

Introduction

Data Science: What and why?

Data science is an interdisciplinary field about processes and systems to extract knowledge or insights from data. The goals of Data Science include discovering new phenomena or trends from data, enabling decisions based on facts derived from data, and communicating findings from data. It is a continuation of some of the data analysis fields such as statistics, data mining, and predictive analytics.

Data Science is at the heart of the scientific method, which starts with making data-driven observations to formulate testable hypotheses. It furthermore comes into play to visualize and assess experimental results. Data science skills are therefore necessary to any field of scientific research. Data science is the main tools of epidemiology, the study of health and disease in populations, which largely relies on observational data. Moreover data science is important in the industry, to understand operational process, and in business analytics, to understand a particular market. Hence, with the rise of big data in all areas of society, data science skills are some of the most demanded skills on the job market. Last, but not least, in an era of fake news, data science skills are important for citizens of modern societies.

What you will learn and not learn

The goal of this course is to allow you to provide you general analytic techniques to extract knowledge, patterns, and connections between samples and variables, and how to communicate these in an intuitive and clear way.

This course focuses on front-end data science. This means, it teaches practical skills to analyse data. We will focus on tidy data, visualizations, and data manipulation in R. To only then dive into the math required to understand and interpret analysis results.

This course does not teach back-end data science, i.e. it does not teach how to develop your own statistical or machine learning models, nor how to develop scalable data processing software.

Other courses offered by the faculty of Informatics cover data science back-end skills.

The R language

R is a statistical programming language designed for data analytics. It is a great language for front-end data science, i.e. to rapidly manipulate, visualize and come to raising interesting hypotheses.

Seen from a software developer point of view (i.e. from a back-end data science point of view), R can be seen cumbersome and not using memory and computing resources efficiently. The purpose of the R language is to reduce time spent in coding to maximize user's brain time on looking at the data and thinking about it, rather than reducing

computer's running time. Of course, there are ways to develop efficient R software, notably by relying on implementation in lower languages such as C. This course does not cover such R developer skills.

Another advantage of R is that it offers a very large set of libraries from many application areas.

Course overview

The lecture is structured into three main parts covering the major steps of data analysis:

1. **Get the data:** After basic introduction to R, learn how to fetch and manipulate real-world datasets. How to structure them to most conveniently work with them (tidy data).
2. **Look at the data:** Basic and advanced visualization techniques allows navigating large and complex datasets, identifying interesting signal, and formulating hypotheses. Typical sources of confounding are discussed. Recommendation to present an analysis in compelling fashion are also given.
3. **Conclude:** Concepts of hypothesis testing will allow concluding about the statistical robustness of discovered associations. Also, methods from supervised learning will allow to model data and build accurate predictors.

The chapters of this script corresponds to individual lectures. Appendices provide further technical details as well as R tricks and tips.

Complementary reading

These books offer complementary information to this script:

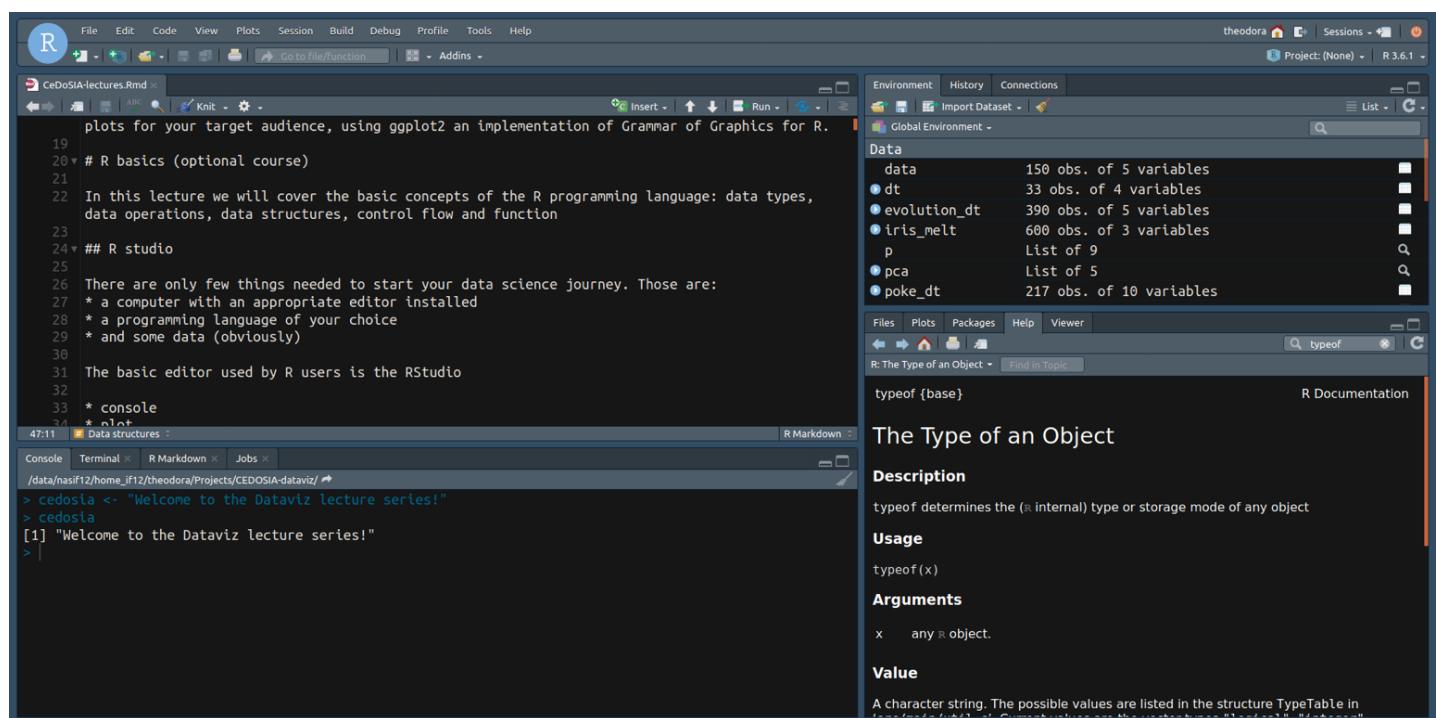
- Introduction to Data Science, Rafael A. Irizarry [<https://rafalab.github.io/dsbook/>]
- R for Data Science, Garrett Grolemund and Hadley Wickham [<https://r4ds.had.co.nz/>]
- Statistical Inference via Data Science, Chester Ismay and Albert Y. Kim [<https://moderndive.com/>]
- Fundamentals of Data Visualization, Claus O. Wilke [<https://clauswilke.com/dataviz/>]
- Advanced R, Hadley Wickham [<https://adv-r.hadley.nz/>]

Chapter 1 R basics

This chapter provides a quick introduction to the programming language R and to using the software RStudio.

1.1 Rstudio

Rstudio is a software that allows to program in R and interactively analyse data with R. It succinctly organizes your session into 4 panels each set up for you to do certain tasks in each panel: Edit and write code (source panel), run and execute code (console panel), list objects that you have in your environment and have a history of your past commands (environment/history panel), and a panel to show your folders' structure, see plots, install/load packages, and read the documentation of functions (files/plots/packages/help panel).



- the main script section, for writing scripts [top left section] (Ctrl+1 to focus)
- the console tab, for typing R commands directly [bottom left section as tab] (Ctrl+2 to focus)
- the terminal tab, for direct access to your system shell [bottom left section] (Shift+Alt+T to focus)
- the plot tab, where you see the last plot generated [bottom right section as tab]
- the help tab, with useful documentation of R functions [bottom right section as tab] (F1 on the name of a function or Ctrl+3)
- the history tab, with the list of the R commands used [top right section as tab]
- the environment tab, with the created variables and functions loaded [top right section as tab]
- and the packages tab, with the available/loaded R packages [bottom right section as tab]

Check the View menu to find out the rest of useful shortcuts!

1.2 First steps with R

This section is largely borrowed from the book Introduction to Data Science by Rafael Irizarry.

[<https://rafalab.github.io/dsbook>]

1.2.1 Objects

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

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

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

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

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

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

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

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

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

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

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

We use the term *object* to describe stuff that is stored in R.

1.2.2 The workspace

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

```
ls()
```

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

In RStudio, the variables of the environment are displayed in the *Environment* tab.

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

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

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

```
## [1] 0.618034
```

```
(-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
```

```
## [1] -1.618034
```

1.2.3 Functions

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

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

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

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

log(8)

```
## [1] 2.079442
```

log(a)

```
## [1] 0
```

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

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

For most functions, we can also use this shorthand:

```
?log
```

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

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

```
args(log)
```

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

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

```
log(8, base = 2)
```

```
## [1] 3
```

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

```
log(x = 8, base = 2)
```

```
## [1] 3
```

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

```
log(8, 2)
```

```
## [1] 3
```

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

```
log(base = 2, x = 8)
```

```
## [1] 3
```

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

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

```
2 ^ 3
```

```
## [1] 8
```

You can see the arithmetic operators by typing:

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

or

```
?"+"
```

and the relational operators by typing:

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

or

```
?">"
```

1.2.4 Other prebuilt objects

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

```
data()
```

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

```
co2
```

R will show you Mauna Loa atmospheric CO₂ concentration data.

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

```
pi
```

```
## [1] 3.141593
```

```
Inf+1
```

```
## [1] Inf
```

1.2.5 Variable names

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

```
install.packages <- 2 .
```

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

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

For more advice, we highly recommend studying Hadley Wickham's style guide¹.

1.2.6 Reusing scripts

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

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

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

1.2.7 Commenting your code

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

```
## Code to compute solution to quadratic equation of the form ax^2 + bx + c
## define the variables
a <- 3
b <- 2
c <- -1
## now compute the solution
(-b + sqrt(b^2 - 4*a*c)) / (2*a)
(-b - sqrt(b^2 - 4*a*c)) / (2*a)
```

1.3 Data types

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

```
a <- 2
class(a)

## [1] "numeric"
```

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

1.3.1 Data frames

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

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

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

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

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

1.3.2 Examining an object

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

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

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

```
head(murders)
```

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

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

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

1.3.3 The accessor: \$

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

```
murders$population
```

```
## [1] 4779736 710231 6392017 2915918 37253956 5029196 3574097 897934
## [9] 601723 19687653 9920000 1360301 1567582 12830632 6483802 3046355
## [17] 2853118 4339367 4533372 1328361 5773552 6547629 9883640 5303925
## [25] 2967297 5988927 989415 1826341 2700551 1316470 8791894 2059179
## [33] 19378102 9535483 672591 11536504 3751351 3831074 12702379 1052567
## [41] 4625364 814180 6346105 25145561 2763885 625741 8001024 6724540
## [49] 1852994 5686986 563626
```

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

```
names(murders)
```

```
## [1] "state"       "abb"        "region"      "population" "total"
```

It is important to know that the order of the entries in `murders$population` preserves the order of the rows in our data table.

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

1.3.4 Vectors: numerics, characters, and logical

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

```
pop <- murders$population
```

```
length(pop)
```

```
## [1] 51
```

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

```
class(pop)
```

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

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

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

```
class(murders$state)
```

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

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

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

```
z <- 3 == 2
```

```
z
```

```
## [1] FALSE
```

```
class(z)
```

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

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

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

```
?Comparison
```

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

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

1.3.5 Factors

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

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

```
## [1] "factor"
```

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

```
levels(murders$region)
```

```
## [1] "Northeast"      "South"          "North Central"   "West"
```

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

Note that the levels have an order that is different from the order of appearance in the factor object. The **default** in R is for the levels to follow **alphabetical order**. However, often we want the levels to follow a different order. You can specify an order through the `levels` argument when creating the factor with the `factor` function. For example, in the `murders` dataset regions are ordered from east to west. The function `reorder` lets us **change the order of the levels** of a factor variable based on a summary computed on a numeric vector. We will demonstrate this with a simple example, and will see more advanced ones in the Data Visualization part of the book.

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

```

region <- murders$region
value <- murders$total
region <- reorder(region, value, FUN = sum)
levels(region)

## [1] "Northeast"      "North Central"   "West"          "South"

```

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

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

1.3.6 Lists

Data frames are a special case of *lists*. Lists are useful because you can **store any combination of different types**. You can create a list using the `list` function like this:

```

record <- list(name = "John Doe",
            student_id = 1234,
            grades = c(95, 82, 91, 97, 93),
            final_grade = "A")

```

The function `c` is described in Section 1.4.

This list includes a character, a number, a vector with five numbers, and another character.

```
record
```

```

## $name
## [1] "John Doe"
##
## $student_id
## [1] 1234
##
## $grades
## [1] 95 82 91 97 93
##
## $final_grade
## [1] "A"

```

```
class(record)
```

```
## [1] "list"
```

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

```
record$student_id
```

```
## [1] 1234
```

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

```
record[["student_id"]]
```

```
## [1] 1234
```

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

You might also encounter lists without variable names.

```
record2 <- list("John Doe", 1234)  
record2
```

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

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

```
record2[[1]]
```

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

1.3.7 Matrices

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

Yet matrices have a major advantage over data frames: we can perform **matrix algebra operations**, a powerful type of mathematical technique. We do not describe these operations in this book, but much of what happens in the background when you perform a data analysis involves matrices.

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

```
mat <- matrix(1:12, 4, 3)
```

```
mat
```

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

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

```
mat[2, 3]
```

```
## [1] 10
```

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

```
mat[2, ]
```

```
## [1] 2 6 10
```

Notice that this returns a vector, not a matrix.

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

```
mat[, 3]
```

```
## [1] 9 10 11 12
```

This is also a vector, not a matrix.

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

```
mat[, 2:3]
```

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

You can subset both rows and columns:

```
mat[1:2, 2:3]
```

```
##      [,1] [,2]
## [1,]     5    9
## [2,]     6   10
```

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

```
as.data.frame(mat)
```

```
##   V1 V2 V3
## 1  1  5  9
## 2  2  6 10
## 3  3  7 11
## 4  4  8 12
```

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

```
data("murders")
murders[25, 1]

## [1] "Mississippi"

murders[2:3, ]

##      state abb region population total
## 2  Alaska  AK    West     710231    19
## 3 Arizona  AZ    West    6392017   232
```

1.4 Vectors

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

1.4.1 Creating vectors

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

```
codes <- c(380, 124, 818)  
codes
```

```
## [1] 380 124 818
```

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

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

In R you can also use single quotes:

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

But be careful not to confuse the single quote '`'` with the *back quote* ` By now you should know that if you type:

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

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

1.4.2 Names

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

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

```
##   italy canada egypt  
##     380      124      818
```

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

```
class(codes)

## [1] "numeric"
```

but with names:

```
names(codes)

## [1] "italy"  "canada" "egypt"
```

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

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

##  italy canada egypt
##  380     124     818
```

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

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

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

##  italy canada egypt
##  380     124     818
```

1.4.3 Sequences

Another useful function for creating vectors generates sequences:

```
seq(1, 10)

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

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

```
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

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

```
1:10
```

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

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

```
class(1:10)
```

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

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

```
class(seq(1, 10, 0.5))
```

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

1.4.4 Subsetting

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

```
codes[2]
```

```
## canada
##      124
```

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

```
codes[c(1,3)]
```

```
## italy egypt
##   380    818
```

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

```
codes[1:2]
```

```
##  italy canada
##   380     124
```

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

```
codes["canada"]
```

```
## canada
##   124
```

```
codes[c("egypt","italy")]
```

```
## egypt italy
##   818    380
```

1.5 Coercion

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

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

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

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

```
x
```

```
## [1] "1"      "canada" "3"
```

```
class(x)

## [1] "character"
```

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

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

```
x <- 1:5
y <- as.character(x)
y

## [1] "1" "2" "3" "4" "5"
```

You can turn it back with as.numeric :

```
as.numeric(y)

## [1] 1 2 3 4 5
```

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

1.5.1 Not availables (NA)

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

```
x <- c("1", "b", "3")
as.numeric(x)

## Warning: NAs introduced by coercion

## [1] 1 NA 3
```

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

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

1.6 Sorting

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

1.6.1 sort

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

```
library(dslabs)
data(murders)
sort(murders$total)
```

```
## [1] 2 4 5 5 7 8 11 12 12 16 19 21 22 27 32
## [16] 36 38 53 63 65 67 84 93 93 97 97 99 111 116 118
## [31] 120 135 142 207 219 232 246 250 286 293 310 321 351 364 376
## [46] 413 457 517 669 805 1257
```

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

1.6.2 order

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

```
x <- c(31, 4, 15, 92, 65)
sort(x)
```

```
## [1] 4 15 31 65 92
```

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

```
index <- order(x)
x[index]
```

```
## [1] 4 15 31 65 92
```

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

x

```
## [1] 31 4 15 92 65
```

```
order(x)
```

```
## [1] 2 3 1 5 4
```

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

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

```
murders$state[1:6]
```

```
## [1] "Alabama"      "Alaska"       "Arizona"      "Arkansas"     "California"
## [6] "Colorado"
```

```
murders$abb[1:6]
```

```
## [1] "AL" "AK" "AZ" "AR" "CA" "CO"
```

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

```
ind <- order(murders$total)
```

```
murders$abb[ind]
```

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

According to the above, California had the most murders.

1.6.3 **max and which.max**

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

```
max(murders$total)
```

```
## [1] 1257
```

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

```
i_max <- which.max(murders$total)
murders$state[i_max]
```

```
## [1] "California"
```

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

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

1.6.4 rank

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

```
x <- c(31, 4, 15, 92, 65)
rank(x)
```

```
## [1] 3 1 2 5 4
```

1.6.5 Beware of recycling

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

```
x <- c(1, 2, 3)
y <- c(10, 20, 30, 40, 50, 60, 70)
x+y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length

## [1] 11 22 33 41 52 63 71
```

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

1.7 Vector arithmetics

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

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

## [1] "California"
```

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

1.7.1 Rescaling a vector

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

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

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

```
inches * 2.54
```

```
## [1] 175.26 157.48 167.64 177.80 177.80 185.42 170.18 185.42 170.18 177.80
```

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

```
inches - 69
```

```
## [1] 0 -7 -3 1 1 4 -2 4 -2 1
```

1.7.2 Two vectors

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

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

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

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

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

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

```
murders$abb[order(murder_rate)]
```

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

1.8 Indexing

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

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

1.8.1 Subsetting with logicals

We have now calculated the murder rate using:

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

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

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

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

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

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

```
murders$state[ind]

## [1] "Hawaii"      "Iowa"        "New Hampshire" "North Dakota"
## [5] "Vermont"
```

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

```
sum(ind)
```

```
## [1] 5
```

1.8.2 Logical operators

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

```
TRUE & TRUE
```

```
## [1] TRUE
```

```
TRUE & FALSE
```

```
## [1] FALSE
```

```
FALSE & FALSE
```

```
## [1] FALSE
```

For our example, we can form two logicals:

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

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

```
ind <- safe & west
murders$state[ind]

## [1] "Hawaii"  "Idaho"   "Oregon"  "Utah"    "Wyoming"
```

1.8.3 `which`

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

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

## [1] 3.374138
```

1.8.4 `match`

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

```
ind <- match(c("New York", "Florida", "Texas"), murders$state)
ind
```

```
## [1] 33 10 44
```

Now we can look at the murder rates:

```
murder_rate[ind]

## [1] 2.667960 3.398069 3.201360
```

1.8.5 %in%

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

```
c("Boston", "Dakota", "Washington") %in% murders$state

## [1] FALSE FALSE TRUE
```

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

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

```
match(c("New York", "Florida", "Texas"), murders$state)

## [1] 33 10 44

which(murders$state %in% c("New York", "Florida", "Texas"))

## [1] 10 33 44
```

1.9 R programming

We teach R because it greatly facilitates data analysis. By coding in R, we can efficiently perform exploratory data analysis, build data analysis pipelines, and prepare data visualization to communicate results. However, R is not just a data analysis environment but a programming language. Advanced R programmers can develop complex packages and

even improve R itself, but we do not cover advanced programming. Nonetheless, in Appendix B, we introduce three key programming concepts: conditional expressions, for-loops, and functions. These are not just key building blocks for advanced programming, but are sometimes useful during data analysis.

1. <http://adv-r.had.co.nz/Style.html> ↵

Chapter 2 Data wrangling

Data wrangling refers to the task of processing raw data into useful formats. This Chapter introduces basic data wrangling operations in R using `data.table`.

2.1 Data.tables

2.1.1 Overview

`data.table` objects are a modern implementation of tables containing variables stored in columns and observations stored in rows. Base R provides a similar structure called `data.frame`. However, we will exclusively use `data.table` in this course because `data.frame` objects are a lot slower and often a little more complicated to use.

A `data.table` is a memory efficient and faster implementation of `data.frame`. It is more efficient because it **operates on its columns by reference**. In contrast modifying a `data.frame` leads R to copy the entire `data.frame`.

Like a `data.frame`, each column can have a different type. Unlike a `data.frame`, it doesn't have row names. It accepts all `data.frame` functions to ensure compatibility, but it has a shorter and more flexible syntax. This may be not so straightforward in the beginning but pays off and saves time on two fronts:

- programming (easier to code, read, debug and maintain)
- computing (fast and memory efficient)

The general basic form of the `data.table` syntax is:

```
DT[ i, j, by ] #  
|   |   |  
|   |   -----> grouped by what?  
|   -----> what to do with the columns?  
---> on which rows?
```

The way to read this out loud is: “Take `DT`, subset rows by `i`, then compute `j` grouped by `by`”.

We will now describe some basic usage examples expanding on this definition. First of all, let us create and inspect some `data.tables` to get a first impression.

2.1.2 Creating and loading tables

To create a `data.table`, we just name its columns and populate them. All the columns have to have the same length. If vectors of different lengths are provided upon creation of a `data.table`, R automatically recycles the values of the shorter vectors. Here is an example:

```
# install.packages("data.table")
library(data.table)
DT <- data.table(x = rep(c("a","b","c"), each = 3), y = c(1, 3, 6), v = 1:9)
DT # note how column y was recycled

##      x y v
## 1: a 1 1
## 2: a 3 2
## 3: a 6 3
## 4: b 1 4
## 5: b 3 5
## 6: b 6 6
## 7: c 1 7
## 8: c 3 8
## 9: c 6 9
```

If we want to convert any other R object to a `data.table`, all we have to do is to call the `as.data.table()` function. This is typically done for `data.frame` objects.

```
# This way we can for example convert any built-in dataset
# coming as a data.frame into a data.table:
titanic_dt <- as.data.table(Titanic)
class(titanic_dt)

## [1] "data.table" "data.frame"
```

Here you can see that the `class` function informs us that `titanic_dt` is both a `data.table` and a `data.frame` as `data.table` inherits from `data.frame`.

Alternatively, we can read files from disk and process them using `data.table`. The easiest way to do so is to use the function `fread()`. Here is an example using a subset of the Kaggle flight and airports dataset that is limited to flights going in or to the Los Angeles airport. We refer to the description Kaggle flights and airports challenge for more details [<https://www.kaggle.com/tylerx/flights-and-airports-data>].

To run the following code, save the comma-separated value file `flightsLAX.csv` into a local folder of your choice and replace the string `"path_to_file"` with the actual path to your `flightsLAX.csv` file. For example `"path_to_file"` could be substituted with `"/Users/samantha/mydataviz_folder/extdata"`. See Appendix I “Importing data” for more details.

```
flights <- fread('path_to_file/flightsLAX.csv')
```

Typing the name of the newly created `data.table` (`flights`) in the console displays its first and last rows. We observe that reading the file was successful.

```
flights
```

```
##          YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER
## 1: 2015     1    1        4       AA      2336   N3KUAA
## 2: 2015     1    1        4       AA      258    N3HYAA
## 3: 2015     1    1        4       US     2013   N584UW
## 4: 2015     1    1        4       DL     1434   N547US
## 5: 2015     1    1        4       AA      115    N3CTAA
##    ---
## 389365: 2015 12   31        4       AA     1538   N866AA
## 389366: 2015 12   31        4       AS      175    N431AS
## 389367: 2015 12   31        4       AS      471    N570AS
## 389368: 2015 12   31        4       AA      219    N3LYAA
## 389369: 2015 12   31        4       B6      688    N657JB
##          ORIGIN_AIRPORT DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE
## 1:           LAX             PBI                  2     263    2330
## 2:           LAX             MIA                 15     258    2342
## 3:           LAX             CLT                 44     228    2125
## 4:           LAX             MSP                 35     188    1535
## 5:           LAX             MIA                103     255    2342
##    ---
## 389365:       LAX             MIA                2357     250    2342
## 389366:       LAX             ANC                2350     291    2345
## 389367:       LAX             SEA                2353     132    954
## 389368:       LAX             ORD                2358     198   1744
## 389369:       LAX             BOS                2355     272   2611
##          ARRIVAL_TIME
## 1:        741
## 2:        756
## 3:        753
## 4:        605
## 5:        839
##    ---
## 389365:      731
## 389366:      400
## 389367:      225
## 389368:      544
## 389369:      753
```

2.1.3 Inspecting tables

A first step in any analysis should involve inspecting the data we just read in. This often starts by looking the first and last rows of the table as we did above. The next information we are often interested in is the size of our data set. We can use the following commands to obtain it:

```
ncol(flights) # nrow(flights) for number of rows
```

```
## [1] 13
```

```
dim(flights) # returns nrow and ncol
```

```
## [1] 389369 13
```

Next, we are often interested in **basic statistics on the columns**. To obtain this information we can call the `summary()` function on the table.

```
summary(flights[, 1:6])
```

```
##      YEAR        MONTH       DAY     DAY_OF_WEEK
##  Min.   :2015   Min.   : 1.000  Min.   : 1.0  Min.   :1.000
##  1st Qu.:2015   1st Qu.: 3.000  1st Qu.: 8.0  1st Qu.:2.000
##  Median :2015   Median : 6.000  Median :16.0  Median :4.000
##  Mean    :2015   Mean   : 6.198  Mean   :15.7  Mean   :3.934
##  3rd Qu.:2015   3rd Qu.: 9.000  3rd Qu.:23.0  3rd Qu.:6.000
##  Max.    :2015   Max.   :12.000  Max.   :31.0  Max.   :7.000
##      AIRLINE      FLIGHT_NUMBER
##  Length:389369  Min.   : 1
##  Class :character  1st Qu.: 501
##  Mode  :character  Median :1296
##                  Mean   :1905
##                  3rd Qu.:2617
##                  Max.   :6896
```

This provides us already a lot of information about our data. We can for example see that all data is from 2015 as all values in the YEAR column are 2015. But for categorical data this is not very insightful, as we can see for the AIRLINE column.

To investigate **categorical columns** we can have a look at their **unique elements** using:

```
flights[, unique(AIRLINE)]
```

```
## [1] "AA" "US" "DL" "UA" "OO" "AS" "B6" "NK" "VX" "WN" "HA" "F9" "MQ"
```

This command provided us the airline identifiers present in the dataset. Another valuable information for categorical variables is **how often each category occurs**. This can be obtained using the following commands:

```
flights[, table(AIRLINE)]  
  
## AIRLINE  
## AA AS B6 DL F9 HA MQ NK OO UA US VX WN  
## 65483 16144 8216 50343 2770 3112 368 8688 73389 54862 7374 23598 75022
```

2.2 Row subsetting

As mentioned, the general basic form of the data.table syntax is:

```
DT[ i, j, by ] #  
| | |  
| | -----> grouped by what?  
| -----> what to do with the columns?  
---> on which rows?
```

Let us first look at the `i` argument, i.e. row indexing. The parameter `i` can be any vector of integers, corresponding to the row indices to select, or some logical vectors indicating which rows to select. Here are some typical examples.

2.2.1 Subsetting rows by indices

If we want to see the second element of the table, we can do the following:

```
flights[2, ] # Access the 2nd row (also flights[2] or flights[i = 2])
```

```
## YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER ORIGIN_AIRPORT  
## 1: 2015 1 1 4 AA 258 N3HYAA LAX  
## DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE ARRIVAL_TIME  
## 1: MIA 15 258 2342 756
```

A shorter writing allows leaving out the comma:

```
flights[2] # Access the 2nd row (also flights[2] or flights[i = 2])
```

```
##   YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER ORIGIN_AIRPORT
## 1: 2015     1    1        4      AA          258       N3HYAA        LAX
##   DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE ARRIVAL_TIME
## 1:                 MIA           15      258      2342        756
```

For accessing multiple consecutive rows we can use the `start:stop` syntax as for example:

```
flights[1:3]
```

```
##   YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER ORIGIN_AIRPORT
## 1: 2015     1    1        4      AA          2336       N3KUAA        LAX
## 2: 2015     1    1        4      AA          258       N3HYAA        LAX
## 3: 2015     1    1        4      US          2013       N584UW        LAX
##   DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE ARRIVAL_TIME
## 1:                 PBI           2       263      2330        741
## 2:                 MIA           15      258      2342        756
## 3:                 CLT           44      228      2125        753
```

Accessing multiple rows that `are not necessarily consecutive` can be done by creating an `index vector with c()`:

```
flights[c(3, 5)]
```

```
##   YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER ORIGIN_AIRPORT
## 1: 2015     1    1        4      US          2013       N584UW        LAX
## 2: 2015     1    1        4      AA          115       N3CTAA        LAX
##   DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE ARRIVAL_TIME
## 1:                 CLT           44      228      2125        753
## 2:                 MIA           103     255      2342        839
```

2.2.2 Subsetting rows by logical conditions

Often, a more useful way to subset rows is using logical conditions, using for `i` a logical vector. We can create such logical vectors using the following binary operators:

- `==`
- `<`
- `>`
- `!=`
- `%in%`

For example, entries of flights operated by “AA” (American Airlines) can be extracted using:

```
flights[AIRLINE == "AA"]
```

```
##          YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER
## 1: 2015     1    1        4      AA         2336   N3KUAA
## 2: 2015     1    1        4      AA         258    N3HYAA
## 3: 2015     1    1        4      AA         115    N3CTAA
## 4: 2015     1    1        4      AA        2410   N3BAAA
## 5: 2015     1    1        4      AA        1515   N3HMAA
## --- 
## 65479: 2015 12   31        4      AA        1116   N029AA
## 65480: 2015 12   31        4      AA        246    N863AA
## 65481: 2015 12   31        4      AA        1927   N837AW
## 65482: 2015 12   31        4      AA        1538   N866AA
## 65483: 2015 12   31        4      AA        219    N3LYAA
##          ORIGIN_AIRPORT DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE
## 1:             LAX           PBI            2       263     2330
## 2:             LAX           MIA            15      258     2342
## 3:             LAX           MIA           103      255     2342
## 4:             LAX           DFW            600      150     1235
## 5:             LAX           ORD            557      202     1744
## --- 
## 65479:          LAX           ATL           2256      207     1947
## 65480:          KOA           LAX           2335      272     2504
## 65481:          LAX           IAD           2346      239     2288
## 65482:          LAX           MIA           2357      250     2342
## 65483:          LAX           ORD           2358      198     1744
##          ARRIVAL_TIME
## 1:        741
## 2:        756
## 3:        839
## 4:       1052
## 5:       1139
## --- 
## 65479:      553
## 65480:      635
## 65481:      712
## 65482:      731
## 65483:      544
```

Alternatively, if we are interested in all flights from any destination to the airports in NYC ("JFK" and "LGA"), we can subset the rows using the following command:

```
flights[DESTINATION_AIRPORT %in% c("LGA", "JFK")]
```

```

##      YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER
## 1: 2015     1    1          4      B6           24   N923JB
## 2: 2015     1    1          4      DL          476   N196DN
## 3: 2015     1    1          4      AA          118   N788AA
## 4: 2015     1    1          4      VX          404   N621VA
## 5: 2015     1    1          4      UA          275   N598UA
## --- 
## 12011: 2015 12   31          4      AA          180   N796AA
## 12012: 2015 12   31          4      B6          524   N934JB
## 12013: 2015 12   31          4      B6          624   N942JB
## 12014: 2015 12   31          4      DL          1262  N394DL
## 12015: 2015 12   31          4      B6          1124  N943JB
##      ORIGIN_AIRPORT DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE
## 1:          LAX            JFK             620       279      2475
## 2:          LAX            JFK             650       274      2475
## 3:          LAX            JFK             650       268      2475
## 4:          LAX            JFK             728       268      2475
## 5:          LAX            JFK             806       277      2475
## --- 
## 12011:        LAX            JFK            1640      259      2475
## 12012:        LAX            JFK            1645      261      2475
## 12013:        LAX            JFK            2107      280      2475
## 12014:        LAX            JFK            2244      256      2475
## 12015:        LAX            JFK            2349      274      2475
##      ARRIVAL_TIME
## 1:        1413
## 2:        1458
## 3:        1436
## 4:        1512
## 5:        1606
## --- 
## 12011:        18
## 12012:        18
## 12013:        513
## 12014:        625
## 12015:        748

```

Additionally, we can concatenate multiple conditions using the logical OR `|` or the logical AND `&` operator. For instance, if we want to inspect all flights departing between 6am and 7am operated by American Airlines we can use the following statement:

```
flights[AIRLINE == "AA" & DEPARTURE_TIME > 600 & DEPARTURE_TIME < 700]
```

```

##      YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER
## 1: 2015     1    1          4    AA        1686   N4XXAA
## 2: 2015     1    1          4    AA        1361   N3KYAA
## 3: 2015     1    1          4    AA        2420   N3ERAA
## 4: 2015     1    1          4    AA        338    N3DJAA
## 5: 2015     1    1          4    AA        2424   N3DLAA
## ---
## 3820: 2015   12   31          4    AA        169    N787AA
## 3821: 2015   12   31          4    AA        1352   N7CAAA
## 3822: 2015   12   31          4    AA        146    N3MKAA
## 3823: 2015   12   31          4    AA        2453   N869AA
## 3824: 2015   12   31          4    AA        118    N791AA
##      ORIGIN_AIRPORT DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE
## 1:           LAX             STL        609       183      1592
## 2:           BNA             LAX        607       255      1797
## 3:           LAX             DFW        619       149      1235
## 4:           LAX             SF0        644        55      337
## 5:           LAX             DFW        641       146      1235
## ---
## 3820:         SF0             LAX        623        54      337
## 3821:         MIA             LAX        651       303      2342
## 3822:         LAX             BOS        650       268      2611
## 3823:         LAX             DFW        651       142      1235
## 3824:         LAX             JFK        659       272      2475
##      ARRIVAL_TIME
## 1:      1134
## 2:      847
## 3:     1119
## 4:      803
## 5:     1149
## ---
## 3820:      740
## 3821:      913
## 3822:     1446
## 3823:     1134
## 3824:     1505

```

2.3 Column operations

You may have wondered why R correctly runs code such as `flights[AIRLINE == "AA"]` although `AIRLINE` is not a variable of the environment but a column of the `data.table` `flights`. Such a call would not execute properly with a `data.frame`. The reason is that code entered inside the `[]` brackets of a `data.table` is interpreted using the

`data.table` environment. Inside this environment, columns are seen as variables already. This makes the syntax very light and readable for row subsetting as we just saw. It becomes particularly powerful for column operations. We now look at this.

Although feasible, it is not advisable to access a column by its number since the ordering or number of columns can easily change. Also, if you have a data set with a large number of columns (e.g. 50), how do you know which one is column 18? Therefore, **use the column name instead**. Using column names prevents bugs.

2.3.1 Working with columns

Accessing columns by name also makes the code more readable: `flights[, TAIL_NUMBER]` instead of `flights[, 7]`.

```
flights[1:10, TAIL_NUMBER]      # Access column x (also DT$x or DT[j=x]).  
  
## [1] "N3KUAA" "N3HYAA" "N584UW" "N547US" "N3CTAA" "N76517" "N925SW" "N719SK"  
## [9] "N435SW" "N560SW"
```

For accessing a specific cell (i.e. specific column and specific row), we can use the following syntax:

```
flights[4, TAIL_NUMBER]      # Access a specific cell.  
  
## [1] "N547US"
```

This command for **accessing multiple columns** would return a vector:

```
flights[1:2, c(TAIL_NUMBER, ORIGIN_AIRPORT)]  
  
## [1] "N3KUAA" "N3HYAA" "LAX"      "LAX"
```

However, when accessing many columns, we probably want to **return a `data.table`** instead of a vector. For that, we need to provide R with a list, so we use `list(colA, colB)` or its simplified version `.(colA, colB)`:

```
flights[1:2, list(TAIL_NUMBER, ORIGIN_AIRPORT)]  
  
##    TAIL_NUMBER ORIGIN_AIRPORT  
## 1:      N3KUAA          LAX  
## 2:      N3HYAA          LAX
```

```
# Same as before.

flights[1:2, .(TAIL_NUMBER, ORIGIN_AIRPORT)]
```

	TAIL_NUMBER	ORIGIN_AIRPORT
## 1:	N3KUAA	LAX
## 2:	N3HYAA	LAX

2.3.2 Column operations

We already saw that inside the `[]` environment, columns are seen as variables, so we can apply functions to them.

```
# Similar to mean(flights[, AIR_TIME])

flights[, mean(AIR_TIME, na.rm=TRUE)]
```

```
## [1] 162.1379
```

```
flights[AIRLINE == "00", mean(AIR_TIME, na.rm=TRUE)]
```

```
## [1] 68.02261
```

To compute operations in multiple columns, we must provide a list (unless we want the result to be a vector).

```
# Same as flights[, .(mean(AIR_TIME), median(AIR_TIME))]

flights[, list(mean(AIR_TIME, na.rm=TRUE), median(AIR_TIME, na.rm=TRUE))]
```

```
##          V1    V2
## 1: 162.1379 150
```

To give meaningful names to the computations from before, we can use the following command:

```
flights[, .(mean_AIR_TIME = mean(AIR_TIME, na.rm=TRUE), median_AIR_TIME = median(AIR_TIME, na.rm=TRUE))]
```

```
##      mean_AIR_TIME median_AIR_TIME
## 1:     162.1379         150
```

Any operation can be applied to the columns, just as with variables. This code computes the average speed as the ratio of `AIR_TIME` over `DISTANCE` for the 5 first entries of the table `flights`:

```
flights[1:5,AIR_TIME/DISTANCE]

## [1] 0.1128755 0.1101623 0.1072941 0.1224756 0.1088813
```

2.3.3 Advanced commands: *apply() over columns

The columns of a `data.table` are exposed as a list to the environment. Therefore functions applying to a list can be applied to them, including those of the `*apply` family such as `sapply()`. For example:

```
sapply(flights, class) # Try the same with lapply

##          YEAR          MONTH          DAY      DAY_OF_WEEK
##     "integer"     "integer"     "integer"     "integer"
##    AIRLINE    FLIGHT_NUMBER    TAIL_NUMBER ORIGIN_AIRPORT
##   "character"     "integer"     "character"     "character"
## DESTINATION_AIRPORT DEPARTURE_TIME        AIR_TIME       DISTANCE
##   "character"     "integer"     "integer"     "integer"
## ARRIVAL_TIME           "integer"
##                      "integer"

# Note that we can access columns stored as variables by setting with=F.
# In this case, `colnames(iris_dt)!="Species"` returns a logical vector and `
# iris_dt` is subsetted by the logical vector

# Same as sapply(iris_dt[, 1:4], sum)
#sapply(iris_dt[, colnames(iris_dt)!="Species", with = F], sum)
```

2.4 The ‘by’ option

The `by` option allows executing the `j` command **by groups**. For example, we can use `by =` to compute the mean flight time per airline:

```
flights[, .(mean_AIRTIME = mean(AIR_TIME, na.rm=TRUE)), by = AIRLINE]
```

```
##      AIRLINE mean_AIRTIME
## 1:     AA    219.48133
## 2:     US    210.39488
## 3:     DL    207.07201
## 4:     UA    211.62008
## 5:     OO     68.02261
## 6:     AS    141.01870
## 7:     B6    309.79568
## 8:     NK    179.55828
## 9:     VX    185.36374
## 10:    WN    105.19976
## 11:    HA    307.95961
## 12:    F9    159.94041
## 13:    MQ    102.15210
```

This way we can easily spot that one airline conducts on average shorter flights. Moreover, we can compute the mean and standard deviation of the air time of every airline:

```
flights[, .(mean_AIRTIME = mean(AIR_TIME, na.rm=TRUE), sd_AIR_TIME = sd(AIR_TIME, na.rm=TRUE)), by = AIRLINE]
```

```
##      AIRLINE mean_AIRTIME sd_AIR_TIME
## 1:     AA    219.48133   92.889719
## 2:     US    210.39488   105.224833
## 3:     DL    207.07201   88.908566
## 4:     UA    211.62008   94.832456
## 5:     OO     68.02261   41.065036
## 6:     AS    141.01870   51.806424
## 7:     B6    309.79568   28.457740
## 8:     NK    179.55828   78.194706
## 9:     VX    185.36374   113.504572
## 10:    WN    105.19976   69.257334
## 11:    HA    307.95961   23.905491
## 12:    F9    159.94041   61.412379
## 13:    MQ    102.15210   8.531046
```

Although we could write `flights[i = 5, j = AIRLINE]`, we usually omit the `i =` and `j =` from the syntax, and write `flights[5, AIRLINE]` instead. However, for clarity we usually include the `by =` in the syntax.

2.5 Counting occurrences with `.N`

The `.N` is a special in-built variable that counts the number observations within a table. Evaluating `.N` alone is equal to `nrow()` of a table.

```
flights[, .N]
```

```
## [1] 389369
```

```
nrow(flights)
```

```
## [1] 389369
```

But the `.N` command becomes a lot more powerful when used with grouping or conditioning. We already saw earlier how we can use it to count the number of occurrences of elements in categorical columns. For instance, we can get the number of flights for each airline:

```
flights[, .N, by = 'AIRLINE']
```

```
##      AIRLINE     N
## 1:      AA 65483
## 2:      US  7374
## 3:      DL 50343
## 4:      UA 54862
## 5:      OO 73389
## 6:      AS 16144
## 7:      B6  8216
## 8:      NK  8688
## 9:      VX 23598
## 10:     WN 75022
## 11:     HA  3112
## 12:     F9  2770
## 13:     MQ   368
```

Remembering the `data.table` definition: “Take **DT**, subset rows using **i**, then select or calculate **j**, grouped by **by**”, we can build even more powerful statements using all three elements. For example, we can, for each airline, get the number of flights arriving to the airport JFK:

```
flights[DESTINATION_AIRPORT == "JFK", .N, by = 'AIRLINE']
```

```
##      AIRLINE     N
## 1:      B6 2488
## 2:      DL 2546
## 3:      AA 3804
## 4:      VX 1652
## 5:      UA 1525
```

2.6 Extending tables

2.6.1 Creating new columns (the := command)

The `:=` operator updates the `data.table` we are working in place, so writing `DT <- DT[, ... := ...]` is redundant. This operator, plus all `set` functions (e.g. `setnames` for column name renaming, `setcolorder` for changing the positions in which the columns positioned are inside the `DT`, etc), change their input by *reference*. No copy of the object is made, which makes the operation faster and less memory-consuming.

As an example, we can add a new column called `SPEED` (in miles per hour) whose value is the `DISTANCE` divided by `AIR_TIME` times 60:

```
flights[, SPEED := DISTANCE / AIR_TIME * 60]
head(flights)
```

```
##      YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER TAIL_NUMBER ORIGIN_AIRPORT
## 1: 2015     1    1        4      AA         2336       N3KUAA        LAX
## 2: 2015     1    1        4      AA         258        N3HYAA        LAX
## 3: 2015     1    1        4      US         2013       N584UW        LAX
## 4: 2015     1    1        4      DL         1434       N547US        LAX
## 5: 2015     1    1        4      AA         115        N3CTAA        LAX
## 6: 2015     1    1        4      UA         1545       N76517        LAX
##      DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE ARRIVAL_TIME      SPEED
## 1:                  PBI             2       263     2330          741 531.5589
## 2:                  MIA            15       258     2342          756 544.6512
## 3:                  CLT            44       228     2125          753 559.2105
## 4:                  MSP            35       188     1535          605 489.8936
## 5:                  MIA           103       255     2342          839 551.0588
## 6:                  IAH            112       156     1379          607 530.3846
```

Having computed a new column using the `:=` operator, we can use it for further analyses. For instance, we can compute the average speed, air time and distance for each airline:

```
flights[, .(mean_AIR_TIME = mean(AIR_TIME, na.rm=TRUE),
            mean_SPEED = mean(SPEED, na.rm=TRUE),
            mean_DISTANCE = mean(DISTANCE, na.rm=TRUE)
), by=AIRLINE]
```

	AIRLINE	mean_AIR_TIME	mean_SPEED	mean_DISTANCE
## 1:	AA	219.48133	461.2839	1739.2331
## 2:	US	210.39488	452.1641	1658.2581
## 3:	DL	207.07201	466.0330	1656.2165
## 4:	UA	211.62008	464.2928	1693.5504
## 5:	OO	68.02261	349.5549	437.2337
## 6:	AS	141.01870	439.0120	1040.0340
## 7:	B6	309.79568	484.8242	2486.1489
## 8:	NK	179.55828	450.0221	1402.1591
## 9:	VX	185.36374	433.0870	1432.5384
## 10:	WN	105.19976	409.3803	760.2593
## 11:	HA	307.95961	497.3118	2537.8107
## 12:	F9	159.94041	461.0684	1235.6664
## 13:	MQ	102.15210	435.5580	737.0000

Now we can see that the flights by the carrier “OO” are not just shorter, but also slow. This could for example lead us to the hypothesis, that “OO” is a small regional carrier, which operates slower planes.

Additionally we can use the `:=` operator to remove columns. If we for example observe that tail numbers are not important for our analysis we can remove them with the following statement:

```
flights[, TAIL_NUMBER := NULL]
head(flights)

##      YEAR MONTH DAY DAY_OF_WEEK AIRLINE FLIGHT_NUMBER ORIGIN_AIRPORT
## 1: 2015     1    1          4      AA           2336        LAX
## 2: 2015     1    1          4      AA           258         LAX
## 3: 2015     1    1          4      US           2013        LAX
## 4: 2015     1    1          4      DL           1434        LAX
## 5: 2015     1    1          4      AA           115         LAX
## 6: 2015     1    1          4      UA           1545        LAX
##      DESTINATION_AIRPORT DEPARTURE_TIME AIR_TIME DISTANCE ARRIVAL_TIME SPEED
## 1:                 PBI              2       263     2330        741 531.5589
## 2:                 MIA              15       258     2342        756 544.6512
## 3:                 CLT              44       228     2125        753 559.2105
## 4:                 MSP              35       188     1535        605 489.8936
## 5:                 MIA             103       255     2342        839 551.0588
## 6:                 IAH              112       156     1379        607 530.3846
```

Here we observe, that the tail numbers are gone from the `data.table`.

2.6.2 Advanced: Multiple assignments

With the following syntax we can assign multiple new columns at once. We use the base R dataset `iris`,² which we first transform into a `data.table`.

```
# load the Iris data table
iris_dt <- as.data.table(iris)

# Add columns with sepal and petal area. Note the syntax of multiple assignment.
iris_dt[, `:=` (Sepal.Area = Sepal.Length * Sepal.Width,
               Petal.Area = Petal.Length * Petal.Width)][1:3]

##      Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Area
## 1:        5.1       3.5        1.4       0.2   setosa    17.85
## 2:        4.9       3.0        1.4       0.2   setosa    14.70
## 3:        4.7       3.2        1.3       0.2   setosa    15.04
##      Petal.Area
## 1:     0.28
## 2:     0.28
## 3:     0.26
```

You can also delete columns by using the `:=` command.

```
# Let's assume setosa flowers are orange, versicolor purple and virginica pink.
# Add a column with these colors.

iris_dt[Species == "setosa", color := "orange"]
iris_dt[Species == "versicolor", color := "purple"]
iris_dt[Species == "virginica", color := "pink"]
unique(iris_dt[, .(Species, color)])

##      Species color
## 1:    setosa orange
## 2: versicolor purple
## 3:  virginica  pink

# We can delete this new column by setting it to NULL
iris_dt[, color := NULL]
colnames(iris_dt)
```

```
## [1] "Sepal.Length" "Sepal.Width"   "Petal.Length" "Petal.Width"   "Species"
## [6] "Sepal.Area"    "Petal.Area"
```

2.6.3 Copying tables

What do we mean when we say that `data.table` modifies columns by *reference*?

It means that no new copy of the object is made in the memory, unless we actually create one using `copy()`.

```
or_dt <- data.table(a = 1:10, b = 11:20)
# No new object is created, both new_dt and or_dt point to the same memory chunk.

new_dt <- or_dt
new_dt[, ab := a*b]
colnames(or_dt) # or_dt was also affected by changes in new_dt

## [1] "a" "b" "ab"

or_dt <- data.table(a = 1:10, b = 11:20)
copy_dt <- copy(or_dt) # By creating a copy, we have 2 objects in memory
copy_dt[, ab := a*b]
colnames(or_dt) # Changes in the copy don't affect the original

## [1] "a" "b"
```

2.7 Summary

By now, you should be able to answer the following questions:

- How to subset by rows or columns? Remember: `DT[i, j, by]`.
- How to add columns?
- How to make operations with different columns?

2.8 Data.table resources

The help page for `data.table`.

<https://cran.r-project.org/web/packages/data.table/>

<https://s3.amazonaws.com/..../assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf>

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

<http://adv-r.had.co.nz/Environments.html>

2. https://en.wikipedia.org/wiki/Iris_flower_data_set ↵

Chapter 3 Tidy data and combining tables

3.1 Introduction

3.1.1 Motivation

Without good practices, much of the time of a data analyst can be wasted in data wrangling rather than visualization or analysis. The concept of tidy data (Wickham 2014) addresses this issue by offering a standard representation of data, that is easy to manipulate, model and visualize. This chapter introduces the notion of tidy data and operations for tidying up messy datasets. Moreover, we describe how to easily concatenate tables with the same format and merge tables with common variables. This will set us ready for data visualization and analytics.

This chapter is partially adopted from “Introduction to Data Science” by Rafael A. Irizarry (<https://rafalab.github.io/dsbook/>) and uses concepts and material introduced by the developers of the `tidyverse` package.

3.1.2 Datasets used in this chapter

The following code chunks load libraries and tables used throughout this chapter.

```
library(data.table) # melt, dcast, ...
library(tidyverse) # separate, unite, ...
```

```
DATADIR <- "extdata"

election_results <- fread(
  file.path(DATADIR, "US-pres16results.csv"),
  na.strings=c("NULL", "NA"), encoding = "UTF-8", sep = ","
)

election_results <- election_results[
  is.na(county) & st != "US",
  .(cand, st, votes, total_votes)
]

setnames(election_results, "cand", "candidate")
setnames(election_results, "st", "state")

table1 <- fread(
  file.path(DATADIR, "table1_alternate.csv"),
  na.strings=c("NULL", "NA"), encoding = "UTF-8", sep = ","
)

table2 <- fread(
  file.path(DATADIR, "table2_alternate.csv"),
  na.strings=c("NULL", "NA"), encoding = "UTF-8", sep = ","
)

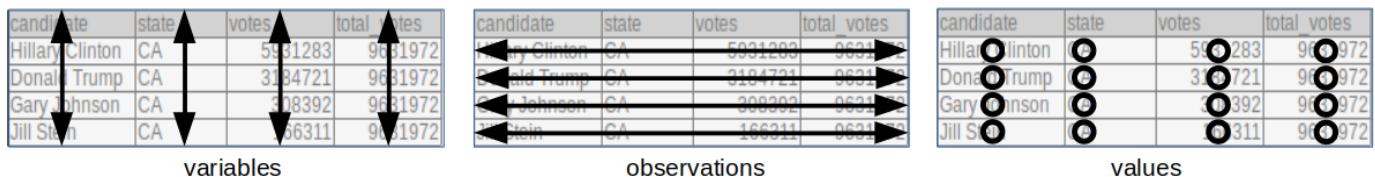
table3 <- fread(
  file.path(DATADIR, "table3_alternate.csv"),
  na.strings=c("NULL", "NA"), encoding = "UTF-8", sep = ","
)

table4 <- fread(
  file.path(DATADIR, "table4_alternate.csv"),
  na.strings=c("NULL", "NA"), encoding = "UTF-8", sep = ","
)

table5 <- fread(
  file.path(DATADIR, "table5_alternate.csv"),
  na.strings=c("NULL", "NA"), encoding = "UTF-8", sep = ","
)
```

3.2 Tidy and untidy data

3.2.1 Definition of tidy data



Tidy data table layout. Each variable has a column, each observation a row and each value a cell.

We say that a data table is in *tidy* format if:

1. Each **variable** has its own **column**.
2. Each **observation** has its own **row**.
3. Each **value** has its own **cell**.

The following dataset from the 2016 US presidential vote³ is an example of a tidy dataset:

```
head(election_results)
```

```
##           candidate state   votes total_votes
## 1: Hillary Clinton    CA 5931283    9631972
## 2: Donald Trump       CA 3184721    9631972
## 3: Gary Johnson       CA 308392     9631972
## 4: Jill Stein         CA 166311     9631972
## 5: Gloria La Riva     CA  41265    9631972
## 6: Donald Trump       FL 4605515   9386750
```

Each row represents a state and a candidate with each of the four values related to these states stored in the four variables: candidate, state, votes, and total_votes.

3.2.2 Advantages of tidy data

Organizing data in a tidy fashion reduces the burden to frequently reorganize the data. In particular, the advantages are:

- Easier manipulation using `data.table` commands such as sub-setting by rows and columns, as well as by operations
- Vectorized operations become easier to use
- Many other tools work better with tidy data, including plotting functions, hypothesis testing functions, and modeling functions such as linear regression. These advantages will become striking in the following chapters.

3.2.3 Common signs of untidy datasets

Often, untidy datasets can be identified by one or more of the following issues (Wickham 2014):

- Column headers are values, not variable names
- Multiple variables are stored in one column
- Variables are stored in both rows and columns
- A single observational unit is stored in multiple tables

Wickham (2014) furthermore mentions “Multiple types of observational units stored in the same table” as a sign of untidy data. This point is discussed in Section 3.6.

3.3 Tidying up datasets

In this part of the chapter, we show how to transform untidy datasets into tidy ones. To this end, we will present some of the most often encountered untidy formats and present specific solutions to each of them.

3.3.1 Melting (wide to long)

One of the most used operations to obtain tidy data is to transform a wide table into a long table. This operation, which transforms a wide table into a long table is called melting, by analogy with melting a piece of metal. It is useful in particular when data is untidy because column headers are values, and not variable names.

As an example, consider the table below which reports vote counts for two US states, California and Florida. In this table, the column names CA and FL are values of the variable state. Therefore, we can say that this table is in an untidy format:

table4

```
##           candidate    CA     FL
## 1: Hillary Clinton 5931283 4485745
## 2: Donald Trump    3184721 4605515
## 3: Gary Johnson    308392  206007
## 4: Jill Stein      166311  64019
```

candidate	state	votes	candidate	CA	FL
Hillary Clinton	CA	5931283	Hillary Clinton	5931283	4485745
Donald Trump	CA	3184721	Donald Trump	3184721	4605515
Gary Johnson	CA	308392	Gary Johnson	308392	206007
Jill Stein	CA	166311	Jill Stein	166311	64019
Hillary Clinton	FL	4485745			
Donald Trump	FL	4605515			
Gary Johnson	FL	206007			
Jill Stein	FL	64019			

table4

This can be achieved by using the **data.table** function `melt()`:

```
melt(table4,
  id.vars = "candidate",
  measure.vars = c("CA", "FL"),
  variable.name = "state",
  value.name = "votes")

##      candidate state  votes
## 1: Hillary Clinton    CA 5931283
## 2: Donald Trump       CA 3184721
## 3: Gary Johnson       CA 308392
## 4: Jill Stein         CA 166311
## 5: Hillary Clinton    FL 4485745
## 6: Donald Trump       FL 4605515
## 7: Gary Johnson        FL 206007
## 8: Jill Stein          FL 64019
```

We remark that the previous chunk of code would work as well without specifying either `measure.vars` OR `id.vars`. However, specifying neither will not work.

When melting, all values in the columns specified by the `measure.vars` argument are gathered into one column whose name can be specified using the `value.name` argument. Additionally, a new column, which can be named using the argument `variable.name`, is created containing all values which were previously stored in the column names.

Now we have a table in a tidy format where a row represents the number of votes for a candidate in a state. The new table also makes clear that the quantities are numbers of votes thanks to the column name.

3.3.2 Casting (long to wide)

The other way around also happens frequently. It is helpful when multiple variables are stored in one column. In the table below, multiple values, namely the number of votes for a candidate and the total number of votes, are reported in one column. It is not easy to compute the percentage of votes given to a candidate in this format. To tidy up this table we have to separate those values into two columns:

```
table2
```

```
##      candidate state      type  value
## 1: Hillary Clinton    CA   votes 5931283
## 2: Hillary Clinton    CA total_votes 9631972
## 3: Donald Trump       CA   votes 3184721
## 4: Donald Trump       CA total_votes 9631972
## 5: Gary Johnson       CA   votes 308392
## 6: Gary Johnson       CA total_votes 9631972
```

The diagram illustrates the process of casting a long table into a wide table. On the left, 'table2' is shown as a long table with four columns: candidate, state, type, and value. The data rows are: Hillary Clinton (CA, votes: 5931283), Hillary Clinton (CA, total_votes: 9631972), Donald Trump (CA, votes: 3184721), Donald Trump (CA, total_votes: 9631972), Gary Johnson (CA, votes: 308392), and Gary Johnson (CA, total_votes: 9631972). On the right, the resulting wide table has four columns: candidate, state, votes, and total_votes. The data rows are: Hillary Clinton (CA, 5931283, 9631972), Donald Trump (CA, 3184721, 9631972), and Gary Johnson (CA, 308392, 9631972). Arrows indicate the mapping from the 'value' column of table2 to the 'votes' column of the wide table.

candidate	state	type	value	candidate	state	votes	total votes
Hillary Clinton	CA	votes	5931283	Hillary Clinton	CA	5931283	9631972
Hillary Clinton	CA	total_votes	9631972	Donald Trump	CA	3184721	9631972
Donald Trump	CA	votes	3184721	Gary Johnson	CA	308392	9631972
Donald Trump	CA	total_votes	9631972				
Gary Johnson	CA	votes	308392				
Gary Johnson	CA	total_votes	9631972				

table2

Casting the election dataset

This operation, which transforms a long table into a wide table is called casting, following up with the metal forging analogy employed with the term “melting”.

Data table casting can be achieved using the `dcast()` function whose most frequent usage is:

```
dcast(data, formula, value.var = guess(data))
```

Casting requires specifying which column contains the categories by which the new columns should be created. This is provided via the `formula` argument. Setting `formula` to be `... ~ type` instructs `dcast` to `create new columns` in the table `containing the categories named in the column “type”` and that all other columns will get rearranged accordingly. Furthermore, we force the argument `value.var`, which refers to `which column the values have to be extracted from`, to be “value” rather than letting `dcast` guessing as by default. The call is then:

```
dcast(table2, ... ~ type, value.var = "value")
```

```
##           candidate state total_votes   votes
## 1:    Donald Trump   CA      9631972 3184721
## 2:    Gary Johnson   CA      9631972 308392
## 3: Hillary Clinton   CA      9631972 5931283
```

The function `dcast` has many more arguments. Also the formula can allow for more sophisticated ways to handle the columns. We refer to the “datatable-reshape” vignette (see section Resources) and the help of `dcast()` for more details.

3.3.3 Separating columns

Sometimes `single variables can be spread across multiple columns` as in the following table.

```
## One column contains multiple variables
print(table3)
```

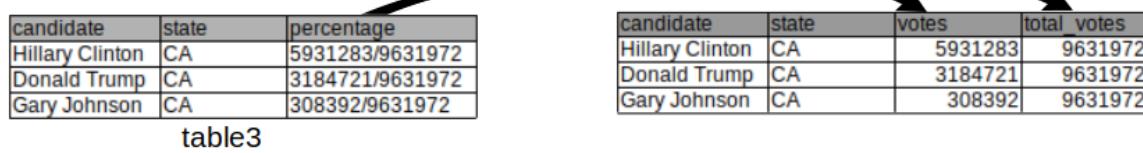
```
##           candidate state      proportion
## 1: Hillary Clinton   CA 5931283/9631972
## 2: Donald Trump     CA 3184721/9631972
## 3: Gary Johnson     CA  308392/9631972
```

The number of votes per candidate is displayed in the numerator of the `proportion` column and the total number in the denominator.

We can solve both problems using the `separate()` function from the `tidyverse` package. The code below splits up the `proportion` column into two columns, one containing the votes and the other one containing the total votes. By default, columns are separated by any non-alphanumeric character (such as “,”, “;”, “/”,...).

```
separate(table3, col = proportion,
         into = c("votes", "total_votes"))

##           candidate state      votes total_votes
## 1: Hillary Clinton   CA    5931283     9631972
## 2: Donald Trump     CA    3184721     9631972
## 3: Gary Johnson     CA    308392      9631972
```



The drawing above visualizes the operation performed above.

3.3.4 Uniting columns

In this example the first and last names are separated columns without a real need for it (we will not be interested in computing any statistics over all Hillary's):

```
table5
```

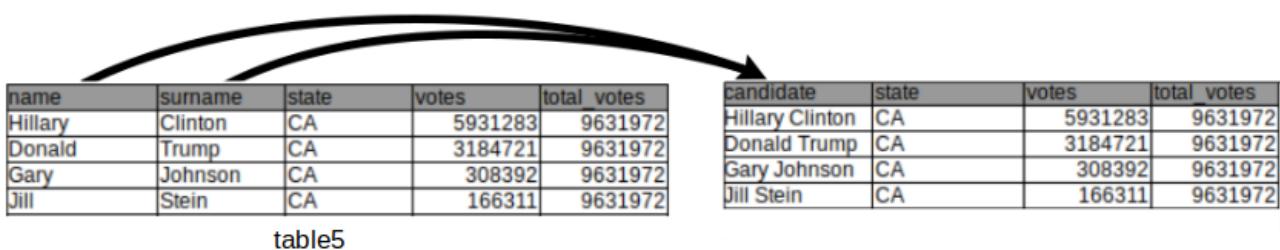
```
##      name surname state   votes total_votes
## 1: Hillary Clinton    CA 5931283    9631972
## 2: Donald Trump       CA 3184721    9631972
## 3: Gary Johnson       CA 308392     9631972
## 4: Jill Stein         CA 166311     9631972
## 5: Gloria La Riva     CA  41265     9631972
```

We unite **multiple variables into a single variable** with the function `unite()` from the `tidyverse` package:

```
unite(table5, col = candidate, name, surname, sep = " ")
```

```
##      candidate state   votes total_votes
## 1: Hillary Clinton    CA 5931283    9631972
## 2: Donald Trump       CA 3184721    9631972
## 3: Gary Johnson       CA 308392     9631972
## 4: Jill Stein         CA 166311     9631972
## 5: Gloria La Riva     CA  41265     9631972
```

The `sep` argument defines the **separating character(s)** used to unite the different column values into one.



United election dataset

3.3.5 Advanced: Columns containing sets of values

Kaggle, a machine learning platform, conducts a yearly survey among its users. Below are a few columns of the answers from the 2017 survey. In those columns, we observe another type of untidy data. In this survey, multiple choice questions were asked from which multiple answers could be selected. For each individual the selected answers are concatenated into a string.

```
options(width = 60)
survey <- fread('extdata/kaggle-survey-2017/multipleChoiceResponses.csv')
survey[, .(LanguageRecommendationSelect, LearningPlatformSelect, PastJobTitlesSelect)]
```

```
##      LanguageRecommendationSelect
## 1:                  F#
## 2:                  Python
## 3:                   R
## 4:                  Python
## 5:                  Python
## ---
## 16712:
## 16713:                  Python
## 16714:
## 16715:
## 16716:
##                                         LearningPl
## 1:                                         College/University,Conferences,Podcast
## 2:
## 3:                                         Arxiv,College/University,Kaggle,Online courses,Yo
## 4: Blogs,College/University,Conferences,Friends network,Official documentation,Online courses,Perso
## 5:                                         Arxiv,Conferences,Kag
## ---
## 16712:
## 16713: Kaggle,Non-Kaggle online communities,Online courses,Stack Overflow Q&A,Yo
## 16714:
## 16715:
## 16716:
## 
## 1:
## 2:
## 3:
## 4:                                         Business Ar
## 5: Computer Scientist,Data Analyst,Data Miner,Data Scientist,Engineer,Machine Learning Engineer,Pre
## ---
## 16712:
## 16713:
## 16714:
## 16715:
## 16716:
```

Below is one solution, of how the `LearningPlatformSelect` column could be transformed into a tidy format.

```

survey_split <- survey[,tstrsplit(LearningPlatformSelect, ',')]
survey_split[, individual := 1:nrow(survey)]
LearningPlatformMelt <- melt(survey_split,
                           id.vars = 'individual',
                           na.rm = TRUE)[, variable := NULL]

LearningPlatformMelt[order(individual)] %>% head(n=5)

##      individual      value
## 1:          1 College/University
## 2:          1        Conferences
## 3:          1       Podcasts
## 4:          1     Trade book
## 5:          2       Kaggle

```

3.4 Concatenating tables

One frequently has to concatenate (i.e. append) tables with a same format. Such tables may already be loaded into a list or shall be read from multiple files.

For instance, assume a service generates a new file of data per day in a given directory. One is interested in analyzing the files of multiple days jointly. This requires to list all files of the directory, to read each file and to concatenate them into one.

Here is an example with daily COVID-19 data. We first get all file names of the directory into a list called `files` :

```

files <- list.files('path_to_your_directory')

files <- list.files('extdata/cov_concatenate', full.names = TRUE)
head(files)

## [1] "extdata/cov_concatenate/covid_cases_01_03_2020.csv"
## [2] "extdata/cov_concatenate/covid_cases_02_03_2020.csv"
## [3] "extdata/cov_concatenate/covid_cases_03_03_2020.csv"
## [4] "extdata/cov_concatenate/covid_cases_04_03_2020.csv"
## [5] "extdata/cov_concatenate/covid_cases_05_03_2020.csv"
## [6] "extdata/cov_concatenate/covid_cases_06_03_2020.csv"

```

Next, we load all file contents with `fread` using `lapply`, which passes the function `fread` to every element in the list `files` and returns a list of `data.tables` called `tables`.

```
# name the list elements by the filenames
names(files) <- basename(files)

# read all files at once into a list of data.tables
tables <- lapply(files, fread)
```

Let us know look at the first table:

```
head(tables[[1]])

##      cases deaths countriesAndTerritories geoId
## 1:     54      0             Germany       DE
## 2:   240      8             Italy        IT
##      countryterritoryCode popData2019 continentExp
## 1:           DEU    83019213      Europe
## 2:           ITA    60359546      Europe
##      Cumulative_number_for_14_days_of_COVID-19_cases_per_100000
## 1:                           0.1156359
## 2:                         1.8638311
```

We notice that the variable data is only encoded in the filepath so that we additionally need to introduce a new variable in the new table, which defines, from which list the original table came from. We do this to avoid loosing information. In this manner, we can state which case / population numbers came from which country.

To do so, we can use the `data.table` function `rbindlist()` which gives us the option to introduce a new column `idcol` containing the list names:

```
# bind all tables into one using rbindlist,
# keeping the list names (the filenames) as an id column.
dt <- rbindlist(tables, idcol = 'filepath')
head(dt)
```

```

##               filepath cases deaths
## 1: covid_cases_01_03_2020.csv     54      0
## 2: covid_cases_01_03_2020.csv    240      8
## 3: covid_cases_02_03_2020.csv    18      0
## 4: covid_cases_02_03_2020.csv   561      6
## 5: covid_cases_03_03_2020.csv    28      0
## 6: covid_cases_03_03_2020.csv   347     17
##   countriesAndTerritories geoId countryterritoryCode
## 1:           Germany     DE        DEU
## 2:           Italy       IT        ITA
## 3:           Germany     DE        DEU
## 4:           Italy       IT        ITA
## 5:           Germany     DE        DEU
## 6:           Italy       IT        ITA
##   popData2019 continentExp
## 1: 83019213        Europe
## 2: 60359546        Europe
## 3: 83019213        Europe
## 4: 60359546        Europe
## 5: 83019213        Europe
## 6: 60359546        Europe
##   Cumulative_number_for_14_days_of_COVID-19_cases_per_100000
## 1:                               0.1156359
## 2:                             1.8638311
## 3:                            0.1373176
## 4:                            2.7932616
## 5:                            0.1710447
## 6:                            3.3681499

```

3.5 Merging tables

Merging two data tables into one by common column(s) is frequently needed. This can be achieved using the `merge` function whose core signature is:

```

merge(
  x, y,                                     # tables to merge
  by = NULL, by.x = NULL, by.y = NULL,        # by which columns
  all = FALSE, all.x = all, all.y = all        # types of merge
)

```

The four types of merges (also commonly called joins) are:

- **Inner (default)**: consider only rows with matching values in the `by` columns.
- **Outer or full (all)**: return all rows and columns from `x` and `y`. If there are no matching values, return NAs.

- **Left (all.x):** consider all rows from `x`, even if they have no matching row in `y`.
- **Right (all.y):** consider all rows from `y`, even if they have no matching row in `x`.

We now provide examples of each type using the following made up tables:

```
dt1 <- data.table(p_id = c("G008", "F027", "L051"),
                   value = rnorm(3))

dt1

##      p_id     value
## 1: G008  0.5271984
## 2: F027  1.7752485
## 3: L051  0.5082788

dt2 <- data.table(p_id = c("G008", "F027", "U093"),
                   country = c("Germany", "France", "USA"))

dt2

##      p_id country
## 1: G008 Germany
## 2: F027 France
## 3: U093 USA
```

3.5.1 Inner merge

An inner merge returns only rows with matching values in the `by` columns and discards all other rows:

```
# Inner merge, default one, all = FALSE
m <- merge(dt1, dt2, by = "p_id", all = FALSE)

m

##      p_id     value country
## 1: F027  1.7752485 France
## 2: G008  0.5271984 Germany
```

Note that the row order got changed after the merging. To prevent this and, therefore, to keep the original ordering we can use the argument `sort` and set it to `FALSE`:

```
m <- merge(dt1, dt2, by = "p_id", all = FALSE, sort = FALSE)

m
```

```
##   p_id      value country
## 1: G008 0.5271984 Germany
## 2: F027 1.7752485 France
```

Note that the row order got changed after the merging.

3.5.2 Outer (full) merge

An outer merge returns all rows and columns from `x` and `y`. If there are no matching values, it yields missing values (`NA`):

```
# Outer (full) merge, all = TRUE
merge(dt1, dt2, by = "p_id", all = TRUE)
```

```
##   p_id      value country
## 1: F027 1.7752485 France
## 2: G008 0.5271984 Germany
## 3: L051 0.5082788 <NA>
## 4: U093       NA    USA
```

3.5.3 Left merge

Returns all rows from `x`, even if they have no matching row in `y`. Rows from `x` with no matching in `y` lead to missing values (`NA`).

```
# Left merge, all.x = TRUE
merge(dt1, dt2, by = "p_id", all.x = TRUE)
```

```
##   p_id      value country
## 1: F027 1.7752485 France
## 2: G008 0.5271984 Germany
## 3: L051 0.5082788 <NA>
```

3.5.4 Right merge

Returns all rows from `y`, even if they have no matching row in `x`. Rows from `y` with no matching in `x` lead to missing values (`NA`).

```
# Right, all.y = TRUE
merge(dt1, dt2, by = "p_id", all.y = TRUE)
```

```
##      p_id      value country
## 1: F027 1.7752485 France
## 2: G008 0.5271984 Germany
## 3: U093       NA     USA
```

3.5.5 Merging by several columns

Merging can also be done using several columns. Here are two made-up tables to illustrate this use case:

```
dt1 <- data.table(firstname = c("Alice", "Alice", "Bob"),
                  lastname = c("Coop", "Smith", "Smith"), x=1:3)
dt1
```

```
##      firstname lastname x
## 1:      Alice     Coop 1
## 2:      Alice    Smith 2
## 3:       Bob    Smith 3
```

```
dt2 <- data.table(firstname = c("Alice", "Bob", "Bob"),
                  lastname = c("Coop", "Marley", "Smith"),
                  y=LETTERS[1:3])
dt2
```

```
##      firstname lastname y
## 1:      Alice     Coop A
## 2:       Bob    Marley B
## 3:       Bob    Smith C
```

We merge now `dt1` and `dt2` by first name and last name:

```
merge(dt1, dt2, by=c("firstname", "lastname"))
```

```
##      firstname lastname x y
## 1:      Alice     Coop 1 A
## 2:       Bob    Smith 3 C
```

Notice that merging by first name only gives a different result (as expected):

```
merge(dt1, dt2, by="firstname")
```

```
##   firstname lastname.x x lastname.y y
## 1: Alice     Coop 1     Coop A
## 2: Alice     Smith 2    Coop A
## 3: Bob      Smith 3   Marley B
## 4: Bob      Smith 3  Smith C
```

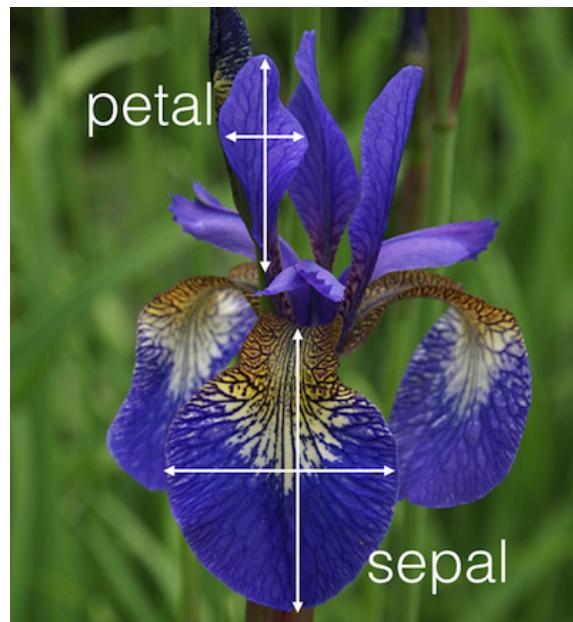
Also notice that in this case the merge tables has a column `lastname.x` and a column `lastname.y`. This is because the two original data tables have a column named the same way (“lastname”), but this column was not part of the “by” argument. Hence, it is assumed that they do not necessarily correspond to the same variable. Hence, they receive distinct names in the returned table.

3.6 Tidy representations are not unique

While untidy data should be avoided, there can be multiple tidy representations for a particular dataset. We explain this regarding i) alternative forms of a single table and ii) the practical utility of non-normalized representations (i.e. with redundant information).

3.6.1 Alternative tidy forms of a table

There can be alternative tidy representations for a same table. Here is an example based on Fisher’s Iris dataset. This classic dataset contains measurements of 4 different attributes for 150 iris flowers from 3 different species. See https://en.wikipedia.org/wiki/Iris_flower_data_set.



Here is one tidy representation where each row represents one flower:

```
# Iris dataset, usual representation
iris_dt[1:3,]
```

```

##      Flower Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1:    F_1          5.1         3.5        1.4       0.2
## 2:    F_2          4.9         3.0        1.4       0.2
## 3:    F_3          4.7         3.2        1.3       0.2
##      Species
## 1:  setosa
## 2:  setosa
## 3:  setosa

```

Here is another tidy representation where each row represents one measurement:

```

# Another tidy representation
iris_melt[1:3,]

##      Flower Species   Attribute value
## 1:    F_1  setosa Sepal.Length     5.1
## 2:    F_2  setosa Sepal.Length     4.9
## 3:    F_3  setosa Sepal.Length     4.7

```

Both representations are **tidy** and can be more or less useful **depending on the downstream analysis**. For instance the first, wide, representation is handy to assess the relationship between sepal length and sepal width, say by plotting one against the other one or by computing their correlations. The second, long, representation can be useful to compute means by attributes or by attributes and species. In the wide form, computing those group means would require to select columns by names which is tedious and leads to not-easily maintainable code. The decisive criteria between using one or the other tidy representation is the definition on what is considered as an observation in the use case.

3.6.2 On multiple types of observational units in the same table

Another important remark for handling tidy data in practice relates to the last common sign of messy datasets according to Wickham (2014), i.e. “Multiple types of observational units are stored in the same table”. Applying this criteria actually depends on the context.

Consider the following table which combines product and customer data:

```

##      productCode quant0rdered price customerNumber
## 1:      p018            1    450           c001
## 2:      p030            2    600           c001
## 3:      p018            1    450           c002
##      customerName state
## 1:      Smith     CA
## 2:      Smith     CA
## 3:      Lewis     AZ

```

This table is tidy. Each row corresponds to an order. The columns are variables. However, it contains repetitive information: the product code, product name and its price on the one hand, the customer number, name and state on the other hand. The information could be stored in separate tables without data repetitions, namely:

- a consumer table:

```
##   customerNumber customerName state
## 1:          c001      Smith    CA
## 2:          c002     Lewis    AZ
```

- a product table:

```
##   productCode price
## 1:      p018    450
## 2:      p030    600
```

- an order table:

```
##   productCode customerNumber quantOrdered
## 1:      p018          c001            1
## 2:      p030          c001            2
## 3:      p018          c002            1
```

The three-table representation, where each table has unique entries is called a normalized representation. Normalized representations ensure that no multiple types of observational units are stored in the same table. It is a good habit to have normalized representations for database back-ends because it facilitates maintenance of the data consistency by reducing redundancy. One should not enter all customer details at each order but do it one central place and link the information with a customer number.

However, on the data analysis side (front-end), we are not interested in maintaining a database (back-end), rather in having the desired data in a ready-to-use format which depends on our needs. To this end, the merge table is very handy and can be the common denominator of multiple analyses like:

```
# vectorized operations e.g. total price of each order
prod_dt[, totalPrice := quantOrdered * price]

# group by operations, e.g. number of products per states
prod_dt[, N_prod := .N, by = state]
```

Hence, the choice of the representation (normalized or not) depends on the context: back-end or front-end.

3.7 Summary

By now, you should be able to:

- define what a tidy dataset is
- recognize untidy data
- perform the operations of melting and casting
- perform the operations of uniting and splitting
- append tables with the same format by rows
- understand and perform the 4 merging operations

3.8 Tidy data resources

Tidy data: H. Wickham, Journal of Statistical Software, 2014, Volume 59, Issue 10

<https://www.jstatsoft.org/v59/i10/paper>

Melt and cast: <https://cran.r-project.org/web/packages/data.table/vignettes/datatable-reshape.html>

References

Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software, Articles* 59 (10): 1–23.

<https://doi.org/10.18637/jss.v059.i10>.

3. <https://www.kaggle.com/stevepalley/2016uspresidentialvotebycounty?select=pres16results.csv> ↩

Chapter 4 Low dimensional visualizations

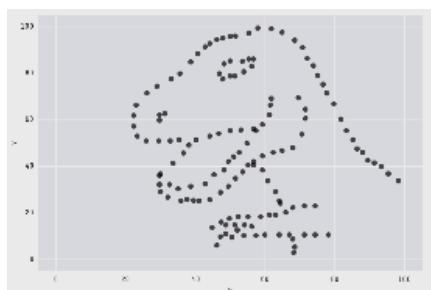
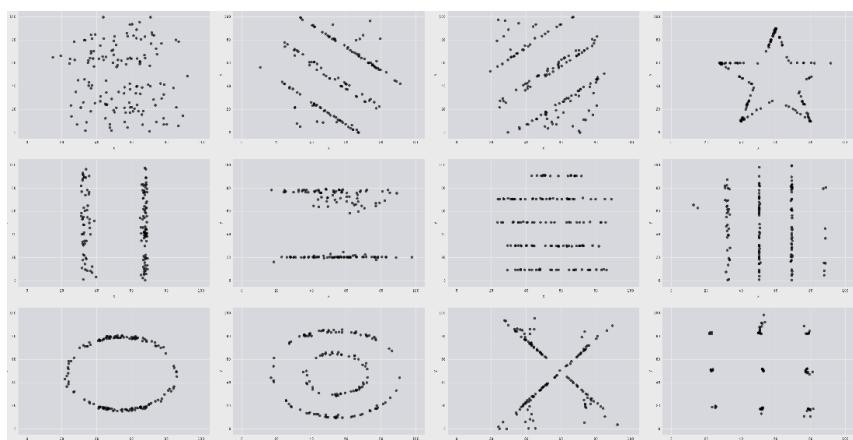
4.1 Why plotting?

Plotting is crucial to data science because:

- It facilitates making **new observations** by discovering associations or patterns in the data (the initial step of the scientific method⁴). The human brain is particularly good at detecting patterns in images, this what we evolved for. Visual display, over staring at table of numbers is very effective
- It facilitates **communicating findings**
- Only relying on summary statistics (mean, correlation, etc.) is dangerous. **Summary statistics reduce data to a single number, therefore carry much less information** than 2D representations. Section 4.1.1 provide examples
- It **helps debugging** either the code by visually checking whether particular operations performed as expected on the data, or by identifying “**bugs in the data**” such as wrong entries or outliers. Section 4.1.2 provides an example

4.1.1 Plotting versus summary statistics

What do those 13 datasets have in common?



All those plots, including the infamous datasaurus share the same following statistics:

- X mean: 52.26
- Y mean: 47.83
- X standard deviation: 16.76
- Y standard deviation: 29.93
- Pearson correlation: -0.06

When only looking at the statistics, we would have probably wrongly assumed that the datasets were identical. This example highlights why it is important to visualize data and not just rely on summary statistics. See [\[https://github.com/lockeddata/datasauRus\]](https://github.com/lockeddata/datasauRus) or Anscombe's quartet [\[https://en.wikipedia.org/wiki/Anscombe%27s_quartet\]](https://en.wikipedia.org/wiki/Anscombe%27s_quartet) for more examples.

4.1.2 Plotting helps finding bugs in the data

Consider the following vector `height` containing (hypothetical) height measurements for 500 adults:

```
head(height_dt, n=5)
```

```
##      height
## 1: 1.786833
## 2: 1.715319
## 3: 1.789841
## 4: 1.787259
## 5: 1.748659
```

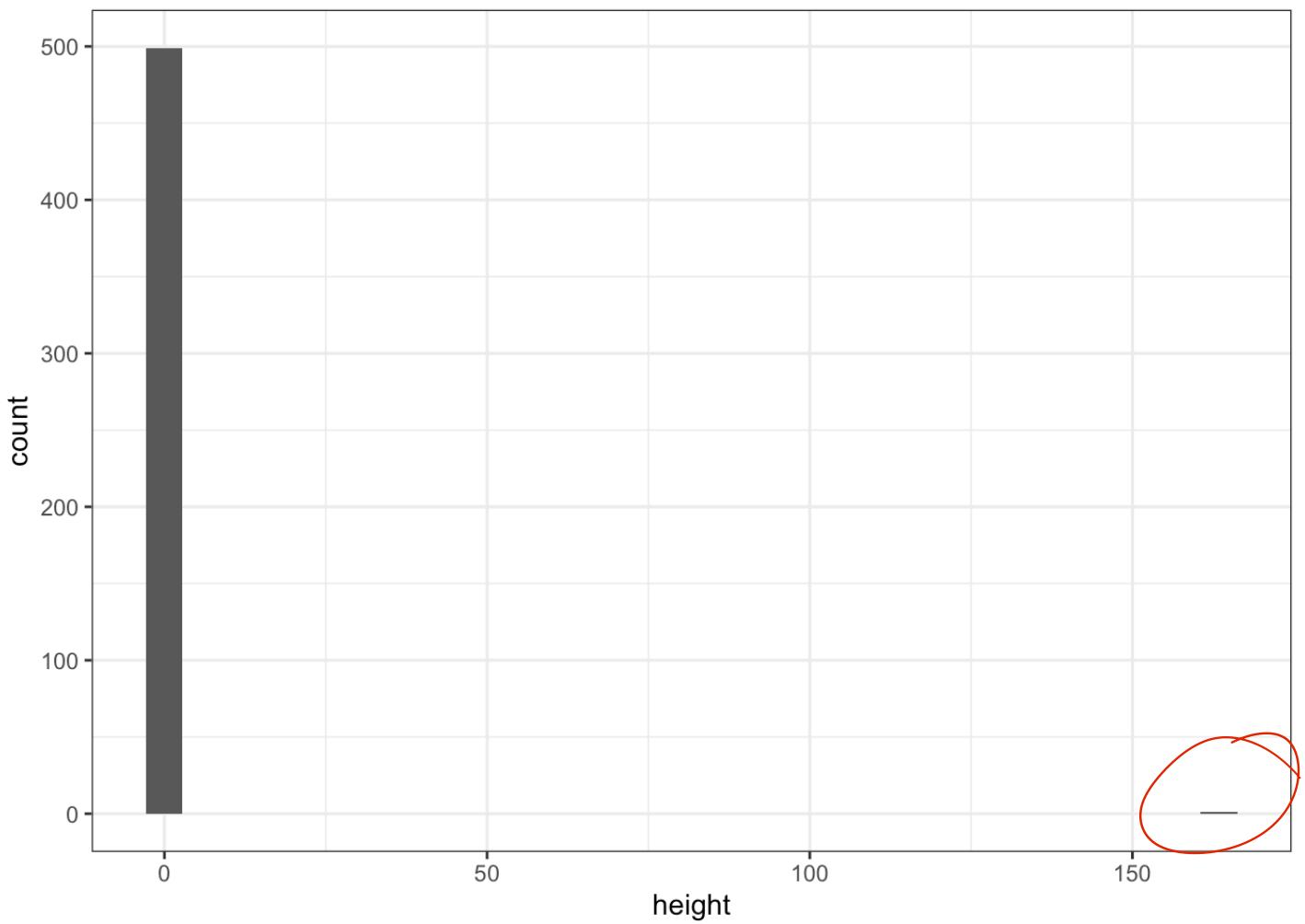
Calculating the mean height returns the following output:

```
height_dt[, mean(height)]
```

```
## [1] 2.056583
```

There is something obviously wrong. We can plot the data to investigate.

```
# You can adjust the number of bins with the bins parameter
ggplot(height_dt, aes(height)) + geom_histogram() + mytheme
```



There is an outlier ($\text{height}=165$). One particular value has probably been entered in centimeters rather than meters. As a result, the mean is inflated.

A quick way to fix our dataset is to remove the outlier, for instance with:

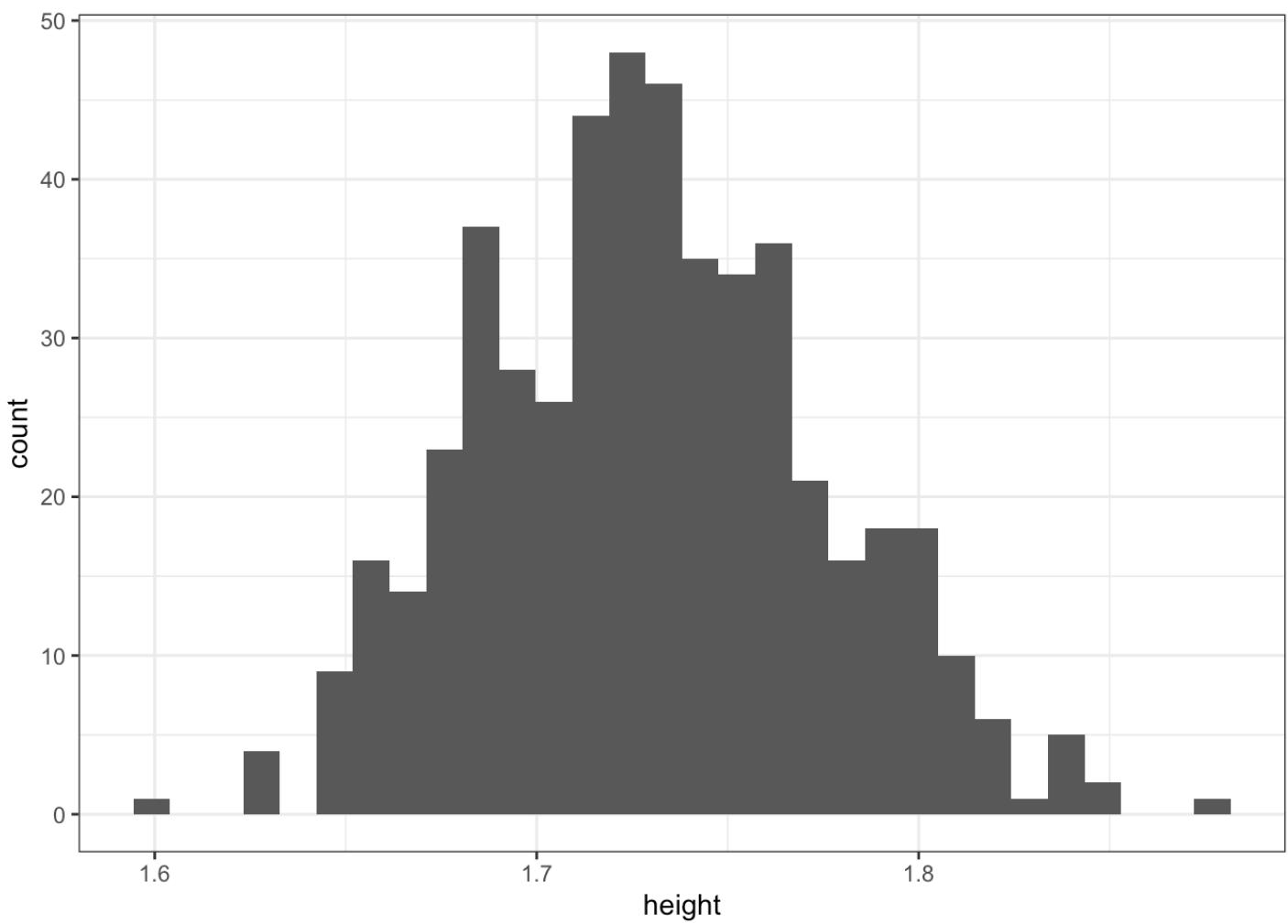
```
height_dt <- height_dt[height < 3]
```

Now our plotted data seems more realistic and the mean height makes sense.

```
height_dt[, mean(height)]
```

```
## [1] 1.730043
```

```
ggplot(height_dt , aes(height)) + geom_histogram() + mytheme
```



While developing analysis scripts, we recommend to frequently visualize the data to make sure no mistake in the input or during the processing occurred.

4.2 Grammar of graphics

The grammar of graphics is a visualization theory developed by Leland Wilkinson in 1999. It has influenced the development of graphics and visualization libraries alike. It is based on 3 key principles:

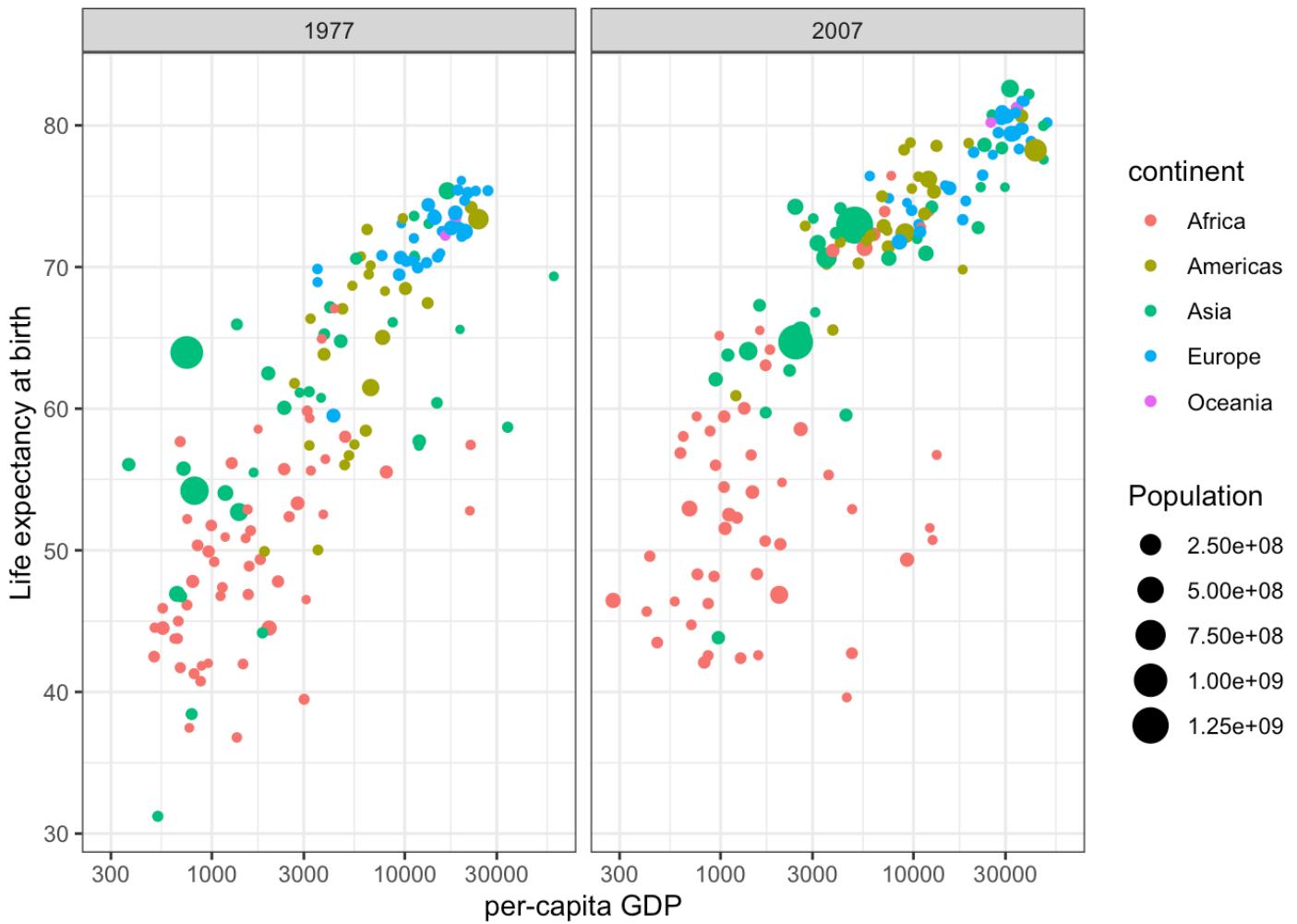
- Separation of data from aesthetics (e.g. x and y-axis, color-coding)
- Definition of common plot/chart elements (e.g. scatter plots, box-plots, etc.)
- Composition of these common elements (one can combine elements as layers)

The library `ggplot2` is a powerful implementation of the grammar of graphics. It has become widely used by R programmers.

Here is a sophisticated motivating example. The plot shows the relationship between per-capita gross domestic product (GDP) and life expectancy at birth for the years 1997 and 2007 from the dataset `gapminder`:

```
#install.packages('gapminder')
library(gapminder)
gm_dt <- as.data.table(gapminder)[year %in% c(1977, 2007)]

ggplot(data=gm_dt, aes(x=gdpPercap, y=lifeExp)) +
  geom_point(aes(color=continent, size=pop)) +
  facet_grid(~year) + scale_x_log10() +
  labs(y="Life expectancy at birth", x="per-capita GDP", size = 'Population') + mytheme
```



We may, for instance, use such visualization to find differences in the life expectancy of each country and each continent.

The following section shows how to create such a sophisticated plot step by step.

4.3 Components of the layered grammar

Grammar of graphics composes plots by combining layers. The major layers are:

- **Always used:**

Data: `data.table` (or `data.frame`) object where columns correspond to variables

Aesthetics: mapping of data to visual characteristics - what we will see on the plot (`aes`) - position (x,y), color, size, shape, transparency

Geometric objects: geometric representation defining the type of the plot data (`geom_`) - points, lines, boxplots, ...

- Often used:

Scales: for each aesthetic, describes how a visual characteristic is converted to display values (`scale_`) - log scales, color scales, size scales, shape scales, ...

Facets: describes how data is split into subsets and displayed as multiple sub graphs (`facet_`)

- Useful, but with care:

Stats: statistical transformations that typically summarize data (`stat`) - counts, means, medians, regression lines, ...

- Domain-specific usage:

Coordinate system: describes 2D space that data is projected onto (`coord_`) - Cartesian coordinates, polar coordinates, map projections, ...

4.3.1 Components of the grammar of graphics

The following components are considered in the context of the grammar of graphics:

Data: `data.table` (or `data.frame`) object where columns correspond to variables

Aesthetics: visual characteristics that represent data (`aes`) - e.g. position, size, color, shape, transparency, fill

Layers: geometric objects that represent data (`geom_`) - e.g. points, lines, polygons, ...

Scales: for each aesthetic, describes how visual characteristic is converted to display values (`scale_`) - e.g. log scales, color scales, size scales, shape scales, ...

Facets: describes how data is split into subsets and displayed as multiple subgraphs (`facet_`)

Stats: statistical transformations that typically summarize data (`stat`) - e.g. counts, means, medians, regression lines, ...

Coordinate system: describes 2D space that data is projected onto (`coord_`) - e.g. Cartesian coordinates, polar coordinates, map projections, ...

