R-Introduction - Part 1

A Workshop for Modul University Vienna Faculty

Gunther Maier

Table of contents

1 Introduction

This is part 1 of a very brief introduction to R. I compiled this information for a presentation to faculty at Modul University Vienna.

1.1 Why use R?

There are a number of reasons why you should use R

- It is Open Source software and can be used for free
- It is used heavily in academia and also in some data-intensive industries
- It has strong support from experts who constantly implement new functionality
- There are thousands of packages available that extend the functions of R
- It is very flexible and has a very wide range of applications
- It is the computing workhorse for a wider range of applications; in particular in publishing
- It is part of a wider network of interacting open source tools (Pandoc, LaTeX, etc.)

1.2 Why not use R?

There are also a number of reasons why you may not want to use R

- For a beginner, it is more difficult to use than most other statistics packages
- The syntax of commands and parameters is not consistent
- If you need quantitative data analysis only occasionally, it may not be worth the effort

1.3 The ways to use R

There are different ways to use R. They range from very direct, where you type R commands and send them to R for execution, to very indirect, where you cannot even see the R code.

1.3.1 Via the R console

The most direct form is the R console which you reach either through the RGui that you installed with R, or through the Console window of RStudio. In this mode you enter and execute one command after the other.

1.3.2 Via an R script

You can collect all the commands that you want to run in a text file, an R script, and then execute the sequence of commands in one run. This is particularly useful for repeated tasks or when it is important to document the process.

1.3.3 Via RMarkdown or Quarto

R scripts focus on the R commands. You may structure them with embedded comments so that after some time you can retrace your analysis. Depending how much you comment, you may have describing text embedded into your code.

RMarkdown and Quarto are tools that go the other way. They put the text in the foreground and allow you to embed R code into the text. Both use markdown for structuring and formatting the text (this document, for example, was written in Quarto). The embedded R code is executed and typically the respective output is inserted underneath the respective chunk of code.

With these tools, you can produce a wide range of output; from articles and books to presentations, web pages, blogs, and probably others.

1.3.4 Via Shiny

When you create a Shiny application, the R code stays in the background. A Shiny application is typically run via some parameters on a web-page that the user sets or changes interactively. These new parameters are sent to your R code on a server and the result of the computations are transferred back to the browser and displayed. I just mention this as one of the ways to run R code. In the rest of the text, I will not talk about Shiny. This topic is beyond the scope of this presentation / document.

2 R and RStudio

What is R and what is RStudio? I do not try to define what R is. It is many different things like

- a programming language,
- a statistics package,
- a tool for data management,
- and many more.

The relation between R and RStudio is much clearer. Using the analogy of an automobile, R is the engine and RStudio is the cockpit. R does all the work. RStudio provides us with a more user friendly interface. In principle, we can do everything we do in RStudio also in the R console alone or via an R-script that we call from the R console. RStudio just makes our tasks a little easier. Because of that, we will exclusively use RStudio and always interact through it with R.

2.1 The many Windows of RStudio

RStudio has four windows. Two on the left,

- "Source" (on top) and
- "Console" (at the bottom),

and two on the right

- "Environment" (on top) and
- "Files" (at the bottom).

Each one of them has tabs to sub-windows, buttons and tool bars. You can change the size of those windows by moving the border between them. Move the cursor to the vertical space between the windows on the left and those on the right. The cursor will change to four arrows. Click and hold the left mouse button and move the mouse to change the width of the left and right windows. You can do the same with the horizontal space between the windows. There, you can change the height of the windows (either on the left or on the right).

Within the two columns, you can switch the size of each window between "minimum", "shared" and "maximum" through the buttons on the top right of each window.

2.1.1 Console

The console window at the bottom left is practically the same as the console that opens when you start the RGui (R Graphical User Interface). In this window, you can enter R commands, click enter and see the result (if there is any that goes to the console). Also, when you execude some R commands from the source window, you will see the interaction in the console window.

