



Data Science Project Management

DS400 Winter Term 2024/25

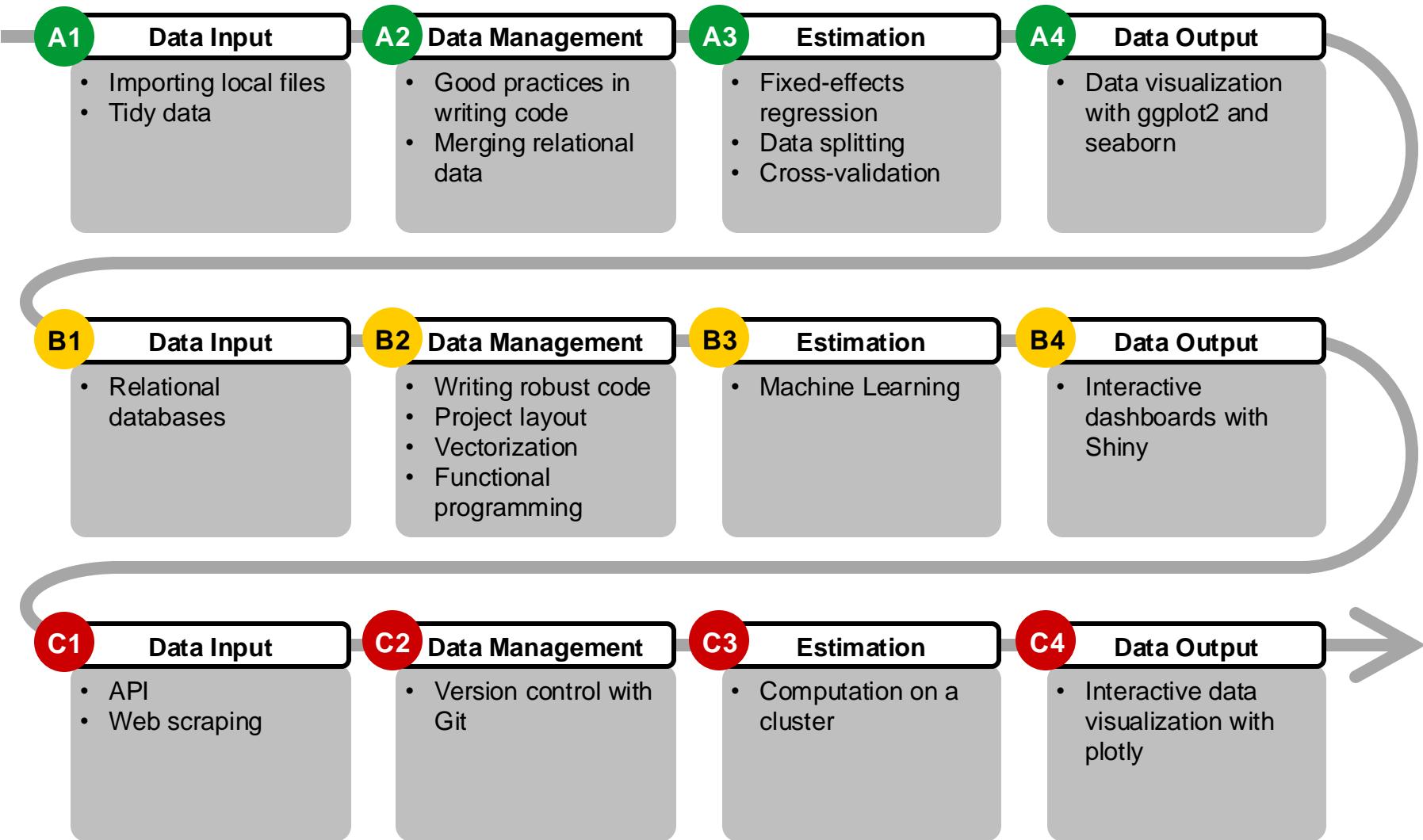
Prof. Dr. Stefan Mayer

Assistant Professor of Marketing Analytics

School of Business and Economics

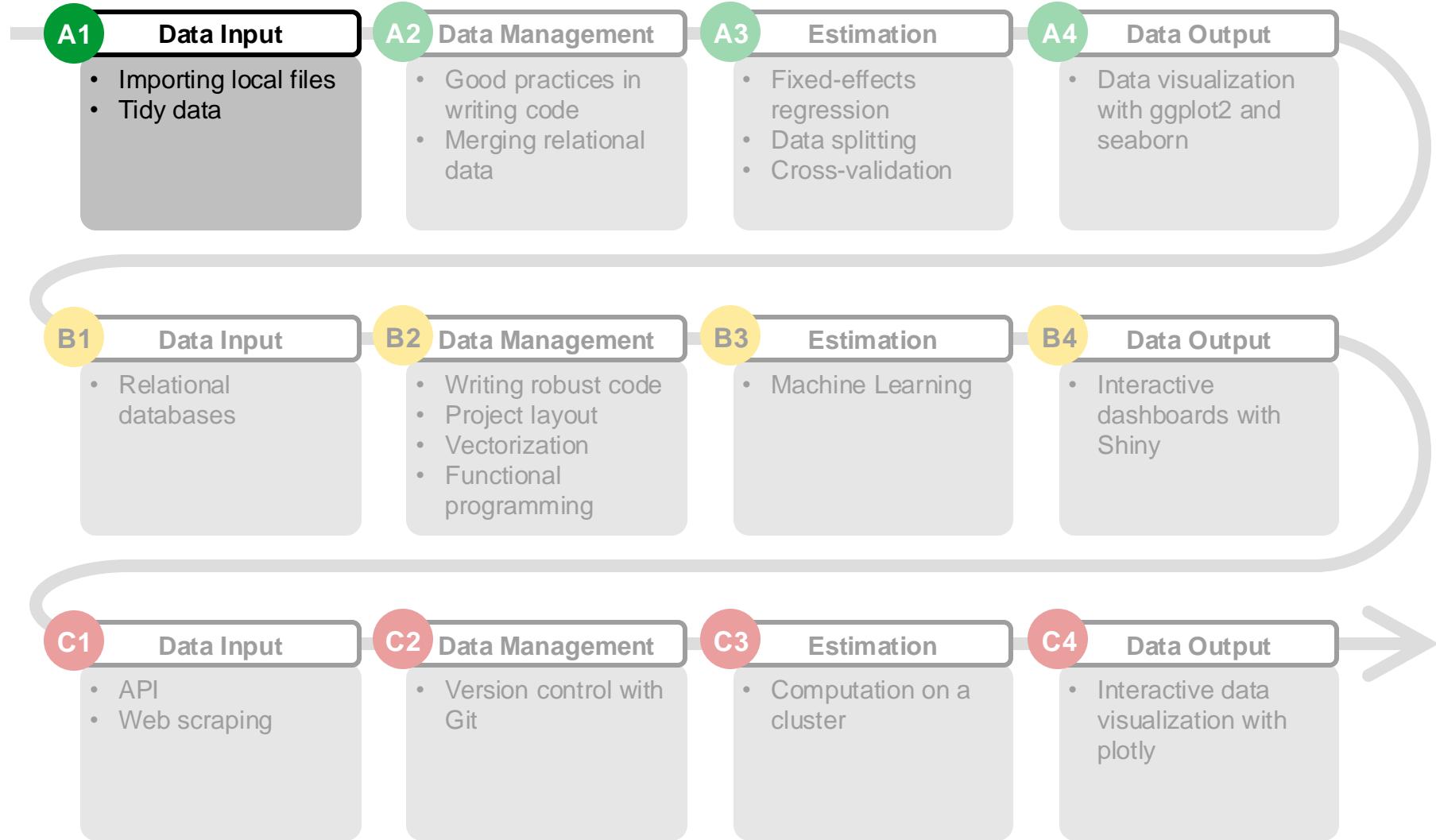
University of Tübingen

Data Science Project Management: Course Outline



All of the code snippets and examples in this lecture have been tested in a macOS environment.

Data Science Project Management: Course Outline



Flat files can come in very different layouts

Plain text files

- Often, data is stored in plain text format that can be interpreted in a simple text editor
- Frequently used file types are .csv, .tsv or .txt
- Their layout is not standardized and must be identified before the import

```
"Name", "Platform", "Year_of_Release", "Genre", "Publisher", "NA_Sales", "E  
"Wii Sports", "Wii", "2006", "Sports", "Nintendo", 41.36, 28.96, 3.77, 8.45, 8  
"Super Mario Bros.", "NES", "1985", "Platform", "Nintendo", 29.08, 3.58, 6.8  
"Mario Kart Wii", "Wii", "2008", "Racing", "Nintendo", 15.68, 12.76, 3.79, 3.  
"Wii Sports Resort", "Wii", "2009", "Sports", "Nintendo", 15.61, 10.93, 3.28  
"Pokemon Red/Pokemon Blue", "GB", "1996", "Role-Playing", "Nintendo", 11.2
```

- Separator: “,”
- Decimal: “.”

```
"Name"; "Platform"; "Year_of_Release"; "Genre"; "Publisher"; "NA_Sales"; "E  
"Wii Sports"; "Wii"; "2006"; "Sports"; "Nintendo"; 41,36; 28,96; 3,77; 8,45; 8  
"Super Mario Bros."; "NES"; "1985"; "Platform"; "Nintendo"; 29,08; 3,58; 6,8  
"Mario Kart Wii"; "Wii"; "2008"; "Racing"; "Nintendo"; 15,68; 12,76; 3,79; 3.  
"Wii Sports Resort"; "Wii"; "2009"; "Sports"; "Nintendo"; 15,61; 10,93; 3,28  
"Pokemon Red/Pokemon Blue"; "GB"; "1996"; "Role-Playing"; "Nintendo"; 11,2
```

- Separator: “;”
- Decimal: “,”

Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales
Wii Sports	Wii	2006	Sports	Nintendo	41.36
Super Mario Bros.	NES	1985	Platform	Nintendo	
Mario Kart Wii	Wii	2008	Racing	Nintendo	
Wii Sports Resort	Wii	2009	Sports	Nintendo	
Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nint	

- Separator: tab stop (“\t”)
- Decimal: “.”

Handling flat file formats



{xyz} stands for package xyz

	R (package {readr})	Python (package {pandas})
Main function ⇒ specify layout using:	<code>read_delim()</code>	<code>pd.read_csv()</code>
<ul style="list-style-type: none"> • Separator • Decimal • Values that are NA 	<pre>delim = "," locale = locale(decimal_mark = ".") na = c("", "NA")</pre>	<pre>sep = "," decimal = "." na_values = None</pre>

Tailored wrappers in {readr}

- In R, you can use specific functions for specific file layouts
- These call `read_delim()` behind the scenes with different default arguments

file layout	wrapper
• Separator ",", decimal "."	<code>read_csv()</code>
• Separator ";", decimal ","	<code>read_csv2()</code>
• Tab-separated values	<code>read_tsv()</code>

Different layouts need different import functions or arguments

“, “ as separator, “.” as decimal

```
"Name", "Platform", "Year_of_Release", "Genre", "Publisher", "NA_Sales", "E  
"Wii Sports", "Wii", "2006", "Sports", "Nintendo", 41.36, 28.96, 3.77, 8.45, 8  
"Super Mario Bros.", "NES", "1985", "Platform", "Nintendo", 29.08, 3.58, 6.8  
"Mario Kart Wii", "Wii", "2008", "Racing", "Nintendo", 15.68, 12.76, 3.79, 3.  
"Wii Sports Resort", "Wii", "2009", "Sports", "Nintendo", 15.61, 10.93, 3.28  
"Pokemon Red/Pokemon Blue", "GB", "1996", "Role-Playing", "Nintendo", 11.2
```

`read_csv()`

“;“ as separator, “;“ as decimal

```
"Name"; "Platform"; "Year_of_Release"; "Genre"; "Publisher"; "NA_Sales"; "E  
"Wii Sports"; "Wii"; "2006"; "Sports"; "Nintendo"; 41,36; 28,96; 3,77; 8,45; 8  
"Super Mario Bros."; "NES"; "1985"; "Platform"; "Nintendo"; 29,08; 3,58; 6,8  
"Mario Kart Wii"; "Wii"; "2008"; "Racing"; "Nintendo"; 15,68; 12,76; 3,79; 3,  
"Wii Sports Resort"; "Wii"; "2009"; "Sports"; "Nintendo"; 15,61; 10,93; 3,28  
"Pokemon Red/Pokemon Blue"; "GB"; "1996"; "Role-Playing"; "Nintendo"; 11,2
```

`read_csv2()`

Tab stop (“\t”) as separator, “.” as decimal

```
"Name" "Platform" "Year_of_Release" "Genre" "Publisher"  
"Wii Sports" "Wii" "2006" "Sports" "Nintendo" 41.36  
"Super Mario Bros." "NES" "1985" "Platform" "Nintendo"  
"Mario Kart Wii" "Wii" "2008" "Racing" "Nintendo"  
"Wii Sports Resort" "Wii" "2009" "Sports" "Nintendo"  
"Pokemon Red/Pokemon Blue" "GB" "1996" "Role-Playing" "Nint
```

`read_tsv()`

This slide contains content exclusive to R

Different options for importing in R

Advantages of {readr} over {utils}

- Note that we use the importing functions from {readr}. There are also similar functions that come with base R and need no additional package to work, e.g., `read.delim()` instead of `read_delim()`
- There are several advantages of {readr} :
 - Typically much faster
 - More consistent
 - Easy to use
 - Infers from the data if it contains column headers
 - More reproducible, results do not depend on environment or operating system

There are multiple ways to import flat files

Special case: Very large datasets

Use `fread()` from `{data.table}` package

- Extremely fast
 - Similar to `read.table()`, but easier to use
 - Infers column names, types and separators without manual specification
 - Manual specifications possible if necessary
- Basically an improved version of `read.table()` that is faster, more convenient and customizable

Example: File size ~146 MB (29 variables, ~240k cases)

- ```
slow <- read.csv("gun-violence-data.csv")
Computation time of 2.607658 mins
```
- ```
faster <- read_csv("gun-violence-data.csv")  
Computation time of 4.302173 secs
```
- ```
fastest <- fread("gun-violence-data.csv")
Computation time of 1.757922 secs
```

## A checklist for importing flat files

Correct separators:

" , " or ";" or "\t" or "/" or ...?

Correct decimal:

". ." or ", "?

Are variable names in the first row imported as such?

e.g., `header = TRUE`

Are missing values recognized as such?

e.g., `na.strings = "NA"`

...

# Binary files require specific functions to load

## Binary files

- Frequently, proprietary export file types are not interpretable as plain text but are encoded binarily
- These files are highly standardized but require specific functions to import
- While Pandas features functions for most common file types, we require additional packages when using R

|               | R                                           | Python                         |
|---------------|---------------------------------------------|--------------------------------|
| Excel files   | <code>read_excel()</code><br>from {readxl}  | <code>pd.read_excel()</code>   |
| Stata files   | <code>read_stata()</code>                   | <code>pd.read_stata()</code>   |
| SPSS files    | <code>read_spss()</code>                    | <code>pd.read_spss()</code>    |
| SAS files     | <code>read_sas()</code>                     | <code>pd.read_sas()</code>     |
| Parquet files | <code>read_parquet()</code><br>from {arrow} | <code>pd.read_parquet()</code> |

## Exercises



1. Download and unzip the file *data.zip* from ILIAS. This archive contains all data files from the lecture and the tutorials and will be used throughout the full semester.
2. Install the necessary packages and load them at the beginning of your script.
3. Import the file *vgsales1.csv*, a data set on video game sales, with a function of your choice. Write your code such that
  - a) The following values convert to missing values (NAs): "NA", "N/A", "*tbd*", and empty cells
  - b) The variable *Rating* has the class *factor* in R and the type *category* in Python

## Exercises

more exercises for you to do at home



4. Import the files *vgsales2*, *vgsales3*, *vgsales4*, and *vgsales5*. Again, consider the steps from above. (Hint: Look at the files in a text editor (e.g., Windows Editor or Notepad++) first to figure out the file format.)
5. Import the files *price\_usage.dta*, *ice\_cream.sav*, and *demand.sas7bdat*. Make sure that labelled values from other software packages are transformed into appropriate data types.

# Tidy house, tidy mind!

## Features of tidy data

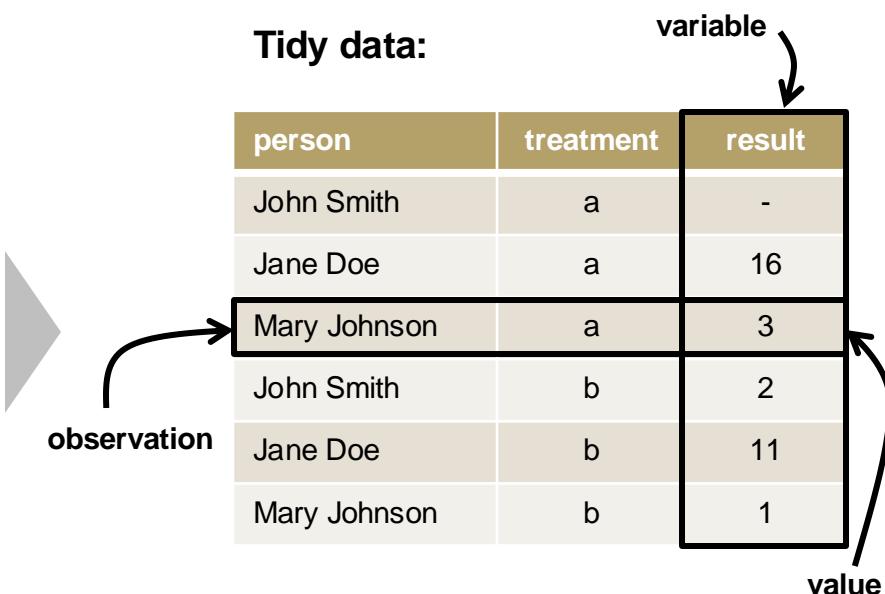
1. Each variable in the data set is placed in its own column.
2. Each observation is placed in its own row.
3. Each value is placed in its own cell

**Messy data:**

|              | treatment_a | treatment_b |
|--------------|-------------|-------------|
| John Smith   | -           | 2           |
| Jane Doe     | 16          | 11          |
| Mary Johnson | 3           | 1           |

|             | John Smith | Jane Doe | Mary Johnson |
|-------------|------------|----------|--------------|
| treatment_a | -          | 16       | 3            |
| treatment_b | 2          | 11       | 1            |

**Tidy data:**



# Some useful functions to tidy messy data

## From long to wide format

- We move the unique values of one column into the column names, spreading the values of one or more other columns across the new columns

| country | month    | variable          | value |
|---------|----------|-------------------|-------|
| A       | jan_2019 | unemployment_rate | 3.0%  |
| A       | jan_2019 | unemp_perc_change | +0.1% |
| ...     | ...      | ...               | ...   |
| A       | dec_2019 | unemployment_rate | 3.6%  |
| A       | dec_2019 | unemp_perc_change | +0.2% |



| country | month    | unemployment_rate | unemp_perc_change |
|---------|----------|-------------------|-------------------|
| A       | jan_2019 | 3.0%              | +0.1%             |
| ...     | ...      | ...               | ...               |
| A       | dec_2019 | 3.6%              | +0.2%             |

R {tidyverse}

Python {pandas}

specify variables using:

- row identifier (index)
- new columns
- values in new columns

`pivot_wider()`

`id_cols = ...`  
`names_from = ...`  
`values_from = ...`

`pd.DataFrame.pivot()`

`index = ...`  
`columns = ...`  
`values = ...`

- If no index is given, {pandas} drops other columns while {tidyverse} keeps them

# Some useful functions to tidy messy data

## From wide to long format

- We move column names into a key column, gathering the column values into a single value column:

| country | jan_2019 | ... | dec_2020 |
|---------|----------|-----|----------|
| A       | 3.0%     | ... | 3.6%     |
| B       | 8.4%     | ... | 5.9%     |



| country | month    | unemployment_rate |
|---------|----------|-------------------|
| A       | jan_2019 | 3.0%              |
| ...     | ...      | ...               |
| A       | dec_2019 | 3.6%              |
| B       | jan_2019 | 8.4%              |
| ...     | ...      | ...               |
| B       | dec_2019 | 5.9%              |

R {tidyverse}

Python {pandas}

specify variables using:

- row index
- columns to be pivoted
- name of the key column
- name of the value column

`pivot_longer()`

`-`  
`cols = ...`  
`names_to = ...`  
`values_to = ...`

`pd.DataFrame.melt()`

`id_vars = ...`  
`value_vars = ...`  
`var_name = ...`  
`value_name = ...`

- {pandas} pivots all columns not in `id_vars` while {tidyverse} uses all columns not in `cols` as row indices

# Some useful functions to tidy messy data

## Split columns

- We separate the text in one column into two columns based on a rule
- In R, this can be done comfortably using:

```
separate(data, col = month, into = c("month", "year"), sep = "_")
```

- In Python, there is no pre-defined function for this. We use this syntax:

```
data[["month", "year"]] = data["month"].str.split("_", expand=True)
```

| country | month    | unemployment_rate |
|---------|----------|-------------------|
| A       | jan_2019 | 3.0%              |
| ...     | ...      | ...               |
| A       | dec_2019 | 3.6%              |
| B       | jan_2019 | 8.4%              |
| ...     | ...      | ...               |
| B       | dec_2019 | 5.9%              |



| country | month | year | unemployment_rate |
|---------|-------|------|-------------------|
| A       | jan   | 2019 | 3.0%              |
| ...     | ...   | ...  | ...               |
| A       | dec   | 2019 | 3.6%              |
| B       | jan   | 2019 | 8.4%              |
| ...     | ...   | ...  | ...               |
| B       | dec   | 2019 | 5.9%              |

# Some useful functions to tidy messy data

## Join columns

- We collapses cells across several columns to make a single column
- In R, this can be done comfortably using:

```
unite(data, month, year, col = "month_year", sep = " ")
```

- In Python, there is no pre-defined function for this. We use this syntax:

```
data["month_year"] = data["month"] + " " + data["year"]
```

| country | month | year | unemployment_rate |
|---------|-------|------|-------------------|
| A       | jan   | 2019 | 3.0%              |
| ...     | ...   | ...  | ...               |
| A       | dec   | 2019 | 3.6%              |
| B       | jan   | 2019 | 8.4%              |
| ...     | ...   | ...  | ...               |
| B       | dec   | 2019 | 5.9%              |



| country | month_year | unemployment_rate |
|---------|------------|-------------------|
| A       | jan 2019   | 3.0%              |
| ...     | ...        | ...               |
| A       | dec 2019   | 3.6%              |
| B       | jan 2019   | 8.4%              |
| ...     | ...        | ...               |
| B       | dec 2019   | 5.9%              |

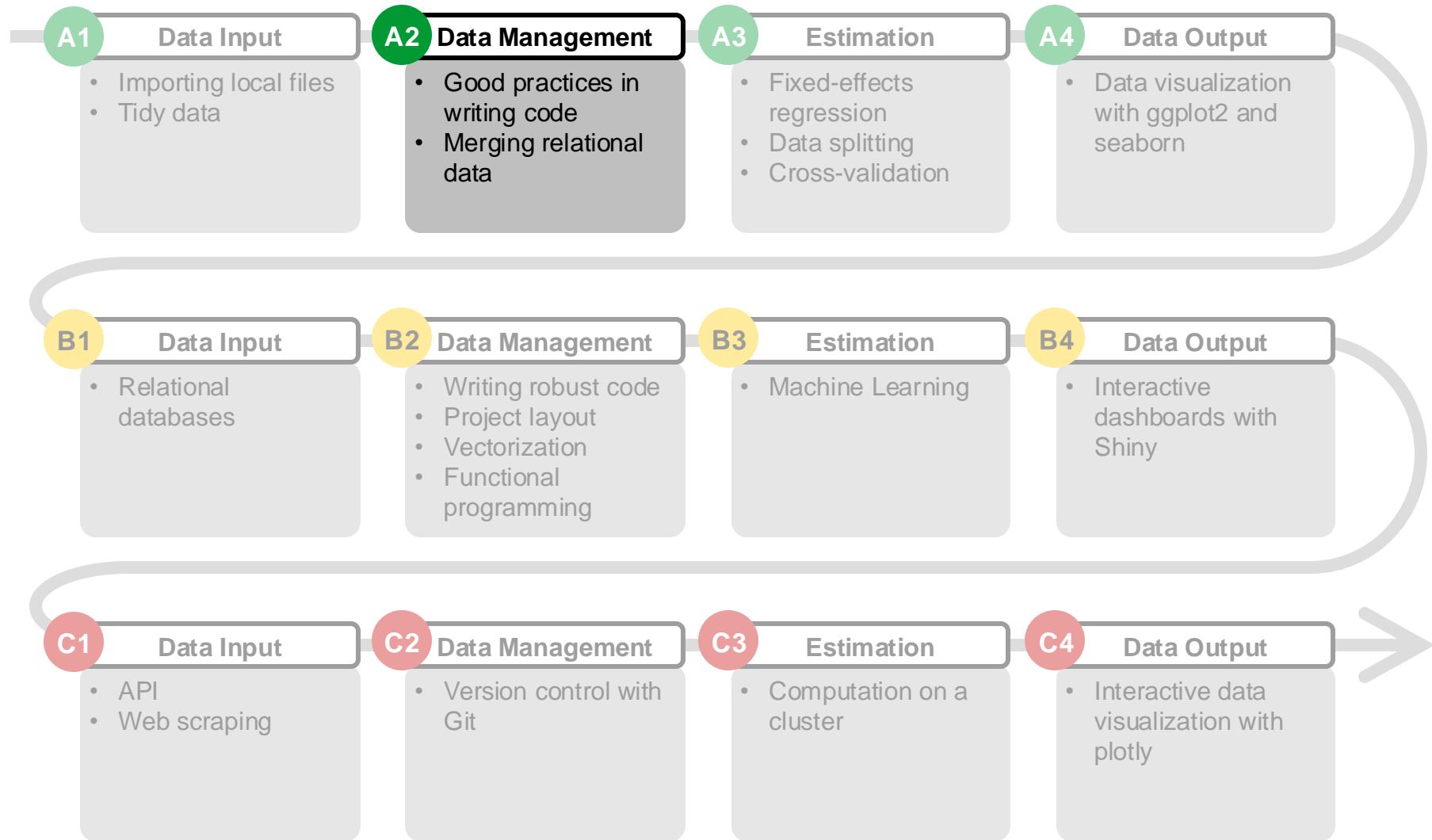
## Exercises



6. Import the file *tuberculosis.csv*, a data set containing information on annual tuberculosis cases in different age brackets on the level of individual countries.

As it is, the data set does not comply with Hadley Wickham's understanding of "tidy data". Make the transformations needed so that the data can be described as "tidy".

# Data Science Project Management: Course Outline

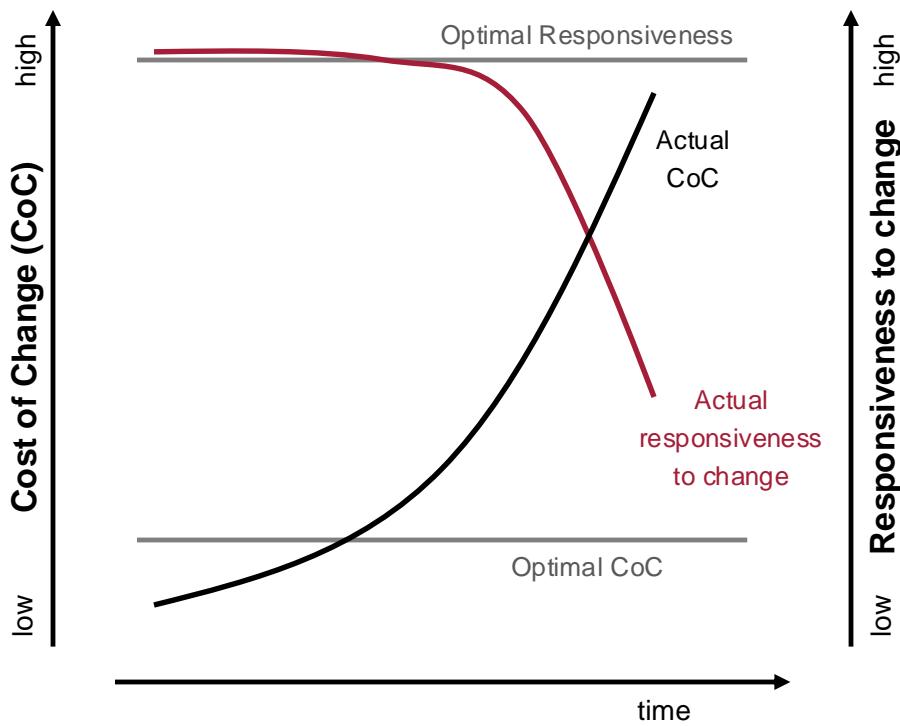


# Why should we worry about clean code?

- Code is clean if it can be understood easily
- With understandability comes:
  - Readability
  - Changeability
  - Extensibility
  - Maintainability

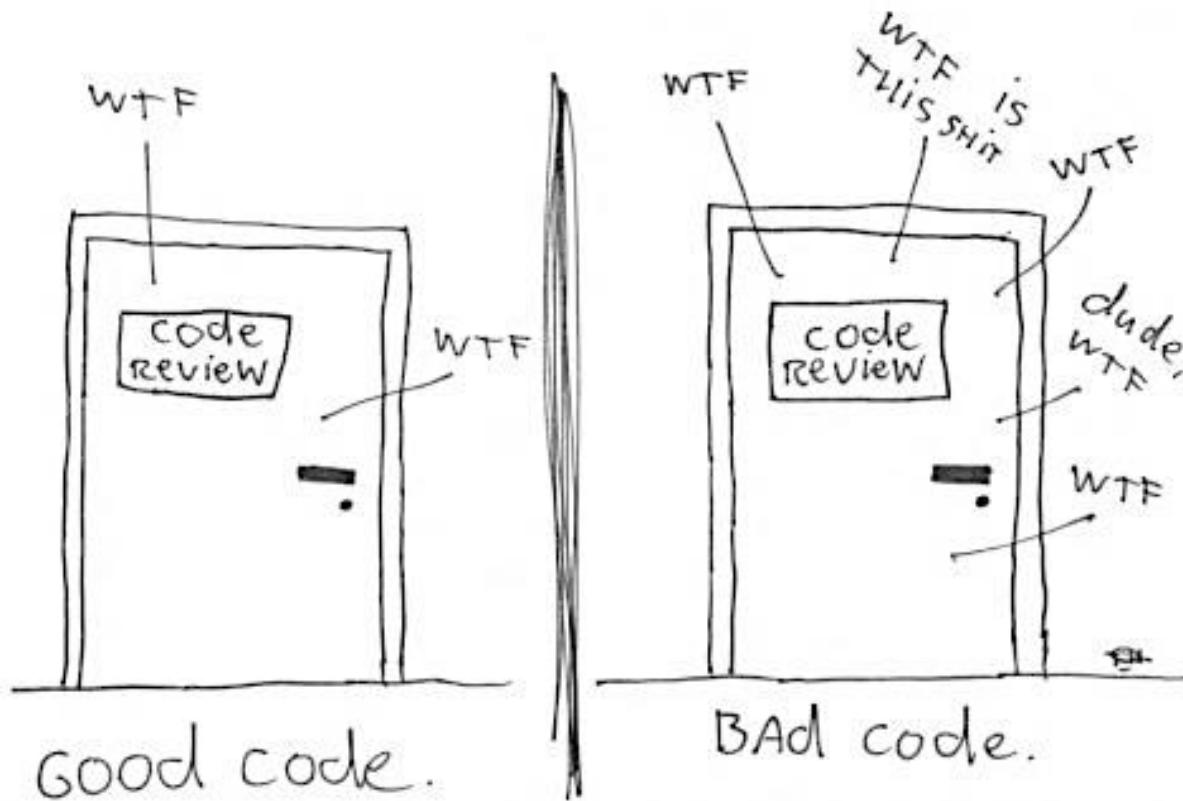
## Clean code: a worthwhile investment

The initial cost of change is a bit higher when writing clean code (red line) than quick and dirty programming (black line), but is paid back quite soon!



<https://www.planetgeek.ch/wp-content/uploads/2013/06/Clean-Code-V2.1.pdf> (last retrieved on September 13, 2018)

# The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/minute



## Good practices in writing code



---

»*Good coding style is like correct punctuation: you can manage without it, but it sure makes things easier to read.*«

— Hadley Wickham  
(Chief Scientist at RStudio & Programming Guru)

---

»*All style guides are fundamentally opinionated. Some decisions genuinely do make code easier to use [...], but many decisions are arbitrary. The most important thing about a style guide is that it provides consistency, making code easier to write because you need to make fewer decisions.*«



## You'll never work alone!

- Every project you are working on is collaborative work between you, your past- and your future-self
- Don't fall for the illusion that you think you can remember each little detail in your code – you won't
- Sticking to coding conventions can save valuable time in understanding your own code

# Good practices in writing code

## Naming

- Names should be concise and meaningful.
- Variable names should be nouns and function names should be verbs.
- Do not use names of already existing functions and variables (TRUE, mean, ...).
- Different naming conventions:
  - `goose_neck`: [variable\\_name](#) (tidyverse style guide recommendation)
  - `ant.notation`: [variable.name](#) (Google Style Guide recommendation)
  - `camelCase`: [variableName](#) (Google Style Guide alternative)
  - `snakeback`: [variablename](#) (not recommended)
- Use an understandable name prefix or suffix for transformed variables  
(e.g. `sales` --> `logSales`)

→ It is not important which style you choose, but choose **one** style and **be consistent!**

<http://style.tidyverse.org/> (last retrieved on October 9, 2018)

<https://google.github.io/styleguide/Rguide.xml> (last retrieved on October 9, 2018)

Kruschke J. K. (2015) *Doing Bayesian Data Analysis*, Second Edition, Academic Press / Elsevier, p. 61f.

# Good practices in writing code

## Spacing & Indentation

- Put spaces around operators such as (=, ==, +, -, <-, etc.)
- Always put a space after a comma, never before one.
- Do not use spaces around code in square brackets or parentheses (`sum(x)` and not `sum( x )`)
- Use a maximum line length of 80 characters. If a function call exceeds that length, use a separate indented line for the function call, each argument and the closing parenthesis.
- Use 2-4 spaces for indentation, not tabs (here's why: <https://bit.ly/29oQu40>). This is especially important in Python
- Always indent the code inside curly braces.

## Assignment

- In R, use `<-`, not `=`, for assignment

<http://style.tidyverse.org/> (last retrieved on October 9, 2018)

<https://google.github.io/styleguide/Rguide.xml> (last retrieved on October 9, 2018)

Kruschke J. K. (2015) *Doing Bayesian Data Analysis*, Second Edition, Academic Press / Elsevier, p. 61f.

# Good practices in writing code

## Comments

- Comment your code if it is not self-explanatory
- Full line comments should start with `#` followed by one space
- Comments after some code need two spaces before `#` and one space after

## Other

- File names should be meaningful
- “If you have copied-and-pasted twice, it’s time to write a function” – Hadley Wickham

Hadley Wickham @hadleywickham Folgen

Time to write a function

Fluff Society @FluffSociety "Ctr+C, Ctr+V, Ctr+V, Ctr+V."

<http://style.tidyverse.org/> (last retrieved on October 9, 2018)

<https://google.github.io/styleguide/Rguide.xml> (last retrieved on October 9, 2018)

Kruschke J. K. (2015) *Doing Bayesian Data Analysis*, Second Edition, Academic Press / Elsevier, p. 61f.

# Good practices in writing code

## Tips & Tricks in RStudio

- Get R Style diagnostics in RStudio:  
Tools -> Global Options -> Code -> Diagnostics -> Provide R style diagnostics
- Automatically reformat code according to spacing and indenting guidelines:
  - Ctrl + Shift + A
- (Un-)Comment multiple lines at once:
  - Ctrl + Shift + C



# Good practices in writing code

## Tips & Tricks in Spyder

- Show code style warnings:
  - Source -> Show code style warnings
- Underline errors and warnings:
  - Source -> Underline errors and warnings
- (Un-)Comment multiple lines at once:
  - Ctrl + 1



Relational data consist of multiple tables containing different information

### Relational data

- It is rare that a data analysis involves only a single table of data
- Instead, a researcher often needs to **combine various tables** to address the research question of interest
- Collectively, **multiple tables of data are called relational data** because it is the relations, not just the individual datasets, that are important
- A relation is a link between a pair of tables

### Keys

- A key is a variable (or set of variables) that uniquely identifies an observation
- It can be used to connect a pair of tables
- Two types of keys:
  - A **primary key** uniquely identifies an observation in its own table.
  - A **foreign key** uniquely identifies an observation in another table.

# Relational data: an example

**Week-level data**  
(weeks):

| week | total Streams |
|------|---------------|
| 20   | ...           |
| 21   | ...           |
| 22   | ...           |
| 23   | ...           |
| ...  | ...           |

**Song-level data**  
(songs):

| id         | duration | energy | tempo  | danceability |
|------------|----------|--------|--------|--------------|
| 0azC730... | 231.0    | .660   | 124.04 | .845         |
| 0ENSn4f... | 221.8    | .747   | 132.55 | .641         |
| 17RdcRY... | 225.2    | .481   | 89.40  | .544         |
| 6rPO02o... | 210.1    | .398   | 120.06 | .723         |
| ...        | ...      | ...    | ...    | ...          |

**Artist-level data**  
(artists):

| artist           | topTen Singles | nationality |
|------------------|----------------|-------------|
| Childish Gambino | 1              | USA         |
| Drake            | 15             | CAN         |
| Post Malone      | 4              | USA         |
| ...              | ...            | ...         |

**Chart data**  
(charts):

| week | rank | id                     | artist           | title           | streams    |
|------|------|------------------------|------------------|-----------------|------------|
| 21   | 1    | 17RdcRYeBFrenw9ktO996Z | Childish Gambino | This Is America | 15,758,906 |
| 21   | 2    | 6rPO02ozF3bM7NnOV4h6s2 | Drake            | Nice For What   | 12,687,517 |
| 21   | 3    | 0azC730Exh71aQlOt9Zj3y | Post Malone      | Better Now      | 11,831,862 |
| 21   | 4    | 0ENSn4fwAbCGeFGVUbXEU3 | Post Malone      | Psycho          | 10,307,897 |
| ...  | ...  | ...                    | ...              | ...             | ...        |

# Relational data: an example

**Week-level data**  
(weeks):

| week | total Streams |
|------|---------------|
| 20   | ...           |
| 21   | ...           |
| 22   | ...           |
| 23   | ...           |
| ...  | ...           |

**Song-level data**  
(songs):

| id         | duration | energy | tempo  | danceability |
|------------|----------|--------|--------|--------------|
| 0azC730... | 231.0    | .660   | 124.04 | .845         |
| 0ENSn4f... | 221.8    | .747   | 132.55 | .641         |
| 17RdcRY... | 225.2    | .481   | 89.40  | .544         |
| 6rPO02o... | 210.1    | .398   | 120.06 | .723         |
| ...        | ...      | ...    | ...    | ...          |

**Artist-level data**  
(artists):

| artist           | topTen Singles | nationality |
|------------------|----------------|-------------|
| Childish Gambino | 1              | USA         |
| Drake            | 15             | CAN         |
| Post Malone      | 4              | USA         |
| ...              | ...            | ...         |

**Primary key**

**Chart data**  
(charts):

| week | rank | id                     | artist           | title           | streams    |
|------|------|------------------------|------------------|-----------------|------------|
| 21   | 1    | 17RdcRYeBFrenw9ktO996Z | Childish Gambino | This Is America | 15,758,906 |
| 21   | 2    | 6rPO02ozF3bM7NnOV4h6s2 | Drake            | Nice For What   | 12,687,517 |
| 21   | 3    | 0azC730Exh71aQlOt9Zj3y | Post Malone      | Better Now      | 11,831,862 |
| 21   | 4    | 0ENSn4fwAbCGeFGVUbXEU3 | Post Malone      | Psycho          | 10,307,897 |
| ...  | ...  | ...                    | ...              | ...             | ...        |

**Foreign key**

# Verifying and creating keys

## Useful functions

- Once primary keys are identified, it's good practice to **verify that they do indeed uniquely identify** each observation



```
data %>% count(key) %>% filter(n > 1)
```



```
data.value_counts("key").where(lambda x: x > 1).dropna()
```

- If a table lacks a primary key, it's sometimes useful to add a **surrogate key**



```
data %>% mutate(key = row_number())
```



```
data["key"] = range(len(data))
```

## Examples



```
> songs %>% count(AlbumId) %>% filter(n > 1) %>% nrow()
[1] 265
```



```
> songs.value_counts("AlbumId") \
.where(lambda x: x > 1).dropna().count()
265
```

# What to do when one key is not enough?

## Song data (songs)

```
> songs %>% count(id) %>% filter(n > 1)

A tibble:## # ... with 2 0 × 2
variables: id <chr>,
n <int>
```

## Chart data (charts)

```
> charts %>% count(id) %>% filter(n > 1)
A tibble: 405 × 2
id n
<fct> <int>
1 00b50UtOuCsj4VFoX6vIZx 23
2 00cP99zN0bsUZSpxbA1QXg 7
3 00iCRJ7pk3onHab1kLyVH1 11
4 00IH8ZjI9ZGB51WRX10esj 8
5 00o6POeoYnf971C6EZ8Fii 3
6 00vqDUiq1UoIEYK1x8s1CS 3
7 00wNUmB4MzvMH1At1hSLzL 34
8 011gUCrvLDn2gEApodohve 2
... with 397 more rows
```

## Chart data (charts)

```
> charts %>%
 count(id, week) %>% filter(n > 1)

A tibble: 0 × 3
... with 3 variables: id <chr>,
week <num>, n <int>
```

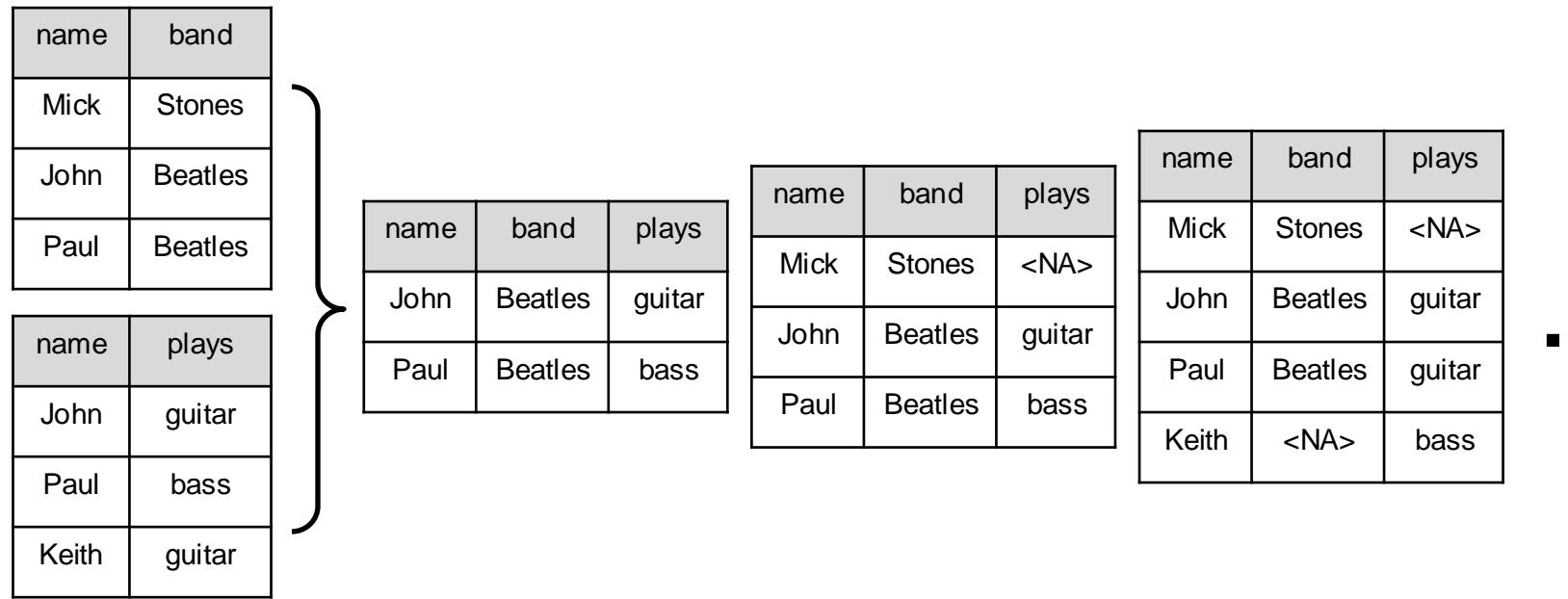
**id + week = primary key**

## Exercises



1. In R, load the packages from the *tidyverse*. In Python, load *Pandas*.
2. Read the files *persons.csv* and *persons\_social.csv* into your environment. Identify the key(s) connecting the two datasets.
3. Check whether each observation is uniquely identified.

# Pairs of tables can be combined in different ways



...?

- **Mutating joins** combine variables from the two data frames  
(we cover four types: inner, left, right and full joins)
- **Filtering joins** keep cases from the left-hand data frame  
(we cover two types: semi and anti joins)

Inner joins combine only matched rows and drop unmatched ones

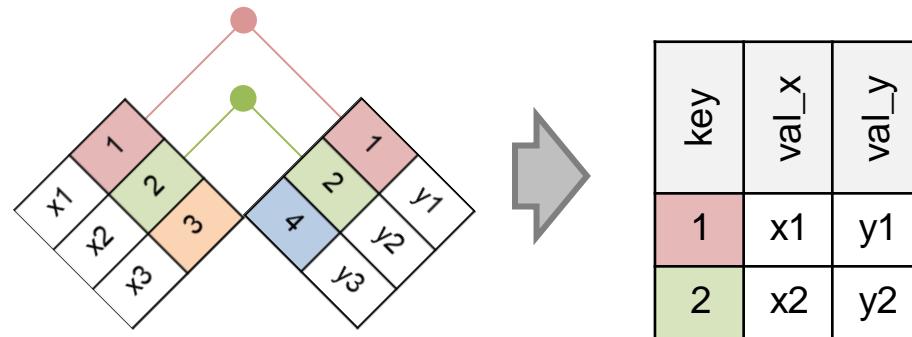
### Mutating joins: Inner joins

- An **inner join** matches pairs of observations whenever their keys are equal:

| x | y  |
|---|----|
| 1 | x1 |
| 2 | x2 |
| 3 | x3 |

| y | x  |
|---|----|
| 1 | y1 |
| 2 | y2 |
| 4 | y3 |



R

```
inner_join(x, y,
 by = "key")
```

Python

```
pd.merge(x, y, how = "inner",
 on = "key")
```

- The most important property of an inner join is that **unmatched rows are not included in the result**
- This means that generally inner joins are usually not appropriate for use in analysis because it's **too easy to lose observations**

# Outer joins exist in three different specifications

| x | y  |
|---|----|
| 1 | x1 |
| 2 | x2 |
| 3 | x3 |

| y  |
|----|
| 1  |
| 2  |
| 4  |
| y1 |
| y2 |
| y3 |

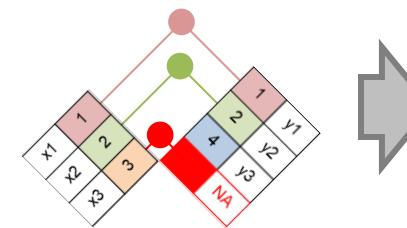
## Mutating joins: Outer joins

- An **outer join** keeps observations that appear in at least one of the tables

A **left join** keeps all observations in x

```
R | left_join(x, y, by = "key")
```

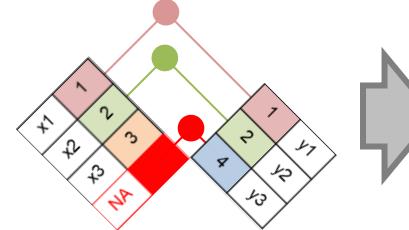
```
Python | pd.merge(x, y, how = "left", on = "key")
```



A **right join** keeps all observations in y

```
R | right_join(x, y, by = "key")
```

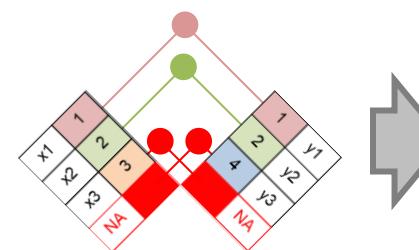
```
Python | pd.merge(x, y, how = "right", on = "key")
```



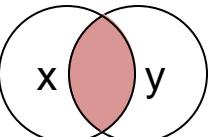
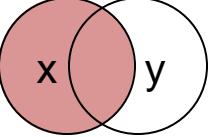
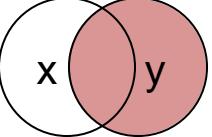
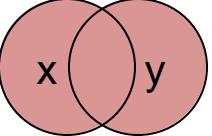
A **full join** keeps all observations in x and y

```
R | full_join(x, y, by = "key")
```

```
Python | pd.merge(x, y, how = "outer", on = "key")
```



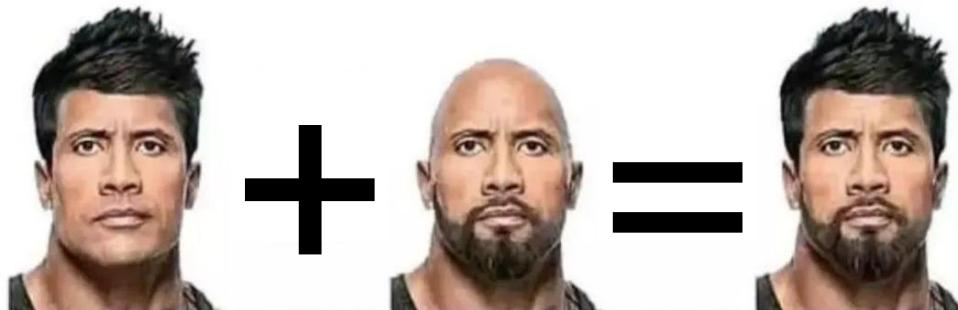
# The dplyr package works more efficiently than the merge() function

|                                                                                   | {dplyr}           | {base}                                   |
|-----------------------------------------------------------------------------------|-------------------|------------------------------------------|
|  | <b>Inner join</b> | <code>  inner_join(x, y)</code>          |
|  | <b>Left join</b>  | <code>  left_join(x, y)</code>           |
|  | <b>Right join</b> | <code>  right_join(x, y)</code>          |
|  | <b>Full join</b>  | <code>  full_join(x, y)</code>           |
|                                                                                   |                   | <code>  merge(x, y)</code>               |
|                                                                                   |                   | <code>  merge(x, y, all.x = TRUE)</code> |
|                                                                                   |                   | <code>  merge(x, y, all.y = TRUE)</code> |
|                                                                                   |                   | <code>  merge(x, y, all = TRUE)</code>   |

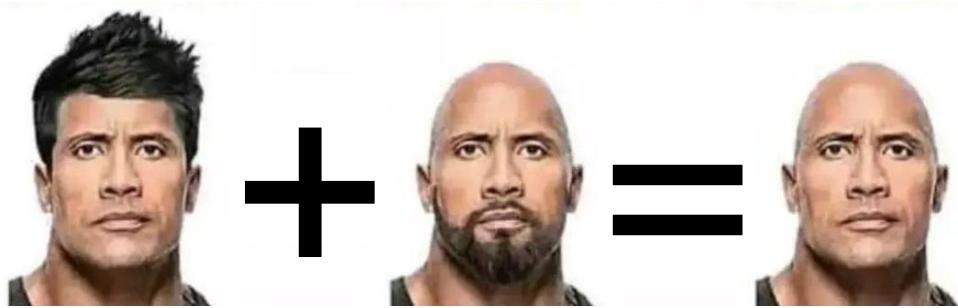
The specific `{dplyr}` verbs have several advantages over `merge()`:

- the intent of your code is conveyed more clearly
- rows are kept in existing order
- `{dplyr}` tells you what keys you're merging by (if you don't supply)
- `{dplyr}`'s joins are considerably faster

## Joins: An illustrative example



**Full outer join**



**Inner join**

inspired by [https://twitter.com/anna\\_pryslopska/status/1538604465506459648](https://twitter.com/anna_pryslopska/status/1538604465506459648) (last retrieved on October 31, 2022)

## Exercises



4. Join the *person.csv* and the *person\_social.csv* data.

Before joining the data, think about how many observations the new data frame should have and check if your expectation is met.

5. Additionally join the *persons\_job.csv* data.

Before joining the data, think about which type of join is appropriate here.

*Hint:* You might need to check the join function's documentation to join the data.

# When you join duplicated keys, you get all possible combinations

## Mutating joins: Duplicate keys

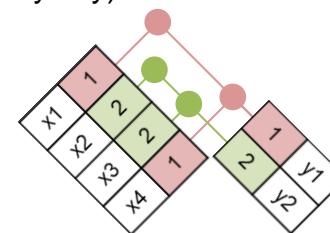
- Join keys are not always unique within a table

### Duplicate keys in one table

| x | y  |
|---|----|
| 1 | x1 |
| 2 | x2 |
| 2 | x3 |
| 1 | x4 |

(key is a foreign key in x and a primary key in y)

```
left_join(x, y, by = "key")
pd.merge(x, y, how = "left",
 on = "key")
```



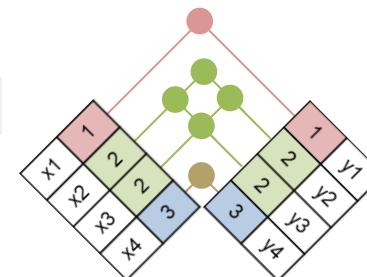
| key | val_x | val_y |
|-----|-------|-------|
| 1   | x1    | y1    |
| 2   | x2    | y2    |
| 2   | x3    | y2    |
| 1   | x4    | y1    |

### Duplicate keys in two tables

| x | y  |
|---|----|
| 1 | x1 |
| 2 | x2 |
| 2 | x3 |
| 3 | x4 |

(key is a foreign key in x and a foreign key in y)

```
left_join(x, y, by = "key")
pd.merge(x, y, how = "left",
 on = "key")
```



| key | val_x | val_y |
|-----|-------|-------|
| 1   | x1    | y1    |
| 2   | x2    | y2    |
| 2   | x2    | y3    |
| 2   | x3    | y2    |
| 2   | x3    | y3    |
| 3   | x4    | y4    |

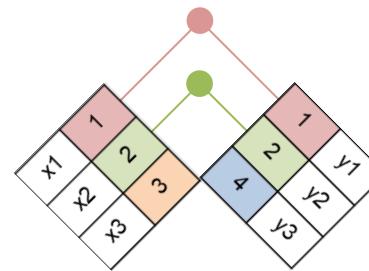
# Semi-joins keep matched cases from the left-hand data frame

## Filtering joins: Semi-joins

- The Semi-join keeps all observations in x that have a match in y

| x |    |
|---|----|
| 1 | x1 |
| 2 | x2 |
| 3 | x3 |

| y |    |
|---|----|
| 1 | y1 |
| 2 | y2 |
| 4 | y3 |

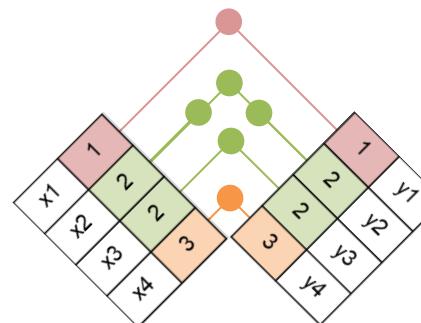


| key | val_x |
|-----|-------|
| 1   | x1    |
| 2   | x2    |

- Only the **existence of a match is important**; it doesn't matter which observation is matched
- This means that **filtering joins never duplicate rows** like mutating joins do

| x |    |
|---|----|
| 1 | x1 |
| 2 | x2 |
| 2 | x3 |
| 3 | x4 |

| y |    |
|---|----|
| 1 | y1 |
| 2 | y2 |
| 2 | y3 |
| 3 | y4 |

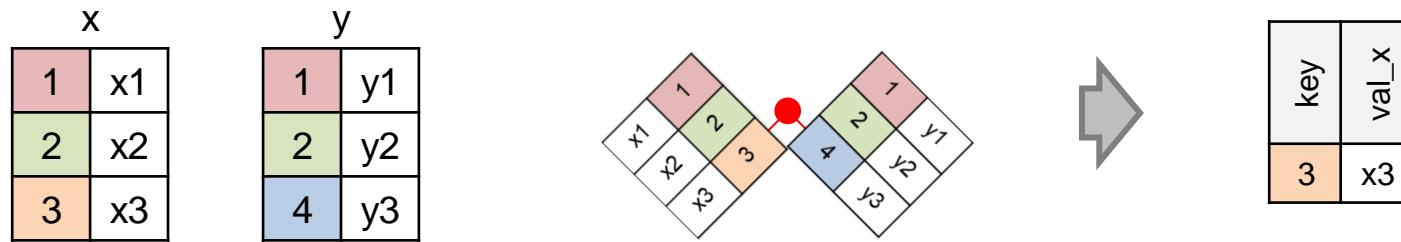


| key | val_x |
|-----|-------|
| 1   | x1    |
| 2   | x2    |
| 2   | x3    |
| 3   | x4    |

# Anti-joins keep the cases that *don't* have a match

## Filtering joins: Anti-joins

- The inverse of a semi-join is an anti-join; an anti-join keeps the rows that **don't have a match**
- It drops all observations in x that have a match in y



- Anti-joins are useful for **diagnosing join mismatches**
- For example, when connecting two data tables, you might be interested to know whether there are many observations in table A that don't have a match in table B

# Filtering joins: Implementation

## Filtering joins: Semi-joins

- {dplyr} features implementations of filtering joins:



```
semi_join(x, y, by = "key")
```

- Within Pandas, pd.merge() does not allow filtering joins.  
Instead, we use:



```
x[x["key"].isin(y["key"])]
```

## Filtering joins: Anti-joins



```
anti_join(x, y, by = "key")
```



```
x[~x["key"].isin(y["key"])]
```

## Exercises



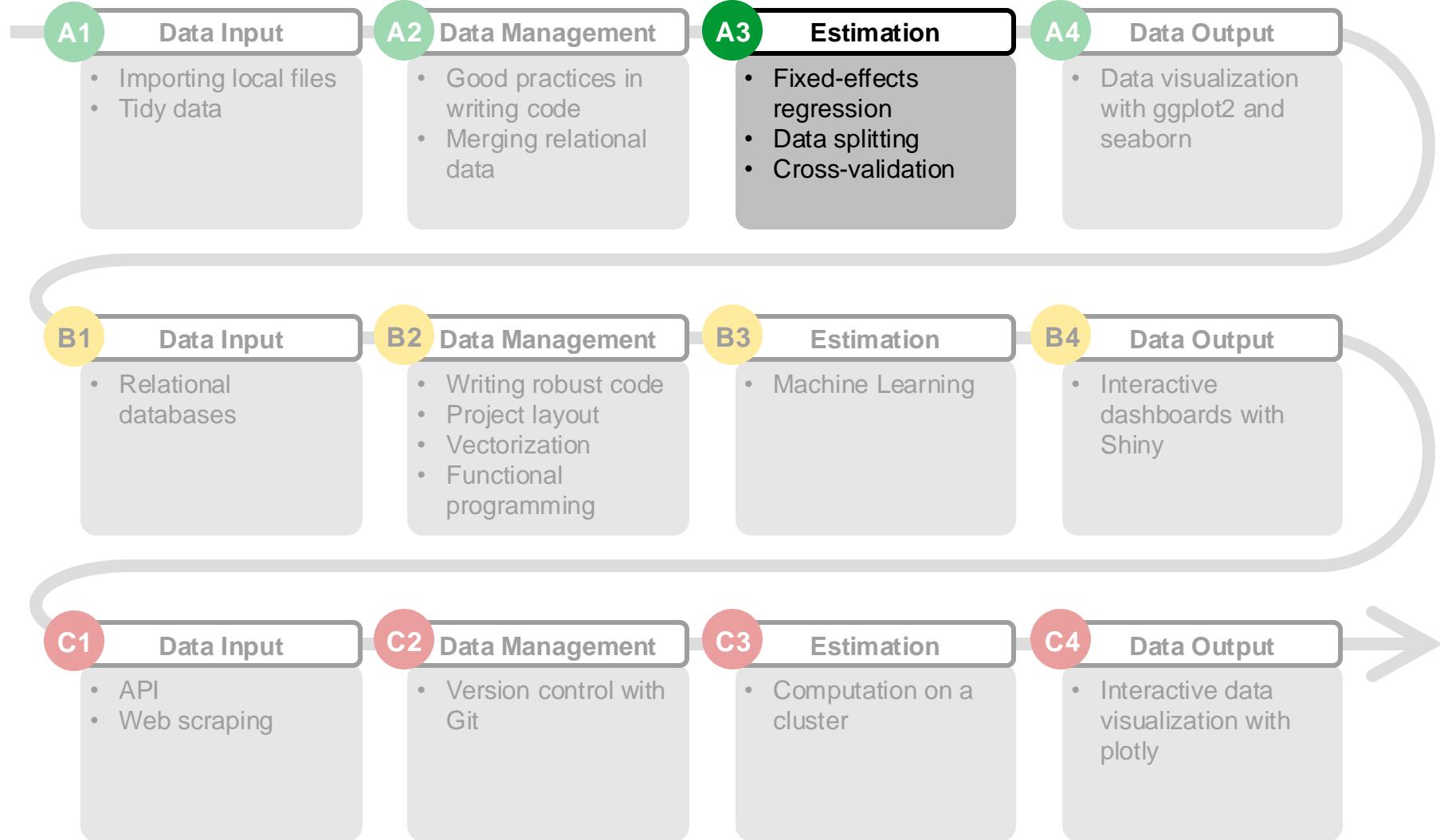
6. Read the files *flights.csv*, *airports.csv*, *airlines.csv*, *weather.csv*, and *planes.csv* into your environment.
7. Familiarize yourself with the data sets and identify the keys connecting:
  - a) *airlines* and *flights*
  - b) *airports* and *flights*
  - c) *weather* and *flights*
8. Does the combination of the date and the flight number uniquely identify observations in flights?

## Exercises



9. Create a new data frame *airportsNew* based on *airports* and *flights*. This data frame should contain the airport information (*lat*, *lon*, *alt*) for those airports that have a match between the *faa* variable from *airports* and the *dest* variable from *flights*.  
Before joining the data, think about how many observations the new data frame should have and then check if your expectation is met.
10. Compute the average arrival delay per destination airport. Store the information in a data frame named *delays*.
11. Join *airportsNew* and *delays* such that the combined data frame contains all columns, but only the matched rows.

# Data Science Project Management: Course Outline



Accounting for different sources of variation can reduce endogeneity concerns

- Often, data have multiple sources of variation
- We refer to these data sets as panel data. This typically means that units are repeatedly measured
  - Stores / brands / respondents / countries and time
  - Students and schools
  - Countries and firms

|        |         | Time     |          |     |          |
|--------|---------|----------|----------|-----|----------|
|        |         | Week1    | Week2    | ... | Week t   |
| Stores | Store1  | $x_{11}$ | $x_{12}$ | ... | $x_{13}$ |
|        | Store2  | $x_{21}$ | $x_{22}$ | ... | ...      |
|        | ...     |          |          |     |          |
|        | Store s | $x_{s1}$ | $x_{s2}$ | ... | $x_{st}$ |



How can we utilize these different sources of variation?

## Fixed-effects regression controls for unobserved heterogeneity

- Consider the following sales response model for a brand of beer across a set of stores  $i$  and time  $t$ :

$$y_{it} = \beta_0 + \beta_1 x_{it} + c_i + \varepsilon_{it}$$

- Let's assume,  $c_i$  is unobserved, i.e., it contains unobserved store characteristics (e.g., management quality, attractiveness of location, income or status of residents nearby)



When will the estimate of  $\beta$  be consistent?

- Because  $c_i$  is unobserved, it becomes part of the error term
- That is no problem as long as  $Cov(x; c) = 0$
- If this assumption is violated, the estimate of  $\beta$  is inconsistent



- Panel data are a powerful way of removing the unobserved effect
- $\beta$  is consistent if  $Cov(x; \varepsilon_{it}) = 0$
- Panel data solve the endogeneity problem that arises from time-invariant unobserved effects

## Removing the unobserved effect: the within-estimator

- Consider the standard panel model:
$$y_{it} = \mathbf{x}_{it}\boldsymbol{\beta} + c_i + \varepsilon_{it}$$
- Time-demeaning removes the unobserved effect
  - Calculate for each  $i$  the mean across all  $t$

$$\bar{y}_i = \bar{\mathbf{x}}_i\boldsymbol{\beta} + c_i + \bar{\varepsilon}_i$$

- Taking the difference of both equations yields

$$y_{it} - \bar{y}_i = (\mathbf{x}_{it} - \bar{\mathbf{x}}_i)\boldsymbol{\beta} + (c_i - c_i) + (\varepsilon_{it} - \bar{\varepsilon}_i)$$

$$\rightarrow y_{it} - \bar{y}_i = (\mathbf{x}_{it} - \bar{\mathbf{x}}_i)\boldsymbol{\beta} + (\varepsilon_{it} - \bar{\varepsilon}_i)$$

$$\ddot{y}_{it} = \ddot{\mathbf{x}}_{it}\boldsymbol{\beta} + \ddot{\varepsilon}_{it}$$

- This equation can be consistently estimated with OLS under the usual assumptions
- Fixed-effects estimation only considers variation over time

# Panel Data Regression: Implementation in R

## A | {stats}

There are multiple ways of estimating linear models in R. Often, the base/{stats} function `lm()` is sufficient. Pooled models can be estimated this way:

```
res <- lm(y ~ x, data)
```

Make sure to convert any fixed effects into a factors to estimate within models:

```
res <- lm(y ~ x + as.factor(t), data)
```

## B | The plm() function

```
library(plm)
res <- plm(y ~ x, index = "t", model = "within", data)
```

- Uses `lm()` on transformed, i.e., (time-)demeaned data
- Recommended if you only have one factor with many levels (e.g., time)

## C | The feols() function

```
library(fixest)
res <- feols(y ~ x | as.factor(i), paneldata)
```

- Fastest for linear models with multiple group fixed effects, factors with a large number of levels



# Panel Data Regression: Implementation in Python

## Python

- We recommend using the {statsmodels} package along with Pandas

```
import statsmodels.formula.api as smf
```

- The formula API of statsmodels allows us to use R-style formulas to define the model. Pooled Models can be estimated like so:

```
mod = smf.ols("y ~ x", data = data)
res = mod.fit()
```

- Including fixed effects works by passing a string variable to the formula. Note: In {statsmodels}, "C(i)" is equivalent to "as.factor(i)"

```
mod = smf.ols("y ~ x + C(i) + C(t)", data = data)
res = mod.fit()
```

- The object of the fitted model contains all relevant information. For example, the summary can be produced like so:

```
res.summary()
```

## Example data set



- Data of 25 large grocery stores that sell beer, among other things.
- Data set contains data per store and per week:
  - The number of six packs of beer sold per week per store in 2011
  - The advertising spending by store allocated to beer in 2011

# How does the data structure look like?

## EXAMPLE OF BEER DATA

| week     | storeID | sales    | price    | adv      | feature | logTemp  | storeSize |
|----------|---------|----------|----------|----------|---------|----------|-----------|
| 1        | 1       | 895      | 3.463413 | 380.028  | 0       | 3.385714 | 530       |
| 2        | 1       | 566      | 3.35452  | 0        | 0       | 6.514286 | 530       |
| ⋮        | ⋮       | ⋮        | ⋮        | ⋮        | ⋮       | ⋮        | ⋮         |
| 51       | 1       | 321      | 3.718569 | 0        | 0       | 4.071429 | 530       |
| 52       | 1       | 534      | 4.660315 | 410.2547 | 0       | 5.55     | 530       |
| 1        | 2       | 667      | 3.078761 | 0        | 0       | 3.385714 | 680       |
| 2        | 2       | 374      | 4.160088 | 338.7377 | 0       | 6.514286 | 680       |
| ⋮        | ⋮       | ⋮        | ⋮        | ⋮        | ⋮       | ⋮        | ⋮         |
| 51       | 2       | 254      | 3.400144 | 278.0995 | 0       | 4.071429 | 680       |
| 52       | 2       | 372      | 4.593462 | 386.2    | 0       | 5.55     | 680       |
| 1        | 3       | 678      | 3.874766 | 508.1367 | 0       | 3.385714 | 618       |
| 2        | 3       | 1070     | 3.718861 | 469.3613 | 0       | 6.514286 | 618       |
| ⋮        | ⋮       | ⋮        | ⋮        | ⋮        | ⋮       | ⋮        | ⋮         |
| 51       | 3       | 530      | 3.725908 | 401.6539 | 0       | 4.071429 | 618       |
| 52       | 3       | 280      | 4.344697 | 0        | 0       | 5.55     | 618       |
| $Y_{it}$ |         | $X_{it}$ |          |          | $X_t$   |          | $X_i$     |

- We can now use these data to estimate the model on a per store / per week basis.
- The number of observations is 25 stores x 52 weeks = 1,300.

## Estimation results of the pooled OLS model in R

```
Call:
lm(formula = logSales ~ logPrice + logAdv + feature + logTemp + storeSize, data = beerSales)

Residuals:
 Min 1Q Median 3Q Max
-1.42748 -0.33732 -0.00362 0.34345 1.55842

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
(Intercept) 8.522e+00 2.320e-01 36.732 <2e-16 ***
logPrice -2.301e+00 1.357e-01 -16.960 <2e-16 ***
logAdv 1.032e-01 4.573e-03 22.575 <2e-16 ***
feature 6.219e-01 3.659e-02 16.995 <2e-16 ***
logTemp 5.149e-01 1.955e-02 26.338 <2e-16 ***
storeSize 4.616e-05 9.463e-05 0.488 0.626

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5073 on 1294 degrees of freedom
Multiple R-squared: 0.5466, Adjusted R-squared: 0.5448
F-statistic: 312 on 5 and 1294 DF, p-value: < 2.2e-16
```

- The model does not know that 52 observations come each from one store, it thinks all observations come from independent units.
- But: Stores may have different strategies, locations, and other unobserved factors that this model does not consider.

## Equivalent approach to the within-estimator: dummy variable regression

- Let us define a set of variables that tell the model from which store the observations come from.

| week | storeID | sales | price    | ... | sd1 | sd2 | sd3 | ... | sd25 |
|------|---------|-------|----------|-----|-----|-----|-----|-----|------|
| 1    | 1       | 895   | 3.463413 | ... | 1   | 0   | 0   | ... | 0    |
| 2    | 1       | 566   | 3.35452  | ... | 1   | 0   | 0   | ... | 0    |
| ⋮    | ⋮       | ⋮     | ⋮        | ... | ⋮   | ⋮   | ⋮   | ... | ⋮    |
| 51   | 1       | 321   | 3.718569 | ... | 1   | 0   | 0   | ... | 0    |
| 52   | 1       | 534   | 4.660315 | ... | 1   | 0   | 0   | ... | 0    |
| 1    | 2       | 667   | 3.078761 | ... | 0   | 1   | 0   | ... | 0    |
| 2    | 2       | 374   | 4.160088 | ... | 0   | 1   | 0   | ... | 0    |
| ⋮    | ⋮       | ⋮     | ⋮        | ... | ⋮   | ⋮   | ⋮   | ... | ⋮    |
| 51   | 2       | 254   | 3.400144 | ... | 0   | 1   | 0   | ... | 0    |
| 52   | 2       | 372   | 4.593462 | ... | 0   | 1   | 0   | ... | 0    |
| 1    | 3       | 678   | 3.874766 | ... | 0   | 0   | 1   | ... | 0    |
| 2    | 3       | 1070  | 3.718861 | ... | 0   | 0   | 1   | ... | 0    |
| ⋮    | ⋮       | ⋮     | ⋮        | ... | ⋮   | ⋮   | ⋮   | ... | ⋮    |
| 51   | 3       | 530   | 3.725908 | ... | 0   | 0   | 1   | ... | 0    |
| 52   | 3       | 280   | 4.344697 | ... | 0   | 0   | 1   | ... | 0    |

Include these 25-1 store dummies into the estimation to account for unobserved differences between stores.

