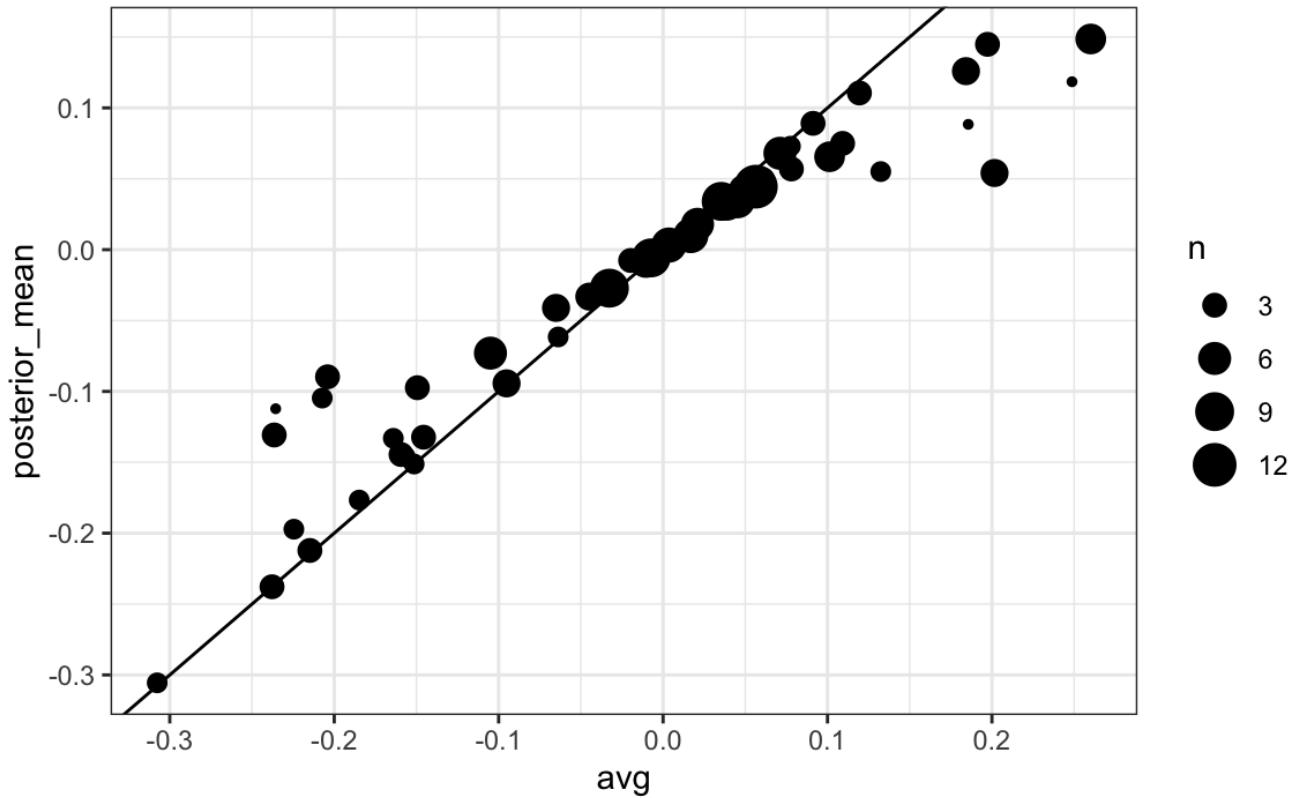


Figure 3:

```
#> [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 introduced, they assumed the standard error was  $\sqrt{\sigma^2/N}$ . With large  $N$ , this estimate is substantially closer to 0 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.

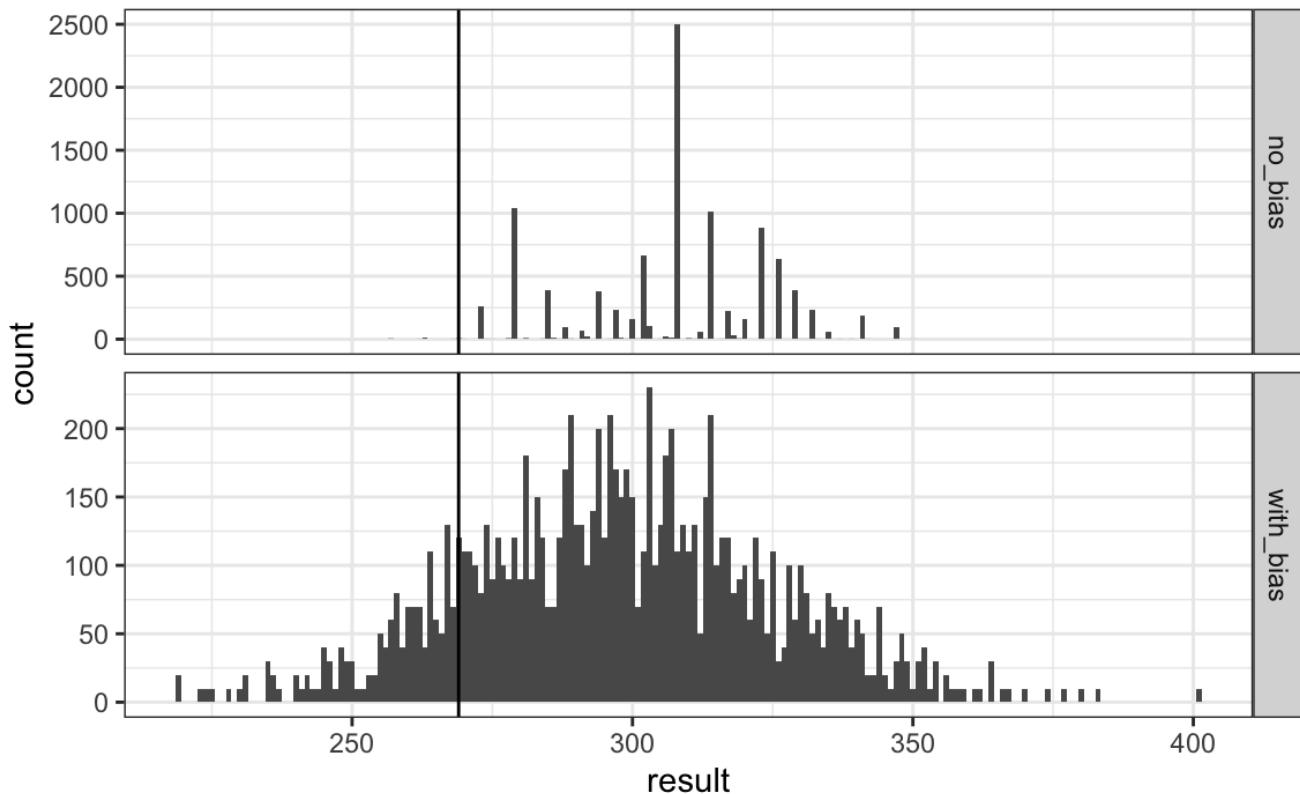


Figure 4:

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%.

### 13.6 Forecasting

Forecasters like to make predictions well before the election. The predictions are adapted as new poll results are released. 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:

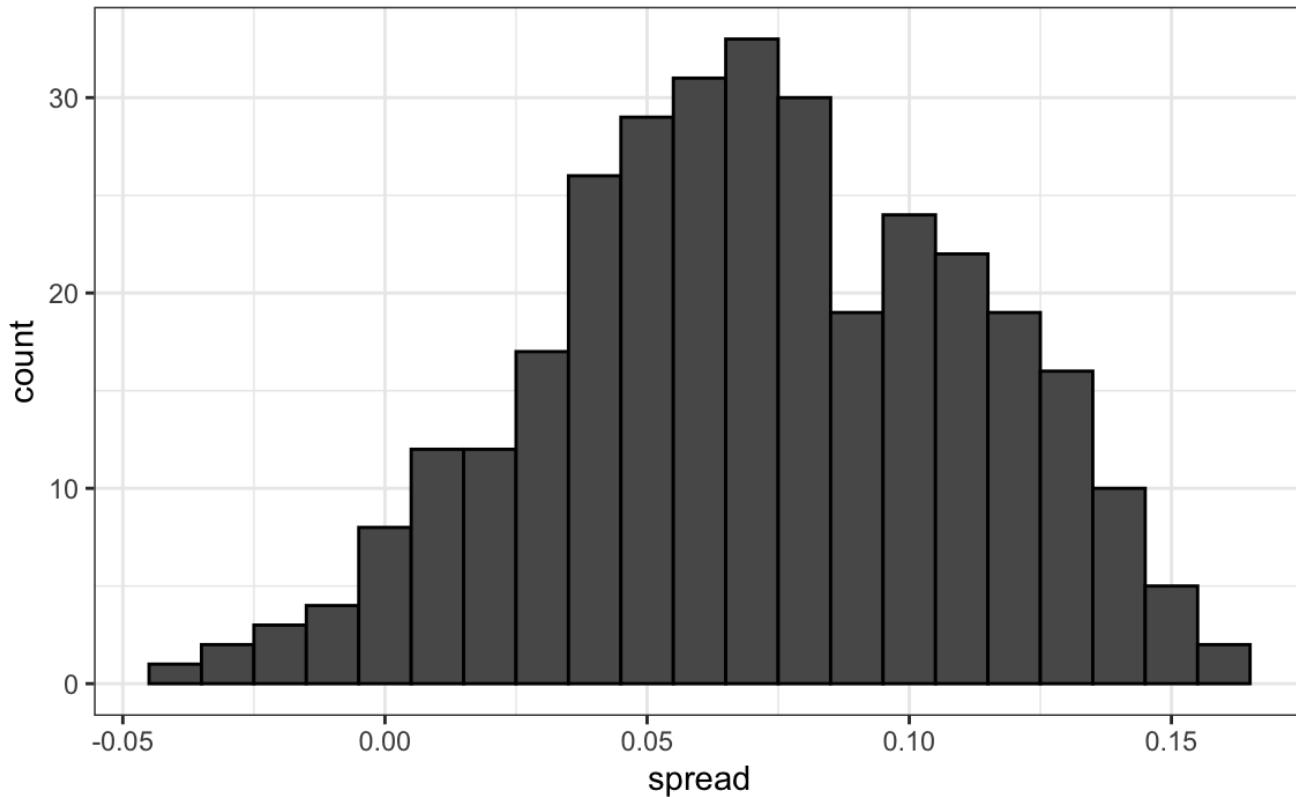


Figure 5:

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  $\langle p \rangle$  is fixed:

```
#> `geom_smooth()` using formula = 'y ~ x'
```

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 that the peaks and valleys are consistent across several pollsters:

```
#> `geom_smooth()` using formula = 'y ~ x'
```

This implies that if we are going to forecast, our model must include a term to account for the time effect. We need to write a model including a bias term for time, denoted as  $\langle b_t \rangle$ . The standard deviation of  $\langle b_t \rangle$  would depend on  $\langle t \rangle$  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 them into their predictions. We can model the time trend  $\langle b_t \rangle$  with a smooth function. We usually see the trend estimate 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 which we discuss in the section on Machine Learning.

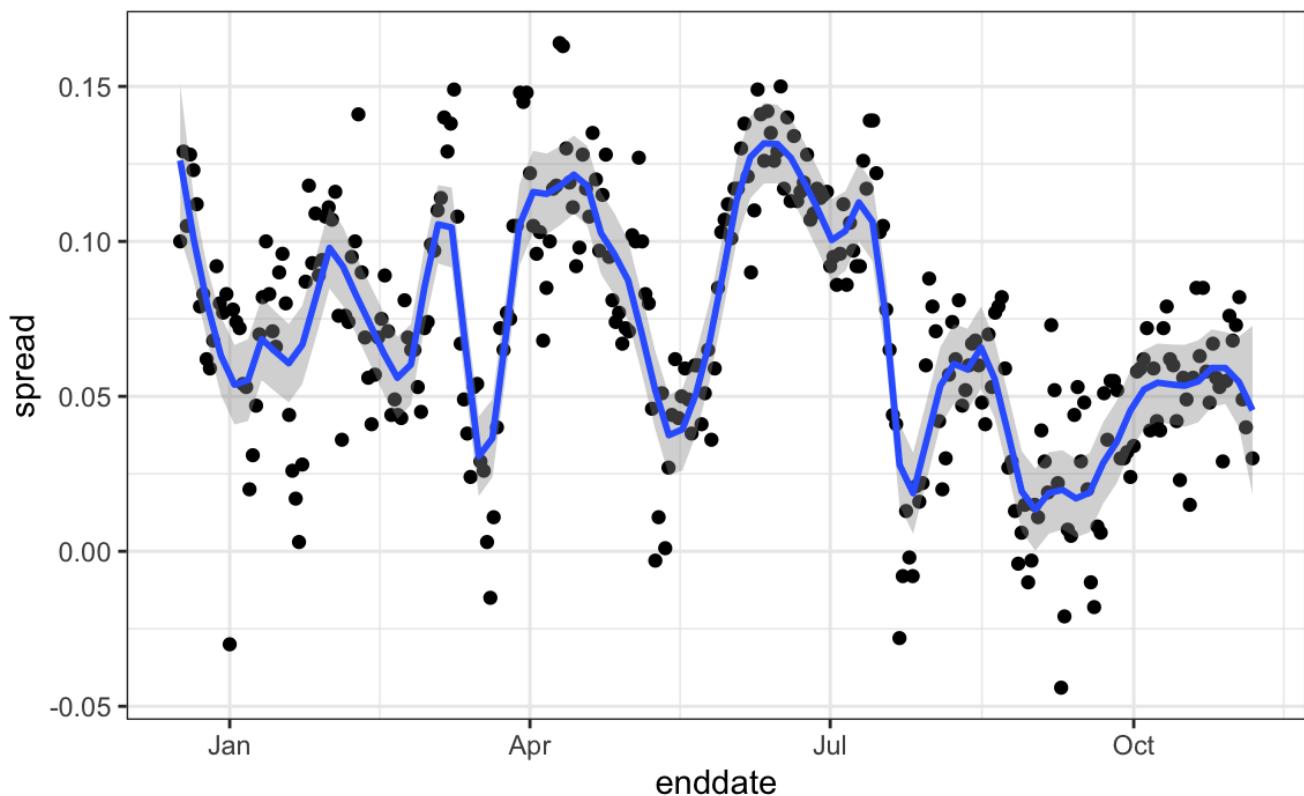


Figure 6:

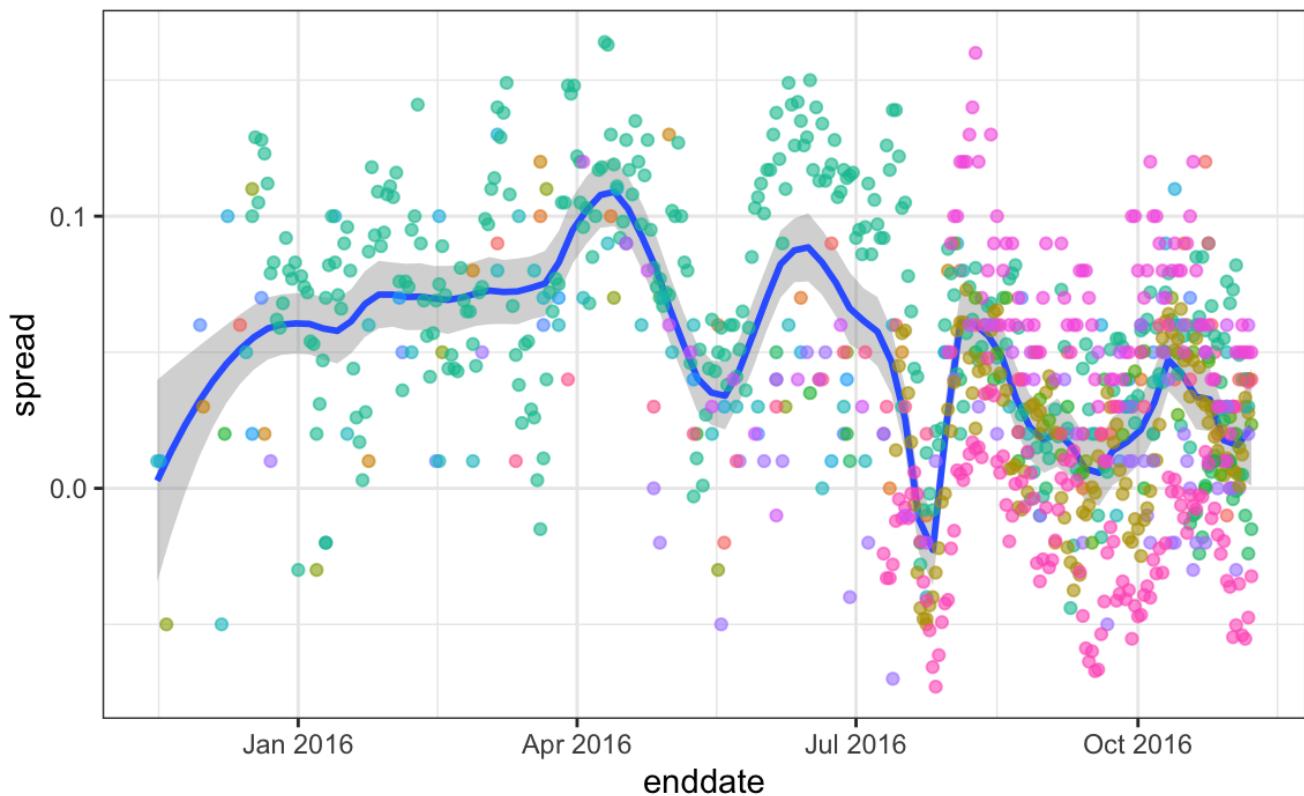


Figure 7:

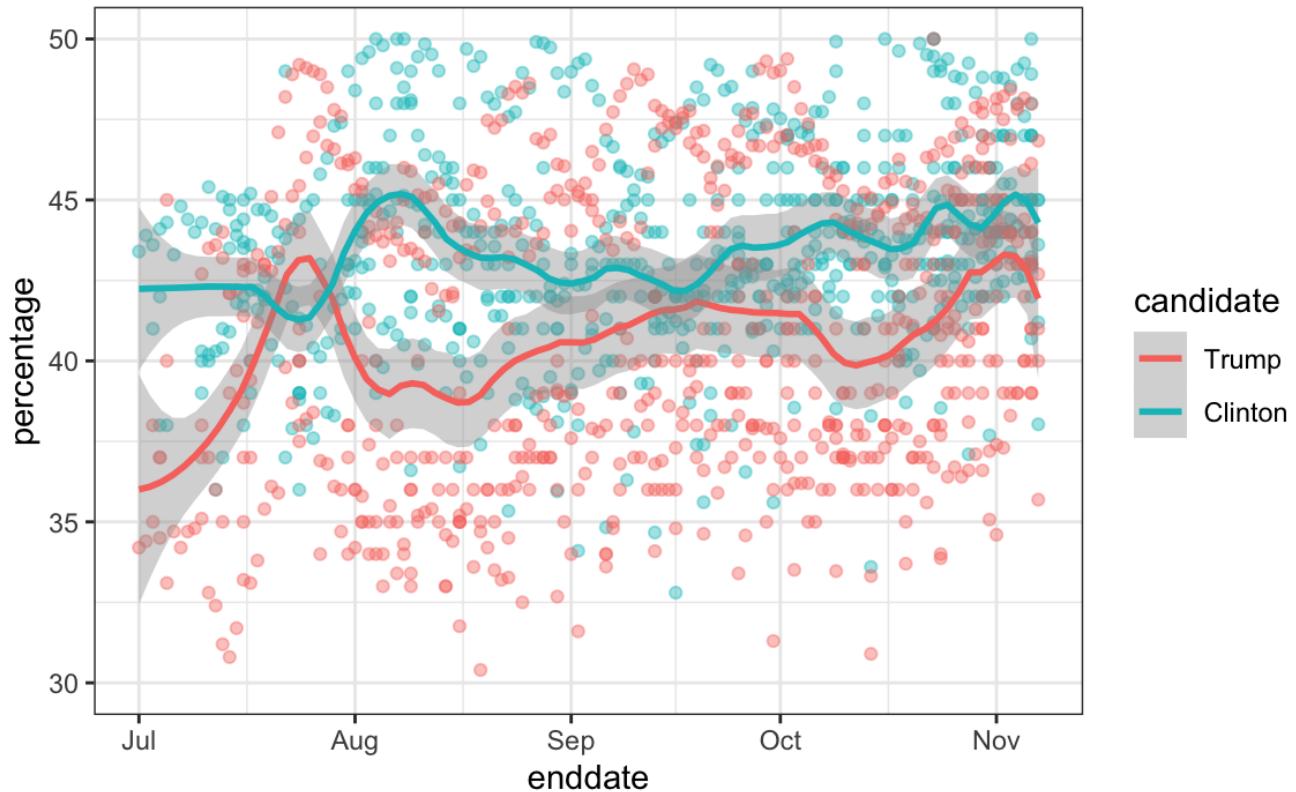


Figure 8:

### 13.7 Exercises

1. Create this table:

```
library(tidyverse)
library(dslabs)
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 election night result stored in `actual_spread`.

3. Repeat this, but show the proportion of hits for each pollster. Consider 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 with the election night result.
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 related to the *general bias* described in Section 13.2. 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 explore concepts related to random effects and mixed models.
11. In April 2013, José Iglesias, a professional baseball player was starting his career. He was performing exceptionally well, with an excellent *batting average* (AVG) of .450. The batting average statistic is one way of measuring success. Roughly speaking, it tells us the success rate when batting. José had 9 successes out of 20 tries. An AVG of .450 means José has been successful 45% of the times he has batted, which is rather high historically speaking. In fact, no one has finished a season with an AVG of .400 or more since Ted Williams did it in 1941! We want to predict José's batting average at the end of the season after players have had about 500 tries or *at bats*. With the frequentist techniques, we have no choice but to predict that his AVG will be .450 at the end of the season. Compute a confidence interval for the success rate.
12. Despite the frequentist prediction of  $\langle .450 \rangle$ , not a single baseball enthusiast would make this prediction. Why is this? One reason is that they know the estimate has much uncertainty. However, the main reason is that they are implicitly using a hierarchical model that factors in information from years of following baseball. Use the following code to explore the distribution of batting averages in the three seasons prior to 2013, and describe what this tells us.
13. 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. 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,  $\langle \mu \rangle$ . Then, we see 20 random outcomes with success probability  $\langle \mu \rangle$ . What model would you use for the first level of your hierarchical model?
14. Describe the second level of the hierarchical model.
15. Apply the hierarchical model to José's data. Suppose we want to predict his innate ability in the form of his *true* batting average  $\langle \mu \rangle$ . Write down the distributions of the hierarchical model.
16. We now are ready to compute a the distribution of  $\langle \mu \rangle$  conditioned on the observed data  $\langle \bar{X} \rangle$ . Compute the expected value of  $\langle \mu \rangle$  given the current average  $\langle \bar{X} \rangle$ , and provide an intuitive explanation for the mathematical formula.
17. We started with a frequentist 95% confidence interval that ignored data from other players and summarized just José's data: .450  $\pm$  0.220. Construct a credible interval for  $\langle \mu \rangle$  based on the hierarchical model.
18. The 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 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 |

Which of the two approaches provided a better prediction?

---

1. <https://www.nytimes.com/interactive/2016/upshot/presidential-polls-forecast.html>
2. <https://fivethirtyeight.com/features/trump-is-just-a-normal-polling-error-behind-clinton/>
3. <https://projects.fivethirtyeight.com/2016-election-forecast/>

# Introduction to Data Science - Linear Models

Rafael A. Irizarry

Up to this point, this book has focused mainly on datasets consisting of a single variable. However, in data analyses challenges, it is very common to be interested in the relationship between two or more variables. In this part of the book we introduce *linear models*, a general framework that unifies approaches used for analyzing association between variables, such as simple and multivariate regression, treatment effect models, and association test. We will illustrate these using case studies related to understanding if height is hereditary, described in detail in Chapter Chapter 14, using data to build a baseball team on a budget, described in detail in Chapter Chapter 15, determining if a high-fat diet makes mice heavier, described in detail in Chapter Chapter 17, and examining if there is gender bias in research funding in the Netherlands, described in detail in Chapter Chapter 18.

# Introduction to Data Science - 14 Regression

Rafael A. Irizarry

## 14.1 Case study: is height hereditary?

To understand the concepts of correlation and simple regression, 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, as well as many other circumstances.

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/>.

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)

set.seed(1983)
galton_heights <- GaltonFamilies |>
 filter(gender == "male") |>
 group_by(family) |>
 sample_n(1) |>
 ungroup() |>
 select(father, childHeight) |>
 rename.son = childHeight)
```

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 × 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)
```

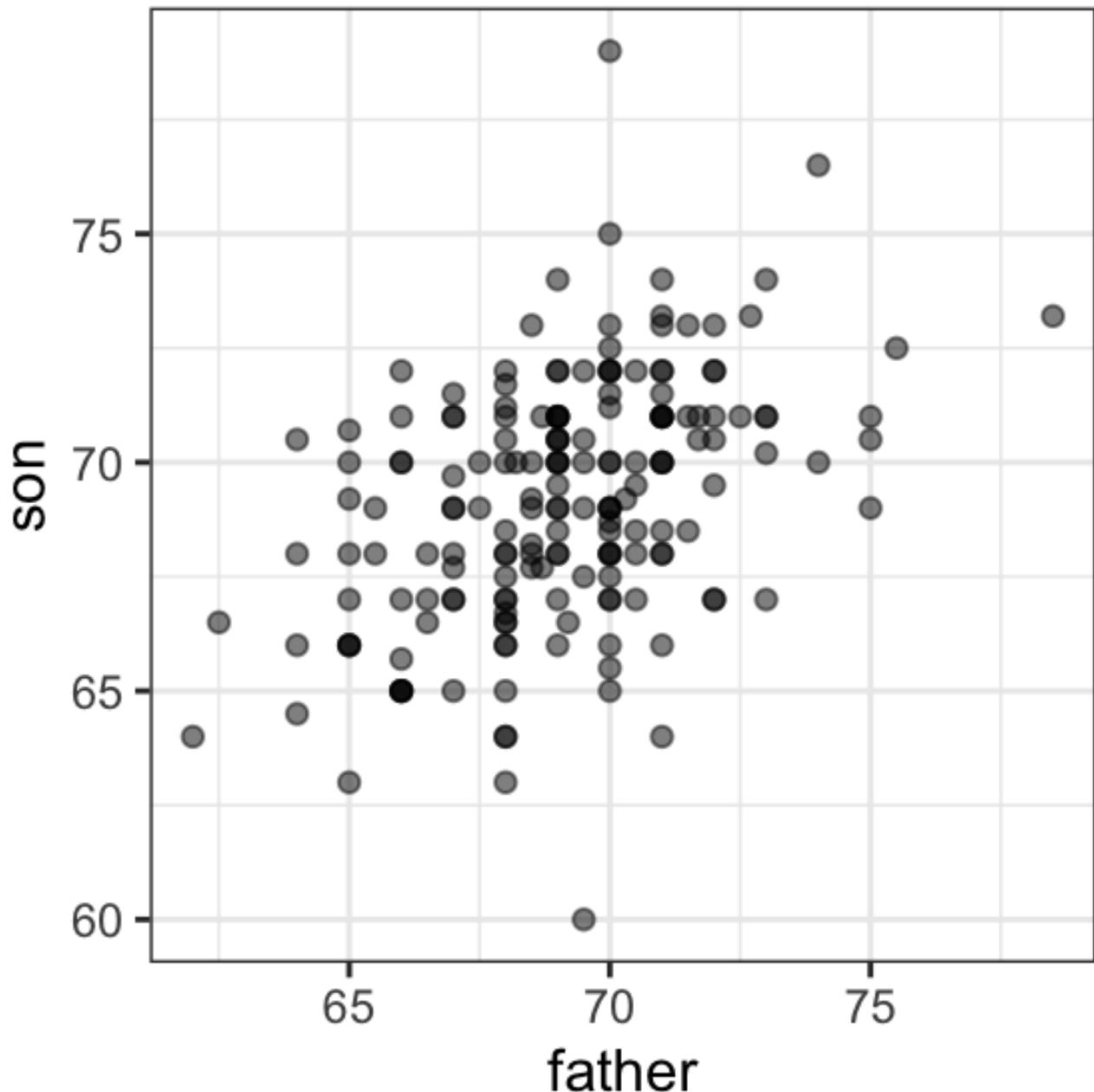


Figure 1:

We will learn that the correlation coefficient is an informative summary of how two variables move together and then motivate simple regression by noting how this can be used to predict one variable using the other.

## 14.2 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 for  $(r)$ ,  $(\rho)$  is commonly used in statistics books to denote the correlation. It is not a coincidence that  $(r)$  is the first letter in “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:

$$\left[ \rho = \frac{\sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right) \left( \frac{y_i - \mu_y}{\sigma_y} \right)}{\sqrt{\sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right)^2} \sqrt{\sum_{i=1}^n \left( \frac{y_i - \mu_y}{\sigma_y} \right)^2}} = \frac{\sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right) \left( \frac{y_i - \mu_y}{\sigma_y} \right)}{\sqrt{n} \sigma_x \sigma_y} = \frac{\sum_{i=1}^n \left( \frac{x_i - \mu_x}{\sigma_x} \right) \left( \frac{y_i - \mu_y}{\sigma_y} \right)}{\sqrt{n} \sigma_x \sigma_y} = 1 \right]$$

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 between -1 and 1. The correlation, computed with the function `cor`, between father and son's heights is about 0.5:

```
galton_heights |> summarize(r = cor(father, son)) |> pull(r)
#> [1] 0.433
```

The function `cor(x, y)` computes the sample correlation, which divides the sum of products by `length(x)-1` rather than `length(x)`. The rationale for this is akin to the reason we divide by `length(x)-1` when computing the sample standard deviation `sd(x)`. Namely, this adjustment helps account for the degrees of freedom in the sample, which is necessary for unbiased estimates.

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:

### 14.2.1 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)) |>
```

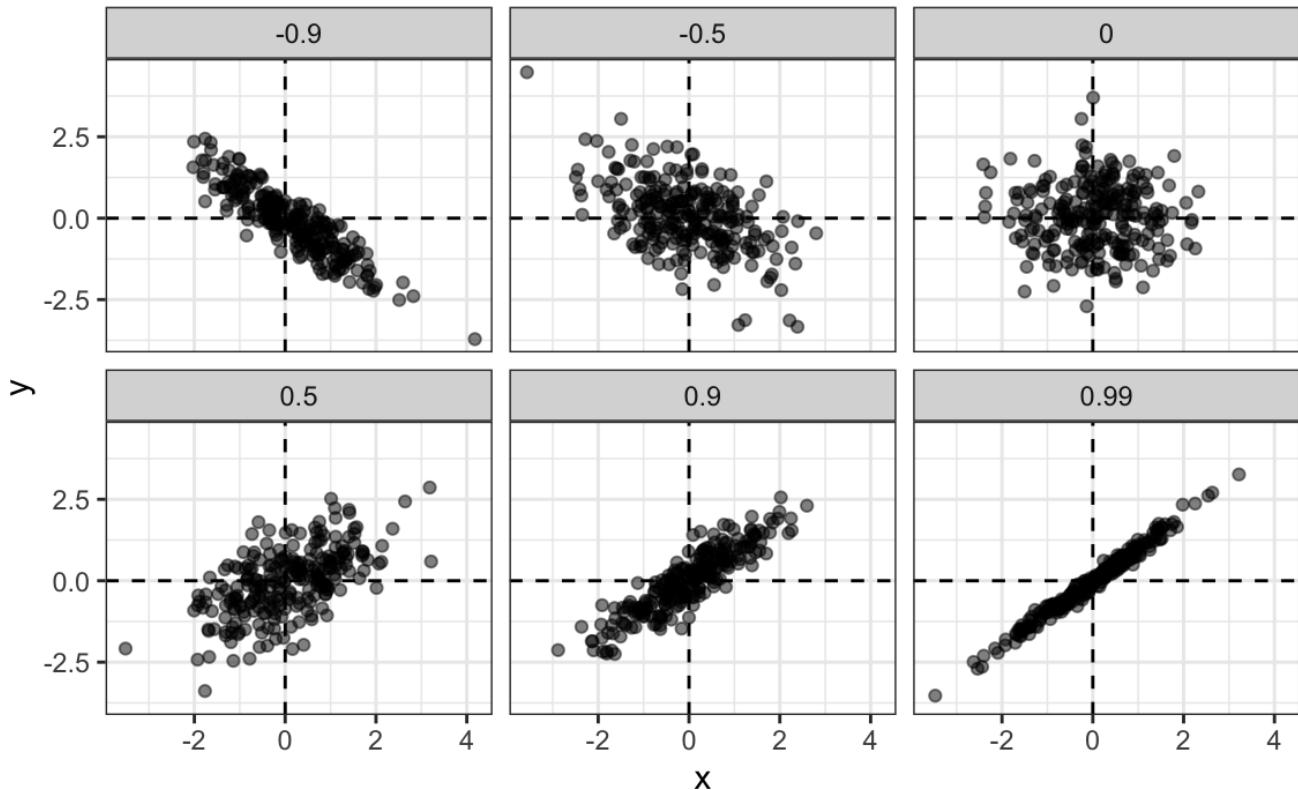


Figure 2:

```
pull(r)
})
hist(R, breaks = 20)
```

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.

## Histogram of R

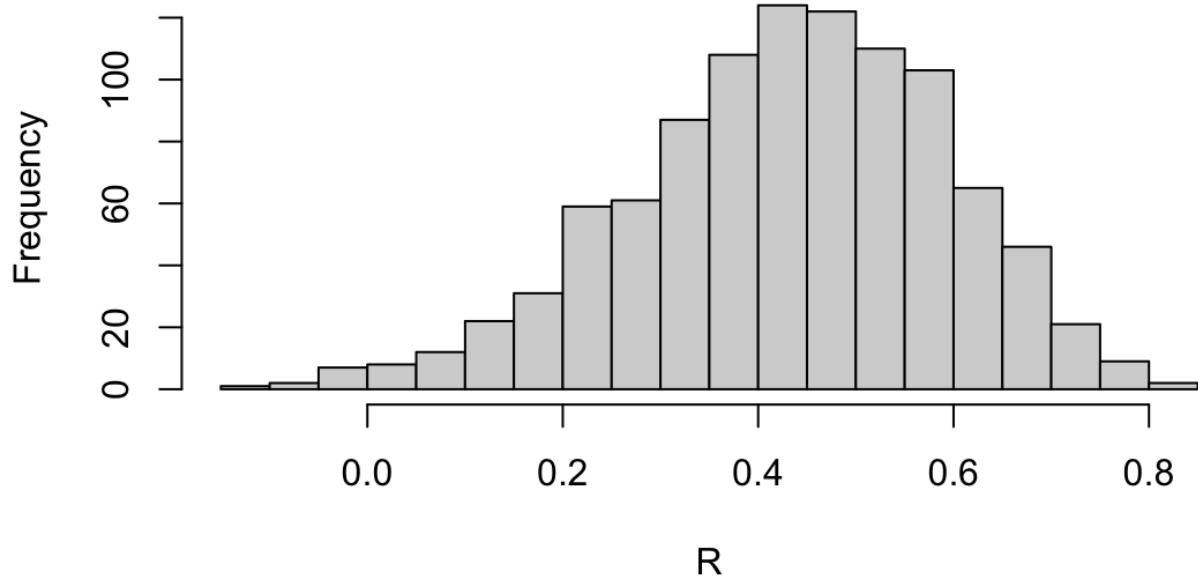


Figure 3:

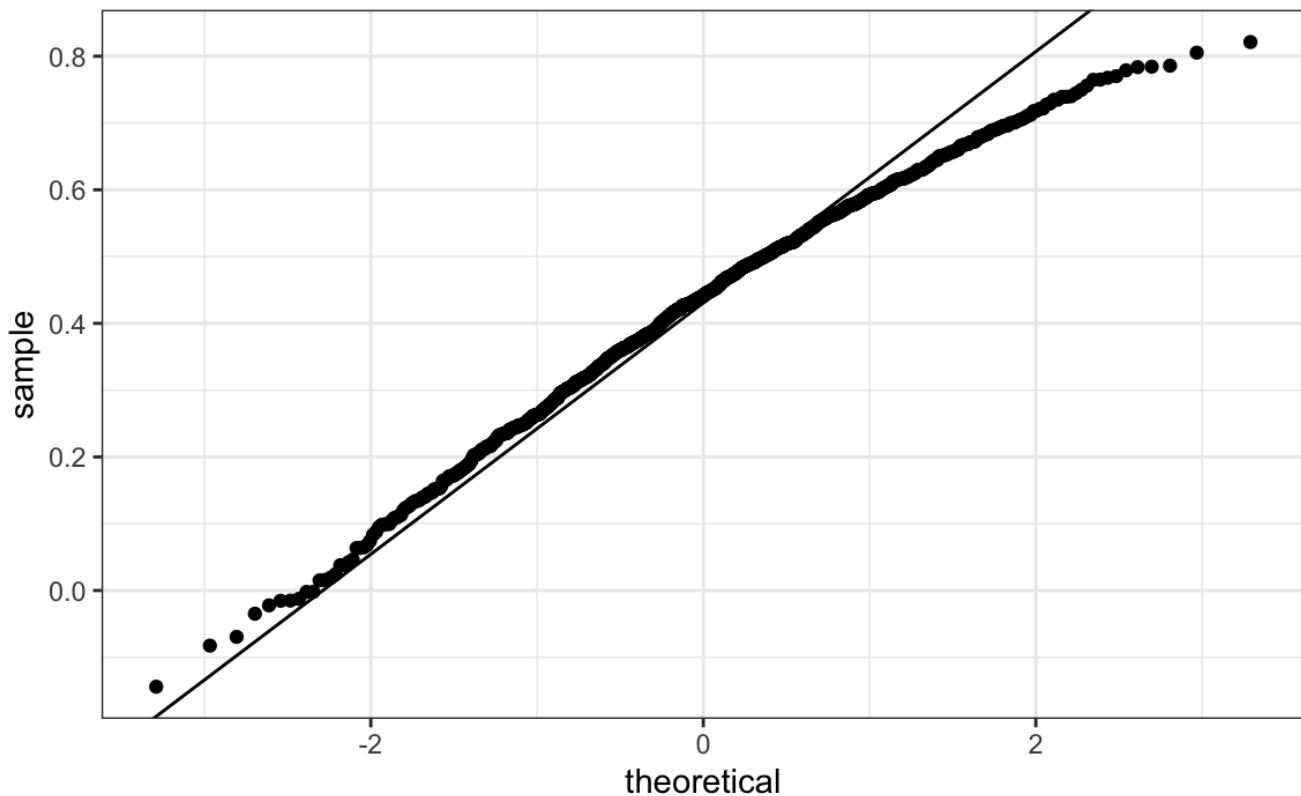


Figure 4:

### 14.2.2 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:

```
#> `geom_smooth()` using formula = 'y ~ x'
```

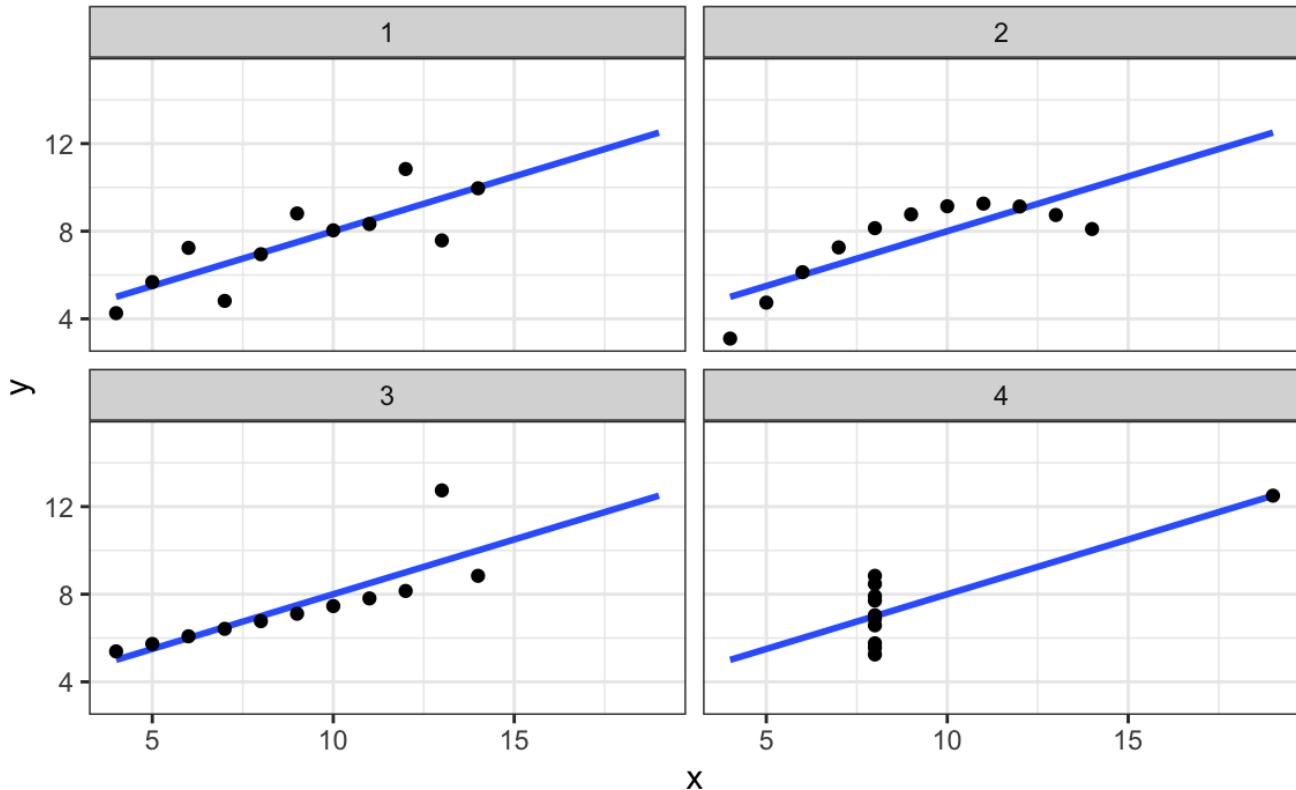


Figure 5:

Correlation is only meaningful in a particular context. To help us understand when correlation is meaningful as a summary statistic, we 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.

### 14.3 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. To provide a mathematical description of conditioning, consider that 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:

$$\mathbb{E}\{Y \mid X = x\}$$

with  $\{x\}$  representing the fixed value that defines that subset, for example 72 inches. Similarly, we denote the standard deviation of the strata with:

$$\text{SD}\{Y \mid X = x\} = \sqrt{\text{Var}\{Y \mid X = x\}}$$

Because the conditional expectation  $\mathbb{E}(Y \mid X = x)$  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  $\mathbb{E}(Y \mid X = 72)$  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 observations with similar value 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  $(72.0 - 69.1)/2.5 = 1.1$  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 inches, 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.

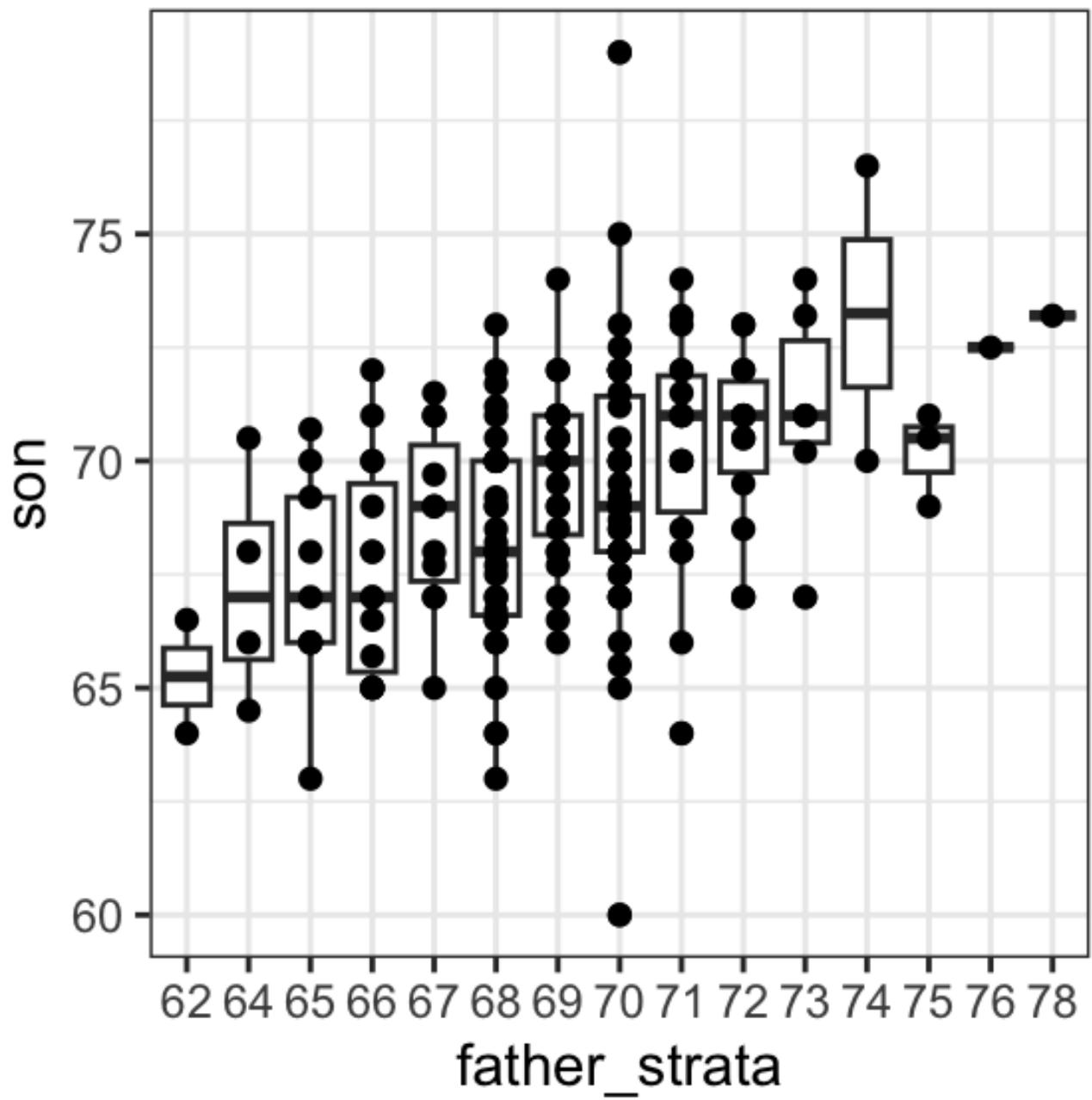


Figure 6:

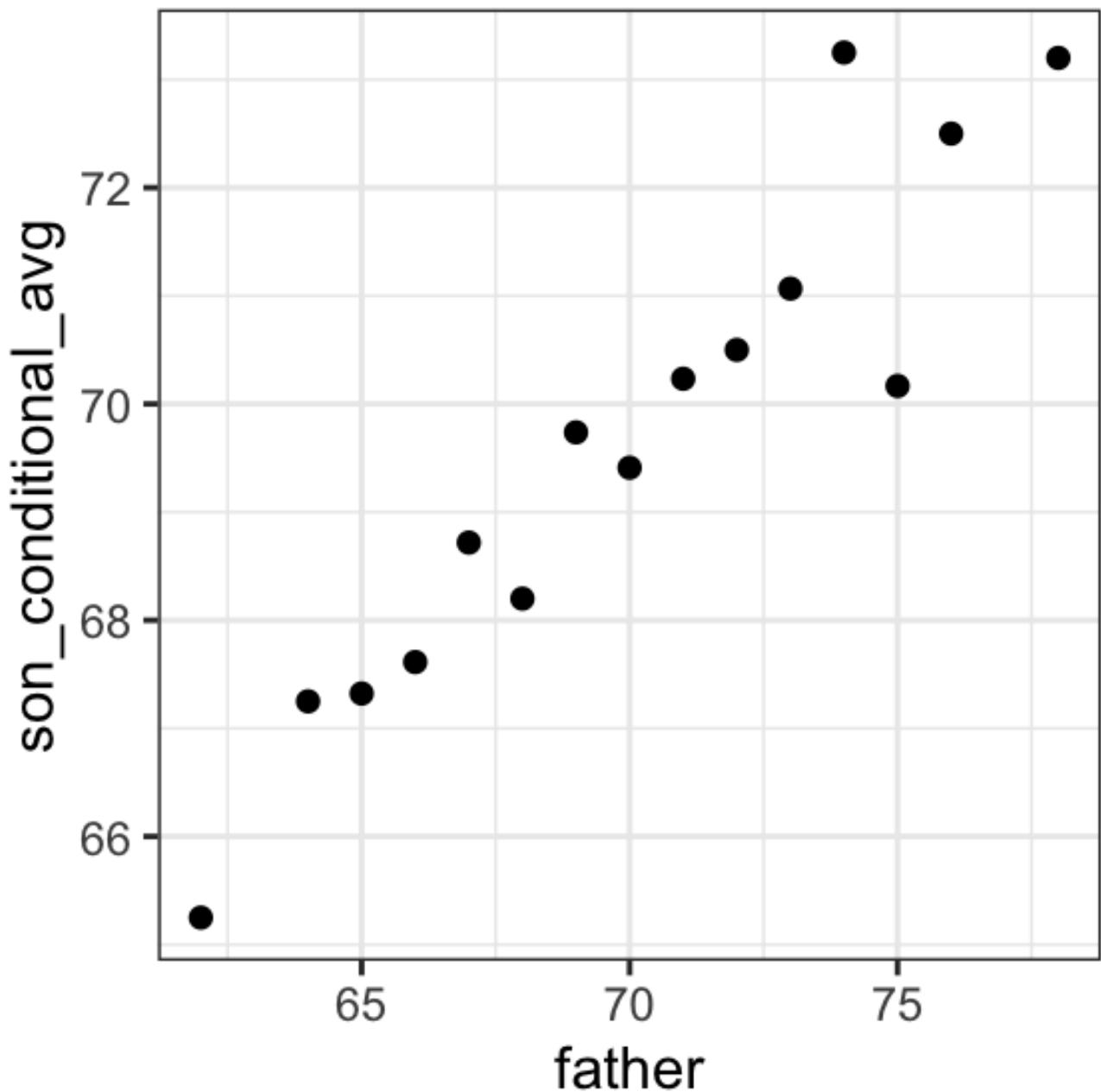


Figure 7:

## 14.4 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  $(\mu_X)$ , our prediction  $(\hat{Y})$  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:

$$[\left( \frac{\hat{Y} - \mu_Y}{\sigma_Y} \right) = \rho \left( \frac{x - \mu_X}{\sigma_X} \right)]$$

We can rewrite it like this:

$$[\hat{Y} = \mu_Y + \rho \left( \frac{x - \mu_X}{\sigma_X} \right) \sigma_Y]$$

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:

$$[\hat{Y} = b + mx \text{ with slope } m = \rho \frac{\sigma_y}{\sigma_x} \text{ and intercept } b = \mu_y - \rho \mu_x]$$

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)
```

## 14.5 Regression improves precision

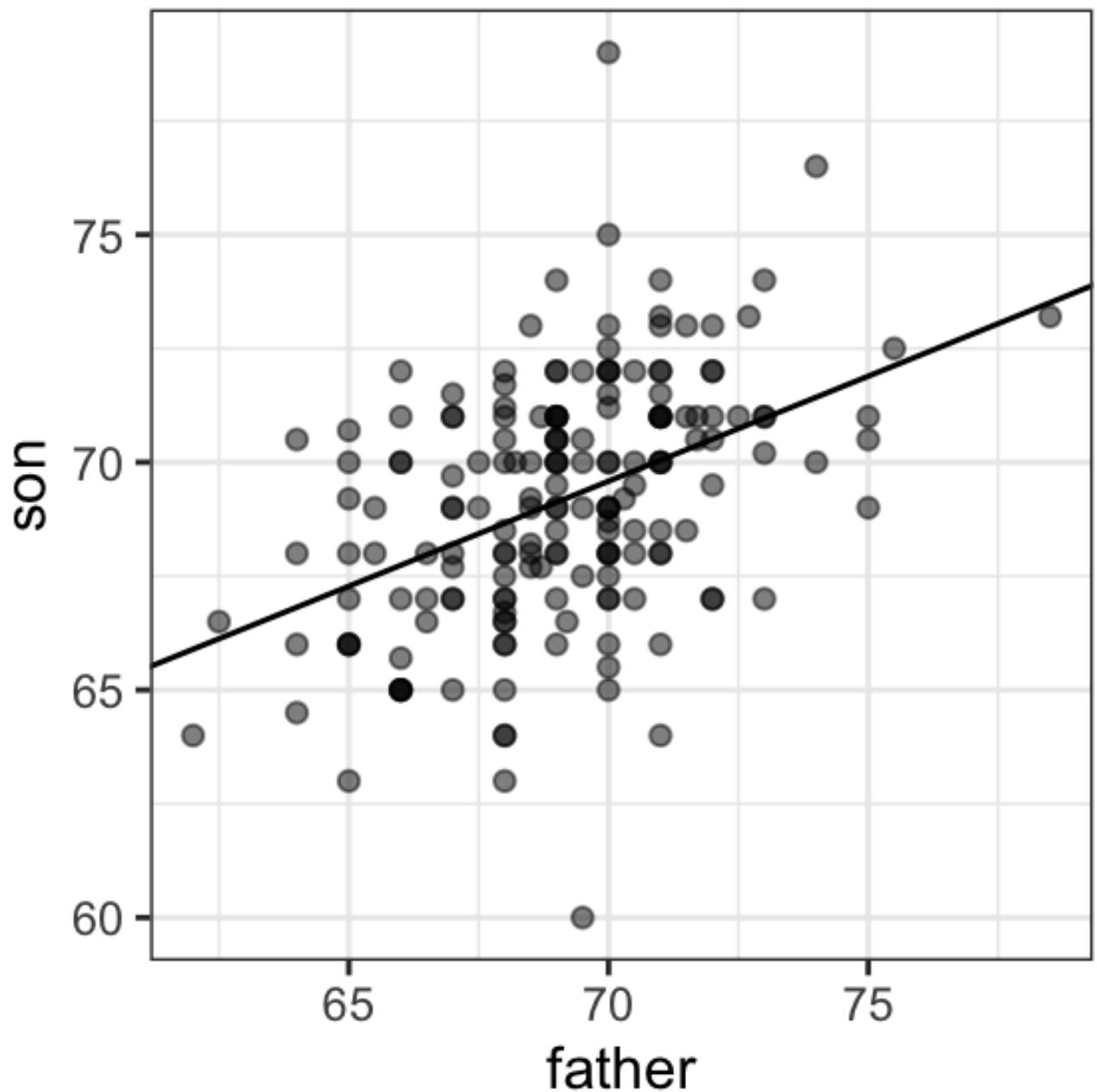
Let's compare the two approaches to prediction that we have presented:

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(optional_avg, na.rm = TRUE)
#> [1] 70.5
mean(regression_prediction)
#> [1] 70.5

```

The standard error for the regression prediction is substantially smaller:

```

sd(optional_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.

## 14.6 Bivariate normal distribution

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 in which the correlation is not a useful summary. But there are many real-life examples.

The main way we motivate appropriate use of correlation as a summary, involves 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 14.2, 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.

When three or more variables have the property that each pair is bivariate normal, we say the variables follow a *multivariate* normal distribution or that they are *jointly* 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)
```

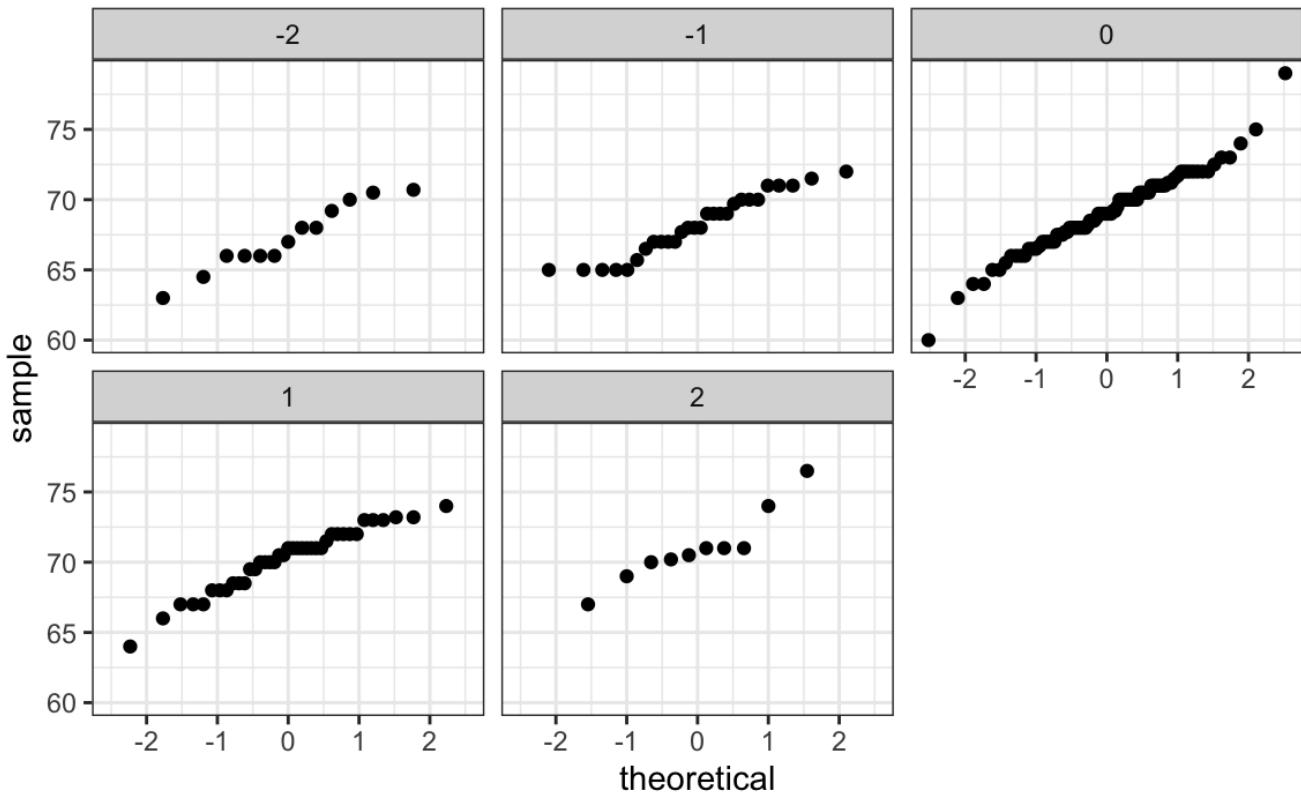


Figure 9:

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  $\langle x \rangle$ , the expected value of the  $\langle Y \rangle$  in pairs for which  $\langle X=x \rangle$  is:

$$\langle E(Y | X=x) \rangle = \mu_Y + \rho \frac{x - \mu_X}{\sigma_X} \sigma_Y$$

This is the regression line, with slope  $\rho \frac{\sigma_Y}{\sigma_X}$  and intercept  $\mu_Y - \rho \frac{\sigma_Y}{\sigma_X} \mu_X$ . It is equivalent to the regression equation we showed earlier which can be written like this:

$$\langle Y | X=x \rangle = \mu_Y + \rho \frac{x - \mu_X}{\sigma_X} \sigma_Y$$

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  $\langle Y \rangle$  given we know the value of  $\langle X \rangle$ , is given by the regression line.

## 14.7 Variance explained

The bivariate normal theory also tells us that the standard deviation of the *conditional* distribution described above is:

$$\text{SD}(Y | X=x) = \sigma_Y \sqrt{1-\rho^2}$$

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.

## 14.8 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  $(Y | X=x) = 37.3 + 0.46x$ .

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:  $(X = \{ Y | X=x\}) = 37.3 - 0.5x$ .

We need to compute  $(X | Y=y)$ . Since the data is approximately bivariate normal, the theory described earlier 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  $(X | Y=y) = 40.9 + 0.41y$ . Again, we see regression to the average: the prediction for the father is closer to the father average than the son heights  $(y)$  is to the son average.

Here is a plot showing the two regression lines, with blue for the predicting son heights with father heights, and red for predicting father heights with son heights:

```
galton_heights |>
 ggplot(aes(father, son)) +
 geom_point(alpha = 0.5) +
 geom_abline(intercept = b_1, slope = m_1, col = "blue") +
 geom_abline(intercept = -b_2/m_2, slope = 1/m_2, col = "red")
```

## 14.9 Linear models

We are now ready to understand the title of this part of the book. Specifically, the connection between regression and *linear models*. We have described how, if data is bivariate normal, then the conditional expectations follow the regression line. The fact that the conditional expectation is a line is not an extra assumption, but rather a derived result. However, in practice it is common to explicitly write down a model that describes the relationship between two or more variables using a *linear model*.

We note that *linear* here does not refer to lines exclusively, but rather to the fact that the conditional expectation is a linear combination of known quantities. In mathematics, when we multiply each variable by a constant and then add them together, we say we formed a *linear combination* of the variables. For example,  $(3x - 4y + 5z)$  is a linear combination of  $(x)$ ,  $(y)$ , and  $(z)$ . We can also add a constant so  $(2 + 3x - 4y + 5z)$  is also a linear combination of  $(x)$ ,  $(y)$ , and  $(z)$ .

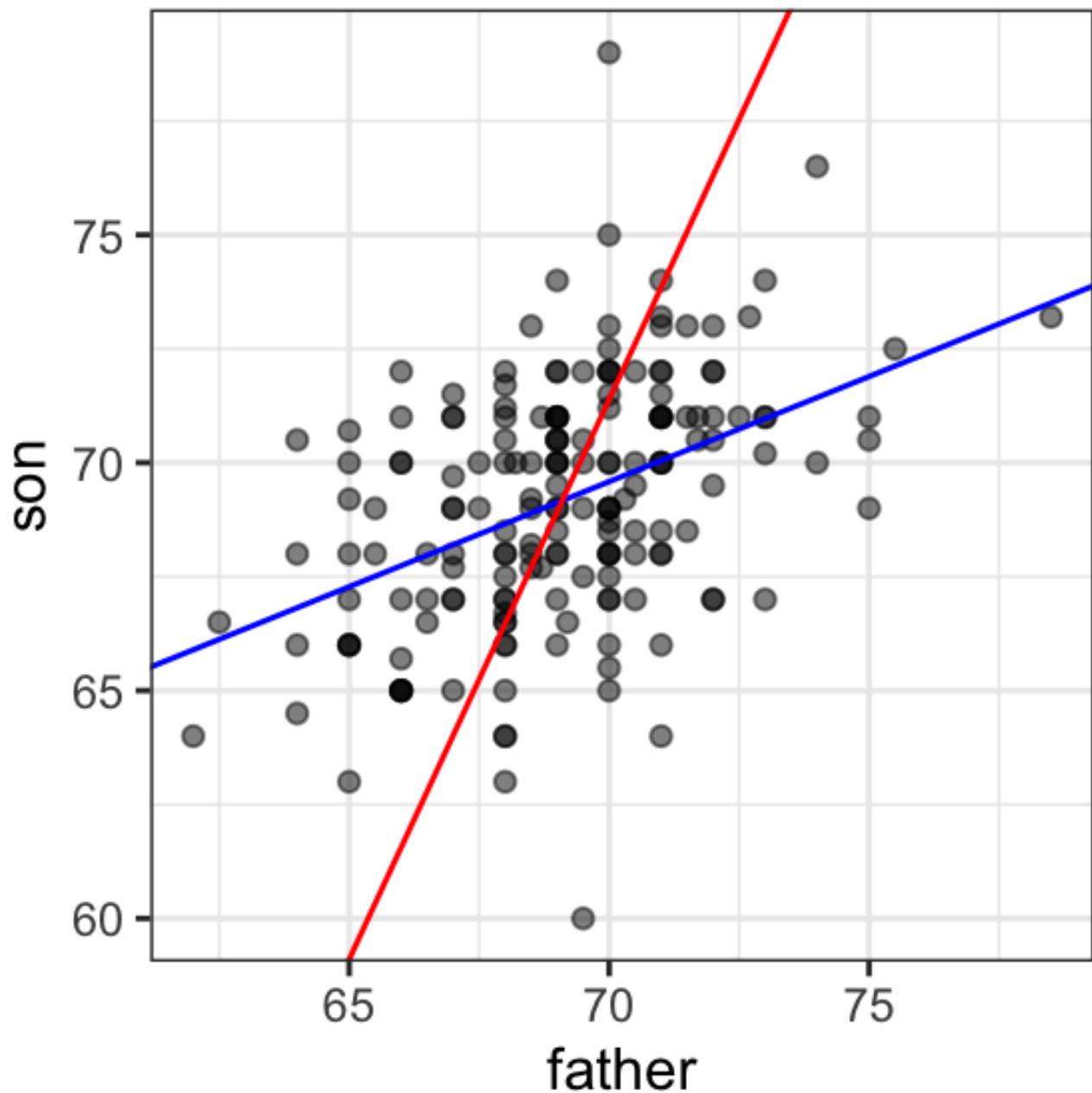


Figure 10:

We previously described how if  $(X)$  and  $(Y)$  are bivariate normal, then if we look at only the pairs with  $(X=x)$ , then  $(Y \mid X=x)$  follows a normal distribution with expected value  $(\mu_Y + \rho \frac{x-\mu_X}{\sigma_X} \sigma_Y)$ , which is a linear function of  $(x)$ , and standard deviation  $\sqrt{(\sigma_Y^2(1-\rho^2))}$  that does not depend on  $(x)$ . Note that if we write:

$$[ Y = \beta_0 + \beta_1 x + \varepsilon ]$$

If we assume  $(\varepsilon)$  follows a normal distribution with expected value 0 and fixed standard deviation, then  $(Y)$  has the same properties as the regression setup gave us: it follows a normal distribution, the expected value is a linear function  $(x)$ , and the standard deviation does not depend on  $(x)$ .

In statistical textbooks, the  $(\varepsilon)$ s are referred to as “errors,” which originally represented measurement errors in the initial applications of these models. These errors were associated with inaccuracies in measuring height, weight, or distance. However, the term “error” is now used more broadly, even when the  $(\varepsilon)$ s do not necessarily signify an actual error. For instance, in the case of height, if someone is 2 inches taller than expected, based on their parents’ height, those 2 inches should not be considered an error. Despite its lack of descriptive accuracy, the term “error” is employed to elucidate the unexplained variability in the model, unrelated to other included terms.

If we were to specify a linear model for Galton’s data, we would denote the  $(N)$  observed father heights with  $(x_1, \dots, x_n)$ , then we model the  $(N)$  son heights we are trying to predict with:

$$[ Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, i=1,\dots,N ]$$

Here  $(x_i)$  is the father’s height, which is fixed (not random) due to the conditioning, and  $(Y_i)$  is the random son’s height that we want to predict. We can further assume that  $(\varepsilon_i)$  are independent from each other and all have the same standard deviation.

In the above model, we know the  $(x_i)$ , but to have a useful model for prediction, we need  $(\beta_0)$  and  $(\beta_1)$ . We estimate these from the data. Once we do this, we can predict son’s heights for any father’s height  $(x)$ . We show how to do this in the next section.

Although this model is exactly the same one we derived earlier by assuming bivariate normal data, a somewhat nuanced difference is that, in the first approach, we assumed the data was bivariate normal and the linear model was derived, not assumed. In practice, linear models are just assumed without necessarily assuming normality: the distribution of the  $(\varepsilon)$ s is not necessarily specified. Nevertheless, if your data is bivariate normal, the above linear model holds. If your data is not bivariate normal, then you will need to have other ways of justifying the model.

One reason linear models are popular is that they are *interpretable*. In the case of Galton’s data, we can interpret the data like this: due to inherited genes, the son’s height prediction grows by  $(\beta_1)$  for each inch we increase the father’s height  $(x)$ . Because not all sons with fathers of height  $(x)$  are of equal height, we need the term  $(\varepsilon)$ , which explains the remaining variability. This remaining variability includes the mother’s genetic effect, environmental factors, and other biological randomness.

Given how we wrote the model above, the intercept  $(\beta_0)$  is not very interpretable, as it is the predicted height of a son with a father with no height. Due to regression to the mean, the prediction will usually be a bit larger than 0. To make the slope parameter more interpretable, we can rewrite the model slightly as:

$$[ Y_i = \beta_0 + \beta_1 (x_i - \bar{x}) + \varepsilon_i, i=1,\dots,N ]$$

with  $(\bar{x} = 1/N \sum_{i=1}^N x_i)$  the average of the  $(x)$ . In this case,  $(\beta_0)$  represents the height when  $(x_i = \bar{x})$ , which is the height of the son of an average father.

Later, specifically in Sections Chapter 15 and @treatment-effect-models, we will see how the linear model representation permits us to use the same mathematical frameworks in other contexts and to achieve more complicated goals than predict one variable from another.

## 14.10 Least Squares Estimates

For linear models to be useful, we have to estimate the unknown  $(\beta)$ s. The standard approach is to find the values that minimize the distance of the fitted model to the data. Specifically, we find the  $(\beta)$ s that minimize

the least squares (LS) equation show below. For Galton's data, the LS equation looks like this:

$$\text{RSS} = \sum_{i=1}^n \left( y_i - (\beta_0 + \beta_1 x_i) \right)^2$$

The quantity we try to minimize is called the residual sum of squares (RSS).

Once we find the values that minimize the RSS, we will call the values the least squares estimates (LSE) and denote them by placing a *hat* over the parameters. In our example we use  $\hat{\beta}_0$  and  $\hat{\beta}_1$ .

We will demonstrate how we find these values using the previously defined `galton_heights` dataset. Let's start by writing a function that computes the RSS for any pair of values  $\beta_0$  and  $\beta_1$ .

```
rss <- function(beta0, beta1, data){
 resid <- galton_heights$son - (beta0 + beta1*galton_heights$father)
 return(sum(resid^2))
}
```

So for any pair of values, we get an RSS. Here is a plot of the RSS as a function of  $\beta_1$ , when we keep the  $\beta_0$  fixed at 25.

```
beta1 = seq(0, 1, length = nrow(galton_heights))
results <- data.frame(beta1 = beta1,
 rss = sapply(beta1, rss, beta0 = 25))
results |> ggplot(aes(beta1, rss)) + geom_line() +
 geom_line(aes(beta1, rss))
```

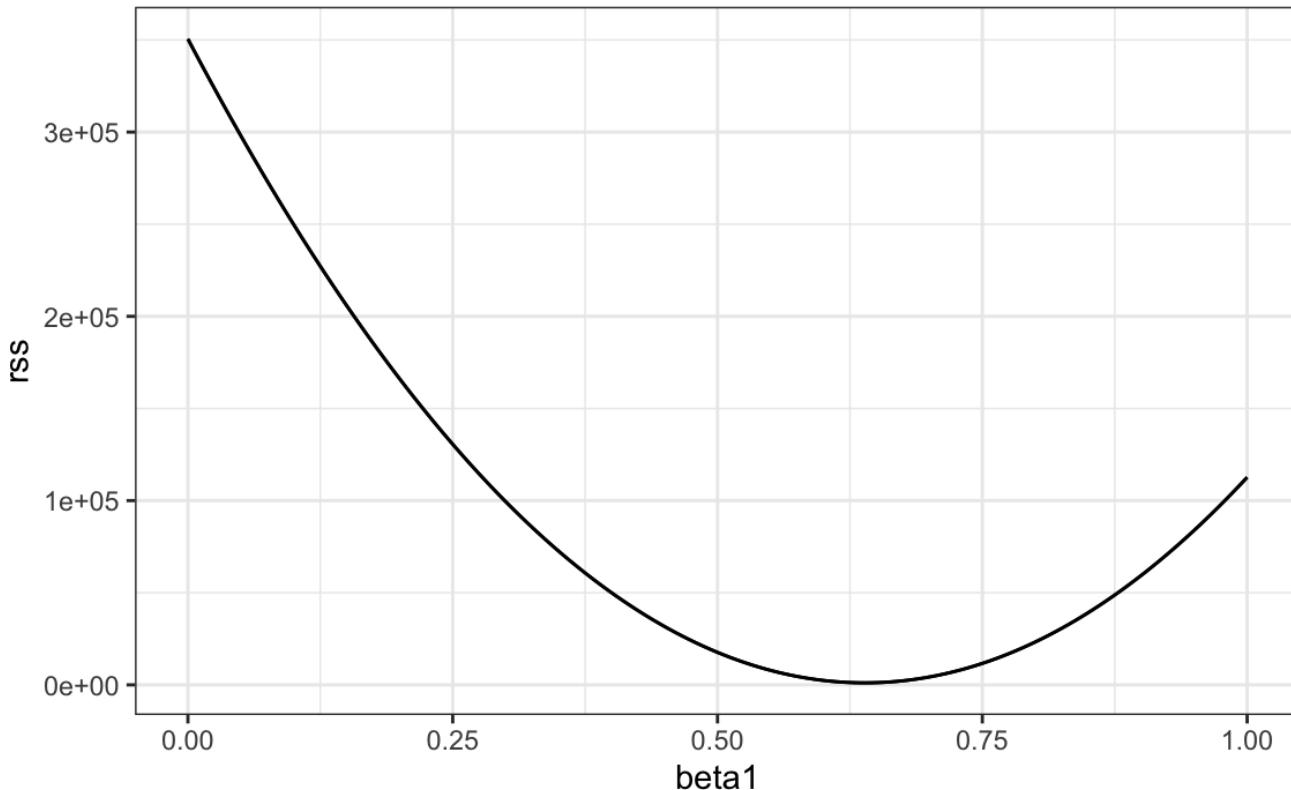


Figure 11:

We can see a clear minimum for  $\beta_1$  at around 0.65. However, this minimum for  $\beta_1$  is for when  $\beta_0 = 25$ , a value we arbitrarily picked. We don't know if  $(25, 0.65)$  is the pair that minimizes the equation across all possible pairs.

Trial and error is not going to work in this case. We could search for a minimum within a fine grid of  $\beta_0$  and  $\beta_1$  values, but this is unnecessarily time-consuming since we can use calculus. Specifically, we take

the partial derivatives, set them to 0, and solve for  $(\beta_1)$  and  $(\beta_2)$ . Of course, if we have many parameters, these equations can get rather complex. But there are functions in R that do these calculations for us. We will study these next. To learn the mathematics behind this, you can consult a book on linear models.

## 14.11 The `lm` function

In R, we can obtain the least squares estimates using the `lm` function. To fit the model:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

with  $(Y_i)$  being the son's height and  $(x_i)$  being the father's height, we can use this code to obtain the least squares estimates.

```
fit <- lm(son ~ father, data = galton_heights)
fit$coefficients
#> (Intercept) father
#> 37.288 0.461
```

The most common way we use `lm` is by using the character `~` to let `lm` know which is the variable we are predicting (left of `~`) and which we are using to predict (right of `~`). The intercept is added automatically to the model that will be fit.

The object `fit` includes more information about the fit. We can use the function `summary` to extract more of this information (not shown):

```
summary(fit)
#>
#> Call:
#> lm(formula = son ~ father, data = galton_heights)
#>
#> Residuals:
#> Min 1Q Median 3Q Max
#> -9.354 -1.566 -0.008 1.726 9.415
#>
#> Coefficients:
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 37.2876 4.9862 7.48 3.4e-12 ***
#> father 0.4614 0.0721 6.40 1.4e-09 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.45 on 177 degrees of freedom
#> Multiple R-squared: 0.188, Adjusted R-squared: 0.183
#> F-statistic: 40.9 on 1 and 177 DF, p-value: 1.36e-09
```

To understand some of the information included in this summary, we need to remember that the LSE are random variables. Mathematical statistics gives us some ideas of the distribution of these random variables.

In Chapter 15, after describing a more complex case study, we gain further insights into the application of regression in R.

## 14.12 LSE are random variables

The LSE is derived from the data  $(y_1, \dots, y_N)$ , which are a realization of random variables  $(Y_1, \dots, Y_N)$ . This implies that our estimates are random variables. To see this, we can run a Monte Carlo simulation in which we assume the son and father height data defines a population, take a random sample of size  $(N=50)$ , and compute the regression slope coefficient for each one:

```
B <- 1000
N <- 50
lse <- replicate(B, {
```

```

sample_n(galton_heights, N, replace = TRUE) |>
 lm(son ~ father, data = _) |>
 coef()
})
lse <- data.frame(beta_0 = lse[1,], beta_1 = lse[2,])

```

We can see the variability of the estimates by plotting their distributions:

```

#>
#> Attaching package: 'gridExtra'
#> The following object is masked from 'package:dplyr':
#>
#> combine

```

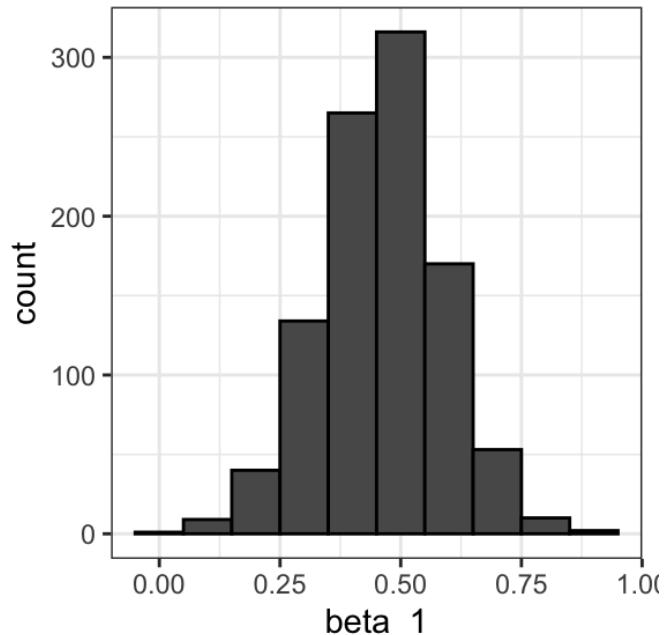
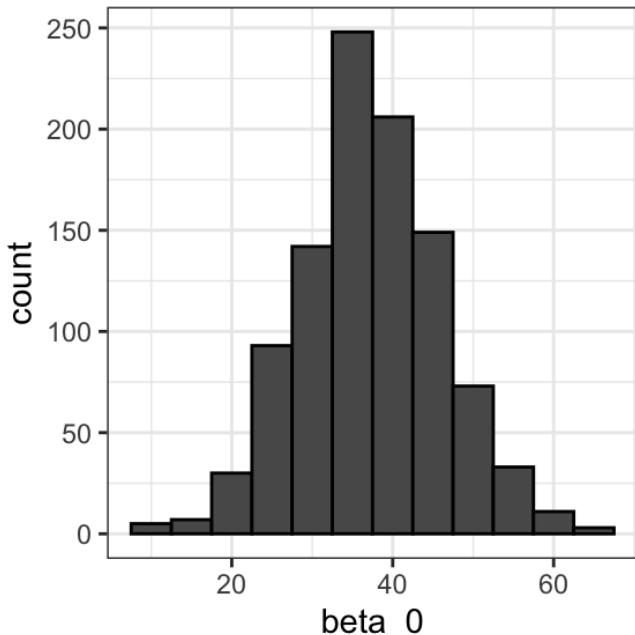


Figure 12:

The reason these look normal is because the central limit theorem applies here as well: for large enough  $N$ , the least squares estimates will be approximately normal with expected value  $\beta_0$  and  $\beta_1$ , respectively.

The standard errors are a bit complicated to compute, but mathematical theory does allow us to compute them and they are included in the summary provided by the `lm` function. The function `summary` shows us the standard error estimates:

```

sample_n(galton_heights, N, replace = TRUE) |>
 lm(son ~ father, data = _) |>
 summary() |>
 coef()
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 19.28 11.656 1.65 1.05e-01
#> father 0.72 0.169 4.25 9.79e-05

```

You can see that the standard errors estimates reported above are close to the standard errors from the simulation:

```

lse |> summarize(se_0 = sd(beta_0), se_1 = sd(beta_1))
#> se_0 se_1
#> 1 8.84 0.128

```

The `summary` function also reports t-statistics (`t value`) and p-values (`Pr(>|t|)`). The t-statistic is not actually

based on the central limit theorem, but rather on the assumption that the  $\hat{\beta}_0$ s follow a normal distribution. Under this assumption, mathematical theory tells us that the LSE divided by their standard error,  $(\hat{\beta}_0 / \text{SE}(\hat{\beta}_0))$  and  $(\hat{\beta}_1 / \text{SE}(\hat{\beta}_1))$ , follow a t-distribution with  $(N-p)$  degrees of freedom, with  $p$  the number of parameters in our model. In our example ( $p=2$ ), and the two p-values are obtained from testing the null hypothesis that  $(\beta_0 = 0)$  and  $(\beta_1 = 0)$ , respectively.

Remember that, as we described in Section 11.2.3, for large enough  $(N)$ , the CLT works and the t-distribution becomes almost the same as the normal distribution. Also, notice that we can construct confidence intervals, but we will soon learn about **broom**, an add-on package that makes this easy.

Although we do not show examples in this book, hypothesis testing with regression models is commonly used in epidemiology and economics to make statements such as “the effect of A on B was statistically significant after adjusting for X, Y, and Z”. However, several assumptions have to hold for these statements to be true.

### 14.13 Predicted values are random variables

Once we fit our model, we can obtain prediction of  $(Y)$  by plugging in the estimates into the regression model. For example, if the father's height is  $(x)$ , then our prediction  $(\hat{Y})$  for the son's height will be:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

When we plot  $(\hat{Y})$  versus  $(x)$ , we see the regression line.

Keep in mind that the prediction  $(\hat{Y})$  is also a random variable and mathematical theory tells us what the standard errors are. If we assume the errors are normal, or have a large enough sample size, we can use theory to construct confidence intervals as well. In fact, the **ggplot2** layer `geom_smooth(method = "lm")` that we previously used plots  $(\hat{Y})$  and surrounds it by confidence intervals:

```
galton_heights |> ggplot(aes(son, father)) +
 geom_point() +
 geom_smooth(method = "lm")
#> `geom_smooth()` using formula = 'y ~ x'
```

The R function `predict` takes an `lm` object as input and returns the prediction. If requested, the standard errors and other information from which we can construct confidence intervals is provided:

```
fit <- galton_heights |> lm(son ~ father, data = _)

y_hat <- predict(fit, se.fit = TRUE)

names(y_hat)
#> [1] "fit" "se.fit" "df" "residual.scale"
```

### 14.14 Diagnostic plots

When the linear model is assumed, rather than derived, all interpretations depend on the usefulness of the model. The `lm` function will fit the model and return summaries even when the model is wrong and not useful.

Visually inspecting residuals, defined as the difference between observed values and predicted values:

$r = Y - \hat{Y} = Y - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$ , and summaries of the residuals, is a powerful way to diagnose if the model is useful. Note that the residuals can be thought of estimates of the errors since:

$r = Y - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$ . In fact residuals are often denoted as  $(\hat{\epsilon})$ . This motivates several *diagnostic* plots. Because we observe  $(r)$ , but don't observe  $(\epsilon)$ , we based the plots on the residuals.

- Because the errors are assumed not to depend on the expected value of  $(Y)$ , a plot of  $(r)$  versus the fitted values  $(\hat{Y})$  should show no relationship.

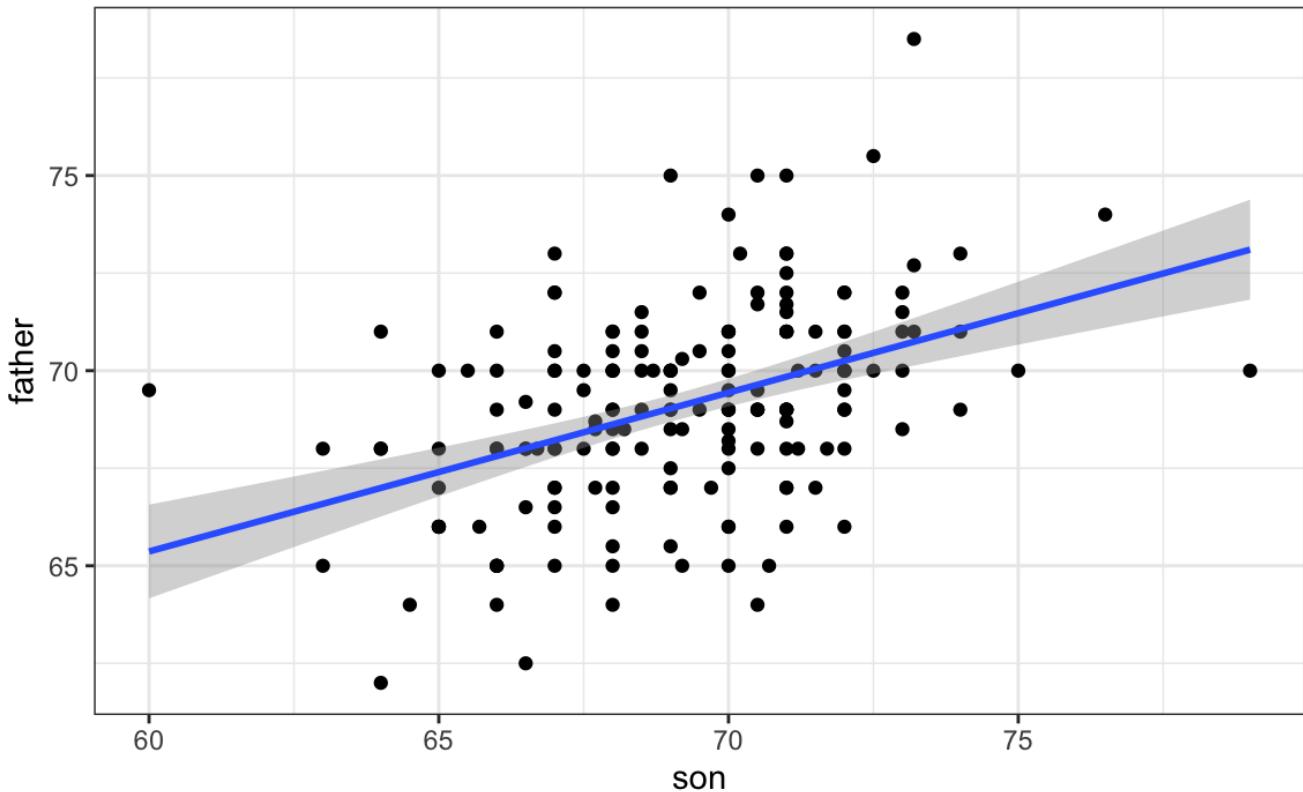


Figure 13:

2. In cases in which we assume the errors follow a normal distribution, a qqplot of standardized  $\backslash(r\backslash)$  should fall on a line when plotted against theoretical quantiles.
3. Because we assume the standard deviation of the errors is constant, if we plot the absolute value of the residuals, it should appear constant.

We prefer plots rather than summaries based on, for example, correlation because, as noted in Section @ascombe, correlation is not always the best summary of association. The function `plot` applied to an `lm` object automatically plots these.

```
plot(fit, which = 1:3)
```

This function can produce six different plots, and the argument `which` let's you specify which you want to see. You can learn more by reading the `plot.lm` help file. However, some of the plots are based on more advanced concepts beyond the scope of this book. To learn more, we recommend an advanced book on regression analysis.

In Chapter 15 and Chapter 17, we introduce data analysis challenges in which we may decide to not to include certain variables in the model. In these cases, an important diagnostic test to add checks if the residuals are related to variables not included in the model.

## 14.15 The regression fallacy

Wikipedia defines the *sophomore slump* as:

A sophomore slump or sophomore jinx or sophomore jitters refers to an instance in which a second, or sophomore, effort fails to live up to the standards of the first effort. It is commonly used to refer to the apathy of students (second year of high school, college or university), the performance of athletes (second season of play), singers/bands (second album), television shows (second seasons) and movies (sequels/prequels).

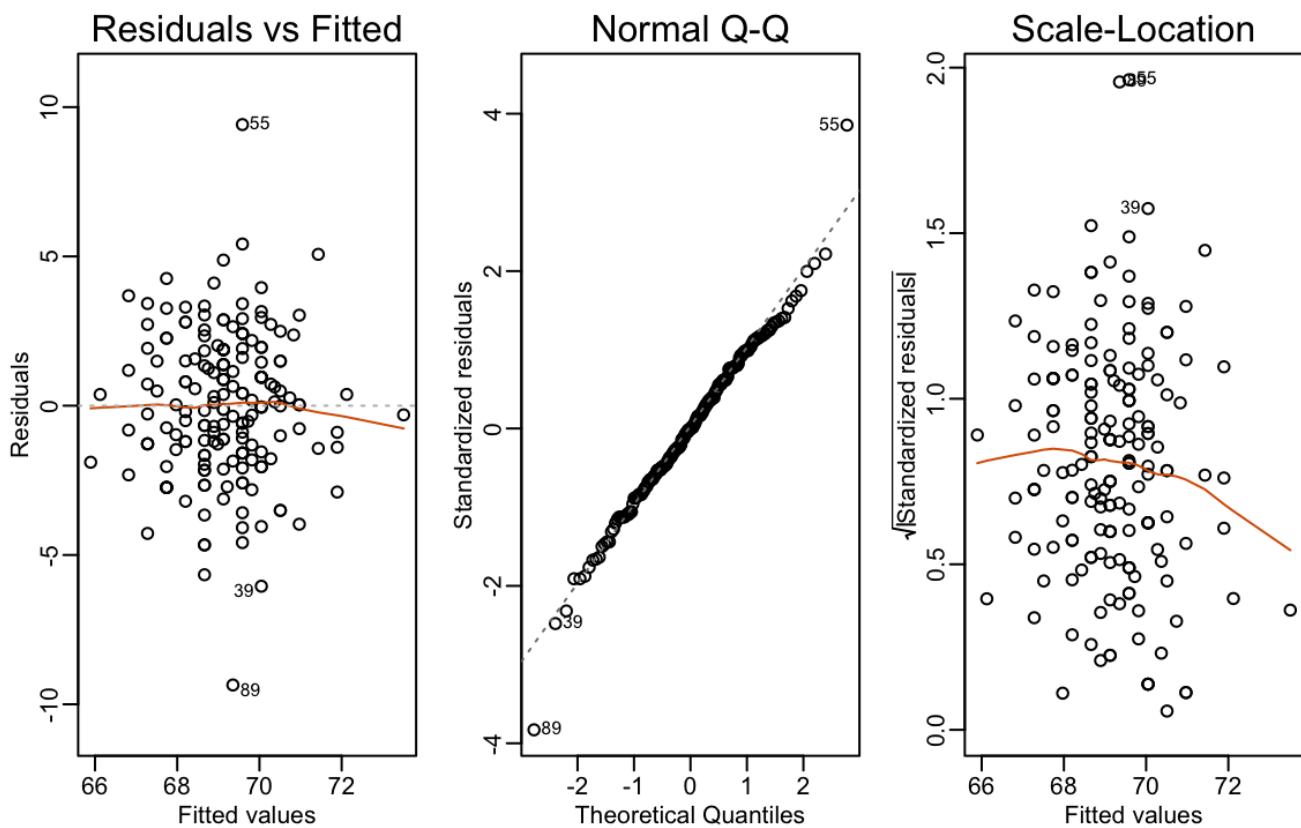


Figure 14:

In Major League Baseball, the rookie of the year (ROY) award is given to the first-year player who is judged to have performed the best. The *sophomore slump* phrase is used to describe the observation that ROY award winners don't do as well during their second year. For example, this Fox Sports article<sup>2</sup> asks "Will MLB's tremendous rookie class of 2015 suffer a sophomore slump?"

Does the data confirm the existence of a sophomore slump? Let's take a look. Examining the data for a widely used measure of success, the batting average, we see that this observation holds true for the top performing ROYs:

| nameFirst | nameLast | rookie_year | rookie | sophomore |
|-----------|----------|-------------|--------|-----------|
| Willie    | McCovey  | 1959        | 0.354  | 0.238     |
| Ichiro    | Suzuki   | 2001        | 0.350  | 0.321     |
| Al        | Bumbry   | 1973        | 0.337  | 0.233     |
| Fred      | Lynn     | 1975        | 0.331  | 0.314     |
| Albert    | Pujols   | 2001        | 0.329  | 0.314     |

In fact, the proportion of players that have a lower batting average during their sophomore year is 0.7037037.

So is it "jitters" or "jinx"? To answer this question, let's turn our attention to all the players that played the 2013 and 2014 seasons and batted more than 130 times (minimum to win Rookie of the Year).

The same pattern arises when we look at the top performers: batting averages go down for most of the top performers.

| nameFirst | nameLast | 2013  | 2014  |
|-----------|----------|-------|-------|
| Miguel    | Cabrera  | 0.348 | 0.313 |
| Hanley    | Ramirez  | 0.345 | 0.283 |
| Michael   | Cuddyer  | 0.331 | 0.332 |

| nameFirst | nameLast | 2013  | 2014  |
|-----------|----------|-------|-------|
| Scooter   | Gennett  | 0.324 | 0.289 |
| Joe       | Mauer    | 0.324 | 0.277 |

But these are not rookies! Also, look at what happens to the worst performers of 2013:

| nameFirst | nameLast | 2013  | 2014  |
|-----------|----------|-------|-------|
| Danny     | Espinosa | 0.158 | 0.219 |
| Dan       | Uggla    | 0.179 | 0.149 |
| Jeff      | Mathis   | 0.181 | 0.200 |
| B. J.     | Upton    | 0.184 | 0.208 |
| Adam      | Rosales  | 0.190 | 0.262 |

Their batting averages mostly go up! Is this some sort of reverse sophomore slump? It is not. There is no such thing as a sophomore slump. This is all explained with a simple statistical fact: the correlation for performance in two separate years is high, but not perfect:

The correlation is 0.460254 and the data look very much like a bivariate normal distribution, which means we predict a 2014 batting average  $\langle Y \rangle$  for any given player that had a 2013 batting average  $\langle X \rangle$  with:

$$\left[ \frac{Y - .255}{.032} = 0.46 \left( \frac{X - .261}{.023} \right) \right]$$

Because the correlation is not perfect, regression tells us that, on average, expect high performers from 2013 to do a bit worse in 2014. It's not a jinx; it's just due to chance. The ROY are selected from the top values of  $\langle X \rangle$ , so it is expected that  $\langle Y \rangle$  will regress to the mean.

## 14.16 Exercises

- Load the `GaltonFamilies` data from the `HistData`. The children in each family are listed by gender and then by height. Create a dataset called `galton_heights` by picking a male and female at random.
- Make a scatterplot for heights between mothers and daughters, mothers and sons, fathers and daughters, and fathers and sons.
- Compute the correlation in heights between mothers and daughters, mothers and sons, fathers and daughters, and fathers and sons.

- 
- [https://en.wikipedia.org/wiki/Francis\\_Galton](https://en.wikipedia.org/wiki/Francis_Galton)
  - <https://web.archive.org/web/20160815063904/http://www.foxsports.com/mlb/story/kris-bryant-carlos-correa-rookies-of-year-award-matt-duffy-francisco-lindor-kang-sano-120715>

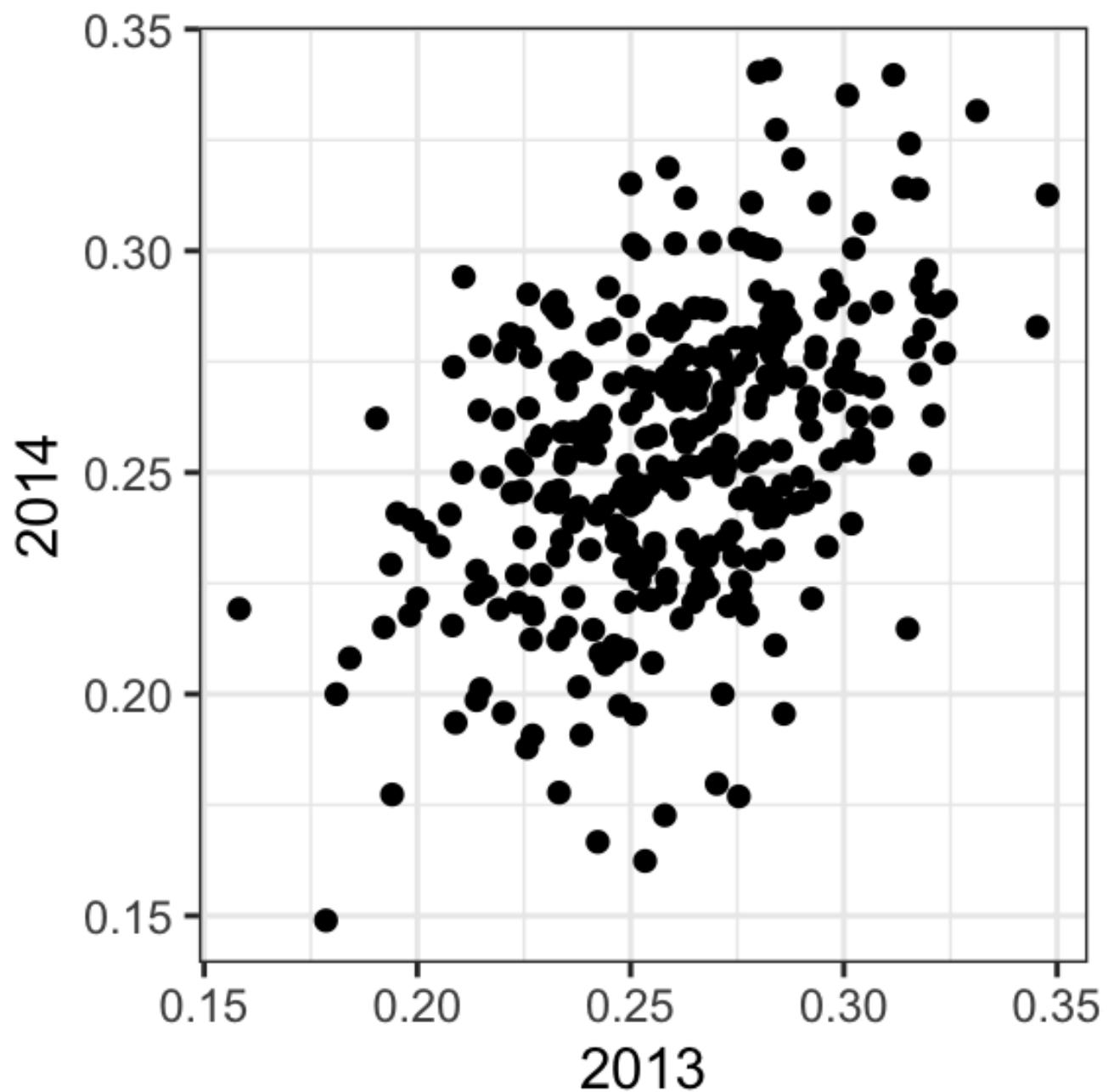


Figure 15:

# Introduction to Data Science - 15 Multivariate Regression

Rafael A. Irizarry

Since Galton's original development, regression has become one of the most widely used tools in data analysis. One reason is the fact that an adaptation of the original regression approach, based on linear models, permits us to find relationships between two variables taking into account the effects of other variables that affect both. This has been particularly popular in fields where randomized experiments are hard to run, such as economics and epidemiology.

When we are unable to randomly assign each individual to a treatment or control group, confounding becomes particularly prevalent. For instance, consider estimating the effect of eating fast foods on life expectancy using data collected from a random sample of people in a jurisdiction. Fast food consumers are more likely to be smokers, drinkers, and have lower incomes. Consequently, a naive regression model may lead to an overestimate of the negative health effects of fast food. So, how do we account for confounding in practice? In this chapter, we learn how *multivariate regression* can help with such situations and can be used to describe how one or more variables affect an outcome variable. We illustrate with a real-world example in which data was used to help pick underappreciated players to improve a resource limited sports team.

## 15.1 Case study: Moneyball

*Moneyball: The Art of Winning an Unfair Game* by Michael Lewis focuses on the Oakland Athletics (A's) baseball team and its general manager, Billy Beane, the person tasked with building the team.

Traditionally, baseball teams use *scouts* to help them decide what players to hire. These scouts evaluate players by observing them perform, tending to favor athletic players with observable physical abilities. For this reason, scouts generally agree on who the best players are and, as a result, these players are often in high demand. This in turn drives up their salaries.

From 1989 to 1991, the A's had one of the highest payrolls in baseball. They were able to buy the best players and, during that time, were one of the best teams. However, in 1995, the A's team owner changed and the new management cut the budget drastically, leaving then general manager, Sandy Alderson, with one of the lowest payrolls in baseball. He could no longer afford the most sought-after players. As a result, Alderson began using a statistical approach to find inefficiencies in the market. Alderson was a mentor to Billy Beane, who succeeded him in 1998 and fully embraced data science, as opposed to relying exclusively on scouts, as a method for finding low-cost players that data predicted would help the team win. Today, this strategy has been adapted by most baseball teams. As we will see, regression plays a significant role in this approach.

As motivation for this part of the book, let's imagine it is 2002, and attempt to build a baseball team with a limited budget, just like the A's had to do. To appreciate what you are up against, note that in 2002 the Yankees' payroll of \$125,928,583 more than tripled the Oakland A's payroll of \$39,679,746:

Statistics have been used in baseball since its beginnings. The dataset we will be using, included in the **Lahman** library, goes back to the 19th century. For example, a summary statistics we will describe soon, the *batting average*, has been used for decades to summarize a batter's success. Other statistics<sup>1</sup>, such as home runs (HR), runs batted in (RBI), and stolen bases (SB), are reported for each player in the game summaries included in the sports section of newspapers, with players rewarded for high numbers. Although summary statistics such as these were widely used in baseball, data analysis per se was not. These statistics were arbitrarily chosen without much thought as to whether they actually predicted anything or were related to helping a team win.

This changed with Bill James<sup>2</sup>. In the late 1970s, this aspiring writer and baseball fan started publishing articles describing more in-depth analysis of baseball data. He named the approach of using data to determine which

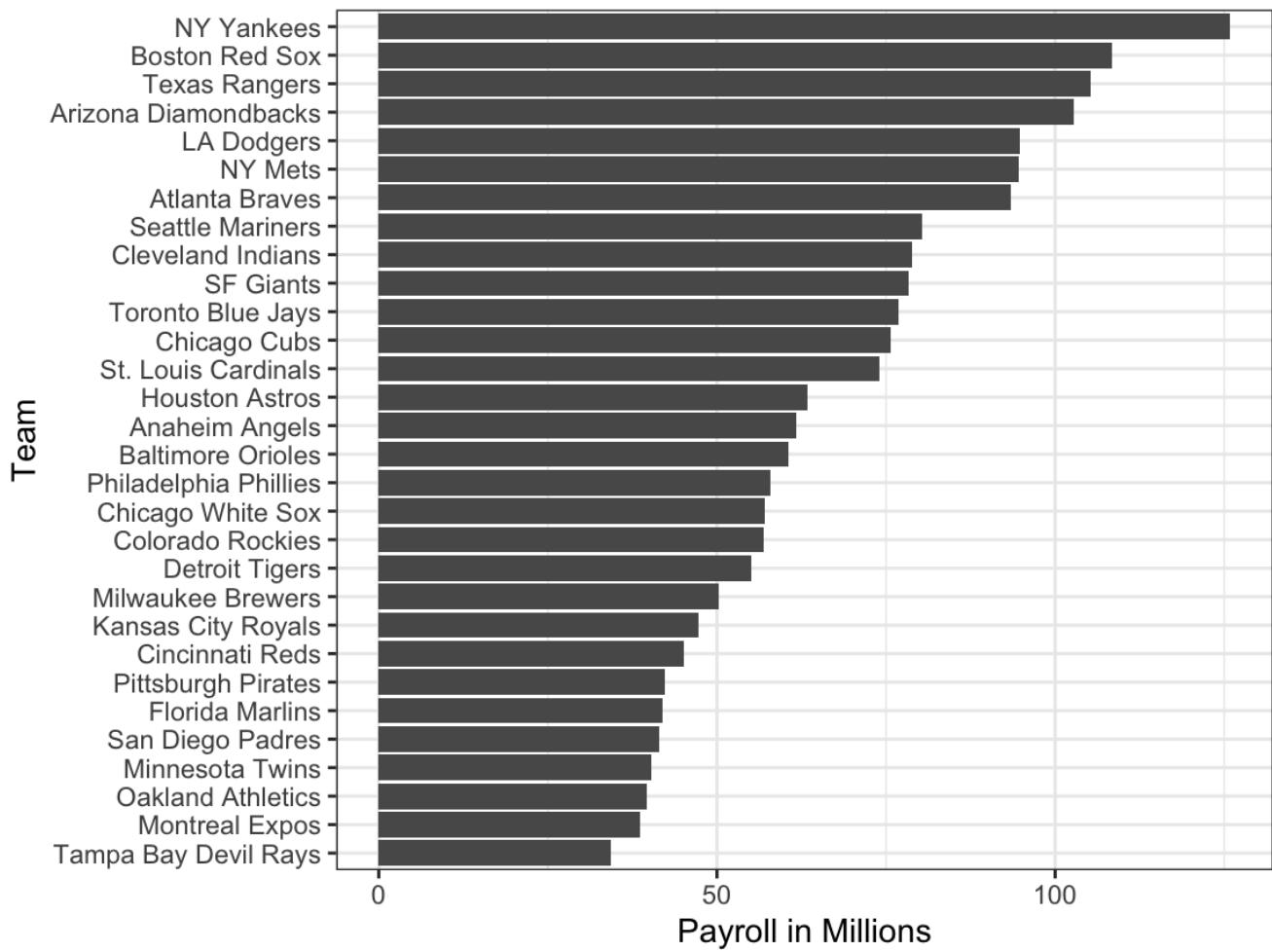


Figure 1:

outcomes best predicted if a team would win *sabermetrics*<sup>3</sup>. Yet until Billy Beane made sabermetrics the center of his baseball operation, Bill James' work was mostly ignored by the baseball world. Currently, sabermetrics popularity is no longer limited to just baseball, with other sports also adopting this approach.

To simplify the exercise, we will focus on scoring runs and ignore the two other important aspects of the game: pitching and fielding. We will see how regression analysis can help develop strategies to build a competitive baseball team with a constrained budget. The approach can be divided into two separate data analyses. In the first, we determine which recorded player-specific statistics predict runs. In the second, we examine if players were undervalued based on the predictions from our first analysis.

### 15.1.1 Baseball basics

To understand how regression helps us find undervalued players, we don't need to delve into all the details of the game of baseball, which has over 100 rules. Here, we distill the sport to the basic knowledge necessary for effectively addressing the data science problem.

The goal of a baseball game is to score more runs (points) than the other team. Each team has 9 batters that have an opportunity to hit a ball with a bat in a predetermined order. After the 9th batter has had their turn, the first batter bats again, then the second, and so on. Each time a batter has an opportunity to bat, we call it a plate appearance (PA). At each PA, the other team's *pitcher* throws the ball and the batter tries to hit it. The PA ends with an binary outcome: the batter either makes an *out* (failure) and returns to the bench, or the batter doesn't (success) and can run around the bases, potentially scoring a run (reaching all 4 bases). Each team gets nine tries, referred to as *innings*, to score runs, and each inning ends after three outs (three failures).

Here is a video showing a success: <https://www.youtube.com/watch?v=HL-XjMCPfio>. And here is one showing a failure: <https://www.youtube.com/watch?v=NeloljCx-1g>. In these videos, we see how luck is involved in the process. When at bat, the batter wants to hit the ball hard. If the batter hits it hard enough, it is a HR, the best possible outcome as the batter gets at least one automatic run. But sometimes, due to chance, the batter hits the ball very hard and a defender catches it, resulting in an out. In contrast, sometimes the batter hits the ball softly, but it lands in just the right place. The fact that there is chance involved hints at why probability models will be involved.

Now, there are several ways to succeed. Understanding this distinction will be important for our analysis. When the batter hits the ball, the batter wants to pass as many *bases* as possible. There are four bases, with the fourth one called *home plate*. Home plate is where batters start by trying to hit, so the bases form a cycle.

(Courtesy of Cburnett<sup>4</sup>. CC BY-SA 3.0 license<sup>5</sup>.)

A batter who *goes around the bases* and arrives home, scores a run.

We are simplifying a bit, but there are five ways a batter can succeed, meaning not make an out:

- Bases on balls (BB): The pitcher fails to throw the ball through a predefined area considered to be hittable (the strike zone), so the batter is permitted to go to first base.
- Single: The batter hits the ball and gets to first base.
- Double (2B): The batter hits the ball and gets to second base.
- Triple (3B): The batter hits the ball and gets to third base.
- Home Run (HR): The batter hits the ball and goes all the way home and scores a run.

Here is an example of a HR: <https://www.youtube.com/watch?v=xYxSZJ9GZ-w>. If a batter reaches a base, the batter still has a chance of reaching home and scoring a run if the next batter succeeds with a hit. While the batter is *on base*, the batter can also try to steal a base (SB). If a batter runs fast enough, the batter can try to advance from one base to the next without the other team tagging the runner. Here is an example of a stolen base: <https://www.youtube.com/watch?v=JSE5kfxkzfk>.

All these events are tracked throughout the season and are available to us through the **Lahman** package. Now, we can begin discussing how data analysis can help us determine how to use these statistics to evaluate players.

### 15.1.2 No awards for BB

Historically, the *batting average* has been considered the most important offensive statistic. To define this average, we define a *hit* (H) and an *at bat* (AB). Singles, doubles, triples, and home runs are hits. The fifth way to be successful, BB, is not a hit. An AB is the number of times in which you either get a hit or make an out; BBs are excluded.

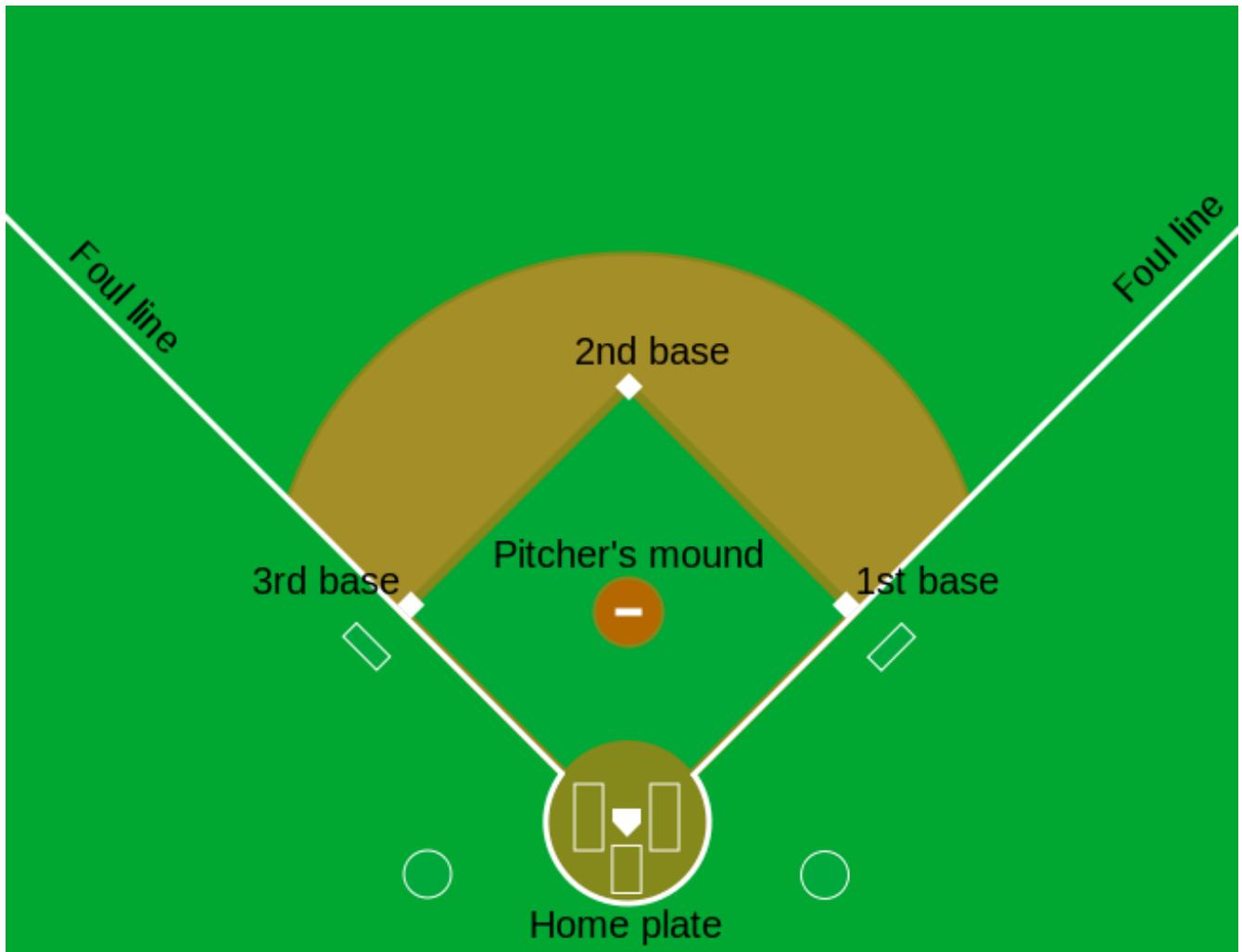


Figure 2:

The batting average is simply H/AB and is considered the main measure of a success rate. Today, this success rate ranges from 20% to 38%. We refer to the batting average in thousands so, for example, if your success rate is 28%, we call it *batting 280*.



Figure 3:

(Picture courtesy of Keith Allison<sup>6</sup>. CC BY-SA 2.0 license<sup>7</sup>.)

One of Bill James' first important insights is that the batting average ignores BB, but a BB is a success. Instead of batting average, James proposed the use of the *on-base percentage* (OBP), which he defined as  $(H+BB)/(AB+BB)$ , or simply the proportion of plate appearances that don't result in an out, a very intuitive measure. He noted that a player that accumulates many more BB than the average player might go unrecognized if the batter does not excel in batting average. But is this player not helping produce runs? No award is given to the player with the most BB. However, bad habits are hard to break and baseball did not immediately adopt OBP as an important statistic. In contrast, total stolen bases were considered important and an award<sup>8</sup> given to the player with the most. But players with high totals of SB also made more outs as they did not always succeed. Does a player with high SB total help produce runs? Can we use data science to determine if it's better to pay for players with high BB or SB?

### 15.1.3 Base on balls or stolen bases?

One of the challenges in this analysis is that it is not obvious how to determine if a player produces runs because so much depends on his teammates. Although we keep track of the number of runs scored by a player, remember that if player X bats right before someone who hits many HRs, batter X will score many runs. Note these runs don't necessarily happen if we hire player X, but not his HR hitting teammate.

However, we can examine team-level statistics. How do teams with many SB compare to teams with few? How about BB? We have data! Let's examine some. We start by creating a data frame with statistics from 1962 (the first year all teams played 162 games, like today, instead of 154) to 2001 (the year before the year for which we will construct a team). We convert the data to a *per game* rate, because a small proportion of seasons had less games than usual due to strikes, and some teams played extra games due to tie breakers.

