

Fundamentals of Biological Data Analysis

Stefano Allesina and Dmitry Kondrashov

Invalid Date

Table of contents

Organization of the class	9
Learning goals	9
Approach	9
Materials	10
Acknowledgements	11
1 Refresher	12
1.1 Goal	12
1.2 Motivation	12
1.3 Before we start	13
1.4 What is R?	13
1.5 RStudio	14
1.6 How to write a simple program	14
1.6.1 The most basic operation: assignment	14
1.6.2 Data types	15
1.6.3 Operators and functions	17
1.6.4 Getting help	19
1.6.5 Data structures	19
1.7 Reading and writing data	31
1.8 Conditional branching	33
1.9 Looping	35
1.10 Useful Functions	37
1.11 Packages	39
1.11.1 Installing a package	39
1.11.2 Loading a package	39
1.11.3 Example	40
1.12 Random numbers	40
1.13 Writing functions	41
1.14 Organizing and running code	43
1.15 Documenting the code using knitr	47
1.16 Resources	49
2 Visualizing data using ggplot2	50
2.1 Goal	50
2.2 Introduction to the Grammar of Graphics	50

2.3	Basic <code>ggplot2</code>	51
2.4	Building a well-formed graph	53
2.5	Scatterplots	57
2.6	Histograms, density and boxplots	58
2.7	Scales	66
2.8	List of aesthetic mappings	69
2.9	List of geometries	70
2.10	List of scales	70
2.11	Themes	71
2.12	Faceting	71
2.13	Setting features	74
2.14	Saving graphs	76
2.15	Multiple layers	77
2.16	Try on your own data!	77
2.17	Resources	77
3	Fundamentals of probability	79
3.1	Sample spaces and random variables	79
3.2	Probability axioms	81
3.3	Probability distributions	81
3.4	Measures of center: medians and means	82
3.5	Measures of spread: quartiles and variances	86
3.6	Data as samples from distributions: statistics	88
3.6.1	Law of large numbers	88
3.6.2	Central Limit Theorem	89
3.7	Exploration: misleading means	89
3.8	References	91
4	Data wrangling	92
4.1	Goal	92
4.2	What is data wrangling?	92
4.3	A new data type, <code>tibble</code>	93
4.4	Selecting rows and columns	93
4.5	Creating pipelines using <code>%>%</code>	95
4.6	Producing summaries	96
4.7	Summaries by group	97
4.8	Ordering the data	98
4.9	Renaming columns	99
4.10	Adding new variables using <code>mutate</code>	100
4.11	Data wrangling	102
4.12	From narrow to wide	103
4.13	From wide to narrow	103
4.14	Separate: split a column into two or more	105

4.15 Separate rows: from one row to many	106
4.16 Example: brown bear, brown bear, what do you see?	107
4.17 Resources	109
5 Distributions and their properties	110
5.1 Objectives:	110
5.2 Independence	110
5.2.1 Conditional probability	110
5.2.2 Independence	111
5.2.3 Usefulness of independence	112
5.3 Probability distribution examples (discrete)	114
5.3.1 Uniform	114
5.3.2 Binomial	115
5.3.3 Geometric	116
5.3.4 Poisson	118
5.4 Probability distribution examples (continuous)	119
5.4.1 Uniform	119
5.4.2 exponential	120
5.4.3 normal distribution	121
5.5 Application of normal distribution: confidence intervals	123
5.6 Identifying type of distribution in real data	125
6 Hypothesis testing	131
6.1 Test results vs. the truth	131
6.2 Types of errors	132
6.3 Test parameters and p-values	132
6.4 Multiple comparisons	134
6.5 Corrections for multiple comparisons	136
6.6 Two problems with science	136
6.6.1 Selective reporting	136
6.6.2 P-hacking	137
6.7 Readings	137
6.8 How to fool yourself with p-hacking (and possibly get fired!)	137
7 Likelihood and Bayes	143
7.1 Likelihood and estimation	143
7.1.1 likelihood vs. probability	143
7.1.2 maximizing likelihood	144
7.1.3 discrete probability distributions	145
7.1.4 continuous probability distributions	146
7.2 Bayesian thinking	147
7.2.1 Bayes' formula	148
7.2.2 positive predictive value	148

7.2.3	prosecutor's fallacy	150
7.2.4	reproducibility in science	151
7.3	Bayesian inference	151
7.3.1	Example: capture-recapture	152
7.3.2	MCMC	155
7.4	Reading:	156
8	Review of linear algebra	157
8.1	Solving multivariate linear equations	157
8.2	Fitting a line to data	159
8.2.1	Least-squares line	160
8.3	Linearity and vector spaces	164
8.3.1	Linear independence and basis vectors	166
8.3.2	Projections and changes of basis	167
8.4	Matrices as linear operators	169
8.4.1	Matrices transform vectors	169
8.4.2	calculating eigenvalues	170
9	Linear models	174
9.1	Regression toward the mean	174
9.2	Finding the best fitting line: Linear Regression	177
9.2.1	Solving a linear model — some linear algebra	178
9.2.2	Minimizing the sum of squares	180
9.2.3	Assumptions of linear regression	182
9.3	Linear regression in action	182
9.4	A regression gone wild	184
9.5	More advanced topics	188
9.5.1	Categorical variables in linear models	188
9.5.2	Interactions in linear models	189
9.5.3	Regression diagnostics	190
9.5.4	Plotting the residuals	192
9.5.5	Q-Q Plot	195
9.5.6	Cook's distance	198
9.5.7	Leverage	200
9.5.8	Running all diagnostics	203
9.6	Transforming the data	203
10	ANOVA	210
10.1	Analysis of variance	210
10.1.1	ANOVA assumptions	211
10.1.2	How one-way ANOVA works	211
10.2	Inference in one-way ANOVA	212
10.2.1	Example of comparing diets	213

10.2.2	Comparison of theory and ANOVA output	215
10.3	Further steps	217
10.3.1	Post-hoc analysis	217
10.3.2	Example of plant growth data	218
10.3.3	Two-way ANOVA	220
10.4	Investigate the UC salaries dataset	220
10.4.1	A word of caution about unbalanced designs	221
11	Model Selection	222
11.1	Goal	222
11.2	Problems	223
11.3	Approaches based on maximum-likelihoods	223
11.3.1	Likelihood function	223
11.3.2	Discrete probability distributions	224
11.3.3	Continuous probability distributions	224
11.4	Likelihoods for linear regression	224
11.5	Likelihood-ratio tests	225
11.6	AIC	233
11.7	Other information-based criteria	234
11.8	Bayesian approaches to model selection	234
11.8.1	Marginal likelihoods	234
11.8.2	Bayes factors	235
11.8.3	Bayes factors in practice	235
11.9	Using tidyverse for modeling and cross-validation	237
11.9.1	Prediction and cross-validation	243
11.10	Other approaches	247
11.10.1	Minimum description length	247
11.10.2	Cross validation	247
11.11	References and further reading:	253
12	Principal Component Analysis	254
12.1	Input	254
12.2	Singular Value Decomposition	255
12.3	SVD and PCA	259
12.3.1	PCA in R—from scratch	262
12.3.2	PCA in R — the easy way	266
12.4	Multidimensional scaling	267
12.4.1	Goal of MDS	267
12.4.2	Classic MDS	268
12.5	Readings	272
12.5.1	Exercise: PCA sommelier	272

13 Clustering	273
13.1 K-means algorithm	273
13.1.1 Assumptions of K-means algorithm	278
13.2 Hierarchical clustering	282
13.2.1 Agglomerative clustering	282
13.2.2 Clustering penguin data using hierarchical methods	285
13.3 Clustering analysis and validation	287
13.3.1 Hopkins statistic	287
13.3.2 Elbow method	288
13.3.3 Silhouette Plot	289
13.3.4 Lazy way: use all the methods!	290
13.3.5 Validation using bootstrapping	290
13.4 Application to breast cancer data	294
13.5 References:	298
14 Generalized linear models	299
14.1 Goal	299
14.2 Introduction	300
14.2.1 Model structure	300
14.3 Binary data	300
14.3.1 Logistic regression	301
14.3.2 A simple example	303
14.3.3 Exercise in class: College admissions	306
14.4 Count data	307
14.4.1 Poisson regression	307
14.4.2 Exercise in class: Number of genomes	308
14.4.3 Underdispersed and Overdispersed data	308
14.4.4 Exercise in class: Number of genomes	309
14.4.5 Separate distribution for the zeros	309
14.5 Other GLMs	310
14.6 Readings and homework	310
15 Machine learning methods for classification	311
15.1 Introduction	311
15.2 Naive Bayes classifier	311
15.2.1 Penguin data	312
15.2.2 Breast cancer data	313
15.3 Decision Trees	315
15.3.1 Penguin data	317
15.3.2 Breast cancer data	318
15.4 Random Forests	319
15.4.1 Penguin data	320
15.4.2 Cancer data	322

15.5 References	324
---------------------------	-----

Organization of the class

Learning goals

- R tools for visualizing and analyzing data
 - exploration of `tidyverse`
 - `dplyr`, `tidyr` and `readr` for data wrangling and organization
 - `ggplot2` for visualization
 - specific packages and functions for statistical analysis
- Theory to perform statistical inference
 - assumptions of different methods
 - hypothesis testing
 - estimation of parameters
 - model building and selection
- Avoiding common errors
 - when (not) to use a statistical method
 - sneaky paradoxes
 - phantom effects
- Work on your own data
 - analyze data
 - produce graphics
 - write up a report
 - present to class

Approach

- Mix of theory and practice
- Apply what you're learning to your own data

Materials

Week 0

- R refresher @ref(refresher)

Week 1

- Using `ggplot2` to produce publication-ready figures
- Review of probability

Week 2

- Data wrangling in `tidyverse`
- Probability distributions

Week 3

- Hypothesis testing
- Likelihood

Week 4

- Linear algebra refresher
- Linear models

Week 5

- Analysis of variance
- Model selection

Week 6

- Principal Component Analysis
- Multidimensional Scaling and Clustering

Week 7

- Generalized Linear Models
- Machine Learning and cross validation

Week 8

- Phylogenetic reconstruction (?)
- Modeling time-series data (?)

Week 9

Thanksgiving break

Week 10

- Student presentations 1
- Student presentations 2

Acknowledgements

Zach Miller for TAing the first iteration of the class, and for contributing materials and comments; Julia Smith for TAing the second iteration; Cassie Manrique for TAing the third iteration; Amatullah Mir for this year. Development of the class was partially supported by the Burroughs Wellcome Fund through the program “[Quantitative and statistical thinking in the life sciences](#)” (Stefano Allesina, PI).

1 Refresher

1.1 Goal

Introduce the statistical software R, and show how it can be used to analyze biological data in an automated, replicable way. Showcase the RStudio development environment, illustrate the notion of assignment, present the main data structures available in R. Show how to read and write data, how to execute simple programs, and how to modify the stream of execution of a program through conditional branching and looping. Introduce the use of packages and user-defined functions.

1.2 Motivation

When it comes to analyzing data, there are two competing paradigms. First, one could use point-and-click software with a graphical user interface, such as Excel, to perform calculations and draw graphs; second, one could write programs that can be run to perform the analysis of the data, the generation of tables and statistics, and the production of figures automatically.

This latter approach is to be preferred, because it allows for the automation of analysis, it requires a good documentation of the procedures, and is completely replicable.

A few motivating examples:

- You have written code to analyze your data. You receive from your collaborators a new batch of data. With simple modifications of your code, you can update your results, tables and figures automatically.
- A new student joins the laboratory. The new student can read the code and understand the analysis without the need of a lab mate showing the procedure step-by-step.
- The reviewers of your manuscript ask you to slightly alter the analysis. Rather than having to start over, you can modify a few lines of code and satisfy the reviewers.

Here we introduce R, which can help you write simple programs to analyze your data, perform statistical analysis, and draw beautiful figures.

1.3 Before we start

To follow this tutorial, you will need to install R and RStudio

- Install R: download and install R from [this page](#). Choose the right architecture (Windows, Mac, Linux). If possible, install the latest release.
- Install RStudio: go to [this page](#) and download the “RStudio Desktop Open Source License”.
- Install R packages: launch RStudio. Click on “Packages” in the bottom-right panel. Click on “Install”: a dialog window will open. Type `tidyverse` in the field “Packages” and click on “Install”. This might take a few minutes, and ask you to download further packages.

1.4 What is R?

R is a statistical software that is completely programmable. This means that one can write a program (script) containing a series of commands for the analysis of data, and execute them automatically. This approach is especially good as it makes the analysis of data well-documented, and completely replicable.

R is free software: anyone can download its source code, modify it, and improve it. The R community of users is vast and very active. In particular, scientists have enthusiastically embraced the program, creating thousands of packages to perform specific types of analysis, and adding many new capabilities. You can find a list of official packages (which have been vetted by R core developers) [here](#); many more are available on GitHub and other websites.

The main hurdle new users face when approaching R is that it is based on a command line interface: when you launch R, you simply open a console with the character > signaling that R is ready to accept an input. When you write a command and press Enter, the command is interpreted by R, and the result is printed immediately after the command. For example,

```
1 + 1
```

```
[1] 2
```

A little history: R was modeled after the commercial statistical software S by Robert Gentleman and Ross Ihaka. The project was started in 1992, first released in 1994, and the first stable version appeared in 2000. Today, R is managed by the *R Core Team*.

1.5 RStudio

For this introduction, we're going to use **RStudio**, an Integrated Development Environment (IDE) for **R**. The main advantage is that the environment will look identical irrespective of your computer architecture (Linux, Windows, Mac). Also, **RStudio** makes writing code much easier by automatically completing commands and file names (simply type the beginning of the name and press **Tab**), and allowing you to easily inspect data and code.

Typically, an **RStudio** window contains four panels:

- **Console** This is a panel containing an instance of **R**. For this tutorial, we will work mainly in this panel.
- **Source code** In this panel, you can write a program, save it to a file pressing **Ctrl + S** and then execute it by pressing **Ctrl + Shift + S**.
- **Environment** This panel lists all the variables you created (more on this later); another tab shows you the history of the commands you typed.
- **Plots** This panel shows you all the plots you drew. Other tabs allow you to access the list of packages you have loaded, and the help page for commands (just type `help(name_of_command)` in the Console) and packages.

1.6 How to write a simple program

An **R** program is simply a list of commands, which are executed one after the other. The commands are written in a text file (with extension `.R`). When **R** executes the program, it will start from the beginning of the file and proceed toward the end of the file. Every time **R** encounters a command, it will execute it. Special commands can modify this basic flow of the program by, for example, executing a series of commands only when a condition is met, or repeating the execution of a series of commands multiple times.

Note that if you were to copy and paste (or type) the code into the **Console** you would obtain exactly the same result. Writing a program is advantageous, however, because the analysis can be automated, and the code shared with other researchers. Moreover, after a while you will have a large code base, so that you can recycle much of your code.

We start by working on the console, and then start writing simple scripts.

1.6.1 The most basic operation: assignment

The most basic operation in any programming language is the assignment. In **R**, assignment is marked by the operator `<-` (can be typed quickly using **Alt -**). When you type a command in **R**, it is executed, and the output is printed in the **Console**. For example:

```
sqrt(9)
```

```
[1] 3
```

If we want to save the result of this operation, we can assign it to a variable. For example:

```
x <- sqrt(9)  
x
```

```
[1] 3
```

What has happened? We wrote a command containing an assignment operator (`<-`). R has evaluated the right-hand-side of the command (`sqrt(9)`), and has stored the result (3) in a newly created variable called `x`. Now we can use `x` in our commands: every time the command needs to be evaluated, the program will look up which value is associated with the variable `x`, and substitute it. For example:

```
x * 2
```

```
[1] 6
```

1.6.2 Data types

R provides different types of data that can be used in your programs. For each variable `x`, calling `class(x)` prints the type of the variable. The basic data types are:

- `logical`, taking only two possible values: `TRUE` and `FALSE`

```
v <- TRUE  
class(v)
```

```
[1] "logical"
```

- `numeric`, storing real numbers (actually, their approximations, as computers have limited memory and thus `cannot store` numbers like `,` or even `0.2`)

```
v <- 3.77  
class(v)
```

```
[1] "numeric"
```

- Real numbers can also be specified using scientific notation:

```
v <- 6.022e23 # 6.022 10^23 (Avogadro's number)
class(v)
```

```
[1] "numeric"
```

- **integer**, storing whole numbers

```
v <- 23L # the L signals that this should be stored as integer
class(v)
```

```
[1] "integer"
```

- **complex**, storing complex numbers (i.e., with a real and an imaginary part)

```
v <- 23 + 5i # the i marks the imaginary part
class(v)
```

```
[1] "complex"
```

- **character**, for strings, characters and text

```
v <- 'a string' # you can use single or double quotes
class(v)
```

```
[1] "character"
```

In R, the value and type of a variable are evaluated at run-time. This means that you can recycle the names of variables. This is very handy, but can make your programs more difficult to read and to debug (i.e., find mistakes). For example:

```
x <- '2.3' # this is a string
x
```

```
[1] "2.3"
```

```
x <- 2.3 # this is numeric  
x
```

```
[1] 2.3
```

1.6.3 Operators and functions

Each data type supports a certain number of operators and functions. For example, numeric variables can be combined with + (addition), - (subtraction), * (multiplication), / (division), and ^ (or **, exponentiation). A possibly unfamiliar operator is the modulo (%%), calculating the remainder of an integer division:

```
5 %% 3
```

```
[1] 2
```

meaning that $5 \% 3$ (5 integer divided by 3) is 1 with a remainder of 2

The modulo operator is useful to determine whether a number is divisible for another: if y is divisible by x , then $y \% x$ is 0.

R provides many built-in functions: each function has a name, followed by round parentheses surrounding the (possibly optional) function *arguments*. For example, these functions operate on **numeric** variables:

- `abs(x)` absolute value
- `sqrt(x)` square root
- `round(x, digits = 3)` round x to three decimal digits
- `cos(x)` cosine (also supported are all the usual trigonometric functions)
- `log(x)` natural logarithm (use `log10` for base 10 logarithms)
- `exp(x)` calculating e^x

Similarly, **character** variables have their own set of functions, such as:

- `toupper(x)` make uppercase
- `nchar(x)` count the number of characters in the string
- `paste(x, y, sep = "_")` concatenate strings, joining them using the separator _
- `strsplit(x, "_")` separate the string using the separator _

Calling a function meant for a certain data type on another will cause errors. If sensible, you can convert a type into another. For example:

```
v <- "2.13"
class(v)

[1] "character"

# if we call v * 2, we get an error.
# to avoid it, we can convert v to numeric:
as.numeric(v) * 2
```

```
[1] 4.26
```

If sensible, you can use the comparison operators `>` (greater), `<` (lower), `==` (equals), `!=` (differs), `>=` and `<=`, returning a logical value:

```
2 == sqrt(4)
```

```
[1] TRUE
```

```
2 < sqrt(4)
```

```
[1] FALSE
```

```
2 <= sqrt(4)
```

```
[1] TRUE
```

Exercise:

Why are two equal signs (`==`) used to check that two values are equal? What happens if you use only one `=` sign?

Similarly, you can concatenate several comparison and logical variables using `&` (and), `|` (or), and `!` (not):

```
(2 > 3) & (3 > 1)
```

```
[1] FALSE
```

```
(2 > 3) | (3 > 1)
```

```
[1] TRUE
```

1.6.4 Getting help

If you want to know more about a function, type `?my_function_name` in the console (e.g., `?abs`). This will open the help page in one of the panels on the right. The same can be accomplished calling `help(abs)`. For more complex questions, check out stackoverflow.

1.6.5 Data structures

Besides these simple types, R provides structured data types, meant to collect and organize multiple values.

1.6.5.1 Vectors

The most basic data structure in R is the vector, which is an ordered collection of values of the same type. Vectors can be created by concatenating different values with the function `c()` (“combine”):

```
x <- c(2, 3, 5, 27, 31, 13, 17, 19)  
x
```

```
[1] 2 3 5 27 31 13 17 19
```

You can access the elements of a vector by their index: the first element is indexed at 1, the second at 2, etc.

```
x[3]
```

```
[1] 5
```

```
x[8]
```

```
[1] 19
```

```
x[9] # what if the element does not exist?
```

```
[1] NA
```

NA stands for “Not Available”. Other special values are NaN (Not a Number, e.g., $0/0$), Inf (Infinity, e.g., $1/0$), and NULL (variable undefined). You can test for special values using `is.na(x)`, `is.infinite(x)`, `is.null(x)`, etc.

Note that in R a single number (string, logical) is a vector of length 1 by default. That’s why if you type 3 in the console you see [1] 3 in the output.

You can extract several elements at once (i.e., create another vector), using the colon (`:`) command, or by concatenating the indices:

```
x[1:3]
```

```
[1] 2 3 5
```

```
x[4:7]
```

```
[1] 27 31 13 17
```

```
x[c(1,3,5)]
```

```
[1] 2 5 31
```

You can also use a vector of logical variables to extract values from vectors. For example, suppose we have two vectors:

```
sex <- c("M", "M", "F", "M", "F") # sex of Drosophila
weight <- c(0.230, 0.281, 0.228, 0.260, 0.231) # weight in mg
```

and that we want to extract only the weights for the males.

```
sex == "M"
```

```
[1] TRUE TRUE FALSE TRUE FALSE
```

returns a vector of logical values, which we can use to subset the data:

```
weight[sex == "M"]
```

```
[1] 0.230 0.281 0.260
```

Given that R was born for statistics, there are many statistical functions you can perform on vectors:

```
length(x)
```

```
[1] 8
```

```
min(x)
```

```
[1] 2
```

```
max(x)
```

```
[1] 31
```

```
sum(x) # sum all elements
```

```
[1] 117
```

```
prod(x) # multiply all elements
```

```
[1] 105436890
```

```
median(x) # median value
```

```
[1] 15
```

```
mean(x) # arithmetic mean
```

```
[1] 14.625
```

```
var(x) # unbiased sample variance  
  
[1] 119.4107  
  
mean(x ^ 2) - mean(x) ^ 2 # population variance  
  
[1] 104.4844  
  
summary(x) # print a summary  
  
Min. 1st Qu. Median Mean 3rd Qu. Max.  
2.00 4.50 15.00 14.62 21.00 31.00
```

You can generate vectors of sequential numbers using the colon command:

```
x <- 1:10  
x  
  
[1] 1 2 3 4 5 6 7 8 9 10
```

For more complex sequences, use `seq`:

```
seq(from = 1, to = 5, by = 0.5)  
  
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

To repeat a value or a sequence several times, use `rep`:

```
rep("abc", 3)  
  
[1] "abc" "abc" "abc"  
  
rep(c(1, 2, 3), 3)
```

```
[1] 1 2 3 1 2 3 1 2 3
```

Exercise:

- Create a vector containing all the even numbers between 2 and 100 (inclusive) and store it in variable `z`.
- Extract all the elements of `z` that are divisible by 12. How many elements match this criterion?
- What is the sum of all the elements of `z`?
- Is it equal to $51 \cdot 50$?
- What is the product of elements 5, 10 and 15 of `z`?
- Does `seq(2, 100, by = 2)` produce the same vector as `(1:50) * 2`?
- What happens if you type `z ^ 2`?

1.6.5.2 Matrices

A matrix is a two-dimensional table of values. In case of numeric values, you can perform the usual operations on matrices (product, inverse, decomposition, etc.):

```
A <- matrix(c(1, 2, 3, 4), 2, 2) # values, nrows, ncols  
A
```

```
[,1] [,2]  
[1,]    1     3  
[2,]    2     4
```

```
A %*% A # matrix product
```

```
[,1] [,2]  
[1,]    7    15  
[2,]   10    22
```

```
solve(A) # matrix inverse
```

```
[,1] [,2]  
[1,] -2  1.5  
[2,]  1 -0.5
```

```
A %*% solve(A) # this should return the identity matrix
```

```
[,1] [,2]  
[1,] 1 0  
[2,] 0 1
```

```
B <- matrix(1, 3, 2) # you can fill the whole matrix with a single number (1)  
B
```

```
[,1] [,2]  
[1,] 1 1  
[2,] 1 1  
[3,] 1 1
```

```
B %*% t(B) # transpose
```

```
[,1] [,2] [,3]  
[1,] 2 2 2  
[2,] 2 2 2  
[3,] 2 2 2
```

```
Z <- matrix(1:9, 3, 3) # by default, matrices are filled by column  
Z
```

```
[,1] [,2] [,3]  
[1,] 1 4 7  
[2,] 2 5 8  
[3,] 3 6 9
```

To determine the dimensions of a matrix, use `dim`:

```
dim(B)
```

```
[1] 3 2
```

```
dim(B)[1]
```

```
[1] 3
```

```
nrow(B)
```

```
[1] 3
```

```
dim(B)[2]
```

```
[1] 2
```

```
ncol(B)
```

```
[1] 2
```

```
nrow(B)
```

```
[1] 3
```

Use indices to access a particular row/column of a matrix:

```
Z
```

```
[,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5    8  
[3,]    3    6    9
```

```
Z[1, ] # first row
```

```
[1] 1 4 7
```

```

Z[, 2] # second column

[1] 4 5 6

Z [1:2, 2:3] # submatrix with coefficients in first two rows, and second and third column

[,1] [,2]
[1,]    4    7
[2,]    5    8

Z[c(1, 3), c(1, 3)] # indexing non-adjacent rows/columns

[,1] [,2]
[1,]    1    7
[2,]    3    9

```

Some functions use all the elements of the matrix:

```
sum(Z)
```

```
[1] 45
```

```
mean(Z)
```

```
[1] 5
```

Some functions apply the operation across a given dimension (e.g., columns) of the matrix:

```
rowSums(Z) # returns a vector of the sums of the values in each row
```

```
[1] 12 15 18
```

```
colSums(Z) # does the same for columns
```

```
[1] 6 15 24
```

```
rowMeans(Z) # returns a vector of the means of the values in each row
```

```
[1] 4 5 6
```

```
colMeans(Z) # does the same for columns
```

```
[1] 2 5 8
```

1.6.5.3 Arrays

If you need tables with more than two dimensions, use arrays:

```
M <- array(1:24, c(4, 3, 2))  
M
```

```
, , 1
```

```
[,1] [,2] [,3]  
[1,] 1 5 9  
[2,] 2 6 10  
[3,] 3 7 11  
[4,] 4 8 12
```

```
, , 2
```

```
[,1] [,2] [,3]  
[1,] 13 17 21  
[2,] 14 18 22  
[3,] 15 19 23  
[4,] 16 20 24
```

You can still determine the dimensions using:

```
dim(M)
```

```
[1] 4 3 2
```

and access the elements as done for matrices. One thing you should be paying attention to: R drops dimensions that are not needed. So, if you access a “slice” of a 3-dimensional array:

```
M[, , 1]
```

```
[,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

you obtain a matrix:

```
dim(M[, , 1])
```

```
[1] 4 3
```

This can be problematic, for example, when your code expects an array and R turns your data into a matrix (or you expect a matrix but find a vector). To avoid this behavior, add `drop = FALSE` when subsetting:

```
dim(M[, , 1, drop = FALSE])
```

```
[1] 4 3 1
```

1.6.5.4 Lists

Vectors are good if each element is of the same type (e.g., numbers, strings). Lists are used when we want to store elements of different types, or more complex objects (e.g., vectors, matrices, even lists of lists). Each element of the list can be referenced either by its index, or by a label:

```
mylist <- list(Names = c("a", "b", "c", "d"), Values = c(1, 2, 3))
mylist
```

```
$Names
[1] "a" "b" "c" "d"
```

```
$Values
[1] 1 2 3
```

```

mylist[[1]] # access first element using index

[1] "a" "b" "c" "d"

mylist[[2]] # access second element by index

[1] 1 2 3

mylist$Names # access second element by label

[1] "a" "b" "c" "d"

mylist[["Names"]][3] # another way to access by label

[1] "a" "b" "c" "d"

mylist[["Values"]][3] # access third element in second vector

[1] 3

```

1.6.5.5 Data frames

Data frames contain data organized like in a spreadsheet. The columns (typically representing different measurements) can be of different types (e.g., a column could be the date of measurement, another the weight of the individual, or the volume of the cell, or the treatment of the sample), while the rows typically represent different samples.

When you read a spreadsheet file in R, it is automatically stored as a data frame. The difference between a matrix and a data frame is that in a matrix all the values are of the same type (e.g., all numeric), while in a data frame each column can be of a different type.

Because typing a data frame by hand would be tedious, let's use a data set that is already available in R:

```

data(trees) # girth, height and volume of cherry trees
str(trees) # structure of data frame

```

```

'data.frame':   31 obs. of  3 variables:
$ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
$ Height: num  70 65 63 72 81 83 66 75 80 75 ...
$ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...

  ncol(trees)

[1] 3

  nrow(trees)

[1] 31

  head(trees) # print the first few rows

  Girth Height Volume
1    8.3     70   10.3
2    8.6     65   10.3
3    8.8     63   10.2
4   10.5     72   16.4
5   10.7     81   18.8
6   10.8     83   19.7

  summary(trees) # Quickly get an overview of the data frame.

  Girth          Height         Volume
Min.   : 8.30   Min.   :63   Min.   :10.20
1st Qu.:11.05  1st Qu.:72   1st Qu.:19.40
Median :12.90  Median :76   Median :24.20
Mean   :13.25  Mean   :76   Mean   :30.17
3rd Qu.:15.25  3rd Qu.:80   3rd Qu.:37.30
Max.   :20.60  Max.   :87   Max.   :77.00

  trees$Girth # select column by name

[1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4 11.4 11.7 12.0
[16] 12.9 12.9 13.3 13.7 13.8 14.0 14.2 14.5 16.0 16.3 17.3 17.5 17.9 18.0 18.0
[31] 20.6

```

```
trees$Height[1:5] # select column by name; return first five elements
```

```
[1] 70 65 63 72 81
```

```
trees[1:3, ] #select rows 1 through 3
```

	Girth	Height	Volume
1	8.3	70	10.3
2	8.6	65	10.3
3	8.8	63	10.2

```
trees[1:3, ]$Volume # select rows 1 through 3; return column Volume
```

```
[1] 10.3 10.3 10.2
```

```
trees <- rbind(trees, c(13.25, 76, 30.17)) # add a row  
trees_double <- cbind(trees, trees) # combine columns  
colnames(trees) <- c("Circumference", "Height", "Volume") # change column names
```

Exercise:

- What is the average height of the cherry trees?
- What is the average girth of those that are more than 75 ft tall?
- What is the maximum height of trees with a volume between 15 and 35 ft³?

1.7 Reading and writing data

In most cases, you will not generate your data in R, but import it from a file. By far, the best option is to have your data in a comma separated value text file or in a tab separated file. Then, you can use the function `read.csv` (or `read.table`) to import your data. The syntax of the functions is as follows:

```
read.csv("MyFile.csv") # read the file MyFile.csv  
read.csv("MyFile.csv", header = TRUE) # the file has a header  
read.csv("MyFile.csv", sep = ';') # specify the column separator  
read.csv("MyFile.csv", skip = 5) # skip the first 5 lines
```

Note that columns containing strings are typically converted to *factors* (categorical values, useful when performing regressions). To avoid this behavior, you can specify `stringsAsFactors = FALSE` when calling the function.

Similarly, you can save your data frames using `write.table` or `write.csv`. Suppose you want to save the data frame `MyDF`:

```
write.csv(MyDF, "MyFile.csv")
write.csv(MyDF, "MyFile.csv", append = TRUE) # append to the end of the file
write.csv(MyDF, "MyFile.csv", row.names = TRUE) # include the row names
write.csv(MyDF, "MyFile.csv", col.names = FALSE) # do not include column names
```

Let's look at an example: Read a file containing data on the 6th chromosome for a number of Europeans (Data adapted from [Stanford HGDP SNP Genotyping Data](#) by John Novembre). This example shows that you can read data directly from the internet!

```
# The actual URL is
# https://github.com/StefanoAllesina/BSD-QBio4/raw/master/tutorials/basic_computing_1/data
ch6 <- read.table("https://tinyurl.com/y7vctq3v",
                  header = TRUE, stringsAsFactors = FALSE)
```

where `header = TRUE` means that we want to take the first line to be a header containing the column names. How big is this table?

```
dim(ch6)
```

```
[1] 43141      7
```

we have 7 columns, but more than 40k rows! Let's see the first few:

```
head(ch6)
```

	CHR	SNP	A1	A2	nA1A1	nA1A2	nA2A2
1	6	rs4959515	A	G	0	17	107
2	6	rs719065	A	G	0	26	98
3	6	rs6596790	C	T	0	4	119
4	6	rs6596796	A	G	0	22	102
5	6	rs1535053	G	A	5	39	80
6	6	rs12660307	C	T	0	3	121

and the last few:

```
tail(ch6)
```

CHR	SNP	A1	A2	nA1A1	nA1A2	nA2A2
43136	6 rs10946282	C	T	0	16	108
43137	6 rs3734763	C	T	19	56	48
43138	6 rs960744	T	C	32	60	32
43139	6 rs4428484	A	G	1	11	112
43140	6 rs7775031	T	C	26	56	42
43141	6 rs12213906	C	T	1	11	112

The data contains the number of homozygotes (`nA1A1`, `nA2A2`) and heterozygotes (`nA1A2`), for 43,141 single nucleotide polymorphisms (SNPs) obtained by sequencing European individuals:

- `CHR` The chromosome (6 in this case)
- `SNP` The identifier of the Single Nucleotide Polymorphism
- `A1` One of the alleles
- `A2` The other allele
- `nA1A1` The number of individuals with the particular combination of alleles.

Exercise:

- How many individuals were sampled? Find the maximum of the sum `nA1A1 + nA1A2 + nA2A2`. Note: you can access the columns by index (e.g., `ch6[,5]`), or by name (e.g., `ch6$nA1A1`, or also `ch6[, "nA1A1"]`).
- Try using the function `rowSums` to obtain the same result.
- For how many SNPs do we have that all sampled individuals are homozygotes (i.e., all `A1A1` or all `A2A2`)?
- For how many SNPs, are more than 99% of the sampled individuals homozygous?

1.8 Conditional branching

Now we turn to writing actual programs in the **Source code** panel. To start a new R program, press **Ctrl + Shift + N**. This will open an **Untitled** script. Save the script by pressing **Ctrl + S**: save it as `conditional.R` in the directory `programming_skills/sandbox/`. To make sure you're working in the directory where the script is contained, on the menu on the top choose **Session -> Set Working Directory -> To Source File Location**.

Now type the following script:

```
print("Hello world!")
x <- 4
print(x)
```

and execute the script by pressing **Ctrl + Shift + S**. You should see **Hello World!** and **4** printed in your console.

As you saw in this simple example, when R executes the program, it starts from the top and proceeds toward the end of the file. Every time it encounters a command (for example, `print(x)`, printing the value of `x` into the console), it executes it.

When we want a certain block of code to be executed only when a certain condition is met, we can write a conditional branching point. The syntax is as follows:

```
if (condition is met){
  # execute this block of code
} else {
  # execute this other block of code
}
```

For example, add these lines to the script `conditional.R`, and run it again:

```
print("Hello world!")
x <- 4
print(x)
if (x %% 2 == 0){
  my_message <- paste(x, "is even")
} else {
  my_message <- paste(x, "is odd")
}
print(my_message)
```

We have created a conditional branching point, so that the value of `my_message` changes depending on whether `x` is even (and thus the remainder of the integer division by 2 is 0), or odd. Change the line `x <- 4` to `x <- 131` and run it again.

Exercise: What does this do?

```
x <- 36
if (x > 20){
  x <- sqrt(x)
} else {
  x <- x ^ 2
```

```

}
if (x > 7) {
  print(x)
} else if (x %% 2 == 1){
  print(x + 1)
}

```

1.9 Looping

Another way to change the flow of the program is to write a loop. A loop is simply a series of commands that are repeated a number of times. For example, you want to run the same analysis on different data sets that you collected; you want to plot the results contained in a set of files; you want to test your simulation over a number of parameter sets; etc.

R provides you with two ways to loop over blocks of commands: the `for` loop, and the `while` loop. Let's start with the `for` loop, which is used to iterate over a vector (or a list): for each value of the vector, a series of commands will be run, as shown by the following example, which you can type in a new script called `forloop.R`.

```

myvec <- 1:10 # vector with numbers from 1 to 10

for (i in myvec) {
  a <- i ^ 2
  print(a)
}

```

In the code above, the variable `i` takes the value of each element of `myvec` in sequence. Inside the block defined by the `for` loop, you can use the variable `i` to perform operations.

The anatomy of the `for` statement:

```

for (variable in list_or_vector) {
  execute these commands
} # automatically moves to the next value

```

For loops are used when you know that you want to perform the analysis using a given set of values (e.g., run over all files of a directory, all samples in your data, all sequences of a fasta file, etc.).

The `while` loop is used when the commands need to be repeated while a certain condition is true, as shown by the following example, which you can type in a script called `whileloop.R`:

```
i <- 1

while (i <= 10) {
  a <- i ^ 2
  print(a)
  i <- i + 1
}
```

The script performs exactly the same operations we wrote for the `for` loop above. Note that you need to update the value of `i`, (using `i <- i + 1`), otherwise the loop will run forever (infinite loop—to terminate click on the stop button in the top-right corner of the console). The anatomy of the `while` statement:

```
while (condition is met) {
  execute these commands
} # beware of infinite loops: remember to update the condition!
```

You can break a loop using the command `break`. For example:

```
i <- 1

while (i <= 10) {
  if (i > 5) {
    break
  }
  a <- i ^ 2
  print(a)
  i <- i + 1
}
```

Exercise: What does this do? Try to guess what each loop does, and then create and run a script to confirm your intuition.

```
z <- seq(1, 1000, by = 3)
for (k in z) {
  if (k %% 4 == 0) {
    print(k)
  }
}
```

```

z <- readline(prompt = "Enter a number: ")
z <- as.numeric(z)
isthisspecial <- TRUE
i <- 2
while (i < z) {
  if (z %% i == 0) {
    isthisspecial <- FALSE
    break
  }
  i <- i + 1
}
if (isthisspecial == TRUE) {
  print(z)
}

```

1.10 Useful Functions

Here's a short list of useful functions that will help you write your programs:

- `range(x)`: minimum and maximum of a vector `x`
- `sort(x)`: sort a vector `x`
- `unique(x)`: remove duplicate entries from vector `x`
- `which(x == a)`: returns a vector of the indices of `x` having value `a`
- `list.files("path_to_directory")`: list the files in a directory (current directory if not specified)
- `table(x)` build a table of frequencies

Exercises: What does this code do? For each snippet of code, first try to guess what will happen. Then, write a script and run it to confirm your intuition.

```

v <- c(1, 3, 5, 5, 3, 1, 2, 4, 6, 4, 2)
v <- sort(unique(v))
for (i in v){
  if (i > 2){
    print(i)
  }
  if (i > 4){
    break
  }
}

```

```

x <- 1:100
x <- x[which(x %% 7 == 0)]


my_amount <- 10
while (my_amount > 0){
  my_color <- NA
  while(is.na(my_color)){
    tmp <- readline(prompt="Do you want to bet on black or red? ")
    tmp <- tolower(tmp)
    if (tmp == "black") my_color <- "black"
    if (tmp == "red") my_color <- "red"
    if (is.na(my_color)) print("Please enter either red or black")
  }
  my_bet <- NA
  while(is.na(my_bet)){
    tmp <- readline(prompt="How much do you want to bet? ")
    tmp <- as.numeric(tmp)
    if (is.numeric(tmp) == FALSE){
      print("Please enter a number")
    } else {
      if (tmp > my_amount){
        print("You don't have enough money!")
      } else {
        my_bet <- tmp
        my_amount <- my_amount - tmp
      }
    }
  }
  lady_luck <- sample(c("red", "black"), 1)
  if (lady_luck == my_color){
    my_amount <- my_amount + 2 * my_bet
    print(paste("You won!! Now you have", my_amount, "gold doubloons"))
  } else {
    print(paste("You lost!! Now you have", my_amount, "gold doubloons"))
  }
}

```

1.11 Packages

R is the most popular statistical computing software among biologists due to its highly specialized packages, often written by biologists for biologists. You can contribute a package too! The RStudio support (goo.gl/harVqF) provides guidance on how to start developing R packages and Hadley Wickham's free online book (r-pkgs.had.co.nz) will make you a pro.

You can find highly specialized packages to address your research questions. Here are some suggestions for finding an appropriate package. The Comprehensive R Archive Network (CRAN) offers several ways to find specific packages for your task. You can either browse packages (goo.gl/7oVyKC) and their short description or select a scientific field of interest (goo.gl/0WdIcu) to browse through a compilation of packages related to each discipline.

From within your R terminal or RStudio you can also call the function `RSiteSearch("KEYWORD")`, which submits a search query to the website search.r-project.org. The website rseek.org casts an even wider net, as it not only includes package names and their documentation but also blogs and mailing lists related to R. If your research interests relate to high-throughput genomic data, you should have a look the packages provided by Bioconductor (goo.gl/7dwQ1q).

1.11.1 Installing a package

To install a package type

```
install.packages("name_of_package")
```

in the **Console**, or choose the panel **Packages** and then click on *Install* in RStudio.

1.11.2 Loading a package

To load a package type

```
library(name_of_package)
```

or call the command into your script. If you want your script to automatically install a package in case it's missing, use this boilerplate:

```
if (!require(needed_package, character.only = TRUE, quietly = TRUE)) {  
  install.packages(needed_package)  
  library(needed_package, character.only = TRUE)  
}
```

1.11.3 Example

For example, say we want to access the dataset `bacteria`, which reports the incidence of *H. influenzae* in Australian children. The dataset is contained in the package `MASS`.

First, we need to load the package:

```
library(MASS)
```

Now we can load the data:

```
data(bacteria)
bacteria[1:3,]
```

```
y ap hilo week ID      trt
1 y  p    hi     0 X01 placebo
2 y  p    hi     2 X01 placebo
3 y  p    hi     4 X01 placebo
```

1.12 Random numbers

To perform randomization, or any simulation, we typically need to draw random numbers. R has functions to sample random numbers from very many different statistical distributions. For example:

```
runif(5) # sample 5 numbers from the uniform distribution between 0 and 1
```

```
[1] 0.6266466 0.8684844 0.2766648 0.1832905 0.7143638
```

```
runif(5, min = 1, max = 9) # set the limits of the uniform distribution
```

```
[1] 7.584096 4.480323 6.867704 7.920640 2.482157
```

```
rnorm(3) # three values from standard normal
```

```
[1] -1.597643 -1.187024 -1.368285
```

```
rnorm(3, mean = 5, sd = 4) # specify mean and standard deviation
```

```
[1] 11.908289 10.298927 4.154934
```

To sample from a set of values, use `sample`:

```
v <- c("a", "b", "c", "d")
sample(v, 2) # without replacement
```

```
[1] "c" "d"
```

```
sample(v, 6, replace = TRUE) # with replacement
```

```
[1] "a" "b" "a" "a" "a" "c"
```

```
sample(v) # simply shuffle the elements
```

```
[1] "a" "d" "c" "b"
```

1.13 Writing functions

The R community provides about 7,000 packages. Still, sometimes there isn't an already made function capable of doing what you need. In these cases, you can write your own functions. In fact, it is generally a good idea to always divide your analysis into functions, and then write a small “master” program that calls the functions and performs the analysis. In this way, the code will be much more legible, and you will be able to recycle the functions for your other projects.

A function in R has this form:

```
my_function_name <- function(optional, arguments, separated, by_commas){
  # Body of the function
  # ...
  #
  return(return_value) # this is optional
}
```

A few examples:

```
sum_two_numbers <- function(a, b){  
  apb <- a + b  
  return(apb)  
}  
sum_two_numbers(5, 7.2)
```

```
[1] 12.2
```

You can set a default value for some of the arguments: if not specified by the user, the function will use these defaults:

```
sum_two_numbers <- function(a = 1, b = 2){  
  apb <- a + b  
  return(apb)  
}  
sum_two_numbers()
```

```
[1] 3
```

```
sum_two_numbers(3)
```

```
[1] 5
```

```
sum_two_numbers(b = 9)
```

```
[1] 10
```

The return value is optional:

```
my_factorial <- function(a = 6){  
  if (as.integer(a) != a) {  
    print("Please enter an integer!")  
  } else {  
    tmp <- 1  
    for (i in 2:a){  
      tmp <- tmp * i  
    }  
  }
```

```
    print(paste(a, "! = ", tmp, sep = ""))
}
my_factorial()
```

```
[1] "6! = 720"
```

```
my_factorial(10)
```

```
[1] "10! = 3628800"
```

You can return **only one** object. If you need to return multiple values, organize them into a vector/matrix/list and return that.

```
order_two_numbers <- function(a, b){
  if (a > b) return(c(a, b)) #nothing after the first return is executed
  return(c(b,a))
}

order_two_numbers(runif(1), runif(1))
```

```
[1] 0.7603180 0.5706694
```

1.14 Organizing and running code

During the class, we will write a lot of code, of increasing complexity. Here is what you should do to ensure that your programs are well-organized, easy to understand, and easy to debug.

1. Take the problem, and divide it into its basic building blocks. Each block should be its own function.
2. Write the code for each building block separately, and test it thoroughly.
3. Extensively document the code, so that you can understand what you did, how you did it, and why.
4. Combine the building blocks into a master program.

For example, let's write code that takes the data on Chromosome 6 we have seen above, and tries to identify which SNPs deviate the most from Hardy-Weinberg equilibrium. Remember that in an infinite population, where mating is random, there is no selection and no mutations, the proportion of people carrying the alleles $A1A1$ should be approximately $p_{11} = p^2$ (where p is the frequency of the first allele in the population $p = p_{11} + \frac{1}{2}p_{12}$), those carrying $A1A2$ should be $p_{12} = 2pq$ (where $q = 1 - p$) and finally those carrying $A2A2$ should be $p_{22} = q^2$. This is called the Hardy-Weinberg equilibrium.

We want to test this on a number of different SNPs. First, we write a function that takes as input the data and a given SNP, and computes the probability p of carrying the first allele.

```
compute_probabilities_HW <- function(my_data, my_SNP = "rs1535053"){
  # Take a SNP and compute the probabilities
  # p = frequency of first allele
  # q = frequency of second allele (1 - p)
  # p11 = proportion homozygous first allele
  # p12 = proportion heterozygous
  # p22 = proportion homozygous second allele
  my_SNP_data <- my_data[my_data$SNP == my_SNP,]
  AA <- my_SNP_data$nA1A1
  AB <- my_SNP_data$nA1A2
  BB <- my_SNP_data$nA2A2
  tot_observations <- AA + AB + BB
  p11 <- AA / tot_observations
  p12 <- AB / tot_observations
  p22 <- BB / tot_observations
  p <- p11 + p12 / 2
  q <- 1 - p
  return(list(SNP = my_SNP,
              p11 = p11,
              p12 = p12,
              p22 = p22,
              p = p,
              q = q,
              tot = tot_observations,
              AA = AA,
              AB = AB,
              BB = BB))
}
```

Now we can test our function:

```
compute_probabilities_HW(ch6)
```

```
$SNP  
[1] "rs1535053"
```

```
$p11  
[1] 0.04032258
```

```
$p12  
[1] 0.3145161
```

```
$p22  
[1] 0.6451613
```

```
$p  
[1] 0.1975806
```

```
$q  
[1] 0.8024194
```

```
$tot  
[1] 124
```

```
$AA  
[1] 5
```

```
$AB  
[1] 39
```

```
$BB  
[1] 80
```

If the allele conformed to Hardy-Weinberg, we should find approximately $p^2 \cdot n$ people with $A1A1$, where n is the number of people sampled. Let's see whether these assumptions are met by the data:

```
observed_vs_expected_HW <- function(SNP_data){  
  # compute expectations under Hardy-Weinberg equilibrium  
  # organize expected and observed in a table  
  observed <- c("AA" = SNP_data$AA, "AB" = SNP_data$AB, "BB" = SNP_data$BB)  
  expected <- c("AA" = SNP_data$p^2 * SNP_data$tot,
```

```

    "AB" = 2 * SNP_data$p * SNP_data$q * SNP_data$tot,
    "BB" = SNP_data$q^2 * SNP_data$tot)
return(rbind(observed, expected))
}

```

And test it:

```

my_SNP_data <- compute_probabilities_HW(ch6)
observed_vs_expected_HW(my_SNP_data)

```

	AA	AB	BB
observed	5.000000	39.000000	80.000000
expected	4.840726	39.31855	79.84073

Pretty good! This SNP seems very close to the theoretical expectation.

Let's try another one

```

observed_vs_expected_HW(compute_probabilities_HW(ch6, "rs1316662"))

```

	AA	AB	BB
observed	26.000000	62.000000	36.000000
expected	26.20161	61.59677	36.20161

Because we have so many SNPs, we will surely find some that do not comply with the expectation. For example:

```

my_SNP_data <- compute_probabilities_HW(ch6, "rs6596835")
observed_vs_expected_HW(my_SNP_data)

```

	AA	AB	BB
observed	17.000000	24.0000	82.0000
expected	6.837398	44.3252	71.8374

To find those with the largest deviations, we can compute for the statistic:

$$\sum_i \frac{(e_i - o_i)^2}{e_i}$$

In genetics, this is called χ^2 statistics, because if the data were to follow the assumptions, these quantities would follow the χ^2 distribution.

```

compute_chi_sq_stat <- function(my_obs_vs_expected){
  observed <- my_obs_vs_expected["observed",]
  expected <- my_obs_vs_expected["expected",]
  return(sum((expected - observed)^2 / expected))
}

```

Now let's compute the statistic for each SNPs:

```

# because this might take a while, we're going to only analyze the first 1000 SNPs
all_SNPs <- ch6$SNP[1:1000]
results <- data.frame(SNP = all_SNPs, ChiSq = 0)
for (i in 1:nrow(results)){
  results[i, 2] <- compute_chi_sq_stat(observed_vs_expected_HW(compute_probabilities_HW(ch
} 

```

To find the ones with the largest discrepancy, run

```

results <- results[order(results$ChiSq, decreasing = TRUE),]
head(results)

```

	SNP	ChiSq
10	rs2281351	53.993853
221	rs1933650	27.724832
36	rs6596835	25.862675
681	rs689035	9.802277
178	rs6930805	9.491511
179	rs1737539	9.491511

This example showed how a seemingly difficult problem can be decomposed in smaller problems that are easier to solve.

1.15 Documenting the code using knitr

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do.

Donald E. Knuth, Literate Programming, 1984

When doing experiments, we typically keep track of everything we do in a laboratory notebook, so that when writing the manuscript, or responding to queries, we can go back to our documentation to find exactly what we did, how we did it, and possibly why we did it. The same should be true for computational work.

RStudio makes it very easy to build a computational laboratory notebook. First, create a new R Markdown file (choose **File -> New File -> R Markdown** from the menu).

The gist of it is that you write a text file (.Rmd). The file is then read by an interpreter that transforms it into an .html or .pdf file, or even into a Word document. You can use special syntax to render the text in different ways. For example, type

```
*****  
  
*Test* **Test2**  
  
# Very large header  
  
## Large header  
  
### Smaller header  
  
## Unordered lists  
  
* First  
* Second  
    + Second 1  
    + Second 2  
  
1. This is  
2. A numbered list  
  
You can insert `inline code`  
  
-----
```

The most important feature of R Markdown, however, is that you can include blocks of code, and they will be interpreted and executed by R. You can therefore combine effectively the code itself with the description of what you are doing.

For example, including

```
{r, eval=FALSE} print("hello world!")
```

will become

```
print("hello world!")
```

```
[1] "hello world!"
```

If you don't want to run the R code, but just display it, use `{r, eval = FALSE}`; if you want to show the output but not the code, use `{r, echo = FALSE}`.

You can include plots, tables, and even render equations using LaTeX. In summary, when exploring your data or writing the methods of your paper, give R Markdown a try!

You can find inspiration in the notes for this class: all are written in R Markdown.

1.16 Resources

There are very many excellent books and tutorials you can read to become a proficient programmer in R. For example:

- [Intro to R](#)
- [Advanced R](#)
- [DataCamp](#)
- [ComputerWorld](#)
- [R Style guide](#)
- [R for Data Science](#)
- [RStudio Cheat Sheet](#)
- [Base R Cheat Sheet](#)
- [Advanced R Cheat Sheet](#)
- [X in Y minutes](#)
- [Intro to Data Wrangling](#)
- [R Boot Camp](#)

2 Visualizing data using `ggplot2`

2.1 Goal

Introduce the package `ggplot2`, which is part of the `tidyverse` bundle. Learn how to use `ggplot2` to produce publication-quality figures. Discuss the philosophical underpinnings of the “Grammar of Graphics”, showcase the `ggplot2` syntax, produce examples of the different types of graphs. Learn how to change colors, legends, scales. Visualize histograms, barplots, scatterplots, etc.

2.2 Introduction to the Grammar of Graphics

The most salient feature of scientific graphs should be clarity. Each figure should make crystal-clear a) what is being plotted; b) what are the axes; c) what do colors, shapes, and sizes represent; d) the message the figure wants to convey. Each figure is accompanied by a (sometimes long) caption, where the details can be explained further, but the main message should be clear from glancing at the figure (often, figures are the first thing editors and referees look at).

Many scientific publications contain very poor graphics: labels are missing, scales are unintelligible, there is no explanation of some graphical elements. Moreover, some color graphs are impossible to understand if printed in black and white, or difficult to discern for color-blind people.

Given the effort that you put into your science, you want to ensure that it is well presented and accessible. The investment to master some plotting software will be rewarded by pleasing graphics that convey a clear message.

In this section, we introduce `ggplot2`, a plotting package for R. This package was developed by Hadley Wickham who contributed many important packages to R (all included in the `tidyverse` bundle we’re going to use for the remainder of the class). Unlike many other plotting systems, `ggplot2` is deeply rooted in a “philosophical” vision. The goal is to conceive a grammar for all graphical representation of data. Leland Wilkinson and collaborators proposed The Grammar of Graphics. It follows the idea of a well-formed sentence that is composed of a subject, a predicate, and an object. The Grammar of Graphics likewise aims at describing

a well-formed graph by a grammar that captures a very wide range of statistical and scientific graphics. This might be more clear with an example – Take a simple two-dimensional scatterplot. How can we describe it? We have:

- **Data** The data we want to plot.
- **Mapping** What part of the data is associated with a particular visual feature? For example: Which column is associated with the x-axis? Which with the y-axis? Which column corresponds to the shape or the color of the points? In `ggplot2` lingo, these are called *aesthetic mappings* (`aes`).
- **Geometry** Do we want to draw points? Lines? In `ggplot2` we speak of *geometries* (`geom`).
- **Scale** Do we want the sizes and shapes of the points to scale according to some value? Linearly? Logarithmically? Which palette of colors do we want to use?
- **Coordinate** We need to choose a coordinate system (e.g., Cartesian, polar).
- **Faceting** Do we want to produce different panels, partitioning the data according to one (or more) of the variables?

This basic grammar can be extended by adding statistical transformations of the data (e.g., regression, smoothing), multiple layers, adjustment of position (e.g., stack bars instead of plotting them side-by-side), annotations, and so on.

Exactly like in the grammar of a natural language, we can easily change the meaning of a “sentence” by adding or removing parts. Also, it is very easy to completely change the type of geometry if we are moving from say a histogram to a boxplot or a violin plot, as these types of plots are meant to describe one-dimensional distributions. Similarly, we can go from points to lines, changing one “word” in our code. Finally, the look and feel of the graphs is controlled by a theming system, separating the content from the presentation.

2.3 Basic `ggplot2`

`ggplot2` ships with a simplified graphing function, called `qplot`. In this introduction we are not going to use it, and we concentrate instead on the function `ggplot`, which gives you complete control over your plotting. First, we need to load the package:

```
library(tidyverse)
```

To explore the features of `ggplot2`, we are going to use a data set detailing the total number of COVID cases and deaths in US counties. The data are provided by the [New York Times](#).

```

# read the data
# original URL https://github.com/nytimes/covid-19-data/raw/master/live/us-counties.csv
dt <- read_csv("https://rb.gy/zr65gg")

Rows: 3257 Columns: 6
-- Column specification -----
Delimiter: ","
chr (3): county, state, fips
dbl (2): cases, deaths
date (1): date

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

```

```
head(dt)
```

```

# A tibble: 6 x 6
  date      county    state        fips   cases  deaths
  <date>    <chr>     <chr>       <chr>  <dbl>   <dbl>
1 2023-03-24 McPherson South Dakota 46089    534     16
2 2023-03-24 Meade      South Dakota 46093   8404     68
3 2023-03-24 Mellette   South Dakota 46095    654      8
4 2023-03-24 Miner     South Dakota 46097    542     15
5 2023-03-24 Jennings  Indiana     18079   8178    119
6 2023-03-24 Johnson   Indiana     18081  51093    664

```

we are going to work with `date`, `county`, `state`, `cases` and `deaths`.

Let's select Illinois, and take only the counties with more than 10k cases (to have a less crowded graph):

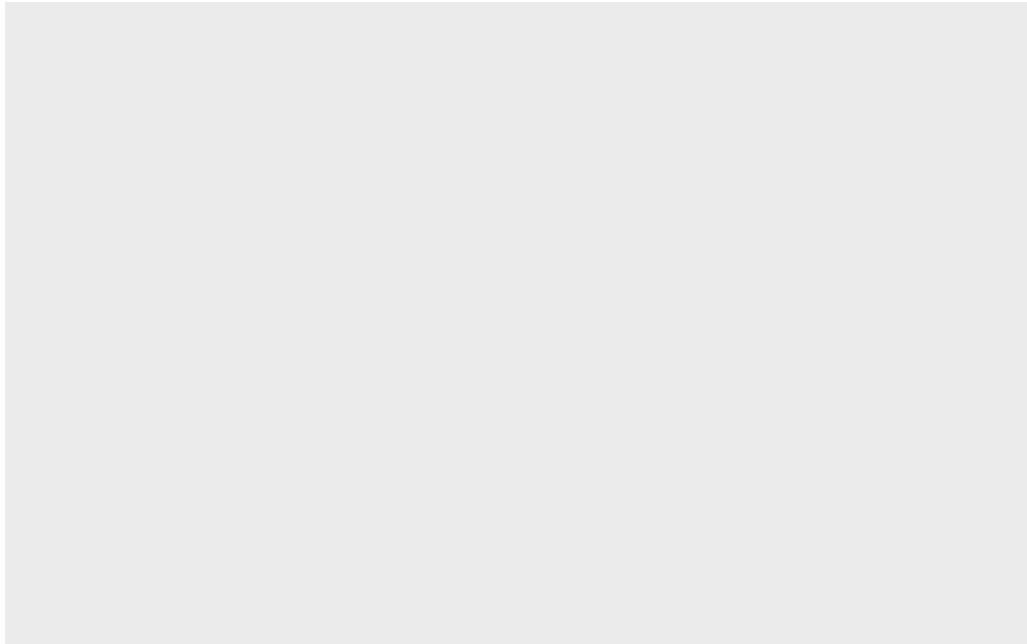
```
dti <- dt[(dt$state == "Illinois") & (dt$cases > 10^4), ]
```

A particularity of `ggplot2` is that it accepts exclusively data organized in tables (a `data.frame` or a `tibble` object—more on tibbles later). Thus, all of your data needs to be converted into a data frame format for plotting.

2.4 Building a well-formed graph

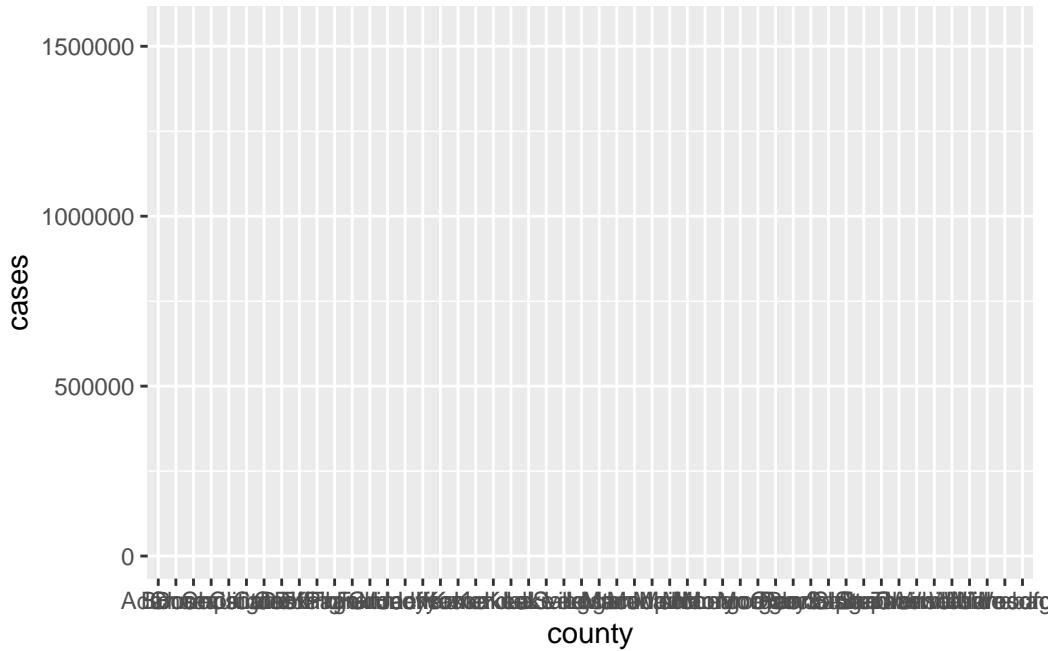
For our first plot, we're going to produce a barplot detailing how many cases have been reported in each County:

```
ggplot(data = dti)
```



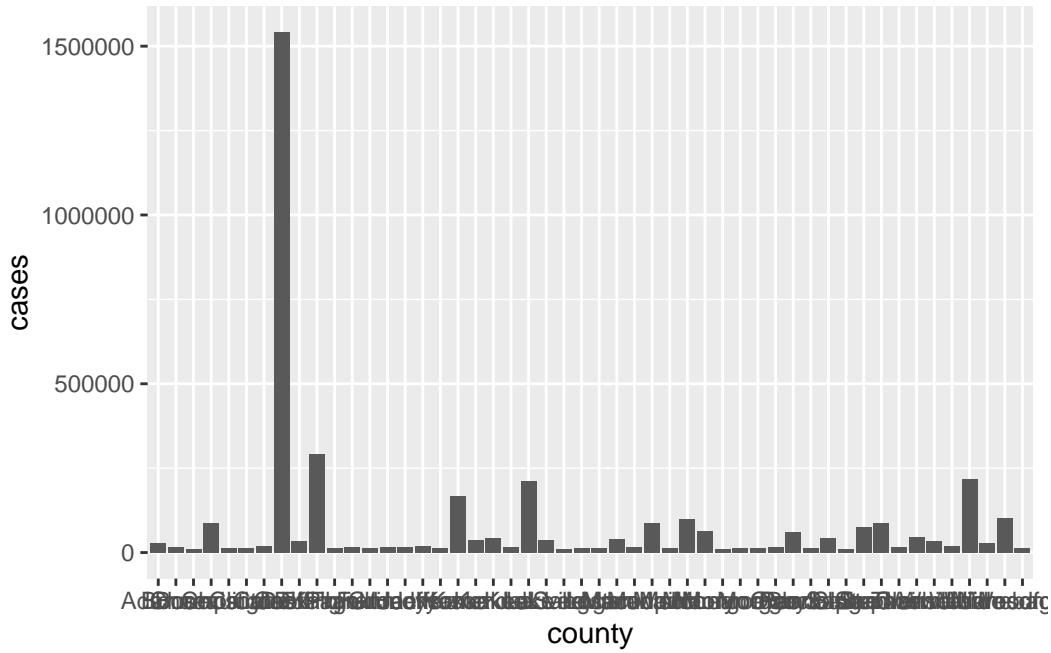
As you can see, nothing is drawn: we need to specify what we would like to associate to the x axis, and what to the y axis, etc. (i.e., we want to set as the *aesthetic mappings*). A barplot typically has classes on the x axis, while the y axis reports the counts in each class.

```
ggplot(data = dti) + aes(x = county, y = cases)
```



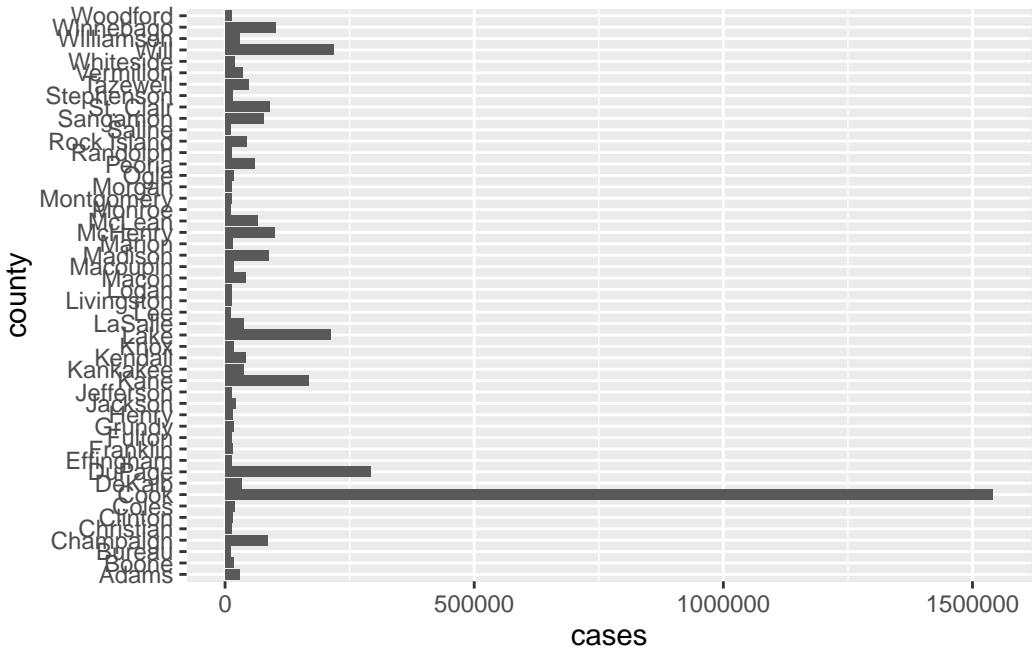
Note that we concatenate pieces of our “sentence” using the `+` sign! We’ve got the aesthetic mappings figured out, but still no graph... we need to specify a geometry, i.e., the type of graph we want to produce. In this case, a barplot where the height of the bars is specified by the `y` value:

```
ggplot(data = dti) + aes(x = county, y = cases) + geom_col()
```



Because it is very difficult to see the labels, let's swap the axes:

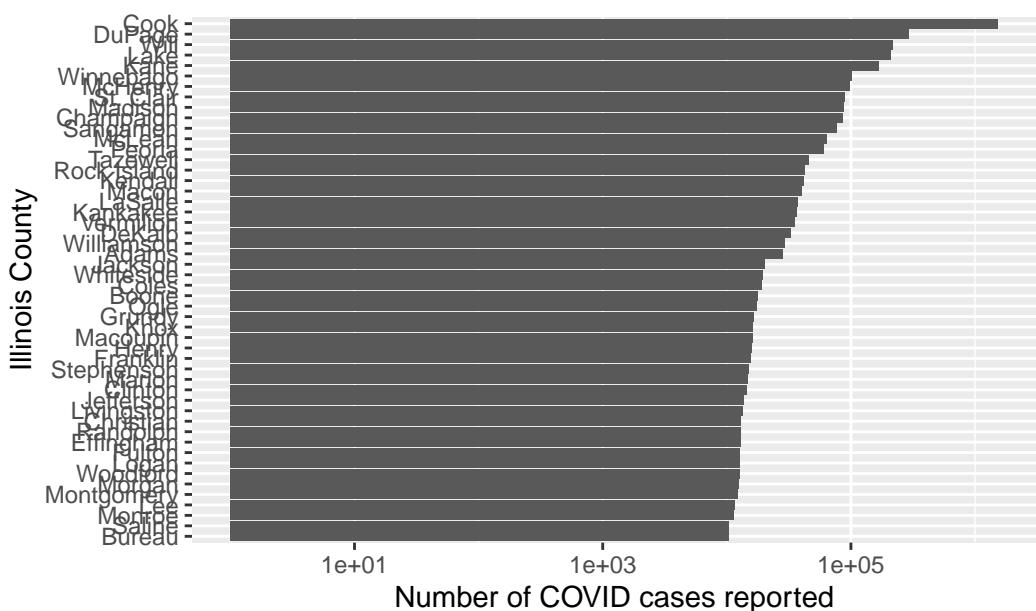
```
ggplot(data = dti) +  
  aes(x = county, y = cases) +  
  geom_col() +  
  coord_flip()
```



The graph shows that, naturally, the vast majority of cases was reported in Cook county. We have written a “well-formed sentence”, composed of **data + mapping + geometry**, and this is sufficient to produce a graph. We can add “adjectives” and “adverbs” to our graph, to make it clearer:

```
ggplot(data = dti) +
  aes(x = reorder(county, cases), y = cases) + # order labels according to cases
  geom_col() +
  ylab("Number of COVID cases reported") + # x label
  xlab("Illinois County") + # y label
  scale_y_log10() + # transform the counts to logs
  coord_flip()+
  ggtitle(dti$date[1]) # main title (use current date)
```

2023–03–24



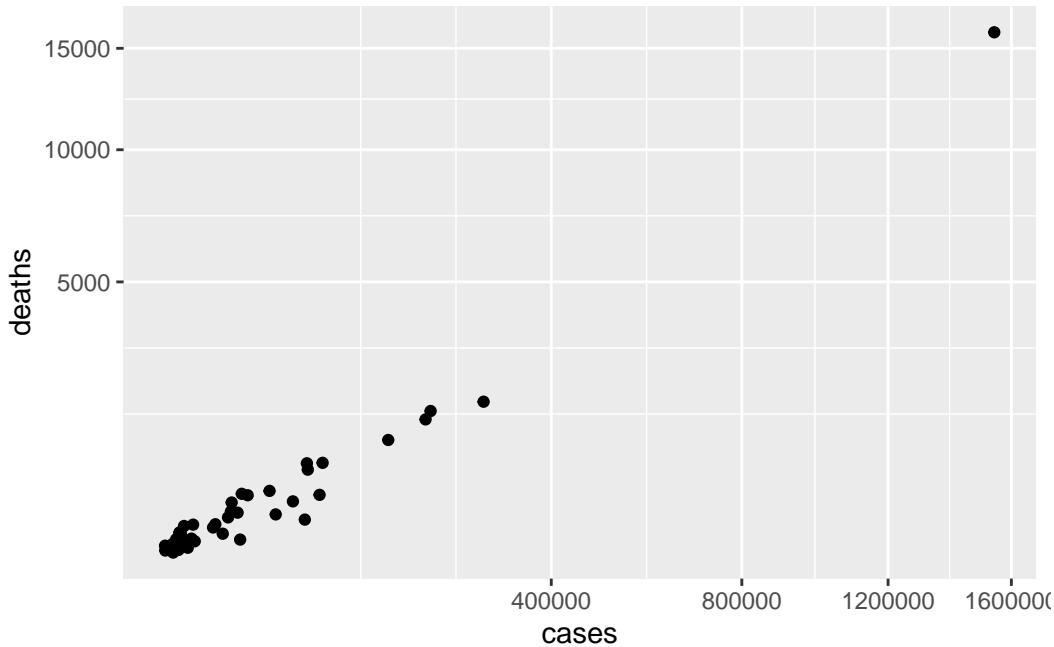
2.5 Scatterplots

Using `ggplot2`, one can produce very many types of graphs. The package works very well for 2D graphs (or 3D rendered in two dimensions), while it lacks capabilities to draw proper 3D graphs, or networks.

The main feature of `ggplot2` is that you can tinker with your graph fairly easily, and with a common grammar. You don't have to settle on a certain presentation of the data until you're ready, and it is very easy to switch from one type of graph to another.

For example, let's plot the number of cases vs. number of deaths:

```
# you can store the graph in a variable
pl <- ggplot(data = dti)
pl <- pl + aes(x = cases, y = deaths) # for a scatter plot, we need two aes mappings!
pl <- pl + geom_point() # draw points in a scatterplot
pl <- pl + scale_x_sqrt() + scale_y_sqrt() # transform axes
pl # or show(pl)
```

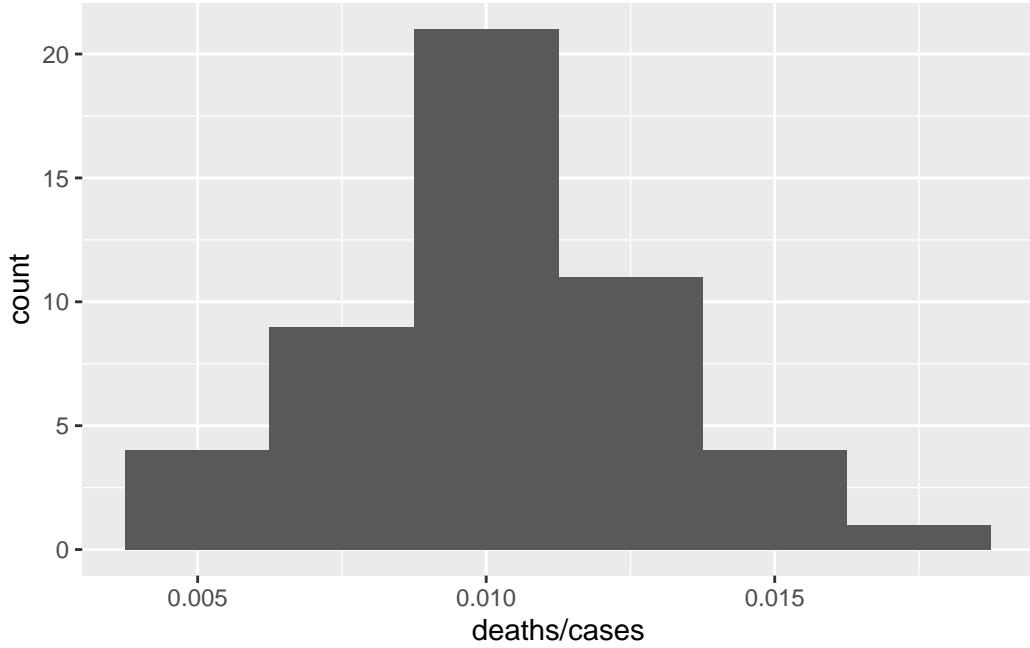


Showing that number of daily cases and number of daily deaths are highly correlated (but it would be a stronger correlation if we were to plot past cases vs. current deaths).