Luckily, we do not have to generate the dummy variables manually

- The storeID variable contains all the information that we need to run the dummy variable regression

| week | storeID | sales | price    | ... |
|------|---------|-------|----------|-----|
| 1    | 1       | 895   | 3.463413 | ... |
| 2    | 1       | 566   | 3.35452  | ... |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 1       | 321   | 3.718569 |     |
| 52   | 1       | 534   | 4.660315 |     |
| 1    | 2       | 667   | 3.078761 |     |
| 2    | 2       | 374   | 4.160088 |     |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 2       | 254   | 3.400144 |     |
| 52   | 2       | 372   | 4.593462 |     |
| 1    | 3       | 678   | 3.874766 |     |
| 2    | 3       | 1070  | 3.718861 |     |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 3       | 530   | 3.725908 |     |
| 52   | 3       | 280   | 4.344697 |     |

R

```
lm(logSales ~ logPrice + logAdv + feature + logTemp +
 storeSize + as.factor(storeID), data = beerSales)
```

storeSize cannot be included in the model any longer because of perfect multicollinearity with storeID

discrete predictor variables are considered in a model by including dummy variables for all but one level

Python

```
smf.ols("logSales ~ logPrice + logAdv + feature +
 logTemp + storeSize + C(storeID)",
 data = data)
```

# The store dummy coefficients are interpreted relative to the base store

| week | storeID | sales | price    | ... |
|------|---------|-------|----------|-----|
| 1    | 1       | 895   | 3.463413 | ... |
| 2    | 1       | 566   | 3.35452  | ... |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 1       | 321   | 3.718569 |     |
| 52   | 1       | 534   | 4.660315 |     |
| 1    | 2       | 667   | 3.078761 |     |
| 2    | 2       | 374   | 4.160088 |     |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 2       | 254   | 3.400144 |     |
| 52   | 2       | 372   | 4.593462 |     |
| 1    | 3       | 678   | 3.874766 |     |
| 2    | 3       | 1070  | 3.718861 |     |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 3       | 530   | 3.725908 |     |
| 52   | 3       | 280   | 4.344697 |     |

```
> FEmodel <- lm(logSales ~ logPrice + logAdv + feature + logTemp +
 as.factor(storeID), data = beerSales)
> summary(FEmodel)

Call:
lm(formula = logSales ~ logPrice + logAdv + feature + logTemp +
 as.factor(storeID), data = beerSales)

Residuals:
 Min 1Q Median 3Q Max
-1.41233 -0.32838 -0.00679 0.33785 1.49508

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
(Intercept) 8.845103 0.252555 35.022 < 2e-16 ***
logPrice -2.498110 0.153068 -16.320 < 2e-16 ***
logAdv 0.103595 0.004617 22.439 < 2e-16 ***
feature 0.510792 0.061418 8.317 2.3e-16 ***
logTemp 0.516162 0.019530 26.429 < 2e-16 ***
as.factor(storeID)2 0.053410 0.099430 0.537 0.59125
as.factor(storeID)3 -0.081906 0.099465 -0.823 0.41039
[...]
as.factor(storeID)25 0.353485 0.124740 2.834 0.00467 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5066 on 1271 degrees of freedom
Multiple R-squared: 0.5558, Adjusted R-squared: 0.546
F-statistic: 56.8 on 28 and 1271 DF, p-value: < 2.2e-16
```



# The store dummy coefficients are interpreted relative to the base store

| week | storeID | sales | price    | ... |
|------|---------|-------|----------|-----|
| 1    | 1       | 895   | 3.463413 | ... |
| 2    | 1       | 566   | 3.35452  | ... |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 1       | 321   | 3.718569 |     |
| 52   | 1       | 534   | 4.660315 |     |
| 1    | 2       | 667   | 3.078761 |     |
| 2    | 2       | 374   | 4.160088 |     |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 2       | 254   | 3.400144 |     |
| 52   | 2       | 372   | 4.593462 |     |
| 1    | 3       | 678   | 3.874766 |     |
| 2    | 3       | 1070  | 3.718861 |     |
| ⋮    | ⋮       | ⋮     | ⋮        | ... |
| 51   | 3       | 530   | 3.725908 |     |
| 52   | 3       | 280   | 4.344697 |     |

```

>>> mod = smf.ols("logSales ~ logPrice + logAdv + feature + logTemp + C(storeID)",
 data = beerSales)
>>> res = mod.fit()
>>> res.summary()
<class 'statsmodels.iolib.summary.Summary'>
"""

OLS Regression Results
=====
Dep. Variable: logSales R-squared: 0.556
Model: OLS Adj. R-squared: 0.546
Method: Least Squares F-statistic: 56.80
Date: Mon, 13 Jun 22 Prob (F-statistic): 2.68e-201
Time: 14:26:30 Log-Likelihood: -945.96
No. Observations: 1300 AIC: 1950.
Df Residuals: 1271 BIC: 2100.
Df Model: 28
Covariance Type: nonrobust
=====
 coef std err t P>|t| [0.025 0.975]

Intercept 8.8451 0.253 35.022 0.000 8.350 9.341
C(storeID)[T.2.0] 0.0534 0.099 0.537 0.591 -0.142 0.248
[...]
C(storeID)[T.25.0] 0.3535 0.125 2.834 0.005 0.109 0.598
logPrice -2.4981 0.153 -16.320 0.000 -2.798 -2.198
logAdv 0.1036 0.005 22.439 0.000 0.095 0.113
feature 0.5108 0.061 8.317 0.000 0.390 0.631
logTemp 0.5162 0.020 26.429 0.000 0.478 0.554
=====
Omnibus: 2.604 Durbin-Watson: 2.036
Prob(Omnibus): 0.272 Jarque-Bera (JB): 2.586
Skew: 0.078 Prob(JB): 0.274
Kurtosis: 2.846 Cond. No. 137.
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""

```

# Forgetting to include fixed effects can't be that dramatic after all, can it?

Consider this simulated dataset:

|      | x        | y         | group |
|------|----------|-----------|-------|
| 1    | 4.281934 | 7.876599  | a     |
| 2    | 4.961272 | 30.564851 | b     |
| 3    | 3.153101 | 24.716568 | c     |
| 4    | 7.192283 | 15.565191 | c     |
| 5    | 7.461786 | 21.833280 | b     |
| 6    | 4.155240 | 9.601801  | a     |
| 7    | 3.904113 | 10.198378 | a     |
| ...  | ...      | ...       | ...   |
| 2994 | 7.078067 | 25.997976 | b     |
| 2995 | 3.492918 | 23.109306 | c     |
| 2996 | 8.041955 | 20.593302 | b     |
| 2997 | 6.649849 | 24.515208 | b     |
| 2998 | 3.320080 | 12.248576 | a     |
| 2999 | 5.108386 | 30.901606 | b     |
| 3000 | 5.185628 | 30.464150 | b     |

We fit a simple regression, ignoring group membership:

$$y_i = \alpha + \beta x_i + \varepsilon_i, \quad \text{for observation } i = 1, \dots, 3000$$

```
> modelNoFE <- lm(y ~ x, data = data)
> summary(modelNoFE)

Call:
lm(formula = y ~ x, data = data)

Residuals:
 Min 1Q Median 3Q Max
-18.2375 -6.5262 -0.0647 5.7127 20.0006

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.87551 0.50274 21.63 <2e-16 ***
x 1.64479 0.09748 16.87 <2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.162 on 2998 degrees of freedom
Multiple R-squared: 0.08673, Adjusted R-squared: 0.08642
F-statistic: 284.7 on 1 and 2998 DF, p-value: < 2.2e-16
```

...looks plausible, doesn't it?!

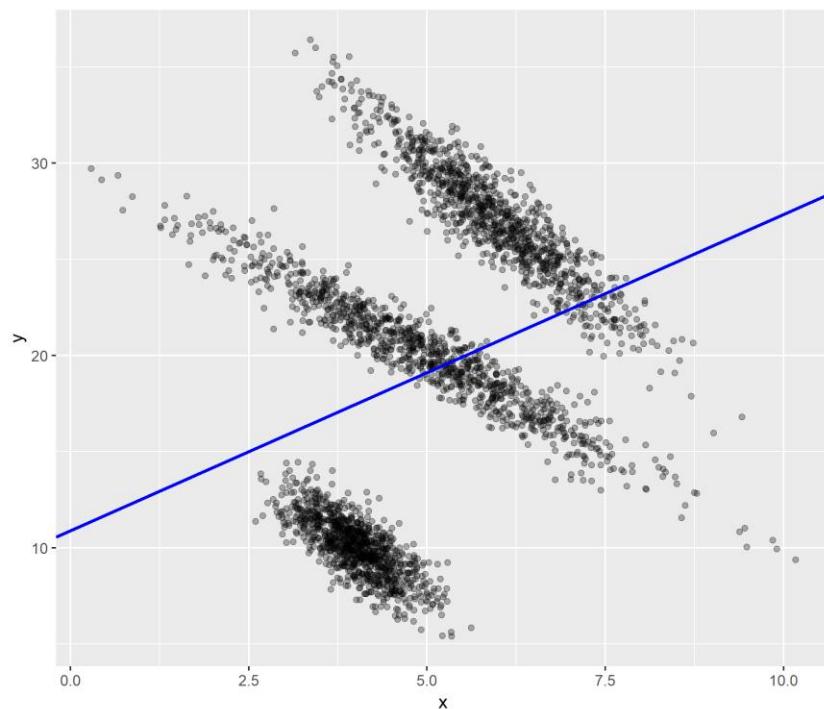
# Forgetting to include fixed effects can't be that dramatic after all, can it?

Consider this simulated dataset:

|      | x        | y         | group |
|------|----------|-----------|-------|
| 1    | 4.281934 | 7.876599  | a     |
| 2    | 4.961272 | 30.564851 | b     |
| 3    | 3.153101 | 24.716568 | c     |
| 4    | 7.192283 | 15.565191 | c     |
| 5    | 7.461786 | 21.833280 | b     |
| 6    | 4.155240 | 9.601801  | a     |
| 7    | 3.904113 | 10.198378 | a     |
| ...  | ...      | ...       | ...   |
| 2994 | 7.078067 | 25.997976 | b     |
| 2995 | 3.492918 | 23.109306 | c     |
| 2996 | 8.041955 | 20.593302 | b     |
| 2997 | 6.649849 | 24.515208 | b     |
| 2998 | 3.320080 | 12.248576 | a     |
| 2999 | 5.108386 | 30.901606 | b     |
| 3000 | 5.185628 | 30.464150 | b     |

We fit a simple regression, ignoring group membership:

$$y_i = \alpha + \beta x_i + \varepsilon_i, \quad \text{for observation } i = 1, \dots, 3000$$



...oops!

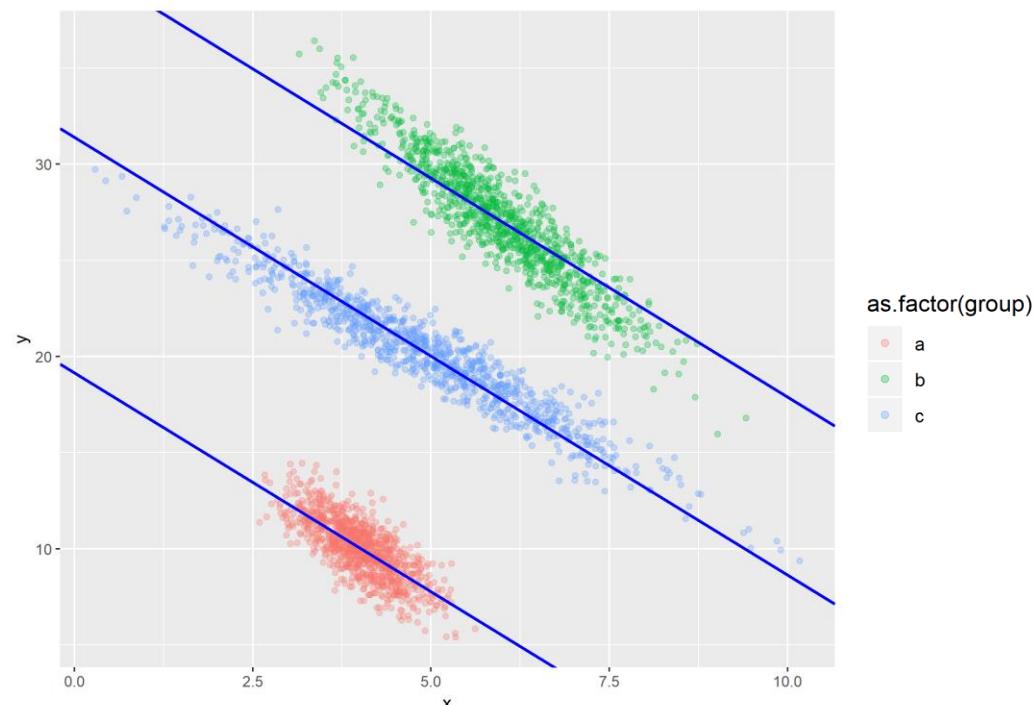
# Including fixed effects reveals a very different picture

Consider this simulated dataset:

|      | x        | y         | group |
|------|----------|-----------|-------|
| 1    | 4.281934 | 7.876599  | a     |
| 2    | 4.961272 | 30.564851 | b     |
| 3    | 3.153101 | 24.716568 | c     |
| 4    | 7.192283 | 15.565191 | c     |
| 5    | 7.461786 | 21.833280 | b     |
| 6    | 4.155240 | 9.601801  | a     |
| 7    | 3.904113 | 10.198378 | a     |
| ...  | ...      | ...       | ...   |
| 2994 | 7.078067 | 25.997976 | b     |
| 2995 | 3.492918 | 23.109306 | c     |
| 2996 | 8.041955 | 20.593302 | b     |
| 2997 | 6.649849 | 24.515208 | b     |
| 2998 | 3.320080 | 12.248576 | a     |
| 2999 | 5.108386 | 30.901606 | b     |
| 3000 | 5.185628 | 30.464150 | b     |

We add group fixed effects  $\eta_j$  to the model:

$$y_{ij} = \alpha + \beta x_{ij} + \eta_j + \varepsilon_{ij}, \quad \text{for observation } i = 1, \dots, 3000 \\ \text{for group } j = 1, \dots, 3$$



## Exercises



1. Load `{tidyverse}` and `{fixest}` in R as well as `{Pandas}` and the `statsmodels.formula` API in Python.
2. Load the data sets `flights.csv` and `weather.csv` (see chapter A2) and merge them.  
Alternatively, load the merged data from `flights_weather_merged.csv`.
3. Estimate a linear model that regresses the arrival delay on the departure delay and the wind speed at the origin airport.
4. Extend the model by accounting for unobserved heterogeneity in the carrier of the corresponding flight (in R, use `Im()` for model estimation).
5. Extend the previous model by also accounting for heterogeneity in the destination airport (in R, use `Im()` to estimate the model).

## Exercises



6. In R, repeat exercises 4 & 5 using `feols()`. Record how long the execution of each function takes. What do you note compared to using the standard `lm()` version?

(Hint: The function `system.time({})` is very useful for speed profiling.)

7. In R, compare the estimates of the effect of `wind_speed` on the `arr_delay` from exercise 4. Did using `feols()` instead of `lm()` change the coefficients or the standard errors?

(Hint: Use the function `tidy()` from the package `{broom}` to turn the regression summary into a neat table.)

## Exercises



8. In Python, look up the documentation for the function `statsmodels.regression.linear_model.OLS()`. Instead of an R formula, it takes the response and independent variables as separate parameters. This is common practice in many other Python packages like `{sklearn}` but also R packages like `{xgboost}`.

Calculate the regression from exercise 5 using this and the formula approach and compare the results.

(Hint: Add fixed effects to the standard model by using `pd.get_dummies()` and remember to add a constant to the model.)

# Descriptive power vs. predictive power

## Descriptive modeling

- Goal is to explain and describe data
- How good does the model fit the complete data set?

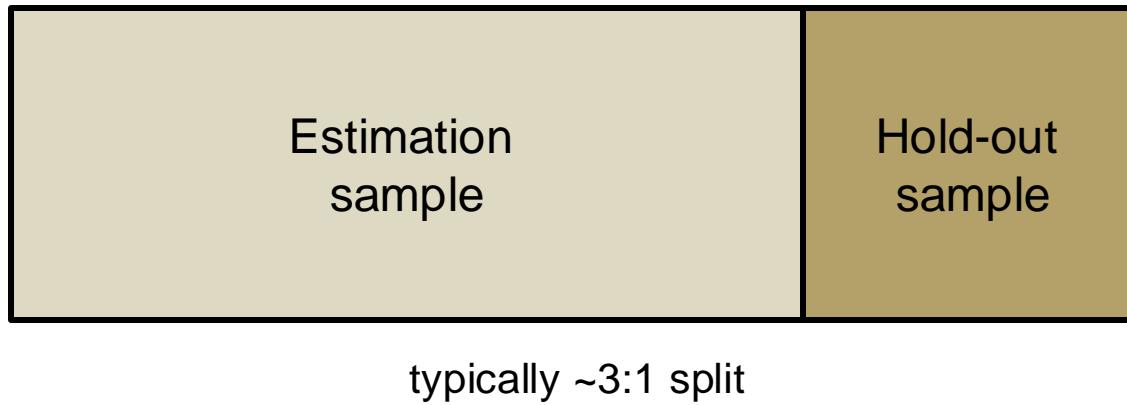
## Predictive modeling

- Goal is to predict unseen data
- How good does the model generalize to new data?

# Data splitting is a simple way of assessing predictive performance

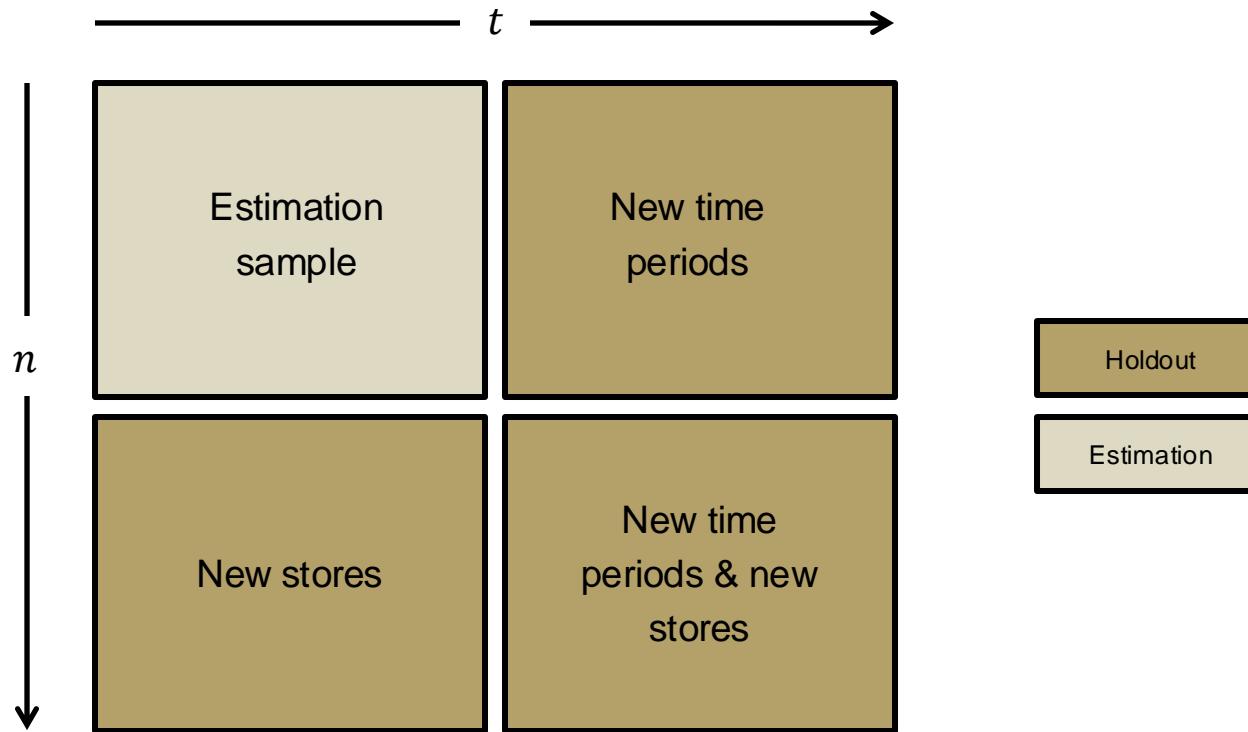
## Data splitting

- Randomly split your data set into two parts
- Estimate a model on the first ~75 % of the data
- Force this model on the remaining ~25 % of the data
- By comparing performance measures for in-sample and out-of-sample predictions, we can tell how well the model generalizes



# Careful with panel data: what do you want to make predictions on?

1. New cross-sectional units (e.g., persons, stores, markets), same time period
2. New time periods, same cross-sectional units
3. Both new cross-sectional units and new time periods



# Data splitting – an example using the video games data

This dataset contains a list of video games with sales greater than 100,000 copies and is based on a web scrape from VGChartz and Metacritic. Complete cases are ~ 6,900.

- **name** - The game's name
- **platform** - Platform of the game's release (i.e., PC, PS4, etc.)
- **yearOfRelease** - Year of the game's release
- **genre** - Genre of the game
- **publisher** - Publisher of the game
- **salesNA** - Sales in North America (in millions)
- **salesEU** - Sales in Europe (in millions)
- **salesJP** - Sales in Japan (in millions)
- **salesOther** - Sales in the rest of the world (in millions)
- **salesGlobal** - Total worldwide sales (in millions)
- **criticScore** - Aggregate score compiled by Metacritic staff
- **criticCount** - The number of critics used in coming up with the **criticScore**
- **userScore** - Score by Metacritic's subscribers
- **userCount** - Number of users who gave the **userScore**
- **developer** - Party responsible for creating the game
- **rating** - The ESRB rating

```
> str(vgsales)
'data.frame': 16719 obs. of 16 variables:
 $ name : chr "Wii Sports" "Super Mario Bros."...
 $ platform : chr "Wii" "NES" "Wii" "Wii" ...
 $ yearOfRelease: chr "2006" "1985" "2008" "2009" ...
 $ genre : chr "Sports" "Platform" "Racing" ...
 $ publisher : chr "Nintendo" "Nintendo" "Nintendo" ...
 $ salesNA : num 41.4 29.1 15.7 15.6 11.3 ...
 $ salesEU : num 28.96 3.58 12.76 10.93 8.89 ...
 $ salesJP : num 3.77 6.81 3.79 3.28 10.22 ...
 $ salesOther : num 8.45 0.77 3.29 2.95 1 0.58 2.88 ...
 $ salesGlobal : num 82.5 40.2 35.5 32.8 31.4 ...
 $ criticScore: int 76 NA 82 80 NA NA 89 58 87 NA ...
 $ criticCount: int 51 NA 73 73 NA NA 65 41 80 NA ...
 $ userScore : num 8 NA 8.3 8 NA NA 8.5 6.6 8.4 NA ...
 $ userCount : int 322 NA 709 192 NA NA 431 129 594 ...
 $ developer : chr "Nintendo" "" "Nintendo" ...
 $ rating : chr "E" " " "E" "E" ...
```

<https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings> (last retrieved on September 21, 2018)

# Where does the variation in the data come from?

- The panel structure in this case does not originate from multiple observations across time, but from multiple observations across platforms:

|          |        | Platform $j$ |            |     |            |
|----------|--------|--------------|------------|-----|------------|
|          |        | Platform 1   | Platform 2 | ... | Platform J |
| Game $i$ | Game 1 | $x_{11}$     | $x_{12}$   | ... | $x_{1J}$   |
|          | Game 2 | $x_{21}$     | $x_{22}$   | ... | $x_{2J}$   |
|          | ...    | ...          | ...        | ... | ...        |
|          | Game I | $x_{I1}$     | $x_{I2}$   | ... | $x_{IJ}$   |

# Data splitting – an example using the video games data

- Consider the following very basic linear regression with platform fixed effects:

$$salesGlobal_{ij} = \beta_0 + \beta_1 criticScore_{ij} + \eta_j + \varepsilon_{ij}, \quad \text{for games } i = 1, \dots, I \\ \text{for platforms } j = 1, \dots, J$$

- However, we do not want to estimate the model on the complete data, but only on a subset. Hence, we split into estimation and hold-out sample:

```
the sample split is a random operation; set.seed() ensures replicability
set.seed(123)
randomly draw 75% of the observations
estIndex <- sample(nrow(vgsales), size = 0.75 * nrow(vgsales), replace = FALSE)
subset dataset
estSample <- vgsales[estIndex,]
we make sure that all groups are represented in the estimation sample, redraw if necessary
n_distinct(vgsales$platform) == n_distinct(estSample$platform)
create holdout sample with the remaining 25%
holdoutSample <- vgsales[-estIndex,]
```

```
randomly draw 75% of the observations
estSample = vgsales.sample(frac = 0.75, random_state = 123)
we make sure that all groups are represented in the estimation sample, redraw if necessary
vgsales[["platform"]].nunique() == estSample[["platform"]].nunique()
create holdout sample with the remaining 25%
holdoutSample = vgsales.drop(estSample.index)
```

# Data splitting – an example using the video games data

- We now estimate the model using only the estimation sample:

 `model <- lm(formula = salesGlobal ~ criticScore + as.factor(platform), data = estSample)`

 `model = smf.ols("salesGlobal ~ criticScore + C(platform)", data = estSample).fit()`

- We now use the model to predict sales for the unseen data in the hold-out sample:

 `predHoldout <- predict(model, newdata = holdoutSample)`

 `predHoldout = model.predict(holdoutSample)`

- ...and compute performance measures for our predictions:

 `library(Metrics) # load package implementing performance measures`

 `mse(holdoutSample$salesGlobal, predHoldout) # mean squared error`  
`rmse(holdoutSample$salesGlobal, predHoldout) # root mean squared error`

 `import statsmodels.api as sm`

 `sm.tools.eval_measures.mse(holdoutSample["salesGlobal"], predHoldout).mean()`  
`sm.tools.eval_measures.rmse(holdoutSample["salesGlobal"], predHoldout).mean()`

- Finally, we compute the same performance measures for the estimation sample (i.e., *in-sample predictions*). These are used for comparison

# Data splitting – an example using the video games data

- Comparing the in-sample and out-of-sample predictive performance gives an idea of the generalizability of the model:

|      | Estimation sample (in-sample prediction) | Hold-out sample (out-of-sample prediction) |
|------|------------------------------------------|--------------------------------------------|
| RMSE | 1.672                                    | 1.877                                      |
| MSE  | 2.798                                    | 3.524                                      |

(Note: These are the results from the estimation in R. The random samples drawn differ because random seeds do work interchangeably between platforms. The python results are however qualitatively identical)

- If out-of-sample predictions are substantially worse than in-sample, it is likely that our model is *overfitted*

## Overfitting

- A model is said to be overfitted if it performs well within the estimation sample, but does not generalize well to the hold-out dataset
- The model is too specific to the estimation data and captures noise instead of true patterns

## Exercises



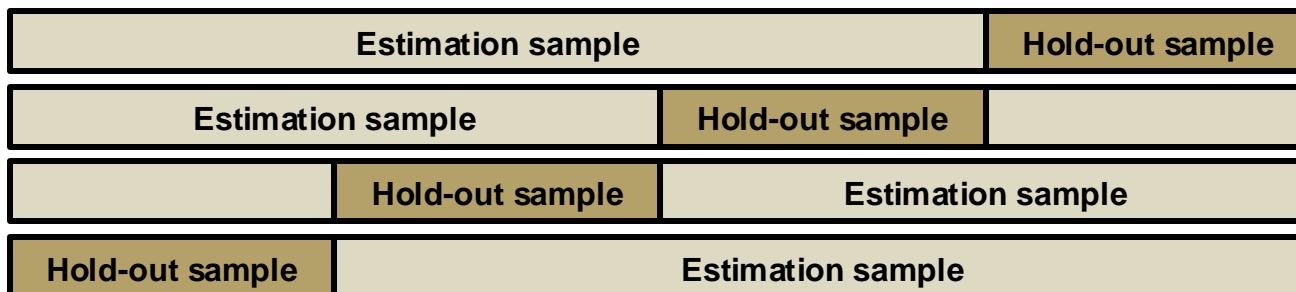
9. In this exercise, we will do a sample split to test our model.
  - a) Estimate the model from exercise 3 only on a subset of the data containing two-thirds of all observations.
  - b) Predict arrival delays for the remaining observations. Evaluate your prediction accuracy using the mean absolute error (MAE).

# Adding robustness to performance measures using cross-validation

- Performance measures can be influenced by sampling and are therefore not stable
- Cross-validation can add robustness to these measures

## k-fold cross-validation

- Data set is split into k folds (in the illustration below: k=4)
- Each fold is  $1/k$  the size of the complete dataset
- Model is estimated multiple times, each time using a different separation of estimation and hold-out sample
- Performance measures on the k hold-out sets are aggregated to form robust measure



[https://s3.amazonaws.com/assets.datacamp.com/course/intro\\_to\\_ml/slides/ch2\\_slides\\_new.pdf](https://s3.amazonaws.com/assets.datacamp.com/course/intro_to_ml/slides/ch2_slides_new.pdf) (last retrieved on September 21, 2018)

# Cross-validation – back to the video games example

```
set.seed(123)

k <- 10 # Define number of folds
n <- nrow(vgsales)
shuffled <- vgsales[sample(n),] # Shuffle order of rows in data set

Initialize the performance measure dataframe
performance <- data.frame(mseHoldout = rep(NA, k), rmseHoldout = rep(NA, k))

for (i in 1:k) {
 # These indices indicate the interval of the hold-out set
 # Fold sizing method borrowed from sklearn
 indices <- ((i-1) * (n %% k) + (i - (i - (n %% k) - 1) * (i > (n %% k))):
 (i * (n %% k) + (i - (i - (n %% k)) * (i > (n %% k)))))

 # Exclude them from the estimation set
 estSample <- shuffled[-indices,]
 # Include them in the hold-out set
 holdoutSample <- shuffled[indices,]

 # A linear model is estimated using each estimation set
 model <- lm(salesGlobal ~ criticScore + as.factor(platform), estSample)
 # Make predictions
 predHoldout <- predict(model, holdoutSample)
 predEst <- predict(model)

 # Assign the performance measures of this model to the respective index DF performance
 performance[i, "mseHoldout"] <- mse(predHoldout, holdoutSample$salesGlobal)
 performance[i, "rmseHoldout"] <- rmse(predHoldout, holdoutSample$salesGlobal)
}

Print out the mean of each performance measure
colMeans(performance)
> mseHoldout rmseHoldout
> 2.987211 1.630414
```



# Cross-validation – back to the video games example

```
Define number of folds
k = 10
n = vgsales.shape[0]

Shuffle order of rows in data set
shuffled = vgsales.sample(frac = 1, random_state = 123)

Initialize the performance measure dataframe
performance = pd.DataFrame({"mseHoldout": [pd.NA] * k, "rmseHoldout": [pd.NA] * k})

for i in range(0,k):
 # These indices indicate the interval of the hold-out set
 # Fold sizing method borrowed from sklearn
 indices = np.arange(i * (n // k) + (i - (i - (n % k) - 1) * (i > (n % k))),
 (i + 1) * (n // k) + (i - (i - (n % k)) * (i > (n % k))))
 # Exclude them from the estimation set
 estSample = shuffled.drop(indices, inplace = False)
 # Include them in the hold-out set
 holdoutSample = shuffled.iloc[indices,]

 # A linear model is estimated using each estimation set
 model = smf.ols(formula = "salesGlobal ~ criticScore + platform", data = estSample)
 res = model.fit()

 # Make predictions
 predHoldout = res.predict(holdoutSample)
 # Assign the performance measures of this model to the respective index DF performance
 performance.loc[i, "mseHoldout"] = \
 sm.tools.eval_measures.mse(predHoldout, holdoutSample.salesGlobal)
 performance.loc[i, "rmseHoldout"] = \
 sm.tools.eval_measures.rmse(predHoldout, holdoutSample.salesGlobal)

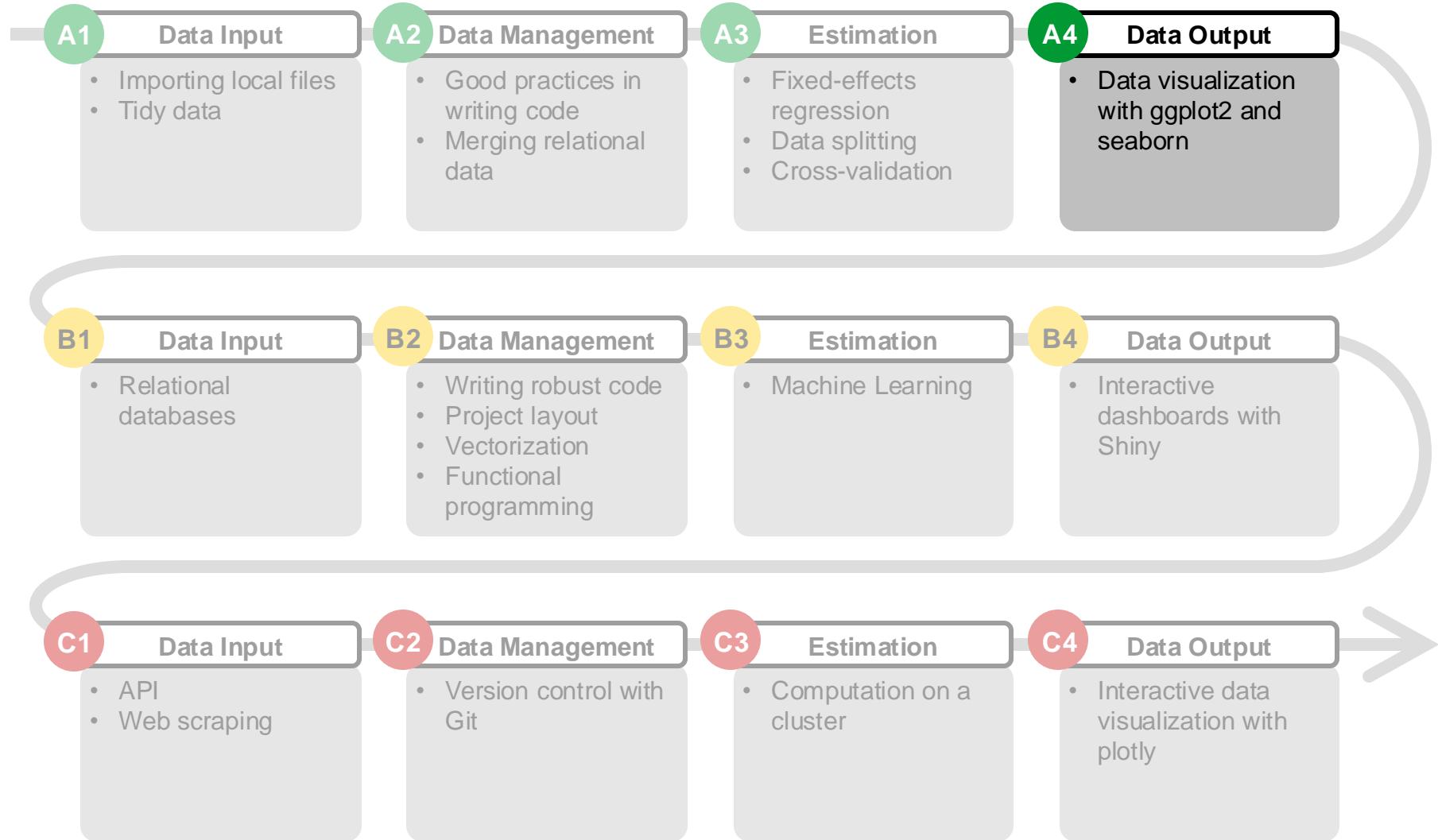
Print out the mean of each performance measure
performance.agg("mean")
mseHoldout 2.997866
rmseHoldout 1.598442
dtype: float64
```

## Exercises



10. Assess the mean absolute error of the predictions of the model from exercise 3 using 10-fold cross-validation.

# Data Science Project Management: Course Outline



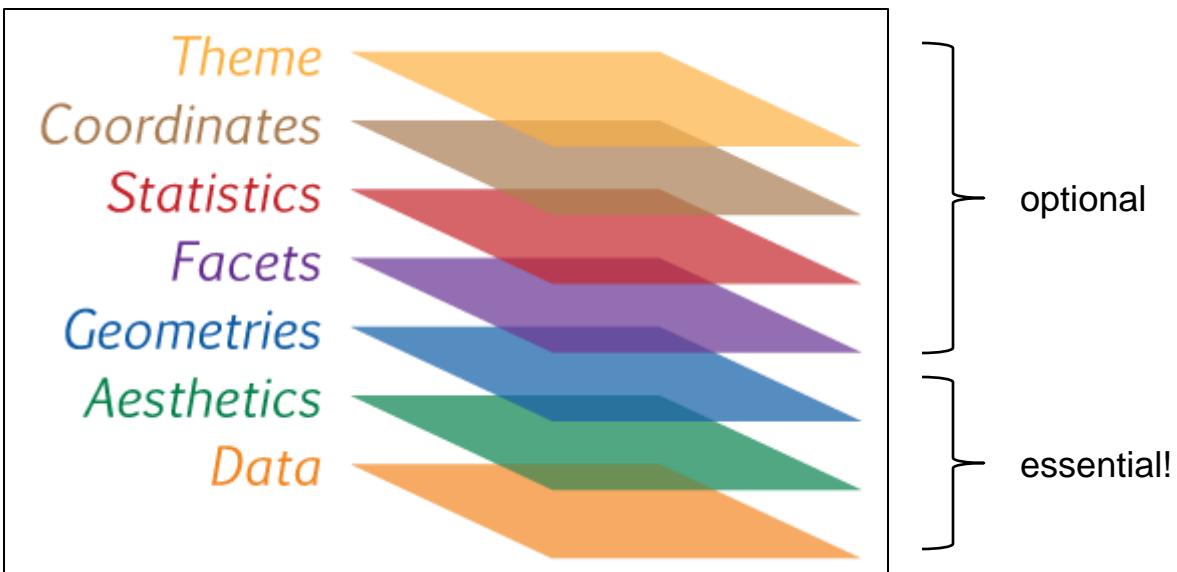
## Goal of this chapter

- This chapter is supposed to ...
  - ... give you an overview of frequently used visualization types
  - ... make you familiar with the general syntax of {ggplot2} in R and {seaborn} in Python
  - ... show how you can customize your plots in both frameworks
  - ... provide you with useful resources that might help you solve your specific visualization problem
- Visualization problems are often highly specific and cannot all be covered in this lecture
- Whatever you can imagine to illustrate with your data – there is probably a way to implement your idea in R or Python!

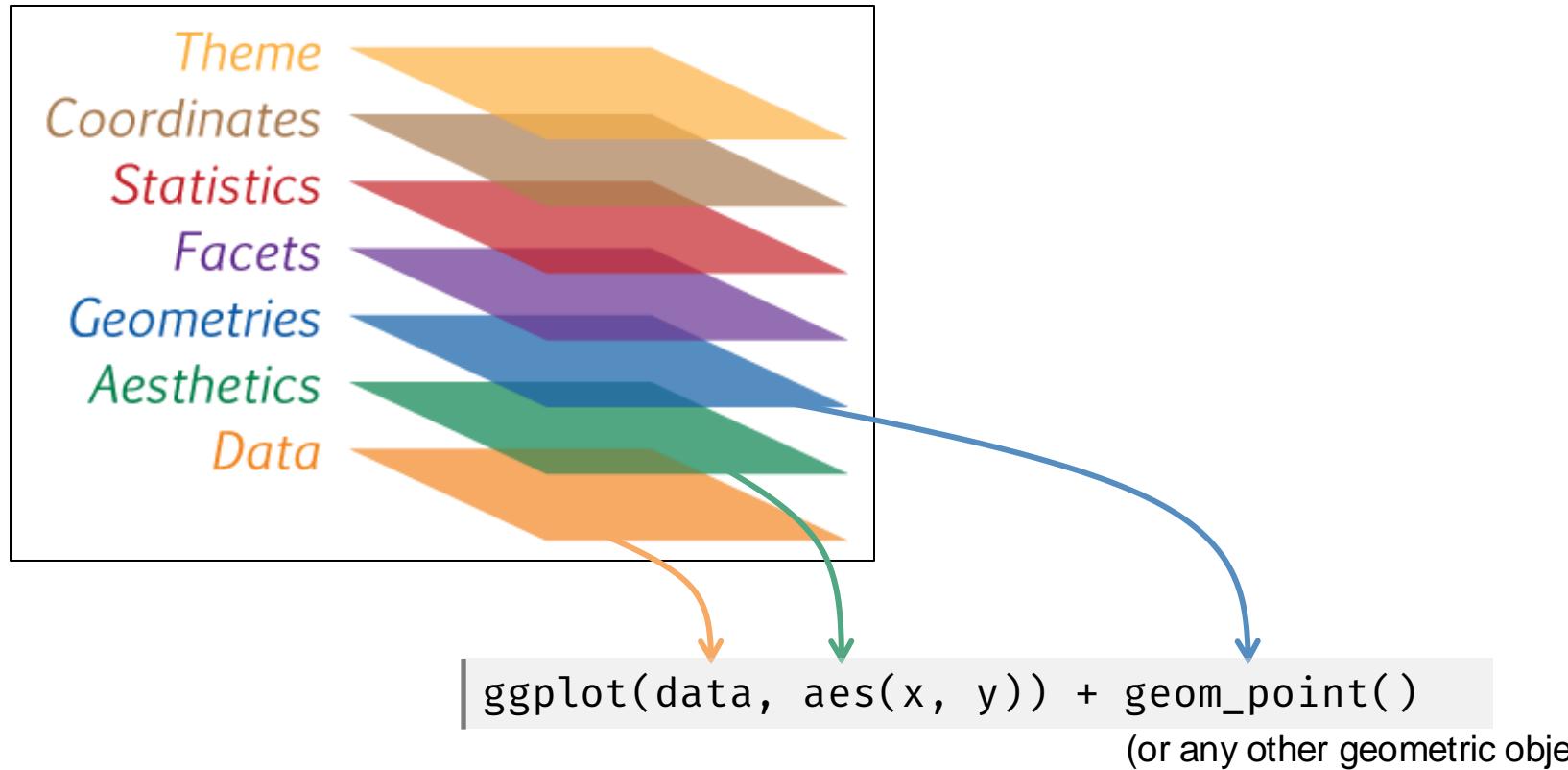
# Implementation

## Visualization in R

- In this lecture, we teach the most elaborate visual framework in R, {ggplot2}
- It is based on the Grammar of Graphics (Wilkinson et. Al., 2005)
- You build a graph by combining different layers



# The `ggplot()` function creates a plot object in `{ggplot2}` syntax



## The `ggplot()` function

- `ggplot()` initializes every ggplot2 object. You then add layers like geometries to it
- It can be used to declare the **input data frame** for a graphic and to specify the **set of plot aesthetics** intended to be common throughout all subsequent layers unless specifically overridden

<https://ggplot2.tidyverse.org/reference/ggplot.html> (last retrieved on October 23, 2018)



# Implementation

## Visualization in Python

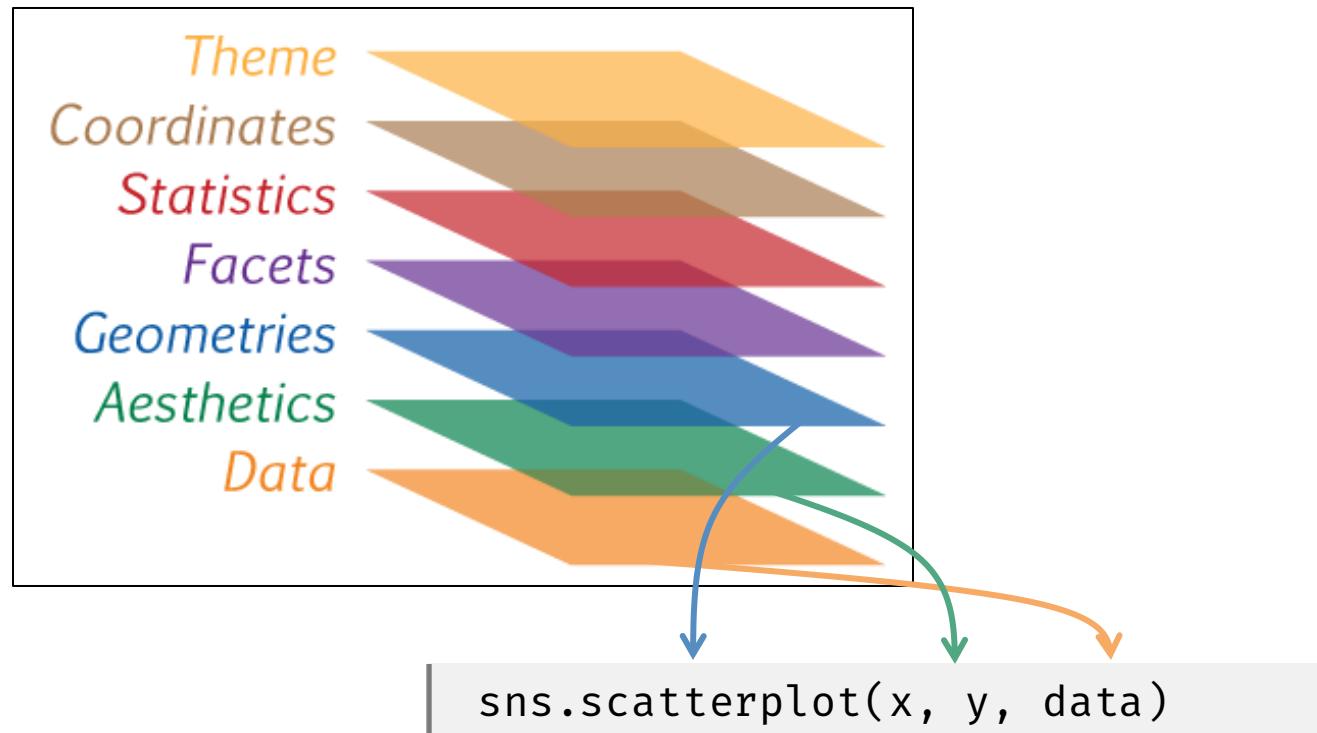
- The most well known Python package for visualization is {matplotlib}
- We use {seaborn}, which uses {matplotlib} in the background and has some nice wrapper functions for frequently used plots
- Keep in mind that {seaborn} follows a different philosophy than {ggplot2} in R. There are substantial differences between the two approaches

## Best practice importing {seaborn}

```
import seaborn as sns
import matplotlib.pyplot as plt
```

- An import of matplotlib alongside seaborn is recommended as you are likely to need it for plot customization

# Applying the layers of {ggplot2} to {seaborn}



(or any other plot type)

## Differences

- You can see that the two approaches work quite differently
- {seaborn} combines plot creation in one command
- Adding different layers to a plot requires the usage of {matplotlib} axes

# Commonly used plots

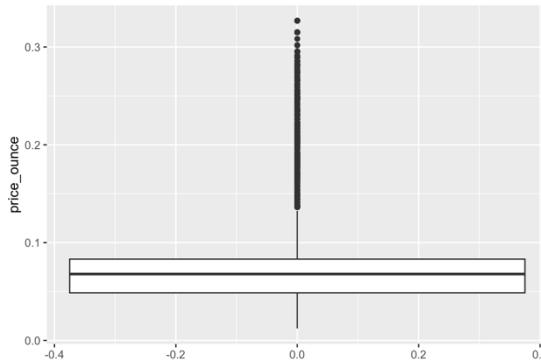
| Type                                                | {ggplot2} geom                                             | {seaborn} function                          | Function                                                                                               |
|-----------------------------------------------------|------------------------------------------------------------|---------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Amount                                              | <code>geom_bar()</code>                                    | <code>barplot()</code>                      | add bars                                                                                               |
| Relationship                                        | <code>geom_point()</code>                                  | <code>scatterplot()</code>                  | add data points                                                                                        |
|                                                     | <code>geom_line()</code>                                   | <code>lineplot()</code>                     | connect data points with a straight line                                                               |
|                                                     | <code>geom_smooth()</code>                                 | <code>regplot()</code>                      | use a “smoother” (i.e., a line that summarizes the data rather than connecting individual data points) |
| Distribution                                        | <code>geom_histogram()</code>                              | <code>histplot()</code>                     | add a histogram                                                                                        |
|                                                     | <code>geom_boxplot()</code>                                | <code>boxplot()</code>                      | add box-whisker diagrams                                                                               |
|                                                     | <code>geom_density()</code>                                | <code>kdeplot()</code>                      | add a density plot                                                                                     |
| Uncertainty                                         | <code>geom_errorbar()</code>                               | added by passing a <code>ci</code> argument | display error bars                                                                                     |
| Other<br>(here, {seaborn}<br>is very<br>restricted) | <code>geom_text()</code>                                   | {matplotlib}'s <code>ax.text()</code>       | add text                                                                                               |
|                                                     | <code>geom_hline()</code> and<br><code>geom_vline()</code> |                                             | add user-defined horizontal or vertical lines, respectively                                            |

See also: the {ggplot2} [Cheat Sheet](#)

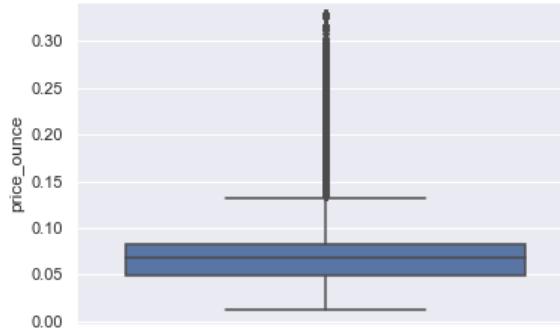
# Visualization with ggplot and seaborn: Examples



```
ggplot(aes(y = price_ounce), data = brandlevel) +
 geom_boxplot()
```



```
sns.boxplot(y = "price_ounce", data = brandlevel)
```



You will use the same  
data in the upcoming  
Exercises

# We use beer sales data for the subsequent visualization examples

## Data description

The data set `brandlevel` contains weekly price and sales information per store aggregated to individual brands

- 169 beer brands
- 82 stores
- 302 weeks

| brand     | week | logmove_ounce | logprice_ounce |
|-----------|------|---------------|----------------|
| BeerLimit | 298  | 7.367709      | -4.006639      |
| BeerLimit | 299  | 7.167038      | -4.049531      |
| ...       | ...  | ...           | ...            |
| Zima      | 322  | 5.886104      | -2.573738      |
| Zima      | 323  | 5.560682      | -2.484679      |

- To start with, we create a subset containing only three brands:



```
threeBrands <- brandlevel %>%
 filter(brand %in% c("OldStyle", "Budweiser", "Miller"))
```

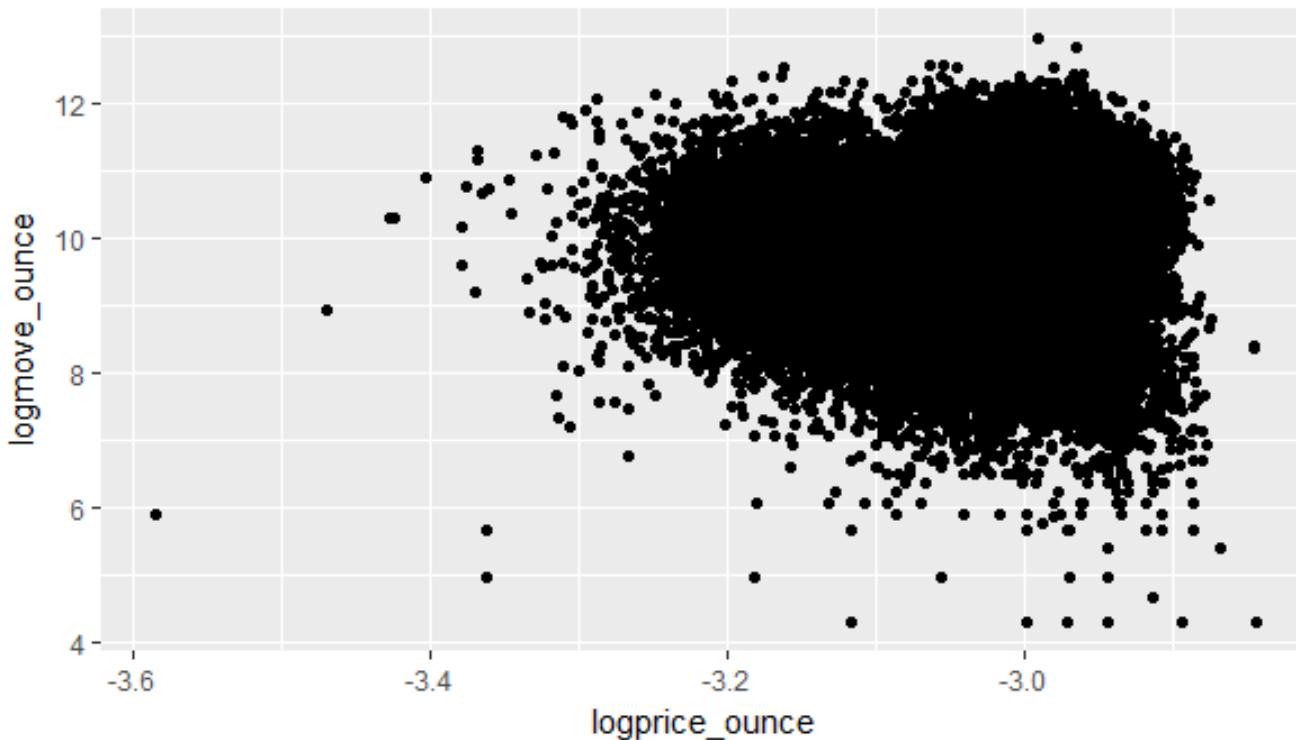


```
threeBrands = brandlevel[
 brandlevel["brand"].isin(["OldStyle", "Budweiser", "Miller"])]
```

We can produce a simple graph using only essential layers

```
ggplot(data, aes(x, y)) + geom_point()
```

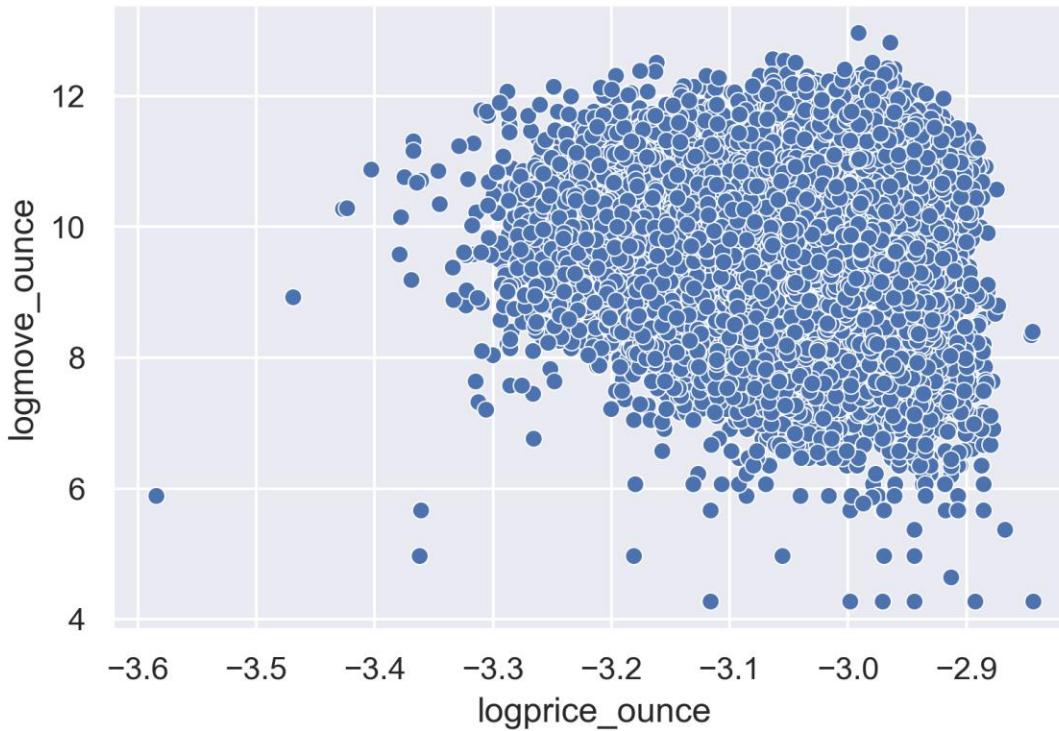
```
ggplot(threeBrands, aes(x = logprice_ounce, y = logmove_ounce)) + geom_point()
```



We can produce a simple graph using only essential layers

```
sns.scatterplot(x, y, data)
```

```
sns.scatterplot(x = "logprice_ounce", y = "logmove_ounce", data = threeBrands)
```



## Exercises



1. The file *brandlevel.csv* contains a subset of a large scanner panel data set provided by the James M. Kilts Center (2008). Build a subset containing the brands *Coors*, *HeinekenBeer*, and *SamAdams*. Store the data in a new data frame named *threeBrands*.
2. Using the subset of data on the three brands, plot the logarithm of the price per ounce (*logprice\_ounce*) against the logarithm of the amount sold (*logmove\_ounce*).
3. Modify the previous plot by visualizing each brand in its own color.

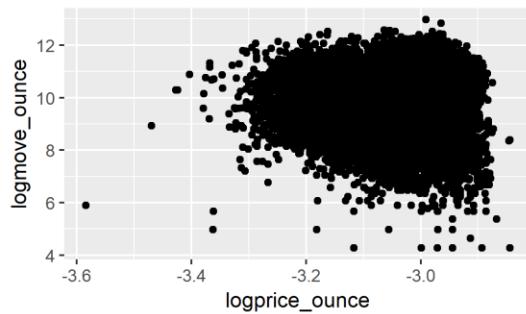
# In R, our code is more flexible if we use a two-step approach

- First, we initiate a ggplot object with data and scales (this object can be “reused”)
- In a second step, we add the geometric layer(s)

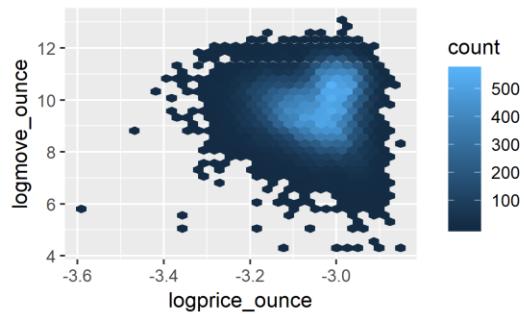
```
STEP 1: Create plot object (data and necessary aesthetics)
plotObj <- ggplot(threeBrands, aes(x = logprice_ounce, y = logmove_ounce))
```

```
STEP 2: Add geometric object(s) and (optional) settings
```

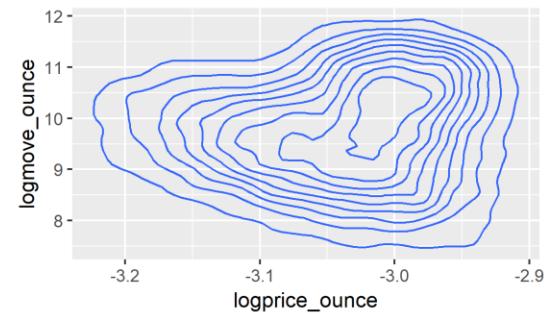
`plotObj + geom_point()`



`plotObj + geom_hex()`



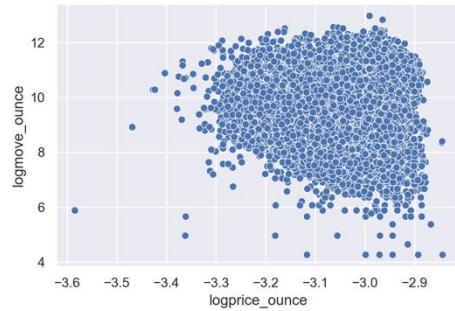
`plotObj +  
geom_density2d()`



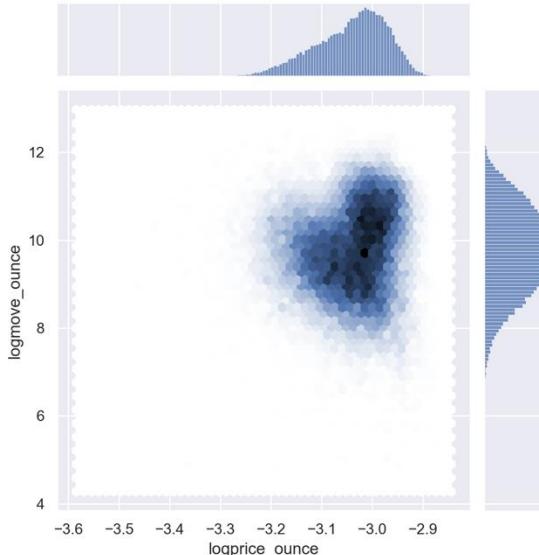
# In Python, we must call individual functions

- {seaborn} can also create these plots, albeit using its individual functions

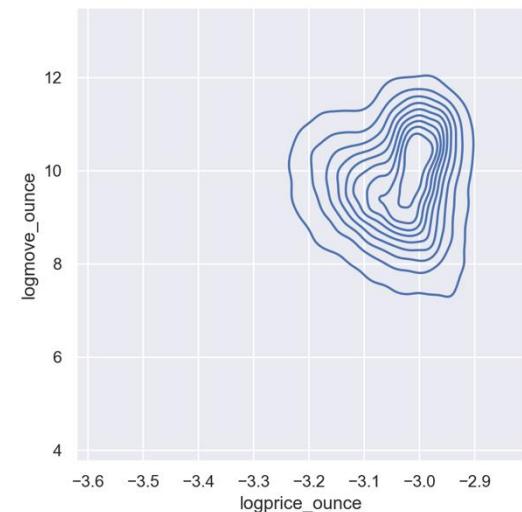
```
sns.scatterplot(
 x = "logprice_ounce",
 y = "logmove_ounce",
 data = threeBrands)
```



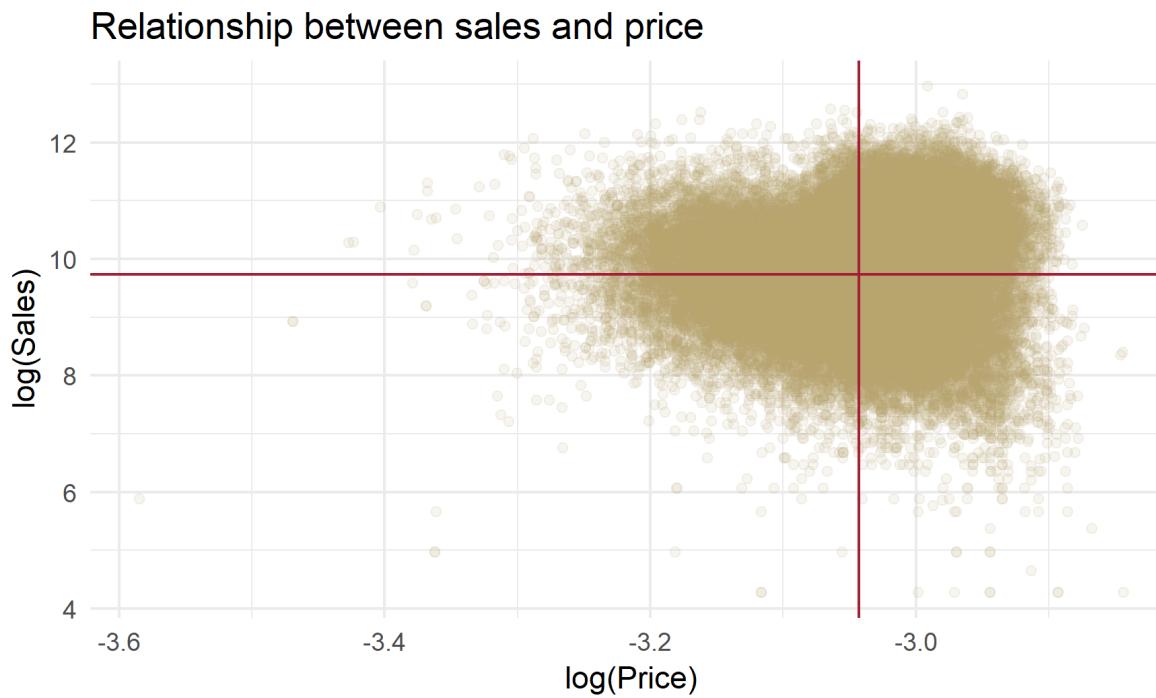
```
sns.jointplot(kind = "hex",
 x = "logprice_ounce",
 y = "logmove_ounce",
 data = threeBrands)
```



```
sns.displot(kind = "kde",
 x = "logprice_ounce",
 y = "logmove_ounce",
 data = threeBrands)
```



In R, we can increase the complexity of the graph by adding further layers



```
plotObj +
 # display data as scatterplot with 10% opacity
 geom_point(alpha = 0.1, col = "#b4a069") +
 geom_vline(xintercept = mean(threeBrands$logprice_ounce), col = "#a51e37") + # vert. line
 geom_hline(yintercept = mean(threeBrands$logmove_ounce), col = "#a51e37") + # horiz. line
 labs(x = "log(Price)", y = "log(Sales)") + # adjust axis labels
 ggtitle("Relationship between sales and price") + # add title
 theme_minimal() # white background
```

# Layering in {seaborn}

## Multiple visual *features* in one plot

- In {ggplot2} you can just add more layers to the plot using the `+` operator
- In {seaborn}, you cannot ‘add’ plots but you must specify the axes you want to plot onto. Keep in mind that axes are a feature of {matplotlib}



```
ggplot(data, aes(x, y)) +
 geom_point() +
 geom_smooth()
```



```
axes = plt.axes()
sns.scatterplot(x = "x", y = "y", data = data, ax = axes)
sns.regplot(x = "x", y = "y", data = data, ax = axes)
```

# The position of the aesthetic arguments matters!

## Setting vs. mapping

Depending on where it is specified, the color option can take different types of arguments and is interpreted differently:

- Inside the `aes()` function:

*Map colors to a variable contained in data (e.g., `color = brand`)*

```
Map color to variable globally, i.e. for all subsequent geometric layers
ggplot(data, aes(x, y, col = _____)) + geom_point()
```

```
Map color to variable locally, i.e. only for one specific geometric layer
ggplot(data, aes(x, y)) + geom_point(aes(col = _____))
```

- Outside the `aes()` function:

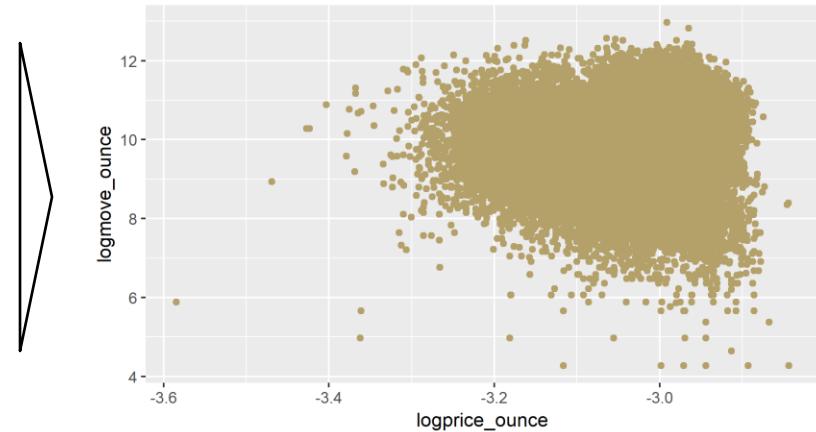
*Set colors using the name of a color either in words<sup>1</sup> or as Hex color code<sup>2</sup> (e.g., `color = "red"`, or `color = "#a51e37"`)*

```
Set color of one geometric layer to a specific value
ggplot(data, aes(x, y)) + geom_point(col = _____)
```

<sup>1</sup> <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf> <sup>2</sup> <https://htmlcolorcodes.com/>

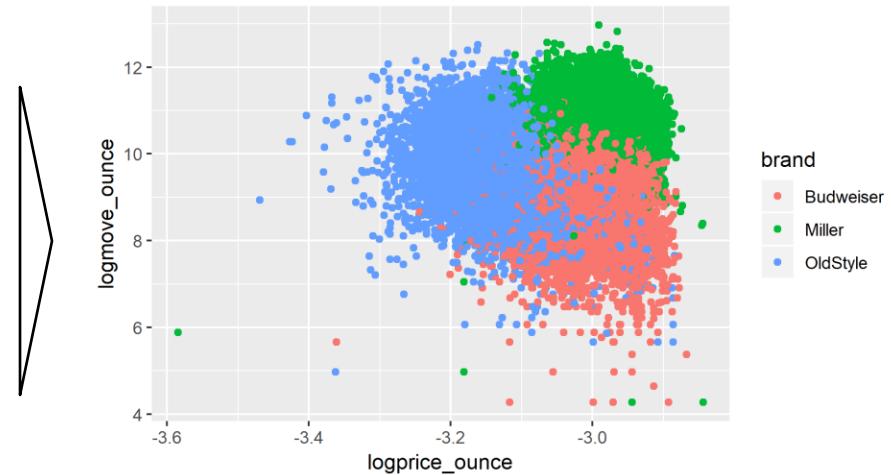
# Mapping allows us to identify observations from different groups

```
Setting color to a specific value
ggplot(threeBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce)) +
 geom_point(color = "#b4a069")
```



```
Mapping color to brands locally
ggplot(threeBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce)) +
 geom_point(aes(color = brands))
```

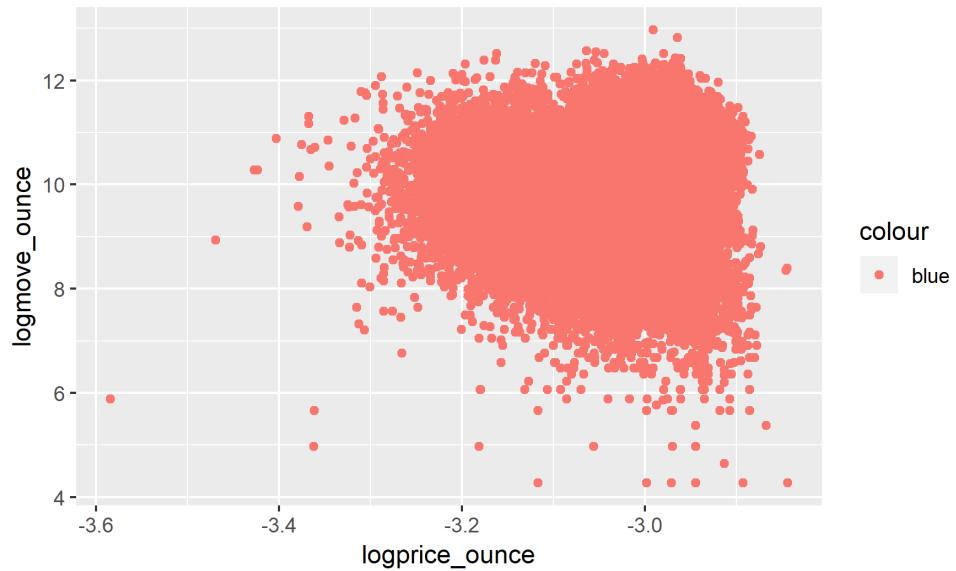
```
Mapping color to brands globally
ggplot(threeBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce,
 color = brands)) +
 geom_point()
```



# Strange things happen if mapping and setting are confounded

```
Attempting to erroneously
set color to "blue" inside
the aes() function with:

ggplot(threeBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce,
 color = "blue")) +
 geom_point()
```



## What went wrong here?

- The color option is specified inside the `aes()` function → ggplot expects a variable as argument
- Since there is no variable called "blue" in `threeBrands`, ggplot implicitly introduces one
- ggplot is completely ignoring the meaning of the word "blue" and rather uses it for categorization of the points → all points belong to the category "blue"

## Exercises



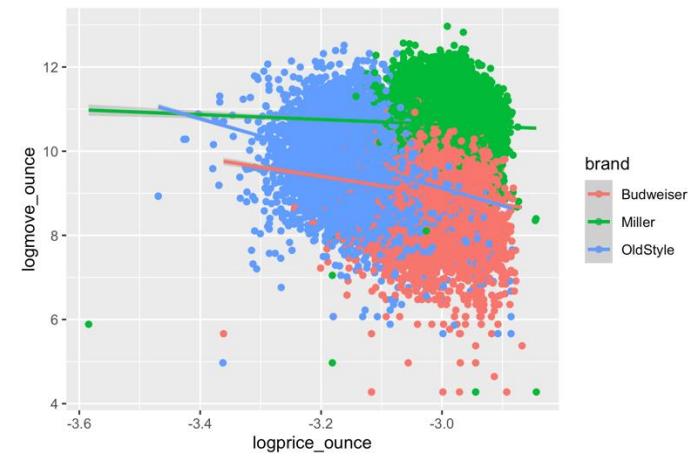
4. Redesign the plot from Exercise 2 to include one linear regression line per brand. Improve clarity by making the points transparent.  
You will find different elasticities between price and sales for each of the three brands. For which brand is the elasticity most negative?

## Exercises



5. Consider the following code. What needs to be changed in order to have only one regression line (all else equal)?

```
ggplot(threeBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce,
 color = brand)) +
 geom_point() +
 geom_smooth(method = "lm")
```



# Some commonly used customization

## Labels



```
ggplot() + labs(title = "Title", x = "xaxis", y = "yaxis")
```



```
ax = sns.*plot()

ax.set_title("title")
ax.set_xlabel("xaxis")
ax.set_ylabel("yaxis")
```

## Color palettes



```
ggplot() + scale_color_brewer(palette = "Dark2")
```



```
sns.*plot(x = "x", y = "y", hue = "c",
 palette = "colorblind", data = data)
```

# Some commonly used customization

## Axis ticks



```
ggplot() + scale_*_(breaks = c(1, 2, ...))
```



```
ax = sns.*plot()
ax.set_ticks([1, 2, ...])
```

## Axis limits



```
dropping all data outside limits
ggplot() + scale_*_continuous(limits = c(-1, 1))
OR: "zooming in" on limits
ggplot() + coord_cartesian(*lim = c(-1, 1))
```



```
ax = sns.*plot()
ax.set_xlim([-1, 1])
```

## Exercises



6. Consider the plot from exercise 3. The three colors used are part of each package's default color palette. Change the colors used to indicate group membership by
- using a predefined colorblind-safe palette
  -  manually specifying three different colors.

## Exercises



7. Build a subset containing the brand *Budweiser*. Store the data in a new object called *budweiser*.
8. In the *budweiser* data, visualize the relationship between sales and price. Map color to week (a numeric variable). What difference does it make to transform color discretely? What is more reasonable here? Can you make out a certain trend based on the graph?

