Introducing R & R Studio

An introduction to R, day 1

Prof. Dr. Claudius Gräbner-Radkowitsch
Europa-University Flensburg, Department of Pluralist Economics
www.claudius-graebner.com | @ClaudiusGraebner | claudius@claudius-graebner.com

Europa-Universität
Flensburg

Europa-Universität
Flensburg
International Institute of Management
and Economic Education
Department of Pluralist Economics

Introduction and outline

Eine kurze Vorstellungsrunde…

- Wer bin ich?
- Was mache ich an der EUF?
- Was sind meine Ziele mit dem Kurs?

Workshop Ausblick

Tag 1: Einleitung und Ausblick

- Installationsprobleme lösen ✓
- Kurze Vorstellung und Erwartungsmanagement ✓
- Grundlagen der Programmiersprache R
- Projekt-Management
- Datentypen

Tag 2: Daten einlesen und aufbereiten

- Importieren von “echten” Daten
- Konzept der “tidy data” und der Analyse Workflow
- Data wrangling und data manipulation

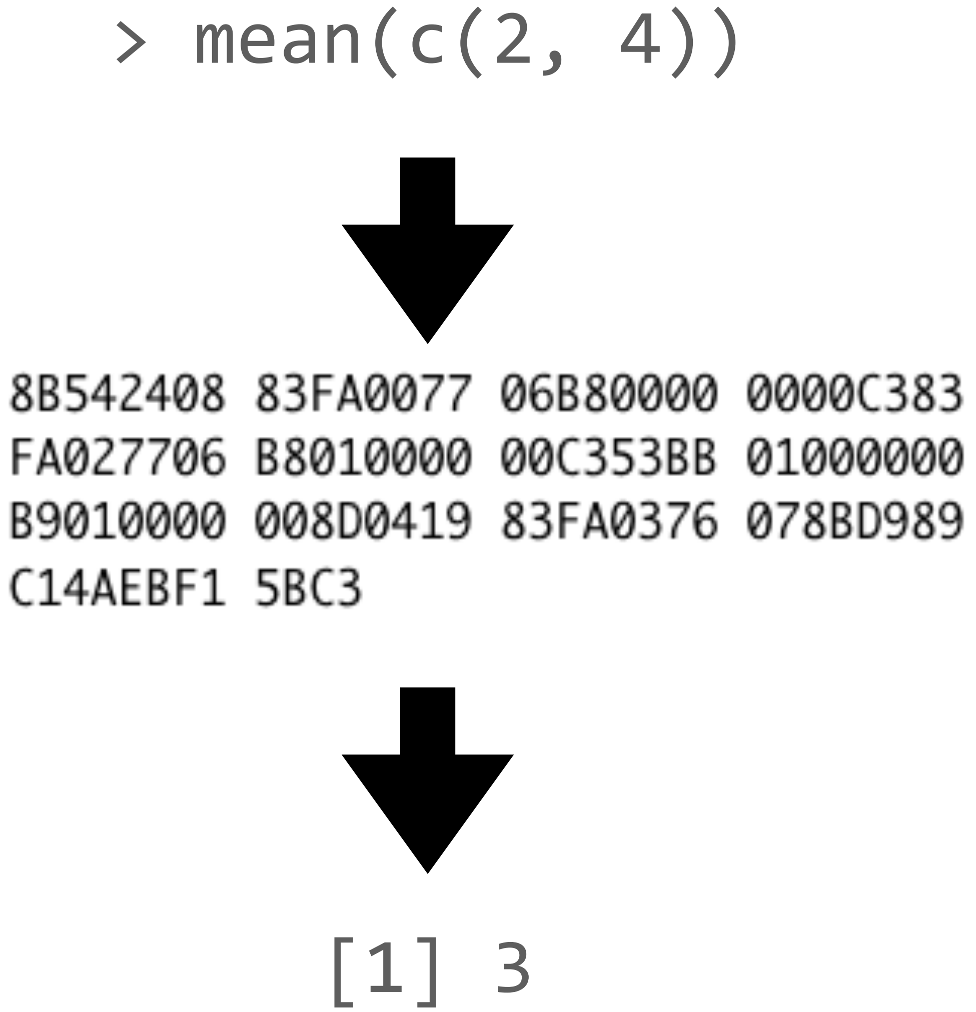
Tag 3: Daten visualisieren

- Visualisierungstheorie
- Generelles Vorgehen in R
- Konkrete Anwendungsfälle (aus euren Bereichen)

Basics about R

R and R-Studio

- R is a **programming language**
  - A language to issue commands to your computer:

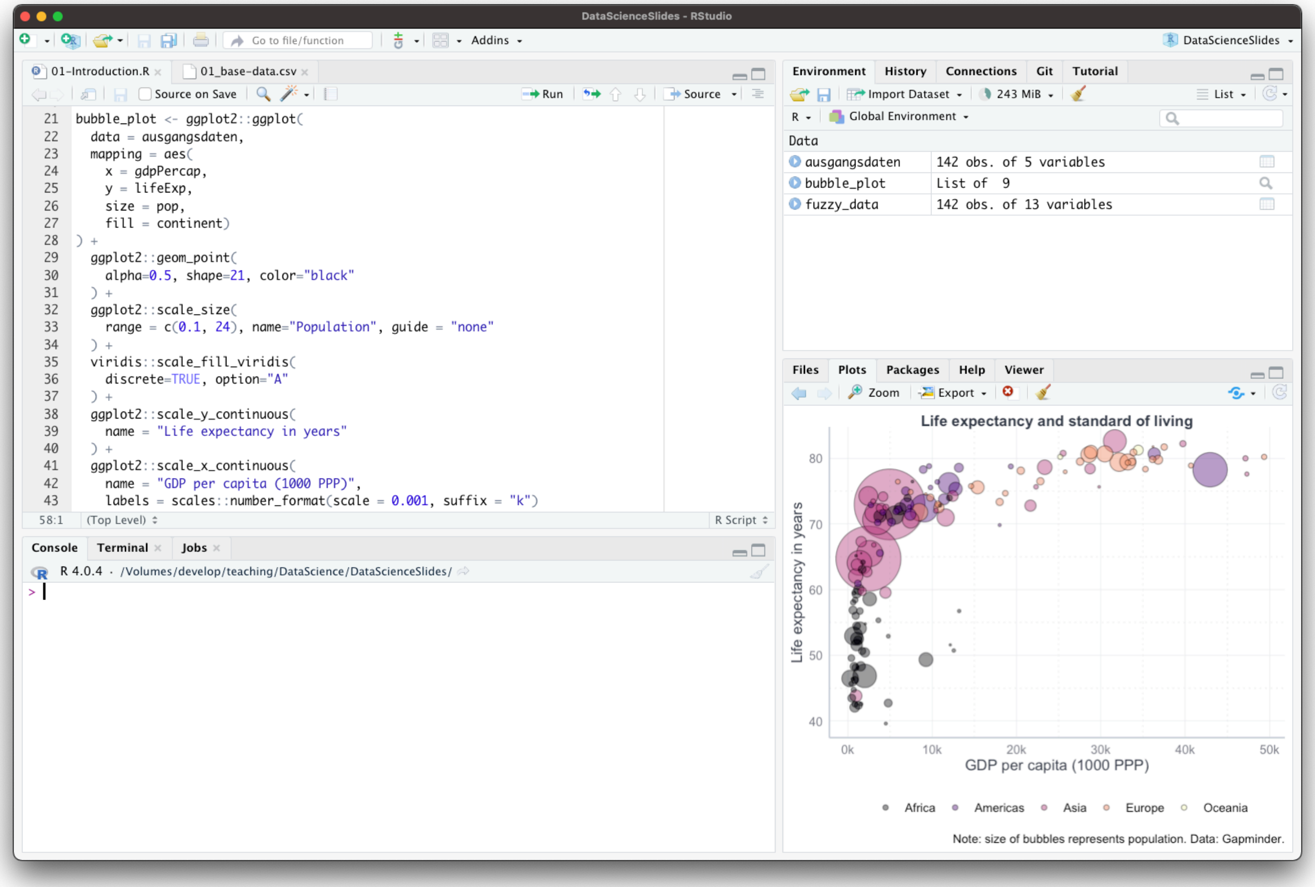


```
> mean(c(2, 4))
```

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD989
C14AEBF1 5BC3
```

```
[1] 3
```

- R-Studio is an **integrated development environment**
  - Basically a fancy text editor with additional features that make programming easy



