

Data preparation

Applied Data Science using R

Prof. Dr. Claudius Gräbner-Radkowitz

Europa-University Flensburg, Department of Pluralist Economics

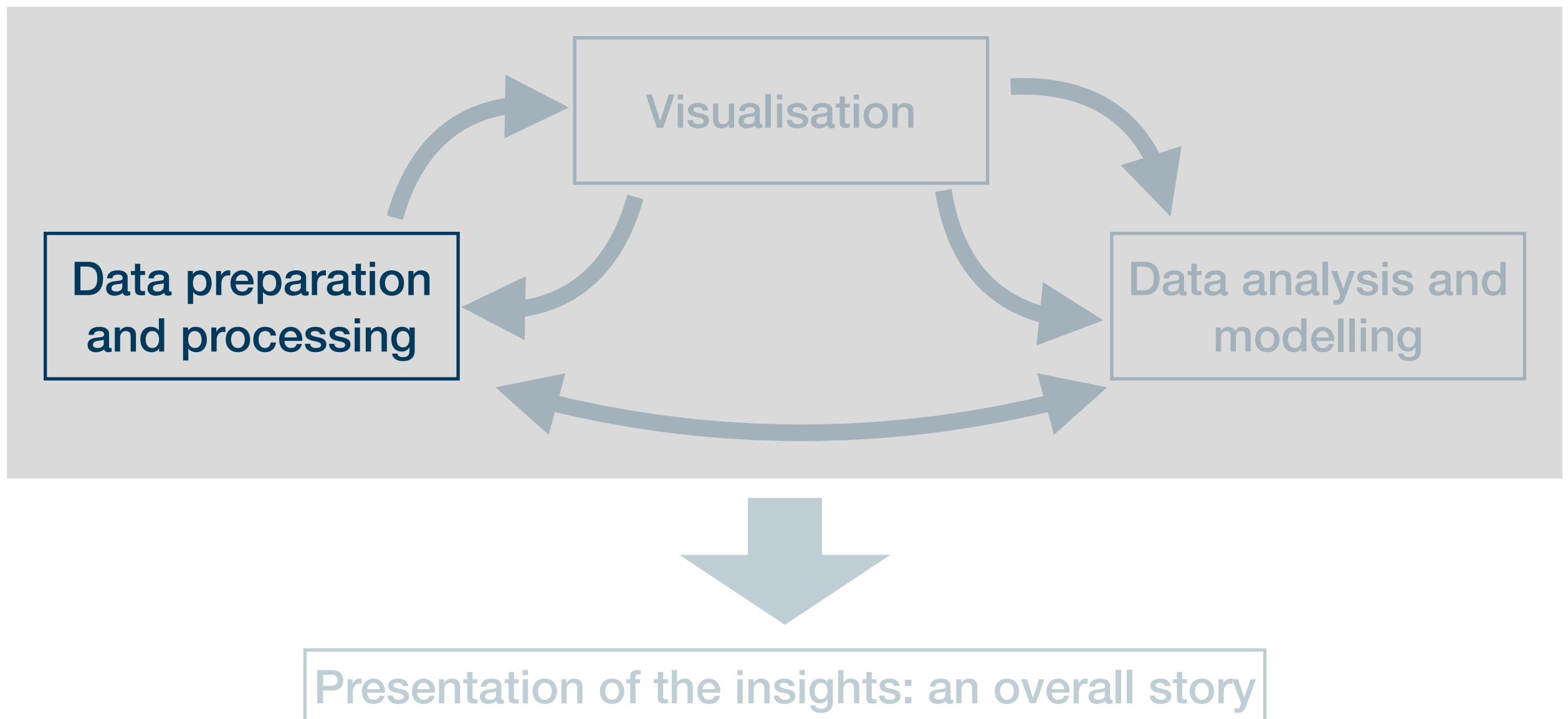
www.claudius-graebner.com | [@ClaudiusGraebner](https://twitter.com/ClaudiusGraebner) | claudius@claudius-graebner.com

Goals for today

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

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 every 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

Recap questions

- What are the three demands a data set needs to fulfil to count as 'tidy'?
- Why do we care about tidy data at all?
- Are there plausible reasons for transforming a tidy data set into a non-tidy data set?
- Consider the following data sets. Are they tidy? If not, what would you need to change to make them tidy?

```
# A tibble: 1 × 4
  country growth_2018 growth_2019 growth_2020
  <chr>         <dbl>         <dbl>         <dbl>
1 Germany      1.09           1.06          -4.57
```

```
# A tibble: 6 × 2
  VarName      Value
  <chr>        <dbl>
1 beer_consumption 81.7
2 beer_price       1.78
3 personal_income 25088
4 beer_consumption 56.9
5 beer_price       2.27
6 personal_income 26561
```

```
# A tibble: 6 × 3
  beer_consumption liquor_price water_price
  <dbl>           <dbl>         <dbl>
1 81.7            6.95          1.11
2 56.9            7.32          0.67
3 64.1            6.96          0.83
4 65.4            7.18          0.75
5 64.1            7.46          1.06
6 58.1            7.47          1.1
```