```
library(tidyverse)
#> Attaching core tidyverse packages tidyverse 2.0.0
#> dplyr 1.1.4 readr 2.1.5
#> forcats 1.0.0 stringr 1.5.1
#> lubridate 1.9.3 tibble 3.2.1
#> purrr 1.0.2 tidyverse_conflicts()
#> Conflicts tidyverse_conflicts()
#> dplyr::filter() masks stats::filter()
```

```

#> dplyr::lag() masks stats::lag()
#> Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
library(Lahman)
dat <- Teams |> filter(yearID %in% 1962:2002) |>
 mutate(team = teamID, year = yearID, r = R/G,
 singles = (H - X2B - X3B - HR)/G,
 doubles = X2B/G, triples = X3B/G, hr = HR/G,
 sb = SB/G, bb = BB/G) |>
 select(team, year, r, singles, doubles, triples, hr, sb, bb)

```

Now let's start with a obvious question: do teams that hit more home runs score more runs? When exploring the relationship between two variables, The visualization of choice is a scatterplot:

```

p <- dat |> ggplot(aes(hr, r)) + geom_point(alpha = 0.5)
p

```

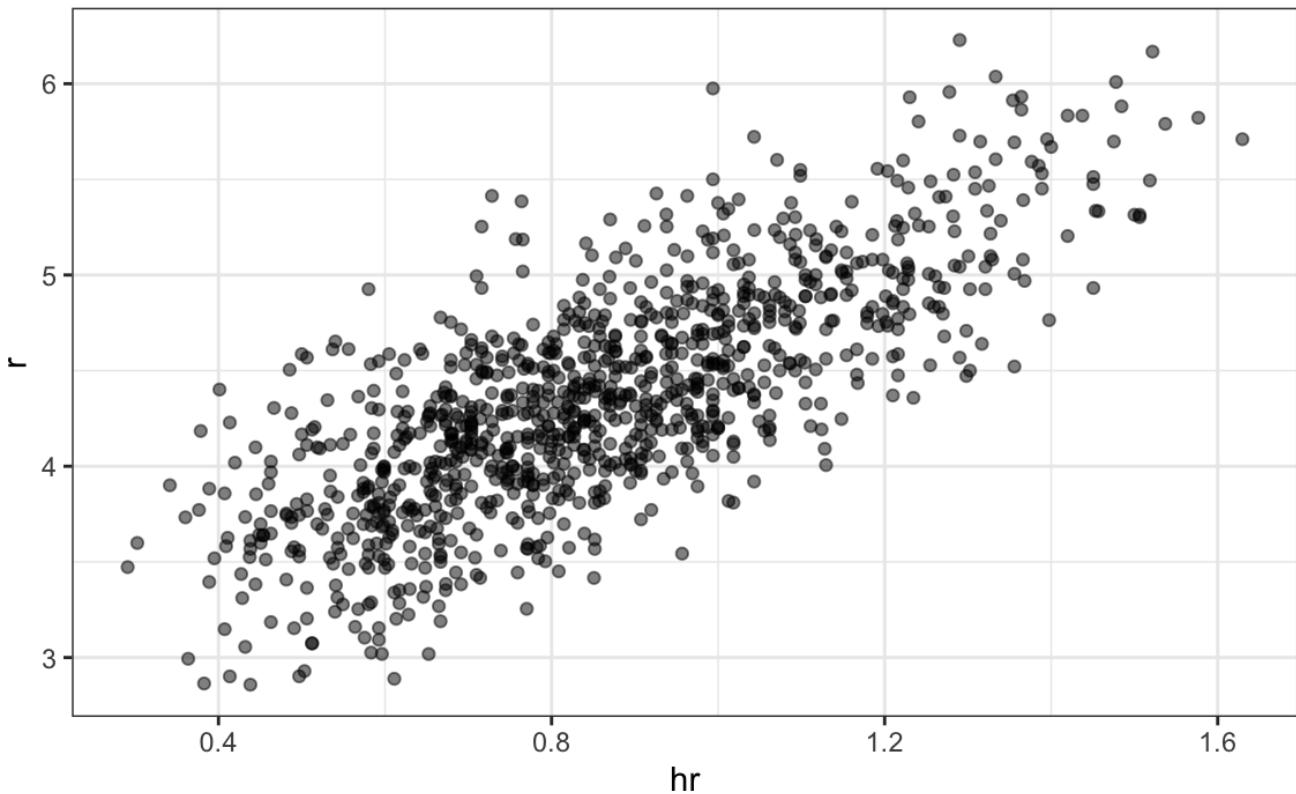


Figure 4:

We defined p because we will add it to this plot latter. The plot shows a strong association: teams with more HRs tend to score more runs. Now let's examine the relationship between stolen bases and runs:

```
dat |> ggplot(aes(sb, r)) + geom_point(alpha = 0.5)
```

Here the relationship is not as clear. Finally, let's examine the relationship between BB and runs:

```
dat |> ggplot(aes(bb, r)) + geom_point(alpha = 0.5)
```

Here again we see a clear association. But does this mean that increasing a team's BBs **causes** an increase in runs? One of the most important lessons you learn in this book is that **association is not causation**. In fact, it looks like BBs and HRs are also associated:

```
dat |> ggplot(aes(hr, bb)) + geom_point(alpha = 0.5)
```

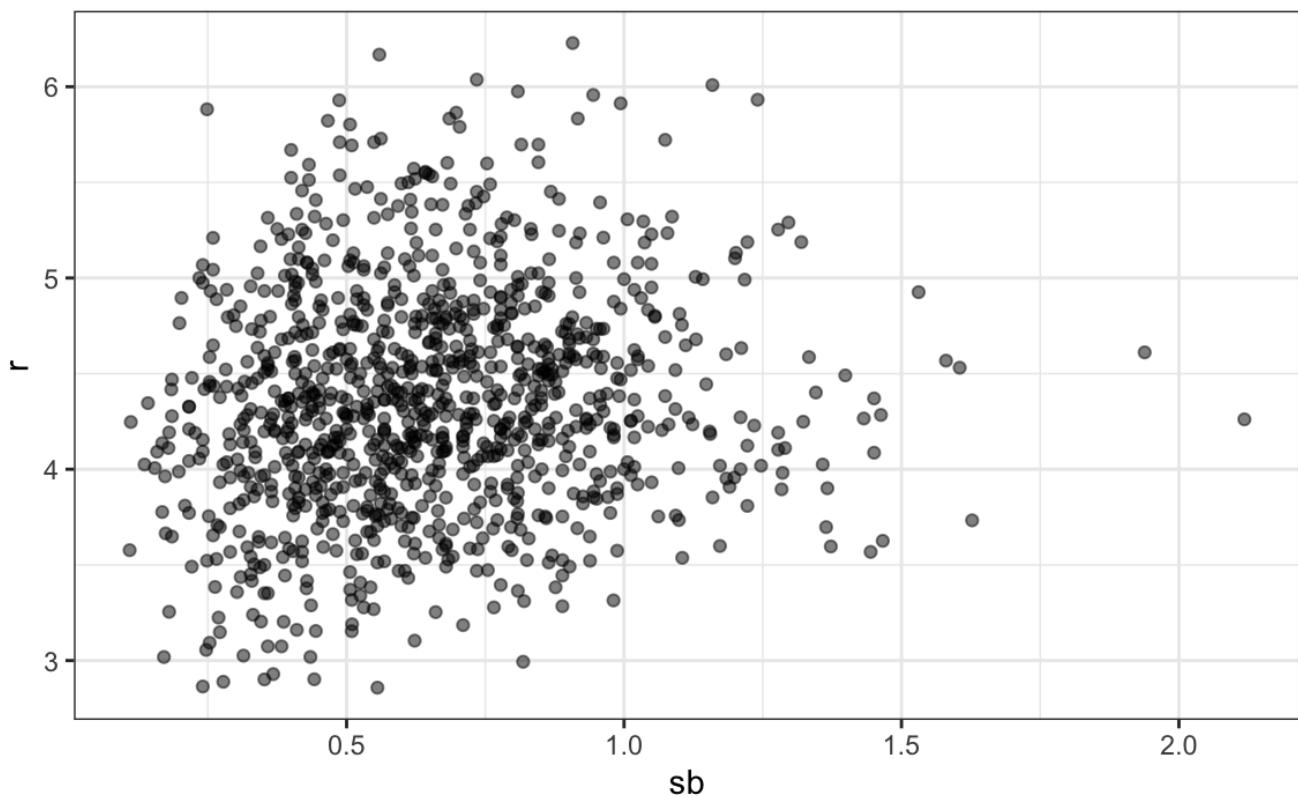


Figure 5:

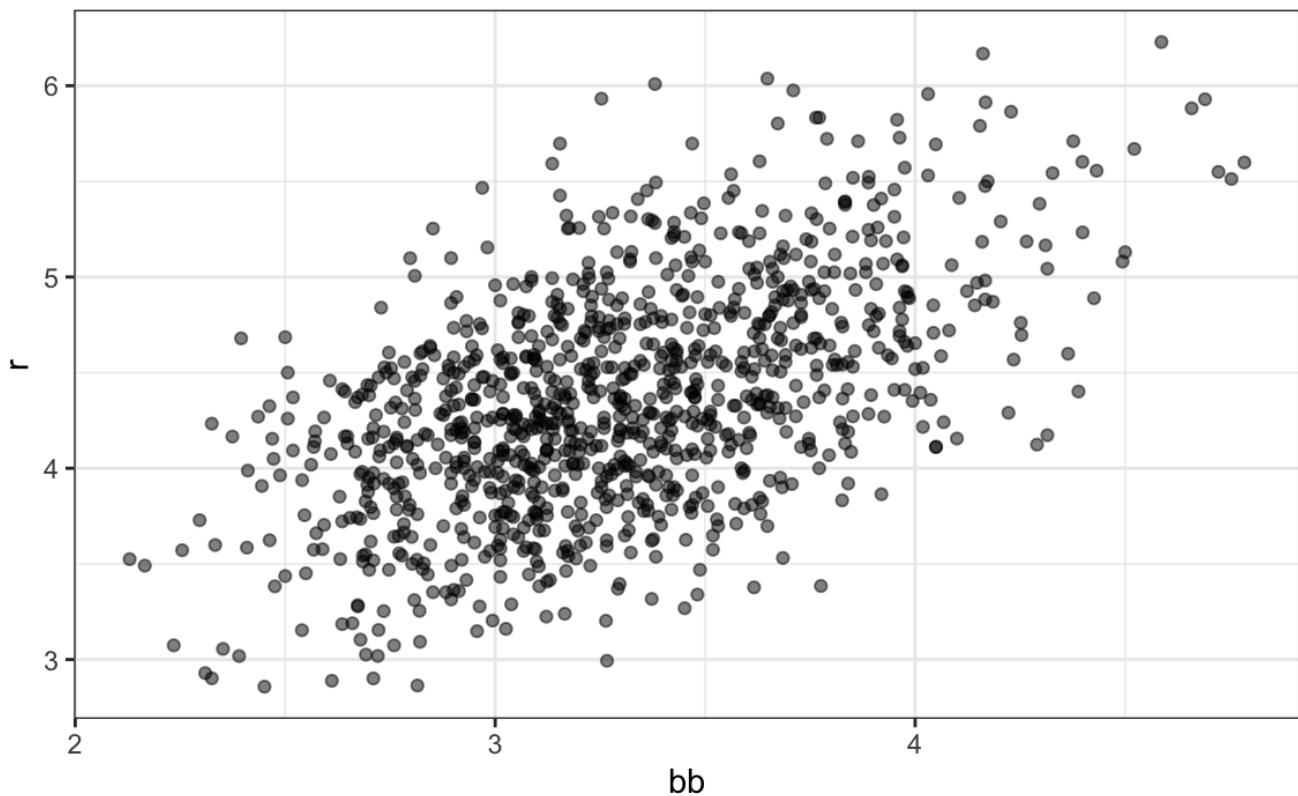


Figure 6:

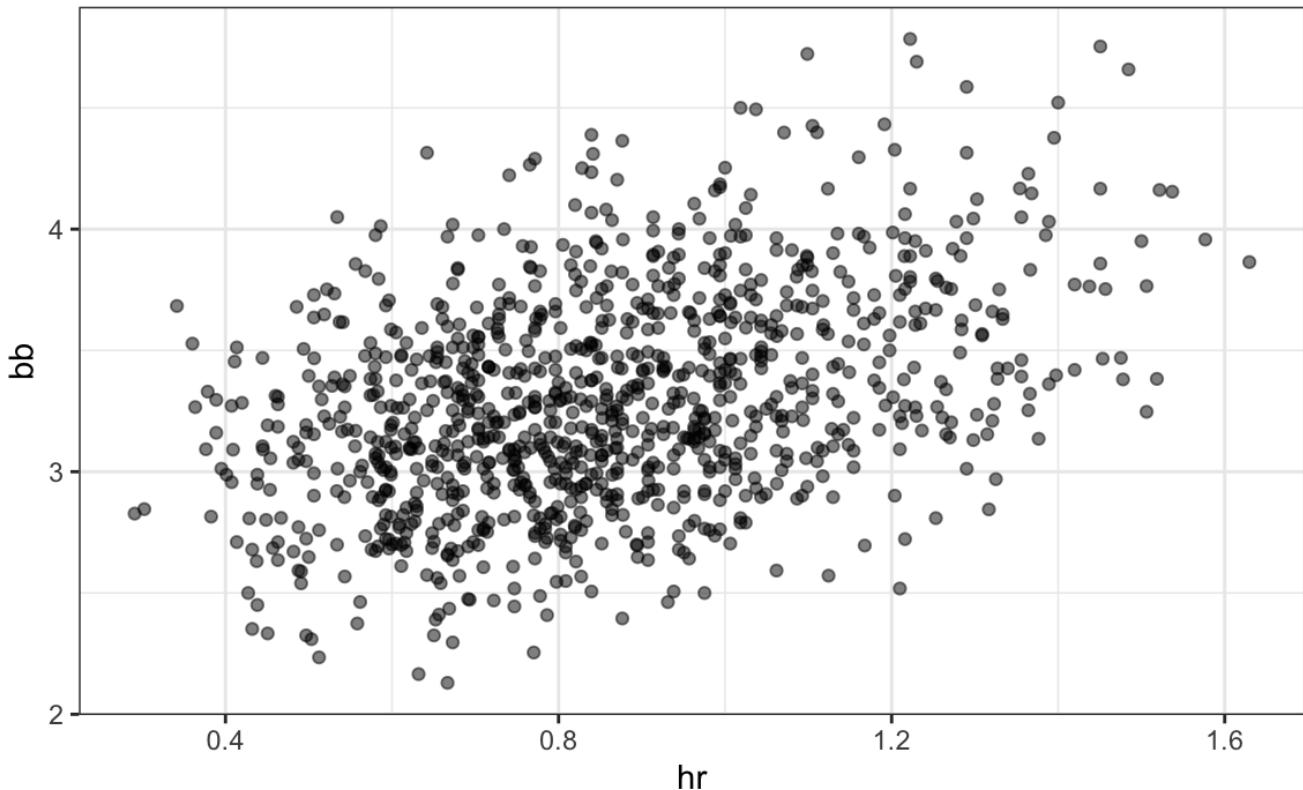


Figure 7:

We know that HRs cause runs because when a player hits a HR, they are guaranteed at least one run. Could it be that HRs also cause BB and this makes it appear as if BB cause runs? When this happens, we say there is *confounding*, an important concept we will learn more about throughout this section.

Linear regression will help us parse out the information and quantify the associations. This, in turn, will aid us in determining what players to recruit. Specifically, we will try to predict things like how many more runs will a team score if we increase the number of BBs, but keep the HRs fixed? Regression will help us answer questions like this one.

#### 15.1.4 Regression applied to baseball statistics

Can we use regression with these data? First, notice that the HR and Run data, shown above, appear to be bivariate normal. Specifically, the qqplots confirm that the normal approximation for each HR strata is useful here:

```
dat |> mutate(z_hr = round(scale(hr))) |>
 filter(z_hr %in% -2:3) |>
 ggplot() +
 stat_qq(aes(sample = r)) +
 facet_wrap(~z_hr)
```

Now we are ready to use linear regression to predict the number of runs a team will score, if we know how many home runs the team hits using regression:

```
hr_fit <- lm(r ~ hr, data = dat)$coef
p + geom_abline(intercept = hr_fit[[1]], slope = hr_fit[[2]])
```

Note that we can obtain the same plot more quickly by using the `ggplot2` function `geom_smooth`, which computes and adds a regression line to plot along with confidence intervals. We use the argument `method = "lm"`, which stands for *linear model*, the title of an upcoming section. We simplify the code above like this:

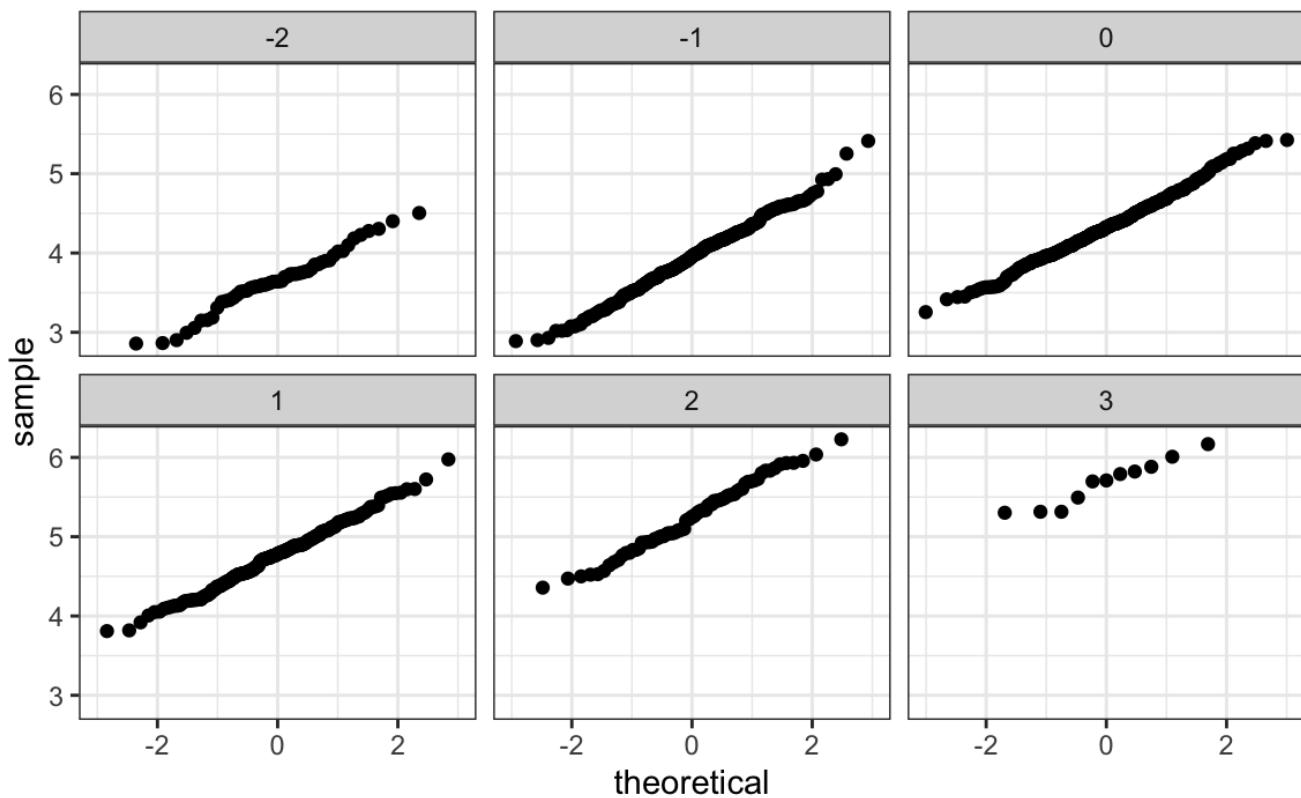


Figure 8:

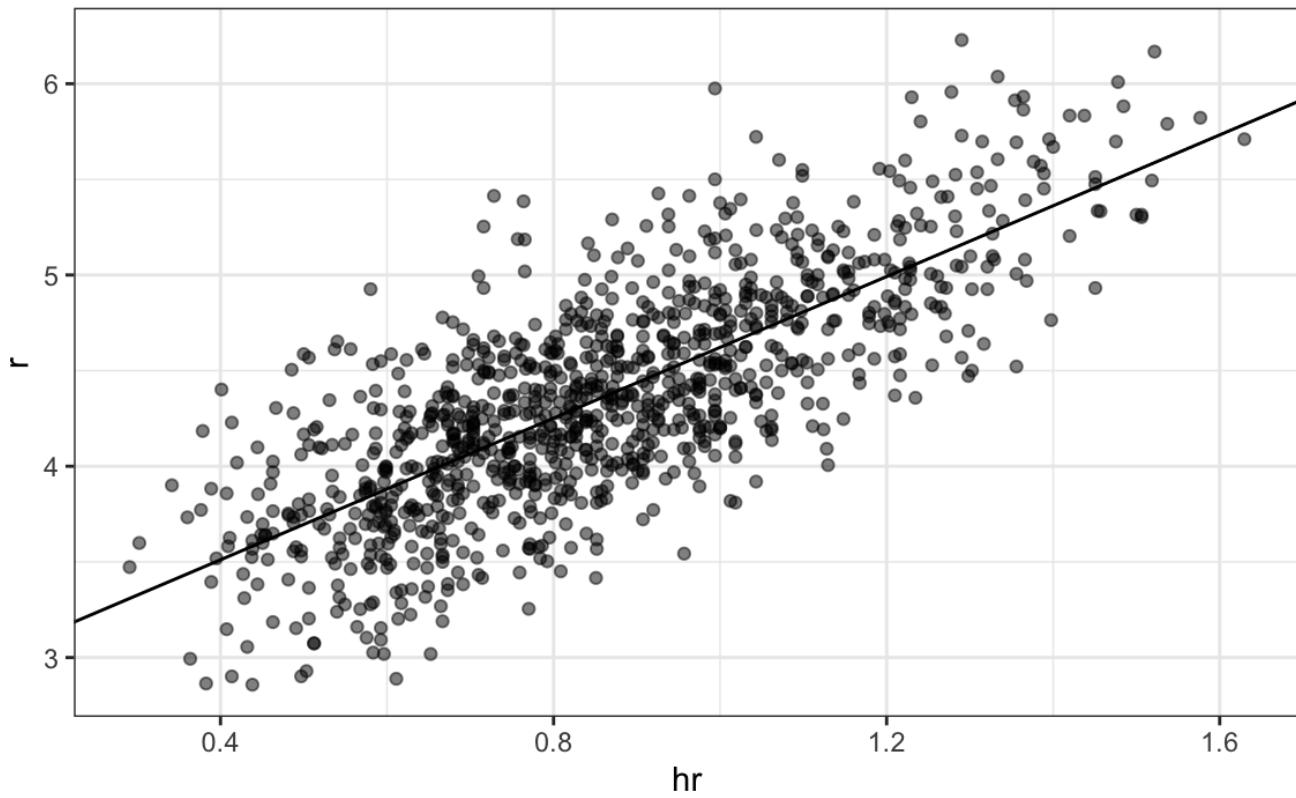


Figure 9:

```
p + geom_smooth(method = "lm")
```

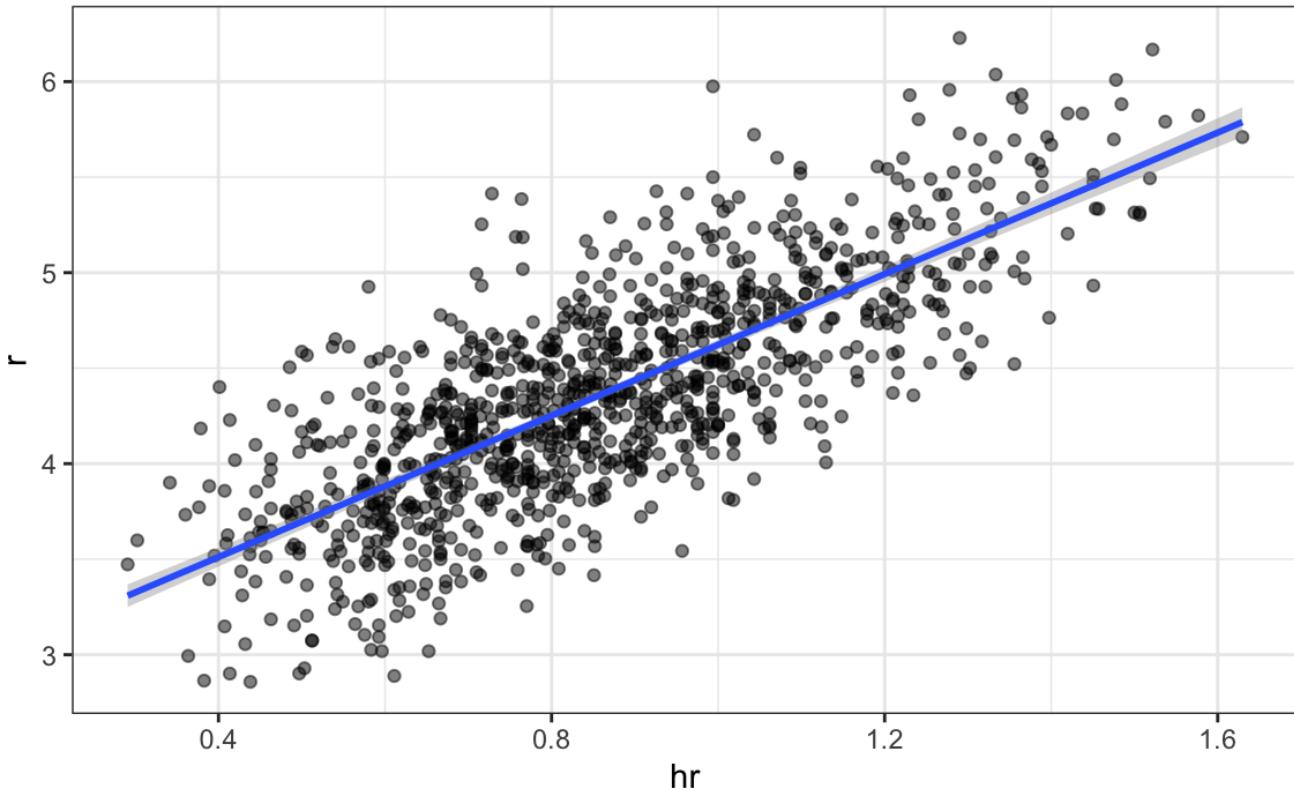


Figure 10:

In the example above, the slope is 1.8517449. This tells us that teams that hit 1 more HR per game than the average team, score 1.8517449 more runs per game than the average team. Given that the most common final score is a difference of one run, this can certainly lead to a large increase in wins. Not surprisingly, HR hitters are very expensive. Because we are working on a budget, we will need to find some other way to increase wins. In the next chapter, we introduce *linear models*, which provide an framework for performing this analysis. In @ref{@moneyball}, we apply what have learned to build a baseball team.

## 15.2 The broom package

The **broom** package facilitates the use of R function, such as `lm`, within the tidyverse. Recall the that `lm` does not take a data frame as a first argument and does not return a data frame, which makes using `lm` in conjunction with the **tidyverse** difficult. It has three main functions, all of which extract information from the object returned by `lm` and returns it in a **tidyverse** friendly data frame. These functions are: `tidy`, `glance`, and `augment`. The `tidy` function returns estimates and related information as a data frame:

```
library(broom)
fit <- lm(r ~ bb, data = dat)
tidy(fit)
#> # A tibble: 2 × 5
#> term estimate std.error statistic p.value
#> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 (Intercept) 1.93 0.116 16.7 1.91e-55
#> 2 bb 0.739 0.0348 21.2 1.90e-83
```

We can add other important summaries, such as confidence intervals:

```
tidy(fit, conf.int = TRUE)
#> # A tibble: 2 × 7
```

```
#> term estimate std.error statistic p.value conf.low conf.high
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 (Intercept) 1.93 0.116 16.7 1.91e-55 1.70 2.15
#> 2 bb 0.739 0.0348 21.2 1.90e-83 0.671 0.807
```

Given that the outcome is a data frame, we can immediately use it with `summarize` to string together the commands that produce the table we are after. Because a data frame is returned, we can filter and select the rows and columns we want, as we will see in the next section.

Now we return to discussing our original task of determining if slopes changed. The plot we just made, using `summarize` and `tidy`, shows that the confidence intervals overlap, which provides a nice visual confirmation that our assumption that the slope does not change is safe.

The other functions provided by `broom`, `glance` and `augment`, relate to model-specific and observation-specific outcomes, respectively.

## 15.3 Confounding

Previously, we noted a strong relationship between Runs and BB. If we find the regression line for predicting runs from bases on balls, we get slope of:

```
bb_slope <- lm(r ~ bb, data = dat)$coef[2]
bb_slope
#> bb
#> 0.739
```

Does this mean that if we go and hire low salary players with many BB, and who therefore increase the number of walks per game by 2, our team will score 1.5 more runs per game?

We are again reminded that association is not causation. The data does provide strong evidence that a team with two more BB per game than the average team, scores 1.5 runs per game. But this does not mean that BB are the cause.

Note that, if we compute the regression line slope for singles, we get:

```
lm(r ~ singles, data = dat)$coef[2]
#> singles
#> 0.432
```

which is a lower value than what we obtain for BB. Remember that a single gets you to first base just like a BB. Baseball fans will point out that with a single, runners on base have a better chance of scoring than with a BB. So how can BB be more predictive of runs? The reason is because of confounding. Here we show the correlation between HR, BB, and singles:

```
dat |> summarize(cor(bb, hr), cor(singles, hr), cor(bb, singles))
#> cor(bb, hr) cor(singles, hr) cor(bb, singles)
#> 1 0.406 -0.186 -0.0513
```

It appears that pitchers, afraid of HRs, will sometimes avoid throwing strikes to HR hitters. As a result, HR hitters tend to have more BBs, and a team with many HRs will also have more BBs. Although it may appear that BBs cause runs, it is actually the HRs that cause most of these runs. We say that BBs are *confounded* with HRs. Nonetheless, could it be that BBs still help? To find out, we somehow have to adjust for the HR effect. Regression can help with this as well.

### 15.3.1 Understanding confounding through stratification

A first approach is to keep HRs fixed at a certain value and then examine the relationship between BB and runs. As we did when we stratified fathers by rounding to the closest inch, here we can stratify HR per game to the closest ten. We filter out the strata with few points to avoid highly variable estimates and then make a scatterplot for each strata:

```
dat |> mutate(hr_strata = round(hr, 1)) |>
 filter(hr_strata >= 0.4 & hr_strata <= 1.2) |>
```

```
ggplot(aes(bb, r)) +
 geom_point(alpha = 0.5) +
 geom_smooth(method = "lm") +
 facet_wrap(~hr_strata)
```

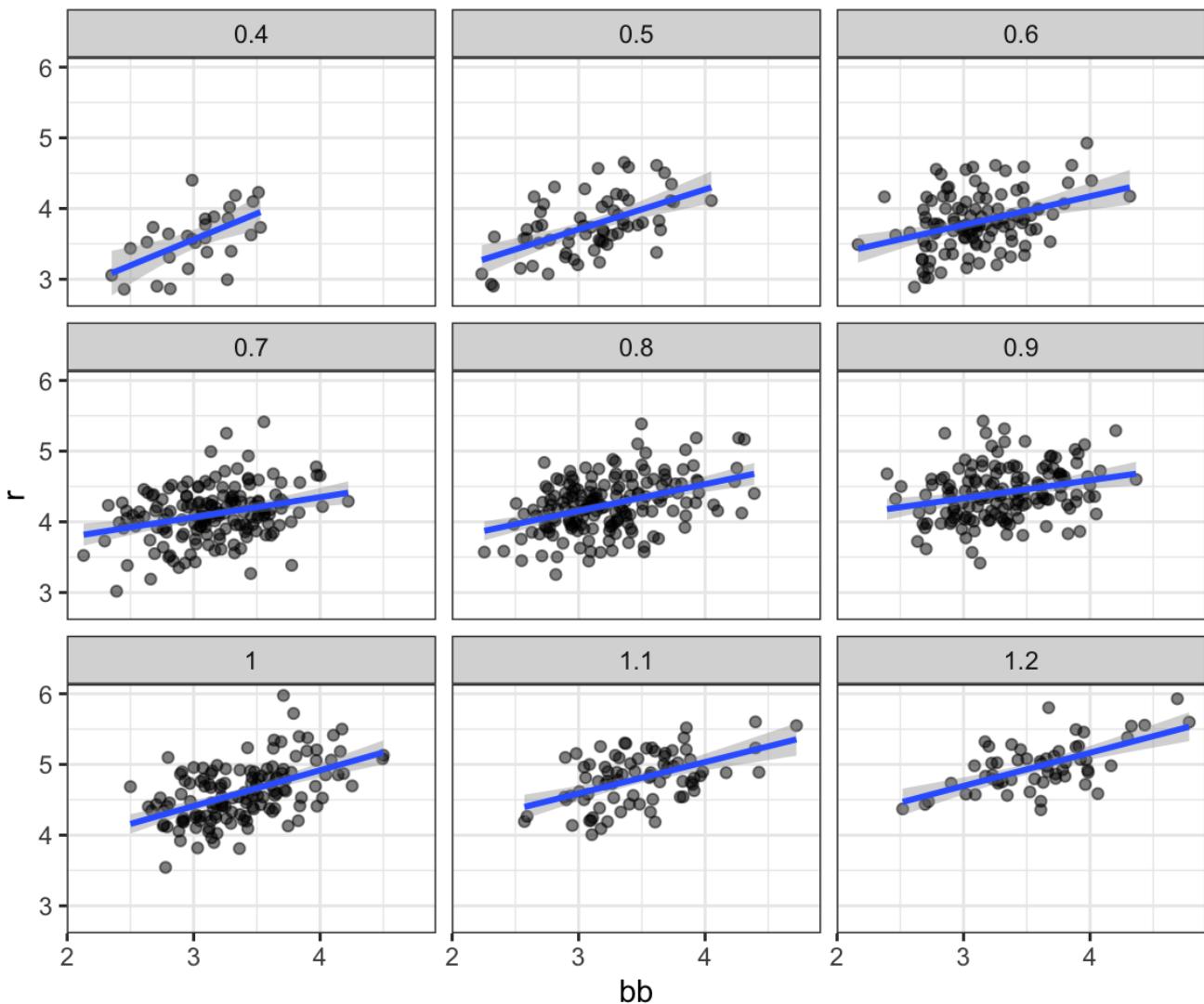


Figure 11:

Remember that the regression slope for predicting runs with BB was 0.7. Once we stratify by HR, these slopes are substantially reduced:

```
dat |> mutate(hr_strata = round(hr, 1)) |>
 filter(hr_strata >= 0.5 & hr_strata <= 1.2) |>
 group_by(hr_strata) |>
 reframe(tidy(lm(r ~ bb))) |>
 filter(term == "bb")
#> # A tibble: 8 × 6
#> hr_strata term estimate std.error statistic p.value
#> <dbl> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 0.5 bb 0.566 0.110 5.14 0.00000302
#> 2 0.6 bb 0.405 0.0984 4.12 0.0000746
#> 3 0.7 bb 0.284 0.0717 3.96 0.000113
#> 4 0.8 bb 0.378 0.0638 5.92 0.0000000175
```

```
#> 5 0.9 bb 0.254 0.0762 3.33 0.00108
#> # 3 more rows
```

Note we use `reframe` instead of `summarize` because `tidy` returns a data frame with two rows.

The slopes are reduced, but they are not 0, which indicates that BBs are helpful for producing runs, just not as much as previously thought. In fact, the values above are closer to the slope we obtained from singles, 0.4, which is more consistent with our intuition. Since both singles and BB get us to first base, they should have about the same predictive power.

Although our understanding of the application tells us that HR cause BB, but not the other way around, we can still check if stratifying by BB makes the effect of BB go down. To do this, we use the same code except that we swap HR and BBs. In this case, the slopes do not change much from the original:

```
dat |> mutate(bb_strata = round(bb, 1)) |>
 filter(bb_strata >= 3 & bb_strata <= 4) |>
 group_by(bb_strata) |>
 reframe(tidy(lm(r ~ hr))) |>
 filter(term == "hr")
#> # A tibble: 11 × 6
#> bb_strata term estimate std.error statistic p.value
#> <dbl> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 3 hr 1.51 0.182 8.31 1.47e-12
#> 2 3.1 hr 1.49 0.168 8.87 3.10e-14
#> 3 3.2 hr 1.61 0.150 10.8 6.96e-18
#> 4 3.3 hr 1.57 0.167 9.39 5.73e-15
#> 5 3.4 hr 1.55 0.153 10.1 3.77e-16
#> # 6 more rows
```

They are reduced slightly from 1.8517449, which is consistent with the fact that BB do in fact cause some runs.

Regardless, it seems that if we stratify by HR, we have bivariate distributions for runs versus BB. Similarly, if we stratify by BB, we have approximate bivariate normal distributions for HR versus runs.

## 15.4 Multivariable regression

It is somewhat complex to be computing regression lines for each strata. We are essentially fitting models like this:

$$\mathbb{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1(x_2)x_1 + \beta_2(x_1)x_2$$

with the slopes for  $\beta_1$  changing for different values of  $\beta_2$  and vice versa. But is there an easier approach?

If we take random variability into account, the slopes in the strata don't appear to change much. If these slopes are in fact the same, this implies that  $\beta_1$  and  $\beta_2$  are constants. This, in turn, implies that the expectation of runs conditioned on HR and BB can be written as follows:

$$\mathbb{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

This model suggests that, if the number of HR is fixed at  $x_2$ , we observe a linear relationship between runs and BB with an intercept of  $\beta_0 + \beta_2 x_2$ . Our exploratory data analysis suggested that this is the case. The model also suggests that as the number of HR grows, the intercept growth is linear as well and determined by  $\beta_1$ . In this analysis, referred to as *multivariable regression*, you will often hear people say that the BB slope  $\beta_1$  is *adjusted* for the HR effect.

Because the data is approximately normal and conditional distributions were also normal, we are justified in using a linear model:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \varepsilon_i$$

with  $Y_i$  runs per game for team  $i$ ,  $x_{i,1}$  walks per game, and  $x_{i,2}$ . To use `lm` here, we need to let the function know we have two predictor variables. We therefore use the `+` symbol as follows:

```

tidy(lm(r ~ bb + hr, data = dat), conf.int = TRUE)
#> # A tibble: 3 × 7
#> term estimate std.error statistic p.value conf.low conf.high
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 (Intercept) 1.74 0.0820 21.2 3.38e- 83 1.58 1.90
#> 2 bb 0.387 0.0269 14.4 8.41e- 43 0.334 0.440
#> 3 hr 1.57 0.0488 32.1 1.39e-157 1.47 1.66

```

When we fit the model with only one variable, the estimated slopes were 0.7388725 and 1.8517449 for BB and HR, respectively. Note that when fitting the multivariable model both go down, with the BB effect decreasing much more.

You are ready to do exercises 1-12, if you want to practice before continuing.

### 15.4.1 Building a baseball team

Now we want to construct a metric to pick players, and we need to consider singles, doubles, and triples as well. Can we build a model that predicts runs based on all these outcomes? We take somewhat of a “leap of faith” and assume that these five variables are jointly normal. This means that, if we pick any one of them and hold the other four fixed, the relationship with the outcome is linear and the slope does not depend on the four values held constant. If this is true, then a linear model for our data is:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \beta_5 x_{i,5} + \varepsilon_i$$

with  $(x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}, x_{i,5})$  representing BB, singles, doubles, triples, and HR respectively.

Using `lm`, we can quickly find the LSE for the parameters using:

```
fit <- dat |> filter(year <= 2001) |> lm(r ~ bb + singles + doubles + triples + hr, data = _)
```

Be aware that we fit the model to data up until 2001, the year before we will construct our team. We can see the coefficients using `tidy`:

```

tidy(fit, conf.int = TRUE) |> filter(term != "(Intercept)")
#> # A tibble: 5 × 7
#> term estimate std.error statistic p.value conf.low conf.high
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 bb 0.370 0.0119 31.2 1.00e-149 0.347 0.393
#> 2 singles 0.517 0.0128 40.5 5.29e-213 0.492 0.543
#> 3 doubles 0.775 0.0229 33.8 7.09e-168 0.730 0.820
#> 4 triples 1.24 0.0778 15.9 4.62e- 51 1.09 1.39
#> 5 hr 1.44 0.0248 58.1 1.98e-323 1.39 1.49

```

To see how well our metric actually predicts runs, we can predict the number of runs for each team in 2002 using the function `predict`, then make a plot:

```

dat |> mutate(r_hat = predict(fit, newdata = dat)) |>
 filter(year == 2002) %>%
 ggplot(aes(r_hat, r, label = team)) +
 geom_point() +
 geom_text(nudge_x = 0.1, cex = 2) +
 geom_abline()

```

Our model does quite a good job, as demonstrated by the fact that points from the observed versus predicted plot fall close to the identity line.

So instead of using batting average, or just number of HR, as a measure of picking players, we can use our fitted model to form a metric that relates more directly to run production. Specifically, to define a metric for player A, we imagine a team made up of players just like player A, and use our fitted regression model to predict how many runs

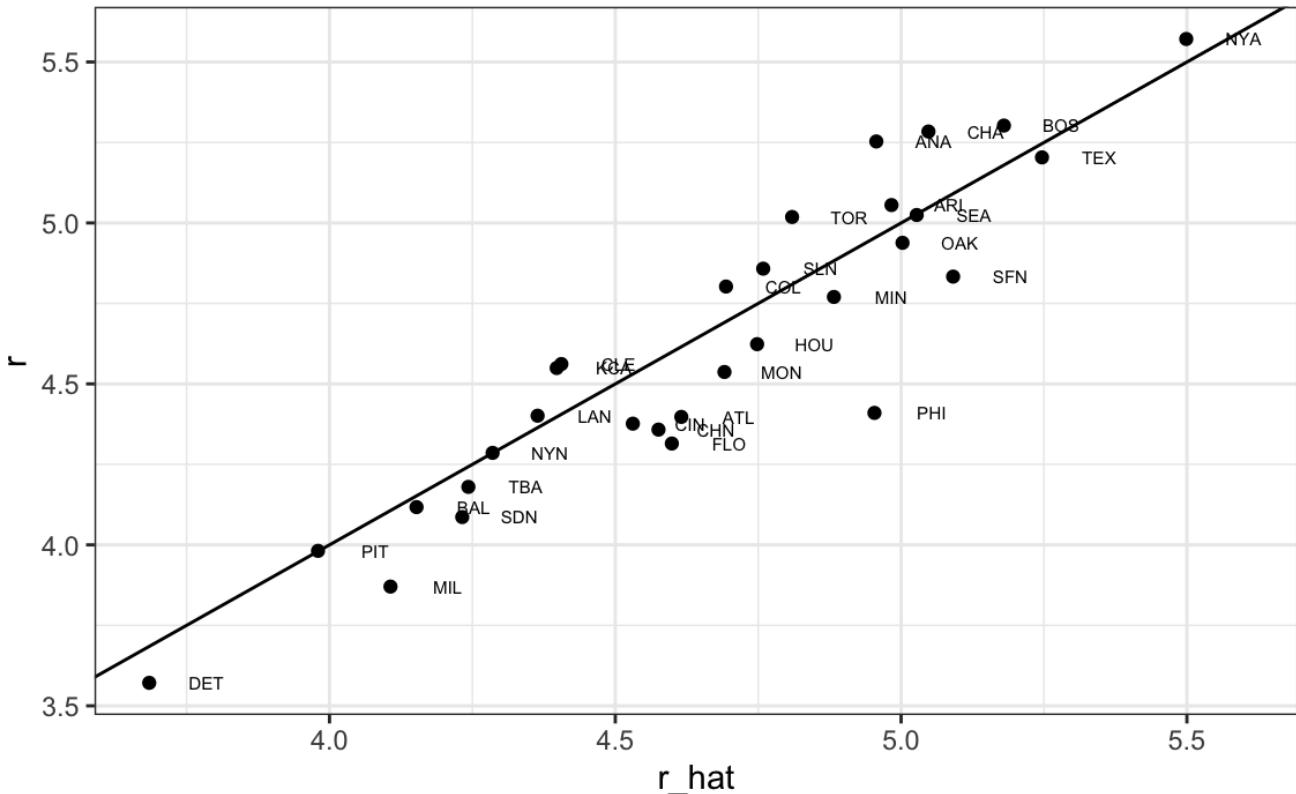


Figure 12:

this team would produce. The formula would look like this:  $-2.7580763 + 0.3699921 \times BB + 0.5174284 \times singles + 0.7750757 \times doubles + 1.2387738 \times triples + 1.4419724 \times HR$ .

To define a player-specific metric, we have a bit more work to do. A challenge here is that we derived the metric for teams, based on team-level summary statistics. For example, the HR value that is entered into the equation is HR per game for the entire team. If we compute the HR per game for a player, it will be much lower since the total is accumulated by 9 batters. Furthermore, if a player only plays part of the game and gets fewer opportunities than average, it is still considered a game played. For players, a rate that takes into account opportunities is the per-plate-appearance rate.

To make the per-game team rate comparable to the per-plate-appearance player rate, we compute the average number of team plate appearances per game:

```
pa_per_game <- Batting |> filter(yearID == 2002) |>
 group_by(teamID) |>
 summarize(pa_per_game = sum(AB + BB)/162) |>
 pull(pa_per_game) |>
 mean()
```

We compute the per-plate-appearance rates for players available in 2002 on data from 1997-2001. To avoid small sample artifacts, we filter players with less than 1,000 plate appearances per year. Here is the entire calculation in one line:

```
players <- Batting |>
 filter(yearID %in% 1997:2001) |>
 group_by(playerID) |>
 mutate(pa = BB + AB) |>
 summarize(g = sum(pa)/pa_per_game,
 bb = sum(BB)/g,
 singles = sum(H - X2B - X3B - HR)/g,
```

```

doubles = sum(X2B)/g,
triples = sum(X3B)/g,
hr = sum(HR)/g,
avg = sum(H)/sum(AB),
pa = sum(pa)) |>
filter(pa >= 1000) |>
select(-g)

players$r_hat = predict(fit, newdata = players)

```

The player-specific predicted runs computed here can be interpreted as the number of runs we predict a team will score if all batters are exactly like that player. The distribution shows that there is wide variability across players:

```
hist(players$r_hat, main = "Predicted runs per game")
```

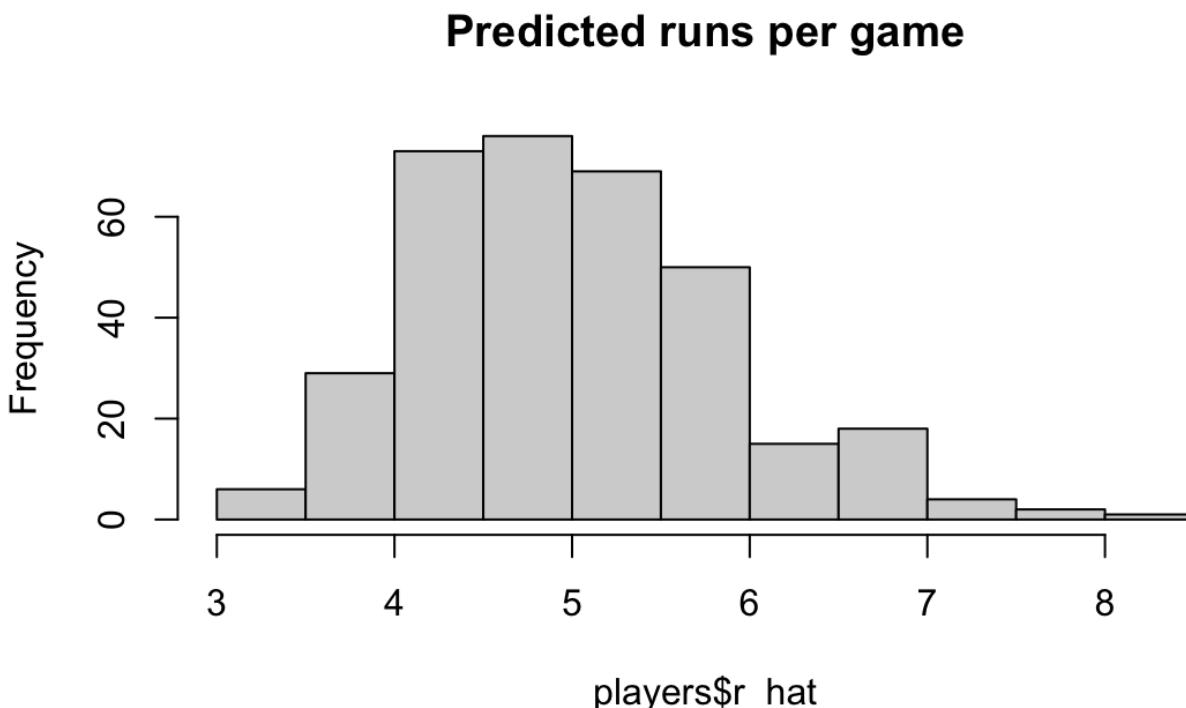


Figure 13:

To actually build the team, we will need to know their salaries as well as their defensive position. For this, we use the `right_join` function to combine the `players` data frame we just created with the player information data frame included in some of the other Lahman data tables.

Start by adding the 2002 salary of each player:

```

players <- Salaries |>
 filter(yearID == 2002) |>
 select(playerID, salary) |>
 right_join(players, by = "playerID")

```

Next, we add their defensive position. This is a somewhat complicated task because players play more than one position each year. The `Lahman` package table `Appearances` specifies how many games each player played in each position, allowing us to pick the position that was most played using `which.max` on each row. We use `apply` to do this. However, as some players are traded and appear more than once on the table, we first sum their appearances

across teams. Here, we pick the one position the player most played using the `top_n` function. To make sure we only pick one position, in the case of ties, we pick the first row of the resulting data frame. We also remove the OF position which stands for outfielder, a generalization of three positions: left field (LF), center field (CF), and right field (RF). We also remove pitchers since they don't bat in the league where the A's play.

```
position_names <-
 paste0("G_", c("p", "c", "1b", "2b", "3b", "ss", "lf", "cf", "rf", "dh"))

tmp <- Appearances |>
 filter(yearID == 2002) |>
 group_by(playerID) |>
 summarize_at(position_names, sum) |>
 ungroup()

pos <- tmp |>
 select(all_of(position_names)) |>
 apply(X = _, 1, which.max)

players <- tibble(playerID = tmp$playerID, POS = position_names[pos]) |>
 mutate(POS = str_to_upper(str_remove(POS, "G_"))) |>
 filter(POS != "P") |>
 right_join(players, by = "playerID") |>
 filter(!is.na(POS) & !is.na(salary))
```

Finally, we add their first and last name:

```
players <- People |>
 select(playerID, nameFirst, nameLast, debut) |>
 mutate(debut = as.Date(debut)) |>
 right_join(players, by = "playerID")
```

If you are a baseball fan, you will recognize the top 10 players:

```
players |> select(nameFirst, nameLast, POS, salary, r_hat) |> arrange(desc(r_hat)) |> head(10)
#> nameFirst nameLast POS salary r_hat
#> 1 Barry Bonds LF 15000000 8.05
#> 2 Larry Walker RF 12666667 7.96
#> 3 Todd Helton 1B 5000000 7.40
#> 4 Manny Ramirez LF 15462727 7.35
#> 5 Sammy Sosa RF 15000000 7.20
#> 6 Jeff Bagwell 1B 11000000 7.05
#> 7 Mike Piazza C 10571429 6.99
#> 8 Jason Giambi 1B 10428571 6.92
#> 9 Edgar Martinez DH 7086668 6.91
#> 10 Jim Thome 1B 8000000 6.89
```

On average, players with a higher metric have higher salaries:

```
players |> ggplot(aes(salary, r_hat, color = POS)) +
 geom_point() +
 scale_x_log10()
```

We can search for good deals by looking at players who generate many more runs than others with similar salaries. We can use this table to decide what players to pick while keeping our total salary below the 40 million dollar budget Billy Beane had to work with. This can be done using what computer scientists call linear programming. This is not something we teach, although here are the position players selected with this approach:

| nameFirst | nameLast | POS | salary   | r_hat |
|-----------|----------|-----|----------|-------|
| Todd      | Helton   | 1B  | 5000000  | 7.40  |
| Mike      | Piazza   | C   | 10571429 | 6.99  |

| nameFirst | nameLast  | POS | salary  | r_hat |
|-----------|-----------|-----|---------|-------|
| Edgar     | Martinez  | DH  | 7086668 | 6.91  |
| Jim       | Edmonds   | CF  | 7333333 | 6.23  |
| Jeff      | Kent      | 2B  | 6000000 | 6.08  |
| Phil      | Nevin     | 3B  | 2600000 | 5.86  |
| Matt      | Stairs    | RF  | 500000  | 5.76  |
| Henry     | Rodriguez | LF  | 300000  | 5.64  |
| John      | Valentin  | SS  | 550000  | 5.00  |

We see that all these players have above average BB and most have above average HR rates, while the same is not true for singles and batting average. Below is a table with statistics standardized across players so that, for example, above average HR hitters have values above 0:

| nameLast  | bb    | singles | doubles | triples | hr     | avg    | r_hat  |
|-----------|-------|---------|---------|---------|--------|--------|--------|
| Helton    | 0.909 | -0.215  | 2.649   | -0.311  | 1.522  | 2.670  | 2.542  |
| Piazza    | 0.328 | 0.423   | 0.204   | -1.418  | 1.825  | 2.199  | 2.093  |
| Martinez  | 2.135 | -0.005  | 1.265   | -1.224  | 0.808  | 2.203  | 2.004  |
| Edmonds   | 1.071 | -0.558  | 0.791   | -1.152  | 0.973  | 0.854  | 1.259  |
| Kent      | 0.232 | -0.732  | 2.011   | 0.448   | 0.766  | 0.787  | 1.093  |
| Nevin     | 0.307 | -0.905  | 0.479   | -1.191  | 1.193  | 0.105  | 0.850  |
| Stairs    | 1.100 | -1.513  | -0.046  | -1.129  | 1.121  | -0.561 | 0.742  |
| Rodriguez | 0.201 | -1.596  | 0.332   | -0.782  | 1.320  | -0.672 | 0.613  |
| Valentin  | 0.180 | -0.929  | 1.794   | -0.435  | -0.045 | -0.472 | -0.088 |

## 15.5 Exercises

We have shown how BB and singles have similar predictive power for scoring runs. Another way to compare the usefulness of these baseball metrics is by assessing their stability across the years. Since we have to pick players based on their previous performances, we prefer metrics that are more stable. In these exercises, we will compare the stability of singles and BBs.

- Before we begin, we want to generate two tables. One for 2002 and another for the average of 1999-2001 seasons. We want to define per plate appearance statistics. Here is how we create the 2017 table, keeping only players with more than 100 plate appearances:

```
library(Lahman)
dat <- Batting |> filter(yearID == 2002) |>
 mutate(pa = AB + BB,
 singles = (H - X2B - X3B - HR)/pa, bb = BB/pa) |>
 filter(pa >= 100) |>
 select(playerID, singles, bb)
```

Now, compute a similar table, but with rates computed over 1999-2001.

- You can use the `inner_join` function to combine the 2001 data and averages in the same table:

```
dat <- inner_join(dat, avg, by = "playerID")
```

Compute the correlation between 2002 and the previous seasons for singles and BB.

- Note that the correlation is higher for BB. To quickly get an idea of the uncertainty associated with this correlation estimate, we will fit a linear model and compute confidence intervals for the slope coefficient. However, first make scatterplots to confirm that fitting a linear model is appropriate.

- Now fit a linear model for each metric and use the `confint` function to compare the estimates.

- In a previous section, we computed the correlation between mothers and daughters, mothers and sons, fathers and daughters, and fathers and sons. We noticed that the highest correlation is between fathers and sons and the lowest is between mothers and sons. We can compute these correlations using:

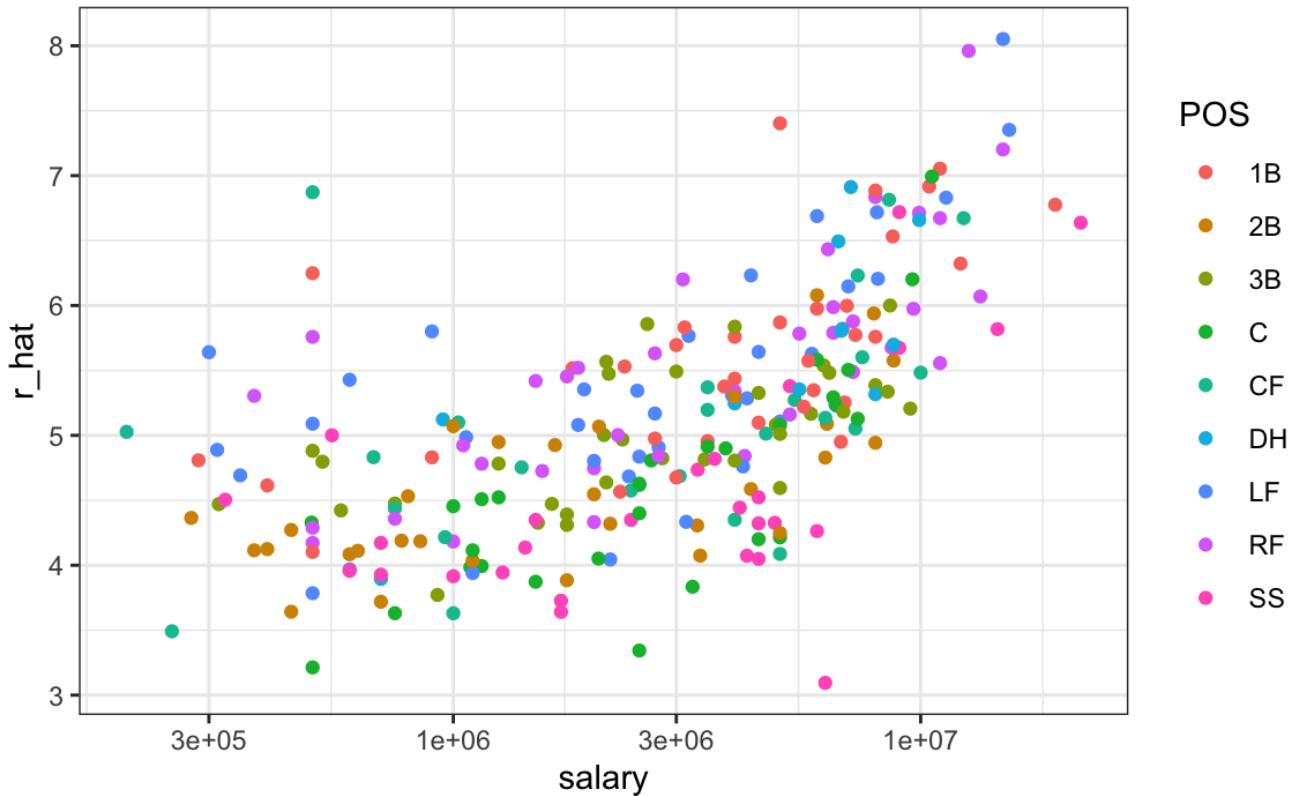


Figure 14:

```

library(HistData)
set.seed(1)
galton_heights <- GaltonFamilies |>
 group_by(family, gender) |>
 sample_n(1) |>
 ungroup()

cors <- galton_heights |>
 pivot_longer(father:mother, names_to = "parent", values_to = "parentHeight") |>
 mutate(child = ifelse(gender == "female", "daughter", "son")) |>
 unite(pair, c("parent", "child")) |>
 group_by(pair) |>
 summarize(cor = cor(parentHeight, childHeight))

```

Are these differences statistically significant? To answer this, we will compute the slopes of the regression line along with their standard errors. Start by using `lm` and the `broom` package to compute the slopes LSE and the standard errors.

6. Repeat the exercise above, but compute a confidence interval as well.
7. Plot the confidence intervals and notice that they overlap, which implies that the data is consistent with the inheritance of height being independent of sex.
8. Because we are selecting children at random, we can actually do something like a permutation test here. Repeat the computation of correlations 100 times taking a different sample each time. Hint: Use similar code to what we used with simulations.
9. Fit a linear regression model to obtain the effects of BB and HR on Runs (at the team level) in 1971. Use the `tidy` function in the `broom` package to obtain the results in a data frame.

10. Now let's repeat the above for each year since 1962 and make a plot. Use `summarize` and the `broom` package to fit this model for every year since 1962.
  11. Use the results of the previous exercise to plot the estimated effects of BB on runs.
  12. **Advanced.** Write a function that takes R, HR, and BB as arguments and fits two linear models:  $R \sim BB$  and  $R \sim BB + HR$ . Then use the `summary` function to obtain the BB for both models for each year since 1962. Then plot these against each other as a function of time.
  13. Since the 1980s, sabermetricians have used a summary statistic different from batting average to evaluate players. They realized walks were important and that doubles, triples, and HRs, should be weighed more than singles. As a result, they proposed the following metric:
- $$OPS = \frac{BB}{PA} + \frac{Singles + 2 \cdot Doubles + 3 \cdot Triples + 4 \cdot HR}{AB}$$
- They called this on-base-percentage plus slugging percentage (OPS). Although the sabermetricians probably did not use regression, here we demonstrate how this metric closely aligns with regression results.
- Compute the OPS for each team in the 2001 season. Then plot Runs per game versus OPS.
14. For every year since 1962, compute the correlation between runs per game and OPS. Then plot these correlations as a function of year.
  15. Keep in mind that we can rewrite OPS as a weighted average of BBs, singles, doubles, triples, and HRs. We know that the weights for doubles, triples, and HRs are 2, 3, and 4 times that of singles. But what about BB? What is the weight for BB relative to singles? Hint: The weight for BB relative to singles will be a function of AB and PA.
  16. Consider that the weight for BB,  $\frac{BB}{PA}$ , will change from team to team. To assess its variability, compute and plot this quantity for each team for each year since 1962. Then plot it again, but instead of computing it for every team, compute and plot the ratio for the entire year. Then, once you are convinced that there is not much of a time or team trend, report the overall average.
  17. So now we know that the formula for OPS is proportional to  $(0.91 \times BB + \text{singles} + 2 \times \text{doubles} + 3 \times \text{triples} + 4 \times \text{HR})$ . Let's see how these coefficients compare to those obtained with regression. Fit a regression model to the data after 1962, as done earlier: using per game statistics for each year for each team. After fitting this model, report the coefficients as weights relative to the coefficient for singles.
  18. We see that our linear regression model coefficients follow the same general trend as those used by OPS, but with slightly less weight for metrics other than singles. For each team in years after 1962, compute the OPS, the predicted runs with the regression model, and compute the correlation between the two, as well as the correlation with runs per game.
  19. We see that using the regression approach predicts runs slightly better than OPS, but not that much. However, note that we have been computing OPS and predicting runs for teams when these measures are used to evaluate players. Let's show that OPS is quite similar to what one obtains with regression at the player level. For the 1962 season and onward, compute the OPS and the predicted runs from our model for each player, and plot them. Use the PA per game correction we used in the previous chapter.
  20. Which players have shown the largest difference between their rank by predicted runs and OPS?
- 

1. [http://mlb.mlb.com/stats/league\\_leaders.jsp](http://mlb.mlb.com/stats/league_leaders.jsp)
2. [https://en.wikipedia.org/wiki/Bill\\_James](https://en.wikipedia.org/wiki/Bill_James)
3. <https://en.wikipedia.org/wiki/Sabermetrics>
4. <https://en.wikipedia.org/wiki/User:Cburnett>
5. <https://creativecommons.org/licenses/by-sa/3.0/deed.en>
6. <https://www.flickr.com/people/27003603@N00>
7. <https://creativecommons.org/licenses/by-sa/2.0>

8. [http://www.baseball-almanac.com/awards/lou\\_brock\\_award.shtml](http://www.baseball-almanac.com/awards/lou_brock_award.shtml)

# Introduction to Data Science - 16 Measurement error models

Rafael A. Irizarry

Another major application of linear models occurs in measurement errors models. In these situations, non-random covariates, such as time, are frequently encountered, with randomness often arising from measurement errors rather than from sampling or inherent natural variability.

## 16.1 Example: modeling a falling object

To understand these models, imagine you are Galileo in the 16th century trying to describe the velocity of a falling object. An assistant climbs the Tower of Pisa and drops a ball, while several other assistants record the position at different times. Let's simulate some data using the equations we currently know and adding some measurement error. The `dslabs` function `rfalling_object` generates these simulations:

```
library(tidyverse)
library(broom)
library(dslabs)
falling_object <- rfalling_object()
```

The assistants hand the data to Galileo, and this is what he sees:

```
falling_object |>
 ggplot(aes(time, observed_distance)) +
 geom_point() +
 ylab("Distance in meters") +
 xlab("Time in seconds")
```

Galileo does not know the exact equation, but by looking at the plot above, he deduces that the position should follow a parabola, which we can write like this:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

The data does not fall exactly on a parabola. Galileo knows this is due to measurement error. His helpers make mistakes when measuring the distance. To account for this, he models the data with:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, i=1, \dots, n$$

with  $(Y_i)$  representing distance in meters,  $(x_i)$  representing time in seconds, and  $(\varepsilon_i)$  accounting for measurement error. The measurement error is assumed to be random, independent from each other, and having the same distribution for each  $(i)$ . We also assume that there is no bias, which means the expected value  $E[\varepsilon_i] = 0$ .

Note that this is a linear model because it is a linear combination of known quantities ( $x$  and  $x^2$ ) are known and unknown parameters (the  $\beta$ s are unknown parameters to Galileo). Unlike our previous examples, here  $x$  is a fixed quantity; we are not conditioning.

## 16.2 Estimating parameters with least squares

To pose a new physical theory and start making predictions about other falling objects, Galileo needs actual numbers, rather than unknown parameters. Using LSE seems like a reasonable approach. How do we find the LSE?

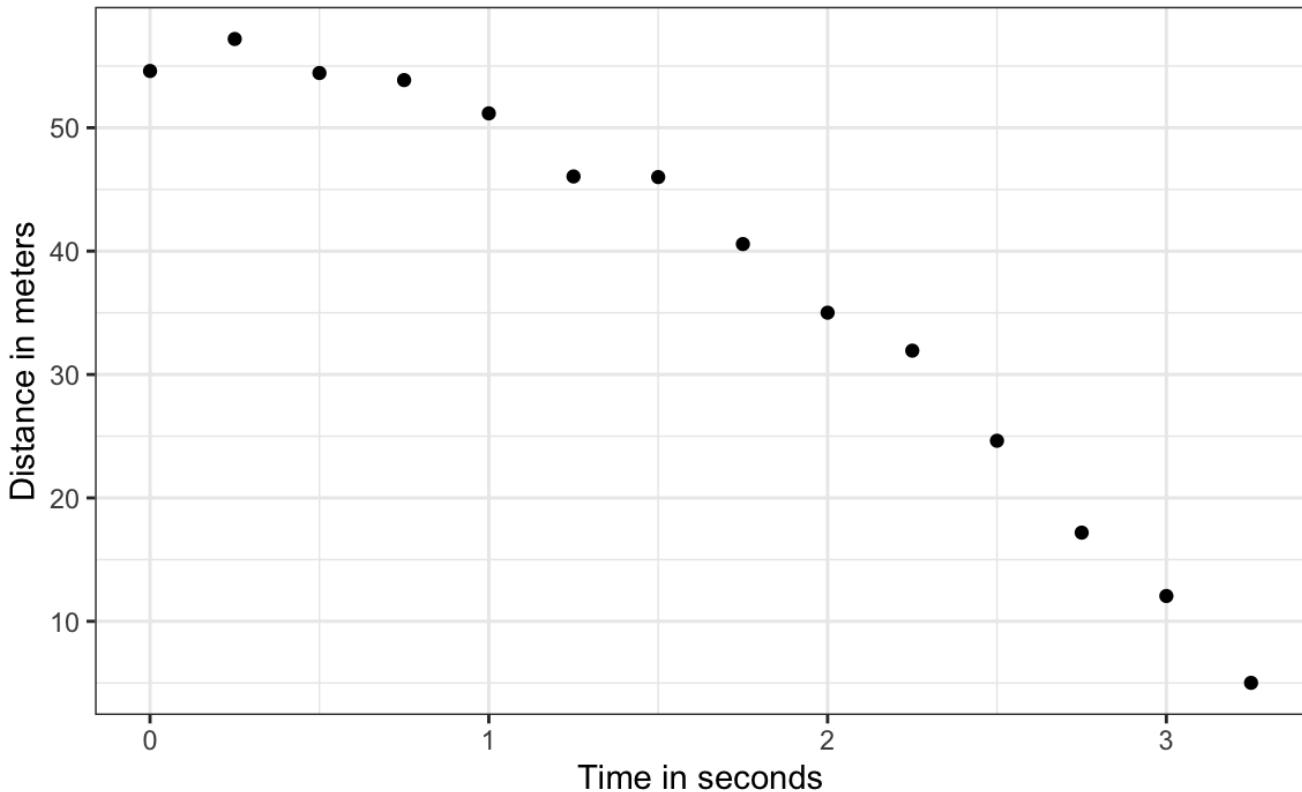


Figure 1:

LSE calculations do not require the errors to be approximately normal. The `lm` function will find the  $(\beta)$ s that will minimize the residual sum of squares:

```
fit <- falling_object |>
 mutate(time_sq = time^2) |>
 lm(observed_distance ~ time + time_sq, data = _)
tidy(fit)
#> # A tibble: 3 × 5
#> term estimate std.error statistic p.value
#> <chr> <dbl> <dbl> <dbl> <dbl>
#> 1 (Intercept) 56.1 0.861 65.1 1.38e-15
#> 2 time -0.618 1.23 -0.503 6.25e- 1
#> 3 time_sq -4.72 0.365 -12.9 5.37e- 8
```

Let's check if the estimated parabola fits the data. The `broom` function `augment` allows us to do this easily:

```
augment(fit) |>
 ggplot() +
 geom_point(aes(time, observed_distance)) +
 geom_line(aes(time, .fitted), col = "blue")
```

Thanks to my high school physics teacher, I know that the equation for the trajectory of a falling object is:

$$d(t) = h_0 + v_0 t - 0.5 \times 9.8 \times t^2$$

with  $(h_0)$  and  $(v_0)$  the starting height and velocity, respectively. The data we simulated above followed this equation, adding measurement error to simulate  $n$  observations for dropping the ball ( $(v_0=0)$ ) from the tower of Pisa ( $(h_0=55.86)$ ).

These are consistent with the parameter estimates:

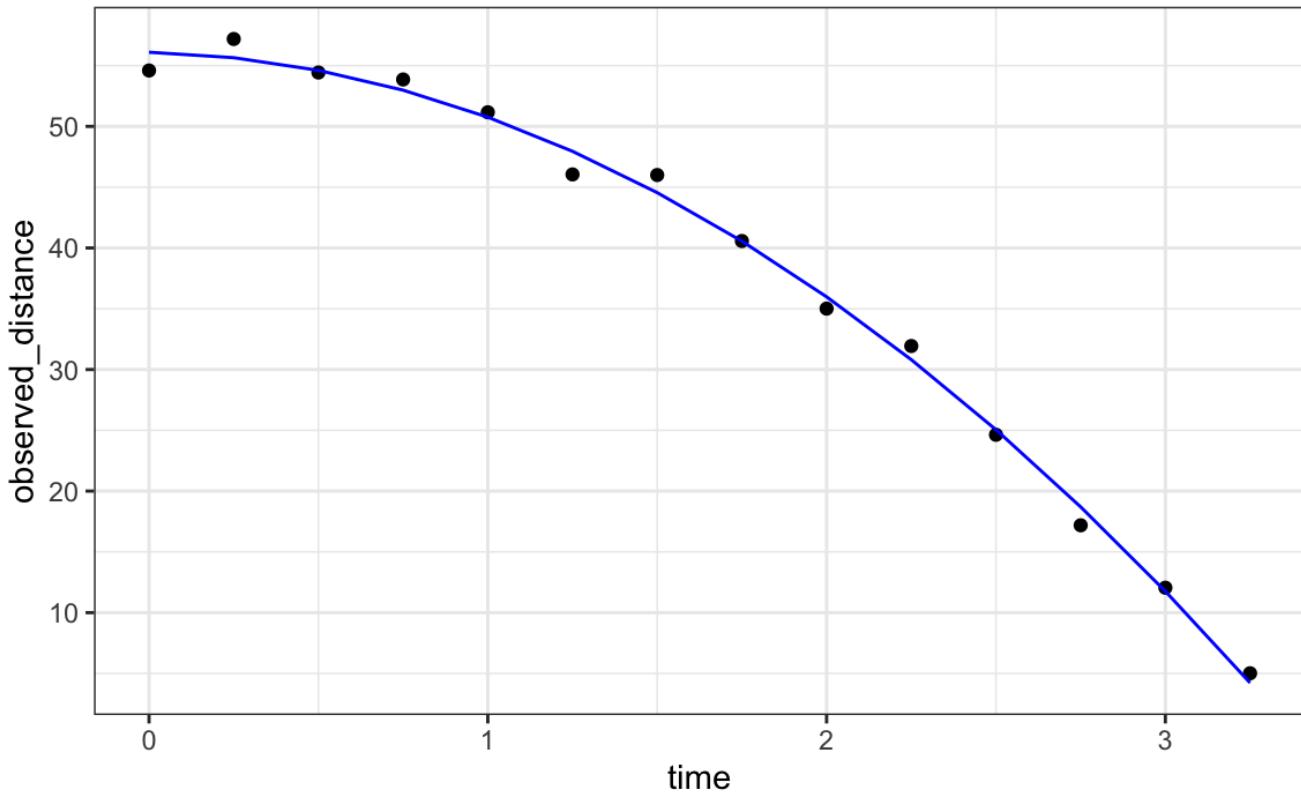


Figure 2:

```
tidy(fit, conf.int = TRUE)
#> # A tibble: 3 × 7
#> term estimate std.error statistic p.value conf.low conf.high
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 (Intercept) 56.1 0.861 65.1 1.38e-15 54.2 58.0
#> 2 time -0.618 1.23 -0.503 6.25e- 1 -3.33 2.09
#> 3 time_sq -4.72 0.365 -12.9 5.37e- 8 -5.52 -3.92
```

The Tower of Pisa height is within the confidence interval for  $(\beta_0)$ , the initial velocity 0 is in the confidence interval for  $(\beta_1)$  (note the p-value is larger than 0.05), and the acceleration constant is in a confidence interval for  $(-2 \times \beta_2)$ .

### 16.3 Exercises

- Plot CO2 levels for the first 12 months of the `co2` dataset and notice it seems to follow a sin wave with a frequency of 1 cycle per month. This means that a measurement error model that might work is

$y_i = \mu + A \sin(2\pi t_i / 12 + \phi) + \epsilon_i$  with  $t_i$  the month number for observation  $i$ . Is this a linear model for the parameters  $(\mu)$ ,  $(A)$  and  $(\phi)$ ?

- Using trigonometry, we can show that we can rewrite this model as:

$$y_i = \beta_0 + \beta_1 \sin(2\pi t_i / 12) + \beta_2 \cos(2\pi t_i / 12) + \epsilon_i$$

Is this a linear model?

- Find least square estimates for the  $(\beta)$ s using `lm`. Show a plot of  $(y_i)$  versus  $(t_i)$  with a curve on the same plot showing  $(\hat{Y}_i)$  versus  $(t_i)$ .

- Now fit a measurement error model to the entire `co2` dataset that includes a trend term that is a parabola as well as the sine wave model.

5. Run diagnostic plots for the fitted model and describe the results.

# Introduction to Data Science - 17 Treatment effect models

Rafael A. Irizarry

Up to now, all our linear models have been applied to two or more continuous random variables. We assume the random variables are multivariate normal and use this to motivate a linear model. This approach covers many real-life examples of linear regression. However, linear models have many other applications. One of the most popular is to quantify treatment effects in randomized and controlled experiments. One of the first applications was in agriculture, where different plots of lands were treated with different combinations of fertilizers to try to determine if they were effective. In fact, the use of  $\backslash(Y\backslash)$  for the outcome in statistics is due to the mathematical theory being developed for crop *yield* as the outcome.

Since then, the same ideas have been applied in other areas, such as randomized trials developed to determine if drugs cure or prevent diseases or if policies have an effect on social or educational outcomes. In the latter example, we think of the policy intervention as a *treatment* and follow the same mathematical procedure. The analyses used in *A/B testing*, widely used today by internet companies, are based on treatment effects models.

Moreover, these models have been applied in observational studies where analysts attempt to use linear models to estimate effects of interest while accounting for potential confounders. For example, to estimate the effect of a diet high in fruits and vegetables on blood pressure, we would have to adjust for *factors* such as age, sex, and smoking status.

In this chapter, we consider an experiment designed to test for the effects of a high-fat diet on mouse physiology. Mice were randomly selected and divided into two groups: one group receiving a high-fat diet, considered the *treatment*, while the other group served as the control and received the usual *chow* diet. The data is included in the **dslabs** package:

```
library(dslabs)
table(mice_weights$diet)
#>
#> chow hf
#> 394 386
```

A boxplot shows that the high fat diet mice are, on average, heavier.

```
with(mice_weights, boxplot(body_weight ~ diet))
```

However, given that we divided the mice randomly, is it possible that the observed difference is simply due to chance? Here, we can compute the sample average and standard deviation of each group and perform statistical inference on the difference of these means, similar to our approach for election forecasting in Chapter 9 and Chapter 11.

## 17.1 Comparing group means

The sample averages for the two groups, high-fat and chow diets, are different:

```
library(tidyverse)
mice_weights |> group_by(diet) |> summarize(average = mean(body_weight))
#> # A tibble: 2 × 2
#> diet average
#> <fct> <dbl>
#> 1 chow 31.5
#> 2 hf 36.7
```

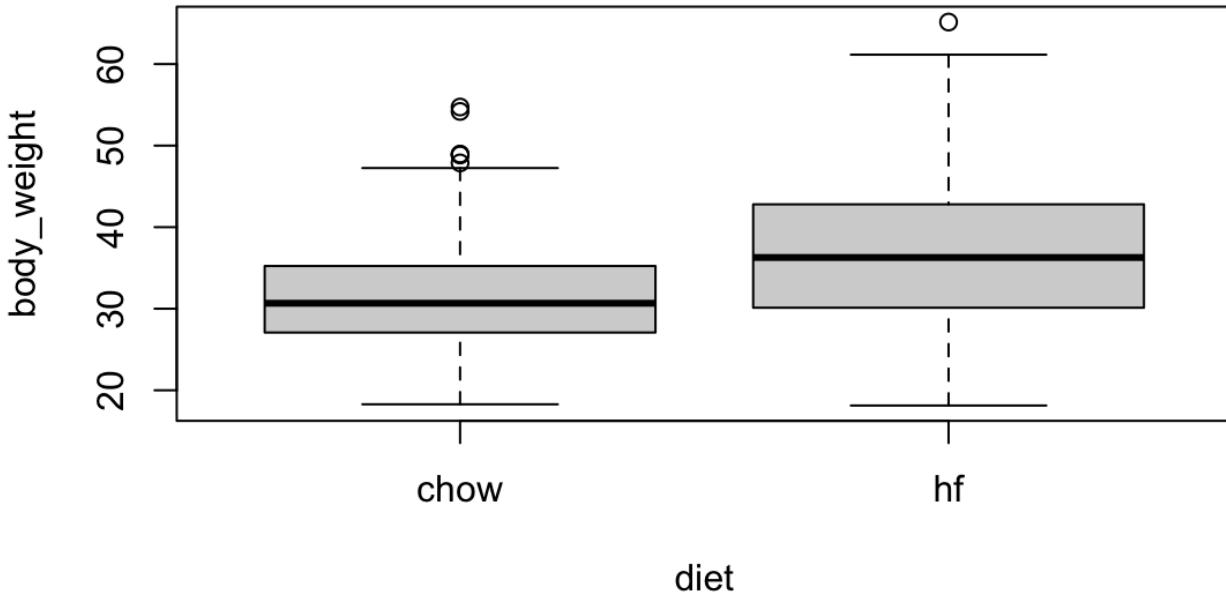


Figure 1:

However, this is a random sample of mice, and the assignment to the diet group is also done randomly. So is this difference due to chance? We will use hypothesis testing, first described in Chapter 9, to answer this question.

Let  $(\mu_1)$  and  $(\sigma_1)$  represent the weight average and standard deviation, respectively, that we would observe if the entire population of mice were on the high-fat diet. Define  $(\mu_0)$  and  $(\sigma_0)$  similarly, but for the chow diet. Define  $(N_1)$  and  $(N_0)$  as the sample sizes, and  $(\bar{X}_1)$  and  $(\bar{X}_0)$  the sample averages, for the for the high-fat and chow diets, respectively.

Since the data comes from a random sample, the central limit theorem tells us that, if the sample is large enough, the difference in averages  $(\bar{X}_1 - \bar{X}_0)$  follows a normal distribution, with expected value  $(\mu_1 - \mu_0)$  and standard error  $(\sqrt{\frac{\sigma_1^2}{N_1} + \frac{\sigma_0^2}{N_0}})$ .

If we define the null hypothesis as the high-fat diet having no effect, or  $(\mu_1 - \mu_0 = 0)$ , this implies that

$$[(\frac{\bar{X}_1 - \bar{X}_0}{\sqrt{\frac{\sigma_1^2}{N_1} + \frac{\sigma_0^2}{N_0}}})]$$

has expected value 0 and standard error 1 and therefore approximately follows a standard normal distribution.

Note that we can't compute this quantity in practice because the  $(\sigma_1)$  and  $(\sigma_0)$  are unknown. However, if we estimate them with the sample standard deviations, denote them  $(s_1)$  and  $(s_0)$  for the high-fat and chow diets, respectively, the central limit still holds and tells us that

$$[ t = \frac{\bar{X}_1 - \bar{X}_0}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_0^2}{N_0}}} ]$$

follows a standard normal distribution when the null hypothesis is true. This implies that we can easily compute the probability of observing a value as large as the one we obtained:

```
stats <- mice_weights |>
 group_by(diet) |>
 summarize(xbar = mean(body_weight), s = sd(body_weight), n = n())
t_stat <- with(stats, (xbar[2] - xbar[1]) / sqrt(s[2]^2/n[2] + s[1]^2/n[1]))
```

```
t_stat
#> [1] 9.34
```

Here  $|t|$  is well over 3, so we don't really need to compute the p-value  $1 - \text{pnorm}(t_{\text{stat}})$  as we know it will be very small.

Note that when  $|N|$  is not large enough, then the CLT does not apply. However, if the outcome data, in this case weight, follows a normal distribution, then  $|t|$  follows a t-distribution with  $(N_1 + N_2 - 2)$  degrees of freedom. So the calculation of the p-value is the same except that we use `pt` instead of `pnorm`. Specifically, we use `1-pt(t_stat, with(stats, n[2]+n[1]-2))`.

Differences in means are commonly examined in the scientific studies. As a result this *t-statistic* is one of the most widely reported summaries. When used to determine if an observed difference is *statistically significant*, we refer to the procedure as "performing a *t test*".

In the computation above, we computed the probability of  $t$  being as large as what we observed. However, when our interest spans both directions, for example, either an increase or decrease in weight, we need to compute the probability of  $t$  being *as extreme* as what we observe. The formula simply changes to using the absolute value:  $1 - \text{pnorm}(\text{abs}(t_{\text{test}}))$  or `1-pt(abs(t_stat), with(stats, n[2]+n[1]-2))`.

## 17.2 One factor design

Although the t-test is useful for cases in which we compare two treatments, it is common to have other variables affect our outcomes. Linear models permit hypothesis testing in these more general situations. We start the description of the use of linear models for estimating treatment effects by demonstrating how they can be used to perform t-tests.

If we assume that the weight distributions for both chow and high-fat diets are normally distributed, we can write the following linear model to represent the data:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

with  $x_i$  1, if the  $i$ -th mice was fed the high-fat diet, and 0 otherwise, and the errors  $\varepsilon_i$  independent and normally distributed with expected value 0 and standard deviation  $\sigma$ . Note that this mathematical formula looks exactly like the model we wrote out for the father-son heights. However, the fact that  $x_i$  is now 0 or 1 rather than a continuous variable, allows us to use it in this different context. In particular, notice that now  $\beta_0$  represents the population average height of the mice on the chow diet and  $\beta_0 + \beta_1$  represents the population average for the weight of the mice on the high-fat diet.

A nice feature of this model is that  $\beta_1$  represents the *treatment effect* of receiving the high-fat diet. The null hypothesis that the high-fat diet has no effect can be quantified as  $\beta_1 = 0$ . To perform hypothesis testing on the effect of the high fat diet we can estimate  $\beta_1$  and compute the probability of an estimates being as large as the observed when the null hypothesis is true. So how do we estimate  $\beta_1$  and compute this probability?

A powerful characteristic of linear models is that we can estimate the  $\beta$ s and their standard errors with the same LSE machinery:

```
fit <- lm(body_weight ~ diet, data = mice_weights)
```

Because `diet` is a factor with two entries, the `lm` function knows to fit the linear model above with a  $x_i$  a indicator variable. The `summary` function shows us the resulting estimate, standard error, and p-value:

```
coefficients(summary(fit))
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 31.54 0.386 81.74 0.00e+00
#> diethf 5.14 0.548 9.36 8.02e-20
```

Using `broom`, we can write:

```
library(broom)
tidy(fit, conf.int = TRUE) |> filter(term == "diethf")
#> # A tibble: 1 × 7
```

```
#> term estimate std.error statistic p.value conf.low conf.high
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 diethf 5.14 0.548 9.36 8.02e-20 4.06 6.21
```

The `statistic` computed here is the estimate divided by its standard error:  $(\hat{\beta}_1 / \text{SE}(\hat{\beta}_1))$ . In the case of the simple one-factor model, we can show that this statistic is almost equivalent to the t-statistics computed in the previous section:

```
c(coefficients(summary(fit))[2,3], t_stat)
#> [1] 9.36 9.34
```

Intuitively, it makes sense, as both  $(\hat{\beta}_1)$  and the numerator of the t-test are estimates of the treatment effect.

The one minor difference is that the linear model does not assume a different standard deviation for each population. Instead, both populations share  $(\text{SD}(\varepsilon))$  as a standard deviation. Note that, although we don't demonstrate it with R here, we can redefine the linear model to have different standard errors for each group.

In the linear model description provided here, we assumed  $(\varepsilon)$  follows a normal distribution. This assumption permits us to show that the statistics formed by dividing estimates by their estimated standard errors follow t-distribution, which in turn allows us to estimate p-values or confidence intervals. However, note that we do not need this assumption to compute the expected value and standard error of the least squared estimates. Furthermore, if the number of observations is large enough, then the central limit theorem applies and we can obtain p-values and confidence intervals even without the normal distribution assumption for the errors.

### 17.3 Two factor designs

Note that this experiment included male and female mice, and male mice are known to be heavier. This explains why the residuals depend on the sex variable:

```
boxplot(fit$residuals ~ mice_weights$sex)
```

This misspecification can have real implications; for instance, if more male mice received the high-fat diet, then this could explain the increase. Conversely, if fewer received it, we might underestimate the diet effect. Sex could be a confounder, indicating that our model can certainly be improved.

From examining the data:

```
mice_weights |> ggplot(aes(diet, log2(body_weight), fill = sex)) + geom_boxplot()
```

we see that the diet effect is observed for both sexes and that males are heavier than females. Although not nearly as obvious, it also appears the diet effect is stronger in males.

A linear model that permits a different expected value for the following four groups, 1) female on chow diet, 2) females on high-fat diet, 3) male on chow diet, and 4) males on high-fat diet, can be written like this:

$$Y_i = \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \varepsilon_i$$

with  $(x_{i,1}, \dots, x_{i,4})$  indicator variables for each of the four groups. Note that with this representation we allow the diet effect to be different for males and females.

However, with this representation, none of the  $(\beta)$ s represent the effect of interest: the diet effect. A powerful feature of linear models is that we can rewrite the model so that the expected value for each group remains the same, but the parameters represent the effects we are interested in. So, for example, in the representation

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,1} x_{i,2} + \varepsilon_i$$

with  $(x_{i,1})$  an indicator that is 1 if individual  $(i)$  is on the high-fat diet  $(x_{i,2})$  an indicator that is 1 if you are male, the  $(\beta_1)$  is interpreted as the diet effect for females,  $(\beta_2)$  as the average difference between males and females, and  $(\beta_3)$  the difference in the diet effect between males and females. In statistics,  $(\beta_3)$  is referred to as an *interaction effect*. The  $(\beta_0)$  is considered the baseline value, which is the average weight of females on the chow diet.

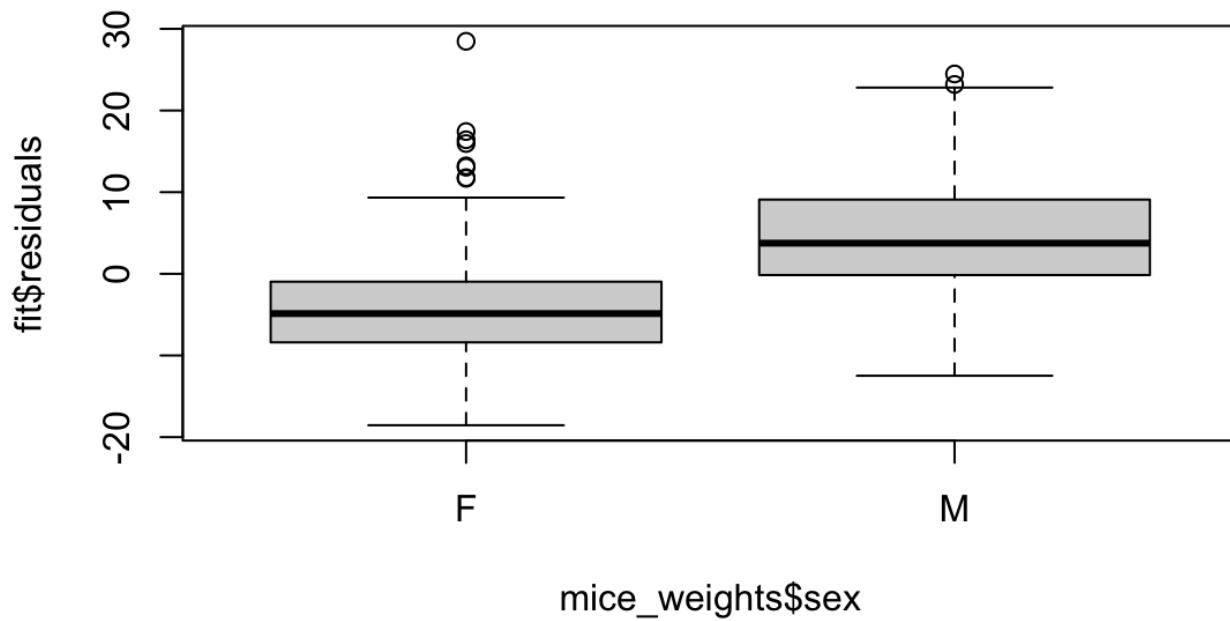


Figure 2:

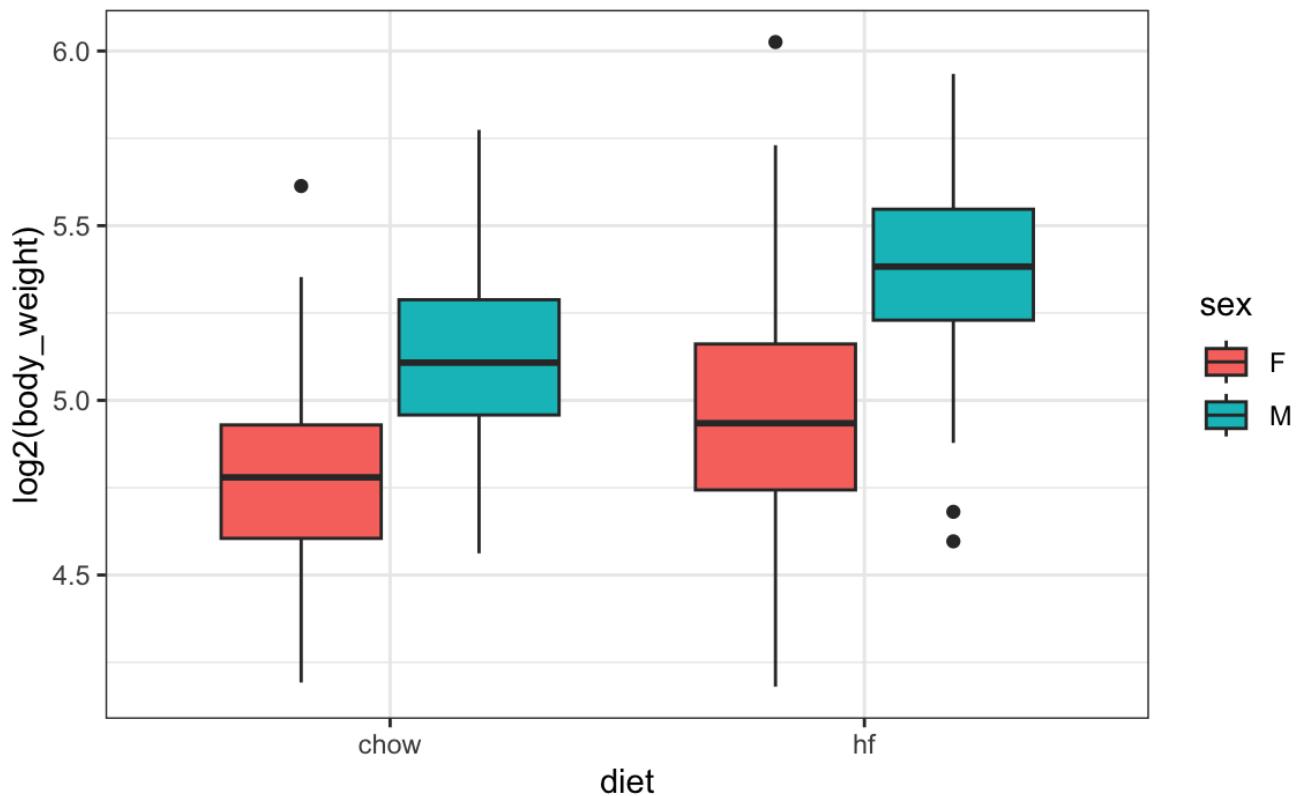


Figure 3:

Statistical textbooks describe several other ways in which the model can be rewritten to obtain other types of interpretations. For example, we might want  $\langle\beta_2\rangle$  to represent the overall diet effect (the average between female and male effect) rather than the diet effect on females. This is achieved by defining what *contrasts* we are interested in.

In R, we can specific the linear model above using the following:

```
fit <- lm(body_weight ~ diet*sex, data = mice_weights)
```

The \* implies that the term that multiplies  $\langle x_{i,1}\rangle$  and  $\langle x_{i,2}\rangle$  should be included, along with the  $\langle x_{i,1}\rangle$  and  $\langle x_{i,2}\rangle$  terms.

```
tidy(fit, conf.int = TRUE) |> filter(!str_detect(term, "Intercept"))
#> # A tibble: 3 × 7
#> term estimate std.error statistic p.value conf.low conf.high
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 diethf 3.88 0.624 6.22 8.02e-10 2.66 5.10
#> 2 sexM 7.53 0.627 12.0 1.27e-30 6.30 8.76
#> 3 diethf:sexM 2.66 0.891 2.99 2.91e- 3 0.912 4.41
```

Note that the male effect is larger than the diet effect, and the diet effect is statistically significant for both sexes, with diet affecting males more by between 1 and 4.5 grams.

A common approach applied when more than one factor is thought to affect the measurement is to simply include an additive effect for each factor, like this:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \varepsilon_i$$

In this model, the  $\langle\beta_1\rangle$  is a general diet effect that applies regardless of sex. In R, we use the following code, employing a + instead of \*:

```
fit <- lm(body_weight ~ diet + sex, data = mice_weights)
```

Note that this model does not account for the difference in diet effect between males and females. Diagnostic plots would reveal this deficiency by showing that the residuals are biased: they are, on average, negative for females on the diet and positive for males on the diet, rather than being centered around 0.

```
plot(fit, which = 1)
```

Scientific studies, particularly within epidemiology and social sciences, frequently omit interaction terms from models due to the high number of variables. Adding interactions necessitates numerous parameters, which in extreme cases may prevent the model from fitting. However, this approach assumes that the interaction terms are zero, and if incorrect, it can skew the interpretation of the results. Conversely, when this assumption is valid, models excluding interactions are simpler to interpret, as parameters are typically viewed as the extent to which the outcome increases with the assigned treatment.

Linear models are highly flexible and applicable in many contexts. For example, we can include many more factors than just 2. We have only just scratched the surface of how linear models can be used to estimate treatment effects. We highly recommend learning more about this by exploring linear model textbooks and R manuals that cover the use of functions such as `lm`, `contrasts`, and `model.matrix`.

## 17.4 Contrasts

In the examples we have examined, each treatment had only two groups: diet had chow/high-fat, and sex had female/male. However, variables of interest often have more than one level. For example, we might have tested a third diet on the mice. In statistics textbooks, these variables are referred to as a *factor*, and the groups in each factor are called its *levels*.

When a factor is included in the formula, the default behavior for `lm` is to define the intercept term as the expected value for the first level, and the other coefficient are to represent the difference, or *contrast*, between the other levels and first. We can see when we estimate the sex effect with `lm` like this:

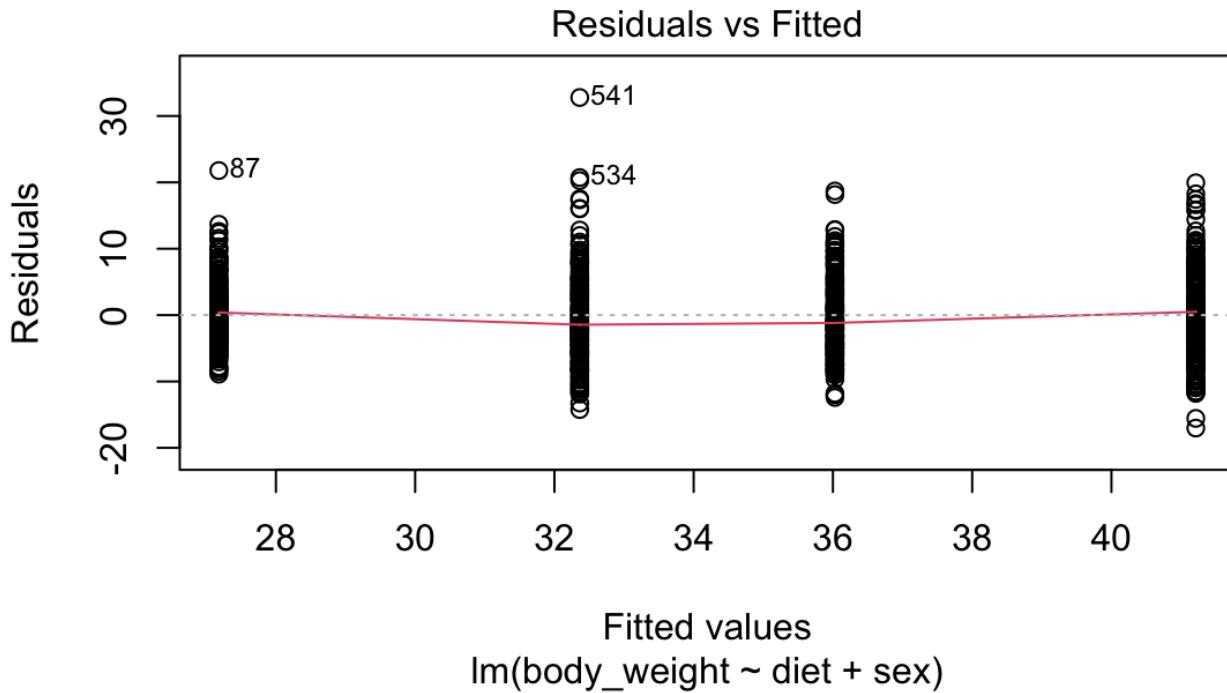


Figure 4:

```
fit <- lm(body_weight ~ sex, data = mice_weights)
coefficients(fit)
#> (Intercept) sexM
#> 29.76 8.82
```

To recover the expected mean for males, we can simply add the two coefficients:

```
sum(fit$coefficients[1:2])
#> [1] 38.6
```

The package **emmeans** simplifies the calculation and also calculates standard errors:

```
library(emmeans)
emmeans(fit, ~sex)
#> sex emmean SE df lower.CL upper.CL
#> F 29.8 0.339 778 29.1 30.4
#> M 38.6 0.346 778 37.9 39.3
#>
#> Confidence level used: 0.95
```

Now, what if we really didn't want to define a reference level? What if we wanted a parameter to represent the difference from each group to the overall mean? Can we write a model like this:

$\{ Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \varepsilon_i \}$  with  $(x_{i,1} = 1)$ , if observation  $(i)$  is female and 0 otherwise, and  $(x_{i,2}=1)$ , if observation  $(i)$  is male and 0 otherwise?

Unfortunately, this representation has a problem. Note that the mean for females and males are represented by  $(\beta_0 + \beta_1)$  and  $(\beta_0 + \beta_2)$ , respectively. This is a problem because the expected value for each group is just one number, say  $(\mu_f)$  and  $(\mu_m)$ , and there is an infinite number of ways  $(\beta_0 + \beta_1 = \mu_f)$  and  $(\beta_0 + \beta_2 = \mu_m)$  (three unknowns with two equations). This implies that we can't obtain a unique least squares estimates. In statistics, we say the model, or parameters, are *unidentifiable*.

The default behavior in R solves this problem by requiring  $(\beta_1 = 0)$ , forcing  $(\beta_0 = \mu_m)$ , which permits us to solve the system of equations.

Keep in mind that this is not the only constraint that permits estimation of the parameters. Any linear constraint will do as it adds a third equation to our system. A widely used constraint is to require  $(\beta_1 + \beta_2 = 0)$ . To achieve this in R, we can use the argument `contrast` in the following way:

```
fit <- lm(body_weight ~ sex, data = mice_weights,
 contrasts = list(sex = contr.sum))
coefficients(fit)
#> (Intercept) sex1
#> 34.17 -4.41
```

We see that the intercept is now larger, reflecting the overall mean rather than just the mean for females. The other coefficient,  $(\beta_1)$ , represents the contrast between females and the overall mean in our model. The coefficient for men is not shown because it is redundant:  $(\beta_1 = -\beta_2)$ .

If we want to see all the estimates, the `emmeans` package also makes the calculations for us:

```
contrast(emmeans(fit, ~sex))
#> contrast estimate SE df t.ratio p.value
#> F effect -4.41 0.242 778 -18.200 <.0001
#> M effect 4.41 0.242 778 18.200 <.0001
#>
#> P value adjustment: fdr method for 2 tests
```

The use of this alternative constraint is more practical when a factor has more than one level, and choosing a baseline becomes less convenient. Furthermore, we might be more interested in the variance of the coefficients rather than the contrasts between groups and the reference level.

As an example, consider that the mice in our dataset are actually from several generations:

```
table(mice_weights$gen)
#>
#> 4 7 8 9 11
#> 97 195 193 97 198
```

To estimate the variability due to the different generations, a convenient model is:

$$[ Y_i = \beta_0 + \sum_{j=1}^J \beta_j x_{i,j} + \varepsilon_i ]$$

with  $(x_{i,j})$  indicator variables:  $(x_{i,j}=1)$  if mouse  $(i)$  is in level  $(j)$  and 0 otherwise,  $(J)$  representing the number of levels, in our example 5 generations, and the level effects constrained with

$$[ \frac{1}{J} \sum_{j=1}^J \beta_j = 0 \text{ implies } \sum_{j=1}^J \beta_j = 0. ]$$

This constraint makes the model identifiable and also allows us to quantify the variability due to generations with:

$$[ \sigma^2_{\text{gen}} \equiv \frac{1}{J} \sum_{j=1}^J \beta_j^2 ]$$

We can see the estimated coefficients using the following:

```
fit <- lm(body_weight ~ gen, data = mice_weights,
 contrasts = list(gen = contr.sum))
contrast(emmeans(fit, ~gen))
#> contrast estimate SE df t.ratio p.value
#> gen4 effect -0.122 0.705 775 -0.174 0.8620
#> gen7 effect -0.812 0.542 775 -1.497 0.3370
#> gen8 effect -0.113 0.544 775 -0.207 0.8620
#> gen9 effect 0.149 0.705 775 0.212 0.8620
#> gen11 effect 0.897 0.540 775 1.663 0.3370
#>
#> P value adjustment: fdr method for 5 tests
```

In the next section, we briefly describe a technique useful to study the variability associated with this factor.

## 17.5 Analysis of variance (ANOVA)

When a factor has more than one level, it is common to want to determine if there is significant variability across the levels rather than specific difference between any given pair of levels. Analysis of variances (ANOVA) provides tools to do this.

ANOVA provides an estimate of  $\sigma^2_{\text{gen}}$  and a statistical test for the null hypothesis that the factor contributes no variability:  $\sigma^2_{\text{gen}} = 0$ .

Once a linear model is fit using one or more factors, the `aov` function can be used to perform ANOVA. Specifically, the estimate of the factor variability is computed along with a statistic that can be used for hypothesis testing:

```
summary(aov(fit))
#> Df Sum Sq Mean Sq F value Pr(>F)
#> gen 4 294 73.5 1.13 0.34
#> Residuals 775 50479 65.1
```

Keep in mind that if given a model formula, `aov` will fit the model:

```
summary(aov(body_weight ~ gen, data = mice_weights))
```

We do not need to specify the constraint because ANOVA needs to constrain the sum to be 0 for the results to be interpretable.

This analysis indicates that generation is not statistically significant.

We do not include many details, for example, on how the summary statistics and p-values shown by `aov` are defined and motivated. There are several books dedicated to the analysis of variance, and textbooks on linear models often include chapters on this topic. Those interested in learning more about these topics can consult one of these textbooks.

### 17.5.1 Multiple factors

ANOVA was developed to analyze agricultural data, which typically included several factors such as fertilizers, blocks of lands, and plant breeds.

Note that we can perform ANOVA with multiple factors:

```
summary(aov(body_weight ~ sex + diet + gen, data = mice_weights))
#> Df Sum Sq Mean Sq F value Pr(>F)
#> sex 1 15165 15165 389.80 <2e-16 ***
#> diet 1 5238 5238 134.64 <2e-16 ***
#> gen 4 295 74 1.89 0.11
#> Residuals 773 30074 39
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This analysis suggests that sex is the biggest source of variability, which is consistent with previously made exploratory plots.