```r
bubble_plot <- ggplot2::ggplot(
  data = ausgangsdaten,
  mapping = aes(
    x = gdpPercap,
    y = lifeExp,
    size = pop,
    fill = continent)
) +
  ggplot2::geom_point(
    alpha=0.5, shape=21, color="black"
  ) +
  ggplot2::scale_size(
    range = c(0.1, 24), name="Population", guide = "none"
  ) +
  viridis::scale_fill_viridis(
    discrete=TRUE, option="A"
  ) +
  ggplot2::scale_y_continuous(
    name = "Life expectancy in years"
  ) +
  ggplot2::scale_x_continuous(
    name = "GDP per capita (1000 PPP)",
    labels = scales::number_format(scale = 0.001, suffix = "k")
```

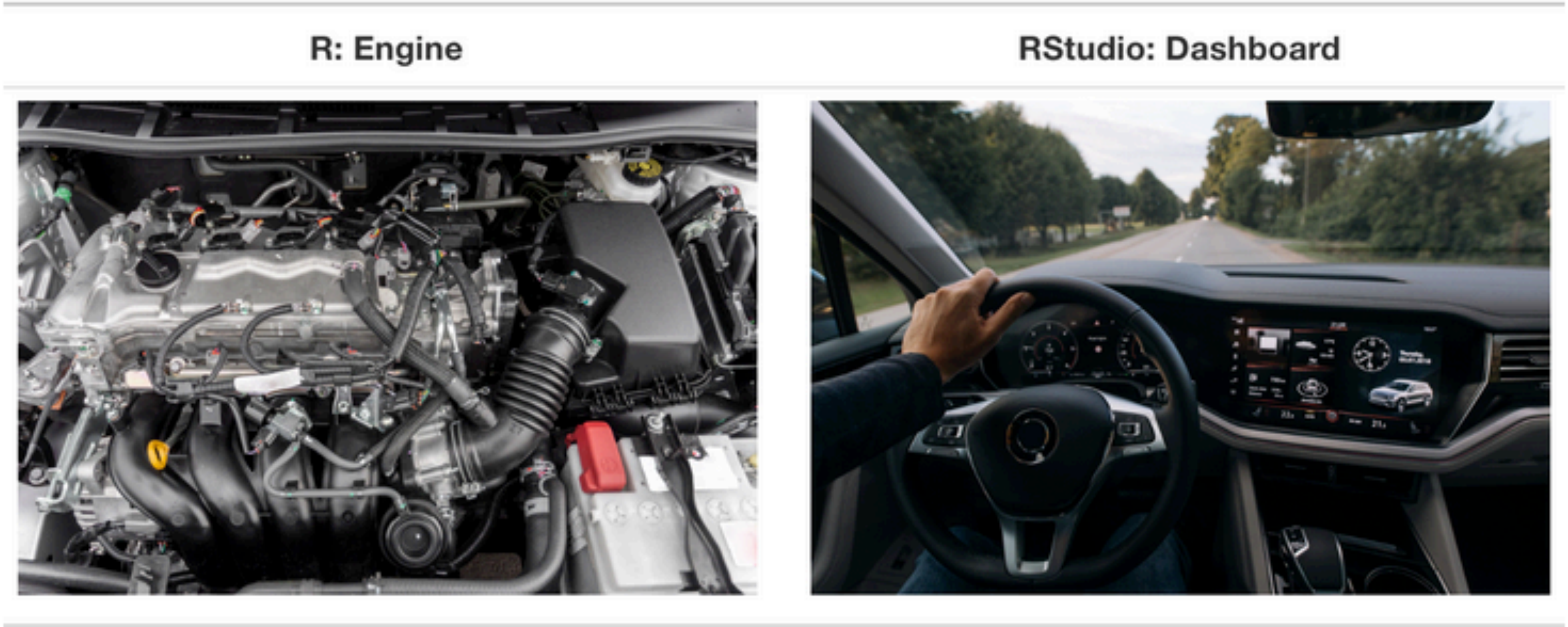
```
R 4.0.4 · /Volumes/develop/teaching/DataScience/DataScienceSlides/
>
```

Data	
ausgangsdaten	142 obs. of 5 variables
bubble_plot	List of 9
fuzzy_data	142 obs. of 13 variables

Note: size of bubbles represents population. Data: Gapminder.

R and R-Studio

- R is a programming language
- R-Studio is an integrated development environment



R: Engine

RStudio: Dashboard

Figure: Ismay & Kim (2022)

- You need to install R first, then you can install R Studio
- After that, you basically only use R Studio → it calls R whenever necessary

R and R packages

- If you install R, you can issue a lot of commands that your computer immediately understands

- **R packages**: a collection of variables and functions written by others that you can install on your computer and use them

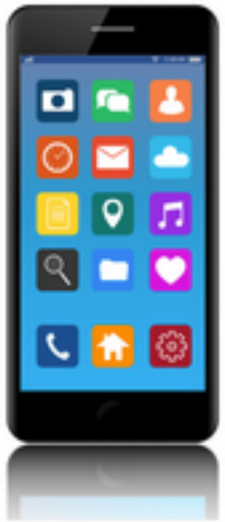
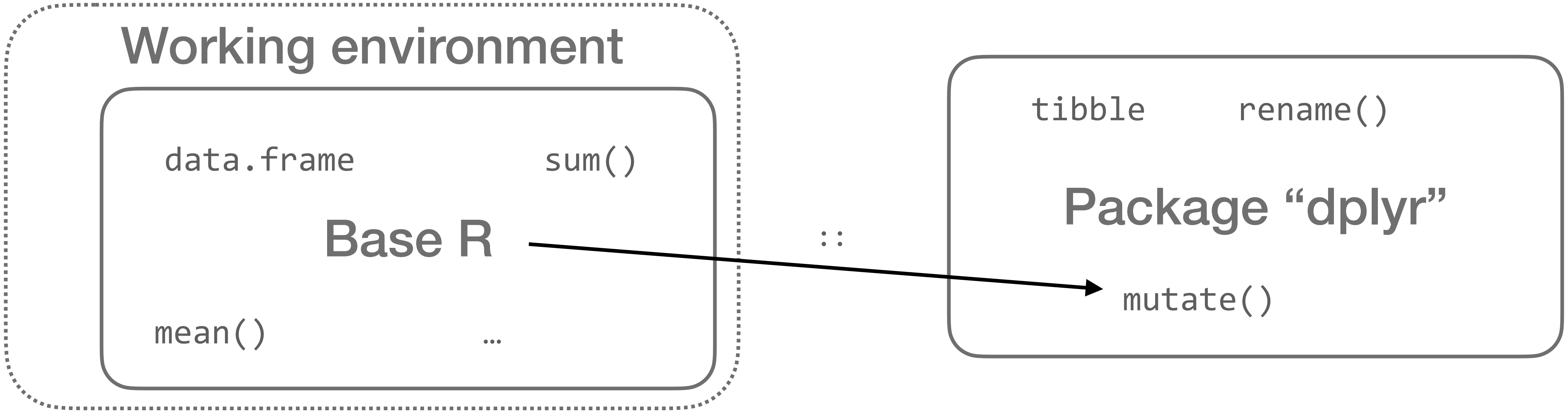
R: A new phone	R Packages: Apps you can download
	Available on the App Store ANDROID APP ON Google play

Figure: Ismay & Kim (2022)

R and R packages

- If you install R, you can issue a lot of commands that your computer immediately understands

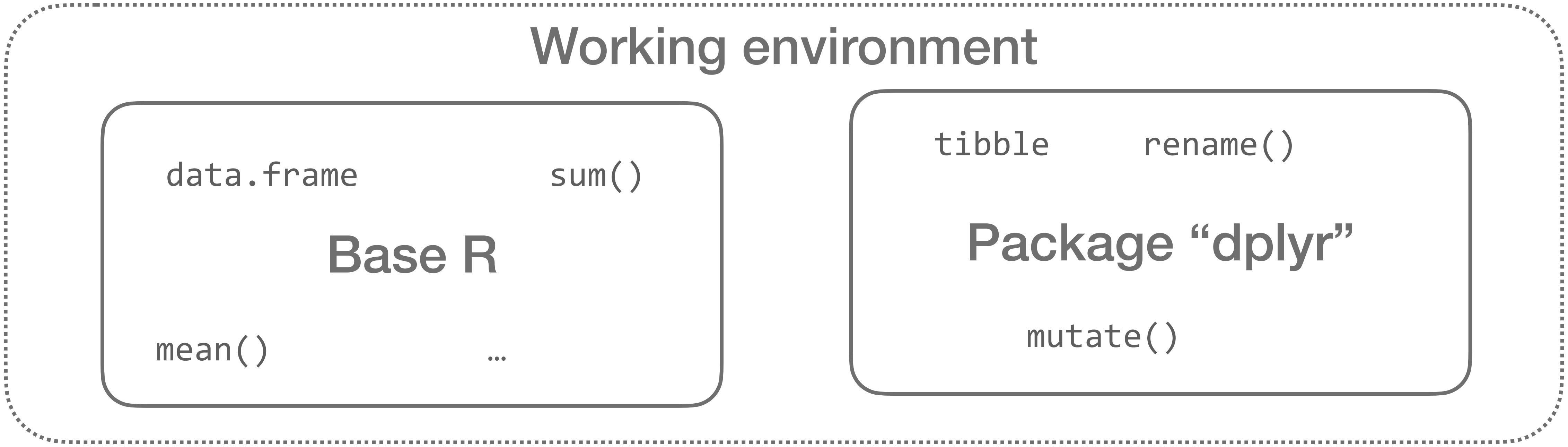
- **R packages**: a collection of variables and functions written by others that you can install on your computer and use them



