Introduction to R – Data Transformation

Niklas Haehn

Session 1

Introduction

What are these Workshops about?

- Introduction to R, with a focus on getting to know and implementing a classic data analysis workflow.

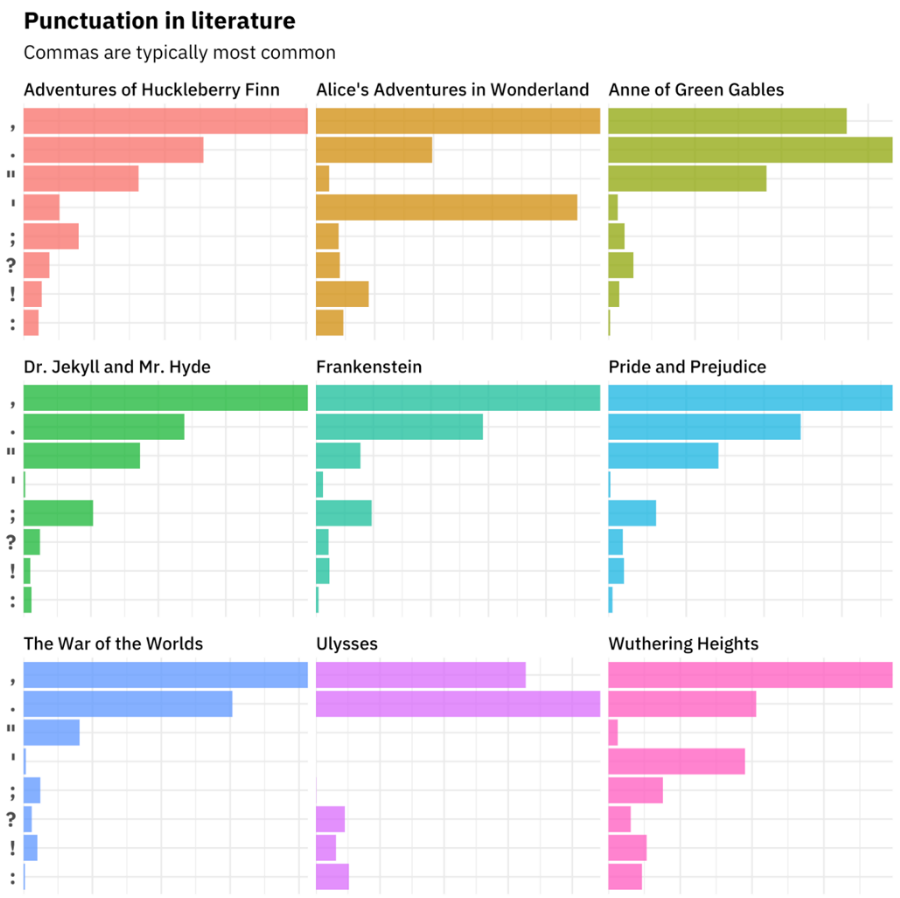
- Q: What prior knowledge is expected?

→ A: No prior knowledge of R - basic knowledge of statistics is an advantage.

- Q: Is this a classical statistics course?

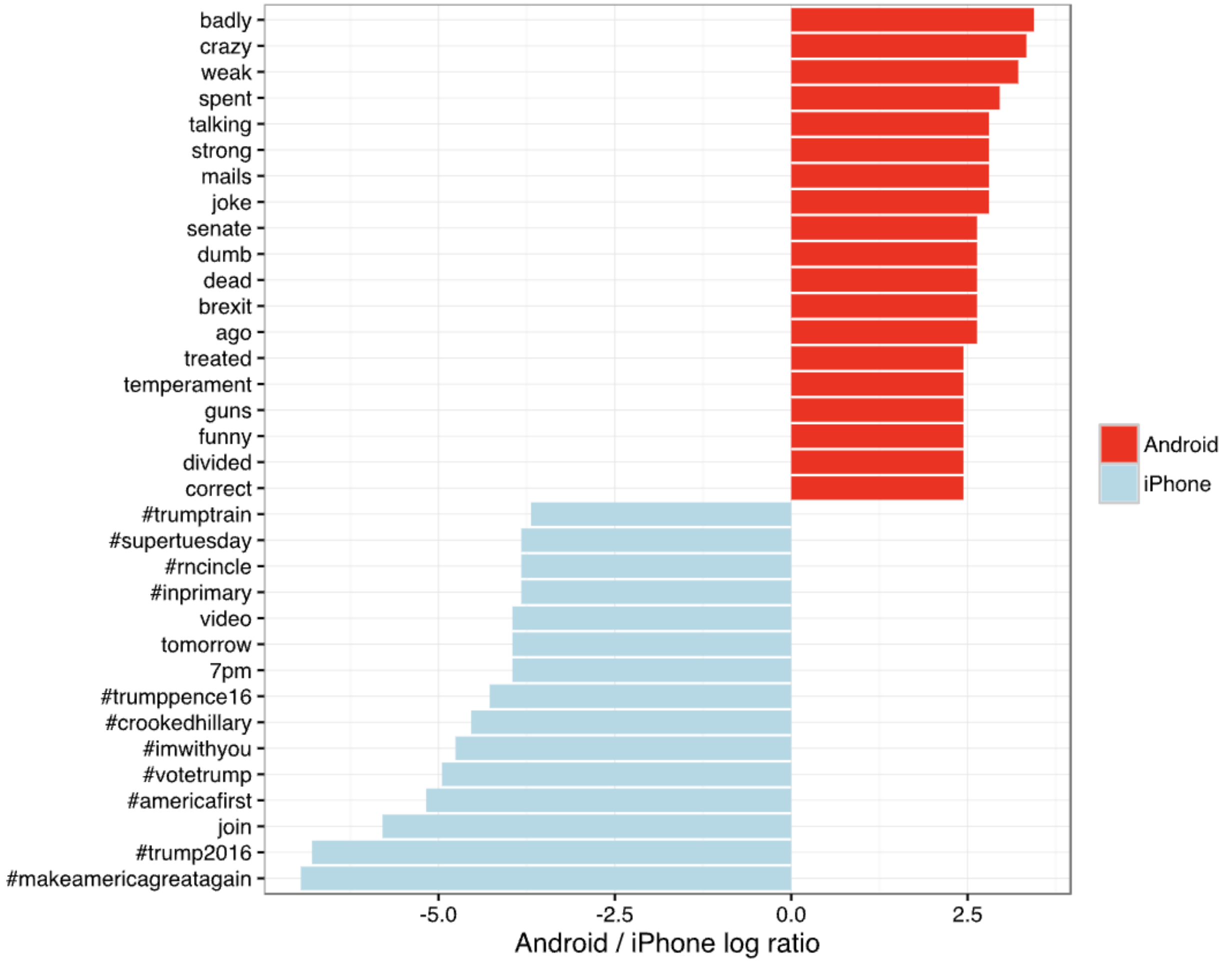
→ A: No!