One of the key aspects of ANOVA (Analysis of Variance) is its ability to decompose the total variance in the data, represented by  $\sum_{i=1}^n Y_i^2$ , into individual contributions attributable to each factor in the study. However, for the mathematical underpinnings of ANOVA to be valid, the experimental design must be balanced. This means that for every level of any given factor, there must be an equal representation of the levels of all other factors. In our study involving mice, the design is unbalanced, requiring a cautious approach in the interpretation of the ANOVA results.

### 17.5.2 Array representation

When the model includes more than one factor, writing down linear models can become cumbersome. For example, in our two factor model, we would have to include indicator variables for both factors:

$$\sum_i Y_i = \beta_0 + \sum_j \beta_j x_{i,j} + \sum_k \beta_{j+k} x_{i,J+k} + \varepsilon_i$$

with  $\sum_j x_{i,j}$  indicator functions for the  $J$  levels in the first factor and  $\sum_k x_{i,J+k}$  indicator functions for the  $K$  levels in the second factor.

An alternative approach widely used in ANOVA to avoid indicators variables, is to save the data in an array, using different Greek letters to denote factors and indices to denote levels:

$$Y_{i,j,k} = \mu + \alpha_j + \beta_k + \varepsilon_{i,j,k}$$

with  $\mu$  the overall mean,  $\alpha_j$  the effect of level  $j$  in the first factor, and  $\beta_k$  the effect of level  $k$  in the second factor. The constraint can now be written as:

$$\sum_j \alpha_j = 0 \text{ and } \sum_k \beta_k = 0$$

This notation lends itself to estimating the effects by computing means across dimensions of the array.

## 17.6 Exercises

- Once you fit a model, the estimate of the standard error  $\sigma$  can be obtained as follows:

```
fit <- lm(body_weight ~ diet, data = mice_weights)
summary(fit)$sigma
```

Compute the estimate of  $\sigma$  using both the model that includes only diet and a model that accounts for sex. Are the estimates the same? If not, why not?

- One of the assumption of the linear model fit by `lm` is that the standard deviation of the errors  $\varepsilon_i$  is equal for all  $i$ . This implies that it does not depend on the expected value. Group the mice by their weight like this:

```
breaks <- with(mice_weights, seq(min(body_weight), max(body_weight), 1))
dat <- mutate(mice_weights, group = cut(body_weight, breaks, include_lowest = TRUE))
```

Compute the average and standard deviation for groups with more than 10 observations and use data exploration to verify if this assumption holds.

- The dataset also includes a variable indicating which litter the mice came from. Create a boxplot showing weights by litter. Use faceting to make separate plots for each diet and sex combination.
- Use a linear model to test for a litter effect, taking into account sex and diet. Use ANOVA to compare the variability explained by litter with that of other factors.
- The `mouse_weights` data includes two other outcomes: bone density and percent fat. Create a boxplot illustrating bone density by sex and diet. Compare what the visualizations reveal about the diet effect by sex.
- Fit a linear model and conduct a separate test for the diet effect on bone density for each sex. Note that the diet effect is statistically significant for females but not for males. Then fit the model to the entire dataset that includes diet, sex and their interaction. Notice that the diet effect is significant, yet the interaction effect is not. Explain how this can happen. Hint: To fit a model to the entire dataset with a separate effect for males and females, you can use the formula `~ sex + diet:sex`
- In Chapter 11, we talked about pollster bias and 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:

```
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 pollster bias? Make a plot showing the spreads for each pollster.

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  $(\mu)$ .

To answer the question “is there an urn model?” we will model the observed data  $(Y_{ij})$  in the following way:

$$Y_{ij} = \mu + b_i + \epsilon_{ij}$$

with  $(i=1,2)$  indexing the two pollsters,  $(b_i)$  the bias for pollster  $(i)$ , and  $(\epsilon_{ij})$  poll to poll chance variability. We assume the  $(\epsilon_{ij})$  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  $(\epsilon_{ij}) = 0$ ?
  - b. How close are the  $(Y_{ij})$  to  $(\mu)$ ?
  - c. Is  $(b_1 \neq b_2)$ ?
  - d. Are  $(b_1 = 0)$  and  $(b_2 = 0)$ ?
9. On the right side of this model, only  $(\epsilon_{ij})$  is a random variable; the other two are constants. What is the expected value of  $(Y_{1j})$ ?
10. Suppose we define  $(\bar{Y}_1)$  as the average of poll results from the first poll,  $(Y_{1,1}, \dots, Y_{1,N_1})$ , where  $(N_1)$  is 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, $(Y_{2,1}, \dots, Y_{2,N_2})$, where (N_2) is 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 (σ_1) and (σ_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 normally distributed variables is also normally distributed.
 - d. The data are not 0 or 1, so CLT does not apply.
18. We have constructed a random variable that has an expected value of $(b_2 - b_1)$, representing the difference in pollster bias. If our model holds, then this random variable has an approximately normal distribution, and we know its standard error. The standard error depends on (σ_1) and (σ_2) , but we can plug the sample standard deviations we computed above. We began by asking: is $(b_2 - b_1)$ different from 0? Using all the information we have gathered above, construct a 95% confidence interval for the difference $(b_2 - b_1)$.

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 $(\bar{b}_2 - \bar{b}_1)$ by its estimated standard error:

$$\frac{\bar{Y}_2 - \bar{Y}_1}{\sqrt{s_2^2/N_2 + s_1^2/N_1}}$$

is 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.

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. Compare this to the variability within the pollsters, summarized by the standard deviation.

Introduction to Data Science - 18 Association tests

Rafael A. Irizarry

The statistical models studied up to now are appropriate for continuous outcomes. We have not yet discussed inference for binary, categorical, and ordinal data. To give a very specific example, we will consider a case study examining funding success rates in the Netherlands, categorized by gender.

18.1 Case study: Funding success rates

A 2014 PNAS paper¹ 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 supporting this conclusion is based on 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)
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
```

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.

18.2 Lady Tasting Tea

R.A. Fisher² 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 and, consequently, 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 identified 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 for all 4. 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 being picked. We can then use combinations to determine each probability. The probability of picking 3 is $\binom{4}{3} / \binom{8}{4}$. The probability of picking all 4 correct is $\binom{4}{4} / \binom{8}{4}$. Thus, the chance of observing a 3 or something more extreme, under the null hypothesis, is $\text{approx } 0.24$. This is the p-value. The procedure that produced this p-value is called *Fisher’s exact test* and it uses the *hypergeometric distribution*.

18.3 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 can result from a pair of binary variables, they display 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

```

18.4 Chi-square Test

Notice that, in a sense, 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 ensured there were four cups with milk poured before tea and four cups with milk poured after, and the lady knew this. Therefore, the answers would also have to include four before and four afters. In this 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 allows 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 a total of 290, 1,345, 177, 1,011 applicants— some are men and some are women, and some get funded while others do not. We saw that the success rates for men and women respectively 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
```

Would we see this again if we randomly assign funding at the overall rate:

```
rate <- with(totals, (yes_men + yes_women))/sum(totals)
rate
#> [1] 0.165
```

The Chi-square test answers this question. The first step is to create the two-by-two data table:

```
o <- with(totals, data.frame(men = c(no_men, yes_men),
                               women = c(no_women, yes_women),
                               row.names = c("no", "yes")))
o
#>   men women
#> no 1345 1011
#> 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:

```
e <- with(totals, data.frame(men = (no_men + yes_men) * c(1 - rate, rate),
                               women = (no_women + yes_women) * c(1 - rate, rate),
                               row.names = c("no", "yes")))
e
#>   men women
#> no 1365 991
#> 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 statistic quantifies how much the observed tables deviates from the expected by:

1. Taking the difference between each observed and expected cell value.
2. Squaring this difference.
3. Dividing each squared difference by the expected value.
4. Summing all these values together to get the final statistic.

```
sum((o - e)^2/e)
#> [1] 4.01
```

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 <- chisq.test(o, correct = FALSE)
```

We see that the p-value is 0.045:

```
chisq_test$p.value
#> [1] 0.0451
```

By default, the `chisq.test` function applies a *continuity correction*. This correction is particularly useful when a cell in the table has values close to 0, as it prevents low observed values from inflating the statistics. This achieved by subtracting 0.5 in the following way:

```
sum((abs(o - e) - 0.5)^2/e)
#> [1] 3.81
```

Note that it matches the default behavior:

```
chisq.test(o)$statistic
#> X-squared
#>      3.81
```

18.5 Generalized linear models

We presented a way to perform hypothesis testing for determining if there is association between two binary outcomes. But we have not yet described how to quantify effects. Can we estimate the effect of being a woman in funding success in the Netherlands? Note that if our outcomes are binary, then the linear models presented in the Chapter 17 are not appropriate because the (β) s and (ε) are continuous. However, an adaptation of these methods, that is widely used in, for example, medical studies, gives us a way to estimate effects along with their standard errors.

The idea is to model a transformation of the expected value of the outcomes with a linear model. The transformation is selected so that any continuous value is possible. The mathematical equation for a model with one variable looks like this:

$$[g(\mathbb{E}(Y_i)) = \beta_0 + \beta_1 x_i]$$

To finish describing the model, we impose a distribution on (Y) , such as binomial or Poisson. These are referred to as *generalized linear models*.

We illustrate this with the funding rates example. We define (Y_i) to be 1 if person (i) received funding and 0 otherwise, and (x_i) to be 1 for person (i) is a woman and 0 if they are a man. For this data, the expected value of (Y_i) is the probability of funding for person (i) $(\Pr(Y_i=1))$. We assume the outcomes (Y_i) are binomial, with $(N=1)$ and probability (p_i) . For binomial data, the most widely used transformation is the logit function $(g(p) = \log(p / (1-p)))$, which takes numbers between 0 and 1 to any continuous number. The model looks like this:

$$[\log \frac{\Pr(Y_i=1)}{1-\Pr(Y_i=1)} = \beta_0 + \beta_1 x_i]$$

18.5.1 The odds ratio

To understand how (β_1) can be used to quantify the effect of being a woman on success rates, first note that $(\Pr(Y_i=1) / (1-\Pr(Y_i=1)) = \Pr(Y_i=1) / \Pr(Y_i=0))$ is the *odds* of person (i) getting funding: the ratio of the probability of success and probability of failure. This implies that (e^{β_0}) is the odds for men and $(e^{\beta_0}e^{\beta_1})$ is the odds for women, which implies (e^{β_1}) is the odds for women divided by the odds for men. This quantity is called the *odds ratio*. To see this, note that if use (p_1) and (p_0) to denote the probability of success for women and men, respectively, then (e^{β_1}) can be rewritten as:

$$[e^{\beta_1} = \frac{p_1}{p_0} = \frac{1-p_0}{p_0}]$$

(β_1) therefore quantifies the *log odds ratio*.

Now how do we estimate these parameters? Although the details are not described in this book, least squares is no longer an optimal way of estimating the parameters and instead we use an approach called *maximum likelihood estimation* (MLE). More advanced mathematical derivations show that a version of the central limit theorem applies, and the estimates obtained this way are approximately normal when the number of observations is large. The theory also provides a way to calculate standard errors for the estimates of the (β) s.

18.5.2 Fitting the model

To obtain the maximum likelihood estimates using R, we can use the `glm` function with the `family` argument set to `binomial`. This defaults to using the logit transformation. Note that we do not have the individual level data, but because our model assumes the probability of success is the same for all women and all men, then the number of success can be modeled as binomial with $(N=1)$ trials and probability (p_1) for women and binomial with

$\backslash(N_0\backslash)$ trials and probability $\backslash(p_0\backslash)$ for men, where $\backslash(N_1\backslash)$ and $\backslash(N_0\backslash)$ are the total number of women and men. In this case, the `glm` function is used as follows:

```
success <- with(totals, c(yes_men, yes_women))
failure <- with(totals, c(no_men, no_women))
gender <- factor(c("men", "women"))
fit <- glm(cbind(success, failure) ~ gender, family = "binomial")
coefficients(summary(fit))
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -1.534     0.0647 -23.7 3.83e-124
#> genderwomen -0.208     0.1041   -2.0  4.54e-02
```

The estimate of the odds ratio is 0.811982, interpreted as the odds being lowered by 20% for women compared to men. But is this due to chance? We already noted that the p-value is about 0.05, but the GLM approach also permits us to compute confidence intervals using the `confint` function. To show the interval for the more interpretable odds ratio, we simply exponentiate:

```
exp(confint(fit, 2))
#> 2.5 % 97.5 %
#> 0.661 0.995
```

We have used a simple version of GLMs in which the only variable is binary. However, the method can be expanded to incorporate multiple variables, including continuous ones. In these contexts, the log odds ratio interpretation becomes more complex. Also, note that we have shown just one version of GLM appropriate for binomial data using a logit transformation. This version is often referred to as *logistic regression*. Nevertheless, GLM can be used with other transformation and distributions. You can learn more by consulting a GLM textbook.

18.5.3 Simple standard error approximation for two-by-two table odds ratio

Using `glm`, we can obtain estimates, standard errors, and confidence intervals for a wide range of models. To do this, we use a rather complex algorithms. In the case of two-by-two tables. we can obtain a standard error for the log odds ratio using a simple approximation.

FIX SEE WHAT FOLLOWS If our two-by-two tables have the following entries:

	Men	Women
Awarded	a	b
Not Awarded	c	d

In this case, the odds ratio is simply $\backslash(\frac{a}{c}\{\frac{b}{d}\} = \frac{ad}{bc}\backslash)$. We can confirm that we obtain the same estimate as when using `glm`:

```
two_by_two <- with(totals, data.frame(awarded = c("no", "yes"),
                                         men = c(no_men, yes_men),
                                         women = c(no_women, yes_women)))

or <- with(two_by_two, women[2]/sum(women) / (women[1]/sum(women)) / ((men[2]/sum(men)) / (men[1]/sum(men)))
c(log(or), fit$coef[2])
#>             genderwomen
#> -0.208      -0.208
```

Statistical theory tells us that when all four entries of the two-by-two table are large enough, the log odds ratio is approximately normal with standard error:

$$\sqrt{\frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}}$$

This implies that a 95% confidence interval for the log odds ratio can be formed by:

$$\left[\log \left(\frac{ad}{bc} \right) \pm 1.96 \sqrt{\frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}} \right]$$

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:

```
se <- two_by_two |> select(-awarded) |>
  summarize(se = sqrt(sum(1/men) + sum(1/women))) |>
  pull(se)
exp(log(or) + c(-1,1) * qnorm(0.975) * se)
#> [1] 0.662 0.996
```

Note that 1 is not included in the confidence interval, implying that the p-value is smaller than 0.05. We can confirm this using:

```
2*(1 - pnorm(abs(log(or)), 0, se))
#> [1] 0.0454
```

Keep in mind that the p-values obtained with `chisq.test`, `glm` and this simple approximation are all slightly different. This is because these are both based on different approximation approaches.

18.6 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 by looking closely at the results, we realize that the 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. Therefore, a very small p-value does not necessarily mean a very large odds ratio. Observe 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_x_10 <- two_by_two |>
  select(-awarded) |>
  mutate(men = men*10, women = women*10)
chisq.test(two_by_two_x_10)$p.value
#> [1] 2.63e-10
```

Also, 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 $\log(ad/bc)$ 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.

18.7 Exercises

1. A famous athlete boasts an impressive career, winning 70% of her 500 career matches. Nevertheless, this athlete is 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 based on t-statistics 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.
6. FIX Use the `research_funding_rates` data to estimate the log odds ratio along and standard errors comparing women to men for each discipline. Compute a confidence interval and report all the disciplines for which one gender appears to be favored over the other.
7. Divide the log odds ratio estimates by their respective standard errors and generate a qqplot comparing these to a standard normal. Do any of the disciplines deviate from what is expected by chance?
8. During the 2016 US presidential election, then candidate Donald J. Trump used his twitter account as a way to communicate with potential voters. Todd Vaziri hypothesized that “Every non-hyperbolic tweet is from iPhone (his staff). Every hyperbolic tweet is from Android (from him).” We will test this hypothesis using association tests. The `dslabs` object `sentiment_counts` provides a table with the counts for several sentiments from each source (Android or iPhone):

```
library(tidyverse)
library(dslabs)
sentiment_counts
```

Compute an odds ratio comparing Android to iPhone for each sentiment and add it to the table.

9. Compute a 95% confidence interval for each odds ratio.
 10. Generate a plot showing the estimated odds ratios along with their confidence intervals.
 11. FIX Test the null hypothesis that there is no difference between tweets from Android and iPhone and report the sentiments with p-values less than 0.05 and more likely to come from Android.
 12. For each sentiment, find the words assigned to that sentiment, keep words that appear at least 25 times, compute the odd ratio for each, and show a barplot for those with odds ratio larger than 2 or smaller than 1/2.
-

1. <http://www.pnas.org/content/112/40/12349.abstract>
2. https://en.wikipedia.org/wiki/Ronald_Fisher

Introduction to Data Science - 19 Association is not causation

Rafael A. Irizarry

Association is not causation is perhaps the most important lesson one can learn in a statistics class. *Correlation is not causation* is another way to say this. Throughout the statistics part of the book, we have described tools useful for quantifying associations between variables. However, we must be careful not to over-interpret these associations.

There are many reasons that a variable $\langle X \rangle$ can be correlated with a variable $\langle Y \rangle$, without having any direct effect on $\langle Y \rangle$. Below we examine four common ways that can lead to misinterpreting data.

19.1 Spurious correlation

The following comical example underscores the concept that correlation is not causation. It shows a very strong correlation between divorce rates and margarine consumption.

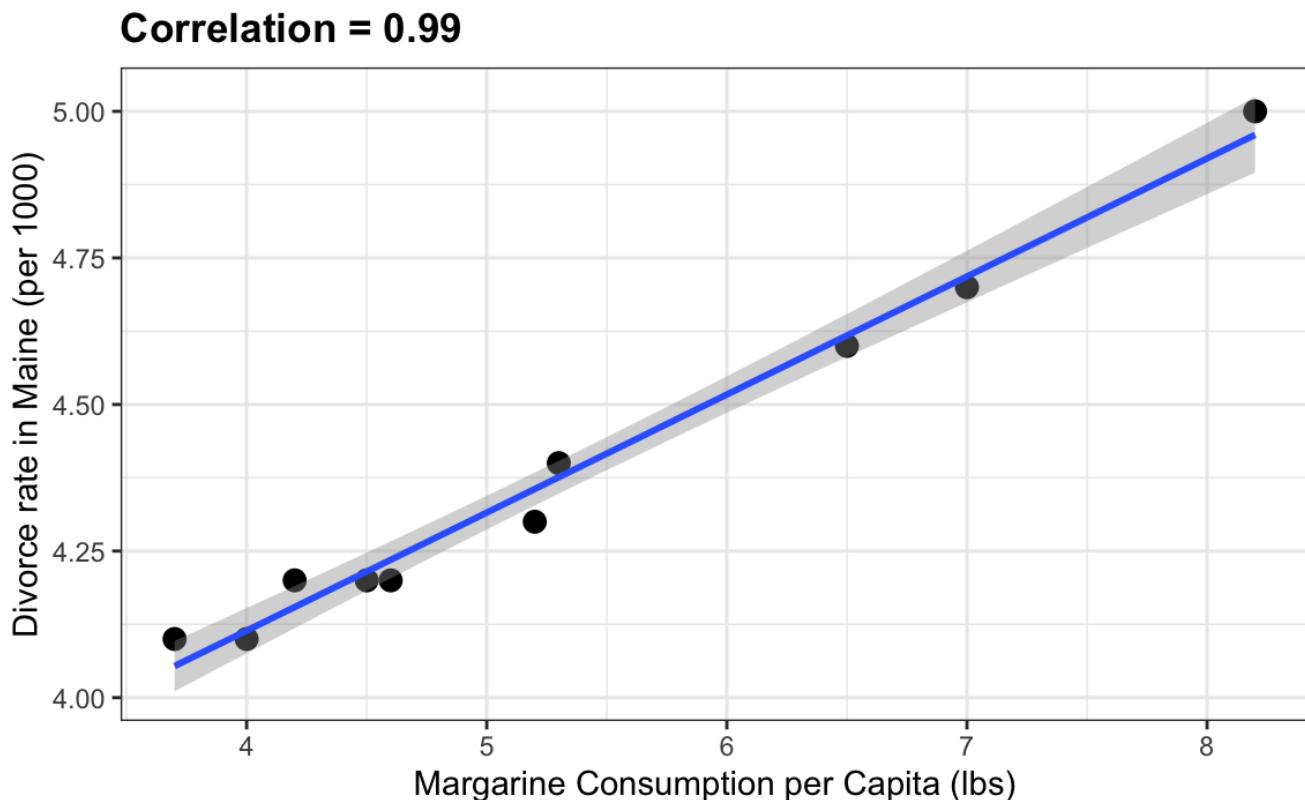


Figure 1:

Does this mean that margarine causes divorces? Or do divorces cause people to eat more margarine? Of course. the answer to both these questions is no. This is just an example of what we call a *spurious correlation*.

You can see many more absurd examples on the Spurious Correlations website¹.

The cases presented on the website are all instances of what is generally called *data dredging*, *data fishing*, or *data snooping*. It's basically a form of what in the US they call *cherry picking*. An example of data dredging would be if you look through many results produced by a random process and pick the one that shows a relationship that supports a theory you want to defend.

A Monte Carlo simulation can be used to show how data dredging can result in finding high correlations among uncorrelated variables. We will save the results of our simulation into a tibble:

```
library(tidyverse)
N <- 25
g <- 1000000
sim_data <- tibble(group = rep(1:g, each = N),
                    x = rnorm(N*g),
                    y = rnorm(N*g))
```

The first column denotes group. We created groups. For each group, we generated a pair of independent vectors, $\langle X \rangle$ and $\langle Y \rangle$, with 25 observations each, stored in the second and third columns. Because we constructed the simulation, we know that $\langle X \rangle$ and $\langle Y \rangle$ are not correlated.

Next, we compute the correlation between X and Y for each group and look at the max:

```
res <- sim_data |>
  group_by(group) |>
  summarize(r = cor(x, y)) |>
  arrange(desc(r))
res
#> # A tibble: 1,000,000 × 2
#>   group     r
#>   <int> <dbl>
#> 1 180152 0.796
#> 2 902354 0.790
#> 3 410920 0.785
#> 4 796016 0.760
#> 5 542451 0.760
#> # 999,995 more rows
```

We see a maximum correlation of 0.7957485. If you just plot the data from the group achieving this correlation, it shows a convincing plot that $\langle X \rangle$ and $\langle Y \rangle$ are in fact correlated:

```
sim_data |> filter(group == res$group[which.max(res$r)]) |>
  ggplot(aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm")
#> `geom_smooth()` using formula = 'y ~ x'
```

Remember that the correlation summary is a random variable. Here is the distribution generated by the Monte Carlo simulation:

```
res |> ggplot(aes(x=r)) + geom_histogram(binwidth = 0.1, color = "black")
```

It's simply a mathematical fact that if we observe random correlations that are expected to be 0, but have a standard error of 0.2039625, the largest one will be close to 1.

If we performed regression on this group and interpreted the p-value, we would incorrectly claim this was a statistically significant relation:

```
library(broom)
sim_data |>
  filter(group == res$group[which.max(res$r)]) |>
  summarize(tidy(lm(y ~ x))) |>
  filter(term == "x")
#> # A tibble: 1 × 5
```

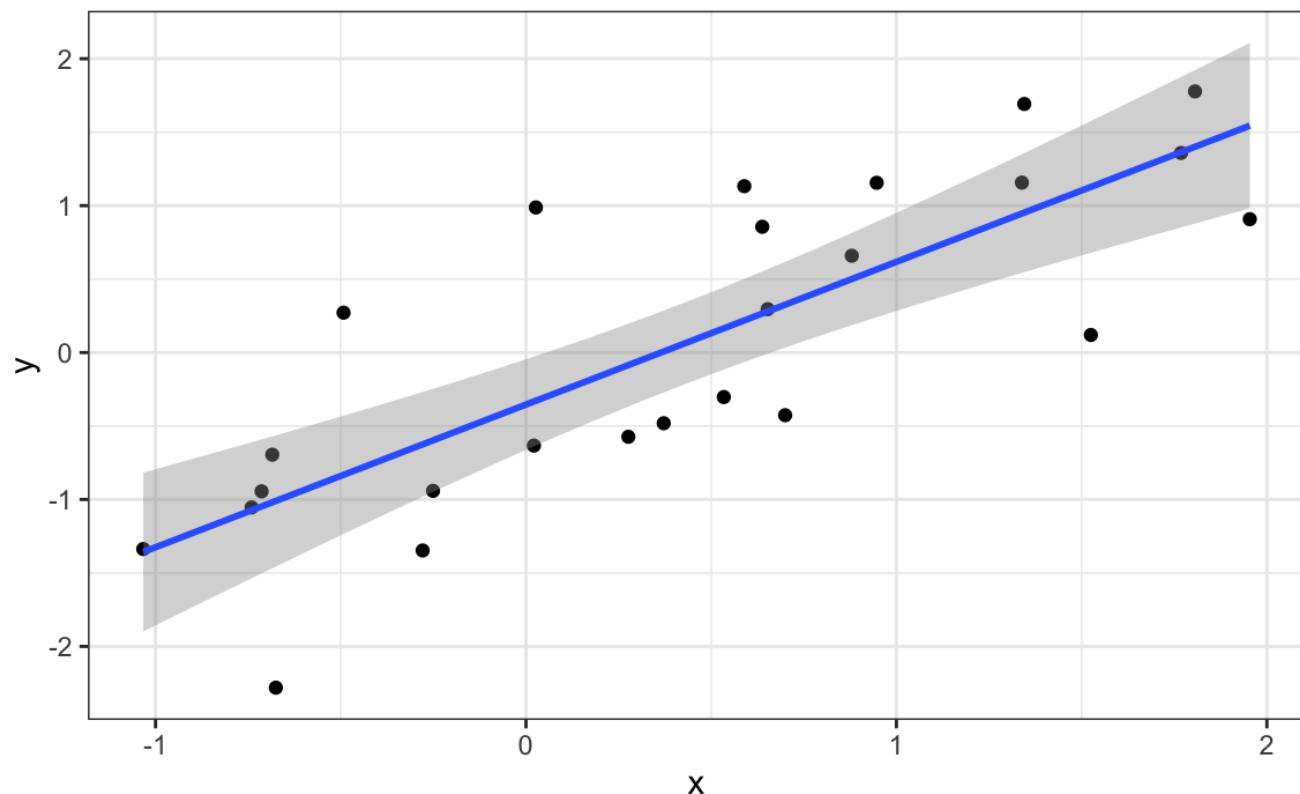


Figure 2:

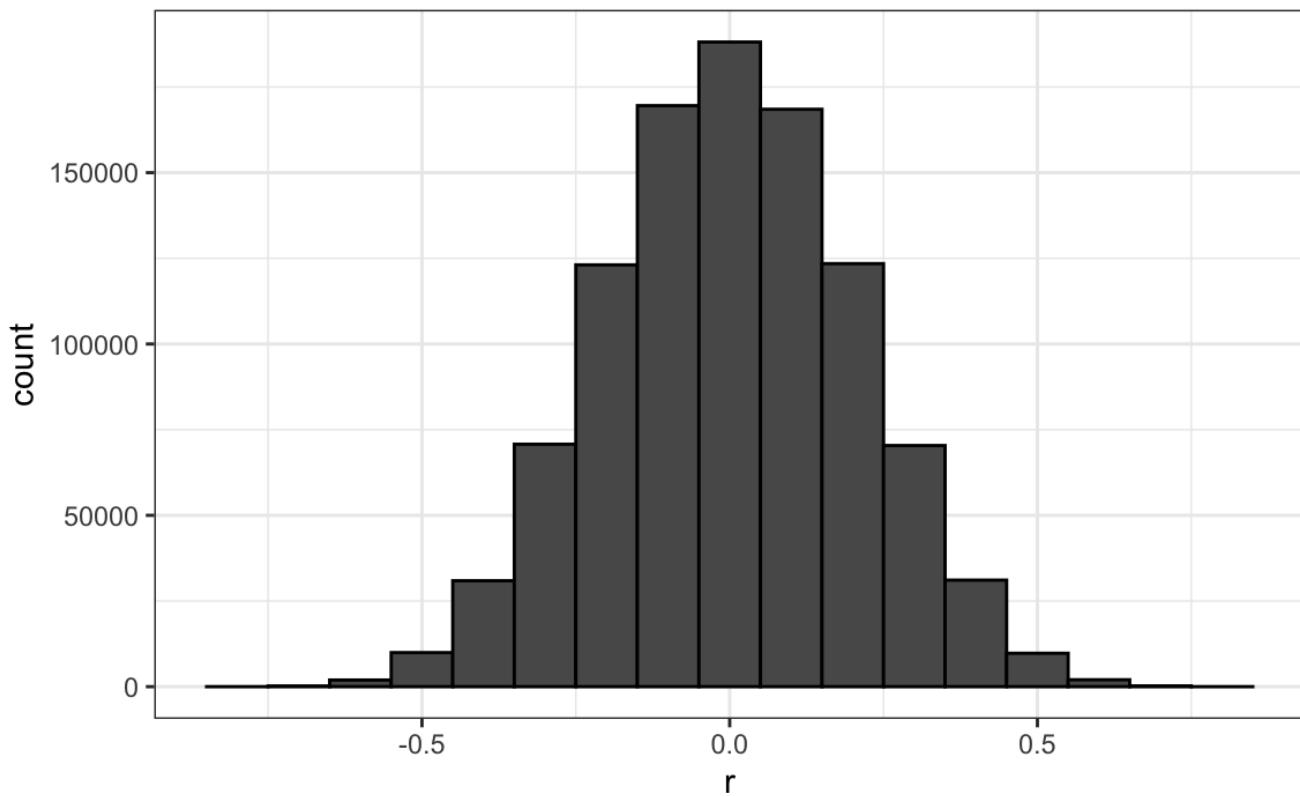


Figure 3:

```
#>   term estimate std.error statistic    p.value
#>   <chr>    <dbl>     <dbl>      <dbl>      <dbl>
#> 1 x        0.971    0.154     6.30  0.00000198
```

This particular form of data dredging is referred to as *p-hacking*. P-hacking is a topic of much discussion because it poses a problem in scientific publications. Since publishers tend to reward statistically significant results over negative results, there is an incentive to report significant results. In epidemiology and the social sciences, for example, researchers may look for associations between an adverse outcome and several exposures, and report only the one exposure that resulted in a small p-value. Furthermore, they might try fitting several different models to account for confounding and choose the one that yields the smallest p-value. In experimental disciplines, an experiment might be repeated more than once, yet only the results of the one experiment with a small p-value reported. This does not necessarily happen due to unethical behavior, but rather as a result of statistical ignorance or wishful thinking. In advanced statistics courses, you can learn methods to adjust for these multiple comparisons.

19.2 Outliers

Suppose we take measurements from two independent outcomes, $\langle X \rangle$ and $\langle Y \rangle$, and we standardize the measurements. However, imagine we make a mistake and forget to standardize entry 23. We can simulate such data using:

```
set.seed(1985)
x <- rnorm(100, 100, 1)
y <- rnorm(100, 84, 1)
x[-23] <- scale(x[-23])
y[-23] <- scale(y[-23])
```

The data look like this:

```
qplot(x, y)
#> Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```

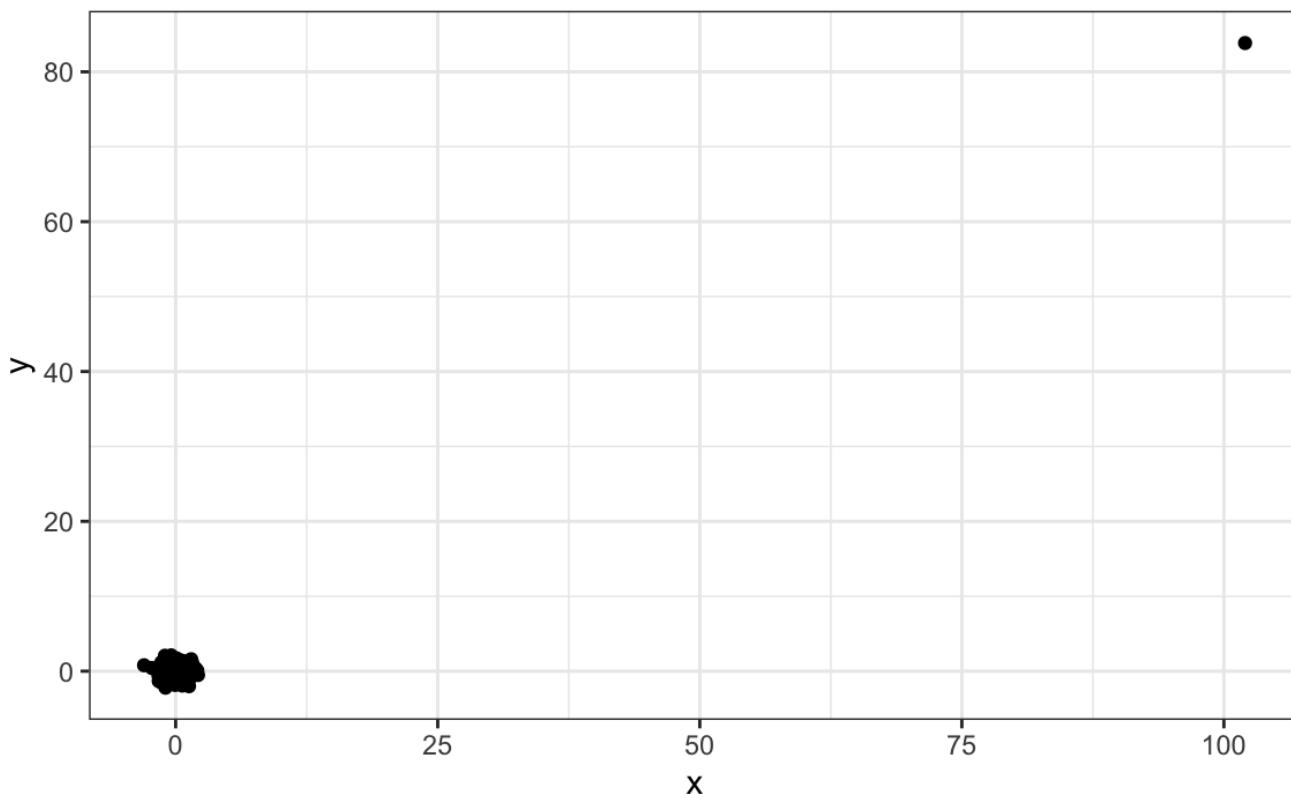


Figure 4:

Not surprisingly, the correlation is very high:

```
cor(x,y)
#> [1] 0.988
```

But this is driven by the one outlier. If we remove this outlier, the correlation is greatly reduced to almost 0, which is what it should be:

```
cor(x[-23], y[-23])
#> [1] -0.0442
```

There is an alternative to the sample correlation for estimating the population correlation that is robust to outliers. It is called *Spearman correlation*. The idea is simple: compute the correlation on the ranks of the values. Here is a plot of the ranks plotted against each other:

```
qplot(rank(x), rank(y))
```

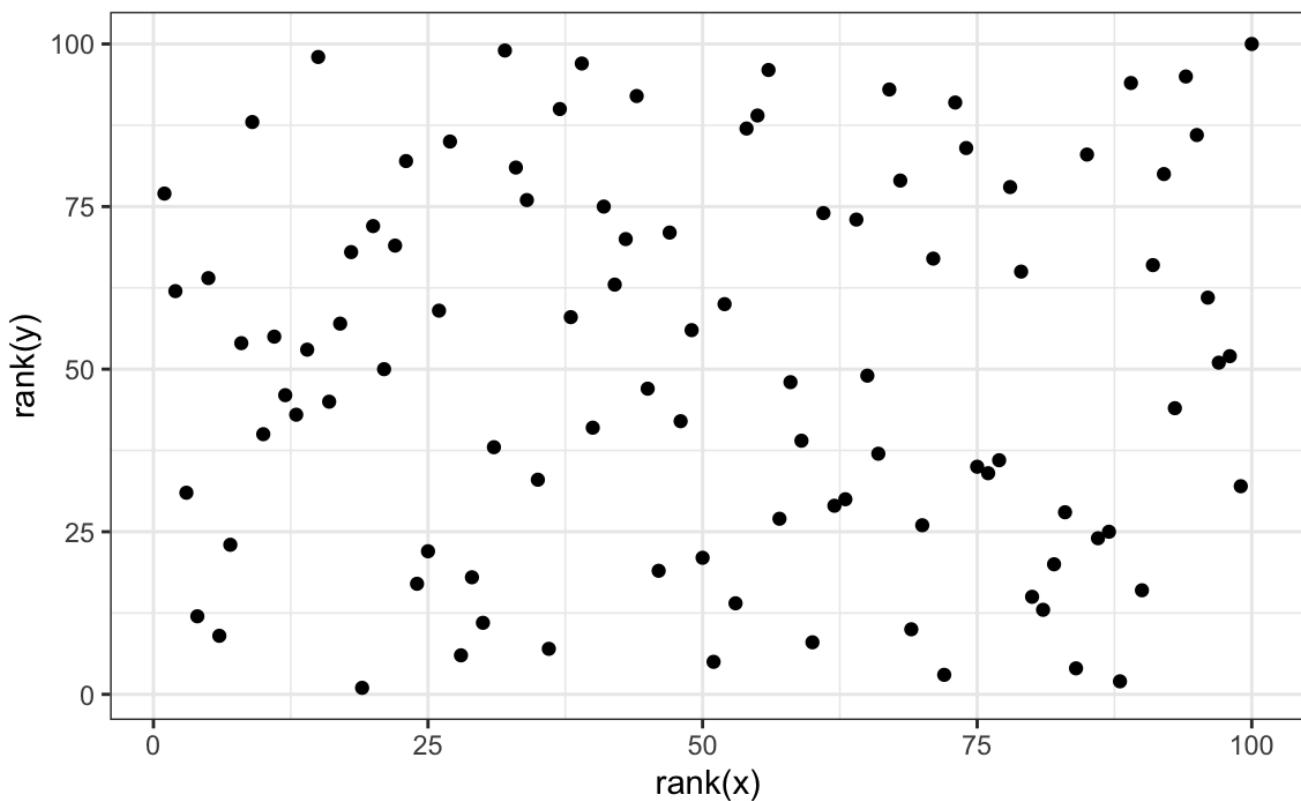


Figure 5:

The outlier is no longer associated with a very large value, and the correlation decreases significantly:

```
cor(rank(x), rank(y))
#> [1] 0.00251
```

Spearman correlation can also be calculated like this:

```
cor(x, y, method = "spearman")
#> [1] 0.00251
```

There are also methods for robust fitting of linear models which you can learn about in, for instance, this book: Robust Statistics: Edition 2 by Peter J. Huber & Elvezio M. Ronchetti.

19.3 Reversing cause and effect

Another way association is confused with causation is when the cause and effect are reversed. An example of this is claiming that tutoring makes students perform worse because they test lower than peers that are not tutored. In this case, the tutoring is not causing the low test scores, but the other way around.

A form of this claim actually made it into an op-ed in the New York Times titled Parental Involvement Is Overrated². Consider this quote from the article:

When we examined whether regular help with homework had a positive impact on children's academic performance, we were quite startled by what we found. Regardless of a family's social class, racial or ethnic background, or a child's grade level, consistent homework help almost never improved test scores or grades... Even more surprising to us was that when parents regularly helped with homework, kids usually performed worse.

A very likely possibility is that the children needing regular parental help, receive this help because they don't perform well in school.

We can easily construct an example of cause and effect reversal using the father and son height data. If we fit the model:

```
\[X_i = \beta_0 + \beta_1 y_i + \varepsilon_i, i=1, \dots, N\]
```

to the father and son height data, where (X_i) is the father height and (y_i) is the son height, we do get a statistically significant result. We use the `galton_heights` dataset defined in Chapter 14:

```
galton_heights |> summarize(tidy(lm(father ~ son)))
#> Warning: Returning more (or less) than 1 row per `summarise()` group was
#> deprecated in dplyr 1.1.0.
#> Please use `reframe()` instead.
#> When switching from `summarise()` to `reframe()`, remember that
#> `reframe()` always returns an ungrouped data frame and adjust
#> accordingly.
#> # A tibble: 2 × 5
#>   term      estimate std.error statistic p.value
#>   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept) 40.9      4.40      9.29 5.47e-17
#> 2 son         0.407     0.0636     6.40 1.36e- 9
```

The model fits the data very well. If we examine the mathematical formulation of the model above, it could easily be misinterpreted so as to suggest that the son being tall caused the father to be tall. However, based on our understanding of genetics and biology, we know it's the other way around. The model is technically correct. The estimates and p-values were obtained correctly as well. What is wrong here is the interpretation.

19.4 Confounders

Confounders are perhaps the most common reason that leads to associations begin misinterpreted.

If (X) and (Y) are correlated, we call (Z) a *confounder* if changes in (Z) cause changes in both (X) and (Y) . Earlier, when studying baseball data, we saw how Home Runs were a confounder that resulted in a higher correlation than expected when studying the relationship between Bases on Balls and Runs. In some cases, we can use linear models to account for confounders. However, this is not always the case.

Incorrect interpretation due to confounders is ubiquitous in the lay press and they are often hard to detect. Here, we present a widely used example related to college admissions.

19.4.1 Example: UC Berkeley admissions

Admission data from six U.C. Berkeley majors, from 1973, showed that more men were being admitted than women: 44% men were admitted compared to 30% women. PJ Bickel, EA Hammel, and JW O'Connell. Science (1975). We can load the data and compute a statistical test, which clearly rejects the hypothesis that gender and admission are independent:

```

two_by_two <- admissions |> group_by(gender) |>
  summarize(total_admitted = round(sum(admitted / 100 * applicants)),
            not_admitted = sum(applicants) - sum(total_admitted)) |>
  select(-gender)

chisq.test(two_by_two)$p.value
#> [1] 1.06e-21

```

But closer inspection shows a paradoxical result. Here are the percent admissions by major:

```

admissions |> select(major, gender, admitted) |>
  pivot_wider(names_from = "gender", values_from = "admitted") |>
  mutate(women_minus_men = women - men)
#> # A tibble: 6 × 4
#>   major    men   women  women_minus_men
#>   <chr> <dbl> <dbl>          <dbl>
#> 1 A        62    82            20
#> 2 B        63    68             5
#> 3 C        37    34           -3
#> 4 D        33    35             2
#> 5 E        28    24            -4
#> # 1 more row

```

Four out of the six majors favor women. More importantly, all the differences are much smaller than the 14.2 difference that we see when examining the totals.

The paradox is that analyzing the totals suggests a dependence between admission and gender, but when the data is grouped by major, this dependence seems to disappear. What's going on? This actually can happen if an uncounted confounder is driving most of the variability.

So let's define three variables: $\chi(X)$ is 1 for men and 0 for women, $\chi(Y)$ is 1 for those admitted and 0 otherwise, and $\chi(Z)$ quantifies the selectivity of the major. A gender bias claim would be based on the fact that $\Pr(Y=1 | X=x)$ is higher for $x=1$ than for $x=0$. However, $\chi(Z)$ is an important confounder to consider. Clearly, $\chi(Z)$ is associated with $\chi(Y)$, as the more selective a major, the lower $\Pr(Y=1 | Z=z)$. But is major selectivity $\chi(Z)$ associated with gender $\chi(X)$?

One way to see this is to plot the total percent admitted to a major versus the percent of women that made up the applicants:

```

admissions |>
  group_by(major) |>
  summarize(major_selectivity = sum(admitted * applicants)/sum(applicants),
            percent_women_applicants = sum(applicants * (gender=="women")) /
                                         sum(applicants) * 100) |>
  ggplot(aes(major_selectivity, percent_women_applicants, label = major)) +
  geom_text()

```

There seems to be association. The plot suggests that women were much more likely to apply to the two “hard” majors, indicating a confounding between gender and major’s selectivity. Compare, for example, major B and major E. Major E is much harder to enter than major B, and over 60% of applicants for major E were women, while less than 30% of the applicants for major B were women.

19.4.2 Confounding explained graphically

The following plot shows the number of applicants that were admitted and those that were not by major and gender. It also breaks down the acceptances by major. This breakdown allows us to see that the majority of accepted men came from two majors, A and B. It also reveals that few women applied to these majors.

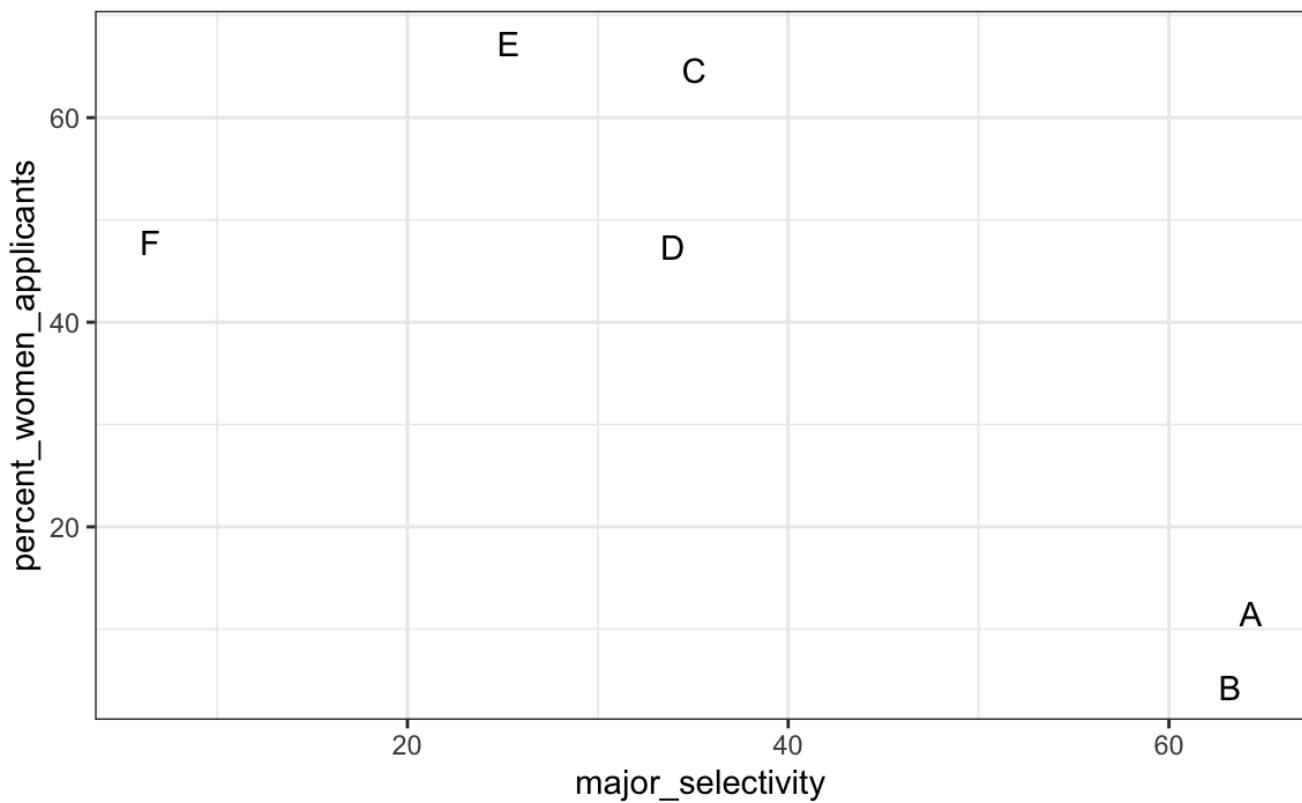


Figure 6:

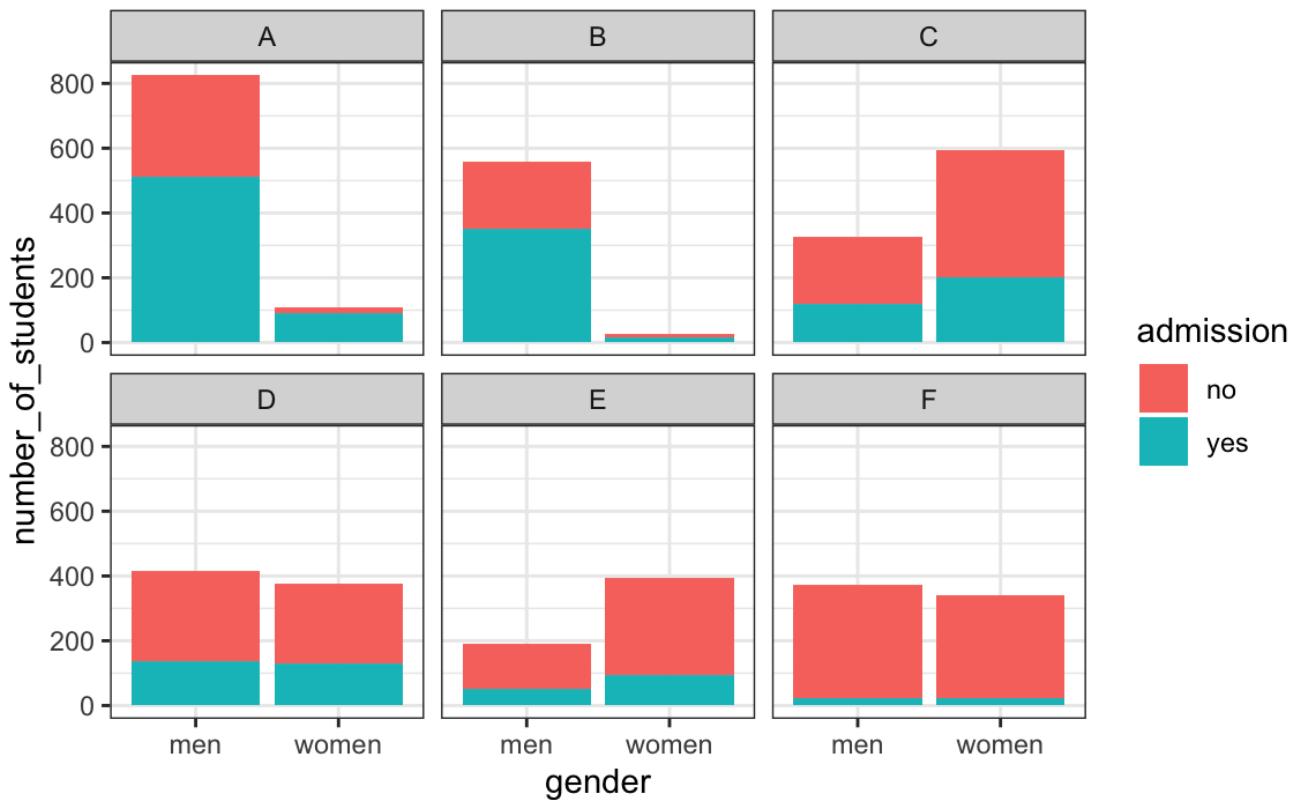


Figure 7:

19.4.3 Average after stratifying

In this plot, we can see that if we condition or stratify by major, and then look at differences, we control for the confounder and this effect goes away:

```
admissions |>
  ggplot(aes(major, admitted, col = gender, size = applicants)) +
  geom_point()
```

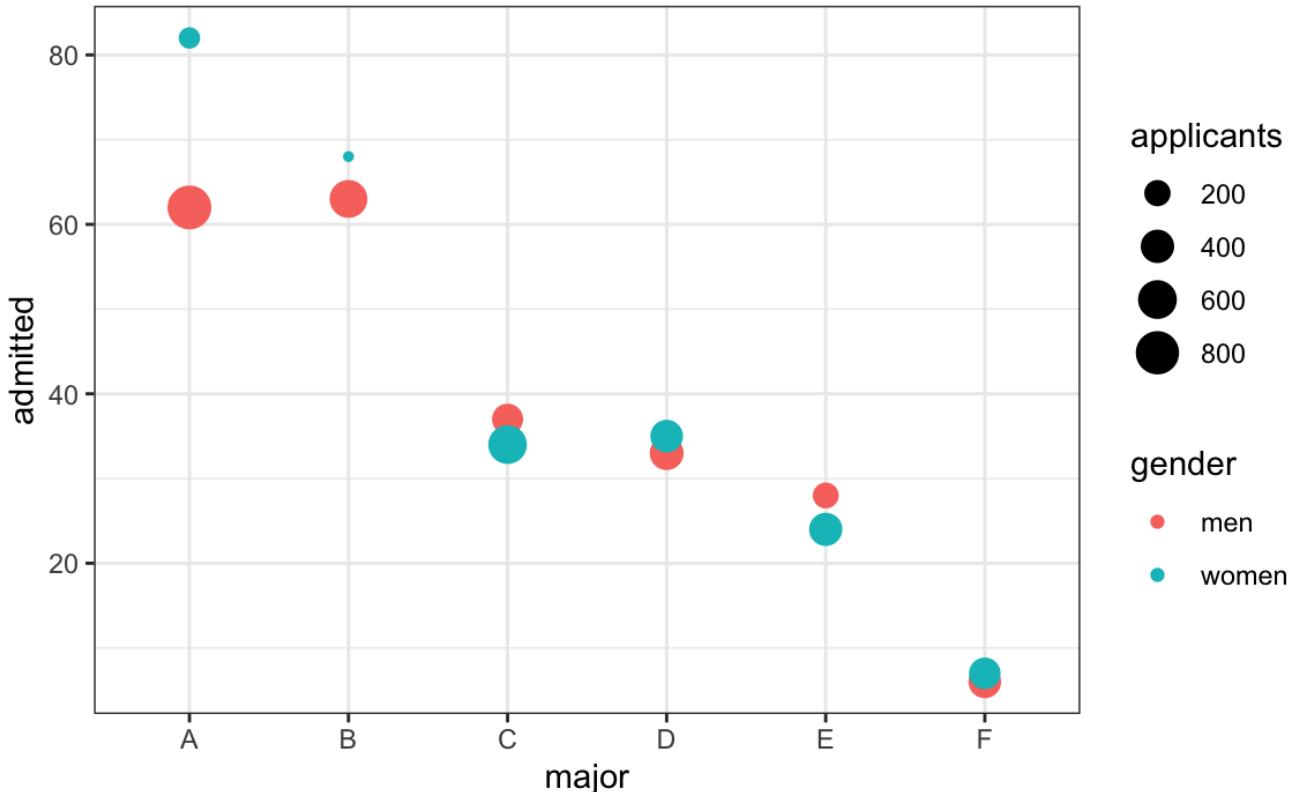


Figure 8:

Now we see that major by major, there is not much difference. The size of the dot represents the number of applicants, and explains the paradox: we see large red dots and small blue dots for the easiest majors, A and B.

If we average the difference by major, we find that the percent is actually 3.5% higher for women.

```
admissions |> group_by(gender) |> summarize(average = mean(admitted))
#> # A tibble: 2 × 2
#>   gender     average
#>   <chr>      <dbl>
#> 1 men        38.2
#> 2 women      41.7
```

19.5 Simpson's paradox

The case we have just covered is an example of Simpson's paradox. It is called a paradox because we see the sign of the correlation flip when comparing the entire publication to specific strata. As an illustrative example, suppose you have three random variables (X) , (Y) , and (Z) , and we observe realizations of these. Here is a plot of simulated observations for (X) and (Y) along with the sample correlation:

You can see that (X) and (Y) are negatively correlated. However, once we stratify by (Z) (shown in different colors below), another pattern emerges:

Correlation = -0.73

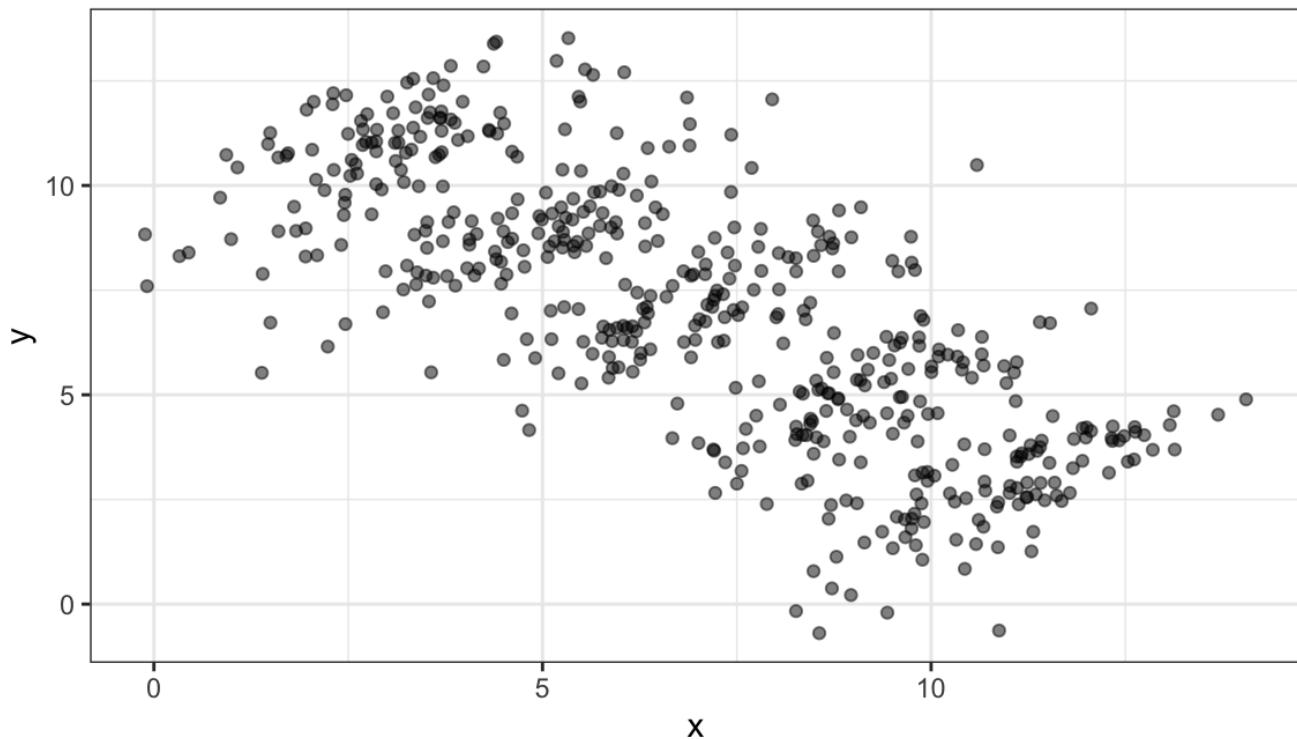


Figure 9:

Correlations = 0.74 0.65 0.76 0.72 0.71

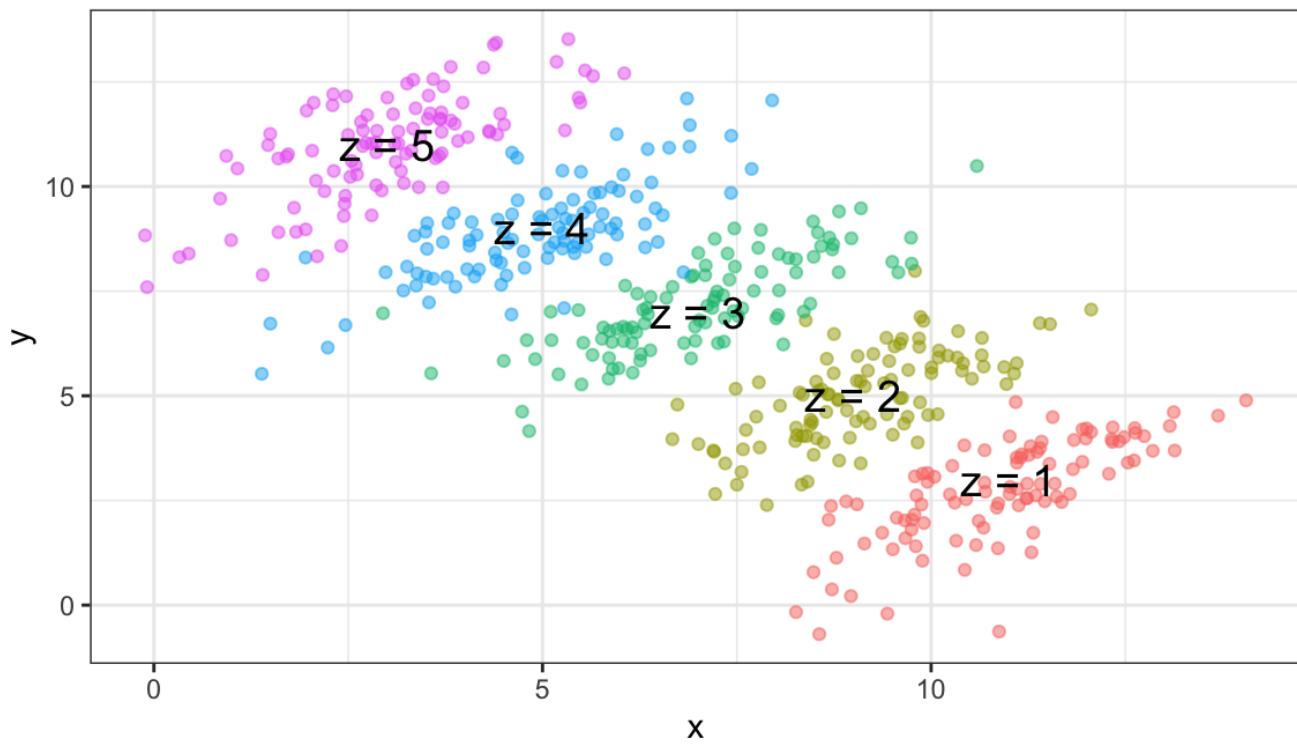


Figure 10:

It is really Z that is negatively correlated with X . If we stratify by Z , the X and Y are actually positively correlated, as seen in the plot above.

19.6 Exercises

For the next set of exercises, we examine the data from a 2014 PNAS paper³ that analyzed success rates from funding agencies in the Netherlands and concluded:

Our 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 used in instructional and evaluation materials.

A response⁴ was published a few months later titled *No evidence that gender contributes to personal research funding success in The Netherlands: A reaction to Van der Lee and Ellemers* which concluded:

However, the overall gender effect borders on statistical significance, despite the large sample. Moreover, their conclusion could be a prime example of Simpson’s paradox; if a higher percentage of women apply for grants in more competitive scientific disciplines (i.e., with low application success rates for both men and women), then an analysis across all disciplines could incorrectly show “evidence” of gender inequality.

Who is correct here, the original paper or the response? Below, you will examine the data and come to your own conclusion.

1. The primary evidence for the conclusion of the original paper relies on a comparison of the percentages. Table S1 in the paper includes the information we need:

```
library(dslabs)
research_funding_rates
```

Construct the two-by-two table used for the conclusion about differences in awards by gender.

2. Compute the difference in percentage from the two-by-two table.
3. In the previous exercise, we noticed that the success rate is lower for women. But is it significant? Compute a p-value using a Chi-square test.
4. We see that the p-value is about 0.05. So there appears to be some evidence of an association. But can we infer causation here? Is gender bias causing this observed difference? The response to the original paper claims that what we see here is similar to the UC Berkeley admissions example. Specifically, they state that this “could be a prime example of Simpson’s paradox; if a higher percentage of women apply for grants in more competitive scientific disciplines, then an analysis across all disciplines could incorrectly show ‘evidence’ of gender inequality.” To settle this dispute, create a dataset with number of applications, awards, and success rate for each gender. Re-order the disciplines by their overall success rate. Hint: use the `reorder` function to re-order the disciplines in a first step, then use `pivot_longer`, `separate`, and `pivot_wider` to create the desired table.
5. To check if this is a case of Simpson’s paradox, plot the success rates versus disciplines, which have been ordered by overall success, with colors to denote the genders and size to denote the number of applications.
6. We definitely do not see the same level of confounding as in the UC Berkeley example. It is hard to say that there is a clear confounder here. However, we do see that, based on the observed rates, some fields favor men and others favor women. We also see that the two fields with the largest difference favoring men are also the fields with the most applications. But, unlike the UC Berkeley example, women are not more likely to apply for the harder subjects. Is it possible some of the selection committees are biased and others are not?

To answer this question we start by checking if any of the differences seen above are statistically significant. Remember that even when there is no bias, we will see differences due to random variability in the review process as well as random variability across candidates. Perform a Chi-square test for each discipline. Hint: define a function that receives the total of a two-by-two table and returns a data frame with the p-value. Use the 0.5 correction. Then use the `summarize` function.

7. In the medical sciences, there appears to be a statistically significant difference, but could this be a spurious correlation? We performed 9 tests. Reporting only the one case with a p-value less than 0.05 might be considered an example of cherry picking. Repeat the exercise above, but instead of a p-value, compute a log odds ratio divided by

their standard error. Then use qq-plot to see how much these log odds ratios deviate from the normal distribution we would expect: a standard normal distribution.

1. <http://tylervigen.com/spurious-correlations>
2. <https://opinionator.blogs.nytimes.com/2014/04/12/parental-involvement-is-overrated>
3. <http://www.pnas.org/content/112/40/12349.abstract>
4. <http://www.pnas.org/content/112/51/E7036.extract>

Introduction to Data Science - High dimensional data

Rafael A. Irizarry

There is a variety of computational techniques and statistical concepts that are useful for analysis of datasets for which each observation is associated with a large number of numerical variables. In this chapter, we provide a basic introduction to these techniques and concepts by describing matrix operations in R, dimension reduction, regularization, and matrix factorization. Handwritten digits data and movie recommendation systems serve as motivating examples.

A task that serves as motivation for this part of the book is quantifying the similarity between any two observations. For example, we might want to know how much two handwritten digits look like each other. However, note that each observation is associated with $(28 \times 28 = 784)$ pixels so we can't simply use subtraction as we would if our data was one dimensional. Instead, we will define observations as *points* in a *high-dimensional* space and mathematically define a *distance*. Many machine learning techniques, discussed in the next part of the book, require this calculation.

Additionally, this part of the book discusses dimension reduction. Here we search for data summaries that provide more manageable lower dimension versions of the data, but preserve most or all the *information* we need. We again use distance between observations as a specific example: we will summarize the data into lower dimensions, but in a way that preserves distance between any two observations. We use *linear algebra* as a mathematical foundation for all the techniques presented here.

Introduction to Data Science - 20 Matrices in R

Rafael A. Irizarry

When the number of variables associated with each observation is large and they can all be represented as a number, it is often more convenient to store them in a matrix and perform the analysis with linear algebra operations, rather than storing them in a data frame and performing the analysis with **tidyverse** or **data.table** functions. With matrices, variables for each observation are stored in a row, resulting in a matrix with as many columns as variables. In statistics, we refer to values represented in the rows of the matrix as the *covariates* or *predictors* and, in machine learning, we refer to them as the *features*.

In linear algebra, we have three types of objects: scalars, vectors, and matrices. We have already learned about vectors in R, and, although there is no data type for scalars, we can represent them as vectors of length 1. In this chapter, we learn how to work with matrices in R and relate them to linear algebra notation and concepts.

20.1 Case study: MNIST

The first step in handling mail received in the post office is to sort letters by zip code:



In the Machine Learning part of this book, we will describe how we can build computer algorithms to read handwritten digits, which robots then use to sort the letters. To do this, we first need to collect data, which in this case is a high-dimensional dataset and best stored in a matrix.

The MNIST dataset was generated by digitizing thousands of handwritten digits, already read and annotated by humans¹. Below are three images of written digits.

The images are converted into $(28 \times 28 = 784)$ pixels and, for each pixel, we obtain a grey scale intensity between 0 (white) and 255 (black). The following plot shows the individual features for each image:

For each digitized image, indexed by $\langle i \rangle$, we are provided with 784 variables and a categorical outcome, or *label*, representing the digit among $\langle 0, 1, 2, 3, 4, 5, 6, 7, 8, \rangle$ and $\langle 9 \rangle$ that the image is representing. Let's load the data using the **dslabs** package:

```
library(tidyverse)
library(dslabs)
mnist <- read_mnist()
```

In these cases, the pixel intensities are saved in a matrix:

```
class(mnist$train$images)
#> [1] "matrix" "array"
```

The labels associated with each image are included in a vector:

```
table(mnist$train$labels)
#>
#>    0    1    2    3    4    5    6    7    8    9
```

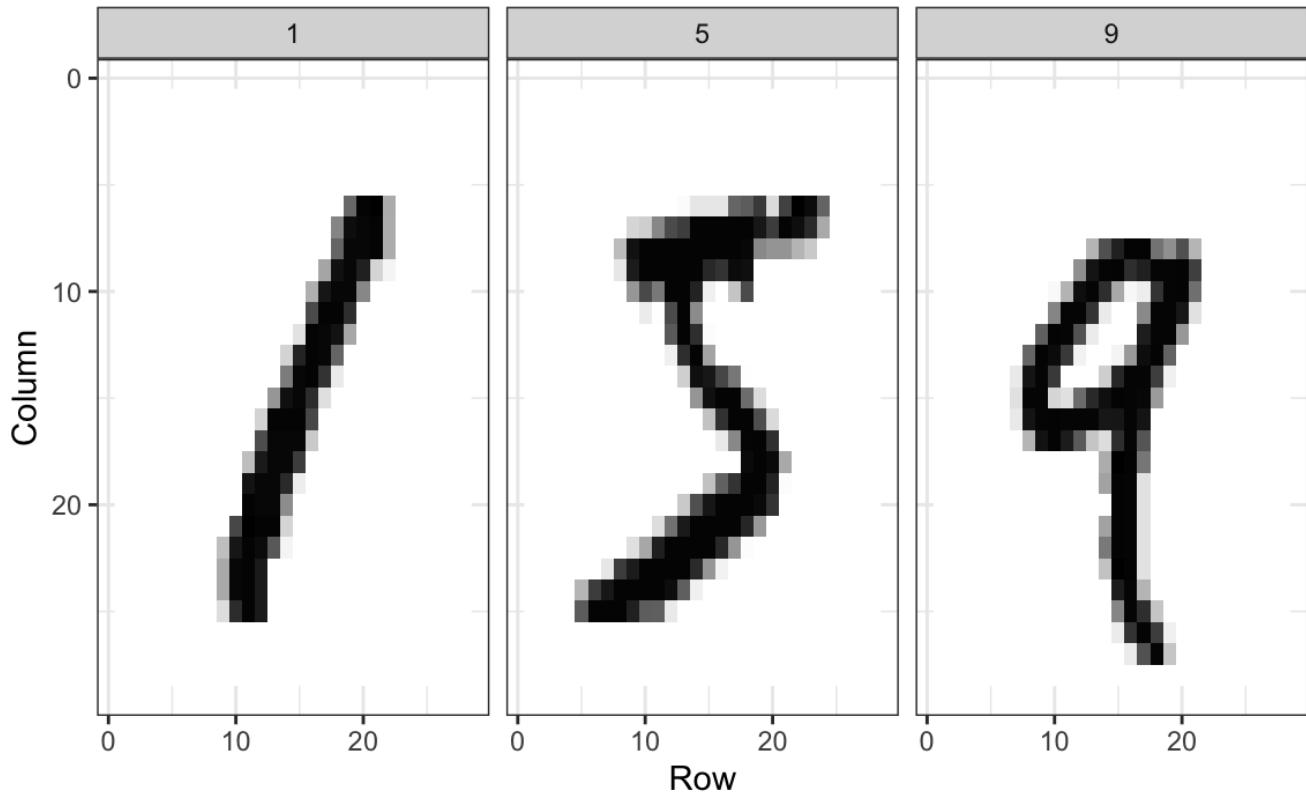


Figure 1:

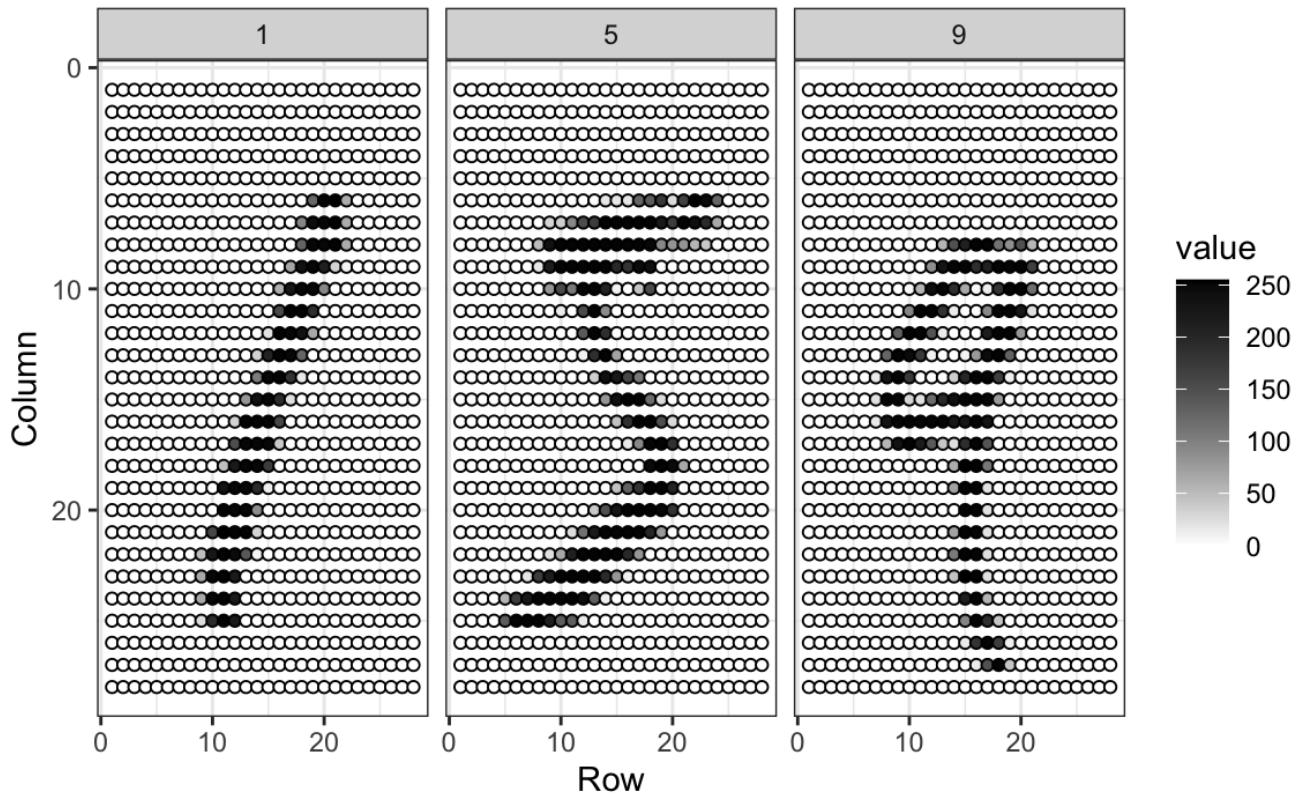


Figure 2:

```
#> 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949
```

20.1.1 Motivating tasks

To motivate the use of matrices in R, we will pose six tasks related to the handwritten digits data and then show the fast and simple code that solves them.

1. Visualize the original image. The pixel intensities are provided as rows in a matrix. We will show how to convert these to a matrix that we can visualize.
2. Do some digits require more ink to write than others? We will study the distribution of the total pixel darkness and how it varies by digits.
3. Are some pixels uninformative? We will study the variation of each pixel across digits and remove predictors (columns) associated with pixels that don't change much and thus can't provide much information for classification.
4. Can we remove smudges? We will first look at the distribution of all pixel values. Next, we will use this to pick a cutoff to define unwritten space. Then, we set anything below that cutoff to 0.
5. Binarize the data. First, we will look at the distribution of all pixel values. We will then use this to pick a cutoff to distinguish between writing and no writing. Afterward, we will convert all entries into either 1 or 0.
6. Standardize the digits. We will scale each of the predictors in each entry to have the same average and standard deviation.

To complete these, we will have to perform mathematical operations involving several variables. The **tidyverse** or **data.table** are not developed to perform these types of mathematical operations. For this task, it is convenient to use matrices.

To simplify the code below, we will rename these **x** and **y** respectively:

```
x <- mnist$train$images  
y <- mnist$train$labels
```

20.2 Dimensions of a matrix

The dimension of a matrix is an important characteristic needed to assure that certain linear algebra operations can be performed. The dimension is a two-number summary defined as the number of rows $\backslash(\backslash \times \backslash)$ the number of columns.

The **nrow** function tells us how many rows that matrix has:

```
nrow(x)  
#> [1] 60000
```

and **ncol** tells us how many columns:

```
ncol(x)  
#> [1] 784
```

We learn that our dataset contains 60,000 observations (images) and 784 features (pixels).

The **dim** function returns the rows and columns:

```
dim(x)  
#> [1] 60000 784
```

20.3 Creating a matrix

In R, we can create a matrix using the **matrix** function. The first argument is a vector containing the elements that will fill up the matrix. The second and third arguments determine the number of row and columns, respectively. So a typical way to create a matrix is to first obtain a vector of numbers containing the elements of the matrix and feeding it to the **matrix** function. For example, to create a $\backslash(100 \backslash \times 2\backslash)$ matrix of normally distributed random variables, we write:

```
z <- matrix(rnorm(100*2), 100, 2)
```

Note that by default the matrix is filled in column by column:

```
matrix(1:15, 3, 5)
#> [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    4    7   10   13
#> [2,]    2    5    8   11   14
#> [3,]    3    6    9   12   15
```

To fill the matrix row by row, we can use the `byrow` argument:

```
matrix(1:15, 3, 5, byrow = TRUE)
#> [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    2    3    4    5
#> [2,]    6    7    8    9   10
#> [3,]   11   12   13   14   15
```

The function `as.vector` converts a matrix back into a vector:

```
as.vector(matrix(1:15, 3, 5))
#> [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

If the product of columns and rows does not match the length of the vector provided in the first argument, `matrix` recycles values. If the length of the vector is a sub-multiple or multiple of the number of rows, this happens **without warning**:

```
matrix(1:3, 3, 5)
#> [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    1    1    1    1
#> [2,]    2    2    2    2    2
#> [3,]    3    3    3    3    3
```

20.4 Subsetting

To extract a specific entry from a matrix, for example the 300th row of the 100th column, we write:

```
x[300,100]
```

We can extract subsets of the matrices by using vectors of indexes. For example, we can extract the first 100 pixels from the first 300 observations like this:

```
x[1:300,1:100]
```

To extract an entire row or subset of rows, we leave the column dimension blank. So the following code returns all the pixels for the first 300 observations:

```
x[,1:300,]
```

Similarly, we can subset any number of columns by keeping the first dimension blank. Here is the code to extract the first 100 pixels:

```
x[,1:100]
```

If we subset just one row or just one column, the resulting object is no longer a matrix. For example:

```
dim(x[300,])
#> NULL
```

To avoid this, we can use the `drop` argument:

```
dim(x[100,,drop = FALSE])
#> [1] 1 784
```

Task 1: Visualize the original image

For instance, let's try to visualize the third observation. From the label, we know this is a:

```
mnist$train$label[3]
#> [1] 4
```

The third row of the matrix $x[3,]$ contains the 784 pixel intensities. We can assume these were entered in order and convert them back to a (28×28) matrix using:

```
grid <- matrix(x[3,], 28, 28)
```

To visualize the data, we can use `image` in the followin way:

```
image(1:28, 1:28, grid)
```

However, because the y-axis in `image` goes bottom to top and x stores pixels top to bottom the code above shows shows a flipped image. To flip it back we can use:

```
image(1:28, 1:28, grid[, 28:1])
```

20.5 Mathematical notation

Matrices are usually represented with bold upper case letters:

$$\begin{bmatrix} \mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,p} \end{bmatrix} \end{bmatrix}$$

with $(x_{i,j})$ representing the (j) -the feature for the (i) -th observation.

We denote vectors with lower case bold letters and represent them as one column matrices, often referred to as *column vectors*. R follows this convention when converting a vector to a matrix:

```
dim(matrix(x[300,]))
#> [1] 784   1
```

However, *column vectors* should not be confused with the columns of the matrix. They have this name simply because they have one column.

Mathematical descriptions of machine learning often make reference to vectors representing the (p) features:

$$\begin{bmatrix} \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \end{bmatrix}$$

To distinguish between features associated with the observations $(i=1,\dots,n)$, we add an index:

$$\begin{bmatrix} \mathbf{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,p} \end{bmatrix} \end{bmatrix}$$

Bold lower case letters are also commonly used to represent matrix columns rather than rows. This can be confusing because (\mathbf{x}_1) can represent either the first row or the first column of (\mathbf{X}) . One way to distinguish is to use notation similar to computer code: using the colon $(:)$ to represent *all*. So $(\mathbf{X}_{1,:})$ represents the first row and $(\mathbf{X}_{:,1})$ is the first column. Another approach is to distinguish by the letter used to index, with (i) used for rows and (j) used for columns. So (\mathbf{x}_i) is the (i) th row and (\mathbf{x}_j) is the (j) th column. With this approach, it is important to clarify which dimension, row or column is being represented. Further confusion can arise because, as aforementioned, it is common to represent all vectors, including the rows of a matrix, as one-column matrices.

20.6 The transpose

A common operation when working with matrices is the *transpose*. We use the transpose to understand several concepts described in the next several sections. This operation simply converts the rows of a matrix into columns. We use the symbols (top) or $(')$ next to the bold upper case letter to denote the transpose:

$$\begin{bmatrix} \text{if } \text{if } \mathbf{X} = \begin{bmatrix} x_{1,1} & \dots & x_{1,p} \\ x_{2,1} & \dots & x_{2,p} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,p} \end{bmatrix} \text{ then } \mathbf{X}^{\text{top}} = \begin{bmatrix} x_{1,1} & x_{2,1} & \dots & x_{n,1} \\ x_{1,2} & x_{2,2} & \dots & x_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,p} & x_{2,p} & \dots & x_{n,p} \end{bmatrix} \end{bmatrix}$$

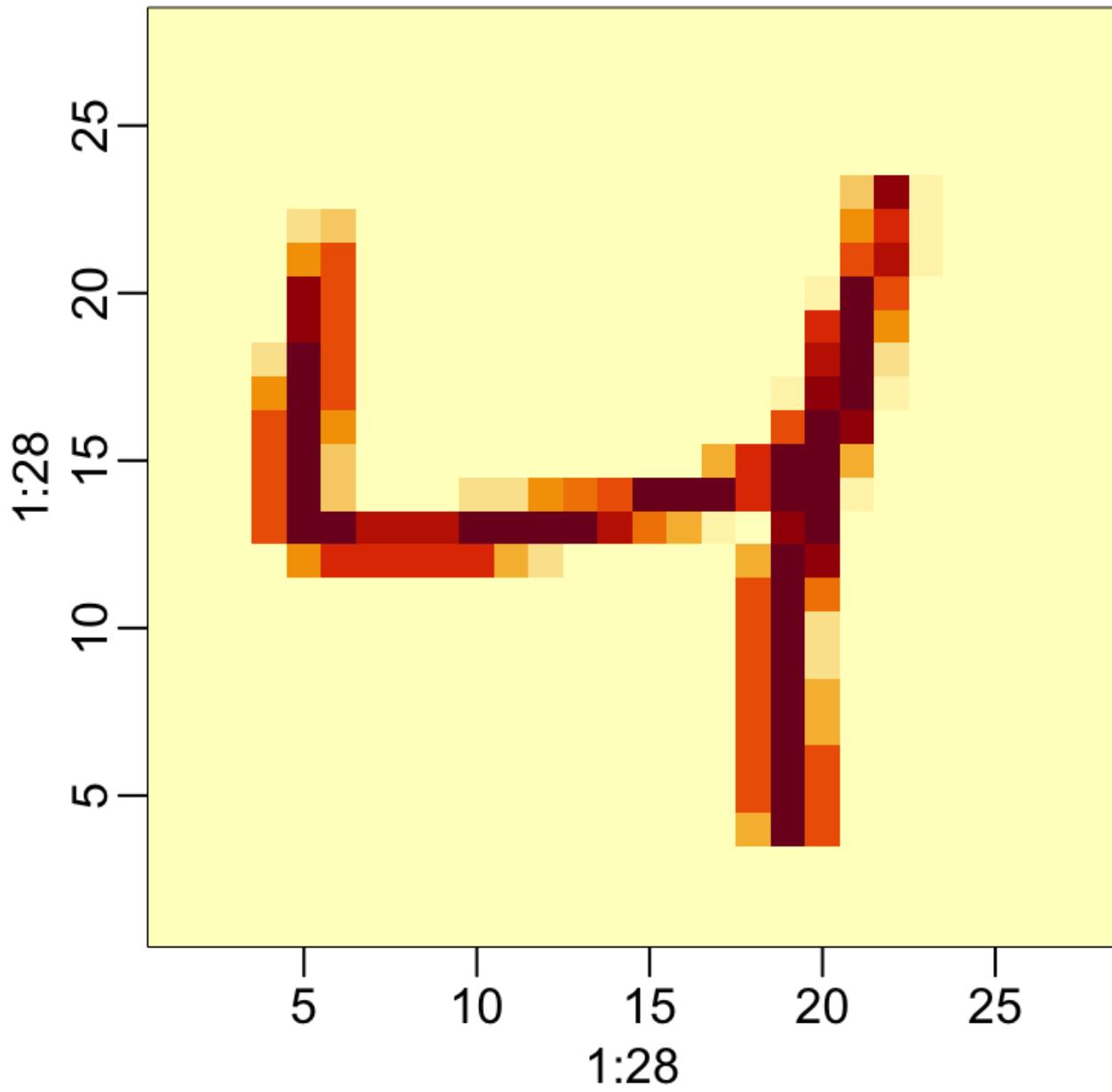


Figure 3:

In R we compute the transpose using the function `t`

```
dim(x)
#> [1] 60000 784
dim(t(x))
#> [1] 784 60000
```

One use of the transpose is that we can write the matrix $\begin{pmatrix} \mathbf{X} \end{pmatrix}$ as rows of the column vectors representing the features for each individual observation in the following way:

```
\[ \mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix}
```

20.7 Row and column summaries

A common operation with matrices is to apply the same function to each row or to each column. For example, we may want to compute row averages and standard deviations. The `apply` function lets you do this. The first argument is the matrix, the second is the dimension, 1 for rows, 2 for columns, and the third is the function to be applied.

So, for example, to compute the averages and standard deviations of each row, we write:

```
avgs <- apply(x, 1, mean)
sds <- apply(x, 1, sd)
```

To compute these for the columns, we simply change the 1 to a 2:

```
avgs <- apply(x, 2, mean)
sds <- apply(x, 2, sd)
```

Because these operations are so common, special functions are available to perform them. So, for example, the functions `rowMeans` computes the average of each row:

```
avg <- rowMeans(x)
```

and the `matrixStats` function `rowSds` computes the standard deviations for each row:

```
library(matrixStats)
sds <- rowSds(x)
```

The functions `colMeans` and `colSds` provide the version for columns. For more fast implementations consider the functions available in `matrixStats`.

Task 2: Do some digits require more ink to write than others?

For the second task, related to total pixel darkness, we want to see the average use of ink plotted against digit. We have already computed this average and can generate a boxplot to answer the question:

```
avg <- rowMeans(x)
boxplot(avg ~ y)
```

From this plot we see that, not surprisingly, 1s use less ink than other digits.

20.8 Conditional filtering

One of the advantages of matrices operations over `tidyverse` operations is that we can easily select columns based on summaries of the columns.

Note that logical filters can be used to subset matrices in a similar way in which they can be used to subset vectors. Here is a simple example subsetting columns with logicals:

```
matrix(1:15, 3, 5)[,c(FALSE, TRUE, TRUE, FALSE, TRUE)]
#>      [,1] [,2] [,3]
#> [1,]    4    7   13
```

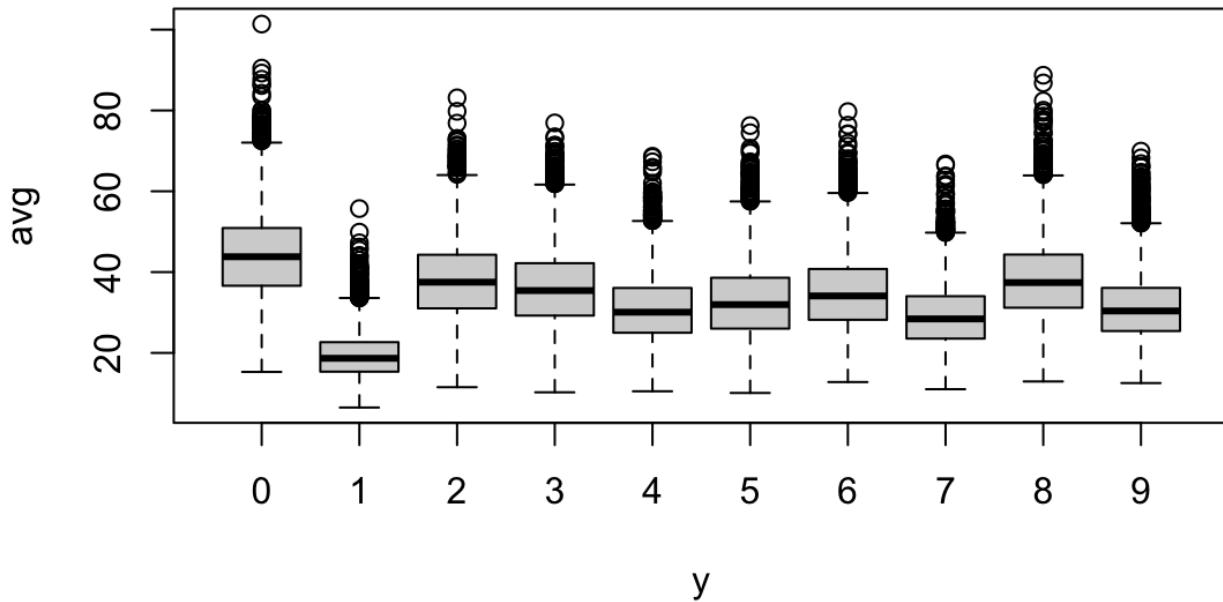


Figure 4:

```
#> [2,]    5    8   14
#> [3,]    6    9   15
```

This implies that we can select rows with conditional expression. In the following example we remove all observations containing at least one NA:

```
x[apply(!is.na(x), 1, all),]
```

This being a common operation, we have a **matrixStats** function to do it faster:

```
x[!rowAnyNAs(x),]
```

Task 3: Are some pixels uninformative?

We can use these ideas to remove columns associated with pixels that don't change much and thus do not inform digit classification. We will quantify the variation of each pixel with its standard deviation across all entries. Since each column represents a pixel, we use the **colSds** function from the **matrixStats** package:

```
sds <- colSds(x)
```

A quick look at the distribution of these values shows that some pixels have very low entry to entry variability:

```
hist(sds, breaks = 30, main = "SDs")
```

This makes sense since we don't write in some parts of the box. Here is the variance plotted by location:

```
image(1:28, 1:28, matrix(sds, 28, 28)[, 28:1])
```

We see that there is little variation in the corners.

We could remove features that have no variation since these can't help us predict.

So if we wanted to remove uninformative predictors from our matrix, we could write this one line of code:

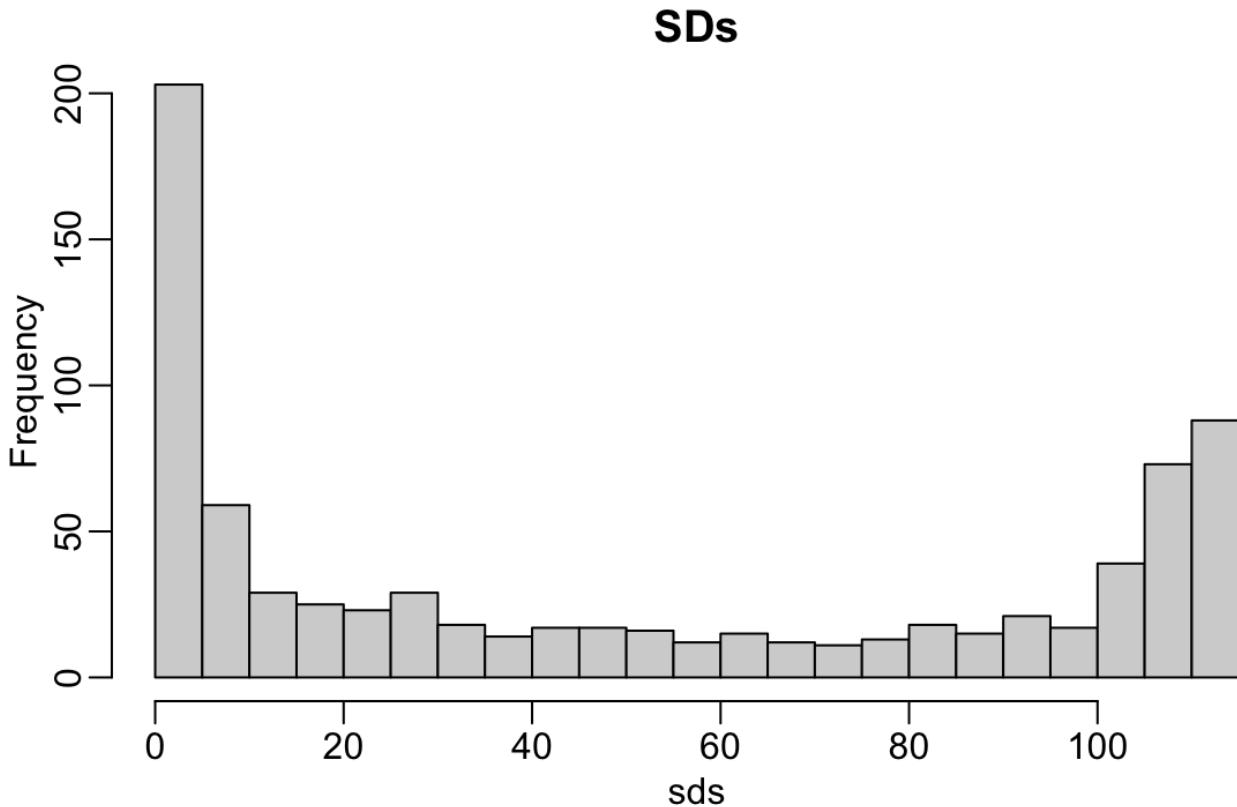


Figure 5:

```
new_x <- x[, colSds(x) > 60]
dim(new_x)
#> [1] 60000   322
```

Only the columns for which the standard deviation is above 60 are kept, which removes over half the predictors.

20.9 Indexing with matrices

An operation that facilitates efficient coding is that we can change entries of a matrix based on conditionals applied to that same matrix. Here is a simple example:

```
mat <- matrix(1:15, 3, 5)
mat[mat > 6 & mat < 12] <- 0
mat
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    4    0    0   13
#> [2,]    2    5    0    0   14
#> [3,]    3    6    0   12   15
```

A useful application of this approach is that we can change all the `NA` entries of a matrix to something else:

```
x[!is.na(x)] <- 0
```

20.9.1 Task 4: Can we remove smudges?

A histogram of all our predictor data:

```
hist(as.vector(x), breaks = 30, main = "Pixel intensities")
```

shows a clear dichotomy which is explained as parts of the image with ink and parts without. If we think that values below, say, 50 are smudges, we can quickly make them zero using:

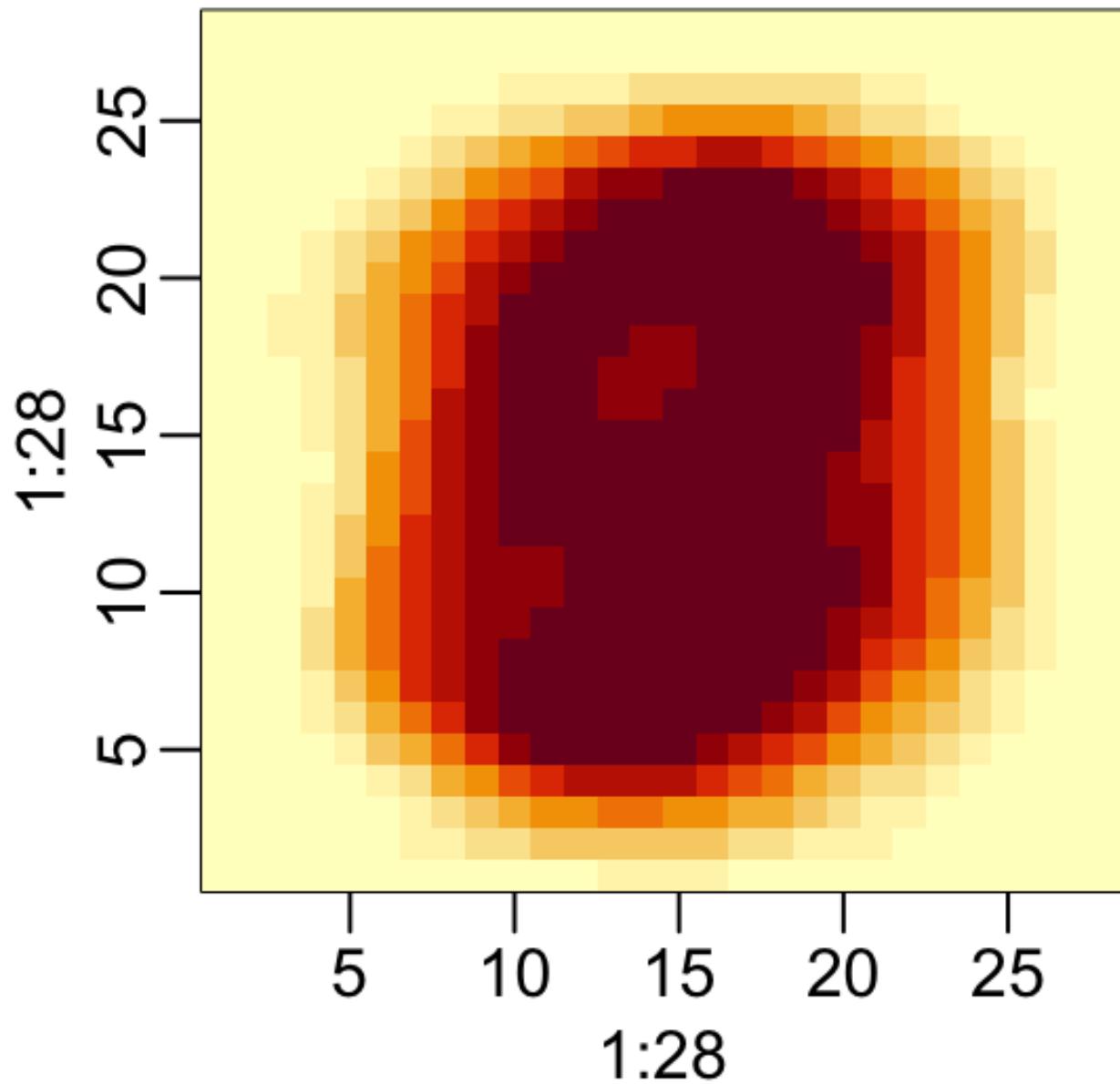


Figure 6:

Pixel intensities

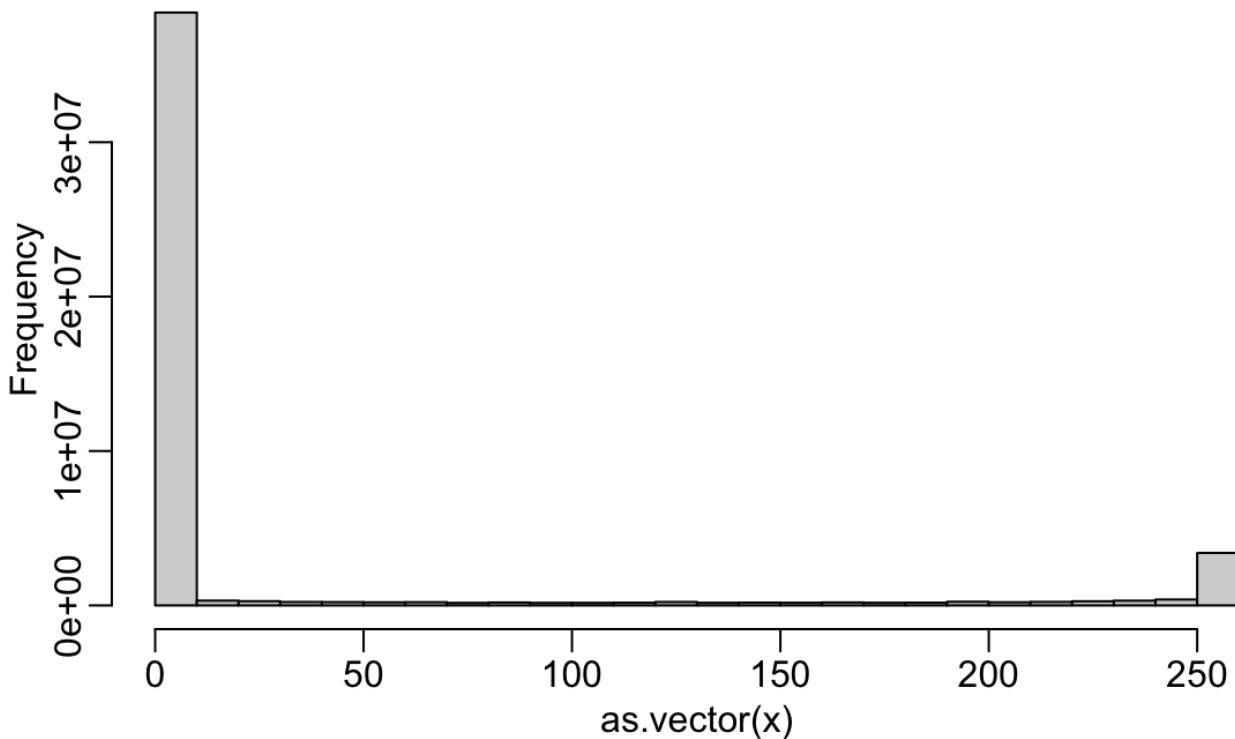


Figure 7:

```
new_x <- x
new_x[new_x < 50] <- 0
```

Task 5: Binarizing the data

The histogram above seems to suggest that this data is mostly binary. A pixel either has ink or does not. Applying what we have learned, we can binarize the data using just matrix operations:

```
bin_x <- x
bin_x[bin_x < 255/2] <- 0
bin_x[bin_x > 255/2] <- 1
```

We can also convert to a matrix of logicals and then coerce to numbers like this:

```
bin_X <- (x > 255/2)*1
```

20.10 Vectorization for matrices

In R, if we subtract a vector from a matrix, the first element of the vector is subtracted from the first row, the second element from the second row, and so on. Using mathematical notation, we would write it as follows:

$$\begin{bmatrix} X_{1,1} & \dots & X_{1,p} \\ X_{2,1} & \dots & X_{2,p} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \dots & X_{n,p} \end{bmatrix} - \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} X_{1,1} - a_1 & \dots & X_{1,p} - a_1 \\ X_{2,1} - a_2 & \dots & X_{2,p} - a_2 \\ \vdots & \ddots & \vdots \\ X_{n,1} - a_n & \dots & X_{n,p} - a_n \end{bmatrix}$$

The same holds true for other arithmetic operations.

The function `sweep` facilitates this type of operation. It works similarly to `apply`. It takes each entry of a vector and applies an arithmetic operation to the corresponding row. Subtraction is the default arithmetic operation. So, for example, to center each row around the average, we can use:

```
sweep(x, 1, rowMeans(x))
```

Task 6: Standardize the digits

The way R vectorizes arithmetic operations implies that we can scale each row of a matrix as follows:

```
(x - rowMeans(x))/rowSds(x)
```

Yet this approach does not work for columns. For columns, we can `sweep`:

```
x_mean_0 <- sweep(x, 2, colMeans(x))
```

To divide by the standard deviation, we change the default arithmetic operation to division as follows:

```
x_standardized <- sweep(x_mean_0, 2, colSds(x), FUN = "/")
```

In R, if you add, subtract, multiple or divide two matrices, the operation is done elementwise. For example, if two matrices are stored in `x` and `y`, then:

```
x*y
```

does not result in matrix multiplication. Instead, the entry in row $\backslash(i\backslash)$ and column $\backslash(j\backslash)$ of this product is the product of the entry in row $\backslash(i\backslash)$ and column $\backslash(j\backslash)$ of `x` and `y`, respectively.

