Data Analysis Coursebook

Marcell Granat & Zoltan Madari

Contents

Welcome

I hope this message finds you well.

This is an online bookdown file, which we plan to *update regularly* with the class material. We suggest that you to write the code simultaneously with us at seminar, but bugs can always occur out of the blue… If you missed something or just want to revisit the topic with additional comments (probably before the exam day) this page is here to help.

At the end of the course you should know:

- What are data types, data quality, and data preprocessing?
- What are the components of `tidyverse` and what are their advantage?
- What are density, distribution function, quantile functions?
- What are data clustering techniques?
- What are the main techniques for association analysis?

We will work with `R`, one the most loved statistical programming language. Do not be afraid if you do not have any programming experience, this is a beginner course. However, by the end of the program, we hope you will find useful the concept and practical tips we offer, and you will be able to solve your own real life data analysis issues.

Syllabus

0.1 Topics

- **Basic R knowledge (Week 1)**
  - Data categorize, sampling, importing-exporting
  - Types, tables, selection, objects, functions
- **Data manipulation in Tidyverse (Week 2)**
  - Filter, group_by, arrange, summarize commands
  - `%\>%`
  - Join (mutating, filtering)
  - tidy data (longer, wider)
- **Visualization with ggplot2 (Week 3)**
  - Layers, facets, geoms
  - Descriptive statistics in R
  - Summary statistics, variability, correlation, covariance
  - Extreme values, problem of missing values
- **Statistical estimation (Week 4)**
  - Distributions
  - Sample techniques, confidence intervals, standard error
- **Hypothesis testing I (Week 5)**
  - Inductive statistics in R
  - Null and alternative hypothesis, t-test, p-value, fals positive/negative, Type I and II error

- **Hypothesis testing II (Week 6)**
  - Relation testing in R
- **Project presentations (Week 7)**

0.2 Requirements

- **Test** (30%): 90 min task solution in R + explanation (exam week)
- **Project** (25%): groups (3-4 students), one detailed report (essay) and one - **presentation**
- **Homeworks** and essays (45%)
- **Extra** tasks (+5%)

0.2.1 Project

By the end of the course you have to make a statistical report about your own research project in small groups. You have to find a proper dataset (kaggle, eurostat, UCI datasets etc.) or conduct an own survey in optional topic.

At week 7 you will hold a presentation about your findings and results. At the end of the week you will upload your detailed essay about your project work.

0.2.2 Homework

During the Study period you get 3-5 homeworks. The number of homeworks depends ont he material progress. It could be R code writing or R code writing and analysis (short essay about methods and results).

Table 1: Grades

0-50%	Fail (1)
51-62%	Pass (2)
63-74%	Satisfactory (3)
75-86%	Good (4)
87-100%	excellent (5)

0.3 Recommended (compulsory) reading

- **Grolemund G & Wickham H: R for Data Science**
- Gábor Békés & Gábor Kézdi: Data Analysis: Patterns, Prediction and Causality