Everyday Data Examples



https://juliasilge.com/blog/punctution-literature/

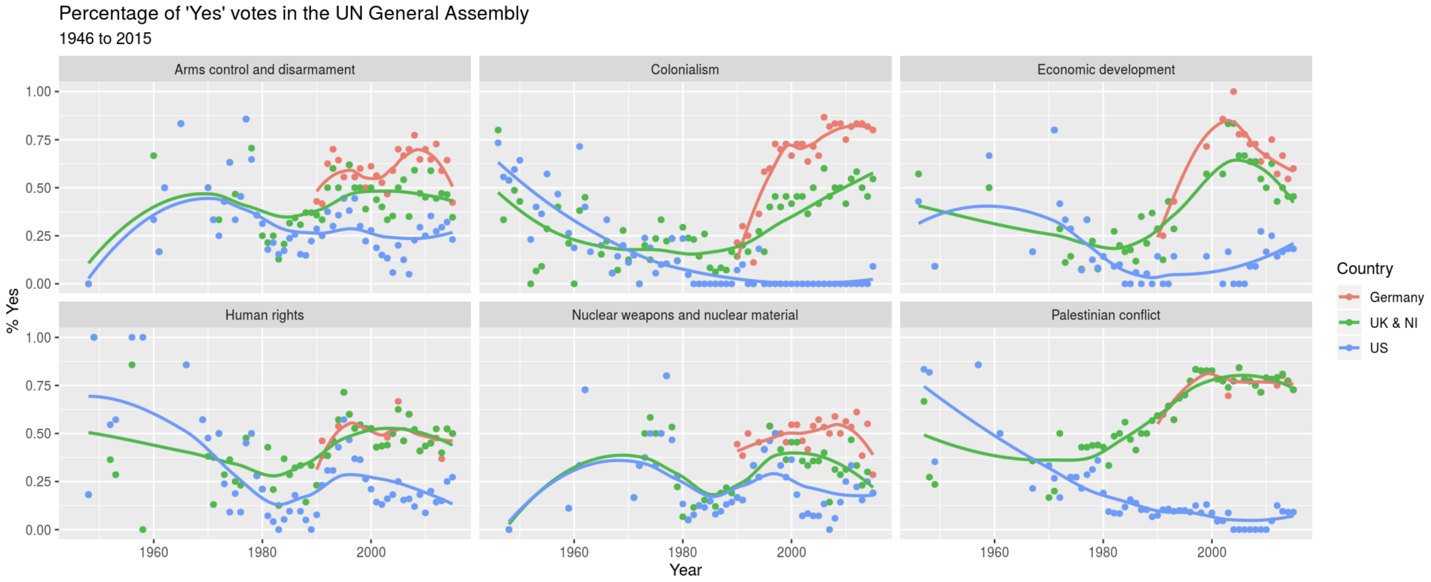
Everyday Data Examples



http://varianceexplained.org/r/trump-tweets/

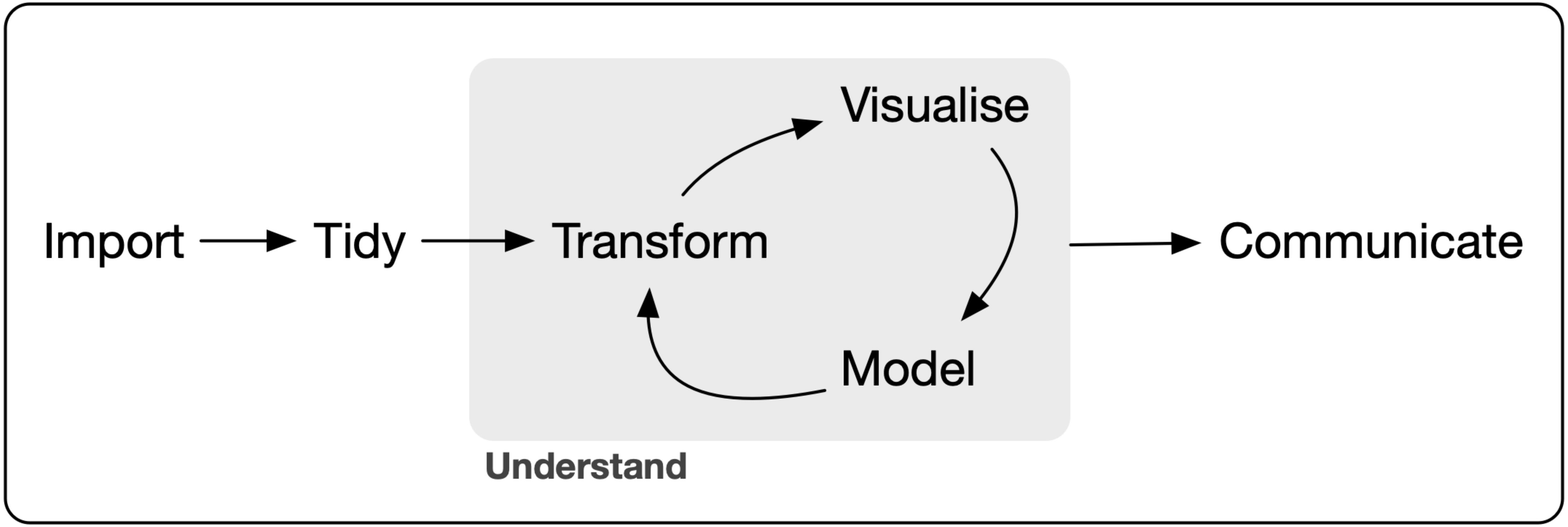
Everyday Data Examples

Voting patterns in the United Nations General Assembly



https://minecr.shinyapps.io/unvotes/

The tidy Cycle



Workshop Outline

- **1. Workshop** (10/21/24): Data Transformation
- **2. Workshop** (10/22/24): Data Modeling
- **3. Workshop** (10/23/24): Data Visualization

…

- **Prediction Competition** (10/23/24 – 10/27/24)
- **Election Watch Party** (11/05/24)

Expectation Management

- R is a (programming) language

- You don't become fluent in other languages within a few weeks - so it will take longer than a few weeks before you are confident in using R.