20.11 Exercises

1. Create a 100 by 10 matrix of randomly generated normal numbers. Put the result in `x`.
2. Apply the three R functions that give you the dimension of `x`, the number of rows of `x`, and the number of columns of `x`, respectively.
3. Add the scalar 1 to row 1, the scalar 2 to row 2, and so on, to the matrix `x`.
4. Add the scalar 1 to column 1, the scalar 2 to column 2, and so on, to the matrix `x`. Hint: Use `sweep` with `FUN = "+".`
5. Compute the average of each row of `x`.
6. Compute the average of each column of `x`.
7. For each digit in the MNIST training data, compute the proportion of pixels that are in a *grey area*, defined as values between 50 and 205. Make a boxplot by digit class. Hint: Use logical operators and `rowMeans`.

-
1. <http://yann.lecun.com/exdb/mnist/>

Introduction to Data Science - 21 Applied Linear Algebra

Rafael A. Irizarry

Linear algebra is the main mathematical technique used to describe and motivate statistical methods and machine learning approaches. In this chapter, we introduce some of the mathematical concepts needed to understand these techniques and demonstrate how to work with matrices in R. We use these concepts and techniques throughout the remainder of the book. We start the chapter with a motivating example.

21.1 Matrix multiplication

A commonly used operation in data analysis is matrix multiplication. Here, we define and motivate the operation.

Linear algebra originated from mathematicians developing systematic ways to solve systems of linear equations. For example:

$$\begin{aligned} x + 3y - 2z &= 5 \\ 3x + 5y + 6z &= 7 \\ 2x + 4y + 3z &= 8 \end{aligned}$$

Mathematicians figured out that by representing these linear systems of equations using matrices and vectors, predefined algorithms could be designed to solve any system of linear equations. A basic linear algebra class will teach some of these algorithms, such as Gaussian elimination, the Gauss-Jordan elimination, and the LU and QR decompositions. These methods are usually covered in detail in university level linear algebra courses.

To explain matrix multiplication, define two matrices: \mathbf{A} and \mathbf{B}

$$\begin{aligned} \mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{pmatrix} \end{aligned}$$

and define the product of matrices \mathbf{A} and \mathbf{B} as the matrix $\mathbf{C} = \mathbf{A}\mathbf{B}$ that has entries c_{ij} equal to the sum of the component-wise product of the i th row of \mathbf{A} with the j th column of \mathbf{B} . Using R code, we can define $\mathbf{C} = \mathbf{A}\mathbf{B}$ as follows:

```
m <- nrow(A)
p <- ncol(B)
C <- matrix(0, m, p)
for(i in 1:m){
  for(j in 1:p){
    C[i,j] <- sum(A[i,] * B[,j])
  }
}
```

Because this operation is so common, R includes a mathematical operator `%*%` for matrix multiplication:

```
C <- A %*% B
```

Using mathematical notation $\mathbf{C} = \mathbf{A}\mathbf{B}$ looks like this:

$$\begin{aligned} c_{11} &= a_{11}b_{11} + \dots + a_{1n}b_{n1} \\
c_{12} &= a_{11}b_{12} + \dots + a_{1n}b_{n2} \\
\vdots & \vdots \\
c_{m1} &= a_{m1}b_{11} + \dots + a_{mn}b_{n1} \\
c_{m2} &= a_{m1}b_{12} + \dots + a_{mn}b_{n2} \end{aligned}$$

$$a_{m1}b_{11} + \dots + a_{mn}b_{n1} & a_{m1}b_{n2} + \dots + a_{mn}b_{n2} & \dots & a_{m1}b_{1p} \\ + \dots + a_{mn}b_{np} \end{pmatrix}$$

Note this definition implies that the multiplication $\mathbf{A}\mathbf{B}$ is only possible when the number of rows of \mathbf{A} matches the number of columns of \mathbf{B} .

So how does this definition of matrix multiplication help solve systems of equations? First, any system of equations with unknowns (x_1, \dots, x_n)

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \end{aligned}$$

can now be represented as matrix multiplication by defining the following matrices:

$$\begin{matrix} \mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \end{matrix}$$

and rewriting the equation simply as:

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

The linear algebra algorithms listed above, such as Gaussian elimination, provide a way to compute the *inverse* matrix \mathbf{A}^{-1} that solves the equation for \mathbf{x} :

$$\mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

To solve the first equation we wrote out in R, we can use the function `qr.solve`:

```
A <- matrix(c(1, 3, -2, 3, 5, 6, 2, 4, 3), 3, 3, byrow = TRUE)
b <- matrix(c(5, 7, 8))
solve(A, b)
```

The function `solve` works well when dealing with small to medium-sized matrices with a similar range for each column and not too many 0s. The function `qr.solve` can be used when this is not the case.

21.2 The identity matrix

The identity matrix, represented with a bold \mathbf{I} , is like the number 1, but for matrices: if you multiply a matrix by the identity matrix, you get back the matrix.

$$\mathbf{I}\mathbf{X} = \mathbf{X}$$

If you define \mathbf{I} as matrix with the same number of rows and columns (referred to as square matrix) with 0s everywhere except the diagonal:

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

you will obtain the desired property.

Note that the definition of an inverse matrix implies that:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{1}$$

Because the default for the second argument in `solve` is an identity matrix, if we simply type `solve(A)`, we obtain the inverse \mathbf{A}^{-1} . This means we can also obtain a solution to our system of equations with:

```
solve(A) %*% b
```

21.3 Distance

Many of the analyses we perform with high-dimensional data relate directly or indirectly to distance. For example, most machine learning techniques rely on being able to define distances between observations, using features or predictors. Clustering algorithms, for example, search of observations that are *similar*. But what does this mean mathematically?

To define distance, we introduce another linear algebra concept: the *norm*. Recall that a point in two dimensions can be represented in polar coordinates as:

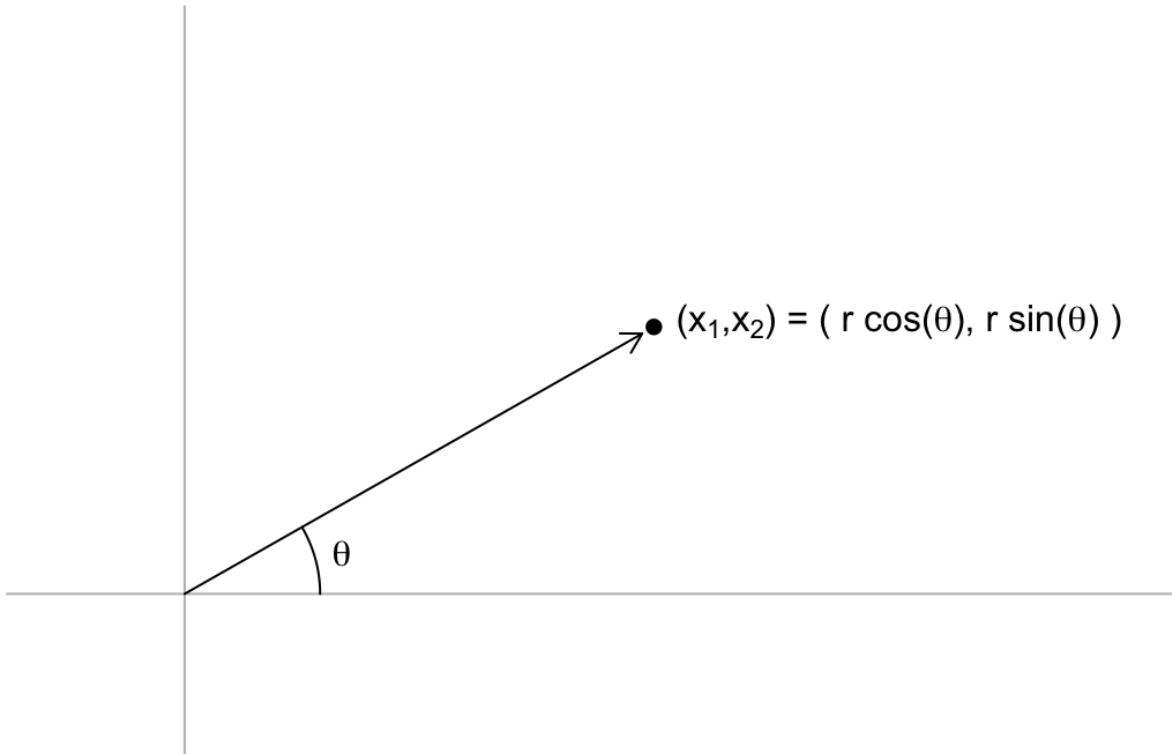


Figure 1:

with $\theta = \arctan\left(\frac{x_2}{x_1}\right)$ and $r = \sqrt{x_1^2 + x_2^2}$. If we think of the point as two dimensional column vector $\mathbf{x} = (x_1, x_2)^T$, r defines the norm of $\|\mathbf{x}\|$. The norm can be thought of as the *size* of the two-dimensional vector disregarding the direction: if we change the angle, the vector changes but the size does not. The point of defining the norm is that we can extrapolate the concept of *size* to higher dimensions. Specifically, we write the norm for any vector \mathbf{x} as:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \dots + x_p^2}$$

We can use the linear algebra concepts we have learned to define the norm like this:

$$\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x}$$

To define distance, suppose we have two two-dimensional points: \mathbf{x}_1 and \mathbf{x}_2 . We can define how similar they are by simply using euclidean distance:

We know that the distance is equal to the length of the hypotenuse:

$$\sqrt{(x_{11} - x_{12})^2 + (x_{21} - x_{22})^2}$$

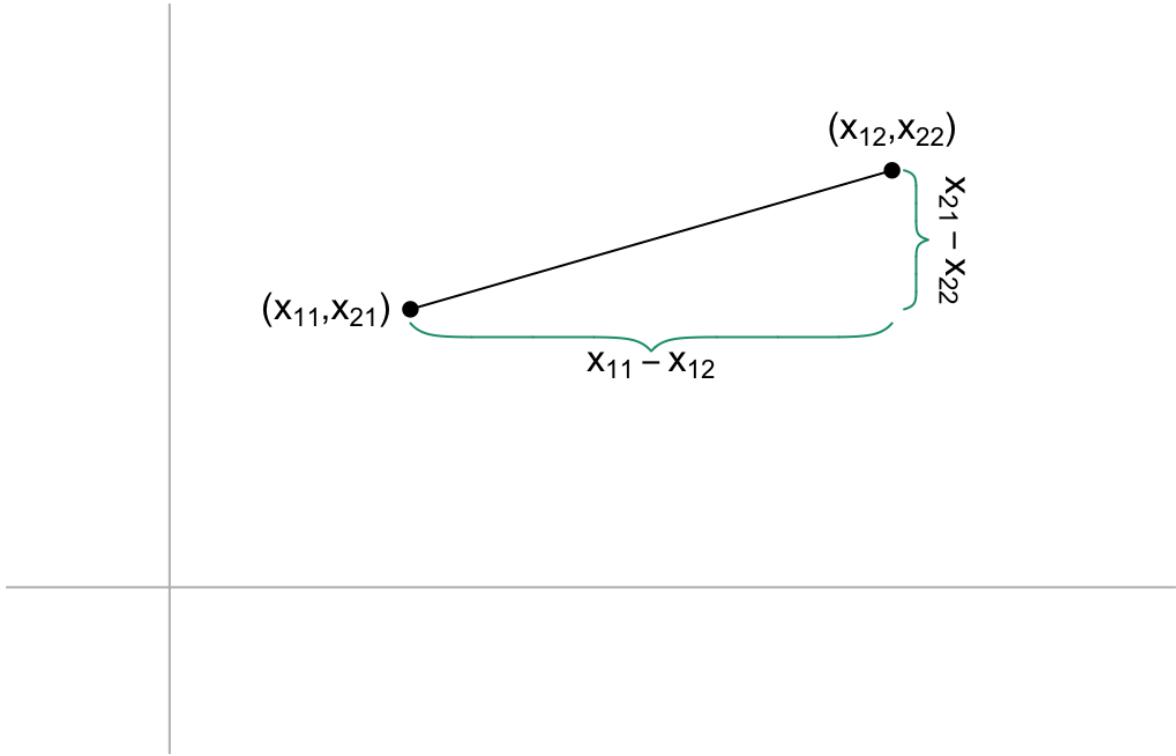


Figure 2:

The reason we introduced the norm is because this distance is the size of the vector between the two points and this can be extrapolated to any dimension. The distance between two points, regardless of the dimensions, is defined as the norm of the difference:

$$\sqrt{\|\mathbf{x}_1 - \mathbf{x}_2\|}$$

If we use the digit data, the distance between the first and second observation will compute distance using all 784 features:

$$\sqrt{\|\mathbf{x}_1 - \mathbf{x}_2\|^2} = \sqrt{\sum_{j=1}^{784} (x_{1,j} - x_{2,j})^2}$$

To demonstrate, let's pick the features for three digits:

```
x_1 <- x[6,]
x_2 <- x[17,]
x_3 <- x[16,]
```

We can compute the distances between each pair using the definitions we just learned:

```
c(sum((x_1 - x_2)^2), sum((x_1 - x_3)^2), sum((x_2 - x_3)^2)) |> sqrt()
#> [1] 2320 2331 2519
```

In R, the function `crossprod(x)` is convenient for computing norms. It multiplies $t(x)$ by x :

```
c(crossprod(x_1 - x_2), crossprod(x_1 - x_3), crossprod(x_2 - x_3)) |> sqrt()
#> [1] 2320 2331 2519
```

Note `crossprod` takes a matrix as the first argument. As a result, the vectors used here are being coerced into single column matrices. Also, note that `crossprod(x,y)` multiples $t(x)$ by y .

We can see that the distance is smaller between the first two. This agrees with the fact that the first two are 2s and the third is a 7.

```
y[c(6, 17, 16)]
#> [1] 2 2 7
```

We can also compute **all** the distances at once relatively quickly using the function `dist`, which computes the distance between each row and produces an object of class `dist`:

```
d <- dist(x[c(6,17,16),])
class(d)
#> [1] "dist"
```

There are several machine learning related functions in R that take objects of class `dist` as input. To access the entries using row and column indices, we need to coerce it into a matrix. We can see the distance we calculated above like this:

```
d
#>      1    2
#> 2 2320
#> 3 2331 2519
```

The `image` function allows us to quickly see an image of distances between observations. As an example, we compute the distance between each of the first 300 observations and then make an image:

```
d <- dist(x[1:300,])
image(as.matrix(d))
```

If we order this distance by the labels, we can see yellowish squares near the diagonal. This is because observations from the same digits tend to be closer than to different digits:

```
image(as.matrix(d)[order(y[1:300]), order(y[1:300])])
```

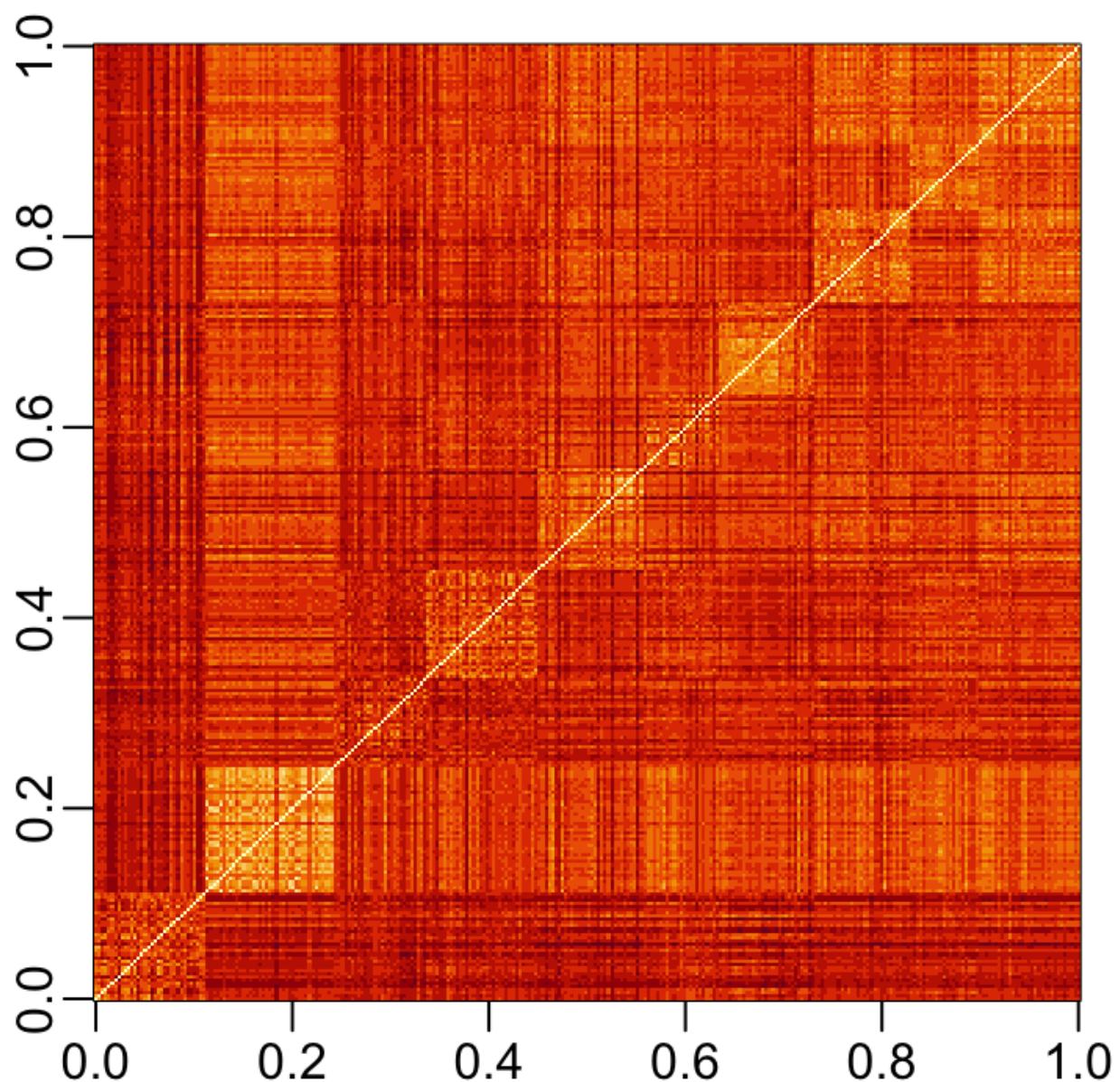


Figure 3:

21.4 Spaces

Predictor space is a concept that is often used to describe machine learning algorithms. The term *space* refers to an advanced mathematical definition for which we provide a simplified explanation to help understand the term predictor space when used in the context of machine learning algorithms.

We can think of all predictors $\{(x_{i,1}, \dots, x_{i,p})\}^{\text{top}}$ for all observations $(i=1, \dots, n)$ as $(n \times p)$ -dimensional points. A *space* can be thought of as the collection of all possible points that should be considered for the data analysis in question. This includes points we could see, but have not been observed yet. In the case of the handwritten digits, we can think of the predictor space as any point $\{(x_1, \dots, x_p)\}^{\text{top}}$ as long as each entry $(x_i, i = 1, \dots, p)$ is between 0 and 255.

Some Machine Learning algorithms also define subspaces. A commonly defined subspace in machine learning are *neighborhoods* composed of points that are close to a predetermined *center*. We do this by selecting a center (\mathbf{x}_0) , a minimum distance (r) , and defining the subspace as the collection of points $\{(\mathbf{x})\}$ that satisfy:

$$[(\|\mathbf{x} - \mathbf{x}_0\| \leq r)]$$

We can think of this subspace as a multidimensional sphere since every point is the same distance away from the center.

Other machine learning algorithms partition the predictor space into non-overlapping regions and then make different predictions for each region using the data in the region. We will learn about these in Section 30.4.

21.5 Exercises

1. Generate two matrix, A and B, containing randomly generated and normally distributed numbers. The dimensions of these two matrices should be (4×3) and (3×6) , respectively. Confirm that $C \leftarrow A \%*% B$ produces the same results as:

```
m <- nrow(A)
p <- ncol(B)
C <- matrix(0, m, p)
for(i in 1:m){
  for(j in 1:p){
    C[i,j] <- sum(A[i,] * B[,j])
  }
}
```

2. Solve the following system of equations using R:

$$\begin{aligned} x + y + z + w &= 10 \\ 2x + 3y - z - w &= 5 \\ 3x - y + 4z - 2w &= 15 \\ 2x + 2y - 2z - 2w &= 20 \end{aligned}$$

3. Define x:

```
mnist <- read_mnist()
x <- mnist$train$images[1:300,]
y <- mnist$train$labels[1:300]
```

and compute the distance matrix:

```
d <- dist(x)
class(d)
```

Generate a boxplot showing the distances for the second row of d stratified by digits. Do not include the distance to itself, which we know is 0. Can you predict what digit is represented by the second row of x?

4. Use the `apply` function and matrix algebra to compute the distance between the second digit `mnist$train$images[4,]` and all other digits represented in `mnist$train$images`. Then generate a boxplot as in exercise 2 and predict what digit is the fourth row.

5. Compute the distance between each feature and the feature representing the middle pixel (row 14 column 14). Create an image plot of where the distance is shown with color in the pixel position.

Introduction to Data Science - 22 Dimension reduction

Rafael A. Irizarry

A typical machine learning task involves working with a large number of predictors, which makes visualization somewhat challenging. We have shown methods for visualizing univariate and paired data, but plots that reveal relationships between many variables are more complicated in higher dimensions. For example, to compare each of the 784 features in our predicting digits example, we would have to create 306,936 scatterplots. Creating one single scatterplot of the data is impossible due to the high dimensionality.

Here we describe powerful techniques useful for exploratory data analysis, among other things, generally referred to as *dimension reduction*. The general idea is to reduce the dimension of the dataset while preserving important characteristics, such as the distance between features or observations. With fewer dimensions, visualization then becomes more feasible. The general technique behind it all, the singular value decomposition, is also useful in other contexts. Principal component analysis (PCA) is the approach we will be showing. We will motivate the ideas with a simple illustrative example and present some mathematical concepts needed to understand PCA. We finish the chapter by demonstrating the use of PCA in two more complex datasets.

22.1 Motivation: preserving distance

We consider an example with twin heights. Some pairs are adults, the others are children. Here we simulate 100 two-dimensional points that represent the number of standard deviations each individual is from the mean height. Each point is a pair of twins. We use the `mvrnorm` function from the **MASS** package to simulate bivariate normal data.

```
set.seed(1983)
library(MASS)
n <- 100
rho <- 0.9
sigma <- 3
s <- sigma^2 * matrix(c(1, rho, rho, 1), 2, 2)
x <- rbind(mvrnorm(n/2, c(69, 69), s),
           mvrnorm(n/2, c(60, 60), s))
```

A scatterplot quickly reveals that the correlation is high and that there are two groups of twins, the adults (upper right points) and the children (lower left points):

Our features are \sqrt{n} two-dimensional points, the two heights. For illustrative purposes, we will pretend that visualizing two dimensions is too challenging and we want to explore the data through a histogram of a one-dimensional variable. We therefore want to reduce the dimensions from two to one, but still be able to understand important characteristics of the data, in particular that the observations cluster into two groups: adults and children. To show the ideas presented here are generally useful, we will standardize the data so that observations are in standard units rather than inches:

```
library(matrixStats)
x <- sweep(x, 2, colMeans(x))
x <- sweep(x, 2, colSds(x), "/")
```

In the figure above, we show the distance between observation 1 and 2 (blue), and observation 1 and 51 (red). Note that the blue line is shorter, which implies that 1 and 2 are closer.

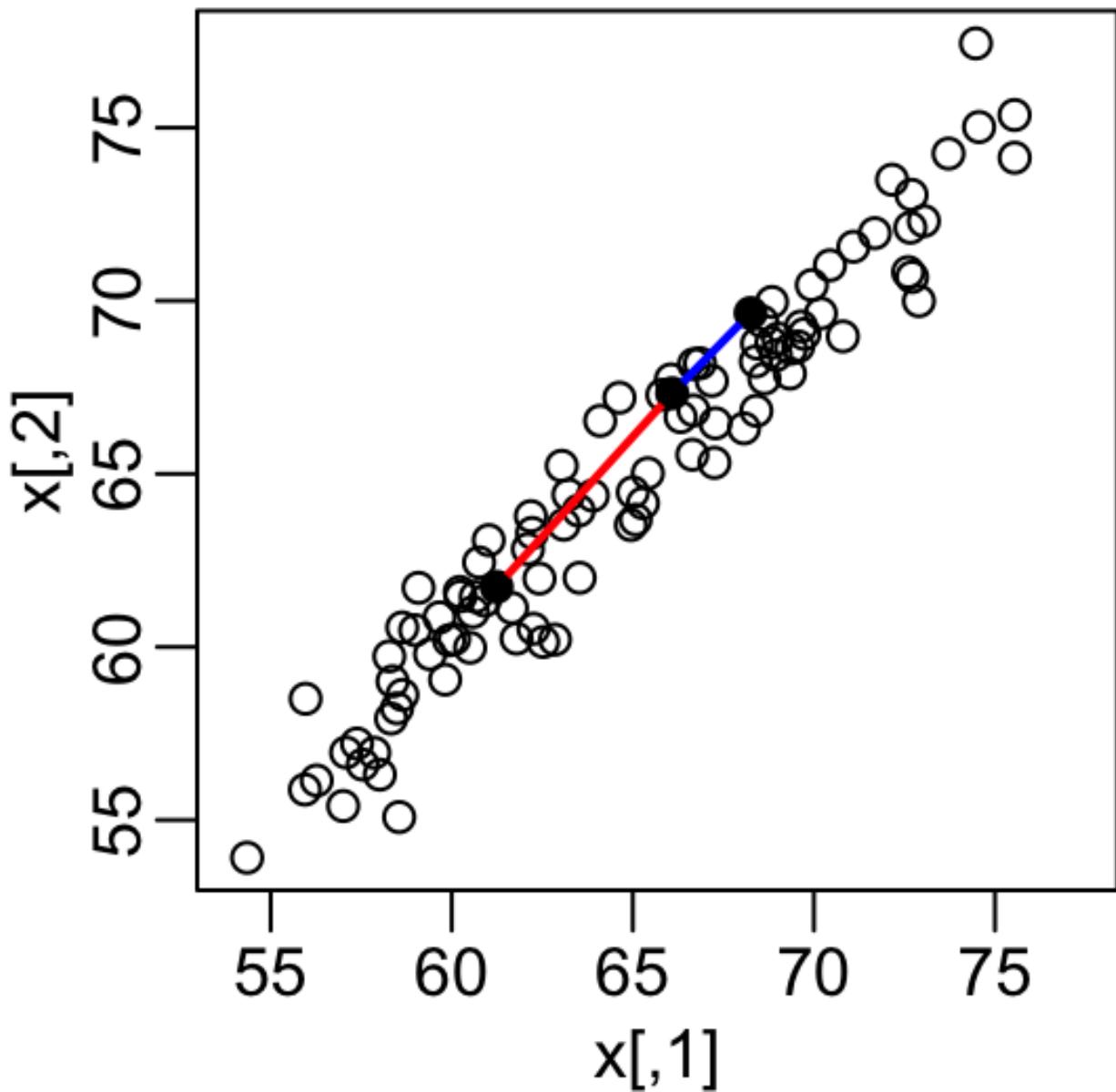


Figure 1:

We can compute these distances using `dist`:

```
d <- dist(x)
as.matrix(d) [1, 2]
#> [1] 0.595
as.matrix(d) [2, 51]
#> [1] 1.39
```

This distance is based on two dimensions and we need a distance approximation based on just one.

Let's start with the naive approach of simply removing one of the two dimensions. Let's compare the actual distances to the distances computed with just the first dimension:

```
z <- x[,1]
```

To make the distances comparable, we divide the sum of squares by the number of dimensions being added. So for the two dimensional case, we have:

$$\sqrt{\frac{1}{2} \sum_{j=1}^2 (x_{1,j} - x_{2,j})^2}$$

To make the distances comparable, we divide by $\sqrt{2}$:

```
plot(dist(x) / sqrt(2), dist(z))
abline(0, 1, col = "red")
```

This one number summary does ok at preserving distances, but, can we pick a one-dimensional summary that improves the approximation?

If we look back at the scatterplot and visualize a line between any pair of points, the length of this line is the distance between the two points. These lines tend to go along the direction of the diagonal. We will learn that we can *rotate* the points in a way that preserve the distance between points, while increasing the variability in one dimension and reducing it on the other. Using this method, we keep more of the *information* about distances in the first dimension. In the next section, we describe a mathematical approach that permits us to find rotations that preserve distance between points. We can then find the rotation that maximizes the variability in the first dimension.

22.2 Rotations

Any two-dimensional point $((x_1, x_2))$ can be written as the base and height of a triangle with a hypotenuse going from $(0,0)$ to $((x_1, x_2))$:

$$x_1 = r \cos \phi, \quad x_2 = r \sin \phi$$

with r the length of the hypotenuse and ϕ the angle between the hypotenuse and the x-axis.

To *rotate* the point $((x_1, x_2))$ around a circle with center $(0,0)$ and radius r by an angle θ we simply change the angle in the previous equation to $\phi + \theta$:

$$z_1 = r \cos(\phi + \theta), \quad z_2 = r \sin(\phi + \theta)$$

We can use trigonometric identities to rewrite $((z_1, z_2))$ as follows:

$$\begin{aligned} z_1 &= r \cos(\phi + \theta) = r \cos \phi \cos \theta - r \sin \phi \sin \theta = x_1 \cos \theta - x_2 \sin \theta \\ z_2 &= r \sin(\phi + \theta) = r \cos \phi \sin \theta + r \sin \phi \cos \theta = x_1 \sin \theta + x_2 \cos \theta \end{aligned}$$

Now we can rotate each point in the dataset by simply applying the formula above to each pair $((x_{i,1}, x_{i,2}))$. Here is what the twin standardized heights look like after rotating each point by -45° degrees:

Note that while the variability of x_1 and x_2 are similar, the variability of z_1 is much larger than the variability of z_2 . Also, notice that the distances between points appear to be preserved. In the next sections, we show mathematically that this in fact the case.

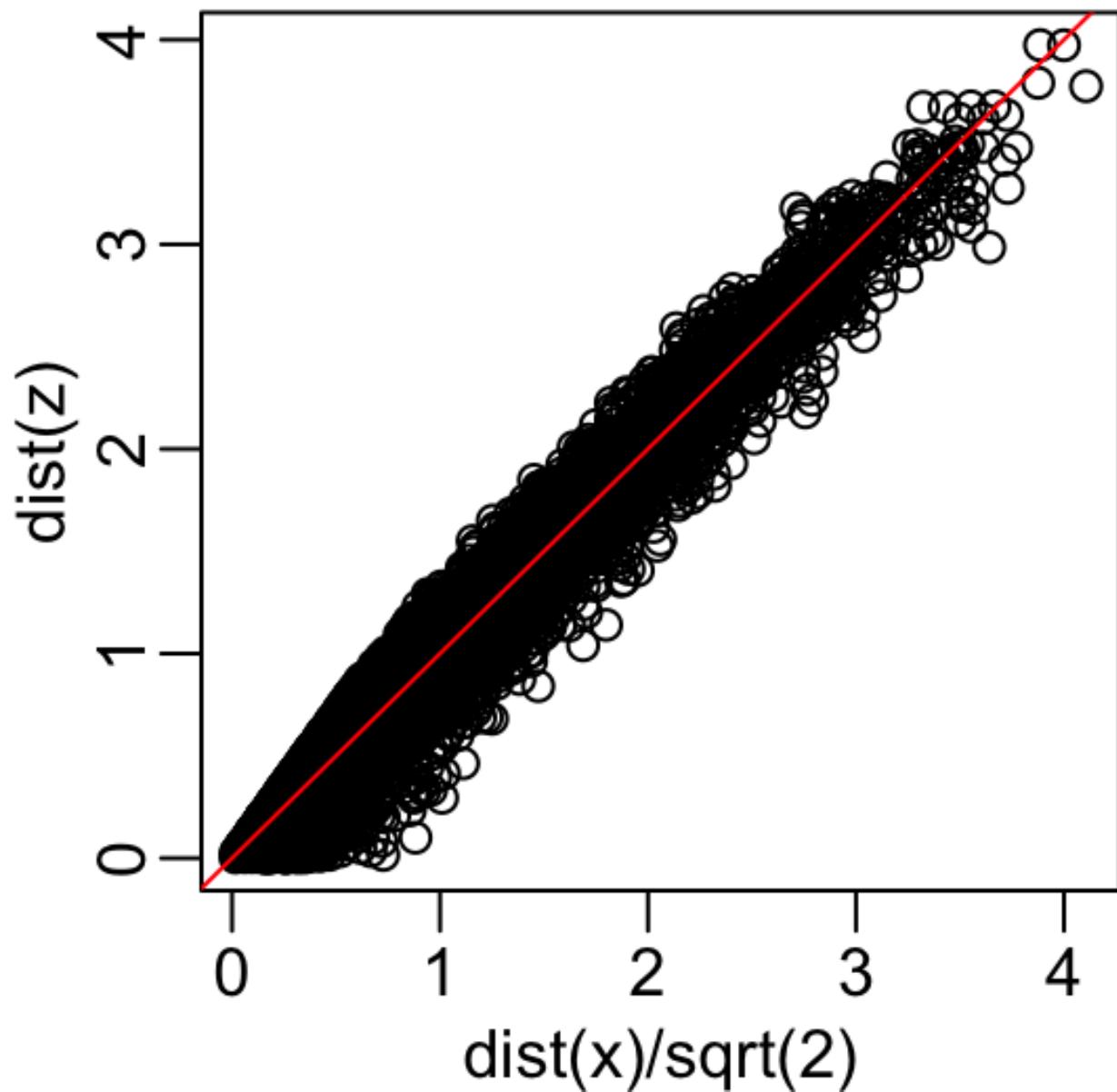


Figure 2:

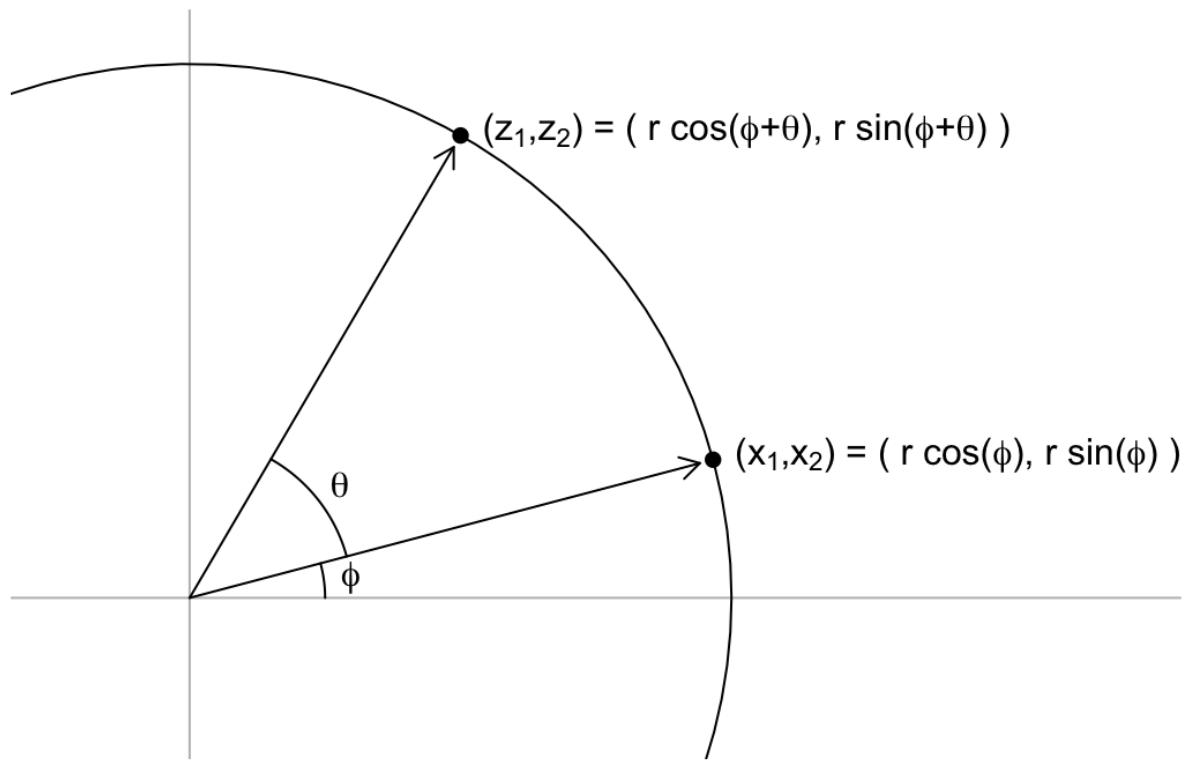


Figure 3:

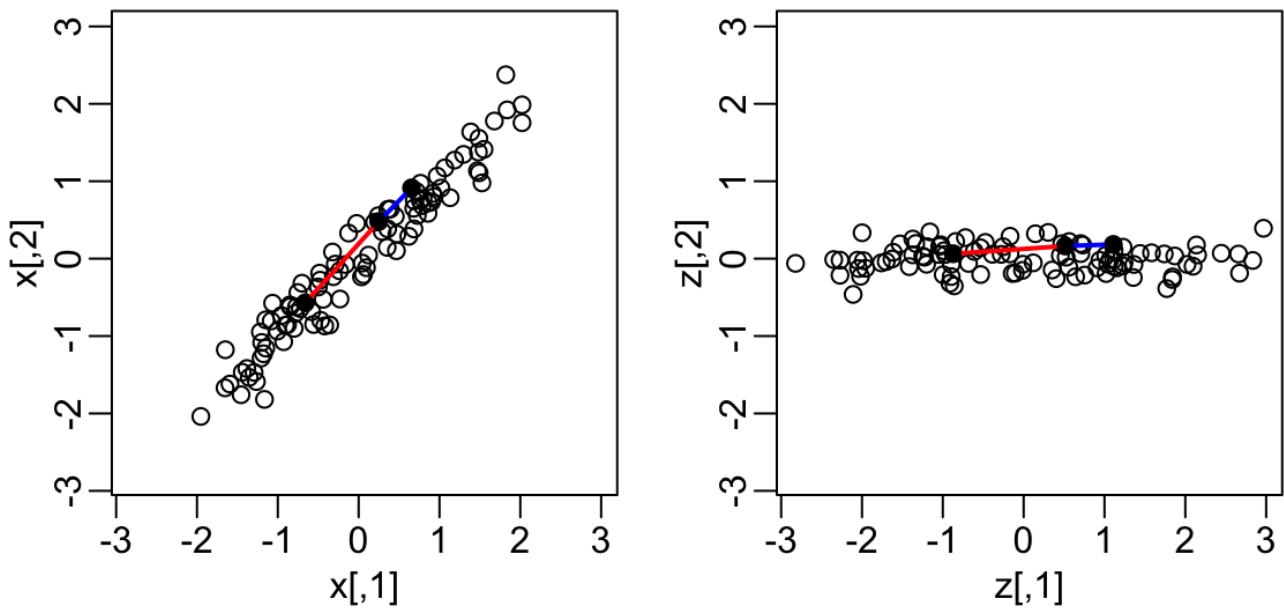


Figure 4:

22.3 Linear transformations

Any time a matrix \mathbf{X} is multiplied by another matrix \mathbf{A} , we refer to the product $\mathbf{Z} = \mathbf{X}\mathbf{A}$ as a linear transformation of \mathbf{X} . Below, we show that the rotations described above are a linear transformation. To see this, note that for any row i , the first entry was:

$$z_{i,1} = a_{1,1}x_{i,1} + a_{2,1}x_{i,2}$$

with $a_{1,1} = \cos\theta$ and $a_{2,1} = -\sin\theta$.

The second entry was also a linear transformation:

$$z_{i,2} = a_{1,2}x_{i,1} + a_{2,2}x_{i,2}$$

with $a_{1,2} = \sin\theta$ and $a_{2,2} = \cos\theta$.

We can write these equations using matrix notation:

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

An advantage of using linear algebra is that we can write the transformation for the entire dataset by saving all observations in a $(N \times 2)$ matrix:

$$\begin{bmatrix} \mathbf{X} \end{bmatrix} \equiv \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} \\ \vdots & \vdots \\ x_{n,1} & x_{n,2} \end{bmatrix}$$

We can then obtain the rotated values \mathbf{z}_i for each row i by applying a *linear transformation* of \mathbf{X} :

$$\begin{bmatrix} \mathbf{Z} = \mathbf{X}\mathbf{A} \end{bmatrix} \text{ with } \mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

If we define:

```
theta <- 2*pi*-45/360 #convert to radians
A <- matrix(c(cos(theta), -sin(theta), sin(theta), cos(theta)), 2, 2)
```

We can write code implementing a rotation by any angle θ using linear algebra:

```
rotate <- function(x, theta){
  theta <- 2*pi*theta/360
  A <- matrix(c(cos(theta), -sin(theta), sin(theta), cos(theta)), 2, 2)
  x %*% A
}
```

The columns of \mathbf{A} are referred to as *directions* because if we draw a vector from $(0,0)$ to $((a_{1,j}, a_{2,j}))$, it points in the direction of the line that will become the $(j\text{-th})$ dimension.

Another advantage of linear algebra is that if we can find the inverse matrix of \mathbf{A}^\top , we can convert \mathbf{Z} back to \mathbf{X} again using a linear transformation.

In this particular case, we can use trigonometry to show that:

$$x_{i,1} = b_{1,1}z_{i,1} + b_{2,1}z_{i,2} \quad x_{i,2} = b_{1,2}z_{i,1} + b_{2,2}z_{i,2}$$

with $b_{1,1} = \cos\theta$, $b_{2,1} = \sin\theta$, $b_{1,2} = -\sin\theta$, and $b_{2,2} = \cos\theta$.

This implies that:

$$\begin{bmatrix} \mathbf{X} = \mathbf{Z}\mathbf{A}^\top \end{bmatrix} \text{ Note that the transformation used above is actually } \mathbf{A}^\top \text{ which implies that}$$

$$\begin{bmatrix} \mathbf{Z} = \mathbf{X}\mathbf{A}^\top \end{bmatrix} = \mathbf{X}\mathbf{A}\mathbf{A}^\top = \mathbf{X}$$

and therefore that \mathbf{A}^\top is the inverse of \mathbf{A} . This also implies that all the information in \mathbf{X} is included in the rotation \mathbf{Z} , and it can be retrieved via a linear transformation.

A consequence is that for any rotation the distances are preserved. Here is an example for a 30 degree rotation, although it works for any angle:

```
all.equal(as.matrix(dist(rotate(x, 30))), as.matrix(dist(x)))
#> [1] TRUE
```

The next section explains why this happens.

22.4 Orthogonal transformations

Recall that the distance between two points, say rows \mathbf{z}_h and \mathbf{z}_i of the transformation \mathbf{Z} , can be written like this:

$$\|\mathbf{z}_h - \mathbf{z}_i\| = (\mathbf{z}_h - \mathbf{z}_i)^T (\mathbf{z}_h - \mathbf{z}_i)$$

with \mathbf{z}_h and \mathbf{z}_i the $p \times 1$ column vectors stored in the (h) -th and (i) -th rows of \mathbf{X} , respectively.

Remember that we represent the rows of a matrix as column vectors. This explains why we use \mathbf{A}^T when showing the multiplication for the matrix $\mathbf{Z} = \mathbf{X}\mathbf{A}$, but transpose the operation when showing the transformation for just one observation: $\mathbf{z}_i = \mathbf{A}^T \mathbf{x}_i$

Using linear algebra, we can rewrite the quantity above as:

$$\|\mathbf{z}_h - \mathbf{z}_i\| = \|\mathbf{A}^T \mathbf{x}_h - \mathbf{A}^T \mathbf{x}_i\|^2 = (\mathbf{x}_h - \mathbf{x}_i)^T \mathbf{A} \mathbf{A}^T (\mathbf{x}_h - \mathbf{x}_i)$$

Note that if $\mathbf{A}^T \mathbf{A} = \mathbf{I}$, then the distance between the (h) -th and (i) -th rows is the same for the original and transformed data.

We refer to transformation with the property $\mathbf{A}^T \mathbf{A} = \mathbf{I}$ as *orthogonal transformations*. These are guaranteed to preserve the distance between any two points.

We previously demonstrated our rotation has this property. We can confirm using R:

```
A %*% t(A)
#>           [,1]      [,2]
#> [1,] 1.00e+00 1.01e-17
#> [2,] 1.01e-17 1.00e+00
```

Notice that \mathbf{A} being orthogonal also guarantees that the total sum of squares (TSS) of \mathbf{X} , defined as $\sum_{i=1}^n \sum_{j=1}^p x_{ij}^2$ is equal to the total sum of squares of the rotation $\mathbf{Z} = \mathbf{X}\mathbf{A}^T$. To illustrate, observe that if we denote the rows of \mathbf{Z} as $\mathbf{z}_1, \dots, \mathbf{z}_n$, then sum of squares can be written as:

$$\sum_{i=1}^n \|\mathbf{z}_i\|^2 = \sum_{i=1}^n \|\mathbf{A}^T \mathbf{x}_i\|^2 = \sum_{i=1}^n \mathbf{x}_i^T \mathbf{A} \mathbf{A}^T \mathbf{x}_i = \sum_{i=1}^n \mathbf{x}_i^T \mathbf{x}_i = \sum_{i=1}^n \sum_{j=1}^p x_{ij}^2$$

We can confirm using R:

```
theta <- -45
z <- rotate(x, theta) # works for any theta
sum(x^2)
#> [1] 198
sum(z^2)
#> [1] 198
```

This can be interpreted as a consequence of the fact that an orthogonal transformation guarantees that all the information is preserved.

However, although the total is preserved, the sum of squares for the individual columns changes. Here we compute the proportion of TSS attributed to each column, referred to as the *variance explained* or *variance captured* by each column, for \mathbf{X} :

```
colSums(x^2)/sum(x^2)
#> [1] 0.5 0.5
```

and $\|\mathbf{Z}\|$:

```
colSums(z^2)/sum(z^2)
#> [1] 0.9848 0.0152
```

In the next section, we describe how this last mathematical result can be useful.

22.5 Principal Component Analysis (PCA)

We have established that orthogonal transformations preserve the distance between observations and the total sum of squares. We have also established that, while the TSS remains the same, the way this total is distributed across the columns can change.

The general idea behind Principal Component Analysis (PCA) is to try to find orthogonal transformations that concentrate the variance explained in the first few columns. We can then focus on these few columns, effectively reducing the dimension of the problem. In our specific example, we are looking for the rotation that maximizes the variance explained in the first column. The following code performs a grid search across rotations from -90 to 0:

```
angles <- seq(0, -90)
v <- sapply(angles, function(angle) colSums(rotate(x, angle)^2))
variance_explained <- v[1]/sum(x^2)
plot(angles, variance_explained, type = "l")
```

We find that a -45 degree rotation appears to achieve the maximum, with over 98% of the total variability explained by the first dimension. We denote this rotation matrix with $\|\mathbf{V}\|$:

```
theta <- 2*pi*-45/360 #convert to radians
V <- matrix(c(cos(theta), -sin(theta), sin(theta), cos(theta)), 2, 2)
```

We can rotate the entire dataset using:

```
\[ \mathbf{Z} = \mathbf{X} \mathbf{V} \]
z <- x %*% V
```

The following animation further illustrates how different rotations affect the variability explained by the dimensions of the rotated data:

The first dimension of \mathbf{z} is referred to as the *first principal component (PC)*. Because almost all the variation is explained by this first PC, the distance between rows in \mathbf{x} can be very well approximated by the distance calculated with just $\mathbf{z[,1]}$.

We also notice that the two groups, adults and children, can be clearly observed with the one number summary, better than with any of the two original dimensions.

```
hist(x[,1], breaks = seq(-4,4,0.5))
hist(x[,2], breaks = seq(-4,4,0.5))
hist(z[,1], breaks = seq(-4,4,0.5))
```

We can visualize these to see how the first component summarizes the data. In the plot below, red represents high values and blue negative values:

This idea generalizes to dimensions higher than 2. As done in our two dimensional example, we start by finding the $(p \times 1)$ vector $\|\mathbf{v}_1\|$ with $\|\mathbf{v}_1\|=1$ that maximizes $\|\mathbf{X} \mathbf{v}_1\|$. The projection $\|\mathbf{X} \mathbf{v}_1\|$ is the first PC. To find the second PC, we subtract the variation explained by first PC from $\|\mathbf{X}\|$:

```
\[ \mathbf{r} = \mathbf{X} - \mathbf{X} \mathbf{v}_1 \mathbf{v}_1^\top \]
```

and then find the vector $\|\mathbf{v}_2\|$ with $\|\mathbf{v}_2\|=1$ that maximizes $\|\mathbf{r} \mathbf{v}_2\|$. The projection $\|\mathbf{r} \mathbf{v}_2\|$ is the second PC. We then subtract the variation explained by the first two PCs, and continue this process until we have the entire *rotation* matrix and matrix of principal components, respectively:

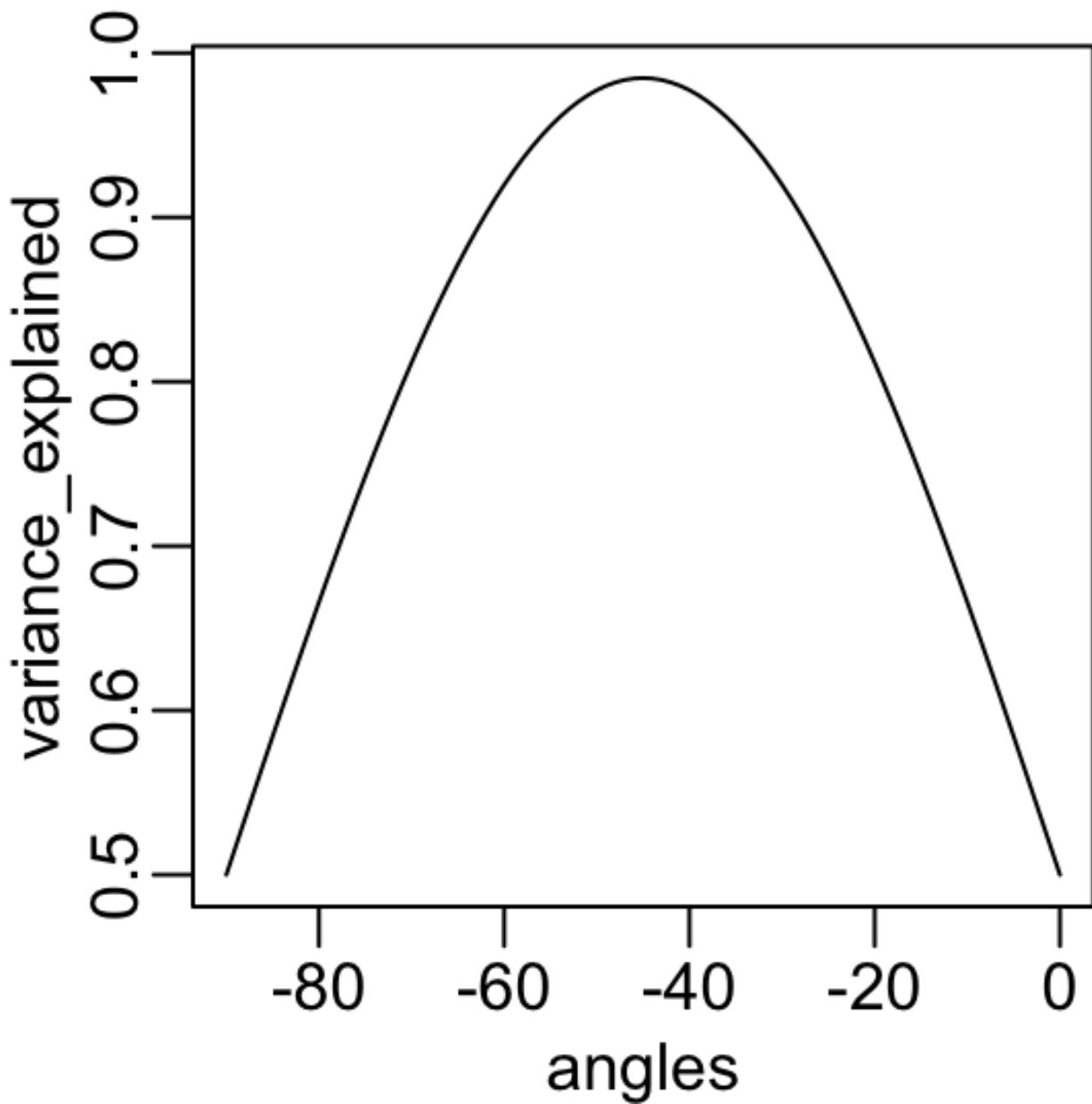
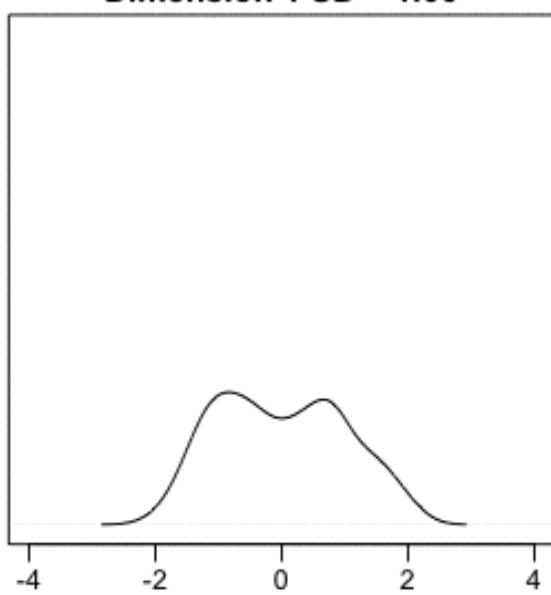
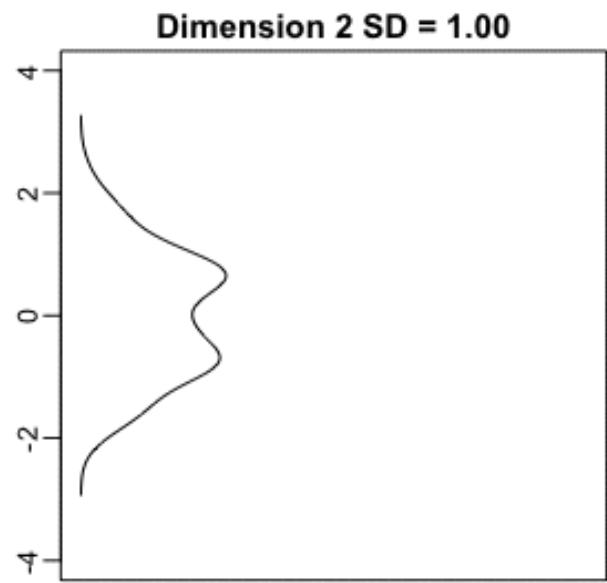
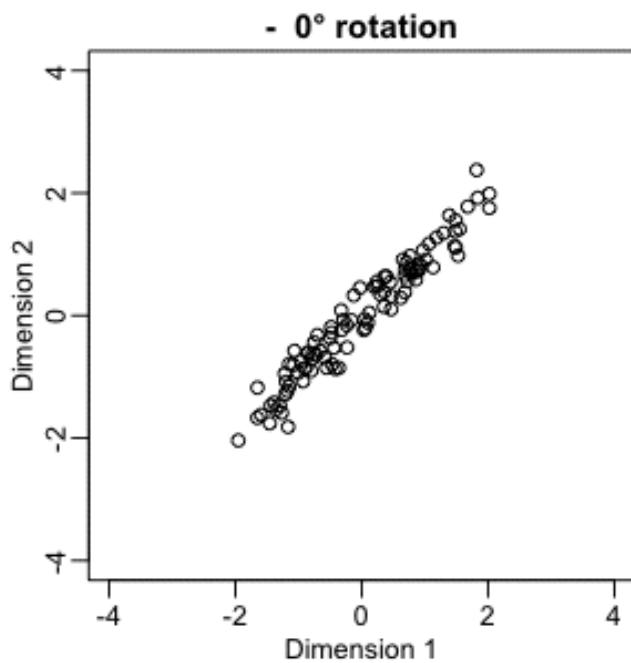


Figure 5:



Rotation matrix

$$\begin{pmatrix} \cos 0^\circ & -\sin 0^\circ \\ \sin 0^\circ & \cos 0^\circ \end{pmatrix}$$

Figure 6:

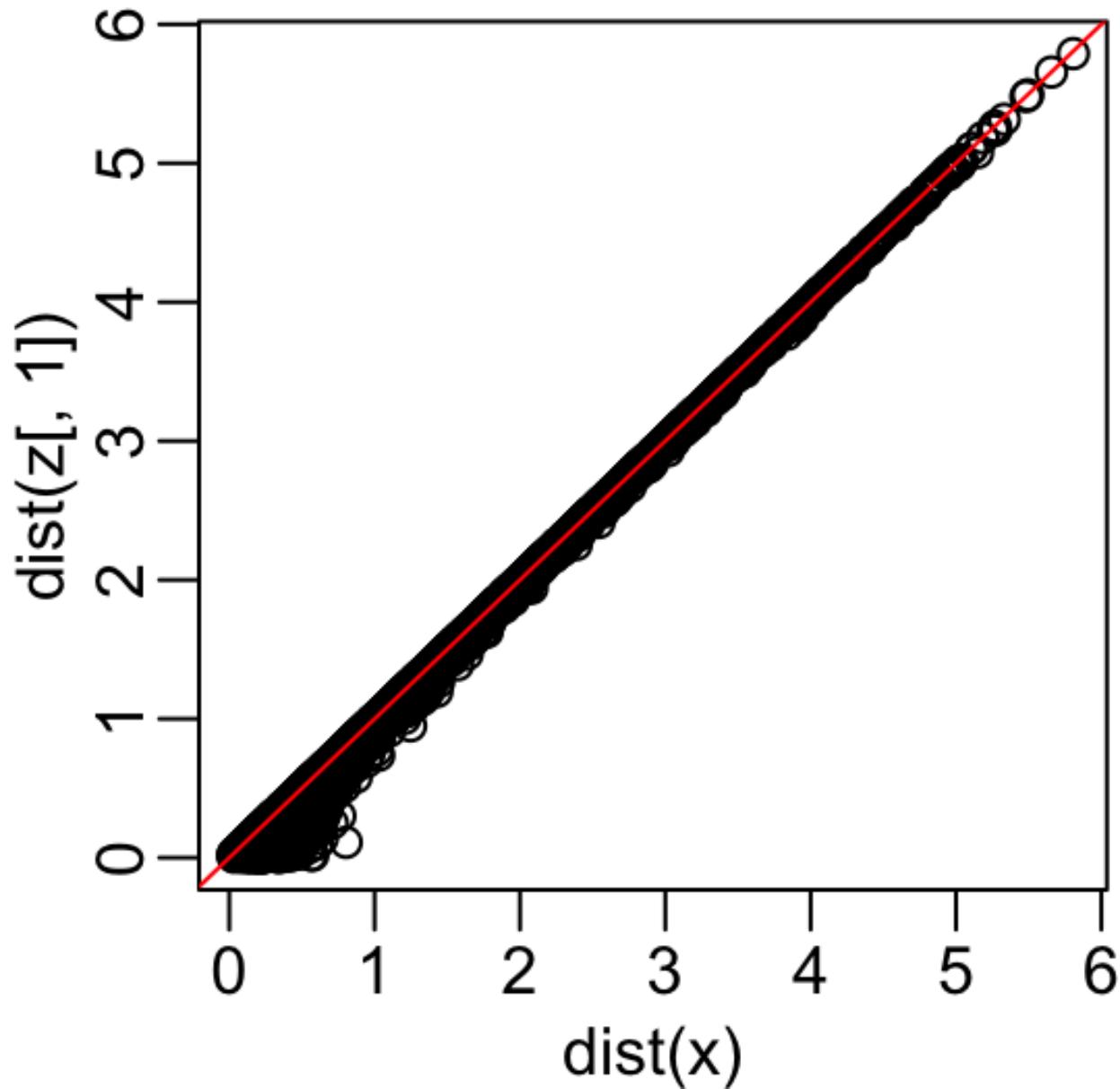


Figure 7:

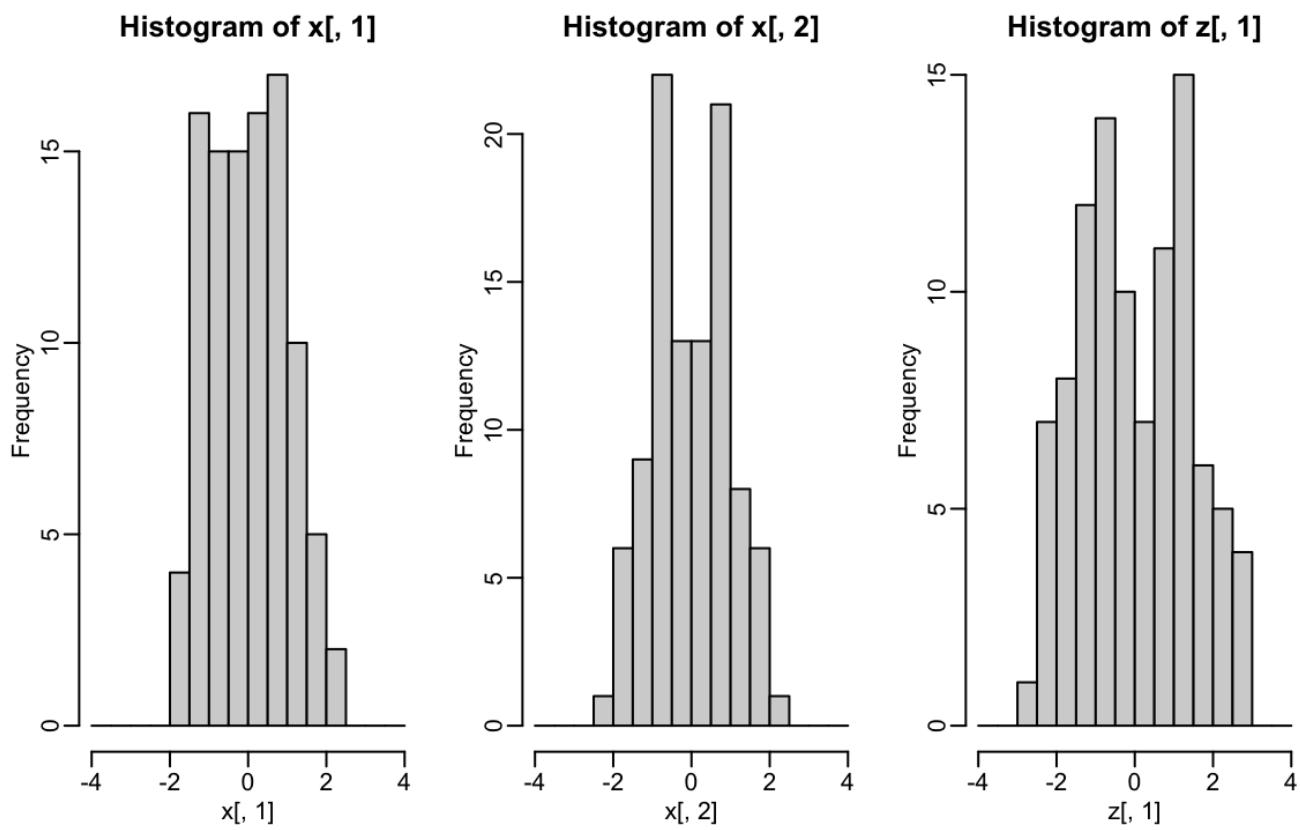


Figure 8:

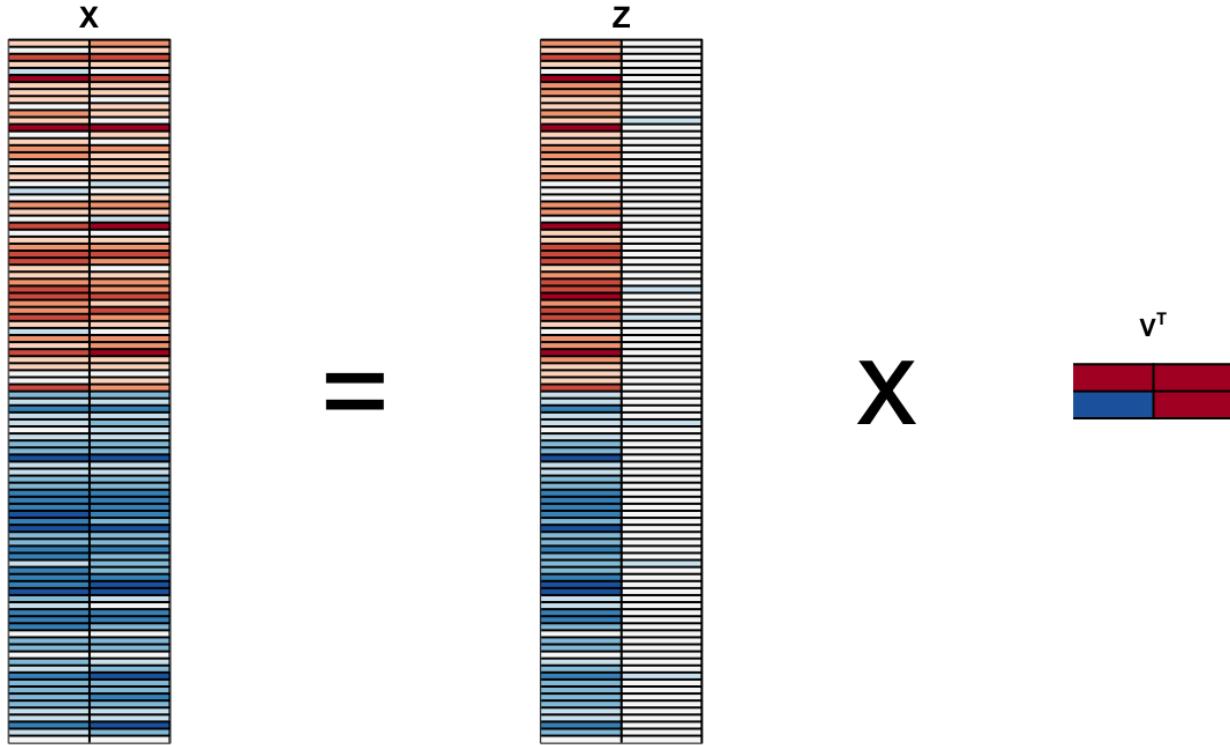


Figure 9:

```
\[ \mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \dots & \mathbf{v}_p \end{bmatrix}, \mathbf{Z} = \mathbf{X}\mathbf{V} \]
```

The idea of distance preservation extends to higher dimensions. For a multidimensional matrix with (p) columns, the (A) transformation preserves the distance between rows, but with the variance explained by the columns in decreasing order. If the variances of the columns (Z_j) , $(j>k)$ are very small, these dimensions have little to contribute to the distance calculation and we can approximate the distance between any two points with just (k) dimensions. If (k) is much smaller than (p) , then we can achieve a very efficient summary of our data.

Notice that the solution to this maximization problem is not unique because $(||\mathbf{X}\mathbf{v}|| = ||\mathbf{X}\mathbf{v}||)$. Also, note that if we multiply a column of (A) by (-1) , we still represent (\mathbf{X}) as $((\mathbf{Z}\mathbf{V})^\top)$ as long as we also multiply the corresponding column of (\mathbf{V}) by -1. This implies that we can arbitrarily change the sign of each column of the rotation (\mathbf{V}) and principal component matrix (\mathbf{Z}) .

In R, we can find the principal components of any matrix with the function `prcomp`:

```
pca <- prcomp(x, center = FALSE)
```

Keep in mind that default behavior is to center the columns of `x` before computing the PCs, an operation we don't need because our matrix is scaled.

The object `pca` includes the rotated data (Z) in `pca$x` and the rotation (\mathbf{V}) in `pca$rotation`.

We can see that columns of the `pca$rotation` are indeed the rotation obtained with -45 (remember the sign is arbitrary):

```
pca$rotation
#>          PC1      PC2
#> [1,] -0.707  0.707
#> [2,] -0.707 -0.707
```

The square root of the variation of each column is included in the `pca$sdev` component. This implies we can compute the variance explained by each PC using:

```
pca$sdev^2/sum(pca$sdev^2)
#> [1] 0.9848 0.0152
```

The function `summary` performs this calculation for us:

```
summary(pca)
#> Importance of components:
#>             PC1      PC2
#> Standard deviation    1.403 0.1745
#> Proportion of Variance 0.985 0.0152
#> Cumulative Proportion  0.985 1.0000
```

We also see that we can rotate `x` ((\mathbf{X})) and `pca$x` ((\mathbf{Z})) as explained with the mathematical formulas above:

```
all.equal(pca$x, x %*% pca$rotation)
#> [1] TRUE
all.equal(x, pca$x %*% t(pca$rotation))
#> [1] TRUE
```

22.6 Examples

22.6.1 Iris example

The iris data is a widely used example in data analysis courses. It includes four botanical measurements related to three flower species:

```
names(iris)
#> [1] "Sepal.Length" "Sepal.Width"   "Petal.Length"  "Petal.Width"
#> [5] "Species"
```

If you print `iris$Species`, you will see that the data is ordered by the species.

If we visualize the distances, we can clearly see the three species with one species very different from the other two:

```
x <- iris[,1:4] |> as.matrix()
d <- dist(x)
image(as.matrix(d), col = rev(RColorBrewer::brewer.pal(9, "RdBu")))
```

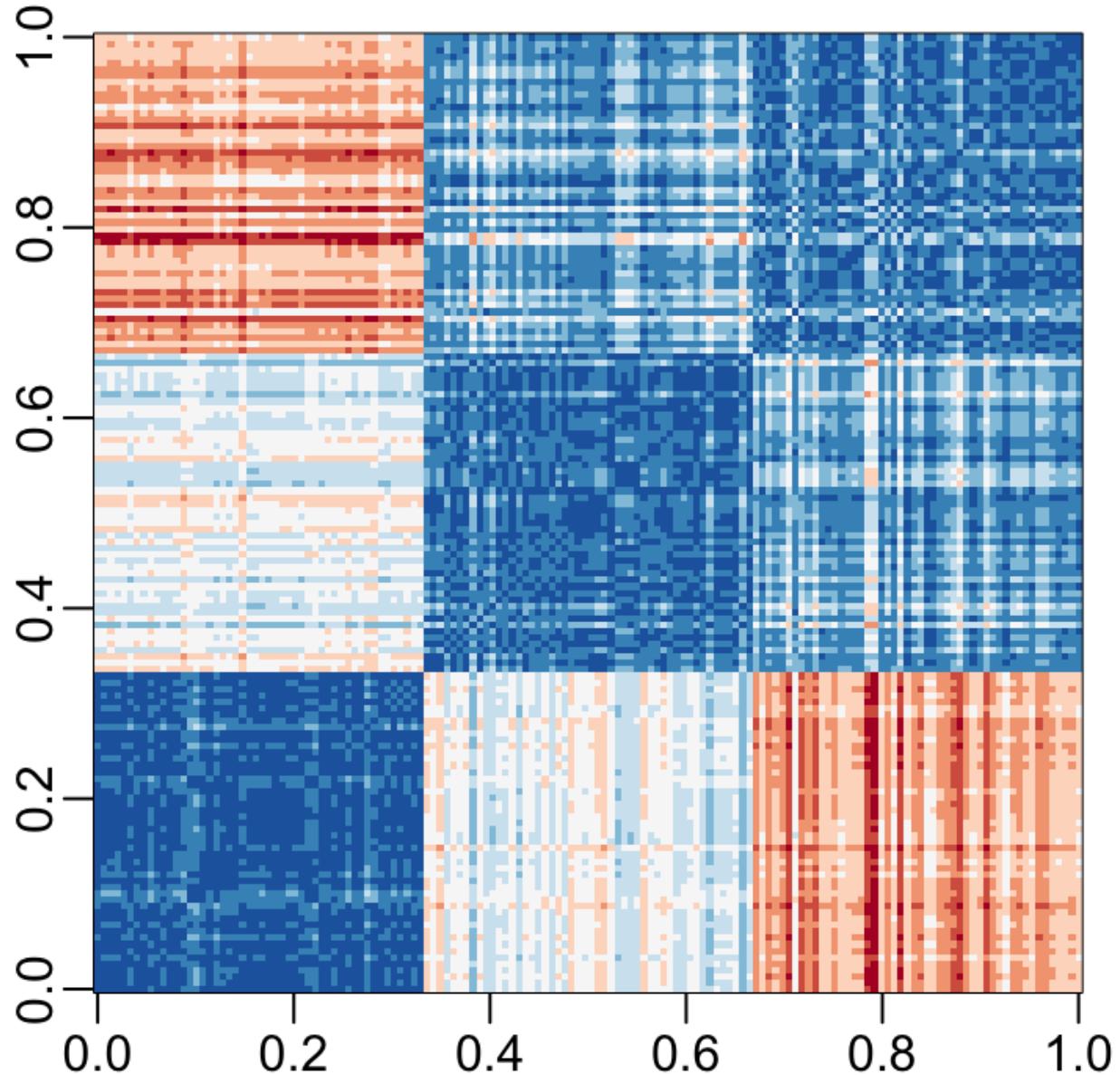


Figure 10:

Our features matrix has four dimensions, but three are very correlated:

```

cor(x)
#>           Sepal.Length Sepal.Width Petal.Length Petal.Width
#> Sepal.Length      1.000     -0.118      0.872      0.818
#> Sepal.Width       -0.118      1.000     -0.428     -0.366
#> Petal.Length      0.872     -0.428      1.000      0.963
#> Petal.Width       0.818     -0.366      0.963      1.000

```

If we apply PCA, we should be able to approximate this distance with just two dimensions, compressing the highly correlated dimensions. Using the `summary` function, we can see the variability explained by each PC:

```

pca <- prcomp(x)
summary(pca)
#> Importance of components:
#>                 PC1    PC2    PC3    PC4
#> Standard deviation   2.056 0.4926 0.2797 0.15439
#> Proportion of Variance 0.925 0.0531 0.0171 0.00521
#> Cumulative Proportion 0.925 0.9777 0.9948 1.00000

```

The first two dimensions account for almost 98% of the variability. Thus, we should be able to approximate the distance very well with two dimensions. We confirm this by computing the distance from first two dimensions and comparing to the original:

```

d_approx <- dist(pca$x[, 1:2])
plot(d, d_approx); abline(0, 1, col = "red")

```

A useful application of this result is that we can now visualize the distance between each observation with a two-dimensional plot.

```

data.frame(pca$x[, 1:2], Species = iris$Species) |>
  ggplot(aes(PC1, PC2, fill = Species)) +
  geom_point(cex = 3, pch = 21) +
  coord_fixed(ratio = 1)

```

We color the observations by their labels and notice that, with these two dimensions, we achieve almost perfect separation.

Looking more closely at the resulting PCs and rotations:

we learn that the first PC is obtained by taking a weighted average of sepal length, petal length, and petal width (red in first column), and subtracting a quantity proportional to sepal width (blue in first column). The second PC is a weighted average of petal length and petal width, minus a weighted average of sepal length and petal width.

22.6.2 MNIST example

The written digits example has 784 features. Is there any room for data reduction? We will use PCA to answer this.

If not already loaded, let's begin by loading the data:

```

library(dslabs)
if (!exists("mnist")) mnist <- read_mnist()

```

Because the pixels are so small, we expect pixels close to each other on the grid to be correlated, meaning that dimension reduction should be possible.

Let's compute the PCs. This will take a few seconds as it is a rather large matrix:

```

pca <- prcomp(mnist$train$images)
plot(pca$sdev^2/sum(pca$sdev^2), xlab = "PC", ylab = "Variance explained")

```

We can see that the first few PCs already explain a large percent of the variability.

Simply by looking at the first two PCs we already see information about the labels. Here is a random sample of 500 digits:

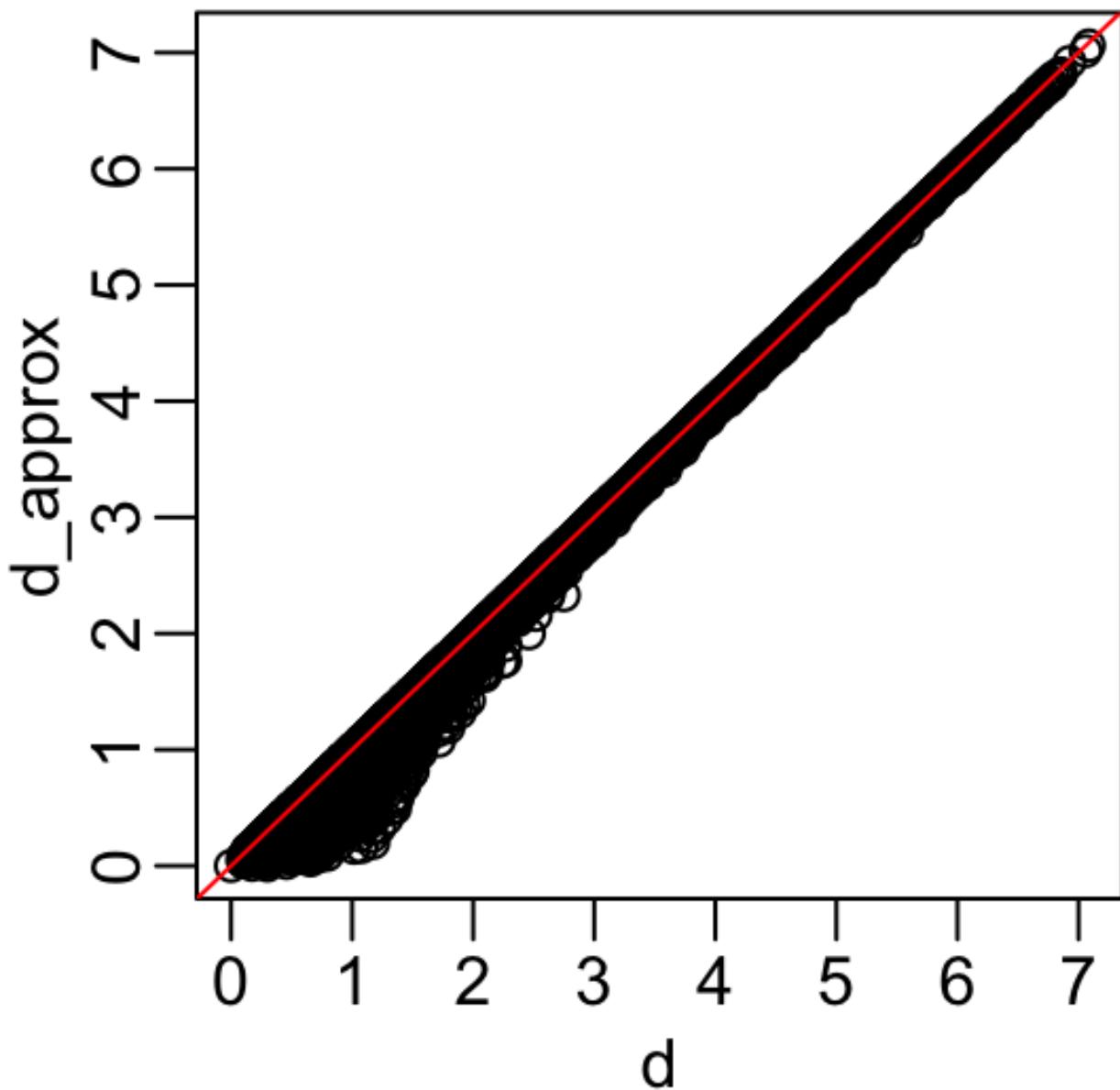


Figure 11:

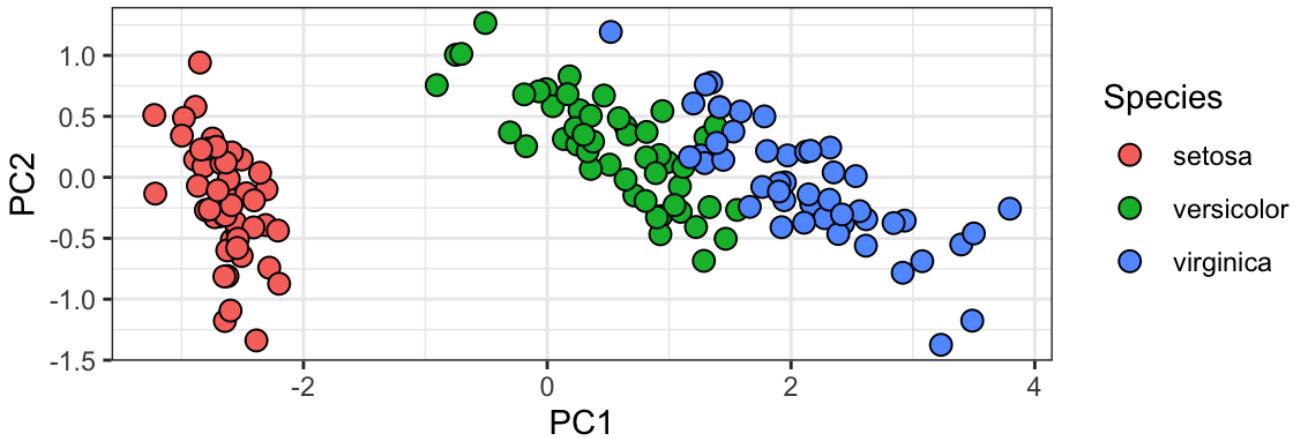


Figure 12:

We can also see the rotation values on the 28×28 grid to get an idea of how pixels are being weighted in the transformations that result in the PCs:

We can clearly see that first PC appears to be separating the 1s (red) from the 0s (blue). We can vaguely discern digits, or parts of digits, in the other three PCs as well. By looking at the PCs stratified by digits, we get further insights. For example, we see that the second PC separates 4s, 7s, and 9s from the rest:

We can also confirm that the lower variance PCs appear related to unimportant variability, mainly smudges in the corners:

22.7 Exercises

1. We want to explore the `tissue_gene_expression` predictors by plotting them.

```
dim(tissue_gene_expression$x)
```

We hope to get an idea of which observations are close to each other, but the predictors are 500-dimensional so plotting is difficult. Plot the first two principal components with color representing tissue type.

2. The predictors for each observation are measured on the same measurement device (a gene expression microarray) after an experimental procedure. A different device and procedure is used for each observation. This may introduce biases that affect all predictors for each observation in the same way. To explore the effect of this potential bias, for each observation, compute the average across all predictors and then plot this against the first PC with color representing tissue. Report the correlation.

3. We see an association with the first PC and the observation averages. Redo the PCA, but only after removing the center.
4. For the first 10 PCs, make a boxplot showing the values for each tissue.
5. Plot the percent variance explained by PC number. Hint: Use the `summary` function.

$$X = Z \times A^T$$

The diagram illustrates the matrix multiplication $X = Z \times A^T$. It features three main components: a tall matrix Z on the left, a short matrix A^T at the bottom right, and a tall matrix X on the far left. The matrices are composed of small colored squares. Matrix Z has a vertical column of blue squares on its left side. Matrix A^T is a 3x3 grid with colors: top-left (orange), top-right (light blue), middle-left (red), middle-right (light blue), bottom-left (orange), and bottom-right (dark red). Matrix X is tall and narrow, with a vertical column of blue squares on its left side, mirroring the structure of Z .

Figure 13:

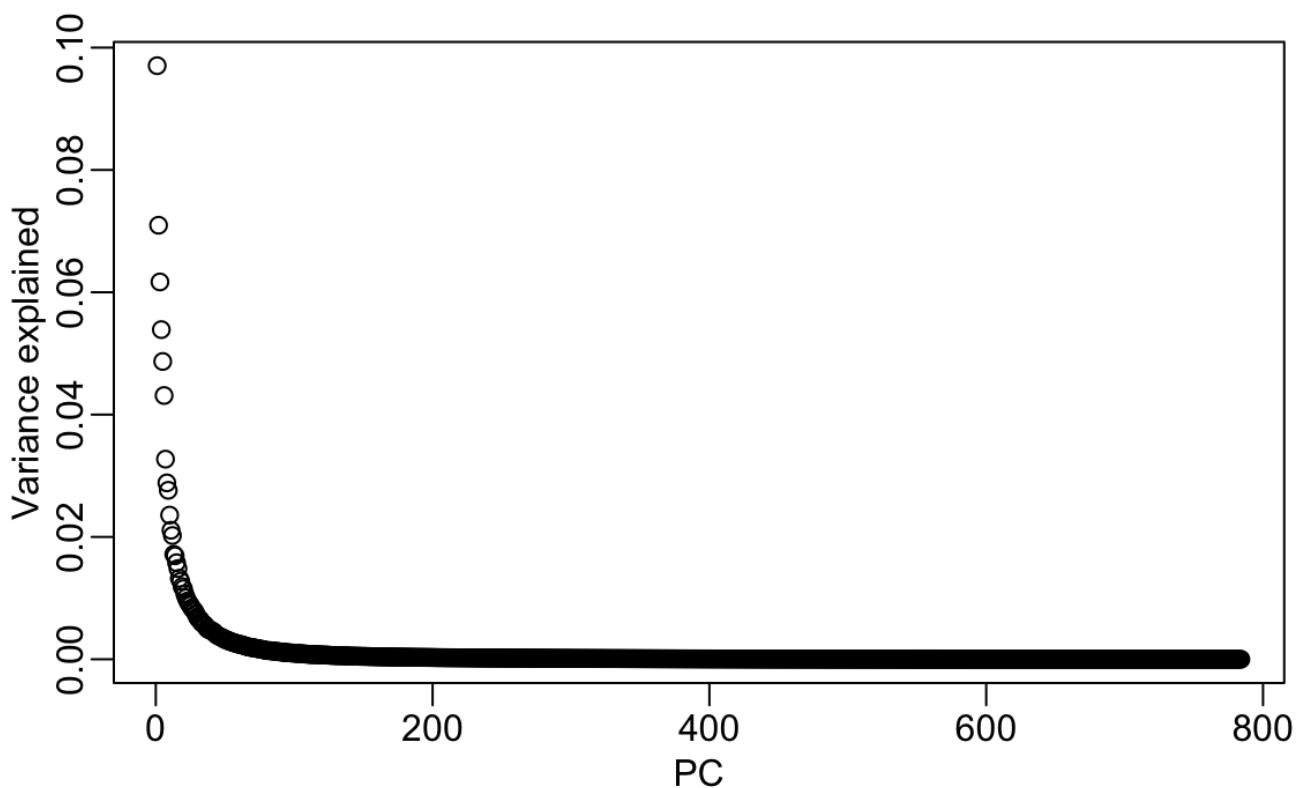


Figure 14:

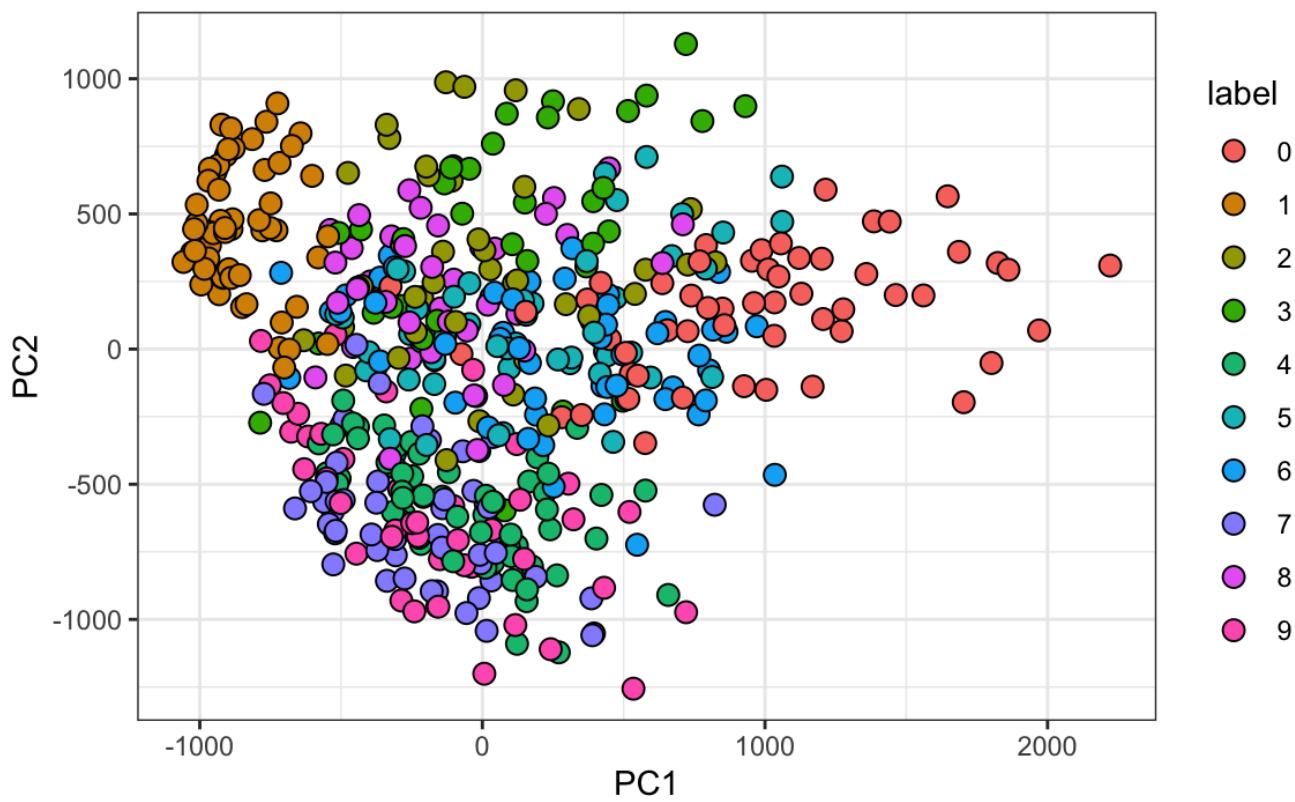


Figure 15:

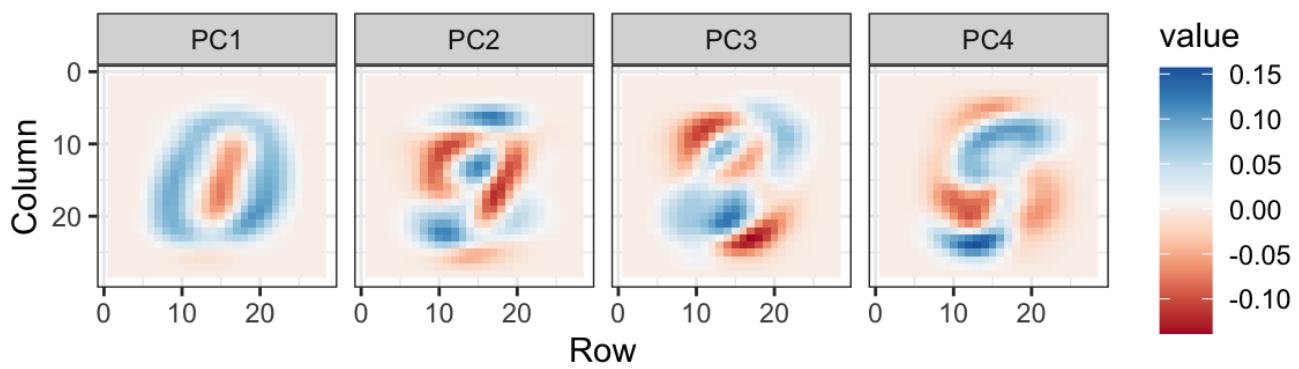


Figure 16:

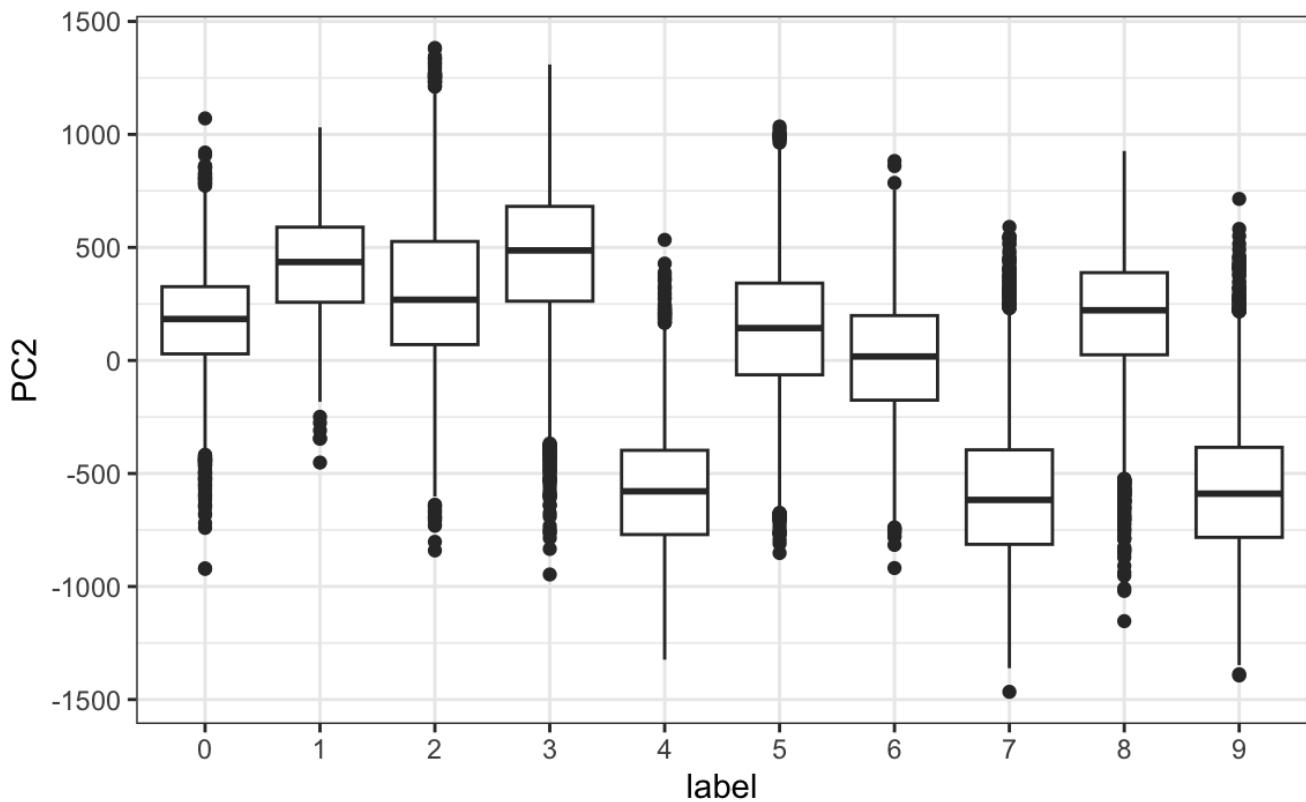


Figure 17:

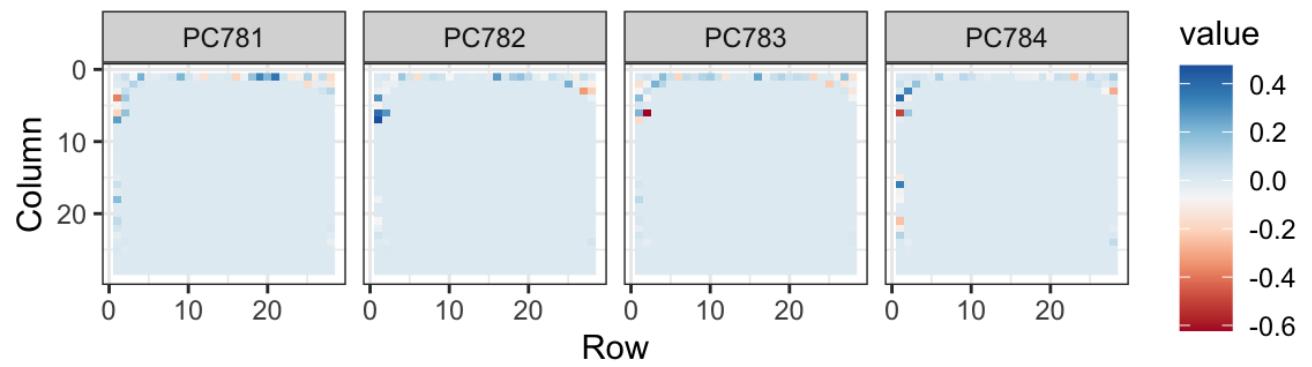


Figure 18:

Introduction to Data Science - 23 Regularization

Rafael A. Irizarry

23.1 Case study: recommendation systems

Recommendation systems, such as the one used by Amazon, operate by analyzing the ratings that customers give to various products. These ratings form a large dataset. The system uses this data to predict how likely a specific user is to favorably rate a particular product. For example, if the system predicts that a user is likely to give a high rating to a certain book or gadget, it will recommend that item to them. In essence, the system tries to guess which products a user will like based on the ratings provided by them and other customers for various items. This approach helps in personalizing recommendations to suit individual preferences.

During its initial years of operation, Netflix used a 5-star recommendation system. One star suggested it was not a good movie, whereas five stars suggested it was an excellent movie. Here, we provide the basics of how these recommendations are made, motivated by some of the approaches taken by the winners of the *Netflix challenges*.

In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars. In September 2009, the winners were announced¹. You can read a summary of how the winning algorithm was put together here: <http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/> and a more detailed explanation here: <https://www2.seas.gwu.edu/~simhaweb/champalg/cf/papers/KorenBellKor2009.pdf>. We will now show you some of the data analysis strategies used by the winning team.

23.1.1 MovieLens data

The Netflix data is not publicly available, but the GroupLens research lab² generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. We make a small subset of this data available via the **dslabs** package:

```
library(tidyverse)
library(janitor)
library(dslabs)
movielens |> as_tibble() |> head(5)
#> # A tibble: 5 × 7
#>   movieId title           year genres userId rating timestamp
#>   <int> <chr>        <int> <fct>  <int> <dbl>    <int>
#> 1     31 Dangerous Minds 1995 Drama      1     2.5  1.26e9
#> 2     1029 Dumbo         1941 Anima...    1     3  1.26e9
#> 3     1061 Sleepers      1996 Thril...    1     3  1.26e9
#> 4     1129 Escape from New York 1981 Actio...    1     2  1.26e9
#> 5     1172 Cinema Paradiso (Nuovo c... 1989 Drama      1     4  1.26e9
```

Each row represents a rating given by one user to one movie.

It will later be convenient that our **userId** and **movieId** are factors, so we change that:

```
movielens <- mutate(movielens, userId = factor(userId), movieId = factor(movieId))
```

We can see the number of unique users that provided ratings and how many unique movies were rated:

```
movielens |> summarize(n_distinct(userId), n_distinct(movieId))
#>   n_distinct(userId) n_distinct(movieId)
#> 1                 671            9066
```

If we multiply those two numbers, we get a number larger than 5 million, yet our data table has about 100,000 rows. This implies that not every user rated every movie. We can think of these data as a very large matrix, with users on the rows and movies on the columns, with many empty cells. Here is the matrix for six users and four movies:

userId	Pulp Fiction	Shawshank Redemption	Forrest Gump	Silence of the Lambs
13	3.5	4.5	5.0	NA
15	5.0	2.0	1.0	5.0
16	NA	4.0	NA	NA
17	5.0	5.0	2.5	4.5
19	5.0	4.0	5.0	3.0
20	0.5	4.5	2.0	0.5

You can think of the task of a recommendation system as filling in the NAs in the table above. To see how *sparse* the matrix is, here is the matrix for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating:

Let's look at some of the general properties of the data to better understand the challenges.

The first thing we notice is that some movies get rated more than others. Below is the distribution. This is not surprising given that there are blockbuster movies watched by millions and artsy, independent movies watched by just a few. Our second observation is that some users are more active than others at rating movies:

We need to build an algorithm with the collected data that will then be applied outside our control when users look for movie recommendations. To test our idea, we will split the data into a training set, which we will use to develop our approach, and a test set in which we will compute the accuracy of our predictions.

We will do this only for users that have provided at least 100 ratings.

```
movielens <- movielens |>
  group_by(userId) |>
  filter(n() >= 100) |>
  ungroup()
```

For each one of these users, we will split their ratings into 80% for training and 20% for testing.

```
set.seed(2006)
indexes <- split(1:nrow(movielens), movielens$userId)
test_ind <- sapply(indexes, function(i) sample(i, ceiling(length(i)*.2))) |>
  unlist() |>
  sort()
test_set <- movielens[test_ind,]
train_set <- movielens[-test_ind,]
```

To make sure we don't include movies in the training set that should not be there, we remove entries using the `semi_join` function:

```
test_set <- test_set |> semi_join(train_set, by = "movieId")
```

We will use the array representation described in Section 17.5, for the training data: we denote ranking for movie \((j)\) by user \((i)\) as \((y_{i,j})\). To create this matrix, we use `pivot_wider`:

```
y <- select(train_set, movieId, userId, rating) |>
  pivot_wider(names_from = movieId, values_from = rating) |>
  column_to_rownames("userId") |>
  as.matrix()
```

To be able to map movie IDs to titles we create the following lookup table:

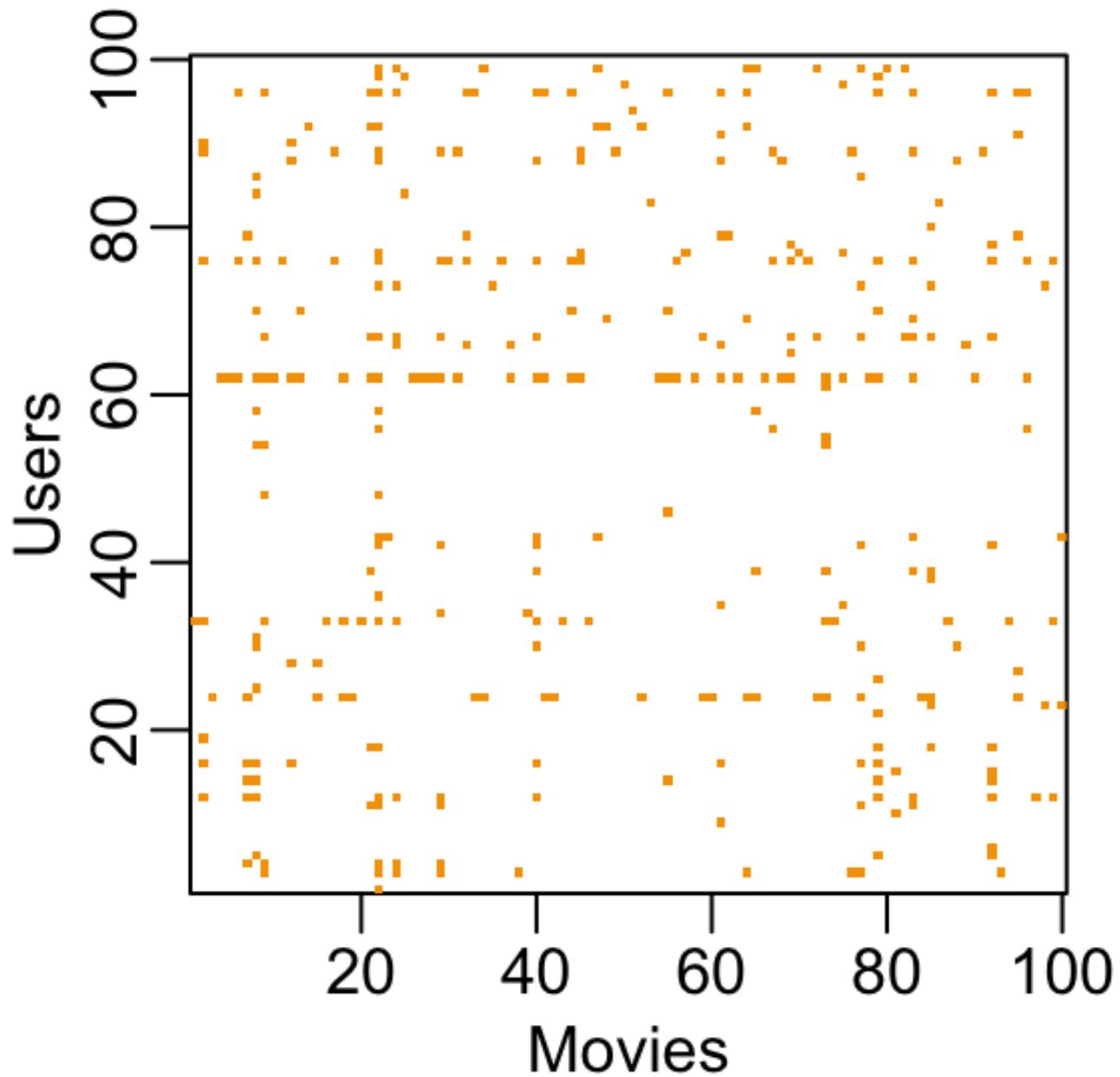


Figure 1:

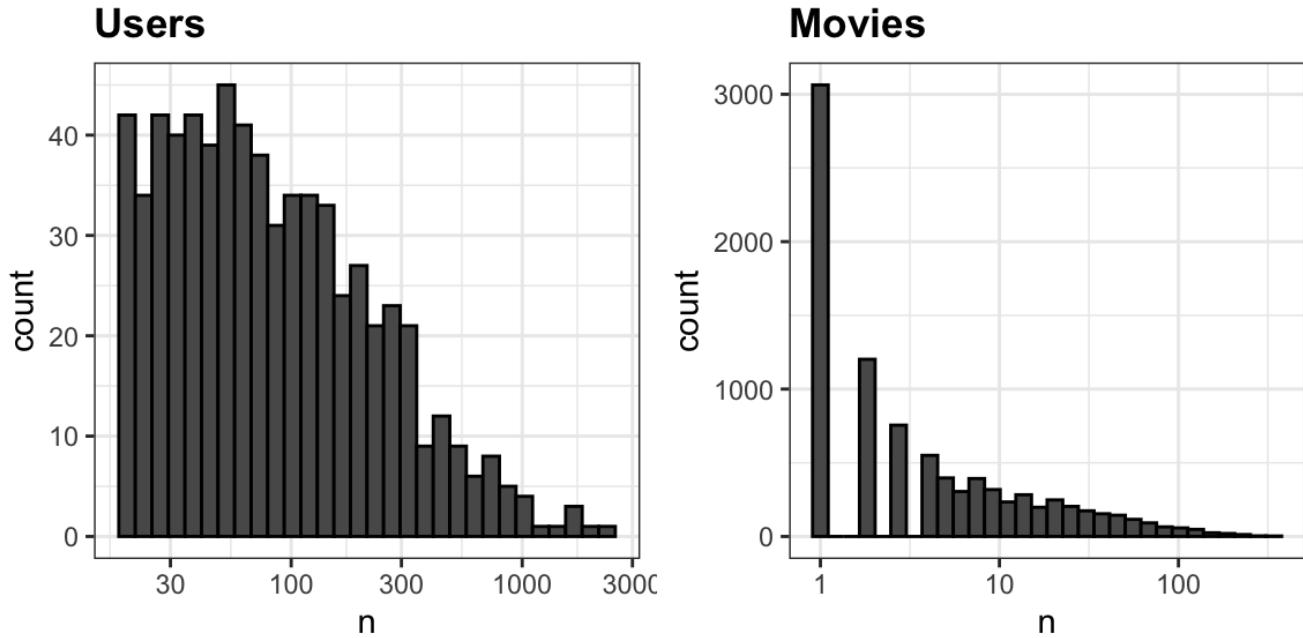


Figure 2:

```
movie_map <- train_set |> select(movieId, title) |> distinct(movieId, .keep_all = TRUE)
```

Note that two different movies can have the same title. For example, our dataset has three movies titled “King Kong”. Titles are therefore not unique and we can't use them as IDs.

23.2 Loss function

The Netflix challenge decided on a winner based on the root mean squared error (RMSE) computed on the test set. Specifically, if $y_{i,j}$ is the rating for movie j by user i in the test set and $\hat{y}_{i,j}$ is our prediction based on the training set, RMSE was defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i,j} (y_{i,j} - \hat{y}_{i,j})^2}$$

with N being the number of user/movie combinations for which we made predictions and the sum occurring over all these combinations.