4.3.2 Defining the data and layers

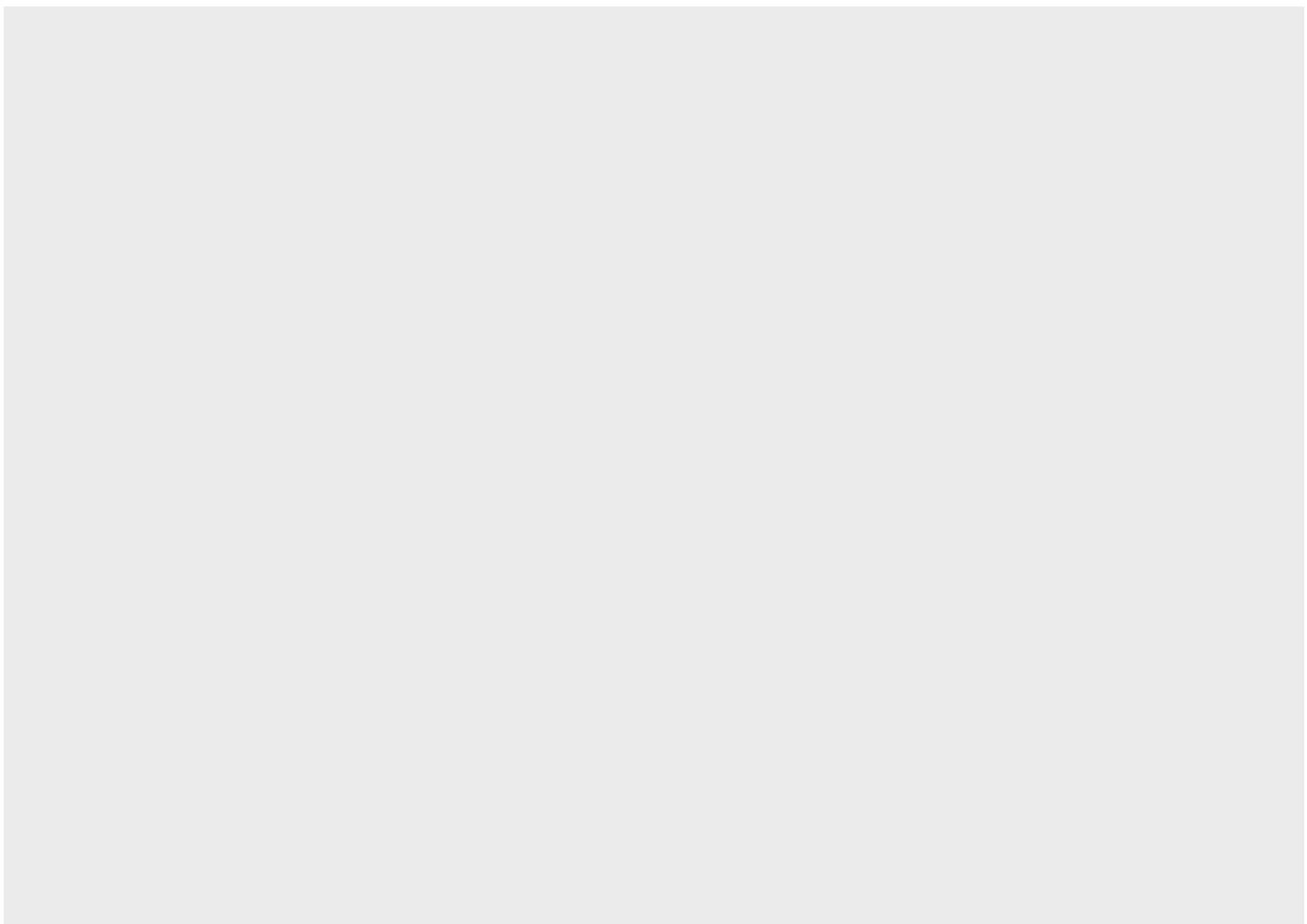
In our example, we consider the `gapminder` dataset, which serves as the data component of our visualization. We want to plot the variable `lifeExp` against the variable `gdpPercap`. First, we have a look at the first lines of the dataset:

```
head(gm_dt[, .(country, continent, gdpPercap, lifeExp, year)])
```

```
##       country continent gdpPercap lifeExp year
## 1: Afghanistan      Asia  786.1134  38.438 1977
## 2: Afghanistan      Asia  974.5803  43.828 2007
## 3:    Albania     Europe 3533.0039  68.930 1977
## 4:    Albania     Europe 5937.0295  76.423 2007
## 5:   Algeria      Africa 4910.4168  58.014 1977
## 6:   Algeria      Africa 6223.3675  72.301 2007
```

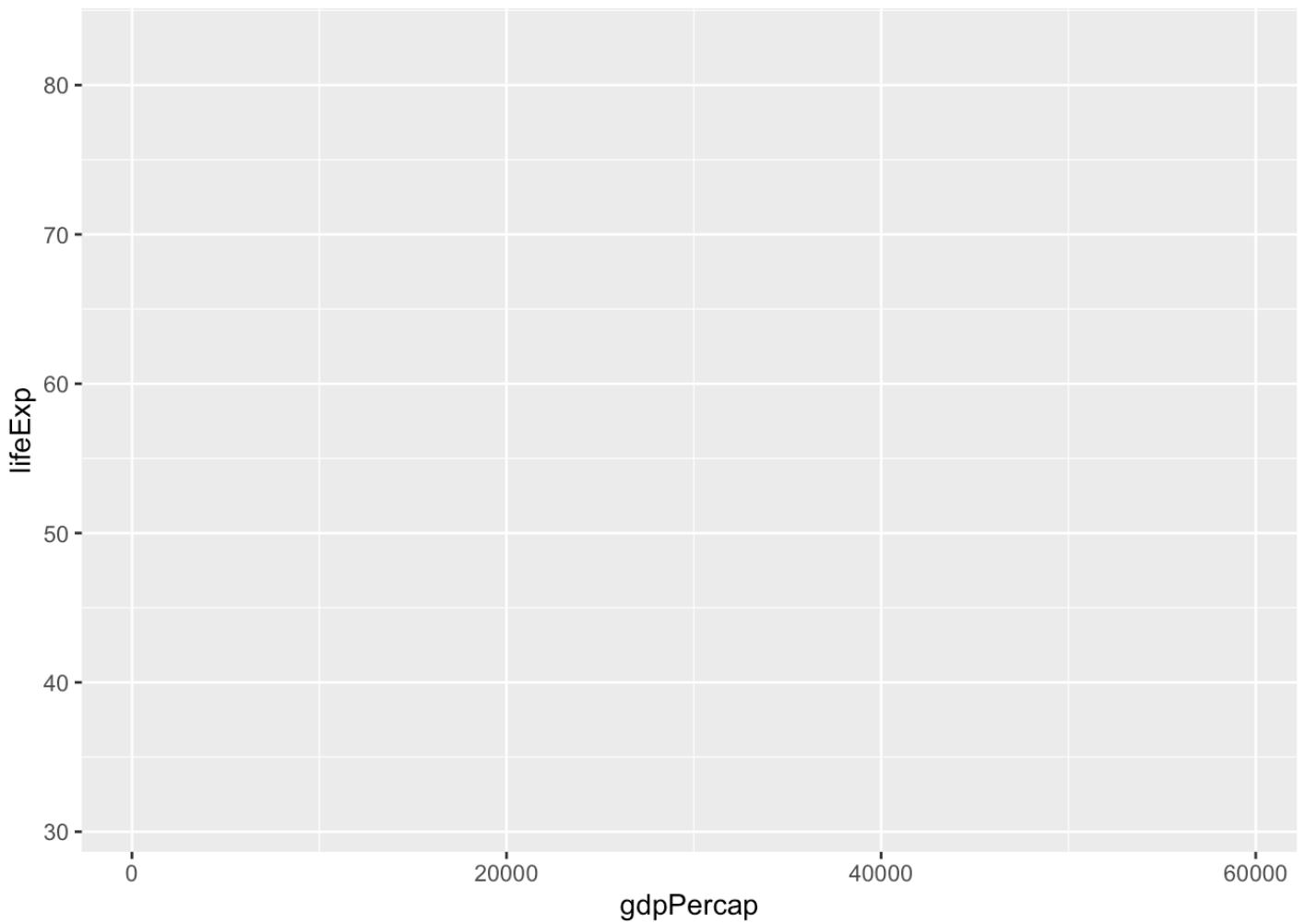
For starting with the visualization we initiate a `ggplot` object which generates a plot with background:

```
ggplot()
```



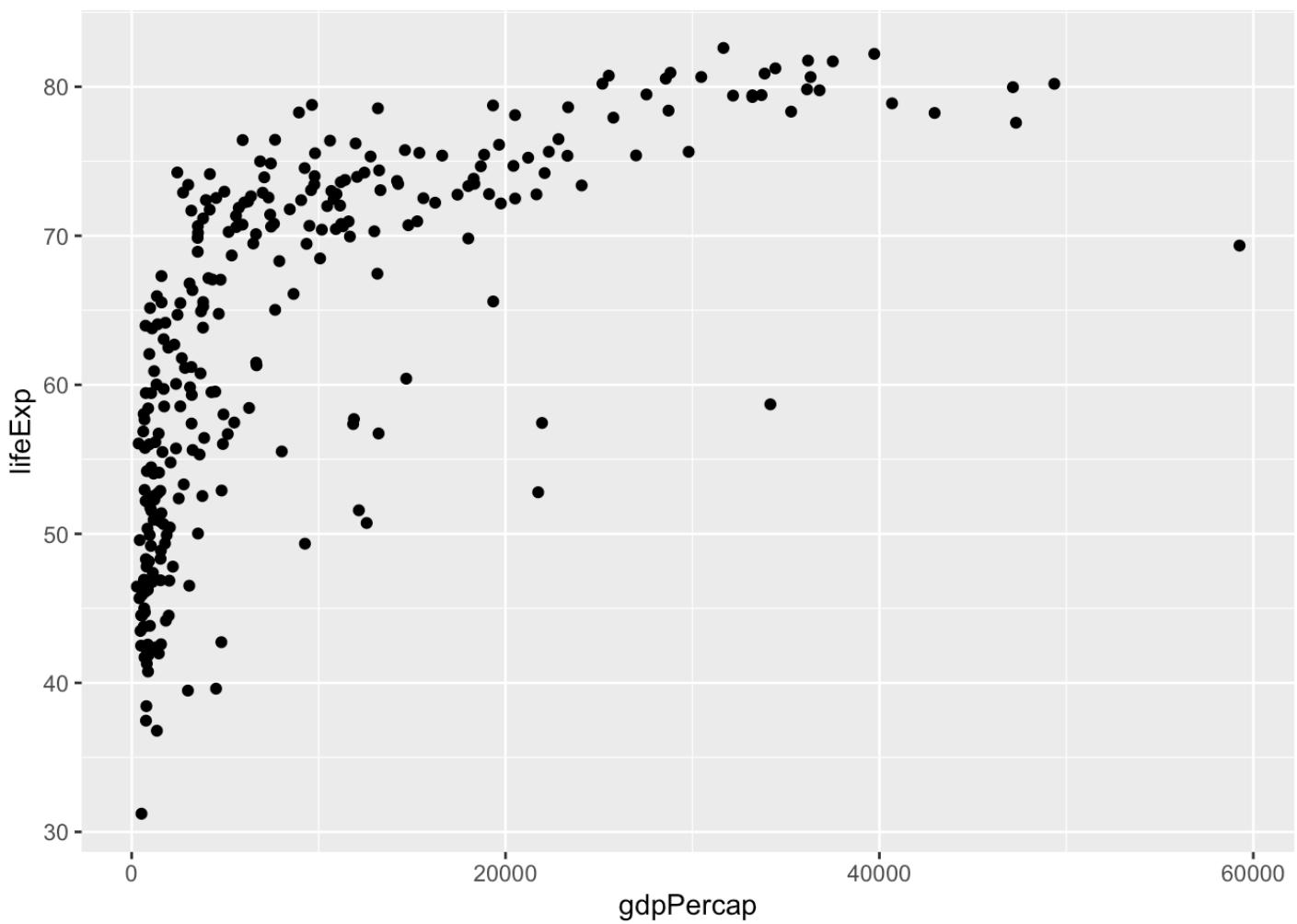
Next, we can define the data to be plotted, which needs to be a `data.table` (or `data.frame`) object and the `aes()` function. This `aes()` function defines which columns in the `data.table` object map to `x` and `y` coordinates and if they should be colored or have different shapes and sizes based on the values in a different column. These elements are called “aesthetic” elements, which we observe in the plot.

```
ggplot(data = gm_dt, aes(x=gdpPercap, y=lifeExp))
```



As we can see, we obtain a plot with labeled axes and ranges. We want to visualize the data with a simple **scatter plot**. In a scatter plot, the values of two variables are plotted along two axes. Each pair of values is represented as a point. In R, a scatter plot can be plotted with `ggplot2` using the function `geom_point`. We want to construct a scatter plot containing the `gdpPercap` on the x-axis and the `lifeExp` on the y-axis. For this we combine the function `geom_point()` to the previous line of code with the operator `+`:

```
ggplot(data = gm_dt, aes(x=gdpPercap, y=lifeExp)) + geom_point()
```



One of the advantages of plotting with `ggplot` is that it returns an object which can be stored (e.g. in a variable called `p`). The stored object can be further edited.

```
p <- ggplot(data = gm_dt, aes(x=gdpPerCap, y=lifeExp)) + geom_point()
```

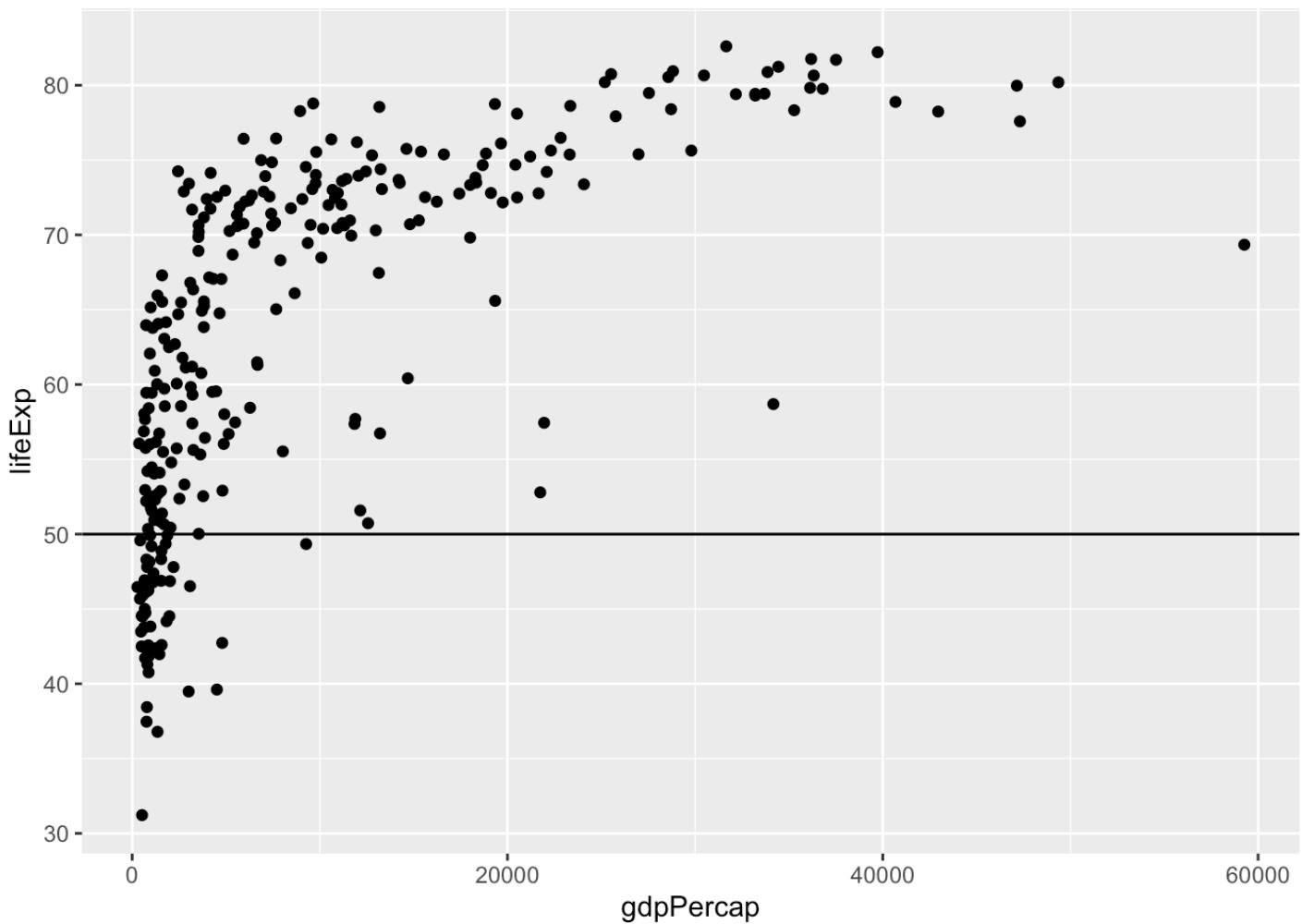
We can inspect the names of elements of the stored object with `names()`:

```
names(p)
```

```
## [1] "data"          "layers"        "scales"        "mapping"
## [5] "theme"         "coordinates"   "facet"         "plot_env"
## [9] "labels"
```

We can also save the `ggplot` object with the help of the function `saveRDS()`. We can read the saved object again in a future R session with the help of the function `readRDS()`. Here is an example where we save a `ggplot` object, then load it, and finally add a horizontal line at `y=50` to it:

```
saveRDS(p, "your/favorite/path/my_first_plot.rds") # save to disk
p2 <- readRDS("your/favorite/path/my_first_plot.rds") # load from disk (can be in another session)
p2 + geom_hline(yintercept = 50)
```

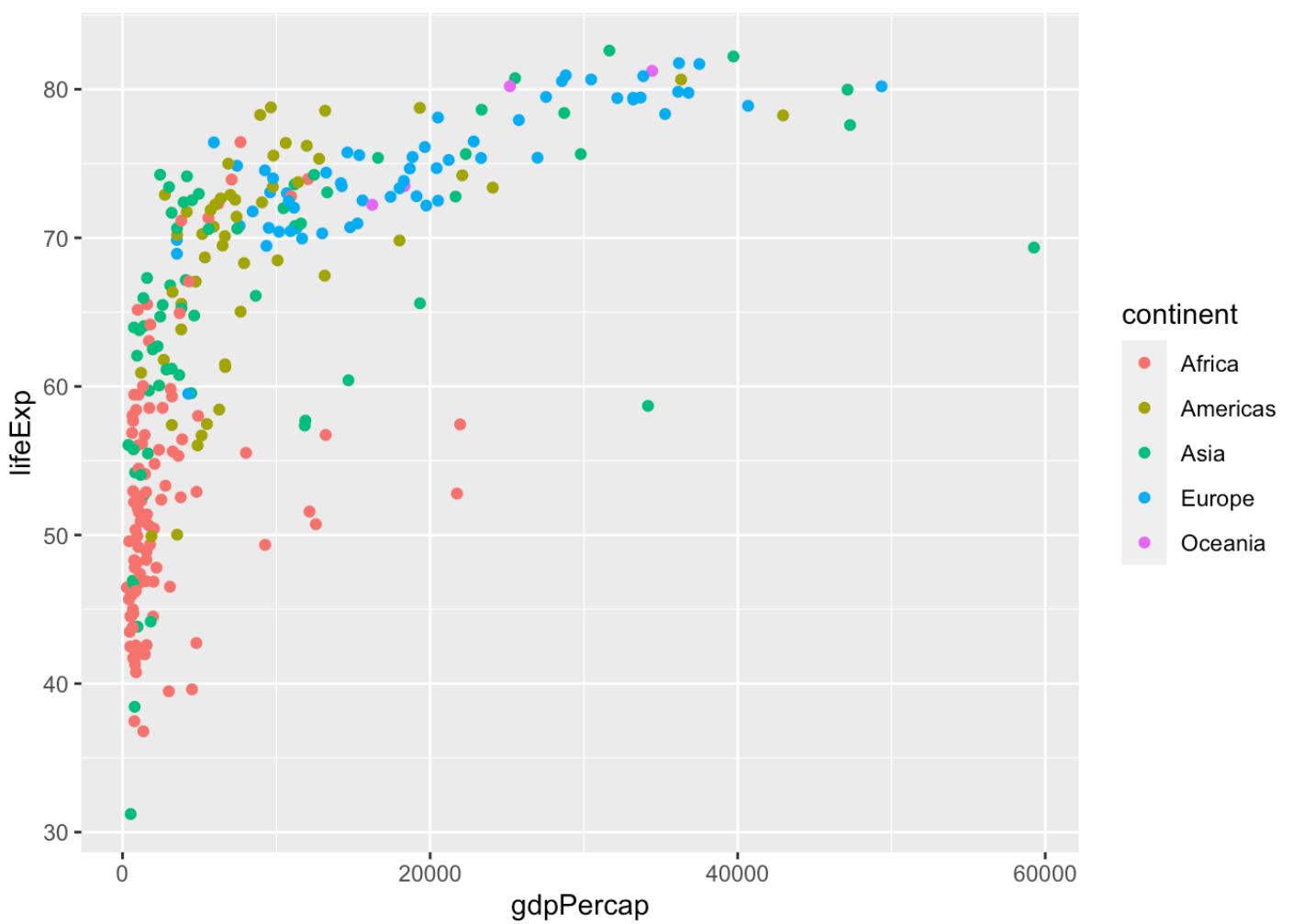


4.3.3 Mapping of aesthetics

4.3.3.1 Mapping of color, shape and size

We can easily map variables to different colors, sizes or shapes depending on the value of the specified variable. To assign each point to its corresponding continent, we can define the variable `continent` as the `color` attribute in `aes()` as follows:

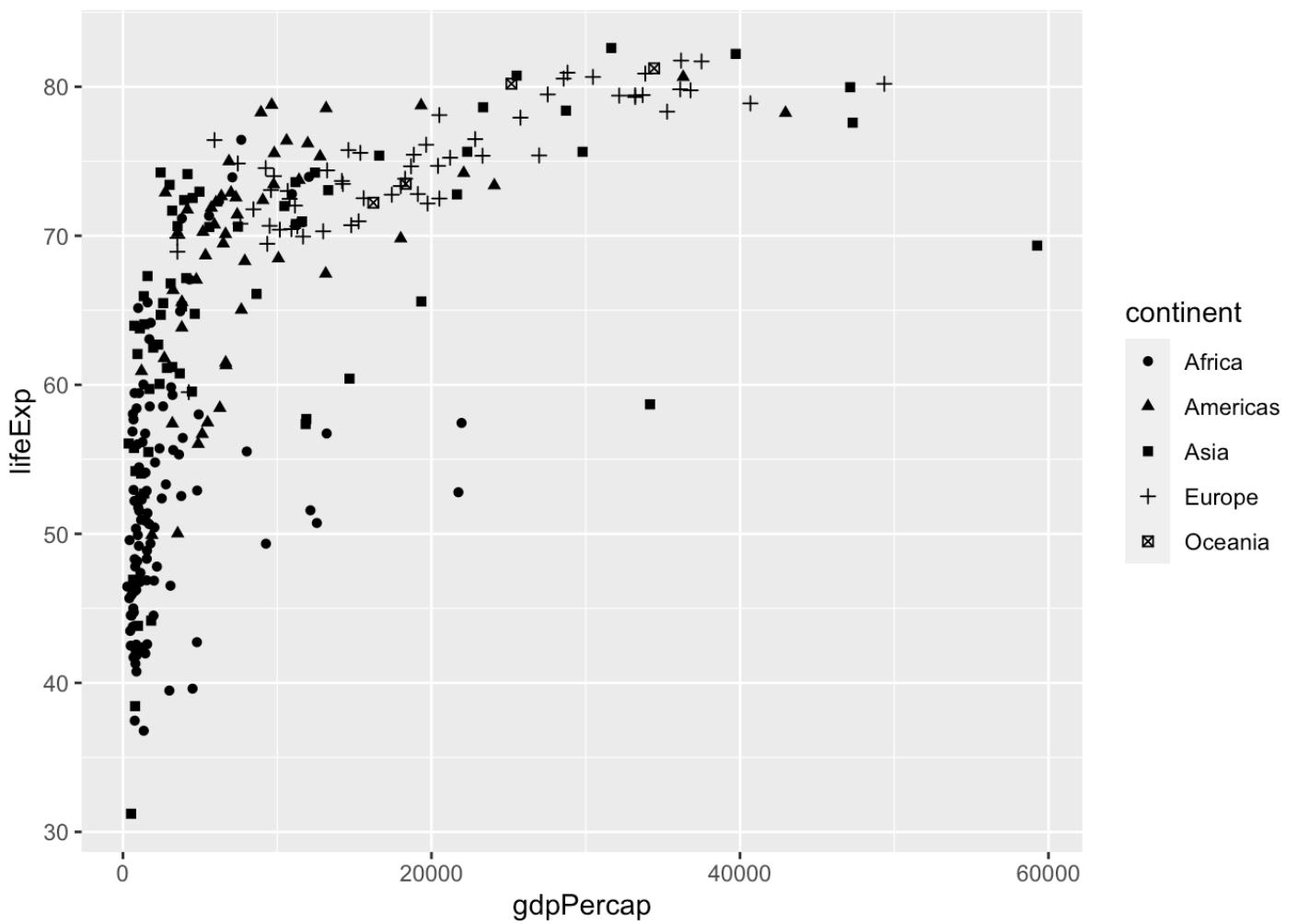
```
ggplot(data = gm_dt, aes(x=gdpPercap, y=lifeExp, color=continent)) + geom_point()
```



American color or British colour are both acceptable as the argument specification.

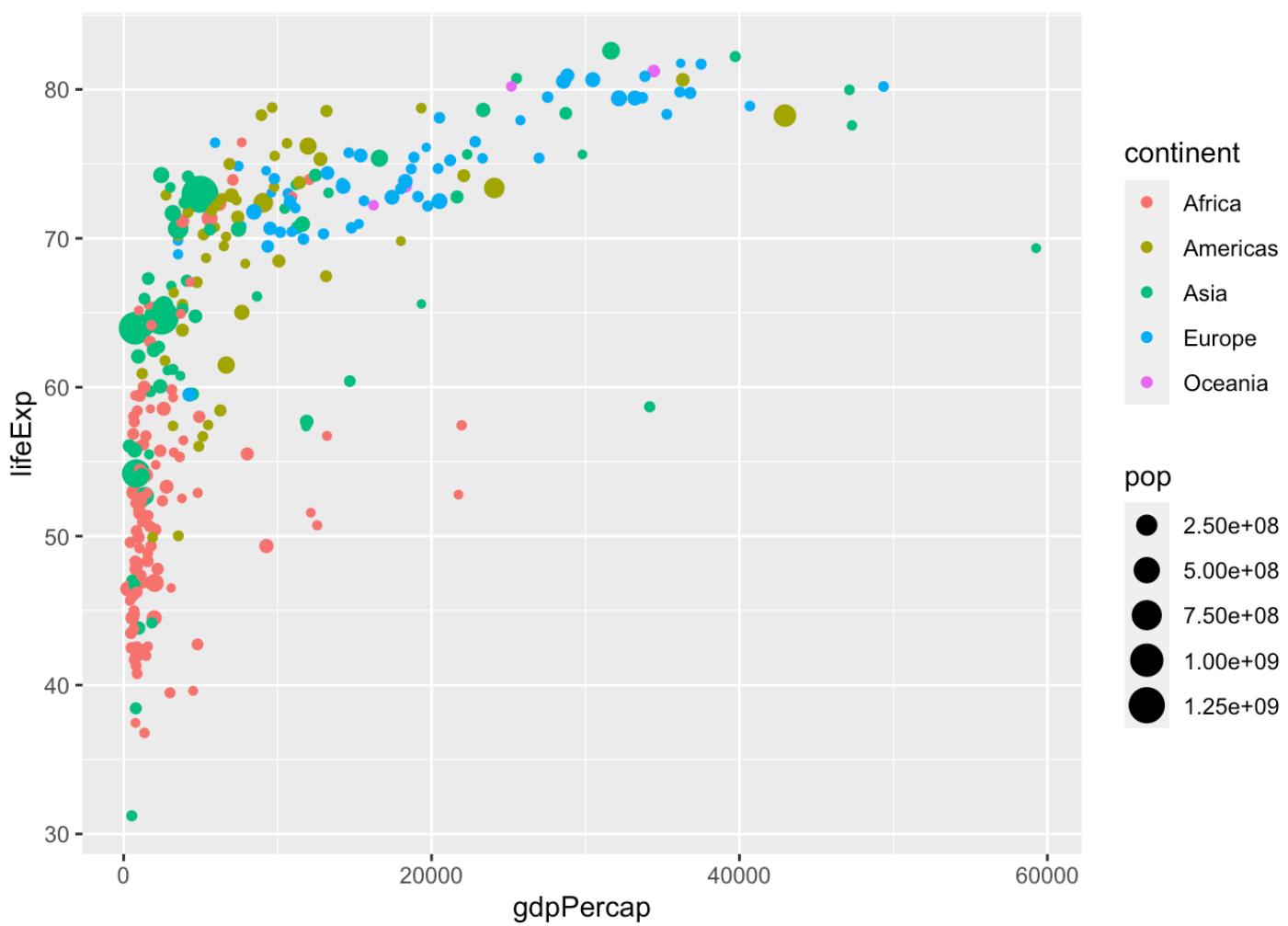
Instead of color, we can also use different shapes for characterizing the different continents in the scatter plot. For this we specify the shape argument in `aes()` as follows:

```
ggplot(data = gm_dt, aes(x=gdpPerCap, y=lifeExp, shape=continent)) + geom_point()
```



Additionally, we distinguish the population of each country by giving a size to the points in the scatter plot:

```
ggplot(data = gm_dt, aes(x=gdpPerCap, y=lifeExp, color=continent, size=pop)) +  
  geom_point()
```

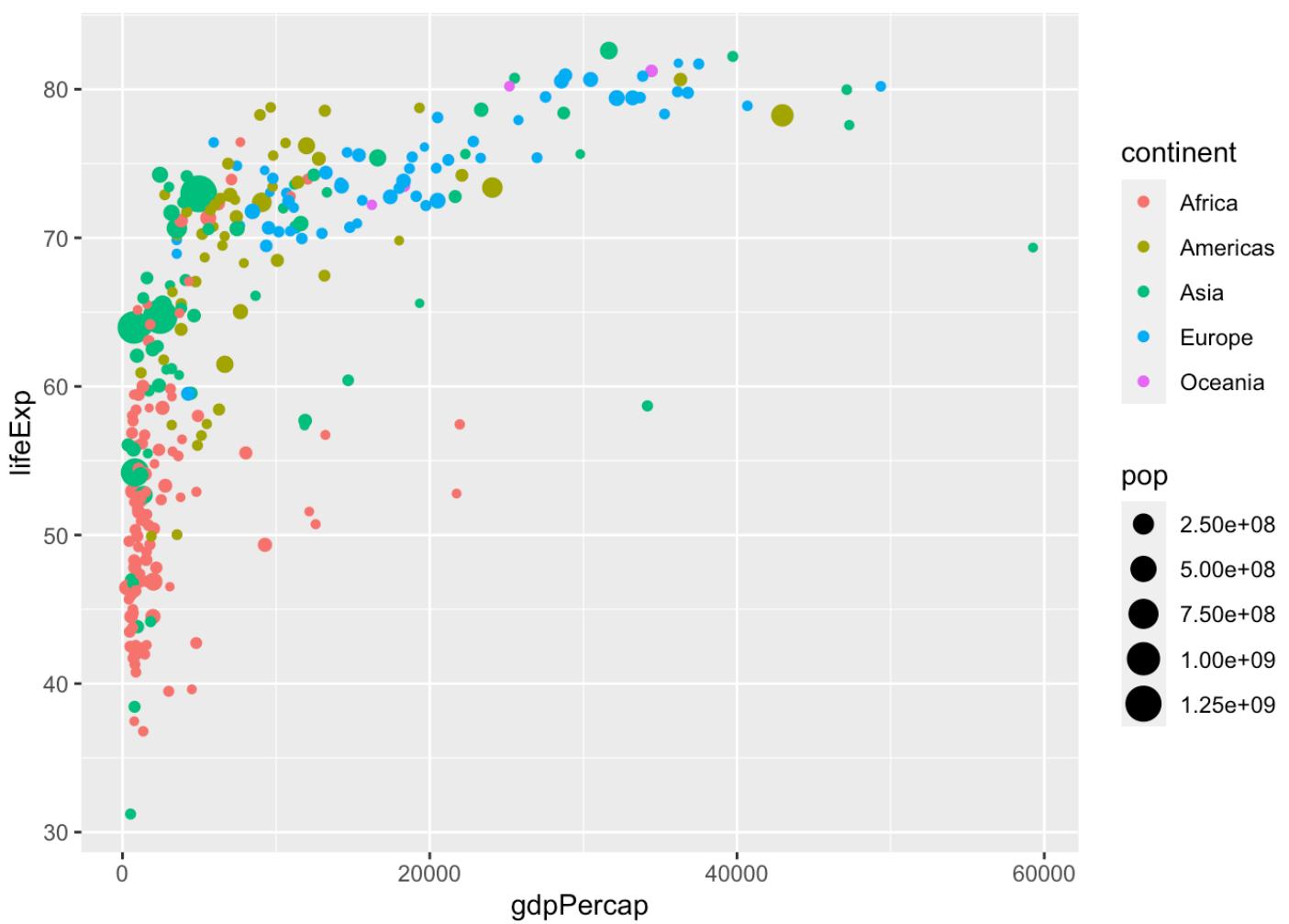


4.3.3.2 Global versus individual mapping

Mapping of aesthetics in `aes()` can be done globally or at individual layers.

In the previous plot we defined the variables `gdpPercap` and `lifeExp` in the `aes()` function inside `ggplot()` for a **global definition**. **Global mapping is inherited by default** to all geom layers (`geom_point` in the previous example), while mapping at **individual layers** is only **recognized at that layer**. For example, we define the `aes(x=gdpPercap, y=lifeExp)` globally, but the color attributes only locally for the layer `geom_point` :

```
ggplot(data = gm_dt, aes(x=gdpPercap, y=lifeExp)) +
  geom_point(aes(color=continent, size=pop))
```

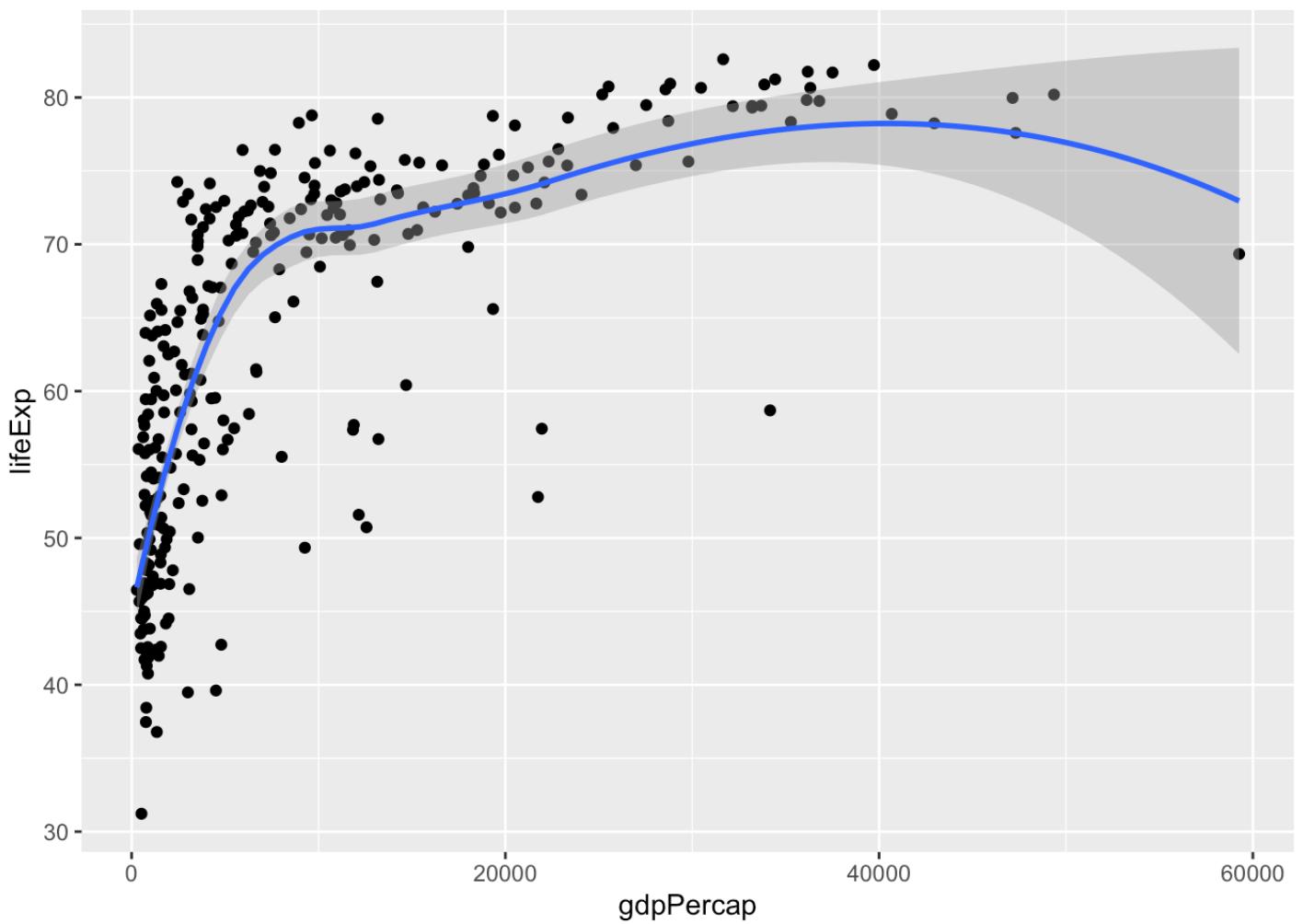


Note that individual layer mapping **cannot be recognized by other layers**. For instance, we can add another layer for smoothing with `stat_smooth()`.

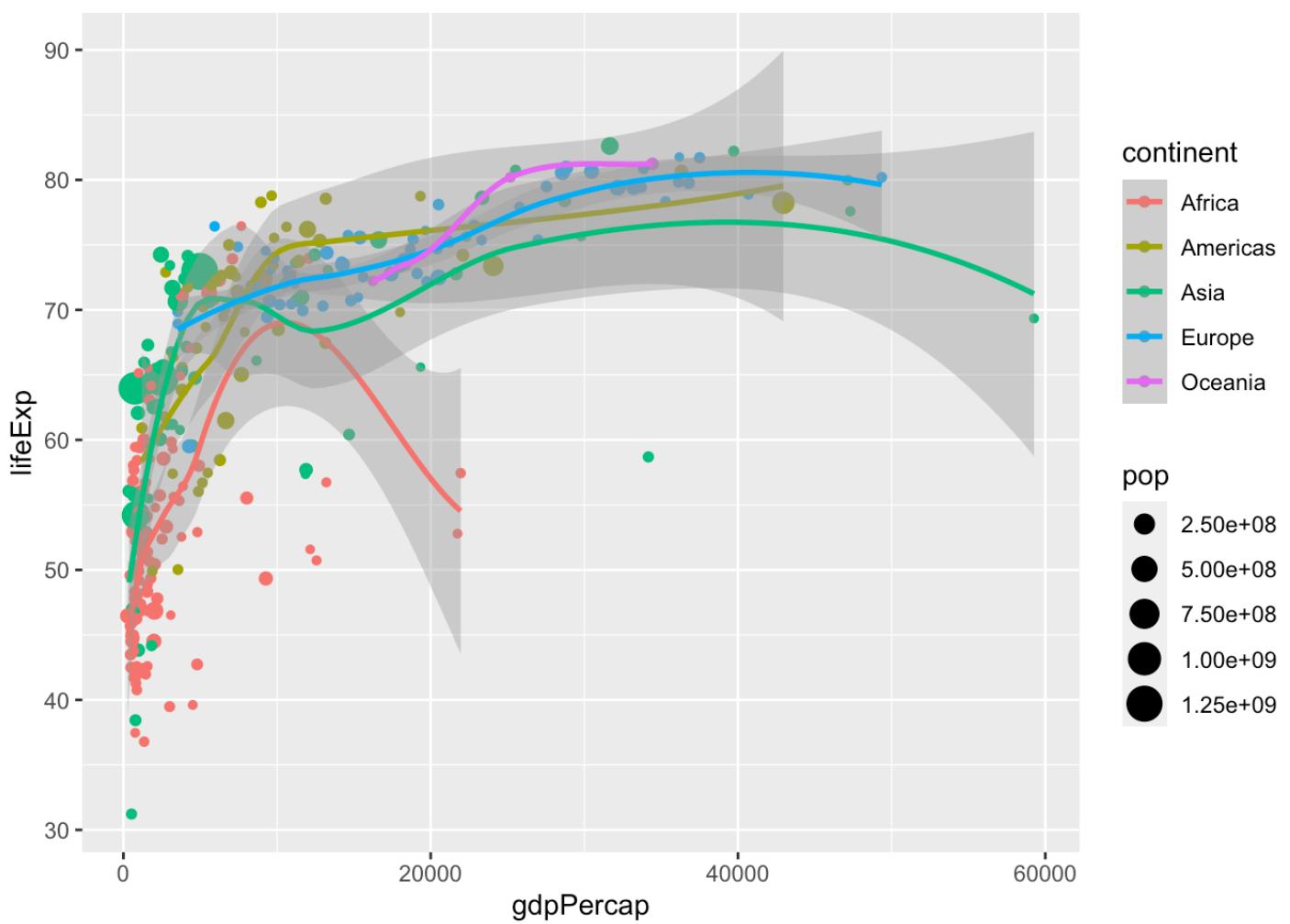
```
# this doesn't work as stat_smooth didn't know aes(x , y)
ggplot(data = gm_dt) +
  geom_point(aes(x=gdpPercap, y=lifeExp)) +
  stat_smooth()

## Error: stat_smooth requires the following missing aesthetics: x and y

# this works but is redundant
ggplot(data = gm_dt) +
  geom_point(aes(x=gdpPercap, y=lifeExp)) +
  stat_smooth(aes(x=gdpPercap, y=lifeExp))
```



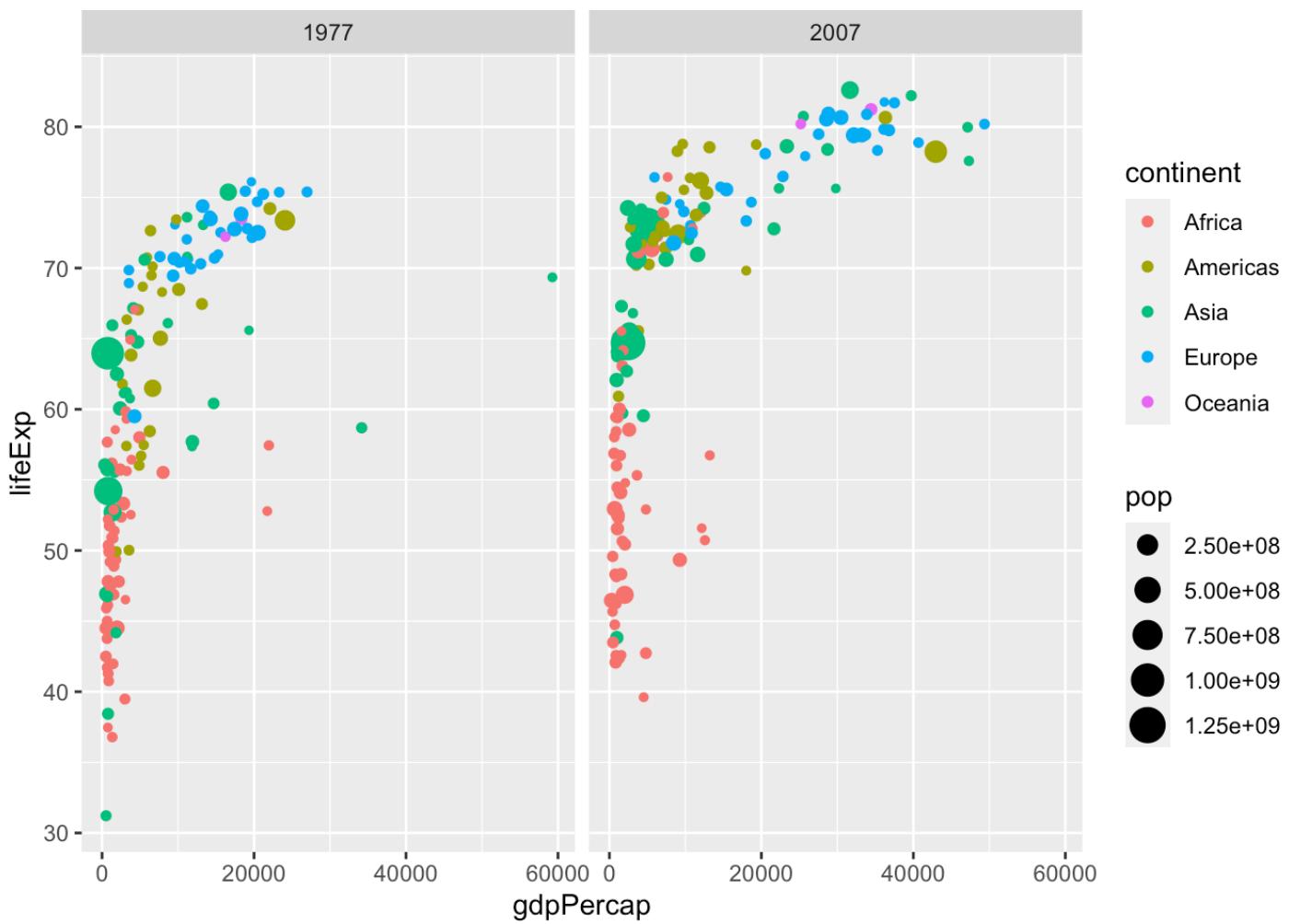
```
# the common aes(x, y) shared by all the layers can be put in the ggplot()
ggplot(data = gm_dt, aes(x=gdpPercap, y=lifeExp, color=continent)) +
  geom_point(aes(size=pop)) +
  stat_smooth()
```



4.3.4 Facets, axes and labels

For comparing the data from the year 1977 with the data from 2007, we can add a facet with `facet_wrap()` :

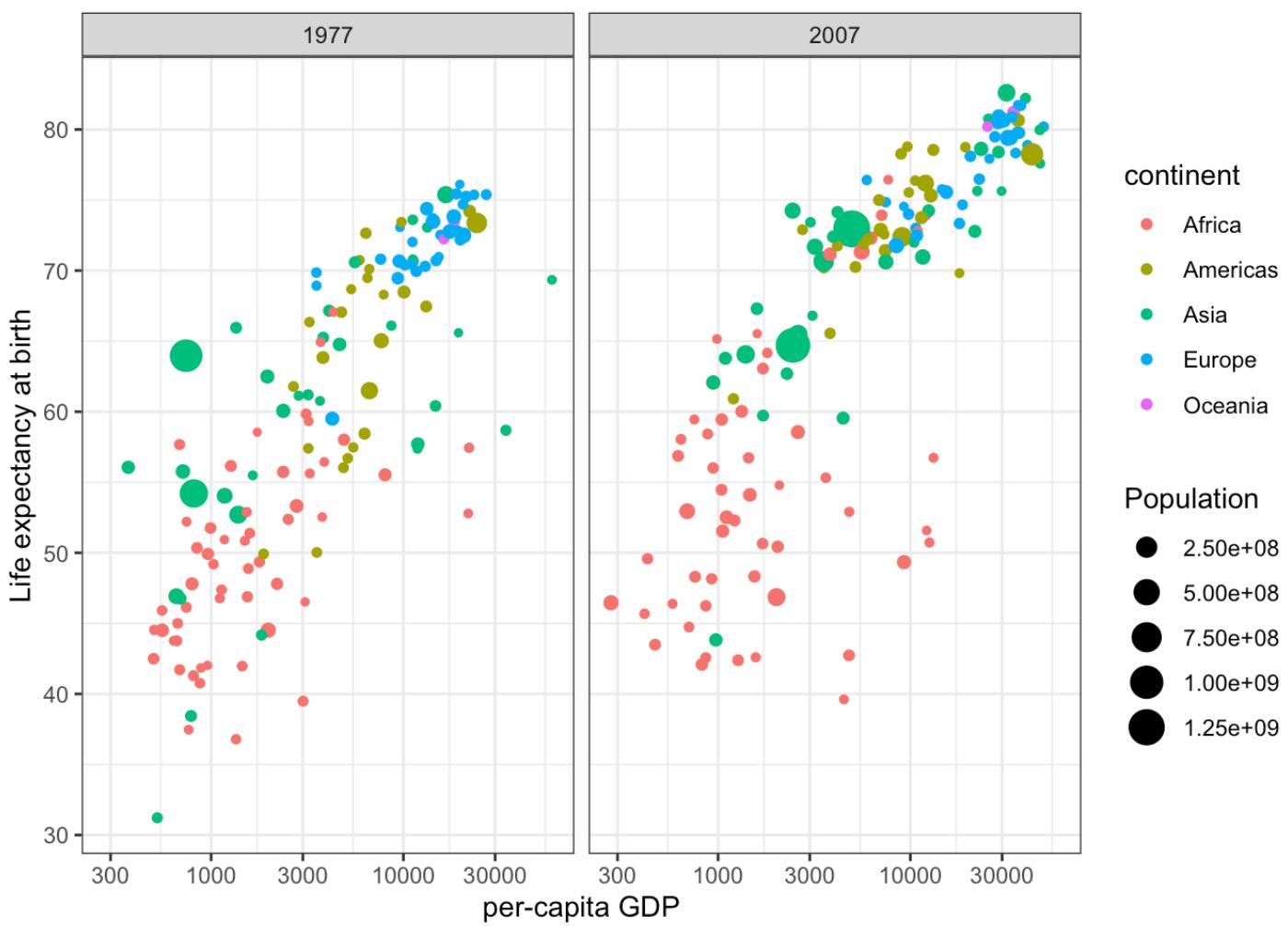
```
ggplot(data = gm_dt, aes(x=gdpPerCap, y=lifeExp, color=continent, size=pop)) +
  geom_point() + facet_wrap(~year)
```



For a better visualization of the data points, we can consider log scaling, which we will describe more in detail later. Finally, we can adapt the axes labels of the plot with `labs()` and define a theme of our plot:

```
mysize <- 15
mytheme <- theme(
  axis.title = element_text(size=mysize),
  axis.text = element_text(size=mysize),
  legend.title = element_text(size=mysize),
  legend.text = element_text(size=mysize)
) + theme_bw()

ggplot(data=gm_dt, aes(x=gdpPercap, y=lifeExp)) + geom_point(aes(color=continent, size=pop)) +
  facet_grid(~year) + scale_x_log10() +
  labs(x="per-capita GDP", y="Life expectancy at birth", size = 'Population') + mytheme
```



We remark here that `ggplot2` allows many further adaptions to plots, such as specifying axis breaks and limits. Some of these are covered in the appendix at the end of this script.

4.4 Different types of one- and two-dimensional plots

In the previous examples, we had a look at scatter plots which are suitable for plotting the relationship between two continuous variables. However, there are many more types of plots (e.g. histograms, boxplots) which can be used for plotting in different scenarios. Mainly, we distinguish between plotting one or two variables and whether the variables are continuous or discrete.

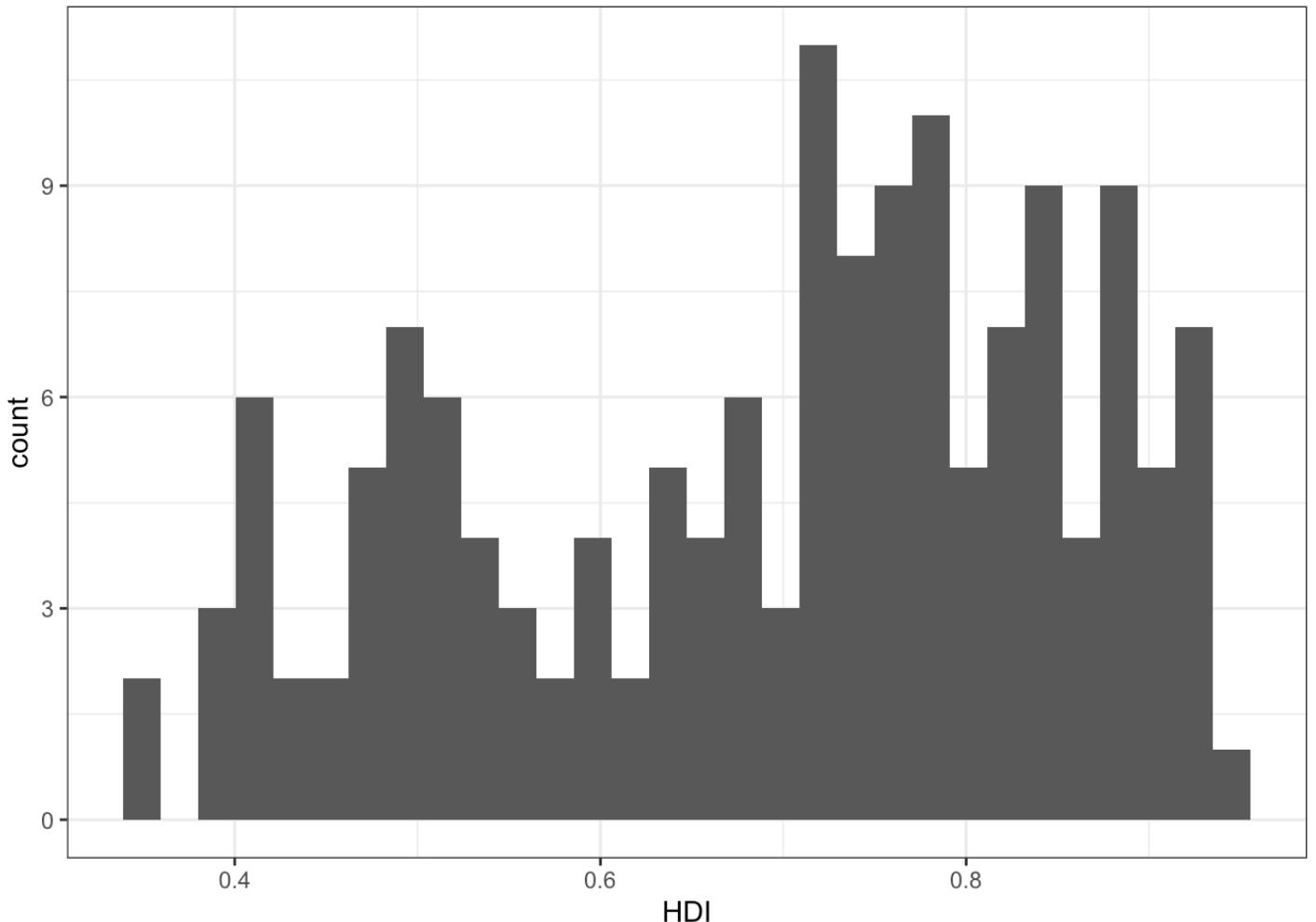
4.4.1 Plots for one single continuous variable

4.4.1.1 Histograms

A histogram represents the **frequencies of values of a variable bucketed into ranges**. It takes as input numeric variables only. A histogram is similar to a bar plot but the difference is that it groups the values into continuous ranges. Each bar in a histogram represents the height of the number of values present in that range. Each bar of the histogram is called a bin.

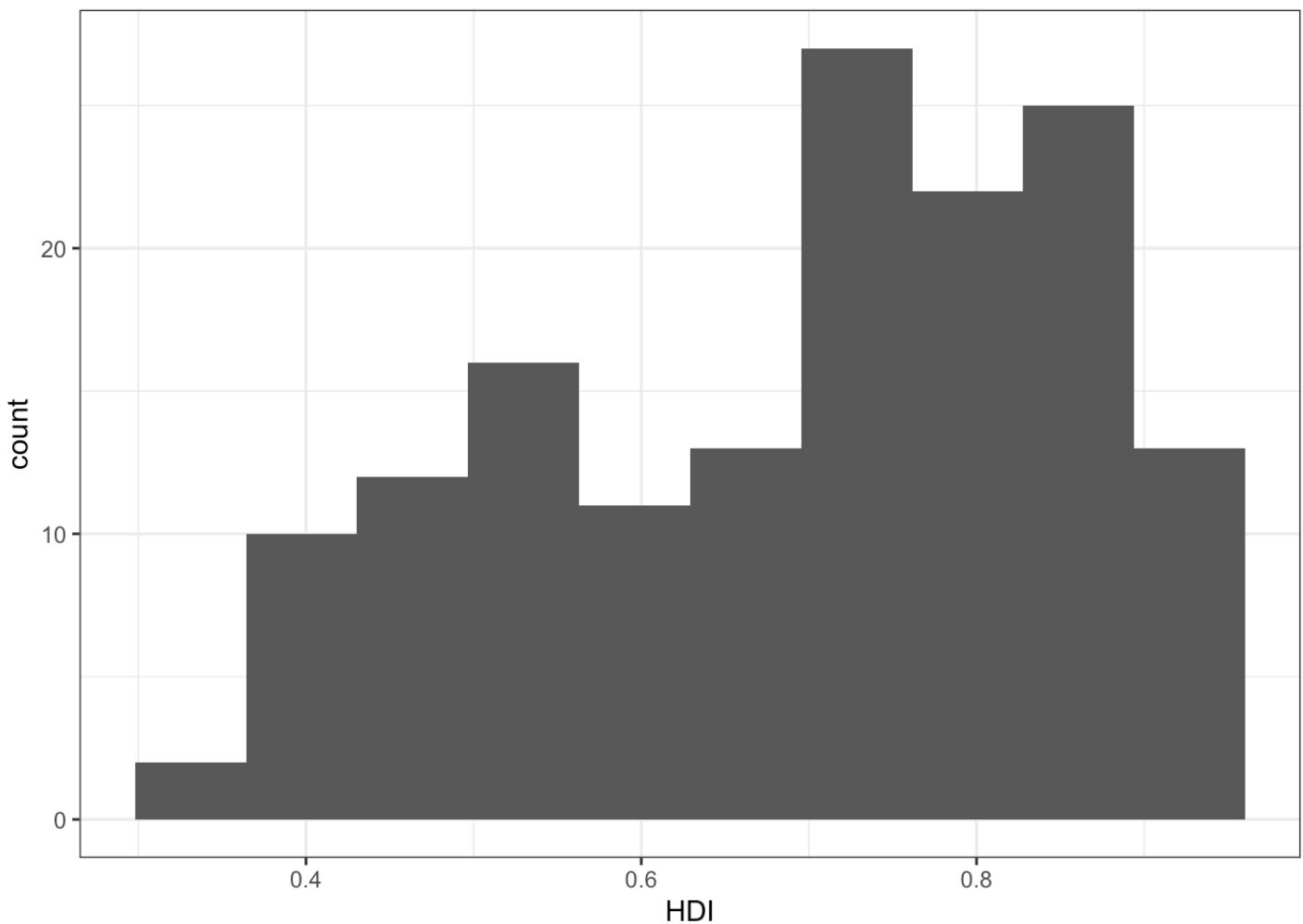
We can construct a histogram of the Human Development Index (HDI) in the `ind` dataset with the function `geom_histogram()`:

```
ggplot(ind, aes(HDI)) + geom_histogram() + mytheme
```



By default, the number of bins in `ggplot2` is 30. We can simply change this by defining the number of desired bins in the `bins` argument of the `geom_histogram()` function:

```
ggplot(ind, aes(HDI)) + geom_histogram(bins=10) + mytheme
```

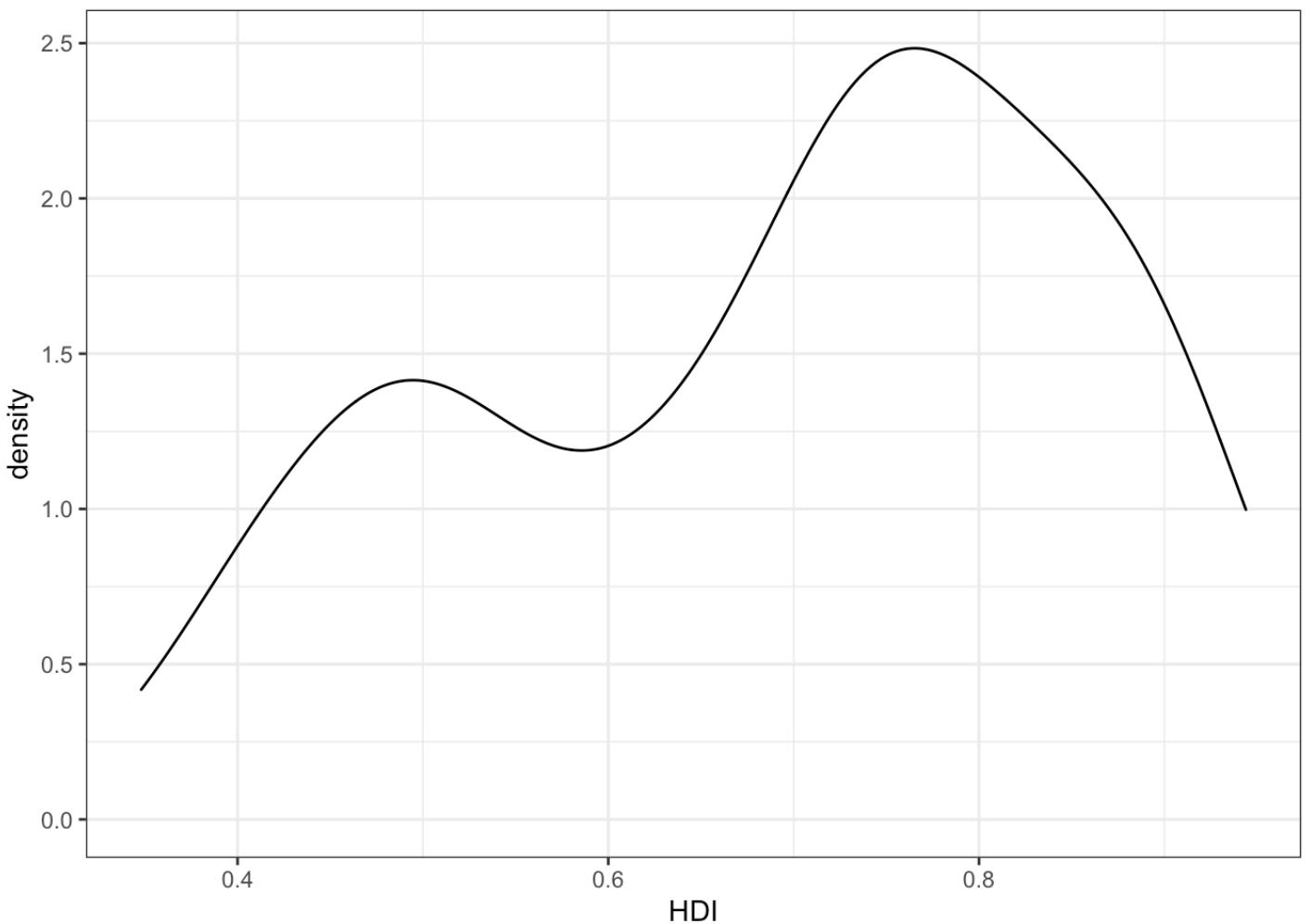


4.4.1.2 Density plots

Histograms are sometimes not optimal to investigate the distribution of a variable due to discretization effects during the binning process. A variation of histograms is given by density plots. They are used to represent the **distribution of a numeric variable**. These distribution plots are typically obtained by kernel density estimation to smoothen out the noise. Thus, the plots are smooth across bins and are not affected by the number of bins, which helps create a more defined distribution shape.

As an example, we can visualize the distribution of the Human Development Index (HDI) in the `ind` dataset by means of a density plot with `geom_density()` :

```
ggplot(ind, aes(HDI)) + geom_density() + mytheme
```

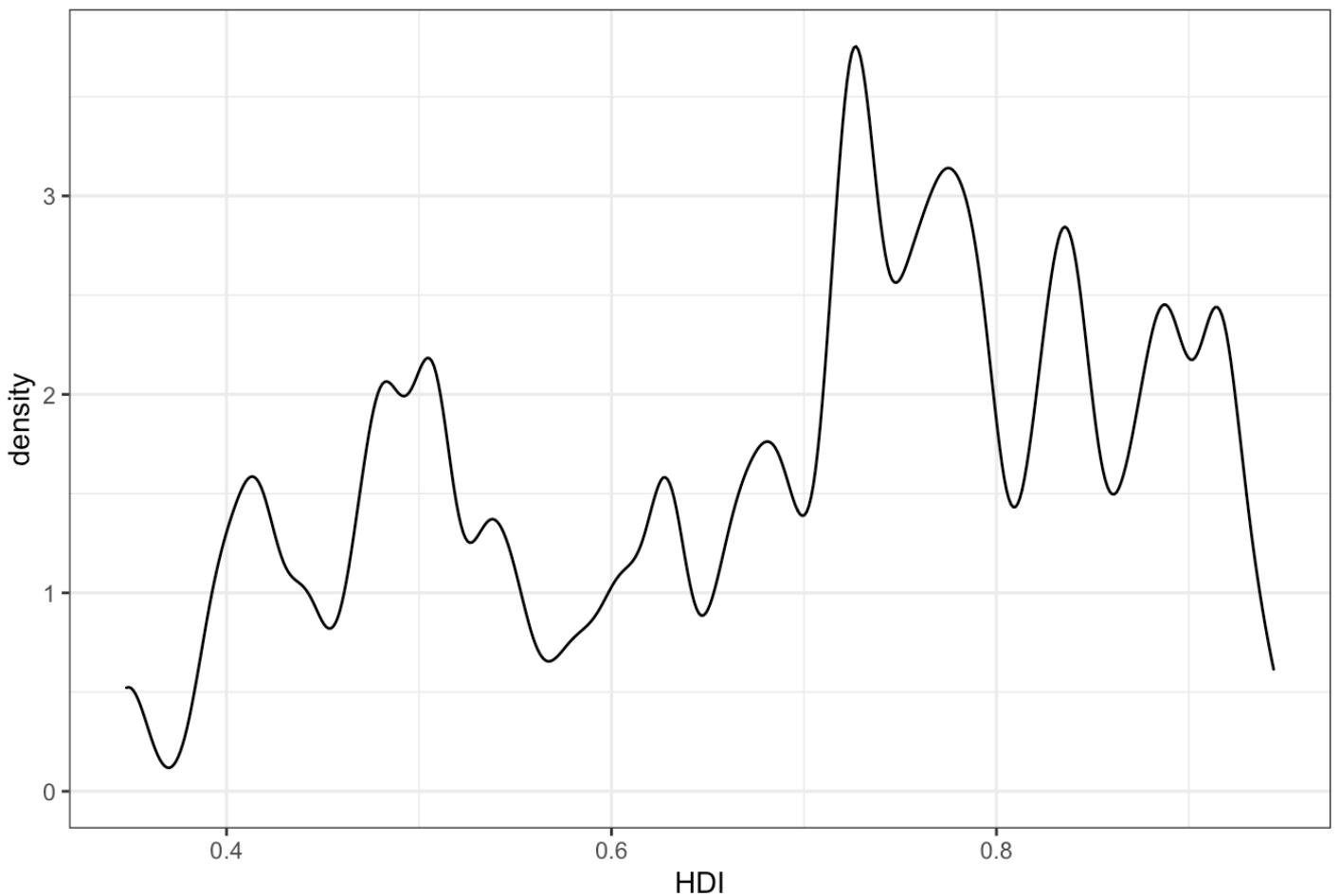


The `bw` argument of the `geom_density()` function allows to tweak the bandwidth of a density plot manually. The default option is a bandwidth rule, which is usually a good choice.

Setting a small bandwidth on the previous plot has a huge impact on the plot:

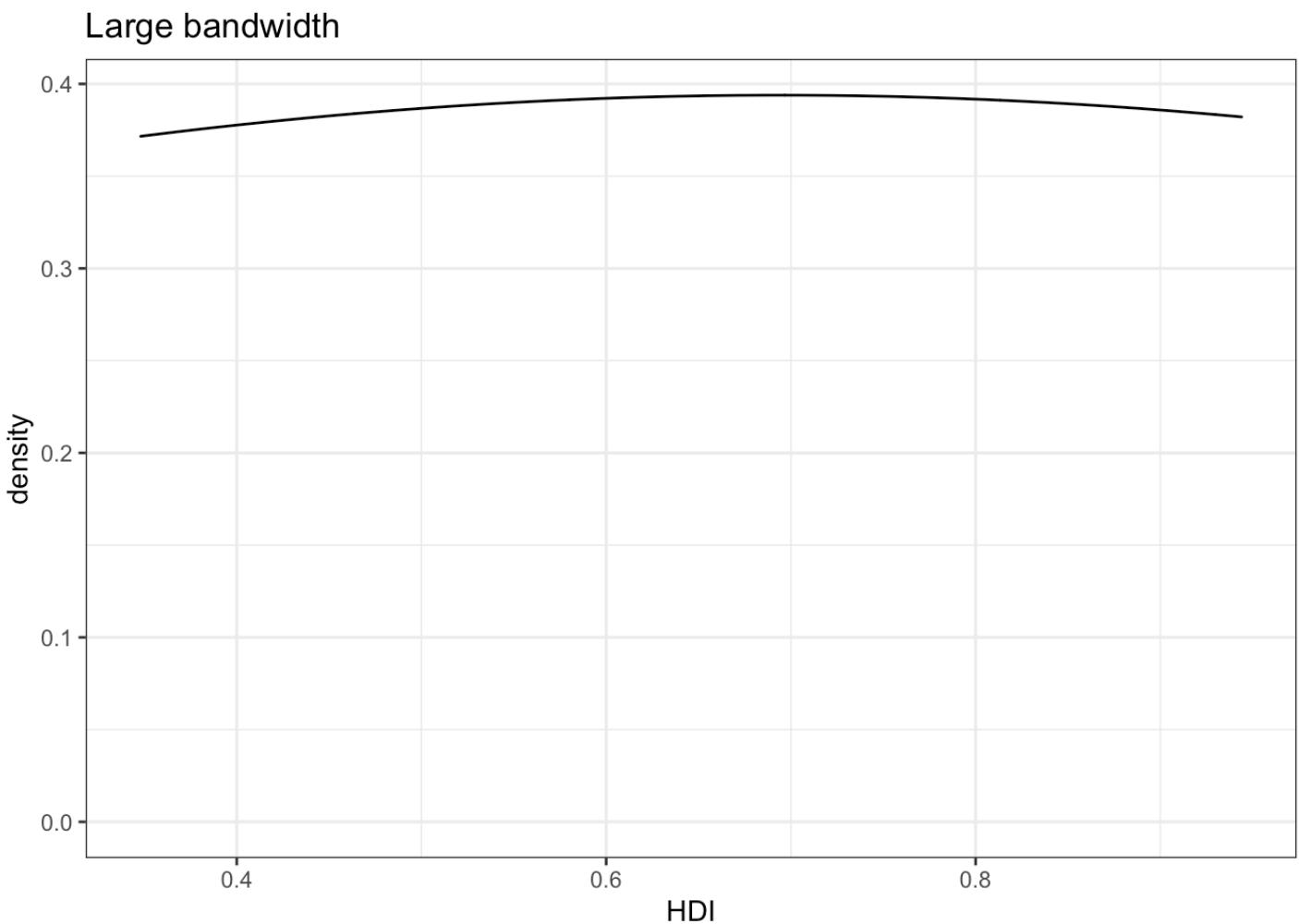
```
ggplot(ind, aes(HDI)) + geom_density(bw=0.01) + ggtitle('Small bandwidth') +  
mytheme
```

Small bandwidth



Setting a large bandwidth has also a huge impact on the plot:

```
ggplot(ind, aes(HDI)) + geom_density(bw=1) + ggtitle('Large bandwidth') +  
mytheme
```



Thus, we should be careful when changing the bandwidth, since we can get a wrong impression from the distribution of a continuous variable.

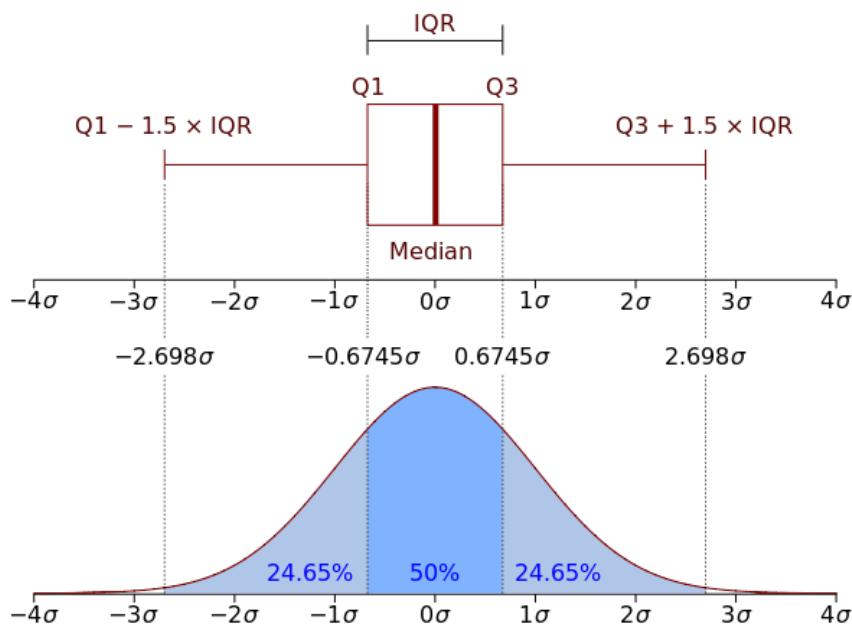
4.4.1.3 Boxplots

Boxplots can give a good graphical insight into the **distribution of the data**. They show the median, quartiles, and how far the extreme values are from most of the data.

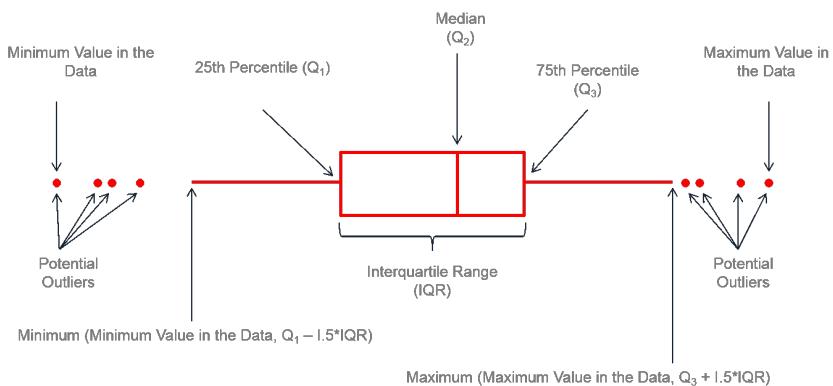
Five values are essential for constructing a boxplot:

- the **median**: the center of the data, **middle value of a sorted list, 50% quartile of the data**
- the **first quartile (Q1)**: **25% quartile of the data**
- the **third quartile (Q3)**: **75% quartile of the data**
- the **interquartile range (IQR)**: the **distance between Q1 and Q3**

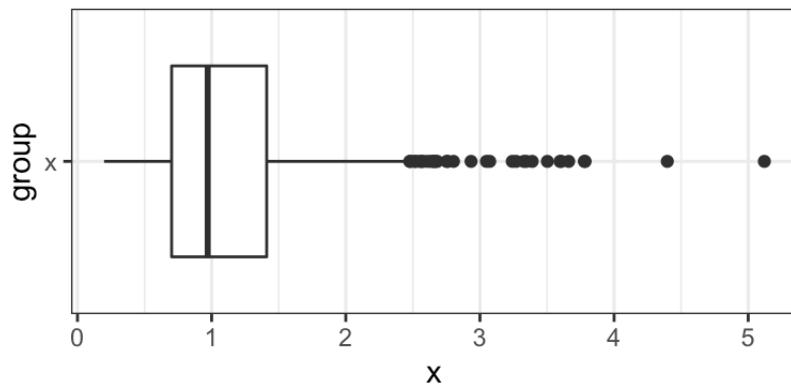
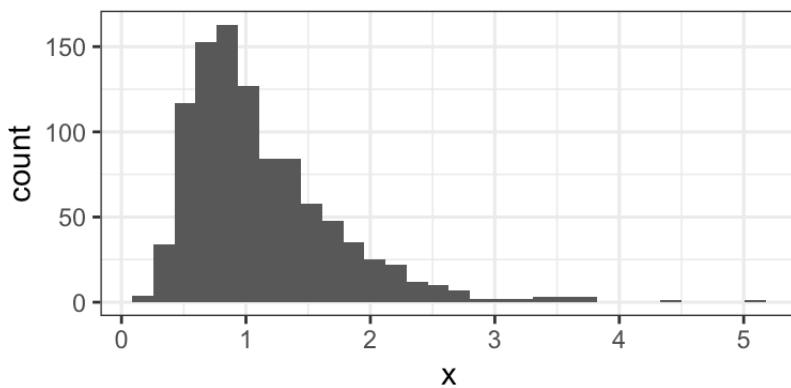
Every boxplot has lines at Q1, the median, and Q3, which together build the box of the boxplot. The other major feature of a boxplot is its whiskers. The whiskers are determined with the help of the IQR. Here, we compute $1.5 \times \text{IQR}$ below Q1 and $1.5 \times \text{IQR}$ above Q3. Anything outside of this range is called an outlier. We then draw lines at the smallest and largest point within this subset ($\text{Q1} - 1.5 \times \text{IQR}$ to $\text{Q3} + 1.5 \times \text{IQR}$) from the dataset. These lines define our whiskers which reach the most extreme data point within $\pm 1.5 \times \text{IQR}$.



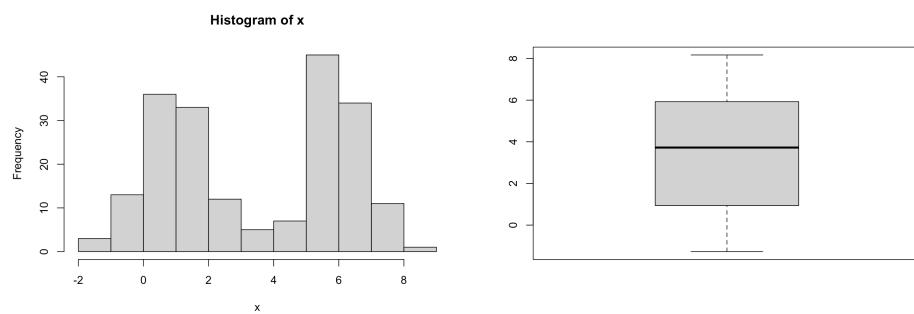
It is possible to not show the outliers in boxplots, as seen in the visualization before. However, we strongly recommend keeping them. Outliers can reveal interesting data points (discoveries “out of the box”) or bugs in data preprocessing.



For instance, we can plot the distribution of a variable x with a histogram and visualize the corresponding boxplot:



Boxplots are particularly suited for plotting non-Gaussian symmetric and non-symmetric data and for plotting exponentially distributed data. However, boxplots are not well suited for bimodal data, since they only show one mode (the median). In the following example, we see a bimodal distribution in the histogram and the corresponding boxplot, which does not properly represent the distribution of the data.



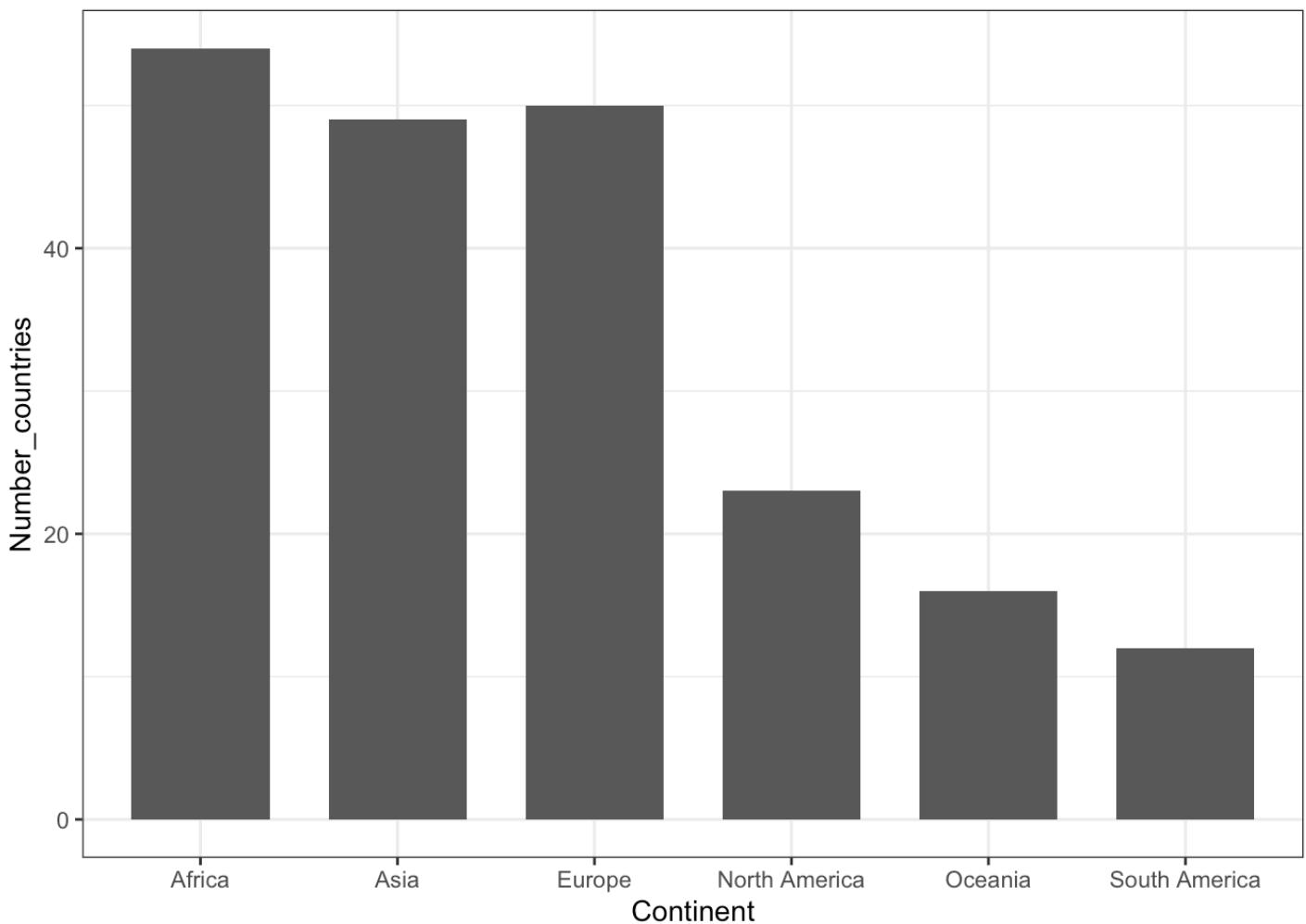
4.4.2 Plots for two variables: one continuous, one discrete

4.4.2.1 Barplots

Barplots are often used to highlight individual quantitative values per category. Bars are visual heavyweights compared to dots and lines. In a barplot, we can combine two attributes of 2-D location and line length to encode quantitative values. In this manner, we can focus the attention primarily on individual values and support the comparison of one to another.

For creating a barplot with `ggplot2` we can use the function `geom_bar()`. In the next example, we visualize the number of countries (defined in the `y` axis) per continent (defined in the `x` axis).

```
ggplot(countries_dt, aes(Continent, Number_countries)) +
  geom_bar(stat = 'identity', width = .7) + mytheme
```



4.4.2.2 Barplots with errorbars

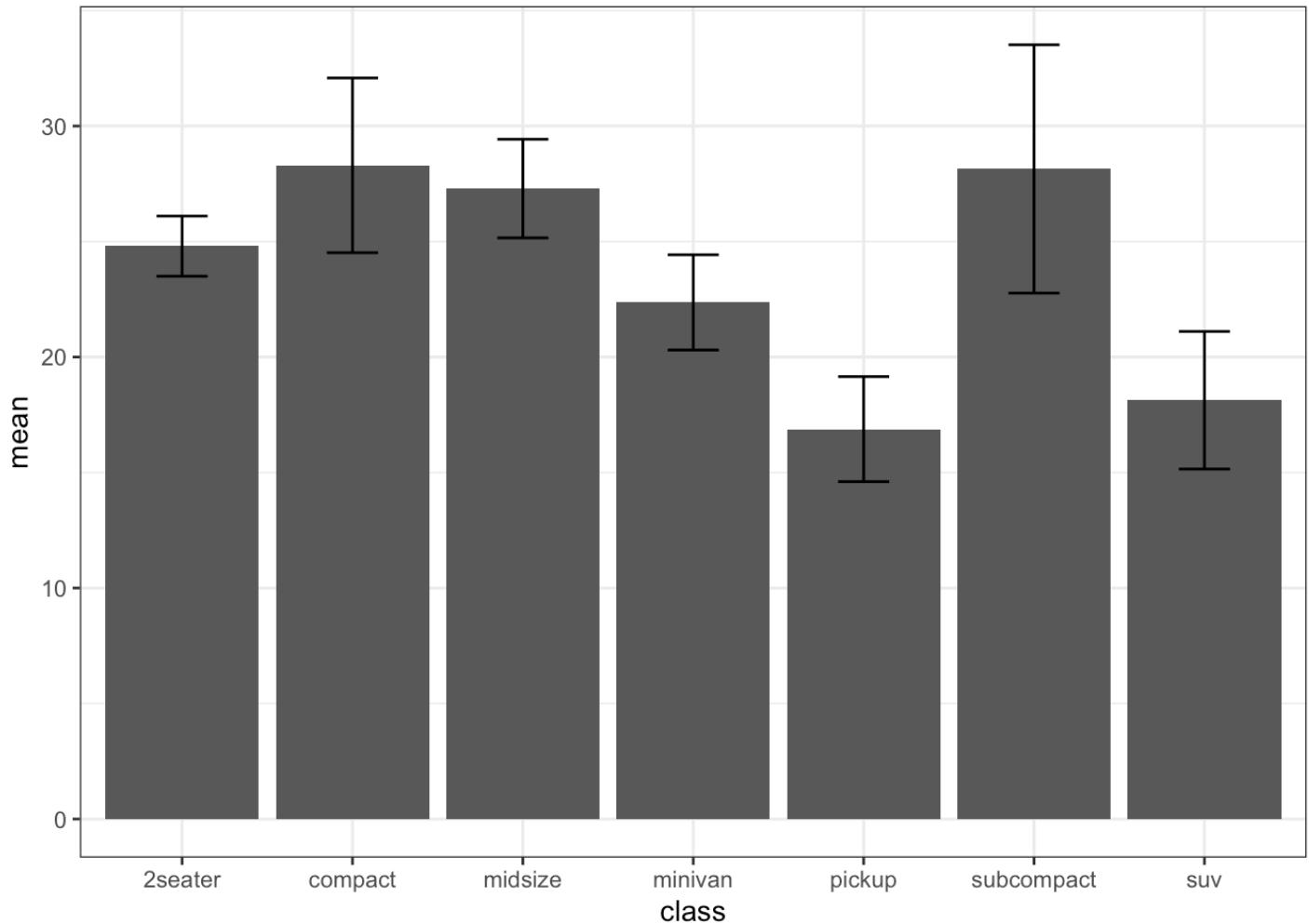
Visualizing uncertainty is important, otherwise, barplots with bars as a result of an aggregation can be misleading. One way to visualize uncertainty is with error bars.

As error bars, we can consider the standard deviation (SD) and the standard error of the mean (SEM). We remark that SD and SEM are completely different concepts. On the one hand, SD indicates the variation of quantity in the sample. On the other hand, SEM represents how well the mean is estimated.

The central limit theorem implies that: $SEM = SD/\sqrt{n}$, where n is the sample size (number of observations). With large n , SEM tends to 0.

In the following example, we plot the average highway miles per gallon `hwy` per vehicle class `class` including error bars computed as the average plus/minus standard deviation of `hwy`:

```
as.data.table(mpg) %>%
  .[, .(mean = mean(hwy),
        sd = sd(hwy)),
    by = class] %>%
  ggplot(aes(class, mean, ymax=mean+sd, ymin=mean-sd)) +
  geom_bar(stat='identity') +
  geom_errorbar(width = 0.3) + mytheme
```

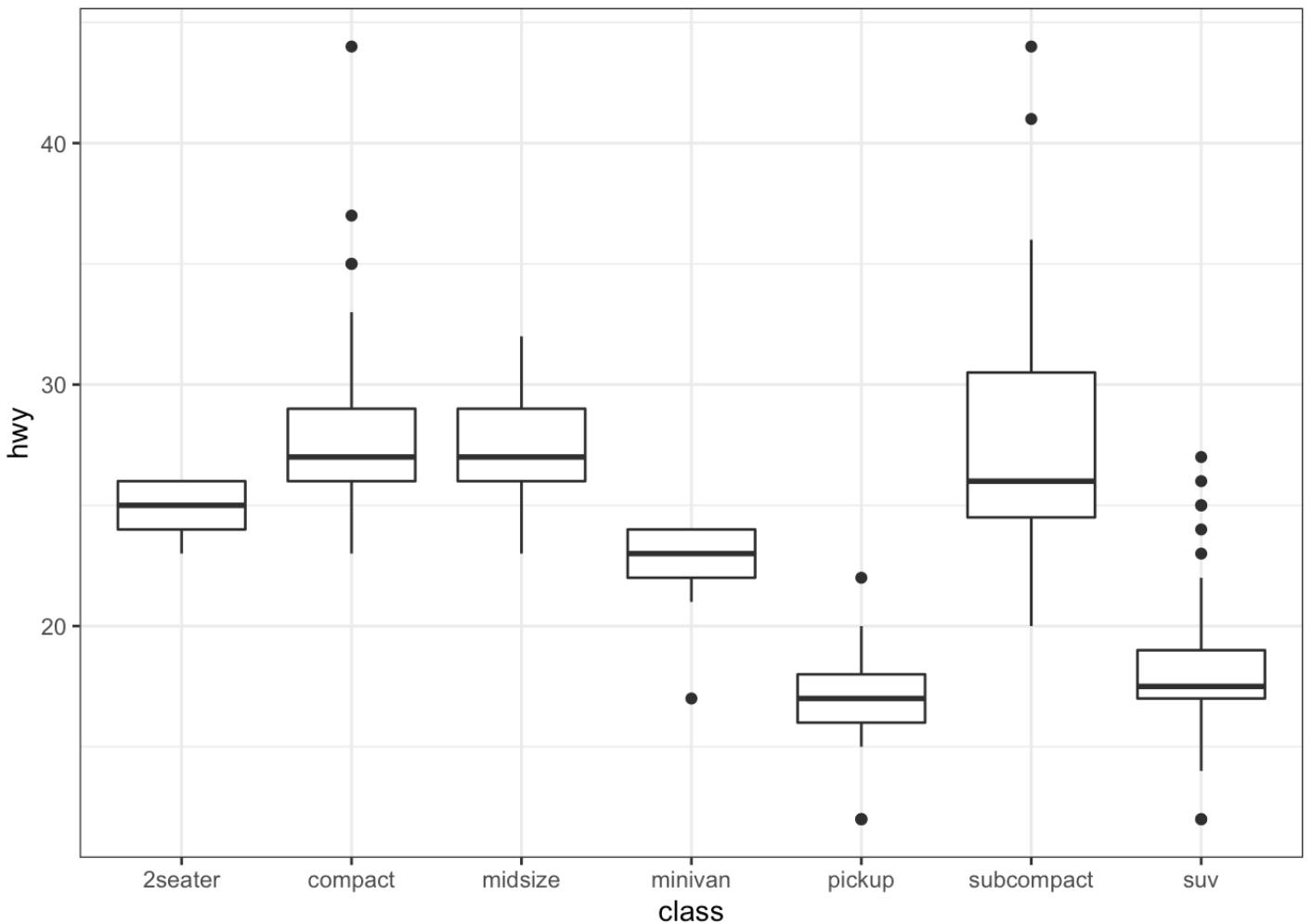


4.4.2.3 Boxplots by category

As illustrated before, boxplots are well suited for plotting one continuous variable. However, we can also use boxplots to show distributions of continuous variables with respect to some categories. This can be particularly interesting for comparing the different distributions of each category.

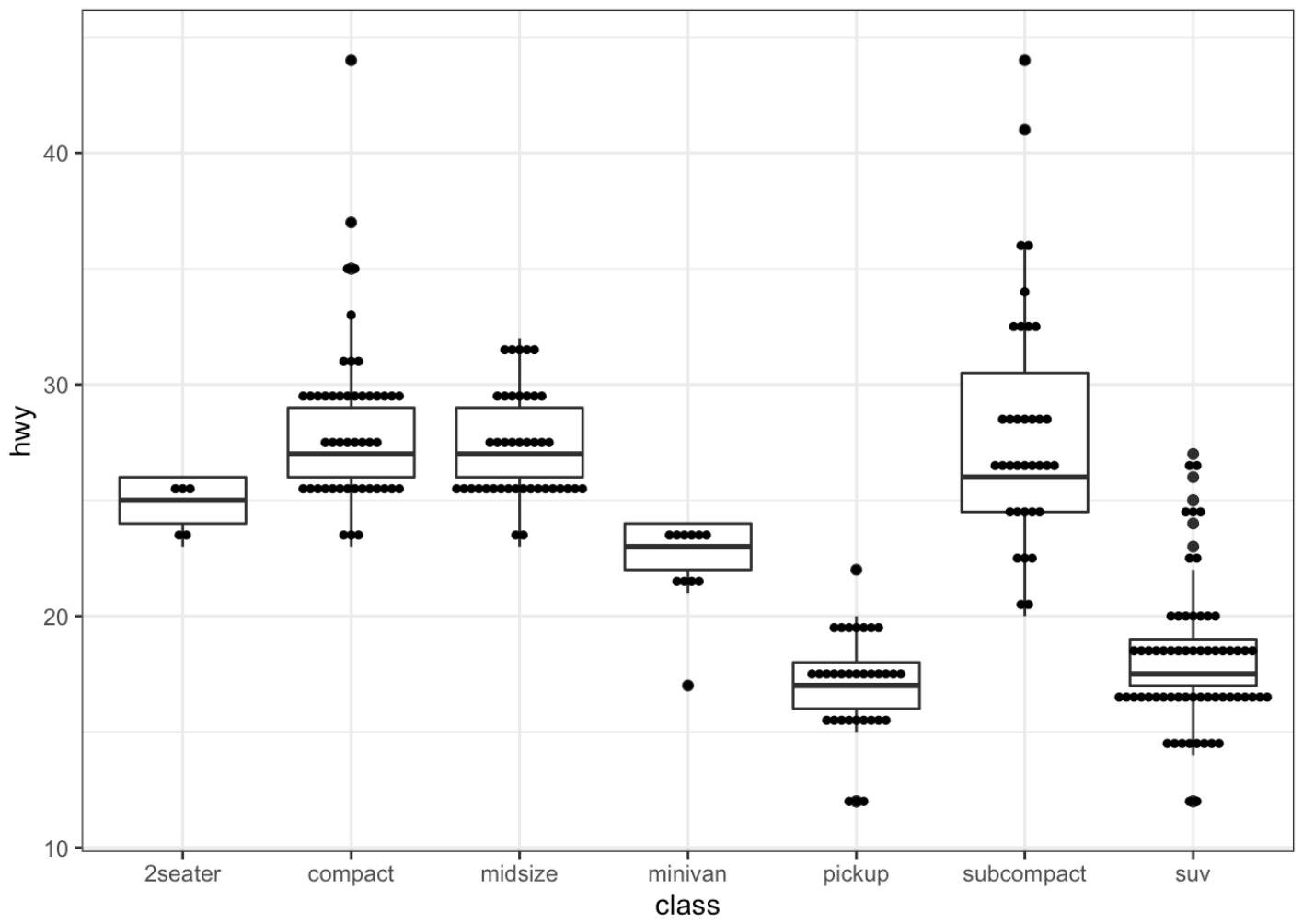
For instance, we want to visualize the highway miles per gallon `hwy` for every one of the 7 vehicle classes (compact, SUV, minivan, etc.). For this, we define the categorical `class` variable on the `x` axis and the continuous variable `hwy` on the `y` axis.

```
ggplot(mpg, aes(class, hwy)) +
  geom_boxplot() + mytheme
```

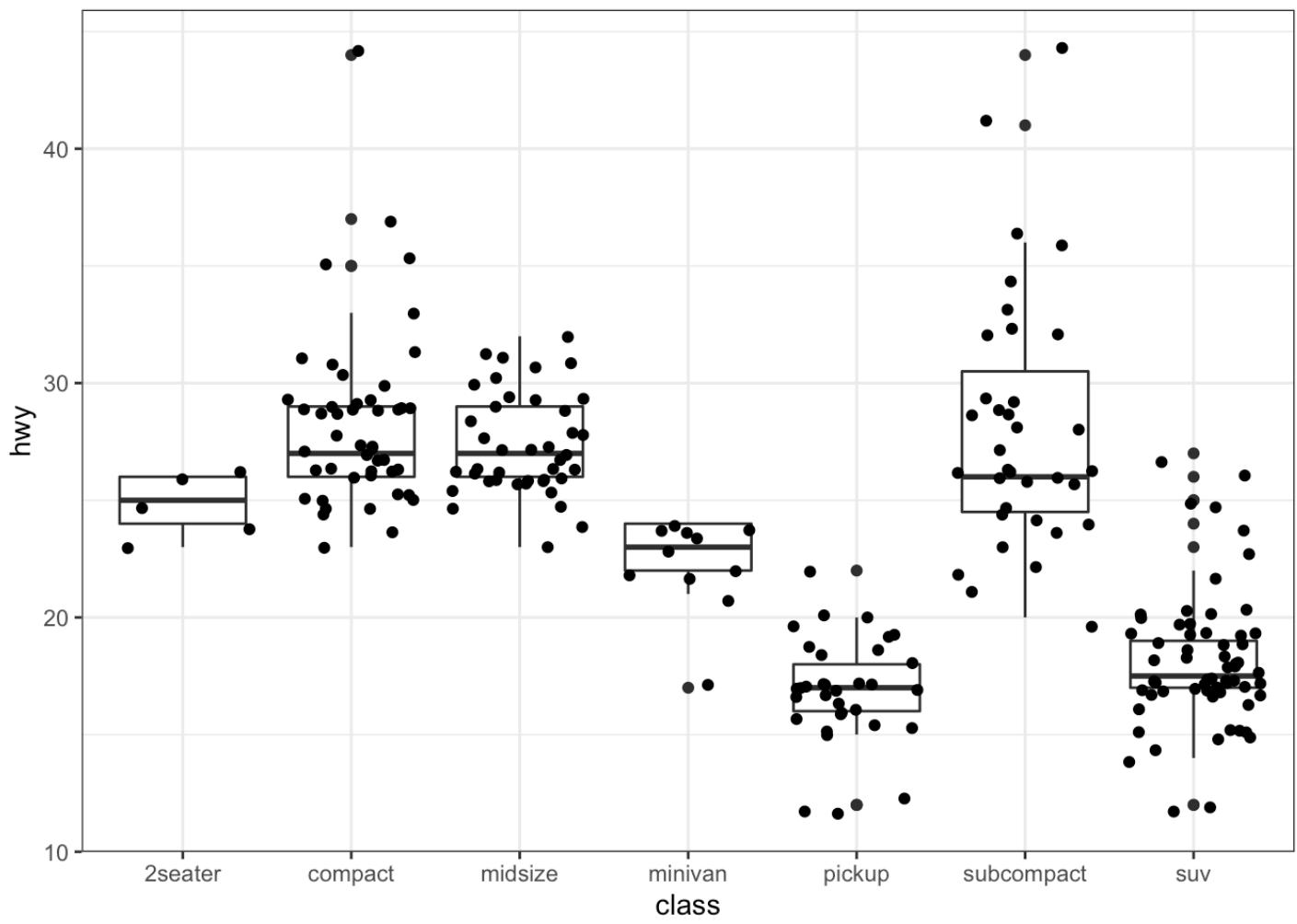


We can also add dots (or points) to a boxplot using the functions `geom_dotplot()` or `geom_jitter()` :

```
p <- ggplot(mpg, aes(class, hwy)) + geom_boxplot() + mytheme  
p + geom_dotplot(binaxis='y', stackdir='center', dotsize=0.3)
```



```
p + geom_jitter()
```

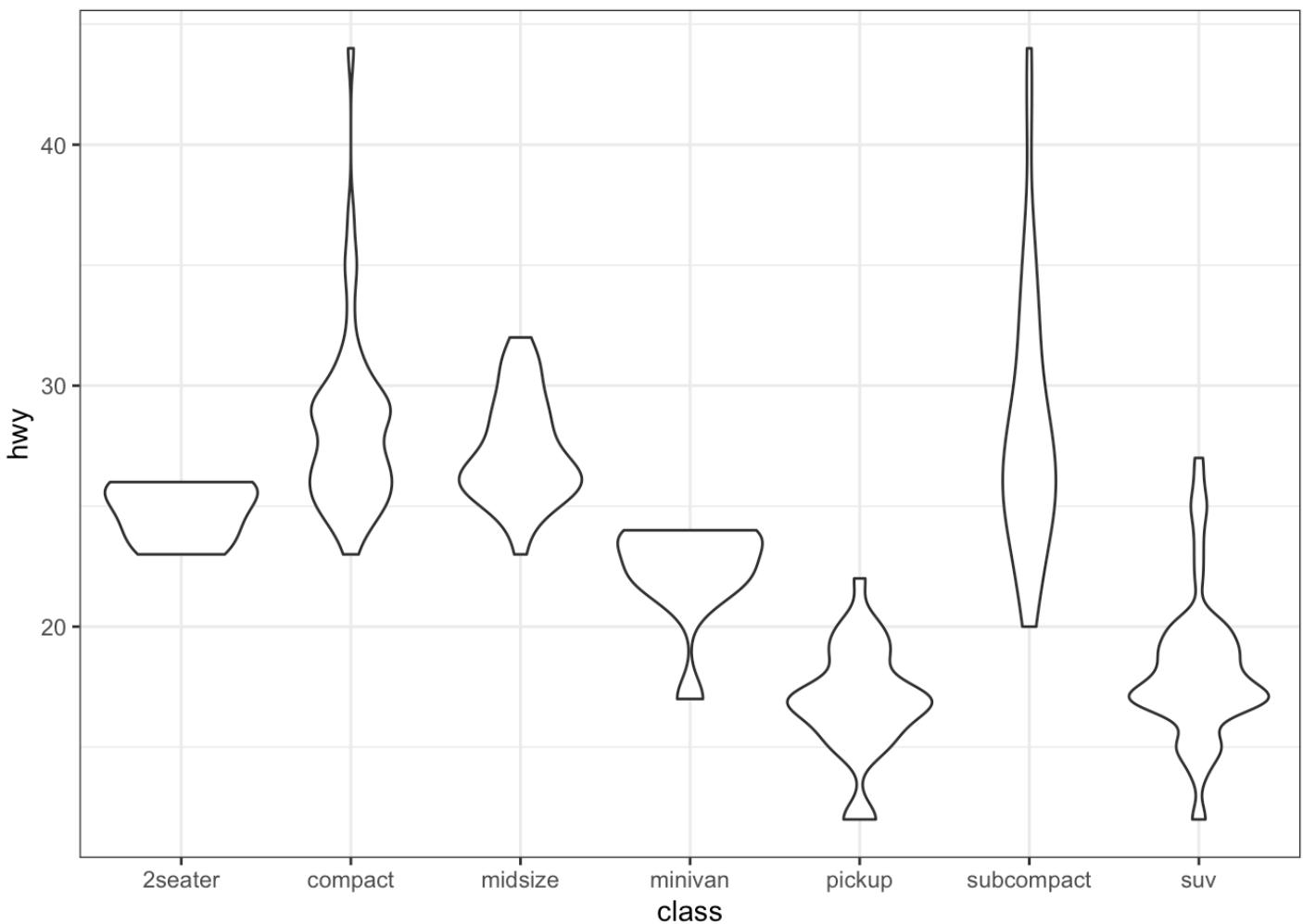


4.4.2.4 Violin plots

A violin plot is an alternative to the boxplot for visualizing one continuous variable (grouped by categories). An advantage of the violin plot over the boxplot is that it also shows the entire distribution of the data. This can be particularly interesting when dealing with multimodal data.

For a direct comparison, we show a violin plot for the `hwy` grouped by `class` as before with the help of the function `geom_violin()` :

```
ggplot(mpg, aes(class, hwy)) +
  geom_violin() + mytheme
```

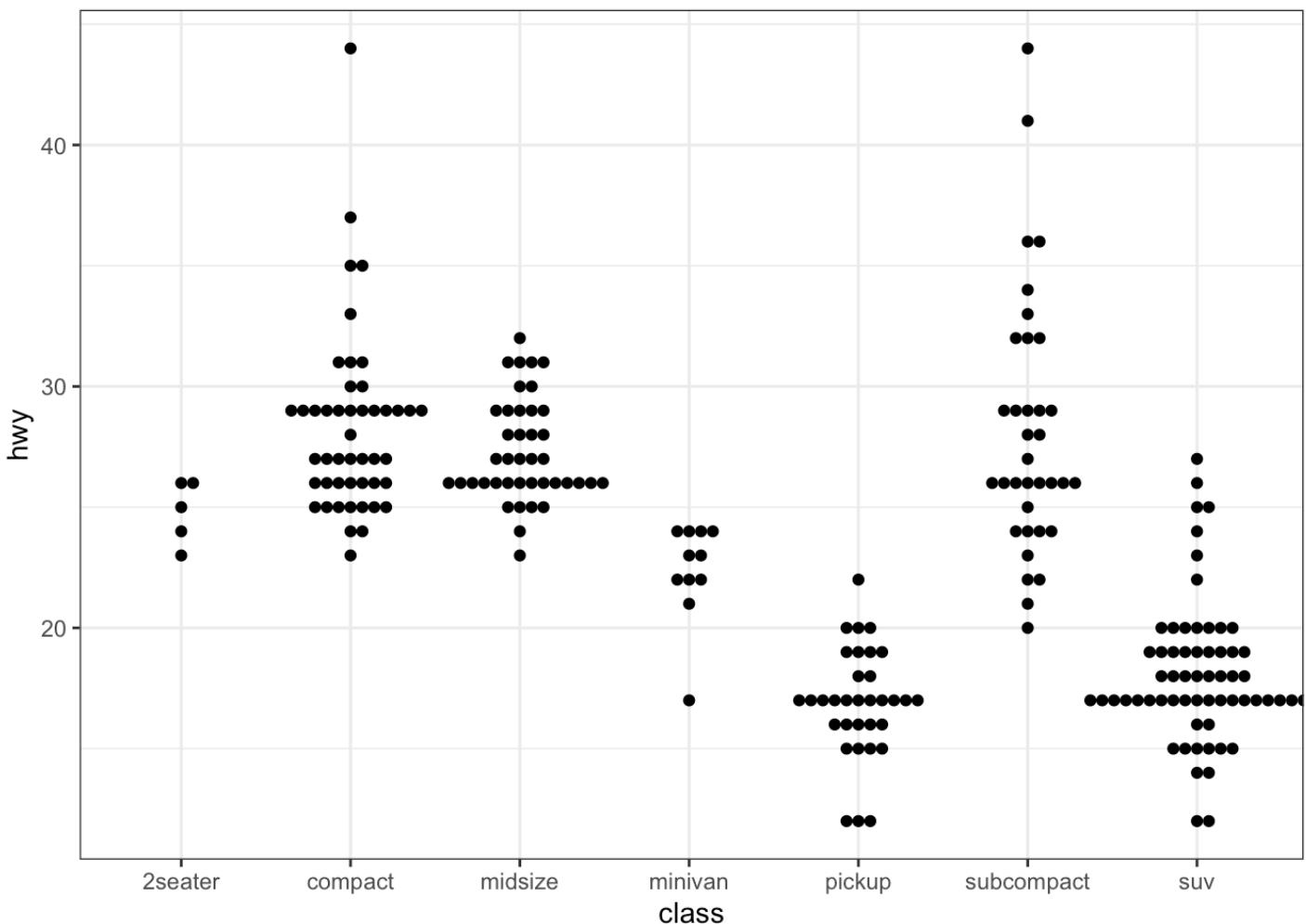


4.4.2.5 Beanplots

Another alternative to the popular boxplot is the beanplot. A beanplot has the advantage that the individual observations are shown as small lines in a one-dimensional scatter plot. Moreover, the estimated density of the distributions is visible in a beanplot. It is easy to compare different groups of data in a beanplot and to see if a group contains enough observations to make the group interesting from a statistical point of view.

For creating beanplots in R, we can use the package `ggbeeswarm`. We use the function `geom_beeswarm()` to create a beanplot to visualize once again the `hwy` grouped by `class`:

```
# install.packages("ggbeeswarm")
library(ggbeeswarm)
ggplot(mpg, aes(class, hwy)) +
  geom_beeswarm() + mytheme
```



We remark that beanplots are useful only up to a certain number of data points. The creation of beanplots for larger datasets may become too expensive.

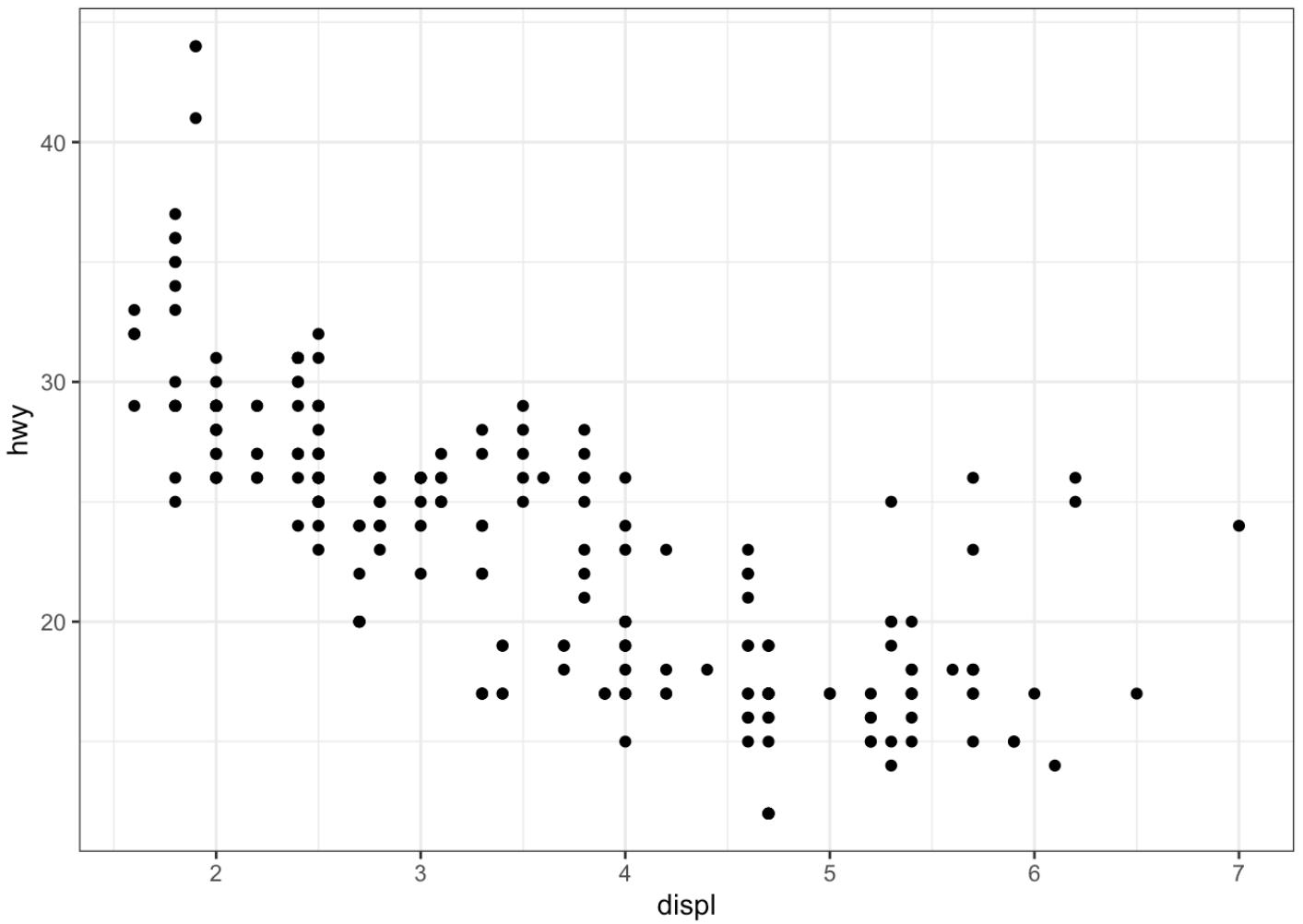
4.4.3 Plots for two continuos variables

4.4.3.1 Scatter plots

Scatter plots are a useful plot type for easily visualizing the **relationship between two continuous variables**. Here, dots are used to represent pairs of values corresponding to the two considered variables. The position of each dot on the horizontal (x) and vertical (y) axis indicates values for an individual data point.

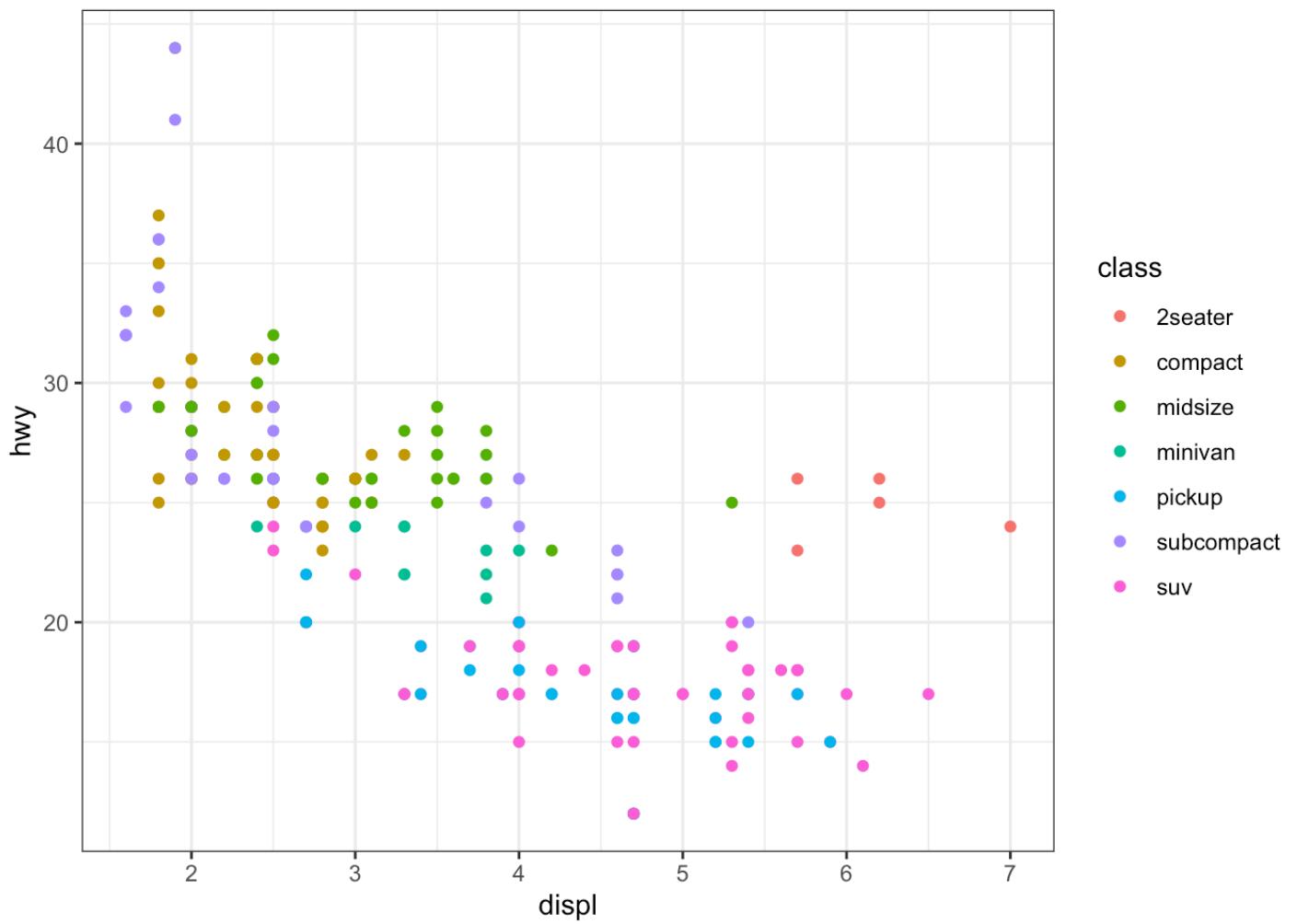
In the next example, we analyze the relationship between the engine displacement in liters `displ` and the highway miles per gallon `hwy` from the `mpg` dataset:

```
ggplot(mpg, aes(displ, hwy)) + geom_point() + mytheme
```



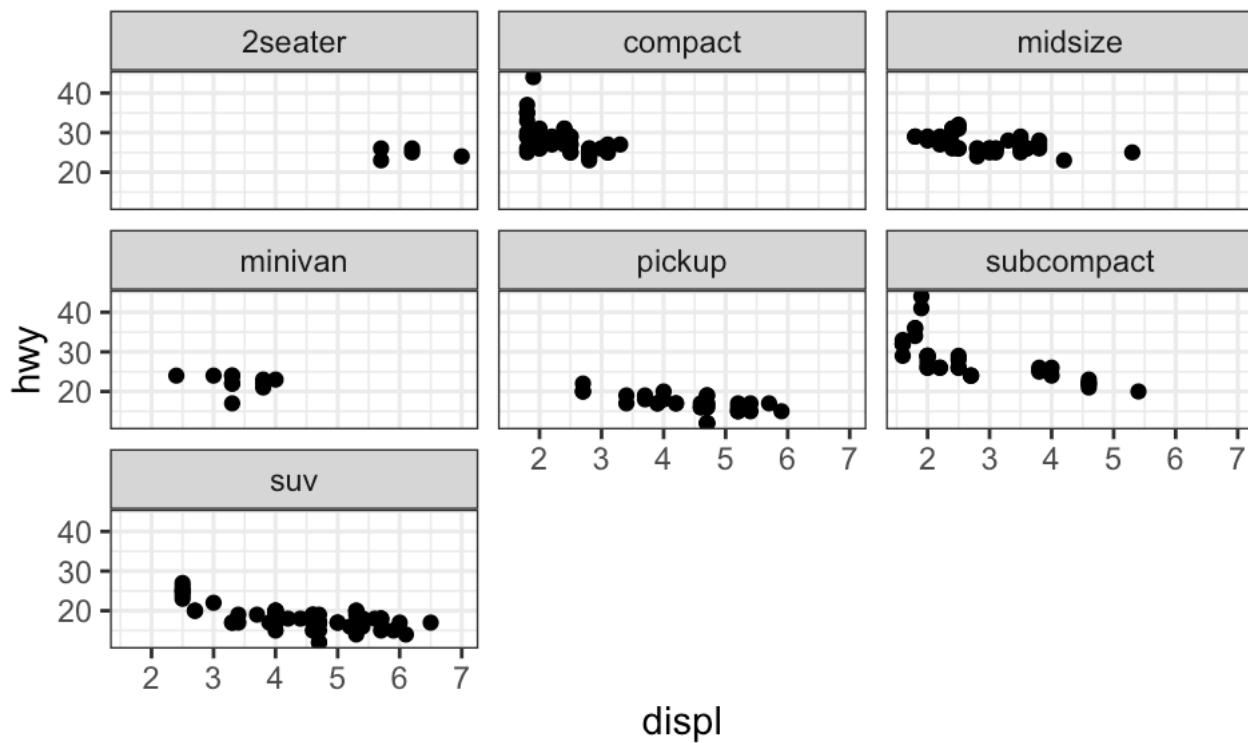
We can modify the previous plot by coloring the points depending on the vehicle class:

```
ggplot(mpg, aes(displ, hwy, color=class)) + geom_point() + mytheme
```



Sometimes, too many colors can be hard to distinguish. In such cases, we can use `facet` to separate them into different plots:

```
ggplot(mpg, aes(displ, hwy)) + geom_point() + facet_wrap(~class) + mytheme
```

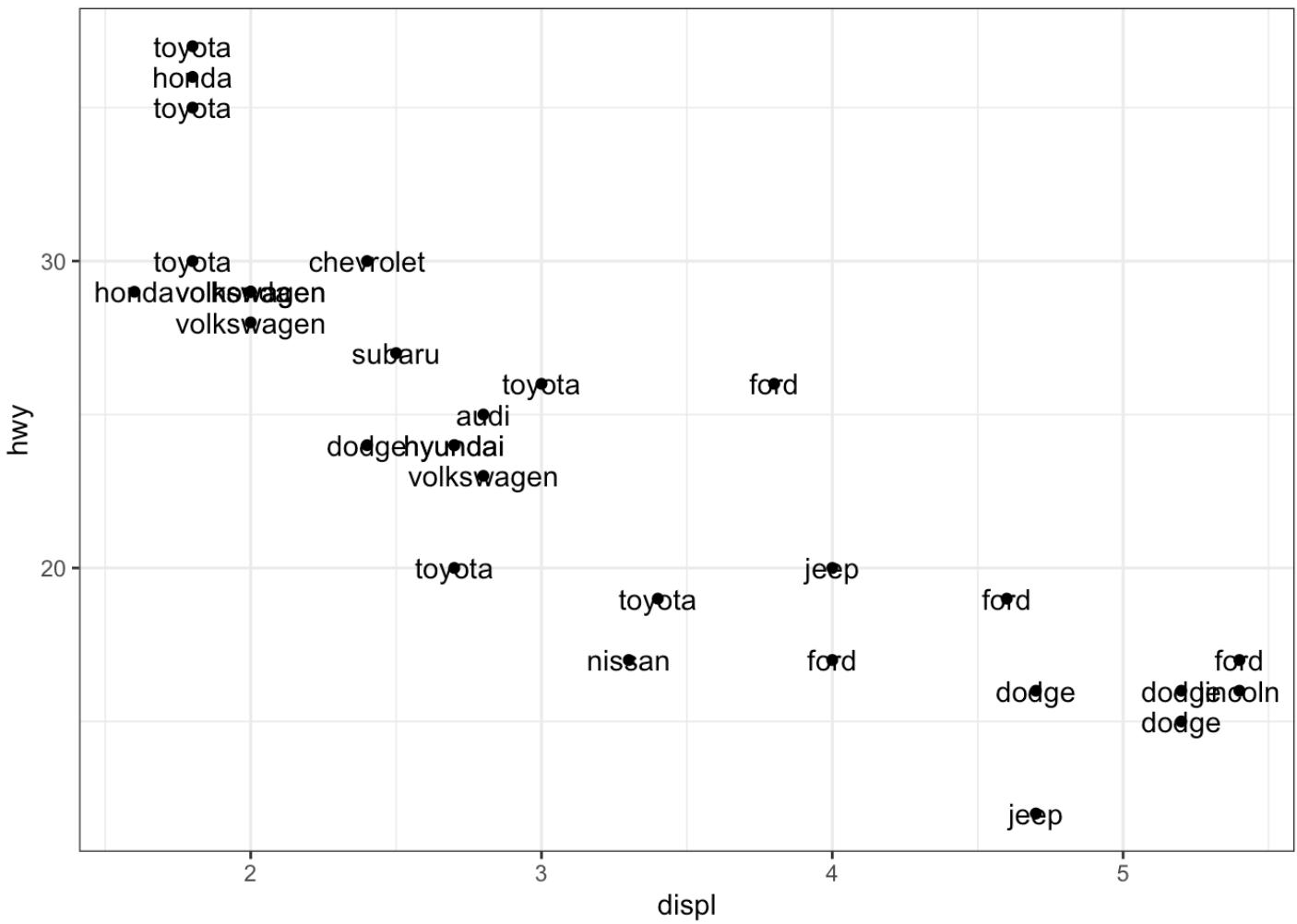


4.4.3.1.1 Text labeling

For labeling the individual points in a scatter plot, `ggplot2` offers the function `geom_text()`. However, these labels tend to overlap. To avoid this, we can use the library `ggrepel` which offers a better text labeling through the function `geom_text_repel()`.