2.6 Histograms, density and boxplots

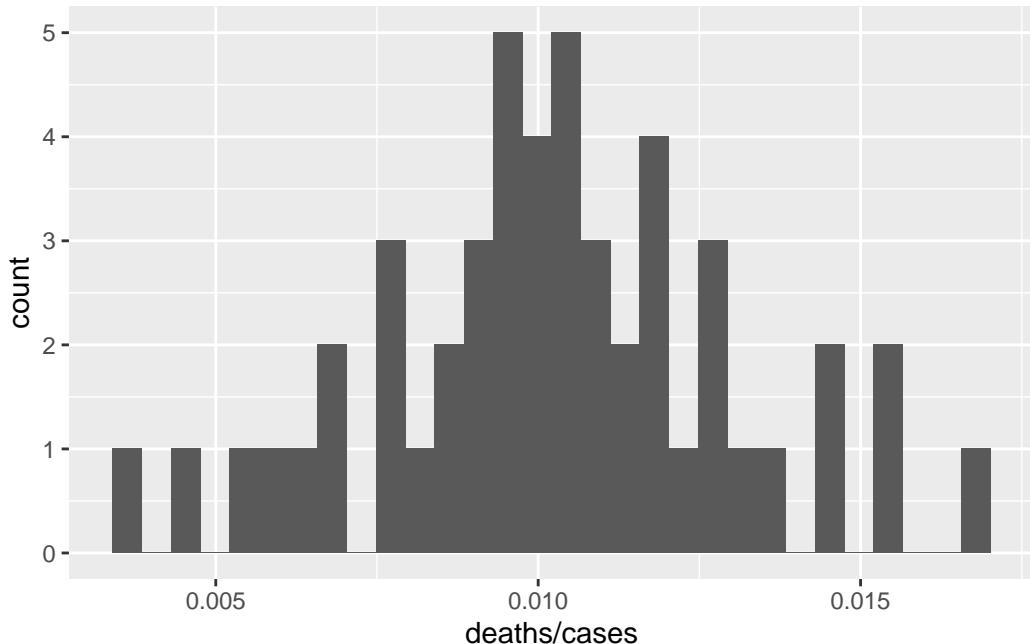
It would be nice to see the distribution of the ratio deaths/cases. To do so, we can produce a histogram:

```
pl <- ggplot(data = dti)
pl <- pl + aes(x = deaths / cases)
pl + geom_histogram(binwidth = 0.0025)
```

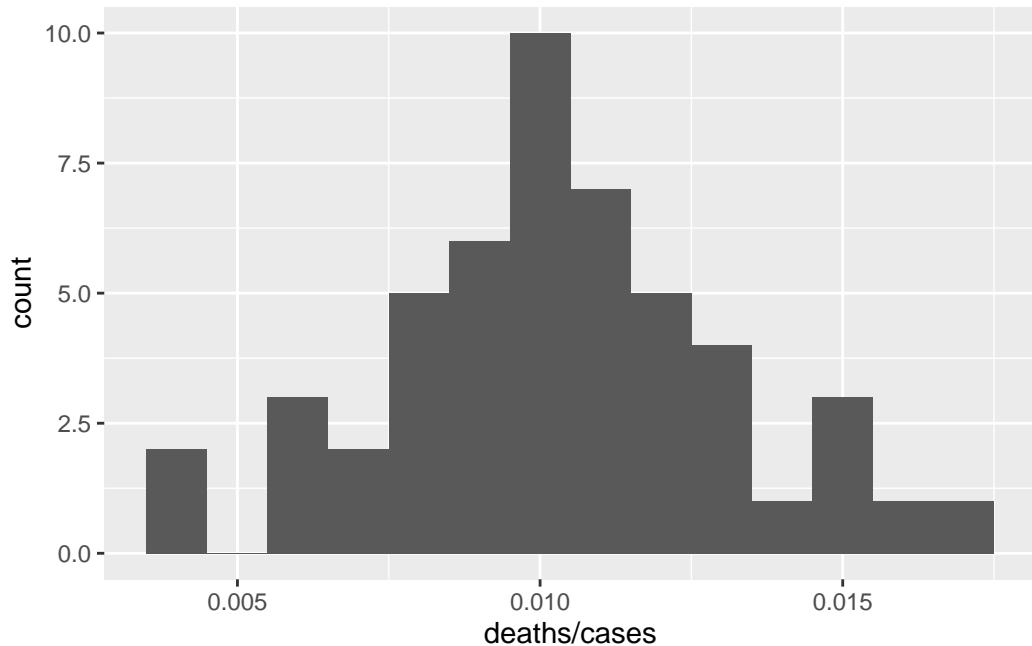


We can control the width of the bins by specifying:

```
pl + geom_histogram(bins = 30) # specify the number of bins
```

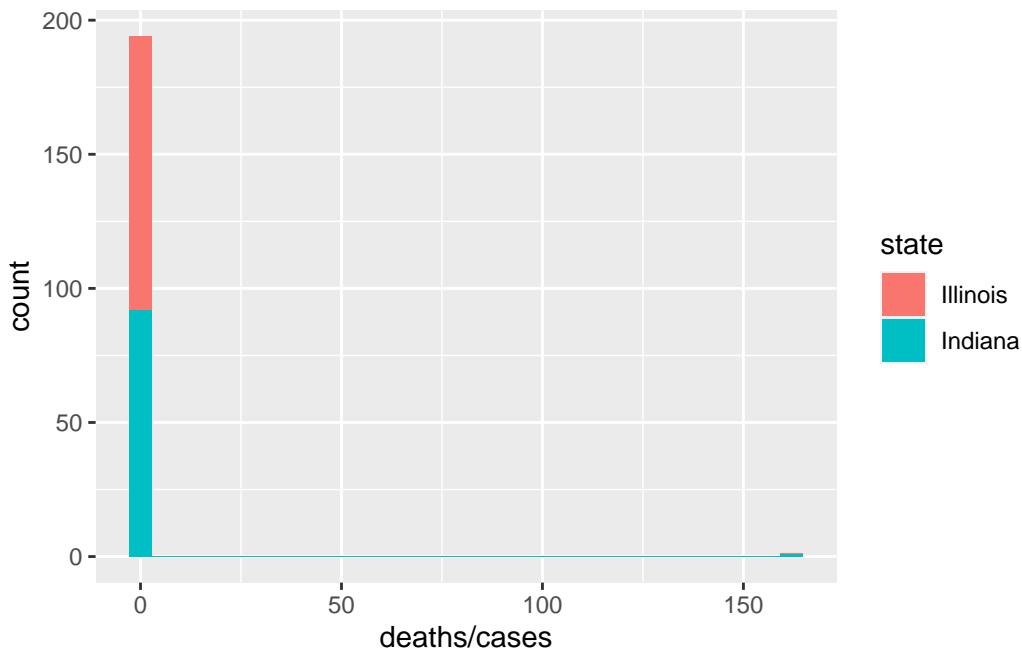


```
pl + geom_histogram(binwidth = 0.001) # specify the bin width
```



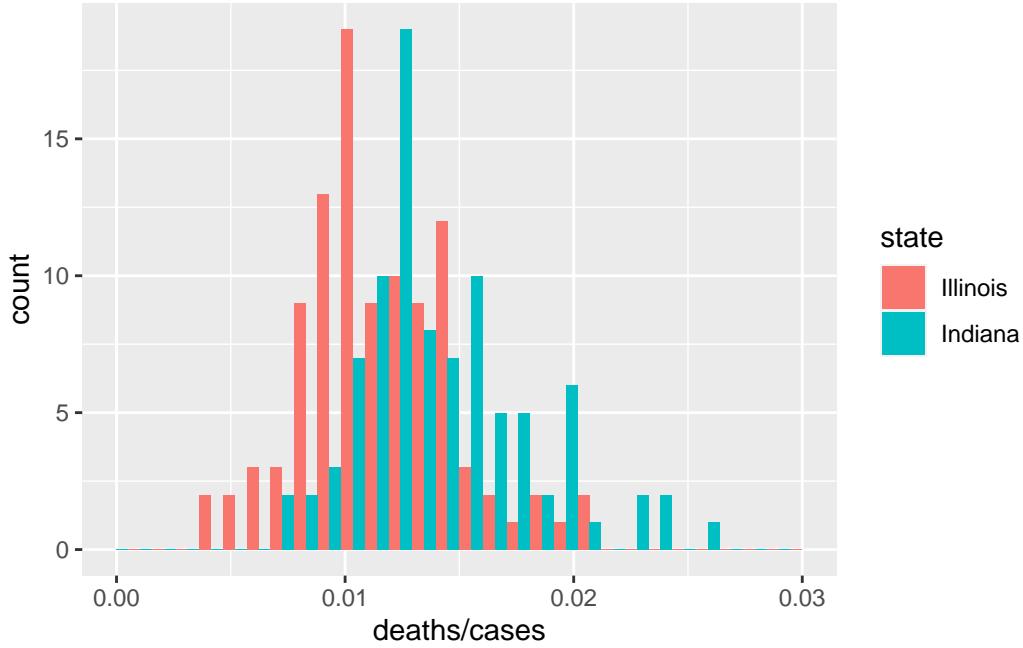
Let's see whether the histograms differ between Illinois and Indiana:

```
ggplot(data = dt[dt$state %in% c("Illinois", "Indiana"),]) +  
  aes(x = deaths / cases, fill = state) + # fill the bar colors by state  
  geom_histogram(bins = 30)
```



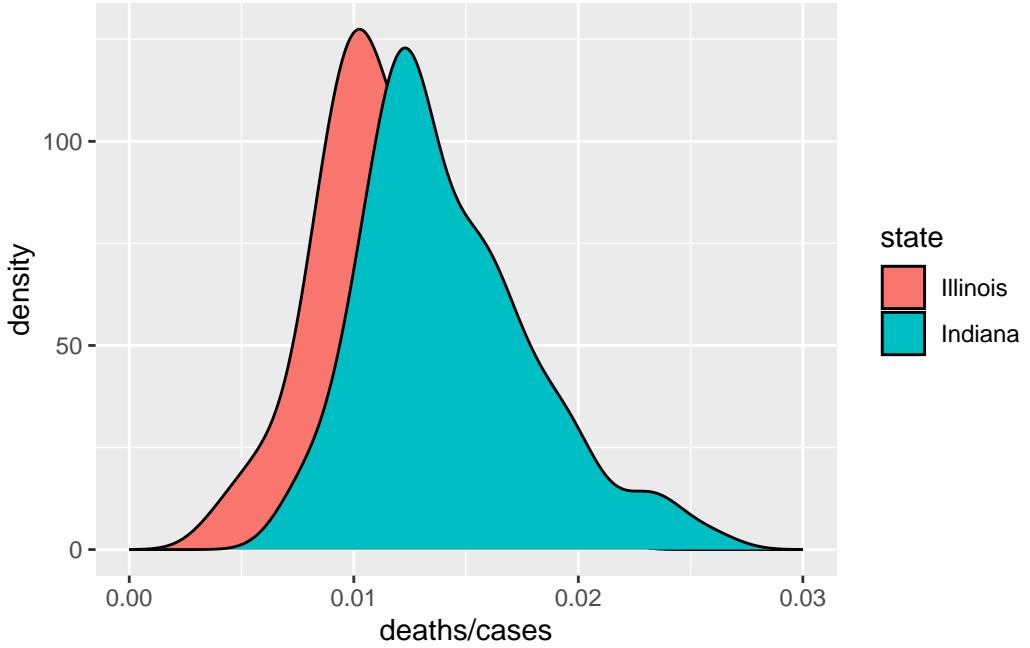
To plot the histogram side by side, use

```
ggplot(data = dt[dt$state %in% c("Illinois", "Indiana"),]) +  
  aes(x = deaths / cases, fill = state) + # fill the bar colors by state  
  geom_histogram(position = "dodge", bins = 30)+  
  xlim(c(0,0.03))
```



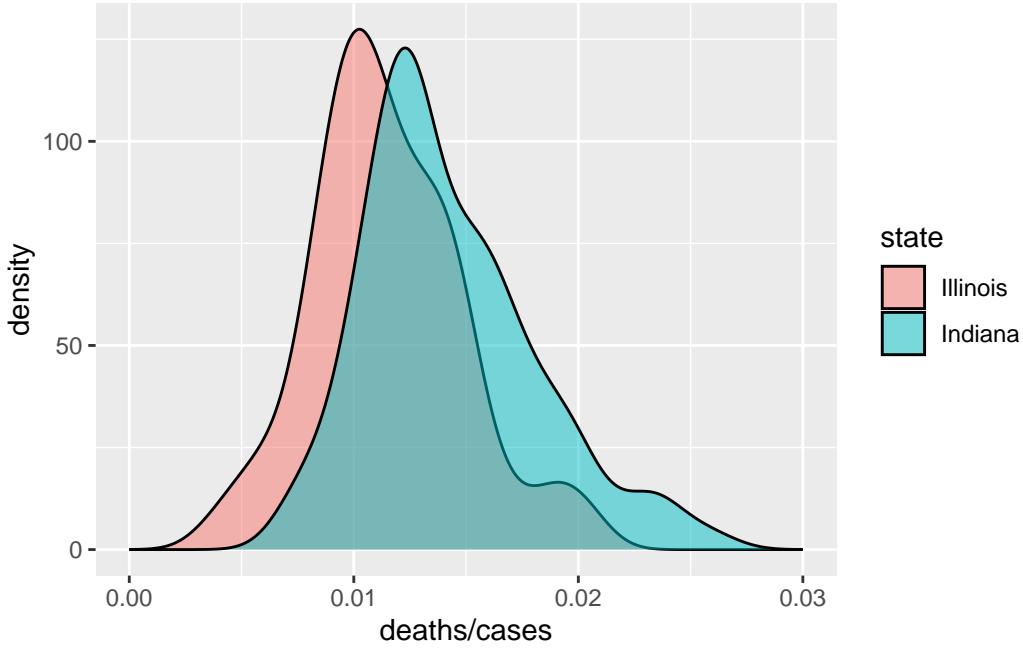
Similarly, we can approximate the histogram using a density plot, which interpolates the bin height to create a smooth distribution:

```
ggplot(data = dt[dt$state %in% c("Illinois", "Indiana"),]) +
  aes(x = deaths / cases, fill = state) + # fill by state
  geom_density() + xlim(c(0,0.03))
```



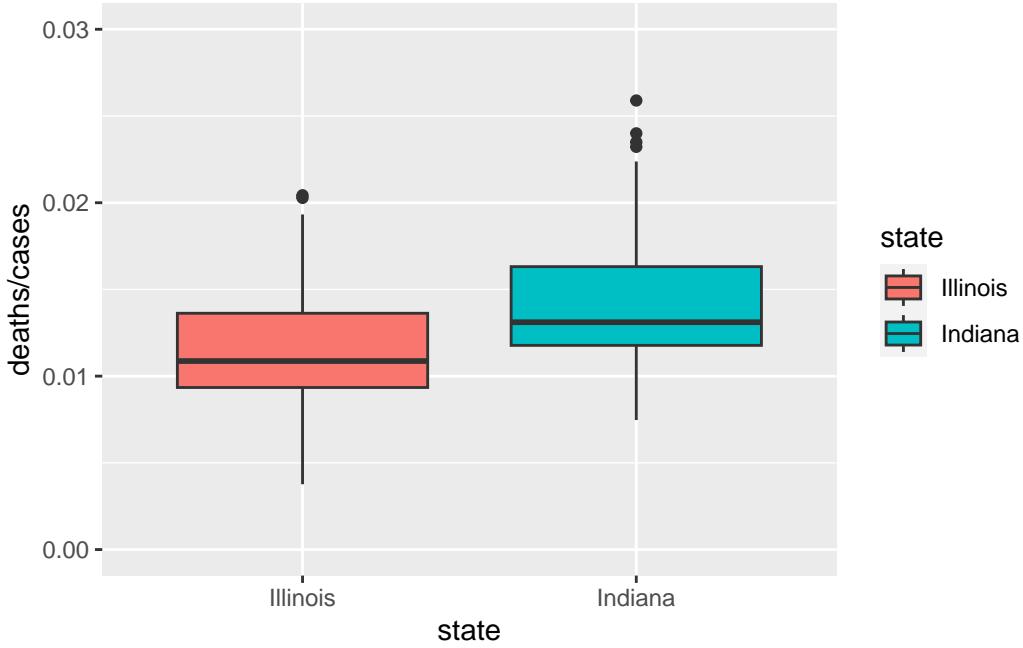
To see the graph better, let's make the coloring semi-transparent:

```
ggplot(data = dt[dt$state %in% c("Illinois", "Indiana"),]) +  
  aes(x = deaths / cases, fill = state) + # fill by state  
  geom_density(alpha = 0.5) + xlim(c(0, 0.03))
```



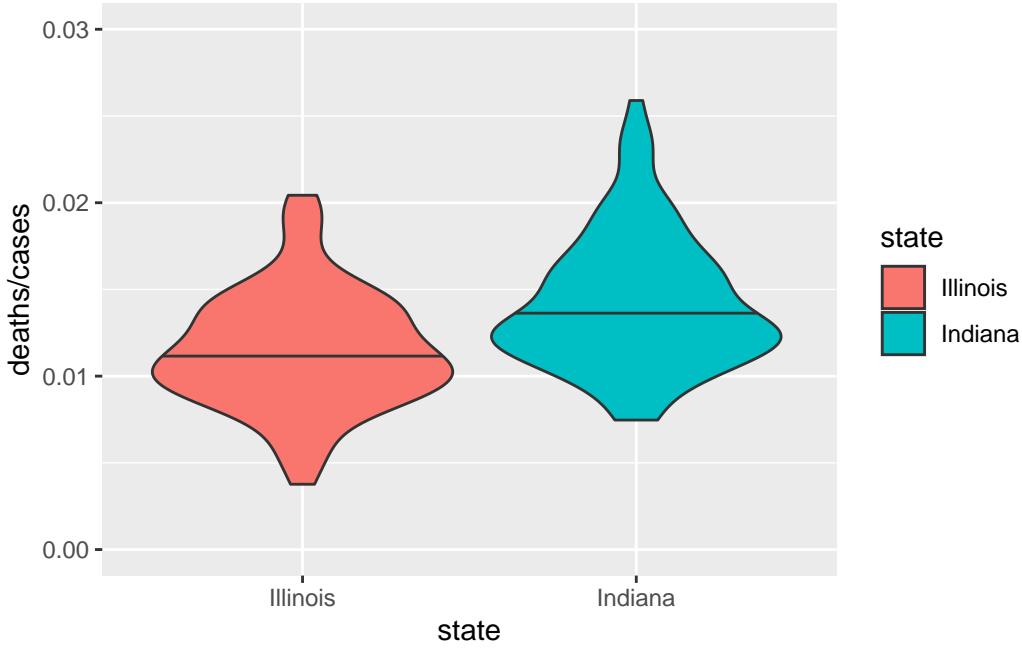
Showing a similar distribution for the death rate in the two states. For this type of comparison, the ideal graph to show is maybe a box-plot or a violin plot:

```
ggplot(data = dt[dt$state %in% c("Illinois", "Indiana"),]) +
  aes(x = state, y = deaths / cases, fill = state) + # we need both x and y
  geom_boxplot() + ylim(c(0, 0.03))
```



A boxplot shows the median (horizontal bar) as well as the inter-quartile range (box size goes from 25th to 75th percentile), as well as the typical range of the data (whiskers). The dots represent “outliers”. To show the full distribution, you can use a violin plot:

```
ggplot(data = dt[dt$state %in% c("Illinois", "Indiana"),]) +
  aes(x = state, y = deaths / cases, fill = state) + # we need both x and y
  geom_violin(draw_quantiles = 0.5) + ylim(c(0, 0.03))
```



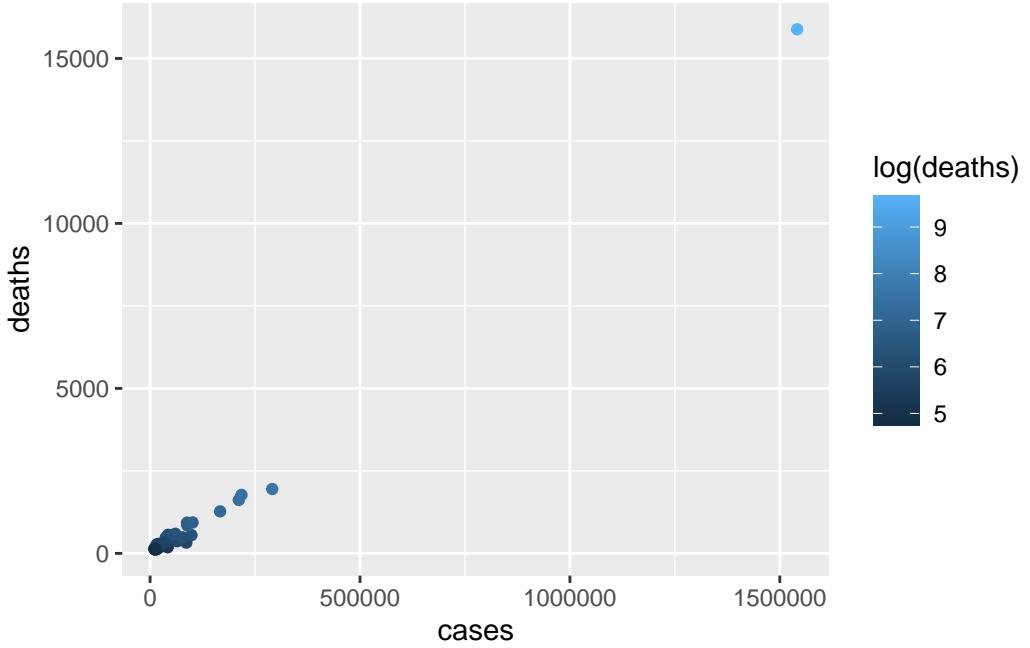
Note that when we're producing “similar” plots (e.g., histogram vs. density, box vs. violin, or any other plot sharing the same aesthetic mappings) changing a single word, we have changed the structure of the graph considerably!

2.7 Scales

We can use scales to determine how the aesthetic mappings are displayed. For example, we could set the *x* axis to be in logarithmic scale, or we can choose how the colors, shapes and sizes are used. `ggplot2` uses two types of scales: `continuous` scales are used for continuous variables (e.g., real numbers); `discrete` scales for variables that can only take a certain number of values (e.g., colors, shapes, sizes).

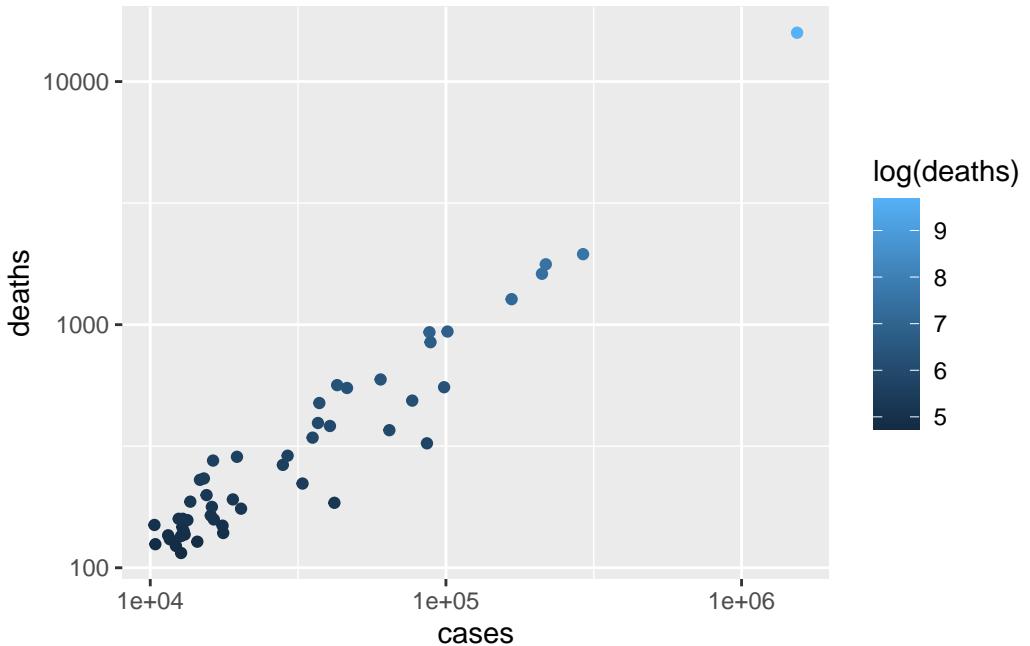
For example, let's plot deaths vs. cases in our `dti` data set:

```
pl <- ggplot(data = dti) +
  aes(x = cases, y = deaths, colour = log(deaths)) +
  geom_point()
pl
```

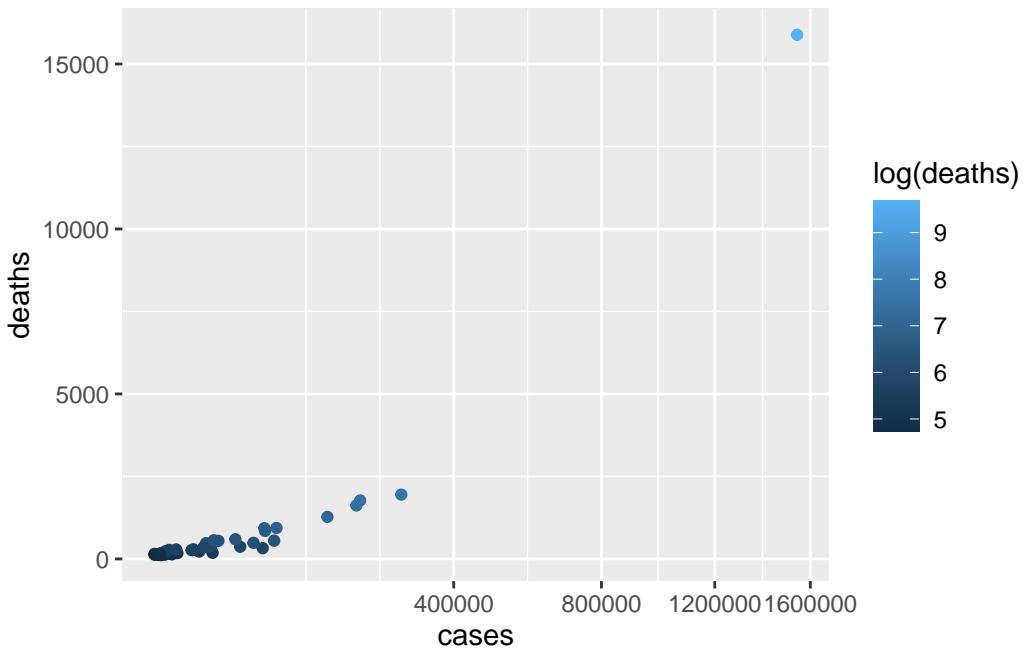


We can change the scale of the x axis by calling:

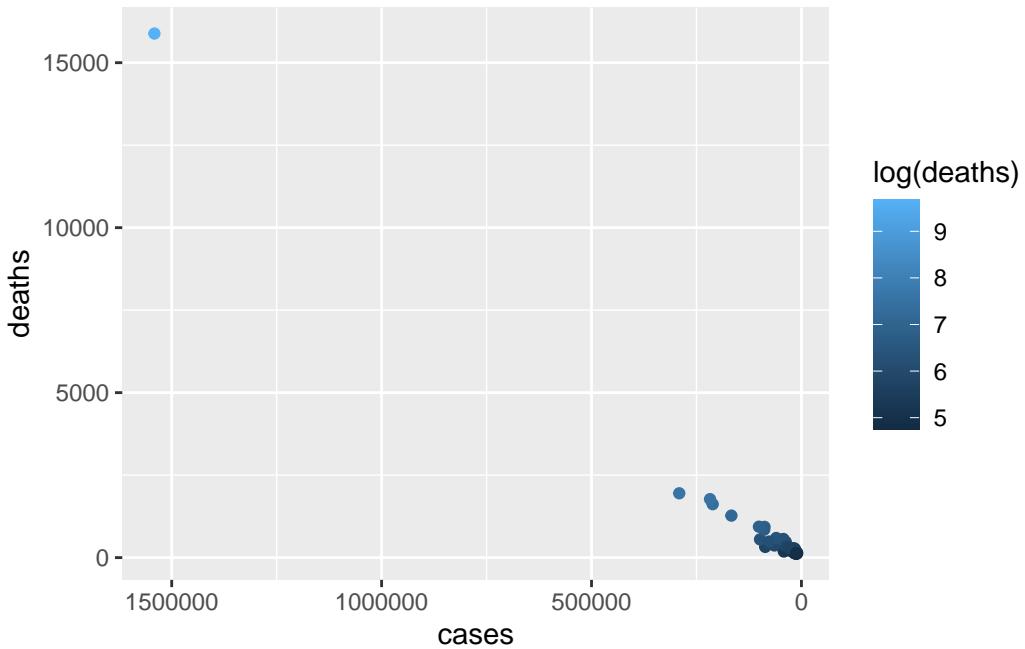
```
pl + scale_x_log10() + scale_y_log10() # log-log plot
```



```
pl + scale_x_sqrt() # sqrt of number of cases
```



```
pl + scale_x_reverse() # from large to small
```



Similarly, we can change the use of colors, points, etc.

2.8 List of aesthetic mappings

We've seen some of the aesthetic mappings. Here's a list of the main `aes`:

- `x` what to use for *x* axis
- `y` what to use for *y* axis
- `color` the color of points and lines
- `fill` the color of shapes (e.g., boxes, bars, etc.)
- `size` the size of points, lines, etc.
- `shape` the shape of points
- `alpha` the level of transparency of the object
- `linetype` the type of line (e.g., solid, dashed, etc.)

```
# a more complex example
ggplot(data = dt) +
  aes(x = cases, y = deaths,
      color = state) +
  geom_point() +
  scale_x_log10() + # note that the points with 0 cases or deaths will not work
  scale_y_log10() +
  theme(legend.position = "bottom")
```



2.9 List of geometries

There are very many geometries; here are a few of the most useful ones:

- Lines: `geom_abline` (line given slope and intercept); `geom_hline`, `geom_vline` (horizontal, vertical line); `geom_line` (connect observation in scatterplot).
- Bars: `geom_bar` (bar height is the count/sum); `geom_col` (bar heights are provided by the data).
- Boxes: `geom_boxplot`.
- Distributions: `geom_violin` (like boxplots, but showing the density of the distribution); `geom_density` (density of 1D distribution), `geom_density2d` (density of bivariate distribution); `geom_histogram`, `geom_bin2d` (histograms).
- Text: `geom_text`.
- Smoothing function: `geom_smooth` (interpolates the points of a scatterplot).
- Error bars: `geom_errorbar`.
- Maps: `geom_map` (polygons from a reference map).

2.10 List of scales

There are also very many scales. Here are a few:

- `xlab`, `ylab`, `xlim`, `ylim` control labels and ranges of the axes.
- `scale_alpha` transparency of the points/shapes.
- `scale_color` (many options) colors of points and lines.
- `scale_fill` (many options) colors of boxes, bars and shapes.
- `scale_shape` shape of the points.
- `scale_linetype` type of lines.
- `scale_size` size of points and lines.
- `scale_x`, `scale_y` (many options) transformations of the axes.

2.11 Themes

Themes allow you to manipulate the look and feel of a graph with just one command. The package `ggthemes` extends the themes collection of `ggplot2` considerably. For example:

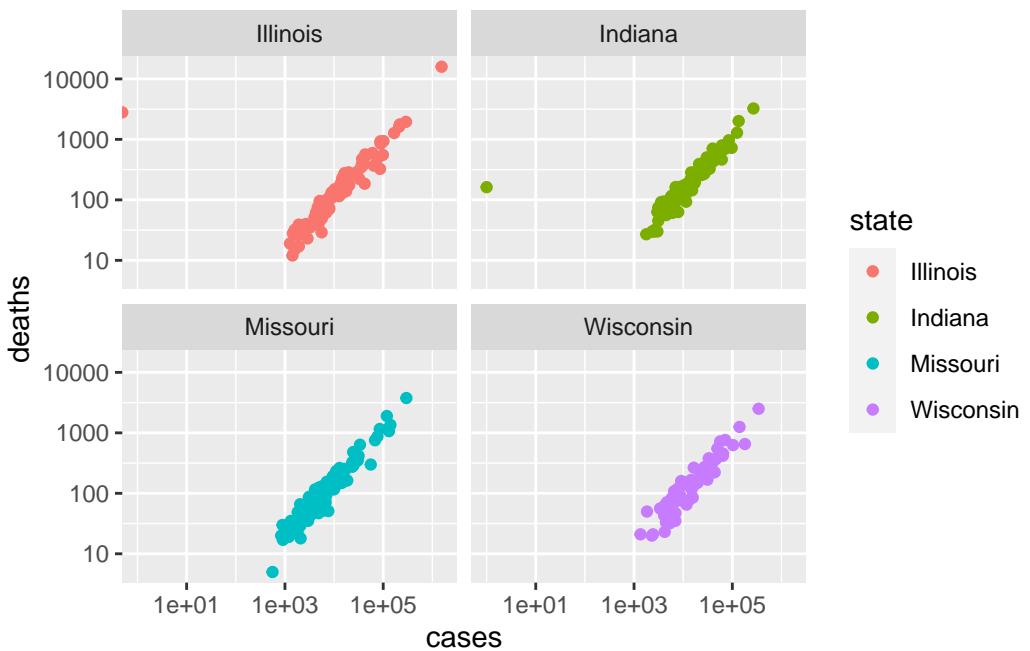
```
# to install, type install.packages("ggthemes") in the console
library(ggthemes)
pl <- ggplot(data = dti) + aes(x = cases, y = deaths) +
  geom_point() + scale_x_log10() + scale_y_log10()
pl + theme_bw() # white background
pl + theme_economist() # like in the magazine "The Economist"
pl + theme_wsj() # like "The Wall Street Journal"
```

2.12 Faceting

In many cases, we would like to produce a multi-panel graph, in which each panel shows the data for a certain combination of parameters. In `ggplot2` this is called *faceting*: the command `facet_grid` is used when you want to produce a grid of panels, in which all the panels in the same row (or column) have axes-ranges in common; `facet_wrap` is used when the different panels do not necessarily have axes-ranges in common.

For example:

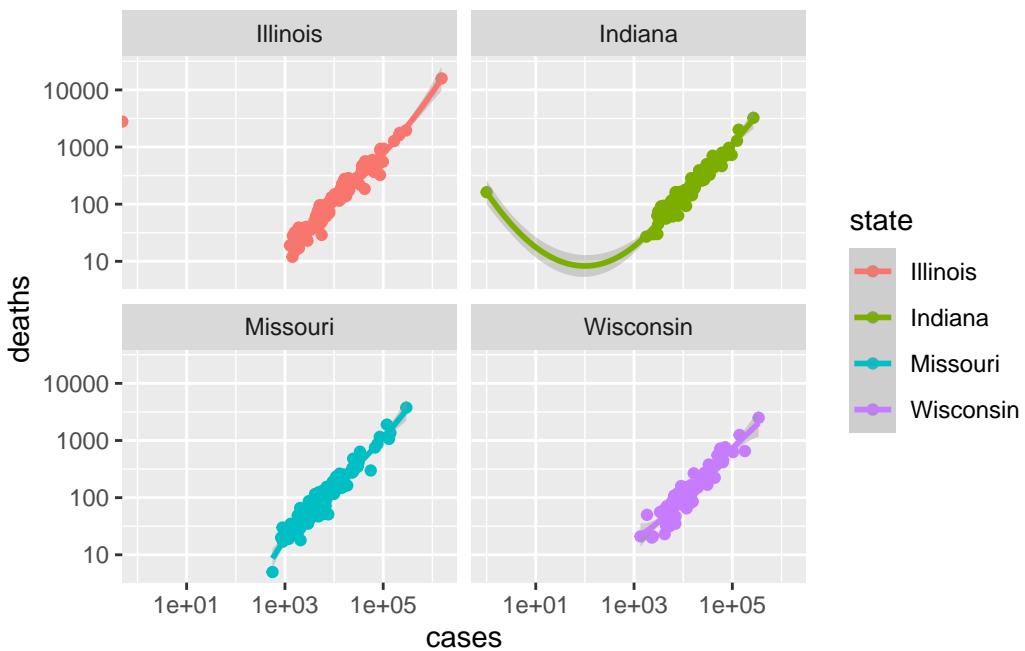
```
pl <- ggplot(data = dt[dt$state %in% c("Illinois", "Missouri", "Wisconsin", "Indiana"), ])
  aes(x = cases, y = deaths, colour = state) + geom_point() + scale_x_log10() + scale_y_log10()
pl <- pl + facet_wrap(~state)
pl
```



Let's add a line separating showing the best-fit line:

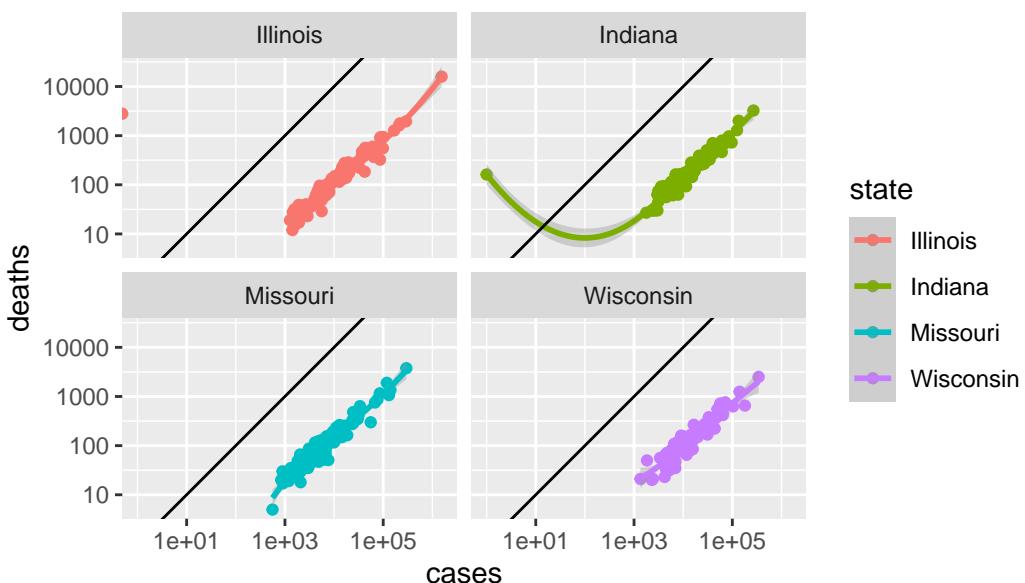
```
pl <- pl + geom_smooth()
pl

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Make ranges on x and y axes equal, and add the 1:1 line:

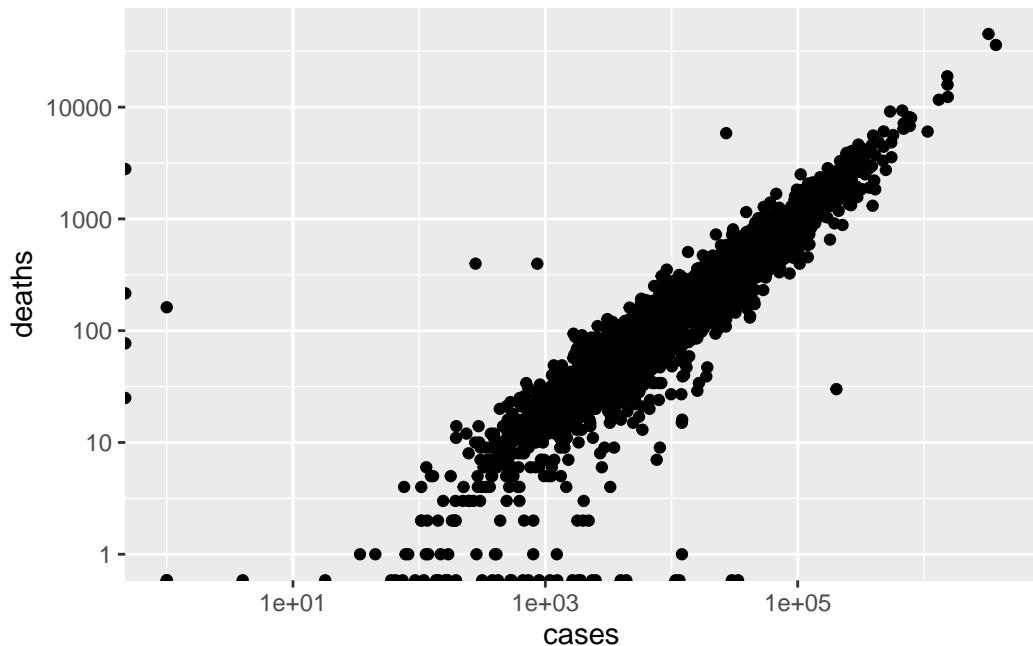
```
pl <- pl + coord_equal() + geom_abline(slope = 1, intercept = 0)
pl
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



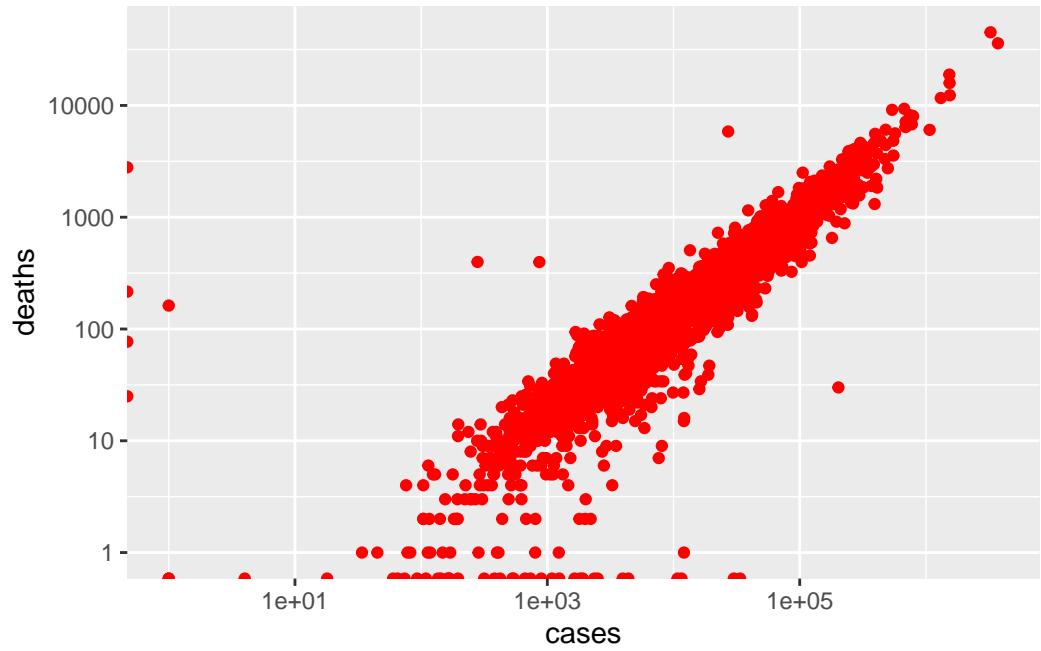
2.13 Setting features

Often, you want to simply set a feature (e.g., the color of the points, or their shape), rather than using it to display information (i.e., mapping some aesthetic). In such cases, simply declare the feature outside the `aes`:

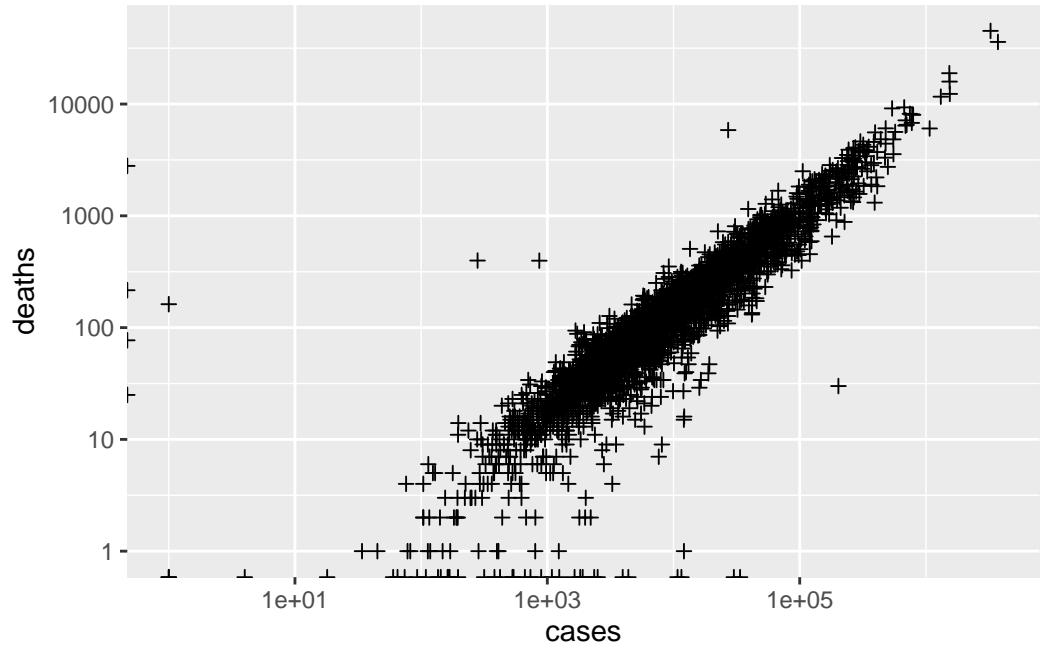
```
pl <- ggplot(data = dt) +  
  aes(x = cases, y = deaths) +  
  scale_x_log10() +  
  scale_y_log10()  
pl + geom_point()
```



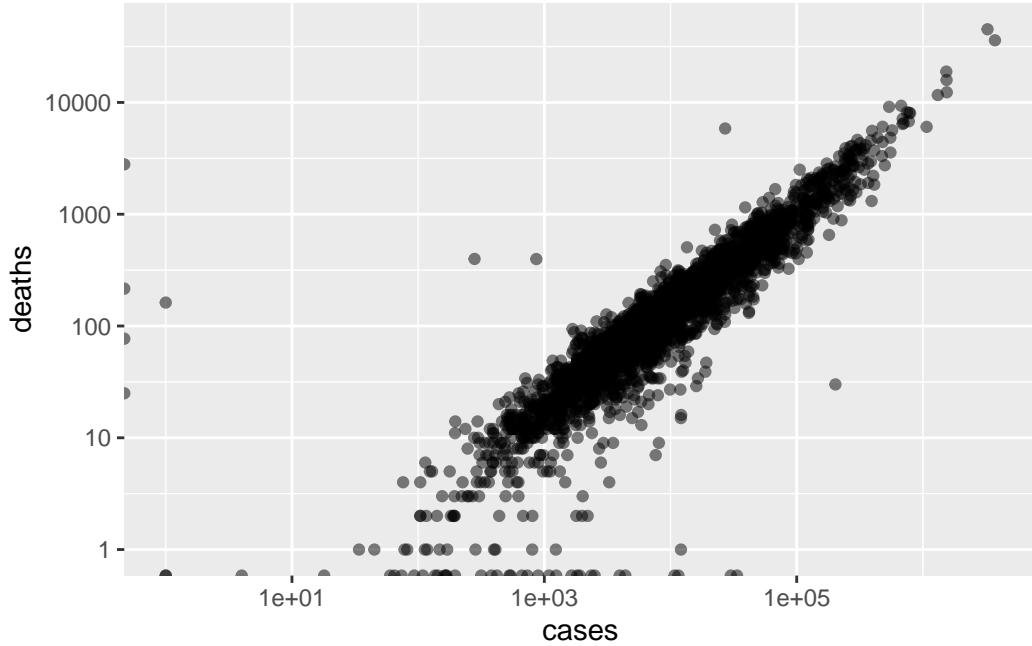
```
pl + geom_point(colour = "red")
```



```
pl + geom_point(shape = 3)
```



```
pl + geom_point(alpha = 0.5)
```



2.14 Saving graphs

You can either save graphs as done normally in R:

```
# save to pdf format
pdf("my_output.pdf", width = 6, height = 4)
print(my_plot)
dev.off()
# save to svg format
svg("my_output.svg", width = 6, height = 4)
print(my_plot)
dev.off()
```

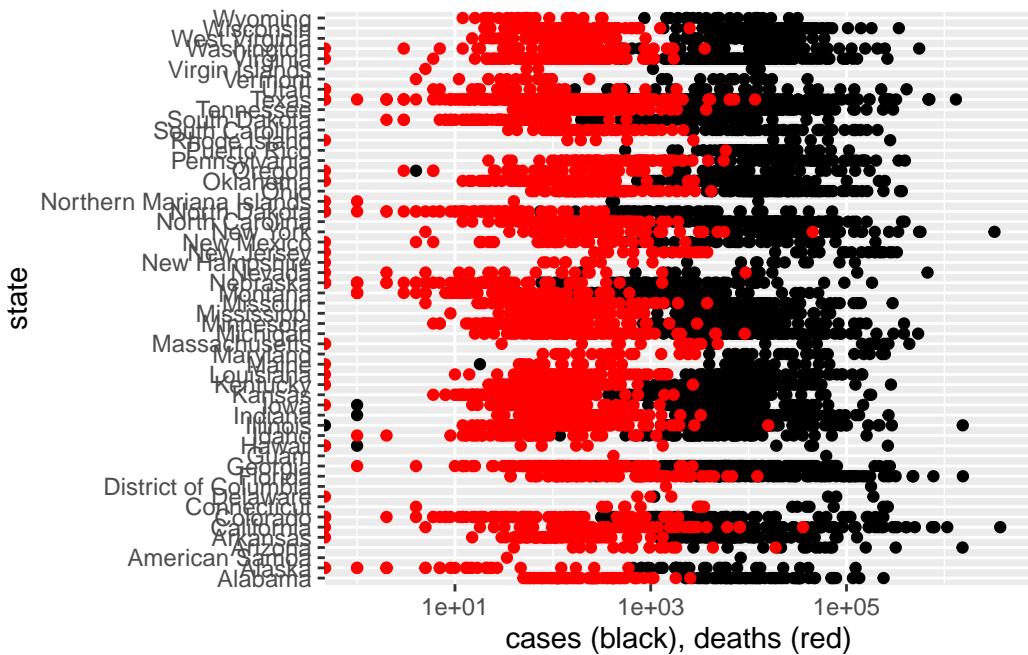
or use the function `ggsave`

```
# save current graph
ggsave("my_output.pdf")
# save a graph stored in ggplot object
ggsave(plot = my_plot, filename = "my_output.svg")
```

2.15 Multiple layers

You can overlay different plots. To do so, however, they must share some of the aesthetic mappings. The simplest case is that in which you have only one dataset:

```
ggplot(data = dt) +  
  geom_point(aes(y = state, x = cases), color = "black") +  
  geom_point(aes(y = state, x = deaths), color = "red") +  
  scale_x_log10() +  
  xlab("cases (black), deaths (red)")
```



2.16 Try on your own data!

Now that you're familiar with `ggplot2`, try producing some meaningful plots for your own data.

2.17 Resources

- [R for Data Science](#)
- [Tidyverse reference website](#)

- Data Visualization Cheat Sheet

3 Fundamentals of probability

3.1 Sample spaces and random variables

No observation or measurement in our world is perfectly reproducible, no matter how carefully planned and executed. The level of uncertainty varies, but randomness always finds a way to creep into a data set. Where does the “random” factor come from? From the classical physics perspective, as articulated by Laplace, most natural phenomena are theoretically deterministic for an omniscient being with an unlimited computational power. Quantum mechanical phenomena are (theoretically) truly random, but the randomness is not observable on the scales of biology or social science. The lack of predictability in the data we work with is usually due either to its intrinsic complexity (e.g., bio-molecular systems, prediction of animal behavior), which essentially makes it impossible to know every detail of the system, or to some external source of noise (e.g., measurement error, weather affecting food availability) that is outside of our control.

In probability terminology, a *random experiment* produces *outcomes* and the collection of all outcomes of an experiment is called its *sample space*.

Example: The specifics of the experiment can affect the degree of uncertainty in the outcome; the same measurement may be random or not, depending on context. For example, measuring the height of a person should be deterministic, if one measures the height of the same person within a short amount of time. So unless you’re interested in studying the error in [stadiometer](#) results, you probably won’t consider this a random experiment. However, measuring the heights of different people is a random experiment, where the source of randomness is primarily due to the selection of people for your study, called *sampling error*, rather than due to the measurement noise of any one person.

The measurement of interest from a random experiment is called a *random variable*. Sometimes the measurement is simply the outcome, but usually it reports some aspect of the outcome and so several outcomes can have the same value of the random variable. The random variable can then be seen as condensing the sample space into a smaller range of values. Random variables can be *numeric* or *categorical*, with the difference that categorical variables cannot be assigned meaningful numbers. For instance, one may report an individual by phenotype (e.g., white or purple flowers), or having a nucleotide A, T, G, C in a particular position, and although one could assign numbers to these categories (e.g., 1, 2, 3, 4) they could not be used in a sensible way—one can compare and do arithmetic with numbers, but A is not less than T and A + T

does not equal G. Thus there are different tools for describing and working with numeric and categorical random variables.

Example: In a DNA sequence a codon triplet represents a specific amino acid, but there is redundancy (several triplets may code for the same amino acid). One may think of a coding DNA sequence as an outcome, but the amino acid (sequence or single one) as a random variable. Extending this framework, one may think of genotype as an outcome, but a phenotype (e.g., eye color) as a random variable—although this is not correct for any phenotype that is not strictly determined by the genotype, because then there are other factors (e.g., environmental or epigenetic) that influence the value of the random variable besides the outcome (genotype).

Exercise: The package `palmerpenguins` contains multiple variables measured in populations of three different species of penguins over three years on three different islands. Identify numeric and categorical variables, and specify whether numeric variables are discrete and continuous.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr    1.3.0
v purrr    1.0.2
-- Conflicts -----
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become non-conflicting
```

```
library(palmerpenguins)
str(penguins)
```

```
tibble [344 x 8] (S3: tbl_df/tbl/data.frame)
$ species      : Factor w/ 3 levels "Adelie", "Chinstrap", ... : 1 1 1 1 1 1 1 1 1 ...
$ island       : Factor w/ 3 levels "Biscoe", "Dream", ... : 3 3 3 3 3 3 3 3 3 ...
$ bill_length_mm: num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
$ bill_depth_mm: num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
$ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
$ body_mass_g   : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
$ sex          : Factor w/ 2 levels "female", "male": 2 1 1 NA 1 2 1 2 NA NA ...
$ year         : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

3.2 Probability axioms

An outcome in sample space can be assigned a *probability* depending on its frequency of occurrence out of many trials, each is a number between 0 and 1. Combinations of outcomes (*events*) can be assigned probabilities by building them out of individual outcomes. These probabilities have a few rules, called the *axioms of probability*, expressed using set theory notation.

1. The total probability of all outcomes in sample space is 1. $P(\Omega) = 1$
2. The probability of nothing (empty set) is 0. $P(\emptyset) = 0$
3. The probability of an event made up of the union of two events is the sum of the two probabilities minus the probability of the overlap (intersection.) $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Example: Let's assign a probability to every possible three-letter codon. There are $4^3 = 64$ codons, so if one assumes that each one has equal probability, then they all equal $1/64$ (by axiom 1.) The probability of a codon having A as the first letter is $1/4$, and so is the probability of A as the second letter. Axiom 3 allows us to calculate the probability of A in either the first or the second letter:

$$P(AXX \cup XAX) = P(AXX) + P(XAX) - P(AAX) = 1/4 + 1/4 - 1/16 = 7/16$$

3.3 Probability distributions

The probability of each value of a random variable can be calculated from the probability of the event that corresponds to each value of the random variable. The collection of the probabilities of all of the values of the random variable is called the *probability distribution function* of the random variable, more formally the *mass function* for a discrete random variable or the *density function* for a continuous random variable.

For a discrete random variable (let's call it X) with a probability mass function f , the probability of X taking the value of a can be written either as $f(X = a)$ or $f(a)$, as long as it's clear that f is the probability distribution function of X . The one ironclad rule of probability is that all values of the mass function have to add up to 1. To state this mathematically, if all the possible values of X can be written as a_1, a_2, \dots (there may be finitely or infinitely many of them, as long as it's a countable infinity), this sum has to be equal to 1:

$$\sum_i f(a_i) = 1$$

A continuous random variable (let's call it Y) with a probability density function g is a bit more complicated. The continuous part means that the random variable has uncountably many values, even if the range is finite (for example, there are uncountably many real numbers between 0 and 1). Thus, the probability of any single value must be vanishingly small (zero), otherwise it would be impossible to add up (integrate) all of the values and get a finite result (let alone 1). We can only measure the probability of a range of values of Y and it is defined by the integral of the density function overall that range:

$$P(a < Y < b) = \int_a^b g(y)dy$$

The total probability over the entire range of Y has to be 1, but it's similarly calculated by integration instead of summation (R represents the range of values of Y):

$$\int_R g(y)dy = 1$$

Example: As codons (DNA triplets) code for amino acids, we can consider the genetic code a random variable on the sample space. Assuming all codons have equal probabilities, the probability of each amino acid is the number of triplets that code for it divided by 64. For example, the probabilities of leucine and arginine are $6/64 = 3/32$, the probability of threonine is $4/64 = 1/16$ and the probabilities of methionine and tryptophan are $1/64$. This defines a probability distribution function of the random variable of the genetic code. Note that the sum of all the probabilities of amino acids has to be 1. Of course there is no inherent reason why each triplet should be equally probable, so a different probability structure on the sample space would result in a different probability distribution (mass) function.

3.4 Measures of center: medians and means

The standard measures described here are applicable only numeric random variables. Some measures of center and spread for categorical variables exist as well.

The *median* of a random variable is the value which is in the middle of the distribution, specifically, that the probability of the random variable being no greater than that value is 0.5.

The *mean* or *expectation* of a random variable is the center of mass of the probability distribution. Specifically, it is defined for a mass function to be:

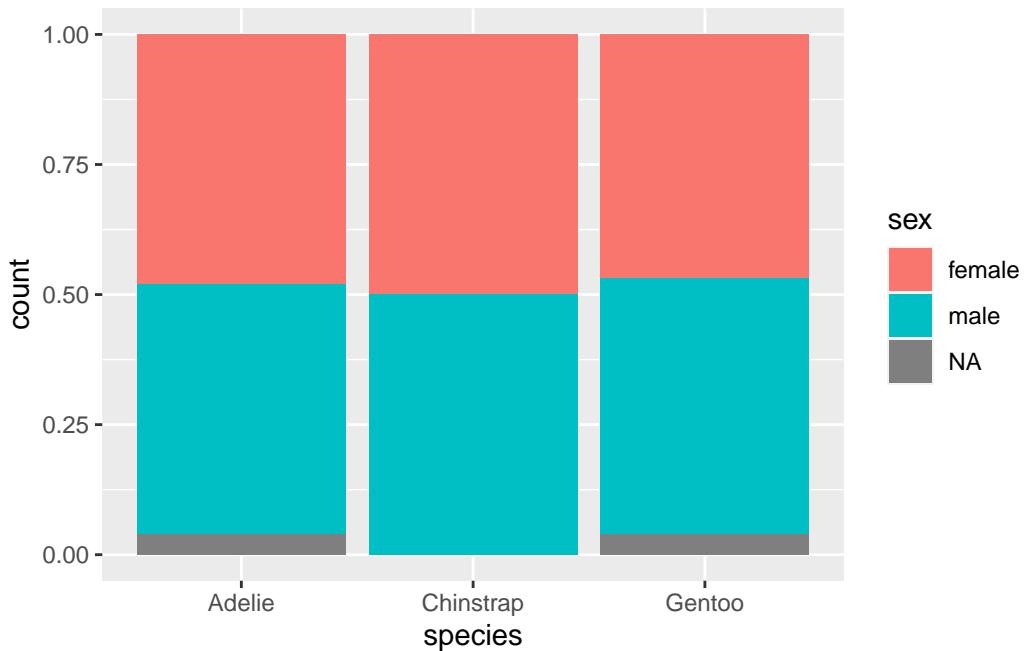
$$E(X) = \sum_i a_i f(a_i)$$

And for a density function it is defined using the integral:

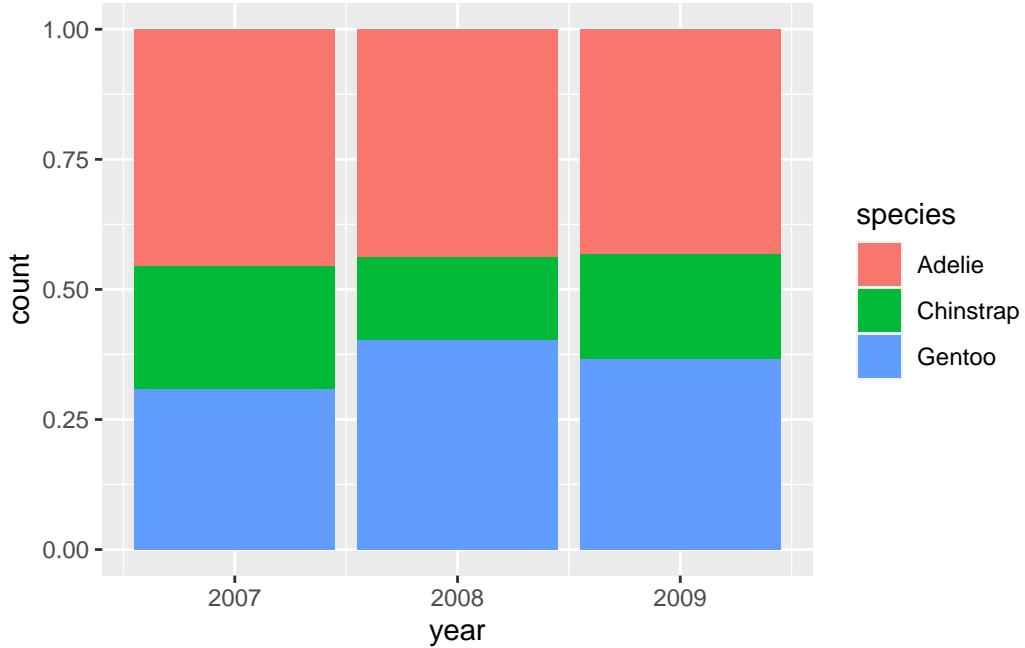
$$E(Y) = \int_R y g(y) dy$$

Example: Let us examine the factors (categorical variables) in the penguins data set. They cannot be described using means and medians, but can be plotted by counts in each category as you learned in the introduction to ggplot2:

```
ggplot(data = penguins) +  
  aes(x = species, fill = sex) +  
  geom_bar(position = "fill")
```



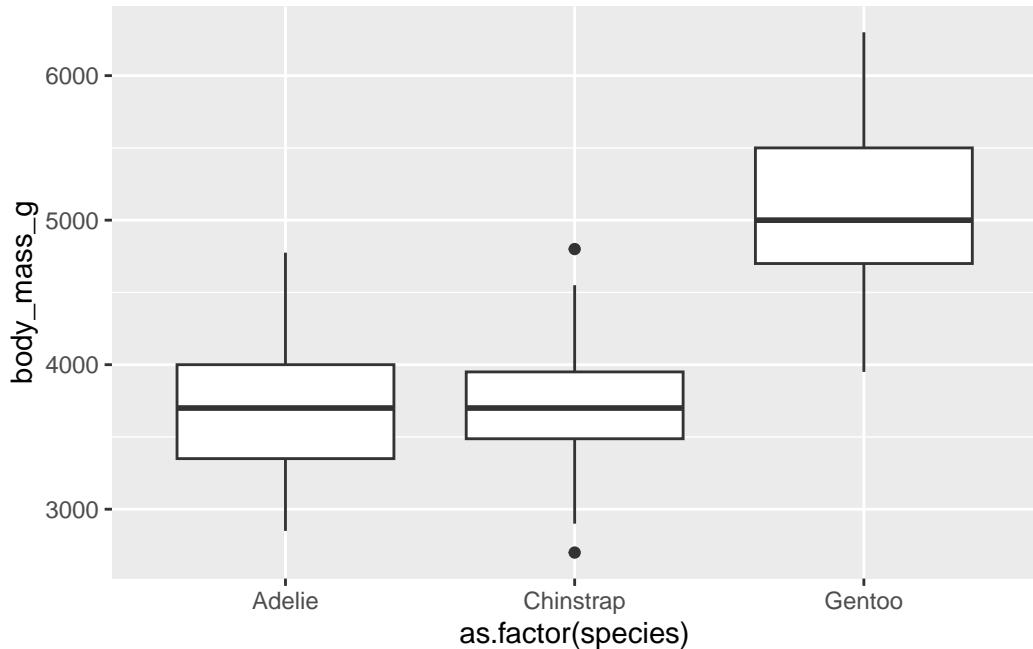
```
ggplot(data = penguins) +  
  aes(x = year, fill = species) +  
  geom_bar(position = "fill")
```



One can plot the distributions of numeric variables like body mass for different penguin species using box plots:

```
ggplot(data = penguins) + aes(x = as.factor(species), y=body_mass_g) + geom_boxplot()
```

Warning: Removed 2 rows containing non-finite values (`stat_boxplot()`).



The following code chunk uses `dplyr` functions that we will learn in the next chapter to calculate the mean and median values of these variables aggregated by species:

```
penguins %>% drop_na() %>% group_by(species) %>% summarise(mean = mean(body_mass_g))
```

```
# A tibble: 3 x 2
  species     mean
  <fct>     <dbl>
1 Adelie    3706.
2 Chinstrap 3733.
3 Gentoo    5092.
```

```
penguins %>% drop_na() %>% group_by(species) %>% summarise(median = median(body_mass_g))
```

```
# A tibble: 3 x 2
  species     median
  <fct>     <dbl>
1 Adelie     3700
2 Chinstrap   3700
3 Gentoo     5050
```

Comment on how the descriptive statistics correspond to the box plots.

3.5 Measures of spread: quartiles and variances

All random variables have spread in their values. The simplest way to describe it is by stating its range (the interval between the minimum and maximum values) and the quartiles (the medians of the two halves of the distribution).

A more standard measure of the spread of a distribution is the variance, defined as the expected value of the squared differences from the mean:

$$\text{Var}(X) = E[X - E(X)]^2 = \sum_i (a_i - E(X))^2 f(a_i)$$

And for a density function it is defined using the integral:

$$\text{Var}(Y) = E[Y - E(Y)]^2 = \int_R (y - E(Y))^2 g(y) dy$$

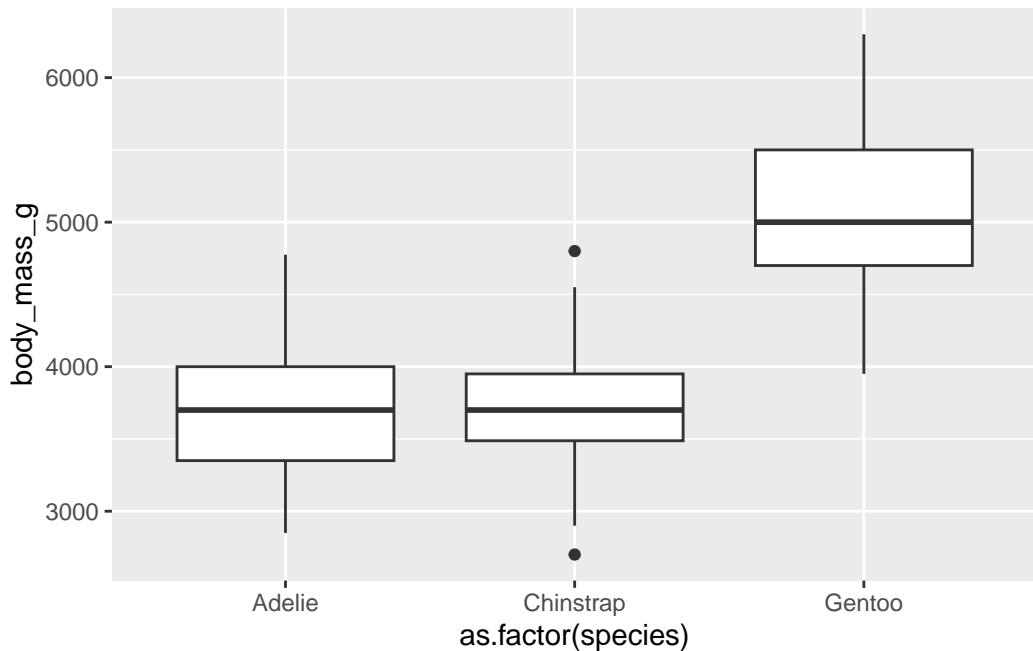
Variances have squared units so they are not directly comparable to the values of the random variable. Taking the square root of the variance converts it into the same units and is called the standard deviation of the distribution:

$$\sigma_X = \sqrt{\text{Var}(X)}$$

Example: Let's go back to the penguins data set and calculate the measures of spread for the variable body mass for different penguin species

```
ggplot(data = penguins) + aes(x = as.factor(species), y=body_mass_g) + geom_boxplot()
```

Warning: Removed 2 rows containing non-finite values (`stat_boxplot()`).



```
penguins %>% drop_na() %>% group_by(species) %>% summarise(var = var(body_mass_g))
```

```
# A tibble: 3 x 2
  species      var
  <fct>     <dbl>
1 Adelie    210332.
2 Chinstrap 147713.
3 Gentoo    251478.
```

```
penguins %>% drop_na() %>% group_by(species) %>% summarise(first_quart = quantile(body_ma
```

```
# A tibble: 3 x 2
  species   first_quart
  <fct>       <dbl>
1 Adelie     3362.
2 Chinstrap   3488.
3 Gentoo     4700
```

```
penguins %>% drop_na() %>% group_by(species) %>% summarise(third_quart = quantile(body_ma
```

```
# A tibble: 3 x 2
  species   third_quart
  <fct>     <dbl>
1 Adelie     4000
2 Chinstrap  3950
3 Gentoo    5500
```

Which species has a wider spread in its body mass? How do the descriptive stats and the box plots correspond?

3.6 Data as samples from distributions: statistics

In scientific practice, we collect data from one or more random variables, called a *sample*, and then try to make sense of it. One of the basic goals is statistical inference: using the data set to describe the *population* distribution from which the sample was drawn. Data sets can be plotted as *histograms* and the frequency/fraction of each value should be an approximation of the underlying probability distribution. In addition, descriptive statistics of the sample data (means, variances, medians, etc.) can be used to estimate the true parameters such as the mean and the variance of the population distribution.

Some of the fundamental questions about the population include:

1. What type of distribution is it?
2. Estimate the parameters of that distribution.
3. Test a hypothesis, e.g., whether two samples were drawn from the same distribution.
4. Describe and test a relationship between two or more variables.

3.6.1 Law of large numbers

First, the sample has to be *unbiased*, that is, no outcomes should be systematically over- or under-represented. But even an unbiased sample will differ from the population due to the inherent randomness of selection (sampling error). The **law of large numbers** states that as the *sample size* increases, the mean of the sample converges to the true mean of the population. Formally, for a set of n independent, identically distributed random variables (the sample) $\{X_i\}$ the sample mean \bar{X}_n converges to the mean of the distribution μ :

$$\lim_{n \rightarrow \infty} \frac{\sum_{i=1}^n X_i}{n} = \lim_{n \rightarrow \infty} \bar{X}_n = \mu$$

3.6.2 Central Limit Theorem

That is nice to know, but doesn't say exactly how large a sample is needed to estimate, for example, the mean of the population to a given precision. For that, we have the **Central Limit Theorem**, which states that the distribution of sample means (from samples of independent, identically distributed random variables) as sample size increases, approaches the normal (Gaussian) distribution with mean equal to the population mean and standard deviation equal to the standard deviation of the population divided by the square root of the sample size. Formally, it states that for a set of n independent, identically distributed random variables (the sample) $\{X_i\}$ with distribution mean μ and variance σ^2 , the probability density function of the sample mean \bar{X}_n converges for large sample size n to the normal distribution:

$$P(\bar{X}_n) \rightarrow N(\mu, \sigma^2/n)$$

where $N(\mu, \sigma^2/n)$ stands for the normal distribution with mean μ and variance σ^2/n . One extremely useful consequence of this theorem is that the variance of the sample mean is reciprocally related to the sample size n . More precisely, it allows the calculation of *confidence intervals* by using the normal distribution to generate an interval around the observed sample mean in which the true mean μ lies with a given likelihood.

This is an amazing result because it applies to any distribution, so it allows for the estimation of means for any situation, as long as the condition of independent, identically distributed variables in the sample is satisfied (the identical distributed condition can actually be relaxed). There are other central limit theorems that apply to other situations, including cases where the random variables in the sample are not independent (e.g., Markov models). The bottom line is that an unbiased sample contains a reflection of the true population, but it is always distorted by uncertainty. Larger sample sizes decrease the uncertainty but are more difficult and expensive to obtain.

Discussion: Suggest examples of biological data sets which are not made up of independent identically distributed random variables.

3.7 Exploration: misleading means

Means are the most common type of descriptive statistic and are sometimes the only numeric quantity used to compare two data sets, e.g. "the average GPA at school A is 3.5 vs 3.8 at school B". However, means can be misleading measures in multiple ways.

First, means are highly sensitive to outliers, or points that are very different from other values. They can skew the mean value, even pulling it completely away from the bulk of the values, in which case the mean ceases to be a measure of a "typical" value.

Second, there can be funny business with combining means of different *subsets* of data. Normally, you might expect if you have group A and group B, and each group has two subgroups divided by another variable (e.g. we are comparing the GPAs of students in school A and school B, and we split up the students in each school by gender), then if the means of each subgroup of A are larger than the means of the same subgroup of B (e.g. the GPA of girls and boys in school A are higher than those of their counterparts in school B), then the same relationship should be true for the combined mean of group A and group B (that is, the overall GPA in school A is higher than school B). That is not necessarily true!

This apparent contradiction is called Simpson's paradox. It can be illustrated in the data set of all the passengers and crew on the doomed ocean liner Titanic. The data set is found in the library `stablelearner` and is loaded by the chunk below:

```
library(stablelearner)
data(titanic)
str(titanic)

'data.frame': 2207 obs. of 11 variables:
 $ name      : chr "Abbing, Mr. Anthony" "Abbott, Mr. Eugene Joseph" "Abbott, Mr. Rossmore Edu...
 $ gender    : Factor w/ 2 levels "female","male": 2 2 2 1 1 2 2 1 2 2 ...
 $ age       : num 42 13 16 39 16 25 30 28 27 20 ...
 $ class     : Factor w/ 7 levels "1st","2nd","3rd",...: 3 3 3 3 3 3 2 2 3 3 ...
 $ embarked  : Factor w/ 4 levels "B","C","Q","S": 4 4 4 4 4 4 2 2 2 4 ...
 $ country   : Factor w/ 48 levels "Argentina","Australia",...: 44 44 44 15 30 44 17 17 26 16 ...
 $ ticketno  : int 5547 2673 2673 2673 348125 348122 3381 3381 2699 3101284 ...
 $ fare      : num 7.11 20.05 20.05 20.05 7.13 ...
 $ sibsp     : Ord.factor w/ 9 levels "0"<"1"<"2"<"3"<...: 1 1 2 2 1 1 2 2 1 1 ...
 $ parch     : Ord.factor w/ 10 levels "0"<"1"<"2"<"3"<...: 1 3 2 2 1 1 1 1 1 1 ...
 $ survived  : Factor w/ 2 levels "no","yes": 1 1 1 2 2 2 1 2 2 2 ...
```

The chunk below calculated the survival probability of passengers of all classes compared to the crew (of all types):

```
titanic %>% group_by(Passenger = class %in% c('1st', '2nd', '3rd'), survived) %>% summarise()

`summarise()` has grouped output by 'Passenger'. You can override using the
`.groups` argument.

# A tibble: 4 x 4
# Groups: Passenger [2]
  Passenger survived   num fraction
```

	<lgl>	<fct>	<int>	<dbl>
1	FALSE	no	679	0.763
2	FALSE	yes	211	0.237
3	TRUE	no	817	0.620
4	TRUE	yes	500	0.380

You can see that about 24% of the crew survived and almost 38% of the passengers survived. In this week's assignment you will calculate and explain what happens when you divide the people in each group by gender.

3.8 References

- Laplace's views on probability and determinism
- Central Limit Theorem in R
- Exploration of the Central Limit Theorem
- Simpson's paradox

4 Data wrangling

4.1 Goal

Learn how to manipulate large data sets by writing efficient, consistent, and compact code. Introduce the use of `dplyr`, `tidyR`, and the “pipe” operator `%>%`. Effortlessly produce statistics for grouped data. Massage data into “tidy” form.

4.2 What is data wrangling?

As biologists living in the XXI century, we are often faced with tons of data, possibly replicated over several organisms, treatments, or locations. We would like to streamline and automate our analysis as much as possible, writing scripts that are easy to read, fast to run, and easy to debug. Base R can get the job done, but often the code contains complicated operations, and a lot of \$ signs and brackets.

We’re going to learn about the packages `dplyr` and `tidyR`, which are part of `tidyverse` and can be used to manipulate large data frames in a simple and straightforward way. These tools are also much faster than the corresponding base R commands, are very compact, and can be concatenated into “pipelines”.

To start, we need to import the libraries:

```
library(tidyverse) # this loads both dplyr and tidyR, along with other packages
library(palmerpenguins) # a nice data set to play with

# make sure function select is the right one...
select <- dplyr::select
```

We are going to use the data set `penguins` from the package `palmerpenguins`, which we have already seen last week.

4.3 A new data type, tibble

The data is stored in a “tibble”:

```
class(penguins)

[1] "tbl_df"     "tbl"        "data.frame"
```

In fact, `dplyr` ships with a new data type, called a `tibble`. To convert a `data.frame` into a `tibble`, use `as_tibble`:

```
# load a data frame
data("trees")
class(trees)
trees <- as_tibble(trees)
class(trees)
```

The nice feature of `tbl` objects is that they will print only what fits on the screen, and also give you useful information on the size of the data, as well as the type of data in each column. Other than that, a `tbl` object behaves very much like a `data.frame`. In some rare cases, you want to transform the `tbl` back into a `data.frame`. For this, use the function `as.data.frame(tbl_object)`.

We can take a look at the data using one of several functions:

- `head(dt)` shows the first few rows
- `tail(dt)` shows the last few rows
- `glimpse(dt)` a summary of the data (similar to `str` in base R)
- `View(dt)` open in spreadsheet-like window

4.4 Selecting rows and columns

There are many ways to subset the data, either by row (subsetting the *observations*), or by column (subsetting the *variables*). For example, let’s select only the rows with observations from the island Torgersen:

```
filter(penguins, island == "Torgersen")
```

```

# A tibble: 52 x 8
  species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>        <dbl>        <dbl>        <int>        <int>
1 Adelie  Torgersen     39.1       18.7         181      3750
2 Adelie  Torgersen     39.5       17.4         186      3800
3 Adelie  Torgersen     40.3        18          195      3250
4 Adelie  Torgersen      NA          NA          NA       NA
5 Adelie  Torgersen     36.7       19.3         193      3450
6 Adelie  Torgersen     39.3       20.6         190      3650
7 Adelie  Torgersen     38.9       17.8         181      3625
8 Adelie  Torgersen     39.2       19.6         195      4675
9 Adelie  Torgersen     34.1       18.1         193      3475
10 Adelie  Torgersen    42          20.2         190      4250
# i 42 more rows
# i 2 more variables: sex <fct>, year <int>

```

We have 52 observations. We have used the command `filter(tbl, conditions)` to select certain observations. We can combine several conditions, by listing them side by side, possibly using logical operators.