One advantage of the console in RStudio over the console of the RGui is the autocomplete function and the immediate help that RStudio provides. Click into the console window of RStudio and type "plot" (do not type Enter). RStudio will display a list with all the commands that begin with "plot". To the right of the list, with yellow background, you will see part of the help page of the currently marked command. When you press the F1-button, the full help pages will pop up in the "Help" pane in the Files window (bottom right). In the console window, move the selection to the first option in the list (saying just "plot" - you may have to type "plot" anew in the console window to get the list again) and press the "Tab" button. RStudio will enter the brackets that the command requires and in a tooltip show what you need to enter between the brackets.

I use the console window only to try out R commands. When I write an R script or a text in RMarkdown or Quarto I sometimes want to try out an R command before I add it to my script or text. In that case, I type it into the console either to try it out or to get support from the console's support system.

All the commands that you enter in the console and execute, are automatically pushed into the history. The history is very useful when you misspelled an R command in the console window. With the up and down keys on the keyboard you can scroll through the history. Correct the spelling and press Enter to run the - now hopefully - correctly spelled command.

Click into the console window of RStudio, type

```
5 + 7
```

and then Enter. R will calculate this function and output the result in the console:

```
[1] 12
```

Now, press the up-key. You will see the function again in the console. Press the back-key to erase the 7, type 9 and enter again.

```
5 + 9
```

```
[1] 14
```

The new result will be displayed again in the console.

Now, make sure you see the "History" tab in the "Environment" window (top right) and click that tab. You will see the two commands ("5 + 7" and "5 + 9") in the history. We will talk more about this below.

2.1.2 Source

When you work more with R, the source window will be the window you will use most often. To organize, manage, and document the workflow of your analysis, it makes sense to collect the respective R commands in an R script, an RMarkdown or a Quarto document. Since I will present RMarkdown and Quarto at the end of the lecture, I will concentrate here only on the R script option.

To avoid confusion, let us first clear the console. Either press Ctrl-L or select "Clear Console" from "Edit" in the menu.

When you created your R project, RStudio opened an empty source window. Move the cursor to this window and click. Then, enter the following lines

```
# R as a calculator

5 + 7
5 + 9

# A first plot

plot(10:1)
```

When you have entered this, nothing will happen in the console. All you did so far is to collect these lines in your R script. You can execute the entire script or only parts of it. Place the cursor at the top of the script in your source window and press Ctrl-Enter. RStudio will jump to the first executable line ("**5 + 7**"), send that to the console (i.e., to R) where it will be executed. Instead of pressing Ctrl-Enter you may also click the Run button. When you select some or all lines in the script and press Ctrl-Enter, the selected lines will be executed.

If you want to execute all the lines in the script, you may want to click the Source button in the task bar of the source window.

The output of the plot command will not appear in the console, but in the Plots pane of the files window (bottom right). You may have to click the "Plots" tab to see the output.

Lines beginning with "#" are comments. As you may have realized by now, R skips such comments as well as any empty lines. The plot command gets as input a list of numbers running from 10 down to 1. This is what "10:1" generates. These values are shown on the vertical axis in the plot. The horizontal axis shows the sequence of the numbers. Change the list to "1:10" and to "10:20" and see how the plot changes.

2.1.3 Environment

The two windows in the right column each contains a number of panes. Also, the list of panes is not constant. For certain types of project RStudio inserts additional panes.

As we will see later, the Environment pane in the Environment window lists all the variables, data frames, functions, etc. that we created in the current session. Currently, the environment is empty. Move the cursor into the console window and click. Then enter

```
a <- 5 + 7
```

and hit Enter. This command creates a variable named "a" and assigns the result of "5 + 7" to it. The assignment operator "<-" consists of the two characters "<" and "-" with nothing in between. It means "Assign the result of the calculation to the right to the variable to the left. After you hit Enter, the variable"a" and its value showed up in the environment pane in the environment window. Note that in the console the result of the calculation, 12, is not displayed. Because of the assignment operator the result is sent to the variable "`a`" and not to the console.