We first show the output of the classic text labeling with `geom_text()` for a random subset of 40 observations of the dataset `mpg`. Here we plot the engine displacement in liters `displ` vs. the highway miles per gallon `hwy` and label by `manufacturer`:

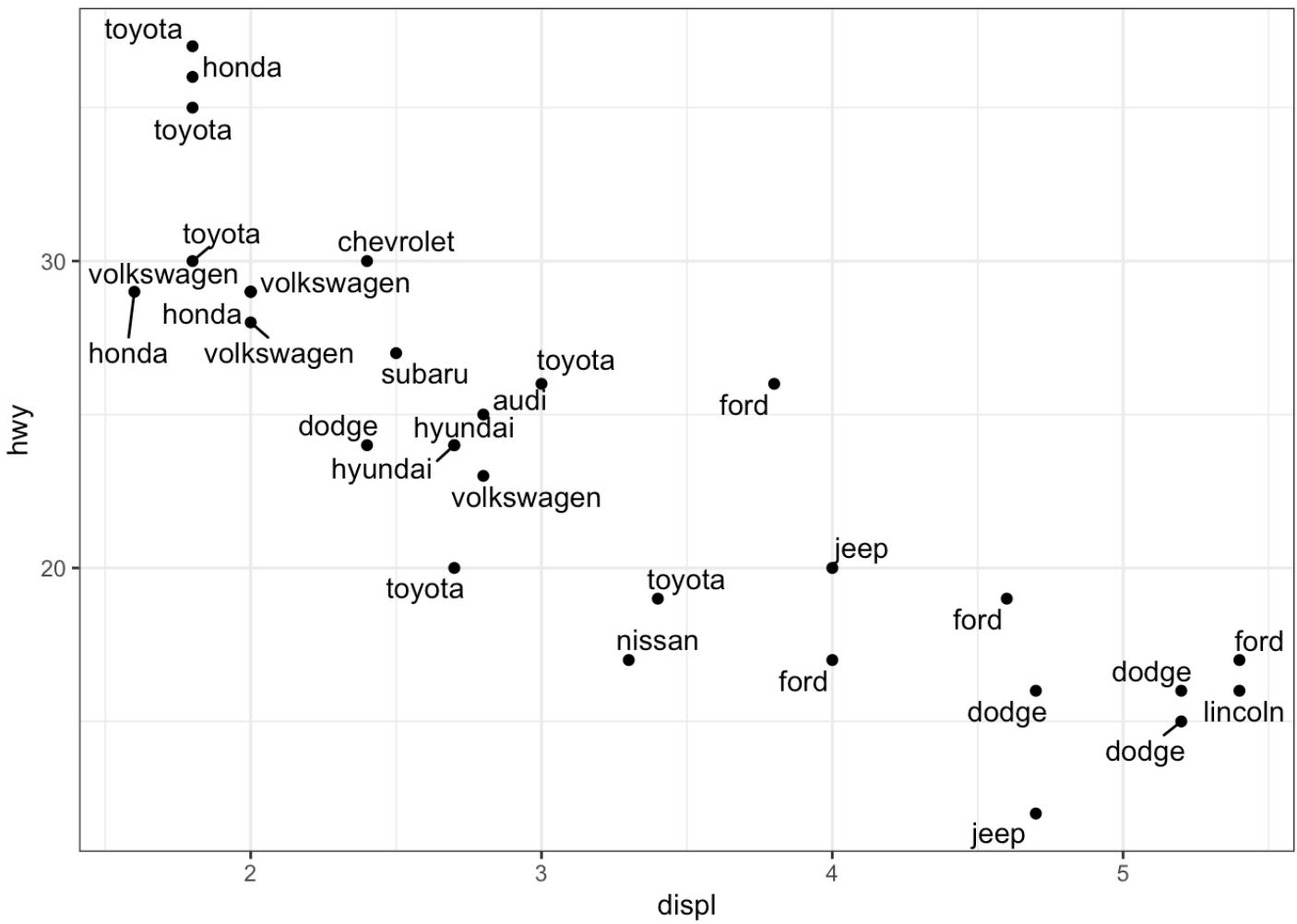
```
set.seed(12)
mpg_subset <- mpg[sample(1:nrow(mpg), 30, replace=FALSE),]
ggplot(mpg_subset, aes(displ, hwy, label=manufacturer)) + geom_point() +
  geom_text() + mytheme
```



As seen in the previous illustration, the text labels overlap. This complicates understanding the plot. Therefore, we exchange the function `geom_text()` by `geom_text_repel()` from the library `ggrepel`:

```
library(ggrepel)

ggplot(mpg_subset, aes(displ, hwy, label=manufacturer)) +
  geom_point() + geom_text_repel() + mytheme
```

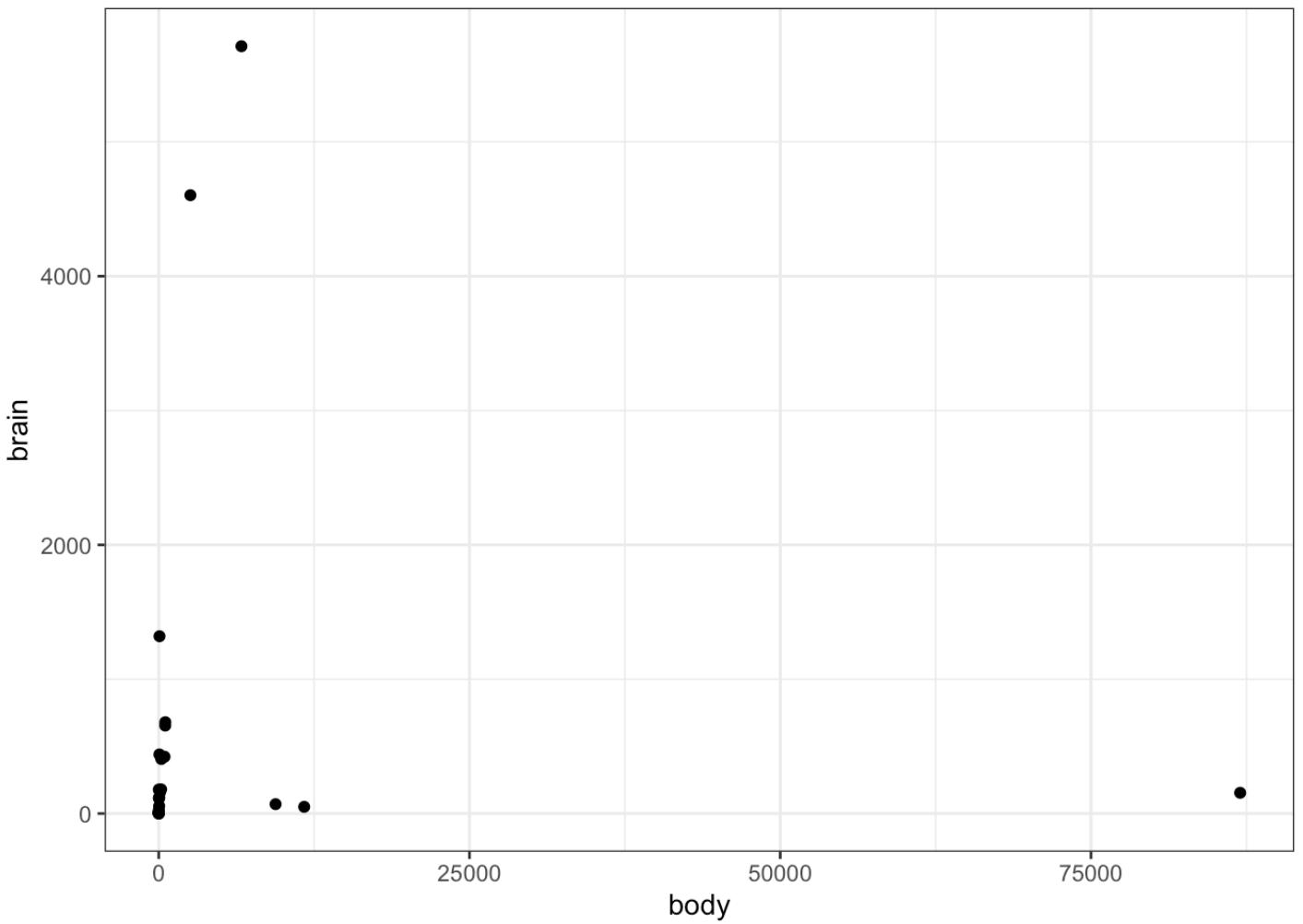


4.4.3.1.2 Log scaling

We consider another example where we want to plot the weights of the brain and body of different animals using the dataset `Animals`. This is what we obtain after creating a scatterplot.

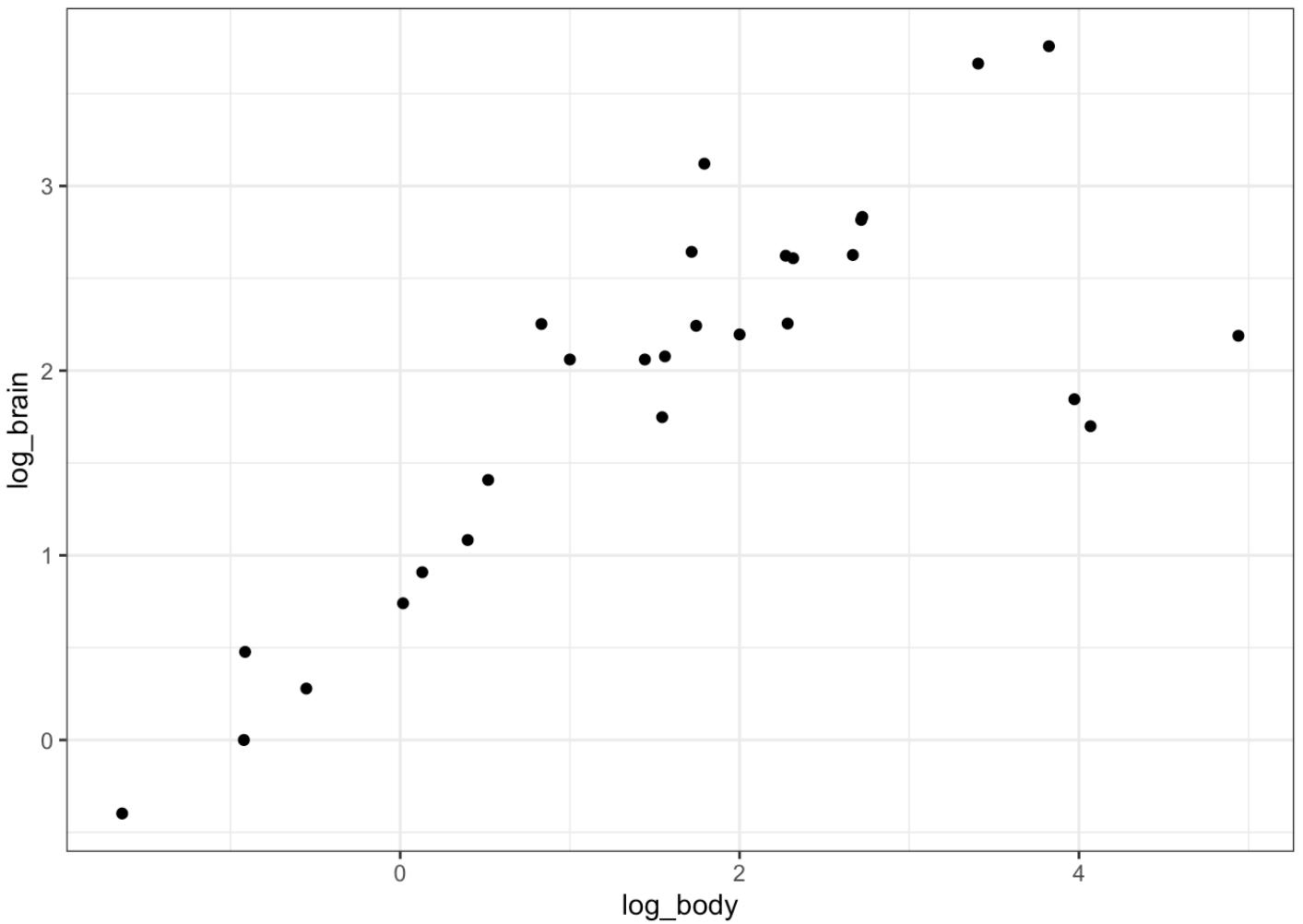
```
library(MASS) # to access Animals data sets
animals_dt <- as.data.table(Animals)

ggplot(animals_dt, aes(x = body, y = brain)) + geom_point() + mytheme
```



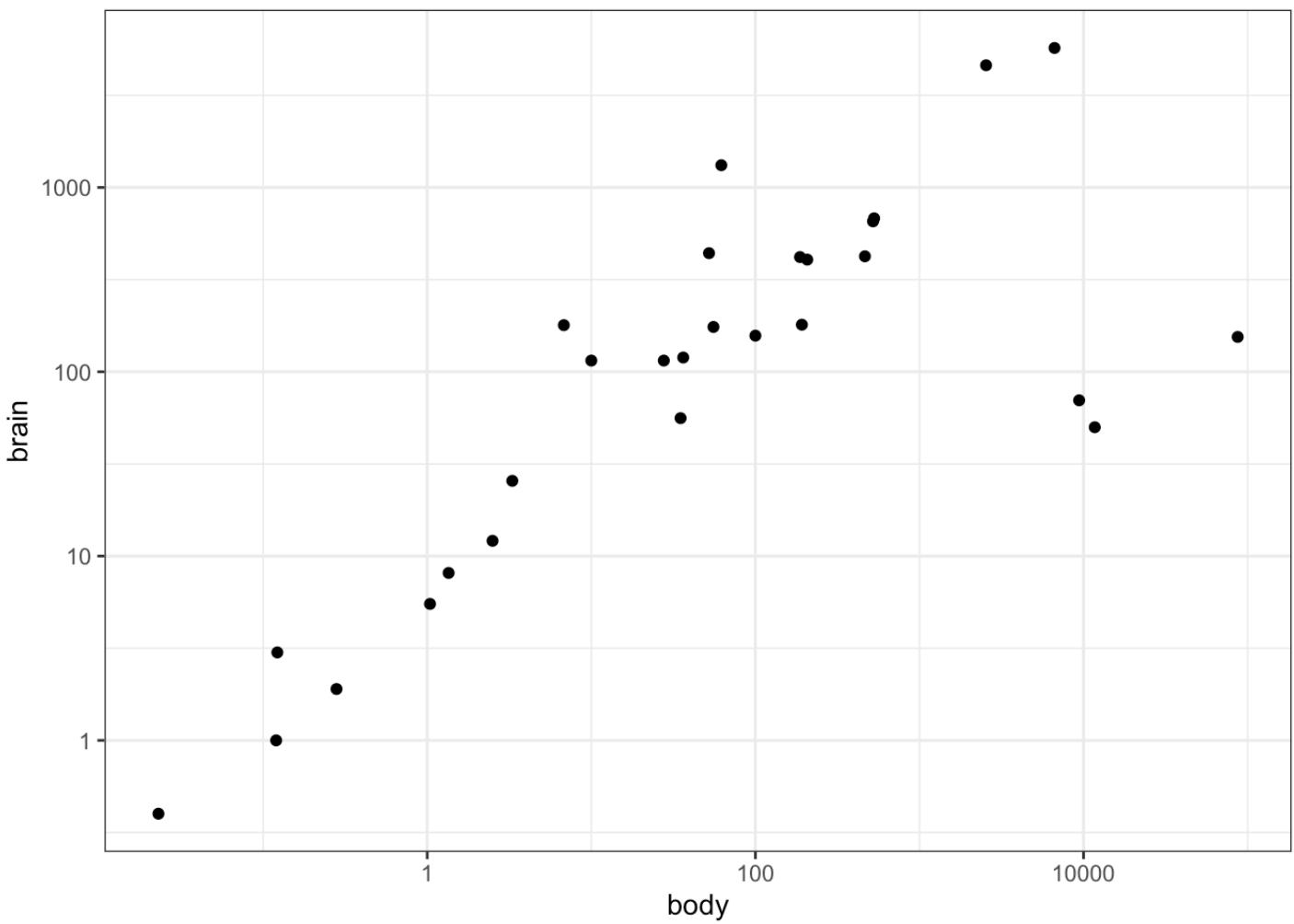
We can clearly see that there are a few points which are notably larger than most of the points. This makes it harder to interpret the relationships between most of these points. In such cases, we can consider logarithmic transformations and/or scaling. More precisely, a first idea would be to manually transform the values into a logarithmic space and plot the transformed values instead of the original values:

```
animals_dt[, c('log_body', 'log_brain') := list(log10(body), log10(brain)) ]  
  
ggplot(animals_dt, aes(x = log_body, y = log_brain)) + geom_point() + mytheme
```



Alternatively, `ggplot2` offers to simply scale the data without the need to transform. This can be done with the help of the functions `scale_x_log10()` and `scale_y_log10()` which allow appropriate scaling and labeling of the axes:

```
ggplot(animals_dt, aes(x = body, y = brain)) + geom_point() +  
  scale_x_log10() + scale_y_log10() + mytheme
```

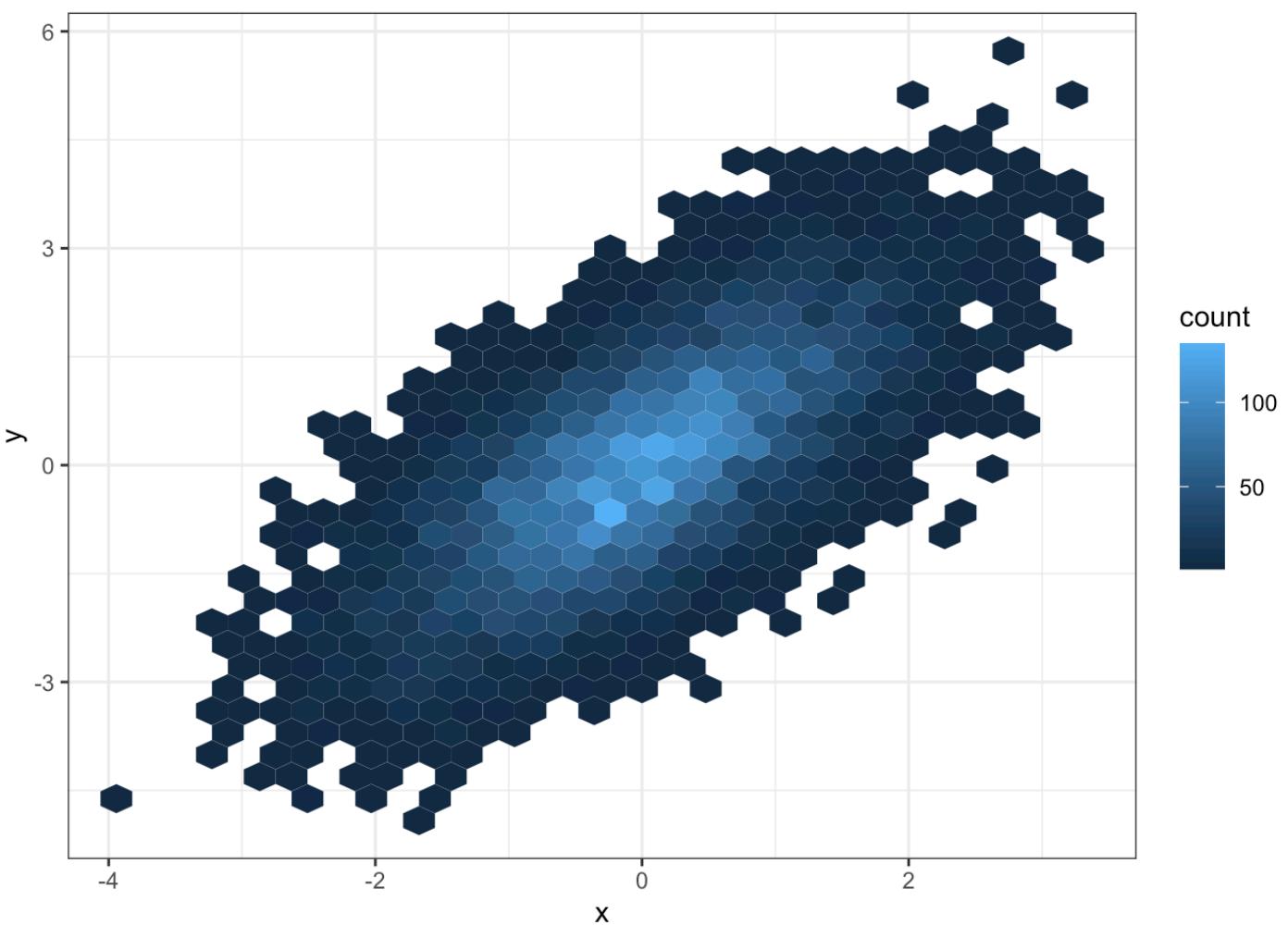


4.4.3.2 Density plots in 2D

Using scatterplots can become **problematic when dealing with a huge number of points**. This is due to the fact that points may overlap and we cannot clearly see how many points are at a certain position. In such cases, a 2D density plot is particularly well suited. This plot counts the number of observations within a particular area of the 2D space.

The function `geom_hex()` can be used for creating 2D density plots in R:

```
x <- rnorm(10000); y=x+rnorm(10000)
data.table(x, y) %>% ggplot(aes(x, y)) +
  geom_hex() + mytheme
```

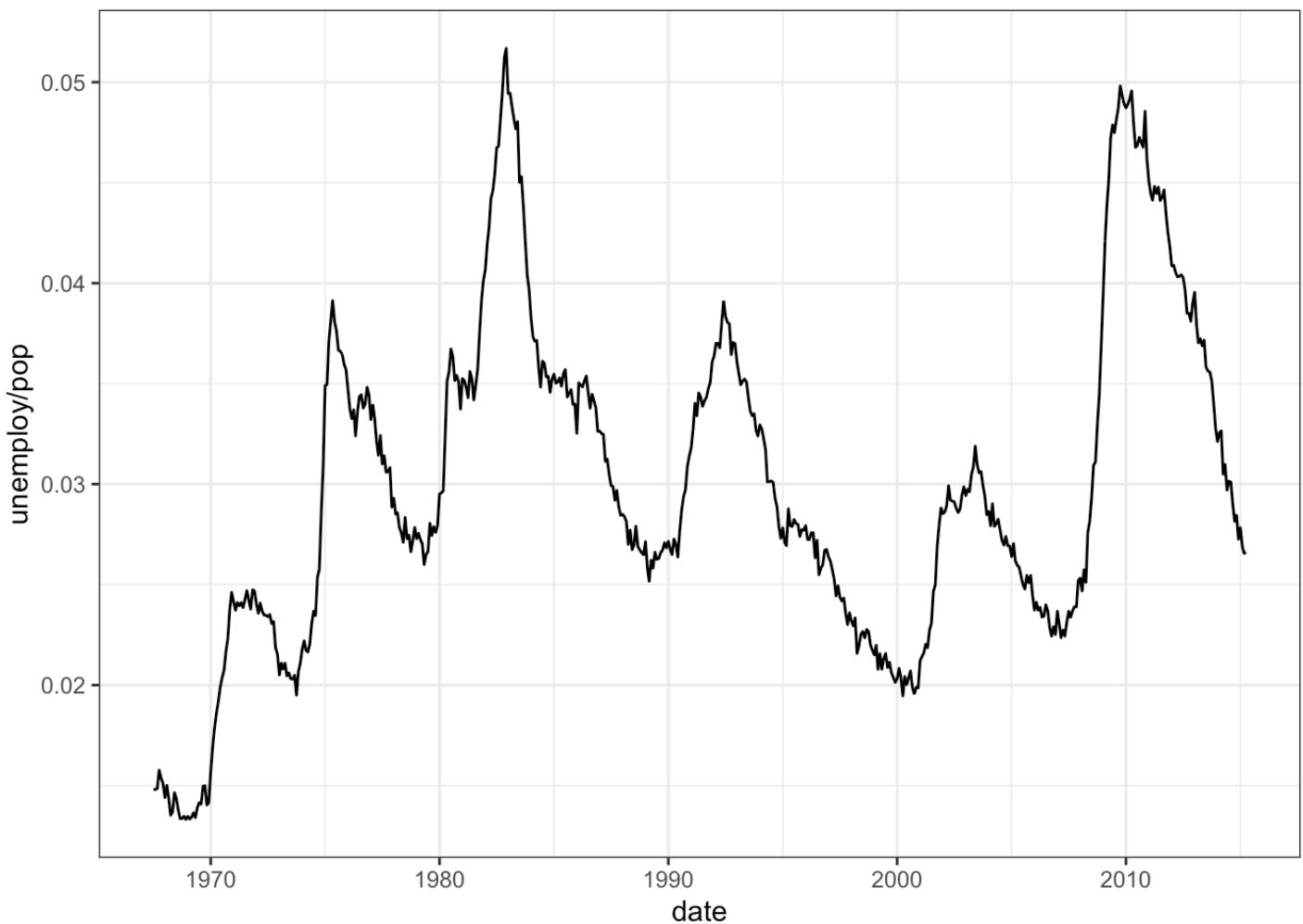


4.4.3.3 Line plots

A line plot can be considered for connecting a series of individual data points or to display the trend of a series of data points. This can be particularly useful to show the shape of data as it flows and changes from point to point. We can also show the strength of the movement of values up and down through time.

As an example we show the connection between the individual datapoints of unemployment rate over the years:

```
ggplot(economics, aes(date, unemploy / pop)) +
  geom_line() + mytheme
```



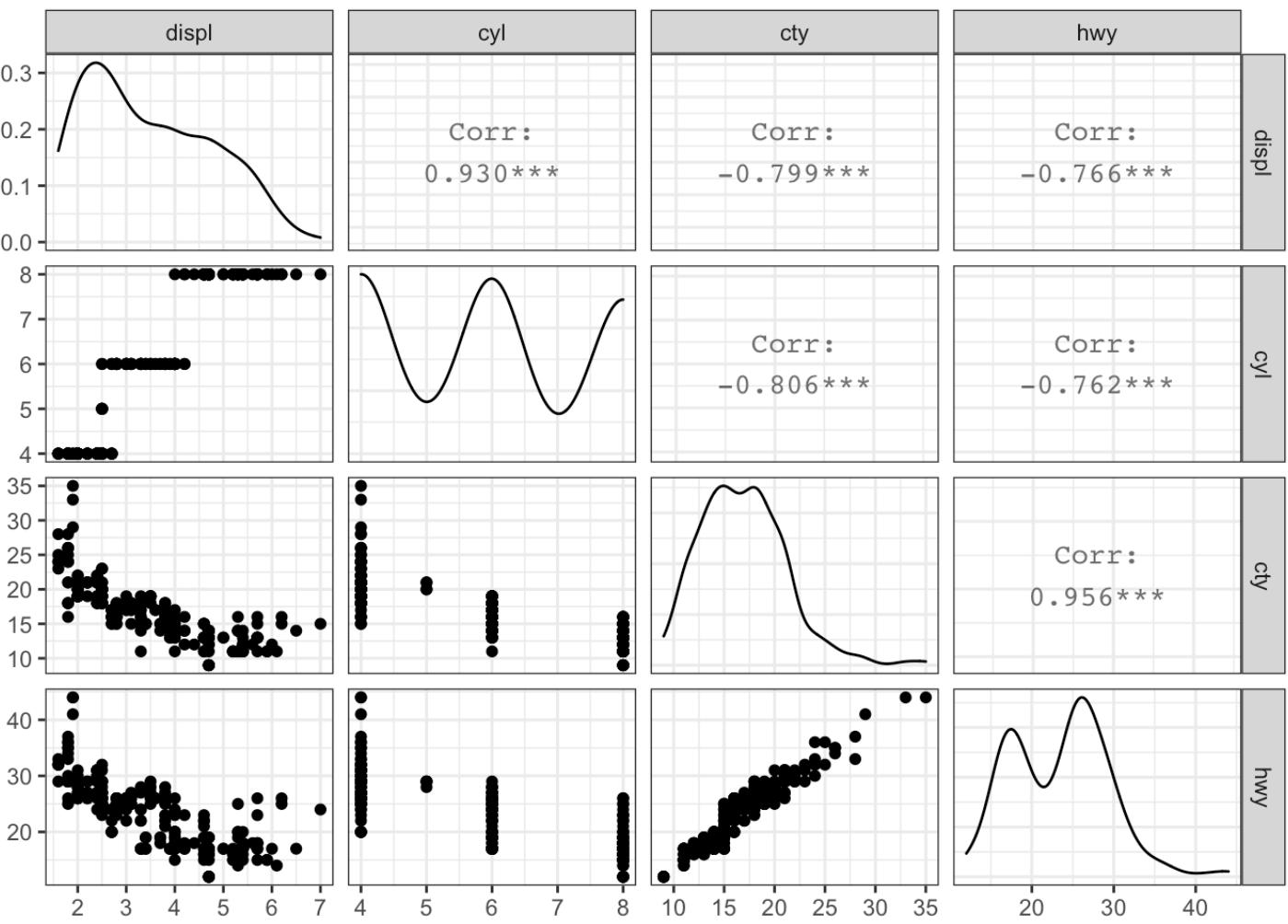
4.5 Further plots for low dimensional data

4.5.1 Plot matrix

A so-called plot matrix is useful for exploring the distributions and correlations of a few variables in a matrix-like representation. Here, for each pair of considered variables, a scatterplot is created and the correlation coefficient is computed. For every single variable, a density plot is created for showing the respective distribution.

We can use the function `ggpairs()` from the library `GGally` for constructing plot matrices. As an example, we analyze the variables `displ`, `cyl`, `cty` and `hwy` from the dataset `mpg`:

```
library(GGally)
ggpairs(mpg, columns = c("displ","cyl","cty","hwy")) + mytheme
```



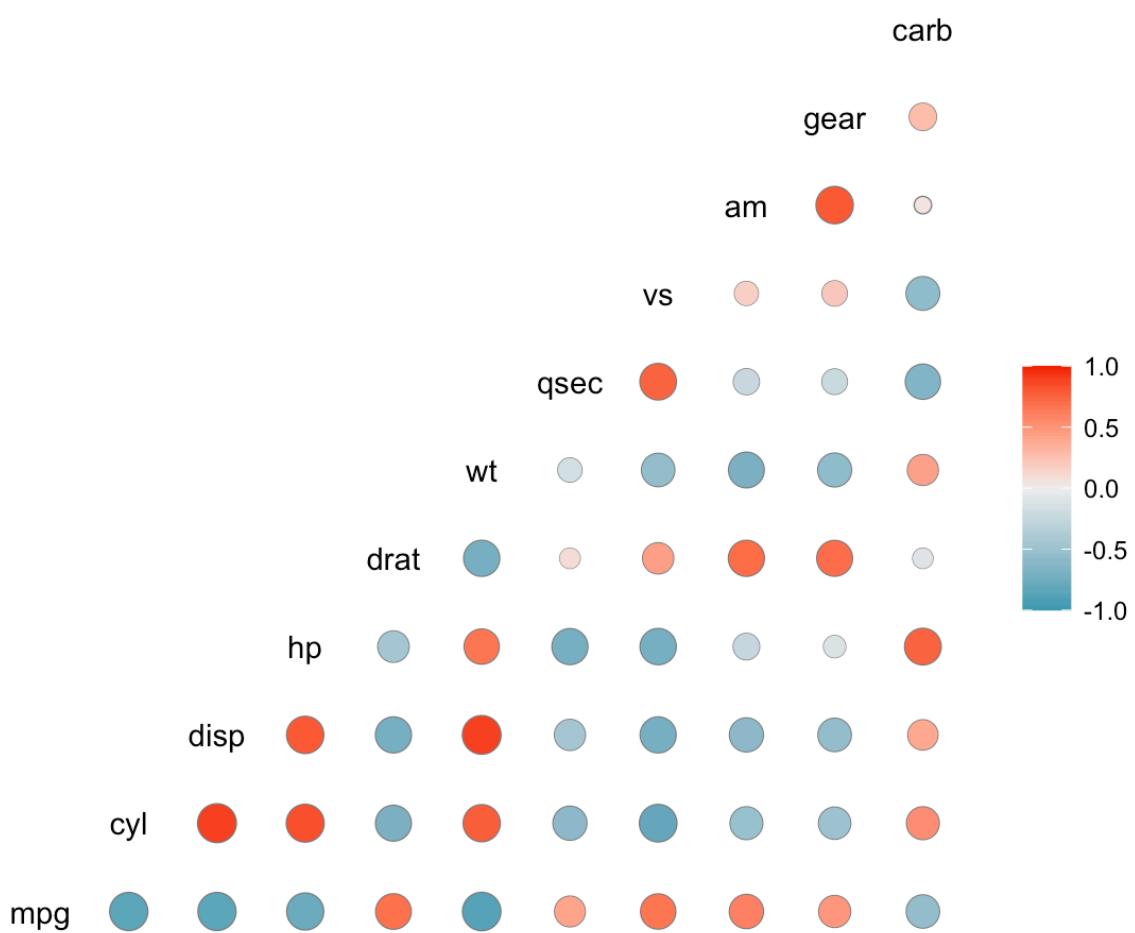
We remark that this plot is not well suited for comparing more than a few variables.

4.5.2 Correlation plot

A correlation plot is a **graphical representation of a correlation matrix**. It is useful to highlight the most correlated variables in a dataset. In this plot, correlation coefficients are colored according to the value. A correlation matrix can be also reordered according to the degree of association between variables.

Correlation plots are also called “corrgrams” or “correlograms”. As an example, we visualize the correlation between the variables of the dataset `mtcars` with the help of the function `ggcorr()` from the library `GGally`:

```
ggcorr(mtcars, geom = 'circle')
```



4.6 Summary

This first chapter of data visualization covered the basics of the grammar of graphics and `ggplot2` to plot low dimensional data. We introduced the different types of plots such as histograms, boxplots or barplots and discussed when to use which plot.

4.7 Resources

- The ggplot book: <https://ggplot2-book.org/>
- Plotting libraries:
 - <http://www.r-graph-gallery.com/portfolio/ggplot2-package/>
 - <http://ggplot2.tidyverse.org/reference/>
 - <https://plot.ly/r/>
 - <https://plot.ly/ggplot2/>
- Udacity's Data Visualization and D3.js
 - <https://www.udacity.com/courses/all>
- Graphics principles

- <https://onlinelibrary.wiley.com/doi/full/10.1002/pst.1912>
- <https://graphicsprinciples.github.io/>

4. https://en.wikipedia.org/wiki/Scientific_method ↪

Chapter 5 High dimensional visualizations

In this chapter, we turn our attention to the visualization of high-dimensional data with the aim to discover interesting patterns. We cover heatmaps, i.e., image representation of data matrices, and useful re-ordering of their rows and columns via clustering methods. To scale up visualization to very high-dimensional data, we furthermore introduce Principal Component Analysis as a dimension reduction technique.

5.1 Notations

Lower cases are used for scalars (e.g. x), bold lower cases for vectors (e.g. \mathbf{x}) and bold upper cases for matrices (e.g. \mathbf{X}). The transpose of a matrix or of a vector is denoted with a T-superscript (e.g. \mathbf{X}^\top). The Euclidean norm of vector \mathbf{x} is denoted $\|\mathbf{x}\|$.

Methods of this chapter assume numeric variables. If encountered, categorical variables can be transformed to numeric variables by one-hot encoding.⁵

We denote n the number of observations, p the number of variables, and \mathbf{X} the $n \times p$ data matrix.

5.2 Data matrix preparation

We use a subset of the base R `mtcars` dataset consisting of 10 rows (cars) and four selected variables. We store this data into the numeric matrix `mat`. For ease, we give full names (rather than abbreviations) to the columns and keep the row names (car names).

```
library(data.table)
mat <- as.matrix(mtcars[1:10, c("mpg", "carb", "hp", "wt")])
rownames(mat) <- rownames(mtcars)[1:10]
colnames(mat) <- c("Miles.per.gallon", "Carburetor", "Horsepower", "Weight")
head(mat) # A look at the first rows
```

```

##          Miles.per.gallon Carburetor Horsepower
## Mazda RX4           21.0         4        110
## Mazda RX4 Wag      21.0         4        110
## Datsun 710          22.8         1         93
## Hornet 4 Drive     21.4         1        110
## Hornet Sportabout   18.7         2        175
## Valiant              18.1         1        105
##                  Weight
## Mazda RX4          2.620
## Mazda RX4 Wag      2.875
## Datsun 710          2.320
## Hornet 4 Drive     3.215
## Hornet Sportabout   3.440
## Valiant              3.460

```

5.3 Heatmaps

Beyond 5 to 10 variables, matrix scatterplots cannot be rendered with enough resolution to be useful. However, **heatmaps** which simply display **data matrices as an image by color-coding its entries**, become handy. Heatmaps allow visualization of data matrices of up to ca. 1,000 rows and columns (order of magnitude), i.e the pixel resolution of your screen (and maybe of your eyes!).

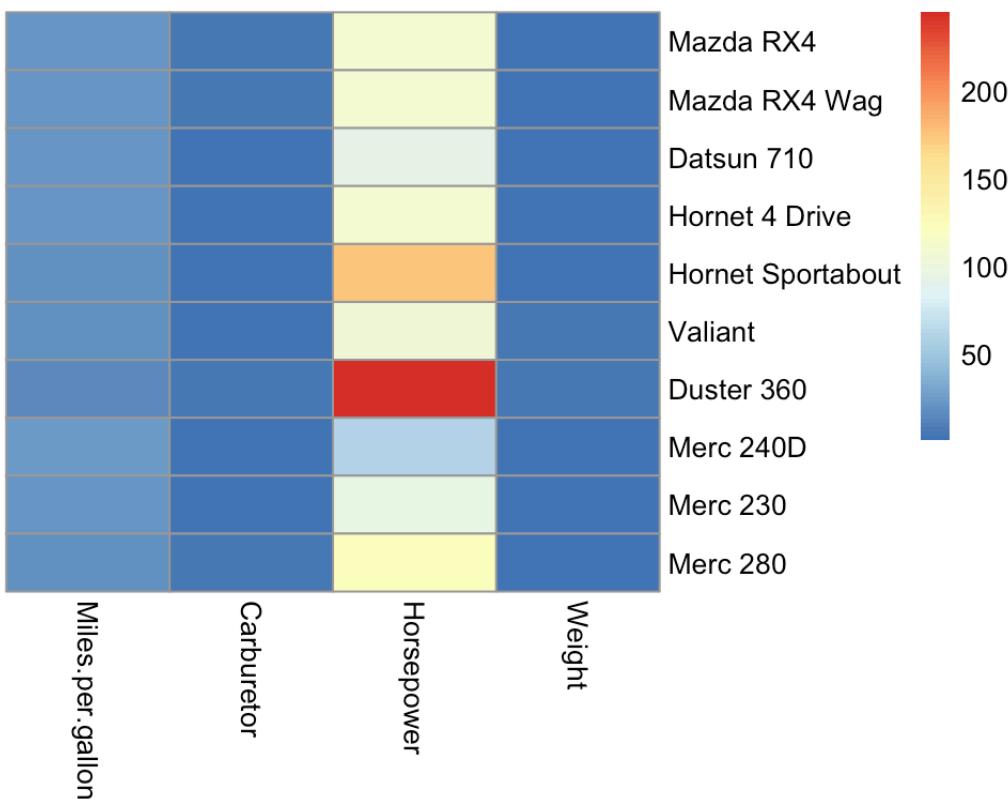
To draw heatmaps, we recommend using the library `pheatmap` (pretty heatmaps), which offers convenient functionalities in particular for clustering (See Section [5.4.2](#)).

Here is a basic call to `pheatmap` on our data matrix `mat`:

```

library(pheatmap) ## pretty heatmap
pheatmap(mat, cluster_rows=FALSE,
         cluster_cols=FALSE)

```



Strikingly, the horsepower variable saturates the color scale because horsepower lives in a different scale than the other variables. Consequently, we barely see variations in the other variables.

5.3.1 Centering and scaling variables

Bringing variables to a common scale is useful for visualization but also for computational and numerical reasons. Moreover, it makes analysis independent of the units chosen. For instance the `mtcars` dataset provide car weights in 1,000 pounds and gas consumption in miles per gallon. We would certainly want our analysis to be the same if these variables were expressed with the metric system.

The widely used operations to bring variables to a same scale are:

- **centering**: subtracting the mean
- **standard scaling or Z-score normalization**: centering then dividing by the standard deviation

These operations are implemented in the base R function `scale()` and are often offered as parameters of other functions. Scaling is usually done for variables (data matrix columns). However, in some application contexts, row-scaling can be considered as well.

With `pheatmap` we can also scale the data by rows or columns by setting the argument `scale` accordingly. We do it here in the classical way, by column:

```
pheatmap(mat, cluster_rows=FALSE, cluster_cols=FALSE,
         scale='column')
```

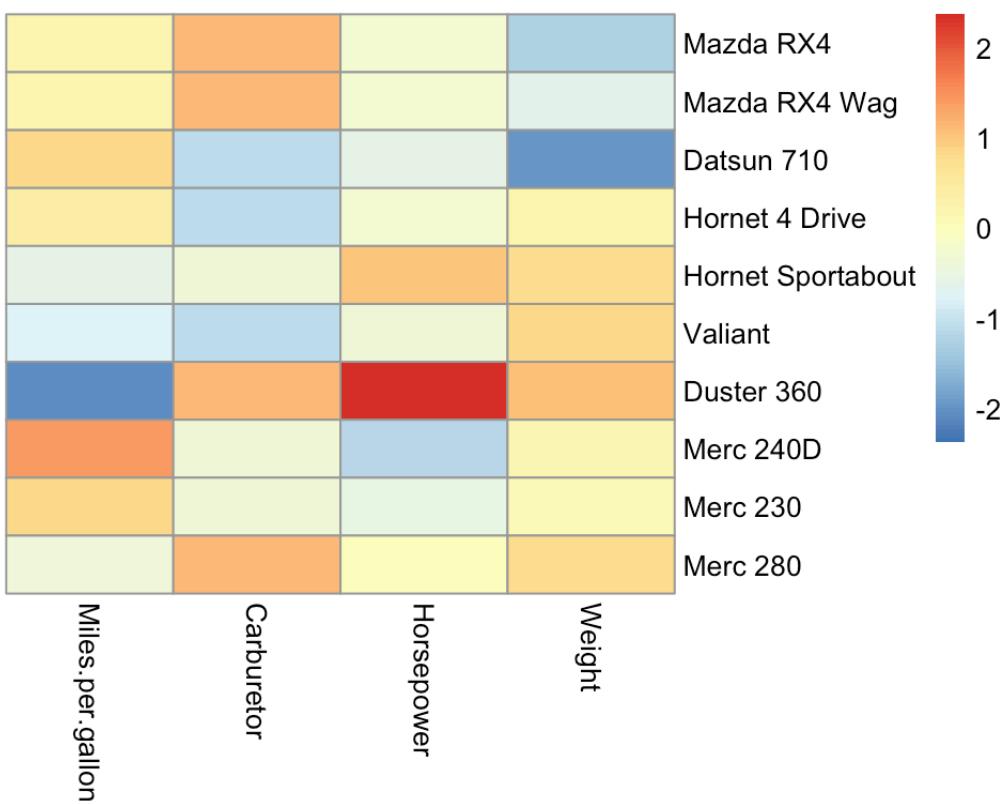


Figure 5.1: Scaled heatmap.

Scaling allows us to appreciate the variation for all variables. The default color scale is centered on 0. Because each column is centered, we find positive (red-ish) and negative (blue-ish) values in each column.

5.4 Clustering

While all data are rendered in the Figure 5.1, it is hard to see a pattern emerging. Which cars are similar to each others? Which variables are similar to each other? Clustering is the task of grouping observations by similarities. Clustering helps finding patterns in data matrices. Clustering can also be applied to variables, by simply applying clustering methods on the transpose of the data matrix.

There are several clustering algorithms. In the following sections, we explain two widely used clustering methods: K-means clustering and hierarchical clustering.

5.4.1 K-Means clustering

5.4.1.1 Objective

K-Means clustering aims to partition the observations into K non-overlapping clusters. The number of clusters K is predefined. The clusters C_1, \dots, C_K define a partition of the observations, i.e., every observation belongs to one and only one cluster. To this end, one makes use of so-called cluster centroids, denoted μ_1, \dots, μ_K , and associates each observation to its closest centroid (Figure 5.2).

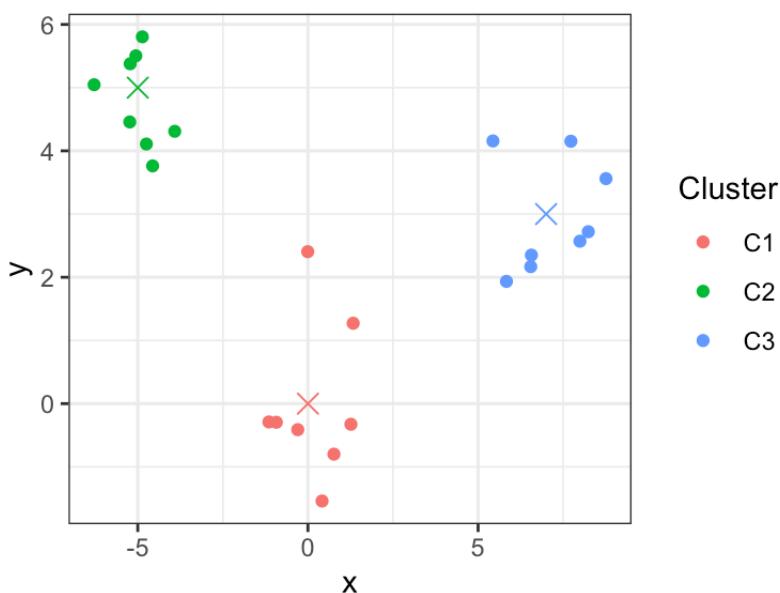


Figure 5.2: K-means clustering partitions observations into K clusters (here K=3) by associating each observation to its closest centroids (crosses).

Formally, one aims to determine the clusters C_1, \dots, C_K and the centroids μ_1, \dots, μ_K in order to minimize the within-cluster sum of squares:

$$\min_{C_1, \dots, C_K, \mu_1, \dots, \mu_K} \sum_{k=1}^K \sum_{i \in C_k} \|\mathbf{x}_i - \mu_k\|^2 \quad (5.1)$$

where $\|\mathbf{x}_i - \mu_k\|^2 = \sum_{j=1}^p (x_{i,j} - \mu_{k,j})^2$ is the squared Euclidean distance between observation \mathbf{x}_i (the i -th row vector of the data matrix) and the centroid μ_k .

5.4.1.2 Algorithm

The minimization problem (Equation (5.1)) is difficult because of the combinatorial number of partitions of n observations into K clusters. However, two useful observations can be made.

First, if we assume that the positions of the centroids μ_k are given, then each summand $\|\mathbf{x}_i - \mu_k\|^2$ in Equation (5.1) can be minimized by including the observation \mathbf{x}_i to the cluster of its closest centroid. The number of clusters K depends on the dataset.

Second, if we now assume that the clusters are given, then the values of the centroids that minimize $\sum_{i \in C_k} \|\mathbf{x}_i - \mu_k\|^2$ are the observation means, i.e. $\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} \mathbf{x}_i$. Hence, the centroids are the cluster means, giving the name of the algorithm.

These two observations lead to an iterative algorithm that provides in practice good solutions, even though it does not guarantee to find the optimal solution.

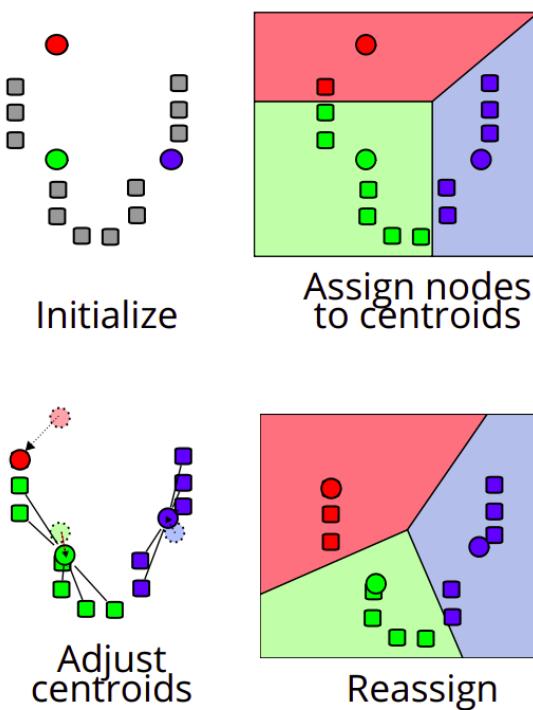


Figure 5.3: K-mean algorithm. Source: https://en.wikipedia.org/wiki/K-means_clustering

K-Means algorithm (Figure 5.3)

1. Choose the K initial centroids (one for each cluster). Different methods such as sampling random observations are available for this task.
2. Assign each observation \mathbf{x}_i to its nearest centroid by computing the Euclidean distance between each observation to each centroid.
3. Update the centroids μ_k by taking the mean value of all of the observations assigned to each previous centroid.
4. Repeat steps 2 and 3 until the difference between new and former centroids is less than a previously defined threshold.

At every iteration, and at every step 2 and 3, the within-cluster sum of squares (Equation (5.1)) decreases.

However, there is no guarantee to reach the optimal solution. In particular, the final clustering depends on the initialization (step 1). To not overly depend on the initialization, the K-means algorithm is typically executed with different random initializations. The clustering with lowest within-cluster sum of squares is then retained.

5.4.1.3 Considerations and drawbacks of K-Means clustering

We have to make sure that the following assumptions are met when performing k-Means clustering (Figure 5.4, Source: scikit-learn⁶):

- The number of clusters K is properly selected
- The clusters are isotropically distributed, i.e., in each cluster the variables are not correlated and have equal variance
- The clusters have equal (or similar) variance
- The clusters are of similar size

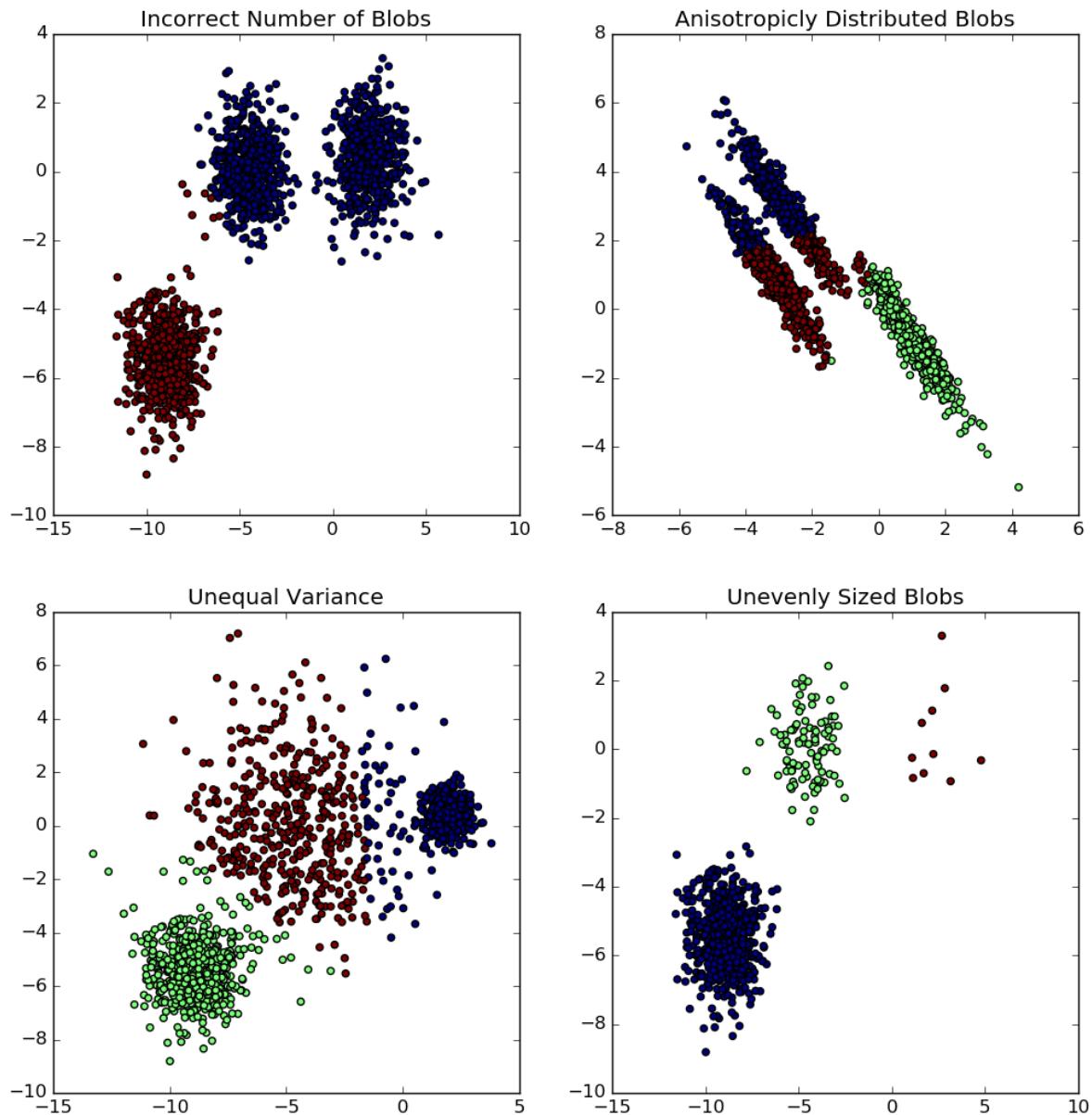


Figure 5.4: Situations for which K-means fail to retrieve underlying clusters

5.4.1.4 K-Means clustering in R

Let us now apply K-means to the `mat` data matrix searching for 2 clusters. This can be easily achieved with the function `kmeans()`. While not necessary, it is a good idea to scale the variables, in order not to give the variables with larger scales too much importance (by dominating the Euclidean distances). Another way to look at it, is that the scaling reduces to some extent the problem of anisotropic clusters. In the following code, we first scale the data matrix with `scale()`. We also use the argument `nstart` of `kmeans()` to perform multiple random initializations.

```

k <- 2
X <- scale(mat) # use the scaled variables for the clustering
clust_km <- kmeans(X, k, nstart = 20) # K-means 20 times
clust_km$cluster # clusters of the best clustering

```

	Mazda RX4	Mazda RX4 Wag	Datsun 710
##	2	2	2
##	Hornet 4 Drive	Hornet Sportabout	Valiant
##	2	1	1
##	Duster 360	Merc 240D	Merc 230
##	1	2	2
##	Merc 280		
##	1		

We now update our heatmap with the results of the clustering. We make use of the `annotation_row` argument of `pheatmap()` which generates color-coded row annotations on the left side of the heatmap. We furthermore order the rows of the data matrix by cluster.

```

# create the row annotation data frame
row.ann <- data.frame(
  kmeans = paste0("C", clust_km$cluster) # we call the cluster C1,...,CK.
)

# rownames are used to match the matrix rows with the row annotation data frame.
# We can now safely reorder the rows of X.
rownames(row.ann) <- rownames(X)

# o: order of the rows to have increasing cluster number
o <- order(clust_km$cluster)

pheatmap(
  X[o,],      # X with ordered rows according to cluster number
  scale='none', # no need to scale, X is scaled
  annotation_row = row.ann,
  cluster_rows=FALSE, cluster_cols=FALSE
)

```

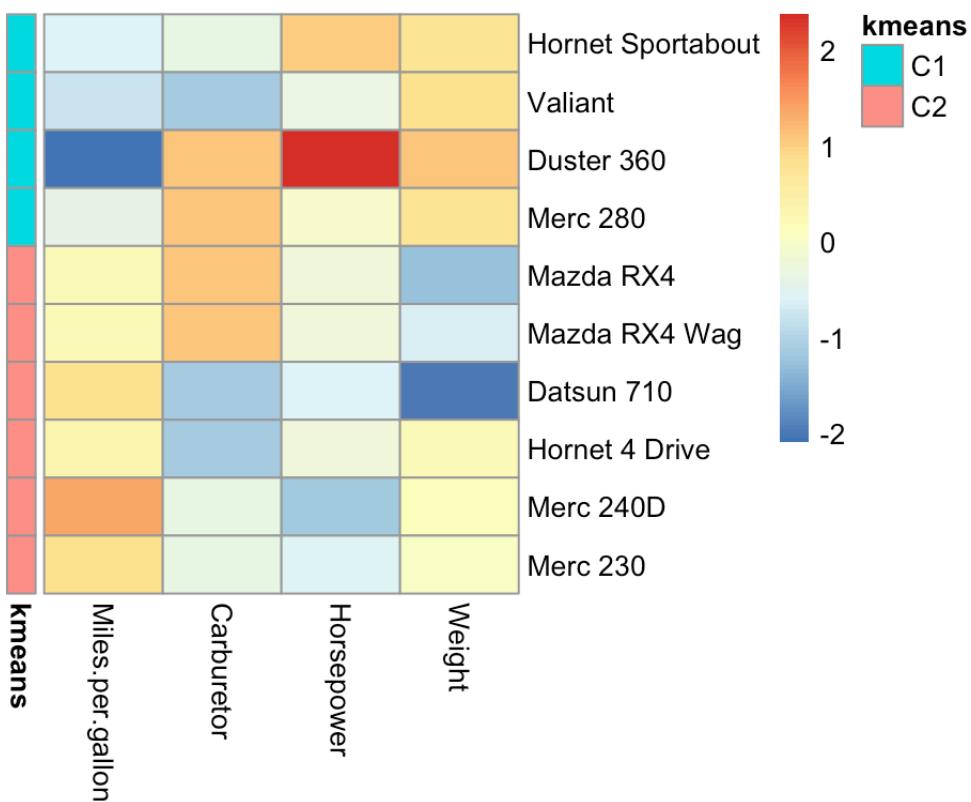


Figure 5.5: Pretty heatmap with K-mean cluster annotation.

Cluster C_1 appears to group the heavy, powerful and gas-consuming cars and cluster C_2 the light, less powerful and more economic cars.

5.4.2 Hierarchical clustering

A major limitation of the K-means algorithm is that it relies on a predefined number of clusters. What if the interesting number of clusters is larger or smaller? Hierarchical clustering allows exploring multiple levels of clustering granularity at once by computing nested clusters. It results in a tree-based representation of the observations, called a dendrogram.

Figure 5.6 shows an example of a hierarchical clustering using two variables only.

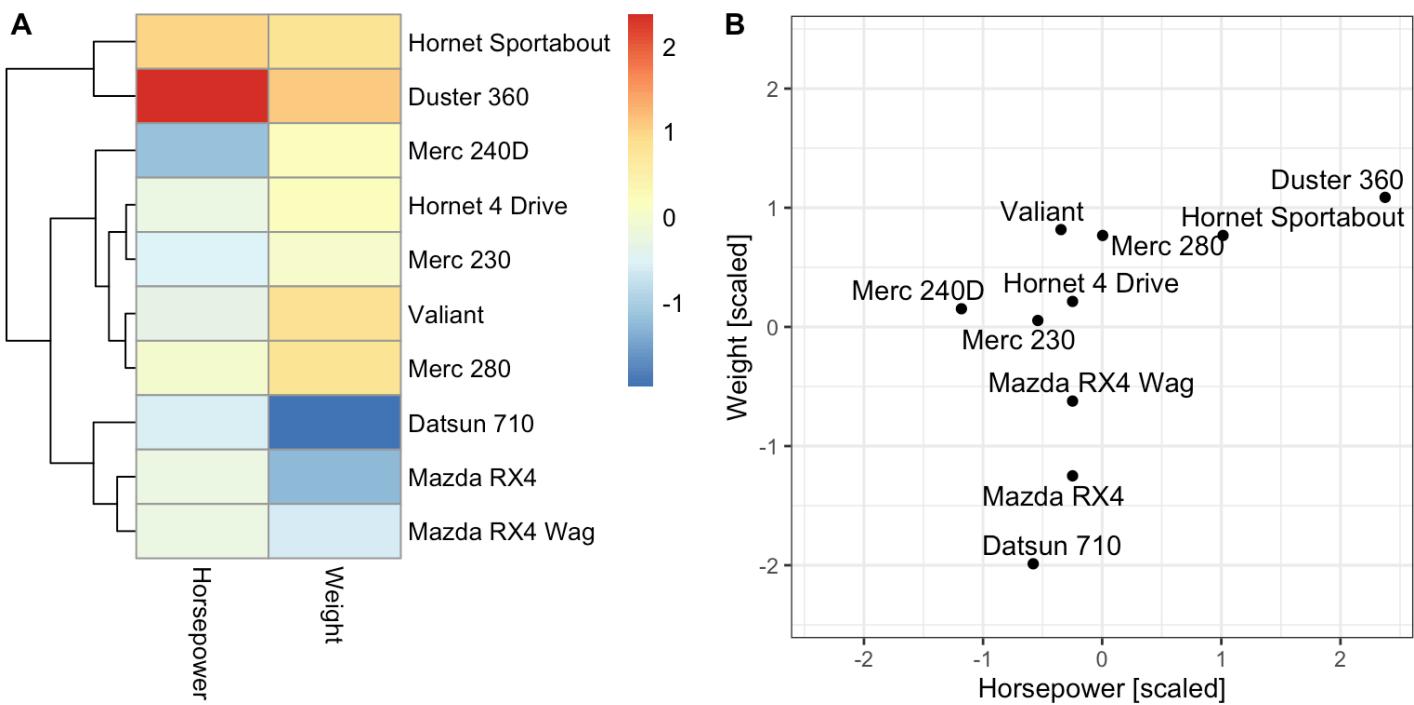


Figure 5.6: Example of a hierarchical clustering for two variables. (A) Dendrogram (tree on the left) along with the heatmap. (B) Scatterplot of the same data as in (A).

Unlike with K-means, there is **no objective function associated with hierarchical clustering**. Hierarchical clustering is simply defined by how it operates.

We describe bottom-up or agglomerative hierarchical clustering:

1. Initialization: Compute all the $n(n - 1)/2$ pairwise dissimilarities between the n observations. Treat each observation as its own cluster. A typically dissimilarity measure is the Euclidean distance. Other dissimilarities can be used (1-correlation), Manhattan distance, etc.
2. For $i = n, n - 1, \dots, 2$:
 - Fuse the two clusters that are least dissimilar. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.
 - Compute the new pairwise inter-cluster dissimilarities among the $i - 1$ remaining clusters using the linkage rule.

The **linkage rules** define **dissimilarity between clusters**. Here are four popular linkage rules:

- **Complete:** The dissimilarity between cluster A and cluster B is the largest dissimilarity between any element of A and any element of B.
- **Single:** The dissimilarity between cluster A and cluster B is the smallest dissimilarity between any element of A and any element of B. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
- **Average:** The dissimilarity between cluster A and cluster B is the average dissimilarity between any element of A and any element of B.
- **Centroid:** The dissimilarity between cluster A and cluster B is the dissimilarity between the centroids (mean vector) of A and B. Centroid linkage can result in undesirable inversions.

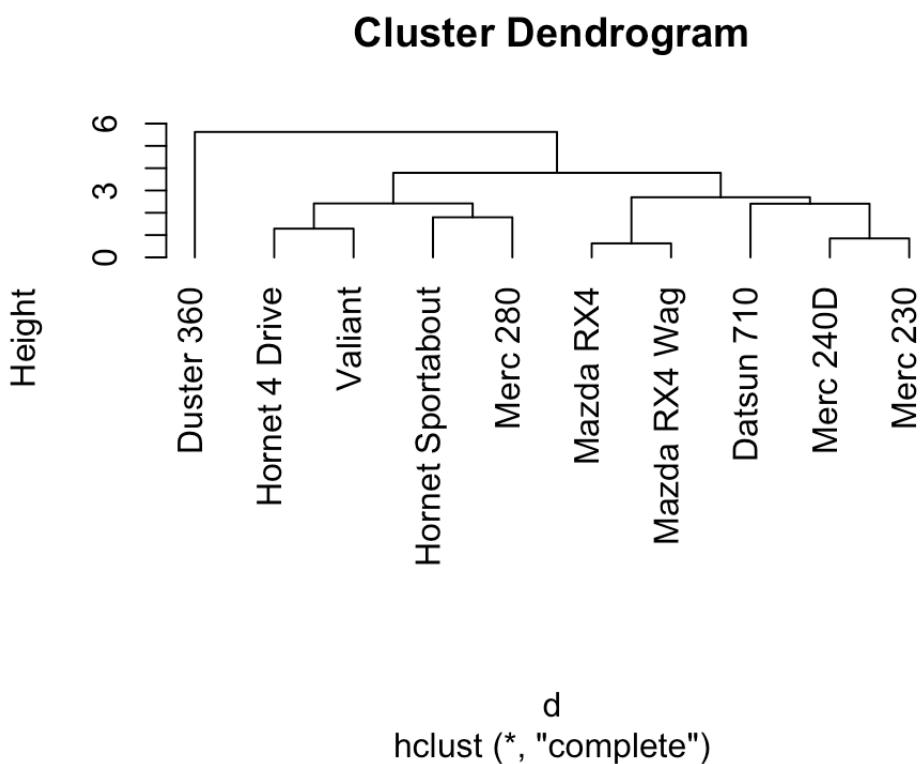
5.4.2.1 Hierarchical clustering in R

In R, Hierarchical clustering can be performed in two simple steps. First, we compute the distance between observations (rows of a `data.table`) across variables (columns of a `data.table`) with the help of the function `dist()`. Here, the Euclidean distance between rows is computed by default. Alternatives include the Manhattan or Minkowski distance. As for K-means, it is recommended to work on scaled variables to not give too much importance to variables with large variance. We therefore compute the pairwise Euclidean distance to our scaled data matrix `X`. Second, we use the resulting Euclidean distance matrix as a dissimilarity matrix to perform Hierarchical clustering with the help of the function `hclust()`. We use here the default linkage rule (`complete`).

```
d <- dist(X) # compute distance matrix with default (Euclidean)
hc <- hclust(d) # apply hierarchical clustering with default (complete linkage rule)
```

The results of hierarchical clustering can be shown using a dendrogram (i.e., a tree representation). Here, observations that are determined to be similar by the clustering algorithm are displayed close to each other in the `x`-axis. The height in the dendrogram at which two clusters are merged represents the distance between those two clusters.

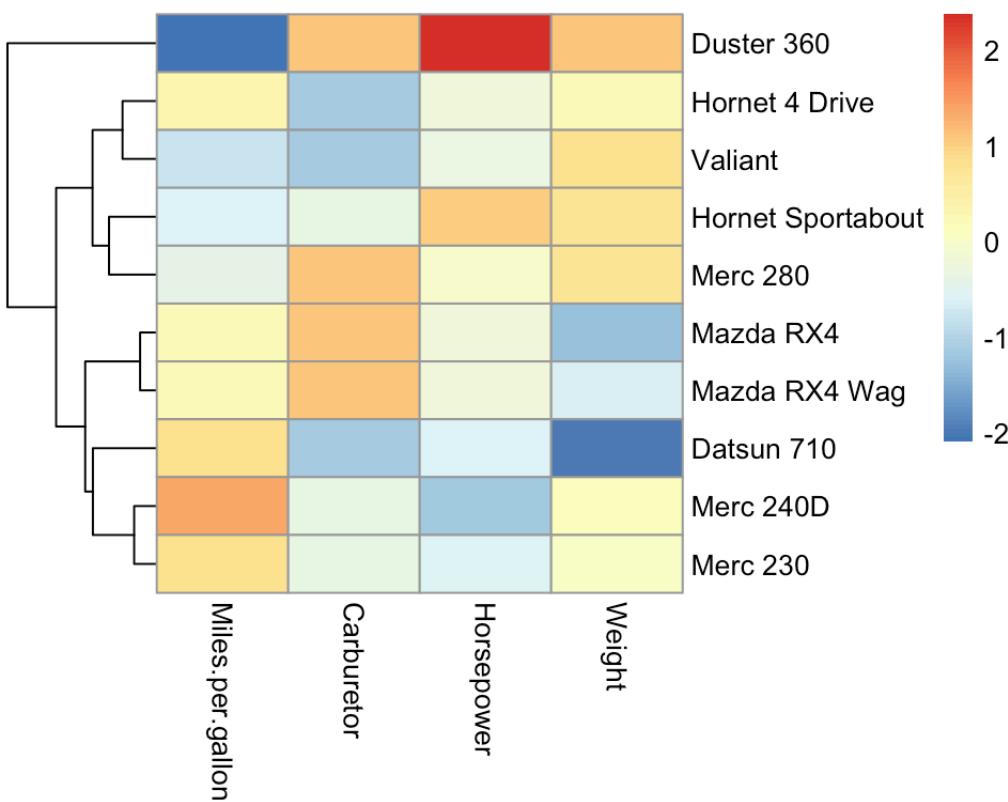
```
plot(hc, hang=-1) # hang=-1 align observation labels at the bottom of the dendrogram
```



5.4.2.2 Pretty heatmaps including hierarchical clustering

As illustrated before, the library `pheatmap` enables the easy creation of heatmaps. In the previous example, we set the parameters `cluster_rows` and `cluster_cols` to `FALSE` to avoid the default computation of hierarchical clustering. If we want to include hierarchical clustering, we can simply set these parameters to `TRUE` or let R consider its default values.

```
pheatmap(X, cluster_rows=TRUE, cluster_cols=FALSE, scale='none')
```



Compared to K-means (Figure 5.5), the hierarchical clustering shows useful different degrees of granularity of the clusters. We see the most similar cars grouping together (the Mercedes 230 and the Mercedes 240D, as well as the Mazda RX4 and the Mazda RX4 Wag). At the high level, the Duster 360 stands out as an outlier.

5.4.2.3 Cutting the tree

After having inspected the result of a hierarchical clustering, it is often interesting to define distinct clusters by cutting the dendrogram at a certain height.

Typically, one cuts dendrogram either at a given height, or in order to obtain a certain number of clusters. The function `cutree(tree, k = NULL, h = NULL)` supports both options. Here is an example of cutting the dendrogram to get 3 clusters.

```
clust_hc <- cutree(hc, k=3)
clust_hc
```

```

##      Mazda RX4     Mazda RX4 Wag       Datsun 710
##            1             1                 1
##    Hornet 4 Drive Hornet Sportabout     Valiant
##            2             2                 2
##    Duster 360     Merc 240D       Merc 230
##            3             1                 1
##    Merc 280
##            2
##
```

5.4.2.4 Differences between K-Means and hierarchical clustering

Both K-means and hierarchical clustering are well established and widely used. Here, we briefly state a few differences that may be considered when deciding which algorithm to apply in practice.

The time complexity of K-Means clustering is linear, while that of hierarchical clustering is quadratic. This implies that hierarchical clustering can not handle extremely large datasets as efficiently as K-Means clustering.

In K-Means clustering, we start with a random choice of centroids for each cluster. Hence, the results produced by the algorithm depend on the initialization. Therefore, the results might differ when running the algorithm multiple times. Hierarchical clustering outputs reproducible results.

Another difference is that K-Means clustering requires the number of clusters a priori. In contrast, the number of clusters we find appropriate in hierarchical clustering can be decided a posteriori by interpreting the dendrogram.

5.4.3 Comparing clusterings with the Rand index

Let us first visualize the outcome of K-means and of the hierarchical clustering cut for 3 clusters thanks to the row annotation option of `pheatmap`.

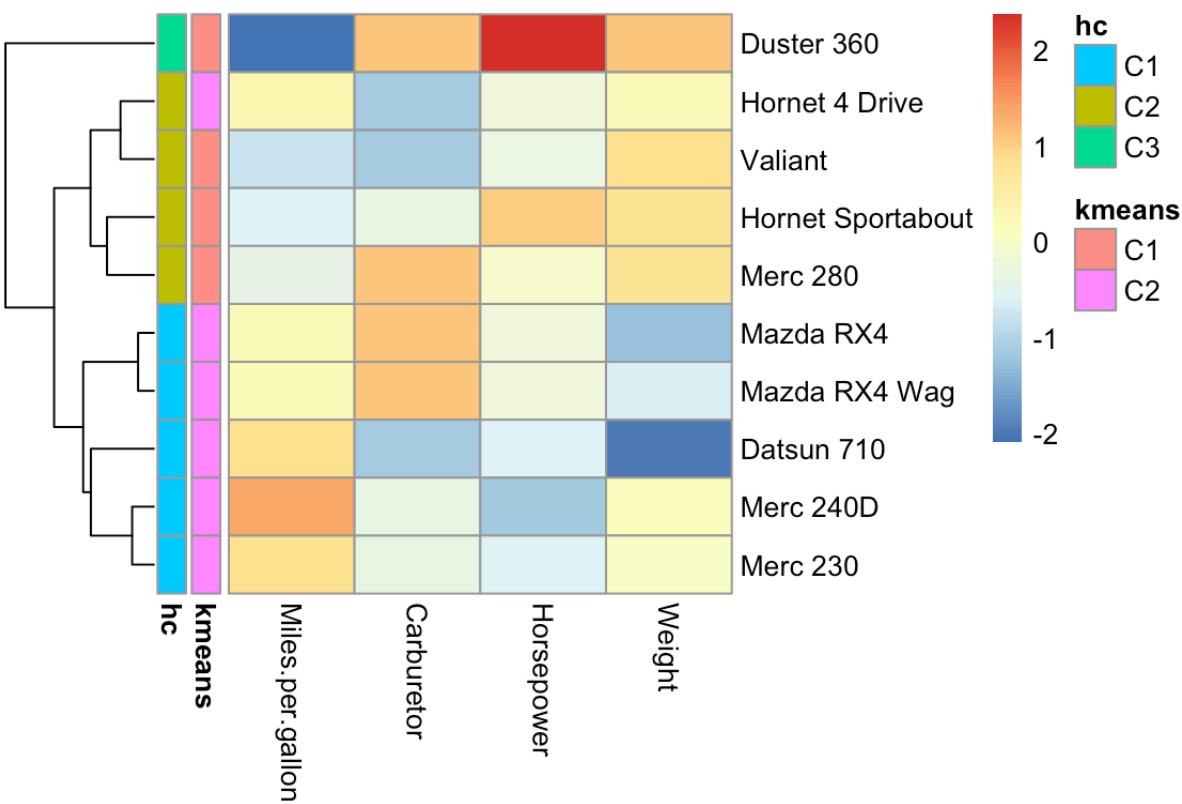
```

# create the row annotation data frame
row.ann <- data.frame(
  kmeans = paste0("C", clust_km$cluster),
  hc = paste0("C", clust_hc)
)

# rownames are used to match the matrix rows with the row annotation data frame.
# We can now safely reorder the rows of X.
rownames(row.ann) <- rownames(X)

pheatmap(
  X,
  scale='none', # no need to scale, X is scaled
  annotation_row = row.ann,
  cluster_rows=TRUE, cluster_cols=FALSE
)

```



Comparing clustering results is a challenging task. When clustering into two groups, we could use evaluation measures from classification, which we will introduce later. Moving from two partitions of the data into arbitrarily many groups requires new ideas.

We remark that a partition is, in our context, the result from a clustering algorithm and, therefore, the divided dataset into clusters. Generally, two partitions (from different clustering algorithms) are considered to be similar when many pairs of points are grouped together in both partitions.

The Rand index is a measure of the similarity between two partitions. Formally, we introduce the following definitions:

- $S = \{o_1, \dots, o_n\}$ a set of n elements (or observations)
- First partition $X = \{X_1, \dots, X_k\}$ of S into k sets
- Second partition $Y = \{Y_1, \dots, Y_l\}$ of S into l sets
- a number of **pairs** of elements of S that are **in the same set** in X and in Y
- b number of **pairs** of elements of S that are **in different sets** in X and in Y
- $\binom{n}{2}$ total number of pairs of elements of S

Then, the Rand index can be computed as

$$R = \frac{a + b}{\binom{n}{2}}$$

where $\binom{n}{2}$ (reads "n choose 2"), the binomial coefficient for $k = 2$, is the number of pairs of observations and is equal to:

$$\binom{n}{2} = \frac{(n-1) \cdot n}{2}.$$

5.4.3.1 Properties of the Rand index

By definition, the Rand index has values between 0 and 1, including them. A Rand index of 1 means that all pairs that are in the same cluster in the partition X are also in the same cluster in the partition Y and all pairs that are not in the same cluster in X are also not in the same cluster in Y . Hence, the two partitions are identical with a Rand index of 1. In general, the higher the Rand index, the more similar are both partitions.

5.4.3.2 Application of the Rand index

We can compute the Rand index between our k-means result and the cut of the hierarchical clustering for a given number of groups. See exercise sheet.

5.5 Dimensionality reduction with PCA

A heatmap is a visualization method of choice for data matrices as long as rows and columns are visually resolved because it satisfies the two main data visualization principles, i.e.,: i) having a high data/ink ratio and ii) showing the data as raw as possible.

However, beyond dimensions exceeding the thousands of variables, dimension reduction techniques are needed. The idea of dimension reduction is simple: if the dimension of our data p is too large, let us consider instead a representation of lower dimension q which retains much of the information of the dataset.

For Principal Component Analysis (Pearson, 1901), this representation is *the projection of the data on the subspace of dimension q that is closest to the data according to the sums of the squared Euclidean distances*.

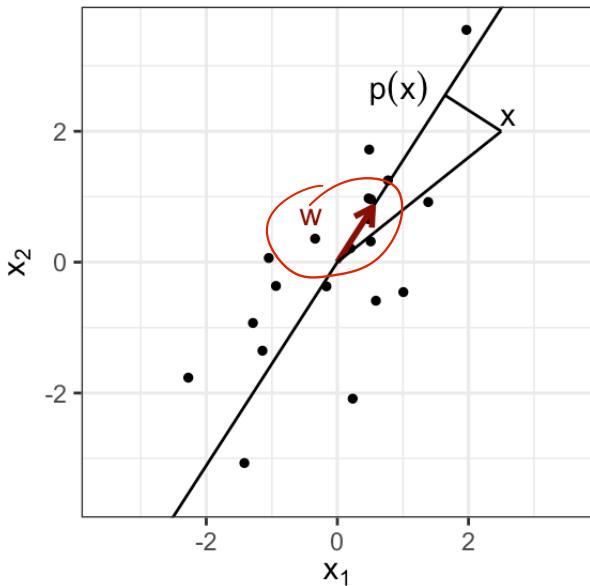
Principal Component Analysis is not only the mother of all data reduction techniques but also still widely used. PCA enjoys several noticeable statistical properties which we will now look at.

5.5.1 A minimal PCA: From 2D to 1D

To get an intuition of PCA, let us consider reducing a 2-dimensional dataset into a single dimension. This application has no visualization purposes but could help defining a linear combination of two variables into an aggregated score. For example, one can want to summarize the weight and horsepower of the cars into a single score.

5.5.1.1 Definition of the first principal component

Geometrically, we search for a line lying as close as possible to the data, in the sense of least squared Euclidean distances.



We will assume all variables to be centered and admit the line passes therefore through the origin. Let us denote \mathbf{w} a direction vector of the line of length 1.

The closest point of the observation vector \mathbf{x}_i to the line is its orthogonal projection $p_{\perp}(\mathbf{x}_i)$ which is equal to the scalar product of the direction vector and the observation vector, times the direction vector:

$$p_{\perp}(\mathbf{x}_i) = (\mathbf{w}^T \mathbf{x}_i) \mathbf{w}$$

Hence, we look for \mathbf{w} such that:

$$\begin{aligned} & \min_{\mathbf{w}} \sum_{i=1}^n \|\mathbf{x}_i - (\mathbf{w}^T \mathbf{x}_i) \mathbf{w}\|^2 \\ & \text{subject to } \|\mathbf{w}\| = 1 \end{aligned} \tag{5.2}$$

There is typically a unique solution to this optimization problem (up to a sign). This direction vector \mathbf{w} is called the *first principal component (PC1)* of the data.

5.5.1.2 PC1 maximizes the variance of the projected data

We defined PC1 with a minimization problem (Equation (5.2)). One can also see it as a maximization problem. To this end, consider the orthogonal triangle defined by an observation vector \mathbf{x}_i , its projection $p_{\perp}(\mathbf{x}_i)$, and the origin. Pythagoras' theorem implies that:

$$\|\mathbf{x}_i\|^2 = \|p_{\perp}(\mathbf{x}_i)\|^2 + \|\mathbf{x}_i - p_{\perp}(\mathbf{x}_i)\|^2$$

Over the entire dataset, the sum of the $\|\mathbf{x}_i\|^2$ is constant independently of the choice of \mathbf{w} . Therefore minimization problem Equation (5.2) is equivalent to:

$$\begin{aligned} & \max_{\mathbf{w}} \sum_{i=1}^n \|p_{\perp}(\mathbf{x}_i)\|^2 \\ & \text{subject to } \|\mathbf{w}\| = 1 \end{aligned} \tag{5.3}$$

As we have centered the data, the origin is the mean of the data. Hence the sum of squared norms of the observation vectors is n times their total variance. By linearity, the origin is also the mean of the projected data and thus:

$$\sum_i \|\mathbf{p}_{\top}(\mathbf{x}_i)\|^2 = n \operatorname{Var}(\mathbf{p}_{\top}(\mathbf{X}))$$

Hence, one can equivalently consider that PC1 maximizes the variance of the projected data.

Result PC1 maximizes the variance of the projected data.

The proportion of variance captured by PC1 is defined as the ratio of the variance of the projected data over the total variance of the data. It is a proportion, hence lies between 0 and 1. The higher it is, the smaller the sum of squared distances, the closer the line is to the data. The proportion of variance hence quantifies how good our dimension reduction is.

5.5.2 PCA in higher dimensions

In the general case, with p variables and n observations, one searches for the q -dimensional plane that is closest to the data in terms of sums of squared Euclidean distances. This is also the q -dimensional plane that maximizes the variance of the projected data.

An important property relates principal components to the eigendecomposition of the covariance matrix.

The covariance matrix is $\frac{1}{n} \mathbf{X}^T \mathbf{X}$. It is a symmetric positive matrix. We denote $\mathbf{w}_1, \dots, \mathbf{w}_j, \dots$ its eigenvectors ordered by decreasing eigenvalues $\lambda_1 > \dots > \lambda_j > \dots$

Result. The PCA q -dimensional plane, i.e., the q -dimensional plane that is closest to the data in terms of sums of squared Euclidean distances, is the plane spanned by the first q eigenvectors of the covariance matrix.

Result. The proportion of variance explained by the PCA q -dimensional plane equals to the sum of the q first eigenvalues of the covariance matrix.

See Bishop's book for proofs.

These results have several implications:

- The PCA planes are nested: the PCA 2D-plane contains PC1, the PCA 3D-plane contains the PCA 2D-plane, etc.
- We call second principal component (PC2) the second eigenvector of the covariance matrix, etc.
- The principal components are linearly uncorrelated. This is because the eigenvectors of a positive matrix are orthogonal to each other. If $n > p$ (more observations than variables) the PCs form an orthonormal basis.

A 3D interactive illustration of PCA is available at [<http://www.joyofdata.de/public/pca-3d/>].

5.5.3 PCA in R

PCA can be easily performed in R by using the built-in function `prcomp()`.

In most applications, scaling the data beforehand is important. Because PCA is based on minimizing squared Euclidean distances, scaling allows to not give too much importance to variables living on larger scales than the other ones. Be careful: for legacy reasons, the default of `prcomp` is to not scale the variables.

In the following examples, we perform PCA on our `mat` dataset. We set the two arguments `scale` and `center` to `TRUE` so that the data is first centered and scaled before performing PCA.

```
pca_res <- prcomp(mat, center = TRUE, scale. = TRUE)
names(pca_res)

## [1] "sdev"      "rotation"   "center"    "scale"     "x"
```

The output can be stored in `pca_res`, which contains information about the center point (`center`), scaling (`scale`), standard deviation (`sdev`) of each principal component, as well as the values of each sample in terms of the principal components (`x`) and the relationship between the initial variables and the principal components (`rotation`).

An overview of the PCA result can be obtained with the function `summary()`, which describes the standard deviation, proportion of variance and cumulative proportion of variance of each of the resulting principal components (`PC1`, `PC2`, ...). We remark that the cumulative proportion is always equal to one for the last principal component.

```
pca_sum <- summary(pca_res)
pca_sum

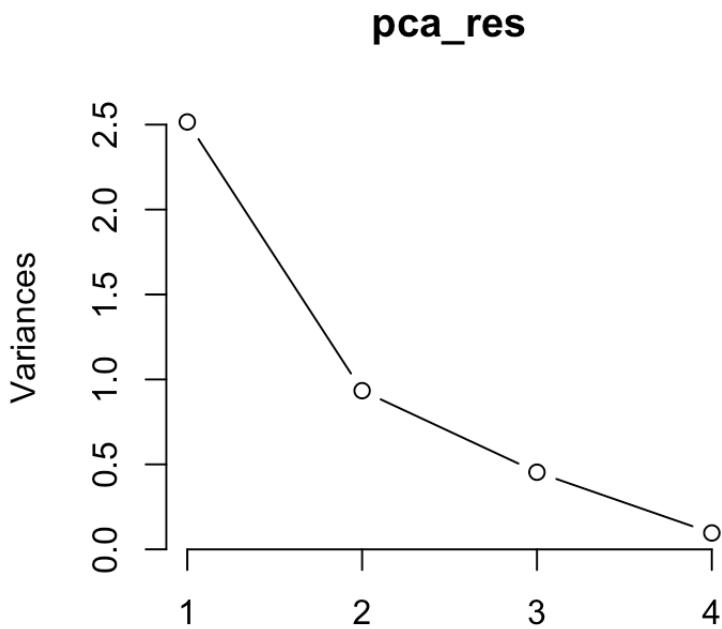
## Importance of components:
##                 PC1    PC2    PC3    PC4
## Standard deviation 1.586 0.9664 0.6737 0.3105
## Proportion of Variance 0.629 0.2335 0.1135 0.0241
## Cumulative Proportion 0.629 0.8624 0.9759 1.0000
```

In this example, the first principal component explains 62.9% of the total variance and the second one 23.35%. So, just `PC1` and `PC2` can explain approximately 86.24% of the total variance.

5.5.4 Plotting PCA results in R

Plotting the results of PCA is particularly important. The so-called **scree plot** is a good first step for visualizing the PCA output, since it may be used as a **diagnostic tool** to **check whether the PCA worked** well on the selected dataset or not.

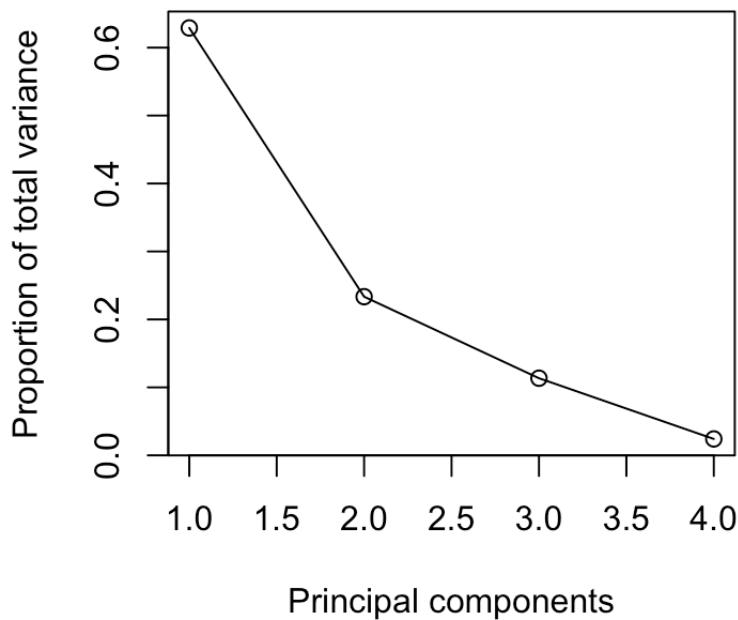
```
plot(pca_res, type='l')
```



The scree plot shows the variance in each projected direction. The y-axis contains the eigenvalues, which essentially stand for the amount of variation. We can use a scree plot to select the principal components to keep. If the scree plot has an 'elbow' shape, it can be used to decide how many principal components to use for further analysis. For example, we may achieve dimensionality reduction by transforming the original four dimensional data (first four variables of `mtcars`) to a two-dimensional space by using the first two principal components.

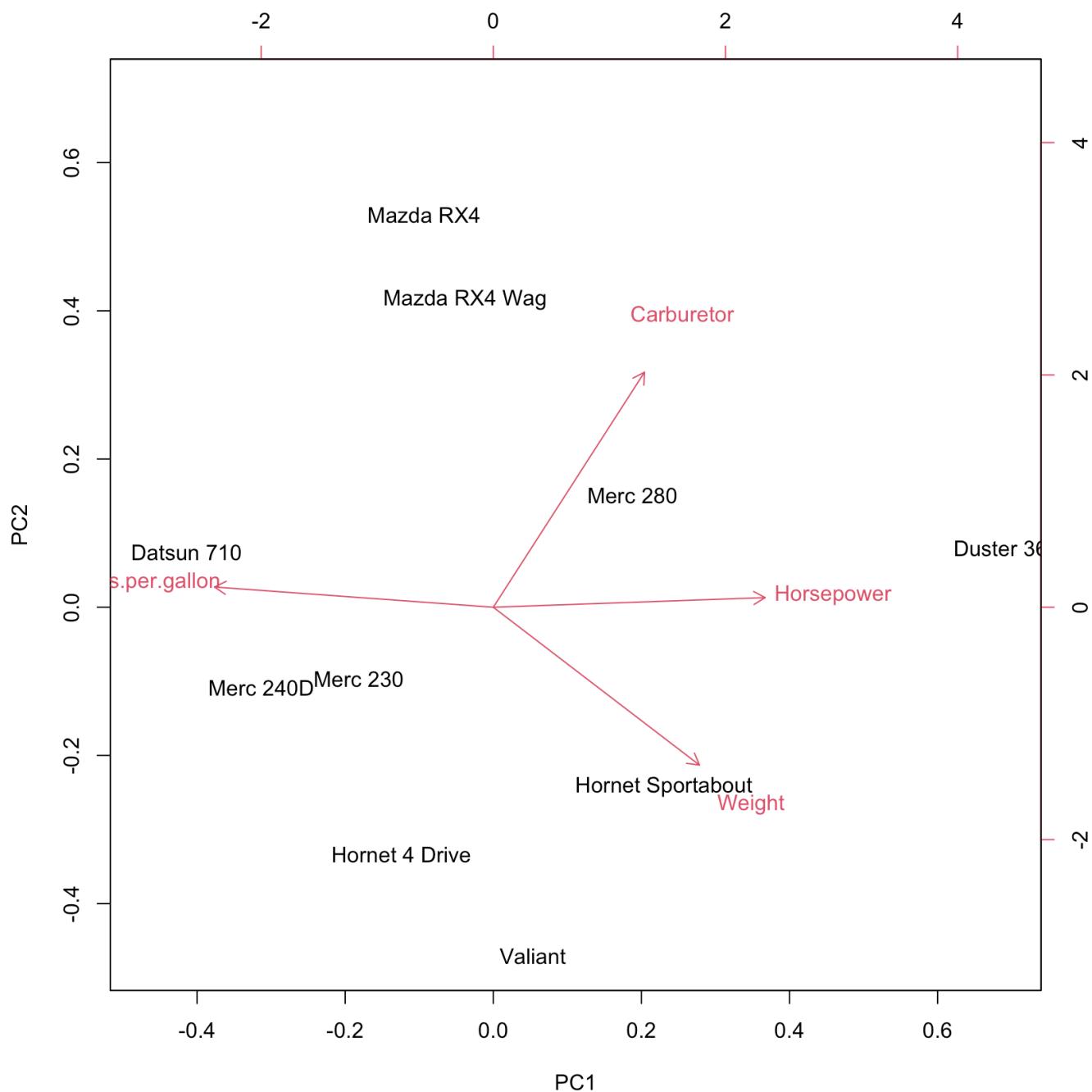
A variant of the scree plot can be considered by plotting the proportion of the total variance for every principal component.

```
plot(pca_sum$importance[2,], type='l',
      xlab='Principal components', ylab="Proportion of total variance")
points(pca_sum$importance[2,])
```



The **biplot** shows the projection of the data on the first two principal components. It includes both the position of each sample in terms of PC1 and PC2 and also shows how the initial variables map onto this. The correlation between variables can be derived from the angle between the vectors. Here, a small angle is related to a high correlation.

```
biplot(pca_res)
```



We can access the projection of the original data on the principal components by using the function `predict` as follows:

```
predict(pca_res)
```

```

##                  PC1      PC2      PC3
## Mazda RX4     -0.4683249 1.6182751 0.17530341
## Mazda RX4 Wag -0.1906974 1.2697111 -0.25876251
## Datsun 710    -2.0776213 0.2274096 1.35795888
## Hornet 4 Drive -0.6221540 -1.0190504 0.13515892
## Hornet Sportabout 1.1556918 -0.7385145 0.28778404
## Valiant        0.2672733 -1.4381382 0.01578234
## Duster 360    3.4765710 0.2447894 0.48646675
## Merc 240D     -1.5716800 -0.3318280 -0.91025054
## Merc 230      -0.9115278 -0.2945376 -0.37473618
## Merc 280      0.9424693 0.4618835 -0.91470512
##                  PC4
## Mazda RX4     0.17047731
## Mazda RX4 Wag 0.08957852
## Datsun 710    0.01780133
## Hornet 4 Drive -0.12922895
## Hornet Sportabout -0.34788481
## Valiant       0.69558672
## Duster 360    -0.16580224
## Merc 240D     -0.26269735
## Merc 230      -0.26813224
## Merc 280      0.20030170

```

5.5.5 PCA summary

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of possibly correlated variables into a set of linearly uncorrelated variables, which are denoted as principal components.

Each principal component explains a fraction of the total variation in the dataset. The first principal component has the largest possible variance. Respectively, the second principal component has the second-largest possible variance.

In this manner, PCA aims to reduce the number of variables, while preserving as much information from the original dataset as possible. High-dimensional data is often visualized by plotting the first two principal components after performing PCA.

5.5.6 Nonlinear dimension reduction

One limitation of PCA is that it is restricted to linear transformation of the data. What if the data lies closer to a parabola rather than a straight line? There are many non-linear alternatives to PCA including Independent Component Analysis, kernel PCA, t-SNE, UMAP. Details of these techniques are beyond the scope of this lecture. However, as long as one uses these techniques as visualization and exploratory tools rather than for making any claim on the data, a profound understanding of their theory is not necessary.

We illustrate here with one example of a PCA and a UMAP representation of same single-cell gene expression matrix of mouse. The input matrix has 15,604 rows (single cells) and 1,951 columns (genes) and comes for the Tabula muris project.⁷

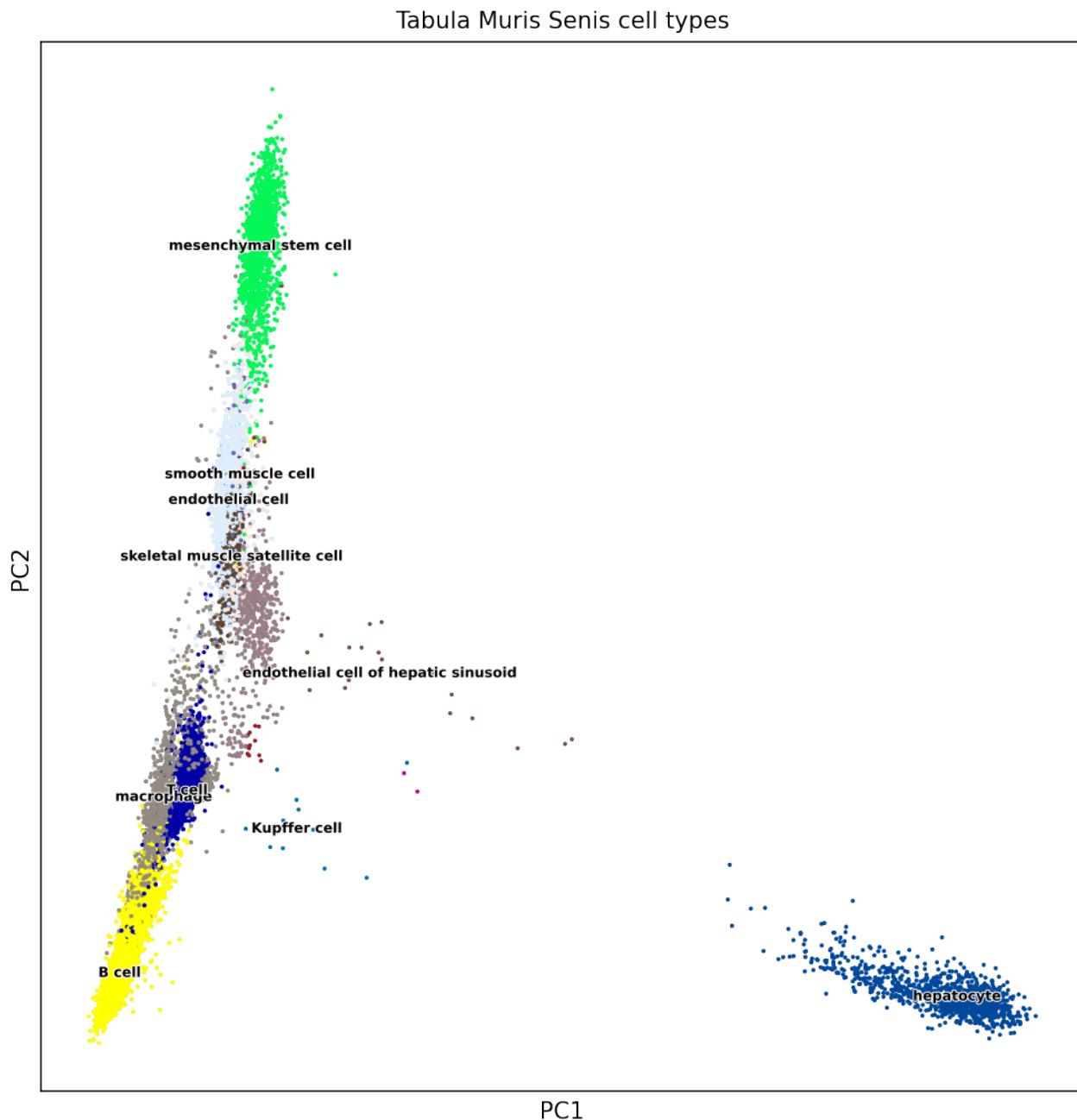


Figure 5.7: PCA on mouse single-cell transcriptome data. Source: Laura Martens, TUM

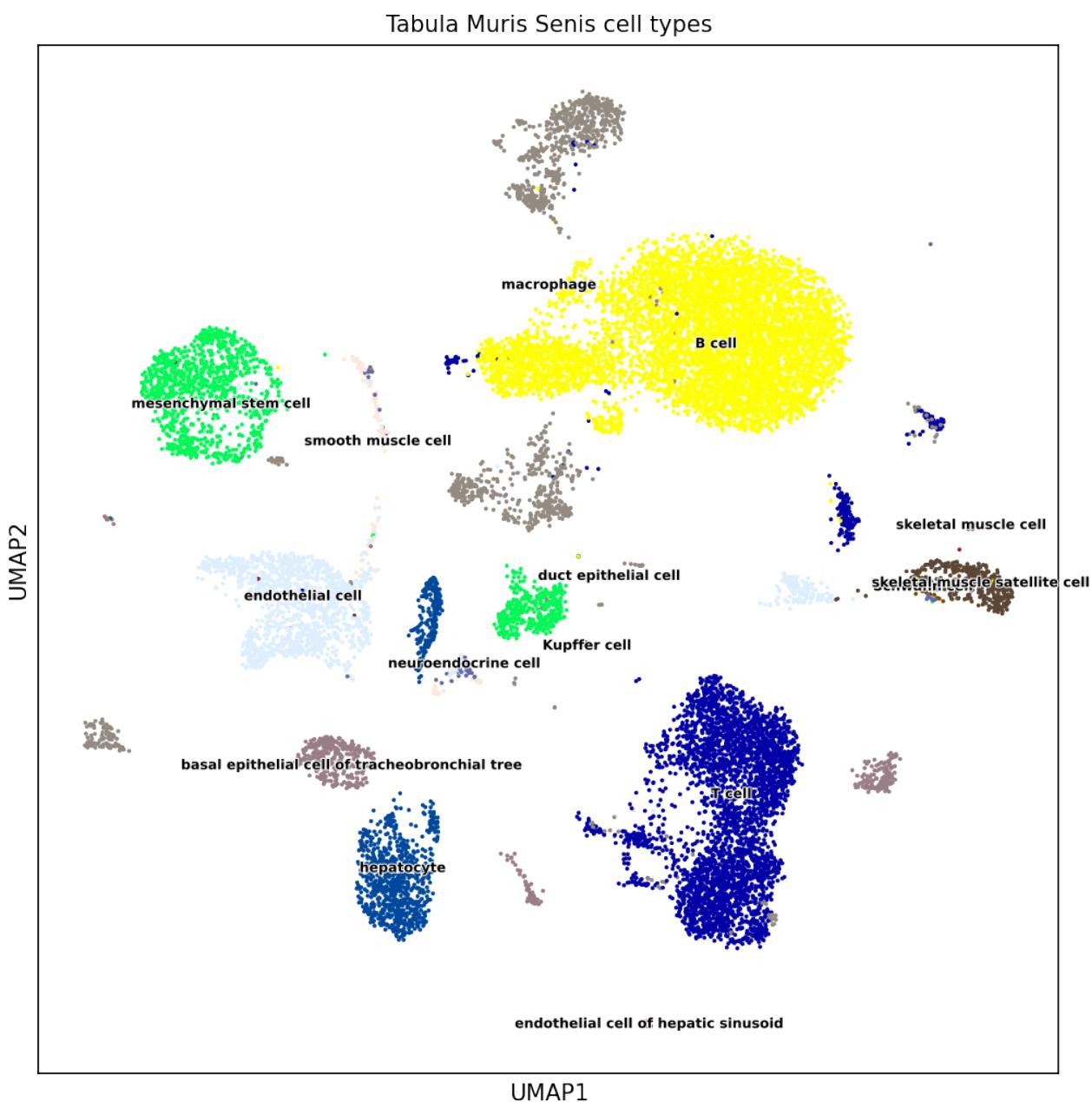


Figure 5.8: UMAP (non-linear dimension reduction) on mouse single-cell transcriptome data. Source: Laura Martens, TUM

5.6 Discussion

- Clustering and dimension reduction techniques belong to the family of unsupervised learning methods. Unlike supervised learning methods (e.g. regression, classification) unsupervised learning methods shall discover patterns in the data without being guided by some ground truth.
- There is no “right” clustering, or “right” subspace in real-life datasets
- Clustering and dimension reduction techniques are exploratory tools meant to help deriving some hypotheses
- These hypotheses are then best tested on independent data

5.7 Summary

By now, you should be able to:

- plot data matrices as pretty heatmaps
- understand the effects of centering and scaling
- describe and apply k-means clustering
- describe and apply agglomerative hierarchical clustering
- PCA:
 - definition
 - property: maximize variance
 - property: uncorrelated components
 - compute and plot a PCA representation in R
 - compute the proportion of explained variance in R

5.8 Resources

G. James, D. Witten, T. Hastie and R. Tibshirani. An Introduction to Statistical Learning with Applications in R. Book and R code available at: <https://www.statlearning.com/>

Advanced (PCA proofs): C. Bishop, Pattern Recognition and Machine Learning. <https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/>

Exceptionally, on these topics, the Introduction to Data Science book by R. Irizarry is not great.

5. <https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding> ↪
6. https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html ↪
7. <https://tabula-muris.ds.czbiohub.org/> ↪

Chapter 6 Graphically supported hypotheses

In the last 2 chapters we presented the different types of plots. We now consider the task of structuring and presenting an entire analysis. We want to be able to provide novel insights from a dataset. Guided by the scientific method, a major aspect will be to give strong support for testable hypotheses. This is important in basic research to suggest experimental validations (does this gene affects growth?), but also for decision makers in public health (does smoking cause cancer?), public governance (would investing in education benefit economy?) or in the industry (would modifying a process increases productivity, modifying a product increases sales, etc.?).

This chapter covers:

- the concepts of descriptive plots for data exploration, and associative plots to support testable hypotheses
- how correlation and causation relate to each other
- good practices to organize reports and presentations and to have visuals to convey one's main message

6.1 Descriptive vs. associative plots

A typical data analysis starts with an exploration of the data. These are illustrated with **descriptive plots** showing how the **data is distributed**. This allows the audience and oneself to get familiar with the dataset in an unbiased fashion (e.g. number of participants in study, age and sex distributions). Afterwards, one typically highlights some interesting **relationships** between variables, often in order to suggest a testable, causal relationship. We call the latter types of plots **associative plots**.

