

Lecture 1: Installing R and RStudio, Sneak Preview

Bas Machielsen

3/2/2020

Introduction

This lecture covers three aspects:

- ▶ Installing R and R Studio
- ▶ Familiarizing ourselves with the layout and functionality of RStudio
- ▶ A sneak preview of R's possibilities

What is R?

Wikipedia:

R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.[6] The R language is widely used among statisticians and data miners for developing statistical software[7] and data analysis.[8] Polls, data mining surveys, and studies of scholarly literature databases show substantial increases in popularity;[9] as of February 2020, R ranks 13th in the TIOBE index, a measure of popularity of programming languages.[10]

Popularity of R

May 2020	May 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	17.07%	+2.82%
2	1	▼	Java	16.28%	+0.28%
3	4	▲	Python	9.12%	+1.29%
4	3	▼	C++	6.13%	-1.97%
5	6	▲	C#	4.29%	+0.30%
6	5	▼	Visual Basic	4.18%	-1.01%
7	7		JavaScript	2.68%	-0.01%
8	9	▲	PHP	2.49%	-0.00%
9	8	▼	SQL	2.09%	-0.47%
10	21	▲	R	1.85%	+0.90%

Popularity of R

R usage is on the rise! What are the reasons?

- ▶ Accessibility
- ▶ Open Source
- ▶ Large community of programmers
- ▶ Amazing Functionality:
 - ▶ Statistics
 - ▶ Machine Learning
 - ▶ OCR
 - ▶ Data Visualization
 - ▶ Web Scraping
 - ▶ Text Mining

Installing R

- ▶ R is the **programming language**, RStudio is a client that makes it accessible.
- ▶ Hard to imagine programming in R without RStudio.
 - ▶ Data Viewer
 - ▶ Keep track of environment
 - ▶ Access to function and package documentation
- ▶ Download the most recent version of R from <https://cran.rstudio.com/>
- ▶ The website will offer you the correct version for your OS
- ▶ As of 16 May, the most recent R version is 4.0.0!











Installing RStudio

- ▶ Go to the official RStudio website
- ▶ Or Google: Download RStudio

All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy.

RStudio 1.2 requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).

OS	Download	Size	SHA-256
Windows 10/8/7	 RStudio-1.2.5033.exe	149.83 MB	7fd3bc1b
macOS 10.12+	 RStudio-1.2.5033.dmg	126.89 MB	b67c9875
Ubuntu 14/Debian 8	 rstudio-1.2.5033-amd64.deb	96.18 MB	89dc2e22
Ubuntu 16	 rstudio-1.2.5033-amd64.deb	104.14 MB	a1591ed7
Ubuntu 18/Debian 10	 rstudio-1.2.5033-amd64.deb	105.21 MB	08eaa295
Fedora 19/Red Hat 7	 rstudio-1.2.5033-x86_64.rpm	120.23 MB	38cf43c6
Fedora 28/Red Hat 8	 rstudio-1.2.5033-x86_64.rpm	120.87 MB	452bc0d0
Debian 9	 rstudio-1.2.5033-amd64.deb	105.45 MB	27c59722
SLES/OpenSUSE 12	 rstudio-1.2.5033-x86_64.rpm	98.87 MB	9c1e200c
OpenSUSE 15	 rstudio-1.2.5033-x86_64.rpm	106.91 MB	98fd2258

Windows Users

- ▶ Select the appropriate installer (for Windows: .exe)
- ▶ Follow the setup wizard: the default options are fine. Make sure to select '64-bit Files', 'Message translations' and 'Core Files', under components.
- ▶ Don't specify the startup options but accept the default.
- ▶ Determine whether you want shortcuts on your desktop, wait until the wizard has finished and start Rstudio.

MacOS Users

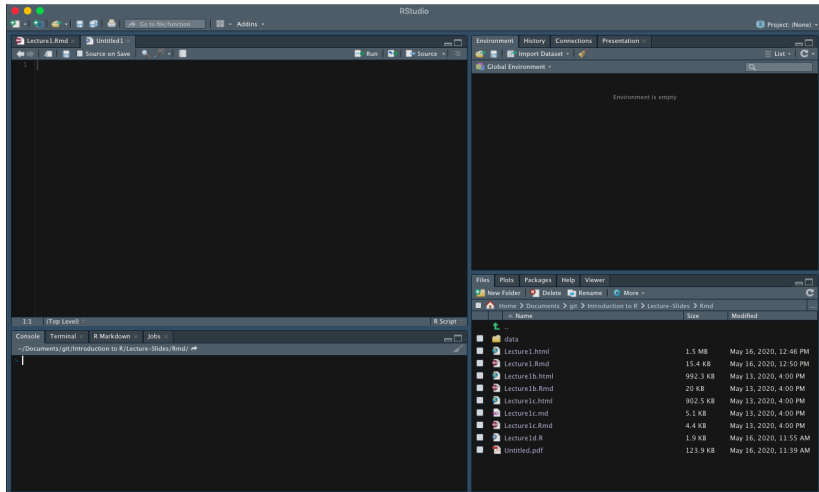
- ▶ Download and install the most recent version of R from <https://cran.rstudio.com/> (This should be straightforward)
- ▶ Go to <https://rstudio.com/products/rstudio/download/#download>
- ▶ Select the appropriate installer (for Mac: .dmg)
- ▶ Double-click it to open, and then drag and drop it to your applications folder.

