Advanced RStudio Labsessions

Quantitative Methods II

Luis Sattelmayer

2024-01-18

Table of contents

Course Overview

This repository contains all the course material for the RStudio Labsessions for the Spring semester 2024 at the School of Research at SciencesPo Paris. The class follows Brenda van Coppenolle's and Jan Rovny's lecture on Quantitative Methods II. Furthermore, the RStudio part of the course is a direct continuation of Malo Jan's RStudio introduction course. If you feel the need to go back to some basics of general R use, data management or visualization, feel free to check out his course's website. Rest assured, however, that 1) we will recap plenty of things, 2) make slow but steady progress, 3) and come back to the essentials of data wrangling again during the semester while constructing statistical models.

Course Structure

In total we will see each other 6 times. The lessons will be structured in such a way that I will first present something to you and explain my script. Ideally, you will then start coding in groups of 2 and work on exercises related to the topic. You can find more information about the exercises in the subsection "course validation". I will of course be there to help you. The rest you solve at home and send me your final script. At the beginning of each next meeting we will go through the solutions together. Also, I upload my own script before each session, so you can use it as a template when solving the tasks and also later, when the course is over, as a template for further coding (if you like of course…).

Session	Description	Course material
Session 1	RStudio Recap & OLS	
Session 2	Logistic Regressions	
Session 3	Multinomial Regression	
Session 4	Causal Inference	
Session 5	Time Series	
Session 6	Text-as-Data	

Course Validation

In the two weeks between each lecture, you will be given exercises to upload to the designated link for each session. The document where you write the solutions must be written in Markdown format.

I will grade your solutions to my exercises on a 0 to 5 scale. I would like to see that you have done something and hopefully finished the exercise. If you are unable to finish the exercise, it is no problem and I do understand that not everybody feels as comfortable with R as some other people might do. Handing something in is key to getting points! This class can be finished by everyone and I do not want you to worry about your grade too much. But I would like that you all at least try to solve the exercises! Work in groups of **two** and try to hand in something after each session. The precise deadline will be communicated in class, the course's GitHub page and on the Moodle page.

Requirements

You must have downloaded R and RStudio by the beginning of the course (you need to install both!) before our sessions. Please let me know if you encounter any problems during the installation. Here is a quick guide on how to do that: https://rstudio-education.github.io/hopr/starting.html

R and RStudio are both free and open source. You need both of them installed in order to operate with the R coding language.

For R, go on the CRAN website and download the file for your respective operating system: https://cran.r-project.org/ For RStudio, you need to do the same thing by clicking on this link: https://posit.co/products/open-source/rstudio/ RStudio has received a new name recently ("posit") but you will still find all the necessary steps behind this link under the name of RStudio.

Otherwise, there are few prerequisites except that you must bring your computer to the sessions with the required programs installed. I will provide you with datasets in each case and I will explain everything else in the course.

Help and Office Hours

There are unfortunately no regular office hours. But please do not hesitate to reach out, if you have any concerns, questions or feedback for me! My inbox is always open. I tend to reply quickly but in the case that I have not replied in under 48h, simply send the email again. I will not be offended!

Learning how to code and working with RStudio can be a struggle and a tough task. I have started out once like you and I will try to keep that in mind. Feel free to always ask questions in class or if you see me on campus. The most important thing, however, is that you try!

Part I

Session 1

1 RStudio Recap & OLS

1.1 Introduction

This is a short recap of things you have seen last year and will need this year as well. It will refresh your understanding of the linear regression method called *ordinary least squares* (OLS). This script is supposed to serve as a cheat sheet for you to which you can always come back to.

1.2 OLS

As a quick reminder, this is the formula for a basic linear model: $\widehat{Y} = \hat{\alpha} + \hat{\beta}X$.

OLS is a certain kind of method of linear model in which we choose the line which has the least prediction errors. This means that it is the best way to fit a line through all the residuals with the least errors. It minimizes the sum of the squared prediction errors SSE $= \sum_{i=1}^{n} \hat{\epsilon}_i^2$

Five main assumptions have to be met to allow us to construct an OLS model:

1. Linearity: Linear relationship between IVs and DVs
2. No endogeneity between y and x
3. Errors are normally distributed
4. Homoscedasticity (variance of errors is constant)
5. No multicolinearity (no linear relationship between the independent variables)

For this example, I will be working with some test scores of a midterm and a final exam which I once had to work through. We are trying to see if there is a relationship between the score in the midterm and the grade of the final exam. Theoretically speaking, we would expect most of the students who did well on the first exam to also get a decent grade on the second exam. If our model indicates a statistical significance between the independent and the dependent variable and a positive coefficient of the former on the latter, this theoretical idea then holds true.

1.3 Coding Recap

RStudio works with packages and libraries. There is something called Base R, which is the basic infrastructure that R always comes with when you install it. The R coding language has a vibrant community of contributors who have written their own packages and libraries which you can install and use. As Malo does, I am of the `tidyverse` school and mostly code with this package. Here and there, I will, however, try to provide you with code that uses Base R or other packages. In coding, there are many ways to achieve the same goal – and I will probably be repeating this throughout the semester – and we always strive for the fastest or most automated way. But as long as you find a way that works for you, that is fine.

To load the packages, we are going to need:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

Next we will import the dataset of grades.

```
data <- read_csv("course_grades.csv")
```

```
Rows: 200 Columns: 1
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): midterm|final_exam|final_grade|var1|var2

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The path which I specify in the `read_csv` file is short as this quarto document has the same working directory to which the data set is also saved. If you, for example, have your dataset on your computer's desktop, you can access it via some code like this one:

```
data <- read_csv("~/Desktop/course_grades.csv")
```

Or if it is within a folder on your desktop:

```
data <- read_csv("~/Desktop/folder/course_grades.csv")
```

> **!** Important
>
> I will be only working within .Rproj files and so should you. [1] This is the only way to ensure that your working directory is always the same and that you do not have to change the path to your data set every time you open a new RStudio session. Further, this is the only way to make sure that other collaborators can easily open your project and work with it as well. Simply zip the file folder in which you have your code and

You can also import a dataset directly from the internet. Several ways are possible that all lead to the same end result:

```
dataset_from_internet_1 <- read_csv("https://www.chesdata.eu/s/1999-2019_CHES_dataset_mean

# this method uses the rio package
library(rio)
dataset_from_internet_2 <- import("https://jan-rovny.squarespace.com/s/ESS_FR.dta")
```

Let's take a first look at the data which we just imported:

```
# tidyverse
glimpse(data)
```

```
Rows: 200
Columns: 1
$ `midterm|final_exam|final_grade|var1|var2` <chr> "17.4990613754243|15.641013~
```

```
# Base R
str(data)
```

```
spc_tbl_ [200 x 1] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ midterm|final_exam|final_grade|var1|var2: chr [1:200] "17.4990613754243|15.64101334897|17
 - attr(*, "spec")=
```

[1] Malo's explanation and way of introducing you to RStudio projects can be found here.

```
  .. cols(
  ..   `midterm|final_exam|final_grade|var1|var2` = col_character()
  .. )
 - attr(*, "problems")=<externalptr>
```

Something does not look right, this happens quite frequently when saving a csv file. It stands for *comma separated value*. R is having trouble reading this file since I have saved all grades with commas instead of points. Thus, we need to use the `read_delim` function. Sometimes the `read_csv2()` function also does the trick. You'd be surprised by how often you encounter this problem. This is simply to raise your awareness to it!

```
data <- read_delim("course_grades.csv", delim = "|")
```

```
Rows: 200 Columns: 5
-- Column specification --------------------------------------------------------
Delimiter: "|"
dbl (3): midterm, final_exam, final_grade
lgl (2): var1, var2

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(data)
```

```
Rows: 200
Columns: 5
$ midterm     <dbl> 17.499061, 17.744633, 13.931662, 10.706824, 17.118799, 17.~
$ final_exam  <dbl> 15.641013, 18.774437, 14.997858, 11.947943, 15.694728, 17.~
$ final_grade <dbl> 17.63, 14.14, 18.20, 19.85, 14.67, 20.26, 16.90, 13.40, 12~
$ var1        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ var2        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
```

This time, it has been properly imported. But by looking closer at it, we can see that there are two columns in the data frame that are empty and do not even have a name. We need to get rid of these first. Here are several ways of doing this. In coding, many ways lead to the same goal. In R, some come with a specific package, some use Base R. It is up to you to develop your way of doing things.

```r
# This is how you could do it in Base R
data <- data[, -c(4, 5)]

# Using the select() function of the dplyr package you can drop the fourth
# and fifth columns by their position using the - operator and the -c() to
# remove multiple columns
data <- data  |>  select(-c(4, 5))

# I have stored the mutated data set in the old object;
# you can also just transform the object itself...
data |> select(-c(4, 5))

# ... or create a new one
data_2 <- data |> select(-c(4, 5))
```

Now that we have set up our data frame, we can build our OLS model. For that, we can simply use the `lm()` function that comes with Base R, it is built into R so to speak. In this function, we specify the data and then construct the model by using the tilde (~) between the dependent variable and the independent variable(s). Store your model in an object which can later be subject to further treatment and analysis.

