FORSCHUNGSPRAKTIKUM I UND II: VERGLEICHENDE SOZIALFORSCHUNG MIT MEHREBENENMODELLEN IN R

Dr. Christian S. Czymara
Introduction to R

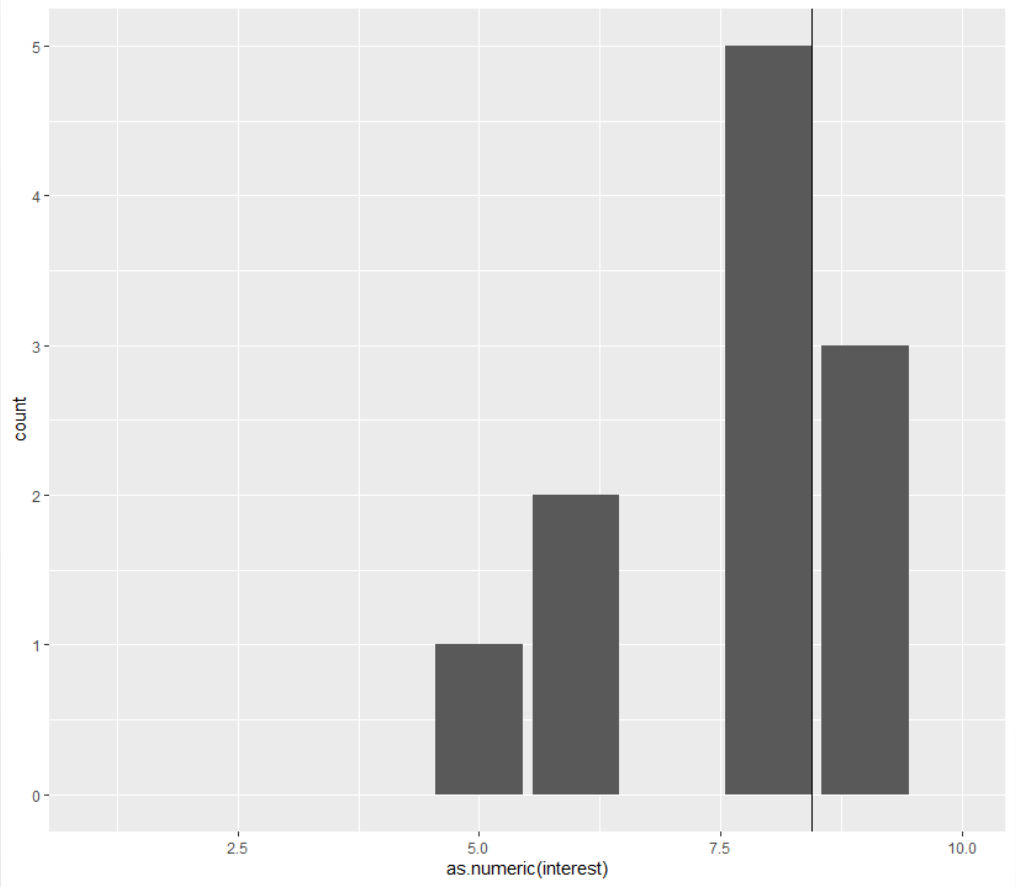
AGENDA

- Results of survey
- R basics
- Data preparation

YOUR INTEREST AND KNOWLEDGE

YOUR INTEREST

- Good news: interest is pretty high (mean: 8.4 of 10)

