

Data wrangling

Applied Data Science using R, Session 8

Prof. Dr. Claudius Gräbner-Radkowitzch

Europa-University Flensburg, Department of Pluralist Economics

www.claudius-graebner.com | [@ClaudiusGraebner](https://twitter.com/ClaudiusGraebner) | claudius@claudius-graebner.com

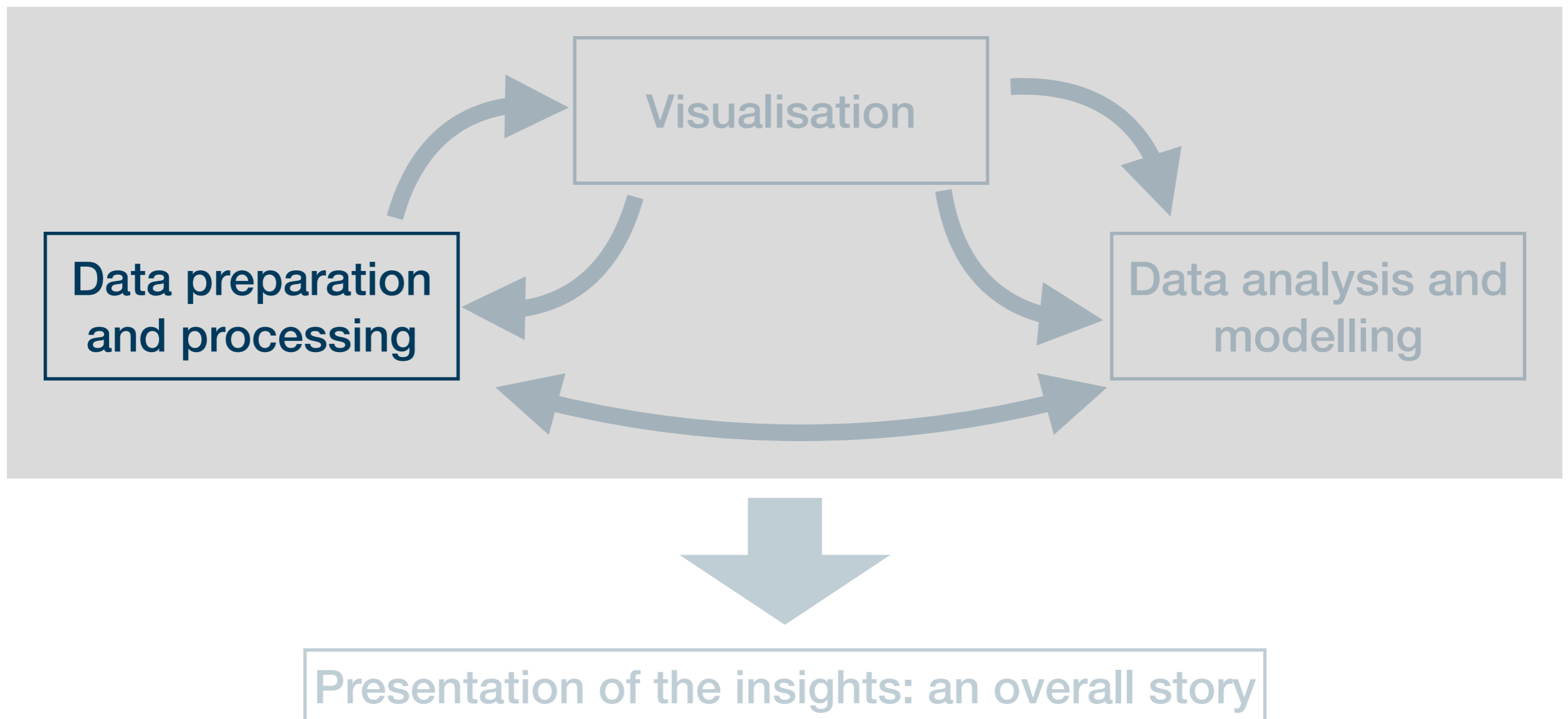
Goals for today

- I. Understand the concept of tidy data
- II. Get an overview over the most common transformation challenges
- III. Master a number of functions from the `tidyr` and `dplyr` packages to address some of these challenges