When you type "`a`" and Enter into the Console, the result will be displayed:

```
a
```

```
[1] 12
```

When you want to assign the result of a calculation to a variable and *also* want to see the result of the calculation, you can enclose the statement in parentheses like in the following example. It creates a variable "`b`" *and* also shows the result.

```
(b <- 224 / 86)
```

```
[1] 2.604651
```

We already mentioned the History and the History pane. The executed commands are accumulated here. If you want to execute a command way back in the history, you do not have to scroll with the up-key until you get there. Just scroll through the history pane, click into the respective line and then click the button "To Console". This will send this command (or a block of marked commands) to the console. In the same way, you can send commands from the history to the source window. The "Disk"-button lets you save the whole history into a text file. With the broom button, you can clean out the history. When you do this, the history is deleted and you cannot scroll through it any more.

2.1.4 Files

Also the Files window has a number of panes. The Files pane shows the files in the current working directory. Through the buttons you can do some standard file management operations.

We have already seen the Plots pane above. With the Zoom button you can display a larger version of the plot in a popup window. Via Export, you can copy the save the plot in various formats or copy it to the clipboard.

The Packages pane lists all the packages that you have installed. The Update button allows you to list all the packages that need an update and update them. The checkbox in the first column shows whether the respective package is loaded or not. You can load and unload a package by setting and removing this checkmark.

The Help pane links to the R help function. When you request help for some topic, the result will be displayed in this pane.

3 R Packages

A lot of the functionality of R is actually in packages. Some packages are so central that they are installed and loaded by default. Others need to be installed and loaded. When you install a package, the respective files are downloaded from a repository and stored on your computer. After you have installed a package, the maintainer of the package may update the code and provide a new version in the repository. In that case, your package may need updating. You may do this via the Packages pane in the Files window or via the menu items Tools - Check for Package Updates.

3.1 The CRAN repository

The most important source of R packages is CRAN (Comprehensive R Archive Network, https://cran.r-project.org/). As a user, you can get tested and verified versions of R packages from this repository. CRAN is the default source for R packages and whenever you call `install.packages("package-name")` R will search for the specified package on CRAN and install it when found.

As the time of this writing, there are over 20,000 packages available on CRAN. The CRAN webpage provides lists by date of submission and by name.

3.2 Installing a package

Let us install the package "readxl", which allows us to read Excel files. In RStudio, we can install this package in one of three ways

1. Go to the Packages pane in the Files window and click the Install button.
2. Select Tools - Install Packages from the menu.
3. Type "install.packages()" into the Console

In all three options you then enter "readxl" and type Enter. In the first two options, you omit the quotation marks, in the third option, the quotation marks are required.

```
install.packages("readxl")
```

You should install a package only once. It does not do any harm to reinstall it, but, we try to avoid the extra traffic on the CRAN server and on the Internet.

3.3 Loading a package

While you install packages only once, you need to load a package every time you want to use it. To load the package "readxl", enter

```r
library(readxl)
```

4 Finding help

There is a lot of help available for R. A Google search usually yields numerous links to webpages, blog entries, discussions, and videos. This is useful when you are looking for a concept or method, but do not know how that is implemented in R.

When you need help on a function or package you know the name of, it makes more sense to use the help feature of R. R also contains its own extensive support system. It is described in https://www.r-project.org/help.html.

You can request help either with the `help()` function or with the `?` operator. For example, to get information about the plot function, you can issue

```
help(plot)
```

or

```
?plot
```

The `?` operator is a somewhat limited version of the `help()`function. It can only be used for available functions where you know the name.

In both cases, the help page of the plot function will be shown in the Help pane of the Files window. It contains a description, a list and description of the parameters, and examples.

If you need help about a package, use the command

```
help(package = "package-name")
```

For help about a function defined in a package or a dataset provided by a package that you have not loaded, use the `help()`function with the package operator. For example, to get information about the dataset "mpg" that is provided by the package "ggplot2", use the command

```
help(mpg, package = "ggplot2")
```

5 Loading data

5.1 Using data available in R and R packages

R comes with a set of data-sets that are handy for testing and demonstration purposes. The data() command without parameters generates a list of all the datasets (the output goes to a separate window and therefore does not show in Quarto).

```
data()
```

If you want to see **all** datasets available in **all** installed packages, use the following command. It is mentioned at the bottom of the output of "data()". Again, the output does not show in Quarto,

```
data(package = .packages(all.available = TRUE))
```

One of the datasets available by default (in the standard package "datasets") is "mtcars". To use this dataset in our analysis, we type

```
data("mtcars")
```

This loads the dataset "mtcars" into the environment. Check the Environment pane in the Environment window. You should see an entry named "mtcars".