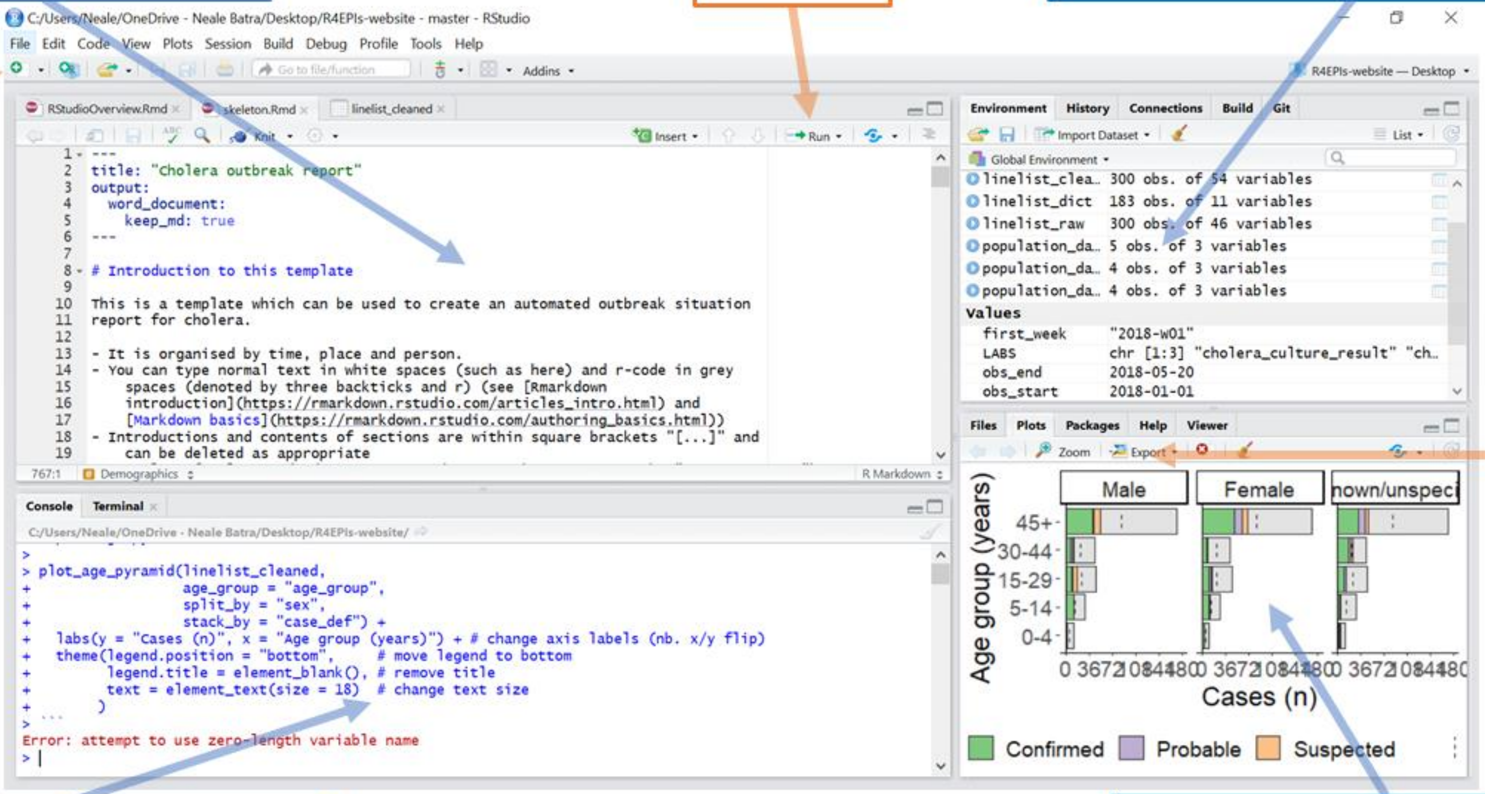
Overview R-Studio

Source Pane
Edit and run scripts (e.g. Rmarkdown templates), and view datasets

Tip: Run script

Environment Pane
Overview of objects (datasets, parameters, lists, etc.) you have imported or created.

Tip: Start new script



Tip: Zoom and export plots

R Console Pane
R commands run are shown here, and non-graphic output and errors are displayed

Plots, Packages, and Help Pane
Commonly used to view graphics, install packages, and view help

Set working Directory

Start your R-script by setting the working directory

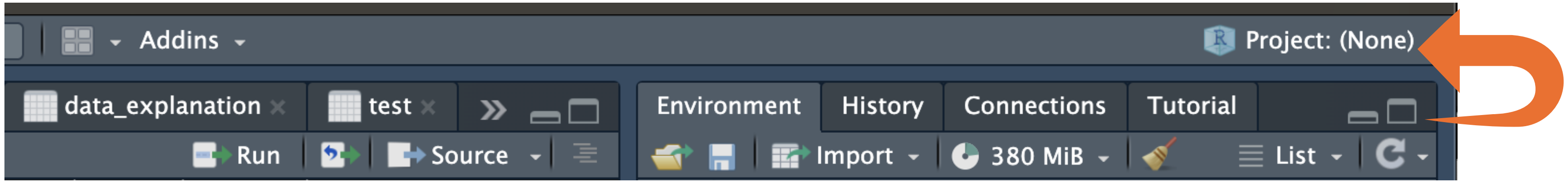
```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
options(scipen = 999)
set.seed(1234)
```

Install and load packages

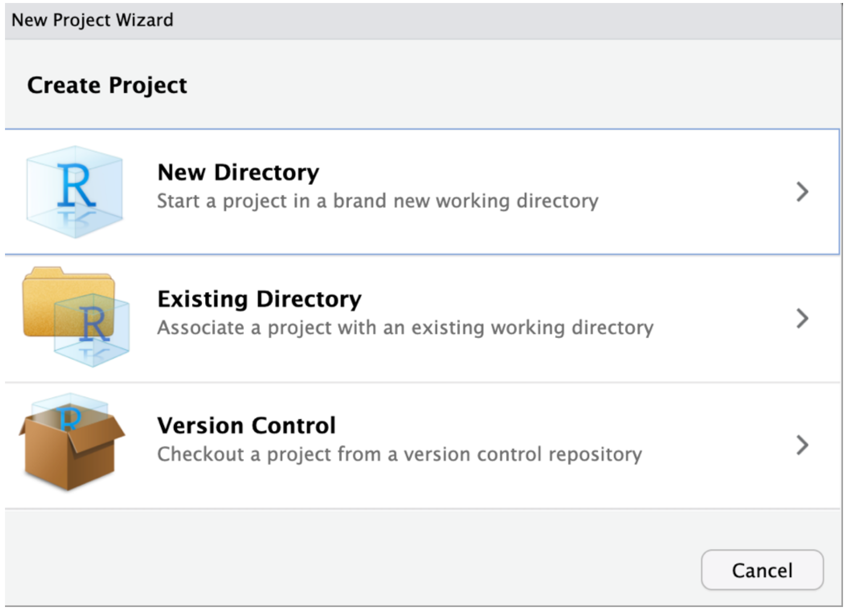
```
install.packages("tidyverse")
library(tidyverse)
```

