R Projects and data import

Applied Data Science using R

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 set up an R project

II. Learn about the difference between absolute and relative paths

III. Learn how to use the here package

Our 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

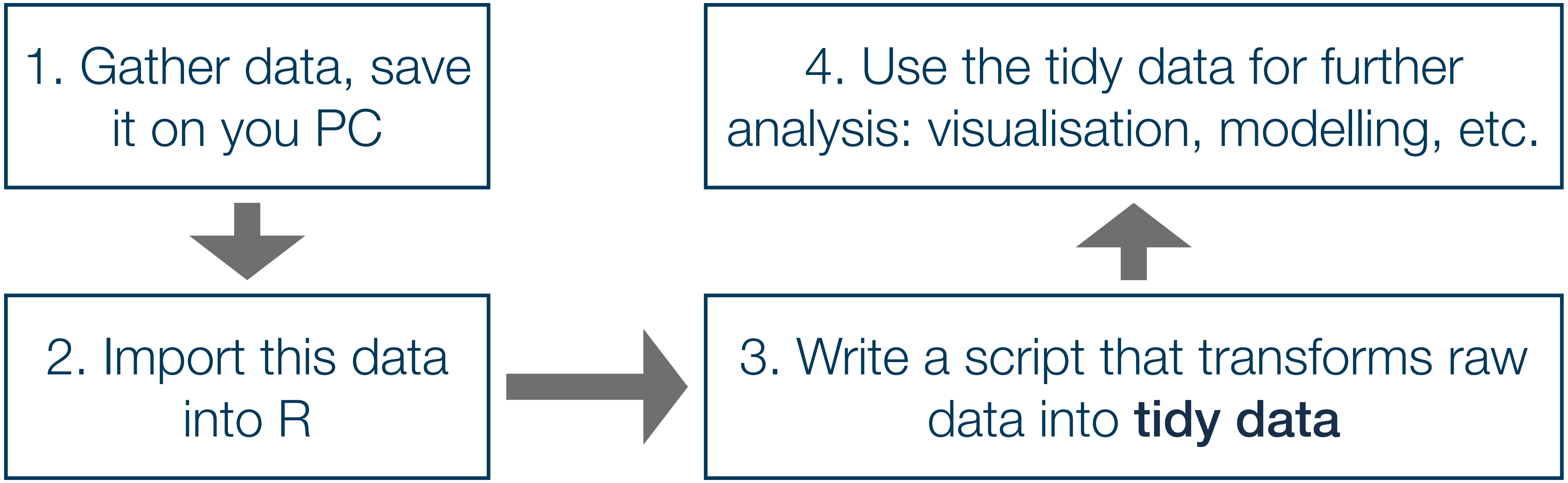
  ☝️Nothing is worse than hating your past-you for not documenting correctly where data came from, or how it has been prepared 🫣

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

- 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
  - Raw data = what you download from the internet, gather through an experiment, or code yourself
  - Focus here: organization of your overall project