The data is now available as a dataframe in your R session. To investigate the data, we can list the first six lines with

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

In one webpage we saw that there is also a dataset "mpg" with information about fuel consumption of various cars. We try to load this datase with

```
data("mpg")
```

```
Warning in data("mpg"): data set 'mpg' not found
```

This command generates an error message. The reason is that the package "ggplot2", which contains this dataset is not available in our R session yet. If we do not want to load this package, we can set the "package" parameter in the data() command:

```
data(mpg, package="ggplot2")
```

Alternatively, we can first load the package and then the dataset.

```
library(ggplot2)
data(mpg)
```

Again, we list the head of the dataframe.

```
head(mpg)
```

```
# A tibble: 6 x 11
  manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
  <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

5.2 Importing an Excel-file

We have already loaded the package `readxl` above. The package contains a function `read_excel`. Take a look at the description of this function through `?read_excel`.

To demonstrate and test, we need data in Excel format. Let us use the Austrian results of the EU-elections 2014. Go to the webpage https://www.data.gv.at/, and select "Daten" - "Datensatz finden" (you may want to witch to English and then select "data" - "find dataset"). Enter "EU Wahl 2014" to the search field and hit Enter. Scroll down to "Ergebnisse der EU-Wahl 2014 (BMI)" and click the green button with the white "X". This will download the Excel-file to your download directory. Move or copy the file to your project directory.

Now, run

```
EU_elections <- read_excel(
  "eu-wahl2014-endgueltiges_ergebnis_mit_briefwahl.xlsx")
```

```
New names:
* `` -> `...6`
```

```
* `` -> `...7`
* `%` -> `%...9`
* `%` -> `%...11`
* `%` -> `%...13`
* `%` -> `%...15`
* `%` -> `%...17`
* `%` -> `%...19`
* `%` -> `%...21`
* `%` -> `%...23`
* `%` -> `%...25`
```

R alerts us that some of the variables were given new names. To view the first six lines and five columns of the dataset, enter

```
head(EU_elections, c(6, 5))
```

```
# A tibble: 6 x 5
  GKZ    Gebietsname       `Wahlbe-\r\nrechtigte` Wahl-\r\nbeteil-\r\n~1 Stimmen
  <chr>  <chr>                              <dbl>                  <dbl> <chr>
1 <NA>   <NA>                                  NA                     NA "abge-~
2 G00000 Österrreich                      6410602                  0.454 "29094~
3 G10000 Burgenland                        233920                  0.537 "12553~
4 G1A000 Burgenland Nord                   122224                  0.533 "65136"
5 G1A099 Wahlkarten - Bur~                      0                     NA "4902"
6 G1B000 Burgenland Süd                    111696                  0.541 "60397"
# i abbreviated name: 1: `Wahl-\r\nbeteil-\r\nigung\r\nin %`
```

The parameter `c(6, 5)` sets the number of rows to display to 6 and the number of columns to 5.

Alternatively, you may want to load the whole data frame into the viewer with

```
View(EU_elections)
```

When we look at the data, we find that there are some problems. Underneath the variable names, we see a row with mainly "NA" values. In columns 5-7 we see that there seems to be a second row for the variable header. Because of the strings in these fields, all the values in columns 5-7 are stored as characters (indicated by the "").

To avoid these problems, we can tell the function `read_excel()` to skip the first two lines.