Create a Project

1



2

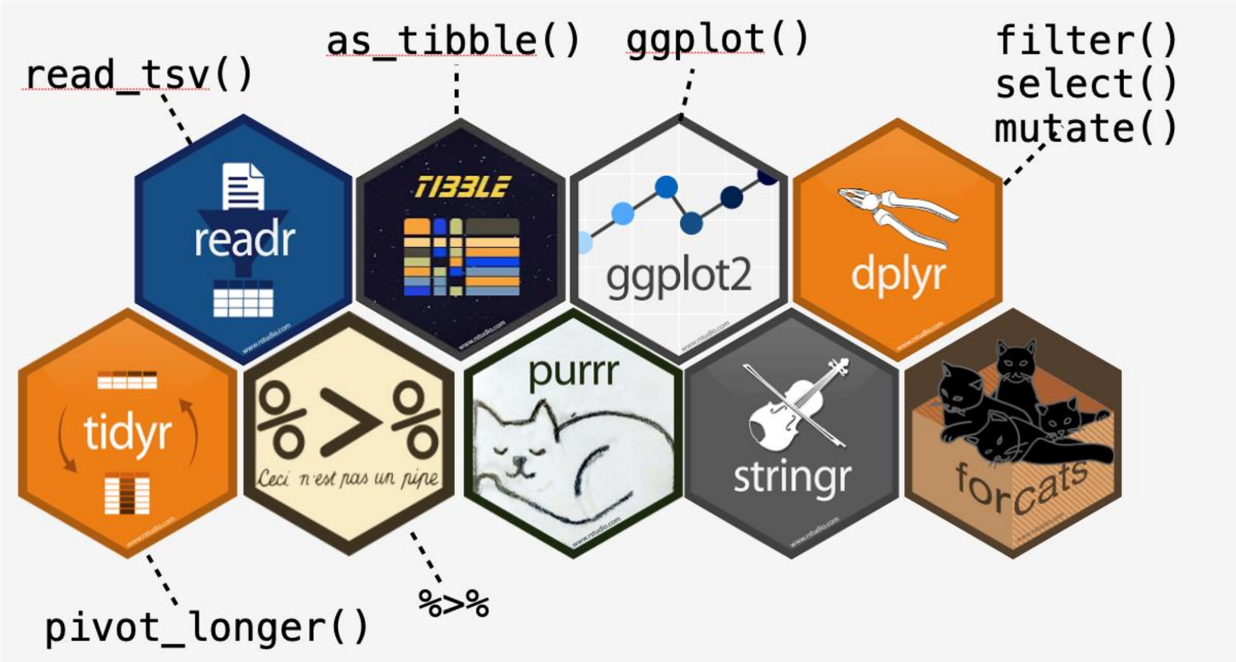


When you use a project you don't need to set a working directory!

Tidy Introduction

Characteristics of "Tidy" Data

- Each variable is a column.
- Each observation is a row.
- Each type of analysis unit is a table.



Importing Data (I)

- readr-package with functions for reading comma- and semicolon-separated (.csv) and tab-delimited (.tsv) files
  - Part of core-tidyverse → does not need to be installed and loaded separately

Comma-separated files (.csv)

```
data <- read_csv("path/to/my/data.csv")
```

Semicolon-separated files (.csv)

```
data <- read_csv2("path/to/my/data.csv")
```

Tab-separated files (.tsv)

```
data <- read_tsv("path/to/my/data.tsv")
```

Importing Data (II)

- haven-package with functions for reading files from other statistic programs (Stata, SPSS)
- readxl-package with function for reading Excel files
  - Not part of core-tidyverse → needs to be installed and loaded separately

read_dta for Stata files (.dta)

```
install.packages("haven")
library(haven)
data <- read_dta("path/to/my/data.dta")
```

read_sav for SPSS files (.sav)

```
data <- read_sav("path/to/my/data.sav")
```

read_xlsx for excel files (.xlsx)

```
install.packages("readxl")
library(readxl)
data <- read_xlsx("path/to/my/data.xlsx")
```

Importing Data (III)

- Never **modify the original version** of the imported data on your hard drive.
  - If changes are made to the dataset, save it under a **new name**.
- Changes should, if possible, **only be made using a script in R**.
  - If you need to switch to Excel or similar, at least document what you have changed.
  - The rule applies: "*Friends don't let friends use Excel for data analysis*."
- After importing data, **check** whether the data has been **imported correctly**.
  - You may need to specify the **delimiter** when reading CSV files.
  - You may need to define how **missing values** are represented in the data.

Saving and Loading of Objects (I)

R makes it possible to store any type of object, not just data records, locally

- For example:

```
numbers <- c(1, 2, 3)
countries <- c("US", "Germany", "France")

save(numbers, file = "data/numbers.RData")
save(countries, file = "data/countries.RData")
save(countries, numbers, file = "data/new_file.RData")
```

- Object name and file name do not have to match, but it often makes sense

Saving and Loading of Objects (II)

- load() loads the objects stored in the RData files into the working memory of R.
- Does not have to be assigned to an object when loading

```
load("data/numbers.RData")
load("data/countries.RData")
load("data/new_file.RData")
```

