

Learning R — Getting Started

Felix Lennert

2020-05-03

Introductory remarks

Dear student, in the following, you will receive a gentle introduction to R. In the first part, I will start with what RStudio is and how you can install it, plus some recommended settings. Then I will introduce you to stuff I wish I had known earlier, as it makes my daily life a million times easier: RStudio projects and GitHub. Since the course requires you to hand in take-home exams written in RMarkdown, a brief tutorial is added as well. After you have familiarized yourself with these things, we can go *in medias res* and start out with actual R coding: how to install packages, how to perform arithmetic operations in R, how to assign values to objects, and the different classes of objects.

There are loads of resources online and I do not claim this introduction to be exhaustive. To acknowledge this shortcoming, every part ends with links leading you to further resources. However, the fact that I included them probably means that I have read them and more or less integrated their content into the tutorial – so keep striving for more resources (and send me links)!

In general, this tutorial was heavily inspired by Richard Cotton’s “Learning R” (Cotton 2013) and Hadley Wickham’s and Garrett Golemund’s “R for Data Science” (abbreviated with R4DS). The latter can be found online (Wickham and Golemund 2016). We will not immediately start out with the packages from the tidyverse (although some strong points have been made in favor of doing so right from the start). I will rather try to build some sort of foundation from where we can proceed to the tidy packages in the following lessons. Hence, one can also understand this tutorial as an introduction to the *tidyverse* (or *hadleyverse*, as it was named originally), even though I will not introduce it in this very first part. When it comes to what I would refer to as the “daily workflow” with R and RStudio, Jennifer Bryan’s blog articles have been a big inspiration for me. You will stumble across her work when we deal with the purrr package as well.

When looking at RMarkdown, I will mainly build on “R Markdown: The Definitive Guide” – which is also freely available online (Xie, Allaire, and Golemund 2018).

Getting started: installing R and RStudio

R

For downloading R, just visit the website of the Comprehensive R Archive Network (CRAN). CRAN is simply a network of ftp and web servers all around the world. Here, things related to R (code, documentation, etc.) are stored and can be downloaded. On top of the page, you will find a box with three links that refer to different versions depending on your operating system. Choose the one that applies. Thereafter, just click the link for downloading the latest version (we will work with R 4.0.0). If you are on a Mac (as I am), it will look like this:

After the download is finished, just execute the installer and, when it is done, you are good to go.

New versions of R are released multiple times per year. If you want to update your R, the process is the same as installing it from scratch: go to <https://cran.r-project.org>, download the latest version, install it, and that is it.



R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) [here](#). Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the [old](#) directory.

[CRAN](#)
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

[About R](#)
[R Homepage](#)
[The R Journal](#)

[Software](#)
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

[Documentation](#)
[Manuals](#)
[FAQs](#)
[Contributed](#)

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

Package binaries for R versions older than 3.2.0 are only available from the [CRAN archive](#) so users of such versions should adjust the CRAN mirror setting (<https://cran-archive.r-project.org>) accordingly.

R 4.0.0 "Arbor Day" released on 2020/04/24

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type

```
md5 R-4.0.0.pkg
```

in the *Terminal* application to print the MD5 checksum for the R-4.0.0.pkg image. On Mac OS X 10.7 and later you can also validate the signature using

```
pkgutil --check-signature R-4.0.0.pkg
```

Click here! Latest release:

[R-4.0.0.pkg](#) (notarized and signed)
SHA1-
hash: 9f0ca09d272694133024fd9af48f6446a5ba1641
(ca. 84MB)

R 4.0.0 binary for macOS 10.13 (High Sierra) and higher, signed and notarized package. Contains R 4.0.0 framework, R.app GUI 1.71 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 6.7. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires [XQuartz](#) to be installed since it is no longer part of OS X. Always re-install XQuartz when upgrading your macOS to a new major version.

Important: this release uses Xcode 10.1 and GNU Fortran 8.2. If you wish to compile R packages from sources, you will need to download and GNU Fortran 8.2 - see the [tools](#) directory.

[NEWS](#) (for Mac GUI)

News features and changes in the R.app Mac GUI

[Mac-GUI-1.70.tar.gz](#)

MD5-hash: b1ef5f285524640680a22965bb8800f8

Sources for the R.app GUI 1.70 for Mac OS X. This file is only needed if you want to join the development of the GUI, it is not intended for regular users. Read the `INSTALL` file for further instructions.