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 🙄



What is 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**

```
# 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 **row** corresponds to one and only one **observation**

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



```
# 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
```

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



```
# 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
```

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



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

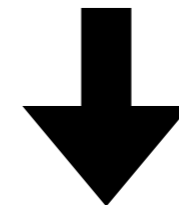
- The goal of data wrangling is to turn such untidy data into tidy data

Typical challenges during data wrangling

- 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
```

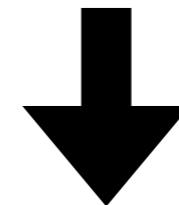


```
# A tibble: 2 × 4
  c_code year exports unemployment
  <chr>  <int>  <dbl>      <dbl>
1 DE     2013   45.4       5.23
2 DE     2014   45.6       4.98
```

Typical challenges during data wrangling

- 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
```



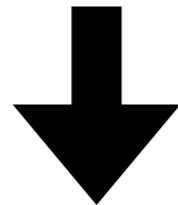
Select columns/variables

```
# 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
```

Typical challenges during data wrangling

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

```
# 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
```

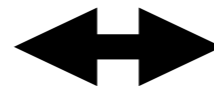
Mutate or create variables

Typical challenges during data wrangling

- 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
```

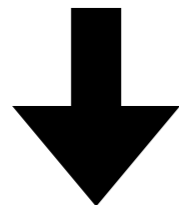


```
# A tibble: 8 × 4
  c_code year variable value
  <chr>  <int> <chr>    <dbl>
1 AT      2013 exports  53.4
2 AT      2013 unemployment 5.34
3 AT      2014 exports  53.4
4 AT      2014 unemployment 5.62
5 DE      2013 exports  45.4
6 DE      2013 unemployment 5.23
7 DE      2014 exports  45.6
8 DE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

Typical challenges during data wrangling

- 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
```

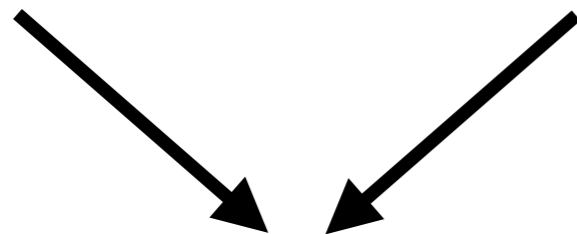
Group and summarise data

Typical challenges during data wrangling

- 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
```



Merge several data sets

```
# 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
```

Typical challenges during data wrangling

- 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)

Filter rows according to conditions

Select columns/variables

Mutate or create variables

Group and **summarise** data

Merge several data sets

- In this, 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

Addressing wrangling challenges

Session content

- We will go through the following challenges via direct demonstration:

Filter rows according to conditions

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

Select columns/variables

Mutate or create variables

Group and **summarise** data

Merge several data sets

- We will use functions from the packages `dplyr` and `tidyr` (both part of the `tidyverse`)
- For documentation purposes check out the lecture notes and the readings
 - The data sets used for the following exercises are all contained in `wrangling_exercises_data.zip`, which is available on the course homepage

Long and wide data

Rather wide data:

```
# 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
```

Rather long data:

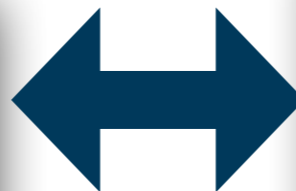
```
# A tibble: 8 × 4
  c_code year variable value
  <chr>  <int> <chr>         <dbl>
1 AT      2013 exports       53.4
2 AT      2013 unemployment  5.34
3 AT      2014 exports       53.4
4 AT      2014 unemployment  5.62
5 DE      2013 exports       45.4
6 DE      2013 unemployment  5.23
7 DE      2014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

- Better understood as *relative* descriptions of data
 - It is more straightforward to speak of a data set that is *longer* relative to another

Using pipes

- While not strictly necessary, you can improve the usability and readability of your code using so called pipes: %>%
- Pipes take the result of one line and 'throw' them into the next line
 - The thrown result can be referred to via .

```
data_sub <- dplyr::select(  
  .data = data_raw,  
  country, year, unemp, gdp)
```

