The software environment R is widely used for data analysis and data visualization in the social sciences and beyond. Additionally, it is becoming increasingly popular as a tool for data and file management.

Readers will learn about:

* the workflow for organizing and conducting complex analyses in R
* creating, editing, and accessing directory hierarchies and their contents
* data merging, data management and data manipulation using tidy R and base R
* the basics of programming and debugging

To illustrate these steps, we will work through an example from comparative political behavior – the question under which conditions center-left parties can successfully mobilize votes among the unemployed. To tackle this question, we will combine micro-level voting data from Round 9 of the [European Social Survey](https://www.europeansocialsurvey.org/data/) with contextual data on election dates from [ParlGov](http://www.parlgov.org/#data), party positions from the [Manifesto Project](https://manifesto-project.wzb.eu/datasets), and unemployment rates from the [World Bank](https://data.worldbank.org/indicator/SL.UEM.TOTL.ZS).

**Setup**

Over the course of this tutorial, we will use the following R packages:

Code: Packages used in this tutorial

## Save package names as a vector of strings

pkgs <-

c(

"foreign", ### read data stored in various formats

"readstata13", ### read data stored by Stata 13-16

"reshape2", ### flexibly reshape data

"countrycode", ### convert country names and country codes

"lubridate", ### dates and time

"dplyr", ### tools for data manipulation

"magrittr", ### piping operations

"tidyr", ### tool to deal with messy data (and get "tidy data")

"ggplot2", ### data visualization using a grammar of graphics

"fabricatr" ### imagine your data before you collect it

)

## Install uninstalled packages

lapply(pkgs[!(pkgs %in% installed.packages())], install.packages)

## Load all packages to library and adjust options

lapply(pkgs, library, character.only = TRUE)

**Workflow for a Reproducible Data Project**

Even the setup of a reproducible data project should follow reproducible steps. And since we need to repeat these steps every time we start a new data project, it is a good idea to keep everything simple and to automate what can be automated. For the following steps to work you need three things:

Let’s start a new project:

* Start a new project on GitHub:
  + Login to your GitHub account.
  + Click on New on the left side where your repositories are displayed.
  + Give your project repository a meaningful name. For this course you could name it: efficient-data-management.
  + Click Create repository.
  + Copy the URL to the repository, it will look something like this: <https://github.com/>/efficient-data-management.git
* Open the project in RStudio:
  + Open RStudio.
  + Click on File > New Project....
  + Create a new project from Version Control.
  + Choose Git.
  + Paste the URL to the repository in Repository URL.
  + Choose a folder on your computer where the project will be stored locally.
  + And finally click on Create.

**Initializing the project with a .Rprofile.**

Now you have a new Rproject that is linked with GitHub. Before start coding, we want to have an efficient folder structure. Ideally, we want to use the same structure for all of our projects. This way, it is easier to focus on the things that really matter – like producing interesting research. To that end, we initialize our Rproject with a .Rprofile that automates the process of setting up our project structure.

* Initialize your Rproject with a .Rprofile:
  + If you follow these steps for the very first time, install the renv package in R.
  + In RStudio, with your new Rproject open, open a new Text File.
  + Paste the following code and save it as .Rprofile.
  + Now close RStudio and re-open your project from your project folder.

.First <- function() {

dir.create(paste0(getwd(), "/figures"), showWarnings = F)

dir.create(paste0(getwd(), "/processed-data"), showWarnings = F)

dir.create(paste0(getwd(), "/raw-data"), showWarnings = F)

dir.create(paste0(getwd(), "/scripts"), showWarnings = F)

dir.create(paste0(getwd(), "/manuscript"), showWarnings = F)

if (!("renv" %in% list.files())) {

renv::init()

} else {

source("renv/activate.R")

}

cat("\nWelcome to your R-Project:", basename(getwd()), "\n")

}

R will automatically source the .Rprofile at startup. Here, we first create some folders with dir.create(). Don’t worry, folders will not be created if they already exist, so this will not overwrite existing files. We then display a nice welcome message with cat() and activate the [renv](https://cran.r-project.org/web/packages/renv/index.html) package.

After re-opening the Rproject, you should now see the project setup in your folder.

**What is renv?**

In the past, sharing your project with someone else and getting R to show the exact same behavior on different machines could be a pain. The renv package makes this super easy. It creates a local package directory for your project. This means that it keeps track of all the packages and package versions that you use in your project. If someone else wants to work with the exact same package environment to reproduce your data project, they can easily restore it from your renv package directory.

**Leveraging git and renv at the end of a working session**

Now that we have our nice project setup, we should not forget to leverage it. At the end of a working session you should follow the following steps:

* Create a renv::snapshot() to save all the packages you used to your local package directory.
* Commit all your changes to git. This can be easily done by using the Git pane in RStudio.
* Push everything to GitHub.

Whenever you re-open your Rproject, make sure to start your working session with a Pull from GitHub. That way, you will always work with the most recent version of your project.

**Managing and Accessing Directories**

Now that we have a directory structure for our project, we need to learn how to work with it. What goes into the different folders should be rather self-explanatory. Yet, it is good practice to also include a simple ReadMe file in each of the folders to explain what it should contain.

**What is a working directory?**

Your working directory is the folder on your computer that R will use to read and save files. With your .Rproject and .Rmd files, you do not have to worry about this too much. There is only one golden rule: **Never change your working directory** (especially, never to absolute paths specific to your computer).