We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good. We define a function to compute this quantity for any set of residuals:

```
rmse <- function(r) sqrt(mean(r^2))
```

In this chapter and the next, we introduce two concepts, regularization and matrix factorization, that were used by the winners of the Netflix challenge to obtain the winning RMSE.

In Chapter 29, we provide a formal discussion of the mean squared error.

23.3 A first model

Let's start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. What number should this prediction be? We can use a model based approach to answer this. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look as follows:

$$Y_{i,j} = \mu + \epsilon_{i,j}$$

with $\{\varepsilon_{i,j}\}$ independent errors sampled from the same distribution centered at 0 and $\{\mu\}$ the *true* rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of $\{\mu\}$ and, in this case, is the average of all ratings:

```
mu <- mean(y, na.rm = TRUE)
```

If we predict all unknown ratings with $\{\hat{\mu}\}$, we obtain the following RMSE:

```
rmse(test_set$rating - mu)
#> [1] 1.04
```

Keep in mind that if you plug in any other number, you get a higher RMSE. For example:

```
rmse(test_set$rating - 3)
#> [1] 1.16
```

To win the grand prize of \$1,000,000, a participating team had to get an RMSE of about 0.857. So we can definitely do better!

23.4 User effects

If we visualize the average rating for each user:

```
hist(rowMeans(y, na.rm = TRUE), nclass = 30)
```

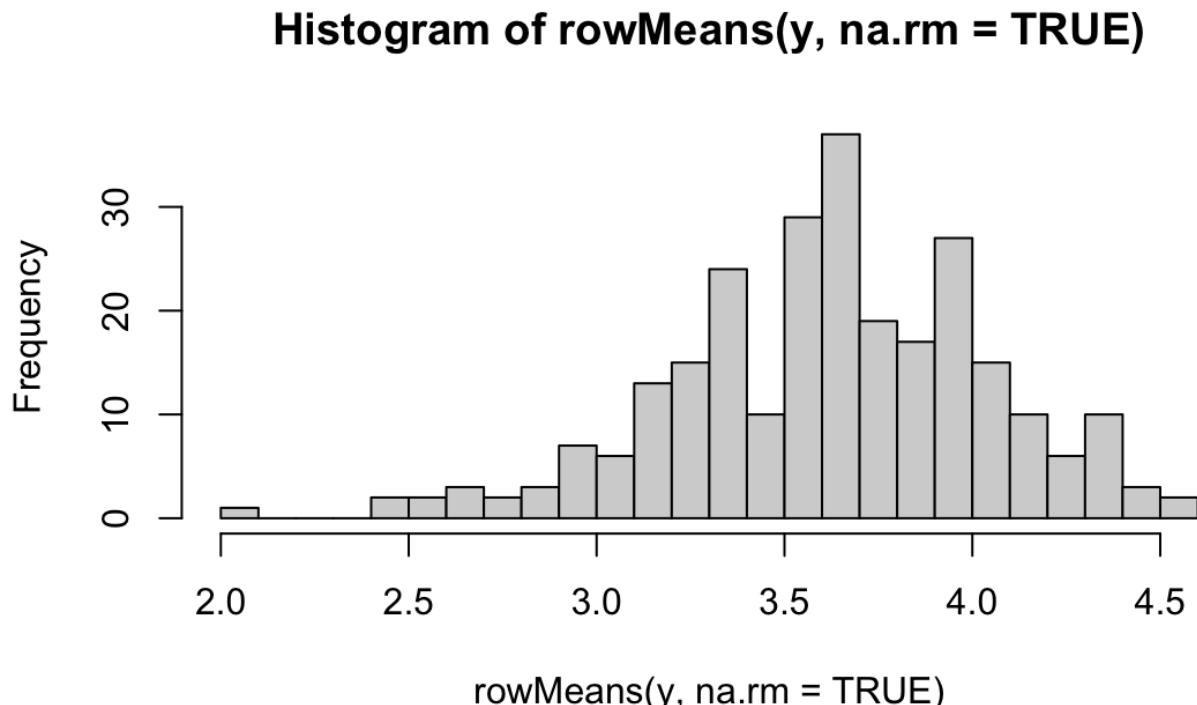


Figure 3:

we notice that there is substantial variability across users: some users are very cranky and others love most movies. To account for this, we can use a linear model with a *treatment effect* $\{\alpha_i\}$ for each user. The sum $\{\mu + \alpha_i\}$ can be interpreted as the typical rating user $\{i\}$ gives to movies. We can write the model as:

$$\{Y_{i,j}\} = \mu + \alpha_i + \varepsilon_{i,j}$$

Statistics textbooks refer to the $\{\alpha\}$ s as treatment effects. In the Netflix challenge papers, they refer to them as *bias*.

We can again use least squares to estimate the (α_i) in the following way:

```
fit <- lm(rating ~ userId, data = train_set)
```

Because there are hundreds of (α_i) , as each movie gets one, the `lm()` function will be very slow here. In this case, we can show that the least squares estimate $(\hat{\alpha}_i)$ is just the average of $(y_{i,j} - \hat{\mu})$ for each user (i) . So we can compute them this way:

```
a <- rowMeans(y - mu, na.rm = TRUE)
```

Note that going forward, we drop the `hat` notation in the code to represent estimates.

Let's see how much our prediction improves once we use $(\hat{y}_{i,j}) = \hat{\mu} + \hat{\alpha}_i$. Because we know ratings can't be below 0.5 or above 5, we define the function `clamp`:

```
clamp <- function(x, min = 0.5, max = 5) pmax(pmin(x, max), min)
```

to keep predictions in that range and then compute the RMSE:

```
test_set |>
  left_join(data.frame(userId = names(a), a = a), by = "userId") |>
  mutate(resid = rating - clamp(mu + a)) |> pull(resid) |> rmse()
#> [1] 0.958
```

We already see an improvement. But can we make it better?

23.5 Movie effects

We know from experience that some movies are just generally rated higher than others. We can use a linear model with a *treatment effect* (β_j) for each movie, which can be interpreted as movie effect or the difference between the average ranking for movie (j) and the overall average (μ) :

$$[Y_{i,j} = \mu + \alpha_i + \beta_j + \varepsilon_{i,j}]$$

We can again use least squares to estimate the (β_j) in the following way:

```
fit <- lm(rating ~ userId + movieId, data = train_set)
```

However, this code generates a very large matrix with all the indicator variables needed to represent all the movies and the code will take time to run. We instead use an approximation by first computing the least square estimate $(\hat{\mu})$ and $(\hat{\alpha}_i)$, and then estimating $(\hat{\beta}_j)$ as the average of the residuals $(y_{i,j} - \hat{\mu} - \hat{\alpha}_i)$:

```
b <- colMeans(y - mu - a, na.rm = TRUE)
```

We can now construct predictors and see how much the RMSE improves:

```
test_set |>
  left_join(data.frame(userId = names(a), a = a), by = "userId") |>
  left_join(data.frame(movieId = names(b), b = b), by = "movieId") |>
  mutate(resid = rating - clamp(mu + a + b)) |> pull(resid) |> rmse()
#> [1] 0.911
```

23.6 Penalized least squares

If we look at the top movies based on our estimates of the movie effect $(\hat{\beta}_j)$, we find that they all obscure movies with just one rating:

```
n <- colSums(!is.na(y))
ind <- which(b == max(b))
filter(movie_map, movieId %in% names(b)[ind]) |> pull(title)
#> [1] "Prisoner of the Mountains (Kavkazsky plennik)"
#> [2] "Dream With the Fishes"
#> [3] "Storefront Hitchcock"
#> [4] "Anatomy (Anatomie)"
```

```

#> [5] "Two Ninas"
#> [6] "Erik the Viking"
#> [7] "Grass Is Greener, The"
#> [8] "Caveman"
n[ind]
#> 1450 1563 1819 3892 4076 4591 4796 5427
#> 1 1 1 1 1 1 1 1

```

Do we really think these are the top movies in our database? The one of these that appears in our test set receives a terrible rating:

```

filter(test_set, movieId %in% names(b)[ind]) |>
  group_by(title, movieId) |>
  summarize(rating = mean(rating), .groups = "drop")
#> # A tibble: 1 × 3
#>   title           movieId rating
#>   <chr>          <dbl>    <dbl>
#> 1 Anatomy (Anatomie) 3892      1

```

Large estimates, negative or positive, should not be trusted when based on a small number of ratings. Because large errors can increase our RMSE, we would rather be conservative when unsure.

In previous sections, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one number, one prediction, not an interval. For this, we introduce the concept of regularization.

Regularization permits us to penalize large estimates that are formed using small sample sizes. It has commonalities with the Bayesian approach that shrunk predictions described in Chapter 12.

The general idea behind regularization is to constrain the total variability of the effect sizes. Why does this help? Consider a case in which we have movie $(j=1)$ with 100 user ratings and 4 movies $(j=2,3,4,5)$ with just one user rating. Suppose we know the average rating is, say, $(\mu = 3)$. If we use least squares, the estimate for the first movie effect is the average of 100 user ratings, which we expect to be quite precise. However, the estimate for movies 2, 3, 4, and 5 will be based on one observation. Note that because the average is based on a single observation, the error for $(j=2,3,4,5)$ is 0, but we don't expect to be this lucky next time, when asked to predict. In fact, ignoring the one user and guessing that movies 2,3,4, and 5 are just average movies might provide a better prediction. The general idea of penalized regression is to control the total variability of the movie effects: $(\sum_{j=1}^5 \beta_j^2)$. Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

$$[\sum_{i,j} (y_{u,i} - \mu - \alpha_i - \beta_j)^2 + \lambda \sum_j \beta_j^2]$$
 The first term is just the sum of squares and the second is a penalty that gets larger when many (β_j) s are large. Using calculus, we can actually show that the values of (β_j) that minimize this equation are:

$$[\hat{\beta}_j(\lambda) = \frac{1}{\lambda + n_j} \sum_{i=1}^{n_j} (Y_{i,j} - \mu - \alpha_i)]$$

where (n_j) is the number of ratings made for movie (j) .

When we estimate the parameters of a linear model with penalized least squares, we refer to the approach as *ridge regression*. The `lm.ridge` function in the **MASS** package can perform the estimation. We don't use it here due to the large numbers of parameters associated with movie effects.

This approach will have our desired effect: when our sample size (n_j) is very large, we obtain a stable estimate and the penalty (λ) is effectively ignored since $(n_j + \lambda \approx n_j)$. Yet when the (n_j) is small, then the estimate $(\hat{\beta}_j(\lambda))$ is shrunk towards 0. The larger the (λ) , the more we shrink.

But how do we select (λ) ? In Chapter 29, we describe an approach to do this. Here we will simply compute the RMSE for different values of (λ) to illustrate the effect:

```

n <- colSums(!is.na(y))
sums <- colSums(y - mu - a, na.rm = TRUE)
lambda <- seq(0, 10, 0.1)
rmses <- sapply(lambda, function(lambda){
  b <- sums / (n + lambda)
  test_set |>
    left_join(data.frame(userId = names(a), a = a), by = "userId") |>
    left_join(data.frame(movieId = names(b), b = b), by = "movieId") |>
    mutate(resid = rating - clamp(mu + a + b)) |> pull(resid) |> rmse()
})

```

Here is a plot of the RMSE versus λ :

```
plot(lambda, rmses, type = "l")
```

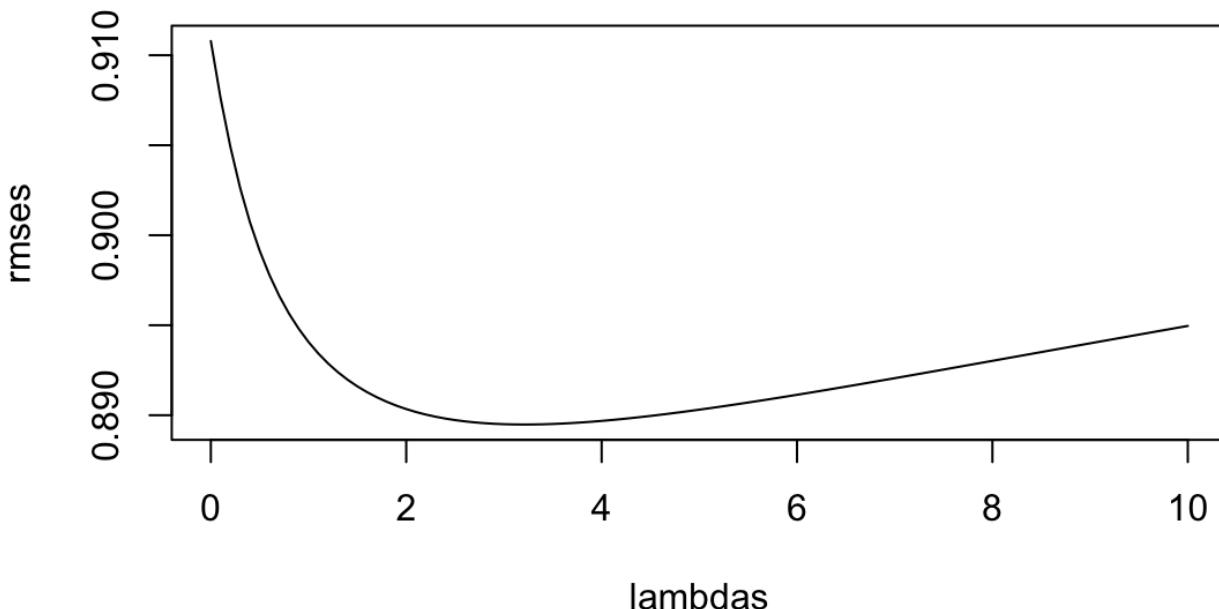


Figure 4:

The minimum is obtained for $\lambda = 3.2$

Using this λ , we can compute the regularized estimates and add to our table of estimates:

```

lambda <- lambda[which.min(rmses)]
b_reg <- sums / (n + lambda)

```

To see how the estimates shrink, let's make a plot of the regularized estimates versus the least squares estimates.

Now, let's look at the top 5 best movies based on the penalized estimates $\hat{b}_i(\lambda)$:

```

#> # A tibble: 10 × 5
#>   title                  year rating b_reg      n
#>   <chr>                 <int>  <dbl> <dbl> <dbl>
#> 1 Shawshank Redemption, The 1994    4.57  0.793   138
#> 2 Chinatown                1974    4.5    0.734    51

```

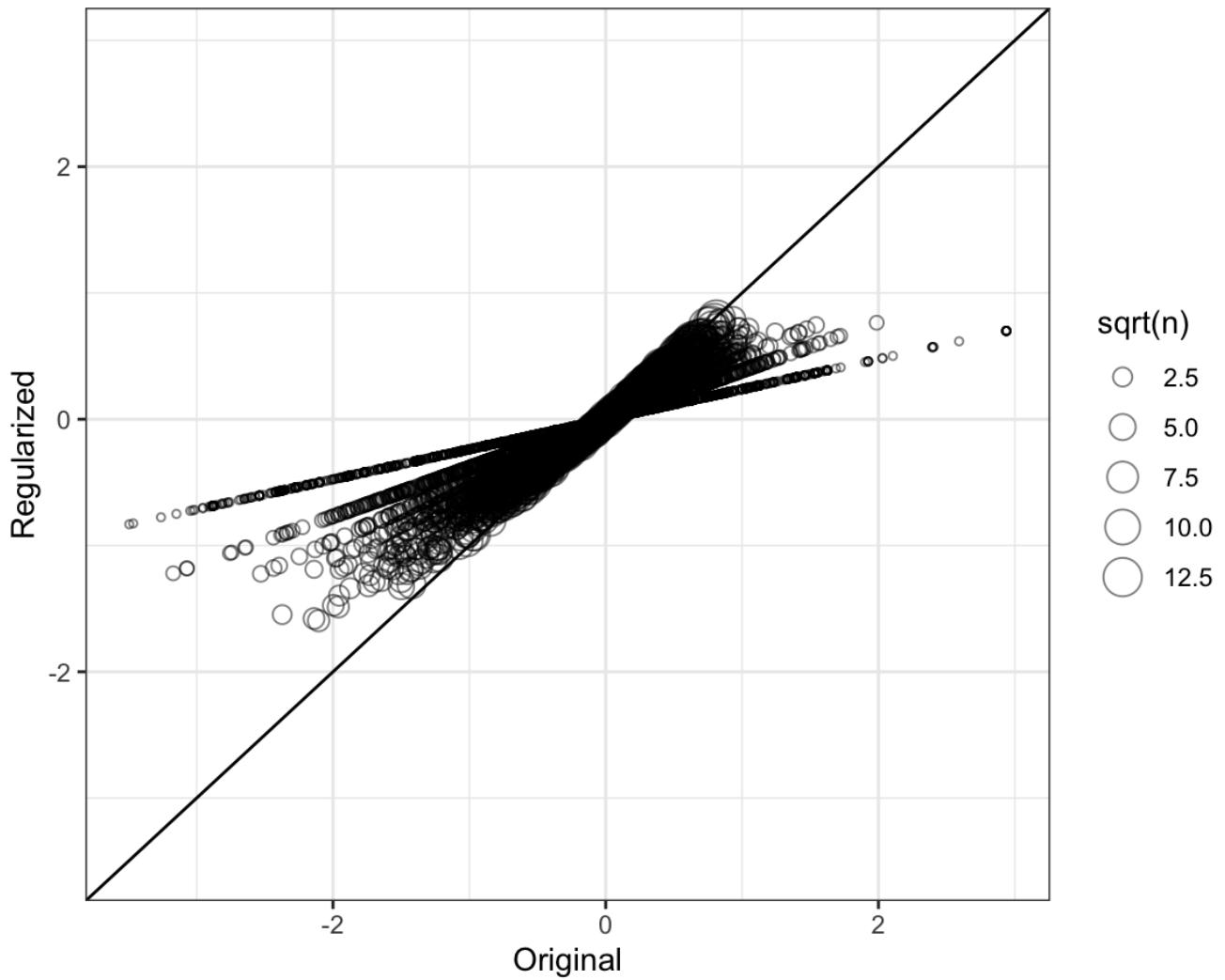


Figure 5:

```
#> 3 Dangerous Beauty      1998  4.5  0.764     2
#> 4 Godfather, The       1972  4.45 0.774    107
#> 5 Usual Suspects, The  1995  4.40 0.752    109
#> #  5 more rows
```

These make more sense with some movies that are watched more and have more ratings in the training set.

Notice *Swinger* has a lower rating than the other top 10, yet a large movie effect estimate. This is due to the fact that it was rated by harsher users.

Note that regularization improves our RMSE:

```
test_set |>
  left_join(data.frame(userId = names(a), a = a), by = "userId") |>
  left_join(data.frame(movieId = names(b_reg), b_reg = b_reg), by = "movieId") |>
  mutate(resid = rating - clamp(mu + a + b_reg)) |> pull(resid) |> rmse()
#> [1] 0.889
```

The penalized estimates provide an improvement over the least squares estimates:

model	RMSE
Just the mean	1.043
User effect	0.958
User + movie effect	0.911
User + regularized movie effect	0.889

23.7 Exercises

1. For the `movielens` data, compute the number of ratings for each movie and then plot it against the year the movie was released. Use the square root transformation on the counts.

2. We see that, on average, movies that were released after 1993 get more ratings. We also see that with newer movies, starting in 1993, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it.

Among movies that came out in 1993 or later, what are the 25 movies with the most ratings per year? Also, report their average rating.

3. From the table constructed in the previous example, we see that the most rated movies tend to have above average ratings. This is not surprising: more people watch popular movies. To confirm this, stratify the post 1993 movies by ratings per year and compute their average ratings. Make a plot of average rating versus ratings per year and show an estimate of the trend.

4. In the previous exercise, we see that the more a movie is rated, the higher the rating. Suppose you are doing a predictive analysis in which you need to fill in the missing ratings with some value. Which of the following strategies would you use?

- a. Fill in the missing values with average rating of all movies.
- b. Fill in the missing values with 0.
- c. Fill in the value with a lower value than the average since lack of rating is associated with lower ratings. Try out different values and evaluate prediction in a test set.
- d. None of the above.

5. The `movielens` dataset also includes a time stamp. This variable represents the time and date in which the rating was provided. The units are seconds since January 1, 1970. Create a new column `date` with the date. Hint: Use the `as_datetime` function in the `lubridate` package.

6. Compute the average rating for each week and plot this average against day. Hint: Use the `round_date` function before you `group_by`.

7. The plot shows some evidence of a time effect. If we define $(d_{u,i})$ as the day for user's (u) rating of movie (i) , which of the following models is most appropriate:

- a. $(Y_{u,i}) = \mu + b_i + \beta_j + d_{u,i} + \epsilon_{u,i}$.
- b. $(Y_{u,i}) = \mu + b_i + \beta_j + d_{u,i}\beta + \epsilon_{u,i}$.
- c. $(Y_{u,i}) = \mu + b_i + \beta_j + d_{u,i}\beta_i + \epsilon_{u,i}$.
- d. $(Y_{u,i}) = \mu + b_i + \beta_j + f(d_{u,i}) + \epsilon_{u,i}$, with f a smooth function of $(d_{u,i})$.

8. The `movielens` data also has a `genres` column. This column includes every genre that applies to the movie. Some movies fall under several genres. Define a category as whatever combination appears in this column. Keep only categories with more than 1,000 ratings. Then compute the average and standard error for each category. Plot these as error bar plots.

9. The plot shows strong evidence of a genre effect. If we define $(g_{u,i})$ as the genre for user's (u) rating of movie (i) , which of the following models is most appropriate:

- a. $(Y_{u,i}) = \mu + b_i + \beta_j + d_{u,i} + \epsilon_{u,i}$.
- b. $(Y_{u,i}) = \mu + b_i + \beta_j + d_{u,i}\beta + \epsilon_{u,i}$.
- c. $(Y_{u,i}) = \mu + b_i + \beta_j + \sum_{k=1}^K x_{u,i} \beta_k + \epsilon_{u,i}$, with $x_{u,i} = 1$ if $(g_{u,i})$ is genre (k) .
- d. $(Y_{u,i}) = \mu + b_i + \beta_j + f(d_{u,i}) + \epsilon_{u,i}$, with f a smooth function of $(d_{u,i})$.

An education expert is advocating for smaller schools. The expert bases this recommendation on the fact that among the best performing schools, many are small schools. Let's simulate a dataset for 100 schools. First, let's simulate the number of students in each school.

```
set.seed(1986)
n <- round(2^rnorm(1000, 8, 1))
```

Now let's assign a *true* quality for each school completely independent from size. This is the parameter we want to estimate.

```
mu <- round(80 + 2 * rt(1000, 5))
range(mu)
schools <- data.frame(id = paste("PS", 1:100),
                       size = n,
                       quality = mu,
                       rank = rank(-mu))
```

We can see that the top 10 schools are:

```
schools |> top_n(10, quality) |> arrange(desc(quality))
```

Now let's have the students in the school take a test. There is random variability in test taking so we will simulate the test scores as normally distributed with the average determined by the school quality and standard deviations of 30 percentage points:

```
scores <- sapply(1:nrow(schools), function(i){
  scores <- rnorm(schools$size[i], schools$quality[i], 30)
  scores
})
schools <- schools |> mutate(score = sapply(scores, mean))
```

10. What are the top schools based on the average score? Show just the ID, size, and the average score.
11. Compare the median school size to the median school size of the top 10 schools based on the score.
12. According to this test, it appears small schools are better than large schools. Five out of the top 10 schools have 100 or fewer students. But how can this be? We constructed the simulation so that quality and size are independent. Repeat the exercise for the worst 10 schools.

13. The same is true for the worst schools! They are small as well. Plot the average score versus school size to see what's going on. Highlight the top 10 schools based on the *true* quality. Use the log scale transform for the size.
14. We can see that the standard error of the score has larger variability when the school is smaller. This is a basic statistical reality we learned in the probability and inference sections. In fact, note that 4 of the top 10 schools are in the top 10 schools based on the exam score.

Let's use regularization to pick the best schools. Remember regularization *shrinks* deviations from the average towards 0. So to apply regularization here, we first need to define the overall average for all schools:

```
overall <- mean(sapply(scores, mean))
```

and then define, for each school, how it deviates from that average. Write code that estimates the score above average for each school, but dividing by $(n + \lambda)$ instead of (n) , with (n) the school size and (λ) a regularization parameter. Try $(\lambda = 3)$.

15. Notice that this improves things a bit. The number of small schools that are not highly ranked is now 4. Is there a better (λ) ? Find the (λ) that minimizes the $\text{RMSE} = \sqrt{(1/100 \sum_{i=1}^{100} (\text{quality}_i - \text{estimate}_i)^2)}$.
 16. Rank the schools based on the average obtained with the best (α) . Note that no small school is incorrectly included.
 17. A common mistake to make when using regularization is shrinking values towards 0 that are not centered around 0. For example, if we don't subtract the overall average before shrinking, we actually obtain a very similar result. Confirm this by re-running the code from exercise 6, but without removing the overall mean.
-

1. <http://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/>
2. <https://grouplens.org/>

Introduction to Data Science - 24 Matrix Factorization

Rafael A. Irizarry

In the previous chapter, we described how the model:

$$Y_{i,j} = \mu + \alpha_i + \beta_j + \epsilon_{i,j}$$

can be used to model movie ratings and help make useful predictions. This model accounts for user differences through α_i and movie effects through β_j . However, the model ignores an important source of information related to the fact that groups of movies, have similar rating patterns and groups of users have similar rating patterns as well.

To see an example of this, we compute residuals:

$r_{i,j} = y_{i,j} - (\hat{\mu} + \hat{\alpha}_i + \hat{\beta}_j)$ using the `mu`, `a` and `b_reg` computed in the previous chapter:

```
r <- sweep(y - mu - a, 2, b_reg)
```

and see how the residuals for three different movies correlate with *The Godfather*:

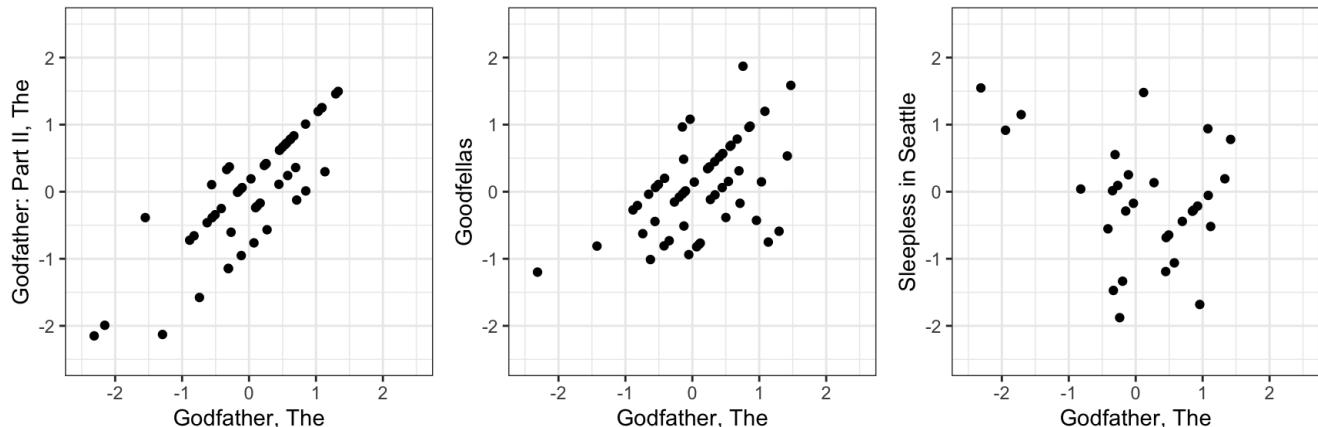


Figure 1:

We see that the correlation varies from very strong to negative across the other three movies.

In this chapter, we introduce Factor Analysis, an approach that permits us to model these correlations and improve our prediction, and the Singular Value Decomposition, which permits us to fit the model. As we will see, this approach is related to principal component analysis (PCA). We describe these concepts in the context of movie recommendation systems.

24.1 Factor analysis

We start with a simple illustration. We simulate $\epsilon_{i,j}$ for 6 movies and 120 users and save it in `e`. If we examine the correlation, we notice a pattern:

```

cor(e)
#>                                              Godfather Godfather 2 Goodfellas Scent of a Woman
#> Godfather                               1.000      0.671      0.558      -0.527
#> Godfather 2                             0.671      1.000      0.471      -0.450
#> Goodfellas                            0.558      0.471      1.000      -0.888
#> Scent of a Woman                      -0.527     -0.450     -0.888      1.000
#> You've Got Mail                      -0.734     -0.649     -0.487      0.451
#> Sleepless in Seattle                  -0.721     -0.739     -0.505      0.475
#>                                         You've Got Mail Sleepless in Seattle
#> Godfather                               -0.734      -0.721
#> Godfather 2                            -0.649      -0.739
#> Goodfellas                            -0.487      -0.505
#> Scent of a Woman                      0.451       0.475
#> You've Got Mail                      1.000       0.756
#> Sleepless in Seattle                  0.756       1.000

```

It seems there is positive correlation within mob and romance movies, and negative across the two genres. In statistics, we define *factors* as unobserved or *latent* variables that are inferred from the patterns of correlations or associations between the observed variables. We can quantify a factor that distinguishes between mob and romance movies with:

```
q <- c(-1, -1, -1, 1, 1, 1)
```

To determine which users prefer each genre, we can fit a linear model to each user:

```
p <- apply(e, 1, function(y) lm(y~q-1)$coef)
```

Notice we use the `-1` because the errors have mean 0 and we don't need an intercept.

There is a much faster way to make this computation using linear algebra. This is because the `lm` function is computing the least squares estimates by taking the derivative of the sum of squares, equaling it to 0, and noting the solution $\hat{\boldsymbol{\beta}}$ satisfies:

$\nabla_{\boldsymbol{\beta}} \sum_{i=1}^n (\mathbf{y}_i - \mathbf{q}^\top \mathbf{x}_i)^2 = \mathbf{q}^\top \mathbf{y}$ with \mathbf{y} the row of \mathbf{e} passed to y in the `apply` function. Because \mathbf{q} does not change for each user, rather than have `lm` recompute the equation for each user, we can perform the calculation on each column of \mathbf{e} to get the $\boldsymbol{\beta}_j$ for all users j like this:

```
p <- t(qr.solve(crossprod(q)) %*% t(q) %*% t(e))
```

The histogram below shows there are three type of users: those that love mob movies and hate romance movies, those that don't care, and those that love romance movies and hate mob movies.

```
hist(p, breaks = seq(-2,2,0.1))
```

To see that we can approximate \mathbf{p}_{iq} with $\mathbf{p}_{iq}\mathbf{q}_j$ we convert the vectors to matrices and use linear algebra:

```
p <- matrix(p); q <- matrix(q)
plot(p %*% t(q), e)
```

However, after removing this mob/romance effect, we still see structure in the correlation:

```

cor(e - p %*% t(q))
#>                                              Godfather Godfather 2 Goodfellas Scent of a Woman
#> Godfather                               1.000      0.185      -0.545      0.557
#> Godfather 2                            0.185      1.000      -0.618      0.594
#> Goodfellas                            -0.545     -0.618      1.000      -0.671
#> Scent of a Woman                      0.557      0.594      -0.671      1.000
#> You've Got Mail                      -0.280     -0.186      0.619      -0.641
#> Sleepless in Seattle                  -0.198     -0.364      0.650      -0.656
#>                                         You've Got Mail Sleepless in Seattle

```

Histogram of p

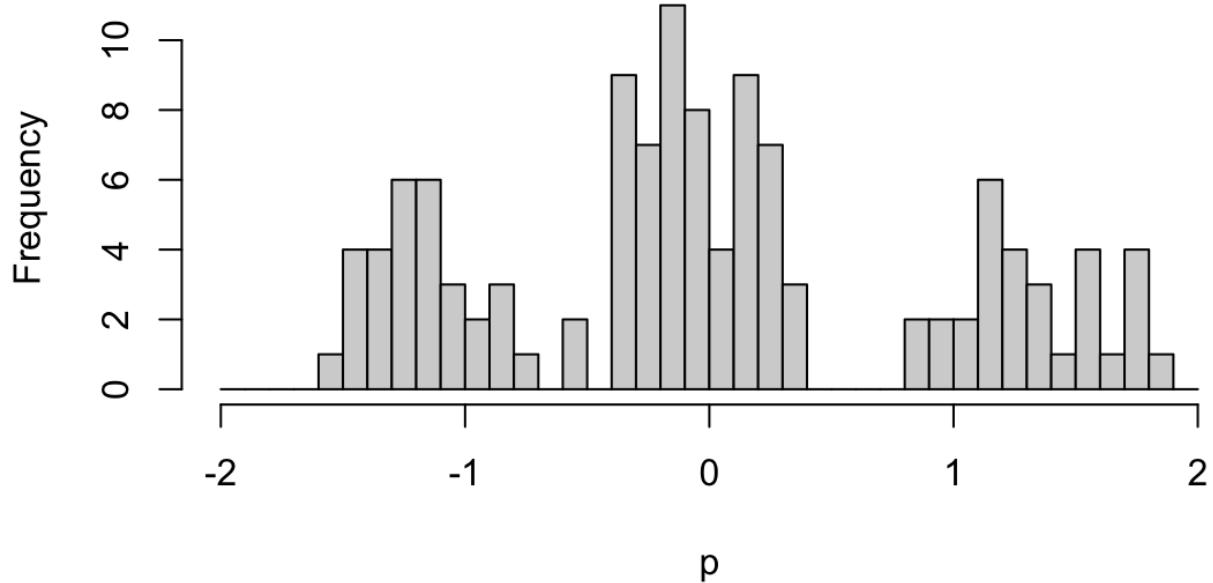


Figure 2:

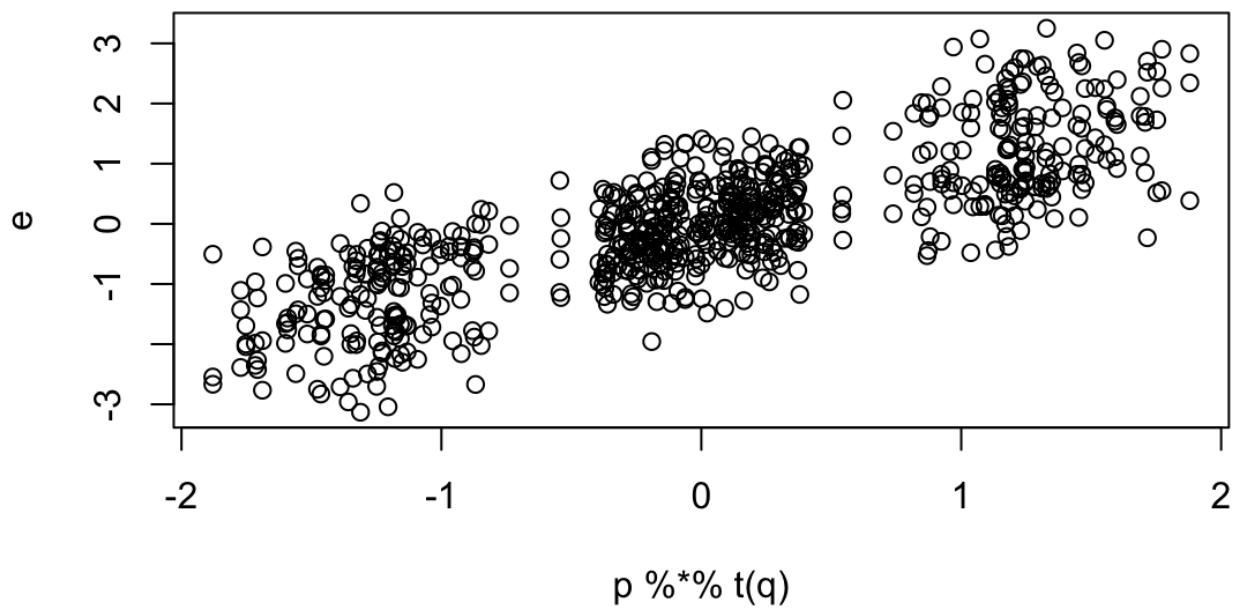


Figure 3:

```

#> Godfather          -0.280      -0.198
#> Godfather 2        -0.186      -0.364
#> Goodfellas         0.619       0.650
#> Scent of a Woman   -0.641      -0.656
#> You've Got Mail    1.000       0.353
#> Sleepless in Seattle 0.353      1.000

```

This structure seems to be driven by Al Pacino being in the movie or not. This implies we could add another factor:

```

q <- cbind(c(-1, -1, -1, 1, 1, 1),
            c(1, 1, -1, 1, -1, 1))

```

We can then obtain estimates for each user:

```

p <- t(apply(e, 1, function(y) lm(y~q-1)$coefficient))

```

Note that we use the transpose `t` because `apply` binds results into columns and we want a row for each user.

Our approximation based on two factors does a even better job of predicting how our residuals deviate from 0:

```

plot(p %*% t(q), e)

```

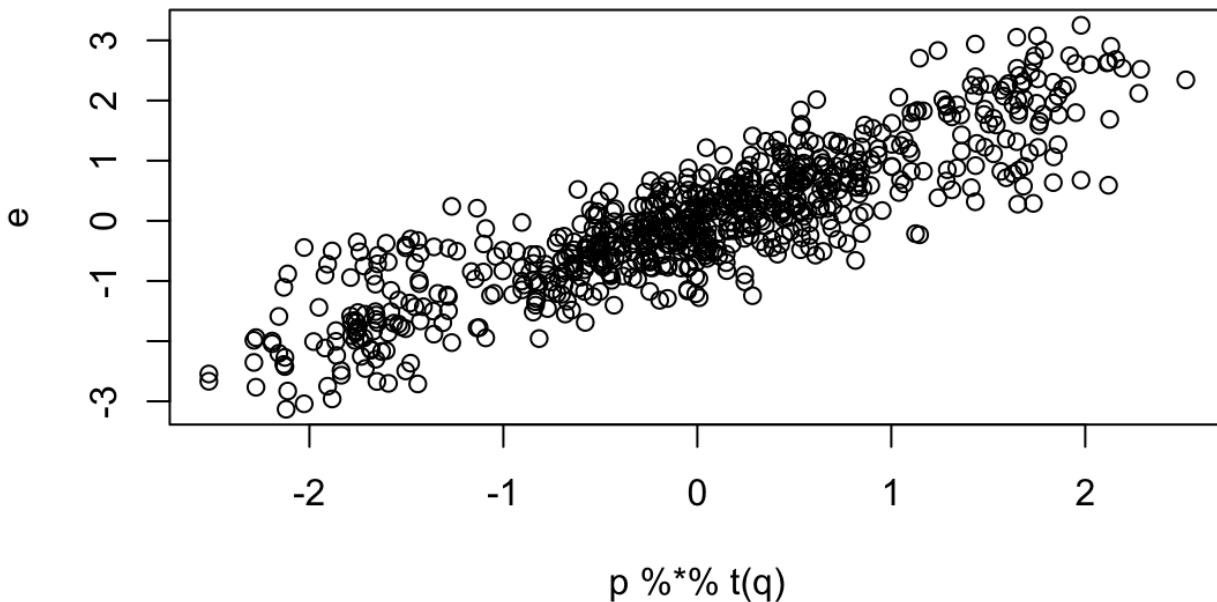


Figure 4:

This analysis provides insights into the process generating our data. Note that it also provides compression: the (120×6) matrix $\boldsymbol{\varepsilon}$, with 720 observation, is well approximated by a matrix multiplication of a (120×2) matrix \mathbf{P} and a (6×2) matrix \mathbf{Q} , a total of 252 parameters.

Our approximation with two factors can be written as:

$$[\varepsilon_{i,j} \approx p_{i,1}q_{j,1} + p_{i,2}q_{j,2} \text{ or } \boldsymbol{\varepsilon} = \mathbf{P}\mathbf{Q}^T]$$

In our example with simulated data, we deduced the factors $\langle(\mathbf{p}_1)\rangle$ and $\langle(\mathbf{p}_2)\rangle$ from the sample correlation and our knowledge of movies. These ended up working well. However, in general deducing factors is not this easy. Furthermore, factors that provide good approximation might be more complicated than containing just two values. For example, *The Godfather III* might be considered both a mob and romance movie and we would not know what value to assign it in \mathbf{q} .

So, can we estimate the factors? A challenge is that if $\langle(\mathbf{P})\rangle$ is unknown our model is no longer linear: we can use `lm` to estimate both $\langle(\mathbf{P})\rangle$ and $\langle(\mathbf{Q})\rangle$. In the next section, we describe a technique that permits us to estimate to this.

24.2 Connection to PCA

Notice that if we perform PCA on the matrix $\langle(\boldsymbol{\varepsilon})\rangle$, we obtain a transformation $\langle(\mathbf{V})\rangle$ that permits us to rewrite:

$$\langle(\boldsymbol{\varepsilon})\rangle = \mathbf{Z} \mathbf{V}^{\text{top}}$$

with $\langle(\mathbf{Z})\rangle$ the matrix of principal components.

Let's perform PCA and examine the results:

```
pca <- prcomp(e, center = FALSE)
```

First, notice that the first two PCs explain over 95% of the variability:

```
pca$sdev^2/sum(pca$sdev^2)
#> [1] 0.6939 0.1790 0.0402 0.0313 0.0303 0.0253
```

Next, notice that the first column of $\langle(\mathbf{V})\rangle$:

```
pca$rotation[,1]
#>      Godfather          Godfather 2        Goodfellas
#>      0.306              0.261            0.581
#> Scent of a Woman      You've Got Mail Sleepless in Seattle
#>      -0.570             -0.294           -0.300
```

is assigning positive values to the mob movies and negative values to the romance movies.

The second column:

```
pca$rotation[,2]
#>      Godfather          Godfather 2        Goodfellas
#>     -0.354             -0.377            0.382
#> Scent of a Woman      You've Got Mail Sleepless in Seattle
#>     -0.437              0.448            0.442
```

is coding for Al Pacino movies.

PCA is automatically finding what we deduced with our knowledge of movies. This is not a coincidence.

Assume that data $\langle(\mathbf{Y})\rangle$ follows the model:

$$\langle Y_{i,j}\rangle = \sum_{k=1}^K p_{i,k} q_{j,k} + \langle \varepsilon_{i,j}\rangle$$

If we define the matrices $\langle(\mathbf{Y})\rangle$ and $\langle(\boldsymbol{\varepsilon})\rangle$ to have $\langle(y_{i,j})\rangle$ and $\langle(\varepsilon_{i,j})\rangle$ in the $\langle(i)\rangle$ th row and $\langle(j)\rangle$ th column, respectively, and $\langle(\mathbf{P})\rangle$ and $\langle(\mathbf{Q})\rangle$ to have entries $\langle(p_{i,k})\rangle$ and $\langle(q_{j,k})\rangle$ in the $\langle(i)\rangle$ th and $\langle(k)\rangle$ th column, respectively, we can rewrite the model as:

$$\langle Y\rangle = \langle P\rangle \langle Q\rangle^{\text{top}} + \langle \varepsilon\rangle$$

Notice this model is not identifiable since we can multiply the $\langle(\mathbf{P})\rangle$ by any positive constant and obtain the same model by dividing $\langle(\mathbf{Q})\rangle$ by this same constant. To avoid this, we impose the constraint that $\langle(\mathbf{Q})\rangle$ is orthogonal:

$$\langle Q\rangle^{\text{top}} \langle Q\rangle = \langle I\rangle$$

The first $\lfloor K \rfloor$ columns of the principal components and associated rotation provide estimates of $\lfloor \mathbf{P} \rfloor$ and $\lfloor \mathbf{Q} \rfloor$ respectively.

24.3 Case study: movie recommendations

Note that if we look at the correlation structure of the movies for which we simulated data in the previous sections, we see structure as well:

```
#> Godfather, The Godfather: Part II, The
#> Godfather, The 1.000 0.842
#> Godfather: Part II, The 0.842 1.000
#> Goodfellas 0.521 0.507
#> Scent of a Woman 0.323 0.209
#> You've Got Mail -0.405 -0.213
#> Sleepless in Seattle -0.334 -0.295
#> Goodfellas Scent of a Woman You've Got Mail
#> Godfather, The 0.5208 0.3231 -0.405
#> Godfather: Part II, The 0.5065 0.2091 -0.213
#> Goodfellas 1.0000 -0.0277 -0.254
#> Scent of a Woman -0.0277 1.0000 -0.312
#> You've Got Mail -0.2542 -0.3119 1.000
#> Sleepless in Seattle -0.4484 -0.4405 0.455
#> Sleepless in Seattle
#> Godfather, The -0.334
#> Godfather: Part II, The -0.295
#> Goodfellas -0.448
#> Scent of a Woman -0.441
#> You've Got Mail 0.455
#> Sleepless in Seattle 1.000
```

This implies that we should be able to improve the predictions made in the previous chapter if we use this information. Ratings on *The Godfather* should inform ratings for the *The Godfather II*, for example. But what other patterns might be useful for prediction?

We will rewrite the model from the previous chapter to include factors to explain similarities between movies:

$$Y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^K p_{i,k} q_{j,k} + \varepsilon_{i,j}$$

Unfortunately, we can't fit this model with `prcomp` due to the missing values. We introduce the **missMDA** package that provides an approach to fit such models when matrix entries are missing, a very common occurrence in movie recommendations, through the function `imputePCA`. Also, because there are small sample sizes for several movie pairs, it is useful to regularize the (p) s. The `imputePCA` function also permits regularization.

We use the estimates for μ , the α s and β s from the previous chapter, and estimate two factors (`ncp = 2`). We fit the model to movies rated more than 25 times, include *Scent of a Woman*, which does not meet this criterion, because we previously used it as an example. Finally, we use regularization by setting the parameter `coeff.ridge` to the same value used to estimate the β s.

```
library(missMDA)
ind <- colSums(!is.na(y)) >= 25 | colnames(y) == "3252"
imputed <- imputePCA(r[,ind], ncp = 2, coeff.ridge = lambda)
```

To see how much we improve our previous prediction, we construct a matrix with the ratings in the test set:

```
y_test <- select(test_set, movieId, userId, rating) |>
  pivot_wider(names_from = movieId, values_from = rating) |>
  column_to_rownames("userId") |>
  as.matrix()
```

and construct our predictor obtained with `imputePCA`. We start by constructing the predictor from the previous chapter:

```

pred <- matrix(0, nrow(y), ncol(y))
rownames(pred) <- rownames(y); colnames(pred) <- colnames(y)
pred <- clamp(sweep(pred + mu + a, 2, b_reg, FUN = "+"))
rmse(y_test - pred[rownames(y_test), colnames(y_test)])
#> [1] 0.889

```

Then we adjust the prediction to include the imputed residuals for the test set:

```
pred[,ind] <- clamp(pred[,ind] + imputed$complete0bs)
```

We see that our prediction improves:

```

rmse(y_test - pred[rownames(y_test), colnames(y_test)])
#> [1] 0.875

```

We note that further improvements can be obtained by 1) optimizing the regularization penalty, 2) considering more than 2 factors, and 3) accounting for the fact that a missing rating provides information: people tend to not watch movies they know they won't like.

24.3.1 Visualizing factors

We can compute the first two principal components used for the prediction using `prcomp`.

```
pca <- prcomp(imputed$complete0bs, center = FALSE, rank. = 3)
```

By adding the movie names to the rotation matrix:

```

v <- pca$rotation
rownames(v) <- with(movie_map, title[match(colnames(r[,ind]), movieId)])

```

and visually explore the results, we see that the mob movies discussed above are close to each other, as are the romance movies without Al Pacino.

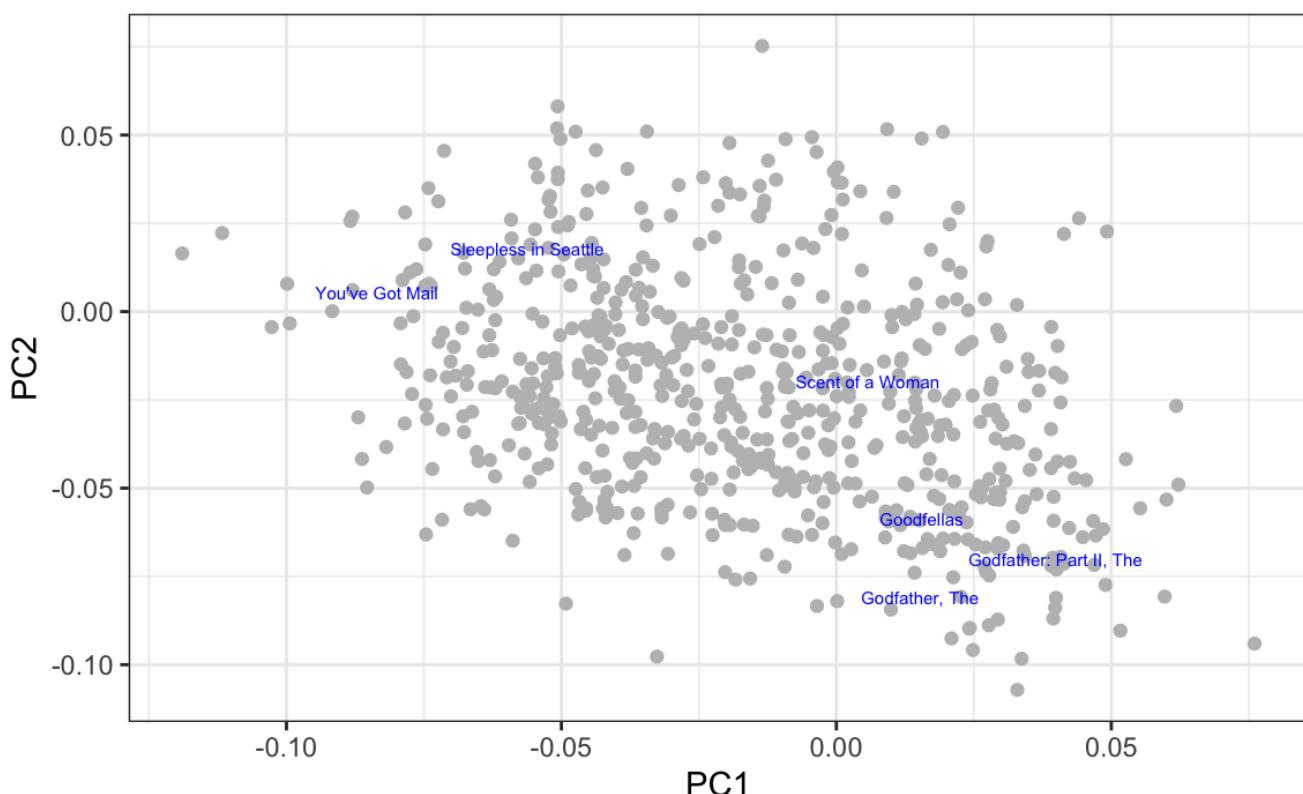


Figure 5:

By looking at the highest and lowest values for the first principal component, we see a meaningful pattern. The first PC shows the difference between Hollywood blockbusters on one side:

```
#> [1] "Armageddon"                  "Pearl Harbor"
#> [3] "X2: X-Men United"          "Dark Knight Rises, The"
#> [5] "X-Men"                      "Independence Day (a.k.a. ID4)"
#> [7] "Con Air"                    "I, Robot"
#> [9] "World Is Not Enough, The"  "Spider-Man 2"
```

and critically acclaimed movies on the other:

```
#> [1] "2001: A Space Odyssey"      "American Psycho"
#> [3] "Royal Tenenbaums, The"     "Harold and Maude"
#> [5] "Apocalypse Now"            "Fear and Loathing in Las Vegas"
#> [7] "Mulholland Drive"          "Clockwork Orange, A"
#> [9] "English Patient, The"      "Dr. Strangelove or: How I L..."
```

24.4 Singular Value Decomposition

The analysis performed here with PCA is often performed with a related technique called the Singular Value Decomposition (SVD). The SVD theorem states that any $(N \times p)$ matrix can be written as:

$\mathbf{Y} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ With (\mathbf{U}) and orthogonal $(N \times p)$ matrix, (\mathbf{V}) an orthogonal $(p \times p)$ matrix, and (\mathbf{D}) a diagonal matrix with $(d_{1,1} \geq d_{2,2} \geq \dots \geq d_{p,p})$. SVD is related to PCA because we can show that (\mathbf{V}) is the rotation that gives us principal components. This implies that $(\mathbf{U}\mathbf{D})$ are the principal components. This also implies that if you square the diagonal entries of (\mathbf{D}) , you obtain the sum of squares of the principal components since:

$$\mathbf{U}^\top \mathbf{D} \mathbf{U} = \mathbf{D}^2$$

In R, we can obtain the SVD using the function `svd`. To see the connection to PCA, notice that:

```
x <- matrix(rnorm(1000), 100, 10)
pca <- prcomp(x, center = FALSE)
s <- svd(x)

all.equal(pca$rotation, s$v, check.attributes = FALSE)
#> [1] TRUE
all.equal(pca$sdev^2, s$d^2/(nrow(x) - 1))
#> [1] TRUE
all.equal(pca$x, s$u %*% diag(s$d), check.attributes = FALSE)
#> [1] TRUE
```

In the exercises, we show that `s$u %*% diag(s$d)` can be computed more efficiently as `sweep(s$u, 2, s$d, "*")`.

24.5 Exercises

In this exercise set, we use the singular value decomposition (SVD) to estimate factors in an example related to the first application of factor analysis: finding factors related to student performance in school.

We construct a dataset that represents grade scores for 100 students in 24 different subjects. The overall average has been removed so this data represents the percentage points each student received above or below the average test score. So a 0 represents an average grade (C), a 25 is a high grade (A+), and a -25 represents a low grade (F). You can simulate the data like this:

```
set.seed(1987)
n <- 100
k <- 8
Sigma <- 64 * matrix(c(1, .75, .5, .75, 1, .5, .5, .5, 1), 3, 3)
m <- MASS::mvrnorm(n, rep(0, 3), Sigma)
m <- m[order(rowMeans(m), decreasing = TRUE),]
```

```

y <- m %x% matrix(rep(1, k), nrow = 1) +
  matrix(rnorm(matrix(n * k * 3)), n, k * 3)
colnames(y) <- c(paste(rep("Math",k), 1:k, sep="_"),
                 paste(rep("Science",k), 1:k, sep="_"),
                 paste(rep("Arts",k), 1:k, sep="_"))

```

Our goal is to describe the student performances as succinctly as possible. For example, we want to know if these test results are all simply random independent numbers. Are all students just about as good? Does being good in one subject imply one will be good in another? How does the SVD help with all this? We will go step by step to show that with just three relatively small pairs of vectors, we can explain much of the variability in this (100×24) dataset.

You can visualize the 24 test scores for the 100 students by plotting an image:

```

my_image <- function(x, zlim = range(x), ...){
  colors = rev(RColorBrewer::brewer.pal(9, "RdBu"))
  cols <- 1:ncol(x)
  rows <- 1:nrow(x)
  image(cols, rows, t(x[rev(rows), , drop=FALSE]), xaxt = "n", yaxt = "n",
        xlab="", ylab="", col = colors, zlim = zlim, ...)
  abline(h=rows + 0.5, v = cols + 0.5)
  axis(side = 1, cols, colnames(x), las = 2)
}

my_image(y)

```

1. How would you describe the data based on this figure?

- a. The test scores are all independent of each other.
- b. The students that test well are at the top of the image and there seems to be three groupings by subject.
- c. The students that are good at math are not good at science.
- d. The students that are good at math are not good at humanities.

2. You can examine the correlation between the test scores directly like this:

```

my_image(cor(y), zlim = c(-1,1))
range(cor(y))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)

```

Which of the following best describes what you see?

- a. The test scores are independent.
- b. Math and science are highly correlated, but the humanities are not.
- c. There is high correlation between tests in the same subject, but no correlation across subjects.
- d. There is a correlation among all tests, but higher if the tests are in science and math and even higher within each subject.

3. Remember that orthogonality means that $\langle U^\top U \rangle$ and $\langle V^\top V \rangle$ are equal to the identity matrix. This implies that we can also rewrite the decomposition as:

$$\langle Y V \rangle = \langle U D \rangle \langle V \rangle^\top \quad \text{or} \quad \langle Y \rangle = \langle U \rangle^\top \langle D \rangle \langle V \rangle^\top$$

We can think of $\langle YV \rangle$ and $\langle U^\top V \rangle$ as two transformations of $\langle Y \rangle$ that preserve the total variability.

Use the function `svd` to compute the SVD of y . This function will return $\langle U \rangle$, $\langle V \rangle$ and the diagonal entries of $\langle D \rangle$.

```

s <- svd(y)
names(s)

```

You can check that the SVD works by typing:

```
y_svd <- sweep(s$u, d) %*% t(s$v)
max(abs(y - y_svd))
```

Compute the sum of squares of the columns of \mathbf{Y} and store them in $\mathbf{ss_y}$. Then compute the sum of squares of columns of the transformed \mathbf{YV} and store them in $\mathbf{ss_yv}$. Confirm that $\text{sum}(\mathbf{ss_y})$ is equal to $\text{sum}(\mathbf{ss_yv})$.

4. We see that the total sum of squares is preserved. This is because \mathbf{V} is orthogonal. Now to start understanding how \mathbf{YV} is useful, plot $\mathbf{ss_y}$ against the column number and then do the same for $\mathbf{ss_yv}$. What do you observe?

5. We see that the variability of the columns of \mathbf{YV} is decreasing. Furthermore, we see that, relative to the first three, the variability of the columns beyond the third is almost 0. Now notice that we didn't have to compute $\mathbf{ss_yv}$ because we already have the answer. How? Remember that $\mathbf{YV} = \mathbf{UD}$ and because \mathbf{U} is orthogonal, we know that the sum of squares of the columns of \mathbf{UD} are the diagonal entries of \mathbf{D} squared. Confirm this by plotting the square root of $\mathbf{ss_yv}$ versus the diagonal entries of \mathbf{D} .

6. From the above we know that the sum of squares of the columns of \mathbf{Y} (the total sum of squares) add up to the sum of $\mathbf{s^2d^2}$, and that the transformation \mathbf{YV} gives us columns with sums of squares equal to $\mathbf{s^2d^2}$. Now compute what percent of the total variability is explained by just the first three columns of \mathbf{YV} .

7. We see that almost 99% of the variability is explained by the first three columns of $\mathbf{YV} = \mathbf{UD}$. So we get the sense that we should be able to explain much of the variability and structure we found while exploring the data with a few columns. Before we continue, let's show a useful computational trick to avoid creating the matrix $\text{diag}(\mathbf{s^2d^2})$. To motivate this, we note that if we write \mathbf{U} out in its columns $(\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p)$, then \mathbf{UD} is equal to:

$$[\mathbf{UD} = [\mathbf{u}_1 \mathbf{d}_{1,1}, \mathbf{u}_2 \mathbf{d}_{2,2}, \dots, \mathbf{u}_p \mathbf{d}_{p,p}]]$$

Use the `sweep` function to compute \mathbf{UD} without constructing $\text{diag}(\mathbf{s^2d^2})$ and without using matrix multiplication.

8. We know that $\mathbf{u}_1 \mathbf{d}_{1,1}$, the first column of \mathbf{UD} , has the most variability of all the columns of \mathbf{UD} . Earlier we saw an image of \mathbf{Y} :

`my_image(y)`

in which we can see that the student to student variability is quite large and that it appears that students that are good in one subject are good in all. This implies that the average (across all subjects) for each student should explain a lot of the variability. Compute the average score for each student and plot it against $\mathbf{u}_1 \mathbf{d}_{1,1}$, and describe what you find.

9. We note that the signs in SVD are arbitrary because:

$$[\mathbf{UDV}^\top = (-\mathbf{U}) \mathbf{D} (-\mathbf{V})^\top]$$

With this in mind, we see that the first column of \mathbf{UD} is almost identical to the average score for each student except for the sign.

This implies that multiplying \mathbf{Y} by the first column of \mathbf{V} must be performing a similar operation to taking the average. Make an image plot of \mathbf{V} and describe the first column relative to others and how this relates to taking an average.

10. We already saw that we can rewrite \mathbf{UD} as:

$$[\mathbf{u}_1 \mathbf{d}_{1,1} + \mathbf{u}_2 \mathbf{d}_{2,2} + \dots + \mathbf{u}_p \mathbf{d}_{p,p}]$$

with \mathbf{u}_j the j-th column of \mathbf{U} . This implies that we can rewrite the entire SVD as:

$$[\mathbf{Y} = \mathbf{u}_1 \mathbf{d}_{1,1} \mathbf{v}_1^\top + \mathbf{u}_2 \mathbf{d}_{2,2} \mathbf{v}_2^\top + \dots + \mathbf{u}_p \mathbf{d}_{p,p} \mathbf{v}_p^\top]$$

with \mathbf{v}_j the jth column of \mathbf{V} . Plot $\mathbf{u}_1 \mathbf{d}_{1,1}$, then plot \mathbf{v}_1^\top using the same range for the y-axis limits. Then make an image of $\mathbf{u}_1 \mathbf{d}_{1,1} \mathbf{v}_1^\top$

\hat{Y}) and compare it to the image of (\mathbf{Y}) . Hint: Use the `my_image` function defined above and use the `drop=FALSE` argument to assure the subsets of matrices are matrices.

11. We see that with just a vector of length 100, a scalar, and a vector of length 24, we actually come close to reconstructing the original (100×24) matrix. This is our first matrix factorization:

$\mathbf{Y} \approx \mathbf{d}_1 \mathbf{u}_1 \mathbf{v}_1 + \mathbf{d}_2 \mathbf{u}_2 \mathbf{v}_2 + \mathbf{d}_3 \mathbf{u}_3 \mathbf{v}_3 + \dots + \mathbf{d}_{24} \mathbf{u}_{24} \mathbf{v}_{24}$

We know it explains $s\$d[1]^2 / sum(s\$d^2) * 100$ percent of the total variability. Our approximation only explains the observation that good students tend to be good in all subjects. But another aspect of the original data that our approximation does not explain was the higher similarity we observed within subjects. We can see this by computing the difference between our approximation and original data and then computing the correlations. You can see this by running this code:

```
resid <- y - with(s,(u[,1, drop=FALSE]*d[1]) %*% t(v[,1, drop=FALSE]))
my_image(cor(resid), zlim = c(-1,1))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```

Now that we have removed the overall student effect, the correlation plot reveals that we have not yet explained the within subject correlation nor the fact that math and science are closer to each other than to the arts. So let's explore the second column of the SVD. Repeat the previous exercise but for the second column: Plot (\mathbf{u}_2) , then plot (\mathbf{v}_2) using the same range for the y-axis limits, then make an image of $(\mathbf{u}_2 \mathbf{d}_{2,2} \mathbf{v}_2)$ and compare it to the image of `resid`.

12. The second column clearly relates to a student's difference in ability in math/science versus the arts. We can see this most clearly from the plot of `s$v[,2]`. Adding the matrix we obtain with these two columns will help with our approximation:

$\mathbf{Y} \approx \mathbf{d}_1 \mathbf{u}_1 \mathbf{v}_1 + \mathbf{d}_2 (\mathbf{u}_2 \mathbf{v}_2) + \mathbf{d}_3 \mathbf{u}_3 \mathbf{v}_3 + \dots + \mathbf{d}_{24} \mathbf{u}_{24} \mathbf{v}_{24}$

We know it will explain:

```
sum(s$d[1:2]^2) / sum(s$d^2) * 100
```

percent of the total variability. We can compute new residuals like this:

```
resid <- y - with(s,sweep(u[,1:2], 2, d[1:2], FUN="*")) %*% t(v[,1:2])
my_image(cor(resid), zlim = c(-1,1))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```

and see that the structure that is left is driven by the differences between math and science. Confirm this by plotting (\mathbf{u}_3) , then plot (\mathbf{v}_3) using the same range for the y-axis limits, then make an image of $(\mathbf{u}_3 \mathbf{d}_{3,3} \mathbf{v}_3)$ and compare it to the image of `resid`.

13. The third column clearly relates to a student's difference in ability in math and science. We can see this most clearly from the plot of `s$v[,3]`. Adding the matrix we obtain with these two columns will help with our approximation:

$\mathbf{Y} \approx \mathbf{d}_1 \mathbf{u}_1 \mathbf{v}_1 + \mathbf{d}_2 \mathbf{u}_2 \mathbf{v}_2 + \mathbf{d}_3 (\mathbf{u}_3 \mathbf{v}_3) + \dots + \mathbf{d}_{24} \mathbf{u}_{24} \mathbf{v}_{24}$

We know it will explain:

```
sum(s$d[1:3]^2) / sum(s$d^2) * 100
```

percent of the total variability. We can compute new residuals like this:

```
resid <- y - with(s,sweep(u[,1:3], 2, d[1:3], FUN="*")) %*% t(v[,1:3])
my_image(cor(resid), zlim = c(-1,1))
axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)
```

We no longer see structure in the residuals: they seem to be independent of each other. This implies that we can describe the data with the following model:

$\mathbf{Y} = \mathbf{d}_1 \mathbf{u}_1 \mathbf{v}_1 + \mathbf{d}_2 \mathbf{u}_2 \mathbf{v}_2 + \mathbf{d}_3 \mathbf{u}_3 \mathbf{v}_3 + \dots + \mathbf{d}_{24} \mathbf{u}_{24} \mathbf{v}_{24} + \varepsilon$

with $\backslash(\backslashvarepsilon)$ a matrix of independent identically distributed errors. This model is useful because we summarize (100×24) observations with $(3 \times (100+24+1) = 375)$ numbers. Furthermore, the three components of the model have useful interpretations: 1) the overall ability of a student, 2) the difference in ability between the math/sciences and arts, and 3) the remaining differences between the three subjects. The sizes $\backslash(d_{1,1})$, $d_{2,2}$) and $\backslash(d_{3,3})$ tell us the variability explained by each component. Finally, note that the components $\backslash(d_{j,j} \mathbf{u}_j \mathbf{v}_j^\top)$ are equivalent to the jth principal component.

Finish the exercise by plotting an image of $\backslash(Y)$, an image of $\backslash(d_{1,1} \mathbf{u}_1 \mathbf{v}_1^\top + d_{2,2} \mathbf{u}_2 \mathbf{v}_2^\top + d_{3,3} \mathbf{u}_3 \mathbf{v}_3^\top)$ and an image of the residuals, all with the same `zlim`.

Introduction to Data Science - Machine Learning

Rafael A. Irizarry

Machine learning has achieved remarkable successes in a variety of applications. These range from the postal service's use of machine learning for reading handwritten zip codes to the development of voice recognition systems like Apple's Siri. Other significant advances include movie recommendation systems, spam and malware detection, housing price prediction algorithms, and the ongoing development of autonomous vehicles.

The field of *Artificial Intelligence (AI)* has been evolving for several decades. Traditional AI systems, including some chess-playing machines, often relied on decision-making based on preset rules and knowledge representation. However, with the advent of data availability, machine learning has gained prominence. It focuses on decision-making through algorithms trained with data. In recent years, the terms AI and Machine Learning have been used interchangeably in many contexts, though they have distinct meanings. AI broadly refers to systems or applications that exhibit intelligent behavior, encompassing both rule-based approaches and machine learning. Machine Learning specifically involves learning from data to make decisions or predictions.

In this part of the book, we will delve into the concepts, ideas, and methodologies of machine learning. We will also demonstrate their practical application, using the example of recognizing handwritten digits, a classic problem that exemplifies the power and utility of machine learning techniques. Machine learning has achieved remarkable successes, ranging from the postal service's handwritten zip code readers to voice recognition systems like Apple's Siri. These advances also include movie recommendation systems, spam and malware detection, housing price prediction algorithms, and the development of driverless cars.

Introduction to Data Science - 25 Notation and terminology

Rafael A. Irizarry

In \ref{@mnist}, we introduced the MNIST handwritten digits dataset. Here we describe how the task of automatically reading these digits can be framed as a machine learning challenge. In doing so, we introduce machine learning mathematical notation and terminology used throughout this part of the book.

Originally, mail sorting in the post office involved humans reading zip codes written on the envelopes. Today, thanks to machine learning algorithms, a computer can read zip codes and then a robot sorts the letters. We will learn how to build algorithms that can read a digitized handwritten digit.

25.1 Terminology

In machine learning, data comes in the form of the *outcome* we want to predict and the *features* that we will use to predict the outcome. We build algorithms that take feature values as input and returns a prediction for the outcome when we don't know the outcome. The machine learning approach is to *train* an algorithm using a dataset for which we do know the outcome, and then apply this algorithm in the future to make a prediction when we don't know the outcome.

Prediction problems can be divided into categorical and continuous outcomes. For categorical outcomes, $\{Y\}$ can be any one of $\{K\}$ classes. The number of classes can vary greatly across applications. For example, in the digit reader data, $\{K=10\}$ with the classes being the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. In speech recognition, the outcomes are all possible words or phrases we are trying to detect. Spam detection has two outcomes: spam or not spam. In this book, we denote the $\{K\}$ categories with indexes $\{k=1,\dots,K\}$. However, for binary data we will use $\{k=0,1\}$ for mathematical conveniences that we demonstrate later.

25.2 Notation

Here we will use $\{Y\}$ to denote the outcome and $\{X_1, \dots, X_p\}$ to denote features. Note that features are sometimes referred to as predictors or covariates. We consider all these to be synonyms.

The first step in building an algorithm is to understand what are the outcomes and features. In Section 20.1, we showed that associated with each digitized image $\{i\}$, there is a categorical outcome $\{Y_i\}$ and features $\{X_{i,1}, \dots, X_{i,p}\}$, with $\{p=784\}$. We use bold face $\{\mathbf{X}_i = (X_{i,1}, \dots, X_{i,p})\}$ to denote the vector of predictors. Notice that we are using the matrix notation described in Section 20.5. When referring to an arbitrary set of features rather than a specific image, we drop the index $\{i\}$ and use $\{Y\}$ and $\{\mathbf{X} = (X_1, \dots, X_p)\}$. We use upper case variables because, in general, we think of the outcome and predictors as random variables. We use lower case, for example $\{\mathbf{X} = \mathbf{x}\}$, to denote observed values. Although, when we code, we adhere to lower case.

The machine learning task is to build an algorithm that returns a prediction for any of the possible values of the features. Here, we will learn several approaches to building these algorithms. Although at this point it might seem impossible to achieve this, we will start with basic examples and build up our knowledge until we can tackle more complex ones. In fact, we start with an artificially simple example with just one predictor and then move on to a slightly more realistic example with two predictors. Once we understand these, we will address real-world machine learning challenges involving many predictors.

25.3 The machine learning challenge

The general setup is as follows. We have a series of features and an unknown outcome we want to predict:

outcome	feature 1	feature 2	feature 3	\dots	feature p
?	(X_1)	(X_2)	(X_3)	\dots	(X_p)

To *build a model* that provides a prediction for any set of observed values $(X_1=x_1, X_2=x_2, \dots, X_p=x_p)$, we collect data for which we know the outcome:

outcome	feature 1	feature 2	feature 3	\dots	feature 5
(y_1)	$(x_{1,1})$	$(x_{1,2})$	$(x_{1,3})$	\dots	$(x_{1,p})$
(y_2)	$(x_{2,1})$	$(x_{2,2})$	$(x_{2,3})$	\dots	$(x_{2,p})$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
(y_n)	$(x_{n,1})$	$(x_{n,2})$	$(x_{n,3})$	\dots	$(x_{n,p})$

When the output is continuous, we refer to the machine learning task as *prediction*, and the main output of the model is a function f that automatically produces a prediction, denoted with \hat{y} , for any set of predictors: $\hat{y} = f(x_1, x_2, \dots, x_p)$. We use the term *actual outcome* to denote what we end up observing. So we want the prediction \hat{y} to match the actual outcome y as best as possible. Because our outcome is continuous, our predictions \hat{y} will not be either exactly right or wrong, but instead we will determine an *error* defined as the difference between the prediction and the actual outcome $y - \hat{y}$.

When the outcome is categorical, we refer to the machine learning task as *classification*, and the main output of the model will be a *decision rule* which prescribes which of the K classes we should predict. In this scenario, most models provide functions of the predictors for each class $f_k(x_1, x_2, \dots, x_p)$, that are used to make this decision. When the data is binary, a typical decision rules looks like this: if $f_1(x_1, x_2, \dots, x_p) > C$, predict category 1, if not the other category, with C a predetermined cutoff. Because the outcomes are categorical, our predictions will be either right or wrong.

Notice that these terms vary among courses, textbooks, and other publications. Often *prediction* is used for both categorical and continuous outcomes, and the term *regression* can be used for the continuous case. Here we avoid using *regression* to avoid confusion with our previous use of the term *linear regression*. In most cases, it will be clear if our outcomes are categorical or continuous, so we will avoid using these terms when possible.

Introduction to Data Science - 26 Evaluation metrics

Rafael A. Irizarry

Before we start describing approaches to optimize the way we build algorithms, we first need to define what we mean when we say one approach is better than another. In this section, we focus on describing ways in which machine learning algorithms are evaluated. Specifically, we need to quantify what we mean by “better”.

For our first introduction to machine learning concepts, we will start with a boring and simple example: how to predict sex using height. As we explain how to build a prediction algorithm with this example, we will start to set down the first building block needed to understand machine learning. Soon enough, we will be undertaking more interesting challenges.

We introduce the **caret** package, which provides useful functions to facilitate machine learning in R, and we describe it in more detail in Section 31.1. For our first example, we use the height data provided by the **dslabs** package.

```
library(caret)
library(dslabs)
```

We start by defining the outcome and predictors.

```
y <- heights$sex
x <- heights$height
```

In this case, we have only one predictor, height, and y is clearly a categorical outcome since observed values are either **Male** or **Female**. We know that we will not be able to predict $\backslash(Y\backslash)$ very accurately based on $\backslash(X\backslash)$ because male and female average heights are not that different relative to within group variability. But can we do better than guessing? To answer this question, we need a quantitative definition of better.

26.1 Training and test sets

Ultimately, a machine learning algorithm is evaluated on how it performs in the real world with completely new datasets. However, when developing an algorithm, we usually have a dataset for which we know the outcomes, as we do with the heights: we know the sex of every student in our dataset. Therefore, to mimic the ultimate evaluation process, we typically split the data into two parts and act as if we don’t know the outcome for one of these. We stop pretending we don’t know the outcome to evaluate the algorithm, but only *after* we are done constructing it. We refer to the group for which we know the outcome, and that we use to develop the algorithm, as the *training* set. We refer to the group for which we pretend we don’t know the outcome as the *test* set.

A standard way of generating the training and test sets is by randomly splitting the data. The **caret** package includes the function **createDataPartition** that helps us generate indexes for randomly splitting the data into training and test sets:

```
set.seed(2007)
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
```

The argument **times** is used to define how many random samples of indexes to return, the argument **p** is used to define what proportion of the data is represented by the index, and the argument **list** is used to decide if we want the indexes returned as a list or not. We can use the result of the **createDataPartition** function call to define the training and test sets as follows:

```
test_set <- heights[test_index, ]
train_set <- heights[-test_index, ]
```

We will now develop an algorithm using **only** the training set. Once we are done developing the algorithm, we will *freeze* it and evaluate it using the test set. The simplest way to evaluate the algorithm when the outcomes are categorical is by simply reporting the proportion of cases that were correctly predicted **in the test set**. This metric is usually referred to as *overall accuracy*.

26.2 Overall accuracy

To demonstrate the use of overall accuracy, we will build two competing algorithms and compare them.

Let's start by developing the simplest possible machine algorithm: guessing the outcome.

```
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE)
```

Note that we are completely ignoring the predictor and simply guessing the sex.

In machine learning applications, it is useful to use factors to represent the categorical outcomes because R functions developed for machine learning, such as those in the **caret** package, require or recommend that categorical outcomes be coded as factors. So convert `y_hat` to factors using the `factor` function:

```
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE) |>
  factor(levels = levels(test_set$sex))
```

The *overall accuracy* is simply defined as the overall proportion that is predicted correctly:

```
mean(y_hat == test_set$sex)
#> [1] 0.51
```

Not surprisingly, our accuracy is about 50%. We are guessing!

Can we do better? Exploratory data analysis suggests we can because, on average, males are slightly taller than females:

```
library(tidyverse)
heights |> group_by(sex) |> summarize(avg = mean(height), sd = sd(height))
#> # A tibble: 2 × 3
#>   sex     avg     sd
#>   <fct>  <dbl>  <dbl>
#> 1 Female  64.9  3.76
#> 2 Male    69.3  3.61
```

But how do we make use of this insight? Let's try another simple approach: predict `Male` if height is within two standard deviations from the average male.

```
y_hat <- factor(ifelse(x > 62, "Male", "Female"), levels(test_set$sex))
```

The accuracy goes up from 0.50 to about 0.80:

```
mean(y == y_hat)
#> [1] 0.793
```

But can we do even better? In the example above, we used a cutoff of 62, but we can examine the accuracy obtained for other cutoffs and then pick the value that provides the best results. But remember, **it is important that we optimize the cutoff using only the training set**: the test set is only for evaluation. Although for this simplistic example it is not much of a problem, later we will learn that evaluating an algorithm on the training set can lead to *overfitting*, which often results in dangerously over-optimistic assessments.

Here we examine the accuracy of 10 different cutoffs and pick the one yielding the best result:

```
cutoff <- seq(61, 70)
accuracy <- sapply(cutoff, function(x){
  y_hat <- factor(ifelse(train_set$height > x, "Male", "Female"), levels = levels(test_set$sex))
  mean(y_hat == train_set$sex)
})
```

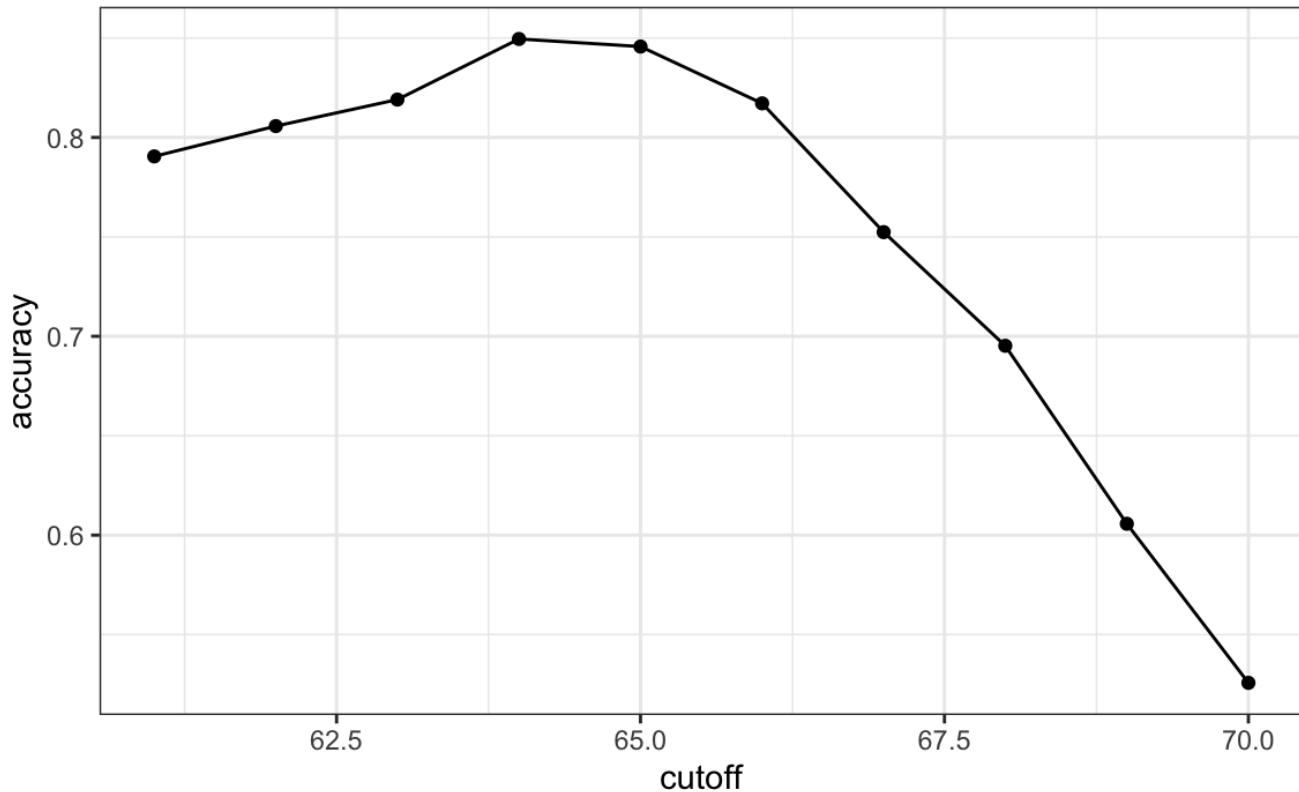


Figure 1:

We can make a plot showing the accuracy obtained on the training set for males and females:

We see that the maximum value is:

```
max(accuracy)
#> [1] 0.85
```

which is much higher than 0.5. The cutoff resulting in this accuracy is:

```
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff
#> [1] 64
```

We can now test this cutoff on our test set to make sure our accuracy is not overly optimistic:

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") |>
  factor(levels = levels(test_set$sex))
y_hat <- factor(y_hat)
mean(y_hat == test_set$sex)
#> [1] 0.804
```

We see that it is a bit lower than the accuracy observed for the training set, but it is still better than guessing. And by testing on a dataset that we did not train on, we know our result is not due to cherry-picking a good result.

26.3 The confusion matrix

The prediction rule we developed in the previous section predicts **Male** if the student is taller than 64 inches. Given that the average female is about 64 inches, this prediction rule seems wrong. What happened? If a student is the height of the average female, shouldn't we predict **Female**?

Generally speaking, overall accuracy can be a deceptive measure. To see this, we will start by constructing what is

referred to as the *confusion matrix*, which basically tabulates each combination of prediction and actual value. We can do this in R simply using `table(predicted = y_hat, actual = test_set$sex)`,

but the `confusionMatrix caret` package computes the confusion matrix and much more:

```
cm <- confusionMatrix(data = y_hat, reference = test_set$sex)
cm$table
#>           Reference
#> Prediction Female Male
#>     Female      48   32
#>     Male        71  374
```

If we study this table closely, it reveals a problem. If we compute the accuracy separately for each sex, we get:

```
cm$byClass[c("Sensitivity", "Specificity")]
#> Sensitivity Specificity
#>      0.403      0.921
```

In the next section, we explain that these two are equivalent to accuracy with females and males, respectively.

We notice an imbalance: too many females are predicted to be male. We are calling almost half of the females male! How can our overall accuracy be so high then? This is because the *prevalence* of males in this dataset is high. These heights were collected from three data sciences courses, two of which had higher male enrollment:

```
cm$byClass["Prevalence"]
#> Prevalence
#>      0.227
```

So when computing overall accuracy, the high percentage of mistakes made for females is outweighed by the gains in correct calls for men. This type of bias can actually be a big problem in practice. If your training data is biased in some way, you are likely to develop algorithms that are biased as well. The fact that we used a test set does not matter because it is also derived from the original biased dataset. This is one of the reasons we look at metrics other than overall accuracy when evaluating a machine learning algorithm.

There are several metrics that we can use to evaluate an algorithm in a way that prevalence does not cloud our assessment, and these can all be derived from the confusion matrix. A general improvement to using overall accuracy is to study *sensitivity* and *specificity* separately.

26.4 Sensitivity and specificity

To define sensitivity and specificity, we need a binary outcome. When the outcomes are categorical, we can define these terms for a specific category. In the digits example, we can ask for the specificity in the case of correctly predicting 2 as opposed to some other digit. Once we specify a category of interest, then we can talk about positive outcomes, $\{Y=1\}$, and negative outcomes, $\{Y=0\}$.

In general, *sensitivity* is defined as the ability of an algorithm to predict a positive outcome when the actual outcome is positive: $\{\hat{Y}=1\}$ when $\{Y=1\}$. Because an algorithm that calls everything positive ($\{\hat{Y}=1\}$ no matter what) has perfect sensitivity, this metric on its own is not enough to judge an algorithm. For this reason, we also examine *specificity*, which is generally defined as the ability of an algorithm to not predict a positive $\{\hat{Y}=0\}$ when the actual outcome is not a positive $\{Y=0\}$. We can summarize in the following way:

- High sensitivity: $\{Y=1 \text{ implies } \hat{Y}=1\}$
- High specificity: $\{Y=0 \text{ implies } \hat{Y}=0\}$

Although the above is often considered the definition of specificity, another way to think of specificity is by the proportion of positive calls that are actually positive:

- High specificity: $\{\hat{Y}=1 \text{ implies } Y=1\}$.

To provide precise definitions, we name the four entries of the confusion matrix:

	Actually Positive	Actually Negative
Predicted positive	True positives (TP)	False positives (FP)

	Actually Positive	Actually Negative
Predicted negative	False negatives (FN)	True negatives (TN)
	TP	FP
Sensitivity	$\frac{TP}{TP+FN}$	$\frac{TN}{TN+FP}$
Specificity	$\frac{TN}{TN+FP}$	$\frac{TP}{TP+FN}$
PPV	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$
Recall	$\frac{TP}{TP+FN}$	$\frac{TN}{TN+FP}$
1-FPR	$\frac{TN}{TN+FP}$	$\frac{TP}{TP+FN}$

Sensitivity is typically quantified by $\frac{TP}{TP+FN}$, the proportion of actual positives (the first column = $\frac{TP}{TP+FN}$) that are called positives ($\frac{TP}{TP+FN}$). This quantity is referred to as the *true positive rate* (TPR) or *recall*.

Specificity is defined as $\frac{TN}{TN+FP}$ or the proportion of negatives (the second column = $\frac{TN}{TP+FN}$) that are called negatives ($\frac{TN}{TP+FN}$). This quantity is also called the true negative rate (TNR). There is another way of quantifying specificity which is $\frac{TP}{TP+FP}$ or the proportion of outcomes called positives (the first row or $\frac{TP}{TP+FP}$) that are actually positives ($\frac{TP}{TP+FP}$). This quantity is referred to as *positive predictive value* (PPV) and also as *precision*. Note that, unlike TPR and TNR, precision depends on prevalence since higher prevalence implies you can get higher precision even when guessing.

The multiple names can be confusing, so we include a table to help us remember the terms. The table includes a column that shows the definition if we think of the proportions as probabilities.

Measure of	Name 1	Name 2	Definition	Probability representation
sensitivity	TPR	Recall	$\frac{P(TP)}{P(TP)+P(FN)}$	$\frac{P(Y=1 X=1)}{P(Y=1 X=1)+P(Y=1 X=0)}$
specificity	TNR	1-FPR	$\frac{P(TN)}{P(TN)+P(FP)}$	$\frac{P(Y=0 X=0)}{P(Y=0 X=0)+P(Y=0 X=1)}$
specificity	PPV	Precision	$\frac{P(TP)}{P(TP)+P(FP)}$	$\frac{P(Y=1 X=1)}{P(Y=1 X=1)+P(Y=1 X=0)}$

The **caret** function `confusionMatrix` computes all these metrics for us once we define which category is the “positive” ($Y=1$). The function expects factors as input, and the first level is considered the positive outcome or ($Y=1$). In our example, `Female` is the first level because it comes before `Male` alphabetically. If you type this into R, you will see several metrics including accuracy, sensitivity, specificity, and PPV.

You can access these directly, for example, like this:

```
cm$overall["Accuracy"]
#> Accuracy
#> 0.804
cm$byClass[c("Sensitivity", "Specificity", "Prevalence")]
#> Sensitivity Specificity Prevalence
#> 0.403 0.921 0.227
```

We can see that the high overall accuracy is possible despite relatively low sensitivity. As we hinted at above, the reason this happens is because of the low prevalence (0.23): the proportion of females is low. Because prevalence is low, failing to predict actual females as females (low sensitivity) does not lower the overall accuracy as much as failing to predict actual males as males (low specificity). This is an example of why it is important to examine sensitivity and specificity and not just accuracy. Before applying this algorithm to general datasets, we need to ask ourselves if prevalence will be the same.

26.5 Balanced accuracy and F_1 score

Although we usually recommend studying both specificity and sensitivity, often it is useful to have a one-number summary, for example, for optimization purposes. One metric that is preferred over overall accuracy is the average of specificity and sensitivity, referred to as *balanced accuracy*. Because specificity and sensitivity are rates, it is more appropriate to compute the *harmonic* average. In fact, the F_1 -score, a widely used one-number summary, is the harmonic average of precision and recall:

$$\sqrt{\frac{1}{2} \left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}} \right)}$$

Because it is easier to write, you often see this harmonic average rewritten as:

\[2 \times \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \]

when defining (F_1) .

Remember that, depending on the context, some types of errors are more costly than others. For instance, in the case of plane safety, it is much more important to maximize sensitivity over specificity: failing to predict a plane will malfunction before it crashes is a much more costly error than grounding a plane when, in fact, the plane is in perfect condition. In a capital murder criminal case, the opposite is true since a false positive can lead to executing an innocent person. The (F_1) -score can be adapted to weigh specificity and sensitivity differently. To do this, we define (β) to represent how much more important sensitivity is compared to specificity and consider a weighted harmonic average:

\[\frac{1}{\frac{1}{\text{precision}} + \frac{\beta^2}{\beta^2 + 1} \frac{1}{\text{recall}}} \]

The `F_meas` function in the `caret` package computes this summary with `beta` defaulting to 1.

Let's rebuild our prediction algorithm, but this time maximizing the F-score instead of overall accuracy:

```
cutoff <- seq(61, 70)
F_1 <- sapply(cutoff, function(x){
  y_hat <- factor(ifelse(train_set$height > x, "Male", "Female"), levels(test_set$sex))
  F_meas(data = y_hat, reference = factor(train_set$sex))
})
```

As before, we can plot these (F_1) measures versus the cutoffs:

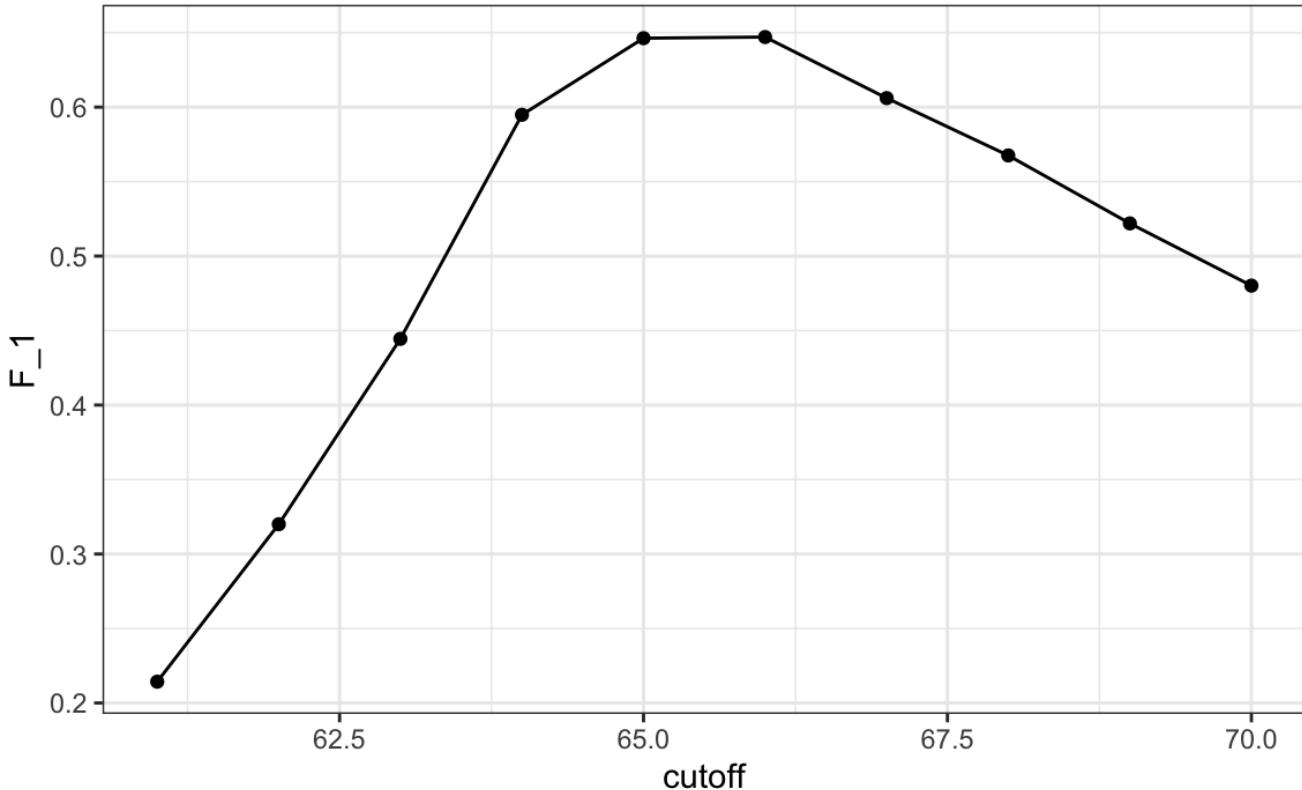


Figure 2:

We see that it is maximized at (F_1) value of:

```
max(F_1)
#> [1] 0.647
```

This maximum is achieved when we use the following cutoff:

```
best_cutoff <- cutoff[which.max(F_1)]
best_cutoff
#> [1] 66
```

A cutoff of 66 makes more sense than 64. Furthermore, it balances the specificity and sensitivity of our confusion matrix:

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") |>
  factor(levels = levels(test_set$sex))
sensitivity(data = y_hat, reference = test_set$sex)
#> [1] 0.63
specificity(data = y_hat, reference = test_set$sex)
#> [1] 0.833
```

We now see that we do much better than guessing, that both sensitivity and specificity are relatively high.

26.6 Prevalence matters in practice

A machine learning algorithm with very high TPR and TNR may not be useful in practice when prevalence is close to either 0 or 1. To see this, consider the case of a doctor that specializes in a rare disease and is interested in developing an algorithm for predicting who has the disease.

The doctor shares data with about 1/2 cases and 1/2 controls and some predictors. You then develop an algorithm with TPR=0.99 and TNR = 0.99. You are excited to explain to the doctor that this means that if a patient has the disease, the algorithm is very likely to predict correctly. The doctor is not impressed and explains that your TNR is too low for this algorithm to be used in practice. This is because this is a rare disease with a prevalence in the general population of 0.5%. The doctor reminds you of Bayes formula:

$$\Pr(Y=1 \mid \hat{Y}=1) = \frac{\Pr(\hat{Y}=1 \mid Y=1)}{\Pr(\hat{Y}=1 \mid Y=0) + \Pr(\hat{Y}=1 \mid Y=1)} = \frac{TPR}{FPR} = \frac{0.99}{0.01} = 99\%$$

Here is plot of precision as a function of prevalence with TPR and TNR are 95%:

Although your algorithm has a precision of about 95% on the data you train on, with prevalence of 50%, if applied to the general population, the algorithm's precision would be just 33%. The doctor can't use an algorithm with 33% of people receiving a positive test actually not having the disease. Note that even if your algorithm had perfect sensitivity, the precision would still be around 33%. So you need to greatly decrease your FPR for the algorithm to be useful in practice.

26.7 ROC and precision-recall curves

When comparing the two methods (guessing versus using a height cutoff), we looked at accuracy and $\sqrt{F_1}$. The second method clearly outperformed the first. However, while we considered several cutoffs for the second method, for the first we only considered one approach: guessing with equal probability. Be aware that guessing `Male` with higher probability would give us higher accuracy due to the bias in the sample:

```
p <- 0.9
n <- length(test_index)
y_hat <- sample(c("Male", "Female"), n, replace = TRUE, prob = c(p, 1 - p)) |>
  factor(levels = levels(test_set$sex))
mean(y_hat == test_set$sex)
#> [1] 0.739
```

But, as described above, this would come at the cost of lower sensitivity. The curves we describe in this section will help us see this.

Remember that for each of these parameters, we can get a different sensitivity and specificity. For this reason, a very common approach to evaluating methods is to compare them graphically by plotting both.

Precision as a function of prevalence

when TPR = 0.95 and TNR = 0.95

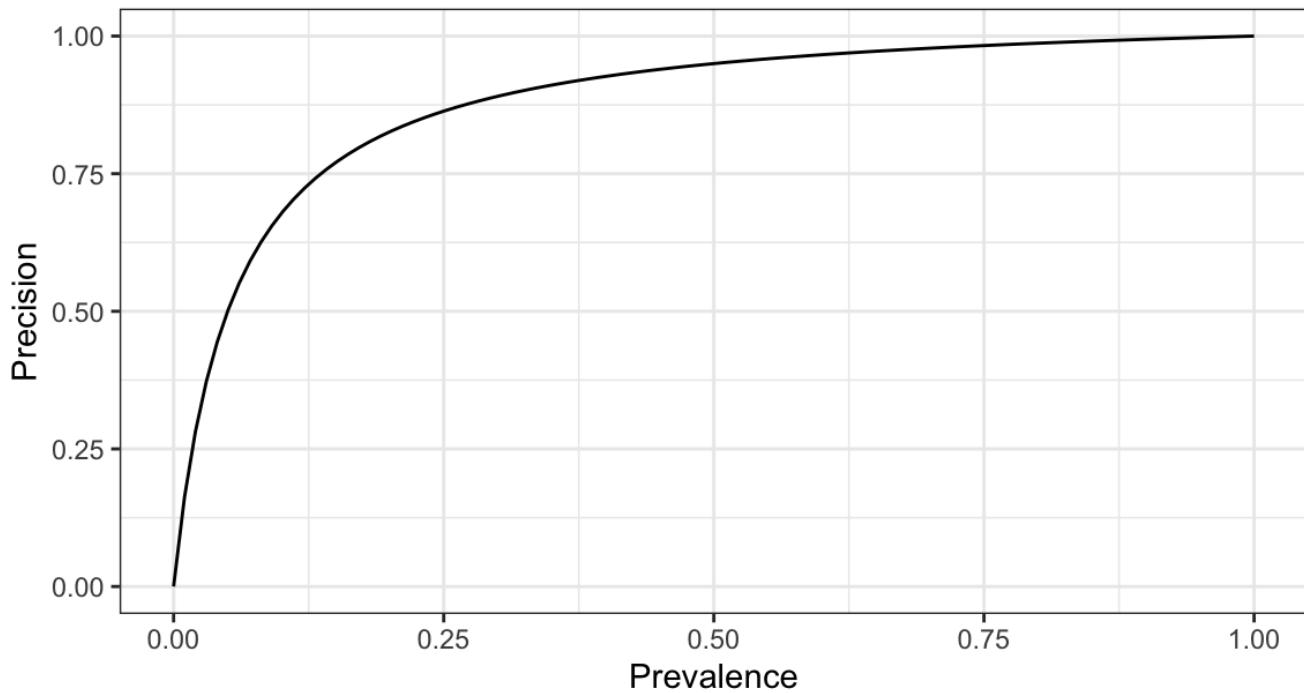


Figure 3:

A widely used plot that does this is the *receiver operating characteristic* (ROC) curve. If you are wondering where this name comes from, you can consult the ROC Wikipedia page¹.

The ROC curve plots sensitivity, represented as the TPR, versus 1 - specificity represented as the false positive rate (FPR). Here we compute the TPR and FPR needed for different probabilities of guessing male:

```
probs <- seq(0, 1, length.out = 10)
guessing <- sapply(probs, function(p){
  y_hat <-
    sample(c("Male", "Female"), nrow(test_set), TRUE, c(p, 1 - p)) |>
    factor(levels = c("Female", "Male"))
  c(FPR = 1 - specificity(y_hat, test_set$sex),
    TPR = sensitivity(y_hat, test_set$sex))
})
```

We can use similar code to compute these values for our second approach. By plotting both curves together, we are able to compare sensitivity for different values of specificity:

We see that we obtain higher sensitivity with this approach for all values of specificity, which implies it is in fact a better method. Keep in mind that ROC curves for guessing always fall on the identity line. Also, note that when making ROC curves, it is often nice to add the cutoff associated with each point.

The packages **pROC** and **plotROC** are useful for generating these plots.

ROC curves have one weakness and it is that neither of the measures plotted depends on prevalence. In cases in which prevalence matters, we may instead make a precision-recall plot. The idea is similar, but we instead plot precision against recall:

From the plot on the left, we immediately see that the precision of guessing is not high. This is because the prevalence is low. From the plot on the right, we also see that if we change $\backslash(Y=1\backslash)$ to mean **Male** instead of **Female**, the precision increases. Note that the ROC curve would remain the same.

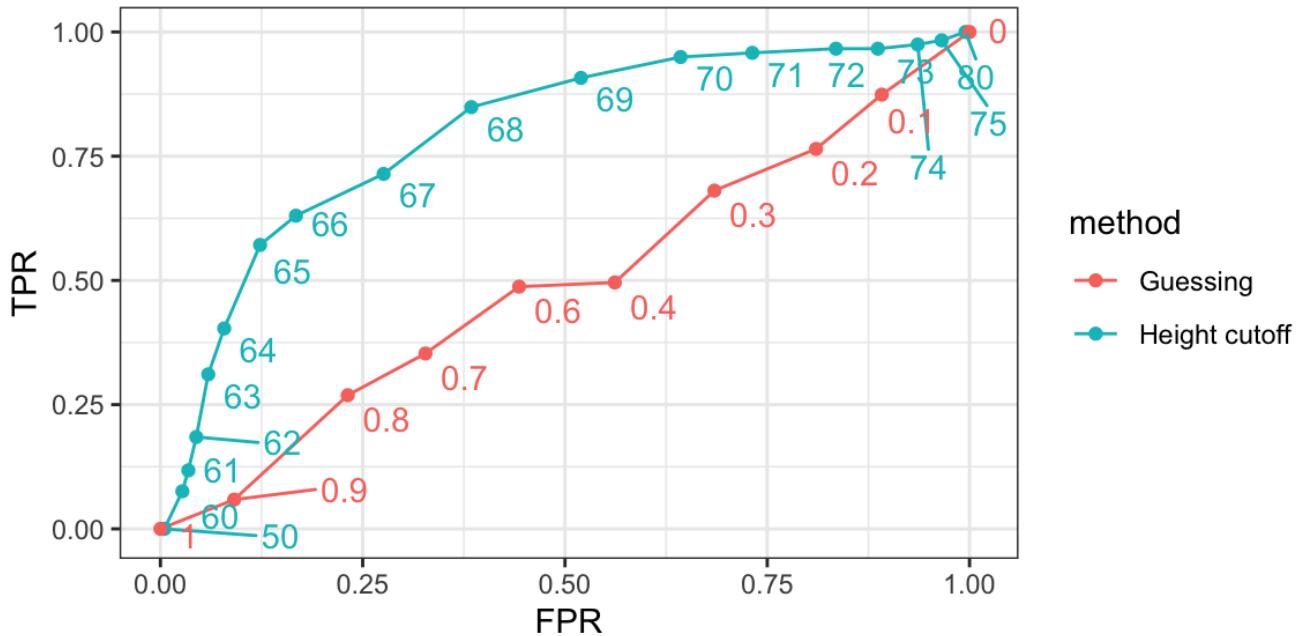


Figure 4:

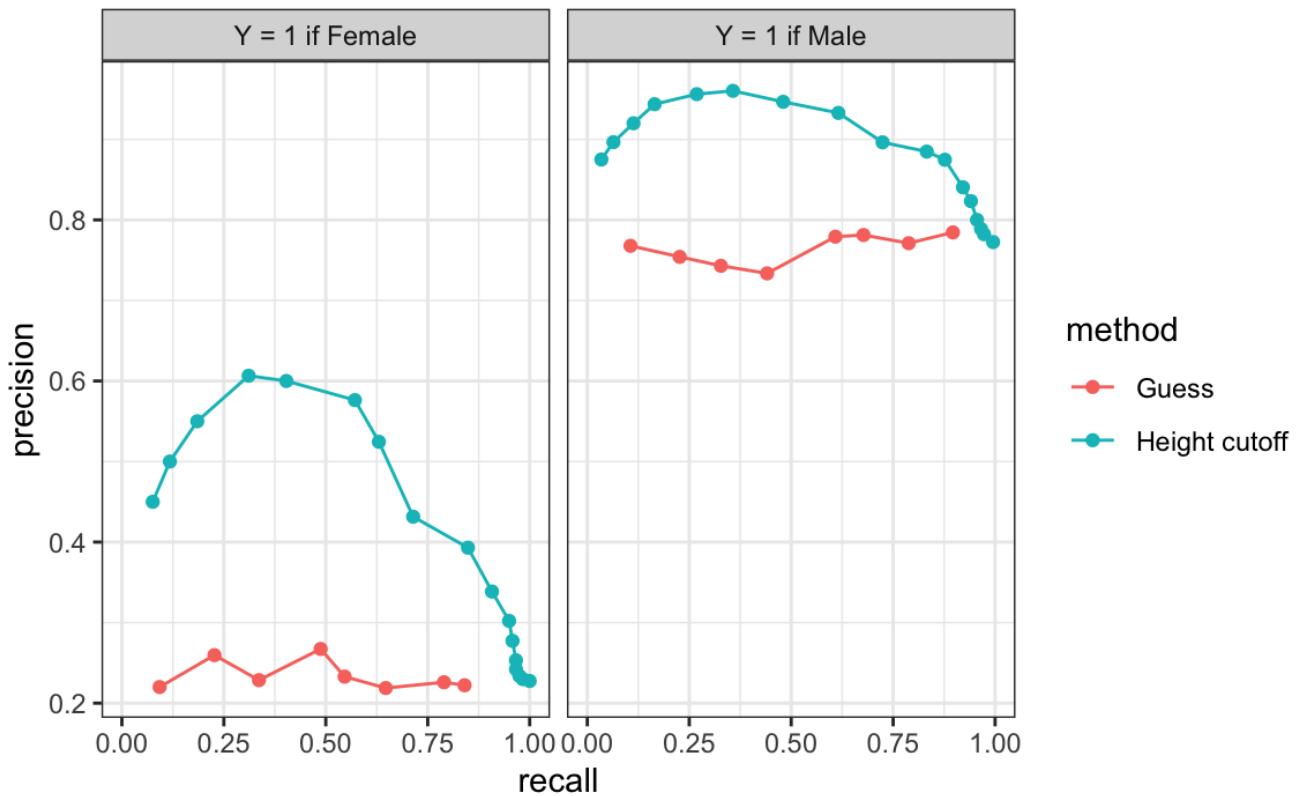


Figure 5:

26.8 Mean Squared Error

Up to now we have described evaluation metrics that apply exclusively to categorical data. Specifically, for binary outcomes, we have described how sensitivity, specificity, accuracy, and \hat{F}_1 can be used as quantification. However, these metrics are not useful for continuous outcomes.

In this section, we describe how the general approach to defining “best” in machine learning is to define a *loss function*, which can be applied to both categorical and continuous data.

The most commonly used loss function is the squared loss function. If \hat{y} is our predictor and y is the observed outcome, the squared loss function is simply: $(\hat{y} - y)^2$.

Because we often model y as the outcome of a random process, theoretically, it does not make sense to compare algorithms based on $(\hat{y} - y)^2$ as the minimum can change from sample to sample. For this reason, we minimize mean squared error (MSE):

$$\text{MSE} \equiv \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

Consider that if the outcomes are binary, the MSE is equivalent to one minus expected accuracy, since $(\hat{y} - y)^2$ is 0 if the prediction was correct and 1 otherwise.

Different algorithms will result in different predictions \hat{y} , and therefore different MSE. In general, our goal is to build an algorithm that minimizes the loss so it is as close to 0 as possible.

However, note that the MSE is a theoretical quantity. How do we estimate this? Because in practice we have tests set with many, say N , independent observations, a commonly used observable estimate of the MSE is:

$$\hat{\text{MSE}} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

with the \hat{y}_i generated completely independently from the y_i .

In practice, we often report the root mean squared error (RMSE), which is simply $\sqrt{\text{MSE}}$, because it is in the same units as the outcomes.

However, the estimate $\hat{\text{MSE}}$ is a random variable. In fact, MSE and $\hat{\text{MSE}}$ are often referred to as the true error and apparent error, respectively. Due to the complexity of some machine learning, it is difficult to derive the statistical properties of how well the apparent error estimates the true error. In Chapter 29, we introduce cross-validation an approach to estimating the MSE.

We end this chapter by pointing out that there are loss functions other than the squared loss. For example, the *Mean Absolute Error* uses absolute values, $|\hat{y}_i - y_i|$ instead of squaring the errors $(\hat{y}_i - y_i)^2$. However, in this book we focus on minimizing square loss since it is the most widely used.