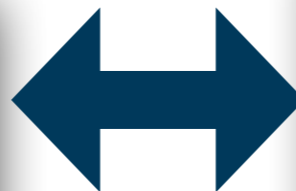


```
data_sub <- data_raw %>%  
  dplyr::select(  
    .data = .,  
    country, year, unemp, gdp)
```

Using pipes

- While not strictly necessary, you can improve the usability and readability of your code using so called pipes: %>%
- Pipes take the result of one line and ‘throw’ them into the next line
 - The thrown result can be referred to via .
 - By default, the thrown result is used as the first argument of the function in the next line

```
data_sub <- dplyr::select(  
  .data = data_raw,  
  country, year, unemp, gdp)
```



```
data_sub <- data_raw %>%  
  dplyr::select(  
    .data = .,  
    country, year, unemp, gdp)
```



```
data_sub <- data_raw %>%  
  dplyr::select(  
    country, year, unemp, gdp)
```

Using pipes

- A more practical example:

```
chain_1 <- tidyr::pivot_longer(  
  data = data_raw_wide,  
  cols = c("gdp", "gini", "unemp"),  
  names_to = "indicator",  
  values_to = "val")
```

```
chain_2 <- tidyr::pivot_wider(  
  data = chain_1,  
  names_from = "year",  
  values_from = "val")
```

```
chain_complete <- pipe_data_raw %>%  
  tidyr::pivot_longer(  
    data = .,  
    cols = c("gdp", "gini", "unemp"),  
    names_to = "indicator",  
    values_to = "val") %>%  
  tidyr::pivot_wider(  
    data = .,  
    names_from = "year",  
    values_from = "val")
```

- Pipes make code almost always easier to read → desired stage at the end
- But it is usually easier to make intermediate steps explicit during code development

Digression: tidy selection helpers

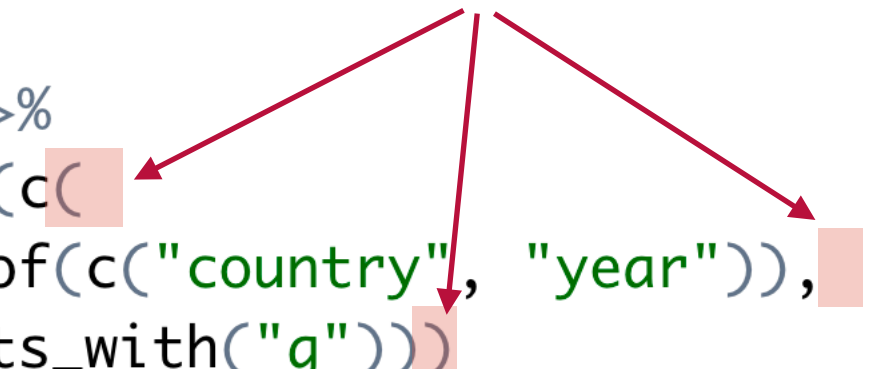
- It can become tedious to select many columns using explicit reference to their names
- The tidy selection helpers are a useful tool to select columns based on common criteria:

```
data_raw_wide %>%  
  dplyr::select(gdp, gini)
```

```
data_raw_wide %>%  
  dplyr::select(  
    tidyr::starts_with("g"))
```

Combine multiple selectors

```
data_raw_wide %>%  
  dplyr::select(c(  
    tidyr::all_of(c("country", "year")),  
    tidyr::starts_with("g")))
```



- For a complete list of helpers see, e.g., [the official reference](#)

