R Projects and data import

24.03.2020, Data Science (SpSe 2022): T6

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

Prologue:

Prologue
Feedback and exercises

- XX of you filled out the feedback survey. Main take-aways:
  - TBA
- What were the main problems with the exercises?

Goals for today

I. Learn how to set up an R project

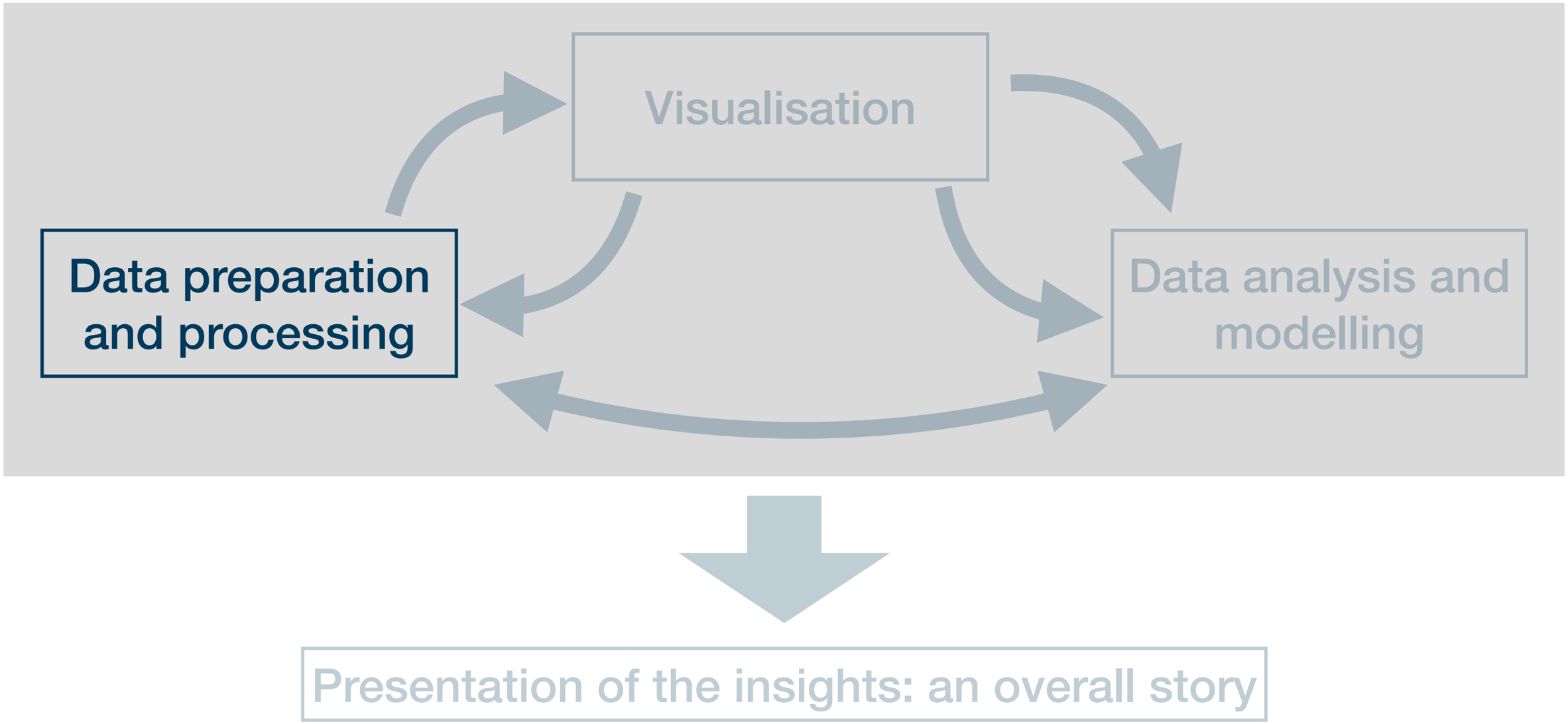
II. Learn how to use the here package and import data