UNIX/Linux

- ▶ Download and install the most recent version of R from <https://cran.rstudio.com/> (This should be straightforward)
- ▶ Go to <https://rstudio.com/products/rstudio/download/#download>
- ▶ Select the appropriate installer (for Ubuntu/Debian: .deb), and an install wizard will guide you through the rest of the process.

Getting Started: Setting our working directory

Once you've opened Rstudio, the standard layout will look something like this:



The RStudio User Interface

By default, RStudio has 4 panels:

- ▶ A script editor
- ▶ The console
- ▶ The environment
- ▶ Viewer

What do you do with each of them?

RStudio User Interface [1]

- ▶ Console:

The console is a place where you enter code, and see the output of the code (in general, but not always).

- ▶ A script editor:

The script editor is the place where you enter your code with the objective of storing it. If you don't see it, click the + icon on the upper left of Rstudio -> New R Script.

RStudio User Interface [2]

- ▶ Environment:

This is a place to keep track of your objects: data.frames, variables, but even connections and history (we'll get there).

- ▶ Viewer:

By default, the viewer Works as Finder on Mac, or Windows Explorer, but it also shows the output of Visualizations under Plots, it gives an overview of all your installed and loaded packages, and shows **function documentation**.

RStudio

Let's first open our `.Rmd` file, the file that contains these lecture notes from the course Github page.

In RStudio, we go to File > Open File, and we select the markdown document `Lecture1.Rmd` which we've just downloaded. Scroll to the part in the lecture where you can see this.

Hint: You can use the search function in your operating system to look for specific terms (and it also includes a Find + Replace tool!).

Getting our Working Directory

We will start by specifying our **working directory**. Our working directory determines from which file path we will create, or load our files by default. It is advisable to put your work somewhere on a hard-drive. Sometimes, a network server frustrates writing and saving files from R to a server-environment. You can see your default working directory by the following command:

```
getwd()
```

Put your mouse cursor in the above chunk, and press CTRL+Enter (cmd + Enter), or alternatively, type `getwd()` in the console, and press enter.

Setting our Working Directory

Now, let's set our working directory. You set your working directory by using the `setwd()` command, specifying a directory between brackets, and using quotation marks:

```
setwd("exampledirectory/bas").
```

The file path to your working directory depends on your operating system: In Windows, you specify your working directory using:

```
setwd("C:/Documents/My/Working/Directory")
```

On a Mac: you specify the directory starting from the root directory with a `/`:

```
setwd("/Users/user/Documents/MyWorkingDirectory")
```

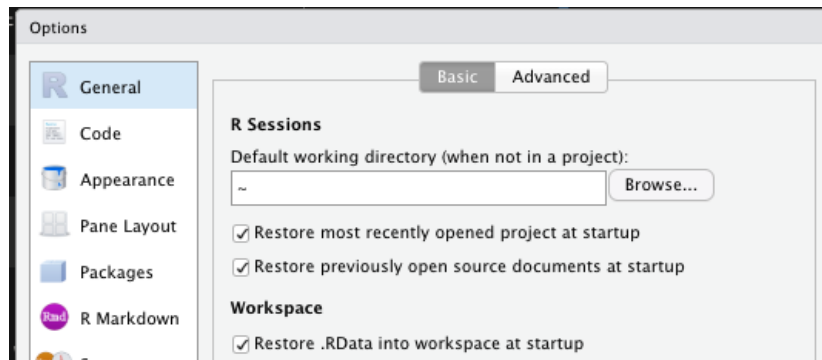
On a UNIX system:

```
setwd("/home/user/Documents/MyWorkingDirectory")
```

Setting a Working Directory

You can also use the GUI of Rstudio to change your working directory: Session > Set Working Directory > Choose Location. Let's now consider to get a *default* R Working Directory.