Exercise 1: filtering and reshaping

- Use the data set `exercise_1.csv` contained in `wrangling_exercises_data.zip`

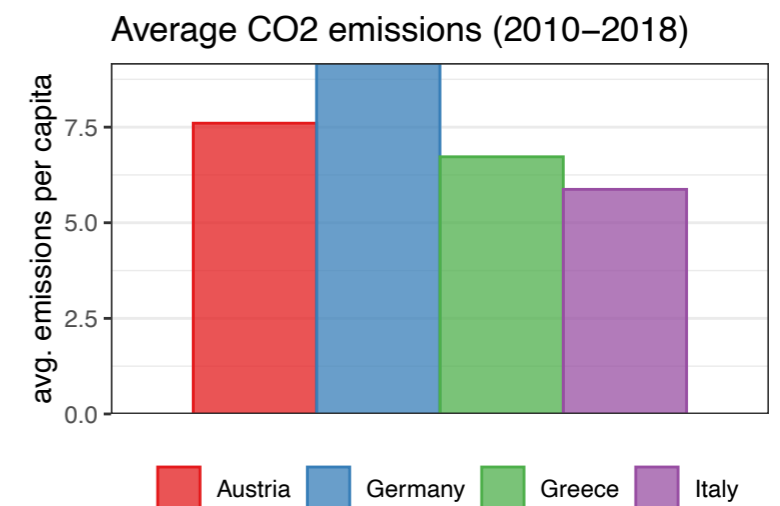
- Import the data and ...
 - ...only consider data on Greece and Germany between 1995 and 2015
 - ...make it wider (and tidy) 🙌
 - ...save it in the subfolder `data/tidy/`

```
# A tibble: 42 × 4
  country year    gdp    co2
  <chr>   <int> <dbl> <dbl>
1 Germany  1995  39366.  10.7
2 Germany  1996  39569.  11.0
3 Germany  1997  40219.  10.6
4 Germany  1998  41023.  10.5
5 Germany  1999  41770.  10.2
6 Germany  2000  42928.  10.1
7 Germany  2001  43577.  10.3
8 Germany  2002  43417.  10.1
9 Germany  2003  43089.  10.1
10 Germany 2004  43605.   9.95
# ... with 32 more rows
```

Exercise 2: mutating, selecting & summarising

- Use the data set `exercise_2.csv` contained in `wrangling_exercises_data.zip`
- Import the data
 - Only keep the variables `gdp`, `share_indus`, and `co2`
 - Divide the industry share in GDP with 100
 - Only keep data between 2010 and 2018
 - Compute the averages over time for all countries
- Bonus:
 - Visualise the resulting CO2 average via a bar plot

```
# A tibble: 12 × 3
  country indicator  time_avg
  <chr>   <chr>         <dbl>
1 Austria co2           7.60
2 Austria gdp          53322.
3 Austria share_indus 0.254
4 Germany co2           9.17
5 Germany gdp          50781.
6 Germany share_indus 0.272
7 Greece  co2           6.72
8 Greece  gdp          29169.
9 Greece  share_indus 0.144
10 Italy   co2           5.87
11 Italy   gdp          41326.
12 Italy   share_indus 0.213
```



Data: World Bank.

Summary & outlook

Summary

- After importing raw data you usually must prepare them → make **tidy**
- Tidy data is the input to any visualisation/modelling task and defined as data where:
 - Every **column** corresponds to one and only one **variable**
 - Every **row** corresponds to one and only one **observation**
 - Every **cell** corresponds to one and only one **value**
- It is usually a good idea to write a script that imports raw, and saves tidy data
- Such script usually makes use of functions from the following packages:
 - `data.table`, `dplyr`, `tidyr`, and `here`

Summary

- These packages provide functions that help you to address some wrangling challenges that regularly await you:
 - Reshaping data: `tidyr::pivot_longer()` and `tidyr::pivot_wider()`
 - Filtering rows: `dplyr::filter()`
 - Selecting columns: `dplyr::select()` and the select helpers
 - Mutating or creating variables: `dplyr::mutate()`
 - Grouping and summarising: `dplyr::group_by()` and `dplyr::summarise()`
 - Merging data sets: `dplyr::*_join()`
- In later sessions we will learn also about some convenience shortcuts

Outlook

- We now covered the basics in all fundamental data science activities
 - We can now turn to the ‘funny’ part: modelling and analysis
- But before we will learn how to write reports using Quarto / R Markdown
 - Learn to communicate your R activities to others by combining them with text, and distributing them online

Tasks until next time:

1. Fill in the **quick feedback survey** on Moodle
2. Read the **lecture notes** posted on the course page and replicate them
3. Have a look at the mandatory readings (step the challenges we did not cover yet)
4. Do the **exercises** provided on the course page and **discuss problems** and difficulties via the Moodle forum