```
EU_elections <- read_excel(
  "eu-wahl2014-endgueltiges_ergebnis_mit_briefwahl.xlsx",
  skip=2)
head(EU_elections, c(6, 7))
```

```
# A tibble: 6 x 7
  G00000 Österreich `6410602` `0.45385706365798406` `2909497` `85936` `2823561`
  <chr>  <chr>           <dbl>                 <dbl>     <dbl>   <dbl>     <dbl>
1 G10000 Burgenland     233920                 0.537    125533    5189    120344
2 G1A000 Burgenland~    122224                 0.533     65136    2655     62481
3 G1A099 Wahlkarten~         0                NA          4902     113      4789
4 G1B000 Burgenland~    111696                 0.541     60397    2534     57863
5 G1B099 Wahlkarten~         0                NA          5259     113      5146
6 G10099 Wahlkarten~         0                NA         10161     226      9935
```

Now, we have the problem that the first line of data is used as variable names. So, we also tell the function to **not** read column names

```
EU_elections <- read_excel(
  "eu-wahl2014-endgueltiges_ergebnis_mit_briefwahl.xlsx",
  skip=2,
  col_names = FALSE)
```

```
New names:
* `` -> `...1`
* `` -> `...2`
* `` -> `...3`
* `` -> `...4`
* `` -> `...5`
* `` -> `...6`
* `` -> `...7`
* `` -> `...8`
* `` -> `...9`
* `` -> `...10`
* `` -> `...11`
* `` -> `...12`
* `` -> `...13`
* `` -> `...14`
* `` -> `...15`
* `` -> `...16`
* `` -> `...17`
* `` -> `...18`
* `` -> `...19`
* `` -> `...20`
* `` -> `...21`
* `` -> `...22`
* `` -> `...23`
* `` -> `...24`
* `` -> `...25`
```

```
head(EU_elections, c(6, 7))
```

```
# A tibble: 6 x 7
  ...1   ...2                            ...3   ...4    ...5  ...6    ...7
  <chr>  <chr>                          <dbl>  <dbl>   <dbl> <dbl>   <dbl>
1 G00000 Österrreich                  6410602  0.454 2909497 85936 2823561
2 G10000 Burgenland                    233920  0.537  125533  5189  120344
3 G1A000 Burgenland Nord               122224  0.533   65136  2655   62481
4 G1A099 Wahlkarten - Burgenland Nord       0 NA        4902   113    4789
5 G1B000 Burgenland Süd                111696  0.541   60397  2534   57863
6 G1B099 Wahlkarten - Burgenland Süd        0 NA        5259   113    5146
```

Instead of `col_names = FALSE`, we can also provide a vector of variable names to the parameter `col_names`. This will give us a nice data frame to use.

```
EU_elections <- read_excel(
  "eu-wahl2014-endgueltiges_ergebnis_mit_briefwahl.xlsx",
  skip=2,
  col_names = c("GKZ", "Name", "Voters", "Turnout", "Votes", "invalid",
                "valid", "ÖVP", "ÖVP_percent", "SPÖ", "SPÖ_percent",
                "FPÖ", "FPÖ_percent", "GRÜNE", "GRÜNE_percent", "BZÖ",
                "BZÖ_percent", "NEOS", "NEOS_percent", "RECOS",
                "RECOS_percent", "ANDERS", "ANDERS_percent", "EUSTOP",
                "EUSTOP_percent"))
head(EU_elections, c(6, 7))
```

```
# A tibble: 6 x 7
  GKZ    Name                           Voters Turnout   Votes invalid   valid
  <chr>  <chr>                           <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1 G00000 Österrreich                   6410602   0.454 2909497   85936 2823561
2 G10000 Burgenland                     233920   0.537  125533    5189  120344
3 G1A000 Burgenland Nord                122224   0.533   65136    2655   62481
4 G1A099 Wahlkarten - Burgenland Nord        0  NA        4902     113    4789
5 G1B000 Burgenland Süd                 111696   0.541   60397    2534   57863
6 G1B099 Wahlkarten - Burgenland Süd         0  NA        5259     113    5146
```

5.3 Importing a CSV-file directly from the Internet

We pack two new topics under this heading: (1) reading CSV-files and (2) reading data directly from the Internet.

CSV is a common data format and stands for "Comma Separated Variables". This name, however, is misleading because in German speaking countries it is usually the semicolon (;), not the comma that separates variables.

R provides two functions for reading a CSV file: `read.csv()` and `read.csv2()`. The first version uses English standards (comma as variable separator and period as decimal separator), the second version uses German standards (semicolon as variable separator and comma as decimal separator). In both cases, you can define the respective separators with the parameters `sep=` and `dec=`.

Some data import functions in R allow you to specify a url instead of a local filepath for the file. The two CSV-functions are among them, `read_excel()`is not. When you go back to the https://www.data.gv.at/ webpage, to the entry "Ergebnisse der EU-Wahl 2014 (BMI)", you will see an orange icon marked "CSV". Right-click this icon and select Copy Link to copy the url into your computer's clipboard. Paste this url into the following call of `read.csv2()`.