**Where is your working directory?**

When we created our .Rproject we had to decide where all the files should be stored. If you open your .Rproject, the working directory will automatically be set to this folder. If you have a .Rmd file somewhere in your project (e.g. in the /manuscript folder) RStudio will automatically set the working directory the folder containing the .Rmd.

**What does this all mean?**

This means you have to access all data or code (stored in the folders of your project) relative to your current working directory.

* If you work with a simple .R script (and you store the file in /scripts) in your .Rproject and you want to open data stored in your /raw-data folder: Simply open the file directly from the folder e.g. with read.csv("raw-data/20200127\_parlgov.csv")
* If you work in a .Rmd file in your manuscript folder you have to navigate up one directory level by using ../: Open the file directly from the folder with read.csv("../raw-data/20200127\_parlgov.csv")
* Similarly, if you want to store processed data or a figure you would directly store it to the right folder.

**Data Management and Data Manipulation**

**Getting to know the tidyverse**

The R universe basically builds upon two (seemingly contradictive) approaches: **base R** and the **tidyverse**. While these two approaches are often seen as two different philosophies, they can form a symbiosis. We therefore recommend to pick whichever works best for you – or to combine the two.



The logic of tidyverse is fairly simple: As you can see in the graphic above, when using the tidyverse style, you start with your main object (me) and pipe (%>%) through this object by filtering, selecting, renaming, … parts of it. With these pipe operators, you read the code from left to right (or, across multiple lines, from top to bottom). In base R you would – in contrast – wrap the commands around your main object and thus read the code from the inside out.

We first familiarize ourselves with the basic logic of the tidyverse style using some examples. For this, we use some generated data.

Code: Generate data

# We set a seed to make our dataset reproducible.

set.seed(68159)

data <- fabricate(

countries = add\_level(

N = 10,

month = recycle(month.abb),

gdppercapita = runif(N, min = 10000, max = 50000),

unemployment\_rate = 5 + runif(N, 30,50) + ((gdppercapita > 50) \* 10),

life\_expectancy = 50 + runif(N, 10, 20) + ((gdppercapita > 30000) \* 10)

)

)

# Drop countries

data <- data %>% select(-countries)

# We add artificial NAs

data <- data %>%

mutate(

gdppercapita = ifelse(gdppercapita > 40000 &

gdppercapita < 44000, NA, gdppercapita),

unemployment\_rate = ifelse(

unemployment\_rate > 50 &

unemployment\_rate < 54,

NA,

unemployment\_rate

)

)

# Have a look at the data

data

## month gdppercapita unemployment\_rate life\_expectancy

## 1 Jan 31819.79 48.90126 70.19808

## 2 Feb 45485.76 58.51337 76.16142

## 3 Mar 18327.78 55.89483 67.68735

## 4 Apr 22275.86 48.61502 60.98838

## 5 May 23069.79 47.18671 66.39749

## 6 Jun 34896.12 NA 72.52299

## 7 Jul 14370.76 58.33964 66.42222

## 8 Aug NA 59.60927 74.91735

## 9 Sep NA NA 73.82523

## 10 Oct 24195.20 NA 65.26416

# If your dataset becomes larger, the command "head(data)" is a useful alternative

# to get a quick glance at the data. It shows by default the first 6 rows.

**Filtering the data**

Our dataset contains information on countries (numeric), months (characters), GDP per capita, unemployment rates, and life expectancies (all numeric). Suppose we are only interested in data from January. For this purpose, we can use the filter() function, provided by the package dplyr.

data %>%

filter(month == "Jan")

## month gdppercapita unemployment\_rate life\_expectancy

## 1 Jan 31819.79 48.90126 70.19808

How would you do it in base R?

data[data$month == "Jan", ]

RStudio also provides **shortcuts** that allow you to write the pipe operator (%>%) quickly:

* Shift + Cmd + M (Mac)
* Shift + Ctrl + M (Windows)

**Selecting specific variables**

Sometimes, you want to select specific variables. We are interested in the unemployment rate and the months but not so much in the other variables. dplyr’s command select() allows you to do exactly this.

data %>%

select(month, unemployment\_rate) %>%

head()

## month unemployment\_rate

## 1 Jan 48.90126

## 2 Feb 58.51337

## 3 Mar 55.89483

## 4 Apr 48.61502

## 5 May 47.18671

## 6 Jun NA

How would you do it in base R?

head(data[, c("month", "unemployment\_rate")])

**Arranging variables**

Let’s say we want to know the largest unemployment rate and the lowest unemployment rate throughout the entire dataset. This can be done with the arrange() command that is added with another %>% operator to our previous code.

By default, the arrange() function sorts the data in ascending order. To display the data in descending order, we add desc().

# Arrange in ascending order

data %>%

select(month, unemployment\_rate) %>%

arrange(unemployment\_rate)

## month unemployment\_rate

## 1 May 47.18671

## 2 Apr 48.61502

## 3 Jan 48.90126

## 4 Mar 55.89483

## 5 Jul 58.33964

## 6 Feb 58.51337

## 7 Aug 59.60927

## 8 Jun NA

## 9 Sep NA

## 10 Oct NA

# Arrange in descending order

data %>%

select(month, unemployment\_rate) %>%

arrange(desc(unemployment\_rate))

## month unemployment\_rate

## 1 Aug 59.60927

## 2 Feb 58.51337

## 3 Jul 58.33964

## 4 Mar 55.89483

## 5 Jan 48.90126

## 6 Apr 48.61502

## 7 May 47.18671

## 8 Jun NA

## 9 Sep NA

## 10 Oct NA

How would you do it in base R?