Exercise: what does this do? `filter(penguins, bill_length_mm > 40, bill_depth_mm > 20, sex == male)`

We can also select particular variables (columns) using the function `select(tbl, cols to select)`. For example, select `species` and `island`:

```
select(penguins, species, island)
```

```

# A tibble: 344 x 2
  species island
  <fct>   <fct>
1 Adelie  Torgersen
2 Adelie  Torgersen
3 Adelie  Torgersen
4 Adelie  Torgersen
5 Adelie  Torgersen
6 Adelie  Torgersen
7 Adelie  Torgersen
8 Adelie  Torgersen
9 Adelie  Torgersen
10 Adelie  Torgersen
# i 334 more rows

```

How many `species` are represented in the data set? We can use the function `distinct(tbl, cols to select)` to retain only the rows that differ from each other:

```
distinct(select(penguins, species))

# A tibble: 3 x 1
  species
  <fct>
1 Adelie
2 Gentoo
3 Chinstrap
```

Showing that there are three species, once we removed the duplicates. There are many other ways to subset observations:

- `slice_sample(tbl, howmany, replace = TRUE)` sample `howmany` rows at random (with replacement)
- `sample_sample(tbl, proportion, replace = FALSE)` sample a certain proportion (e.g. 0.2 for 20%) of rows at random without replacement
- `slice(tbl, 5:20)` extract the rows 5 to 20
- `slice_max(penguins, 10, body_mass_g)` extract the first 10 rows, once ordered by `body_mass_g`

More ways to select columns:

- `select(penguins, contains("mm"))` select all columns containing the string `mm`
- `select(penguins, -year, -body_mass_g)` exclude the columns `year` and `body_mass_g`
- `select(penguins, matches("length|bill"))` select all columns whose names match a regular expression

4.5 Creating pipelines using `%>%`

We've been calling nested functions, such as `distinct(select(penguins, species))`. If you have to add another layer or two, the code would become unreadable. `dplyr` allows you to "un-nest" these functions and create a "pipeline" in which you concatenate commands separated by a special operator, `%>%`. For example:

```
penguins %>% # take a data table
  select(species) %>% # select a column
  distinct() # remove duplicates
```

```
# A tibble: 3 x 1
  species
  <fct>
1 Adelie
2 Gentoo
3 Chinstrap
```

does exactly the same operations as the command above, but is much more readable. By concatenating many commands, you can create incredibly complex pipelines while retaining readability. It is also quite easy to add another piece of the pipeline in between commands, or to comment some of the pipeline out.

Another advantage of pipelines is that they help with name completion. In fact, RStudio is running in the background your pipeline while you type it. Try typing `dt %>% filter(` and then start typing `bill` and press Tab: you will see the options to complete the column name; choose it with your arrows and hit Return. The back tick-marks will be added automatically if needed (e.g., column names containing spaces, or starting with a digit).

4.6 Producing summaries

Sometimes we need to calculate statistics on certain columns. For example, calculate the average body mass of the penguins. We can do this using `summarise` (you can use British or American spelling):

```
penguins %>%
  summarise(avg = mean(body_mass_g, na.rm = TRUE))
```

```
# A tibble: 1 x 1
  avg
  <dbl>
1 4202.
```

```
# alternatively, drop_na(body_mass_g) removes all the observations for which
# body_mass_g is NA
penguins %>%
  drop_na(body_mass_g) %>%
  summarise(avg = mean(body_mass_g, na.rm = TRUE))
```

```
# A tibble: 1 x 1
  avg
  <dbl>
1 4202.
```

where we used `na.rm = TRUE` to ignore missing values. This command returns a `tbl` object with just the average body mass. You can combine multiple statistics (use `first`, `last`, `min`, `max`, `n` [count the number of rows], `n_distinct` [count the number of distinct rows], `mean`, `median`, `var`, `sd`, etc.):

```
penguins %>%
  summarise(avg = mean(body_mass_g, na.rm = TRUE),
            sd = sd(body_mass_g, na.rm = TRUE),
            median = median(body_mass_g, na.rm = TRUE))
```

```
# A tibble: 1 x 3
  avg     sd median
  <dbl> <dbl>  <dbl>
1 4202.   802.    4050
```

4.7 Summaries by group

One of the most useful features of `dplyr` is the ability to produce statistics for the data once subsetted by *groups*. For example, we would like to compute the average body mass by species and sex:

```
penguins %>%
  drop_na() %>%
  group_by(sex, species) %>%
  summarise(mean = mean(body_mass_g, na.rm = TRUE))
```

`'summarise()'` has grouped output by 'sex'. You can override using the `'.groups'` argument.

```
# A tibble: 6 x 3
# Groups:   sex [2]
  sex     species     mean
  <fct>   <fct>     <dbl>
1 female  Adelie    3369.
```

```

2 female Chinstrap 3527.
3 female Gentoo    4680.
4 male   Adelie    4043.
5 male   Chinstrap 3939.
6 male   Gentoo    5485.

```

showing that male penguins are heavier for the three species considered.

Exercise: find the average `bill_depth_mm` and `bill_length_mm` by `species` and `sex`. Filter the data to consider only observations for the year 2008.

4.8 Ordering the data

To order the data according to one or more variables, use `arrange()`:

```

penguins %>%
  arrange(body_mass_g) # ascending

# A tibble: 344 x 8
  species   island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>     <fct>        <dbl>        <dbl>        <dbl>        <int>
1 Chinstrap Dream      46.9       16.6       192       2700
2 Adelie     Biscoe     36.5       16.6       181       2850
3 Adelie     Biscoe     36.4       17.1       184       2850
4 Adelie     Biscoe     34.5       18.1       187       2900
5 Adelie     Dream      33.1       16.1       178       2900
6 Adelie     Torgers~    38.6       17         188       2900
7 Chinstrap Dream      43.2       16.6       187       2900
8 Adelie     Biscoe     37.9       18.6       193       2925
9 Adelie     Dream      37.5       18.9       179       2975
10 Adelie    Dream      37         16.9      185       3000
# i 334 more rows
# i 2 more variables: sex <fct>, year <int>

penguins %>%
  arrange(desc(body_mass_g)) # descending

# A tibble: 344 x 8
  species   island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>     <fct>        <dbl>        <dbl>        <dbl>        <int>
1 Chinstrap Dream      46.9       16.6       192       2700
2 Adelie     Biscoe     36.5       16.6       181       2850
3 Adelie     Biscoe     36.4       17.1       184       2850
4 Adelie     Biscoe     34.5       18.1       187       2900
5 Adelie     Dream      33.1       16.1       178       2900
6 Adelie     Torgers~    38.6       17         188       2900
7 Chinstrap Dream      43.2       16.6       187       2900
8 Adelie     Biscoe     37.9       18.6       193       2925
9 Adelie     Dream      37.5       18.9       179       2975
10 Adelie    Dream      37         16.9      185       3000
# i 334 more rows
# i 2 more variables: sex <fct>, year <int>

```

```

<fct> <fct> <dbl> <dbl> <int> <int>
1 Gentoo Biscoe 49.2 15.2 221 6300
2 Gentoo Biscoe 59.6 17 230 6050
3 Gentoo Biscoe 51.1 16.3 220 6000
4 Gentoo Biscoe 48.8 16.2 222 6000
5 Gentoo Biscoe 45.2 16.4 223 5950
6 Gentoo Biscoe 49.8 15.9 229 5950
7 Gentoo Biscoe 48.4 14.6 213 5850
8 Gentoo Biscoe 49.3 15.7 217 5850
9 Gentoo Biscoe 55.1 16 230 5850
10 Gentoo Biscoe 49.5 16.2 229 5800
# i 334 more rows
# i 2 more variables: sex <fct>, year <int>

```

4.9 Renaming columns

To rename one or more columns, use `rename()`:

```

penguins %>%
  rename(bm = body_mass_g)

# A tibble: 344 x 8
  species island bill_length_mm bill_depth_mm flipper_length_mm   bm sex
  <fct>   <fct>      <dbl>        <dbl>          <dbl> <int> <int> <fct>
1 Adelie  Torgersen     39.1       18.7           152    3750 male
2 Adelie  Torgersen     39.5       17.4           162    3800 female
3 Adelie  Torgersen     40.3       18             172    3250 female
4 Adelie  Torgersen      NA         NA            182    3450 <NA>
5 Adelie  Torgersen     36.7       19.3           142    3650 male
6 Adelie  Torgersen     39.3       20.6           172    3650 male
7 Adelie  Torgersen     38.9       17.8           152    3625 female
8 Adelie  Torgersen     39.2       19.6           162    4675 male
9 Adelie  Torgersen     34.1       18.1           132    3475 <NA>
10 Adelie Torgersen      42         20.2           182    4250 <NA>
# i 334 more rows
# i 1 more variable: year <int>

```

4.10 Adding new variables using `mutate`

If you want to add one or more new columns, with the content being a function of other columns, use the function `mutate`. For example, we are going to add a new column showing the z-score for the body mass of each individual:

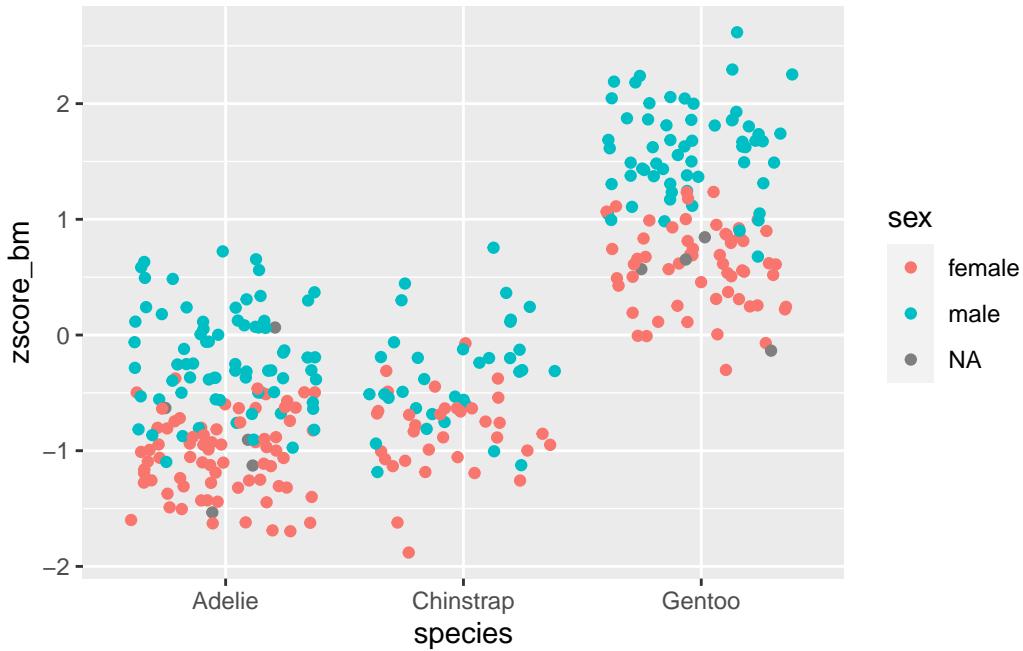
```
penguins %>%
  mutate(zscore_bm = scale(body_mass_g)) %>%
  select(species, sex, body_mass_g, zscore_bm)

# A tibble: 344 x 4
  species   sex   body_mass_g   zscore_bm[,1]
  <fct>     <fct>    <int>           <dbl>
1 Adelie    male      3750        -0.563
2 Adelie    female    3800        -0.501
3 Adelie    female    3250        -1.19 
4 Adelie    <NA>       NA          NA    
5 Adelie    female    3450        -0.937
6 Adelie    male      3650        -0.688
7 Adelie    female    3625        -0.719
8 Adelie    male      4675         0.590
9 Adelie    <NA>       3475        -0.906
10 Adelie   <NA>      4250         0.0602
# i 334 more rows
```

We can pipe the results to `ggplot` for plotting!

```
penguins %>%
  mutate(zscore_bm = scale(body_mass_g)) %>%
  select(species, sex, body_mass_g, zscore_bm) %>%
  ggplot() + aes(x = species, y = zscore_bm, colour = sex) +
  geom_jitter()
```

Warning: Removed 2 rows containing missing values (`geom_point()`).



You can use the function `transmute()` to create a new column and drop the original columns.

Most importantly, you can use `mutate` and `transmute` on grouped data. For example, let's recompute the z-score of the `body_mass_g` once the data is grouped by species and sex:

```
penguins %>%
  drop_na() %>%
  select(species, sex, body_mass_g) %>%
  group_by(species, sex) %>%
  mutate(zscore_bm = scale(body_mass_g)) %>%
  arrange(body_mass_g)
```

```
# A tibble: 333 x 4
# Groups:   species, sex [6]
  species   sex   body_mass_g   zscore_bm[,1]
  <fct>     <fct>     <int>           <dbl>
1 Chinstrap female    2700      -2.90
2 Adelie     female    2850      -1.93
3 Adelie     female    2850      -1.93
4 Adelie     female    2900      -1.74
5 Adelie     female    2900      -1.74
6 Adelie     female    2900      -1.74
7 Chinstrap female    2900      -2.20
```

```

8 Adelie   female      2925      -1.65
9 Adelie   female      3000      -1.37
10 Adelie  female      3000      -1.37
# i 323 more rows

```

4.11 Data wrangling

Data is rarely in a format that is good for computing, and much effort goes into reading the data and wrestling with it to make it into a good format. As the name implies, `tidyverse` strongly advocates for the use of data in *tidy* form. What does this mean?

- Each variable forms a column
- Each observation forms a row
- Each type of observational unit forms a table

This is often called *narrow table* format. Any other form of data (e.g., *wide table* format) is considered *messy*. However, often data are not organized in tidy form, or we want to produce tables for human consumption rather than computer consumption. The package `tidyverse` allows to accomplish just that. It contains only a few, very powerful functions. To explore this issue, we build a data set containing the average body mass by species and sex:

```

penguin_bm <- penguins %>%
  drop_na() %>%
  group_by(sex, species) %>%
  summarise(body_mass = mean(body_mass_g), .groups = "drop") # remove groups after calculation

penguin_bm

# A tibble: 6 x 3
  sex     species   body_mass
  <fct>   <fct>     <dbl>
1 female  Adelie    3369.
2 female  Chinstrap 3527.
3 female  Gentoo    4680.
4 male    Adelie    4043.
5 male    Chinstrap 3939.
6 male    Gentoo    5485.

```

4.12 From narrow to wide

Our data is in tidy form. For a paper, we want to show the difference between males and females in a table:

```
penguin_bm %>%
  pivot_wider(names_from = sex, values_from = body_mass)

# A tibble: 3 x 3
  species   female   male
  <fct>     <dbl>   <dbl>
1 Adelie    3369.  4043.
2 Chinstrap 3527.  3939.
3 Gentoo    4680.  5485.
```

where we have created new column names using the values found in `sex` (hence, `names_from`), and filled each cell with the corresponding value found in `body_mass` (hence, `values_from`). Similarly, if we want to show the data with species as column names, and sex as rows, we can use:

```
penguin_bm %>%
  pivot_wider(names_from = species, values_from = body_mass)

# A tibble: 2 x 4
  sex      Adelie Chinstrap Gentoo
  <fct>    <dbl>    <dbl>   <dbl>
1 female   3369.    3527.   4680.
2 male     4043.    3939.   5485.
```

4.13 From wide to narrow

For a real-world example, we will make data from:

Tree-ring analysis for sustainable harvest of Millettia stuhlmannii in Mozambique,
I.A.D.Remane M.D.Therrell, **South African Journal of Botany** Volume 125,
September 2019, Pages 120-125

You can read a tab-separated file from:

```

dt <- read_tsv("https://raw.githubusercontent.com/StefanoAllesina/BIOS_26318/master/data/all_trees.csv")
select(Age, contains("CAT"))

New names:
Rows: 172 Columns: 55
-- Column specification
----- Delimiter: "\t" dbl
(37): Age, CAT01, CAT03, CAT04A, CAT05B, CAT06, CAT07, CAT08A, CAT09C, C... lgl
(18): ...38, ...39, ...40, ...41, ...42, ...43, ...44, ...45, ...46, .....
i Use `spec()` to retrieve the full column specification for this data. i
Specify the column types or set `show_col_types = FALSE` to quiet this message.
* `Mean` -> `Mean...32`
* `Mean` -> `Mean...35`
* `` -> `...37`
* `` -> `...38`
* `` -> `...39`
* `` -> `...40`
* `` -> `...41`
* `` -> `...42`
* `` -> `...43`
* `` -> `...44`
* `` -> `...45`
* `` -> `...46`
* `` -> `...47`
* `` -> `...48`
* `` -> `...49`
* `` -> `...50`
* `` -> `...51`
* `` -> `...52`
* `` -> `...53`
* `` -> `...54`
* `` -> `...55`

# selecting only age and samples

```

Each column besides YEAR represents a single tree, and each cell contains the diameter (in cm) of the tree when it was at a given age. To make this in tidy form, we first create the columns `tree` and `diameter`:

```

dt <- dt %>%
pivot_longer(-Age, names_to = "tree", values_to = "diameter")

```

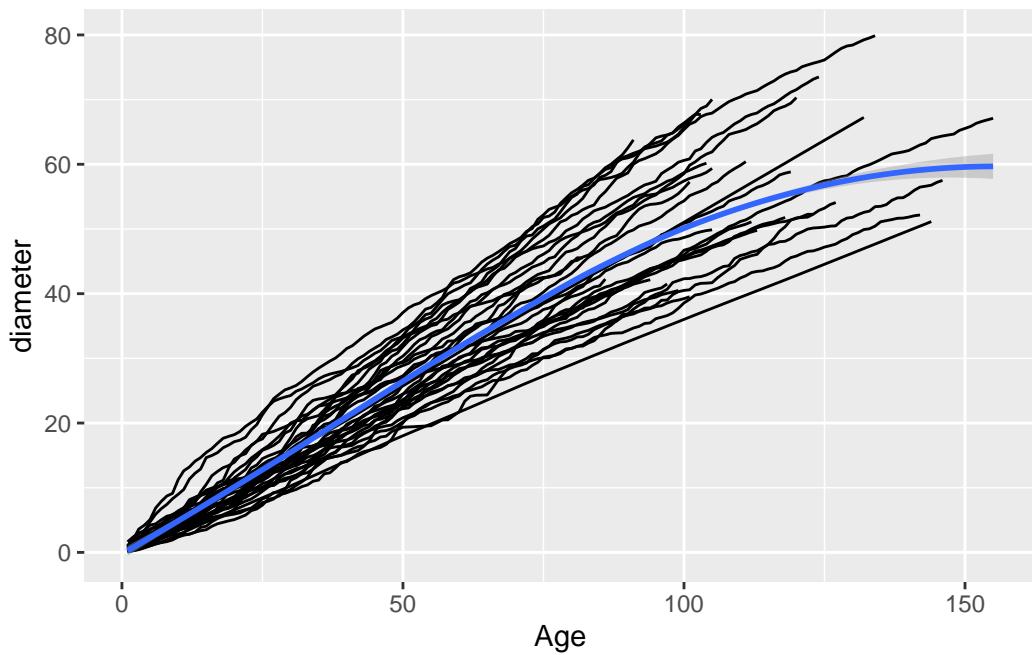
and then remove the NAs:

```
dt <- dt %>% filter(!is.na(diameter))
```

Now it is easy to plot the growth trajectory of each tree (as in Fig. 3 of the original paper):

```
dt %>%
  ggplot() +
  aes(x = Age, y = diameter) +
  geom_line(aes(group = tree)) + # note---this makes a line for each tree
  geom_smooth(method = "loess") # while the smoothing function considers all trees

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



4.14 Separate: split a column into two or more

```
test <- tibble(name = c("Allesina, Stefano", "Kondrashov, Dmitry", "Mir, Amatullah"))
test
```

```

# A tibble: 3 x 1
  name
  <chr>
1 Allesina, Stefano
2 Kondrashov, Dmitry
3 Mir, Amatullah

test %>% separate(name, into = c("last_name", "first_name"), sep = ", ")

```



```

# A tibble: 3 x 2
  last_name first_name
  <chr>     <chr>
1 Allesina   Stefano
2 Kondrashov Dmitry
3 Mir         Amatullah

```

The complement of `separate` is called `unite`.

4.15 Separate rows: from one row to many

```

test <- tibble(id = c(1, 2, 3, 4), records = c("a;b;c", "c;d", "a;e", "f"))
test

# A tibble: 4 x 2
  id records
  <dbl> <chr>
1     1 a;b;c
2     2 c;d
3     3 a;e
4     4 f

```

To make it into tidy form, only one record per row:

```
test %>% separate_rows(records, sep = ";")
```

```
# A tibble: 8 x 2
  id records
  <dbl> <chr>
1     1 a
2     1 b
3     1 c
4     2 c
5     2 d
6     3 a
7     3 e
8     4 f
```

4.16 Example: brown bear, brown bear, what do you see?

This exercise uses a dataset from [GBIF](#), the Global Biodiversity Information Facility. You can download the latest version yourself by doing the following (but just skip ahead if you want to use the data provided by us).

1. Go to [GBIF](#) and click on Occurrences.
2. Under Scientific Name type in *Ursus arctos* (brown bear), and hit enter.
3. To download the data, create an account on GBIF
4. Then click on Download, and select Simple (which should have a tab-delimited .csv file)
5. Save to the data folder in your working folder.

If you don't want to go through all this, you can load this previously downloaded file called `Ursus_GBIF.csv` from our GitHub repository. The code in the following chunk loads and displays the contents of the tibble:

```
# you will need ggmap!
library(ggmap)
Ursus_data <- read_tsv("https://raw.githubusercontent.com/StefanoAllesina/BIOS_26318/master/Ursus_GBIF.csv")
glimpse(Ursus_data)
```

```
Rows: 23,498
Columns: 50
$ gbifID                               <dbl> 2382421192, 2382420986, 2382420916, 2~
$ datasetKey                            <chr> "88d8974c-f762-11e1-a439-00145eb45e9a~
$ occurrenceID                          <chr> "http://arctos.database.museum/guid/U~
$ kingdom                                <chr> "Animalia", "Animalia", "Animalia", "~
$ phylum                                 <chr> "Chordata", "Chordata", "Chordata", "~
$ class                                  <chr> "Mammalia", "Mammalia", "Mammalia", "~
```

```

$ order <chr> "Carnivora", "Carnivora", "Carnivora"~
$ family <chr> "Ursidae", "Ursidae", "Ursidae", "Ursi~
$ genus <chr> "Ursus", "Ursus", "Ursus", "Ursus", "~
$ species <chr> "Ursus arctos", "Ursus arctos", "Ursu~
$ infraspecificEpithet <chr> NA, NA, NA, "horribilis", NA, NA, NA, ~
$ taxonRank <chr> "SPECIES", "SPECIES", "SPECIES", "SUB~
$ scientificName <chr> "Ursus arctos Linnaeus, 1758", "Ursus~
$ verbatimScientificName <chr> "Ursus arctos", "Ursus arctos", "Ursu~
$ verbatimScientificNameAuthorship <chr> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ countryCode <chr> NA, "US", NA, NA, "US", NA, NA, "US", ~
$ locality <chr> "no specific locality recorded", "no ~
$ stateProvince <chr> NA, "Alaska", NA, NA, "Colorado", NA, ~
$ occurrenceStatus <chr> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ individualCount <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ publishingOrgKey <chr> "4cadac10-3e7b-11d9-8439-b8a03c50a862~
$ decimalLatitude <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ decimalLongitude <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ coordinateUncertaintyInMeters <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ coordinatePrecision <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ elevation <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ elevationAccuracy <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ depth <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ depthAccuracy <dbl> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ eventDate <dttm> 1800-01-01, 1800-01-01, 1800-01-01, ~
$ day <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ month <dbl> 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1~
$ year <dbl> 1800, 1800, 1800, 1800, 1914, 1938, 1~
$ taxonKey <dbl> 2433433, 2433433, 2433433, 6163845, 2~
$ speciesKey <dbl> 2433433, 2433433, 2433433, 2433433, 2~
$ basisOfRecord <chr> "PRESERVED_SPECIMEN", "PRESERVED_SPEC~
$ institutionCode <chr> "UCM", "UCM", "UCM", "UCM", "UC~
$ collectionCode <chr> "Mammal specimens", "Mammal specimens~
$ catalogNumber <chr> "UCM:Mamm:5003", "UCM:Mamm:3329", "UC~
$ recordNumber <chr> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ identifiedBy <chr> "T. C. Hart", "unknown", "unknown", "~
$ dateIdentified <dttm> 2013-01-01, 1936-01-01, NA, 2015-10--~
$ license <chr> "CCO_1_0", "CCO_1_0", "CCO_1_0", "CCO~
$ rightsHolder <chr> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ recordedBy <chr> "Collector(s): T. C. Hart", "Collecto~
$ typeStatus <chr> NA, NA, NA, NA, NA, NA, NA, NA, N~
$ establishmentMeans <chr> NA, NA, NA, NA, NA, "MANAGED", NA~
$ lastInterpreted <dttm> 2019-09-03 22:11:14, 2019-09-03 22:1~
$ mediaType <chr> NA, NA, NA, NA, NA, NA, NA, NA, N~

```

```
$ issue <chr> NA, NA, NA, NA, "TAXON_MATCH_HIGHERRA~
```

You see there are 50 variables in the data set, so it may be useful to remove the ones we don't need. For this exercise, our objective is to plot the occurrences of this species on the world map, so we need two variables for certain: `decimalLatitude` and `decimalLongitude`, as well as the `BasisofRecord` for additional information. Use your `tidyverse` skills to create a new tibble with only those variables. In addition, remove duplicate records from the tibble.

```
# your code goes here!
```

Now we can plot this data set on the world map, using the useful package `maps`. To plot, use the `ggplot()` syntax with the following addition:

```
mapWorld <- borders("world", colour="gray50", fill="gray50") # create a layer of borders  
# now you can call  
# ggplot() + mapWorld + ...
```

Note the warning message generated by `ggplot`. Then consider the map with the locations of the brown bear specimens. Do any of them seem strange to you? What may be the explanation behind these strange data point? Now filter out the points that you identified as suspicious and print out their `BasisofRecord`. Does this suggest an explanation for the strangeness?

```
# your code goes here!
```

4.17 Resources

- [R for Data Science](#)
- A [cool class](#) at U of C in Social Sciences
- [Data transformation](#) cheat sheet
- [Dealing with dates](#) cheat sheet
- [Data import](#) cheat sheet

5 Distributions and their properties

5.1 Objectives:

- Apply concepts of conditional probability to practical scenarios and questions
- Describe independence as a concept and apply to data sets
- Use random number generators to simulate various distributions
- Be familiar with the shape of several common distributions and describe the role of their parameters

5.2 Independence

5.2.1 Conditional probability

In the basic definitions of probability, we considered the probabilities of each outcome and events separately. Let us consider how information about one event affects the probability of another event. The concept is that if one event (let's call it B) is true, unless the event is the entire space, it rules out some other outcomes. This may affect the probability of other events (e.g., A) in the sample space, because knowledge of B may rule out some of the outcomes in A as well. Here is the formal definition:

Definition: For two events A and B in a sample space Ω with a probability measure P , the probability of A given B , called the **conditional probability**, defined as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

where $A \cap B$ or A, B is the intersection of events A and B , also known as “ A and B ”—the event consisting of all outcomes that are in both A and B .

In words, given the knowledge that an event B occurs, the sample space is restricted to the subset B , which is why the denominator in the definition is $P(B)$. The numerator encompasses all the outcomes we are interested in, (i.e., A), but since we are now restricted to B , the numerator consists of all the outcomes of A which are also in B , or $A \cap B$. The definition makes sense in two extreme cases: if $A = B$ and if A and B are mutually exclusive:

$$P(B|B) = P(B \cap B)/P(B) = P(B)/P(B) = 1$$

If $P(A \cap B) = 0$, then $P(A|B) = 0/P(B) = 0$

Important note: one common source of confusion about conditional probability is the difference between the probability of A and B and the probability of A given B . This is a result of the discrepancy between everyday word usage and mathematical terminology, because the statement “what are the odds of finding a tall person who also likes tea?” is hard to distinguish from “what are the odds that a person who is tall likes tea?” The critical difference between these two statements is that in the former you start out with no information and are picking out a person from the entire population, while in the latter you start out with the knowledge that a person is tall.

Example: In the classic Mendelian pea experiment, each diploid organism carries two alleles. The allele A is dominant and results in pink flowers, while a is recessive and results in white flowers. There are three possible genotypes (AA , Aa , aa) and two phenotypes (Pink or White). For the questions below, assume that two heterozygous pea plants (each having genotype Aa) are crossed, producing the following table of genotypes with equal probabilities in each cell:

parent	A	a
A	AA (pink)	Aa (pink)
a	Aa (pink)	aa (white)

1. What is the probability that a plant with pink flowers has genotype AA ? Write this down in terms of conditional probability and explain how it's different from the probability of a plant having both pink flower and genotype AA .
2. What is the probability that a plant with genotype AA has pink flowers? Again, write down the conditional probability and explain how it's different from the probability of a plant having both pink flower and genotype AA .

Lesson: in general,

$$P(X|Y) \neq P(Y|X)$$

5.2.2 Independence

Independence is a fundamental concept in probability that may be misinterpreted without careful thinking. Intuitively, two events (or random variables) are independent if one does not influence the other. More precisely, it means that the probability of one event is the same regardless of whether the other one happens or not. This is expressed precisely using conditional probabilities:

Definition: Two events A and B in a sample space Ω with a probability measure P are **independent** if $P(A|B) = P(A)$, or equivalently if $P(B|A) = P(B)$.

Independence is not a straightforward concept. It may be confused with mutual exclusivity, as one might surmise that if A and B have no overlap, then they are independent. That however, is false by definition, since $P(A|B)$ is 0 for two mutually exclusive events. The confusion stems from thinking that if A and B are non-overlapping, then they do not influence each other. But the notion of influence in this definition is about information; so if A and B are mutually exclusive, the knowledge that one of them occurs has an influence on the probability of the other one occurring, specifically it rules the other one out.

Example: In the sample space of weather phenomena, are the events of snowing and hot weather independent?

Example: A slightly more subtle example, the lifetime risk of breast cancer is about 1 in 8 for women and about 1 in 1000 for men. Are sex and breast cancer independent?

5.2.3 Usefulness of independence

Independence is a mathematical abstraction, and reality rarely provides us with perfectly independent variables. But it's a very useful abstraction in that it enables calculations that would be difficult or impossible to carry out without this assumption.

First, independence allows for calculating the probability of two events or two random variables simultaneously. This is a straightforward consequence of the definition conditional probability (first equality) and independence (second equality):

$$\frac{P(A \cap B)}{P(B)} = P(A|B) = P(A)$$

Multiplying both sides by $P(B)$, we get the **product rule** of independence, perhaps the most widely used formula in applied probability:

$$P(A \cap B) = P(A)P(B)$$

Example: The probability that two randomly selected individuals have red hair—assuming that the occurrence of this trait is independent—is the square of the probability of red hair in one individual. (Note that this is never exactly the case for a finite population—why?)

Example: The probability of two alleles of two separate genes (call them A and B) occurring on the same gamete may be independent or may be linked. In population genetics, the concept of *linkage disequilibrium* describes the extent of such linkage; for example, alleles that are located on separate chromosomes (in eukaryotes) are usually not linked and their occurrence is independent. The *coefficient of linkage disequilibrium* is defined as the difference between what is expected from independence and the actual probability of both alleles being present:

$$D_{AB} = P(A \cap B) - P(A)P(B)$$

$P(A)$ and $P(B)$ are the frequencies of the two respective alleles (haplotypes) in the population, while $P(A \cap B)$ is the frequency of the haplotypes occurring together in the same copy of the genome (that is, on the same gamete). For two independent loci, $D_{AB} = 0$, while for loci that usually occur together the coefficient will be positive, and its magnitude is influenced both by physical proximity of the loci on a chromosome, the evolutionary history of the species, and other factors.

Another important consequence of independence has to do with the sum of two independent random variables. The expectation of the sum of any random variables is linear, which can be demonstrated using some work with sums, starting from the definition of expectation (the same can be shown for continuous random variables, using integrals instead of sums):

$$\begin{aligned}
E(X + Y) &= \sum_i \sum_j (x_i + y_j) P(x_i, y_j) = \\
&= \sum_i \sum_j x_i P(x_i, y_j) + \sum_i \sum_j y_j P(x_i, y_j) = \sum_i x_i \sum_j P(x_i, y_j) + \sum_j y_j \sum_i P(x_i, y_j) = \\
&= \sum_i x_i P(x_i) + \sum_j y_j P(y_j) = E(X) + E(Y)
\end{aligned}$$

Summing up a joint probability distribution over all values of one variable removes that variable, $\sum_j P(x_i, y_j) = P(x_i) \sum_i P(x_i, y_j) = P(y_j)$, so this leave us with the two separate expected values:

$$\begin{aligned}
\text{Var}(X + Y) &= E[(X + Y) - (E_X + E_Y)]^2 = \\
&= E[(X - E_X)^2 + (Y - E_Y)^2 - 2(X - E_X)(Y - E_Y)] = \\
&= E(X - E_X)^2 + E(Y - E_Y)^2 - 2E[(X - E_X)(Y - E_Y)] =
\end{aligned}$$

The first two terms are the respective variances, while the third term is called the *covariance* of X and Y :

$$= \text{Var}(X) + \text{Var}(Y) - 2\text{Cov}(X, Y)$$

Covariance describes how much two random variables vary together, or more precisely, how much they deviate from their respective means in the same direction. Thus it should be reasonable to think that two independent random variables have covariance 0, which is demonstrated as follows:

$$E[(X - E_X)(Y - E_Y)] = E(XY) - E_Y E_X - E_Y E_X + E_X E_Y = E(XY) - E_X E_Y$$

We can write the expression for the expectation of the random variable comprised of all pairs of values of X and Y , using the fact that for two independent random variables, $P(x_i, y_j) = P(x_i)P(y_j)$ for all values x_i and y_j :

$$E(XY) = \sum_i \sum_j x_i y_j P(x_i, y_j) = \sum_i x_i P(x_i) \sum_j y_j P(y_j) = E_X E_Y$$

The calculation for two continuous random variables is analogous, only with integrals instead of sums.

This demonstrates that the covariance of two independent random variables is 0, and thus that the variance of a sum of two independent random variables is the sum of the two separate variables.

Example: This property of variance is often used in analysis of noise or error in data. It is commonly assumed in least squares fitting that noise in data is independent of the signal or model underlying the data. This is the foundation for statements like “this linear regression explains 80% of the variance in the data.”

5.3 Probability distribution examples (discrete)

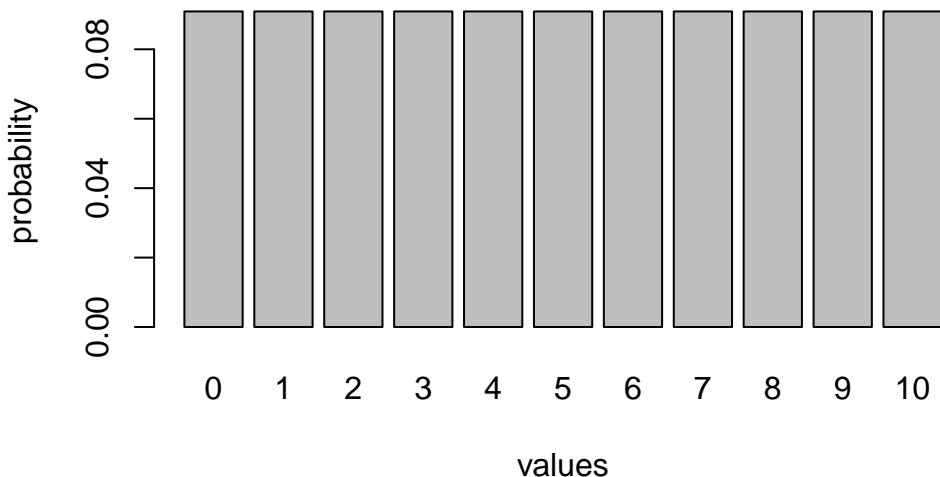
The following are examples of distributions of random variables with discrete values. The first two have finite support (finitely many values) while the second two have infinite support.

5.3.1 Uniform

The simplest probability distribution in which every value has the same probability (and one which is sometimes called “purely random” even though any random variable with any distribution is just as random). The probability distribution for a uniform random variable with n values is $P(x) = 1/n$ for any value x .

```
low <- 0 # minimum value
high <- 10 # maximum value
values <- low:high # vector of discrete values of the RV
num <- length(values)
probs <- rep(1 / num, num) # uniform mass function vector
barplot(probs, names.arg = values, xlab = 'values', ylab = 'probability',
        main = paste("uniform distribution on integers from ", low, "to ", high))
```

uniform distribution on integers from 0 to 10



```
unif.exp <- sum(values*probs)
paste("The expected value of uniform distribution is", unif.exp)
```

```
[1] "The expected value of uniform distribution is 5"
```

```
unif.var <- sum((unif.exp - values)^2*probs)
paste("The variance of uniform distribution is", unif.var)
```

```
[1] "The variance of uniform distribution is 10"
```

Exercise: experiment with the low and high values to see how the expectation and variance depend on them. Can you postulate a relationship without looking it up?

5.3.2 Binomial

Binary or Bernoulli trials have two discrete outcomes (mutant/wild-type, win/lose, etc.). The number of “successes” out of a sequence of n independent binary trials with probability of success p is described by the binomial distribution.

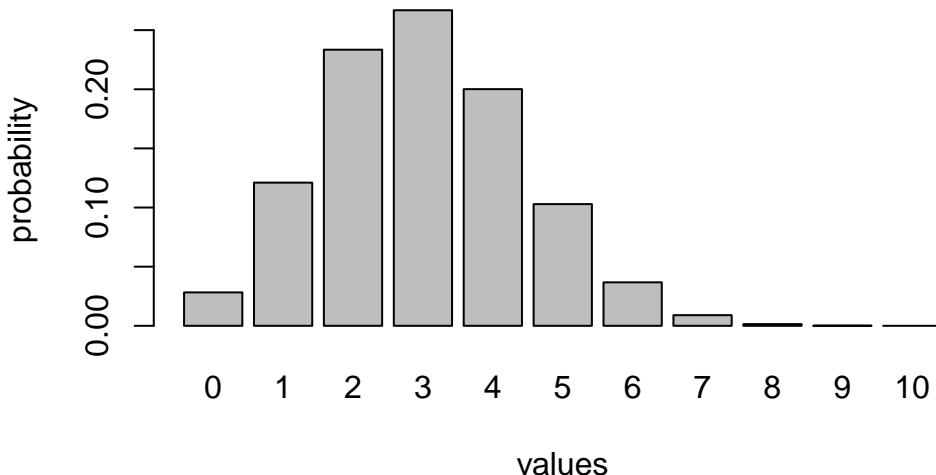
```
n <- 10 # the number of trials
p <- 0.3 # the probability of success in one trial
values <- 0:n # vector of discrete values of the binomial
```

```

probs <- dbinom(values, n, p)
barplot(probs, names.arg = values, xlab = 'values', ylab = 'probability',
        main = paste("binomial distribution with n=", n, "and p=", p))

```

binomial distribution with n= 10 and p= 0.3



```

bin.exp <- sum(values*probs)
paste("The expected value of binomial distribution is", bin.exp)

```

[1] "The expected value of binomial distribution is 3"

```

bin.var <- sum((bin.exp - values)^2*probs)
paste("The variance of binomial distribution is", bin.var)

```

[1] "The variance of binomial distribution is 2.1"

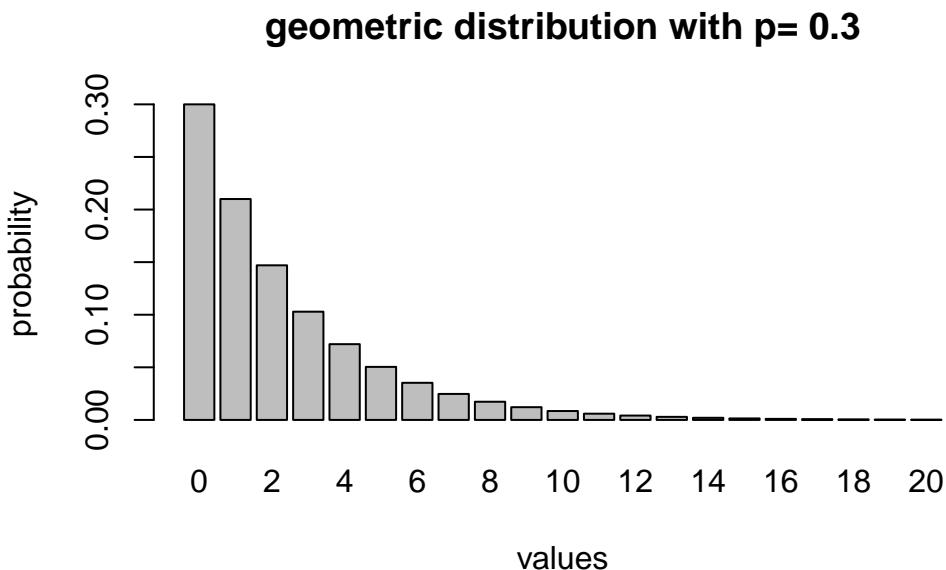
Exercise: Try different values of n and p and postulate a relationship with the expectation and variance.

5.3.3 Geometric

The random variable is the first “success” in a string of independent binary trials and the distribution describes the probability of any non-negative value. It may be pretty intuitive

that since all the trials have the same probability of success, the distribution will have a geometric (exponential) form—try to figure out the exact formula for the probability density without looking it up!

```
p <- 0.3 # the probability of success
low <- 0 # minimum value
high <- 20 # maximum value
values <- low:high # vector of discrete values of the RV
probs <- dgeom(values, p)
barplot(probs, names.arg = values, xlab = 'values', ylab = 'probability', main = paste("geometric distribution with p= 0.3"))
```



```
geom.exp <- sum(values*probs)
paste("The expected value of geometric distribution is", geom.exp)
```

```
[1] "The expected value of geometric distribution is 2.32030059650472"
```

```
geom.var <- sum((geom.exp - values)^2*probs)
paste("The variance of geometric distribution is", geom.var)
```

```
[1] "The variance of geometric distribution is 7.52697882945385"
```

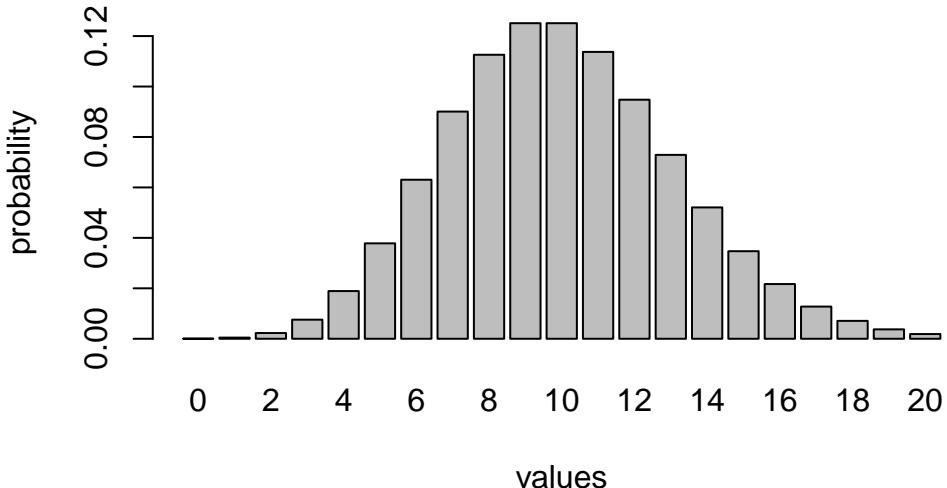
Exercise: Calculate the expectations and variances for different values of p and report how they are related.

5.3.4 Poisson

Suppose that there is a discrete process that occurs with some average rate λ , which describes the expected number of occurrences of these events in a unit of time. The Poisson random variable is the number of such occurrences, and the distribution describes the probability of any non-negative value.

```
low <- 0 # minimum value
high <- 20 # maximum value
lambda <- 10 # Poisson rate
values <- low:high # vector of discrete values of the RV
probs <- dpois(values, lambda)
barplot(probs, names.arg = values, xlab = 'values', ylab = 'probability',
        main = paste("Poisson distribution with lambda=", lambda))
```

Poisson distribution with lambda= 10



```
pois.exp <- sum(values*probs)
paste("The expected value of Poisson distribution is", pois.exp)
```

```
[1] "The expected value of Poisson distribution is 9.96545658024143"
```

```
pois.var <- sum((pois.exp - values)^2*probs)
paste("The variance of Poisson distribution is", pois.var)
```

```
[1] "The variance of Poisson distribution is 9.77875058489889"
```

Exercise: Calculate the expectations and variances for different values of λ and report how they are related.

5.4 Probability distribution examples (continuous)

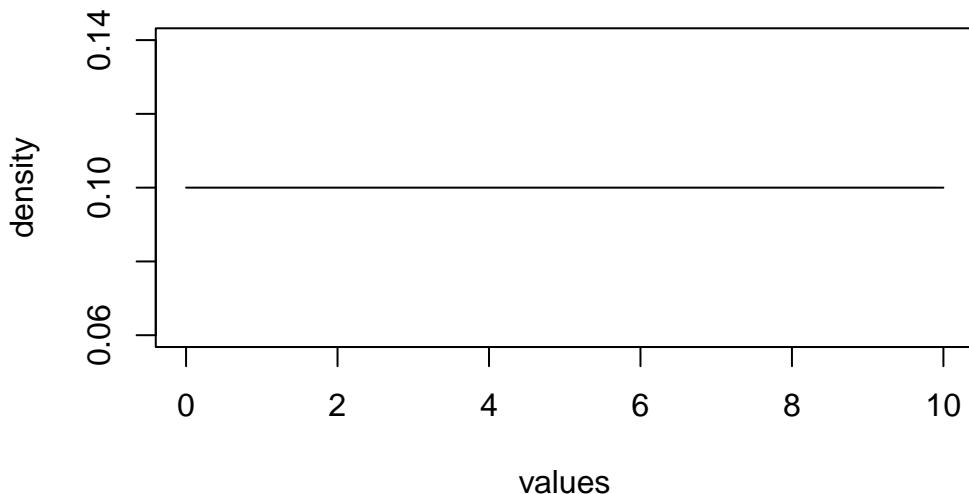
In the following examples with continuous variables we cannot calculate the means and variances directly from the density function. One way to do it is to produce a sample using the random number generator and calculate the mean and variance of that sample.

5.4.1 Uniform

The continuous equivalent of the discrete uniform distribution.

```
low <- 0 # minimum value
high <- 10 # maximum values
number <- 100
values <- seq(low, high, length.out = number) # vector of discrete values of the RV
probs <- dunif(values, min=low, max = high)
plot(values, probs, t='l', xlab = 'values', ylab = 'density',
     main = paste("Uniform distribution on interval from ", low, "to ", high))
```

Uniform distribution on interval from 0 to 10



```

n <- 1000 # sample size
unif.sample <- runif(n, low, high) # generate sample
unif.exp <- mean(unif.sample)
paste("The expected value of uniform distribution is", unif.exp)

```

[1] "The expected value of uniform distribution is 4.91734483863693"

```

unif.var <- var(unif.sample)
paste("The variance of uniform distribution is", unif.var)

```

[1] "The variance of uniform distribution is 8.1764677061904"

Exercise: experiment with the width of the interval to see how it affects the expectation and variance.

5.4.2 exponential

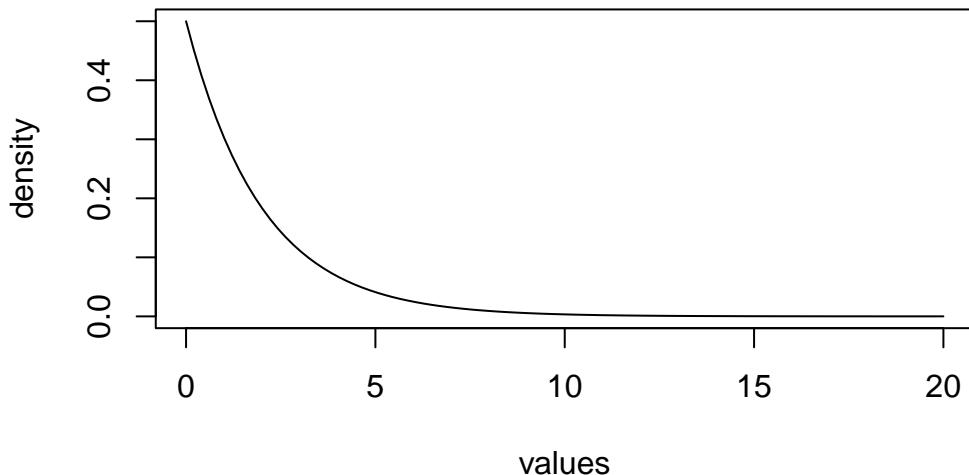
The random variable describes the length of time between independent discrete events occurring with a certain rate, like we saw in the Poisson distribution.

```

low <- 0 # minimum value
high <- 20 # maximum values
number <- 100
r <- 0.5
values <- seq(low,high,length.out = number) # vector of discrete values of the RV
probs <- dexp(values, r)
plot(values, probs, t='l', xlab = 'values', ylab = 'density',
     main = paste("Exponential distribution with rate=", r))

```

Exponential distribution with rate= 0.5



```
n <- 1000 # sample size  
exp.sample <- rexp(n, r) # generate sample  
exp.exp <- mean(exp.sample)  
paste("The expected value of exponential distribution is", exp.exp)
```

```
[1] "The expected value of exponential distribution is 1.95628141847689"
```

```
exp.var <- var(exp.sample)  
paste("The variance of exponential distribution is", exp.var)
```

```
[1] "The variance of exponential distribution is 3.77027126330144"
```

Exercise: What is the relationship between the rate and the expectation and variance?

5.4.3 normal distribution

The normal distribution, sometimes written $N(\mu, \sigma)$ comes up everywhere (e.g., in the limit of the Poisson distribution for large n). The two parameters are simply the mean and the standard deviation. The reason for its ubiquity is that it is that any sum of a large number of independent random variables converges to the normal, formalized by the Central Limit Theorem:

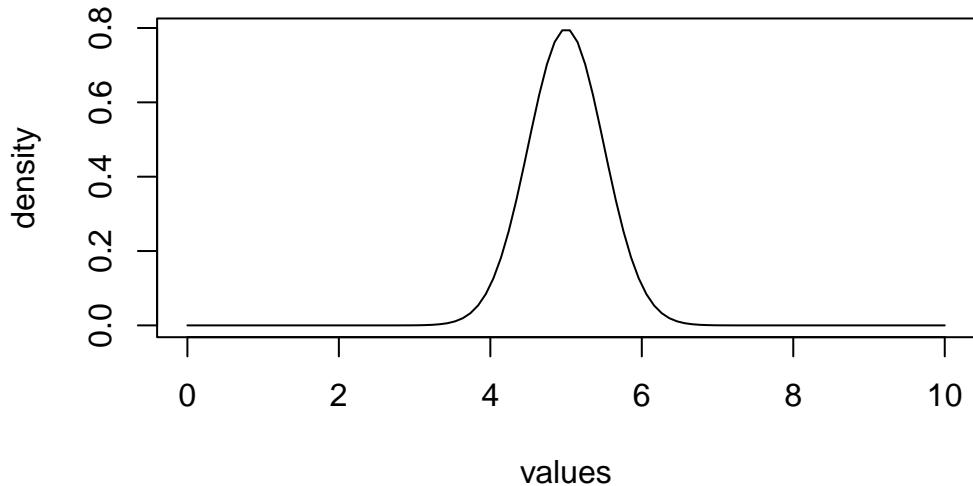
For a set of n IID random variables $\{X_i\}$ with mean μ and standard deviation σ , the sample mean \bar{X}_n has the property:

$$\lim_{n \rightarrow \infty} \frac{\bar{X}_n - \mu}{\sigma} = N(0, 1)$$

where $N(0, 1)$ stands for the normal distribution with mean 0 and standard deviation 1.

```
low <- 0 # minimum value
high <- 10 # maximum values
number <- 100
mu <- 5
sigma <- 0.5
values <- seq(low, high, length.out = number) # vector of discrete values of the RV
probs <- dnorm(values, mu, sigma)
plot(values, probs, t='l', xlab = 'values', ylab = 'density',
     main = paste("Normal distribution with mean=", mu, "and sigma=", sigma))
```

Normal distribution with mean= 5 and sigma= 0.5



```
n <- 1000 # sample size
norm.sample <- rnorm(n, mu, sigma) # generate sample
norm.exp <- mean(norm.sample)
paste("The expected value of normal distribution is", norm.exp)
```

```
[1] "The expected value of normal distribution is 4.98332040872432"
```

```

norm.var <- var(norm.sample)
paste("The variance of normal distribution is", norm.var)

```

```
[1] "The variance of normal distribution is 0.242249732142474"
```

5.5 Application of normal distribution: confidence intervals

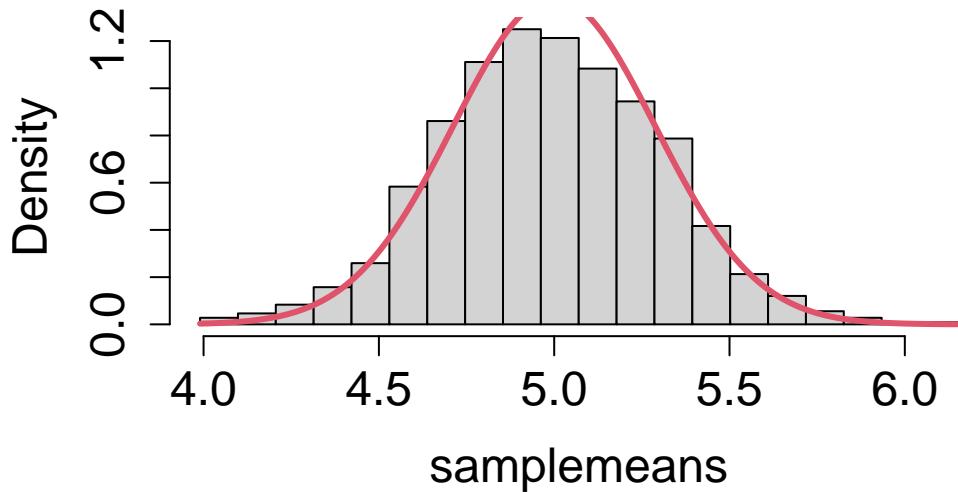
The most important use of the normal distribution has to do with estimation of means, because the normal distribution describes the *sampling distributions* of means of IID samples. The mean of that sampling distribution is the mean of the population distribution that is being sampled, and the standard deviation is called the *standard error* and is related to the standard deviation of the population σ_X as follows: $\sigma_{SE} = \sigma/n$, where n is the sample size.

```

numsamples <- 1000
size <- 100
# compute mean for different samples
samplemeans <- replicate(n = numsamples, mean(sample(0:10, size, replace = TRUE)))
break_points <- seq(min(samplemeans), max(samplemeans),
                      (max(samplemeans) - min(samplemeans)) / 20)
hist(samplemeans, breaks = break_points, freq = FALSE,
      cex.axis = 1.5, cex.lab = 1.5,
      main= '1000 means of samples of size 100')
sigma <- 10 / sqrt(12) / sqrt(size)
mu <- 5
range <- seq(min(samplemeans), max(samplemeans), sigma / 100)
lines(range,
      dnorm(range, mu, sigma),
      t = 'l', lwd = 3, col = 2, lty = 1, cex.axis = 1.5, cex.lab = 1.5)

```

1000 means of samples of size 100



Exercise: Try using different distributions from above and see if the sample means still converge to the normal distribution.

The following script calculates a confidence interval based on a sample.

```
# Computing confidence intervals
qnorm(0.5) # the value that divides the density function in two

[1] 0

qnorm(0.95) # the value such that 95% of density is to its left

[1] 1.644854

size <- 100 # sample size
alpha <- 0.95 # significance level
sample <- runif(size)
s <- sd(sample) / sqrt(size) # standard error
z <- qnorm((1 - alpha) / 2) # z-value
left <- mean(sample) + s * z
right <- mean(sample) - s * z
print(right)
```

```
[1] 0.5449493
```

```
print(left)
```

```
[1] 0.4406134
```

Exercise: Modify that script to report whether the confidence interval captures the true mean. Use a loop structure (as in the script above) to generate 1000 sample means and report how many of them are within the theoretical confidence interval. Does this match the fraction you expect from the significance level? Try different significance levels and sample sizes and report what you discover.

5.6 Identifying type of distribution in real data

Let us consider the penguin data set again:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts -----
x dplyr::filter() masks stats::filter()
x dplyr::lag()   masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become non-conflicting
```

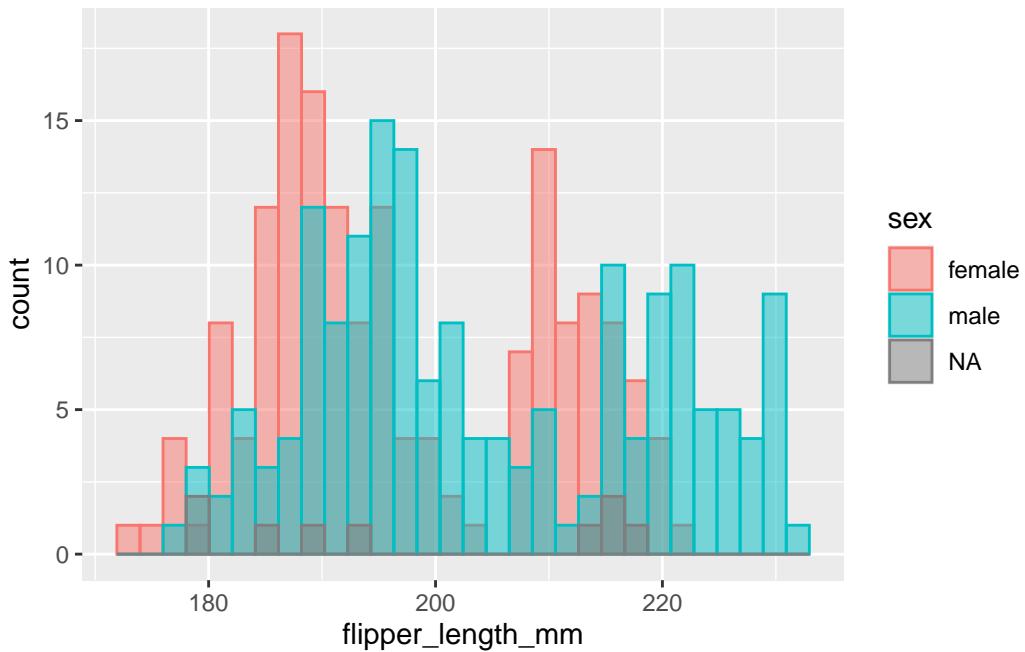
```
library(palmerpenguins)
str(penguins)
```

```
tibble [344 x 8] (S3:tbl_df/tbl/data.frame)
$ species      : Factor w/ 3 levels "Adelie","Chinstrap",...: 1 1 1 1 1 1 1 1 1 ...
$ island       : Factor w/ 3 levels "Biscoe","Dream",...: 3 3 3 3 3 3 3 3 3 ...
$ bill_length_mm: num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
$ bill_depth_mm: num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
```

```
$ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
$ body_mass_g      : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
$ sex              : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2 NA NA ...
$ year             : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

A simple way to visualize a distribution is to plot a histogram: data are binned, and the height of the bin represents counts (or frequencies). Here are the histograms of distributions of flipper lengths of all the species of penguins separated by sex:

```
ggplot(penguins) +
  aes(x = flipper_length_mm, color = sex, fill=sex) + geom_histogram(alpha = 0.5, position = "stack") +
  stat_bin() `using `bins = 30` . Pick better value with `binwidth` .
Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```

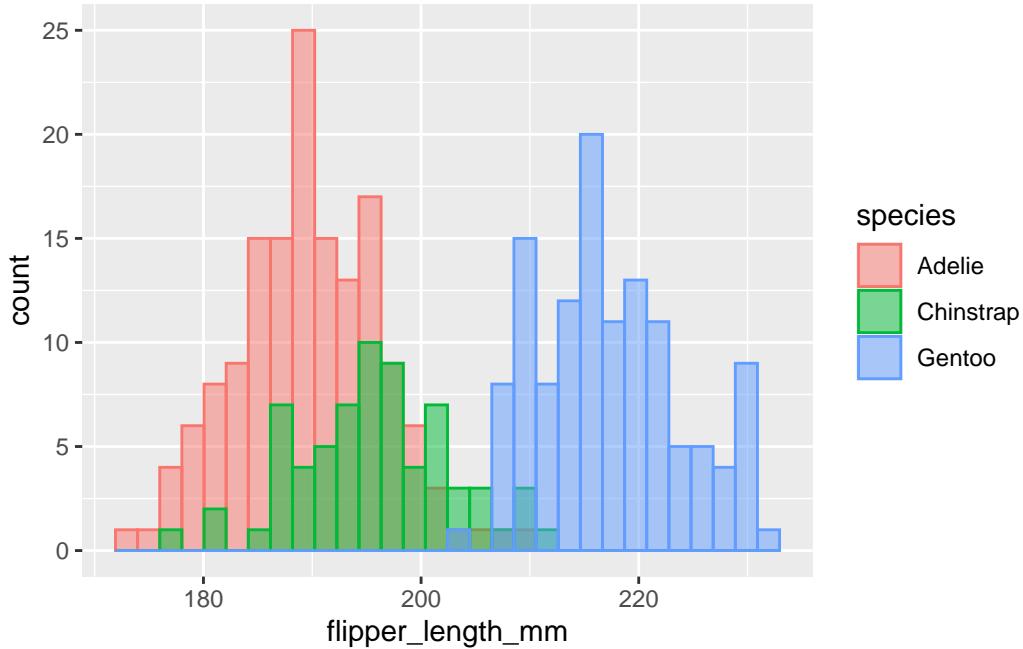


And here are the histograms of flipper lengths separated by species:

```
ggplot(penguins) +
  aes(x = flipper_length_mm, color = species, fill=species) + geom_histogram(alpha = 0.5,
```

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

```
Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```



To decide systematically which of these distributions are closer to a theoretical distribution is via a Quantile-Quantile (QQ) plot, which plots the quantile value from a sample against the quantiles from a given distribution with best-fit parameters. If the data were to follow the distribution closely, you should find all the points lying on the identity line. For example, here is how to compare a data set drawn from the normal random number generator with the normal distribution:

```
library(fitdistrplus)
```

```
Loading required package: MASS
```

```
Attaching package: 'MASS'
```

```
The following object is masked from 'package:dplyr':
```

```
select
```

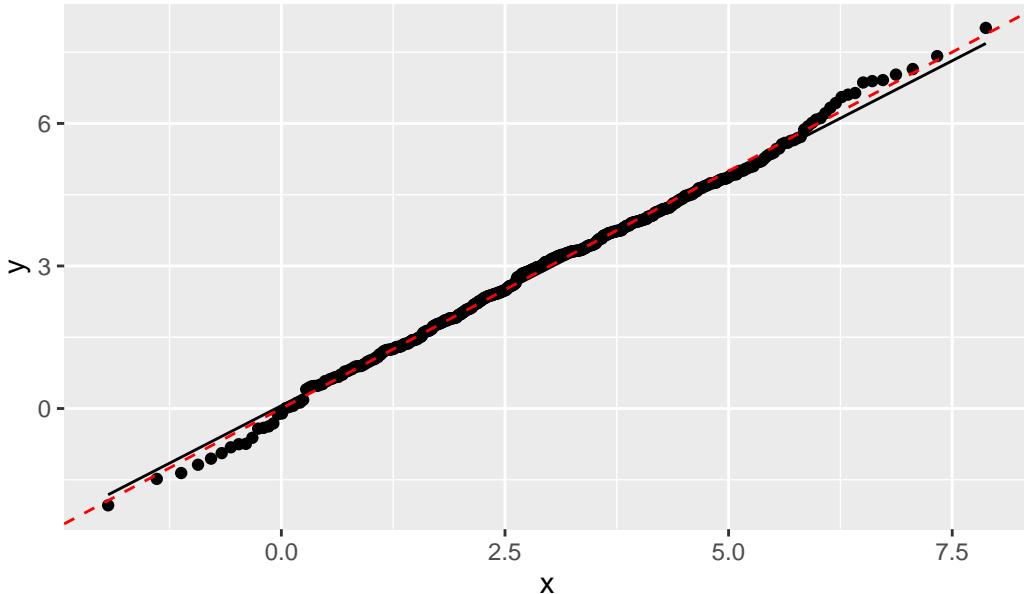
```
Loading required package: survival
```

```
test_data <- tibble(x = rnorm(n = 500, mean = 3, sd = 1.5))
# example: find best-fitting Normal
my_normal <- fitdistr(test_data$x, densfun = "normal")
# note the slight discrepancies
print(my_normal)
```

```
mean           sd
2.96959230   1.58764598
(0.07100169) (0.05020577)
```

```
ggplot(test_data, aes(sample = x)) +
  stat_qq(distribution = qnorm, dparams = my_normal$estimate) +
  stat_qq_line(distribution = qnorm, dparams = my_normal$estimate) +
  geom_abline(intercept = 0, slope = 1, linetype = 2, col = "red") +
  ggtitle("Q-Q plot assuming best-fitting Normal distribution")
```

Q–Q plot assuming best–fitting Normal distribution



Now let us assess the “normality” of the flipper length data separated by sex and separated by species:

```

dataset <- penguins %>% dplyr::filter(sex == 'female') %>% drop_na() %>% dplyr::select(fli
my_normal <- fitdistr(x = as_vector(dataset), densfun = "normal")
# note the slight discrepancies
print(my_normal)

```

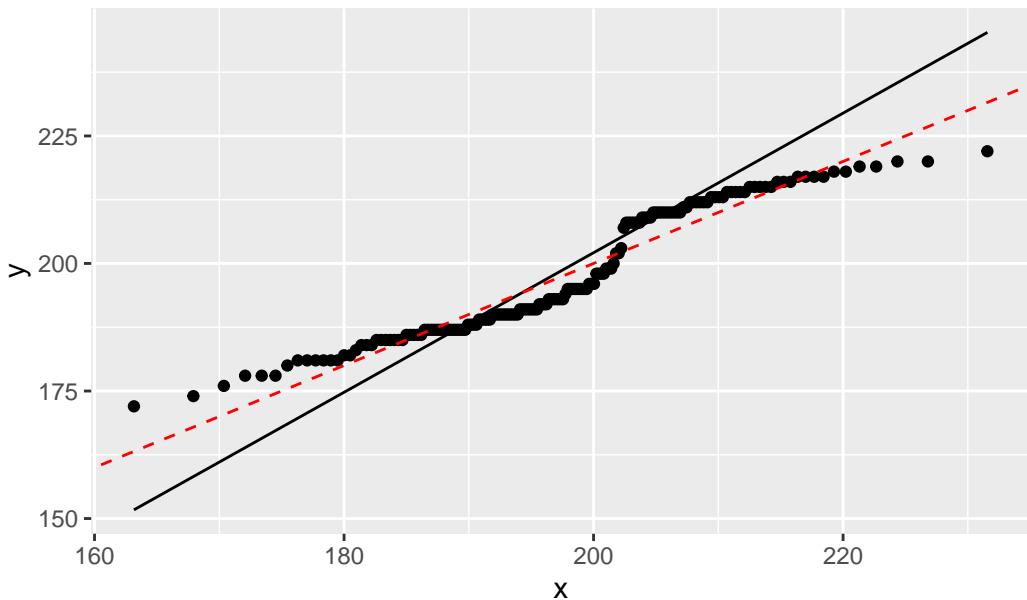
mean	sd
197.3636364	12.4628373
(0.9702306)	(0.6860566)

```

ggplot(dataset, aes(sample = flipper_length_mm)) +
  stat_qq(distribution = qnorm, dparams = my_normal$estimate) +
  stat_qq_line(distribution = qnorm, dparams = my_normal$estimate) +
  geom_abline(intercept = 0, slope = 1, linetype = 2, col = "red") +
  ggtitle("Q-Q plot assuming best-fitting Normal distribution of flipper length for female")

```

Q–Q plot assuming best–fitting Normal distribution of flipper length for female



```

dataset <- penguins %>% dplyr::filter(sex == 'male') %>% drop_na() %>% dplyr::select(fli
my_normal <- fitdistr(x = as_vector(dataset), densfun = "normal")
# note the slight discrepancies
print(my_normal)

```

mean	sd
------	----

```

204.5059524    14.5045137
( 1.1190475) ( 0.7912861)

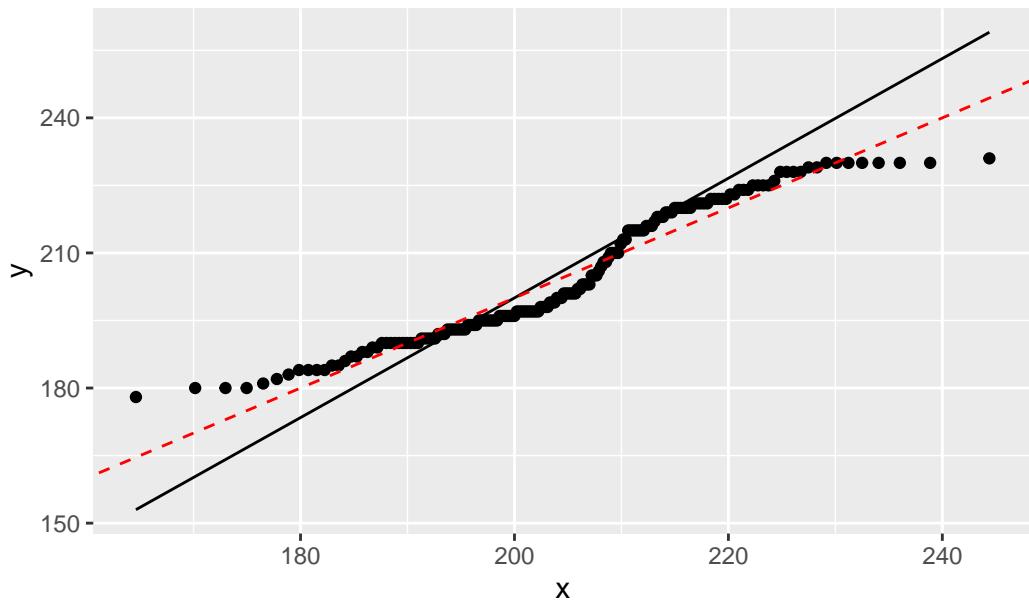
```

```

ggplot(dataset, aes(sample = flipper_length_mm)) +
  stat_qq(distribution = qnorm, dparams = my_normal$estimate) +
  stat_qq_line(distribution = qnorm, dparams = my_normal$estimate) +
  geom_abline(intercept = 0, slope = 1, linetype = 2, col = "red") +
  ggtitle("Q-Q plot assuming best-fitting Normal distribution of flipper length for male penguins")

```

Q-Q plot assuming best-fitting Normal distribution of flipper length for male penguins



In contrast with the simulated data, here the data points and the black line that attempts to capture them is quite different from the identity line (red). This means this distribution is not normal, as can be seen from the histograms of the flipper lengths grouped by sex.

6 Hypothesis testing

A large number of scientific questions can be expressed as an hypothesis test—essentially a yes/no question, such as “are two samples drawn from distributions with the same mean?”, or “Is the frequency of an allele in a population greater than 0.1?”. Several tests have been developed, each with a specific type of question in mind. There is a dangerous tendency to view statistics as a collection of tests, and to practice it by plugging in your data set into the correct test, expecting that the test will spit out the correct decision. The purpose of this lesson is to demonstrate that using and interpreting statistical tests requires careful thinking to avoid serious errors.

6.1 Test results vs. the truth

A statistical test begins by stating the **null hypothesis**, usually one that is expected, or that shows no effect: for example, that two samples come from a distribution with the same mean, or that a rare allele has frequency of less than 0.1. One may state the **alternative hypothesis explicitly**, although it’s usually the logical converse of the null, i.e., the two samples have different population means, or the allele has frequency greater than 0.1.

After the hypothesis is stated, the data are collected and are used to test the hypothesis. By default, the null hypothesis is assumed to be true, and the test assesses whether the data provide sufficient evidence against the null hypothesis—in which case the **null hypothesis is rejected**. There is an adversarial relationship: either the data knock off the hypothesis, or else they fail to do so. Standard terminology reflects this somewhat counter-intuitive setup: rejecting the null hypothesis is called a **positive test result**, while not rejecting it is called a **negative result**.

The fundamental assumption of this process is that the truth value of the hypothesis is set prior to the collection of data. For example, if one could observe all of the genomes, the frequency of the allele would be known exactly, so this truth exists prior to the hypothesis testing. Because we typically can only observe a sample (and not the entire universe of data), we might end up erroneously rejecting the null hypothesis when it is in fact true, or not rejecting it when it is in fact false. The possible outcomes of a test can be organized in the table:

H0	True	False
Reject	False Positive	True Positive
Not Reject	True Negative	False Negative

The values at the top describe the truth status of the hypothesis, while the decisions in the left column are the result of using data to test the hypothesis. Note: the words false and true in describing the test result do not refer to the hypothesis, but to whether the result is correct! For example, if the frequency of the allele were 0.09 but the test for the hypothesis that the frequency is less than 0.1 resulted in rejecting that hypothesis, that would be a false positive result (the null hypothesis is true but the test rejected it.)

6.2 Types of errors

As mentioned above, sometimes a hypothesis test makes the wrong decision, which is called an error. There are two different kinds of errors: rejecting a true null hypothesis, called a Type I error, and not rejecting a false null hypothesis, called a Type II error.

Example: In the case above of testing for the same mean: if the samples are taken from distributions with the same mean, but the hypothesis is rejected, this is called a false positive (Type I error). If the samples come from distributions with different means, but the hypothesis is not rejected, this is called a false negative (Type II error.)

As a scientist, would you rather make a Type I error (make an erroneous discovery), or a Type II error (fail to make a discovery)?

6.3 Test parameters and p-values

The **sensitivity** of a test is the probability of obtaining the positive result, given a false hypothesis; and the **specificity** of a test is the probability of obtaining the negative result, given a true hypothesis. The *Type I error rate* is the probability of obtaining the positive result, given a true hypothesis (complementary to specificity), and the *Type II error rate* is the probability of obtaining the negative result, given a false hypothesis (complementary to sensitivity).

All four parameters (rates) of a binary test are summarized as follows:

$$\text{Sen} = \frac{TP}{TP + FN}; \text{ Spec} = \frac{TN}{TN + FP}$$

$$\text{FPR} = \frac{FP}{TN + FP}; \text{ FNR} = \frac{FN}{TP + FN}$$

The notation TP, FP, etc. represents the frequency or count of true positives, false positives, etc., out of a large number of experiments with known truth status of the hypothesis.

Knowledge of sensitivity and specificity determine the Type I and Type II error rates of a test since they are complementary events.

Of course, it is desirable for a test to be both very sensitive (reject false null hypotheses, detect disease, convict guilty defendants) and very specific (not reject true null hypotheses, correctly identify healthy patients, acquit innocent defendants), but no test is perfect, and sometimes it makes the wrong decision. This is where statistical inference comes into play: given some information about these parameters, a statistician can calculate the error rate in making different decisions.

The probability that a given data set is produced from the model of the null hypothesis is called the **p-value** of a test. More precisely:

For a given data set D and a null hypothesis H_0 , the *p-value* is the probability of obtaining a result *as far from expectation or farther than the observed data, given the null hypothesis*.

The p-value is the most used, misused, and even abused quantity in statistics, so please think carefully about its definition. One reason this notion is frequently misused is because it is very tempting to conclude that the p-value is the probability of the null hypothesis being true, based on the data. That is not the case! The definition has the opposite direction of conditionality—we assume that the null hypothesis is true, and based on that calculate the probability of obtaining a pattern as extreme or more extreme than what observed in the data. There is no way (according to classical “frequentist” statistics) of assigning a probability to the truth of a hypothesis, because it is not the result of an experiment.

Typically, one sets a critical threshold bounding the probability of making a Type I error in a test to a “small” number (often, $\alpha = 0.05$ or 0.01), and calls the result of a test “significant” if the p-value is less than α .

For example, consider samples of size n taken from two normal distributions (with unobserved means μ_1, μ_2). We can generate the data:

```
generate_samples <- function(n, mu1, mu2){  
  return(data.frame(sample1 = rnorm(n = n, mean = mu1, sd = 1),  
                    sample2 = rnorm(n = n, mean = mu2, sd = 1)))  
}  
  
my_sample <- generate_samples(1000, 1, 1.01)
```

and use a Student’s t-test to probe whether the means differ:

```

# two-tailed (diff in means = 0)
# Student's (assumes equal variances)
# (for Welch's t-test, var.equal = FALSE)
t.test(my_sample$sample1,
       my_sample$sample2,
       var.equal = TRUE)

```

Two Sample t-test

```

data: my_sample$sample1 and my_sample$sample2
t = 1.4743, df = 1998, p-value = 0.1406
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.0217314 0.1533310
sample estimates:
mean of x mean of y
1.024352 0.958552

```

Exercise: Can you detect a “significant difference in means” (assuming $\alpha = 0.05$)? What if you take a much larger sample? What if the difference in means is more pronounced?

6.4 Multiple comparisons

What if we were to produce several samples? E.g., measure difference between males and females reflectance in birds at several locations? Suppose that in fact the reflectance is the same for male and female ($\mu_1 = \mu_2 = 1$), that for each location we capture and measure 10 males and 10 females, and that we repeat this across 2500 locations.

First, let’s write a little function that returns the p-values for the t-test

```

get_p_value_t_test <- function(my_sample){
  test_results <- t.test(my_sample$sample1,
                        my_sample$sample2,
                        var.equal = TRUE)
  return(test_results$p.value)
}

```

and now simulate the data:

```
pvalues <- replicate(n = 2500,  
                      expr = get_p_value_t_test(generate_samples(10, 1, 1)))
```

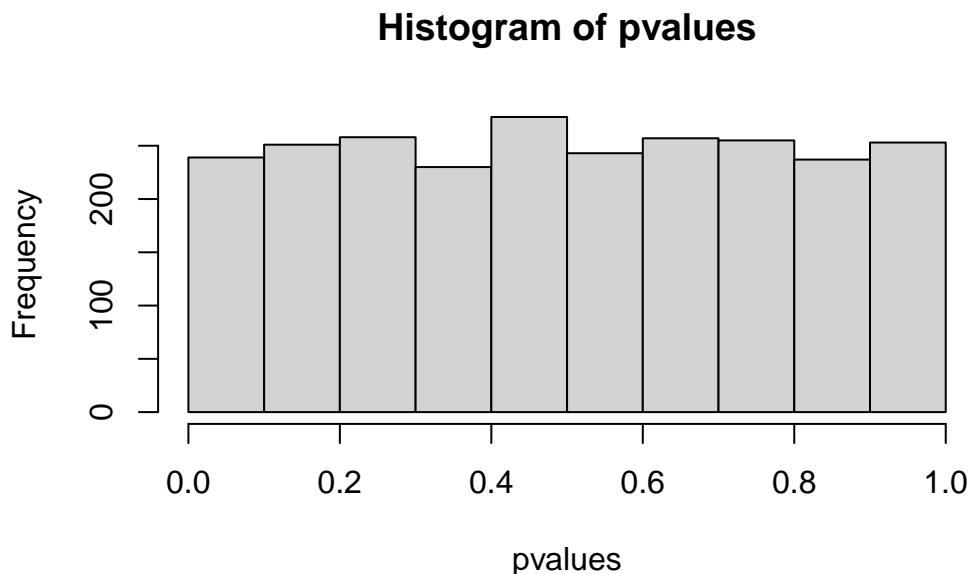
How many times do we detect a “significant difference in reflectance” when setting $\alpha = 0.05$ (even though we know that males and females are sampled from the same distribution)?