- Once a package is installed, access its “namespace” via `::`
  - `dplyr::mutate()` uses the function `mutate` from package `dplyr`
- Alternative: attach a package via `library()`

R and R packages

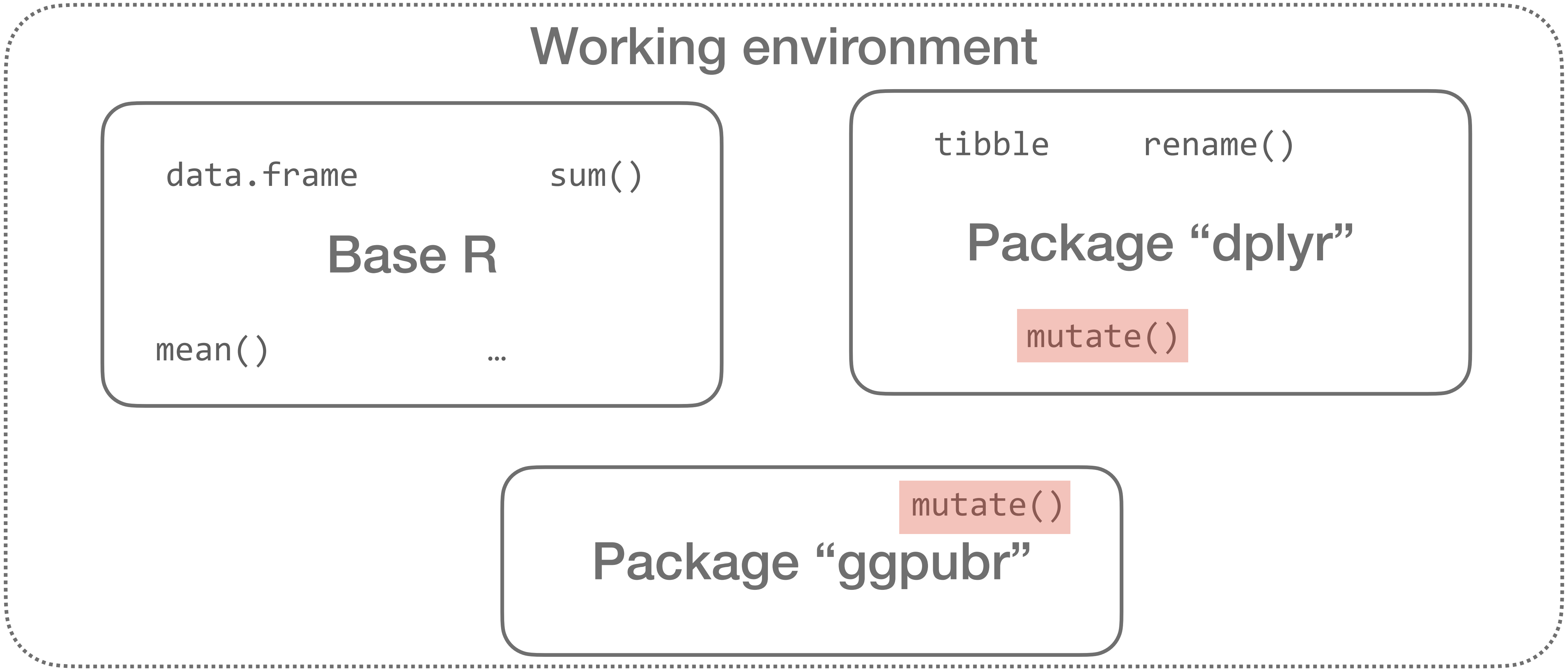
- Using `library(dplyr)` attaches the package to your environment



- Once a package is attached, you can use its objects directly
  - `mutate` is then equivalent to `dplyr::mutate()`
- But be aware of "masking" problems!

R and R packages

- Using `library()` can lead to masking issues



- You receive warnings about masking when attaching packages
  - Then you need to use the `::` notation

Dialects of R

- As natural languages, R has **dialects**
  - There are different “styles” of programming to achieve the same thing
  - Differ in terms of the functions and data types used
- This can be confusing when searching for solutions online
- Strong recommendation: **stick to one dialect** and remain consistent

Base R

- The “original”
- Central role of `data.frames`

Tidyverse

- Update of base R with a modern and consistent syntax
- Central role of `tibbles`

Data.Table

- Developed for big data
- Central role of `data.tables`
- Fast, but a bit advanced

Central take-aways

- **R** is a high-level programming **language**
- **R-Studio** is a fancy **editor** ("IDE") → you always use R-Studio
- **R packages** expand what you can do by supplying additional functions, objects, and data
  - Use `::` to "build a connection" to installed packages, or attach them with `library()`
- R has different **dialects** → try to be consistent
  - Here we stick to the **tidyverse**

Questions?

Using R & R Studio

Exercises I

- Conduct the following operations in R by assigning the result to a variable and then calling it by its name:

1. $4 + 8$
2. $-20 \cdot 3$
3. $(5 \cdot 3)^2$
4. $\frac{8^2 + 5^4}{3}$
5. $\sum_{i=1}^{10} i$

Project management