Grammar of Data Wrangling

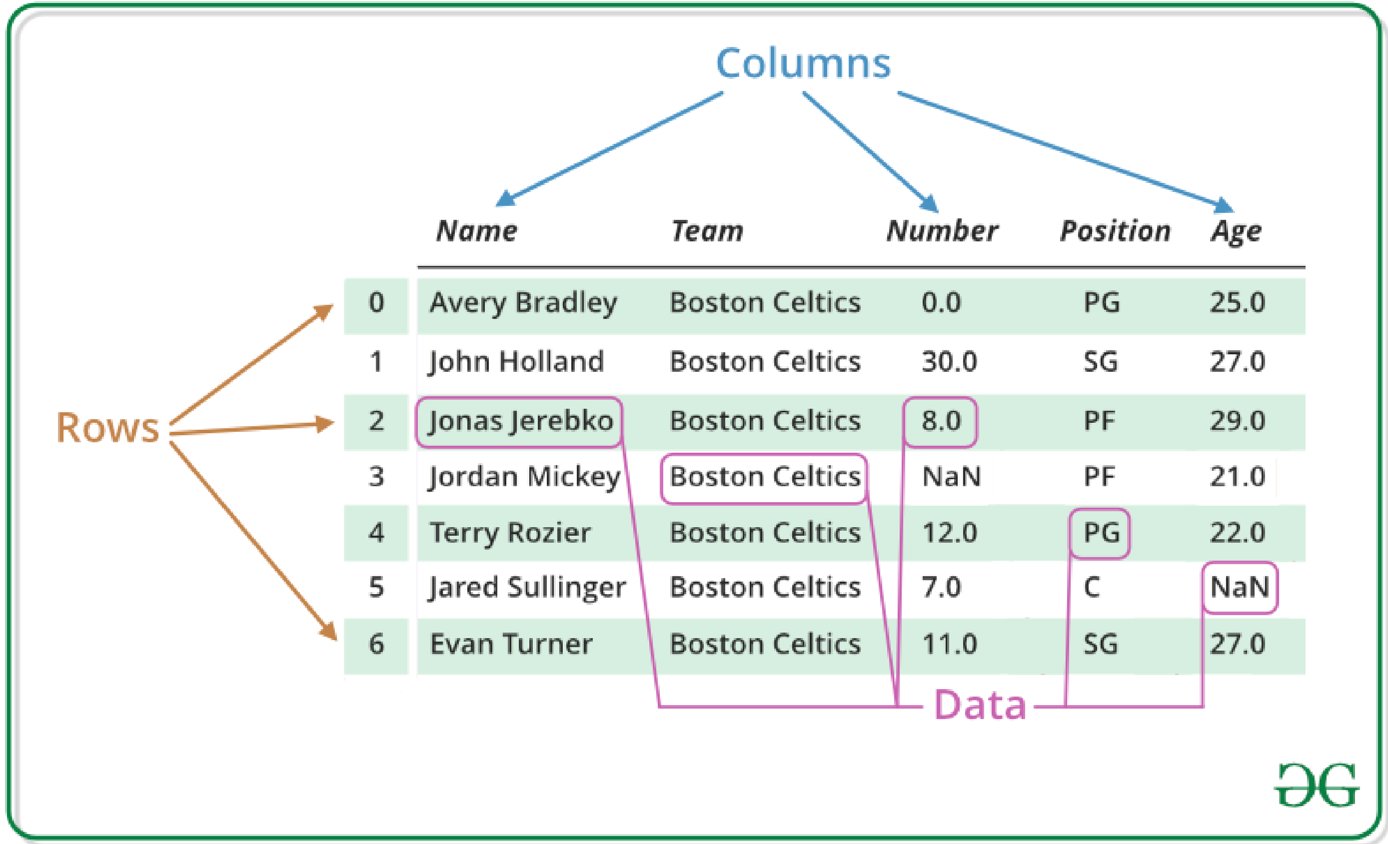
Grammar of Data Wrangling

- dplyr is a package that significantly simplifies data manipulation.
- It offers a series of functions that enable a wide variety of data manipulations:

  - arrange(): Sort data
  - filter(): Subset data by rows
  - select(): Subset data by columns
  - mutate(): Add or replace variables
  - summarize(): Aggregate data
  - group_by(): Group data for mutate(), summarize(), arrange(), or count()
  - ungroup(): Remove grouping structure
  - rename(): Rename variables
  - recode(): Recode values of variables
  - distinct(): Filters for unique values

data.frames & tibbles

- Datasets can be seen as collections of column or row vectors.
- Column vectors/variables don't have to be of the same type.
- However, all column vectors must have the same length



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Sort files

- Sort in ascending order

```
arrange(data, var1)
```

- Sort in descending order

```
arrange(data, desc(var1))
```

- Sort in ascending order with two variables.
- Sorting is hierarchical; first column takes priority.

```
arrange(data, var1, var2)
```

- **Numerical**: Sorted in ascending order by default.
- **Character**: Lexicographically sorted; uppercase comes before lowercase.
- **Factor Variables**: Sorted based on factor levels (predefined order).
- **Logical Values**: FALSE comes before TRUE.
- **Date/Time Variables**: Sorted chronologically.
- **Handling NA Values**: Default: NA values placed at the end.

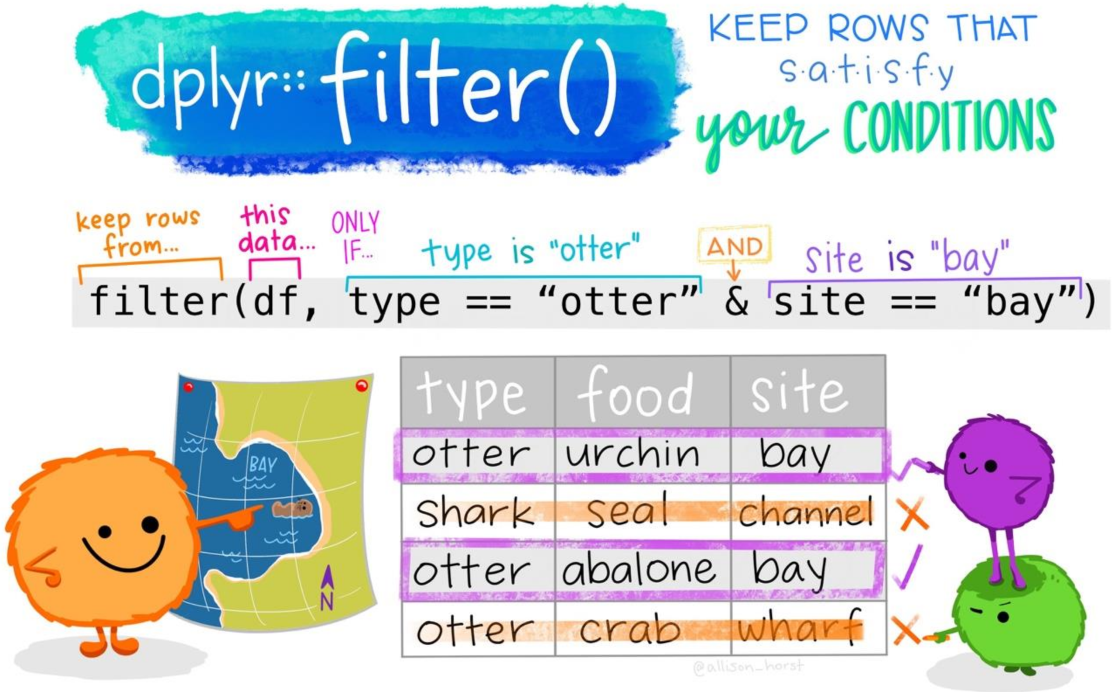
Subsetting (I)

We can subset a dataset based on:

- Variables (columns) and/or
- Observations (rows)

The corresponding dplyr functions are:

- select() for variables (columns)
- filter() for observations (rows)



Subsetting (II)

Data sets can be subsetted based on **column/variable names**

- One variable

```
select(data, var1)
```

- Two variables

```
select(data, var1, var2)
```

Subsetting (III)

Data sets can be subsetted based on **columns**

- Suppose we want to reduce the data set to certain values of a variable

```
filter(data, gender == "female")
```