```
sum(pvalues < 0.05)
```

```
[1] 127
```

You should get a number of “significant” tests that is about $2500 \cdot 0.05 = 125$. In fact, the distribution of p-values when the data are sampled from the null hypothesis is approximately uniform:

```
hist(pvalues)
```



This means that when you are performing multiple tests, some will turn out to find “significant” differences even when there are none. Again, this is better summarized by xkcd:

Exercise: what happens to the distribution of p-values if the means are quite different (e.g., $\mu_1 = 1$, $\mu_2 = 0.9$)?

6.5 Corrections for multiple comparisons

The main approach to deal with the problem of multiple comparisons is to adjust the p-values. For example, in Bonferroni correction one consider as significant test results whose associated p-value is $\leq \alpha/n$, where n is the number of tests performed (equivalently, redefine the p-values as $p' = \min(pn, 1)$). Clearly, this correction becomes overly conservative when the number of tests is large. For example, in biology:

- **Gene expression** In a typical microarray experiment, we contrast the differential expression of tens of thousands of genes in treatment and control tissues.
- **GWAS** In Genomewide Association Studies we want to find SNPs associated with a given phenotype. It is common to test tens of thousands or even millions of SNPs for significant associations.
- **Identifying binding sites** Identifying candidate binding sites for a transcriptional regulator requires scanning the whole genome, yielding tens of millions of tests.

The funniest example of this problem is the fMRI of the [dead salmon](#): a dead salmon “was shown a series of photographs depicting human individuals in social situations with a specified emotional valence. The salmon was asked to determine what emotion the individual in the photo must have been experiencing.” The researchers showed that if multiple comparisons were not accounted for, one would detect a cluster of active voxels in the brain, with a cluster-level significance of $p = 0.001$.

The widespread use of GWAS and other techniques that are trying to find a needle in a haystack led to the development of many interesting techniques. [Here](#) an interesting account.

Adjusting p-values in R:

```
original_pvalues <- c(0.01, 0.07, 0.1, 0.44)
p.adjust(original_pvalues, method = "bonferroni")
```

```
[1] 0.04 0.28 0.40 1.00
```

6.6 Two problems with science

6.6.1 Selective reporting

We have seen above that setting $\alpha = 0.05$ means that we are going to make false discoveries at this rate. In science, we prefer publishing positive results—negative results are difficult to publish and attract little attention. Suppose that 20 research groups around the world set out to test the same hypothesis, which is false. Then there is a good chance at least one group will

reject the null hypothesis, and pursue publication for their “discovery”. The tendency to put negative studies in the files drawer and forget about them causes the so called **publication bias** (aka **selective reporting**): by favoring positive results over negative ones, we greatly increase the chance that our conclusions are wrong. Note that these would cause the results of the paper to be largely impossible to reproduce, and the **reproducibility crisis in the sciences** is partially due to selective reporting.

6.6.2 P-hacking

One big violation of good experimental design is known as p-value “**“fishing”** (or **p-hacking**): repeating the experiment, or increasing the sample size, until the p-value is below the desired threshold, and then stopping the experiment. Using such defective design dramatically lowers the likelihood that the result is a true positive. And of course there is actual fraud, or fudging of data, which contributes to some bogus results.

An insidious cousin of p-hacking was dubbed by Andrew Gelman “**the garden of forking paths**” in this [paper](#). The issue arises in complex problems with multi-variable noisy datasets (aren’t all interesting ones like that?) Essentially, with many choices and degrees of freedom in a problem, it is easy to convince yourself that the choice you made (data cleaning, parameter combinations, etc.) is the correct one because it gives the strongest results. Without a clearly stated hypothesis, experimental design, and data processing details prior to data collection, this enchanted garden can lead even a well-intentioned researcher astray.

6.7 Readings

Good readings on these and related issues:

- [Why Most Published Research Findings Are False](#)
- [Decline effect](#)
- [The truth wears off](#)
- [The Extent and Consequences of P-Hacking in Science](#)
- [A manifesto for reproducible science](#)
- [Spoiled Science](#)

6.8 How to fool yourself with p-hacking (and possibly get fired!)

We are going to try our hand at p-hacking, to show how easy it is to get fooled when you have a sufficiently large and complex data set. The file `data/medals.csv` contains the total number of medals won at the Olympic games (Summer or Winter) by country, sport and gender. We have a simple, and reasonable (?) hypothesis: because the amount of money

available to Olympic teams is finite, whenever a country invests in the male team, this will be at the detriment of the female team. To test this hypothesis, we measure whether the number of medals won by a national female team in a year is negatively correlated with the number of medals won by the male team.

Let's read the data, and take a peak:

```
library(tidyverse)

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr    1.3.0
v purrr    1.0.2

-- Conflicts -----
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become non-conflicting.

dt <- read_csv("data/medals.csv")

Rows: 6915 Columns: 5
-- Column specification -----
Delimiter: ","
chr (2): NOC, Sport
dbl (3): Year, F, M

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

dt

# A tibble: 6,915 x 5
  NOC    Year Sport      F     M
  <chr> <dbl> <chr>    <dbl> <dbl>
1 AFG    2008 Taekwondo 0     1
2 AFG    2012 Taekwondo 0     1
3 AHO    1988 Sailing   0     1
4 ALG    1984 Boxing    0     2
```

```

5 ALG    1992 Athletics    1    0
6 ALG    1992 Boxing      0    1
7 ALG    1996 Athletics    0    1
8 ALG    1996 Boxing      0    2
9 ALG    2000 Athletics    1    3
10 ALG   2000 Boxing     0    1
# i 6,905 more rows

```

First, let's see whether our hypothesis works for the whole data:

```
cor(dt$F, dt$M)
```

```
[1] 0.1651691
```

The correlation is positive: more medals for the men tend to correspond to more medals for the women. This correlation is not very strong, but is it “significant”? We can run a correlation test:

```
cor.test(dt$F, dt$M)
```

```

Pearson's product-moment correlation

data: dt$F and dt$M
t = 13.924, df = 6913, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.1421521 0.1880075
sample estimates:
cor
0.1651691

```

Indeed! The confidence intervals are far from 0: the correlation is definitely positive. Should we give up? Of course not! Just as for the jelly beans, we can p-hack our way to glory by subsetting the data. We are going to test each discipline independently, and see whether we can get a robustly negative correlation for any discipline. Because we are serious scientists, we are going to consider only disciplines for which we have at least 50 data points, to avoid results that are due to small sample sizes. Let's write a code:

```

dt <- dt %>% group_by(Sport) %>% mutate(sample_size = n()) %>% ungroup()
correlations <- dt %>%
  filter(sample_size >= 50) %>%
  group_by(Sport) %>%
  summarise(cor = cor(`M`, `F`),
            pvalue = cor.test(`M`, `F`)$.p.value) %>%
  ungroup()

```

Now let's see whether there are highly significant negative correlations:

```

my_results <- correlations %>% filter(pvalue < 0.05, cor < 0)
my_results

```

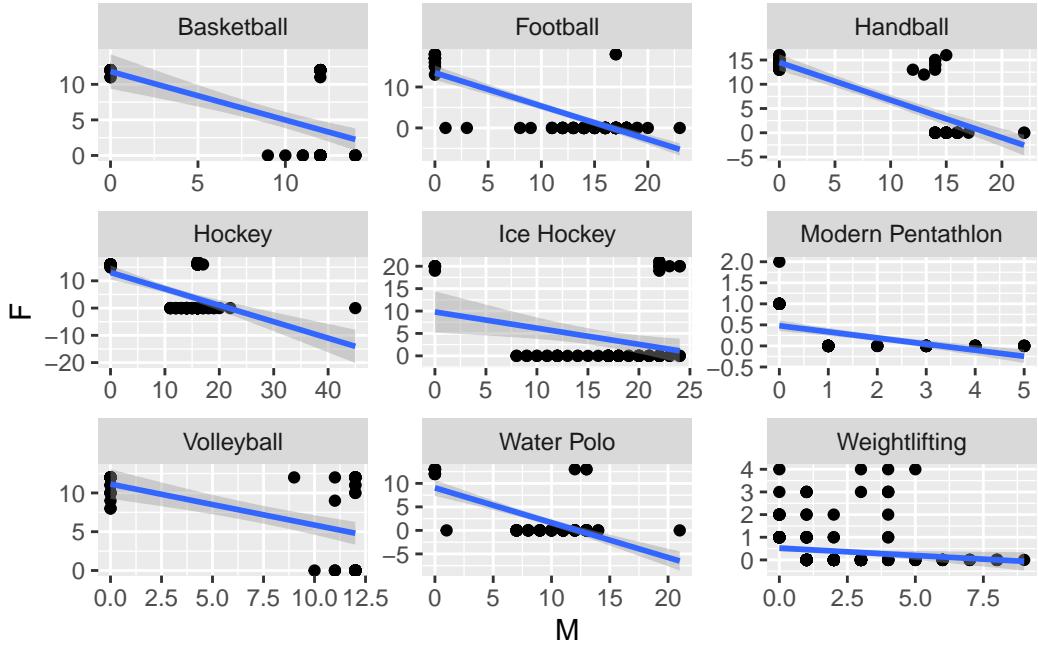
	Sport	cor	pvalue
	<chr>	<dbl>	<dbl>
1	Basketball	-0.579	7.86e- 8
2	Football	-0.796	6.75e-23
3	Handball	-0.810	5.28e-16
4	Hockey	-0.585	4.16e- 9
5	Ice Hockey	-0.302	8.10e- 3
6	Modern Pentathlon	-0.561	3.57e- 8
7	Volleyball	-0.545	2.18e- 6
8	Water Polo	-0.688	3.41e-14
9	Weightlifting	-0.138	2.33e- 2

Let's plot our results to convince ourselves that they are strong:

```

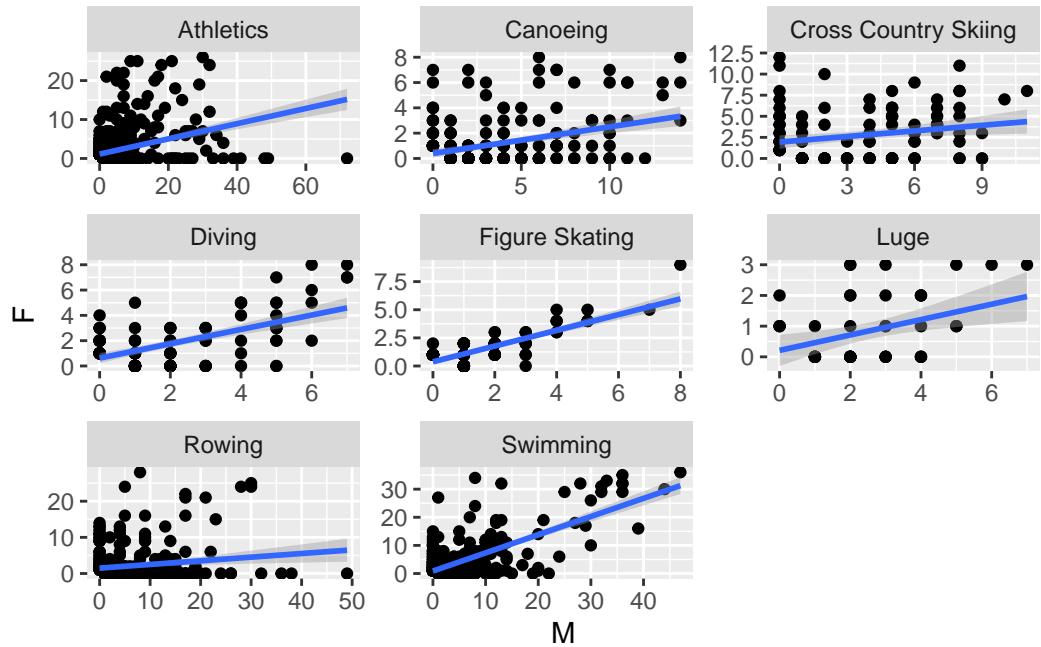
ggplot(dt %>% inner_join(my_results)) +
  aes(x = `M`, y = `F`) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(~Sport, scales = "free")

```



That's it! Should we rush to publish our results? Not quite: we have p-hacked our way to some highly significant results, but we did not correct for the number of tests we've made, and what we would do is to selectively reporting our strong results. In fact, we can do something very simple to convince ourselves that our results do not make much sense: just run the code again, but reporting significant positive correlations...

```
my_results <- correlations %>% filter(pvalue < 0.05, cor > 0)
ggplot(dt %>% inner_join(my_results)) +
  aes(x = `M`, y = `F`) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(~Sport, scales = "free")
```



You can see that we've got about the same number of sports testing significant for positive correlation! **Bonus question** what about figure skating?

7 Likelihood and Bayes

- understand the difference between likelihood and probability
- maximum likelihood estimation
- calculate positive predictive value of a hypothesis test
- interpret the results of Bayesian inference

7.1 Likelihood and estimation

7.1.1 likelihood vs. probability

In everyday English, probability and likelihood are synonymous. In probability and statistics, however, the two are distinct, although related, concepts. The definition of likelihood is based on the notion of conditional probability that we defined in week 2, applied to a data set and a particular probability model M :

$$L(M \mid D) = P(D \mid M)$$

The model is based on a set of assumptions that allow us to calculate probabilities of outcomes of a random experiment, typically a random variable with a well-defined probability distribution function.

Example. M may represent the binomial random variable, based on the assumptions that the data are strings of n independent binary outcomes with a set probability p of “success.” We then have the following formula for the probability of obtaining k successes:

$$P(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Suppose we think we have a fair coin and we flip it ten times and obtain 4 heads and 6 tails. Then the likelihood of our model (a binomial random variable with $p = 0.5$ with $n = 10$) based on our data ($k = 4$) is:

$$L(p = 0.5, n = 10 \mid k = 4) = P(k = 4 \mid n = 10, p = 0.5) = \binom{10}{4} 0.5^4 (0.5)^6$$

To calculate this precisely, it is easiest to use the R function `dbinom()`:

```
print(dbinom(4,10,0.5))
```

```
[1] 0.2050781
```

So the likelihood of this data set being produced by a fair coin is about 20.5%.

This certainly looks like a probability — in fact we calculated it from a probability distribution function, so why do we call it a likelihood? There are two fundamental differences between the two, one mostly abstract, the other more grounded.

First, a model (or model parameters) is not a random variable, because it comes from an assumption we made in our heads, not from an outcome of a random process. This may seem to be an abstract, almost philosophical distinction, but how would you go about assigning probabilities to all the models one can come up with? Would they vary from person to person, because one may prefer to use the binomial random variable, and another prefers Poisson? You see how this can get dicey if we think of these in terms of the traditional “frequency of outcomes” framework of probability.

Second, and more quantitatively relevant, is that likelihoods do not satisfy the fundamental axiom of probability: they do not add up to one. Remember that probabilities were defined on a sample space of all outcomes of a random experiment. Likelihoods apply to models or their parameters, and there are usually uncountably many models - in fact it's not possible to even describe all the possible models in vague terms! Even if we agree that we're evaluating only one type of model, e.g. the binomial random variable, the likelihood parameter p does not work like a probability, because there is a non-zero likelihood for any value p (technically, the coin could have any degree of unfairness!) so adding up all of the likelihoods will result in infinity.

7.1.2 maximizing likelihood

One of the most common applications of likelihood is to find the model or model parameters that give the highest likelihood based on the data, and call those the best statistical estimate. Here are the symbols we will use in this discussion:

- D : the observed data
- θ : the free parameter(s) of the statistical model
- $L(\theta | D)$: the likelihood function, read “the likelihood of θ given the data”
- $\hat{\theta}$: the maximum-likelihood estimates (m.l.e.) of the parameters

7.1.3 discrete probability distributions

The simplest case is that of a probability distribution function that takes discrete values. Then, the likelihood of θ given the data is simply the probability of obtaining the data when parametrizing the model with parameters θ :

$$L(\theta | D) = P(X = D | \theta)$$

Finding the m.l.e. of θ simply means finding the value(s) maximizing the probability of obtaining the given data under the model. In cases when this likelihood function has a simple algebraic form, we can find the maximum value using the classic method of taking its derivative and setting it to zero.

Example. Let's go back to the binomial example. Based on the data set of 4 heads out of 10 coin tosses, what is the maximum likelihood estimate of the probability of a head p ? The range of values of p is between 0 and 1, and since we have a functional expression for $P(k = 4; n = 10, p)$ (see above) we can plot it using the `dbinom()` function:

```
library(tidyverse)

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.1.2     v readr      2.1.4
v forcats   1.0.0     v stringr    1.5.0
v ggplot2   3.4.3     v tibble     3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()   masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

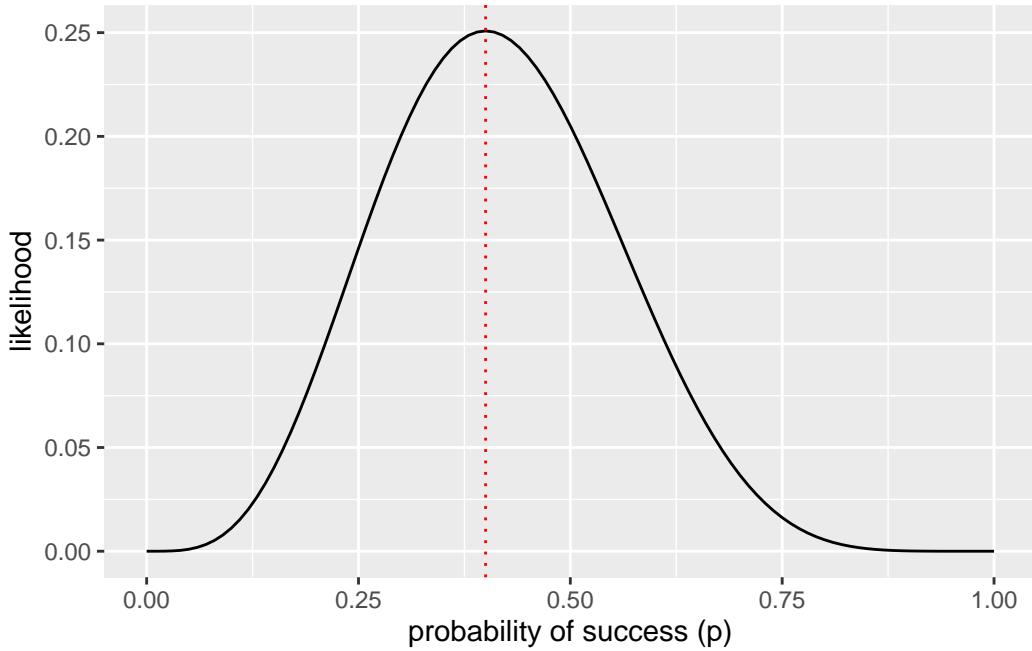


```
n <- 10
k <- 4
pl <- ggplot(data = data.frame(x = 0, y = 0)) + xlim(c(0,1))
like_fun <- function(p) {
  lik <- dbinom(k, n, p)
  return(lik)
}
pl <- pl + stat_function(fun = like_fun) +
  xlab('probability of success (p)') +
  ylab('likelihood') +
```

```

geom_vline(xintercept = 0.4, linetype='dotted', color = 'red')
show(pl)

```



It's probably not surprising that the maximum of the likelihood function occurs at $p = 0.4$, that is the observed fraction of heads! Using the magic of derivatives, we can show that for a data set with k success out of n trials, the maximum likelihood value of p is $\hat{p} = k/n$:

$$\begin{aligned}
L(p \mid n, k) &= \binom{n}{k} p^k (1-p)^{n-k} \\
L'(p|n, k) &= \binom{n}{k} [kp^{k-1}(1-p)^{n-k} - (n-k)(1-p)^{n-k-1}p^k] \\
&= \binom{n}{k} p^{k-1}(1-p)^{n-k-1} [k(1-p) - (n-k)p] = 0 \\
k(1-p) &= (n-k)p \\
\hat{p} &= k/n
\end{aligned}$$

7.1.4 continuous probability distributions

The definition is more complex for continuous variables (because $P(X = x; \theta) = 0$ as there are infinitely many values...). What is commonly done is to use the *density function* $f(x; \theta)$ and considering the probability of obtaining a value $x \in [x_j, x_j + h]$, where x_j is our observed data point, and h is small. Then:

$$L(\theta \mid x_j) = \lim_{h \rightarrow 0^+} \frac{1}{h} \int_{x_j}^{x_j+h} f(x; \theta) dx = f(x_j; \theta)$$

Note that, contrary to probabilities, density values can take values greater than 1. As such, when the dispersion is small, one could end up with values of likelihood greater than 1 (or positive log-likelihoods). In fact, the likelihood function is proportional to but not necessarily equal to the probability of generating the data given the parameters: $L(\theta|X) \propto P(X; \theta)$.

Most classical statistical estimations are based on maximizing a likelihood function. For example, linear regression estimates of slope and intercept are based on minimizing the sum of squares, or more generally, the χ^2 -squared statistic. This amounts to maximizing the likelihood of the underlying model, which is based on the assumptions of normally distributed independent residuals.

7.2 Bayesian thinking

We will formalize the process of incorporation of prior knowledge into probabilistic inference by going back to the notion of conditional probability introduced in week 2. First, if you multiply both sides of the definition by $P(B)$, then we obtain the probability of the intersection of events A and B :

$$P(A \cap B) = P(A \mid B)P(B); \quad P(A \cap B) = P(B \mid A)P(A)$$

Second, we can partition a sample space into two complementary sets, A and \bar{A} , and then the set of B can be partitioned into two parts, that intersect with A and \bar{A} , respectively, so that the probability of B is

$$P(B) = P(A \cap B) + P(\bar{A} \cap B)$$

The two formulas together lead to a very important result called the *law of total probability*:

$$P(B) = P(B \mid A)P(A) + P(B \mid \bar{A})P(\bar{A})$$

It may not be clear at first glance why this is useful: after all, we replaced something simple ($P(B)$) with something much more complex on the right hand side. You will see how this formula enables us to calculate quantities that are not otherwise accessible.

Example: Suppose we know that the probability of a patient having a disease is 1% (called the prevalence of the disease in a population), and the sensitivity and specificity of the test are both 80%. What is the probability of obtaining a negative test result for a randomly selected

patient? Let us call $P(H) = 0.99$ the probability of a healthy patient and $P(D) = 0.01$ the probability of a diseased patient. Then:

$$\begin{aligned} P(\text{Neg}) &= P(\text{Neg} \mid H)P(H) + P(\text{Neg} \mid D)P(D) = \\ &= 0.8 \times 0.99 + 0.2 \times 0.01 = 0.794 \end{aligned}$$

7.2.1 Bayes' formula

Take the first formula in this section, which expresses the probability $P(A \cap B)$ in two different ways. Since the expressions are equal, we can combine them into one equation, and by dividing both sides by $P(B)$, we obtain what's known as *Bayes' formula*:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Another version of Bayes' formula re-writes the denominator using the Law of total probability above:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B \mid A)P(A) + P(B \mid \bar{A})P(\bar{A})}$$

Bayes' formula gives us the probability of A given B from probabilities of B given A and given $\neg A$, and the prior (baseline) probability of $P(A)$. This is enormously useful when it is easy to calculate the conditionals one way and not the other. Among its many applications, it computes the effect of a test result with given sensitivity and specificity (conditional probabilities) on the probability of the hypothesis being true.

7.2.2 positive predictive value

In reality, a doctor doesn't have the true information about the patient's health, but rather the information from the test and hopefully some information about the population where she is working. Let us assume we know the rate of false positives $P(\text{Pos} \mid H)$ and the rate of false negatives $P(\text{Neg} \mid D)$, as well as the prevalence of the disease in the whole population $P(D)$. Then we can use Bayes' formula to answer the practical question, if the test result is positive, what is the probability the patient is actually sick? This is called the *positive predictive value* of a test. The deep Bayesian fact is that one cannot make inferences about the health of the patient after the test without some prior knowledge, specifically the prevalence of the disease in the population:

$$P(D \mid \text{Pos}) = \frac{P(\text{Pos} \mid D)P(D)}{P(\text{Pos} \mid D)P(D) + P(\text{Pos} \mid H)P(H)}$$

Example. Suppose the test has a 0.01 probability of both false positive and false negatives, and the overall prevalence of the disease in the population 0.02. You may be surprised that from an epidemiological perspective, a positive result is far from definitive:

$$P(D | Pos) = \frac{0.99 \times 0.02}{0.99 \times 0.02 + 0.01 \times 0.98} = 0.67$$

This is because the disease is so rare, that even though the test is quite accurate, there are going to be a lot of false positives (about 1/3 of the time) since 98% of the patients are healthy.

We can also calculate the probability of a patient who tests negative of actually being healthy, which is called the *negative predictive value*. In this example, it is far more definitive:

$$\begin{aligned} P(H | Neg) &= \frac{P(Neg | H)P(H)}{P(Neg | H)P(H) + P(Neg | D)P(D)} = \\ &= \frac{0.99 \times 0.98}{0.99 \times 0.98 + 0.01 \times 0.02} = 0.9998 \end{aligned}$$

This is again because this disease is quite rare in this population, so a negative test result is almost guaranteed to be correct. In another population, where disease is more prevalent, this may not be the case.

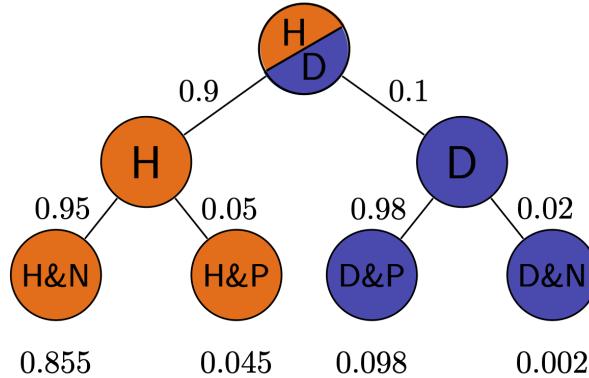


Figure 7.1: Bayesian hypothesis testing tree with prior probability 0.1

Exercise: Simulate medical testing by rolling dice for a rare disease (1/6 prevalence) and a common disease (1/2 prevalence), with both sensitivity and specificity of 5/6. Compare the positive predictive values for the two cases.

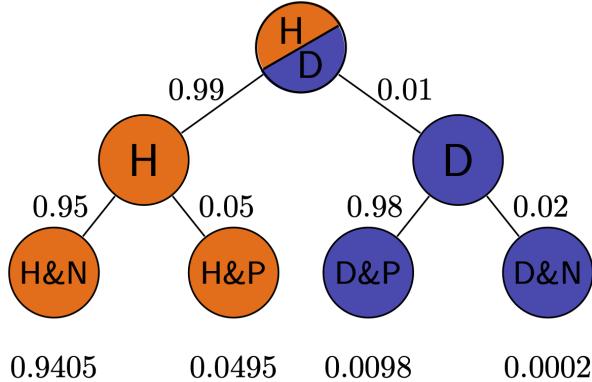


Figure 7.2: Bayesian hypothesis testing tree with prior probability 0.01

7.2.3 prosecutor's fallacy

The basic principle of Bayesian thinking is that one cannot interpret the reliability of a result, e.g. a hypothesis test, without factoring in the prior probability of it being true. This seems like a commonsensical concept, but it is often neglected when such results are interpreted in various contexts, which can lead to perilous mistakes.

Here is a scenario called “the prosecutor’s fallacy”. Suppose that a defendant is accused of a crime, and physical evidence collected at the crime scene matches this person (e.g. a fingerprint or a DNA sample), but no other evidence exists to connect the defendant to the crime. The prosecutor calls an expert witness to testify that fewer than one out of a million randomly chosen people would match this sample. Therefore, she argues, there is overwhelming probability that the defendant is guilty and less than 1 in a million chance they are innocent.

Do you spot the problem with the argument?

It’s the same fallacy as we saw in the medical testing scenario, or that is portrayed in the xkcd cartoon above. The prosecutor is conflating the probability of a match (positive result) given that the person is innocent, and the probability of the person being innocent, given the match. The probability of the former is one in a million, but we want to know the latter! And the latter depends on the prior probability of the person committing the crime, which should have been investigated by the detectives: did the defendant have a conflict with the victim or have they never met? did he have opportunity to commit the crime, or was he in a different city at the time? Without this information, it is impossible to decide whether it’s more likely that the DNA/fingerprint match is a false positive (in a country of 300 million, you can find 300 false matches if everyone is in the database!) or a true positive.

7.2.4 reproducibility in science

In 2005 John Ioannidis published a paper entitled “[Why most published research findings are false](#)”. The paper, as you can see by its title, was intended to be provocative, but it is based solidly on the classic formula of Bayes. The motivation for the paper came from the observation that too often in modern science, big, splashy studies that were published could not be reproduced or verified by other researchers. What could be behind this epidemic of questionable scientific work?

The problem as described by Ioannidis and many others, in a nutshell, is that unthinking use of traditional hypothesis testing leads to a high probability of false positive results being published. The paper outlines several ways in which this can occur.

Too often, a hypothesis is tested and if the resultant p-value is less than some arbitrary threshold (very often 0.05, an absurdly high number), then the results are published. However, if one is testing a hypothesis with low prior probability, a positive hypothesis test result is very likely a false positive. Very often, modern biomedical research involves digging through a large amount of information, like an entire human genome, in search for associations between different genes and a phenotype, like a disease. It is *a priori* unlikely that any specific gene is linked to a given phenotype, because most genes have very specific functions, and are expressed quite selectively, only at specific times or in specific types of cells. However, publishing such studies results in splashy headlines (“Scientists find a gene linked to autism!”) and so a lot of false positive results are reported, only to be refuted later, in much less publicized studies.

Ioannidis performed basic calculations of the probability that a published study is true (that is, that a positive reported result is a true positive), and how it is affected by pre-study (prior) probability, number of conducted studies on the same hypothesis, and the level of bias. His prediction is that for fairly typical scenario (e.g. pre-study probability of 10%, ten groups working simultaneously, and a reasonable amount of bias) the probability that a published result is correct is less than 50%. He then followed up with another paper [2] that investigated 49 top-cited medical research publications over a decade, and looked at whether follow-up studies could replicate the results, and found that a very significant fraction of their findings could not be replicated or were found to have weaker effects by subsequent investigations.

7.3 Bayesian inference

As an alternative to frequentist and maximum likelihood approaches to modeling biological data, Bayesian statistics has seen an impressive growth in recent years, due to the improved computational power.

At the heart of Bayesian inference is an application of Bayes’ theorem: take a model with parameters θ , and some data D . Bayes’ theorem gives us a disciplined way to “update” our belief in the distribution of θ once we’ve seen the data D :

$$P(\theta | D) = \frac{P(D | \theta)P(\theta)}{P(D)}$$

where:

- $P(\theta | D)$ is the **posterior distribution** of θ , i.e., our updated belief in the values of θ .
- $P(D | \theta)$ is the **likelihood function**: $P(DX | \theta) = L(\theta | D)$.
- $P(\theta)$ is the **prior distribution**, i.e. our belief on the distribution of θ before seeing the data.
- $P(D)$ is called the **evidence**: $P(D) = \int P(D | \theta)d\theta$ (in practice, this need not to be calculated).

7.3.1 Example: capture-recapture

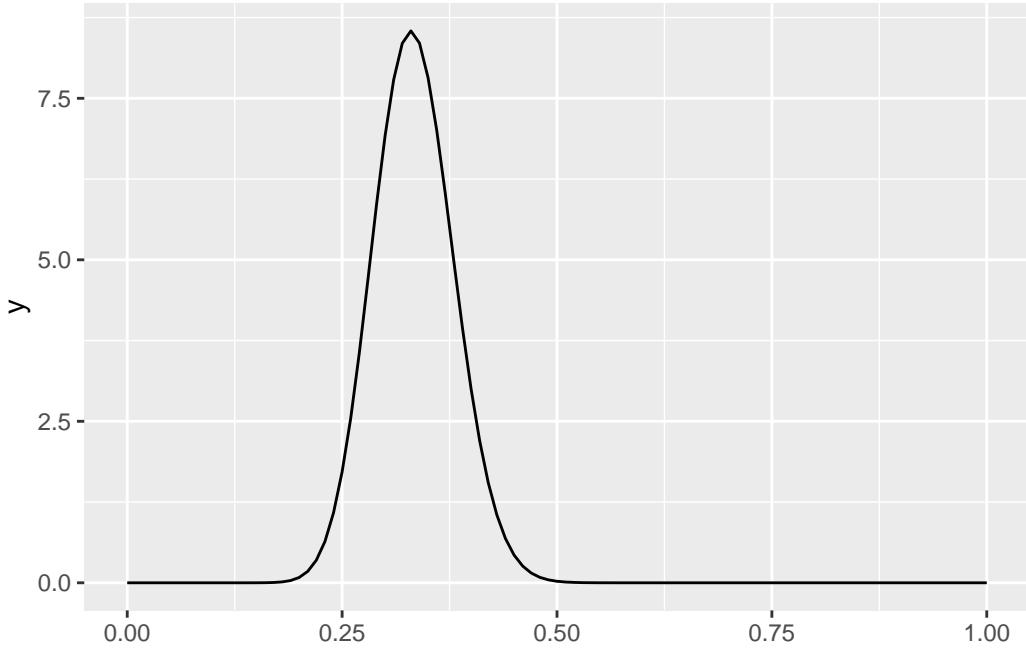
There is a well-established method in population ecology of estimating the size of a population by repeatedly capturing and tagging a number of individuals and later repeating the experiment to see how many are recaptured. Suppose that n were captured initially and k were recaptured later. We assume that the probability p of recapturing an individual is the same for all individuals. Then our likelihood function is once again, based on the binomial distribution.

$$L(p | k, n) = \binom{n}{k} p^k (1-p)^{n-k}$$

and our maximum likelihood estimate is $\hat{p} = k/n$. This allows for estimation of the total population size to be $P = n_2/\hat{p}$, where n_2 is the total number of individuals captured in the second experiment. There are more sophisticated estimators, but this one is reasonable for large enough populations.

Let us plot the likelihood as a function of p for the case in which $n = 100$ and $k = 33$

```
library(tidyverse)
n <- 100
m <- 33
pl <- ggplot(data = data.frame(x = 0, y = 0)) + xlim(c(0,1))
likelihood_function <- function(p) {
  lik <- choose(n, m) * p^m * (1-p)^(n - m)
  # divide by the evidence to make into density function
  return(lik * (n + 1))
}
pl <- pl + stat_function(fun = likelihood_function)
show(pl)
```



Now we choose a prior. For convenience, we choose a Beta distribution, $P(p) = \text{Beta}(\alpha, \beta) = \frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(\alpha, \beta)}$, where $B(\alpha, \beta)$ is the Beta function, $B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt$.

Therefore:

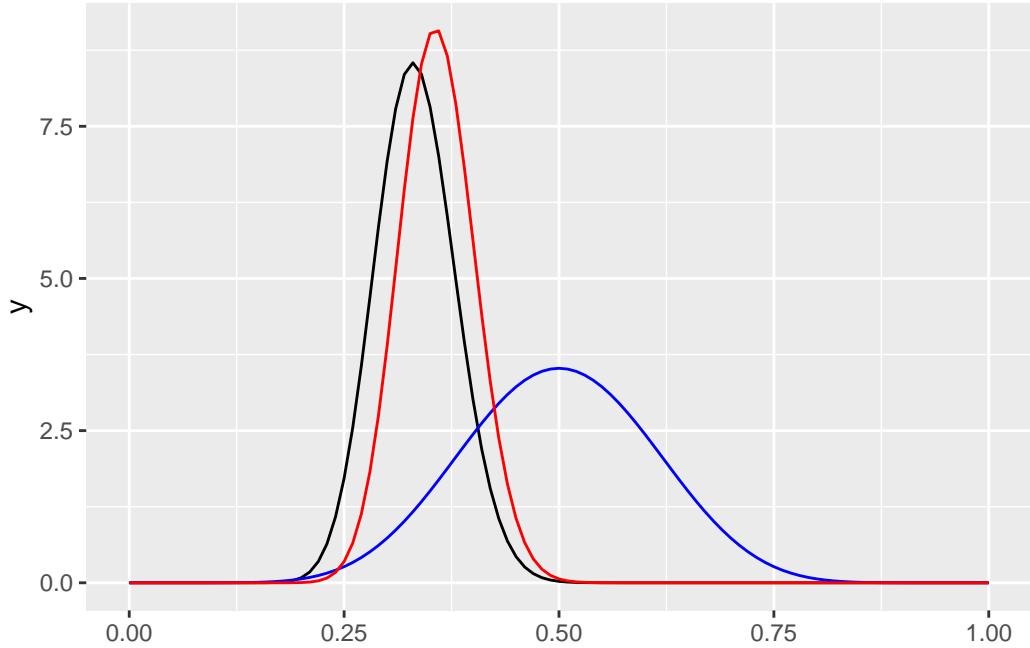
$$\begin{aligned}
 P(p \mid m, n) &\propto L(p \mid m, n)P(p) \\
 &= \left(\binom{n}{m} p^m (1-p)^{n-m} \right) \left(\frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(\alpha, \beta)} \right) \\
 &\propto p^{m+\alpha-1} (1-p)^{n-m+\beta-1} \\
 &\propto \text{Beta}(m + \alpha, \beta + m - n)
 \end{aligned}$$

We can explore the effect of choosing a prior on the posterior. Suppose that in the past we have seen probabilities close to 50%. Then we could choose a prior $\text{Beta}(10, 10)$ (this is what is called a “strong” or “informative” prior). Let’s see what happens to the posterior:

```

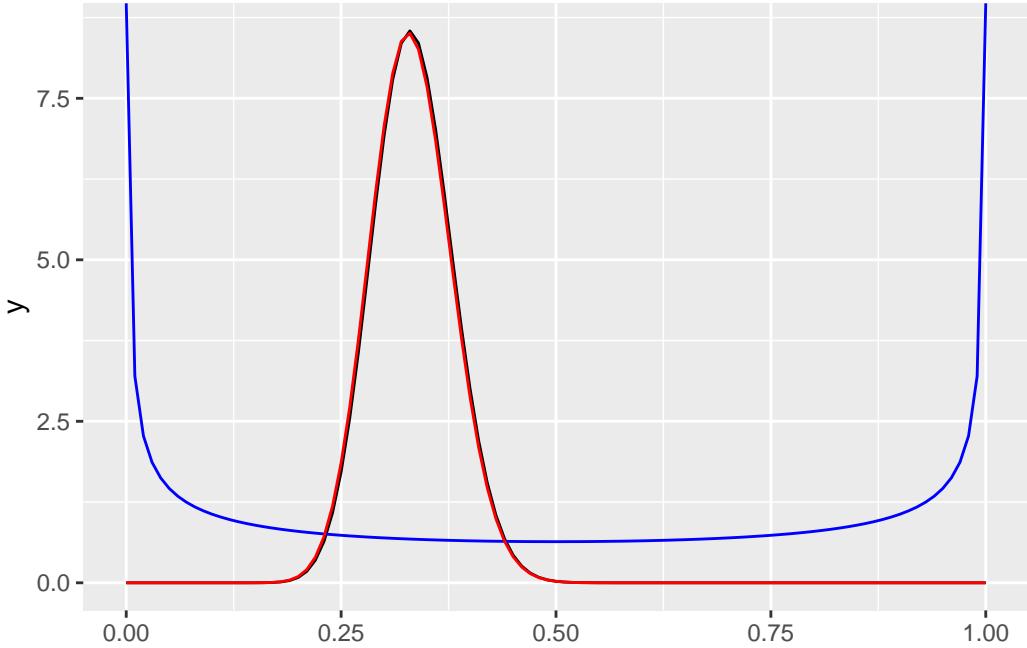
# a strong prior
alpha <- 10
beta <- 10
prior_function <- function(p) dbeta(p, alpha, beta)
posterior_function <- function(p) dbeta(p, alpha + m, beta + n - m)
pl + stat_function(fun = prior_function, colour = "blue") +
  stat_function(fun = posterior_function, colour = "red")

```



You can see that the posterior “mediates” between the prior and the likelihood curve. When we use a weak prior, then our posterior will be closer to the likelihood function:

```
# a weak prior
alpha <- 1/2
beta <- 1/2
pl + stat_function(fun = prior_function, colour = "blue") +
  stat_function(fun = posterior_function, colour = "red")
```



The fact that the posterior depends on the prior is the most controversial aspect of Bayesian inference. Different schools of thought treat this feature differently (e.g., “Subjective Bayes” interprets priors as beliefs before seeing the data; “Empirical Bayes” relies on previous experiments or on the data themselves to derive the prior; “Objective Bayes” tries to derive the least-informative prior given the data). In practice, the larger the data, the cleaner the signal, the lesser the influence of the prior on the resulting posterior.

7.3.2 MCMC

The type of calculation performed above is feasible only for very simple models, and for appropriately chosen priors (called “conjugate priors”). For more complex models, we rely on simulations. In particular, one can use Markov-Chain Monte Carlo (MCMC) to sample from the posterior distribution of complex models. Very briefly, one builds a Markov-Chain in which the states represent sets of parameters; parameters are sampled from the prior, and the probability of moving to one state to another is proportional to the difference in their likelihood. When the MC converges, then one obtains the posterior distribution of the parameters.

Bayesian inference is used for many complex problems, including phylogenetic tree building [5].

7.4 Reading:

1. Maximum likelihood estimation from scratch
2. Phylogenetic Analysis by Maximum Likelihood
3. Why most published scientific studies are false
4. Contradicted and Initially Stronger Effects in Highly Cited Clinical Research
5. MrBayes
6. Quick and Dirty Tree Building in R
7. Mark and Recapture

8 Review of linear algebra

Goals

- Solving linear equations
- Best-fit line through multiple data points
- Concepts of linearity and vector spaces
- Representation of vectors in multiple bases
- Eigenvalues and eigenvectors of matrices

```
library(tidyverse) # our friend the tidyverse
library(ggfortify)
```

8.1 Solving multivariate linear equations

Linear algebra is intimately tied to linear equations, that is, to equations where all variables are multiplied by constant terms and added together. Linear equations with one variable are easily solved by division, for example:

$$4x = 20$$

is solved by dividing both sides by 4, obtaining the unique solution $x = 5$.

The situation gets more interesting when multiple variables are involved, with multiple equations, for example:

$$\begin{aligned} 4x - 3y &= 5 \\ -x + 2y &= 10 \end{aligned}$$

There are multiple ways to solve this, for example solving one equation for one variable in terms of the other, then substituting it into the second equation to obtain a one-variable problem. A more general approach involves writing this problem in terms of a matrix \mathbf{A} that contains the multiplicative constants of x and y and a vector \vec{b} that contains the right-hand side constants:

$$\mathbf{A} = \begin{pmatrix} 4 & -3 \\ -1 & 2 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 5 \\ 10 \end{pmatrix} \quad \vec{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\mathbf{A}\vec{x} = \vec{b}$$

This now looks analogous to the one-variable equation above, which we solved by dividing both sides by the multiple of x . The difficulty is that matrix operations are more complicated than scalar multiplication and division. Matrix multiplication is used in the equation above to multiply all the coefficients in the matrix by their respective variables, which involves a relatively complicated [procedure in general](#).

The “division” equivalent is called [matrix inversion](#) and it is even more complicated. First, we need to define the identity matrix, or the equivalent of the number 1 for matrix multiplication. The identity matrix is defined only for square matrices (equal number of rows and columns), so a size n by n identity matrix is defined to have all 1s on the diagonal and all zeros on the off-diagonal:

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

The identity matrix is special because multiplying any other matrix (of compatible size) by it results in the same exact matrix (this is easy to check on a couple of examples for 2×2 or 3×3 matrices):

$$IA = AI = A$$

Then for an n by n matrix A its inverse A^{-1} is defined to be the matrix multiplication by which results in the identity matrix, that is:

$$A^{-1}A = AA^{-1} = I$$

Defining the inverse is one task, but calculating it for any given matrix, especially of large size, is quite laborious. We will not describe the algorithms here, but you can read about [Gauss-Jordan elimination](#), which is one classic example. One important point is that not all matrices are invertible, and for some no inverse matrix exists, analogous to zero for real number division. The difference is that there are infinitely many matrices for which this is the case, called *singular* matrices.

In the cases in which the inverse matrix exists, the linear system of equations can be solved by multiplying both sides by the inverse matrix, like this:

$$\vec{x} = \mathbf{A}^{-1}\vec{b}$$

Example: Take the linear 2×2 system of equations of above and solve it using matrix inversion. The R function `solve()` calculates the inverse and multiplies it by the constant vector `b`:

```
A <- matrix(c(4,-1,-3,2), nrow = 2)
b <- c(5,10)
solve(A,b)
```

```
[1] 8 9
```

8.2 Fitting a line to data

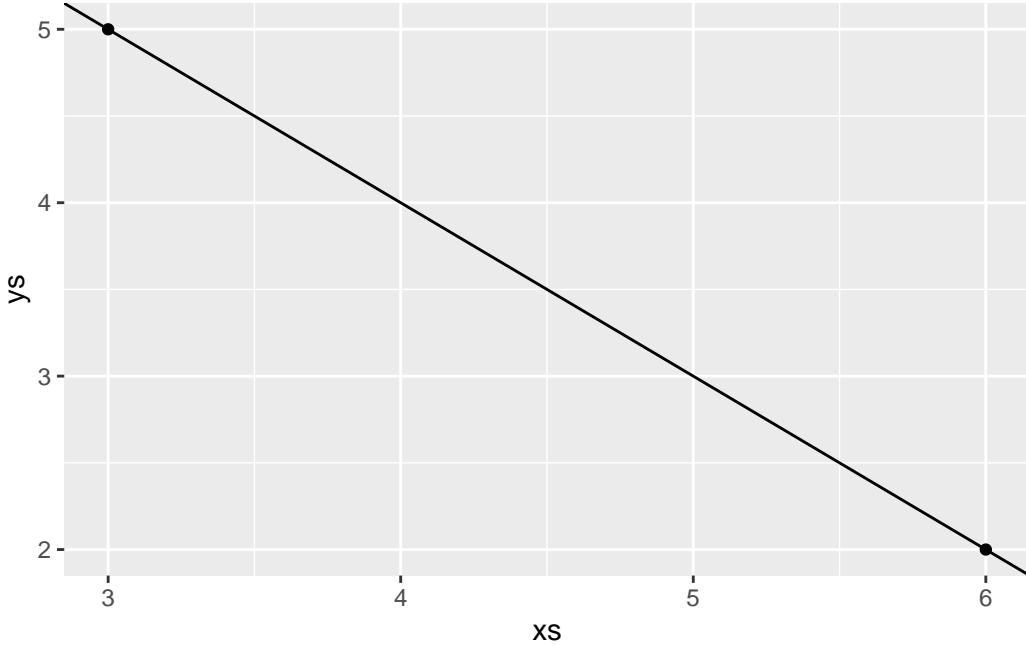
One geometric application of solving multiple linear equations is to find the coefficients of a line that passes through two points in the 2-dimensional plane (or of a plane that passes through three points in three-dimensional space, but we won't go there.) In that case, the coordinates of the points are the data, and the unknown variables are the parameters slope m and intercept b of the line that we want to find.

Example: If the data set consists of two points $(3, 5), (6, 2)$, then finding the best fit values of m and b means solving the following two equations:

$$\begin{aligned} 3m + b &= 5 \\ 6m + b &= 2 \end{aligned}$$

These equations have a solution for the slope and intercept, which can be calculated in R using `solve()` and then plot the line with the parameters from the solution vector `beta`:

```
xs <- c(3, 6)
ys <- c(5, 2)
A <- matrix(c(xs[1], xs[2], 1, 1), nrow = 2)
b <- c(ys[1], ys[2])
beta <- solve(A, b)
data1 <- tibble(xs, ys)
ggplot(data = data1) +
  aes(x = xs, y = ys) +
  geom_point() +
  geom_abline(slope = beta[1], intercept = beta[2])
```



However, a data set with two points is very small and nobody would accept these values as reasonable estimates. Let us add one more data point, to increase our sample size to three: $(3, 5), (6, 2), (9, 1)$. How do you find the best fit slope and intercept?

Bad idea: take two points and find a line, that is the slope and the intercept, that passes through the two. It should be clear why this is a bad idea: we are arbitrarily ignoring some of the data, while perfectly fitting two points. So how do we use all the data? Let us write down the equations that a line with slope m and intercept b have to satisfy in order to fit our data points:

$$3m + b = 5$$

$$6m + b = 2$$

$$9m + b = 1$$

This system has no exact solution, since there are three equations and only two unknowns. We need to find m and b such that they are a “best fit” to the data, not the perfect solution.

8.2.1 Least-squares line

Let us write the equation in matrix form as follows:

$$\mathbf{A} = \begin{pmatrix} 3 & 1 \\ 6 & 1 \\ 9 & 1 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix} \quad \vec{\beta} = \begin{pmatrix} m \\ b \end{pmatrix}$$

$$\mathbf{A}\vec{\beta} = \vec{b}$$

Mathematically, the problem is that one cannot invert a non-square matrix. However, there is a way of turning the matrix into a square one, by multiplying it by its own transpose (same matrix with rows and columns reversed):

$$\mathbf{A}^T \mathbf{A} \vec{\beta} = \mathbf{A}^T \vec{b}$$

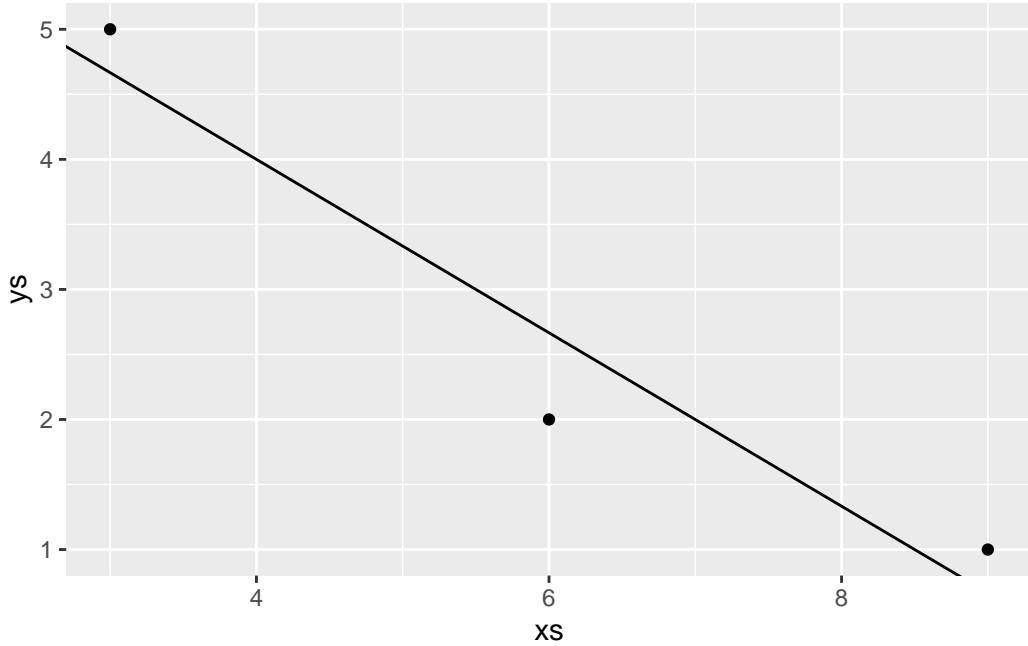
Exercise: Carry out the matrix multiplications to verify that $\mathbf{A}^T \mathbf{A}$ is a 2×2 matrix and $\mathbf{A}^T \vec{b}$ is a vector of length 2.

Now we can solve this equation with a square matrix $\mathbf{A}^T \mathbf{A}$ by multiplying both sides by the inverse! In general, for an n -dimensional data set consisting of a bunch of values of x and y , the process looks like this:

$$\vec{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} = \begin{pmatrix} m \\ b \end{pmatrix} \beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{Y}$$

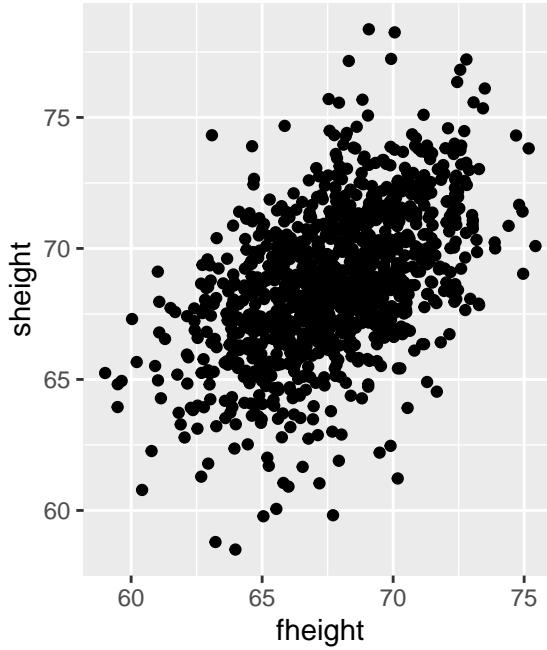
Example: Let us see the best-fit line for the 3-point data set above:

```
xs <- c(3, 6, 9)
ys <- c(5, 2, 1)
A <- matrix(c(xs[1], xs[2], xs[3], 1, 1, 1), nrow = 3)
b <- c(ys[1], ys[2], ys[3])
beta <- solve(t(A) %*% A, t(A) %*% b)
data1 <- tibble(xs, ys)
ggplot(data = data1) +
  aes(x = xs, y = ys) +
  geom_point() +
  geom_abline(slope = beta[1], intercept = beta[2])
```



Let us use the classic data set of Karl Pearson's from 1903 containing the height of fathers and sons, which we will return to next week when we tackle linear regression properly:

```
require(UsingR)
data("father.son")
pl <- ggplot(data = father.son) +
  aes(x = fheight, y = sheight) +
  geom_point() +
  coord_equal()
pl
```



Exercise: Let's try to find the best fit line to this data set (the hard way) using the same process as above for the three - point data set:

Of course, R can do this calculation for you with just one command:

```
best_beta_easy <- lm(sheight ~ fheight, data = father.son)
best_beta_easy
```

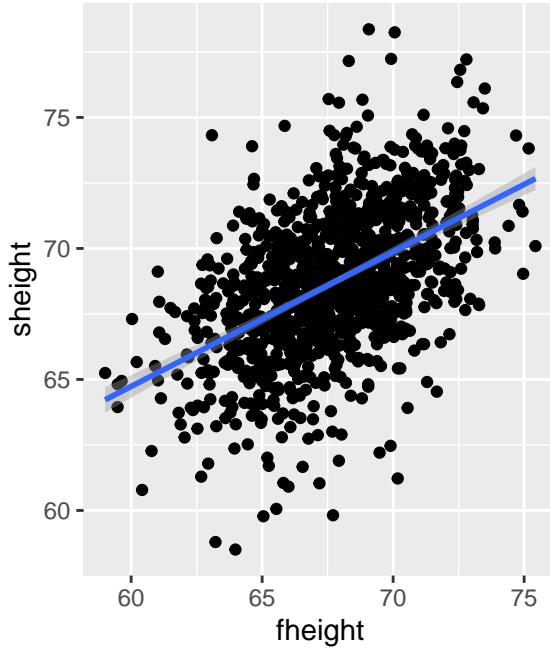
```
Call:
lm(formula = sheight ~ fheight, data = father.son)

Coefficients:
(Intercept)      fheight
            33.8866        0.5141
```

But it feels good to know that this is not black magic! In fact, plotting it on top of the data does not even require computing the coefficients:

```
pl + geom_smooth(method = "lm") # lm stands for linear model

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



8.3 Linearity and vector spaces

We have dealt with linear models in various guises, so now would be a good time to define properly what linearity means. The word comes from the shape of graphs of linear functions of one variable, e.g. $f(x) = ax + b$, but the algebraic meaning rests on the following two general properties:

Definition. A *linear transformation* or *linear operator* is a mapping L between two sets of vectors with the following properties:

1. (*scalar multiplication*) $L(c\vec{v}) = cL(\vec{v})$; where c is a scalar and \vec{v} is a vector
2. (*additive*) $L(\vec{v}_1 + \vec{v}_2) = L(\vec{v}_1) + L(\vec{v}_2)$; where \vec{v}_1 and \vec{v}_2 are vectors

Here we have two types of objects: vectors and transformations/operators that act on those vectors. The basic example of this are vectors and matrices, because a matrix multiplied by a vector (on the right) results another vector, provided the number of columns in the matrix is the same as the number of rows in the vector. This can be interpreted as the matrix transforming the vector \vec{v} into another one: $A \times \vec{v} = \vec{u}$.

Example: Let us multiply the following matrix and vector (specially chosen to make a point):

```

A <- matrix(c(2, 2, 1, 3), nrow = 2)
vec1 <- c(1, -1)
vec2 <- A %*% vec1
print(vec1)

```

[1] 1 -1

```
print(vec2)
```

```

[,1]
[1,]    1
[2,]   -1

```

We see that this particular vector $(1, -1)$ is unchanged when multiplied by this matrix, or we can say that the matrix multiplication is equivalent to multiplication by 1. Here is another such vector for the same matrix:

```

vec1 <- c(1, 2)
vec2 <- A %*% vec1
print(vec1)

```

[1] 1 2

```
print(vec2)
```

```

[,1]
[1,]    4
[2,]    8

```

In this case, the vector is changed, but only by multiplication by a constant (4). Thus the geometric direction of the vector remained unchanged.

The notion of linearity leads to the important idea of combining different vectors:

Definition: A *linear combination* of n vectors $\{\vec{v}_i\}$ is a weighted sum of these vectors with any real numbers $\{a_i\}$:

$$a_1\vec{v}_1 + a_2\vec{v}_2 \dots + a_n\vec{v}_n$$

Linear combinations arise naturally from the notion of linearity, combining the additive property and the scalar multiplication property. Speaking intuitively, a linear combination of vectors produces a new vector that is related to the original set. Linear combinations give a simple way of generating new vectors, and thus invite the following definition for a collection of vectors closed under linear combinations:

Definition. A *vector space* is a collection of vectors such that a linear combination of any n vectors is contained in the vector space.

The most common examples are the spaces of all real-valued vectors of dimension n , which are denoted by \mathbb{R}^n . For instance, \mathbb{R}^2 (pronounced “r two”) is the vector space of two dimensional real-valued vectors such as $(1, 3)$ and $(\pi, -\sqrt{17})$; similarly, \mathbb{R}^3 is the vector space consisting of three dimensional real-valued vectors such as $(0.1, 0, -5.6)$. You can convince yourself, by taking linear combinations of vectors, that these vector spaces contain all the points in the usual Euclidean plane and three-dimensional space. The real number line can also be thought of as the vector space \mathbb{R}^1 .

8.3.1 Linear independence and basis vectors

How can we describe a vector space without trying to list all of its elements? We know that one can generate an element by taking linear combinations of vectors. It turns out that it is possible to generate (or “span”) a vector space by taking linear combinations of a subset of its vectors. The challenge is to find a minimal subset of subset that is not redundant. In order to do this, we first introduce a new concept:

Definition: A set of vectors $\{\vec{v}_i\}$ is called *linearly independent* if the only linear combination involving them that equals the zero vector is if all the coefficients are zero. ($a_1\vec{v}_1 + a_2\vec{v}_2 + \dots + a_n\vec{v}_n = 0$ only if $a_i = 0$ for all i .)

In the familiar Euclidean spaces, e.g. \mathbb{R}^2 , linear independence has a geometric meaning: two vectors are linearly independent if the segments from the origin to the endpoint do not lie on the same line. But it can be shown that any set of three vectors in the plane is linearly dependent, because there are only two dimensions in the vector space. This brings us to the key definition of this section:

Definition: A *basis* of a vector space is a linearly independent set of vectors that generate (or span) the vector space. The number of vectors (cardinality) in such a set is called the *dimension* of the vector space.

A vector space generally has many possible bases, as illustrated in figure. In the case of \mathbb{R}^2 , the usual (canonical) basis set is $\{(1, 0); (0, 1)\}$ which obviously generates any point on the plane and is linearly independent. But any two linearly independent vectors can generate any vector in the plane.

Example: The vector $\vec{r} = (2, 1)$ can be represented as a linear combination of the two canonical vectors: $\vec{r} = 2 \times (1, 0) + 1 \times (0, 1)$. Let us choose another basis set, say $\{(1, 1); (-1, 1)\}$ (this is the canonical basis vectors rotated by $\pi/2$.) The same vector can be represented by a linear combination of these two vectors, with coefficients 1.5 and -0.5 : $\vec{r} = 1.5 \times (1, 1) - 0.5 \times (-1, 1)$. If we call the first basis C for canonical and the second basis D for different, we can write the same vector using different sets of coordinates for each basis:

$$\vec{r}_C = (2, 1); \vec{r}_D = (1.5, -0.5)$$

8.3.2 Projections and changes of basis

The representation of an arbitrary vector (point) in a vector space as a linear combination of a given basis set is called the *decomposition* of the point in terms of the basis, which gives the coordinates for the vector in terms of each basis vector. The decomposition of a point in terms of a particular basis is very useful in high-dimensional spaces, where a clever choice of a basis can allow a description of a set of points (such as a data set) in terms of contributions of only a few basis vectors, if the data set primarily extends only in a few dimensions.

To obtain the coefficients of the basis vectors in a decomposition of a vector \vec{r} , we need to perform what is termed a *projection* of the vector onto the basis vectors. Think of shining a light perpendicular to the basis vector, and measuring the length of the shadow cast by the vector \vec{r} onto \vec{v}_i . If the vectors are parallel, the shadow is equal to the length of \vec{r} ; if they are orthogonal, the shadow is nonexistent. To find the length of the shadow, use the inner product of \vec{r} and \vec{v} , which as you recall corresponds to the cosine of the angle between the two vectors multiplied by their norms: $\langle \vec{r}, \vec{v} \rangle = |\vec{r}| |\vec{v}| \cos(\theta)$. We do not care about the length of the vector \vec{v} we are projecting onto, thus we divide the inner product by the square norm of \vec{v} , and then multiply the vector \vec{v} by this projection coefficient:

$$Proj(\vec{r}; \vec{v}) = \frac{\langle \vec{r}, \vec{v} \rangle}{\langle \vec{v}, \vec{v} \rangle} \vec{v} = \frac{\langle \vec{r}, \vec{v} \rangle}{|\vec{v}|^2} \vec{v} = \frac{|\vec{r}| \cos(\theta)}{|\vec{v}|} \vec{v}$$

This formula gives the projection of the vector \vec{r} onto \vec{v} , the result is a new vector in the direction of \vec{v} , with the scalar coefficient $a = \langle \vec{r}, \vec{v} \rangle / |\vec{v}|^2$.

Example: Here is how one might calculate the projection of the point $(2, 1)$ onto the basis set $\{(1, 1); (-1, 1)\}$:

```
v1 <- c(1, 1)
v2 <- c(-1, 1)
u <- c(2, 1)
ProjMat <- matrix(cbind(v1, v2),
                    byrow = T, nrow = 2)
```

```
print(ProjMat)
```

```
[,1] [,2]
[1,]    1    1
[2,]   -1    1
```

```
ProjMat %*% u
```

```
[,1]
[1,]    3
[2,]   -1
```

This is not quite right: the projection coefficients are off by a factor of two compared to the correct values in the example above. This is because we have neglected to *normalize* the basis vectors, so we should modify the script as follows:

```
v1 <- c(1, 1)
v1 <- v1 / (sum(v1^2))
v2 <- c(-1, 1)
v2 <- v2 / (sum(v2^2))
u <- c(2, 1)
ProjMat <- matrix(cbind(v1, v2),
                    byrow = T, nrow = 2)
print(ProjMat)
```

```
[,1] [,2]
[1,]  0.5  0.5
[2,] -0.5  0.5
```

```
print(ProjMat %*% u)
```

```
[,1]
[1,]  1.5
[2,] -0.5
```

This is an example of how to convert a vector/point from representation in one basis set to another. The new basis vectors, expressed in the original basis set, are arranged in a matrix by row, scaled by their norm squared, and multiplied by the vector that one wants to express in the new basis. The resulting vector contains the coordinates in the new basis.

8.4 Matrices as linear operators

8.4.1 Matrices transform vectors

In this section we will learn to characterize square matrices by finding special numbers and vectors associated with them. At the core of this analysis lies the concept of a matrix as an *operator* that transforms vectors by multiplication. To be clear, in this section we take as default that the matrices A are square, and that vectors \vec{v} are column vectors, and thus will multiply the matrix on the right: $A \times \vec{v}$.

A matrix multiplied by a vector produces another vector, provided the number of columns in the matrix is the same as the number of rows in the vector. This can be interpreted as the matrix transforming the vector \vec{v} into another one: $A \times \vec{v} = \vec{u}$. The resultant vector \vec{u} may or may not resemble \vec{v} , but there are special vectors for which the transformation is very simple.

Example. Let us multiply the following matrix and vector (specially chosen to make a point):

$$\begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 2-1 \\ 2-3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

We see that this particular vector is unchanged when multiplied by this matrix, or we can say that the matrix multiplication is equivalent to multiplication by 1. Here is another such vector for the same matrix:

$$\begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2+2 \\ 2+6 \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix}$$

In this case, the vector is changed, but only by multiplication by a constant (4). Thus the geometric direction of the vector remained unchanged.

Generally, a square matrix has an associated set of vectors for which multiplication by the matrix is equivalent to multiplication by a constant. This can be written down as a definition:

Definition. An *eigenvector* of a square matrix A is a vector \vec{v} for which matrix multiplication by A is equivalent to multiplication by a constant. This constant λ is called its *eigenvalue* of A corresponding to the eigenvector \vec{v} . The relationship is summarized in the following equation:

$$A \times \vec{v} = \lambda \vec{v}$$

Note that this equation combines a matrix (A), a vector (\vec{v}) and a scalar λ , and that both sides of the equation are column vectors.

The definition does not specify how many such eigenvectors and eigenvalues can exist for a given matrix A . There are usually as many such vectors \vec{v} and corresponding numbers λ as the number of rows or columns of the square matrix A , so a 2 by 2 matrix has two eigenvectors and two eigenvalues, a 5x5 matrix has 5 of each, etc. One ironclad rule is that there cannot be more distinct eigenvalues than the matrix dimension. Some matrices possess fewer eigenvalues than the matrix dimension, those are said to have a *degenerate* set of eigenvalues, and at least two of the eigenvectors share the same eigenvalue.

The situation with eigenvectors is trickier. There are some matrices for which any vector is an eigenvector, and others which have a limited set of eigenvectors. What is difficult about counting eigenvectors is that an eigenvector is still an eigenvector when multiplied by a constant. You can show that for any matrix, multiplication by a constant is commutative: $cA = Ac$, where A is a matrix and c is a constant. This leads us to the important result that if \vec{v} is an eigenvector with eigenvalue λ , then any scalar multiple $c\vec{v}$ is also an eigenvector with the same eigenvalue. The following demonstrates this algebraically:

$$A \times (c\vec{v}) = cA \times \vec{v} = c\lambda\vec{v} = \lambda(c\vec{v})$$

This shows that when the vector $c\vec{v}$ is multiplied by the matrix A , it results in its being multiplied by the same number λ , so by definition it is an eigenvector. Therefore, an eigenvector \vec{v} is not unique, as any constant multiple $c\vec{v}$ is also an eigenvector. It is more useful to think not of a single eigenvector \vec{v} , but of a **collection of vectors that can be inter-converted by scalar multiplication** that are all essentially the same eigenvector. Another way to represent this, if the eigenvector is real, is that an eigenvector as a **direction that remains unchanged by multiplication by the matrix**, such as direction of the vector v in the figure below. As mentioned above, this is true only for real eigenvalues and eigenvectors, since complex eigenvectors cannot be used to define a direction in a real space.

To summarize, eigenvalues and eigenvectors of a matrix are a set of numbers and a set of vectors (up to scalar multiple) that describe the action of the matrix as a multiplicative operator on vectors. “Well-behaved” square $n \times n$ matrices have n distinct eigenvalues and n eigenvectors pointing in distinct directions. In a deep sense, the collection of eigenvectors and eigenvalues defines a matrix A , which is why an older name for them is characteristic vectors and values.

8.4.2 calculating eigenvalues

Finding the eigenvalues and eigenvectors analytically, that is on paper, is quite laborious even for 3 by 3 or 4 by 4 matrices and for larger ones there is no analytical solution. In practice, the task is outsourced to a computer, and MATLAB has a number of functions for this purpose. Nevertheless, it is useful to go through the process in 2 dimensions in order to gain an understanding of what is involved. From the definition of eigenvalues and eigenvectors, the condition can be written in terms of the four elements of a 2 by 2 matrix:

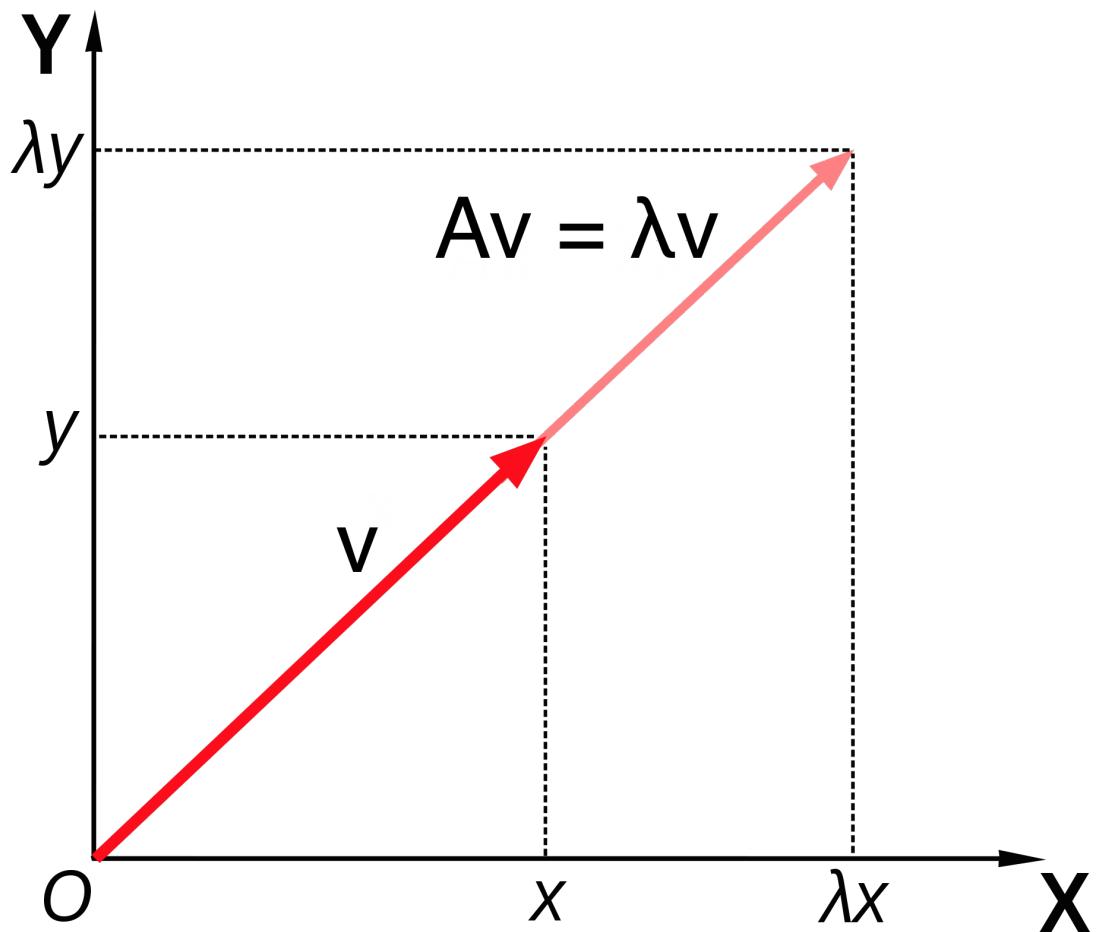


Figure 8.1: Illustration of the geometry of a matrix A multiplying its eigenvector v , resulting in a vector in the same direction λv

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} av_1 + bv_2 \\ cv_1 + dv_2 \end{pmatrix} = \lambda \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

This is now a system of two linear algebraic equations, which we can solve by substitution. First, let us solve for v_1 in the first row, to get

$$v_1 = \frac{-bv_2}{a - \lambda}$$

Then we substitute this into the second equation and get:

$$\frac{-bcv_2}{a - \lambda} + (d - \lambda)v_2 = 0$$

Since v_2 multiplies both terms, and is not necessarily zero, we require that its multiplicative factor be zero. Doing a little algebra, we obtain the following, known as the *characteristic equation* of the matrix:

$$-bc + (a - \lambda)(d - \lambda) = \lambda^2 - (a + d)\lambda + ad - bc = 0$$

This equation can be simplified by using two quantities we defined at the beginning of the section: the sum of the diagonal elements called the trace $\tau = a + d$, and the determinant $\Delta = ad - bc$. The quadratic equation has two solutions, dependent solely on τ and Δ :

$$\lambda = \frac{\tau \pm \sqrt{\tau^2 - 4\Delta}}{2}$$

This is the general expression for a 2 by 2 matrix, showing there are two possible eigenvalues. Note that if $\tau^2 - 4\Delta > 0$, the eigenvalues are real, if $\tau^2 - 4\Delta < 0$, they are complex (have real and imaginary parts), and if $\tau^2 - 4\Delta = 0$, there is only one eigenvalue. This situation is known as degenerate, because two eigenvectors share the same eigenvalue.

Example. Let us take the same matrix we looked at in the previous subsection:

$$A = \begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix}$$

The trace of this matrix is $\tau = 2 + 3 = 5$ and the determinant is $\Delta = 6 - 2 = 4$. Then by our formula, the eigenvalues are:

$$\lambda = \frac{5 \pm \sqrt{5^2 - 4 \times 4}}{2} = \frac{5 \pm 3}{2} = 4, 1$$

These are the multiples we found in the example above, as expected. Of course R has functions to calculate this instead of doing this by hand:

```
A <- matrix(c(2, 2, 1, 3), nrow =2)
eigs <- eigen(A)
eigs$values
```

```
[1] 4 1
```

```
eigs$vectors
```

```
 [,1]      [,2]
[1,] -0.4472136 -0.7071068
[2,] -0.8944272  0.7071068
```

Note: a real-valued matrix can have complex eigenvalues and eigenvectors, but whenever it acts on a real vector, the result is still real. This is because the complex numbers cancel each others imaginary parts.

9 Linear models

Learn how to perform linear regression, how to make sure that the assumptions of the model are not violated, and how to interpret the results.

Note that we need a new library:

```
library(tidyverse)
library(lindia) # regression diagnostic in ggplot2
```

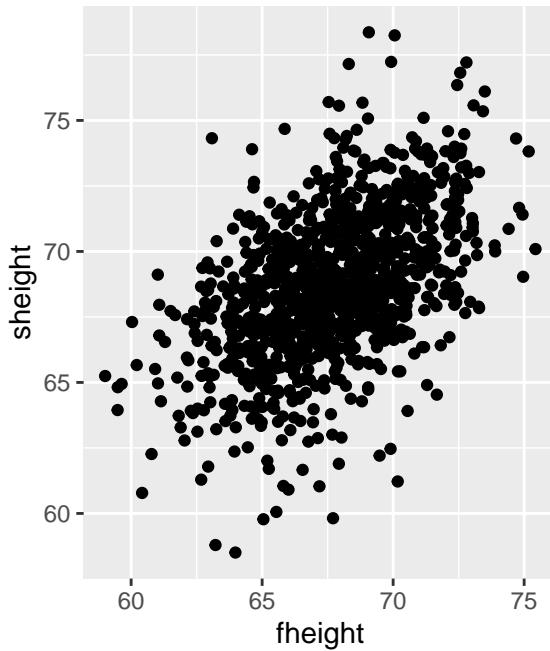
9.1 Regression toward the mean

Francis Galton (Darwin's half-cousin) was a biologist interested in evolution, and one of the main proponents of eugenics (he coined the term himself). To advance his research program, he set out to measure several features in human populations, and started trying to explain the variation he observed, incidentally becoming one of the founding fathers of modern statistics.

In his “Regression towards mediocrity in hereditary stature” he showed an interesting pattern: children of tall parents tended to be shorter than their parents, while children of short parents tended to be taller than their parents. He called this phenomenon “regression toward mediocrity” (now called regression toward [to] the mean).

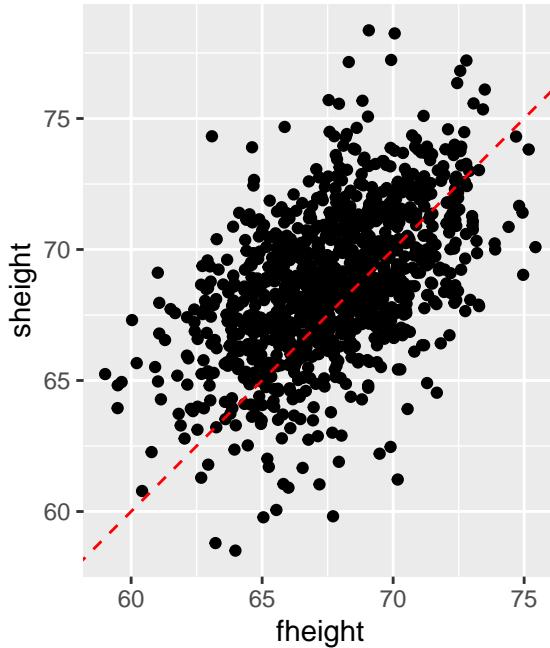
We’re going to explore this phenomenon using Karl Pearson’s (another founding father of statistics) data from 1903, recording the height of fathers and sons:

```
require(UsingR)
data("father.son")
pl <- ggplot(data = father.son) + aes(x = fheight, y = sheight) + geom_point() + coord_equa
pl
```



Let's add the 1:1 line for comparison:

```
pl + geom_abline(slope = 1, intercept = 0, linetype = 2, color = "red")
```



You can see that the sons tend to be taller than their fathers. Let's see of how much:

```
mean(father.son$fheight)
```

```
[1] 67.6871
```

```
mean(father.son$sheight)
```

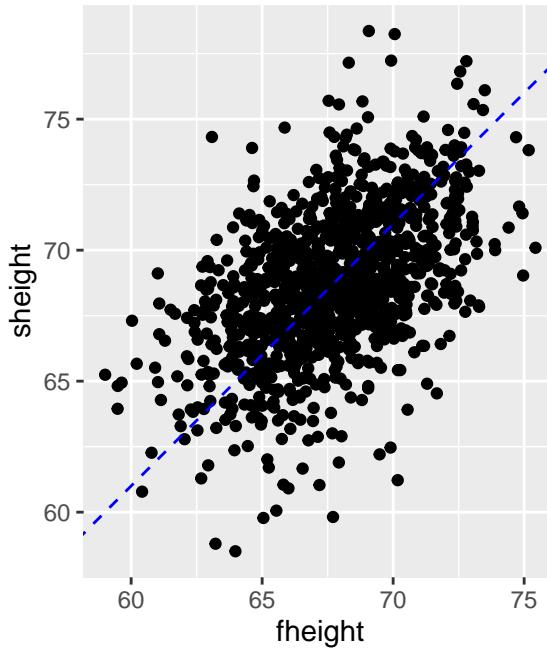
```
[1] 68.68407
```

```
# difference  
mean(father.son$sheight) - mean(father.son$fheight)
```

```
[1] 0.9969728
```

So let's add a line with an intercept of 1:

```
pl <- pl + geom_abline(slope = 1, intercept = 1, linetype = 2, color = "blue")  
pl
```



You can see that the line does not divide the cloud of points evenly: even though tall fathers tend to produce tall sons, and short fathers short sons, the sons of short fathers tend to be taller than their fathers (for example, look at the sons of fathers less than 60 inches tall), while the sons of tall fathers tend to be shorter than their fathers (for example, the sons of fathers taller than 75 inches).

This phenomenon is called “regression toward the mean”: when you take two measurement on the same sample (or related samples, as here), if a variable is extreme on its first measurement, it will tend to be closer to the average on its second measurement; if it is extreme on its second measurement, it will tend to have been closer to the average on its first.

Regression to the mean: dangers of interpretation

- A city sees an unusual growth of crime in a given neighborhood, and they decide to patrol the neighborhood more heavily. The next year, crime rates are close to normal. Was this due to heavy presence of police?
- A teacher sees that scolding students who've had a very low score in a test makes them perform better in the next test. (But would praising those with unusually high scores lead to slacking off in the next test?)
- A huge problem in science: effect sizes tend to decrease through time. Problem of selective reporting?

This phenomenon gave the name to one of the simplest statistical models: the linear regression.

9.2 Finding the best fitting line: Linear Regression

How can we explain the relationship between the height of the fathers and those of their sons? One of the simplest models we can use is called a “Linear Model”. Basically, we want to express the height of the son as a function of the height of the father:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where y_i is the height of the son (**response variable**), x_i is the height of the father (**explanatory variable**), β_0 and β_1 are two numbers (intercept and slope of the line) that do not vary within the population (these are the parameters we want to fit). Finally, the term ϵ_i measures the “error” we are making for the i^{th} son. For simplicity, we assume the $\epsilon_i \stackrel{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$ (and σ is therefore another parameter we want to fit).

When we have multiple explanatory variables (for example, if we had recorded also the height of the mother, whether the son was born at full term or premature, the average caloric intake for the family, etc.), we speak of **Multiple Linear Regression**:

$$y_i = \beta_0 + \sum_{k=1}^n \beta_k x_{ik} + \epsilon_i$$

9.2.1 Solving a linear model — some linear algebra

In this section, we're going to look at the mechanics of linear regression. Suppose that for simplicity we have a single explanatory variable, then we can write the linear model in compact form as:

$$\mathbf{Y} = \mathbf{X} +$$

where:

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

Solving the linear regression means finding the best-fitting β_0 , β_1 and σ (controlling the spread of the distribution of the ϵ_i). Our goal is to find the values of β that minimize σ (meaning that the points fall closer to the line). Rearranging:

$$\sum_i \epsilon_i^2 = \sum_i (y_i - \beta_0 - \beta_1 x_i)^2 = \|\mathbf{Y} - \mathbf{X}\|$$

As such, we want to find the vector β that minimizes the norm $\|\mathbf{Y} - \mathbf{X}\|$. One can prove that this is accomplished using:

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

Where the matrix $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ is known as the (left) Moore-Penrose pseudo-inverse of \mathbf{X} . Let's try to do this in R (the "hard" way):

```
X <- cbind(1, father.son$fheight)
Y <- cbind(father.son$sheight)
best_beta <- solve(t(X) %*% X) %*% t(X) %*% Y
best_beta
```

```
[,1]
[1,] 33.886604
[2,] 0.514093
```

We find that the best fitting line has an intercept of about 34 inches, and a slope of 0.51. Of course, R can do this calculation for you with just one command:

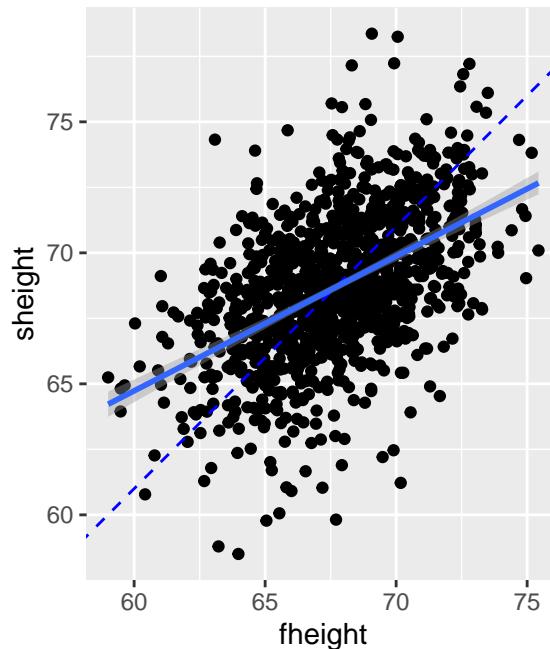
```
best_beta_easy <- lm(sheight ~ fheight, data = father.son)
best_beta_easy
```

```
Call:
lm(formula = sheight ~ fheight, data = father.son)
```

```
Coefficients:
(Intercept)      fheight
            33.8866          0.5141
```

But it feels good to know that this is not black magic! In fact, plotting it on top of the data does not even require computing the coefficients:

```
pl + geom_smooth(method = "lm") # lm stands for linear model
```



9.2.2 Minimizing the sum of squares

What we just did is called “ordinary least-squares”: we are trying to minimize the distance from the data points to their projection on the best-fitting line. We can compute the “predicted” heights as:

$$\hat{\mathbf{Y}} = \mathbf{X}^*$$

Then, we’re minimizing $\|\mathbf{Y} - \hat{\mathbf{Y}}\|$. We call $\hat{\mathbf{Y}} = \mathbf{Y} - \hat{\mathbf{Y}}$ the vector of **residuals**. From this, we can estimate the final parameter, σ :

$$\sigma = \sqrt{\frac{\sum_i \hat{\epsilon}_i^2}{n - p}}$$

where n is the number of data points, and p is the number of parameters in (2 in this case); this measures the number of **degrees of freedom**. Let’s try to compute it:

```
degrees_of_freedom <- length(Y) - 2
degrees_of_freedom
```

```
[1] 1076
```

```
epsilon_hat <- X %*% best_beta - Y
sigma <- sqrt(sum(epsilon_hat^2) / degrees_of_freedom)
sigma
```

```
[1] 2.436556
```

In R, you will find this reported as the **Residual standard error** when you call **summary** on your model:

```
summary(best_beta_easy)
```

```
Call:
lm(formula = sheight ~ fheight, data = father.son)
```

```
Residuals:
```

```

      Min       1Q   Median       3Q      Max
-8.8772 -1.5144 -0.0079  1.6285  8.9685

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 33.88660   1.83235   18.49 <2e-16 ***
fheight     0.51409   0.02705   19.01 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.437 on 1076 degrees of freedom
Multiple R-squared:  0.2513,    Adjusted R-squared:  0.2506
F-statistic: 361.2 on 1 and 1076 DF,  p-value: < 2.2e-16

```

Finally, the **coefficient of determination** R^2 is computed as:

$$R^2 = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}$$

where \bar{y} is the mean of y_i . If the regression has an intercept, then the R^2 can vary between 0 and 1, with values close to 1 indicating a good fit to the data. Again, let's compute it the hard way and then the easy way:

```

y_bar <- mean(Y)
R_2 <- sum((X %*% best_beta - y_bar)^2) / sum((Y - y_bar)^2)
R_2

```

```
[1] 0.2513401
```

```

# look for Multiple R-squared:
summary(best_beta_easy)

```

```

Call:
lm(formula = sheight ~ fheight, data = father.son)

Residuals:
      Min       1Q   Median       3Q      Max
-8.8772 -1.5144 -0.0079  1.6285  8.9685

```

```

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 33.88660   1.83235   18.49 <2e-16 ***
fheight      0.51409   0.02705   19.01 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.437 on 1076 degrees of freedom
Multiple R-squared:  0.2513,    Adjusted R-squared:  0.2506 
F-statistic: 361.2 on 1 and 1076 DF,  p-value: < 2.2e-16

```

9.2.3 Assumptions of linear regression

In practice, when we are performing a linear regression, we are making a number of assumptions about the data. Here are the main ones:

- Model structure: we assume that the process generating the data is linear.
- Explanatory variable: we assume that this is measured without errors (!).
- Residuals: we assume that residuals are i.i.d. Normal.
- Strict exogeneity: the residuals should have conditional mean of 0.

$$\mathbb{E}[\epsilon_i | x_i] = 0$$

- No linear dependence: the columns of \mathbf{X} should be linearly independent.
- Homoscedasticity: the variance of the residuals is independent of x_i .

$$\mathbb{V}[\epsilon_i | x_i] = \sigma^2$$

- Errors are uncorrelated between observations.

$$\mathbb{E}[\epsilon_i \epsilon_j | x] = 0 \quad \forall j \neq i$$

9.3 Linear regression in action

To perform a slightly more complicated linear regression, we take the data from:

Piwowar HA, Day RS, Fridsma DB (2007) [Sharing detailed research data is associated with increased citation rate](#). PLoS ONE 2(3): e308.

The authors set out to demonstrate that sharing data accompanying papers tends to increase the number of citations received by the paper.

```
# original URL
# https://datadryad.org/stash/dataset/doi:10.5061/dryad.j2c4g
dat <- read_csv("data/Piwowar_2011.csv")
# rename variables for easier handling
dat <- dat %>% rename(IF = `Impact factor of journal`,
                           NCIT = `Number of Citations in first 24 months after publication`,
                           SHARE = `Is the microarray data publicly available`) %>%
  dplyr::select(NCIT, IF, SHARE)
```

First, let's run a model in which the logarithm of the number of citations + 1 is regressed against the "Impact Factor" of the journal (which is a measure of "prestige" based on the average number of citations per paper received):

```
my_model <- lm(log(NCIT + 1) ~ log(IF + 1), data = dat)
summary(my_model)
```

Call:

```
lm(formula = log(NCIT + 1) ~ log(IF + 1), data = dat)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.65443	-0.44272	-0.00769	0.43414	1.62817

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.1046	0.2951	0.355	0.724
log(IF + 1)	1.2920	0.1196	10.802	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6887 on 83 degrees of freedom

Multiple R-squared: 0.5844, Adjusted R-squared: 0.5794

F-statistic: 116.7 on 1 and 83 DF, p-value: < 2.2e-16

You can see that the higher the impact factor, the higher the number of citations received (unsurprisingly!). Now let's add another variable, detailing whether publicly available data accompany the paper:

```
my_model2 <- lm(log(NCIT + 1) ~ log(IF + 1) + SHARE, data = dat)
summary(my_model2)
```

Call:

```
lm(formula = log(NCIT + 1) ~ log(IF + 1) + SHARE, data = dat)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.98741	-0.43768	0.08726	0.41847	1.35634

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.4839	0.3073	1.575	0.11918
log(IF + 1)	1.0215	0.1442	7.084	4.4e-10 ***
SHARE	0.5519	0.1802	3.062	0.00297 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6564 on 82 degrees of freedom

Multiple R-squared: 0.627, Adjusted R-squared: 0.6179

F-statistic: 68.92 on 2 and 82 DF, p-value: < 2.2e-16

We find that sharing data is associated with a larger number of citations.

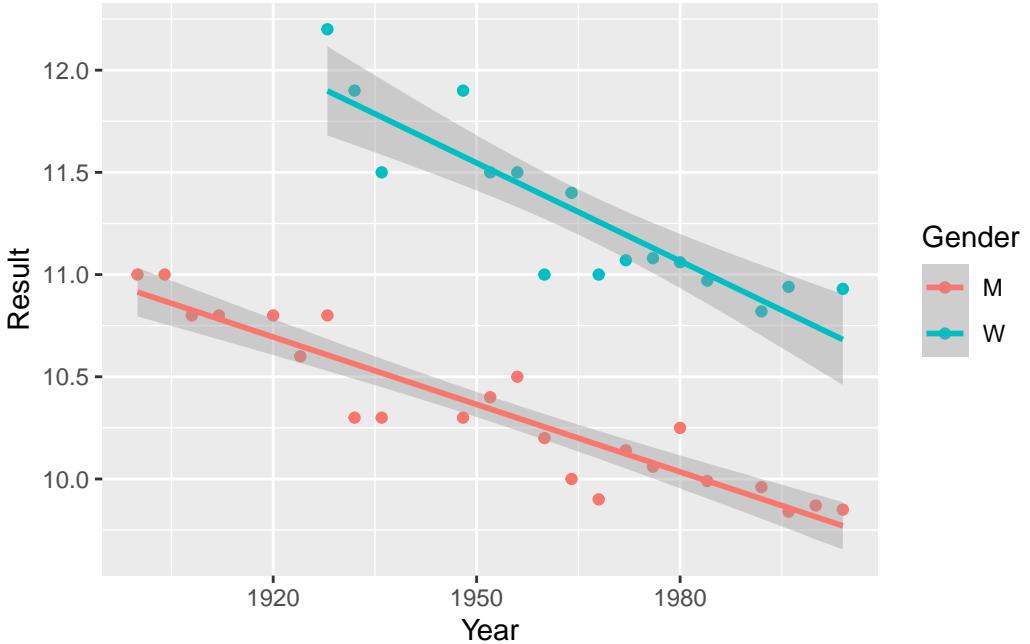
9.4 A regression gone wild

Even when the fit is good, and assumptions are met, one can still end up with a fantastic blunder. To show this, we are going to repeat a study published in *Nature* (no less!) by Tatem *et al.* You can find the study [here](#). Briefly, the Authors gathered data on the 100m sprint at the Olympics from 1900 to 2004, for both men and women. We can do the same:

```
olympics <- read_csv("data/100m_dash.csv")
```

Then, they fitted a linear regression through the points, for both men and women. So far, so good:

```
ggplot(data = olympics %>% filter(Year > 1899, Year < 2005)) +
  aes(x = Year, y = Result, colour = Gender) +
  geom_point() + geom_smooth(method = "lm")
```



The fit is quite good:

```
summary(lm(Result ~ Year*Gender,
  data = olympics %>% filter(Year > 1899, Year < 2005)))
```

Call:

```
lm(formula = Result ~ Year * Gender, data = olympics %>% filter(Year >
  1899, Year < 2005))
```

Residuals:

Min	1Q	Median	3Q	Max
-0.38617	-0.05428	-0.00071	0.08239	0.32174

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	31.808278	2.179491	14.594	< 2e-16 ***
Year	-0.010997	0.001116	-9.855	1.24e-11 ***
GenderW	10.952646	4.371678	2.505	0.0170 *
Year:GenderW	-0.005011	0.002228	-2.249	0.0309 *

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Residual standard error: 0.1707 on 35 degrees of freedom
Multiple R-squared:  0.9304,    Adjusted R-squared:  0.9244
F-statistic: 155.9 on 3 and 35 DF,  p-value: < 2.2e-16

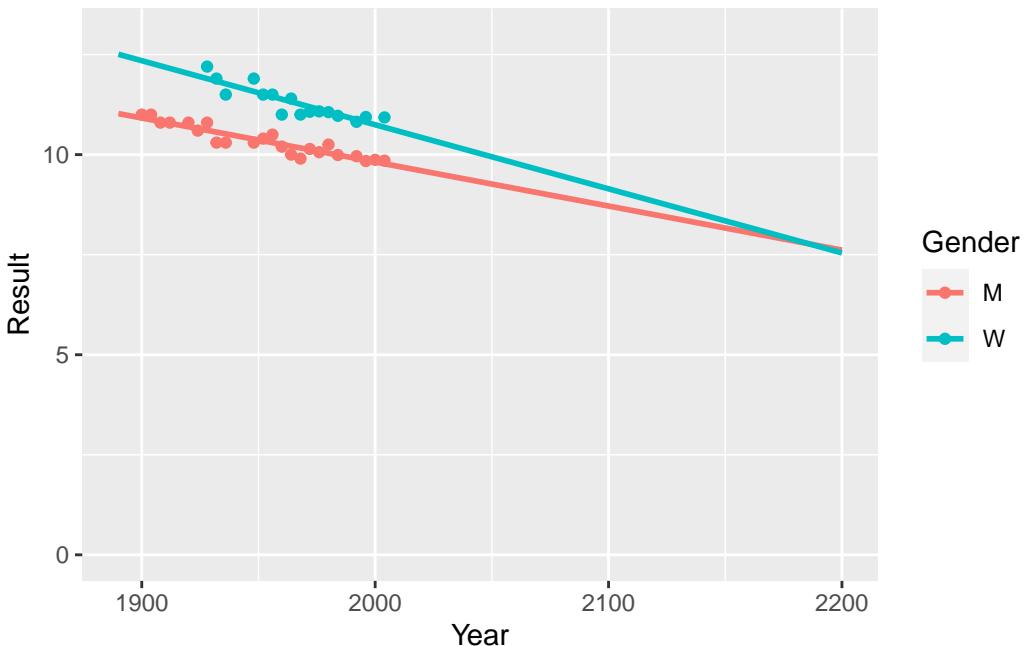
```

An R^2 of 0.93, the pinnacle of a good linear regression. Now however, comes the problem. The Authors noticed that the times recorded for women are falling faster than those for men, meaning that the gender gap is reducing. Will it ever disappear? Just extend the regression and project forward:

```

ggplot(data = olympics %>% filter(Year > 1899, Year < 2005)) +
  aes(x = Year, y = Result, colour = Gender) +
  geom_point() + geom_smooth(method = "lm", fullrange = TRUE, se = FALSE) +
  xlim(c(1890, 2200)) + ylim(c(0, 13))

```



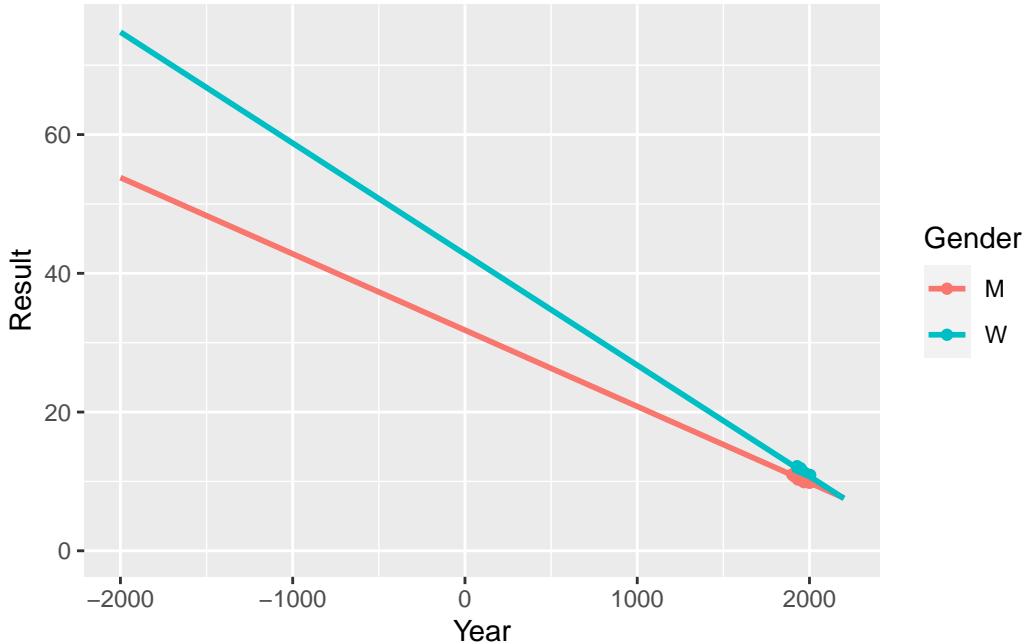
You can see that the lines are touching in sometimes before 2200! Then women will overrun men.

There are a number of things that are wrong with this result. First, by the same logic, computers will soon go faster than the speed of light, the number of people on planet Earth will be in the hundreds of billions, and the price of sequencing will drop so much that we will be paid instead of paying to get our samples sequenced...

Second, if we extend backwards, rather than forward, we would find that Roman women would take more than a minute to run 100m (possibly, because of the uncomfortable tunics

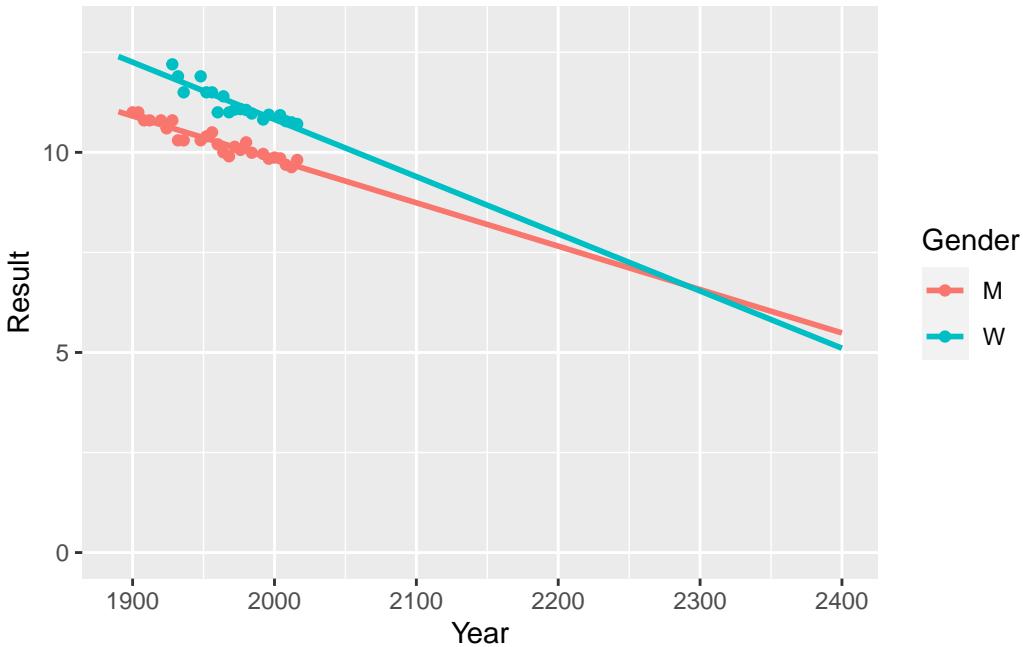
and sandals...).

```
ggplot(data = olympics %>% filter(Year > 1899, Year < 2005)) +  
  aes(x = Year, y = Result, colour = Gender) +  
  geom_point() + geom_smooth(method = "lm", fullrange = TRUE, se = FALSE) +  
  xlim(c(-2000, 2200)) + ylim(c(0, 75))
```



As Neil Bohr allegedly said (but this is disputed), “Prediction is very difficult, especially about the future”. The fact is that any non-linear curve looks quite linear if we are only considering a small range of values on the x-axis. To prove this point, let’s add the data from 2004 to today:

```
ggplot(data = olympics %>% filter(Year > 1899)) +  
  aes(x = Year, y = Result, colour = Gender) +  
  geom_point() + geom_smooth(method = "lm", fullrange = TRUE, se = FALSE) +  
  xlim(c(1890, 2400)) + ylim(c(0, 13))
```



You can see that the process has already slowed down: now it would take an extra century before the “momentous sprint”.

So many things were wrong with this short paper, that *Nature* was showered with replies. My favorite is from a Cambridge statistician (the Authors were from Oxford, ça va sans dire); it is perfectly short and venomous—a good candidate for the Nobel prize in Literature!

Sir — A. J. Tatem and colleagues calculate that women may outsprint men by the middle of the twenty-second century (*Nature* 431, 525; 2004). They omit to mention, however, that (according to their analysis) a far more interesting race should occur in about 2636, when times of less than zero seconds will be recorded. In the intervening 600 years, the authors may wish to address the obvious challenges raised for both time-keeping and the teaching of basic statistics. — Kenneth Rice

9.5 More advanced topics

9.5.1 Categorical variables in linear models

In the example above, we have built the model:

$$\log(\text{NCIT} + 1) = \beta_0 + \beta_1 (\log(\text{IF} + 1))_i + \beta_2 (\text{SHARE})_i + \epsilon_i$$

In this case, the variable SHARE takes values of 1 or 0. As such, when the data were not shared (SHARE = 0) the model reduces to the previous one, in which β_2 was absent. The coefficient β_2 measures the increase in the log of citation count when data are shared.

The same approach can be taken whenever you have categorical values: R will automatically create **dummy variables** each encoding whether the i th data point belongs to a particular category. For example, suppose you want to predict the height of a child based on the height of the father, and that you also collected the gender, in three categories: F for female, M for male, U for unknown. Then you could use this information to build the model:

$$\text{height}_i = \beta_0 + \beta_1(\text{height of father})_i + \beta_2(\text{gender is M})_i + \beta_3(\text{gender is U})_i + \epsilon_i$$

where the variable **gender is M** takes value 1 when the gender is M and 0 otherwise, and **gender is U** takes value 1 when the gender is unknown and 0 otherwise. As such, when the gender is F both variables will be zero, and β_2 and β_3 measure the increase (or decrease) in height for males and those with unspecified gender, respectively. While R does this for you automatically, understanding what is going on “under the hood” is essential for interpreting the results.

9.5.2 Interactions in linear models

Sometimes we think that our explanatory variables could “interact”. For example, suppose you want to predict the BMI of people. What we have available is the average caloric intake, the height, gender, and whether they are vegetarian, vegan, or omnivores. A simple model could be:

$$\text{BMI}_i = \beta_0 + \beta_h \text{height}_i + \beta_c \text{calories}_i + \beta_g \text{gender}_i + \epsilon_i$$

We could add the type of diet as a factor:

$$\text{BMI}_i = \beta_0 + \beta_h \text{height}_i + \beta_c \text{calories}_i + \beta_g \text{gender}_i + \beta_d \text{diet}_i + \epsilon_i$$

However, suppose that we believe the type of diet to affect differentially men and women. Then, we would like to create an “interaction” (e.g., paleo-female, vegan-male):

$$\text{BMI}_i = \beta_0 + \beta_h \text{height}_i + \beta_c \text{calories}_i + \beta_g \text{gender}_i + \beta_d \text{diet}_i + \beta_{gd} \text{gender:diet}_i + \epsilon_i$$

where the colon signals “interaction”. In R, this would be coded as `lm(BMI ~ height + calories + gender * diet)`. A simpler model is one in which we only account for the `gender:diet` interaction, but not for the separate effects of gender and diet:

$$\text{BMI}_i = \beta_0 + \beta_h \text{height}_i + \beta_c \text{calories}_i + \beta_{gd} \text{gender:diet}_i + \epsilon_i$$

which in R can be coded as `lm(BMI ~ height + calories + gender:diet)`. Finally, for some models you believe the intercept should be 0 (note that this makes the R^2 statistics uninterpretable!). In R, just put -1 at the end of the definition of the model (e.g., `lm(BMI ~ height + calories + gender:diet - 1)`).

9.5.3 Regression diagnostics

Now that we know the mechanics of linear regression, we turn to diagnostics: how can we make sure that the model fits the data “well”? We start by analyzing a data set assembled by Anscombe (*The American Statistician*, 1973)

```
dat <- read_csv("data/Anscombe_1973.csv")
```

The file comprised four data sets. We perform a linear regression using each data set separately:

```
lm(Y ~ X, data = dat %>% filter(Data_set == "Data_1"))
```

Call:

```
lm(formula = Y ~ X, data = dat %>% filter(Data_set == "Data_1"))
```

Coefficients:

(Intercept)	X
3.0001	0.5001

```
lm(Y ~ X, data = dat %>% filter(Data_set == "Data_2"))
```

Call:

```
lm(formula = Y ~ X, data = dat %>% filter(Data_set == "Data_2"))
```

Coefficients:

(Intercept)	X
3.001	0.500

```
lm(Y ~ X, data = dat %>% filter(Data_set == "Data_3"))
```

Call:

```
lm(formula = Y ~ X, data = dat %>% filter(Data_set == "Data_3"))
```

Coefficients:

(Intercept)	X
3.0025	0.4997

```
lm(Y ~ X, data = dat %>% filter(Data_set == "Data_4"))
```

Call:

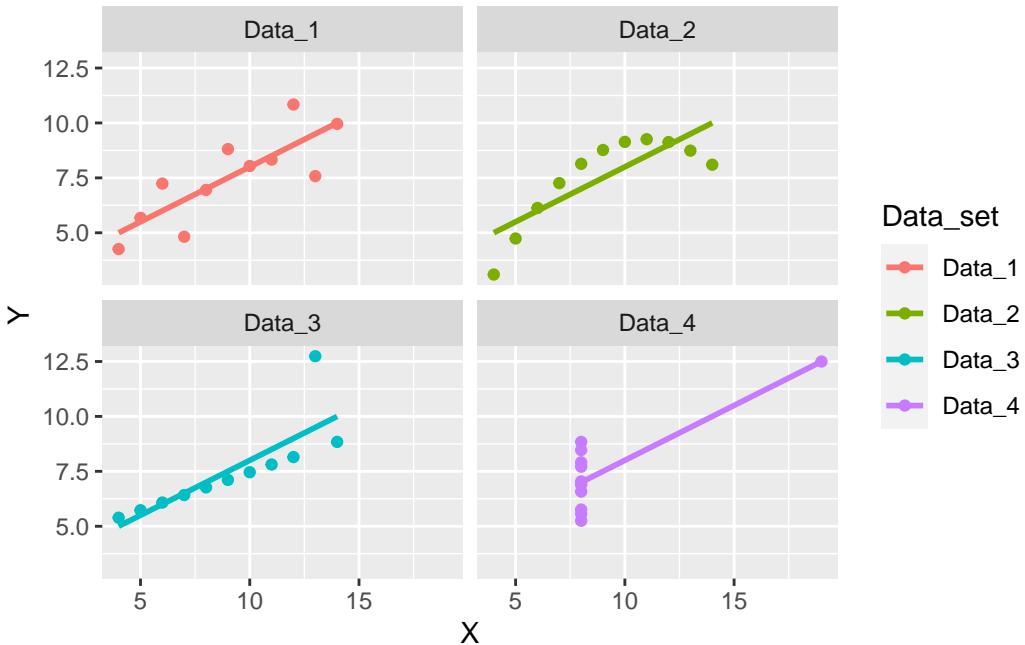
```
lm(formula = Y ~ X, data = dat %>% filter(Data_set == "Data_4"))
```

Coefficients:

(Intercept)	X
3.0017	0.4999

As you can see, each data set is best fit by the same line, with intercept 3 and slope $\frac{1}{2}$. Plotting the data, however, shows that the situation is more complicated:

```
ggplot(data = dat) + aes(x = X, y = Y, colour = Data_set) +
  geom_point() + geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~Data_set)
```

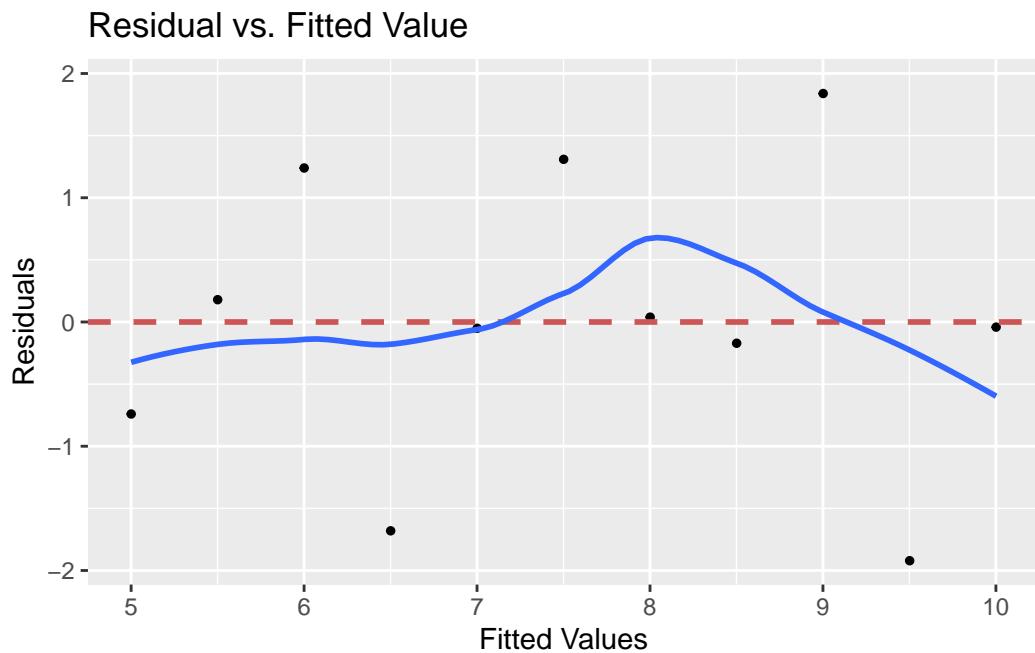


Data_1 is fitted quite well; Data_2 shows a marked nonlinearity; all points but one in Data_3 are on the same line, but a single **outlier** shifts the line considerably; finally, in Data_4 a single point is responsible for the fitting line: all other values of X are exactly the same. Inspecting the graphs, we would conclude that we can trust our model only in the first case. When you are performing a multiple regression, however, it is hard to see whether we're in case 1, or one of the other cases. R provides a number of diagnostic tools which can help you decide whether the fit to the data is good.

9.5.4 Plotting the residuals

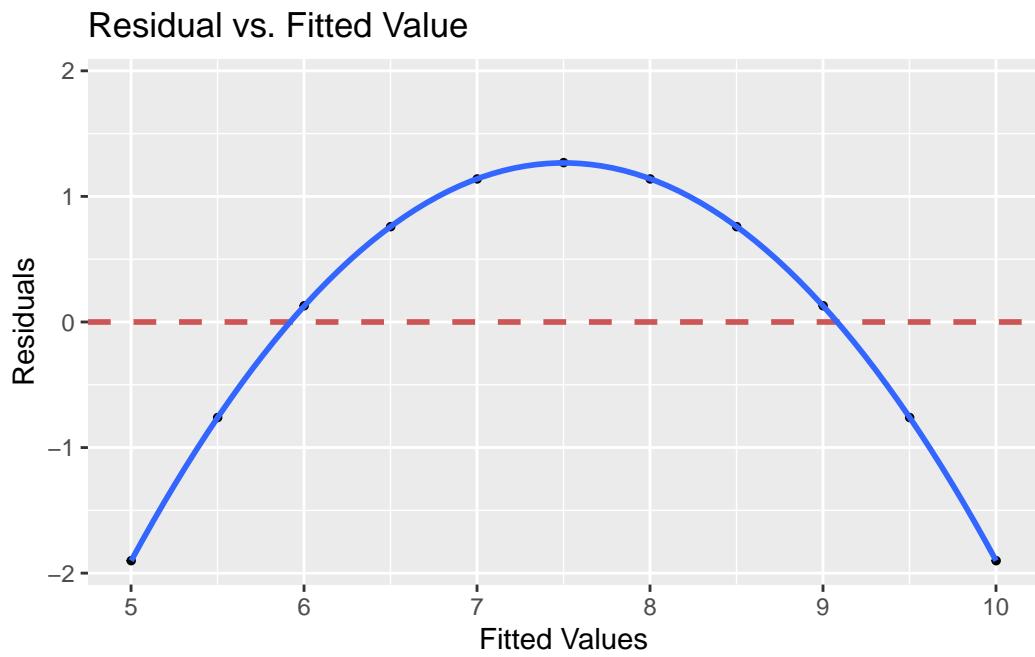
The first thing you want to do is to plot the residuals as a function of the fitted values. This plot should make it apparent whether the data was linear or not. The package `lindia` (linear regression diagnostics) makes it easy to produce this type of plot using `ggplot2`:

```
gg_resfitted(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_1"))) + geom_smooth(method
```

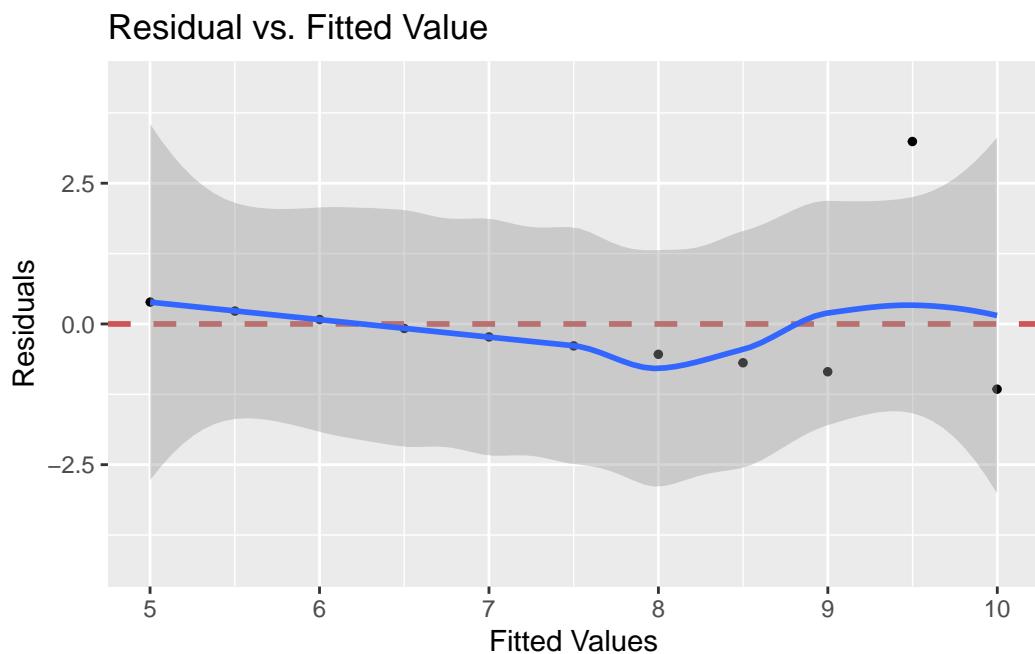


What you are looking for is an approximately flat line, meaning that the residuals are approximately normally distributed with mean zero for each fitted value. This is not the case in the other data sets:

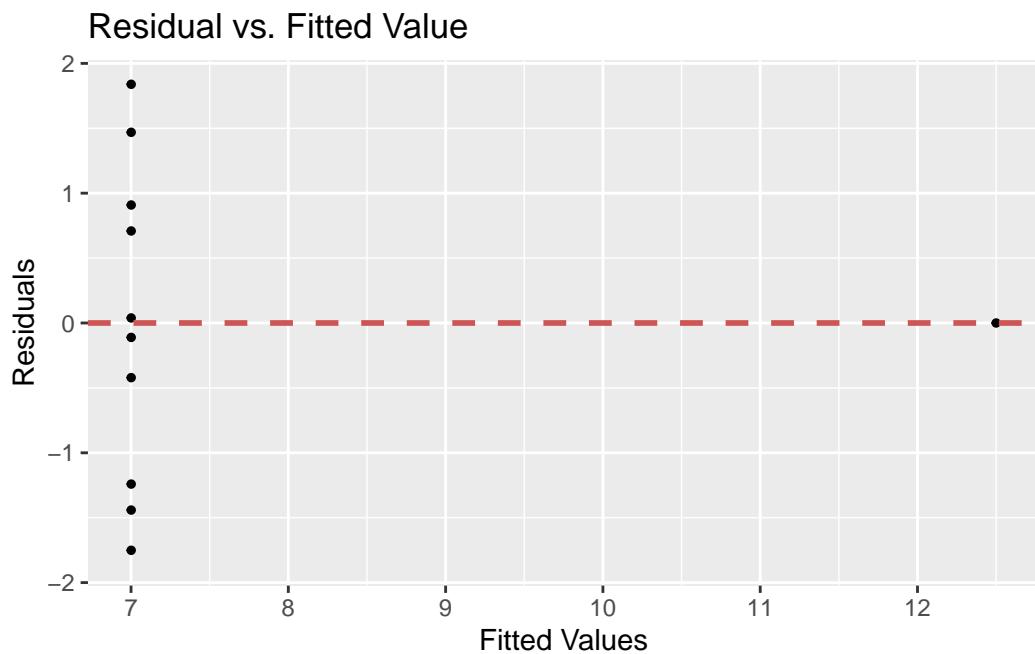
```
gg_resfitted(lm(Y ~ X, data = dat %>%
                  filter(Data_set == "Data_2"))) +
  geom_smooth(method = "loess")
```



```
gg_resfitted(lm(Y ~ X, data = dat %>%
                  filter(Data_set == "Data_3"))) +
  geom_smooth(method = "loess")
```



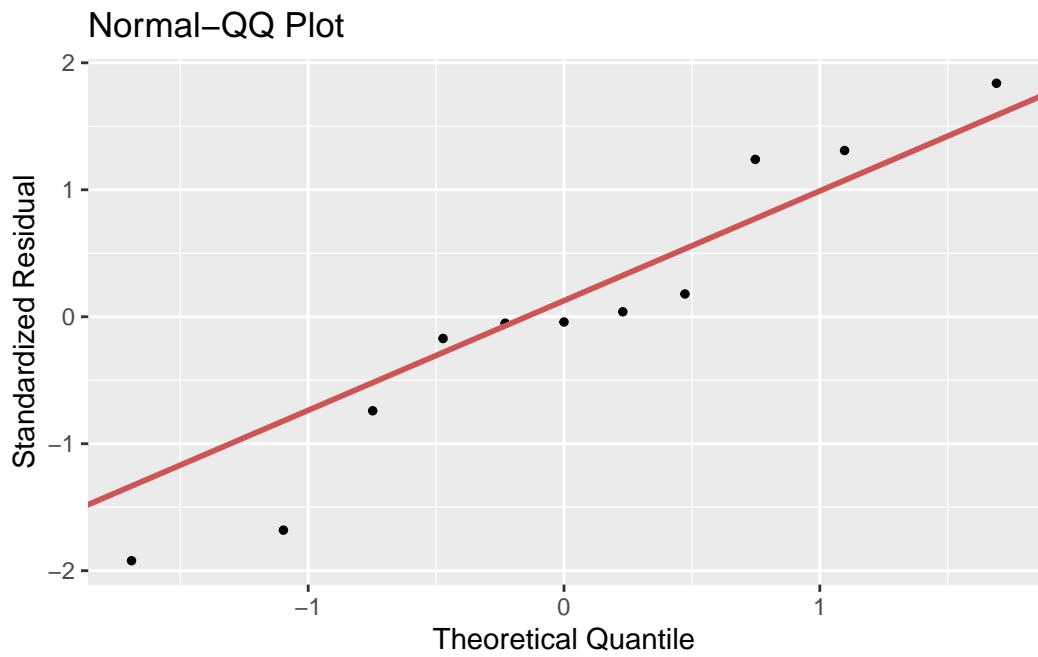
```
gg_resfitted(lm(Y ~ X, data = dat %>%
                  filter(Data_set == "Data_4"))) +
  geom_smooth(method = "loess")
```



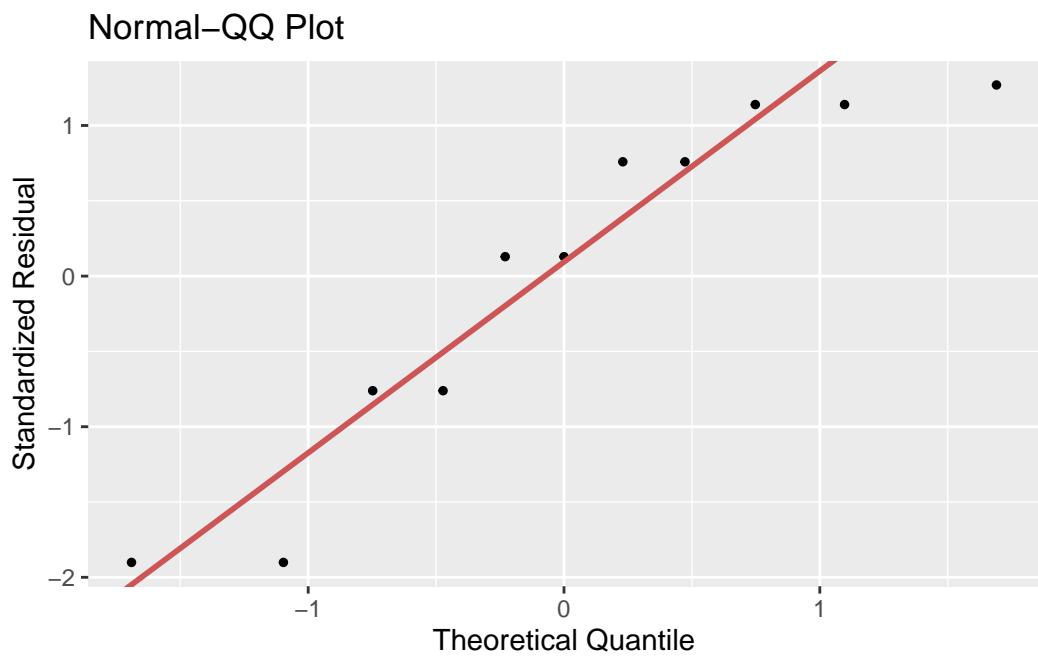
9.5.5 Q-Q Plot

We can take this further, and test whether the residuals follow a normal distribution. In particular, we can estimate the density of the residuals, and plot it against the density of a normal distribution:

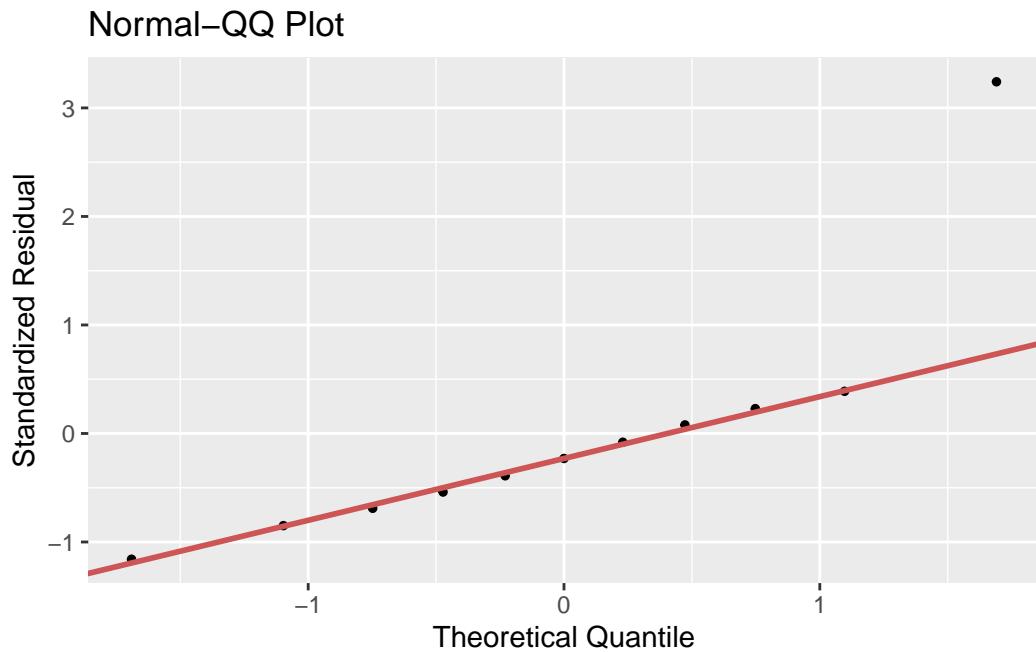
```
gg_qqplot(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_1")))
```



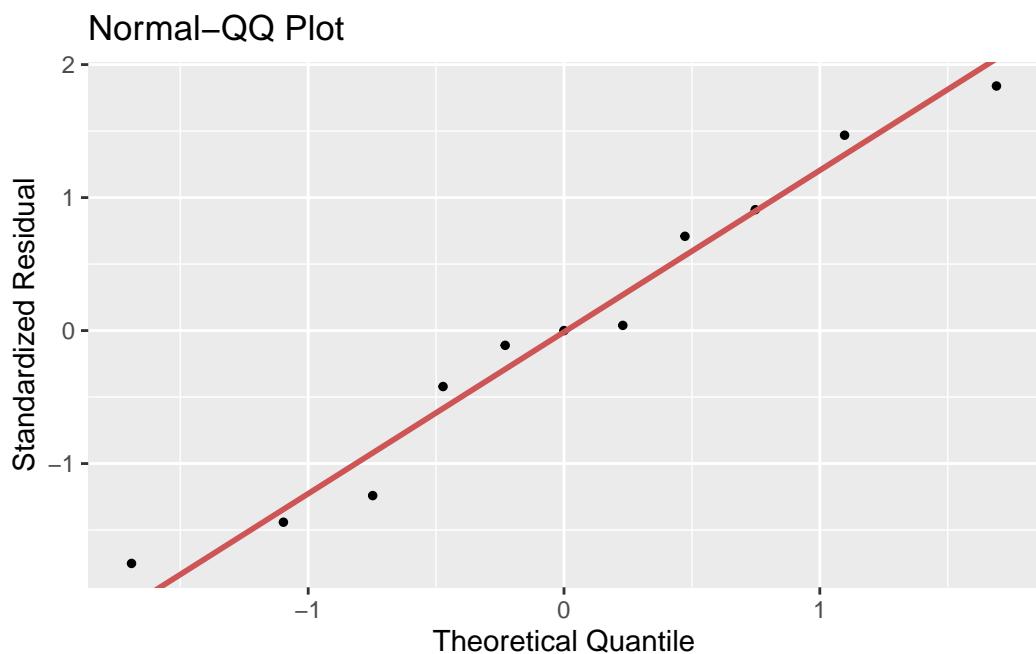
```
gg_qqplot(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_2")))
```



```
gg_qqplot(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_3")))
```



```
gg_qqplot(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_4")))
```

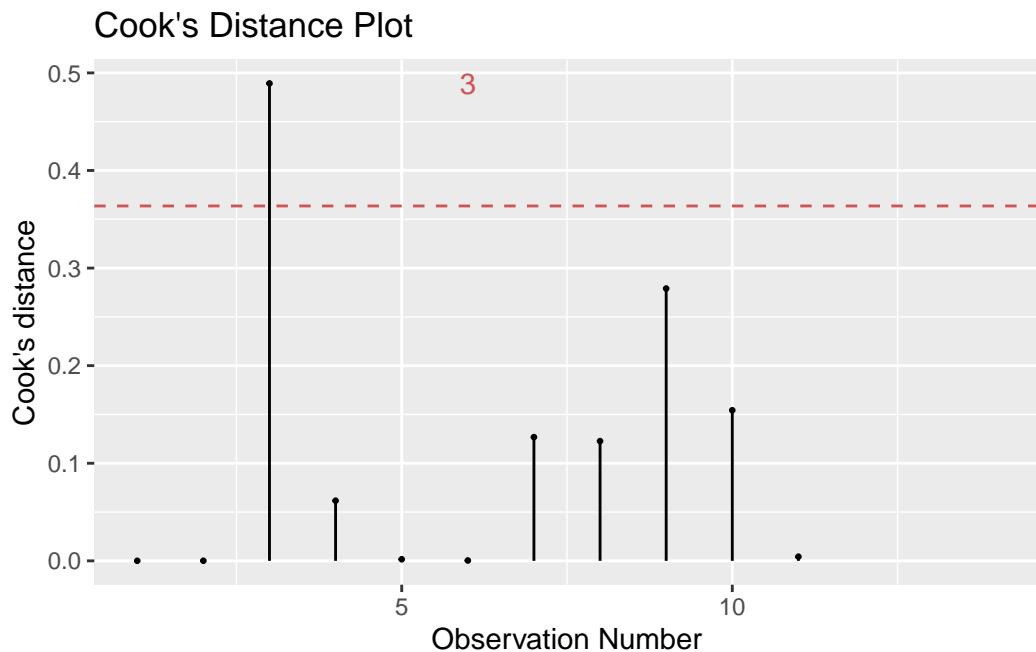


Here, you are looking for a good match to the 1:1 line; outliers will be found far from the line (e.g., case 3).

9.5.6 Cook's distance

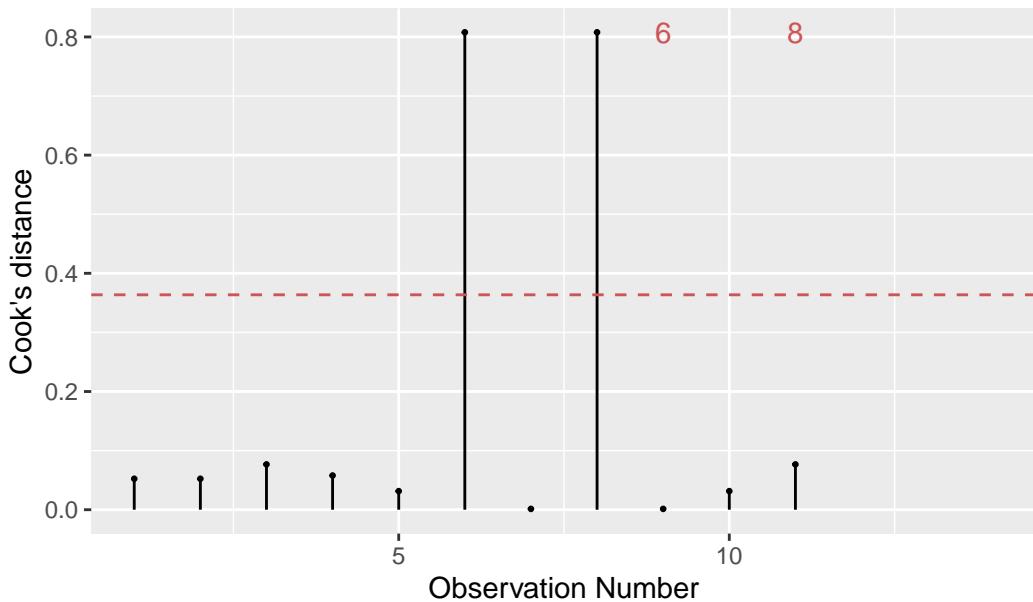
Another way to detect outliers is to compute the Cook's distance for every point. Briefly, this statistic measures the effect on the regression we would obtain if we were to remove a point.

```
gg_cooksd(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_1")))
```



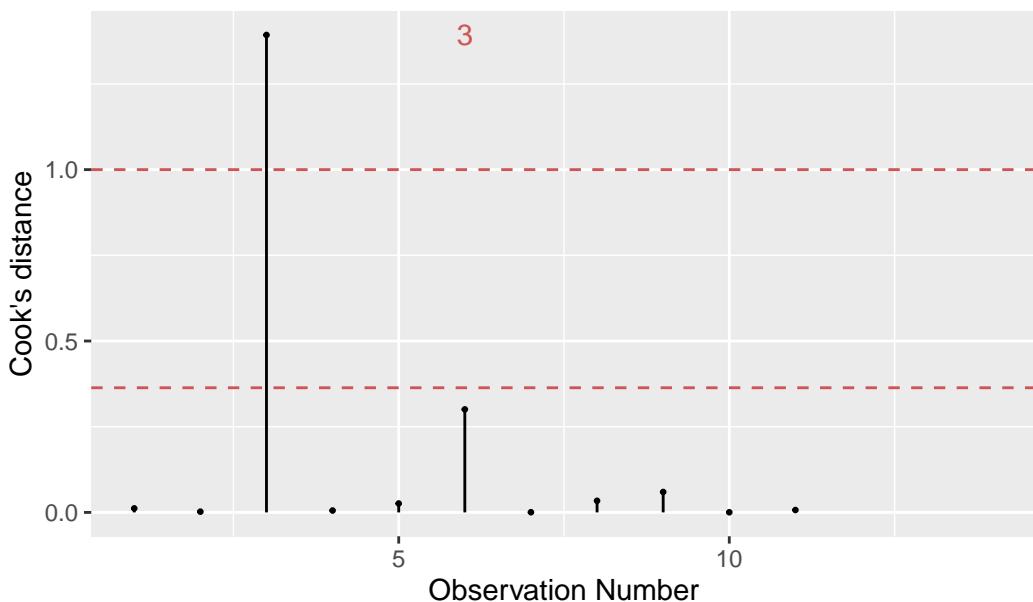
```
gg_cooksd(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_2")))
```

Cook's Distance Plot



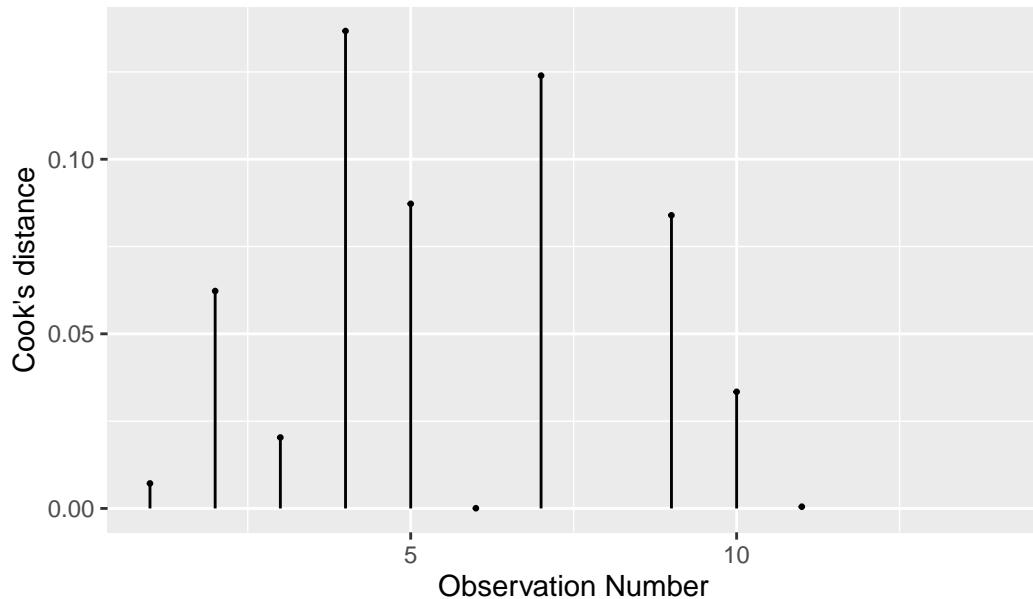
```
gg_cooksd(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_3")))
```

Cook's Distance Plot



```
gg_cooksd(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_4")))
```

Cook's Distance Plot

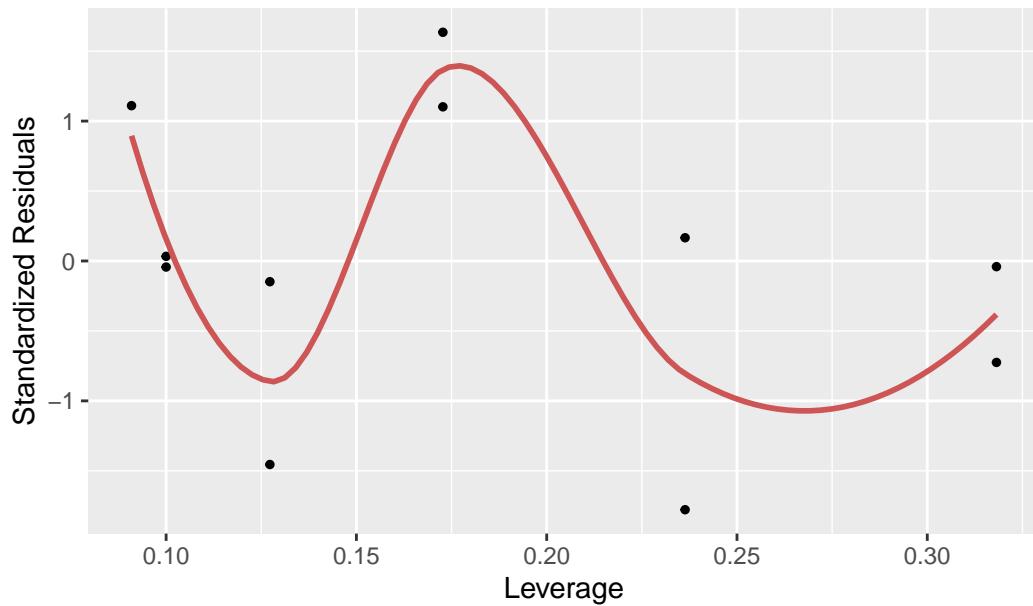


9.5.7 Leverage

Points that strongly influence the regression are said to have much “leverage”:

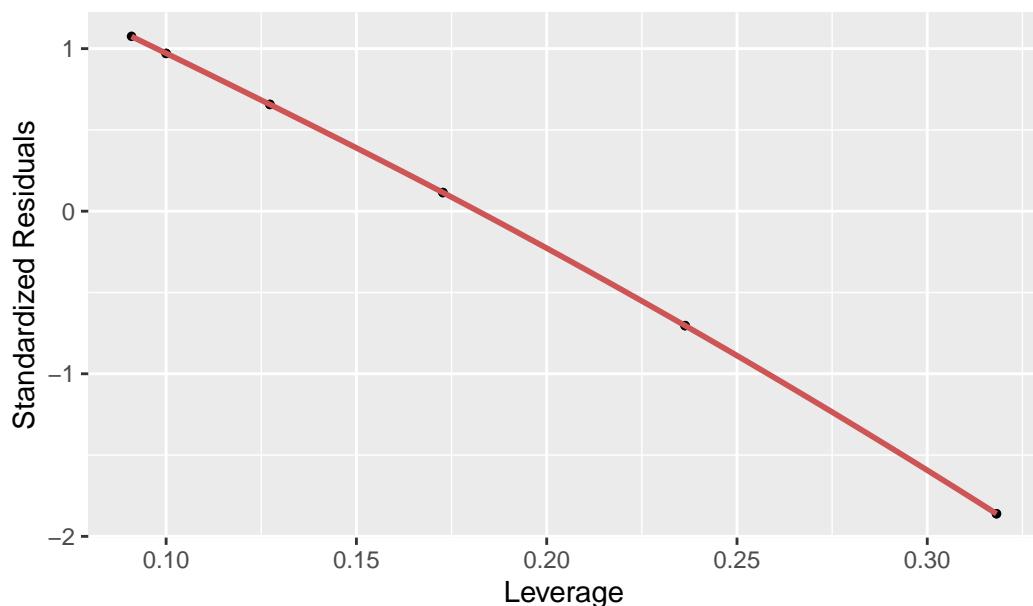
```
gg_resleverage(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_1")))
```

Residual vs. Leverage

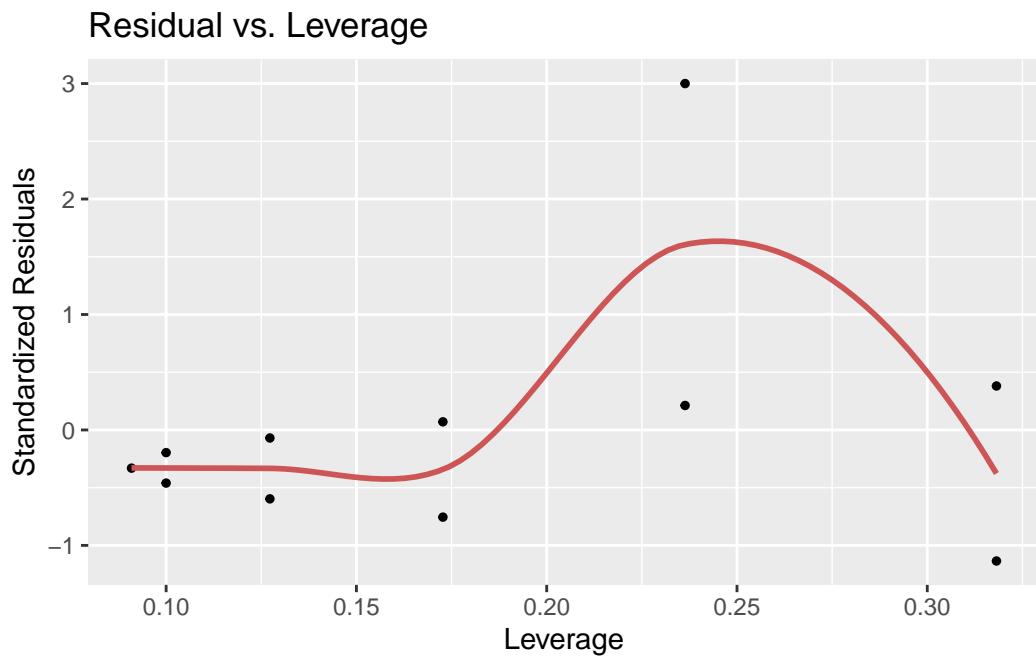


```
gg_resleverage(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_2")))
```

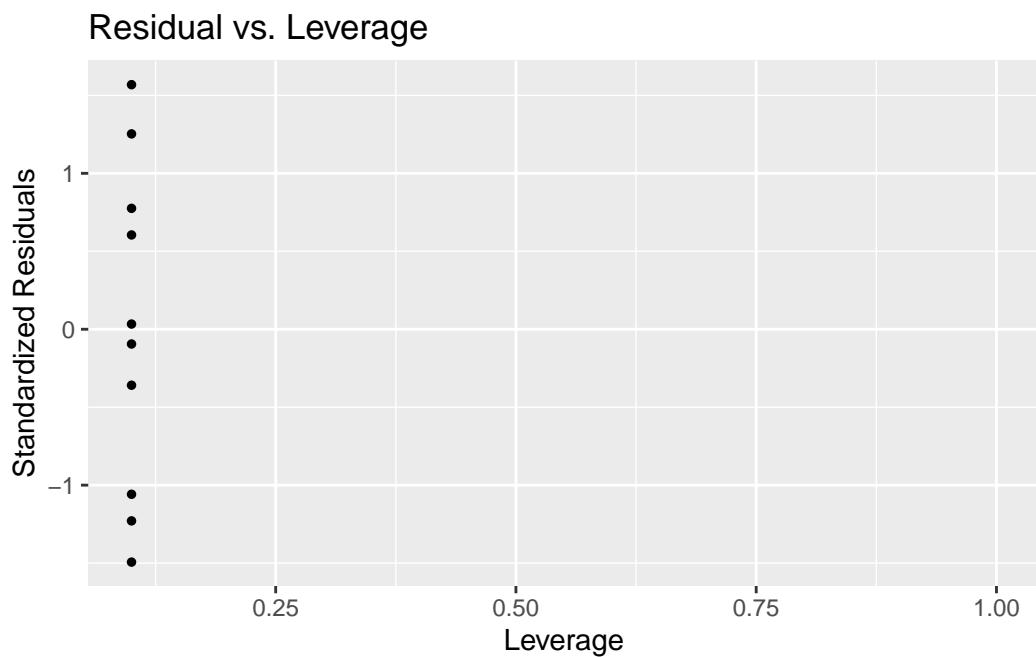
Residual vs. Leverage



```
gg_resleverage(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_3")))
```



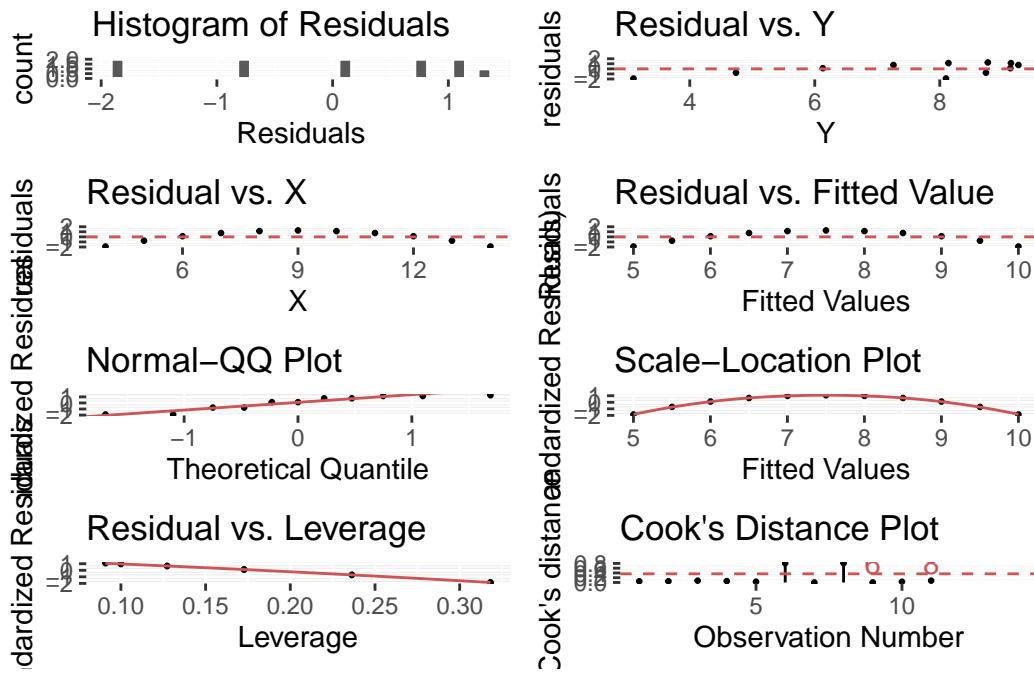
```
gg_resleverage(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_4")))
```



9.5.8 Running all diagnostics

These are but a few of the diagnostics available. To run all diagnostics on a given model, call

```
gg_diagnose(lm(Y ~ X, data = dat %>% filter(Data_set == "Data_2")))
```



9.6 Transforming the data

Often, one needs to transform the data before running a linear regression, in order to fulfill the assumptions. We're going to look at the salary of professors at the University of California to show how this is done.

```
# read the data
# Original URL
dt <- read_csv("https://raw.githubusercontent.com/dailybruin/uc-salaries/master/data/uc_s
                 col_names = c("first_name", "last_name", "title", "a", "pay", "loc", "year"
                 dplyr::select(first_name, last_name, title, loc, pay)
# get only profs
dt <- dt %>% filter(title %in% c("PROF-AY", "ASSOC PROF-AY", "ASST PROF-AY",
                                 "PROF-AY-B/E/E", "PROF-HCOMP", "ASST PROF-AY-B/E/E",
                                 "ASSOC PROF-AY-B/E/E", "ASSOC PROF-HCOMP", "ASST PROF-HCOP"))
```

```

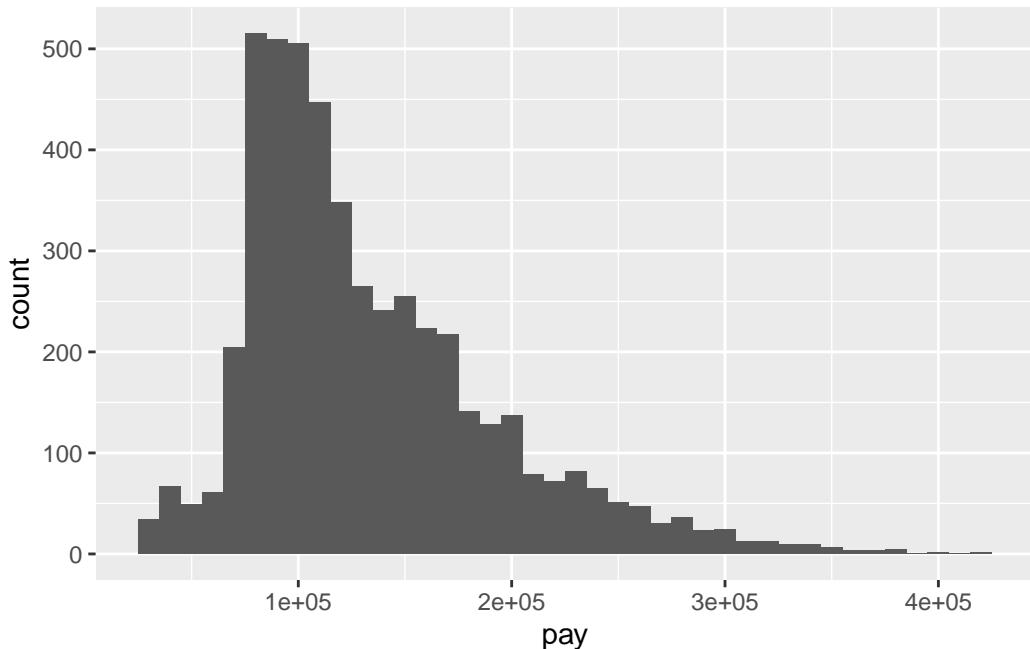
# remove those making less than 30k (probably there only for a period)
dt <- dt %>% filter(pay > 30000)
dt

# A tibble: 4,915 x 5
  first_name    last_name     title      loc       pay
  <chr>        <chr>       <chr>      <chr>     <dbl>
1 CHRISTOPHER U ABANI     PROF-AY    Riverside 151200
2 HENRY DON ISAAC ABARBANEL PROF-AY    San Diego 160450.
3 ADAM R        ABATE      ASST PROF-HCOMP San Francisco 85305.
4 KEVORK N.     ABAZAJIAN   ASST PROF-AY  Irvine    82400.
5 M. ACKBAR     ABBAS      PROF-AY    Irvine    168700.
6 ABUL K        ABBAS      PROF-HCOMP San Francisco 286824.
7 LEONARD J     ABBEDUTO   PROF-HCOMP Davis    200385.
8 DON P          ABBOTT     PROF-AY    Davis    106400.
9 GEOFFREY WINSTON ABBOTT   PROF-HCOMP Irvine   125001.
10 KHALED A.S.    ABDEL-GHAFFAR PROF-AY-B/E/E Davis   120100.
# i 4,905 more rows