The way to tidy data

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

Hadley Wickham

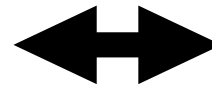


- The starting point to tidy data is always different
- The goal is always the same → so are the steps: **six main routines**
- Two main packages are relevant:
 - `tidyr` provides functions for **reshaping data** into tidy format (‘wrangling’)
 - `dplyr` provides functions for **manipulating data** to extract desired information

The six routines of data preparation

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
```

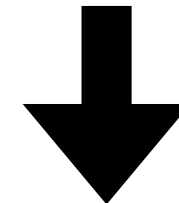


```
# A tibble: 8 × 4
  c_code year variable value
  <chr>  <int> <chr>      <dbl>
1 AT      2013 exports    53.4
2 AT      2013 unemployment 5.34
3 AT      2014 exports    53.4
4 AT      2014 unemployment 5.62
5 DE      2013 exports    45.4
6 DE      2013 unemployment 5.23
7 DE      2014 exports   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

The six routines of data preparation

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
```

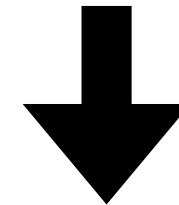


```
# A tibble: 2 × 4
  c_code year exports unemployment
  <chr>  <int>  <dbl>      <dbl>
1 DE     2013   45.4       5.23
2 DE     2014   45.6       4.98
```

The six routines of data preparation

Select columns/variables

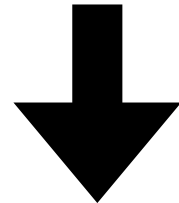
```
# A tibble: 4 × 4
  c_code year exports unemployment
  <chr>  <int>  <dbl>      <dbl>
1 AT      2013    53.4        5.34
2 AT      2014    53.4        5.62
3 DE      2013    45.4        5.23
4 DE      2014    45.6        4.98
```



```
# 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
```

The six routines of data preparation

```
# 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
```

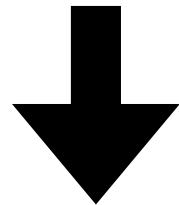


```
# A tibble: 2 × 5
  c_code variable `2013` `2014` change
  <chr> <chr> <dbl> <dbl> <dbl>
1 AT    unemployment 5.34 5.62 0.285
2 DE    unemployment 5.23 4.98 -0.25
```

Mutate or create variables

The six routines of data preparation

```
# 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
```



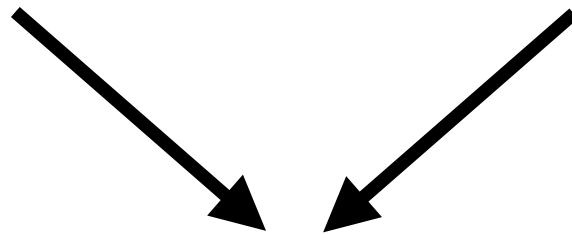
```
# A tibble: 2 × 3
  c_code exports_avg unemployment_avg
  <chr>      <dbl>      <dbl>
1 AT         53.4         5.48
2 DE         45.5         5.11
```

Group and summarise data

The six routines of data preparation

```
# 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
```

The six routines of data preparation

- 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

- With these six routines, you can prepare almost any messy data set
- This way you produce the inputs we used for visualisation...
 - ...and the inputs we will use for modelling

Recap questions

- What is the relation between long and wide data sets?
- Name the six main routines of data preparation and explain what they are used for.
- What does 'data wrangling' mean?
- Which two packages are used most frequently in the context of data preparation? What are their respective areas of application?

Six main routines for data preparation

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

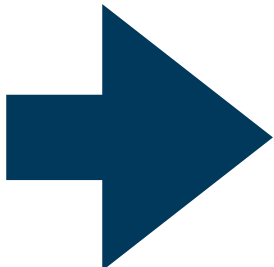
Merge several data sets

- 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

Short recap on reshaping

- Take the data set `data_raw_long` and transform it as follows:

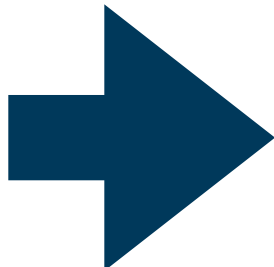
```
> data_raw_long
  country year variable  value
1: Germany 2017   unemp    3.75
2: Germany 2017    gdp 53071.46
3: Germany 2018   unemp    3.38
4: Germany 2018    gdp 53431.39
5:  Greece 2017   unemp   21.49
6:  Greece 2017    gdp 28604.86
7:  Greece 2018   unemp   19.29
8:  Greece 2018    gdp 29141.17
```



```
# A tibble: 4 × 4
  year variable  Germany  Greece
<int> <chr>      <dbl>   <dbl>
1  2017 unemp      3.75    21.5
2  2017 gdp      53071.  28605.
3  2018 unemp      3.38    19.3
4  2018 gdp      53431.  29141.
```

- Take the data set `gini_join` and transform it as follows:

```
> gini_join
# A tibble: 2 × 3
  country year  gini
<chr>   <int> <dbl>
1 Greece  2015  33.1
2 Greece  2017  32.2
```