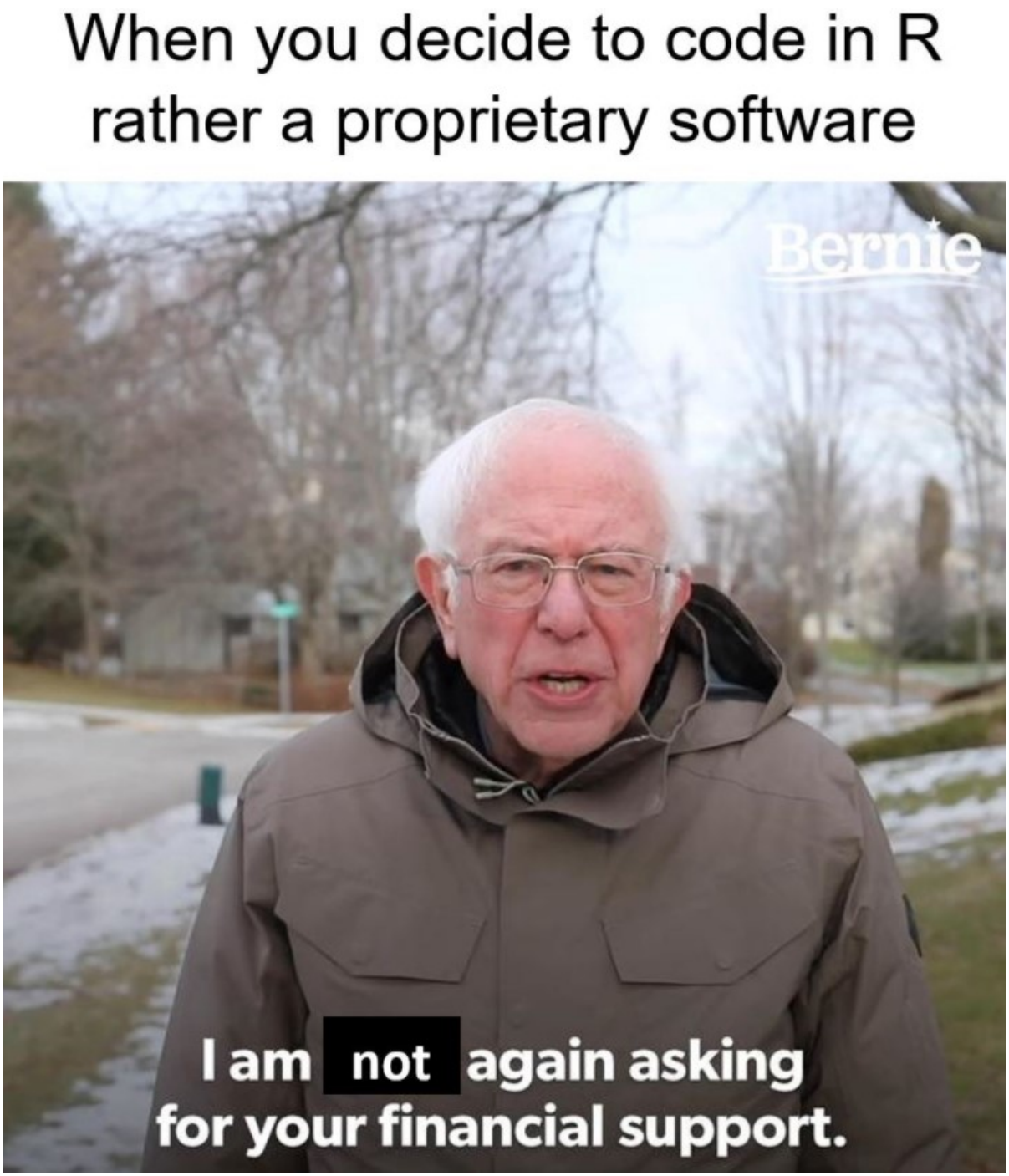
Week 1

1 Lecture 1

2 Introduction to R

2.1 Why R?

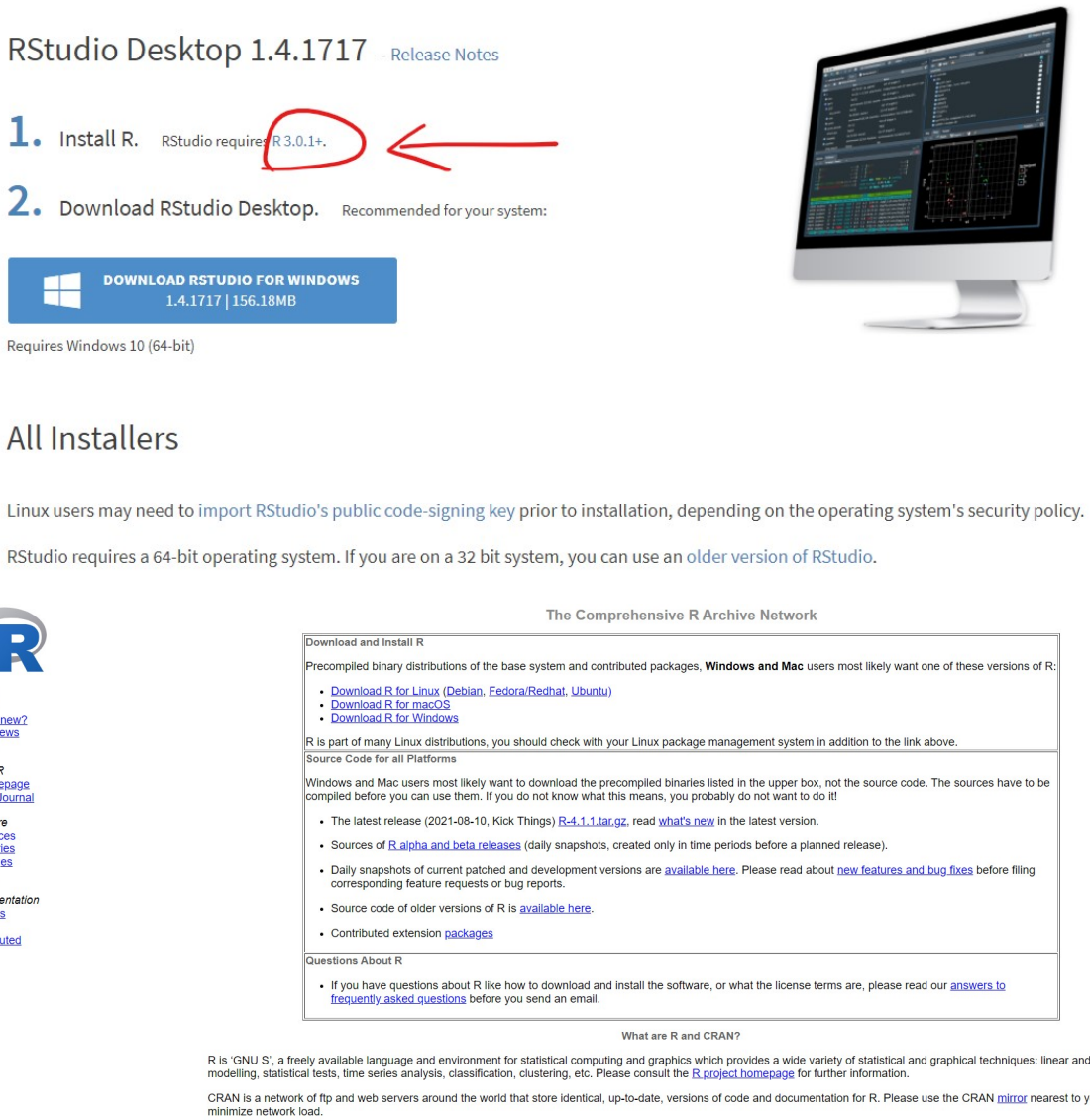
In this chapter we will discuss the basics of R programming. R is a free software, used by millions in the field of statistics, data science, economics and many others.