```

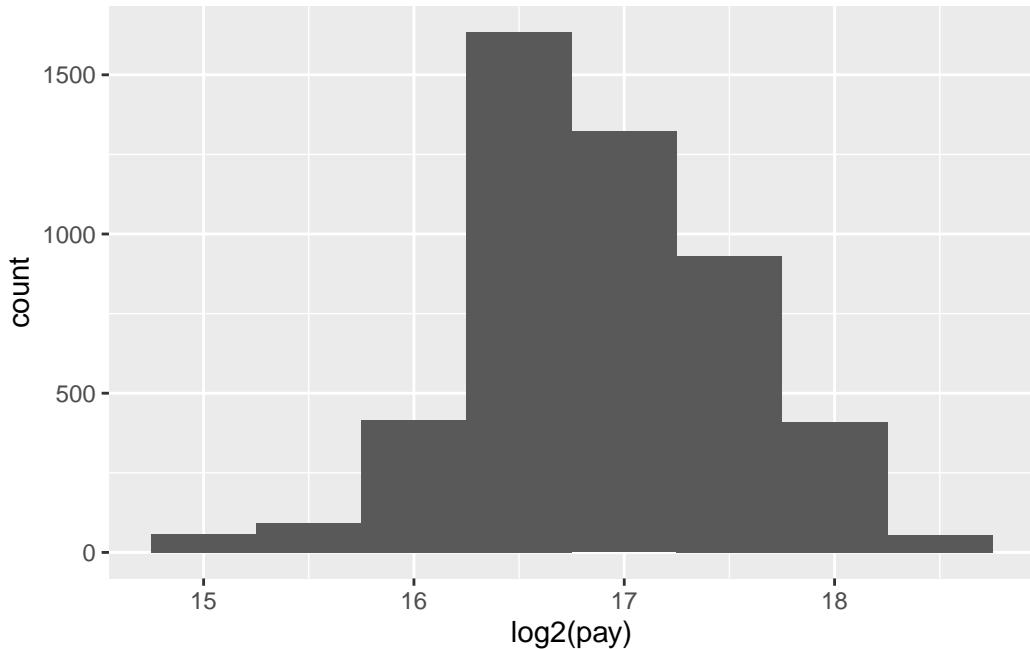
The distribution of salaries is very skewed — it looks like a log-normal distribution:

```
dt %>% ggplot() + aes(x = pay) + geom_histogram(binwidth = 10000)
```



If we set consider the log of pay, we get closer to a normal:

```
dt %>% ggplot() + aes(x = log2(pay)) + geom_histogram(binwidth = 0.5)
```



We can try to explain the pay as a combination of title and location:

```
unscaled <- lm(pay ~ title + loc, data = dt)
summary(unscaled)
```

Call:
lm(formula = pay ~ title + loc, data = dt)

Residuals:
Min 1Q Median 3Q Max
-149483 -25197 -1679 18305 213684

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	98397	2003	49.133	< 2e-16 ***
titleASSOC PROF-AY-B/E/E	46898	3402	13.786	< 2e-16 ***
titleASSOC PROF-HCOMP	25428	3955	6.430	1.40e-10 ***

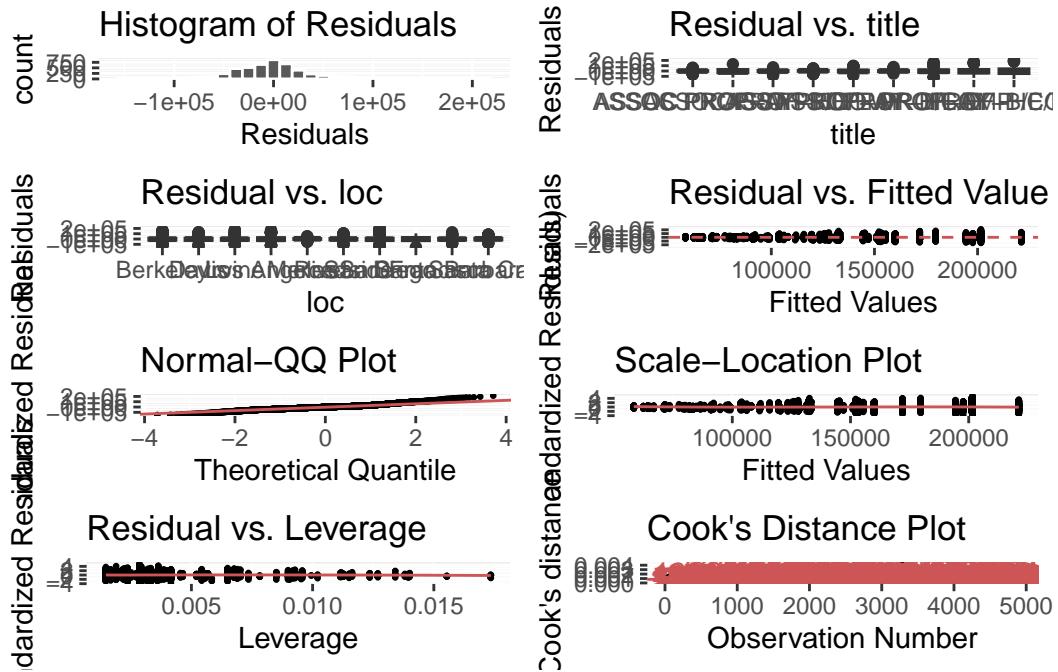
```

titleASST PROF-AY      -15060      2370   -6.356 2.26e-10 ***
titleASST PROF-AY-B/E/E 17405      3949    4.407 1.07e-05 ***
titleASST PROF-HCOMP     5545      4800    1.155  0.24805
titlePROF-AY            46095      1719   26.815 < 2e-16 ***
titlePROF-AY-B/E/E      73586      2283   32.233 < 2e-16 ***
titlePROF-HCOMP         115094     2356   48.855 < 2e-16 ***
locDavis                -19101     2304   -8.291 < 2e-16 ***
locIrvine               -12240      2351   -5.206 2.01e-07 ***
locLos Angeles           7699       2082   3.697  0.00022 ***
locMerced               -20940     4484   -4.669 3.10e-06 ***
locRiverside             -18333     2893   -6.337 2.56e-10 ***
locSan Diego              -11851     2227   -5.322 1.07e-07 ***
locSan Francisco          -15808     3493   -4.525 6.17e-06 ***
locSanta Barbara          -16579     2411   -6.877 6.89e-12 ***
locSanta Cruz              -24973     2930   -8.523 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Residual standard error: 40970 on 4897 degrees of freedom
Multiple R-squared: 0.5058, Adjusted R-squared: 0.504
F-statistic: 294.8 on 17 and 4897 DF, p-value: < 2.2e-16

```
gg_diagnose(lm(pay ~ title + loc, data = dt))
```



To note: Berkeley has been taken as the baseline location. Similarly, ASSOC-PROF AY has been taken as the baseline title.

The Q-Q plot shows that this is a terrible model! Now let's try with the transformed data:

```
scaled <- lm(log2(pay) ~ title + loc, data = dt)
summary(scaled)
```

```
Call:
lm(formula = log2(pay) ~ title + loc, data = dt)

Residuals:
```

	Min	1Q	Median	3Q	Max
-2.23150	-0.22355	0.01801	0.25702	1.24529	

```
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 16.52889 0.02037 811.287 < 2e-16 ***
titleASSOC PROF-AY-B/E/E 0.52397 0.03461 15.141 < 2e-16 ***
titleASSOC PROF-HCOMP 0.34517 0.04023 8.579 < 2e-16 ***
titleASST PROF-AY -0.29772 0.02411 -12.351 < 2e-16 ***
```

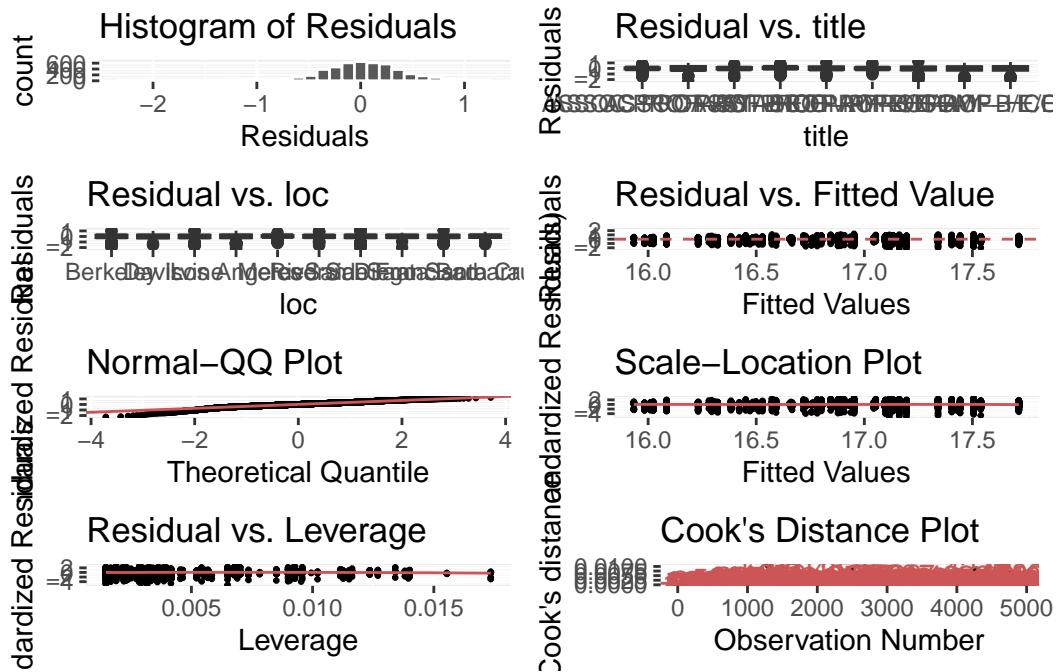
```

titleASST PROF-AY-B/E/E    0.18997   0.04017   4.729 2.32e-06 ***
titleASST PROF-HCOMP       0.06826   0.04883   1.398  0.16220
titlePROF-AY                 0.56942   0.01749  32.562 < 2e-16 ***
titlePROF-AY-B/E/E        0.81217   0.02322  34.971 < 2e-16 ***
titlePROF-HCOMP            1.12262   0.02397  46.841 < 2e-16 ***
locDavis                   -0.20826   0.02344  -8.886 < 2e-16 ***
locIrvine                  -0.14533   0.02392  -6.075 1.33e-09 ***
locLos Angeles              0.06309   0.02118   2.979  0.00291 **
locMerced                  -0.24781   0.04562  -5.432 5.84e-08 ***
locRiverside               -0.22030   0.02943  -7.485 8.43e-14 ***
locSan Diego                -0.14584   0.02266  -6.437 1.33e-10 ***
locSan Francisco            -0.11260   0.03554  -3.168  0.00154 **
locSanta Barbara            -0.20706   0.02453  -8.442 < 2e-16 ***
locSanta Cruz               -0.29716   0.02981  -9.969 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Residual standard error: 0.4168 on 4897 degrees of freedom
 Multiple R-squared: 0.5372, Adjusted R-squared: 0.5356
 F-statistic: 334.3 on 17 and 4897 DF, p-value: < 2.2e-16

```
gg_diagnose(lm(log2(pay) ~ title + loc, data = dt))
```



Much better! Remember to inspect your explanatory and response variables. Ideally, you want the response to be normally distributed. Sometimes one or many covariates can have a nonlinear relationship with the response variable, and you should transform them prior to analysis.

10 ANOVA

```
library(tidyverse) # our friend the tidyverse

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.1.2     v readr      2.1.4
v forcats   1.0.0     v stringr    1.5.0
v ggplot2   3.4.3     v tibble     3.2.1
v lubridate  1.9.2     v tidyverse  1.3.0
v purrr     1.0.2

-- Conflicts -----
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

10.1 Analysis of variance

ANOVA is a method for testing the hypothesis that there is no difference in means of subsets of measurements grouped by factors. Essentially, this is a generalization of linear regression to categorical explanatory variables instead of numeric variables, and it is based on very similar assumptions.

ANOVA perform at its best when we have a particular experimental design: a) we divide the population into groups of equal size (**balanced design**); b) we assign “treatments” to the subjects at random (**randomized design**); in case of multiple treatment combinations, we perform an experiment for each combination (**factorial design**); in most cases, we have a “null” treatment (e.g., placebo).

We speak of **one-way ANOVA** when there is a single axis of variation to our treatment (e.g., no intervention, option A, option B), **two-way ANOVA** when we apply two treatments for each group (e.g., no treatment, Ab, AB, aB), and so forth. Extensions include ANCOVA (ANalysis of COVAriance) and MANOVA (Multivariate ANalysis Of VAriance).

For example, we want to test whether a COVID vaccine protects against infection. We can assign at random a population of volunteers to two classes (vaccine/placebo) and contrast the number of people who got sick within 3 months from treatment in the two classes. Of course,

we can simply perform a t -test. But what if we assign people to different classes (e.g., M/F, under/over 65 y/o), and want to contrast the mean infection rate across all classes?

10.1.1 ANOVA assumptions

ANOVA tests whether samples are taken from distributions with the same mean:

- Null hypothesis: the means of the different groups are the same.
- Alternative hypothesis: **At least one sample mean** is not equal to the others.

Let Y indicate the response variable, and study the simplest case of one-way ANOVA. We have divided the samples in k classes of size J_1, \dots, J_k such that $n = \sum_i J_i$. We write an equation for Y_{ij} (the j th observation in group i):

$$Y_{ij} = \mu + \alpha_i + \epsilon_{ij}$$

where μ is the **overall mean**; $\mu + \alpha_i$ is the mean of group i —we can always choose the parameters such that $\sum_i \alpha_i = 0$; and, finally, ϵ_{ij} is the deviation from the group mean. Practically, we are testing whether at least one of the $\alpha_i \neq 0$.

Note that we are making the same assumptions as for linear regression:

- The observations are obtained independently (and randomly) from the population defined by the factor levels (groups)
- The measurements for each factor level are independent and normally distributed
- These normal distributions have the same variance

10.1.2 How one-way ANOVA works

We have k groups, and define the overall mean $\bar{y} = \sum_i \sum_j Y_{ij}/J_i$, where J_i is the sample size for group i . Then the **total sum of squared deviations** (SSD) is simply:

$$SSD = \sum_{i=1}^k \sum_{j=1}^{J_i} (Y_{ij} - \bar{y})^2$$

and the associated degrees of freedom $n - 1$. We can rewrite this as:

$$SSD = \sum_{i=1}^k J_i (\bar{y}_i - \bar{y})^2 + \sum_{j=1}^{J_i} (Y_{ij} - \bar{y}_i)^2$$

where now \bar{y}_i is the mean for the samples in treatment (factor, group) i . We call the first term in the r.h.s. the **between treatment sum of squares** (SST) and the second term the **within treatment (or residual) ssq** (SSE). As such $SSD = SST + SSE$. Similarly, we can decompose SSD as:

$$SSD = \sum_{j=1}^{J_i} Y_{ij}^2 - n\bar{y}^2 = TSS - SSA$$

where now TSS is the **total sum of squares** and SSA is the **sum of squares due to the average**. Combining the two equations, we can rewrite TSS as the sum of three components:

$$TSS = SSA + SST + SSE$$

Note that the degrees of freedom associated with each term are 1 , $k - 1$ and $n - k$, respectively. What remains to be proved is how to conduct inference.

10.2 Inference in one-way ANOVA

If the null hypothesis were true, then we would expect the between-treatment “variance” SST, divided by the degrees of freedom ($k - 1$) to be the same as the within-treatment “variance” divided by $n - k$.

Let's look at this hypothesis more closely. If the null hypothesis were true, then SST would be the sum of the squares of independent, normally distributed random variables with mean zero and variance σ^2 . If you remember, the distribution of:

$$Q = \sum_{i=1}^d Z_i^2 \sim \chi^2(d)$$

is called the χ^2 distribution with d degrees of freedom. Then, under the null hypothesis, $SST \sim \chi^2(k - 1)$ Similarly, $SSE \sim \chi^2(n - k)$. Taking the ratio (having normalized using the degrees of freedom), we obtain:

$$\frac{SST}{k - 1} \frac{n - k}{SSE} = \frac{MST}{MSE} \sim F(k - 1, n - k)$$

where F is the **F-distribution** (in R, you can sample from this distribution using `df(x, deg1, deg2)`).

10.2.1 Example of comparing diets

For example, the following data contains measurements of weights of individuals before starting a diet, after 6 weeks of dieting, the type of diet (1, 2, 3), and other variables.

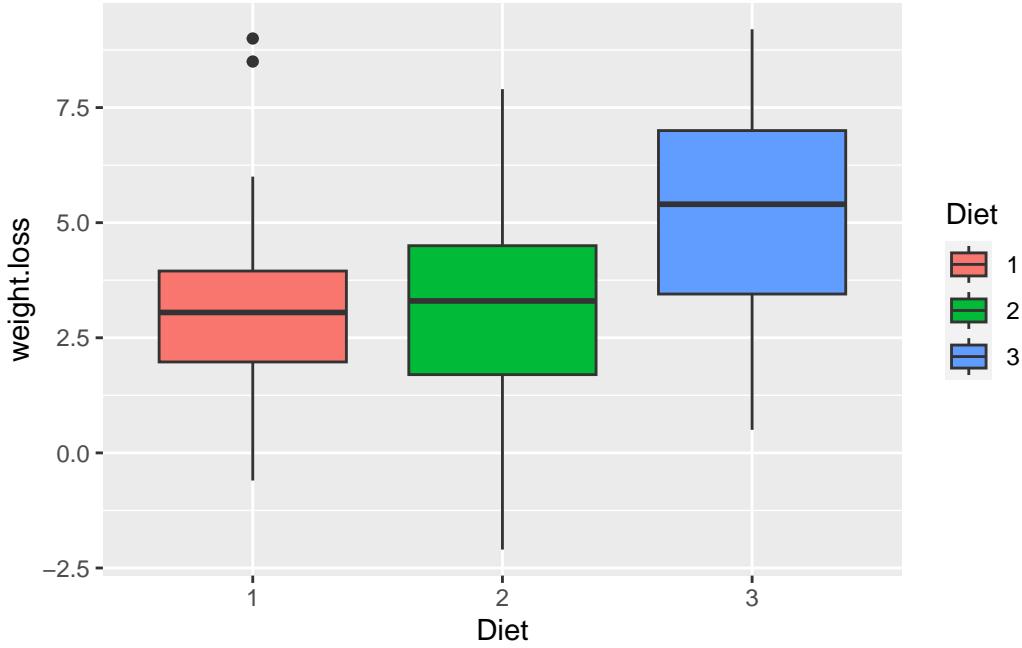
```
library(tidyverse)
# Original URL: "https://www.sheffield.ac.uk/polopoly_fs/1.570199!/file/stcp-Rdataset-Diet
diet <- read_csv("data/Diet.csv")
diet <- diet %>% mutate(weight.loss = pre.weight - weight6weeks)
glimpse(diet)
```

```
Rows: 78
Columns: 8
$ Person      <dbl> 25, 26, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 27-
$ gender       <dbl> NA, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0-
$ Age          <dbl> 41, 32, 22, 46, 55, 33, 50, 50, 37, 28, 28, 45, 60, 48, 4-
$ Height        <dbl> 171, 174, 159, 192, 170, 171, 170, 201, 174, 176, 165, 16-
$ pre.weight   <dbl> 60, 103, 58, 60, 64, 64, 65, 66, 67, 69, 70, 70, 72, 72, ~
$ Diet          <dbl> 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, ~
$ weight6weeks <dbl> 60.0, 103.0, 54.2, 54.0, 63.3, 61.1, 62.2, 64.0, 65.0, 60-
$ weight.loss   <dbl> 0.0, 0.0, 3.8, 6.0, 0.7, 2.9, 2.8, 2.0, 2.0, 8.5, 1.9, 3.~
```

```
# make diet into factors
diet <- diet %>% mutate(Diet = factor(Diet))
```

Write a script below using ggplot to generate boxplots for the weights after three different diets.

```
diet %>% ggplot() +
  aes(y = weight.loss,
      x = Diet,
      fill = Diet) +
  geom_boxplot()
```



We can see that there weight loss outcomes vary for each diet, but diet 3 seems to produce a larger effect on average. But is the difference between the means/medians actually due to the diet, or could it have been produced by sampling from the same distribution, given that we see substantial variation within each diet group?

Here is the result of running ANOVA on the given data set:

```
diet_anova <- aov(weight.loss ~ Diet, data=diet) # note that this looks like lm!
summary(diet_anova)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Diet	2	71.1	35.55	6.197	0.00323 **
Residuals	75	430.2	5.74		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```
print(diet_anova)
```

Call:
`aov(formula = weight.loss ~ Diet, data = diet)`

Terms:

```

      Diet Residuals
Sum of Squares    71.0937  430.1793
Deg. of Freedom       2          75

Residual standard error: 2.394937
Estimated effects may be unbalanced

```

10.2.2 Comparison of theory and ANOVA output

Let's compare this with the calculations from the data set:

```

# 1) compute the overall mean
bar_y <- diet %>%
  summarise(bar_y = mean(weight.loss)) %>%
  as.numeric()

# 2) compute means by diet and sample size by diet
bar_y_i <- diet %>%
  group_by(Diet) %>%
  summarise(bar_y_i = mean(weight.loss),
            J_i = n())
#(NB: almost balanced!)

# 3) compute degrees of freedom
n <- nrow(diet)
k <- nrow(diet) %>% dplyr::select(Diet) %>% distinct()
deg_freedom <- c(1, k - 1, n - k)

# 4) compute SSA, SST and SSE
SSA <- n * bar_y^2
SST <- bar_y_i %>%
  mutate(tmp = J_i * (bar_y_i - bar_y)^2) %>%
  summarise(sst = sum(tmp)) %>%
  as.numeric()
SSE <- diet %>%
  inner_join(bar_y_i, by = "Diet") %>%
  mutate(tmp = (weight.loss - bar_y_i)^2) %>%
  summarise(sse = sum(tmp)) %>%
  as.numeric()

# 5) show that TSS = SSA + SST + SSE
TSS <- diet %>%
  summarise(tss = sum(weight.loss^2)) %>%
  as.numeric()

```

```
print(c(TSS, SSA + SST + SSE))
```

```
[1] 1654.35 1654.35
```

Now that we have all the numbers in place, we can compute our F-statistics, and the associated p-value:

```
Fs <- (SST / (deg_freedom[2])) / (SSE / (deg_freedom[3]))
pval <- 1 - pf(Fs, deg_freedom[2], deg_freedom[3])
```

Contrast these with the output of `aov`:

```
print(deg_freedom[-1])
```

```
[1] 2 75
```

```
print(c(SST, SSE))
```

```
[1] 71.09369 430.17926
```

```
print(c(Fs, pval))
```

```
[1] 6.197447453 0.003229014
```

```
print(summary(aov(weight.loss ~ Diet, data = diet)))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Diet	2	71.1	35.55	6.197	0.00323 **
Residuals	75	430.2	5.74		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

At first glance, this process is not the same as fitting parameters for linear regression, but it is based on exactly the same assumptions: additive noise and additive effect of the factors, with the only difference being that factors are not numeric, so the effect of each one is added separately. One can run linear regression and calculate coefficients that are identical to the mean and the differences between means computed by ANOVA (and note the p-values too!)

```

diet.lm <- lm(weight.loss ~ Diet, data=diet)
summary(diet.lm)

Call:
lm(formula = weight.loss ~ Diet, data = diet)

Residuals:
    Min      1Q  Median      3Q     Max 
-5.1259 -1.3815  0.1759  1.6519  5.7000 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  3.3000    0.4889   6.750 2.72e-09 ***  
Diet2        -0.2741    0.6719  -0.408  0.68449    
Diet3         1.8481    0.6719   2.751  0.00745 **  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.395 on 75 degrees of freedom
Multiple R-squared:  0.1418,    Adjusted R-squared:  0.1189 
F-statistic: 6.197 on 2 and 75 DF,  p-value: 0.003229

```

```

print(diet.lm$coefficients)

(Intercept)      Diet2      Diet3
3.3000000 -0.2740741  1.8481481

```

10.3 Further steps

10.3.1 Post-hoc analysis

The ANOVA F-test tells us whether there is any difference in values of the response variable between the groups, but does not specify which group(s) are different. For this, a *post-hoc* test is used (Tukey's "Honest Significant Difference"):

```

tuk <- TukeyHSD(diet_anova)
tuk

```

```

Tukey multiple comparisons of means
 95% family-wise confidence level

Fit: aov(formula = weight.loss ~ Diet, data = diet)

$Diet
      diff      lwr      upr     p adj
2-1 -0.2740741 -1.8806155 1.332467 0.9124737
3-1  1.8481481  0.2416067 3.454690 0.0201413
3-2  2.1222222  0.5636481 3.680796 0.0047819

```

This compares the three pairs of groups and reports the p-value for the hypothesis that this particular pair has no difference in the response variable.

10.3.2 Example of plant growth data

Example taken from: [One-Way ANOVA Test in R](#)

```

my_data <- PlantGrowth # import built-in data
head(my_data)

  weight group
1   4.17  ctrl
2   5.58  ctrl
3   5.18  ctrl
4   6.11  ctrl
5   4.50  ctrl
6   4.61  ctrl

# Show the levels
my_data %>% dplyr::select(group) %>% distinct()

  group
1  ctrl
2  trt1
3  trt2

group_by(my_data, group) %>%
  summarise(

```

```

    count = n(),
    mean = mean(weight, na.rm = TRUE),
    sd = sd(weight, na.rm = TRUE)
  )

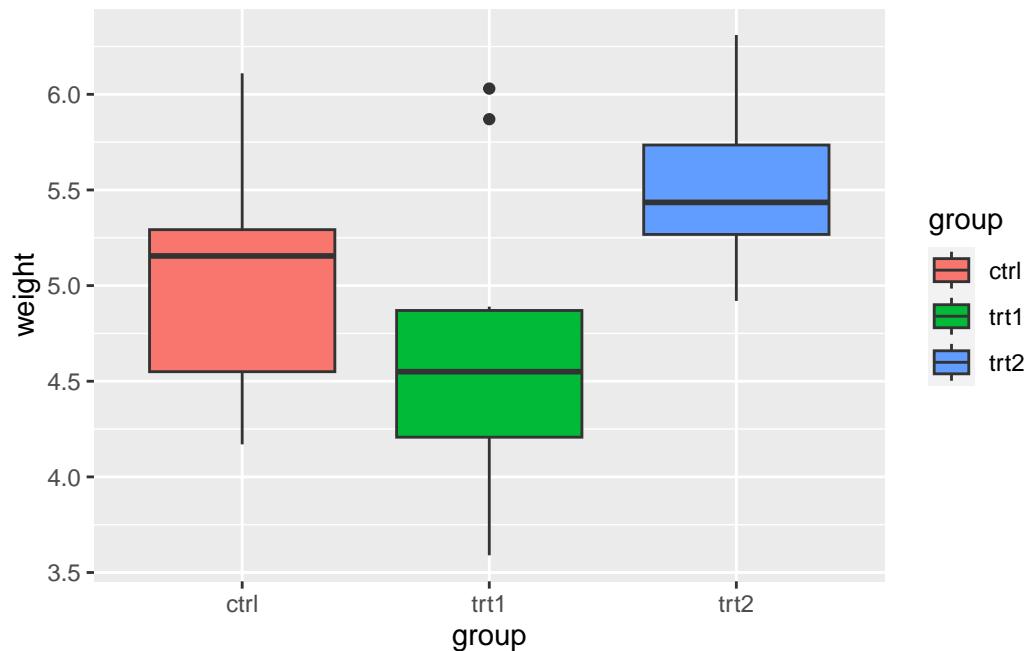
# A tibble: 3 x 4
  group count  mean    sd
  <fct> <int> <dbl> <dbl>
1 ctrl     10  5.03 0.583
2 trt1     10  4.66 0.794
3 trt2     10  5.53 0.443

```

```

my_data %>%
  ggplot() +
  aes(y = weight, x = group,
      fill = group) +
  geom_boxplot()

```



Exercise: perform ANOVA and Tukey's HSD and interpret the results.

10.3.3 Two-way ANOVA

One can compare the effect of two different factors simultaneously and see if considering both explains more of the variance than of one. This is equivalent to the multiple linear regression with two interacting variables. How would you interpret these results?

```
diet.fisher <- aov(weight.loss ~ Diet * gender, data = diet)
summary(diet.fisher)

Df Sum Sq Mean Sq F value    Pr(>F)
Diet        2   60.5   30.264   5.629 0.00541 ***
gender      1     0.2     0.169   0.031 0.85991
Diet:gender 2   33.9   16.952   3.153 0.04884 *
Residuals   70  376.3    5.376
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
2 observations deleted due to missingness
```

10.4 Investigate the UC salaries dataset

```
# read the data
# Original URL
dt <- read_csv("https://raw.githubusercontent.com/dailybruin/uc-salaries/master/data/uc_sa
col_names = c("first_name", "last_name", "title", "a", "pay", "loc", "year", "b", "c", "d"

Rows: 175000 Columns: 10
-- Column specification -----
Delimiter: ","
chr (4): first_name, last_name, title, loc
dbl (6): a, pay, year, b, c, d

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

# get only profs
dt <- dt %>% filter(title %in% c("PROF-AY", "ASSOC PROF-AY", "ASST PROF-AY",
                                    "PROF-AY-B/E/E", "PROF-HCOMP", "ASST PROF-AY-B/E/E",
                                    "ASSOC PROF-AY-B/E/E", "ASSOC PROF-HCOMP", "ASST PROF-HC")
```

```

# simplify titles
dt <- dt %>% mutate(title = ifelse(grepl("ASST", title), "Assistant", title))
dt <- dt %>% mutate(title = ifelse(grepl("ASSOC", title), "Associate", title))
dt <- dt %>% mutate(title = ifelse(grepl("PROF", title), "Full", title))
# remove those making less than 50k (probably there only for a period)
dt <- dt %>% filter(pay > 50000)
glimpse(dt)

```

Rows: 4,795

Columns:

\$ first_name	<chr> "CHRISTOPHER U", "HENRY DON ISAAC", "ADAM R", "KEVORK N.", ~
\$ last_name	<chr> "ABANI", "ABARBANEL", "ABATE", "ABAZAJIAN", "ABBAS", "ABBAS~
\$ title	<chr> "Full", "Full", "Assistant", "Assistant", "Full", "Full", "~
\$ loc	<chr> "Riverside", "San Diego", "San Francisco", "Irvine", "Irvin~
\$ pay	<dbl> 151200.00, 160450.08, 85305.01, 82400.04, 168699.96, 286823~

1. Plot the distributions of pay by location and title. Is it approximately normal? If not, transform the data.
2. Run ANOVA for pay as dependent on the two factors separately, report the variance between means and the variance within groups, and the p-value for the null hypothesis.
3. Run Tukey's test for multiple comparison of means to report which group(s) are substantially different from the rest, if any.
4. Run a two-way ANOVA for both location and title and provide interpretation.

10.4.1 A word of caution about unbalanced designs

When we have a different number of samples in each category, we might encounter some problems, as the order in which we enter the terms might matter: for example, run `aov(pay ~ title + loc)` vs. `aov(pay ~ loc + title)`, and see that the sum of squares for the two models differ. In some cases, this might lead to the puzzling results—depending on how we enter the model, we might determine that a treatment has an effect or not. Turns out, there are three different ways to account for the sum-of-squares in ANOVA, all testing slightly different hypotheses. For balanced designs, they all return the same answer, but if you have different sizes, please read [here](#).

11 Model Selection

Cchiù longa è a pinsata cchiù grossa è a minchiata

[the longer the thought, the bigger the bullshit]

— Sicilian proverb

11.1 Goal

For any data you might want to fit, several competing statistical models seem to do a fairly good job. But which model should you use then?

The goal of model selection is to provide you with a disciplined way to choose among competing models. While there is no consensus on a single technique to perform model selection (we will examine some of the alternative paradigms below), all techniques are inspired by Occam’s razor: given models of similar explanatory power, choose the simplest.

But what does “simplest” mean? Measuring a model’s “complexity” is far from trivial, hence the different schools of thought. Some approaches simply count the number of free parameters, and penalize models with more parameters; others take into account how much each parameter should be “fine-tuned” to fit the data; other approaches are based on entirely different premises.

But why should you choose the simplest model? First, simpler models are easier to analyze, so that for example you could make analytical headway into the mechanics of the process you want to model; simpler models are also considered more beautiful. Second, you want to avoid *over-fitting*: each biological data set—however carefully crafted—is noisy, and you want to fit the signal, not the noise. If you include too much flexibility in your model, you will get what looks like an excellent fit for the specific data set, but you will be unable to fit other data sets to which your model should also apply.

```
library(tidyverse) # our friend
library(BayesFactor) # Bayesian model selection
library(tidymodels) # for the parsnip package, along with the rest of tidymodels
library(palmerpenguins)
# Helper packages
```

```
#library(readr)      # for importing data
library(broom.mixed) # for converting bayesian models to tidy tibbles
library(dotwhisker)  # for visualizing regression results
```

11.2 Problems

1. Over-fitting can lead to wrong inference. (The problem is similar to that of spurious correlations).
2. Identifiability of parameters. Sometimes it is hard/impossible to find the best value for a set of parameters. For example, when parameters only appear as sums or products in the model. In general, it is difficult to prove that the set of parameters leading to the maximum likelihood is unique.
3. Finding best estimates. For complex models, it might be difficult to find the best estimates for a set of parameters. For example, several areas of the parameter space could yield a good fit, and the good sets of parameters could be separated by areas with poor fit. Then, we might get “stuck” in a sub-optimal region of the parameters space.

11.3 Approaches based on maximum-likelihoods

We start by examining methods that are based on maximum likelihoods. For each data set and model, you find the best fitting parameters (those maximizing the likelihood). The parameters are said to be at their maximum-likelihood estimate.

11.3.1 Likelihood function

Some notation:

$D \rightarrow$ the observed data

$\theta \rightarrow$ the free parameter(s) of the statistical model

$L(\theta|D) \rightarrow$ the likelihood function, read “the likelihood of θ given the data”

$\hat{\theta} \rightarrow$ the maximum-likelihood estimates (m.l.e.) of the parameters

$\mathcal{L}(\theta|D) = \log L(\theta|D) \rightarrow$ the log-likelihood

$L(\hat{\theta}|D) \rightarrow$ the maximum likelihood

11.3.2 Discrete probability distributions

The simplest case is that of a probability distribution function that takes discrete values. Then, the likelihood of θ given the data is simply the probability of obtaining the data when parameterizing the model with parameters θ :

$$L(\theta|x_j) = P(X = x_j; \theta)$$

Finding the m.l.e. of θ simply means finding the value(s) maximizing the probability of recovering the data under the model.

11.3.3 Continuous probability distributions

The definition is more complex for continuous variables (because $P(X = x; \theta) = 0$ as there are infinitely many values...). What is commonly done is to use the *density function* $f(x; \theta)$ and considering the probability of obtaining a value $x \in [x_j, x_j + h]$, where x_j is our observed data point, and h is small. Then:

$$L(\theta|x_j) = \lim_{h \rightarrow 0^+} \frac{1}{h} \int_{x_j}^{x_j+h} f(x; \theta) dx = f(x_j; \theta)$$

Note that, contrary to probabilities, density values can take values greater than 1. As such, when the dispersion is small, one could end up with values of likelihood greater than 1 (or positive log-likelihoods). In fact, the likelihood function is proportional to but not necessarily equal to the probability of generating the data given the parameters: $L(\theta|X) \propto P(X; \theta)$.

In many cases, maximizing the likelihood is equivalent to minimizing the sum of square errors (residuals).

11.4 Likelihoods for linear regression

As you remember, we have considered the normal equations:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

Where the residuals have variance σ^2 . The likelihood of the parameters is simply the product of the likelihood for each point:

$$L(\beta_0, \beta_1, \sigma^2 | Y) = \prod_i L(\beta_0, \beta_1, \sigma^2 | Y_i) = \prod_i f(Y_i; \beta_0, \beta_1, \sigma^2) = \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(Y_i - (\beta_0 + \beta_1 X_i))^2}{2\sigma^2}\right)$$

We want to choose the parameters such that they maximize the likelihood. Because the logarithm is monotonic then maximizing the likelihood is equivalent to maximizing the log-likelihood:

$$\mathcal{L}(\beta_0, \beta_1, \sigma^2 | Y) = -\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2} \sum_i (Y_i - (\beta_0 + \beta_1 X_i))^2$$

Showing that by minimizing the sum of squares, we are maximizing the likelihood.

11.5 Likelihood-ratio tests

These approaches contrast two models by taking the ratio of the maximum likelihoods of the sample data based on the models (i.e., when you evaluate the likelihood by setting the parameters to their m.l.e.). The two models are usually termed the *null* model (i.e., the “simpler” model), and the *alternative* model. The ratio of L_a/L_n tells us how many times more likely the data are under the alternative model vs. the null model. We want to determine whether this ratio is large enough to reject the null model and favor the alternative.

Likelihood-ratio is especially easy to perform for *nested* models.

11.5.0.1 Two nested models

Nested means that model \mathcal{M}_1 has parameters θ_1 , and model \mathcal{M}_2 has parameters θ_2 , such that $\theta_1 \in \theta_2$ — by setting some of the parameters of \mathcal{M}_2 to particular values, we recover \mathcal{M}_1 .

For example, suppose we want to model the height of trees. We measure the response variable (height of tree i , h_i) as well as the girth (g_i). We actually have a data set that ships with R that contains exactly this type of data:

```
data(trees)
head(trees)
```

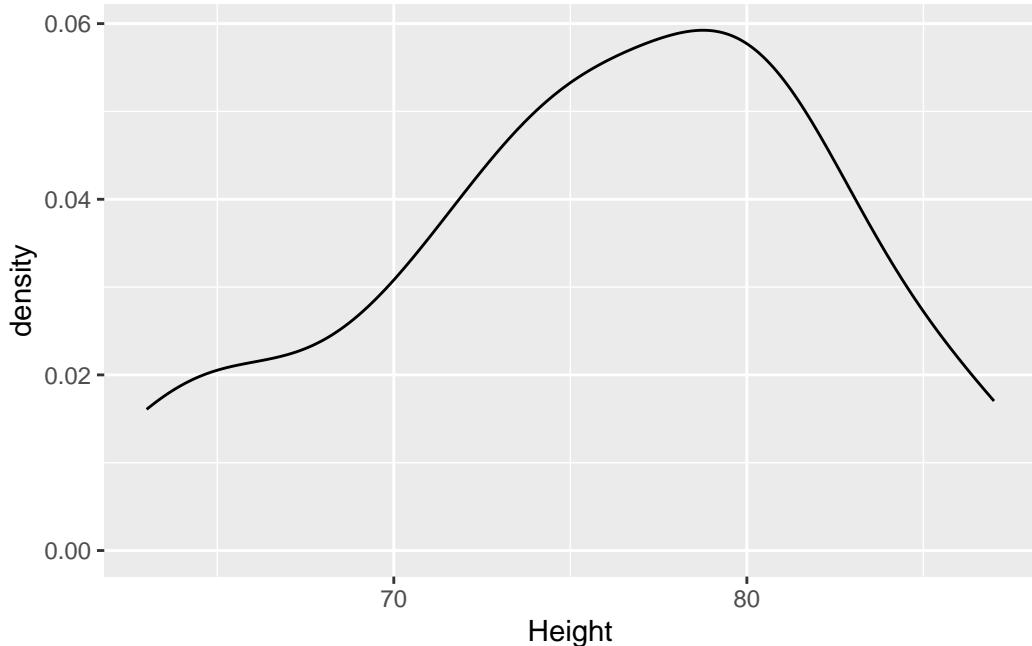
	Girth	Height	Volume
1	8.3	70	10.3
2	8.6	65	10.3
3	8.8	63	10.2
4	10.5	72	16.4
5	10.7	81	18.8
6	10.8	83	19.7

The **Height** of these cherry trees is measured in feet; the **Girth** is the diameter in inches, and the **Volume** is the amount of timber in cubic feet. Let's add a **Radius** measured in feet:

```
trees <- trees %>% mutate (Radius = Girth / (2 * 12)) # diameter to radius; inches to feet
```

Let's look at the distribution of three heights:

```
trees %>% ggplot(aes(x = Height)) + geom_density()
```



A possible simple model is one that says that all tree heights have heights taken from a Gaussian distribution with a given mean. In the context of linear regression, we can write the model \mathcal{M}_0 :

$$h_i = \theta_0 + \epsilon_i$$

where we assume that the errors $\epsilon_i \stackrel{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$. Now fit the model, obtaining $\hat{\theta}_0$, and compute the maximum log-likelihood $\mathcal{L}_0(\hat{\theta}_0, \hat{\sigma}^2 | h)$.

In R, we would call:

```

M0 <- lm(data = trees, Height ~ 1) # only intercept
# the m.l.e. of theta_0
theta0_M0 <- M0$coefficients[1]
theta0_M0

(Intercept)
76

# log likelihood
logLik(M0)

'log Lik.' -100.8873 (df=2)

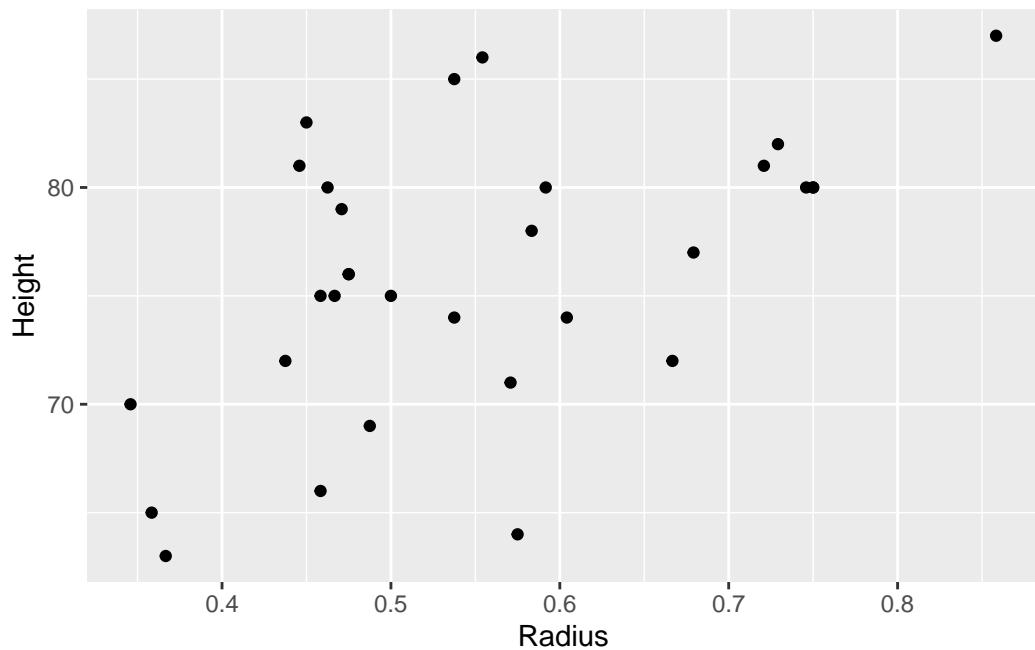
```

Now let's plot the height of the trees vs. their radius:

```

trees %>% ggplot(aes(x = Radius, y = Height)) +
  geom_point()

```



And compute their correlation:

```
cor(trees$Radius, trees$Height)
```

```
[1] 0.5192801
```

Given the positive correlation between radius and height, we can build a more complex model in which the height also depends on radius (\mathcal{M}_1):

$$h_i = \theta_0 + \theta_1 r_i + \epsilon_i$$

as for model \mathcal{M}_0 , fit the parameters (note that $\hat{\theta}_0$ for model \mathcal{M}_0 will in general be different from $\hat{\theta}_0$ for model \mathcal{M}_1), and compute $\mathcal{L}_1(\hat{\theta}_0, \hat{\theta}_1, \hat{\sigma}^2 | h)$. These two models are nested, because when setting $\theta_1 = 0$ we recover \mathcal{M}_0 .

In R:

```
M1 <- lm(data = trees, Height ~ Radius) # intercept and slope
theta0_M1 <- M1$coefficients[1]
theta1_M1 <- M1$coefficients[2]
# note that now theta_0 takes a different value:
print(c(theta0_M1, theta1_M1))
```

```
(Intercept) (Intercept)
62.03131    62.03131
```

```
# the log likelihood should improve
logLik(M1)
```

```
'log Lik.' -96.01663 (df=3)
```

Which model should we use? You can see that adding an extra parameter improved the likelihood somewhat.

Enter the likelihood-ratio test. We want to know whether it's worth using the more complex model, and to do this we need to calculate a likelihood-ratio statistics. We're helped by *Wilks' theorem*: as the sample size $n \rightarrow \infty$, the test statistics $2 \log(L_1/L_0)$ is asymptotically χ^2 distributed with degrees of freedom equal to the difference in the number of parameters between \mathcal{M}_1 and \mathcal{M}_0 .

While there are many caveats [^1] this method is commonly used in practice.

```

# 2 * log-likelihood ratio
lrt <- as.numeric(2 * (logLik(M1) - logLik(M0)))
print("2 log(L1 / L0)")

[1] "2 log(L1 / L0)"

print(lrt)

[1] 9.74125

# difference in parameters
df0 <- length(M0$coefficients)
df1 <- length(M1$coefficients)
k <- df1 - df0
print("Number of extra parameters")

[1] "Number of extra parameters"

print(k)

[1] 1

# calculate (approximate) p-value
res <- pchisq(lrt, k, lower.tail = FALSE)
print(paste("p-value using Chi^2 with", k, "degrees of freedom"))

[1] "p-value using Chi^2 with 1 degrees of freedom"

print(round(res, 4))

[1] 0.0018

```

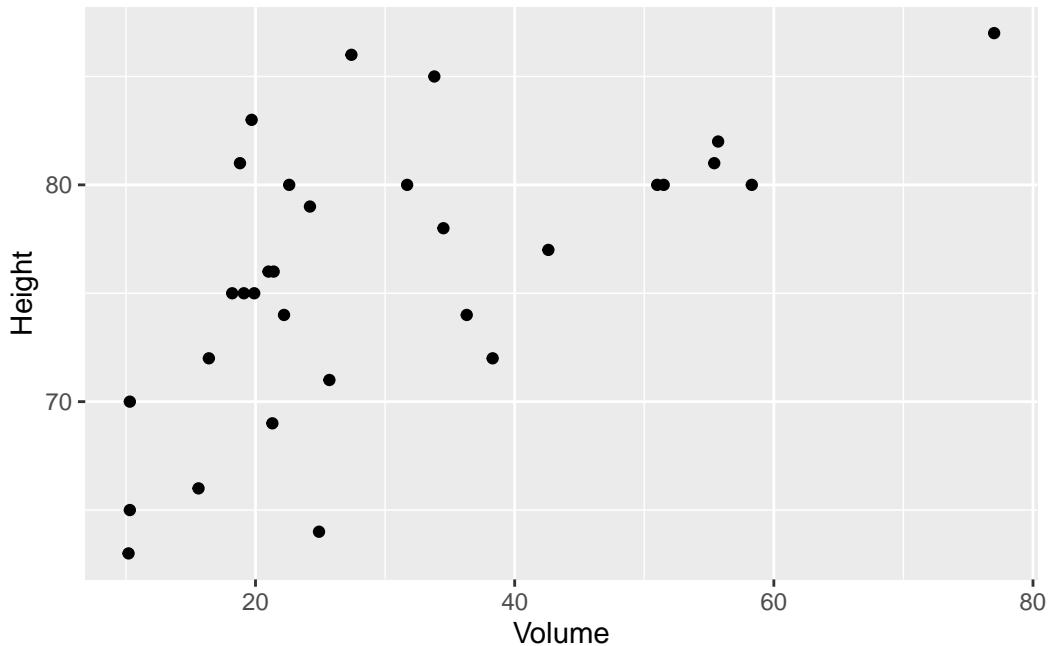
In this case, the likelihood-ratio test would favor the use of the more complex model.

- **Pros:** Straightforward; well-studied for nested models.
- **Cons:** Difficult to generalize to more complex cases.

11.5.0.2 Adding more models

The data also contains a column with the volume. Let's take a look:

```
trees %>% ggplot() + aes(x = Volume, y = Height) + geom_point()
```



And look at the correlation

```
cor(trees$Volume, trees$Height)
```

```
[1] 0.5982497
```

We can build another model:

```
M2 <- lm(data = trees, Height ~ Volume) # intercept and slope
```

Compute the log likelihood:

```
logLik(M2)
```

```
'log Lik.' -94.02052 (df=3)
```

and test whether that's better than the (nested) model 0:

```
# 2 * log-likelihood ratio
lrt <- as.numeric(2 * (logLik(M2) - logLik(M0)))
print("2 log(L2 / L0)")

[1] "2 log(L2 / L0)"

print(lrt)

[1] 13.73348

# difference in parameters
df0 <- length(M0$coefficients)
df1 <- length(M2$coefficients)
k <- df1 - df0
print("Number of extra parameters")

[1] "Number of extra parameters"

print(k)

[1] 1

# calculate (approximate) p-value
res <- pchisq(lrt, k, lower.tail = FALSE)
print(paste("p-value using Chi^2 with", k, "degrees of freedom"))

[1] "p-value using Chi^2 with 1 degrees of freedom"

print(round(res, 4))

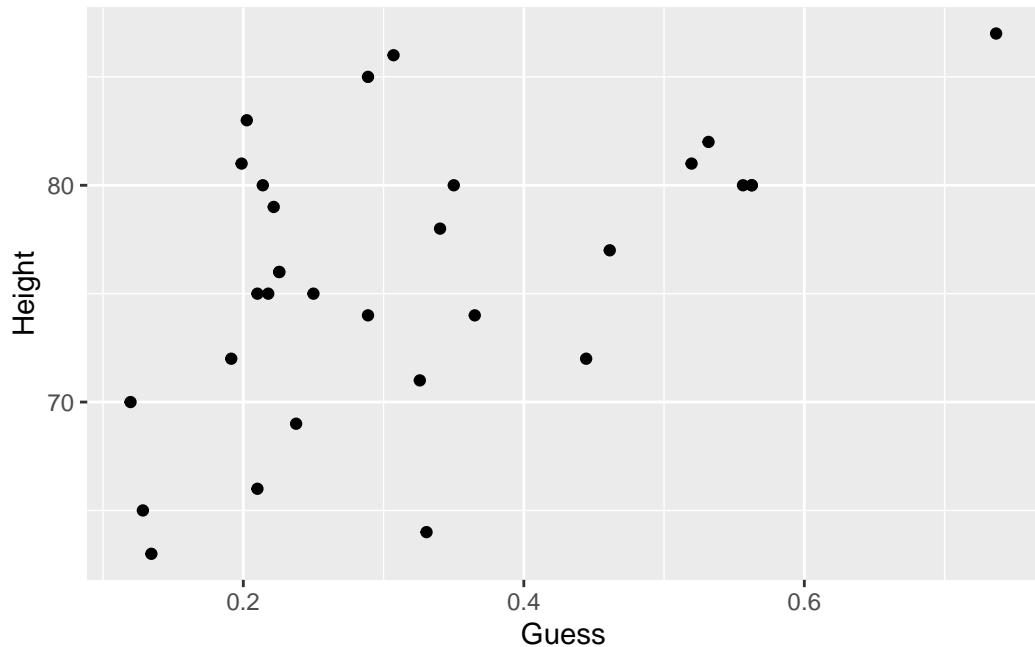
[1] 2e-04
```

Also in this case, the likelihood-ratio test would favor the use of the more complex model. But how can we contrast the two more complex models \mathcal{M}_1 and \mathcal{M}_2 ? They are not nested!

In fact, we can even concoct another model that uses a mix of radius and volume. If we assume that trees are cylinders, then we have $V = \pi r^2 h$, and as such $h = V/(\pi r^2)$. We can test whether this is a good approximation by creating a new variable:

```
trees <- trees %>% mutate(Guess = Radius^2)

trees %>% ggplot() + aes(x = Guess, y = Height) + geom_point()
```



```
cor(trees$Guess, trees$Height)
```

```
[1] 0.5084267
```

Pretty good! Let's add it to our list of models:

```
M3 <- lm(Height ~ Guess, data = trees)
logLik(M3)
```

```
'log Lik.' -96.25156 (df=3)
```

11.6 AIC

Of course, in most cases the models that we want to contrast need not to be nested. Then, we can try to penalize models according to the number of free parameters, such that more complex models (those with many free parameters) should be associated with much better likelihoods to be favored.

In the early 1970s, Hirotugu Akaike proposed “an information criterion” (AIC, now known as Akaike’s Information Criterion), based, as the name implies, on information theory. Basically, AIC is measuring (asymptotically) the information loss when using the model in lieu of the actual data. Philosophically, it is rooted in the idea that there is a “true model” that generated the data, and that several possible models can serve as its approximation. Practically, it is very easy to compute:

$$AIC = -2\mathcal{L}(\theta|D) + 2k$$

where k is the number of free parameters (e.g., 3 for the simplest linear regression [intercept, slope, variance of the residuals]). In R, many models provide a way to access their AIC score:

```
AIC(M0) # only intercept
```

```
[1] 205.7745
```

```
AIC(M1) # use radius
```

```
[1] 198.0333
```

```
AIC(M2) # use volume
```

```
[1] 194.041
```

```
AIC(M3) # use cylinder
```

```
[1] 198.5031
```

You can see that AIC favors the cylinder model over the others. Typically, a difference of about 2 is considered “significant”, though of course this really depends on the size of the data, the values of AIC, etc.

- **Pros:** Easy to calculate; very popular.
- **Cons:** Sometimes it is difficult to “count” parameters; why should each parameter cost the same, when they have different effects on the likelihood?

11.7 Other information-based criteria

The approach spearheaded by Akaike has been followed by a number of researchers, giving rise to many similar criteria for model selection. Without getting too much into the details, here are a few pointers:

- Bayesian Information Criterion $BIC = -2\mathcal{L}(\theta|D) + k \log(n)$ where n is the number of data points. Penalizes parameters more strongly when there are much data.
- Hannan–Quinn information criterion $HQC = -2\mathcal{L}(\theta|D) + k \log(\log(n))$

11.8 Bayesian approaches to model selection

The approaches we’ve examined before are based on “point-estimates”, i.e., only consider the parameters at their maximum likelihood estimate. Bayesian approaches, on the other hand, consider distributions of parameters. As such, parameters that give high likelihoods for a restricted range of values are deemed “more expensive” (because they are “more important” or need to be “fine-tuned”) than those yielding about the same likelihood for a wide range of values.

11.8.1 Marginal likelihoods

A very beautiful approach is based on marginal likelihoods, i.e., likelihoods obtained integrating the parameters out. Unfortunately, the calculation becomes difficult to perform by hand for complex models, but it provides a good approach for simple models. In general, we want to assess the “goodness” of a model. Then, using Bayes’ rule:

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)}$$

Where $P(M|D)$ is the probability of the model given the data; and $P(D)$ is the “probability of the data” (don’t worry, this need not to be calculated), and $P(M)$ is the prior (the probability that we choose the model before seeing the data). $P(D|M)$ is a marginal likelihood: we cannot compute this directly, because the model requires the parameters θ , however, we can write

$$P(D|M) = \int P(D|M, \theta)P(\theta|M)d\theta$$

where $P(D|M, \theta)$ is the likelihood, and $P(\theta|M)$ is a distribution over the parameter values (typically, the priors).

For example, let's compute the marginal likelihood for the case in which we flip a coin $n = a+b$ times, and we obtain a heads and b tails. Call θ the probability of obtaining a head, and suppose that $P(\theta|M)$ is a uniform distribution. Then:

$$P(a, b|M) = \int_0^1 P(a, b|M, \theta) d\theta = \int_0^1 \binom{a+b}{a} \theta^a (1-\theta)^b d\theta = \frac{1}{a+b+1} = \frac{1}{n+1}$$

Interestingly, the marginal likelihood can be interpreted as the expected likelihood when parameters are sampled from the prior.

11.8.2 Bayes factors

Take two models, and assume that initially we have no preference $P(M_1) = P(M_2)$, then:

$$\frac{P(M_1|D)}{P(M_2|D)} = \frac{P(D|M_1)P(M_1)}{P(D|M_2)P(M_2)} = \frac{P(D|M_1)}{P(D|M_2)}$$

The ratio is called the “Bayes factor” and provides a rigorous way to perform model selection.

11.8.3 Bayes factors in practice

In practice, Bayes Factors can be estimated from MCMC. While we're not going to get into this here, we can use a package that a) automatically sets the priors for all the variables (close to the philosophy known as “Objective Bayes”); b) performs the calculation of the Bayes Factors for us.

Let's build very many models. Load the data:

```
data(trees)
head(trees)
```

	Girth	Height	Volume
1	8.3	70	10.3
2	8.6	65	10.3
3	8.8	63	10.2
4	10.5	72	16.4
5	10.7	81	18.8
6	10.8	83	19.7

```
trees$Radius <- trees$Girth / (2 * 12)
trees$Guess <- trees$Volume / trees$Radius^2
```

And build the models:

```
lm_all <- lm(Height ~ ., data = trees) # . means use all cols besides Height
summary(lm_all)
```

Call:

```
lm(formula = Height ~ ., data = trees)
```

Residuals:

Min	1Q	Median	3Q	Max
-6.7669	-2.4752	-0.2354	1.9335	10.5319

Coefficients: (1 not defined because of singularities)

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	22.6671	16.2947	1.391	0.175562
Girth	1.5127	1.2278	1.232	0.228543
Volume	-0.2045	0.2572	-0.795	0.433505
Radius	NA	NA	NA	NA
Guess	0.4291	0.1034	4.152	0.000296 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.023 on 27 degrees of freedom

Multiple R-squared: 0.6413, Adjusted R-squared: 0.6014

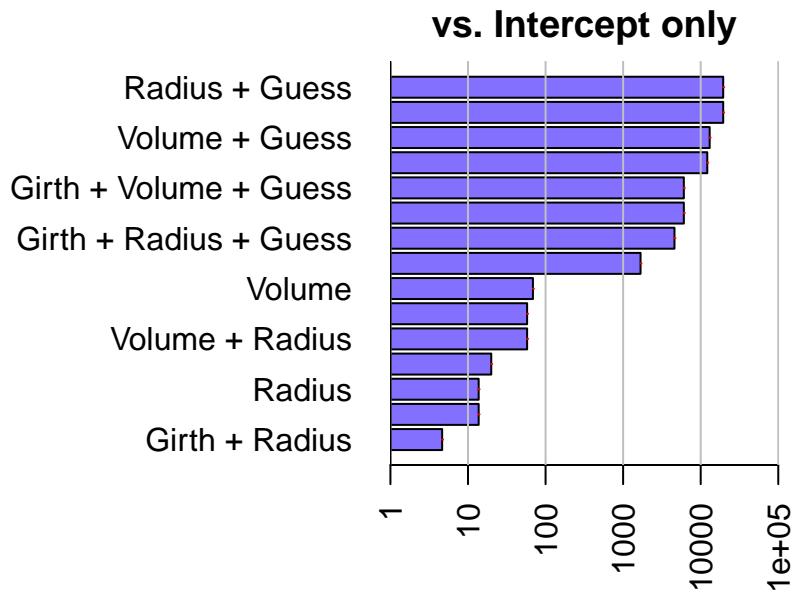
F-statistic: 16.09 on 3 and 27 DF, p-value: 3.391e-06

```
logLik(lm_all)
```

```
'log Lik.' -84.99667 (df=5)
```

Perform selection among all models nested into lm_all:

```
bf_analysis <- regressionBF(Height ~ ., data = trees)
plot(bf_analysis)
```



These ratios measure how many times more probable the model is compared to that with only the intercept (assuming initially that all models are equiprobable). Note that the Bayes Factors automatically penalize for overly complex models (triplets/quadruplets are ranked after pairs or even only `Guess`).

- **Pros:** Elegant, straightforward interpretation.
- **Cons:** Difficult to compute for complex models; requires priors.

11.9 Using `tidymodels` for modeling and cross-validation

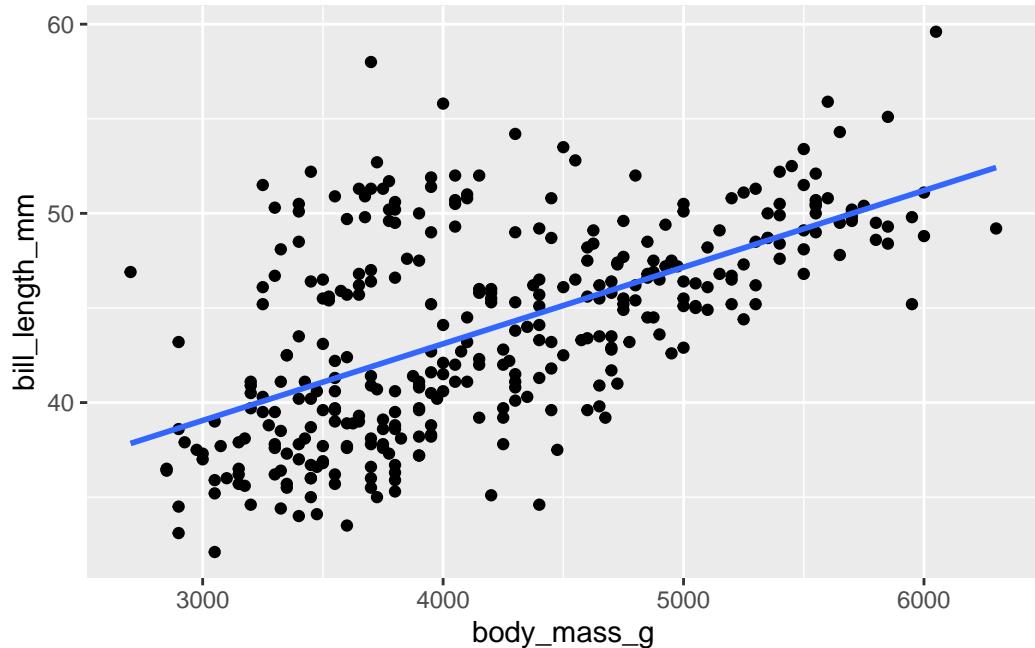
There is an excellent suite of packages called `tidymodels` that offers very beautiful and streamlined tools for building models, training them, and evaluating their results. We will use the Palmer penguins data as an application, with the aim of building a predictive model for the bill length of penguins. Let us first examine the data graphically to see the relationship between body mass and bill length:

```
library(palmerpenguins)
data("penguins")
penguins %>% ggplot(aes( x= body_mass_g, y= bill_length_mm)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  scale_color_viridis_d(option = "plasma", end = .7)
```

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

Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).

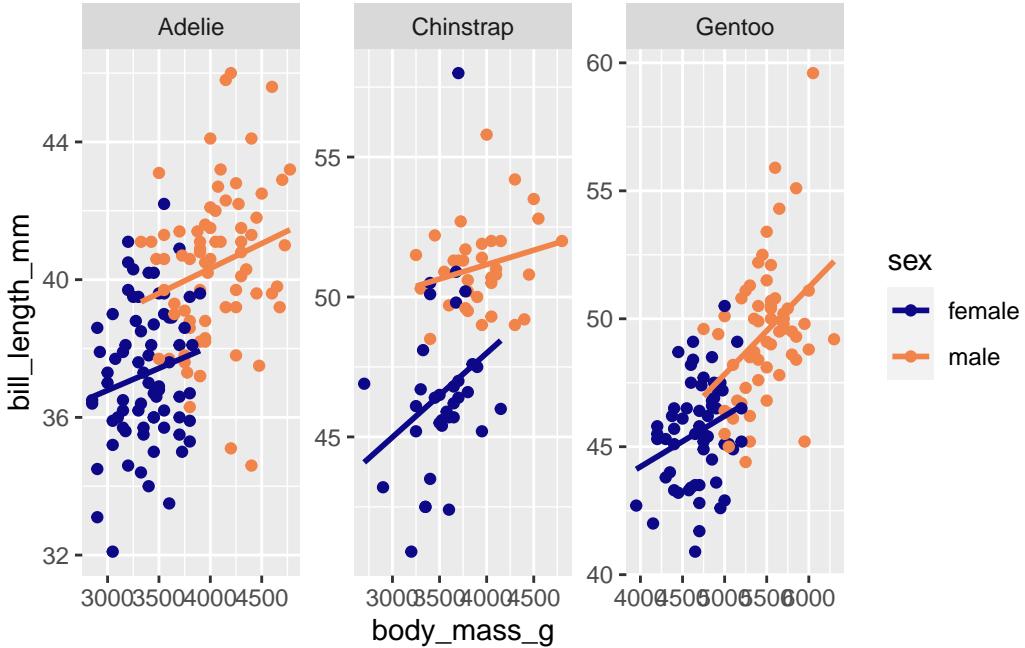
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



There is clearly a relationship, but would it help to add species and sex as variables? Let us see:

```
penguins %>% filter (!is.na(sex)) %>% ggplot(aes( x= body_mass_g, y= bill_length_mm, color = sex)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  facet_wrap(~species, scales = 'free') +
  scale_color_viridis_d(option = "plasma", end = .7)

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



It certainly appears that including sex and species will result in a better fit. Let us try to compare the models we build using the syntax of `tidymodels`.

First, we need to create the model that we will use in the pipeline. This is created like this:

```
lm_mod <-
  linear_reg() %>%
  set_engine("lm")
```

Next, we clean the data and split it into the training and testing sets, and create a *recipe* that specifies the data set and the response variable that we want to model. The other variables are left as the predictors, but we can take them out of consideration by *changing the role* of those variables to “ID”. In this recipe, the only predictor (explanatory) variable in the data set is `body_mass_g`.

```
data("penguins")
pen_clean <- penguins %>% filter(!is.na(bill_length_mm), !is.na(sex), !is.na(species))
# Fix the random numbers by setting the seed for reproducibility
set.seed(314)
# Put 3/4 of the data into the training set
data_split <- initial_split(pen_clean, prop = 3/4)

# Create data frames for the two sets:
```

```

train_data <- training(data_split)
test_data  <- testing(data_split)

pen_recipe <-
  recipe(bill_length_mm ~ ., data = train_data) %>%
  #update_role(sex, island, year, species, bill_depth_mm, flipper_length_mm, new_role = "I")
  update_role(sex, island, species, bill_depth_mm, flipper_length_mm, new_role = "ID")

```

We can now combine the recipe for the data and the model to create a *workflow* for training the data with a model, and then use it to create a fit:

```

# create workflow
pen_wflow <-
  workflow() %>%
  add_model(lm_mod) %>%
  add_recipe(pen_recipe)
# fit the model to the data
pen_fit <-
  pen_wflow %>%
  fit(data = train_data)

```

Finally, we can extract all sorts of information, such as best-fit parameters, errors, p-values, and likelihoods generated by the fit:

```

fit1 <- pen_fit %>%
  extract_fit_parsnip()
tidy(fit1)

# A tibble: 3 x 5
  term      estimate std.error statistic p.value
  <chr>     <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept) -259.       698.      -0.371 7.11e- 1
2 body_mass_g   0.00406   0.000352   11.5   6.35e-25
3 year         0.142      0.347      0.410 6.83e- 1

glance(fit1) #

# A tibble: 1 x 12
r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
<dbl>           <dbl> <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>
```

```

1      0.352          0.346  4.43          66.7 7.27e-24       2   -723. 1453. 1467.
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>

```

The `tidy` and `glance` functions return different summaries of information; the first one information about fitted parameters, the second the R-squared and likelihood of the model.

Let us now modify the recipe to include the species and save the fitting results to a different object `fit2`:

```

pen_recipe2 <-
  recipe(bill_length_mm ~ ., data = train_data) %>%
  #update_role(sex, island, year, bill_depth_mm, flipper_length_mm, new_role = "ID")
  update_role(sex, island, bill_depth_mm, flipper_length_mm, new_role = "ID")

# create workflow
pen_wflow2 <-
  workflow() %>%
  add_model(lm_mod) %>%
  add_recipe(pen_recipe2)
# fit the model to the data
pen_fit2 <-
  pen_wflow2 %>%
  fit(data = train_data)
# summarise the fit
fit2 <- pen_fit2 %>%
  extract_fit_parsnip()
tidy(fit2)

# A tibble: 5 x 5
  term            estimate std.error statistic p.value
  <chr>           <dbl>     <dbl>      <dbl>    <dbl>
1 (Intercept)     -650.      378.       -1.72 8.67e- 2
2 speciesChinstrap  10.0      0.410      24.4  1.51e-67
3 speciesGentoo    3.47      0.569      6.10  4.24e- 9
4 body_mass_g      0.00374   0.000330    11.3  3.53e-24
5 year             0.336      0.188      1.79  7.53e- 2

glance(fit2)

# A tibble: 1 x 12
r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC

```

```

<dbl>      <dbl> <dbl>      <dbl> <dbl> <dbl> <dbl>
1    0.812      0.809  2.40      264. 2.64e-87     4   -568. 1149.
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>

```

The R-squared as well as the log likelihood have improved substantially and the AIC is lower.

Now let us see if we can further improve the model quality by incorporating sex as another explanatory variable:

```

pen_recipe3 <-
  recipe(bill_length_mm ~ ., data = train_data) %>%
  #update_role(island, year, bill_depth_mm, flipper_length_mm, new_role = "ID")
  update_role(island, bill_depth_mm, flipper_length_mm, new_role = "ID")

# create workflow
pen_wflow3 <-
  workflow() %>%
  add_model(lm_mod) %>%
  add_recipe(pen_recipe3)
# fit the model to the data
pen_fit3 <-
  pen_wflow3 %>%
  fit(data = train_data)
# summarise the fit
fit3 <- pen_fit3 %>%
  extract_fit_parsnip()
tidy(fit3)

# A tibble: 6 x 5
  term            estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>      <dbl>    <dbl>
1 (Intercept)    -700.      349.       -2.00  4.63e- 2
2 speciesChinstrap  10.1      0.379      26.5   1.11e-73
3 speciesGentoo    6.26      0.678      9.23  1.36e-17
4 body_mass_g     0.00177    0.000429    4.13  4.93e- 5
5 sexmale         2.59      0.398      6.52  3.96e-10
6 year            0.364     0.174      2.09  3.75e- 2

glance(fit3)

```

```

# A tibble: 1 x 12
  r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
      <dbl>         <dbl>     <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl>
1     0.840        0.837   2.21     255. 1.47e-94     5  -548. 1110. 1135.
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>

```

Adding sex further improves the R-squared and log-likelihood, as the AIC drops again.

11.9.1 Prediction and cross-validation

Now let us the three trained models to predict the values of bill length in the test data that we set aside:

```

bill_fit1 <- predict(fit1, test_data)
bill_fit2 <- predict(fit2, test_data)
bill_fit3 <- predict(fit3, test_data)

prediction1 <- augment(fit1, test_data)
glimpse(prediction1)

```

```

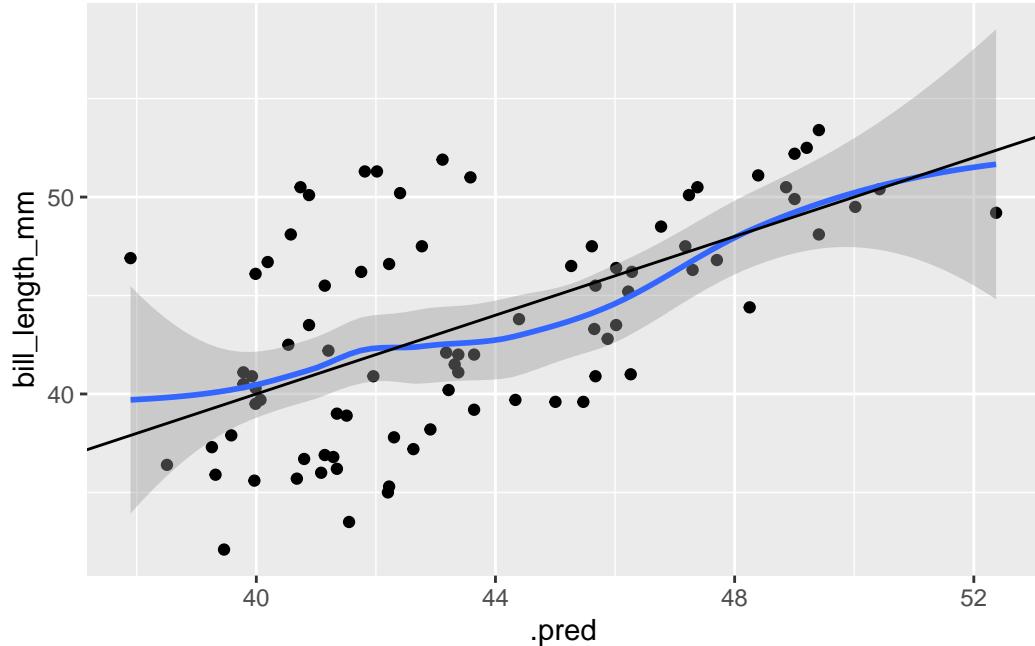
Rows: 84
Columns: 10
$ .pred           <dbl> 39.98780, 40.79993, 41.51054, 39.78476, 42.22115, 39-
$ .resid          <dbl> 0.3122029, -4.0999256, -2.6105380, 1.3152350, -6.921-
$ species        <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel-
$ island          <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgersen, Biscoe, ~
$ bill_length_mm <dbl> 40.3, 36.7, 38.9, 41.1, 35.3, 40.5, 37.9, 39.5, 37.2~
$ bill_depth_mm  <dbl> 18.0, 19.3, 17.8, 17.6, 18.9, 17.9, 18.6, 16.7, 18.1~
$ flipper_length_mm <int> 195, 193, 181, 182, 187, 187, 172, 178, 178, 196, 18-
$ body_mass_g    <int> 3250, 3450, 3625, 3200, 3800, 3200, 3150, 3250, 3900-
$ sex             <fct> female, female, female, female, female, female, fema-
$ year            <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007-

```

```

ggplot(prediction1, aes(x=.pred, y=bill_length_mm)) + geom_point() + geom_smooth() + geom_
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

```



```
metrics(prediction1, truth = bill_length_mm, estimate = .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>        <dbl>
1 rmse    standard     4.42
2 rsq     standard     0.329
3 mae     standard     3.57
```

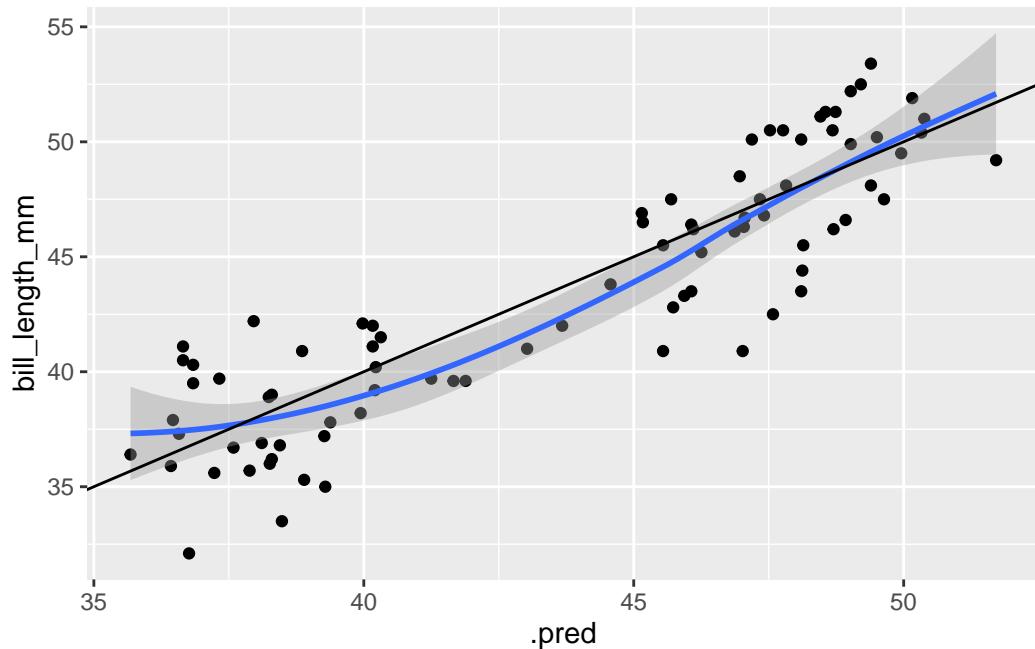
```
bill_fit2 <- predict(fit2, test_data)

prediction2<- augment(fit2, test_data)
glimpse(prediction2)
```

```
Rows: 84
Columns: 10
$ .pred           <dbl> 36.83869, 37.58639, 38.24064, 36.65176, 38.89489, 36~
$ .resid          <dbl> 3.4613143, -0.8863947, 0.6593600, 4.4482415, -3.5948~-
$ species         <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adel~-
$ island          <fct> Torgersen, Torgersen, Torgersen, Torgersen, Biscoe, ~
$ bill_length_mm  <dbl> 40.3, 36.7, 38.9, 41.1, 35.3, 40.5, 37.9, 39.5, 37.2~
```

```
$ bill_depth_mm      <dbl> 18.0, 19.3, 17.8, 17.6, 18.9, 17.9, 18.6, 16.7, 18.1~  
$ flipper_length_mm <int> 195, 193, 181, 182, 187, 187, 172, 178, 178, 196, 18~  
$ body_mass_g       <int> 3250, 3450, 3625, 3200, 3800, 3200, 3150, 3250, 3900~  
$ sex              <fct> female, female, female, female, female, female, fema~  
$ year             <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

```
ggplot(prediction2, aes(x=.pred, y=bill_length_mm)) + geom_point() + geom_smooth() + geom_~`geom_smooth()`~ using method = 'loess' and formula = 'y ~ x'
```



```
metrics(prediction2, truth = bill_length_mm, estimate = .pred)
```

```
# A tibble: 3 x 3  
.metric .estimator .estimate  
<chr>   <chr>        <dbl>  
1 rmse    standard     2.46  
2 rsq     standard     0.795  
3 mae     standard     2.05
```

```

bill_fit3 <- predict(fit3, test_data)

prediction3 <- augment(fit3, test_data)
glimpse(prediction3)

```

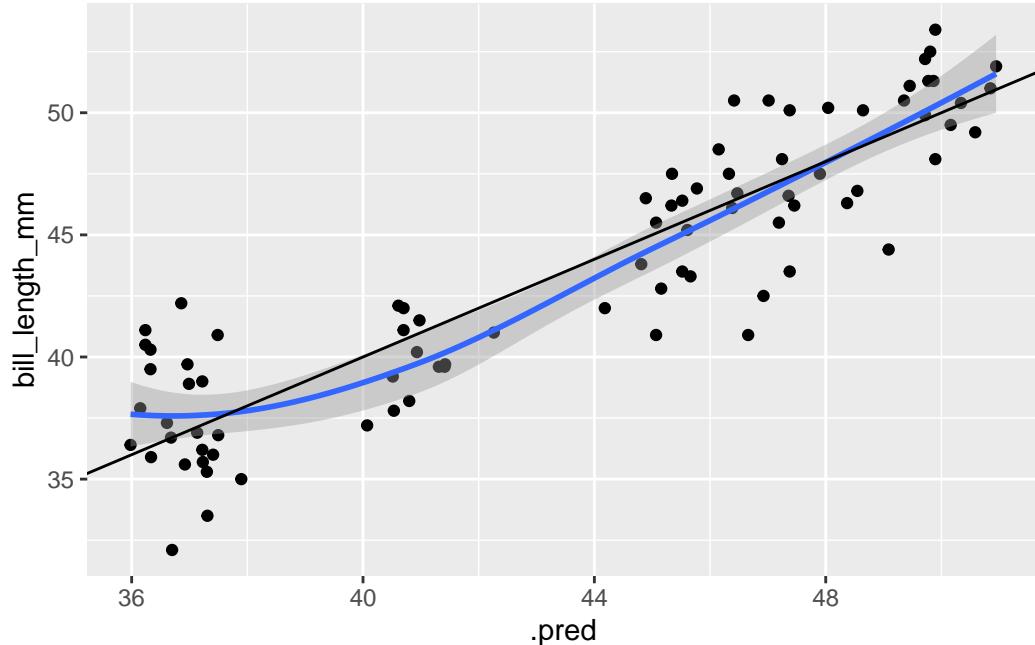
Rows: 84
 Columns: 10

	.pred	.resid	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
\$.pred	36.32554, 36.68002, 36.99019, 36.23692, 37.30036, 36~									
\$.resid	3.97446307, 0.01998059, 1.90980843, 4.86308368, -2.0~									
\$ species	Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~									
\$ island	Torgersen, Torgersen, Torgersen, Torgersen, Biscoe, ~									
\$ bill_length_mm	40.3, 36.7, 38.9, 41.1, 35.3, 40.5, 37.9, 39.5, 37.2~									
\$ bill_depth_mm	18.0, 19.3, 17.8, 17.6, 18.9, 17.9, 18.6, 16.7, 18.1~									
\$ flipper_length_mm	195, 193, 181, 182, 187, 187, 172, 178, 178, 196, 18~									
\$ body_mass_g	3250, 3450, 3625, 3200, 3800, 3200, 3150, 3250, 3900~									
\$ sex	female, female, female, female, female, female, fema~									
\$ year	2007, 2007, 2007, 2007, 2007, 2007, 2007~									

```

ggplot(prediction3, aes(x=.pred, y=bill_length_mm)) + geom_point() + geom_smooth() + geom_~`geom_smooth()`` using method = 'loess' and formula = 'y ~ x'

```



```

metrics(prediction3, truth = bill_length_mm, estimate = .pred)

# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>        <dbl>
1 rmse    standard     2.35
2 rsq     standard     0.811
3 mae    standard     1.92

```

The function `metrics` from package `yardstick` returns several related measures of agreement between prediction and the data in the test set. Probably the most common is the root mean squared error, that is the root of the sum of squared differences between predictions and observations. Notice that the rmse drops for each successive variable that we add to the model.

