Importing data

Theoretical and Empirical Research Methodology,
Implementation Lab 5

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

Goals for today

I. Learn how to handle raw and tidy data in R

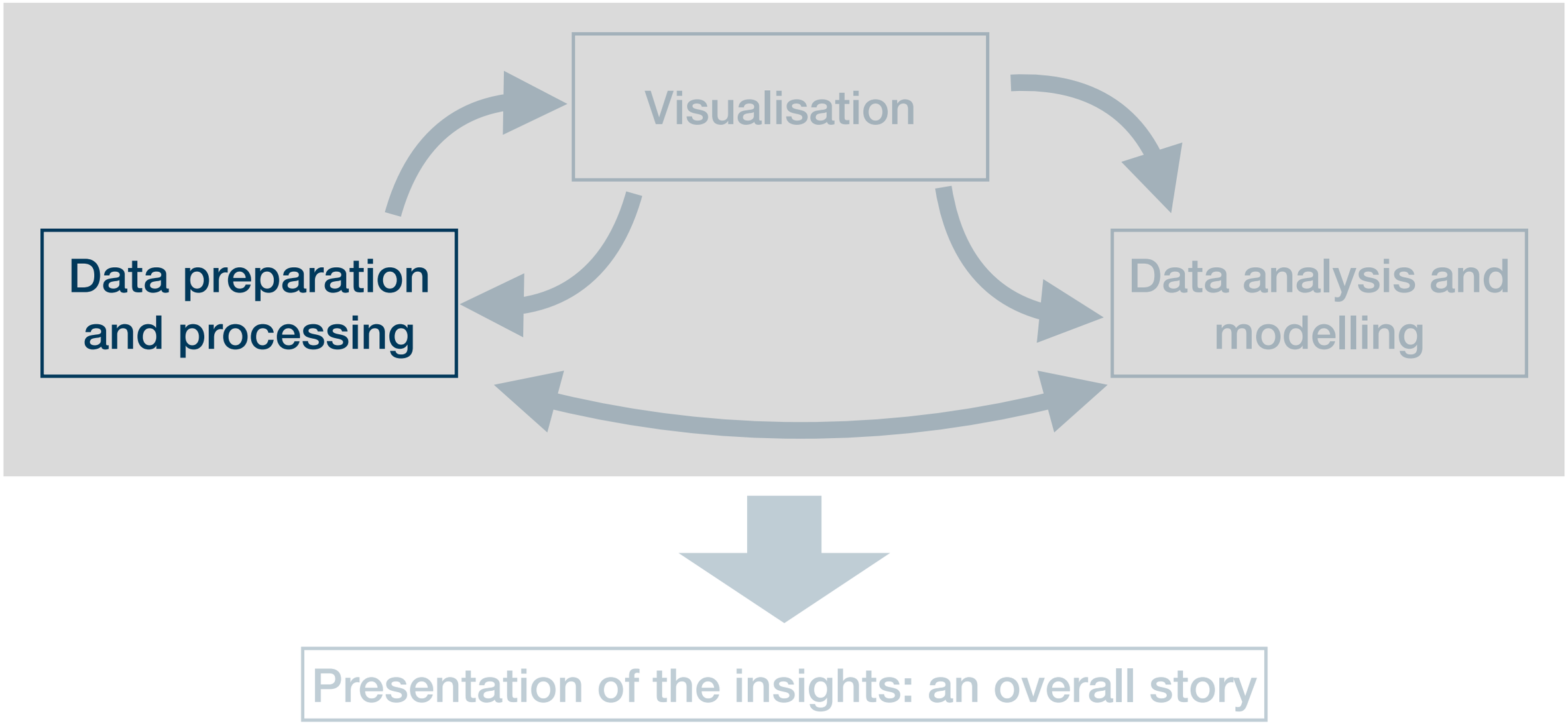
II. Learn how to import data into R using `data.table::fread()`

III. Learn how to save data

Note:
Importing data is an area, in which using AI tools becomes particularly helpful once you understand the basics of how importing data works, and what to keep in mind!