```r
model <- lm(final_exam ~ midterm, data = data)
summary(model)
```

```
Call:
lm(formula = final_exam ~ midterm, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-3.6092 -0.8411 -0.0585  0.8712  3.3086

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.62482    0.73212   6.317 1.72e-09 ***
midterm      0.69027    0.04819  14.325  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.34 on 198 degrees of freedom
Multiple R-squared:  0.5089,    Adjusted R-squared:  0.5064
F-statistic: 205.2 on 1 and 198 DF,  p-value: < 2.2e-16
```

Since the `summary()` function only shows us something in our console and the output is not very pretty, I encourage you to use the `broom` package for a nicer regression table.

```
broom::tidy(model)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)    4.62     0.732       6.32 1.72e- 9
2 midterm        0.690    0.0482     14.3  2.10e-32
```

You can also use the `stargazer` package in order to export your tables to text or LaTeX format which you can then copy to your documents.

```
library(stargazer)
```

```
Please cite as:

 Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Tables.

 R package version 5.2.3. https://CRAN.R-project.org/package=stargazer
```

```
stargazer(model, type = "text", out = "latex")
```

```
===============================================
                        Dependent variable:
                    ---------------------------
                              final_exam
-----------------------------------------------
midterm                       0.690***
                               (0.048)

Constant                      4.625***
                               (0.732)

-----------------------------------------------
```

```
Observations                    200
R2                             0.509
Adjusted R2                    0.506
Residual Std. Error      1.340 (df = 198)
F Statistic          205.196*** (df = 1; 198)
================================================
Note:               *p<0.1; **p<0.05; ***p<0.01
```

1.4 Interpretation of OLS Results

How do we interpret this?

- **R2**: Imagine you're trying to draw a line that best fits a bunch of dots (data points) on a graph. The R-squared value is a way to measure how well that line fits the dots. It's a number between 0 and 1, where 0 means the line doesn't fit the dots at all and 1 means the line fits the dots perfectly. R-squared tells us how much of the variation in the dependent variable is explained by the variation in the predictor variables.
- **Adjusted R2**: Adjusted R-squared is the same thing as R-squared, but it adjusts for how many predictor variables you have. It's like a better indicator of how well the line fits the dots compared to how many dots you're trying to fit the line to. It always adjusts the R-squared value to be a bit lower so you always want your adjusted R-squared value to be as high as possible.
- **Residual Std. Error**: The residual standard error is a way to measure the average distance between the line you've drawn (your model's predictions) and the actual data points. It's like measuring how far off the line is from the actual dots on the graph. Another way to think about this is like a test where you want to get as many answers correct as possible and if you are off by a lot in your answers, the residual standard error would be high, but if you are only off by a little, the residual standard error would be low. So in summary, lower residual standard error is better, as it means that the model is making predictions that are closer to the true values in the data.
- **F Statistics**: The F-statistic is like a test score that tells you how well your model is doing compared to a really simple model. It's a way to check if the model you've built is any better than just guessing. A large F-statistic means that your model is doing much better than just guessing.

Part II

Session 2

2 Logistic Regression

2.1 Introduction

You have seen the logic of Logistic Regressions with Professor Rovny in the lecture. In this lab session, we will understand how to apply this logic to R and how to build a model, interpret and visualize its results and how to run some diagnostics on your models. If the time allows it, I will also show you how automatize the construction of your model and run several logistic regressions for many countries at once.

These are the main points of today's session and script:

1. Getting used to the European Social Survey
2. Cleaning data: dropping rows, columns, creating and mutating variables
3. Building a generalized linear model (`glm()`); special focus on logit/probit
4. Extracting and interpreting the coefficients
5. Visualization of results
6. (Automatizing the models for several countries)

2.2 Data Management & Data Cleaning

As I have mentioned last session, I will try to gradually increase the data cleaning part. It is integral to R and operationalizing our quantitative questions in models. A properly cleaned data set is worth a lot. This time we will work on how to drop values of variables (and thus rows of our dataset) which we are either not interested in or, most importantly, because they skew our estimations.

```
# these are the packages, I will need for this session
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
```

```
v purrr     1.0.1
-- Conflicts ---------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

2.3 Importing the data

We have seen how to import a dataset. Set a working directory (`setwd()`) of your choice to the path where the data set of this lecture resides. You can download this dataset from our Moodle page. I have pre-cleaned it a bit. If you were to download this wave of the European Social Survey from the Internet, it would be a much bigger data set. I encourage you to do this and try to figure out ways to manipulate your data but for now, we'll stick to the slightly cleaner version.

```
# importing the data; if you are unfamiliar with this operator |> , ask me or
# go to my document "Recap of RStudio" which you can find on Moodle
ess <- read_csv("ESS_10_fr.csv")
```

```
Rows: 33351 Columns: 25
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (3): name, proddate, cntry
dbl (22): essround, edition, idno, dweight, pspwght, pweight, anweight, prob...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

As you can see from the dataset's name, we are going to work with the *European Social Survey*. It is the biggest, most comprehensive and perhaps also most important survey on social and political life in the European Union. It comes in waves of two years and all the European states which want to pay for it produce their own data. In fact, the French surveys (of which we are going to use the most recent, 10th wave) are produced at SciencesPo, at the Centre de Données Socio-Politiques (CDSP)!

The ESS is extremely versatile if you need a broad and comprehensive data set for both national politics in Europe or to compare European countries. Learning how to use it, how to manage and clean the ess waves will give you all the instruments to work with almost any data set that is "out there". Also, some of you might want to use the ESS waves for your theses or research papers. There is a lot that can be done with it, not only cross-sectionally but also over time. So give it a try :)

Enough advertisement for the ESS, let's get back to wrangling with our data! As always, the first step is to inspect ("glimpse") at our data and the data frame's structure. We do this to see if obvious issues arise at a first glance.

```
glimpse(ess)
```

```
Rows: 33,351
Columns: 25
$ name     <chr> "ESS10e02_2", "ESS10e02_2", "ESS10e02_2", "ESS10e02_2", "ESS1~
$ essround <dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1~
$ edition  <dbl> 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2.2, 2~
$ proddate <chr> "21.12.2022", "21.12.2022", "21.12.2022", "21.12.2022", "21.1~
$ idno     <dbl> 10002, 10006, 10009, 10024, 10027, 10048, 10053, 10055, 10059~
$ cntry    <chr> "BG", "BG", "BG", "BG", "BG", "BG", "BG", "BG", "BG", "BG", "~
$ dweight  <dbl> 1.9393836, 1.6515952, 0.3150246, 0.6730366, 0.3949991, 0.8889~
$ pspwght  <dbl> 1.2907065, 1.4308782, 0.1131722, 1.4363747, 0.5848892, 0.6274~
$ pweight  <dbl> 0.2177165, 0.2177165, 0.2177165, 0.2177165, 0.2177165, 0.2177~
$ anweight <dbl> 0.28100810, 0.31152576, 0.02463945, 0.31272244, 0.12734002, 0~
$ prob     <dbl> 0.0003137546, 0.0003684259, 0.0019315645, 0.0009040971, 0.001~
$ stratum  <dbl> 185, 186, 175, 148, 138, 182, 157, 168, 156, 135, 162, 168, 1~
$ psu      <dbl> 2429, 2387, 2256, 2105, 2065, 2377, 2169, 2219, 2155, 2053, 2~
$ polintr  <dbl> 4, 1, 3, 4, 1, 1, 3, 3, 3, 3, 1, 4, 2, 2, 3, 3, 2, 2, 4, 2, 3~
$ trstplt  <dbl> 3, 6, 3, 0, 0, 0, 5, 1, 2, 0, 5, 4, 7, 5, 2, 2, 2, 2, 0, 3, 0~
$ trstprt  <dbl> 3, 7, 2, 0, 0, 0, 3, 1, 2, 0, 7, 4, 2, 6, 2, 1, 3, 1, 0, 3, 3~
$ vote     <dbl> 2, 1, 1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 1, 1, 2~
$ prtvtefr <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ clsprty  <dbl> 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 2~
$ gndr     <dbl> 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 1~
$ yrbrn    <dbl> 1945, 1978, 1971, 1970, 1951, 1990, 1981, 1973, 1950, 1950, 1~
$ eduyrs   <dbl> 12, 16, 16, 11, 17, 12, 12, 12, 11, 3, 12, 12, 15, 15, 19, 11~
$ emplrel  <dbl> 1, 3, 3, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1~
$ uemp12m  <dbl> 6, 2, 1, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 2~
$ uemp5yr  <dbl> 6, 2, 1, 6, 6, 6, 1, 6, 6, 6, 6, 6, 6, 6, 6, 2, 6, 6, 6, 6, 2~
```

As we can see, there are many many variables (25 columns) with many many observations (33351). Some are quite straight-forward and the name is clear ("essround", "age") and some much less. Sometimes we can guess the meaning of a variable's name. But most of the time - either because guessing is too annoying or because the abbreviation is not making any sense - we need to turn to the documentation of the data set. You can find the documentation of this specific version of the data set in an html-file on Moodle (session 2).

Every (good and serious) data set has some sort of documentation somewhere. If not, it is not a good data set and I am even tempted to say that we should be careful in using it! The

documentation for data sets is called a *code book*. Code books are sometimes well crafted documents and sometimes just terrible to read. In this class, you will be exposed to both kinds of code books in order to familiarize you with both.

In fact, this dataframe still contains many variables which we either won't need later on or that are simply without any information. Let's get rid of these first. This is a step which you can also do later on but I believe that it is smart to this right at the beginning in order to have a neat and tidy data set from the very beginning.

You can select variables (`select()`) right at the beginning when importing the csv file.