The goal

- Learn about a **default directory structure** and a general way to **document everything** you do in your project

  → Facilitates the collaboration with future-you considerably

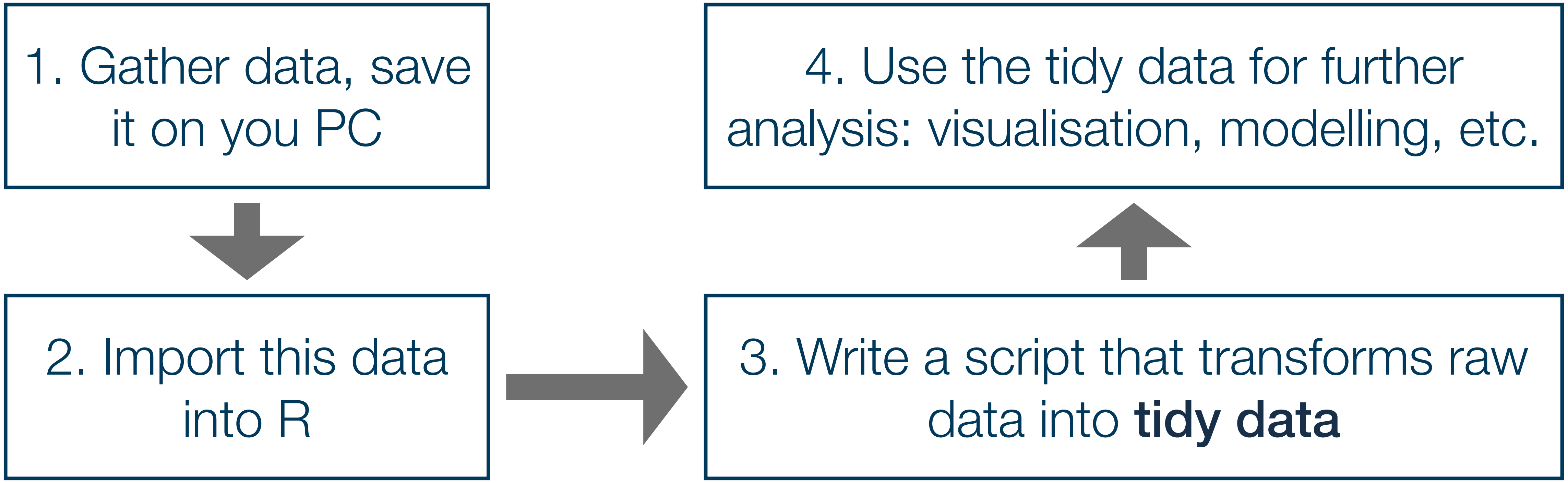
  ☝️Nothing is worse than hating your past-you for not documenting correctly where data came from, or how it has been prepared 🫣

- Central idea: all results must be **reproducible** from the raw data at any time
  - This implies that you **must not manipulate your raw data** at any cost

Introduce general workflow to avoid most editable problems in the context of project management

How to keep your work transparent

- Raw data must not be changed, but is usually not in a state we can work with 🤨



1. Gather data, save it on you PC

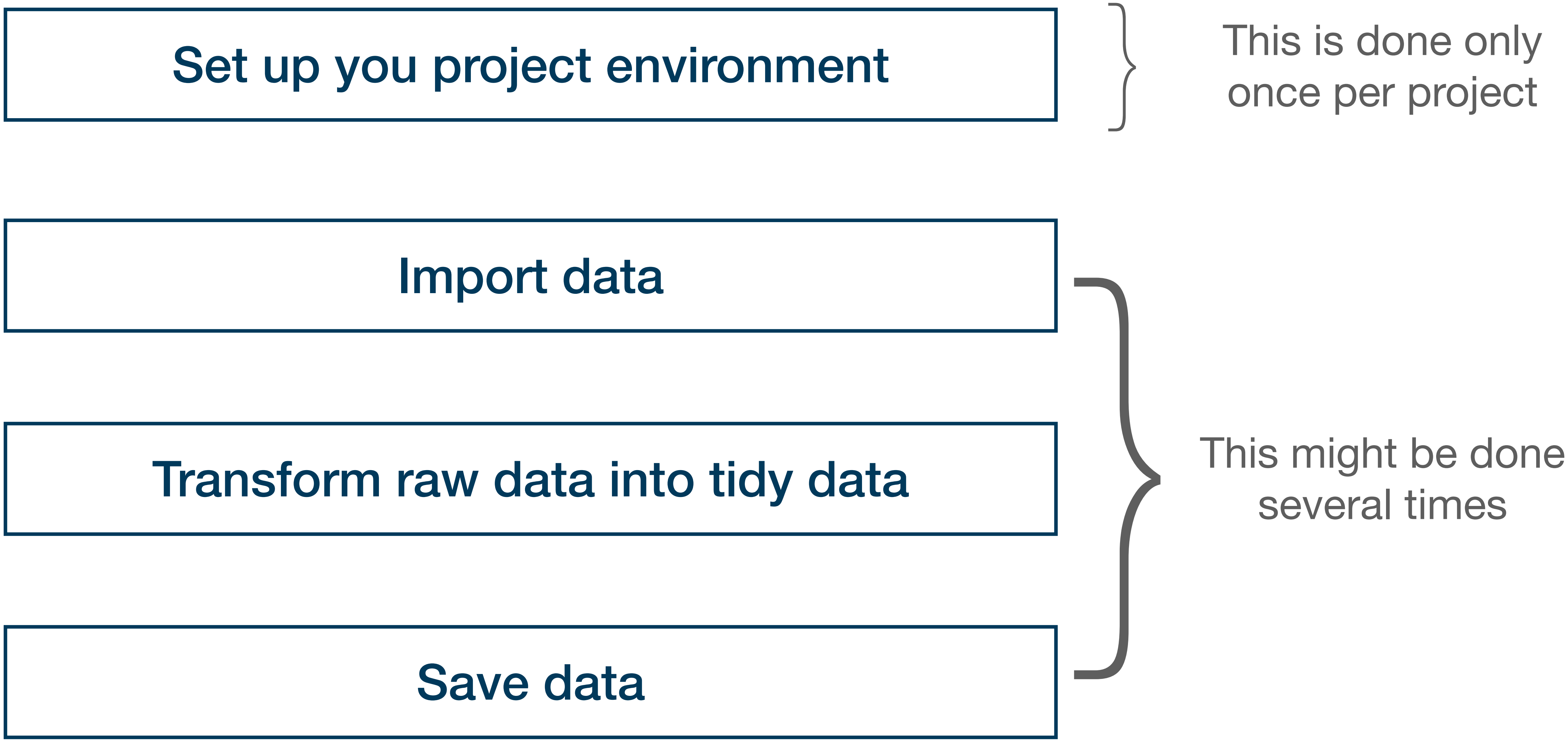
2. Import this data into R

3. Write a script that transforms raw data into **tidy data**

4. Use the tidy data for further analysis: visualisation, modelling, etc.

- Saving the scripts in steps 2 & 3 makes your work **fully reproducible**
- By looking into the script you will always know what you did to your raw data → you can also heal basically every mistake you made, not harm done!

Outlook



Set up you project environment

This is done only once per project

Import data

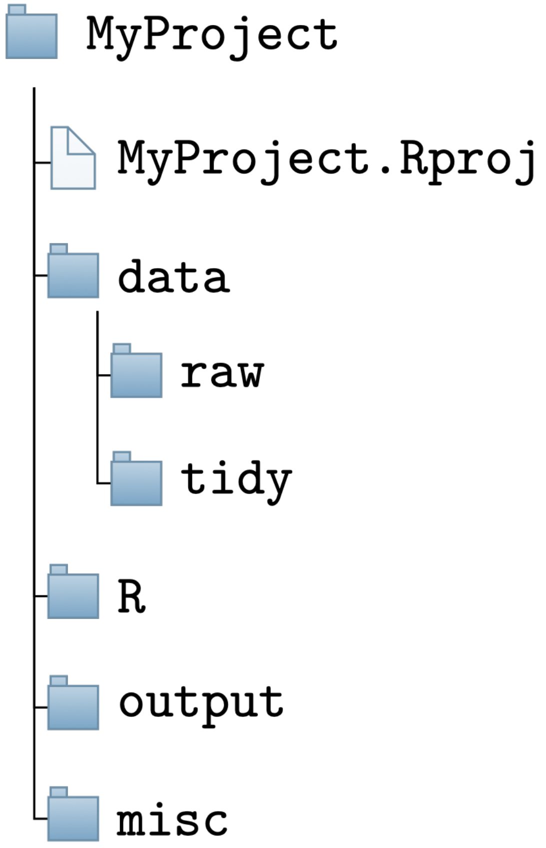
Transform raw data into tidy data

Save data

This might be done several times

Setting up your working environment

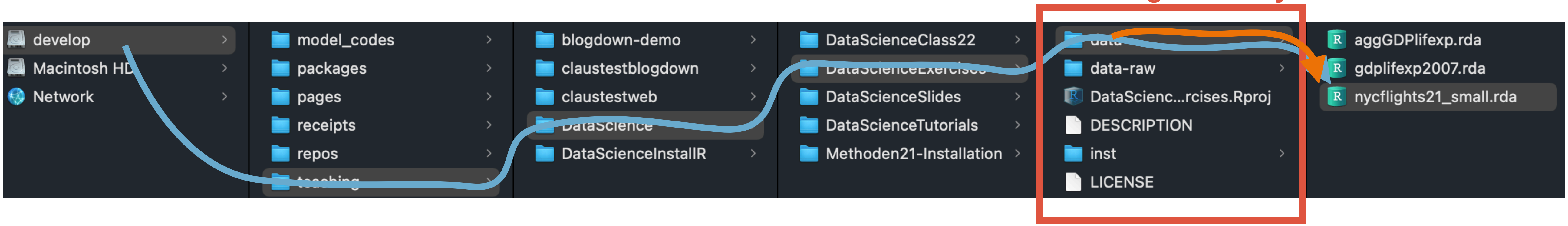
- Before we talk about importing raw data we need to discuss where the raw data should actually be saved
- A prerequisite for a transparent, reproducible, and easy-to-work-with project is the right **directory structure**
- Thus, for every task in R you should set up your project like this:
- All the relevant steps to set this up, and the rationality for this structure are described in the respective tutorial