The role of data preparation

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



Focus of this session

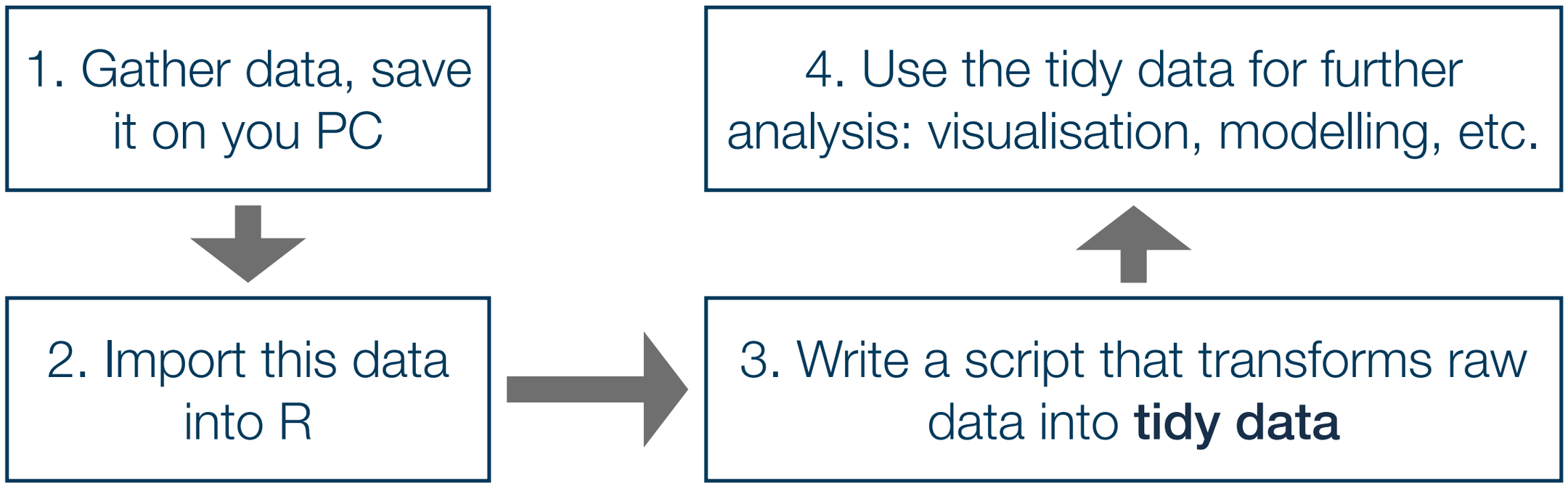
- Use **directory structure** introduced in session on project management
- Learn how to import data using the most widely used file formats, esp. `csv`
- Goal: all results must be **reproducible** from the raw data at any time
  - This implies that you **must not manipulate your raw data** at any cost
  - **Raw data** = what you download from the internet, gather through an experiment, or code yourself
  - This session: how to get the raw data “into R” and “out into the file system”

Builds on session on project management

Predates session on data preparation

Recap: 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 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