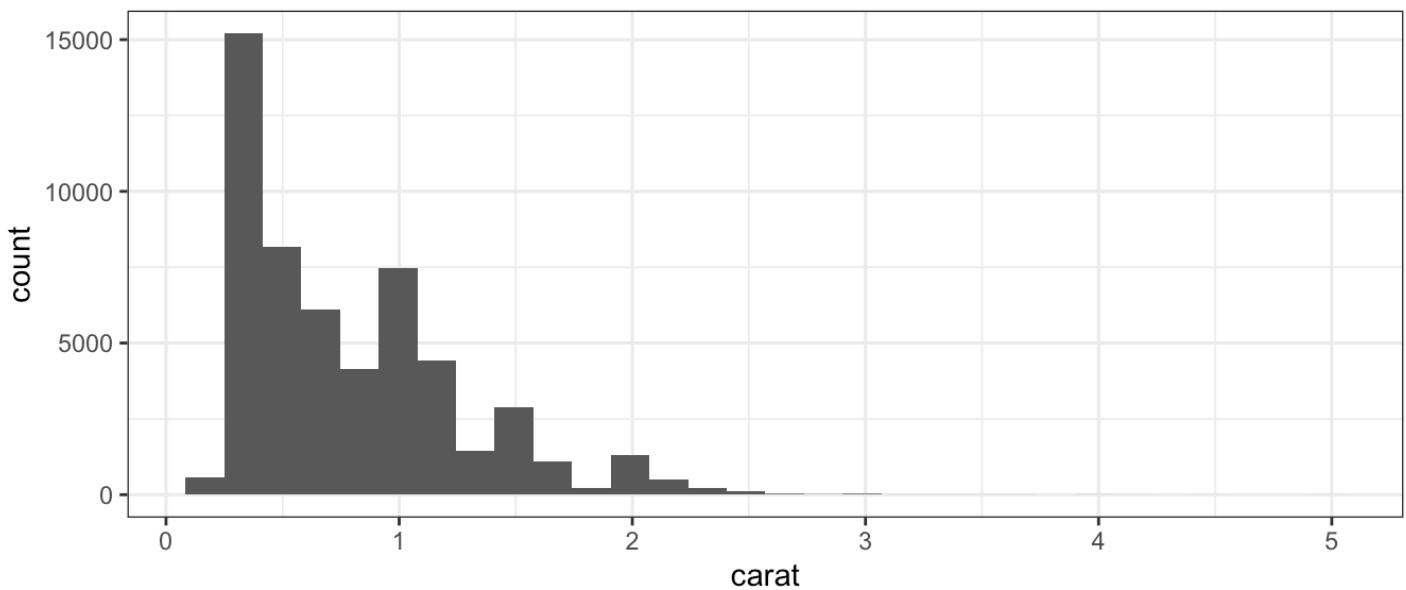
6.1.1 Descriptive plots

Descriptive plots allow exploring **how a variable or a set of variables distributes**. For a **univariate data** these are **histograms, single boxplots or violin plots**. For **multivariate** data these are clustered **heatmaps, PCA projections**, etc.

Descriptive plots correspond to **descriptions of the distribution $p(\mathbf{X})$** , where X is the variable we want to explore. In machine learning terms, they correspond to methods from unsupervised learning e.g. k-means clustering, hierarchical clustering, PCA.

For example, we can consider the `diamonds` dataset which contains the prices and other attributes of several diamonds. To first visualize the distribution of the weights (`carat`) of the diamonds, we can build a histogram:

```
ggplot(diamonds, aes(carat)) + geom_histogram() + mytheme
```



6.1.2 Associative plots

Associative plots show how a variable depends on another variable.

For such plots, one typically uses the y-axis for the response variable (e.g. survival rate, skin cancer occurrence) and the x-axis for the explanatory variables (e.g drug dose, sun exposure). Suitable plots are side-by-side boxplots, scatter plots, etc.

Associative plots are graphical representation of the conditional distributions $p(y|x)$ (pronounced “p of y given x”), where y is the response and x is the explanatory variable(s). When the conditional distribution $p(y|x)$ actually depends on x , we say that x and y are dependent. For instance a scatterplot can show the trend that the response y increases in average as x increases. A boxplot can show that characteristic values of the distribution of y (quartiles, median) depends on the value of the category x , etc.

In machine learning terms, associative plots correspond to methods from supervised learning (e.g. regression, classification). For instance, one can build a predictor that predicts the risk of skin cancer given the age and sun exposure of a person.

Based on the same `diamonds` dataset, we can now construct an associative plot to graphically support the hypothesis that the price of a diamond increases with increasing weight (carat):

```
ggplot(diamonds, aes(carat, price)) +
  geom_point(alpha = 0.05) + # alpha: point transparency
  stat_smooth() + mytheme
```

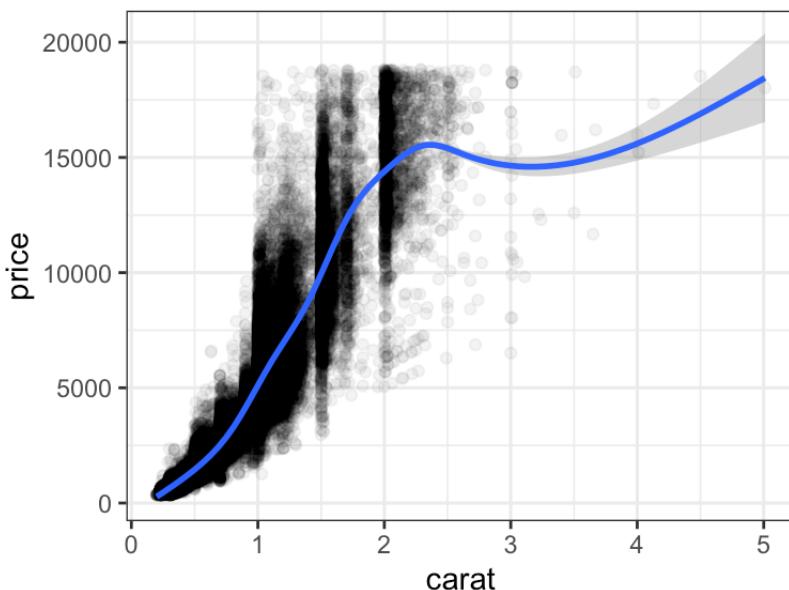


Figure 6.1: Price increases nearly linearly with carat

6.1.3 Correctly using descriptive and demonstrative plots

A typical mistake is to use a descriptive plot to formulate a claim. For instance, a statement such as “My experiments are reproducible because the replicate samples group together on the projection of the first PCA 2D-plane” is wrong. PCA is a visualization designed to capture the joint distribution of all data and not to assess reproducibility between given pair of samples. To make a claim, we use an associative plot between a response variable on the y-axis against a dependent variable on the x-axis. In this example, we can create a boxplot that shows the distribution of the correlation between pairs of samples (y-axis) for two categories (x-axis): the group of replicate samples and the group of sample pairs from different experiments.

6.2 Correlation and causation

One goal of data analysis is to provide a hypothesis about the underlying causal mechanisms.

Causal conclusions can be established in specific cases where the data has been gathered in a controlled fashion. This includes experimental perturbations where a single variable is changed and every other condition is kept the same. For instance, one can vary the dose of a drug given to a bacteria and measure its growth, in otherwise identical growth conditions. Randomized controlled trials constitute another example where a treatment and its placebo are administered to a cohort randomly, therefore preventing by design associations with other factors.

However, we most of the time have access to uncontrolled (one also say observational) data. With observational data, we can assess statistical dependencies, or associations, between variables such as correlations, enrichments, etc. However, these do not necessarily imply a causal link.

In the following sections, we go through the elementary situations of non-causal associations and how to address these issues through data visualization and appropriate wordings of the claims. These are:

- The association is not statistically supported
- The causal relationship is reverse

- The association is induced by a third variable

Associations between variables that are not due to a causal relationship are sometimes called “spurious correlations”. We prefer however to not use the term spurious correlation because genuine statistical dependencies that are non-causal are widespread. It is not the correlations that are spurious, but their causal interpretation.

6.2.1 The association is not statistically supported

Often, the observed association between variables arose by chance. Would the data collection be repeated, or performed for a longer period of time, the association would not show up any longer. There are two main reasons for this:

1. The association is driven by few data points. Graphically showing all data points including outliers can help (See Anscombe's quartet⁸). Hypothesis testing can be used to assess this possibility (See next Chapters).
2. The dataset includes so many variables that the chance to have one pair of variables associating is high. This is generally called data dredging, data fishing, data snooping, or cherry picking. An example of data dredging would be if we look through many results produced by a random process and pick the one that shows a relationship that supports a theory we want to defend. An example is given in Figure 6.2.

In the statistical literature, this problem is called “multiple testing”. It will be treated in a later chapter.

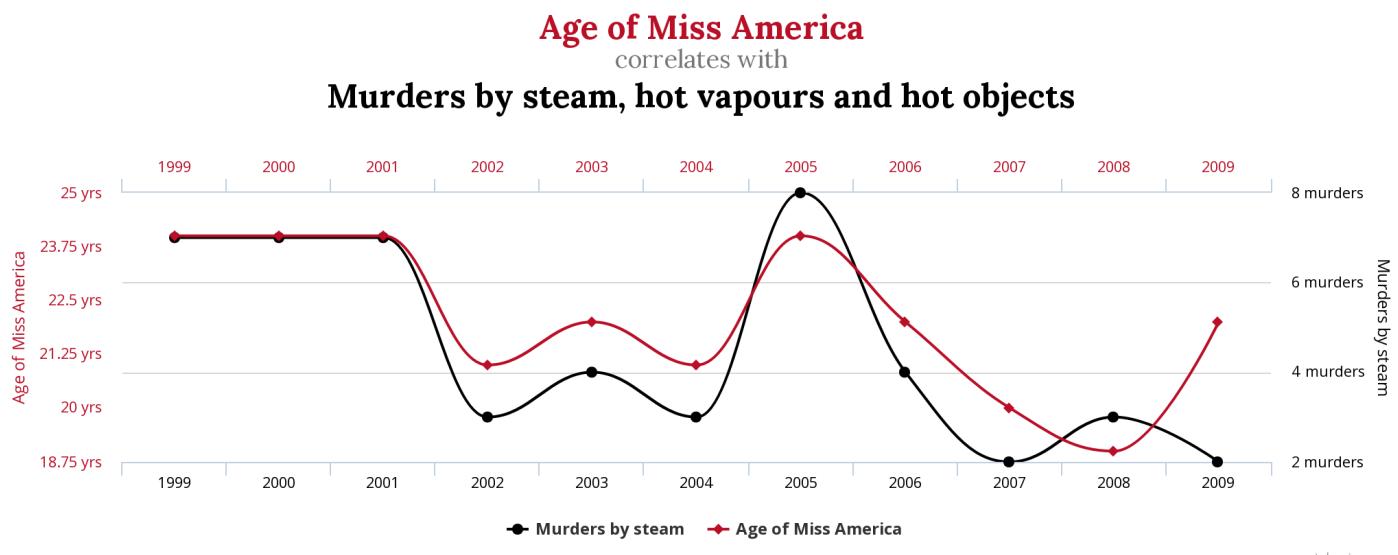


Figure 6.2: Example of a correlation discovered in a dataset with a very large set of variables. Source: <http://www.tylervigen.com/spurious-correlations>.

6.2.2 Reversing cause and effect

We now assume the association did not arise by chance. Unlike causal relationships, statistical dependencies are symmetric (if A correlates with B, then B correlates with A). Hence, a typical mistake are claims where cause and effects are reversed.

A form of this claim actually made it into an op-ed in the New York Times titled Parental Involvement Is Overrated⁹. Consider this quote from the article:

When we examined whether regular help with homework had a positive impact on children's academic performance, we were quite startled by what we found. Regardless of a family's social class, racial or ethnic background, or a child's grade level, consistent homework help almost never improved test scores or grades... Even more surprising to us was that when parents regularly helped with homework, kids usually performed worse.

In fact, a very likely possibility is that the children needing regular parental help receive this help because they do not perform well in school.

Further examples of reversing cause and effect include the following statements:

- People with healthier diet have higher blood pressure.
- Individuals in a low social status have a higher risk of schizophrenia.
- The number of fire engines on a fire associates with higher damages.
- Entering an intensive care unit increases your chances of dying.

There is **no firm way to decide the direction of causality from a mere association**. It is therefore important to consider and discuss both possibilities when interpreting a correlation.

6.2.3 The association is induced by a third variable

Another general issue is when an association is due to a third variable. There are several scenarios.

Given two variables x and y , we will now consider three basic configurations where a third variable z could be causally related to them and how this affects associations between the variables x and y (Figure 6.3). We show causal relationships using causal diagrams where variables are nodes and causal links are depicted as directed arrows. We denote $x \perp y$ when variables x and y are statistically independent and $x \not\perp y$ when they are dependent. Also, $(x \perp y)|z$ means that x and y are independent given z .

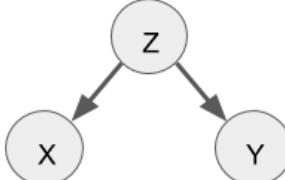
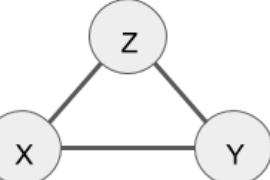
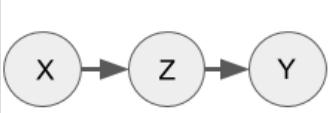
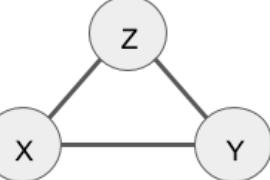
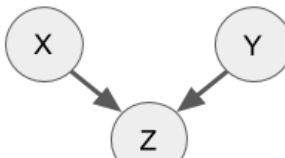
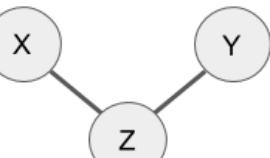
Situation	Causal diagram	Pairwise associations	Conditional independence	Diagnostic plot
Common cause			$(X \perp Y) Z$	Y vs. X faceted by Z
Indirect cause			$(X \perp Y) Z$	Y vs. X faceted by Z
Common consequence			$(X \not\perp Y) Z$	Y vs. X

Figure 6.3: Elementary causal diagrams involving two variables of interest X and Y and a third variable Z.

6.2.3.1 Common cause

The statistical dependency between two variables can be due to a common cause. For instance, it had been reported that among students, smokers had higher university degrees than non-smokers. This association was due to student age as a common cause. With time, students are more likely to have started smoking and to have advanced further in their studies.

Plots can help to identify such situations. One can stratify by age and show that this positive association vanishes. This rules out a causal association because a causal association is expected to hold for all ages (or at least at some age).

To understand the situation we will now generate artificially a dataset where two random variables x and y share a common cause z .

We consider the following process:

1. We toss a fair coin twice and record z the number of heads obtained. z is thus equal to 0, 1, or 2.
2. We draw a value x randomly according to a Gaussian distribution with mean equals to z and standard deviation 1. Hence, x will be a random real number that tend to have higher values with higher values of z .
3. We draw a value y randomly according to a Gaussian distribution with mean equals to $2 \times z$ and standard deviation 1. Hence y will be a random real number that tend to have higher values with higher values of z .

We then repeat this process $n = 1,000$ times.

In R this can done with the following code:

```

#####
## common cause
## x <- z -> y
#####

set.seed(0)

n <- 1000 # number of draws

# z: n draws of the binomial
# with 2 trials and prob=0.5
z <- rbinom(n, 2, prob=0.5)

# x: Gaussian with mean z
x <- rnorm(n, mean=z, sd=1)

# y: Gaussian with mean 2*z
y <- rnorm(n, mean=2*z, sd=1)

# gather into a data table
# we make z a factor for convenience with plotting functions
dt <- data.table(
  x,y,
  z = factor(paste0("z=",z))
)

```

We check the simulations worked. First, z distributes as expected with about 25% draws with 0 heads, 50% with one head, and 25% with 2 heads:

```
table(z)
```

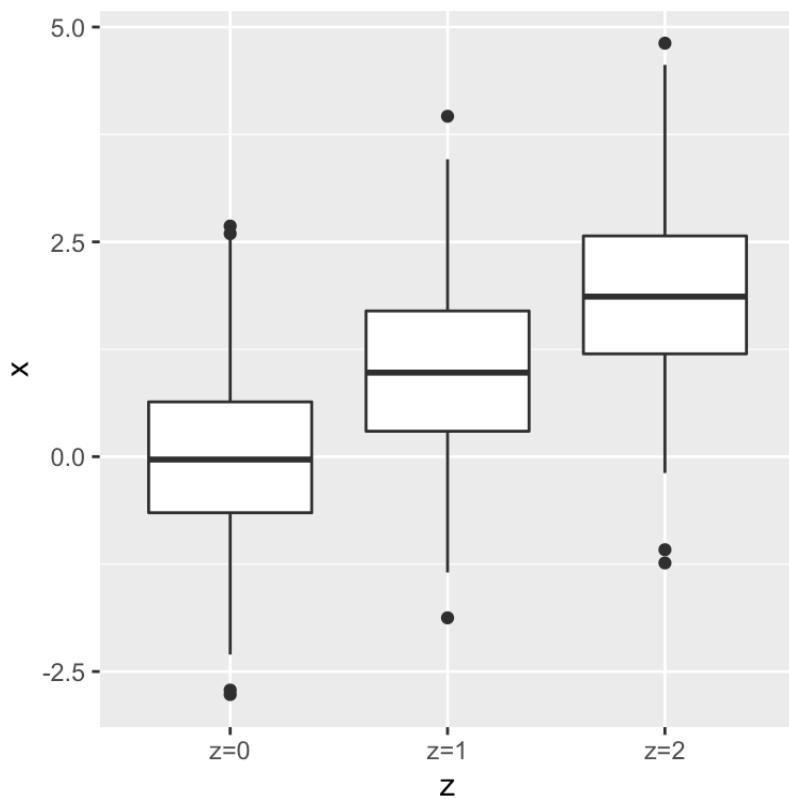
```

## z
##   0   1   2
## 244 506 250

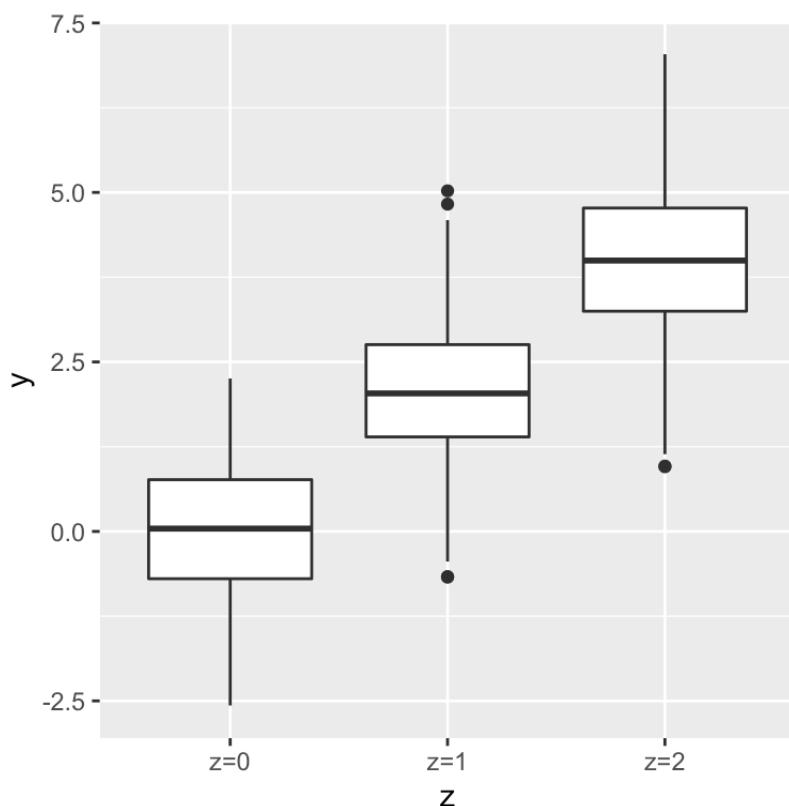
```

Second, x and y depend on z :

```
ggplot(dt, aes(x=z, y=x)) + geom_boxplot()
```

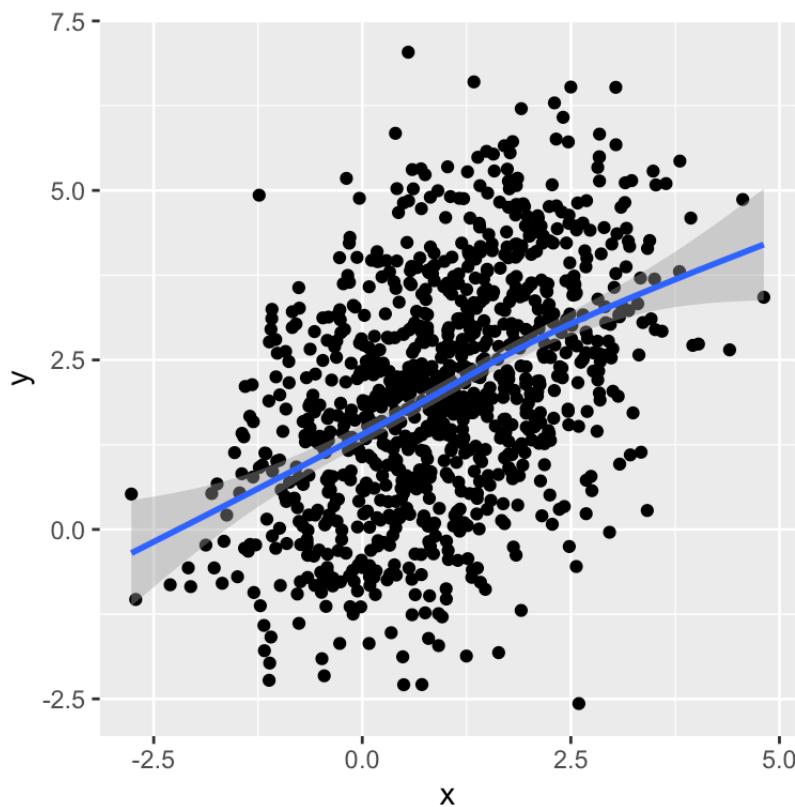


```
ggplot(dt, aes(x=z, y=y)) + geom_boxplot()
```



As a consequence, x and y correlate with each other:

```
# You can pick different formulae in geom_smooth
ggplot(dt, aes(x=x, y=y)) + geom_point() + geom_smooth()
```

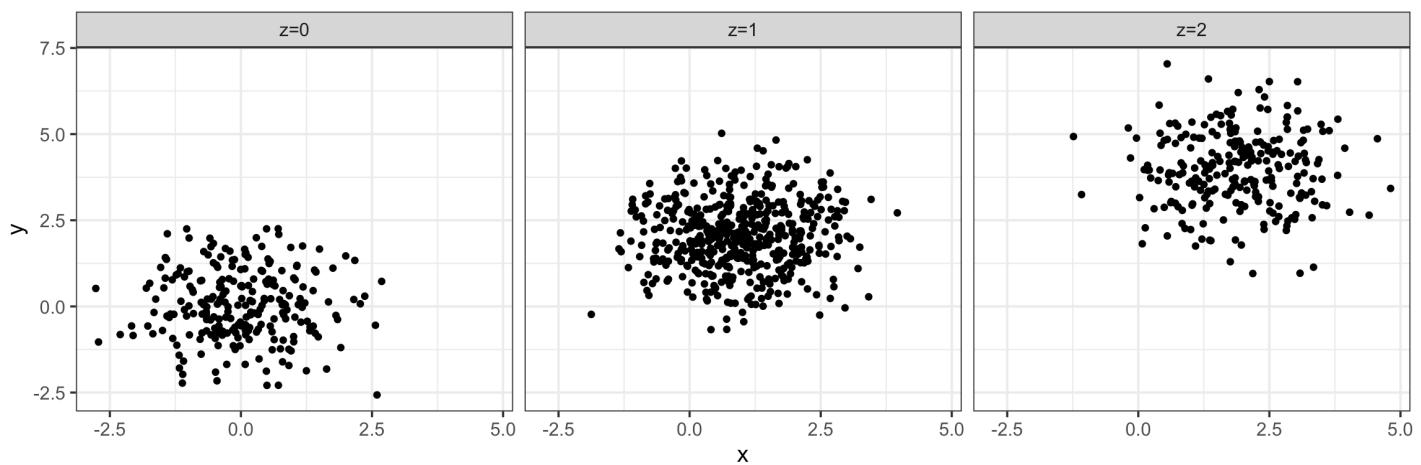


```
cor(x,y)
```

```
## [1] 0.4535741
```

This correlation between x and y is due to the common cause z . To show evidence for this issue, one can plot stratified by the values of z . In each group, there is no association between x and y .

```
ggplot(dt, aes(x=x, y=y)) + geom_point() + facet_wrap(~z) + theme_bw(base_size = 14)
```



Moreover, one can compute the correlation also stratified by the value of z . We see that in each group the correlation is near 0.

```
dt[, .(correlation=cor(x,y)), by=z]
```

```
##      z   correlation
## 1: z=2 -0.0005907227
## 2: z=1  0.0457336976
## 3: z=0 -0.0107107309
```

In our simulation, conditioning on a third variable has ruled out direct correlation and, therefore, causation. Statistically speaking, we can state that the two variables x and y are correlated, but their correlation is explained by z . Conditioned on z , their correlation is basically zero. This rules out a causal relationship from one to the other. Correlation never proves causation. However, conditioning can rule out causation.

6.2.3.2 Indirect association

We now simulate the case of an indirect association. To this end we consider the following process:

1. We toss a coin once. We record $x = 1$ if it is a head, and $x = 0$ if it is a tail.
2. If $x = 0$, we toss the coin once. If $x = 1$, we toss the coins twice. We record z , the number of heads of this second step.
3. We draw a value y randomly according to a Gaussian distribution with mean equal to z and standard deviation 0.5. Hence, y will be a random real number that tends to have higher values with higher values of z .

The following R code simulates such data:

```

# first step: tossing a coin once
# record x=1 if head, x=0 if tail.
x <- rbinom(n, size=1, prob=0.5)

# second step
# if x=0, toss the coin once, if x=1, toss the coins twice
# record z the number of heads of this second step.
z <- rbinom(n, size=x+1, prob=0.5)

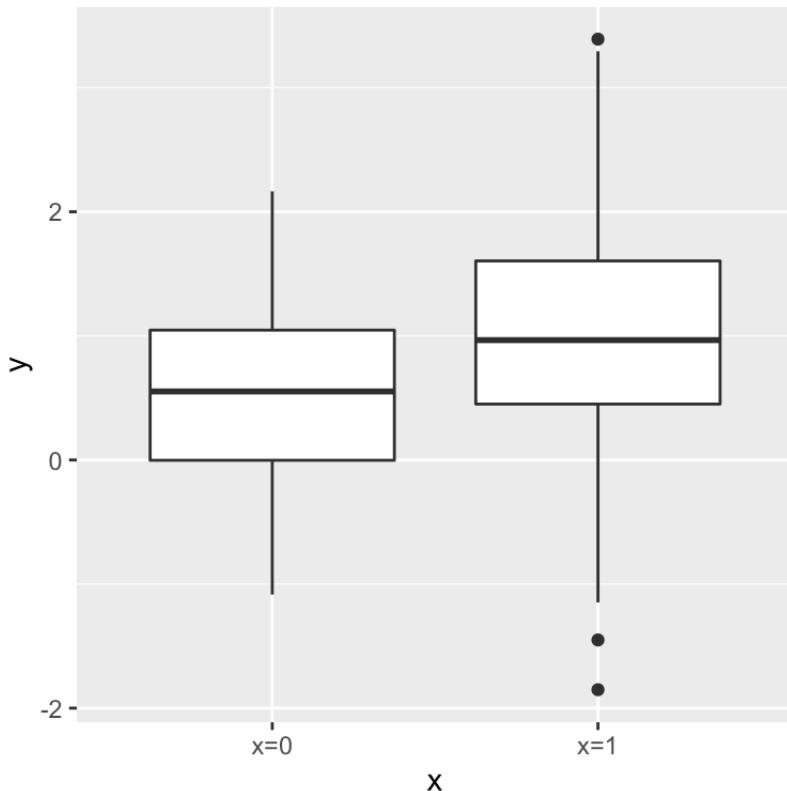
## y: Gaussian with mean=z
y <- rnorm(n, mean=z, sd=0.5)

dt <- data.table(
  x = factor(paste0("x=",x)),
  y,
  z = factor(paste0("z=",z))
)

```

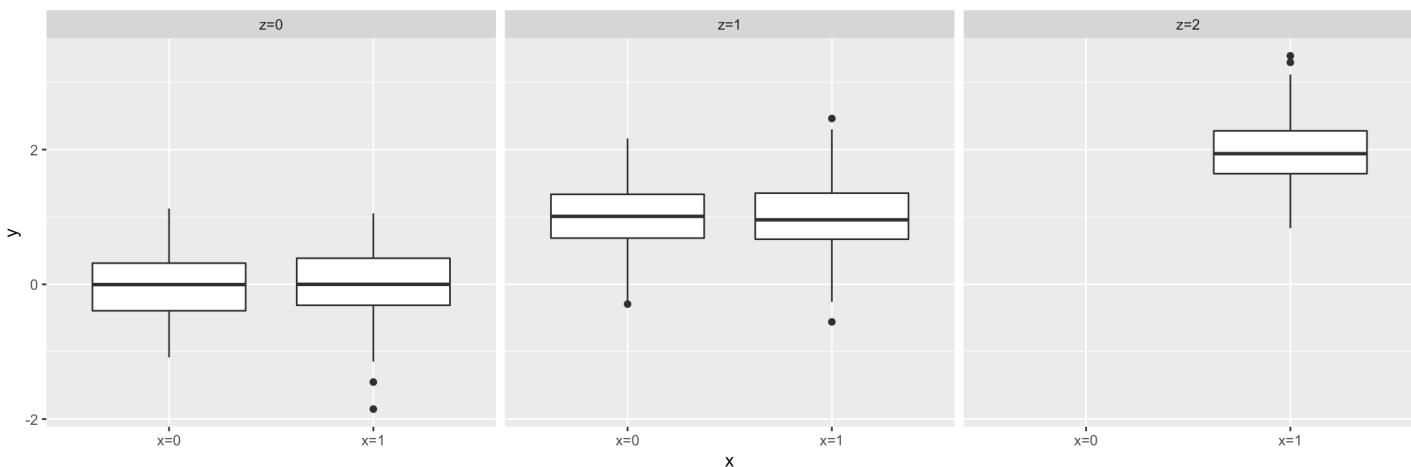
We see that y associates with x :

```
ggplot(dt, aes(x=x, y=y)) + geom_boxplot()
```



However this association vanishes when we condition on z :

```
## y does not associate with x given z
ggplot(dt, aes(x=x, y=y)) + geom_boxplot() + facet_wrap(~z)
```



Conditioning on the third variable has helped ruling out a direct causal relation between x and y . Note, however, that in this simulation there is an indirect causal relation between x and y . From an observational data point of view, the situation is very similar to the common cause situation. Interpretations from the application field are needed to suggest whether such data supports an indirect association or a common cause.

A concrete example of an indirect association is provided by Rafael Irizarry in his book “Introduction to Data Science” which we reproduce in the following.

For this, we consider the admission data from six U.C. Berkeley majors on 1973. The data shows at first sight that more men were being admitted than women: 44% men were admitted compared to 30% women. We can load the data and compute the percent of men and women that were accepted like this:

```
library(dslabs)
data(admissions)
admissions <- as.data.table(admissions)
admissions[, sum(admitted*applicants)/sum(applicants), by=gender]

##      gender      V1
## 1:    men 44.51951
## 2:  women 30.33351
```

Closer inspection shows a paradoxical result. Here are the percent admissions by major:

```
stats <- admissions[, sum(admitted), by=.(major, gender)] %>%
  dcast(...~gender, value.var='V1') %>% .[, women_minus_men := women - men]
head(stats)
```

```
##      major men women women_minus_men
## 1:     A   62    82          20
## 2:     B   63    68           5
## 3:     C   37    34          -3
## 4:     D   33    35           2
## 5:     E   28    24          -4
## 6:     F    6     7            1
```

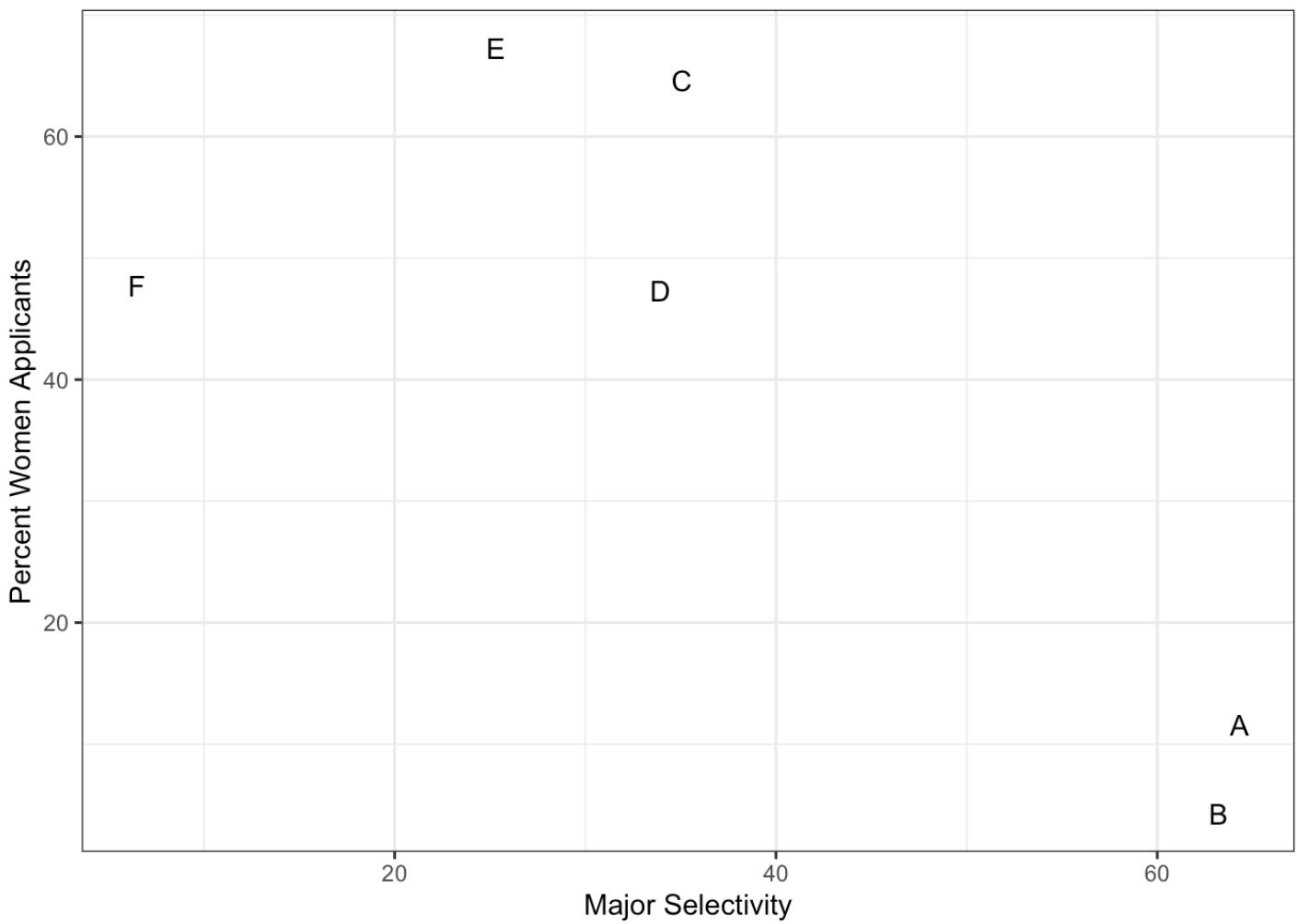
Four out of the six majors favor women. More importantly, all the differences are much smaller than the 14.2 difference that we see when examining the totals.

The paradox is that analyzing the totals suggests a dependence between admission and gender, but when the data is grouped by major, this dependence seems to disappear. What's going on? This actually can happen if an uncounted confounder is driving most of the variability.

We define three variables: X is 1 for men and 0 for women, Y is 1 for those admitted and 0 otherwise, and Z quantifies the selectivity of the major. A gender bias claim would be based on the fact that $\Pr(Y = 1|X = x)$ is higher for $x = 1$ than $x = 0$. However, Z is an important confounder to consider. Clearly Z is associated with Y , as the more selective a major, the lower $\Pr(Y = 1|Z = z)$. But is major selectivity Z associated with gender X ?

One way to see this is to plot the total percent admitted to a major versus the percent of women that made up the applicants:

```
admissions %>%
  .[, .(sum(admitted * applicants)/sum(applicants),
        sum(applicants * (gender=="women")) /sum(applicants) * 100), by=major] %>%
  ggplot(aes(V1, V2, label = major)) +
  geom_text() + labs(x='Major Selectivity', y='Percent Women Applicants') + mytheme
```



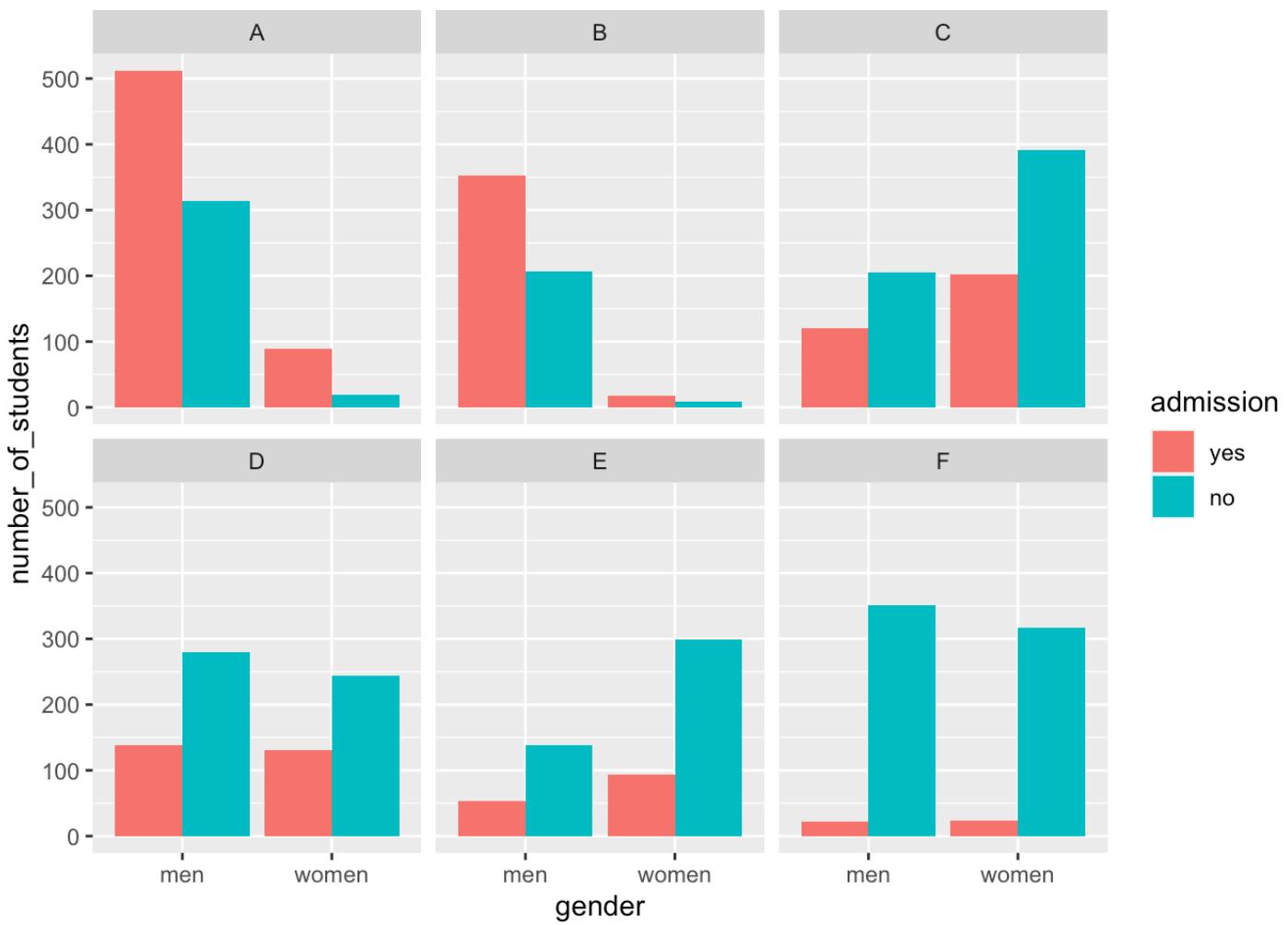
There seems to be association. The plot suggests that women were much more likely to apply to the two “hard” majors: gender and major’s selectivity are confounded. Compare, for example, major B and major E. Major E is much harder to enter than major B and over 60% of applicants to major E were women, while less than 30% of the applicants of major B were women.

The following plot shows the number of applicants that were admitted and those that were not by gender and major:

```

admissions_plot <- admissions[, `:=` (yes = round(admitted/100*applicants),
                                         no = round(applicants - admitted/100*applicants )),
                                         by=.(major, gender)] %>%
  melt(measure.vars=c('yes', 'no'), value.name='number_of_students',
       variable.name='admission')

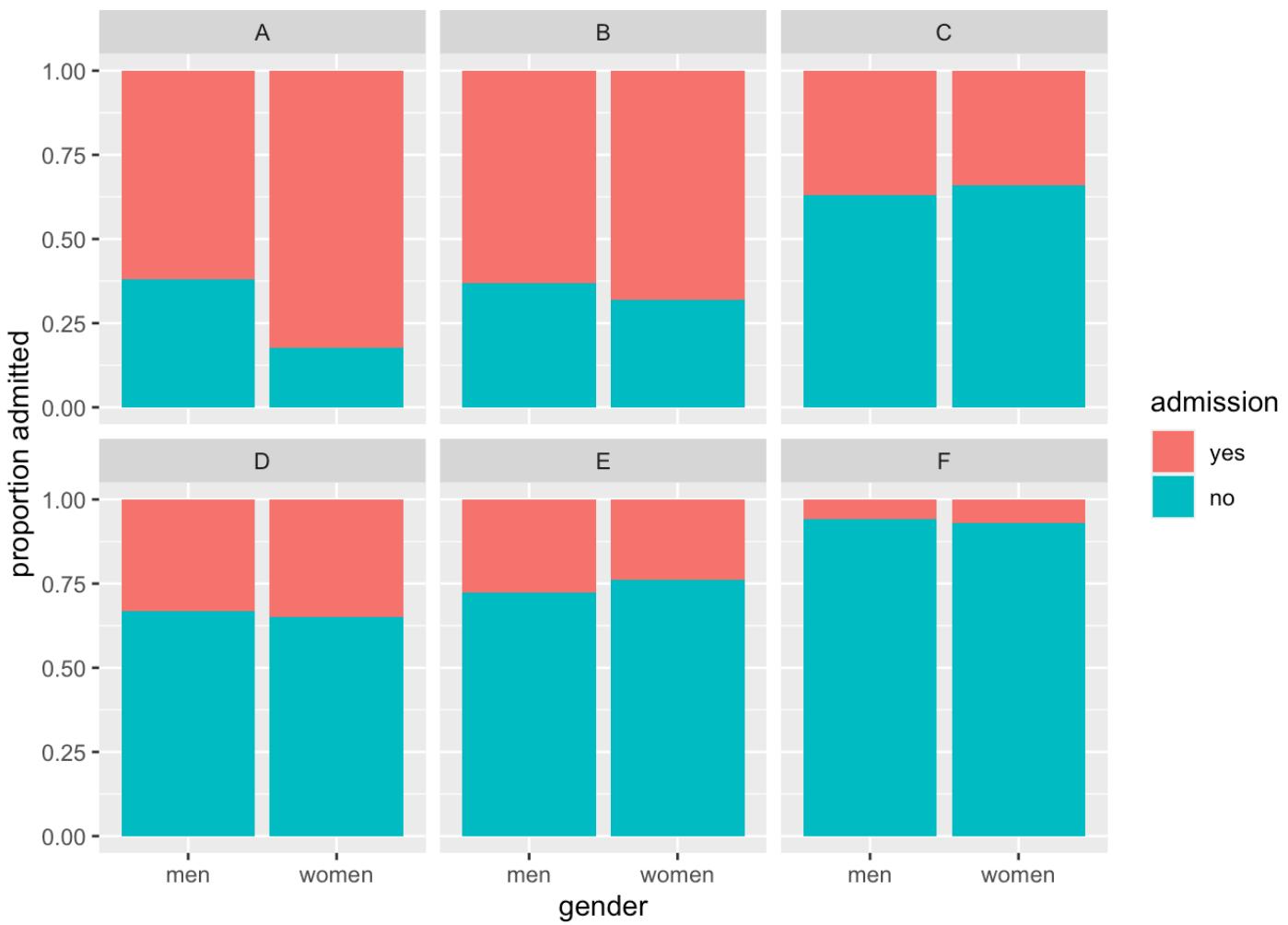
ggplot(admissions_plot, aes(gender, number_of_students, fill = admission)) +
  geom_bar(stat = "identity", position = "dodge") +
  facet_wrap(. ~ major)
  
```



The visualization breaks down the acceptances by major. This breakdown allows us to see that the majority of accepted men came from two majors: A and B. It also lets us see that few women applied to these majors.

Changing the position to "fill" in `geom_bar()` returns the following plot:

```
ggplot(admissions_plot, aes(gender, number_of_students, fill = admission)) +
  geom_bar(stat = "identity", position = "fill") +
  ylab("proportion admitted") +
  facet_wrap(. ~ major)
```



Setting "fill" as the position argument makes each set of stacked bars to have the same height. This makes it easier to compare proportions across groups. However, note that it can be hard to compare some groups with increasing number of stacked bars.

Altogether, this data suggests the following causal interpretation:

Gender → Major → Admission

In this interpretation of this data, women are proportionally less admitted because they choose more selective majors. Hence the association is correct, causal, in the right orientation, but it is indirect.

6.2.3.3 Common consequence

The last elementary situation is the case where the third variable is a common consequence. One fun imaginary example is given by Judea Pearl in "The Book of Why"¹⁰ about talent and beauty among Hollywood stars.

We assume that:

- Talent for acting is a quantitative value that is randomly distributed among the human population.
- Beauty is a quantitative value y randomly distributed among the human population and independently of talent for acting.
- Hollywood stars must have talent for acting and beauty or be very beautiful or very good actors.

We can use the computer to simulate such a situation as:

- draw x with a Gaussian of mean 0 and standard deviation 1
- draw y with a Gaussian of mean 0 and standard deviation 1 independently of x
- let $z = 1$ if $x + y > 1$

The following code simulates such data:

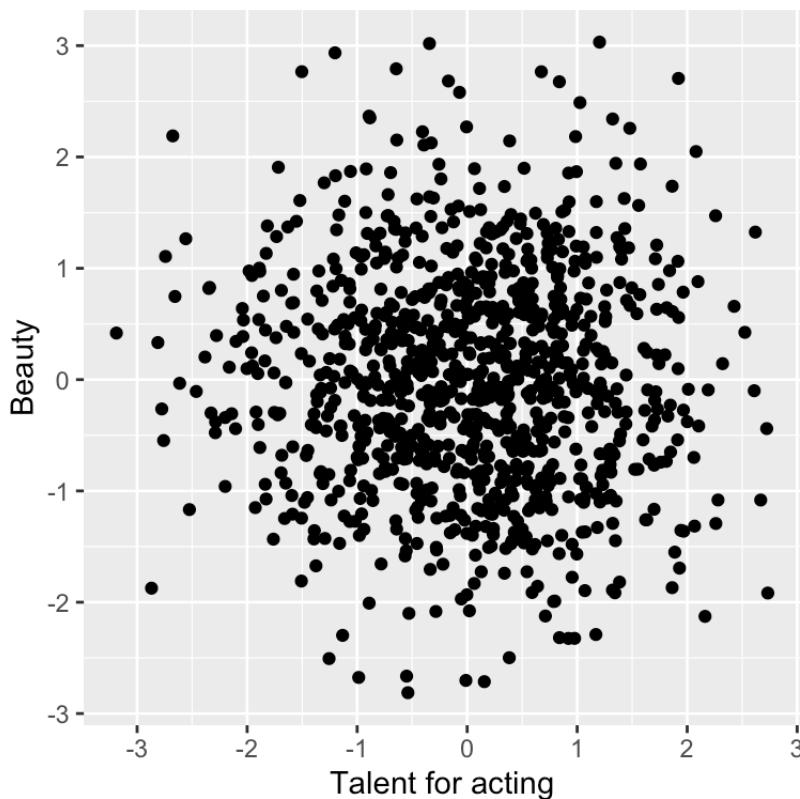
```
#####
## common consequence x->z<-y
#####

x <- rnorm(n) # talent for acting
y <- rnorm(n) # beauty
z <- x+y > 1

dt <- data.table(x, y, z)
```

In our simulated population, beauty and talent for acting do not correlate:

```
## x and y do not associate
ggplot(dt, aes(x=x, y=y)) + geom_point() + labs(x = 'Talent for acting', y = 'Beauty')
```

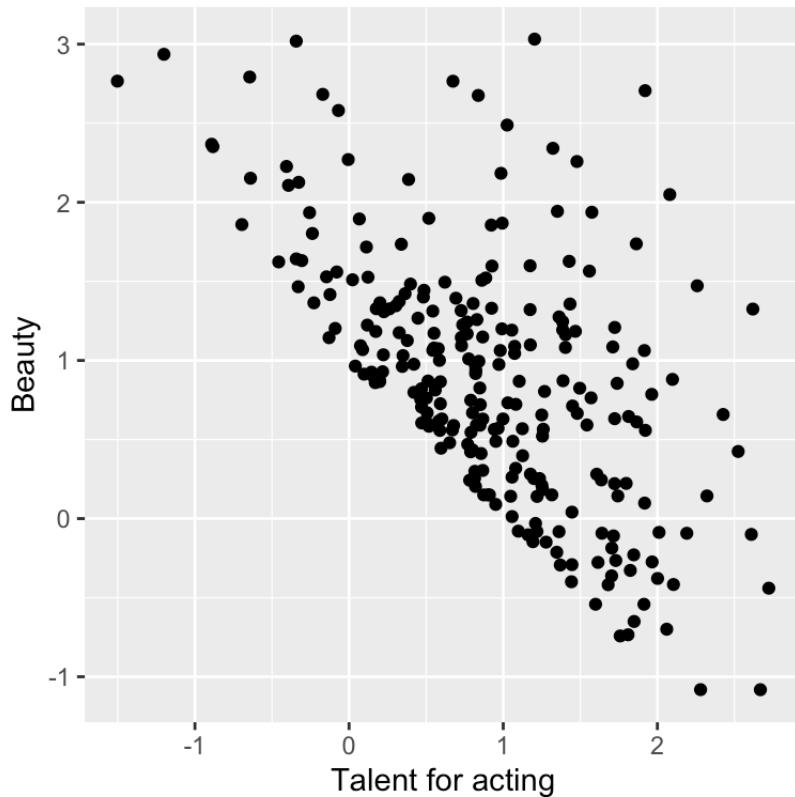


```
cor(x,y)
```

```
## [1] -0.03192247
```

The fun paradox comes when we restrict our analysis to the Hollywood stars:

```
## x associates with y when z is TRUE
ggplot(dt[z == TRUE], aes(x=x, y=y)) + geom_point() + labs(x = 'Talent for acting', y = 'Beauty')
```



```
dt[z, cor(x,y)]
```

```
## [1] -0.5843082
```

Hence, beautiful Hollywood stars tend to act worse than less beautiful stars.

This paradox is known as Berkson's paradox.¹¹ Berkson originally reported the paradox on patient data collected in a hospital. As these patients are in the hospital because they are sick, risk factors for their diseases, which do not correlate in the general population, happen to negatively correlate among them. Studies based on hospital data only can suffer from such bias.

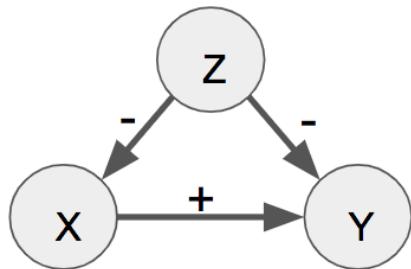
Altogether, the association arises once we condition on the third variable. This has two implications. First, conditioning should not be done systematically. Second, the data may already be conditioned and we are not aware of it! The data may have undergone some filtering during data analysis, or, less obviously, during data collection. Those possibilities should be considered when reporting associations between variables.

6.2.4 Simpson's paradox

In 1951, Simpson demonstrated that a statistical relationship observed within a group of individuals could be reversed within all subgroups that make up that population. This phenomenon, where a variable X seems to relate to a second variable Y in a certain way, but flips direction when the stratifying for another variable Z, has since been referred to as Simpson's paradox.

Simpson's paradox

Causal diagram



Association

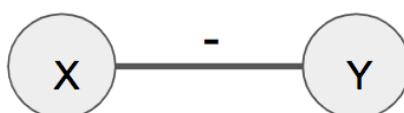


Figure 6.4: Simpson's paradox. The sign (+ or -) designates positive or negative effects (causal diagram, left) or associations (right).

We simulate a dataset for illustrating the Simpson's paradox with the help of the function `simulate_simpson()` from the library `correlation` :

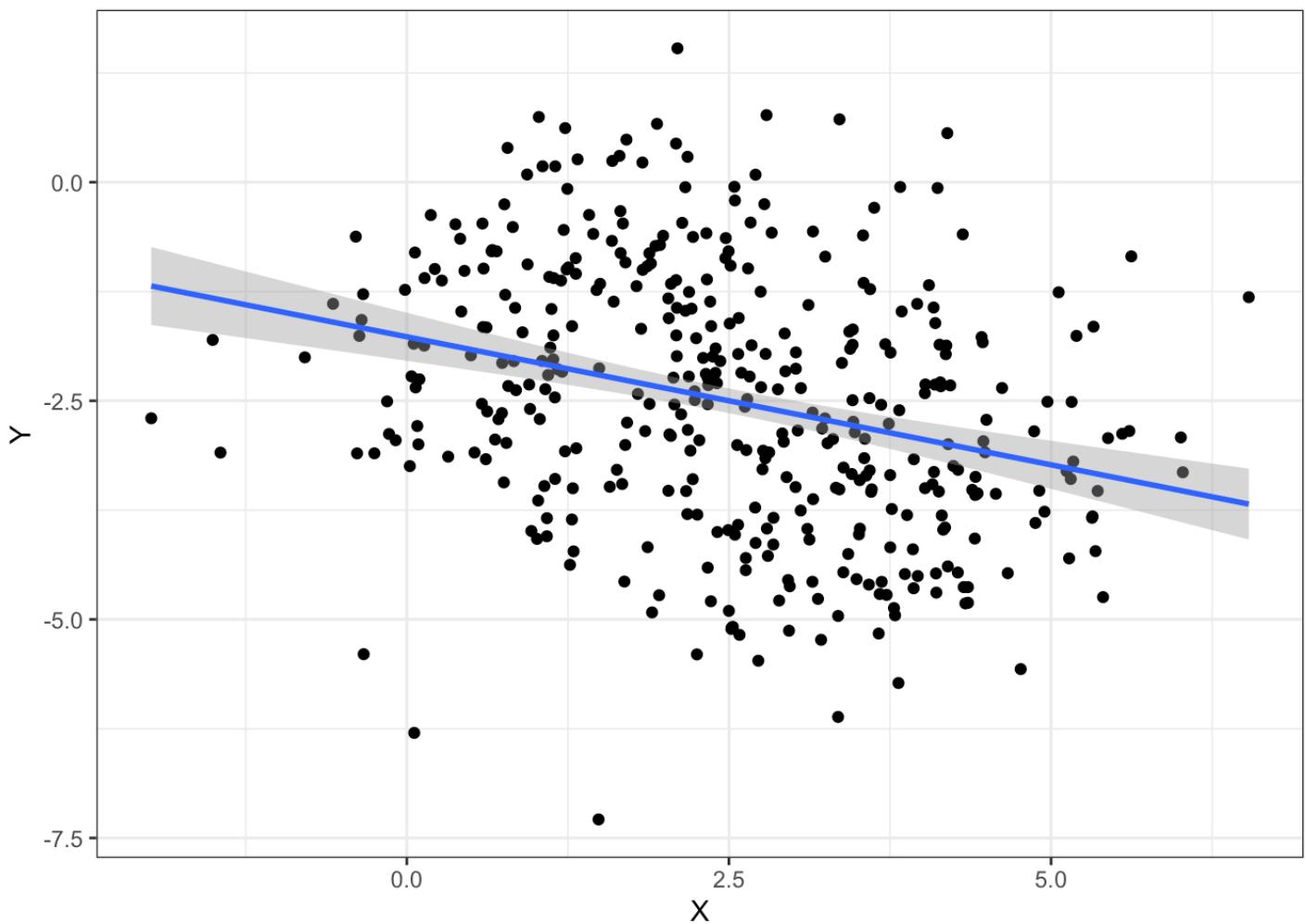
```

# devtools::install_github("easystats/correlation")
library(correlation)
data <- simulate_simpson(n = 100, groups = 4, r = 0.6) %>% as.data.table
colnames(data) <- c('X', 'Y', 'Z')
data

##          X         Y         Z
## 1: 0.7571442 -0.2531470 A
## 2: 1.3253338  0.2617510 A
## 3: 1.1057654 -1.0828252 A
## 4: 0.1859743 -0.3760158 A
## 5: 2.1004333  1.5311882 A
## ...
## 396: 3.3302676 -3.4945929 D
## 397: 3.3872684 -4.4615286 D
## 398: 2.4995125 -4.9025445 D
## 399: 3.7237566 -4.7194693 D
## 400: 4.3878277 -3.5184049 D
  
```

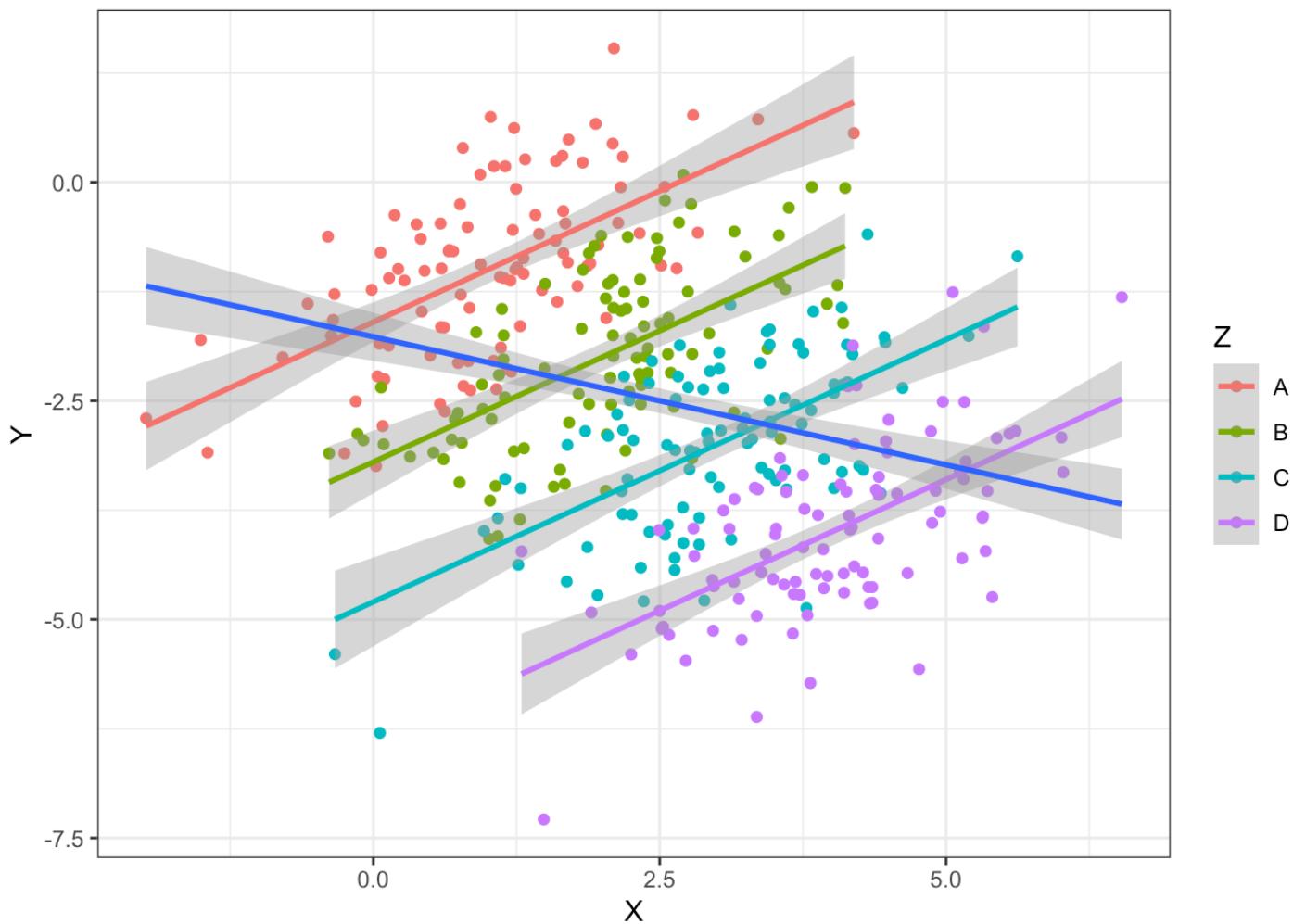
When visualizing the relationship between the variables `X` and `Y`, we observe a negative correlation between `X` and `Y`:

```
ggplot(data, aes(X,Y)) + geom_point() + geom_smooth(method='lm') + mytheme
```



However, when grouping by the variable Z, we observe a positive correlation which is the opposite direction as before:

```
ggplot(data, aes(x = X, y = Y)) +  
  geom_point(aes(color = Z)) +  
  geom_smooth(aes(color = Z), method = "lm") +  
  geom_smooth(method = "lm") + mytheme
```



6.3 Data presentation as story telling

Data presentation plays an essential role in every field from academic studies to professional practices. However, it is often underestimated. Sometimes people work really hard on what they aim to present but fail to present it properly. The actual output of the analysis performed is almost singularly responsible for giving the message we want to give to the audience. For this, we have to first really understand who our audience is considering its background and interests.

After spending some time reflecting on the audience, we want to be able to present the data and its analysis clearly and efficiently. We want to create a good presentation that, like a good story, is easy to understand, has a clear message and is exciting.

6.3.1 What is a story?

Before discussing strategies for turning visualizations into stories, we have to understand what a story actually is. A story is a set of observations, facts, or events, that are presented in a specific order such that they create an emotional reaction in the audience. The emotional reaction is created through the build-up of tension at the beginning of the story followed by some type of resolution towards the end of the story. We refer to the flow from tension to resolution also as the story arc, and every good story has a clear, identifiable arc.

Commonly, a story teller first introduces the topic, then presents the challenge followed by a series of actions. Finally, the resolution is presented.

6.3.2 Presentation structure

Usually, a presentation is clearly divided into three main elements just like a good story: an introduction, a central part, and a closure.

6.3.2.1 Introduction

A presentation should have an introduction which clearly states the motivation to the topic, gives an overview on the background needed to understand the data and finally states the goals of the presentation. The open questions should be formulated here. What is not yet known or unclear in the field that your analysis addresses?

6.3.2.2 Development

The central part of the presentation is all about transforming hypotheses, claims and results into slides.

Two common misconceptions have to be avoided when presenting visualizations in a presentation. First, that the audience can see our figures and immediately infer the points we are trying to make. Second, that the audience can rapidly process complex visualizations and understand the key trends and relationships that are shown. We need to do everything we can to help our audience understand the meaning of our visualizations and see the same patterns in the data that we see. This usually means “less is more”.

Sometimes, however, we do want to show more complex figures that contain a large amount of information at once. In those cases, we can make things easier for our audience if we first show a simplified version of the figure before we show the final one in its full complexity.

Usually, we first show descriptive plots to first present the data and let the audience be able to understand the data. Then, we use associative plots to support claims, results and hypotheses.

Ideally, one slide contains exactly one claim supported by one plot. For this, the plot type should be chosen carefully. The title of the slide or figure is a very important element. The title should give the take-home message of the presented visualizations. It is a clear and simple interpretation of the plot. For instance, a title such as “Scatterplot of diamond price versus carat” for Figure 6.1 would not be informative since it is evident from the type of the plot and the axis labels that we look at a scatter plot of price versus carat. More informative is your interpretation of it, which the reader can judge from the plot. Do you want to claim that “Price increases nearly linearly with carat”, or rather that “There remains much variations in price among diamond with the same carat”? Such statements make informative slide titles.

6.3.2.3 Closure

The presentation should ideally have a strong closure. We can usually start with a summary of the presentation including a clear conclusion. With the conclusion, the audience should be able to answer the fundamental question: what do we learn from the presented data analysis? The conclusions typically correspond to the open questions formulated in the introduction. Often, an outlook containing possible next steps can be briefly summarized at the end of a presentation.

6.4 Guidelines for coloring in data visualization

Colors are an effective and powerful medium for communicating meaning. In data visualization, color sets the tone and enforces a message for the underlying visual display.

6.4.1 Color coding in R

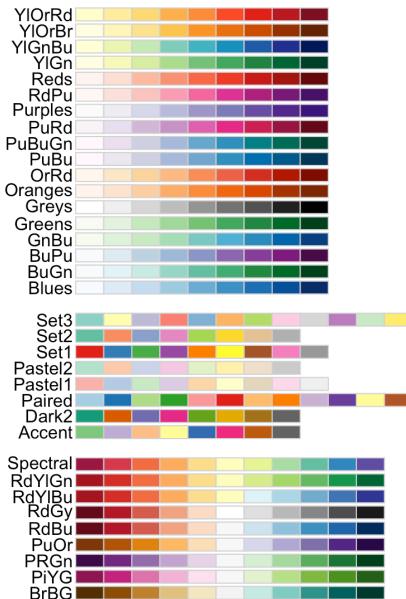
Basically, there are four options for color coding with `ggplot2` in R:

1. Using default colors
2. Explicitly setting color names (e.g. “red”, “blue”)
3. Explicitly setting RGB or HTML color codes (e.g. 00-FF)
4. Explicitly setting color palettes

6.4.1.1 Color palettes

The package `RColorBrewer` provides a comprehensive set of nice and color palettes:

```
library(RColorBrewer)
display.brewer.all()
```



Generally, we use sequential palettes for continuous variables to show quantitative differences:



We use qualitative palettes for categorical variables to separate items into distinct groups and we use diverging palettes for numeric variables that have a meaningful central value or breakout point (e.g. 0).



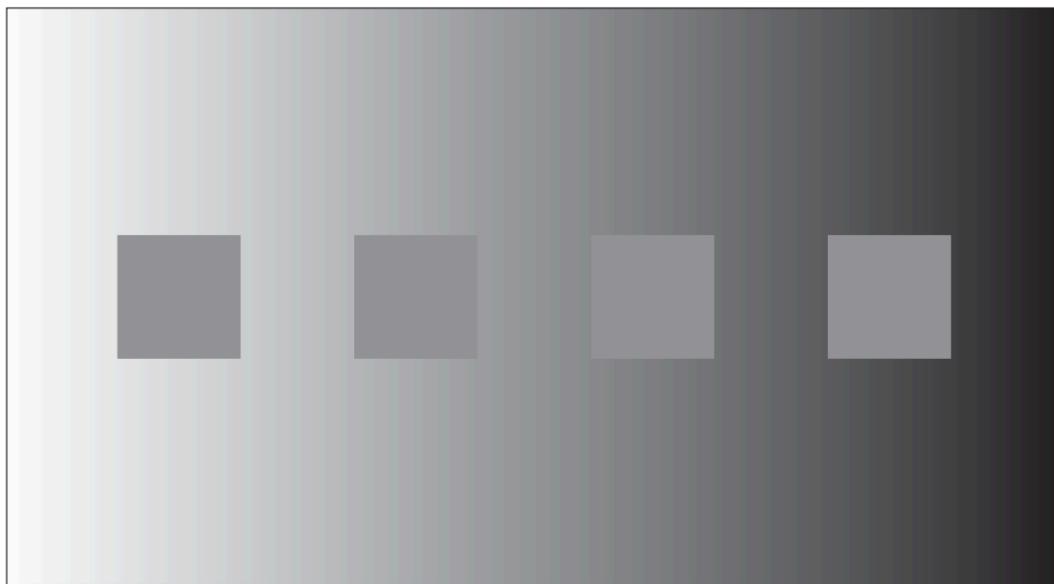
6.4.2 General rules for color coding

Even though color coding can be very helpful in several visualizations, it has to be used with precaution. Like all aspects of visual perception, we do not perceive color in an absolute manner. Perception of an object is influenced by the context. Hence, visual perception is relative, not absolute.

Rule #1: consistent background

If we want different objects of the same color in a table or graph to look the same, we have to make sure that the background is consistent.

In the following example, the squares seem to have a different color but they actually don't. The background makes it hard to compare them.



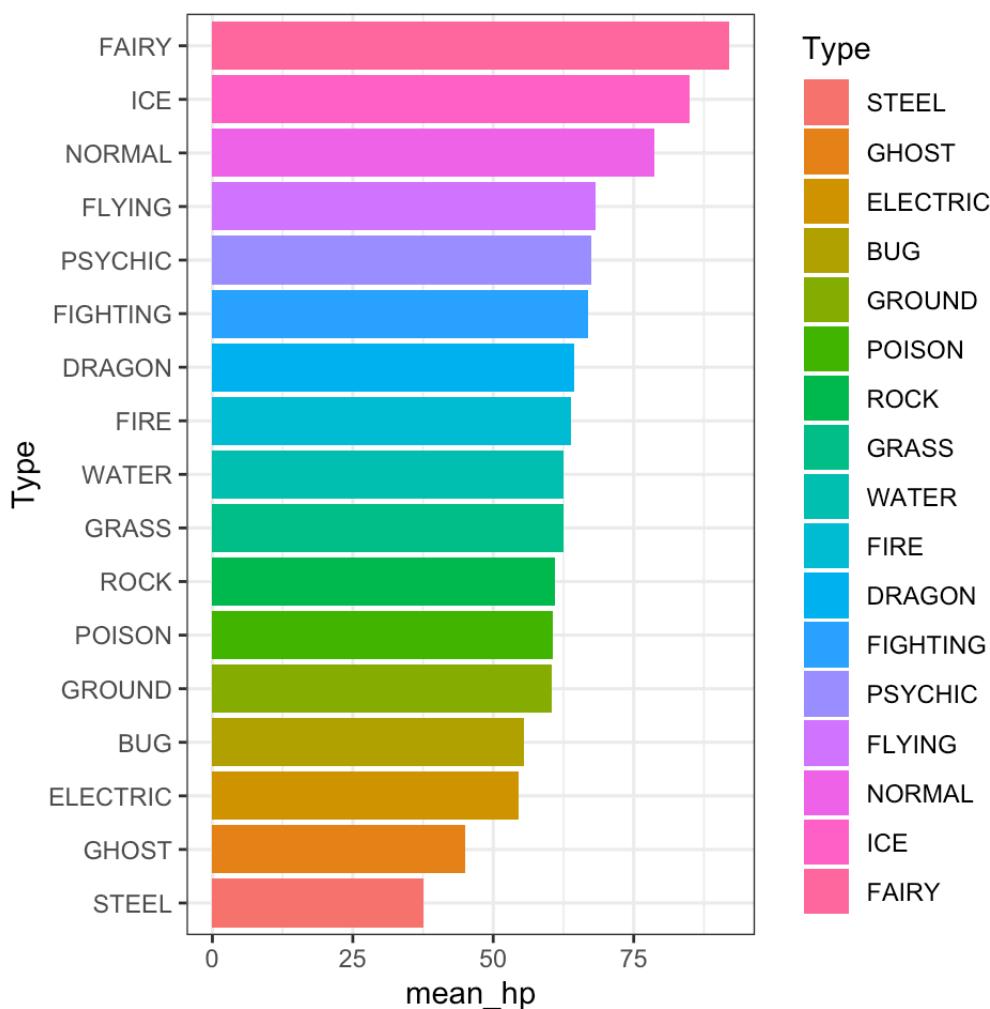
Rule #2: sufficient contrast for visibility

If we want objects in a table or graph to be easily seen, we need to use a background color that contrasts sufficiently with the object that we want to visualize.

Rule #3: meaningful color usage

We use color coding only when we really need it to serve a particular communication goal.

In the following example, the added colors provide no additional information:



Rule #4: color usage with restraint

We use different colors only when they correspond to differences of meaning in the data.

Rule #5: less is more

Use soft, natural colors to display most information and bright and/or dark colors to highlight only particular information that requires greater attention.

6.5 General do's and don'ts in data visualization

6.5.1 Do's

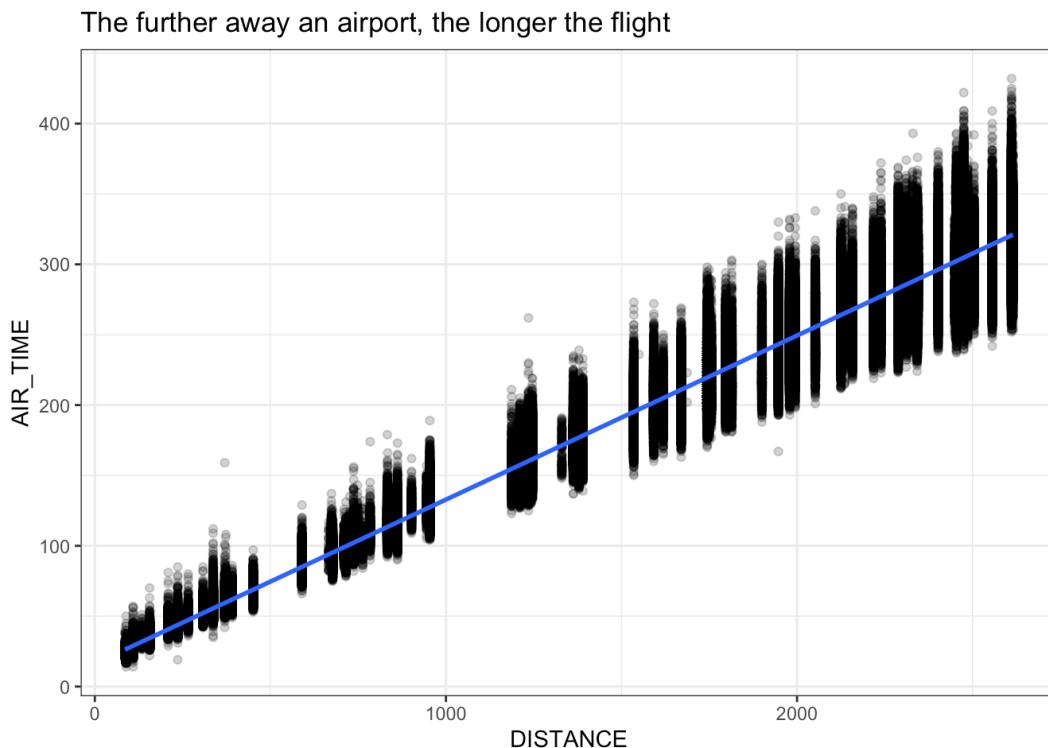
Keep visualizations simple

Keeping simple visualizations is probably the most important guideline. Often people assume that complex plots are useful visualizations, while the opposite is actually true. Good plots should have simple messages and make the visualized data as easy to understand as possible. A visualization should be effective and simple. Do not try to present too much information.

Have meaningful and expressive titles

Figure title (or slide title) states the finding (what do we observe?) and not the methods (how do you do it?). For example a good title is “The further away an airport, the longer the flight” instead of “Scatter-plotting air time and distance”.

```
flightsLAX <- fread('extdata/casestudy1/flightsLAX.csv')
ggplot(flightsLAX, aes(DISTANCE, AIR_TIME)) +
  geom_point(alpha=0.2) +
  geom_smooth(method='lm') +
  labs(title = 'The further away an airport, the longer the flight') + mytheme
```



Always label the axes

Axes, colors, and shapes should be labelled. All labels should be legible for the audience (big fonts).

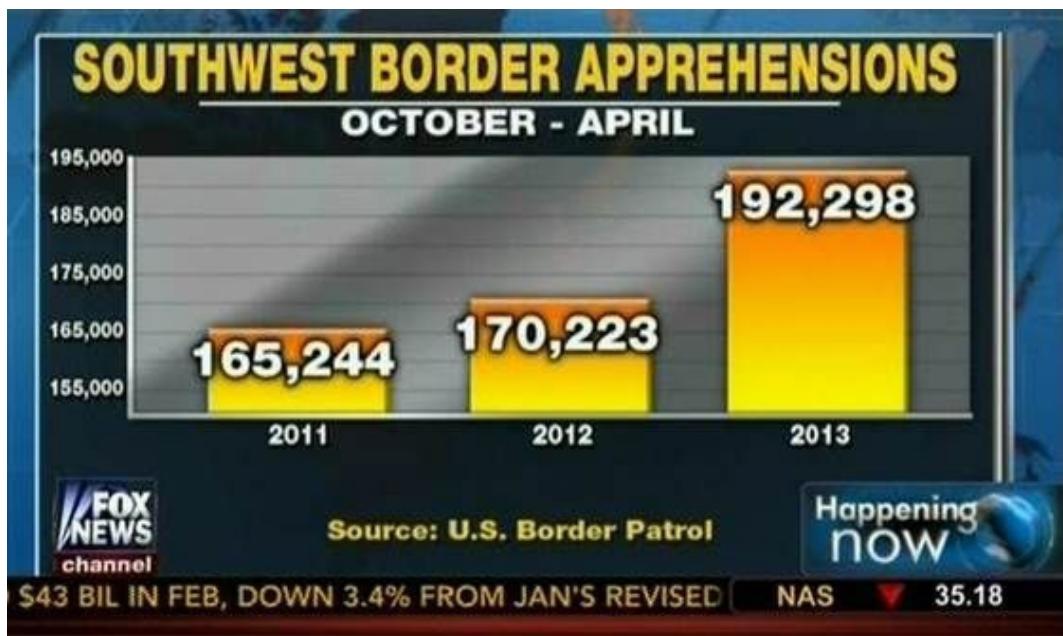
Always keep the goal of the visualization on mind

Always think about the message that needs to be reported with the visualization. What is the goal? Focus on the story and the claims or conclusions you want to make.

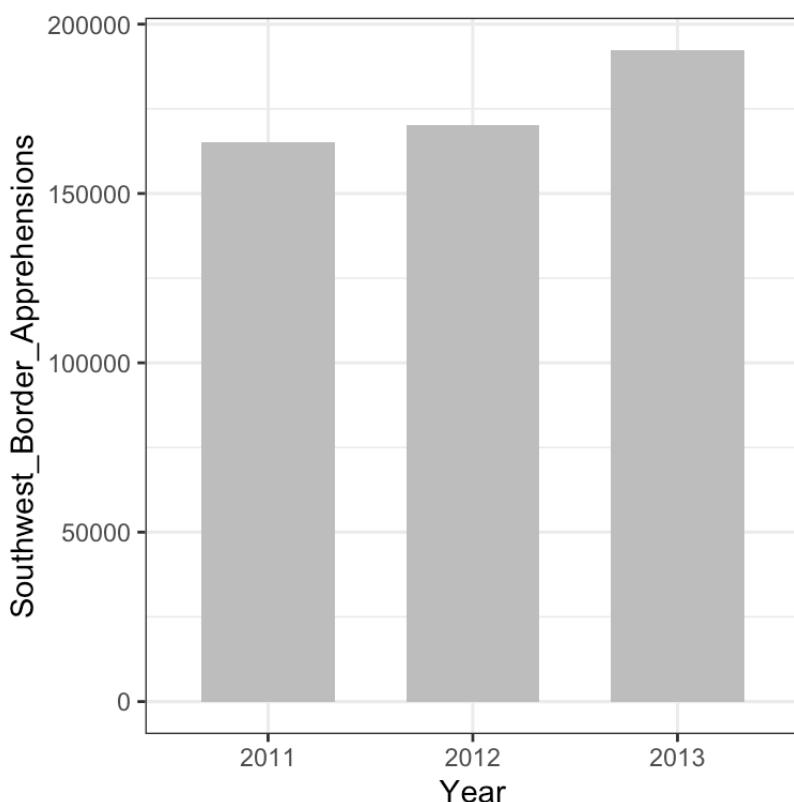
Know when to include 0

When using barplots, it is typically dishonest not to start the bars at 0. This is because, by using a barplot, we are implying the length is proportional to the quantities being displayed. By avoiding 0, relatively small differences can be made to look much bigger than they actually are. This approach is often used by politicians or media organizations trying to exaggerate a difference.

We can have a look at this illustrative example:



The plot was shown by Fox News (<http://mediamatters.org/blog/2013/04/05/fox-news-newest-dishonest-chart-immigration-enf/193507>). From the plot above, it appears that apprehensions have almost tripled, when in fact they have only increased by about 16%. Starting the graph at 0 illustrates this clearly:



6.5.2 don'ts

Good plotting style is like good writing style: say the most with the least. We can here quote Antoine de Saint-Exupery:

"Perfection is achieved not when there is nothing more to add, but when there is nothing left to take away".

The concept of *data-ink ratio* is therefore useful to critically chose a visualization. In particular, visualizations that contain at least one the following criteria can be considered as chart junk and should be avoided:

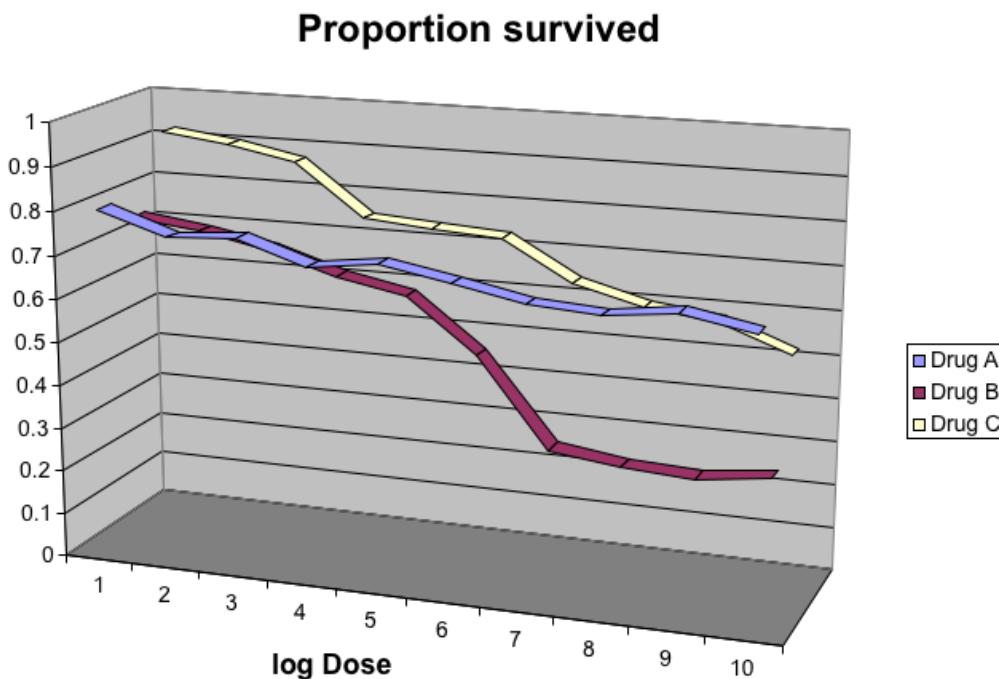
- double encoding (color and axis encode the same)
- heavy or dark grid lines
- unnecessary text
- ornamented chart axes
- pictures within graphs
- shading or pseudo 3D plots

A good example of a transformation of a bad plot into a good plot can be obtained from:

<https://www.darkhorseanalytics.com/blog/data-looks-better-naked/>

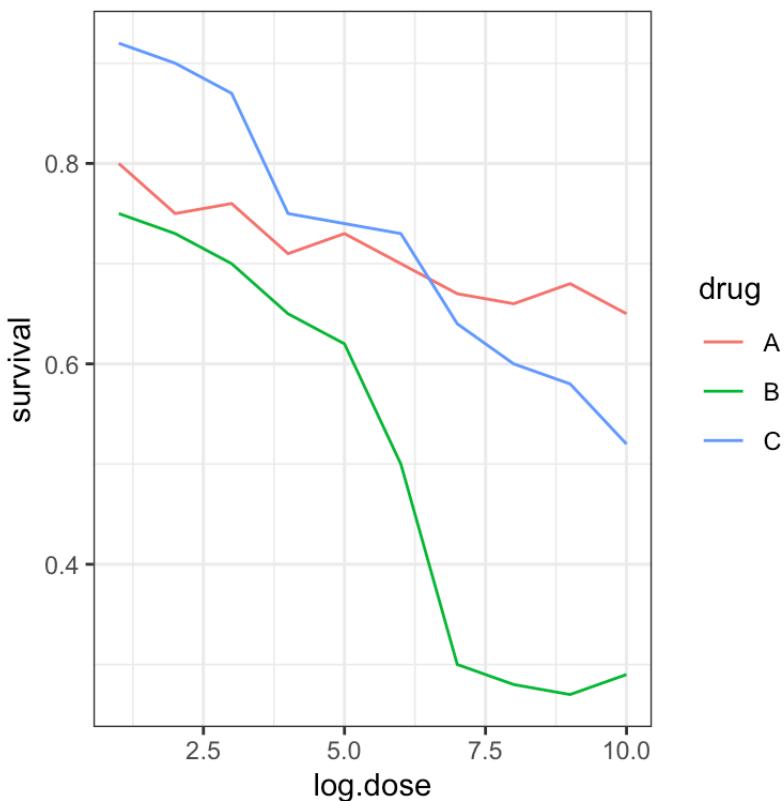
Avoid pseudo three-dimensional plots

The figure below, taken from the scientific literature¹² shows three variables: dose, drug type and survival. Although our screens are flat and two dimensional, the plot tries to imitate three dimensions and assigns a dimension to each variable.



However, humans are not good at seeing in three dimensions (which explains why it is hard to parallel park) and our limitation is even worse with pseudo-three-dimensions. To see this, we can try to determine the values of the survival variable in the plot above. Probably, we cannot really tell when the purple ribbon intersects the red one.

Instead, we can easily use color to represent the categorical variable to avoid the pseudo 3 dimensional construction:



This plot demonstrates that using color is more than enough to distinguish the three lines.

Do not manipulate plots

A good practice in data science is to be as honest as possible with the plots we create. Intentional manipulation is definitely a “don’t”.

6.6 Summary

By now, you should be able to:

- Distinguish exploratory figures showing data distributions, from associative figures
- Know and mind the basic reasons for misleading associations:
 - not robust
 - reverse causal direction
 - common cause
 - indirect effect
 - common consequence
- Mind data/ink ratio by showing more data and reducing decorations
- Prepare your report to provide novel insights. The audience should be able to easily answer the question: “What did I learn?”

6.7 Resources

- <https://humansofdata.atlan.com/2019/02/dos-donts-data-visualization/>
- <http://paldhous.github.io/ucb/2016/dataviz/week2.html>
- Fundamentals of Data Visualization, Claus O. Wilke, <https://clauswilke.com/dataviz/telling-a-story.html>
- Introduction to Data Science, Rafael A. Irizarry, <https://rafalab.github.io/dsbook/index.html>

Advanced (proofs of conditional dependences for the 3-variable elementary causal diagrams):

C. Bishop, Pattern Recognition and Machine Learning. <https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/>

8. https://en.wikipedia.org/wiki/Anscombe%27s_quartet ↪
9. <https://opinionator.blogs.nytimes.com/2014/04/12/parental-involvement-is-overrated> ↪
10. <http://bayes.cs.ucla.edu/WHY/> ↪
11. https://en.wikipedia.org/wiki/Berkson%27s_paradox ↪
12. DNA Fingerprinting: A Review of the Controversy Kathryn Roeder Statistical Science Vol. 9, No. 2 (May, 1994), pp. 222-247 ↪

Chapter 7 Resampling-based Statistical Assessment

Suppose a friend says she can correctly predict the winning team in any football game, due to her deep knowledge of the sport. To test this, we ask for her predictions for two Champions League games. She turns out to be right both times. On a first glance, this seems very impressive: she has a 100% success rate! Should we hence bet a lot of money on her next prediction? What if she is just guessing and got lucky?

Someone who just flips a coin to decide the winner has a 25% chance to get lucky and guess two games correctly. Accordingly, betting all our life savings that she will be right again next game may not be a very good idea. Hence, the danger is to conclude something based on a limited amount of data. Apparent trends can arise purely by chance, and if we are not careful this can lead us into making the wrong conclusions. Now, what if the friend had correctly predicted the outcome in 4 out of 5 games? What about 237 out of 286 games? When should we start taking her claim seriously?

This chapter introduces concepts and methods to answer these types of questions. We cover the concept of hypothesis testing and of **statistical significance**, which is another way of saying that **a trend is unlikely to have arisen purely by chance**. We also introduce the concept of confidence interval which models our uncertainty when estimating parameters such as the mean of a variable. To this end, we provide two largely applicable computational methods: permutation testing and case resampling. These methods are based on resampling the data at hand, thereby making little modeling assumptions.

7.1 The yeast dataset

7.1.1 The experiment

This section introduces a dataset that we will use throughout this chapter, and occasionally in the following chapters.

Yeast is a good old friend of humans. Thanks to yeast, we can make bread, wine, and (not the least for TUM and Munich) beer. Yeast is also very much studied by biologists. The yeast strain that is commonly used in research labs grows poorly on maltose compared to wild yeast strains. Hard to brew malt beer with the lab strain... One may wonder whether the lab strain has acquired a genetic mutation causing this poor fitness in maltose media. If so, on which chromosome, near which gene?

Our dataset (Gagneur et al. 2013) allows addressing these questions (and further yeast genetic questions). The lab strain was crossed with a wild isolate growing well on maltose. Overall, 184 offsprings, also called segregants, were obtained. During a cross, parental chromosomes are recombined at discrete random locations in a process called cross-over.¹³ Consequently, the chromosomes of the segregants consist of alternated segments inherited from either parent. Yeast has 16 chromosomes. Figure 7.1 illustrates this crossing process for one chromosome.

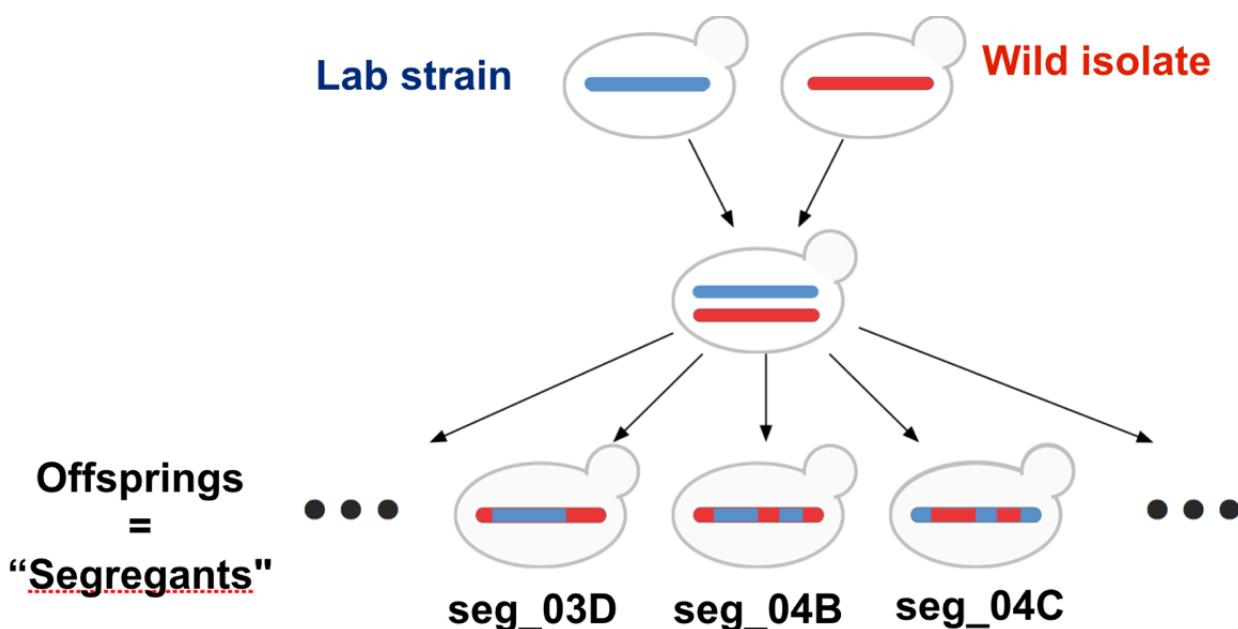


Figure 7.1: Cross of the lab strain and wild isolate. Meiotic recombination implies that chromosomes of the offsprings consist of alternated segments inherited from either parent.

This shuffling of the genetic information is helpful to identify on which chromosomal location(s) genetic variations responsible for the growth rate difference could reside.

7.1.2 Genotype

The `genotype` table reports the genotype of each the 184 yeast strains at 1,000 genomic locations called genetic markers. At each marker, the genotype values are either “Lab strain” or “Wild isolate” (Figure 7.2).

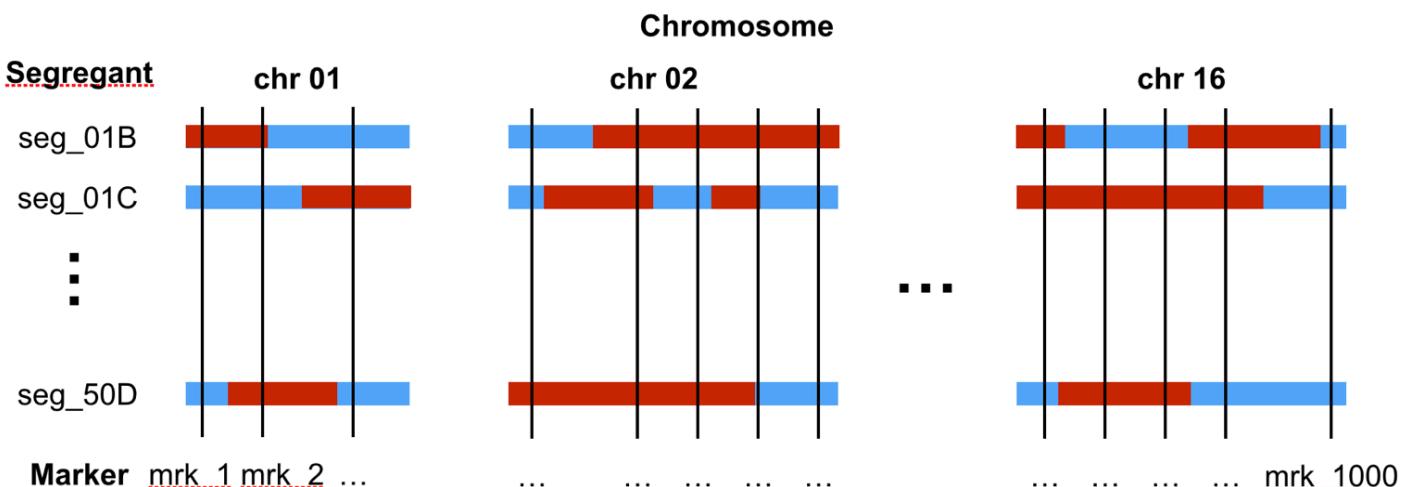


Figure 7.2: Sketch of the genotype of segregants (rows) across the 16 chromosomes. The genotypes are provided at 1,000 genomic positions (called markers, vertical line).

See below for a section of the data table:

```

genotype <- fread("extdata/eqtl/genotype.txt")
genotype <- genotype %>%
  melt(id.vars = 'strain', variable.name = 'marker', value.name = 'genotype')
genotype

##           strain     marker   genotype
## 1: seg_01B     mrk_1    Lab strain
## 2: seg_01C     mrk_1 Wild isolate
## 3: seg_01D     mrk_1    Lab strain
## 4: seg_02B     mrk_1    Lab strain
## 5: seg_02C     mrk_1 Wild isolate
## ---
## 157996: seg_49A mrk_13314    Lab strain
## 157997: seg_49B mrk_13314 Wild isolate
## 157998: seg_50A mrk_13314    Lab strain
## 157999: seg_50B mrk_13314 Wild isolate
## 158000: seg_50D mrk_13314    Lab strain

```

If we want to know where the markers are located in the genome, we can consult the `marker` table. This table reports genomic coordinates of the markers (chromosome, start, and stop):

```

marker <- fread("extdata/eqtl/marker.txt")
marker

```

```

##           id chrom  start   end
## 1:     mrk_1 chr01  1512 2366
## 2:     mrk_14 chr01 29161 29333
## 3:     mrk_27 chr01 38275 38317
## 4:     mrk_40 chr01 47695 47695
## 5:     mrk_54 chr01 56059 56059
## ---
## 996: mrk_13260 chr16 928119 928130
## 997: mrk_13274 chr16 931402 931594
## 998: mrk_13287 chr16 934388 934624
## 999: mrk_13300 chr16 939647 939679
## 1000: mrk_13314 chr16 944640 944667

```

7.1.3 Growth rates

The `growth` table contains the growth rates expressed in generations per day for each strain in five different growth media. These growth media are YPD (glucose), YPD_BPS (low iron), YPD_Rapa (Rapamycin), YPE (Ethanol), YPMalt (Maltose).

```

growth <- fread("extdata/eqtl/growth.txt")
growth <- growth %>% melt(id.vars = "strain", variable.name = 'media', value.name = 'growth_rate')
growth

##      strain  media growth_rate
## 1: seg_01B    YPD   12.603986
## 2: seg_01C    YPD   10.791144
## 3: seg_01D    YPD   12.817268
## 4: seg_02B    YPD   10.299210
## 5: seg_02C    YPD   11.132778
##   ---
## 786: seg_49A YPMalt   4.592395
## 787: seg_49B YPMalt   5.702087
## 788: seg_50A YPMalt   4.303382
## 789: seg_50B YPMalt   6.583852
## 790: seg_50D YPMalt   7.421968

```

7.1.4 Genotype-growth rate association in maltose at a specific marker

In this Chapter, we focus on a simple, targeted question. We know beforehand that the gene MAL13 is important for maltose metabolism. Could genetic variation between the lab strain and the wild isolate near the gene MAL13 be responsible for the growth difference in maltose?

Marker 5211, which starts at positions 1069229 of chromosome 07, is the closest marker to the gene MAL13. We thus ask whether genotype at marker 5211 associates with growth rate in maltose.

To assess this hypothesis, we first create a data table called `dt` that contains the relevant data and visualize with a boxplot how growth rates distributes depending on the genotype at marker 5211:

```

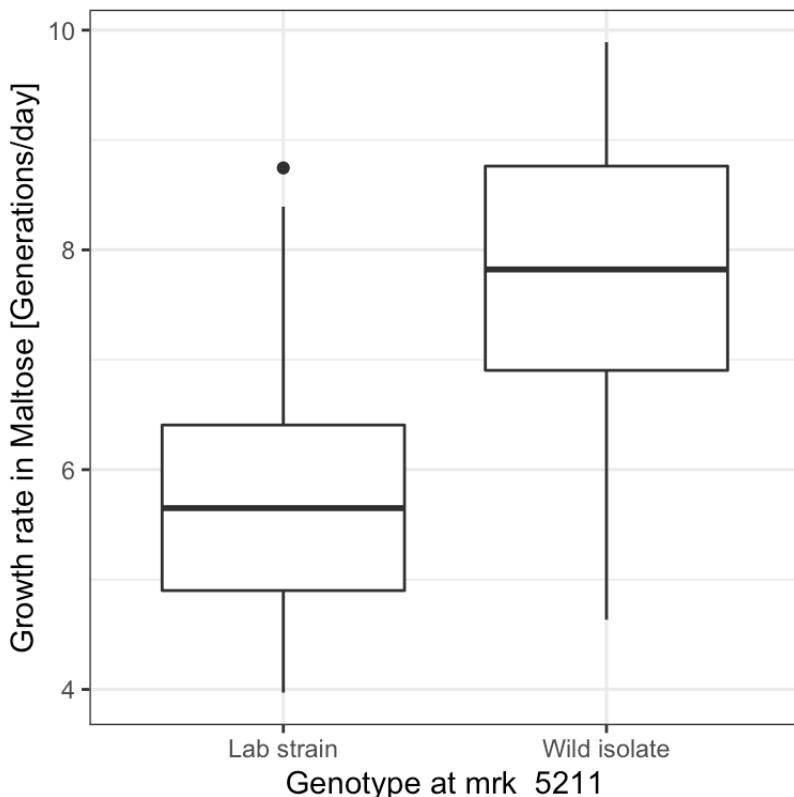
mk <- marker[chrom == "chr07" & start == 1069229, id]

dt <- merge(
  growth[media == 'YPMalt'],
  genotype[marker == mk, .(strain, genotype)],
  by = 'strain'
)

p <- dt %>%
  ggplot(., aes(genotype, growth_rate)) +
  geom_boxplot() +
  xlab(paste0("Genotype at ", mk)) +
  ylab("Growth rate in Maltose [Generations/day]") +
  mytheme

```

p



```

dt[genotype == 'Wild isolate', median(growth_rate, na.rm=T)] -
dt[genotype == 'Lab strain', median(growth_rate, na.rm=T)]

## [1] 2.172018

```

We see that genotype at that marker indeed associates with a strong difference in growth rates in the Maltose media, with a difference between the medians of 2.17 generations per day.

But, as we already discussed in the motivating section, we need to be careful before making any conclusions. Maybe the pattern we see is an artifact of random variation and would disappear if we had more data. In the following we approach this issue with two concepts. We first look at statistical hypothesis testing, assessing whether the association could have arisen by chance. Next, we will consider parameter uncertainty, which will provide error bars around our difference of medians estimate.

7.2 Statistical hypothesis testing

Statistical hypothesis testing, often just referred to as hypothesis testing, is a method to assess whether an observed trend could have arisen by chance. We first describe intuitively a specific hypothesis testing method called **permutation testing**, to then describe the general concept.

7.2.1 Permutation testing: An intuitive build-up

We take the proverbial “**Devil’s Advocate**” standpoint. We **consider the possibility that such a large difference of growth rate medians could often arise by chance**, would we make arbitrary groups of the same size than those defined by the genotype.