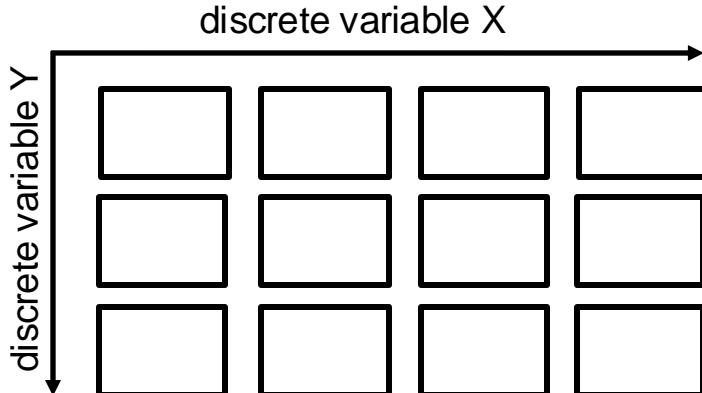
# Use small multiples to separate large data

## The benefit of small multiples

- Sometimes, large datasets overcrowd a single figure panel
- Displaying subsets in *small multiples* is a very common workaround

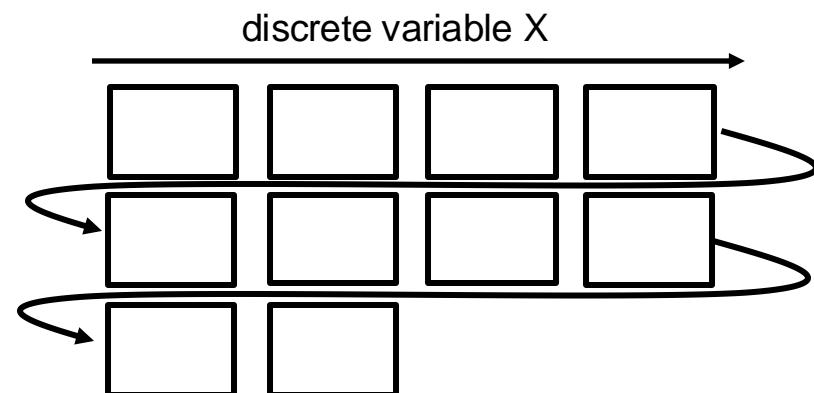
### Multiple grouping variables

- If you have more than one grouping variable, you map the grouping to the axes (rows and columns)



### One large grouping variable

- If you have one grouping variable with many levels, consider wrapping after a fixed number of columns

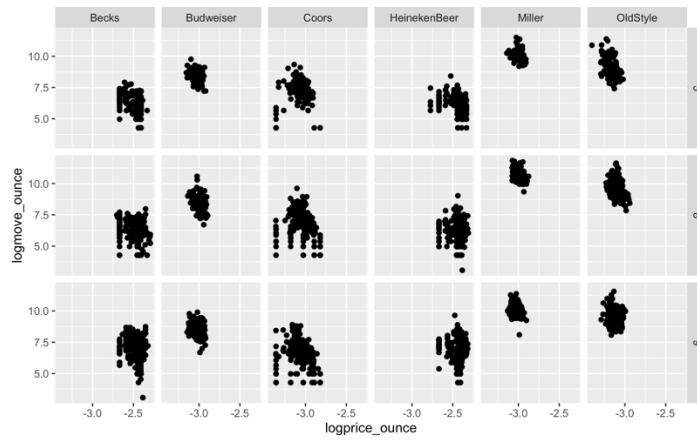


# Small multiples with multiple grouping variables

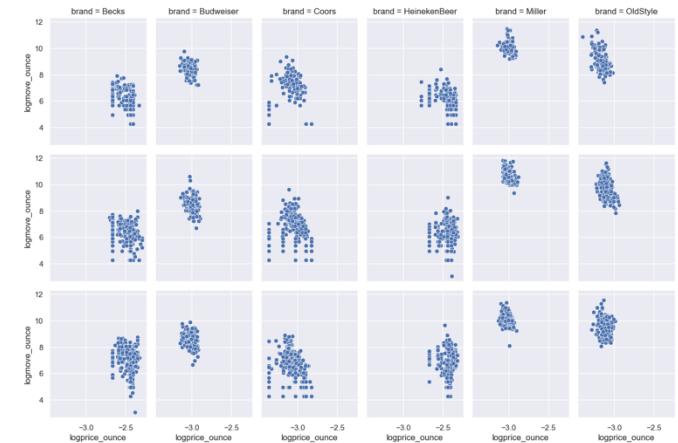
- Instead of displaying all data in one plot, we now draw one plot for each brand and store separately, using a larger subset of the beer sales data containing six brands:



```
ggplot(sixBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce)) +
 geom_point() +
 facet_grid(rows = vars(store),
 cols = vars(brand))
```



```
g = sns.FacetGrid(sixBrands,
 col = "brand", row = "store",
 margin_titles = True)
g.map(sns.scatterplot,
 "logprice_ounce",
 "logmove_ounce")
```

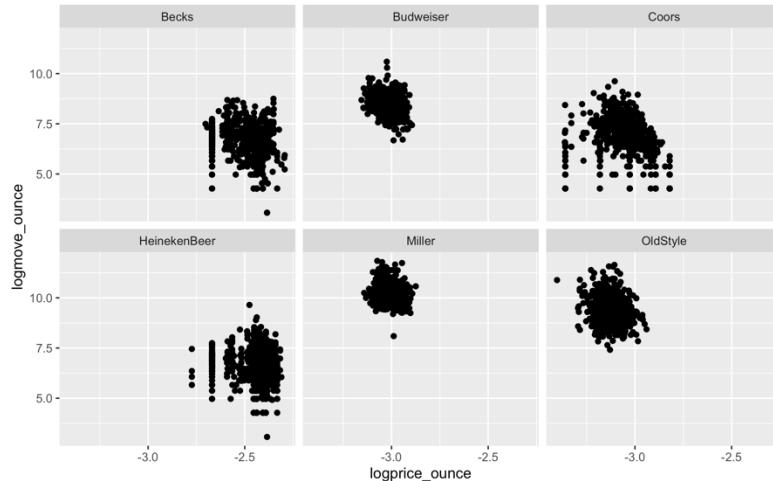


# Small multiples with one large grouping variable

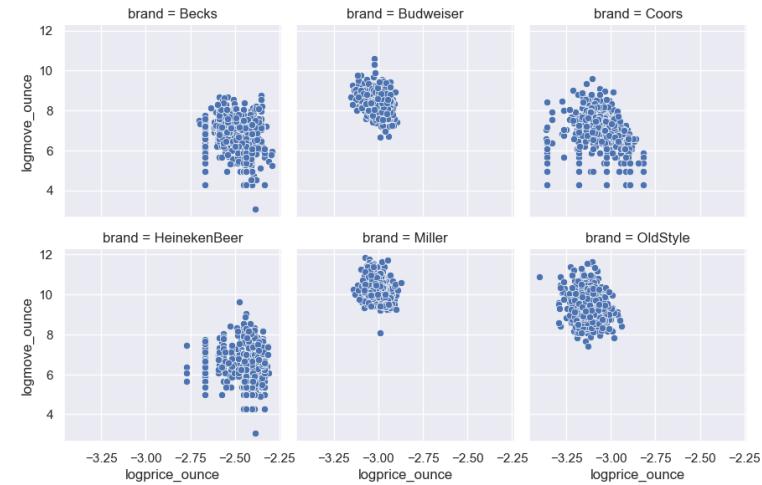
- Now we focus on separating the brands but wrap after 3 plots in one row:



```
ggplot(sixBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce)) +
 geom_point() +
 facet_wrap(vars(brand),
 ncol = 3)
```



```
g = sns.FacetGrid(sixBrands,
 col = "brand", col_wrap = 3,
 margin_titles = True)
g.map(sns.scatterplot,
 "logprice_ounce",
 "logmove_ounce")
```



# Compound figures

## Tools to arrange multiple plots together

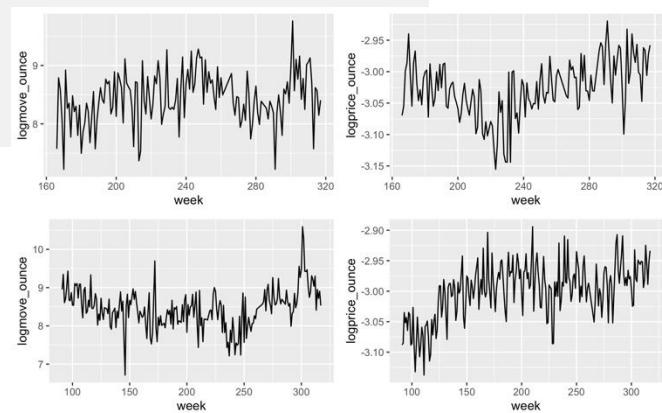
- Often you want to combine several plot in a grid of subplots
- In both R and Python, there are solutions available for that

```
library(ggpubr)

subset data
bud_5 <- brandlevel %>% filter(brand == "Budweiser", store == 5)
bud_8 <- brandlevel %>% filter(brand == "Budweiser", store == 8)

create and save the subplots
plot_bud_5_sales <- ggplot(bud_5, aes(x = week, y = logmove_ounce)) + geom_line()
plot_bud_5_price <- ggplot(bud_5, aes(x = week, y = logprice_ounce)) + geom_line()
plot_bud_8_sales <- ggplot(bud_8, aes(x = week, y = logmove_ounce)) + geom_line()
plot_bud_8_price <- ggplot(bud_8, aes(x = week, y = logprice_ounce)) + geom_line()

arrange the subplots
ggarrange(plot_bud_5_sales, plot_bud_5_price,
 plot_bud_8_sales, plot_bud_8_price,
 nrow = 2, ncol = 2)
```



# Compound figures

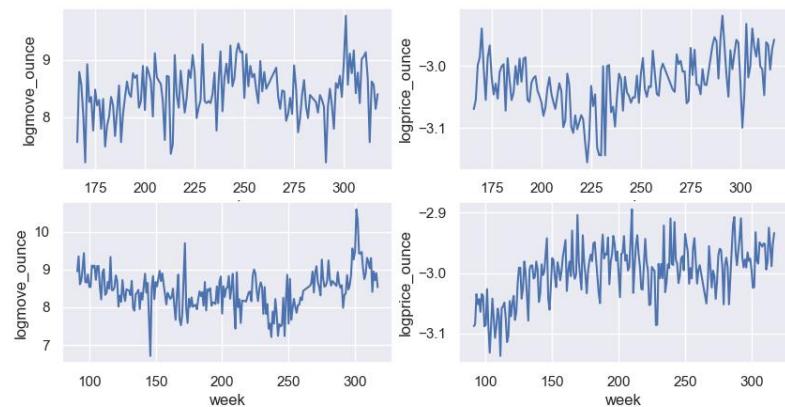
## Tools to arrange multiple plots together

- Often you want to combine several plot in a grid of subplots
- In both R and Python, there are solutions available for that

```
subset data
bud_5 = brandlevel[(brandlevel["brand"] == "Budweiser") & (brandlevel["store"] == 5)]
bud_8 = brandlevel[(brandlevel["brand"] == "Budweiser") & (brandlevel["store"] == 8)]

use matplotlib to create 2x2 canvas
fig, ax = plt.subplots(nrows = 2, ncols = 2)

create and save the subplots on the respective axis
sns.lineplot(x = "week", y = "logmove_ounce",
 data = bud_5, ax = ax[0,0])
sns.lineplot(x = "week", y = "logprice_ounce",
 data = bud_5, ax = ax[0,1])
sns.lineplot(x = "week", y = "logmove_ounce",
 data = bud_8, ax = ax[1,0])
sns.lineplot(x = "week", y = "logprice_ounce",
 data = bud_8, ax = ax[1,1])
```



# Compound figures - Aligning axes

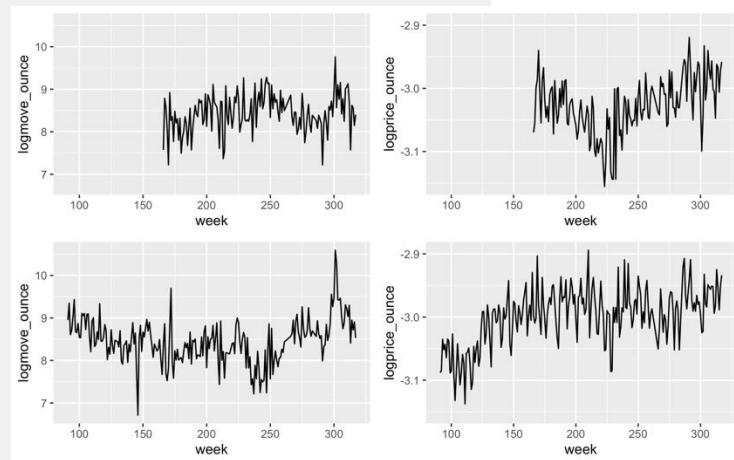
## Axes should always be aligned

- To be able to compare the data, axes should be aligned column- and rowwise
- In {ggplot2}, this has to be done “by hand”

```
calculate axes ranges
lims_x <- c(min(c(bud_5$week, bud_8$week)), max(c(bud_5$week, bud_8$week)))
lims_y1 <- c(min(c(bud_5$logmove_ounce, bud_8$logmove_ounce)),
 max(c(bud_5$logmove_ounce, bud_8$logmove_ounce)))
lims_y2 <- c(min(c(bud_5$logprice_ounce, bud_8$logprice_ounce)),
 max(c(bud_5$logprice_ounce, bud_8$logprice_ounce)))

create and save the subplots
plot_bud_5_sales <-
 ggplot(bud_5, aes(x = week, y = logmove_ounce)) +
 geom_line() + expand_limits(x = lims_x, y = lims_y1)
plot_bud_5_price <-
 ggplot(bud_5, aes(x = week, y = logprice_ounce)) +
 geom_line() + expand_limits(x = lims_x, y = lims_y2)
plot_bud_8_sales <-
 ggplot(bud_8, aes(x = week, y = logmove_ounce)) +
 geom_line() + expand_limits(x = lims_x, y = lims_y1)
plot_bud_8_price <-
 ggplot(bud_8, aes(x = week, y = logprice_ounce)) +
 geom_line() + expand_limits(x = lims_x, y = lims_y2)

arrange the subplots
ggarrange(plot_bud_5_sales, plot_bud_5_price,
 plot_bud_8_sales, plot_bud_8_price,
 nrow = 2, ncol = 2)
```



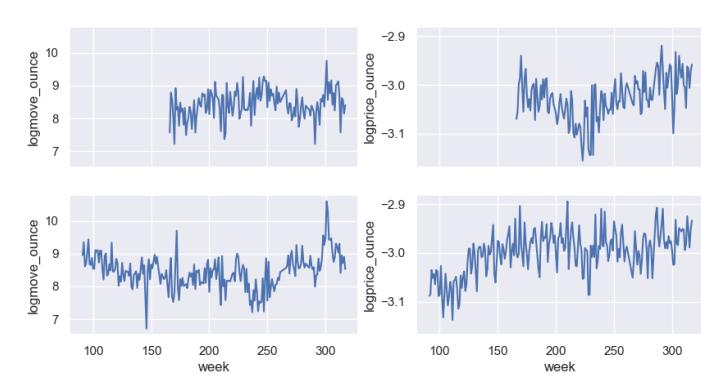
# Compound figures - Aligning axes

## Axes should always be aligned

- To be able to compare the data, axes should be aligned column- and rowwise
- {matplotlib} makes this process very easy

```
use matplotlib to create 2x2 canvas
fig, ax = plt.subplots(nrows = 2, ncols = 2,
 sharex = "all", sharey = "col")

create and save the subplots on the respective axis
sns.lineplot(x = "week", y = "logmove_ounce",
 data = bud_5, ax = ax[0,0])
sns.lineplot(x = "week", y = "logprice_ounce",
 data = bud_5, ax = ax[0,1])
sns.lineplot(x = "week", y = "logmove_ounce",
 data = bud_8, ax = ax[1,0])
sns.lineplot(x = "week", y = "logprice_ounce",
 data = bud_8, ax = ax[1,1])
```



## More flexible illustrations

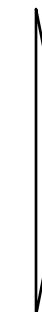
- In R, use the `grid.arrange()` function from the `{gridExtra}` package if you want to deviate from a normal tabular plot layout
- In Python, the `plt.subplot_mosaic()` function has similar capabilities



```
grid.arrange(...,
 layout_matrix = rbind(c(1, 1, 1, 2, 3),
 c(1, 1, 1, 4, 5),
 c(6, 7, 8, 9, 9)))
```



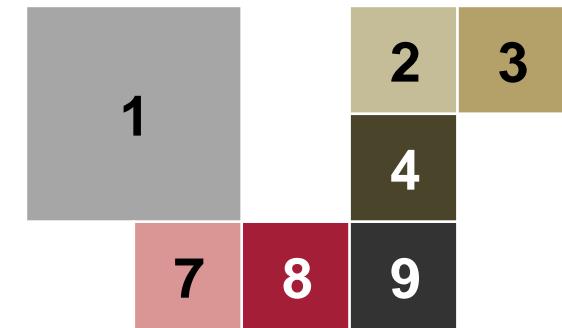
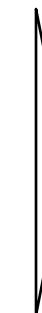
```
fig, ax = plt.subplot_mosaic([[1, 1, 1, 2, 3],
 [1, 1, 1, 4, 5],
 [6, 7, 8, 9, 9]])
```



```
grid.arrange(...,
 layout_matrix = rbind(c(1, 1, NA, 2, 3),
 c(1, 1, NA, 4, NA),
 c(NA, 7, 8, 9, NA)))
```



```
fig, ax = plt.subplot_mosaic([[1, 1, "..", 2, 3],
 [1, 1, "..", 4, "."],
 ["..", 7, 8, 9, "."]])
```



## Exercises

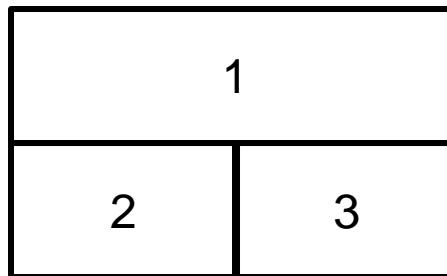


9. Plot the price per ounce over time but facet by stores 8, 9 and 12 in the rows and the three brands in the columns.

## Exercises



10. Combine the graph from exercise 3, one of the graphs from exercise 6 and the continuous plot from exercise 8 using the following layout:



# {ggplot2} doesn't do the trick? Probably, there's an add-on for it!

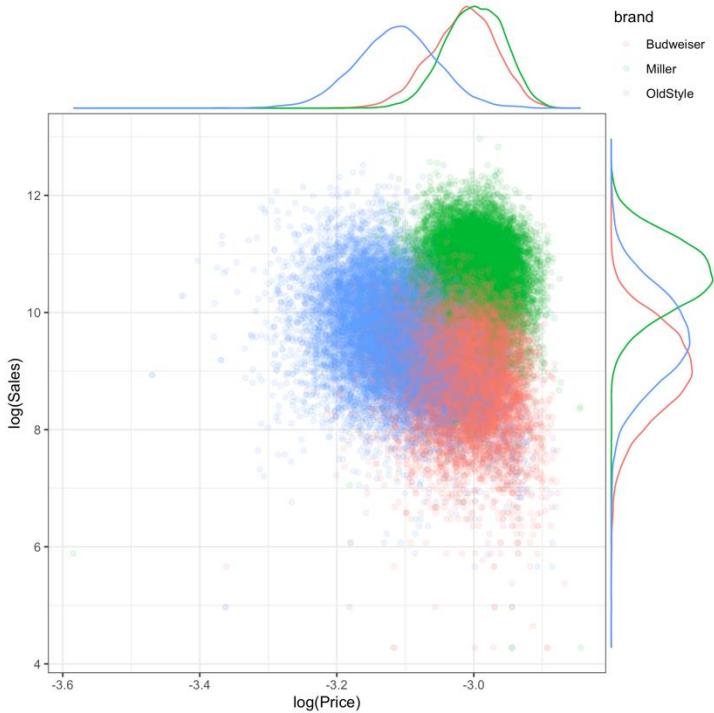
## Marginal densities

- Marginal densities are a useful visualization method for comparison of different groups across different dimensions
- However, the ggplot2 package has no function for that
- {ggExtra} is an extension to {ggplot2}, that displays marginal densities or histograms
- There are other extensions to ggplot2 for many different purposes (e.g., {ggstatsplot})

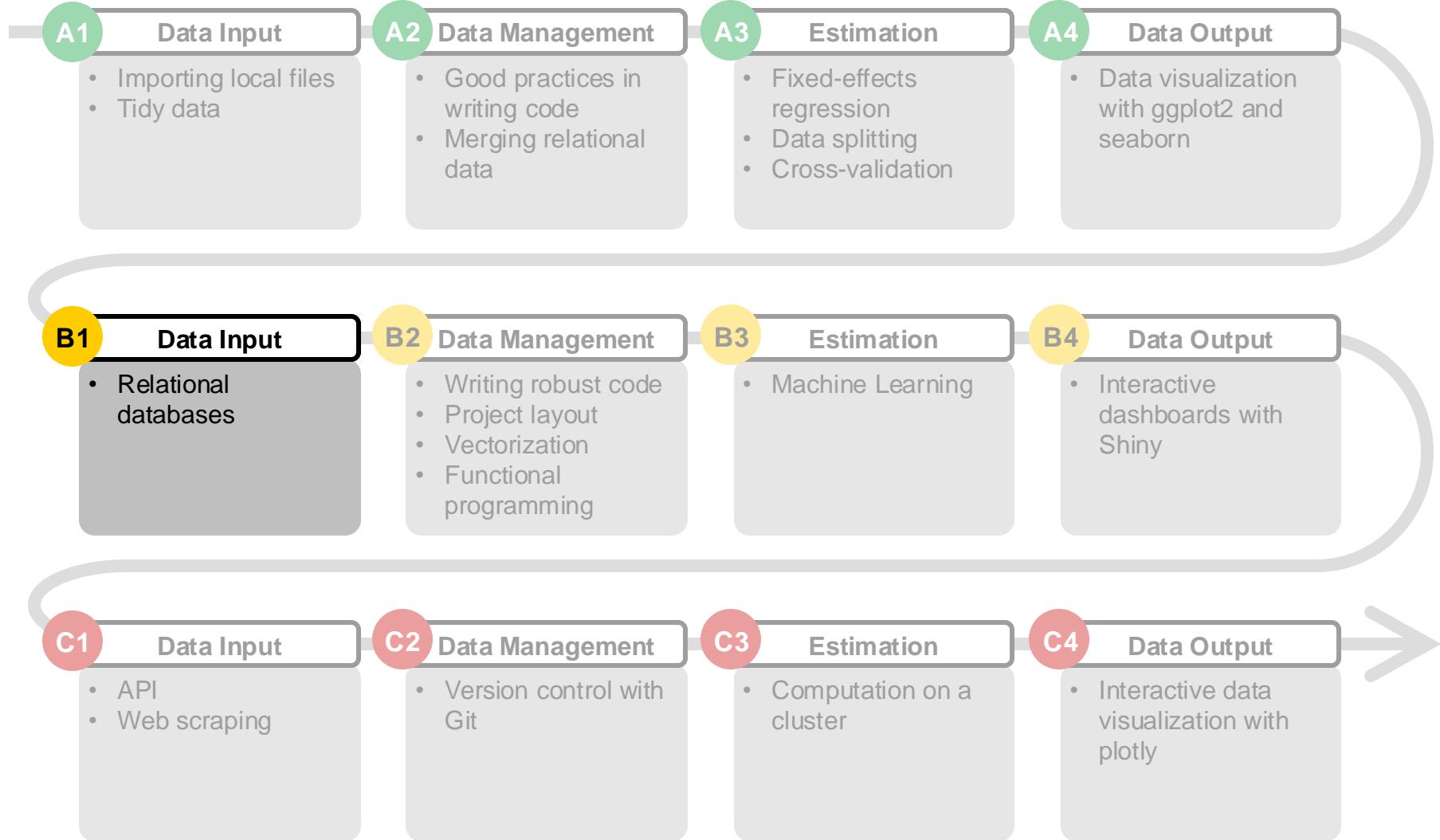
```
library(ggExtra)

Create plot object using ggplot2 syntax
scatterPlot <- ggplot(threeBrands,
 aes(x = logprice_ounce,
 y = logmove_ounce,
 color = brand)) +
 geom_point(alpha = 0.1) +
 theme_bw() +
 theme(legend.justification = c(0, 0),
 legend.position = c(1, 1)) +
 labs(x = "log(Price)", y = "log(Sales)")

Add densities to the plot with the ggMarginal() function
mdPlot <- ggMarginal(scatterPlot,
 type = "density", groupColour = TRUE)
```



# Data Science Project Management: Course Outline



# Large amounts of data are often stored within a database

## What is a relational database?

- A collection of data that consists of different tables that store information in an organized way
- Within a table, rows are called “records” and columns are called “fields”
- Each table contains one or more key columns that connects the different tables with each other
- Is accessed (queried) and maintained using Structured Query Language (SQL)

## Relational Database Management Systems (RDBMS)

- Set of programs and an operating system that create and maintain a database which is used to order and store a multitude of information
- Allows users to store, change and extract data using so called queries
- Open Source: MySQL, PostgreSQL, SQLite, ...
- Commercial: Oracle Database, Microsoft SQL Server, ...

# Using databases makes data analysis faster and more convenient

## — Why use relational databases? —

- Advantage of “relational” aspect: Information only has to be stored once in a single table that is connected to other tables (saves storage space)
- Memory Efficiency:
  - All data that R and Python work with are loaded into memory
  - Some functions even create several copies of the data while being executed
    - Therefore, both are not designed to handle extremely large data sets, might run out of memory for data objects that are larger than a (few) hundred megabytes
- DBMS provide fast access to required parts of large databases
- With DBMS, you can select the specific data you are interested in on the database side, and then only import this data into R or Python
- DBMS allow for joining and calculations on the database side
- DBMS can be accessed by multiple users concurrently
- Relational Database Services often offer safety, security and privacy features (frequent backups, limited access for collaborators, ...)

# Relational Database – Example: Examination Office

| courses |                               |                          |      |         |
|---------|-------------------------------|--------------------------|------|---------|
| course  | full_name                     | lecturer                 | ects | type    |
| B421    | eBusiness                     | Prof. Dr. Dominik Papies | 6    | Lecture |
| S422    | Advanced Microeconometrics    | Prof. Dr. Martin Biewen  | 9    | Lecture |
| B440    | Personnel Economics           | Prof. Dr. Kerstin Pull   | 9    | Lecture |
| B520    | Research Seminar on Marketing | Prof. Dr. Dominik Papies | 9    | Seminar |
| ...     | ...                           | ...                      | ...  | ...     |

| exams      |        |       |
|------------|--------|-------|
| student_id | course | grade |
| 1234567    | B421   | 1.3   |
| 1234567    | S422   | 3.0   |
| 2345678    | B421   | 1.7   |
| 3456789    | B440   | 2.0   |
| 4456987    | B520   | 2.7   |
| ...        | ...    | ...   |

| students   |                |                        |
|------------|----------------|------------------------|
| student_id | name           | program                |
| 1234567    | Max Mustermann | General Management     |
| 2345678    | Erika Eifrig   | General Management     |
| 3456789    | Kathleen Black | European Management    |
| 4456987    | Kong T'ien     | International Business |
| ...        | ...            | ...                    |

# Relational Database – Example: Examination Office

| courses |                               |                          |      |         |
|---------|-------------------------------|--------------------------|------|---------|
| course  | name                          | lecturer                 | ects | type    |
| B421    | eBusiness                     | Johannes Auer, M.Sc.     | 6    | Lecture |
| S422    | Advanced Microeconometrics    | Prof. Dr. Martin Biewen  | 9    | Lecture |
| B440    | Personnel Economics           | Prof. Dr. Kerstin Pull   | 9    | Lecture |
| B520    | Research Seminar on Marketing | Prof. Dr. Dominik Papies | 9    | Seminar |
| ...     | ...                           | ...                      | ...  | ...     |



| exams      |        |       |
|------------|--------|-------|
| student_id | course | grade |
| 1234567    | B421   | 1.3   |
| 1234567    | S422   | 3.0   |
| 2345678    | B421   | 1.7   |
| 3456789    | B440   | 2.0   |
| 4456987    | B520   | 2.7   |
| ...        | ...    | ...   |

| students   |                |                        |
|------------|----------------|------------------------|
| student_id | name           | program                |
| 1234567    | Max Mustermann | General Management     |
| 2345678    | Erika Eifrig   | General Management     |
| 3456789    | Kathleen Black | European Management    |
| 4456987    | Kong T'ien     | International Business |
| ...        | ...            | ...                    |



# Databases are accessed through specific packages and conventions

- There are significant differences in how to access different RDBMS
- However, if you want to access (query) data, the syntax is mostly unified
- There are different packages for different types of databases

| RDBMS      | R package     | Python package                          |
|------------|---------------|-----------------------------------------|
| MySQL      | {RMySQL}      | {mysql-connector-python}<br>+ {pymysql} |
| SQLite     | {RSQLite}     | {sqlite3}                               |
| PostgreSQL | {RPostgreSQL} | {psycopg2}                              |
| OracleDB   | {Roracle}     | {cx_Oracle}                             |

## R : The {DBI} package

- Interaction conventions are specified in package {DBI}
- Contains the R functions for accessing and manipulating databases
- Serves as interface for other packages (e.g., {DBI} is called the “interface”, packages like {RMySQL} are called the “implementation”)
- Automatically installed with the packages, but can be loaded separately (`library(DBI)`)

# A connection object is the basis for all database tasks

## 1 | Establish a connection to a database

```
con <- dbConnect(drv, ...)
```

- Specifies how connections are made
- `drv` = driver that is used to connect to database
- Additional arguments: `dbname`, `host`, `port`, `user`, `password`, ...
- `con` is a DBIConnection object which is needed for every function that wants to interact with a database

### Example:

```
library(DBI)

con <- dbConnect(RMySQL::MySQL(), ← SQL driver, alternatively call dbDriver("MySQL")
 dbname = "company",
 host = "courses.csrrinzqubik.us-
 east-1.rds.amazonaws.com",
 port = 3306,
 user = "student",
 password = "datacamp")
```

DBIConnection  
object



# A connection object is the basis for all database tasks

## 1 | Establish a connection to a database

- In Python, we use {sqlalchemy} for SQL connections
- You define parameters of the connection in a single string
- You still need to have installed the respective database's package:
  - SQLite: {sqlite3} (ships with Python), MySQL: {mysql-connector-python}, ...
- `con` is returned as a connection object which is needed for every function that wants to interact with a database

Example:

```
from sqlalchemy import create_engine

use for SQLite
sql_type, path = "sqlite", "local_path/company.db"
db_connection_str = f'{sql_type}:///{path}'

use for MySQL
sql_type, user, password = "mysql+pymysql", "student", "datacamp"
host, database = "courses.csrrinzqubik.us-east-1.rds.amazonaws.com", "company"
db_connection_str = f'{sql_type}://{user}:{password}@{host}/{database}'

use for both
con = create_engine(db_connection_str)
```

# SQL queries return the specific data you are interested in

## 2 | SQL Queries

- In most cases, you will only need a fraction of the data in a huge database
- It is useful to select the data of interest prior to importing it
- Selection process then happens on the database side. You import only those elements that you actually need
- Selective importing is way more efficient for big databases
- The data is retrieved based on specific criteria that are specified within a SQL query

## Examples:

```
query <- "SELECT name FROM employees
WHERE started_at > \"2012-09-01\""
```



```
out <- dbGetQuery(con, query)
```



```
out = pd.read_sql(query, con)
```

# Pulling an entire table from the database

## 2 | SQL Queries

- If you want to import an entire table into memory, {DBI} provides a nice wrapper.  
In Python, you have to query all entries.
- Note that this is not advisable for large tables

### Examples:



```
dbReadTable(con, "employees")
```



```
query = "SELECT * FROM employees"
pd.read_sql(query, con)
```

# Different functions allow you to learn about the content of a database

## 3 | Exploring the contents of a database

- Often you do not know the structure of a database
- In R, the {DBI} provides some handy functions for exploration
- In Python the approach is highly specific to the RDBMS

### Examples: List tables in a database



```
dbListTables(con)
```

```
MySQL: query for tables in Information Schema
databaseName = "company"
query = (f"SELECT TABLE_NAME FROM information_schema.tables " +
 f"WHERE TABLE_SCHEMA = '{databaseName}'")
```



```
SQLite: query for tables in Master table
query = "SELECT name FROM sqlite_master WHERE type='table'"
```

```
execute the query manually and return as array
pd.read_sql(query, con).values
```



# Different functions allow you to learn about the content of a database

## 3 | Exploring the contents of a database

- Often you do not know the structure of a database
- In R, the {DBI} provides some handy functions for exploration
- In Python the approach is highly specific to the RDBMS

### Examples: List fields in a table



```
dbListFields(con, "employees")
```

```
MySQL: query for fields in Information Schema
databaseName = "company"
tableName = "employees"
query = (f"SELECT `COLUMN_NAME` FROM information_schema.columns " +
 f"WHERE `TABLE_SCHEMA` = '{databaseName}' " +
 f"AND `TABLE_NAME` = '{tableName}'")
```



```
SQLite: query for fields using a PRAGMA statement
tableName = "employees"
query = f"PRAGMA table_info('{tableName}')"
```

```
pd.read_sql(query, con).values
```

Disconnect from the database once you're done working with it

#### 4 | Disconnect from database

- Hosting and maintaining a database is costly
- Active connections cost the host money
- It is good practice to disconnect from databases after working with them in order to minimize the number of active connections to the server
- When working with {sqlalchemy} and {pandas} in Python, you do not need to close the connection as this is done automatically

Examples:



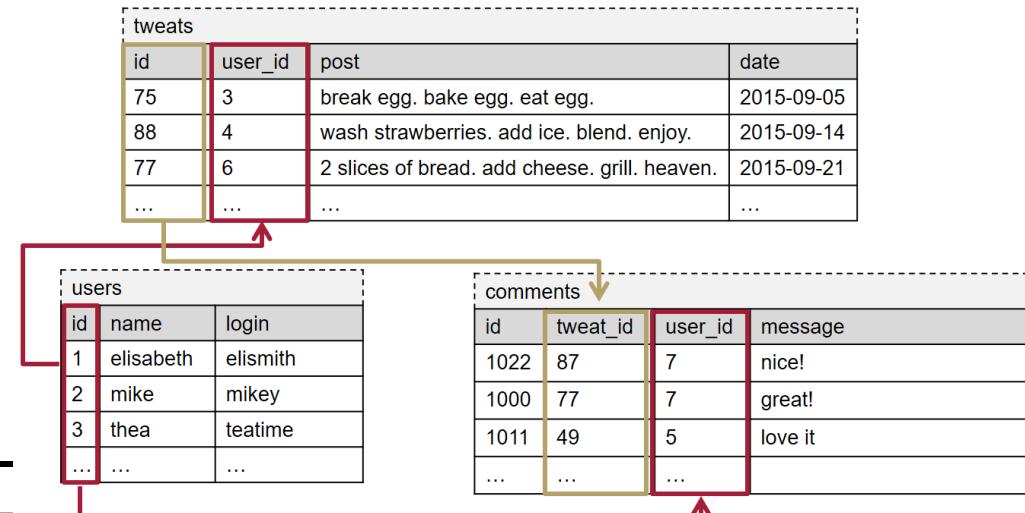
```
dbDisconnect(con)
```

# Exercises



For the first part of this week's exercises, we will work with the database outlined below.

- *comments* are what other users add below a tweet.
- The *id* variable in *tweats* refers to the *tweet*, in *users* it refers to the user, in *comments* it refers to the comment.
- The *user\_id* variable in *tweats* refers to the person who posted the *tweet*, whereas in *comments* the *user\_id* refers to the person who commented on a *tweet*.



## Exercises



1. Establish a connection to the MySQL database named *tweater*.

|          |                                                  |
|----------|--------------------------------------------------|
| host     | courses.csrrinzqubik.us-east-1.rds.amazonaws.com |
| port     | 3306                                             |
| username | student                                          |
| password | datacamp                                         |

2. List all tables that are contained in the database. Further, list the fields contained in each table.

# Introduction to SQL for Data Science – SELECTing Columns

| Statement                            | Explanation                                       | Example                                                |
|--------------------------------------|---------------------------------------------------|--------------------------------------------------------|
| <code>SELECT column</code>           | Selects a column from a table                     |                                                        |
| <code>FROM table</code>              | Specifies the table                               | <code>SELECT name<br/>FROM students</code>             |
| <code>SELECT columnA, columnB</code> | Selects multiple columns                          | <code>SELECT name, program<br/>FROM students</code>    |
| <code>SELECT *</code>                | Selects all columns in table                      | <code>SELECT *<br/>FROM students</code>                |
| <code>SELECT DISTINCT column</code>  | Selects all unique values from column             | <code>SELECT DISTINCT program<br/>FROM students</code> |
| <code>SELECT COUNT(*)</code>         | Returns the number of rows in table               | <code>SELECT COUNT(*)<br/>FROM students</code>         |
| <code>SELECT AVG(column)</code>      | Returns the average value of the specified column | <code>SELECT AVG(grade)<br/>FROM exams</code>          |
| <code>SELECT MAX(column)</code>      | Returns the maximum value of the specified column | <code>SELECT MAX(grade)<br/>FROM exams</code>          |
| <code>SELECT MIN(column)</code>      | Returns the minimum value of the specified column | <code>SELECT MIN(grade)<br/>FROM exams</code>          |
| <code>SELECT SUM(column)</code>      | Returns the sum of the numeric values in a column | <code>SELECT SUM(ects)<br/>FROM courses</code>         |

# Introduction to SQL for Data Science – Filtering Rows

| Statement                                               | Explanation                                                                                                                            | Example                                                                                                                                                                                           |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>SELECT column<br/>FROM table</code>               | Needed <b>before</b> any of the following statements                                                                                   | <code>SELECT student_id<br/>FROM exams</code>                                                                                                                                                     |
| <code>WHERE comparison</code>                           | Filters rows based on text or numerical values in a table                                                                              | <code>WHERE course = 'B421'</code>                                                                                                                                                                |
| <code>WHERE comparison1<br/>AND comparison2</code>      | Multiple conditions, <u>all</u> need to be met                                                                                         | <code>WHERE course = 'B421'<br/>AND grade &lt; 2</code>                                                                                                                                           |
| <code>WHERE comparison1<br/>OR comparison2</code>       | Multiple conditions, <u>just one</u> needs to be met                                                                                   | <code>WHERE course = 'B421'<br/>OR grade &lt; 2</code>                                                                                                                                            |
| <code>WHERE column<br/>BETWEEN value1 AND value2</code> | Filters values within a specified range (inclusive)                                                                                    | <code>WHERE grade<br/>BETWEEN 2 AND 4</code>                                                                                                                                                      |
| <code>WHERE column IN (v1, v2, v3, ...)</code>          | Specify multiple values in WHERE clause                                                                                                | <code>WHERE grade IN (1.0, 1.3, 1.7)</code>                                                                                                                                                       |
| <code>NULL</code>                                       | Missing or unknown value                                                                                                               |                                                                                                                                                                                                   |
| <code>WHERE column IS NULL</code>                       | Selects only rows <u>with</u> missing values in specified column                                                                       | <code>WHERE grade IS NULL</code>                                                                                                                                                                  |
| <code>WHERE column IS NOT NULL</code>                   | ... rows <u>without</u> missing values                                                                                                 | <code>WHERE grade IS NOT NULL</code>                                                                                                                                                              |
| <code>WHERE column LIKE 'pattern'</code>                | Searches for pattern in column<br>- Wildcard 1: % (matches 0, 1, or multiple characters)<br>- Wildcard 2: _ (matches single character) | <code>WHERE course LIKE<br/>                  'B%'</code><br><i>(every course that starts with B)</i><br><code>                  '_4%'</code><br><i>(every course that has 4 in second place)</i> |
| <code>WHERE column NOT LIKE 'pattern'</code>            | Returns all rows that do not contain specified pattern                                                                                 | <code>WHERE course NOT LIKE 'B%'</code><br><i>(every course that starts <u>not</u> with B)</i>                                                                                                    |

# Introduction to SQL for Data Science – Some Other Useful Statements

| Statement                                 | Explanation                                          | Example                                                         |
|-------------------------------------------|------------------------------------------------------|-----------------------------------------------------------------|
| <code>SELECT column<br/>FROM table</code> | Needed <b>before</b> any of the following statements | <code>SELECT *<br/>FROM courses</code>                          |
| <code>LIMIT number</code>                 | Returns only the specified number of rows            | <code>LIMIT 3</code>                                            |
| <code>ORDER BY column</code>              | Sorts in ascending order according to column values  | <code>ORDER BY ects</code>                                      |
| <code>ORDER BY column DESC</code>         | Sorts in descending order according to column values | <code>ORDER BY ects DESC</code>                                 |
| <code>GROUP BY column</code>              | Group a result by one or more columns                | <code>SELECT COUNT(*)<br/>FROM courses<br/>GROUP BY ects</code> |

## Logical Operators in SQL and R/Python

| SQL | Comparison Operator      | R/Python |
|-----|--------------------------|----------|
| =   | Equal                    | ==       |
| <>  | Not equal                | !=       |
| <   | Less than                | <        |
| >   | Greater than             | >        |
| <=  | Less than or equal to    | <=       |
| >=  | Greater than or equal to | >=       |



# SQL JOINS can merge tables before importing the data into R / Python

## SQL JOIN Queries

- Since the data you need is probably stored in different tables, you need to join the tables to get all variables of interest
- You *could* selectively import the relevant data from each table (one data frame for each table) and use the respective R / Python functions to join the data (see A2)
- However, it is way *more efficient* to do the joining on the database side using joins in your SQL queries

### Possible Joins:

|            |                                                         |
|------------|---------------------------------------------------------|
| INNER JOIN | Keeps only the rows for which the key is in both tables |
| LEFT JOIN  | Keeps all rows in left table                            |
| RIGHT JOIN | Keeps all rows in right table                           |
| FULL JOIN* | Combines left join & right join                         |
| CROSS JOIN | Creates all possible combinations of two tables         |

\*FULL JOIN does not work with MySQL databases, but does work in PostgreSQL. However, there is a workaround in MySQL using the UNION keyword to combine a LEFT and a RIGHT JOIN

# SQL JOINS can merge tables before importing the data into R / Python

## General Syntax

```
SELECT columns
FROM left_table
 JOIN right_table
ON left_table.key = right_table.key
```

## R Example

```
dbGetQuery(con,
 "SELECT *
 FROM students
 INNER JOIN exams
 ON students.student_id = exams.student_id")
```

## Tips

- If the key has the same name in both tables, you can use `USING (key)` instead of `ON left_table.key = right_table.key`
- If identical column names are used in different tables for different variables, use `table.name` to identify (e.g. `SELECT students.name, courses.name`)

# Aliasing makes your code easier to read

## Aliasing

- You can rename the variables by “aliasing”, i.e., assigning a temporary name to it.
- Use the AS keyword after the column or table you want to rename. It is also useful for shortening table names. Aliases make the code and the output more readable
- Example:

```
SELECT s.name AS studentName, c.name AS courseName
```

```
FROM students AS s
```

```
INNER JOIN exams AS e
```

```
ON s.student_id = e.student_id
```

```
INNER JOIN courses AS c
```

```
USING (course)
```

| students AS s |                     |                        |
|---------------|---------------------|------------------------|
| student_id    | name AS studentName | program                |
| 1234567       | Max Mustermann      | General Management     |
| 2345678       | Erika Eifrig        | General Management     |
| 3456789       | Kathleen Black      | European Management    |
| 4456987       | Kong T'ien          | International Business |
| ...           | ...                 | ...                    |

| courses AS c |                               |                          |      |         |
|--------------|-------------------------------|--------------------------|------|---------|
| course       | name AS courseName            | lecturer                 | ects | type    |
| B421         | eBusiness                     | Johannes Auer, M.Sc.     | 6    | Lecture |
| S422         | Advanced Microeconomics       | Prof. Dr. Martin Biewen  | 9    | Lecture |
| B440         | Personnel Economics           | Prof. Dr. Kerstin Pull   | 9    | Lecture |
| B520         | Research Seminar on Marketing | Prof. Dr. Dominik Papies | 9    | Seminar |
| ...          | ...                           | ...                      | ...  | ...     |

| exams AS e |        |       |
|------------|--------|-------|
| student_id | course | grade |
| 1234567    | B421   | 1.3   |
| 1234567    | S422   | 3.0   |
| 2345678    | B421   | 1.7   |
| 3456789    | B440   | 2.0   |
| 4456987    | B520   | 2.7   |
| ...        | ...    | ...   |

## Exercises



3. Write and execute SQL queries such that you ...
  - a) Import the *name* column from the *users* table
  - b) Import all columns for all *tweats* that were written after September 21st, 2015
  - c) Import all columns from *comments* for the *message* that starts with the letter "s" and was written by the user with the *user\_id* "1"

*Hint:* In Python, import the function `text()` from `{sqlalchemy}` and use it on your query string to escape the "%" -sign
  - d) Import all columns from *comments* and *users* simultaneously for all users that appear in both tables (Hint: The key in *comments* is *user\_id*, in *users* it is *id*.)

## Exercises



3. Write and execute SQL queries such that you...
  - e) Import the columns *user\_id* and *message* of all *comments* that contain either a question mark or an exclamation mark. In Python, remember to escape the "%"-sign
  - f) Import the columns *name*, *user\_id*, *date* and *post* from *users* and *tweats* simultaneously. Do this even if a value for the key in the left table has no match in the right table.

# Exercises



3. Write and execute SQL queries such that you...
  - g) Go back to the code of the last task and edit it such that rows with missing values for the column *post* are not imported. Order the results by *date*.
  - h) First, left join *comments* and *tweats* on the appropriate key. Then, left join *users* as well, using a key from *tweats*. Import the columns *tweat\_id*, *name*, *user\_id* (from *tweats*), *post*, *message*, *user\_id* (from *comments*) and *message*. Rename the columns in a way that the column name makes intuitive sense (e.g., there should not be two columns named *user\_id*, if they refer to two different things). Lastly, order the results by *tweat\_id*.

## Exercises

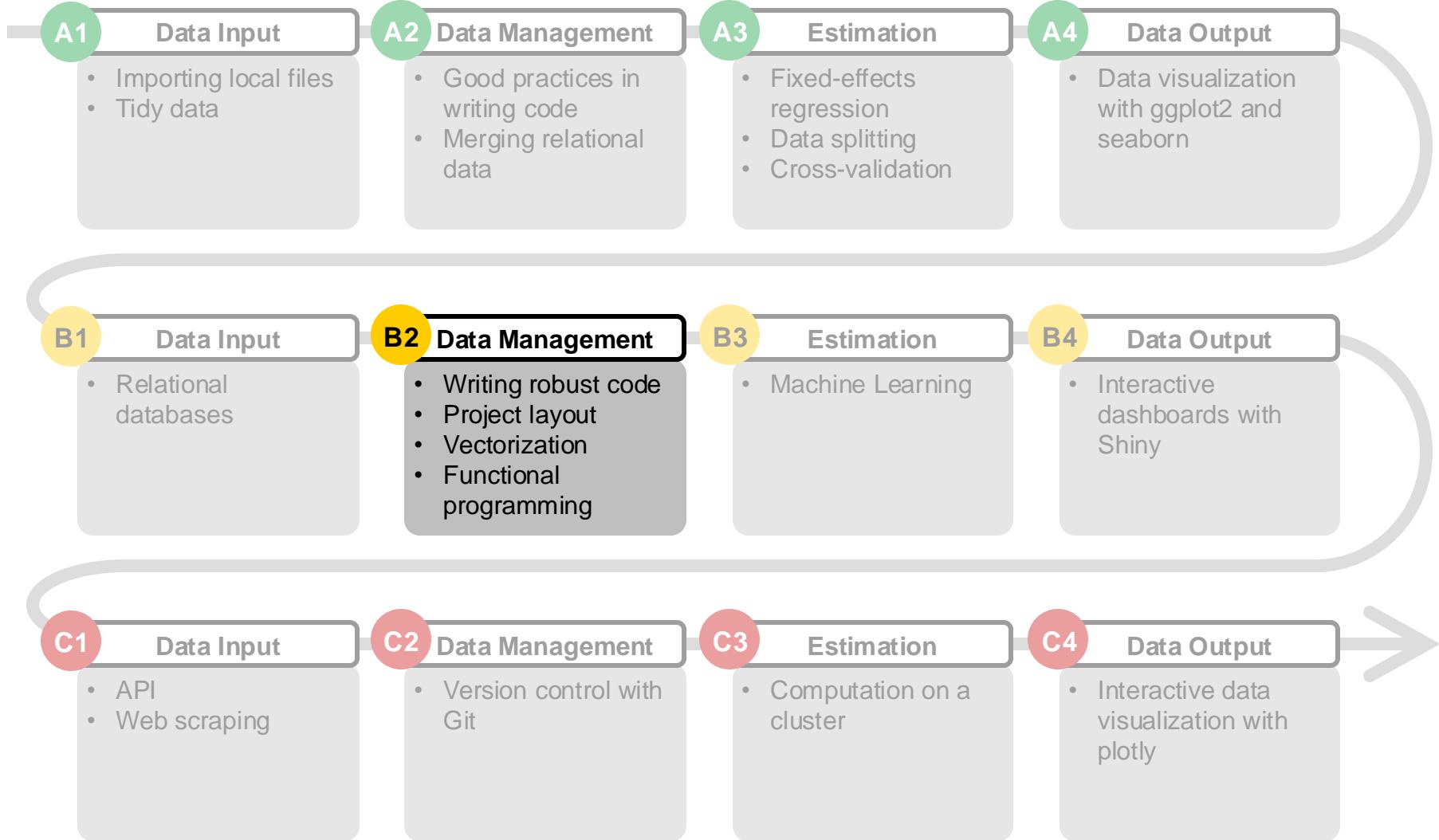


4. Safely disconnect from the database.



- Make sure to also exercise working on SQLite databases with the fifth exercise available online

# Data Science Project Management: Course Outline



# There are good and bad ways to refer to a directory

- **Hardcode file paths each time you refer to a file**

e.g., "C:/Users/Mayer S/Documents/project/data/file.csv"



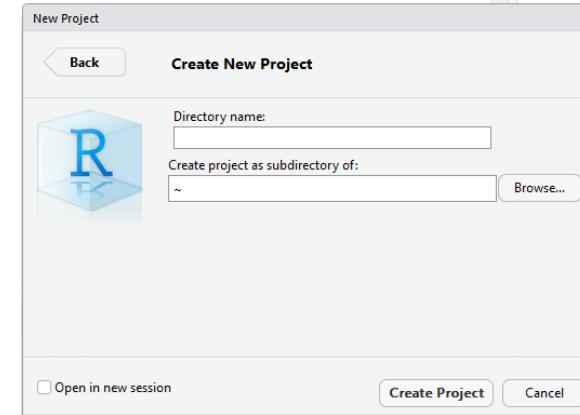
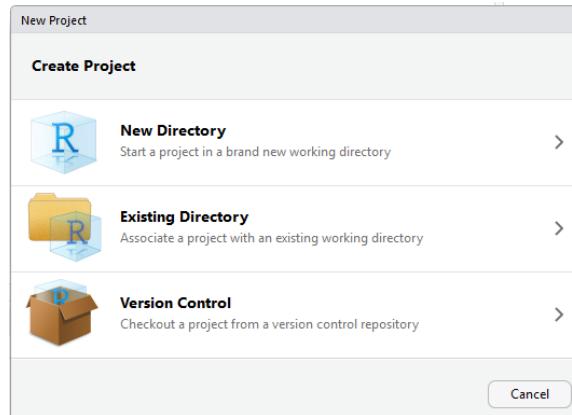
- **Set a working directory at the beginning of your R session**

e.g., "C:/Users/Mayer S/Documents/project"



- **Set up projects in your IDE for each research project**

e.g., File > New Project...





# Bad practice: Hardcode file paths each time you refer to a file

## How to

- You can refer to any path on your PC by entering the full file path each time you load/save an object
- e.g.,

```
do_sth("C:/Users/user/Documents/data/file1.csv")
do_sth("C:/Users/user/Documents/data/file2.csv")
```

## Problems

- The script is specific to the computer of the author and does not run elsewhere
- A script can potentially contain lots of references to files/directories
- Revision of the script may be necessary for a number of reasons:
  - Archiving a project (moving it from a “current projects” directory to a new projects directory)
  - Giving the code to somebody else (colleague, supervisor, ...)
  - Uploading the code with your manuscript submission for review
  - New computer and new directory layout

# Better practice: Define a working directory

## How to

- The working directory is **where R/Python looks for files** that you ask it to load and **where it will put files** that you ask it to save
- The default in Windows (in MacOS) is "C:/Users/<user>/Documents" ("~/Documents")
- Display the current working directory:  
 `getwd()`  
 `os.getcwd()`
- Change the current working directory:  
 `setwd()`  
 `os.chdir()`
- Subfolders within the working directory can then be accessed using relative file paths:  
`do_sth("./data/file1.csv")` or `do_sth("data/file1.csv")`

## Note

In Python, a package has to be loaded:

```
import os
```

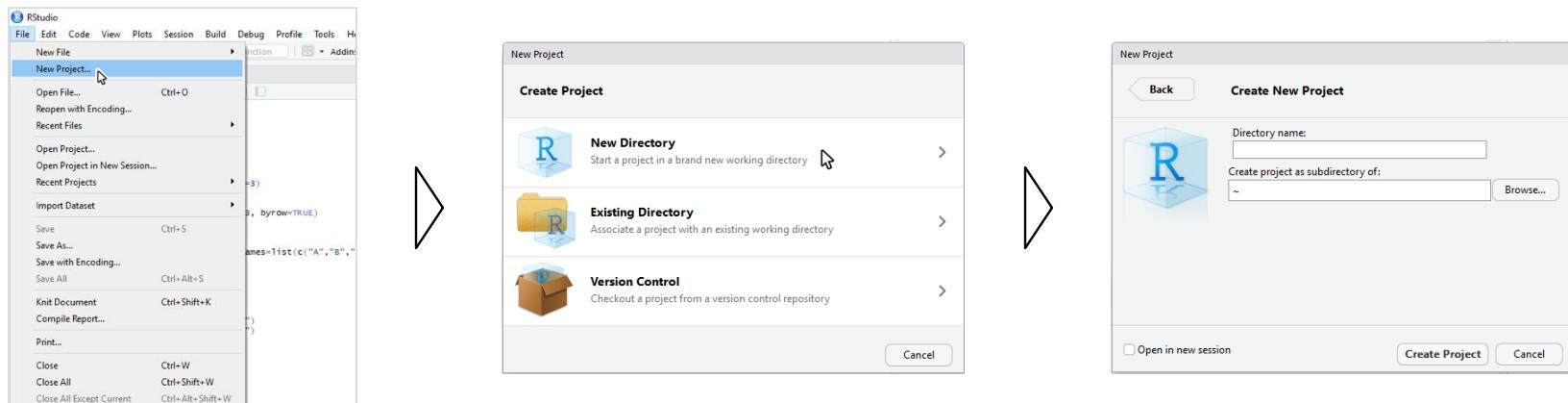
## Problems

- Although the specific part the code is limited to one line of code, the script still runs only on the computer of the author

# Recommended practice when using RStudio: Set up an RStudio project

## How to

- Click on File > New Project... and save the project in a directory of your choice – whenever you refer to a file with a relative path it will look for it here
- You will find an `*.Rproj`-file in the specified directory
- To re-open your project, click on this file and you will get back where you left off
- Everything you need is in one place, and cleanly separated from all the other projects

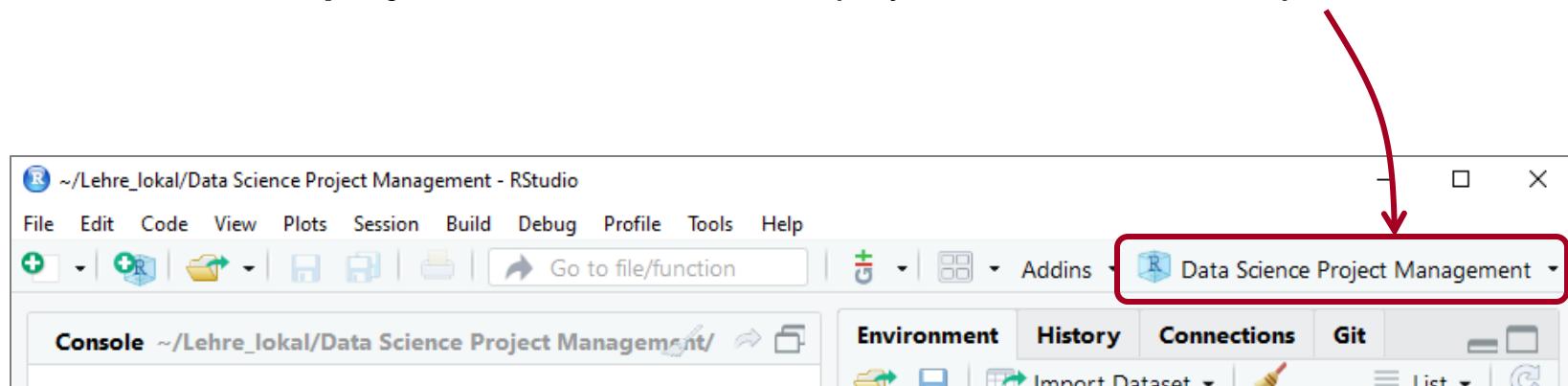


## Problems

- Only works in RStudio, not necessarily for other code editors

# Working with RStudio Projects

- When a new project is created, Rstudio...
  - ... creates a **project file** (with an *.Rproj* extension) within the project directory. This file contains various project options and can also be used as a shortcut for opening the project directly from the filesystem.
  - ... creates a **hidden directory** (named *.Rproj.user*) where project-specific temporary files are stored.
  - ... **loads the project** into RStudio and displays its name in the Projects toolbar



<https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects> (last retrieved on September 16, 2018)



# Working with RStudio Projects

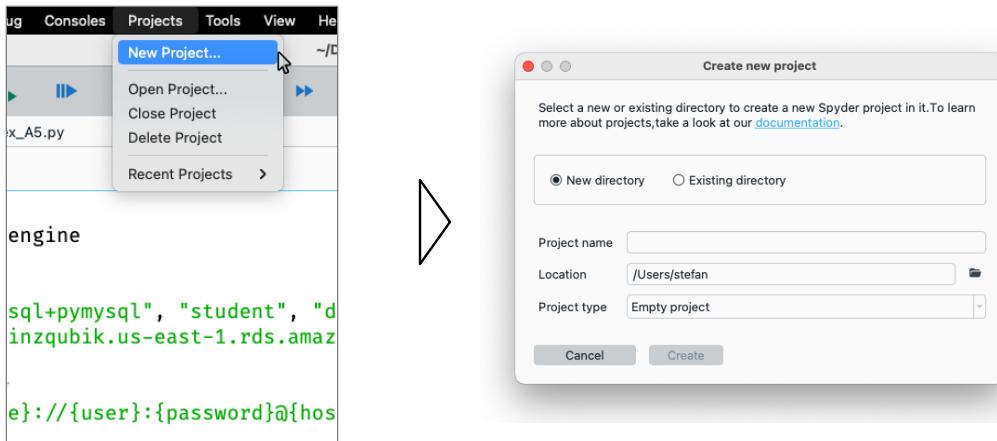
- When a project is opened within RStudio the following actions are taken:
  - A new R session (process) is started
  - The *.Rprofile* file in the project's main directory (if any) is sourced by R
  - The *.RData* file in the project's main directory is loaded (if project options indicate that it should be loaded).
  - The *.Rhistory* file in the project's main directory is loaded into the RStudio History pane (and used for Console Up/Down arrow command history).
  - The current working directory is set to the project directory.
  - Previously edited source documents are restored into editor tabs
  - Other RStudio settings (e.g. active tabs, splitter positions, etc.) are restored to where they were the last time the project was closed.



# Recommended practice when using Spyder: Set up an Spyder project

## How to

- Click on Projects > New Project... and save the project in a directory of your choice – whenever you refer to a file with a relative path it will look for it here
- Everything you need is in one place, and cleanly separated from all the other projects
- To learn more about projects in Spyder, have a look at the [documentation](#)
- Note: The project name is automatically also the folder name and vice versa.



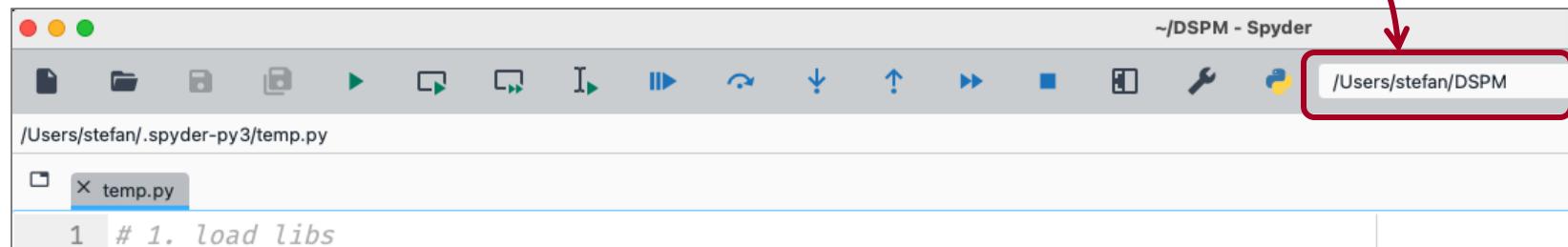
## Problems

- Only works in Spyder, not necessarily for other code editors



# Working with Spyder Projects

- When a new project is created, Spyder ...
  - ... creates a **hidden directory** (named `.spyproject`) where project-specific temporary files and configurations are stored.
  - ... **sets the working directory** to the project directory



<https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects> (last retrieved on September 16, 2018)

## Document what you have done – reproducible research

- Data analyses need to be reproducible...

### **... for us:**

People are forgetful. After we get back to work after a break, we need to be able to understand our data and our methods.

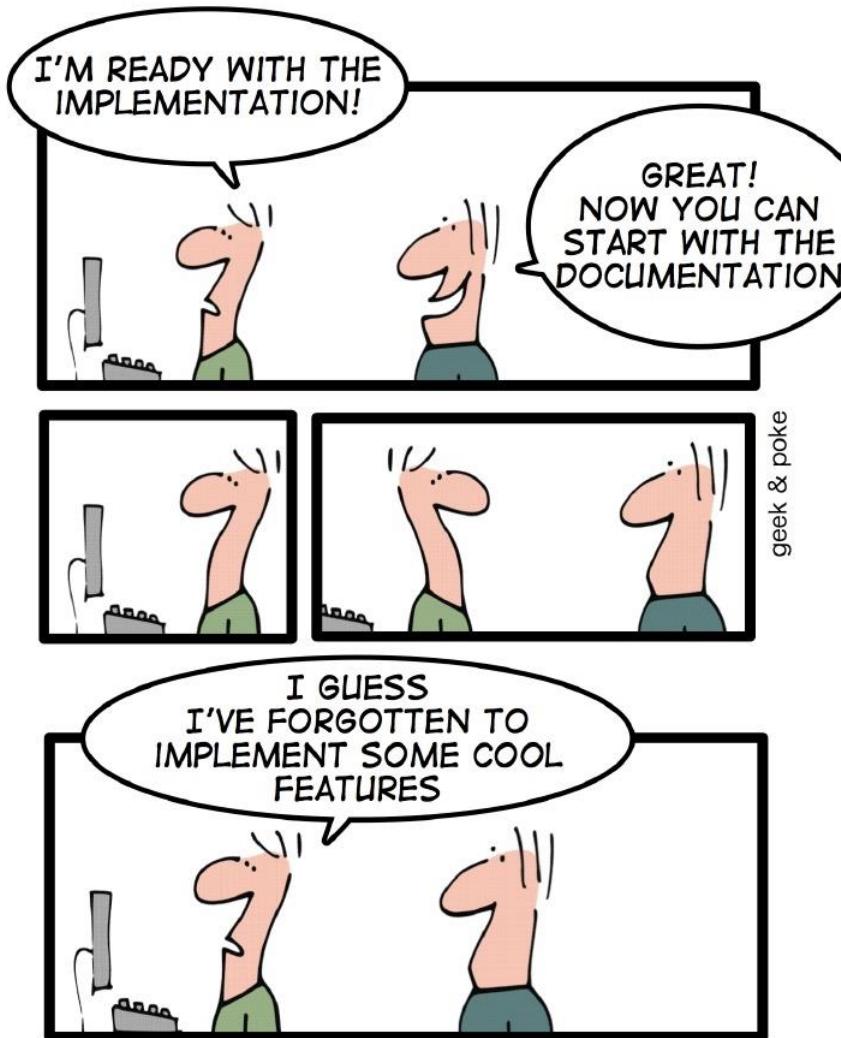
### **... for our friends:**

Our colleagues and co-authors need to understand what we are doing. Only then they can give helpful feedback.

### **... for our enemies:**

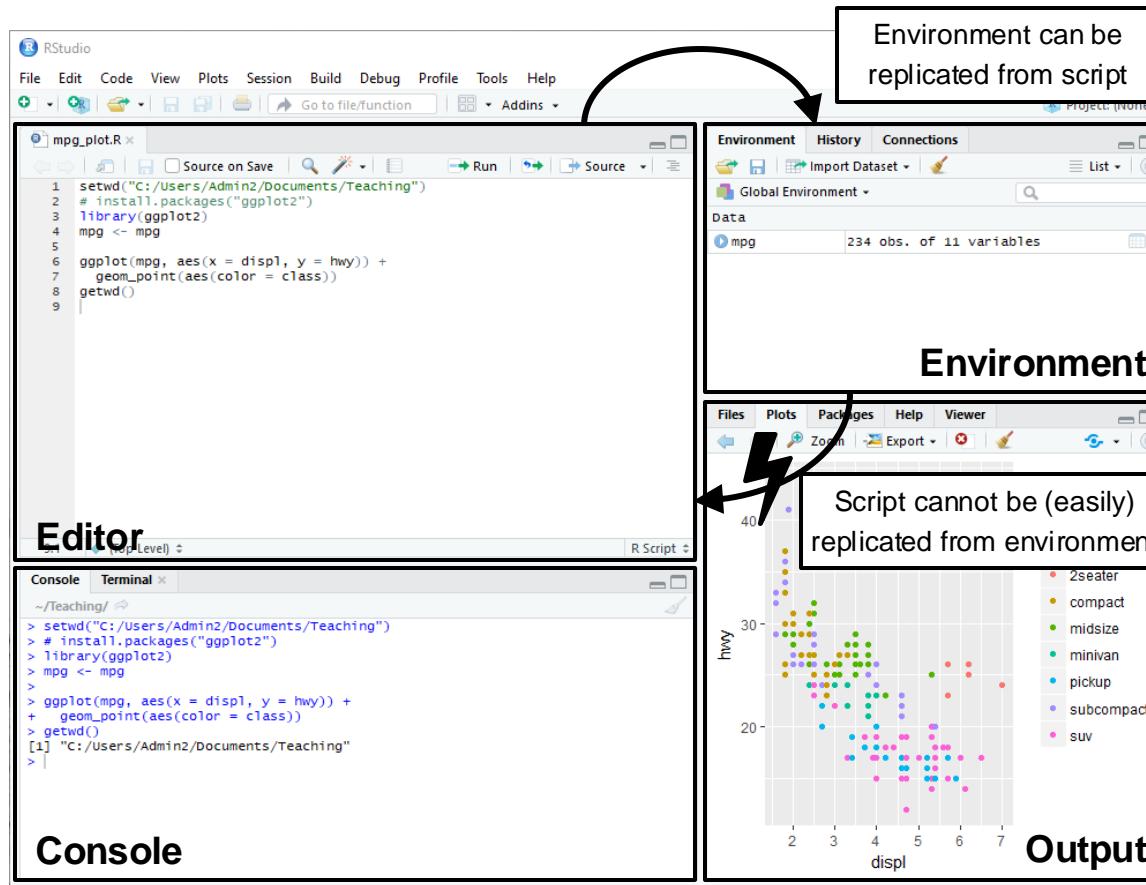
Critical reviewers or the evil supervisor of our master's thesis might be skeptical about our analyses. Hence, we should always (even years after) be able to prove our results exactly.

# Programmers love to comment. But not yet.



[https://www.startitup.sk/learn-to-code/geek\\_and\\_poke\\_coders-love-jpeg/](https://www.startitup.sk/learn-to-code/geek_and_poke_coders-love-jpeg/) (last retrieved on September 16, 2018)

# Scripts are the only lasting record of what happened



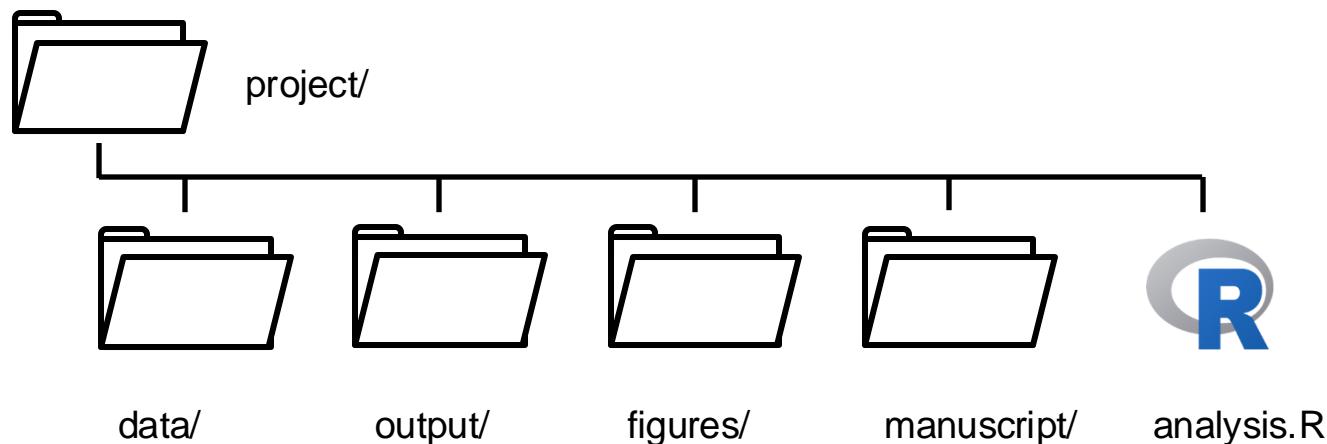
With your scripts (and your data files), you can recreate the environment. It's much harder to recreate your scripts from your environment.

# A good project layout can make life easier

An adequate project structure can help to...

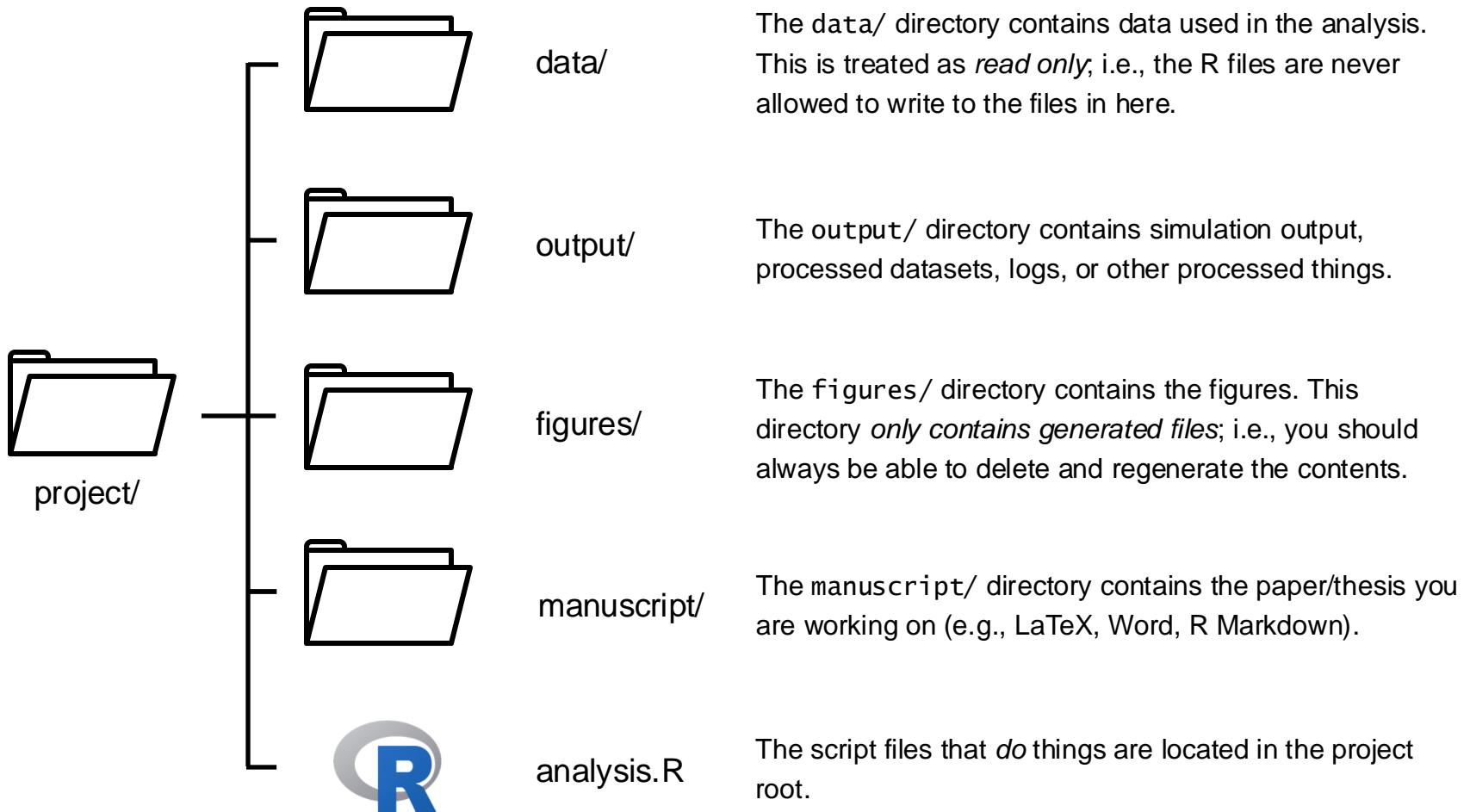
- ... ensure the integrity of data
- ... guarantee portability of the project
- ... facilitate collaboration with colleagues
- ... pick up the project easily after a break

A basic project structure might look like this:



<https://nicercode.github.io/blog/2013-04-05-projects/> (last retrieved on May 15, 2018)

# A good project layout can make life easier



# Make sure your code complies with two important rules

## **Treat data as read only**

- Within your scripts you might generate derived data sets either temporarily (in a session only) or semi-permanently (as a file in output/), but the original data is always left in an untouched state

## **Treat generated output as disposable**

- Files in directories figures/ and output/ are all generated by the scripts
- Before submitting your work, delete all the generated files and rerun the analysis to make sure that you can create all the analyses and figures from the data

# It can be useful to create different scripts for different analysis steps

- We use scripts instead of the “interactive mode” (i.e., programming in the console)
- Often, it is useful to create different scripts for different analysis steps

e.g.:



data\_analysis.R



data\_preparation.R



figures.R

## Problem

- Replication can be difficult as the order of execution may be unclear

- Automation of execution is needed:
  - (A) Automate everything that can be automated
  - (B) Write a single script that executes all code from beginning to end.

A single script should execute all code from beginning to end



data\_analysis.R



data\_preparation.R



figures.R

Order of execution:

2<sup>nd</sup>

1<sup>st</sup>

3<sup>rd</sup>

## Possible solution

- Create a new script that “sources” and executes the scripts file by file
- This “master file” executes all other scripts in the desired order

R

```
setwd("C:/User/project")

Execute scripts in intended
order of execution:
source("data_preparation.R")
source("data_analysis.R")
source("figures.R")
```

Python

```
os.chdir("C:/User/project")

`source()` does not exist in python
We make our own:
def source(file):
 with open(file) as file_stream:
 exec(file_stream.read())

Execute scripts in intended
order of execution:
source('data_preparation.py')
source('data_analysis.py')
source('figures.py')
```

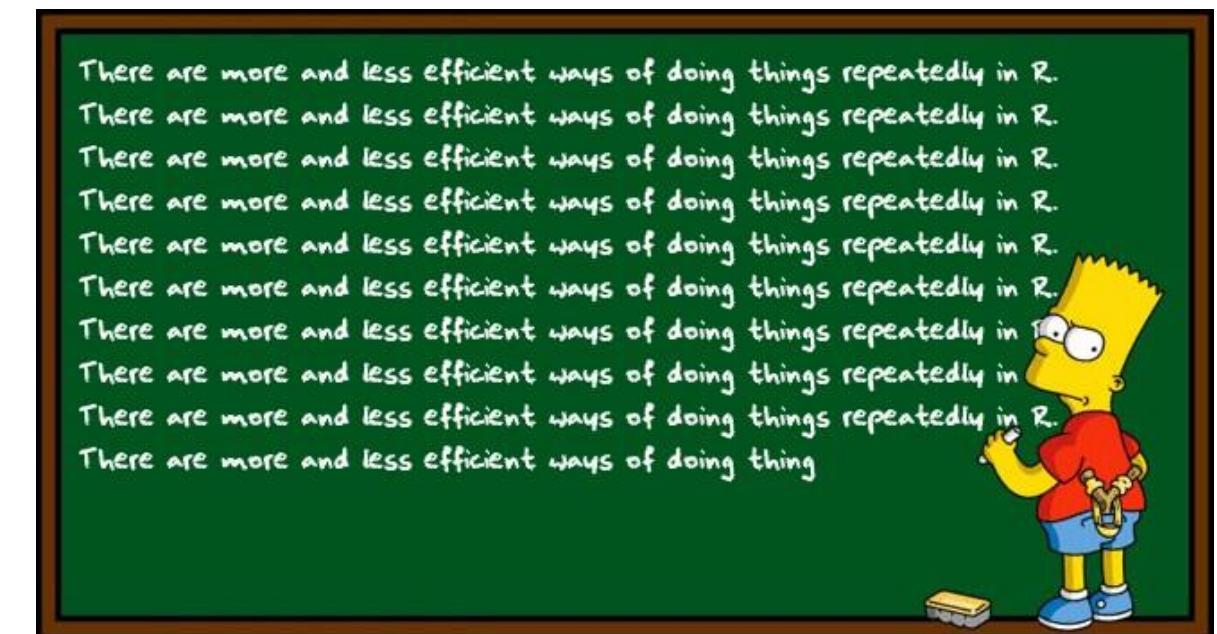
## Exercises



1. Create a folder structure as presented earlier in this chapter.
2. Store the data file *txhousing.csv* and the script *txhousing.R* (both contained in *data.zip*) in the respective locations in the folder structure.
3. The script, as it is, will not run on your computer. Make the adjustments needed to get it running on your computer by:
  - a) Creating a new RStudio project.
  - b) Adjusting the file paths in the script.
4. Split the script into reasonable entities (e.g., data preparation, data analysis, and visualization).
5. Create an additional R script that executes all the other scripts from beginning to end.