Figure 1: Fig. 1: the CRAN webpage

RStudio

Download and installation

When you are on a Windows or Mac machine and you click the R icon, a window (the so-called Mac or Windows GUI – graphic user interface) that looks pretty much like your machine’s terminal will appear. You could now just type R code in there and execute it – and, in fact, that is how the users did it the ancient way. In my opinion, this is fairly inconvenient. Luckily, we have progressed a lot from that and come up with IDEs (integrated development environments) for R. The most popular among them is RStudio which we will use as well.

To install RStudio, just click on this link, choose the right version (i.e., RStudio Desktop – Open Source License), and hit the download button. After downloading it, you simply install it, and thereafter you are good to go.

Please note that you have to install both, R and RStudio. The latter is only a tool that helps you working with the former.

Setting up RStudio

After installing RStudio, you can open it just like every other application on your machine. When you open it for the very first time, a window will appear that looks like this:

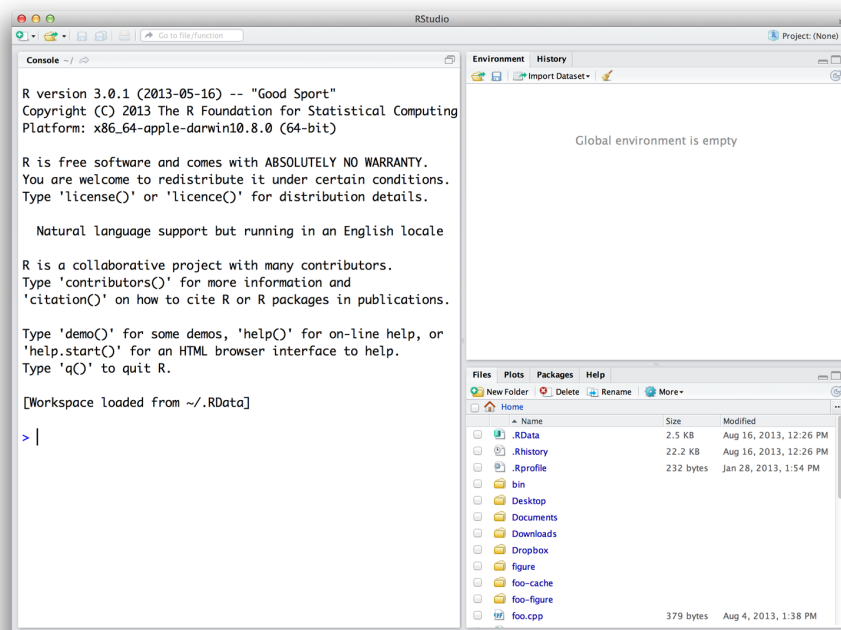


Figure 2: Fig. 2: RStudio; obtained from Grolemond (2014)

As you can see, there are three panes. When you open a script (for instance, by clicking File >> New File >> R Script), a fourth one will appear in the upper part on the left side. If you open multiple scripts, they will be organized in tabs as well¹.

Every pane contains different things:

- By default, the console can be found in the lower left pane. This is basically how the GUI would look like. You can either manually enter commands, and execute them by pressing return, or write

¹more on scripts at the end of this document.

code in the script (upper left pane), and run it by either clicking “Run” or hitting cmd/ctrl+return (Mac/Windows). Then, the console will show, (1), the code you ran and, (2), depending on what you executed, the output.

- In the upper right pane, there are two tabs: the environment and the history. The former will show you all the objects you have defined, the latter all the commands you have executed.
- The lower left pane now consists of four tabs: “Files” simply shows you the files that are stored in the working direction you are in. “Plots” is only used if you have plotted something – and is then called automatically, so you will probably never click on it. In “Packages”, as the name hints on, all the packages that are installed are listed. In theory, you could call them from there as well – but I strongly recommend not to. And finally, the “Help” tab provides you with documentations of packages etc. This sounds handy, but I hardly use it – simple googling or calling the `?[name_of_function]` has worked out best for me so far.

In the following, I will tell you more about what has appeared to work best *FOR ME*:

Disclaimer: *Every user has their own preferences when it comes to their preferred setup. Hence, you should see the following paragraphs only as recommendations that originate from my experience.*

As you may have noticed, there are a lot of different things RStudio provides you with. However, in every-day use, you will mostly use these five: the script you are actually working on. The console for seeing what your code has produced. The environment for a quick overview of the objects you are actually working with. The “Files” section for seeing the files in your working directory (this is where R projects will come in extremely handy). The “Plots” section for seeing your visualizations.