```
ess <- read_csv("ESS_10_fr.csv")  |>
  dplyr::select(cntry, polintr, trstplt, trstprt, vote, prtvtefr, clsprty, gndr, yrbrn, ed
```

```
Rows: 33351 Columns: 25
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (3): name, proddate, cntry
dbl (22): essround, edition, idno, dweight, pspwght, pweight, anweight, prob...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

However, I am realizing that when looking at the number of rows that my file is a bit too large for only one wave and only one country. By inspecting the `ess$cntry` variable, I can see that I made a mistake while downloading the dataset because it contains *all* countries of wave 10 instead of just one. We can fix this really easily when importing the dataset:

```
ess <- read_csv("ESS_10_fr.csv") |>
  dplyr::select(cntry, polintr, trstplt, trstprt, vote, prtvtefr, clsprty, gndr, yrbrn, ed
  filter(cntry == "FR")
```

```
Rows: 33351 Columns: 25
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (3): name, proddate, cntry
dbl (22): essround, edition, idno, dweight, pspwght, pweight, anweight, prob...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

This only leaves us with the values for France!

2.3.0.1 2.2 Cleaning our DV

At this point, you should all check out the codebook of this data set and take a look at what the values mean. If we take the variable of `ess$vote` for example, we can see that there are many numeric values of which we can make hardly any sense (without guessing and we don't do this over here) of what they might stand for.

```
summary(ess$vote)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   1.000   1.799   2.000   8.000
```

```
# remember that you can summarize() both dataframes and individual variables
```

Or in a table containing the amount of times that a value was given:

```
table(ess$vote)
```

```
   1    2    3    7    8
1025  590  307    9   46
```

Here we can see that the variable vote contains the numeric values of 1 to 3 and then 7, 8, and 9. If we take a look at the code book, we can see what they stand for:

- 1 = yes (meaning that the respondent voted)
- 2 = no (meaning that the respondent did **not** vote)
- 3 = not eligible to vote
- 7 = Refusal
- 8 = Don't know
- 9 = No answer

The meaning behind the values of 7, 8, and 9 are quite common and you will find them in almost all data sets which were made out of surveys. Respondents might not want to give an answer, the answer was not able to be read, or they indicated that they did not remember.

Spoiler alert: We will work on voter turnout this session: Therefore, we will try to see what makes people vote and what decreases the likelihood that they vote at election day. The question of our dependent variable will thus be: *Has the respondent voted or not?*. Mathematically, this question cannot be answered with a linear regression which uses the OLS method as the dependent variable is *binary* meaning that 0 = has not voted/1 = has voted.

There is only two possible outcomes and the variable is not continuous (one of the assumptions of an OLS).

But if we were to use the variable on voting turnout as it is right now, we would neither have a binary variable nor have a reliable variable as it contains values in which we are both not interested in and that will skew our estimations strongly. In fact, we cannot do a logistic regression (logit) on variables other than binary.

Thus, we first need to transform our dependent variable. We need to get rid of unwanted values and transform the 1s and 2s in 0s and 1s.

First we will get rid of the unwanted values in which we are not interested.

```
# dplyr
ess <- ess |>
  filter(!vote %in% c(3, 7, 8))
```

Here are two other ways to do it:

```
# this one would be in base R
ess[!vote %in% c(3, 7, 8, 9)]

# using the subset() function, this returns a logical vector which elements of
# vote are not in the set of values 3, 7, 8, or 9
ess <- subset(ess, vote %in% c(3,7,8,9) == F)

# Alternatively you can use the %in% function with ! operator as well like this:
ess <- subset(ess, !vote %in% c(3,7,8,9))
```

Quick check to see if we got rid of all the values:

```
table(ess$vote)
```

```
   1    2
1025  590
```

Perfect, now we just need to transform the ones and twos into zeros and ones. This is both out of convention and also to torture you with some more data management. Since we are interested in people who do **not** vote, we will code those people as 0 and those who **did** vote as 1.

```r
ess <- ess |>
  mutate(vote = ifelse(vote == 1, 1, 0))
```

The mutate() function is not perfectly intuitive at first sight. Here, I use the `ifelse()` function within `mutate()` to check if vote is equal to 1, if it is then it will be replaced by 0 and if not it will be replaced by 1.

In Base R, you could do it like this but I believe that the `mutate()` function is probably the most elegant way of doing things… It's up to you though:

```r
# only use the ifelse() function
ess$vote <- ifelse(ess$vote == 1, 1, 0)

# This will leave the value of 1 at 1 and change 2 to 0 for the column vote.
ess$vote[ess$vote == 1] <- 1
ess$vote[ess$vote == 2] <- 0
```

We are *this* close to having finished our data management part and to being able to finally work on our model. But we still have many many variables which are as "untidy" as our initial dependent variable was. If you know what the values are that you do not want – and if you are **absolutely certain** that they are not important in other variables – there is a quick way of getting rid of these. How am I absolutely certain that I can confidently transform the rows containing certain values without losing information? The answer always lies in the code book :)

```r
library(naniar)
unwanted_numbers <- c(66, 77, 88, 99, 7777, 8888, 9999)
ess_clean <- ess |>
  replace_with_na_all(condition = ~.x %in% unwanted_numbers)
```

Lastly, and I promise that this is the last data wrangling part for today and that we will get to our model in a moment, we need to check in the code book if specific values that we cannot simply replace over the whole data frame are still void of interest.

Our independent variables of interest:

1. political interest (polintr): `c(7:9)` needs to be dropped
2. trust in politicians (trstplt): no recoding necessary (unwanted values already NAs)
3. trust in political parties (trstprt): already done
4. feeling close to a party (clsprty): transform 1/2 into 0/1, drop `c(7:8)`
5. gender (gndr): transform into 0/1 and drop the no answers
6. year born (yrbrn): already done
7. years of full-time education completed (eduyrs): already done

We will do every single step at once now using the pipes `%>%` (`tidyverse`) or `|>` (Base R) to have a tidy and elegant way of doing everything at once:

```
# specify a string of numbers we are absolutely certain we won't need
unwanted_numbers <- c(66, 77, 88, 99, 7777, 8888, 9999)

# make sure to create a new object/data frame; if you don't and re-run your code
# a second time, it will transform some mutated values again!
ess_final <- ess |>
  # filtering the dependent variable to get rid of any unnecessary rows
  filter(!vote %in% c(3, 7, 8, 9)) |>
  # using the naniar package, we can transform unwanted values to NAs
  naniar::replace_with_na_all(condition = ~.x %in% unwanted_numbers) |>
  # mutate allows us to transform values within variables into other
  # values or NAs
  # vote as binary 1 (voted) & 0 (abstention)
  mutate(vote = ifelse(vote == 1, 1, 0),
         # replace values 7 to 9 with NAs
         polintr = replace(polintr, polintr %in% c(7:9), NA),
         # replace values 7 to 9 with NAs
         clsprty = replace(clsprty, clsprty %in% c(7:9), NA),
         # recode the variable to 0 and 1
         clsprty = recode(clsprty, `1` = 1, `2` = 0),
         # same for gender
         gndr = recode(gndr, `1` = 0, `2` = 1))
```

2.3.1 Constructing the logit-model

If you have made it this far and still bear with me, you have made it to the fun part! Specifying the model and running it, literally only takes one line of code (or two depending on the amount of independent variables). And as you can see, it is really straightforward. The `glm()` function stands for *generalized linear model* and comes with Base R.

In Professor Rovny's lecture, we have seen that for a Maximum Likelihood Estimation (MLE) you need to know or have an assumption about the distribution of your dependent variable. And according to this distribution, you need to find the right linear model. If you have a binary outcome, your distribution is *binomial*. Within the function, we thus specify the family of the distribution as such. Note that you could also specify other families such as *Gaussian*, *poisson*, *gamma* or many more. We are not going to touch further on that but the `glm()` function is quite powerful. We can specify, within the `family =` argument, that we are doing a logistic regression. This can be done by adding `link = logit` to the argument. If ever you wanted to be precise and call a *probit* or *cauchy* link, it is here that you can specify this. The

standard, however, is set to *logit*, so we would technically not be forced to specify it in this case.

In terms of the model we are building right now, it follows the idea that voting behavior (voted/not-voted) is a function of political interest, trust in politicians, trust in parties, feeling close to a specific party, as well as usual control variables such as gender, age, and education:

By no means is this regression just extensive enough to be published. It is just one example in which I suspect that political interest, trust in politics and politicians, and party affiliation are explanatory factors.