A default working directory is the directory that is your Working Directory if you start up RStudio and do not indicate any other WD. You can go to RStudio > Preferences > General:



Setting a default directory

Alternatively, you can change your *.Rprofile* file. An *.Rprofile* file is a script that automatically runs. You should be careful with putting things into this file, but a default working directory could be a good thing to put in there.

The easiest way to change your *.Rprofile* is the following:

```
usethis::edit_r_profile()
```

Don't remove anything from this file, and simply add the command: `setwd("File/Path/To/Your/New/WD/")`

Make sure to specify the file path in a way appropriate to your OS (e.g. Windows users: `C://My/Dir`)

Changing the default look of Rstudio

In order to make this a little bit less boring, let us pick a nice custom look of Rstudio first (this will also make you look like an experienced programmer):

Let's go to Tools > Global Options > Appearance, pick a nice Editor theme and/or font, and click apply!

Installing our first packages

Now, our RStudio is fully set-up! We can continue by installing our **first packages!**

Let us install the **tidyverse**! In fact, tidyverse is not so much a package as it is a **set** of packages facilitating data treatment. Try and run the following code chunk:

```
install.packages('tidyverse')
```

Using packages

Packages contain functions that help us in doing what we want to do in an efficient, and often easy way. Not using packages comes down to attempting to reinvent the wheel

We would have to do everything using base R syntax, whereas some other people have already used base R syntax to write very complicated functions, of which we can make use.

Installation of tidyverse

After we've installed our package, which just means that R put the appropriate files in the appropriate folder on your computer, we have to load it before we can use its functions. Loading a package is done as follows:

```
library(tidyverse)
```

And it will give you some output akin to the following picture:

```
> library(tidyverse)
── Attaching packages ─────────────────────────── tidyverse 1.3.0 ─
✔ ggplot2 3.2.1    ✔ dplyr   0.8.3
✔ tidyr   1.0.0    ✔ stringr 1.4.0
✔ readr   1.3.1    ✔ forcats 0.4.0
✔ purrr   0.3.3
── Conflicts ─────────────────────────── tidyverse_conflicts() ─
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()     masks stats::lag()
> |
```

Packages

This is an overview of the attached packages in the tidyverse, followed by an overview of **Conflicts**. Conflicts happen when two packages use the same function names (we'll get to this in a future lecture).

In particular, RStudio automatically loads a number of fundamental packages without letting you know, among which is **stats**.

`library(tidyverse)` loads **dplyr**, and **dplyr** contains the functions `filter()` and `lag()`, which are also functions in **stats**!

Therefore, R decides to replace the functions in **stats** by the functions **dplyr**. Why not the other way around? Because we've loaded **dplyr** after **stats**.

In general, conflicts can be seen using the following command:

```
conflicts()
```


Installing other packages

We can also install several packages at the same time. In order to do so, we have to **concatenate** the names of various packages to the argument of `install.packages`:

```
install.packages(c("stargazer", "zoo"))
```

Concatenation is used quite often in R. Every time you want to tell a function in R to evaluate apply itself to not only 1, but more arguments, you have to concatenate them. We will talk about basic R syntax in the next lecture, but consider this:

```
print(1,2)
```

```
## [1] 1
```

```
print(c(1,2))
```

```
## [1] 1 2
```

Objects and names

We have already seen a small demonstration of R's logic, which can be summarized as **doing things to objects**:

Objects can be (for example):

- ▶ Data.frames
- ▶ Functions
- ▶ Variables
- ▶ Regression output
- ▶ A World map

Functions can encompass:

- ▶ Taking the mean
- ▶ Performing a regression
- ▶ Making a graph

Brief showcase of R's capabilities

Now we've finished our start-up, I will show some of R's standard and non-standard functionalities:

- ▶ Data wrangling (Data tidying)
- ▶ Data visualization
- ▶ Web scraping
- ▶ Regression

Data wrangling

Let's have a look at a dataset with historical GDP per capita numbers:

```
library(readxl)

datasource <- "../data/GDPperCapita_Compact.xlsx"

gdppercapita <- read_xlsx(datasource)
```

Data wrangling [2]

This is what the dataset looks like:

ccode	country name	1500	1501	1502	1503	1504
4	Afghanistan	NA	NA	NA	NA	NA
8	Albania	NA	NA	NA	NA	NA
12	Algeria	NA	NA	NA	NA	NA
24	Angola	NA	NA	NA	NA	NA
32	Argentina	NA	NA	NA	NA	NA
51	Armenia	NA	NA	NA	NA	NA
36	Australia	NA	NA	NA	NA	NA
40	Austria	NA	NA	NA	NA	NA
31	Azerbaijan	NA	NA	NA	NA	NA
48	Bahrain	NA	NA	NA	NA	NA

