HSE

G. Moroz

2020

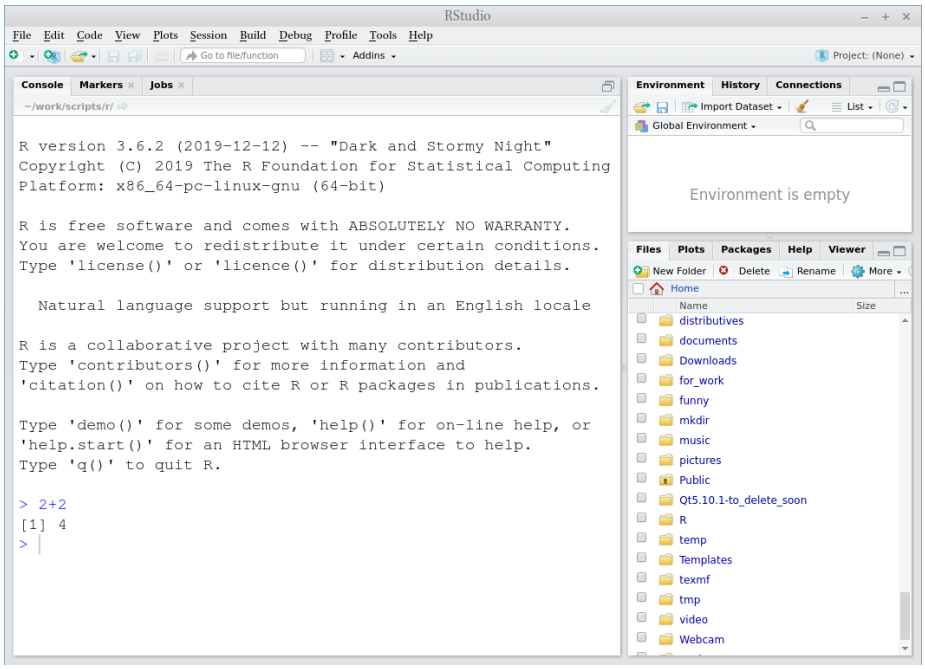
Contents

Chapter 1

Prerequisites

, :

- **R**, : https://cloud.r-project.org/
- **RStudio**, : https://rstudio.com/products/rstudio/download/#download (FREE version, !)
- RStudio, 2+2, `Enter`.



- , .

- https://rstudio.cloud/. ,
, - .
— .

Chapter 2

R RStudio

2.1

2.1.1 data science?

Data science (- - : , , - .) —
, . ,
, data science .
, data science:

-
-
-
-
-
- …

“Data Science for …”:

- psychologists (Hansjörg, 2019)
- immunologists (Thomas and Pallett, 2019)
- business (Provost and Fawcett, 2013)
- public policy (Brooks and Cooper, 2013)
- fraud detection (Baesens et al., 2015)
- …

data science :

-
-
-
- ,
-

•

2.1.2 R?

R (R Core Team, 2019) — ,

.

, :

- Python (VanderPlas, 2016; Grus, 2019)
- Julia (Bezanson et al., 2017)
- bash (Janssens, 2014)
- java (Brzustowicz, 2017)
- ...

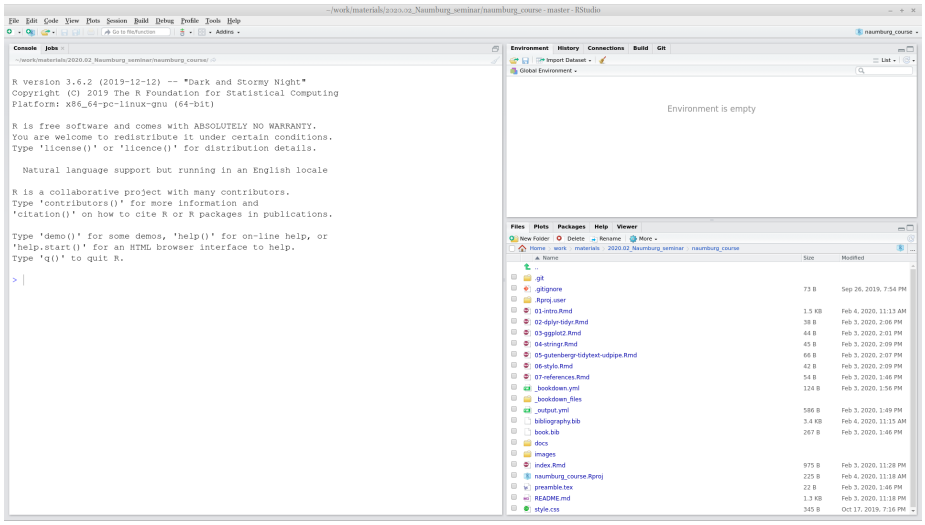
R :