# Oftentimes, the same operation is made repeatedly

- There are many occasions where you need to do the same actions multiple times:
  - Reading multiple files from your working directory
  - Creating a plot for different subsets of a data frame (i.e., plot sales against price separately for different brands)
  - Calculating summary statistics for different columns in a data frame
  - ...



<https://www.pianojoe.de/code/simpsongenerator/index.php> (last retrieved on October 31, 2018)

## There are different ways of repeating an action

- Let's assume we want to perform the simple task of calculating the means for each row of a 10,000 x 10 matrix:

```
mat
> [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 18.790 2.0795 514.45 39.270 1109.5 107.860 38.056 1209.465 208.61 84.388
[2,] 25.396 2.5492 459.09 47.113 1025.4 54.371 -16.964 689.922 268.46 101.180
[3,] 61.174 1.9822 423.84 40.826 1020.3 107.099 32.646 47.983 185.84 112.354
[4,] 31.410 1.9840 478.56 28.818 889.8 119.809 49.898 1148.600 235.78 73.313
[5,] 32.586 2.1314 455.63 26.838 1039.7 93.580 23.675 -55.552 220.49 233.860
[6,] 64.301 2.4012 490.95 47.625 1064.4 105.785 82.610 468.457 192.19 75.351
[...]
[9999,] 35.626 1.8217 605.75 29.117 1106.73 127.774 6.7623 1399.84 178.02 70.866
[10000,] 53.635 2.2315 548.92 38.238 939.59 46.457 29.2109 495.09 202.38 264.278
```

Using loops is not very fast, but sometimes the only option we have

### Option 1: Using a for loop

- Loops are the most flexible of the options
- for loops are very common in other languages
- They can be hard to read
- They tend to be very slow
- All the variables are stored in the global scope



```
means <- rep_len(NA, nrow(mat))

for (i in 1:nrow(mat)){
 means[i] <- mean(mat[i,])
}

> Time difference of 0.03261 secs
```



```
means = [None] * mat.shape[0]

for i in range(mat.shape[0]):
 means[i] = np.mean(mat.iloc[i,:])

> Time difference of 0.51038 secs
```

Using loops is not very fast, but sometimes the only option we have

### Option 2: Using functions from the apply family

- Technically similar to option 1 (calls a loop behind the scenes)
- Hence, comparable to loops in terms of speed
- Often easier to read, as it forces you to write a function rather than separate calls
- Order of iteration is not important



```
means2 <- apply(matrix,
 MARGIN = 1,
 FUN = mean)

> Time difference of 0.02454 secs
```



```
means2 = matrix.apply(np.mean,
 axis = 1)

> Time difference of 0.21273 secs
```

If vectorized functions are available, they are the fastest option

### Option 3: Using vectorized functions

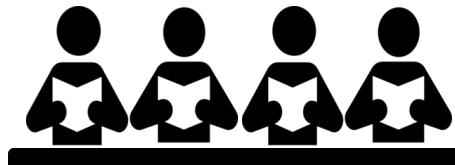
- A “vectorized” function  $f()$  takes a vector  $[x_1, x_2, \dots, x_n]$  as input and returns the vector  $[f(x_1), f(x_2), f(x_3), \dots, f(x_n)]$ , i.e., it works not just on a single value, but on a whole vector of values at the same time
- Many functions in R and Python are written in compiled languages, such as C, C++, or FORTRAN
- Vectorized functions usually involve a behind-the-scenes loop in one of these languages which runs way faster



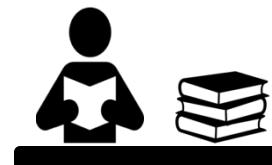
```
means3 <- rowMeans(matrix)
> Time difference of 0.00027 secs
```



```
means3 = matrix.mean(axis = 1)
> Time difference of 0.00040 secs
```



vs.



<https://swcarpentry.github.io/r-novice-inflammation/15-supply-loops-in-depth/> (last retrieved on October 31, 2018)  
<https://nicercode.github.io/guides/repeating-things/> (last retrieved on October 31, 2018)

# A lot of work is done for you – at the cost of speed

## R and Python – High-level programming languages

- R and Python are high-level, interpreted languages, that do a lot of work for you
- When typing `x <- 1.0` in R (or `x = 1.0` in Python), you do *not* have to tell your computer:
  - 1) That “1.0” is a floating-point number
  - 2) That “x” should store numeric-type data
  - 3) To find a place in memory for where to put “1”
  - 4) To register “x” as a pointer to that place in memory
  - 5) To convert “`x <- 1.0`” to binary code (this compilation is done when you hit ‘Enter’)

## ...vs. lower-level programming languages

- In a lower-level language such as C, you might write:

```
int i
i = 5
```
- Here, the program doesn’t have to figure out what type of data is represented by `i`, as it is already explicitly stated, and this is part of what makes it faster

## ... but what does this have to do with loops?

- Consider the simple task of adding two vectors a and b in R...

```
a <- c(1:10000000)
b <- c(2:10000001)
```

- a) ...element-wise using a for loop:

```
sum_a <- NULL
for (i in 1:length(a)){
 sum_a[i] <- a[i] + b[i]
}
sum_a
```

- R has to figure out the data type for each single element (i.e., 20 mn times)

- b) ...vector-wise using a vectorized operation:

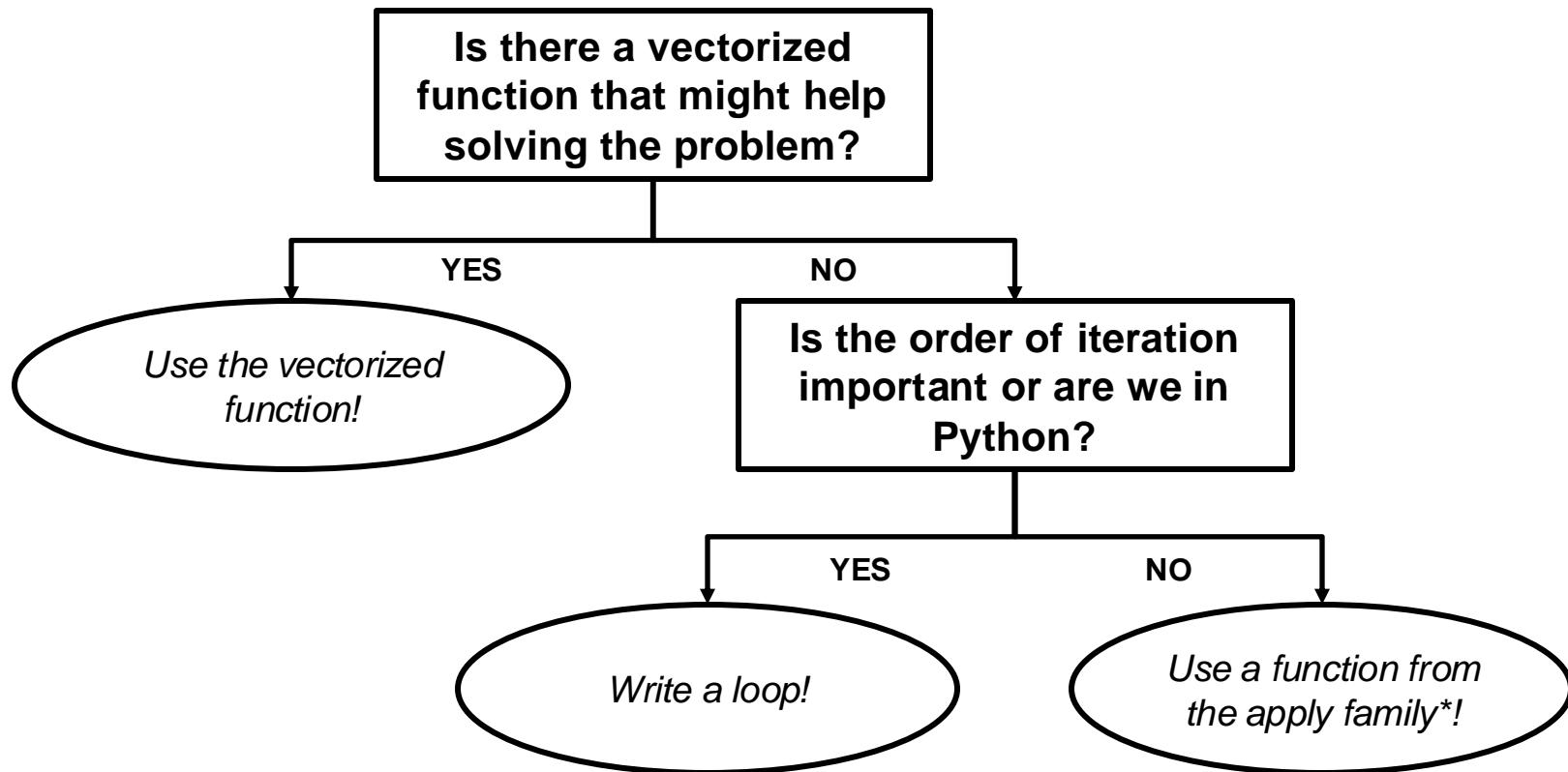
```
sum_b <- a + b
sum_b
```

- As vectors in R can only contain one data type, R only has to identify the type once per vector (i.e., 2 times)
    - Note: The "+" operator is doing the vectorized operation in this case

► In this example, the vectorized solution is more than 100x faster than the for loop!

<http://www.noamross.net/blog/2014/4/16/vectorization-in-r--why.html> (last retrieved on October 31, 2018)

# A decision tree for repeated actions



\*...or, if you are a fan of the tidyverse in R, try out the equivalent map() functions from the {purrr} package!

# We can speed up loops in R at least to some extent

- Back to the loop from earlier:

```
sum_a <- NULL
for (i in 1:length(a)){
 sum_a[i] <- a[i] + b[i]
}
sum_a
```

- The repeated identification of data types is not the only thing that slows down the code
- Here, we let `sum_a` grow with every iteration
- Hence, after each iteration, R needs to re-size the vector and re-allocate memory
- This can be avoided by pre-allocating a vector that fits all the values, e.g., with:  

```
sum_a <- rep(NA, length(a))
```

## Some rules for writing loops

1. Don't use a loop when a vectorized alternative exists
2. Don't grow objects (via `c`, `cbind`, etc.) during the loop - R has to create a new object and copy across the information just to add a new element or row/column
3. Allocate an object to hold the results and fill it in during the loop (something that `apply()` does automatically)
4. Operations that only need to be done once should be placed outside the loop

Once you've copied a piece of code twice, it might be time for writing a function

Hadley Wickham **Hadley Wickham**   
@hadleywickham

Time to write a function

Fluff Society **Fluff Society**  
@FluffSociety

"Ctr+C, Ctr+V, Ctr+V, Ctr+V."



<https://twitter.com/hadleywickham/status/909242896691466240> (last retrieved on October 31, 2018)



# How functions work – a (very) simple example



```
define
function_name <-
 function(ARGUMENT_1, ...) {
 OPERATIONS
 return(VALUE)
 }

call
function_name(ARGUMENT_1)
```



```
define
def function_name(ARGUMENT_1, *args):
 OPERATIONS
 return(VALUE)

call
function_name(ARGUMENT_1)
```

## Examples



```
square <- function(x){
 x_sq <- x^2
 return(x_sq)
}

square(5)
[1] 25
```



```
def square(x):
 x_sq = x**2
 return(x_sq)

square(5)
> 25
```

# A function can return literally anything

```
square_num <- function(x){
 x_sq <- x^2
 return(x_sq)
}
```

```
square_num(5)
[1] 25
```

```
square_list <- function(x){
 x_sq <- x^2
 return(list(value = x, value_squared = x_sq))
}
```

```
square_list(5)
$value
[1] 5

$value_squared
[1] 25
```

```
square_df <- function(x){
 x_sq <- x^2
 return(tibble(value = x,
 value_squared = x_sq))
}
```

```
square_df(5)
A tibble: 1 × 2
 value value_squared
 <dbl> <dbl>
1 5 25
```

# We can specify default values for our functions



```
square_df <- function(x = 1){
 x_sq <- x^2
 return(tibble(value = x,
 value_squared = x_sq))
}

square_df()
A tibble: 1 × 2
 value value_squared
 <dbl> <dbl>
1 1 1
```



```
def square_df(x = 1):
 x_sq = x**2
 return(pd.DataFrame({
 "value": [x],
 "value_squared": x_sq}))

square_df()
> value value_squared
> 0 1 1
```

Is the argument deliberately left blank? Messages are helpful to inform users about what happens:

```
square_df <- function(x = NULL){
 if (is.null(x)) {
 x <- 1 # pre-assign default to 1
 message("No input value provided.
 Using default value of 1.")
 }
 x_sq <- x^2
 return(tibble(value = x,
 value_squared = x_sq))
}

square_df()
No input value provided. Using default value
of 1.
A tibble: 1 × 2
 value value_squared
 <dbl> <dbl>
1 1 1
```

```
def square_df(x = None):
 if x is None:
 x = 1 # pre-assign default to 1
 print("No input value provided. "+
 "Using default value of 1.")
 x_sq = x**2
 return(pd.DataFrame({
 "value": [x],
 "value_squared": x_sq}))

square_df()
No input value provided. Using default value
of 1.
> value value_squared
> 0 1 1
```

# Hadley Wickham on the key idea of functional programming



---

*»R, at its heart, is a functional programming (FP) language. This means that it provides many tools for the creation and manipulation of functions. In particular, R has what's known as first class functions. You can do anything with functions that you can do with vectors: you can assign them to variables, store them in lists, pass them as arguments to other functions, create them inside functions, and even return them as the result of a function.«*

— Hadley Wickham

(Chief Scientist at RStudio & Author of the Tidyverse)

---

<https://twitter.com/hadleywickham> (last retrieved on October 9, 2018)  
<http://adv-r.had.co.nz/Functional-programming.html> (last retrieved on September 08, 2020)

## Passing functions as arguments to other functions? Hey, we've been there!

- Remember that earlier in this chapter, we have applied the same function repeatedly using the `*apply()` functions?
- In these cases we've passed a function (e.g., `mean()`) as argument to other functions (e.g., `apply()`).

### Example

Consider a numeric vector `data` and this function that takes a numeric value as input, computes the squared value, and returns a data frame containing the provided value and the squared value:



```
square_df <- function(x){
 x_sq <- x^2
 return(tibble(value = x,
 value_squared = x_sq))
}

square_df(1)
A tibble: 1 × 2
 value value_squared
 <dbl> <dbl>
1 1 1
```



```
def square_df(x):
 x_sq = x**2
 return(pd.DataFrame({
 "value": [x],
 "value_squared": x_sq}))

square_df(1)
> value value_squared
> 0 1 1
```

Let's assume we want to apply this function to all values in our vector `data` and store the resulting data frames in a list. How can we approach this?



# Passing functions as arguments to other functions? Hey, we've been there!

## Example (contd.)



```
data <- c(1, 2, 3)

square_df <- function(x){
 x_sq <- x^2
 return(tibble(value = x,
 value_squared = x_sq))
}

lapply(data, square_df)

[[1]]
A tibble: 1 × 2
 value value_squared
 <dbl> <dbl>
1 1 1

[[2]]
A tibble: 1 × 2
 value value_squared
 <dbl> <dbl>
1 2 4

[[3]]
A tibble: 1 × 2
 value value_squared
 <dbl> <dbl>
1 3 9
```



```
data = pd.Series([1,2,3])

def square_df(x):
 x_sq = x**2
 return(pd.DataFrame({
 "value": [x], "value_squared": x_sq}))

data.apply(square_df)

> 0 value value_squared
> 0 1 1
> 1 value value_squared
> 0 2 4
> 2 value value_squared
> 0 3 9
> dtype: object
```



## Exercises



6. Consider the following matrix:



```
mat <- matrix(c(1:10, 11:20, 21:30),
 nrow = 10, ncol = 3)
```



```
mat = pd.DataFrame(np.linspace([1, 11, 21],
 [10, 20, 30],
 10))
```

For each row, calculate the standard error of the mean (Hint:

$SEM(x): x \rightarrow \frac{sd(x)}{\sqrt{length(x)}}$  yields the SEM for a vector  $x$ ), using ...

- a) ... a for loop
- b) ... the `apply()` function (Hint: Create a function that computes the SEM and pass it to `apply()`)



## Exercises



7. The following piece of code loops through a vector of length 10 mn and assigns a missing value to all values that are smaller than .05. Moreover, it computes the processing time for the operation:



```
set.seed(123)
x <- runif(10000000)

system.time(
 for (i in 1:length(x)) {
 if (x[i] < 0.05) {
 x[i] <- NA
 }
 }
)
```



```
import package for time keeping
import time

np.random.seed(123)
x = np.random.rand(10000000)

start_time = time.time()

for i in range(x.size):
 if x[i] < 0.05:
 x[i] = np.NaN

round(time.time() - start_time, 5)
```

Can you come up with a vectorized function that does the job more efficiently?

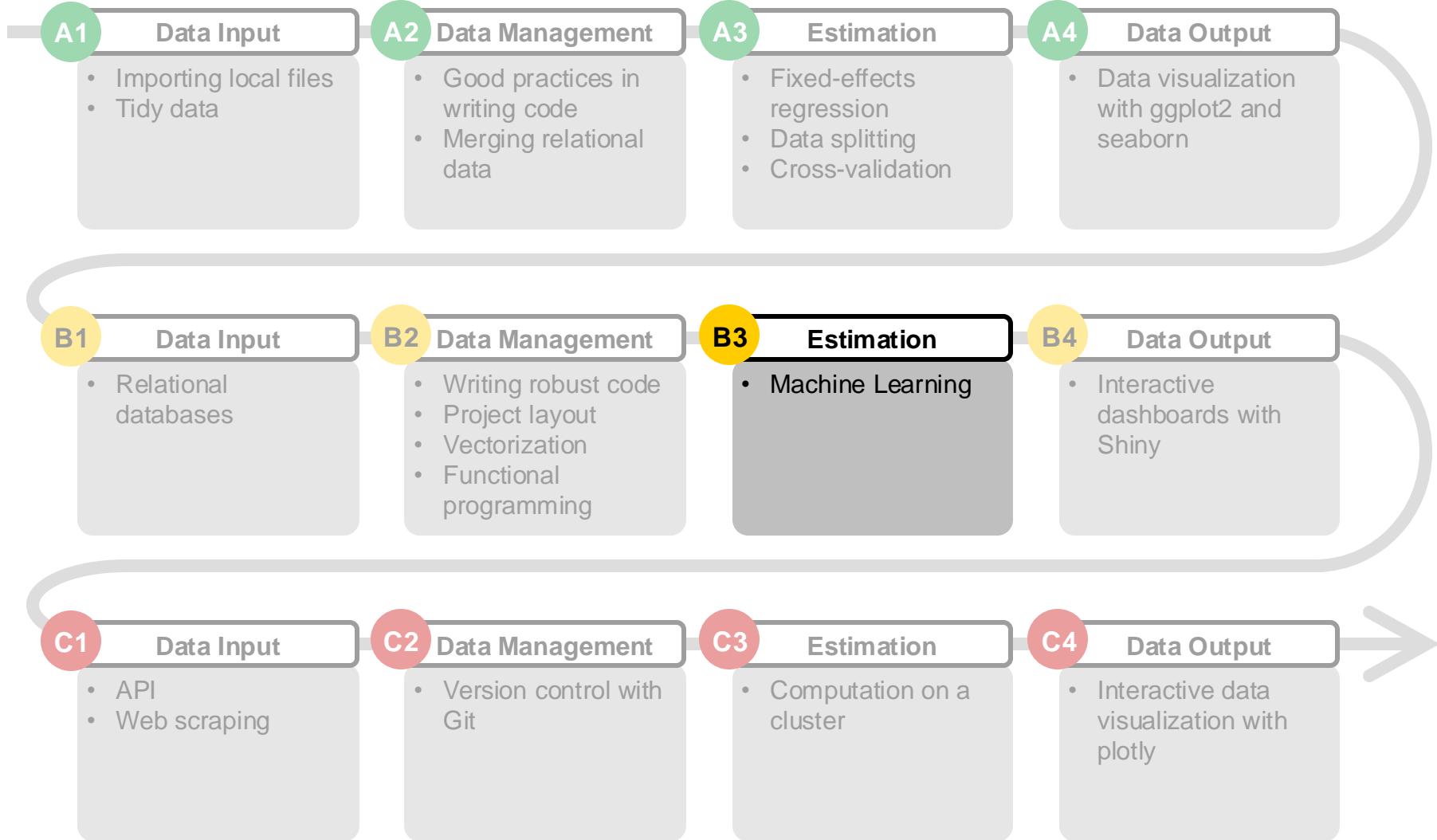


To conclude the chapter, let's again listen to Hadley himself...

The image is a composite of two frames. On the left is a dark slide with the title "Functional programming" in large white serif font, and a subtitle "Or, why for loops are 'bad'" in smaller white sans-serif font. On the right is a video frame showing a man with a beard, wearing a striped shirt, standing at a desk and speaking. He is holding a small white rectangular object, possibly a remote or a small screen. The background is a red curtain.

[https://www.youtube.com/watch?time\\_continue=372&v=GyNqIOjhPCQ&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=372&v=GyNqIOjhPCQ&feature=emb_logo) (last retrieved on September 09, 2020)

# Data Science Project Management: Course Outline



# Machine Learning is a central part of Data Science Projects

## Machine learning

- It is a Data Scientist's job to gain a thorough understanding of data
- Using some modeling assumptions, Machine Learning helps us to learn and predict patterns
- In this chapter, we want to take a quick look at some of the most frequently used models and their application in both R and Python
- Caveat: term “machine learning” is sometimes thrown around as if it is some kind of magic pill: apply machine learning to your data, and all your problems will be solved!
- **Without domain knowledge, you will not be able to correctly apply machine learning!**

# The sound of machine learning that is just logistic regression



<https://twitter.com/MaartenvSmeden/status/1083832145779535872> (last retrieved on December 5, 2022)

# Some central concepts

| Term                                      | Explanation                                                           |
|-------------------------------------------|-----------------------------------------------------------------------|
| outcome, response, target variable, label | variable that should be predicted by ML algorithm                     |
| labels, ground truth                      | values the target variable takes on                                   |
| features, attributes, predictors          | variables used for prediction of a target variable                    |
| training                                  | the process of fitting a particular model to a dataset                |
| training set                              | data set that is used to estimate a model                             |
| test set                                  | data set that is used to evaluate the fit of the model on unseen data |

courtesy of Prof. D. Pampigiani (Tübingen)

# Types of learning

## Supervised learning

- **Predictive** model (find best model to predict the outcome on unseen data)
- Input: Labeled data (i.e. with observed outcome)
- Two central subcategories
  - Classification: Predicting categories
  - Regression: Predicting numeric values
- In this lecture: kNN, Random Forests

## Unsupervised learning

- **Descriptive** model (discover patterns or structure in the data)
- Input: Unlabeled data (i.e. no outcome is considered)
- Two central subcategories
  - Clustering: Grouping similar observations into homogenous clusters
  - Dimensionality reduction: Grouping features to reduce high dimensional data
- In this lecture: k-Means Clustering, PCA

# Common Machine Learning Models - Supervised learning

## Classification

- Naive Bayes
- Logistic Regression
- k-Nearest Neighbors
- Decision Trees and Random Forests
- Support Vector Machines

## Regression

- Linear Regression
- Polynomial Regression
- Local Regression

# Common Machine Learning Models - Unsupervised learning

## Clustering

- k-Means
- agglomerative Clustering
- DBSCAN

## Dimensionality Reduction

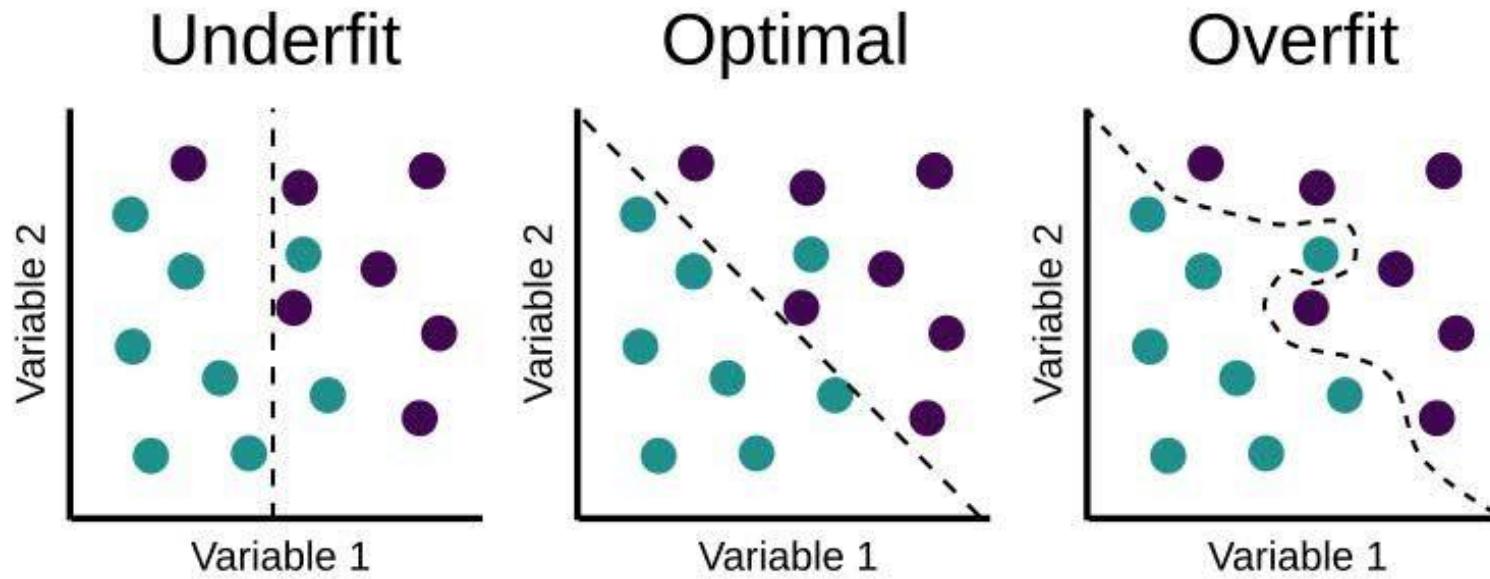
- Principal Component Analysis
- Linear Discriminant Analysis
- t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Uniform Manifold Approximation and Projection (UMAP)

# Modeling considerations

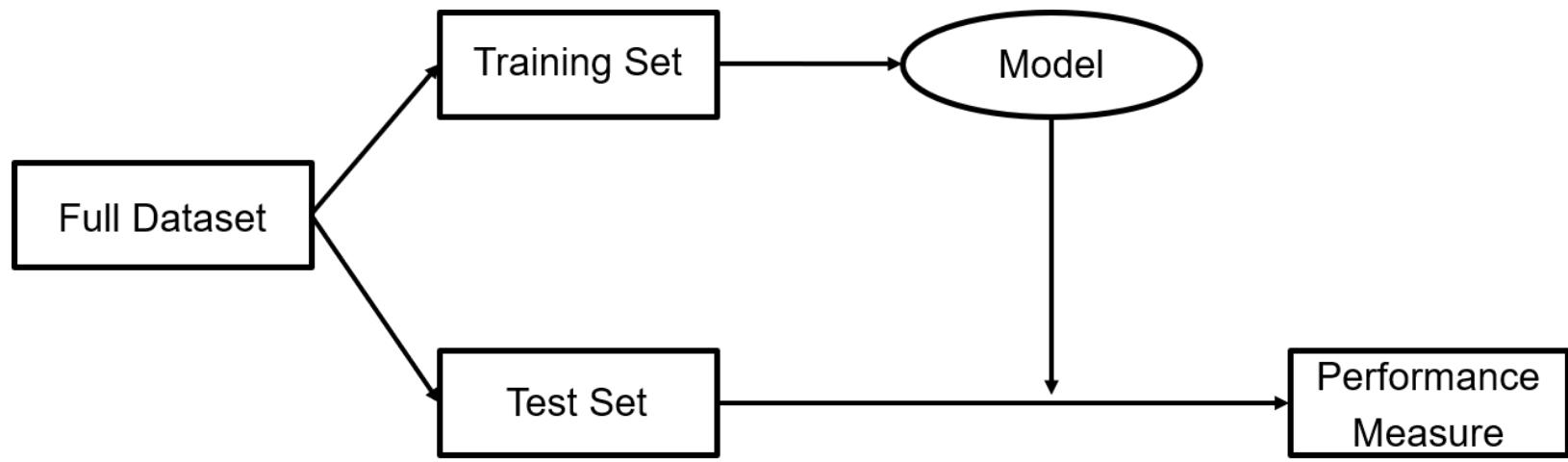
## Under- and Overfitting

- Always keep in mind: We look to achieve good results on unseen data
- Underfitting: missing local differences near the decision boundary (classification) / in the relationship (regression)
- Overfitting: model places too much importance on / is too sensitive to local differences (i.e., noise)
- Balancing overfitting and underfitting data is called the bias-variance tradeoff
- Recall the methods of **sample splitting** and **cross validation** from A3. These come in very handy to ensure predictive power

# Modeling considerations



# Modeling considerations



# Data preprocessing

## Sample splits

- In R, use the workflow we discussed in A3:



```
the sample split is a random operation; set.seed() ensures replicability
set.seed(123)
randomly draw 80% of the observations
estIndex <- sample(nrow(data), size = 0.8 * nrow(data), replace = FALSE)
subset dataset
train_data <- data[estIndex,]
create holdout sample with the remaining 20%
test_data <- data[-estIndex,]
```

- In Python, there is a much simpler approach using Scikit-Learn:



```
from sklearn.model_selection import train_test_split

X = data.drop("Outcome", axis = "columns")
y = data["Outcome"]

X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.2, shuffle = True, random_state = 123,
)
```

## Exercises

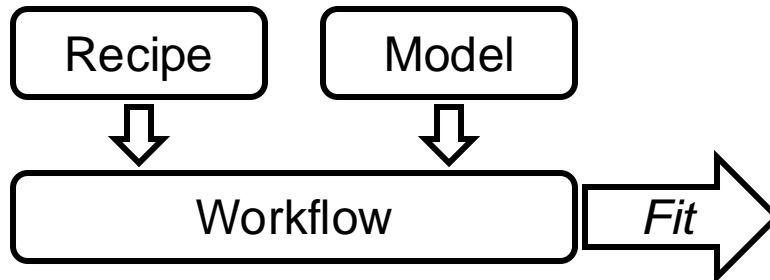


1. Load the iris dataset (*iris.csv* contained in *data.zip*).  
Familiarize yourself with the data set. It contains measurements for 150 flowers of three species of iris. The labels we later want to predict are stored in the column *Species*. All other columns are used as predictors.
2. Perform a sample split retaining 80% of all data for training and 20% for testing.  
Use 123 as seed in both R and Python.

# ML in R

## Machine Learning in R

- In R, the `{tidymodels}` framework offers unified access to all ML algorithms we will use
- With `{tidymodels}`, you create a *recipe* containing the formula and defining any preprocessing as well as a *model* specifying the algorithm and package used
- Both are combined in a *workflow* and then fit using the data



- Keep in mind that `{tidymodels}` does not do the computations itself but only serves as an interface to ML packages that need to be installed separately

# ML in R

## Machine Learning in R

- Schematically, every usage of {tidymodels} will look like this:

```
library(tidymodels)

randomly draw ...% of the observations
estIndex <- sample(nrow(data), ...)

create train and test data
train_data <- data[estIndex,]
test_data <- data[-estIndex,]

define the recipe
rec <- recipe(y ~ X, data = train_data)

define the model
mod <- SomeClassifier() %>%
 set_engine("SomePackage") %>%
 set_mode("classification")

combine both in a workflow
wf <- workflow() %>% add_recipe(rec) %>% add_model(mod)

train the model
clf <- wf %>% fit(data = train_data)

test the model on unseen data
clf %>% predict(test_data)
```



# Scikit-Learn: A Machine Learning Framework

## sklearn: Estimation by API

- scikit-learn is our Python ML framework of choice because of its simplicity and broad range of available methods
- Every algorithm is accessible via a consistent interface, the Estimator API. The only difference between different estimators is the choice of tunable hyperparameters
- Schematically, every usage of scikit-learn will look like this:

```
from sklearn.classifiers import SomeClassifier

create train and test data, split features and label
X_train, X_test, y_train, y_test = train_test_split(X, y, ...)

set up model with hyperparameters
clf = SomeClassifier(precision = 1, overfitting = 0)

train the model
clf = clf.fit(X, y)

test the model on unseen data
clf.predict(X_test, y_test)
```



# Scikit-Learn: A Machine Learning Framework

## sklearn: A basic workflow

1. Choose model class by importing the appropriate estimator class from Scikit-Learn
2. Choose model hyperparameters (instantiate class with desired values)
3. Arrange data into a features matrix and target vector
4. Fit the model to your data by calling the `.fit()` method of the model instance
5. Apply the model to new data

## sklearn: Data Representation

- Scikit-Learn requires you to separate the data into two objects:
  - Features need to be organized in 2-d matrix or array `X`. The data should be tidy, one sample per row, one variable per column
  - The outcome should be stored in a separate array `y`

# A first classification algorithm: kNN

## k-Nearest neighbor (kNN) Classifier

- kNN Classification assumes that examples within each class share similar properties
- Unlabeled examples are assigned the label that the majority of “similar” examples have
- The kNN classifier performs well with high-dimensional data and when relationships between features and outcome is complex
- Similarity is often determined using the Euclidean distance
  - This requires prior scaling of variables!
- The  $k$  term specifies the number of neighbors to use
  - Choose keeping possible overfitting in mind!

# A first classification algorithm: kNN

## kNN: Implementation

```
library(tidymodels)

define the recipe
rec <- recipe(y ~ X, data = train_data)

define the model
mod <- nearest_neighbor() %>%
 set_engine("kknn") %>%
 set_mode("classification")

combine both in a workflow
wf <- workflow() %>% add_recipe(rec) %>% add_model(mod)

train the model
clf <- wf %>% fit(data = train_data)

test the model on unseen data
clf %>% predict(test_data)
```



```
import the required module
from sklearn.neighbors import KNeighborsClassifier

estimate the model
clf = KNeighborsClassifier().fit(X_train, y_train)

predict labels in test sample
clf.predict(X_test)
```



# A first classification algorithm: kNN

## kNN: Example

- Load the packages and set seed

```
import the required packages
library(tidyverse)
library(tidymodels)
the sample split is a random operation; set.seed() ensures replicability
set.seed(123)
```

- Import the data and perform a sample split

```
read the data
penguins <- read_csv("penguins.csv") %>% drop_na()
randomly draw 80% of the observations
estIndex <- sample(nrow(penguins), size = 0.8 * nrow(penguins), replace = FALSE)
subset dataset
train_data <- penguins[estIndex,]
test_data <- penguins[-estIndex,]
```

# A first classification algorithm: kNN

## kNN: Example

- Train and test

```
define the recipe
rec <- recipe(species ~ flipper_length_mm + bill_length_mm, data = train_data)

define the model
mod <- nearest_neighbor() %>%
 set_engine("kknn") %>%
 set_mode("classification")

combine both in a workflow
wf <- workflow() %>% add_recipe(rec) %>% add_model(mod)

train the model
clf <- wf %>% fit(data = train_data)

test the model on unseen data
sum(predict(clf, test_data) == test_data %>% select(species)) / nrow(test_data)
> [1] 0.9701493
```

# A first classification algorithm: kNN



## Example

- Load the modules

```
load the required modules
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

- Import and separate the data

```
read the data
penguins = pd.read_csv("penguins.csv").dropna()
separate X and y
X = penguins[["flipper_length_mm", "bill_length_mm"]]
y = penguins["species"]
```

- Perform the sample split

```
random 80% sample split
X_train, X_test, y_train, y_test = train_test_split(
 X, y,
 test_size = 0.2, shuffle = True, random_state = 123,
)
```

- Train and test

```
estimate the model
clf = KNeighborsClassifier().fit(X_train, y_train)

get the share of correct classifications in test sample
clf.score(X_test, y_test)
Out[1]: 0.9701493
```

# Exercises

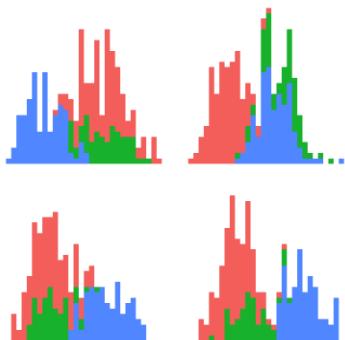


3. Classify the species of iris using a k-nearest-neighbors classifier. Make sure to use all available features and only train using the designated training data
4. Calculate the share of correct classifications when predicting labels in the test data

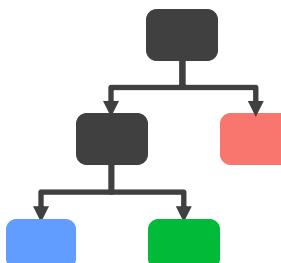
# A second classification algorithm: Random Forest Classifiers

## Random Forests

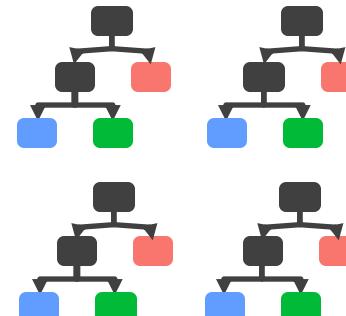
- Using flexible and simple randomly generated decision trees to classify data. This is usually very fast
- Given single trees tend to overfit the data, an ensemble of trees ("Forest") improves out-of-sample performance ("bagging")
- However, final decision rules are hard to interpret



Histograms of  
Data Variables



Single Decision Tree



Forest of many  
decision trees

# A second classification algorithm: Random Forest Classifiers

## RF: Implementation

```
library(tidymodels)

define the recipe
rec <- recipe(y ~ X, data = train_data)

define the model
mod <- rand_forest() %>%
 set_engine(engine = "randomForest") %>%
 set_mode("classification")

combine both in a workflow
wf <- workflow() %>% add_recipe(rec) %>% add_model(mod)

train the model
clf <- wf %>% fit(data = train_data)

test the model with unseen data
clf %>% predict(test_data)

import the required module
from sklearn.ensemble import RandomForestClassifier

estimate the model
clf = RandomForestClassifier(random_state=123).fit(X_train, y_train)

predict labels in test sample
clf.predict(X_test)
```



# Performance Measures: Confusion Matrix

## Confusion Matrix

- Compare predicted with actual labels in the test data set
- Example for True/False-labeled outcome
  - True Positives (TP): actual true, predicted true
  - True Negatives (TN): actual false, predicted false
  - False Positives (FP): actual false, predicted true
  - False Negatives (FN): actual true, predicted false



```
library(caret)
both predictions and actuals
(reference) need to be factors
confusionMatrix(y_predicted, y_test)
```



```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_predicted)
```

|           |       | Actual |       |
|-----------|-------|--------|-------|
|           |       | True   | False |
| Predicted | True  | TP     | FP    |
|           | False | FN     | TN    |

# Performance Measures: Accuracy, Precision, Recall and Specificity

## Accuracy

- Share of correctly specified cases
- Good measure when outcome classes in data are nearly balanced



```
caret::confusionMatrix(y_predicted, y_test)[["overall"]][["Accuracy"]]
```



```
sklearn.metrics.accuracy_score(y_true, y_predicted)
```

|           |       | Actual |       |
|-----------|-------|--------|-------|
|           |       | True   | False |
| Predicted | True  | TP     | FP    |
|           | False | FN     | TN    |

$$\frac{TP + TN}{TP + FP + FN + TN}$$

## Precision

- Share of positives correctly specified
- Useful if FP should be avoided



```
caret::precision(y_predicted, y_test)
```



```
sklearn.metrics.precision_score(y_true, y_predicted)
```

|           |       | Actual |       |
|-----------|-------|--------|-------|
|           |       | True   | False |
| Predicted | True  | TP     | FP    |
|           | False | FN     | TN    |

$$\frac{TP}{TP + FP}$$

# Performance Measures: Accuracy, Precision, Recall and Specificity

## Recall (Sensitivity)

- Share of actual true values found
- Useful if FN should be avoided



```
caret::recall(y_predicted, y_test)
```



```
sklearn.metrics.recall_score(y_true, y_predicted)
```

|           |       | Actual               |       |
|-----------|-------|----------------------|-------|
|           |       | True                 | False |
| Predicted | True  | TP                   | FP    |
|           | False | FN                   | TN    |
|           |       | $\frac{TP}{TP + FN}$ |       |

## Specificity

- Opposite of recall
- Performance w.r.t. false positives



```
caret::specificity(y_predicted, y_test)
```



→ needs to be computed manually (e.g. using {numpy})

|           |       | Actual               |       |
|-----------|-------|----------------------|-------|
|           |       | True                 | False |
| Predicted | True  | TP                   | FP    |
|           | False | FN                   | TN    |
|           |       | $\frac{TN}{FP + TN}$ |       |

## Exercises



5. Calculate a Random Forest Model for the classification problem from exercise 3
6. Calculate the confusion matrix for the Random Forest Model. How accurate is your model?



7. Repeat the random sample split – this time, however, using **42** as seed. Subsequently, re-estimate the random forest model and inspect the model's performance on the test data. What do you notice? How can this be explained?

# A first unsupervised algorithm: k-means Clustering

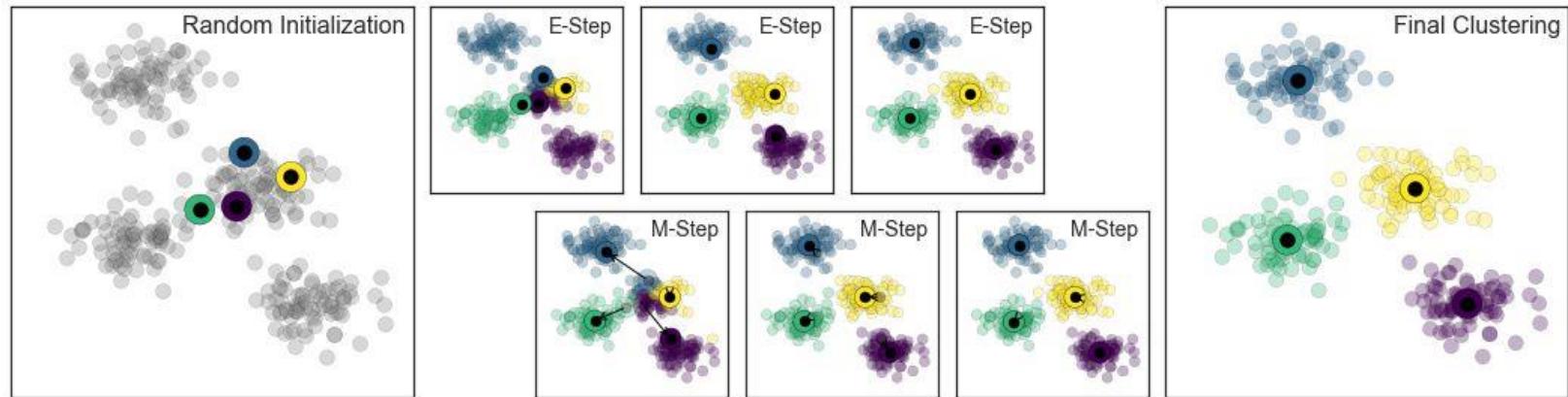
## k-means Clustering

- Clustering seeks to divide the data into clusters, discovering natural groupings of data
- k-means is perhaps the most frequently used clustering algorithm
- k-means searches for a predetermined number of clusters  $k$ 
  - Defining the correct number of clusters is difficult. Use a-priori-knowledge if available

# A first unsupervised algorithm: k-means Clustering

## k-means Clustering: Procedure

1. Guess some cluster centers
2. Repeat the following until convergence:
  - a) E-Step: assign points to the nearest cluster center
  - b) M-Step: set new cluster centers at the mean of the clusters



# A first unsupervised algorithm: k-means Clustering

## k-means clustering: Implementation



```
k-means is not (yet) implemented in {tidymodels}
→ we use the {stats} approach

calculate the clusters
kclust ← kmeans(data, centers = 3, nstart = 25)

get cluster sizes
kclust$size

get cluster centroids
kclust$centers
```



```
import the required modules
from sklearn.cluster import KMeans

calculate the clusters
kclust = KMeans(n_clusters = 3, n_init = 25).fit(X)

get cluster sizes
pd.Series(kclust.labels_).value_counts()

get cluster centroids
kclust.cluster_centers_
```

## Exercises



8. Calculate a reasonable number of clusters using the k-means algorithm. Display a scatterplot, plotting petal length against petal width and mapping each observation's cluster to a color.

# A second unsupervised algorithm: Principal Component Analysis

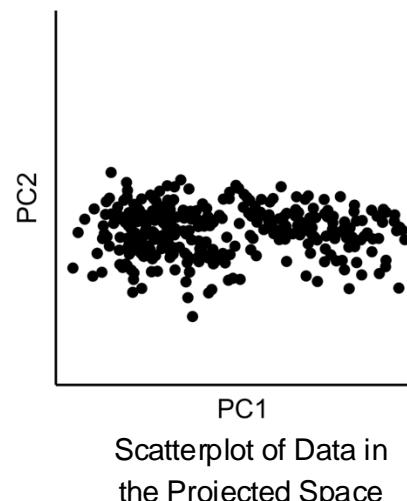
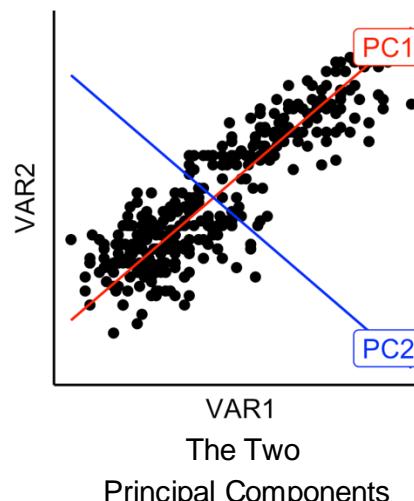
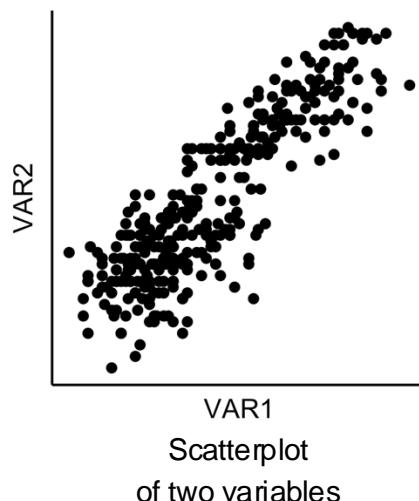
## Principal Component Analysis (PCA)

- PCA can be used for **Dimension Reduction**, decreasing the number of variables
- This is very useful for visualization and to circumvent collinearity
- After PCA, you can project data in the new space that is spanned by the Principal Components (PCs)
- PCA can deliver as many PCs as variables
- There are rules of thumb on how many variables to retain, i.e., looking at the ‘elbow’ in the Screeplot or at the Eigenvalues
- Keep in mind that PCA is **unsupervised** and information on any labels is disregarded

# A second unsupervised algorithm: Principal Component Analysis

## PCA: Intuition

- PCA is greedy: It looks for new variables that each contain as much information as possible
- Below you see an example of a PCA on the flipper length and body mass of penguins, delivering two PCs





# A second unsupervised algorithm: Principal Component Analysis

## PCA: Implementation

- Note that in {tidymodels}, PCA is a preprocessing step. It is added to the recipe like this:

```
define the recipe
rec <- recipe(y ~ X, data = train_data) %>%
 step_scale(all_predictors()) %>%
 step_pca(all_predictors())
```

- If you want to perform the PCA and have full access to the calculated variables, use this approach:

```
calculate the PCA; remember to scale your data
pca <- prcomp(data, scale = TRUE)

project the data on the new axes
data_projected <- predict(pca, data)
```



# A second unsupervised algorithm: Principal Component Analysis

## PCA: Implementation

```
import the required modules
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

always scale variables!
data = StandardScaler().fit_transform(data)

calculate the PCA
pca = PCA().fit(data)

project the data on the new axes
data_projected = pca.transform(data)
```

# A second unsupervised algorithm: Principal Component Analysis

## PCA: Implementation, Screeplot



```
load the required package
library(factoextra)

display the Eigenvalues
get_eigenvalue(pca)

plot a screeplot and look for 'elbow'
fviz_eig(pca)
```



```
import the required modules
import seaborn as sns

display the Eigenvalues
pca.explained_variance_

plot a screeplot and look for 'elbow'
ax = sns.lineplot(x = range(len(pca.explained_variance_ratio_)),
 y = pca.explained_variance_ratio_)

add some visual improvements
ax.set_xticks(range(len(pca.explained_variance_ratio_)))
ax.set_xlabel('number of components')
ax.set_ylabel('eigenvalues')
```

# A second unsupervised algorithm: Principal Component Analysis

## PCA: Implementation, Example



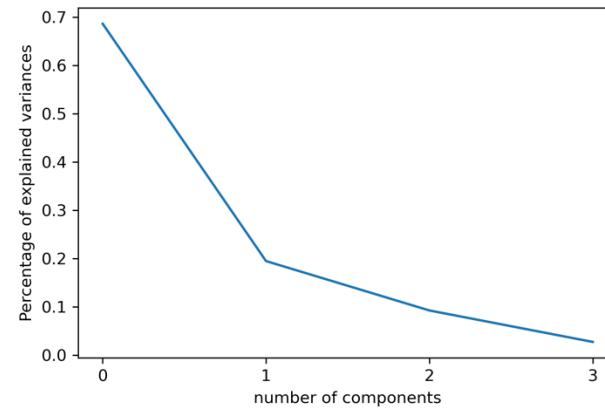
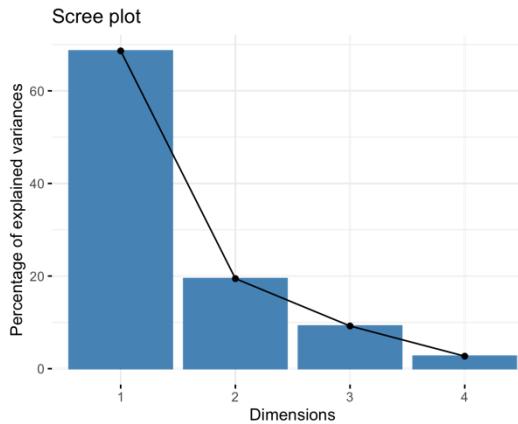
```
display the Eigenvalues
get_eigenvalue(pca)$"eigenvalue"

> [1] 2.7453557 0.7781172 0.3686425
0.1078846
```

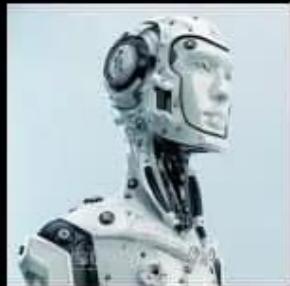


```
display the Eigenvalues
pca.explained_variance_

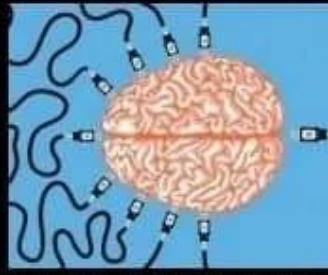
> array([2.75362487, 0.7804609 ,
0.36975289, 0.10820954])
```



# Machine Learning



What society thinks I do.



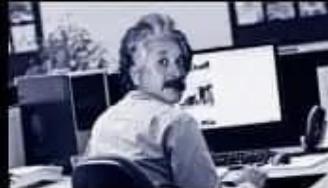
What my friends thinks I do.



What computer scientists think I do.



What my boss thinks I do.

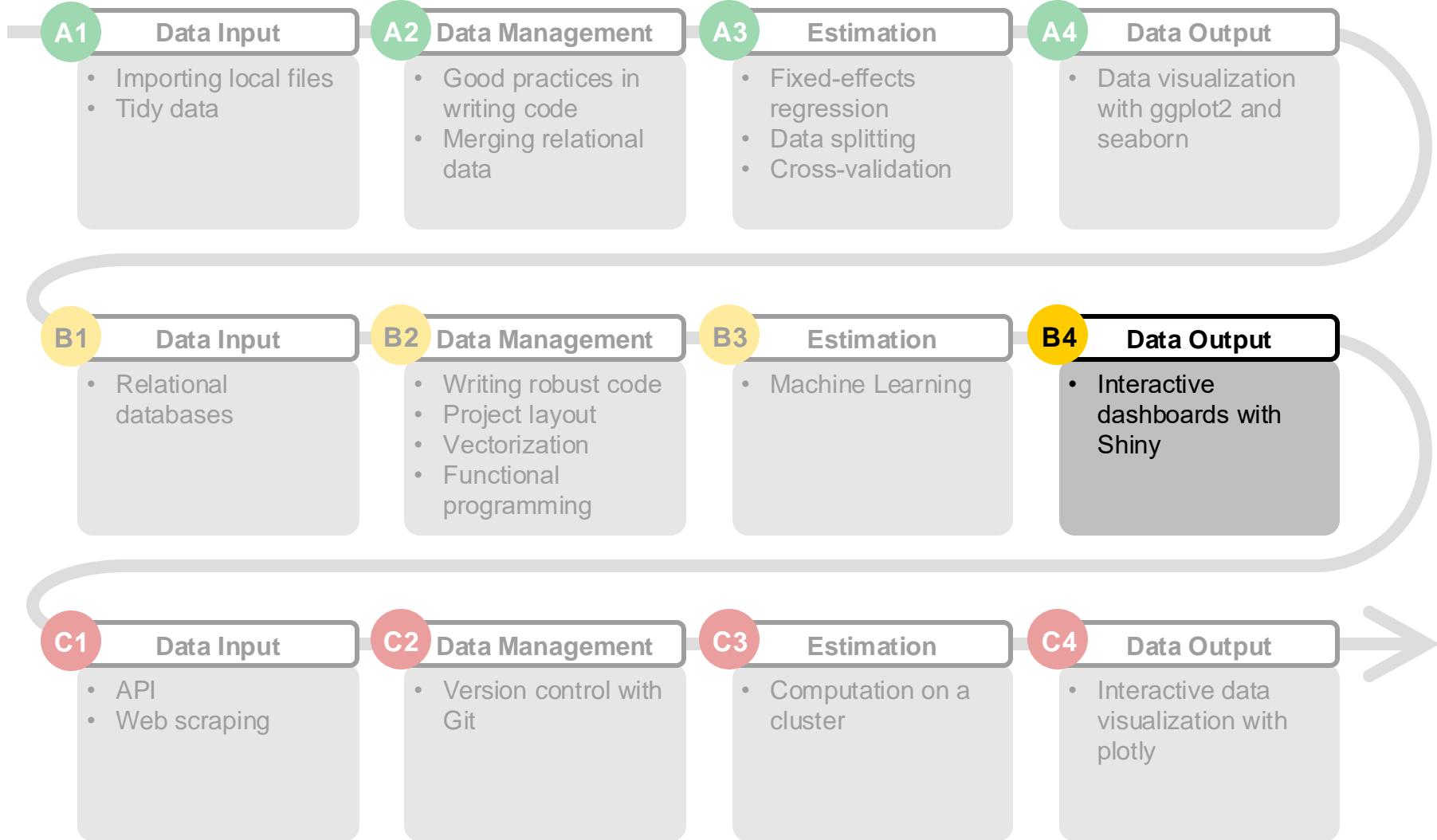


What I think I do.



What I really do.

# Data Science Project Management: Course Outline



# Dashboards - An example

## k-means clustering (Michael Jackson Songs)



[https://davidgremminger.shinyapps.io/DSPM\\_spotify\\_shiny\\_app/](https://davidgremminger.shinyapps.io/DSPM_spotify_shiny_app/)

## A picture is worth a thousand words (or a thousands of lines of code)

- A data scientist's most sophisticated model and code is worth nothing unless someone in the business is able to use the insight to make a better decision.
- Intuitive, self-explanatory, and flexible visualization of insights is therefore essential for informing decision-makers.
- Dashboards offer an ideal solution, combining complex information with interactivity accessible through a web browser.
- There are various dashboarding solutions in R and Python. In this course, we focus on Shiny.

# Shiny – what it is and what it can do

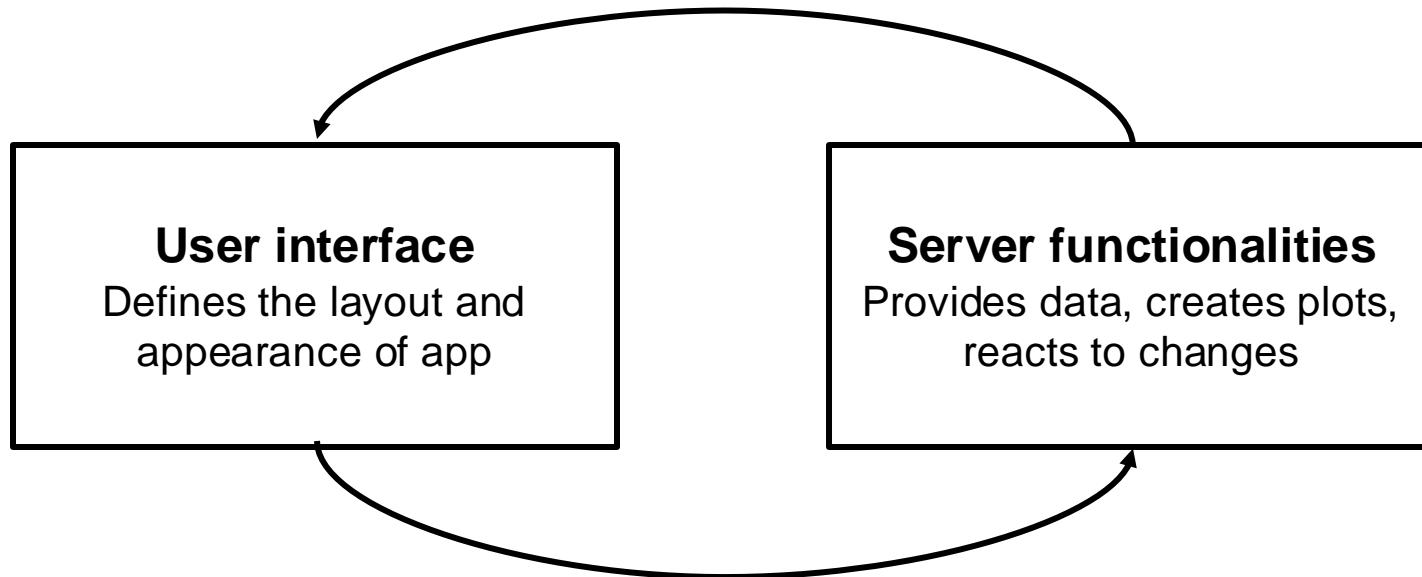


- Web application framework (user interface is an HTML web page)
- Anyone with knowledge of coding in R or Python can start to build applications that sit in a web browser (there are no requirements for web developers to get involved)
- Dashboards are basically an interface to the programming language – meaning your application can do whatever R/Python can (if you allow it to)
- Example use cases of a Shiny app:
  - Creating interactive dashboards to perform what-if analyses
  - Programming apps that produce common reports, letting users upload their own data that can be viewed in a standard way

## But what if the decision-makers don't know code? —

- The web application can be deployed centrally and shared as a URL, just like any other web page
- There is no need for the decision-maker to have any knowledge of coding

# The anatomy of a Dashboard





## ... and how it looks in code – the skeleton



```
Import necessary packages
Install via install.packages("shiny")
library(shiny)

Define the user interface
ui <- fluidPage(
 ...
)

Define server functionalities
server <- function(input, output) {
 ...
}

Putting both together into the app
shinyApp(ui = ui, server = server)
```



```
Import necessary packages
Install via pip install shiny
from shiny import ui, render, App

Define the user interface
app_ui = ui.page_fluid(
 ...
)

Define server functionalities
def server(input, output, session):
 ...

Putting both together into the app
app = App(app_ui, server)
```

# hello world! example



```
library(shiny)

user interface
(here, a single page with a single text)
ui <- fluidPage(
 "Hello world!"
)

server logic
(here, a function that does nothing)
server <- function(input, output) {

}

putting both together into the app
shinyApp(ui = ui, server = server)
```



```
from shiny import ui, render, App

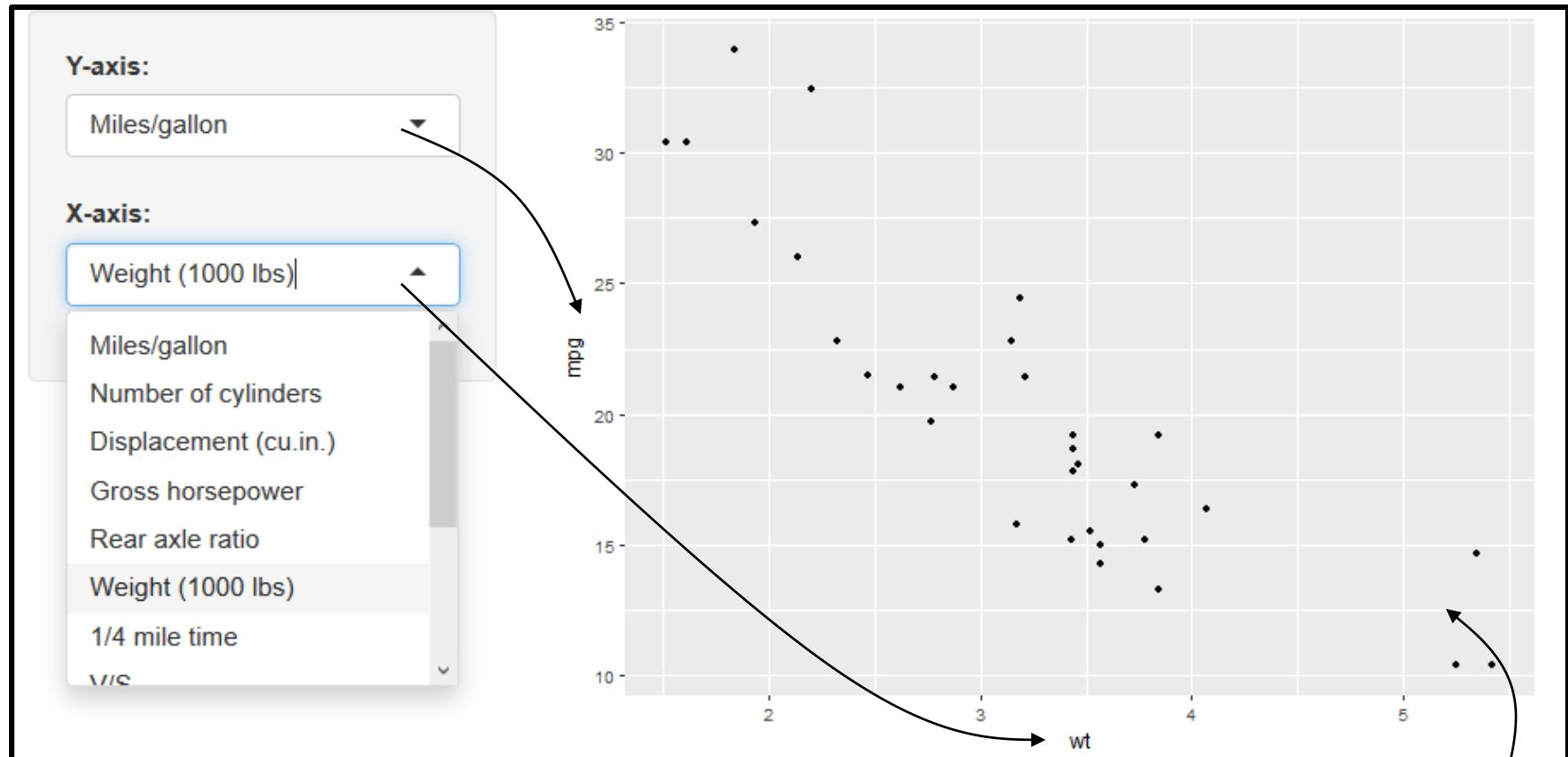
user interface
(here, a single page with a single text)
app_ui = ui.page_fluid(
 "Hello world!"
)

server logic
(here, a function that does nothing)
def server(input, output, session):
 ...

putting both together into the app
app = App(app_ui, server)
```

User inputs are declared in the UI, the contents in the server function

ui



server

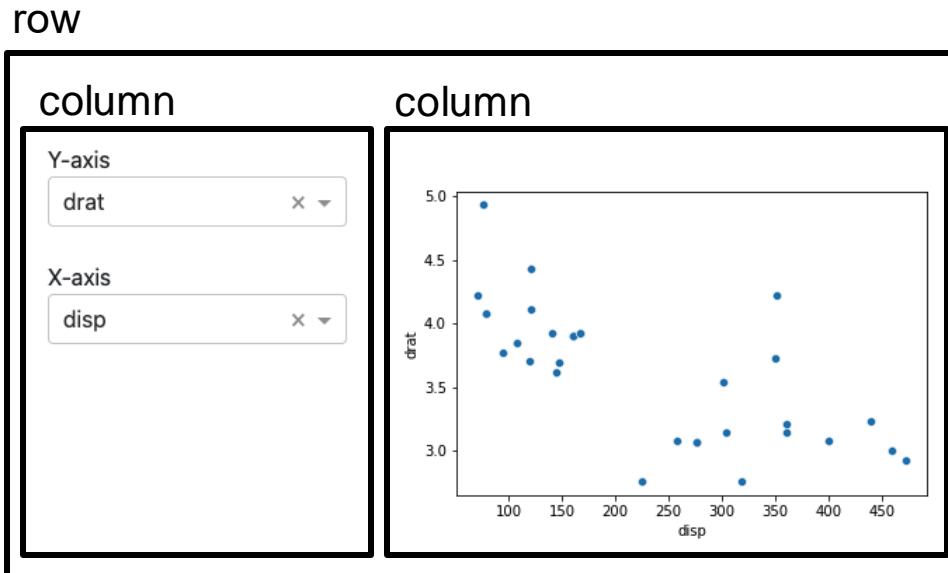
```
ggplot(data = mtcars, aes_string(x = input$x, y = input$y)) + geom_point()
```

[https://davidgremminger.shinyapps.io/DSPM\\_mtcars\\_shiny\\_app/](https://davidgremminger.shinyapps.io/DSPM_mtcars_shiny_app/)

# Dashboard UI - A (very) short introduction to Bootstrap HTML

## HTML

- We will use Bootstrap layouts for our HTML frontends
  - Bootstrap builds webpages as rows and columns
  - The dashboard below features one row and two columns
- In Shiny you can use lots of predefined building blocks



# Building a first dashboard - UI layout



```
ui <- fluidPage(
 fluidRow(
 column(htmltools::h1("Left column"),
 width = 3),
 column(htmltools::h1("Center column"),
 width = 6),
 column(htmltools::h1("Right column"),
 width = 3)
)
)
```

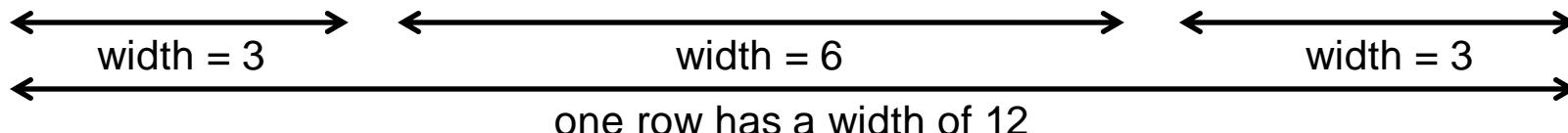


```
app_ui = ui.page_fluid(
 ui.row(
 ui.column(3, ui.h1("Left column")),
 ui.column(6, ui.h1("Center column")),
 ui.column(3, ui.h1("Right column"))
))
```

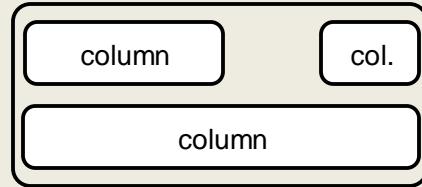
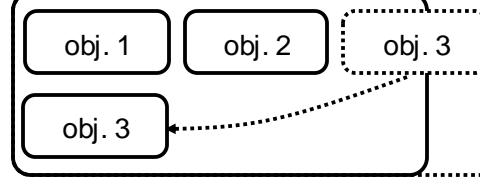
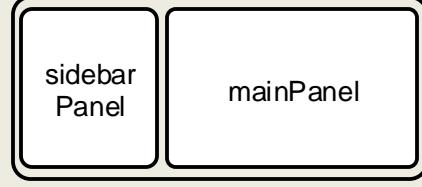
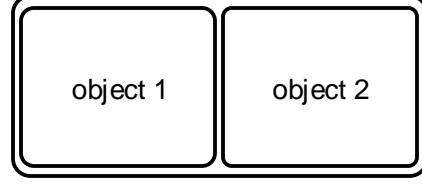
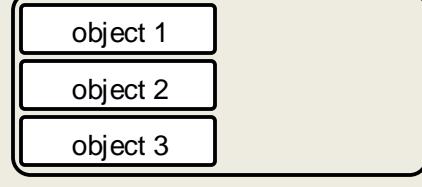
**Left  
column**

**Center column**

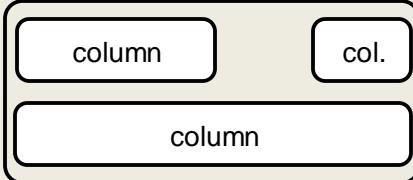
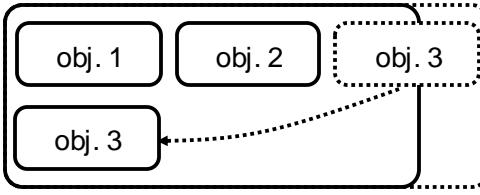
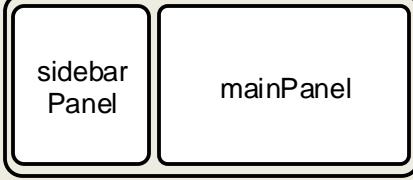
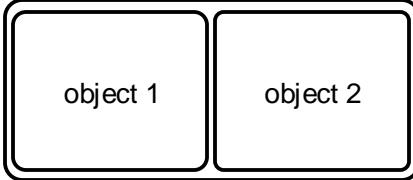
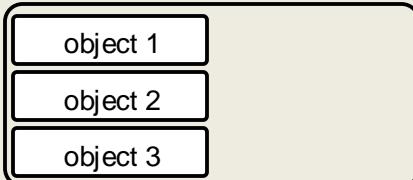
**Right  
column**



# Shiny provides a wide range of predefined Layouts

| Layout                  | Description                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>fluidRow()</b>       | <ul style="list-style-type: none"><li>Rows for the purpose of making sure their elements appear on the same line.</li><li>Columns to define how much horizontal space within a 12-unit wide grid its elements should occupy.</li></ul>                                                              |
| <b>flowLayout()</b>     | <ul style="list-style-type: none"><li>Lays out elements in a left-to-right, top-to-bottom arrangement.</li><li>The elements on a given row will be top-aligned with each other.</li></ul>                                                                                                           |
| <b>sidebarLayout()</b>  | <ul style="list-style-type: none"><li>Creates a layout with a sidebar and main area.</li><li>The sidebar is displayed with a distinct background color and typically contains input controls.</li><li>The main area occupies 2/3 of the horizontal width and typically contains outputs.</li></ul>  |
| <b>splitLayout()</b>    | <ul style="list-style-type: none"><li>Lays out elements horizontally, dividing the available horizontal space into equal parts (by default).</li></ul>                                                                                                                                             |
| <b>verticalLayout()</b> | <ul style="list-style-type: none"><li>Create a container that includes one or more rows of content (each element passed to the container will appear on its own line in the UI)</li></ul>                                                                                                         |

# Shiny provides a wide range of predefined Layouts

| Layout                  | Sample code                                                                                                                                               |                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <b>fluidRow()</b>       | <pre>ui &lt;- fluidPage(<br/>  fluidRow(column(width = 6),<br/>            column(width = 3, offset = 3)),<br/>  fluidRow(column(width = 12))<br/>)</pre> |    |
| <b>flowLayout()</b>     | <pre>ui &lt;- fluidPage(<br/>  flowLayout(<br/>    object 1,<br/>    object 2,<br/>    object 3<br/>)</pre>                                               |    |
| <b>sidebarLayout()</b>  | <pre>ui &lt;- fluidPage(<br/>  sidebarLayout(<br/>    sidebarPanel(),<br/>    mainPanel()<br/>)<br/>)</pre>                                               |    |
| <b>splitLayout()</b>    | <pre>ui &lt;- fluidPage(<br/>  splitLayout(<br/>    object 1,<br/>    object 2<br/>)<br/>)</pre>                                                          |   |
| <b>verticalLayout()</b> | <pre>ui &lt;- fluidPage(<br/>  verticalLayout(<br/>    object 1,<br/>    object 2,<br/>    object 3<br/>)</pre>                                           |  |



Most of these layouts have a python equivalent



fluidPage()

ui.page\_fluid()

sidebarLayout()

ui.layout\_sidebar()

...

...

## Shiny for python is quite new

- Since Shiny for Python is a relatively new development, it does not yet support all features that Shiny has in R.
- However, the development is actively ongoing. Hence, more and more features are translated to Shiny for Python.

# A variety of user inputs can be defined in the UI

Type of user input...



...and what it can look like in the app:

```
actionButton(inputId, label, ...)
```

```
ui.inputActionButton(id, label, ...)
```

[Click here](#)

```
checkboxInput(inputId, label, value)
```

```
ui.inputCheckbox(id, label...)
```

Click me

```
checkboxGroupInput(inputId, label, choices, selected, inline, ...)
```

```
ui.inputCheckboxGroup(id, label, ...)
```

Choice 1  Choice 2

```
dateInput(inputId, label, value, min, max, format, ...)
```

```
ui.inputDate(id, label, value, min, max, format, ...)
```



```
dateRangeInput(inputId, label, start, end, min, max, format, ...)
```

```
ui.inputDateRange(id, label, start, end, min, max, format, ...)
```



# A variety of user inputs can be defined in the UI (contd.)

Type of user input...



...and what it can look like in the app:

```
numericInput(inputId, label,
 value, min, max, step)
```

```
ui.input_numeric(id, label,
 value, min, max, ...)
```

```
radioButtons(inputId, label,
 choices, selected, ...)
```

```
ui.input_radio_buttons(id, label,
 choices, selected, ...)
```

Choice 1  Choice 2

```
selectInput(inputId, label,
 choices, selected, multiple,
 ...)
```

```
ui.input_select(id, label,
 choices, selected, multiple,
 ...)
```

Choice 1

Choice 2

Choice 3

```
sliderInput(inputId, label,
 min, max, value, step,
 ticks, ...)
```

```
ui.input_slider(id, label,
 min, max, value, step,
 ticks, ...)
```





# Building a first dashboard - adding user input

R

```
ui <-
 fluidPage(
 fluidRow(
 selectInput(inputId = "dspm_great",
 label = "How great is this lecture?",
 choices = list("Great" = 1,
 "Greater" = 2,
 "The Best" = 3),
 selected = 3)
)
)
```

Python

```
app_ui = ui.page_fluid(
 ui.row(
 ui.input_select(id="dspm_great",
 label="How great is this lecture?",
 choices=["Great",
 "Greater",
 "The Best"],
 selected="The Best")
)
)
```

How great is this lecture?

The screenshot shows a dropdown menu with the following options:

- Great
- Greater
- The Best**



# Running your app

## Running your app locally

- For testing and debugging you can host the dashboard on your local machine
- It can then be accessed via a web browser



```
shinyApp(ui = ui, server = server)
Listening on http://127.0.0.1:3406
Browsing http://127.0.0.1:3406
```



```
app = App(app_ui, server)
```

```
$ shiny run --reload my_app/app.py
```

```
INFO: Started server process [91136]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```



# Exercises



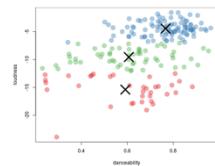
1. Build a very basic application that contains a title and a dropdown menu. Populate the values of the two objects exactly as displayed in the image below. For the server, just use an empty function.