```
MyProject
├ MyProject.Rproj
├ data
│ ├ raw
│ └ tidy
├ R
├ output
└ misc
```

Paths and the here-package

- There are two ways in which you tell your computer where a certain file is located:
  - Via an absolute path: description starts at the root directory 🌲
  - Via a relative path: description starts at your current position in the file system


Working directory

- Assuming we are 'located' in the folder `DataScienceExercises`: and want to point to the file `nycflights21_small.rda`:
  - `"/Volumes/develop/teaching/DataScience/DataScienceExercises/data/nycflights21_small.rda"`
  - `"data/nycflights21_small.rda"`

Relative paths and setwd()

- The relative path seems nicer…
  - Its shorter 😇 and you can share code without forcing others to adjust the path
- Problem: how to set our location to the directory `DataScienceExercises`?
- We can do this using `setwd()`, providing the absolute path to `DataScienceExercises` as an argument:

  -
    ```
    setwd("/Volumes/develop/teaching/
          DataScience/DataScienceExercises")
    ```
  - Then we can use `"data/nycflights21_small.rda"`
- Many people put `setwd()` at the top of their scripts
  - **BUT YOU MUST NEVER EVER DO THIS!!!!!!!!!!!!!!!!!!!!!!!!!!!**

Why setwd() is evil and not to be used

- You should never ever use `setwd()` in your scripts
- First, it does not help avoiding absolute paths because you have to provide an absolute path to `setwd()` 🤯
- Second, it makes people hate you:

Abby writes amazing_script.R 👩🏼‍💻

```
setwd("/Volumes/Macintosh HD/Users/AbbysUserName/
          PathToFolderThatOnlyExistsHere/ProjectName”)
data_file <- data.table::fread(“data/file.csv”)
```

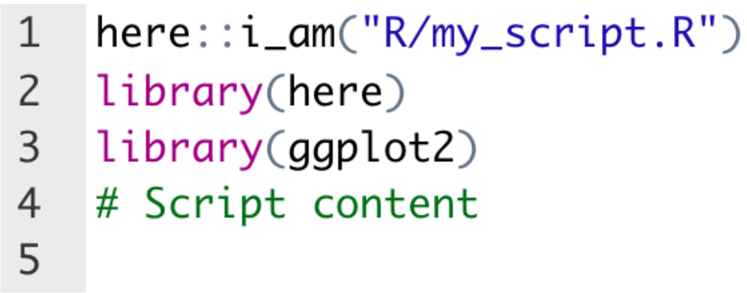
Sends file to Ellie 📧

Ellie opens file and executes it 😀

```
> setwd("/Volumes/Macintosh HD/Users/AbbysUserName/PathToF
olderThatOnlyExistsHere/ProjectName/file.txt")
Error in setwd("/Volumes/Macintosh HD/Users/AbbysUserName/
PathToFolderThatOnlyExistsHere/ProjectName/file.txt") :
  cannot change working directory
```