- “R for data science” (Wickham, 2016)
- R community
- stackoverflow
-
- ...

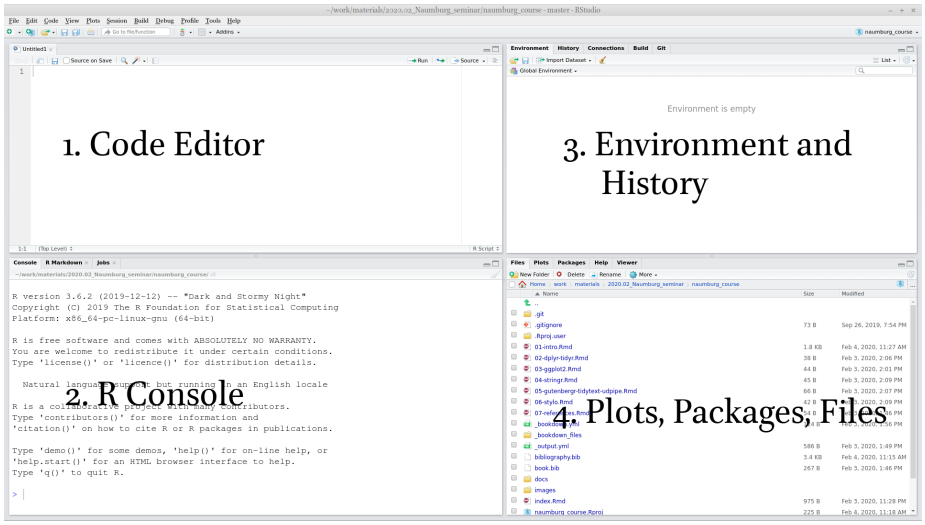
2.2 Introduction to RStudio

R is the programming language. RStudio is the most popular IDE (Integrated Development Environment) for R language.

When you open RStudio for the first time you can see something like this:



When you press  button at the top of the left window you will be able to see all four panels of RStudio.



2.3 R as a calculator

Lets first start with the calculator. Press in R console

```
2+9
```

```
## [1] 11
```

```
50*(9-20)
```

```
## [1] -550
```

```
3^3
```

```
## [1] 27
```

```
9^0.5
```

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

```
9+0.5
```

```
## [1] 9.5
```

```
9+.5
```

```
## [1] 9.5
```

```
pi
```

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

Remainder after division

```
10 %% 3
```

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

So you are ready to solve some really hard equations (round it four decimal places):

$$\frac{\pi + 2}{2^{3-\pi}}$$

list of hints

Are you sure that you rounded the result? I expect the answer to be rounded to four decimal places: `0.87654321` becomes `0.8765`.

Are you sure you didn't get into the brackets trap? Even though there isn't any brackets in the mathematical notation, you need to add them in R, otherwise the operation order will be wrong.