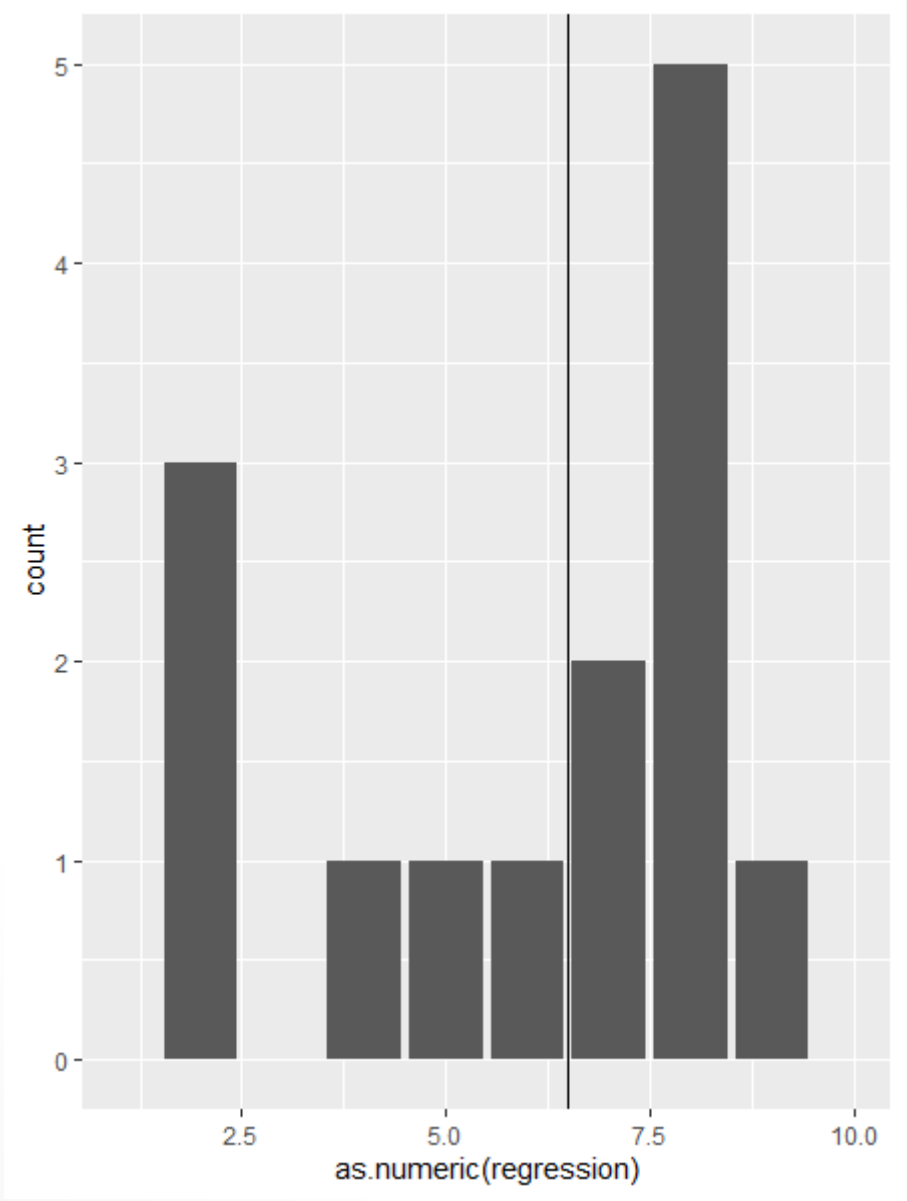


YOUR EXPECTATIONS

- *enough time to get familiar with R*
- *exploring new functions of R, Stata and SPSS*

YOUR KNOWLEDGE

- Knowledge unequally distributed
- Half of the class are very familiar with regression (>=8)
- 12.5% are experts (10)
- However, ~20% are quite unfamiliar (<=2)