III. Basic data wrangling, but noch all: next session

Data wrangling in R

The role of data preparation

- Importing and preparing is the most fundamental task in data science
  - It is also largely under-appreciated 🙄



Data wrangling is an essential skill

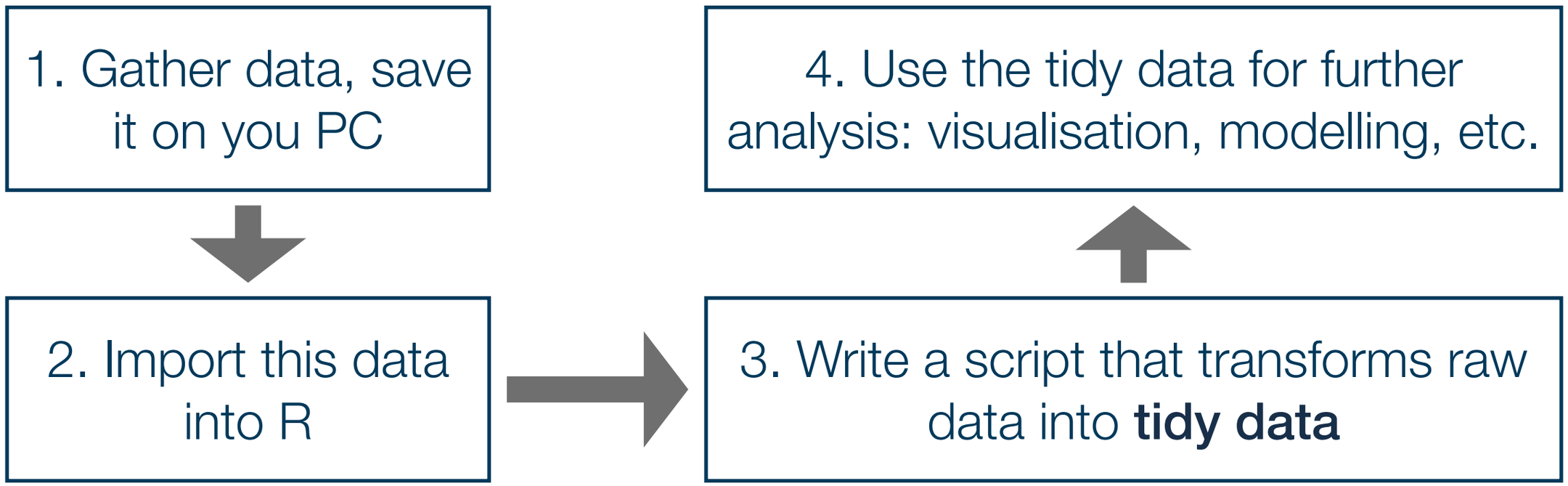
- According to several surveys, data scientists usually spend **about 80% of their working time** with importing and preparing data
- At the same time, few people learn how to do it properly
- Although it might sound a bit boring in the first place, few skills will
  - …save so much time
  - …save you so much nerves and frustration
  - …help you making so many new friends
- …in the medium run as skills in data wrangling!

Our goal

- The goal:
  - This will also facilitate the collaboration with future-you considerable
  - Nothing is worse than hating your past-you for not documenting correctly where data came from, or how it has been prepared
- Here we will learn a general workflow that, once mastered, helps you to avoid all editable problems **with certainty**
- A central idea is that all your research results must be **reproducible** from the raw data at any time
  - This implies that you **must not manipulate your raw data** at any cost
  - Raw data is what you download from the internet, gather through an experiment, or code yourself

How to keep your work transparent

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



- Saving the scripts in step 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

- The remainder will be organised as follows:

Set up you project environment

This is done only once per project

Import data

Transform raw data into tidy data

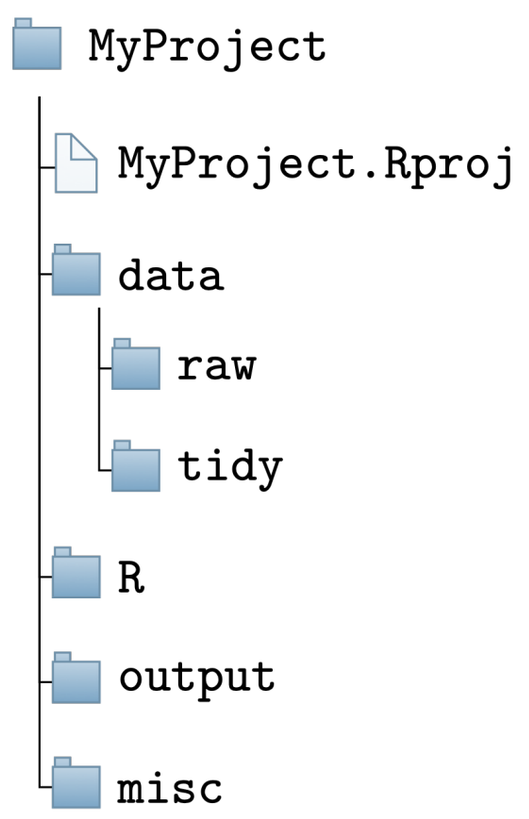
Save data

This might be done several times

Setup your R project

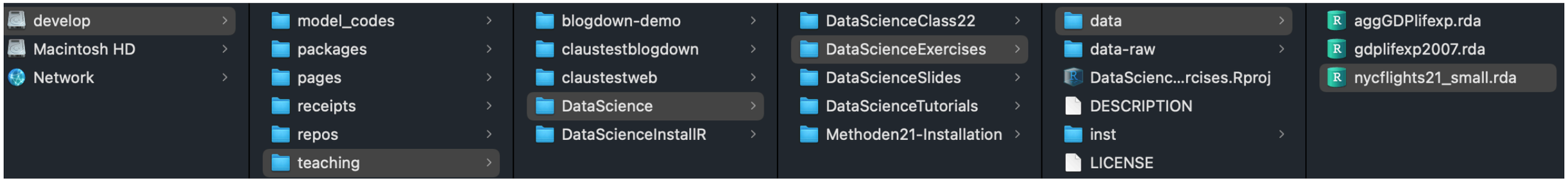
Setting up your working environment

- 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



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



- 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 📧

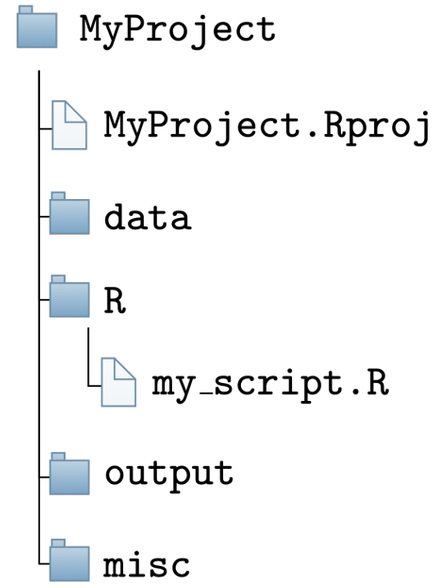
Ellie opens file and executes it 😀

```
> 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`

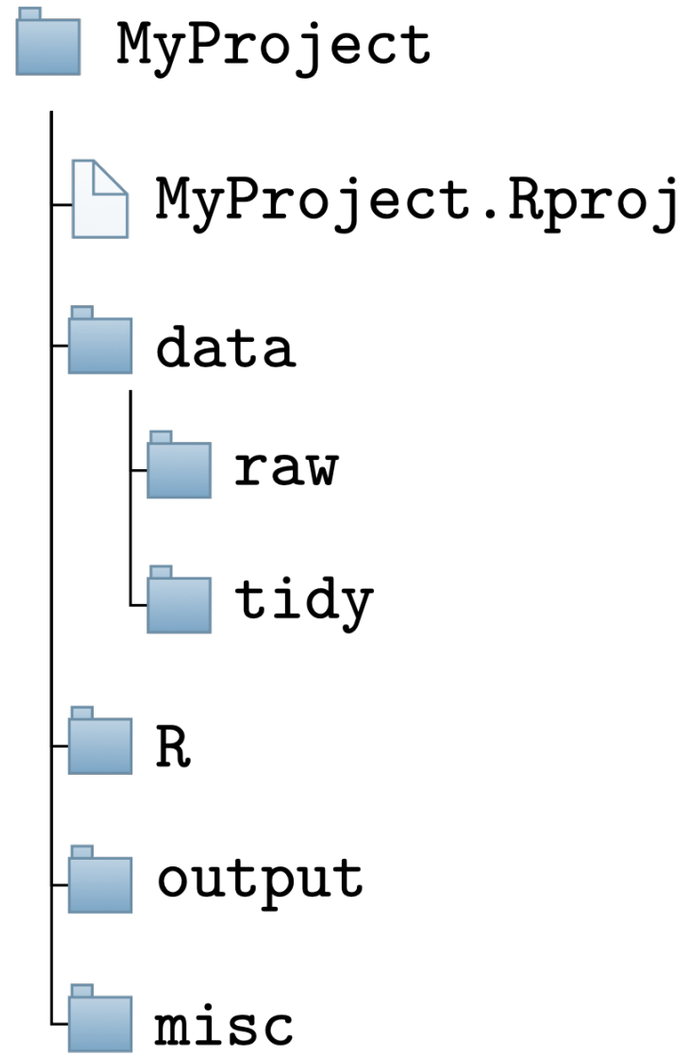
- Thankfully, there is a very simple solution: the package **here**
- It allows you to set an anchor ⚓ in you project directory
- Then you can create paths relative to this anchor using the function `here::here()`
  - These commands will always work on every machine