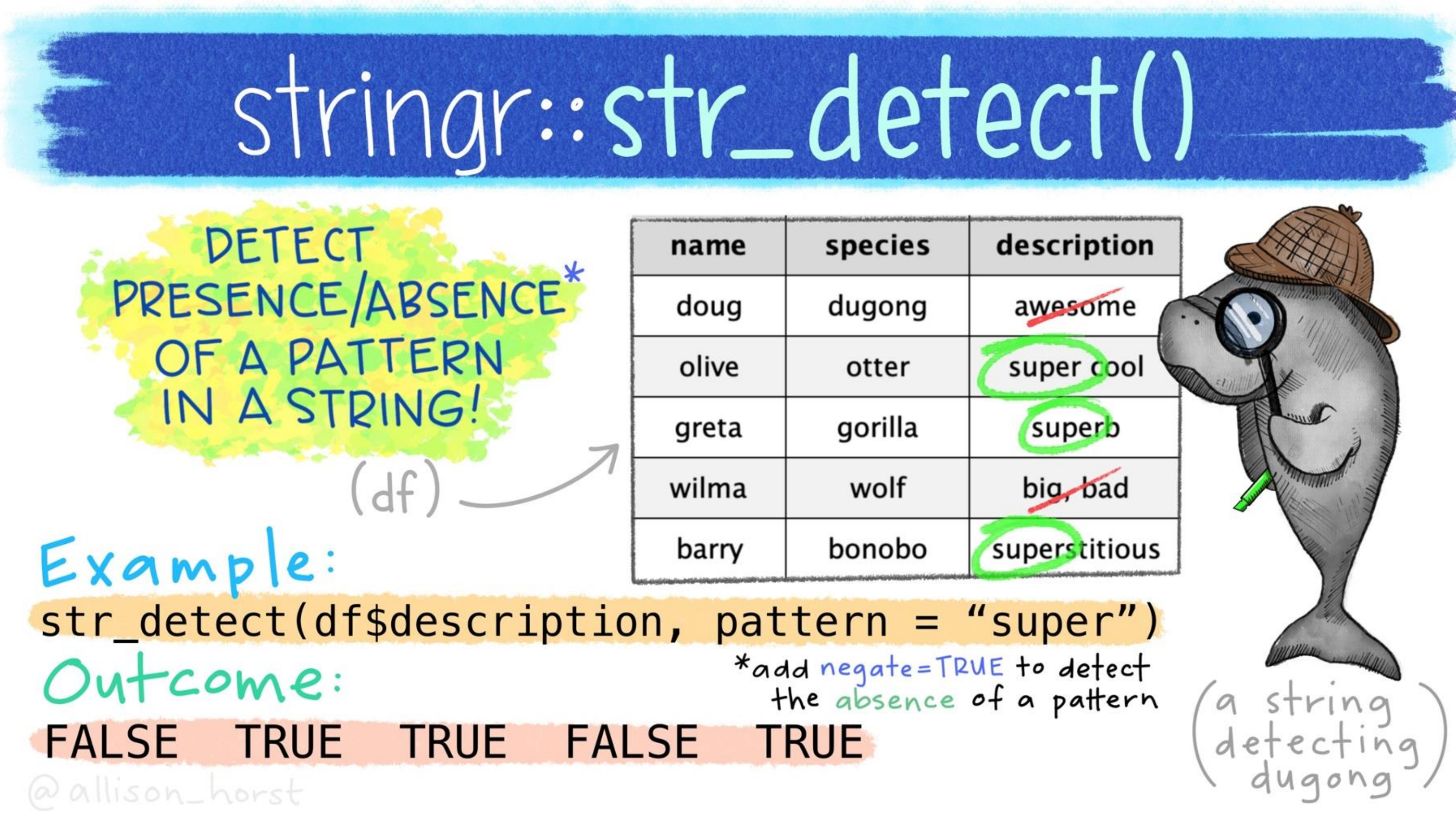
- Suppose we want to reduce the data set to certain values of a combination of several variables

```
filter(data, gender == "female" & age > 50)
```

- Suppose we want to know which is the maximum value of a specific variable

```
filter(data, age == max(age))
```

Subsetting (IV)



Data sets can be subsetted based on **columns**

- Suppose we want to reduce the data set to certain values of a variable

```
filter(data, str_detect(name, "a"))
```

Excurse Logical and Relational Operators

Logical

- &: “and”
- |: “or”
- !: “not”

Excursion: Relational Operators

- x > y: If x is greater than y, TRUE is returned.
- x < y: If x is less than y, TRUE is returned.
- x ≤ y: If x is less than or equal to y, TRUE is returned.
- x ≥ y: If x is greater than or equal to y, TRUE is returned.
- x == y: If x is equal to y, TRUE is returned.
- x != y: If x is not equal to y, TRUE is returned.

Piping (|>)

|> is the so-called "piping" operator.

- It passes the result of one function to the next, allowing code to be written in the order of execution.
- It can be read as "then".

```
data |>  filter(variable == max(variable))
# is easier to read than
filter(data, variable == max(variable))
```

Please note that some people might use the %>% operator instead. The functionality is almost the same

Create Variables (I)

- mutate() creates new variables

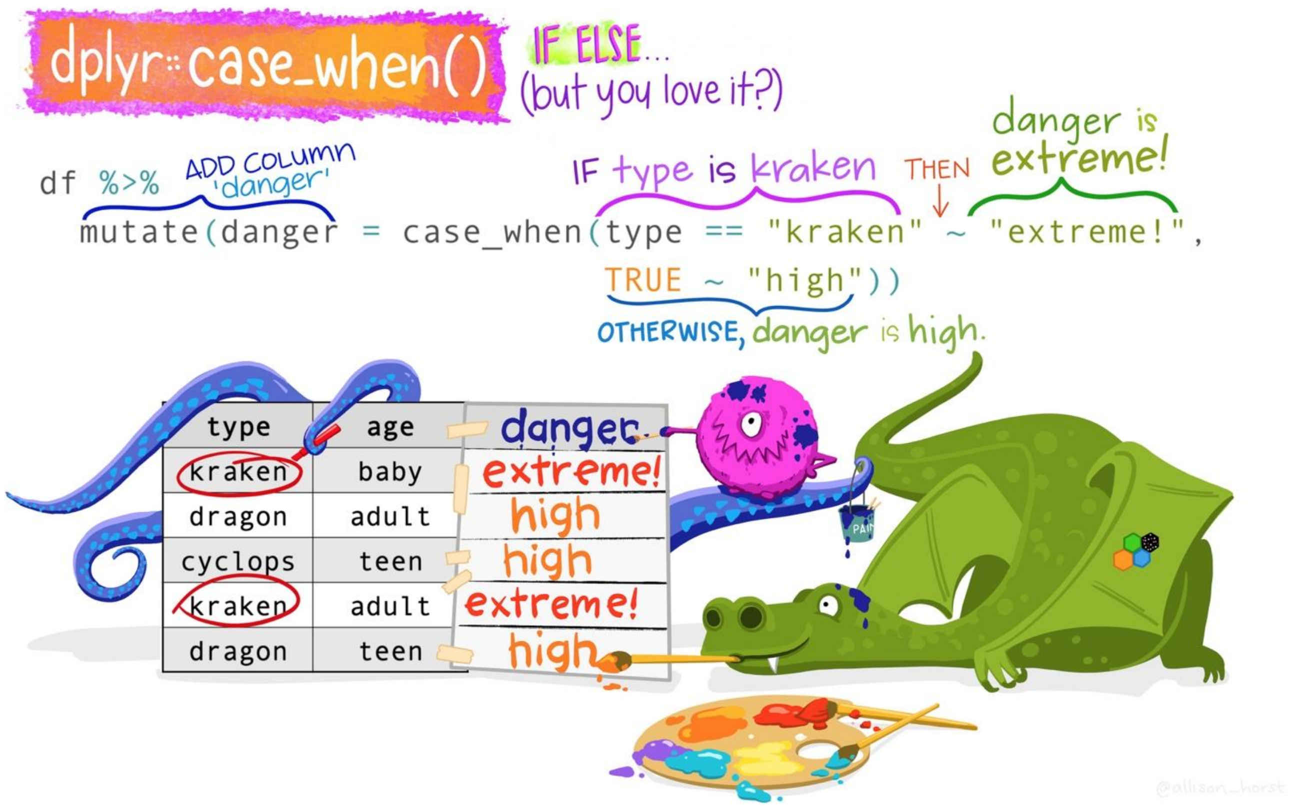
```
data |>  mutate(new_variable = variable - mean(variable))
```

```
# Create a dummy variable with "if_else()"
data |>
  mutate(new_variable = if_else(gender == "Male", 1, 0))
```