YOUR KNOWLEDGE

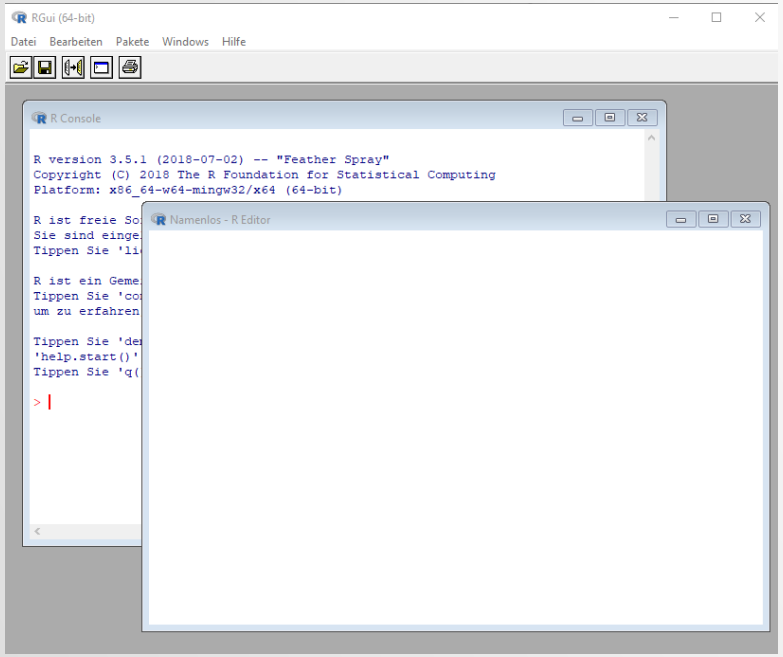
- Over two thirds have used linear regression before, about one third logistic regression
- … but there are also students how don't know what linear or logistic regression are
- Half of the already work with R

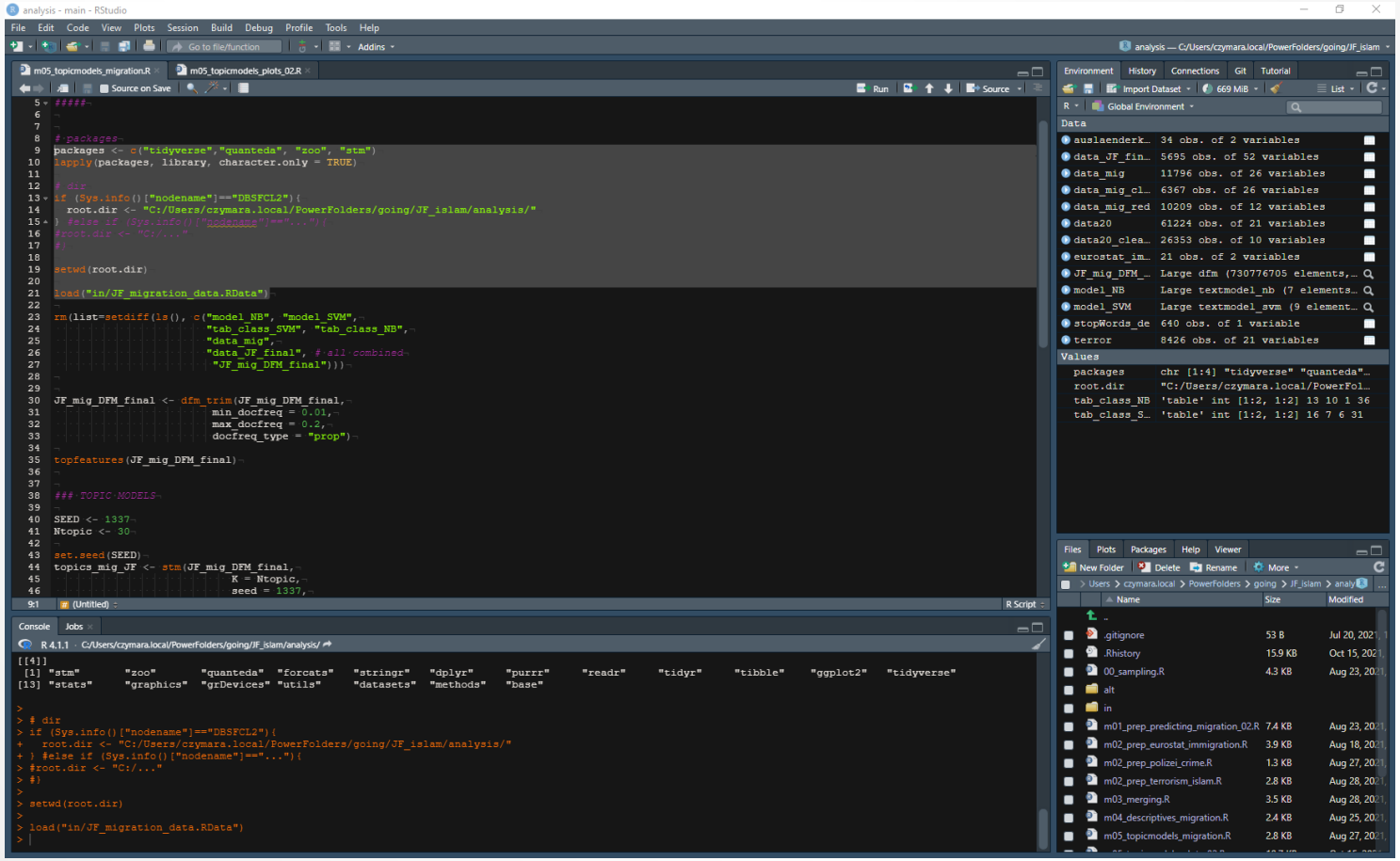
WHAT IS R?