As the latter opens by itself as soon as you plot something, I mainly use the former four. However, if you use RStudio projects – as you definitely should – you will only occasionally need the “Files” section. Furthermore, your screen is wider than it is high. Hence, vertical space is scarcer than horizontal. At the heart of your coding lies your script, you should therefore give it the utmost space possible. On the right then there is space for two panes you will always have to give quick glimpses: the console and the environment. I put the former to the bottom and the latter on top. When the “Plots” section opens up, you will manually have to return to your former tab. Therefore, I put it to the lower left side where it does not bother me and I can minimize it with one click when I do not need it anymore.

My RStudio layout looks like this:

How you accomplish this? Preferences >> Pane Layout.

Other strongly recommended settings (ordered by Options section):

- General: Never save your work space, this makes you lazy (read more here)
- Code: some “Display” settings make your life easier; also set default encoding to UTF-8 in the “Save” section
- Appearance: check out some themes (I use: RStudio theme “Modern”, Editor font “Courier”, Font size 11, and Editor theme “Cobalt”)
- Panes layout: feel free to set them up the way I did

More can be found here.

Further links

- If you require a more extensive description of how to install R and RStudio, click here: Hands-on Programming with R – Appendix A, a book by one of the co-authors of R for Data Science.
- If you are already more advanced, you can read more useful stuff on using R and RStudio by two RStudio employees: What they forgot to teach you about R