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

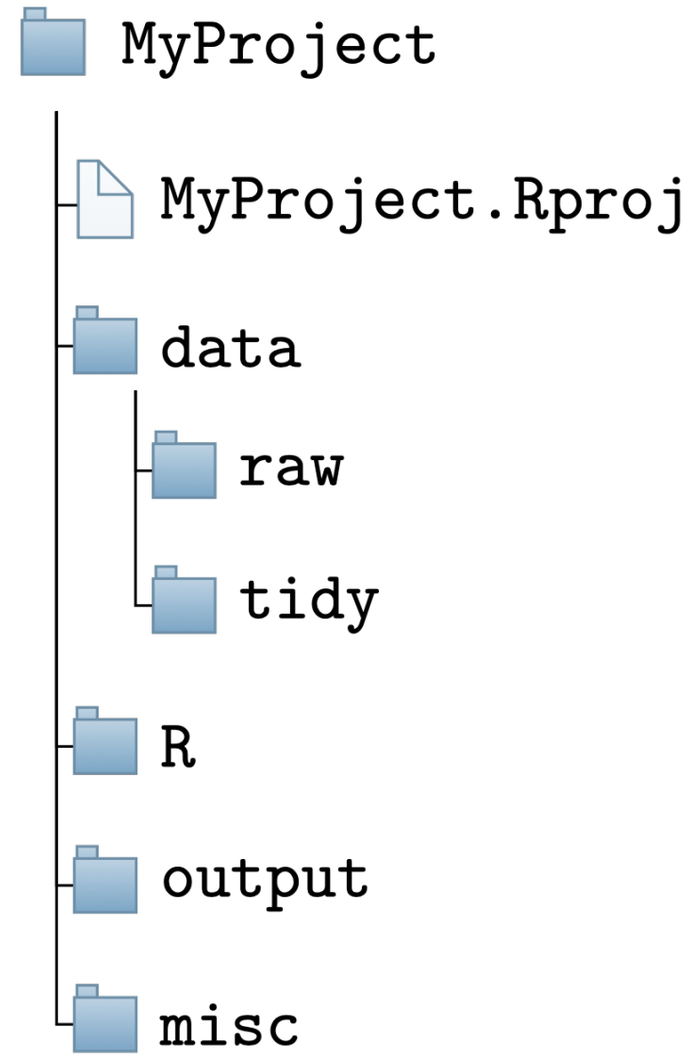
Importing data

Preparation: set up your working environment

See separate session on project management

- Raw data should be saved in `data/raw`
- If you have very few data sets, you might also use only data
- Tidied up data should be saved in `data/tidy`
  → keep it separate

See separate session on data preparation



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

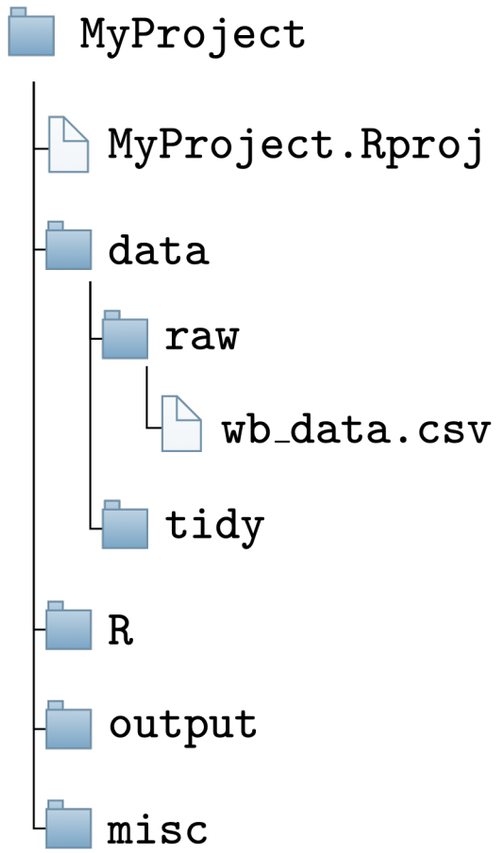
- Good practice: save path to file in a vector:

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

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

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

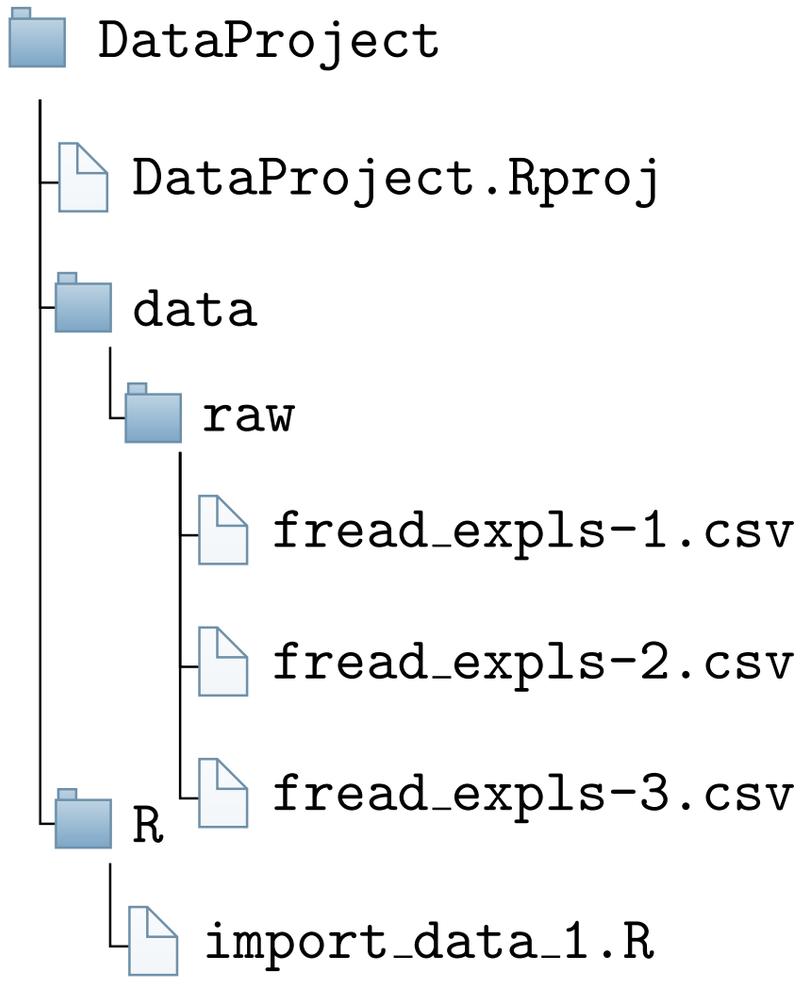
  - In general, I recommend using `data.table::fread()`
  - But: alternatives available, including from `tidyverse`



- 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

Exercise 1

- Download the zip file **`fread_expls.zip`** from the course homepage
- Extract the zip file within the folder **`data/raw/`** in your R project
- Write a script that imports the data set saved in the file **`fread_expls-1.csv`** into your session



How to use data.table::fread()

- *See also the tutorial on data import*
- In the following we will learn when and how to use the following arguments of `data.table::fread()`:
  - `file`: the relative path to the csv file you want to read → use `here::here()`
  - `sep`: symbol that separates columns
  - `dec`: symbol used as decimal sign
  - `colClasses`: what object type should be used for the columns?
- For other widely used commands check the tutorial and do the exercises
  - But note that there are even more specification options → `help(fread)`

How to use data.table::fread()
Specify column separator

- *See also the tutorial on data import*
- In the following we will learn when and how to use the following arguments of `data.table::fread()`:
  - `file`: the relative path to the csv file you want to read → use `here::here()`
  - `sep`: **symbol that separates columns**
  - `dec`: symbol used as decimal sign
  - `colClasses`: what object type should be used for the columns?
- For other widely used commands check the tutorial and do the exercises
  - But note that there are even more specification options → `help(fread)`

How to use data.table::fread()
Specify column separator

c_code; year; exports; unemployment
AT; 2013; 53.44; 5.34
AT; 2014; 53.39; 5.62
DE; 2013; 45.4; 5.23
DE; 2014; 45.64; 4.98

- Especially in Germany, columns are often separated via `;` instead of `,`
- We can pass a string to `sep` indicating how the columns are separated
  - In the above case: `sep = ";"`

How to use data.table::fread()
Specify column separator

- *See also the tutorial on data import*
- In the following we will learn when and how to use the following arguments of `data.table::fread()`:
  - `file`: the relative path to the csv file you want to read → use `here::here()`
  - `sep`: symbol that separates columns
  - `dec`: **symbol used as decimal sign**
  - `colClasses`: what object type should be used for the columns?
- For other widely used commands check the tutorial and do the exercises
  - But note that there are even more specification options → `help(fread)`

How to use data.table::fread()
Specify decimal separator

```
c_code; year; exports; unemployment
AT; 2013; 53,44; 5,34
AT; 2014; 53,39; 5,62
DE; 2013; 45,4; 5,23
DE; 2014; 45,64; 4,98
```

- Again in Germany, decimal places are often separated via `,` instead of `.`
- We can pass a string to `dec` indicating how the columns are separated
  - In the above case: `dec = ","`

Exercise 2

- Write a script that imports the data set `fread_expls-2.csv` into your session such that the following `tibble` results:

```
# 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
```

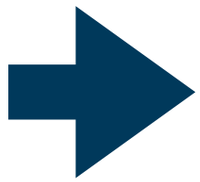
How to use data.table::fread()
Specifying column types using colClasses