R

- Why "R"? → *"R is an implementation of the S programming language"* (Wikipedia)
- R is a programming language for data analysis
- Rstudio is a so called Integrated Development Environment (IDE), making your work a lot easier
  - Writing and running R Code
  - Overview of stored objects
  - Projects containing multiple files
  - Git connection
  - Etc.

R VS. RSTUDIO


RGui (64-bit)
Datei Bearbeiten Pakete Windows Hilfe
R Console
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
Namenlos - R Editor


analysis - main - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
m05_topicmodels_migration.R
m05_topicmodels_plots_02.R
Environment History Connections Git Tutorial
Data
Values
Console Jobs
Files Plots Packages Help Viewer

R BENEFITS

- Free and open source
- Large and very helpful community
- Plethora of user-written packages on basically everything
- Very powerful tools for data manipulation and data visualization
- In addition to analyzing data, you can write programs, websites, books, and much more with R (and R Markdown)
- … and integrate with other languages

R BASICS

MATH OPERATORS

- Addition (+), subtraction (–), multiplication (*), Division (/), exponentiation (^), exponential (`exp()`), logarithm (`log()`), and basically everything else
- For example: `3+2` will return `5`
- Operators can also be combined: `(3+5)/(4*2)` will return `1`
- But we wouldn't need R for that…

OBJECTS

- Crucially, R allows to store information (e. g., numbers or text) in an object
- To create an object, use the *assignment operator*: <-
  - E. g.: `result_1 <- 3+2`
  - `result_2 <- 4*2`
- These objects can be recalled:
  `result_3 <- result_1 / result_2` will again return `1`
- Note that object names can be anything, better avoid generic names such as `result_1, result_2, result_3` ☺

LOGICAL OPERATORS

- Tests whether something is `True` or `False`
- For example: `result_1 == result_2` will be `True` (because 8=8)
- `result_1 == result_3` will be `False` (because 8≠1)
- But `result_1 != result_3` will again be `True`
- Similarly, `result_1 > result_3` will be `True`

LOGICAL OPERATORS

- Logical operators can be combined using `&` ("and") and `|` ("or")
- Example 1: `8 == 8 & 8 > 1`
- Example 2: `8 == 8 & 8 == 1`
- Example 3: `8 == 8 | 8 == 1`
- Example 3: `8 != 8 | 8 > 1`
- This will often be relevant when you create new variables or define your sample of analysis (e. g.: relevant age range from 18 to 65 and only natives)

VECTORS

- Storing a single value (as in object `result_1`) is not very interesting in most cases
- If you want to store many values simultaneously, you work with vectors
- For example: `variable_num <- c(8, 8, 1)`
- Importantly, vectors don't have to be numerical but can also be strings ("text") `variable_char <- c("a", "b", "c")`
- We can also combine the numerical objects we created before: `variable_num <- c(result_1, result_2, result_3)`

INDEXING

- What if we want to recall a certain value of `variable_num`?
- Use indexing, which is done via `[]`
- `vector[elements]`
- Let's say we want to access the first element of `variable_num`
  → `variable_num[1]` will return `8`
  → `Variable_num[2]` will return `8`
  → `variable_num[3]` will return `1`
- You can also nest these: What will `variable_num[variable_num[3]]` return?