26.9 Exercises

The `reported_height` and `height` datasets were collected from three classes taught in the Departments of Computer Science and Biostatistics, as well as remotely through the Extension School. The Biostatistics class was taught in 2016 along with an online version offered by the Extension School. On 2016-01-25 at 8:15 AM, during one of the lectures, the instructors asked students to fill in the sex and height questionnaire that populated the `reported_height` dataset. The online students filled the survey during the next few days, after the lecture was posted online. We can use this insight to define a variable, call it `type`, to denote the type of student: `inclass` or `online`:

```
library(lubridate)
dat <- mutate(reported_heights, date_time = ymd_hms(time_stamp)) |>
  filter(date_time >= make_date(2016, 01, 25) &
         date_time < make_date(2016, 02, 1)) |>
  mutate(type = ifelse(day(date_time) == 25 & hour(date_time) == 8 &
                        between(minute(date_time), 15, 30),
                        "inclass", "online")) |> select(sex, type)

x <- dat$type
y <- factor(dat$sex, c("Female", "Male"))
```

1. Show summary statistics that indicate that the `type` is predictive of sex.
 2. Instead of using height to predict sex, use the `type` variable.
 3. Show the confusion matrix.
 4. Use the `confusionMatrix` function in the `caret` package to report accuracy.
 5. Now use the `sensitivity` and `specificity` functions to report specificity and sensitivity.
 6. What is the prevalence (% of females) in the `dat` dataset defined above?
-

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

Introduction to Data Science - 27 Conditional probabilities and expectations

Rafael A. Irizarry

In machine learning applications, we rarely can predict outcomes perfectly. For example, spam detectors often miss emails that are clearly spam, Siri often misunderstands the words we are saying, and sometimes your bank thinks your card was stolen when it was not. The most common reason for not being able to build perfect algorithms is that it is impossible. To see this, consider that most datasets will include groups of observations with the same exact observed values for all predictors, but with different outcomes.

Because our prediction rules are functions, equal inputs (the predictors) implies equal outputs (the predictions). Therefore, for a challenge in which the same predictors are associated with different outcomes across different individual observations, it is impossible to predict correctly for all these cases. We saw a simple example of this in the previous section: for any given height $\langle x \rangle$, you will have both males and females that are $\langle x \rangle$ inches tall.

However, none of this means that we can't build useful algorithms that are much better than guessing, and in some cases better than expert opinions. To achieve this in an optimal way, we make use of probabilistic representations of the problem based on the ideas presented in Section 14.3. Observations with the same observed values for the predictors may not all be the same, but we can assume that they all have the same probability of this class or that class. We will write this idea out mathematically for the case of categorical data.

27.1 Conditional probabilities

We use the notation $\langle (X_1 = x_1, \dots, X_p = x_p) \rangle$ to represent the fact that we have observed values $\langle x_1, \dots, x_p \rangle$ for covariates $\langle X_1, \dots, X_p \rangle$. This does not imply that the outcome $\langle Y \rangle$ will take a specific value. Instead, it implies a specific probability. In particular, we denote the *conditional probabilities* for each class $\langle k \rangle$ with:

$$\langle [\Pr\{Y=k \mid X_1 = x_1, \dots, X_p = x_p\}, \dots, \Pr\{Y=k \mid X_1 = x_1, \dots, X_p = x_p\}] \rangle$$

To avoid writing out all the predictors, we will use the bold letters like this: $\langle (\mathbf{X} \equiv (X_1, \dots, X_p))^{\text{top}} \rangle$ and $\langle (\mathbf{x} \equiv (x_1, \dots, x_p))^{\text{top}} \rangle$. We will also use the following notation for the conditional probability of being class $\langle k \rangle$:

$\langle p_k(\mathbf{x}) \rangle = \Pr\{Y=k \mid \mathbf{X}=\mathbf{x}\}$, $\langle [\Pr\{Y=k \mid \mathbf{X}=\mathbf{x}\}, \dots, \Pr\{Y=k \mid \mathbf{X}=\mathbf{x}\}] \rangle$ Notice that the $\langle p_k(\mathbf{x}) \rangle$ have to add up to 1 for each $\langle \mathbf{x} \rangle$, so once we know $\langle K-1 \rangle$, we know all. When the outcome is binary, we only need to know 1, so we drop the $\langle k \rangle$ and use the notation $\langle p(\mathbf{x}) \rangle = \Pr\{Y=1 \mid \mathbf{X}=\mathbf{x}\}$.

Do not be confused by the fact that we use $\langle p \rangle$ for two different things: the conditional probability $\langle p(\mathbf{x}) \rangle$ and the number of predictors $\langle p \rangle$.

These probabilities guide the construction of an algorithm that makes the best prediction: for any given $\langle \mathbf{x} \rangle$, we will predict the class $\langle k \rangle$ with the largest probability among $\langle p_1(\mathbf{x}), p_2(\mathbf{x}), \dots, p_K(\mathbf{x}) \rangle$. In mathematical notation, we write it like this:

$$\langle \hat{Y} = \max_k p_k(\mathbf{x}) \rangle$$

In machine learning, we refer to this as *Bayes' Rule*. But this is a theoretical rule since, in practice, we don't know $\{p_k(\mathbf{x})\}_{k=1}^K$. In fact, estimating these conditional probabilities can be thought of as the main challenge of machine learning. The better our probability estimates $\{\hat{p}_k(\mathbf{x})\}$, the better our predictor $\{\hat{Y}\}$.

So how well we predict depends on two things: 1) how close are the $\{\max_k p_k(\mathbf{x})\}$ to 1 or 0 (perfect certainty) and 2) how close our estimates $\{\hat{p}_k(\mathbf{x})\}$ are to $\{p_k(\mathbf{x})\}$. We can't do anything about the first restriction as it is determined by the nature of the problem, so our energy goes into finding ways to best estimate conditional probabilities.

The first restriction does imply that we have limits as to how well even the best possible algorithm can perform. You should get used to the idea that while in some challenges we will be able to achieve almost perfect accuracy, with digit readers for example, in others, our success is restricted by the randomness of the process, such as with movie recommendations.

Keep in mind that defining our prediction by maximizing the probability is not always optimal in practice and depends on the context. As discussed in Chapter 26, sensitivity and specificity may differ in importance. But even in these cases, having a good estimate of the $\{p_k(\mathbf{x})\}_{k=1}^K$ will suffice for us to build optimal prediction models, since we can control the balance between specificity and sensitivity however we wish. For instance, we can simply change the cutoffs used to predict one outcome or the other. In the plane example, we may ground the plane anytime the probability of malfunction is higher than 1 in a million as opposed to the default 1/2 used when error types are equally undesired.

27.2 Conditional expectations

For binary data, you can think of the probability $\Pr(Y=1 \mid \mathbf{X}=\mathbf{x})$ as the proportion of 1s in the stratum of the population for which $\{\mathbf{X}=\mathbf{x}\}$. Many of the algorithms we will learn can be applied to both categorical and continuous data due to the connection between *conditional probabilities* and *conditional expectations*.

Because the expectation is the average of values $\{y_1, \dots, y_n\}$ in the population, in the case in which the $\{y_i\}$ s are 0 or 1, the expectation is equivalent to the probability of randomly picking a one since the average is simply the proportion of ones:

$$\mathbb{E}(Y \mid \mathbf{X}=\mathbf{x}) = \Pr(Y=1 \mid \mathbf{X}=\mathbf{x}).$$

As a result, we often only use the expectation to denote both the conditional probability and conditional expectation.

Just like with categorical outcomes, in most applications the same observed predictors do not guarantee the same continuous outcomes. Instead, we assume that the outcome follows the same conditional distribution. We will now explain why we use the conditional expectation to define our predictors.

27.3 Conditional expectations minimizes squared loss function

Why do we care about the conditional expectation in machine learning? This is because the expected value has an attractive mathematical property: it minimizes the MSE. Specifically, of all possible predictions $\{\hat{Y}\}$,

$$\mathbb{E}(\hat{Y}) = \mathbb{E}(Y \mid \mathbf{X}=\mathbf{x}), \quad \text{minimizes } \mathbb{E}((\hat{Y} - Y)^2 \mid \mathbf{X}=\mathbf{x})$$

Due to this property, a succinct description of the main task of machine learning is that we use data to estimate:

$$f(\mathbf{x}) \equiv \mathbb{E}(Y \mid \mathbf{X}=\mathbf{x})$$

for any set of features $\{\mathbf{x} = (x_1, \dots, x_p)\}$.

This is easier said than done, since this function can take any shape and $\{p\}$ can be very large. Consider a case in which we only have one predictor $\{x\}$. The expectation $\{\mathbb{E}(Y \mid X=x)\}$ can be any function of $\{x\}$: a line, a parabola, a sine wave, a step function, anything. It gets even more complicated when we consider instances with large $\{p\}$, in which case $\{f(\mathbf{x})\}$ is a function of a multidimensional vector $\{\mathbf{x}\}$. For example, in our digit reader example $\{p = 784\}!$

The main way in which competing machine learning algorithms differ is in their approach to estimating this conditional expectation.

27.4 Exercises

1. Compute conditional probabilities for being Male for the `heights` dataset. Round the heights to the closest inch. Plot the estimated conditional probability $\Pr(\text{Male} | \text{height}=x)$ for each x .
2. In the plot we just made, we see high variability for low values of height. This is because we have few data points in these strata. This time use the `quantile` function for quantiles $(0.1, 0.2, \dots, 0.9)$ and the `cut` function to assure each group has the same number of points. Hint: For any numeric vector x , you can create groups based on quantiles as we demonstrate below.

```
cut(x, quantile(x, seq(0, 1, 0.1)), include.lowest = TRUE)
```

3. Generate data from a bivariate normal distribution using the **MASS** package like this:

```
Sigma <- 9*matrix(c(1,0.5,0.5,1), 2, 2)
dat <- MASS::mvrnorm(n = 10000, c(69, 69), Sigma) |>
  data.frame() |> setNames(c("x", "y"))
```

You can make a quick plot of the data using `plot(dat)`. Use an approach similar to the previous exercise to estimate the conditional expectations and make a plot.

Introduction to Data Science - 28 Smoothing

Rafael A. Irizarry

Before continuing learning about machine learning algorithms, we introduce the important concept of *smoothing*. Smoothing is a very powerful technique used all across data analysis. Other names given to this technique are *curve fitting* and *low pass filtering*. It is designed to detect trends in the presence of noisy data in cases in which the shape of the trend is unknown. The *smoothing* name comes from the fact that to accomplish this feat, we assume that the trend is *smooth*, as in a smooth surface. In contrast, the noise, or deviation from the trend, is unpredictably wobbly:

Part of what we explain in this section are the assumptions that permit us to extract the trend from the noise.

28.1 Example: Is it a 2 or a 7?

To motivate the need for smoothing and make the connection with machine learning, we will construct a simplified version of the MNIST dataset with just two classes for the outcome and two predictors. Specifically, we define the challenge as building an algorithm that can determine if a digit is a 2 or 7 from the proportion of dark pixels in the upper left quadrant ((X_1)) and the lower right quadrant ((X_2)). We also selected a random sample of 1,000 digits, 500 in the training set and 500 in the test set. We provide this dataset in the `mnist_27` object in the `dslabs` package.

For the training data, we have $(n=500)$ observed outcomes (y_1, \dots, y_n) , with (Y) defined as (1) if the digit is 7 and 0 if it's 2, and $(n=500)$ features $(\mathbf{x}_1, \dots, \mathbf{x}_n)$, with each feature a two-dimensional point $(\mathbf{x}_i = (x_{i,1}, x_{i,2}))$. Here is a plot of the (x_2) s versus the (x_1) s with color determining if (y) is 1 (blue) or 0 (red):

```
library(caret)
library(dslabs)
mnist_27$train |> ggplot(aes(x_1, x_2, color = y)) + geom_point()
```

We can immediately see some patterns. For example, if (x_1) (the upper left panel) is very large, then the digit is probably a 7. Also, for smaller values of (x_1) , the 2s appear to be in the mid range values of (x_2) .

To illustrate how to interpret (x_1) and (x_2) , we include four example images. On the left are the original images of the two digits with the largest and smallest values for (x_1) and on the right we have the images corresponding to the largest and smallest values of (x_2) :

We can start getting a sense for why these predictors are useful, but also why the problem will be somewhat challenging.

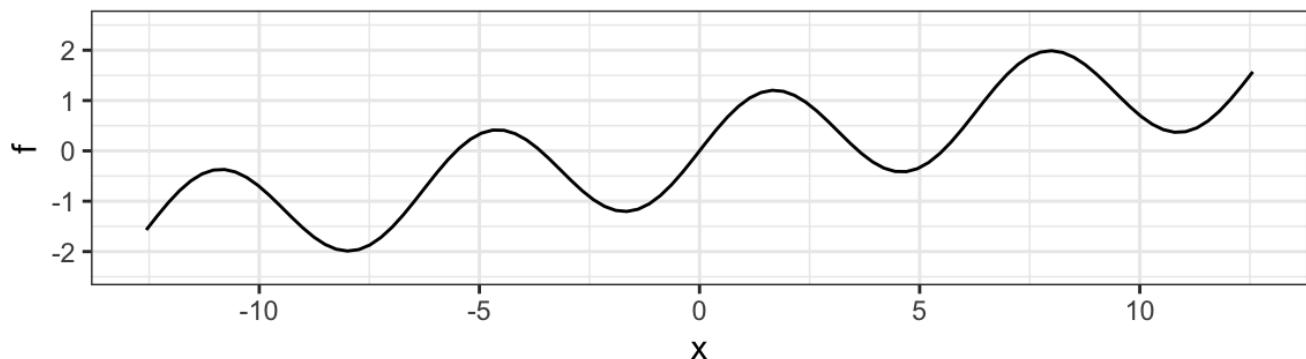
We haven't really learned any algorithms yet, so let's try building an algorithm using multivariable regression. The model is simply:

$$p(\mathbf{x}) = \Pr(Y=1 \mid \mathbf{X}=\mathbf{x}) = \Pr(Y=1 \mid X_1=x_1, X_2=x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

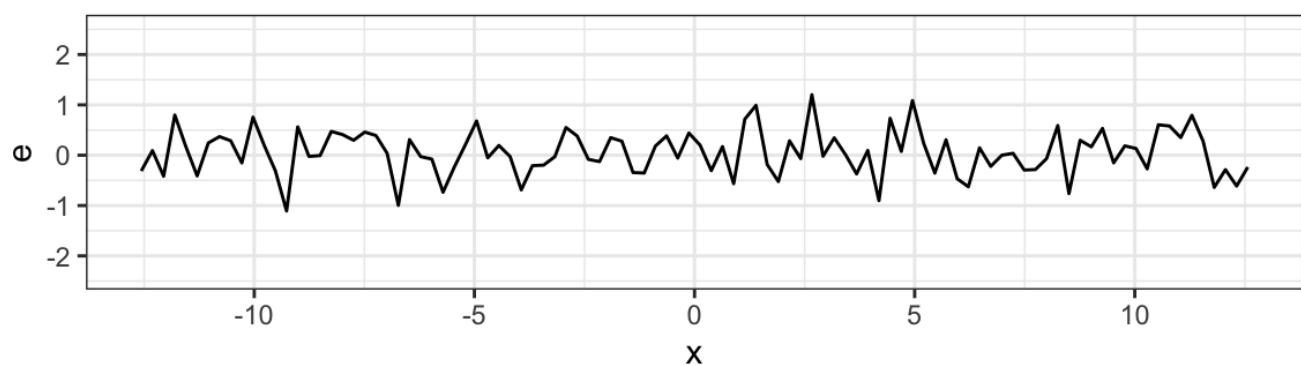
We fit this model using least squares and obtain an estimate $(\hat{p}(\mathbf{x}))$ by using the least square estimates $(\hat{\beta}_0)$, $(\hat{\beta}_1)$ and $(\hat{\beta}_2)$. We define a decision rule by predicting $(\hat{y}=1)$ if $(\hat{p}(\mathbf{x}) > 0.5)$ and 0 otherwise.

We get an accuracy of 0.775, well above 50%. Not bad for our first try. But can we do better?

smooth trend



noise



data = smooth trend + noise

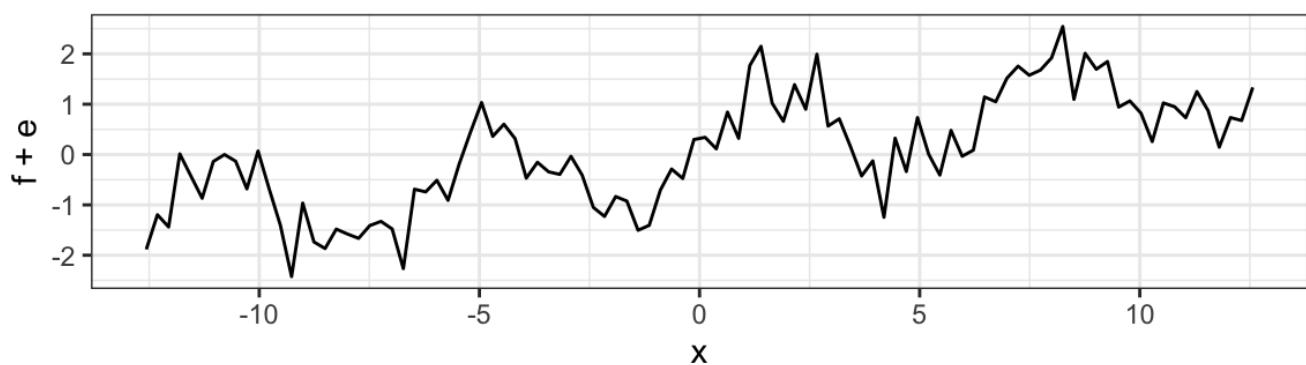


Figure 1:

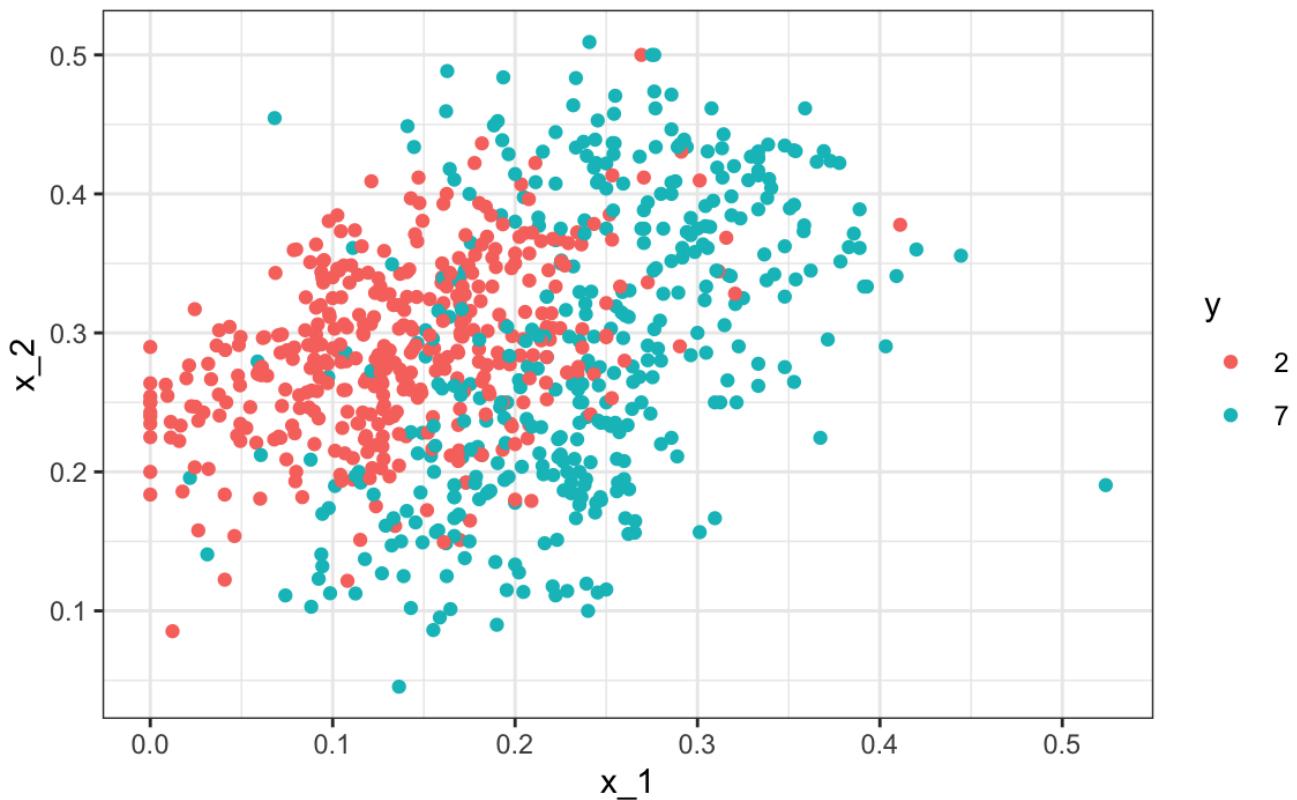


Figure 2:

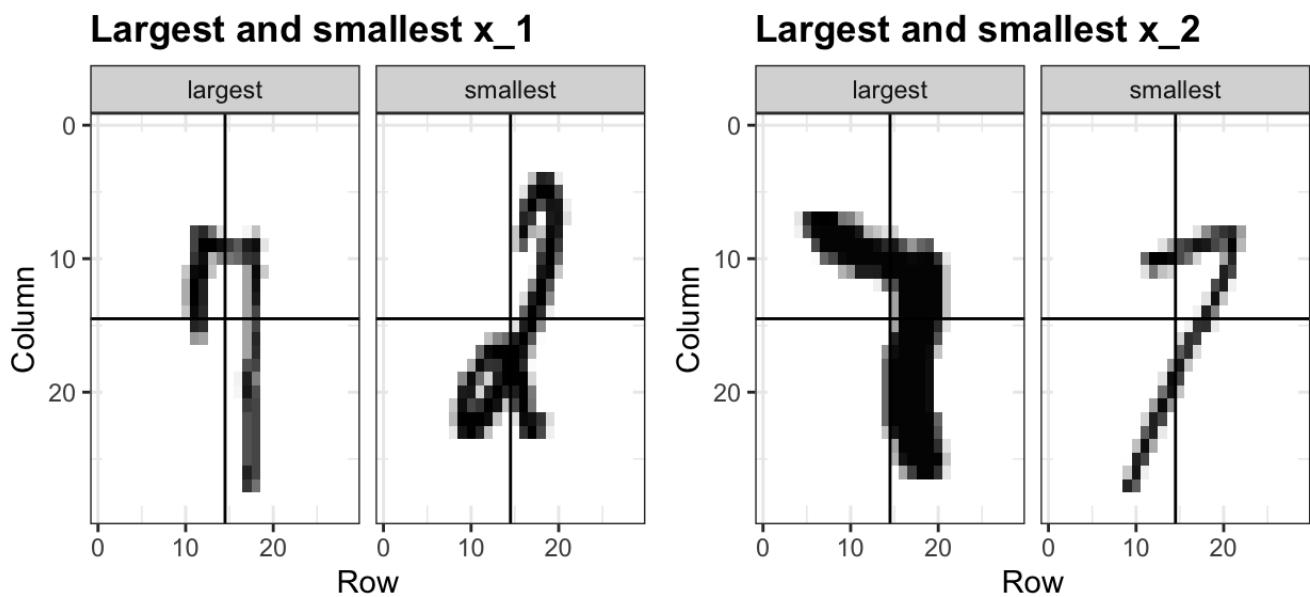


Figure 3:

Because we constructed the `mnist_27` example and we had at our disposal 60,000 digits in just the MNIST dataset, we used this to build the *true* conditional distribution $(p(\mathbf{x}))$. Keep in mind that in practice we don't have access to the true conditional distribution. We include it in this educational example because it permits the comparison of $(\hat{p}(\mathbf{x}))$ to the true $(p(\mathbf{x}))$. This comparison teaches us the limitations of different algorithms. We have stored the true $(p(\mathbf{x}))$ in the `mnist_27` and can plot it as an image. We draw a curve that separates pairs $((\mathbf{x}))$ for which $(p(\mathbf{x}) > 0.5)$ and pairs for which $(p(\mathbf{x}) < 0.5)$:

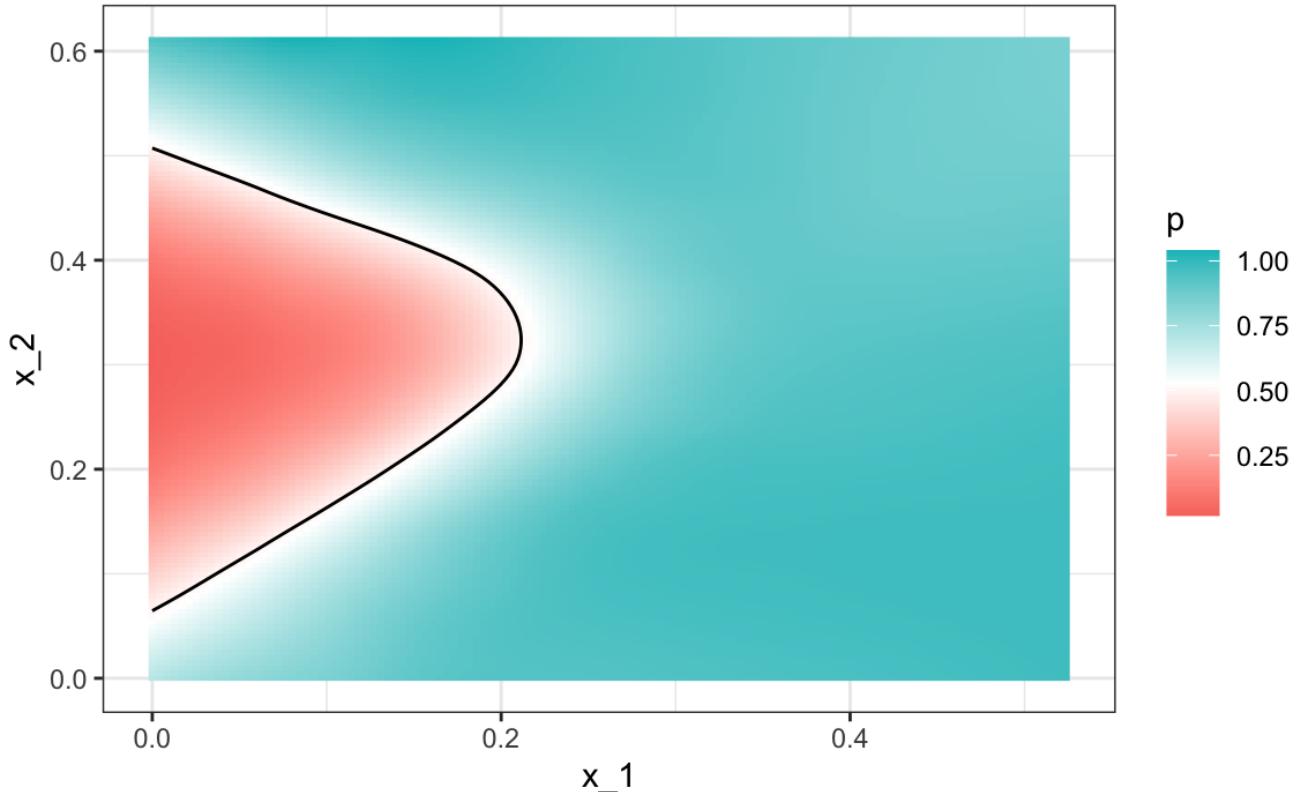


Figure 4:

To start understanding the limitations of regression, first note that with regression $(\hat{p}(\mathbf{x}))$ has to be a plane, and as a result the boundary defined by the decision rule is given by: $(\hat{p}(\mathbf{x}) = 0.5)$:

$$[\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0.5 \text{ implies } \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0.5 \text{ implies } x_2 = (0.5 - \hat{\beta}_0)/\hat{\beta}_2 - \hat{\beta}_1/\hat{\beta}_2 x_1]$$

This implies that for the boundary, (x_2) is a linear function of (x_1) , which suggests that our regression approach has no chance of capturing the non-linear nature of the true $(p(\mathbf{x}))$. Below is a visual representation of $(\hat{p}(\mathbf{x}))$ which clearly shows how it fails to capture the shape of $(p(\mathbf{x}))$:

We need something more flexible: a method that permits estimates with shapes other than a plane. Smoothing techniques permit this flexibility. We will start by describing nearest neighbor and kernel approaches. To understand why we cover this topic, remember that the concepts behind smoothing techniques are extremely useful in machine learning because conditional expectations/probabilities can be thought of as *trends* of unknown shapes that we need to estimate in the presence of uncertainty.

28.2 Signal plus noise model

To explain these concepts, we will focus first on a problem with just one predictor. Specifically, we try to estimate the time trend in the 2008 US popular vote poll margin (the difference between Obama and McCain). Later we will learn about methods, such as k-nearest neighbors, that can be used to smooth with higher dimensions.

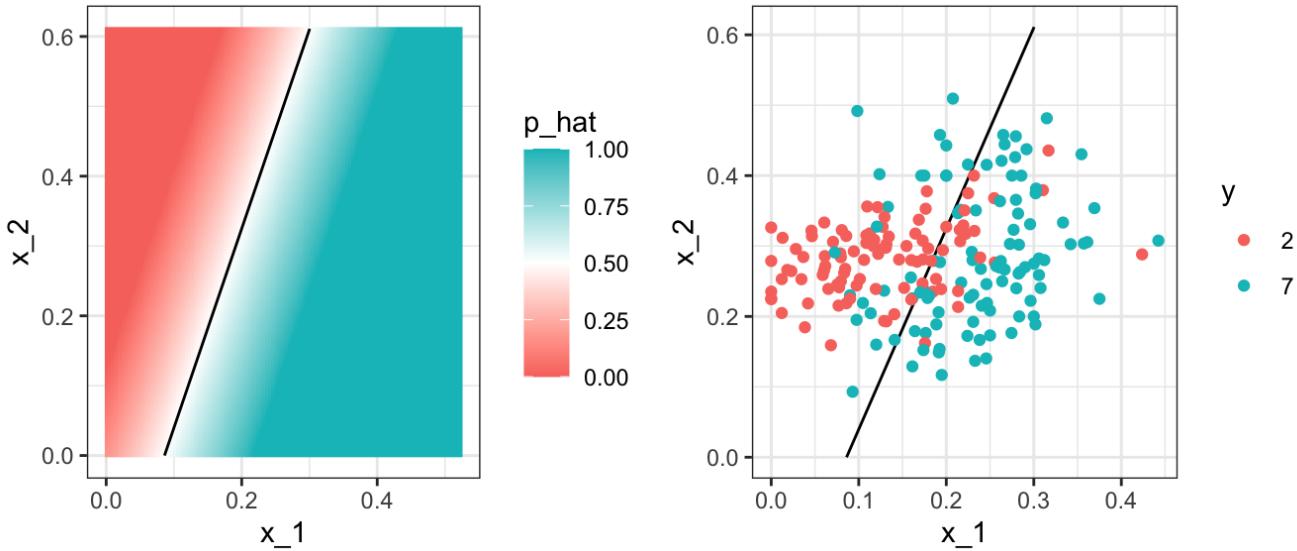


Figure 5:

```
polls_2008 |> ggplot(aes(day, margin)) + geom_point()
```

For the purposes of the popular vote example, do not think of it as a forecasting problem. Instead, we are simply interested in learning the shape of the trend *after* the election is over.

We assume that for any given day $\langle x \rangle$, there is a true preference among the electorate $\langle f(x) \rangle$, but due to the uncertainty introduced by the polling, each data point comes with an error $\langle varepsilon \rangle$. A mathematical model for the observed poll margin $\langle Y_i \rangle$ is:

$$\langle Y_i \rangle = f(x_i) + varepsilon_i$$

To think of this as a machine learning problem, consider that we want to predict $\langle Y \rangle$ given a day $\langle x \rangle$. If we knew the conditional expectation $\langle f(x) = \text{mbox}\{E\}(Y \mid X=x) \rangle$, we would use it. But since we don't know this conditional expectation, we have to estimate it. Let's use regression, since it is the only method we have learned up to now.

```
#> `geom_smooth()` using formula = 'y ~ x'
```

The fitted regression line does not appear to describe the trend very well. For example, on September 4 (day -62), the Republican Convention was held and the data suggest that it gave John McCain a boost in the polls. However, the regression line does not capture this potential trend. To see the *lack of fit* more clearly, we note that points above the fitted line (blue) and those below (red) are not evenly distributed across days. We therefore need an alternative, more flexible approach.

28.3 Bin smoothing

The general idea of smoothing is to group data points into strata in which the value of $\langle f(x) \rangle$ can be assumed to be constant. We can make this assumption when we think $\langle f(x) \rangle$ changes slowly and, as a result, $\langle f(x) \rangle$ is almost constant in small windows of $\langle x \rangle$. An example of this idea for the `poll_2008` data is to assume that public opinion remained approximately the same within a week's time. With this assumption in place, we have several data points with the same expected value.

If we fix a day to be in the center of our week, call it $\langle x_0 \rangle$, then for any other day $\langle x \rangle$ such that $\langle |x - x_0| \leq 3.5 \rangle$, we assume $\langle f(x) \rangle$ is a constant $\langle f(x) = \mu \rangle$. This assumption implies that:

$$\langle E[Y_i \mid X_i = x_i] \rangle \approx \mu \text{ if } |x_i - x_0| \leq 3.5$$

In smoothing, we call the size of the interval satisfying $\langle |x_i - x_0| \leq 3.5 \rangle$ the *window size*, *bandwidth* or *span*. Later we will see that we try to optimize this parameter.

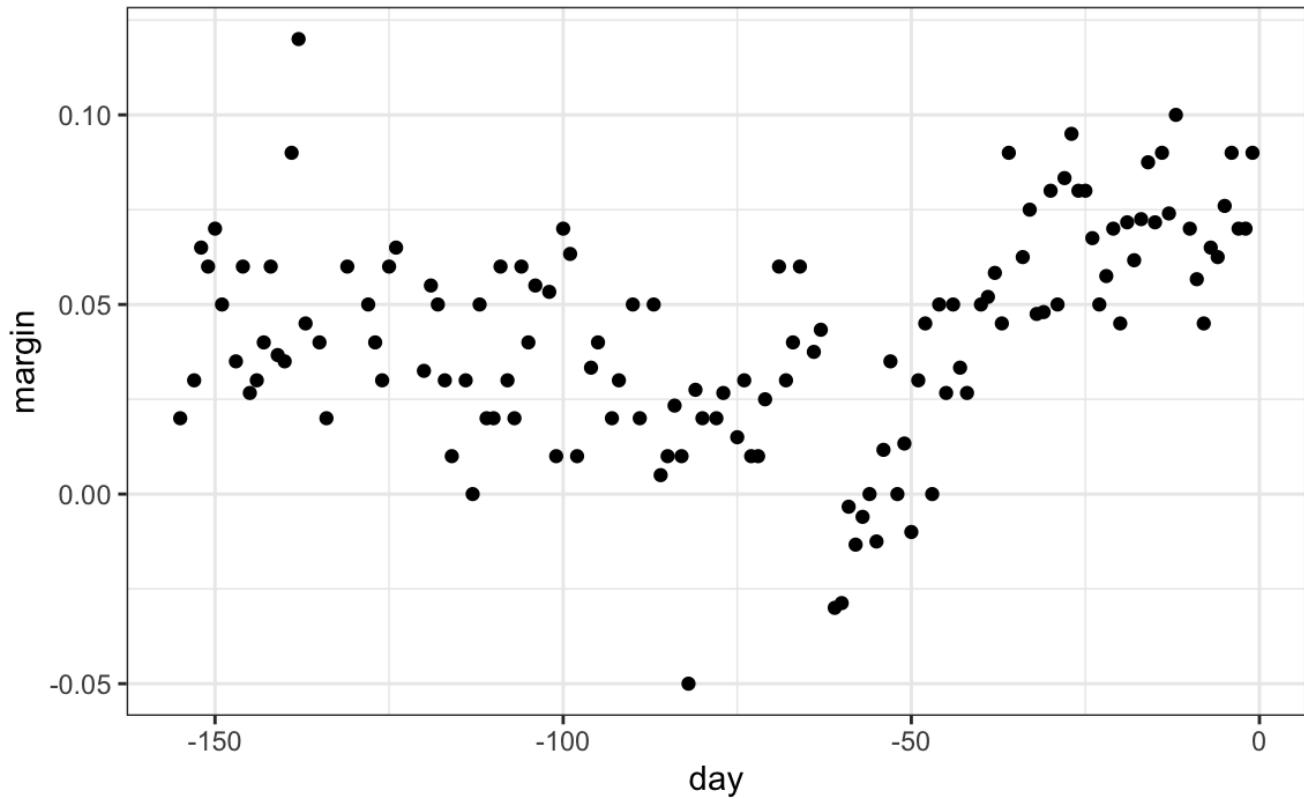


Figure 6:

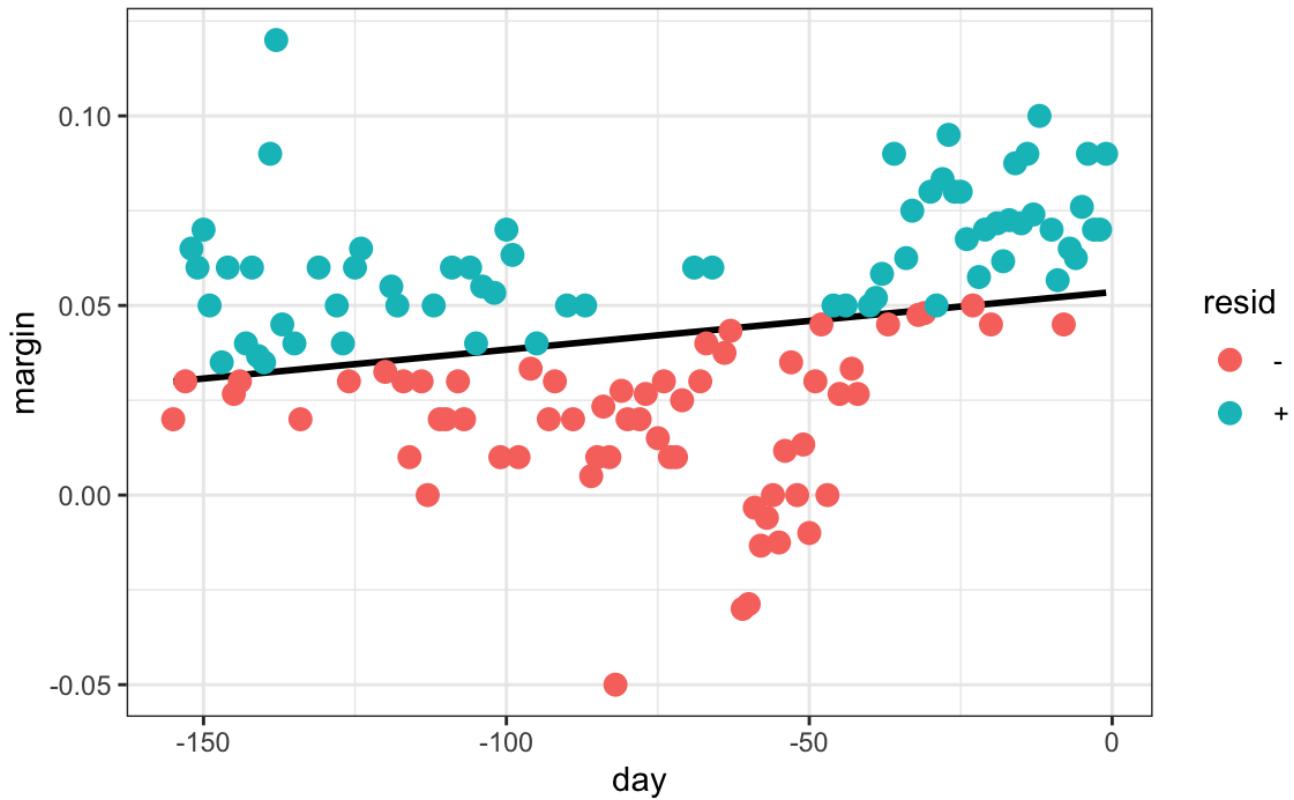


Figure 7:

This assumption implies that a good estimate for $\langle f(x_0) \rangle$ is the average of the $\langle Y_i \rangle$ values in the window. If we define (A_0) as the set of indexes (i) such that $(|x_i - x_0| \leq 3.5)$ and (N_0) as the number of indexes in (A_0) , then our estimate is:

$$\hat{f}(x_0) = \frac{1}{N_0} \sum_{i \in A_0} Y_i$$

We make this calculation with each value of (x) as the center. In the poll example, for each day, we would compute the average of the values within a week with that day in the center. Here are two examples: $(x_0 = -125)$ and $(x_0 = -55)$. The blue segment represents the resulting average.

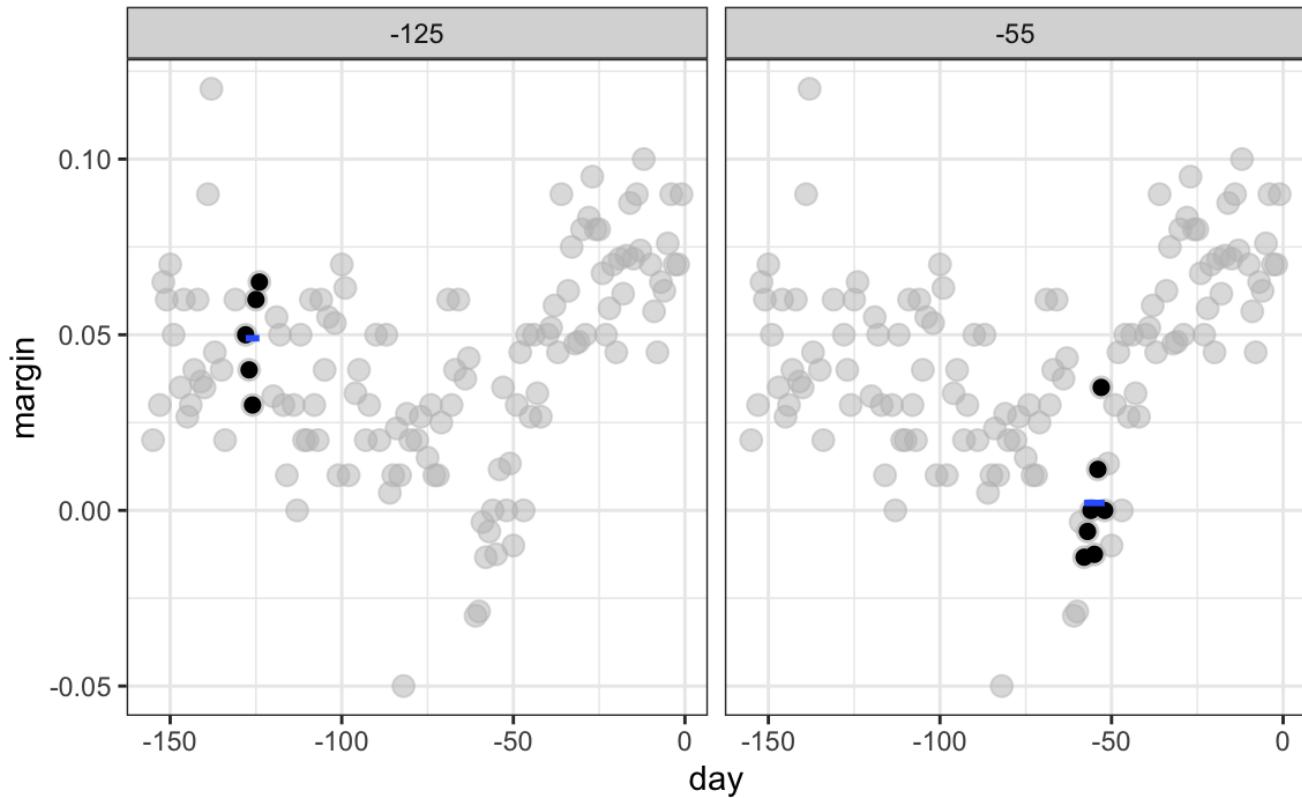


Figure 8:

By computing this mean for every point, we form an estimate of the underlying curve $\langle f(x) \rangle$. Below we show the procedure happening as we move from the -155 up to 0. At each value of (x_0) , we keep the estimate $(\langle \hat{f}(x_0) \rangle)$ and move on to the next point:

The final code and resulting estimate look like this:

```
span <- 7
fit <- with(polls_2008, ksmooth(day, margin, kernel = "box", bandwidth = span))

polls_2008 |> mutate(fit = fit$y) |>
  ggplot(aes(x = day)) +
  geom_point(aes(y = margin), size = 3, alpha = .5, color = "grey") +
  geom_line(aes(y = fit), color = "red")
```

28.4 Kernels

The final result from the bin smoother is quite wiggly. One reason for this is that each time the window moves, two points change. We can attenuate this somewhat by taking weighted averages that give the center point more weight than far away points, with the two points at the edges receiving very little weight.

You can think of the bin smoother approach as a weighted average:

x0 = -155

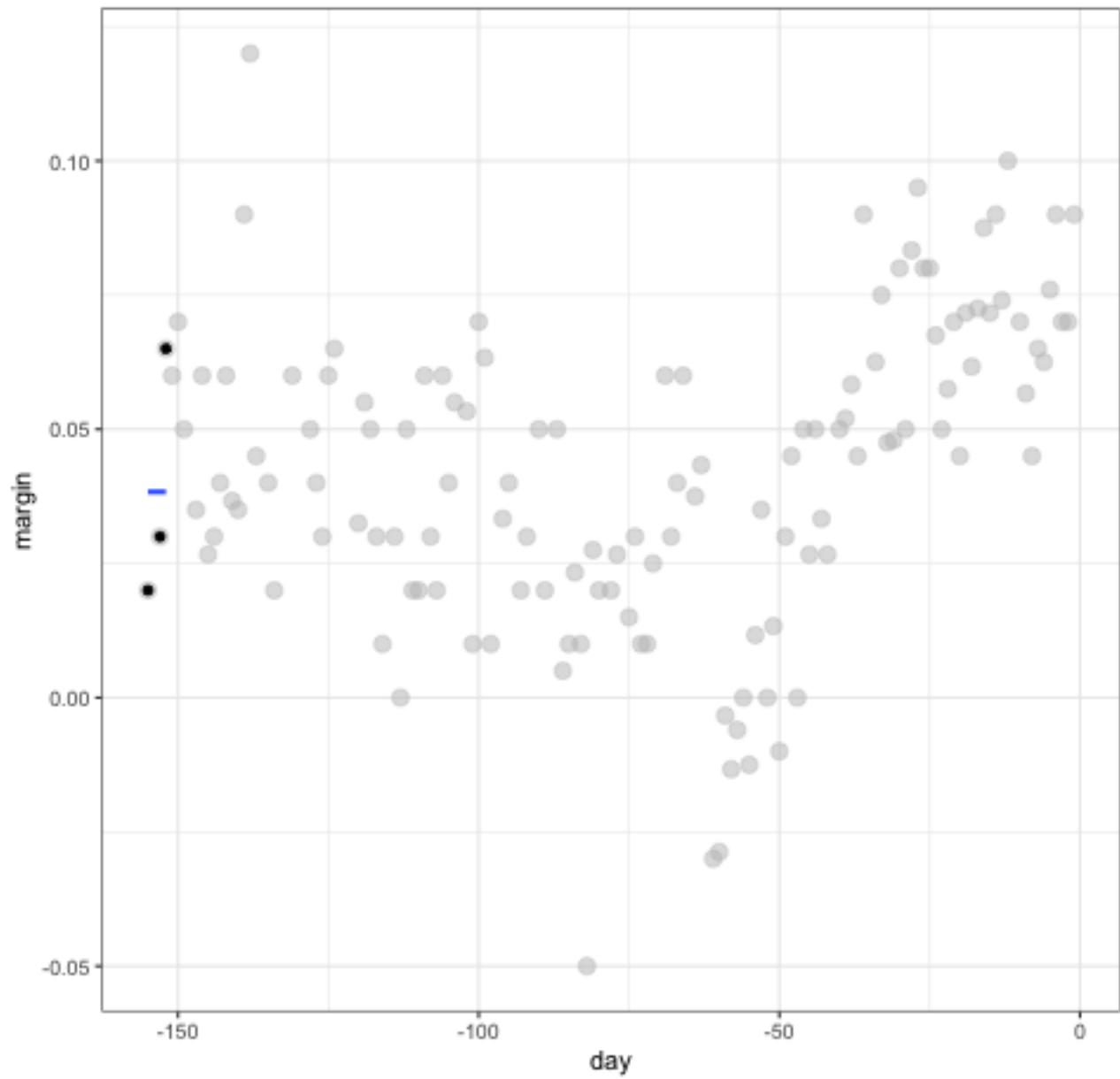


Figure 9:

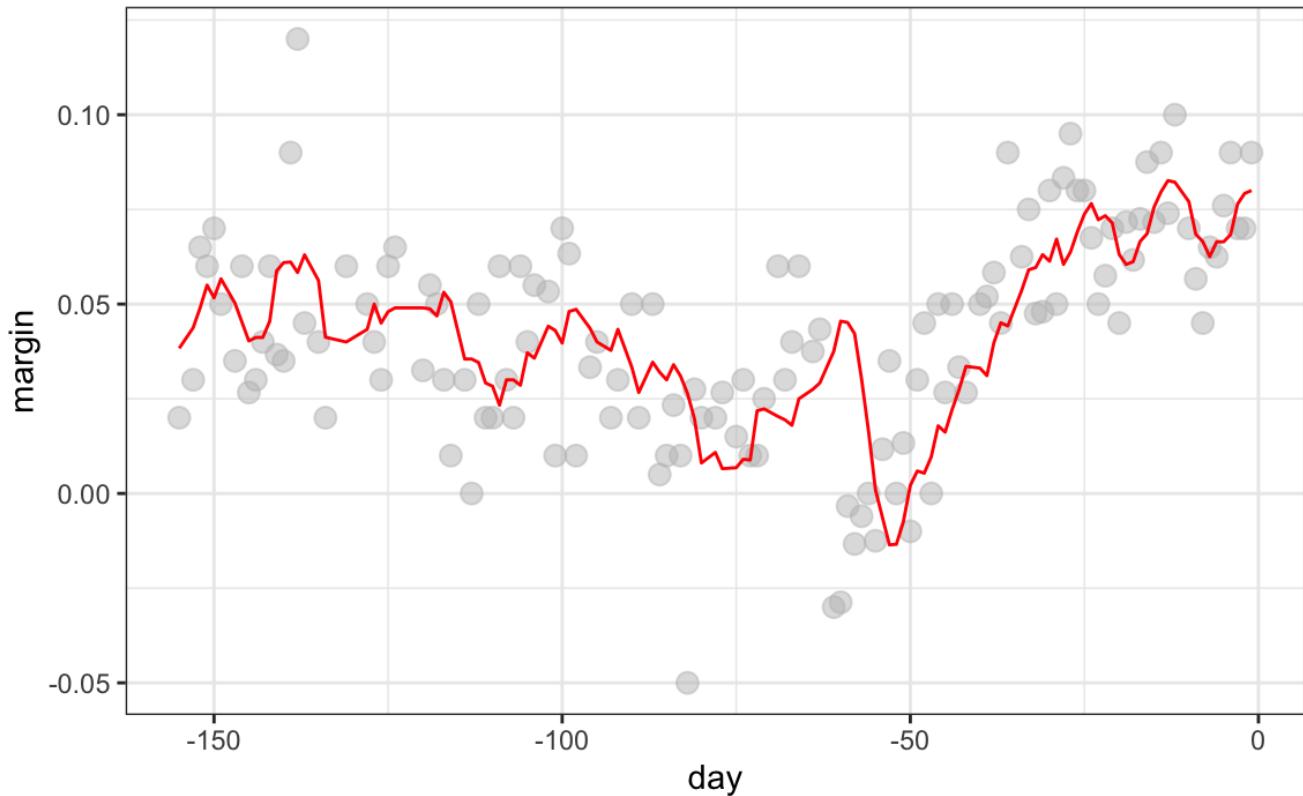


Figure 10:

$$\hat{f}(x_0) = \sum_{i=1}^N w_i(x_i) Y_i$$

in which each point receives a weight of either $\frac{1}{N}$ or $\frac{1}{N_0}$, with N_0 the number of points in the week. In the code above, we used the argument `kernel="box"` in our call to the function `ksmooth`. This is because the weight function looks like a box. The `ksmooth` function provides a “smoother” option which uses the normal density to assign weights.

The final code and resulting plot for the normal kernel look like this:

```
span <- 7
fit <- with(polls_2008, ksmooth(day, margin, kernel = "normal", bandwidth = span))

polls_2008 |> mutate(smooth = fit$y) |>
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = .5, color = "grey") +
  geom_line(aes(day, smooth), color = "red")
```

Notice that this version looks smoother.

There are several functions in R that implement bin smoothers. One example is `ksmooth`, shown above. In practice, however, we typically prefer methods that use slightly more complex models than fitting a constant. The final result above, for example, is still somewhat wiggly in parts we don’t expect it to be (between -125 and -75, for example). Methods such as `loess`, which we explain next, improve on this.

28.5 Local weighted regression (loess)

A limitation of the bin smoother approach just described is that we need small windows for the approximately constant assumptions to hold. As a result, we end up with a small number of data points to average and obtain imprecise estimates $\hat{f}(x)$. Here we describe how *local weighted regression* (loess) permits us to consider larger window sizes. To do this, we will use a mathematical result, referred to as Taylor’s theorem, which tells us

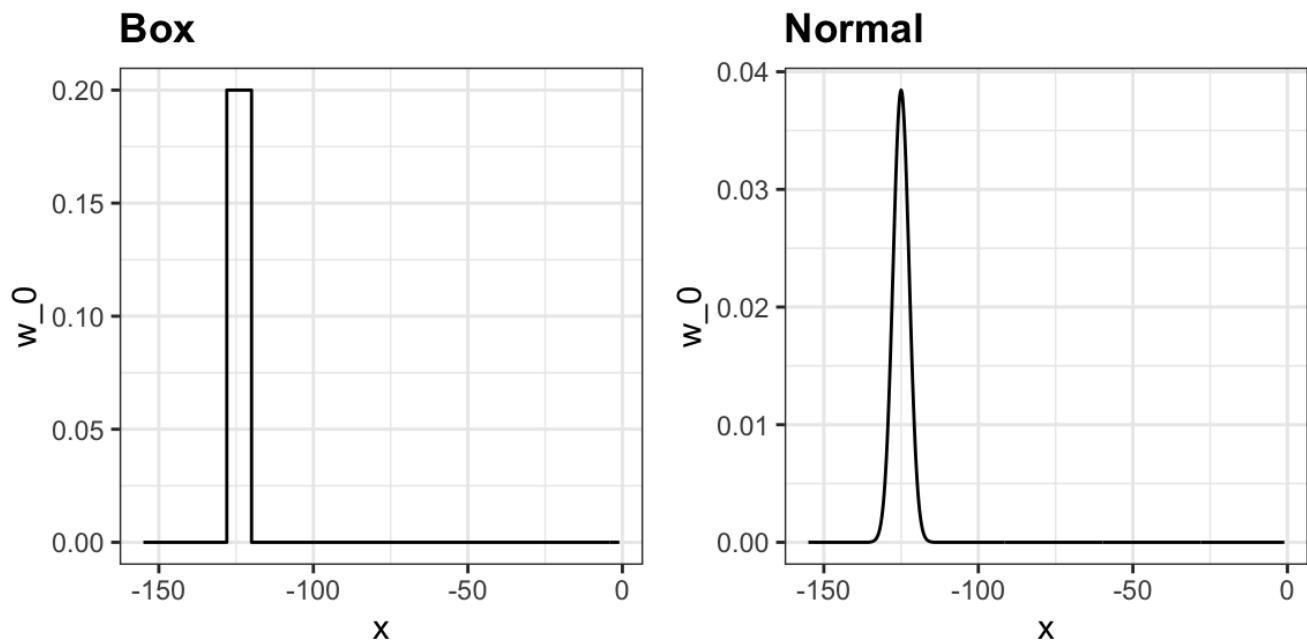


Figure 11:

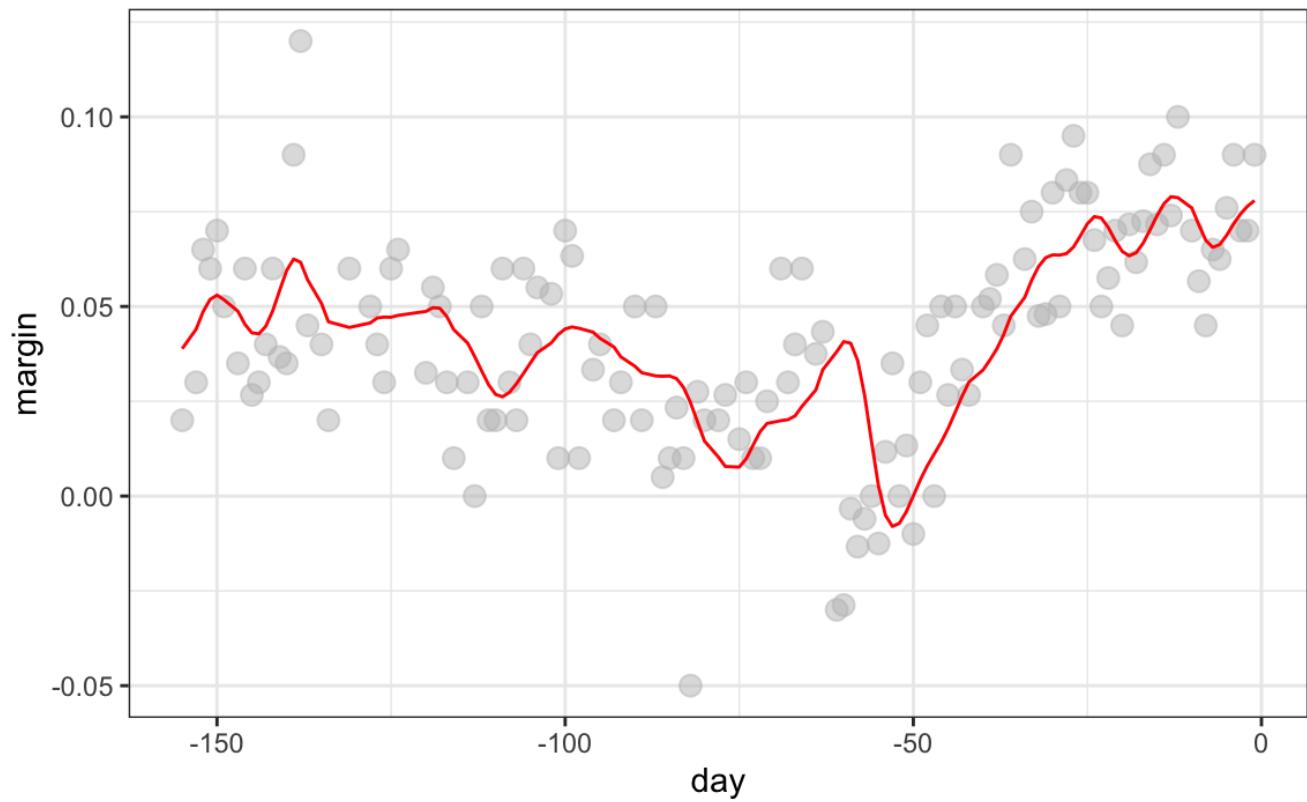


Figure 12:

that if you look closely enough at any smooth function $\backslash(f(x)\backslash)$, it will look like a line. To see why this makes sense, consider the curved edges gardeners make using straight-edged spades:



(“Downing Street garden path edge”¹ by Flickr user Number 10². CC-BY 2.0 license³.)

Instead of assuming the function is approximately constant in a window, we assume the function is locally linear. We can consider larger window sizes with the linear assumption than with a constant. Instead of the one-week window, we consider a larger one in which the trend is approximately linear. We start with a three-week window and later consider and evaluate other options:

```
\[ E[Y_i | X_i = x_i] = \beta_0 + \beta_1 (x_i - x_0) \boxed{ \text{if } |x_i - x_0| \leq 21 } \]
```

For every point $\backslash(x_0\backslash)$, loess defines a window and fits a line within that window. Here is an example showing the fits for $\backslash(x_0=-125\backslash)$ and $\backslash(x_0 = -55\backslash)$:

```
#> `geom_smooth()` using formula = 'y ~ x'
```

The fitted value at $\backslash(x_0\backslash)$ becomes our estimate $\backslash(\hat{f}(x_0)\backslash)$. Below we show the procedure happening as we move from the -155 up to 0:

The final result is a smoother fit than the bin smoother since we use larger sample sizes to estimate our local parameters:

```
total_days <- diff(range(polls_2008$day))
span <- 21/total_days
fit <- loess(margin ~ day, degree = 1, span = span, data = polls_2008)
polls_2008 |> mutate(smooth = fit$fitted) |>
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = .5, color = "grey") +
  geom_line(aes(day, smooth), color = "red")
```

Different spans give us different estimates. We can see how different window sizes lead to different estimates:

Here are the final estimates:

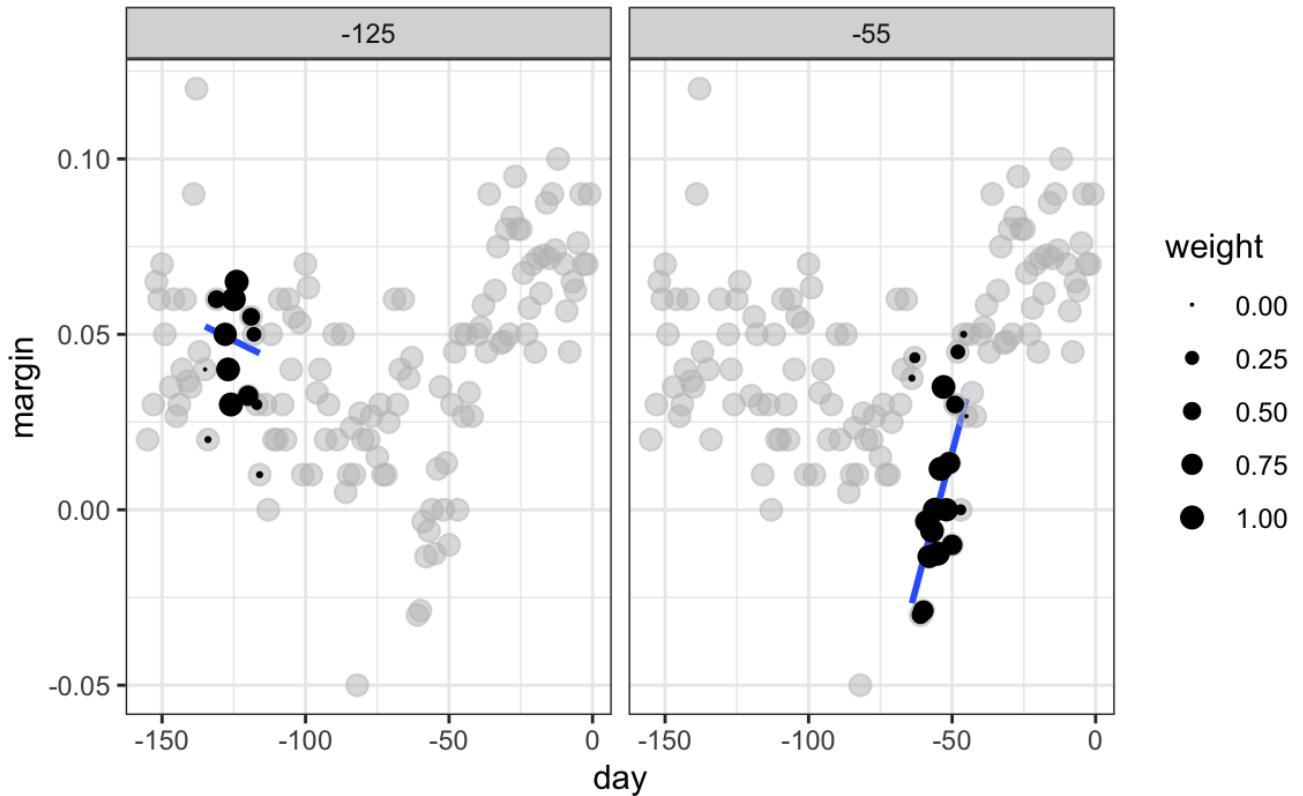


Figure 13:

There are three other differences between `loess` and the typical bin smoother.

1. Rather than keeping the bin size the same, `loess` keeps the number of points used in the local fit the same. This number is controlled via the `span` argument, which expects a proportion. For example, if N is the number of data points and `span=0.5`, then for a given \mathbf{x} , `loess` will use the $0.5 * N$ closest points to \mathbf{x} for the fit.
2. When fitting a line locally, `loess` uses a *weighted* approach. Basically, instead of using least squares, we minimize a weighted version:

$$[\sum_{i=1}^N w_0(x_i) \left[Y_i - (\beta_0 + \beta_1 (x_i - x_0)) \right]^2]$$

However, instead of the Gaussian kernel, `loess` uses a function called the Tukey tri-weight:

$$W(u) = \begin{cases} (1 - |u|^3)^3 & \text{if } |u| \leq 1 \\ 0 & \text{if } |u| > 1 \end{cases}$$

To define the weights, we denote $(2h)$ as the window size and define:

$$w_0(x_i) = W\left(\frac{x_i - x_0}{h}\right)$$

This kernel differs from the Gaussian kernel in that more points get values closer to the max:

3. `loess` has the option of fitting the local model *robustly*. An iterative algorithm is implemented in which, after fitting a model in one iteration, outliers are detected and down-weighted for the next iteration. To use this option, we use the argument `family="symmetric"`.

28.5.1 Fitting parabolas

Taylor's theorem also tells us that if you look at any mathematical function closely enough, it looks like a parabola. The theorem also states that you don't have to look as closely when approximating with parabolas as you do when approximating with lines. This means we can make our windows even larger and fit parabolas instead of lines.

$$E[Y_i | X_i = x_i] = \beta_0 + \beta_1 (x_i - x_0) + \beta_2 (x_i - x_0)^2 \quad \text{if } |x_i - x_0| \leq h$$

$x_0 = -155$

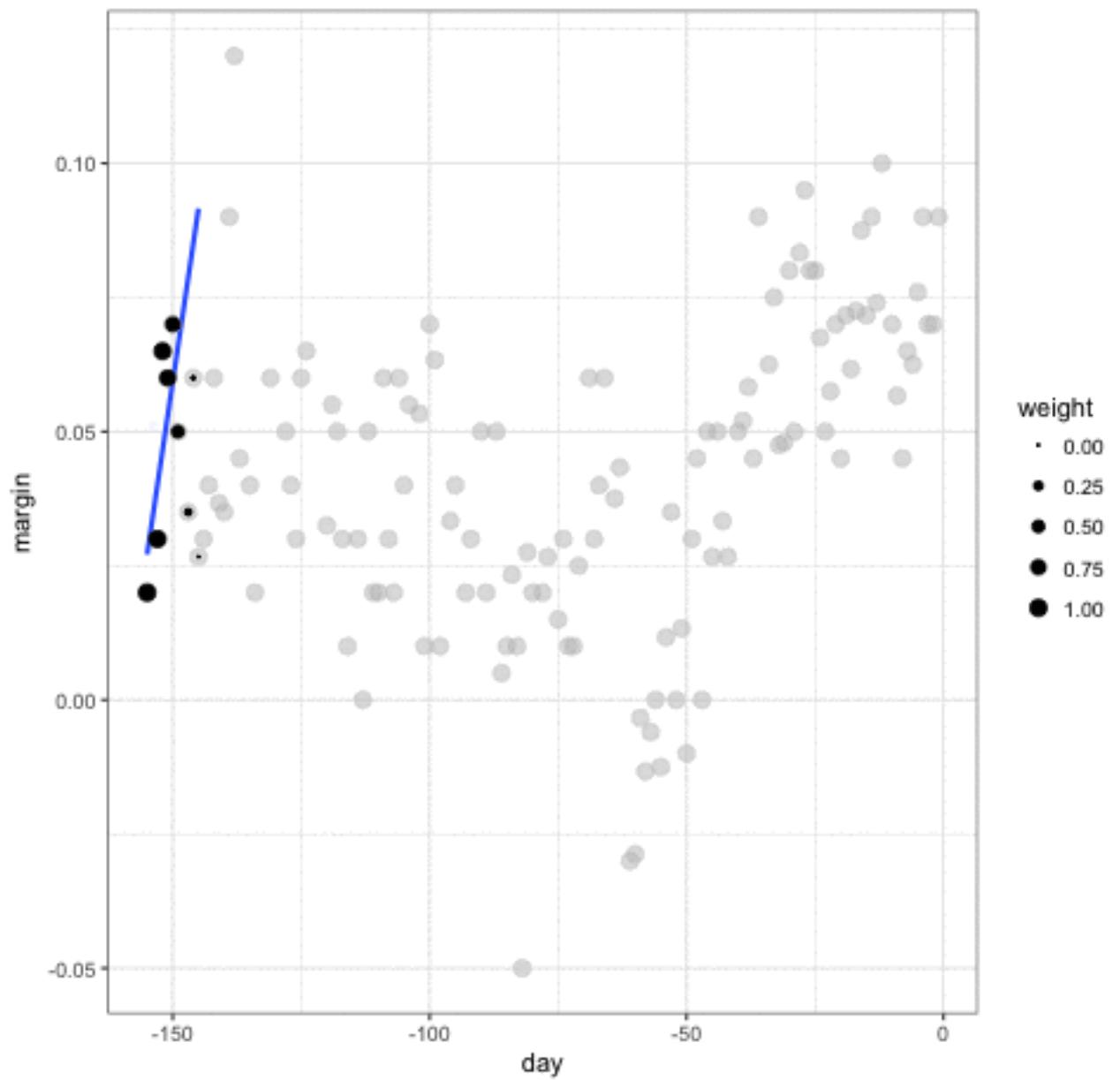


Figure 14:

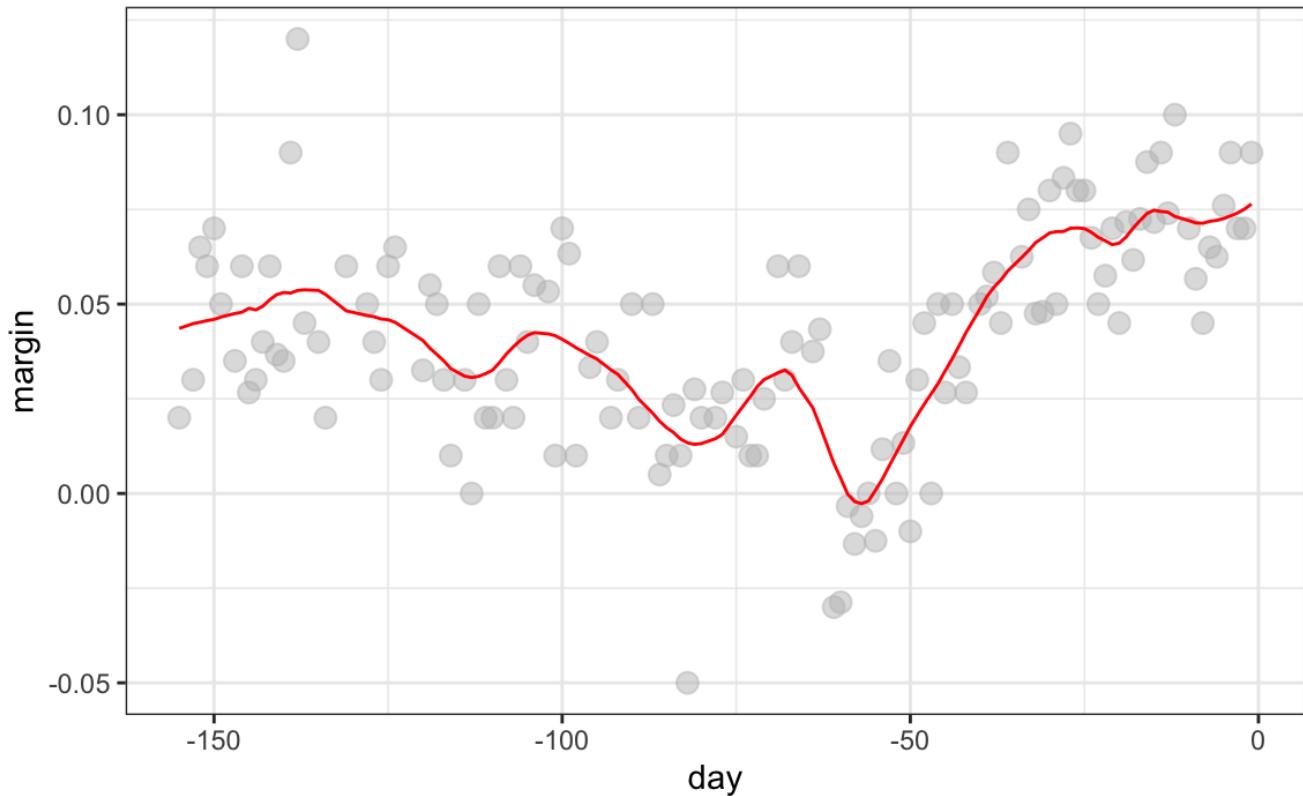


Figure 15:

You may have noticed that when we showed the code for using loess, we set `degree = 1`. This tells loess to fit polynomials of degree 1, a fancy name for lines. If you read the help page for loess, you will see that the argument `degree` defaults to 2. By default, loess fits parabolas not lines. Here is a comparison of the fitting lines (red dashed) and fitting parabolas (orange solid):

```
total_days <- diff(range(polls_2008$day))
span <- 28/total_days
fit_1 <- loess(margin ~ day, degree = 1, span = span, data = polls_2008)
fit_2 <- loess(margin ~ day, span = span, data = polls_2008)

polls_2008 |> mutate(smooth_1 = fit_1$fitted, smooth_2 = fit_2$fitted) |>
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = .5, color = "grey") +
  geom_line(aes(day, smooth_1), color = "red", lty = 2) +
  geom_line(aes(day, smooth_2), color = "orange", lty = 1)
```

The `degree = 2` gives us more wiggly results. In general, we actually prefer `degree = 1` as it is less prone to this kind of noise.

28.5.2 Beware of default smoothing parameters

`ggplot` uses loess in its `geom_smooth` function:

```
polls_2008 |> ggplot(aes(day, margin)) +
  geom_point() +
  geom_smooth(method = loess)
```

But be careful with default parameters as they are rarely optimal. However, you can conveniently change them:

```
polls_2008 |> ggplot(aes(day, margin)) +
```

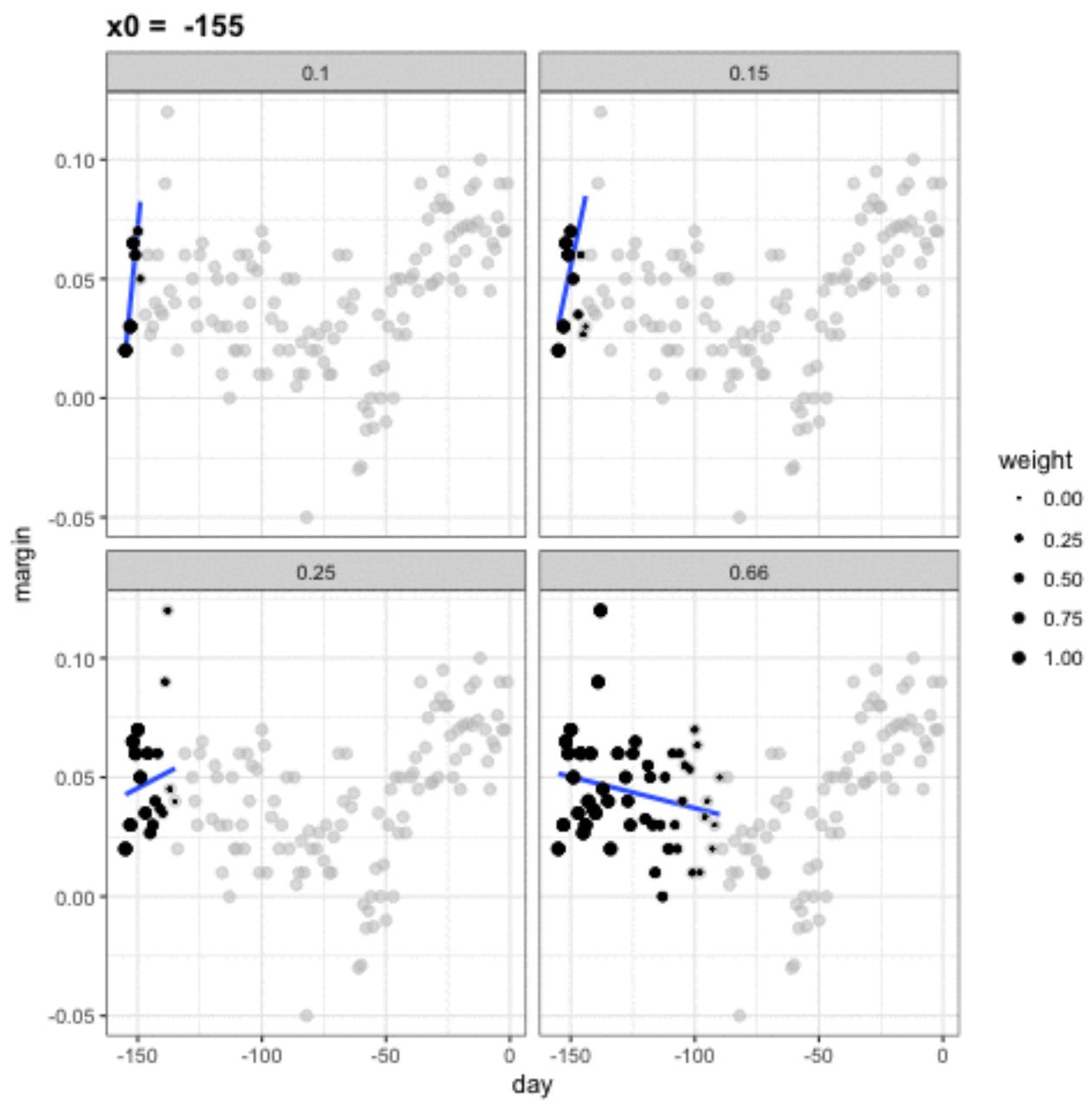


Figure 16:

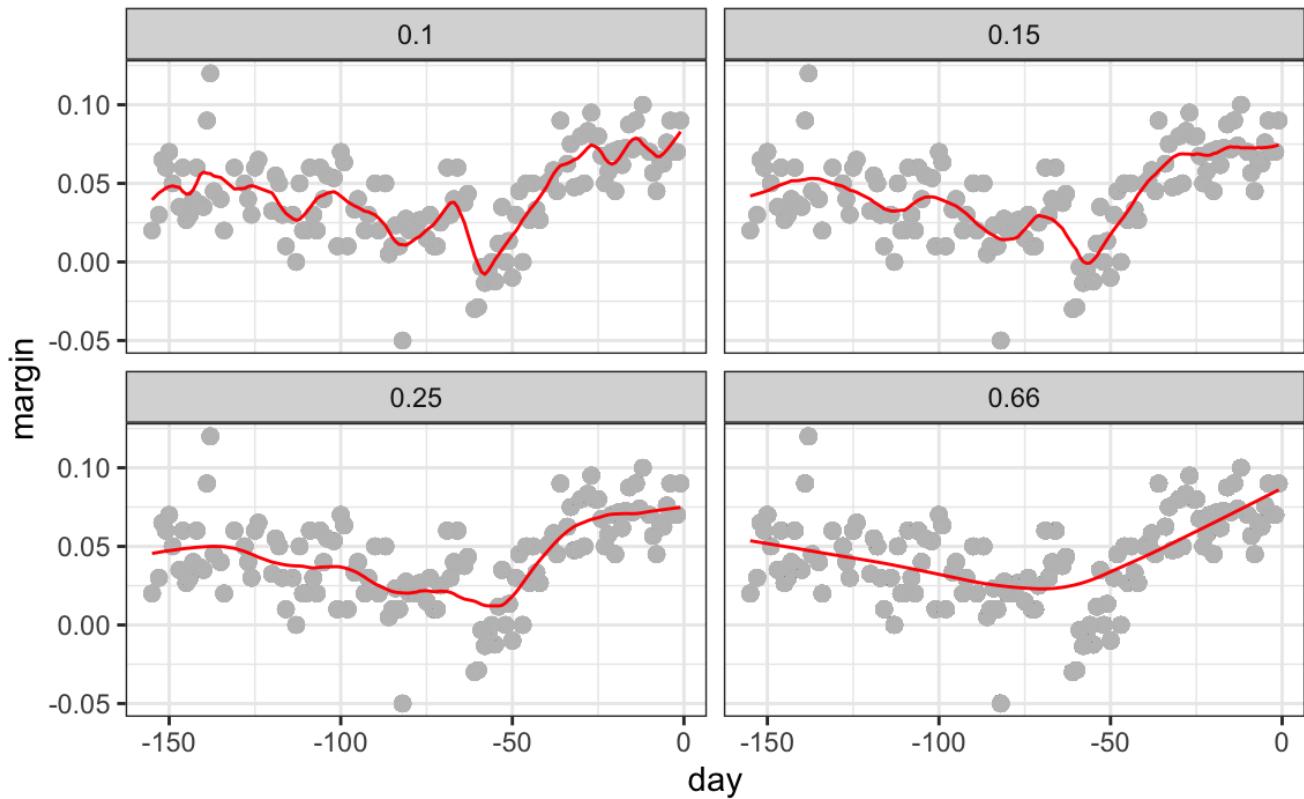


Figure 17:

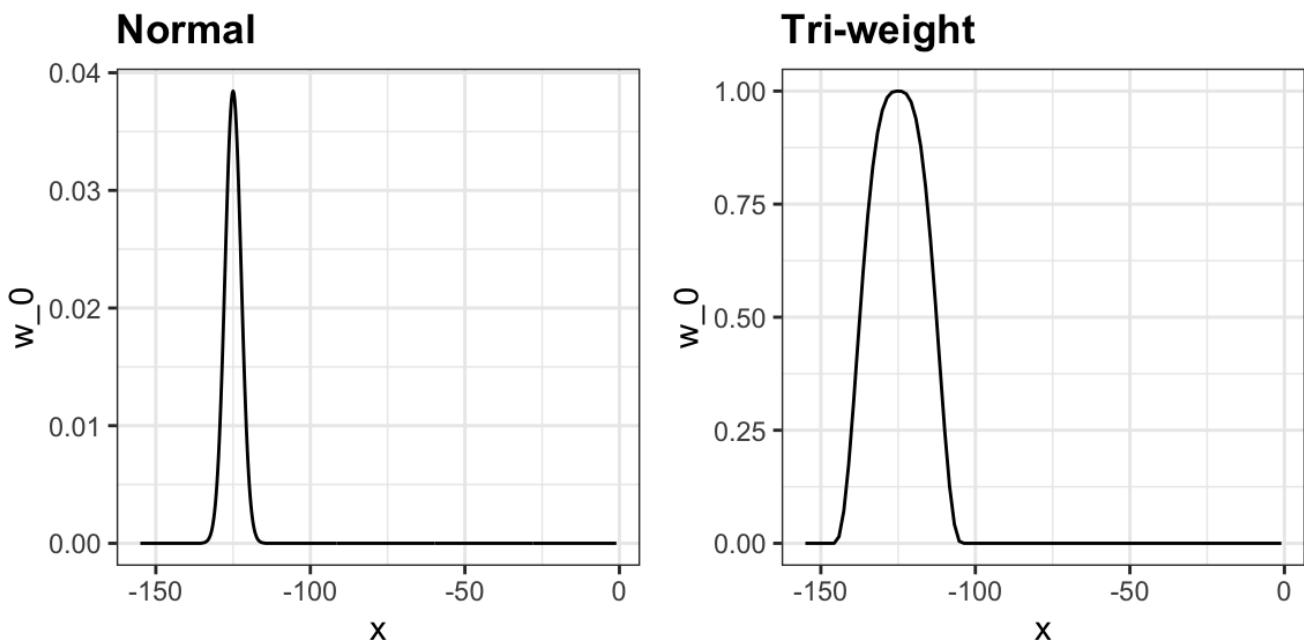


Figure 18:

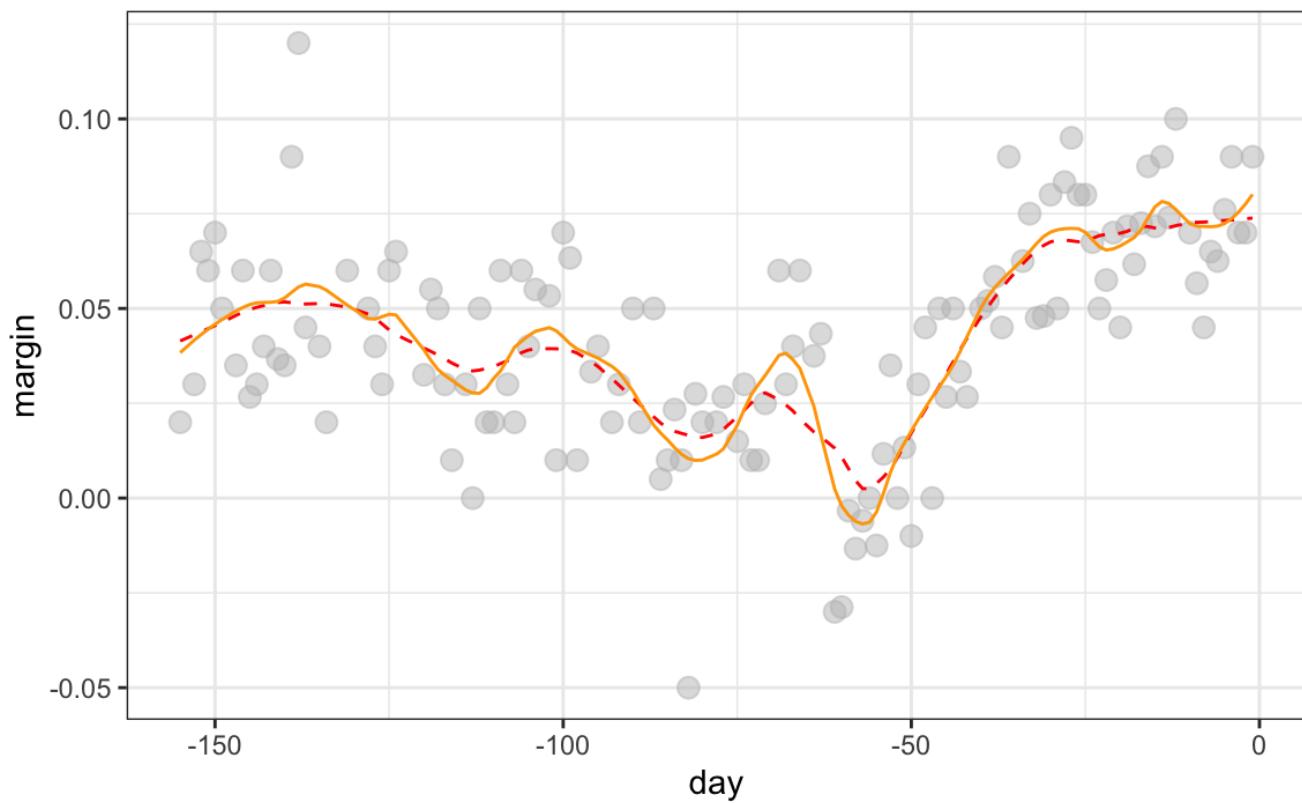


Figure 19:

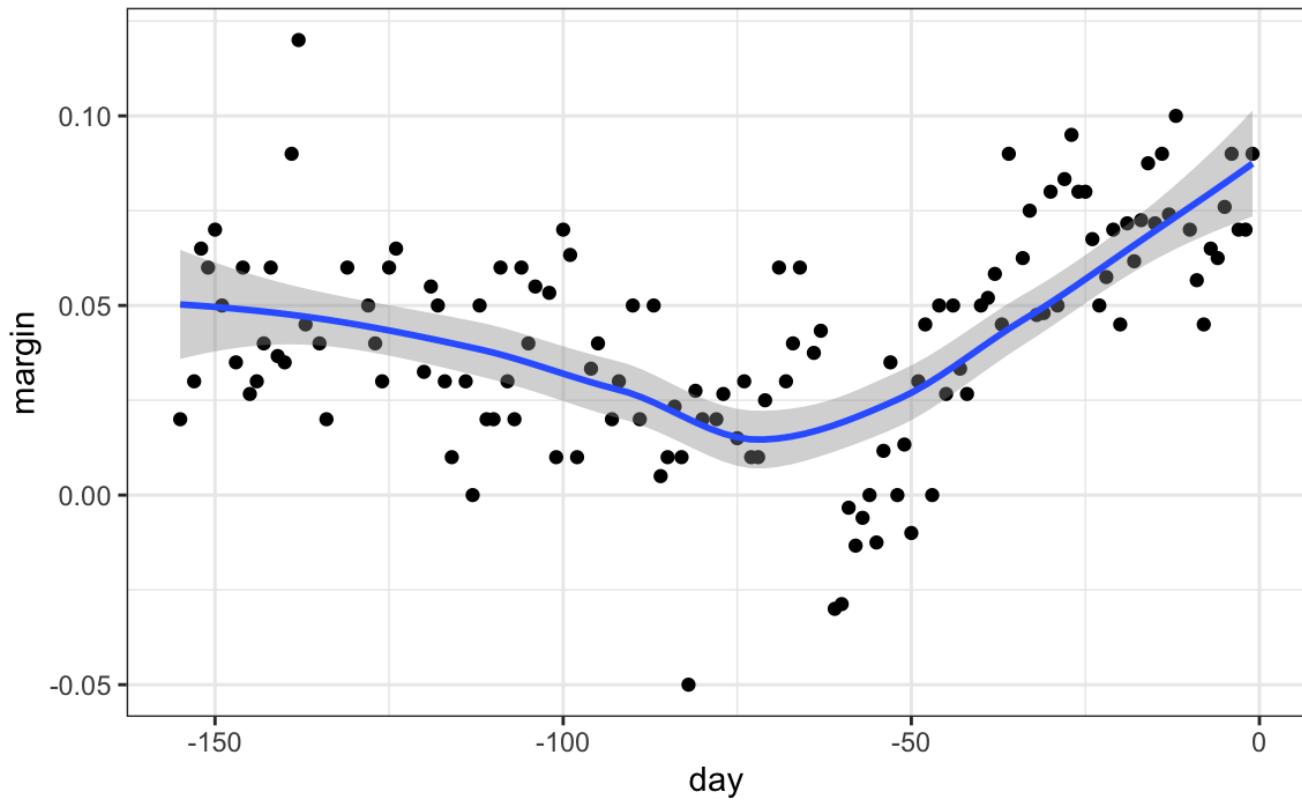


Figure 20:

```
geom_point() +
  geom_smooth(method = loess, method.args = list(span = 0.15, degree = 1))
```

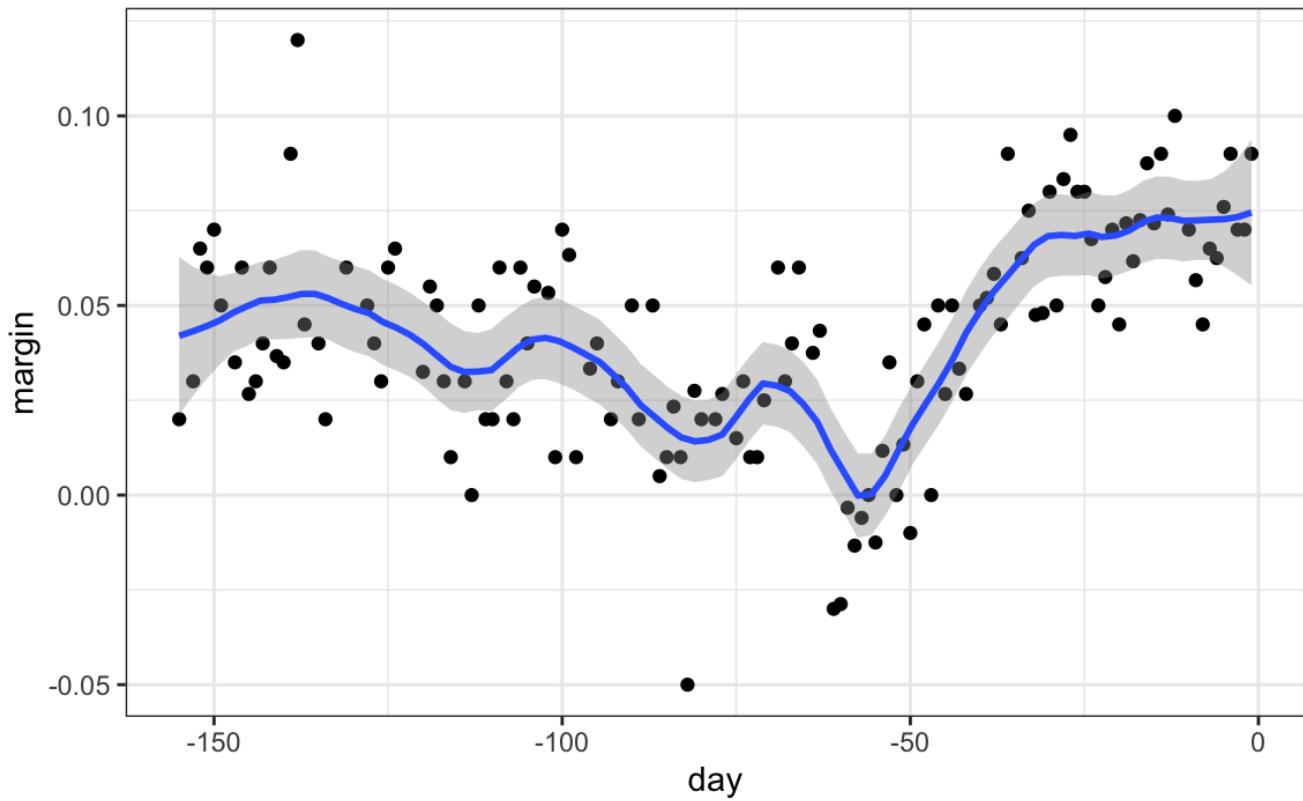


Figure 21:

28.6 Connecting smoothing to machine learning

To see how smoothing relates to machine learning with a concrete example, consider again our Section 28.1 example. If we define the outcome $(Y=1)$ for digits that are seven and $(Y=0)$ for digits that are 2, then we are interested in estimating the conditional probability:

$$[p(\mathbf{x}) = \Pr(Y=1 \mid X_1=x_1, X_2=x_2).]$$

with (x_1) and (x_2) the two predictors defined in Section 28.1. In this example, the 0s and 1s we observe are “noisy” because for some regions the probabilities $(p(\mathbf{x}))$ are not that close to 0 or 1. We therefore need to estimate $(p(\mathbf{x}))$. Smoothing is an alternative to accomplishing this. In Section 28.1, we saw that linear regression was not flexible enough to capture the non-linear nature of $(p(\mathbf{x}))$, thus smoothing approaches provide an improvement. In Section 29.1, we describe a popular machine learning algorithm, k-nearest neighbors, which is based on the concept of smoothing.

28.7 Exercises

1. The `dslabs` package provides the following dataset with mortality counts for Puerto Rico for 2015-2018.

```
library(dslabs)
head(pr_death_counts)
```

Remove data from before May 2018, then use the `loess` function to obtain a smooth estimate of the expected number of deaths as a function of date. Plot this resulting smooth function. Make the span about two months long.

2. Plot the smooth estimates against day of the year, all on the same plot but with different colors.

3. Suppose we want to predict 2s and 7s in our `mnist_27` dataset with just the second covariate. Can we do this? On first inspection it appears the data does not have much predictive power. In fact, if we fit a regular logistic regression, the coefficient for `x_2` is not significant!

```
library(broom)
library(dslabs)
mnist_27$train |>
  glm(y ~ x_2, family = "binomial", data = _) |>
  tidy()
```

Plotting a scatterplot here is not useful since `y` is binary:

```
qplot(x_2, y, data = mnist_27$train)
```

Fit a loess line to the data above and plot the results. Notice that there is predictive power, except the conditional probability is not linear.

1. <https://www.flickr.com/photos/49707497@N06/7361631644>
2. <https://www.flickr.com/photos/number10gov/>
3. <https://creativecommons.org/licenses/by/2.0/>

Introduction to Data Science - 29 Resampling methods

Rafael A. Irizarry

In this chapter, we introduce resampling, one of the most important ideas in machine learning. Here we focus on the conceptual and mathematical aspects. We will describe how to implement resampling methods in practice with the **caret** package later in Section 31.1.3. To motivate the concept, we will use the two predictor digits data presented in @ref-two-or-seven and introduce k-nearest neighbors (kNN), to demonstrate the ideas.

29.1 Motivation with k-nearest neighbors

We are interested in estimating the conditional probability function:

$$\Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2)$$

as defined in Section 28.6.

With k-nearest neighbors (kNN) we estimate $\Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2)$ in a similar way to bin smoothing. First, we define the distance between all observations based on the features. Then, for any point x_0 , we estimate $\Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2)$ by identifying the k nearest points to x_0 and afterwards taking an average of the y s associated with these points. We refer to the set of points used to compute the average as the *neighborhood*.

Due to the connection we described earlier between conditional expectations and conditional probabilities, this gives us $\hat{p}(Y = 1 \mid X_1 = x_1, X_2 = x_2)$, just like the bin smoother gave us an estimate of a trend. As with bin smoothers, we can control the flexibility of our estimate through the k parameter: larger k s result in smoother estimates, while smaller k s result in more flexible and wiggly estimates.

To implement the algorithm, we can use the **knn3** function from the **caret** package. Looking at the help file for this package, we see that we can call it in one of two ways. We will use the first way in which we specify a *formula* and a data frame. The data frame contains all the data to be used. The formula has the form `outcome ~ predictor_1 + predictor_2 + predictor_3` and so on. Therefore, we type `y ~ x_1 + x_2`. If we are going to use variables in the data frame, we can use the `.` like this `y ~ .` We also need to pick k , which is set to $k = 5$ by default. The final call looks like this:

```
library(dslabs)
library(caret)
knn_fit <- knn3(y ~ ., data = mnist_27$train, k = 5)
```

In this case, since our dataset is balanced and we care just as much about sensitivity as we do about specificity, we will use accuracy to quantify performance.

The `predict` function for **knn3** produces a probability for each class. We can keep the probability of being a 7 as the estimate $\hat{p}(Y = 1 \mid X_1 = x_1, X_2 = x_2)$ using `type = "prob"`. Here we obtain the actual prediction using `type = "class"`:

```
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(y_hat_knn, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.815
```

We see that kNN, with the default parameter, already beats regression. To see why this is the case, we plot $\hat{p}(Y = 1 \mid X_1 = x_1, X_2 = x_2)$ and compare it to the true conditional probability $\Pr(Y = 1 \mid X_1 = x_1, X_2 = x_2)$:

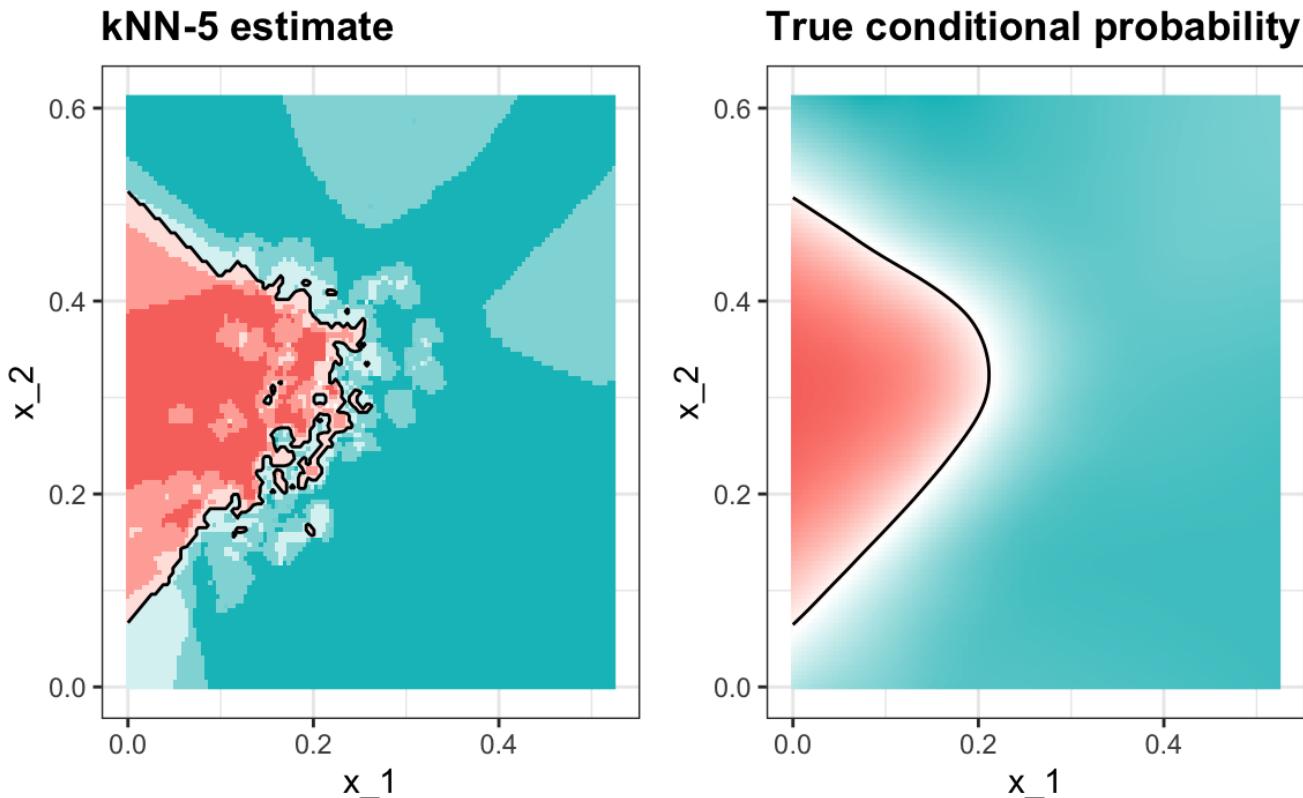


Figure 1:

We see that kNN better adapts to the non-linear shape of $p(\mathbf{x})$. However, our estimate has some islands of blue in the red area, which intuitively does not make much sense. We notice that we have higher accuracy in the train set compared to the test set:

```
y_hat_knn <- predict(knn_fit, mnist_27$train, type = "class")
confusionMatrix(y_hat_knn, mnist_27$train$y)$overall[["Accuracy"]]
#> Accuracy
#> 0.859
```

```
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(y_hat_knn, mnist_27$test$y)$overall[["Accuracy"]]
#> Accuracy
#> 0.815
```

This is due to what we call *over-training*.

29.2 Over-training

With kNN, over-training is at its worst when we set $k = 1$. With $k = 1$, the estimate for each \mathbf{x} in the training set is obtained with just the y corresponding to that point. In this case, if the x_1 and x_2 are unique, we will obtain perfect accuracy in the training set because each point is used to predict itself (if the predictors are not unique and have different outcomes for at least one set of predictors, then it is impossible to predict perfectly).

Here we fit a kNN model with $k = 1$ and confirm we get nearer to perfect accuracy in the training set:

```
knn_fit_1 <- knn3(y ~ ., data = mnist_27$train, k = 1)
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$train, type = "class")
confusionMatrix(y_hat_knn_1, mnist_27$train$y)$overall[["Accuracy"]]
```

```
#> [1] 0.995
```

But in the test set, accuracy is actually worse than what we obtained with regression:

```
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$test, type = "class")
confusionMatrix(y_hat_knn_1, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.81
```

We can see the over-fitting problem by plotting the decision rule boundaries produced by $\hat{p}(\mathbf{x})$:

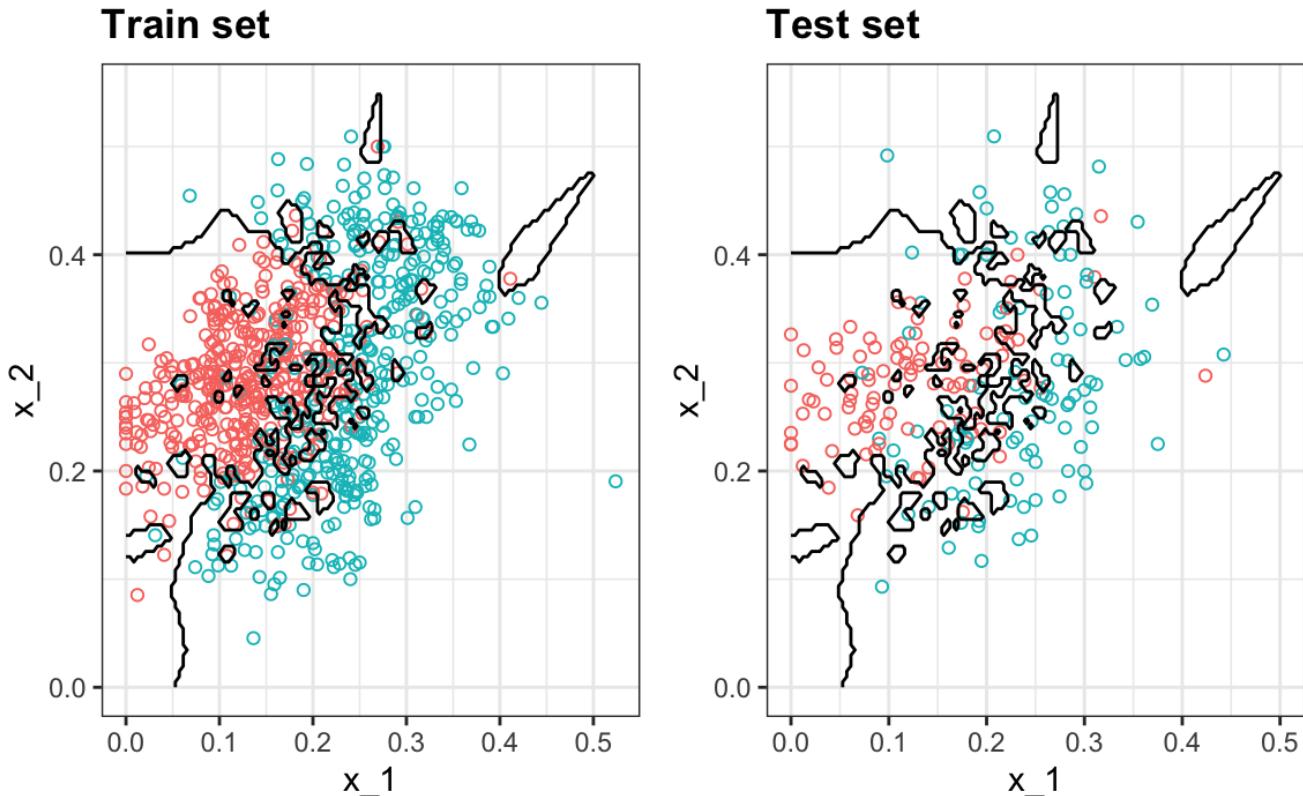


Figure 2:

The estimate $\hat{p}(\mathbf{x})$ follows the training data too closely (left). You can see that, in the training set, boundaries have been drawn to perfectly surround a single red point in a sea of blue. Because most points \mathbf{x} are unique, the prediction is either 1 or 0 and the prediction for that point is the associated label. However, once we introduce the test set (right), we see that many of these small islands now have the opposite color and we end up making several incorrect predictions.

29.3 Over-smoothing

Although not as badly as with $k=1$, we saw that with $k=5$ we also over-trained. Hence, we should consider a larger k . Let's try, as an example, a much larger number: $k=401$.

```
knn_fit_401 <- knn3(y ~ ., data = mnist_27$train, k = 401)
y_hat_knn_401 <- predict(knn_fit_401, mnist_27$test, type = "class")
confusionMatrix(y_hat_knn_401, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.76
```

The estimate turns out to be similar to the one obtained with regression:

In this case, k is so large that it does not permit enough flexibility. We call this *over-smoothing*.