# Arrange in ascending order

data[order(data$unemployment\_rate), c("month","unemployment\_rate")]

# Arrange in descending order

data[order(data$unemployment\_rate, decreasing = TRUE),

c("month","unemployment\_rate")]

As we can see, the highest unemployment rate was in August with 59.61% and the lowest unemployment rate in May with 47.19%.

**Group-wise operations**

To get the highest unemployment rates by month, we use a combination of group\_by() (to group our results by month) and the summarise() function.

data %>%

group\_by(month) %>%

summarise(max\_unemployment = max(unemployment\_rate))

## # A tibble: 10 x 2

## month max\_unemployment

##

## 1 Apr 48.6

## 2 Aug 59.6

## 3 Feb 58.5

## 4 Jan 48.9

## 5 Jul 58.3

## 6 Jun NA

## 7 Mar 55.9

## 8 May 47.2

## 9 Oct NA

## 10 Sep NA

How would you do it in base R?

aggregate(data$unemployment\_rate, by = list(data$month), max)

In some cases, you might no longer need your variables grouped after performing transformations. In this case, you can pipe another command with %>% ungroup() at the end to ungroup your data.

**Extracting unique observations**

Which unique months are included in the dataset? To get this information, we use the distinct() command in dplyr. As we see, all months from January to December are present.  
R allows us to also sort this data alphabetically with arrange().

# Get the distinct months in our dataset

data %>%

distinct(month)

## month

## 1 Jan

## 2 Feb

## 3 Mar

## 4 Apr

## 5 May

## 6 Jun

## 7 Jul

## 8 Aug

## 9 Sep

## 10 Oct

# Sort data alphabetically

data %>%

distinct(month) %>%

arrange(month)

## month

## 1 Apr

## 2 Aug

## 3 Feb

## 4 Jan

## 5 Jul

## 6 Jun

## 7 Mar

## 8 May

## 9 Oct

## 10 Sep

How would you do it in base R?

# Get the distinct months in our dataset

unique(data$month)

# Sort data alphabetically

sort(unique(data$month))

**Renaming variables**

The variable name gdppercapita is hard to read. We therefore want rename it to gdp\_per\_capita using the rename() function from the dplyr package. The compound assignment pipe operator %<>% from the magrittr package simultaneously serves as the first pipe in a chain of commands and assigns the transformed data to the left-hand side object.

# Rename the variable "gdppercapita" to "gdp\_per\_capita"

data %<>%

rename(gdp\_per\_capita = gdppercapita)

How would you do it in base R?

names(data)[names(data) == "gdppercapita"] <- "gdp\_per\_capita"

**Creating new variables**

The mutate() command allows you to generate a new variable. Let’s say you want to generate a dummy that indicates if it is summer or not. We call this variable summer.

# Create a new variable called "summer"

data %>%

mutate(summer = ifelse(month %in% c("Jun", "Jul", "Aug"), 1, 0))

## month gdp\_per\_capita unemployment\_rate life\_expectancy summer

## 1 Jan 31819.79 48.90126 70.19808 0

## 2 Feb 45485.76 58.51337 76.16142 0

## 3 Mar 18327.78 55.89483 67.68735 0

## 4 Apr 22275.86 48.61502 60.98838 0

## 5 May 23069.79 47.18671 66.39749 0

## 6 Jun 34896.12 NA 72.52299 1

## 7 Jul 14370.76 58.33964 66.42222 1

## 8 Aug NA 59.60927 74.91735 1

## 9 Sep NA NA 73.82523 0

## 10 Oct 24195.20 NA 65.26416 0

How would you do it in base R?

data$summer <- ifelse(data$month %in% c("Jun", "Jul", "Aug"), 1, 0)

**Additional features**

mutate() and summarise() also have several scoped variants that allow us to apply our commands simultaneously to several variables:

* \_all() affects all variables
* \_at() affects specific selected variables
* \_if() affects conditionally selected variables

How does this work in practice? Let’s say we have good reason to believe that all NAs in our observations are 0 and we want to transform all variables with NAs at once. We then use mutate\_if():

data %<>% mutate\_if(is.numeric, replace\_na, 0)

How would you do it in base R?

data <- lapply(data, function(x) replace(x, [is.na](http://is.na)(x), 0))

**Data wrangling for complex data structures: A walkthrough**

Following this short intro to the tidyverse, we want to showcase how sequences of data-transforming operations can be used to manage and combine complex data structures. We use a classical multi-level setup: We want to enrich multinational survey data with country-specific contextual information. For this purpose, we use data from Round 9 of the [European Social Survey](https://www.europeansocialsurvey.org/data/) (ESS), a cross-national survey of nearly 20 European countries fielded between August 2018 and May 2019 and augment it with party positions from the [Manifesto Project](https://manifesto-project.wzb.eu/datasets) and unemployment rates from the [World Bank](https://data.worldbank.org/indicator/SL.UEM.TOTL.ZS).

*Note:* As shown in the examples above, a sequence of commands may be combined using a series of piped operations. For didactic purposes, we break these sequences into shorter segments in the following examples.

**Micro-level survey data from the European Social Survey**

We start with some initial data management for the ESS data. The code below selects a range of relevant variables. First, we select some administrative variables, including country identifiers (cntry), information on the start time of the interview (inwyys, inwmms, and inwdds), and the design weights (dweight). Secondly, we select variables on individuals’ voting behavior and party affinity. Next to two variables that establish whether respondents voted in the last general election (vote) or feel close to any party (clsprty), respectively, this includes a range of country-specific vote recall and party ID variables (starting with prtv and prtcl, respectively). For respondents who voted (feel close to any party), these variables specify which national party they voted for (feel closest to).