To simulate such random data, we **permute the values of the genotype keeping the growth rate values fixed**. To permute values of a vector we can use the R function `sample()` with default parameters as in the example below:

```
LETTERS[1:8]
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H"
```

```
sample(LETTERS[1:8])
```

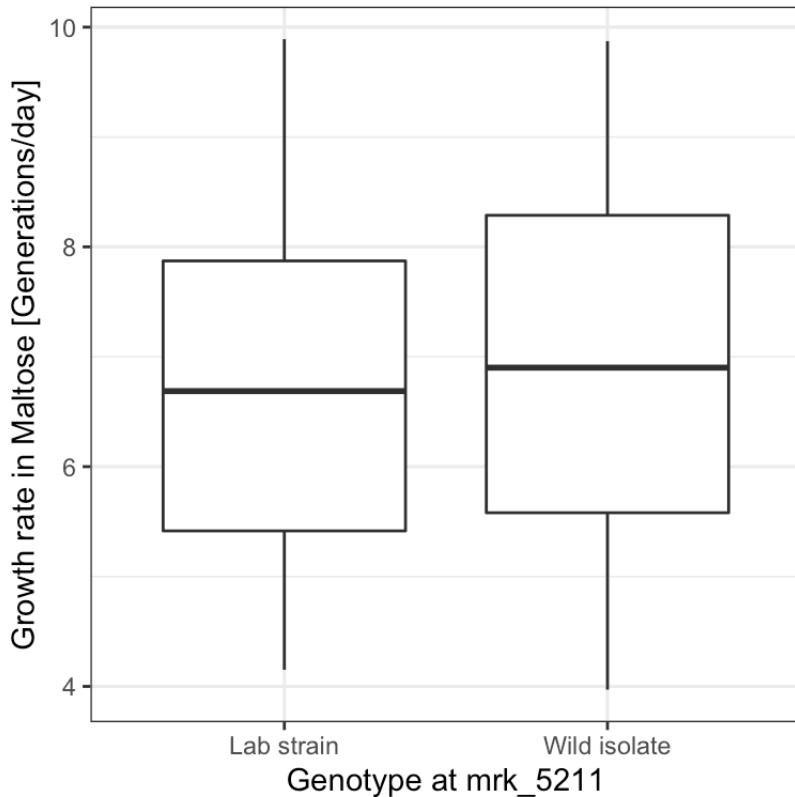
```
## [1] "C" "G" "D" "H" "E" "A" "F" "B"
```

We now shuffle the genotype column. To keep the original data safe, we work on `dt_permuted`, a copy of the table `dt`.

```
dt_permuted <- copy(dt)
set.seed(0) # the seed of the random number generator
dt_permuted[, genotype:=sample(genotype)]
```

For this simulated data, the boxplot looks less impressive:

```
# The %+% operator updates the dataset of a ggplot object
# convenient, isn't it?
p <- p %+% dt_permuted
p
```



We can also recompute the difference of medians. To not have repeated code, let us [define a function](#) (See Appendix B) that takes a table as input. We check immediately that our function properly returns the original difference of medians when applied to `dt`.

```
diff_median <- function(tab){
  tab[genotype == 'Wild isolate', median(growth_rate, na.rm=T)] -
  tab[genotype == 'Lab strain', median(growth_rate, na.rm=T)]
}
T_obs <- diff_median(dt)
T_obs
```

```
## [1] 2.172018
```

The difference of medians in this permuted dataset is now only 0.21 generations per day:

```
diff_median(dt_permuted)
```

```
## [1] 0.2135481
```

This is not fully convincing yet. Maybe our “devil’s advocate” has been unlucky with this one randomization. However, we can easily repeat this operation many times, e.g. 1,000 times. We denote the number of permutations m . We iterate m times using a for loop (See Appendix B). We record the difference of medians of the i -th iteration in a vector called `T_permuted`.

```
# number of permutations
m <- 1000

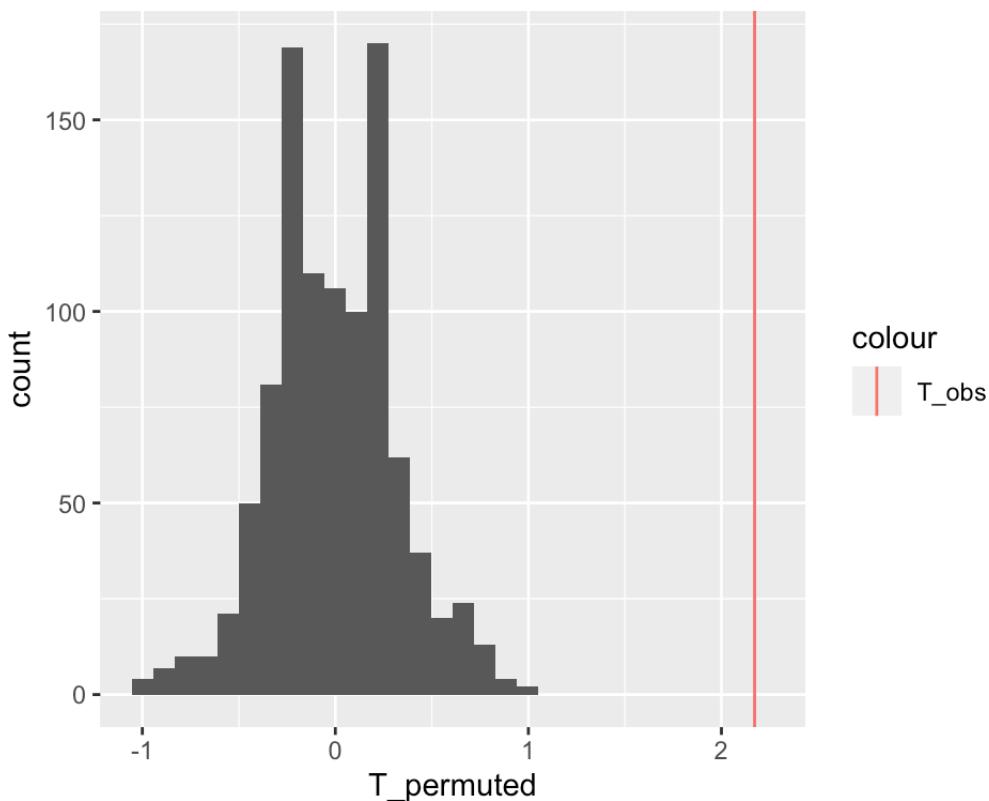
# initialize T_permuted with missing values
# (safer than with 0's)
T_permuted <- rep(NA, m)

# iterate for i=1 to m
for(i in 1:m){
  # permute the genotype column in place
  dt_permuted[, genotype:=sample(genotype)]
  # store the difference of medians in the i-th entry of T_permuted
  T_permuted[i] <- diff_median(dt_permuted)
}
```

Let us look at how these values distribute with a histogram and mark our original observation with a vertical line:

```
ggplot( data.table(T_permuted), aes(x = T_permuted) ) +
  geom_histogram() +
  geom_vline( aes(xintercept=T_obs, color = "T_obs") )

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



The observed difference of medians stands far out from the distribution of the permuted data. We never observed a difference equal or larger than the original one among 1,000 permutations. We can conclude it is unlikely that such a large difference could have arisen by chance.

This empirical approach is quite intuitive. Let us now formalize it and precisely specify the underlying assumptions in order to understand when and how we can apply it.

7.2.2 Concepts of Statistical Hypothesis Testing

We just implemented one type of Hypothesis test. Figure 7.3 provides an overview of Hypothesis testing.

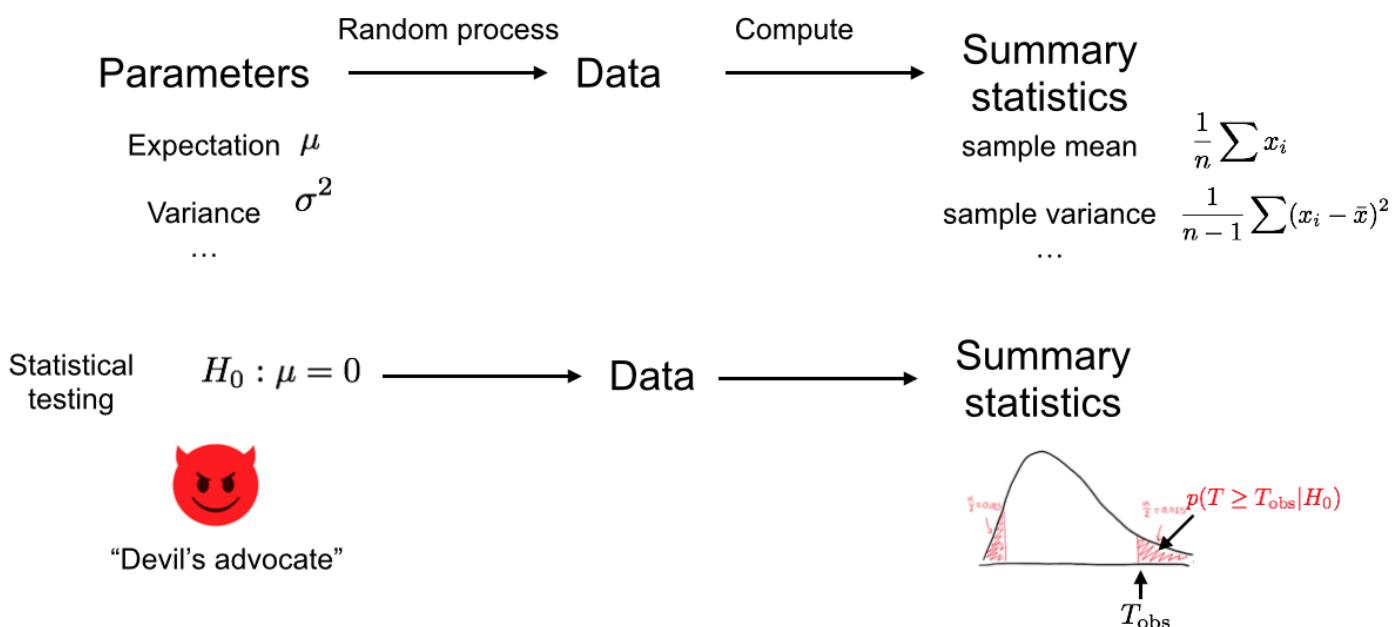


Figure 7.3: We assume an underlying random process (i.e. ‘Nature’). We collected data which is a particular realization of this random process, and from this data we computed a test statistic. In the bottom row, we now play the role of the Devil’s Advocate and assume that the underlying random process conforms to the null hypothesis. Based on this assumption, different realization datasets could arise as different realizations of the random process, for which the test statistics would get different values. Then we compute how likely it is to see the test statistics as, or more, extreme as the ones we got from our actual data. We use this probability to reject or not the null hypothesis.

7.2.2.1 Test statistic

To develop our test, we first need to define a **test statistic** (Figure 7.3). This is a single number that summarizes the data and captures the trend. The more extreme the test statistic, the stronger the trend.

The test statistic is often denoted T .

Here we have considered the difference of the medians:

$$T = \text{median}_{i \in \text{Wild}}(y_i) - \text{median}_{i \in \text{Lab}}(y_i)$$

We could equally consider the difference of the means, or the difference of the means divided by the within-group standard deviation, etc. Some statistics are more useful than others, because one can work analytically with them (see next chapter) or because they are more sensitive. Clearly, if we had considered as test statistics the difference between just two random values of each group rather than the difference of medians, discriminating the observed data from the permuted ones would have been more difficult.

7.2.2.2 The Null Hypothesis

Our test statistic is calculated from a limited number of observations. In our data we see a large difference in median growth rates, but maybe if we had much more data, this difference would disappear, or even change sign. To assess this, we need a negative control. We get such a negative control by setting a **null hypothesis** H_0 , i.e. by assuming that the trend we observe is not real and arose purely by chance (Figure 7.3).

The null hypothesis can be compared to the proverbial “Devil’s Advocate”. To test whether a trend is real, we take the skeptical position and assume it is not. It is the same thing we also did in the football example, when we assumed that the friend was just guessing the outcome of the games.

The exact null hypothesis depends on the problem. In our example, the **null hypothesis was statistical independence of genotype and growth rate**. It can also be that a mean is 0, a Pearson correlation is 0, etc.

7.2.2.3 The P-value

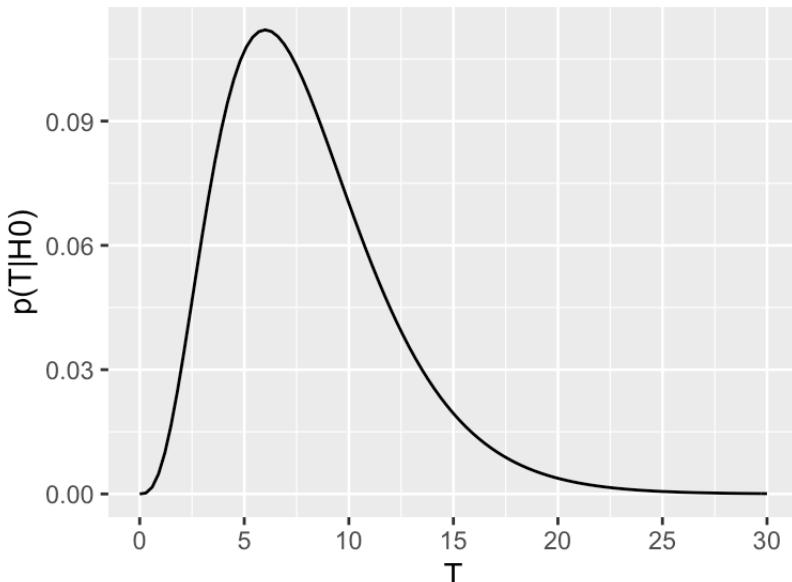
Under the null hypothesis H_0 , the test statistic T follows a certain distribution $p(T|H_0)$. The **P-value is the probability of obtaining a test statistic the same as or more extreme than the one we actually observed, under the assumption that the null hypothesis is true** (See Figure 7.3).

The formal definition of the P -value depends on whether we take “more extreme” to mean greater, less, or either way:

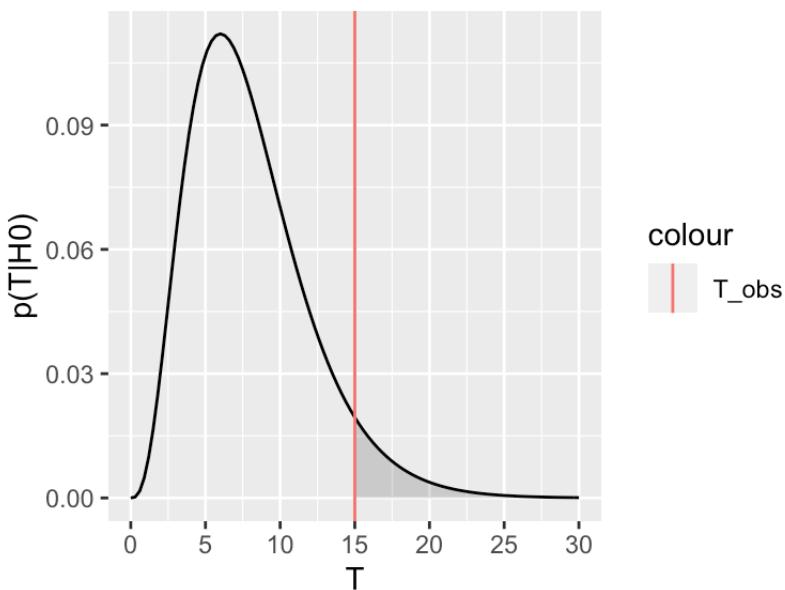
- For right-tail events: $P = p(T \geq T_{\text{obs}}|H_0)$
- For left-tail events: $P = p(T \leq T_{\text{obs}}|H_0)$
- For double tail events: $P = 2 \min\{p(T \leq T_{\text{obs}}|H_0), p(T \geq T_{\text{obs}}|H_0)\}$

The **null hypothesis is said to be rejected** for sufficiently small P -values. In this case we say the **result is statistically significant**. It is common practice in the scientific literature to set a **significance level of $\alpha = 0.05$** and **rejecting the null hypothesis if $P < \alpha$** .

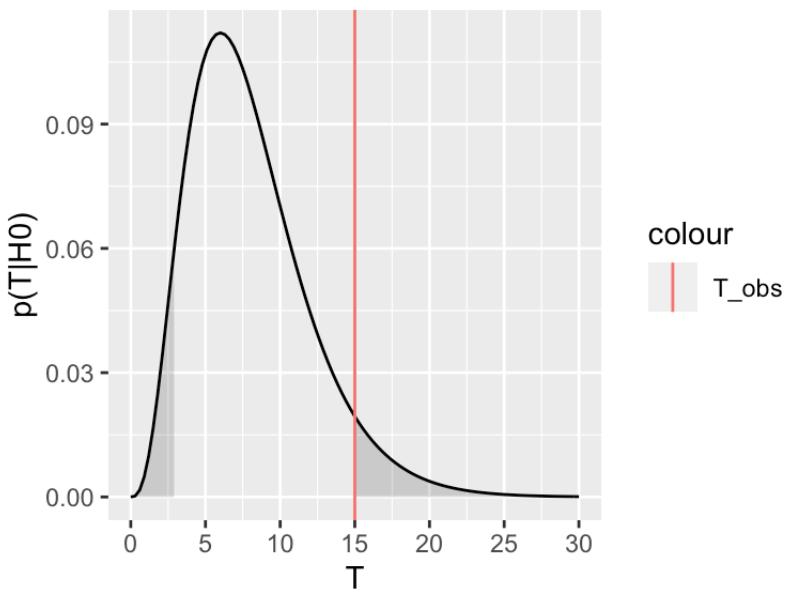
We can explore this definition visually. Assume $p(T|H_0)$, the distribution of the test statistic under the null hypothesis, looks like this:



Now assume the test statistic we observe is $T_{\text{obs}} = 15$. Then the one-sided P -value is given by the shaded area which corresponds to $p(T \geq T_{\text{obs}}|H_0)$:



For the two-tailed test, we are not expecting the test statistic to be on the upper or the lower side a priori. We therefore consider where it turned out to be (upper or lower) and double the probability. Graphically, we are summing up the area under the curve on the tail of the observed test statistic (upper or lower) with the equi-probable one of the other tail.



This also explains the rather complicated formulation for the two-sided P -value, which we recall is:

$$P = 2 \min\{p(T \leq T_{\text{obs}}|H_0), p(T \geq T_{\text{obs}}|H_0)\}$$

The \min in this formula is to select the correct tail. If our observed test-statistic is more towards the right tail, as in the above picture, then $p(T \geq T_{\text{obs}}|H_0)$ will be smaller than $p(T \leq T_{\text{obs}}|H_0)$, so the minimum will correctly select the right tail. Then we double this probability, to account for the possibility of observing equally extreme events on the other tail.

When to apply one-sided or two-sided tests?

Say we have as null hypothesis that the true growth rate difference is zero. There are two ways this can be violated: (1) if the true growth rate difference is > 0 or (2) if the true growth difference is < 0 . In a one-tailed test, we only account for one of these possibilities. We test H_0 : true difference is zero versus H_1 (alternative): true difference is > 0 , in the first

case. In a two-tailed test, we allow both options. We test H_0 : true difference is zero vs. H_1 (alternative): true difference is not zero, and we do not care if it ends up being smaller or larger. In most scenarios the two-tailed test will be most appropriate, as generally there is no reason to privilege effects in one direction over another direction. A one-tailed test will only make sense if you have very good reason (before looking at the data!) that only the effect in one direction is important.

7.2.2.4 Some intuition on Hypothesis Testing and the P-value

To get some intuition on how to interpret the P -value and this idea of rejecting the null hypothesis, think about a legal trial. Suppose a murder has been committed, and a man has been accused of being the murderer. Under German law he is considered innocent until proven guilty. So, our null hypothesis is that the man is innocent. But we also collect some evidence. For example, we discover that the murder weapon had his finger prints on it, that a witness saw him near the crime scene and that he bought chemicals used to dispose of corpses one day before the crime. None of these facts constitute hard proof that he did commit the crime, but assuming he was innocent, it would require a lot of unlikely coincidences. This corresponds to a scenario where the P -value is low. Thus, we reject the null hypothesis of innocence and convict him.

Conversely, imagine another trial, where the only evidence we have is that an old lady, who sees rather badly, thinks she maybe saw the accused near the crime scene. This corresponds to a scenario where the P -value is high. The accused could be guilty, but it also does not seem implausible that he is innocent and the old lady is just mistaking him for someone else. If we start convicting people based on such flimsy evidence, the jail would quickly be full of innocent people. So we do not reject the null hypothesis of innocence.

7.2.2.5 What the P-value is not

The P -value is not the probability of the observed test statistic given that the null hypothesis is true:

$$p(T \geq T_{\text{obs}} | H_0) \neq p(T = T_{\text{obs}} | H_0)$$

The problem with basing a test on $p(T = T_{\text{obs}} | H_0)$ is that it is dependent on the space of possibilities. This is most apparent for continuous variables: if T is continuous, then the probability of observing a specific value for the test-statistic, such as $T = 0.34257385692956$, will be zero (recall that, for continuous variables, probabilities are nonzero for intervals only). So $p(T = T_{\text{obs}} | H_0) = 0$ for all T , thus this would not give useful P -values.

Also, the P -value is not the probability that the null hypothesis is true given the data:

$$p(T \geq T_{\text{obs}} | H_0) \neq p(H_0 | T = T_{\text{obs}})$$

Consider again the example with the old lady witness. Surely we cannot convict someone of a murder on such weak evidence, thus we do not reject the null hypothesis of innocence. This being said, we also have no evidence to suggest the accused actually is innocent, so we should not conclude that this is definitely the case either! In other words: "absence of evidence is not evidence of absence".

Related to this, it is important to note the terminology we used above: when the P -value is less than the chosen significance level, we reject the null hypothesis. But, in this framework, there is no mechanism to accept the null hypothesis. We can only fail to reject it.

7.2.3 Permutation testing, formally

Formally, the strategy we implemented in Section (7.2.1) is a permutation test.

Generally a **permutation test** is used to test the statistical dependence between two variables x and y . In our example, we had one quantitative and one qualitative but they can be of any kind.

The **test statistics** can be any measure that **captures the dependence**.

We assumed that the observations are **identically and independently distributed** (i.i.d.). Denoting each observation (a row of the data table) (x_i, y_i) with $i = 1 \dots n$. The data generating process is the same for all observations (identically distributed). Moreover, the observations are independent. In particular the order of the indexing (the order of the rows of the data table) can be considered arbitrary.

The **i.i.d. assumption** is often taken in Hypothesis testing. It is however a **tricky one**. For instance if you have longitudinal data or **confounders** (hidden groups in the data). In our case, if the measurement of growth was done in separate day for the segregants of distinct genotypes, the i.i.d assumption could have not held. It is important in real applications to question this assumption, and if possible, to address it, for instance by **stratifying the data**.

The null hypothesis of a permutation test is that the two variables x and y are statistically independent:

$$H_0 : x \perp y$$

Hence, under the H_0 , the data generation process of x is independent of the one of y . Combined with the *i. i. d* assumption, this implies that the **values of x_i could have occurred in any other order with the very same likelihood**.

This gives us a mechanism to simulate data under the null (Figure 7.3).

An exact permutation test considers all possible distinct permutations (See next chapter). With a large number of observations as here ($n = 184$), we can also draw enough random permutations to have decent idea of the distribution of $p(T|H_0)$.

For a one-sided p-value we do:

- m be the number of random (Monte Carlo) permutations
- $r = \#\{T^* \geq T_{\text{obs}}\}$ be the number of these random permutations that produce a test statistic greater than or equal to that calculated for the actual data.

Then the estimated one-sided P-value, \hat{P} is (Davison and Hinkley 1997; Phipson and Smyth 2010)

$$\hat{P} = \frac{r + 1}{m + 1} \tag{7.1}$$

Permutation P-values should never be zero (Phipson and Smyth 2010). Do not use $\frac{r}{m}$ as often done!

In our case, we observed no single permutation with larger test statistics. Hence $r = 0$. We thus get:

$$\hat{P} = \frac{1}{1001} \simeq 0.01$$

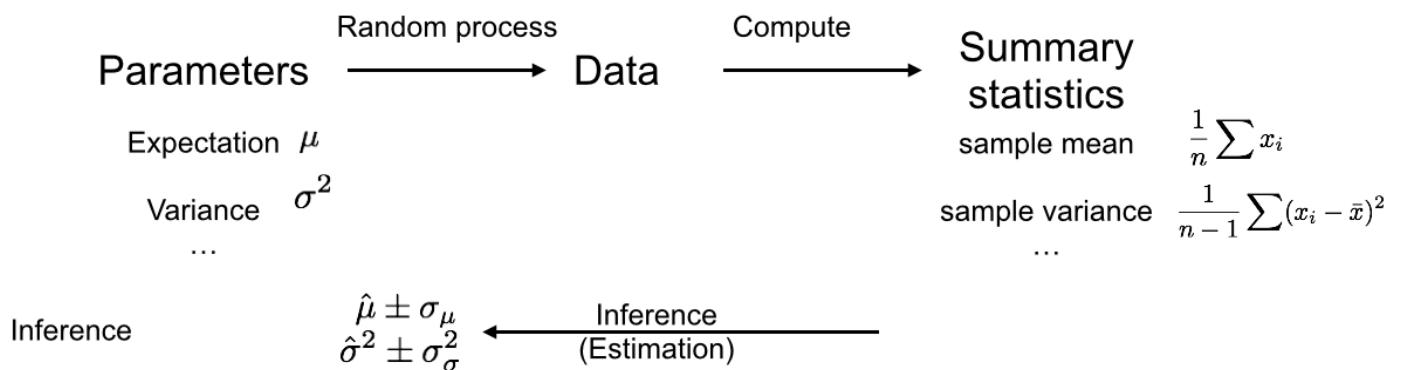
So, if we assume that the null hypothesis is true, the probability of observing a difference in median growth rates as, or more, extreme as the ones we actually observed, is less than one in one thousand. We would thus need to be quite unlucky to get results like this by chance. So, we reject the null hypothesis and we say that the association between genotype at marker 5211 and growth rates in maltose is statistically significant.

7.3 Confidence intervals: Quantifying uncertainty in parameter estimates

Hypothesis testing is a very effective way to guard us from being fooled by random patterns in our data. But it only answers a very specific question.

For the yeast example, we observed a difference of median growth rates of about 2.2 between yeast strains with different genotypes at marker 5211. Based on our permutation test, we rejected the null hypothesis that growth rate in Maltose is independent of the genotype at marker 5211. In other words, we concluded that the true difference in growth rates is unlikely to be zero. But does that mean that 2.2 is a good estimate of the true difference of median growth rates? How certain are we about this number?

We often face scenarios like this one, where we would like to estimate a certain quantity and report on the uncertainty of this estimate. This framework is called parameter estimation, and it is summarized in the following diagram:



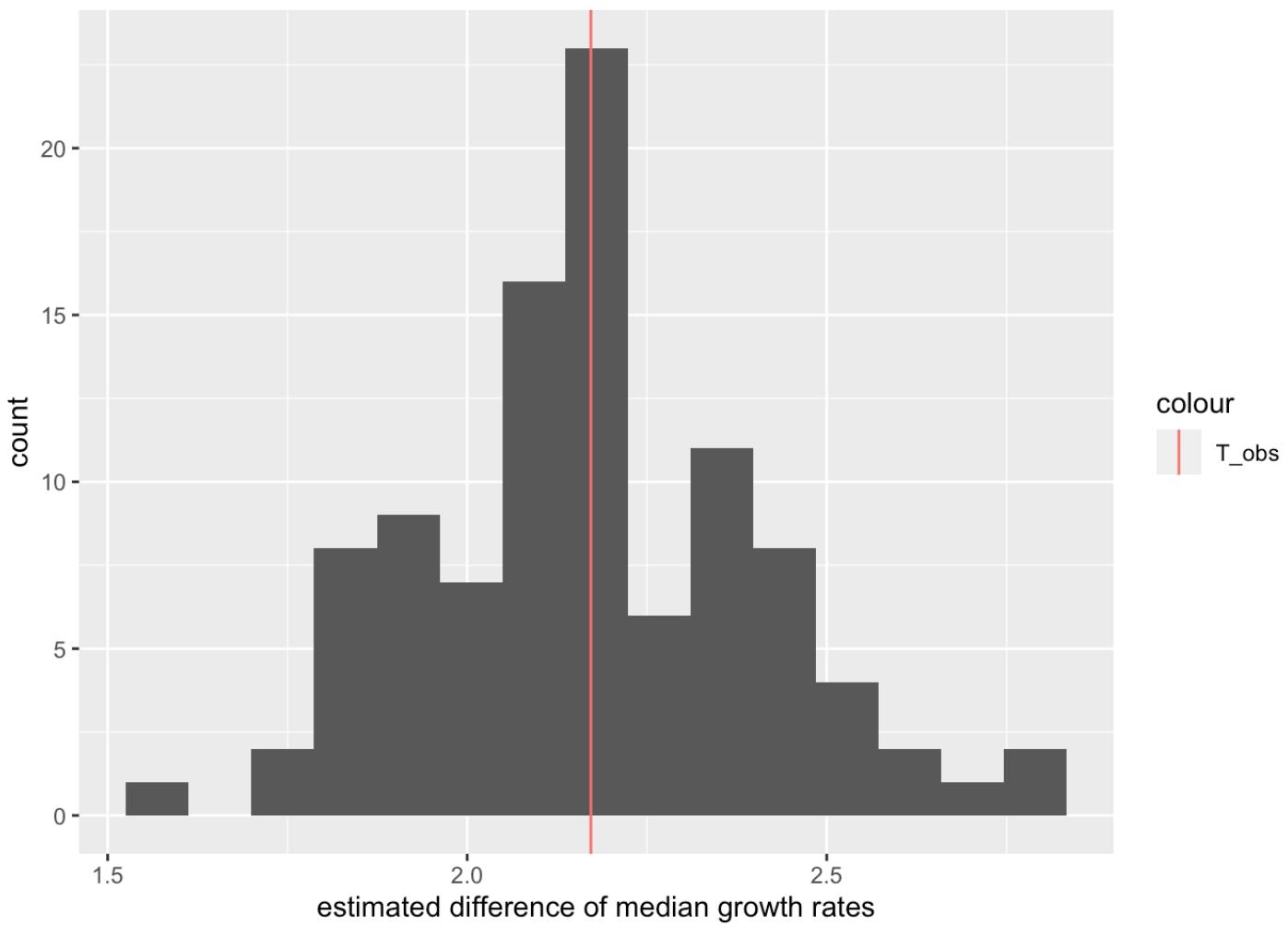
As before, there is a random process which produced our data, on which we compute summary statistics. But rather than just rejecting a null hypothesis, we now want to *infer* a parameter from our summary statistics, and also get an idea how precise our inference is. The **confidence interval is a method to quantify our uncertainty about a parameter estimate**. note that in this scenario we do not need ways of assessing the distribution under a null data generation process, but rather under the actual data generation process.

7.3.1 Repeating experiments to quantify uncertainty

We will first show the intuition behind the confidence interval and show how it can be computed in practice. Only then will we formally define it.

Imagine we have an unlimited budget and can repeat the entire yeast experiment 100 times. Every time we follow the same experimental protocol, and every time we compute the difference of median growth rates between yeast strains with different genotypes at marker 5211. This gives us a distribution of parameter estimates (note that, if the null hypothesis is true, this corresponds to the null distribution).

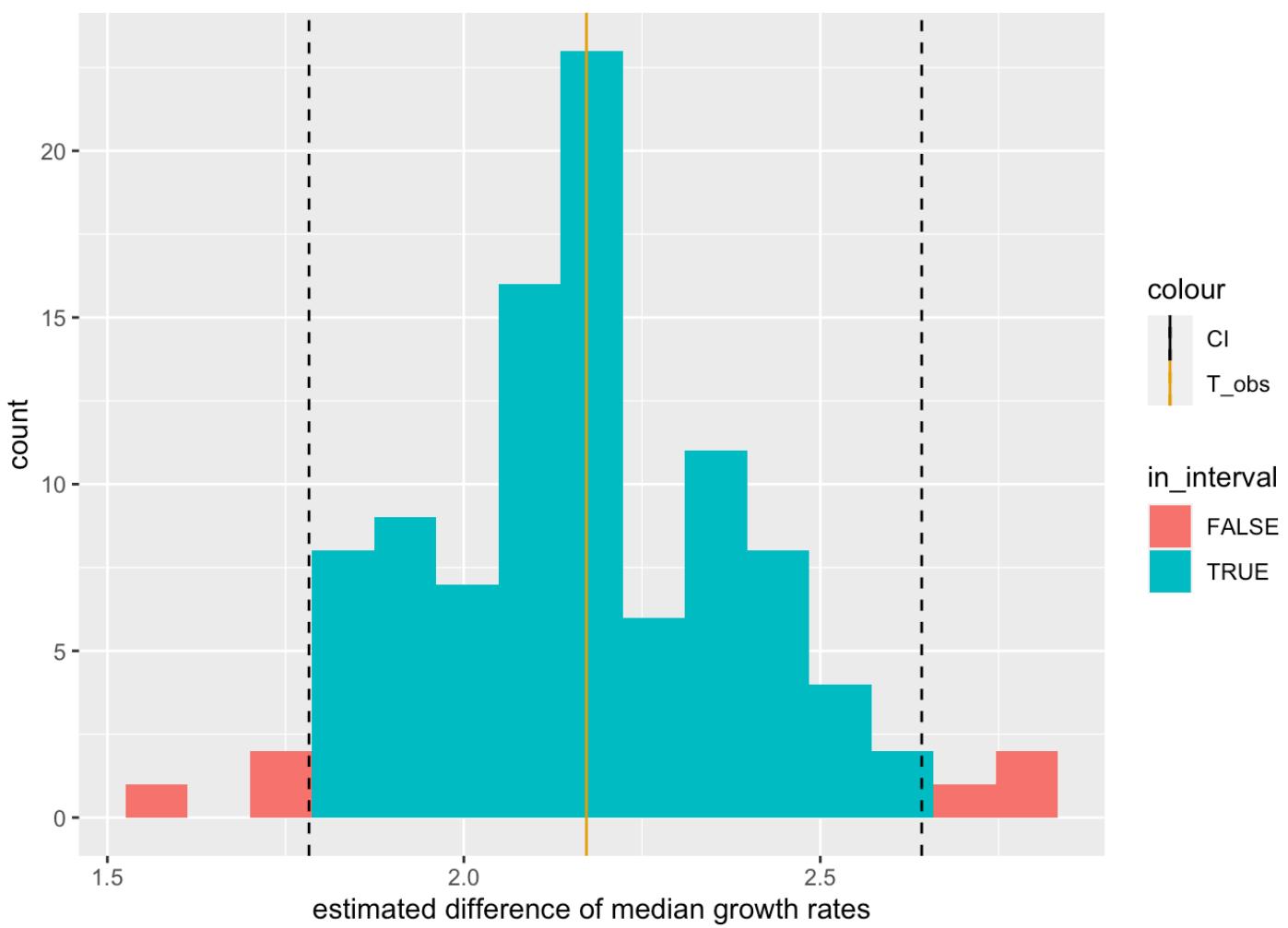
Assume we get the following distribution:



We see that many estimates are quite close to the one we measured in our first experiment. However, we also see that we get a range of results.

We do not know the true difference of medians, and in theory any of these estimates could be correct. So one way we could quantify our uncertainty is by reporting the full range of estimates we computed. However, this interval can quickly become very big. If we do the same experiment very often, it is quite plausible that we will have a bad day at some point and, for example, contaminate the samples, leading to an estimate that is very different than the others. We don't want the size of our interval to be entirely determined by one or two such outliers.

A more robust alternative is to report an interval that covers the central 95% of the values we got:



This interval thus covers the estimates derived from 95 of our 100 experiments, and only excludes the 5 most extreme ones. It seems very plausible that the true difference of medians, whatever it is, is somewhere in this interval, unless we got quite unlucky.

7.3.2 Simulating repeated experiments

The method of repeating experiments is a great way to quantify uncertainty. But in practice, we usually have to work with the data we actually have and cannot just rerun every experiment many times. This would be way too expensive and time consuming.

What we can do, however, is simulate reruns of the experiment by sampling from the data we already have. To this end, the concept of a cumulative distribution function will be useful.

7.3.2.1 Empirical distribution

Consider a random variable X and a random sample of n independent realizations drawn from it: $\{x_1, x_2, \dots, x_n\}$. The empirical distribution is the distribution that gives equal probability to each of these observations.

A single random draw from the empirical distribution amounts to picking one data point with probability $\frac{1}{n}$. Independent random draws of any size m are thus equivalent to sampling with replacement.

In R this is obtained using the `sample` function:

```
set.seed(10)
x <- rnorm(20) # 20 random numbers normally distributed
xrnd <- sample(x, 15, replace = TRUE) # 15 random draws from the data in x
xrnd

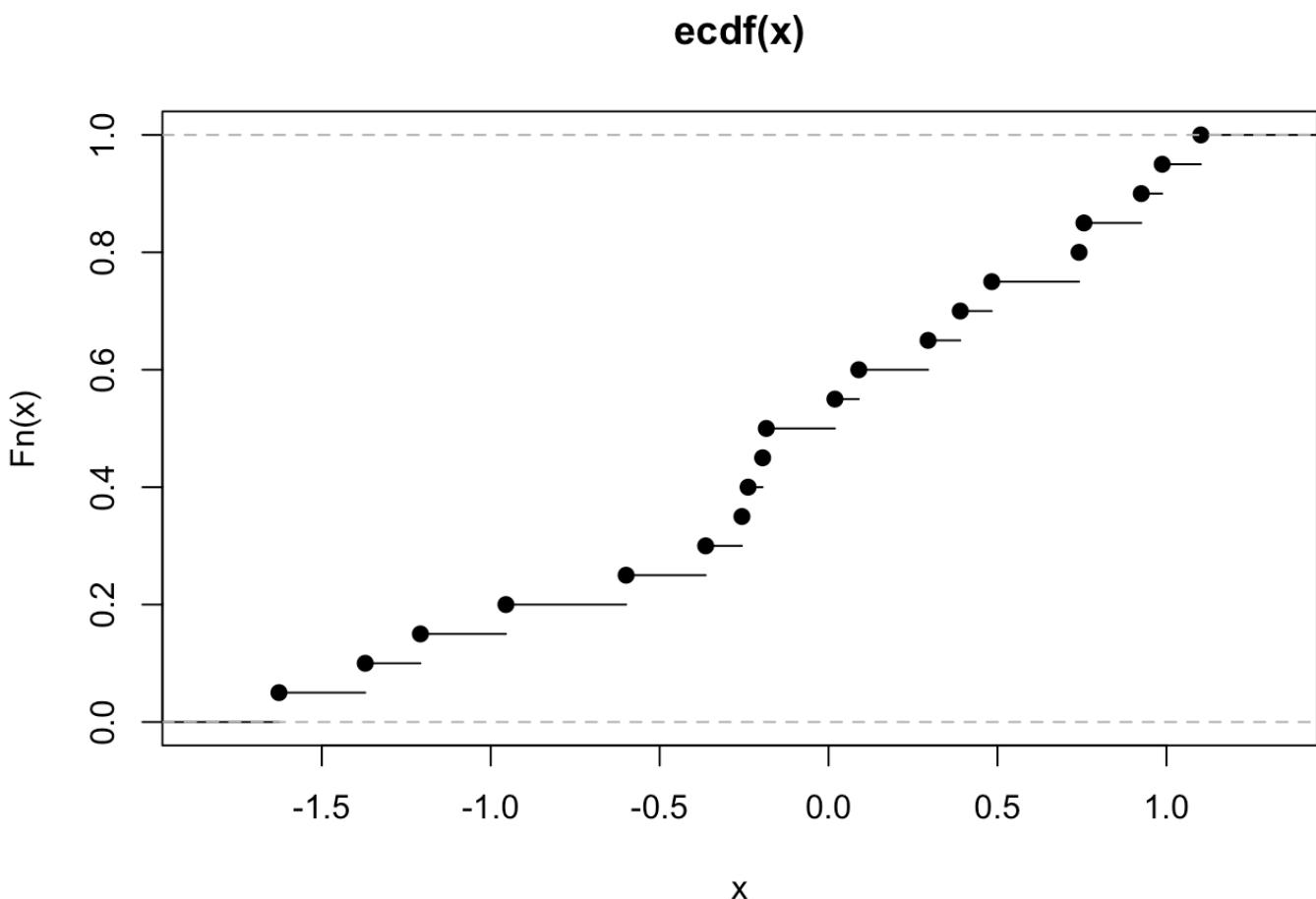
## [1] -1.20807618 -0.19515038 -0.59916772 -0.19515038
## [5]  0.92552126 -0.19515038  1.10177950  0.74139013
## [9]  0.01874617 -0.25647839 -0.25647839  0.98744470
## [13] -0.23823356  0.01874617 -1.62667268
```

A fundamental result is that the empirical distribution converges to the underlying distribution. This is best seen when considering cumulative distribution function. The empirical cumulative distribution function (eCDF) is a step function that jumps up by $1/n$ at each of the n data points.

$$F_n(x) = \frac{1}{n} \sum_{i=1..n} I_{x_i \leq x}$$

In R, it is obtained by `ecdf()`.

```
plot(ecdf(x))
```



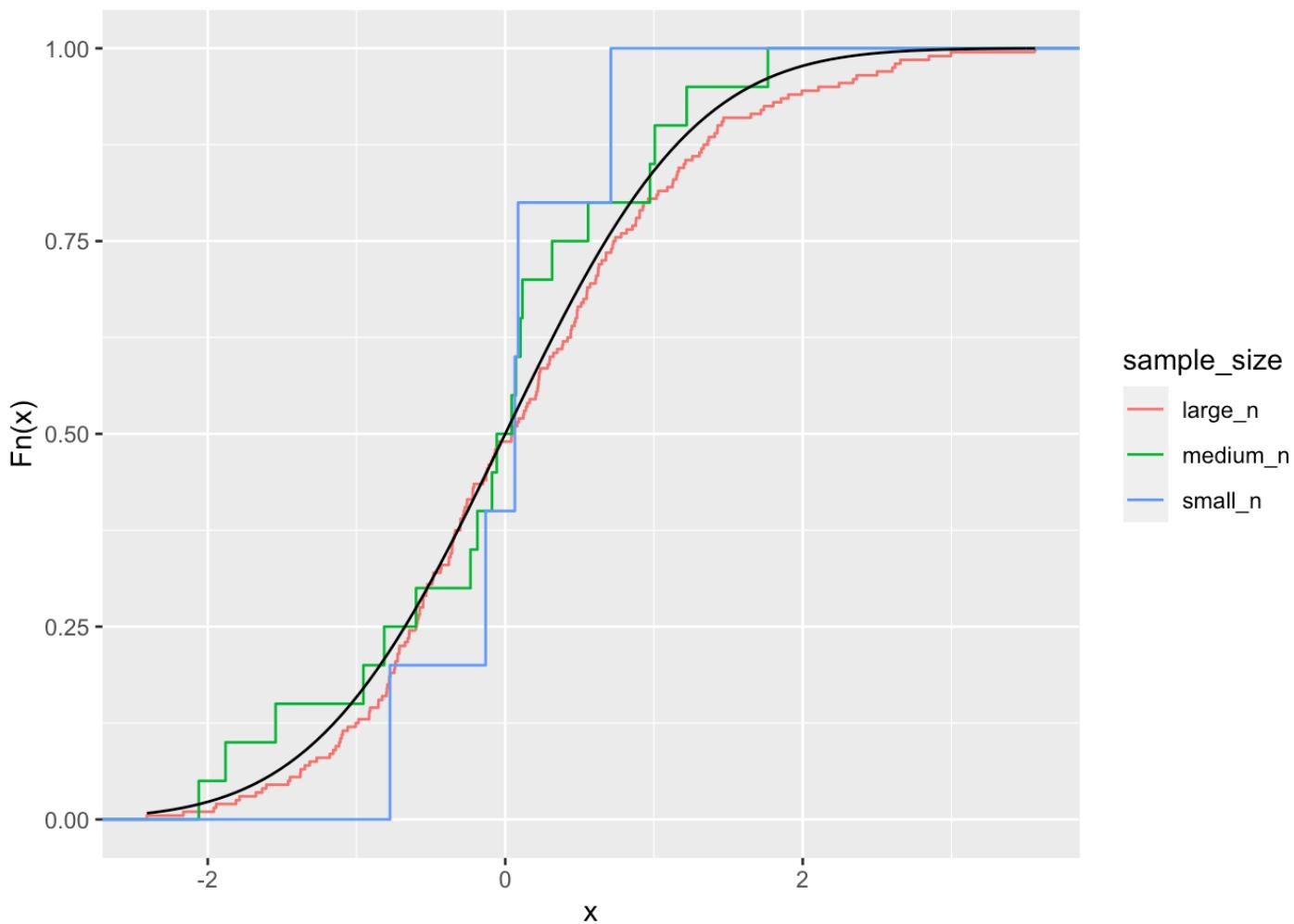
The eCDF tells us, for any value x , the proportion of observations less than or equal to x . For example, from the eCDF above, we see that about half of the observations are less than or equal to 0. This is equivalent to say that the eCDF tells us, for any value x , the probability of one randomly picked observation among $\{x_1, x_2, \dots, x_n\}$ to be less than or equal to x . Hence, the eCDF is nothing else than the cumulative distribution of the process of randomly drawing from $\{x_1, x_2, \dots, x_n\}$.

The empirical distribution function converges almost surely to the distribution function of X . This means that as n goes to infinity, the empirical distribution and the actual distribution will become more and more alike:

```

x_small <- rnorm(5)
x_middle <- rnorm(20)
x_big <- rnorm(200)
x_lbl <- c(rep("small_n", 5), rep("medium_n", 20), rep("large_n", 200))
x_combined <- c(x_small, x_middle, x_big)
dt_ecdf <- data.table(sample_size = x_lbl, x = x_combined)
ggplot(dt_ecdf) + stat_ecdf(aes(x, colour = sample_size)) +
  stat_function(fun = pnorm) +
  ylab("Fn(x)")

```



The implication is that drawing from the empirical distribution is a justified proxy for drawing from the actual underlying distribution. It is more accurate with large sample sizes.

7.3.2.2 Case resampling bootstrap

Using this idea of drawing from the empirical distribution function, we can simulate experiments. After all, an experiment is like drawing from the true distribution, so if our **empirical distribution** is close enough to the true distribution, then drawing from it is comparable to doing a new experiment.

Concretely, we take a sample of size n , with replacement, from our observed data, to make a new dataset. This is called the case resampling bootstrap.

Of course, this “new” data will resemble the old data. But, provided that n is not extremely small, it will almost certainly not be the same. This is because we are sampling with replacement, meaning that we will select some data points several times, and other points may not be selected at all.

Let us perform one bootstrap for the yeast data and recompute the difference of median growth rates:

```
dt_resampled <-
  dt[sample(nrow(dt), replace = TRUE)]
diff_median(dt_resampled)
```

```
## [1] 2.666546
```

As we see, this value is indeed somewhat different from the one we computed from our original sample.

7.3.3 Quantifying uncertainty using the case resampling bootstrap

Now let us do R random simulations of the data by case resampling. Each gives a random value for the parameter denoted T_i^* . Let's rank them by increasing order and denote them:

$$T_1^* \leq T_2^* \leq \dots \leq T_R^*$$

We can do this for our yeast data and visualize the result as a histogram of parameter estimates.

```
# number of random simulations
R <- 1000

# initialize T_boot with missing values
# (safer than with 0's)
T_bootstrap <- rep(NA, 1000)

# iterate for i=1 to R
for(i in 1:R){
  # sample the original data with same size with replacement
  dt_boot <- dt[sample(nrow(dt), replace=TRUE)]
  # store the difference of medians in the i-th entry of T_permuted
  T_bootstrap[i] <- diff_median(dt_boot)
}
```

The 95% bootstrap percentile confidence interval can now be obtained using the quantiles.

More concretely, we use the same idea as we had in the beginning when we actually repeated experiments: we again try to cover the central $(1 - \alpha) * 100\%$ of the distribution of estimates, where we can choose $(1 - \alpha) * 100\%$ to be bigger or smaller to depending on how conservative we want to be.

This is achieved by using the interval:

$$(T_{(R+1)\alpha/2}^*, T_{(R+1)(1-\alpha/2)}^*)$$

To get a concrete feeling of what this means, assume $R = 99$ and $\alpha = 0.1$. Then, $(R + 1)\alpha/2 = 5$ and $(R + 1)(1 - \alpha/2) = 95$. Thus our 90% interval is given by (T_5^*, T_{95}^*) , i.e. the interval ranging from the 5-th smallest bootstrap parameter estimate to the 95-th smallest bootstrap parameter estimate. It is nothing else than the interval containing 90% of the bootstrap estimates and with equal fraction of the remaining bootstrap estimates on either side (so-called “equi-tailed”).

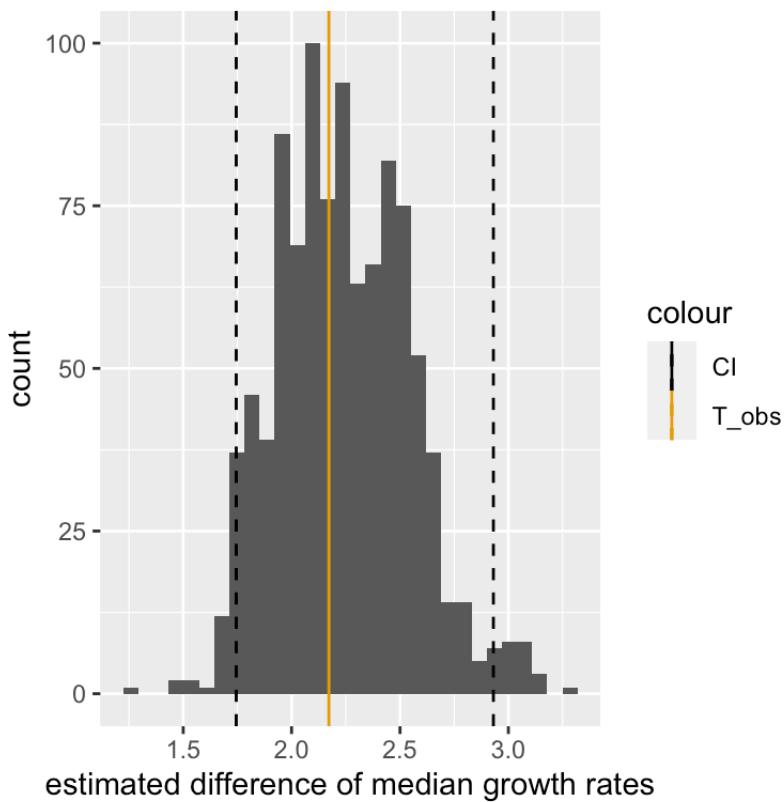
In R, we can use the quantile function to compute this. For $(1 - \alpha) * 100\% = 95\%$ interval, we do:

```
conf_int <- quantile(T_bootstrap, c(0.025, 0.975))
conf_int

##      2.5%    97.5%
## 1.74471 2.93088
```

The following plot shows the entire distribution along with the observed value and the 95% bootstrap percentile confidence interval.

```
ggplot(data.table(T_bootstrap), aes(T_bootstrap)) +
  geom_histogram(bins=30) +
  geom_vline(aes(xintercept=T_obs, color = "T_obs")) +
  geom_vline(aes(xintercept=conf_int[1], color="CI"), linetype="dashed") +
  geom_vline(aes(xintercept=conf_int[2], color="CI"), linetype="dashed") +
  scale_color_manual(values=cbPalette) +
  xlab("estimated difference of median growth rates")
```



7.3.4 Confidence Intervals: Formal definition

Let us now define a confidence interval formally.

A **confidence interval** of confidence level $1 - \alpha$ for a parameter θ is an interval $C = (a, b)$, which would the data generation process be repeated, would contain the parameter with probability $1 - \alpha$, i.e. $p(\theta \in C) = 1 - \alpha$. A typical value is $\alpha = 0.05$ which leads to 95% confidence intervals.

Note that a and b are functions of the data and thus C is the random variable here, not θ !

To get some intuition for this, consider again the scenario where we repeat the yeast experiment 100 times. But instead of computing one 95% interval from all the experiments, we instead compute a separate 95% interval for each of the experiments (using, for example, the case resampling bootstrap). Then the true difference of medians, whatever it is, will be contained in about 95 of the computed intervals. In other words, it will happen relatively rarely (about 5% of the time) that we get an interval that happens not to include the true difference of medians.

But note carefully what this means. It means that before we do an experiment, we have a 95% chance to end up with an interval that contains the true value. It does not mean that the specific interval we compute after the experiment has been done has a 95% chance of including the true value. This statement would not even make sense. The true value is a fixed number, so either it is in the interval, or it is not. There is no notion of probability there.

It should be noted that the $(1 - \alpha)$ case resampling bootstrap interval is only an approximate $(1 - \alpha)$ confidence interval. This means it does not guarantee that $p(\theta \in C) = 1 - \alpha$, but only that $p(\theta \in C) \approx 1 - \alpha$.

There are other ways to compute confidence intervals, which usually require making further assumptions, such as that the data is normally distributed. See Davison AC, Hinkley DV (1997) for an overview.

7.3.5 Visualizing the formal definition of Confidence Intervals

To visualize the meaning of this definition, we will now consider an example where we know the true value of the parameter. Specifically we assume that we are trying to use the sample mean as estimate of the true mean of a standard normal distribution. Thus, the true mean is zero.

Firstly, let us run the experiment:

```
set.seed(100)
exp_1 <- rnorm(30) # original data (30 values drawn from the standard normal distribution)
```

Now we compute the sample mean and do a bootstrap resampling:

```
# Compute observed sample mean
observed_mean <- mean(exp_1)

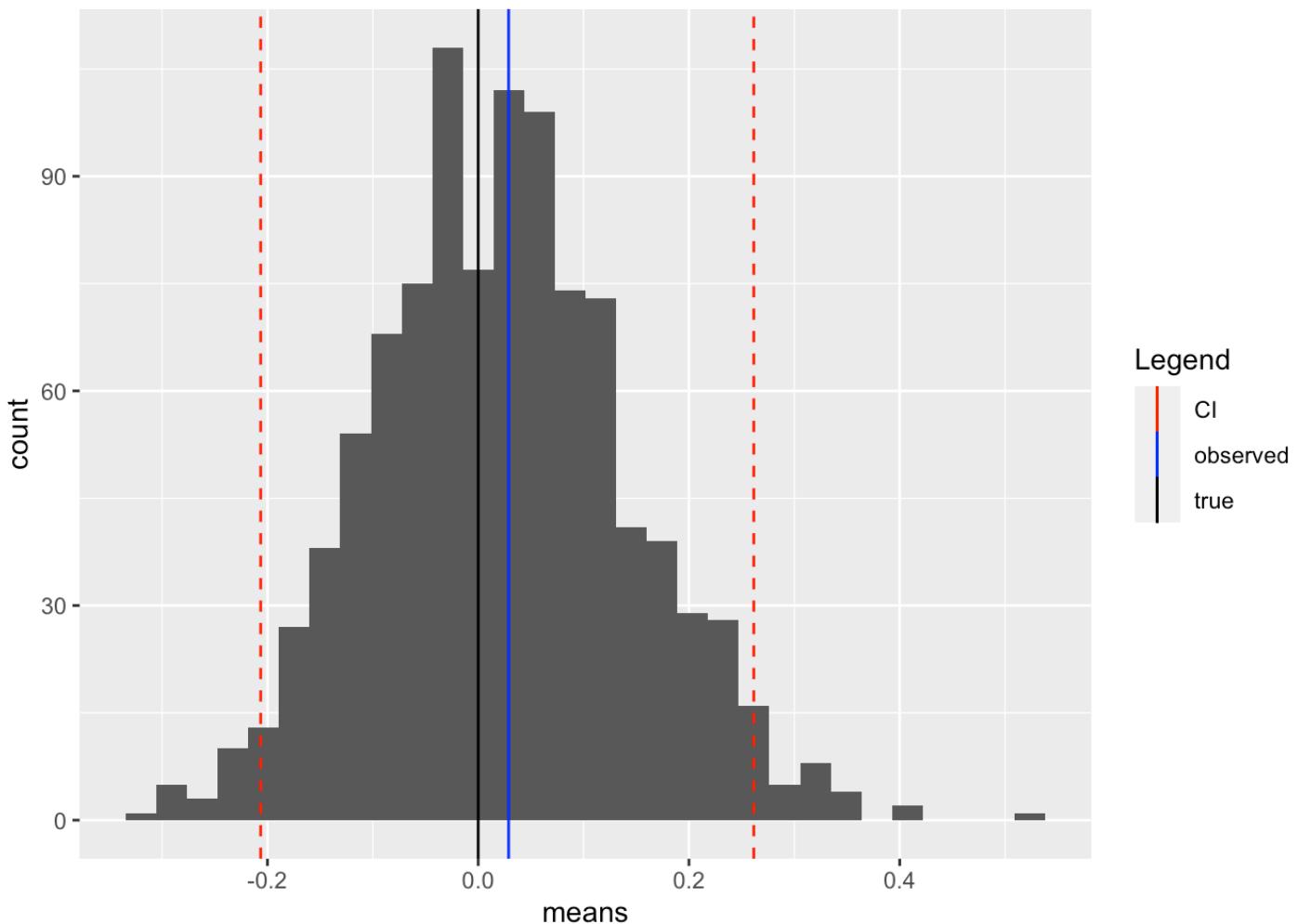
# Do bootstrap and compute sample mean for each simulation
boot <- lapply(1:1000, function(i){sample(exp_1, 30, replace = TRUE)})
sample_means <- sapply(boot, mean)
```

This creates a distribution of estimates. We build our 95% case resampling bootstrap confidence interval:

```
# 95% C.I. is given by the 2.5% and the 97.5% quantile
conf_int = quantile(sample_means, c(0.025, 0.975))

# Plot histogram
bootstrap_tbl = data.table(means = sample_means)
ggplot(data = bootstrap_tbl, aes(x = means)) +
  geom_histogram() +
  geom_vline(aes(xintercept=observed_mean, color="observed")) +
  geom_vline(aes(xintercept=0, color="true")) +
  geom_vline(aes(xintercept=conf_int[1], color="CI"), linetype="dashed") +
  geom_vline(aes(xintercept=conf_int[2], color="CI"), linetype="dashed") +
  scale_color_manual(name = "Legend", values = c(true="black", observed = "blue", CI = "red"))

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



We see that our interval covers all but the most extreme estimates (the tails of the distribution). It also covers the true value, which is slightly lower than the observed value.

It now remains to show that this interval keeps what it promises, namely that we capture the true value about 95% of the time if we repeat the “experiment”.

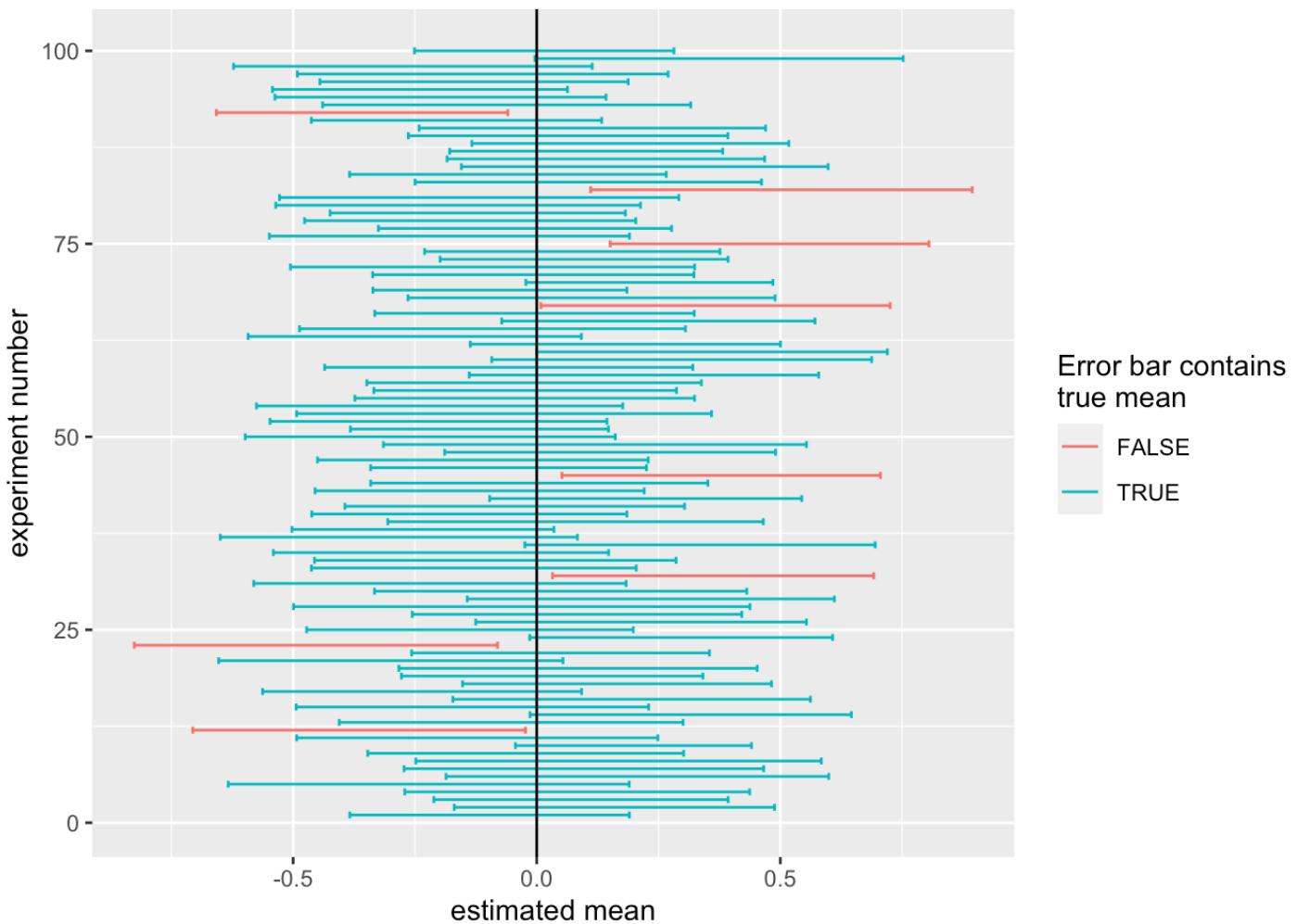
```

rerun_experiment <- function(j) {
  exp <- rnorm(30)
  boot <- lapply(1:1000, function(i){sample(exp, 30, replace = TRUE)})
  sample_means <- sapply(boot, mean)
  conf_int = quantile(sample_means, c(0.025, 0.975))
  return(conf_int)
}

rerun <- sapply(1:100, rerun_experiment)

intervals <- data.table(t(rerun))
intervals[, idx := 1:100]
intervals[, mid := ('97.5%' + '2.5%')/2]
intervals[, contains_true := (('97.5%' >= 0) & ('2.5%' <= 0))]
ggplot(data = intervals, aes(mid, idx, color=contains_true)) +
  geom_errorbar(aes(xmin='2.5%', xmax='97.5%')) +
  geom_vline(aes(xintercept=0)) + xlab("estimated mean") + ylab("experiment number") +
  labs(color

```



This plot shows for each repetition of the experiment the confidence interval we computed. The true mean is at 0, but in every experiment, the estimated mean (or sample mean) and deviates from the true mean. The boundaries of the 95% confidence intervals as well are random variables. They fluctuate from experiment to experiment. We want these confidence intervals to contain the true mean 95% of the time. The confidence intervals which contain the true mean (0, marked by the black line) are in blue, those that do not are in red. We see that most of the time, our interval does indeed capture the true value.

In fact we capture the true value 92% of the time. This is slightly worse than what we expected, but that is not too surprising because the simulation procedure we have used above is approximate. If we use more bootstrap samples, and replicate our experiment more often, we will reach the 95%.

7.3.6 Hypothesis testing with the Confidence Interval

It is relatively common in the scientific literature to perform hypothesis tests using the confidence interval. If our null hypothesis is that a given parameter, e.g. a mean, is zero, and our $(1 - \alpha) * 100\%$ confidence interval for this parameter does not include zero, we could say that we reject the null hypothesis at a significance level of α . In that sense, hypothesis tests and confidence intervals are related.

This being said, in this chapter we have used approximate methods to compute P -values and confidence intervals. Thus it need not necessarily be the case that if one of them rejects, the other will too (although, most of the time, they should agree).

In analyses where two groups are compared, as in our yeast example where median growth rates are compared between genotypes, people will often use a different procedure to test hypotheses using the confidence interval. In this procedure, we construct a 95% confidence interval for the median growth rate of each genotype separately. We then reject the null hypothesis if and only if the confidence intervals do not overlap.

It is important to note that this is *not* the same as rejecting if and only if the confidence interval for the difference of medians does not include zero, even if it may seem so intuitively. In fact, this “overlap” procedure is too conservative, and will fail to reject more often than the confidence level suggests. In the next chapter, we will give a technical reason for this.

7.4 Discussion

In this chapter we explored ways to avoid being fooled by randomness. Specifically, we discussed hypothesis testing, which is our go-to method to distinguish statistically significant results from noise, and we discussed confidence intervals, which help us to know how uncertain we are about quantities we estimate from our data.

Cares should be taken though. Many misuses of p-values in the scientific literature have been reported, the most obvious being to repeat an experiments until one finally gets $P < 0.05$ and only report this observation in a publication. See Wasserstein and Lazar (2016) for an extensive discussion.

We have looked only at resampling strategies: permutation testing as Hypothesis testing when assessing statistical dependence of variables, and case resampling for confidence intervals. Resampling methods have the advantage that they are simple to implement and make little assumptions about the underlying distribution. However, they are compute intensive. We finally noticed that the i.i.d assumption may be violated in practice. Therefore, careful thinking of possible hidden dependencies (such as confounders) shall be done when applying these methods.

7.5 Conclusion

By now you should be able to:

- Understand what we mean when we say a result is statistically significant
- Understand the terms test statistic, null hypothesis, P -value and confidence interval and explain their purpose
- Understand that the P -value is *not* the probability that the null hypothesis is true
- Use permutation to perform Hypothesis testing of associations
- Use case resampling to compute bootstrap confidence intervals

References

Davison, A. C., and D. V. Hinkley. 1997. *Bootstrap Methods and Their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press. <https://doi.org/10.1017/CBO9780511802843>.

Gagneur, Julien, Oliver Stegle, Chenchen Zhu, Petra Jakob, Manu M. Tekkedil, Raeka S. Aiyar, Ann-Kathrin Schuon, Dana Pe'er, and Lars M. Steinmetz. 2013. "Genotype-Environment Interactions Reveal Causal Pathways That Mediate Genetic Effects on Phenotype." *PLOS Genetics* 9 (9): 1–10. <https://doi.org/10.1371/journal.pgen.1003803>.

Phipson, Belinda, and Gordon K Smyth. 2010. "Permutation P-Values Should Never Be Zero: Calculating Exact P-Values When Permutations Are Randomly Drawn." *Statistical Applications in Genetics and Molecular Biology* 9 (1).

<https://doi.org/https://doi.org/10.2202/1544-6115.1585>.

Wasserstein, Ronald L., and Nicole A. Lazar. 2016. "The Asa Statement on P-Values: Context, Process, and Purpose." *The American Statistician* 70 (2): 129–33. <https://doi.org/10.1080/00031305.2016.1154108>.

13. https://en.wikipedia.org/wiki/Chromosomal_crossover

Chapter 8 Analytical Statistical Assessment

In the last chapter we discussed how trends in data can arise by chance, leading us to wrong conclusions. We saw that statistical hypothesis testing can help to guard us from being fooled by randomness in this way. We developed the permutation test as an empirical way to perform hypothesis tests.

While permutation testing is very general and requires few assumptions, it has its limitations, as we will see shortly. In this chapter, we will therefore discuss a more analytical approach to testing. We will see several classical tests, such as the binomial test and the t-test. These tests often make stronger assumptions about the underlying data. Thus, it is important to understand when they can and cannot be used. We will discuss the quantile-quantile plot (Q-Q plot) as a method to check some of these assumptions.

In the last chapter we also touched on the topic of confidence intervals, which help us quantify the uncertainty of our estimates, and also developed an empirical way to compute them. In this chapter, we will briefly describe how to compute them analytically.

8.1 Motivation: Hypothesis testing in large datasets

We have already discussed how to test specific hypotheses, for instance considering the association between a genetic marker and growth rate in the yeast dataset (See Chapter 7). However, in the era of big data, we often do not restrict ourselves to testing just one single hypothesis. Molecular biologists can nowadays measure RNA abundance of all genes of a cell population. So what if we test the association of the RNA abundance of all ~8,000 yeast genes with every single genetic markers? For 1,000 genetic markers, this means we will have to do more than 8 million tests!

Doing this many tests can lead to misleading results. Let us assume, for the sake of argument, that our null hypothesis is always true and there is never an association between RNA abundance and markers. If we reject this null hypothesis every time we observe $P \leq 0.05$, we will falsely reject the null hypothesis in roughly 5% of the tests we do. With 8 million tests, we will then falsely reject the null hypothesis 400,000 times.

This issue is called **multiple testing** and strategies to deal with this problem will be discussed in detail in Chapter 9. For now, it suffices to say that when we do many tests, we will usually require far lower P -values to reject the null hypothesis, to guard against the problem described above.

With permutation testing, we estimated P -values using $P = \frac{r+1}{m+1}$, where m is the number of permutations (Equation (7.1)). It follows that, with this method, the P -values we can compute will never be smaller than $\frac{1}{m+1}$. If we now say that we will only reject the null hypothesis if, for example, $P \leq 0.001$, then we will need at least $m = 1,000$ permutations, otherwise our test simply cannot reject the null hypothesis, regardless of the true associations in the data. Since we are doing 8 million tests, this means we will end up doing more than 8 billion permutations. Hence, permutation testing can

become very costly in terms of computing power and time. We thus require more scalable ways to estimate P -values for large datasets. This Chapter provides methods for which P -values are computed from the observed test statistics directly.

8.2 The Binomial Test: testing hypotheses for a single binary variable

The first test we will look at is the binomial test. We use it when we want to test hypotheses concerning one binary variable.

8.2.1 Abstraction: Tossing a coin

To develop the binomial test, we consider an abstract example, namely testing whether a coin is biased.

We first introduce some notation. Let:

- n : the total number of independent random tosses of the coin.
- X_i : the value of the i -th toss, with $X_i = 1$ if the result is head and $X_i = 0$ if it is tail.
- $\mu = E(X_i) = p(X_i = 1)$: the probability of getting a head.

We assume the X_i to be i.i.d.

As we will see again and again in this chapter, to develop a statistical test we require three ingredients:

- A null hypothesis H_0 (and a suitable alternative hypothesis H_1 , either one or two-sided)
- A test statistic T
- The distribution of this test statistic under the null hypothesis, $p(T|H_0)$

(Note that in permutation testing the sampling procedure simulated the distribution of our test statistic under the null hypothesis.)

To test whether a coin is biased, our null hypothesis is that the coin is fair:

$$H_0 : \mu = 0.5$$

And our alternative is that it is biased (either towards heads or tails):

$$H_1 : \mu \neq 0.5$$

As test statistic, we will use the total number of heads, i.e. $T = \sum_i X_i$.

8.2.1.1 A single coin toss

Now assume, for the sake of argument, we toss the coin only once ($n = 1$) and get a head ($T_{\text{obs}} = 1$). What is the two-sided P -value in this case?

In this scenario, there are of course only 2 possible outcomes. Either we get one head or we get one tail. Under the null hypothesis, both outcomes are equally likely. Therefore, the distribution of the test statistic under the null hypothesis is given by:

$$p(T = 0|H_0) = 0.5 = p(T = 1|H_0)$$

The two-sided P -value is then given by:

$$\begin{aligned} P &= 2 \min\{p(T \leq T_{\text{obs}}|H_0), p(T \geq T_{\text{obs}}|H_0)\} \\ &= 2 \times 0.5 \\ &= 1 \end{aligned}$$

Thus, if we only performed a single coin toss, the data cannot provide sufficient evidence for rejecting the null hypothesis in a two-sided test. This, of course, does not allow us to conclude that the null hypothesis is correct. In particular, we cannot write that " $p(H_0) = p(\mu = 0.5) = 1$ ". After all, we could have generated the same data with a coin that has heads on both sides, for which the null hypothesis clearly does not hold. Another way to look at it, is to state that the data does not provide sufficient evidence to conclude that the coin is biased.

8.2.1.2 Tossing the coin several times

Now assume we toss the coin $n > 1$ times and observe T_{obs} heads. What is the distribution of the test statistic under the null hypothesis now?

We can easily simulate data under this assumption in R by sampling with replacement from a vector `c(0, 1)`. The probability for each outcome can be provided with the `prob` argument. Here is one such simulation of $n = 10$ trials under the null hypothesis $\mu = 0.5$.

```
# set.seed is optional
# we just pick an arbitrary seed of the random number generator to ensure reproducibility
# See https://en.wikipedia.org/wiki/Random_number_generation
set.seed(7)
n <- 10
x <- sample(c(0,1), n, replace=TRUE, prob=c(0.5,0.5))
x

## [1] 0 1 1 1 1 0 1 0 1 1
```

Our test statistic T is the sum of heads:

```
t <- sum(x)
t

## [1] 7
```

The probability of observing T heads after tossing a coin n times is given by the binomial distribution, which is the binomial coefficient, i.e. the number of possible sequences of events with the same total number of heads, times the probability of a given sequence, which is itself the product of the probability of each individual realization (i.i.d. assumption):

$$p(T|n, \mu) = \binom{n}{T} \mu^T (1 - \mu)^{n-T}$$

Hence, assuming the null distribution is true, we get that:

$$p(T|n, \mu = 0.5) = \binom{n}{T} 0.5^n$$

This is implemented in R with the function `dbinom`¹⁴. The probability to have observed exactly 7 heads is therefore:

```
dbinom(x=t, size=n, prob=0.5)
```

```
## [1] 0.1171875
```

We recall that the P -value is defined as the probability, under the null hypothesis, of observing a test statistic as or more extreme as the one we actually observed. Since we just want to know *whether* the coin is biased, and do not care in which direction, we need a two sided p-value. This is given by (Figure 8.1):

$$\begin{aligned} P &= 2 \min\{p(T \leq T_{\text{obs}}|H_0), p(T \geq T_{\text{obs}}|H_0)\} \\ &= 2 \min\left\{\sum_{T \leq T_{\text{obs}}} \binom{n}{T} 0.5^n, \sum_{T \geq T_{\text{obs}}} \binom{n}{T} 0.5^n\right\} \end{aligned} \quad (8.1)$$

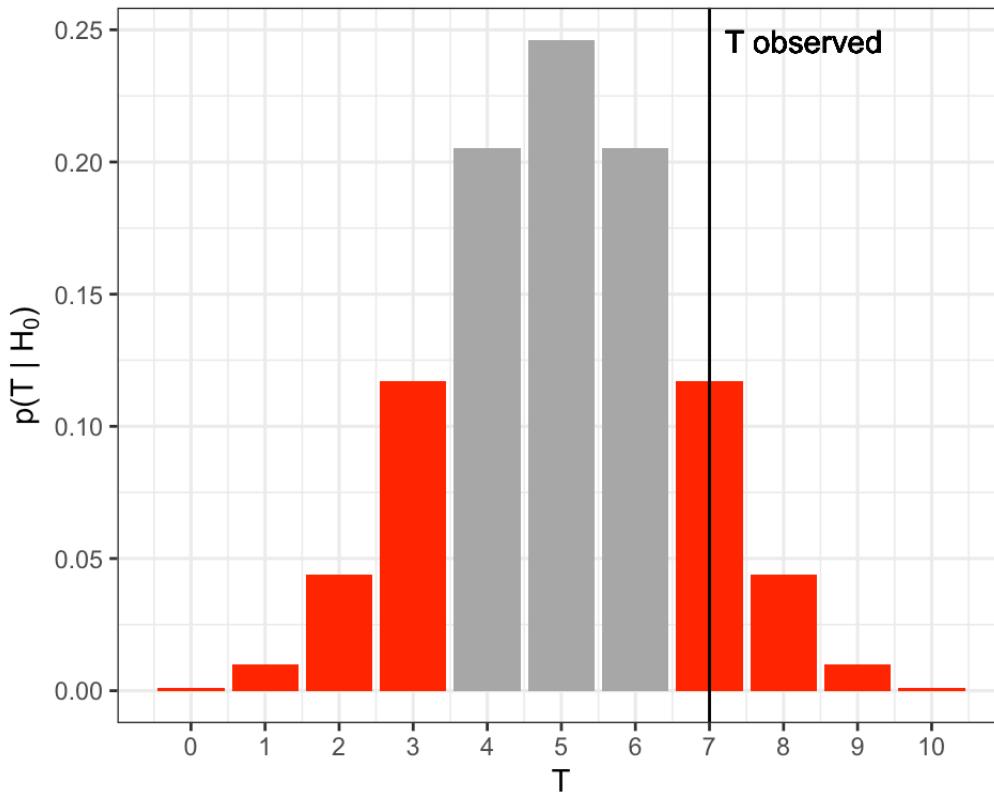


Figure 8.1: Two-sided p-value for the coin tossing example. The two-sided p-value equals to the sum of the probabilities (total red area) under the null hypothesis of the realizations equal or more extreme than the observed one (vertical line).

To apply the formula (8.1), one can compute the smaller of the two terms which is here $\sum_{T \geq 7} \binom{10}{T} 0.5^{10}$, and corresponds to the right tail marked in red in Figure 8.1. This is $1 - \sum_{T \leq 6} \binom{10}{T} 0.5^{10}$ and is obtained in R with:

```
1-pbinom(q=t-1, size=n, prob=0.5)
```

```
## [1] 0.171875
```

Hence the two-sided p-value is twice this value, yielding:

```
2*(1-pbinom(q=t-1, size=n, prob=0.5))
```

```
## [1] 0.34375
```

Altogether we have $P = 0.34375$. We do not reject the null hypothesis that the coin is fair at a significance level of 0.05.

8.2.2 Computing a binomial test with R

In actual applications, we use the `binom.test` function of R. For the example above, we do:

```
binom.test(t, n, p = 0.5, alternative = c("two.sided"))

##
## Exact binomial test
##
## data: t and n
## number of successes = 7, number of trials = 10,
## p-value = 0.3438
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
## 0.3475471 0.9332605
## sample estimates:
## probability of success
## 0.7
```

We see that the function has three parameters, which correspond to T_{obs} , n and the μ under H_0 respectively (R calls the last one p). Additionally, we can specify whether we want a two-sided or one-sided test using the “alternative” option. The options are “two.sided”, “greater” and “less”.

We also see that the function returns an object, which summarizes the test that was performed. If we want to just get the P -value, we do:

```
tst <- binom.test(t, n, p = 0.5, alternative = c("two.sided"))
tst$p.value
```

```
## [1] 0.34375
```

Note that `binom.test` also returns sample estimates of the probability of success and confidence intervals. Section 8.8 provides explanations.

8.3 Fisher's exact test: Testing the association between two binary variables

Suppose we are trying to determine whether people who smoke are more likely to develop severe symptoms if they contract a respiratory virus than non-smokers. For this we collect data from $n = 110$ randomly sampled patients.

Assume we receive the following table as a result:

	Severe	Mild
Smoker	10	20
Non-smoker	10	70

We see that 30 of the patients were smokers, whereas 80 were non-smokers. We further observe that only $\frac{1}{8}$ th of non-smokers developed severe symptoms, whereas $\frac{1}{3}$ rd of the smokers did. The odds are 1:2 (10 severe versus 20 mild) for infected smokers to develop severe symptoms against 1:7 (10 severe versus 70 mild) for non-smokers. Hence, these data suggests that there is relationship between smoking and developing severe symptoms, with odds about 3.5 times higher for smokers than for non-smokers.

Once again, we need to make sure that these results are statistically significant. We cannot use the binomial test, because now we are not just considering one binary variable, but rather we are investigating the relationship between two binary variables.¹⁵

8.3.1 Permutation testing and the hypergeometric distribution

It is enlightening to approach first this problem with permutation testing (See Chapter 7). To this end, one shall first consider the underlying, not aggregated, dataset of individual cases. Such tidy dataset, where one row is one patient and each column one variable, would have the following structure:

Patient	Smoker	Symptoms
patient_1	no	mild
patient_2	yes	severe
patient_3	no	severe
patient_4	yes	severe
patient_5	yes	mild
...

For permutation testing, the null hypothesis is the independence of the Smoker and the Symptoms variables. With permutation testing, data under the null hypothesis are simulated by permuting values in one column (say “Symptoms”) keeping the order of the other column (say “Smoker”) fixed. For each permutation, we get a different *2x2 contingency table* which we will denote as:

	Severe	Mild	Row total
Smoker	a	b	a + b
Non-smoker	c	d	c + d
Column total	a + c	b + d	n = a + b + c + d

Note that any such permutation keeps the size of the dataset, the total number of smokers as well as the total number of patients with severe symptoms constant. We say that these permutations keep the margins (row and column totals) of the contingency table constant.

Consequently, one cell in the 2×2 contingency table suffices to characterize the entire table because all other counts can then be derived using the margins. Therefore, we can use any cell in the table as test statistic, but we usually use a , i.e. the upper left corner.

The good news is that the distribution of a under the null hypothesis (i.e. its frequency among all possible distinct permutations) can be exactly computed. It is given by the hypergeometric distribution:¹⁶

$$p(k = a | H_0) = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{a!b!c!d!n!}$$

8.3.2 Fisher's exact test

Using the hypergeometric distribution, we can now derive P -values, in the same way as we did before, namely by summing the probability of observing a test statistic as, or more, extreme as the one we observed. So, to compute the one-sided P -value, we would use:

$$P = \sum_{i \geq a} p(k = i | H_0)$$

This is called Fisher's exact test.

For our application purposes, we do not need to know the formula of the hypergeometric distribution, nor how it is derived. However, it is important to know that a formula exists and what the underlying assumptions are, i.e. that the margins of the 2x2 contingency table are considered to be fixed. That means that we do not consider n , $a + b$, $b + d$, $a + b$ and $c + d$ as random variables, but instead take these quantities as given (we “condition” on them). Note that this assumption is often violated in practice, as in the example above where we randomly sampled patients rather than sampling a fixed amount from each subgroup. But Fisher’s exact test is nevertheless applied as an exact instance of permutation testing.

There are alternatives to Fisher’s exact test that do not need all margins fixed assumptions. One is the formerly popular Chi-squared test, which is based on large number approximations. It is rarely needed nowadays, as Fisher’s exact test is exact and fast to compute. Another approach is based on logistic regression and will be addressed in a later Chapter.

8.3.3 Fisher’s exact test in R

In R, we can perform Fisher’s exact test using `fisher.test`. This requires a contingency table as input (See the base R function `table` to create them). For our contingency table, we get:

```
tbl = data.table(
  severe = c(10, 10),
  mild = c(20, 70)
)
tst <- fisher.test(tbl, alternative = "greater")
tst

##
## Fisher's Exact Test for Count Data
##
## data:  tbl
## p-value = 0.01481
## alternative hypothesis: true odds ratio is greater than 1
## 95 percent confidence interval:
##  1.316358      Inf
## sample estimates:
## odds ratio
##  3.453224
```

The one-sided p-value is 0.0148095. At the level $\alpha = 0.05$, one would therefore reject the null hypothesis of independence of symptom severity and smoking status.

As for the binomial test, the p-value can be extracted with `$p.value`. The function `fisher.test` also returns an estimate of the odds ratio and its confidence interval. The estimate of the odds ratio is based on a estimation procedure robust for low counts, giving close yet slightly different estimates than the sample odds ratio (we obtained 3.5 at the start of Section 8.3).

8.4 Testing the association between one quantitative and one binary variable

We asked earlier on the yeast dataset (Section 7.1) whether the genotype at marker 5211 significantly associates with growth rates in Maltose media. We saw that yeast strains which have the wild isolate genotype seemed to generally grow faster than those which had the lab strain genotype at this particular marker:

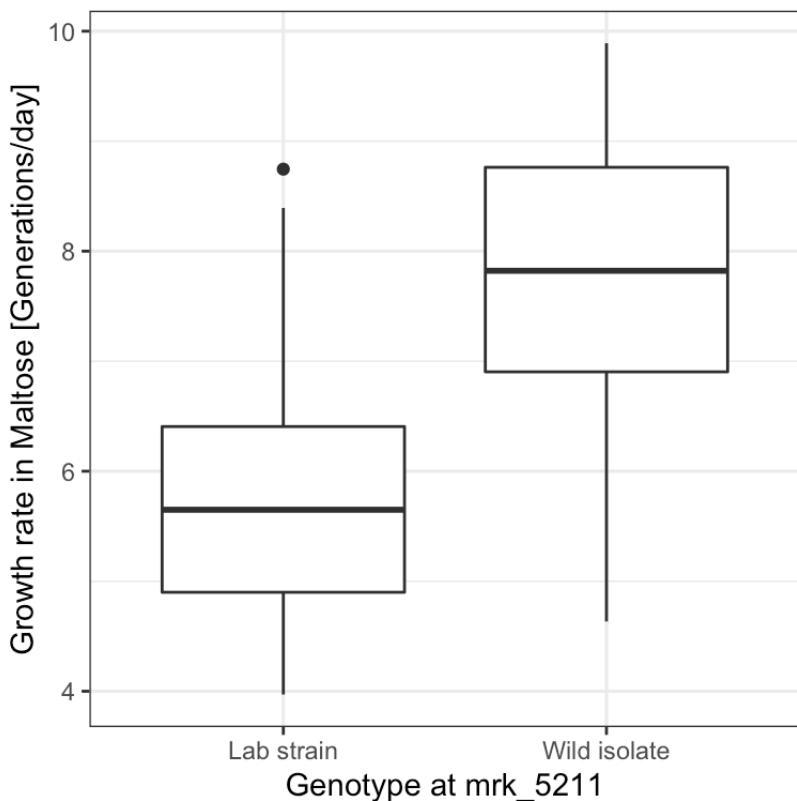
```
genotype <- fread("extdata/eqtl/genotype.txt")
genotype <- genotype %>%
  melt(id.vars = 'strain', variable.name = 'marker', value.name = 'genotype')
marker <- fread("extdata/eqtl/marker.txt")
growth <- fread("extdata/eqtl/growth.txt")
growth <- growth %>% melt(id.vars = "strain", variable.name = 'media', value.name = 'growth_rate')

mk <- marker[chrom == "chr07" & start == 1069229, id]

dt <- merge(
  growth[media == 'YPMalt'],
  genotype[marker == mk, .(strain, genotype)],
  by = 'strain'
)

p <- dt%>%
  ggplot(., aes(genotype, growth_rate)) +
  geom_boxplot() +
  xlab(paste0("Genotype at ", mk)) +
  ylab("Growth rate in Maltose [Generations/day]") +
  mytheme

p
```



Here we are evaluating the association between a binary variable (the genotype at marker 5211) and a quantitative variable (the growth rate in Maltose media). This scenario does not fit the tests we have seen previously. We have to develop a new one.

To formalize this problem, we first note that the binary variable splits the quantitative data into two groups. Let $X = x_1, \dots, x_{n_x}$ be the quantitative data of the first group (i.e. the growth rates of yeast strains with lab strain genotype), and $Y = y_1, \dots, y_{n_y}$ be the quantitative data of the second group (i.e. the growth rates of yeast strains with wild isolate genotype).

To develop a test, we again need a null hypothesis, a test statistic and a distribution of the test statistic under the null hypothesis. For this problem, we will consider two different tests.

8.4.1 The t-test

The first test statistic we will look at is **Student's t** , defined as:

$$t = c \frac{\bar{x} - \bar{y}}{s} \quad (8.2)$$

where \bar{x} and \bar{y} are the sample means of X and Y respectively, s is the pooled standard deviation, and c is a constant that depends on the sample size of each group. In details:

$$\bar{x} = \frac{1}{n_x} \sum_i x_i$$

$$\bar{y} = \frac{1}{n_y} \sum_i y_i$$

$$s_p = \sqrt{\frac{\sum_i (x_i - \bar{x})^2 + \sum_i (y_i - \bar{y})^2}{n_x + n_y - 2}}$$

$$c = \sqrt{\frac{n_x n_y}{n_x + n_y}}$$

While the details can always be looked up, understanding Equation (8.2) is useful. Intuitively, the t -statistic compares, up to the constant c , the “signal” of group difference, namely the estimated difference of the means of the two groups, to the “noise”, i.e. how uncertain we are about our estimate of this difference. This “noise” in our estimate is itself the ratio of the typical variations within the groups (s) over a term capturing the sample size (c). One can thus interpret it as a signal-to-noise ratio. If the t -statistic is large, then we see a clear difference in means. By contrast, if the t -statistic is small, then the difference in means is not large compared to the noise. Larger sample size (more data) or larger between-group differences compared to within-group differences lead to larger t -statistics.

8.4.1.1 Student's t-distribution

Before we can derive the distribution of this test statistic under the null hypothesis, we need to make some additional assumptions about the data, namely:

- All observations $x_1, \dots, x_{n_x}, y_1, \dots, y_{n_y}$ are independent of each other
- we assume that X and Y both follow Gaussian distributions
- X and Y have the same unknown variance

A consequence of these assumptions is that our null hypothesis simplifies. If both X and Y are Gaussian with the same variance, the only way the two groups can differ is if the Gaussians have different means. Therefore, the null hypothesis is that the expectations are equal:

$$H_0 : E(X) = E(Y)$$

Under H_0 , our test statistic t follows a Student's t-distribution with $\nu = n_x + n_y - 2$ degrees of freedom (the degrees of freedom ν is the parameter of Student's t-distribution). Figure 8.2 shows the shape of Student's t -distribution for different degrees of freedom ν .

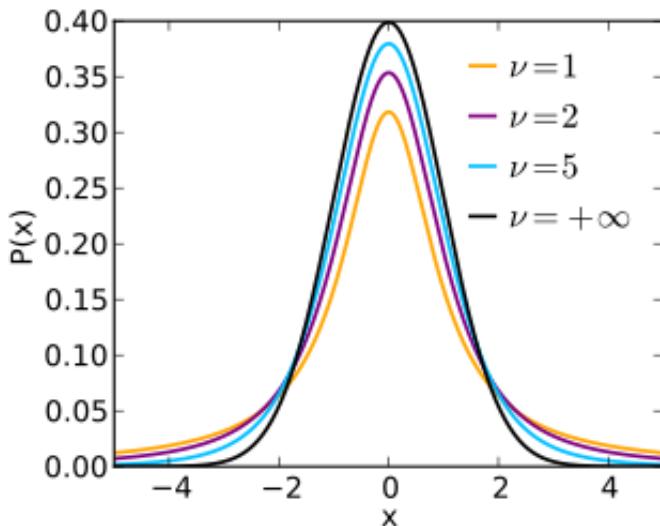


Figure 8.2: Student's t-distribution for various degrees of freedom. Source:

https://en.wikipedia.org/wiki/Student%27s_t-distribution

We can make two observations. Firstly, the distribution of the t -statistic under H_0 does not depend on the (unknown) variance. Secondly, Student's t -distribution has heavier tails than the Gaussian. This intuitively comes from the fact that, while the numerator of the t -statistic is normally distributed, the estimate of the standard deviation in the denominator is noisy. **The smaller the sample size n , the noisier the estimate. Hence, the smaller the degrees of freedom, the heavier the tails.** For infinite degrees of freedom, Student's t -distribution equals the normal distribution.

8.4.1.2 Student's t-test in R

In R we can perform a t-test using the `t.test` function. Since in the basic Student's t-test we assume **equal variances**, we have to set the argument `var.equal` to `True`. One can extract the values for each group and perform the test. Rather than manually extracting the two groups, we use the formula syntax (`growth_rate ~ genotype`) and let the `t.test` function do it for us:

```
t.test(growth_rate ~ genotype, data=dt, var.equal=TRUE)

##
##  Two Sample t-test
##
## data: growth_rate by genotype
## t = -10.77, df = 152, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -2.406400 -1.660387
## sample estimates:
## mean in group Lab strain mean in group Wild isolate
##                      5.763086                      7.796480
```

Note that the function reports the t -statistic and the degrees of freedom, the confidence intervals for the difference of the means, in addition to the p -value. Note also that the function helpfully reminds us what null hypothesis we are testing against.

8.4.1.3 Unequal variance (Welch's test) in R

In practice, we generally do **not assume equal variances**. This is called Welch's test and slightly changes the degrees of freedom. This test is performed in R by default if we do not set `var.equal` to True.

```
t.test(growth_rate ~ genotype, data=dt)

##
## Welch Two Sample t-test
##
## data: growth_rate by genotype
## t = -10.805, df = 152, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -2.405189 -1.661599
## sample estimates:
## mean in group Lab strain mean in group Wild isolate
##                 5.763086                  7.796480
```

8.4.2 Wilcoxon rank-sum test: An alternative to the t-test for non-Gaussian data

8.4.2.1 Assumptions

As we saw, the t -test assumes the data follows a specific distribution, namely a Gaussian. There are many situations where this is reasonable, but in general we cannot guarantee that this assumption holds. Using the t -test if the data is not normal can lead to wrong conclusions.

The Wilcoxon Rank-Sum test is a popular alternative to the t-test. It makes very few assumptions about the data, namely that:

- All observations $x_1, \dots, x_{n_x}, y_1, \dots, y_{n_y}$ are **independent** of each other
- The **responses are ordinal**, i.e. we can rank them

Specifically, we assume that under the null hypothesis H_0 , the probability of an observation from the population X exceeding an observation from the second population Y equals the probability of an observation from Y exceeding an observation from X:

$$H_0 : p(X > Y) = p(Y > X)$$

In other words, if we randomly take observations $x \in X$ and $y \in Y$, we would expect that $x > y$ occurs as often as $y > x$ (ignoring ties).

A stronger null hypothesis commonly used is “The distributions of both populations are equal” which implies the previous hypothesis.

For a two-sided test, the alternative hypothesis is “the probability of an observation from the population X exceeding an observation from the second population Y is different from the probability of an observation from Y exceeding an observation from X: $p(X > Y) \neq p(Y > X)$.” The alternative may also be stated in terms of a one-sided test, for example: $p(X > Y) > p(Y > X)$. This would mean that if we randomly take observations $x \in X$ and $y \in Y$, we would expect that $x > y$ occurs more often than $y > x$.

8.4.2.2 The Mann-Whitney U statistic and the Wilcoxon rank-sum test

Consider first that we rank all observed values (and ignore ties), e.g.:

$$x_5 < y_{10} < y_{12} < y_3 < x_4 < x_{17} < \dots$$

The idea of the Wilcoxon rank-sum test is that under the null hypothesis, the x_i 's and y_i 's should be well interleaved in this ranking. In contrast, if say X tend to be smaller than Y , then the x_i 's will get lower ranks. The test statistics is therefore based on the sum of the ranks of the realizations of one the two variables.

Specifically, we define the quantity U_x as:

$$U_x = R_x - \frac{n_x(n_x + 1)}{2}$$

where R_x is the sum of the ranks of the x_i 's. In the example above, $R_x = 1 + 5 + 6 + \dots$, and n_x is the number of observations of set X . The term $\frac{n_x(n_x + 1)}{2}$ (this is the famous Gauss sum) is a constant so that $U_x = 0$ when all the first values are from X .

U_y is defined analogously.

The Mann-Whitney U statistic is defined as:

$$U = \min\{U_x, U_y\}$$

P-values are then based on the distribution of Mann-Whitney U statistic under the null hypothesis. It combines tabulated values for small sample sizes and Central Limit Theorem approximation for large sample sizes (exploiting that the expectation and the variance of U under the null can be analytically derived.¹⁷)

8.4.2.3 Wilcoxon rank-sum test in R

In R we can perform the Wilcoxon rank-sum test using the `wilcox.test` function, whose usage is analogous to the usage of `t.test`:

```
wilcox.test(growth_rate ~ genotype, data=dt)
```

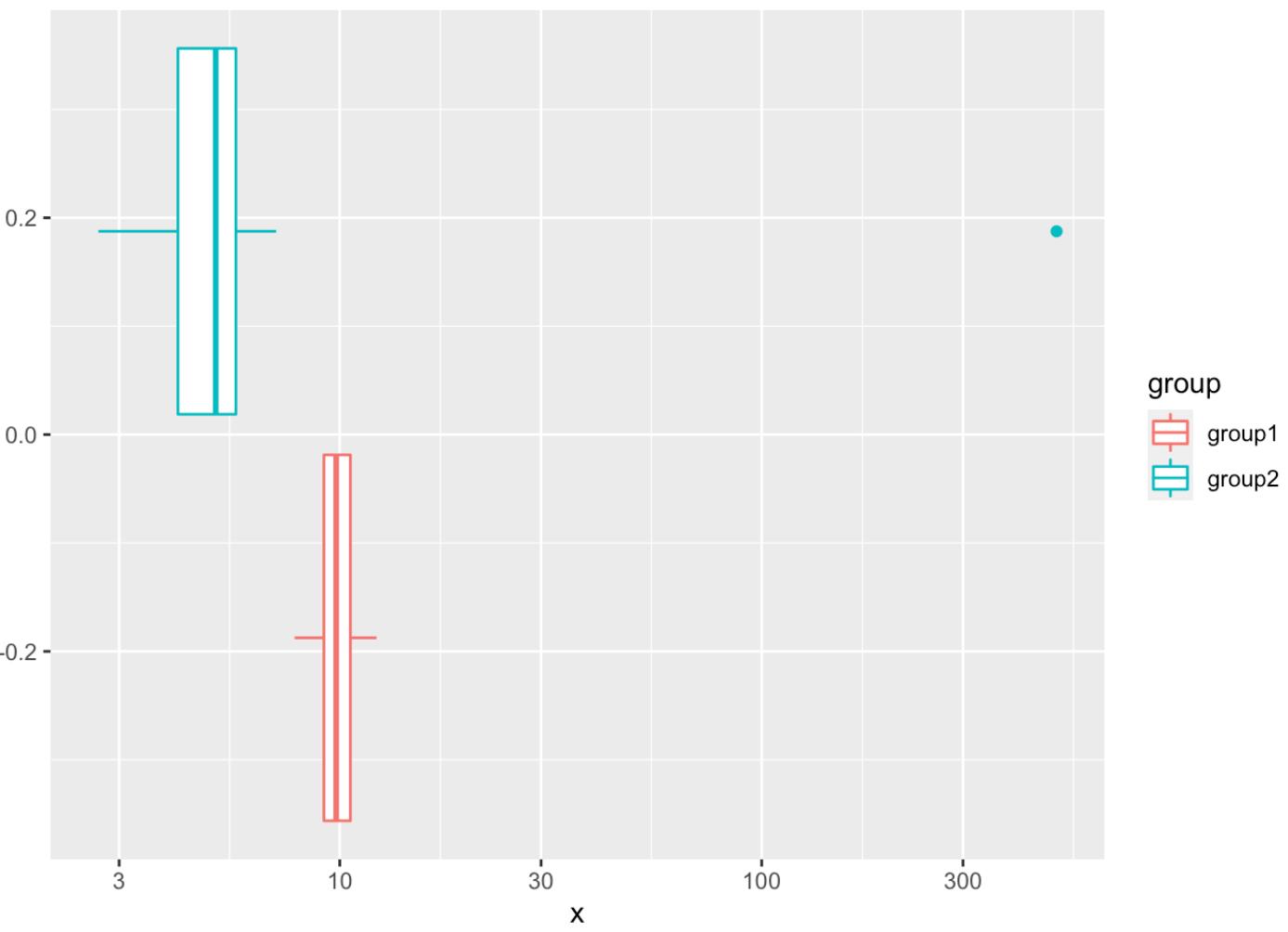
```
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: growth_rate by genotype  
## W = 690, p-value = 2.264e-16  
## alternative hypothesis: true location shift is not equal to 0
```

8.4.3 Why bother with the Wilcoxon rank-sum test?

The Wilcoxon rank-sum test makes less assumptions than the *t*-test, specifically because it does not require that the data follows a Gaussian distribution. We will now see an example to illustrate this.

We construct a highly pathological example:

```
set.seed(10)  
x1 <- rnorm(100, 10)  
x2 <- c(rnorm(99, 5), 500)  
grp_tbl <- data.table(  
  group = rep(c("group1", "group2"), each=100),  
  x = c(x1,x2)  
)  
ggplot(data = grp_tbl, aes(x=x, color=group)) + geom_boxplot() + scale_x_log10()
```



In this example, the groups are sampled from normal distributions with a different mean. However, we add a non-normal outlier to the second group, which ensures that the overall mean looks the same.

Recall that, as a consequence of assuming that the data is Gaussian, the null hypothesis of the *t*-test is that the difference in means is zero. There is no difference in means here, so the *t*-test cannot reject the null hypothesis:

```
t.test(x1, x2)

##
##  Welch Two Sample t-test
##
## data: x1 and x2
## t = -0.00052129, df = 99.072, p-value = 0.9996
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -9.829688 9.824525
## sample estimates:
## mean of x mean of y
## 9.863451 9.866033
```

But clearly, these groups are overall quite different, and their means only appear similar due to this one outlier. The null hypothesis of the Wilcoxon rank-sum test is not about the means. Instead, the Wilcoxon rank-sum test uses the rank distribution, and in our example most observations of the second group will rank above the observations of the first. The one outlier will not affect the ranking much. Thus, the Wilcoxon rank-sum test will reject here:

```
wilcox.test(x1, x2)

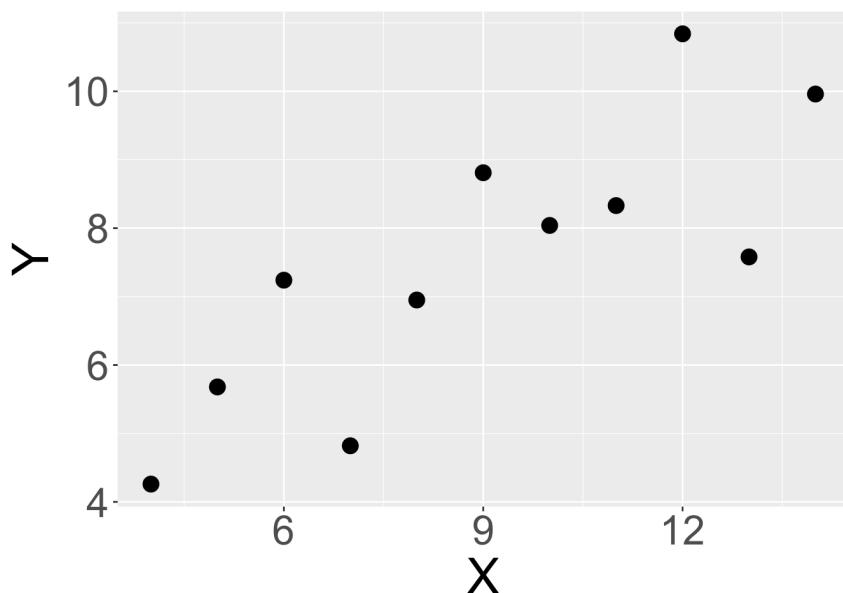
##
##  Wilcoxon rank sum test with continuity correction
##
## data: x1 and x2
## W = 9900, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

This is a rather synthetic example, but the underlying point is very general: if we are **unsure whether the distributional assumption is met, a test like the Wilcoxon rank-sum test** will generally be **more robust** than a test making distributional assumptions like the t -test. But do note that there is a flip side to this: **if the data is indeed Gaussian, then the t -test will be more powerful** (i.e. more sensitive in detecting violations of the null hypothesis) than the more generally applicable Wilcoxon rank-sum test.

8.5 Association between two quantitative variables

The last scenario we will consider in this chapter concerns the dependence between two quantitative variables. That is, we assume we have quantitative data in the form of tuples $(X, Y) : (x_1, y_1), \dots, (x_n, y_n)$ and we want to see if knowing one of the values in such a tuple gives us information about the other one.

As a visual example, we consider a synthetic dataset, namely Anscombe's first dataset¹⁸:



Looking at the plot above, it sure seems that there is a positive relationship between X and Y in this data. Specifically, if we know that x_i is relatively high, it seems that we can usually assume that y_i will be high too (and vice-versa). But once again, we need a test to prevent us from being fooled by randomness.

This means we again need null and alternative hypotheses, a test statistic and a distribution of the test statistic under the null hypothesis.

We will consider two different tests which are based on different notions of the concept of correlation.

8.5.1 The Pearson correlation test

An important property is that when two variables (X, Y) form a bivariate Gaussian distribution,¹⁹ their independence is equivalent to their population Pearson correlation coefficient $\rho_{X,Y}$ equals 0 (See Appendix D). This motivates for a Hypothesis test, called the Pearson correlation coefficient test.

8.5.1.1 Pearson's correlation coefficient

The Pearson correlation coefficient test is based on the sample estimate of the population Pearson correlation coefficient, defined as:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (8.3)$$

where $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ is the sample mean, and analogously for \bar{y} .

Let us look at the components of this. The numerator compares the deviation of the x_i and y_i to their respective means. Terms of the sum are positive if both x_i and y_i vary in the same direction (larger or lesser) compared to their mean and negative otherwise. Hence, the numerator is largely positive when deviations from the means agree in direction, largely negative when they are opposite, and about 0 when deviations are independent of each other. More formally, the numerator is proportional to the sample covariance (See Appendix D). The terms in the denominator is proportional to the individual sample standard deviations of X and Y (See Appendix D). Hence, r compares how much the X and Y vary together to the product of how much they vary individually.

The Pearson correlation coefficient is symmetric. Moreover, it is invariant to affine transformations of the variables. It ranges from -1 to 1, where:

- $r = 1$ implies that x and y are perfectly linearly related with a positive slope
- $r = -1$ implies that x and y are perfectly linearly related with a negative slope

8.5.1.2 The test

The assumptions of the Pearson correlation test are:

- (X, Y) is a bivariate Gaussian distribution
- The observations (X_i, Y_i) are i.i.d.

The null hypothesis is that the two variables are statistically independent, which under the above assumptions amounts to state that:

$$H_0 : \rho_{(X,Y)} = 0$$

The test statistic is given by:

$$t = r \sqrt{\frac{n - 2}{1 - r^2}}$$

Under H_0 , the test statistic t defined above follows a Student's t -distribution with degrees of freedom $n - 2$.

In R, we can use `cor.test` with `method="pearson"` to perform a Pearson correlation test.

```
cor.test(anscombe$x1, anscombe$y1, method="pearson")

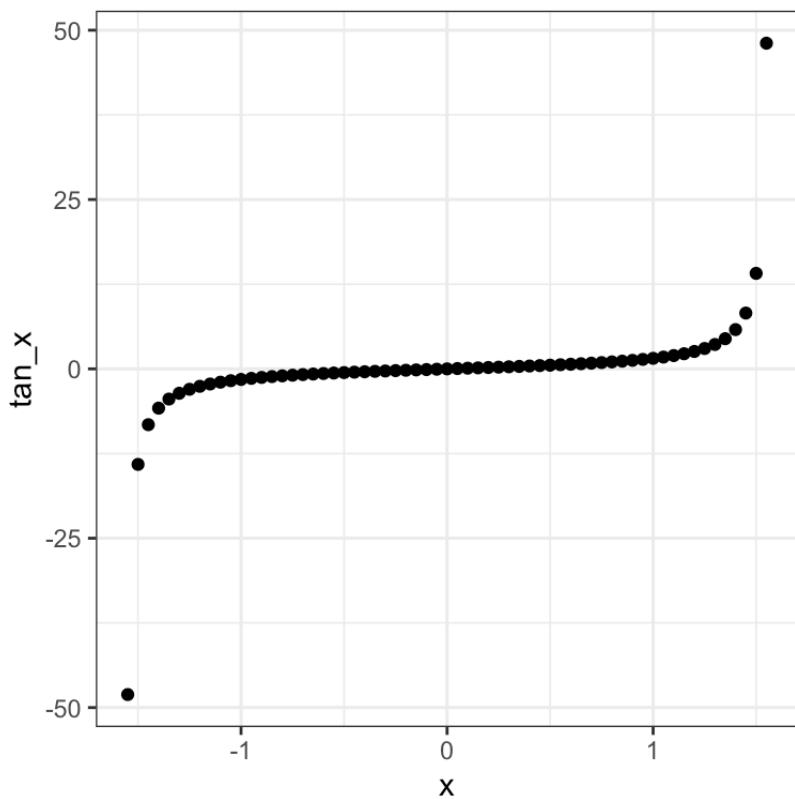
##
##  Pearson's product-moment correlation
##
## data: anscombe$x1 and anscombe$y1
## t = 4.2415, df = 9, p-value = 0.00217
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.4243912 0.9506933
## sample estimates:
##      cor
## 0.8164205
```

8.5.2 The Spearman rank correlation test

8.5.2.1 Motivation

Pearson's correlation captures linear relationship between variables, which is quite restrictive. For instance, if one of the variables is in log-scale or quadratic scale, then the linear relationship is lost. Here is a constructed example:

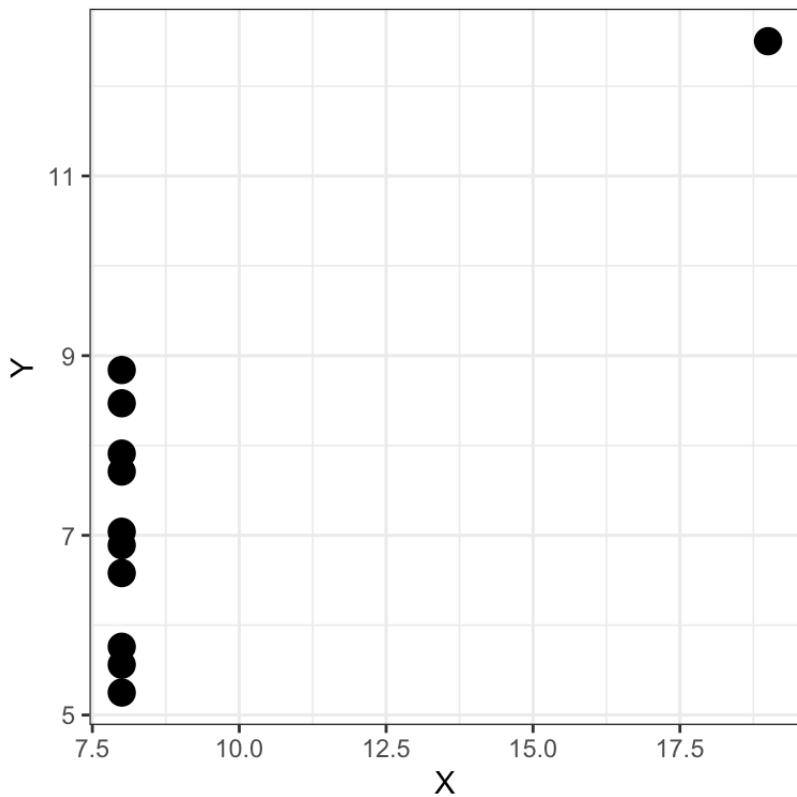
```
x_vec <- seq(-1.55, 1.55, 0.05)
tan_x <- tan(x_vec)
dt_tan <- data.table(x = x_vec, tan_x = tan_x)
ggplot(data=dt_tan, aes(x=x, y=tan_x)) + geom_point() + mytheme
```



These two constructed variables relate exactly to each other by a monotonic relationship (the tangent function).²⁰
However their Pearson correlation is modest:

```
cor(x_vec, tan_x, method="pearson")  
  
## [1] 0.5643079
```

Conversely Pearson' correlation can be excessively large in presence of outliers. Anscombe's quartet provides an example:



```
cor.test(anscombe$x4, anscombe$y4, method="pearson")

##
## Pearson's product-moment correlation
##
## data: anscombe$x4 and anscombe$y4
## t = 4.243, df = 9, p-value = 0.002165
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.4246394 0.9507224
## sample estimates:
## cor
## 0.8165214
```

We see there is a high Pearson correlation between X and Y . Furthermore, if we use a significance level $\alpha = 0.05$, we reject the null hypothesis that $H_0 : r = 0$ and conclude there is a statistically significant association between X and Y . The plot however tells us this is driven by a single outlier.²¹ The data is probably not Gaussian.

8.5.2.2 Spearman's correlation coefficient

Spearman's correlation (or rank-correlation, denoted ρ) addresses those issues by computing the correlation not on the original scale but on rank-transformed values. To compute ρ , we rank the variables X and Y separately, yielding rankings such as:

$$x_7 < x_3 < x_5 < x_1 < \dots$$

and

$$y_3 < y_5 < y_7 < y_2 < \dots$$

We then compute the position of each data point in the ranking, yielding the transformed dataset:

$$\text{rank}_x(X), \text{rank}_y(Y) = (\text{rank}_x(x_1), \text{rank}_y(y_1)), \dots, (\text{rank}_x(x_n), \text{rank}_y(y_n))$$

For the rankings above, we would have for example that $(\text{rank}_x(x_7), \text{rank}_y(y_7)) = (1, 3)$.

Spearman's ρ is then computed as the Pearson correlation of the rank-transformed data.

In R we can compute it using:

```
cor(rank(anscombe$x4), rank(anscombe$y4), method="pearson")

## [1] 0.5
```

Or more directly, by specifying `method="spearman"`.

```
cor(anscombe$x4, anscombe$y4, method="spearman")

## [1] 0.5
```

8.5.2.3 The test

Based on the Spearman correlation, we can also define a test for the relationship between two variables.

This test does **not make distributional assumptions**.

The null hypothesis is:

H_0 : The **population rank-correlation is 0**.

R implements a statistical test based on tabulated exact permutations for small sample sizes and approximations for larger sample sizes.

Applied to the Anscombe's dataset we get:

```
cor.test(anscombe$x4, anscombe$y4, method="spearman")

## Warning in cor.test.default(anscombe$x4, anscombe$y4, method =
## "spearman"): Cannot compute exact p-value with ties
```

```

## 
##  Spearman's rank correlation rho
## 
## data: anscombe$x4 and anscombe$y4
## S = 110, p-value = 0.1173
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
## rho
## 0.5

```

We see that the Spearman test would not reject, which makes sense, as the rank is less likely to be misled by the outlier data point.²² Generally, the Spearman test is less powerful than Pearson when the data is actually Gaussian, but it is more robust to outliers and captures monotonic, yet non-linear, relationships. In practice, the Spearman test is often used.

8.6 Testing associations of two variables: Overview

Figure 8.3 summarizes the different tests we have seen for the association of two variables, together with the typical companion plots:

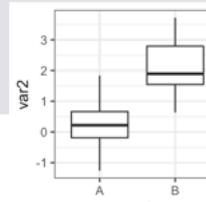
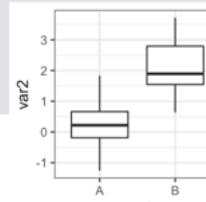
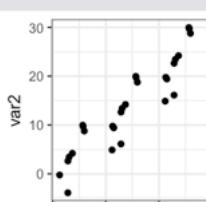
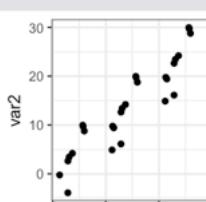
Variable 1	Variable 2	Assumption	H_0	Test	R	Plot	Plot code
Binary	Binary	Fixed margins	Independence	Fisher exact test	fisher.test(tab)	var2 var1 A B A 7 6 B 7 10	table(var1, var2)
Binary	Continuous	Gaussian	Equal means	Welch test (Student t-test with unequal variance)	t.test(var2 ~ var)		ggplot(dt, aes(var1, var2)) + geom_boxplot()
Binary	Continuous	-	$P(X>Y) = P(Y>X)$	Wilcoxon rank-sum test	wilcox.test(var2 ~ var1)		ggplot(dt, aes(var1, var2)) + geom_boxplot()
Continuous	Continuous	Gaussian	Correlation = 0 ($r = 0$)	Pearson correlation test	cor.test(var2 ~ var1)		ggplot(dt, aes(var1, var2)) + geom_point()
Continuous	Continuous	-	Rank correlation = 0 ($\rho = 0$)	Spearman correlation test	cor.test(var2 ~ var1, method="spearman")		ggplot(dt, aes(var1, var2)) + geom_point()

Figure 8.3: Overview of two-variable tests

8.7 Assessing distributional assumptions with Q-Q Plots

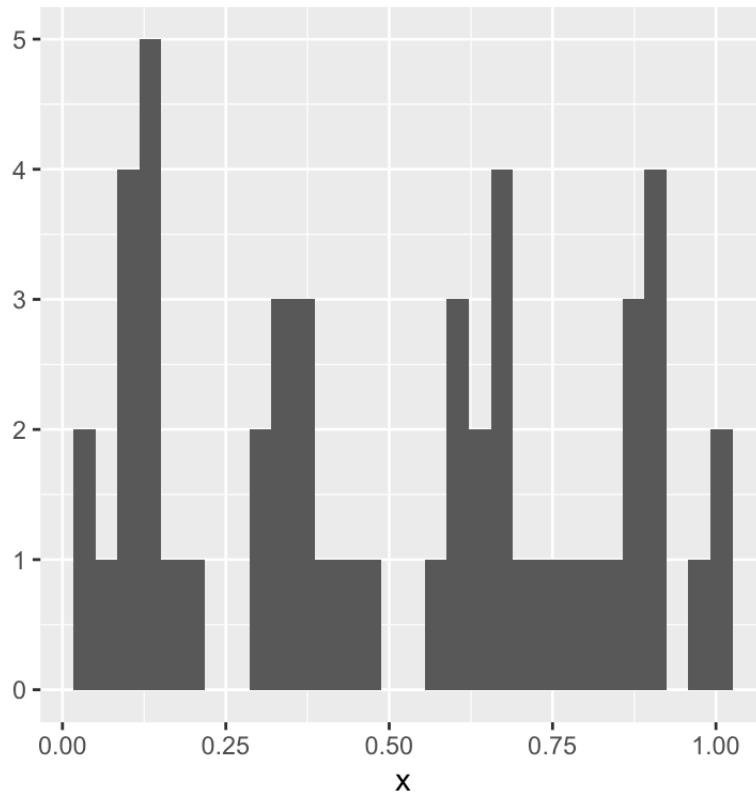
As we saw in this chapter, several tests assume that the **data follows a particular distribution**. We will now explore a plot which we can use to **check whether such an assumption is reasonable**.

8.7.1 Limitations of Histograms

We already know a plot which can be used to visualize distributions, namely the histogram. We might think that it could be used to check distributional assumptions. However, this is somewhat complicated by the **difficulty of choosing the right bin size**. Consider, for example, the following histogram, visualizing a sample taken from a uniform distribution on the interval 0 to 1.:

```
x <- runif(50) ## uniformly distributed data points
# qplot is a quick plotting function
qplot(x, geom="histogram")

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



Just looking at the histogram, it is hard to see that the underlying data comes from the uniform distribution.