2.4 Comments

Any text after a hash `#` within the same line is considered a comment.

```
2+2 # it is four
```

```
## [1] 4
```

```
# you can put any comments here
3+3
```

```
## [1] 6
```

2.5 Functions

The most important part of R is functions: here are some of them:

```
sqrt(4)
```

```
## [1] 2
```

```
abs(-5)
```

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

```
sin(pi/2)
```

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

```
cos(pi)
```

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

```
sum(2, 3, 9)
```

```
## [1] 14
```

```
prod(5, 3, 9)
```

```
## [1] 135
```

```
sin(cos(pi))
```

```
## [1] -0.841471
```

Each function has a name and zero or more arguments. All arguments of the function should be listed in parenthesis and separated by comma:

```
pi
```

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

```
round(pi, 2)
```

```
## [1] 3.14
```

Each function's argument has its own name and serial number. If you use names of the function's arguments, you can put them in any order. If you do not use names of the function's arguments, you should put them according the serial number.

```
round(x = pi, digits = 2)
```

```
## [1] 3.14
```

```
round(digits = 2, x = pi)
```

```
## [1] 3.14
```

```
round(x = pi, d = 2)
```

```
## [1] 3.14
```

```
round(d = 2, x = pi)
```

```
## [1] 3.14
```

```
round(pi, 2)
```

```
## [1] 3.14
```

```
round(2, pi) # this is not the same as all previous!
```

```
## [1] 2
```

There are some functions without any arguments, but you still should use parenthesis:

```
Sys.Date() # correct
```

```
## [1] "2020-04-13"
```

```
Sys.Date # wrong
```

```
## function ()
## as.Date(as.POSIXlt(Sys.time()))
## <bytecode: 0x60e5ef924068>
## <environment: namespace:base>
```

Each function in R is documented. You can read its documentation typing a question mark before the function name:

```
?Sys.Date
```

Explore the function `log()` and calculate the following logarithm:

$$\log_3(3486784401)$$

list of hints

A-a-a! I don't remember anything about logarithms... The logarithm is the inverse function to exponentiation. That means the logarithm of a given number x is the exponent to which another fixed number, the base b, must be raised, to produce that number x.

$$10^n = 1000, \text{ what is n?}$$

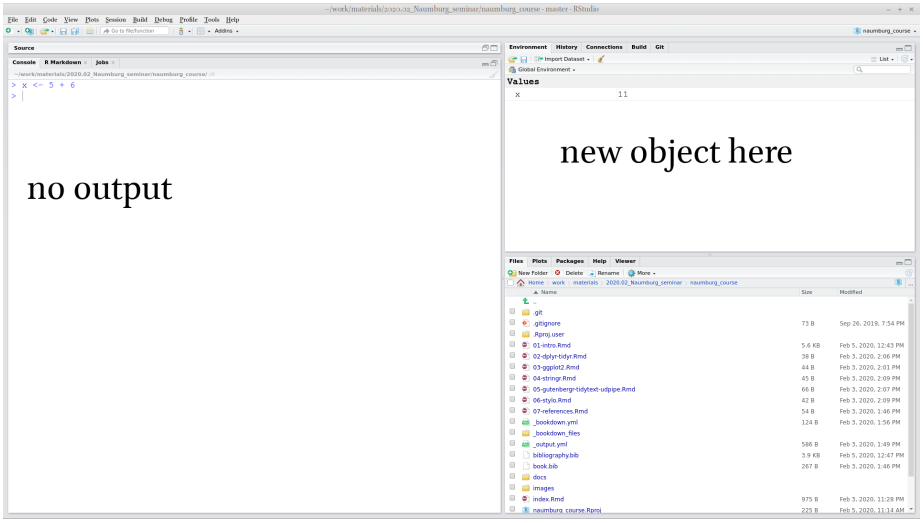
$$n = \log_{10}(1000)$$

What does this small 3 in the task mean? This is the base of the logarithm. So the task is: what is the exponent to which another fixed number, the base *3*, must be raised, to produce that number *3486784401*.

2.6 Variables

Everything in R can be stored in a variable:

```
x <- 5 + 6
```



As a result, no output in the Console, and a new variable x appear in the Environment window. From now on I can use this new variable:

```
x + x
```

```
## [1] 22
```

```
sum(x, x, 7)
```

```
## [1] 29
```

All those operations don't change the variable value. In order to change the variable value you need to make a new assignment:

```
x <- 5 + 6 + 7
```

The fast way for creating `<-` in RStudio is to press `Alt -` on your keyboard.