- *See also the tutorial on data import*
- In the following we will learn when and how to use the following arguments of `data.table::fread()`:
  - `file`: the relative path to the csv file you want to read → use `here::here()`
  - `sep`: symbol that separates columns
  - `dec`: symbol used as decimal sign
  - `colClasses`: **what object type should be used for the columns?**
- For other widely used commands check the tutorial and do the exercises
  - But note that there are even more specification options → `help(fread)`

How to use data.table::fread()
Specifying column types using colClasses

- Whenever numbers should be saved as character, the guessing algorithm of `data.table::fread()` often fails:

```
c_code,year,exports, PROD_CODE
AT, 2013, 53.44, 0011
AT, 2014, 53.39, 0011
DE, 2013, 45.4, 0011
DE, 2014, 45.64, 0011
```



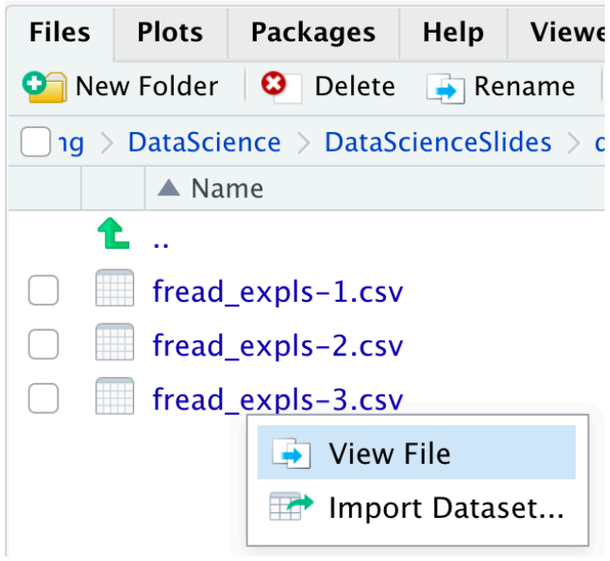
```
# A tibble: 4 × 4
  c_code  year exports PROD_CODE
  <chr>  <int>   <dbl>     <int>
1 AT      2013    53.4        11
2 AT      2014    53.4        11
3 DE      2013    45.4        11
4 DE      2014    45.6        11
```

- We can specify the column types explicitly by passing a vector to `colClasses`:
  - `colClasses = c("character", rep("double", 2), "character")`
- Usually, this is often a good idea to make your code more transparent
- You can also combine it with `select` and only read selected columns (see tutorial)

Exercise 3

- Now read in the file `fread_expls-3.csv` and use all the arguments you consider to be necessary
- Make sure that the column `cgroup` is stored as a `factor`

- Hint:
  - To get an idea about the raw data, click on the file and select “View File” to see it in its raw form → helps you to choose the right arguments:
  - Infeasible for very large files → use `nrows` and `select` to read a representative subset (see tutorial)



And what about saving data?

- Saving data is much easier than reading data
- The only relevant question is about the format
  - If there are no good arguments for using a different format, go for csv
- This can be achieved by `data.table::fwrite()` with the main arguments:
  - `x`: the name of the object to be saved
  - `file`: the file name under which the object should be saved
- Example: save object `exp_tab` to file `data/exp_tab.csv`:

```
data.table::fwrite(
    x = exp_tab,
    file = here::here("data/exp_tab.csv")
)
```

Data import - the general idea

Make yourself comfortable before reading in data - expect frustration!

- General idea: you import the data and bind it to an R object - usually a `data.frame` or whatever aligns with your preferred dialect

- 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 🎉🍾🥂

- We will cover the transformation steps in the next session

Summary and conclusion

- You learned how to import data into R
- Main focus: importing `csv` data files using `data.table::fread()`
  - Other **functions** for `csv` provided, e.g., via the `tidyverse` packages
  - Other **formats**: specialised functions available, esp. in the `haven` package
- Importing standard data often works well with **default options**
  - In other cases, optional arguments must be used → check **function documentation**
- If **speed or memory restrictions** are an issue, comparing import functions is advisable
- **Note:** once you understand the basics, using **AI** to assist with coding becomes very useful in the context of data import