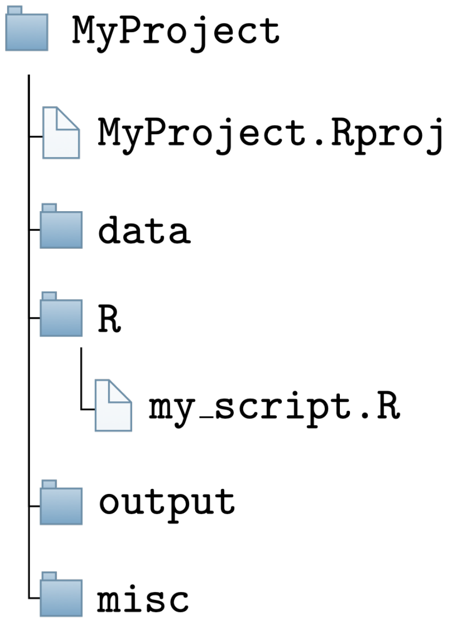
The better alternative to setwd() is `here`

- Thankfully, there is a very simple solution: the package **here**
- It allows you to set an anchor ⚓ in you project directory
- Then you can create paths relative to this anchor using the function `here::here()`
  - These commands will always work on every machine
- Always put `here::i_am()` into the first line of your scripts
  - As an argument, provide the location of the script relative to the project root
- From now on, only provide paths relative to this root using `here::here()`



```r
here::i_am("R/my_script.R")
library(here)
library(ggplot2)
# Script content

```



MyProject

MyProject.Rproj

data

R

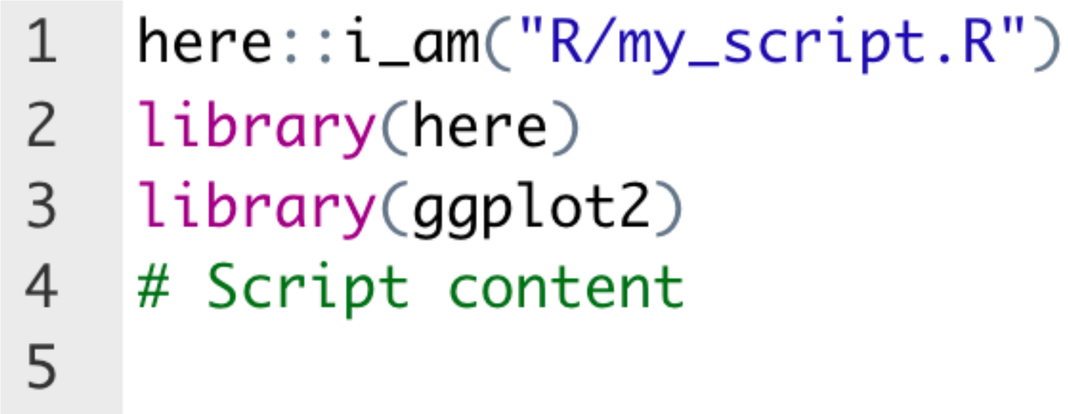
my_script.R

output

misc

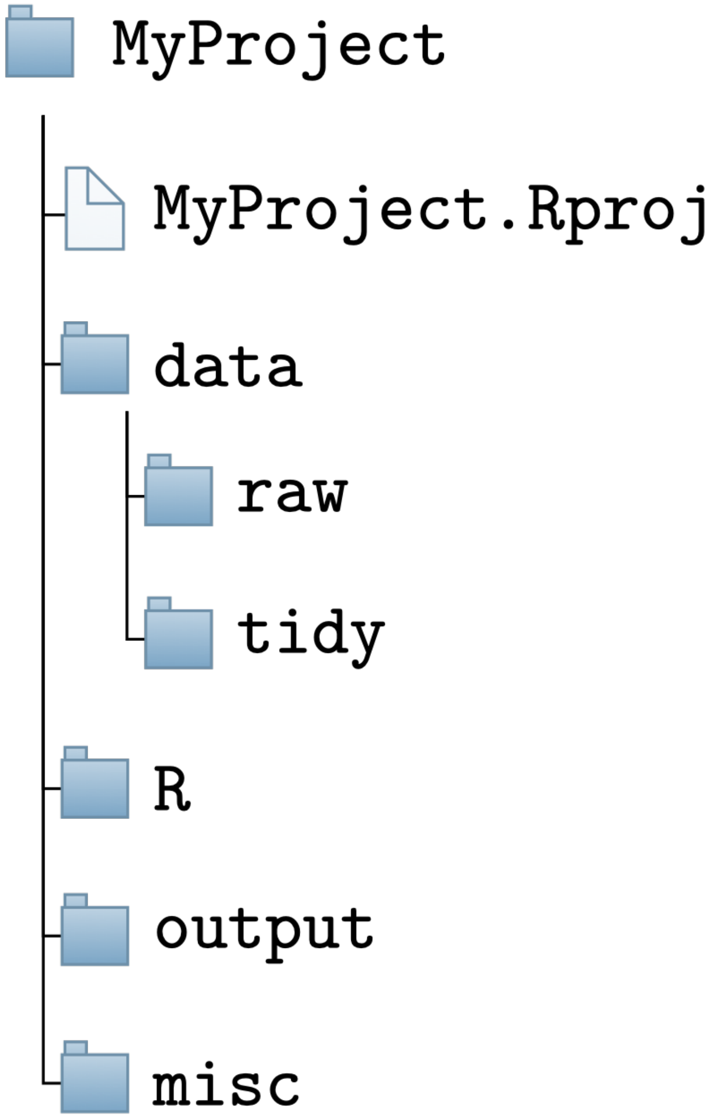
Exercises II

- Create a new R-Project on your computer
- Create all the required folders
- Write an R script, put it into the right directory, and make it usable for the `here`-package
- Check out what the function `here::here()` returns and experiment with its use



```r
here::i_am("R/my_script.R")
library(here)
library(ggplot2)
# Script content

```



MyProject

MyProject.Rproj

data

raw

tidy

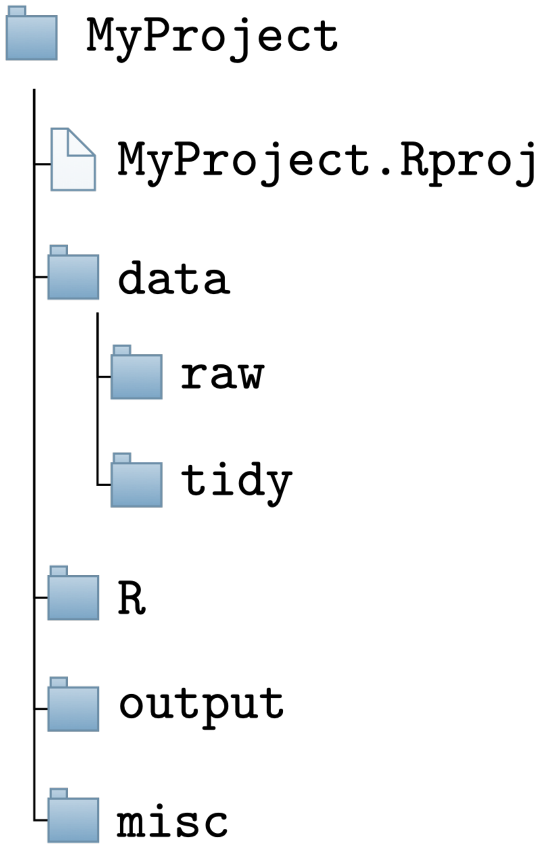
R

output

misc

Central take-aways

- For each self-contained endeavour you should create an **R project**
- Always set up a **clear folder structure**
- Keep your raw data separate and never change it
- **Document** your scripts to help you and others understand them
- Always use the **here** package when you import or save objects