Our main predictor is unemployment. We therefore select a variable that records whether respondents have ever been unemployed for longer than three months (uemp3m) and whether any such period was during the past five years (uemp5yr). Combining the information from both variables allows us to distinguish individuals with experiences of long-term unemployment during the last five years from those without such experiences. Additionally, we keep information on respondents’ gender, age, education, and ethnic minority status.

Code: European Social Survey – Initial data management

ess <- read.dta13("efficient-data-r/raw-data/202090127\_ESS9e01\_1.dta",

fromEncoding = "UTF-8") %>%

select( ### select relevant variables

cntry, ### ISO2C country codes

inwyys, ### start of interview, year

inwmms, ### start of interview, month

inwdds, ### start of interview, day

dweight, ### design weights

vote, ### voted in last general election?

starts\_with("prtv"), ### country-specific vote recall

clsprty, ### close to any party?

starts\_with("prtcl"), ### country-specific party ID

uemp3m,

uemp5yr, ### unemployed in the past 5 years

gndr, ### gender

agea, ### age

eisced, ### education (ISCED)

blgetmg ### ethnic minority

) %>%

mutate\_if(is.factor, as.character) %>% ### factor -> character

mutate(uemp5yr = ifelse(uemp3m == "No", ### recode uemp5yr

"No", uemp5yr)) %>%

mutate\_if(is.character, ### recode responses to missing values

function (x)

ifelse(

x %in% c("Refusal", "Don't know",

"No answer", "Not applicable"),

NA,

x

))

**Election Dates**

Now that we have selected and partly recoded our micro-level variables, we are confronted with a new challenge: The ESS records respondents’ recall of their voting behavior in the most recent national election. We thus need to identify the correct electoral context for each country before we can add information on party strategies. This requires that we identify the date of the most recent national election prior to a given interview.

As a first step, we need to collect information on the country-specific ESS field times. We use information on the date on which individual interviews were started, which are stored in separate variables for year, month, and day: inwyys, inwmms, and inwdds. We combine these three variables in YYYYMMDD format using the base R sprintf() command and save them as a date using the ymd() command from the tidyverse’s lubridate package. After this, we can drop the three variables from our data using dplyr’s select() command.

ess %<>%

mutate(inwdate = sprintf('%04d%02d%02d', inwyys, inwmms, inwdds)) %>%

mutate(inwdate = ymd(inwdate)) %>%

select(-inwyys,-inwmms,-inwdds)

To retrieve information on country-specific field periods, we need to find the earliest and latest interview dates within each country. We therefore use the dplyr commands group\_by() and summarize() to generate an auxiliary data frame named ess\_dates.

ess\_dates <- ess %>%

group\_by(cntry) %>%

summarize(field\_start = min(inwdate),

field\_end = max(inwdate)) %>%

print()

## # A tibble: 19 x 3

## cntry field\_start field\_end

##

## 1 AT 2018-09-18 2019-01-12

## 2 BE 2018-09-20 2019-01-27

## 3 BG 2018-11-16 2018-12-15

## 4 CH 2018-08-31 2019-02-11

## 5 CY 2018-09-18 2019-05-26

## 6 CZ 2018-11-17 2019-02-02

## 7 DE 2018-09-03 2019-03-03

## 8 EE 2018-10-01 2019-03-02

## 9 FI 2018-09-06 2019-02-18

## 10 FR 2018-10-20 2019-04-01

## 11 GB 2018-09-01 2019-02-16

## 12 HU 2019-01-31 2019-05-22

## 13 IE 2018-11-07 2019-03-31

## 14 IT 2018-12-17 2019-03-10

## 15 NL 2018-09-03 2019-01-20

## 16 NO 2018-10-04 2019-05-16

## 17 PL 2018-10-26 2019-03-20

## 18 RS 2018-09-28 2019-02-24

## 19 SI 2018-09-24 2019-02-01

Now that we know the country-specific field periods, we need to find the election date preceding the respective field periods. For this purpose, we use the comprehensive collection on parliaments and elections from ParlGov. The data carries information on parties performance in both national and European parliamentary elections. As the data set comes in .csv format, we use the read.csv() command to load it into R. In order to use the data, we need to do a bit of housekeeping first:

1. We use the countrycode package to convert country names to ISO2C codes. This allows us to combine them with the data from ess\_dates later on.
2. We retain information on national parliamentary elections, thus dropping observations for elections to the European parliament.
3. We select three variables we need: Country codes in ISO2C format, election dates, and unique election IDs.
4. As the data set still contains many observations per election (one for each party), we use group\_by()–summarize\_all()–ungroup() to collapse the data frame. As a result, it now contains one unique value for cntry and election\_date for each value of election\_id.
5. Lastly, we combine the election dates data from pgov with the field times from ess\_dates, using the harmonized cntry identifiers. Using right\_join() ensure that we only keep the election dates for the 19 countries included in the right-hand side data in ess\_times.

pgov <- read.csv("efficient-data-r/raw-data/20200127\_parlgov.csv") %>%

mutate(cntry = countrycode(country\_name, "[country.name](http://country.name)", "iso2c")) %>%

filter(election\_type == "parliament") %>%

select(cntry, election\_date, election\_id) %>%

group\_by(election\_id) %>%

summarize\_all(unique) %>%

ungroup() %>%

right\_join(ess\_dates, by = "cntry")

At this point, we have added *all* available election dates for each available country. What we want, however, is to find only the date of the most recent election preceding a country’s ESS field period. So what do we do?