Create Variables (II)

- Use mutate() with case_when to creates new variables

```
data |>
  mutate(new_variable = case_when(
  gender == "Male" ~ 1,
  gender == "Female" ~ 2,
  gender == "non-binary" ~ 3,
  True ~ NA
))
```



Aggregate Data

- summarize() aggregates the values into a single value

```
data |> summarize(variable_mean = mean(variable))
```

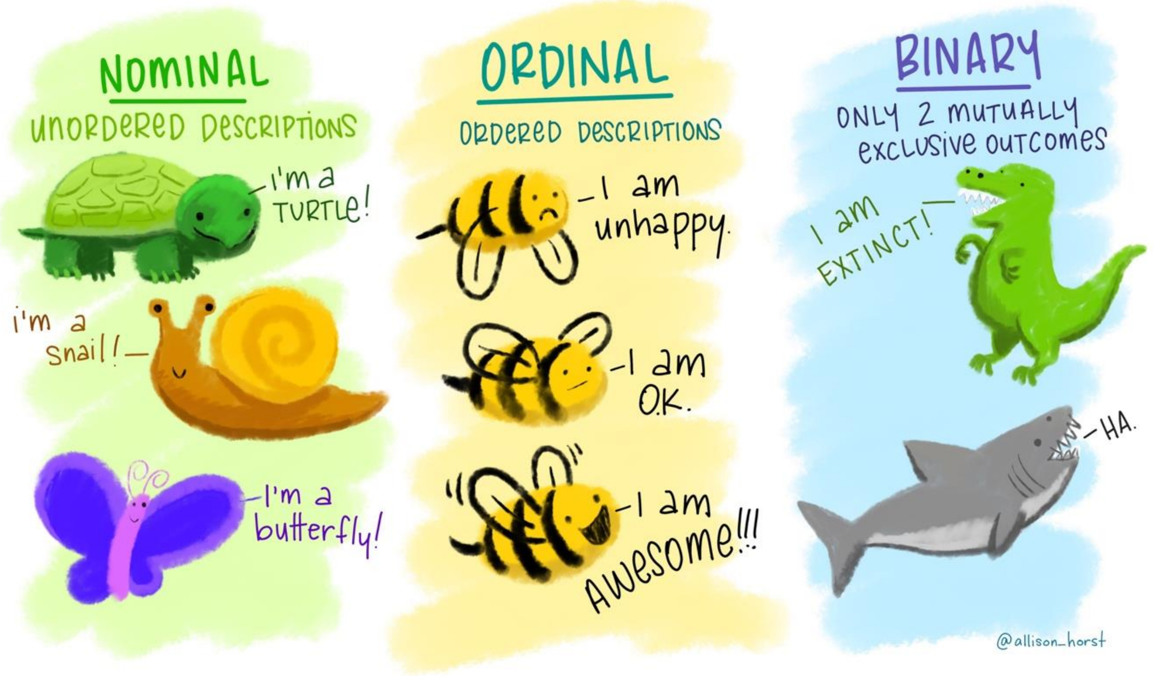
With group_by(), data can be grouped based on the values of one or more variables.

```
data |>
  group_by(gender) |>
  summarize(variable_mean = mean(variable)) |>
  ungroup()
```

- This is equal to

```
data |>
  summarize(variable_mean = mean(variable), .by = gender)
```

Recoding Variables



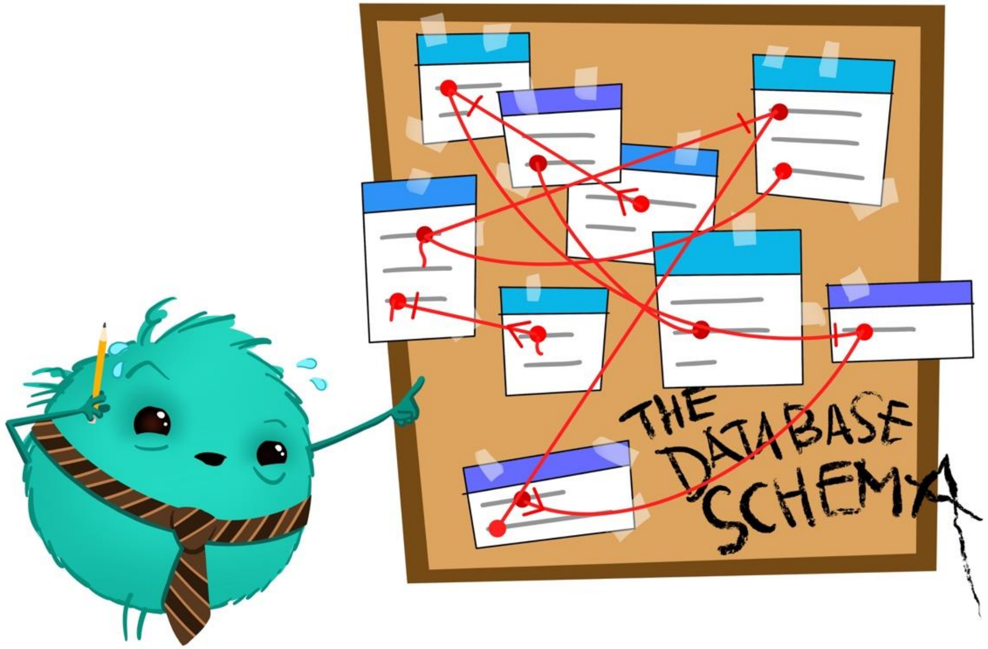

```
#library(labelled)
data |>
  mutate(variable = recode(variable,
                           `0` = "blue",
                           `1` = "red"))
```

For DTA files (Stata), you might need to load the labelled package before using recode() (and install it if it's your first time using it). This is because, when reading the data with the haven package, Stata's labeled variables are carried over.