Select the variable to be displayed

bill length

- bill length
- bill depth
- flipper length

# A variety of user outputs can be defined in the UI



`plotOutput()`

|                                                     | danceability | loudness |
|-----------------------------------------------------|--------------|----------|
| Michael Jackson - Earth Song                        | 0.5          | -7.521   |
| Michael Jackson - Euphoria                          | 0.4905       | -9.027   |
| Michael Jackson - Girl Don't Take Your Love From Me | 0.472        | -9.117   |
| Michael Jackson - Greatest Show On Earth            | 0.485        | -9.544   |
| Michael Jackson - Speechless                        | 0.459        | -8.041   |

Showing 1 to 5 of 5 entries

`DT::dataTableOutput()`

| danceability | loudness |
|--------------|----------|
| 0.63         | -17.84   |
| 0.53         | -18.48   |
| 0.62         | -17.92   |

`tableOutput()`



[1] "You have selected 3 clusters."

You have selected 3 clusters.

`imageOutput()`

`verbatimTextOutput()`

`textOutput()`

`ui.output_plot()`

`ui.output_data_frame()`

`ui.output_image()`

`ui.output_text_verbatim()`

`ui.output_text()`

# Providing the content for Tables, Graphs, Pictures etc.

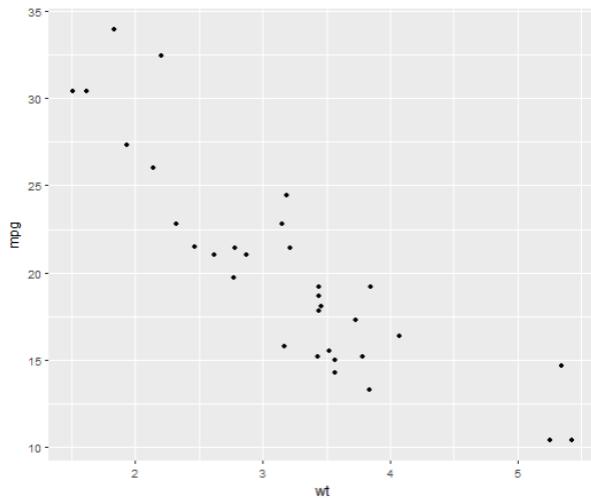
## Dashboard Data

- Data is processed, plots are generated and pictures are converted in the backend or server
- If the content is static (should not be changed through UI), you can just pass some content object to the frontend
  - In R, you **need** to use a `render*` function within the `server` so that the UI understands the data format
  - In Python, you define a function inside the `server` function that is called like the `id` in the `ui.output_*` function and decorate it respectively
- If the content is dynamic (responsive to changes in the UI), the data needs to be processed within the `server`

In the server function, we provide the content of the app

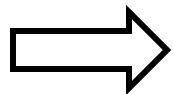
```
server <- function(input, output) {

 # Create the scatterplot object the plotOutput function is expecting
 output$scatterplot <- renderPlot({
 ggplot(data = mtcars, aes_string(x = input$x, y = input$y)) +
 geom_point()
})
```



### Rules of server functions

1. Save objects to display to `output$id`
2. Build objects to display with `render*`
3. Use input values from UI with `input$id`



The plot will be updated automatically when the input changes

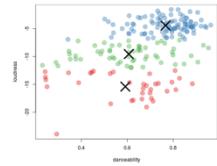
# UI and server function correspondence

**server() function**

`render*`()

**ui() function**

`*Output()`



`renderPlot()`

`plotOutput()`

|                                                     | danceability | loudness |
|-----------------------------------------------------|--------------|----------|
| Michael Jackson - Earth Song                        | 0.5          | -7.521   |
| Michael Jackson - Euphoria                          | 0.4905       | -9.027   |
| Michael Jackson - Girl Don't Take Your Love From Me | 0.472        | -9.117   |
| Michael Jackson - Greatest Show On Earth            | 0.485        | -9.544   |
| Michael Jackson - Speechless                        | 0.459        | -8.041   |

`DT::renderDataTable()`

`DT::dataTableOutput()`

| danceability | loudness |
|--------------|----------|
| 0.63         | -17.84   |
| 0.53         | -18.48   |
| 0.62         | -17.92   |

`renderTable()`

`tableOutput()`



`renderImage()`

`imageOutput()`

```
[1] "You have selected 3 clusters."
```

You have selected 3 clusters.

`renderPrint()`

`verbatimTextOutput()`

`renderText()`

`textOutput()`

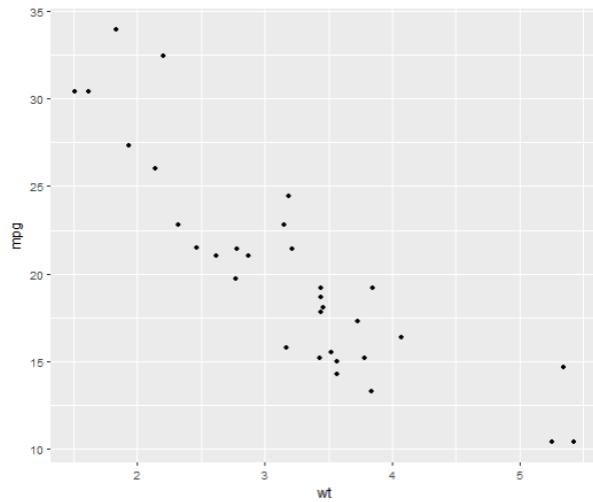


In the server function, we provide the content of the app

```
app_ui = ui.page_fluid(
 ui.output_plot(id="a_scatter_plot"),
)

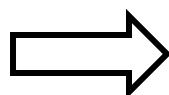
def server(input, output, session):
 # Within the server function, create a function that returns the plot
 @output
 @render_plot
 def a_scatter_plot():
 return plt.scatter(mtcars[input.x()], mtcars[input.y()])

④
```



### Rules of server functions

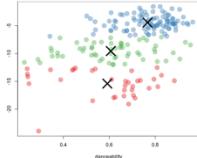
1. Create a function that is called like the id passed to the respective UI function
2. Add respective function decorator
3. Retrieve input values by `input.*()` where \* is the id of the user input in the UI
4. Create and return the object (e.g. the plot)



The plot will be updated automatically when the input changes

# UI and server function correspondence

Match `ui.output_*` functions to `@render.*`

| decorator                                                                                                                                                                                                              | ui function                |                                        |      |        |      |        |      |        |                            |                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|----------------------------------------|------|--------|------|--------|------|--------|----------------------------|--------------------------------|
|                                                                                                                                       | <code>@render.plot</code>  | <code>ui.output_plot()</code>          |      |        |      |        |      |        |                            |                                |
| <table border="1"><thead><tr><th>danceability</th><th>loudness</th></tr></thead><tbody><tr><td>0.63</td><td>-17.84</td></tr><tr><td>0.53</td><td>-18.48</td></tr><tr><td>0.62</td><td>-17.92</td></tr></tbody></table> | danceability               | loudness                               | 0.63 | -17.84 | 0.53 | -18.48 | 0.62 | -17.92 | <code>@render.table</code> | <code>ui.output_table()</code> |
| danceability                                                                                                                                                                                                           | loudness                   |                                        |      |        |      |        |      |        |                            |                                |
| 0.63                                                                                                                                                                                                                   | -17.84                     |                                        |      |        |      |        |      |        |                            |                                |
| 0.53                                                                                                                                                                                                                   | -18.48                     |                                        |      |        |      |        |      |        |                            |                                |
| 0.62                                                                                                                                                                                                                   | -17.92                     |                                        |      |        |      |        |      |        |                            |                                |
|                                                                                                                                       | <code>@render.image</code> | <code>ui.output_image()</code>         |      |        |      |        |      |        |                            |                                |
| <pre>[1] "You have selected 3 clusters."</pre>                                                                                                                                                                         | <code>@render.text</code>  | <code>ui.output_text_verbatim()</code> |      |        |      |        |      |        |                            |                                |
| You have selected 3 clusters.                                                                                                                                                                                          | <code>@render_text</code>  | <code>ui.output_text()</code>          |      |        |      |        |      |        |                            |                                |

<https://rstudio.github.io/cheatsheets/html/shiny-python.html#outputs> (last retrieved on October 18, 2023)

## Exercises



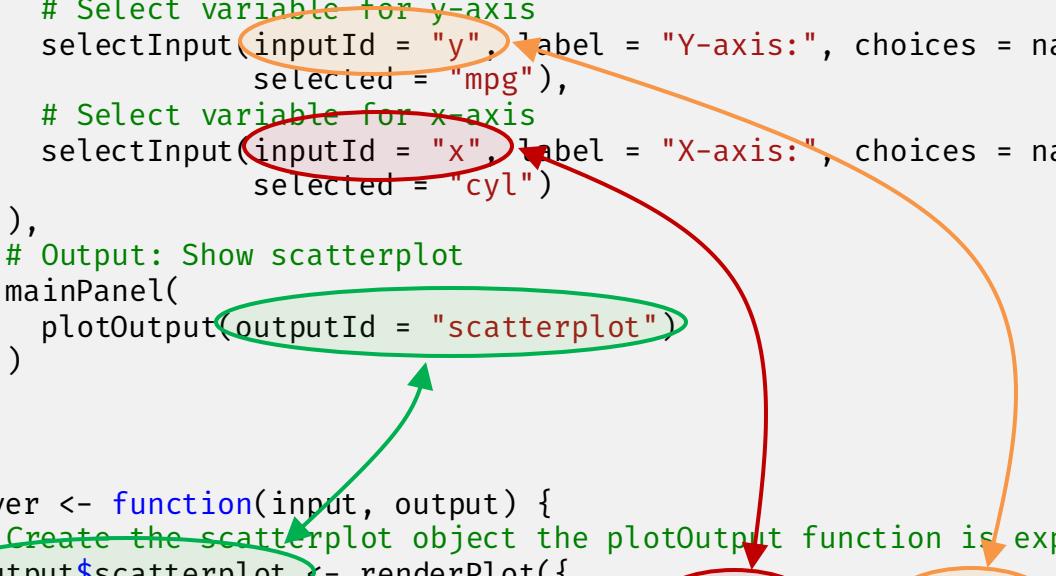
- Now let us add some (static) output to the app. Import *penguins.csv* and plot a histogram of *bill\_length\_mm*, mapping each species to a different color. Insert the plot in a new row below the input in your dashboard.



Customizable input elements receive an ID that is used to refer to them throughout the code

```
ui <- fluidPage(
 # Sidebar layout with input and output definitions
 sidebarLayout(
 # Inputs
 sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:", choices = names(mtcars),
 selected = "mpg"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:", choices = names(mtcars),
 selected = "cyl")
),
 # Output: Show scatterplot
 mainPanel(
 plotOutput(outputId = "scatterplot")
)
)

server <- function(input, output) {
 # Create the scatterplot object the plotOutput function is expecting
 output$scatterplot <- renderPlot({
 ggplot(data = mtcars, aes_string(x = input$x, y = input$y)) +
 geom_point()
 })
}
```





Customizable input elements receive an ID that is used to refer to them throughout the code

```
app_ui = ui.page_fluid(
 # Sidebar layout with input and output definitions
 ui.layout_sidebar(
 # Inputs
 ui.panel_sidebar(
 # Select variable for y-axis
 ui.input_select(id = "y", label = "Y-axis:",
 choices = mtcars.columns.values.tolist(), selected = "mpg"),
 # Select variable for x-axis
 ui.input_select(id = "x", label = "X-axis:",
 choices = mtcars.columns.values.tolist(), selected = "cyl")
),
 # Output: Show scatterplot
 ui.panel_main(
 ui.output_plot(id = "scatterplot")
)
)

def server(input, output, session):
 # Create the scatterplot object the plotOutput function is expecting
 @output
 @render.plot
 def scatterplot():
 return sns.scatterplot(data = mtcars, x = input.x(), y = input.y())
```

# A Shiny app allows for interactions with a plot

- The `plotOutput()` function takes some additional arguments that allow for interaction with graphs:

```
plotOutput("plotID",
 click = "plot_click",
 dblclick = "plot_dblclick",
 hover = "plot_hover",
 brush = "plot_brush"
)
```

- The plot will send coordinates to the server whenever it is clicked, and the value will be accessible via `input$plot_click`.
- The value will be a named list with `x` and `y` elements indicating the mouse position.

Works like the `click` argument, but requires a double-click.

- The plot will send coordinates to the server when the cursor pauses on the plot.
- The value will be accessible via `input$plot_hover`. The value will be a named list with `x` and `y` elements indicating the mouse position.

- The plot will allow the user to "brush" in the plotting area (i.e., to draw a rectangle in the plotting area), and will send information about the brushed area to the server.
- The values will be accessible via `input$plot_brush` (a named list with `xmin`, `xmax`, `ymin`, and `ymax` elements).

## NOTE

- the Python equivalent `ui.output_plot()` has the same parameters
- see [https://shiny.posit.co/py/api/ui.output\\_plot.html](https://shiny.posit.co/py/api/ui.output_plot.html) for details



## Exercises



3. Finally, we want to be able to choose the variable to display using the dropdown menu in the dashboard. Make the app reactive! (Hint: use `aes_string()` to pass the x-axis variable for `{ggplot2}` as a string.)

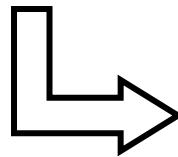
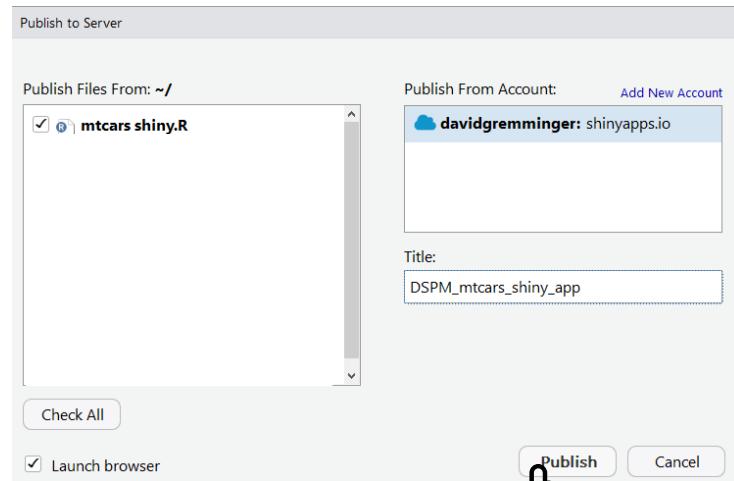
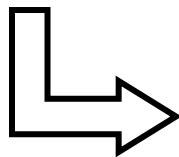
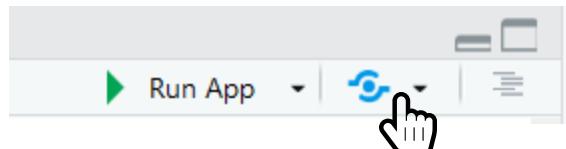


4. Download historical BTC-USD exchange rates from [Yahoo! Finance](#) and use Shiny to reproduce the interactive diagram on [finanzen.net](#) as close as possible.

Shiny apps can easily be published and then be shared/accessed via a hyperlink



NOTE: This requires you to sign up either at ShinyApps.io or RStudio Connect, where your apps are hosted. For free users, the number of publishable apps is limited.



[https://davidgremminger.shinyapps.io/DSPM\\_mtcars\\_shiny\\_app/](https://davidgremminger.shinyapps.io/DSPM_mtcars_shiny_app/)

## Useful material



- <https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/index.html>
- <https://rstudio.github.io/cheatsheets/html/shiny.html>



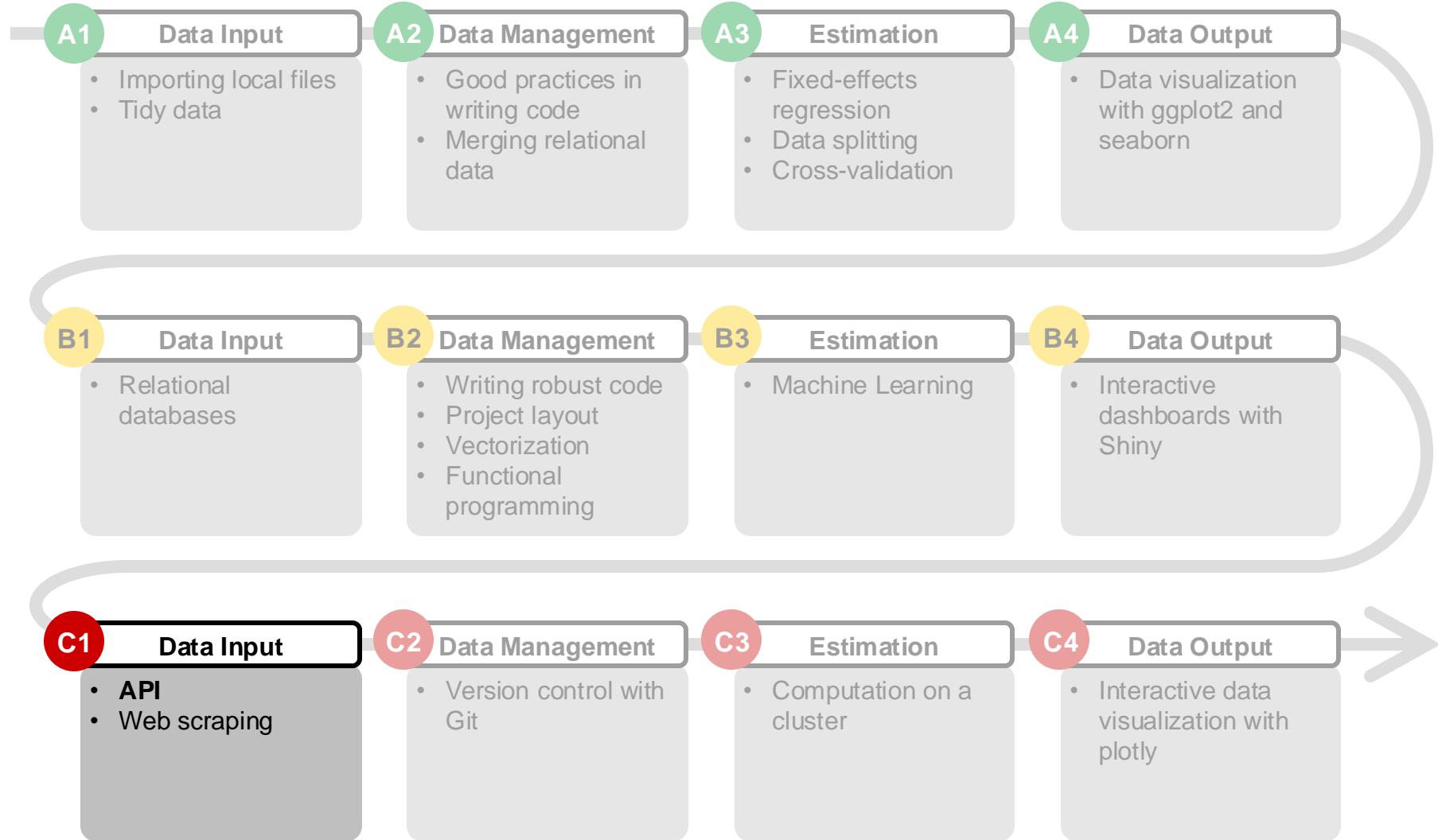
- <https://shiny.posit.co/py/docs/overview.html>
- <https://rstudio.github.io/cheatsheets/html/shiny-python.html>



## Other frameworks in python

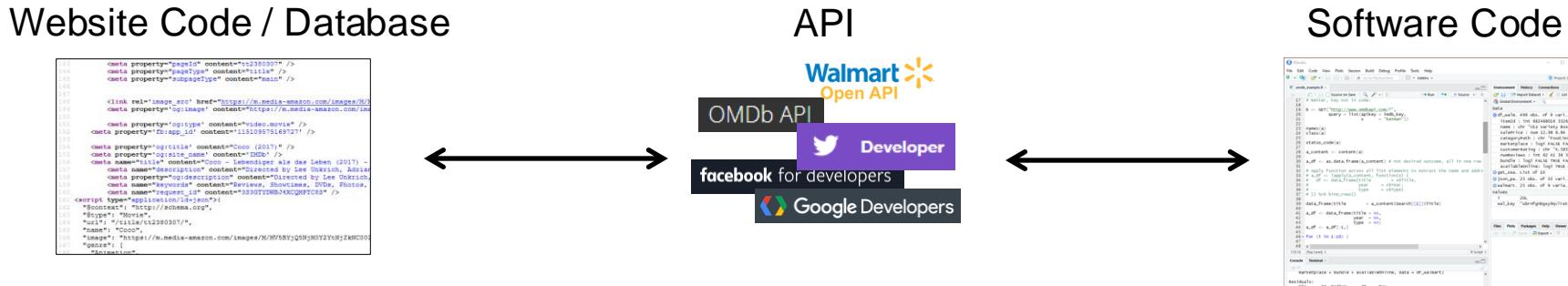
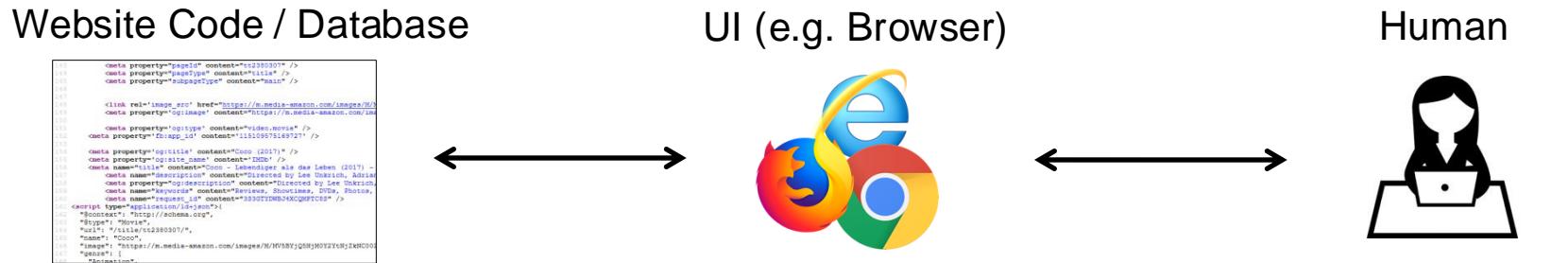
- There are several other popular frameworks to build dashboards in python:
  - [\*\*Dash by Plotly\*\*](#): Interactive visualizations with {plotly}, flask-based backend, react.js components
  - [\*\*Streamlit\*\*](#)
  - [\*\*IPywidgets\*\*](#): Interactivity within Jupyter notebooks
  - [\*\*Voila\*\*](#): Converts Jupyter notebooks into standalone web applications
  - ...

# Data Science Project Management: Course Outline



APIs allow for interactions between applications, data, and devices

- Application Programming Interfaces (APIs) provide methods by which software can communicate with other software
  - Web APIs specifically let code interact with a website through HTTP messages
  - A user interface (UI) (e.g., a web browser) allows humans to interact with the code of a website, while an API allows the code of a program to interact with a website
  - The goal of a user interface is to make code human-readable, while the goal of an API is to make code machine-readable

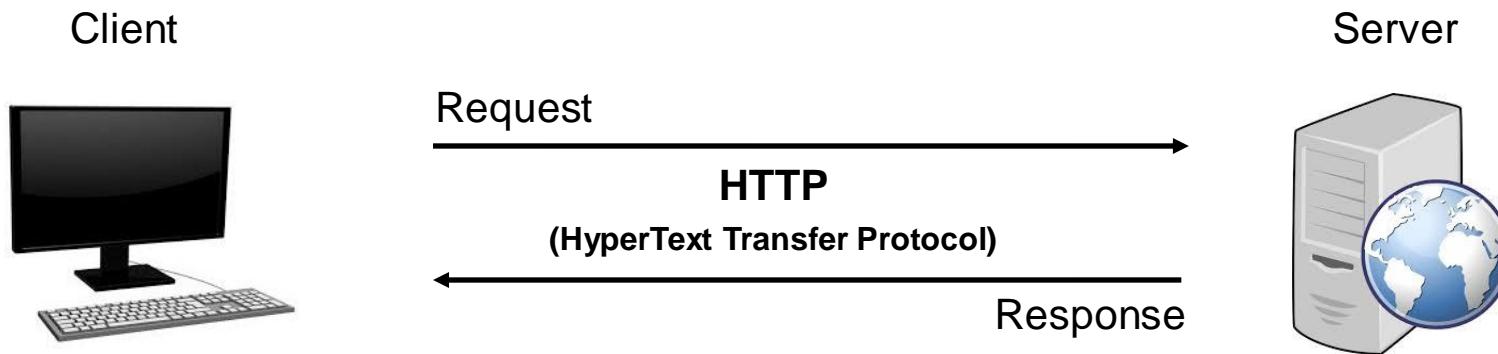


# What is an API?



<https://www.youtube.com/watch?v=s7wmiS2mSXY> (last retrieved on August 05, 2024)

# Client and server communicate exchange information via HTTP

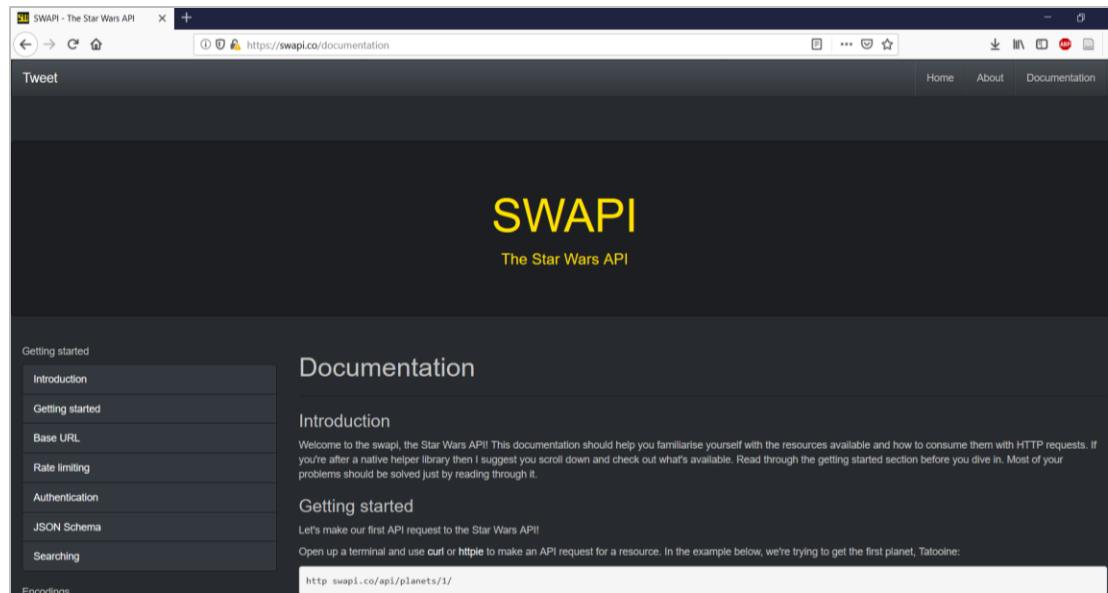


- Interactions with API = conversation between your computer and the server
- Clients says what he wants to happen (request)
- Second step: Server sends the requested data or functionality (response)

## HTTP methods

- `GET()`: retrieve data from the server
- `POST()`: submit data to the server (uploading files, user authentication, ...)
- `HEAD()`: returns metadata about the content (GET without the body of the response)
- `DELETE()`: remove a resource on server

# API documentations state what information can be requested and how



<https://swapi.dev/documentation> (last retrieved on September 03, 2020)

- The documentation shows how the API is structured and how it can be accessed
- Explains what information can be requested
- States how URLs should be constructed to get to specific pieces of information
- Gives example response
- Specifies the format of the response (most common: JSON, XML)
- APIs can be structured very differently and some are documented better than others

# HTTP methods can be applied using R and Python

## Scripted access to Web APIs

- Most data we find on the internet is dynamic and not static, meaning that there are changes being made regularly (e.g. Wikipedia, Facebook, ...)
- In order to keep the data up to date one would have to continuously copy or download the data every time something changes
- We can use R or Python to access and extract data directly from APIs
- We can run the same script on the same API at different times and always get the most up to date data
- Examples:  
*Google Analytics, Google Maps, X, Spotify, Walmart, IMDb, ...*

## Exercises



- In order to be able to access the API needed for the exercises, you are required to request your personal API tokens on <http://www.omdbapi.com>. Once you have your personal key, it is common practice to store it as a string variable in a separate script and access the key like this:

```
store in separate script (e.g. 'API_key.R')
omdb_key <- "yourKey0123456789"
```

```
access the key from main script using
source("API_key.R")
```

```
store in separate script (e.g. 'API_key.py')
omdb_key = "yourKey0123456789"
```

```
access the key from main script using
with open("API_key.py") as script:
exec(script.readline())
```

- This procedure allows you to share your code \*without\* sharing your API key. Note that all the activity that is carried out using your API key is associated with your name and email personal information. Hence, an API key should be treated like a password.

1



# OMDb API

The Open Movie Database

The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

If you find this service useful, please consider making a [one-time donation](#) or [become a patron](#).

## Attention Users

04/08/19 - Added support for eight digit IMDb IDs.  
01/20/19 - Supressed adult content from search results.  
01/20/19 - Added Swagger files ([YAML](#) [JSON](#)) to expose current API abilities and upcoming REST functions.

[Become a Patron](#)

## Sponsors

Emby, Trakt, FileBot, Reelgood, Xirvik Servers, Yidio, mi.tv, Couchpop, What's on Netflix, Edu Reviewer, Flixboss, Str Scripts on Screen, Writers Per Hour, Classic Movies, Medium.com, Write my paper, Best Movie Apps For Android, R sale in Lake Como, Download.it - Streaming Guide, Trading Indicator, iStarTips

Poster API  
The  
Cur  
with

## OMDb API

## API Key

10/05/22 Email Delays! If your requested key doesn't show up within an hour, please contact me directly.

### Generate API Key

Account Type  Patreon  FREE! (1,000 daily limit)

Email

Name

Use

A short description of the application or website that will use this API.

**Submit**

All fields are required.

[Legal](#) [Donate](#)

API by Brian Fritz.

All content licensed under [CC BY-NC 4.0](#).

This site is not endorsed by or affiliated with [IMDb.com](#).

## Libraries for scripted access to Web APIs



{httr}

- tidyverse package that implements the HTTP methods in R
- Includes functions for the most important HTTP verbs: `GET()`, `HEAD()`, `PATCH()`, `PUT()`, `DELETE()`, `POST()`



{requests}

- Provides very similar functionalities in Python
- Also includes all frequently used HTTP verbs

# Libraries for scripted access to Web APIs

## The request & the response

- Send a request to a specified URL ...



```
library(httr)
response <- GET("https://swapi.dev/api/planets/1/")
```



```
import requests
response = requests.get("https://swapi.dev/api/planets/1/")
```

- ... and the server sends back a response



```
class(response)
[1] "response"
```



```
status_code(response)
[1] 200
```



```
type(response)
> requests.models.Response

response.status_code
> 200
```

Status codes inform you in case there is a problem with your request

## Status Codes

- 1xx: Informational Response
- 2xx: Success
- 3xx: Redirecting (additional client action needed)
- 4xx: Client Error (probably error in code)
- 5xx: Server Error (probably error on server side)

[https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

Note: A status code of 2xx only means that the server received our request. It does not tell us whether the request was valid for the API or found the data we were looking for.

Endpoints are the specific URLs from which we request information

### API Endpoint

= URL where the information is requested from

- base endpoint: e.g. <https://swapi.dev/api/>
- Endpoints get more specific depending on searched information, e.g.:  
<https://swapi.dev/api/planets/1/> (directory-based) or  
<https://swapi.dev/api/people/?search=skywalker> (parameter-based)
- The specific endpoints can be found in the documentation. In many cases, we want to use an URL multiple times, but only change a small part of it (e.g., the search query, the page number, ...)

Identify the parts of the URL that change between subsequent requests

## Endpoint URL construction

### 1. Directory-based URLs

- Use directories to represent different parameters  
(e.g.: <https://swapi.dev/api/planets/1/> corresponds to the directory of *planets* and then the *first* entry of this directory)
- Stitch together a query by concatenating directories separated by `" / "`



```
paste("https://swapi.dev/api", "planets", "1", sep = "/")
```



```
"/".join(["https://swapi.dev/api", "planets", "1"])
```

Identify the parts of the URL that change between subsequent requests

## Endpoint URL construction

### 2. Parameter-based URLs

- Use key-value pairs in the URL
- Write the base URL, followed by a question mark ("?") and finally parameters. Write the parameters as `key = value`, separated by "amp;" (e.g. <https://swapi.dev/api/people/?search=skywalker> passes the parameters `search=skywalker` to the endpoint `people`)
- Both `{httr}` and `{requests}` allow you to pass such parameters along using the `query` and `params` arguments respectively. This is especially useful in cases with multiple parameters



```
GET("https://swapi.dev/api/people/",
 query = list("search" = "skywalker"))
```



```
requests.get("https://swapi.dev/api/people/",
 params = {"search": "skywalker"})
```

# Working with the response

## The content

- The response object contains the list element that we want to extract: the content
- Is in raw format (Unicode), which is not human-readable

```
7b 22 6e 61 6d 65 22 3a 22 54 61 74 6f 6f 69 6e 65 22 2c 22 72 6f 74 61
74 69 6f 6e 5f 70 65 72 69 6f 64 22 3a 22 32 33 22 2c 22 6f 72 62
```

- Converting this raw response to a string returns a JSON-formatted string. JSON is a standardized format for data exchange. Data is passed as plain text and can have a complex nested structure
- Recognize familiar objects in the JSON string below:

R List Python Dictionary

```
{'name': 'Tatooine',
 'residents': ['https://swapi.dev/api/people/1/'
 'https://swapi.dev/api/people/62/']}
```

R Vector Python List

## Extract the content from the response object

### Extraction of JSON objects

- Both Libraries provide functions that convert the Unicode to JSON and then immediately parse the JSON to a nested structure of objects



```
content(response)
```



```
response.json()
```

- Rarely, the response of APIs is not JSON. In this case, these functions will not manage to convert the response. If you suspect this is an issue, look at the decoded response



```
content(response, as = "text")
```



```
response.text
```

# There is no one-size-fits-all solution for parsing JSON files

## Tidying

- The exact structure of the content object / JSON / list will vary widely between APIs and requests
- Using an API almost always requires some programming in order to get the nested JSON data into a data frame for comfortable visualization and analysis
- Loops may be necessary when API stores the data on multiple pages

JSON

```
{"name": "Alderaan",
 "films": ["https://swapi.dev/api/films/6/",
 "https://swapi.dev/api/films/1/"],
 "url": "https://swapi.dev/api/planets/2/"}
```

(R) object structure

```
$`name`
[1] "Alderaan"

$films
[1] "https://swapi.dev/api/films/6/"
 "https://swapi.dev/api/films/1/"

$url
[1] "https://swapi.dev/api/planets/2/"
```

Typically, APIs require you to sign up and identify yourself with a key

## Access keys

If too many applications simultaneously request large amounts of data from an API, it can overwhelm the system

Many APIs use access tokens to identify the users making the request and to restrict them in case of problems

Typically, to get an access key, you have to ...

- register your email address
- sometimes explanation of intended use required

→ Access token / API key then must be provided in the request to the API

## Example: Walmart Open API (<https://developer.walmartlabs.com>)



```
R
wal_key <- "ubrnfgnbxy9qv7*****"
res_chips <- GET(url = "http://api.walmartlabs.com/v1/search?",
 query = list(apiKey = wal_key,
 query = "chips"))
```



**Register**

Get access keys and usage  
reports for your apps

### Note:

It is good practice to always store your identification (here: the API key) outside of your main code. You will want to share your code, but not your private authentication.

# Most APIs limit the amount and/or rate of requests per user

## Different accessibility of APIs

- Some are publicly available (e.g. Github, swapi)
- Some are only available after registration and obtaining an API key (e.g. Twitter, Walmart, Spotify, omdb, ...)
- Some provide only limited access for free users, more access and convenience for paid users
- Some are only available after paying a fee

## Rate limiting

- Many APIs limit the number of requests you can make in a given time period
- Thereby, they prevent overwhelming the system
- If you make more requests than allowed, you will get blocked
- Make sure there is an interval between your requests (= rate limiting)
- Use `Sys.sleep()` and `time.sleep()` in R and Python respectively to stall within a loop

| Key Rate Limits |                  |
|-----------------|------------------|
| 5               | Calls per second |
| 5,000           | Calls per day    |

<https://developer.walmartlabs.com/member>

## Exercises



1. Use the OMDb API in order to import all movies (not series or episodes) which contain the word "batman" in the movie title.

Hints:

1. Make yourself familiar with the API by carefully reading its documentation
2. As it is common for many APIs, results of the OMDb API are chunked and presented on different pages. Identify how many movies there are in total for the given search term and on how many pages they are presented via the API
3. Construct an empty data frame where you can store the movie information later
4. Use a loop or `apply()`/`map()`-approach in order to get the movies from all pages

# API clients can simplify our work a lot!

## Using specific API clients

- For many APIs there is a package that wraps API calls, makes the request in the background and parses the response into a clean format
- Very easy to use: take care of the complex structure of an API and return the requested data in convenient objects
- Rule of thumb: Always use a client if there is a good one for your problem  
→ You should only write code that is necessary
- BUT: Be careful, the people who have written the client might not have had the same goals or problems that you do (e.g. some endpoints might not be covered by the package, but are of interest for your problem)

## Example: A client for the Star Wars API



```
devtools::install_github("davidgremminger/rwars")
library(rwars)
people <- get_all_people()
people2 <- get_all_people(getElement(people, "next"))
planet17 <- get_planet(id = 17)
```

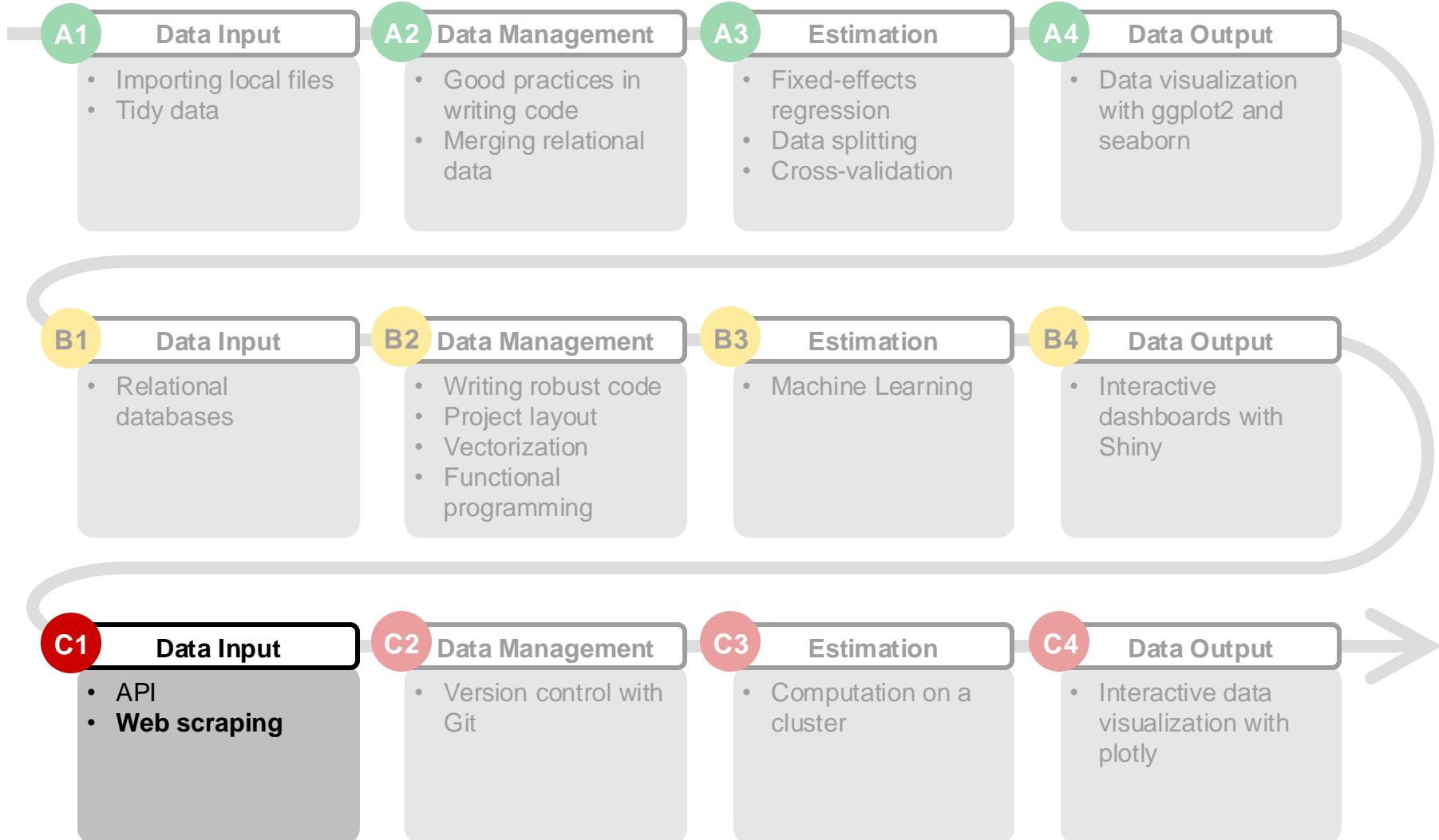
The original [{rwars}](#) package has been removed from the CRAN and is no longer maintained by the author. Minor changes have been made to the deprecated code and a functioning version (as of September 2020) of the package is available on <https://github.com/davidgremminger/rwars>.

## Exercises



2. There are clients available to access the OMDb API (`{imdbapi}` in R, `{omdb}` in Python). Repeat exercise 1 using an appropriate function from these packages (the documentation will help figuring out what function to use).

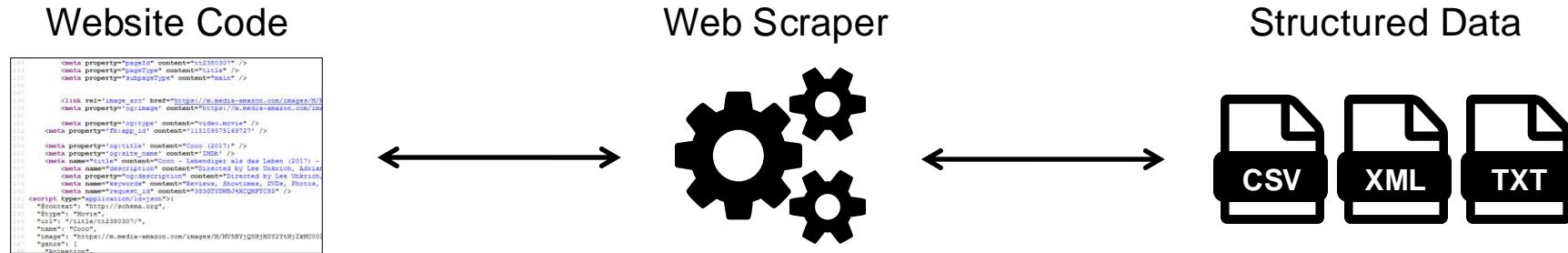
# Data Science Project Management: Course Outline



# Web scraping – what it is and what it does

## Scraping...

- ... describes a process used for extracting and automatically retrieving data from websites and documents
  - ... converts human readable into machine readable data
  - ... separates data from format



# Web scraping - terminology

## Web Scraping

Web scraping is the act of automatically downloading a web page's data and extracting very specific information from it

## Web Crawling

Web crawling is the act of automatically downloading a web page's data, extracting the hyperlinks it contains and following them

## Legal aspects

- Web scraping and crawling are not illegal by themselves.
- Problems can arise when you scrape or crawl the website of somebody else, without obtaining their prior written permission, or in disregard of their Terms of Service



# Some general rules for web scraping should be respected

## Nettiquette

- Use an API if one is provided, instead of scraping data
- Respect the Terms of Service and the rules of robots.txt
- Use a reasonable crawl rate, i.e. don't bombard the site with requests.  
Respect the crawl-delay setting provided in robots.txt; if there's none, use a conservative crawl rate (e.g., 1 request per second)
- Identify your web scraper or crawler with a legitimate user agent string
- If ToS or robots.txt prevent you from crawling or scraping, ask a written permission to the owner of the site, prior to doing anything else
- If you doubt on the legality of what you're doing, don't do it



# A robots.txt file is often provided that tells you what is (not) allowed

```
Every bot that might possibly read and respect this file.
User-agent: *
Allow: /*?lang=
Allow: /hashtag/*?src=
Allow: /search?q=%23
Disallow: /search/realtime
Disallow: /search/users
Disallow: /search/*grid

Disallow: /*?
Disallow: /*/followers
Disallow: /*/following

Disallow: /account/deactivated
Disallow: /settings/deactivated

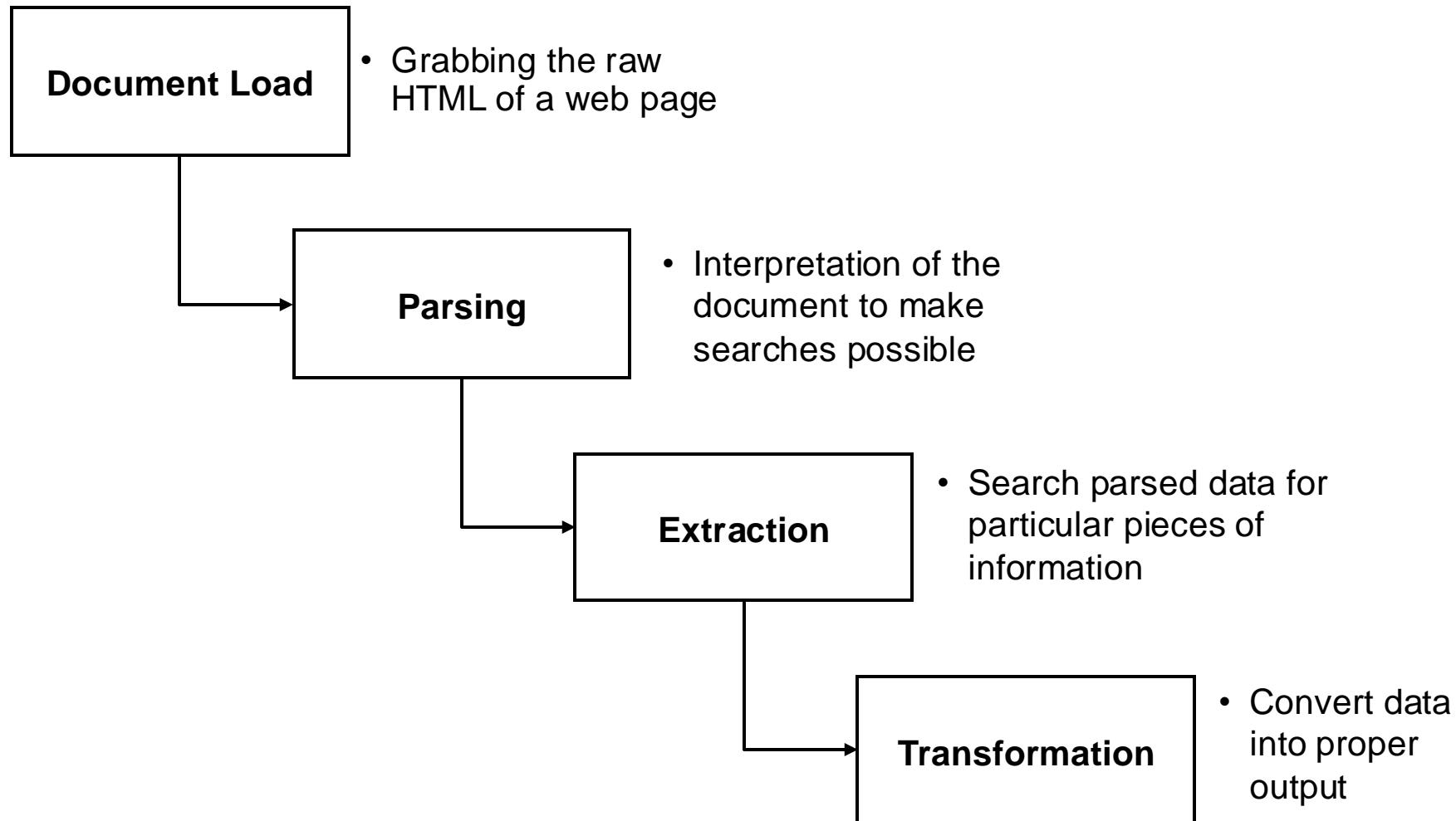
Disallow: /oauth
Disallow: /1/oauth

Disallow: /istreams
Disallow: /i/hello

Wait 1 second between successive requests. See ONBOARD-2698 for details.
Crawl-delay: 1
```

<https://twitter.com/robots.txt> (last retrieved on December 05, 2018)

# Web scraping means reducing HTML code to the information you need



We will figure out identifiers for specific parts in the HTML code

### From raw HTML code to specific information

- Web scraping works by filtering the HTML code of a web page to just the bits you want
- This can be done using identifiers for specific parts in the HTML code
- We will start with simple examples where we can figure out the identifiers on our own
- Later, we will use a so-called *selector* – a browser plug-in that helps us figuring out the identifier on more complex web pages

# Before we can go on, we need to understand the structure of XML files

- The structure of an XML file can be divided into **markup** (i.e., the structure of the data) **and content** (i.e., the data itself)
- Markup typically comes in the form of tags, with **opening and closing tags** (`<tagname>...</tagname>`), content is in between tag pairs
- A **tag can also contain attributes** in the form of name-value pairs (see `lang='en'`)
- There is no standard way of storing data, but usually **attributes are reserved for meta-data**



```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
 <movie>
 <title lang='en'>A New Hope</title>
 <year>1977</year>
 </movie>
 <movie>
 <title lang='en'>The Empire Strikes Back</title>
 <year>1980</year>
 </movie>
</movies>
```

# XML documents are trees of nodes

**A**

```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
 <movie>
 <title>A New Hope</title>
 <year>1977</year>
 </movie>
 <movie>
 <title>The Empire Strikes Back</title>
 <year>1980</year>
 </movie>
</movies>
```

Content  
(everything between two tags)

Pair of tags

Element  
(everything from and incl. a start tag to and incl. the matching end tag)

**B**

```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
 <movie>
 <title year = "1977">A New Hope</title>
 </movie>
 <movie>
 <title>The Empire Strikes Back</title>
 <year>1980</year>
 </movie>
</movies>
```

Attribute (name-value pair)

We can think of the XML file's structure like a family tree

```
<?xml version="1.0" encoding="UTF-8"?>
<movies> ← movie is a child of movies
 <movie> = movie is a parent of title
 <title>A New Hope</title> ←
 <year>1977</year>
 </movie>
 <movie>
 <title>The Empire Strikes Back</title> ←
 <year>1980</year> ←
 </movie>
</movies>
```

year and title are siblings

## XPATHs identify so-called *nodes* or *node-sets* of an XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<title>Star Wars</title>
<movies>
 <movie>
 <title lang='en'>A New Hope</title>
 <year>1977</year>
 </movie>
 <movie>
 <title lang='en'>The Empire Strikes Back</title>
 <year>1980</year>
 </movie>
</movies>
```

The XML code shows two movie entries. The first movie's title is highlighted with a yellow box, and its XPATH is indicated as /movies/movie/title. The second movie's title is also highlighted with a yellow box.

### Specifying paths

- With a single forward slash “/”, you can define paths to nodes as you would in specifying file paths

## XPATHs identify so-called *nodes* or *node-sets* of an XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<title>Star Wars</title>
<movies>
 <movie>
 <title lang='en'>A New Hope</title>
 <year>1977</year>
 </movie>
 <movie>
 <title lang='en'>The Empire Strikes Back</title>
 <year>1980</year>
 </movie>
</movies>
```

The diagram illustrates the XPATH expression //title. A box labeled "XPATH: //title" has arrows pointing to each title element in the XML document. The first title element, "Star Wars", is highlighted with a yellow box. The second title element, "A New Hope", is also highlighted with a yellow box. The third title element, "The Empire Strikes Back", is also highlighted with a yellow box.

### Nodes on multiple levels

- A double forward slash “//” selects all nodes that match the selection, no matter where they are in the document

## XPATHs identify so-called *nodes* or *node-sets* of an XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<title>Star Wars</title>
<movies>
 <movie>
 <title lang='en'>A New Hope</title>
 <year>1977</year>
 </movie>
 <movie>
 <title lang='en'>The Empire Strikes Back</title>
 <year>1980</year>
 </movie>
</movies>
```

XPATH: //@lang

### Selecting attributes

- An “@” sign is used to refer to attributes in an XML document

## XPATHs identify so-called *nodes* or *node-sets* of an XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<title>Star Wars</title>
<movies>
 <movie>
 <title lang='en'>A New Hope</title>
 <year>1977</year>
 </movie>
 <movie>
 <title lang='en'>The Empire Strikes Back</title>
 <year>1980</year>
 </movie>
</movies>
```

XPATH: movies/movie/\*

### Using wildcards

- A “\*” is used as a wildcard to match any element node

# A selection of useful path expressions

```

<a>
 1
 <c lang='en'>Hello world!</c>

<a>
 6

```

	<b>Path expression</b>	<b>Description</b>
<b>Selecting nodes</b>	/	Selects from the root node
	//	Selects nodes in the document from the current node that match the selection no matter where they are
	@	Selects attributes
<b>Specific nodes</b>	/a/b[1]	Selects the first b element that is the child of the a element.
	/a[b>5]	Selects all the a elements that have a b element with a value greater than 5
	/a[b>5 and b<7]	Selects all the a elements that have a b element with a value between 5 and 7
<b>Selecting unknown nodes</b>	*	Matches any element node
	@*	Matches any attribute node
	//*[1]	Selects all elements in the document
<b>Selecting several paths</b>	//a/b   //a/c	Selects all the b AND c elements of all a elements
	//b   //c	Selects all the b AND c elements in the document
	/a/c   //b	Selects all the c elements of the a element AND all the b elements in the document

More expressions and examples can be found on [https://www.w3schools.com/xml/xpath\\_syntax.asp](https://www.w3schools.com/xml/xpath_syntax.asp)

# Practical Scraping: Reading a HTML page

## Tools for web scraping

- The packages `{rvest}` in R and `{lxml}` in Python provide all web scraping utilities we need, most importantly extracting pieces of information
  - `{rvest}` also handles reading in the HTML
  - In Python we will need the `{requests}` package for imports
- Both packages store the XML hierarchy of a HTML page in a nested tree structure



```
library(rvest)

import and store hierarchy
tree <- read_html(URL)
```



```
import requests
from lxml import etree

import
page = requests.get(URL).content

store hierarchy
tree = etree.HTML(page)
```



# Practical Scraping: Reading a HTML page

## A note on Beautiful Soup

- Beautiful Soup (`{BS4}`) is an alternative way to parse a HTML page in Python. It provides a unified frontend to several XML parsers like `{lxml}`
- The package is frequently used because of its simplicity
- We do not show `{BS4}` here because it does not allow for XPATH queries
- Using the knowledge on XML you gain in this chapter you will however be able to grasp the syntax of `{BS4}` quickly if the need arises
- Feel free to find out more about the differences between the two by investigating the [documentation](#) of `{BS4}`

# Now, what to do with the identified XPATHs?

- Our goal is to extract all main headings of the Wikipedia article about Tübingen:

**Tübingen**

From Wikipedia, the free encyclopedia

**Tübingen** (German: [ty̯.bɪŋən] ⓘ; Swabian: *Dibenga*) is a traditional university city in central Baden-Württemberg, Germany. It is situated 30 km (19 mi) south of the state capital, Stuttgart, and developed on both sides of the Neckar and Ammer rivers. As of 2014<sup>[3]</sup> about one in three of the 90,000 people<sup>[citation needed]</sup> living in Tübingen is a student. As of the 2018/2019 winter semester, 27,665 students attend the Eberhard Karls University of Tübingen.<sup>[citation needed]</sup> The city has the lowest median age in Germany, in part due to its status as a university city. As of December 31, 2015, the average age of a citizen of Tübingen is 39.1 years.<sup>[citation needed]</sup>

Immediately north of the city lies the Schönbuch, a densely wooded nature park. The Swabian Alb mountains rise about 13 km (8 mi) (beeline Tübingen City to Roßberg - 869 m) to the southeast of Tübingen.

The Ammer and Steinach rivers are tributaries of the Neckar river, which flows in an easterly direction through the city, just south of the medieval old town. Large parts of the city are hilly, with the Schlossberg and the Österberg in the city centre and the Schnarrenberg and Herrlesberg, among others, rising immediately adjacent to the inner city.

The highest point is at about 500 m (1,640.42 ft) above sea level near Bebenhausen

Contents hide

- (Top)
- History
- Overview
- Main sights
- Government
  - Regional structure
  - Districts
- Culture
  - Events
- Population
  - Population development
  - Historical population
- Climate
- Twin towns – sister cities
- Infrastructure
- Sport
- Education

Read Edit View history Tools

Coordinates: 48°31'12"N 09°03'20"E

**Tübingen**  
Dibenga (Swabian)  
Town



Tübingen seen from above in June 2018



Coat of arms

Location of Tübingen within Tübingen [show]  
district

```
library(rvest)
URL <- "https://en.wikipedia.org/wiki/T%C3%BCbingen"

import and store hierarchy
page_tuewiki <- read_html(URL)
```



```
import requests
from lxml import etree

import
URL = "https://en.wikipedia.org/wiki/T%C3%BCbingen"
page = requests.get(URL).content

store hierarchy
page_tuewiki = etree.HTML(page)
```

<https://en.wikipedia.org/wiki/T%C3%BCbingen> (last retrieved on August 05, 2024)

# We first need to detect the desired information in the XML file

- If the XML file is rather complex, it may help to store the XML file and inspect it with a code editor (e.g., Notepad++, RStudio, ...)



```
xml2::write_xml(page_tuewiki, 'pretty.html')
```



```
with open('pretty.html', 'wb') as file:
 file.write(etree.tostring(page_tuewiki, pretty_print = True))
```

```

110 <h2>Regional structure<span class="mw-editsection-
111 <p>Tübingen is the capital of an eponymous district a
112 </p><p>Tübingen is, with nearby Reutlingen (about 15 km (9.3 mi) east),
113 </p><p>Administratively, it is not part of the Stuttgart Region, bordering
114 </p>
115 <h2>History[<a href=
116 <style data-mw-deduplicate="TemplateStyles:r1033289096">.mw-parser-output .hatnote{font-style:italic}.mw-parser-output div.hatnote{padding-left:1em} .mw-
117 <p>The area was probably first settled by ancient humans in the 12th millennium BC. The Roman Empire first appears in official records in 1191. The local castle, <i>Hohentübingen</i>, has records going back to 1146, Count Hugo V (1125–52) was promoted to count palatine
118 </p><p>From 1262, an Augustinian monastery was established by Pope Alexander VII in 1342, the county palatine was sold to Ulrich III, Count of Württemberg
119 </p><p>In 1262, an Augustinian monastery was established by Pope Alexander VII in 1342, the county palatine was sold to Ulrich III, Count of Württemberg
120 </p><p>In 1342, the county palatine was sold to Ulrich III, Count of Württemberg
121 </p><p>In 1342, the county palatine was sold to Ulrich III, Count of Württemberg
122 </p>
```

- Some detective work browsing the XML file reveals that the target information is stored between the *first* pair of `<span>` tags which are children of elements called `h2`
- Translated into an XPATH, this yields the following locator: `"//h2/span[1]"`

# Once the XPATH is identified we extract the information

- Having identified the correct XPATH, we can now extract a node-set:

```
headings_nodes <- html_elements(page_tuewiki, xpath = "//h2")
```

```
headings_nodes
```

```
[1] Regional
structure
```

```
...
```

```
[16] External links
```

```
headings_nodes = page_tuewiki.xpath("//h2/span")
```

```
convert the tree to string for printing
```

```
[etree.tostring(node) for node in headings_nodes]
```

```
b'Regional
structure,
```

```
...
```

```
b'External links']
```

# Once the XPATH is identified we extract the information

- Then we extract the text from the node-set:

```
headings_text <- html_text(headings_nodes)
```



```
headings_text
[1] "Regional structure" "History" "Overview"
...
[16] "External links"
```

```
headings_text = [node.text for node in headings_nodes]
```



```
headings_text
['Regional structure', 'History', 'Overview',
...
'External links']
```

## Exercises



1. Import the file *album\_catalog.xml* (contained in *data.zip*) into R and Python.  
(R: `xml2::read_xml()`, Python: `etree.parse()`)

Inspecting the HTML in a code editor of your choice, identify the XPATHs and extract the node sets locating the following information:

- The names of all artists
- The names of artists from the UK
- All information on CDs that were released in 1990 or later
- The hyperlinks to all albums
- The titles of CDs by British artists that were released in 1990 or later

Web scraping should be the last resort in case no API is available

### Web scraping vs. APIs

- Changes in the website layout can cause problems to your code
- Your XPATH locators are not robust to changes and may need to be adapted after a website structure has been updated
- APIs are way more robust: You can access the data irrespective of changes in the website layout.

# Admittedly, there are more complex web pages than Wikipedia

- An XML file can easily comprise ten thousands of lines of code
- Identifying a specific bit of information and locating to it will be practically unfeasible for us
- Thankfully, there are browser plug-ins available which make life a lot easier for us
- The [SelectorGadget](#), for example, allows for point-and-click selection of visual elements and generates the XPATH for us
- To use the tool, you need to do the following:

## Using the SelectorGadget

---

- Drag the code of the tool (see [here](#)) to the bookmark bar in your web browser
- Open the website that you want to scrape data from
- Click on the SelectorGadget bookmark
- Then click on the elements that you (do not) want to extract
- Copy the XPATH generated by the SelectorGadget
- Proceed as presented earlier

<https://selectorgadget.com/> (last retrieved on December 06, 2018)

# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 1 | Identify a website that you want to scrape

e.g.: <https://www.fussballdaten.de/bundesliga/2021/tore/>

The screenshot shows the homepage of fussballdaten.de. The navigation bar includes links for Bundesliga, 2020/2021, Spieltag, Tabellen, Statistik (which is highlighted), Relegation, Spielplan, Aktuelles, and Historie. Below the navigation is a search bar with a magnifying glass icon. A green banner at the top says "BUNDESLIGA - TORE". There are dropdown menus for "Gesamt" and "2020/2021". Below these are several filter buttons: ALLE, ZUSAMMENFASSUNG, TORE (highlighted in green), VORLAGEN, EINSÄTZE, KARTEN, ELF METER, TORHÜTER, ZUSCHAUER, ERFAHRUNG, and ALTER. A section titled "TORSCHÜTZENLISTE" displays a table of top scorers. The table has columns: #, SPIELER, EINSATZZEIT, MIN/TOR, ELFER, and TORE. The data is as follows:

#	SPIELER	EINSATZZEIT	MIN/TOR	ELFER	TORE
1.	Robert Lewandowski FC Bayern München	2.462	60	9	41
2.	Andre Silva Eintracht Frankfurt	2.771	99	7	28

<https://www.fussballdaten.de/bundesliga/2021/tore/> (last retrieved on September 23, 2021)

# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 2 | Check if the website provider allows you to scrape data

<https://www.fussballdaten.de/robots.txt>

```
User-agent: Slurp
Crawl-delay: 2

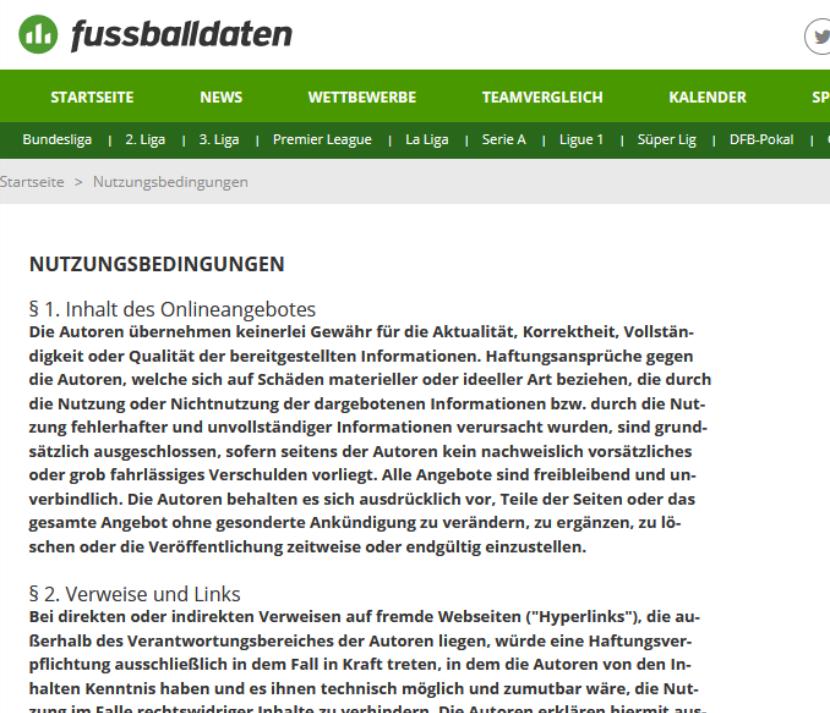
User-agent: wget
Disallow: /

Sitemap: https://www.fussballdaten.de/sitemap.xml
Sitemap: https://www.fussballdaten.de/sitemap-1.xml
Sitemap: https://www.fussballdaten.de/sitemap-2.xml
Sitemap: https://www.fussballdaten.de/sitemap-3.xml

User-agent: *
Disallow: /*json/*
Disallow: /wettbewerb/tabelle-startseite/
Disallow: /news-monat-ajax/

Allow: /
```

<https://www.fussballdaten.de/nutzungsbedingungen/>

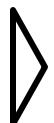


The screenshot shows the fussballdaten.de website's terms of service page. The header features the logo and navigation links for Startseite, News, Wettbewerbe, Teamvergleich, Kalender, and Sp. Below the header, a breadcrumb trail shows 'Startseite > Nutzungsbedingungen'. The main content area is titled 'NUTZUNGSBEDINGUNGEN' and contains two sections: § 1. Inhalt des Onlineangebotes and § 2. Verweise und Links. The first section discusses liability for the accuracy and completeness of the information provided. The second section covers links to external websites.

**NUTZUNGSBEDINGUNGEN**

§ 1. Inhalt des Onlineangebotes  
Die Autoren übernehmen keinerlei Gewähr für die Aktualität, Korrektheit, Vollständigkeit oder Qualität der bereitgestellten Informationen. Haftungsansprüche gegen die Autoren, welche sich auf Schäden materieller oder ideeller Art beziehen, die durch die Nutzung oder Nichtnutzung der dargebotenen Informationen bzw. durch die Nutzung fehlerhafter und unvollständiger Informationen verursacht wurden, sind grundsätzlich ausgeschlossen, sofern seitens der Autoren kein nachweislich vorsätzliches oder grob fahrlässiges Verschulden vorliegt. Alle Angebote sind freibleibend und unverbindlich. Die Autoren behalten es sich ausdrücklich vor, Teile der Seiten oder das gesamte Angebot ohne gesonderte Ankündigung zu verändern, zu ergänzen, zu löschen oder die Veröffentlichung zeitweise oder endgültig einzustellen.

§ 2. Verweise und Links  
Bei direkten oder indirekten Verweisen auf fremde Webseiten ("Hyperlinks"), die außerhalb des Verantwortungsbereiches der Autoren liegen, würde eine Haftungsverpflichtung ausschließlich in dem Fall in Kraft treten, in dem die Autoren von den Inhalten Kenntnis haben und es ihnen technisch möglich und zumutbar wäre, die Nutzung im Falle rechtswidriger Inhalte zu verhindern. Die Autoren erklären hiermit aus-



In this case, the website owner seems to be fine with us scraping the data. As there is no crawl-delay given explicitly for our scraper, we choose a conservative rate of 0.5 requests per second.

# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 2 | Check if the website provider allows you to scrape data



The screenshot shows the idealo homepage with a CAPTCHA challenge. The page features the idealo logo at the top, followed by a navigation bar with 'Kategorien', a search bar, and user icons for 'Merkzettel' and 'Mein idealo'. Below the search bar, a message reads 'Ups - Das kommt uns seltsam vor!' accompanied by a blue owl icon. A text box states: 'Jetzt hast Du so schnell geklickt, dass wir nicht mehr sicher sind, ob Du wirklich Du oder ein Roboter bist. Zum Weitermachen setze bitte ein Häkchen, denn das können Roboter nicht.' Below this is a 'Vielen Dank!' message. A reCAPTCHA form is displayed with the text 'Ich bin kein Roboter.' and a checkbox. The reCAPTCHA logo and links for 'Datenschutzerklärung' and 'Nutzungsbedingungen' are visible. An orange 'weiter' button is at the bottom of the form. At the bottom of the main content area, there's a section titled 'Hinweise auf geistige Eigentumsrechte' with a note about respecting intellectual property rights, including a statement against scraping.

**Hinweise auf geistige Eigentumsrechte**

Die Webseiten enthalten Daten, Suchergebnisse, Texte, Grafiken, Software und sonstige Informationen, die zu Gunsten von idealo und ihrer Mitarbeiter mittels geistiger Eigentumsrechte, insbesondere nach dem Urheber- und Markenrecht, geschützt sind. Vervielfältigungen, öffentliche Wiedergaben, Modifikationen und sonstige Eingriffe in diese Rechte sind sowohl ganz als auch teilweise nur mit schriftlicher Genehmigung idealo erlaubt, es sei denn, dass eine gesetzliche Schranke eingreift. Jeder nicht genehmigte Eingriff stellt einen Verstoß gegen die genannten rechtlichen Bestimmungen dar und wird rechtlich verfolgt. Insbesondere sind das automatisierte Auslesen der Datenbank und Webseiten (durch z. B. Scraping) und das Einbinden von Inhalten der Webseiten im Frame ohne Einwilligung von idealo nicht gestattet. Der Nutzer darf die Anwendung nur zum individuellen Vergleich von Angeboten von Online-Shops für den eigenen Bedarf nutzen.



You can easily get blocked by a website provider if you violate the Terms of Service!

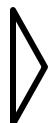
# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 2 | Check if the website provider allows you to scrape data

An extract from the robots.txt file of the online dictionary Linguee:

```
In ANY CASE, you are NOT ALLOWED to train Machine Translation Systems
on data crawled on Linguee.
#
Linguee contains fake entries - changes in the wording of sentences,
complete fake entries.
These entries can be used to identify even small parts of our material
if you try to copy it without our permission.
Machine Translation systems trained on these data will learn these errors
and can be identified easily. We will take all legal measures against anyone
training Machine Translation systems on data crawled from this website.
[...]
```

<https://www.linguee.de/robots.txt> (last retrieved on December 10, 2018)



Some websites take pre-emptive measures to deter undesired scraping activities.

# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 3 | Identify the URL structure of related web pages

- Scorers of season 2020/21 can be found on:  
<https://www.fussballdaten.de/bundesliga/2021/tore/>
- Scorers of season 2019/20 can be found on:  
<https://www.fussballdaten.de/bundesliga/2020/tore/>
- Scorers of season 2018/19 can be found on:  
<https://www.fussballdaten.de/bundesliga/2019/tore/>
- ...
- Data are available since the season of 1963/64
- Hence, a vector of all 58 URLs can be created by calling:



```
url_root <- "https://www.fussballdaten.de/bundesliga/"
urls <- paste0(url_root, 1964:2021, "/tore/")
```



```
url_root = "https://www.fussballdaten.de/bundesliga/"
urls = [url_root + str(i) + "/tore/" for i in range(1964, 2021 + 1)]
```



# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 4 | Use the SelectorGadget to identify the location of the desired data

The screenshot shows the 'BUNDESLIGA - TORE' section of the fussballdaten.de website. The 'Statistik' tab is active. The 'TORSCHÜTZENLISTE' table displays the top scorers for the 2020/2021 season. The first three rows are highlighted with green and yellow boxes. A red box highlights the CSS selector `#grid-spieler .table-link.hidden-xs` applied to the table row for Erling Haaland. Another red box highlights the XPath `//*[@id='grid-spieler']/tr[3]/td[1]/a` applied to the same row. The SelectorGadget interface at the bottom right shows the selected path and various options like 'Clear (15)', 'Toggle Position', 'XPath', 'Help', and 'X'.

#	SPIELER	EINSATZ	MIN/TOR	ELFER	TORE
1.	Robert Lewandowski FC Bayern München	2.462	60	9	41
2.	Andre Silva Eintracht Frankfurt	2.771	99	7	28
3.	Erling Haaland Borussia Dortmund	2.410	89	4	27
	Wout Weghorst VfL Wolfsburg				

**CSS selector** #grid-spieler .table-link.hidden-xs

**XPATH** `//*[@id='grid-spieler']/tr[3]/td[1]/a`

A CSS selector is an alternative, oftentimes much shorter representation of node locations in web page code.

# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 5 | Read the HTML into R, parse it to XML, and extract the information

We start with only one season (2020/21):



We retrieve and import the page

```
page_topscorer <-
 read_html(tail(urls, n = 1))
```

```
page = requests.get(urls[-1]).content
page_topscorer = etree.HTML(page)
```

We query for the XPATH

```
XPATH <- '//*[@id = "grid-spieler"]//*[contains(concat(" ", @class, " "),
 concat(" ", "table-link", " ")) and contains(concat(" ", @class, " "),
 concat(" ", "hidden-xs", " "))]'
```

```
topscorer_nodes <-
 html_elements(page_topscorer,
 xpath = XPATH)
```

```
topscorer_nodes =
 page_topscorer.xpath(XPATH)
```

Alternatively, we can query for the CSS-Selector

```
CSS = "#grid-spieler .table-link.hidden-xs"
```

```
topscorer_nodes <-
 html_elements(page_topscorer, css = CSS)
```

Requires the package  
{cssselect} to be installed

```
topscorer_nodes =
 page_topscorer.cssselect(CSS)
```

# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 5 | Read the HTML into R, parse it to XML, and extract the information

We start with only one season (2020/21):



Finally, we extract the players' names

```
topscorer_names <-
 html_text(topscorer_nodes)
```

```
topscorer_names
[1] "Robert Lewandowski"
...
[15] "Ihlas Bebou"
```

```
topscorer_names =
 [node.text for node in topscorer_nodes]
```

```
topscorer_names
['Robert Lewandowski',
 ...
 'Ihlas Bebou']
```

# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 5 | Read the HTML into R, parse it to XML, and extract the information

```
library(magrittr)

start <- 1964
end <- 2021
num_players <- 15 # the number of players in each year

topScorers <- rep_len(NA, length(start:end)) %>% as.list()

for(year in start:end){
 url <- paste0("https://www.fussballdaten.de/bundesliga/", year, "/tore/")
 page_topscorer <- read_html(url)
 topscorer_nodes <- html_elements(page_topscorer,
 css = "#grid-spieler .table-link.hidden-xs")
 topScorers[[year - 1963]] <-
 list(season = paste0(year - 1, "/", year),
 rank = 1:num_players,
 name = html_text(topscorer_nodes))

 Sys.sleep(2) # this pauses the system for 2 seconds before the next request is made
}

data.table::rbindlist(topScorers)
```

	season	rank	name
1	1963/1964	1	Uwe Seeler
2	1963/1964	2	Friedhelm Konietzka
3	1963/1964	3	Rudolf Brunnenmeier
4	1963/1964	4	Wilhelm Huberts
5	1963/1964	5	Klaus Matischak
[ omitted 1095 rows ]			



# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

## 5 | Read the HTML into R, parse it to XML, and extract the information

```
import time
import pandas as pd

start = 1964
end = 2021
num_players = 15 # the number of players in each year

topScorers = [None] * len(range(start, end + 1))

for year in range(start, end + 1):
 url = "https://www.fussballdaten.de/bundesliga/" + str(year) + "/tore/"
 page = requests.get(url).content
 page_topscorer = etree.HTML(page)
 topscorer_nodes = page_topscorer.cssselect("#grid-spieler .table-link.hidden-xs")

 topScorers[year - start] = pd.DataFrame({ "season": str(year - 1) + "/" + str(year),
 "rank": [(x + 1) for x in range(num_players)],
 "name": [node.text for node in topscorer_nodes]})

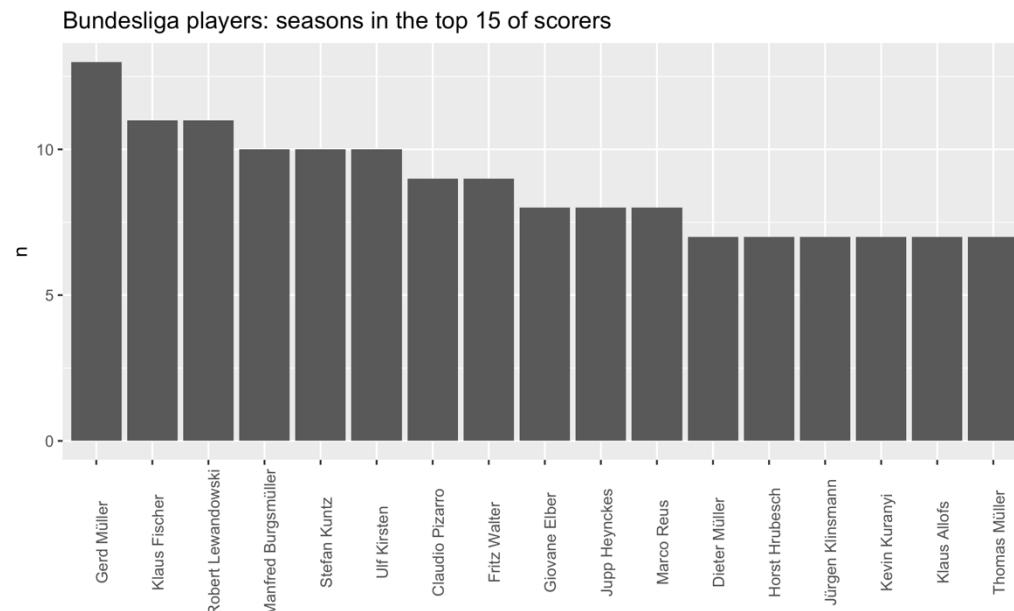
 time.sleep(2)

pd.concat(topScorers)
```

	season	rank	name
1	1963/1964	1	Uwe Seeler
2	1963/1964	2	Friedhelm Konietzka
3	1963/1964	3	Rudolf Brunnenmeier
4	1963/1964	4	Wilhelm Huberts
5	1963/1964	5	Klaus Matischak
[ omitted 1095 rows ]			

## Web scraping in practice: Bundesliga top scorers from fussballdaten.de

```
topScorers <- rbindlist(topScorers, fill = TRUE) #convert to dataframe in order to use count
topScorers %>%
 count(name) %>%
 filter(n > 6) %>%
 ggplot(aes(x = reorder(name, n, decreasing = T), y = n)) +
 geom_bar(stat = "identity") +
 labs(title = "Bundesliga players: seasons in the top 15 of scorers",
 x = "") +
 theme(axis.text.x = element_text(angle = 90))
```

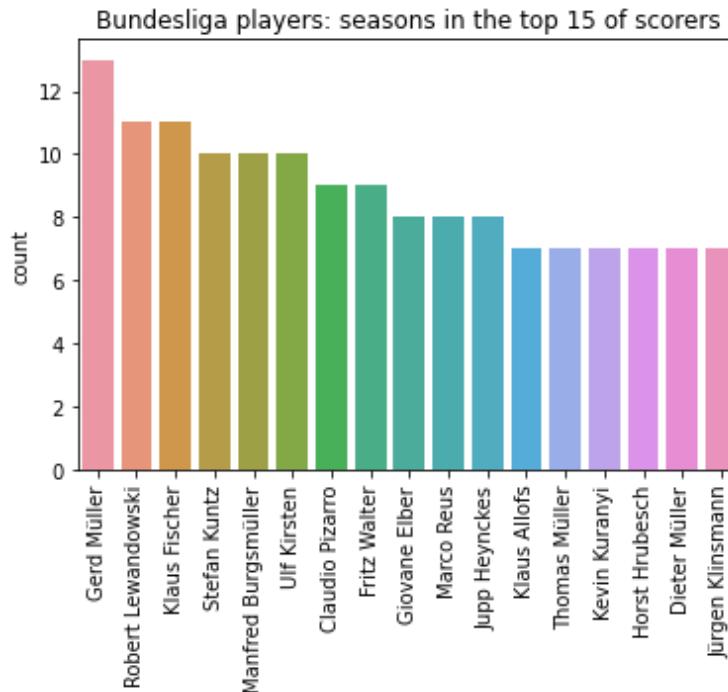


# Web scraping in practice: Bundesliga top scorers from fussballdaten.de

```
import seaborn as sns

ax = sns.countplot(x = "name", data = topScorers,
 order = topScorers["name"].value_counts().\
 where(lambda x: x > 6).dropna().index)

ax.set(title = "Bundesliga players: seasons in the top 15 of scorers",
 xlabel = "")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```



## Exercises



2. Inspect the following web page:

[https://en.wikipedia.org/wiki/List\\_of\\_Christmas\\_films](https://en.wikipedia.org/wiki/List_of_Christmas_films)

- Import the website's HTML code
- Without using the SelectorGadget, identify one XPATH that locates all tables in the XML document and extract the table nodes. This is best done by exporting the XML and looking at it in a text editor. (Hint: In Python use `etree.ElementTree(tree).write(path)` for the export)
- Turn the identified HTML tables into data frames. The result should be a list of all tables that are shown on the web page.

In R, look up the documentation on the function `rvest::html_table()` and apply it to the table nodes extracted in the previous step.

In Python this process is more complicated. First convert each element to a string using `etree.tostring(element)`. Then parse these strings into Pandas using `pd.read_html(element_string)`

## Exercises



- d) Concatenate all list elements that contain movie tables from the different categories (Hint: In R, `rlist::list.stack()` is a way of doing this. The option `idcol = 'colname'` adds a column indicating from which list element an observation comes from.)

This is what the result of 2d) might look like (note that website contents are dynamic and hence the results may vary over time):

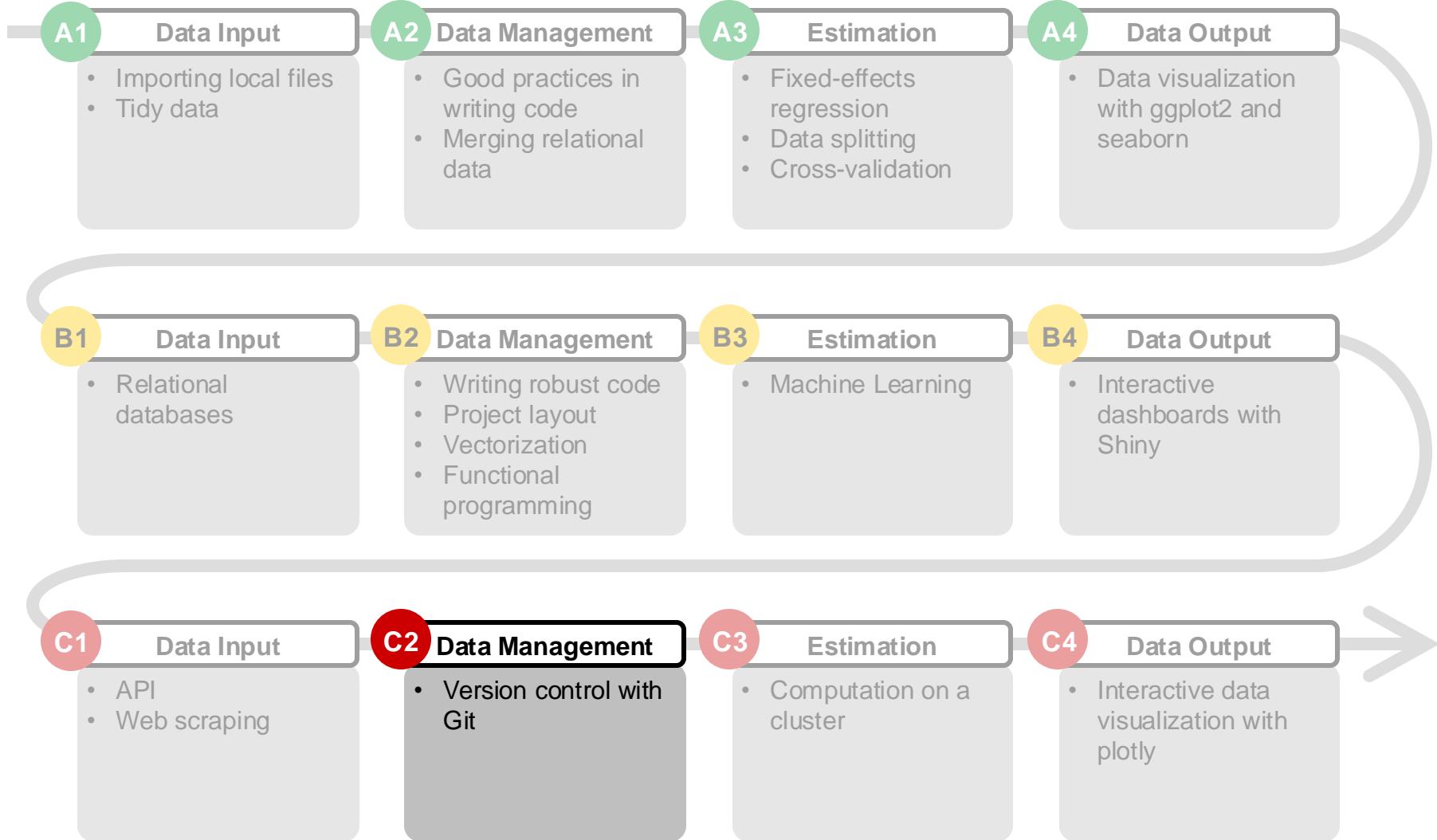
▲	Category	Title	Year	Description
1	1	The Unholy Three	1930	A trio of thieves plan a jewel heist on Christmas Eve.
2	1	A Bill of Divorcement	1932	A man returns home on Christmas Eve after spending fifteen ...
3	1	The Thin Man	1934	Nick and Nora Charles solve a mystery during Christmas.
4	1	Babes in Toyland	1934	Metro-Goldwyn-Mayer's adaption of Victor Herbert's operett...
5	1	Three Godfathers	1936	At Christmastime in the Old West, three outlaws find a woma...
6	1	Love Finds Andy Hardy	1938	Andy Hardy finds himself romantically entangled with three di...
7	1	Bachelor Mother	1939	A department store salesgirl is roped into caring for an orpha...

## Exercises

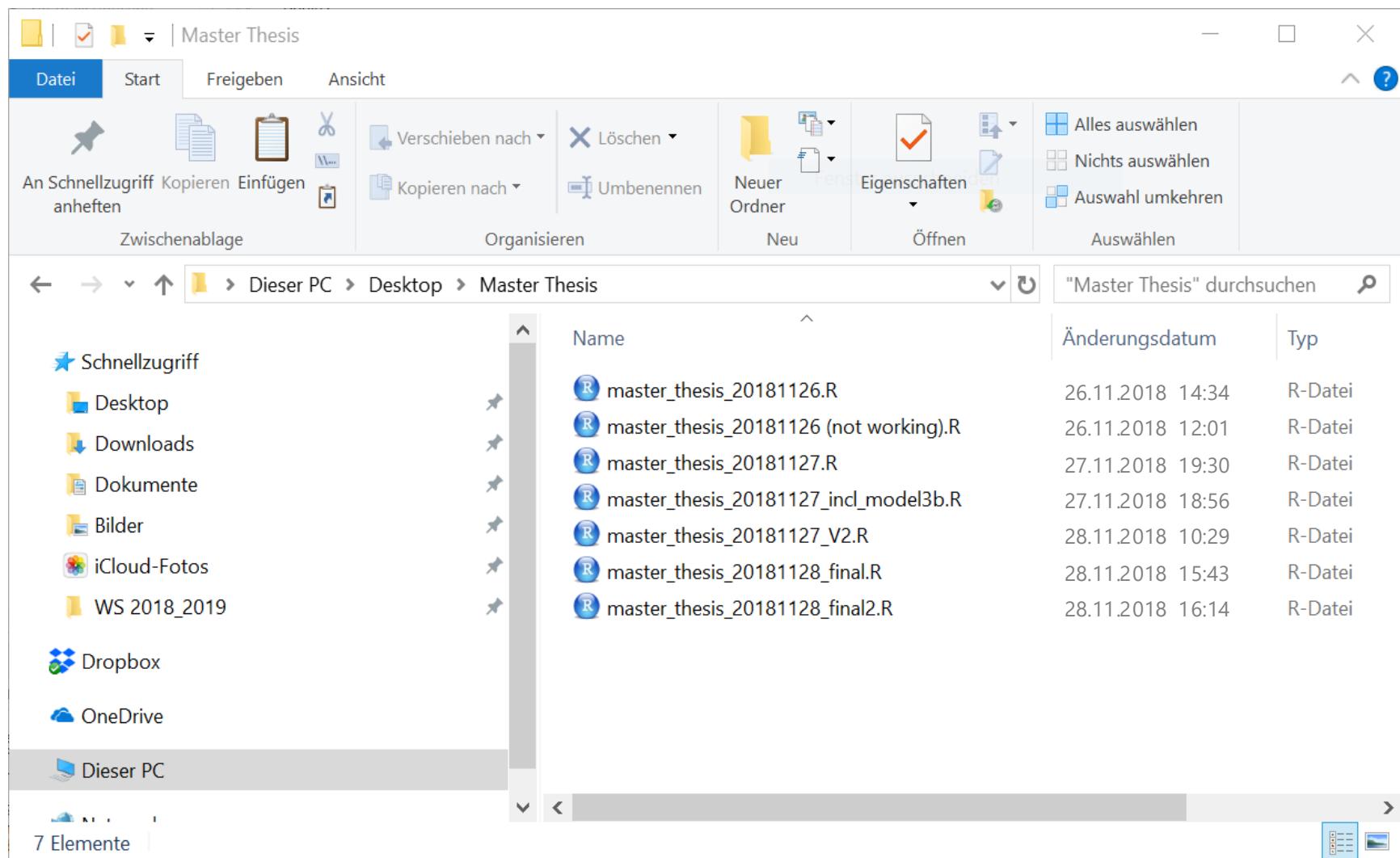


1. On <https://www.konzerte-tuebingen.de/konzerte/wintersaison-2024-2025>, you can find a list of the concerts that take place in the Neue Aula in Tübingen during the winter season 2024/25. Use the SelectorGadget to extract the date, the artist, and the title of the concerts.

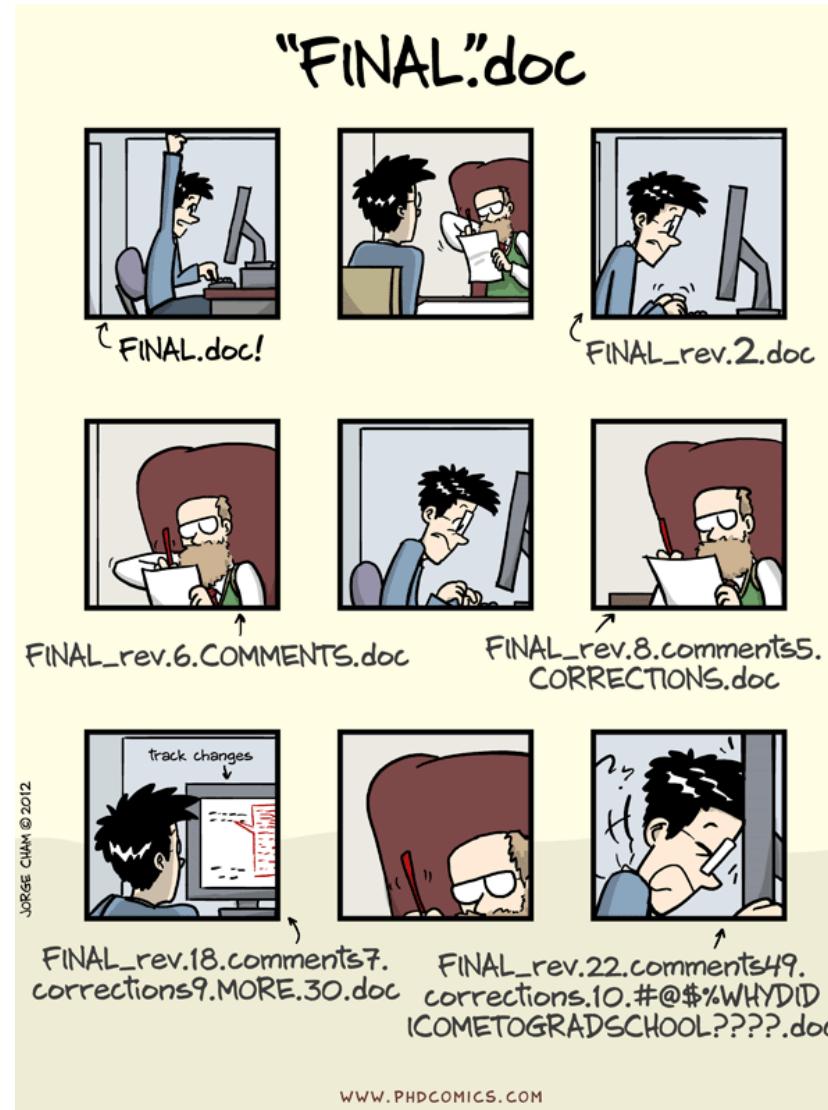
# Data Science Project Management: Course Outline



# We all know the problem...

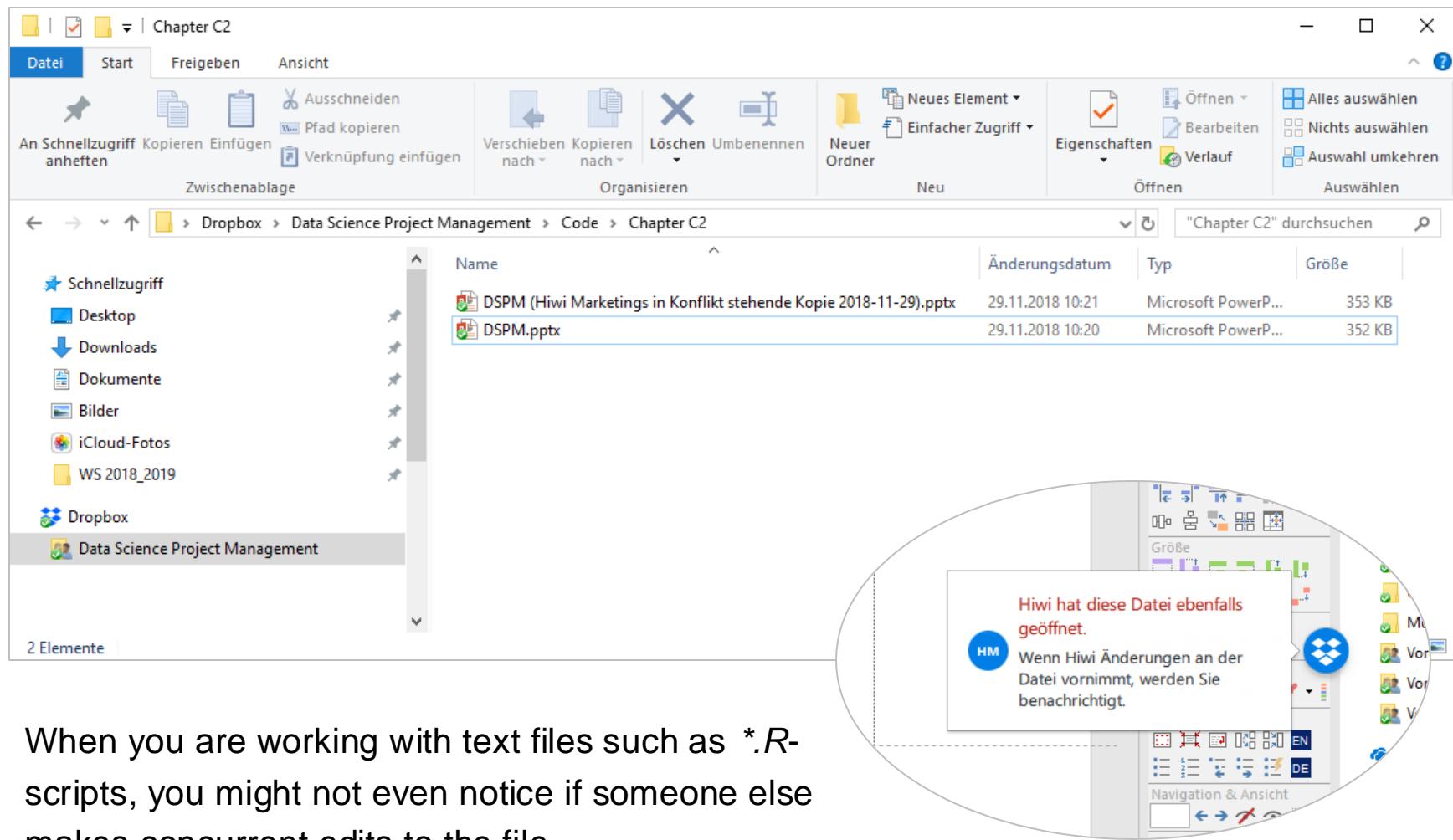


Grrrrr....!!!!1!11!!



<http://phdcomics.com/comics/archive/phd101212s.gif> (last retrieved on September 08, 2020)

# Collaboration can be very nasty without a proper tool



When you are working with text files such as `*.R`-scripts, you might not even notice if someone else makes concurrent edits to the file.

# What is version control?

## Version Control System (VCS)

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows to:
  - Revert selected files back to a previous state
  - Revert an entire project to a previous state
  - Compare changes over time
  - See who last modified something that might be causing a problem

- We can classify VCS into three types:
  - Local VCS
  - Centralized VCS (CVCS)
  - Distributed VCS (DVCS)



I'm confused. Can we talk straight, please?

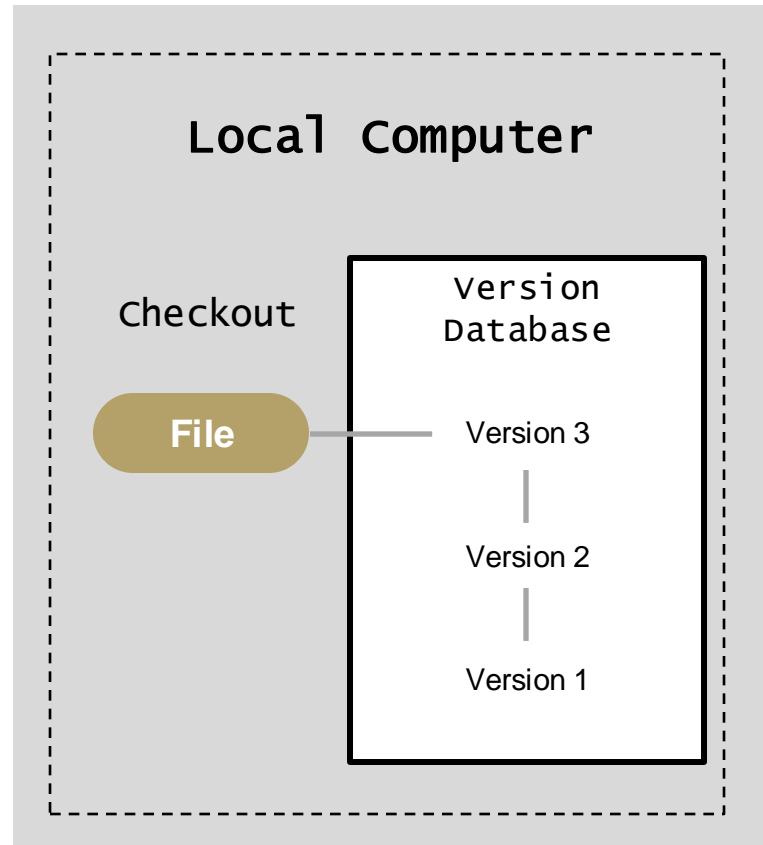
### Why Git?

- Git is a distributed version control system
- Imagine if Dropbox and the "Track changes" feature in MS Word had a baby. Git would be that baby.
- Git's (and GitHub's) role in global software development is not in question. There's a high probability that your favorite app, program or package is built using Git-based tools. (Rstudio, Spyder, VSCode, Jupyter Notebook are all cases in point.)

# A local VCS helps with versioning, but not with collaborating

## Local Version Control System

- Uses intuitive principle of storing different versions of files with a time stamp (e.g., *master\_thesis\_20181020.R*, *master\_thesis\_20181021.R*, ...)
- Local VCS improve this approach by storing only incremental changes to previous versions (= patch-set)
- By adding up patch-sets, any version of the file can be recreated
- But: Problem of collaboration persists



# Centralized VCS allow for versioning and collaboration

## Centralized Version Control System

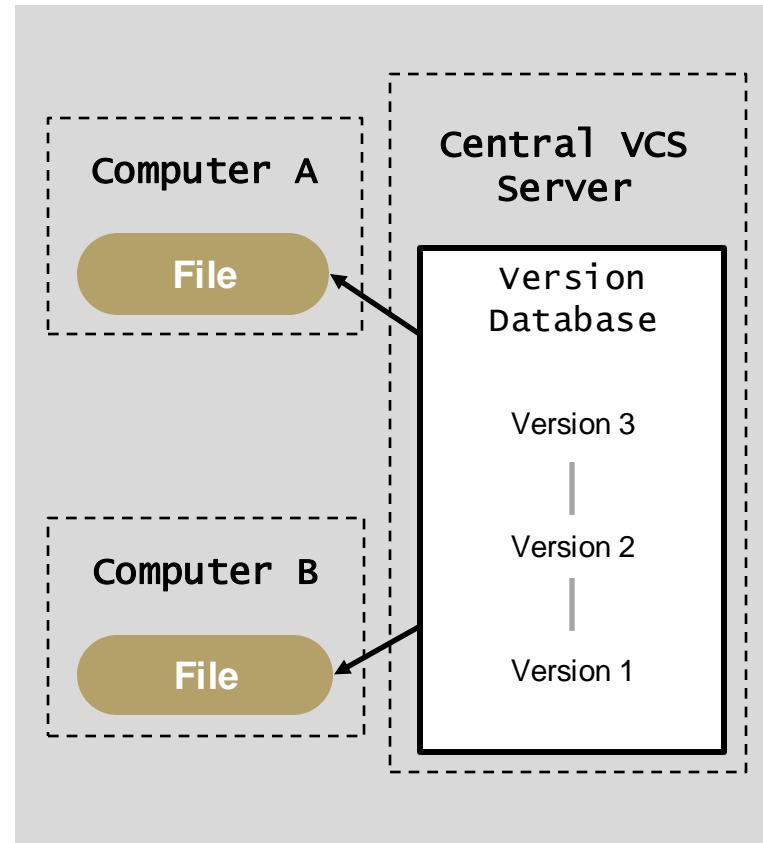
- Uses a single server that contains all the versioned files, and a number of clients that check out files from that central place.

### Advantages:

- Everyone knows to a certain degree what everyone else on the project is doing
- Administrators have fine-grained control over who can do what

### Disadvantage:

- Whenever the entire history of a project is stored in a single place, you risk losing everything



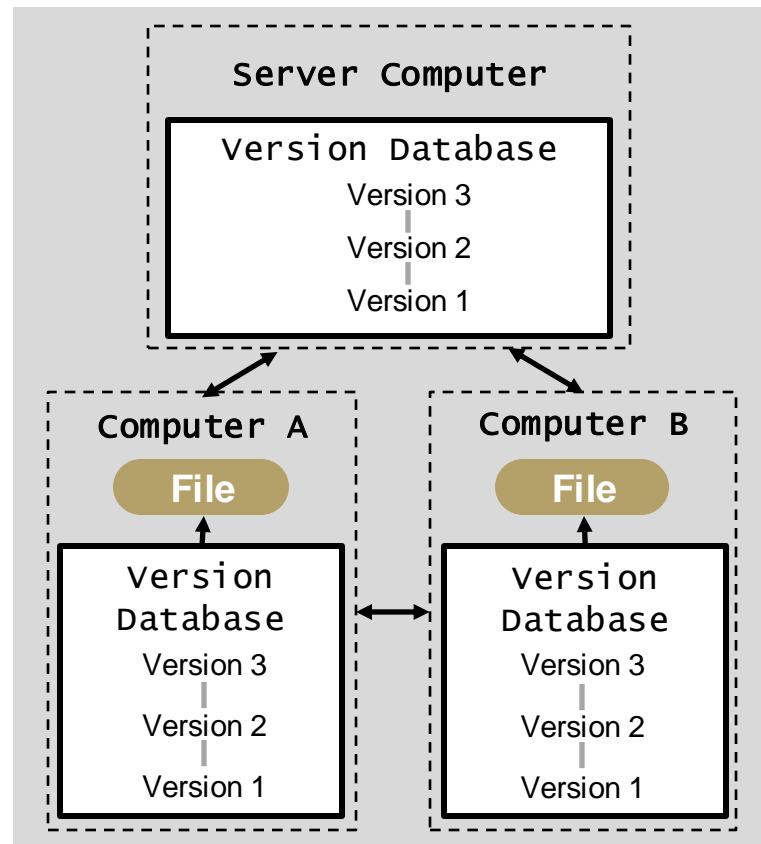
In a distributed VCS, every client mirrors the full project history

### Distributed Version Control System

- Clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.

#### Advantages:

- All advantages of a CVCS
- Every clone is really a full backup of all the data; thus, if any server dies, any of the client repositories can be copied back up to the server to restore it.



# Git is a very popular and widespread DVCS



A screenshot of a Twitter post from user **jackwilliambell** (@jackwilliambell). The tweet reads: "It is easy to shoot your foot off with #git, but also easy to revert to a previous foot and merge it with your current leg. #revisioncontrol". The timestamp is 10:20 - 18. Aug. 2010. The post has 555 Retweets and 88 likes. Below the tweet, there are icons for replying, retweeting, and favoriting.

– Jack William Bell

## Why git?

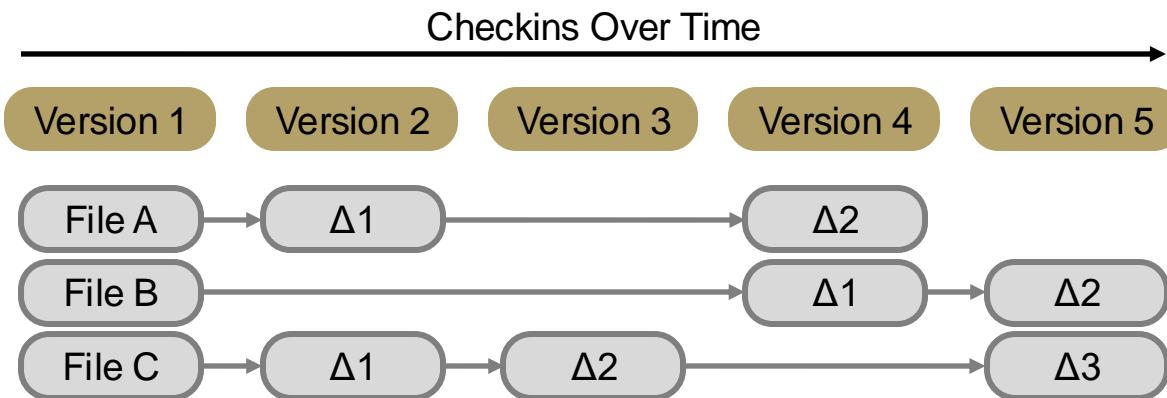
- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects efficiently (speed and data size)
- Note: Git is primarily designed to track text files!

<https://twitter.com/jackwilliambell/status/21506465315> (last retrieved on August 14, 2018)  
Chacon, S. & Straub, B. (2014): *Pro Git*. Apress. (p. 13)

Git thinks of its data like a stream of snapshots rather than a list of file-based changes

**Delta-based VCS**

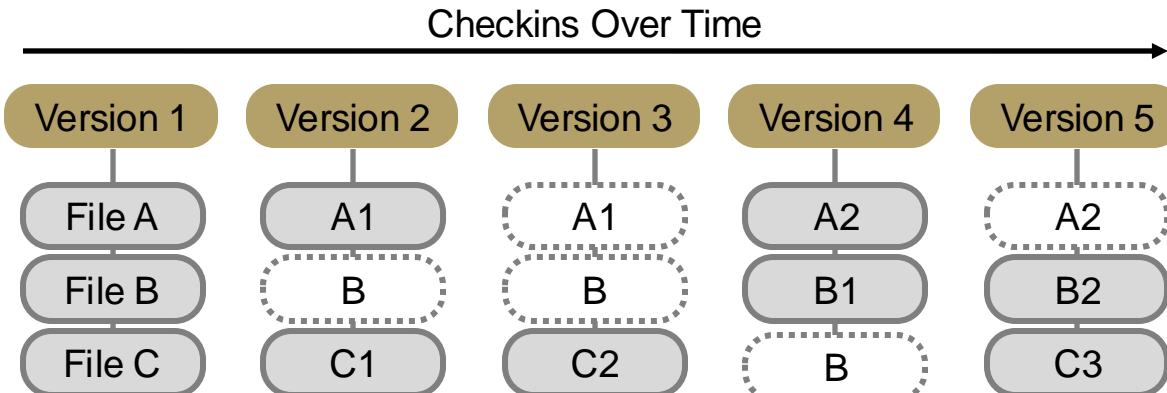
- Storing data as changes to a base version of each file.



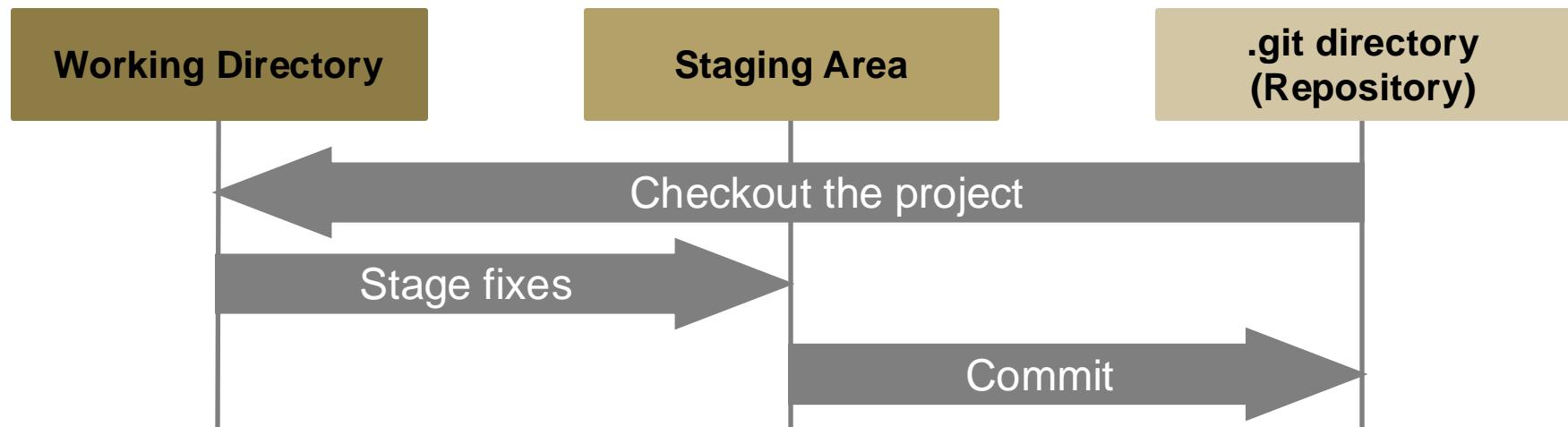
 **git**

**Stream of snapshots**

- Storing data as snapshots of the project over time.
- Allows for efficient branching in Git.



Your files can reside in three states: *committed*, *modified*, and *staged*



- *Modified* means that you have changed the file but have not committed it to your database yet.
- *Staged* means that you have marked a modified file in its current version to go into your next commit snapshot.
- *Committed* means that the data is safely stored in your local database.

# A Git project consists of three main sections

## .git directory (Repository)

The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

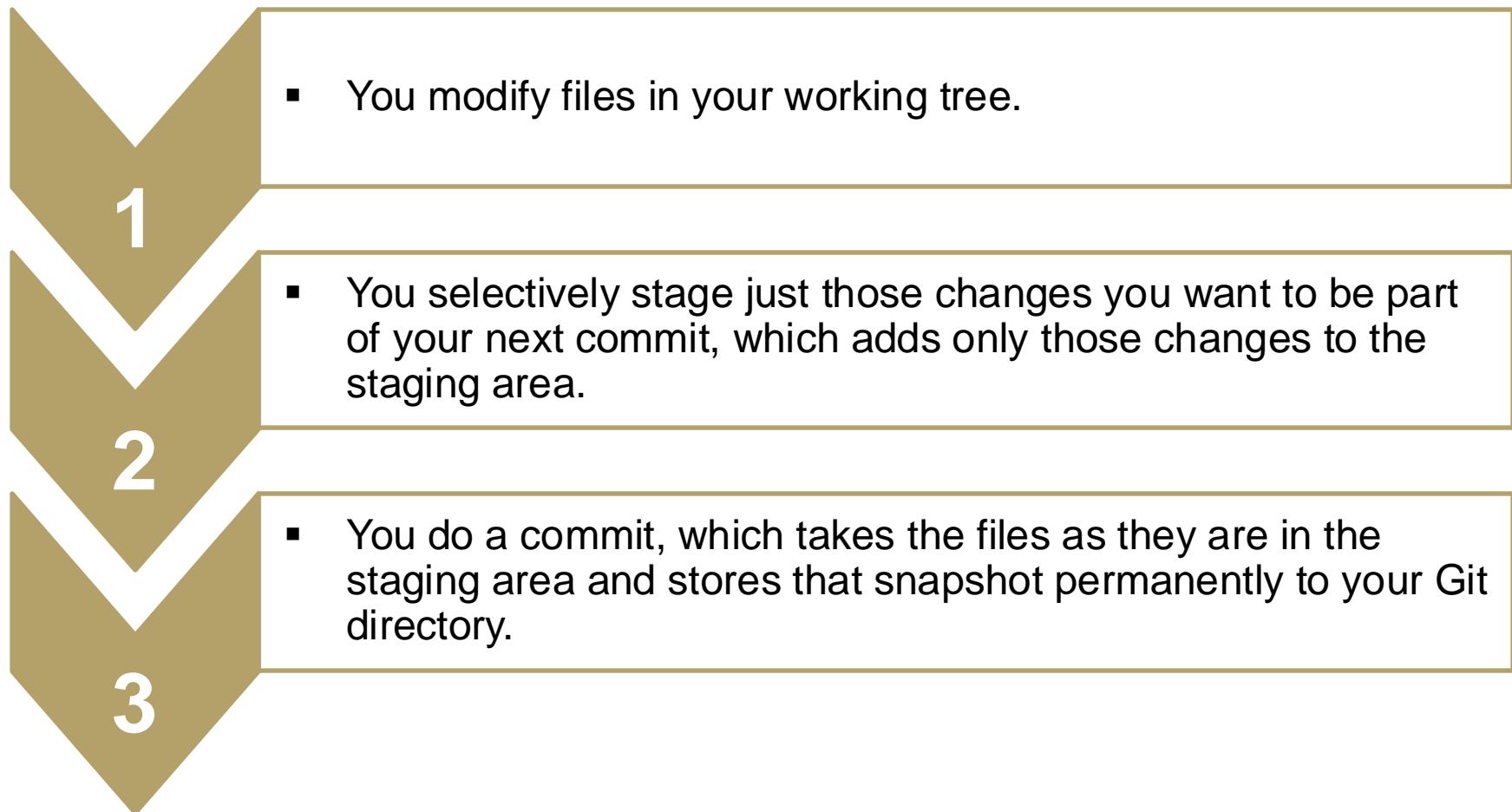
## Staging Area

The staging area is a file, generally contained in your Git directory, that stores information about what will go into your next commit.

## Working Directory

The working directory, or working tree, is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.

The basic workflow can be summarized as: *modify – stage – commit*



# Installing Git on your computer

The most recent versions of Git for your OS are available for free download at:

<https://git-scm.com/download>

The screenshot shows the official Git website (<https://git-scm.com/>) with a focus on the 'Downloads' section. The page features the Git logo and the tagline '--distributed-is-the-new-centralized'. On the left, there's a sidebar with links for 'About', 'Documentation', 'Downloads' (which is highlighted in red), and 'Community'. Below the sidebar, a box mentions the 'Pro Git book' by Scott Chacon and Ben Straub. The main content area has a large 'Downloads' heading. Underneath it, there are three buttons for 'Mac OS X', 'Windows', and 'Linux/Unix'. To the right, a large monitor icon displays a window for the 'Latest source Release 2.18.0' with a 'Download 2.18.0 for Windows' button. Below the monitor, there's a section for 'GUI Clients' explaining that Git comes with built-in tools like `git-gui` and `gitk`, and a link to 'View GUI Clients →'. Another section for 'Logos' describes various Git logos available in PNG and EPS formats, with a link to 'View Logos →'. A search bar is located at the top right of the main content area.

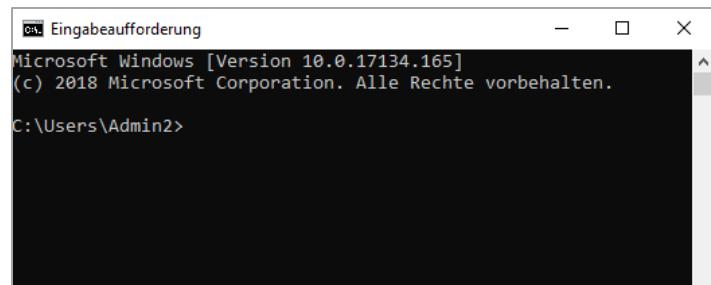
<https://git-scm.com/downloads> (last retrieved on August 14, 2018)

# There are different ways of using Git

- The command line is the only place that supports *all* Git commands
- Different graphical user interfaces (GUIs) implement a partial subset of Git functionality for simplicity
- In this course, we will access Git **via the command line**, because...
  - ... the choice of a graphical client is a matter of personal taste
  - ... if you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true

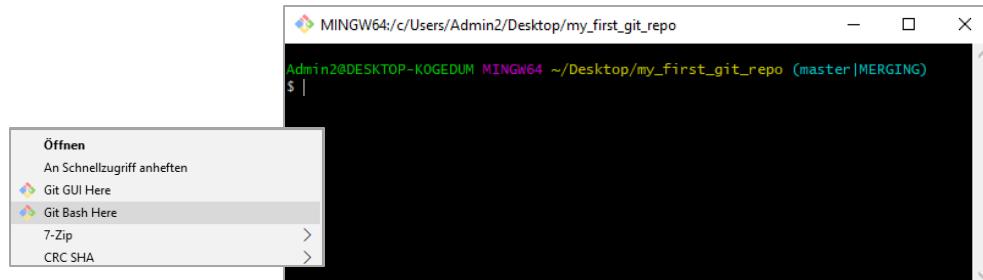
Open *Windows Command Prompt*  
(also known as *cmd.exe* or *cmd*):

- Click *Start*.
- Type ‘cmd’ and press *Enter*.



Alternatively, you can use the  
Git command line editor:

- Right-click on target folder
- Click on ‘Git Bash Here’



## Before you start, you need to customize your Git environment once

Configure your Git username and email using the following commands, (replacing the values with your own name and email address). These details will be associated with all commits that you create.

```
$ git config --global user.name "Your Name"
$ git config --global user.email "your.name@student.uni-tuebingen.de"
```

### On the structure of shell commands

- A shell command line can be broken into three parts:
  - The command name (i.e., the name of the command or the program)
  - An option to the command (changes the behavior of the command)
  - An argument to the command (additional information to the command)



git config --global user.name "Your Name"

command      option      argument

## Exercises



1. Download and install Git
2. Adjust the configuration of your Git environment (user name and email)



# A reference guide for some common Git commands

	<b>command</b>	<b>description</b>
Configuration	git config --global user.name “[full name]”	Sets the name you want attached to your commit transactions
	git config --global user.email “[email]”	Sets the email you want attached to your commit transactions
	git config --global --list	Checks if you’re set up is correctly specified
Creating repository	git init	Creates a new local repository with the specified name
	git clone [url]	Downloads a project and its entire version history
Make changes	git status	Lists modified files in working directory, staged for your next commit
	git add [file]	Snapshots the file in preparation for versioning
	git reset [file]	Unstage a file while retaining the changes in working directory
	git diff	Shows file differences not yet staged
	git diff --staged	Shows file differences between staging and the last file version
	git commit -m “[descriptive message]”	Records file snapshots permanently in version history

<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf> (last retrieved on November 29, 2018)



## A reference guide for some common Git commands (cont'd)

	<b>command</b>	<b>description</b>
Refactoring filenames	git rm [file]	Deletes the file from the working directory and stages the deletion
	git rm --cached [file]	Removes the file from version control but preserves the file locally
	git mv [file-original] [file-renamed]	Changes the file name and prepares it for commit
Undoing changes	git reset [commit]	Undoes all commits after [commit], preserving changes locally
	git reset --hard [commit]	Discards all history and changes back to the specified commit
Branching & merging	git revert [commit]	Creates new commit that undoes all of the changes made in [commit], then applies it to the current branch.
	git reset [file]	Removes [file] from the staging area, but leaves the working directory unchanged. This unstages a file without overwriting any changes.
Branching & merging	git branch [new-branch]	Creates a new branch based on current HEAD
	git checkout [branch]	Switches HEAD branch to [branch]
	git merge [branch]	Merges [branch] into your current HEAD branch

## First, a Git repository needs to be initialized

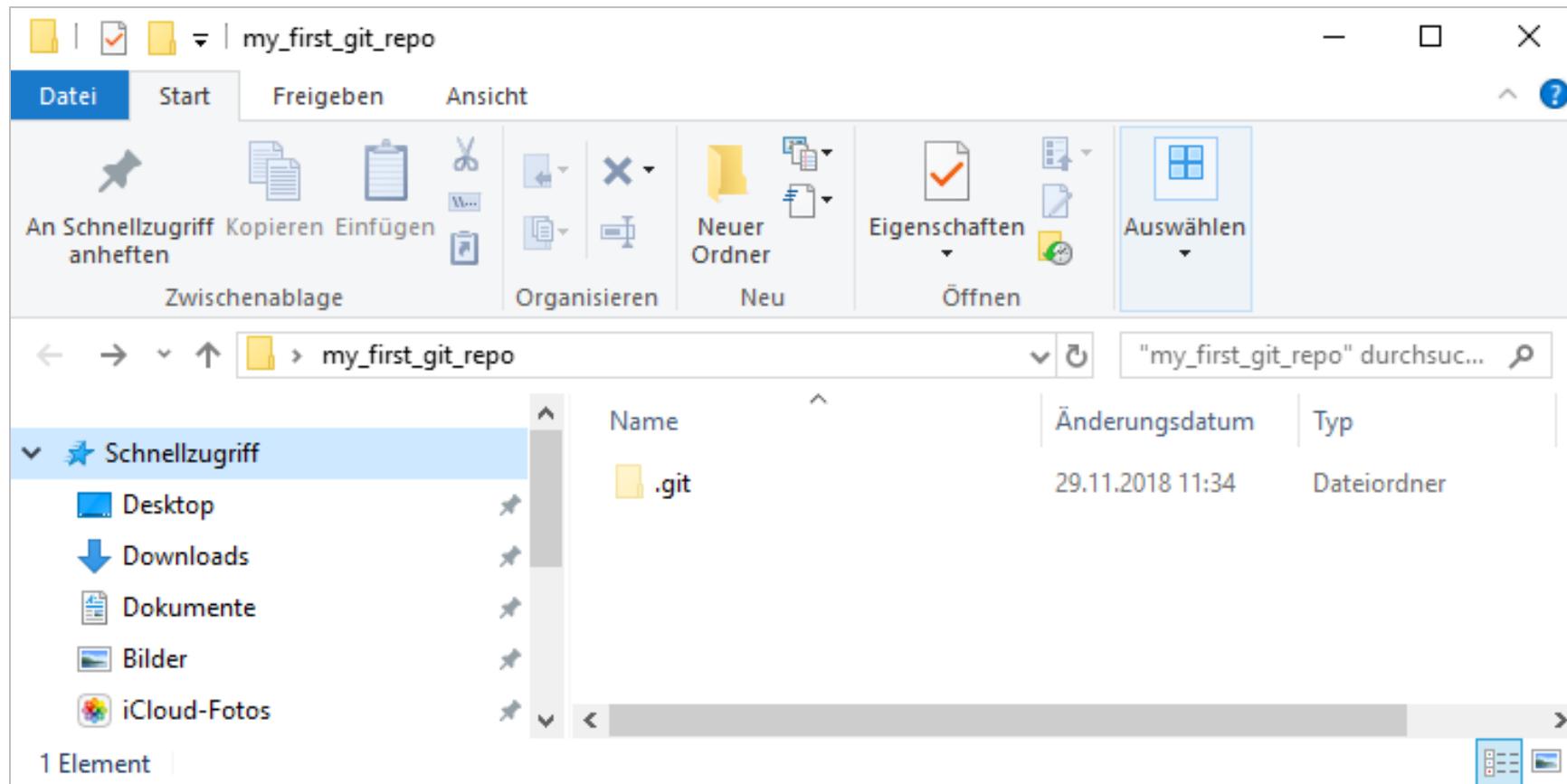
- We initialize a new local Git repository and change the working directory to this repository:

```
$ git init C:/Users/Stefan/Desktop/my_first_git_repo
 Initialized empty Git repository in
 C:/Users/Stefan/Desktop/my_first_git_repo/.git/
$ cd C:/Users/Stefan/Desktop/my_first_git_repo
$ git status
 On branch master
 No commits yet
 nothing to commit (create/copy files and use "git add" to track)
```

- As we have just created this folder, there are no files that could be staged (this can be checked using `git status`)

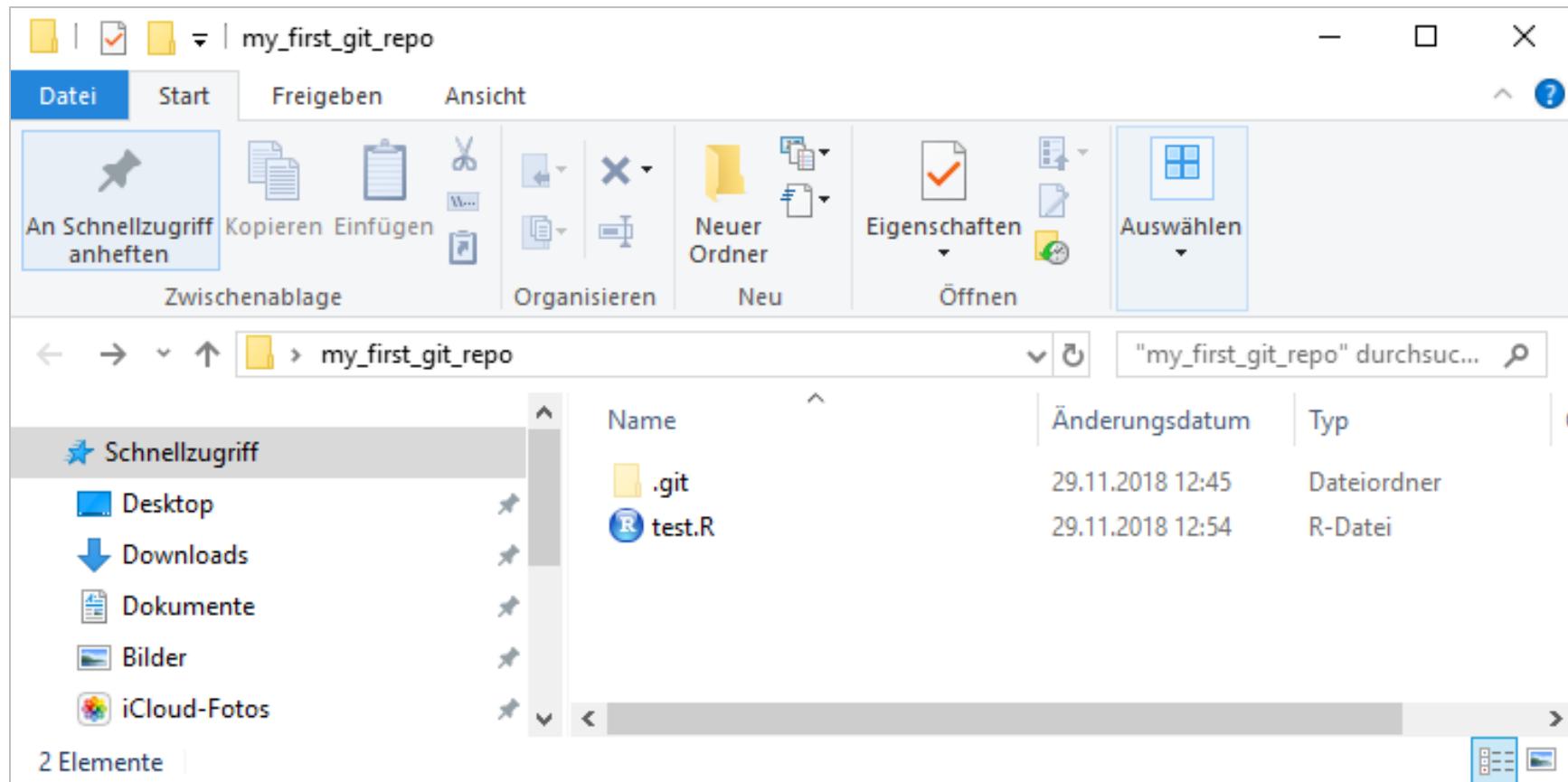
## First, a Git repository needs to be initialized

- The `git init` command has created a hidden folder called “`.git`” with all necessary repository information (configuration, commit history, etc.)



Each change we make to the tracked files can be registered by Git

- Let us now create a new `*.R`-script “`test.R`” and save it to our Git repository



## Each change we make to the tracked files can be registered by Git

- If we now check the current status of the project repository, we see that something has changed:

```
$ git status
 On branch master

 No commits yet

 Untracked files:
 (use "git add <file>..." to include in what will be committed)

 test.R

 nothing added to commit but untracked files present (use "git add" to track)
```

- The comments tell us that there are changes in the repository, but that these have not yet been “added to commit” (i.e., moved to the stage)

## Each change we make to the tracked files can be registered by Git

- Before we can commit a change (i.e., record it permanently in the version history), we need to stage the change using `git add`:

```
$ git add test.R

$ git status
 On branch master

 No commits yet

 Changes to be committed:
 (use "git rm --cached <file>..." to unstage)

 new file: test.R

$ git commit -m "Created new R-script test.R"
[master (root-commit) 8c003cc] Created new R-script test.R
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.R
```

## The history of commits is stored permanently

- We can display the history of commits using `git log`. Each commit has a unique ID:

```
$ git log
commit 8c003cc97c3041e8e80cc5b705a5cd3696e5ce3d (HEAD -> master)
Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
Date: Thu Nov 29 13:14:16 2022 +0100

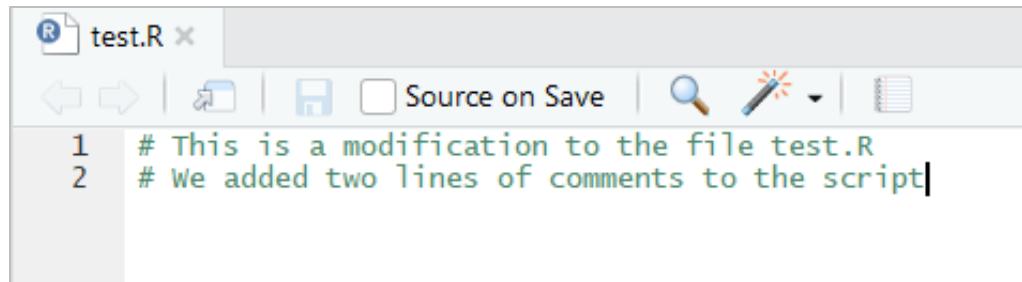
 Created new R-script test.R
```

- `git status` now again tells us that there are no uncommitted changes :

```
$ git status
On branch master
nothing to commit, working tree clean
```

## The history of commits is stored permanently

- We modify the tracked file *test.R* and save the changes made:



A screenshot of an RStudio interface showing a code editor window titled "test.R". The window contains the following R script:

```
1 # This is a modification to the file test.R
2 # We added two lines of comments to the script|
```

The code editor has a toolbar with various icons for file operations, search, and edit.

- `git diff` shows us what has been changed since the last commit:

```
$ git diff
diff --git a/test.R b/test.R
index e69de29..ca601ce 100644
--- a/test.R
+++ b/test.R
@@ -0,0 +1,2 @@
+# This is a modification to the file test.R
+# We added two lines of comments to the script
\ No newline at end of file
```

## The history of commits is stored permanently

- Again, we stage the changes using `git add` (the option "`--all`" / "`-A`" adds all unstaged changes to the stage) and then commit the change:

```
$ git add -A

$ git commit -m "Added two lines of comments"
[master 32a2983] Added two lines of comments
1 file changed, 2 insertions(+)
```

- `git log` now has a second entry:

```
$ git log
commit 32a29835348afe1fe5c48d3bfac7cc32dbdb5863 (HEAD -> master)
Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
Date: Fri Nov 30 13:39:52 2022 +0100

 Added two lines of comments

commit 8c003cc97c3041e8e80cc5b705a5cd3696e5ce3d
Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
Date: Thu Nov 29 13:14:16 2022 +0100

 Created new R-script test.R
```

## A 40-character ID identifies each commit that was ever made

- If we note that we've made a faulty commit, we can directly undo it by calling `git revert [commit ID]`, where the commit ID is the 40-character SHA-1 hash that can be found in the log:

```
$ git revert 32a29835348afe1fe5c48d3bfac7cc32dbdb5863

$ git log
 commit 2becc83c14b7c38aea0543e5796c53652caf8757 (HEAD -> master)
 Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
 Date: Fri Nov 30 13:45:15 2022 +0100

 Revert "Added two lines of comments"

 This reverts commit 32a29835348afe1fe5c48d3bfac7cc32dbdb586

 commit 32a29835348afe1fe5c48d3bfac7cc32dbdb5863
 Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
 Date: Fri Nov 30 13:39:52 2022 +0100

 Added two lines of comments

 commit 8c003cc97c3041e8e80cc5b705a5cd3696e5ce3d
 Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
 Date: Thu Nov 29 13:14:16 2022 +0100

 Created new R-script test.R
```

## In practice, a shortened version of the commit ID is sufficient

- Git is smart enough to figure out what commit you're referring to if you provide the first few characters of the SHA-1 hash, as long as that partial hash is at least four characters long and unambiguous
- If you pass `--abbrev-commit` to the `git log` command, the output will use shorter values but keep them unique; it defaults to using seven characters but makes them longer if necessary to keep the SHA-1 unambiguous:

```
$ git log --abbrev-commit --oneline
 32a2983 (HEAD -> master) Added two lines of comments
 8c003cc Created new R-script test.R
```

- Hence, instead of using the full ID to revert a commit, we can use the shortened version:

```
$ git revert 32a2983

$ git log --abbrev-commit --oneline
 2becc83 (HEAD -> master) Revert "Added two lines of comments"
 32a2983 Added two lines of comments
 8c003cc Created new R-script test.R
```

## We can easily reset the repository's status to previous states

- If a whole series of commits turns out to be flawed, you may want to return to an earlier version of a file
- This can be done by calling `git reset [commit ID]`:

```
$ git reset 8c003cc

$ git log
 commit 8c003cc97c3041e8e80cc5b705a5cd3696e5ce3d
 Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
 Date: Thu Nov 29 13:14:16 2022 +0100

 Created new R-script test.R
```

- While `git log` does not show the history of commits that followed the commit that we have reset, `git reflog` still shows the full history:

```
$ git reflog
 789856f (HEAD -> master) HEAD@{0}: reset: moving to 8c003cc
 2becc83 HEAD@{1}: revert: Revert "Added two lines of comments"
 32a2983 HEAD@{2}: commit: Added two lines of comments
 8c003cc (HEAD -> master) HEAD@{3}: commit (initial): Created new R-script test.R
```

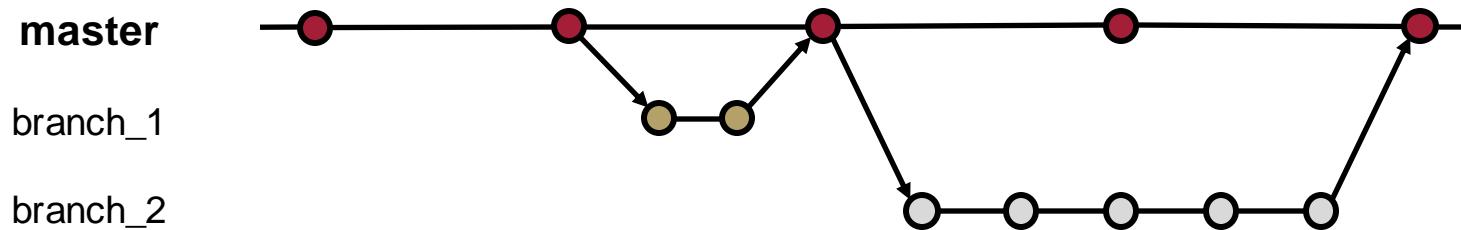
## Exercises



3. Initiate a new local Git repository
4. Create a script and commit your first changes to the repository

# Git supports non-linear development of code by means of *branching*

- Branching means you diverge from the main line of development (*master*) and continue to do work without messing with that main line
- It is good practice to create a branch whenever you are developing a new feature or experimenting with your code
- Only if your feature works flawlessly, you merge it back into the master branch



- You create a new branch with `git branch [branch name]` and you switch to this branch with `git checkout [branch name]` (or you do both at the same time by calling `git checkout -b [branch name]`)

## Before committing changes to a branch, it needs to be “checked out”

- We create a new branch called “test” and we switch to that branch:

```
$ git branch test

$ git status
 On branch master
 nothing to commit, working tree clean

$ git checkout test
 Switched to branch ‘test’

$ git status
 On branch test
 nothing to commit, working tree clean

$ git log --abbrev-commit
 commit 8c003cc (HEAD -> test, master)
 Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
 Date: Thu Nov 29 13:14:16 2022 +0100

 Created new R-script test.R
```

## Changes to branches work as before

- We make a change to our file, stage the change and commit it:

```
$ git add -A

$ git commit -m "A line has been added (on 'test' branch)"
[testing 4d34080] A line has been added (on 'test' branch)
 1 file changed, 1 insertion(+)

$ git log --abbrev-commit
commit 4d34080 (HEAD -> test)
Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
Date: Thu Nov 29 14:01:01 2022 +0100

 A line has been added (on 'test' branch)

commit 8c003cc (master)
Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
Date: Thu Nov 29 13:14:16 2022 +0100

 Created new R-script test.R
```

## Branches can be merged back into the master branch using git merge

- To merge the branch “test” back into the master branch, we need to switch to the master call `git merge test`:

```
$ git checkout master
 Switched to branch 'master'

$ git merge test
 Updating 8c003cc..4d34080
 Fast-forward
 test.R | 1 +
 1 file changed, 1 insertion(+)

$ git log --abbrev-commit
 commit 4d34080 (HEAD -> master, test)
 Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
 Date: Thu Nov 29 14:01:01 2022 +0100

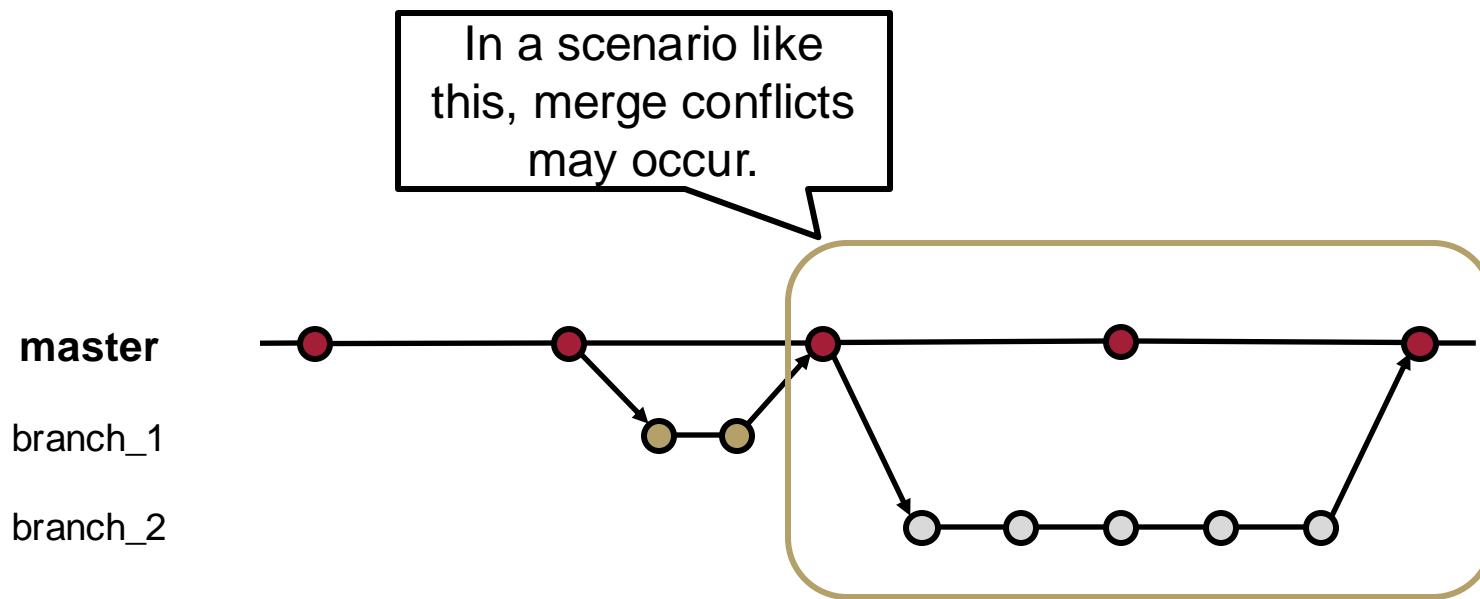
 A line has been added (on 'test' branch)

 commit 8c003cc
 Author: Stefan Mayer <stefan.mayer@uni-tuebingen.de>
 Date: Thu Nov 29 13:14:16 2022 +0100

 Created new R-script test.R
```

<https://git-scm.com/book/de/v2/Git-Branching-Basic-Branching-and-Merging> (last retrieved on January 07, 2019)

## Merging branches is not always unambiguous



# Watch out for merge conflicts!



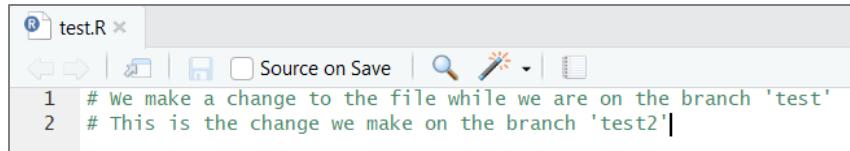
<https://giphy.com/gifs/git-merge-cFkiFMDg3iF0I> (last retrieved on April 06, 2021)

# Concurrent commits to different branches can cause conflicts

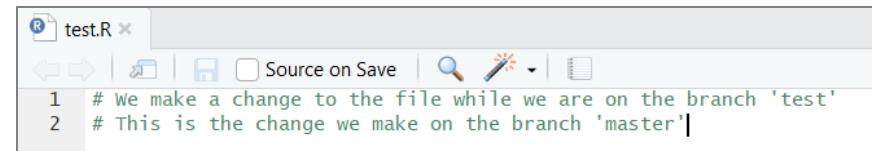
- When changes are made simultaneously on different branches, conflicts may occur when merging:

```
$ git checkout -b test2
Switched to a new branch 'test2'
```

```
$ git checkout master
Switched to branch 'master'
```



```
R test.R ×
Source on Save |
1 # We make a change to the file while we are on the branch 'test'
2 # This is the change we make on the branch 'test2'|
```



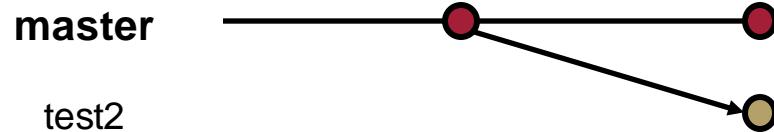
```
R test.R ×
Source on Save |
1 # We make a change to the file while we are on the branch 'test'
2 # This is the change we make on the branch 'master'|
```

```
$ git add -A

$ git commit -m "Changed line 2 on branch test2"
[test2 f47ae51] Changed line 2 on branch test2
1 file changed, 2 insertions(+), 1 deletion(-)
```

```
$ git add -A

$ git commit -m "Changed line 2 on branch master"
[master 3310e60] Changed line 2 on branch master
1 file changed, 2 insertions(+), 1 deletion(-)
```

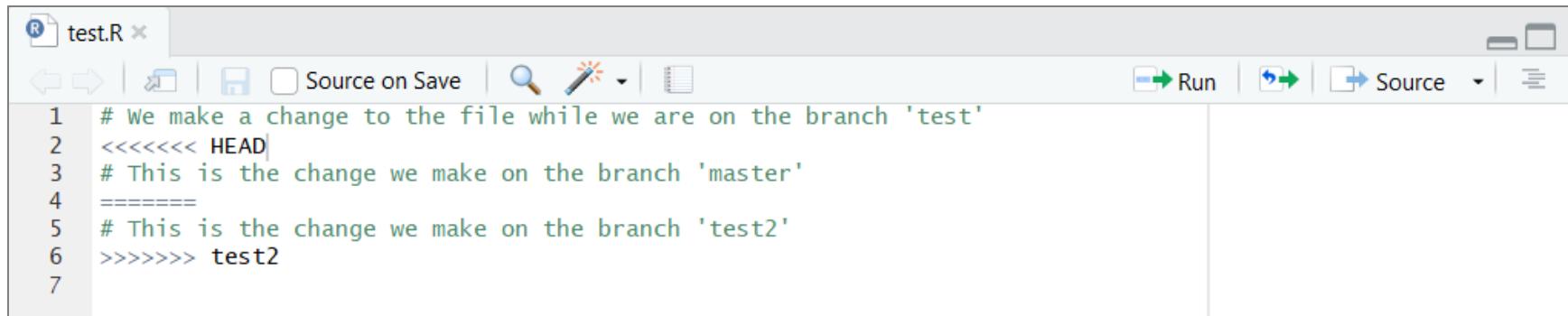


**Which of the conflicting versions of line 2 should be kept?**

## You need to decide what version to keep

```
$ git merge test2
 Auto-merging test.R
 CONFLICT (content): Merge conflict in test.R
 Automatic merge failed; fix conflicts and then commit the result.
```

- This is what our file looks like after the merging the conflicting branches:



The screenshot shows a code editor window titled "test.R". The code content is as follows:

```
1 # We make a change to the file while we are on the branch 'test'
2 <<<<< HEAD
3 # This is the change we make on the branch 'master'
4 =====
5 # This is the change we make on the branch 'test2'
6 >>>>> test2
7
```

The code editor interface includes tabs for "Run", "Source", and other development tools.

- Git recognizes the conflict, informs us about it, highlights it in the merged document, and asks us to resolve it
- Hence, we need to delete the obsolete lines in the document and commit the changes

# Merge tools may help for easier handling of merge conflicts

- Merge tools like for example [Meld](#) can be used with Git to resolve more complex merge conflicts with a graphical interface

```
Loading relevant package
library(ggplot2)

Creating plot
scatterplot <- ggplot(diamonds, aes(x = carat, y = price)) +
 geom_point() +
 theme_minimal() +
 geom_title("Diamonds Scatterplot")

Saving plot
ggsave("./plots/plot.jpg")
```

```
Loading relevant package
library(ggplot2)

Creating plot
scatterplot <- ggplot(diamonds, aes(x = carat, y = price, col = clarity)) +
 geom_point() +
 theme_minimal()