MyProject

MyProject.Rproj

data

raw

tidy

R

output

misc

Object types

Basic object types in R

> “ To understand computations in R, two slogans are helpful:
> Everything that exists is an object.
> Everything that happens is a function call.
>
> John Chambers

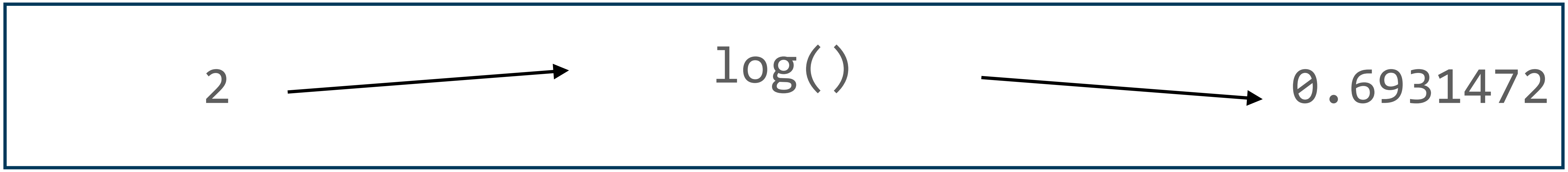
- The operation `2 + 3` refers to three objects:
  - The numbers `2` and `3`, as well as the function `+` (addition)
  - It executes the addition function and produces a further object: the number `5`
- A function is an algorithm, which takes an **input**, applies a **routine**, and returns an **output**

Input → Function routine → Output

Basic object types in R

> “To understand computations in R, two slogans are helpful:
> Everything that exists is an object.
> Everything that happens is a function call.
>
> John Chambers

- The operation `2 + 3` refers to three objects:
  - The numbers `2` and `3`, as well as the function `+` (addition)
  - It executes the addition function and produces a further object: the number `5`
- A function is an algorithm, which takes an **input**, applies a **routine**, and returns an **output**



2

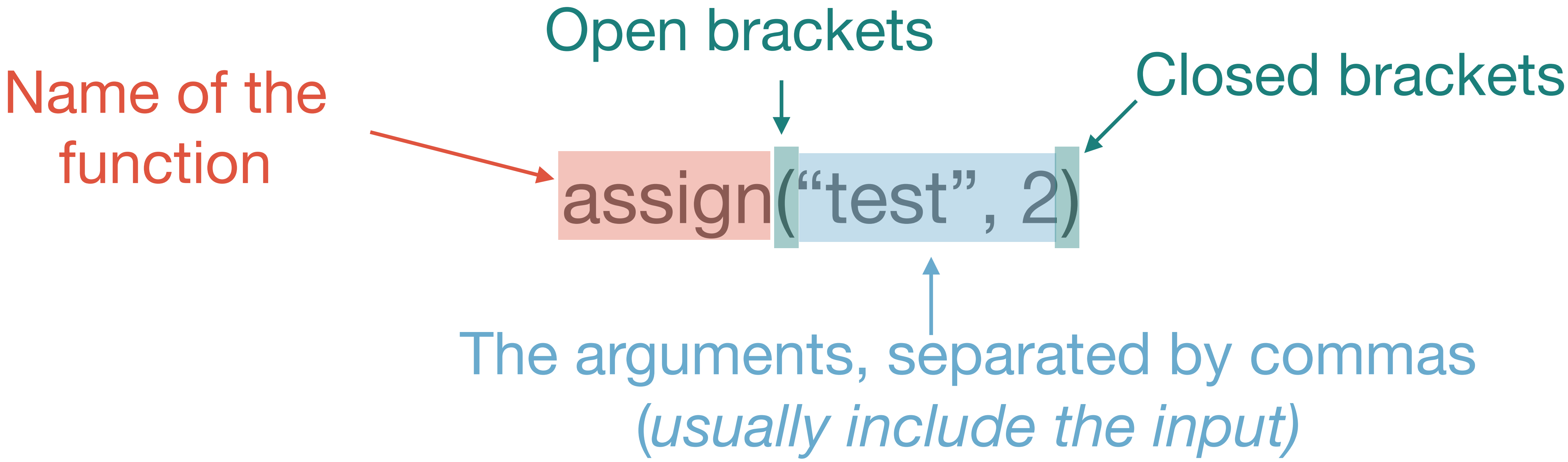
log()

0.6931472

Functions

Calling functions

- The most common form is the **prefix** form:



Name of the function

Open brackets

Closed brackets

assign("test", 2)

The arguments, separated by commas *(usually include the input)*

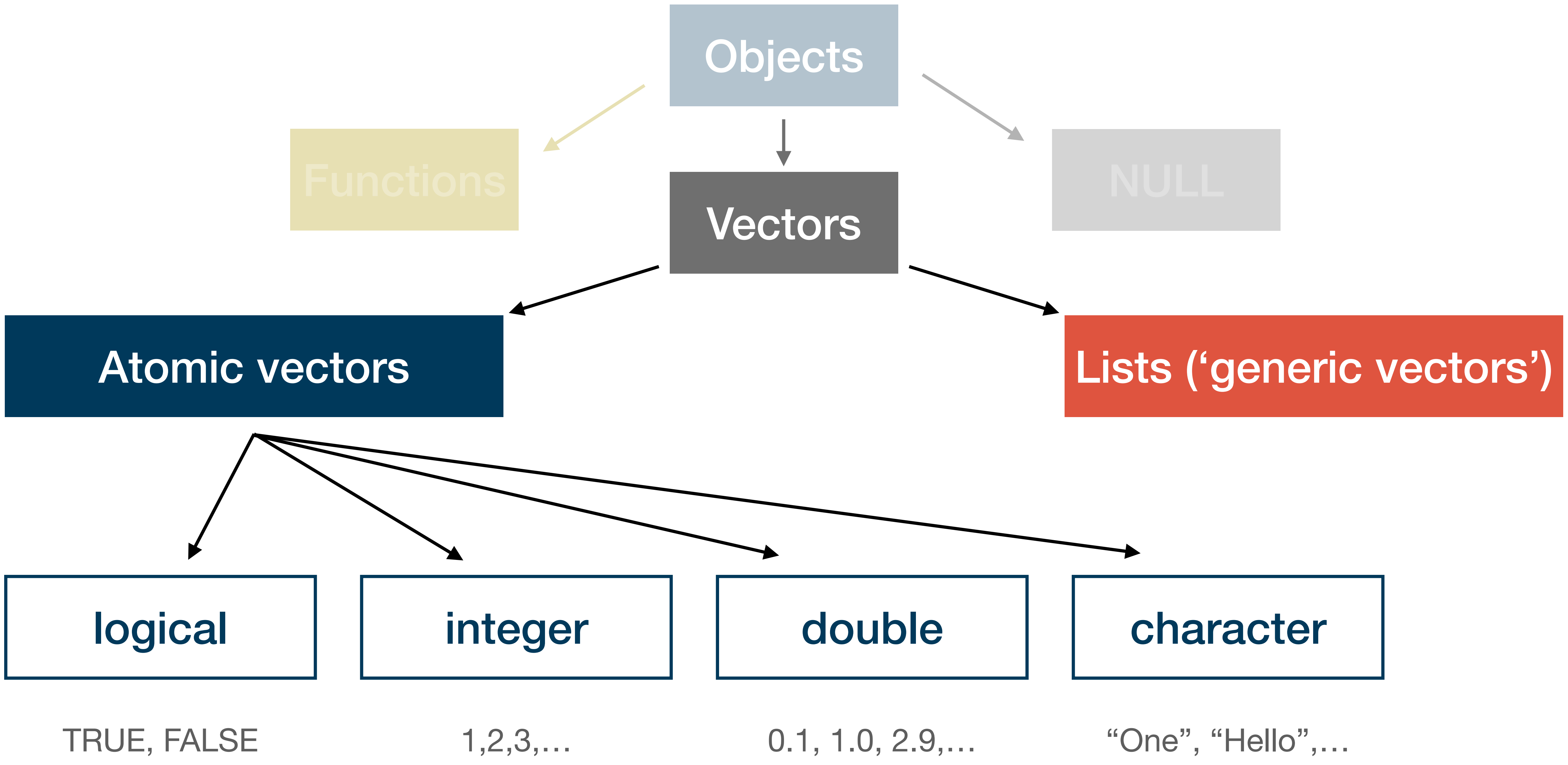
- You need to distinguish **mandatory** and **optional** arguments

```
t_vec <- c(1, 2, 3, 4, NA)
```