- Always put `here::i_am()` into the 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()`

```
1  here::i_am("R/my_script.R")
2  library(here)
3  library(ggplot2)
4  # Script content
5
```



Your turn

- 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



Importing data

Import functions

- Now that we have set up the project environment we can import data
- In the following we will assume that you raw data is stored in the folder `data/raw`
- The function we use to import a data set depends on the file type:

csv/tsv files	.Rds/RData files	Specific formats

`data.table::fread()`	`readRDS()`
	`load()`
		`haven::read_dta()`
		`haven::read_sas()`
		`haven::read_spss()`

- Basic procedure the same in all cases → focus on reading csv files here

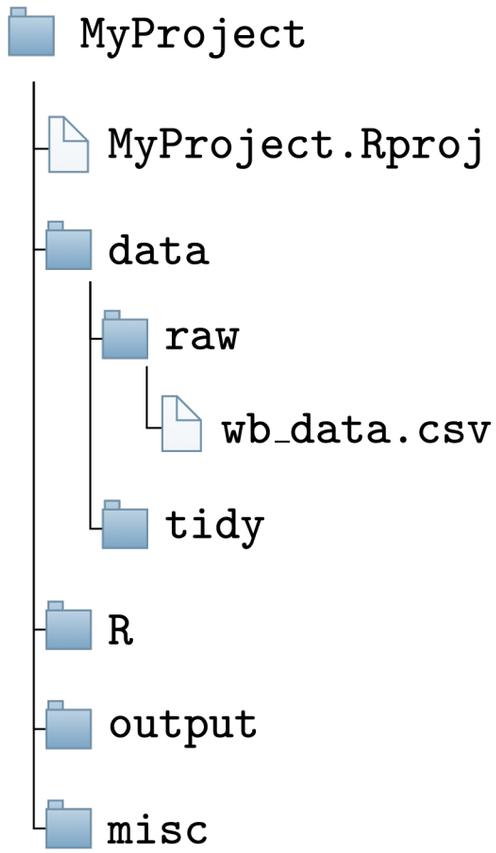
How to import data

- Good practice: save path to file in a vector:

```
data_path <- here("data/raw/wb_data.csv")
```

- Since its a csv file we use `data.table::fread()`:

```
data.table::fread(file = data_path)
```



- This uses default options to import the file
  - Works often for clean data files
  - But for the sake of transparency and since data files are often not clean, we should specify several optional arguments

- In contrast to the exposition in Wickham and Grolemund (2022) I strongly recommend using **`data.table::fread()`**

  → See tutorial for documentation

Your turn

- Download the example file from the course homepage
- Write a script that imports the data into your session

Tidy data

The goal: tidy data

> “Tidy datasets are all alike, but every messy dataset is messy in its own way.
>
> Hadley Wickham

- Translation into plain English:
  - We find data sets in all kind of ***-up forms in the world
  - We must turn them into a form that’s a good starting point for any further tasks
- Good thing: this form is unique – and its called **tidy**

The goal: tidy data

Every **column** corresponds to one and only one **variable**

Every **row** corresponds to one and only one **observation**

```
# A tibble: 4 × 4
  c_code  year exports unemployment
  <chr>  <int>   <dbl>        <dbl>
1 AT      2013    53.4         5.34
2 AT      2014    53.4         5.62
3 DE      2013    45.4         5.23
4 DE      2014    45.6         4.98
```

Every **cell** corresponds to one and only one **value**

- Every data set that satisfies these three demands is called tidy
- Excellent start for basically any further task – but maybe not the best way to represent data to humans

The goal: tidy data

Every **row** corresponds to one and only one **observation**

Every **column** corresponds to one and only one **variable**

Every **cell** corresponds to one and only one **value**

```
# A tibble: 2 × 4
  c_code variable     `2013` `2014`
  <chr>  <chr>         <dbl>  <dbl>
1 AT     unemployment   5.34   5.62
2 DE     unemployment   5.23   4.98
```

```
# A tibble: 4 × 4
  c_code  year exports variable
  <chr>  <int>   <dbl> <chr>
1 AT      2013    53.4 exports
2 AT      2014    53.4 exports
3 DE      2013    45.4 exports
4 DE      2014    45.6 exports
```

```
# A tibble: 2 × 2
  country important_industries
  <chr>   <chr>
1 DE      Cars (25%) and meat (10%)
2 AT      Wine (5%) and milk (2%)
```

- The goal of data wrangling is to turn such untidy data into tidy data

Importing data

Make yourself comfortable before reading in data - expect frustration and pain!

- General idea: you import the data and bind it to an R object - usually a `data.frame`

- Then you proceed with transforming this `data.frame` until it satisfies the demands for tidy data

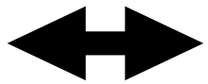
- Then you save the data under a new name, save the script, and celebrate yourself 🎉🍾🥂

Outlook

- After having imported your data into R, you can usually make it tidy using a sequential combination of the following routines:

Reshaping data from long to wide format (and vice versa)

```
# A tibble: 4 × 4
  c_code  year exports unemployment
  <chr>  <int>   <dbl>        <dbl>
1 AT      2013    53.4         5.34
2 AT      2014    53.4         5.62
3 DE      2013    45.4         5.23
4 DE      2014    45.6         4.98
```



```
# A tibble: 8 × 4
  c_code  year variable     value
  <chr>  <int> <chr>        <dbl>
1 AT      2013 exports       53.4
2 AT      2013 unemployment   5.34
3 AT      2014 exports       53.4
4 AT      2014 unemployment   5.62
5 DE      2013 exports       45.4
6 DE      2013 unemployment   5.23
7 DE      2014 exports       45.6
8 DE      2014 unemployment   4.98
```

Outlook

- After having imported your data into R, you can usually make it tidy using a sequential combination of the following routines:

Filter rows according to conditions