Saving plot
ggsave("./plots/plot.jpg")
```

<http://meldmerge.org/> (last retrieved on January 07, 2019)

## Exercises



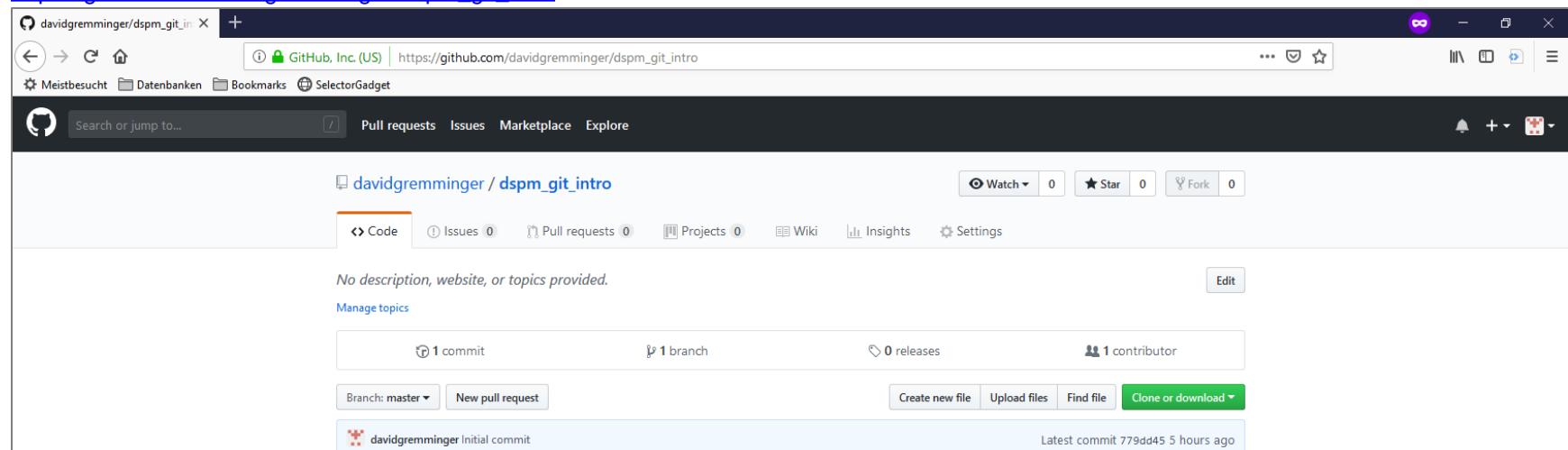
5. Try out branching and merging. Create a branch, commit a change and merge back into the master

# Git unleashes its full potential if it is combined with GitHub



- GitHub is a web-based hosting service for version control using Git
- ~31 million developers are hosting ~100 million repositories on GitHub worldwide (November 2018)
- GitHub provides paid enterprise solutions, but even with a free account, you can host an unlimited number of public repositories as well as private repositories with up to three collaborators

[https://github.com/davidgremminger/dspm\\_git\\_intro](https://github.com/davidgremminger/dspm_git_intro)



A screenshot of a GitHub repository page for 'davidgremminger / dspm\_git\_intro'. The page shows basic repository statistics: 1 commit, 1 branch, 0 releases, and 1 contributor. It also features tabs for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. A prominent green button at the bottom right says 'Clone or download'.

<https://blog.github.com/2019-01-07-new-year-new-github/> (last retrieved on January 08, 2019)

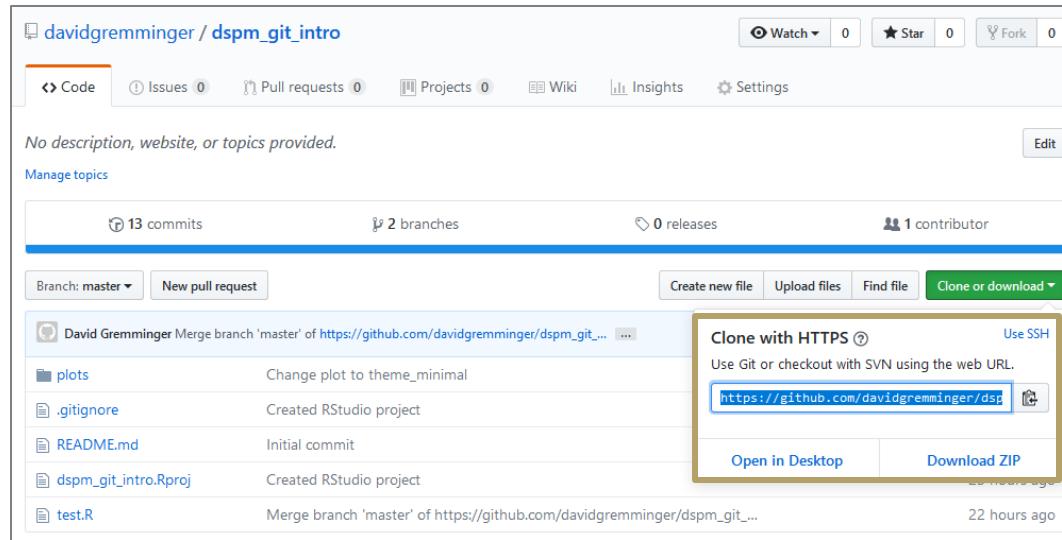
<https://github.com/> (last retrieved on January 08, 2019)

Git unleashes its full potential if it is combined with GitHub



- It is important to realize that Git and GitHub are distinct things
- GitHub is an online hosting platform that provides an array of services built on top of the Git system (Similar platforms include Bitbucket and GitLab)
- Just like we don't *need* RStudio to run R code, we don't *need* GitHub to use Git... but it will make our lives so much easier

# To work with a remote repo, you need the corresponding GitHub URL



- Call `git clone [repo URL]` to create a local copy of the remote repository in your working directory:

```
$ git clone https://github.com/davidgremminger/dspm_git_intro/
Cloning into 'dspm_git_intro'...
```

- You can now work with the clone of the remote repository as with any local repository (`status`, `log`, `add`, `commit`, ...)

[https://github.com/davidgremminger/dspm\\_git\\_intro](https://github.com/davidgremminger/dspm_git_intro) (last retrieved on January 09, 2019)

The latest version of a remote repo can be checked out using `git pull`

- Once you have a local clone of a remote repository on your drive, you can update to the most recent version calling `git pull`
- This is of particular importance if, e.g., a collaborator is simultaneously committing changes to your repository

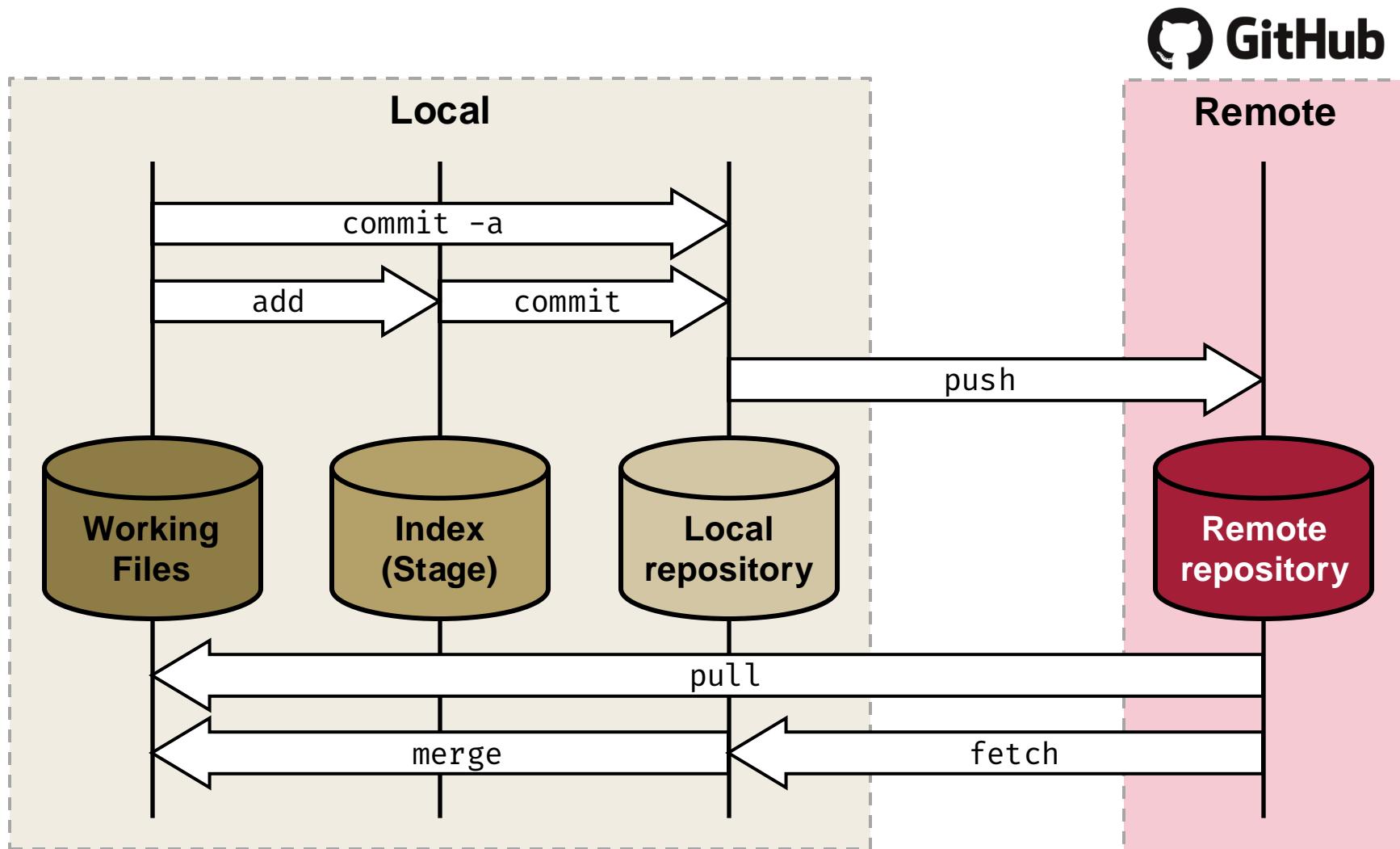
```
$ git pull
Already up to date.
```

This notification indicates that no commits have been made since you have last updated your clone.

- After you have made and committed one or more changes to your local clone, you can send the changes back to the remote repository by using `git push`:

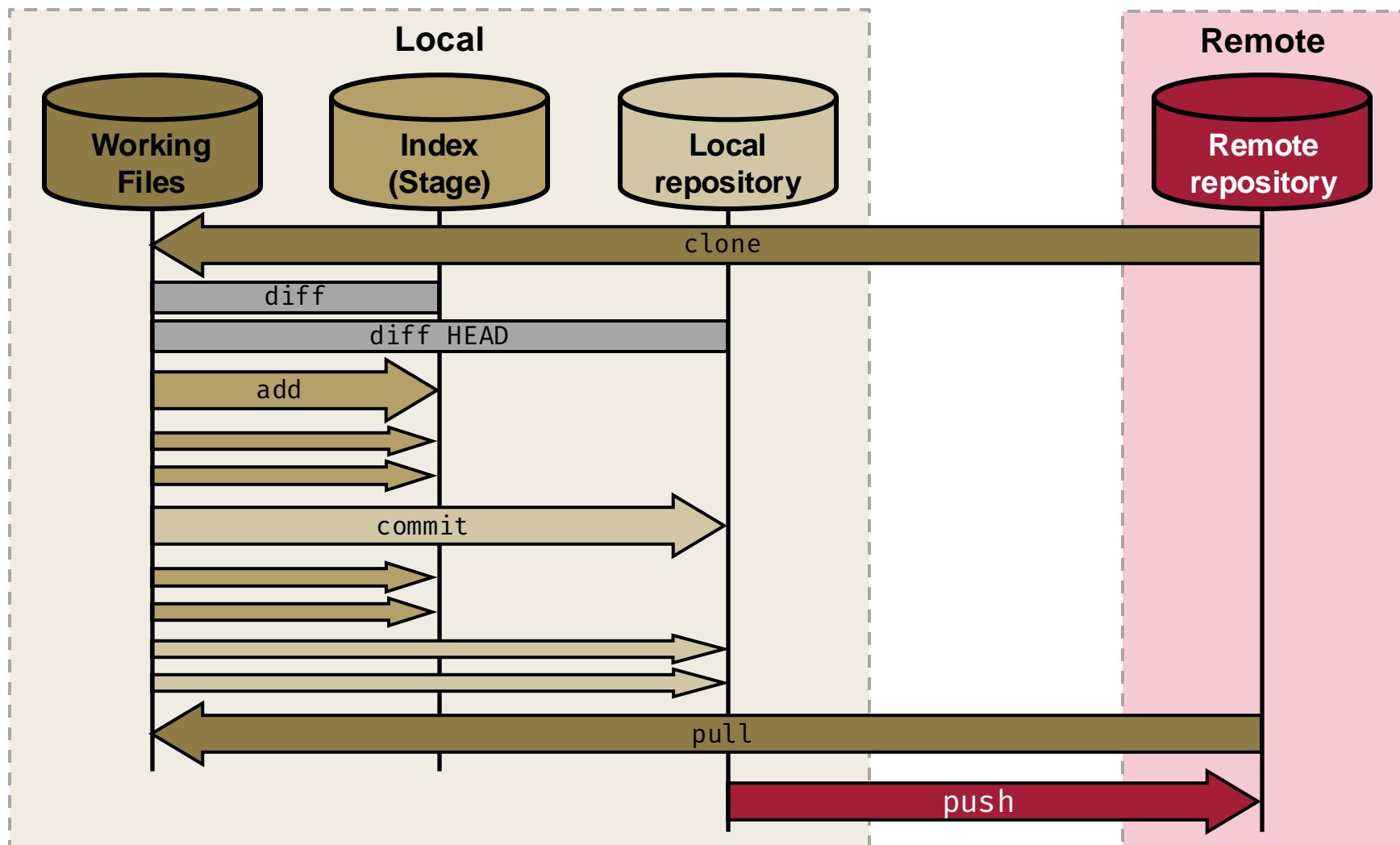
```
$ git push
Counting objects: 3, done.
[...]
To https://github.com/davidgremminger/dspm_git_intro/
 eff4c30..5266b75 master -> master
```

# Exchanging data between the remote and the local repository



Own representation based on: <https://blog.osteele.com/2008/05/my-git-workflow/> (last retrieved on January 09, 2019)

# The schematic workflow of a Git project



Own representation based on: <https://blog.osteele.com/2008/05/my-git-workflow/> (last retrieved on January 09, 2019)

## What if we have files that we do not want Git to track for us?

- Often, you'll have a class of files that you don't want Git to automatically add or even show you as being untracked
- These are generally automatically generated files such as log files or files produced by your build system
- In such cases, you can create a file listing patterns to match them named `.gitignore`

### Rules for the patterns in the `.gitignore` file:

- Blank lines or lines starting with # are ignored
- An asterisk (\*) matches zero or more characters
- [abc] matches any character inside the brackets (in this case a, b, or c)
- A question mark (?) matches a single character
- Brackets enclosing characters separated by a hyphen ([0-9]) matches any character between them (in this case 0 through 9)
- You can end patterns with a forward slash (/) to specify a directory
- You can negate a pattern by starting it with an exclamation point (!)

## An exemplary .gitignore file

```
no .a files
*.a

but do track lib.a, even though you're ignoring .a files above
!lib.a

only ignore the TODO file in the current directory, not subdir/TODO
/TODO

ignore all files in the build/ directory
build/

ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

**Note:**

GitHub maintains a fairly comprehensive list of good .gitignore file examples for dozens of projects and languages at <https://github.com/github/gitignore> if you want a starting point for your project.

## Exercises



6. Create a GitHub account and repeat the workflow from above - now storing the history of changes in a remote repository

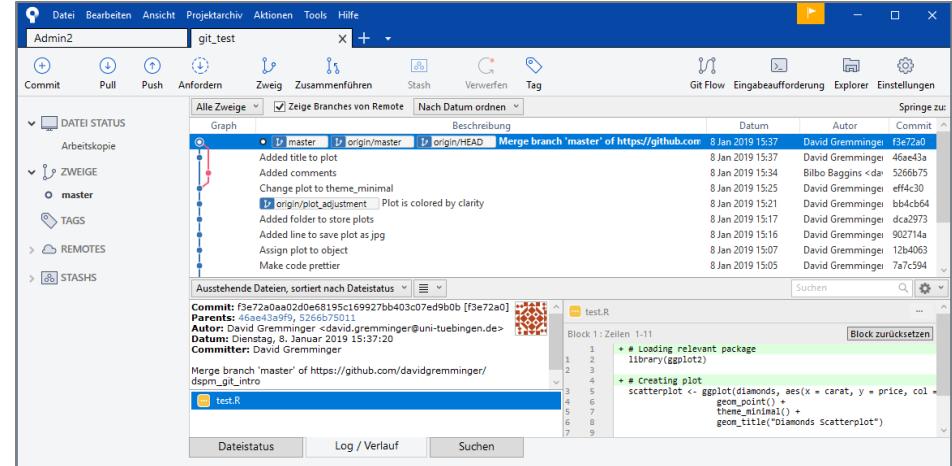
# The use of a Git GUI client might facilitate your workflow

**Remember why we have accessed Git via the command line, so far:**

- The command line is the only place that supports *all* Git commands
- Different GUIs implement a partial subset of Git functionality for simplicity
- In this course, we have accessed Git **via the command line**, because...
  - ... the choice of a graphical client is a matter of personal taste
  - ... if you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true
- In practice, however, you might be working with a Git GUI and switch to the command line only for functions that the GUI does not support.
- Some of the most popular open-source Git GUI clients are:
  - [Gitk](#) (comes installed with Git)
  - [Sourcetree](#)
  - [GitHub Desktop](#)
  - [TortoiseGit](#) (Windows only)
  - [Git Extensions](#) (native Windows only)
  - [GitKraken](#) (only free for local/public repositories)
- Moreover, RStudio supports some basic Git version control functionality when you are working with an RStudio project



# The use of a Git GUI client might facilitate your workflow



**SourceTree**

Commit history:

- Added title to plot
- Added comments
- Change plot to theme\_minimal
- origin/plot\_adjustment Plot is colored by clarity
- Added folder to store plots
- Added line to save plot as jpg
- Assign plot to object
- Make code prettier
- Created scatterplot in 'test.R'
- Created RStudio project

Code diff for test.R:

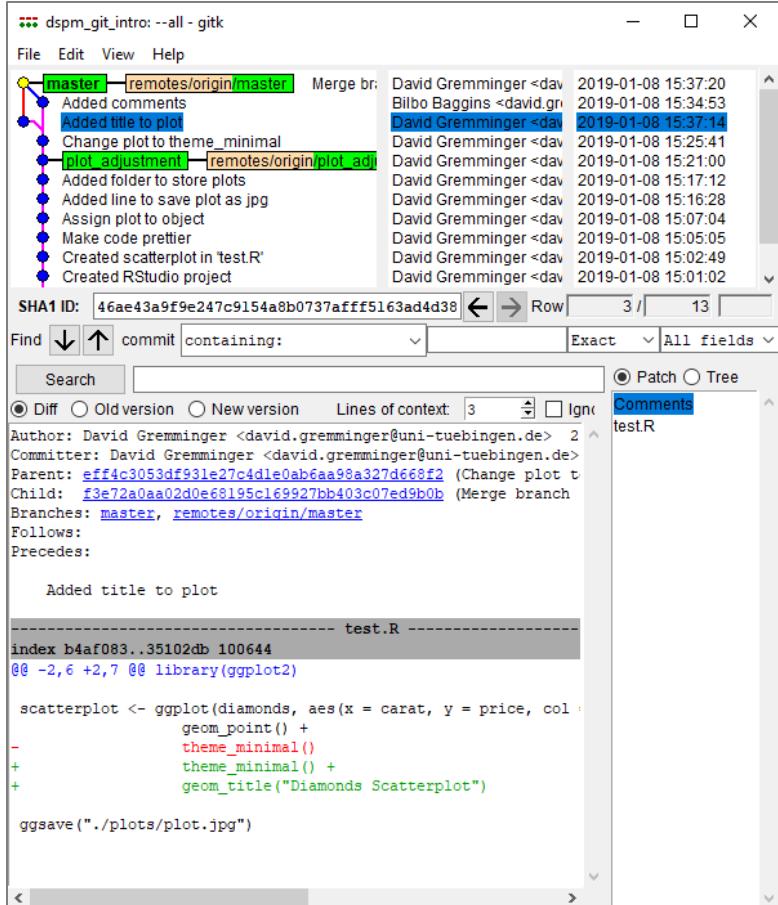
```

index b4af083..35102db 100644
@@ -2,6 +2,7 @@ library(ggplot2)

scatterplot <- ggplot(diamonds, aes(x = carat, y = price, col =
- geom_point() +
+ theme_minimal()
+ theme_minimal() +
+ geom_title("Diamonds Scatterplot"))

ggsave("./plots/plot.jpg")

```



**gitk**

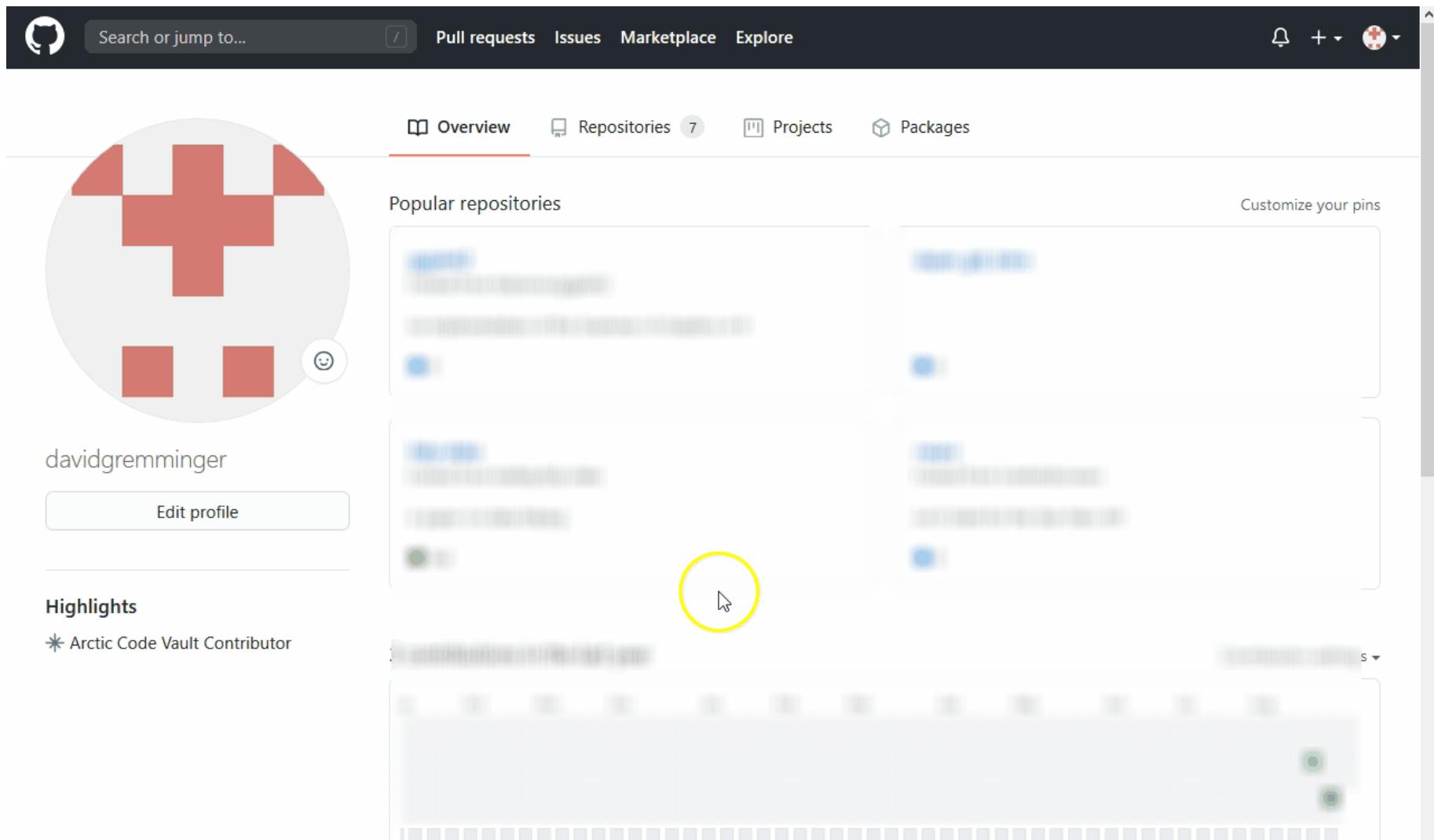




# Git(Hub) and RStudio – integrating Git into your everyday workflow

- The starting point for our workflow is to link a GitHub repository (i.e. “repo”) to an RStudio Project. Here are the steps we’re going to follow:
  1. Create the repo on GitHub and initialize with a README
  2. Copy the HTTPS/SSH link (the green “Code” button)
  3. Open up RStudio
  4. Navigate to **File -> New Project -> Version Control -> Git**
  5. Paste your copied link into the “Repository URL:” box
  6. Choose the project path (“Create project as subdirectory of:”) and click **Create Project**

# Integrating a new GitHub repo into RStudio – a quick walkthrough



A screenshot of a GitHub profile page for the user `davidgremminger`. The page includes a profile picture with a red cross, a search bar, and navigation links for Overview, Repositories (7), Projects, and Packages. The 'Popular repositories' section displays a grid of repository cards. A yellow circle highlights the cursor hovering over the first repository in the top-left row.



## Track your changes right from RStudio

- Once the new project is opened in RStudio, open the “Git” tab on the upper right-hand side
- Open the README file and make a change
- While you save the change, see what happens in the “Git” panel: The README file is now listed with the status *Modified*
- We can now execute all the basic steps that we’ve learned in this chapter:

```
modify - stage - commit - pull - push - ...
```



# Make your first local change in RStudio – a quick walkthrough

The screenshot shows the RStudio interface with the following components:

- Console Pane:** Displays the R startup message and a single character input '>'.

```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

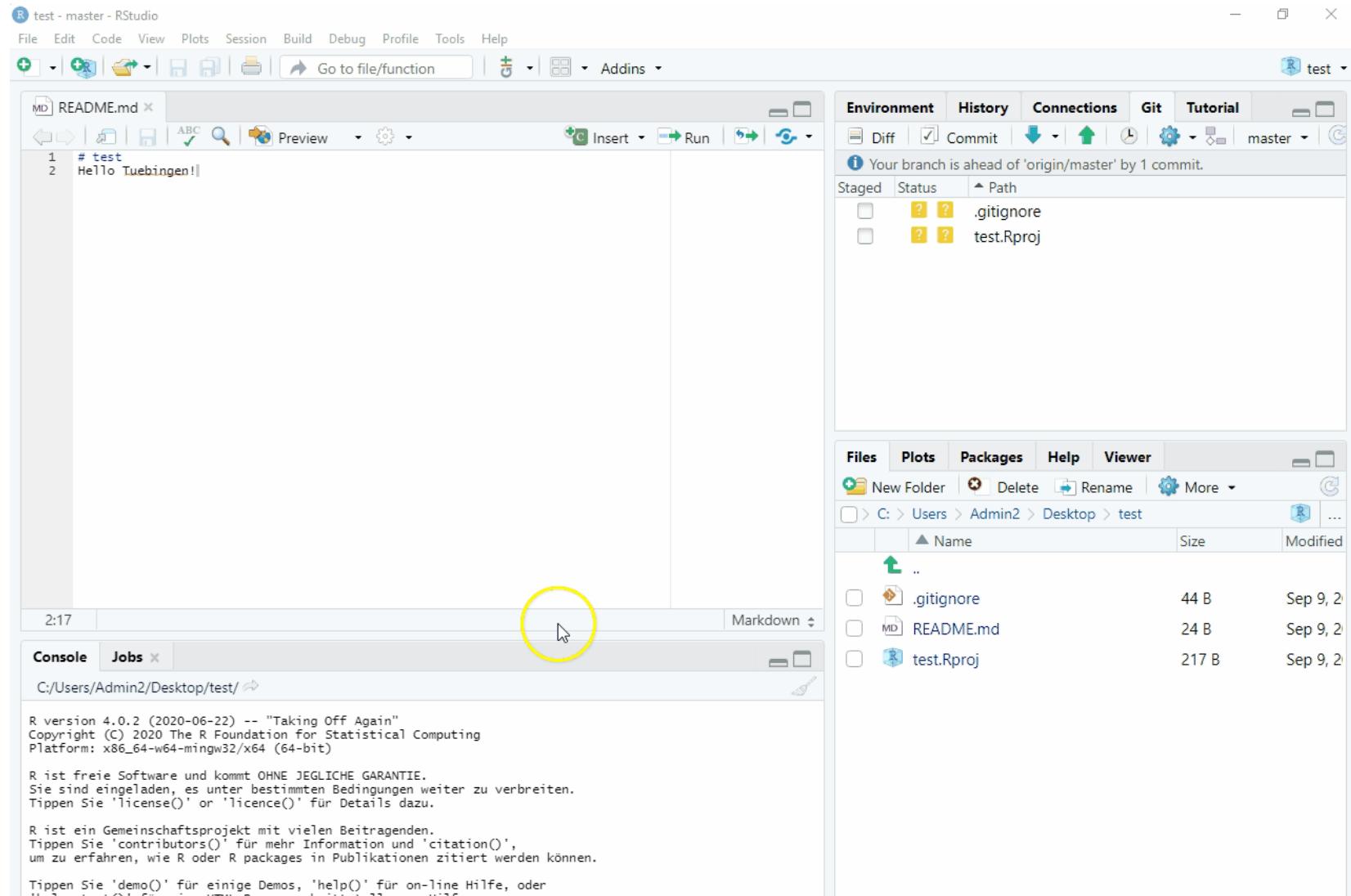
R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()', um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.

> |
```
- Environment Pane:** Shows the Global Environment section with the message "Environment is empty".
- Files Pane:** Shows the file structure in the 'test' directory:

Name	Size	Modified
..		Sep 9, 2
.gitignore	44 B	Sep 9, 2
README.md	14 B	Sep 9, 2
test.Rproj	217 B	Sep 9, 2

# Push project from RStudio to GitHub – a quick walkthrough



The screenshot shows the RStudio interface with a project titled "test" open. The left pane displays a file named "README.md" containing the text:

```
1 # test
2 Hello Tuebingen!!
```

The right pane shows the "Git" tab selected, indicating the branch is ahead of "origin/master" by 1 commit. The "Staged" section lists files to be committed:

Path
.gitignore
test.Rproj

The bottom pane shows the R console output:

```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()', um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder
```

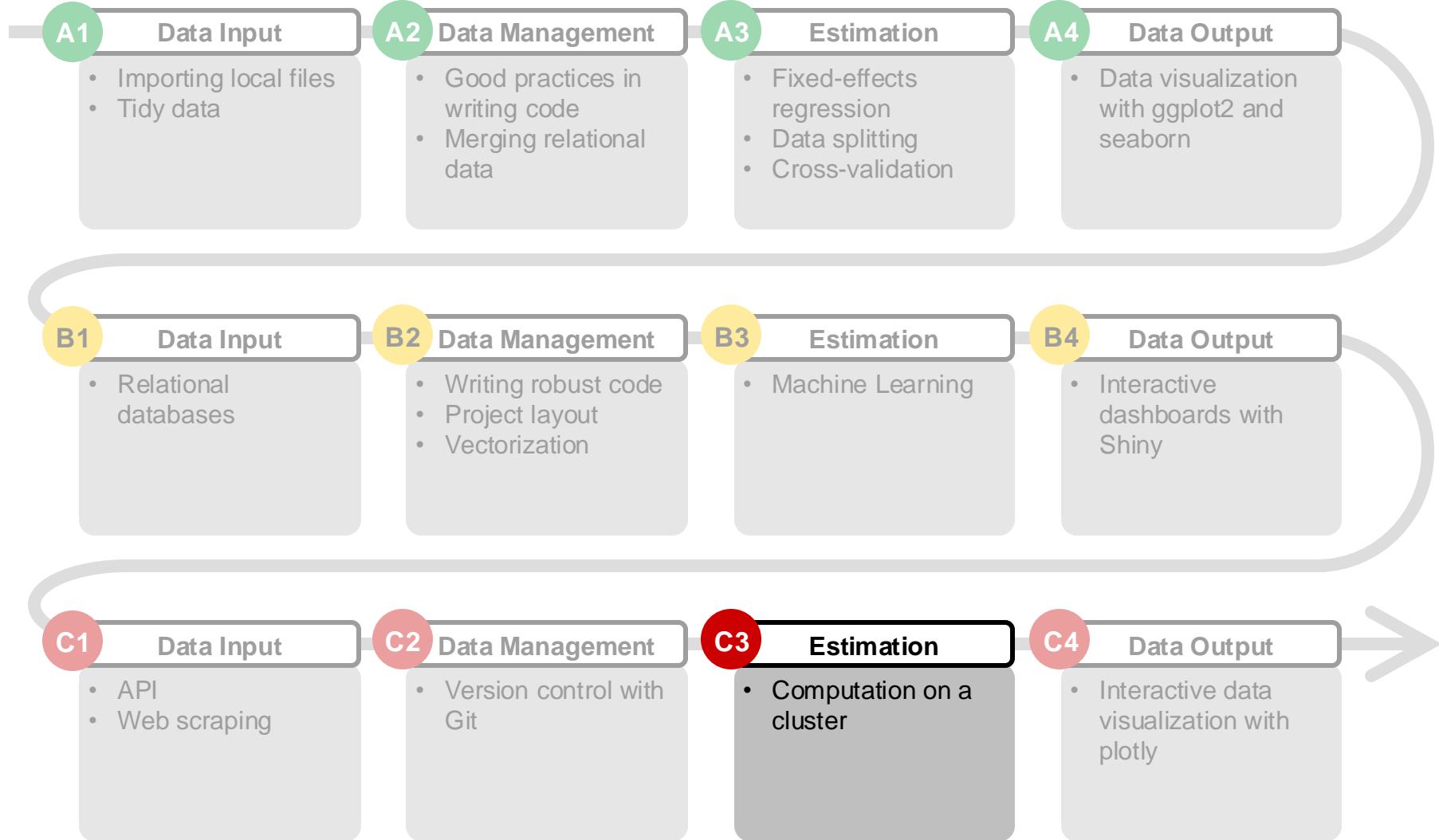
A yellow circle highlights the cursor in the R console area.

## Exercises



7. Create a new project with a README file on GitHub and again `pull` – modify – `stage` – `commit` – `pull` – `push` – ..., this time using the RStudio GUI to Git(Hub)

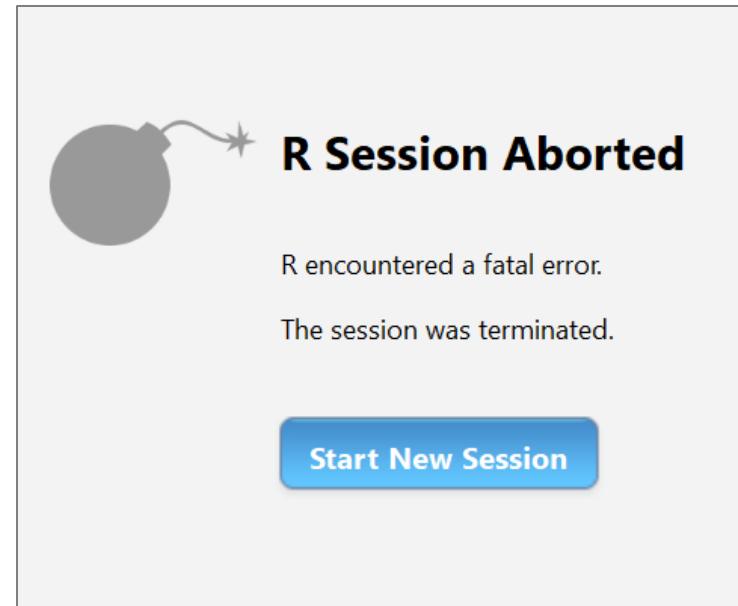
# Data Science Project Management: Course Outline



# Why do we need cluster computing?

## Limitations of R on local machine

- All R objects are stored in RAM
- Most computers have between 8 and 32 GB of RAM
- Many datasets are bigger than that
- Parallelization on local machine is possible, but the number of available cores/nodes is very limited



A computer cluster can alleviate these limitations!

As a network of computers, a computer cluster enables faster data processing

## What is a “cluster”? —

- A computer cluster is a network of multiple computers that are connected in order to provide a more powerful computing environment.
- Benefits:
  - Numerous processors
  - Large memory
  - Access to powerful GPUs
  - High reliability (failure-resistant)
  - ...
- Often intended to utilize parallel computing: Splitting a large task into smaller parts, run them on different processors “in parallel”

## Some terminology —

- “Nodes” = computers that make up a cluster
- “Jobs” = client programs running on a node

## Tübingen students can utilize the bwUniCluster 2.0 for their projects

- bwUniCluster 2.0 is Baden-Wuerttemberg's general high performance computing (HPC) cluster
- Consists of parallel computers with distributed memory
- Runs the cluster workload management package Slurm
  - Schedules, manages, monitors and reports cluster workloads
- Can be accessed remotely via Secure Shell protocol (SSH)



<https://uni-tuebingen.de/en/einrichtungen/zentrum-fuer-datenverarbeitung/dienstleistungen/server/computing/resources/bwunicluster-20/>  
[https://wiki.bwhpc.de/e/BwUniCluster\\_2.0\\_Hardware\\_and\\_Architecture](https://wiki.bwhpc.de/e/BwUniCluster_2.0_Hardware_and_Architecture) (last retrieved on October 20, 2024)

# Tübingen students can utilize the bwUniCluster 2.0 for their projects

	Compute nodes "Thin"	Compute nodes "HPC"	Compute nodes "IceLake"	Compute nodes "Fat"	GPU x4	GPU x8	IceLake + GPU x4	Login
<b>Number of nodes</b>	200 + 60	260	272	6	14	10	15	3
<b>Processors</b>	Intel Xeon Gold 6230	Intel Xeon Gold 6230	Intel Xeon Platinum 8358	Intel Xeon Gold 6230	Intel Xeon Gold 6230	Intel Xeon Gold 6248	Intel Xeon Platinum 8358	
<b>Number of sockets</b>	2	2	2	4	2	2	2	2
<b>Processor frequency (GHz)</b>	2.1 Ghz	2.1 Ghz	2.6 Ghz	2.1 Ghz	2.1 Ghz	2.6 Ghz	2.5 Ghz	
<b>Total number of cores</b>	40	40	64	80	40	40	64	40
<b>Main memory</b>	96 GB / 192 GB	96 GB	256 GB	3 TB	384 GB	768 GB	512 GB	384 GB
<b>Local SSD</b>	960 GB SATA	960 GB SATA	1,8 TB NVMe	4,8 TB NVMe	3,2 TB NVMe	15 TB NVMe	6,4 TB NVMe	
<b>Accelerators</b>	-	-	-	-	4x NVIDIA Tesla V100	8x NVIDIA Tesla V100	4x NVIDIA A100 / 4x NVIDIA H100	-
<b>Accelerator memory</b>	-	-	-	-	32 GB	32 GB	80 GB / 94 GB	-
<b>Interconnect</b>	IB HDR100 (blocking)	IB HDR100	IB HDR200	IB HDR	IB HDR	IB HDR	IB HDR200	IB HDR100 (blocking)

<https://uni-tuebingen.de/en/einrichtungen/zentrum-fuer-datenverarbeitung/dienstleistungen/server/computing/resources/bwunicluster-20/>  
[https://wiki.bwhpc.de/e/BwUniCluster\\_2.0\\_Hardware\\_and\\_Architecture](https://wiki.bwhpc.de/e/BwUniCluster_2.0_Hardware_and_Architecture) (last retrieved on October 20, 2024)

# The workflow of the bwUniCluster 2.0 • an outline

## Setup at first use:

- 1 Apply for entitlement
- 2 Request user account on the registration server (+ questionnaire)
- 3 Install necessary software on your own computer
- 4 Connect to university network via VPN
- 5 Connect to the bwUniCluster via SSH client
- 6 Create workspace
- 7 Install R packages

## Workflow for every session:

- 1 Connect to university network via VPN
- 2 Connect to the bwUniCluster via SSH client
- 3 Transfer files using File Transfer Protocol software
- 4 Customize Slurm script
- 5 Start batch job

# First of all, you need to apply for entitlement

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- Open [ZDV website](#)
- Server services → Computing → Login → bwUniCluster → Click on [form](#)

### Apply for bwUniCluster Entitlement

First and Last Name \*

ZDV Login ID \*

ZDV E-Mail \* *Your university e-mail address*

Area of Expertise \* *e.g., “WiSo” or “Wiwi”*

Team \* *e.g., “Marketing”*

**Send**

- Submit your request

# Once your application is sent, wait until the request has been handled

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- Wait until you receive an e-mail from the "**ZDV-HPC Master**" arrives with the content „[...] from now on you can use the bw\*\*\*Cluster. [...]"
- Once you receive this mail, your entitlement is stored as an attribute of your login ID
- Usually this takes about 24 hours

# With the entitlement in place, you need to register for the service

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- Open <https://bwidm.scc.kit.edu/>
- Choose "Universität Tübingen" as your home institution
- Login with your Tübingen University ID and password
- Select "Register"
- Set up 2-factor authentication  
(See instructions on following slides.)
- Define (service-specific) password  
(It has to be different from your ZDV password!)
- Fill in the [questionnaire](#), otherwise you won't be able to login after 14 days!

The following services are available:

**bwUniCluster**

Der am Steinbuch Centre for Computing (SCC) des Karlsruher Institut für Technologie (KIT) betriebene bwUniCluster ist eines von mehreren zentralen Systemen für eine flächendeckende Grundversorgung der baden-württembergischen Universitäten und Hochschulen mit Hochleistungsrechnerkapazität.

» Service description  
» Register

**bwUniCluster**

Der am Steinbuch Centre for Computing (SCC) des Karlsruher Institut für Technologie (KIT) betriebene bwUniCluster ist eines von mehreren zentralen Systemen für eine flächendeckende Grundversorgung der baden-württembergischen Universitäten und Hochschulen mit Hochleistungsrechnerkapazität.

» Service description  
» Registry info  
» Set password

# 2-factor authentication ensures maximum security on the cluster

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

The figure consists of three vertically stacked screenshots of the bwUniCluster 2.0 User Access interface. The top screenshot shows the main dashboard with a 'My Tokens' button highlighted. The middle screenshot shows a 'Create new token here' section with 'New smartphone token' and 'New yubikey token' buttons. The bottom screenshot shows a 'Start' button in a 'Create new smartphone token' section.

- A second hardware device is needed to create a time-dependent six-digit-token that allows you to connect with the cluster

[https://wiki.bwhpc.de/e/BwUniCluster\\_2.0\\_User\\_Access/2FA\\_Tokens](https://wiki.bwhpc.de/e/BwUniCluster_2.0_User_Access/2FA_Tokens)

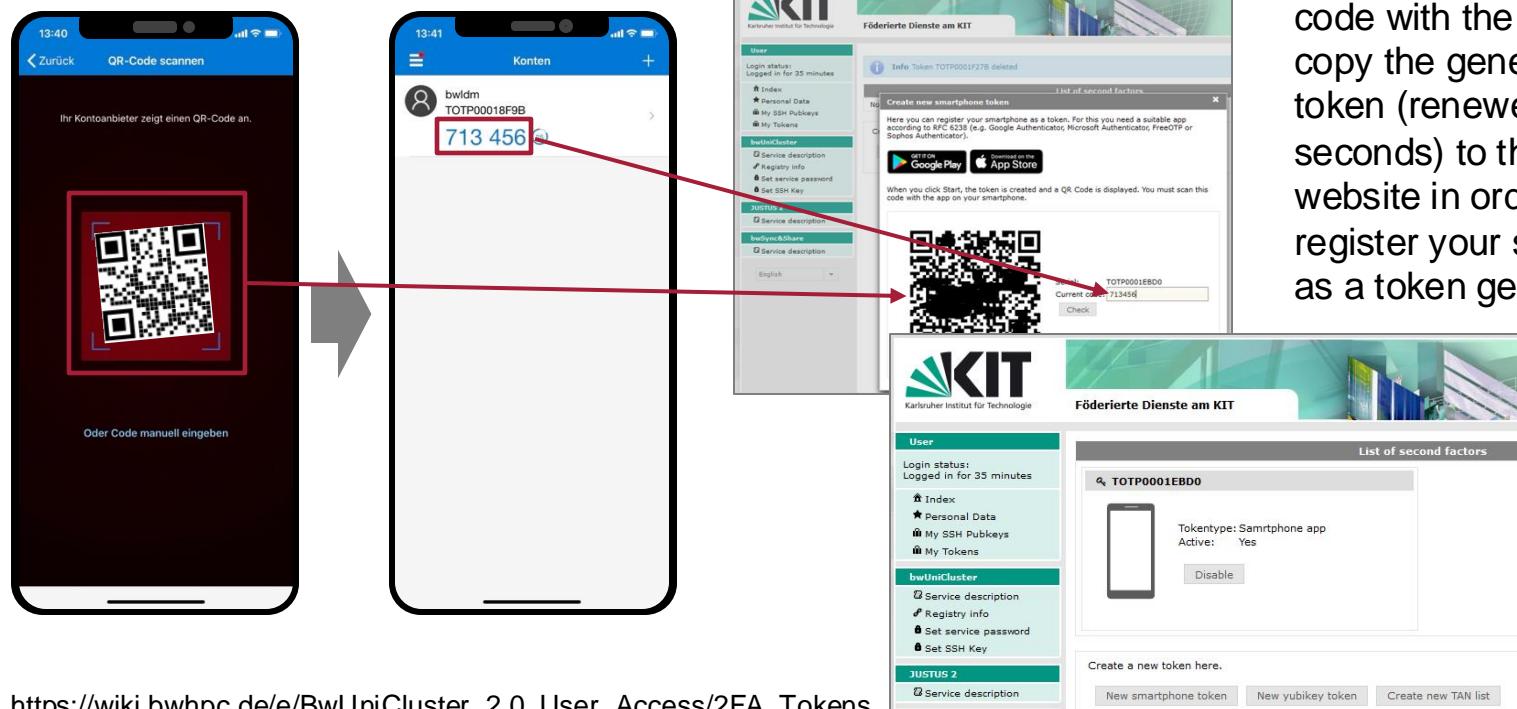
# Register a second device for 2-factor authentication

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

The most common solution is to use a mobile device (e.g., your smartphone) as a Software Token by installing one of the following Software Token apps:

- FreeOTP for [Android](#) or [iOS](#)
- Google Authenticator for [Android](#) or [iOS](#)
- Microsoft Authenticator for [Android](#), [iOS](#) or [Windows](#)
- LastPass Authenticator for [Android](#), [iOS](#) or [Windows](#)



Scan the (secret!) QR code with the app and copy the generated token (renewed every 30 seconds) to the KIT website in order to register your smartphone as a token generator.

# Register a second device for 2-factor authentication

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

## Backup TAN

- Make sure to create a **Backup TAN list**
  - You will need this in case you lose your second factor (e.g. your phone)
  - [Link](#) → Log in → ‘Index’ → ‘My Tokens’ → ‘CREATE NEW TAN LIST’
  - Physically print this list and store it in a safe place

# Install the necessary software on your own computer

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

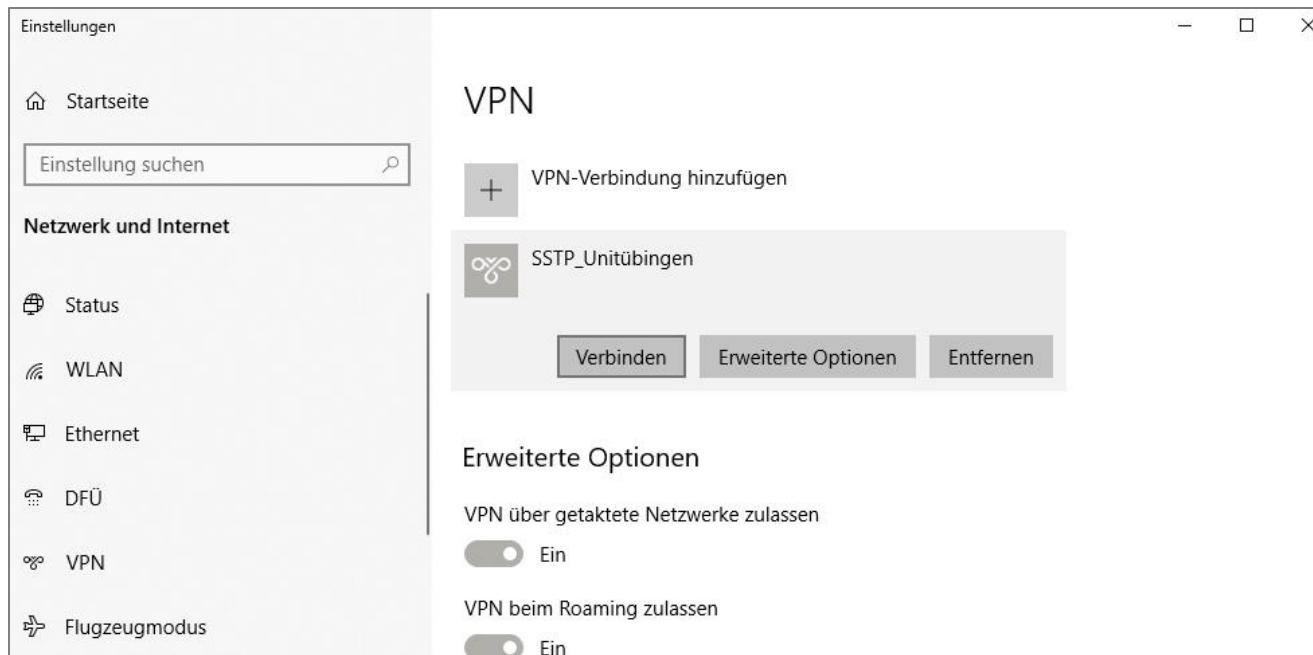
- Download and install the following programs:
  - SSH-client, e.g., PuTTY ([Download](#))  
(is used to establish secure and encrypted connections to the cluster)  
(on Linux and Mac you can skip this step and just use the Terminal)
  - File Transfer Protocol (FTP) client  
e.g., WinSCP ([Download](#)), FileZilla ([Download](#))  
(is used for secure and encrypted data transfer)

# Connect to university network via VPN

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- Make sure you are connected to the university network either via the university Wi-Fi or via VPN remote access (see [here](#) for a detailed explanation how to set up a VPN connection)

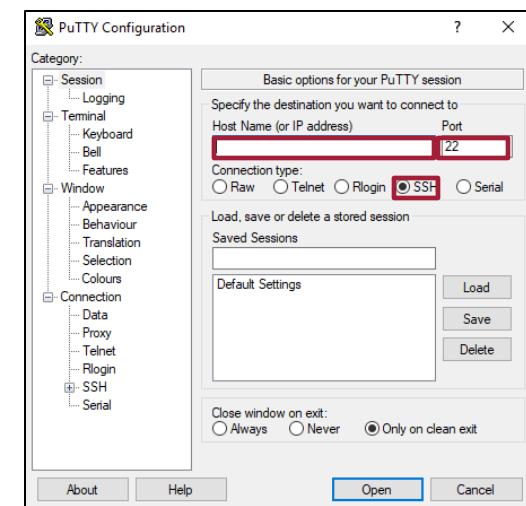


# Connect to the bwUniCluster 2.0 via SSH client (e.g., PuTTY)

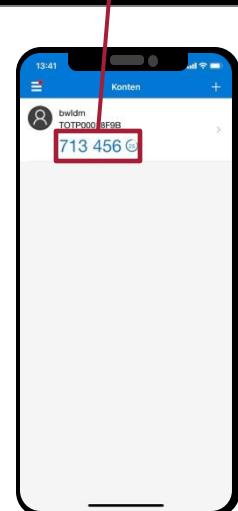
Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- Start PuTTY and select the following options :
  - Host Name: <UserID>@bwunicluster.scc.kit.edu  
(<UserID>): prefix „tu\_“ + ZDV-ID (e.g., tu\_wwxyz01)
  - Connection Type: SSH (default)
  - Port: 22
- Click on "Open" to establish connection
- Log in using 2-factor authentication



```
tu_wwzgr01@bwunicluster.scc.kit.edu
Using username "tu_wwzgr01".
Keyboard-interactive authentication prompts from server:
| Your OTP: 713456
```



```
tu_wwzgr01@bwunicluster.scc.kit.edu
Using username "tu_wwzgr01".
Keyboard-interactive authentication prompts from server:
| Your OTP: 713456
| Password:
```

**service-specific password**  
(Characters are not displayed when entered;  
confirm with *Enter*)

After successful login, the  
console window of the  
bwUniCluster 2.0 is displayed:

```
tu_wwzgr01@uc2n996:~
Using username "tu_wwzgr01".
Your OTP: 713456
Keyboard-interactive authentication prompts from server:
| Your OTP: 768406
| Password:
End of keyboard-interactive prompts from server
Last login: Sun Jul 19 10:13:53 2020 from uni-vpn119.vpn.uni-tuebingen.de
*****[REDACTED]*****
(KITE 2.0, RHEL 7.7, Lustre 2.12.3_ddn36)
wiki: https://wiki.bwhpc.de/e/bwUniCluster_2.0
ticket system: https://www.bwhpc.de/supportportal
email: bwunicluster@bwhpc.de
training: https://training.bwhpc.de
email: training@bwhpc.de
*****[REDACTED]*****
```

## Connect to the bwUniCluster 2.0 via the command line (Linux, Mac)

## Setup at first use:

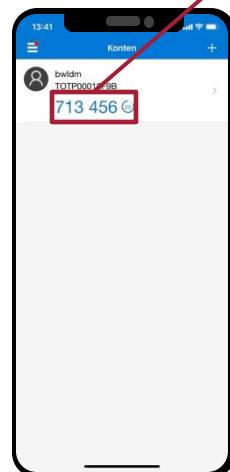
- 1 2 3 4 5 6 7

- Open a command-line Terminal on your local machine
  - Connect to the Cluster using the `ssh` command:

```
$ ssh <UserID>@bwunicluster.scc.kit.edu
```

- (<UserID>): prefix „tu\_“ + ZDV-ID (e.g., tu\_wwxyz01)
  - Log in using 2-factor authentication

```
$ ssh tu_wwxyz01@bwunicluster.scc.kit.edu
$ Your OTP: 713456
```



```
$ ssh tu_wwxyz01@bwunicluster.scc.kit.edu
$ Your OTP: 713456
$ Password:
```

## service-specific password

(Characters are not displayed when entered  
confirm with *Enter*)

After successful login, the console window of the bwUniCluster 2.0 is displayed:



# Create a workspace

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- Enter in PuTTY console:

```
$ ws_allocate <directory name> <duration in days>
```

e.g.: "ws\_allocate first\_test 45"

(creates a workspace directory with the name "first\_test" that will exist for 45 days.)

- Display path of all existing workspaces:

```
$ ws_list
```

e.g.: /pfs/work2/workspace/scratch/tu\_wwxyz01-first\_test-0

→ Path required later in WinSCP

- You should use workspaces instead of the home directory, because there is much more storage space available!
- The lifetime of a workspace can be extended three times for up to 60 days using:

```
$ ws_extend <directory name> 60
```

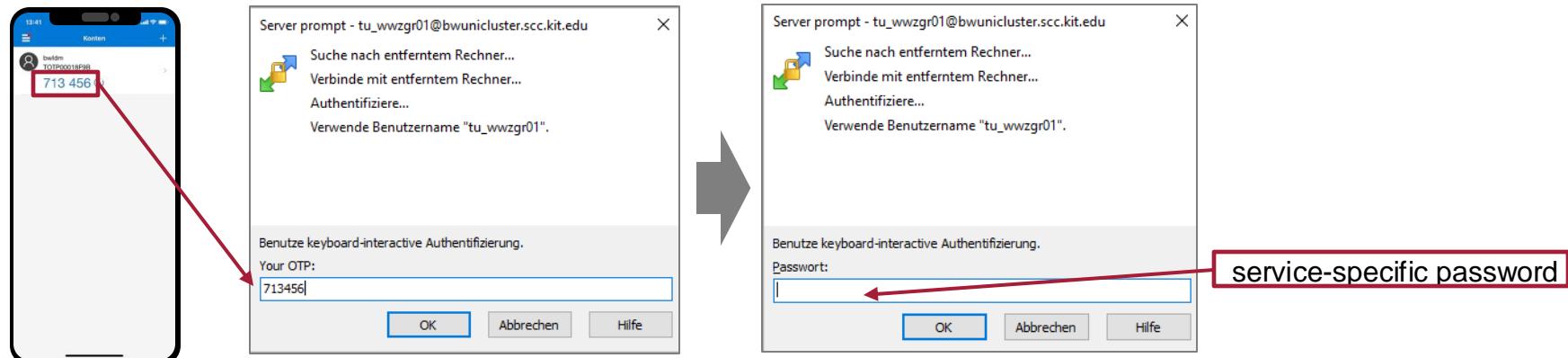
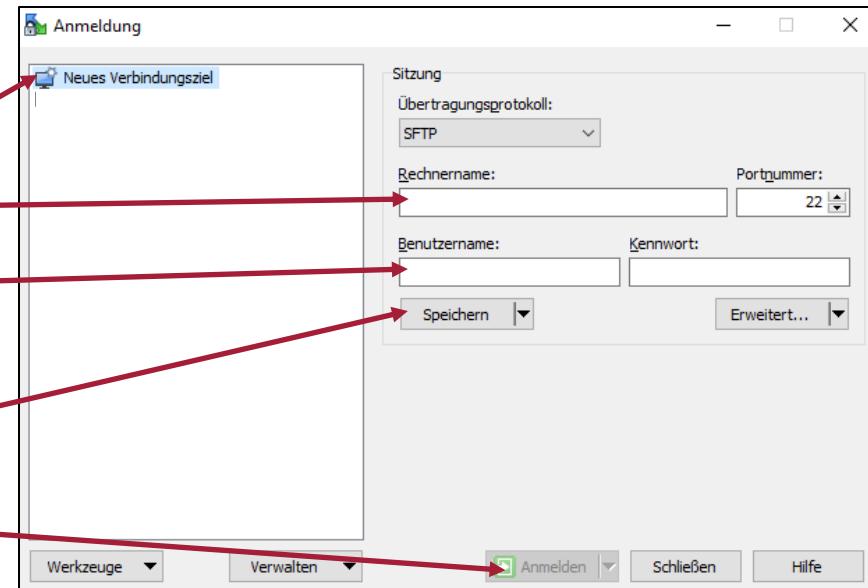
- A new workspace can be created at any time during or after the lifetime of the existing one.

# Transfer files via File Transfer Protocol (FTP) software (e.g., WinSCP)

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- Start WinSCP
- Select "New Site" on the left
- Host name: **uc2.scc.kit.edu**
- User name: ZDV-ID with “tu\_”-prefix
- Click on "Save" to save the computer name and user name.
- Click on “Login” to continue
- Identify yourself with 2-factor authentication:



# Install R packages

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

→ start PuTTY/Terminal and establish a SSH connection

## Create directory for packages (once)

```
$ cp $HOME/.bashrc $HOME/.bashrc.backup
$ echo "export R_LIBS='${HOME}/R_LIBS'">> $HOME/.bashrc
$ source $HOME/.bashrc
$ mkdir $R_LIBS
```

- ← Make a backup copy of your bashrc
- ← Setting the environment variable R\_LIBS permanently in your bashrc
- ← Sourcing bashrc to make R\_LIBS available
- ← Create the R\_libs folder in your HOME directory

## Install packages (repeatedly)

```
$ module load math/R/4.1.2
$ R
> install.packages('package_name',
repos="http://cran.r-project.org")
```

- ← Loading the most recent R software module (call module avail math/R to check which version is available)
- ← Start an interactive R session
- ← Installing your R package and the dependencies



# Install Python packages

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

→ start PuTTY/Terminal and establish a SSH connection

## Load Python and install packages

```
$ module load devel/python

$ pip install --user [package]

$ python
```

- ← Loading the most recent Python software module (call `module avail devel/python` to check which version is available)
- ← Install a Python package using pip (setting `--user` is very important!)
- ← Start an interactive Python session

Note: packages are installed into `$HOME/.local/lib/python3.[X]/site-packages/`

# Alternatives to Loading Modules

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

## Conda and Virtual Environments

- On the cluster you are not permitted to install system packages that might be a requirement for certain R or Python packages
- In this case or if you want to separate workspaces and package installations, cluster support suggests using Conda
- Look at <https://wiki.bwhpc.de/e/Conda> for instructions

## JupyterHub

- There is also the possibility to use the convenient JupyterHub
  - With this, you can request resources, access, edit and run your R and Python scripts using the Browser
- Look at [https://wiki.bwhpc.de/e/Jupyter at SCC](https://wiki.bwhpc.de/e/Jupyter_at_SCC) for instructions

# Install R packages - troubleshooting

Setup at first use:

- 1
- 2
- 3
- 4
- 5
- 6
- 7

- For some packages, the first installation attempt will fail because the dependencies (other packages on which a package is based) are not automatically installed
- Often, the **second-last** ERROR message tells you which package is still missing
- Install these packages first, then try again
- E.g., some attempts may be necessary to install the package `{dplyr}`, e.g. previous installations of `{rlang}`, `{Rcpp}`, `{bindrcpp}`, `{glue}`, `{tibble}`, `{purr}` and `{tidyselect}` are required.
- After the installation, R can be closed with the command: `quit("yes")`
  
- For other problems, the support of the bwUniCluster is available:
  - Ticket System: <http://www.support.bwhpc-c5.de>
  - Email: [bwunicluster-hotline@lists.kit.edu](mailto:bwunicluster-hotline@lists.kit.edu)

## Exercises



1. Follow the instructions given on the slides and run your first job on the bwUniCluster 2.0. The initial setup includes the following steps:
  - Apply for entitlement
  - Wait up to 24 hours
  - Request user account on the registration server
  - Install necessary software on your own computer
  - Connect to the bwUniCluster via SSH client
  - Create workspace
  - Install R/Python packages

# The workflow of the bwUniCluster 2.0 - an outline

## Setup at first use:

- 1 Apply for entitlement
- 2 Request user account on the registration server (+ questionnaire)
- 3 Install necessary software on your own computer
- 4 Connect to university network via VPN
- 5 Connect to the bwUniCluster via SSH client
- 6 Create workspace
- 7 Install R packages

## Workflow for every session:

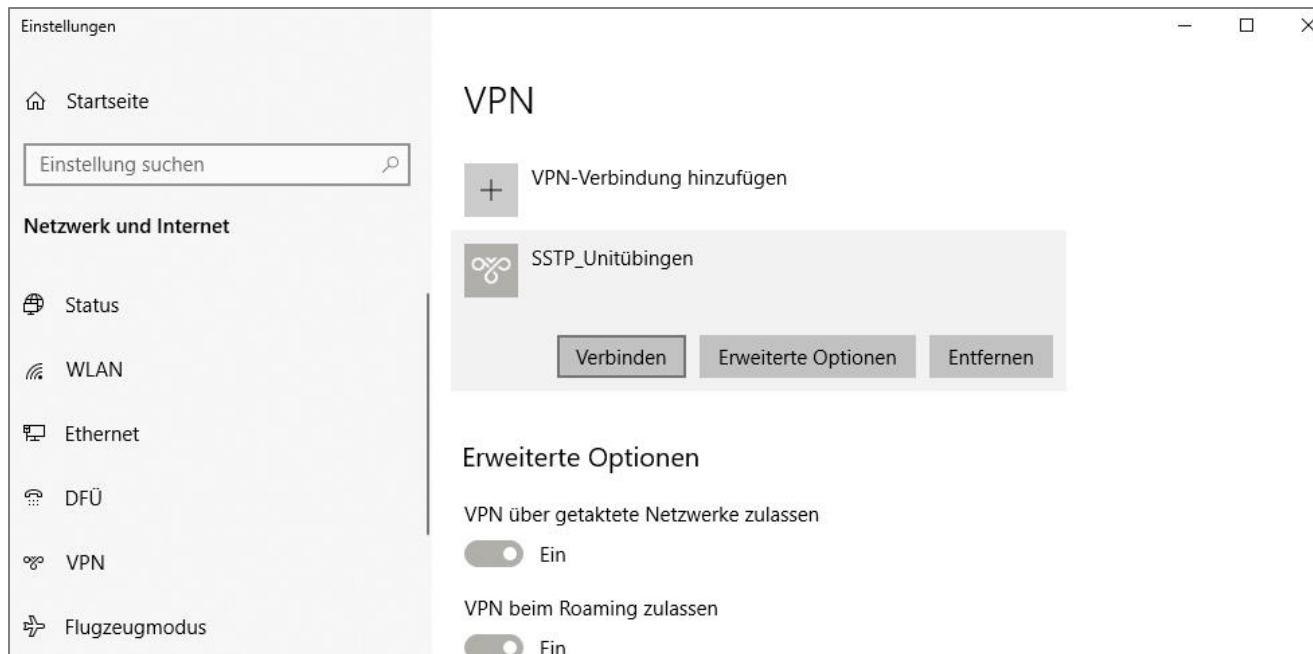
- 1 Connect to university network via VPN
- 2 Connect to the bwUniCluster via SSH client
- 3 Transfer files using File Transfer Protocol software
- 4 Customize Slurm script
- 5 Start batch job

# Connect to university network via VPN

Workflow for every session:

- 1
- 2
- 3
- 4
- 5

- Make sure you are connected to the university network either via the university Wi-Fi or via VPN remote access (see [here](#) for a detailed explanation how to set up a VPN connection)

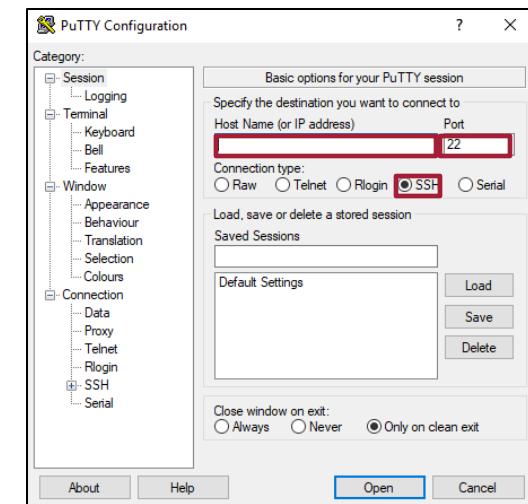


# Connect to the bwUniCluster via SSH client (e.g., PuTTY)

Workflow for every session:

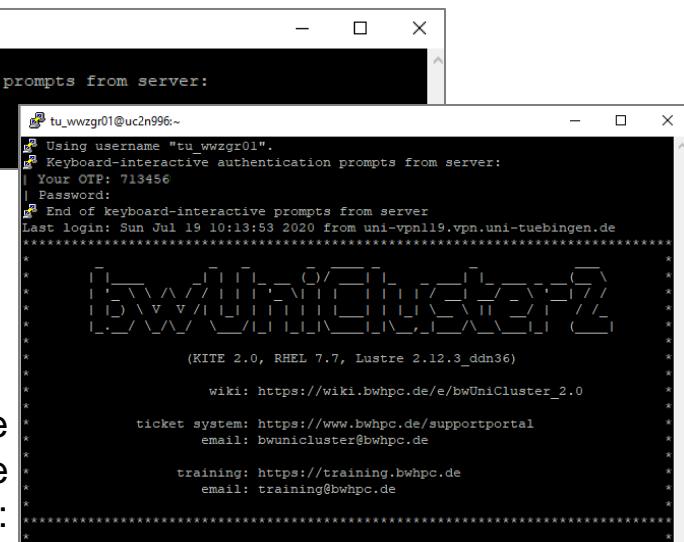
- 1
- 2
- 3
- 4
- 5

- Start PuTTY and select the following options :
  - Host Name: <UserID>@bwunicluster.scc.kit.edu  
(<UserID>): prefix „tu\_“ + ZDV-ID (e.g., tu\_wwxyz01)
  - Connection Type: SSH (default)
  - Port: 22
- Click on "Open" to establish connection
- Log in using 2-factor authentication



**service-specific password**  
(Characters are not displayed when entered;  
confirm with *Enter*)

After successful login, the  
console window of the  
bwUniCluster 2.0 is displayed:

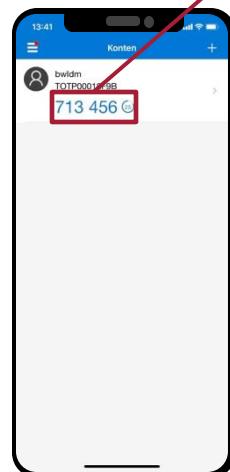


Connect to the bwUniCluster 2.0 via the command line (Linux, Mac)

## Workflow for every session:

- 1 2 3 4 5

- Open a command-line Terminal on your local machine
  - Connect to the Cluster using the `ssh` command:



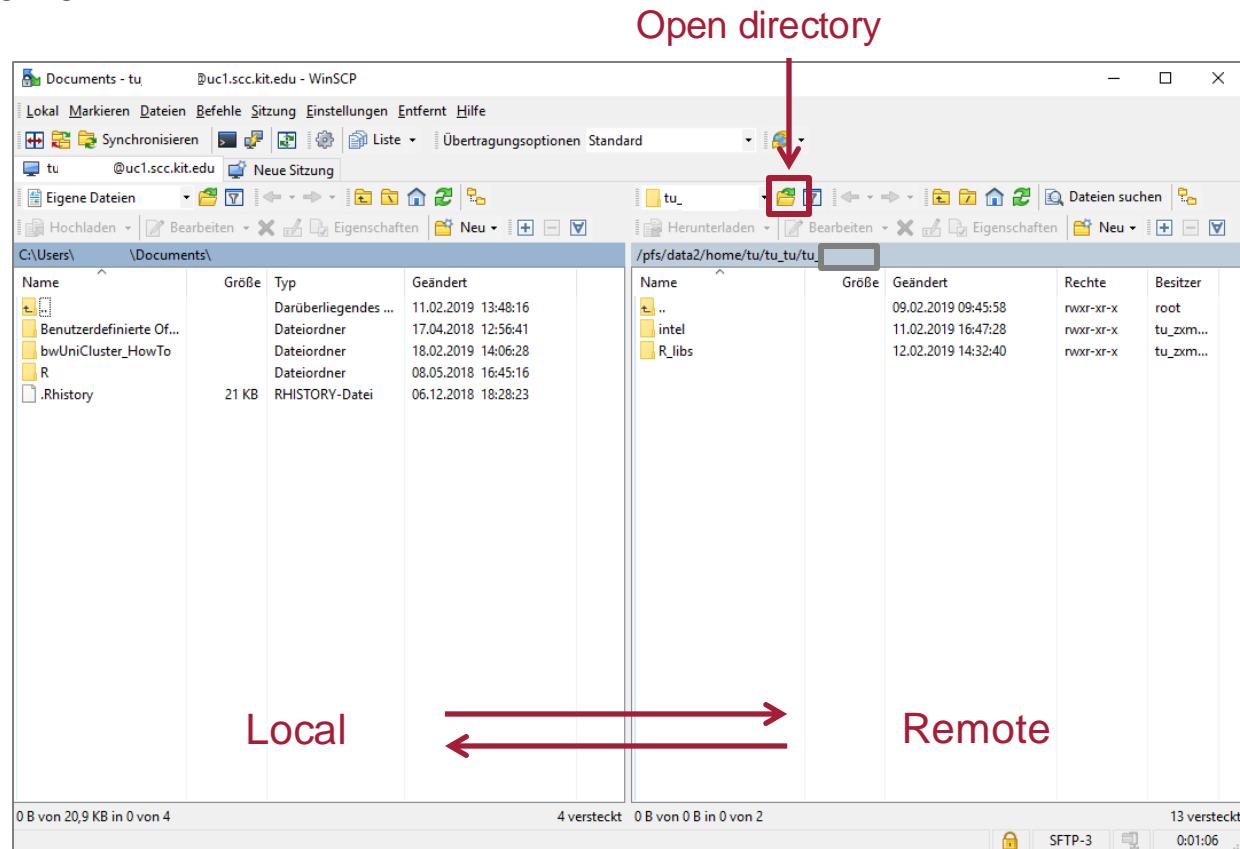
After successful login, the console window of the bwUniCluster 2.0 is displayed:

# Transfer files via File Transfer Protocol (FTP) software (e.g., WinSCP)

Workflow for every session:

- 1
- 2
- 3
- 4
- 5

- "Open directory", then enter the path identified with `ws_list` (in the example from above: `"/pfs/work2/workspace/scratch/tu_wwxyz01-first_test-0"`)
- Then transfer all required files (data sets, R script, Slurm script,...) from the local computer to the server.



# The Slurm script (\*.sh) tells the job scheduler what to do

Workflow for every session:

- 1
- 2
- 3
- 4
- 5

## File: slurm\_script\_example.sh

- Best Practice: Copy Slurm script template to server via WinSCP, open only then and adapt to requirements (keeps the correct formatting)
- Advanced alternative: Transfer using sftp application of your choice and reformat line breaks from Windows to Linux using `dos2unix` bash command

```
#!/bin/bash
```

```
#SBATCH --job-name=[YOUR_JOB_NAME].job
#SBATCH --output=[YOUR_OUTPUT_NAME].out
#SBATCH --export=ALL
```

Initialize your script, specify job name and output file name

Export environment variables from submission environment to job (default)

# Different computational resources are available in the different queues

Workflow for every session:

- 1
- 2
- 3
- 4
- 5

```
#SBATCH --partition=single
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=2000mb
#SBATCH --time=00:20:00
```

**dev\_single** → intended for test jobs, starts fast  
**single** → Standard, sufficient for many purposes  
**multiple** → Requests several nodes (further setup necessary)  
**fat** → "fat" nodes, more memory and processors available  
**gpu4** → Benefits e.g., Neural Networks strongly  
Various nodes at bwUniCluster:

- Many „thin“ nodes:
  - 96-192 GB memory, two 20-Core processors (40x)
  - Relatively short waiting times
- Only few "fat" nodes:
  - 3 TB memory, four 20-Core processors (80x)
  - Greater performance, but longer waiting times

# Different queues have distinct resource limitations

Workflow for every session:

- 1
- 2
- 3
- 4**
- 5

queue	node	default resources	minimum resources	maximum resources
dev_single	thin	time=10, mem-per-cpu=1125mb		time=30, nodes=1, mem=180000mb, ntasks-per-node=40, (threads-per-core=2) 6 nodes are reserved for this queue. Only for development, i.e. debugging or performance optimization ...
single	thin	time=30, mem-per-cpu=1125mb		time=72:00:00, nodes=1, mem=180000mb, ntasks-per-node=40, (threads-per-core)=2
dev_multiple	thin	time=10, mem-per-cpu=1125mb	nodes=2	time=30, nodes=4, mem=90000mb, ntasks-per-node=40, (threads-per-core=2) 8 nodes are reserved for this queue. Only for development, i.e. debugging or performance optimization ...
multiple	thin	time=30, mem-per-cpu=1125mb	nodes=2	time=72:00:00, mem=90000mb, nodes=128, ntasks-per-node=40, (threads-per-core=2)
dev_multiple_il	thin (IceLake)	time=10, mem-per-cpu=1950mb	nodes=2	time=30, nodes=8, mem=249600mb, ntasks-per-node=64, (threads-per-core=2) 8 nodes are reserved for this queue Only for development, i.e. debugging or performance optimization ...
multiple_il	thin (IceLake)	time=10, mem-per-cpu=1950mb	nodes=2	time=72:00:00, nodes=128, mem=249600mb, ntasks-per-node=64, (threads-per-core=2)
dev_gpu_4_a100	gpu4 (IceLake + A100)	time=10, mem-per-gpu=127500mb, cpu-per-gpu=16		time=30, nodes=1, mem=510000mb, ntasks-per-node=64, (threads-per-core=2)
gpu_4_a100	gpu4 (IceLake + A100)	time=10, mem-per-gpu=127500mb, cpu-per-gpu=16		time=48:00:00, nodes=9, mem=510000mb, ntasks-per-node=64, (threads-per-core=2)
gpu_4_h100	gpu4 (IceLake + H100)	time=10, mem-per-gpu=127500mb, cpu-per-gpu=16		time=48:00:00, nodes=5, mem=510000mb, ntasks-per-node=64, (threads-per-core=2)
fat	fat	time=10, mem-per-cpu=18750mb	mem=180001mb	time=72:00:00, nodes=1, ntasks-per-node=80, (threads-per-core=2)
dev_gpu_4	gpu4	time=10, mem-per-gpu=94000mb, cpu-per-gpu=10		time=30, nodes=1, mem=376000, ntasks-per-node=40, (threads-per-core=2) 1 node is reserved for this queue Only for development, i.e. debugging or performance optimization ...
gpu_4	gpu4	time=10, mem-per-gpu=94000mb, cpu-per-gpu=10		time=48:00:00, mem=376000, nodes=14, ntasks-per-node=40, (threads-per-core=2)
gpu_8	gpu8	time=10, mem-per-cpu=94000mb, cpu-per-gpu=10		time=48:00:00, mem=752000, nodes=10, ntasks-per-node=40, (threads-per-core=2)

[https://wiki.bwhpc.de/e/BwUniCluster2.0/Batch\\_Queues](https://wiki.bwhpc.de/e/BwUniCluster2.0/Batch_Queues) (last retrieved on Jan 18, 2023)

# Define the requirements that are suitable for your job

Workflow for every session:

- 1
- 2
- 3
- 4
- 5

**nodes:** Number of requested nodes

**cpus-per-task:** number of cores per process

**time:** Estimated runtime of the job [dd:hh:mm]

```
#SBATCH --partition=single
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=2000mb
#SBATCH --time=00:20:00
```

Two different options for requesting memory:

**--mem-per-cpu:** Memory per thread/process  
(total memory = mem-per-cpu \* cpus-per-task)

**--mem:** Memory of the entire job

Only one of the two possibilities should be specified; if both are specified, only the value of "mem" will be considered.