```
logit <- glm(vote ~ polintr + trstplt + trstprt + clsprty + gndr +
               yrbrn + eduyrs,
             data = ess_final,
             family = binomial(link = logit))
```

The object called `logit` contains our model with its coefficients, confidence intervals and many more things that we will play with! But as you can see, the actual construction of the model is more than simple…

```
library(broom)
tidy(logit)
```

```
# A tibble: 8 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept) 100.       8.69       11.5   1.17e-30
2 polintr      -0.436    0.0726     -6.01  1.88e- 9
3 trstplt       0.0829   0.0481      1.72  8.46e- 2
4 trstprt       0.0344   0.0515      0.668 5.04e- 1
5 clsprty       0.710    0.130       5.48  4.32e- 8
6 gndr         -0.0280   0.122      -0.230 8.18e- 1
7 yrbrn        -0.0510   0.00448   -11.4   4.54e-30
8 eduyrs        0.113    0.0195      5.82  5.76e- 9
```

You have seen both the `broom` package as well as `stargazer` in the last session.

```
stargazer::stargazer(logit,
          type = "text",
          dep.var.labels = "Voting Behavior",
          dep.var.caption = c("Voting turnout; 0 = abstention | 1 = voted"),
          covariate.labels = c("Political Interest", "Trust in Politicians",
```

```
                          "Trust in Parties", "Feeling Close to a Party",
                          "Gender", "Year of Birth", "Education")
          )
```

```
=================================================================
                        Voting turnout; 0 = abstention | 1 = voted
                        ----------------------------------------
                                    Voting Behavior
-----------------------------------------------------------------
Political Interest                      -0.436***
                                         (0.073)

Trust in Politicians                     0.083*
                                         (0.048)

Trust in Parties                         0.034
                                         (0.051)

Feeling Close to a Party                 0.710***
                                         (0.130)

Gender                                   -0.028
                                         (0.122)

Year of Birth                           -0.051***
                                         (0.004)

Education                                0.113***
                                         (0.019)

Constant                                100.041***
                                         (8.692)

-----------------------------------------------------------------
Observations                              1,546
Log Likelihood                           -828.497
Akaike Inf. Crit.                       1,672.994
=================================================================
Note:                                *p<0.1; **p<0.05; ***p<0.01
```

2.3.2 4. Interpretation of a logistic regression

Interpreting the results of a logistic regression can be a bit tricky because the predictions are in the form of probabilities, rather than actual outcomes. This sounds quite abstract and you are right, it is abstract. However, with a proper understanding of the coefficients and odds ratios, you can gain insights into the relationship between your independent variables and the binary outcome variable even without transforming your coefficients into more easily intelligible values.

First the really boring and technical definition: The coefficients of a logistic regression model represent the change in the *log-odds* of the outcome for a *one-unit change* in the predictor variable (holding all other predictors constant). The sign of the coefficient indicates the direction of the association: positive coefficients indicate that as the predictor variable increases, the odds of the outcome also increase, while negative coefficients indicate that as the predictor variable increases, the odds of the outcome decrease.

The odds ratio, which can be calculated from the coefficients and we will see how that works in a second (exponentation is the key word), represents the ratio of the odds of the outcome for a particular value of the predictor variable compared to the odds of the outcome for a reference value of the predictor variable. An odds ratio greater than 1 indicates that the predictor variable is positively associated with the outcome, while an odds ratio less than 1 indicates that the predictor variable is negatively associated with the outcome.

It's also important to keep in mind that a logistic regression model makes assumptions about the linearity, independence and homoscedasticity of the data, if these assumptions are not met it can affect the model's performance and interpretation. We will see the diagnostics of logistic regression models again next session.

Is this really dense and did I lose you? It is dense but I hope you bear with me because we will see that it becomes much clearer once we apply this theory to our model but also once we exponentiate the coefficients (reversing the logarithm so to speak) and interpret them as odds-ratios.

But from a first glimpse at our model summary we can see that political interest, trust in politicians, closeness to a party, age and education are all statistically significant, meaning that their p-value is $<.05$! I will not regard any other variable that is not statistically significant as you do not usually interpret non-significant variables.

Next, we can already say that the association of interest, closeness to party and age with voting behavior is negative. This is quite logical and makes sense in our case. If we look at the scales on which these variables are coded (code book!), we can see that the higher the value of the variable, the less interested, close or aged the respondents were. Thus, it decreases their likelihood to vote on voting day. Trust in politicians is coded the other way around. If I had been a little more thorough, it would have been good to put each independent variable on the

same scale… But it means that trust in politicians (in fact meaning that they trust them less) raises the likelihood of not voting somehow (positive association).

2.3.2.1 4.1 Odds-ratio

If you exponentiate the coefficients of your model, you can interpret them as odds-ratios. Odds ratios (ORs) are often used in logistic regression to describe the relationship between a predictor variable and the outcome. ORs are easier to interpret than the coefficients of a logistic regression because they provide a measure of the change in the odds of the outcome for a unit change in the predictor variable.

An OR greater than 1 indicates that an increase in the predictor variable is associated with an increase in the odds of the outcome, and an OR less than 1 indicates that an increase in the predictor variable is associated with a decrease in the odds of the outcome.

The OR can also be used to compare the odds of the outcome for different levels of the predictor variable. For example, an OR of 2 for a predictor variable means that the odds of the outcome are twice as high for one level of the predictor variable compared to another level. Therefore, odds ratios are often preferred to coefficients for interpreting the results of a logistic regression, especially in applied settings.

I will try to rephrase this and make it more accessible so that odds-ratios maybe become more intelligible (they are really nasty statistical stuff):

Imagine you're playing a game where you have to guess whether a coin will land on heads or tails. If the odds of the coin landing on heads is the same as the odds of it landing on tails, then the odds-ratio would be 1. This means that the chances of getting heads or tails are the same. But if the odds of getting heads is higher than the odds of getting tails, then the odds-ratio would be greater than 1. This means that the chances of getting heads is higher than the chances of getting tails. On the other hand, if the odds of getting tails is higher than the odds of getting heads, then the odds-ratio would be less than 1. This means that the chances of getting tails is higher than the chances of getting heads. In logistic regression, odds-ratio is used to understand the relationship between a predictor variable (let's say "X") and an outcome variable (let's say "Y"). Odds ratio tells you how much the odds of Y happening change when X changes.

So, for example, if the odds ratio of X is 2, that means that if X happens, the odds of Y happening are twice as high as when X doesn't happen. And if the odds ratio of X is 0.5, that means that if X happens, the odds of Y happening are half as high as when X doesn't happen.

```
# simply use exp() on the coefficients of the logit
exp(coef(logit))
```

```
 (Intercept)      polintr      trstplt      trstprt      clsprty         gndr
2.800183e+43 6.464189e-01 1.086470e+00 1.034965e+00 2.033083e+00 9.723649e-01
       yrbrn       eduyrs
9.502474e-01 1.120084e+00
```

```
# here would be a second way of doing it
exp(logit$coefficients)
```

```
 (Intercept)      polintr      trstplt      trstprt      clsprty         gndr
2.800183e+43 6.464189e-01 1.086470e+00 1.034965e+00 2.033083e+00 9.723649e-01
       yrbrn       eduyrs
9.502474e-01 1.120084e+00
```

We can also, and should, add the 95% confidence intervals (CI). As a quick reminder, the CI is a range of values that is likely to contain the true value of a parameter (the coefficients of our predictor variables in our case). This comes at a certain level of confidence. The most commonly used levels (attention, this is only a statistical convention!) of confidence are 95% and sometimes 99%.

A 95% CI for a parameter, for example, means that if the logistic regression model were fitted to many different samples of data, the true value of the parameter would fall within the calculated CI for 95% of those samples.

```
# most of the times the extra step in the next lines is not necessary and this
# line of code is enough
exp(cbind(OR = coef(logit), confint(logit)))
```

```
Waiting for profiling to be done...
```

```
                      OR        2.5 %       97.5 %
(Intercept) 2.800183e+43 1.426529e+36 9.118492e+50
polintr     6.464189e-01 5.599783e-01 7.445568e-01
trstplt     1.086470e+00 9.892863e-01 1.194783e+00
trstprt     1.034965e+00 9.351755e-01 1.144451e+00
clsprty     2.033083e+00 1.578414e+00 2.623582e+00
gndr        9.723649e-01 7.653280e-01 1.235391e+00
yrbrn       9.502474e-01 9.418138e-01 9.585074e-01
eduyrs      1.120084e+00 1.078523e+00 1.164144e+00
```

```
# here, however, we must combine both the exponentiate coefficients with the 95% confidenc
# the format() function, helps me to show the numbers without the exponentiated
# "e" and without scientific notation; the round() function within this function gives me
format(round(exp(cbind(OR = coef(logit), confint(logit))), 5),
       scientific = FALSE, digits = 4)
```