It is possible to use equal sign `=` for assignment operation, but the recommendations are to use arrow `<-` for the assignment, and equal sign `=` for giving arguments' value inside the functions.

For removing vector you need to use the function `rm()`:

```
rm(x)
x
```

```
## Error in eval(expr, envir, enclos): object 'x' not found
```

2.6.1 Variable comparison

It is possible to compare different variables

```
x <- 18
x > 18
```

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

```
x >= 18
```

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

```
x < 100
```

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

```
x <= 18
```

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

```
x == 18
```

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

```
x != 18
```

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

Operator `!` can work by itself changing logical values into reverse:

```
!TRUE
```

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

```
!FALSE
```

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

2.6.2 Variable types

There are several types of variables in R. In this course the only important types will be `double` (all numbers), `character` (or strings), and `logical`:

```
x <- 2+3
typeof(x)
```

```
## [1] "double"
```

```
y <- "Cześć"
typeof(y)
```

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

```
z <- TRUE
typeof(z)
```

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

2.7 Vector

An R object that contains multiple values of the same type is called **vector**. It could be created with the command `c()`:

```
c(3, 0, pi, 23.4, -53)
```

```
## [1]   3.000000   0.000000   3.141593  23.400000 -53.000000
```

```
c("Kraków", "Warszawa", "Cieszyn")
```

```
## [1] "Kraków"   "Warszawa" "Cieszyn"
```

```
c(FALSE, FALSE, TRUE)
```

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

```
a <- c(2, 3, 4)
b <- c(5, 6, 7)
c(a, b)
```

```
## [1] 2 3 4 5 6 7
```

For the number sequences there is an easy way:

```
1:10
```

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

```
3:-5
```

```
## [1]  3  2  1  0 -1 -2 -3 -4 -5
```

From now on you can understand that everything we have seen before is a vector of length one. That is why there is `[1]` in all outputs: it is just an index of elements in a vector. Have a look here:

```
1:60
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51 52 53 54 55 56 57 58 59 60
```

```
60:1
```

```
##  [1] 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36
## [26] 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11
## [51] 10  9  8  7  6  5  4  3  2  1
```

There is also a function `seq()` for creation of arithmetic progressions:

```
1:20
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
seq(from = 1, to = 20, by = 1)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
seq(from = 2, to = 100, by = 13)
```

```
## [1]  2 15 28 41 54 67 80 93
```

Use the argument `length.out` of function `seq()` and create an arithmetic sequence from π to 2π of length 50.

There are also some built-in vectors:

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
month.name
```

```
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
```

```
month.abb
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

2.7.1 Vector coercion

Vectors are R objects that contain multiple values of **the same type**. But what if we merged together different types?

```
c(1, "34")
```

```
## [1] "1"  "34"
```

```
c(1, TRUE)
```

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

```
c(TRUE, "34")
```

```
## [1] "TRUE" "34"
```

It is clear that there is a hierarchy: strings > double > logical. It is not universal across different programming languages. It doesn't correspond to the amount of values of particular type:

```
c(1, 2, 3, "34")
```

```
## [1] "1"  "2"  "3"  "34"
```

```
c(1, TRUE, FALSE, FALSE)
```

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

The same story could happen during other operations:

```
5+TRUE
```

```
## [1] 6
```

2.7.2 Vector operations

All operations, that we discussed earlier, could be done with vectors of the same length:

```
1:5 + 6:10
```

```
## [1]  7  9 11 13 15
```

```
1:5 - 6:10
```

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

```
1:5 * 6:10
```

```
## [1]  6 14 24 36 50
```

There are operations where the vector of any length and vector of length one is involved:

```
1:5 + 7
```

```
## [1]  8  9 10 11 12
```

```
1:5 - 7
```

```
## [1] -6 -5 -4 -3 -2
```

```
1:5 / 7
```

```
## [1] 0.1428571 0.2857143 0.4285714 0.5714286 0.7142857
```

There are a lot of functions in R that are **vectorised**. That means that applying this function to a vector is the same as applying this function to each element of the vector:

```
sin(1:5)
```

```
## [1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243
```

```
sqrt(1:5)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```
abs(-5:3)
```

```
## [1] 5 4 3 2 1 0 1 2 3
```