```
# A tibble: 4 × 4
  c_code  year exports unemployment
  <chr>  <int>   <dbl>        <dbl>
1 AT      2013    53.4         5.34
2 AT      2014    53.4         5.62
3 DE      2013    45.4         5.23
4 DE      2014    45.6         4.98
```



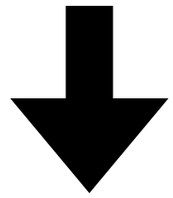
```
# A tibble: 2 × 4
  c_code  year exports unemployment
  <chr>  <int>   <dbl>        <dbl>
1 DE      2013    45.4         5.23
2 DE      2014    45.6         4.98
```

Outlook

- After having imported your data into R, you can usually make it tidy using a sequential combination of the following routines:

Select columns/variables

```
# A tibble: 4 × 4
  c_code  year exports unemployment
  <chr>  <int>   <dbl>        <dbl>
1 AT      2013    53.4         5.34
2 AT      2014    53.4         5.62
3 DE      2013    45.4         5.23
4 DE      2014    45.6         4.98
```



```
# A tibble: 4 × 3
  c_code  year exports
  <chr>  <int>   <dbl>
1 AT      2013    53.4
2 AT      2014    53.4
3 DE      2013    45.4
4 DE      2014    45.6
```

Outlook

- After having imported your data into R, you can usually make it tidy using a sequential combination of the following routines:

Mutate or create variables

```
# A tibble: 2 × 4
  c_code variable     `2013` `2014`
  <chr>  <chr>         <dbl>  <dbl>
1 AT     unemployment   5.34   5.62
2 DE     unemployment   5.23   4.98
```



```
# A tibble: 2 × 5
  c_code variable     `2013` `2014` change
  <chr>  <chr>         <dbl>  <dbl>  <dbl>
1 AT     unemployment   5.34   5.62  0.285
2 DE     unemployment   5.23   4.98 -0.25
```

Outlook

- After having imported your data into R, you can usually make it tidy using a sequential combination of the following routines:

```
# A tibble: 4 × 4
  c_code  year exports unemployment
  <chr>  <int>   <dbl>        <dbl>
1 AT      2013    53.4         5.34
2 AT      2014    53.4         5.62
3 DE      2013    45.4         5.23
4 DE      2014    45.6         4.98
```

```
# A tibble: 2 × 3
  c_code exports_avg unemployment_avg
  <chr>        <dbl>            <dbl>
1 AT            53.4             5.48
2 DE            45.5             5.11
```

Group and **summarise** data

Outlook

- After having imported your data into R, you can usually make it tidy using a sequential combination of the following routines:

```
# A tibble: 4 × 3
  c_code  year exports
  <chr>  <int>   <dbl>
1 AT      2013    53.4
2 AT      2014    53.4
3 DE      2013    45.4
4 DE      2014    45.6
```

```
# A tibble: 4 × 3
  c_code  year unemployment
  <chr>  <int>        <dbl>
1 AT      2013         5.34
2 AT      2014         5.62
3 DE      2013         5.23
4 DE      2014         4.98
```

```
# A tibble: 4 × 4
  c_code  year exports unemployment
  <chr>  <int>   <dbl>        <dbl>
1 AT      2013    53.4         5.34
2 AT      2014    53.4         5.62
3 DE      2013    45.4         5.23
4 DE      2014    45.6         4.98
```

Merge several data sets

Outlook

- After having imported your data into R, you can usually make it tidy using a sequential combination of the following routines:

Reshaping data from long to wide format (and vice versa)

Mutate or create variables

Filter rows according to conditions

Group and **summarise** data

Select columns/variables

Merge several data sets

- In the next, and a later session we will go through these operation
  - Then you are fit to tidy up raw data yourself
- This way you produce the inputs we used for visualisation…
  - …and the inputs we will use for modelling

Summary and outlook

- Next week we will learn how to visualise data that is stored in data frames
- This will be the first big intro session into data science fundamentals, succeeded by sessions on data wrangling and project management

Tasks until next week:

1. Fill in the **quick feedback survey** on Moodle
2. Read the **tutorials** posted on the course page
3. Do the **exercises** provided on the course page and **discuss problems** and difficulties via the Moodle forum