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

Outlook

Set up you project environment

This is done only once per project

Import data

Transform raw data into tidy data

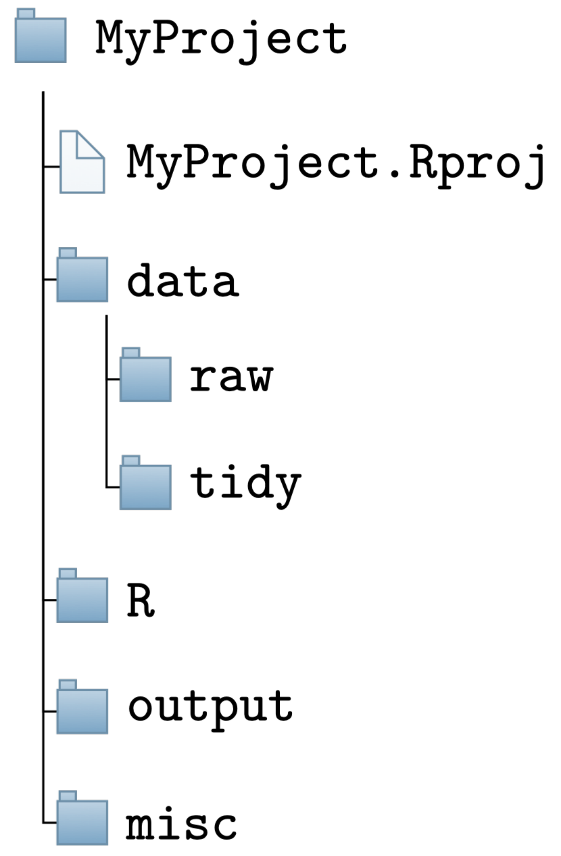
Save data

This might be done several times

Set up 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

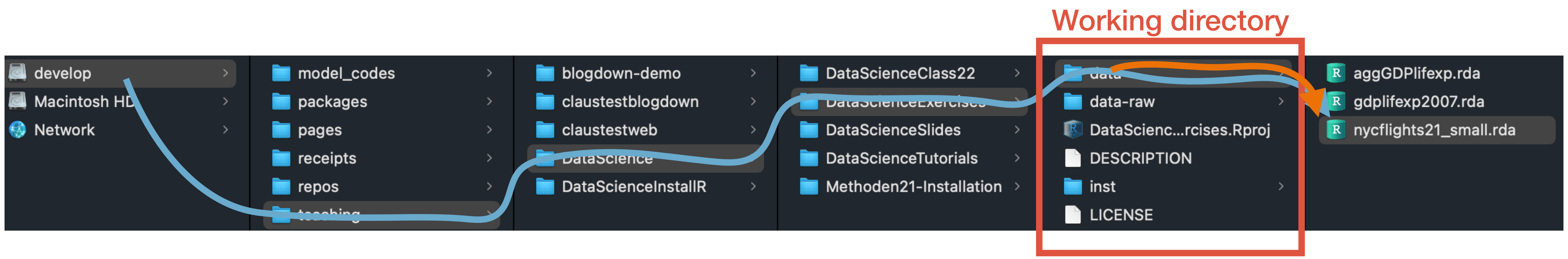


Creating an R project

See the associated tutorial and video for the documentation of the relevant steps (slides focus on selected background concepts only)

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 😀

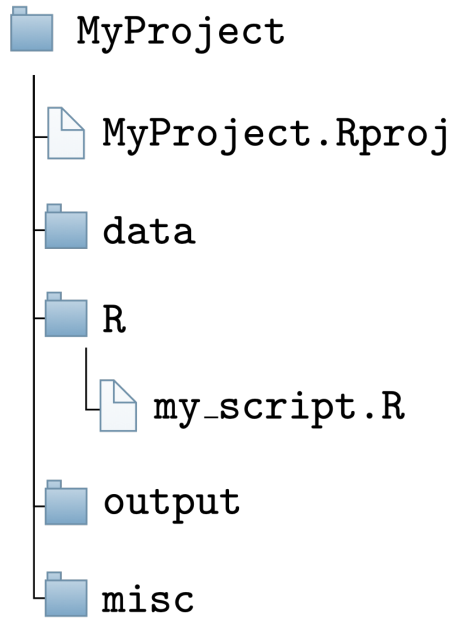
```
> setwd("/Volumes/Macintosh HD/Users/AbbysUserName/PathToF
olderThatOnlyExistsHere/ProjectName/file.txt")
Error in setwd("/Volumes/Macintosh HD/Users/AbbysUserName/
PathToFolderThatOnlyExistsHere/ProjectName/file.txt") :
  cannot change working directory
```