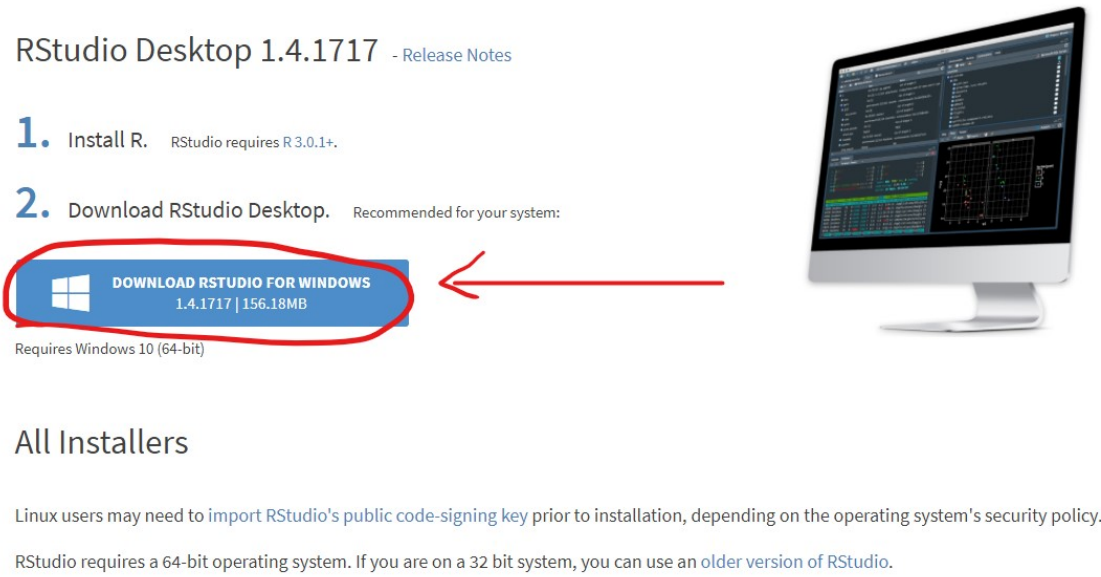
The R programming language is an important tool for data related tasks, but it is much more. Just like other programming languages, R has many additional packages, which can extend its basic functionality. R has a great (probably the best) graphical tools to create your charts, and with shiny, you can easily build your minimalist web applications. We will learn about data manipulation, analysis and how to create awesome reports, like dashboards.

2.2 Setup

You can download R and RStudio from the official site of RStudio. Please install the appropriate version based on your OS, and do not forget that you also have to install R as well.



Run R's installer file after the downloading process is finished. Next, we will also need the RStudio.



If the installation process of R and RStudio is finished, then we can open RStudio and start to learn the software.

2.3 Our first meet with R

RStudio is dedicated IDEE for R, which means, that it will make our life much simplier. In stead of writing each line of code ourself, RStudio has many built-in functions to help us. We see some panes if we open RStudio:

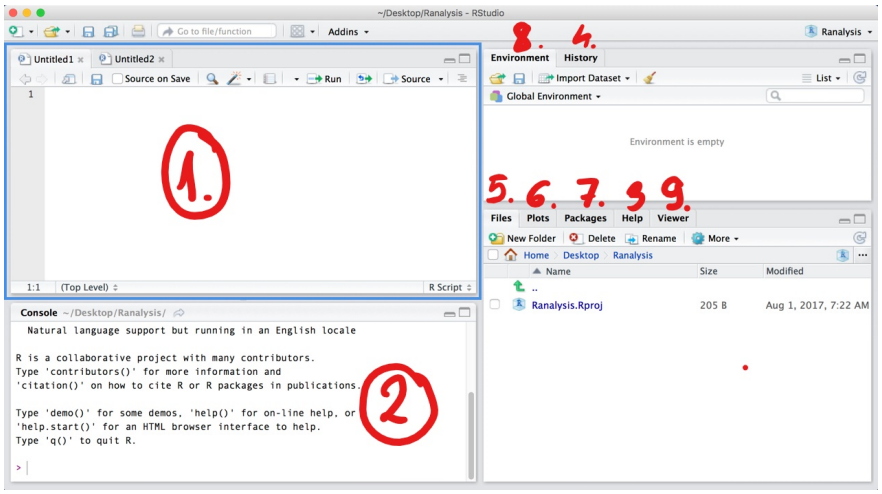


Figure 1: Panes in RStudio

1. Source
   - We will write here our codes, which we would like to save. The basic extension of our codes are `.R`, but this is not the only possibility (we will cover this later). Once you save your code for later use, you can open your script also with a simple text editor (like Notepad), since this is only plain text. If you hit `enter` your code wont be executed, you will just simply start a new line. If you want to run your code hit `ctrl + enter` to execute a single line, and `ctrl+shift+enter` to execute your full script.