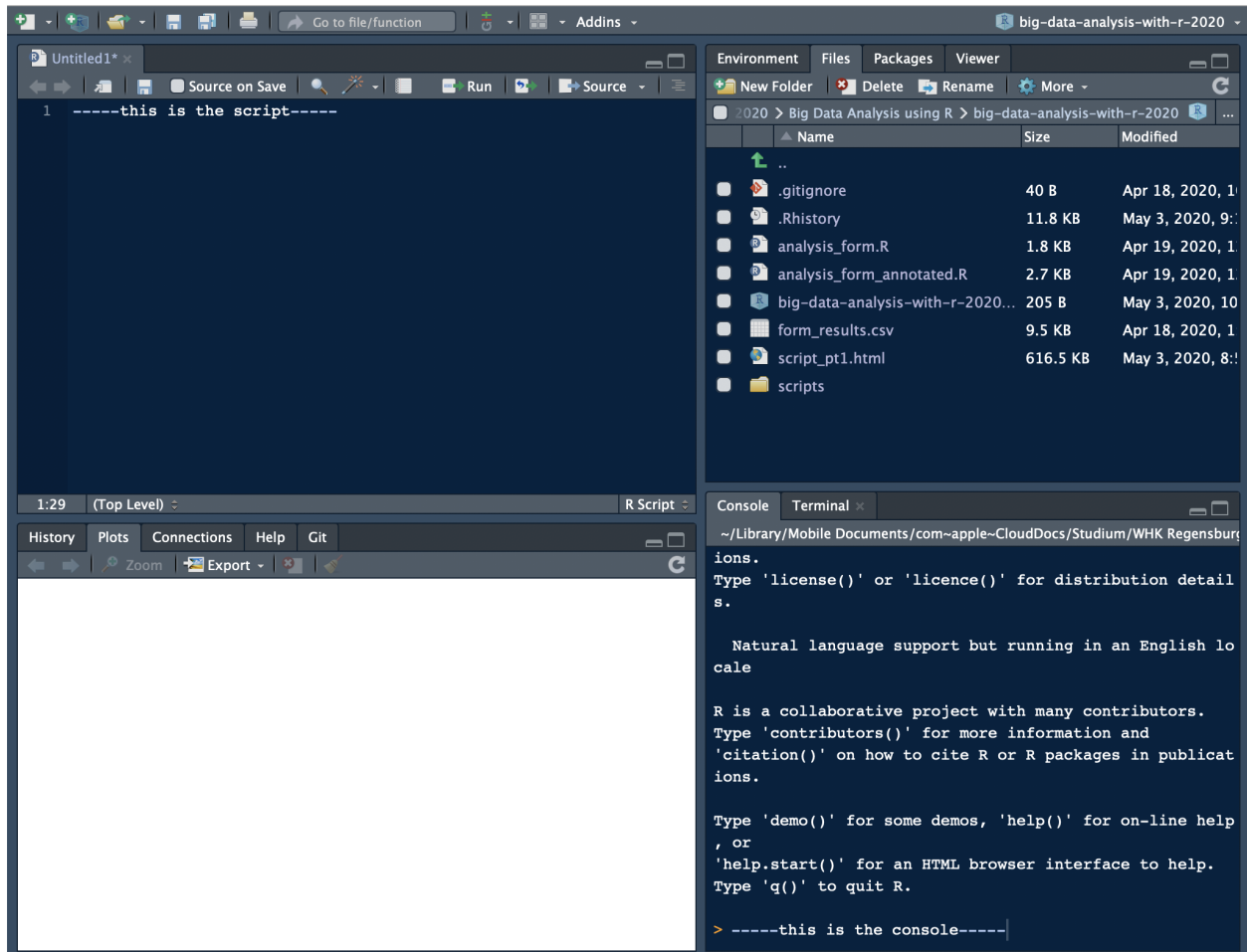


Figure 3: Fig. 3: my RStudio layout

Not ready to start coding yet: some remarks on your daily workflow

As some of you are beginners, it might be hard for you to see the point in setting up projects and a GitHub account already. The intermediate and advanced users among you, who are not familiar with projects and GitHub yet though, might also wonder what they would need it for: working with R has gone pretty well in the past, so why should you change this running system?

I start out making points on why using Projects is useful. Then, I will provide step-by-step guidance on how to set them up. Since using GitHub is not that straight-forward, I will motivate why to use it and then link to a bigger tutorial covering the setup process (again by Jennifer Bryan, a statistic professor who also works at RStudio).

RStudio projects

Motivation

If you analyze data with R, one of the first things you do is to load in the data that you want to perform your analyses on. Then, you perform your analyses on them, and save the results in the (probably) same directory.

When you load a data set into R, you might use the `readr` package and do `read_csv(absolute_file_path.csv)`. This becomes fairly painful if you need to read in more than one data set. Then, relative paths (i.e., where you start from a certain point in your file structure, e.g., your file folder) become more useful. How you CAN go across this is to use the `setwd(absolute_file_path_to_your_directory)` function. Here, `set` stands for set and `wd` stands for working directory. If you were not sure about what the current working directory actually is, you can use `getwd()` which is the equivalent to `setwd(file_path)`. This enables you to read in a data set – if the file is in the working directory – by only using `read_csv(file_name.csv)`.

However, if you have ever worked on an R project with other people in a group and exchanged scripts regularly, you may have encountered one of the big problems with this `setwd(file_path)` approach: as it only takes absolute paths like this one: “/Users/felixlennert/Library/Mobile Documents/com.apple.CloudDocs/Studium/WHK Regensburg/Kurse/SS 2020/Big Data Analysis/big-data-analysis-with-r-2020/scripts/”, no other person will be able to just run this script without making any changes². There are no two machines which have the exact same file structure.

This is where RStudio Projects come into play: they make every file path relative. The Project file (ends with `.Rproj`) basically sets the working directory to the folder it is in. Hence, if you want to send your work to a peer or a teacher, just send a folder which also contains the `.Rproj` file and they will be able to keep on working on your project without the hassle of pasting file paths into `setwd()` commands.

How to create an RStudio Project?

I strongly suggest that you set up a project which is dedicated to this course.

1. In RStudio, click File >> New Project... >> New Project...
2. A windows pops up which lets you select between “New Directory”, “Existing Directory”, and “Version Control”. The first option creates a new folder which is named after your project, the second one “associates a project with an existing working directory”, and the third one only applies to version control (like, for instance, GitHub) users. I suggest that you click “New Directory”.
3. Now you need to specify the type of the project (Empty project, R package, or Shiny Web Application). In our case, you will need an “Empty project”. Hit it!
4. The final step is to choose the folder the project will live in. If you have already created a folder which is related to this course, choose this one and let the project live in there as a sub-directory.

²This becomes especially painful if you teach R to your students and have to grade 18 submissions and, hence, have to paste your personal directory’s file path into each of these submissions.

5. When you write code for our course in the future, you *first* open the R project and then create either a new script or open a former one (e.g., by going through the “Files” tab in the respective pane which will show the right directory already.)