VARIABLE TYPES

THE THREE (MOST IMPORTANT) TYPES OF VARIABLES

1. *Logical*: Binary variable with values `True` and `False` → `class(True)`
2. *Character* (string): Text (including symbols and numbers that are treated as text) → `class("this is 1 character")`
3. *Numeric*: Numbers for mathematical operations → `class(123)`

- Often *character* and *numeric* correspond to *categorical* and *continuous* variables
- `NA` is an value that means *missing value* ("not available"), important if you work with real-world data

DATA FRAMES

VARIABLES AND DATA FRAMES

- So far, we learned about single variables (logical, numerical, character)
- However, in most cases we won't analyze a bunch of unrelated variables but rather several variables of one (or more) data sets
- Data sets (called data frames in R) are a collection of variables that are organized in a two-dimensional table
  - Column: variable
  - Row: observations
  - Cells: values
- You can turn variables into a data frame using `as.data.frame()`

EXAMPLE OF DATA FRAME

Filter

	trstplc Trust in the police	trstprl Trust in country's parliament	trstlgl Trust in the legal system	trstplt Trust in politicians	trstprt Trust in political parties	trstep Trust in the European Parliament	trstun Trust in the United Nations	discrim	blgetmg Belong to minori
1	10	9	10	0	*NA*	*NA*	9	no	2
2	5	0	8	0	*NA*	0	6	yes	2
3	8	6	4	2	*NA*	7	*NA*	no	*NA*
4	9	8	10	4	*NA*	7	8	no	2
5	4	6	7	4	*NA*	4	5	no	2
6	6	0	5	0	*NA*	2	*NA*	yes	2
7	6	6	6	5	*NA*	5	*NA*	no	2
8	7	9	7	4	*NA*	6	6	no	2
9	8	5	7	3	*NA*	2	0	no	2
10	5	0	3	5	*NA*	*NA*	*NA*	no	1
11	7	2	2	0	*NA*	0	0	no	*NA*
12	3	5	5	3	*NA*	3	3	no	2
13	2	0	0	0	*NA*	0	0	no	*NA*
14	8	5	10	5	*NA*	5	5	no	*NA*
15	7	8	5	5	*NA*	5	8	no	*NA*
16	4	6	6	4	*NA*	6	5	no	2
17	8	9	9	6	*NA*	7	6	no	2
18	6	5	8	5	*NA*	6	5	no	2
19	8	5	7	3	*NA*	2	2	no	2

HOW TO ACCESS A VARIABLE WITHIN A DATA FRAME

- Use the `$` operator
- For example, to access the variable `discrim` in the data frame `ESS`, type `ESS$discrim`

```
> ESS$discrim
  [1] no  yes no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  n
 [34] no  no  no  no  no  no  no  no  no  no  no  no  <NA> no  no  no  no  no  yes no  no  no  no  no  no  no  no  n
 [67] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  n
[100] no  no  <NA> <NA> no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no  no  n
[133] no  no  no  no  <NA> no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  <NA> no  n
[166] no  no  no  <NA> no  no  yes no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  y
[199] no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  yes no  no  no  no  <NA> n
[232] yes no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  n
[265] no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  n
[298] yes no  no  yes no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  n
[331] no  no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  <NA> yes no  yes yes no  no  no  no  no  no  n
[364] no  no  no  no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  n
[397] no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  n
[430] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  n
[463] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  <NA> no  no  no  no  no  no  no  no  n
[496] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  <NA> <NA> no  no  no  no  n
[529] no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  yes no  no  <NA> n
```