8.7.2 Q-Q plots: Comparing empirical to theoretical quantiles

What could be a better approach here? One thing we can do is look at the quantiles.

The basic idea here is as follows: if the data actually follows a uniform distribution on the interval 0 to 1, then we expect 10% of the data in the interval [0,0.1], 20% in the interval [0,0.2], and so on...

We can now compute whether our data conforms to this expectation. We get that:

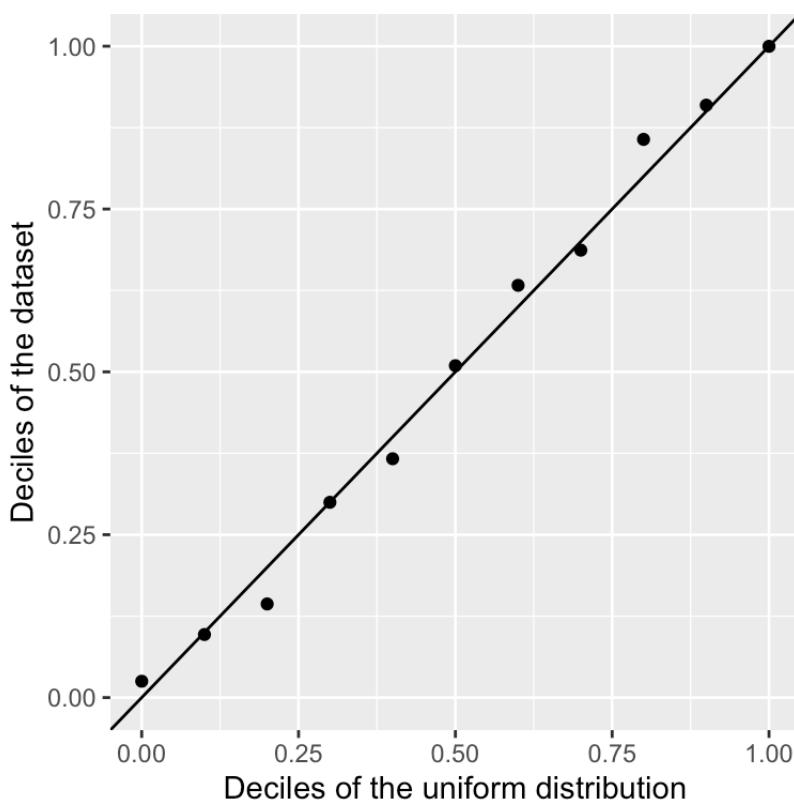
```
dec <- quantile(x, seq(0,1,0.1))
dec

##          0%        10%       20%       30%       40%
## 0.02503686 0.09677631 0.14369892 0.29974460 0.36662386
##          50%        60%       70%       80%       90%
## 0.50953583 0.63289259 0.68685499 0.85694255 0.90948354
##         100%
## 0.99981636
```

Here we implicitly chose to always make jumps of 10%. These quantiles are therefore called deciles.

We can make a scatter plot which compares the expected and the theoretical deciles:

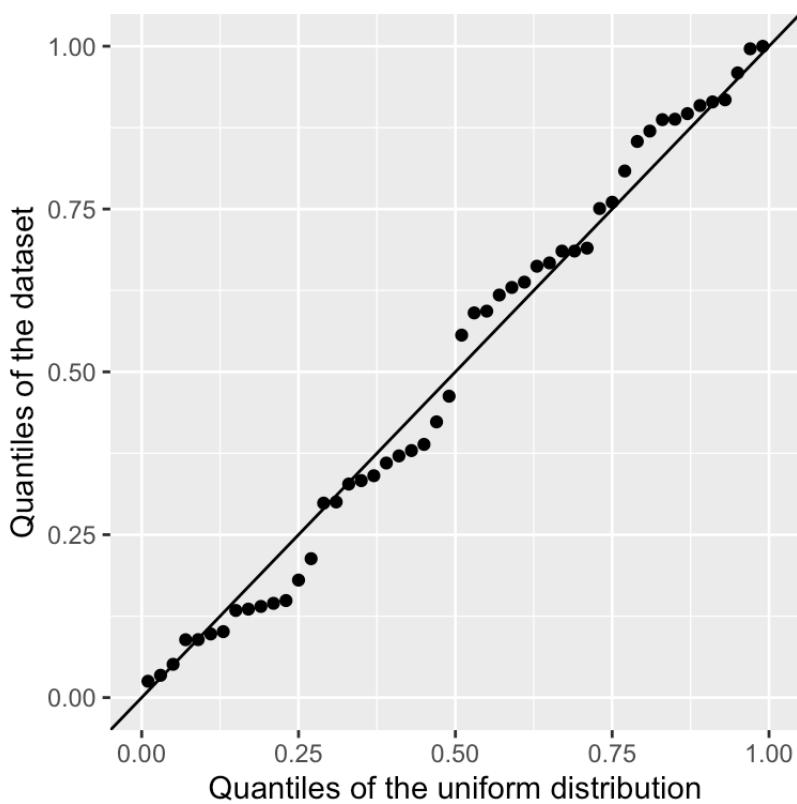
```
ggplot(
  data.table(
    x=seq(0,1,0.1),
    y=dec
  ),
  aes(x,y)
) + geom_point() +
  xlim(c(0,1)) + ylim(c(0,1)) +
  xlab("Deciles of the uniform distribution") +
  ylab("Deciles of the dataset") +
  geom_abline(intercept=0,slope=1) ## diagonal y=x
```



We see that they match quite well.

For a finite sample we can estimate the quantile for every data point. One way is to use as expected quantile $(r - 0.5)/N$ (Hazen, 1914), where r is the rank of the data point. The R function `ppoints` gives more accurate values.

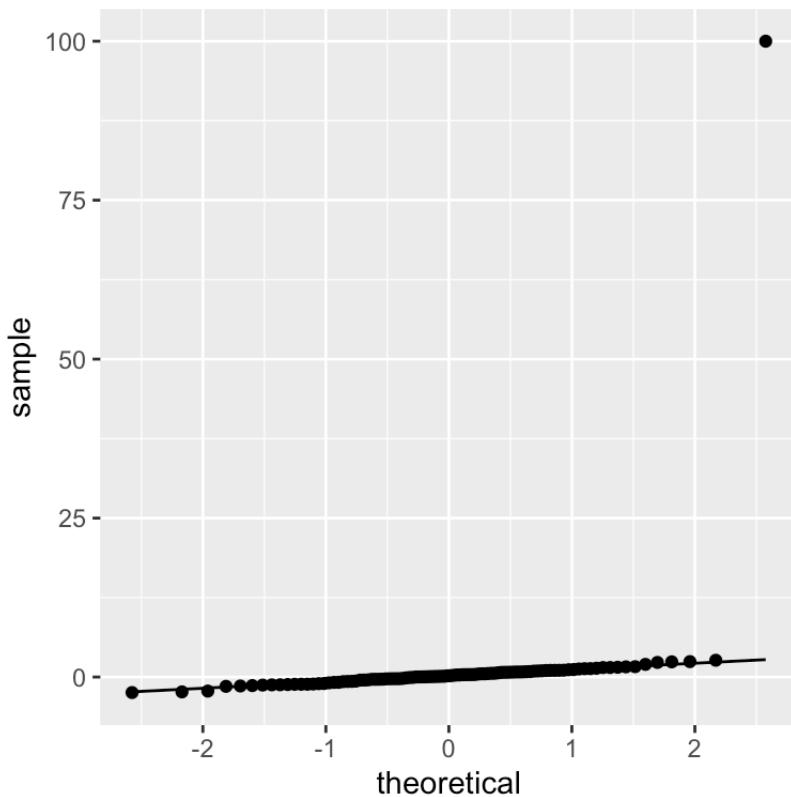
```
ggplot(
  data.table(
    x=ppoints(length(x)),
    y=sort(x)
  ),
  aes(x,y)
) + geom_point() +
  xlim(c(0,1)) + ylim(c(0,1)) +
  xlab("Quantiles of the uniform distribution") +
  ylab("Quantiles of the dataset") +
  geom_abline(intercept=0,slope=1) ## diagonal y=x
```



This is called a Q-Q plot, which is short for Quantile-Quantile plot. When the distribution matches the data, as above, the points should be close the diagonal.

Let us now recall the example we used to justify the Wilcoxon test. There we added an extreme outlier to a gaussian, which mislead the t -test. Can we discover, using a Q-Q plot, that this data violates an assumption of normality?

```
group_qq <- c(rnorm(99, 0), 100)
qq_tbl <- data.table(sample = group_qq)
ggplot(data = qq_tbl, aes(sample = sample)) + geom_qq() + stat_qq_line()
```



Our artificially injected outlier shows up very clearly in the Q-Q plot as a strong deviation from what we expect from a normal distribution.

8.7.3 Typical Q-Q plots

Figure 8.4 give more examples. We assume here the Normal distribution (Gaussian with mean 0 and variance 1) as reference theoretical distribution. These plots show how different violations of the distributional assumption translate to different deviations from the diagonal in a Q-Q plot.

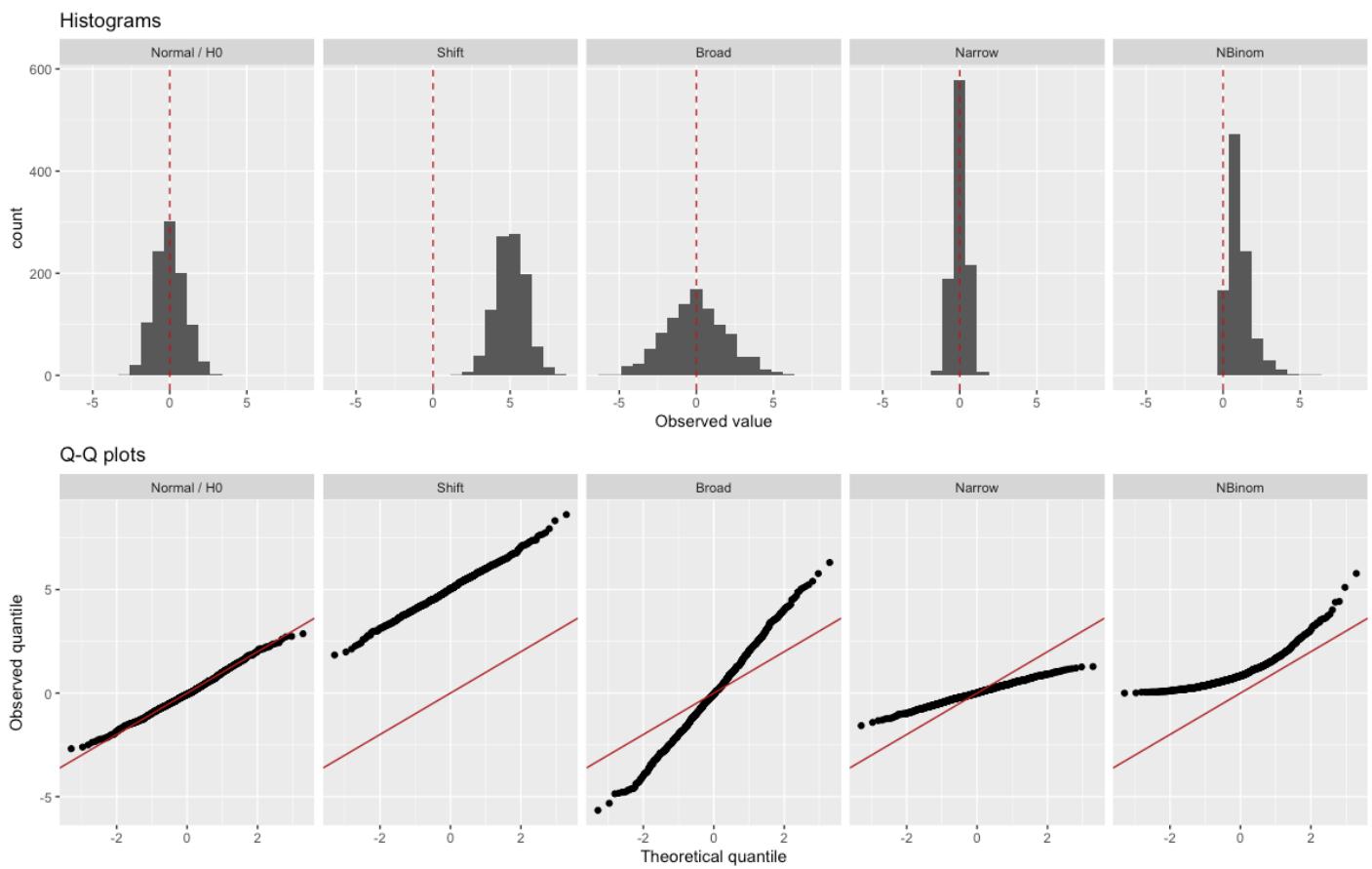


Figure 8.4: Examples of Q-Q plots. The theoretical distribution is in each case the Normal distribution (Gaussian with mean 0 and variance 1). The upper row shows histograms of some observations, the lower row shows the matching Q-Q plots. The vertical red dashed line marks the theoretical mean (0, top row) and the red lines the $y=x$ diagonal (bottom row).

The middle three plots show what happens when one particular aspect of the distributional assumption is incorrect. The second from the left shows what happens if the data has a mean higher than we expected, but otherwise follows the distribution. The middle one shows what happens if the data has fatter tails (i.e. more outliers) than we expected - this occurs frequently in practice. The second from the right shows what happens if the distribution is narrower than expected. The last plot shows a combination of these phenomena. There the data come from a non-negative asymmetric distribution.²³ The Q-Q plot shows a lack of low values (capped at 0) and an excess of high values.

8.8 Analytical Confidence intervals

Remember the definition of confidence intervals: A **confidence interval** of confidence level $1 - \alpha$ for a parameter θ is an interval, which would the data generation process be repeated, would contain the parameter with probability $1 - \alpha$.

For instance, a 95% confidence interval for the expectation μ , would be an interval $[a, b]$ such that:

$$p(a \leq \mu \leq b) = 0.95$$

Remember also that here, a and b are random, μ is not!

We have seen how to approximate confidence intervals using the case-resampling bootstrap in the previous chapter. But, confidence intervals can also be computed analytically under some assumptions. We will see this in detail for the binomial case first.

8.8.1 Binomial case

We use the same setup as previously:

- We make N independent random tosses of a coin.
- X_i : the value of the i -th toss. $X_i = 1$ for head $X_i = 0$ for tail.
- $\mu = E(X_i) = p(X_i = 1)$ the probability of getting a head (same for all tosses).

The sample mean is then given by:

$$\bar{X} = \frac{1}{N} \sum_i X_i$$

And the estimated standard deviation is given by:

$$\hat{\text{SE}}(\bar{X}) = \sqrt{\frac{\bar{X}(1 - \bar{X})}{N}}$$

We want to know the probability that the interval $[\bar{X} - \hat{\text{SE}}(\bar{X}), \bar{X} + \hat{\text{SE}}(\bar{X})]$ contains the true proportion μ . (And do not forget that \bar{X} and $\hat{\text{SE}}$ are random variables, whereas μ is not!)

To determine the probability that the interval includes μ , we need to compute this:

$$p\left(\bar{X} - \hat{\text{SE}}(\bar{X}) \leq \mu \leq \bar{X} + \hat{\text{SE}}(\bar{X})\right)$$

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

$$p\left(-1 \leq \frac{\bar{X} - \mu}{\hat{\text{SE}}(\bar{X})} \leq 1\right)$$

8.8.1.1 Normal approximation interval using the Central Limit Theorem

The Central Limit Theorem implies that the sample mean distributes for large N as a Normal distribution with mean $E(\bar{X})$ and variance $\text{Var}(\bar{X})/N$:

$$p(\bar{X}) = N(\mu, \text{Var}(\bar{X})/N)$$

It is known that $\text{Var}(X_i) = \mu(1 - \mu)$ (this is because the underlying data is bernoulli).

Hence,

$$p\left(-1 \leq \frac{\bar{X} - \mu}{\hat{\text{SE}}(\bar{X})} \leq 1\right) = p(-1 \leq Z \leq 1)$$

where Z is normally distributed with mean 0 and variance 1.

This can be computed in R using the cumulative distribution function of the normal distribution:

```
pnorm(1) - pnorm(-1)
```

```
## [1] 0.6826895
```

proving that we have approximately 68% probability.

8.8.1.2 Defining the interval for a predefined confidence level

If we want to have a larger probability, say 99%, we need to multiply by whatever z satisfies the following:

$$\Pr(-z \leq Z \leq z) = 0.99$$

This is obtained using the quantile function of the normal distribution. In R, using:

```
z <- qnorm(0.995)
```

```
z
```

```
## [1] 2.575829
```

will achieve this because by definition `pnorm(qnorm(0.995))` is 0.995 and by symmetry `pnorm(1-qnorm(0.995))` is $1 - 0.995$. As a consequence, we have that:

```
pnorm(z) - pnorm(-z)
```

```
## [1] 0.99
```

is $0.995 - 0.005 = 0.99$.

8.8.1.3 Boundaries of equi-tailed 95% confidence interval

We can use this approach for any confidence level $1 - \alpha$.

To obtain an equi-tailed confidence interval of level $1 - \alpha$ we set `z = qnorm(1 - alpha/2)` because $(1 - \alpha/2) - \alpha/2 = 1 - \alpha$.

For $\alpha = 0.05$, $1 - \alpha/2 = 0.975$ and we get the typically used 1.96 factor:

```
qnorm(0.975)
```

```
## [1] 1.959964
```

8.8.2 Confidence intervals in R

Most statistical tests in R provide confidence intervals for the relevant statistics. This is reported as part of the returned test object. For example, for the binomial test we get:

```
mu <- 0.45
N <- 1000
x <- sample(c(0, 1), size = N, replace = TRUE, prob = c(1-mu, mu))
binom.test(sum(x), length(x))

##
## Exact binomial test
##
## data: sum(x) and length(x)
## number of successes = 421, number of trials = 1000,
## p-value = 6.537e-07
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
## 0.3901707 0.4522958
## sample estimates:
## probability of success
## 0.421
```

You can see that the binom.test function automatically gives us a 95 percent confidence interval. It is reported in the conf.int slot. So we can extract it using:

```
binom.test(sum(x), length(x))$conf.int
```

```
## [1] 0.3901707 0.4522958
## attr(),"conf.level")
## [1] 0.95
```

We can set the confidence level with the conf.level parameter. So if we want a 99% interval, we do:

```
binom.test(sum(x), length(x), conf.level=0.99)$conf.int
```

```
## [1] 0.3807435 0.4620192
## attr(),"conf.level")
## [1] 0.99
```

For some tests, you first need to set conf.int to TRUE to receive a confidence interval:

```
wilcox.test(growth_rate ~ genotype, data=dt, conf.int=TRUE)
```

```

## 
## Wilcoxon rank sum test with continuity correction
## 
## data: growth_rate by genotype
## W = 690, p-value = 2.264e-16
## alternative hypothesis: true location shift is not equal to 0
## 95 percent confidence interval:
## -2.537024 -1.753245
## sample estimates:
## difference in location
## -2.134131

```

Sometimes R will use more accurate estimations than the Normal approximation we have just described. Details can usually be found in the documentation.

8.8.3 Advanced: A note on overlapping confidence intervals

Consider again a scenario where we are comparing two groups, X and Y , in terms of their means, \bar{X} and \bar{Y} . Assume, for simplicity, that:

$$X \sim N(\mu_x, \sigma^2)$$

$$Y \sim N(\mu_y, \sigma^2)$$

with σ^2 known. Assume further that we have samples of each group of size $n_y = n_x = n$. We then know that:

$$\bar{x} \sim N\left(\mu_x, \frac{\sigma^2}{n}\right)$$

$$\bar{y} \sim N\left(\mu_y, \frac{\sigma^2}{n}\right)$$

We can now construct two analytical 95% confidence intervals, one for each mean. We use the same procedure as previously. We set up an interval:

$$\Pr\left(\bar{x} - z\frac{\sigma}{\sqrt{n}} \leq \mu_x \leq \bar{x} + z\frac{\sigma}{\sqrt{n}}\right) = 0.95$$

And rearrange to get:

$$\Pr\left(-z \leq \frac{\bar{x} - \mu_x}{\frac{\sigma}{\sqrt{n}}} \leq z\right) = \Pr(-z \leq Z \leq z) = 0.95$$

As before, $Z \sim N(0, 1)$ and thus we get $z \approx 1.96$.

We round $1.96 \approx 2$, to make the math nicer, yielding us the intervals:

$$\bar{x} \pm 2\frac{\sigma}{\sqrt{n}}$$

$$\bar{y} \pm 2 \frac{\sigma}{\sqrt{n}}$$

Now assume we want to test the null hypothesis that the true mean difference is zero. In the literature it is quite common practice to say that we reject this null hypothesis if and only if the two confidence intervals do not overlap.

It is important to note that this is *not* the same as constructing a 95 confidence interval for the difference in means $\bar{x} - \bar{y}$ and rejecting if and only if that interval does not include zero. The difference comes from how we add standard errors.

In our overlap test, we would reject whenever $\bar{y} + 2 \frac{\sigma}{\sqrt{n}} < \bar{x} - 2 \frac{\sigma}{\sqrt{n}}$ (assuming $\bar{y} < \bar{x}$). We can rearrange to get that we reject whenever:

$$4 \frac{\sigma}{\sqrt{n}} < \bar{x} - \bar{y}$$

Now let us construct a confidence interval for $\bar{x} - \bar{y}$. A basic property of normal random variables tells us that:

$$\bar{x} - \bar{y} \sim N \left(\mu_x - \mu_y, 2 \frac{\sigma^2}{n} \right)$$

Thus the correct confidence interval for $\bar{x} - \bar{y}$ is (using again $1.96 \approx 2$):

$$(\bar{x} - \bar{y}) \pm 2\sqrt{2} \frac{\sigma}{\sqrt{n}}$$

Thus we reject whenever:

$$2\sqrt{2} \frac{\sigma}{\sqrt{n}} < \bar{x} - \bar{y}$$

Now $2\sqrt{2} < 4$, thus this will reject more often than the “overlap” test. in other words, the overlap test is too conservative.

Nevertheless, this sort of overlap test is very often used in the literature. When you see it being used, or when you use it yourself, keep in mind that it generally will fail to reject more often than the confidence level indicates.

8.9 Discussion

In the last chapter, we saw Permutation-based testing, which is very general: we can use it to test for any ad hoc statistics such as mean difference, median difference and so on. However, in the beginning of this chapter, we saw that Permutation-based testing is computationally intensive and not always appropriate in the context of big data. This is why we discussed a number of analytical tests which can serve as alternatives, to compute P -values and confidence intervals.

Some of tests are parametric, i.e. they assume some parameterized function for the data distribution, leading to the null hypothesis is based on the parameters (for instance that two groups distribute according to the Gaussian distribution and that the means of the two groups are equal). We saw that for many scenarios, non-parametric tests exist that do not make little assumptions on the distribution functions. Examples of such non-parametric tests are the Fisher, the

Wilcoxon rank-sum test, and the Spearman rank-correlation test. In general, these should be preferred to their parametric counterparts, unless we have good reason to believe that the more restrictive assumptions of the parametric test are met. We can check distributional assumptions using Q-Q plots.

The two-variable tests we discussed assess in some ways the dependencies between variables. By themselves, they cannot tell whether these relationships are causal and, if so, what the direction of causality is (See Chapter 6). For instance, we may be tempted to conclude from the viral infection example that smoking is a cause of severe symptoms, but that study design unfortunately cannot guarantee this. People who smoke could, for example, on average be older, have a less healthy diet, or have other risk-factors, compared to non-smokers. In other words, there could be confounding variables not taken into account here.

8.10 Conclusion

By now you should know:

- the assumptions of the binomial test and how to apply it.
- how to recognize and apply all the 2-variable tests in Figure 8.3
- Interpret a Q-Q plot

In applications, always report:

- a significance assessment (P -value or confidence interval)
- a plot showing the effect assessed
- (if relevant) a Q-Q plot showing that the distributional assumptions of the test are met

8.11 Resources

The Testing Chapter of Modern Statistics for Modern Biology, by Holmes and Huber.

An online version is available at: <https://www.huber.embl.de/msmb/>

14. R typically provides 4 functions per distribution starting with the letters r, d, p, and q and standing for random draws (rbinom, rnorm,...), density or probability mass (dbinom, dnorm,...), cumulative distribution (pbinom, pnorm,...), and quantile (qbinom, qnorm,...) ↪
15. It would also not be ideal to use the binomial test on the smoker data by fixing the probability under the null to the probability estimated on the non-smokers, because that probability would be a noisy estimate ↪
16. More details at https://en.wikipedia.org/wiki/Hypergeometric_distribution ↪
17. https://en.wikipedia.org/wiki/Mann%20Whitney_U_test ↪
18. https://en.wikipedia.org/wiki/Anscombe%27s_quartet ↪
19. https://en.wikipedia.org/wiki/Multivariate_normal_distribution ↪

20. In some cases, we may also be interested in non-monotonic data (e.g. a U-shaped relationship). In this case, both Pearson and Spearman correlations will fail and we have to use more complex measures, such as the distance correlation (https://en.wikipedia.org/wiki/Distance_correlation) or an information-theoretic measure such as mutual information. ↵
21. always give a plot along with your stats! ↵
22. This being said, the test rightfully warns of the large number of ties, so in this case the Spearman P -value may not be the last word on the matter either. With data like this, we should always take a step back and think carefully whether it can really answer the questions we are trying to ask of it. ↵
23. simulated with the Negative binomial distribution `rnbino(n, mu=100, size=2)/100` . ↵

Chapter 9 Statistical Assessments for Big Data

In the last two chapters, we took an in depth look at how to assess the statistical significance of patterns in our data. We know how to compute and interpret P -values to test specific hypotheses in the context of relatively small datasets.

In this chapter, we will look at what happens when we move to a world where we have large datasets, and are not sure beforehand which specific hypothesis we want to test. After reading this chapter, you should:

- Understand what an **effect size** is
- Be able to explain the **difference between statistical significance and practical importance**
- Understand the problem of **multiple testing**
- Define the terms **FWER** and **FDR**
- Apply the **Bonferroni** and **Benjamini-Hochberg corrections** and understand the difference between them

9.1 Motivation: Statistical Significance in a Big Data context

Imagine a huge consortium of scientists gathers a variety of different demographic, social and health data from millions of Germans. After many days of analysis, they report that:

"There is a statistically significant association ($P < 0.05$) between drinking coffee and all-cause mortality in university students aged 19-25 whose birthday falls in an odd-numbered month"

If you are a coffee drinking university student aged 19-25 who was born in an odd-numbered month, should you be worried?

In this chapter, we will see some reasons why the sentence above - without further information - is not a convincing reason to swear off Espresso forever.

We will look at two major issues stemming from big data applications. The first issue is the issue of **small effect size detection**, which is a consequence of having **large sample sizes** ("**large n**", referring to the numbers of **rows of a data matrix**). The second issue is the issue of **multiple testing**, which is consequence of assessing many variables ("**large p**", referring to the **number of columns of a data matrix**).

9.2 Effect Size: Actually important or just significant?

The first issue we encounter in a big data world is the problem that statistical significance does not necessarily reflect effects of practical relevance. To understand this distinction, we first need to introduce the notion of effect size.

The **effect size** is a quantitative measure of the **magnitude of a phenomenon**. Examples of effect sizes are the **correlation** between two variables, the **regression coefficient** in a regression, the **mean difference**, or even the risk with which something happens, such as how many people survive after a heart attack for every one person that does not

survive.

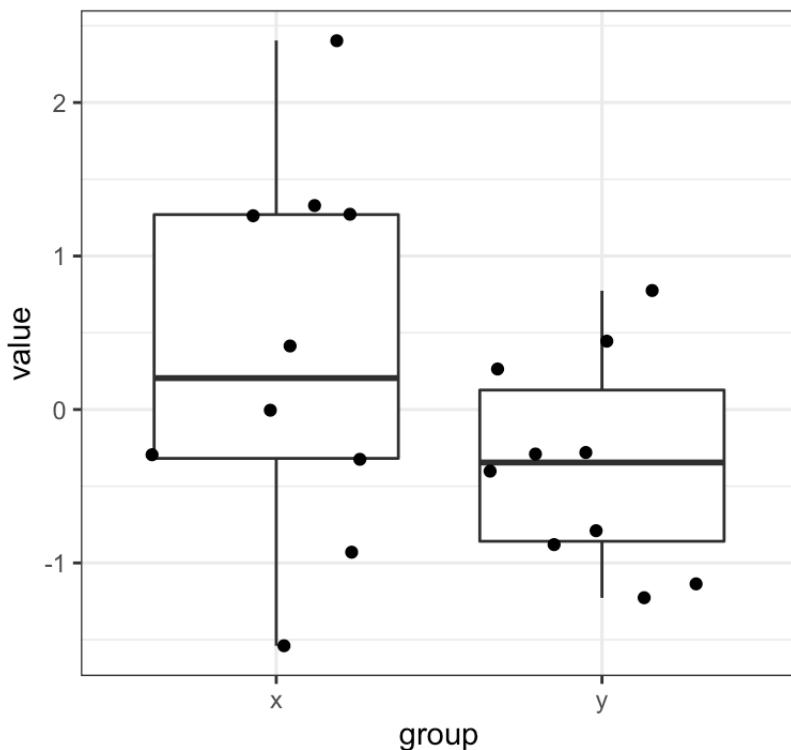
The effect size determines whether an effect is actually important. If we discover that drinking one cup of coffee per day decreases life expectancy by two years on average, this is a substantial effect with large implications for millions of people. If, by contrast, we find that life expectancy is only decreased if you drink 20 cups or more per day, then this is largely irrelevant, as probably no sane person would do this.

9.2.1 The relationship of sample size and significance

Assume now that we have a very small effect size. For instance, assume we have two groups, X and Y , that are both normally distributed with variance 1. But $E[X] = \mu_x = 0$ whereas $E[Y] = \mu_y = 0.01$. We sample from these distribution and do a t -test to try to reject the null hypothesis that $E[X] = E[Y]$ (i.e. the effect size is exactly 0).

If our sample size is small, e.g. $n_x = n_y = 10$, the difference in means is unlikely to be statistically significant.

```
mu_x  <- 0
mu_y  <- 0.01
n <- 10
x <- rnorm(n, mu_x)
y <- rnorm(n, mu_y)
dt <- data.table(
  value = c(x,y),
  group =rep(c("x", "y"), each=n)
)
ggplot(dt, aes(x=group, y=value)) +
  geom_boxplot() +
  geom_jitter() +
  ggtitle(paste0("n=",n)) + mytheme
```

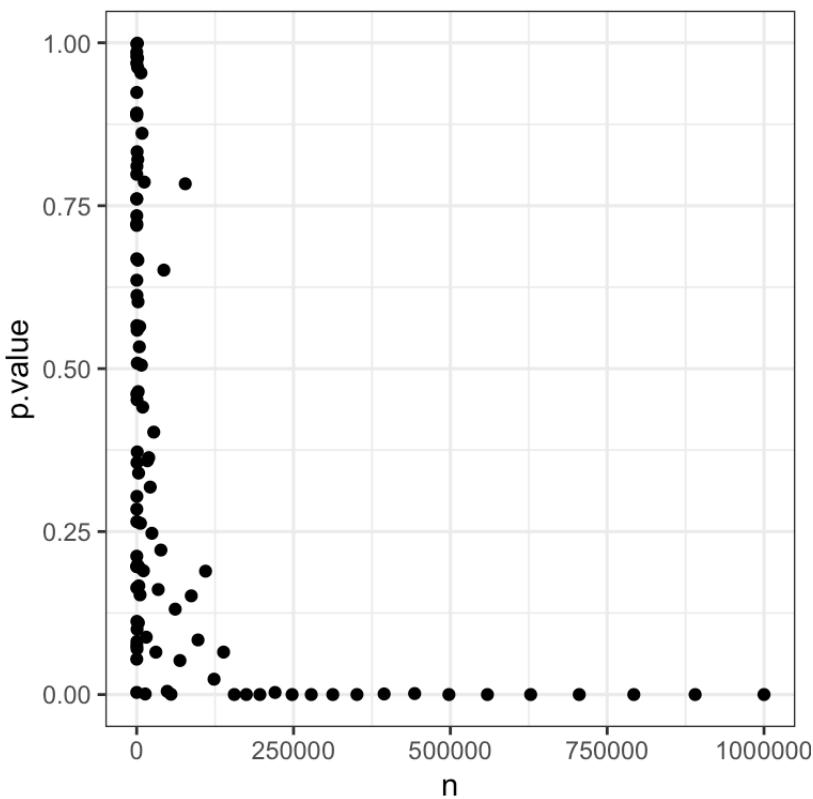
n=10

```
t.test(x,y)$p.value
```

```
## [1] 0.1259009
```

But things change as we start to increase the sample size:

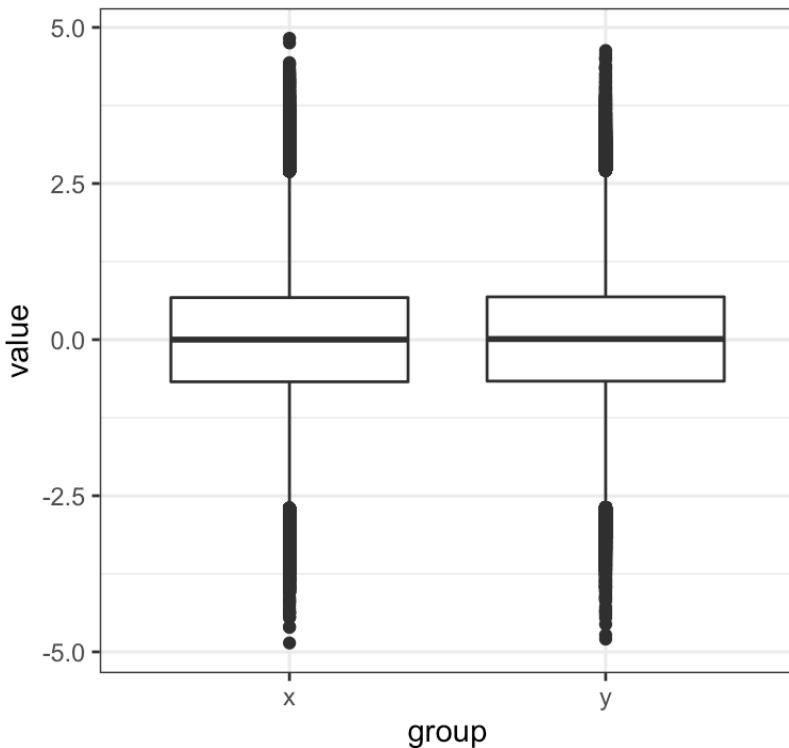
```
ns <- round(10^seq(1,6, length.out=100)) # a exponential grid of values of n
pvs <- sapply( ns, # for each N...
  function(n){
    x <- rnorm(n, mu_x) # draw N values of x according to N(mu_x, 1)
    y <- rnorm(n, mu_y) # draw N values of y according to N(mu_y, 1)
    t.test(x, y)$p.value # Welch t-test
  }
)
data.table(n=ns, p.value=pvs) %>%
  ggplot(aes(x=n, y=p.value)) + geom_point() + mytheme
```



We see that as the sample size gets bigger and bigger, the P -values quickly drop.

Once we have a million observations, we can easily reject the null hypothesis that the true effect is zero:

$n=1e+06, P=5e-12$



The conclusion is that, as the sample size grows, any true effect, no matter how small, will be detected.

This creates a problem. In reality, the null hypothesis, will seldom be literally true. Because we live in a complex and interconnected world, many variables will be somewhat correlated, even if the correlation is very weak. The effect size will rarely be exactly 0.0.

This means, if our dataset is big enough (i.e. many hundreds of thousands of datapoints), it is likely that whatever association we test, it will be significant. But many of these associations will likely be too small to be of any practical importance.

9.2.2 Report P-value, effect size, and plot

Because of this, only reporting that an association is statistically significant is misleading and not particularly helpful. To avoid this problem, we should always report the P -value, an estimate of the effect size and a plot of our data that supports our conclusions.

9.3 Multiple Testing

In addition to the problem of detecting very small effects, big data also poses the problem of detecting false effects due to multiple testing. We will first discuss why multiple testing can undermine the conclusions we derive from data. Then we will explore ways to guard against being misled in this way, using a simulated dataset as an example.

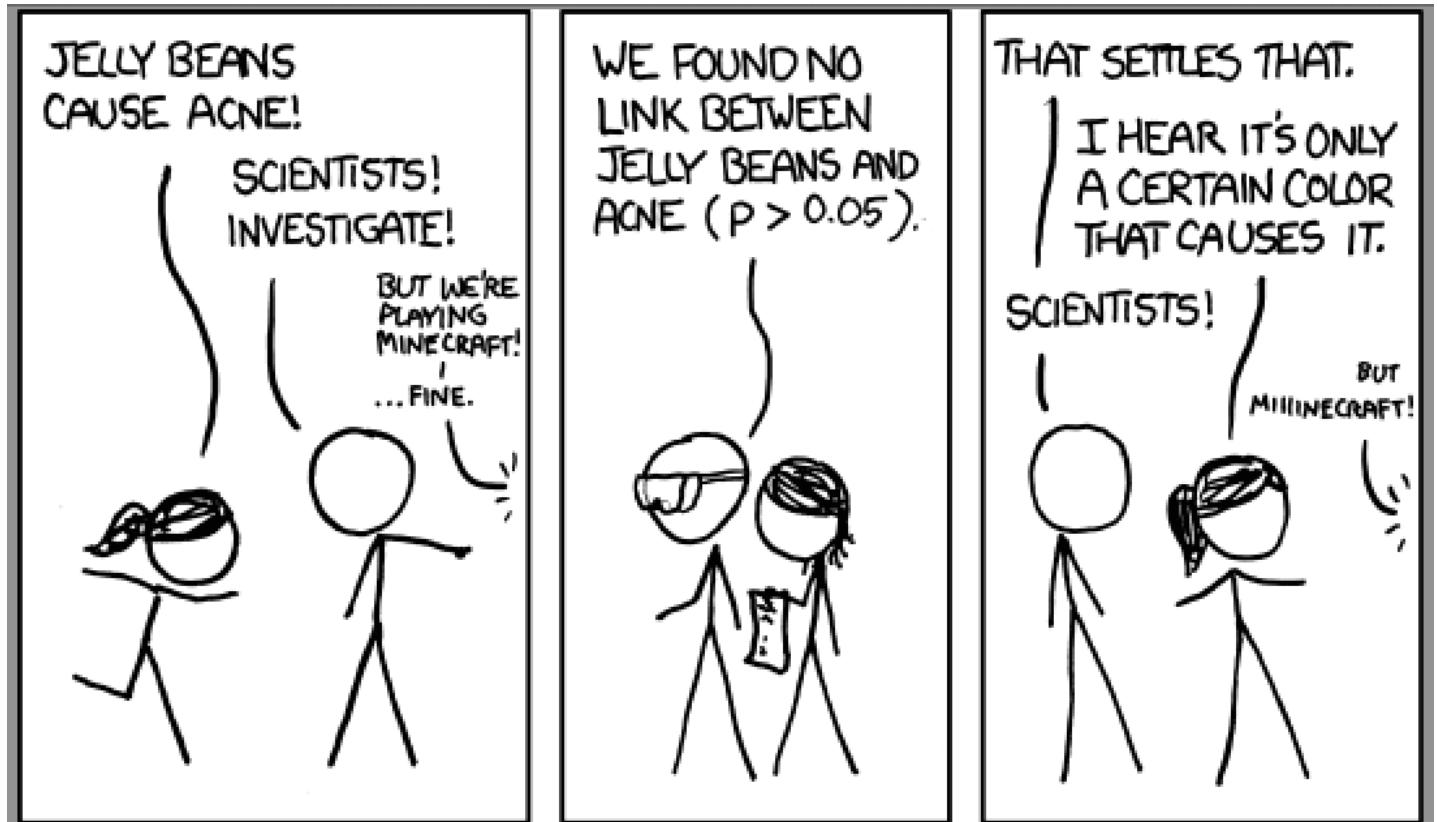
9.3.1 Multiple testing in real life: p-Hacking and fishing expeditions

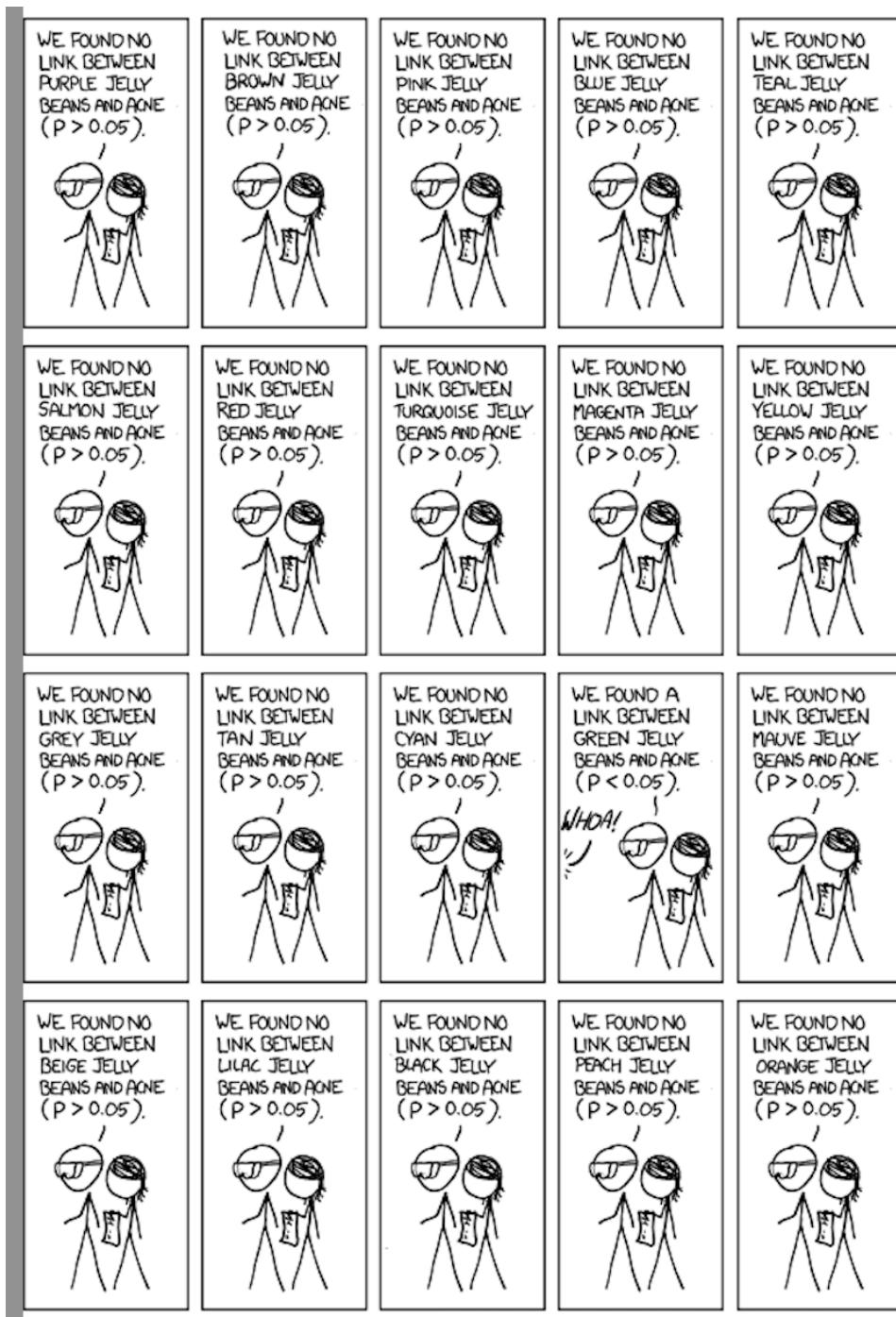
Expressed very simply, multiple testing refers to the issue that, if we test enough hypotheses at a given significance level, say $\alpha = 0.05$, we are bound to eventually get a significant result, even if the null hypothesis is always true.

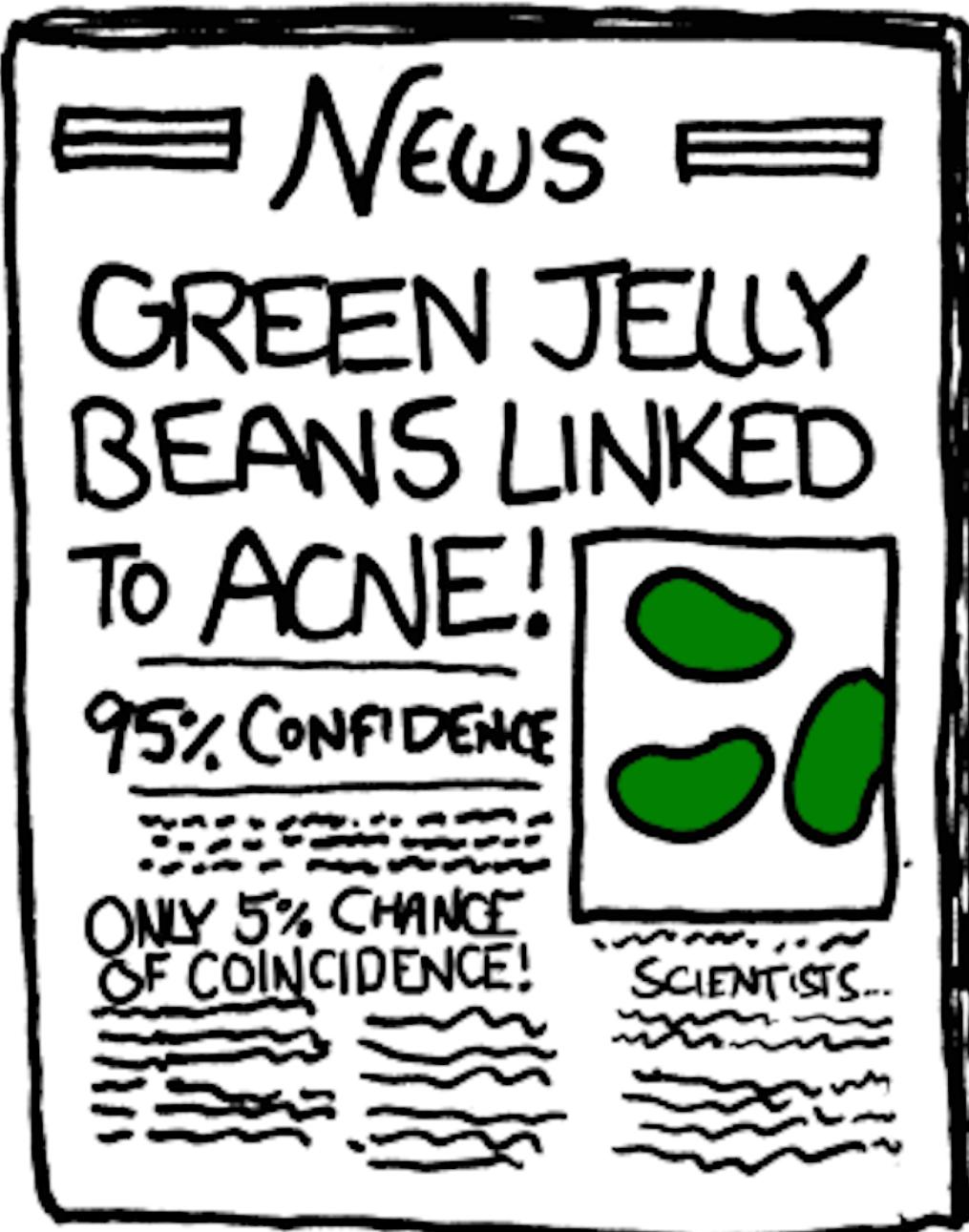
Multiple testing is an ever-present problem also in real life data analysis. It appears particularly once we start stratifying large datasets into subgroups, testing a separate hypothesis for each one.

Say we want to test the effect of a new medication. We find it has no effect in the general population. But maybe it only works in women? Maybe it affects people of different ages differently? Maybe smokers metabolize it differently to non-smokers? If our data is large enough and we keep stratifying it into subgroups, we are bound to find one where $P < 0.05$, even if our medication is completely useless.

The following xkcd comic illustrates the issue in a humorous way:







This phenomenon is often referred to as " P -hacking" or fishing expedition. It refers to the practice of testing different hypotheses until one finally finds one that yields a significant P -value. For more detail on this issue, the following article by FiveThirtyEight gives a good overview. In particular, you can interactively "Hack Your Way To Scientific Glory".

This is not to say that stratifying data and analyzing it in different ways is a bad thing. But it does mean that, when we do so, we should make sure that we apply a correction that accounts for the multiple tests we have performed.

9.3.2 The Land of Counterfeit (fake) coins

To see multiple testing in action on a concrete example, and to explore ways to correct for it, we simulate a dataset based on flipping coins. We will tell a little fairytale to make the details less dry:

Long, long ago, in a country far far away, a king suspected counterfeit coins were circulating in his kingdom. Original coins were fair: When tossed, each coin has an equal chance to come up heads or tails. Counterfeits instead were more biased towards more heads. The king requested each one of his 1,000 subjects to toss his or her coin (subjects owned only one coin) 20 times.

9.3.3 Simulation

Here we simulate data for this scenario. We (not the king/queen) know that in reality 90% of the subjects own a fair coin. For simplicity, we assume that all counterfeits have probability 0.8 for the head.

```
library(data.table)
library(ggplot2)
library(magrittr)
library(ggbeeswarm)

set.seed(10)
m <- 1e3
m0 <- 0.9*m
n <- 20
fair <- c(rep("fair", m0), rep("biased", m-m0))
heads <- rbinom(m,n,ifelse(fair=="fair",0.5,0.8))

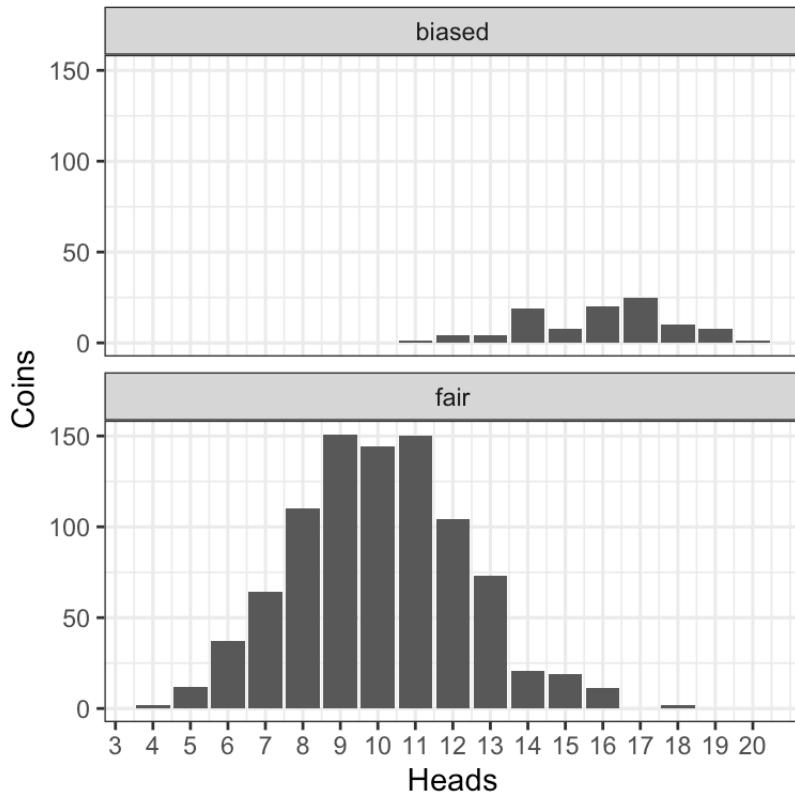
dt <- data.table(fair=fair, heads=heads)

dt

##          fair heads
## 1:    fair    10
## 2:    fair     9
## 3:    fair    10
## 4:    fair    11
## 5:    fair     7
## --- 
## 996: biased   15
## 997: biased   18
## 998: biased   16
## 999: biased   18
## 1000: biased  12
```

We look now at the distribution of heads for the fair and the biased coins.

```
dt2 <- dt[, .N, by=.(fair,heads)]  
dt2 %>% ggplot(aes(heads,N)) +  
  geom_bar(stat="identity") +  
  facet_wrap(~ fair, nrow=2) +  
  scale_x_continuous("Heads", 0:n) +  
  scale_y_continuous("Coins") +  
  mytheme
```



9.3.4 Nominal p-values

The king declares: "I presume innocence, i.e. I will consider the null hypothesis that each coin is fair. I will reject this hypothesis using a one-sided binomial test at p-value ≤ 0.05 ."

The king computes the p-values.

```
dt[,  
  p.value := sapply(  
    heads,  
    function(x)  
      binom.test(x, n = n, alternative = "greater")$p.value  
  )  
]  
dt
```

```

##          fair heads      p.value
## 1:    fair     10 0.5880985260
## 2:    fair      9 0.7482776642
## 3:    fair     10 0.5880985260
## 4:    fair     11 0.4119014740
## 5:    fair      7 0.9423408508
## ---
## 996: biased    15 0.0206947327
## 997: biased    18 0.0002012253
## 998: biased    16 0.0059089661
## 999: biased    18 0.0002012253
## 1000: biased   12 0.2517223358

```

We can compute the number of rejected tests:

```
sum(dt$p.value<=0.05)
```

```
## [1] 104
```

The king is furious. There are 104 declared guilty. He requests the death penalty for all of them.

9.3.4.1 Explanation: nominal P -values can lead to many false rejections when testing multiple hypotheses

Since we (simulators) know the truth, we can compute the contingency table of rejected tests versus the ground truth:

```
table(dt[,. (fair, rejected = p.value<=0.05)])
```

	rejected	
	FALSE	TRUE
fair	28	72
biased	868	32

We see that this will mean that **32 innocent subjects will be punished**, despite the fact that the King used rigorous statistical testing against a null hypothesis of innocence to establish guilt. What went wrong?

To explore what is happening here, we first need to note a very important fact about P -values: under the null hypothesis, we have that $p(P < \alpha | H_0) = \alpha$.

To see why, let us first recall the definition of the **P -value**: it is the **probability of obtaining a test statistic the same as or more extreme than the one we actually observed, under the assumption that the null hypothesis is true**. Assume, for simplicity, we are doing a **right-tailed one-sided test**. The P -value is:

$$P = p(T \geq T_{\text{obs}} | H_0)$$

Let $T_{0.05}$ be a test statistic just big enough so that:

$$0.05 = p(T \geq T_{0.05} | H_0)$$

Now, given this definition, what is the probability under the null hypothesis of getting $P < 0.05$?

Well, to get a P -value smaller than 0.05, we need to observe a test statistic T bigger than $T_{0.05}$. And we know that under the null hypothesis the probability of that happening is:

$$p(T \geq T_{0.05} | H_0)$$

which we just said is equal to 0.05! So $p(P < 0.05 | H_0) = 0.05$, and the same logic generalizes for any other $\alpha \in [0, 1]$.

This, in turn, means that when we test 900 fair coins and reject the null hypothesis that the coin is fair whenever $P < 0.05$, then on average we will reject the null hypothesis about 5% of the time, so $900 * 0.05 = 45$ times. Thus, in some sense the King actually got lucky that he “only” falsely accused 72.

Another way of stating this fact is to say that, under the null hypothesis, the P -value follows a uniform distribution on the $[0, 1]$ interval. Figure 9.1 illustrates the P -value distribution, compared to the $[0, 1]$ uniform distribution, for the fair and for the biased coins:

```
ggplot(dt, aes(sample = p.value)) +
  # we make a qqplot comparing the p-values with the [0,1] uniform
  geom_qq(distribution = stats::qunif) +
  # we add a diagonal
  geom_abline(slope=1, intercept=0) +
  # we make the plot separately for fair and biased coins
  facet_wrap(~fair) +
  mytheme
```

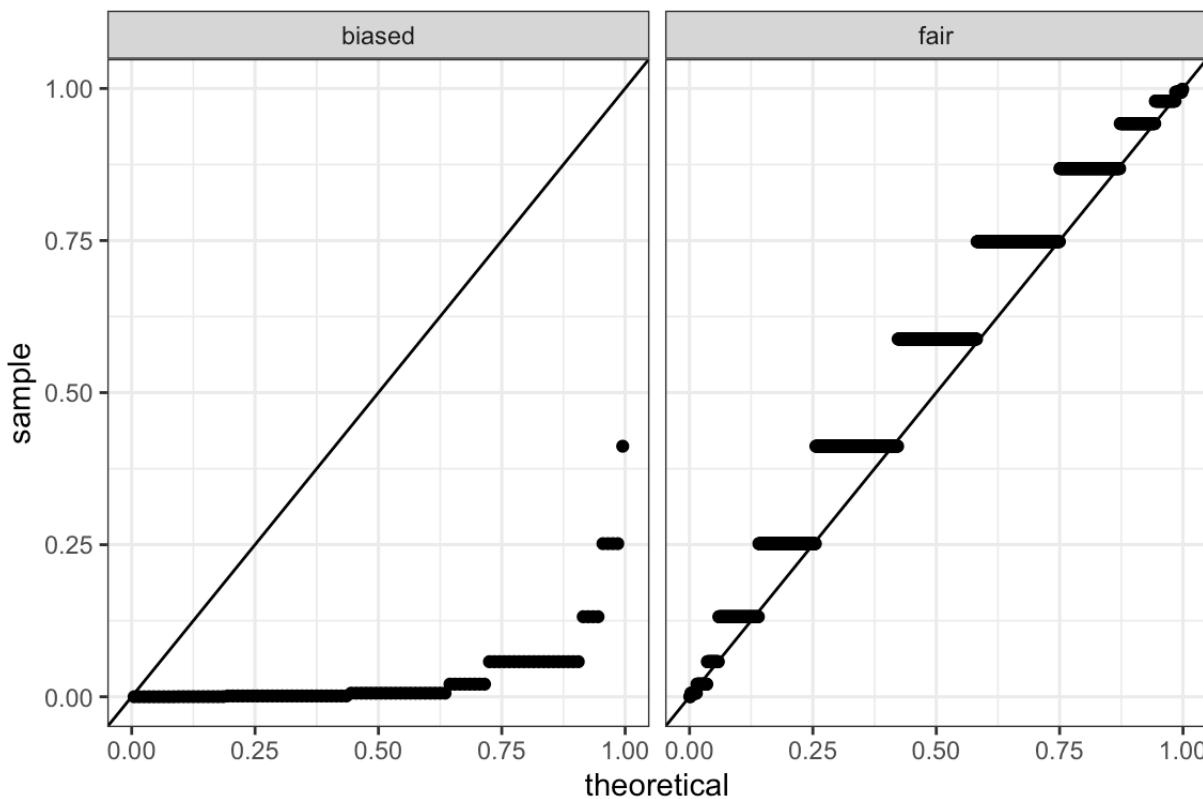


Figure 9.1: P-value Q-Q plots against the uniform distribution for the fair and the biased coins.

We see that the P -values for the fair coins match the uniform distribution quite well (values along diagonal). In contrast, the p -values for the biased coins are excessively small compared to the uniform distribution. This is exactly what we want: **small p -values when the null hypothesis does not hold.** (Note that the distribution of p -values for discrete statistics actually shows these stairs due to ties. One could visually fix it but this is not a major point).

9.3.5 Family-wise error rate

The queen, who is well-versed in matters of probability, is rightfully appalled at the prospect of many innocent subjects being executed.

She states: “If they were all innocent, you would declare $0.05 * 1000 = 50$ subjects guilty and falsely sentence them to death. We shall not accept that **any** innocent person is killed”.

King: “Mmmh, good point. But we cannot be 100% sure. What if I make sure that there is **at most** 5% chance that an **innocent person is mistakenly killed?**”

Queen: “Fine.”

Now the King must control such that the probability that one or more innocent is killed is smaller than 5%. He uses Bonferroni correction, which amounts to multiplying the P -values by $m=1,000$ or setting them to 1, whichever is smaller (because probability can't be larger than 1).

```
dt[, bonferroni := p.adjust(p.value, method="bonferroni")]
dt
```

```

##      fair heads      p.value bonferroni
## 1:   fair    10 0.5880985260 1.0000000
## 2:   fair     9 0.7482776642 1.0000000
## 3:   fair    10 0.5880985260 1.0000000
## 4:   fair    11 0.4119014740 1.0000000
## 5:   fair     7 0.9423408508 1.0000000
## ---
## 996: biased   15 0.0206947327 1.0000000
## 997: biased   18 0.0002012253 0.2012253
## 998: biased   16 0.0059089661 1.0000000
## 999: biased   18 0.0002012253 0.2012253
## 1000: biased   12 0.2517223358 1.0000000

```

Here is the new contingency table. In this case, no innocent person was killed (He did kill 9 guilty people). It could have happened, though, but with less than 5% chance.

```
table(dt[,(fair, rejected = bonferroni<=0.05)])
```

```

##      rejected
## fair      FALSE TRUE
## biased    91   9
## fair      900   0

```

9.3.5.1 Explanation: the Bonferroni correction is a very conservative method to address multiple testing

To understand how the correction the King applied works, and why it makes sense, we first introduce some notation:

	Not rejected	Rejected	Total
True null hypotheses	U	V	m_0
False null hypotheses	T	S	$m - m_0$
Total	$m - R$	R	m

The Queen wants to keep V , the number of true null hypotheses which were falsely rejected (leading to an innocent being executed), as low as possible. The King can do that by controlling the so-called Family-wise error rate:

Family-wise error rate (FWER): $p(V > 0)$, the probability of one or more false positives (i.e. the probability of one or more innocent being executed).

Specifically, the King has opted to keep the FWER below $\alpha = 0.05$.

We have just established that, under the null hypothesis, if we do m tests, and reject if $P < \alpha$, then we will falsely reject about $m\alpha$ times. To control the FWER, the so-called **Bonferroni correction** is a lot more demanding, and only reject if $P < \frac{\alpha}{m}$.

More formally, suppose we conduct our m hypothesis tests for each $g = 1, \dots, m$, producing each a p-value P_g . Then the Bonferroni-adjusted P -values are:

$$\tilde{P}_g = \min\{mP_g, 1\}$$

Selecting all tests with $\tilde{P}_g \leq \alpha$ controls the FWER at level α , i.e., $p(V > 0) \leq \alpha$.

In R we can adjust the P -values using:

```
p.adjust(p_values, method = "bonferroni")
```

The proof goes as follows. We assume to have applied Bonferroni correction, i.e. that we have rejected all hypotheses i such that $P_i < \frac{\alpha}{m}$. We then start from the definition: $\text{FWER} = p(V > 0)$ which is the probability that at least one of the true null hypotheses got rejected:

$$\begin{aligned}\text{FWER} &= p(V > 0) \\ &= p(\{H_{0,1} \text{ rejected}\} \cup \{H_{0,2} \text{ rejected}\} \cup \dots \cup \{H_{0,m_0} \text{ rejected}\})\end{aligned}$$

Now, among a set of binary random events (here rejection, or not, of each true null hypothesis), the probability that at least one random event realizes is less than or equal to the sum of the probabilities of each random event to realize.²⁴ It would be equal to the sum, only if those events were mutually exclusive. We cannot exclude the possibility that rejections of the true null hypotheses are mutually exclusive although in practice this is a very conservative assumption. So, conservatively, we have the following upper bound:

$$\text{FWER} \leq \sum_{i=1}^{m_0} p(H_{0,i} \text{ rejected})$$

What is $p(H_{0,i} \text{ rejected})$ when $i = 1 \dots m_0$, i.e. when i is the index of a true null hypothesis? Having applied Bonferroni's correction means that, for all i :

$$p(H_{0,i} \text{ rejected}) = p\left(P_i \leq \frac{\alpha}{m}\right)$$

Moreover, by definition of the p-value, we have that $p(P_i \leq \frac{\alpha}{m}) = \frac{\alpha}{m}$ for all $i = 1 \dots m_0$, as these are the indices of the true null hypotheses. Hence we get:

$$\text{FWER} \leq m_0 \frac{\alpha}{m} \leq \alpha$$

This shows that the Bonferroni correction controls the FWER at the level α .

This control does not require any assumption about dependence among the P -values or about how many of the null hypotheses are true. The drawback is that it is very conservative. If we run 1,000 tests at $\alpha = 0.05$, we will only reject if the (unadjusted) P -value is smaller than 0.00005.

9.3.6 False Discovery Rate

The king sleeps on it and comes back angry to the queen.

King: " If they were all innocent as you assumed, there should have been 50 with a p-value less than 0.05 in the first place. But they were instead 104. There are still a lot of counterfeiters unpunished. What if I introduced a fine rather than the death penalty?"

Queen: "That is more reasonable. It would be tolerable that **no more** than 5% of fined subjects are **actually innocent.**"

The king now needs to control for the proportion of innocent people among the subjects called guilty (rejected = True).

He uses the **False Discovery Rate with Benjamini-Hochberg correction**. This controls the expected value of that proportion.

```
dt[ , BH := p.adjust(p.value, method="BH")]
dt

##          fair heads      p.value bonferroni      BH
## 1:    fair     10 0.5880985260 1.0000000 0.942465587
## 2:    fair      9 0.7482776642 1.0000000 0.965519567
## 3:    fair     10 0.5880985260 1.0000000 0.942465587
## 4:    fair     11 0.4119014740 1.0000000 0.858128071
## 5:    fair      7 0.9423408508 1.0000000 0.992982983
## ---
## 996: biased    15 0.0206947327 1.0000000 0.198987814
## 997: biased    18 0.0002012253 0.2012253 0.009582156
## 998: biased    16 0.0059089661 1.0000000 0.076739819
## 999: biased    18 0.0002012253 0.2012253 0.009582156
## 1000: biased   12 0.2517223358 1.0000000 0.765113483
```

Here is the new contingency table:

```
table(dt[,(fair, rejected = BH<=0.05)])
```

	rejected
fair	FALSE TRUE
biased	56 44
fair	898 2

A higher number of counterfeiters were discovered, while the proportion of innocents among the subjects declared guilty (2 out of 46) remained lower than about 5%.

9.3.6.1 Explanation: the Benjamini-Hochberg correction controls the False Discovery Rate

When applying the Bonferroni correction, we took a very conservative view: we created a worst-case scenario where **everyone** was innocent, and then minimized the probability of innocents being executed based on this assumption.

But everyone in the kingdom can see that this worst-case scenario is not correct by looking at the Q-Q plot (this time including all subjects, and not split by the innocent ones and the non-innocent):

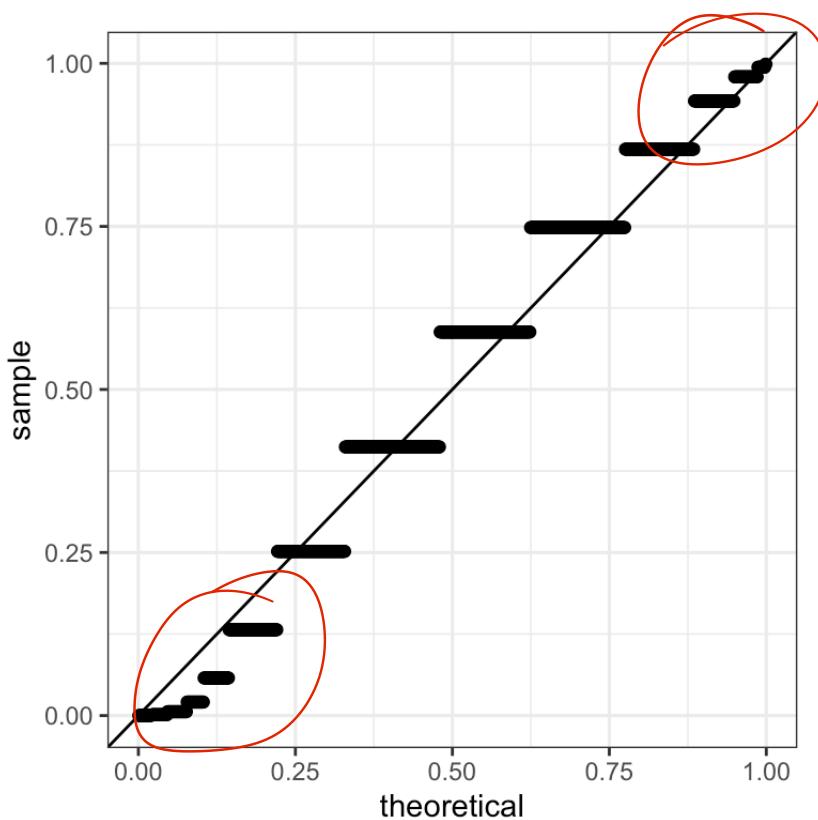


Figure 9.2: P-value Q-Q plots against the uniform distribution for the entire population.

We notice a **deviation from the assumption of uniformity at the low end of the distribution**. This is due to the presence of the guilty subjects, whose coins do not obey the null hypothesis (the entire population is a mixture of fair and unfair coins, whose respective Q-Q plots are shown in Figure 9.1). If we apply the Bonferroni correction, very few of these will be punished. Controlling the Family-wise error rate **ensures we have few false positives, but it comes at the cost of many false negatives**.

Instead of conservatively minimizing the probability of false positives, the **Benjamini-Hochberg correction**²⁵ takes a more balanced view and instead controls the so-called **False discovery rate (FDR)** defined as the **expected fraction of false positives among all discoveries**:

$$\text{FDR} = E\left[\frac{V}{\max(R, 1)}\right]$$

, where $\max(R, 1)$ ensures the denominator to not be 0.

To do this, we first order our (unadjusted) P -values. Let the ordered unadjusted P -values be: $P_1 \leq P_2 \leq \dots \leq P_m$. Let j refer to a rank in this ordering, so P_j is the j -th smallest P -value, and $H_{0,j}$ is the respective null hypothesis.

To control the FDR at level α , we let:

$$j^* = \max\{j : P_j \leq \frac{j\alpha}{m}\}$$

And we reject $H_{0,j}$ if $j \leq j^*$.

Why does this control the FDR?

We say there are m_0 hypotheses where $H_{0,j}$ is true (in our example $m_0 = 900$, out of $m = 1000$ tests we conduct). As we discussed, when H_0 is true, the corresponding P -value follows a uniform distribution (assuming independence).

This means that on average $m_0 \frac{j^* \alpha}{m}$ P -values of true null hypotheses will fall into the interval $[0, \frac{j^* \alpha}{m}]$. In other words, we expect:

$$m_0 \frac{j^* \alpha}{m}$$

false rejections. Overall, we reject j^* times. Thus we get an FDR of:

$$\frac{m_0 \frac{j^* \alpha}{m}}{j^*} = \frac{m_0}{m} \alpha \leq \alpha$$

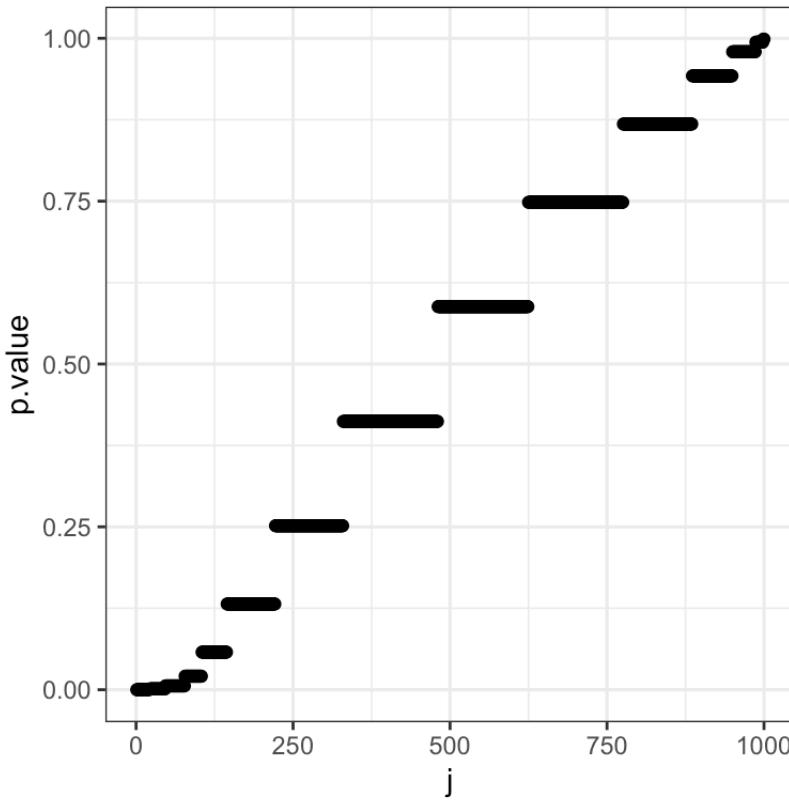
The Benjamini-Hochberg correction is valid for independent test statistics and some types of dependence²⁶

In R we do:

```
p.adjust(p_values, method = "BH")
```

We can also do the Benjamini-Hochberg correction graphically. For this we plot the rank j of P -values against their actual value:

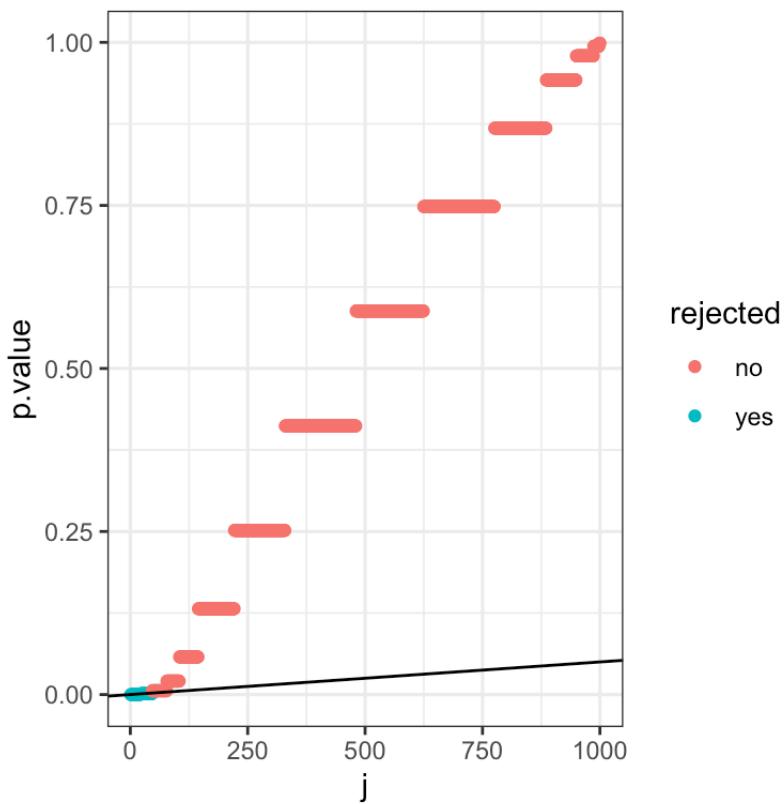
```
dt_rank <- data.table(p.value = sort(dt$p.value), j = 1:1000)
ggplot(data=dt_rank, aes(x=j, y=p.value)) + geom_point() + mytheme
```



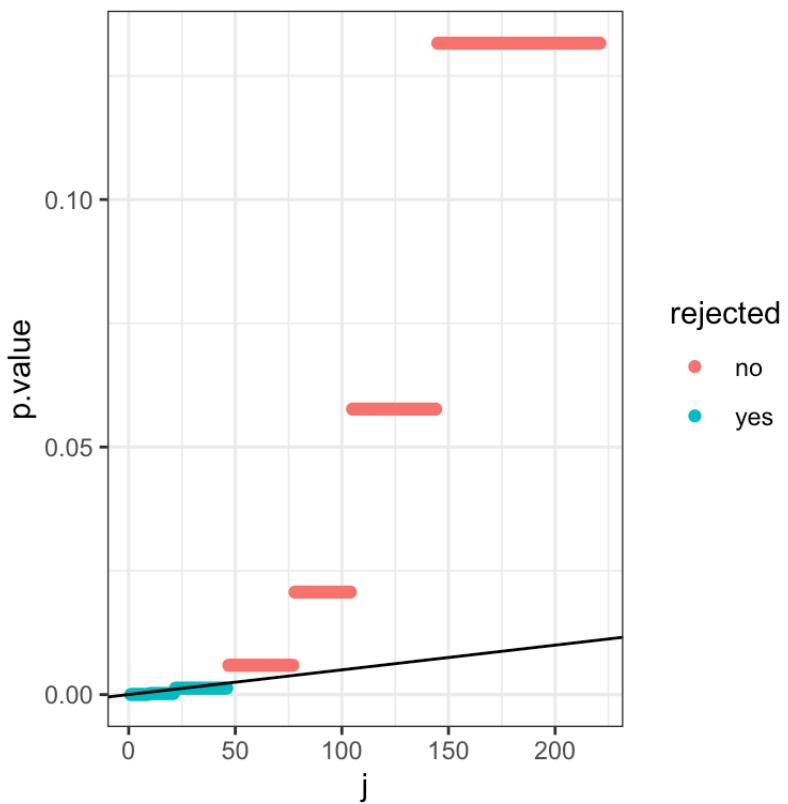
Note the visual similarity with the Q-Q plot.

Next we draw a line with slope $\frac{\alpha}{m}$ and find the largest P -value that falls below this line. We then reject for this test and all those with smaller P -values:

```
dt_rank[ , BH := p.adjust(p.value, method="BH")]
dt_rank[ , rejected := ifelse(BH<=0.05, "yes", "no")]
ggplot(data=dt_rank) + geom_point(aes(x=j,y=p.value,color=rejected)) +
  geom_abline(aes(intercept=0, slope=0.05/1000)) +
  #geom_label(aes(x= 150, y=0.04,label="Slope: (j*alpha)/m")) +
  mytheme
```

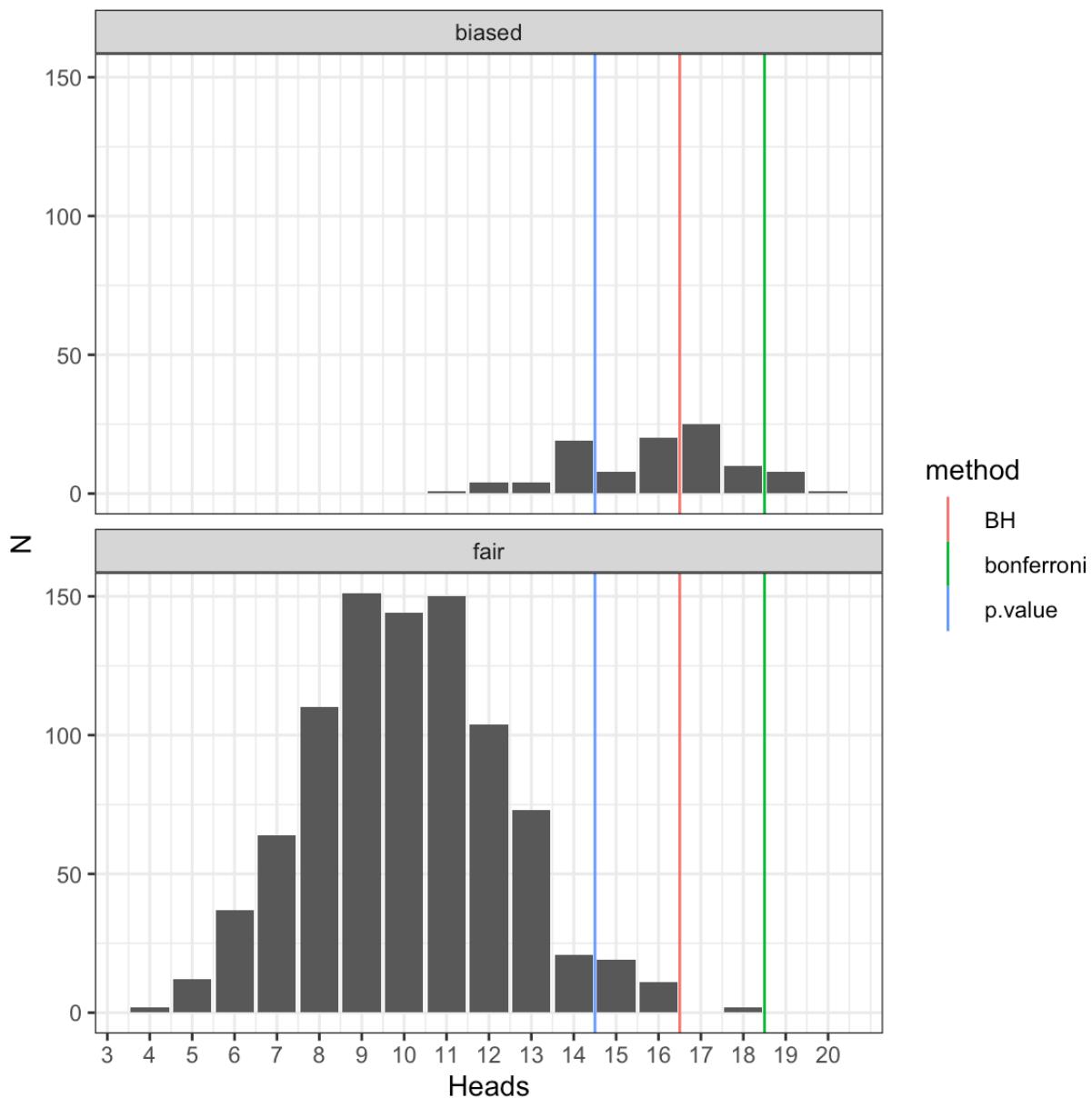


We zoom in a bit on the lower quadrant to make it more clear:



9.3.7 Overview figure

The following figure compares the methods once more:



As expected, we see that the nominal p-value cutoff is the most lenient, the FWER one (Bonferroni) the most stringent and the FDR ones (Benjamini-Hochberg) is intermediate.

9.4 Conclusions

In this chapter, we saw some reasons why we need to be careful when we apply hypothesis testing procedures in a big data context. Specifically we saw that as the sample size increases, even very small effects may become significant, but that does not mean that they actually *matter*. Moreover, we saw that when we run many tests, some are bound to reject the null, even if the null is always true. We thus need to apply a correction.

Given what we learnt, let us return to the original question:

Scientists report that:

"There is a statistically significant association ($P < 0.05$) between drinking coffee and all-cause mortality."

If you are a coffee drinking TUM student aged 19–25 who was born in an odd-numbered month, should you be worried?

We now know that we should be skeptical, and ask two questions:

- What is the effect size? After all, there are many thousands of university students in Germany. If enough of them were sampled, then even a small effect may have been detectable
- Was a multiple testing correction applied? The statement refers to some very specific subgroups (by occupation, by age, by birthday), so likely this hypothesis was not the only one which was tested. Thus the mere fact that $P < 0.05$ is not convincing.

9.4.1 To remember

Now you should:

- Understand what an effect size is
- Be able to explain the difference between statistical significance and practical importance
- Understand the problem of multiple testing
- Define the terms FWER and FDR
- Apply the Bonferroni and Benjamini-Hochberg corrections and understand the difference between them

9.5 References

- Introduction to Data Science, Rafael Irizarry, Confidence intervals chapter
<https://rafalab.github.io/dsbook/confidence-intervals.html>
 - Modern dive An Introduction to Statistical and Data Sciences via R, Imay & Kim Confidence intervals and Hypothesis testing chapters <https://moderndive.com/index.html>
 - Modern Statistics for Modern Biology, by Holmes and Huber <https://www.huber.embl.de/msmb/>
-
24. Boole's inequality. See https://en.wikipedia.org/wiki/Boole%27s_inequality ↪
 25. Among the top 100 cited scientific papers of all times. <https://www.nature.com/news/the-top-100-papers-1.16224> ↪
 26. While Benjamini-Hochberg is widely used, the independence assumption is questionable in practice. A modified version of the correction, called Benjamini-Yekutieli (`p.adjust(..., method="BY")`), does not require independence and shall be favored in applications. It goes beyond the scope of this lecture (and exam). ↪

Chapter 10 Linear Regression

10.1 Motivation and overview

10.1.1 Testing conditional dependence

Up to this point, we have focused on testing associations between pairs of variables. However, in data science applications, it is very common to study three or more variables jointly.

For pairs of variables, say x and y , one variable may be considered as the explanatory variable (let us take x) and the other the response variable (y). Asking whether y depends on x amounts to ask whether the conditional distribution of y given x varies when x varies. For instance, one may state that body height depends on sex by showing that the distribution of body heights is not same among males than among females. In general statistical terms, testing for independence can be expressed as the null hypothesis:

$$H_0 : p(y|x) = p(y). \quad (10.1)$$

Let us assume now we are considering multiple explanatory variables x_1, \dots, x_p and a response variable y . We are still interested in the association of one particular explanatory variable, say x_j with the response variable y but shall take into consideration all the other explanatory variables. In other words, we would like to know whether y depends on x_j *everything else being the same*. For instance, we may observe that drinking coffee positively correlates with lung cancer, but, neither among smokers, nor among non-smokers, drinking coffee associates with lung cancer. In this example, everything else being same (i.e. for same smoking status), there is no association between lung cancer and drinking coffee. In this thought experiment, the association found in the overall population probably comes from the fact that smokers tend to be coffee drinkers. Generally, we ask *whether the conditional distribution of y given all explanatory variables varies when x_j alone varies*. Hence, to move beyond pairs of variables, we need a strategy to assess null hypotheses such as:

$$H_0 : p(y|x_1, \dots, x_j, \dots, x_p) = p(y|x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_p) \quad (10.2)$$

Considered so generally, i.e. for any type of variable y and for any number and types of explanatory variables x_1, \dots, x_p , the null hypothesis in Equation (10.2) is impractical because:

1. We need to be able to deal with any type of distribution: Gaussian, Binomial, Poisson, but also mixtures, or even distributions that are not functionally parameterized.
2. We need to be able to condition on continuous variables. Conditioning on categorical variables leads to obvious stratification of the data. However, how shall we deal with continuous ones? Shall we do some binning? Based on what criteria? Could the binning strategy affect the results?

3. Even when all variables are categorical, there is a combinatorial explosion of strata, as each stratum gets further split by every new added variables (say by smoking status, and further by sex, and further by diabetes,...). For instance, there are 2^p different strata for p binary variables. This combinatorial explosion makes the analysis difficult. Moreover the statistical power, which is driven by the number of samples in each stratum, gets drastically reduced.

10.1.2 Linear regression

This Chapter introduces linear regression as an effective way to deal with these three issues. Linear regression addresses these three issues by making the following assumptions:

- The conditional distribution $p(y|x_1, \dots, x_p)$ is a Gaussian distribution whose variance is independent of x_1, \dots, x_p , simplifying greatly issue number 1.
- The conditional expectation of y is a simple linear combinations of the explanatory variables, that is:

$$E[y|x_1, \dots, x_p] = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \quad (10.3)$$

We can think of Equation (10.3) as a simple score for which explanatory variables contribute in a weighted fashion and independently of each other to variations in the expected value of the response. For instance, one could model the expected body height of an adult as:²⁷

$$\begin{aligned} E[\text{height}|\text{sex, mother, father}] &= 165 + 15 \times \text{sex} + 0.5 \times (\text{mother} - 165) \\ &\quad + 0.5 \times (\text{father} - 180), \end{aligned}$$

where sex is 1 for male and 0 for female, and mother and father designate each parent's body height.

The linear model in Equation (10.3), can deal with continuous as well as discrete explanatory variables, solving the issue number 2. Moreover, because Equation (10.3) models independent additive contributions of the explanatory variables, it has just one parameter per explanatory variable. The effects of the explanatory variables do not depend on the value of the other variables. These effects do not change between strata. Hence, linear regression does not suffer from a combinatorial explosion of strata (issue number 3).

10.1.3 Limitations

These assumptions of linear regression makes the task easier, but how limiting are they?

The Gaussian assumption is limiting. We cannot deal with binary response variables for instance. The next chapter will address such cases.

The assumption that variance is independent of the strata, if violated, can be a problem for fitting this model as well as for statistical testing. We show how to diagnose such issue in Section 10.4.

The assumption of additivity in Equation (10.3) seems more limiting than it actually is. First, there are many real-life situations where additivity of the effects of the explanatory variables turns out to be reasonable. Moreover, there is always the possibility to pre-compute non-linear transformations of explanatory variables and include them to the model. For instance this model of expected body weight is a valid linear model with respect to body height cubed – it just needs the cube to be pre-computed²⁸:

$$E[\text{weight}|\text{height}] = \beta_0 + \beta_1 \text{height}^3$$

All in all, linear regression turns out to be in practice often reasonable.

10.1.4 Applications

Linear regression can be used for various purposes:

- To **test conditional dependence**. This is done by testing the null hypothesis:

$$H_0 : \beta_j = 0.$$

- To **estimate the effects of one variable on the response variable**. This is done by providing an estimate of the coefficient β_j .
- To **predict** the value of the response variable given values of the explanatory variables. The predicted value is then an estimate of the conditional expectation $E[y|x]$.
- To **quantify how much variation** of a response variable can be explained by a set of explanatory variables.

This Chapter explains first univariate linear regression using a historical dataset of body height. We will then go to the multivariate case using an example from baseball. Finally we will assess what is the practical impact of violations of the modeling assumptions, and how to diagnose them. A substantial part of the Chapter is based on an adaptation of Rafael Irizzary's book.

10.2 Univariate regression

10.2.1 Galton's height dataset

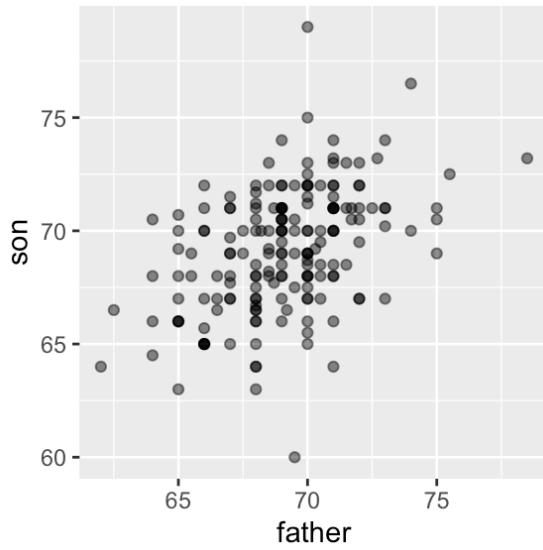
We start with **linear regression against a single variable, or univariate regression**. We use the dataset from which regression was born. The example is from genetics. Francis Galton²⁹ studied the variation and heredity of human traits. Among many other traits, Galton collected and studied height data from families to try to understand heredity³⁰. While doing this, he developed the concepts of correlation and regression, as well as a connection to pairs of data that follow a normal distribution. A very specific question Galton tried to answer was: how well can we predict a child's height based on the parents' height?

We have access to Galton's family height data through the **HistData** package. This data records height in inches from several dozen families: mothers, fathers, daughters, and sons. To imitate Galton's analysis, we will create a dataset with the heights of fathers and a randomly selected son of each family:

```
library(tidyverse)
library(HistData)
library(data.table)
data("GaltonFamilies")
GaltonFamilies <- as.data.table(GaltonFamilies)
set.seed(1983)
galton_heights <- GaltonFamilies[gender == 'male'][, .SD[sample(.N, 1L)], by = family][,(father, childHeight)
setnames(galton_heights, "childHeight", "son")
```

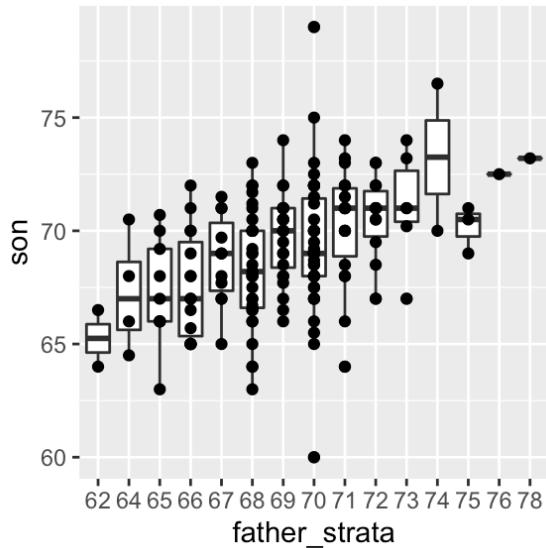
Plotting clearly shows that the taller the father, the taller the son.

```
galton_heights %>% ggplot(aes(father, son)) +
  geom_point(alpha = 0.5)
```



One can visualize more concretely the conditional distributions of the son heights given the father heights by stratifying the father heights (i.e. $p(\text{son}|\text{father})$):

```
galton_heights[, father_strata := factor(round(father))] %>%
  ggplot(aes(father_strata, son)) +
  geom_boxplot() +
  geom_point()
```



The centers of the groups are increasing with the father height, giving a first glimpse as how hereditary body height is.

By strata, we can estimate the conditional expectations. In our example, we end up with the following prediction for the son of a father who is 72 inches tall:

```
conditional_avg <- galton_heights[round(father) == 72,
                                mean(son)]
conditional_avg

## [1] 70.5
```

Furthermore, these centers appear to follow a linear relationship. Below we plot the averages of each group. If we take into account that these averages are random variables, the data is consistent with these points following a straight line:

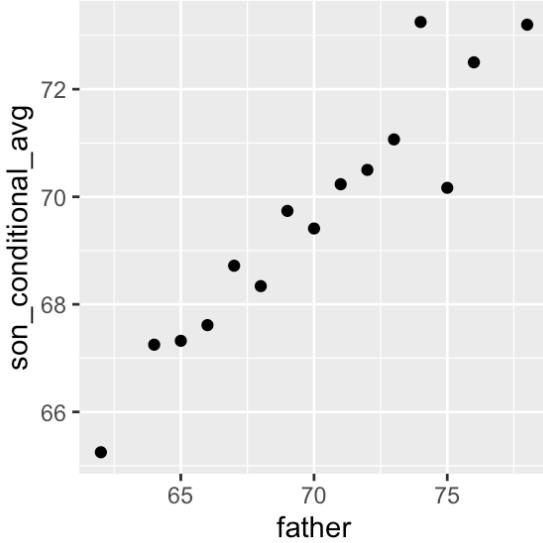


Figure 10.1: Average son heights by strata of the father heights

Hence this plot suggests that the linear model:

$$E[\text{son}|\text{father}] = \beta_0 + \beta_1 \text{father}$$

is reasonable. Fitting such a model would allow us to avoid doing stratifications, as we could continuously estimate the expected value of son's height. How do we estimate these coefficients? To this end, we need a bit more theory.

10.2.2 Maximum likelihood and least squares estimates

Fitting a linear regression, i.e. estimating the parameters, is based on a widely used principle called **maximum likelihood**. The maximum likelihood principle consists in choosing as parameter values those for which the data is most probable. What is the probability of our data?

Here we model the probability of the heights of the sons given the heights of their fathers. Generally, we will always consider the values of the explanatory variables to be given. Furthermore we will assume that, conditioned on the values of the explanatory variable, the observations are independent. Here, this means that the height of the sons are independent given the heights of the fathers. Hence the likelihood is:

$$p(\text{Data}|\beta_0, \beta_1) = \prod_i p(y_i|x_i, \beta_0, \beta_1)$$

Hence, we look for the values of the coefficients β_0 and β_1 to maximize $\prod_i p(y_i|x_i, \beta_0, \beta_1)$.

Taking the logarithm of the likelihood, which is a monotonically increasing function, does not change the value of the optimal parameters. Plugging in furthermore the density of the Gaussian³¹ and discarding terms not affected by the values of β_0 β_1 , we obtain that:

$$\begin{aligned}\arg \max_{\beta_0, \beta_1} \prod_i p(y_i | x_i, \beta_0, \beta_1) &= \arg \max_{\beta_0, \beta_1} \sum_i \log(N(y_i | x_i, \beta_0, \beta_1)) \\ &= \arg \min_{\beta_0, \beta_1} \sum_i (y_i - (\beta_0 + \beta_1 x_i))^2\end{aligned}$$

The differences between the observed values and their expected values denoted $\epsilon_i = y_i - (\beta_0 + \beta_1 x_i)$ are called the errors. Hence, maximizing the likelihood of this model is equivalent to minimizing the sum of the squared errors. One talks about **least squares estimates** (LSE).

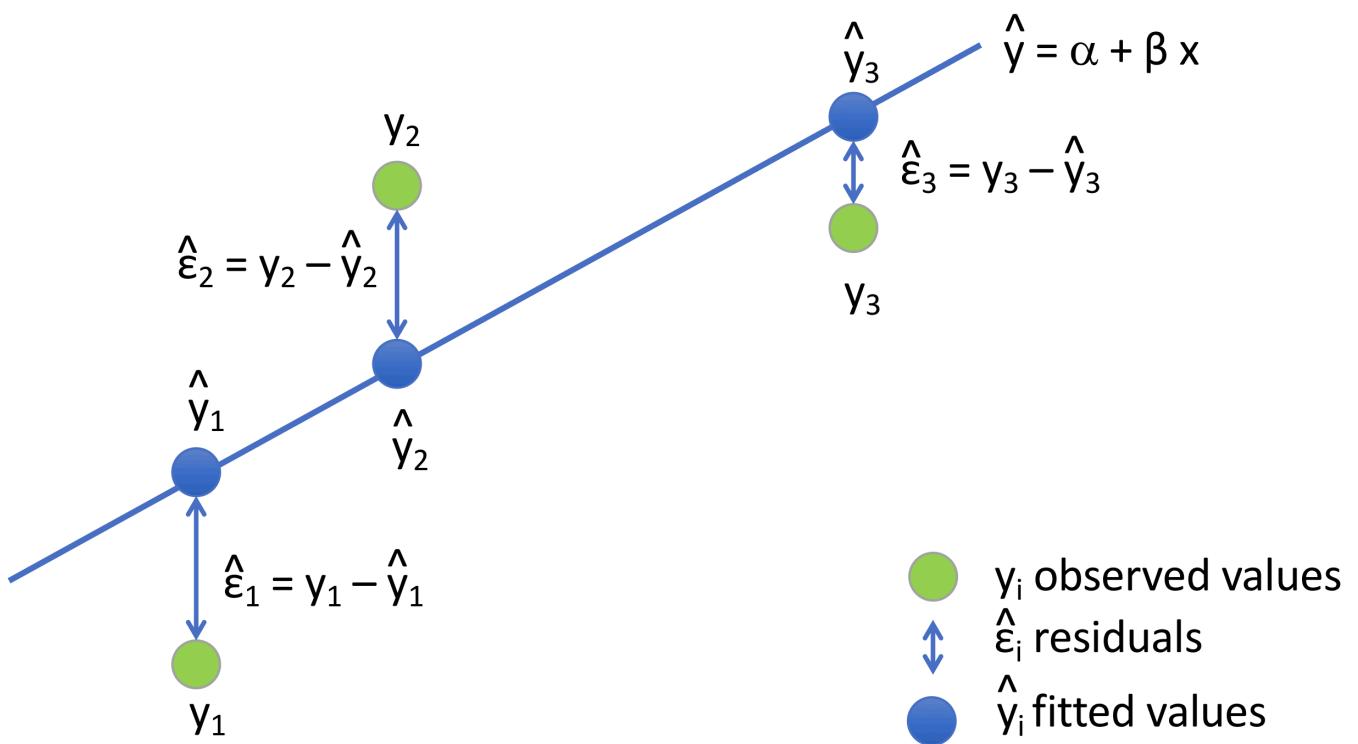


Figure 10.2: Visualization of a Linear Regression model.

10.2.3 Interpretation of the fitted coefficients

Minimizing the squared errors for a linear model can be solved analytically (see Appendix D.4). We obtain that our estimated conditional expected value, denoted \hat{y} is:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x \text{ with slope } \hat{\beta}_1 = \rho \frac{\sigma_y}{\sigma_x} \text{ and intercept } \hat{\beta}_0 = \mu_y - \hat{\beta}_1 \mu_x \quad (10.4)$$

where:

- μ_x, μ_y are the means of x and y
- σ_x, σ_y are the standard deviations of x and y
- $\rho = \frac{1}{n} \sum_{i=1}^n \left(\frac{x_i - \mu_x}{\sigma_x} \right) \left(\frac{y_i - \mu_y}{\sigma_y} \right)$ is the Pearson correlation coefficient between x and y .

We can rewrite this result as:

$$\hat{y} = \mu_y + \rho \left(\frac{x - \mu_x}{\sigma_x} \right) \sigma_y$$

If there is **perfect correlation**, the regression line predicts an increase in the response by the same number of standard deviations. If there is 0 correlation, then we don't use x at all for the prediction and simply predict the population average μ_y . For values between 0 and 1, the prediction is somewhere in between. If the correlation is negative, we predict a reduction instead of an increase. Note that if the correlation is positive but smaller than 1, our prediction of y is closer to its average than x to its average (in standard units).

In R, we can obtain the least squares estimates using the `lm` function:

```
fit <- lm(son ~ father, data = galton_heights)
coefficients(fit)
```

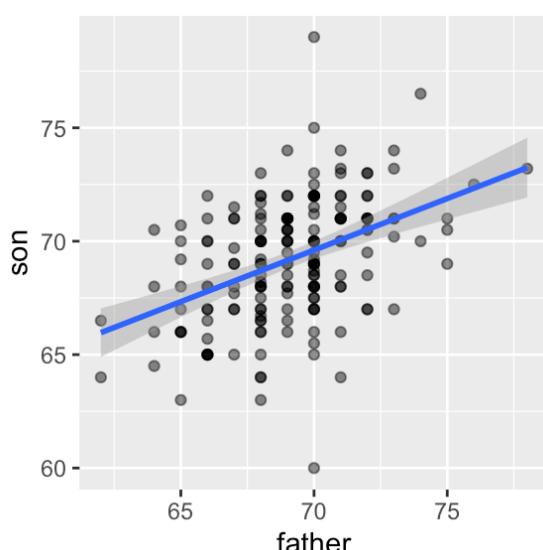
```
## (Intercept)      father
##   37.775451    0.454742
```

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

Here we add the regression line to the original data using the `ggplot2` function `geom_smooth(method = "lm")` which computes and adds the linear regression line to a plot along with confidence intervals:

```
galton_heights %>%
  ggplot(aes(father, son)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm")

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



In our example, the correlation between sons' and fathers' heights is about 0.5. Our predicted value for the son's height $\hat{y} = 70.5$ for a 72-inch father was only 0.48 standard deviations larger than the average son in contrast to the father's height which was 1.1 standard deviations above average. This is why we call it *regression*: the son regresses to the average height. In fact, the title of Galton's paper was: *Regression toward mediocrity in hereditary stature*.

It is a fact that children of extremely tall parents are taller than average, yet typically shorter than their parents. You can appreciate it on Figure 10.1. Extremely tall parents have an extreme combinations of alleles, and have also benefited from environmental factors by chance, which may not repeat in the next generation. The same phenomenon, called *regression toward the mean*, happens in sport. Athletes who perform extremely well one year are certainly good (high expectation) but have also probably been lucky (a positive error) so they are more likely to perform relatively worse the next year, etc. This phenomenon is ubiquitous and can lead to fallacies.³²

10.2.4 Predicted values are random variables

Once we fit our model, we can obtain predictions of y by plugging the estimates into the regression model. For example, if the father's height is x , then our prediction \hat{y} for the son's height is:

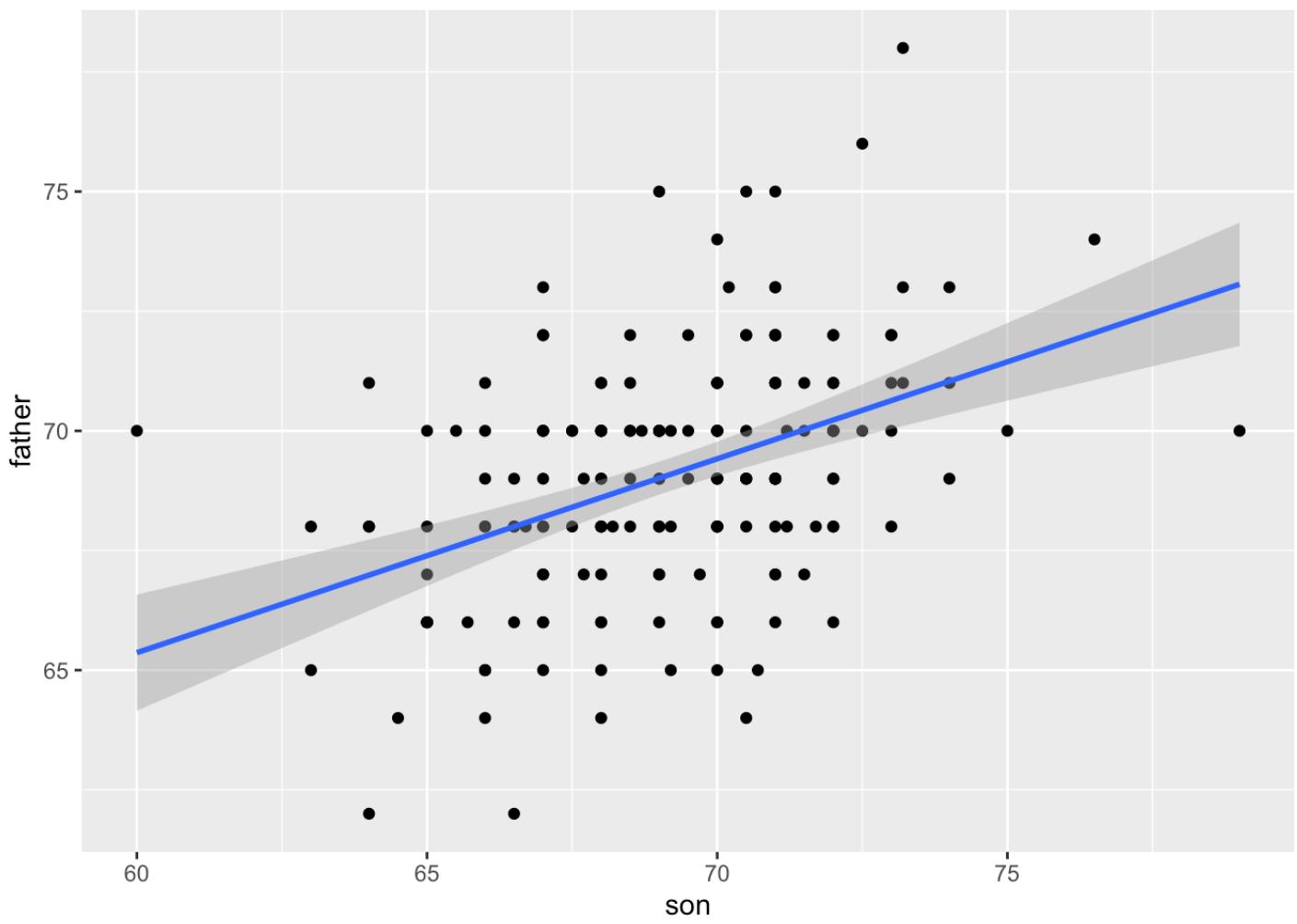
$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

When we plot \hat{y} versus x , we see the regression line.

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

```
galton_heights %>% ggplot(aes(son, father)) +
  geom_point() +
  geom_smooth(method = "lm")

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



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

```
fit <- galton_heights %>% lm(son ~ father, data = .)
y_hat <- predict(fit, se.fit = TRUE)
names(y_hat)

## [1] "fit"           "se.fit"        "df"
## [4] "residual.scale"
```

10.2.5 Explained variance

Any dataset will give us estimates of the model, but how well does the model represent our data? To investigate this we can compute quality metrics and visually assess the model, which is explained later.