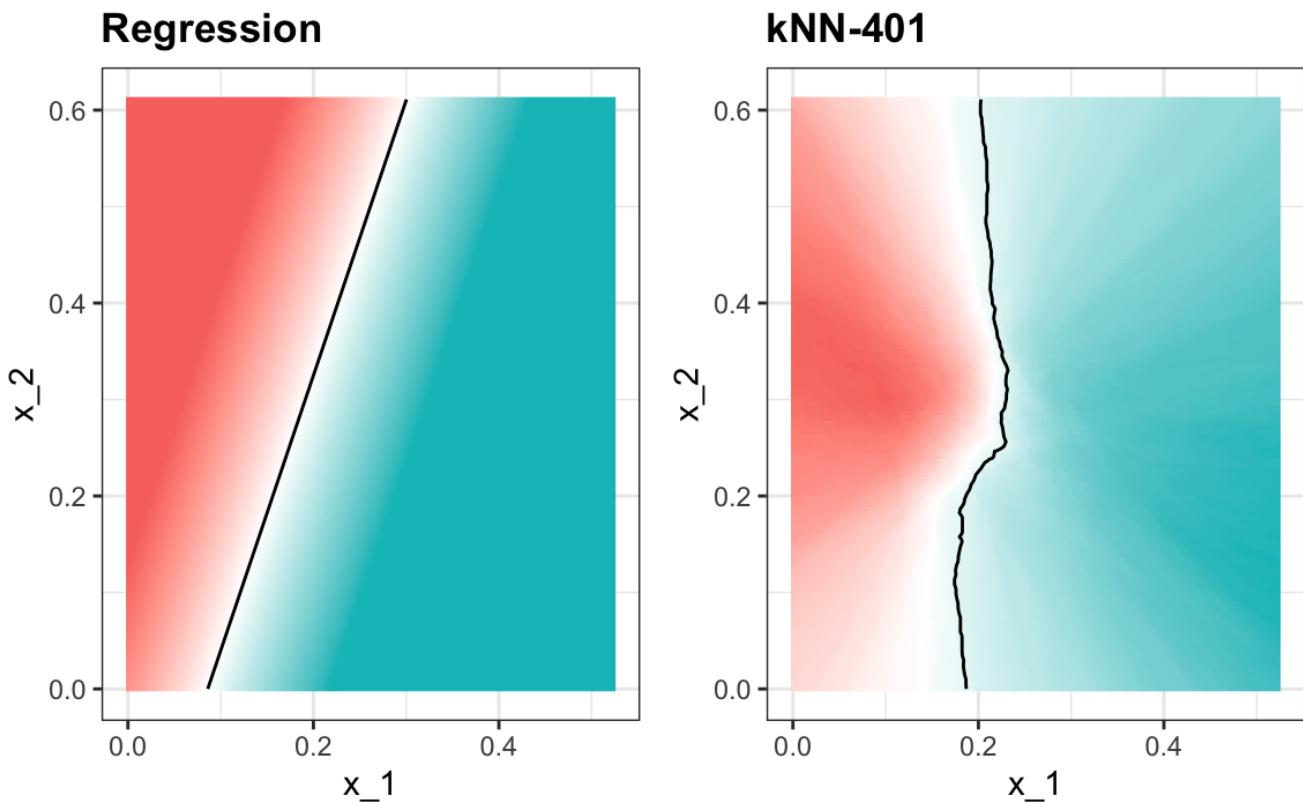


Figure 3:

29.4 Parameter tuning

It is very common for machine learning algorithms to require that we set a value, or values, before we even fit the model. An example is the $\backslash(k\backslash)$ in kNN. In Chapter 30 we learn of other examples. These values are referred to as *parameters* and an important step in machine learning in practice is picking or *tuning* those parameters.

So how do we tune these parameters? For example, how do we pick the $\backslash(k\backslash)$ in kNN? In principle, we want to pick the $\backslash(k\backslash)$ that maximizes accuracy or minimizes the expected MSE as defined in Section 26.8. The goal of resampling methods is to estimate these quantities for any given algorithm and set of tuning parameters such as $\backslash(k\backslash)$. To understand why we need a special method to do this, let's repeat what we did above, comparing the training set and test set accuracy, but for different values of $\backslash(k\backslash)$. We can plot the accuracy estimates for each value of $\backslash(k\backslash)$:

First, note that the estimate obtained on the training set is generally lower than the estimate obtained with the test set, with the difference larger for smaller values of $\backslash(k\backslash)$. This is due to over-training.

So do we simply pick the $\backslash(k\backslash)$ that maximizes accuracy and report this accuracy? There are two problems with this approach:

1. The accuracy versus $\backslash(k\backslash)$ plot is quite jagged. We do not expect this because small changes in $\backslash(k\backslash)$ should not affect the algorithm's performance so much. The jaggedness is explained by the fact that the accuracy is computed on a sample and therefore is a random variable.
2. Although for each $\backslash(k\backslash)$ we estimated MSE using the test set, we used the test set to pick the best $\backslash(k\backslash)$. As a result, we should not expect this minimum test set accuracy to extrapolate to the real world.

Resampling methods provide a solution to both these problems.

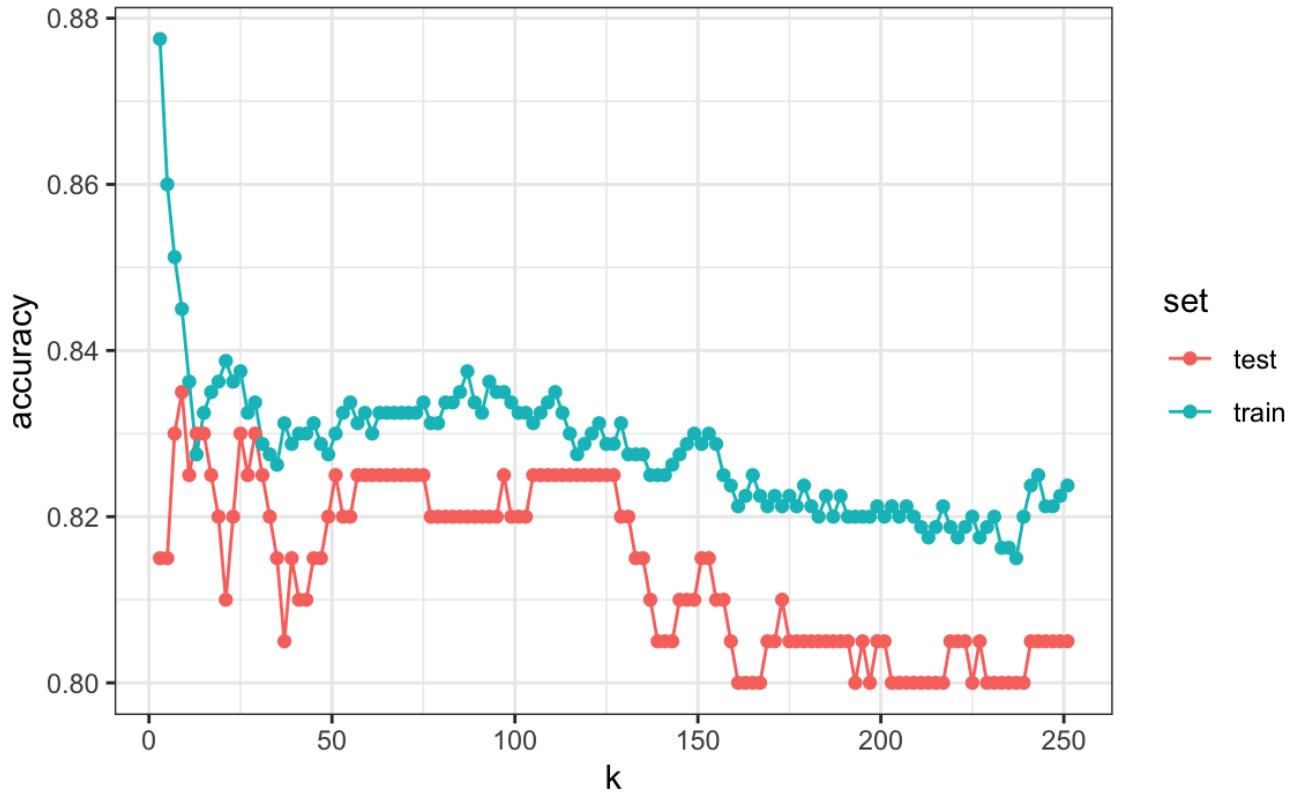


Figure 4:

29.5 Mathematical description of resampling methods

In the previous section, we introduced kNN as an example to motivate the topic of this chapter. In this particular case, there is just one parameter, $\lambda(k)$, that affects the performance of the algorithm. However, in general, machine algorithms may have multiple parameters so we use the notation $\lambda(\lambda)$ to represent any set of parameters needed to define a machine learning algorithm. We also introduce notation to distinguish the predictions you get with each set of parameters with $\hat{y}(\lambda)$ and the MSE for this choice with $MSE(\lambda)$. Our goal is to find the λ that minimizes $MSE(\lambda)$. Resampling method help us estimate $MSE(\lambda)$.

A intuitive first attempt is the apparent error defined in Section 26.8 and used in the previous section:

$$\frac{1}{N} \sum_{i=1}^N (\hat{y}_i(\lambda) - y_i)^2$$

As noted in the previous section, this estimate is a random error, based on just one test set, with enough variability to affect the choice of the best λ substantially.

Now imagine a world in which we could obtain data repeatedly, say from new random samples. We could take a very large number B of new samples, split them into training and test sets, and define:

$$\frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N (\hat{y}_i(\lambda) - y_i)^2$$

with y_i the i th observation in sample b and $\hat{y}_i(\lambda)$ the prediction obtained with the algorithm defined by parameter λ and trained independently of y_i . The law of large numbers tells us that as B becomes larger, this quantity gets closer and closer to $MSE(\lambda)$. This is of course a theoretical consideration as we rarely get access to more than one dataset for algorithm development, but the concept inspires the idea behind resampling methods.

The general idea is to generate a series of different random samples from the data at hand. There are several approaches to doing this, but all randomly generate several smaller datasets that are not used for training, and instead are used to estimate MSE. Next, we describe *cross validation*, one of the most widely used resampling

resampling methods.

29.6 Cross validation

Overall, we are provided a dataset (blue) and we need to build an algorithm, using this dataset, that will eventually be used in completely independent datasets (yellow) that we might not even see.

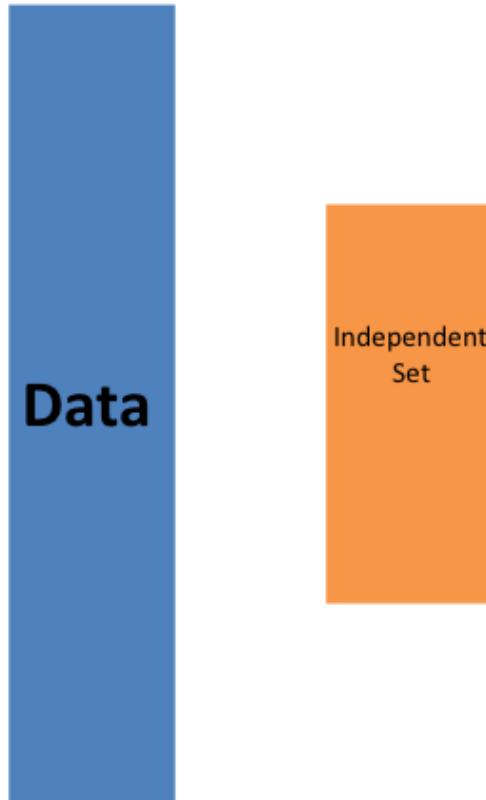


Figure 5:

So to imitate this situation, we start by carving out a piece of our dataset and pretend it is an independent dataset: we divide the dataset into a *training set* (blue) and a *test set* (red). We will train the entirety of our algorithm, including the choice of parameter λ , exclusively on the training set and use the test set only for evaluation purposes.

We usually try to select a small piece of the dataset so that we have as much data as possible to train. However, we also want the test set to be large so that we obtain a stable estimate of the MSE without fitting an impractical number of models. Typical choices are to use 10%-20% of the data for testing.

Let's reiterate that it is indispensable that we not use the test set at all: not for filtering out rows, not for selecting features, not for anything!

But then how do we optimize λ ? In cross validation, we achieve this by splitting the training set into two: the training and validation sets.

We will do this many times assuring that the estimates of MSE obtained in each dataset are independent from each other. There are several proposed methods for doing this. Here we describe one of these approaches, K-fold cross validation, in detail to provide the general idea used in all approaches.



Figure 6:

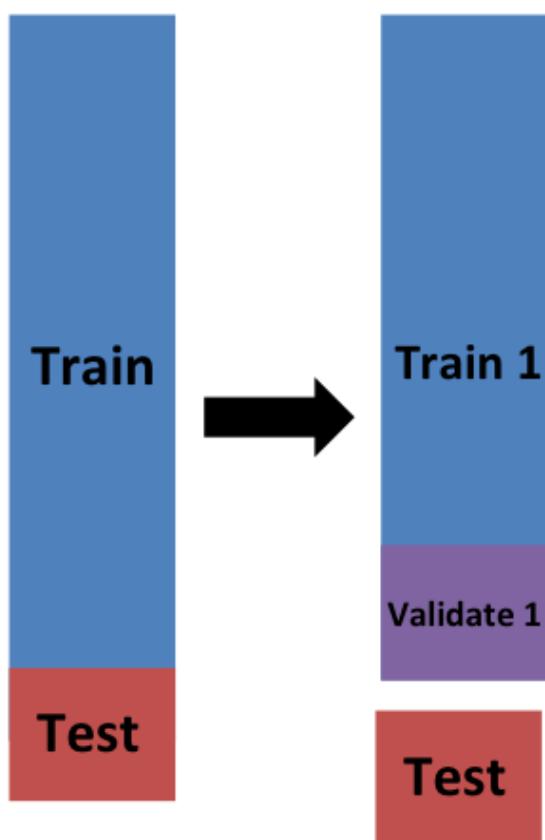


Figure 7:

29.6.1 K-fold cross validation

As a reminder, we are going to imitate the concept used when introducing this version of the MSE:

$$\left[\hat{\text{MSE}}(\lambda) \approx \frac{1}{B} \sum_{b=1}^B \frac{1}{N} \sum_{i=1}^N \left(\hat{y}_i - y_i \right)^2 \right]$$

We want to generate a dataset that can be thought of as independent random sample, and do this (B) times. The K in K-fold cross validation, represents the number of time (B) . In the illustrations, we are showing an example that uses $(B = 5)$.

We will eventually end up with (B) samples, but let's start by describing how to construct the first: we simply pick $(M = N/B)$ observations at random (we round if (M) is not a round number) and think of these as a random sample (y_1^b, \dots, y_M^b) , with $(b = 1)$. We call this the validation set.

Now we can fit the model in the training set, then compute the apparent error on the independent set:

$$\left[\hat{\text{MSE}}_b(\lambda) = \frac{1}{M} \sum_{i=1}^M \left(\hat{y}_i - y_i \right)^2 \right]$$

As a reminder, this is just one sample and will therefore return a noisy estimate of the true error. In K-fold cross validation, we randomly split the observations into (B) non-overlapping sets:

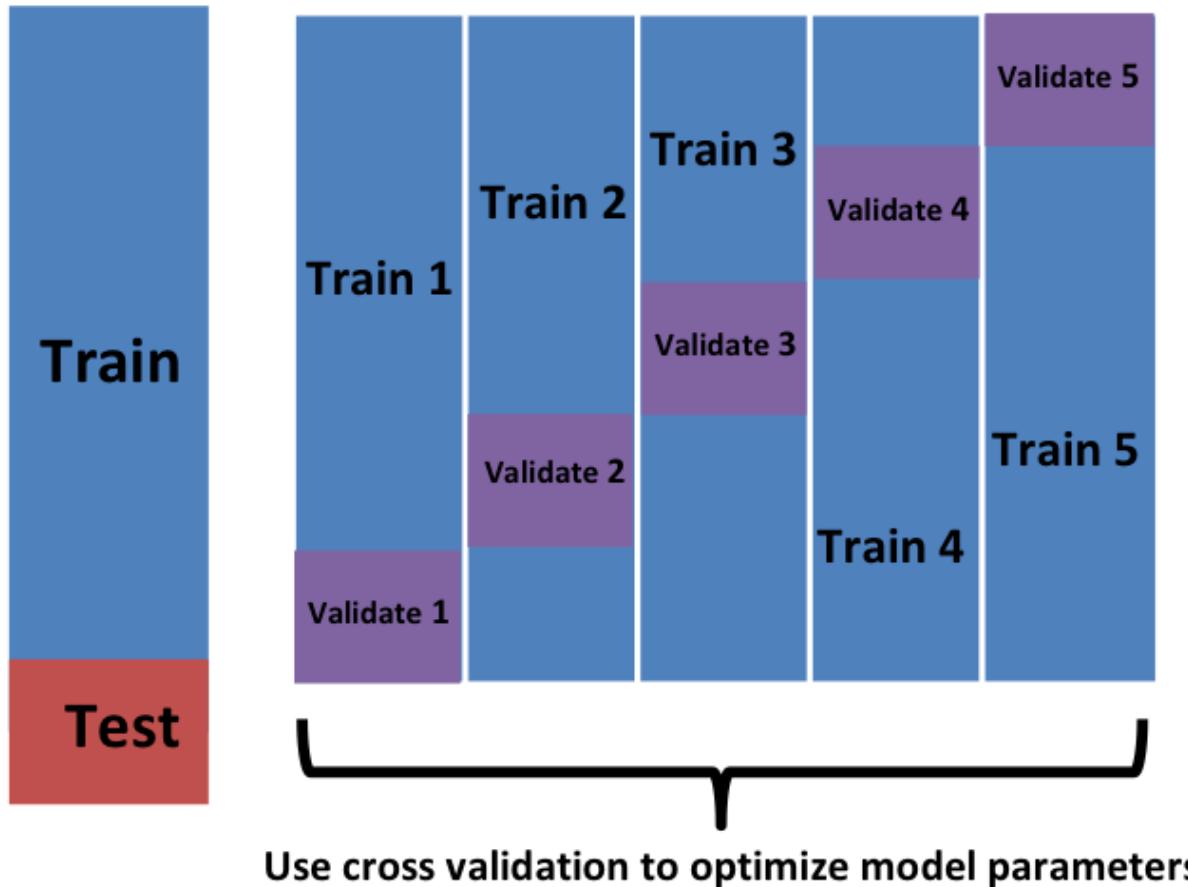


Figure 8:

Now we repeat the calculation above for each of these sets $(b = 1, \dots, B)$ and obtain $(\hat{\text{MSE}}_1(\lambda), \dots, \hat{\text{MSE}}_B(\lambda))$. Then, for our final estimate, we compute the average:

$\hat{\text{MSE}}(\lambda) = \frac{1}{B} \sum_{b=1}^B \hat{\text{MSE}}_b(\lambda)$
 and obtain an estimate of our loss. A final step would be to select the (λ) that minimizes the MSE.

29.6.2 How many folds?

Now how do we pick the cross validation fold? Large values of (B) are preferable because the training data better imitates the original dataset. However, larger values of (B) will have much slower computation time: for example, 100-fold cross validation will be 10 times slower than 10-fold cross validation. For this reason, the choices of $(B = 5)$ and $(B = 10)$ are popular.

One way we can improve the variance of our final estimate is to take more samples. To do this, we would no longer require the training set to be partitioned into non-overlapping sets. Instead, we would just pick (B) sets of some size at random.

29.6.3 Estimate MSE of our optimized algorithm

We have described how to use cross validation to optimize parameters. However, we now have to take into account the fact that the optimization occurred on the training data and we therefore need an estimate of our final algorithm based on data that was not used to optimize the choice. Here is where we use the test set we separated early on:

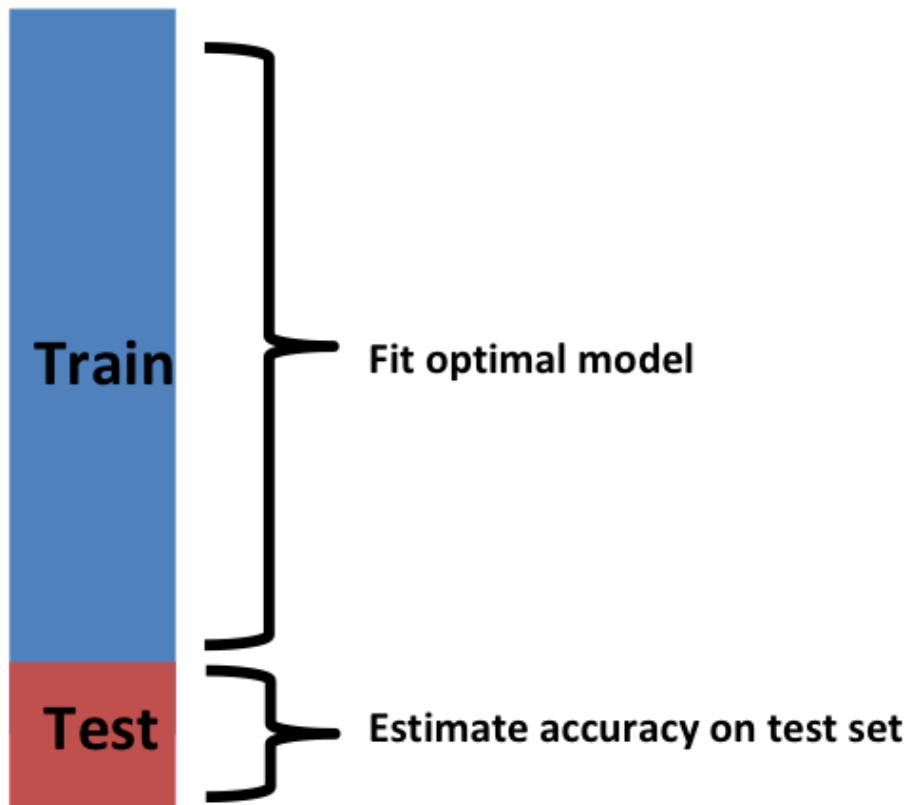


Figure 9:

We can actually do cross validation again:

and obtain a final estimate of our expected loss. However, note that last cross validation iteration means that our entire compute time gets multiplied by (K) . You will soon learn that fitting each algorithm takes time because we

Estimate accuracy of entire procedure with CV

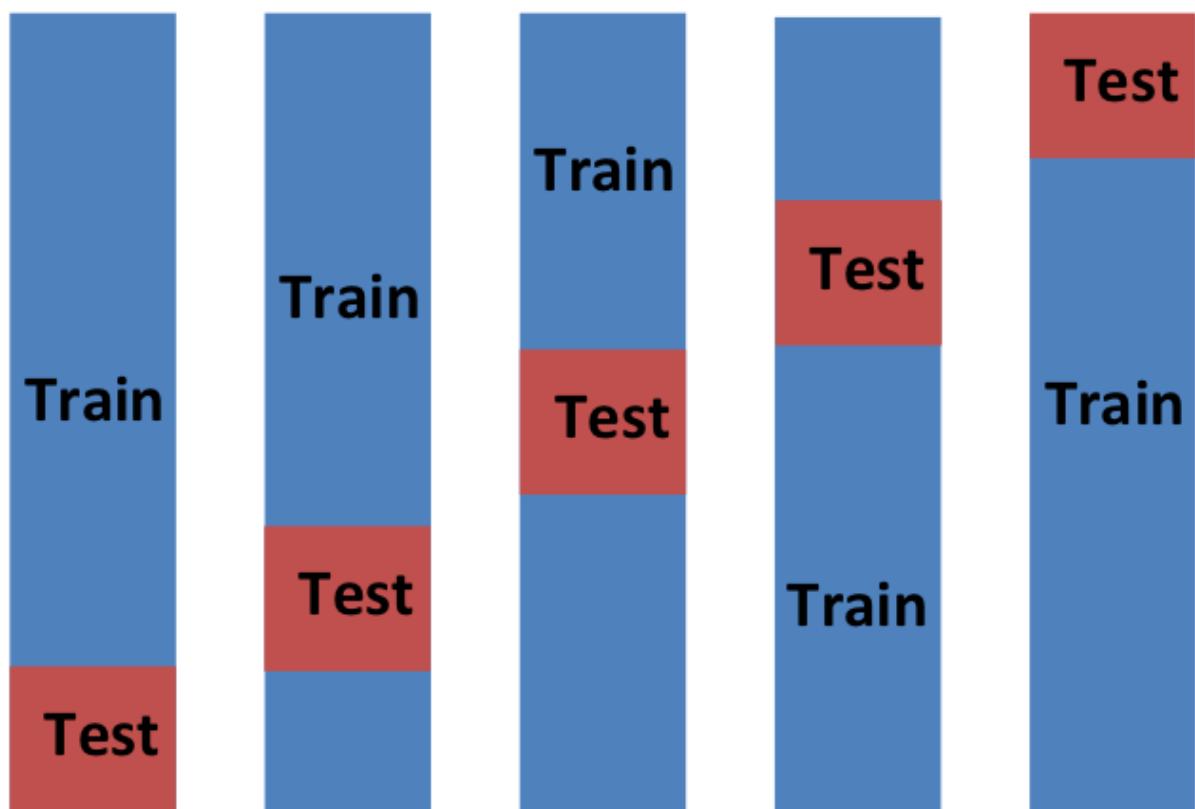


Figure 10:

are performing many complex computations. As a result, we are always looking for ways to reduce this time. For the final evaluation, we often just use the one test set.

Once we are satisfied with this model and want to make it available to others, we could refit the model on the entire dataset, without changing the optimized parameters.

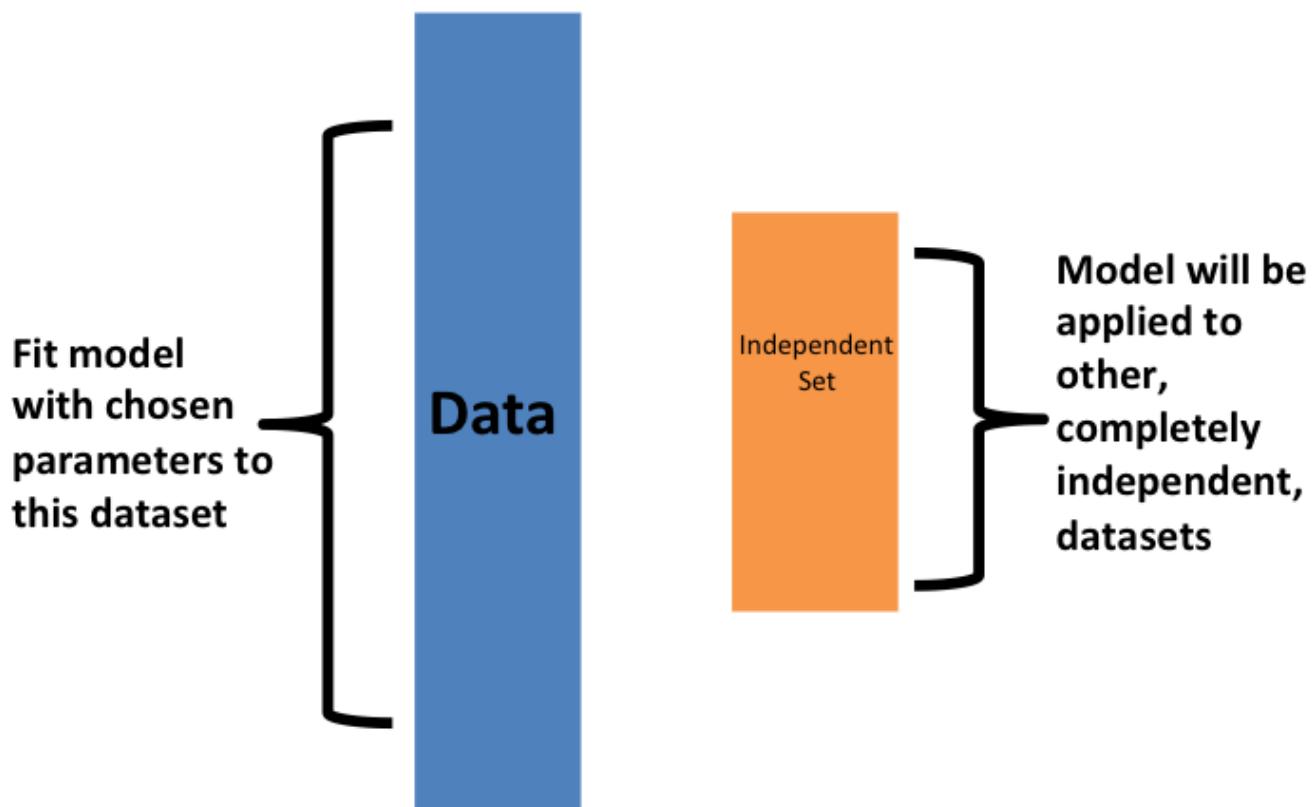


Figure 11:

29.7 Bootstrap resampling

Typically, cross-validation involves partitioning the original dataset into a training set to train the model and a testing set to evaluate it. With the bootstrap approach, based on the ideas described in Chapter 10, you can create multiple different training datasets via bootstrapping. This method is sometimes called bootstrap aggregating or bagging.

In bootstrap resampling, we create a large number of bootstrap samples from the original training dataset. Each bootstrap sample is created by randomly selecting observations with replacement, usually the same size as the original training dataset. For each bootstrap sample, we fit the model and compute the MSE estimate on the observations not selected in the random sampling, referred to as the *out-of-bag observations*. These out-of-bag observations serve a similar role to a validation set in standard cross-validation.

We then average the MSEs obtained in the out-of-bag observations from each bootstrap sample to estimate the model's performance.

This approach is actually the default approach in the **caret** package. We describe how to implement resampling methods with the **caret** package in the next chapter.

29.7.1 Comparison of MSE estimates

In Section 29.1, we computed an estimate of MSE based just on the provided test set (shown in red in the plot below). Here we show how the cross-validation techniques described above help reduce variability. The green curve below shows the results of applying K-fold cross validation with 10 folds, leaving out 10% of the data for validation. We can see that the variance is reduced substantially. The blue curve is the result of using 100 bootstrap samples to estimate MSE. The variability is reduced even further, but at the cost of a 10 fold increase in computation time.

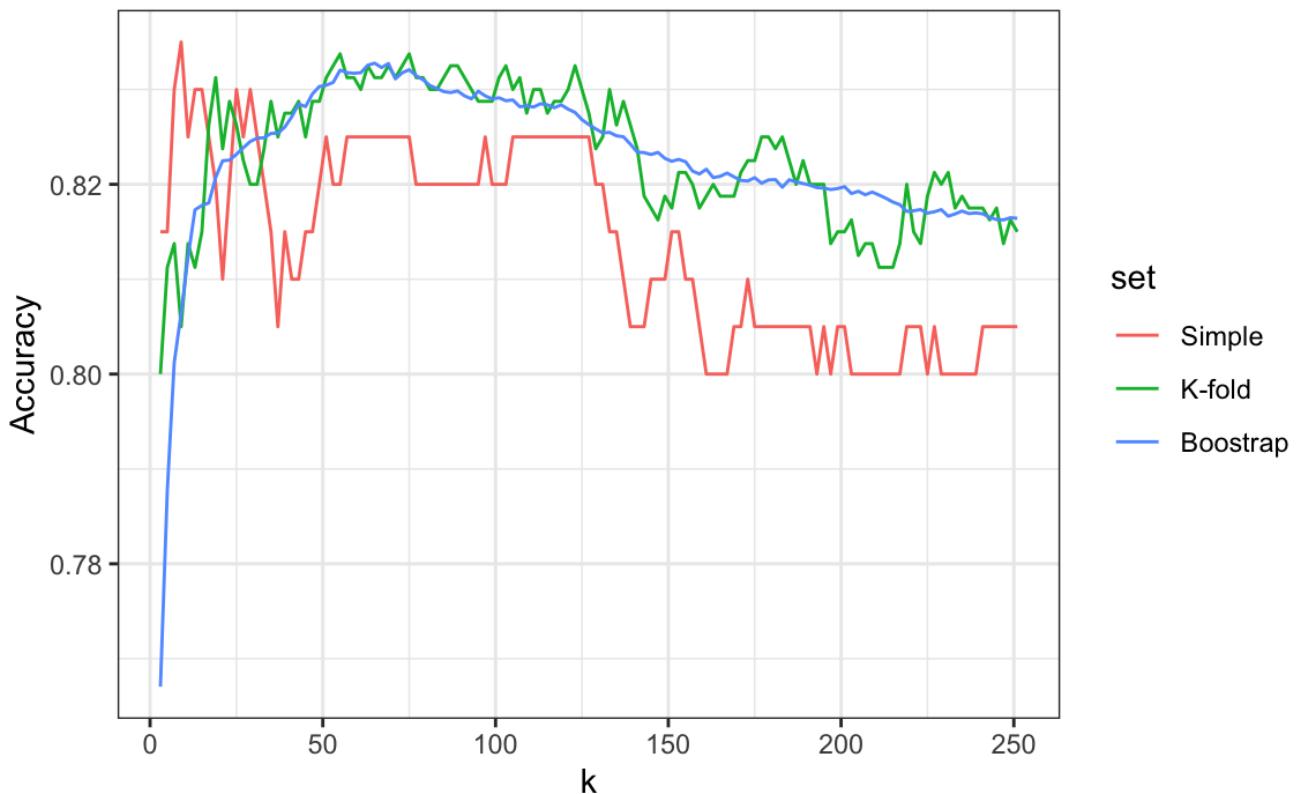


Figure 12:

29.8 Exercises

Generate a set of random predictors and outcomes like this:

```
set.seed(1996)
n <- 1000
p <- 10000
x <- matrix(rnorm(n * p), n, p)
colnames(x) <- paste("x", 1:ncol(x), sep = "_")
y <- rbinom(n, 1, 0.5) |> factor()

x_subset <- x[, sample(p, 100)]
```

1. Because x and y are completely independent, you should not be able to predict y using x with accuracy larger than 0.5. Confirm this by running cross validation using logistic regression to fit the model. Because we have so many predictors, we selected a random sample x_{subset} . Use the subset when training the model. Hint: use the caret `train` function. The `results` component of the output of `train` shows you the accuracy. Ignore the warnings.
2. Now instead of a random selection of predictors, we are going to search for those that are most predictive of the outcome. We can do this by comparing the values for the $\{y = 1\}$ group to those in the $\{y = 0\}$ group, for each predictor, using a t-test. You can perform this step as follows:

```
devtools::install_bioc("genefilter")
install.packages("genefilter")
library(genefilter)
tt <- colttests(x, y)
```

Create a vector of the p-values and call it `pvals`.

3. Create an index `ind` with the column numbers of the predictors that were “statistically significantly” associated with `y`. Use a p-value cutoff of 0.01 to define “statistically significant”. How many predictors survive this cutoff?
4. Re-run the cross validation but after redefining `x_subset` to be the subset of `x` defined by the columns showing “statistically significant” association with `y`. What is the accuracy now?
5. Re-run the cross validation again, but this time using kNN. Try out the following grid of tuning parameters: `k = seq(101, 301, 25)`. Make a plot of the resulting accuracy.
6. In exercises 3 and 4, we see that despite the fact that `x` and `y` are completely independent, we were able to predict `y` with accuracy higher than 70%. We must be doing something wrong then. What is it?
 - a. The function `train` estimates accuracy on the same data it uses to train the algorithm.
 - b. We are over-fitting the model by including 100 predictors.
 - c. We used the entire dataset to select the columns used in the model. This step needs to be included as part of the algorithm. The cross validation was done **after** this selection.
 - d. The high accuracy is just due to random variability.
7. Advanced. Re-do the cross validation but this time include the selection step in the cross validation. The accuracy should now be close to 50%.
8. Load the `tissue_gene_expression` dataset. Use the `train` function to predict tissue from gene expression. Use kNN. What `k` works best?
9. The `createResample` function can be used to create bootstrap samples. For example, we can create 10 bootstrap samples for the `mnist_27` dataset like this:

```
set.seed(1995)
indexes <- createResample(mnist_27$train$y, 10)
```

How many times do 3, 4, and 7 appear in the first re-sampled index?

10. We see that some numbers appear more than once and others appear no times. This must be so for each dataset to be independent. Repeat the exercise for all the re-sampled indexes.

Introduction to Data Science - 30 Examples of algorithms

Rafael A. Irizarry

There are hundreds of machine learning algorithms. Here we provide a few examples spanning rather different approaches. Throughout the chapter, we will be using the two predictor digits data introduced in Section 28.1 to demonstrate how the algorithms work. We focus on the concepts and ideas behind the algorithms using illustrative datasets from the **dslabs** package.

```
library(tidyverse)
library(caret)
library(dslabs)
```

Later, in Chapter 31, we show an efficient way to implement these ideas using the **caret** package.

30.1 Logistic regression

In Section 28.1, we used linear regression to predict classes by fitting the model:

$\Pr(Y=1 | \mathbf{X}_1=x_1, \mathbf{X}_2=x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

using least squares after assigning numeric values of 0 and 1 to the outcomes \mathbf{y} , and applied regression as if the data were continuous. A obvious problem with this approach is that $\hat{p}(\mathbf{x})$ can be negative and larger than 1:

```
fit_lm <- lm(y ~ x_1 + x_2, data = mutate(mnist_27$train, y = ifelse(y == 7, 1, 0)))
range(fit_lm$fitted)
#> [1] -0.22  1.92
```

To avoid this, we can apply the approach described in Section 18.5 that is more appropriate for binary data. We write the model like this:

$\log \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

We can then find the *maximum likelihood estimates* (MLE) of the model parameters and predict using the estimate $\hat{p}(\mathbf{x})$ to obtain an accuracy of 0.775. We see that logistic regression performs similarly to regression. This is not surprising given that the estimate of $\hat{p}(\mathbf{x})$ looks similar as well:

Just like regression, the decision rule is a line, a fact that can be corroborated mathematically. Defining $g(\mathbf{x}) = \log \frac{p(\mathbf{x})}{1-p(\mathbf{x})}$, we have:

$\hat{g}^{-1}(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2) = 0.5 \implies \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = g(0.5) = 0 \implies x_2 = -\hat{\beta}_0/\hat{\beta}_2 - \hat{\beta}_1/\hat{\beta}_2 x_1$

Thus, much like with regression, x_2 is a linear function of x_1 . This implies that our logistic regression approach has no chance of capturing the non-linear nature of the true $p(\mathbf{x})$. We now describe some techniques that estimate the conditional probability in a more flexible way.

You are ready to do exercises 1 - 11.

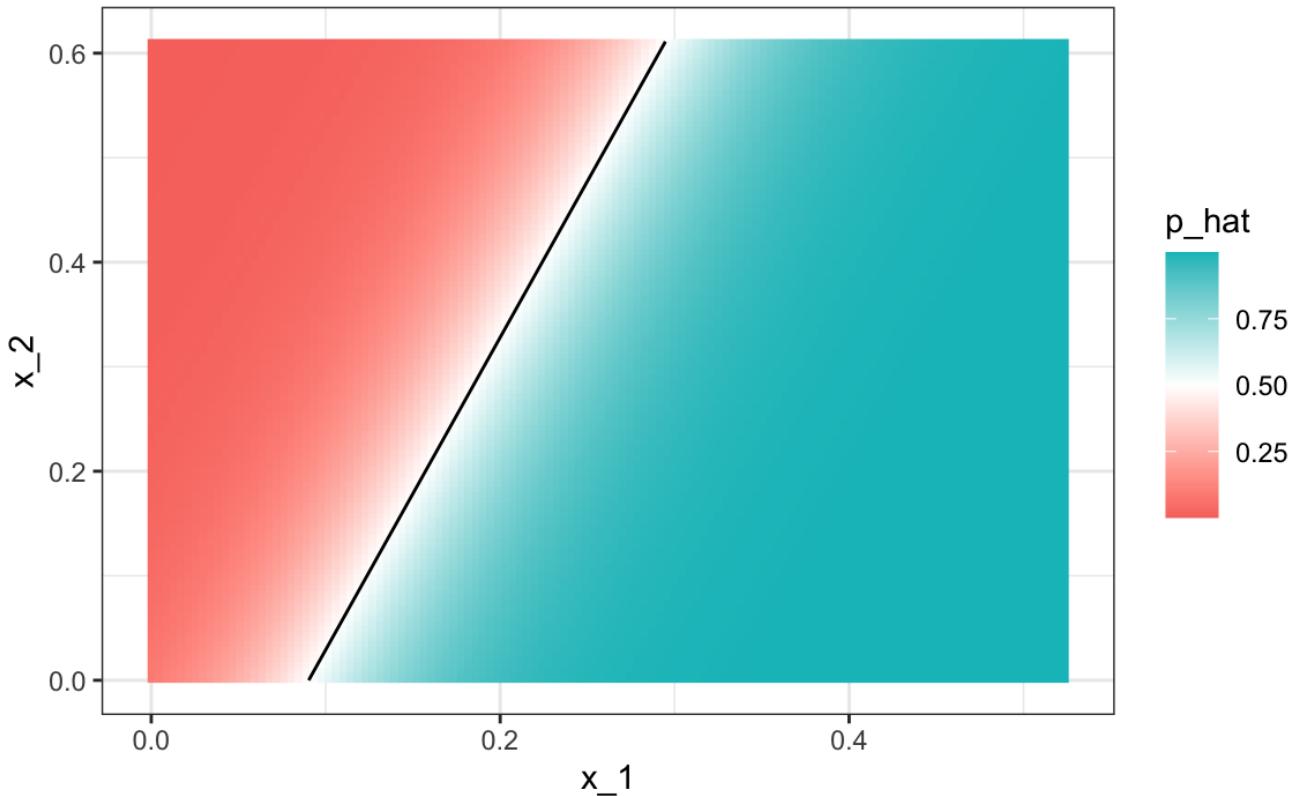


Figure 1:

30.2 k-nearest neighbors

We introduced the kNN algorithm in Section 29.1. In Section 29.7.1, we noted that $\backslash(k=31\backslash)$ provided the highest accuracy in the test set. Using $\backslash(k=31\backslash)$, we obtain an accuracy 0.825, an improvement over regression. A plot of the estimated conditional probability shows that the kNN estimate is flexible enough and does indeed capture the shape of the true conditional probability.

You are ready to do exercises 12 - 13.

30.3 Generative models

We have described how, when using squared loss, the conditional expectation provides the best approach to developing a decision rule. In a binary case, the smallest true error we can achieve is determined by Bayes' rule, which is a decision rule based on the true conditional probability:

$$\backslash[p(\mathbf{x}) = \Pr(Y = 1 \mid \mathbf{X} = \mathbf{x}) \backslash]$$

We have described several approaches to estimating $\backslash(p(\mathbf{x})\backslash)$. In all these approaches, we estimate the conditional probability directly and do not consider the distribution of the predictors. In machine learning, these are referred to as *discriminative* approaches.

However, Bayes' theorem tells us that knowing the distribution of the predictors $\backslash(\mathbf{X}\backslash)$ may be useful. Methods that model the joint distribution of $\backslash(Y\backslash)$ and $\backslash(\mathbf{X}\backslash)$ are referred to as *generative models* (we model how the entire data, $\backslash(\mathbf{X}\backslash)$ and $\backslash(Y\backslash)$, are generated). We start by describing the most general generative model, Naive Bayes, and then proceed to describe two specific cases, quadratic discriminant analysis (QDA) and linear discriminant analysis (LDA).

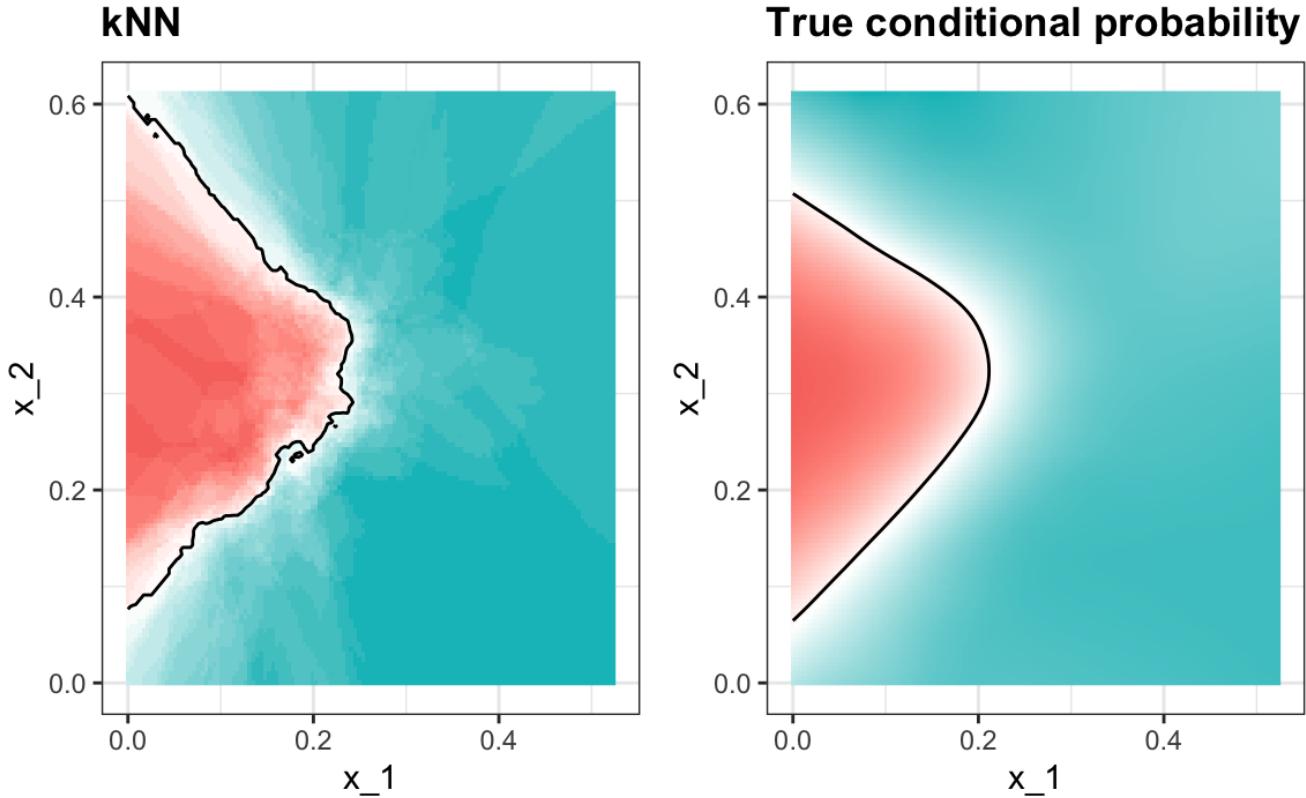


Figure 2:

30.3.1 Naive Bayes

Recall that Bayes rule tells us that we can rewrite $p(\mathbf{x})$ as follows:

$$p(\mathbf{x}) = \Pr(Y=1|\mathbf{X}=\mathbf{x}) = \frac{f_{\mathbf{X}|Y=1}(\mathbf{x})}{\Pr(Y=1) f_{\mathbf{X}|Y=0}(\mathbf{x}) \Pr(Y=0) + f_{\mathbf{X}|Y=1}(\mathbf{x}) \Pr(Y=1)}$$

with $f_{\mathbf{X}|Y=1}(\mathbf{x})$ and $f_{\mathbf{X}|Y=0}(\mathbf{x})$ representing the distribution functions of the predictor \mathbf{X} for the two classes $(Y=1)$ and $(Y=0)$. The formula implies that if we can estimate these conditional distributions of the predictors, we can develop a powerful decision rule. However, this is a big *if*.

As we go forward, we will encounter examples in which \mathbf{X} has many dimensions and we do not have much information about the distribution. In these cases, Naive Bayes will be practically impossible to implement. However, there are instances in which we have a small number of predictors (not much more than 2) and many categories in which generative models can be quite powerful. We describe two specific examples and use our previously described case studies to illustrate them.

Let's start with a very simple and uninteresting, yet illustrative, case: the example related to predicting sex from height.

```
set.seed(1995)
y <- heights$height
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- heights |> slice(-test_index)
test_set <- heights |> slice(test_index)
```

In this case, the Naive Bayes approach is particularly appropriate because we know that the normal distribution is a good approximation for the conditional distributions of height given sex for both classes $(Y=1)$ (female) and $(Y=0)$ (male). This implies that we can approximate the conditional distributions $f_{\mathbf{X}|Y=1}$ and $f_{\mathbf{X}|Y=0}$

$0\})$ by simply estimating averages and standard deviations from the data:

```
params <- train_set |> group_by(sex) |> summarize(avg = mean(height), sd = sd(height))
params
#> # A tibble: 2 × 3
#>   sex      avg     sd
#>   <fct> <dbl> <dbl>
#> 1 Female  64.8  4.14
#> 2 Male    69.2  3.57
```

The prevalence, which we will denote with $\pi = \Pr(Y = 1)$, can be estimated from the data with:

```
pi <- train_set |> summarize(pi = mean(sex == "Female")) |> pull(pi)
pi
#> [1] 0.212
```

Now we can use our estimates of average and standard deviation to get an actual rule:

```
x <- test_set$height
f0 <- dnorm(x, params$avg[2], params$sd[2])
f1 <- dnorm(x, params$avg[1], params$sd[1])
p_hat_bayes <- f1*pi / (f1*pi + f0*(1 - pi))
```

Our Naive Bayes estimate $\hat{p}(x)$ looks a lot like a logistic regression estimate:

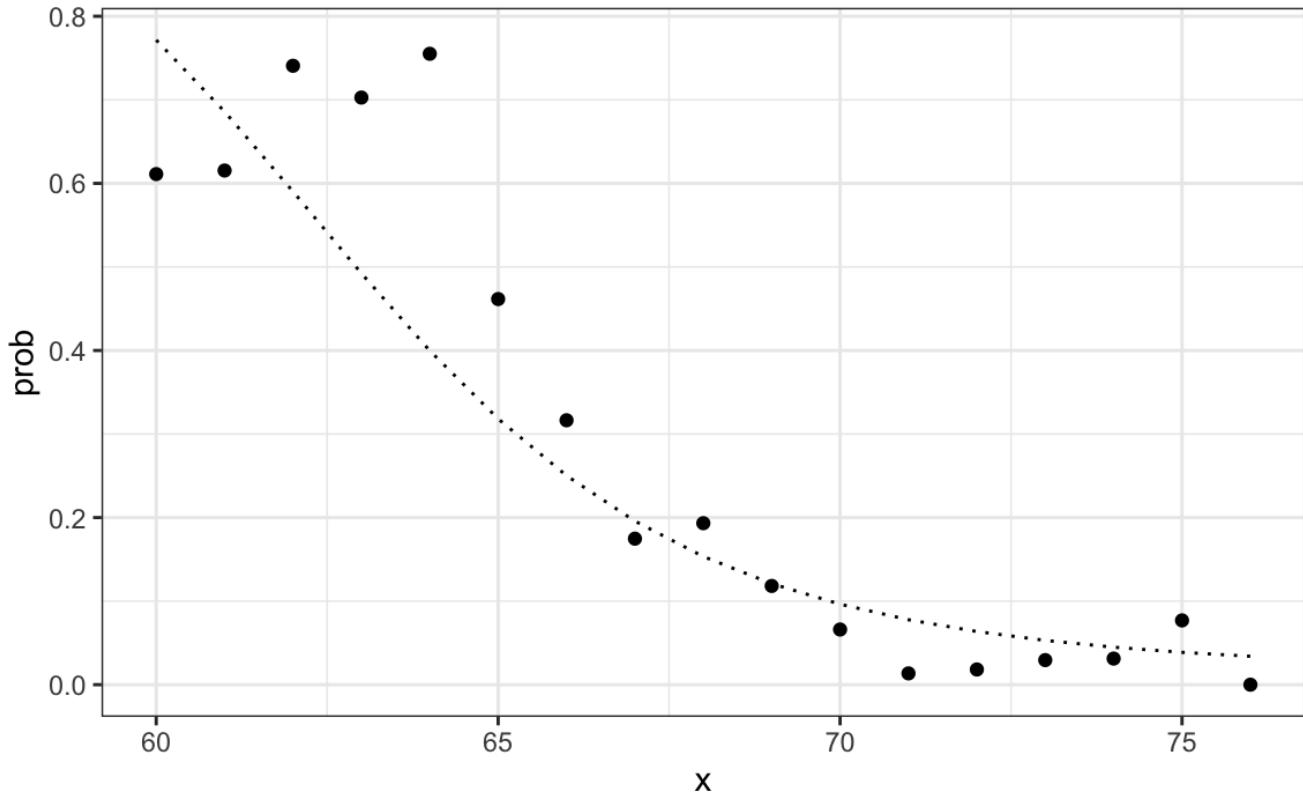


Figure 3:

In fact, we can show that the Naive Bayes approach is similar to the logistic regression prediction mathematically. However, we leave the demonstration to a more advanced text, such as the Elements of Statistical Learning¹. We can see that they are similar empirically by comparing the two resulting curves.

30.3.2 Controlling prevalence

One useful feature of the Naive Bayes approach is that it includes a parameter to account for differences in prevalence. Using our sample, we estimated $\hat{f}_1(X|Y=1)$, $\hat{f}_0(X|Y=0)$ and $\hat{\pi}$. If we use hats to denote the estimates, we can write $\hat{p}(x) = \frac{\hat{f}_1(X|Y=1)(\hat{\pi})}{\hat{f}_1(X|Y=1)(\hat{\pi}) + \hat{f}_0(X|Y=0)(1-\hat{\pi})}$ as:

$$\hat{p}(x) = \frac{\hat{f}_1(X|Y=1)(\hat{\pi})}{\hat{f}_1(X|Y=1)(\hat{\pi}) + \hat{f}_0(X|Y=0)(1-\hat{\pi})}$$

As we discussed earlier, our sample has a much lower prevalence, 0.21, than the general population. So if we use the rule $(\hat{p}(x) > 0.5)$ to predict females, our accuracy will be affected due to the low sensitivity:

```
y_hat_bayes <- ifelse(p_hat_bayes > 0.5, "Female", "Male")
sensitivity(data = factor(y_hat_bayes), reference = factor(test_set$sex))
#> [1] 0.213
```

Again, this is because the algorithm gives more weight to specificity to account for the low prevalence:

```
specificity(data = factor(y_hat_bayes), reference = factor(test_set$sex))
#> [1] 0.967
```

This is mainly due to the fact that $\hat{\pi}$ is substantially less than 0.5, so we tend to predict **Male** more often. It makes sense for a machine learning algorithm to do this in our sample because we do have a higher percentage of males. But if we were to extrapolate this to a general population, our overall accuracy would be affected by the low sensitivity.

The Naive Bayes approach gives us a direct way to correct this since we can simply force $\hat{\pi}$ to be whatever value we want it to be. So to balance specificity and sensitivity, instead of changing the cutoff in the decision rule, we could simply change $\hat{\pi}$ to 0.5 like this:

```
p_hat_bayes_unbiased <- f1 * 0.5 / (f1 * 0.5 + f0 * (1 - 0.5))
y_hat_bayes_unbiased <- ifelse(p_hat_bayes_unbiased > 0.5, "Female", "Male")
```

Note the difference in sensitivity with a better balance:

```
sensitivity(factor(y_hat_bayes_unbiased), factor(test_set$sex))
#> [1] 0.693
specificity(factor(y_hat_bayes_unbiased), factor(test_set$sex))
#> [1] 0.832
```

The new rule also gives us a very intuitive cutoff between 66-67, which is about the middle of the female and male average heights:

```
plot(x, p_hat_bayes_unbiased)
abline(h = 0.5, lty = 2) +
abline(v = 67, lty = 2)
#> integer(0)
```

30.3.3 Quadratic discriminant analysis

Quadratic Discriminant Analysis (QDA) is a version of Naive Bayes in which we assume that the distributions $(p_{\mathbf{X}|Y=1}(x))$ and $(p_{\mathbf{X}|Y=0}(x))$ are multivariate normal. The simple example we described in the previous section is actually QDA. Let's now look at a slightly more complicated case: the 2 or 7 example.

In this example, we have two predictors so we assume each one is bivariate normal. This implies that we need to estimate two averages, two standard deviations, and a correlation for each case $(Y=1)$ and $(Y=0)$. Once we have these, we can approximate the distributions $(f_{X_1, X_2|Y=1})$ and $(f_{X_1, X_2|Y=0})$. We can easily estimate parameters from the data:

```
params <- mnist_27$train |>
  group_by(y) |>
  summarize(avg_1 = mean(x_1), avg_2 = mean(x_2),
            sd_1 = sd(x_1), sd_2 = sd(x_2),
```

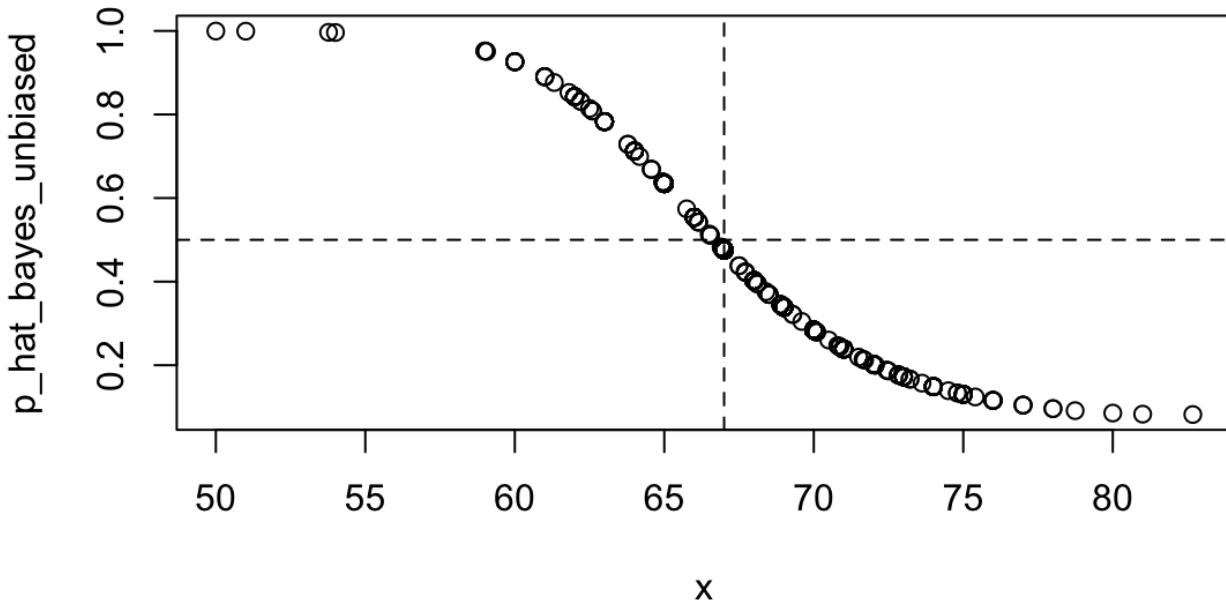


Figure 4:

```
r = cor(x_1, x_2))
params
#> # A tibble: 2 × 6
#>   y      avg_1 avg_2   sd_1   sd_2      r
#>   <fct> <dbl> <dbl> <dbl> <dbl>
#> 1 2      0.136 0.287 0.0670 0.0600 0.415
#> 2 7      0.238 0.290 0.0745 0.104   0.468
```

With these estimates in place, all we need are the prevalence (π) to compute:

$$\hat{p}(\mathbf{x}) = \frac{\hat{p}_1(\mathbf{x})}{\hat{p}_1(\mathbf{x}) + \hat{p}_2(\mathbf{x})}$$

Note that the densities (f) are bivariate normal distributions. Here we provide a visual way of showing the approach. We plot the data and use contour plots to give an idea of what the two estimated normal densities look like (we show the curve representing a region that includes 95% of the points):

We can fit QDA using the `qda` function the **MASS** package:

```
train_qda <- MASS::qda(y ~ ., data = mnist_27$train)
y_hat <- predict(train_qda, mnist_27$test)$class
```

We see that we obtain relatively good accuracy:

```
confusionMatrix(y_hat, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.815
```

The conditional probability looks relatively good, although it does not fit as well as the kernel smoothers:

One reason QDA does not work as well as the kernel methods is because the assumption of normality does not quite hold. Although for the 2s it seems reasonable, for the 7s it does seem to be off. Notice the slight curvature in the

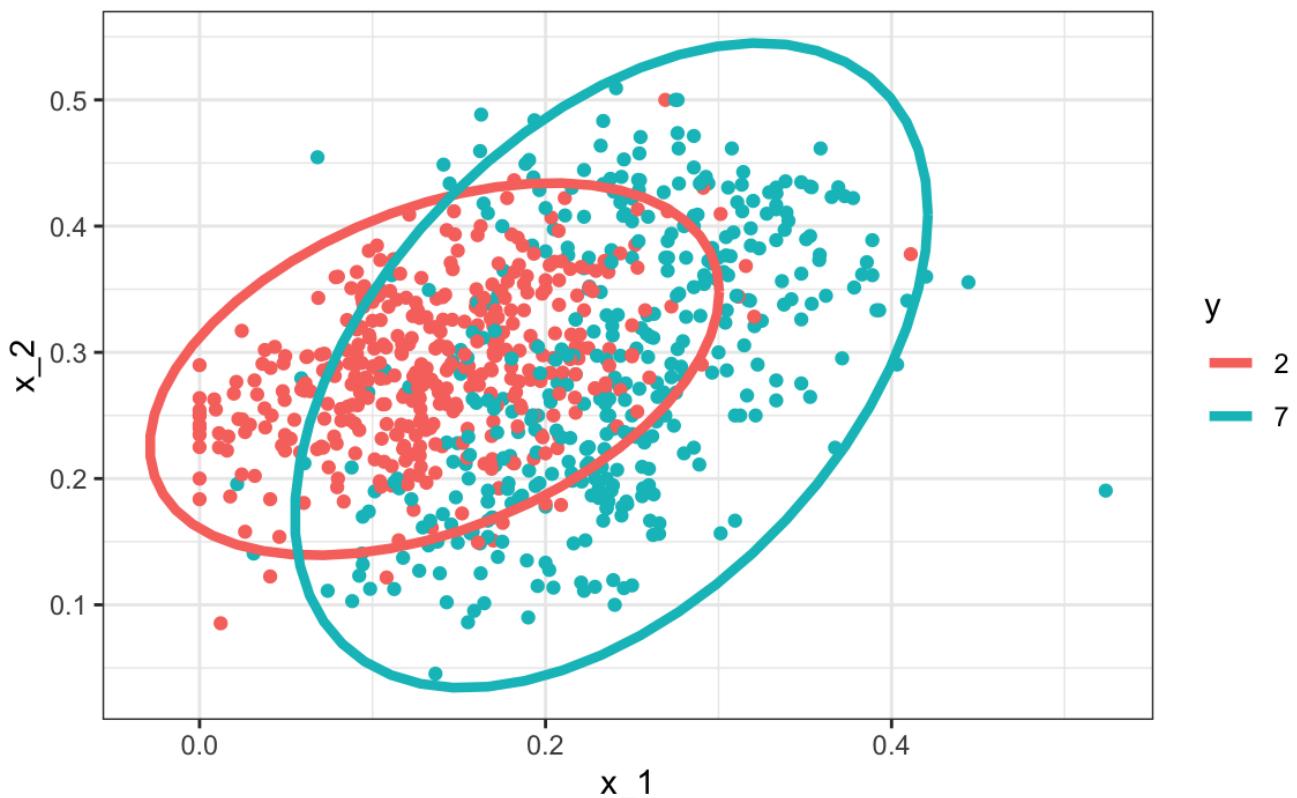


Figure 5:

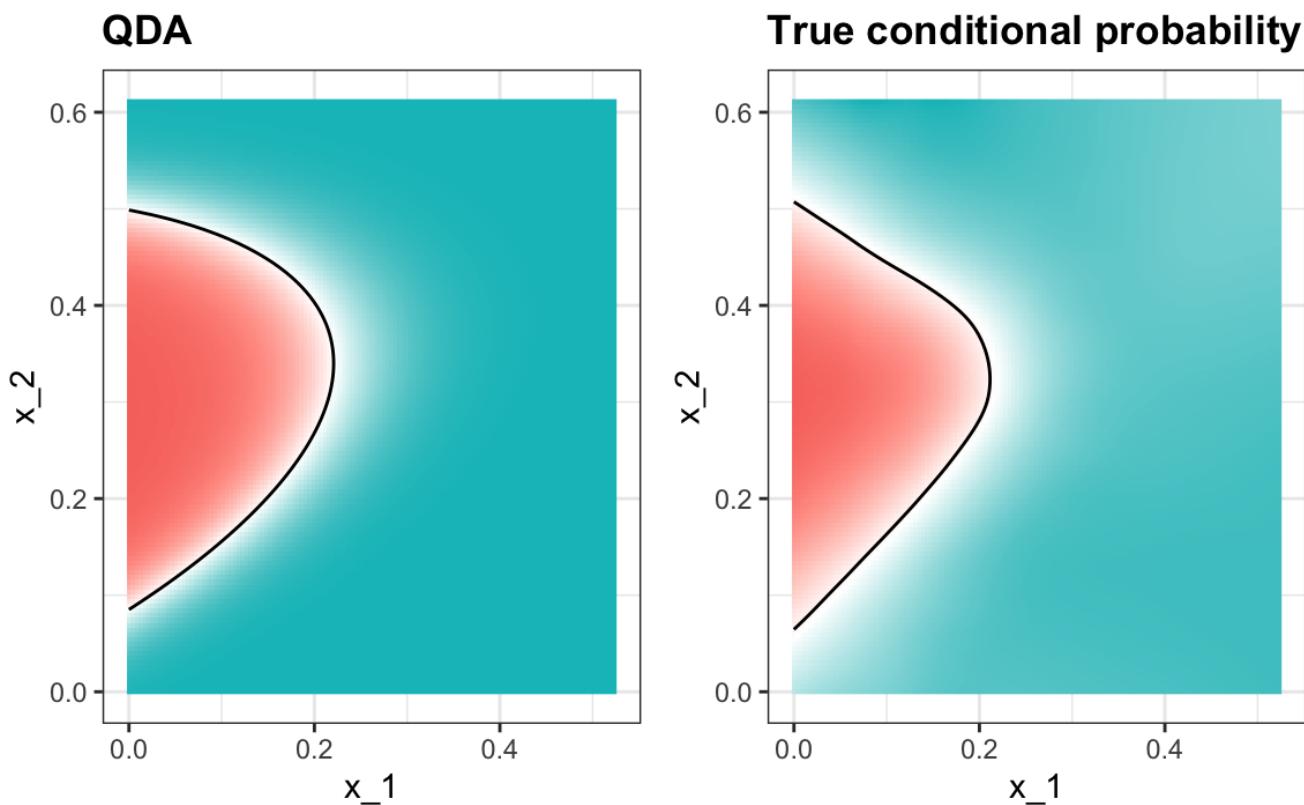


Figure 6:

points for the 7s:

```
mnist_27$train |> mutate(y = factor(y)) |>
  ggplot(aes(x_1, x_2, fill = y, color = y)) +
  geom_point(show.legend = FALSE) +
  stat_ellipse(type = "norm") +
  facet_wrap(~y)
```

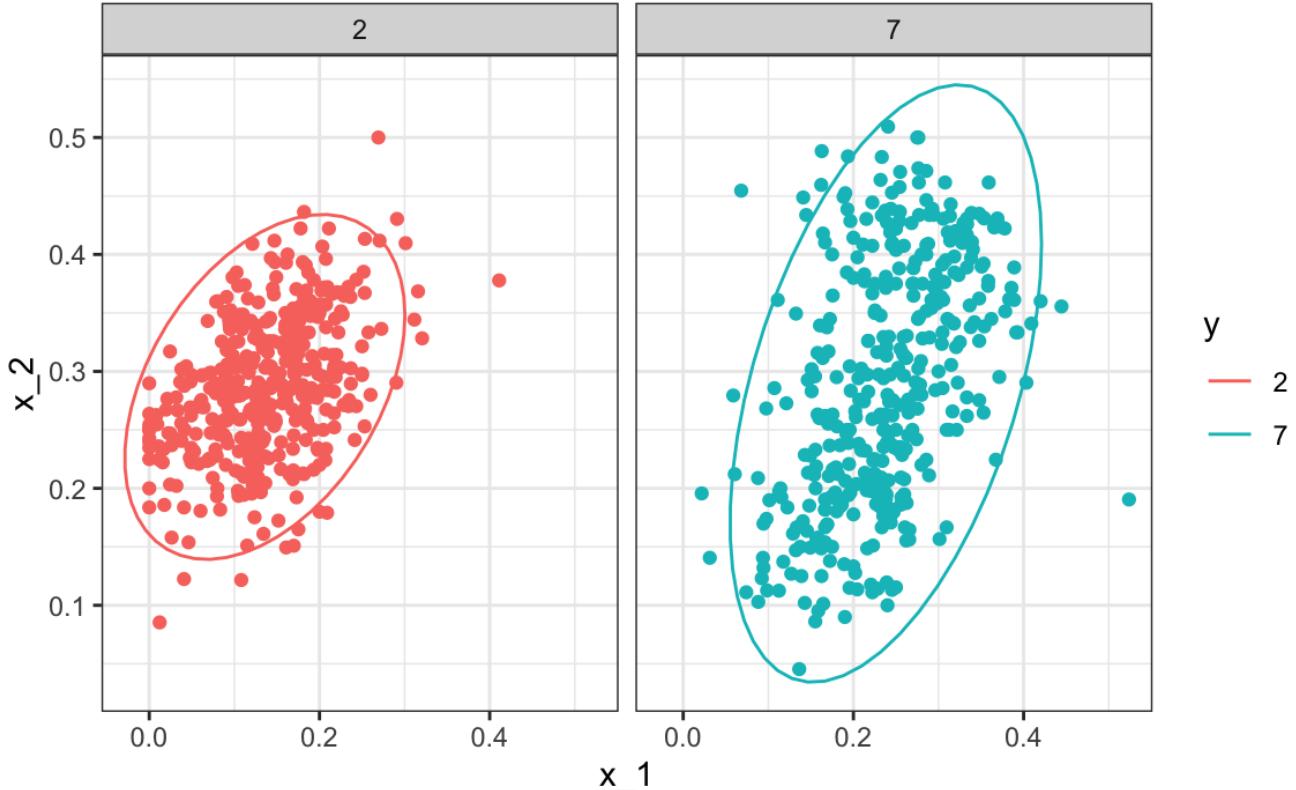


Figure 7:

QDA can work well here, but it becomes harder to use as the number of predictors increases. Here we have 2 predictors and had to compute 4 means, 4 SDs, and 2 correlations. Notice that if we have 10 predictors, we have 45 correlations for each class. In general, the formula is $(K \times p(p-1)/2)$, which gets big fast. Once the number of parameters approaches the size of our data, the method becomes impractical due to overfitting.

30.3.4 Linear discriminant analysis

A relatively simple solution to QDA's problem of having too many parameters is to assume that the correlation structure is the same for all classes, which reduces the number of parameters we need to estimate. In this case, the distributions look like this:

We can LDA using the **MASS** `lda` function:

Now the size of the ellipses as well as the angles are the same. This is because they are assumed to have the same standard deviations and correlations. Although this added constraint lowers the number of parameters, the rigidity lowers our accuracy to:

```
confusionMatrix(y_hat, mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.775
```

When we force this assumption, we can show mathematically that the boundary is a line, just as with logistic regression. For this reason, we call the method *linear* discriminant analysis (LDA). Similarly, for QDA, we can show

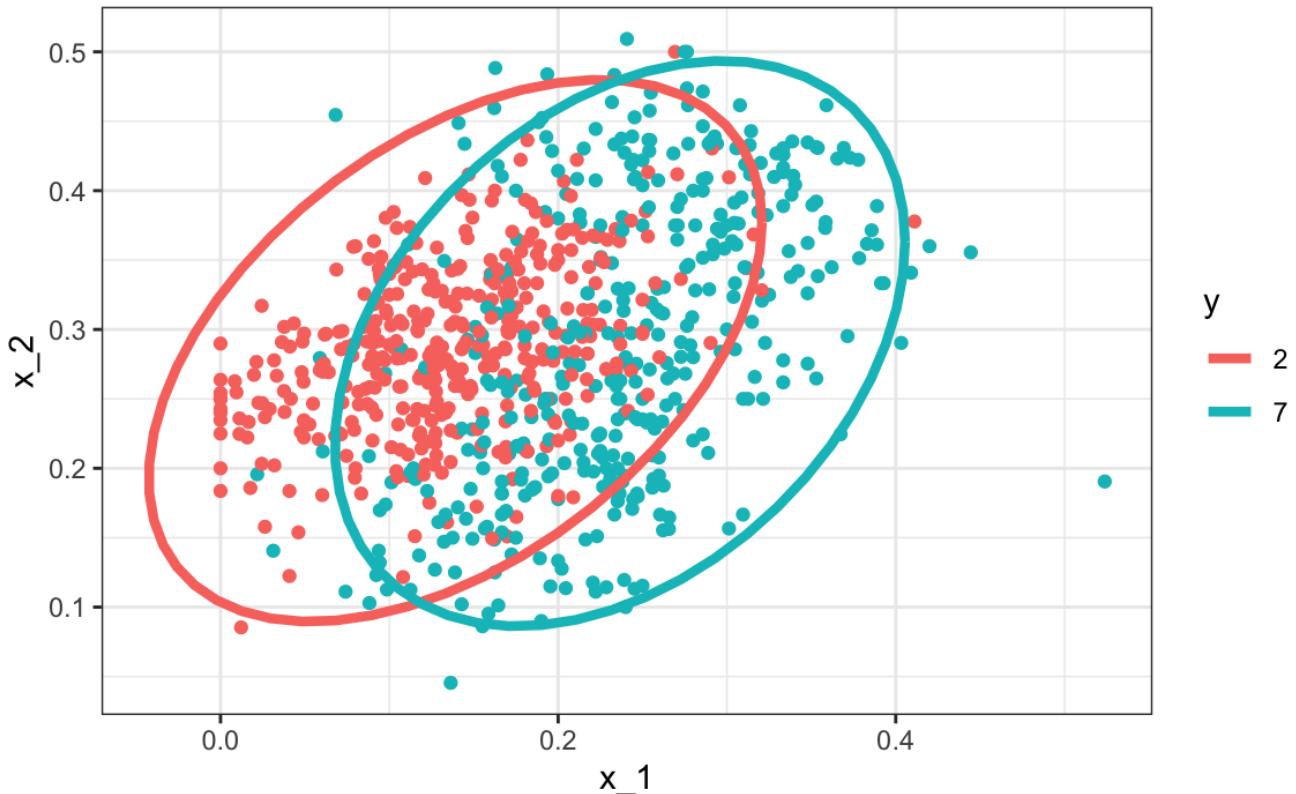


Figure 8:

that the boundary must be a quadratic function.

In the case of LDA, the lack of flexibility does not permit us to capture the non-linearity in the true conditional probability function.

30.3.5 Connection to distance

The normal density is:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

If we remove the constant $(1/\sqrt{2\pi}\sigma)$ and then take the log, we get:

$$-\frac{(x-\mu)^2}{2\sigma^2}$$

which is the negative of a distance squared scaled by the standard deviation. For higher dimensions, the same is true except the scaling is more complex and involves correlations.

You are now ready to do exercises 14-22.

30.4 Classification and regression trees (CART)

30.4.1 The curse of dimensionality

We described how methods such as LDA and QDA are not meant to be used with many predictors (p) because the number of parameters that we need to estimate becomes too large. For example, with the digits example ($p = 784$), we would have over 600,000 parameters with LDA, and we would multiply that by the number of classes for QDA. Kernel methods, such as kNN or local regression, do not have model parameters to estimate. However, they also face a challenge when multiple predictors are used due to what is referred to as the *curse of dimensionality*.

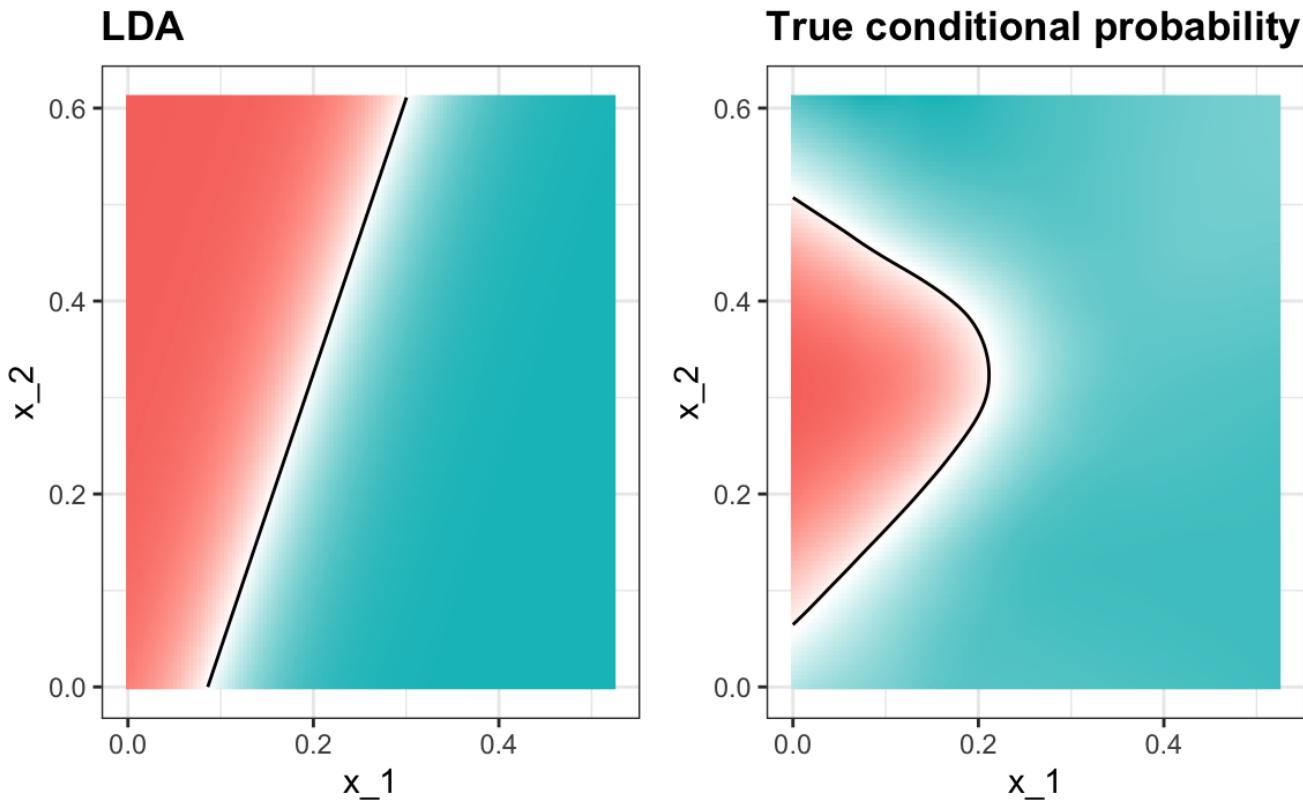


Figure 9:

The *dimension* here refers to the fact that when we have (p) predictors, the distance between two observations is computed in (p) -dimensional space.

A useful way of understanding the curse of dimensionality is by considering how large we have to make a span/neighborhood/window to include a given percentage of the data. Remember that with larger neighborhoods, our methods lose flexibility, and to be flexible we need to keep the neighborhoods small.

To see how this becomes an issue for higher dimensions, suppose we have one continuous predictor with equally spaced points in the $[0,1]$ interval and we want to create windows that include 1/10th of the data. Then it's easy to see that our windows have to be of size 0.1:

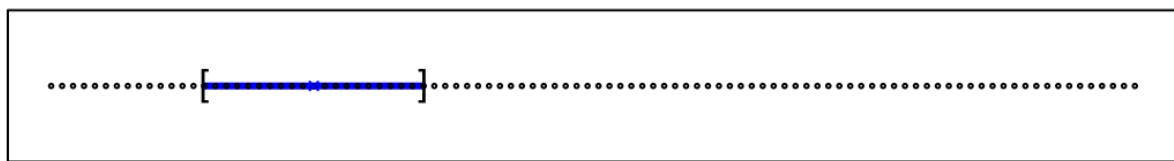


Figure 10:

Now for two predictors, if we decide to keep the neighborhood just as small, 10% for each dimension, we include only 1 point. If we want to include 10% of the data, then we need to increase the size of each side of the square to $(\sqrt{.10}) \approx .316$:

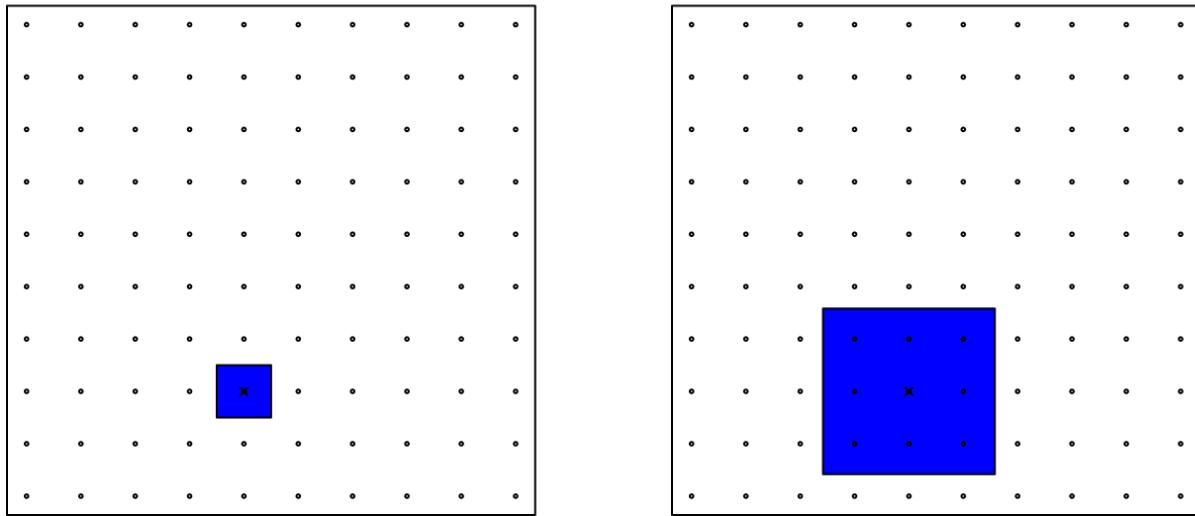


Figure 11:

Using the same logic, if we want to include 10% of the data in a three-dimensional space, then the side of each cube is $\sqrt[3]{0.10} \approx 0.464$. In general, to include 10% of the data in a case with p dimensions, we need an interval with each side of size $\sqrt[p]{0.10}$ of the total. This proportion gets close to 1 quickly, and if the proportion is 1 it means we include all the data and are no longer smoothing.

By the time we reach 100 predictors, the neighborhood is no longer very local, as each side covers almost the entire dataset.

Here we look at a set of elegant and versatile methods that adapt to higher dimensions and also allow these regions to take more complex shapes while still producing models that are interpretable. These are very popular, well-known and studied methods. We will concentrate on regression and decision trees and their extension to random forests.

30.4.2 CART motivation

To motivate this section, we will use a new dataset that includes the breakdown of the composition of olive oil into 8 fatty acids:

```
names(olive)
#> [1] "region"        "area"          "palmitic"      "palmitoleic"
#> [5] "stearic"       "oleic"         "linoleic"      "linolenic"
#> [9] "arachidic"     "eicosenoic"
```

For illustrative purposes, we will try to predict the region using the fatty acid composition values as predictors.

```
table(olive$region)
#>
#> Northern Italy      Sardinia Southern Italy
#>           151            98            323
```

We remove the `area` column because we won't use it as a predictor.

```
olive <- select(olive, -area)
```

Using kNN, we can achieve a test set accuracy of 0.9717332. However, a bit of data exploration reveals that we should be able to do even better. For example, if we look at the distribution of each predictor stratified by region we see that eicosenoic is only present in Southern Italy and that linoleic separates Northern Italy from Sardinia.

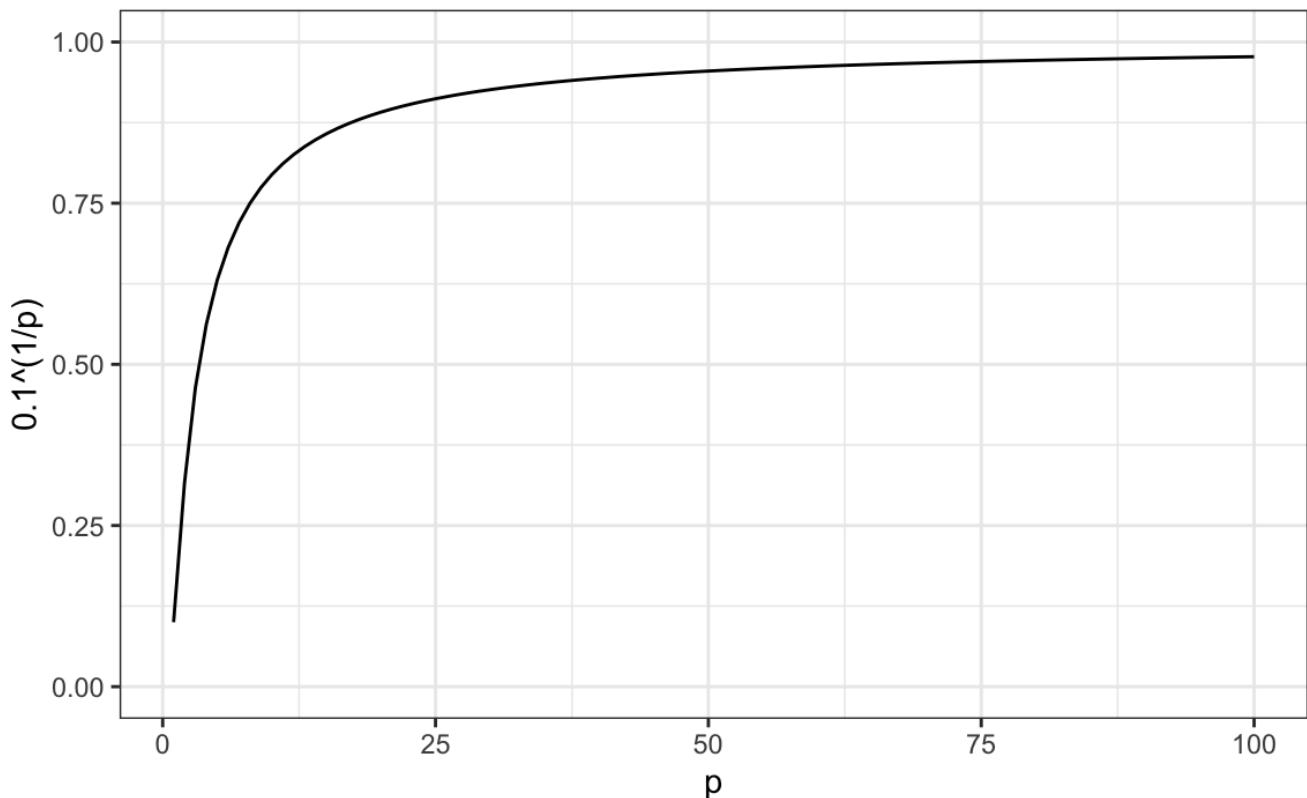


Figure 12:

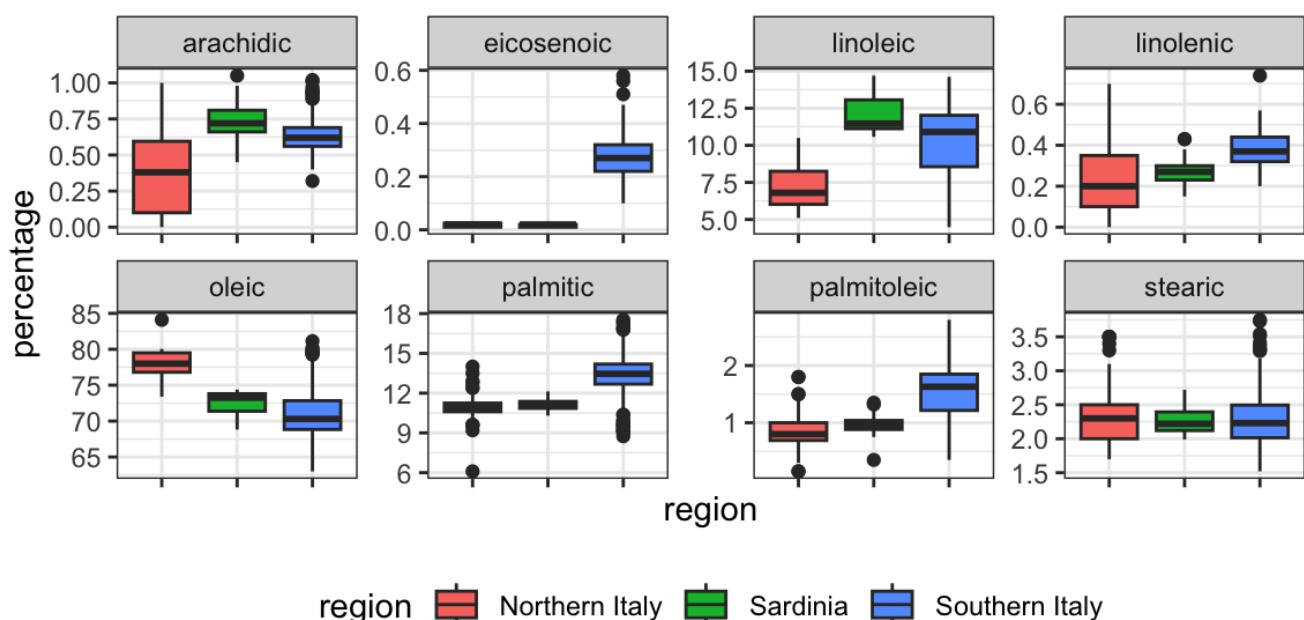


Figure 13:

This implies that we should be able to build an algorithm that predicts perfectly! We can see this clearly by plotting the values for eicosenoic and linoleic.

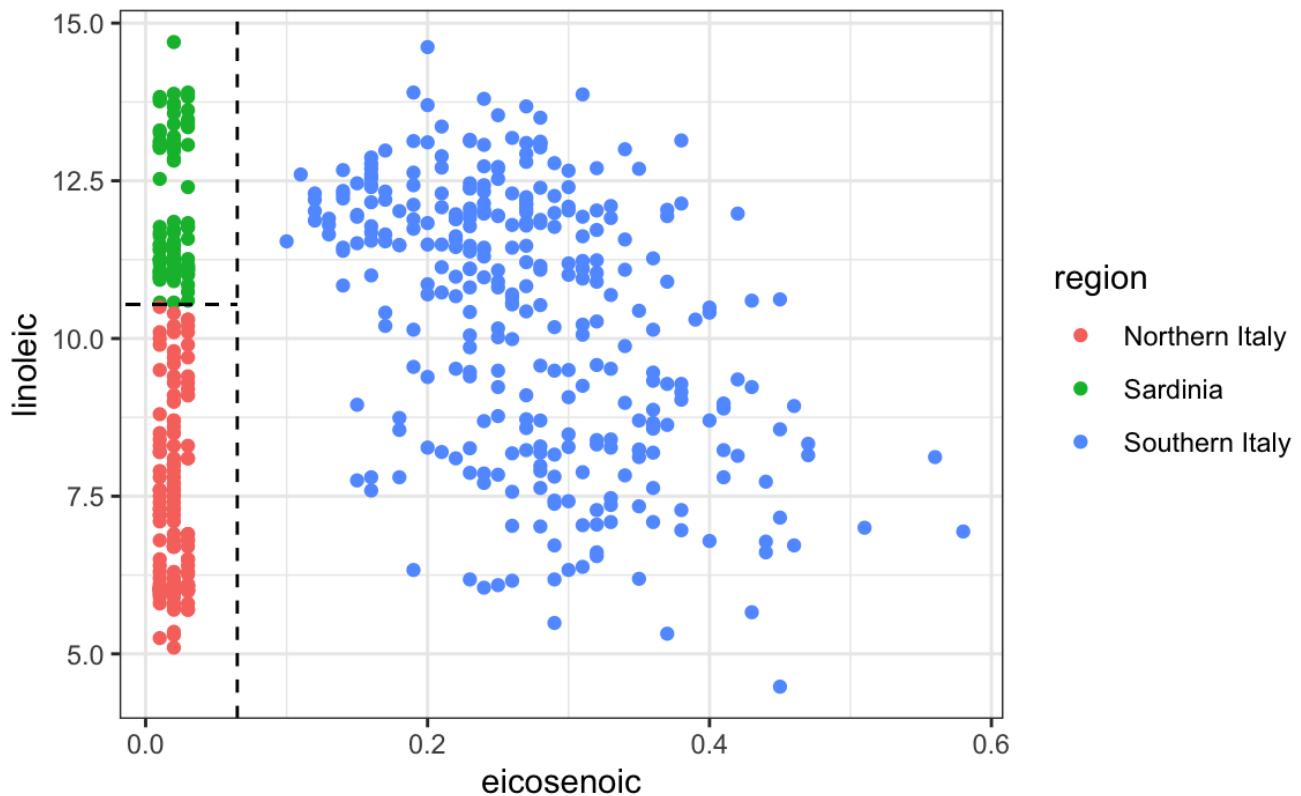


Figure 14:

In Section 21.4, we defined *predictor spaces*, which in this case consists of eight-dimensional points with values between 0 and 100. In the plot above, we show the space defined by the two predictors eicosenoic and linoleic, and, by eye, we can construct a prediction rule that partitions the predictor space so that each partition contains only outcomes of a one category.

This in turn can be used to define an algorithm with perfect accuracy. Specifically, we define the following decision rule: if eicosenoic is larger than 0.065, predict Southern Italy. If not, then if linoleic is larger than $\backslash(10.535\backslash)$, predict Sardinia, and if lower, predict Northern Italy. We can draw this decision tree as follows:

Decision trees like this are often used in practice. For example, to decide on a person's risk of poor outcome after having a heart attack, doctors use the following:

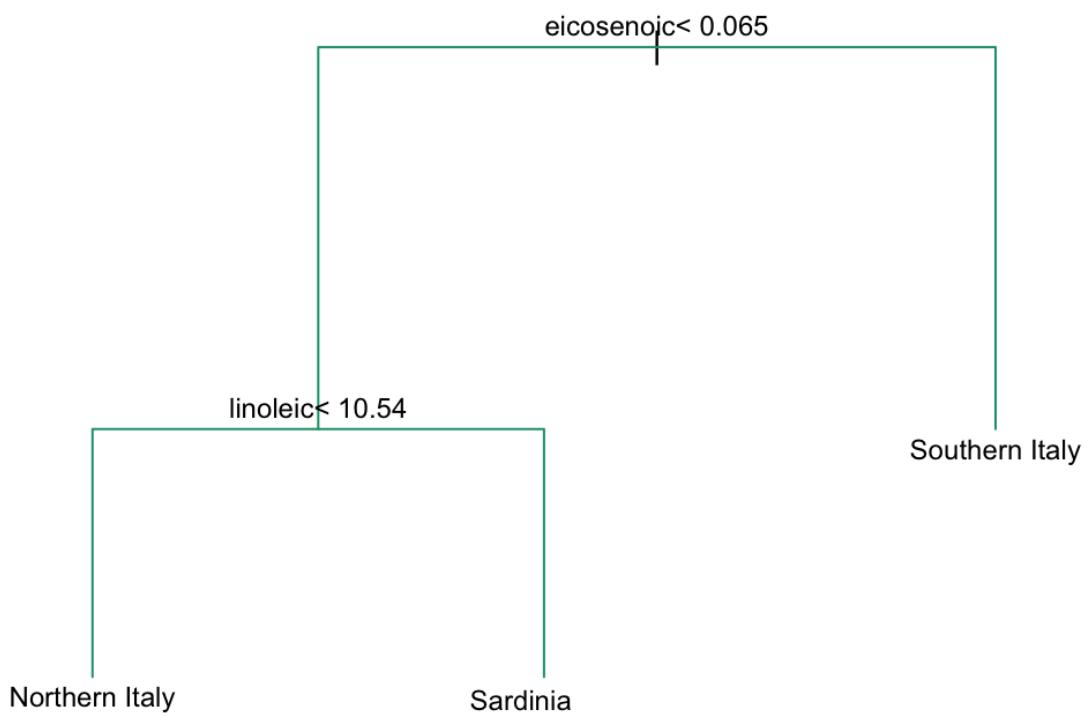
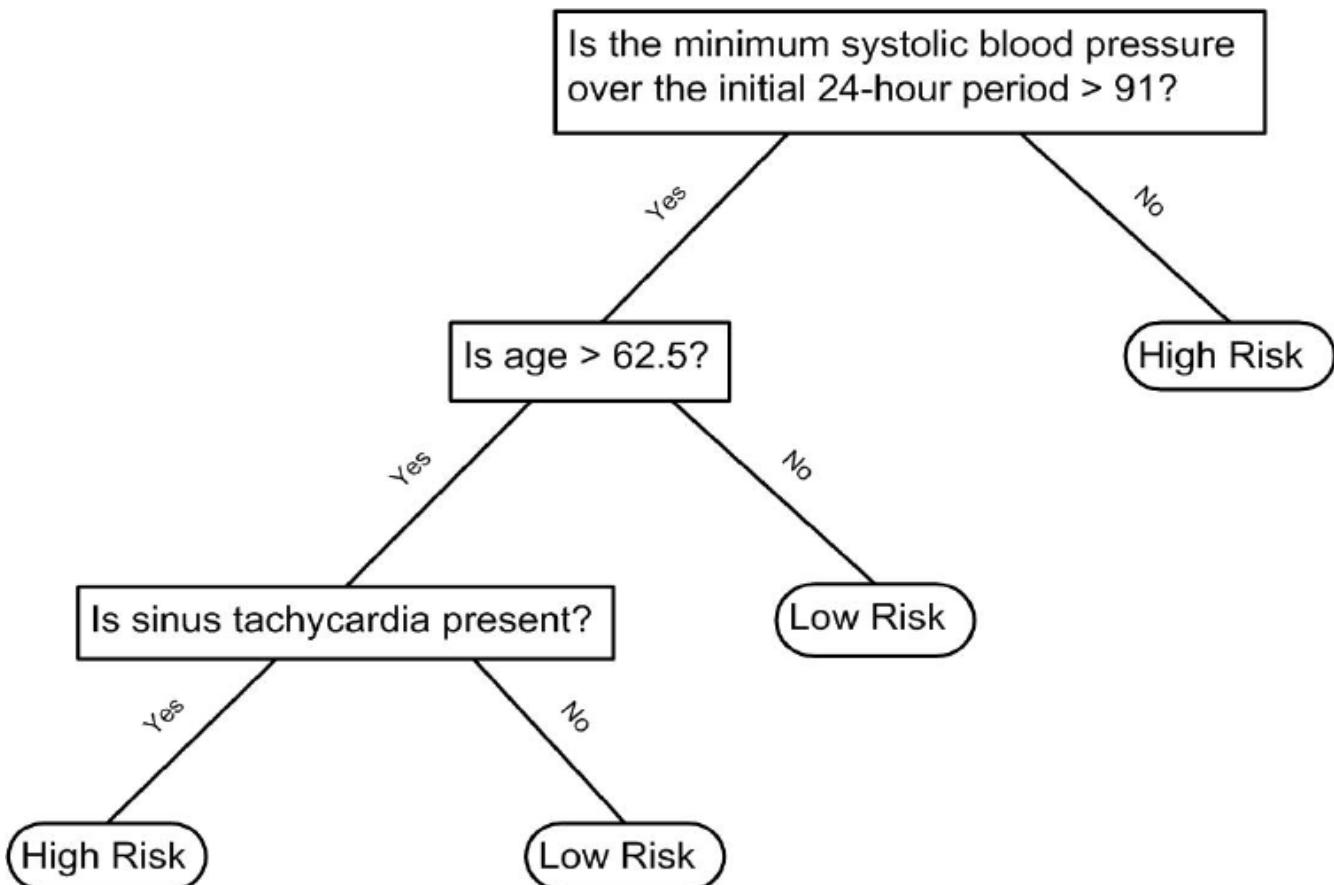


Figure 15:



(Source: Walton 2010 Informal Logic, Vol. 30, No. 2, pp. 159-184².)

A tree is basically a flow chart of yes or no questions. The general idea of the methods we are describing is to define an algorithm that uses data to create these trees with predictions at the ends, referred to as *nodes*. Regression and decision trees operate by predicting an outcome variable $\langle y \rangle$ by partitioning the predictors.

30.4.3 Regression trees

When using trees, and the outcome is continuous, we call the approach a *regression tree*. To introduce regression trees, we will use the 2008 poll data used in previous sections to describe the basic idea of how we build these algorithms. As with other machine learning algorithms, we will try to estimate the conditional expectation $\langle f(x) = \text{mbox}\{E\}(Y | X = x) \rangle$ with $\langle Y \rangle$ the poll margin and $\langle x \rangle$ the day.

The general idea here is to build a decision tree and, at the end of each *node*, obtain a predictor $\langle \hat{y} \rangle$. A mathematical way to describe this is: we are partitioning the predictor space into $\langle J \rangle$ non-overlapping regions, $\langle R_1, R_2, \dots, R_J \rangle$, and then for any predictor $\langle x \rangle$ that falls within region $\langle R_j \rangle$, estimate $\langle f(x) \rangle$ with the average of the training observations $\langle y_i \rangle$ for which the associated predictor $\langle x_i \rangle$ is also in $\langle R_j \rangle$.

But how do we decide on the partition $\langle R_1, R_2, \dots, R_J \rangle$ and how do we choose $\langle J \rangle$? Here is where the algorithm gets a bit complicated.

Regression trees create partitions recursively. We start the algorithm with one partition, the entire predictor space. In our simple first example, this space is the interval $[-155, 1]$. But after the first step, we will have two partitions. After the second step, we will split one of these partitions into two and will have three partitions, then four, and so on. Later we describe how we pick the partition to further partition, and when to stop.

For each existing partition, we find a predictor $\langle j \rangle$ and value $\langle s \rangle$ that define two new partitions, which we will call $\langle R_{1(j,s)} \rangle$ and $\langle R_{2(j,s)} \rangle$, that split our observations in the current partition by asking if $\langle x_j \rangle$ is bigger than $\langle s \rangle$:

$$\langle R_{1(j,s)} = \{ \mathbf{x} \mid x_j < s \} \text{ and } R_{2(j,s)} = \{ \mathbf{x} \mid x_j \geq s \} \rangle$$

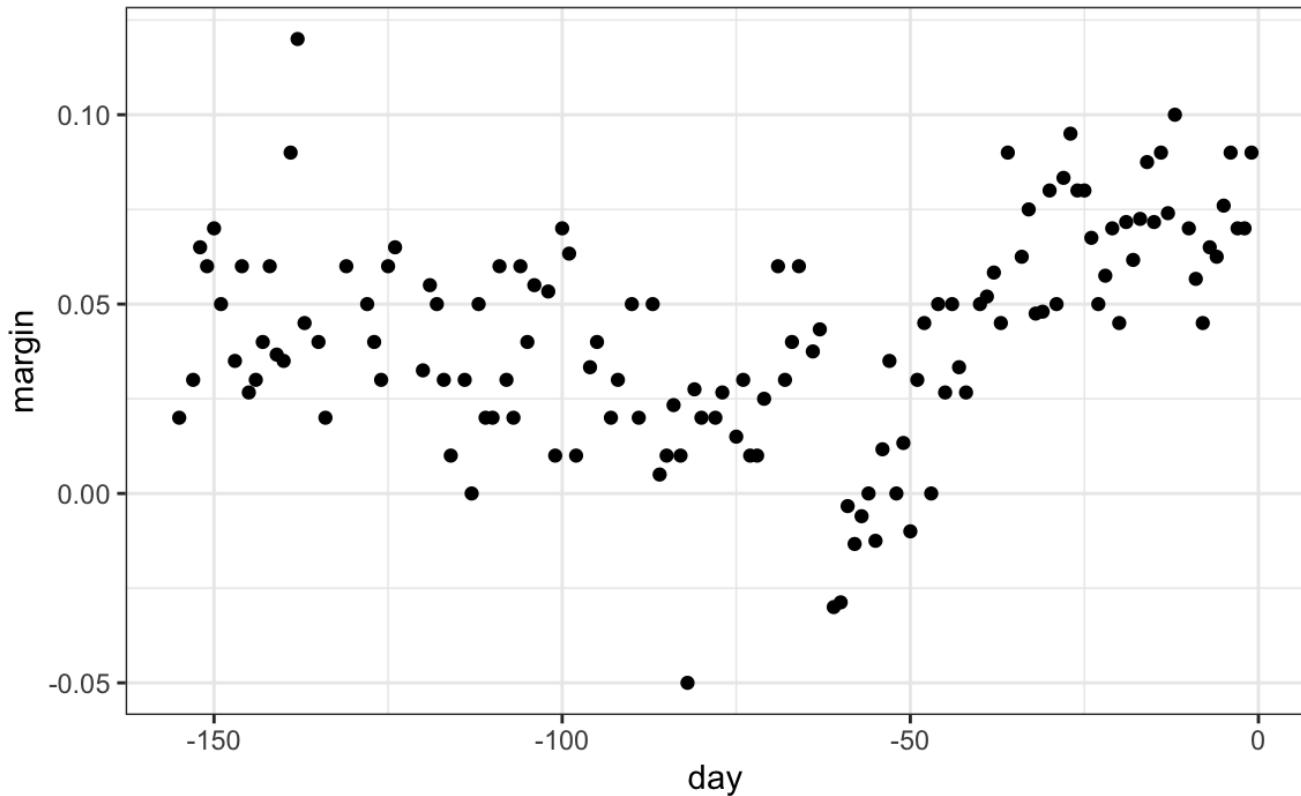


Figure 16:

In our current example, we only have one predictor, so we will always choose $\langle j = 1 \rangle$, but in general this will not be the case. Now after we define the new partitions $\langle R_1 \rangle$ and $\langle R_2 \rangle$, and we decide to stop the partitioning, we compute predictors by taking the average of all the observations $\langle y \rangle$ for which the associated $\langle \mathbf{x} \rangle$ is in $\langle R_1 \rangle$ and $\langle R_2 \rangle$. We refer to these two as $\langle \hat{y}_{R_1} \rangle$ and $\langle \hat{y}_{R_2} \rangle$ respectively.

But how do we pick $\langle j \rangle$ and $\langle s \rangle$? Basically we find the pair that minimizes the residual sum of squares (RSS):

$$\left[\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \right]$$

This is then applied recursively to the new regions $\langle R_1 \rangle$ and $\langle R_2 \rangle$. We describe how we stop later, but once we are done partitioning the predictor space into regions, in each region a prediction is made using the observations in that region.