2.7.3 Indexing vectors

How to get some value or banch of values from a vector? You need to index them:

```
x <- c(3, 0, pi, 23.4, -53)
y <- c("Kraków", "Warszawa", "Cieszyn")

x[4]
```

```
## [1] 23.4
```

```
y[2]
```

```
## [1] "Warszawa"
```

It is possible to have a vector as index:

```
x[1:2]
```

```
## [1] 3 0
```

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

```
## [1] "Kraków"  "Cieszyn"
```

It is possible to index something that you **do not** want to see in the result:

```
y[-2]
```

```
## [1] "Kraków"  "Cieszyn"
```

```
x[-c(1, 4)]
```

```
## [1]   0.000000   3.141593 -53.000000
```

It is possible to have other variables as an index

```
z <- c(3, 2)
x[z]
```

```
## [1] 3.141593 0.000000
```

```
y[z]
```

```
## [1] "Cieszyn"  "Warszawa"
```

It is possible to index with a logical vector:

```
x[c(TRUE, FALSE, TRUE, TRUE, FALSE)]
```

```
## [1]  3.000000  3.141593 23.400000
```

That means that we could use `TRUE/FALSE`-vector produced by comparison:

```
x[x > 2]
```

```
## [1]  3.000000  3.141593 23.400000
```

It works because `x > 2` is a vector of logical values:

```
x > 2
```

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

It is possible to use `!` operator here changing all `TRUE` values to `FALSE` and vice versa.

```
x[!(x > 2)]
```

```
## [1]   0 -53
```

How many elements in the vector `g` if expression `g[pi < 1000]` does not return an error?

2.7.4 NA

Sometimes there are some missing values in the data, so it is represented with `NA`

```
NA
```

```
## [1] NA
```

```
c(1, NA, 9)
```

```
## [1]  1 NA  9
```

```
c("Kraków", NA, "Cieszyn")
```

```
## [1] "Kraków"  NA        "Cieszyn"
```

```
c(TRUE, FALSE, NA)
```

```
## [1]  TRUE FALSE    NA
```

It is possible to check, whether there are missing values or not

```
x <- c("Kraków", NA, "Cieszyn")
y <- c("Kraków", "Warszawa", "Cieszyn")
is.na(x)
```

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

```
is.na(y)
```

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

Some functions doesn't work with vecotors that contain missed values, so you need to add argument `na.rm = TRUE`:

```
x <- c(1, NA, 9, 5)
mean(x)
```

```
## [1] NA
```

```
mean(x, na.rm = TRUE)
```

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

```
min(x, na.rm = TRUE)
```

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

```
max(x, na.rm = TRUE)
```

```
## [1] 9
```

```
median(x, na.rm = TRUE)
```

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

```
range(x, na.rm = TRUE)
```

```
## [1] 1 9
```

2.8 Packages

The most important and useful part of R is hidden in its packages. Everything that we discussed so far is basic R functionality invented back in 1979. Since then a lot of different things changed, so all new practices for data analysis, visualisation and manipulation are packed in packages. During our class we will learn the most popular *"dialect"* of R called `tidyverse`.

In order to install packages you need to use a command. Let's install the `tidyverse` package:

```
install.packages("tidyverse")
```

For today we also will need the `readxl` package:

```
install.packages("readxl")
```

After you have downloaded packages nothing will change. You can not use any fucntionality from packages unless you load the package with the `library()` function:

```
library("tidyverse")
```

Not loading a package is the most popular mistake of my students. So remember:

- `install.packages("...")` is like you are buying a screwdriver set;
- `library("...")` is like you are starting to use your screwdriver.

install.packages("...")

library("...")

For the further lectures we will need `tidyverse` package.

Please install `tidyverse` package and load it.