Exercise Add one or several more variables to the list of predictors by modifying the recipe, calculate the predictions, and compare their performance (e.g. the rmse on the test data) to the simpler models.

11.10 Other approaches

11.10.1 Minimum description length

Another completely different way to perform model selection is based on the idea on “Minimum Description Length”, where models are seen as a way to “compress” the data, and the model leading to the strongest compression should be favored. While we do not cover it here, you can read about it in [4].

11.10.2 Cross validation

One very robust method to perform model selection, often used in machine learning, is cross-validation. The idea is simple: split the data in three parts: a small data set for exploring; a large set for fitting; a small set for testing (for example, 5%, 75%, 20%). You can use the first data set to explore freely and get inspired for a good model. These data are then discarded. You use the largest data set for accurately fitting your model(s). Finally, you validate your model or select over competing models using the last data set.

Because you haven’t used the test data for fitting, this should dramatically reduce the risk of over-fitting. The downside of this is that we’re wasting precious data. There are less expensive methods for cross validation, but if you have much data, or data is cheap, then this has the virtue of being fairly robust.

11.10.2.1 Exercise: Do shorter titles lead to more citations?

To test the power of cross-validation, we are going to examine a bold claim by Letchford *et al.*, 2015: that papers with shorter titles attract more citations than those with longer titles. We are going to use their original data:

Letchford A, Moat HS, Preis T (2015) [The advantage of short paper titles](#). Royal Society Open Science 2(8): 150266.

```
# original URL
# https://datadryad.org/stash/dataset/doi:10.5061/dryad.hg3j0
dt <- read_csv("data/LMP2015.csv")

Rows: 140000 Columns: 4
-- Column specification -----
Delimiter: ","
chr (1): journal
dbl (3): year, title_length, cites

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The data set reports information on the top 20000 articles for each year from 2007 to 2013. The Author's claim is that shorter titles lead to more citations:

```
dt %>%
  group_by(year) %>%
  summarise(correlation = cor(title_length, cites, method = "kendall"))

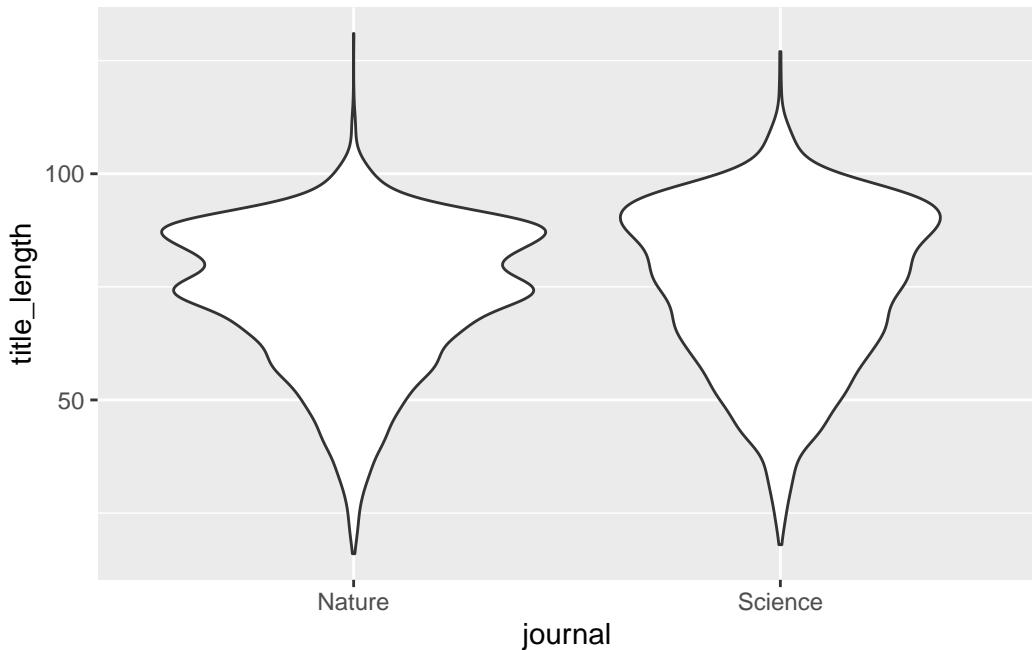
# A tibble: 7 x 2
  year correlation
  <dbl>      <dbl>
1 2007     -0.0535
2 2008     -0.0687
3 2009     -0.0560
4 2010     -0.0655
5 2011     -0.0525
6 2012     -0.0528
7 2013     -0.0451
```

As you can see, title length is anti-correlated (using rank correlation) with the number of citations.

There are several problems with this claim:

- The authors selected papers based on their citations. As such their claim would need to be stated as “among top-cited papers there is a correlation”.
- The journals cover a wide array of disciplines. The title length could reflect different publishing cultures.
- Most importantly, different journals have different requirements for title lengths. For example, Nature requires titles to be less than 90 characters:

```
dt %>% filter(journal %in% c("Nature", "Science")) %>%
  ggplot() + aes(x = journal, y = title_length) + geom_violin()
```



But then, is the effect the Authors are reporting only due to the fact that high-profile journals mandate short titles? Let's see whether their claims hold water when considering specific journals:

```
# only consider journals with more than 1000 papers in the data set
dt <- dt %>%
  group_by(journal) %>%
  mutate(num_papers = n()) %>%
```

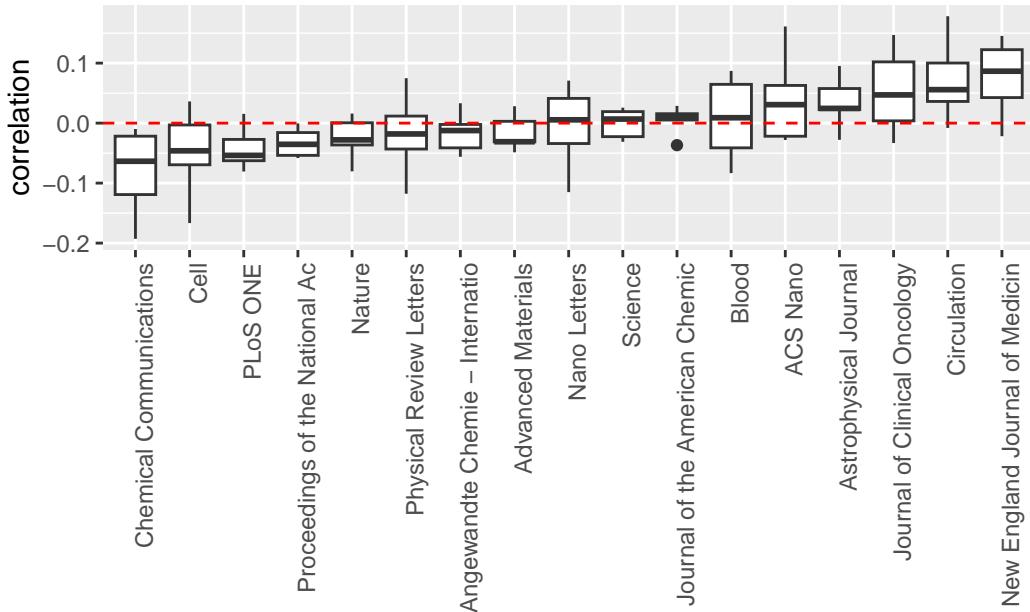
```

filter(num_papers > 1000) %>%
ungroup()

# now compute correlation and plot
dt %>%
  group_by(year, journal) %>%
  summarise(correlation = cor(title_length, cites, method = "kendall")) %>%
  ggplot() +
  aes(x = reorder(substr(journal, 1, 30), (correlation)), y = correlation) +
  geom_boxplot() +
  geom_hline(yintercept = 0, colour = "red", linetype = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) + # rotate labels x axis
  xlab("")

```

``summarise()` has grouped output by 'year'. You can override using the `groups` argument.`

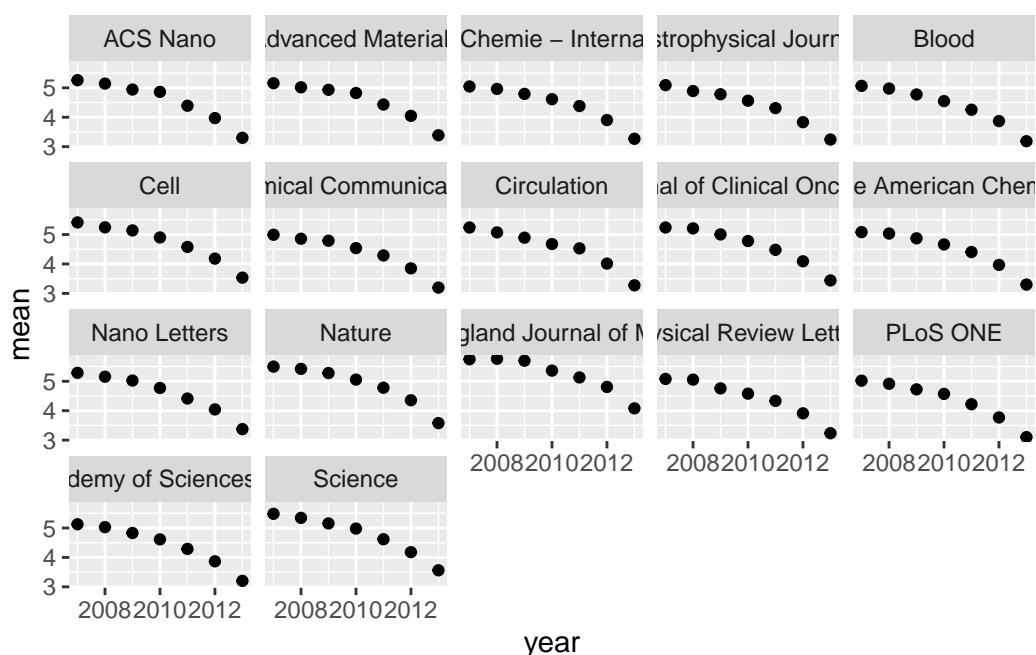


It seems that in several medical journals (NEJM, Circulation, J Clin Oncology) longer titles fare better than shorter ones. In Nature and PNAS we see a negative correlation, while Science gives no clear trend.

Let's look at the mean and standard deviation of citations by journal/year

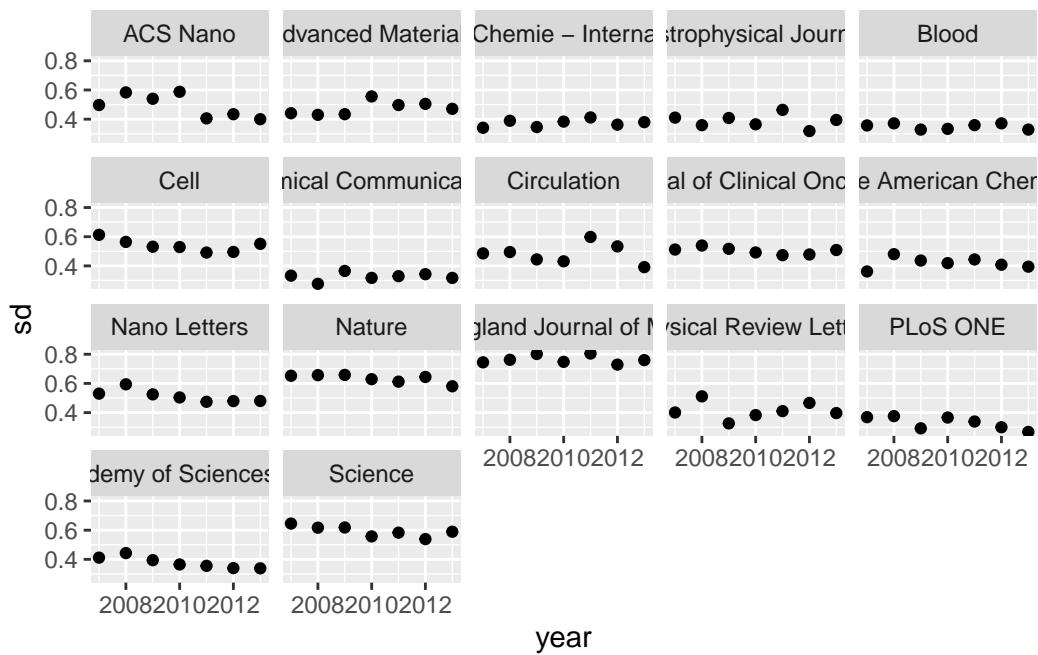
```
dt %>%
  group_by(journal, year) %>%
  summarise(mean = mean(log(cites + 1)), sd = sd(log(cites + 1))) %>%
  ggplot() +
  aes(x = year, y = mean) +
  geom_point() +
  facet_wrap(~journal)
```

`summarise()` has grouped output by 'journal'. You can override using the `.groups` argument.



```
dt %>%
  group_by(journal, year) %>%
  summarise(mean = mean(log(cites + 1)), sd = sd(log(cites + 1))) %>%
  ggplot() +
  aes(x = year, y = sd) +
  geom_point() +
  facet_wrap(~journal)
```

`summarise()` has grouped output by 'journal'. You can override using the `.groups` argument.



11.10.2.2 Two models

Let's consider two competing models.

Model1: each journal year has its mean

$$\log(\text{cits} + 1) \sim \text{journal : year}$$

Model2: the length of titles influences citations

$$\log(\text{cits} + 1) \sim \text{journal : year} + \text{title-length}$$

We are going to fit the model using 90% of the data; we are going to use the remaining data for cross-validation.

```
set.seed(4)
dt <- dt %>% mutate(logcit = log(cites + 1))
# sample 10% of the data
data_test <- dt %>% sample_frac(0.3)
data_fit <- anti_join(dt, data_test) # get all those not in data_test
```

Joining with `by = join_by(year, journal, title_length, cites, num_papers, logcit)`

Now fit the models:

```
M1 <- lm(logcit ~ factor(year)*journal, data = data_fit)
M2 <- lm(logcit ~ factor(year)*journal + title_length, data = data_fit)
```

Now let's try to predict out-of-fit the data that we haven't used:

```
M1_predictions <- predict(M1, newdata = data_test)
SSQ_M1 <- sum((log(data_test$cites + 1) - M1_predictions)^2)
M2_predictions <- predict(M2, newdata = data_test)
SSQ_M2 <- sum((log(data_test$cites + 1) - M2_predictions)^2)
print(SSQ_M1)
```

```
[1] 2465.712
```

```
print(SSQ_M2)
```

```
[1] 2465.96
```

We do not gain anything by including the information on titles.

- **Pros:** Easy to use; quite general; asymptotically equivalent to AIC.
- **Cons:** Sensitive to how the data was split (you can average over multiple partitions); need much data (instability in parameter estimates due to “data loss”)

11.11 References and further reading:

1. Pinheiro, José C.; Bates, Douglas M. (2000), Mixed-Effects Models in S and S-PLUS, Springer-Verlag, pp. 82–93
2. [Tidymodels tutorial](#)
3. [Emil Hvitfeldt, Tidymodels for Introduction to Statistical Learning in R](#)
4. [Mark H Hansen and Bin Yu Model Selection and the Principle of Minimum Description Length.](#)

12 Principal Component Analysis

Goal

Introduce Principal Component Analysis (PCA), one of the most popular techniques to perform “dimensionality reduction” of complex data sets. If we see the data as points in a high-dimensional space, we can project the data onto a new set of coordinates such that the first coordinate captures the largest share of the variance in the data, the second coordinates captures the largest share of the remaining variance and so on. In this way, we can project large-dimensional data sets onto low-dimensional spaces and lose the least information about the data.

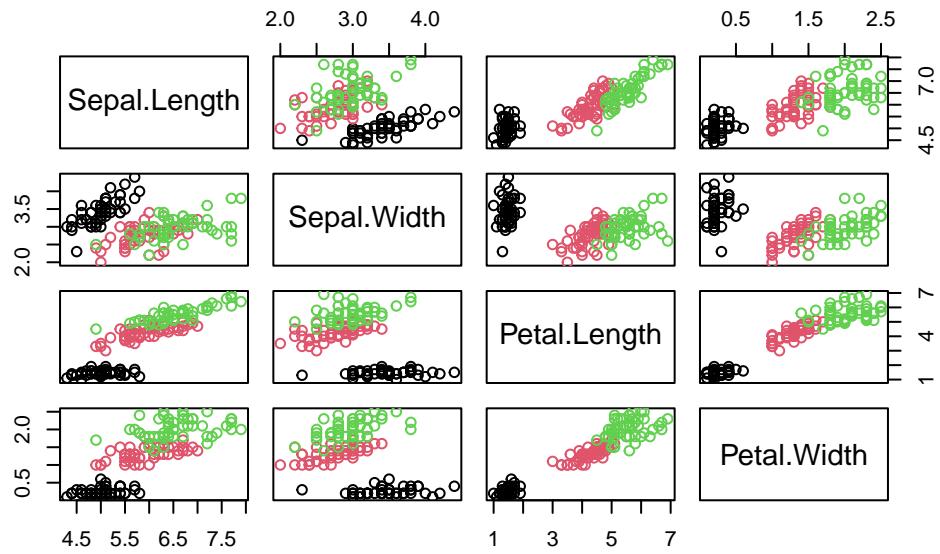
```
library(tidyverse)
library(ggmap) # for ggimage
library(ggfortify) # for autoplot
```

12.1 Input

We have collected the $n \times m$ data matrix X (typically, with $n \gg m$), in which the rows are samples and the columns are m measures on the samples. Each row of this matrix defines a point in the Euclidean space \mathbb{R}^m , i.e., each point in this space is a potential sample. Naturally, samples with similar measurements are “close” in this space, and samples that are very different are “far”. However, m can be quite large, and therefore we cannot easily visualize the position of the points. One way to think of PCA is as the best projection of the points in a r -dimensional space (with $r \leq m$), for visualization and clustering.

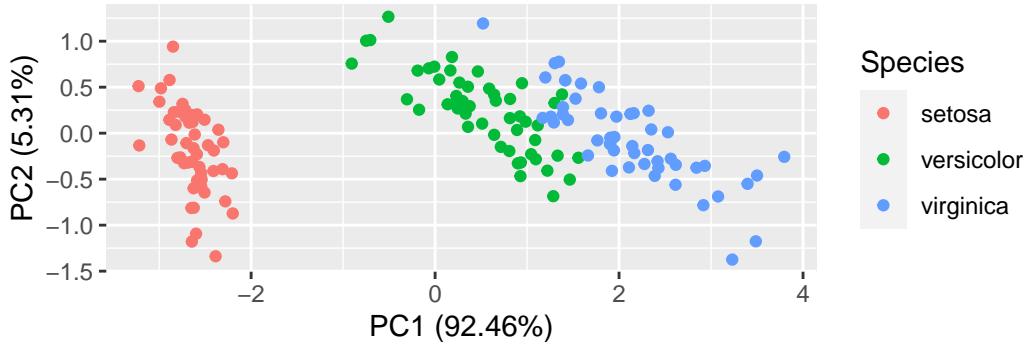
For example, take the iris data set:

```
data("iris")
ir <- iris %>% dplyr::select(-Species)
sp <- iris %>% dplyr::select(Species)
pairs(ir, col = sp$Species)
```



We can separate the clusters better by finding the best projection in 2D:

```
autoplot(prcomp(ir, center = TRUE),
        data = iris,
        colour = "Species",
        scale = FALSE) +
coord_equal()
```



12.2 Singular Value Decomposition

At the heart of PCA is a particular matrix decomposition (or factorization): we represent the matrix X as a product of other matrices (or, equivalently, a sum of matrices). In particular, SVD is defined by the equation:

$$X = U\Sigma V^T$$

X is a $n \times m$ matrix, U is an $n \times n$ **orthogonal, unitary** matrix and V is an $m \times m$ orthogonal, unitary matrix, and Σ is a $m \times n$ rectangular, diagonal matrix with non-negative values on the diagonal. If V is a (real) unitary matrix, then $VV^T = I_m$ (the $m \times m$ identity matrix), and if U is also unitary, then $UU^T = I_n$. Another way to put this is $U^{-1} = U^T$.

(Note: this defines the “full” SVD of A ; equivalently, one can perform a “thin”, or “reduced” SVD by having U of dimension $n \times p$, and Σ and V of dimension $p \times p$, where $p \leq m$ is the rank of A —by default R returns a “thin” SVD; read the details [here](#)).

The values on the diagonal of Σ are the **singular values** of X , i.e., the nonzero eigenvalues of XX^T (or X^TX). In this context, the matrix U contains the **left singular vectors** of X and V its **right singular vectors**. Let’s rearrange the rows/cols of Σ , U and V such that we have the singular values in decreasing order: $\text{diag}(\Sigma) = (\sigma_1, \sigma_2, \dots, \sigma_m)$.

Through SVD, the matrix X can be seen as a sum of m matrices:

$$X = \sum_{i=1}^m U_i \Sigma_{ii} V_i^T = X_1 + X_2 + X_3 + \dots$$

Where U_i is the i th column of U . Most importantly, you can prove that at each step (r), you are computing the **“best” approximation of X** as a sum of r rank-1 matrices. I.e., for each r we have that $\|X - (X_1 + X_2 + \dots + X_r)\|$ is as small as possible (Eckart–Young–Mirsky theorem).

Let’s look at a concrete example. A monochromatic image can be represented as a matrix where the entries are pixels taking values in (for example, using 8 bits) $0, 1, \dots, 255$:

```
stefano <- as.matrix(read.csv("data/stefano.txt"))
# invert y axis and transpose for visualization
stefano <- t(stefano[,ncol(stefano):1])
# rescale values to suppress warning from ggimage
stefano <- stefano / max(stefano)
ggimage(stefano)
```



Now let's perform SVD, and show that indeed we have factorized the image:

```
s_svd <- svd(stefano)
U <- s_svd$u
V <- s_svd$v
Sigma <- diag(s_svd$d)
# this should be equal to the original matrix
stefano_2 <- U %*% Sigma %*% t(V)
# let's plot the difference
ggimage(round(stefano - stefano_2, 10))
```



Now we can visualize the approximation we're making when we take only the first few singular values. We're going to plot X_k (on the left), and $\sum_{i=1}^k X_i$ (on the right). Even with only a few iterations (7, out of 255) we obtain a recognizable image:

```
r <- 7
Xdec <- array(0, c(dim(stefano), r))
Xsum <- array(0, c(dim(stefano), r))
# store the first matrix
Xdec[,1] <- (U[,1] %*% t(V[,1])) * Sigma[1,1]
# the first term in the sum is the matrix itself
Xsum[,1] <- Xdec[,1]
# store the other rank one matrices, along with the partial sum
for (i in 2:r){
  Xdec[,i] <- (U[,i] %*% t(V[,i])) * Sigma[i,i]
  Xsum[,i] <- Xsum[,i - 1] + Xdec[,i]
}
# now plot all matrices and their sum
plots <- list()
for (i in 1:r){
  plots[[length(plots) + 1]] <- ggimage(Xdec[,i])
  plots[[length(plots) + 1]] <- ggimage(Xsum[,i])
}
```

```
gridExtra::grid.arrange(grobs = plots, ncol = 2)
```

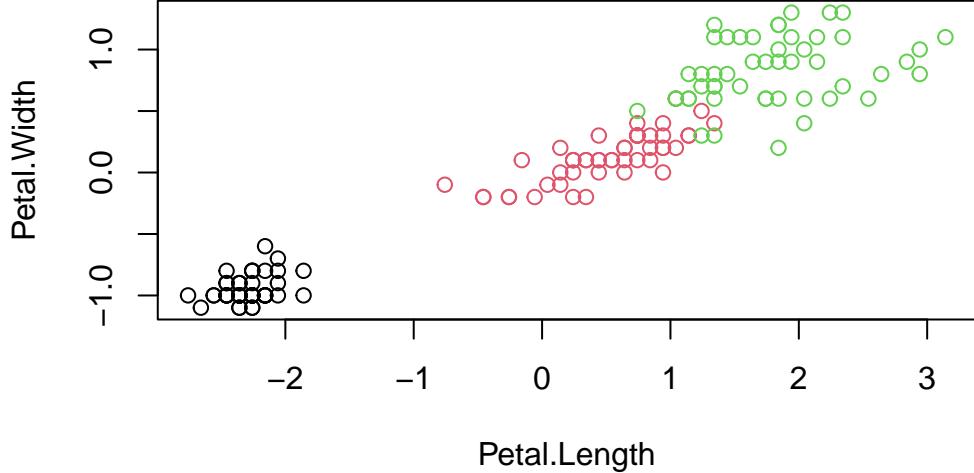


12.3 SVD and PCA

Let's go back to our data matrix X , and its representation as n points (the samples) in m dimensions (the measurements). For the moment, consider the case in which **each column of X** sums to zero (i.e., for each measurement, we have removed the mean—this is called “centering”). We would like to represent the data as best as possible in few dimensions, such that a) the axes are orthogonal; b) the axes are aligned with the principal sources of variation in the data. More precisely, PCA is an **orthogonal linear transformation** that transforms the data to a **new coordinate system** such that the direction of greatest variance of the data is aligned with the first coordinate, the second greatest with the second coordinate, and so on.

For example, let's take the Petal.Length and Petal.Width in `iris`:

```
X <- iris %>% dplyr::select(Petal.Length, Petal.Width) %>% as.matrix()
X <- scale(X, center = TRUE, scale = FALSE) # remove mean
colors <- iris$Species
plot(X, col = colors)
```



You can see that now the points are centered at (0,0).

In practice, we want to produce a new “data matrix” Y :

$$Y = XW$$

where W is an appropriate change of basis, transforming the data such that the directions of main variation are exposed. While we could choose any $m \times m$ matrix, we want a) W to be orthogonal (i.e., a “rotation” of the data), and b) all columns of W to be unit vectors (no stretching of the data).

The new columns (i.e., the transformed “measurements”) Y_i can be written as:

$$Y_i = XW_i$$

Where Y_i is the i th column of Y and W_i the i th column on W . Let’s start with the first column Y_1 : we want to choose W_1 such that the variance of Y_i is maximized. Because the mean of each column of X is zero, then also the mean of Y_i is zero. Thus, the variance is simply $\frac{1}{n-1} \sum_{j=1}^n Y_{ij}^2 = \frac{1}{n-1} \|Y_i\|^2$. We can write this in matrix form:

$$\frac{1}{n-1} \|Y_i\|^2 = \frac{1}{n-1} \|XW_i\|^2 = \frac{1}{n-1} W_i^T X^T X W_i$$

Note that $S = \frac{1}{n-1} X^T X$ is the $m \times m$ sample covariance matrix of X . Because $\|W_i\| = 1$, we can rewrite this as:

$$\frac{1}{n} \|Y_i\|^2 = \frac{W_i^T S W_i}{W_i^T W_i}$$

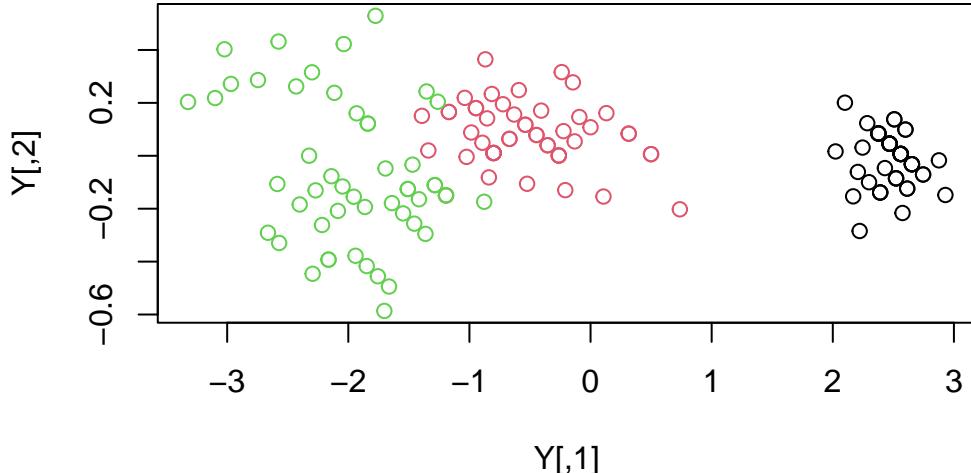
Which is maximized (over W_i) when W_i is the eigenvector of S associated with the largest eigenvalue (see the [Rayleigh quotient](#)), in which case:

$$\frac{1}{n-1} \|Y_i\| = \frac{W_i^T S W_i}{W_i^T W_i} = \lambda_1$$

Therefore, the first column of Y is given by the projection of the data on the first eigenvector of S . The variance captured by this first axis is given by the largest eigenvalue of S . To find the other columns of Y , you can subtract from X the matrix $Y_1 W_1^T$ and repeat.

Note that the first axis captures $\lambda_1 / \sum_{i=1}^m \lambda_i$ of the total variance in X . This is typically reported in PCA as the “loadings” of the various components.

```
# build sample covariance matrix
S <- (1 / (nrow(X) - 1)) * t(X) %*% X
# compute eigenvalues and eigenvectors
eS <- eigen(S, symmetric = TRUE)
# W is the matrix of eigenvectors
W <- eS$vectors
# check
Y <- X %*% W
plot(Y, col = colors)
```



```
eS$values
```

```
[1] 3.66123805 0.03604607
```

```
apply(Y, 2, var)
```

```
[1] 3.66123805 0.03604607
```

Therefore, PCA amounts to simply taking the eigenvectors of S ordered by the corresponding eigenvalues. We can use the SVD to accomplish this task efficiently:

$$X = U\Sigma V^T$$

$$\begin{aligned}(n-1)S &= X^T X = (V\Sigma^T U^T)(U\Sigma V^T) \\ &= V\Sigma^T \Sigma V^T \\ &= V\tilde{\Sigma}^2 V^T\end{aligned}$$

where $\tilde{\Sigma}^2 = \Sigma^T \Sigma$ (or, equivalently the square of the square version of Σ). But contrasting $S = W\Lambda W^T$ and $S = V(\tilde{\Sigma}^2/(m-1))V^T$ we see that $V = W$. Finally, we have:

$$Y = XW = U\Sigma V^T V = U\Sigma$$

Therefore, we can perform PCA efficiently by decomposing X using SVD.

12.3.1 PCA in R—from scratch

```
dt <- read_csv("data/handwritten_digits.csv") %>%
  arrange(id, x, y)
```

```
Rows: 123392 Columns: 6
-- Column specification -----
Delimiter: ","
dbl (6): id, label, pixel, value, x, y
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(dt)
```

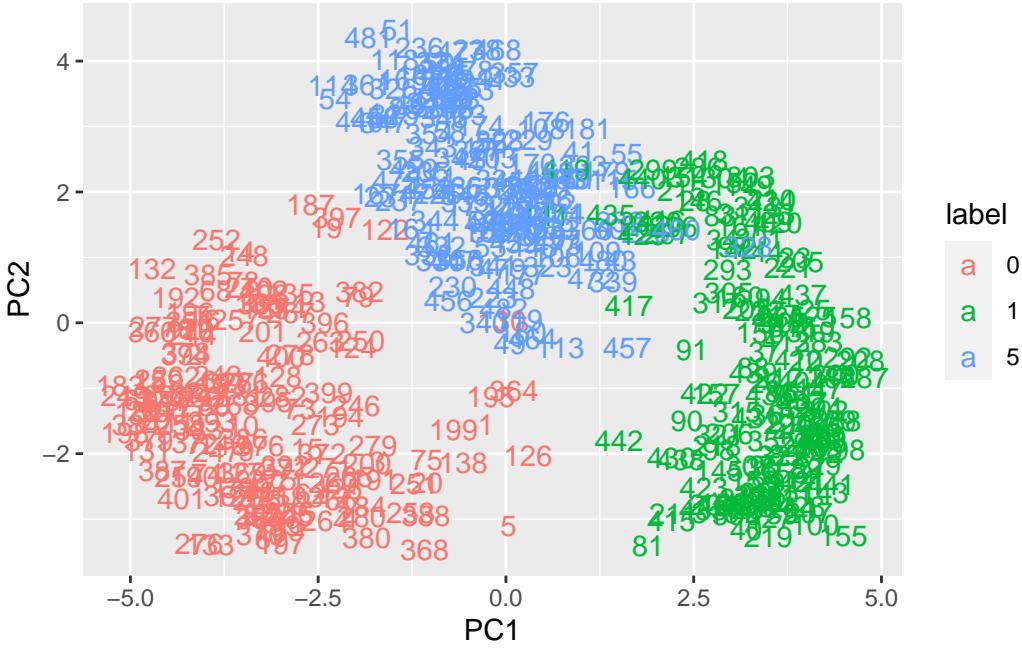
```

# A tibble: 6 x 6
  id label pixel value     x     y
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1     1     0     0     0     1     1
2     1     0    16     0     1     2
3     1     0    32     0     1     3
4     1     0    48     0     1     4
5     1     0    64     0     1     5
6     1     0    80     0     1     6

# make into a data matrix with pixels as cols
dt_wide <- pivot_wider(dt %>% dplyr::select(-x, -y),
                        names_from = pixel,
                        values_from = value)
X <- (as.matrix(dt_wide %>% dplyr::select(-id, -label)))
# make col means = 0
Xs <- scale(X, center = TRUE, scale = FALSE)
# compute SVD
X_svd <- svd(Xs)
# Y = US is the transformed data
Y <- X_svd$u %*% diag(X_svd$d)

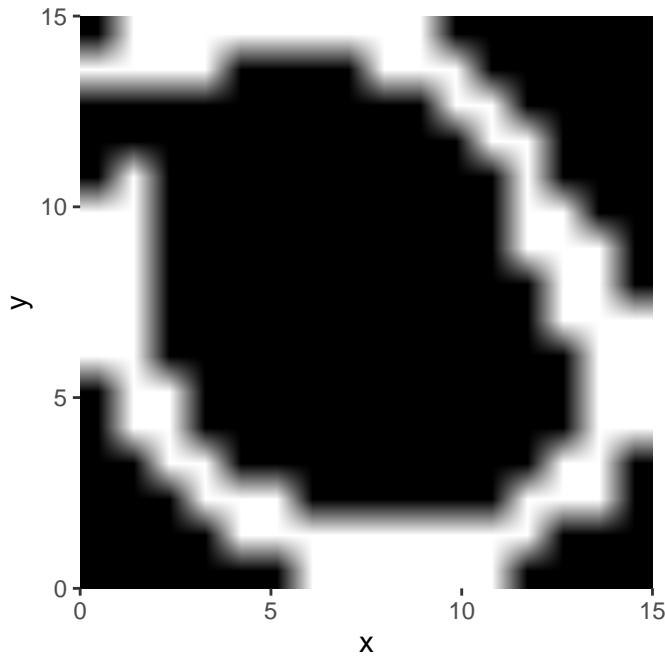
PCA_1 <- dt_wide %>%
  dplyr::select(id, label) %>%
  mutate(label = as.character(label)) %>%
  add_column(PC1 = Y[,1], PC2 = Y[,2])
ggplot(PCA_1) +
  aes(x = PC1, y = PC2, label = id, group = label, colour = label) +
  geom_text()

```

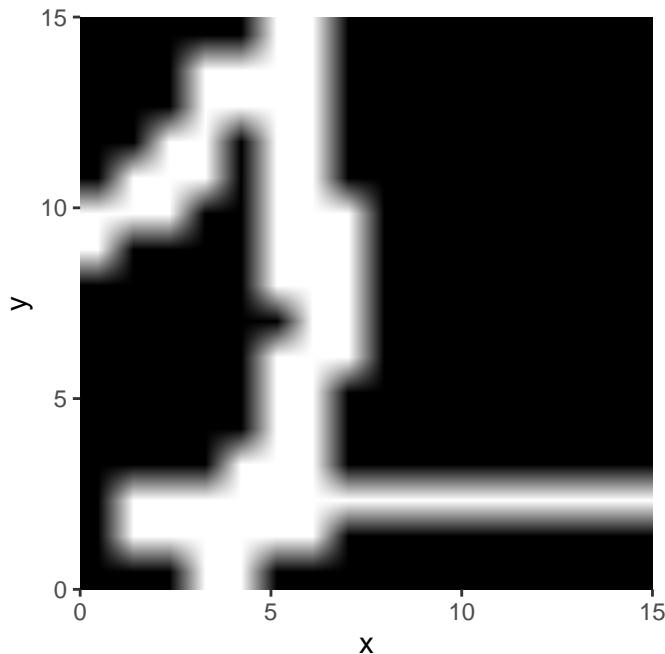


Pretty good! Let's see some of the poorly classified points:

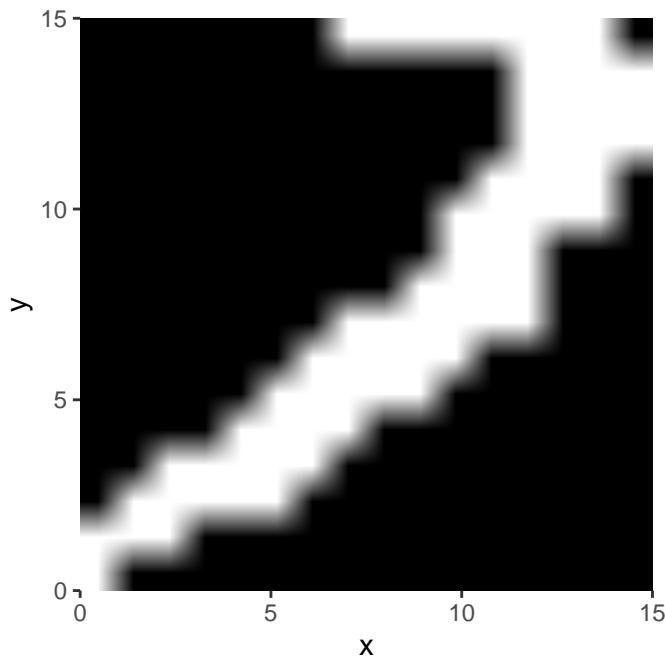
```
# This should be a 0
ggimage(matrix(X[122,], 16, 16, byrow = FALSE), fullpage = FALSE)
```



```
# This should be a 1  
ggimage(matrix(X[141,], 16, 16, byrow = FALSE), fullpage = FALSE)
```



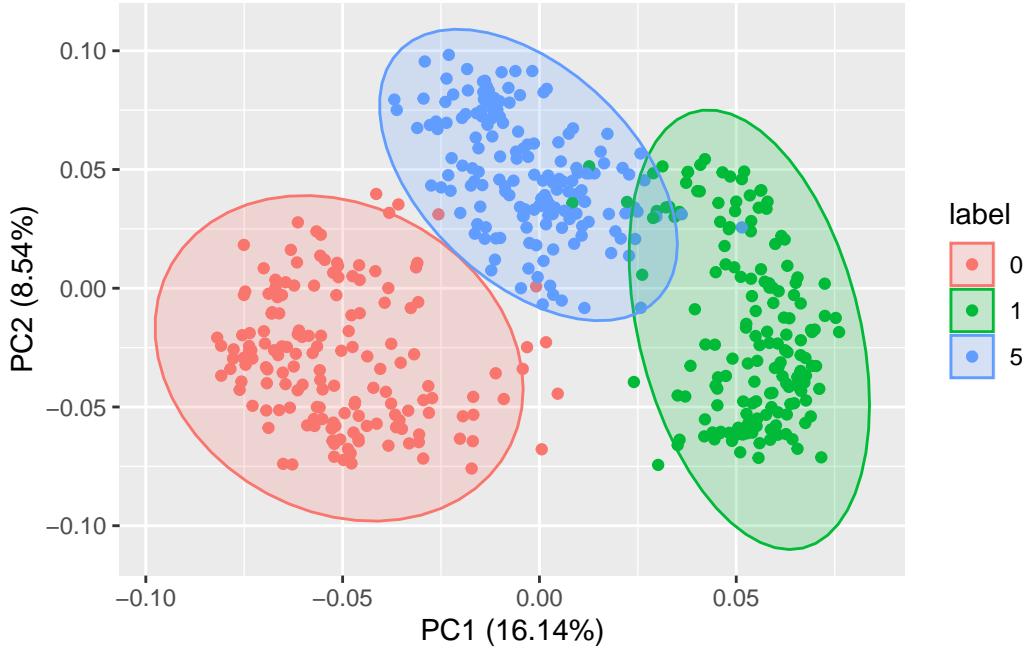
```
# This should be a 5  
ggimage(matrix(X[322,], 16, 16, byrow = FALSE), fullpage = FALSE)
```



You can also scale the variables turning the sample covariance matrix S into a correlation matrix (this is useful when the variance of different measurements varies substantially).

12.3.2 PCA in R — the easy way

```
library(ggfortify)
# for prcomp, you need only numeric data
X <- dt_wide %>% dplyr::select(-id, -label)
PCA_3 <- prcomp(X)
autoplot(PCA_3,
        data = dt_wide %>% mutate(label = as.character(label)),
        colour = "label",
        frame = TRUE, frame.type = 'norm')
```



12.4 Multidimensional scaling

The input is the matrix of dissimilarities D , potentially representing distances $d_{ij} = d(x_i, x_j)$. A distance function is “metric” if:

- $d(x_i, x_j) \geq 0$ (non-negativity)
- $d(x_i, x_j) = 0$ only if $x_i = x_j$ (identity)
- $d(x_i, x_j) = d(x_j, x_i)$ (symmetry)
- $d(x_i, x_k) \leq d(x_i, x_j) + d(x_j, x_k)$ (triangle inequality)

Given a set of dissimilarities, we can therefore ask whether they are distances, and particularly whether they represent Euclidean distances.

12.4.1 Goal of MDS

Given the $n \times n$ matrix D , find a set of coordinates $x_1, \dots, x_n \in \mathbb{R}^p$, such that $d_{ij} \approx \|x_i - x_j\|_2$ (as close as possible). The operator $\|\cdot\|_2$ is the Euclidean norm, measuring Euclidean distance.

As such, if we can find a perfect solution, then the dissimilarities can be mapped into Euclidean distances in a k -dimensional space.

12.4.2 Classic MDS

Suppose that the elements of D measure Euclidean distances between n points, each of which has k coordinates:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix}$$

We consider the centered coordinates:

$$\sum_i x_{ij} = 0$$

And the matrix $B = XX^t$, whose coefficients are $B_{ij} = \sum_k x_{ik}x_{jk}$. We can write the square of the distance between point i and j as:

$$d_{ij}^2 = \sum_k (x_{ik} - x_{jk})^2 = \sum_k x_{ik}^2 + \sum_k x_{jk}^2 - 2 \sum_k x_{ik}x_{jk} = B_{ii} + B_{jj} - 2B_{ij}$$

Note that, because of the centering:

$$\sum_i B_{ij} = \sum_i \sum_k x_{ik}x_{jk} = \sum_k x_{jk} \sum_i x_{ik} = 0$$

Now we compute:

$$\sum_i d_{ij}^2 = \sum_i (B_{ii} + B_{jj} - 2B_{ij}) = \sum_i B_{ii} + \sum_i B_{jj} - 2 \sum_i B_{ij} = \text{Tr}(B) + nB_{jj}$$

Similarly (distances are symmetric):

$$\sum_j d_{ij}^2 = \text{Tr}(B) + nB_{ii}$$

And, finally:

$$\sum_i \sum_j d_{ij}^2 = 2n\text{Tr}(B)$$

From these three equations, we obtain:

$$B_{ii} = \frac{\sum_j d_{ij}^2}{n} - \frac{\sum_i \sum_j d_{ij}^2}{2n^2}$$

and

$$B_{jj} = \frac{\sum_i d_{ij}^2}{n} - \frac{\sum_i \sum_j d_{ij}^2}{2n^2}$$

Therefore:

$$B_{ij} = -\frac{1}{2}(d_{ij}^2 - B_{ii} - B_{jj}) = -\frac{1}{2} \left(d_{ij}^2 - \frac{\sum_i d_{ij}^2}{n} - \frac{\sum_j d_{ij}^2}{n} + \frac{\sum_i \sum_j d_{ij}^2}{n^2} \right)$$

With some algebra, one can show that this is equivalent to:

$$B = -\frac{1}{2}CD^{(2)}C$$

Where $D^{(2)}$ is the matrix of squared distances, and C is the centering matrix $C = 1 - \frac{1}{n}\mathcal{O}$ (and \mathcal{O} is the matrix of all ones). Thus, we can obtain B directly from the distance matrix. Once we've done this, X can be found by taking the eigenvalue decomposition:

$$B = XX^t = Q\Lambda Q^t$$

(where Q is the matrix of eigenvectors of B , and Λ a diagonal matrix of the eigenvalues of B). Therefore:

$$X = Q\Lambda^{\frac{1}{2}}$$

For example, let's look at the driving distance in km between cities in the US:

```
# read distances US
usa <- read_csv("data/dist_US.csv")
```

```
Rows: 265356 Columns: 3
-- Column specification -----
Delimiter: ","
chr (2): from, to
dbl (1): dist
```

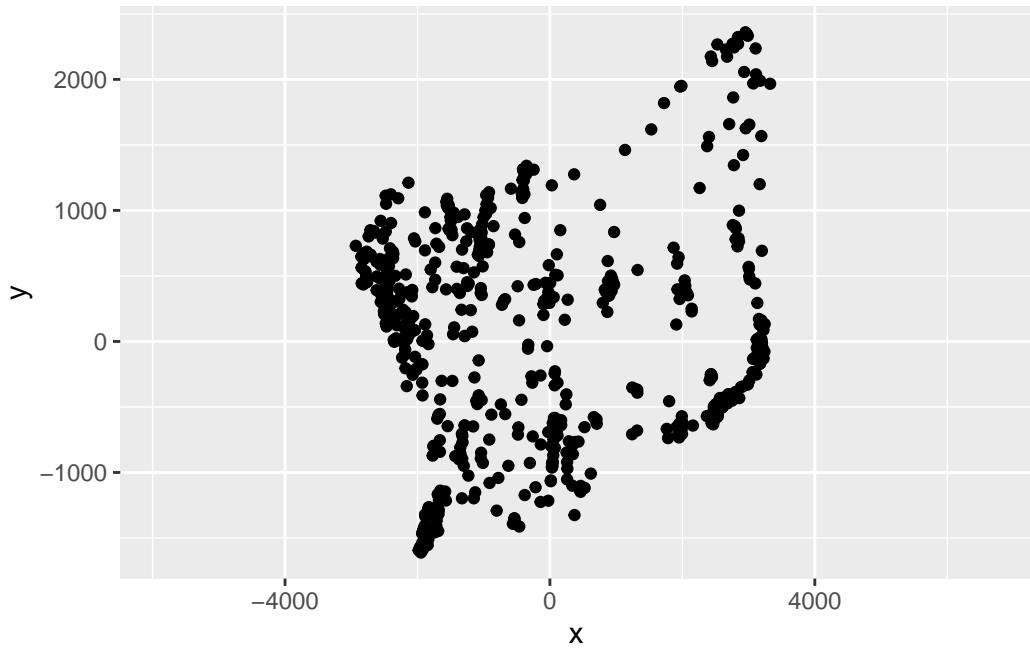
```
i Use `spec()` to retrieve the full column specification for this data.  
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# make into a matrix of distances  
M <- usa %>% pivot_wider(names_from = to, values_from = `dist`) %>%  
  dplyr::select(-from) %>%  
  as.matrix()  
M[is.na(M)] <- 0  
rownames(M) <- colnames(M)  
# make symmetric  
M <- M + t(M)  
M[1:2, 1:2]
```

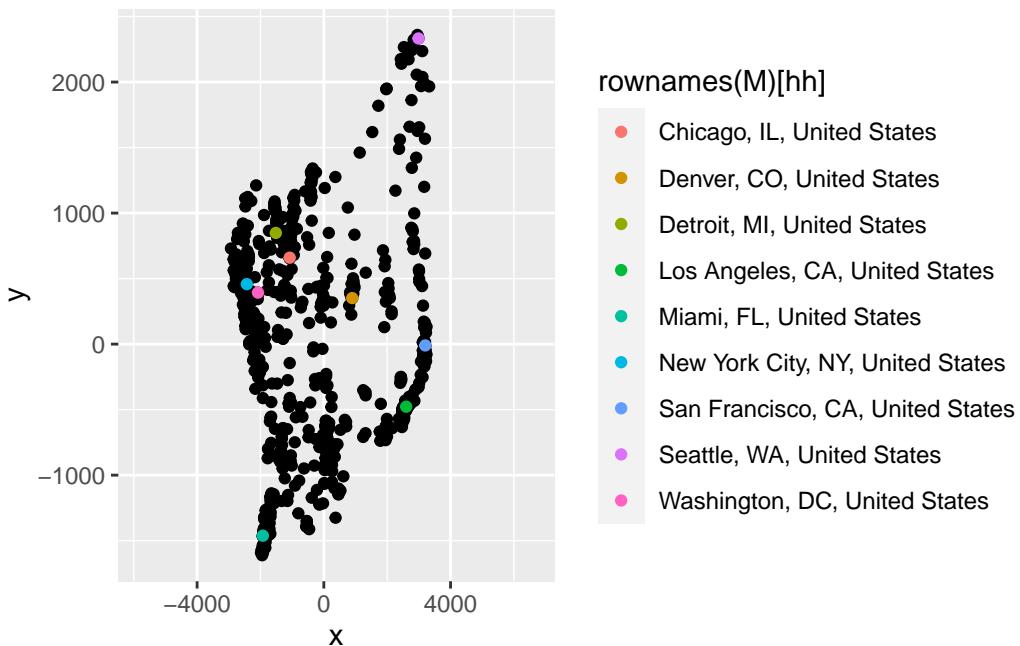
	Abilene, TX, United States	
Abilene, TX, United States		0.00
Ahwatukee Foothills, AZ, United States		1487.19
	Ahwatukee Foothills, AZ, United States	
Abilene, TX, United States		1487.19
Ahwatukee Foothills, AZ, United States		0.00

And perform classic MDS using two dimensions:

```
mds_fit <- cmdscale(M, k = 2) # k is the dimension of the embedding  
mds_fit <- tibble(id = rownames(M),  
                  x = mds_fit[,1], y = mds_fit[,2])  
pl <- mds_fit %>%  
  ggplot() +  
  aes(x = x, y = y) +  
  geom_point() +  
  xlim(2 * range(mds_fit$x))  
  
show(pl)
```



```
# highlight some major cities
hh <- c(122, 175, 177, 373, 408, 445, 572, 596, 691)
mds_highlight <- mds_fit %>% slice(hh)
show(pl + geom_point(data = mds_highlight, aes(colour = rownames(M)[hh])))
```



12.5 Readings

SVD is the most important decomposition, but several interesting variations have been proposed for data science. Read this [very cool paper](#) on face recognition using Non-negative Matrix Factorization.

12.5.1 Exercise: PCA sommelier

The file `Wine.csv` contains several measures made on 178 wines from Piedmont, produced using three different grapes (column `Grape`, with 1 = Barolo, 2 = Grignolino, 3 = Barbera). Use the 13 measured variables (i.e., all but `Grape`) to perform a PCA. First, do it “the hard way” using SVD, and then, calling the `prcomp` function. Can you recover the right classification of grapes?

13 Clustering

Goals

- Learn about partitional clustering
- Learn about hierarchical clustering
- Use clustering validation methods
- Apply different methods to larger data sets

```
library(tidyverse)
library(ggfortify)
library(factoextra)
library(NbClust)
library(fpc)
library(clustertend)
library(palmerpenguins)
```

The goal of clustering is to classify data points into groups (clusters) by without giving the algorithm any knowledge of the correct classification. This type of approach is called *unsupervised learning* and it is appropriate when the “truth” for your data classification is unavailable or difficult to obtain.

If the truth is unknown, we need a way of deciding which data points belong together. One common set of approaches relies on a measure of closeness, or distance between points. Of those, the classic K-means approach is the most straightforward.

13.1 K-means algorithm

- divide data into K clusters
- calculate centroids for each
- go through each data point until nothing changes
 - calculate distance to each centroid
 - assign to nearest centroid
 - recalculate centroids for the two affected clusters

Let us apply the k-means algorithm to our well-studied penguin data set. In the script below, we remove the NAs, and select out the categorical variables, as they are not directly useful for the distance-based algorithm, leaving the four numeric variables to define similarity between individuals. The question is, will they cluster penguins according to species?

```
#set.seed(20)  
glimpse(penguins)
```

```
pen_data <- penguins %>% drop_na()
#pen_train <- pen_data  %>% dplyr::select(-species,-island, -sex, -year) # remove species
pen_train <- pen_data  %>% dplyr::select(-species,-island, -sex) # remove species (the tru
pen_km <- kmeans(pen_train, 3) #k-means with 3 clusters
pen_km
```

K-means clustering with 3 clusters of sizes 140, 113, 80

Cluster means:

```

bill_length_mm bill_depth_mm flipper_length_mm body_mass_g      year
1        41.12214     17.94643       189.6286    3461.250 2008.021
2        44.24336     17.44779       201.5487    4310.619 2008.000
3        48.66250     15.39750       219.9875    5365.938 2008.138

```

Clustering vector:

```
[223] 2 3 2 3 2 3 2 3 2 3 3 3 3 3 3 2 3 3 3 3 3 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 2 3 3 3 2 3 3
[260] 3 3 3 3 3 3 1 2 1 1 1 2 1 1 2 1 1 1 1 2 1 2 1 1 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 2 1
[297] 2 1 2 1 2 1 2 1 2 2 1 1 1 1 2 1 2 1 1 1 2 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1
```

Within cluster sum of squares by cluster:

```
[1] 9724908 9318106 9718878  
(between_SS / total_SS =  86.6 %)
```

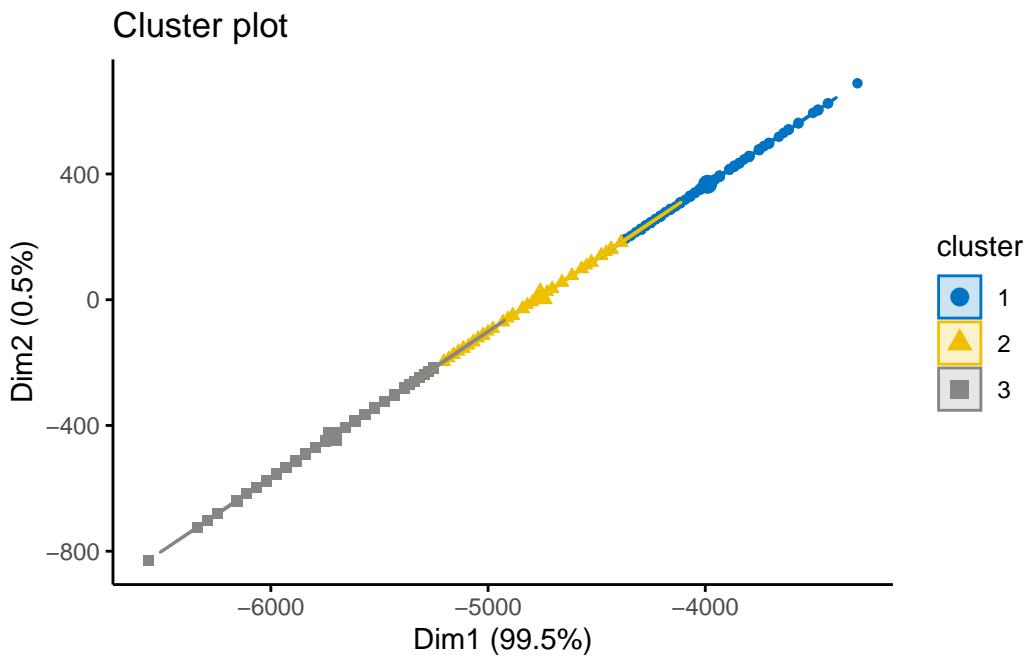
Available components:

```
[1] "cluster"      "centers"       "totss"        "withinss"      "tot.withinss"  
[6] "betweenss"    "size"          "iter"         "ifault"
```

```
table(pen_km$cluster, pen_data$species)
```

	Adelie	Chinstrap	Gentoo
1	94	46	0
2	52	22	39
3	0	0	80

```
fviz_cluster(list(data = pen_train, cluster = pen_km$cluster),  
ellipse.type = "norm", geom = "point", stand = FALSE, palette = "jco", ggtheme = theme_cla
```



The plot is produced by performing PCA to reduce the number of variables from 4 to 2, helping present the data points in the way that optimizes their visual separation. Notice that the clusters are not well separated, and when compared with the actual classification given by `species`, they do not do well.

However, the four measurements have very different variances, so we try scaling them to make them all have equal variance of one:

```
pen_data <- penguins %>% drop_na()
#pen_scaled <- scale(pen_data %>% dplyr::select(-species, -island, -sex, -year) )
pen_scaled <- scale(pen_data %>% dplyr::select(-species, -island, -sex) )
pen_km <- kmeans(pen_scaled, 3)
pen_km
```

K-means clustering with 3 clusters of sizes 131, 83, 119

Cluster means:

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	year
1	-0.56094130	0.4690575	-0.5913040	-0.6503351	0.66192839
2	-0.05199786	0.8382938	-0.7308966	-0.5500394	-1.08914719
3	0.65377423	-1.1010497	1.1607163	1.0995561	0.03097981

Clustering vector:

Within cluster sum of squares by cluster:

[1] 265.0704 189.7556 250.6269
(between_SS / total_SS = 57.5 %)

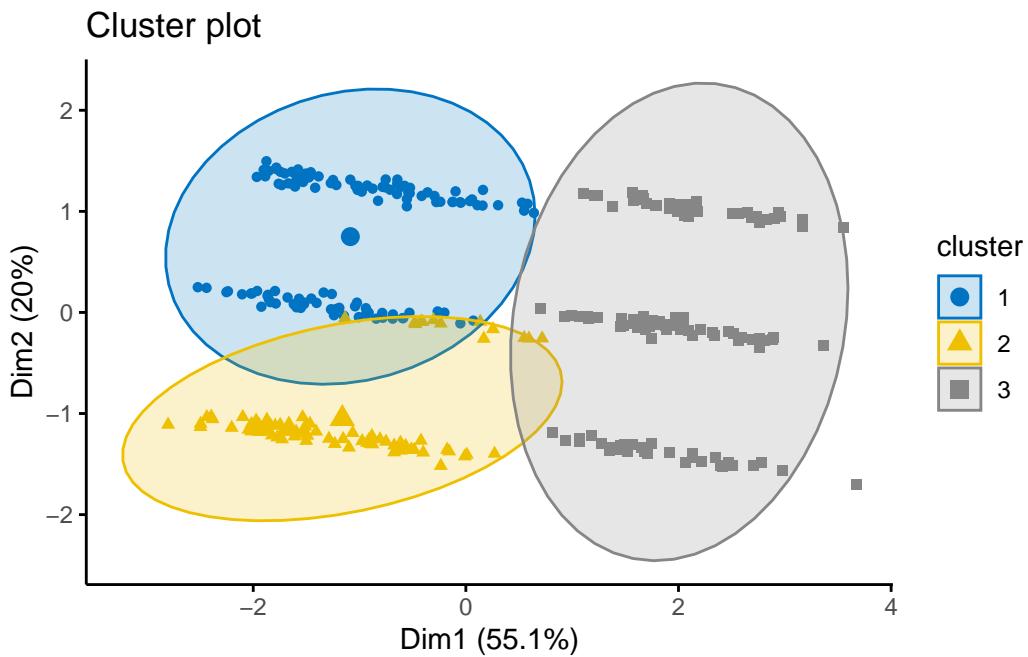
Available components:

```
[1] "cluster"        "centers"        "totss"          "withinss"        "tot.withinss"  
[6] "betweenss"      "size"           "iter"           "ifault"
```

```
table(pen_km$cluster, pen_data$species)
```

	Adelie	Chinstrap	Gentoo
1	100	31	0
2	46	37	0
3	0	0	119

```
fviz_cluster(list(data = pen_scaled, cluster = pen_km$cluster),  
ellipse.type = "norm", geom = "point", stand = FALSE, palette = "jco", ggtheme = theme_cla
```



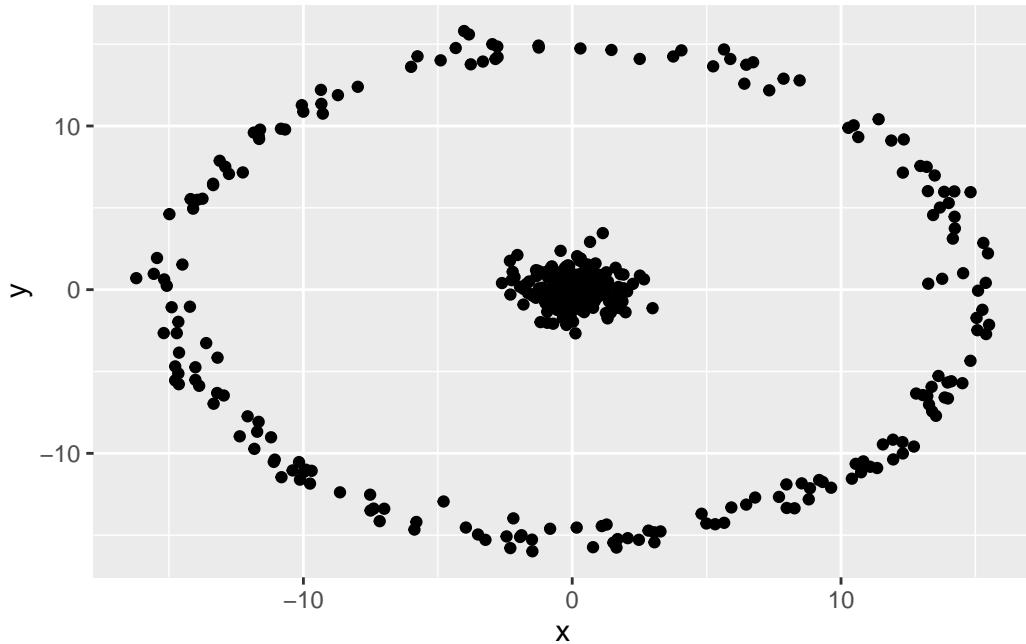
Now we get much better separation, as well as much better prediction quality. However, if you run the above code several times, you will see different results, because k-means starts with a random selection of centroids. In cases like this, where there is not very obvious clusters, it may converge to different classifications. Here, for some trials we see very good prediction quality for all three species, but other times two of the species are commingled.

13.1.1 Assumptions of K-means algorithm

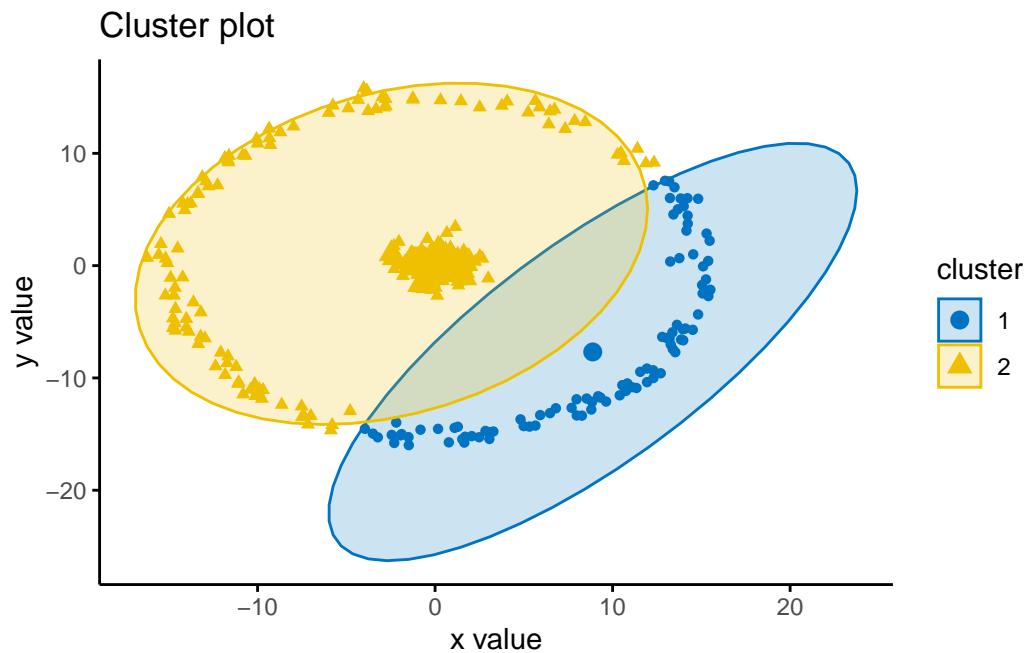
- There is a meaningful distance measure
- Clusters are roughly spherical
- Clusters are of similar size

```
# Generate random data which will be first cluster
clust1 <- data_frame(x = rnorm(200), y = rnorm(200))
# Generate the second cluster which will 'surround' the first cluster
clust2 <- data_frame(r = rnorm(200, 15, .5),
                      theta = runif(200, 0, 2 * pi),
                      x = r * cos(theta), y = r * sin(theta)) %>%
  dplyr::select(x, y)
#Combine the data
dataset_cir <- rbind(clust1, clust2)
#see the plot
```

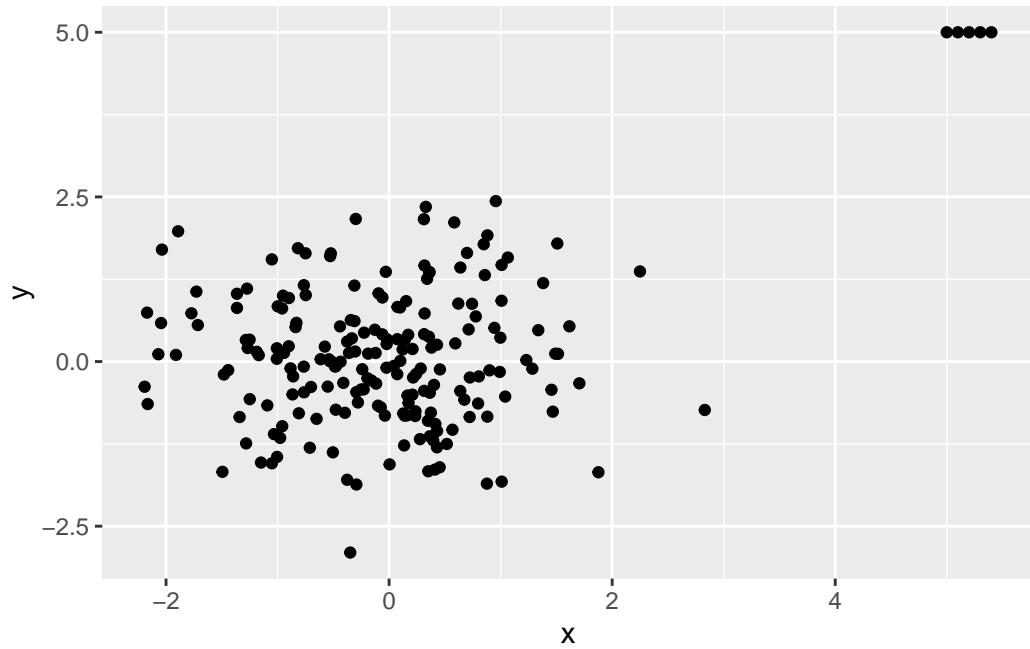
```
dataset_cir %>% ggplot() + aes(x = x, y = y) + geom_point()
```



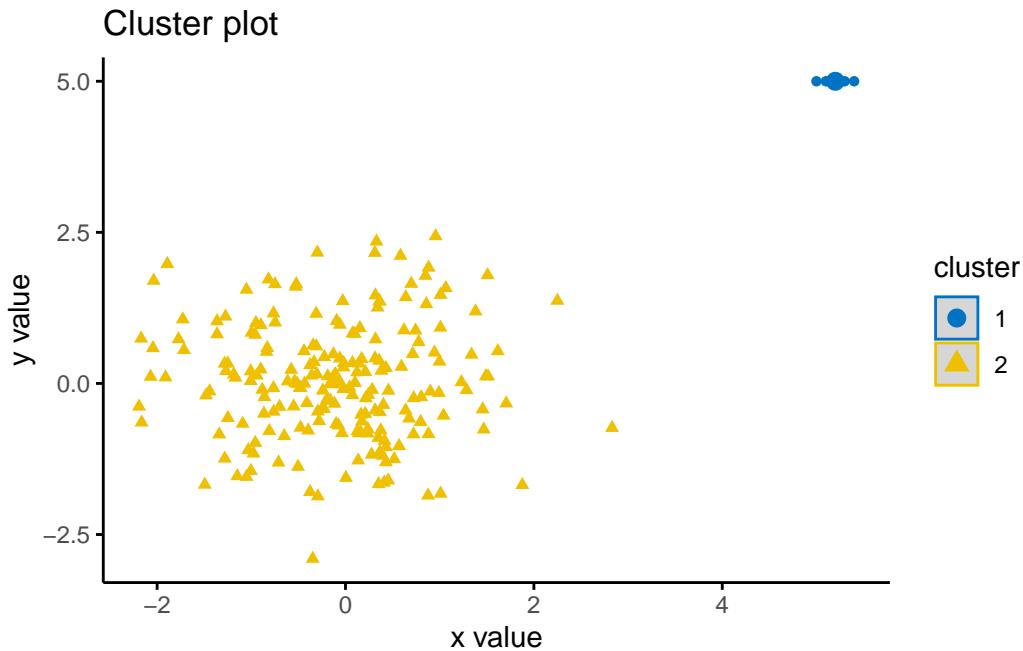
```
#Fit the k-means model  
k_clust_spher1 <- kmeans(dataset_cir, centers=2)  
#Plot the data and clusters  
fviz_cluster(list(data = dataset_cir,  
                  cluster = k_clust_spher1$cluster),  
             ellipse.type = "norm",  
             geom = "point", stand = FALSE,  
             palette = "jco",  
             ggtheme = theme_classic())
```



```
# Make the first cluster with 200 random values
clust1 <- data_frame(x = rnorm(200),
                      y = rnorm(200))
# Keep 10 values together to make the second cluster
clust2 <- data_frame(x=c(5,5.1,5.2,5.3,5.4),
                      y=c(5,5,5,5,5))
#Combine the data
dataset_uneven <- rbind(clust1,clust2)
dataset_uneven %>% ggplot() + aes(x = x, y = y) + geom_point()
```