```
Waiting for profiling to be done...
```

```
            OR
(Intercept) "       280018299135147974657610234370768785144545280000"
polintr     "                                                   0.6464"
trstplt     "                                                   1.0865"
trstprt     "                                                   1.0350"
clsprty     "                                                   2.0331"
gndr        "                                                   0.9724"
yrbrn       "                                                   0.9503"
eduyrs      "                                                   1.1201"
            2.5 %
(Intercept) "              1426529172752230200649154036500529152.0000"
polintr     "                                                   0.5600"
trstplt     "                                                   0.9893"
trstprt     "                                                   0.9352"
clsprty     "                                                   1.5784"
gndr        "                                                   0.7653"
yrbrn       "                                                   0.9418"
eduyrs      "                                                   1.0785"
            97.5 %
(Intercept) "91184919921641183926834588314691831368354120073216 0.0000"
polintr     "                                                   0.7446"
trstplt     "                                                   1.1948"
trstprt     "                                                   1.1444"
clsprty     "                                                   2.6236"
gndr        "                                                   1.2354"
yrbrn       "                                                   0.9585"
eduyrs      "                                                   1.1641"
```

This exponentiated value, the **odds ratio** (OR), now allows us to say that for a one unit increase in political interest, for example, the odds of voting (versus not voting) decrease. The same goes for the other variables.

The last thing about odds-ratio and I hope that this is the easiest to interpret, is when you try to make percentages out of it:

```r
# the [-1] drops the value of the intercept as it is statistically meaningless
# we put another minus one to get rid of 1 as a threshold for interpreting the
# odds-ratio
# we multiply by 100 to have percentages
100*(exp(logit$coefficients[-1])-1)
```

```
   polintr    trstplt    trstprt    clsprty       gndr      yrbrn     eduyrs
-35.358112   8.646994   3.496496 103.308308  -2.763507  -4.975257  12.008377
```

This allows us to say that being politically uninterested decreases the *odds* of voting by 35%. Much more straightforward right?

2.3.2.2 4.2 Predicted Probabilities

Predicted probabilities also allow us to understand our logistic regression. In logistic regressions, the predicted probabilities and ORs are two different ways of describing the relationship between the predictor variables and the outcome. Predicted probabilities refer to the probability that a specific outcome will occur, given a set of predictor variables. They are calculated using the logistic function, which maps the linear combination of predictor variables (also known as the log-odds) to a value between 0 and 1.

The importance here is that we chose the predictor variables and at which values of those we are trying to predict the outcome. This is what we call "holding independent variables constant" while we calculate the predicted probability for a specific independent variable of interest.

I will repeat this to make sure that everybody can follow along. With the predicted probabilities, we are trying to make out the effect of one specific variable of interest on our dependent variable, while we hold every other variable at their mean, median in some cases or, in the case of a dummy variable, at one of the two possible values. By holding them constant, we can be sure to see the singular effect of our independent variable of interest.

In our case, let "feeling close to a party" (1 = yes; 0 = no) be our independent variable of interest. We take our old `ess_final` dataframe and create a new one. In the `newdata` dataframe, we hold all values at their respective means or put our binary/dummy variables to 1. It is an arbitrary choice to put it to one here. We could also put it to 0. The only variable that we allow to alternate freely to find the predicted probabilities is our variable of interest `clsprty`.

```r
# creating the new dataframe newdata with the old dataframe ess_final
newdata <- with(
  # the initial dataframe contains NAs, we must get rid of them!
```

```
  na.omit(ess_final),
  # construct a new dataframe
  data.frame(
    # hold political interest at its mean
    polintr = mean(polintr),
    # hold trust in politicians at its mean
    trstplt = mean(trstplt),
    # hold trust in parties at its mean
    trstprt = mean(trstprt),
    # let it vary on our IV of interest
    clsprty = c(0, 1),
    # gender is set to 1
    gndr = 1,
    # mean of age
    yrbrn = mean(yrbrn),
    # mean of education
    eduyrs = mean(eduyrs)
    ))
```

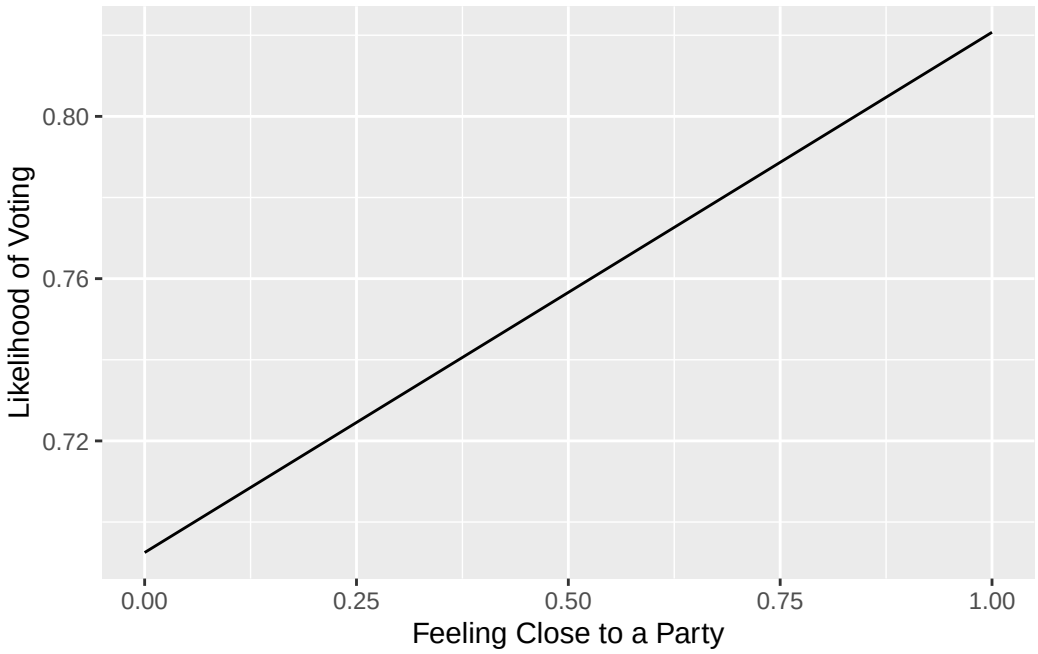
If that all worked out, we can predict the values for this specific independent variable by using the Base R `predict()` function:

```
newdata$preds <- predict(logit, newdata = newdata, type = "response")
```

Now, let's plot the values:

```
library(ggplot2)

ggplot(newdata, aes(x=clsprty, y=preds)) +
    geom_line() +
    ylab("Likelihood of Voting") + xlab("Feeling Close to a Party")
```



We can also do the same thing to see the predicted probability of political interest on voting behavior. This is a bit more interesting as the variable is not binary like `ess$clsprty`:

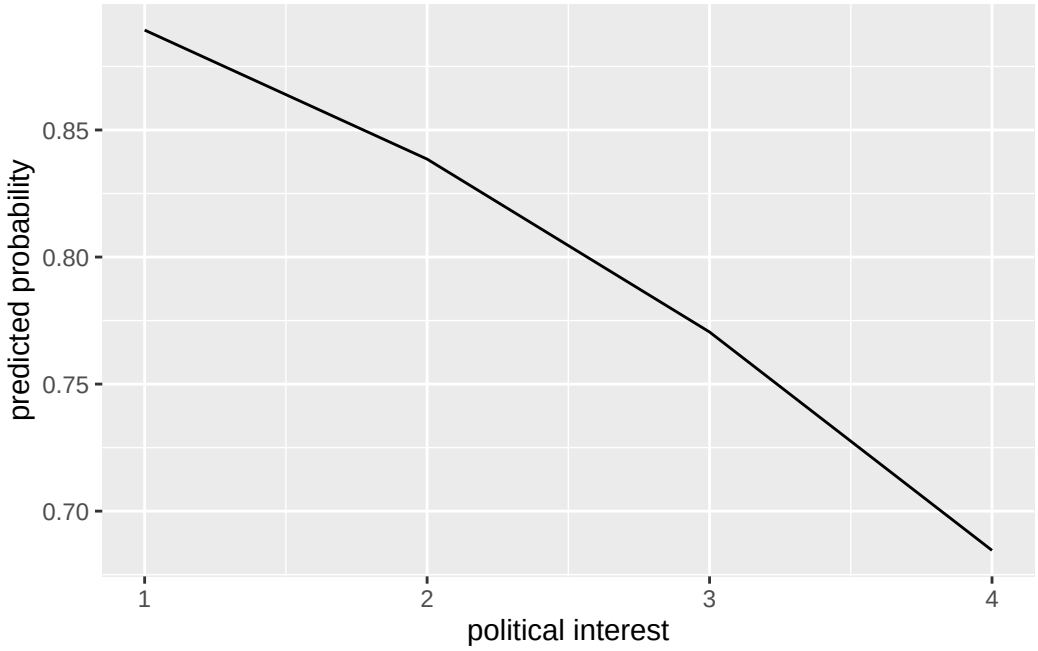
```
# creating the new dataframe newdata with the old dataframe ess_final
newdata_1 <- with(
  # the initial dataframe contains NAs, we must get rid of them!
  na.omit(ess_final),
  # construct a new dataframe
  data.frame(
    # hold political interest at its mean
    polintr = c(1:4),
    # hold trust in politicians at its mean
    trstplt = mean(trstplt),
    # hold trust in parties at its mean
    trstprt = mean(trstprt),
    # let it vary on our IV of interest
    clsprty = 1,
    # gender is set to 1
    gndr = 1,
    # mean of age
    yrbrn = mean(yrbrn),
    # mean of education
```