Under the linear regression assumptions, the conditional standard deviation is:

$$\text{SD}(y | x) = \sigma_y \sqrt{1 - \rho^2}$$

To see why this is intuitive, notice that without conditioning, $\text{SD}(y) = \sigma_y$, we are looking at the variability of all the sons. But once we condition, we are only looking at the variability of the sons with the same father's height (for instance all those with a 72-inch, father). This group will tend to have similar heights so the standard deviation is reduced.

This is usually quantified in terms of proportion of variance. So we say that X explains $1 - (1 - \rho^2) = \rho^2$ (the Pearson correlation squared) of the variance. The proportion of variance explained by the model is commonly called the coefficient of determination or R^2 . Another way of deriving R^2 is described in the following steps:

First, we compute the model predictions:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i.$$

Then, we compute the residuals (by comparing the predictions with the actual values)

$$\hat{\epsilon}_i = \hat{y}_i - y_i,$$

and the residual sum of squares

$$RSS = \sum_{i=1}^N \hat{\epsilon}_i^2.$$

Lastly, we can compare the residual sum of squares to the total sum of squares (SS) of y

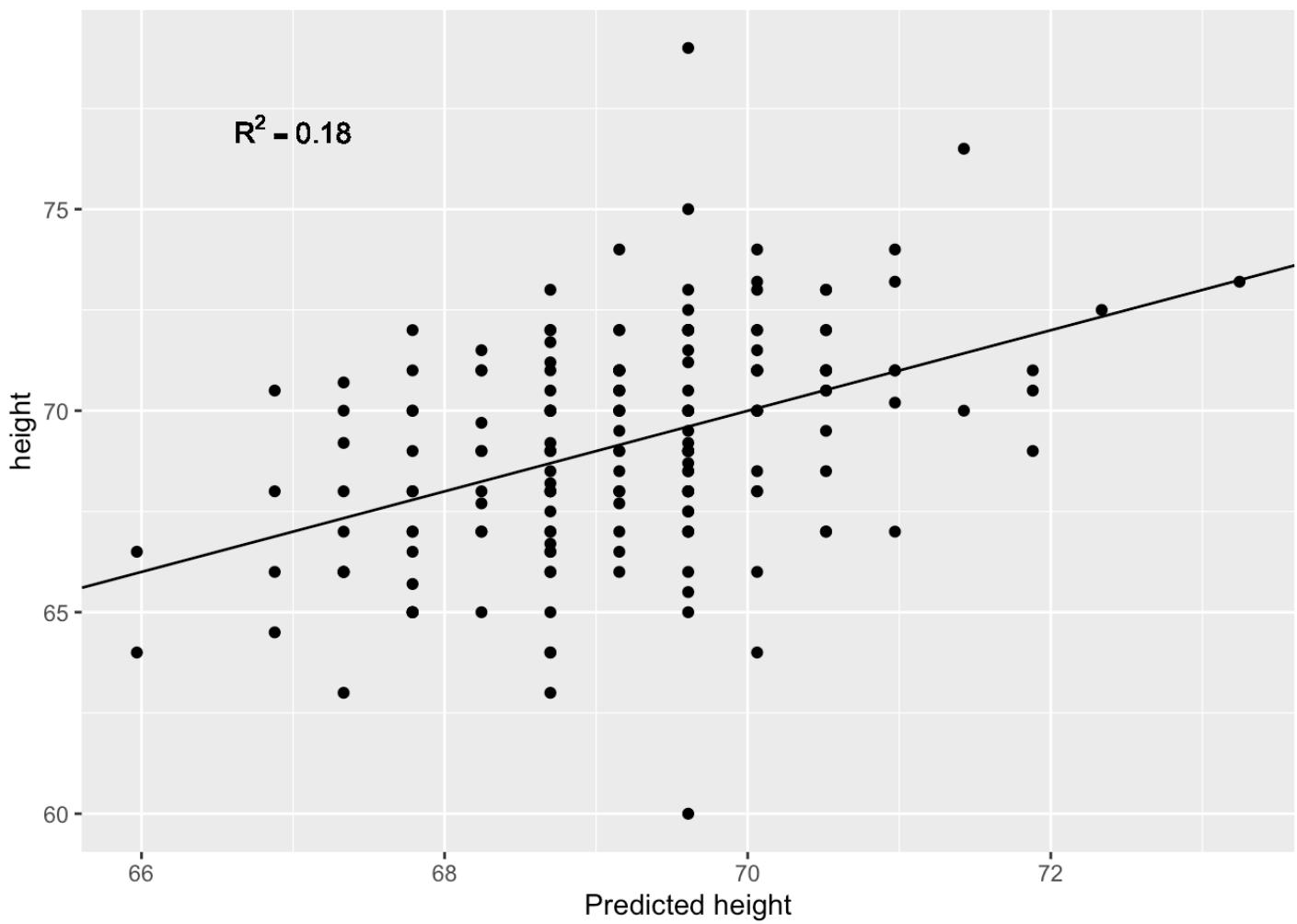
$$R^2 = 1 - \frac{\sum_{i=1}^N \hat{\epsilon}_i^2}{\sum_{i=1}^N (y_i - \bar{y})^2} = 1 - \frac{RSS}{SS}.$$

It can take any value between 0 and 1, since the sum of squares represents the variation around the global mean and the residual sum of squares represents the variation around the model predictions. In case the model learned no variation, it learned at least the global mean. In this case the sum of squares and the residual sum of squares are equal and $R^2 = 0$. In case the model is perfect, the residuals are zero and hence the second term vanishes and the R^2 becomes 1.

The R^2 is usually best combined with a scatter plot of y against \hat{y} .

```
m <- lm(son ~ father, data = galton_heights)
r2 <- summary(m)$r.squared

ggplot(galton_heights, aes(x=predict(m), y=son)) + geom_point() +
  geom_abline(intercept=0, slope=1) +
  geom_text(aes(x=67, y=77,
                label=deparse(bquote(R^2 == .(signif(r2, 2)))))
            ), parse=TRUE) +
  labs(x="Predicted height", y="height")
```



10.2.6 Testing the relationship between y and x

The LSE is derived from the data y_1, \dots, y_n , which are a realization of random variables. This implies that our estimates are random variables. Using the assumption of independent Gaussian noise, we obtain the following distribution:

$$p(\hat{\beta}) = N(\beta, \sigma^2 / ns_x^2)$$

Hence, if the modeling assumptions hold, the estimates are unbiased, meaning that their expected value are the true values of the parameter $E[\hat{\beta}] = \beta$. Moreover they are consistent, meaning that for infinitely large sample size they converge to the true values.

Another result allows us to build a hypothesis test, namely:

$$p\left(\frac{\hat{\beta} - \beta}{\hat{se}(\hat{\beta})}\right) = t_{n-2}\left(\frac{\hat{\beta} - \beta}{\hat{se}(\hat{\beta})}\right)$$

where t_{n-2} denotes the student's t distribution with $n - 2$ degrees of freedom.

Remember, the p-value of a statistical test is the **probability** of the value of a **test statistic** being at least as extreme as the one observed in our data **under the null hypothesis**.

In our case we can formulate the null hypothesis, that y does not depend on x_i as follows:

- The null hypothesis for parameter β_i is $H_0 : \beta_i = 0$
- The test statistic is $\hat{t} = \frac{\hat{\beta} - \beta_i}{\text{se}(\hat{\beta})} = \frac{\hat{\beta}}{\text{se}(\hat{\beta})}$
- The probability under the null model is $P(t \geq \hat{t})$, where $t \sim t_{n-2}$

To confirm a linear relationship between y and x , we need to reject the null hypothesis at significance level $\alpha (= 0.05)$:

- Accept H_0 if $P(|t| \geq |\hat{t}|) > \alpha$
- Reject H_0 if $P(|t| \geq |\hat{t}|) \leq \alpha$

Now the question remains, how do we do this in R? Luckily the `summary(fit)` function helps us here. It reports t-statistics (`t value`) and p-values (`Pr(>|t|)`) for each estimate.

```
lm(son ~ father, data = galton_heights) %>%
  summary %>% .$coef

##             Estimate Std. Error   t value    Pr(>|t|)
## (Intercept) 37.775451 4.97272309 7.596532 1.693821e-12
## father       0.454742 0.07192596 6.322363 2.034076e-09
```

10.3 Multivariate regression

Since Galton's original development, regression has become one of the most widely used tools in data science. One reason for this has to do with the fact that regression permits us to find relationships between two variables taking into account the effects of other variables. This has been particularly popular in fields where randomized experiments are hard to run, such as economics and epidemiology.

When we are not able to randomly assign each individual to a treatment or control group, **confounding** is particularly prevalent. For example, consider estimating the effect of eating fast foods on life expectancy using data collected from a random sample of people in a jurisdiction. Fast food consumers are more likely to be smokers, drinkers, and have lower incomes. Therefore, a naive regression model may lead to an overestimate of the negative health effect of fast food. So how do we account for confounding in practice? In this section we learn how **linear regression against multiple variables**, or multivariate regression, can help with such situations and can be used to describe how one or more variables affect an outcome variable.

10.3.1 A multivariate example: The baseball dataset

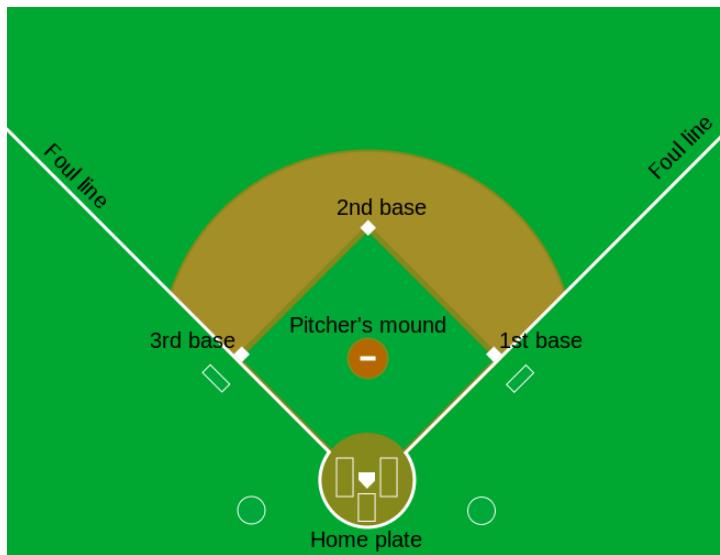
We will use data from baseball, leveraging a famous real application of regression which has led to improved estimations of baseball player values in the 90's³³. Here we will not build a model of player value but, instead, we will focus on predicting the game scores of a team.

10.3.1.1 Baseball basics

The goal of a baseball game is to score more runs (points) than the other team. Each team has 9 batters that have an opportunity to hit a ball with a bat in a predetermined order. After the 9th batter, the first batter bats again, then the second, and so on.

Each time a batter has an opportunity to bat, the other team's *pitcher* throws the ball and the batter tries to hit it. The batter either makes an *out* and returns to the bench or the batter hits it.

When the batter hits the ball, the batter wants to pass as many *bases* as possible before the opponent team catches the ball. There are four bases with the fourth one called *home plate*. Home plate is where batters start by trying to hit, so the bases form a cycle.



(Courtesy of Cburnett³⁴. CC BY-SA 3.0 license³⁵.)

A batter who stops at one of the intermediate three bases can resume running along the base cycle when next batters hit the ball. A batter who goes around the bases and arrives home (directly or indirectly with stops at intermediate bases), scores a **run**.

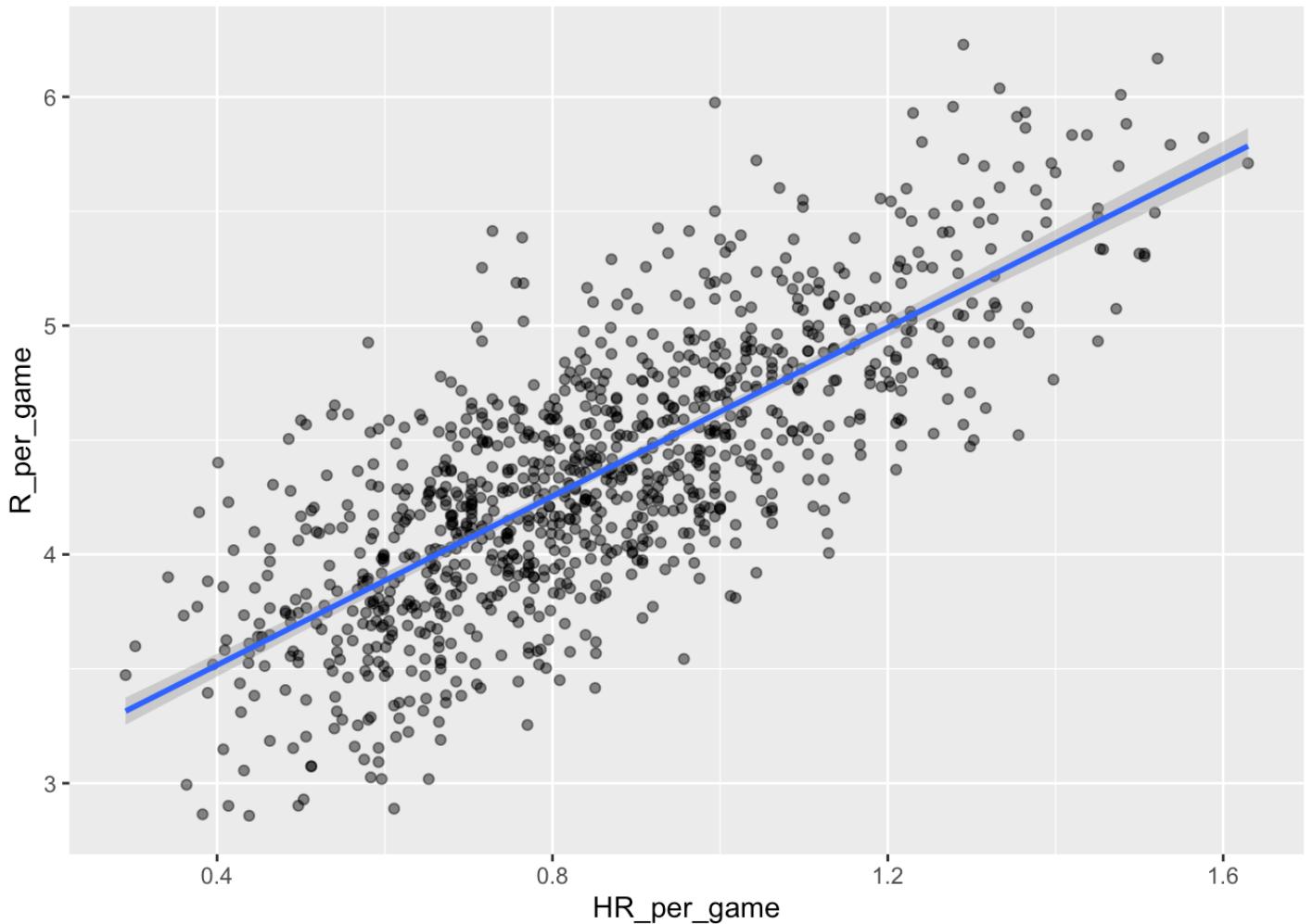
We want to understand what makes a team score well on average. Hence, the average number of runs is our response variable.

10.3.1.2 Home runs

A **home run** happens when the batter who hits the ball goes all the way home. This happens when the ball is hit very far, giving time for the batter to perform the full run. It is very good for the score because not only the batter scores a run, but the other players of the same team who are already on the pitch standing at intermediate bases typically finish their runs too.

Not surprisingly, average home runs positively correlate with average runs:

```
library(Lahman)
Teams <- as.data.table(Teams)
Teams_filt <- Teams[yearID %in% 1961:2001]
Teams_filt[,c('HR_per_game', 'R_per_game') := list(HR/G, R/G)]%>%
  ggplot(aes(HR_per_game, R_per_game)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method="lm")
```



A univariate linear regression gives the following parameter estimates:

```
fit_r_vs_hr <- lm(R_per_game~HR_per_game, data=Teams_filt)
fit_r_vs_hr
```

```

## 
## Call:
## lm(formula = R_per_game ~ HR_per_game, data = Teams_filt)
## +1,845 + 1
## Coefficients:
## (Intercept)  HR_per_game
##           2.778      1.845

```

So this tells us that teams that hit 1 more HR per game than the average team, score 1.8448241 more runs per game than the average team. Given that the most common final score is a difference of a run, this can certainly lead to a large increase in wins. Not surprisingly, HR hitters are very expensive. Because we are working on a budget, we will need to find some other way to increase wins. So, in the next section, we move our attention to another possible predictive variable of runs.

10.3.1.3 Base on balls

Obviously the pitcher is encouraged to throw the ball at the batter. This is achieved thanks to the so-called **base on balls** rule stating that if the pitcher fails to throw the ball through a predefined area considered to be hittable (the strikezone), the batter is permitted to go to the first base.

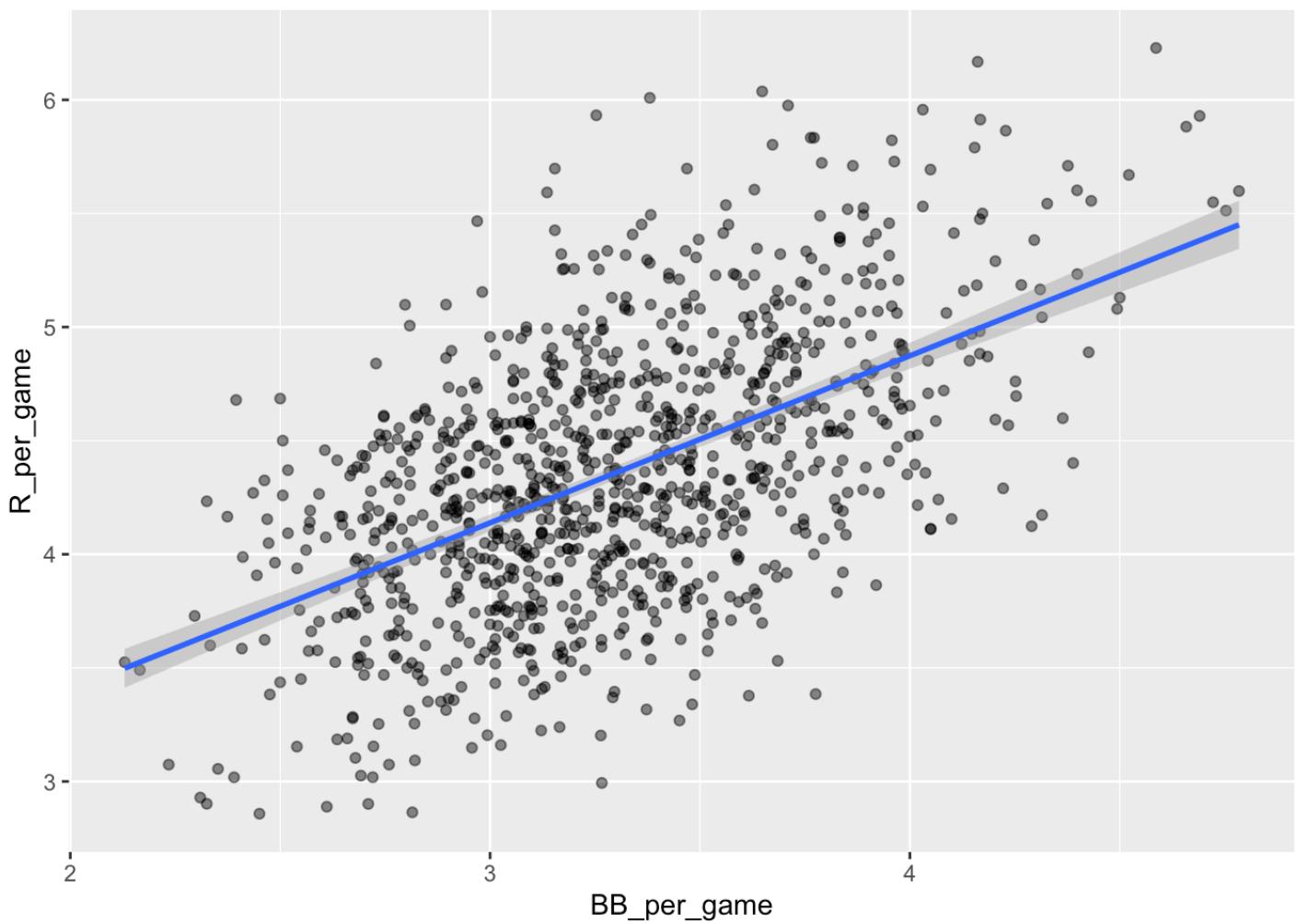
A base on ball is not as great as a home run but it makes the batter progress by one base so it should be rather beneficial for the score. Let us now scatter plot average runs against average base on balls:

```

Teams_filt[,BB_per_game := BB/G] %>%
  ggplot(aes(BB_per_game, R_per_game)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method="lm")

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

```



Here, again we see a clear association. If we find the regression line for predicting runs from bases on balls, we get a slope of:

```
get_slope <- function(x, y) cor(x, y) * sd(y) / sd(x)
bb_slope <- Teams_filt[,(slope = get_slope(BB_per_game, R_per_game) )]
bb_slope

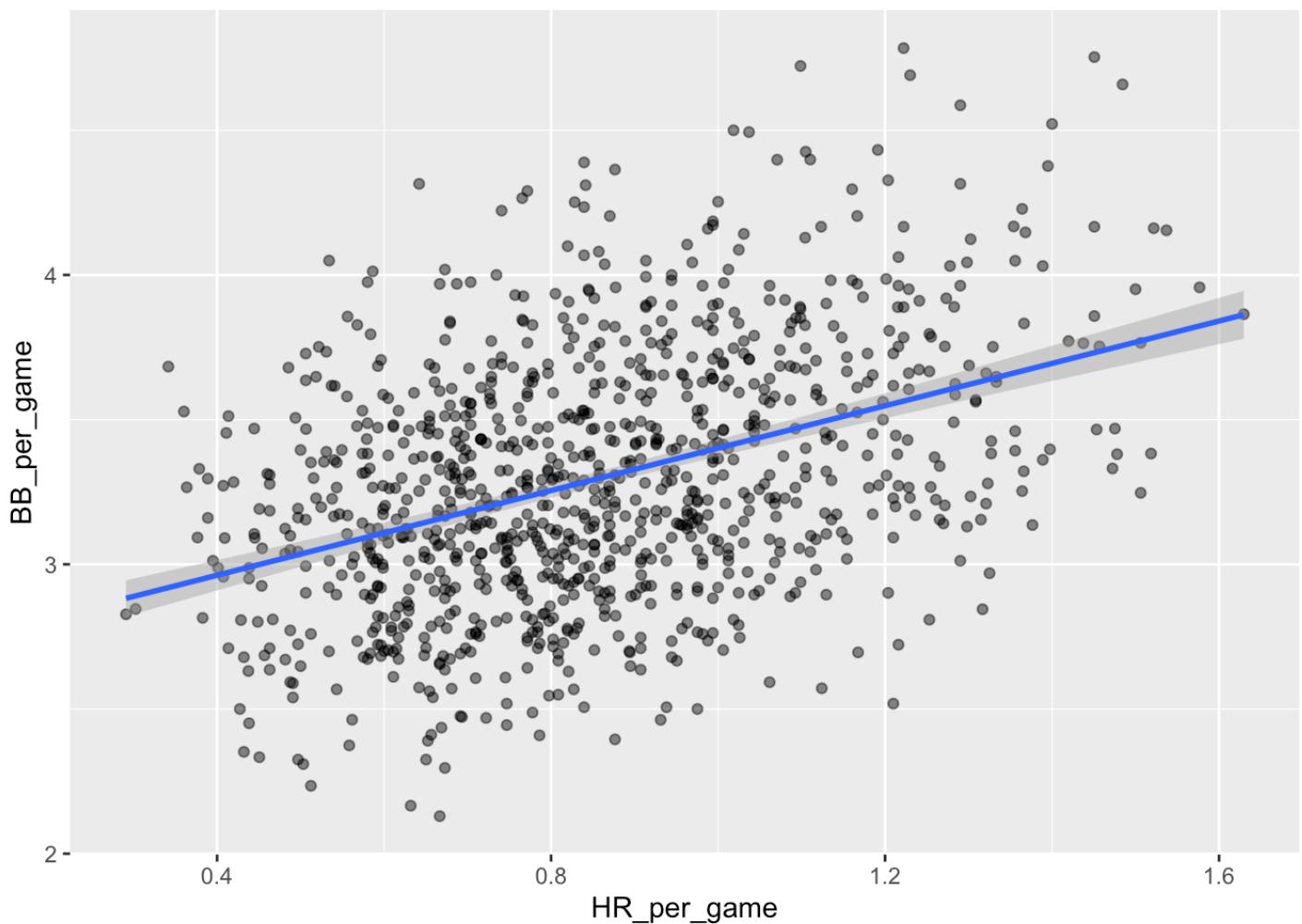
##           slope
## 1: 0.7353288
```

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

In fact, it looks like BBs and HRs are also associated:

```
Teams_filt %>%
  ggplot(aes(HR_per_game, BB_per_game)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method="lm")
```

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



We know that HRs cause runs because, as the name “home run” implies, when a player hits a HR they are guaranteed at least one run. Could it be that HRs also cause BB and this makes it appear as if BB cause runs?

It turns out that pitchers, afraid of HRs, will sometimes avoid throwing strikes to HR hitters. As a result, HR hitters tend to have more BBs and a team with many HRs will also have more BBs. Although it may appear that BBs cause runs, it is actually the HRs that cause most of these runs. We say that BBs are *confounded* with HRs. Nonetheless, could it be that BBs still help? To find out, we somehow have to adjust for the HR effect. Linear regression will help us parse all this out and quantify the associations. This can in turn help determine what players to recruit.

10.3.1.4 Understanding confounding through stratification

We first untangle the direct from the indirect effects **step-by-step by stratification** to get a concrete understanding of the situation.

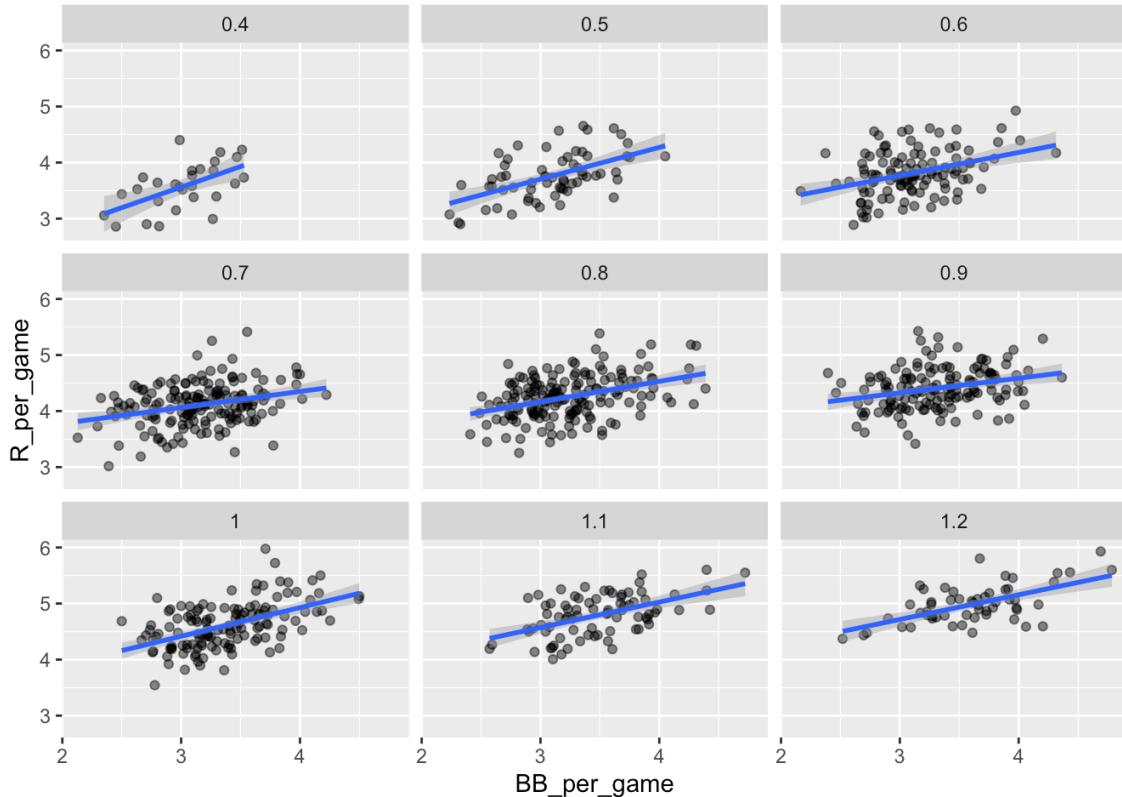
A first approach is to keep HRs fixed at a certain value and then examine the relationship between BB and runs. As we did when we stratified fathers by rounding to the closest inch, here we can stratify HR per game to the closest tenth. We filter out the strata with few points to avoid highly variable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

```
dat <- Teams_filt[,HR_strata := round(HR/G, 1)][HR_strata >= 0.4 & HR_strata <=1.2]
```

and then make a scatterplot for each strata:

```
dat %>%
  ggplot(aes(BB_per_game, R_per_game)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  facet_wrap(~ HR_strata)
```

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



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

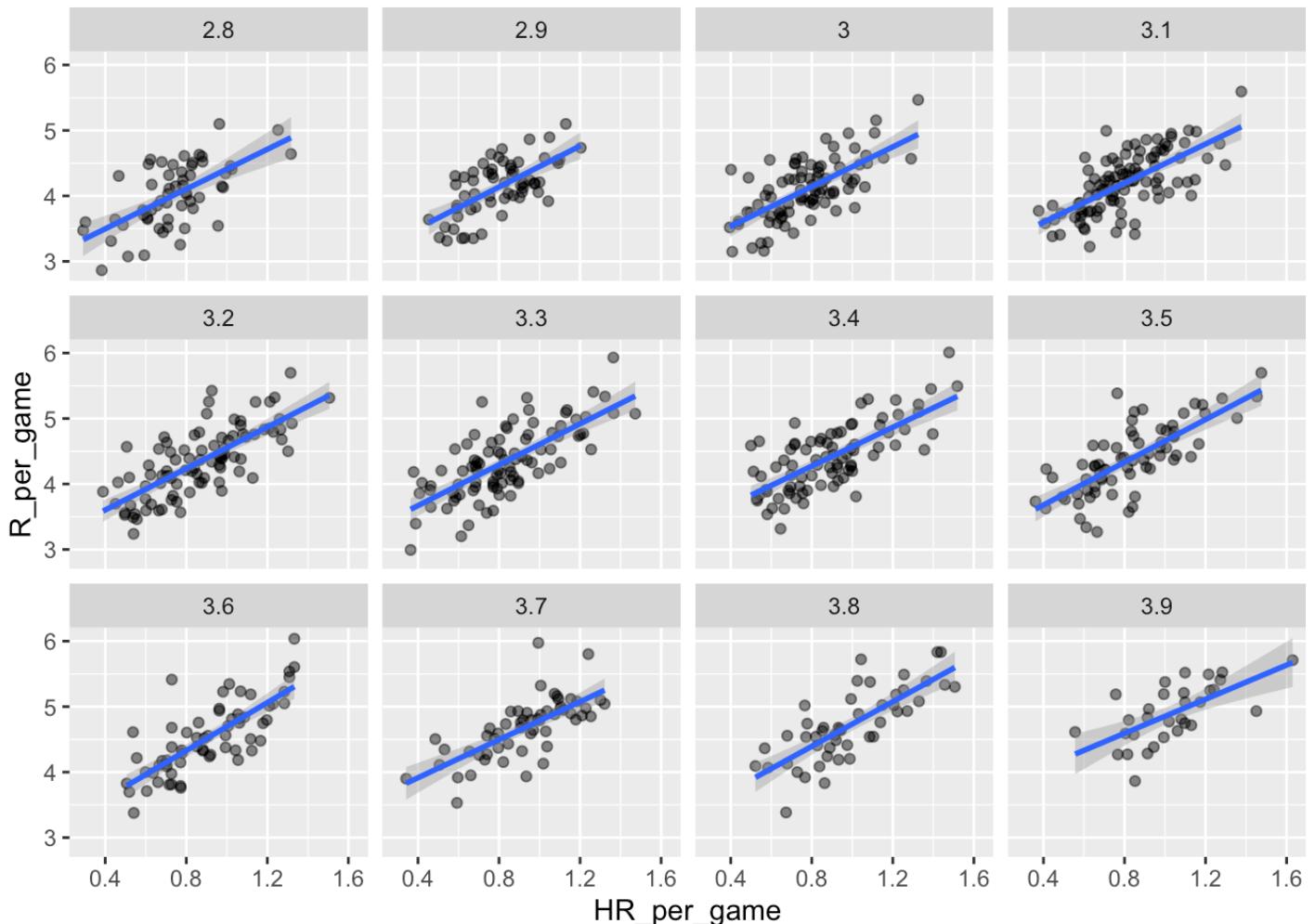
```
dat[order(HR_strata), .(slope = get_slope(BB_per_game, R_per_game)), by='HR_strata']
```

```
##      HR_strata      slope
## 1:      0.4 0.7342910
## 2:      0.5 0.5659067
## 3:      0.6 0.4119129
## 4:      0.7 0.2853933
## 5:      0.8 0.3650361
## 6:      0.9 0.2608882
## 7:      1.0 0.5115687
## 8:      1.1 0.4539252
## 9:      1.2 0.4403274
```

The slopes are reduced, but they are not 0, which indicates that BBs are helpful for producing runs, just not as much as previously thought³⁶

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

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



In this case, the slopes are reduced a bit, which is consistent with the fact that BB do in fact cause some runs:

```
dat[order(BB_strata), .(slope = get_slope(HR_per_game, R_per_game)), by = 'BB_strata']

##      BB_strata     slope
## 1:      2.8 1.518056
## 2:      2.9 1.567879
## 3:      3.0 1.518179
## 4:      3.1 1.494498
## 5:      3.2 1.582159
## 6:      3.3 1.560302
## 7:      3.4 1.481832
## 8:      3.5 1.631314
## 9:      3.6 1.829929
## 10:     3.7 1.451895
## 11:     3.8 1.704564
## 12:     3.9 1.302576
```

Compared to the original:

```
hr_slope <- Teams_filt[.(slope = get_slope(HR_per_game, R_per_game))]

hr_slope
```

```
##      slope
## 1: 1.844824
```

10.3.1.5 Data suggests additive effects

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

$$E[R | BB = x_1, HR = x_2] = \beta_0 + \beta_1(x_2)x_1 + \beta_2(x_1)x_2$$

with the slopes for x_1 changing for different values of x_2 and vice versa. But is there an easier approach?

If we take random variability into account, the slopes in the strata don't appear to change much. If these slopes are in fact the same, this implies that $\beta_1(x_2)$ and $\beta_2(x_1)$ are constants. This in turn implies that the expectation of runs conditioned on HR and BB can be written like this:

$$E[R | BB = x_1, HR = x_2] = \beta_0 + \beta_1x_1 + \beta_2x_2$$

This model suggests that if the number of HR is fixed at x_2 , we observe a linear relationship between runs and BB with an intercept of $\beta_0 + \beta_2x_2$. Our exploratory data analysis suggested this. The model also suggests that as the number of HR grows, the intercept growth is linear as well and determined by β_1x_1 .

In this analysis, referred to as *multivariate regression*, you will often hear people say that the BB slope β_1 is *adjusted* for the HR effect. If the model is correct then confounding has been accounted for. But how do we estimate β_1 and β_2 from the data? For this, we need to derive some theoretical results that generalize the results from univariate linear regression.

10.3.2 Fitting multivariate regression

For a data set (\mathbf{x}_i, y_i) with $i \in \{1 \dots n\}$ and \mathbf{x}_i a vector of length p , the multiple linear regression model is defined as:

$$y_i = \beta_0 + \sum_{j=1}^p \beta_j x_{i,j} + \epsilon_i$$

with free parameters β_0 and β and a random error $\epsilon_i \sim N(0, \sigma^2)$ that is i.i.d. (independently and identically distributed).

The model can be written in matrix notation

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

here the matrix \mathbf{X} is of dimension $(n \times p + 1)$ where each row i corresponds to the vector \mathbf{x}_i with a 1 prepended to accommodate the intercept. The error is distributed as $\epsilon \sim N(\mathbf{0}, \Sigma)$ as a multivariate Gaussian with covariance $\Sigma = \sigma^2 \mathbf{I}$ (i.i.d.).

In the same way as for the simple linear model, **parameters can be estimated by finding the LSE**. For multiple linear regression, we obtain

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

$$\hat{\sigma}^2 = \frac{\hat{\epsilon}^\top \hat{\epsilon}}{n - p}.$$

As for the univariate case, we have that, under the assumptions of the model, the least squares estimates are unbiased. This means that, if the data truly originates from such a data generative model, the expected value of the estimates over repeated random realizations, equals the true underlying parameter values:

$$E[\hat{\beta}_j] = \beta_j$$

Moreover, the **estimates are consistent**: They converge **to the true values with increasing sample sizes**:

$$\hat{\beta}_j \xrightarrow{n \rightarrow \infty} \beta_j$$

Remarkably, this holds true even if the explanatory variables are correlated (unless perfectly correlated, in which case the parameters become not identifiable).

To fit a multiple linear regression model in R, we can use the same `lm` function as for the simple linear regression, we only need to adapt the formula to include all predictor variables. Here is the application to fitting the average runs against the home runs and the bases on balls:

```
fit <- lm(R_per_game ~ HR_per_game + BB_per_game,
            data=Teams_filt)
coef(fit)
```

```
## (Intercept) HR_per_game BB_per_game
## 1.7443011 1.5611689 0.3874238
```

We see that the coefficient for home runs and for bases on balls are similar to those we tediously estimated by stratifications.

The object `fit` includes more information about the fit, including statistical assessments. We can use the function `summary` to extract more of this information:

```
summary(fit)
```

```
##
## Call:
## lm(formula = R_per_game ~ HR_per_game + BB_per_game, data = Teams_filt)
##
## Residuals:
##    Min      1Q  Median      3Q     Max
## -0.87325 -0.24507 -0.01449  0.23866  1.24218
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.74430   0.08236  21.18   <2e-16 ***
## HR_per_game 1.56117   0.04896  31.89   <2e-16 ***
## BB_per_game 0.38742   0.02701  14.34   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3484 on 1023 degrees of freedom
## Multiple R-squared:  0.6503, Adjusted R-squared:  0.6496
## F-statistic: 951.2 on 2 and 1023 DF, p-value: < 2.2e-16
```

Other useful functions to extract information from `lm` objects are

- `predict` to compute the fitted values or predict response for new data
- `resid` to compute the residuals

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

10.3.3 Testing sets of parameters

10.3.3.1 Nested models

Hypothesis testing can be done on individual coefficients of a multivariate regression, with an appropriate t-test just as in the univariate case. But testing individual parameters may not suffice. Sometimes, we are interested in testing an entire set of variables. We then compare so-called **nested models**. We compare a full model Ω to a reduced model ω where model ω is a special case of the more general model Ω . For multivariate regression, this typically consists of setting a set of parameters to 0 in the reduced model.

Consider the following example model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$

For testing whether the coefficients of two explanatory variables x_1 and x_2 should be 0, one would consider:

- Full model: all β 's can take any value.
- Reduced model: $\beta_1 = \beta_2 = 0$ (only the mean β_0 and the third parameter β_3 can take any value).

10.3.3.2 F-test and ANOVA

The F-test can be applied to compare two nested linear regressions. It is based on comparing the fit improvements as measured by the change residual sum of squares. The idea is that the larger model, which has more parameters, always fits better to the data. With the F-test, we ask whether this improvement is significant under the null hypothesis that the smaller model is the correct one.

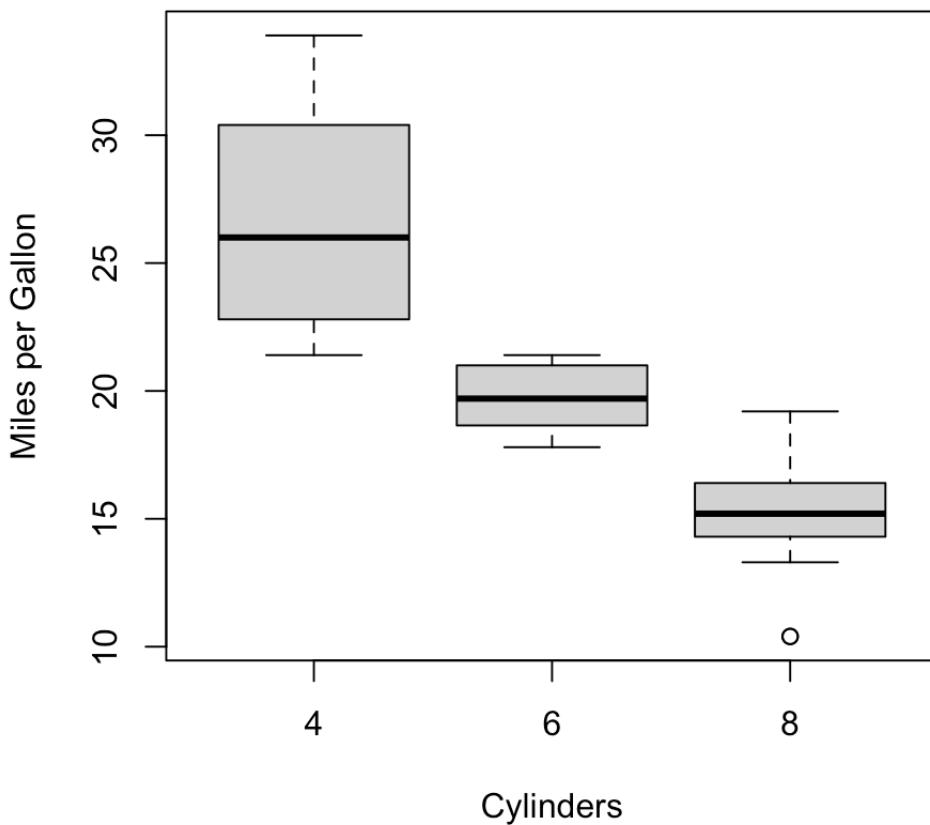
$$F = \frac{(RSS_{\omega} - RSS_{\Omega})/(q - p)}{RSS_{\Omega}/(n - q)},$$

where q is the number of parameters in (dimension of) model Ω and p the number of parameters in (dimension of) model ω and RSS designates residual sums of squares.

The statistic F is distributed according to the F distribution with $(q - p)$ and $(n - q)$ degrees of freedom respectively. We reject the LRT if F is larger than the critical value corresponding to the significance level. This analysis is also frequently referred to as "Analysis of Variance" or **ANOVA**.

Example: Testing the difference of means in 3 groups

As an example let us consider testing whether fuel consumption of cars depend on the number of cylinders using the `mtcars` dataset:



In this example, we are interested in whether there is a difference in the fuel consumption of cars depending on the number of cylinders in the car.

For this purpose, we will work with the following model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2.$$

Here, x_1 and x_2 will be indicator variables:

- group “6 cylinders”: $x_1 = 1$
- group “8 cylinders”: $x_2 = 1$

We want to test the effect of both indicators at the same time:

- **H0:** $\beta_1 = \beta_2 = 0$
- **Full model:** Ω is the space where all three β can take any value.
- **Reduced model:** ω is the space where only β_0 can take any value.

In R, we can easily test this:

```

data("mtcars")
## for the example we need a factor
## else it will be interpreted as number
mtcars$cyl <- as.factor(mtcars$cyl)
## fit the full model
full <- lm(mpg ~ cyl, data=mtcars)
## have a look at the model matrix
## which is automatically created
head(model.matrix(full))

```

	(Intercept)	cyl6	cyl8
## Mazda RX4	1	1	0
## Mazda RX4 Wag	1	1	0
## Datsun 710	1	0	0
## Hornet 4 Drive	1	1	0
## Hornet Sportabout	1	0	1
## Valiant	1	1	0

```

## fit the reduced model (only the intercept "1")
reduced <- lm(mpg ~ 1, data=mtcars)

```

```

## compare the models
anova(reduced, full)

```

```

## Analysis of Variance Table

##
## Model 1: mpg ~ 1
## Model 2: mpg ~ cyl
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     31 1126.05
## 2     29  301.26  2    824.78 39.697 4.979e-09 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

From the result, we can see that the full model models the data significantly better than the reduced model containing only the intercept. Therefore, we can conclude that there is a difference in the means of the 3 groups.

10.4 Diagnostic plots

The assumptions of a mathematical model never hold exactly in practice. The questions for the practitioners are threefold: 1) How badly are the assumptions violated on the dataset at hand? 2) What are the implications for the claimed conclusions? How can the issue be addressed?

The assumptions of linear regressions are:

- The expected values of the response are a linear combinations of the explanatory variables
- Errors are identically and independently distributed.
- Errors follow a normal distribution.

We will see that two diagnostic plots will be helpful: the residual plot and the q-q plot of the residuals.

10.4.1 Assessing non-linearity with residual plot

Diagnostic plot

Non-linearity is typically revealed by noticing that the average of the residual depends on the predicted values. A smooth fit (`geom_smooth default`) on the residual plot can help spotting systematic non-linear dependencies (See Figure 10.3).

```
## `geom_smooth()` using method = 'loess'
```

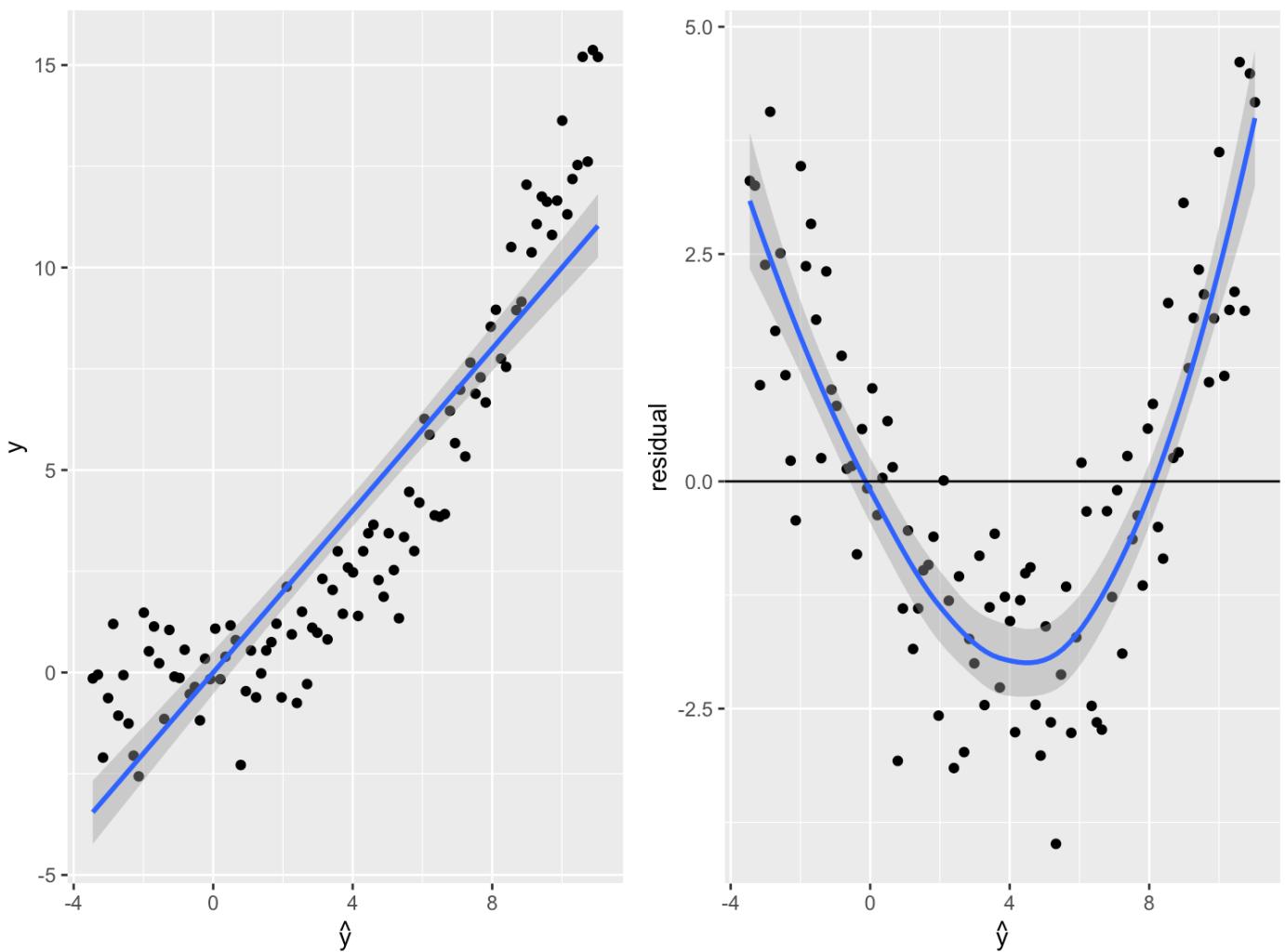


Figure 10.3: Detecting non-linearity. (Left) Observed value against predictions for linear fit on data generated as $y = x^3 + \text{noise}$. The fitted line is shown in blue. (Right) Residual versus predicted value of the same data. A smooth fit to the residual is shown in blue. The smooth fit highlights that the errors depend on the predicted expected values, indicative of non-linearity.

So what? The implications of non-linearity depends on the application purposes:

- Predictions are suboptimal. They could be improved with a more complex model. However, they may be good enough for the use case, and they would not necessarily deteriorate on unseen data.
- Explained variance is underestimated.
- The i.i.d assumption is violated: The residuals depend on the predicted mean, suggesting that the errors depend on $E[y|x]$ and therefore on each other. Therefore, statistical tests are flawed.
- Conditional dependencies can be affected. To see the latter assume a model in which two variables x and y depend on a common cause z in a non-linear fashion (eg. $y = z^2 + \epsilon$ and $x = z^2 + \epsilon'$). These two variables x and y are independent conditioned on their common cause z . However, a linear regression of y on x and z would fail to discard the contribution of x .

What to do? If non-linearity is revealed in the fit, one can either transform the explanatory variables or the response (eg log-transformtaion, powers, etc). Note that the appropriate transformation is difficult to know in practice and finding it likely requires trying out several options. The `formula` in R implements a few convenience functions to build such non-

linear transformation on the fly while calling `lm()`. For instance, the R call: `model <- lm(y ~ poly(x,3))` fits a polynomial of degree 3.

10.4.2 When error variance is not constant: Heteroscedascity

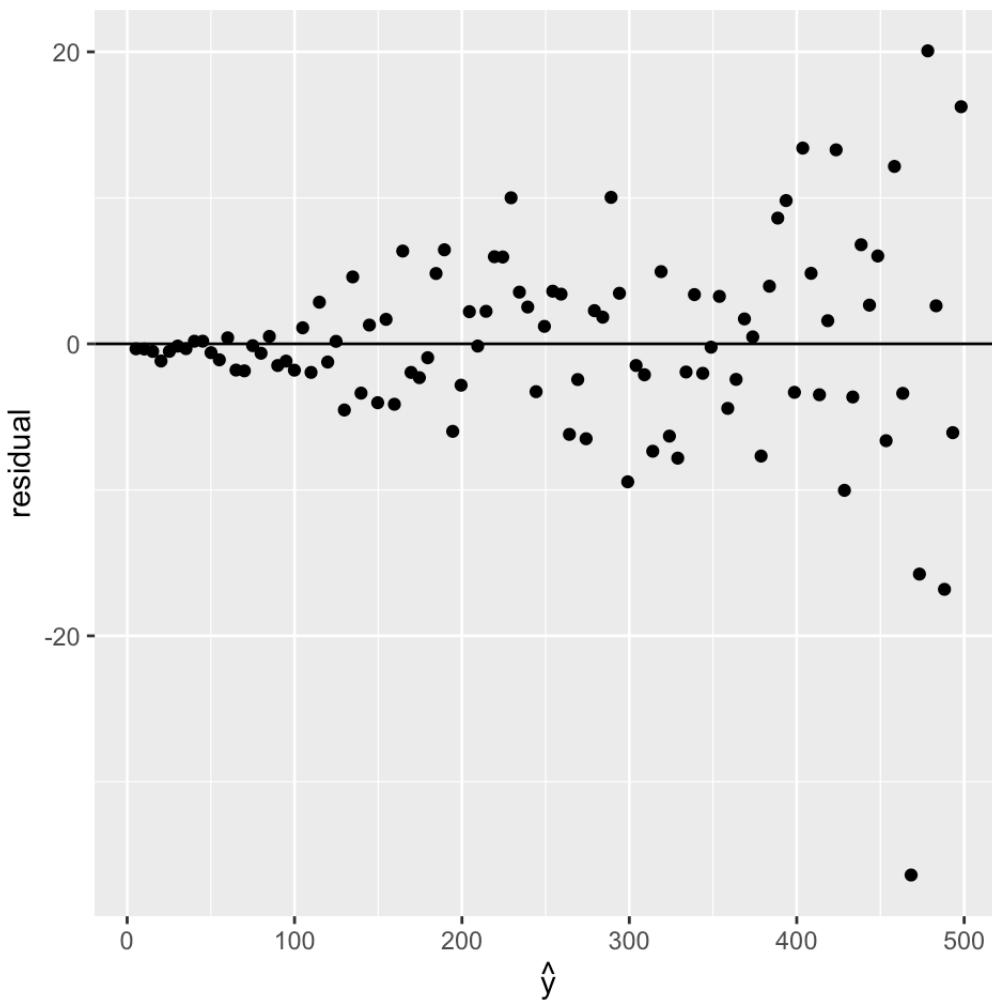
It happens not so rarely that the **variance of the residuals is not constant across all data points**. This property is called **heteroscedascity**. Heteroscedascity violates the i.i.d assumption of the errors.

Diagnostic plot

The **residual plots** can help spotting when error variance depends on response mean. Here is a synthetic example:

```
x <- 1:100
y <- rnorm(100, mean=5 * x, sd=0.1*x)
m <- lm(y ~ x)

ggplot(data=NULL, aes(predict(m), resid(m))) +
  geom_point() + geom_abline(intercept=0, slope=0) +
  labs(x=expression(hat(y)), y="residual")
```



So what? For prediction, the problem may or not be a real issue. Indeed the fit can be driven by a few data points because the least squares errors give too much importance to the points with high noise. This is particularly a problem with low number of points in areas with large noise.

As the residuals are not i.i.d., the statistical tests are flawed.

What to do?

In case of heteroscedascity, one can try to transform the response variable y , such as: log-transformation, square root, or variance stabilizing transformation.³⁷

However, as before, the appropriate transformation is difficult to know in practice and finding it likely requires trying out several options. Alternatively, one can use methods with a different noise model. One possibility is to consider weighted least squares, when there is the possibility to estimate before end the relative error variances on the data. This is sometimes the case, for instance, if one has access to experimental uncertainties. Another direction is to use generalized linear models.³⁸

10.4.3 Gaussianity: Q-Q-plot of the residuals

The Gaussian assumption of the errors is key to all statistical tests. An implication that the errors follow a Gaussian distribution is that the residuals also follow a Gaussian distribution.

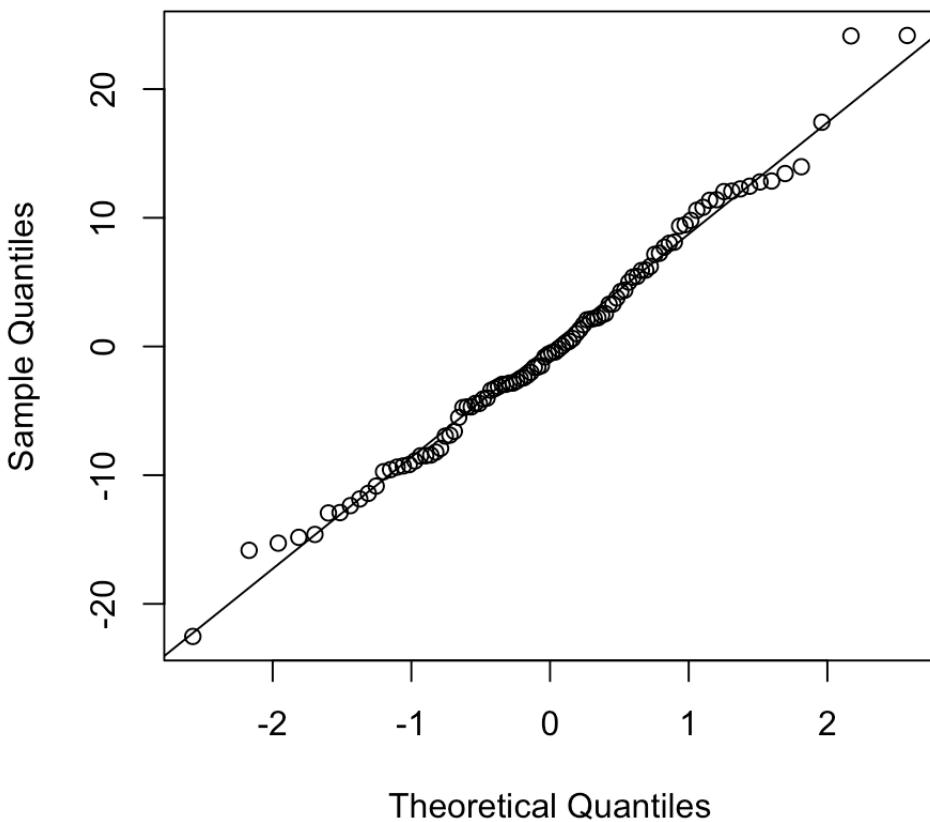
Diagnostic plot

We use here a QQ-plot of the residuals against the normal distribution.

The R default function `qqnorm()` and its companion `qqline()` generate qq-plot against a Gaussian fitted to the input vector. We do not have to worry about the mean and the standard deviation of the input vector. Here is a “good” example based on simulated data.

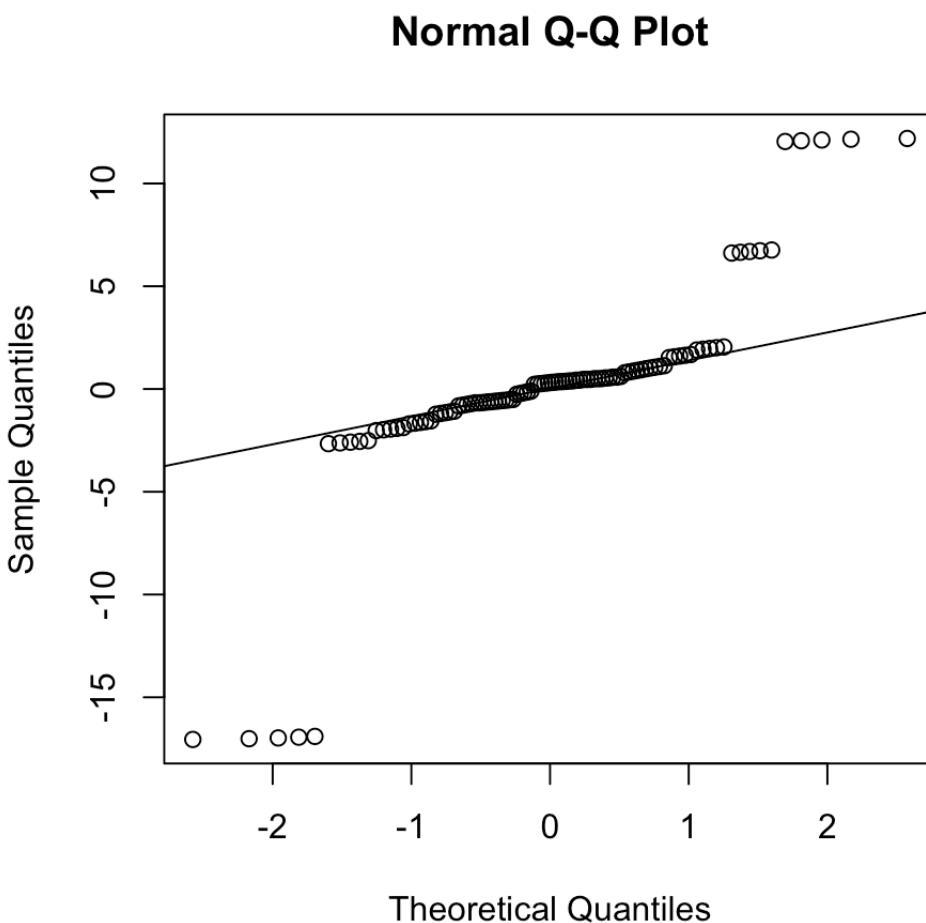
```
set.seed(0)
x <- 1:100
y <- rnorm(100, mean=5 * x, sd=10)
m <- lm(y ~ x)
qqnorm(residuals(m))
qqline(residuals(m))
```

Normal Q-Q Plot



And here is a “bad” example based on a simulated example, in which the errors are Student t-distributed with degree of freedom 1:

```
set.seed(0)
x <- 1:100
y <- x + rt(20, df=1)
m <- lm(y ~ x)
qqnorm(residuals(m))
qqline(residuals(m))
```



So what? If Gaussianity of the residuals is not satisfied, this means that the noise model is wrong. This can have the following practical implications:

- With enough data, the regression lines might not be too severely affected the least squares estimates converge to the expected values. Applying least squares to fit and predict does not depend on the Gaussian assumption!
- Hypothesis testing can be flawed.

What to do We may work with fundamentally non-Gaussian data (like Poisson distributed data, which are count data), or data with long tails and outliers. If one has an idea of what could be a better noise model, consider a generalized linear model with another distribution. Alternatively, use case resampling to estimate confidence intervals.

10.5 Conclusions

Linear models:

- are a powerful and versatile tool
- can be used to
 - predict future data
 - quantify explained variance
 - assess linear relations between variables (hypothesis testing)
 - control for confounding (multiple linear regression)

- assumptions need to be checked (diagnostic plots)
- can be generalized for different distributions (GLMs for classification: next lecture)

27. this is a guessed formula for the sake of the explanation. You will build your own model on true data in the exercise and see why the 0.5 coefficients is probably not correct. ↵
28. The popular deep neural networks are able, with enough data, to automatically learn these transformations. Nevertheless, their final operation (so-called last layer), is typically a linear combination as in Equation (10.3) ↵
29. https://en.wikipedia.org/wiki/Francis_Galton ↵
30. Galton made important contributions to statistics and genetics, but he was also one of the first proponents of eugenics, a scientifically flawed philosophical movement favored by many biologists of Galton's time but with horrific historical consequences. You can read more about it here: [<https://pged.org/history-eugenics-and-genetics>]. ↵
31. $N(x|0, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$. ↵
32. See the best seller Thinking fast and slow by Nobel prize-winning psychologist Daniel Kahneman (thank you, Lisa!) or https://en.wikipedia.org/wiki/Regression_toward_the_mean. ↵
33. A lengthier case study on this dataset is given in R. Irizzary's book. ↵
34. <https://en.wikipedia.org/wiki/User:Cburnett> ↵
35. <https://creativecommons.org/licenses/by-sa/3.0/deed.en> ↵
36. One can extract data of so-called singles, i.e the number of runs reaching the first base, and find that the effects on runs are similar than for BBs, which is consistent with the intuition that BB directly contributes to runs because it brings the batter to the first base. See R. Irizzary's book. ↵
37. https://en.wikipedia.org/wiki/Variance-stabilizing_transformation ↵
38. https://en.wikipedia.org/wiki/Generalized_linear_model ↵

Chapter 11 Logistic Regression

In the previous Chapter, we described linear regression to predict a quantitative response y using explanatory variables x_1, \dots, x_p .

However, in many applications the **response y is a category**. Examples of prediction tasks where the response is categorical include:

- **diagnostic** (to have a disease or not)
- **spam** email, not spam email
- handwritten digit recognition (0, 1, ..., 9)
- speech recognition (words)

Those prediction tasks for which the **response is a category** are called *classification tasks*. A special type of *classification*, with exactly **two categories**, is called **binary classification**. This chapter focuses on *binary classification*, where we encode $y \in \{0, 1\}$ the two categories.

11.1 A univariate example: predicting sex given the height

The students of this course have provided us with data on their heights, sex and height of their parents. A first classification task could be to predict the student's sex given the student's height.

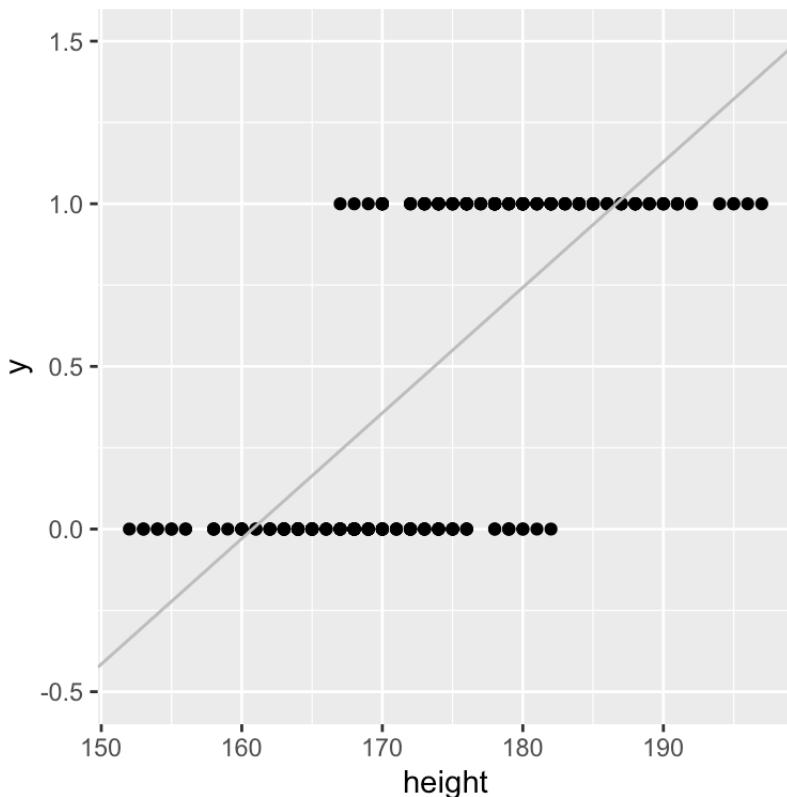
```
library(dslabs)
library(ggplot2)
library(magrittr)
library(data.table)
library(dplyr)

heights <- fread("extdata/height.csv") %>% na.omit() %>%
  .[, sex:=as.factor(toupper(sex))]
head(heights)
```

```
##      height sex mother father
## 1:    197   M    175    191
## 2:    196   M    163    192
## 3:    195   M    171    185
## 4:    195   M    168    191
## 5:    194   M    164    189
## 6:    192   M    168    197
```

We can attempt to do so using linear regression. If we assign a 1 for the male category and a 0 for the female category, then we can fit a linear regression whose prediction gives us a value for the category and whose input is the height of the student. Let us plot the predicted linear regression line and the (height, sex) pairs.

```
heights[, y:=as.numeric(sex == "M")]
lm_fit0 <- lm(y~height, data=heights)
ggplot(heights, aes(height, y)) +
  geom_point() +
  geom_abline(intercept = lm_fit0$coef[1], slope = lm_fit0$coef[2], col="grey") +
  scale_y_continuous(limits=c(-0.5,1.5))
```



Looking at the figure, we realize that linear regression is not appropriate for this classification task. While y is only defined for the two classes 0 and 1, the regression line does not make that assumption.

We need to step back and consider a different modeling approach for categorical responses. Instead of modeling the response category directly, we could model the proportion of males per height. The following code does exactly this using 2-cm bins to stratify height:

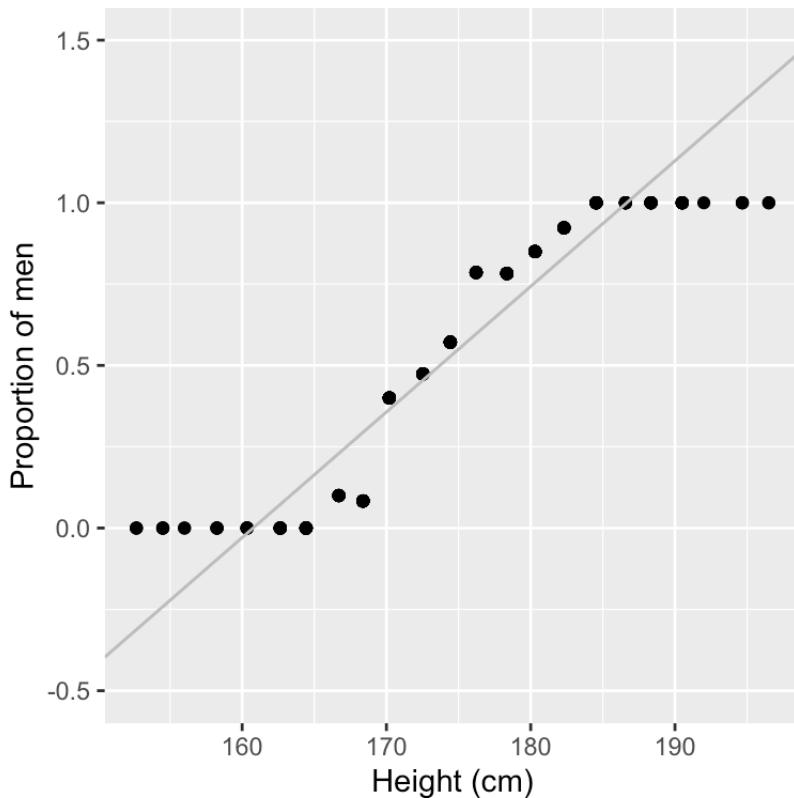
```

heights[, height_bins := cut(height, breaks=seq(min(height)-1, max(height), 2))]
heights[, mean_height_bins := mean(height), by=height_bins]
props <- heights[, prop:=mean(sex=="M"), by=height_bins]

lm_fit <- lm(prop ~ height, data=props)

ggplot(props, aes(mean_height_bins, prop)) +
  geom_point() +
  xlab("Height (cm)") +
  ylab("Proportion of men") +
  geom_abline(intercept = lm_fit$coef[1], slope = lm_fit$coef[2], col="grey") +
  scale_y_continuous(limits=c(-0.5,1.5))

```



Modeling the proportion of classes rather than the classes themselves already looks a lot better. However, we still have the problem that the predictions can be outside of the [0,1] interval where the proportions are not defined. Furthermore, the relationship seems to smoothly bend in an S-shape fashion between bins with only females for short heights, and bins with only males for tall heights. This S-shape is not well captured with a linear fit.

Let us now consider another scale for the response, in search for some better linear relationship. Instead of looking at the probabilities of male per stratum, let us consider the odds. The odds for a binary variable y are defined as $\frac{P(y)}{1-P(y)}$. We have introduced this concept when defining the Fisher's test 8.3.

We can estimate the overall population odds easily using the table of males and females:

```
table(heights$sex)
```

```
##  
##   F   M  
## 122 131
```

If we take as an estimate for $p(\text{male})$ the frequency of males in our dataset then, the odds for a student to be a male is simply estimated as the male to female ratio:

Population odds \simeq male:female = 131/122 = 1.07.

To investigate how the odds depend on height we can estimate them for each height stratum:

```
heights[, odds := sum(y==1)/sum(y==0), by=height_bins]
```

Plotting the odds in logarithmic scale suggests a linear relationship with height (Figure 11.1). Note that the odds are poorly estimated for high and low values of height, as there are no males in the low strata (division by 0) and no females in the high strata. Hence, we shall ignore the most extreme bins for now.

```
library(scales)  
breaks <- 10^(-10:10)  
minor_breaks <- rep(1:9, 21)*(10^rep(-10:10, each=9))  
  
p_log_odds <- ggplot(props, aes(mean_height_bins, odds,  
                           color=(!(odds==0 | is.infinite(odds)))) +  
  geom_point() +  
  xlab("Height (cm)") +  
  ylab("Male:Female") +  
  scale_y_log10(breaks=breaks, minor_breaks =minor_breaks) +  
  annotation_logticks(side="l") +  
  scale_color_manual(values=c("#999999","#000000")) +  theme(legend.position = "none")  
  
p_log_odds
```

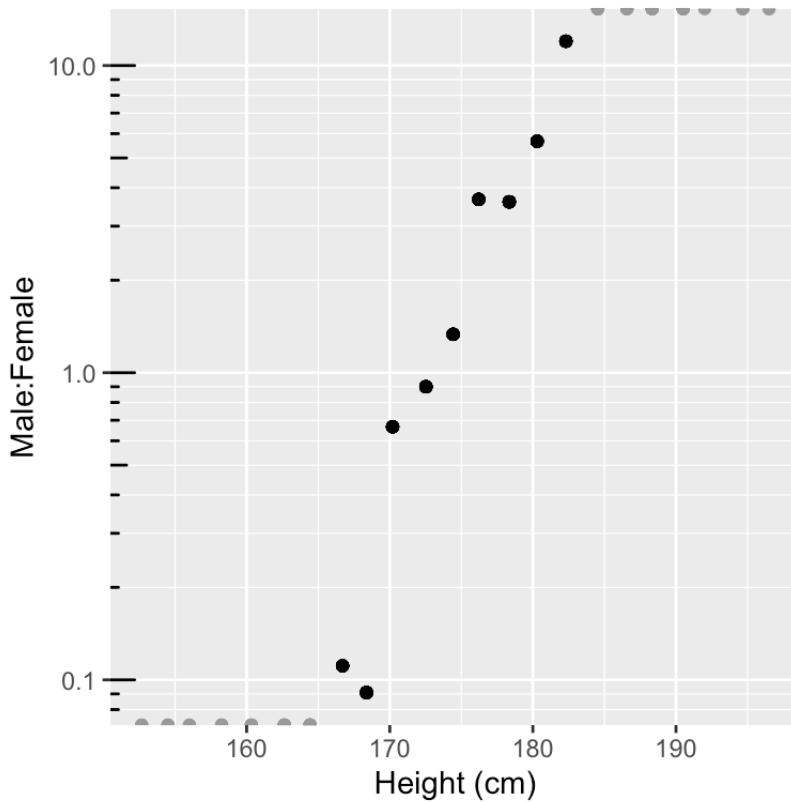


Figure 11.1: Odds male:female versus height (logarithmic y-scale). The grey halved dots indicate censored bins which have either 0 males or 0 females leading to infinite log-odds estimates.

These observations indicate that, approximately³⁹:

$$\log\left(\frac{p(\text{male})}{1 - p(\text{male})}\right) \simeq \beta_0 + \beta_1 \text{height} \quad (11.1)$$

This makes apparent a useful function for classification tasks, called the *logit* function, which is defined as :

$$\text{logit}(z) = \log\left(\frac{z}{1 - z}\right), z \in (0, 1).$$

Its reverse function, called the *logistic* function or *sigmoid* function is equally often used and important. The sigmoid function is denoted $\sigma(z)$ and it is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, z \in \mathbb{R}.$$

The sigmoid function is named so due to its S-shaped graph (Figure 11.2). It is symmetric and maps real numbers to the $(0, 1)$ interval. It is often found in statistics and in physics, as it can describe a variety of naturally occurring phenomena.

```
df <- data.table(z=seq(-5, 5, length.out=1000))
df[, y:=1/(1+exp(-z))]
ggplot(df, aes(z, y)) +
  geom_line() +
  xlab("z") +
  ylab(expression(sigma(z)))
```

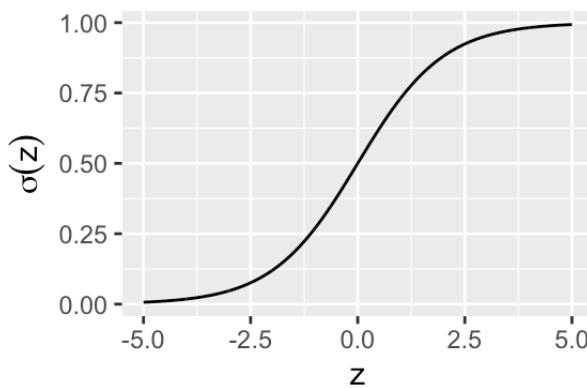


Figure 11.2: The logistic or sigmoid function

Hence, we found a **linear relationship!** We could in principle make predictions given height of log-odds and, passing them through the sigmoid, of probabilities of being a male. However, the approach we laid down is not satisfactory. It requires some arbitrary binning. We are not able to make use of the data in the most extreme bins which have either no male or no female. Also the log-odds of some bins are better estimated than others. Estimating odds of say 0.5 by observing 1 males and 2 females is not as precise as if we observe 100 males and 200 females. This uncertainty is not factored in with our approach. We need to step back to theory.

11.1.1 From linear regression to logistic regression

Remember that in Section 10.2.2, we mentioned that linear regression can be derived equivalently with the least squares criterion or with the maximum likelihood criterion. It turns out that it is the maximum likelihood criterion that will allow us to derive a principled way to model the classification problem. To see this, we start from the likelihood of the linear regression model and will adapt it to our new problem. This was (Section 10.2.2):

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}) &= \prod_i p(y_i|\mathbf{x}_i) \text{ conditional independence} \\ p(y_i|\mathbf{x}_i) &= N(y_i|\mu_i, \sigma^2) \\ \mu_i &= \beta_0 + \sum_{j=1}^p \beta_j x_{ij} \end{aligned} \tag{11.2}$$

Hence, **linear regression** models $\mu_i := E(y_i|x_{i1}, \dots, x_{ip})$, the **expectation** of the outcome **conditioned on the features**, in particular, such expectation is obtained as a **linear combination of the features**.

Rather than predicting an unbounded continuous outcome given some features, the **classification problem can be approached by predicting the probability of a class given some features** (ex. predicting the probability of the student being male given that the height is 160 cm). Modeling a probability with a linear function is not ideal because we can make predictions <0 or >1 . Logistic regression addresses this problem by using the logistic function discussed above, which maps linear combinations of features to the $(0,1)$ interval.

A logistic regression models the data:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}) &= \prod_i p(y_i|\mathbf{x}_i) \text{ conditional independence} \\ p(y_i|\mathbf{x}_i) &= B(y_i|1, \mu_i) \\ \mu_i &:= E[y_i|\mathbf{x}_i] = \sigma(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}) \end{aligned} \tag{11.3}$$

where $B(y_i|1, \mu_i)$ stands for the binomial distribution for 1 trial and probability μ_i , which in this particular case, where the number of trials is one, is also called a Bernoulli distribution. The Bernoulli distribution is a discrete distribution where there are two possible outcomes: failure ($y = 0$) or success ($y = 1$) (ex. 1 trial of tossing a coin: heads or tails). The success occurs with probability μ and failure occurs with probability $1 - \mu$.

In the case of logistic regression, the probability μ is the expectation of the success class ($y=1$) conditioned on the features, $E(y_i = 1|x_{i1}, \dots, x_{ip}) = \mu_i$.

We can alternatively write an equivalent expression to the latter by using the inverse function of the sigmoid, the *logit* function:

$$\eta_i := \text{logit}(\mu_i) = \beta_0 + \sum_{j=1}^p \beta_j x_{ij}$$

11.2 Maximum likelihood estimates and the cross-entropy criterion

As for linear regression, we estimate the parameters of the model using the maximum likelihood criterion. Plugging the binomial probability⁴⁰ and taking the logarithm, we obtain:

$$\begin{aligned} \arg \max_{\beta} \prod_i p(y_i|x_i, \beta) &= \arg \max_{\beta} \sum_i \log(B(y_i|1, \mu_i(x_i, \beta))) \\ &= \arg \min_{\beta} - \sum_i (y_i \log(\mu_i(x_i, \beta)) + (1 - y_i) \log(1 - \mu_i(x_i, \beta))) \end{aligned}$$

The term:

$$- \sum_i (y_i \log(\mu_i) + (1 - y_i) \log(1 - \mu_i))$$

is called the *cross-entropy* between the model predictions (the predicted probabilities μ_i) and the observations y_i .

Hence, maximum likelihood leads to a different minimization objective for classification than for linear regression. For classification, we are not minimizing the squared errors but the cross-entropy.

These minimization objectives are employed by a wide variety of models. Modern neural networks, which model complex non-linear relationships between input and output, also typically use cross-entropy to optimize their parameters for classification tasks and typically use the least squares criterion when it comes to quantitative predictions.

It turns out that there is no analytical solution to the maximum likelihood estimates of a logistic regression. Instead, algorithms are employed that numerically minimize cross-entropy until reaching parameter values that cannot be optimized further. For logistic regression, such algorithms are pretty fast and robust due to good mathematical properties of the problem which apply in typical real-life datasets.

11.3 Logistic regression as a generalized linear model

Generalized linear models (GLM) generalize linear regression by allowing the linear model to be related to the response variable via a *link* function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

As we have seen in the special case of the logistic regression, which is an instance of a GLM, we changed the linear regression by applying a transformation to the linear combination of the features in order to use it for another type of predictions.

- Logistic regression is one instance of generalized linear models, which all exploit the same idea:
 - 1. A probability distribution from the exponential family
 - Logistic regression: Bernoulli
 - 2. A linear predictor $\eta = \mathbf{X}\beta$
 - Logistic regression: $\text{logit}(\mu_i) = \eta_i = \beta_0 + \sum_{j=1}^p \beta_j x_{ij}$
 - 3. A link function g such that $E(y) = \mu = g^{-1}(\eta)$
 - Logistic regression: $g = \text{logit}$ and $g^{-1} = \text{sigmoid}$

The inverse of the link function is called the *activation* function, which in logistic regression is the logistic function.

Other popular examples of GLMs include Poisson regression⁴¹ and Gamma regression.

11.3.1 Logistic regression with R

To fit a logistic regression to our data we can use the function `glm`, which stands for generalized linear model, with the parameters below. The fitted model can be applied to data (seen or unseen) using `predict()`. By default it returns the linear predictor η , or in logistic regression, the logit of the predicted probabilities. Use `type='response'` to have the predicted probabilities on the natural scale:

```
logistic_fit <- glm(y ~ height, data=heights, family = "binomial")

heights[, mu_hat := predict(logistic_fit, heights, type="response")]
heights
```

```

##      height sex mother father y height_bins
## 1:    197   M    175    191 1  (195,197]
## 2:    196   M    163    192 1  (195,197]
## 3:    195   M    171    185 1  (193,195]
## 4:    195   M    168    191 1  (193,195]
## 5:    194   M    164    189 1  (193,195]
## ---
## 249:    152   F    150    165 0  (151,153]
## 250:    172   F    165    188 0  (171,173]
## 251:    154   F    155    165 0  (153,155]
## 252:    178   M    169    174 1  (177,179]
## 253:    175   F    171    189 0  (173,175]

##      mean_height_bins      prop      odds      mu_hat
## 1:        196.5000 1.0000000 Inf 0.9996635228
## 2:        196.5000 1.0000000 Inf 0.9995264608
## 3:        194.6667 1.0000000 Inf 0.9993336047
## 4:        194.6667 1.0000000 Inf 0.9993336047
## 5:        194.6667 1.0000000 Inf 0.9990622786
## ---
## 249:        152.6667 0.0000000 0.000000 0.0006193516
## 250:        172.5263 0.4736842 0.900000 0.3660058183
## 251:        154.5000 0.0000000 0.000000 0.0012262897
## 252:        178.3478 0.7826087 3.600000 0.8178215789
## 253:        174.4286 0.5714286 1.333333 0.6168344547

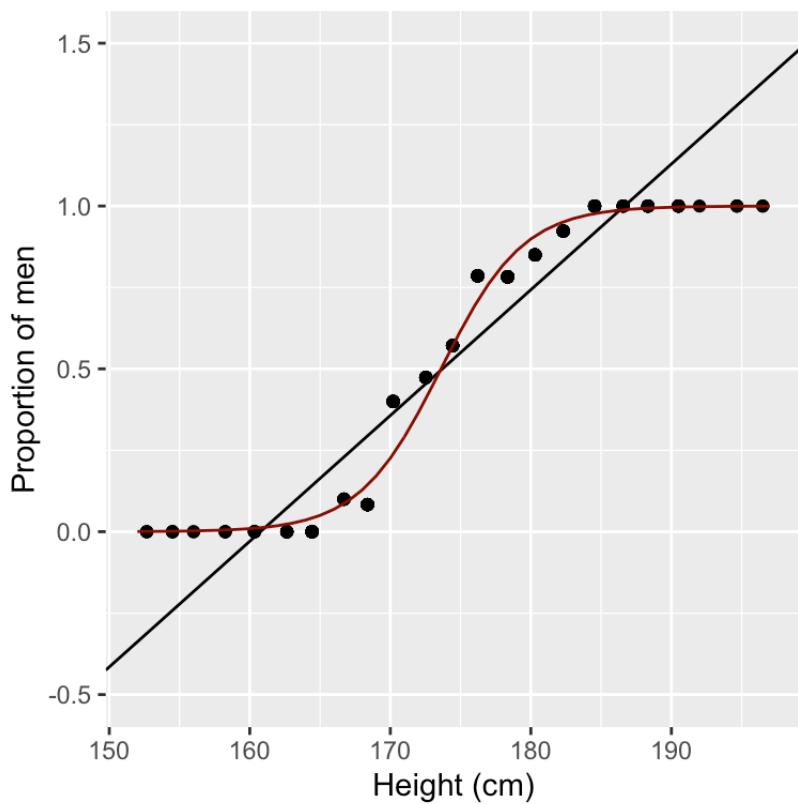
```

Here are the predicted values using logistic regression (red) compared to a linear regression of the proportions (black). Overall, logistic regression fits better to our data and fixes the issue of having predictions outside the [0,1] interval.

```

ggplot(props, aes(mean_height_bins, prop)) +
  geom_point() +
  xlab("Height (cm)") +
  ylab("Proportion of men") +
  geom_abline(intercept = lm_fit$coef[1], slope = lm_fit$coef[2]) +
  geom_line(aes(height, mu_hat), col='darkred') +
  scale_y_continuous(limits=c(-0.5,1.5))

```



11.3.2 Overview plot of the univariate example

We finish this section, for didactic purposes, with an overview plot (Figure 11.3) from the raw data down to the logistic fit.

```
library(patchwork)

ys <- 10

p_hist_male <- ggplot(heights[sex=='M'], aes(height)) +
  geom_histogram() +
  scale_x_continuous(limits=c(150,200)) +
  geom_vline(aes(xintercept = median(height)), color="red", linetype="dashed", size=0.5) +
  theme(axis.title.y = element_text(size = ys), axis.title.x=element_blank())+
  labs(x="Height (cm)", y="Number of Males")

p_hist_female <- ggplot(heights[sex=='F'], aes(height)) +
  geom_histogram() +
  scale_x_continuous(limits=c(150,200)) +
  geom_vline(aes(xintercept = median(height)), color="red", linetype="dashed", size=0.5) +
  theme(axis.title.y = element_text(size = ys), axis.title.x=element_blank())+
  labs(x="Height (cm)", y="Number of Females")

p_fit_male <- ggplot(props, aes(mean_height_bins, prop)) +
  geom_point() +
  labs(x="Height (cm)", y="Estimated P(Male)") +
  geom_line(aes(height, mu_hat), col='darkred') +
  theme(axis.title.y = element_text(size = ys), axis.title.x = element_text(size = 10))+ 
  scale_x_continuous(limits=c(150,200))

p_log_odds <- p_log_odds + scale_x_continuous(limits=c(150,200)) +
  theme(axis.title.y = element_text(size = ys), axis.title.x=element_blank())

p_hist_male / p_hist_female / p_log_odds / p_fit_male
```

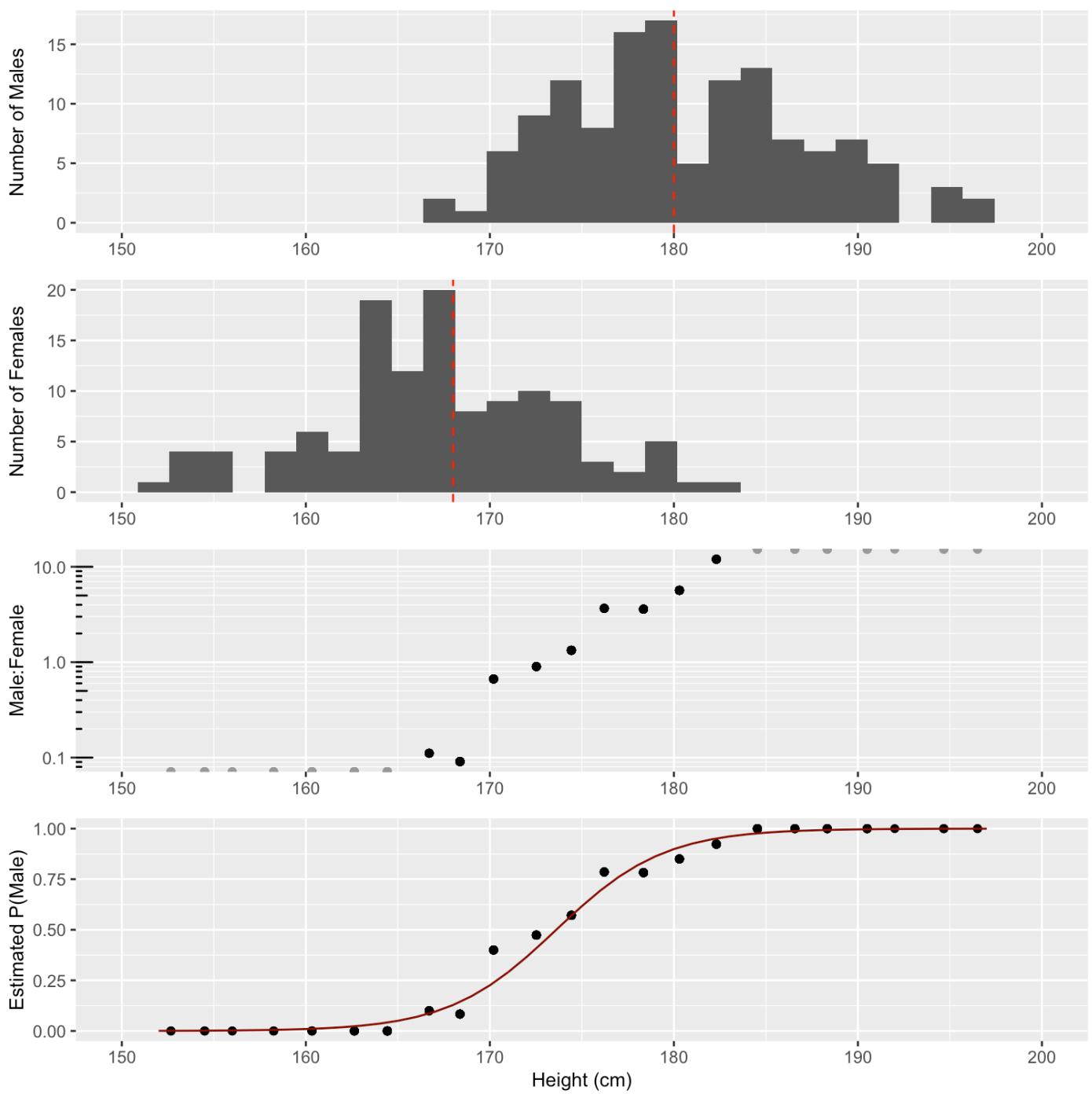


Figure 11.3: Overview of predicting sex from height. From top to bottom: distribution of heights for i) males and ii) females, iii) Male to female ratio in log-scale, and iv) proportion of males (dots) along with logistic regression fit (red curve). Note that while 2-cm bins are used throughout the plot for visualization purposes, the logistic regression fit in contrast is performed on the raw data, i.e. sex (0/1) versus height.

11.4 Interpreting a logistic regression fit

11.4.1 Predicted odds

In logistic regression, the logit of the predicted response/probability for a certain input is the predicted log odds for the positive class ($y=1$) on that input. For example, for a height of 178 cm the log odds is:

```
log_odds_178 <- predict(logistic_fit, data.table(height=178))
log_odds_178

##          1
## 1.501658
```

To get the odds we can exponentiate this number. The obtained result means that the odds for someone to be a male (positive class) at height 178 cm is:

```
exp(log_odds_178)

##          1
## 4.489124
```

11.4.2 Coefficients of the logistic regression

The β values from logistic regression are log odds ratios associated with an increase by one unit of the corresponding explanatory variable. Odds ratios can thus be obtained by applying `exp()`.

```
coef(logistic_fit)

## (Intercept)      height
## -59.3461056   0.3418414

OR_height <- exp(coef(logistic_fit)[2])
OR_height

##      height
## 1.407537
```

As we have seen, in logistic regression the log odds is predicted as a linear combination of the features, where the coefficients are log odds ratios. Therefore, in our example of predicting sex from height, increasing the height by h centimeters changes the log odds by $h \times 0.342$, or equivalently, it multiplies the odds by $e^{h \times 0.342} = 1.408^h$.

11.4.3 Effects on probabilities

One important difference between linear regression and logistic regression is that for the latter, the relationship between input and predicted value is not linear. A drastic example is shown in Figure 11.4, where increases of 5 cm produce different increases in the probability for male depending on the starting point.

If we start at odds 1:1, the probability is 0.5, and our corresponding height can be obtained by solving the equation for height on:

$$\log(\text{odds}) = \beta_0 + \beta_1 \text{height}$$

where β_0 and β_1 are the coefficients of the logistic regression. This gives us a height value of 173.6 cm. Note that the point of steepest increase in the logistic curve corresponds to the predicted probability of 0.5.⁴²

```
## (Intercept)
## 173.6072
```

An increase in 5 cm leads as to a **probability** of:

```
prob_5 <- predict(logistic_fit, data.table(height =173.6+5), type="response")
prob_5
```

```
## 1
## 0.8464159
```

If we increase 5 cm again, then we will have a predicted probability for male of:

```
prob_10 <- predict(logistic_fit, data.table(height =173.6+10), type="response")
prob_10
```

```
## 1
## 0.9681999
```

An increase in height from 173.6 cm to 178.6 cm increases the estimated probability by 0.35, while an increase in height from 178.6 cm to 183.6 cm increases the estimated probability by 0.12.

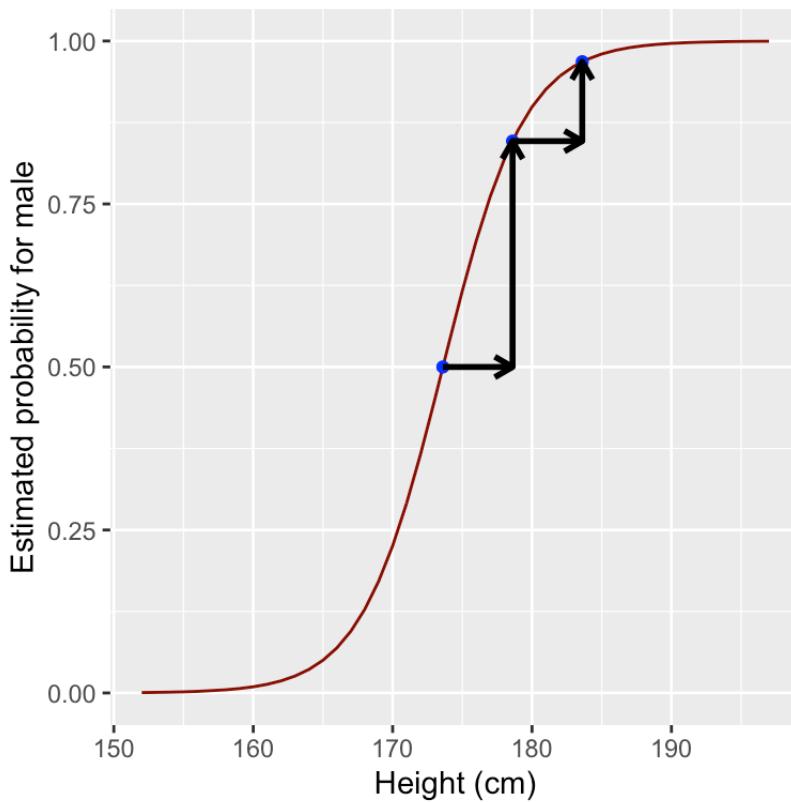


Figure 11.4: Estimated probability given the same 5 cm increase in height on different points of the logistic curve.

11.4.4 Class imbalance

Now we turn to a situation of *class imbalance*, which refers to having one class with substantially more instances than the other. Let's create a class imbalanced heights dataset with ~20% males and ~80% females.

```
set.seed(123)
imbalanced_heights <- heights[!sample(which(sex=="M")), 100]
imbalanced_heights[, table(sex)]

## sex
##   F   M
## 122  31
```

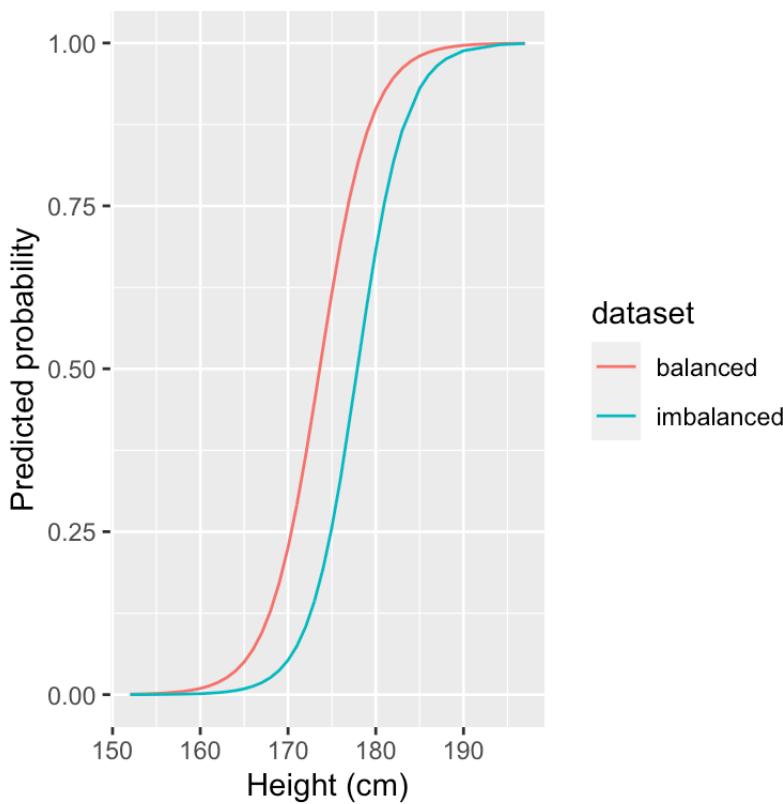
Now let's fit a logistic regression, similar to previous steps:

```
logistic_fit_imbalanced <- glm(y ~ height, data=imbalanced_heights, family = "binomial")
logistic_fit_imbalanced
```

```
##  
## Call: glm(formula = y ~ height, family = "binomial", data = imbalanced_heights)  
##  
## Coefficients:  
## (Intercept)      height  
## -64.9762        0.3652  
##  
## Degrees of Freedom: 152 Total (i.e. Null); 151 Residual  
## Null Deviance:    154.2  
## Residual Deviance: 72.43    AIC: 76.43
```

Now we can plot our logistic fit to the imbalanced dataset and compare it to our previous logistic fit obtained in the balanced dataset.

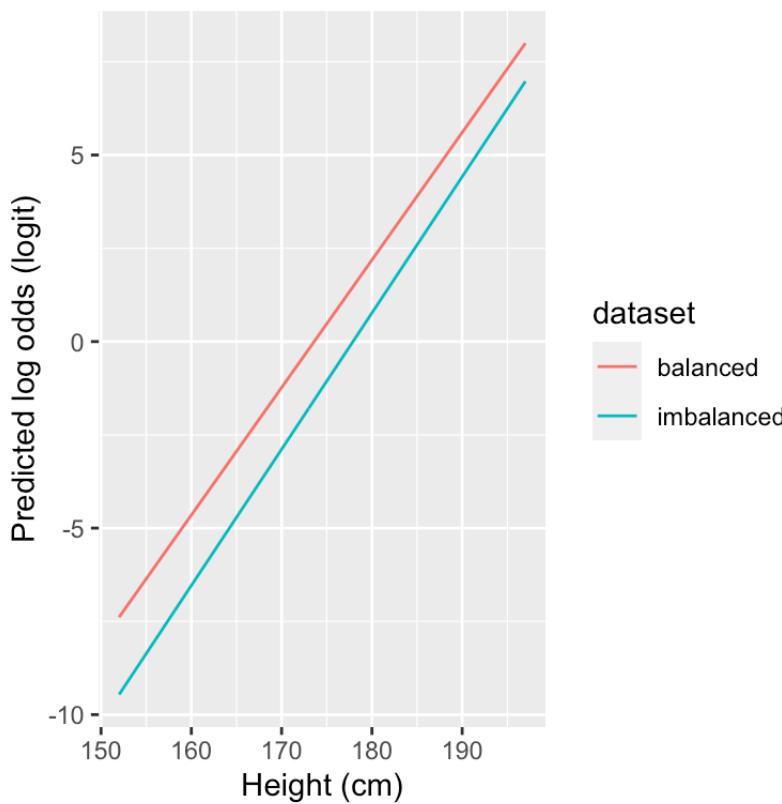
```
imbalanced_props <- imbalanced_heights[, prop:=mean(sex=="M"), by=height]  
  
imbalanced_props[, mu_hat := predict(logistic_fit_imbalanced, imbalanced_props, type="response")]  
  
imbalanced_props[, dataset=="imbalanced"]  
props[, dataset=="balanced"]  
  
rbind(imbalanced_props, props, fill=T) %>%  
ggplot(aes(height, mu_hat, color=dataset)) +  
  geom_line() +  
  xlab("Height (cm)") +  
  ylab("Predicted probability")
```



The logistic fit on the imbalanced dataset looks different. In particular, the curve has been shifted to the right. The predicted probability of being male for a given height is lower compared to the balanced logistic fit. Let's look at the logit of the predicted response and compare it to the previous balanced dataset.

```
imbalanced_props[, logistic_logit := predict(logistic_fit_imbalanced, imbalanced_props)]
props[, logistic_logit:=predict(logistic_fit, props)]

rbind(imbalanced_props, props, fill=T) %>%
ggplot(aes(height, logistic_logit, color=dataset)) +
  geom_line() +
  xlab("Height (cm)") +
  ylab("Predicted log odds (logit)")
```



As seen from this figure, the intercept of the logistic regression is now lower and the lines are nearly parallel. This means that the estimated log odds for male in the imbalanced logistic fit are generally lower. Indeed, if we have lower quantities of males for each value of height in general, then the odds for male will be overall lower for all heights.

Why can the effect of class imbalance be seen on the intercept? While the proportion among males in each stratum are independent of the number of males, the ratios of males over females scales with the overall number of males. Hence, The odds male:female per stratum proportionally change when the overall population odds change. In a log scale this translates into an added constant and thus a vertical shift. In other words, changes in class imbalance affects the intercept β_0 of a logistic regression by adding a constant.⁴³

11.4.5 Multiple Logistic regression

We have been performing logistic regression with one variable as input to predict one class. However, in our formulation of logistic regression we allowed a set of features to predict the probability of the class. In multiple logistic regression there can be several input variables. We can use multiple logistic regression to predict the student's sex given the heights of the student and of the student's parents.