- Better overview:

```
> table(ESS$discrim)

    no    yes
361710  25988
>
```

RECODING VARIABLES

- Accessing existing variables is nice, but often we need to create *new* variables based on old ones
- Example: What year did an immigrant arrive in a country? → create new variable `migr_year` based on `livecnta` ("What year you first came to live in country") and `inwyye` (year of interview)
- `ESS$migr_year <- ESS$inwyye - ESS$livecnta`

INDEXING

- Like indexing for one-dimensional vectors, there is also indexing for the two-dimensional data frames

→`dataframe[rows, columns]`

- The first value refers to the *rows* you want to access
- The second value refers to the *columns* you want to access
- So `dataframe[1, 1]` will show the first observation's value of the first variable
- If we are interested in all values of the first observation (row), we can use `dataframe[1, ]`
- If we are interested in all values of the first variable, we can use `dataframe[, 1]`

SUBSETTING DATA

- Subsetting means reducing the data, either drop columns (variables) or rows (observations)
- Example code for keeping variables:

```
    modelvars <- c("trstplc", "discrim", "blgetmg",
"livecnty_comb1", "migr", "continent", "educ")
    ESS_reduc <- ESS[modelvars]
```

- Example code for keeping observations (here: listwise deletion):

```
    ESS_reduc <- ESS__reduc[complete.cases(ESS_reduc[modelvars]), ]
```

FUNCTIONS

WHAT ARE FUNCTIONS?

- You will mostly work with functions in R
- Functions (often) require an input (often between `()`)and will create an output
- Example: `mean()` function: `mean(variable_num)` will return `5.666667` (the mean of 8, 8 and 1)
- Or `range(variable_num)`
- To access R's help, type `?function` (e.g. `?mean`)
- Even better yet: Use Google

PACKAGES

- There are several functions included in “base R” (e. g. `mean()`)
- But a lot of the things that make R *really* interesting have to be loaded into your working environment as *packages*
- Packages are a collection of (user-written) functions
- To install packages, use the `install.packages()` function
- You have to *install* a package only once, but you will have to *load* it every time you want to use it
- To load a package, use `library()`


tidyverse
www.rstudio.com

TIDYVERSE

- "The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures." (https://www.tidyverse.org/)
- The goal of the Tidyverse packages is to make data "tidy"
- Tidy is here defined as
  - Variables in columns
  - Observations in rows
  - Values in cells
- This is how we organized our data frame before

UNTIDY DATA: EUROSTAT

First instance decisions on applications by citizenship, age and sex - annual aggregated data (rounded)
(online data code: MIGR_ASYDCFSTA)
Source of data: Eurostat

Settings: Default

Table | Line | Bar | Map

GEO	SEX	TIME	2017	2018	2019	2020
European Union - 27 countries (from 2020)	Total		504 800	345 575	334 805	309 185
European Union - 27 countries (from 2020)	Males		373 780	247 975	235 635	206 075
European Union - 27 countries (from 2020)	Females		130 680	97 550	99 135	103 085
European Union - 28 countries (2013-2020)	Total		524 070	364 470	348 335	:
European Union - 28 countries (2013-2020)	Males		386 765	260 650	244 905	:
European Union - 28 countries (2013-2020)	Females		136 950	103 760	103 390	:
Belgium	Total		11 460	9 340	10 640	10 650
Belgium	Males		7 795	6 430	7 285	7 270
Belgium	Females		3 665	2 915	3 355	3 380
Bulgaria	Total		3 045	1 370	850	1 375
Bulgaria	Males		2 400	1 190	750	1 300
Bulgaria	Females		640	180	100	70
Czechia	Total		1 045	1 230	1 255	855
Czechia	Males		750	850	935	630
Czechia	Females		290	380	320	225
Denmark	Total		4 510	1 315	1 455	765
Denmark	Males		3 130	845	875	450
Denmark	Females		1 380	470	580	310
Germany (until 1990 former territory of the FRG)	Total		262 575	103 175	83 855	66 120
Germany (until 1990 former territory of the FRG)	Males		193 815	68 315	52 550	42 660
Germany (until 1990 former territory of the FRG)	Females		68 430	34 815	31 285	23 445
Estonia	Total		60	55	45	45
Estonia	Males		35	40	35	25
Estonia	Females		25	15	10	20
Ireland	Total					

WHY SHOULD DATA BE TIDY?