Data wrangling [3]

And this is how easy it is to 'pivot' the data into long format:

```
library(tidyverse)
range <- 3:518 #Which columns do you want to wrangle

gdppercapita <- gdppercapita %>%
  pivot_longer(all_of(range),
               names_to = "year",
               values_to = "gdppercapita")
```

Data wrangling [4]

This is the result:

ccode	country name	year	gdppercapita
4	Afghanistan	1500	NA
4	Afghanistan	1501	NA
4	Afghanistan	1502	NA
4	Afghanistan	1503	NA
4	Afghanistan	1504	NA
4	Afghanistan	1505	NA
4	Afghanistan	1506	NA
4	Afghanistan	1507	NA
4	Afghanistan	1508	NA
4	Afghanistan	1509	NA

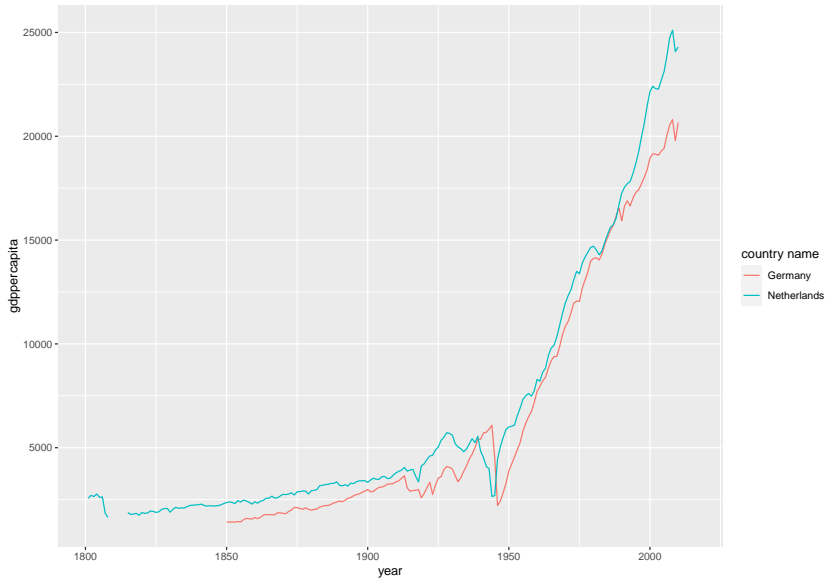
Data visualization

Let us now select some countries and create a graph:

```
countries <- c("Netherlands", "Germany")

gdppercapita %>%
  mutate(year = as.numeric(year)) %>%
  filter(is.element(`country name`, countries),
         year > 1800) %>%
  ggplot(aes(x = year,
             y = gdppercapita,
             group = `country name`,
             color = `country name`)) + geom_line()
```


Data visualization [2]



Web scraping

Web scraping, the art of taking information on web pages and organizing them in a systematic manner, is something else which is very easy in R.

Let us attempt to scrape the table we showed before, about popularity of programming languages here:

```
library(rvest)

read_html("https://www.tiobe.com/tiobe-index/") %>%
  html_nodes("#top20") %>%
  html_table()
```

Result

May.2019	Change	Programming.Language	Ratings	Change.1
2	NA	C	17.07%	+2.82%
1	NA	Java	16.28%	+0.28%
4	NA	Python	9.12%	+1.29%
3	NA	C++	6.13%	-1.97%
6	NA	C#	4.29%	+0.30%
5	NA	Visual Basic	4.18%	-1.01%
7	NA	JavaScript	2.68%	-0.01%
9	NA	PHP	2.49%	-0.00%
8	NA	SQL	2.09%	-0.47%
21	NA	R	1.85%	+0.90%

Regression

Regressions are also very straightforward in R (we do some data cleaning beforehand):

```
model1 <- gdppercapita %>%  
  filter(`country name` == "Netherlands") %>%  
  mutate(year = as.numeric(year)) %>%  
  lm(formula = gdppercapita ~ year)
```

Regression

And the results show a default output of regression coefficients, significance, etc.

```
summary(model1)
```

```
##
## Call:
## lm(formula = gdppercapita ~ year, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -6256  -2929  -1682   1248   16229
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -94721.319   7270.125  -13.03  <2e-16 ***
## year          53.297     3.827    13.93  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4441 on 210 degrees of freedom
## (304 observations deleted due to missingness)
## Multiple R-squared:  0.4802, Adjusted R-squared:  0.4777
## F-statistic: 194 on 1 and 210 DF, p-value: < 2.2e-16
```