A) `mean(t_vec)` vs. `mean(x=t_vec)`

B) `mean(t_vec)` vs. `mean(t_vec, na.rm=TRUE)` vs. `mean(t_vec, na.rm=FALSE)`

Basic object types in R



Objects

Functions

Vectors

NULL

Atomic vectors

Lists (‘generic vectors’)

logical

integer

double

character

TRUE, FALSE

1,2,3,…

0.1, 1.0, 2.9,…

“One”, “Hello”,…

Atomic vectors

- Atomic vectors are composed only of objects of the same type
  - We say that an atomic vector is of the same type as are its elements
  - We can test for this type using the function `typeof()`
- There are four main types of atomic vector that are most important:

Logical values: logical

- Only two* options: `TRUE` or `FALSE`
- Often the result of logical operations (e.g. `4>2`)

Whole numbers: integer

- A whole number, followed by L:
- `1L, 2L, 100L`, etc.
- Often the result of counting

Decimal numbers: double

- A number with the decimal sign `.`
- `2.0, 0.8, -7.5`, etc.
- The ‘standard’ number you will use

Letters and words: character

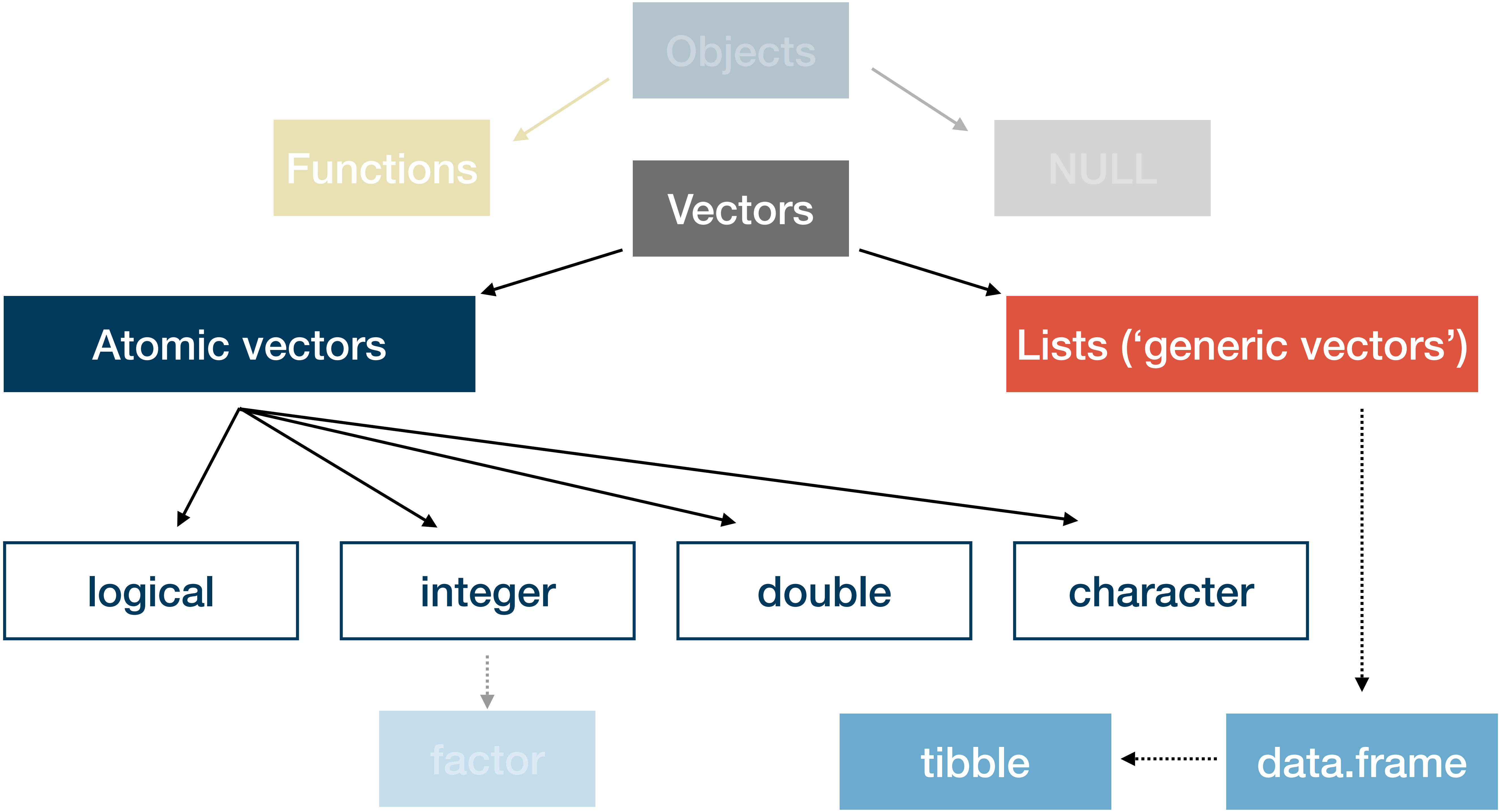
- Might contain all kinds of tokens and start and end with `"`
- `"2", "Hello!", “vec_1"`, etc.

*: We will see later that missing values are also considered logical in some instances, but this is basically irrelevant now.

Exercises III

1. Create a vector containing the numbers 2, 5, 2.4 and 11.
2. What is the type of this vector?
3. Transform this vector into the type `integer`. What happens?
4. Do you think you can create a vector containing the following elements: `"2"`, `"Hallo"`, `4.0`, and `TRUE`? Why? Why not?

Advanced object types in R



Exercises IV

- Create a data frame with two columns, one called `"nb"` containing the numbers `1` to `5` as double, the other called `"char"` containing the numbers `6` to `10` as character

- Transform this data frame into a `tibble`!

- Extract the second column of this `tibble` such that you have a vector

Wrap-up day 1

Key take aways from day 1

- R is a high-level programming language with dialects, R Studio an IDE
- R packages as a way to expand the capabilities of base R
- For your projects use R projects, a clear folder structure and the here package
- "Everything that exists is an object, everything that happens is a function call."
- Four basic object types, advanced ones as modifications of the basic types

Check the course website for…

- Suggestions to recap this day
- Necessary preparations for day 2
- A link to the course forum for questions