2.8.1 `tidyverse`

The `tidyverse` is a set of packages:

- `tibble`, for tibbles, a modern re-imagining of data frames — analogue of tables in R
- `readr`, for data import
- `dplyr`, for data manipulation
- `tidyr`, for data tidying (we will discuss it later today)
- `ggplot2`, for data visualisation
- `purrr`, for functional programming

2.9 Dataframe (tibble)

A data frame is a collection of variables of the same number of rows with unique row names. Here is an example dataframe with the Tomm Moore filmography:

```
moore_filmography <- tibble(title = c("The Secret of Kells",
                                      "Song of the Sea",
                                      "Kahlil Gibran's The Prophet",
                                      "The Breadwinner",
                                      "Wolfwalkers"),
                            year = c(2009, 2014, 2014, 2017, 2020),
                            director = c(TRUE, TRUE, TRUE, FALSE, TRUE))
moore_filmography
```

```
## # A tibble: 5 x 3
##   title                        year director
##   <chr>                       <dbl> <lgl>
## 1 The Secret of Kells          2009 TRUE
## 2 Song of the Sea              2014 TRUE
## 3 Kahlil Gibran's The Prophet  2014 TRUE
## 4 The Breadwinner              2017 FALSE
## 5 Wolfwalkers                  2020 TRUE
```

There are a lot of built-in dataframes:

```
mtcars
iris
```

You can find information about them:

```
?mtcars
?iris
```

Dataframe consists of vectors that could be called using `$` sign:

```
moore_filmography$year
```

```
## [1] 2009 2014 2014 2017 2020
```

```
moore_filmography$title
```

```
## [1] "The Secret of Kells"         "Song of the Sea"
## [3] "Kahlil Gibran's The Prophet" "The Breadwinner"
## [5] "Wolfwalkers"
```

It is possible to add a vector to an existing dataframe:

```
moore_filmography$producer <- c(TRUE, TRUE, FALSE, TRUE, TRUE)
moore_filmography
```

```
## # A tibble: 5 x 4
##   title                        year director producer
##   <chr>                       <dbl> <lgl>    <lgl>
## 1 The Secret of Kells          2009 TRUE     TRUE
## 2 Song of the Sea              2014 TRUE     TRUE
## 3 Kahlil Gibran's The Prophet  2014 TRUE     FALSE
## 4 The Breadwinner              2017 FALSE    TRUE
## 5 Wolfwalkers                  2020 TRUE     TRUE
```

There are some useful functions that tell you somethig about a dataframe:

```
nrow(moore_filmography)
```

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

```
ncol(moore_filmography)
```

```
## [1] 4
```

```
summary(moore_filmography)
```

```
##     title                year       director        producer
##  Length:5           Min.   :2009   Mode :logical   Mode :logical
##  Class :character   1st Qu.:2014   FALSE:1         FALSE:1
##  Mode  :character   Median :2014   TRUE :4         TRUE :4
##                     Mean   :2015
##                     3rd Qu.:2017
##                     Max.   :2020
```

```
str(moore_filmography)
```

```
## tibble [5 x 4] (S3: tbl_df/tbl/data.frame)
##  $ title   : chr [1:5] "The Secret of Kells" "Song of the Sea" "Kahlil Gibran's The
##  $ year    : num [1:5] 2009 2014 2014 2017 2020
##  $ director: logi [1:5] TRUE TRUE TRUE FALSE TRUE
##  $ producer: logi [1:5] TRUE TRUE FALSE TRUE TRUE
```

We will work exclusively with dataframes. But it is not the only data structure in R.

How many rows are in the `iris` dataframe?

How many columns are in the `mtcars` dataframe?