```
    eduyrs = mean(eduyrs)))
```

```
newdata_1$preds <- predict(logit, newdata = newdata_1, type = "response")
```

Now, let's plot the values:

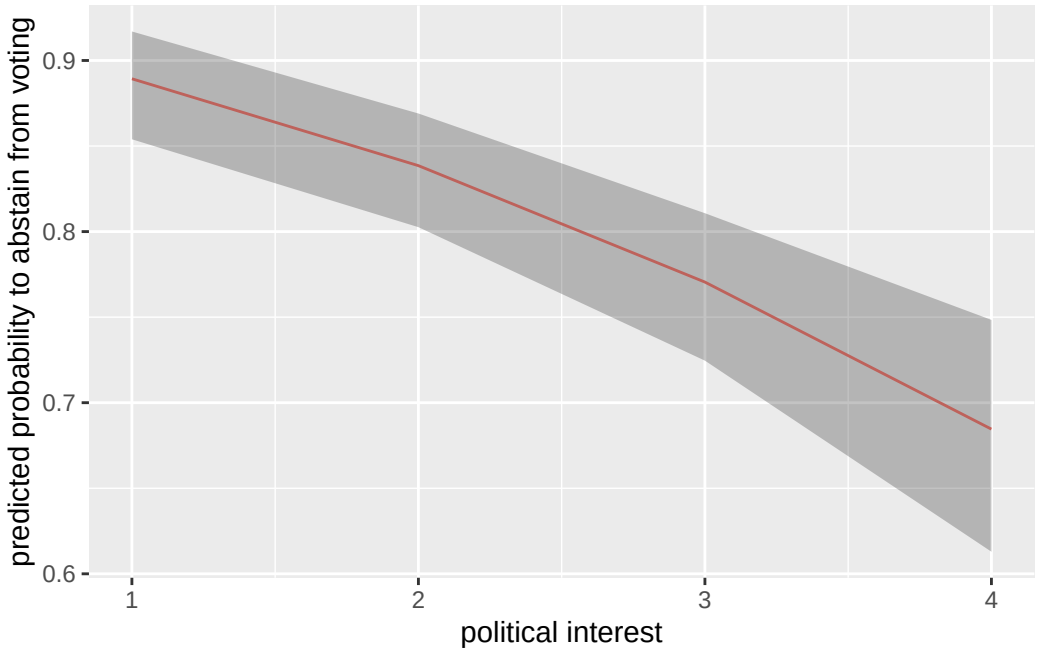
```
library(ggplot2)
ggplot(newdata_1, aes(x=polintr, y=preds)) +
   geom_line() +
   ylab("predicted probability") + xlab("political interest")
```




```
#combines value data frame created above with predicted probabilities evaluated
# at the data values
newdata_1<-cbind(newdata_1, predict(logit, newdata = newdata_1, type = "link", se = TRUE))
```

```
newdata_1 <- within(newdata_1, {
  pp <- plogis(fit)                    # predicted probability
  lb <- plogis(fit - (1.96 * se.fit)) # builds lower bound of CI
  ub <- plogis(fit + (1.96 * se.fit)) # builds upper bound of CI
})
```

```r
ggplot(newdata_1, aes(x=polintr, y=pp)) +
  geom_line(aes(x=polintr, y=pp, color=as.factor(gndr))) +
  geom_ribbon(aes(ymin=lb, ymax=ub), alpha=0.3) +
  theme(legend.position = "none") +
  ylab("predicted probability to abstain from voting") +
  xlab("political interest")
```



2.3.3 5. Making life easiest

You are going to hate me if I tell you that all these steps which we just computed by hand... can be done by using a package. This is only 0.01% of me trying to be mean but mostly because it is extremely helpful and **necessary** to understand what is going on under the hood of *predicted probabilities*. The interpretation of logistic regressions is tricky and if you do not know what you are computing, it is even more complicated.

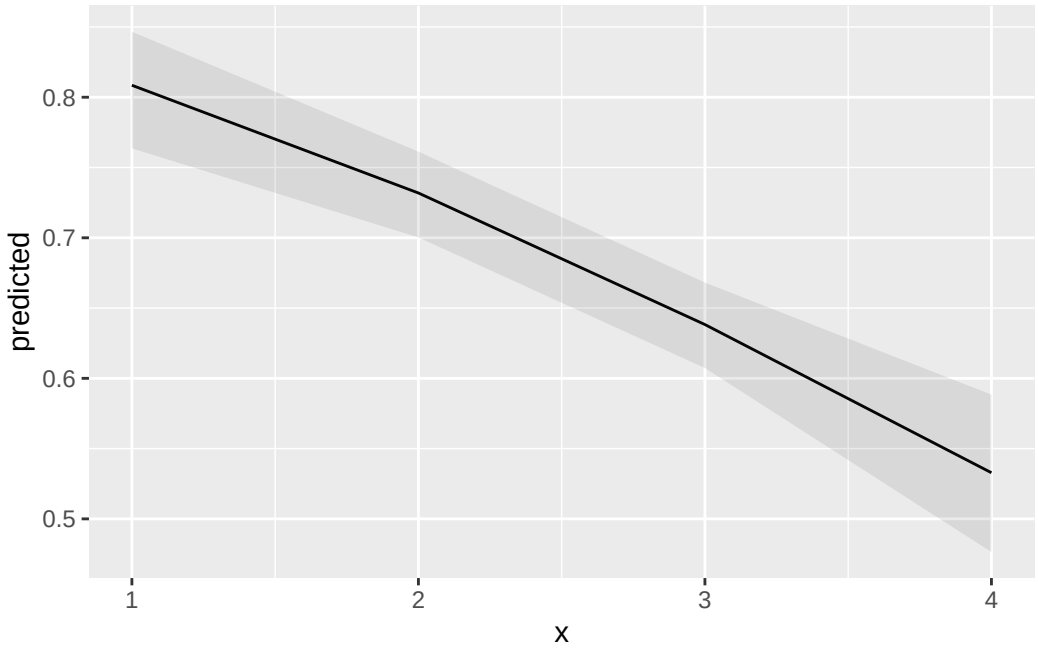
Working with packages is great, and I am aware that I always encourage you to use packages that make your life easier. **But** and this is an important "but" we do not always understand what is going on under the hood of a package. It is like putting your logistic regression into a black box, shaking it really well, and then taking a look at the output and putting it on shaky interpretational terms.

But enough of personal defense, as to why I made you suffer through all this. Here is my code to do most of the steps at once:

```r
# this package contains everything we need craft predicted probabilities and
# visualize them as well
library(ggeffects)

# like the predcit() function of Base R, we use ggpredict() and specify
# our variable of interest
df <- ggpredict(logit, terms = "polintr")

# this is the simplest way of plotting this
ggplot(df, aes(x = x, y = predicted)) +
  # our graph is more or less a line, so geom_line() applies
  geom_line() +
  # geom_ribbon() with the values that ggpredict() provided for the confidence
  # intervals then gives
  # us a shade around the geom_()line as CIs
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .1)
```



And voilà, your output it less than 10 lines of code.

2.3.4 Automatic regressions for several countries (credits to Malo Jan)

This is absolutely only **optional**. I do not ask you to reproduce anything of this at any point in this class. I simply wanted to show you what you can do in R and what I mean when I say that automatizing stuff *makes life easier*.

With a huge thanks to **Malo Jan** who gave me the idea and inspiration for the code because he thought I should show you this, I present you here with a code that does the logistic regression we have been doing but on all countries of the ESS at the same time and then plots us the odds-ratio of our variable of interest "political interest", as well as comparing McFadden pseudo R2 (we'll see this term next session again).

```
# library for McFadden pseudo R2
library(pscl)
```

```
Classes and Methods for R developed in the
Political Science Computational Laboratory
Department of Political Science
Stanford University
Simon Jackman
hurdle and zeroinfl functions by Achim Zeileis
```

```
library(broom)

country_model <- function(df) {
  glm(vote ~ polintr + trstplt + trstprt + clsprty + gndr + yrbrn + eduyrs,
      family = binomial(link = "logit"), data = df)
}

ess <- read_csv("ESS_10_fr.csv") |>
  select(cntry,vote, polintr, trstplt, trstprt, clsprty, gndr, yrbrn, eduyrs)
```