Merging data sets

With left_join() (see also ?join), datasets can be merged.

- Input:
  - 2 datasets
  - Identification variable(s)




```
df1 <- read_csv("Data/df1.csv")
df2 <- read_csv("Data/df2.csv")

# Both datasets have a variable "countryname":
df_merged <- left_join(df1, df2, by = "countryname")

# Both datasets have a variable containing country names;
# in df1 it's called "countryname", in df2 it's called "CNTRY":
df_merged <- left_join(df1, df2, by = c("countryname" = "CNTRY"))
```

Useful when working with data from different countries is the countrycode package

for standardizing country names and IDs.

Transforming Datasets ("Reshaping") (I)

- "Tidy Data" refers to data where each row is an observation and each column is a variable.
- This is also known as the "long" format, in contrast to the "wide" format.
- Many functions (e.g., ggplot2 for data visualization) require data in the "long" format.
- However, in practice, data doesn't always come in this format.

→ Use the tidyr package to reshape datasets.

Transforming Datasets ("Reshaping") (II)

- Example: Fictional election results for two parties in different states.

```
df_messy <- data.frame(state = c("CT", "CA", "NY"),
                       party_1 = c("60%", "50%", "30%"),
                       party_2 = c("40%", "50%", "70%"))
```

```
head(df_messy)
#>   state party_1 party_2
#> 1    CT     60%     40%
#> 2    CA     50%     50%
#> 3    NY     30%     70%
```

In principle, we have three variables here:

- State
- Party name
- Election result

Transforming Datasets ("Reshaping") (III)

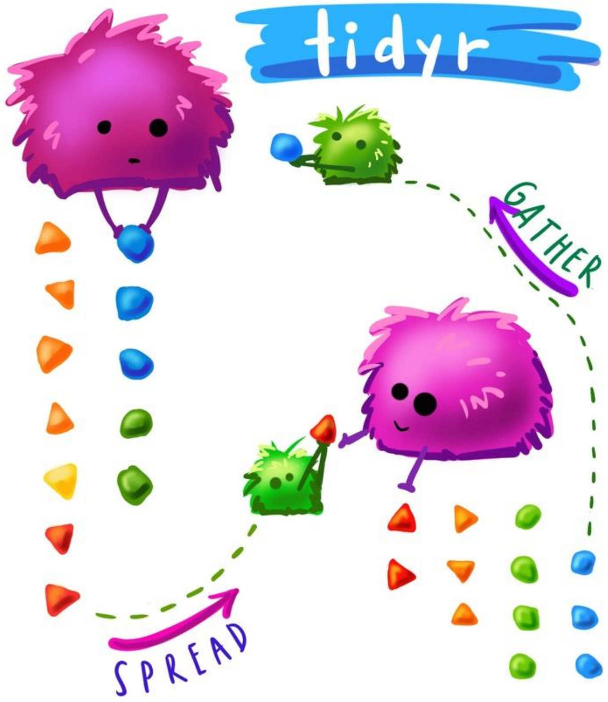
Use tidyr's pivot_longer() to transform a dataset into the "long" format.

- Data: The dataset to be transformed.
- cols: Columns that should be reshaped into the long format.
- names_to: The name of the new column that will contain the variable names.
- values_to: The name of the new column that will contain the values of the reshaped variables.

```
df_tidy <- pivot_longer(df_messy,
                        cols = party_1:party_2,
                        names_to = "party",
                        values_to = "vote_share")
```

Transforming Datasets ("Reshaping") (IV)

```
head(df_messy)
#>    state party_1 party_2
#> 1     CT      60%     40%
#> 2     CA      50%     50%
#> 3     NY      30%     70%
```




```
head(df_tidy)
#> # A tibble: 6 × 3
#>   state party   vote_share
#>   <chr> <chr>   <chr>
#> 1 CT    party_1 60%
#> 2 CT    party_2 40%
#> 3 CA    party_1 50%
#> 4 CA    party_2 50%
#> 5 NY    party_1 30%
#> 6 NY    party_2 70%
```

Transforming Datasets ("Reshaping") (V)

```r
df_messy <- pivot_wider(df_tidy,
                        id_cols = "state",
                        names_from = "party",
                        values_from = "vote_share")
```

```r
head(df_tidy)
#> # A tibble: 6 × 3
#>   state party   vote_share
#>   <chr> <chr>   <chr>
#> 1 CT    party_1 60%
#> 2 CT    party_2 40%
#> 3 CA    party_1 50%
#> 4 CA    party_2 50%
#> 5 NY    party_1 30%
#> 6 NY    party_2 70%
```

```r
head(df_messy)
#> # A tibble: 3 × 3
#>   state party_1 party_2
#>   <chr> <chr>   <chr>
#> 1    CT     60%     40%
#> 2    CA     50%     50%
#> 3    NY     30%     70%
```

Using LLMs for Tidy Data Transformation

Chat GPT

Write the following R Code in tidy: Convert an event dataset containing columns for year, month, and day into a panel dataset based on months

To convert an event dataset that has columns for `year`, `month`, and `day` into a panel dataset based on months using the tidyverse, you can follow these steps:

1. Combine the `year`, `month`, and `day` columns into a `date` column.
2. Use `floor_date` from the `lubridate` package to aggregate the data by month.
3. Optionally, summarize or arrange the data by months, depending on the goal.

Here's an example using tidyverse functions:

r Code kopieren

```r
# Load necessary libraries
library(dplyr)
library(lubridate)

# Example dataset
events <- tibble(
  year = c(2021, 2021, 2022, 2022),
  month = c(1, 2, 3, 3),
```

Tidyverse Coder

You

Convert an event dataset containing columns for year, month, and day into a panel dataset based on months

2024-10-18 13:27:04

Get Embed C

Tidyverse Coder

To convert an event dataset containing separate columns for year, month, and day into a panel dataset based on months, we need to perform several steps. The idea is to group events by month, creating a time series where each row represents a unique year-month combination, instead of individual events. The exact implementation may vary depending on the structure of the dataset and the information you want to retain in the panel dataset.

Steps:

1. **Combine year and month**: Create a single `` `year-month` `` column or a proper date column representing the month.

The End

Have fun wrangling!

Practice!!

- How many people have brown hair?
- What is the average height of individuals from Tatooine or Alderaan?
- How many characters appeared in each movie? Create a ranked list from fewest to most.
- Create a dummy variable indicating whether a person is female or not.
- Create a density variable (mass/height).
- Generate a dataset where each column represents a film (7 in total), and the values are the characters that appeared in each respective film.

Resources

- Graphics by @allison_horst
- Content based on
  - “Data Science in a Box” and ‘Introduction to Data Science’ by Mine Çetinkaya-Rundel
  - “R for Data Science” by Wickham et al. (https://r4ds.hadley.nz/)
  - ZfS-Kurs “Einführung in die Statistik R” by Verena Kunz