Version control: GitHub

If you are a more advanced user of Microsoft Word, you may know the “Track Changes” feature. You can, for instance, activate it if you do somebody a favor and proof-read their thesis draft. “Track Changes” then keeps track of the changes you make – as you might have guessed already.

This exists for coding as well. The times when you named your files “regression_analysis_firsttry.R”, “regression_analysis_secondtry.R”, not to forget to mention “regression_analysis_final_version.R” and “regression_analysis_the_real_final_version.R” are over as soon as you put GitHub into your toolbox – and trust me, you will not miss them.

GitHub also goes well together with RStudio Projects. However, it is fairly complicated to get one’s head around it in the very beginning. There is an excellent tutorial by – again – Jennifer Bryan, who has also written an article on this very subject (Bryan 2017), which you should definitely have a look at. I just recently (i.e., three months ago) really took the time (to be specific, one afternoon and half the subsequent evening) to set up my GitHub account properly and copy my current projects in there. I am convinced that it will definitely make many things a lot easier if you integrate it in your working process from the very beginning of your coding career. The tutorial, named “Happy Git and GitHub for the useR”, can be found [here](#).

Further links

- Hadley Wickham and Garrett Golemund wrote an entire chapter in R4DS on Projects.
- Need more motivation? Jennifer Bryan tells you under which circumstances she would set your machine on fire: Project-oriented workflow.
- If you have created your project folder and are now unsure how to structure, read Chris von Csefalvay blog post on how to do it.

Before you start: R scripts and RMarkdown

In this course, you will work with two sorts of documents store your code in: R scripts (suffix: .R) and RMarkdown documents (suffix: .Rmd). In the following, I will briefly introduce you to them.

R scripts

The console, where you can only execute your code, is great for experimenting with R. If you want to store it – e.g., for sharing – you need something different. This is where R scripts come in handy. When you are in RStudio, you create a new script by either clicking File >> New File >> R Script or ctrl/cmd+shift+n. There are multiple ways to run code in the script:

- * cmd/ctrl+return (Mac/Windows) – execute entire expression and jump to next line
- * option/alt+return (Mac/Windows) – execute entire expression and remain in line
- * cmd/ctrl+shift+return (Mac/Windows) – execute entire script from the beginning to the end (rule: every script you hand in or send to somebody else should run smoothly from the beginning to the end)

If you want to make annotations to your code (which you should do because it makes everything easier to read and understand), just insert ‘#’ into your code. Every expression that stands to the right of the ‘#’ sign will not be executed when you run the code.

RMarkdown

A time will come where you will not just do analyses for yourself in R, but you also will have to communicate them. Let's take a bachelor's or master's thesis as an example: you need a type of document that is able to encapsulate: text (properly formatted), visualizations (tables, graphs, maybe images), and references. An RMarkdown document can do it all, plus, your entire analysis can live in there as well. So there is no need anymore for the cumbersome process of copying data from MS Excel or IBM SPSS into an MS Word table, you just tell Markdown what it should communicate and what not. (Note that all the project work for this course has to be handed in using R Markdown files.) In the following, I will not provide you with an exhaustive introduction to RMarkdown. Instead, I will focus on getting you started and then referring you to better, more exhaustive resources. It is not that I am too lazy to write a big tutorial, but there are state-of-the-art tutorials and resources (which mainly come straight from people who work on the forefront of the development of these tools) which are available for free. By linking to them, I want to encourage you to get involved and dig into this stuff. So, let's get you started!

You create an RMarkdown file by clicking File >> New File >> R Markdown. . . . Then, a window pops up that looks like this:

New R Markdown

Document

Presentation

Shiny

From Template

Title: Untitled

Author: Felix Lennert

Default Output Format:

☒ **HTML**
Recommended format for authoring (you can switch to PDF or Word output anytime).

☐ **PDF**
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

☐ **Word**
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

OK **Cancel**

Figure 4: Fig. 4: new RMarkdown

Note that you could also do a presentation (with the `beamer` package), a shiny app, or use templates. We will focus on simple RMarkdown documents³. Here, you can type in a title, the name(s) of the author(s),

³but feel free to approach me if you want to get some additional resources on the other types.

and choose the default output format. For now you have to choose one, but later you can switch to one of the others whenever you want to.