2.9.1 Indexing dataframes

Since dataframes are two-dimensional objects it is possible to index its rows and columns. Rows are the first index, columns are the second index:

```
moore_filmography[3, 2]
```

```
## # A tibble: 1 x 1
##    year
##   <dbl>
## 1  2014
```

```
moore_filmography[3,]
```

```
## # A tibble: 1 x 4
##   title                        year director producer
##   <chr>                       <dbl> <lgl>    <lgl>
## 1 Kahlil Gibran's The Prophet  2014 TRUE     FALSE
```

```
moore_filmography[,2]
```

```
## # A tibble: 5 x 1
##    year
##   <dbl>
## 1  2009
## 2  2014
## 3  2014
## 4  2017
## 5  2020
```

```
moore_filmography[,1:2]
```

```
## # A tibble: 5 x 2
##   title                        year
##   <chr>                       <dbl>
## 1 The Secret of Kells          2009
## 2 Song of the Sea              2014
```

```
## 3 Kahlil Gibran's The Prophet  2014
## 4 The Breadwinner              2017
## 5 Wolfwalkers                  2020
```

```
moore_filmography[,-3]
```

```
## # A tibble: 5 x 3
##   title                        year producer
##   <chr>                       <dbl> <lgl>
## 1 The Secret of Kells          2009 TRUE
## 2 Song of the Sea              2014 TRUE
## 3 Kahlil Gibran's The Prophet  2014 FALSE
## 4 The Breadwinner              2017 TRUE
## 5 Wolfwalkers                  2020 TRUE
```

```
moore_filmography[,-c(1:2)]
```

```
## # A tibble: 5 x 2
##   director producer
##   <lgl>    <lgl>
## 1 TRUE     TRUE
## 2 TRUE     TRUE
## 3 TRUE     FALSE
## 4 FALSE    TRUE
## 5 TRUE     TRUE
```

```
moore_filmography[,"year"]
```

```
## # A tibble: 5 x 1
##    year
##   <dbl>
## 1  2009
## 2  2014
## 3  2014
## 4  2017
## 5  2020
```

```
moore_filmography[,c("title", "year")]
```

```
## # A tibble: 5 x 2
##   title                        year
##   <chr>                       <dbl>
## 1 The Secret of Kells          2009
```

```
## 2 Song of the Sea              2014
## 3 Kahlil Gibran's The Prophet  2014
## 4 The Breadwinner              2017
## 5 Wolfwalkers                  2020
```

```
moore_filmography[moore_filmography$year > 2014,]
```

```
## # A tibble: 2 x 4
##   title            year director producer
##   <chr>           <dbl> <lgl>    <lgl>
## 1 The Breadwinner  2017 FALSE    TRUE
## 2 Wolfwalkers      2020 TRUE     TRUE
```

2.10 Data import

2.10.1 `.csv` files

A `.csv` files (comma-separated values) is a delimited text file that uses a comma (or other delemeters such as tabulation or semicolon) to separate values. It is broadly used bacause it is possible to parse such a file using computers and people can edit it in the Office programs (Microsoft Excel, LibreOffice Calc, Numbers on Mac). Here is our `moore_filmography` dataset in the `.csv` format:

```
title,year,director,producer
The Secret of Kells,2009,TRUE,TRUE
Song of the Sea,2014,TRUE,TRUE
Kahlil Gibran's The Prophet,2014,TRUE,FALSE
The Breadwinner,2017,FALSE,TRUE
Wolfwalkers,2020,TRUE,TRUE
```

Let's create a variable with this file:

```
our_csv <- "title,year,director,producer
The Secret of Kells,2009,TRUE,TRUE
Song of the Sea,2014,TRUE,TRUE
Kahlil Gibran's The Prophet,2014,TRUE,FALSE
The Breadwinner,2017,FALSE,TRUE
Wolfwalkers,2020,TRUE,TRUE"
```

Now we are ready to use `read_csv()` function:

```
read_csv(our_csv)
```

```
## # A tibble: 5 x 4
##   title                        year director producer
```

```
##   <chr>                       <dbl> <lgl>    <lgl>
## 1 The Secret of Kells          2009 TRUE     TRUE
## 2 Song of the Sea              2014 TRUE     TRUE
## 3 Kahlil Gibran's The Prophet  2014 TRUE     FALSE
## 4 The Breadwinner              2017 FALSE    TRUE
## 5 Wolfwalkers                  2020 TRUE     TRUE
```