```
Rows: 33351 Columns: 25

-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (3): name, proddate, cntry
dbl (22): essround, edition, idno, dweight, pspwght, pweight, anweight, prob...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
ess_model <- ess |>
    mutate(vote = ifelse(vote == 1, 1, 0)) |>
    filter(vote %in% c(0:1),
           polintr %in% c(1:4),
           clsprty %in% c(1:2),
           trstplt %in% c(0:10),
           trstprt %in% c(0:10),
           gndr %in% c(1:2),
           yrbrn %in% c(1900:2010),
           eduyrs %in% c(0:50)) |>
    group_by(cntry) |>
    nest() |>
    mutate(
        model = map(data, country_model),
        tidied = map(model, ~ tidy(.x, conf.int = TRUE, exponentiate = TRUE)),
        glanced = map(model, glance),
        augmented = map(model, augment),
        mcfadden = map(model, ~ pR2(.x)[4])
    )
```

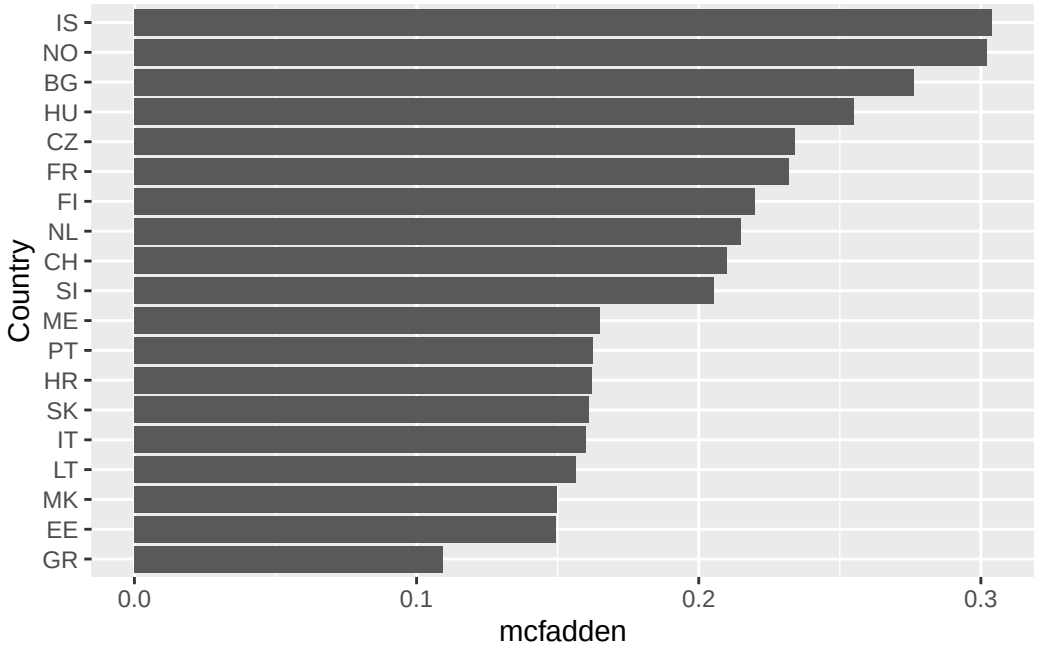
```
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
fitting null model for pseudo-r2
```

```
# Comparing AICs
pR2(logit)
```

```
fitting null model for pseudo-r2
```

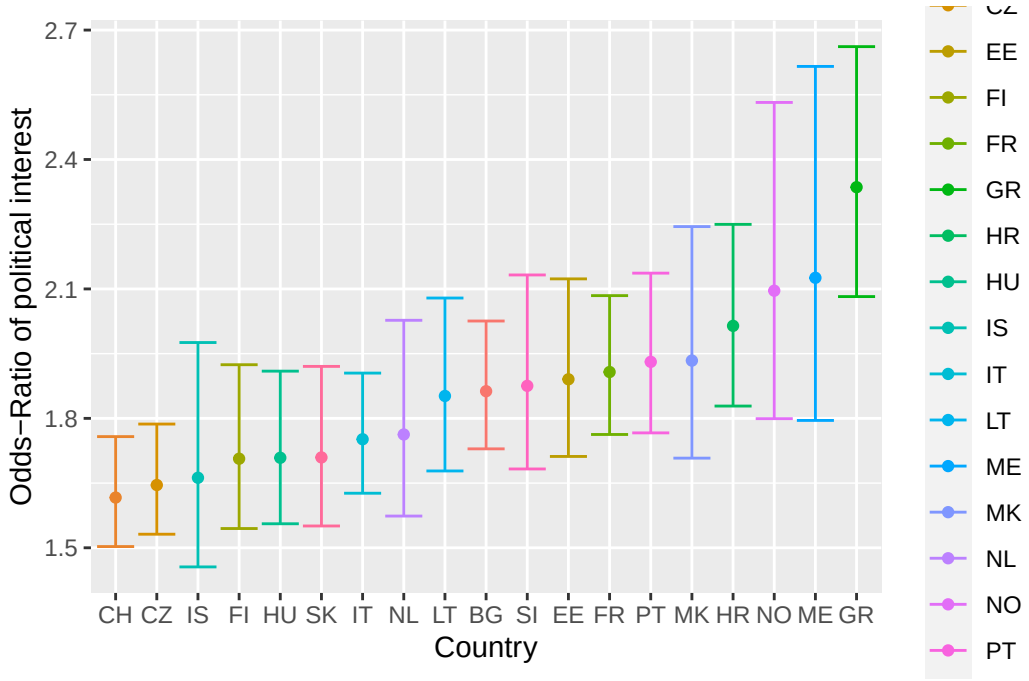
```
          llh       llhNull            G2      McFadden          r2ML
-828.4972459 -1011.5800658   366.1656399     0.1809870     0.2108881
        r2CU
   0.2889617
```

```
ess_model |>
    unnest(mcfadden) |>
    ggplot(aes(fct_reorder(cntry, mcfadden), mcfadden)) +
    geom_col() + coord_flip() +
    scale_x_discrete("Country")
```




```
# Comparing coefficients
ess_model |>
    unnest(tidied) |>
    filter(term == "polintr") |>
    ggplot(aes(
```

```
    reorder(cntry, estimate),
    y = exp(estimate),
    color = cntry,
    ymin = exp(conf.low),
    ymax = exp(conf.high)
)) +
geom_errorbar() +
geom_point() +
scale_x_discrete("Country") +
ylab("Odds-Ratio of political interest") +
xlab("Country")
```



Part III

Session 3

3 Multinomial Regressions in R

4 The Logic of multinomial (logistic) Regressions

In this script, I will show you how to construct a multinomial logistic regression in R. For this, we will work on the *European Social Survey* (ESS) again. I have chosen four countries out of which you will be able to choose one later one when I ask you to work on some exercises. For now, I will mainly work on Germany. One of the classic applications of multinomial models in political science is the question of voting behavior, more precisely vote choice. Last week, we have seen the linear model of a logistic regression (logit). It is used in cases when our dependent variable (DV) is binary (0 or 1; true or false; yes or no) which means that we are not allowed to use OLS. The idea of logit can be extended to *unordered categorical or nominal variables* with more than **two** categories, e.g.: Vote choice, Religion, Brands…

Instead of one equation modelling the log-odds of $P(X = 1)$, we do the same thing but for the amount of categories that we have. In fact, this means that a multinomial model runs several single logistic regressions on something we call a *baseline.* R will choose this baseline to which the categorical values of our DV will then relate. But we can also change it (this is called releveling). This allows us to make very interesting inferences with categorical (or ordinal) variables. If this sounds confusing, you should trust me when I tell you that this will become more straightforward in a second!

However, this also makes the interpretation of these models a bit intricate and opaque at times. Nevertheless, you will see that once you have understood the basic idea of a multinomial regression and how to interpret the values in accordance to the baseline, it is not much different from logistic regressions on binary variables (and in my eyes even a bit simpler…). If the logic of logit is not 100% clear at this point, I recommend you go back to last session's script on logit and work through my explanations. And if that does not help, try to follow this lecture attentively. As I said, the logic is the same, so I will repeat myself :) And if it is still unclear, you can always ask in class or come see me after the session!

But enough small talk, let's first do some data wrangling which you all probably dread at this point…

5 Data Management for Multinomial Regression

As I have said, we will work on voting choice in four different countries. I selected Denmark and Germany. Germany I have chosen because I was working on this model a couple of months ago and Denmark is for fun.

The data which we will use for this session is the 9th round of the ESS published in 2018. The goal of this session is to understand predictors that tell us more about why people vote for Populist Radical-Right Parties, henceforth called *PRRP* (Mudde, 2007). For this I have two main hypotheses in mind, as well as some predictors which I know are important based on the literature. Finally we also need some control variables which we need to control for in almost any regression analysis using survey data.

My two hypotheses (H_1 and H_2) are as follows:

H_1: Thinking that immigrants enrich a country's culture, decreases the likelihood of voting for PRRPs.

H_2: Having less trust in politicians increases the likelihood of voting for PRRPs than voting for other parties.

Now you might notice two things. First, my hypotheses are relatively self-explanatory and you are absolutely right. They are more than that, they are perhaps even self-evident. But to this, I would just reply that this is supposed to be an easy exercise which is supposed to expose you to a multinomial regression and the logic of it. Second, you might see that my hypotheses are relatively broadly formulated. This is because I would like you, later in class, to choose one of the countries of the 9th wave of the ESS and build a model yourselves. By giving you broad hypotheses, you can do this ;) And again, it is only an exercise. We will speak about the good formulation of testable hypotheses again.