1. We save election\_date as an actual date variable. This allows us to compare it to other date variables.
2. For each country, we exclude all elections that took place after the respective start date of the ESS field period.[2](https://www.mzes.uni-mannheim.de/socialsciencedatalab/article/efficient-data-r/#fn2)
3. We group\_by() countries and arrange() the observations by election\_date from earliest to latest. We then identify the most\_recent election that has the smallest duration between the start of the field period and the preceding election date (while we’re at it, we also store the date of previous election for each country, which will come in handy later on). We thenungroup().
4. We can now use filter() to keep only those observation where election\_date == most\_recent evaluates to true.

pgov %<>%

mutate(election\_date = ymd(election\_date)) %>%

filter(election\_date <= field\_start) %>%

group\_by(cntry) %>%

arrange(election\_date) %>%

mutate(most\_recent = election\_date[which.min(field\_start - election\_date)],

prev\_election\_date = lag(election\_date)) %>%

ungroup() %>%

filter(election\_date == most\_recent)

The rest is just a little housekeeping: We extract the years of the most recent and previous election dates and select only those variables which we are going to need later on.

Code: Election dates (housekeeping)

pgov %<>%

mutate(

election\_year = year(election\_date),

prev\_election\_year = year(prev\_election\_date)

) %>%

select(cntry,

election\_date,

prev\_election\_date,

election\_year,

prev\_election\_year)

**Party Positions**

We now know the exact dates of the elections in which individuals cast their votes as reported in the ESS. This finally allows us to combine the survey data with information on the policy positions on which center-left parties campaigned in the respective elections. We therefore load the data set from the [Manifesto Project](https://manifesto-project.wzb.eu/datasets), which comes in Stata 13+ format and therefore requires the read.dta13() command from the readStata13 package. We start with a little housekeeping, converting country names to ISO2C codes, subsetting the data to include only social democratic parties, and storing election dates as actual date variables.

Code: Manifesto Project – Initial data management

cmp <- read.dta13("efficient-data-r/raw-data/20200127\_cmp2019b.dta") %>%

mutate(cntry = countrycode(countryname, "[country.name](http://country.name)", "iso2c")) %>%

filter(parfam == "soc social democratic") %>%

mutate(election\_date = as.Date(edate))

We then right\_join() the data frame pgov to the Manifesto data in cmp, matching observations by both cntry and election\_date, and select all relevant variables. Next to pervote, which contain parties’ vote shares in the most recent election, we keep rile and welfare. The former is parties’ general left-right position, the latter their position on welfare issues.

cmp %<>%

right\_join(pgov, by = c("cntry", "election\_date")) %>%

select(

cntry,

election\_date,

election\_year,

prev\_election\_date,

prev\_election\_year,

party,

partyname,

partyabbrev,

pervote,

rile,

welfare

)

One problem we still need to address is that some countries have multiple parties of the social democratic party family. For the sake of simplicity, we choose to focus only on one party per country, namely the strongest. Therefore, we again group by country and keep only those observations with the highest vote percentage. We now have a data set that contains the positions of each countries’ main center-left party, based on the most recent election preceding the ESS, which we can easily merge with our ESS survey data using the cntry identifier.

cmp %<>%

group\_by(cntry) %>%

filter(pervote == max(pervote, na.rm = TRUE)) %>%

ungroup()

## View data

cmp %>%

select(cntry, election\_year, partyname, partyabbrev, welfare)

## Add to ESS

ess %<>% left\_join(cmp, by = "cntry")

**Unemployment Rates**

Now, suppose we also want to retrieve information on the average national unemployment rate during the legislative period preceding the most recent national election. For this purpose, we use data on annual national unemployment rates from the [World Bank](https://data.worldbank.org/indicator/SL.UEM.TOTL.ZS).

Code: World Bank unemployment rates – Initial data management

unemp <-

read.csv("efficient-data-r/raw-data/20200114\_worldbank\_unemprate.csv",

skip = 4L) %>%

mutate(cntry = countrycode(Country.Name, "[country.name](http://country.name)", "iso2c")) %>%

filter(cntry %in% cmp$cntry)

Following some initial data management, we can see that the data is in wide format: Every country has only one row while their annual unemployment rates are stored in multiple columns named X1960, …, X2019. So what do we do?

First, we select our country-identifier along with all yearly observations starting with X (except an unspecified additional variable named X). We then use the melt() command from the reshape2 package. This changes the data format from wide to long: We now have 60 observations per country (for years 1960-2019), uniquely identified by a new variable with we rename to year, with their corresponding values stored in the new variable we rename to unemprate. The year variable contains the previous variable names (X1960, …, X2019). After omitting the leading X character, for which we use the base R substr() command, we store the years as numeric values.

unemp %<>%

select(cntry, starts\_with("X"),-X) %>%

melt(id.vars = "cntry") %>%

rename(year = variable,

unemprate = value) %>%

mutate(year = as.numeric(substr(year, 2L, 5L)))

Using the long-format data, we then proceed as follows:

1. We select the country-specific information in the most recent and previous election years from the pgov data frame, and left\_join() it with the unemp data frame using our country identifiers. Thereby, we supplement each of the 60 yearly observations per country with the information on the relevant time span.
2. After grouping the data set by cntry, we can use this information to filter() only the years of each country’s past legislative period.
3. The summarize()function, lastly, allows us to average across these observation to retrieve a single observation per country. This gives us the desired average unemployment rate during the past legislative period, which we may merge with the survey data using the cntry identifier.

unemp %<>%

left\_join(pgov %>% select(cntry, election\_year, prev\_election\_year),

by = c("cntry")) %>%

group\_by(cntry) %>%

filter(year >= prev\_election\_year &

year <= election\_year) %>%

summarize(unemprate = mean(unemprate))

ess %<>% left\_join(unemp, by = "cntry")

**Programming and Debugging**

**Identifying center-left parties in the ESS**

Now we need a variable which indicates whether a respondent of the ESS voted for a center-left party. Unfortunately, the variables indicating the party a respondent voted for at the last election are rather messy: Each country has its own variable and party names are not consistent: sometimes.

To save the information, we generate a new auxiliary data frame which we use later on.

Code: Preparation of auxiliary data

## Main center-left parties

aux <- list(

AT = "SPÖ",

BE = "SP.A",

BG = "Balgarska sotsialisticheska partiya (BSP)" ,

CH = "Social Democratic Party / Socialist Party" ,

CY = "Progressive Party of Working People (AKEL)" ,

CZ = "ČSSD",

DE = "SPD" ,

EE = "Sotsiaaldemokraatlik Erakond",

FI = "Social Democratic Party",

FR = "PS (Parti Socialiste)",

GB = "Labour",

HU = "MSZP (Magyar Szocialista Párt)",

IE = "Labour",

IT = "Partido Democratico (PD)",

NL = "Socialist Party",

NO = "Arbeiderpartiet",

PL = "Zjednoczona Lewica",

RS = "Boris Tadic, Cedomir Jovanovic - Savez za bolju Srbiju - LDP, LSV, SDS",

SI = "L - Levica"

) %>%

melt() %>%

rename(

ess\_party\_name = value,

cntry = L1

)

## Match vote variables and country codes

vote\_var <- ess %>%

select(starts\_with("prtv"),-prtvede1) %>%

names()

vote\_var\_order <-

sapply(aux$cntry, function (x)

which(grepl(x, toupper(vote\_var))))

## Add to auxiliary data frame

aux %<>%

mutate(vote\_var = vote\_var[vote\_var\_order]) %>%

mutate\_if(is.factor, as.character)

**Programming a function**

Now, we want to add a variable that indicates whether a respondent voted for a center left party (as opposed to voting for a different party or not voting at all). To this end, we program a function named ess\_center\_left(). This function takes three input arguments. micro\_data, aux\_data, and group\_var. micro\_data is a data frame containing the survey data, i.e., our ess object. aux\_datais a data frame with auxiliary information, i.e., our aux object. group\_var, lastly, is a character string containing the name of the grouping variable (cntry) in both data frames. After several operations which we discuss in detail below, the function returns the modified data frame micro\_data, now with an added variable v\_center\_left.

ess\_center\_left <- function(micro\_data, aux\_data, group\_var) {

## A priori consistency checks

...

## Main function

...

## A posteriori consistency checks

...

## Return output

return(micro\_data)

}

So what goes into our function body? We divide our function into three parts:

1. A priori consistency checks
2. The main function
3. A posteriori consistency check

In the following, we will walk through each part separately and subsequently combine them into one function.

**A priori consistency checks**

A priori consistency checks test whether we supply the correct inputs to our function. In a *first step*, we check if both micro\_data and aux\_data actually contain a variable of the name specified in group\_var. If either of them does not, the function stops and returns an error message. In a *second step*, also check for the unique values (i.e., country codes) of group\_var in micro\_data and aux\_data and establish their intersection in group\_vals\_both. The *third step* establishes which, if any, values of group\_var are missing in the micro and auxiliary data. In the *fourth and final step*, we print corresponding warning messages that inform us which countries are not included in both the micro and auxiliary data. For these, the new variable v\_center\_left will not be created.

# 1) Check if group\_var is a variable in micro\_data and aux\_data

if (not(group\_var %in% names(micro\_data))) {

stop("group\_var is not a variable in micro\_data.")

} else if (not(group\_var %in% names(aux\_data))) {

stop("group\_var is not a variable in aux\_data.")

} else {

# 2) Unique values of group\_var

group\_vals\_micro <- unique(micro\_data[, group\_var])

group\_vals\_aux <- unique(aux\_data[, group\_var])

group\_vals\_both <- group\_vals\_micro[group\_vals\_micro %in% group\_vals\_aux]

# 3) Missing values of group\_var in micro\_data and aux\_data

group\_vals\_only\_micro <-

group\_vals\_micro[not(group\_vals\_micro %in% group\_vals\_both)]

group\_vals\_only\_aux <-

group\_vals\_aux[not(group\_vals\_aux %in% group\_vals\_both)]

# 4) Missing group\_var values in micro\_data or aux\_data

if (length(group\_vals\_only\_micro) > 0) {

warning(paste("Group values only in micro\_data:",

group\_vals\_only\_micro, sep = " "))

}

if (length(group\_vals\_only\_aux) > 0) {

warning(paste("Group values only in aux\_data:",

group\_vals\_only\_aux, sep = " "))

}

}

**Main function**

If our data inputs pass the initial consistency checks, the function generates the new variable v\_center\_left within each available country. We begin by defining an auxiliary identifier, aux\_id that allows us to uniquely identify observations in the ESS data. This will allow us to merge the newly created variable to the original data later on. We then generate a list, vote\_recoded that serves a container for the country-specific data that we generate in the subsequent loop through the different countries.