Let's take a look at what this algorithm does on the 2008 presidential election poll data. We will use the **rpart** function in the **rpart** package.

```
library(rpart)
fit <- rpart(margin ~ ., data = polls_2008)
```

In this case, there is only one predictor. Thus we do not have to decide which predictor $\langle j \rangle$ to split by, we simply have to decide what value $\langle s \rangle$ we use to split. We can visually see where the splits were made:

```
plot(fit, margin = 0.1)
text(fit, cex = 0.75)
```

The first split is made on day 39.5. One of those regions is then split at day 86.5. The two resulting new partitions are split on days 49.5 and 117.5, respectively, and so on. We end up with 8 partitions. The final estimate $\langle \hat{f}(x) \rangle$ looks like this:

```
polls_2008 |>
  mutate(y_hat = predict(fit)) |>
```

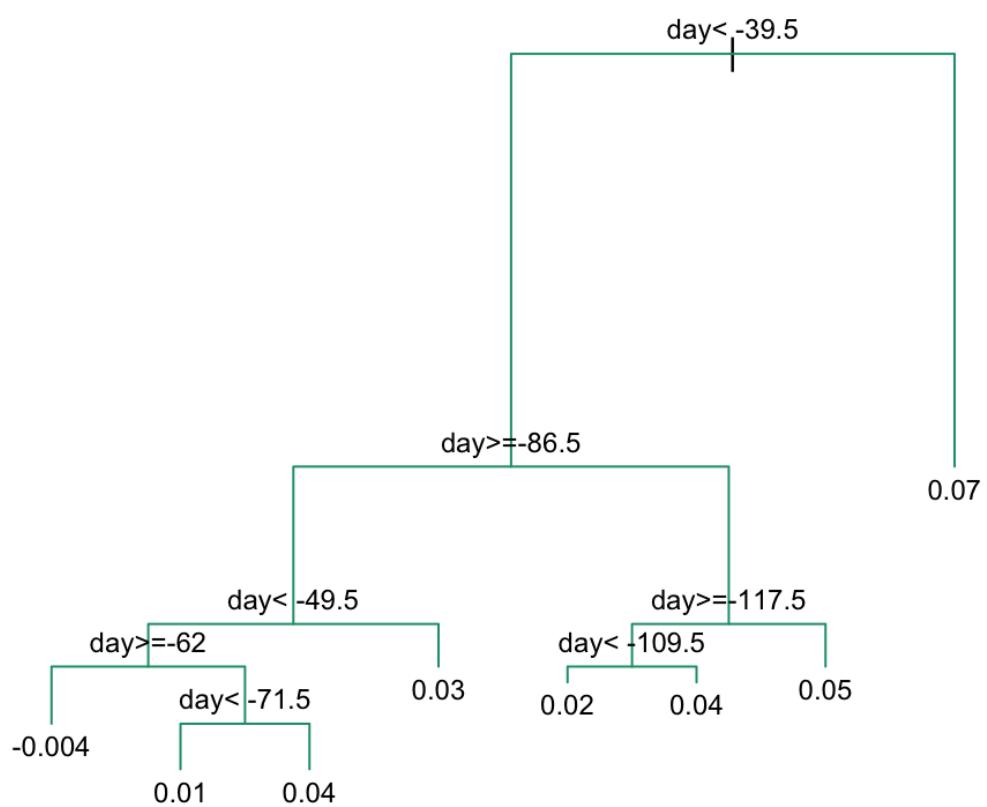


Figure 17:

```
ggplot() +
  geom_point(aes(day, margin)) +
  geom_step(aes(day, y_hat), col = "red")
```

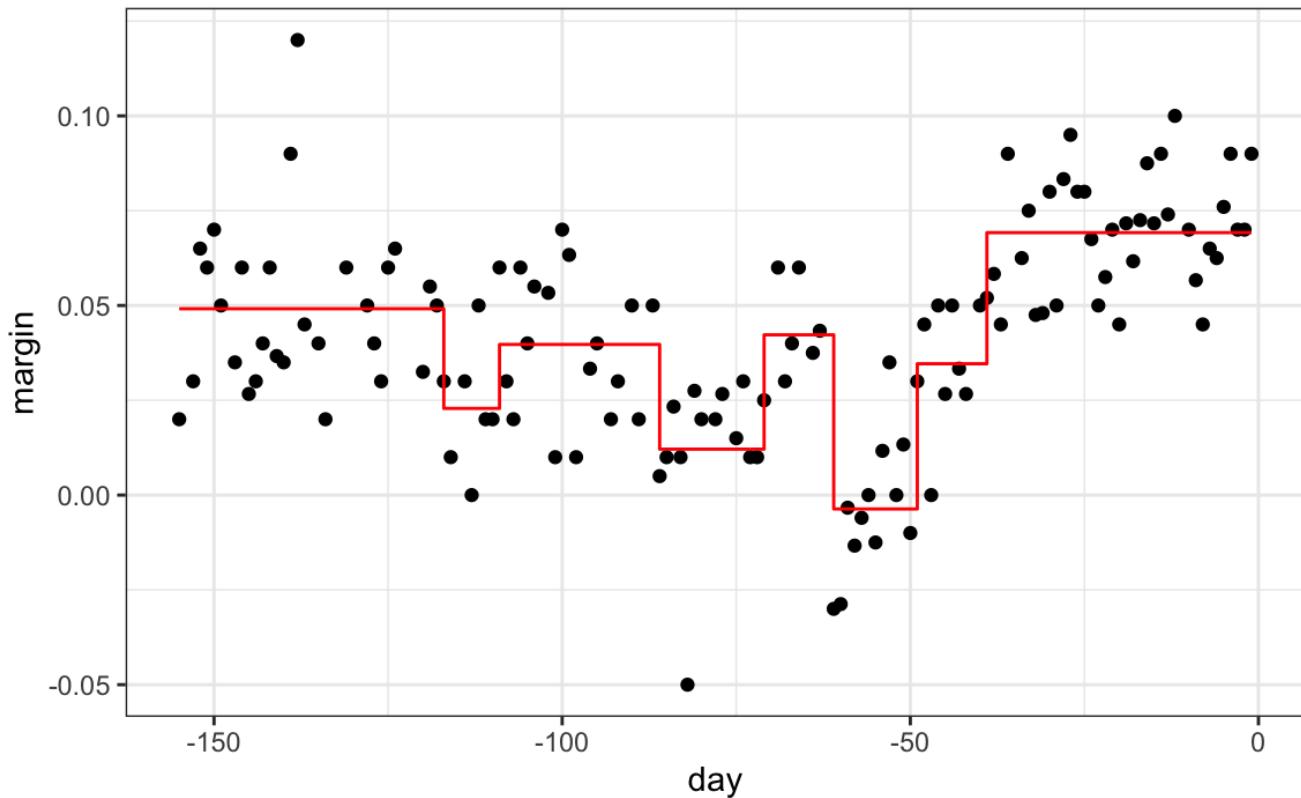


Figure 18:

Note that the algorithm stopped partitioning at 8. The decision is made based on a measure referred to as *complexity parameter* (cp). Every time we split and define two new partitions, our training set RSS decreases. This is because with more partitions, our model has more flexibility to adapt to the training data. In fact, if you split until every point is its own partition, then RSS goes all the way down to 0 since the average of one value is that same value. To avoid this, the algorithm sets a minimum for how much the RSS must improve for another partition to be added. This parameter is referred to as the *complexity parameter* (cp). The RSS must improve by a factor of cp for the new partition to be added. Large values of cp will therefore force the algorithm to stop earlier, which results in fewer nodes.

However, cp is not the only parameter used to decide if we should partition a current partition or not. Another common parameter is the minimum number of observations required in a partition before partitioning it further. The argument used in the `rpart` function is `minsplit` and the default is 20. The `rpart` implementation of regression trees also permits users to determine a minimum number of observations in each node. The argument is `minbucket` and defaults to `round(minsplit/3)`.

As expected, if we set `cp = 0` and `minsplit = 2`, then our prediction is as flexible as possible and our predictor is our original data:

Intuitively we know that this is not a good approach as it will generally result in over-training. These `cp`, `minsplit`, and `minbucket`, three parameters can be used to control the variability of the final predictors. The larger these values are the more data is averaged to compute a predictor and thus reduce variability. The drawback is that it restricts flexibility.

So how do we pick these parameters? We can use cross validation, just like with any tuning parameter. Here is the resulting tree when we use cross validation to choose cp:

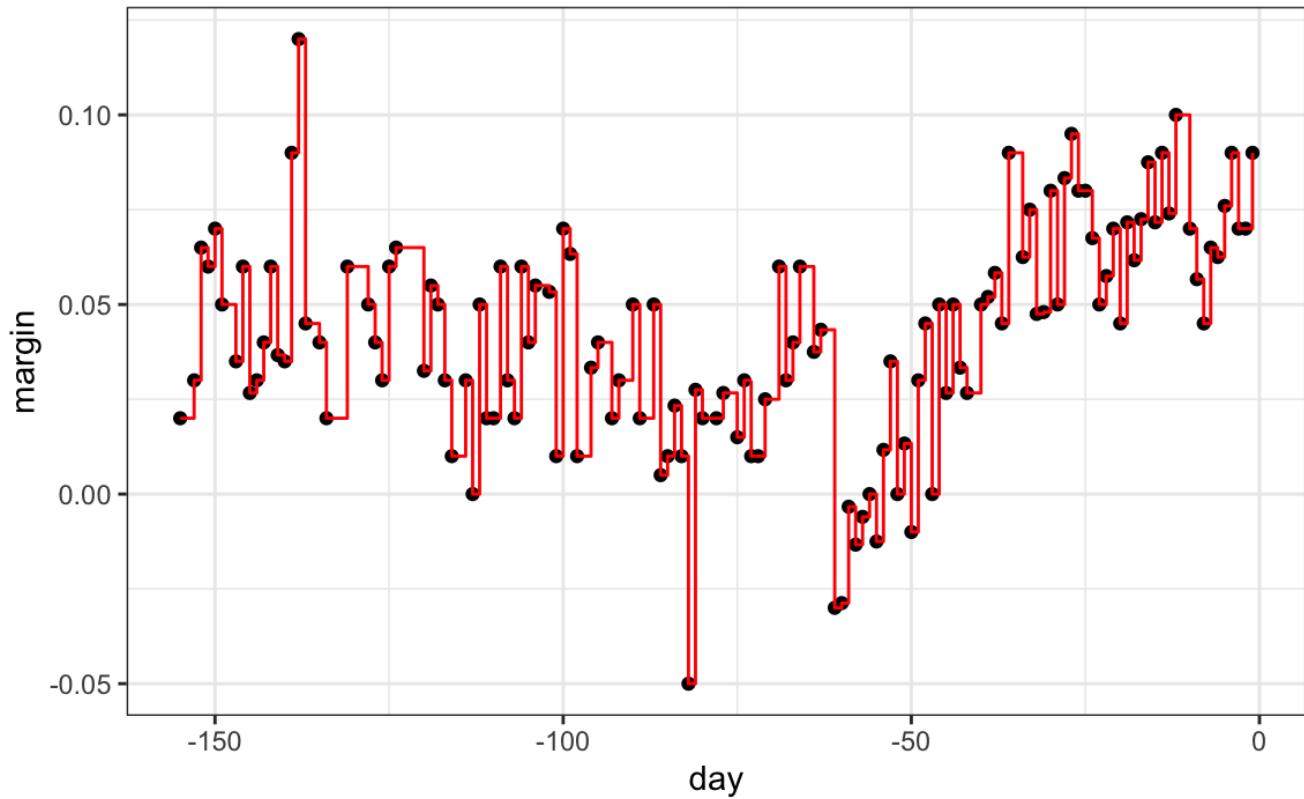


Figure 19:

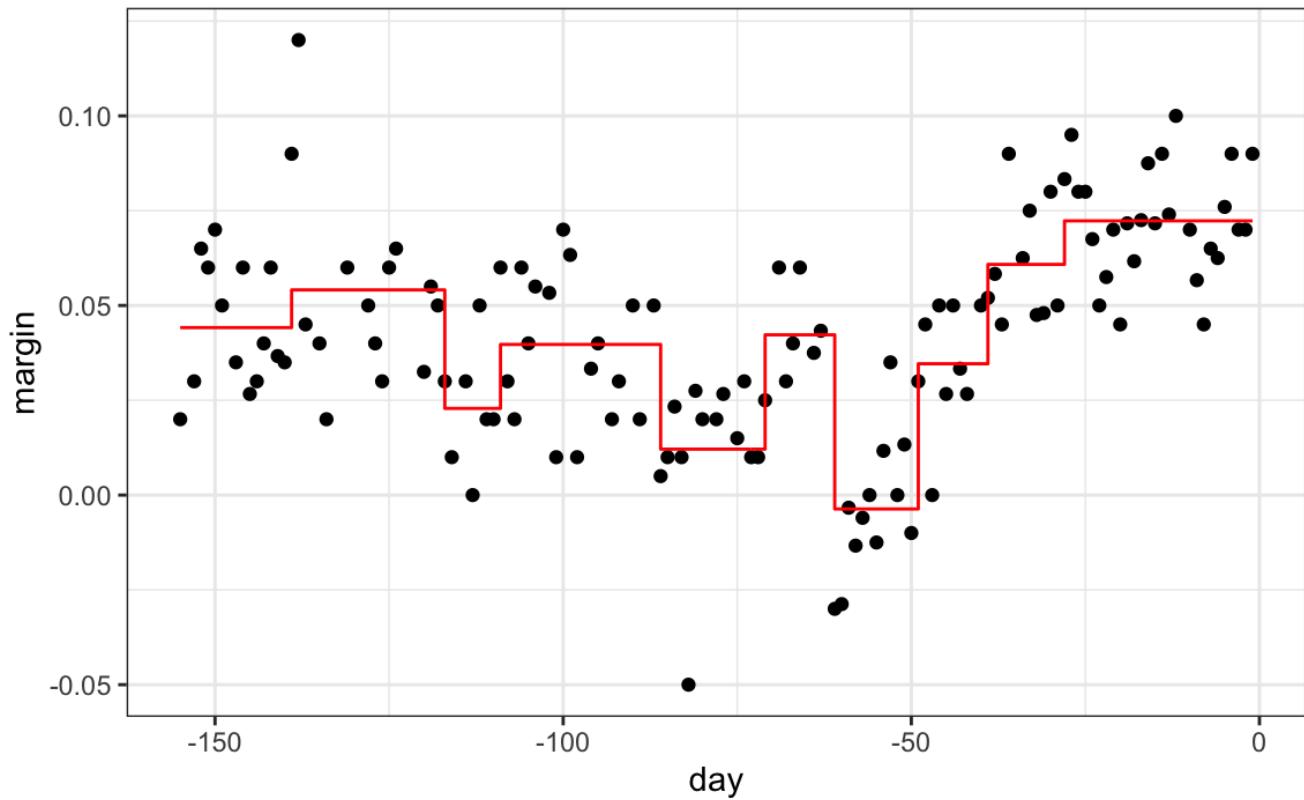


Figure 20:

Note that if we already have a tree and want to apply a higher cp value, we can use the `prune` function. We call this *pruning* a tree because we are snipping off partitions that do not meet a cp criterion. Here is an example where we create a tree that used a `cp = 0` and then we prune it back:

```
fit <- rpart(margin ~ ., data = polls_2008, control = rpart.control(cp = 0))
pruned_fit <- prune(fit, cp = 0.01)
```

30.4.4 Classification (decision) trees

Classification trees, or decision trees, are used in prediction problems where the outcome is categorical. We use the same partitioning principle with some differences to account for the fact that we are now working with a categorical outcome.

The first difference is that we form predictions by calculating which class is the most common among the training set observations within the partition, rather than taking the average in each partition (as we can't take the average of categories).

The second is that we can no longer use RSS to choose the partition. While we could use the naive approach of looking for partitions that minimize training error, better performing approaches use more sophisticated metrics. Two of the more popular ones are the *Gini Index* and *Entropy*.

In a perfect scenario, the outcomes in each of our partitions are all of the same category since this will permit perfect accuracy. The *Gini Index* is going to be 0 in this scenario, and become larger the more we deviate from this scenario. To define the Gini Index, we define $\sum_{j=1}^K \hat{p}_{j,k} (1 - \hat{p}_{j,k})$ as the proportion of observations in partition j that are of class k . The Gini Index is defined as:

$$\text{Gini}(j) = \sum_{k=1}^K \hat{p}_{j,k} (1 - \hat{p}_{j,k})$$

If you study the formula carefully, you will see that it is in fact 0 in the perfect scenario described above.

Entropy is a very similar quantity, defined as:

$$\text{entropy}(j) = -\sum_{k=1}^K \hat{p}_{j,k} \log(\hat{p}_{j,k}), \text{ with } 0 \times \log(0) \text{ defined as } 0$$

If we use a classification tree on the 2 or 7 example, we achieve an accuracy of 0.81 which is better than regression, but is not as good as what we achieved with kernel methods.

The plot of the estimated conditional probability shows us the limitations of classification trees:

Note that with decision trees, it is difficult to make the boundaries smooth since each partition creates a discontinuity.

Classification trees have certain advantages that make them very useful. They are highly interpretable, even more so than linear models. They are also easy to visualize (if small enough). Finally, they can model human decision processes and don't require use of dummy predictors for categorical variables. On the other hand, the approach via recursive partitioning can easily over-train and is therefore a bit harder to train than, for example, linear regression or kNN. Furthermore, in terms of accuracy, it is rarely the best performing method since it is not very flexible and is highly unstable to changes in training data. Random forests, explained next, improve on several of these shortcomings.

30.5 Random forests

Random forests are a **very popular** machine learning approach that addresses the shortcomings of decision trees using a clever idea. The goal is to improve prediction performance and reduce instability by *averaging* multiple decision trees: a *forest* of trees constructed with *randomness*. It has two features that help accomplish this.

The first step is *bootstrap aggregation* or *bagging*. The general idea is to generate many predictors, each using regression or classification trees, and then forming a final prediction based on the average prediction of all these trees. To assure that the individual trees are not the same, we use the bootstrap to induce randomness. These two features combined explain the name: the bootstrap makes the individual trees **randomly** different, and the combination of trees is the **forest**. The specific steps are as follows.

1. Build B decision trees using the training set. We refer to the fitted models as (T_1, T_2, \dots, T_B) .

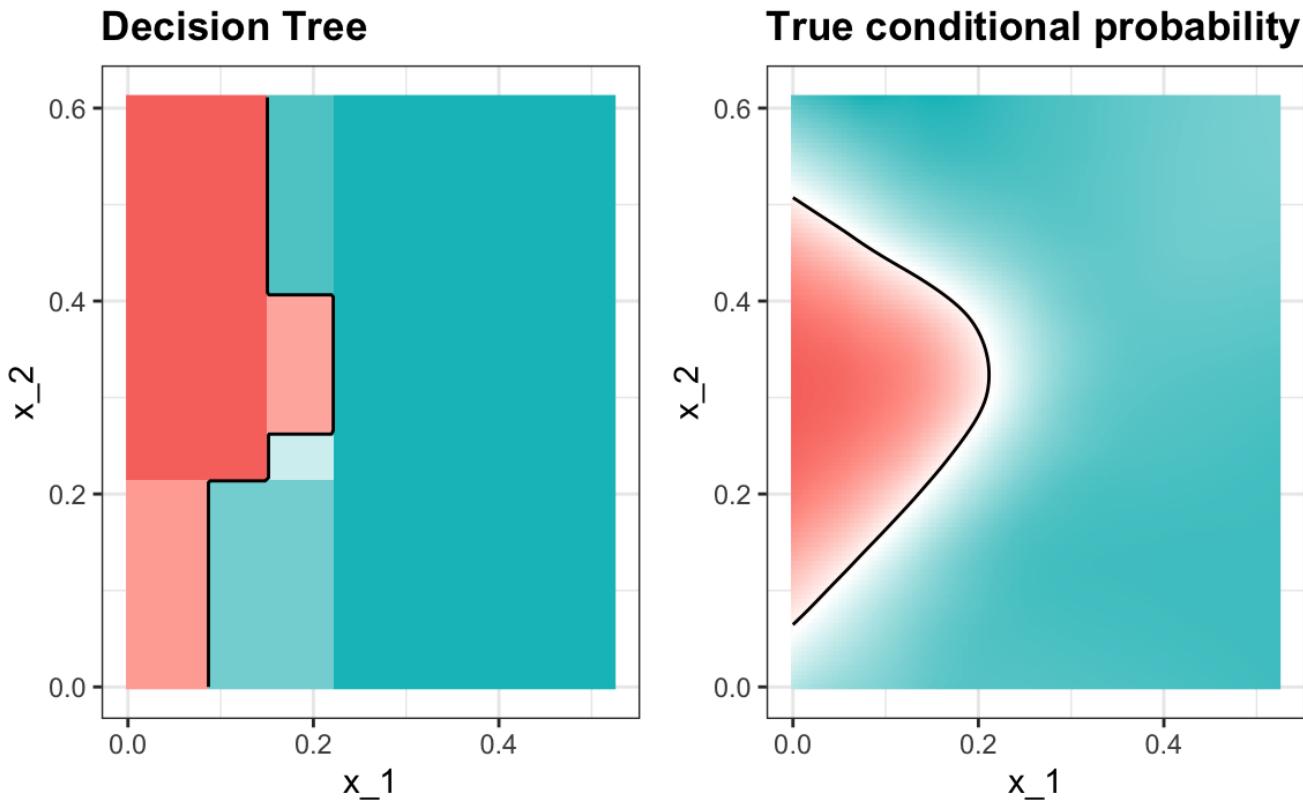


Figure 21:

2. For every observation in the test set, form a prediction \hat{y}_j using tree T_j .
3. For continuous outcomes, form a final prediction with the average $\hat{y} = \frac{1}{B} \sum_{j=1}^B \hat{y}_j$. For categorical data classification, predict \hat{y} with majority vote (most frequent class among $\hat{y}_1, \dots, \hat{y}_T$).

So how do we get different decision trees from a single training set? For this, we use randomness in two ways which we explain in the steps below. Let N be the number of observations in the training set. To create $(T_j, j = 1, \dots, B)$ from the training set we do the following:

1. Create a bootstrap training set by sampling N observations from the training set **with replacement**. This is the first way to induce randomness.
2. A large number of features is typical in machine learning challenges. Often, many features can be informative but including them all in the model may result in overfitting. The second way random forests induce randomness is by randomly selecting features to be included in the building of each tree. A different random subset is selected for each tree. This reduces correlation between trees in the forest, thereby improving prediction accuracy.

To illustrate how the first steps can result in smoother estimates, we will fit a random forest to the 2008 polls data. We will use the `randomForest` function in the `randomForest` package:

```
library(randomForest)
fit <- randomForest(margin ~ ., data = polls_2008)
```

Note that if we apply the function `plot` to the resulting object, we see how the error rate of our algorithm changes as we add trees:

```
rafalib::mypar()
plot(fit)
```

In this case, the accuracy improves as we add more trees until we have used about 30 trees after which accuracy stabilizes.

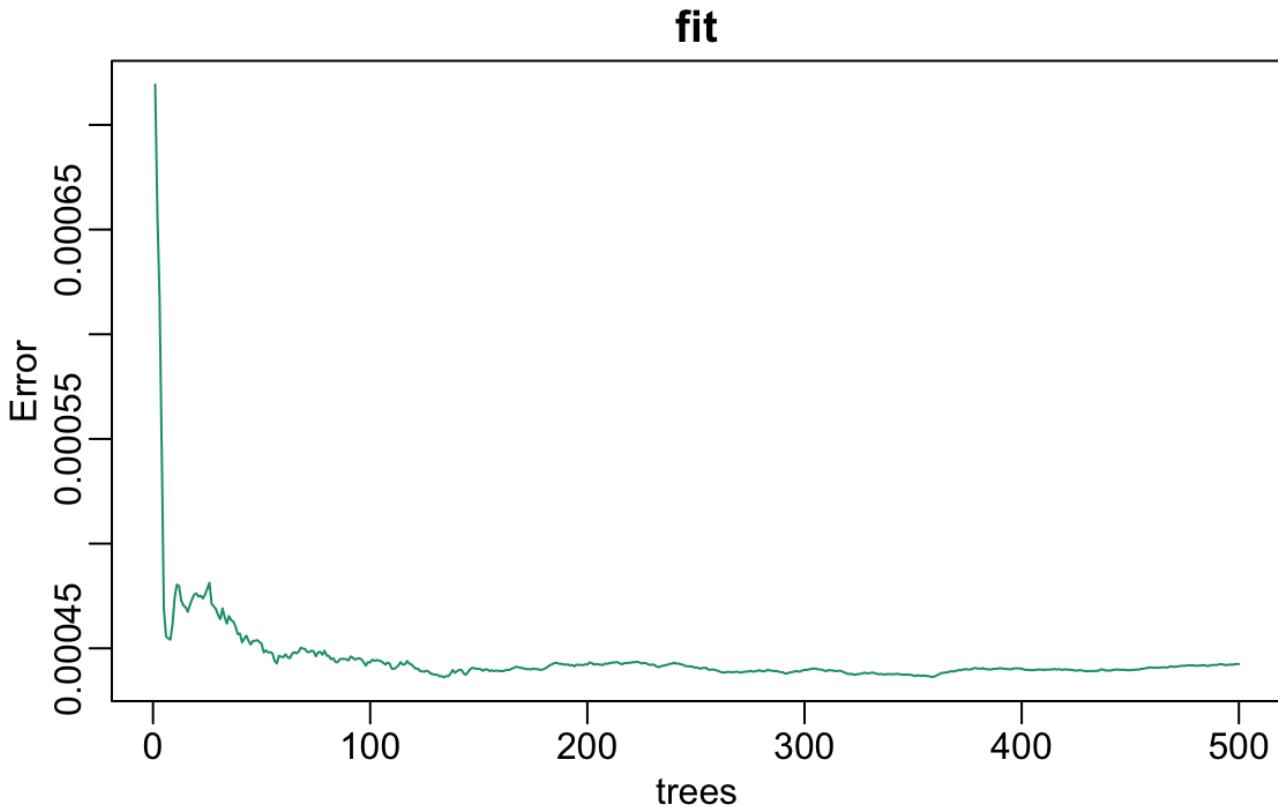


Figure 22:

The resulting estimate for this random forest, obtained with

```
y_hat <- predict(fit, newdata = polls_2008)
```

is shown with the red curve below:

Notice that the random forest estimate is much smoother than what we achieved with the regression tree in the previous section. This is possible because the average of many step functions can be smooth. We can see this by visually examining how the estimate changes as we add more trees. In the following figure, you see each of the bootstrap samples for several values of b and for each one we see the tree that is fitted in grey, the previous trees that were fitted in lighter grey, and the result of averaging all the trees estimated up to that point.

```
library(randomForest)
train_rf <- randomForest(y ~ ., data = mnist_27$train)
```

The accuracy for the random forest fit for our 2 or 7 example is `confusionMatrix(predict(train_rf, mnist_27$test), mnist_27$test$y)$overall["Accuracy"]`. Here is what the conditional probabilities look like:

Visualizing the estimate shows that, although we obtain high accuracy, it appears that there is room for improvement by making the estimate smoother. This could be achieved by changing the parameter that controls the minimum number of data points in the nodes of the tree. The larger this minimum, the smoother the final estimate will be. If we use a node size of 31, the number of neighbors we used with kNN, we get an accuracy of `confusionMatrix(predict(train_rf_2, mnist_27$test), mnist_27$test$y)$overall["Accuracy"]`. The selected model improves accuracy and provides a smoother estimate:

Random forest performs better than trees in all the examples we have considered. However, a disadvantage of random forests is that we lose interpretability. An approach that helps with interpretability is to examine *variable importance*. To define *variable importance*, we count how often a predictor is used in the individual trees. You can learn more about *variable importance* in an advanced machine learning book³. The `caret` package includes the function `varImp` that extracts variable importance from any model in which the calculation is implemented. We give an example on how we use variable importance in the next section.

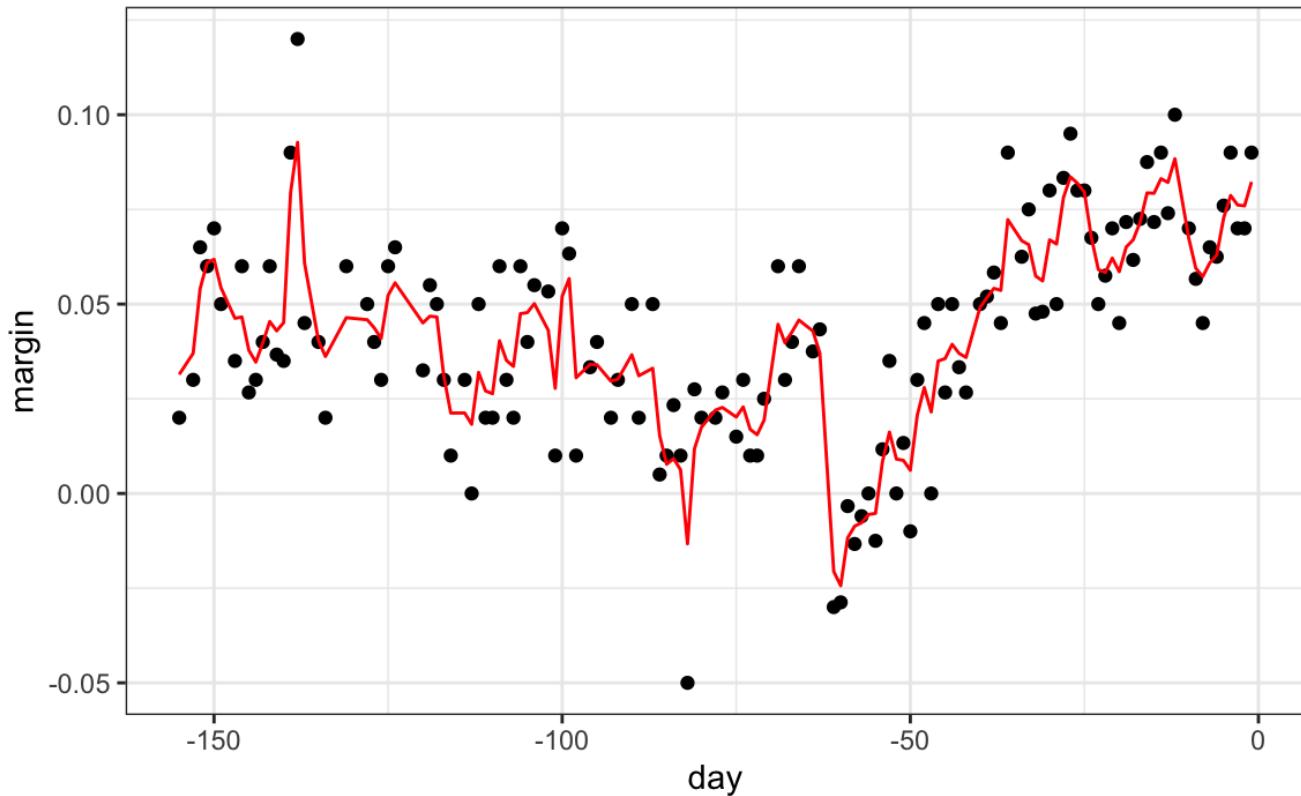


Figure 23:

30.6 Exercises

1. Create a dataset using the following code:

```
n <- 100
Sigma <- 9*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) |>
  data.frame() |> setNames(c("x", "y"))
```

Use the **caret** package to partition into a test and training set of equal size. Train a linear model and report the RMSE. Repeat this exercise 100 times and make a histogram of the RMSEs and report the average and standard deviation. Hint: adapt the code shown earlier like this:

```
library(caret)
y <- dat$y
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- dat |> slice(-test_index)
test_set <- dat |> slice(test_index)
fit <- lm(y ~ x, data = train_set)
y_hat <- fit$coef[1] + fit$coef[2]*test_set$x
sqrt(mean((y_hat - test_set$y)^2))
```

and put it inside a call to **replicate**.

2. Now we will repeat the above but using larger datasets. Repeat exercise 1 but for datasets with `n <- c(100, 500, 1000, 5000, 10000)`. Save the average and standard deviation of RMSE from the 100 repetitions. Hint: use the **sapply** or **map** functions.

3. Describe what you observe with the RMSE as the size of the dataset becomes larger.

- a. On average, the RMSE does not change much as `n` gets larger, while the variability of RMSE does decrease.

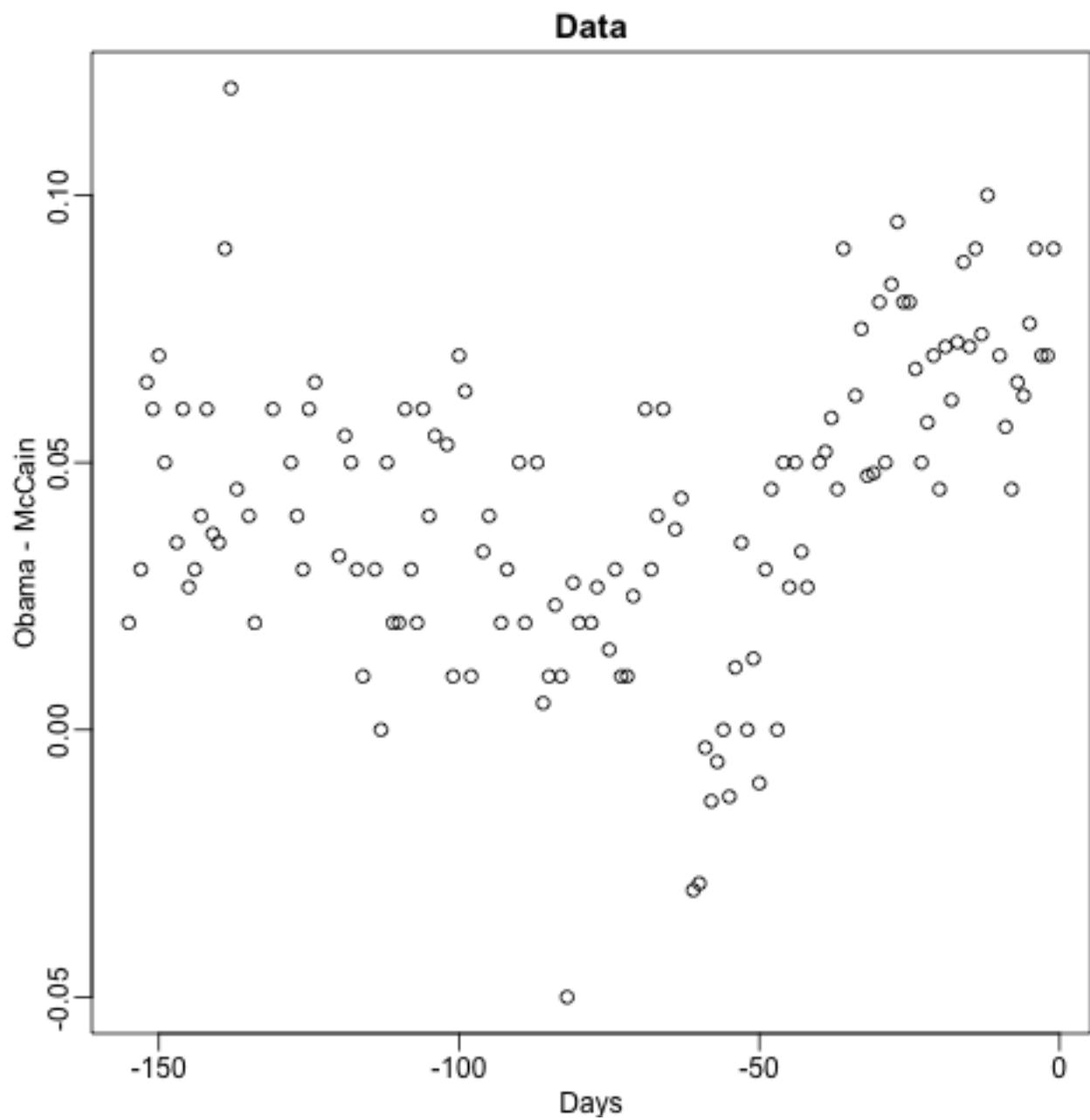


Figure 24:

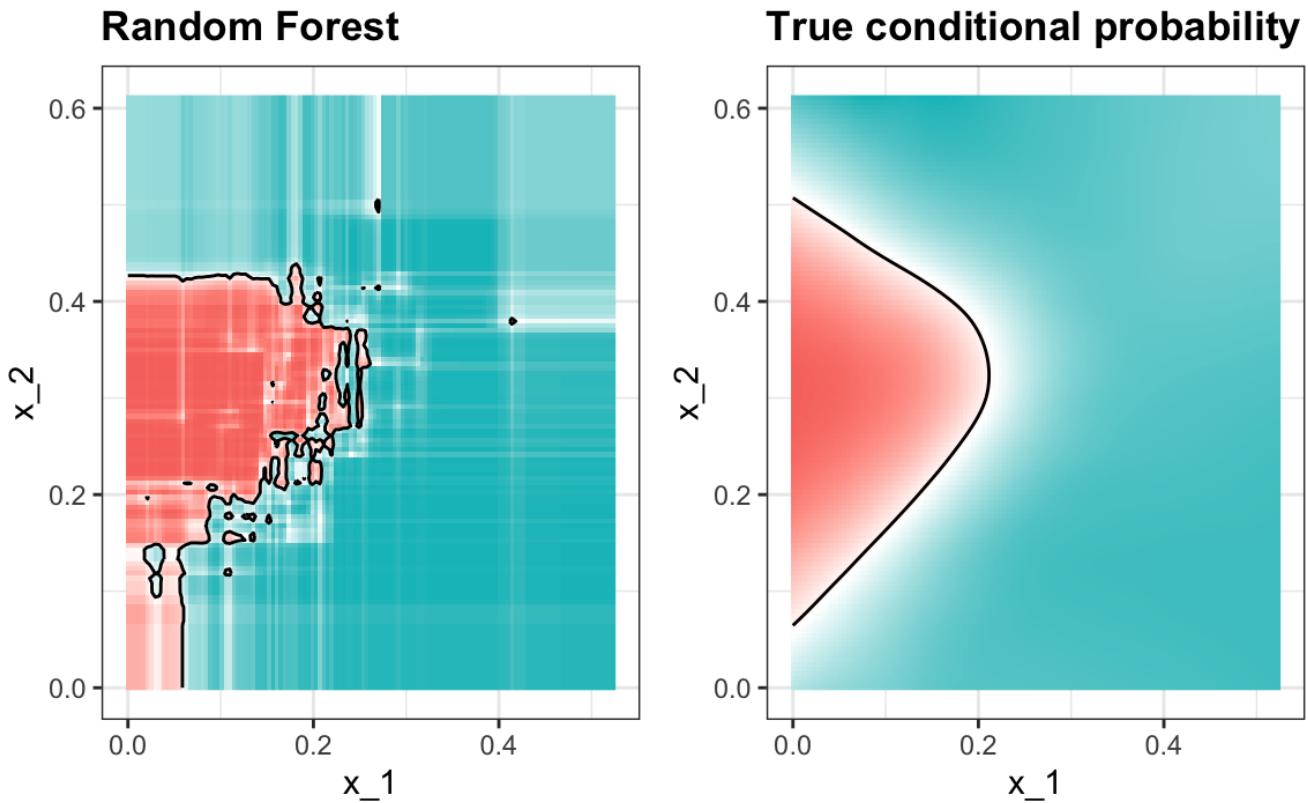


Figure 25:

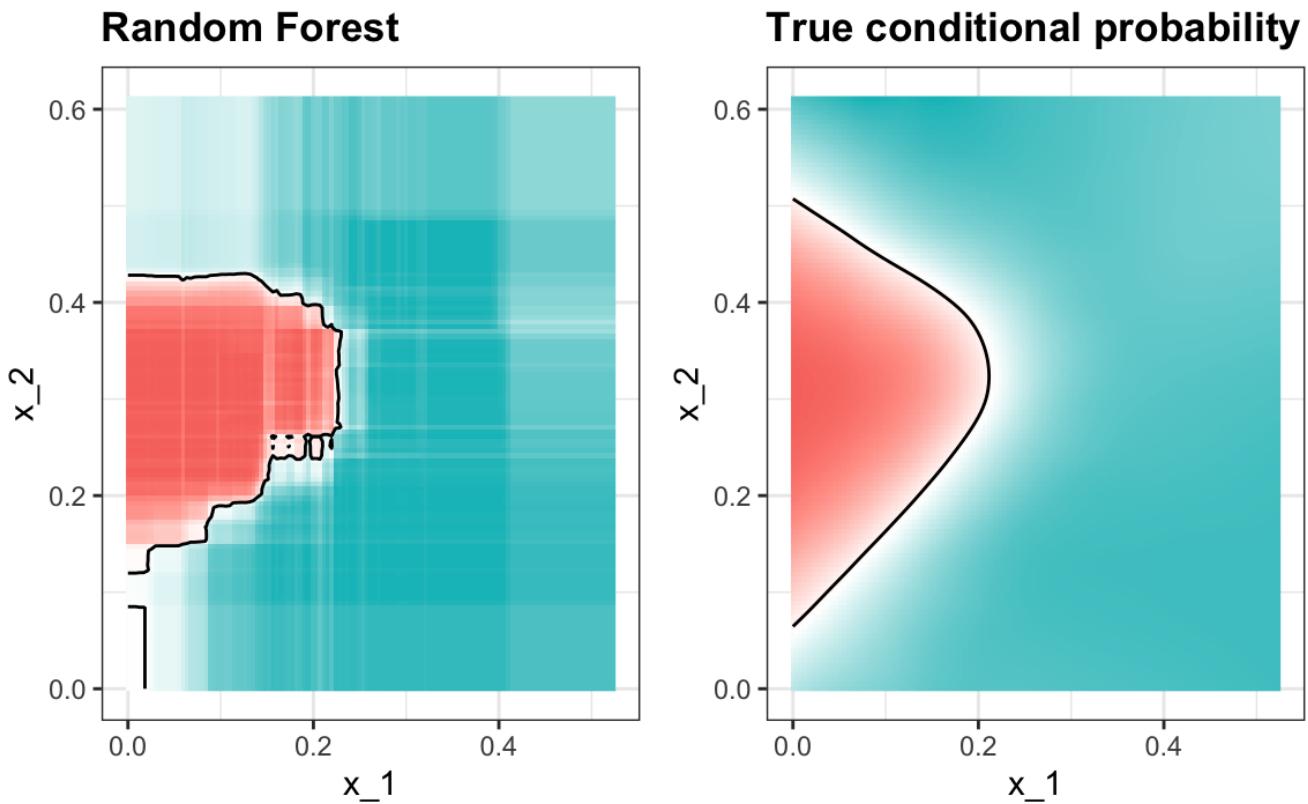


Figure 26:

- b. Because of the law of large numbers, the RMSE decreases: more data, more precise estimates.
c. $n = 10000$ is not sufficiently large. To see a decrease in RMSE, we need to make it larger.
d. The RMSE is not a random variable.
4. Now repeat exercise 1, but this time make the correlation between x and y larger by changing Σ like this:
- ```
n <- 100
Sigma <- 9*matrix(c(1, 0.95, 0.95, 1), 2, 2)
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) |>
 data.frame() |> setNames(c("x", "y"))
```
- Repeat the exercise and note what happens to the RMSE now.
5. Which of the following best explains why the RMSE in exercise 4 is so much lower than exercise 1:
- It is just luck. If we do it again, it will be larger.
  - The Central Limit Theorem tells us the RMSE is normal.
  - When we increase the correlation between  $x$  and  $y$ ,  $x$  has more predictive power and thus provides a better estimate of  $y$ . This correlation has a much bigger effect on RMSE than  $n$ . Large  $n$  simply provide us more precise estimates of the linear model coefficients.
  - These are both examples of regression, so the RMSE has to be the same.

6. Create a dataset using the following code:

```
n <- 1000
Sigma <- matrix(c(1, 3/4, 3/4, 3/4, 1, 0, 3/4, 0, 1), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) |>
 data.frame() |> setNames(c("y", "x_1", "x_2"))
```

Note that  $y$  is correlated with both  $x_1$  and  $x_2$ , but the two predictors are independent of each other.

```
cor(dat)
```

Use the **caret** package to partition into a test and training set of equal size. Compare the RMSE when using just  $x_1$ , just  $x_2$ , and both  $x_1$  and  $x_2$ . Train a linear model and report the RMSE.

7. Repeat exercise 6 but now create an example in which  $x_1$  and  $x_2$  are highly correlated:

```
n <- 1000
Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.95, 0.75, 0.95, 1.0), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) |>
 data.frame() |> setNames(c("y", "x_1", "x_2"))
```

Use the **caret** package to partition into a test and training set of equal size. Compare the RMSE when using just  $x_1$ , just  $x_2$ , and both  $x_1$  and  $x_2$ . Train a linear model and report the RMSE.

8. Compare the results in 6 and 7 and choose the statement you agree with:

- Adding extra predictors can improve RMSE substantially, but not when they are highly correlated with another predictor.
- Adding extra predictors improves predictions equally in both exercises.
- Adding extra predictors results in over fitting.
- Unless we include all predictors, we have no predicting power.

9. Define the following dataset:

```
make_data <- function(n = 1000, p = 0.5,
 mu_0 = 0, mu_1 = 2,
 sigma_0 = 1, sigma_1 = 1){
 y <- rbinom(n, 1, p)
 f_0 <- rnorm(n, mu_0, sigma_0)
 f_1 <- rnorm(n, mu_1, sigma_1)
 x <- ifelse(y == 1, f_1, f_0)
 test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
 list(train = data.frame(x = x, y = as.factor(y)) |>
```

```

 slice(-test_index),
 test = data.frame(x = x, y = as.factor(y)) |>
 slice(test_index)
}

```

Note that we have defined a variable `x` that is predictive of a binary outcome `y`.

```
dat$train |> ggplot(aes(x, color = y)) + geom_density()
```

Compare the accuracy of linear regression and logistic regression.

10. Repeat the simulation from exercise 1 100 times and compare the average accuracy for each method and notice they give practically the same answer.

11. Generate 25 different datasets changing the difference between the two class: `delta <- seq(0, 3, len = 25)`. Plot accuracy versus `delta`.

12. We can see what the data looks like if we add 1s to our 2 or 7 examples using this code:

```
library(dslabs)
mnist_127$train |> ggplot(aes(x_1, x_2, color = y)) + geom_point()
```

Fit QDA using the `qda` function in the **MASS** package to create a confusion matrix for predictions on the test. Which of the following best describes the confusion matrix:

- a. It is a two-by-two table. b. Because we have three classes, it is a two-by-three table. c. Because we have three classes, it is a three-by-three table. d. Confusion matrices only make sense when the outcomes are binary.

13. The `byClass` component returned by the `confusionMatrix` object provides sensitivity and specificity for each class. Because these terms only make sense when data is binary, each row represents sensitivity and specificity when a particular class is 1 (positives) and the other two are considered 0s (negatives). Based on the values returned by `confusionMatrix`, which of the following is the most common mistake:

- a. Calling 1s either a 2 or 7. b. Calling 2s either a 1 or 7. c. Calling 7s either a 1 or 2. d. All mistakes are equally common.

14. Create a grid of `x_1` and `x_2` using:

```
GS <- 150
new_x <- with(mnist_127$train,
 expand.grid(x_1 = seq(min(x_1), max(x_1), len = GS),
 x_2 = seq(min(x_2), max(x_2), len = GS)))
```

then visualize the decision rule by coloring the regions of the Cartesian plane to represent the label that would be called in that region.

14. Repeat exercise 13 but for LDA. Which of the following explains why LDA has worse accuracy:

- a. LDA separates the space with lines making it too rigid.
- b. LDA divides the space into two and there are three classes.
- c. LDA is very similar to QDA the difference is due to chance.
- d. LDA can't be used with more than one class.

15. Now repeat exercise 13 for kNN with  $\backslash(k=31\backslash)$  and compute and compare the overall accuracy for all three methods.

16. To understand how a simple method like kNN can outperform a model that explicitly tries to emulate Bayes' rule, explore the conditional distributions of `x_1` and `x_2` to see if the normal approximation holds. Generative models can be very powerful, but only when we are able to successfully approximate the joint distribution of predictors conditioned on each class.

17. Earlier we used logistic regression to predict sex from height. Use kNN to do the same. Use the code described in this chapter to select the  $\backslash(F\_1\backslash)$  measure and plot it against  $\backslash(k\backslash)$ . Compare to the  $\backslash(F\_1\backslash)$  of about 0.6 we obtained with regression.

18. Create a simple dataset where the outcome grows 0.75 units on average for every increase in a predictor:

```

n <- 1000
sigma <- 0.25
x <- rnorm(n, 0, 1)
y <- 0.75 * x + rnorm(n, 0, sigma)
dat <- data.frame(x = x, y = y)

```

Use `rpart` to fit a regression tree and save the result to `fit`.

19. Plot the final tree so that you can see where the partitions occurred.
20. Make a scatterplot of `y` versus `x` along with the predicted values based on the fit.
21. Now model with a random forest instead of a regression tree using `randomForest` from the `randomForest` package, and remake the scatterplot with the prediction line.
22. Use the function `plot` to see if the random forest has converged or if we need more trees.
23. It seems that the default values for the random forest result in an estimate that is too flexible (not smooth). Re-run the random forest but this time with `nodesize` set at 50 and `maxnodes` set at 25. Remake the plot.
24. This `dslabs*` dataset includes the `tissue_gene_expression` with a matrix `x`:

```

library(dslabs)
dim(tissue_gene_expression$x)

```

with the gene expression measured on 500 genes for 189 biological samples representing seven different tissues. The tissue type is stored in `tissue_gene_expression$y`.

```
table(tissue_gene_expression$y)
```

Fit a random forest using the `randomForest` function in the package `randomForest`. Then use the `varImp` function to see which are the top 10 most predictive genes. Make a histogram of the reported importance to get an idea of the distribution of the importance values.

1. <https://web.stanford.edu/~hastie/Papers/ESLII.pdf>
2. [https://papers.ssrn.com/sol3/Delivery.cfm/SSRN\\_ID1759289\\_code1486039.pdf?abstractid=1759289&mirid=1&type=2](https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID1759289_code1486039.pdf?abstractid=1759289&mirid=1&type=2)
3. <https://web.stanford.edu/~hastie/Papers/ESLII.pdf>

# Introduction to Data Science - 31 Machine learning in practice

Rafael A. Irizarry

Now that we have learned several methods and explored them with simple examples, we will try them out on a real example: the MNIST digits.

We can load this data using the following **dslabs** package:

```
library(dslabs)
mnist <- read_mnist()
```

The dataset includes two components, a training set and a test set:

```
names(mnist)
#> [1] "train" "test"
```

Each of these components includes a matrix with features in the columns:

```
dim(mnist$train$images)
#> [1] 60000 784
```

and vector with the classes as integers:

```
class(mnist$train$labels)
#> [1] "integer"
table(mnist$train$labels)
#>
#> 0 1 2 3 4 5 6 7 8 9
#> 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949
```

Because we want this example to run on a small laptop and in less than one hour, we will consider a subset of the dataset. We will sample 10,000 random rows from the training set and 1,000 random rows from the test set:

```
set.seed(1990)
index <- sample(nrow(mnist$train$images), 10000)
x <- mnist$train$images[index,]
y <- factor(mnist$train$labels[index])
index <- sample(nrow(mnist$test$images), 1000)
x_test <- mnist$test$images[index,]
y_test <- factor(mnist$test$labels[index])
```

When fitting models to large datasets, we recommend using matrices instead of data frames, as matrix operations tend to be faster. In the **caret** package, predictor matrices must have column names to track features accurately during prediction on the test set. If the matrices lack column names, you can assign names based on their position:

```
colnames(x) <- 1:ncol(mnist$train$images)
colnames(x_test) <- colnames(x)
```

## 31.1 The caret package

We have already learned about several machine learning algorithms. Many of these algorithms are implemented in R. However, they are distributed via different packages, developed by different authors, and often use different syntax. The **caret** package tries to consolidate these differences and provide consistency. It currently includes over

200 different methods which are summarized in the **caret** package manual<sup>1</sup>. Keep in mind that **caret** does not include the packages needed to run each possible algorithm. To apply a machine learning method through **caret** you still need to install the library that implement the method. The required packages for each method are described in the package manual.

The **caret** package also provides a function that performs cross validation for us. Here we provide some examples showing how we use this helpful package. We will first use the 2 or 7 example to illustrate and, in later sections, we use the package to run algorithms on the larger MNIST dataset.

### 31.1.1 The **train** function

The R functions that fit machine algorithms are all slightly different. Functions such as **lm**, **glm**, **qda**, **lda**, **knn3**, **rpart** and **randomForrest** use different syntax, have different argument names and produce objects of different types.

The **caret train** function lets us train different algorithms using similar syntax. So, for example, we can type the following to train three different models:

```
library(caret)
train_glm <- train(y ~ ., method = "glm", data = mnist_27$train)
train_qda <- train(y ~ ., method = "qda", data = mnist_27$train)
train_knn <- train(y ~ ., method = "knn", data = mnist_27$train)
```

As we explain in more detail in Section 31.1.3, the **train** function selects parameters for you using a resampling method to estimate the MSE, with bootstrap as the default.

### 31.1.2 The **predict** function

The **predict** function is very useful for machine learning applications. This function takes an object from a fitting function and a data frame with features  $\{\mathbf{x}\}$  for which to predict, and returns predictions for these features.

Here is an example with logistic regression:

```
fit <- glm(y ~ ., data = mnist_27$train, family = "binomial")
p_hat <- predict(fit, newdata = mnist_27$test)
```

In this case, the function is simply computing

$$\hat{p}(\mathbf{x}) = g^{-1}(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2)$$

with  $g(p) = \log\frac{p}{1-p} \implies g^{-1}(p) = \frac{1}{1+e^{-\mu}}$

for the  $x_1$  and  $x_2$  in the test set **mnist\_27\$test**. With these estimates in place, we can make our predictions and compute our accuracy:

```
y_hat <- factor(ifelse(p_hat > 0.5, 7, 2))
```

However, note that **predict** does not always return objects of the same type; it depends on what type of object it is applied to. To learn about the specifics, you need to look at the help file specific for the type of fit object that is being used.

**predict** is actually a special type of function in R called a *generic function*. Generic functions call other functions depending on what kind of object it receives. So if **predict** receives an object coming out of the **lm** function, it will call **predict.lm**. If it receives an object coming out of **glm**, it calls **predict.glm**. If the fit is from **knn3**, it calls **predict.knn3**, and so on. These functions are similar but not exactly. You can learn more about the differences by reading the help files:

```
?predict.glm
?predict.qda
?predict.knn3
```

There are many other versions of **predict** and many machine learning algorithms define their own **predict** function.

As with **train**, the **caret** packages unifies the use of **predict** with the function **predict.train**. This function takes the output of **train** and produces prediction of categories or estimates of  $\{p(\mathbf{x})\}$ .

The code looks the same for all methods:

```
y_hat_glm <- predict(train_glm, mnist_27$test, type = "raw")
y_hat_qda <- predict(train_qda, mnist_27$test, type = "raw")
y_hat_knn <- predict(train_knn, mnist_27$test, type = "raw")
```

This permits us to quickly compare the algorithms. For example, we can compare the accuracy like this:

```
fits <- list(glm = y_hat_glm, qda = y_hat_qda, knn = y_hat_knn)
sapply(fits, function(fit) confusionMatrix(fit, mnist_27$test$y)$overall[["Accuracy"]])
#> glm qda knn
#> 0.775 0.815 0.835
```

### 31.1.3 Resampling

When an algorithm includes a tuning parameter, `train` automatically uses a resampling method to estimate MSE and decide among a few default candidate values. To find out what parameter or parameters are optimized, you can read the `caret` manual<sup>2</sup> or study the output of:

```
modelLookup("knn")
```

To obtain all the details of how `caret` implements kNN you can use:

```
getModelInfo("knn")
```

If we run it with default values:

```
train_knn <- train(y ~ ., method = "knn", data = mnist_27$train)
```

you can quickly see the results of the cross validation using the `ggplot` function. The argument `highlight` highlights the max:

```
ggplot(train_knn, highlight = TRUE)
```

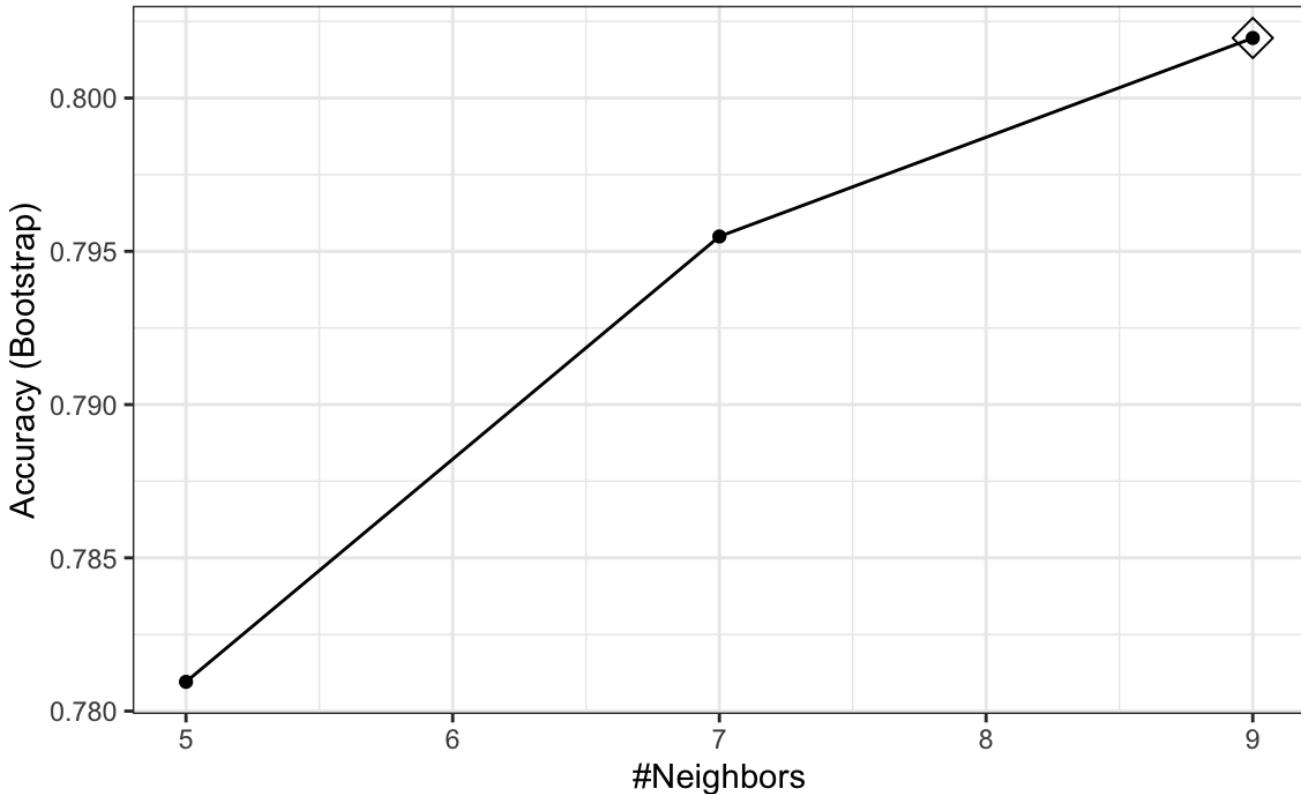


Figure 1:

By default, the resampling is performed by taking 25 bootstrap samples, each comprised of 25% of the observations. For the kNN method, the default is to try  $\{k=5,7,9\}$ . We change this using the `tuneGrid` argument. The grid of values must be supplied by a data frame with the parameter names as specified in the `modelLookup` output.

Here we present an example where we try out 38 values between 1 and 75. To do this with `caret`, we need to define a column named `k`, so we use this: `data.frame(k = seq(1, 75, 2))`. Note that when running this code, we are fitting 38 versions of kNN to 25 bootstrapped samples. Since we are fitting  $\{38 \times 25 = 950\}$  kNN models, running this code will take several seconds.

```
train_knn <- train(y ~ ., method = "knn",
 data = mnist_27$train,
 tuneGrid = data.frame(k = seq(1, 75, 2)))
ggplot(train_knn, highlight = TRUE)
```

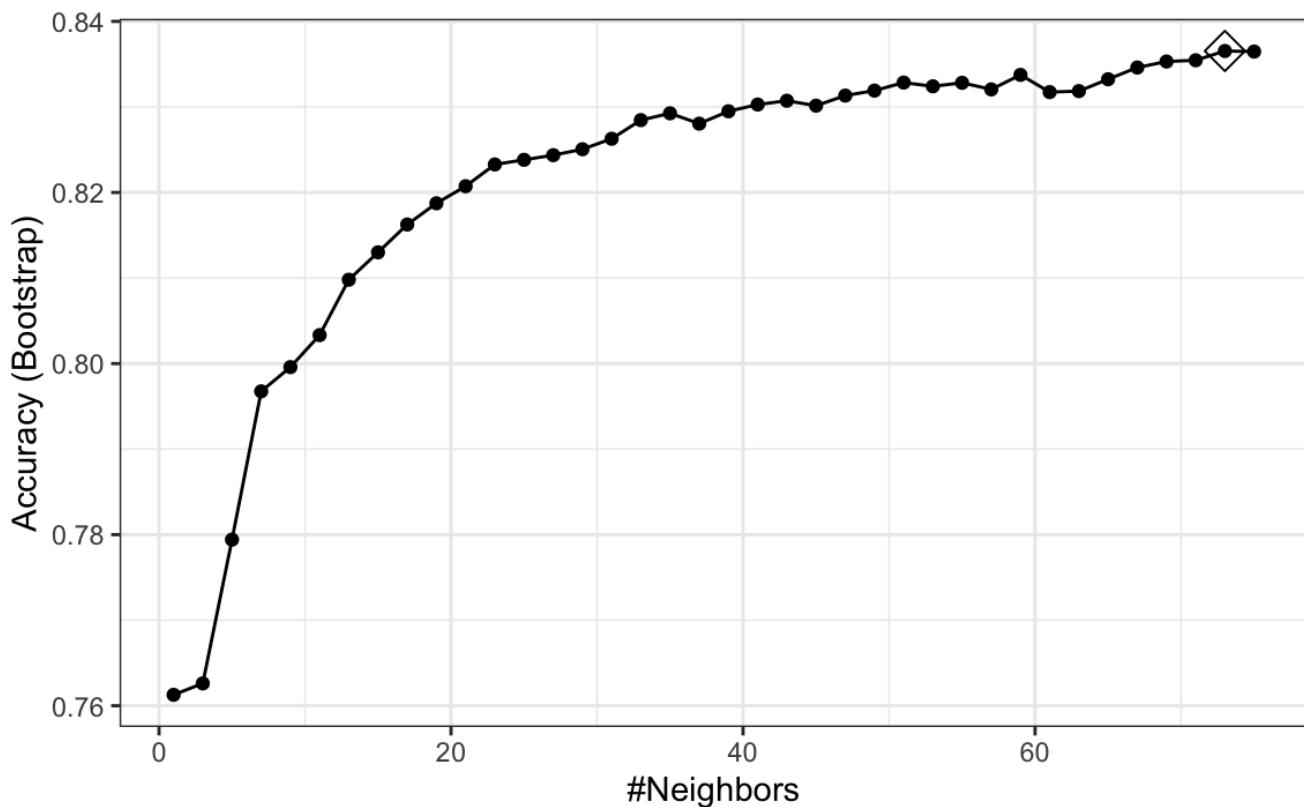


Figure 2:

Because resampling methods are random procedures, the same code can result in different results. To assure reproducible results you should set the seed, as we did at the start of this chapter.

To access the parameter that maximized the accuracy, you can use this:

```
train_knn$bestTune
#> k
#> 37 73
```

and the best performing model like this:

```
train_knn$finalModel
#> 73-nearest neighbor model
#> Training set outcome distribution:
#>
#> 2 7
#> 401 399
```

The function `predict` will use this best performing model. Here is the accuracy of the best model when applied to the test set, which we have not yet used because the cross validation was done on the training set:

```
confusionMatrix(predict(train_knn, mnist_27$test, type = "raw"),
 mnist_27$test$y)$overall["Accuracy"]
#> Accuracy
#> 0.825
```

Bootstrapping is not always the best approach to resampling (see @knn-in-practice, for an example). If we want to change our resampling method, we can use the `trainControl` function. For example, the code below runs 10-fold cross validation. This means we have 10 samples using 90% of the observations to train in each sample. We accomplish this using the following code:

```
control <- trainControl(method = "cv", number = 10, p = .9)
train_knn_cv <- train(y ~ ., method = "knn",
 data = mnist_27$train,
 tuneGrid = data.frame(k = seq(1, 71, 2)),
 trControl = control)
```

The `results` component of the `train` output includes several summary statistics related to the variability of the cross validation estimates:

```
names(train_knn$results)
#> [1] "k" "Accuracy" "Kappa" "AccuracySD" "KappaSD"
```

You can learn many more details about the `caret` package, from the manual<sup>3</sup>.

## 31.2 Preprocessing

We often transform predictors before running the machine algorithm. We also remove predictors that are clearly not useful. We call these steps *preprocessing*.

Examples of preprocessing include standardizing the predictors, taking the log transform of some predictors, removing predictors that are highly correlated with others, and removing predictors with very few non-unique values or close to zero variation.

For example, we can run the `nearZero` function from the `caret` package to see that several features do not vary much from observation to observation. We can see that there is a large number of features with close to 0 variability:

```
library(matrixStats)
sds <- colSds(x)
hist(sds, breaks = 256)
```

This is expected because there are parts of the image that rarely contain writing (dark pixels).

The `caret` packages includes a function that recommends features to be removed due to *near zero variance*:

```
nzv <- nearZeroVar(x)
```

We can see the columns recommended for removal are the near the edges:

```
image(matrix(1:784 %in% nzv, 28, 28))
rafalib::mpar()
image(matrix(1:784 %in% nzv, 28, 28))
```

So we end up removing

```
length(nzv)
#> [1] 532
```

predictors.

The `caret` package features the `preProcess` function, which allows users to establish a predefined set of preprocessing operations based on a training set. This function is designed to apply these operations to new datasets without

## Histogram of sds

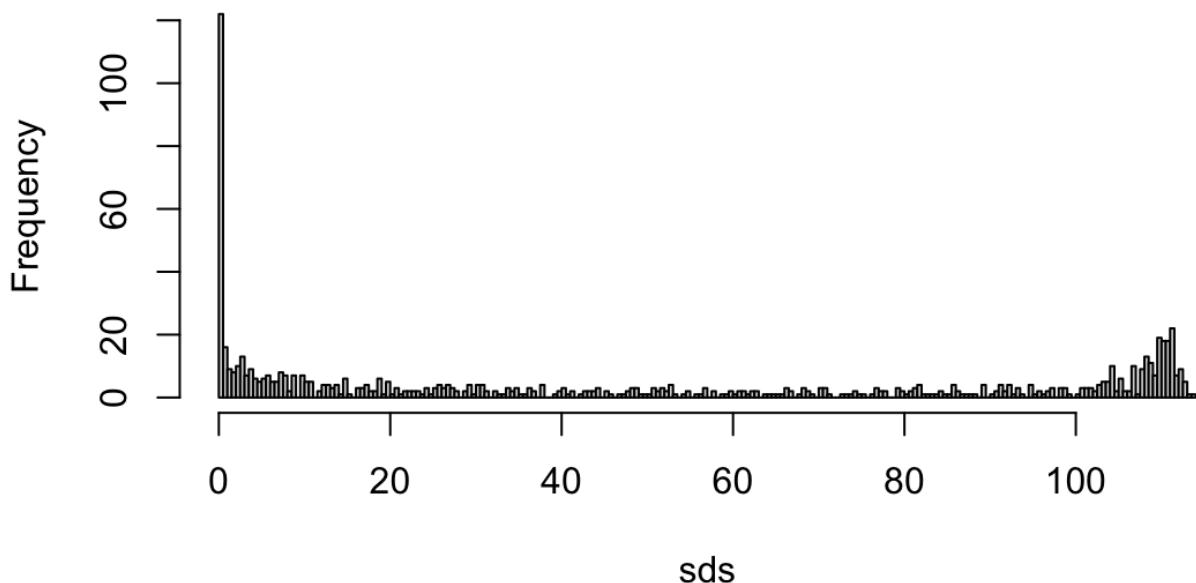


Figure 3:

recalculating anything on the test set, ensuring that all preprocessing steps are consistent and derived solely from the training data.

Below is an example demonstrating how to remove predictors with near-zero variance and then center the remaining predictors:

```
pp <- preProcess(x, method = c("nzv", "center"))
centered_subsetted_x_test <- predict(pp, newdata = x_test)
dim(centered_subsetted_x_test)
#> [1] 1000 252
```

Additionally, the `train` function in caret includes a `preProcess` argument that allows users to specify which preprocessing steps to apply automatically during model training. We will explore this functionality further in the context of a k-Nearest Neighbors model @knn-in-practice.

### 31.3 Parallelization

During cross-validation or bootstrapping, the process of fitting models to different samples or using varying parameters can be performed independently. Imagine you are fitting 100 models; if you had access to 100 computers, you could theoretically speed up the process by a factor of 100 by fitting each model on a separate computer and then aggregating the results. In reality, most modern computers, including many personal computers, are equipped with multiple processors that allow for such parallel execution. This method, known as parallelization, leverages these multiple processors to conduct several computational tasks simultaneously, significantly accelerating the model training process. By distributing the workload across different processors, parallelization makes it feasible to manage large datasets and complex modeling procedures efficiently.

The `caret` package is set up to run in parallel but you have to let R know that you want to parallelize your work. To do this we can use the `doParallel` package:

```
library(doParallel)
```

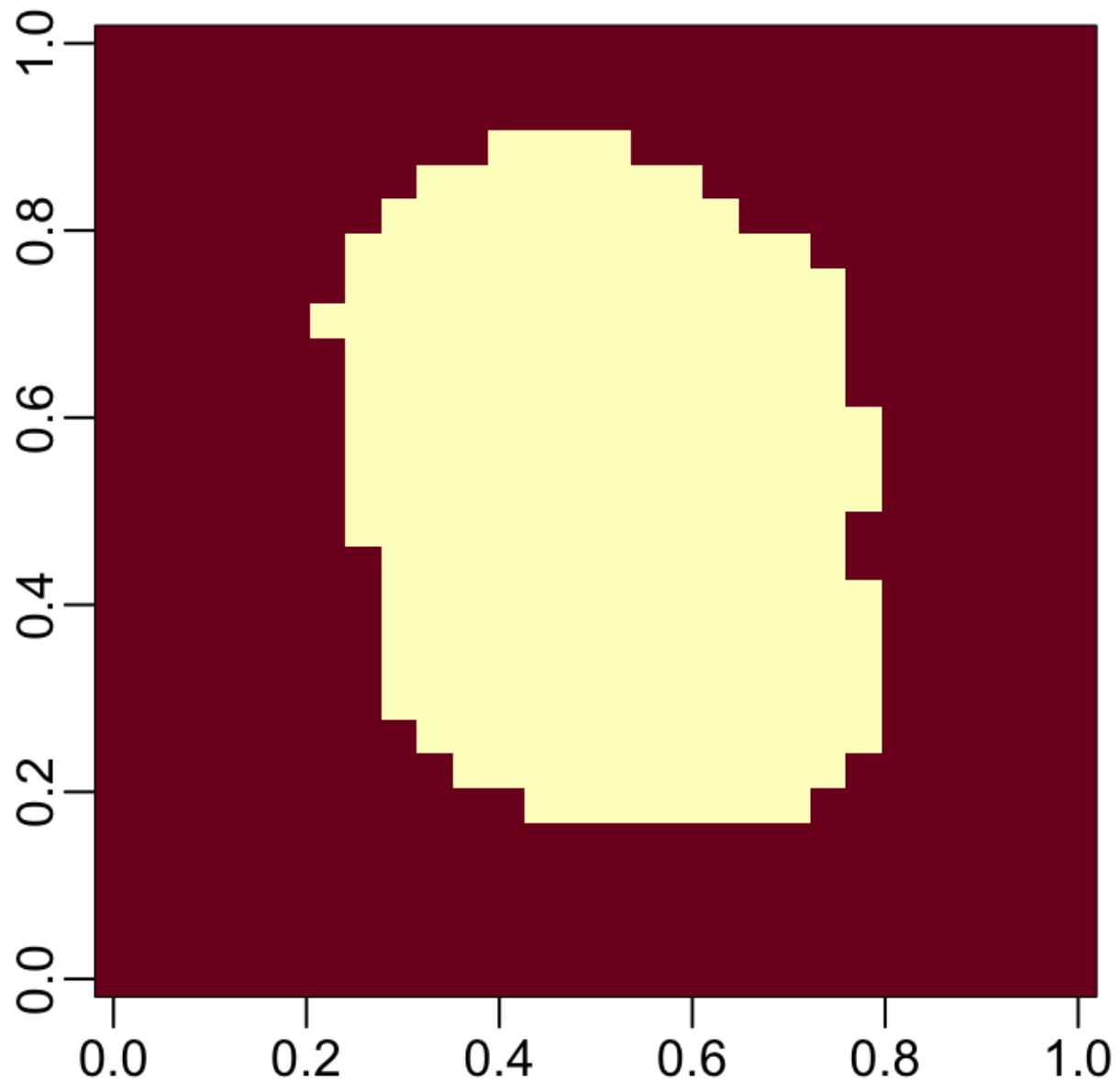


Figure 4:

```

#> Loading required package: foreach
#> Loading required package: iterators
#> Loading required package: parallel
nc <- detectCores() - 1
cl <- makeCluster(nc) # it is convention to leave 1 core for OS
registerDoParallel(cl)

```

If you do use parallelization, make sure to let R know you are done with the following lines of code:

```

stopCluster(cl)
stopImplicitCluster()

```

When parallelizing tasks across multiple processors, it's important to consider the risk of running out of memory. Each processor might require a copy of the data or substantial portions of it, which can multiply overall memory demands. This is especially challenging if the data or models are large.

## 31.4 k-nearest neighbors

Before starting this section, note that the first two calls to the `train` function in the code below can take several hours to run. This is a common challenge when training machine learning algorithms since we have to run the algorithm for each cross validation split and each set of tuning parameters being considered. In the next section, we will provide some suggestions on how to predict the duration of the process and ways to reduce.

As we will see soon, the optimal  $\backslash(k\backslash)$  for the MNIST data is between 1 and 7. For small values of  $\backslash(k\backslash)$ , bootstrapping can be problematic for estimating MSE for a k-Nearest Neighbors (kNN). This is because bootstrapping involves sampling with replacement from the original dataset which implies the closest neighbor will often appear twice but considered two independent observations by kNN. This is a unrealistic scenario that can distort the estimate MSE when, for example,  $\backslash(k=3\backslash)$ . We therefore us cross-validation to estimate our MSE.

The first step is to optimize for  $\backslash(k\backslash)$ .

```

train_knn <- train(x, y, method = "knn",
 preProcess = "nzv",
 trControl = trainControl("cv", number = 20, p = 0.95),
 tuneGrid = data.frame(k = seq(1, 7, 2)))

```

Once we optimize our algorithm, the `predict` function defaults to using the best performing algorithm fit with the entire training data:

```
y_hat_knn <- predict(train_knn, x_test, type = "raw")
```

We achieve relatively high accuracy:

```

confusionMatrix(y_hat_knn, factor(y_test))$overall["Accuracy"]
#> Accuracy
#> 0.953

```

### 31.4.1 Dimension reduction with PCA

An alternative to removing low variance columns directly is to use dimension reduction on the feature matrix before applying the algorithms.

```
pca <- prcomp(x)
```

We can actually explain, say, 75% of the variability in the predictors with a small number of dimensions:

```
p <- which(cumsum(pca$sdev^2)/sum(pca$sdev^2) >= 0.75)[1]
```

We can then re-run our algorithm on these 33 features:

```
fit_knn_pca <- knn3(pca$x[, 1:p], y, k = train_knn$bestTune)
```

When predicting, it is important that we not use the test set when finding the PCs nor any summary of the data, as this could result in overtraining. We therefrom compute the averages needed for centering and the rotation on the training set:

```
newdata <- sweep(x_test, 2, colMeans(x)) %*% pca$rotation[,1:p]
y_hat_knn_pca <- predict(fit_knn_pca, newdata, type = "class")
```

We obtain similar accuracy, while using only 33 dimensions:

```
confusionMatrix(y_hat_knn_pca, factor(y_test))$overall["Accuracy"]
#> Accuracy
#> 0.959
```

In this example, we used the  $\backslash(k\backslash)$  optimized for the raw data, not the principal components. Note that to obtain an unbiased MSE estimate we have to recompute the PCA for each cross-validation sample and apply to the validation set. Because the `train` function includes PCA as one of the available preprocessing operations we can achieve this with this modification of the code above:

```
train_knn_pca <- train(x, y, method = "knn",
 preProcess = c("nzv", "pca"),
 trControl = trainControl("cv", number = 20, p = 0.95,
 preProcOptions = list(pcaComp = p)),
 tuneGrid = data.frame(k = seq(1, 7, 2)))
y_hat_knn_pca <- predict(train_knn_pca, x_test, type = "raw")
confusionMatrix(y_hat_knn_pca, factor(y_test))$overall["Accuracy"]
#> Accuracy
#> 0.964
```

A limitation of this approach is that we don't get to optimize the number of PCs used in the analysis. To do this we need to write our own method. The `caret` manual<sup>4</sup> describe how to do this.

## 31.5 Random Forest

With the random forest algorithm several parameters can be optimized, but the main one is `mtry`, the number of predictors that are randomly selected for each tree. This is also the only tuning parameter that the `caret` function `train` permits when using the default implementation from the `randomForest` package.

```
library(randomForest)
train_rf <- train(x, y, method = "rf",
 preProcess = "nzv",
 tuneGrid = data.frame(mtry = seq(5, 15)))
y_hat_rf <- predict(train_rf, x_test, type = "raw")
```

Now that we have optimized our algorithm, we are ready to fit our final model:

```
y_hat_rf <- predict(train_rf, x_test, type = "raw")
```

As with kNN, we also achieve high accuracy:

```
confusionMatrix(y_hat_rf, y_test)$overall["Accuracy"]
#> Accuracy
#> 0.952
```

By optimizing some of the other algorithm parameters, we can achieve even higher accuracy.

## 31.6 Testing and improving computation time

The default method for estimating accuracy used by the `train` function is to test prediction on 25 bootstrap samples. This can result in long compute times. For example, if we are considering several values, say 10, of the tuning parameters, we will fit the algorithm 250 times. We can use the `system.time` function to estimate how long it takes to run the algorithm once:

```

nzv <- nearZeroVar(x)
system.time({fit_rf <- randomForest(x[, -nzv], y, mtry = 9)})
#> user system elapsed
#> 61.656 0.053 62.368

```

and use this to estimate the total time for the 250 iterations. In this case it will be several hours.

One way to reduce run time is to use k-fold cross validation with a smaller number of test sets. A popular choice is leaving out 5 test sets with 20% of the data. To use this we set the `trControl` argument in `train` to `trainControl(method = "cv", number = 5, p = .8)`.

For random forest, we can also speed up the training step by running less trees per fit. After running the algorithm once, we can use the `plot` function to see how the error rate changes as the number of trees grows.

Here we can see that error rate stabilizes after about 200 trees:

```
plot(fit_rf)
```

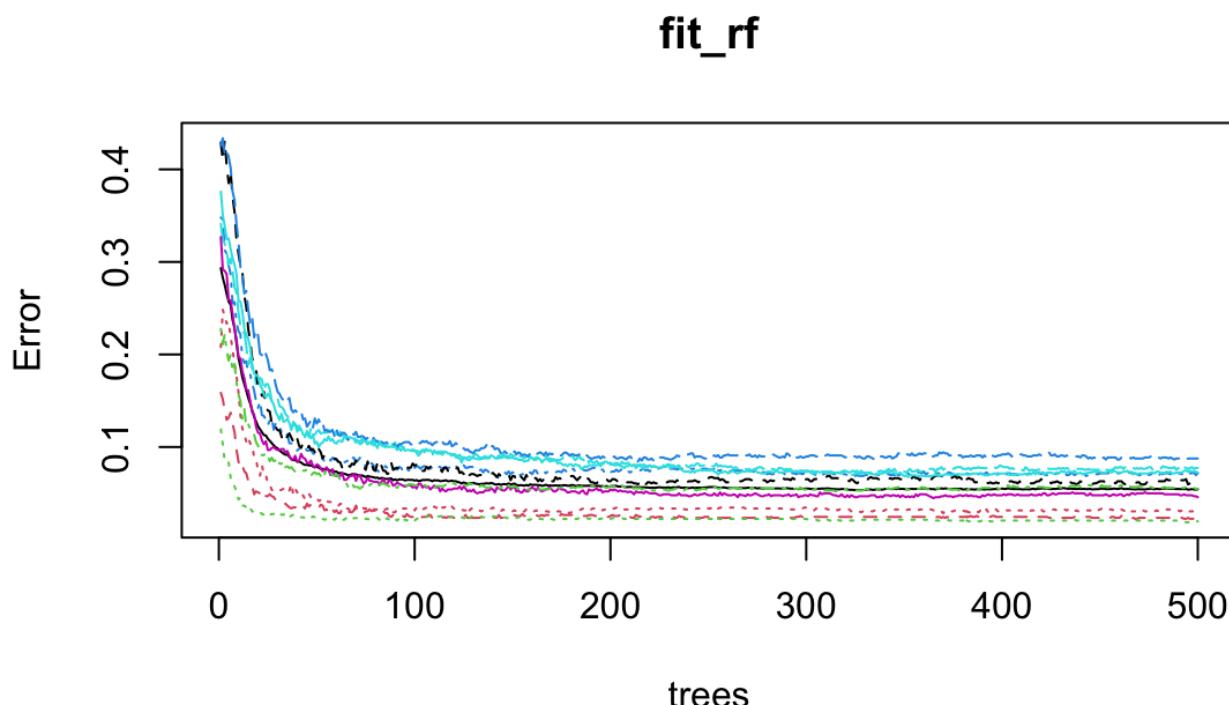


Figure 5:

We can use this finding to speed up the cross validation procedure. Specifically, because the default is 500, by adding the argument `ntree = 200` to the call to `train` above, the procedure will finish 2.5 times faster.

### 31.7 Variable importance

The following function computes the importance of each feature:

```
imp <- importance(fit_rf)
```

We can see which features are being used most by plotting an image:

```

mat <- rep(0, ncol(x))
mat[-nzv] <- imp
image(matrix(mat, 28, 28))

```

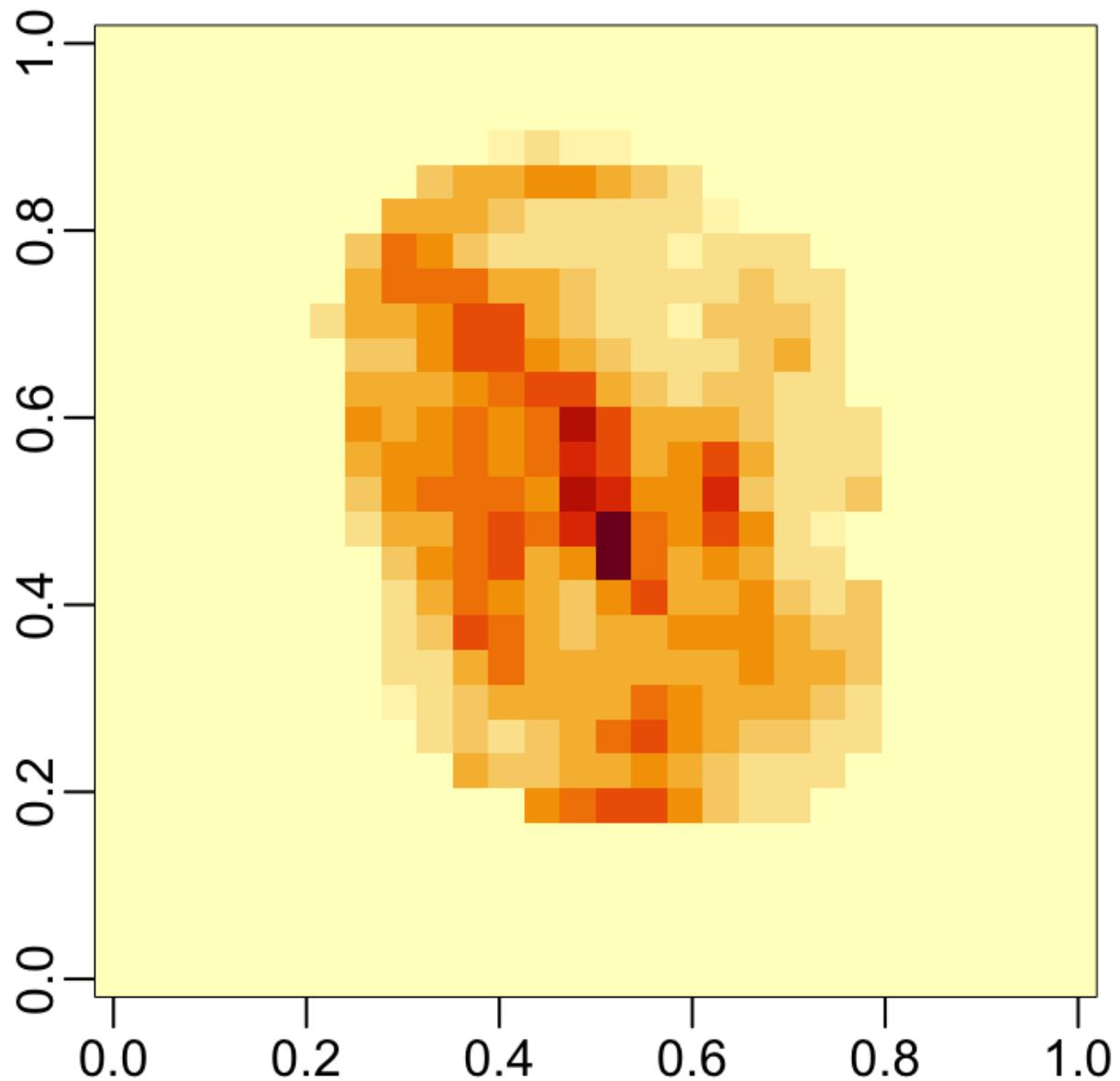


Figure 6:

## 31.8 Diagnostics

An important part of data analysis is visualizing results to determine why we are failing. How we do this depends on the application. Below we show the images of digits for which we made an incorrect prediction. Here are some errors for the random forest:

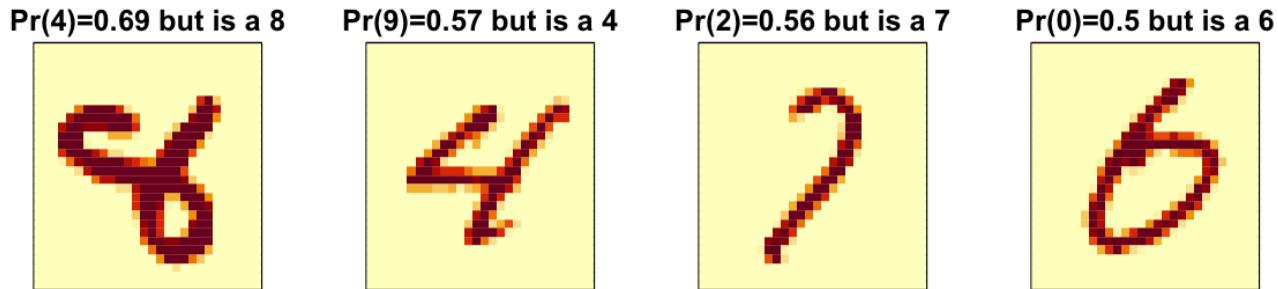


Figure 7:

By examining errors like this, we often find specific weaknesses to algorithms or parameter choices and can try to correct them.

## 31.9 Ensembles

The idea of an ensemble is similar to the idea of combining data from different pollsters to obtain a better estimate of the true support for each candidate.

In machine learning, one can usually greatly improve the final results by combining the results of different algorithms.

Here is a simple example where we compute new class probabilities by taking the average of random forest and kNN. We can see that the accuracy improves:

```
p_rf <- predict(fit_rf, x_test[,-nzv], type = "prob")
p_rf <- p_rf / rowSums(p_rf)
p_knn_pca <- predict(train_knn_pca, x_test, type = "prob")
p <- (p_rf + p_knn_pca)/2
y_pred <- factor(apply(p, 1, which.max) - 1)
confusionMatrix(y_pred, y_test)$overall["Accuracy"]
#> Accuracy
#> 0.971
```

We have just built an ensemble with just two algorithms. By combining more similarly performing, but uncorrelated, algorithms we can improve accuracy further.

## 31.10 Exercises

1. In the exercises in Chapter 30 we saw that changing `maxnodes` or `nodesize` in the `randomForest` function improved our estimate. Let's use the `train` function to help us pick these values. From the `caret` manual we see that we can't tune the `maxnodes` parameter or the `nodesize` argument with `randomForest`, so we will use the `Rborist` package and tune the `minNode` argument. Use the `train` function to try values `minNode <- seq(5, 250, 25)`. See which value minimizes the estimated RMSE.

2. This `**dslabs*` dataset includes a matrix `x`:

```
library(dslabs)
dim(tissue_gene_expression$x)
```

with the gene expression measured on 500 genes for 189 biological samples representing seven different tissues. The tissue type is stored in `y`:

```
table(tissue_gene_expression$y)
```

Split the data in training and test sets, then use kNN to predict tissue type and see what accuracy you obtain. Try it for  $\backslash(k = 1, 3, \dots, 11\backslash)$ .

3. We are going to apply LDA and QDA to the `tissue_gene_expression` dataset. We will start with simple examples based on this dataset and then develop a realistic example.

Create a dataset with just the classes `cerebellum` and `hippocampus` (two parts of the brain) and a predictor matrix with 10 randomly selected columns. Estimate the accuracy of LDA.

```
set.seed(1993)
tissues <- c("cerebellum", "hippocampus")
ind <- which(tissue_gene_expression$y %in% tissues)
y <- droplevels(tissue_gene_expression$y[ind])
x <- tissue_gene_expression$x[ind,]
x <- x[, sample(ncol(x), 10)]
```

4. In this case, LDA fits two 10-dimensional normal distributions. Look at the fitted model by looking at the `finalModel` component of the result of `train`. Notice there is a component called `means` that includes the estimate `means` of both distributions. Plot the mean vectors against each other and determine which predictors (genes) appear to be driving the algorithm.

5. Repeat exercises 3 with QDA. Does it have a higher accuracy than LDA?

6. Are the same predictors (genes) driving the algorithm? Make a plot as in exercise 3.

7. One thing we see in the previous plot is that the value of predictors correlate in both groups: some predictors are low in both groups while others are high in both groups. The mean value of each predictor, `colMeans(x)`, is not informative or useful for prediction and often, for interpretation purposes, it is useful to center or scale each column. This can be achieved with the `preProcessing` argument in `train`. Re-run LDA with `preProcessing = "scale"`. Note that accuracy does not change but see how it is easier to identify the predictors that differ more between groups in the plot made in exercise 4.

8. In the previous exercises, we saw that both approaches worked well. Plot the predictor values for the two genes with the largest differences between the two groups in a scatterplot to see how they appear to follow a bivariate distribution as assumed by the LDA and QDA approaches. Color the points by the outcome.

9. Now we are going to increase the complexity of the challenge slightly: we will consider all the tissue types.

```
set.seed(1993)
y <- tissue_gene_expression$y
x <- tissue_gene_expression$x
x <- x[, sample(ncol(x), 10)]
```

What accuracy do you get with LDA?

10. We see that the results are slightly worse. Use the `confusionMatrix` function to learn what type of errors we are making.

11. Plot an image of the centers of the seven 10-dimensional normal distributions.

12. Make a scatterplot along with the prediction from the best fitted model.

13. Use the `rpart` function to fit a classification tree to the `tissue_gene_expression` dataset. Use the `train` function to estimate the accuracy. Try out `cp` values of `seq(0, 0.05, 0.01)`. Plot the accuracy to report the results of the best model.

14. Study the confusion matrix for the best fitting classification tree. What do you observe happening for placenta?

15. Notice that placentas are called endometrium more often than placenta. Note also that the number of placentas is just six, and that, by default, `rpart` requires 20 observations before splitting a node. Thus it is not possible with these parameters to have a node in which placentas are the majority. Rerun the above analysis, but this time permit `rpart` to split any node by using the argument `control = rpart.control(minsplit = 0)`. Does the accuracy increase? Look at the confusion matrix again.

16. Plot the tree from the best fitting model obtained in exercise 11.
17. We can see that with just six genes, we are able to predict the tissue type. Now let's see if we can do even better with a random forest. Use the `train` function and the `rf` method to train a random forest. Try out values of `mtry` ranging from, at least, `seq(50, 200, 25)`. What `mtry` value maximizes accuracy? To permit small `nodesize` to grow as we did with the classification trees, use the following argument: `nodesize = 1`. This will take several seconds to run. If you want to test it out, try using smaller values with `ntree`. Set the seed to 1990.
18. Use the function `varImp` on the output of `train` and save it to an object called `imp`.
19. The `rpart` model we ran above produced a tree that used just six predictors. Extracting the predictor names is not straightforward, but can be done. If the output of the call to train was `fit_rpart`, we can extract the names like this:

```
ind <- !(fit_rpart$finalModel$frame$var == "<leaf>")
tree_terms <-
 fit_rpart$finalModel$frame$var[ind] |>
 unique() |>
 as.character()
tree_terms
```

What is the variable importance in the random forest call for these predictors? Where do they rank?

20. Extract the top 50 predictors based on importance, take a subset of `x` with just these predictors and apply the function `heatmap` to see how these genes behave across the tissues. We will introduce the `heatmap` function in Chapter 32.

21. Previously, we compared the conditional probability  $p(\mathbf{x})$  give two predictors  $(\mathbf{x}_1, \mathbf{x}_2)$  to the fit  $\hat{p}(\mathbf{x})$  obtained with a machine learning algorithm by making image plots. The following code can be used to make these images and include a curve at the values of  $(x_1)$  and  $(x_2)$  for which the function is 0.5:

```
plot_cond_prob <- function(x_1, x_2, p){
 data.frame(x_1 = x_1, x_2 = x_2, p = p) |>
 ggplot(aes(x_1, x_2)) +
 geom_raster(aes(fill = p), show.legend = FALSE) +
 stat_contour(aes(z = p), breaks = 0.5, color = "black") +
 scale_fill_gradientn(colors = c("#F8766D", "white", "#00BFC4"))
}
```

We can see the true conditional probability for the 2 or 7 example like this:

```
with(mnist_27$true_p, plot_cond_prob(x_1, x_2, p))
```

Fit a kNN model and make this plot for the estimated conditional probability. Hint: Use the argument `newdata = mnist27$train` to obtain predictions for a grid points.

22. Notice that, in the plot made in exercise 1, the boundary is somewhat wiggly. This is because kNN, like the basic bin smoother, does not use a kernel. To improve this we could try loess. By reading through the available models part of the `caret` manual, we see that we can use the `gamLoess` method. We need to install the `gam` package, if we have not done so already. We see that we have two parameters to optimize:

```
modelLookup("gamLoess")
#> model parameter label forReg forClass probModel
#> 1 gamLoess span Span TRUE TRUE TRUE
#> 2 gamLoess degree Degree TRUE TRUE TRUE
```

Use cross-validation to pick a span between 0.15 and 0.75. Keep `degree = 1`. What span does cross-validation select?

23. Show an image plot of the estimate  $\hat{p}(\mathbf{x}, y)$  resulting from the model fit in the exercise 2. How does the accuracy compare to that of kNN? Comment on the difference between the estimate obtained with kNN.

24. Use the `mnist_27` training set to build a model with several of the models available from the `caret` package. For example, you can try these:

```
models <- c("glm", "lda", "naive_bayes", "svmLinear", "gamboost",
 "gamLoess", "qda", "knn", "kknn", "loclda", "gam", "rf",
 "ranger", "wsrf", "Rborist", "avNNet", "mlp", "monmlp", "gbm",
 "adaboost", "svmRadial", "svmRadialCost", "svmRadialSigma")
```

We have not explained many of these, but apply them anyway using `train` with all the default parameters. Keep the results in a list. You might need to install some packages. Keep in mind that you will likely get some warnings.

25. Now that you have all the trained models in a list, use `sapply` or `map` to create a matrix of predictions for the test set. You should end up with a matrix with `length(mnist_27$test$y)` rows and `length(models)` columns.

26. Compute accuracy for each model on the test set.

27. Build an ensemble prediction by majority vote and compute its accuracy.

28. Earlier we computed the accuracy of each method on the training set and noticed they varied. Which individual methods do better than the ensemble?

29. It is tempting to remove the methods that do not perform well and re-do the ensemble. The problem with this approach is that we are using the test data to make a decision. However, we could use the accuracy estimates obtained from cross validation with the training data. Obtain these estimates and save them in an object.

30. Now let's only consider the methods with an estimated accuracy of 0.8 when constructing the ensemble. What is the accuracy now?

31. Note that if two machine algorithms predict the same outcome, ensembling them will not change the prediction. For each pair of algorithms compare the percent of observations for which they make the same prediction. Use this to define a function and then use the `heatmap` function to visualize the results. Hint: use the `method = "binary"` argument in the `dist` function.

32. Note that each method can also produce an estimated conditional probability. Instead of majority vote, we can take the average of these estimated conditional probabilities. For most methods, we can the use the `type = "prob"` in the `train` function. Note that some of the methods require you to use the argument `trControl=trainControl(classProbs=TRUE)` when calling `train`. Also, these methods do not work if classes have numbers as names. Hint: change the levels like this:

```
dat$train$y <- recode_factor(dat$train$y, "2"="two", "7"="seven")
dat$test$y <- recode_factor(dat$test$y, "2"="two", "7"="seven")
```

33. In this chapter, we illustrated a couple of machine learning algorithms on a subset of the MNIST dataset. Try fitting a model to the entire dataset.

- 
1. <https://topepo.github.io/caret/available-models.html>
  2. <http://topepo.github.io/caret/available-models.html>
  3. <https://topepo.github.io/caret/available-models.html>
  4. <https://topepo.github.io/caret/using-your-own-model-in-train.html>

# Introduction to Data Science - 32 Clustering

Rafael A. Irizarry

The algorithms we have described up to now are examples of a general approach referred to as *supervised* machine learning. The name comes from the fact that we use the outcomes in a training set to *supervise* the creation of our prediction algorithm. There is another subset of machine learning referred to as *unsupervised*. In this subset, we do not necessarily know the outcomes and instead are interested in discovering groups. These algorithms are also referred to as *clustering* algorithms since predictors are used to define *clusters*.

In the two examples we have shown here, clustering would not be very useful. In the first example, if we are simply given the heights, we will not be able to discover two groups, males and females, because the intersection is large. In the second example, we can see from plotting the predictors that discovering the two digits, two and seven, will be challenging.

However, there are applications in which unsupervised learning can be a powerful technique, in particular as an exploratory tool.

A first step in any clustering algorithm is defining a distance between observations or groups of observations. Then we need to decide how to join observations into clusters. There are many algorithms for doing this. Here we introduce two as examples: hierarchical and k-means.

We will construct a simple example based on movie ratings. Here we quickly construct a matrix `x` that has ratings for the 50 movies with the most ratings.

```
library(tidyverse)
library(janitor)
library(dslabs)
top_movies <- movielens |> count(movieId) |> top_n(50, n) |> pull(movieId)
x <- movielens |> filter(movieId %in% top_movies)
top_users <- x |> count(userId) |> top_n(50, n) |> pull(userId)
x <- x |> filter(userId %in% top_users) |>
 select(title, userId, rating) |>
 pivot_wider(names_from = userId, values_from = rating) |>
 column_to_rrownames("title") |>
 as.matrix()
rownames(x) <- str_remove(rownames(x), ": Episode") |> str_trunc(20)
x <- x - rowMeans(x, na.rm = TRUE)
```

We want to use these data to find out if there are clusters of movies based on the ratings from 51 movie raters. A first step is to find the distance between each pair of movies using the `dist` function:

```
d <- dist(x)
```

## 32.1 Hierarchical clustering

With the distance between each pair of movies computed, we need an algorithm to define groups, based on these distances. Hierarchical clustering starts by defining each observation as a separate group, then the two closest groups are joined into new groups. We then continue joining the closest groups into new groups iteratively until there is just one group including all the observations. The `hclust` function implements this algorithm and takes a distance as input.

```
h <- hclust(d)
```

We can see the resulting groups using a *dendrogram*. The function `plot` applied to an `hclust` object creates a dendrogram:

```
plot(h, cex = 0.65, main = "", xlab = "")
```

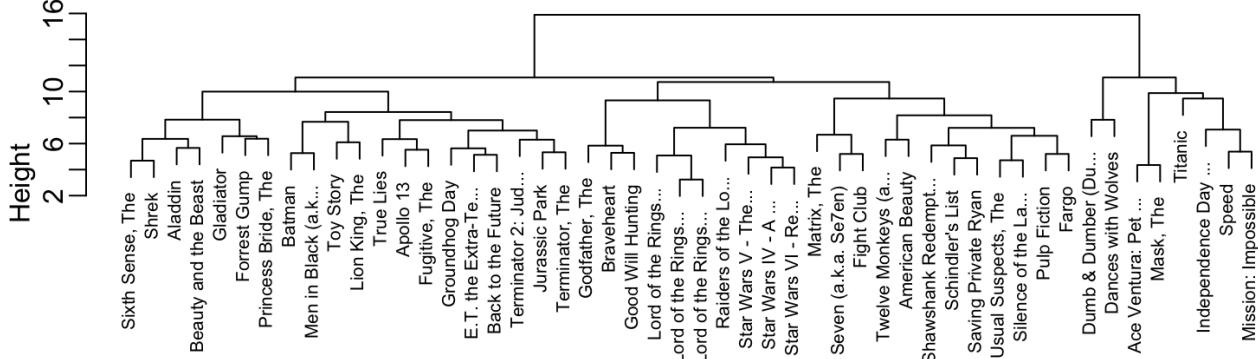


Figure 1:

This graph gives us an approximation between the distance between any two movies. To find this distance, we find the first location, from top to bottom, where these movies split into two different groups. The height of this location is the distance between these two groups. So, for example, the distance between the three *Star Wars* movies is 8 or less, while the distance between *Raiders of the Lost Ark* and *Silence of the Lambs* is about 17.

To generate actual groups, we can do one of two things: 1) decide on a minimum distance needed for observations to be in the same group or 2) decide on the number of groups you want and then find the minimum distance that achieves this. The function `cutree` can be applied to the output of `hclust` to perform either of these two operations and generate groups.

```
groups <- cutree(h, k = 10)
```

Note that the clustering provides some insights into types of movies. Group 4 appears to be blockbusters:

```
names(groups)[groups == 4]
#> [1] "Braveheart" "Godfather, The" "Good Will Hunting"
```

And Group 9 appears to be nerd movies:

```
names(groups)[groups == 6]
#> [1] "Star Wars IV - A ..." "Star Wars V - The..." "Raiders of the Lo..."
#> [4] "Star Wars VI - Re..." "Lord of the Rings..." "Lord of the Rings..."
#> [7] "Lord of the Rings..."
```

We can change the size of the group by either making `k` larger or `h` smaller.

We can also explore the data to see if there are clusters of movie raters:

```
h_2 <- hclust(dist(t(x)))
```

## 32.2 k-means

To use the k-means clustering algorithm we have to pre-define  $\backslash(k\backslash)$ , the number of clusters we want to define. The k-means algorithm is iterative.

The first step is to define  $\backslash(k\backslash)$  centers. Then each observation is assigned to the cluster with the closest center to that observation. In a second step, the centers are redefined using the observation in each cluster: the column means are used to define a *centroid*. We repeat these two steps until the centers converge.

The `kmeans` function included in R-base does not handle NAs. For illustrative purposes, we will fill out the NAs with 0s. In general, the choice of how to fill in missing data, or if one should do it at all, should be made with care.

```
x_0 <- x
x_0[is.na(x_0)] <- 0
k <- kmeans(x_0, centers = 10)
```

The cluster assignments are in the `cluster` component:

```
groups <- k$cluster
```

Note that because the first center is chosen at random, the final clusters are random. We impose some stability by repeating the entire function several times and averaging the results. The number of random starting values to use can be assigned through the `nstart` argument.

```
k <- kmeans(x_0, centers = 10, nstart = 25)
```

### 32.3 Heatmaps

A powerful visualization tool for discovering clusters or patterns in your data is the heatmap. The idea is simple: plot an image of your data matrix with colors used as the visual cue and both the columns and rows ordered according to the results of a clustering algorithm. We will demonstrate this with the `tissue_gene_expression` dataset.

We start by scaling the columns of the gene expression matrix because we only care about relative differences in gene expression. After scaling, we compute perform clustering on both the observations and the predictors:

```
x <- sweep(tissue_gene_expression$x, 2, colMeans(tissue_gene_expression$x))
h_1 <- hclust(dist(x))
h_2 <- hclust(dist(t(x)))
```

Now we can use the results of this clustering to order the rows and columns:

```
image(x[h_1$order, h_2$order])
```

The `heatmap` function that does all this for us:

```
heatmap(x, col = RColorBrewer::brewer.pal(11, "Spectral"))
```

Note we do not show the results of the `heatmap` function because there are too many features for the plot to be useful. We will therefore filter some columns and remake the plots.

#### 32.3.1 Filtering features

If only a few features are different between clusters, including all the features can add enough noise that making cluster detection challenging. A simple approach to avoid this is to assume low variability features are not informative and include only high variance features. For example, in the movie example, users with low variance in their ratings are not really distinguishing movies: all the movies seem about the same to them.

Here is an example code showing how we can include only the features with high variance in a heatmap:

```
library(matrixStats)
sds <- colSds(x, na.rm = TRUE)
o <- order(sds, decreasing = TRUE)[1:25]
heatmap(x[,o], col = RColorBrewer::brewer.pal(11, "Spectral"))
```

Note there are several other heatmap functions in R. A popular example is the `heatmap.2` in the `gplots` package.

### 32.4 Exercises

1. Load the `tissue_gene_expression` dataset. Remove the row means and compute the distance between each observation. Store the result in `d`.

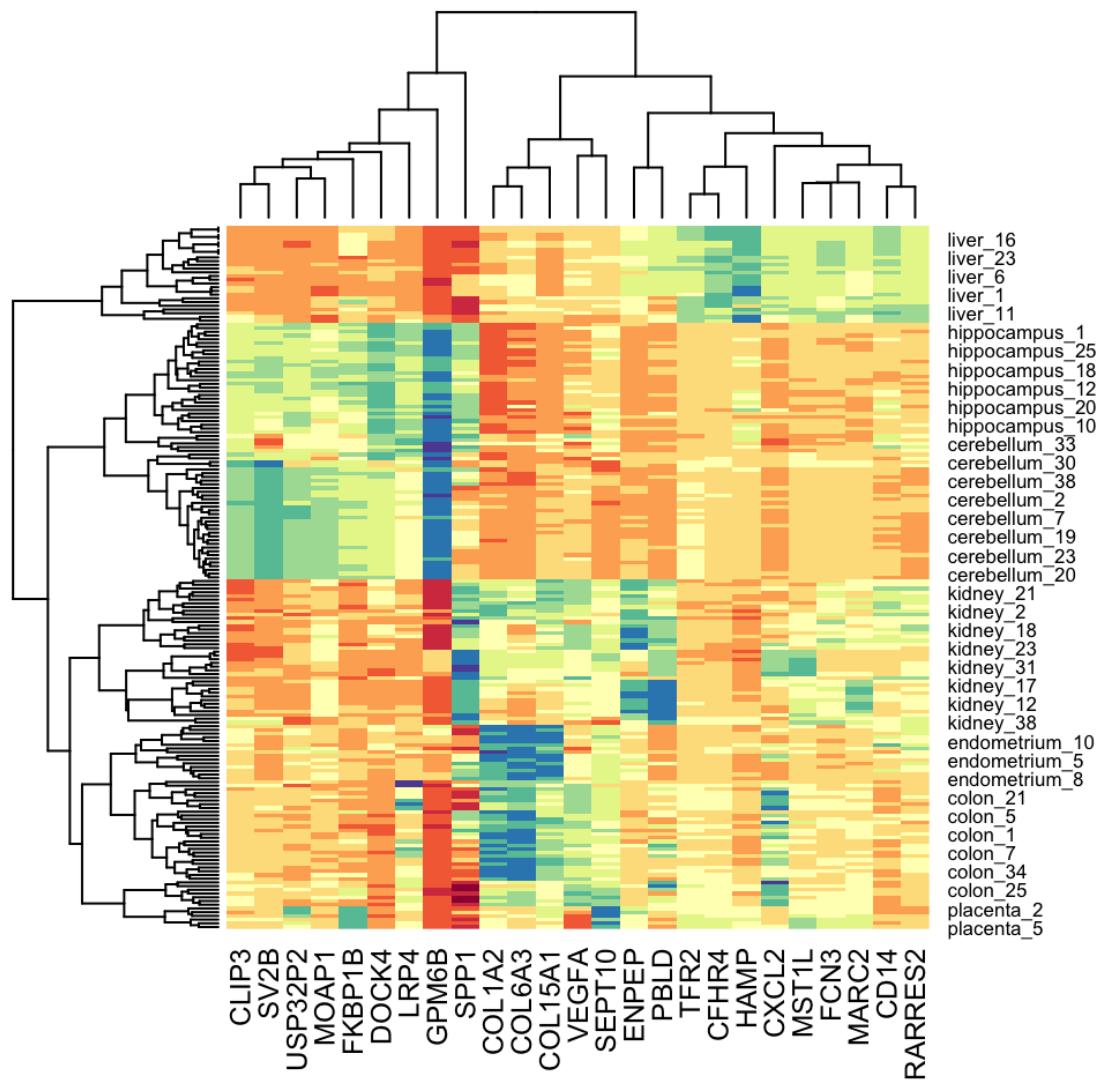


Figure 2:

2. Make a hierarchical clustering plot and add the tissue types as labels.
3. Run a k-means clustering on the data with  $\backslash(K=7\backslash)$ . Make a table comparing the identified clusters to the actual tissue types. Run the algorithm several times to see how the answer changes.
4. Select the 50 most variable genes. Make sure the observations show up in the columns, that the predictors are centered, and add a color bar to show the different tissue types. Hint: use the `ColSideColors` argument to assign colors. Also, use `col = RColorBrewer::brewer.pal(11, "RdBu")` for a better use of colors.