```
csvurl <- paste("https://www.data.gv.at/katalog/dataset/",
                "2b10a91b-51d5-4e34-b992-8fd3a3121f0d/resource/",
                "a5ccea30-a505-4376-bfa3-fa2585c69192/download/",
                "eu-wahl2014-endgueltiges_ergebnis_mit_briefwahl.csv",
            sep="")
EU_elections_csv <- read.csv2(csvurl)
```

This does not work. The reason is again the issue with the column names, which are not unique. We can again skip the first two lines with `skip = 2`. Unfortunately, this function does not allow us to provide variable names as before. We can only set `header = FALSE` to prevent the function from interpreting the first line as variable names.

```
csvurl <- paste("https://www.data.gv.at/katalog/dataset/",
                "2b10a91b-51d5-4e34-b992-8fd3a3121f0d/resource/",
                "a5ccea30-a505-4376-bfa3-fa2585c69192/download/",
                "eu-wahl2014-endgueltiges_ergebnis_mit_briefwahl.csv",
            sep="")
EU_elections_csv <- read.csv2(csvurl,
    skip=2,
    header = FALSE)
head(EU_elections_csv, c(6, 7))
```

```
     V1                              V2     V3     V4     V5   V6     V7
1 G10000                     Burgenland 233920 53,66% 125533 5189 120344
2 G1A000                 Burgenland Nord 122224 53,29%  65136 2655  62481
3 G1A099   Wahlkarten - Burgenland Nord      0          4902  113   4789
4 G1B000               Burgenland S\xfcd 111696 54,07%  60397 2534  57863
5 G1B099 Wahlkarten - Burgenland S\xfcd      0          5259  113   5146
6 G10099        Wahlkarten - Burgenland      0         10161  226   9935
```

Note that the "ü" in "Burgenland Süd" is not read correctly. This issue is related to the import of the file from the Internet. German Umlaut characters are treated differently in various character systems, socalled "encodings". To fix this, we need to use the parameter

`encoding` and specify the correct encoding. An alternative fix would be to download the file to the local machine and then read it into R in a second step.

We use the following code to read the file with the correct encoding:

```
csvurl <- paste("https://www.data.gv.at/katalog/dataset/",
                "2b10a91b-51d5-4e34-b992-8fd3a3121f0d/resource/",
                "a5ccea30-a505-4376-bfa3-fa2585c69192/download/",
                "eu-wahl2014-endgueltiges_ergebnis_mit_briefwahl.csv",
            sep="")
EU_elections_csv <- read.csv2(csvurl,
    skip=2,
    header = FALSE,
    encoding = "latin1")
head(EU_elections_csv, c(6, 7))
```

```
     V1                           V2     V3     V4     V5   V6     V7
1 G10000                   Burgenland 233920 53,66% 125533 5189 120344
2 G1A000               Burgenland Nord 122224 53,29%  65136 2655  62481
3 G1A099 Wahlkarten - Burgenland Nord      0           4902  113   4789
4 G1B000                Burgenland Süd 111696 54,07%  60397 2534  57863
5 G1B099  Wahlkarten - Burgenland Süd      0           5259  113   5146
6 G10099       Wahlkarten - Burgenland      0          10161  226   9935
```

Now the Umlaut characters are displayed correctly.

With `read.csv2()`, we cannot specify the variable names and have to set the correct names in a separate step. With the function `colnames()` we can get and set the column names of a data frame.