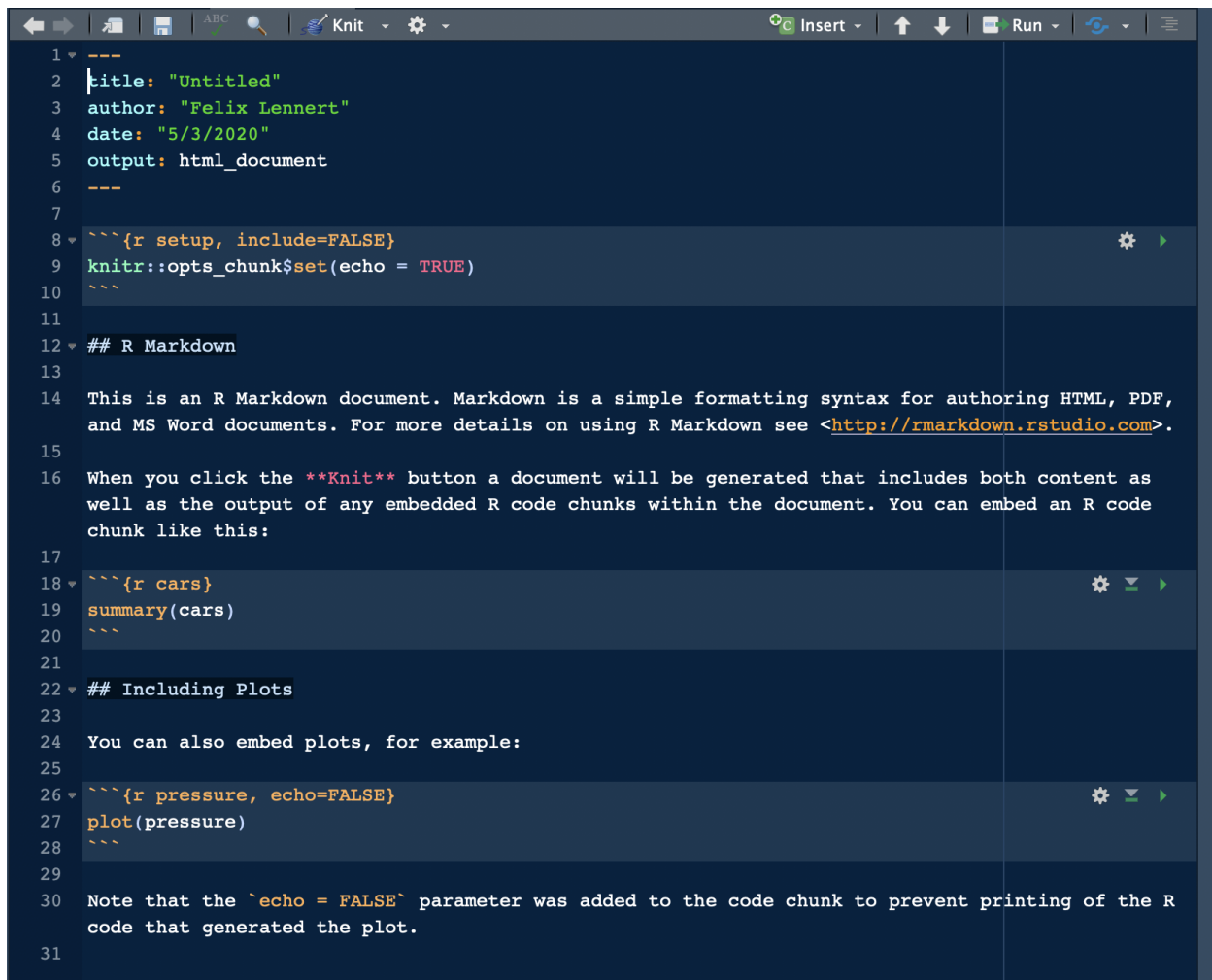
- * *HTML* is handy for lightweight, quickly knitted files, or if you – like me – want to publish it on a website.

- * *PDF* is good if you are experienced with LATEX and want to further modify it in terms of formatting etc., or imply want to get a more formally looking document (I use it if I need to hand in something that is supposed to be graded). If you want to knit to PDF, you need a running LATEX version on your machine. If you do not have one, I recommend you to install *tinytex*. I linked installation instructions down below.

- * Word puts out an MS Word document – especially handy if you collaborate with people who are either not experienced in R, like older faculty, or want some parts to be proof-read (remember the Track-Changes function?). Note that you need to have MS Word installed on your machine.