```
# read_csv from the tidyverse package
ess <- read_csv("ESS9e03_1.csv") |>
  # dplyr allows me to select only those variables I want to use later
  select(cntry, prtvtdfr, prtvede1, prtvtddk, prtvtdpl,
         imueclt, yrbrn, eduyrs, hinctnta, stflife, trstplt,
```

```
        blgetmg, gndr) |>
  # based on the selected variables, I filter the dataframe so that I am only
  # left with the data for Germany and Denmark
  filter(cntry %in% c("DE", "DK"))
```

```
Rows: 49519 Columns: 572
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (10): name, proddate, cntry, ctzshipd, cntbrthd, lnghom1, lnghom2, fbrn...
dbl (562): essround, edition, idno, dweight, pspwght, pweight, anweight, pro...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Again, every transformation and mutation of variables which you see below is done based on my knowledge of the dataset which I solely gained from looking at the code book. The code book can be found on the Moodle page (or the ESS' website). It is highly important that you get used to reading a code book in general but especially to familiarize yourselves with the data which you will use by looking at the way that the variables are coded **in the code book**. There, for example, you will find information on the numeric values which are stored in the variables `prtvtdfr`, `prtvede1`, `prtvtddk` and `prtvtdpl`. They all stand for a category or, in our case, a party name which you can only identify if you open the code book. You will see that I only selected some parties in the `mutate()` function below. This is more or less to get rid of those parties that did not make it into the national parliament at the last national election of each country.

You have seen a similar chunk of code in the last script. See how, once you have a code that works for one dataset, you can use it again?

```
# I specify a string of numbers I am 100 % certain that I am not going to need
unwanted_numbers <- c(66, 77, 88, 99, 7777, 8888, 9999)

# cleaning the dependent variables all over the dataframe
ess_clean <- ess |>
  # getting rid of specified unwanted numbers (transforming them into NAs)
  naniar::replace_with_na_all(condition = ~.x %in% unwanted_numbers) |>
  # get rid of unwanted parties by transforming the numeric values into NAs
  # repeat it for the party voted variables per country
    mutate(prtvtdfr = replace(prtvtdfr, prtvtdfr %in% c(1, 2, 10, 12:99), NA),
           prtvede1 = replace(prtvede1, !prtvede1 %in% c(1:6), NA),
           prtvtddk = replace(prtvtddk, !prtvtddk %in% c(1:10), NA),
```

```
            prtvtdpl = replace(prtvtdpl, !prtvtdpl %in% c(1:8), NA),
            # get rid of unwanted values indicating no response etc
            blgetmg = replace(blgetmg, !blgetmg %in% c(1:2), NA),
            # gender recoded to 1 = 0, 2 = 1 (my personal preference)
            gndr = recode(gndr, `1` = 0, `2` = 1))
```

In fact, you could already build the model now and start the multinomial regression. However, I add an additional data management step by placing the numeric values of the election variable in a new variable called `vote_de`, where I convert the numeric values to character values and at the same time give them the names of the parties. This will automatically transform NAs in all the rows in which the country is not that in which the person has voted.

But more importantly, once I run the regression, it will display the parties' names instead of the numbers. This means that I won't have to go back to the code book every time to check what the 1s or 2s correspond to.

```
# this is simple base R creating a new column/variable with character
# values corresponding to the parties' names behind the numeric values
ess_clean$vote_de[ess_clean$prtvede1==1]<-"CDU/CSU"
ess_clean$vote_de[ess_clean$prtvede1==2]<-"SPD"
ess_clean$vote_de[ess_clean$prtvede1==3]<-"Die Linke"
ess_clean$vote_de[ess_clean$prtvede1==4]<-"Grüne"
ess_clean$vote_de[ess_clean$prtvede1==5]<-"FDP"
ess_clean$vote_de[ess_clean$prtvede1==6]<-"AFD"
```

Here is a way to mutate all the variables at once. However, this somehow creates conflicts with a package used further below.

```
ess_test <- ess |>
  mutate(
    vote_dk = case_when(prtvtddk == 1 ~ "Socialdemokratiet",
                        prtvtddk == 2 ~ "Det Radikale Venstre",
                        prtvtddk == 3 ~ "Det Konservative Folkeparti",
                        prtvtddk == 4 ~ "SF Socialistisk Folkeparti",
                        prtvtddk == 5 ~ "Dansk Folkeparti",
                        prtvtddk == 6 ~ "Kristendemokraterne",
                        prtvtddk == 7 ~ "Venstre",
                        prtvtddk == 8 ~ "Liberal Alliance",
                        prtvtddk == 9 ~ "Enhedslisten",
                        prtvtddk == 10 ~ "Alternativet",
                        TRUE ~ NA_character_),
    vote_de = case_when(prtvede1 == 1 ~ "CDU/CSU",
```

```
                prtvede1 == 2 ~ "SPD",
                prtvede1 == 3 ~ "Die Linke",
                prtvede1 == 4 ~ "Grüne",
                prtvede1 == 5 ~ "FDP",
                prtvede1 == 6 ~ "AFD"))
```

6 Constructing the Model

Now that the data management process is finally over, we can specify our model. For this, you need to install the `nnet` package and load it to your library. Once this is done, we will take the exact same steps as you would do for an OLS or logit model. You specify your DV followed by a `~` and then you only need to add all your IVs. Lastly, you need to specify the data source. `Hess = TRUE` will provide us with a Hessian matrix that we need for a package later. If you don't know what that is... that is absolutely fine!

```
library(nnet)
model_de <- multinom(vote_de ~ imueclt  + stflife + trstplt + blgetmg +
                     gndr + yrbrn + eduyrs + hinctnta,
                      data = ess_clean,
                      Hess = TRUE)
```

```
# weights:  60 (45 variable)
initial  value 2512.046776
iter  10 value 2043.014063
iter  20 value 2023.586615
iter  30 value 1980.080494
iter  40 value 1938.289705
iter  50 value 1927.868641
iter  60 value 1926.043042
iter  70 value 1925.949503
iter  80 value 1925.873772
final  value 1925.814902
converged
```

```
model_dk <- multinom(vote_dk ~ imueclt  + stflife + trstplt + blgetmg + gndr +
                     yrbrn + eduyrs + hinctnta,
                     data = ess_test,
                     Hess = TRUE)
```

```
# weights:  100 (81 variable)
initial  value 2795.338303
```

```
iter  10 value 2506.234222
iter  20 value 2461.859464
iter  30 value 2453.873307
iter  40 value 2432.673849
iter  50 value 2358.985280
iter  60 value 2304.034636
iter  70 value 2276.886671
iter  80 value 2259.889542
iter  90 value 2249.913529
iter 100 value 2246.110042
final  value 2246.110042
stopped after 100 iterations
```

6.0.1 Re-leveling your DV

In my case, the German PRRP is called *Alternative für Deutschland* meaning it starts with an "A". R tends to take the alphabetical order as a criterion for the baseline meaning that the baseline for your multinomial model is chosen based on the party which comes first in alphabetical order. Depending on what you want to show, you might want to change the baseline which we can do with the `relevel()` function. Let's say we are not interested in vote choice regarding the PRRP but conservative parties and thus want to put the German Christian conservative party, the CDU/CSU, as a baseline. Here is how we could do this in R:

```r
# you need to specify your DV as a factor for this; further, the ref must
# contain the exact character label of the party
ess_clean$vote_de <- relevel(as.factor(ess_clean$vote_de), ref = "CDU/CSU")
```

7 Interpreting a Multinomial Model

You already know that I like the `stargazer` package for displaying a regression table. This time I paid attention to what level of statistical significance leads to a star (*). I changed it so that, like in the `summary()` function, p-values below 0.05 will be used as the minimum level of statistical significance instead of 0.1. `dep.var.caption =` allows be to specify a caption for our DV and we can use our own labels for the IVs instead of the variables' names by using the `covariate.labels =` argument.

I have specified in the first chunk of code which arguments concern the generated output in LaTeX. I still recommend you start learning how to write papers in LaTeX. This is just to say that some arguments are not useful at all when `type = "text`. But LaTeX generates more beautiful tables ;)

```
# specifying the object in which the model is stored
stargazer::stargazer(model_de,
                     # adding a title to the table
                     title = "Multinomial Regression Results Germany",
                     # the type of the output, it is going to be in LaTeX
                     type = "latex",
                     # some LaTeX information
                     float = TRUE,
                     # font size of the LaTeX table
                     font.size = "small",
                     # column width in final LaTeX table
                     column.sep.width = "-10pt",
                     # specifying the p-values which lead to stars in our
                     # table
                     star.cutoffs = c(.05, .01, .001),
                     # caption for the DV
                     dep.var.caption = c("Vote Choice"),
                     # labels for our IVs; must be in the same order as our
                     # IVs in the initial model
                     covariate.labels = c("Positivity Immigration",
                             "Satisfaction w/ Life", "Trust in Politicians",
                             "Ethnic Minority", "Gender", "Age", "Education",
                             "Income"))
```

```r
# the annotations of the above model would be the same for this model
stargazer::stargazer(model_dk,
                     title = "Multinomial Regression Results Denmark",
                     type = "latex",
                     float = TRUE,
                     font.size = "tiny",
                     star.cutoffs = c(.05, .01, .001),
                     dep.var.labels = c("Germany"),
                     dep.var.caption = c("Vote Choice"),
                     covariate.labels = c("Positivity Immigration",
                             "Satisfaction w/ Life", "Trust in Politicians",
                             "Ethnic Minority", "Gender", "Age", "Education",
                             "Income"))
```

The output of these `stargazer` table can be found on the next page. They are in LaTeX format! The format of the regression table on our Danish model is not ideal since the names of the parties are quite long and overlap. Blame this on my lack of knowledge of abbreviations of Danish parties...