## Define when you want to receive notifications and include the R script name

Workflow for every session:

- 1
- 2
- 3
- 4
- 5

```
#SBATCH --mail-user=[your_name]@uni-tuebingen.de
#SBATCH --mail-type=BEGIN,END,FAIL
```

**--mail-user:** Your e-mail address for infos regarding your job

**--mail-type:** Events for which you want to be notified

```
Load module
```

```
module load math/R/4.1.2
```

Current/used version of R

```
Start R program
```

```
R CMD BATCH --no-save --no-restore [your_R_script].R
```

Name of your R script in  
the workspace

# Submit your Slurm script using the sbatch command

Workflow for every session:

- 1
- 2
- 3
- 4
- 5

- In PuTTY/Terminal, change the directory with "cd <workspace-Directory>" to the directory where your data, R/Python script and Slurm script are located, e.g.:

```
$ cd /pfs/work2/workspace/scratch/tu_wwwxx01-first_test-0
```

- Enter in PuTTY:

```
$ sbatch <Your_scriptname>.sh
```

- A job ID is assigned to the request
- The status of the job can be checked with

```
$ squeue
```

- The output files appear in the workspace:
  - "<YOUR-OUTPUT-NAME>.out" → Output file from bwUniCluster 2.0
  - <R-Script-Name>.Rout → contains R-specific notifications
  - Exported files produced by the R/Python script, if applicable (.RData, .csv, .jpg, ...)
- A job can be terminated with:

```
$ scancel <Job-ID>
```

You want to save the workspace after running the R script?

**Export functions for the end of the R script:** —————

Export single data frame:

```
save(data_frame_name, file = "export_filename.RData")
```

Export current workspace:

```
save.image("export_filename.RData")
```

Equivalent functions that work in R can cause problems here (e.g., `saveRDS()`).

# The more, the better? How many resources to apply for

- More than one node only necessary if calculation has been parallelized in script (R and Python usually calculate by default only on one processor)
- “fat” only necessary for jobs that require more than 90,000 MB memory
- Best practice: start with "single" and mem=2,000; if memory is not sufficient, slowly work up in steps to the maximum of 90,000 MB
- Why not just apply for a huge amount of resources, such that it will definitely be enough for the job?
  - The Slurm Workload Management System is responsible for scheduling the cluster. It distributes the resources of the cluster in such a way that the capacity is used as efficiently as possible.
  - The more resources you request, the longer you have to wait for your job to get done.
  - The resources requested in the past also influence your waiting time.
  - The participation costs of each university are calculated on the basis of how many resources it has requested over a period of time.

```
slurmstepd: error: Job 17966765 exceeded memory limit (4301896 > 4096000), being killed
slurmstepd: error: Exceeded job memory limit
slurmstepd: error: *** JOB 17966765 ON uc1n252 CANCELLED AT 2020-02-11T09:56:23 ***
```

# The more, the better? How many resources to apply for

## We can check the amount of resources that a completed job has used

1. Submit the Slurm script via PuTTY/Terminal as usual
  - Make sure you have defined a valid `--output` file name in the Slurm script
2. After successful completion, the output file appears in the workspace
3. Check out the output file. It contains all information about the resources used while executing the Slurm script:

```
===== JOB FEEDBACK =====

NodeName=uc2n376
Job ID: 21657072
Cluster: uc2
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 4
CPU Utilized: 11:59:13
CPU Efficiency: 96.61% of 12:24:28 core-walltime
Job Wall-clock time: 03:06:07
Memory Utilized: 17.60 GB
Memory Efficiency: 56.31% of 31.25 GB
```

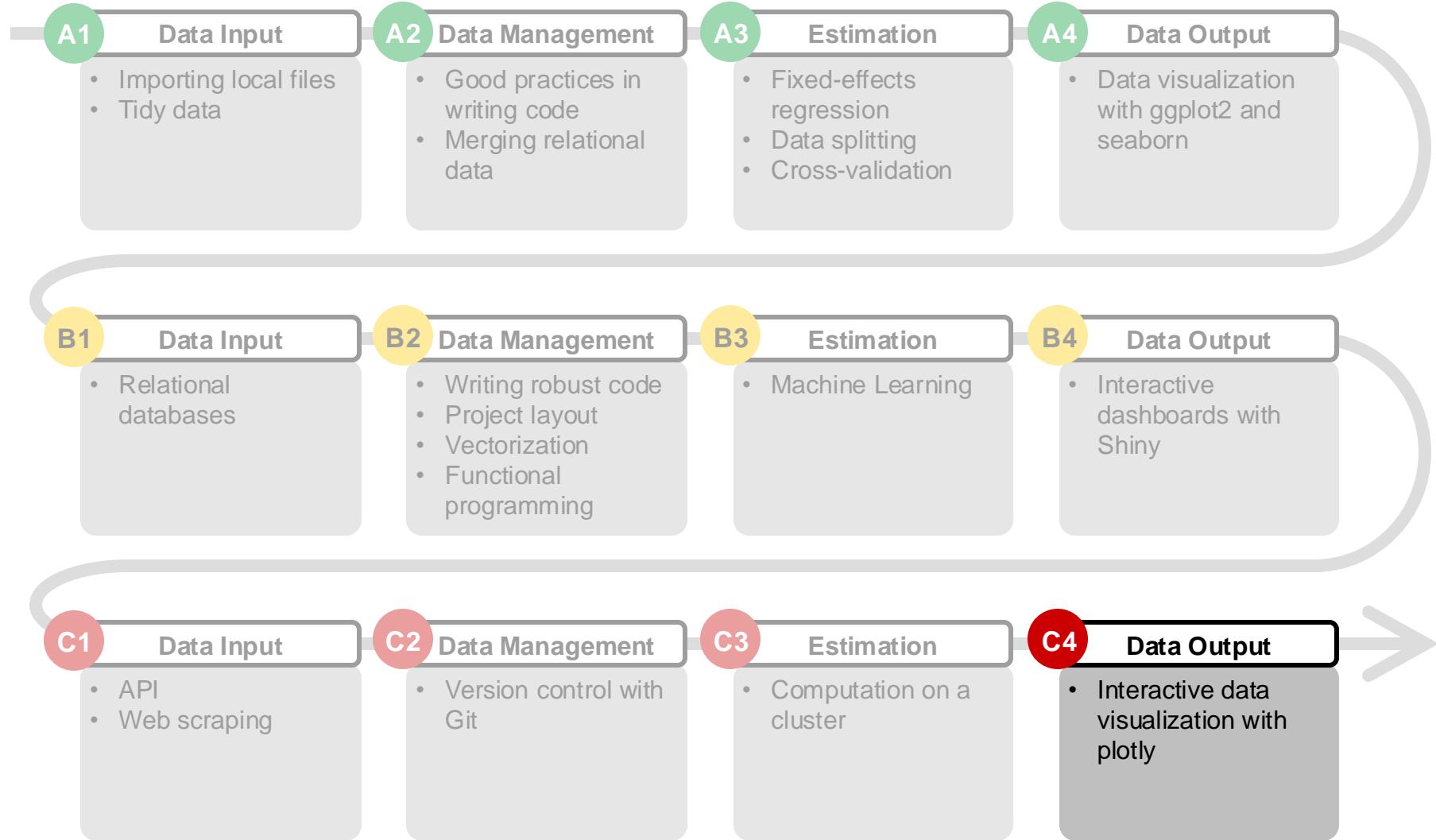
4. Use these values in order to adapt the resources *nodes*, *time* and *mem* in your Slurm script

## Exercises

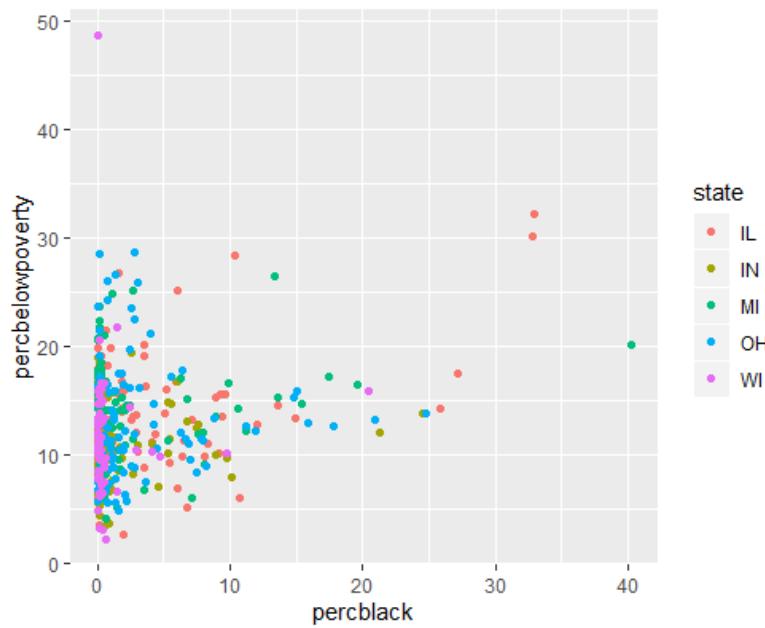


2. Write a simple script (e.g., code that creates and saves a `{ggplot2}` graphic) and execute this script on the bwUniCluster in the batch mode. The necessary steps include:
- Connect to the bwUniCluster via SSH client
  - Transfer files using File Transfer Protocol software
  - Customize Slurm script
  - Start batch job

# Data Science Project Management: Course Outline

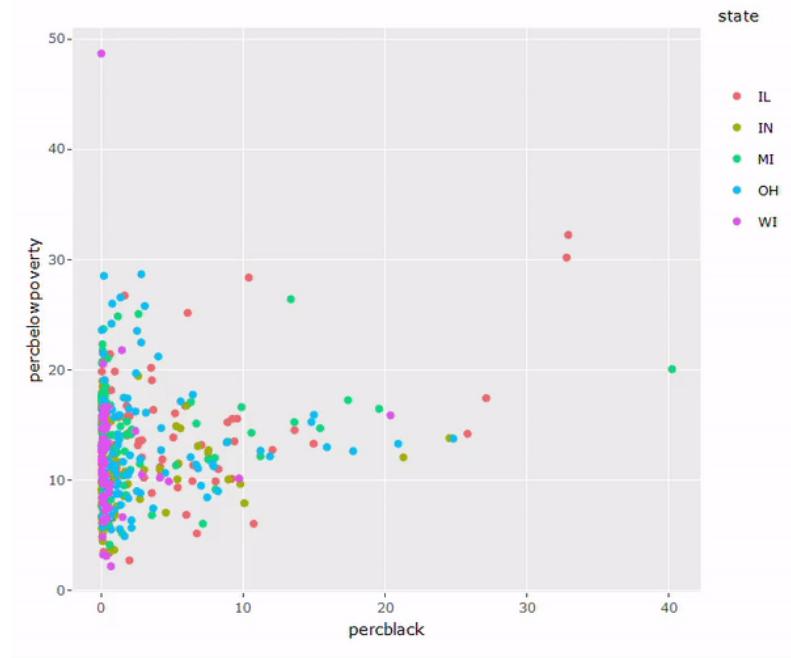


# Static vs. interactive graphs – choose the tool that suits your needs



## Static graphics:

- Remains permanently fixed
- Displays what the author wants to highlight
- Best option for printed materials



## Interactive graphs:

- Can be updated, subset, or drilled down to specific observations
- Allows for deeper and more interactive exploration of the data
- Suitable for graphics on webpages or dashboards

## Plotly is a JavaScript based library that produces interactive graphs

- The `{plotly}` packages for R and Python provide interfaces to the plotly JavaScript graphing library
- `{plotly}` allows you to build interactive web-based graphics



<https://plotly.com/r/> (last retrieved on August 05, 2024)

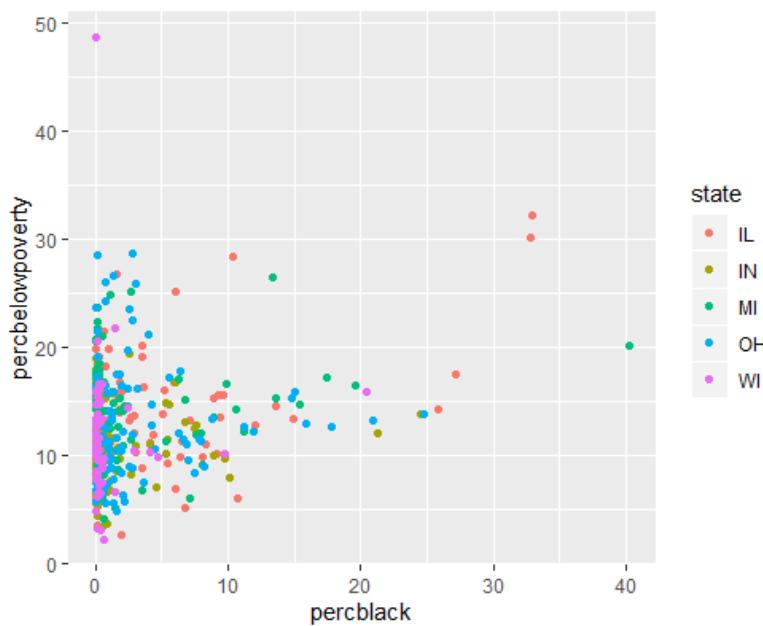


In this chapter we will be working with the `midwest` data included in `{ggplot2}`

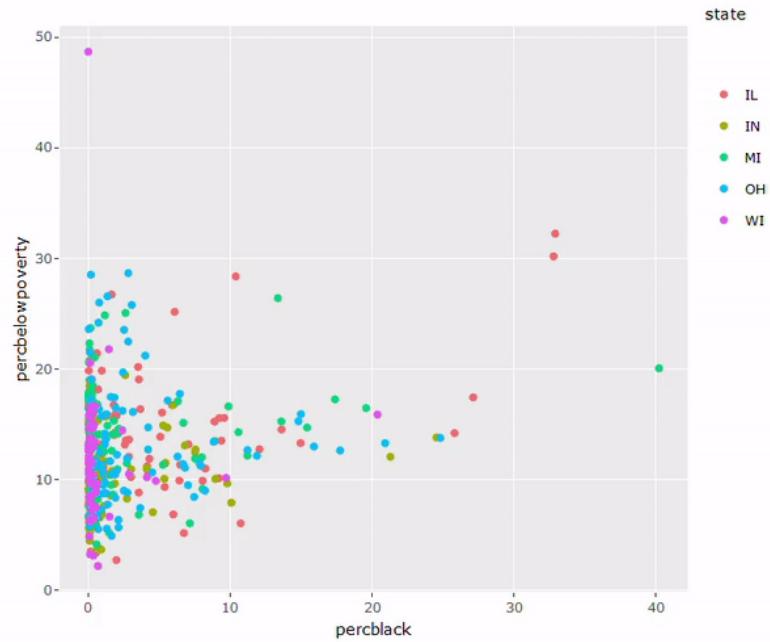
```
glimpse(midwest)

Observations: 437
Variables: 28
$ PID <int> 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 5...
$ county <chr> "ADAMS", "ALEXANDER", "BOND", "BOONE", "BROWN", "BU...
$ state <chr> "IL", "IL", "IL", "IL", "IL", "IL", "IL", "IL...
$ area <dbl> 0.052, 0.014, 0.022, 0.017, 0.018, 0.050, 0.017, 0....
$ poptotal <int> 66090, 10626, 14991, 30806, 5836, 35688, 5322, 1680...
$ popdensity <dbl> 1270.9615, 759.0000, 681.4091, 1812.1176, 324.2222, ...
$ popwhite <int> 63917, 7054, 14477, 29344, 5264, 35157, 5298, 16519...
$ popblack <int> 1702, 3496, 429, 127, 547, 50, 1, 111, 16, 16559, 8...
$ popamerindian <int> 98, 19, 35, 46, 14, 65, 8, 30, 8, 331, 51, 26, 17, ...
$ popasian <int> 249, 48, 16, 150, 5, 195, 15, 61, 23, 8033, 89, 36, ...
$ popother <int> 124, 9, 34, 1139, 6, 221, 0, 84, 6, 1596, 20, 7, 7, ...
$ percwhite <dbl> 96.71206, 66.38434, 96.57128, 95.25417, 90.19877, 9...
$ percbblack <dbl> 2.57527614, 32.90043290, 2.86171703, 0.41225735, 9...
$ percamerindan <dbl> 0.14828264, 0.17880670, 0.23347342, 0.14932156, 0.2...
$ percasian <dbl> 0.37675897, 0.45172219, 0.10673071, 0.48691813, 0.0...
$ percother <dbl> 0.18762294, 0.08469791, 0.22680275, 3.69733169, 0.1...
$ popadults <int> 43298, 6724, 9669, 19272, 3979, 23444, 3583, 11323, ...
$ perchsd <dbl> 75.10740, 59.72635, 69.33499, 75.47219, 68.86152, 7...
$ percollege <dbl> 19.63139, 11.24331, 17.03382, 17.27895, 14.47600, 1...
$ percpref <dbl> 4.355859, 2.870315, 4.488572, 4.197800, 3.367680, 3...
$ poppovertyknown <int> 63628, 10529, 14235, 30337, 4815, 35107, 5241, 1645...
$ percpovertyknown <dbl> 96.27478, 99.08714, 94.95697, 98.47757, 82.50514, 9...
$ percbelowpoverty <dbl> 13.151443, 32.244278, 12.068844, 7.209019, 13.52024...
$ percchildbelowpovert <dbl> 18.011717, 45.826514, 14.036061, 11.179536, 13.0228...
$ percadultpoverty <dbl> 11.009776, 27.385647, 10.852090, 5.536013, 11.14321...
$ percelderlypoverty <dbl> 12.443812, 25.228976, 12.697410, 6.217047, 19.2000...
$ inmetro <int> 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, ...
$ category <chr> "AAR", "LHR", "AAR", "ALU", "AAR", "AAR", "LAR", "A...
```

## You can convert from {ggplot2} to {plotly}



```
library(ggplot2)
static <- ggplot(midwest,
 aes(x = percbblack,
 y = percbelowpoverty,
 col = state)) +
 geom_point()
```



```
library(plotly)
interactive <- ggplotly(static)
```

Not all {ggplot2}-objects can be converted to {plotly} objects this easily. The converter from {matplotlib} to {plotly} is even more limited -- especially when working with {seaborn}. Hence, in the remainder of the chapter we will build interactive graphics from scratch.

# Plotting a single variable

## {plotly} syntax

- If you only want to plot a single kind of graphical object, you can use the high-level wrapper functions `plot_ly(type = "★")` and `px.*()` respectively
- Pass along the data and define the variable (use a formula in R)
- Notice how the plot type is defined differently in the examples below



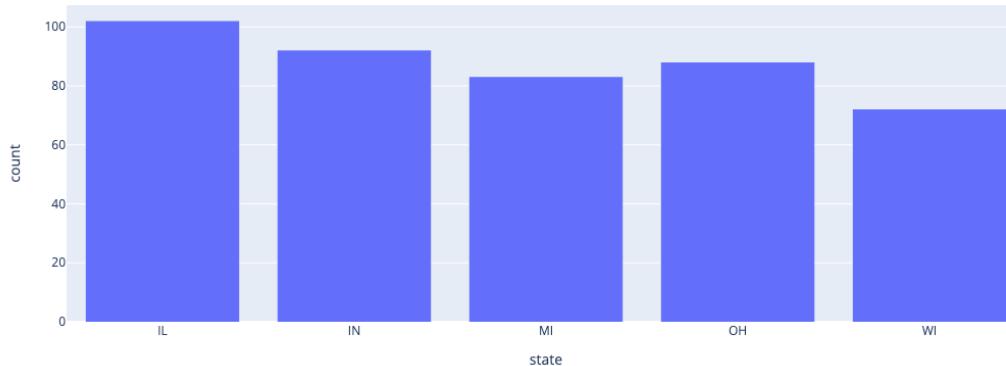
```
library(dplyr)
library(plotly)

plot_ly(midwest,
 x = ~ state,
 type = "histogram")
```



```
import pandas as pd
import plotly.express as px

px.histogram(midwest, x = "state")
```



# Plotting multiple variables – scatter boxplot

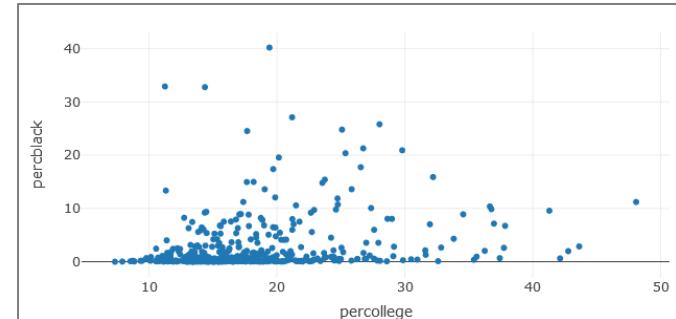
## Two continuous variables:



```
plot_ly(midwest,
 x = ~percollege,
 y = ~percblack,
 type = "scatter")
```



```
px.scatter(midwest,
 x = "percollege",
 y = "percblack")
```



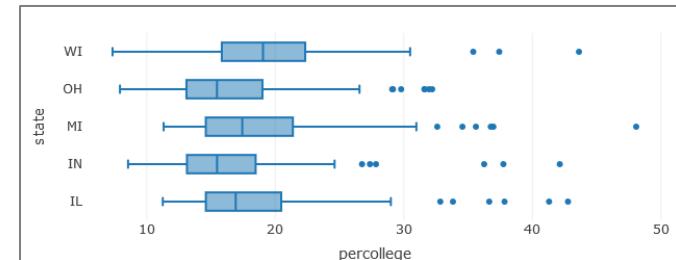
## Continuous and discrete variable:



```
plot_ly(midwest,
 x = ~percollege,
 y = ~state,
 type = "box")
```



```
px.box(midwest,
 x = "percollege",
 y = "state")
```



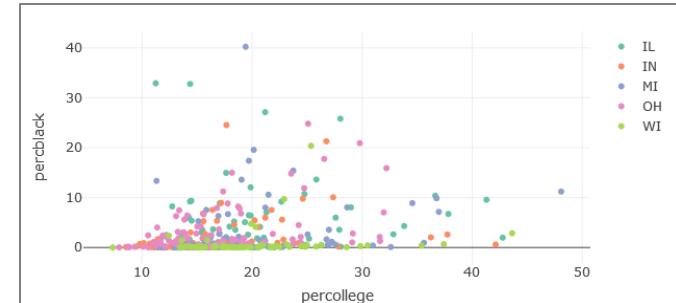
## More than two variables:



```
plot_ly(midwest,
 x = ~percollege,
 y = ~percblack,
 color = ~state,
 type = "scatter")
```



```
px.scatter(midwest,
 x = "percollege",
 y = "percblack",
 color = "state")
```



# Customizing Axes

## {plotly} customization

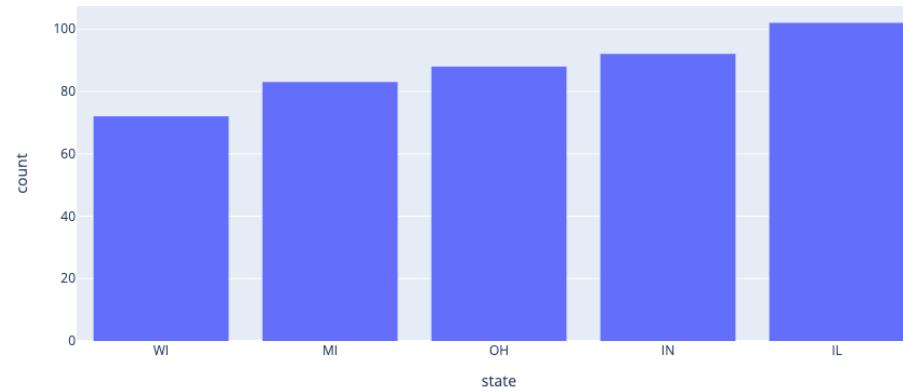
- If you want to edit the axes of a plot, you can do so after you have created and saved the plot
- In R, all customization is done within the `layout()` function
- In Python, use the `update_*` functions



```
fig %>% layout(xaxis = list(categoryorder = "total ascending"))
```



```
fig.update_xaxes(categoryorder = "total ascending")
```



## Exercises



1. Load `vgsales1.csv`, a data set on video game sales using the following pieces of code:



```
vgsales <- read_csv("data/vgsales1.csv",
 na = c("NA", "", "N/A", "tbd"))
```



```
vgsales = pd.read_csv("data/vgsales1.csv",
 na_values = ["NA", "", "N/A", "tbd"])
```

- a) Create a bar chart for a subset of games that have been released in 2009. The plot should display the number of games (x-axis) per genre (y-axis). Add a title and rename the x-axis appropriately.
- b) Change the plot from a) into a stacked bar chart where different colors indicate the platform on which a game has been released. Use the color palette *viridis*. (Hint: In Python choose the color palette using `color_discrete_sequence = px.colors.sequential.Viridis`)

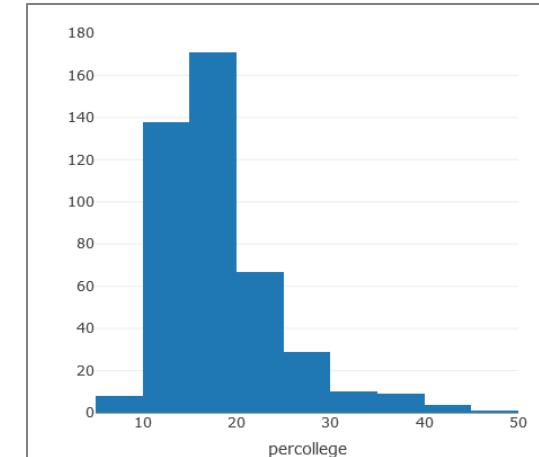
# The width of histogram buckets can be adjusted in two ways



```
plot_ly(midwest,
 x = ~ percollege,
 type = "histogram",
 nbinsx = 10)
```



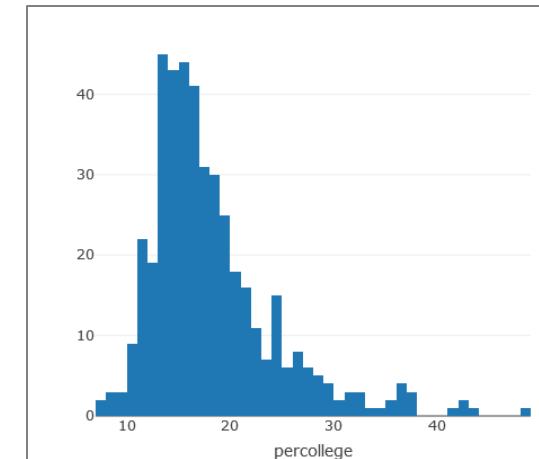
```
px.histogram(midwest,
 x = "percollege",
 nbins = 10)
```



```
plot_ly(midwest,
 x = ~ percollege,
 type = "histogram",
 xbins = list(start = 0,
 end = 50,
 size = 1))
```



```
fig = px.histogram(midwest,
 x = "percollege")
fig.update_traces(
 xbins = {"start": 0,
 "end": 50,
 "size": 1})
```



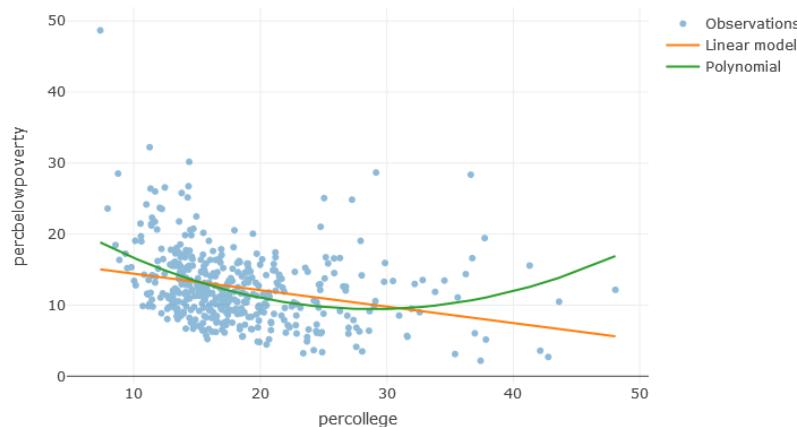
# Adding multiple traces to a plot: adding smoothers to a scatterplot

## {plotly} syntax

- If you want your plot to contain more than one kind of graphical object (“trace”), use the object-oriented approach of {plotly}

```
linearModel <- lm(percbelowpoverty ~ percollege, midwest)
quadraticModel <- lm(percbelowpoverty ~ poly(percollege, 2), midwest)

initialize a base object
plot_ly(midwest, x = ~percollege, y = ~percbelowpoverty) %>%
 # add traces (inherit aesthetics from base object)
 add_markers(name = "Observations", opacity = 0.3) %>%
 add_lines(y = fitted(linearModel), name = "Linear model") %>%
 add_lines(y = fitted(quadraticModel), name = "Polynomial")
```





# Adding multiple traces to a plot: adding smoothers to a scatterplot

## {plotly} syntax

- If you want your plot to contain more than one kind of graphical object (“trace”), use the object-oriented approach of {plotly}

```
import plotly.graph_objects as go
import statsmodels.formula.api as smf

make sure your values are sorted by the x-axis-variable
midwest_sorted = midwest.sort_values("percollege")

linearModel = smf.ols("percbelowpoverty ~ percollege", data = midwest_sorted).fit()
quadraticModel = smf.ols("percbelowpoverty ~ percollege+ I(percollege ** 2)",
 data = midwest_sorted).fit()

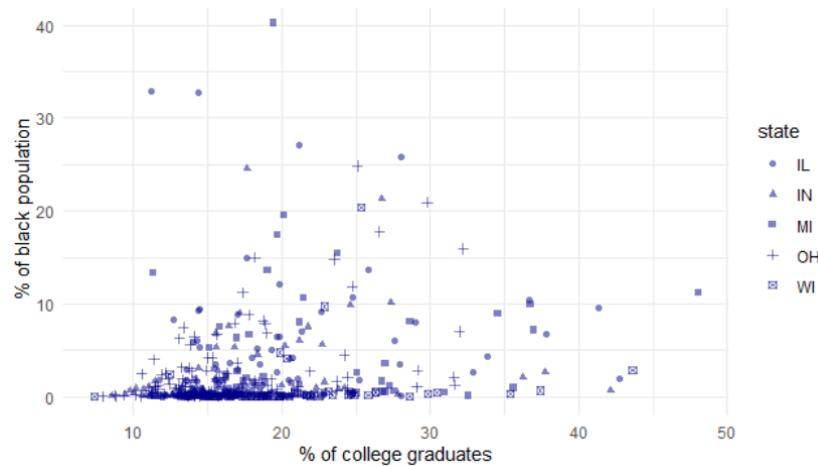
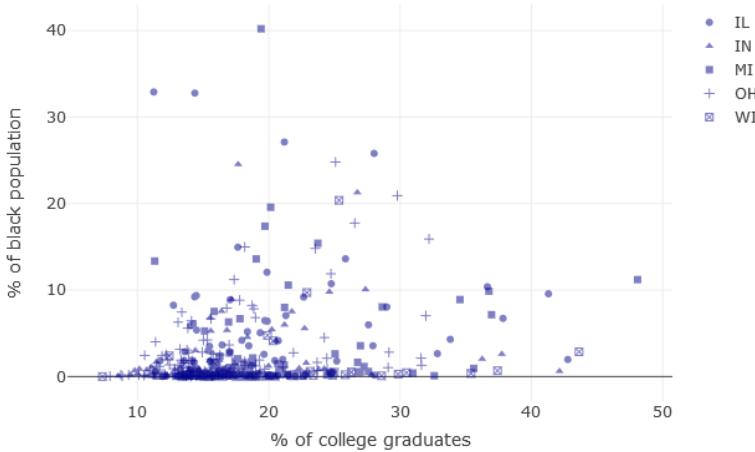
initialize a base object
fig = go.Figure()

add traces to the base object
fig.add_trace(go.Scatter(x = midwest["percollege"], y = midwest["percbelowpoverty"],
 mode = "markers", name = "Observations"))
fig.add_trace(go.Scatter(x = midwest_sorted["percollege"], y = linearModel.fittedvalues,
 mode = "lines", name = "Linear model"))
fig.add_trace(go.Scatter(x = midwest_sorted["percollege"], y = quadraticModel.fittedvalues,
 mode = "lines", name = "Polynomial"))
```

The structure of the `{plotly}` syntax is not entirely different from `{ggplot2}`

```
plot_ly(midwest,
 x = ~ percollege, y = ~ percblack,
 symbol = ~ state) %>%
add_markers(color = I("darkblue"),
 opacity = 0.5) %>%
layout(
 xaxis = list(
 title = "% of college graduates"),
 yaxis = list(
 title = "% of black population"))
```

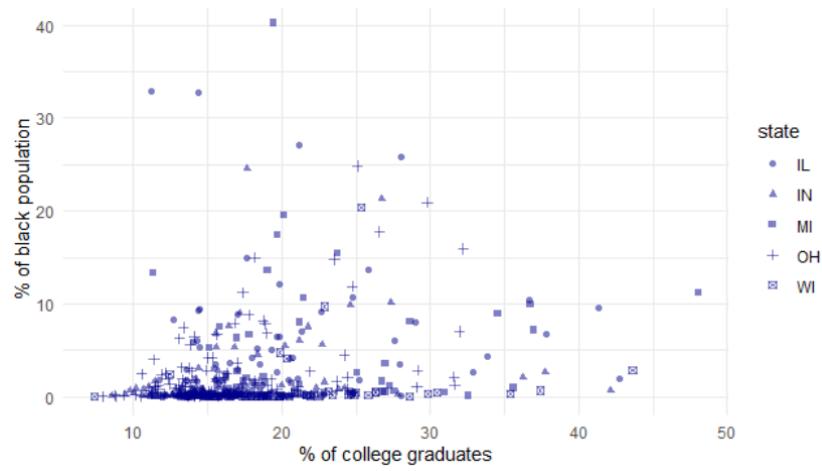
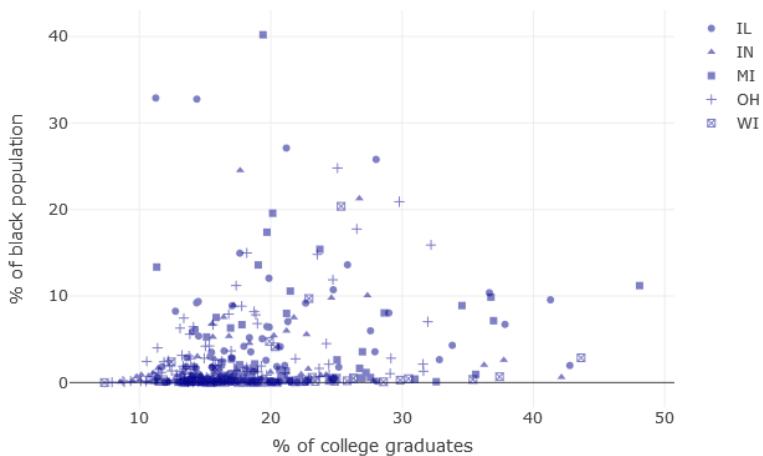
```
ggplot(midwest, aes(x = percollege,
 y = percblack,
 shape = state)) +
 geom_point(color = "darkblue",
 alpha = 0.5) +
 labs(x = "% of college graduates",
 y = "% of black population") +
 theme_minimal()
```



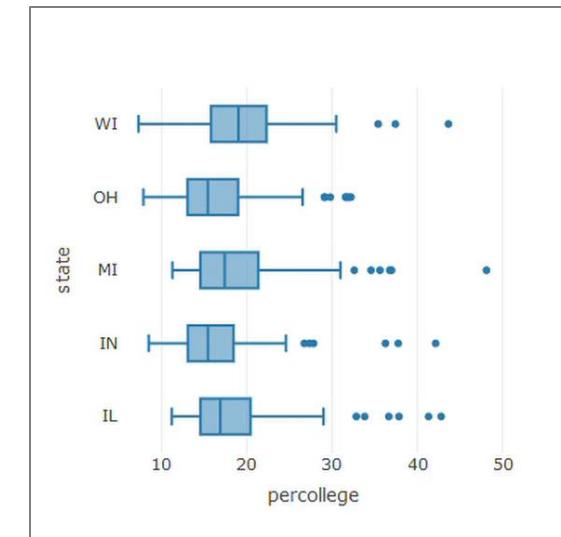
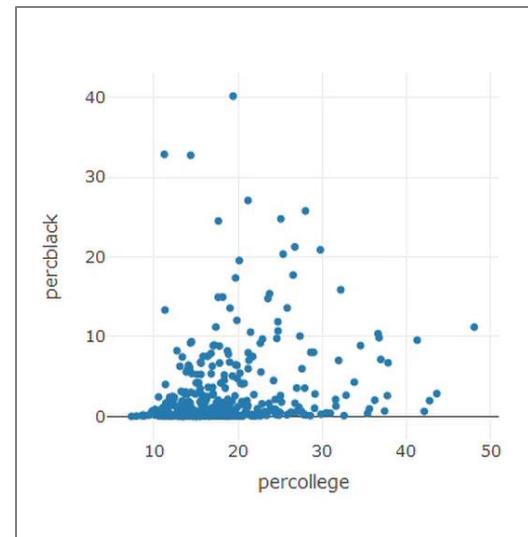
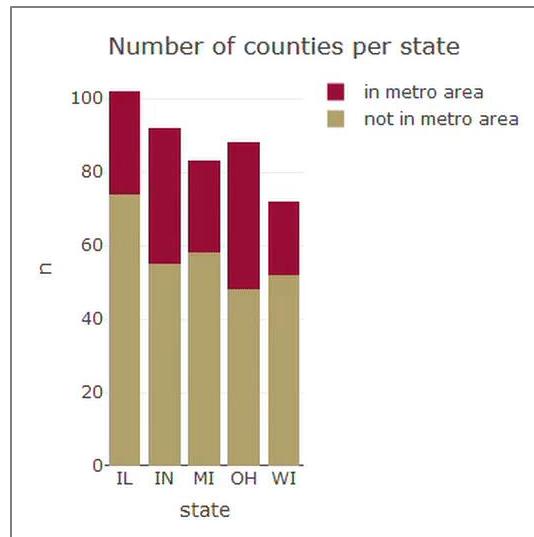
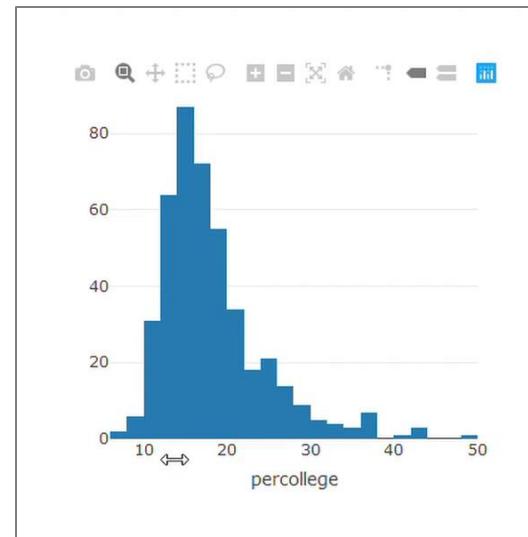
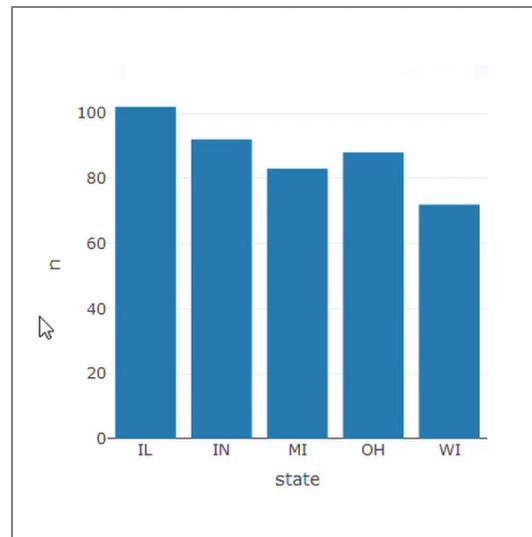
## We can draw many parallels to {ggplot2} code

```
plot_ly(midwest,
 x = ~ percollege, y = ~ percblack,
 symbol = ~ state) %>%
add_markers(color = I("darkblue"),
 opacity = 0.5) %>%
layout(
 xaxis = list(
 title = "% of college graduates"),
 yaxis = list(
 title = "% of black population"))
```

```
ggplot(midwest, aes(x = percollege,
 y = percblack,
 shape = state)) +
geom_point(color = "darkblue",
 alpha = 0.5) +
labs(x = "% of college graduates",
 y = "% of black population") +
theme_minimal()
```

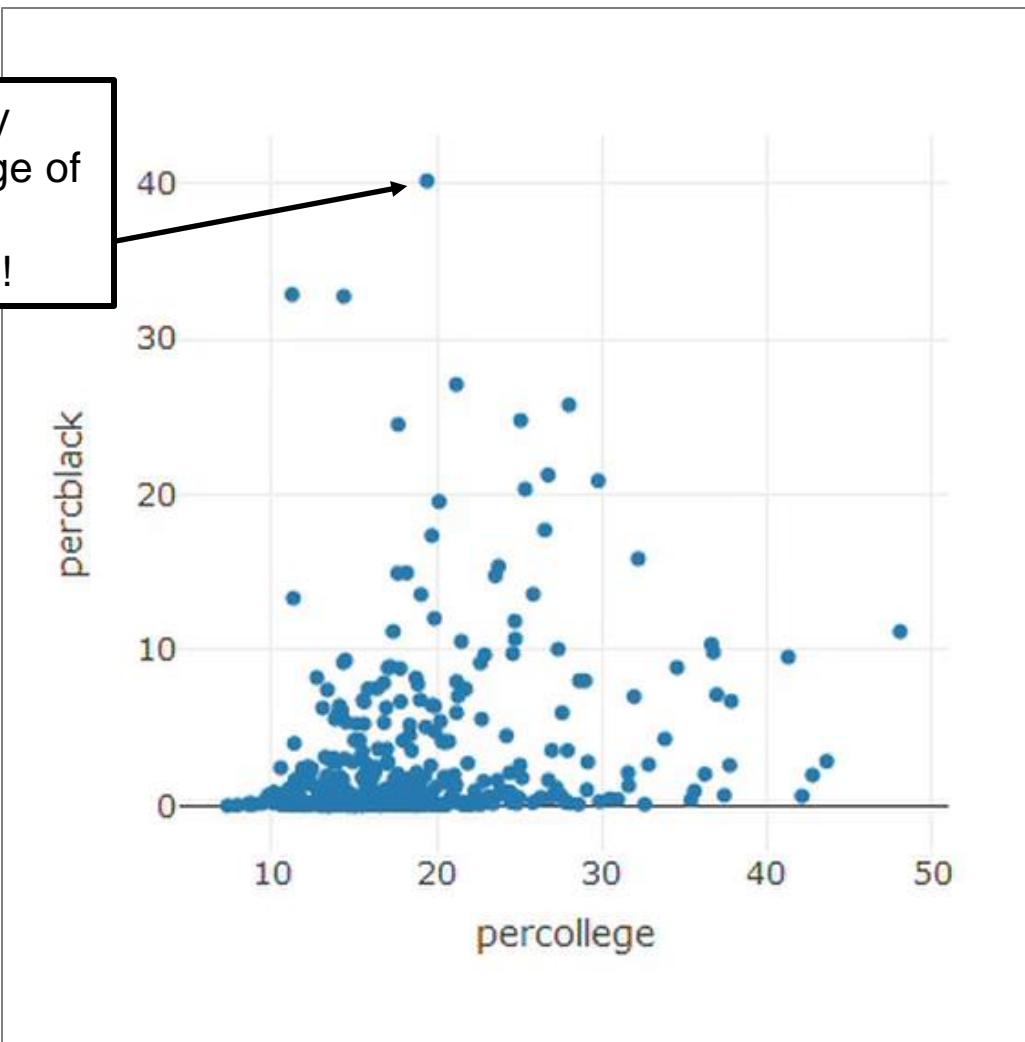


Per default, plotly shows the coordinates of observations in the hover info



## Adjustments to the default can be reasonable in some cases

What's the name of the county  
that has the highest percentage of  
black population by far?  
We cannot tell from this graph!

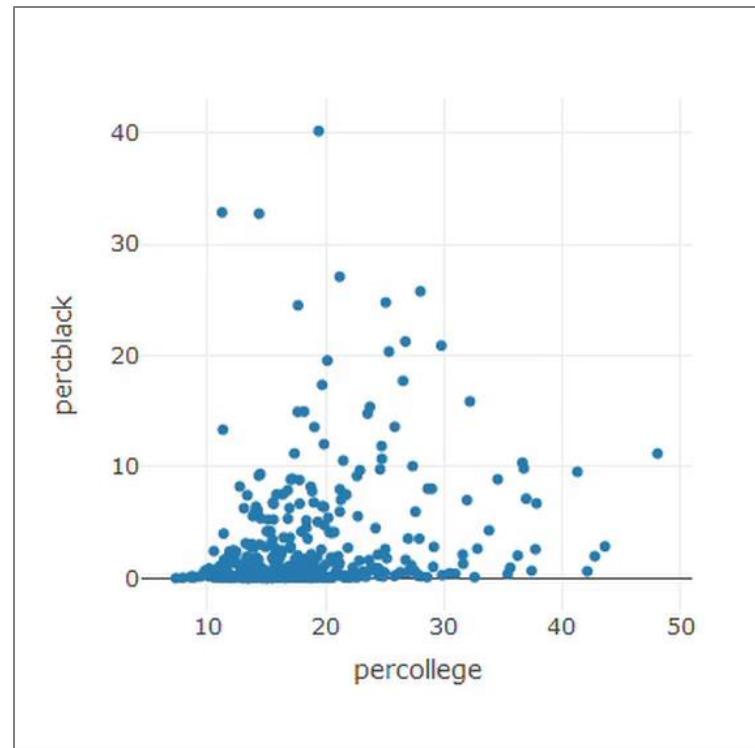


In R, the argument `hoverinfo` carries the relevant information

```
plot_ly(midwest,
 x = ~percollege,
 y = ~percblack,
 hoverinfo = ...,
 type = "scatter")
```

Valid values to the `hoverinfo` argument are:

- "all" (the default)
- "x+y"
- "x"
- "y"
- "none"
- "x+y+z" (for multidimensional graphs, such as 2D histograms)
- "text" (fully customizable text)



# We can fully customize the displayed information



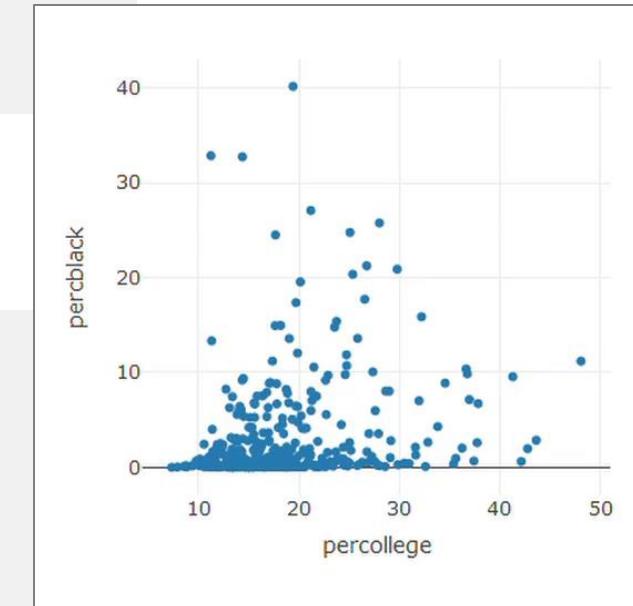
```
plot_ly(midwest,
 x = ~ percollege,
 y = ~ percblack,
 type = "scatter",
 hoverinfo = "text",
 text = ~ paste0("The share of black citizens in ", county,
 " is ", round(percblack, 2),
 "%.
 The percentage of college graduates is ",
 round(percollege, 2), "%.")
)
```



```
fig = px.scatter(midwest,
 x = "percollege", y = "percblack",
 # pass any additional data needed for hoverinfo
 custom_data = ["county"]
)

add the hoverinfo
fig.update_traces(hovertemplate = "".join([
 "The share of black citizens in %{customdata[0]} is %{y}%.
",
 "The percentage of college graduates is %{x}%."]))

```



## Exercises



2. In this exercise we want to display the airport delay data we worked with in the tutorial of chapter A2 on a map using {plotly}  
First, we repeat the data wrangling we did in chapter A2:

```
flights <- read_csv("data/flights.csv")
airports <- read_csv("data/airports.csv")

delays <-
flights %>%
group_by(dest) %>%
summarise(across(arr_delay, mean, na.rm = T), .groups = "drop")

airportsNew <- semi_join(airports, flights, by = c("faa" = "dest"))

avgDestDelays <- inner_join(airportsNew, delays, by = c("faa" = "dest"))
```

```
flights = pd.read_csv("data/flights.csv")
airports = pd.read_csv("data/airports.csv")

delays = flights.groupby("dest")["arr_delay"].mean()

airportsNew = airports[airports["faa"].isin(flights["dest"])]

avgDestDelays = pd.merge(airportsNew, delays, how = "inner", left_on = "faa", right_on = "dest")
```



## Exercises



2. In this exercise we want to display the airport delay data we worked with in the tutorial of chapter A2 on a map using {plotly}
  - a) Now create a {plotly} graphic that displays a map of the US with markers for the airport locations colored by their average arrival delay  
(Hints: In R use `plot_geo()` instead of `plot_ly()` to create a map. Per default, the graph shows the entire world. If you want to restrict the displayed portion of the earth to the USA, add `layout(geo = list(scope = "usa"))` to your code. In Python use `px.scatter_geo()` and define the `geo_scope = "usa"` in `fig.update_layout()`)
  - b) If we hover over the markers, the {plotly} graph shows us the exact coordinates of the respective airlines. However now we want to display the name of the respective airport when hovering over the marker. Can you identify the airport that is located off the southern coast of Florida in the extreme south-east of the USA?

# Scatterplot matrices are a powerful way to explore data

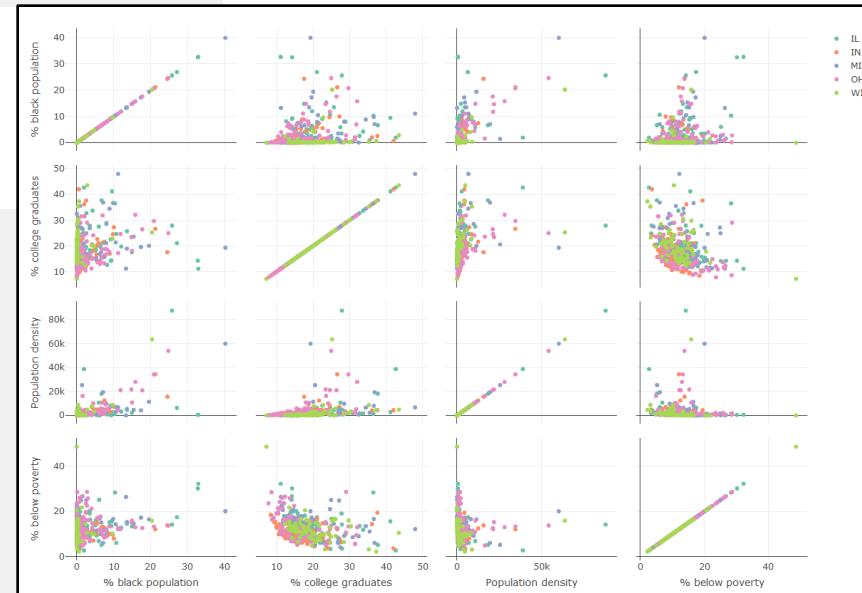


```
plot_ly(midwest,
 color = ~state,
 type = 'splom',
 dimensions = list(
 list(label = "% black population", values = ~perblack),
 list(label = "% college graduates", values = ~percollege),
 list(label = "Population density", values = ~popdensity),
 list(label = "% below poverty", values = ~percbelowpoverty)
)
)
```

Note that the subplots are linked to each other. Points that you highlight in one subplot using the lasso tool are also highlighted in all other plots!



```
px.scatter_matrix(midwest,
 color = "state",
 dimensions = ["perblack", "percollege",
 "popdensity", "percbelowpoverty"],
 labels = {"perblack": "% black population",
 "percollege": "% college graduates",
 "popdensity": "Population density",
 "percbelowpoverty": "% below poverty"})
```

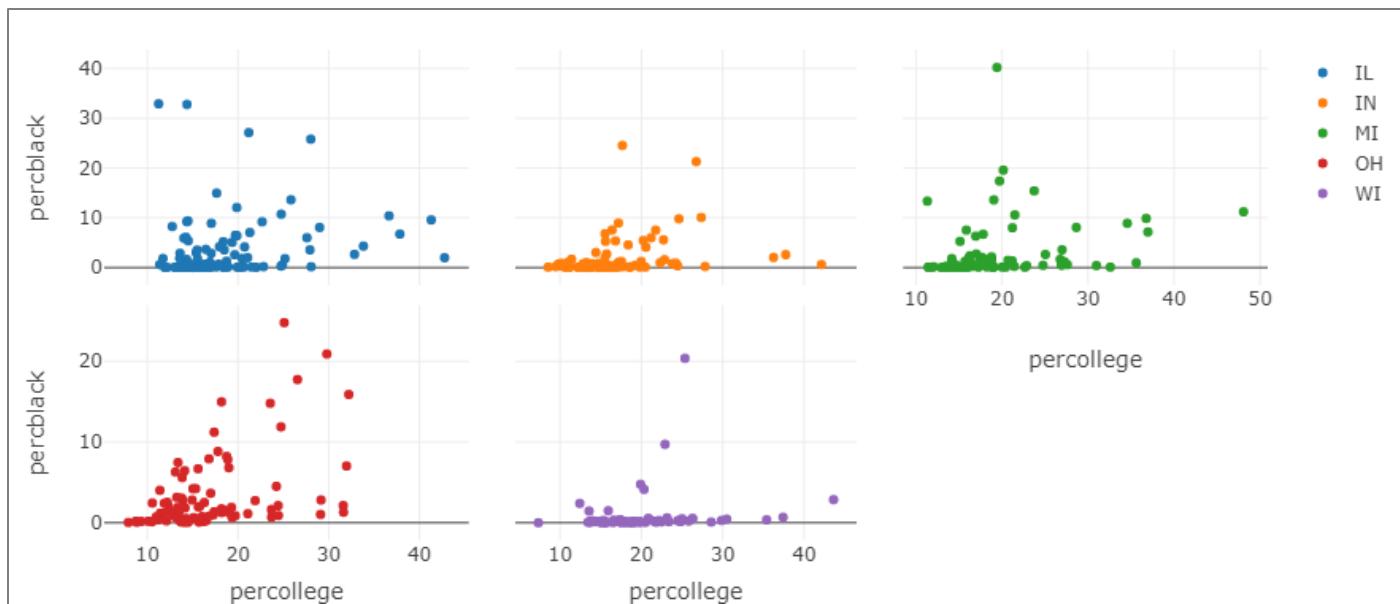


[https://rpubs.com/dvdgrmmngr/splom\\_midwest](https://rpubs.com/dvdgrmmngr/splom_midwest)

## Faceting graphs can be done automatically...

```
midwest %>%
 group_by(state) %>%
 do(plot =
 plot_ly.,
 x = ~ percollege,
 y = ~ percblack,
 name = ~ state,
 type = "scatter")) %>%
 subplot(nrows = 2,
 shareX = TRUE, shareY = TRUE)
```

`do()` performs the same operation for all subgroups (here: states) of the grouped data frame. In this case, it returns a data frame-like list with one line per state and a column named `plot` that holds the subplots for each state.

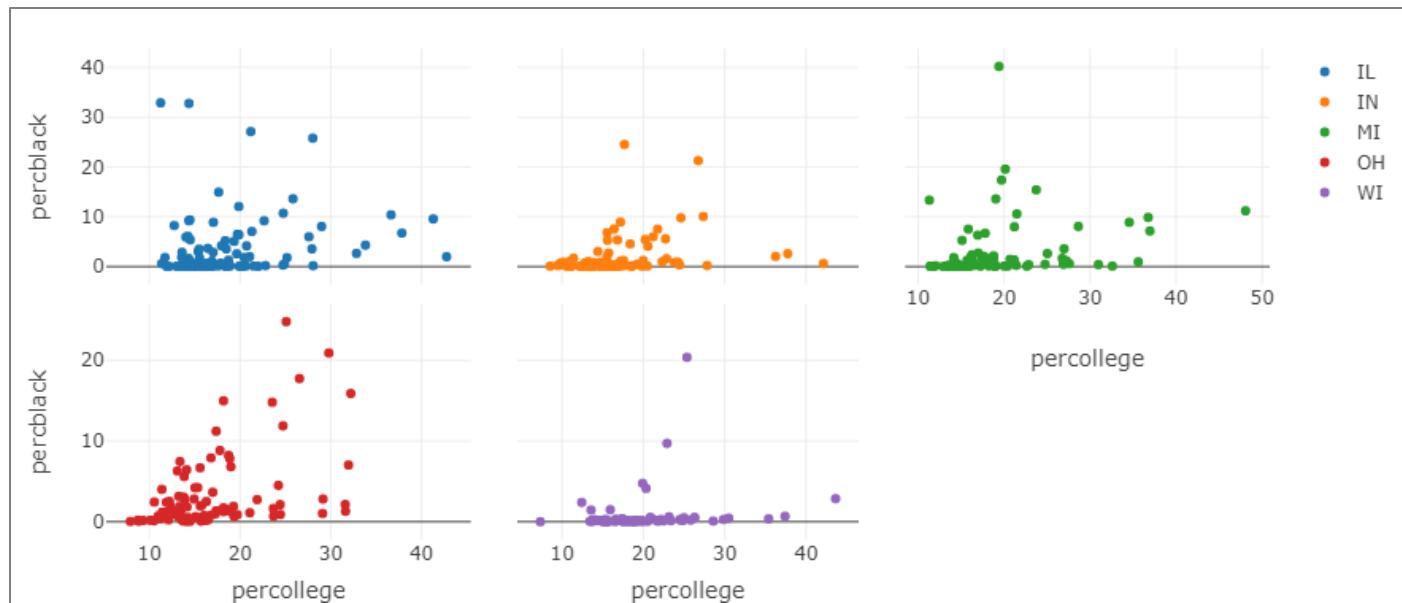




## Faceting graphs can be done automatically...

```
px.scatter(midwest,
 x = "percollege",
 y = "percblack",
 color = "state",
 facet_col = "state",
 facet_col_wrap = 3)
```

Similar to {ggplot2},  
facet\_col\_wrap defines the  
number of columns after which a  
new line is begun



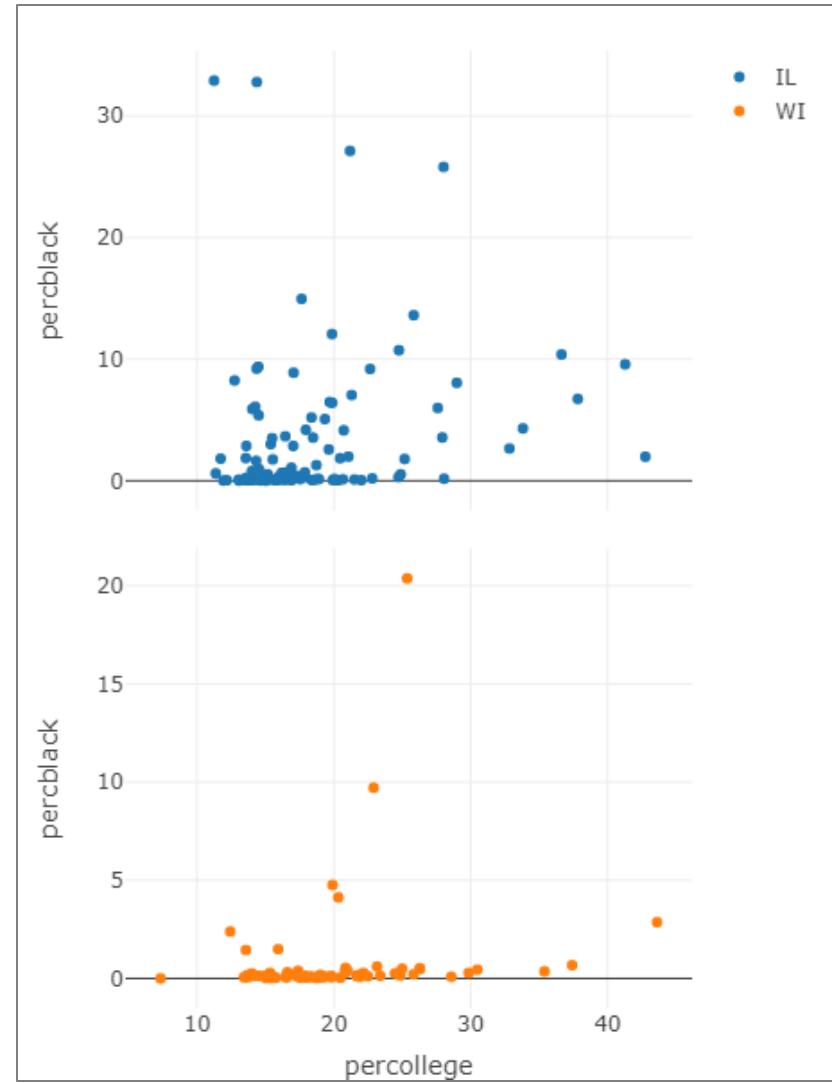
... or manually

```
illinois <- midwest %>%
 filter(state == "IL") %>%
 plot_ly(
 x = ~ percollege,
 y = ~ percblack,
 name = ~ state,
 type = "scatter"
)

wisconsin <- midwest %>%
 filter(state == "WI") %>%
 plot_ly(
 x = ~ percollege,
 y = ~ percblack,
 name = ~ state,
 type = "scatter"
)

subplot(illinois, wisconsin,
 nrows = 2,
 shareX = TRUE,
 shareY = TRUE)
```

With these options, we indicate  
that the x-axis and the y-axis are  
identical for the subplots.





## ... or manually

With these options, we indicate that the x-axis and the y-axis are identical for the subplots.

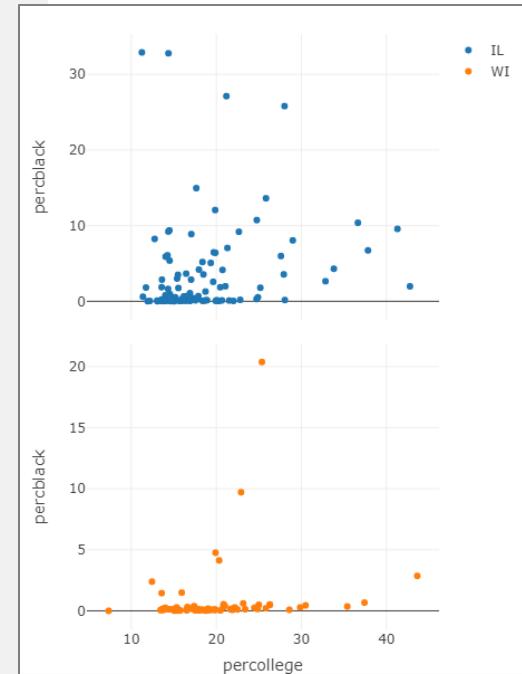
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

fig = make_subplots(rows=2, cols=1,
 shared_xaxes = True, shared_yaxes = True)

fig.add_trace(
 go.Scatter(x = midwest.loc[midwest["state"] == "IL", "percollege"],
 y = midwest.loc[midwest["state"] == "IL", "percblack"],
 mode = "markers", name = "IL"),
 row = 1, col = 1)

fig.add_trace(
 go.Scatter(x = midwest.loc[midwest["state"] == "WI", "percollege"],
 y = midwest.loc[midwest["state"] == "WI", "percblack"],
 mode = "markers", name = "WI"),
 row = 2, col = 1)

fig.update_xaxes(title_text = "percollege")
fig.update_yaxes(title_text = "percblack")
```



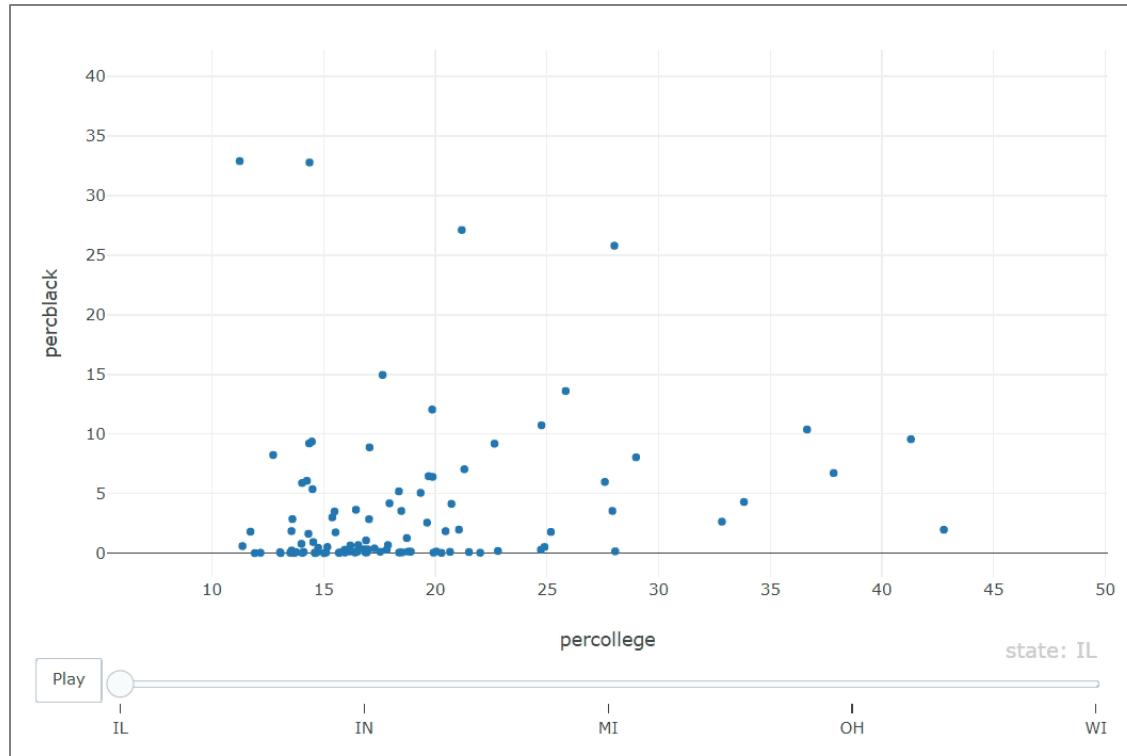
# Animations can be useful to highlight a particular piece of information



```
plot_ly(midwest,
 x = ~ percollege, y = ~ percblack,
 frame = ~ state,
 type = "scatter") %>%
 layout(showlegend = FALSE)
```



```
px.scatter(midwest,
 x = "percollege", y = "percblack",
 animation_frame = "state")
```



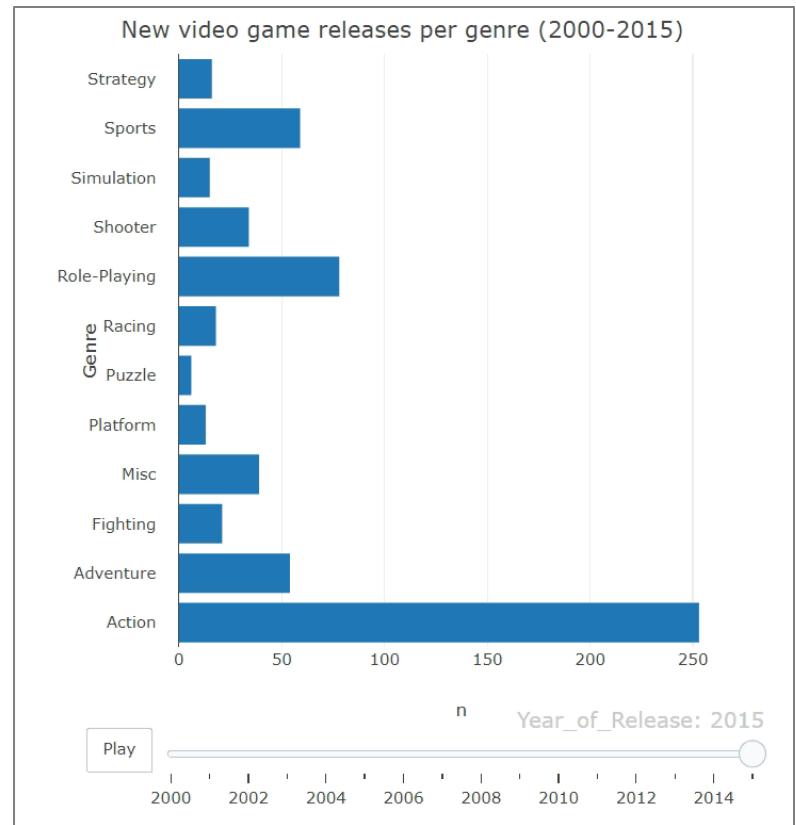
An animation consists of individual frames (here: one frame per state) that are combined with smooth transitions.

## Animations are particularly useful to illustrate developments over time

Consider the video game data from chapter A1 and Plotly in R:

```
vgsales %>%
 filter(
 Year_of_Release %in% c(2000:2015)) %>%
 plot_ly(y = ~Genre,
 frame = ~Year_of_Release,
 type = "histogram") %>%
 layout(showlegend = FALSE)
```

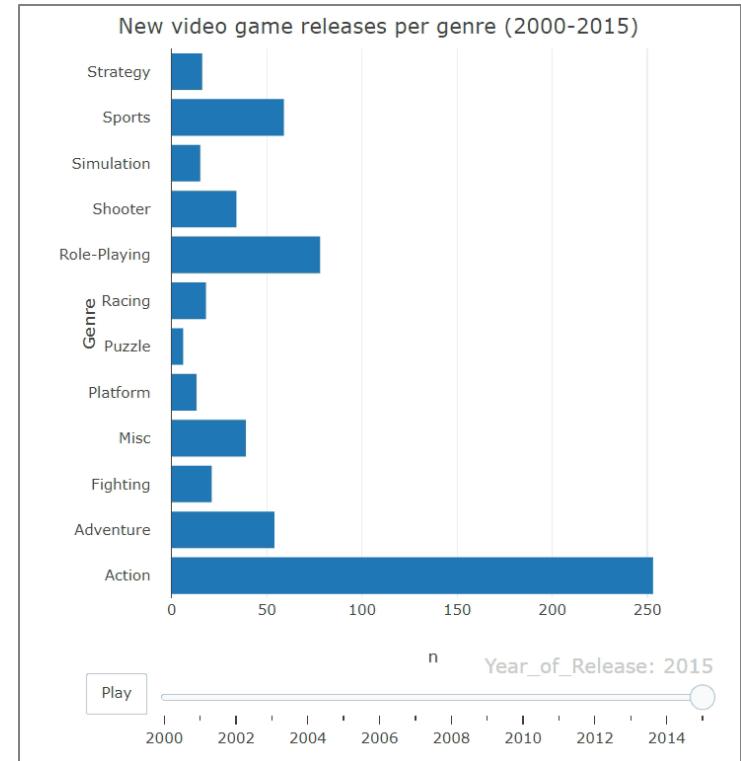
We create one frame per Year\_of\_Release and define Genre as identifier for the different time series.



## Customize animations with `animation_opts()` and `animation_slider()`

```
animation <- vgsales %>%
 filter(Year_of_Release %in% c(2000:2015)) %>%
 plot_ly(y = ~Genre,
 frame = ~Year_of_Release,
 type = "histogram") %>%
 layout(showlegend = FALSE)

animation %>%
 animation_opts(
 frame = 600, # ms between frames
 transition = 500, # duration of transition
 easing = "linear" # type of transition
) %>%
 animation_slider()
```

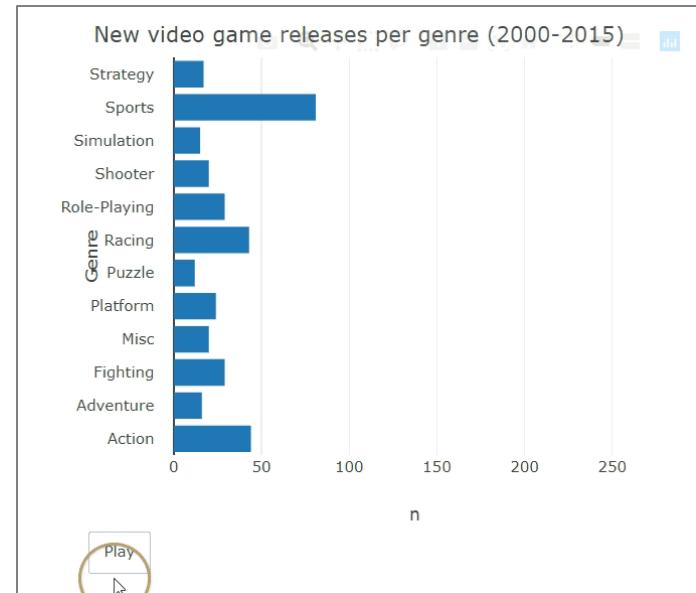
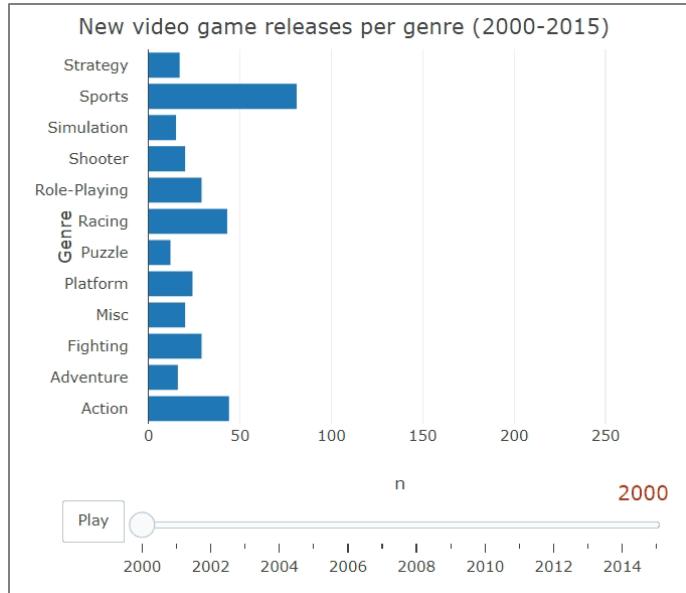


To accomplish this in Python, you have to create the slider from scratch. See <https://plotly.com/python/animations/> for instructions

## Customize animations with `animation_opts()` and `animation_slider()`

```
animation %>%
 animation_opts(
 frame = 600,
 transition = 300,
 easing = "bounce"
) %>%
 animation_slider(
 currentvalue =
 list(prefix = NULL,
 font = list(color = "red")))
```

```
animation %>%
 animation_opts(
 frame = 600,
 transition = 300,
 easing = "back"
) %>%
 animation_slider(hide = TRUE)
```



# Keep working on your Data Science skills!



The End