Did you notice the term “knit”? The logic behind RMarkdown documents is that you edit them in RStudio and then “knit” them. This means that it calls the `knitr` package. Thereby, all the code you include into the document is executed from scratch. If the code does not work and throws an error, the document will not knit – hence, it needs to be properly written to avoid head-scratching. The `knitr` package creates a markdown file (suffix: `.md`). This is then processed by `pandoc`, a universal document converter. The big advantage of this two-step approach is that it enables a wide range of output formats.

For your first RMarkdown document, choose HTML and click “OK”. Then, you see a new plain-text file which looks like this:



```
1 ---
2 title: "Untitled"
3 author: "Felix Lennert"
4 date: "5/3/2020"
5 output: html_document
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF,
15 and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
16
17 When you click the **Knit** button a document will be generated that includes both content as
18 well as the output of any embedded R code chunks within the document. You can embed an R code
19 chunk like this:
20
21 ```{r cars}
22 summary(cars)
23 ```
24
25 ## Including Plots
26
27 You can also embed plots, for example:
28
29 ```{r pressure, echo=FALSE}
30 plot(pressure)
31 ```
32
33 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R
34 code that generated the plot.
```

Figure 5: Fig. 5: fresh and clean RMarkdown document

In the top section, surrounded by ---, you can see the so-called YAML header (or YAML metadata, or YAML frontmatter – check out Wikipedia for more information on it). YAML stands for “YAML Ain’t Markup Language” and it is a human-readable data-serialization language. Quick heads-up: indentation matters in your YAML header. However, this is the metadata of your document. In this minimalistic example, the title, the author, the date, and the desired output are specified (as you specified it in Figure 4). Hence, you can always change them.

After the YAML header, there is a code chunk. Code chunks start with ````{r}` and end with `````. Inside the code chunk, you can write R code which can be executed by either clicking the green “Play” button or by using the same keyboard shortcuts as in scripts. There are several chunk options available: either click on the sprocket or check them out online and include them in the chunk’s header (like this: ````{r include=FALSE}`). Beyond that, you can (and should) name your chunks. This makes it easier to find the flawed ones when your document fails to knit. This is done by simply including the name into the title like this: ````{r cars}`. Find more on chunk options here.

The double hashtags imply that “R Markdown” is a header. In the text, there are examples on how to include links (“<>”), how to make text bold (double asterixes), etc. For more information on how to format plain text in an RMarkdown document, check out the RMarkdown cheatsheet and Reference guide.

Further links

- Hadley Wickham and Garrett Grolmund wrote an entire chapter in R4DS on scripts – and on RMarkdown (<https://r4ds.had.co.nz/r-markdown.html>).
- Yihui Xie, J. J. Allaire, and Garrett Grolmund wrote what they call “R Markdown: The Definitive Guide”. It is freely available online.
- An introduction to RMarkdown from RStudio can be found here.
- Yihui Xie published a manual for installing the tinytex package.
- If you want to write academic papers using RMarkdown, you need to be able to include references properly. When I decided to do so, i found it sort of complicated. However, I stuck to this tutorial and since then it has worked like a charm.

Assignment #1 (to be handed in on 2020-06-14 at latest)

1. Install R and RStudio.
2. Set RStudio up as you wish to. Choose a nice pane layout and a dope theme.
3. Create a folder which is dedicated to this course. Set up your folder with folders for literature (yeah, I am sorry, but you will have to read some papers for this course in the future), slides, and one Project folder. Think deliberately about getting into GitHub (if your answer is yes, you should set up the GitHub repo before the project).
4. Explore R scripts. Run the command `install.packages("tidyverse")`.
5. Set up RMarkdown. Write a short text on your expectations for this course. It should at least contain: a header, an image, an unordered list. Knit it to HTML and PDF. Bonus points for including references to papers that you want to talk about in the theoretical sessions.
—to be continued—

References

- Bryan, Jennifer. 2017. *Excuse Me, Do You Have a Moment to Talk About Version Control? Preprint*. PeerJ Preprints.
- Cotton, Richard. 2013. *Learning R*. First Edition. Beijing ; Sebastopol, CA: O'Reilly.
- Wickham, Hadley and Garrett Grolemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. First edition. Sebastopol, CA: O'Reilly.
- Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton: Taylor & Francis, CRC Press.