In this loop, we first retrieve the name of the ESS vote choice variable and the name of the center-left party in a given country. We then subset the data to observations from this country and retain three variables only: aux\_id, vote (i.e., whether someone voted or not), and the country-specific vote choice variable, which we subsequently rename to vote\_choice and store in our system’s native encoding.

Following this, we assign one out of four possible values to the new variable v\_center\_left:

1. If we have no information on respondents’ voting behavior: NA
2. If the respondent did not vote or was not eligible to vote: Did not vote
3. If the respondent voted for the country’s main center-left party: Yes
4. If the respondent voted for a different party: No

After this, we collapse the list of country-specific data frames to a single data frame which we then add to the original data frame micro\_data by our identifier aux\_id.

# Auxiliary ID

micro\_data$aux\_id <- seq\_len(nrow(micro\_data))

# List container for data frames containing v\_center\_left

vote\_recoded <- list()

# Loop through groups

for (j in group\_vals\_both) {

# Name of the group's vote choice variable

vote\_var\_j <- aux\_data$vote\_var[aux\_data$cntry == j]

# Name of the group's center left party (in native encoding)

ess\_party\_name\_j <- aux\_data$ess\_party\_name[aux\_data$cntry == j]

# Generate v\_center\_left for this group

vote\_recoded[[j]] <- micro\_data %>%

filter(cntry == j) %>% # subset to group

select(aux\_id, vote, vote\_var\_j) %>% # select variables

rename(vote\_choice = !![as.name](http://as.name)(vote\_var\_j)) %>% # rename vote choice

mutate(vote\_choice = enc2native(vote\_choice), # harmonize encoding

v\_center\_left = # create v\_center\_left

ifelse(

[is.na](http://is.na)(vote) & [is.na](http://is.na)(vote\_choice),

NA, # missing information

ifelse(

vote %in% c("No", "Not eligible to vote"),

"Did not vote", # non-voters

ifelse(vote\_choice == ess\_party\_name\_j,

"Yes", # center-left voters

"No") # voted for other party

)

))

}

# Collapse list of data frames to a single data frame

vote\_recoded %<>%

bind\_rows() %>%

select(aux\_id, v\_center\_left)

# Remove old versions of v\_center\_left if present

if ("v\_center\_left" %in% names(micro\_data)) {

micro\_data %<>% select(-v\_center\_left)

}

# Add new variable to micro\_data by aux\_id

micro\_data %<>%

full\_join(vote\_recoded, by = "aux\_id") %>% # add new variable

select(-aux\_id) # drop auxiliary ID

**A posteriori consistency checks**

After generating our new variable v\_center\_left within each country, we want to know if our function actually worked. For this purpose, we check whether any country has zero percent of center-left voters. Given that we supplied a name for one center-left party per country, this would likely suggest a mismatch in our inputs.

# Proportions of center left voters within each group

sample\_prop <- micro\_data %>%

select(group\_var, v\_center\_left) %>%

group\_by(!![as.name](http://as.name)(group\_var)) %>%

summarize(prop\_v\_center\_left = mean(v\_center\_left == "Yes", na.rm = TRUE))

# Check if any group has 0% center left voters

if (any(sample\_prop$prop\_v\_center\_left == 0)) {

warning(paste(

"No center-left voters in",

paste(sample\_prop$cntry[sample\_prop$prop\_v\_center\_left == 0],

collapse = " "),

"- check your inputs!",

sep = " "

))

}

Now, we can put the parts together and define our function.

Code: Full function

ess\_center\_left <- function(micro\_data, aux\_data, group\_var) {

## A priori consistency checks

# 1) Check if group\_var is a variable in micro\_data and aux\_data

if (not(group\_var %in% names(micro\_data))) {

stop("group\_var is not a variable in micro\_data.")

} else if (not(group\_var %in% names(aux\_data))) {

stop("group\_var is not a variable in aux\_data.")

} else {

# 2) Unique values of group\_var

group\_vals\_micro <- unique(micro\_data[, group\_var])

group\_vals\_aux <- unique(aux\_data[, group\_var])

group\_vals\_both <- group\_vals\_micro[group\_vals\_micro %in% group\_vals\_aux]

# 3) Missing values of group\_var in micro\_data and aux\_data

group\_vals\_only\_micro <-

group\_vals\_micro[not(group\_vals\_micro %in% group\_vals\_both)]

group\_vals\_only\_aux <-

group\_vals\_aux[not(group\_vals\_aux %in% group\_vals\_both)]

# 4) Missing group\_var values in micro\_data or aux\_data

if (length(group\_vals\_only\_micro) > 0) {

warning(paste("Group values only in micro\_data:",

group\_vals\_only\_micro, sep = " "))

}

if (length(group\_vals\_only\_aux) > 0) {

warning(paste("Group values only in aux\_data:",

group\_vals\_only\_aux, sep = " "))

}

}

## Main function

# Auxiliary ID

micro\_data$aux\_id <- seq\_len(nrow(micro\_data))

# List container for data frames containing v\_center\_left

vote\_recoded <- list()

# Loop through groups

for (j in group\_vals\_both) {

# Name of the group's vote choice variable

vote\_var\_j <- aux\_data$vote\_var[aux\_data$cntry == j]