```
multi_logistic_fit <- glm(y ~ height + mother + father, data=heights, family = "binomial")
multi_logistic_fit
```

```

## 
## Call: glm(formula = y ~ height + mother + father, family = "binomial",
##           data = heights)
## 
## Coefficients:
## (Intercept)      height      mother      father
## -37.4913       0.7307     -0.2899     -0.2360
## 
## Degrees of Freedom: 252 Total (i.e. Null);  249 Residual
## Null Deviance:      350.4
## Residual Deviance:  83.05    AIC: 91.05

```

Similarly to the previous univariate logistic fit, an increase in the student's height increases the odds for male. Furthermore, this model gives negative coefficients to the height of the mother and of the father. This makes sense: the taller either parent, the taller the children. It is therefore more likely that a tall person is a female, if the parents are tall.

Is this model, which integrates the heights of the parent, doing a better job at predicting sex than the simpler model we first developed? And if so by how much? We will see now ways to answer these questions.

11.5 Assessing the performance of a classifier

11.5.1 Classification with logistic regression

Logistic regression predicts a probability, therefore it conveys some uncertainty in the prediction. Classification implies that we attribute one class per instance, and not a probability.

Hard classification is usually performed by the following simple rule: If $\mu > 0.5$ (or equivalently $\eta > 0$), predict class 1, else predict class 0.

Let's follow this rule for the previous multiple logistic regression by using the function `round`. Moreover, let's inspect the predictions of the classes male (0) and female (1) by computing a contingency table between the predicted and true classes:

```

heights[, y_multi_pred := round(predict(multi_logistic_fit, heights, type="response"))]
heights[, table(y, y_multi_pred)]

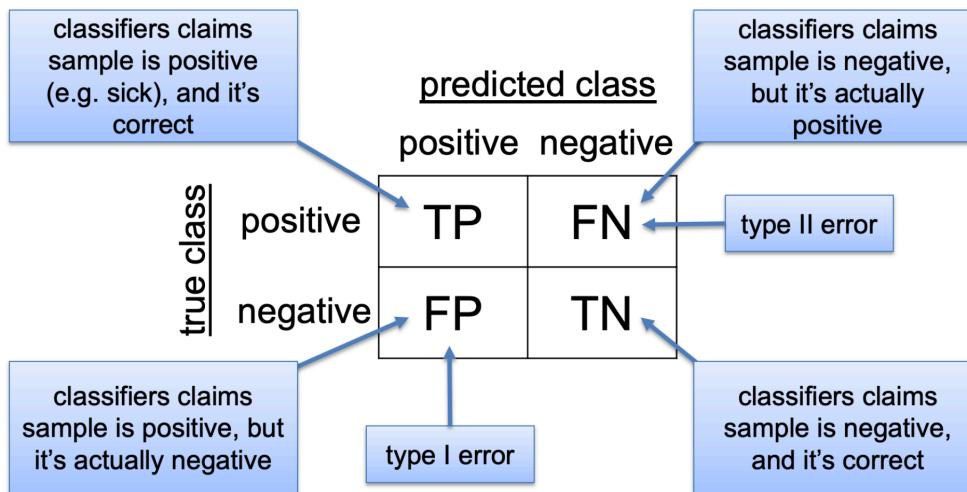
##      y_multi_pred
##      y      0      1
##      0 113    9
##      1 10 121

```

10 males where predicted as female (false negative) and 9 females were predicted as male (false positive).

11.5.2 Confusion Matrix

The previous table is also termed confusion matrix. The following figure depicts the structure of a confusion matrix:



Such matrix allows us to access the quality of the classifications. A classifier should maximize true positives (TP) and true negatives (TN), and minimize false negatives (FN) and false positives (FP)

11.5.3 Classification performance metrics

From the confusion matrix, various quality metrics have been defined. Moreover, depending on the application domain, the same quantities are referred to with different names.⁴⁴ We focus here on three metrics:

- The *sensitivity* refers to the fraction of actual positives that is predicted to be positive:

$$\text{Sensitivity} = \frac{TP}{P} = \frac{TP}{TP+FN}$$

The sensitivity is also referred to as “recall”, “true positive rate”, or “power”.

- The *specificity* refers to the fraction of actual negatives that is predicted to be negative:

$$\text{Specificity} = \frac{TN}{N} = \frac{TN}{TN+FP}$$

The specificity is also known as “true negative rate” or “sensitivity of the negative class”

- The *precision* refers to the fraction of predicted positives that are indeed positives:

$$\text{Precision} = \frac{TP}{TP+FP}$$

The precision is also called the positive predictive value. Note that, in the hypothesis testing context, we discussed a related concept, the false discovery rate (FDR). The FDR relates to the precision as follows:

$$\text{FDR} = E\left[\frac{FP}{TP+FP}\right] = E\left[1 - \frac{TP}{TP+FP}\right] = 1 - E[\text{Precision}]$$

11.5.4 Choosing a classification cutoff

Many classification methods do not directly assign a class, but rather output a quantitative score. For instance, the logistic regression predicts a probability. Other methods may output a real number that is not aimed to represent a probability yet for which the larger, the more likely the class is positive. This would be the case if we had insisted in using linear regression as we first attempted at the start of this Chapter, but it is also the case if we use the logit instead of the predicted probability in logistic regression or for modern classifiers like support-vector machines.⁴⁵

Hence, we typically need to set a *cutoff* above which we classify as “positive”. Previously we defined our cutoff at $\mu = 0.5$ or $\text{logit}(\mu) = 0$, but it was an arbitrary decision, we could have chosen a different one. The choice of the cutoff influences the performance metrics.

Let's go back to the example of predicting the sex of the student given its height and consider the logits of the logistic regression as scores. If the logistic regression predictions are somewhat informative, we expect the distribution of the scores for the negative and positive classes to be to some extent separated. Indeed, as depicted in Figure (11.5), most students can be correctly separated in male or female given its score. However, a certain amount of students from different sexes have overlapping scores.

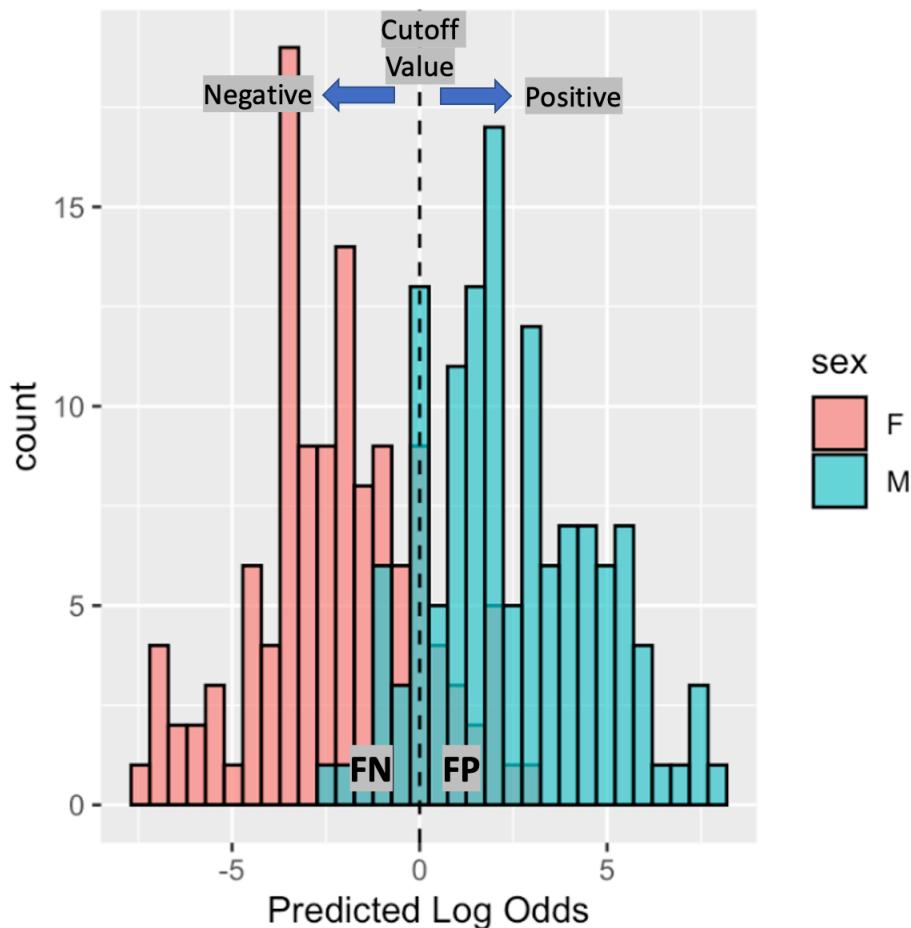


Figure 11.5: When informative, a classifier score separates to some extent the positive from the negative class. Such can be seen in the logistic regression model for sex prediction given height. The choice of a classification cutoff leads to various types of correct and incorrect predictions.

As seen from the previous figure, picking a cutoff at a certain value like the one in the dashed line, will produce classifications which are false positives or false negatives.

Now imagine we move our cutoff in the way of the next figure:

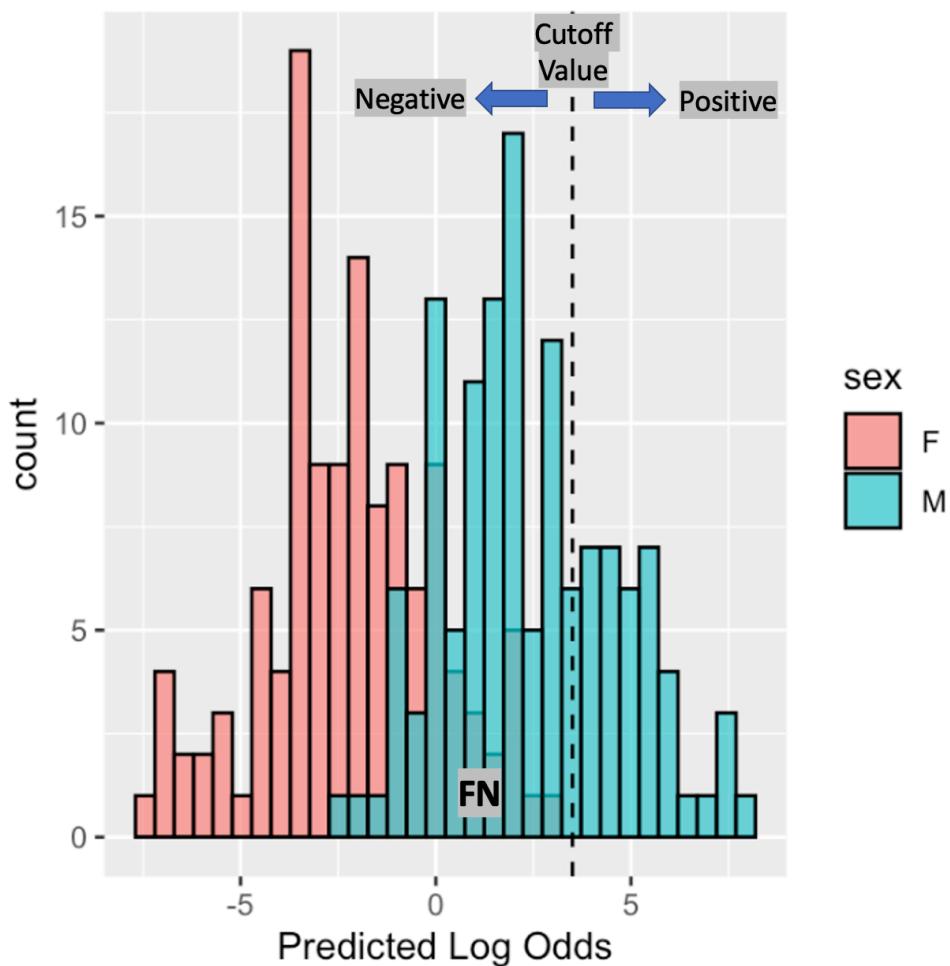


Figure 11.6: Setting a classification cutoff is always a trade-off between sensitivity and specificity.

Now, every female student is classified as such (True negative (TP)), however some male students are classified as female (False negative (FN)). Choosing the cutoff is a trade-off between sensitivity and specificity. How the trade-off is made is problem-dependent.

11.5.5 ROC curve

The receiver operating characteristic curve or ROC curve is a way of evaluating the quality of a binary classifier at different cutoffs.

It describes on the x axis the false positive rate (1-specificity), $FPR = \frac{FP}{N} = \frac{FP}{FP+TN}$ and on the y axis the true positive rate (sensitivity), $TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$

Let's plot the ROC curve for the univariate logistic regression model we fitted on heights and the ROC curve. As comparisons we look also at a random classifier (i.e. a model that outputs random values for probabilities of positive class). To do this we use `geom_roc` which comes from the `plotROC` package.

```

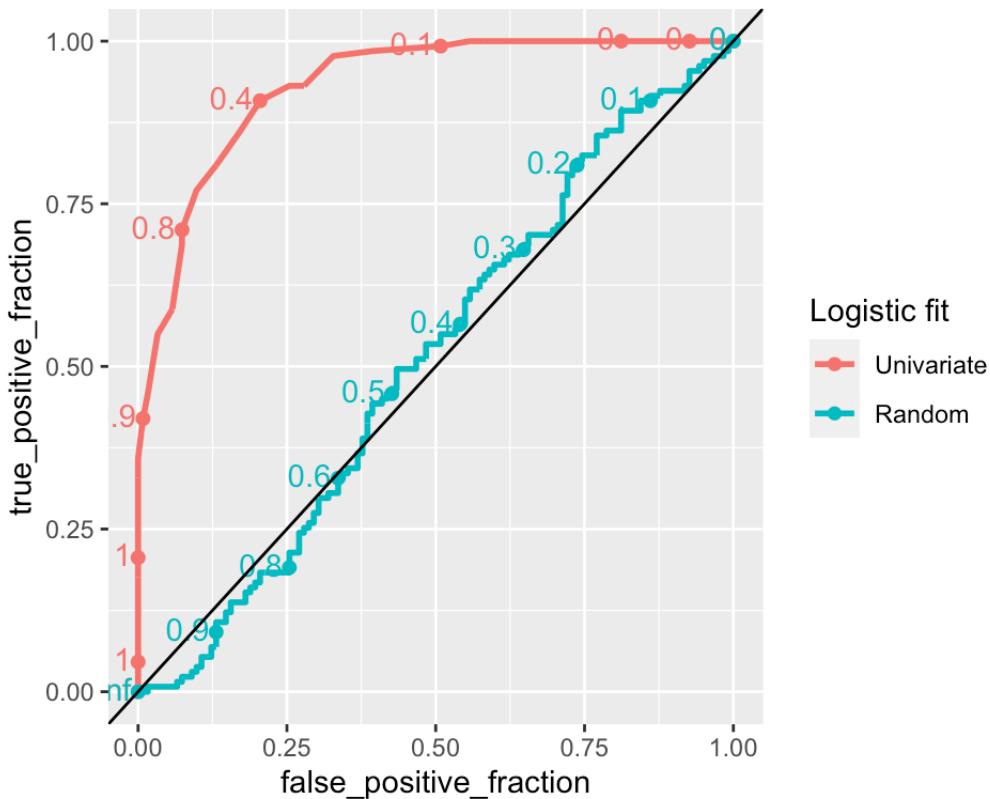
library(plotROC)
heights[, random_scores:=runif(.N)]

heights_melted <- heights[, .(y, mu_hat, random_scores)] %>%
  melt(id.vars="y", variable.name = "logistic_fit", value.name="response")

ggroc <- ggplot(heights_melted, aes(d=y, m=response, color=logistic_fit)) +
  geom_roc() +
  scale_color_discrete(name = "Logistic fit", labels = c("Univariate", "Random")) +
  geom_abline()

ggroc

```



The points along the lines represent the cutoff value. If all instances are classified as positive (cutoff=0) then the false positive rate is 1 and so is the true positive rate. On the other hand, if all instances are classified as negative (cutoff=1) then, the false positive rate is 0 and so is the true positive rate.⁴⁶ A random classifier has a ROC curve which always approximates a diagonal, such can be seen in the previous figure as the close to diagonal line, while the other curve is the ROC for the logistic prediction.⁴⁷ Now let's compare the ROC curve on the univariate logistic regression against the one on the multiple logistic regression:

```

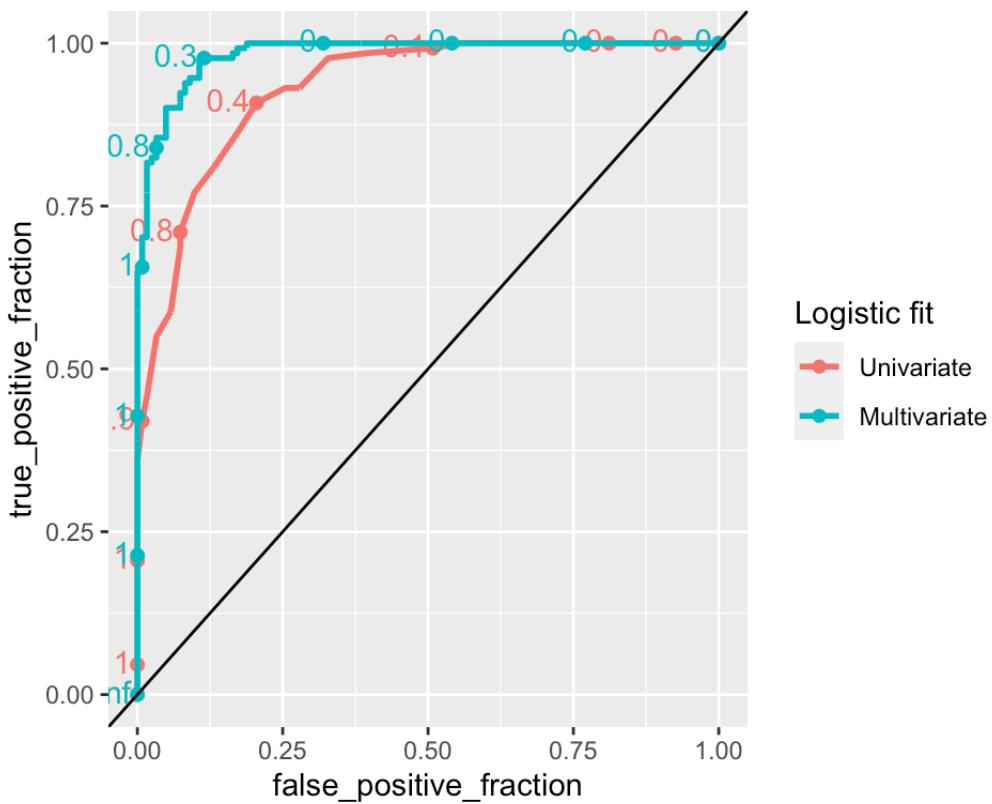
heights[, multi_logistic_mu_hat := predict(multi_logistic_fit, heights, type = "response")]

heights_melted <- heights[, .(y, mu_hat, multi_logistic_mu_hat)] %>%
  melt(id.vars="y", variable.name = "logistic_fit", value.name="response")

ggroc <- ggplot(heights_melted, aes(d=y, m=response, color=logistic_fit)) +
  geom_roc() +
  scale_color_discrete(name = "Logistic fit", labels = c("Univariate", "Multivariate")) +
  geom_abline()

ggroc

```



We can see that for lower values of false positive rates, the true positive rate is higher on the multiple logistic regression model. Such suggests this model has a better performance. Classification performance can also be measured by the AUC (area under the ROC curve). An AUC of 1 means that the model is perfectly able to distinguish between the positive and negative classes. An AUC of 0.5, which corresponds to the AUC of a ROC curve which is a diagonal line, means the classifier model is no better than random classification.

```
calc_auc(ggroc)
```

```

##   PANEL group      AUC
## 1     1    1 0.9313290
## 2     1    2 0.9833563

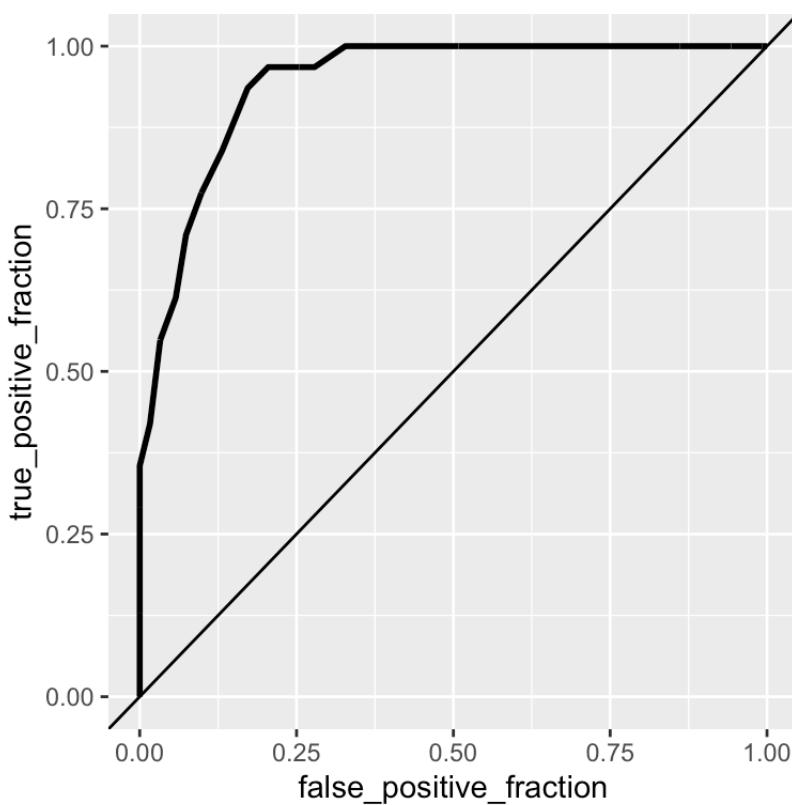
```

The AUC for the multiple logistic regression is ~ 0.983 , indicating a better classification performance compared to the univariate logistic regression ($AUC \sim 0.931$), which only takes the height of the student to predict its sex.

11.5.6 Precision Recall curve

Let's compute the ROC curve and its AUC on the logistic regression on the imbalanced dataset.

```
imbalanced_heights[, mu_hat:=predict(logistic_fit_imbalanced, imbalanced_heights, type="response")]
ggroc<- ggplot(imbalanced_heights, aes(d=y, m=mu_hat)) +
  geom_roc(n.cuts=0) +
  geom_abline()
ggroc
```



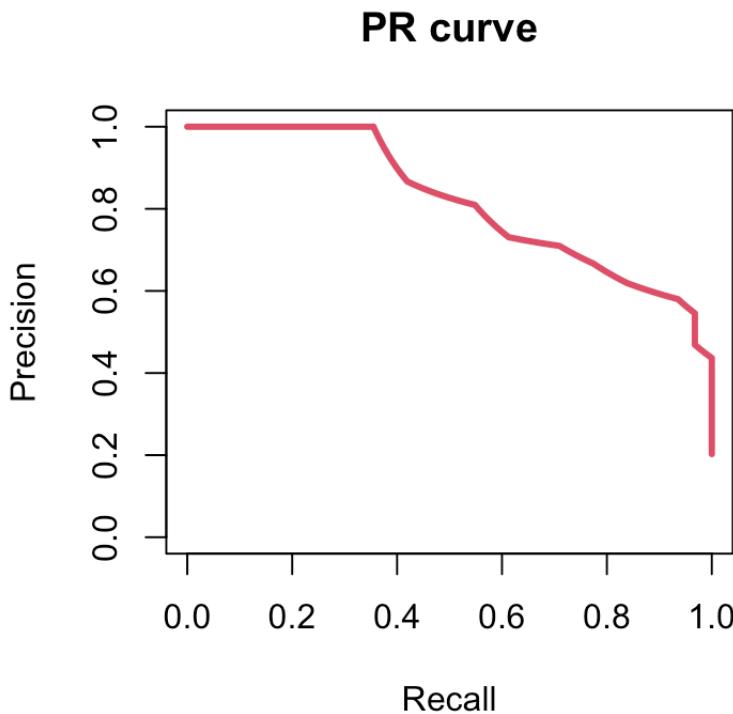
```
calc_auc(ggroc) ["AUC"]
```

```
##          AUC
## 1 0.9435484
```

From its AUC value, the performance of this model seems similar to the model on the balanced dataset. Let's inspect such performances using a Precision Recall curve, which, for different cutoffs plots the precision ($\frac{TP}{TP+FP}$) against the recall ($\frac{TP}{P}$). To plot a precision recall curve we can use the package PRROC.

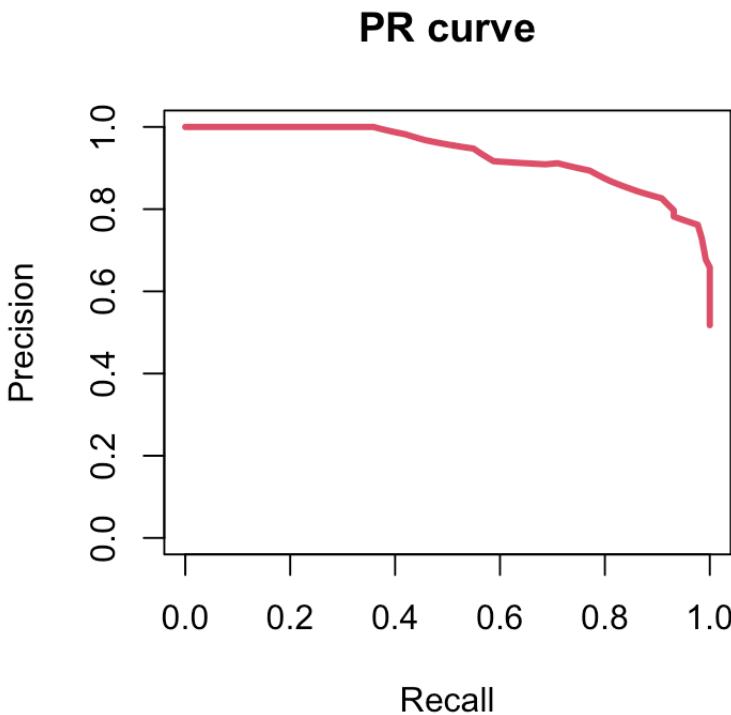
```
library(PRROC)
```

```
PRROC_obj <- pr.curve(scores.class0 = imbalanced_heights$mu_hat, weights.class0=imbalanced_heights$y,
                        curve=TRUE)
plot(PRROC_obj, auc.main=FALSE , color=2)
```



Now let's plot the precision recall curve of the logistic regression model on the balanced dataset:

```
PRROC_obj <- pr.curve(scores.class0 = heights$mu_hat, weights.class0=heights$y,
                        curve=TRUE)
plot(PRROC_obj, auc.main=FALSE , color=2)
```



Although the ROC curves of both models look similar, the precision recall curves look different. Because the dataset is imbalanced for the female (negative class), the corresponding logistic regression model provides lower precisions for the same recall, meaning that such model is worse at classifying males. The PR curves is hence used in very imbalanced situations in which the positive class is strongly under-represented. These are “finding a needle in a haystack” situations, such as detecting a rare disease or retrieving a relevant web page among billions of web pages across the web. Then, the precision-recall curves emphasizes better the performance of the model among the top scoring predictions than the ROC curve.

11.6 Conclusions

11.6.1 To remember

Now you should be able to:

- define and identify classification problems
- know the modeling assumption of logistic regression
- know the criterion of minimal cross-entropy
- fit a logistic regression in R, extract coefficients and predictions
- interpret coefficients of logistic regression fits
- know the definitions of TP, TN, FP, FN
- know the definitions of sensitivity, specificity, and precision
- know what a ROC curve is
- know how to compare performance of classifiers using a ROC curve
- know what a PR curve is

- know how to compare performance of classifiers using a PR curve
39. It turns out that Equation (11.1) is exact assuming that the distribution of the height for either sex is Gaussian and that both have the same variance. A good theoretical exercise is to prove it. As a hint use Bayes theorem. Moreover, these assumptions seem to hold in our data. Check it. Hint: use qqnorm(), qqline() and the sd() functions. ↵
40. $B(y_i|1, \mu_i) = \binom{1}{y_i} \mu_i^{y_i} (1 - \mu_i)^{1-y_i}$ ↵
41. Used for instance to model football scores. See, e.g. <http://opisthokonta.net/?p=276> ↵
42. How can you find the height corresponding to probability=0.5? Can you prove this is the point of steepest increase?
↪
43. A formal proofs can be made using Bayes theorem. ↵
44. In doubt, wikipedia provides a comprehensive overview on classifier performance metrics
https://en.wikipedia.org/wiki/Sensitivity_and_specificity ↵
45. https://en.wikipedia.org/wiki/Support-vector_machine ↵
46. How would the ROC curve look like if we took plain height as a predictive score and not the logistic regression prediction? ↵
47. Can you prove why? ↵

Chapter 12 Supervised Learning

12.1 Introduction

12.1.1 Motivation

In the last two chapters we introduced linear and logistic regression models. The primary motivation lied in the interpretation of the coefficients, seen as the effect of explanatory variables on the response adjusted for the effects of other, potentially correlated, variables. Under some assumptions this even allowed us – holy grail – to conclude about conditional independence. Often, these models were considered as interpretable data generative models, supposed to reflect the underlying process.

In this chapter, we will take a supervised machine learning angle. Here, we are interested in good predictions rather than identifying the most predictive features or drawing conclusions about statistical independence. On the one hand, the goal is less ambitious. On the other hand, focusing on prediction accuracy alone allows unleashing a wide variety of modeling techniques. The application area is immense and contains the most prominent successes of today's artificial intelligence including medical diagnosis, image recognition, automatized translation, etc. We will learn how to evaluate the prediction performance on an independent test set and avoid problems related to poor generalization of a model to an independent test set. Additionally, we will introduce random forests as an alternative machine learning model for both regression and classification tasks.

12.1.2 Supervised learning vs. unsupervised learning

Machine learning is the study of computer algorithms that improve automatically through experience⁴⁸. Most machine learning problems can be mainly categorized into two: supervised or unsupervised problems. In **supervised learning**, the goal is to build a powerful algorithm that takes feature values as input and returns a prediction for an outcome, even when we do not know the value for the actual outcome. For this, we *train* an algorithm using a data set for which we know the outcome, and then use this trained model to make predictions. In particular, for building the model, we associate each set of feature values to a certain outcome. In **unsupervised learning**, we do not associate feature values to an outcome. The situation is referred to as unsupervised because we lack an outcome variable that can supervise our analysis and model building. Instead, unsupervised learning algorithms identify patterns in the distribution of data. Typically, clustering problems and dimensionality reduction are attributed to unsupervised machine learning problems.

A powerful framework to study supervised and unsupervised learning is statistical learning, which express these problems in statistical terms. Supervised learning then maps to the task of fitting conditional distribution $p(y|x)$ where y is the outcome and x are the features (or fitting aspects of the conditional distribution: the expectation, etc.). Linear

regression is one method we already saw. Unsupervised learning in contrast refers to the task of fitting the distribution of the data $p(\mathbf{x})$, or some aspects of it (covariance: PCA, mixture components: clustering, etc.)

This chapter focuses on supervised learning. In fact, logistic and linear regression are two classical methods in supervised learning for solving classification and regression tasks.

12.1.3 Notation

In this chapter, we will use the classical machine learning nomenclature:

- y denotes the *outcome* (or response) that we want to predict
- x_1, \dots, x_p denote the *features* that we will use to predict the outcome.

12.1.4 Basic approach in supervised machine learning

In regression and classification tasks, we have a series of features and an unknown numeric or categorial outcome we want to predict:

outcome	feature_1	feature_2	feature_3	feature_4	feature_5
?	x_1	x_2	x_3	x_4	x_5

To build a model that provides a prediction for any set of observed values x_1, x_2, \dots, x_5 , we collect data for which we know the outcome:

outcome	feature_1	feature_2	feature_3	feature_4	feature_5
y_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$
y_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$
y_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$
y_4	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$	$x_{4,5}$
y_5	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$	$x_{5,5}$
y_6	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$	$x_{6,5}$
y_7	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	$x_{7,4}$	$x_{7,5}$
y_8	$x_{8,1}$	$x_{8,2}$	$x_{8,3}$	$x_{8,4}$	$x_{8,5}$
y_9	$x_{9,1}$	$x_{9,2}$	$x_{9,3}$	$x_{9,4}$	$x_{9,5}$
y_{10}	$x_{10,1}$	$x_{10,2}$	$x_{10,3}$	$x_{10,4}$	$x_{10,5}$

We use this data to train the model and then use the trained model to apply to our (new) data for which we do not know the outcome.

Now... how do we evaluate our model? And how do we make sure our model will perform well on our (new) data?

12.2 Over- and Under-fitting

Analyzing whether our built model is generalizing well and capturing the trend of the data is an essential aspect of supervised machine learning. Two situations should be avoided: (1) that the model does not capture the trends of the data resulting in a high measured error between actual and predicted outcomes (under-fitting) and (2) that the model does not generalize well but fits the data used for training the model too well (over-fitting).

12.2.1 Example: polynomial curve fitting

As an example, we consider a polynomial curve fitting as a regression task with a dataset of $n = 10$ points shown in Figure 12.1 , each comprising an observation of the input variable x along with the corresponding outcome variable y . In this example, the data is generated from the function

$$f(x) = \sin(2\pi x)$$

(shown in blue) with added random noise from a normal distribution.

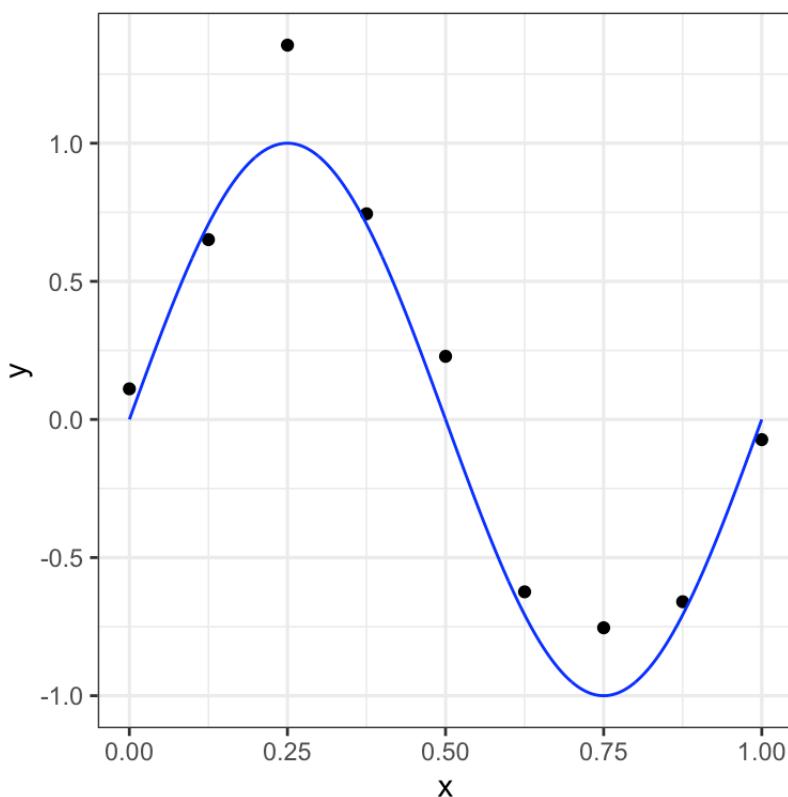


Figure 12.1: Data set of $N = 10$ points with a blue curve showing the function $\sin(2\pi x)$ used to generate the data

Assuming that we do not know the function plotted in the blue curve, the goal is to predict the value of y for some new arbitrary value of x . For this, we want to exploit the given dataset of n data points in order to make predictions of the value y of the outcome variable for some new value x , which involves implicitly trying to discover the underlying

function $f(x) = \sin(2\pi x)$. This is a difficult problem considering that we have to generalize from a finite dataset of only n points. Furthermore, the given data points are include noise, and so for a given x , there is a certain uncertainty regarding the correct value for y .

We consider a simple approach based on curve fitting to make predictions for a given value of x . More precisely, we fit the data using a polynomial function of the form:

$$y(x, \beta) = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_m x^m = \sum_{j=0}^m w_j x^j$$

Here, m is the order of the polynomial function and x^j denotes x raised to the power of j . The polynomial coefficients β_0, \dots, β_m are denoted by the vector β . Note that, the polynomial function $y(x, \beta)$ is a nonlinear function of x but it is a linear function of the coefficients β . Hence, a linear regression model (see Chapter 10) can be applied to find optimal values of β assuming that we have selected a value for the order m of the polynomial fit.

As described in Chapter 10, an optimal linear model (with optimal β values) can be found by minimizing an error function (sum of squared errors) that measures the misfit between the function $y(x, \beta)$ and the data points in the considered dataset.

Selecting different values for the order m of the polynomial has a huge impact on the resulting model. In Figure 12.2, we show four examples of the results of fitting polynomials with different orders $m = 0, 1, 3$ and 9 to our dataset of $n = 10$ points.

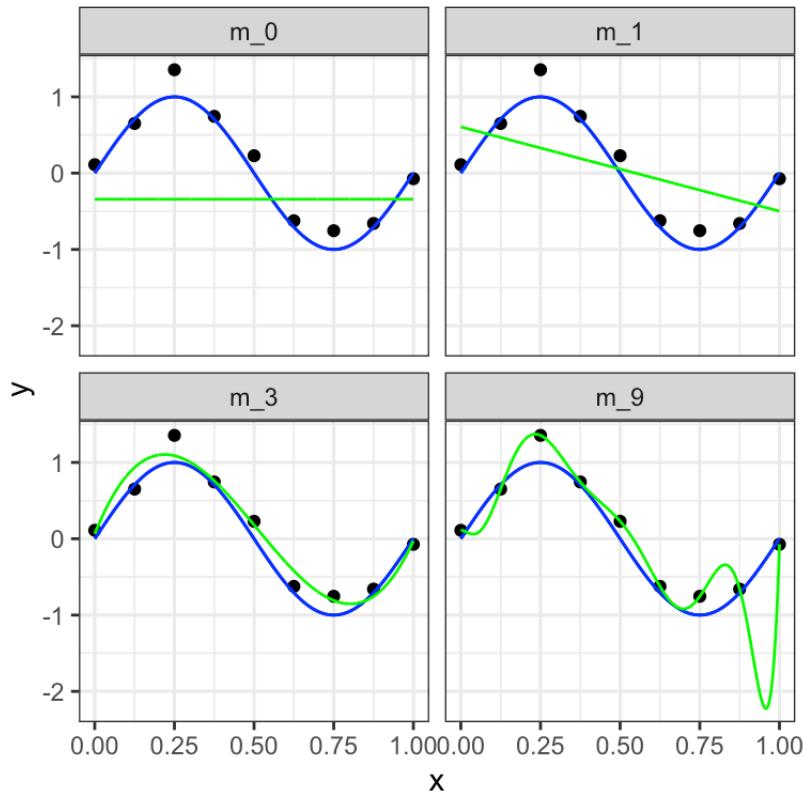


Figure 12.2: Plots of polynomials having various orders M , shown as green curves, fitted to the considered data set with underlying expected value function in blue.

We can clearly see that the constant ($m = 0$) and first order ($m = 1$) polynomials give rather poor fits to the data and consequently rather poor representations of the function $\sin(2\pi x)$. These two polynomial fits fail to capture the underlying trend of the data. A high error can be computed between the predicted and the actual data samples. These

are examples of under-fitting.

The third order ($m = 3$) polynomial seems to give the best fit to the function $\sin(2\pi x)$. By selecting $m = 3$ in the previous example, we observe an appropriate balance between capturing the trends in the data to achieve good generalization and making accurate predictions for outcome values.

When we go to a much higher order polynomial ($m = 9$), we obtain a polynomial that passes exactly through each data point. However, the fitted polynomial fails to give a good representation of the function $\sin(2\pi x)$. The fitted curve considers each deviation in the data points (including noise). The model is too sensitive to noise and captures random patterns which are present only in the current dataset. Poor generalization to other datasets is expected. This latter behavior is known as over-fitting.

It is particularly interesting to examine the behavior of a given model as the size of the data set varies. We can see that, for a given model complexity, the over-fitting problem become less severe as the size of the data set increases. In our example, by considering not only 10, but $n = 15$ or $n = 100$ data points, we observe that increasing the size of the data set reduces the over-fitting problem (see Figure 12.3).

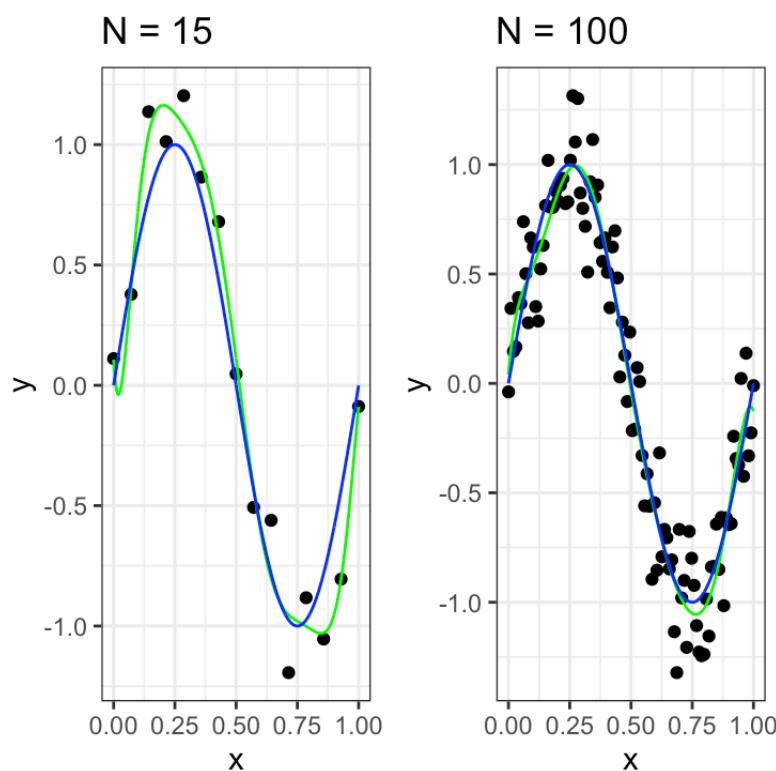


Figure 12.3: Polynomial curve fitting with $n = 15$ and $n = 100$ data points and polynomial order $m = 9$ in green and $\sin(2\pi x)$ in blue.

This example, borrowed from the Christopher Bishop's textbook, is very informative about the approach taken in and the issues faced by supervised learning.

First, supervised learning methods are more or less agnostic of the underlying processes. In our example, the underlying function is a sine. We may be looking at tides, an oscillating spring, etc. A physicist could come up with a model that leads to the right class of mathematical function. In supervised learning, we simply work with a generic class of flexible mathematical functions (here the polynomials, but there are many others), which can in principle approximate any function. One implication is that we will not aim at interpreting the coefficients. We would not pretend that the coefficient of say x^4 has any physical meaning. A physicist would (and could!) in contrast interpret the amplitude or the period of

the sine. In supervised learning, what guides our choice of the fit is data and only data. In particular, the complexity of the function we chose (here the order of polynomials) depends on how much data we have and not on theoretical considerations about the underlying process.

Second, the key issue is that it is easy with flexible mathematical functions to fit extremely well to a dataset. The challenge is not reducing the error on data at hand but the error on unseen data. Expected error on unseen data is called the **generalization error**. We will now investigate practical techniques to control the generalization error.

12.3 Splitting the dataset for performance assessment

As previously stated, a supervised learning task starts with an available dataset that we use to build a model so that this model will eventually be used in completely independent datasets, as shown in Figure 12.4.

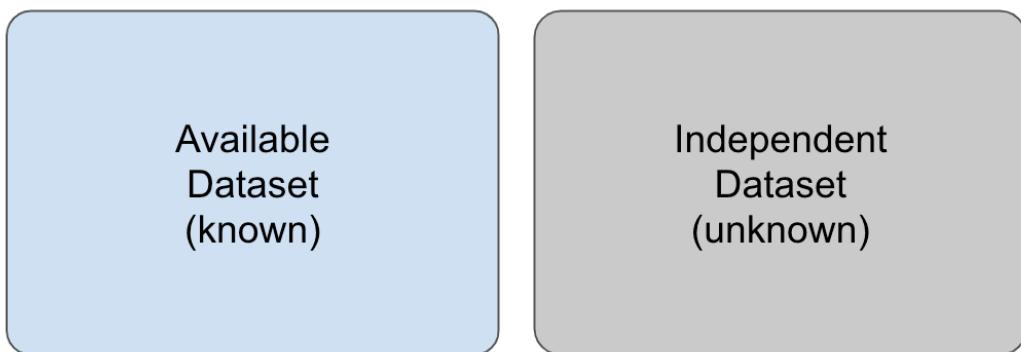


Figure 12.4: Illustration of an available set (in blue), whose outcome values are known and which is used to train a model, and an independent dataset (in grey), whose outcome values are unknown.

The first challenge arises when we do not know these independent datasets. However, in such cases, we still want to make sure that our model is not only working well when applied to the available dataset but can generalize well to independent unknown datasets. In particular, we want to make sure that we are not over-fitting to the dataset that the model was trained on.

How to minimize error on data we have never seen? We will make the following assumption:

We assume that the observations $\mathbf{x}_i, i = 1 \dots n$ of our dataset and of unseen data are i.i.d., meaning they are independent observations of the same population.

Under this assumption, one common strategy to evaluate the performance of the model on independent datasets is to select a subset of our dataset and pretend it is an independent dataset. As illustrated in Figure 12.5, we divide the available dataset into a training set and a test set. We will train our algorithm exclusively on the training set and use the test set only for evaluation purposes.



Figure 12.5: Separation of an available dataset into a training dataset (in green) and a test dataset (in orange) for building and evaluating a model.

Typically, we select a small piece of the dataset so that we have as much data as possible to train. However, we also want the test set to be large enough to obtain a stable estimate of the performance of the model on independent datasets. Common choices for the size of the train set are to use 10%-30% of the data for testing.

12.3.1 Over-fitting to the training dataset

Over-fitting can be detected when the measured error computed from a defined error function is notably larger for the test dataset than for the training dataset. Alternatively, one can use performance measurements such as precision or recall or visualize the performance of the model on the training vs. test dataset with ROC or precision-recall curves. Notable differences with the measurements or the plotted curves are signs of over-fitting.

In the previous example of the polynomial curve fitting, we can split the data consisting of $n = 10$ data points into train and test datasets and compare the computed error for different polynomial orders n . For $m = 9$, the computed root-mean-squared error for the test dataset notably increases while the error for the training dataset decreases.

12.3.2 Cross-validation

Cross-validation is often used as a strategy to assess the performance of a machine learning model that can help to prevent over-fitting.

In cross-validation the data is randomly split into a number k of folds (e.g. 3, 5 or 10), as illustrated in Figure 12.6.

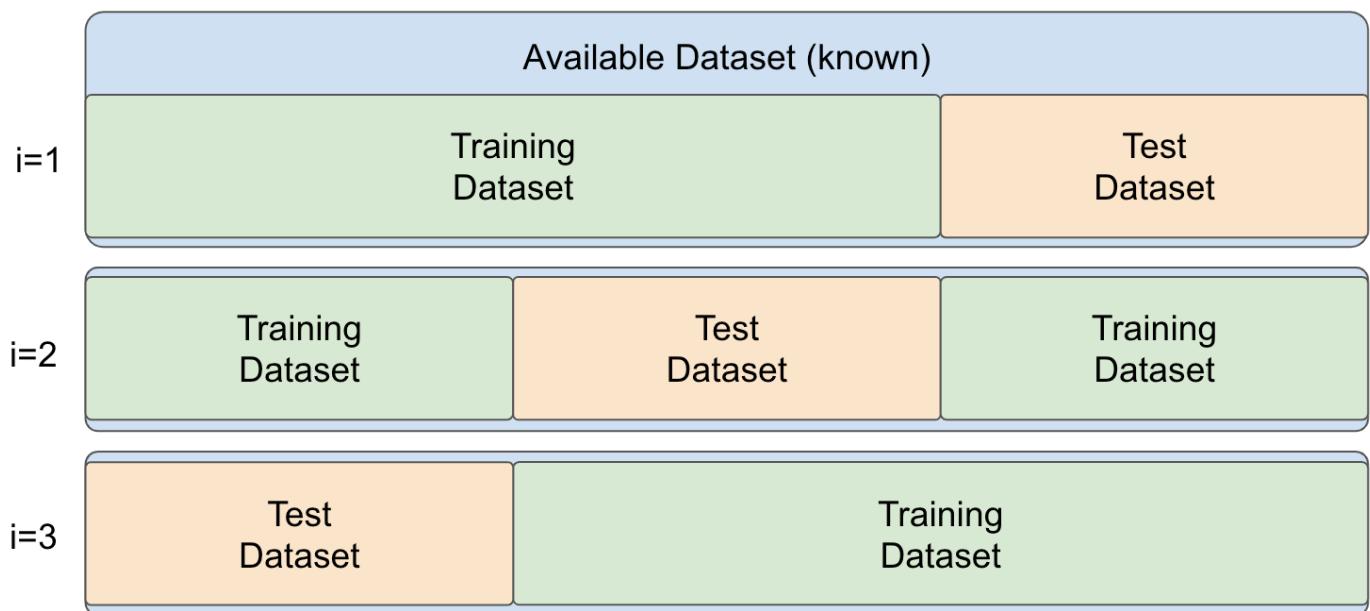


Figure 12.6: Schematic Illustration of cross-validation with $k = 3$ folds.

Then, a model is repeatedly trained on $k - 1$ folds and evaluated on the fold not used for training. In this manner, we train k models with a different training dataset on each fold i with $i \in \{1, \dots, k\}$. After the k models have been trained and evaluated on the respective left-out folds, evaluation metrics can be summarized into a combined evaluation metric.

Now, how do we pick k for the number of folds? Large values of k are preferable because the training data better imitates the original dataset. However, larger values of k will have much slower computation time: for example, 100-fold cross-validation will be 10 times slower than 10-fold cross-validation. For this reason, the choices of $k = 5$ and $k = 10$ are usually considered in practice.

12.3.2.1 Pitfalls of cross-validation

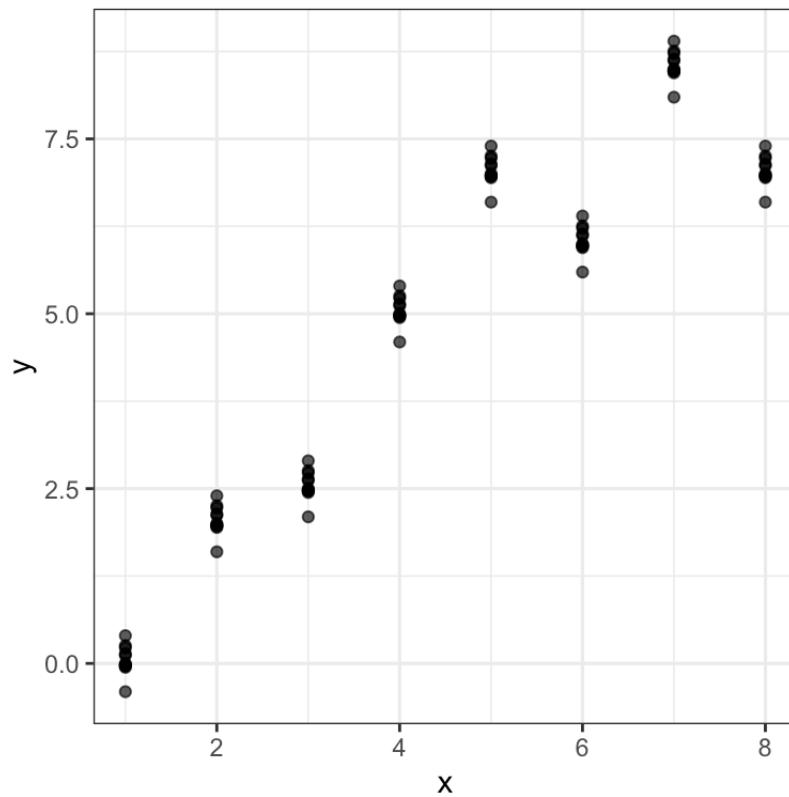
For considering cross-validation one fundamental assumption has to be met, namely that the training samples and the test samples are independently and identically distributed (i.i.d.). As a trivial example, we would consider a non identical distribution if our training set consisted of red apples and green pears, but our test set also contained green apples. The issue of having non independent distributions may arise if we have data coming in clusters that we are not aware of (e.g: repeated measures, people from same families, homologous genes).

As an example of this issue, we consider a simulated dataset `simulated_dt` consisting of a feature variable `x` and an outcome variable `y` :

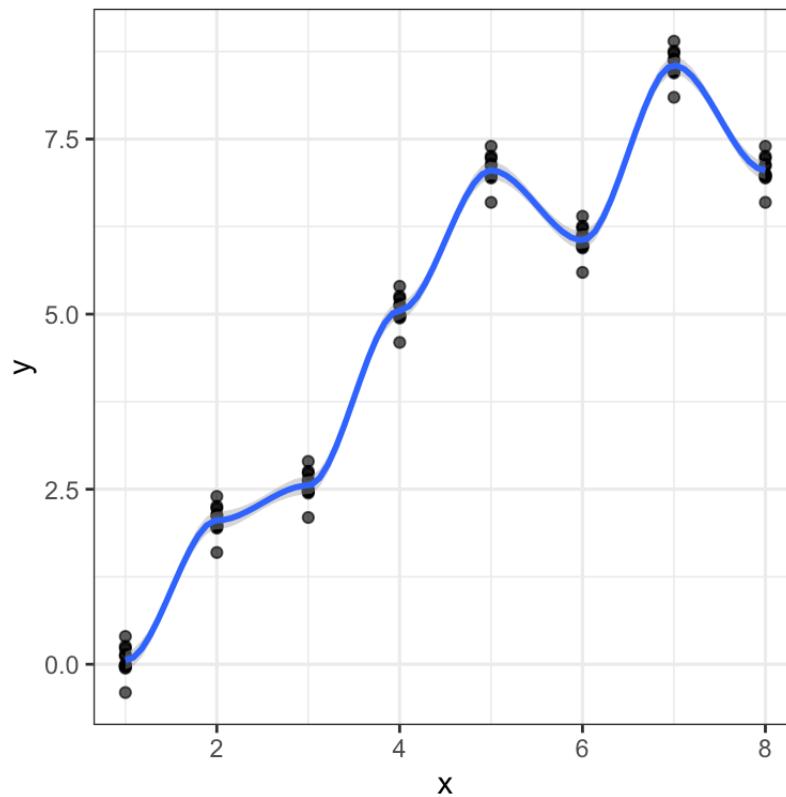
```
head(simulated_dt, n=10)
```

```
##      x          y
## 1: 1  0.00000000
## 2: 1 -0.03377742
## 3: 1  0.25579509
## 4: 1  0.22420401
## 5: 1  0.11684048
## 6: 1 -0.05580587
## 7: 1  0.39764426
## 8: 1  0.13625628
## 9: 1 -0.02500456
## 10: 1 -0.02142251
```

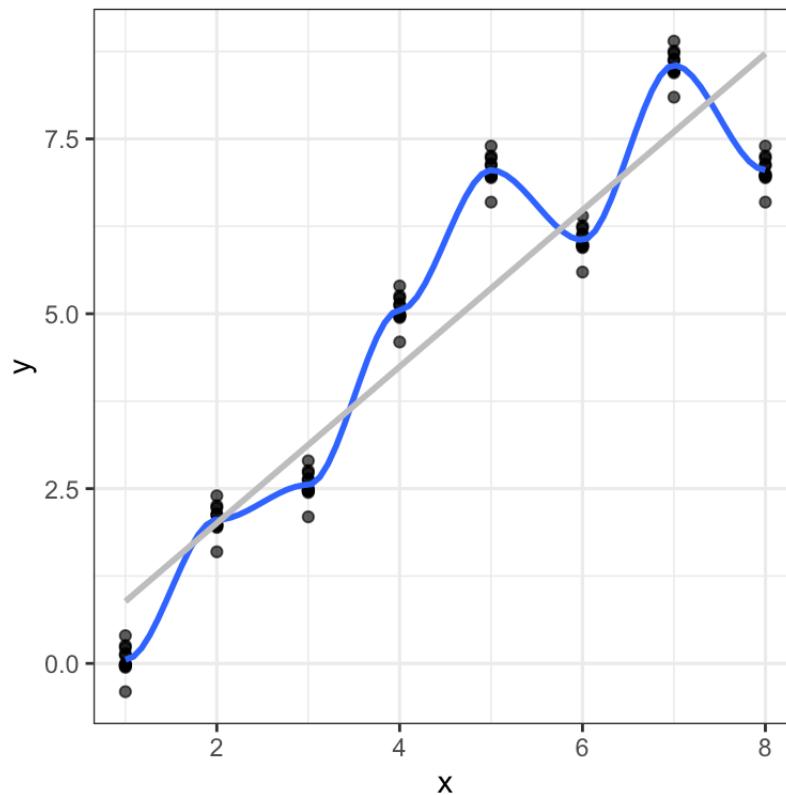
We observe that the dataset contains repeated measures and that the between replicate simulated trend is $x=y$



Performing cross-validation at the level of individual data points will favor models that learn the clusters:



We need to perform cross-validation at the cluster level to learn the trend across clusters. However, this is difficult without application knowledge. Visualization techniques can help. Application areas may have their own strategies. In genetics, models are required to be replicated on independent populations (genetic associations on say Swedes must be replicated among say Germans). The issue also arises with temporal data. Cross-validation cannot be done by taking random subsets of past data points. For temporal data, it is common practice (and common sense) to test models by mimicking predictions of future time points.



12.3.2.2 Cross-validation in R

In R, one approach to implement a cross validated model is using the `caret` package. The `caret` package consolidates different machine learning algorithms and tools from different authors into one library. It conveniently provides a function that performs cross-validation.

As an example, we implement a logistic regression model to solve the binary classification task from Chapter 11, which consists in predicting the gender of a person based on height values of him/herself and his/her parents. First, we load the heights dataset:

```
heights_dt <- fread("extdata/height.csv") %>% na.omit() %>%
  .[, sex:=as.factor(toupper(sex))]
heights_dt
```

Then, we define the validation specification with the help of the function `trainControl()`. Here, we set the method as `cv` for cross-validation with `k=5` folds as the `number` argument. Additionally, we define `twoClassSummary` as the summary function for computing the sensitivity, specificity and the area under the ROC curve for performance measurement in binary classification tasks.

```
# install.packages("caret")
library(caret)

# generate control structure
k <- 5      # number of folds
fitControl <- trainControl(method = "cv", # cv for cross-validation
                           number = k,
                           classProbs=TRUE, # compute class probabilities
                           summaryFunction = twoClassSummary
                           )
```

Next, we can train the model with the `train()` function. Here, we set the previously defined `fitControl` as the training control for implementing cross-validation:

```
# run CV

lr_fit <- train(## formula and dataset definition
  sex~.,
  data = heights_dt,
  ## model specification
  method = "glm",
  family = "binomial",
  ## validation specification
  trControl = fitControl,
  ## Specify which metric to optimize
  metric = "ROC",
  )

lr_fit

## Generalized Linear Model
##
## 253 samples
##   3 predictor
##   2 classes: 'F', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 203, 202, 202, 203, 202
## Resampling results:
##
##   ROC      Sens      Spec
##   0.9811125 0.911  0.9310541
```

As we can observe, the `lr_fit` object returns the average values for the computed values of sensitivity, specificity and the area under the ROC curve of the `k=5` trained models. The performance measurements for each model can be obtained as follows:

```
lr_fit$resample

##          ROC      Sens      Spec Resample
## 1 0.9967949 1.0000000 0.9615385    Fold1
## 2 0.9830769 0.9200000 0.9230769    Fold2
## 3 0.9953704 0.9583333 0.9629630    Fold3
## 4 0.9887821 0.9166667 0.9615385    Fold4
## 5 0.9415385 0.7600000 0.8461538    Fold5
```

Additionally, the final model can be obtained by accessing the attribute `finalModel` from the `lr_fit` object:

```

lr_fit$finalModel

##
## Call:  NULL
##
## Coefficients:
## (Intercept)    height     mother     father
## -37.4913      0.7307    -0.2899    -0.2360
##
## Degrees of Freedom: 252 Total (i.e. Null);  249 Residual
## Null Deviance:      350.4
## Residual Deviance:  83.05   AIC: 91.05

```

12.4 Random Forests as alternative models

There is a plethora of supervised learning algorithms out there. Providing a reasonable survey of them is out of the scope of this module. We therefore introduce here a single supervised learning method beyond linear and logistic regression which is working well in many situations: Random forests. Just as linear and logistic regression, should be the first baseline models to try, random forests can be seen as the first non-interpretable supervised learning model to give a shot for. From this basis, many other options can be considered, often depending on the application areas.

Random forests can be applied to both regression and classification tasks in supervised learning. In this section, we introduce random forests as a further type of model for solving both tasks, as these are easy to understand and achieve state-of-the-art performance in many prediction tasks.

Random forests are based on decision trees. More precisely, they are an instance of tree-based ensemble learning, which is a strategy robust to over-fitting and allows fitting very flexible functions. Each random forest is an ensemble of several decision trees, which are machine learning models for both classification and regression tasks and will be introduced in the following section.

12.4.1 The basics of decision trees

A decision tree is a method that involves partitioning or segmenting the training data set into a number of simple regions. Typically, once the input space of the training dataset has been segmented, we can make a prediction for a new input sample by using the mean or the mode of the training observations in the region to which the new input sample is assigned to.

12.4.1.1 Decision trees for regression tasks

As an example for a decision tree model for a regression task, we consider the `Hitters` dataset⁴⁹ to predict a baseball player's (log-transformed) `Salary` based on the feature `Years` (the number of years that he has played in the major leagues) and on the feature `Hits` (the number of hits that he made in the previous year). A regression tree for this dataset is shown in Figure 12.7. The constructed tree consists of 3 splitting rules. First, observations having `Years < 4.5`

get passed to the right branch and observations having $\text{Years} \geq 4.5$ to the left branch. The predicted log-transformed salary for these players in the right is given by the mean log-transformed salary of all training samples in this node, which is in this case 5.11. Players assigned to the left branch are further subdivided by the feature Hits.

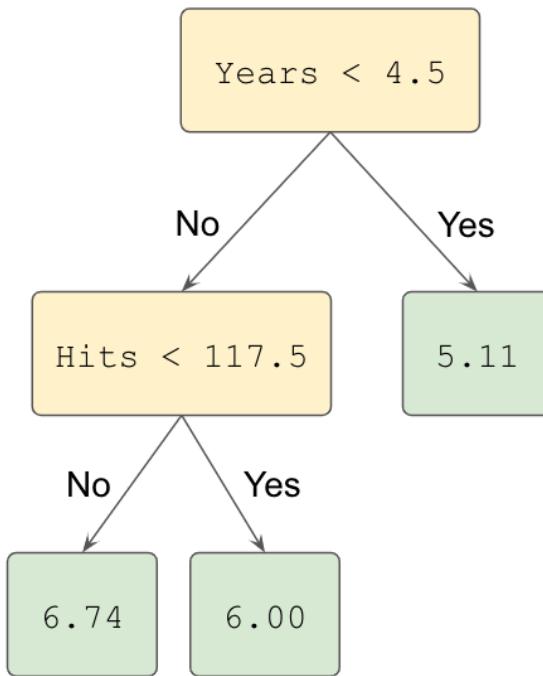


Figure 12.7: Illustration of a decision tree for predicting the logarithmic salary of a baseball player based on the features Years and Hits.

After the training is finalized, the tree regions the players into three regions: players who have played for four or fewer years, players who have played for five or more years and who made fewer than 118 hits last year, and players who have played for five or more years and who made at least 118 hits last year. A graphical representation of this resulting segmentation can be observed in Figure 12.8.

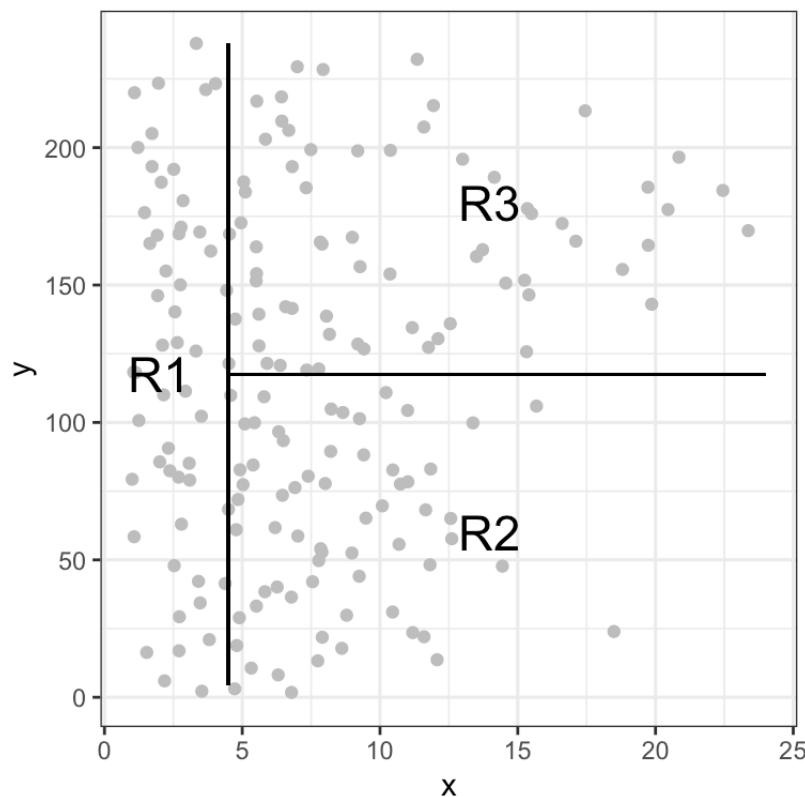


Figure 12.8: Illustration of the resulting three regions from the trained decision tree based on Hits and Years.

Formally, the three resulting regions can be written as:

$$\begin{aligned} R_1 &= \{X \mid \text{Years} < 4.5\} \\ R_2 &= \{X \mid \text{Years} \geq 4.5, \text{Hits} < 117.5\} \\ R_3 &= \{X \mid \text{Years} \geq 4.5, \text{Hits} \geq 117.5\} \end{aligned}$$

Here, X is the dataset consisting of all training samples and the regions R_1 , R_2 and R_3 are the leaf nodes of the tree. All other nodes are denoted as internal nodes of the constructed decision tree.

Decision trees are easy to interpret. In our example, we can interpret the tree as follows:

- `Years` is the most important feature for predicting the log-transformed salary of a player. Players with less experience earn lower salaries than more experienced players.
- If a player is more experienced, the number of hits that he made in the previous year is important for determining his salary.

Now... why and how did we choose the feature `Years` and the threshold of `4.5` for the top split? Overall, the goal of a decision tree for regression tasks is to find the optional regions R_1, R_2, \dots, R_J that minimize the residual residual sum of squares (RSS) given by:

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where \hat{y}_{R_j} is the prediction for all samples in region R_j , which is computed from the mean of the outcome value for the training samples in region R_j .⁵⁰

Unfortunately, it is computationally unfeasible to consider every possible partition of the training dataset. This is why decision trees follow a top-down greedy approach to build the tree. The approach is top-down because it begins at the top of the tree with the complete dataset and then successively splits the dataset. Here, each split results into two new branches further down on the tree. Note that the approach is greedy because at each step of the process, the best split is made at that particular step.

In order to perform recursive binary splitting, we first select the feature X_j and a threshold s such that splitting the samples into the regions $R_1(j, s) = X|X_j < s$ and $R_2(j, s) = X|X_j \geq s$ results into the greatest possible reduction in RSS. For this, we consider all possible features X_1, \dots, X_p , and all possible values threshold values s for each feature. In greater detail, for any j and s , we want to minimize the equation:

$$\sum_{i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2,$$

Note that finding optimal values for thresholds and features can be achieved quickly with the current advances in computer science assuming that the number of features is not too large.

The iterative process of finding features and thresholds that define splitting rules continues until a stopping criterion is met. This prevents to consider every possible partition of the training dataset. Stopping criteria include setting a minimum number of samples in a region or defining a maximum number of resulting regions.

Once the regions R_1, \dots, R_J have been created, we can predict the outcome of any given sample (from the test set) by assigning this sample to a region and returning the mean of the training observations in the region to which that (test) sample belongs.

12.4.1.2 Decision trees for classification tasks

As stated before, decision trees can be applied for both regression and classification tasks. The construction process of classification decision trees is very similar to the one that we previously described for regression tasks. One difference is that for a regression tree, the predicted outcome for an observation is given by the mean response of the training observations that belong to the same leaf node. In contrast, for a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

Additionally, in contrast to regression scenarios, RSS cannot be used as a criterion for making the tree splits for classification tasks. Typically, cross-entropy is used to fit the model (See Section 11.2).

As an example we consider the binary classification task of deciding whether a person should play tennis on a given day or not. We consider the simulated data for training a model. The data contains four feature variables ("Outlook" , "Temperature" , "Humidity" and "Windy") and the outcome variable "Play.Tennis" .

```
play_tennis_dt
```

```
##      Day Outlook Temperature Humidity Windy Play.Tennis
## 1:   1  Sunny        15.5  Normal   Yes     No
## 2:   2  Rainy        24.5    High   Yes     No
## 3:   3  Rainy        24.0  Normal   No    Yes
## 4:   4  Rainy        15.0  Normal   Yes    Yes
## 5:   5  Rainy        27.0  Normal   Yes    Yes
## 6:   6  Rainy        27.5  Normal   No    Yes
## 7:   7  Sunny        16.0    High   No    Yes
## 8:   8  Sunny        29.5  Normal   Yes    Yes
## 9:   9  Sunny        17.0    High   No    Yes
## 10: 10  Rainy        20.5  Normal   Yes    Yes
```

Figure 12.9 shows an example of a decision tree for solving the described classification task. The initial splitting rule tests whether the value of the `Outlook` feature of each sample is either sunny or rainy. Samples with the feature value being sunny are passed to the right node. If the outlook is rainy for a subset of the initial samples in the dataset, these samples are passed to the left node. Here, a second feature test is performed. This test involves checking the humidity feature for all samples. If the humidity value is normal, a third splitting rule is defined for checking whether the day is windy or not. In total, four regions results from the training process of the decision tree.

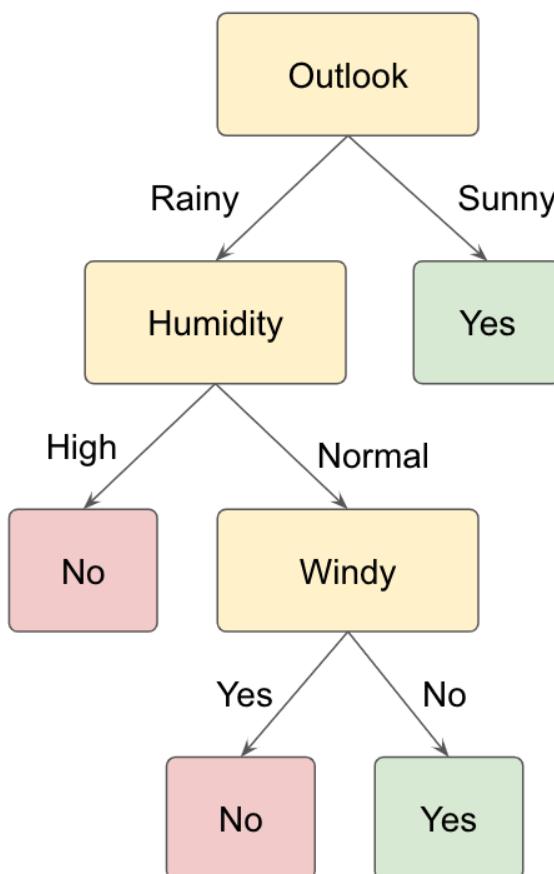


Figure 12.9: Example of a decision tree for the binary classification task of deciding whether to play tennis or not based on the features Outlook, Humidity and Windy.

Note that decision trees for classification tasks can be constructed with both numerical and categorical features, even though our tennis example only contains categorical data. Feature tests for numerical features can be formulated with an (un)equality for a certain feature, as described before for regression trees. For instance, a node in a decision tree could test whether the temperature feature is higher than 20 degrees Celsius (`Temperature > 20`).

12.4.2 Random Forests for classification and regression tasks

For constructing a random forest, the first step is called aggregation (or bagging), which involves generating many predictors, each using regression or classification trees. To ensure that the individual trees are not the same, we use bootstrap to induce randomness. These two features combined explain the name: the bootstrap makes the individual trees randomly different, and the combination of trees is the forest. In a random forest for classification tasks, the predicted output is the majority vote of the contained decision trees. In contrast, the predicted value of a random forest regressor is the mean of the predictions from each decision tree in the random forest.

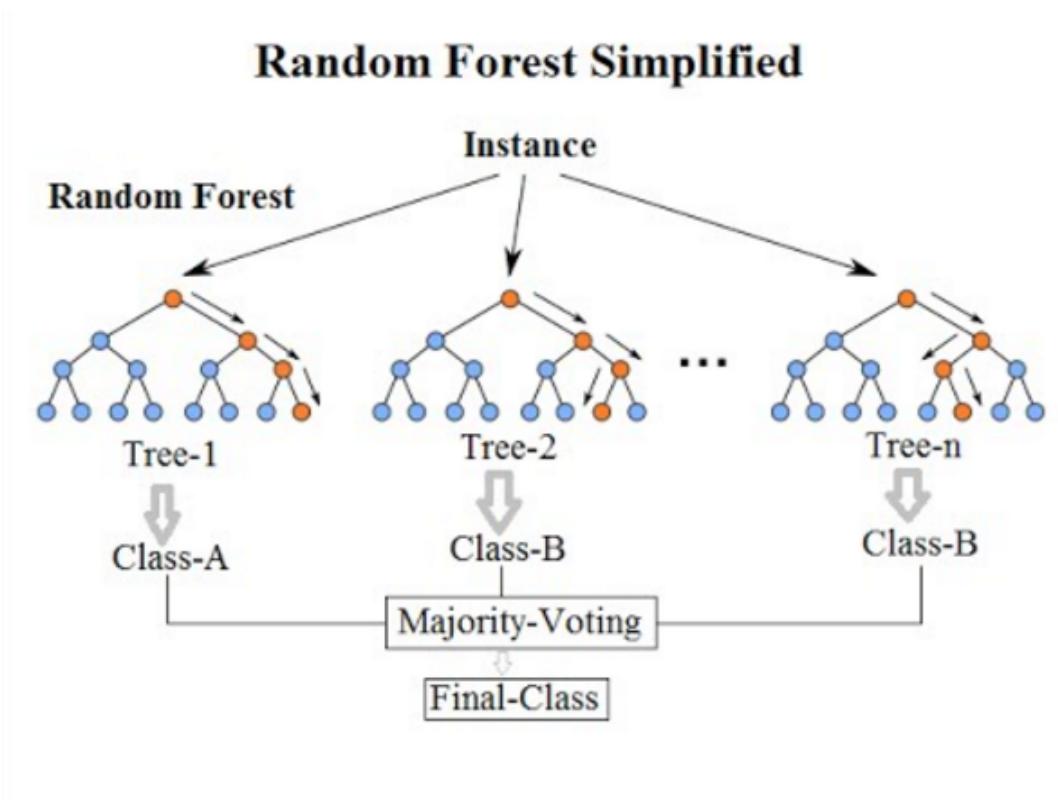


Figure 12.10: Illustration of a random forest for classification tasks. Source:

https://en.wikipedia.org/wiki/Random_forest#/media/File:Random_forest_diagram_complete.png

The specific steps for building a random forest can be formulated as follows:

1. Build B decision trees using the training set. We refer to the fitted models as T_1, T_2, \dots, T_B .
2. For every observation in the test set, form a prediction \hat{y}_j using tree T_j
3. For continuous outcomes, form a final prediction with the average $\hat{y} = \frac{1}{B} \sum_{j=1}^B \hat{y}_j$. For categorical outcomes, predict \hat{y} with majority vote (most frequent class among $\hat{y}_1, \dots, \hat{y}_B$).

Now... how do we assure that the decision trees are different in the random forest, even if they are trained on the same training dataset? For this, we use randomness in two ways. Let N be the number of observations in the training set. To create the trees T_1, T_2, \dots, T_B from the training set we do the following with the feature and sample selection:

1. Create a bootstrap training set by sampling N observations from the training set with replacement. In this manner, each tree is trained on a different dataset.
2. Randomly select a subset of the features to be included in the building of each tree so that not all features are considered for each tree. A different random subset is selected for each tree. This reduces correlation between trees in the forest and prevents over-fitting.

12.4.3 Random Forests in R

We use function `randomForest` from the package `randomForest` for training random forests in R.

```
# install.packages("randomForest")
library(randomForest)
```

As an example, we train a random forest classifier for predicting the gender of each person in the `heights_dt` data table given the height values of the person and the person's parents as before. We define the values for a few hyper parameters such as the number of trees in the random forest (`ntree`), the minimum number of samples in a leaf node (`nodesize`) and the maximum number of leaf nodes in a tree (`maxnodes`).

```
rf_classifier <- randomForest(## Define formula and data
                                sex~.,
                                data=heights_dt,
## Hyper parameters
                                ntree=100,      # Define number of trees
                                nodesize = 5,   # Minimum size of leaf nodes
                                maxnodes = 30,  # Maximum number of leaf nodes
## Output the feature importances
                                importance=TRUE)

rf_classifier
```

```

## 
## Call:
##   randomForest(formula = sex ~ ., data = heights_dt, ntree = 100,      nodesize = 5, maxnodes = 30, impo
##                 Type of random forest: classification
##                 Number of trees: 100
## No. of variables tried at each split: 1
##
##          OOB estimate of  error rate: 12.65%
## Confusion matrix:
##   F   M class.error
## F 106 16  0.1311475
## M  16 115  0.1221374

```

We remark that the number of trees (`ntree`), the minimum size of leaf nodes (`nodesize`) and the maximum amount of leaf nodes (`maxnodes`) in each tree are hyper parameters of random forests. These should be optimized (or tuned) for each application to achieve a better performance of the models. However, hyper-parameter tuning is out scope in this lecture.

As stated before, tree-based models are widely used in practice partly because of their high interpretability. The random forest classifier in R offers a closer look at the feature importances of the model. We can observe in this example that the height of the person is the most important feature for predicting his/her gender for both females and males. Additionally, the mother's height seems to be more important for predicting the female gender than the father's height and the father's height seems to be more determinant for predicting the gender male than the mother's height.

```
rf_classifier$importance
```

	F	M	MeanDecreaseAccuracy
## height	0.30866100	0.32522524	0.31588077
## mother	0.01286732	0.01533178	0.01439007
## father	0.02484089	0.04097106	0.03349557
##			MeanDecreaseGini
## height		77.20067	
## mother		12.86759	
## father		16.67947	

For predicting the gender of an input sample using the trained model stored in the variable `rf_classifier`, we can use the function `predict` as follows:

```

heights_dt[, sex_predicted:= predict(rf_classifier,
                                     heights_dt[,-c("sex")])]

heights_dt

```

```

##      height sex mother father sex_predicted
## 1:    197   M     175    191           M
## 2:    196   M     163    192           M
## 3:    195   M     171    185           M
## 4:    195   M     168    191           M
## 5:    194   M     164    189           M
##   ---
## 249:   152   F     150    165           F
## 250:   172   F     165    188           F
## 251:   154   F     155    165           F
## 252:   178   M     169    174           M
## 253:   175   F     171    189           F

```

12.5 Conclusion

This chapter introduced concepts from supervised machine learning for prediction tasks. Here, the main focus is to build and evaluate models that identify trends in the training dataset but are still able to generalize to (new) independent datasets. We described cross-validation as a strategy to prevent over-fitting.

Additionally, we introduced random forests as alternative machine learning models to linear and logistic regression for regression and classification tasks, since they are easy to understand and interpret and still show good performance in many application tasks. However, there are many more algorithms for solving regression and/or classification tasks in supervised learning. A few of the most popular methods include support vector machines (SVMs), (Deep) Neural Networks, k-Nearest Neighbors and Naive Bayes classifiers.

12.6 Resources

- Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.
- Rafael A. Irizarry. 2019. Introduction to Data Science. Data Analysis and Prediction Algorithms with R .
- Gareth James, Daniela Witten, Trevor Hastie & Robert Tibshirani. 2013. An Introduction to Statistical Learning with Applications in R.
- The caret package manual: <https://topepo.github.io/caret/>
- The Datacamp machine learning courses: <https://www.datacamp.com/courses/machine-learning-toolbox>

48. https://en.wikipedia.org/wiki/Machine_learning ↪

49. <https://rdrr.io/cran/ISLR/man/Hitters.html> ↪

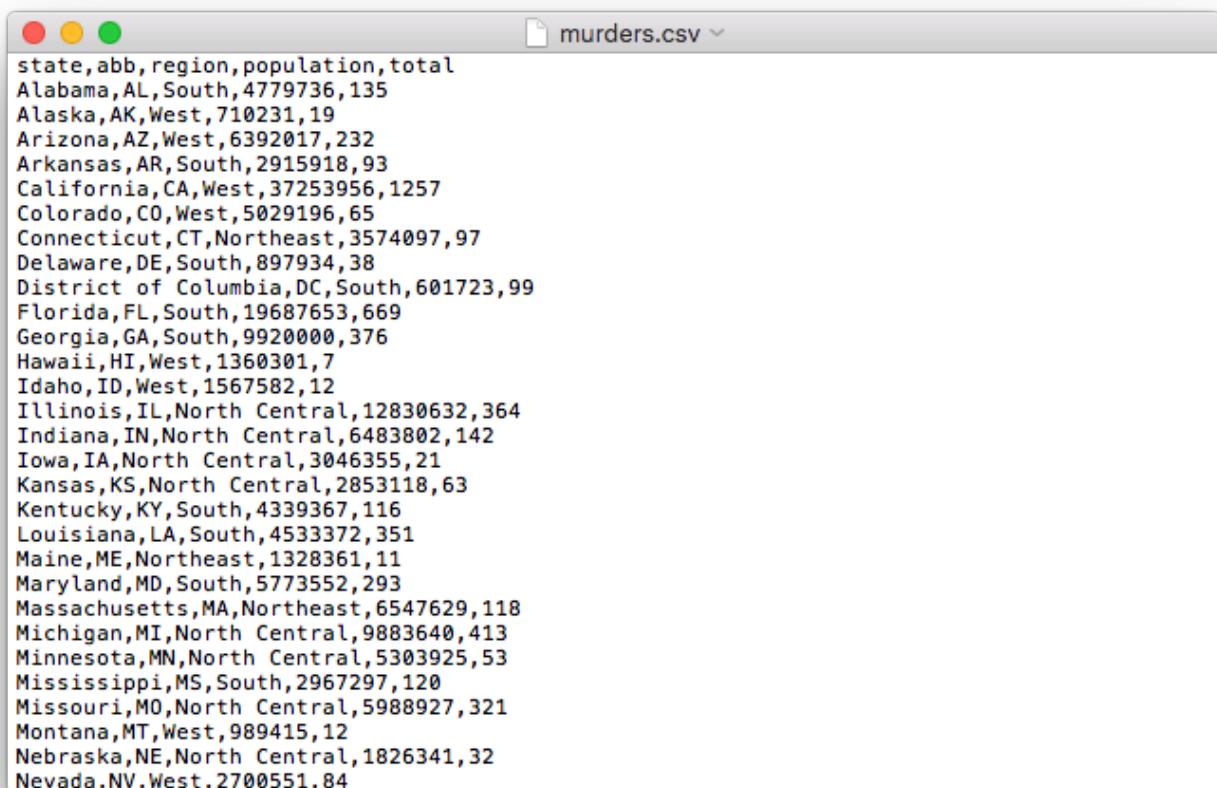
50. Show that sample mean minimizes the least square errors to a set of observations. ↪

A Importing data

This chapter is largely adopted from “Introduction to Data Science” by Rafael A. Irizarry (<https://rafalab.github.io/dsbook/>) and covers the reading of files.

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

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



The screenshot shows a text editor window with a light gray background. At the top, there are three colored window control buttons (red, yellow, green) on the left and a title bar on the right that reads "murders.csv". The main area of the window contains a list of data rows, each consisting of five fields separated by commas. The fields are: state, abb, region, population, and total. The data includes entries for various US states and their respective statistics. The text is black and clearly legible against the white background of the code editor.

state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	South	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21
Kansas	KS	North Central	2853118	63
Kentucky	KY	South	4339367	116
Louisiana	LA	South	4533372	351
Maine	ME	Northeast	1328361	11
Maryland	MD	South	5773552	293
Massachusetts	MA	Northeast	6547629	118
Michigan	MI	North Central	9883640	413
Minnesota	MN	North Central	5303925	53
Mississippi	MS	South	2967297	120
Missouri	MO	North Central	5988927	321
Montana	MT	West	989415	12
Nebraska	NE	North Central	1826341	32
Nevada	NV	West	2700551	84

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

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

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

A.1 Paths and the working directory

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

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

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

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

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

```
library(data.table)
dt <- fread(filename)
```

Here the data is read into a `data.table` called `dt`.

Currently `fread` is the fastest option and suitable for large data sets. However, there are alternative options like the `read_csv` function from the `readr` package, which is part of the tidyverse.

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

The data is imported and stored in `dat`. The rest of this section defines some important concepts and provides an overview of how we write code that tells R how to find the files we want to import. Rafael A. Irizarry's Chapter on Productivity Tools covers this in depth.

A.1.1 The filesystem

You can think of your computer's filesystem as a series of nested folders, each containing other folders and files. Data scientists refer to folders as *directories*. We refer to the folder that contains all other folders as the *root directory*. We refer to the directory in which we are currently located as the *working directory*. The working directory therefore changes as you move through folders: think of it as your current location.

A.1.2 Relative and full paths

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

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

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

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

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

## [1] "data"        "DESCRIPTION"  "extdata"     "help"
## [5] "html"        "INDEX"       "Meta"        "NAMESPACE"
## [9] "R"           "script"
```

These relative paths give us the location of the files or directories if we start in the directory with the full path. For example, the full path to the `help` directory in the example above is

`/Library/Frameworks/R.framework/Versions/3.5/Resources/library/dslabs/help`.

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

A.1.3 The working directory

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

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

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

A.1.4 Generating path names

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

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

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

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

## [1] TRUE
```

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

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

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

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

A.1.5 Copying files using paths

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

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

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

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

```
list.files()
```

A.2 The `readr` and `readxl` packages

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

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

A.2.1 `readr`

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

```
library(readr)
```

The following functions are available to read-in spreadsheets:

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

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

```
read_lines("murders.csv", n_max = 3)

## [1] "state,abb,region,population,total"
## [2] "Alabama,AL,South,4779736,135"
## [3] "Alaska,AK,West,710231,19"
```

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

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

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

```
View(dat)
```

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

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

A.2.2 readxl

You can load the `readxl` package using

```
library(readxl)
```

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

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

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

A.3 Exercises

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

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

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

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

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

```
names(dat)
```

to see that the names are not informative.

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

A.4 Downloading files

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

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

The `read_csv` file can read these files directly:

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

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

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

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

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

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

A.5 R-base importing functions

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

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

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

```
class(dat2$abb)
```

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

```
class(dat2$region)
```

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

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

```
dat <- read.csv("murders.csv", stringsAsFactors = FALSE)
class(dat$state)

## [1] "character"
```

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

scan

When reading in spreadsheets many things can go wrong. The file might have a multiline header, be missing cells, or it might use an unexpected encoding⁵¹. We recommend you read this post about common issues found here: <https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/>.

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

```
path <- system.file("extdata", package = "dslabs")
filename <- "murders.csv"
x <- scan(file.path(path, filename), sep=",", what = "c")
x[1:10]

## [1] "state"      "abb"        "region"     "population"
## [5] "total"       "Alabama"    "AL"         "South"
## [9] "4779736"    "135"
```

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

A.6 Text versus binary files

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

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

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

A.7 Unicode versus ASCII

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

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

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

A.8 Organizing data with spreadsheets

Although this book focuses almost exclusively on data analysis, data management is also an important part of data science. As explained in the introduction, we do not cover this topic. However, quite often data analysts need to collect data, or work with others collecting data, in a way that is most conveniently stored in a spreadsheet. Although filling out a spreadsheet by hand is a practice we highly discourage, we instead recommend the process be automated as much as possible, sometimes you just have to do it. Therefore, in this section, we provide recommendations on how to organize data in a spreadsheet. Although there are R packages designed to read Microsoft Excel spreadsheets, we generally want to avoid this format. Instead, we recommend Google Sheets as a free software tool. Below we summarize the recommendations made in paper by Karl Broman and Kara Woo⁵². Please read the paper for important details.

- **Be Consistent** - Before you commence entering data, have a plan. Once you have a plan, be consistent and stick to it.
- **Choose Good Names for Things** - You want the names you pick for objects, files, and directories to be memorable, easy to spell, and descriptive. This is actually a hard balance to achieve and it does require time and thought. One important rule to follow is **do not use spaces**, use underscores _ or dashes instead – . Also, avoid symbols; stick to letters and numbers.

- **Write Dates as YYYY-MM-DD** - To avoid confusion, we strongly recommend using this global ISO 8601 standard.
- **No Empty Cells** - Fill in all cells and use some common code for missing data.
- **Put Just One Thing in a Cell** - It is better to add columns to store the extra information rather than having more than one piece of information in one cell.
- **Make It a Rectangle** - The spreadsheet should be a rectangle.
- **Create a Data Dictionary** - If you need to explain things, such as what the columns are or what the labels used for categorical variables are, do this in a separate file.
- **No Calculations in the Raw Data Files** - Excel permits you to perform calculations. Do not make this part of your spreadsheet. Code for calculations should be in a script.
- **Do Not Use Font Color or Highlighting as Data** - Most import functions are not able to import this information. Encode this information as a variable instead.
- **Make Backups** - Make regular backups of your data.
- **Use Data Validation to Avoid Errors** - Leverage the tools in your spreadsheet software so that the process is as error-free and repetitive-stress-injury-free as possible.
- **Save the Data as Text Files** - Save files for sharing in comma or tab delimited format.

51. https://en.wikipedia.org/wiki/Character_encoding ↵

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

B R programming

This Appendix covers the basic concepts of the R programming language: data types, data operations, data structures, control flow and functions. It is largely based on Rafael Irizzary's book Introduction to Data Science [<https://rafalab.github.io/dsbook/>].

B.1 Conditional expressions

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

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

```
a <- 0

if (a != 0) {
  print(1/a)
} else {
  print("No reciprocal for 0.")
}

## [1] "No reciprocal for 0."
```

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

```
library(dslabs)
data(murders)
murder_rate <- murders$total/murders$population * 1e+05
```

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

```

ind <- which.min(murder_rate)

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

## [1] "Vermont"

```

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

```

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

## [1] "No state has a murder rate that low."

```

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

```

a <- 0
ifelse(a > 0, 1/a, NA)

## [1] NA

```

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

```

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

```

This table helps us see what happened:

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

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

```
data(na_example)
no_nas <- ifelse(is.na(na_example), 0, na_example)
sum(is.na(no_nas))

## [1] 0
```

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

```
z <- c(TRUE, TRUE, FALSE)
any(z)
```

```
## [1] TRUE
```

```
all(z)
```

```
## [1] FALSE
```

B.2 Defining functions

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

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

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

```
x <- 1:100
identical(mean(x), avg(x))
```

```
## [1] TRUE
```

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

```
s <- 3
avg(1:10)
```

```
## [1] 5.5
```

```
s
```

```
## [1] 3
```

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

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

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

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

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

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

B.3 Namespaces

Once you start becoming more of an R expert user, you will likely need to load several add-on packages for some of your analysis. Once you start doing this, it is likely that two packages use the same name for two different functions. And often these functions do completely different things. Functions of different packages live in different *namespaces*. R will follow a certain order when searching for a function in these *namespaces*. You can see the order by typing:

```
search()
```

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

If we want to be absolutely sure that R uses the function of specific package, we shall use double colons (`::`). For instance to force R to use the `filter` of the **stats** package, we can use

```
stats::filter
```

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

For more on this more advanced topic we recommend the R packages book⁵³.

B.4 For-loops

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

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

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

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

```
for (i in 1:5) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

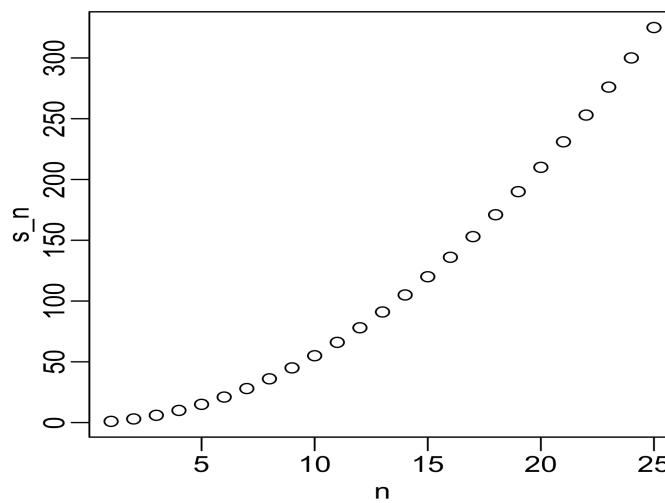
Here is the for-loop we would write for our S_n example:

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

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

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

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



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

B.5 Vectorization and functionals

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

```
x <- 1:10
sqrt(x)

## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
## [9] 3.000000 3.162278

y <- 1:10
x * y

## [1] 1 4 9 16 25 36 49 64 81 100
```

To make this calculation, there is no need for for-loops. However, not all functions work this way. For instance, the function we just wrote, `compute_s_n`, does not work element-wise since it is expecting a scalar. This piece of code does not run the function on each entry of `n`:

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

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

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

```
x <- 1:10
sapply(x, sqrt)

## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
## [9] 3.000000 3.162278
```

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

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

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

B.6 R Markdown

- This is an R Markdown presentation. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
- Simply go to File → New File → R Markdown
- Select PDF and you get a template.
- You most likely won't need more commands than in on the first page of [this cheat sheet](#).
- When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

B.7 Resources

- Advanced R by Hadley Wickham [Advanced R](#)
- In-depth documentations:
 - [Introduction to R](#)
 - [R language definition](#)
 - [R Internals](#)
- Last but not least:
 - [Stackoverflow](#)

53. <http://r-pkgs.had.co.nz/namespace.html> ↵

C Additional plotting tools

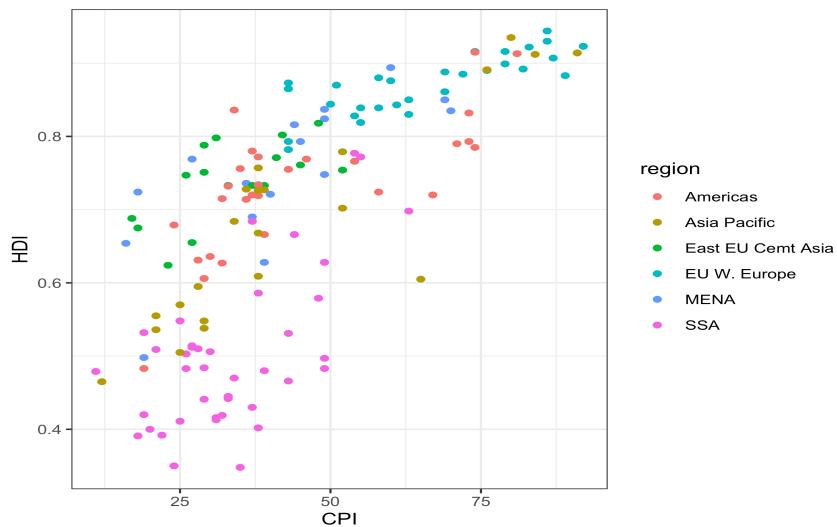
C.1 Plotting themes

Themes control non-data parts of your plots, such as:

- Overall appearance
- Axes
- Plot title
- Legends

They control the appearance of your plots (size, color, position) but not how the data is represented. One example is `theme_bw` which provides a white background.

```
ggplot(ind, aes(CPI, HDI, color = region)) + geom_point() + theme_bw()
```



Other complete themes are: `theme_classic` , `theme_minimal` , `theme_light` , `theme_dark` .

More themes are available in the `ggthemes` package.

```
library(ggthemes)
ggplot(ind, aes(CPI, HDI, color = region)) + geom_point() + theme_wsj() + scale_colour_wsj("colors6",
 "")
```

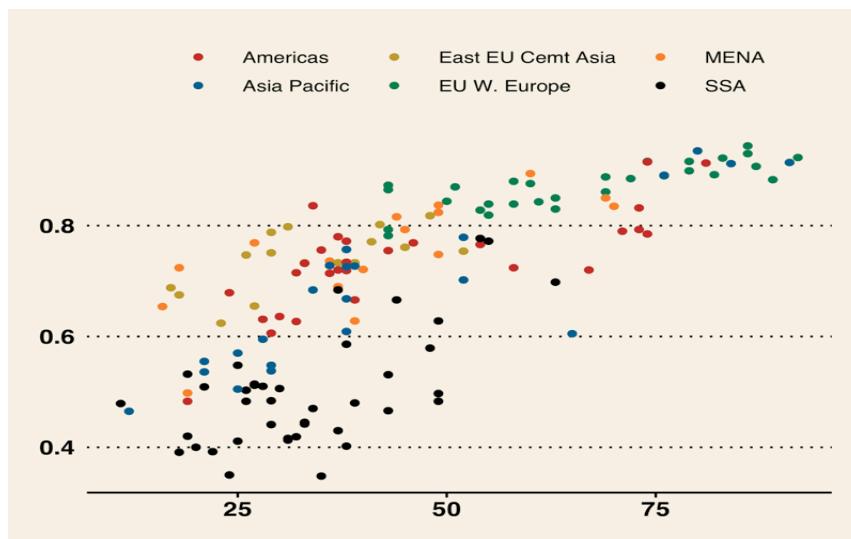
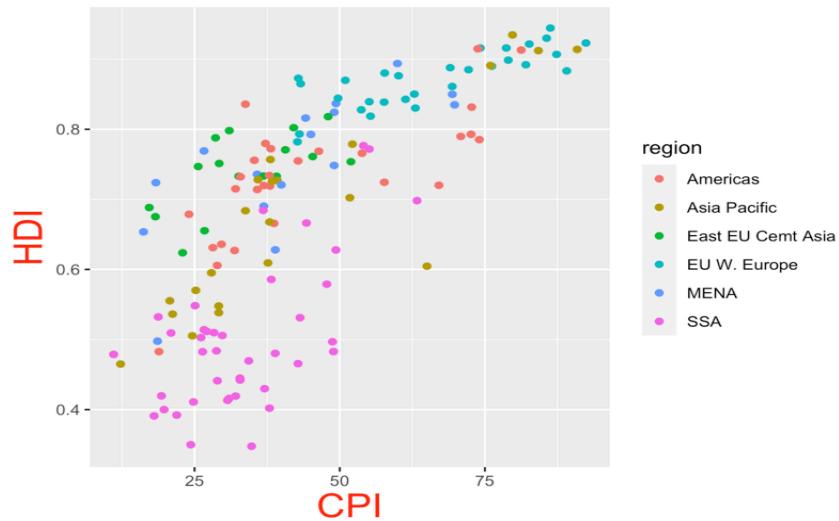


Figure C.1: Example of Wall Street Journal theme

C.2 Axes

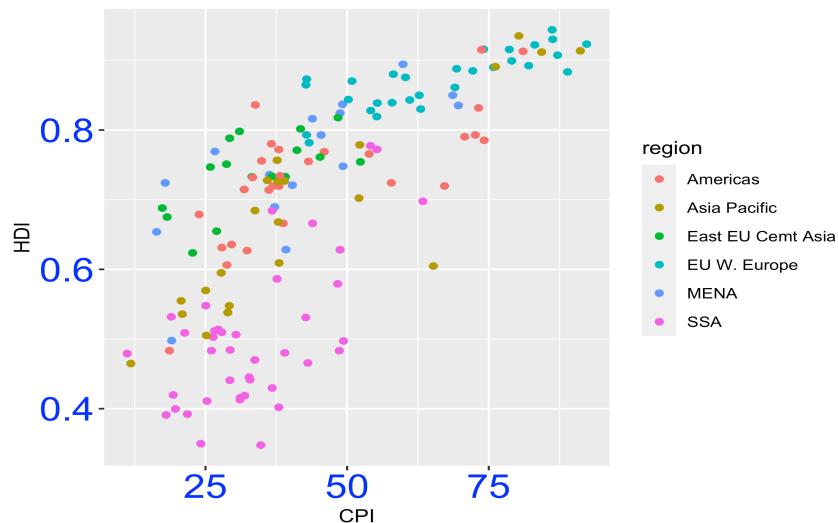
The appearance of the axis titles can be controlled with the global variable `axis.title`, or independently for *x* and *y* using `axis.title.x` and `axis.title.y`.

```
ggplot(ind, aes(CPI, HDI, color = region)) + geom_jitter() + theme(axis.title = element_text(size = 20,
  color = "red"))
```



Similar for `axis.text`:

```
ggplot(ind, aes(CPI, HDI, color = region)) + geom_jitter() + theme(axis.text = element_text(size = 20,
  color = "blue"))
```



C.2.1 Axis elements

The axis elements control the appearance of the axes:

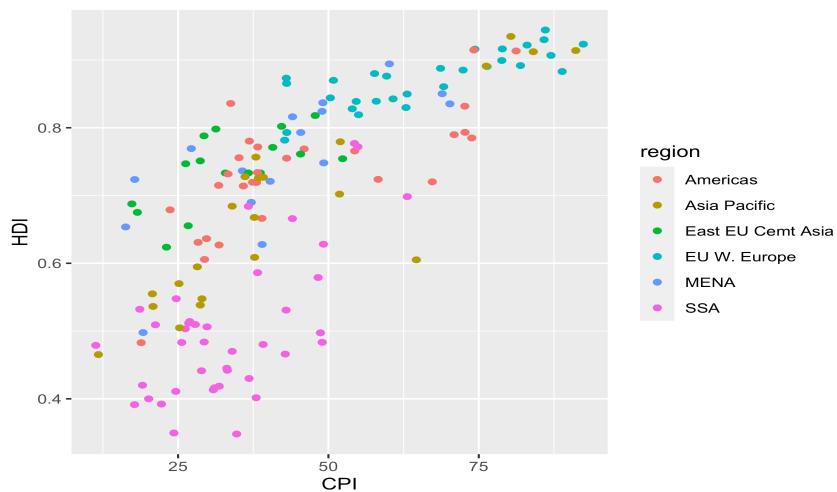
Element	Setter	Description
axis.line	element_line()	line parallel to axis (hidden in default themes)
axis.text	element_text()	tick labels
axis.text.x	element_text()	x-axis tick labels
axis.text.y	element_text()	y-axis tick labels
axis.title	element_text()	axis titles
axis.title.x	element_text()	x-axis title
axis.title.y	element_text()	y-axis title
axis.ticks	element_line()	axis tick marks
axis.ticks.length	unit()	length of tick marks

C.3 Plot title

The appearance of the plot's title can be controlled with the variable `plot.title`.

```
ggplot(ind, aes(CPI, HDI, color = region)) + geom_jitter() + ggtitle("CPI vs HDI") +
  theme(plot.title = element_text(size = 20, face = "bold"))
```

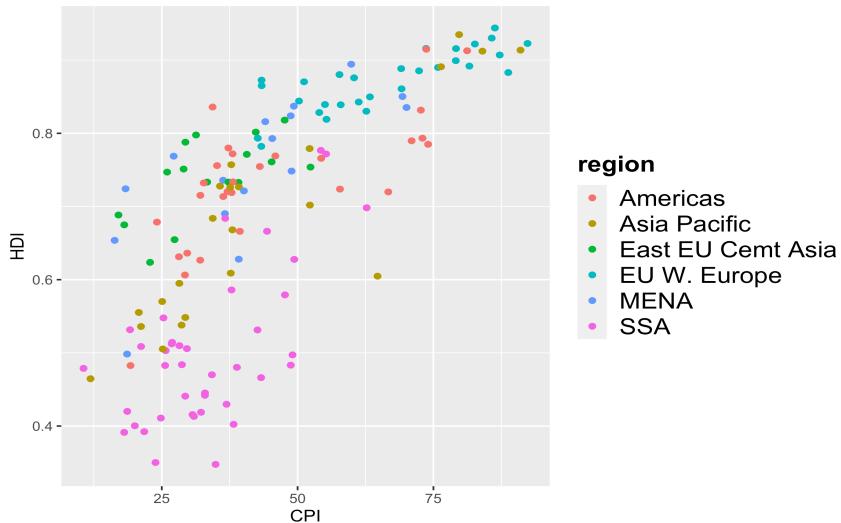
CPI vs HDI



C.4 Legend

The appearance of the legend can be controlled with `legend.text` and `legend.title`.

```
base <- ggplot(ind, aes(CPI, HDI, color = region)) + geom_jitter()
base + theme(legend.text = element_text(size = 15), legend.title = element_text(size = 15,
  face = "bold"))
```



The legend elements control the appearance of all legends. You can also modify the appearance of individual legends by modifying the same elements in `guide_legend()` or `guide_colourbar()`.

Element	Setter	Description
legend.background	element_rect()	legend background
legend.key	element_rect()	background of legend keys
legend.key.size	unit()	legend key size
legend.key.height	unit()	legend key height
legend.key.width	unit()	legend key width
legend.margin	unit()	legend margin
legend.text	element_text()	legend labels
legend.text.align	0–1	legend label alignment (0 = right, 1 = left)
legend.title	element_text()	legend name
legend.title.align	0–1	legend name alignment (0 = right, 1 = left)

C.5 Interactive plots

Interactive plots can improve presentations. The library `plotly` allows to interactively construct plots. We can simply construct a plot with `ggplot2` and pass this object to the function `ggplotly()`:

```
library(plotly)
p <- ggplot(mtcars, aes(factor(cyl), mpg)) + geom_boxplot()
ggplotly(p)
```

We can also save the generated `plotly` object into an `html` file as follows:

```
plotly_path <- "assets/fig/my_plotly_fig.html"
save_plotly_as_widget(ggplotly(p), file.path(getwd(), plotly_path))
```

D Probabilities

This Appendix covers basic definitions and results of probabilities. It does not aim to replace a lecture on probabilities.

D.1 Probability, conditional probability, and dependence

We indistinguishably denote $p(a)$:

- the probability of a logical event A to occur
- the probability of a discrete random variable A to take the value a
- the probability mass density of a continuous random variable A at the value a

The **joint probability** of two events to occur (two random variables to take particular values) is denoted $p(a, b)$.

The **conditional probability** of an event a given that b occurs, denoted $p(a|b)$ and said “probability of a given b”, is defined as:

$$p(a|b) := \frac{p(a, b)}{p(b)}$$

The random variables a and b are **independent**, denoted $a \perp b$, if and only if

$$p(a, b) = p(a)p(b)$$

This equivalent to say that $p(a|b) = p(a)$ and that $p(b|a) = p(b)$.

Otherwise, the random variables a and b are dependent, denoted $a \not\perp b$.

These results generalize to discrete random variables and to probability densities of continuous random variables.

D.2 Expected value, variance, and covariance

If X is a random variable with a probability density function $p(x)$, then the **expected value** is defined as the sum (for discrete random variables) or integral (for univariate continuous random variables):⁵⁴

$$\mathbb{E}[X] = \int xp(x) dx$$

The **variance** is defined as:

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

The **standard deviation** is the squared root of the variance:

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

The **covariance** of two random variables X and Y is defined as:

$$\text{Cov}[(X, Y)] = E[(X - E[X])(Y - E[Y])]$$

The **Pearson correlation coefficient** $\rho_{X,Y}$ of two random variables X and Y is defined as:

$$\rho_{X,Y} = \frac{\text{Cov}[(X, Y)]}{\text{SD}[X] \text{SD}[Y]}$$

The expected value of multidimensional random variables is defined per component. That is,

$$E[(X_1, \dots, X_n)] = (E[X_1], \dots, E[X_n])$$

The **covariance matrix** of a multidimensional random variables X is the matrix of all pairwise covariances, i.e. with (i, j) -th element being:

$$(\text{Cov}[X])_{i,j} = \text{Cov}[(X_i, X_j)]$$

D.3 Sample estimates

Let $\{x_1, \dots, x_n\}$ a finite sample of size n of independent realizations of a random variable X . Considered as random variables, the x_i are independently and identically distributed (i.i.d.).

The **sample mean**, often denoted \bar{x} , is defined as:

$$\bar{x} = \frac{1}{n} \sum_i x_i$$

The sample mean is an unbiased estimator of the expected value. That is, $E[\bar{x}] = E[X]$.

The **sample variance** is defined as:

$$\sigma_x^2 = \frac{1}{n} \sum_i (x_i - \bar{x})^2$$

The sample variance is not an unbiased estimator of the variance. Therefore, one often uses the **unbiased sample variance**, defined as:

$$s_x^2 = \frac{1}{n-1} \sum_i (x_i - \bar{x})^2$$

for which $E[s_x^2] = \text{Var}[X]$ holds.

The **sample standard deviation** and the **unbiased sample standard deviation** are defined as the squared root of their variance counterparts.

D.4 Linear regression

This is the proof for the univariate linear regression estimates.

For a data set $(x, y)_i$ with $i \in \{1 \dots N\}$ the univariate linear model is defined as

$$y_i = \alpha + \beta x_i + \epsilon_i$$

with free parameters α and β and a random error $\epsilon_i \sim N(0, \sigma^2)$ that is i.i.d. (independently and identically distributed).

The normal distribution is defined as

$$N(\epsilon|0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right).$$

The assumption that the errors ϵ_i are independent and identically distributed allows us to factorize the Likelihood of the data under the linear model as

$$L(\alpha, \beta, \sigma^2) = \prod_{i=1}^N N(\epsilon_i|0, \sigma^2),$$

using the fact that the probability of independent events is the product of the probabilities of each individual event.

We are interested in finding the parameters α , β and σ^2 that best model our data. This can be achieved by finding the parameters that maximize the likelihood of our data. As maximizing the likelihood is equivalent to maximizing the log likelihood and as the log likelihood is easier to handle, we will use the log likelihood in the following.

The log likelihood of our data is defined as follows:

$$\begin{aligned} \log(L(\alpha, \beta, \sigma^2)) &= \log\left(\prod_{i=1}^N N(\epsilon_i|0, \sigma^2)\right) \\ &= \sum_{i=1}^N \log(N(y_i - (\alpha + \beta x_i)|0, \sigma^2)) \\ &= -0.5N \log(2\pi\sigma^2) + \sum_{i=1}^N -\frac{(y_i - (\alpha + \beta x_i))^2}{2\sigma^2}. \end{aligned}$$

We can maximize a quadratic function by computing its gradient and setting it to zero, this yields:

$$\begin{aligned} \hat{\alpha} &= \bar{y} - \hat{\beta} \bar{x} \\ \hat{\beta} &= \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^N (x_i - \bar{x})^2} \\ \hat{\sigma}^2 &= \frac{1}{N} \sum_{i=1}^N (y_i - (\hat{\alpha} + \hat{\beta} x_i))^2 \end{aligned}$$

with means denoted by \bar{x} and \bar{y} .

D.5 Resources

The chapters on probability and random variables of Rafael Irizzary's book Introduction to Data Science Chapters gives related primer material [<https://rafalab.github.io/dsbook/>].

Davison, A. C., and D. V. Hinkley. 1997. *Bootstrap Methods and Their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press. <https://doi.org/10.1017/CBO9780511802843>.

Gagneur, Julien, Oliver Stegle, Chenchen Zhu, Petra Jakob, Manu M. Tekkedil, Raeka S. Aiyar, Ann-Kathrin Schuon, Dana Pe'er, and Lars M. Steinmetz. 2013. "Genotype-Environment Interactions Reveal Causal Pathways That Mediate Genetic Effects on Phenotype." *PLOS Genetics* 9 (9): 1–10. <https://doi.org/10.1371/journal.pgen.1003803>.

Hipson, Belinda, and Gordon K Smyth. 2010. "Permutation P-Values Should Never Be Zero: Calculating Exact P-Values When Permutations Are Randomly Drawn." *Statistical Applications in Genetics and Molecular Biology* 9 (1). <https://doi.org/https://doi.org/10.2202/1544-6115.1585>.

Wasserstein, Ronald L., and Nicole A. Lazar. 2016. "The Asa Statement on P-Values: Context, Process, and Purpose." *The American Statistician* 70 (2): 129–33. <https://doi.org/10.1080/00031305.2016.1154108>.

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software, Articles* 59 (10): 1–23. <https://doi.org/10.18637/jss.v059.i10>.

54. This loose definition suffices for our purposes. Correct mathematical definitions of the expected value are involved.

See https://en.wikipedia.org/wiki/Expected_value ↪