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()`

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

```

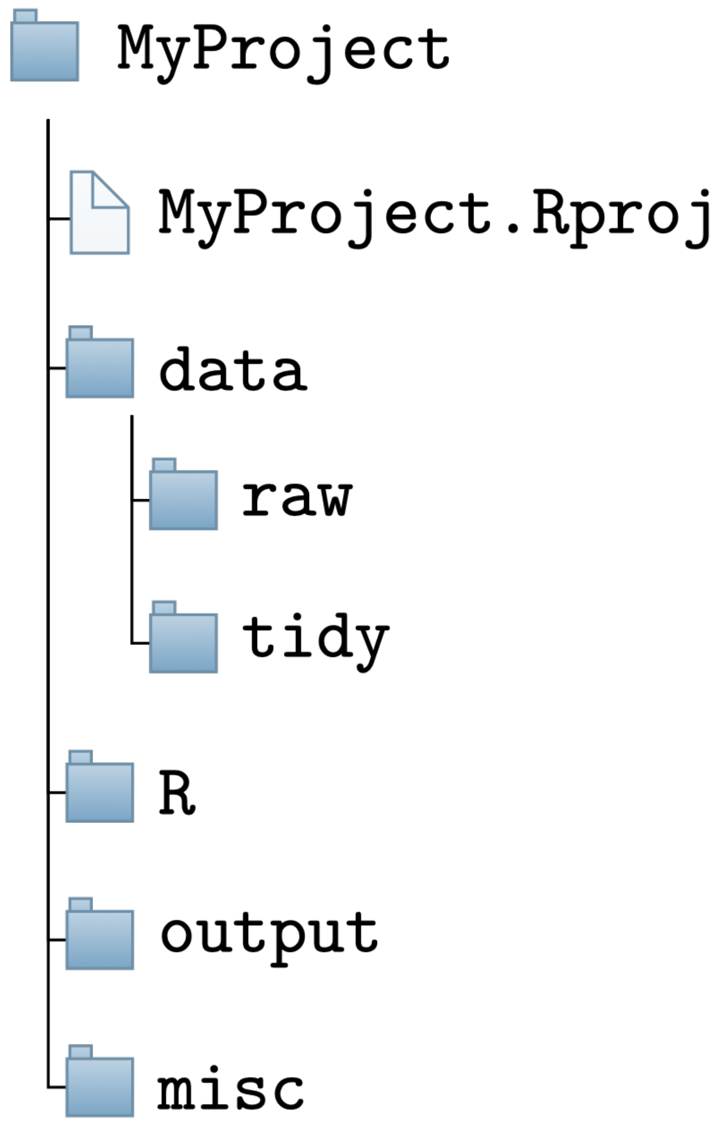


Your turn: final exercise

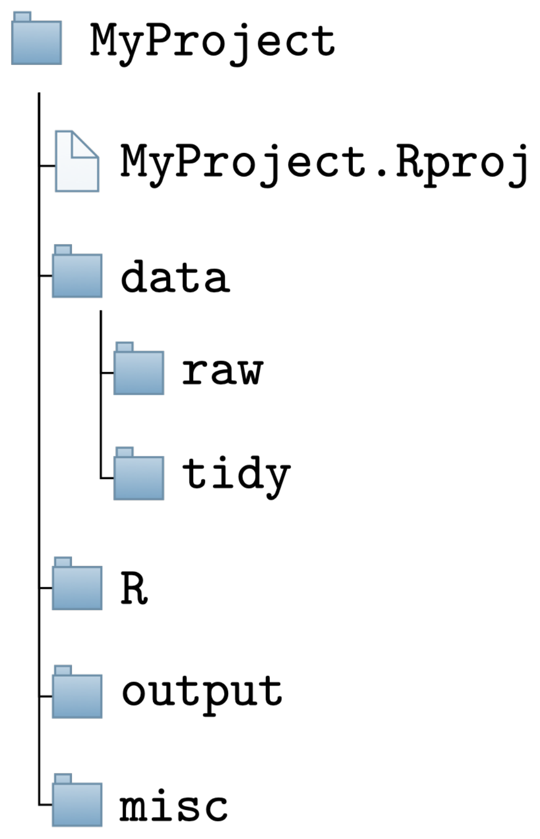
- Create a new R-Project on your computer
- Create all the required folders
- Write an R script, put it into the right directory, and make it usable for the `here`-package
- Check out what the function `here::here()` returns and experiment with its use

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

```



Summary and outlook



- We now know how to organise our **working directory**
- Important difference between **absolute and relative paths**
- Challenge of using code on different machines can be addressed using the **here package**
  - Better alternative than using `setwd()`
- Project management essential but often under-appreciated!
- Further topics:
  - Using a version control system (such as Git)
  - Using virtual programming environments (e.g. via the package `renv`)