# Name of the group's center left party (in native encoding)

ess\_party\_name\_j <- aux\_data$ess\_party\_name[aux\_data$cntry == j]

# Generate v\_center\_left for this group

vote\_recoded[[j]] <- micro\_data %>%

filter(cntry == j) %>% # subset to group

select(aux\_id, vote, vote\_var\_j) %>% # select variables

rename(vote\_choice = !![as.name](http://as.name)(vote\_var\_j)) %>% # rename vote choice

mutate(vote\_choice = enc2native(vote\_choice), # harmonize encoding

v\_center\_left = # create v\_center\_left

ifelse(

[is.na](http://is.na)(vote) & [is.na](http://is.na)(vote\_choice),

NA, # missing information

ifelse(

vote %in% c("No", "Not eligible to vote"),

"Did not vote", # non-voters

ifelse(vote\_choice == ess\_party\_name\_j,

"Yes", # center-left voters

"No") # voted for other party

)

))

}

# Collapse list of data frames to a single data frame

vote\_recoded %<>%

bind\_rows() %>%

select(aux\_id, v\_center\_left)

# Remove old versions of v\_center\_left if present

if ("v\_center\_left" %in% names(micro\_data)) {

micro\_data %<>% select(-v\_center\_left)

}

# Add new variable to micro\_data by aux\_id

micro\_data %<>%

full\_join(vote\_recoded, by = "aux\_id") %>% # add new variable

select(-aux\_id) # drop auxiliary ID

## A posteriori consistency checks

# Proportions of center left voters within each group

sample\_prop <- micro\_data %>%

select(group\_var, v\_center\_left) %>%

group\_by(!![as.name](http://as.name)(group\_var)) %>%

summarize(prop\_v\_center\_left = mean(v\_center\_left == "Yes", na.rm = TRUE))

# Check if any group has 0% center left voters

if (any(sample\_prop$prop\_v\_center\_left == 0)) {

warning(paste(

"No center-left voters in",

paste(sample\_prop$cntry[sample\_prop$prop\_v\_center\_left == 0],

collapse = " "),

"- check your inputs!",

sep = " "

))

}

## Return output

return(micro\_data)

}

**Debugging**

Now we apply this function:

ess %<>%

ess\_center\_left(aux\_data = aux, group\_var = "cntry")

## Warning in ess\_center\_left(., aux\_data = aux, group\_var = "cntry"): No

## center-left voters in DE - check your inputs!

There seems to be something wrong with Germany. Let’s have a look:

ess %>%

filter(cntry == "DE") %>%

select(prtvede2) %>%

table()

## .

## Alliance 90/The Greens (Bündnis 90/Die Grünen)

## 289

## Alternative for Germany (AFD)

## 111

## Christian Democratic Union/Christian Social Union (CDU/CSU)

## 558

## Free Democratic Party (FDP)

## 147

## National Democratic Party (NPD)

## 2

## Other

## 36

## Pirate Party (Piratenpartei)

## 4

## Social Democratic Party (SPD)

## 355

## The Left (Die Linke)

## 125

aux %>%

filter(cntry == "DE")

## ess\_party\_name cntry vote\_var

## 1 SPD DE prtvede2

As we can see, we do have a sizable number of center-left voters in Germany. However, the correct code for the SPD is not “SPD” but “Social Democratic Party (SPD)”. So let’s fix this bug and apply the function again:

aux$ess\_party\_name[aux$cntry == "DE"] <- "Social Democratic Party (SPD)"

# Apply the function

ess %<>%

ess\_center\_left(aux\_data = aux, group\_var = "cntry")

Lastly, we save the data in the gen-data folder:

save(ess, file = "efficient-data-r/gen-data/ess-proc.RData")

**Working with the data**

We can now use the data to approach our initial question through some simple descriptives. We first generate a binary version of our outcome variable v\_center\_left, where Yes equals TRUE whereas both No and Did not vote equal FALSE. We then split the data into a list of 19 country-specific data frames. Using sapply(), we then apply the a function for estimation and prediction to each of 19 data sets. Specifically, we run binary logistic regressions of v\_center\_left on uemp5yr. We then use the predict() function to retrieve the predicted probabilities of center-left votes among the unemployed along with their 95% confidence intervals for each country.

ess\_est <- ess %>%

mutate(v\_center\_left = (v\_center\_left == "Yes")) %>%

group\_split(cntry) %>%

lapply(function(dat) {

mod <- glm(v\_center\_left ~ uemp5yr,

data = dat,

weights = dweight)

pred <- predict(mod,

newdata = data.frame(uemp5yr = "Yes"),

se.fit = TRUE)

pos\_welfare <- unique(dat$welfare)

output <- data.frame(

est = pred$fit,

lower95 = pred$fit + qnorm(.025) \* pred$se,

upper95 = pred$fit + qnorm(.975) \* pred$se,

welfare = unique(dat$welfare)

)

return(output)

}) %>%

bind\_rows()

We can then plot the relationship between center-left support among the unemployed and center-left parties’ welfare policy positions. As we can see below, based on 14 out of the 19 countries for which data on party positions is available, there does not seem to be a straightforward relationship between the two variables.

ess\_est %>%

ggplot(aes(x = welfare, y = est)) +

geom\_errorbar(aes(ymin = lower95, ymax = upper95)) +

geom\_point() +

labs(x = "Center Left Welfare Policy Position",

y = "Proportion of Unemployed Voting for Center Left")