- Easier to read and process
- Standardized workflows of many functions
- A lot of possibilities to manipulate tidy data with the Tidyverse

(SOME) IMPORTANT FUNCTIONS OF THE TIDYVERSE

DPLYR()

- One of the most useful packages of the Tidyverse is `dplyr()`
- It includes
  - `filter()`: Filters observations, e. g.: `filter(ESS, discrim == "yes")`
  - `mutate()`: Create variables, e. g.: `ESS <- mutate(ESS, migr_year = inwyye - livecnta)`
  - `rename()`: Changes name of variable, e. g.: `rename(ESS, discrim = dscrgrp)`
  - `summarize()`: Get some aggregate statistic (example later)
  - …
- `mutate()` most potent when combined `ifelse()` to make conditional statements
- Idea: `ifelse(logical test, value if TRUE, value if FALSE)`
- Example: `ifelse(1 == 1, "This is TRUE", "This is FALSE")`
- Or: `mutate(ESS, discrim = ifelse(dscrgrp == 1, "yes", "no")`

DEFINE MIGRATION BACKGROUND USING `MUTATE()`

```
mutate(ESS, migr =
     ifelse(brncntr == 1 & mocntr == 1 & facntr ==
     1, "native",
          ifelse(brncntr == 2,
          "first gen immigrant",
               ifelse(brncntr == 1
               & (mocntr == 2 | facntr == 2),
               "second gen immigrant",
     NA))))

# 1=yes, 2=no
```

PIPING

%>%

- Let's say we want to get the logarithm of 4, and round it to the first decimal:
  - `x <- log(4)`
  - `round(x, 1)`
- To have less code, R allows *nesting* functions: `round(log(4), 1)`
- But that's hard to read (from inside out), especially when nesting many functions
- Piping allows to read from start to end:

```
4 %>%
 log() %>%
 round(1)
```

- Note: `magrittr` package must be loaded to use piping

COMBINING IT ALL

```
ESS_allwav %<>%
  mutate(migr = ifelse(brncntr == 1
                       & mocntr == 1
                       & facntr == 1, # 1=Yes
                       "native",
                ifelse(brncntr == 2, # 2=no
                       "first gen immigrant",
                       ifelse(brncntr == 1
                              & (mocntr == 2
                              | facntr == 2),
                              "second gen immigrant",
                              NA)))) %>%
  mutate(unempl = ifelse(uempla == 1 | # actively looking for job
                         uempli == 1, # not actively looking for job, 1 = marked
                         "Unemployed",
                         ifelse(is.na(uempla) == T |
                                is.na(uempli) == T,
                                NA,
                                "Not unemployed"))) %>%
  mutate(educ = ifelse(eisced <= 2,
                       "Low (<= ISCED 2)",
                       ifelse(eisced == 3 | eisced == 4,
                              "Medium low (ISCED 3)",
                              ifelse(eisced == 5,
                                     "Medium high (ISCED 4)",
                                     ifelse(eisced >= 6 & eisced <= 50,
                                            "High (>= ISCED 5)",
                                            NA))))) %>%
  mutate(minority = ifelse(blgetmg == 1,
                           "yes",
                           ifelse(blgetmg == 2,
                                  "no",
                                  NA))) %>%
  mutate(discrim = ifelse(dscrgrp == 1,
                          "yes",
                          ifelse(dscrgrp == 2,
                                 "no",
                                 NA)))
```

NOT COVERED HERE, BUT AWESOME

- A bazillion more functions
- RStudio projects (incl. version control using GitHub)
- R Markdown: Create documents, websites, presentations etc. that automatically update numbers and figures if you change the data (no manual copy & pasting anymore!)

LITERATURE

- The slides are inspired by this great intro from Fabio Votta
- Wickham & Grolemund (2017). R for Data Science. O'Reilly