2. Console
    - Here you find the executed codes, and the response to that. For example, if you type `2 + 2` and hit `enter`, R will execute the expression, and response that it is 4.

```
2 + 2
#> [1] 4
```

3. Help
    - You can use this pane if you are not familier with a function. For example, you want to know what input you can specify while using `mean`, you can type `?mean` on the console, or use the search field on this pane. The description of the function will be presented on this pane. (This pane is super useful on the exam)
4. History
5. Files
    - You can see the list of your files which are in the current working directory. Working directory is the folder, from where R want currently read the files. If you want to import a dataset, just click on a file on this pane.
    - I highly recommend you to set a project folder for the class and any later job. This means that, R creates a folder and puts an `.Rproj` file into it. You can always click on this `.Rproj` file to return your unfinished work. You can customise if R should put the variables into your environtent as you left them last time, you have a history about the used codes, and you see all the data you copy + paste into this folder.
6. Plots
7. Packages
    - You can install packages from this pane. If you need a given package, click on install, and start typing its name. After that, you have to activate packages each time you open R again with the `library(eurostat)` command. You can also use a function from a package if you just simly type `eurostat::get_eurostat()`.
8. Environment
    - Here you can see the list of the variables you have already created. For example you can type `x = 3` on the console. Now and x variable will appear in the environment pane, and you can check its value if you type `x` on the console. You can also save these variables into an `.RData` data format if you wish.
9. Viewer

2.4 Data types

Lets see first, what kind of datatypes exist in R. Lets assign a variable called `x`.

```
x <- 4
```

So, what is the type of `x`? We can use the `class` command to answer this.

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

Its numeric[1]. This means that you can use `+`, `-`, `*` operators on it.

Lets see other types.

[1] Integer and double also exist in R, but these are not the default, and variables will be always coerced automatically

```
y <- "blue"
class(y)
#> [1] "character"
```

Its a character, basically can contain any kind of letter, digits, or white space.

```
does_it_rain <- TRUE
class(does_it_rain)
#> [1] "logical"
```

Its a logical value. It can be `TRUE` or `FALSE`

2.4.1 vectors

We can create a vector with the `c` function. (combine)

```
x <- c(11, 201, 301)
x
#> [1]  11 201 301
```

We can asses a given element of it by:

```
x[2]
#> [1] 201
```

Or we can use functions on it:

```
sum(x)
#> [1] 513
```

We can also easily create sequence with the syntax `start:stop`

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

If we combine characters, I mentiont that we can convert this vector to **factor** type. This is useful if we can enclose an order to the vector or we want to control for the possible values. Lets see a minimal example

```
my_vector <- c("First", "Second", "Third", "Fourth")
sort(my_vector)
#> [1] "First"  "Fourth" "Second" "Third"
```

If we want to sort the vector, we see that *Fourth* comes right after *First*. It is because character vectors are sorted in alphabetical order. We can solve it with `factor`

```
my_vector2 <- factor(my_vector, ordered = TRUE, levels = c("First", "Second", "Third", "Fourth"))
sort(my_vector2)
#> [1] First  Second Third  Fourth
#> Levels: First < Second < Third < Fourth
```

We can merge these vectors into a data.frame, which is basically like an excel table. Each column is a variable (with a header), and each row is an observation.

```
avengers_df <- data.frame(name = c("Captain America", "Hulk", "Dr. Strange"),
           color = c("blue", "green", NA))

avengers_df
#>              name color
#> 1 Captain America  blue
```

```
#> 2            Hulk green
#> 3     Dr. Strange  <NA>
```

NA stands for "not available", so these values are missing. Most of the times we will work with data.frames (similarly like pandas in python), so it is the most important data type we learn.

Storing more complex data, you can use the `list`. To use `data.frame` you need vectors with equal length. If this does not hold, or a more frequent case, you want to store a collection of data.frames, then `list` is a perfect solution! It is not a rare issue, big panel dataset are usually stored in separated files (a different file to each year, like: `cis_survey2016.csv`, `cis_survey2017.csv`). In this situations its suggested to store your data in a list.

```
mylist <- list(avengers_df, my_vector, x)
```

Now `mylist` stores a data.frame and two vector. You can access the components with a `[[ ]]`. For example, the first element:

```
mylist[[1]]
#>              name color
#> 1 Captain America  blue
#> 2            Hulk green
#> 3     Dr. Strange  <NA>
```

2.5 Data manipulation

2.5.1 Import data into R

We mentioned formely that the easiest way to import your data is to click on it in the files pane. However, this manual step is useful if you have to import and analyse the data once, but probably you want to use your data next time as well. That is way it is a good idea to copy and paste the code for importing the data into your script.