```
colnames(EU_elections_csv) <- c("GKZ", "Name", "Voters", "Turnout",
      "Votes", "invalid", "valid", "ÖVP", "ÖVP_percent", "SPÖ",
      "SPÖ_percent", "FPÖ", "FPÖ_percent", "GRÜNE", "GRÜNE_percent",
      "BZÖ", "BZÖ_percent", "NEOS", "NEOS_percent", "RECOS",
      "RECOS_percent", "ANDERS", "ANDERS_percent", "EUSTOP",
      "EUSTOP_percent")
head(EU_elections_csv, c(6, 7))
```

```
     GKZ                           Name Voters Turnout  Votes invalid  valid
1 G10000                   Burgenland 233920  53,66% 125533    5189 120344
2 G1A000               Burgenland Nord 122224  53,29%  65136    2655  62481
3 G1A099 Wahlkarten - Burgenland Nord      0           4902     113   4789
4 G1B000                Burgenland Süd 111696  54,07%  60397    2534  57863
5 G1B099  Wahlkarten - Burgenland Süd      0           5259     113   5146
6 G10099       Wahlkarten - Burgenland      0          10161     226   9935
```

6 Viewing data

In any statistical software, it is always a good idea, to check and validate your data before you use it. That the routine you used to read the data does not generate an error message, does not imply that your data has been read *correctly*. Some data may be distorted, read in the wrong format, some numbers may end up in the wrong variables, etc.

I do not want to go into depth about checking and verifying data. At this stage, I just want to collect a few tools that you can use to look at your data.

6.1 `head()`, `tail()`, and `data.table()`

We have already seen the command `head()` at work above. By default, it shows the first 6 rows for all the variables in the data frame. When the variables do not fit horizontally, the function prints them in blocks underneath one another. This output may be somewhat confusing and difficult to read. As you saw above, you can explicitly specify the numbers of rows and columns to display.

The command `tail()` does the same as `head()`, but with the last six lines of the dataframe. The function `data.table()`, which is available in the package "data.table", combines the functionality of `head()` and `tail()` and shows the first and the last six lines of the dataset.

6.2 `View()`

The command `View()` (note the capital "V") displays the data frame in a spreadsheet like form. The output is placed in a new tab in the Source window of RStudio. This is probably the best option to inspect your data because you can scroll through all the rows and columns. The function may fail, however, when you have to deal with a very large data frame.

When you move the cursor over the name of a column, you will see more information about this variable in a tooltip. Run `View(EU_elections)`, place the cursor over "Turnout", and after a second you will see that this is column 4 of the data frame, that the format is "numeric" and that the numbers are in the range "0.1 - 0.8".

You can call this function also from the Environment pane of the Environment window. To the right of every listed data frame there is a light-blue icon with a data grid. Clicking that will load this data frame into `View()`.

6.3 The glimpse command

Somewhere between `head()` and `tail()` on the one hand and `View()` on the other is the `glimpse()` command from the package "dplyr". This package is loaded automatically when you load the package "tidyverse" (see below).

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
glimpse(EU_elections)
```

```
Rows: 2,707
Columns: 25
$ GKZ            <chr> "G00000", "G10000", "G1A000", "G1A099", "G1B000", "G1B0~
$ Name           <chr> "Österrreich", "Burgenland", "Burgenland Nord", "Wahlka~
$ Voters         <dbl> 6410602, 233920, 122224, 0, 111696, 0, 0, 10479, 10479,~
$ Turnout        <dbl> 0.4538571, 0.5366493, 0.5329232, NA, 0.5407266, NA, NA,~
$ Votes          <dbl> 2909497, 125533, 65136, 4902, 60397, 5259, 10161, 5681,~
$ invalid        <dbl> 85936, 5189, 2655, 113, 2534, 113, 226, 167, 157, 10, 4~
$ valid          <dbl> 2823561, 120344, 62481, 4789, 57863, 5146, 9935, 5514, ~
$ ÖVP            <dbl> 761896, 37331, 18266, 1342, 19065, 1674, 3016, 2111, 18~
$ ÖVP_percent    <dbl> 0.2698351, 0.3102024, 0.2923449, 0.2802255, 0.3294852, ~
$ SPÖ            <dbl> 680180, 40361, 20740, 1648, 19621, 1621, 3269, 1126, 98~
$ SPÖ_percent    <dbl> 0.2408944, 0.3353802, 0.3319409, 0.3441219, 0.3390941, ~
$ FPÖ            <dbl> 556835, 21446, 11485, 632, 9961, 617, 1249, 806, 746, 6~
$ FPÖ_percent    <dbl> 0.1972102, 0.1782058, 0.1838159, 0.1319691, 0.1721480, ~
$ GRÜNE          <dbl> 410089, 9686, 5581, 570, 4105, 603, 1173, 777, 691, 86,~
$ GRÜNE_percent  <dbl> 0.14523823, 0.08048594, 0.08932315, 0.11902276, 0.07094~
$ BZÖ            <dbl> 13208, 280, 154, 12, 126, 10, 22, 15, 12, 3, 4, 4, 0, 4~
$ BZÖ_percent    <dbl> 0.004677781, 0.002326664, 0.002464749, 0.002505742, 0.0~
$ NEOS           <dbl> 229781, 5955, 3341, 366, 2614, 385, 751, 422, 375, 47, ~
$ NEOS_percent   <dbl> 0.08137986, 0.04948315, 0.05347226, 0.07642514, 0.04517~
$ RECOS          <dbl> 33224, 916, 493, 37, 423, 31, 68, 60, 57, 3, 1, 1, 0, 1~
$ RECOS_percent  <dbl> 0.011766702, 0.007611514, 0.007890399, 0.007726039, 0.0~
$ ANDERS         <dbl> 60451, 1393, 781, 63, 612, 81, 144, 73, 68, 5, 7, 7, 0,~
$ ANDERS_percent <dbl> 0.021409490, 0.011575151, 0.012499800, 0.013155147, 0.0~
$ EUSTOP         <dbl> 77897, 2976, 1640, 119, 1336, 124, 243, 124, 112, 12, 2~
$ EUSTOP_percent <dbl> 0.02758821, 0.02472911, 0.02624798, 0.02484861, 0.02308~
```