```
# A tibble: 2 × 4
  country year Indicator Observation
<chr>   <int> <chr>      <dbl>
1 Greece  2015  gini      33.1
2 Greece  2017  gini      32.2
```

Short recap on manipulation basics

- Consider the data set `wine2dine` from the package `DataScienceExercises`



1. Filter the data set such that it only contains white wines
2. Then remove the column `'kind'`
3. Change the type of the column `'quality'` into `double`
4. Divide the values in the columns `'alcohol'` and `'residual sugar'` by 100
5. Filter the data such that you only keep the wines with the highest quality score

Short recap on summarising and grouping

- What is the difference between `dplyr::mutate()` and `dplyr::summarize()`?
 - Consider again the data set `wine2dine` from the package `DataScienceExercises`
1. Summarise the data by computing the mean alcohol, mean sugar, and mean quality of white and red wines
 2. Compute a variable indicating how the quality of each wine deviates from the average quality of all wines.



Short recap on joining data sets

- Consider the data sets `join_x.csv` and `join_y.csv` and join them on the columns `time` and `id` using the functions `left_join()`, `right_join()`, and `full_join()`!
- Try for yourself what the function `inner_join()` does. How does it differ from `left_join()`, `right_join()`, and `full_join()`?
- Consider the data sets `join_x.csv` and `join_y.csv` and the function `dplyr::full_join()`. What is the difference of joining on columns `time` and `id` vs joining only on column `id`?



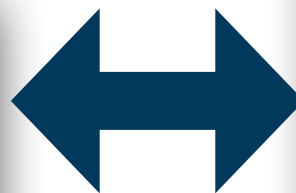
Helpful tools I: Pipes



Using pipes

- While not strictly necessary, you can improve the usability and readability of your code using so called pipes: %>%
- Pipes take the result from their left and ‘throw’ them on the right
 - The thrown result can be referred to via .
 - Usually they are used at the end of a line and ‘throw’ the result of one line into the next one

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

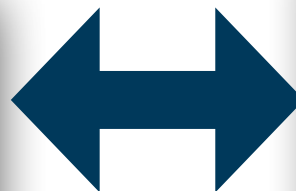


```
data_sub <- data_raw %>%  
  dplyr::select(  
    .data = .,  
    country, year, unemp, gdp)
```

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

Short recap on piping

- Explain what the pipe `%>%` does.
- When can the pipe be useful?
- Should you develop code with pipes right from the start? Why? Why not?



- Rewrite the following code 🙏
using pipes (data available via
course page)

```
pipedata_v1 <- data.table::fread(here("data/recap2.csv"))
```

```
pipedata_v2 <- tidyr::pivot_longer(  
  data = pipedata_v1,  
  cols = c("lifeExp", "gdpPerCap"),  
  names_to = "Indicator",  
  values_to = "Value")
```

```
pipedata_v3 <- tidyr::pivot_wider(  
  data = pipedata_v2,  
  names_from = "year",  
  values_from = "Value")
```

- Look at the introduction to the R package `magrittr`, which defines even more pipes: <https://magrittr.tidyverse.org/articles/magrittr.html>

Helpful tools II: Selection helpers

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`

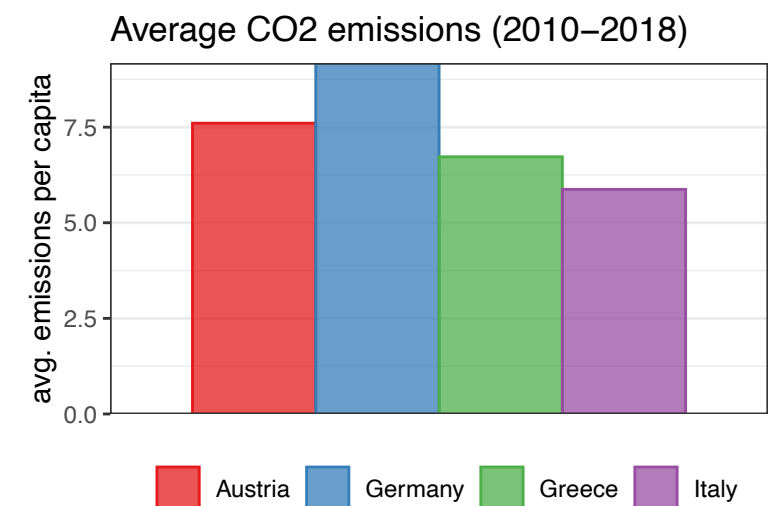
- Import the data and ...
 - ...only consider data on Greece and Germany between 1995 and 2015
 - ...make it wider (and tidy) 🙌
 - ...save it in the subfolder `data/tidy/`

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

General recap questions

- What are the three demands a data set needs to fulfil to count as ‘tidy’?
- Why do we care about tidy data at all?
- What is the relation between long and wide data sets?
- What are the six main routines of data preparation? What are they used for?
- What does ‘data wrangling’ mean?
- Which two packages are used most frequently in the context of data preparation? What are their respective areas of application?
- Explain what the pipe `%>%` does. When can the pipe be useful?

Data preparation is mainly about practice, so the practical exercises are particularly recommended 