It is also possible to read files from your computer. Download this file on your computer (press `Ctrl S` or `Cmd S`) and read into R:

```
read_csv("C:/path/to/your/file/moore_filmography.csv")
```

```
## # A tibble: 5 x 4
##   title                       year director producer
##   <chr>                      <dbl> <lgl>    <lgl>
## 1 The Secret of Kells          2009 TRUE     TRUE
## 2 Song of the Sea              2014 TRUE     TRUE
## 3 Kahlil Gibran's The Prophet  2014 TRUE     FALSE
## 4 The Breadwinner              2017 FALSE    TRUE
## 5 Wolfwalkers                  2020 TRUE     TRUE
```

It is also possible to read files from the Internet:

```
read_csv("https://raw.githubusercontent.com/agricolamz/2020.02_Naumburg_R/master/data/m
```

```
## Parsed with column specification:
## cols(
##   title = col_character(),
##   year = col_double(),
##   director = col_logical(),
##   producer = col_logical()
## )
```

```
## # A tibble: 5 x 4
##   title                       year director producer
##   <chr>                      <dbl> <lgl>    <lgl>
## 1 The Secret of Kells          2009 TRUE     TRUE
## 2 Song of the Sea              2014 TRUE     TRUE
## 3 Kahlil Gibran's The Prophet  2014 TRUE     FALSE
## 4 The Breadwinner              2017 FALSE    TRUE
## 5 Wolfwalkers                  2020 TRUE     TRUE
```

Because of the 2019–20 Wuhan coronavirus outbreak the city of Wuhan is on media everywhere. In Russian for some reason Wuhan is sometimes masculine and sometimes it is feminin. I looked into other Slavic languages

and recorded obtained data into the `.csv` file. Download this files to R. What variables does it have?

All file manipulations in R are somehow connected with space on your computer via working directory. You can get information about your current working directory using `getwd()` function. You can change your working directory using `setwd()` function. If a file you want to read is in the working directory you don't need to write the whole path to file:

```
read_csv("moore_filmography.csv")
```

The same simple function will create your `.csv` file:

```
write_csv(moore_filmography, "moore_filmography_v2.csv")
```

Sometimes reading `.csv` files into Microsoft Excel is complicated, please follow the following instructions.

2.10.2 `.xls` and `.xlsx` files

There is a package `readxl` that allows to open and save `.xsl` and `.xslx` files. Install and load the package:

```
library(readxl)
```

Here is a test file. Download it to your computer and put it to your working directory:

```
read_xlsx("moore_filmography.xlsx")
```

```
## # A tibble: 5 x 4
##   title                        year director producer
##   <chr>                       <dbl> <chr>    <chr>
## 1 The Secret of Kells          2009 TRUE     TRUE
## 2 Song of the Sea              2014 TRUE     TRUE
## 3 Kahlil Gibran's The Prophet  2014 TRUE     FALSE
## 4 The Breadwinner              2017 FALSE    TRUE
## 5 Wolfwalkers                  2020 TRUE     TRUE
```

`.xls` and `.xlsx` files could have multiple tables on different sheets:

```
read_xlsx("moore_filmography.xlsx", sheet = "iris")
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

Chapter 3

: dplyr ggplot2

- , :

```
library(tidyverse)
```

Chapter 4

: stringr, gutenbergr, tidytext, udpipe

stringr (tidyverse), :

```
library(tidyverse)
```

Bibliography

Baesens, B., Van Vlasselaer, V., and Verbeke, W. (2015). *Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection*. John Wiley & Sons.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Brooks, H. and Cooper, C. L. (2013). *Science for public policy*. Elsevier.

Brzustowicz, M. R. (2017). *Data Science with Java: Practical Methods for Scientists and Engineers*. O'Reilly Media, Inc.

Grus, J. (2019). *Data science from scratch: first principles with python*. O'Reilly Media, Inc.

Hansjörg, N. (2019). *Data Science for Psychologists*. self published.

Janssens, J. (2014). *Data Science at the Command Line: Facing the Future with Time-tested Tools*. O'Reilly Media, Inc.

Provost, F. and Fawcett, T. (2013). *Data Science for Business: What you need to know about data mining and data-analytic thinking*. O'Reilly Media, Inc.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Thomas, N. and Pallett, L. (2019). *Data Science for Immunologists*. CreateSpace Independent Publishing Platform.

VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data*. O'Reilly Media, Inc.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.