In fact, if the data is in your working directory, you can refer to it with "**relative referencing**". This means that you have to type only the name of the file, not the full path, because R will automatically start to look for the file in the working directory[2].

```
library(readr)
df <- read_delim("da_q.csv", delim = ";", escape_double = FALSE, trim_ws = TRUE)
```

```
df <- read_delim(str_c(WD, "/data/da_q.csv"), delim = ";", escape_double = FALSE, trim_ws = TRUE)
#> Rows: 21 Columns: 11
#> -- Column specification --------------------------------------------------------
#> Delimiter: ";"
#> chr (9): What is your zodiac? (https://www.astrology-zodiac-signs.com/), Do ...
#> dbl (2): ID, How many countries have you been to so far?
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Now we have imported a tidy dataset. Each column is variable, and each row is an observation. Lets see how to select specific data from that. If you want to analyse only one column of it, you can use $ operator.

```
pizza <- df$`How many slices of pizza can you it at once?`

pizza
#>  [1] "8"
```

[2] you can download this example dataset from the GitHub page of this bookdown

```
#>  [2] "12"
#>  [3] "Depends on size. Can be up to 5 slices of the medium pizza"
#>  [4] "2"
#>  [5] "3"
#>  [6] "4"
#>  [7] "4"
#>  [8] "4"
#>  [9] "3"
#> [10] "2"
#> [11] "4"
#> [12] "3"
#> [13] "3"
#> [14] "2"
#> [15] "2"
#> [16] "4 and It depends how much I am hungry"
#> [17] "4"
#> [18] "3"
#> [19] "2"
#> [20] "6"
#> [21] "3"
```

The output `pizza` is a character vector currently, because some of the answers contain letters. We have to options here:

1. Using `as.numeric` function to force R using the values as numerical data.

```
as.numeric(pizza)
#> Warning: NAs introduced by coercion
#>  [1]  8 12 NA  2  3  4  4  4  3  2  4  3  3  2  2 NA  4  3  2  6  3
```

We got a warning message. Where letters appear R cannot convert the values to numbers, so this values became NA (Not Available) values.

2. Remove the letters from the answers and convert the vector to the correct datatype.

To manage this, we have to use the syntax called **regular expressions**. I want to show you 4 expressions now and a function. The function `gsub` will detect a given letter in a character and replace it with something. Lets see how!

```
gsub(x = "Awesome 12", pattern = "\\w", replacement = "B") # every non-white space
#> [1] "BBBBBBB BB"
gsub(x = "Awesome 12", pattern = "\\s", replacement = "B") # every white space
#> [1] "AwesomeB12"
gsub(x = "Awesome 12", pattern = "\\d", replacement = "B") # every digit
#> [1] "Awesome BB"
gsub(x = "Awesome 12", pattern = "\\D", replacement = "B") # every non-digit value
#> [1] "BBBBBBBB12"
```

So we can use the last example to solve our problem.

```
pizza_only_digits <- gsub(x = pizza, pattern = "\\D", replacement = "")

pizza_only_digits
#>  [1] "8"  "12" "5"  "2"  "3"  "4"  "4"  "4"  "3"  "2"  "4"  "3"  "3"  "2"  "2"
#> [16] "4"  "4"  "3"  "2"  "6"  "3"

as.numeric(pizza_only_digits)
```

```
#>  [1]  8 12  5  2  3  4  4  4  3  2  4  3  3  2  2  4  4  3  2  6  3
```

2.6 Conditional statements

We offen use conditional statement in programming. It has a clean concept: **If the condition is TRUE, then evaluate the following task.**

```
cat(
  '<iframe width="560" height="315" src="https://www.youtube.com/embed/m2Ux2PnJe6E" title="YouTube vide
)
```