This function displays the size of the data frame (number of rows and number of columns) and then for every variable its name, its data type, and the first few values (whatever fits into the available horizontal space).

6.4 Nicer looking tables

There are a number of packages that provide functions for nicely formatted tables. One example is the function `gt()` from the "gt" package. Try it out with the following code.

```
library(gt)
gt(EU_elections[1:8, 1:7])
```

GKZ	Name	Voters	Turnout	Votes	invalid	valid
G00000	Österrreich	6410602	0.4538571	2909497	85936	2823561
G10000	Burgenland	233920	0.5366493	125533	5189	120344
G1A000	Burgenland Nord	122224	0.5329232	65136	2655	62481
G1A099	Wahlkarten - Burgenland Nord	0	NA	4902	113	4789
G1B000	Burgenland Süd	111696	0.5407266	60397	2534	57863
G1B099	Wahlkarten - Burgenland Süd	0	NA	5259	113	5146
G10099	Wahlkarten - Burgenland	0	NA	10161	226	9935
G10100	Eisenstadt(Stadt)	10479	0.5421319	5681	167	5514

I added `[1:8, 1:7]` to the name of the data frame to restrict the table to the first eight rows and the first seven columns. This is solely for display purposes for the PDF-output. When you leave this out and use `gt(EU_elections)`, the whole data frame will be shown in the table.

When you call this function from the Console, the output will show up in the Viewer pane of the Files window. Click the Zoom button to get a larger view in a separate window. With the Export button you can also save the table as a graphic, as a webpage, or copy it to the clipboard to paste it into another program.

7 Conclusion

This presentation provides a first quick overview of R, RStudio, and some of their exciting new features. It roughly contains the material that we covered in our first workshop. None of the aspects was covered comprehensively. Practically all R commands that I have mentioned offer more parameters that influence their behavior.