```
k_clust_spher3 <- kmeans(dataset_uneven, centers=2)
fviz_cluster(list(data = dataset_uneven,
                  cluster = k_clust_spher3$cluster),
             ellipse.type = "norm",
             geom = "point",
             stand = FALSE,
             palette = "jco",
             ggtheme = theme_classic())
```



13.2 Hierarchical clustering

Hierarchical clustering is different approach from k-means, although it is also based on a notion of distance. The goal is to create a tree, akin to phylogeny, based on proximity of different points to each other, and then to divide it into groups by *cutting* the tree a certain depth from the root.

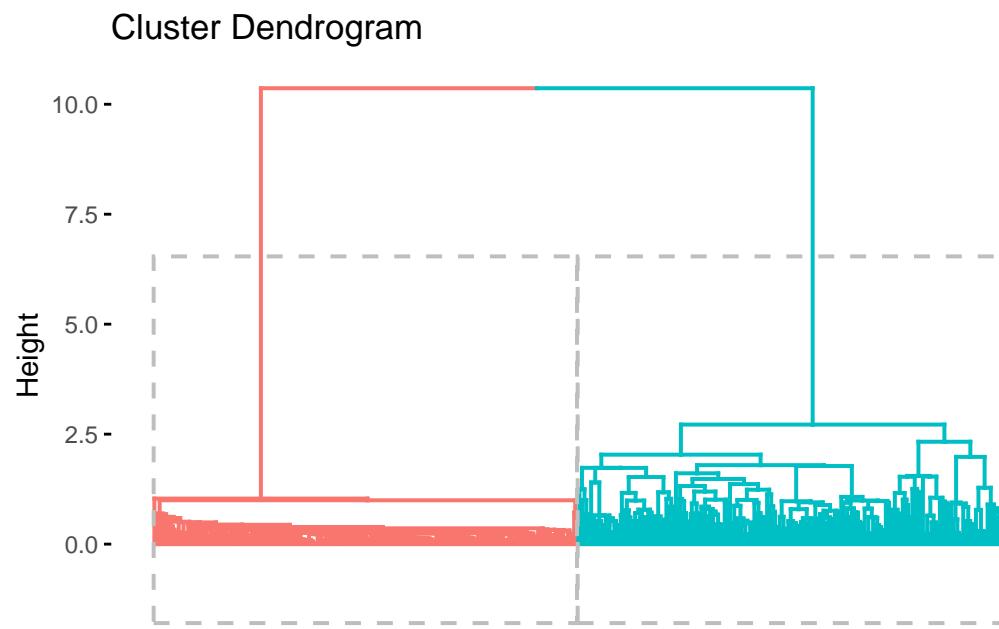
13.2.1 Agglomerative clustering

Start with single data points as “clusters,” then iteratively combine the closest pair of clusters. The closeness may be defined in the following ways:

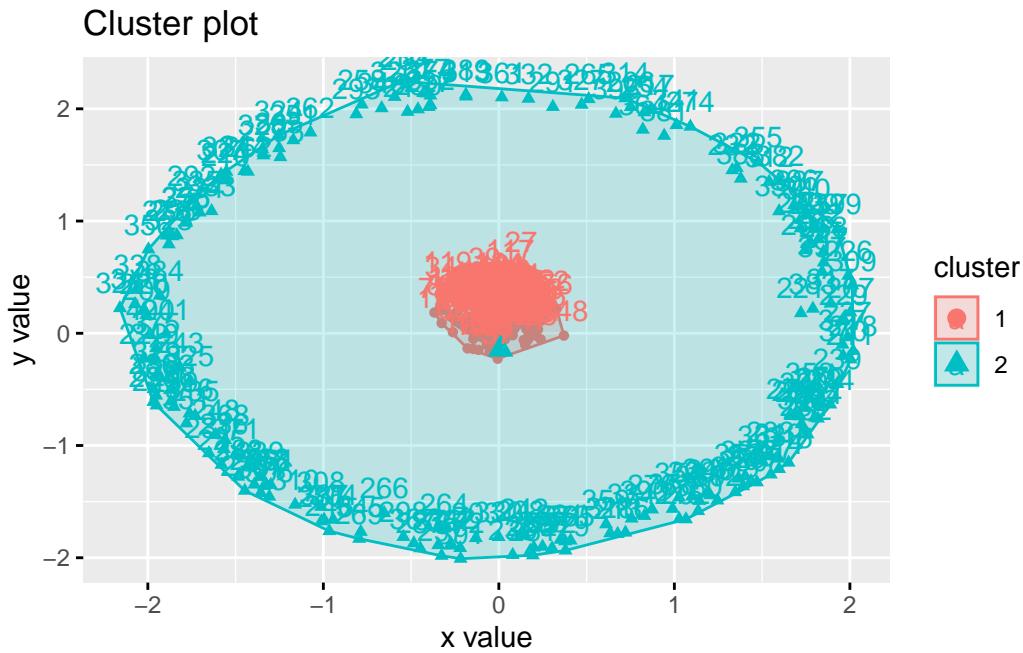
1. Single Linkage: In single linkage, we define the distance between two clusters as the minimum distance between any single data point in the first cluster and any single data point in the second cluster.
2. Complete Linkage: In complete linkage, we define the distance between two clusters to be the maximum distance between any single data point in the first cluster and any single data point in the second cluster.
3. Average Linkage: In average linkage, we define the distance between two clusters to be the average distance between data points in the first cluster and data points in the second cluster.

4. Centroid Method: In centroid method, the distance between two clusters is the distance between the two mean vectors of the clusters.
5. Ward's Method: This method does not directly define a measure of distance between two points or clusters. It is an ANOVA based approach. One-way univariate ANOVAs are done for each variable with groups defined by the clusters at that stage of the process. At each stage, two clusters merge that provide the smallest increase in the combined error sum of squares.

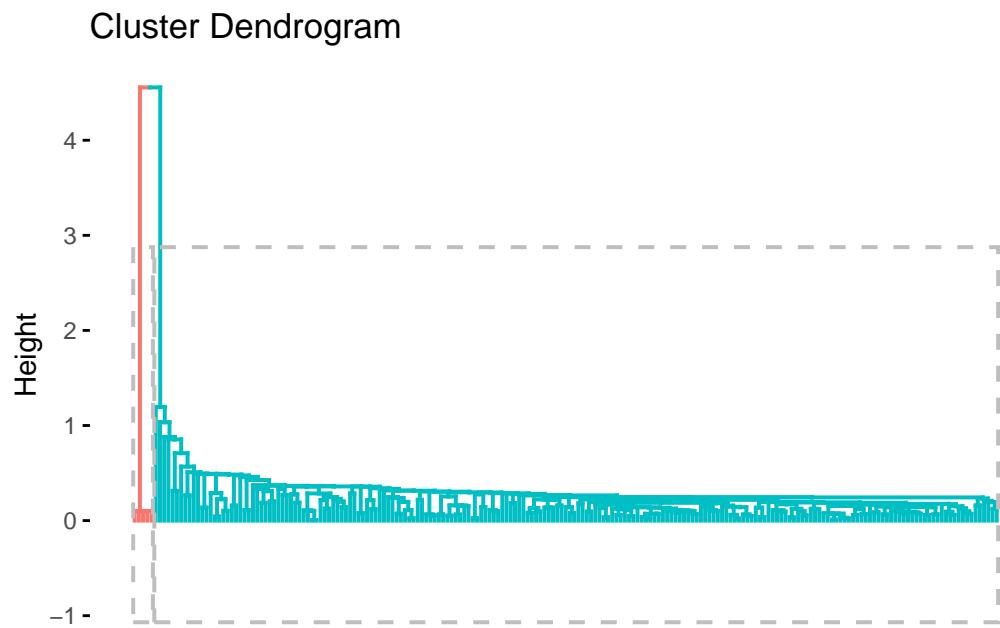
```
# Use hcut() which compute hclust and cut the tree
cir_hc <- hcut(dataset_cir, k = 2, hc_method = "single")
# Visualize dendrogram
fviz_dend(cir_hc, show_labels = FALSE, rect = TRUE)
```



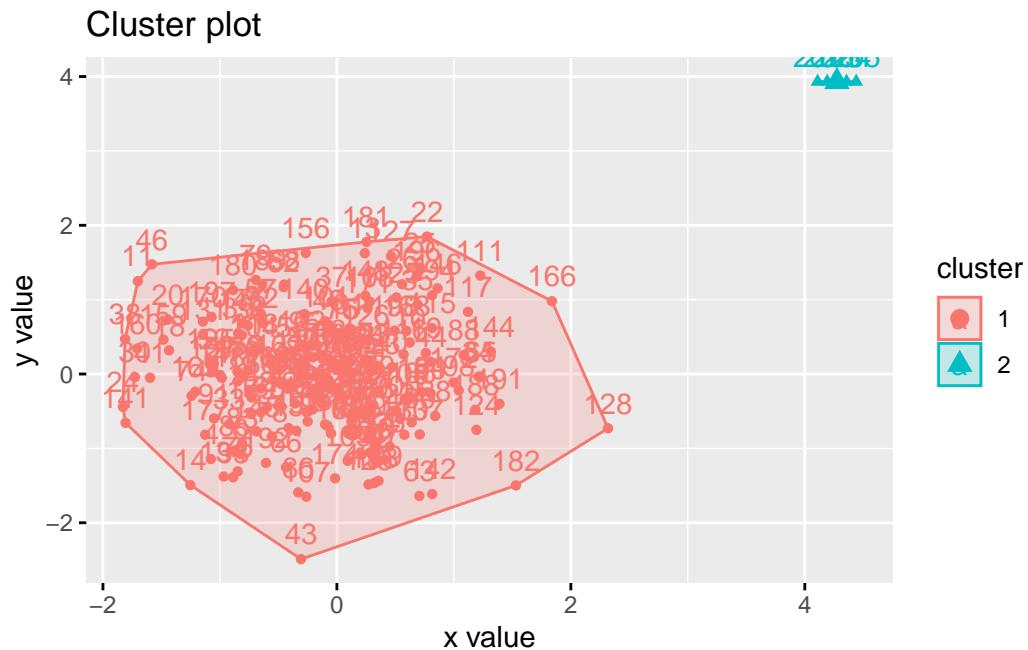
```
# Visualize cluster
fviz_cluster(cir_hc, ellipse.type = "convex")
```



```
# Use hcut() which compute hclust and cut the tree
uneven_hc <- hcut(dataset_uneven, k = 2, hc_method = "single")
# Visualize dendrogram
fviz_dend(uneven_hc, show_labels = FALSE, rect = TRUE)
```



```
# Visualize cluster  
fviz_cluster(uneven_hc, ellipse.type = "convex")
```

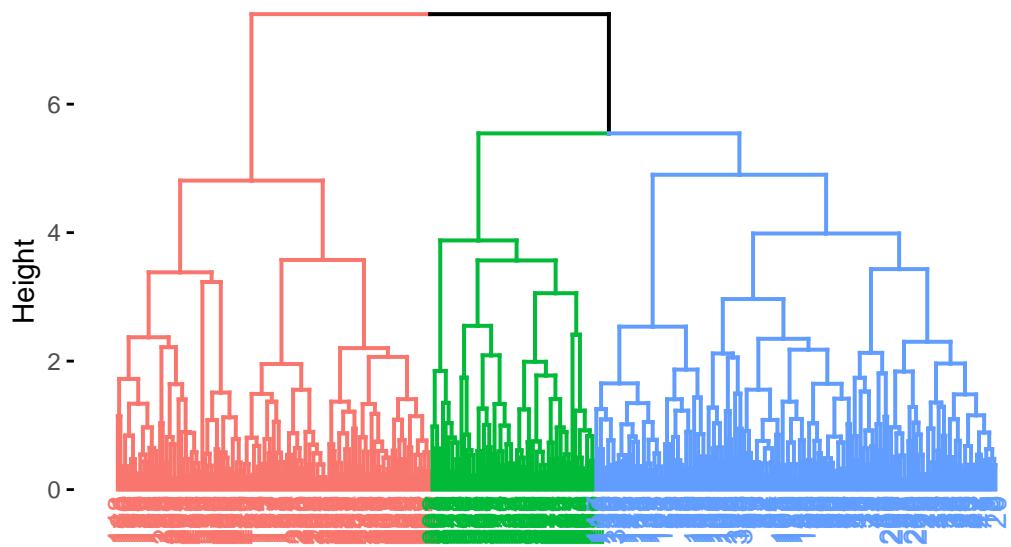


13.2.2 Clustering penguin data using hierarchical methods

Try different methods and see which one generates the best results.

```
# Hierarchical clustering  
# ++++++  
# Use hcut() which compute hcclust and cut the tree  
pen_hc <- hcut(pen_scaled, k = 3, hc_method = "complete")  
# Visualize dendrogram  
fviz_dend(pen_hc)
```

Cluster Dendrogram



```
table(pen_hc$cluster, pen_data$species)
```

	Adelie	Chinstrap	Gentoo
1	145	7	0
2	1	61	0
3	0	0	119

Exercise Try using different clustering methods!

13.3 Clustering analysis and validation

13.3.1 Hopkins statistic

Comparing the mean nearest-neighbor distance between uniformly generated sample points and mean nearest-neighbor distance within the data set.

$$H = 1 - \frac{\sum u_i^d}{\sum u_i^d + \sum w_i^d}$$

This quantifies the “clustering tendency” of the data set.

```
# Check Cluster Tendency--Hopkins Statistic
hopkins(pen_scaled, n = 30) # n should be about 20% of the data
```

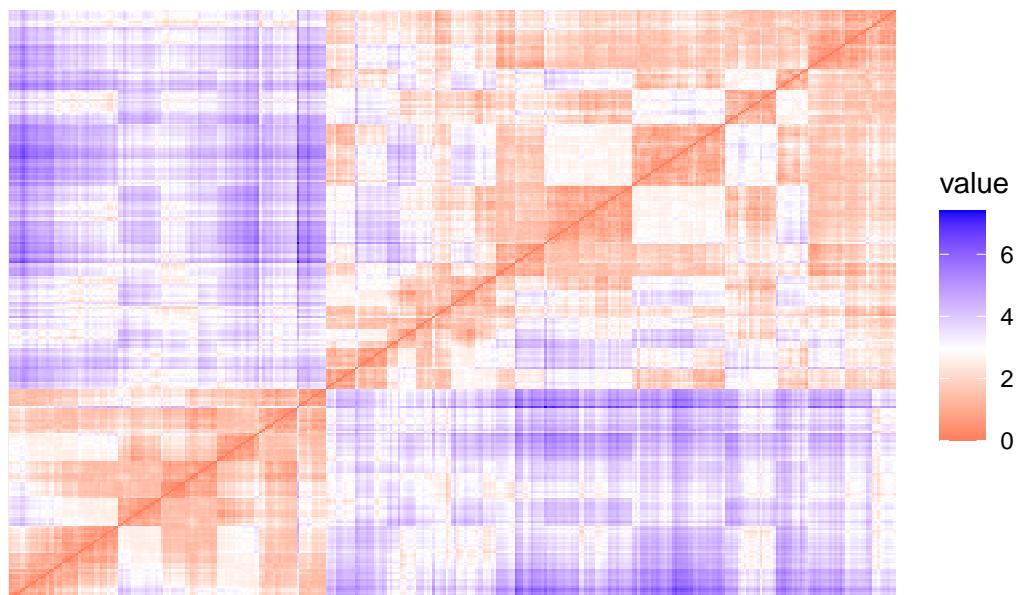
```
$H
[1] 0.2514601
```

```
# run a couple times to sample repeatedly
```

If H is below 0.5 reject the null hypothesis, which is that the data are generated by a Poisson point process (uniformly distributed.)

```
# Visual Assessment of Cluster Tendency
fviz_dist(dist(pen_scaled), show_labels = FALSE)+ labs(title = "Scaled penguin data")
```

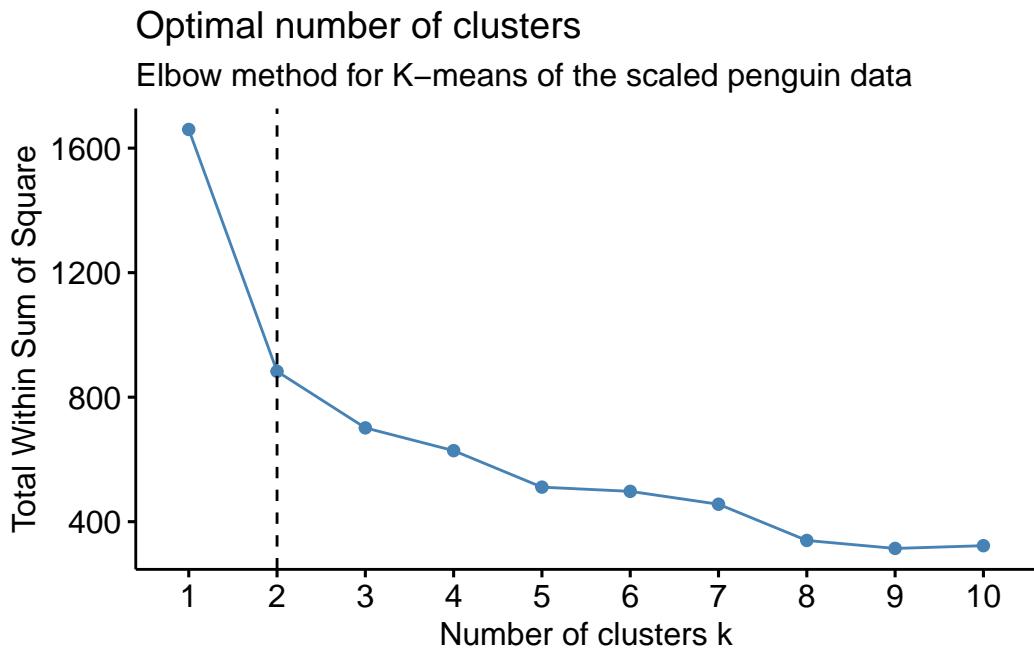
Scaled penguin data



- Red is high similarity (low dissimilarity)
- Blue is low similarity (high dissimilarity)

13.3.2 Elbow method

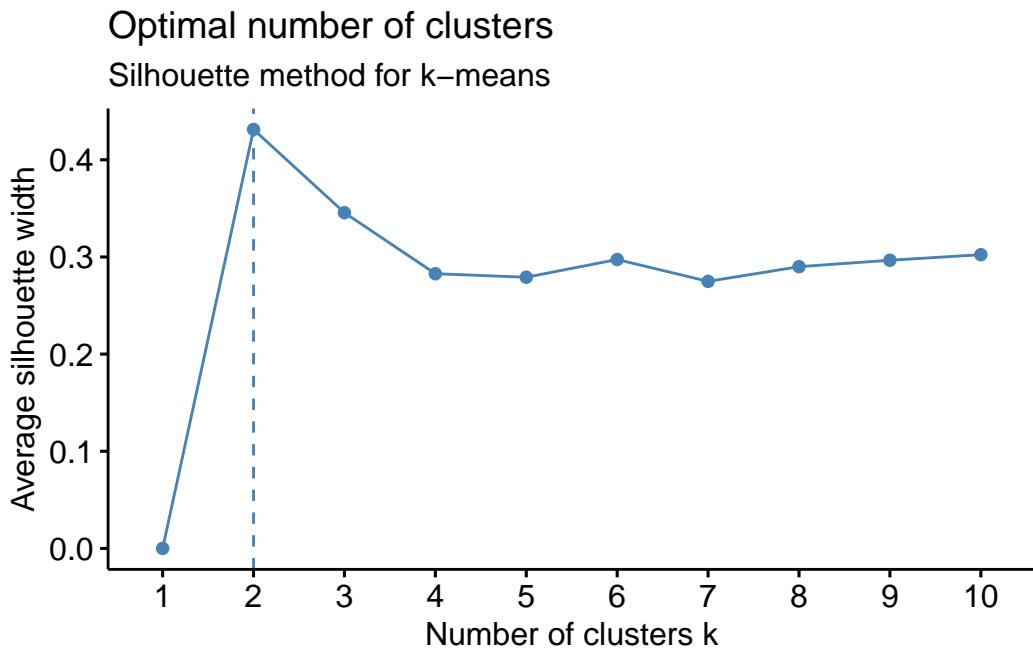
```
# Elbow method  
fviz_nbclust(pen_scaled, kmeans, method = "wss") + geom_vline(xintercept = 2, linetype = 2)  
labs(subtitle = "Elbow method for K-means of the scaled penguin data")
```



13.3.3 Silhouette Plot

Measures how similar an object i is to the other objects in its same cluster versus the objects outside of its cluster; S_i values range from -1 to 1. Close to 1 means very similar to objects in its own group and dissimilar to others

```
# Silhouette method
fviz_nbclust(pen_scaled, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")
```



13.3.4 Lazy way: use all the methods!

```
# not evaluating because it does not run on my computer SA Sept 22 2022
nb <- NbClust(pen_scaled, distance = "euclidean", min.nc = 2,
               max.nc = 10, method = "kmeans")
fviz_nbclust(nb)
```

13.3.5 Validation using bootstrapping

One common approach to validating clustering is to use the approach called bootstrapping which involves repeatedly sampling from the data set, running the clustering algorithm and comparing the results. One algorithm uses the Jaccard coefficient to quantify similarity between sets, which is defined as the number of points in the intersection of the two sets (those which are in both sets), divided by the number of points in the union of the two sets (the point that are in either one or the other set):

$$J = \frac{|A \cap B|}{|A \cup B|}$$

The vertical lines indicate the number of points (cardinality) in the set.

```
k <- 3
cboot.hclust <- clusterboot(pen_scaled, clustermethod=kmeansCBI, k= k)

boot 1
boot 2
boot 3
boot 4
boot 5
boot 6
boot 7
boot 8
boot 9
boot 10
boot 11
boot 12
boot 13
boot 14
boot 15
boot 16
boot 17
boot 18
boot 19
boot 20
boot 21
boot 22
boot 23
boot 24
boot 25
boot 26
boot 27
boot 28
boot 29
boot 30
boot 31
boot 32
boot 33
boot 34
boot 35
boot 36
boot 37
boot 38
boot 39
```

boot 40
boot 41
boot 42
boot 43
boot 44
boot 45
boot 46
boot 47
boot 48
boot 49
boot 50
boot 51
boot 52
boot 53
boot 54
boot 55
boot 56
boot 57
boot 58
boot 59
boot 60
boot 61
boot 62
boot 63
boot 64
boot 65
boot 66
boot 67
boot 68
boot 69
boot 70
boot 71
boot 72
boot 73
boot 74
boot 75
boot 76
boot 77
boot 78
boot 79
boot 80
boot 81
boot 82

```
boot 83
boot 84
boot 85
boot 86
boot 87
boot 88
boot 89
boot 90
boot 91
boot 92
boot 93
boot 94
boot 95
boot 96
boot 97
boot 98
boot 99
boot 100

print(cboot.hclust)

* Cluster stability assessment *
Cluster method: kmeans
Full clustering results are given as parameter result
of the clusterboot object, which also provides further statistics
of the resampling results.
Number of resampling runs: 100

Number of clusters found in data: 3

Clusterwise Jaccard bootstrap (omitting multiple points) mean:
[1] 0.5994643 0.5494741 0.7072403
dissolved:
[1] 33 67 0
recovered:
[1] 23 15 27

#cboot.hclust <- clusterboot(bcdata, clustermethod=hclustCBI,
#                                method="single", k=2)
```

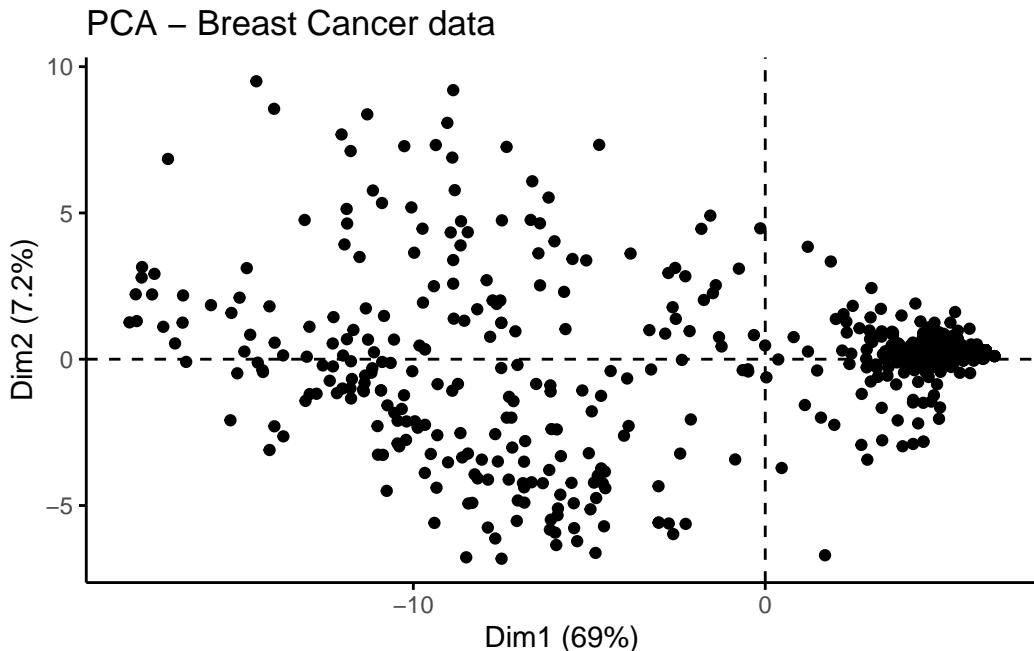
13.4 Application to breast cancer data

The following measurements are based on biopsy data on patients with suspected breast cancer (see [5]). It contains several measurements of cell characteristics, as well as the classification of each biopsy into malignant or benign (2 or 4). Let us see if using clustering

```
# Import Breast Cancer Data Set
fulldata <- read_csv("data/Wisconsin_Breast_Cancers.csv")
bcdata <- fulldata %>% drop_na() %>% dplyr::select(-Sample, -Class)
glimpse(fulldata)

Rows: 684
Columns: 11
$ Sample              <dbl> 1000025, 1002945, 1015425, 1016277, 101702~
$ Clump_Thickness     <dbl> 5, 5, 3, 6, 4, 8, 1, 2, 2, 4, 1, 2, 5, 1, ~
$ Size_Uniformity     <dbl> 1, 4, 1, 8, 1, 10, 1, 1, 1, 2, 1, 1, 3, 1, ~
$ Shape_Uniformity    <dbl> 1, 4, 1, 8, 1, 10, 1, 2, 1, 1, 1, 1, 3, 1, ~
$ Marginal_Adhesion   <dbl> 1, 5, 1, 1, 3, 8, 1, 1, 1, 1, 1, 1, 3, 1, ~
$ Single_Epithelial_Cell_Size <dbl> 2, 7, 2, 3, 2, 7, 2, 2, 2, 2, 1, 2, 2, 2, ~
$ Bare_Nuclei          <dbl> 1, 10, 2, 4, 1, 10, 10, 1, 1, 1, 1, 1, 3, ~
$ Bland_Chromatin      <dbl> 3, 3, 3, 3, 9, 3, 3, 1, 2, 3, 2, 4, 3, ~
$ Normal_Nucleoli      <dbl> 1, 2, 1, 7, 1, 7, 1, 1, 1, 1, 1, 4, 1, ~
$ Mitoses              <dbl> 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, ~
$ Class                <dbl> 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 4, 2, ~

# Visually Inspect Data (PCA)
fviz_pca_ind(prcomp(bcdata), title = "PCA - Breast Cancer data", geom = "point", ggtheme =
```



```
bc_km <- kmeans(scale(bcdata), 2)  
bc_km
```

K-means clustering with 2 clusters of sizes 231, 453

Cluster means:

	Clump_Thickness	Size_Uniformity	Shape_Uniformity	Marginal_Adhesion
1	0.9752406	1.1970884	1.1888401	1.0181299
2	-0.4973081	-0.6104358	-0.6062297	-0.5191788
	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli
1	1.0066757	1.1562984	1.0783707	1.042569
2	-0.5133379	-0.5896356	-0.5498977	-0.531641
	Mitoses			
1	0.6021640			
2	-0.3070638			

Clustering vector:

Within cluster sum of squares by cluster:

[1] 2156.785 573.108
(between_SS / total_SS = 55.6 %)

Available components:

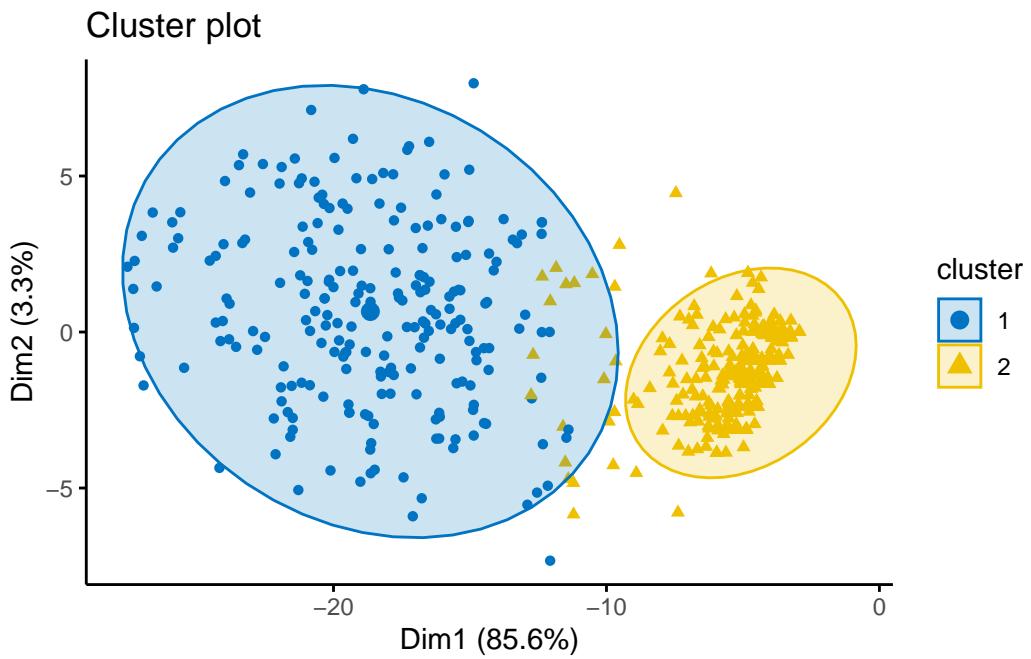
```
[1] "cluster"        "centers"        "totss"          "withinss"        "tot.withinss"  
[6] "betweenss"      "size"           "iter"           "ifault"
```

```
table(bc_km$cluster, fulldata$Class)
```

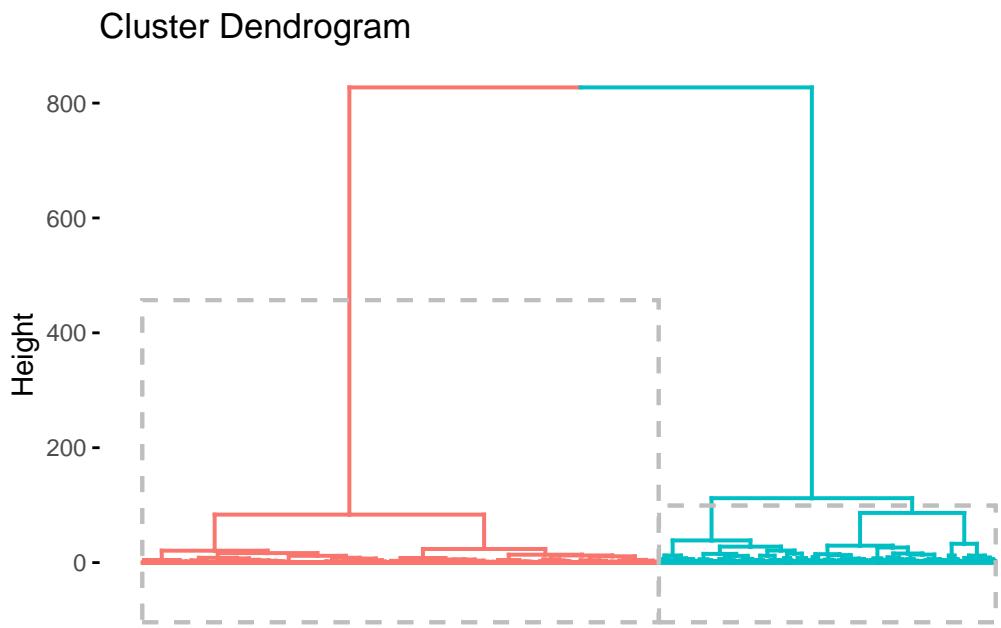
$$\begin{array}{r} 2 \quad 4 \\ 1 \quad 10 \quad 221 \\ 2 \quad 434 \quad 19 \end{array}$$

```
#irisCluster$cluster <- as.factor(irisCluster$cluster)
#ggplot(iris, aes(Petal.Length, Petal.Width, color = iris$cluster)) + geom_point()

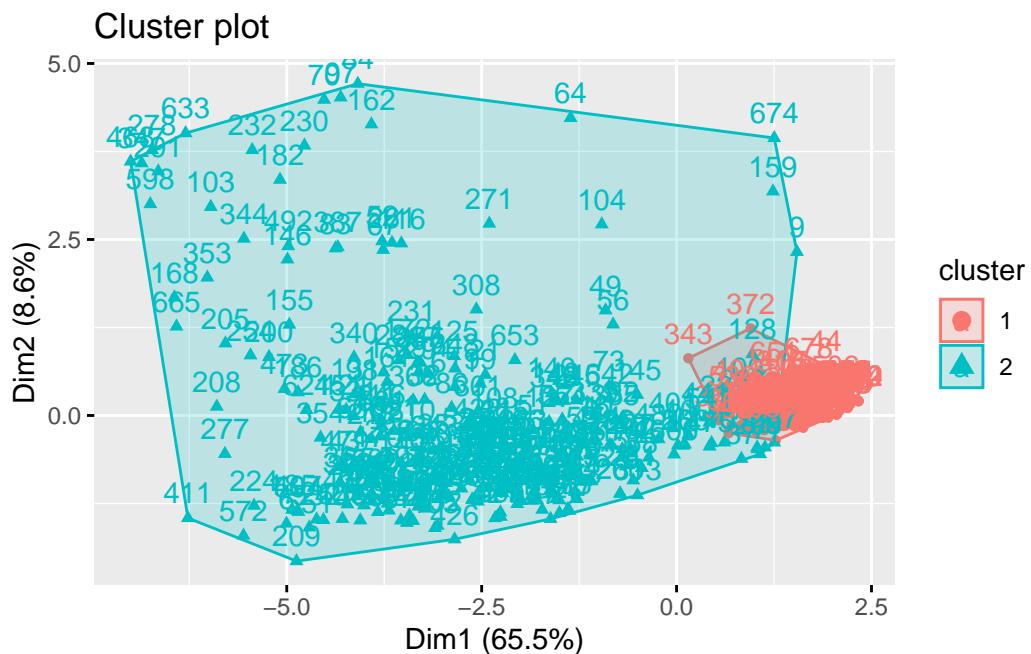
fviz_cluster(list(data = bcdata, cluster = bc_km$cluster),
ellipse.type = "norm", geom = "point", stand = FALSE, palette = "jco", ggtheme = theme_cla
```



```
# Use hcut() which compute hclust and cut the tree
bc_hc <- hcut(scale(bcdata), k = 2, hc_method = "ward")
# Visualize dendrogram
fviz_dend(bc_hc, show_labels = FALSE, rect = TRUE)
```



```
# Visualize cluster  
fviz_cluster(bc_hc, ellipse.type = "convex")
```



```
table(bc_hc$cluster, fulldata$Class)
```

$$\begin{array}{r} & 2 & 4 \\ 1 & 412 & 2 \\ 2 & 32 & 238 \end{array}$$

13.5 References:

1. <https://www.r-bloggers.com/exploring-assumptions-of-k-means-clustering-using-r/>
 2. <https://onlinecourses.science.psu.edu/stat505/node/143/>
 3. <https://github.com/hhundiwala/hierarchical-clustering>
 4. <https://www.r-bloggers.com/bootstrap-evaluation-of-clusters/>
 5. [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

14 Generalized linear models

14.1 Goal

Learn about Generalized Linear Models (GLMs), and be able to decide which model is most appropriate for the problem at hand.

Let's load some packages:

```
library(tidyverse) # our friend the tidyverse

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate  1.9.2     v tidyr    1.3.0
v purrr     1.0.2

-- Conflicts -----
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become non-conflicting
```

```
library(MASS) # negative binom regression
```

Attaching package: 'MASS'

The following object is masked from 'package:dplyr':

```
select
```

14.2 Introduction

The linear regression we've explored during the past weeks attempts to estimate the expected value for **response** (dependent) variable Y given the **predictors** X . It assumes that the response variable changes continuously, and that errors are normally distributed around the mean. In many cases, however:

- the response variable does not have support in the whole real line (e.g., binary, count, only positive values)
- the errors are not normally distributed (e.g., the response variable can take only positive values)
- the variance changes with the mean (heteroscedasticity)

In these cases, you can use **Generalized Linear Models** (GLMs) to fit the data. In the simplest form of GLMs,

- The response variable is modeled by a single-parameter distribution from the exponential family (Gaussian, Gamma, Binomial, Poisson, etc.)
- A **link function** linearizes the relationship between the fitted values and the predictors.
- Parameters are estimated through a least squares algorithm.

14.2.1 Model structure

In practice, we need to determine three parts of the model:

- **Random component** the entries of the response variable (Y) are assumed to be independently drawn from a certain distribution (e.g., Binomial)—typically a distribution that can be modeled using a single parameter.
- **Systematic component** the explanatory variables (X_1, X_2, \dots) are combined linearly to form a **linear predictor** (e.g., $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots$). The explanatory variables can be continuous, categorical, or mixed.
- **Link function** $g(u)$ specifies how the random and systematic components are connected.

14.3 Binary data

The most extreme case of departure from normality is when the response variable can assume only values 0 or 1 (no/yes, survived/deceased, lost/won, etc.). A Bernoulli random variable can take values 0 or 1, and therefore provides the **Random component** of the model:

$$P(Y_i = y_i | \pi_i) = \pi_i^{y_i} (1 - \pi_i)^{1-y_i}$$

Saying that the probability $P(Y_i = 1) = \pi_i$, and $P(Y_i = 0) = 1 - \pi_i$. Now we want to relate the parameter π_i to the **linear predictor** (i.e., choose a link function). This can be accomplished in a number of ways.

14.3.1 Logistic regression

The most popular choice is to use the **Logit** function as the **link function**:

$$\text{Logit}(\pi_i) = \beta_0 + \beta_1 x_i$$

where the function can be written as:

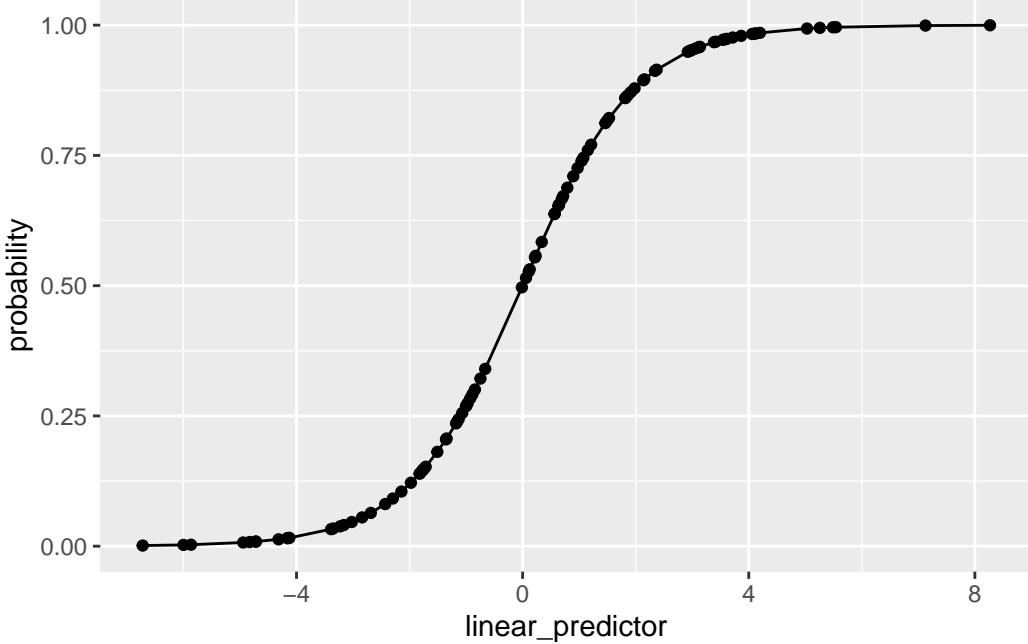
$$\text{Logit}(\pi_i) = \log\left(\frac{\pi_i}{1 - \pi_i}\right) = \log(\pi_i) - \log(1 - \pi_i)$$

Practically, this means that

$$\pi_i = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}} = 1 - \frac{1}{1 + e^{\beta_0 + \beta_1 x_i}}$$

Clearly, when $\beta_0 + \beta_1 x_i = 0$, the probability $\pi_i = 1/2$, while the probability tends to 1 when $(\beta_0 + \beta_1 x_i) \rightarrow \infty$ and to zero when $(\beta_0 + \beta_1 x_i) \rightarrow -\infty$.

```
# some random data
X <- rnorm(100)
beta_0 <- 0.35
beta_1 <- -3.2
linear_predictor <- beta_0 + beta_1 * X
predicted_pi_i <- exp(linear_predictor) / (1 + exp(linear_predictor))
ggplot(data = tibble(linear_predictor = linear_predictor, probability = predicted_pi_i)) +
  aes(x = linear_predictor, y = probability) +
  geom_point() + geom_line()
```



As you can see, this is a logistic curve, hence the name. The parameters β_0 and β_1 control the location of the inflection point and the steepness of the curve, allowing you to model binary response variables (and, with a slight abuse of the error structure, proportions or probabilities).

Other choices of link functions are possible. For example, in economics the *probit* function is preferred:

$$\text{Probit}(\pi_i) = \beta_0 + \beta_1 x_i$$

where

$$\text{Probit}(\pi_i) = \Phi(\pi_i)$$

and $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution:

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^z e^{-t^2/2} dt$$

Clearly, you could alternatively use the cumulative distribution function of any distribution that has support on the real line.

14.3.2 A simple example

We want to know whether being in first, second and third class, as well as gender (women and women first!) influenced the probability of survival in the Titanic disaster. We start with a null model (all passengers have the same probability of survival):

```
library(titanic)
# model 0: probability of survival in general
# regress against an intercept
model0 <- glm(Survived ~ 1, # only intercept
               data = titanic_train,
               family = "binomial") # logistic regression
summary(model0)
```

```
Call:
glm(formula = Survived ~ 1, family = "binomial", data = titanic_train)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.47329    0.06889   -6.87  6.4e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 1186.7 on 890 degrees of freedom
Residual deviance: 1186.7 on 890 degrees of freedom
AIC: 1188.7

Number of Fisher Scoring iterations: 4

# the best fitting (alpha) intercept should lead to
# e^alpha / (1 + e^alpha) = mean(Survived)
mean(titanic_train$Survived)

[1] 0.3838384

exp(model0$coefficients) / (1 + exp(model0$coefficients))
```

```
(Intercept)
0.3838384
```

Now let's include gender:

```
model1 <- glm(Survived ~ Sex, # one sex as baseline, the other modifies intercept
                data = titanic_train,
                family = "binomial")
summary(model1)
```

```
Call:
glm(formula = Survived ~ Sex, family = "binomial", data = titanic_train)

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) 1.0566    0.1290   8.191 2.58e-16 ***
Sexmale     -2.5137    0.1672 -15.036 < 2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 1186.7 on 890 degrees of freedom
Residual deviance: 917.8 on 889 degrees of freedom
AIC: 921.8

Number of Fisher Scoring iterations: 4
```

What is the best-fitting probability of survival for male/female?

```
coeffs <- model1$coefficients
# prob women
as.numeric(1 - 1 / (1 + exp(coeffs[1])))

[1] 0.7420382

# prob men
as.numeric(1 - 1 / (1 + exp(coeffs[1] + coeffs[2])))
```

```
[1] 0.1889081
```

Now let's see whether we can explain better the data using the class:

```
model2 <- glm(Survived ~ Sex + factor(Pclass), # combine Sex and Pclass
                data = titanic_train,
                family = "binomial")
summary(model2)
```

Call:

```
glm(formula = Survived ~ Sex + factor(Pclass), family = "binomial",
     data = titanic_train)
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	2.2971	0.2190	10.490	< 2e-16 ***
Sexmale	-2.6419	0.1841	-14.351	< 2e-16 ***
factor(Pclass)2	-0.8380	0.2447	-3.424	0.000618 ***
factor(Pclass)3	-1.9055	0.2141	-8.898	< 2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 1186.66 on 890 degrees of freedom
Residual deviance: 826.89 on 887 degrees of freedom
AIC: 834.89

Number of Fisher Scoring iterations: 4

A woman in first class would have survival probability:

```
coeffs <- model2$coefficients
# prob women first class
as.numeric(1 - 1 / (1 + exp(coeffs[1])))
```

```
[1] 0.9086385
```

While a man in third class:

```
as.numeric(1 - 1 / (1 + exp(coeffs[1] + coeffs[2] + coeffs[4])))
```

```
[1] 0.09532814
```

Consider the alternative models `Survived ~ Sex * factor(Pclass)`, `Survived ~ Sex + Pclass`, `Survived ~ Sex * Pclass`, `Survived ~ Sex:factor(Pclass)`, `Survived ~ Sex:Pclass`. Explain what each model is doing in English.

14.3.3 Exercise in class: College admissions

With slight abuse of notation, you can fit probabilities using the logistic regression (the only problem is that you don't know how many values contributed to the calculations of the probabilities—i.e., sample sizes). Read in the file `admission_rates.csv`, containing data on admissions to several universities. Your goal is to find a good prediction (or a good combination of predictors) for the `Admission_rate`. You can use `State`, `Ownership` (public/private), `Citytype` (town, suburb, city), `SAT` (typical SAT score of admits), `AvgCost` (tuition). Fit the models using:

```
dt <- read_csv("data/admission_rates.csv")
```

```
Rows: 195 Columns: 7
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (4): Name, State, Ownership, Citytype
```

```
dbl (3): SAT, AvgCost, Admission_rate
```

```
i Use `spec()` to retrieve the full column specification for this data.
```

```
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# example
```

```
logit_1 <- glm(Admission_rate ~ AvgCost, data = dt, family = "binomial")
```

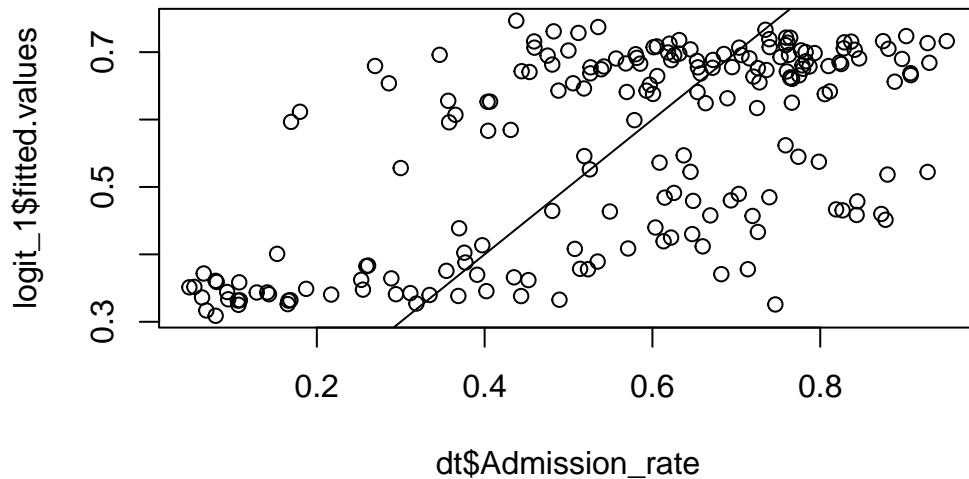
```
Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

(do not worry about the warning `non-integer #successes in a binomial glm!`).

- Plot fitted vs. observed admission rates, when using different combinations of predictors.

For the example above:

```
plot(dt$Admission_rate, logit_1$fitted.values)
abline(c(0,1))
```



- Score the models using AIC: which is the single best predictor of acceptance rate? (Note: as we will see later this week, the lower the AIC, the better).

```
AIC(logit_1)
```

[1] 220.7783

- Which the best combination of two predictors?

14.4 Count data

14.4.1 Poisson regression

Suppose your response variables are non-negative integers. For example, we are counting the number of eggs females lay as a function of their age, body size, etc. A possible model for this case is to think of the response variable as being sampled from a Poisson distribution:

$$Y_i \sim \text{Pois}(\lambda_i)$$

and that the logarithm of the parameter λ_i depends linearly on the predictors:

$$\mathbb{E}[\lambda_i] = \mathbb{E}[\log(Y_i|X_i)] = \beta_0 + \beta_1 X_i$$

In this case, our *link function* is the logarithm, transforming the relationship between the fitted values and the predictors into a linear regression.

14.4.2 Exercise in class: Number of genomes

The file `data/genomes.csv` contains the year in which the genome of a given animal was published. The file `sequence_cost.csv` the estimated cost per sequencing a Mb in a given year.

- Count the number of genomes published per year (store the value as `n`) and store it in the tibble `num_genomes` along with the values `Year` and `Dollars_per_Mb` (note: you need to use `inner_join` to pull this off);
- Fit the number of genomes published in a given year:
 - using only an intercept (your predictions should match the mean) (Code: `pois_1 <- glm(n ~ 1, data = num_genomes, family = "poisson")`)
 - using the year as a predictor
 - using the cost of sequencing as a predictor
- For each model, plot the observed `n` vs its predicted value, and compute AIC. Is the fit superior when we use `Year` or `Dollars_per_Mb`?

14.4.3 Underdispersed and Overdispersed data

The main feature of the Poisson distribution is that the mean and the variance are both equal to λ . You might remember (Taylor expansion) that:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Then, for X sampled from a Poisson distribution:

$$\begin{aligned} \mathbb{E}[X] &= \sum_{x=0}^{\infty} xP(X=x) \\ &= \sum_{x=0}^{\infty} xe^{-\lambda} \frac{\lambda^x}{x!} \\ &= \lambda e^{-\lambda} \sum_{(x-1)=0}^{\infty} \frac{\lambda^{(x-1)}}{(x-1)!} \\ &= \lambda e^{-\lambda} e^{\lambda} \\ &= \lambda \end{aligned}$$

Similarly, using

$$\begin{aligned}\mathbb{V}[X] &= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \\ &= \left(\sum_{x=0}^{\infty} x^2 e^{-\lambda} \frac{\lambda^x}{x!} \right) - \lambda^2 \\ &= \dots \\ &= \lambda\end{aligned}$$

The fact that the variance equals the mean is a hard constraint, rarely matched by real data. When you encounter **over-dispersion** (i.e., the variance in the data is much larger than what assumed by Poisson), you need to choose a different model. This happens very often, and the main solution to use is a **Negative Binomial Regression** (a negative binomial distribution can be thought of as a Poisson with a scaled variance). In practice, this amounts to fitting:

$$\mathbb{E}[\lambda_i] = \beta_0 + \beta_1 X_i$$

and

$$\mathbb{E}[\lambda_i^2] - \mathbb{E}[\lambda_i]^2 = \mathbb{V}[\lambda_i] = \phi \lambda_i$$

Where ϕ controls the dispersion of the data. A value $\phi > 1$ signals over-dispersion, while (the very rare case of) $\phi < 1$ under-dispersion. The Poisson regression is appropriate only when $\phi \approx 1$. A simple way to test for dispersion is to fit a **quasipoisson** model, which returns a dispersion parameter (anything larger than 1 means over-dispersion).

14.4.4 Exercise in class: Number of genomes

- For the models above, change the family to **quasipoisson** to check the dispersion (e.g., `qpois_1 <- glm(n ~ 1, data = num_genomes, family = "quasipoisson")`).
- Do you have over-dispersion?
- If the data are over-dispersed, fit them again using **glm.nb** (a negative binomial regression model provided by the package **MASS**).

14.4.5 Separate distribution for the zeros

In several biologically-relevant cases, we have an excess of zeros. For example, you might have animals, that, if they reach the age of 1, will go on to a live a number of years—say well-described by a Poisson distribution. However, mortality immediately after birth is high. In such cases, you can use zero-inflated or zero-hurdle models.

In zero-inflated models, you can think of having a conditional branching: with probability p_z your count is zero; if not (prob. $1 - p_z$) it is sampled from a given distribution. As such a count of zero can stem from two different processes: either because you got a zero at the first step, or because you have sampled a zero from the distribution.

Zero-hurdle models are slightly different: you first decide whether you're going to have a zero; if not, you sample your data from a truncated distribution, such that you cannot sample a zero from this second source.

Zero-inflated and zero-hurdle models are examples of **mixture models**.

14.5 Other GLMs

Historically, GLMs have been defined for the canonical families:

- Gaussian: linear regression
- Gamma and Inverse Gaussian: Positive, continuous
- Poisson: count data
- Negative Binomial: count data (fit an ancillary parameter for over-dispersion)
- Binary/Binomial (logistic): binary responses; number of successes; probabilities/proportions (with slight abuse).

However, the same basic idea led to the development of “non-canonical” GLMs:

- Log-normal: Positive, continuous
- Log-gamma: survival models
- Probit: binary

and many others. Fitting the models can be done using Maximum Likelihoods, or in a Bayesian framework (typically, through MCMC).

14.6 Readings and homework

- There are two useful swirls in the course [Regression Models: Binary Outcomes](#) and [Count Outcomes](#)
- [An excellent book on GLMs in R](#)
- [Regression Models for Count Data in R](#)

15 Machine learning methods for classification

15.1 Introduction

The classification problem is a very common one in practice, and we have already seen the use of GLMs to systematically predict binary response variables. We have also used clustering to perform *unsupervised* learning, where we do not have any information about correct labels for data points. We now turn to *supervised* classification problems and introduce two different approaches: a Bayesian one and a tree-based one.

15.2 Naive Bayes classifier

Suppose that we wish to classify an observation into one of K classes, which means there is a response variable Y can take on K different values, or labels. Let π_k be the *prior probability* that a randomly chosen observation comes from the k-th class. Let $f_k(X) = \Pr(X|Y = k)$ be the density function of X for an observation that comes from the k-th class.

Assuming we have the prior probabilities and the conditional probability distributions of the observations within each category k , we can use Bayes' theorem to compute the probability of each class, given a set of observations x by turning around the conditionality:

$$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_i^K \pi_i f_i(x)}$$

And let us use the notation $p_k(x) = P(Y = k|X = x)$ to mean the *posterior probability* that an observation x belongs to class k .

Let us use the penguin data as an example, where we want to classify the observations by species. Then, if we take a training set with known classifications, we can take the prior probabilities π_k to be the fractions of observed birds of each species, and the probability distributions of each explanatory variable for each species $f_k(X)$ can be estimated from the observed distributions of the explanatory variables (flipper lengths, etc.) for Adelie, Gentoo, and Chinstrap subsets of observations.

The difficult part in the above example is estimating the distributions $f_k(X)$, which is especially challenging for joint distributions of multiple variables. One method, called Linear

Discriminant Analysis, assumes the distributions have the same covariance matrices for all classes and only differ in their mean values. Another, called Quadratic Discriminant Analysis, assumes different covariance matrices for different classes.

The Naive Bayes classifier instead assumes that within each class the explanatory variables X_i are independent, and thus

$$f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times \dots \times f_{kn}(x_n)$$

where $f_{ki}(x_i)$ is the probability distribution of the i-th explanatory variable x_i for class k .

This modifies the Bayes' formula to look like this:

$$P(Y = k|X = x) = \frac{\pi_k f_{k1}(x_1) \times f_{k2}(x_2) \times \dots \times f_{kn}(x_n)}{\sum_i^K \pi_i f_{k1}(x_1) \times f_{k2}(x_2) \times \dots \times f_{kn}(x_n)}$$

Although it looks more complicated, we can compute each distribution function separately, so as long as there is enough data in the training set to estimate each explanatory variable for each class, the calculation is manageable.

15.2.1 Penguin data

```
data("penguins")
pen_clean <- penguins %>% drop_na()
# Fix the random numbers by setting the seed for reproducibility
#set.seed(314)
# Put 3/4 of the data into the training set
pen_split <- initial_split(pen_clean, prop = 3/4)

# Create data frames for the two sets:
pen_train <- training(pen_split)
pen_test <- testing(pen_split)

nb_spec <- naive_Bayes() %>%
  set_mode("classification") %>%
  set_engine("naivebayes") %>%
  set_args(usekernel = FALSE)

pen_recipe <-
  recipe(species ~ ., data = pen_train) %>%
```

```

update_role(island,  new_role = "ID")

#nb_fit <- nb_spec %>%
#  fit(Direction ~ Lag1 + Lag2, data = pen_)

pen_workflow_nb <- workflow() %>%
  add_model(nb_spec) %>%
  add_recipe(pen_recipe)

fit_nb <- pen_workflow_nb %>% fit(pen_train)

compare_pred <- augment(fit_nb, new_data = pen_test)

compare_pred %>% conf_mat(truth = species, estimate = .pred_class)

  Truth
Prediction Adelie Chinstrap Gentoo
  Adelie      28       3      0
  Chinstrap    1      15      0
  Gentoo       0       0     37

compare_pred %>% accuracy(truth = species, estimate = .pred_class)

# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>        <dbl>
1 accuracy  multiclass  0.952

```

15.2.2 Breast cancer data

```

data("breastcancer")
glimpse(breastcancer)

Rows: 699
Columns: 10
$ `Clump Thickness`          <int> 5, 5, 3, 6, 4, 8, 1, 2, 2, 4, 1, 2, 5, 1~
$ `Uniformity of Cell Size` <int> 1, 4, 1, 8, 1, 10, 1, 1, 1, 2, 1, 1, 3, ~

```

```

$ `Uniformity of Cell Shape`      <int> 1, 4, 1, 8, 1, 10, 1, 2, 1, 1, 1, 1, 3, ~
$ `Marginal Adhesion`            <int> 1, 5, 1, 1, 3, 8, 1, 1, 1, 1, 1, 1, 3, 1~
$ `Single Epithelial Cell Size` <int> 2, 7, 2, 3, 2, 7, 2, 2, 2, 2, 1, 2, 2, 2~
$ `Bare Nuclei`                 <int> 1, 10, 2, 4, 1, 10, 10, 1, 1, 1, 1, 1, 3~
$ `Bland Chromatin`             <int> 3, 3, 3, 3, 9, 3, 3, 1, 2, 3, 2, 4, 3~
$ `Normal Nucleoli`             <int> 1, 2, 1, 7, 1, 7, 1, 1, 1, 1, 1, 4, 1~
$ Mitoses                        <int> 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1~
$ Class                           <fct> benign, benign, benign, benign, benign, ~

cancer_clean <- breastcancer %>% drop_na()
# Fix the random numbers by setting the seed for reproducibility
#set.seed(314)
# Put 3/4 of the data into the training set
can_split <- initial_split(cancer_clean, prop = 1/2)

# Create data frames for the two sets:
can_train <- training(can_split)
can_test  <- testing(can_split)

nb_spec <- naive_Bayes() %>%
  set_mode("classification") %>%
  set_engine("naivebayes") %>%
  set_args(usekernel = FALSE)

can_recipe <-
  recipe(Class ~ ., data = can_train) %>%
  update_role(`Uniformity of Cell Shape`, `Uniformity of Cell Size`, `Bland Chromatin`, m

can_workflow_nb <- workflow() %>%
  add_model(nb_spec) %>%
  add_recipe(can_recipe)

can_fit_nb <- can_workflow_nb %>% fit(can_train)

compare_pred <- augment(can_fit_nb, new_data = can_test)

compare_pred %>% conf_mat(truth = Class, estimate = .pred_class)

```

Truth

```

Prediction benign malignant
benign      211       5
malignant    11      115

compare_pred %>%  accuracy(truth = Class, estimate = .pred_class)

# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>        <dbl>
1 accuracy  binary      0.953

```

15.3 Decision Trees

Suppose instead that we represent the classification process as a sequence of binary choices, that eventually lead to a category. This can be represented by a *decision tree*, whose *internal nodes* are separators of the space of observations (all the values of explanatory variables) that divide it into regions, and the *leaves* are the labels of these regions. (Decision trees can also be used for quantitative response variables, but we will focus on classification.) For example, here is a decision tree for accepting a job offer:

Building a decision tree for classification happens by sequential splitting the space of observations, starting with the decision that gives the most bang for the buck. Let us define the quality of the split into M region by calculating how many observations in that regions actually belong to each category k . The *Gini index* (or impurity) is defined as the product of the probability of an observation (in practice, the fraction of observations in the training set) being labeled correctly with label k (p_k) times the probability of it being labeled incorrectly ($1 - p_k$), summed over all the labels k :

$$G = \sum_k (1 - p_k)p_k$$

Alternatively, one can use the Shannon information or entropy measure:

$$S = - \sum_k p_k \log(p_k)$$

Notice that both measures are smallest when p_k is close to 1 or 0, so they both tell the same story for a particular region: if (almost) all the points are points are either classified or incorrectly, these measures are close to 0.

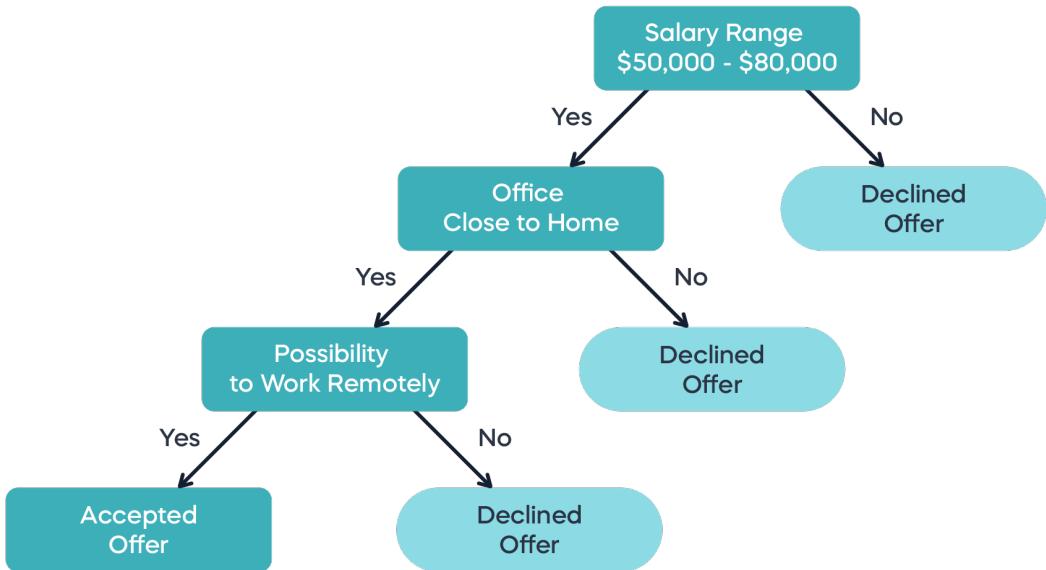


Figure 15.1: Should you take this job? (from <https://365datascience.com/tutorials/machine-learning-tutorials/decision-trees>)

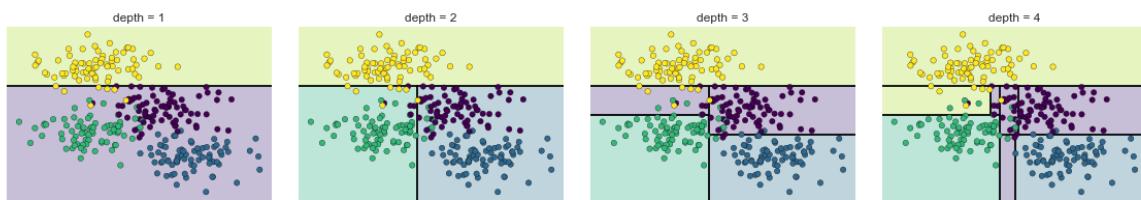


Figure 15.2: Recursive splitting of a two-dimensional set of observations (from <https://jakevdp.github.io/PythonDataScienceHandbook/>)

One of these measures is used to create the sequential splits in the training data set. The biggest problem with this decision tree method is that it's a greedy algorithm that easily leads to overfitting: as you can see in the figure above, it can create really complicated regions in the observation space that may not correspond to meaningful distinctions.

15.3.1 Penguin data

```
tree_spec <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")

pen_recipe <-
  recipe(species ~ ., data = pen_train) %>%
  update_role(island, new_role = "ID")

pen_workflow_tree <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(pen_recipe)

fit_tree <- pen_workflow_tree %>% fit(pen_train)

compare_pred <- augment(fit_tree, new_data = pen_test)

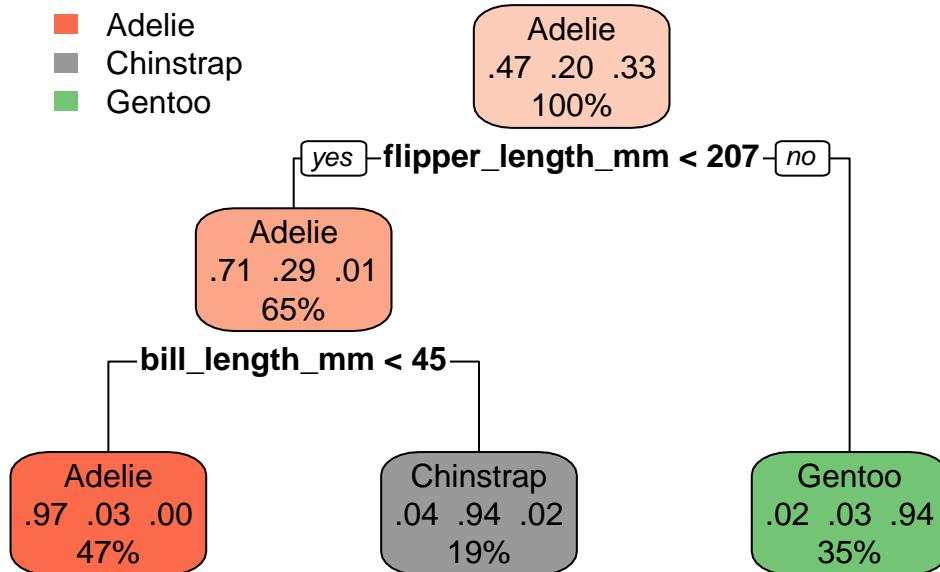
compare_pred %>% conf_mat(truth = species, estimate = .pred_class)

  Truth
Prediction Adelie Chinstrap Gentoo
  Adelie      28        3       0
  Chinstrap     1       13       0
  Gentoo       0        2      37

compare_pred %>% accuracy(truth = species, estimate = .pred_class)

# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>    <chr>        <dbl>
1 accuracy  multiclass  0.929
```

```
fit_tree %>%
  extract_fit_engine() %>%
  rpart.plot()
```



15.3.2 Breast cancer data

```
can_recipe <-
  recipe(Class ~ ., data = can_train) %>%
  update_role(`Uniformity of Cell Shape`, `Uniformity of Cell Size`, `Bland Chromatin`, ...)

can_workflow_tree <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(can_recipe)

fit_tree <- can_workflow_tree %>% fit(can_train)

compare_pred <- augment(fit_tree, new_data = can_test)

compare_pred %>% conf_mat(truth = Class, estimate = .pred_class)
```

Truth

```

Prediction  benign malignant
benign       208        4
malignant     14       116

```

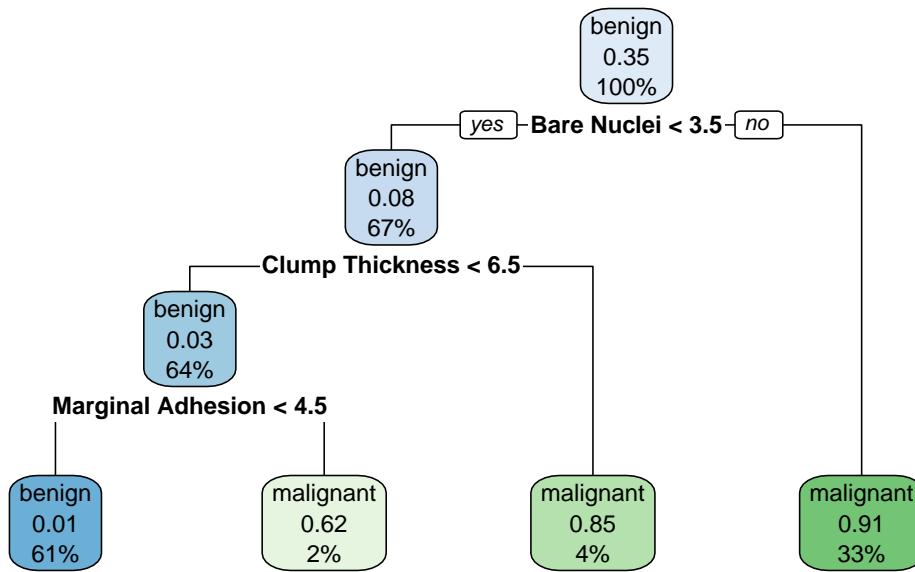
```
compare_pred %>% accuracy(truth = Class, estimate = .pred_class)
```

```

# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>    <chr>      <dbl>
1 accuracy  binary      0.947

```

```
fit_tree %>%
  extract_fit_engine() %>%
  rpart.plot()
```



15.4 Random Forests

Overfitting usually results from modeling meaningless noise in the data instead of real differences. This gave rise to the idea to “shake up” the algorithm and see if the splits it produces are robust if the training set is different. In fact, let’s use multiple trees and look at what the consensus of the *ensemble* can produce. This approach is called *bagging*, which makes use of an random ensemble of parallel classifiers, each of which over-fits the data, it combines the

results to find a better classification. An ensemble of randomized decision trees is known as a *random forest*.

Essentially, the process is as follows: use random sampling from the data set (bootstrapping) to generate different training sets and train different decision trees on each. Then for each observation, find its consensus classification among the whole ensemble; that is, how does the plurality of the trees classify it.

Since each data point is left out of a number of trees, one can estimate an unbiased error of classification by computing the “out-of-bag” error: for each observation, used the classification produced by all the trees that did not have this one points in the bag. This is basically a built-in cross-validation measure.

15.4.1 Penguin data

```
rf_spec <- rand_forest(mtry = 4) %>%
  set_engine("ranger") %>%
  set_mode("classification")

rf_recipe <-
  recipe(species ~ ., data = pen_clean) %>%
  update_role(island, new_role = "ID")

pen_workflow_rf <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(pen_recipe)

fit_rf <- pen_workflow_rf %>% fit(pen_clean)

compare_pred <- augment(fit_rf, new_data = pen_clean)

compare_pred %>% conf_mat(truth = species, estimate = .pred_class)
```

		Truth		
Prediction	Adelie	Chinstrap	Gentoo	
Adelie	144	1	0	
Chinstrap	2	67	0	
Gentoo	0	0	119	

```
compare_pred %>% accuracy(truth = species, estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>        <dbl>
1 accuracy multiclass     0.991
```

Importance measures of different variables:

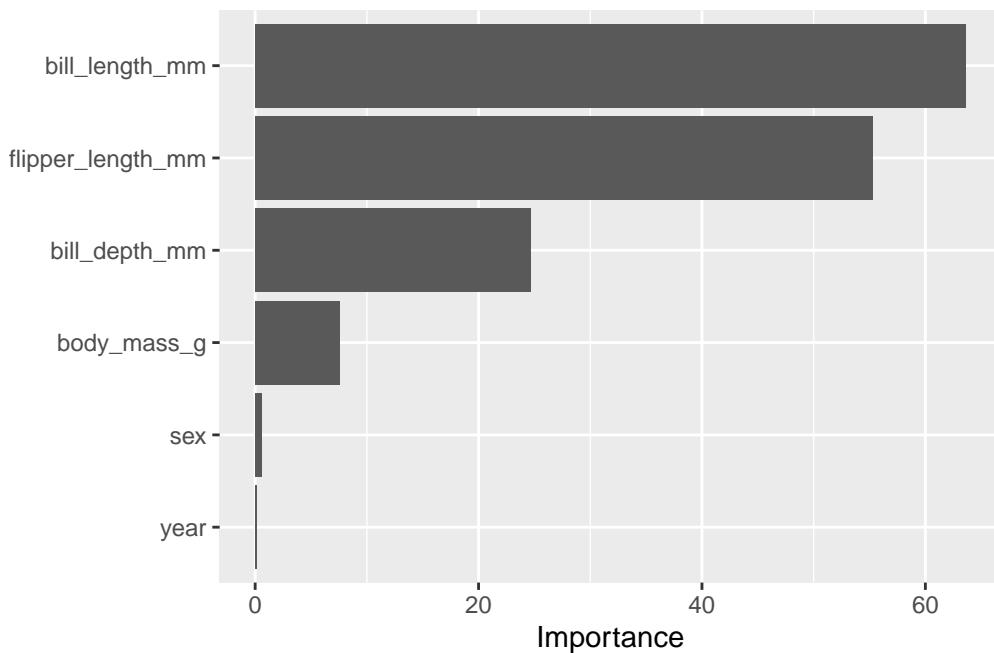
```
last_rf <-
  rand_forest(mtry = 4, trees = 100) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
# the last workflow
last_workflow <-
  pen_workflow_rf %>%
  update_model(last_rf)

# the last fit
set.seed(3)
last_rf_fit <-
  last_workflow %>%
  last_fit(pen_split)

last_rf_fit %>%
  collect_metrics()

# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>        <dbl> <chr>
1 accuracy multiclass     0.976 Preprocessor1_Model1
2 roc_auc   hand_till      0.999 Preprocessor1_Model1

last_rf_fit %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip(num_features = 20)
```



15.4.2 Cancer data

```

rf_spec <- rand_forest(mtry = 4) %>%
  set_engine("ranger") %>%
  set_mode("classification")

can_rf_recipe <-
  recipe(Class ~ ., data = cancer_clean)

can_workflow_rf <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(can_rf_recipe)

fit_rf <- can_workflow_rf  %>%
  fit(cancer_clean)

compare_pred <- augment(fit_rf, new_data = can_test)

compare_pred %>% conf_mat(truth = Class, estimate = .pred_class)

```

Truth

```

Prediction benign malignant
benign      217       1
malignant     5      119

compare_pred %>%  accuracy(truth = Class, estimate = .pred_class)

# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>        <dbl>
1 accuracy  binary      0.982

```

Importance measures of different variables:

```

last_rf <-
  rand_forest(mtry = 4, trees = 100) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
# the last workflow
last_workflow <-
  can_workflow_rf %>%
  update_model(last_rf)

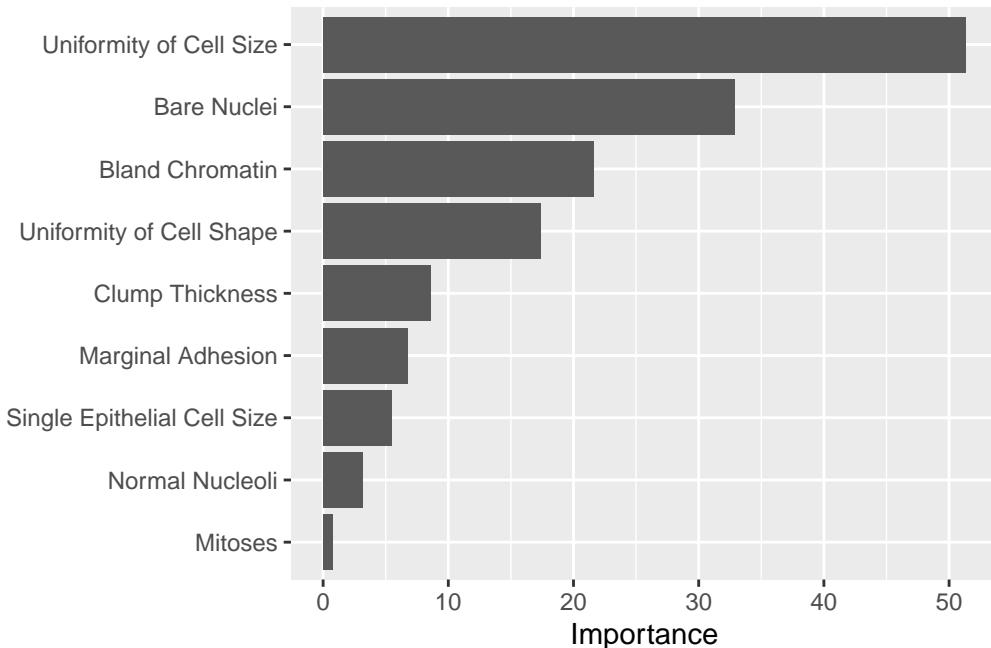
# the last fit
set.seed(3)
last_rf_fit <-
  last_workflow %>%
  last_fit(can_split)

last_rf_fit %>%
  collect_metrics()

# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>        <dbl> <chr>
1 accuracy  binary      0.968 Preprocessor1_Model1
2 roc_auc   binary      0.991 Preprocessor1_Model1

```

```
last_rf_fit %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip(num_features = 20)
```



15.5 References

1. [Introduction to Statistical Learning](#)
2. [Introcution to Statistical Learning Labs with Tidymodels](#)
3. [Tidy models tutorial](#)
4. [How Decision Trees Work](#)
5. [Python Data Science Handbook](#)
6. [Visualization of Random Forests](#)