How to Use R for Data Science

Lecture Notes

© Prof. Dr. Stephan Huber (Stephan.Huber@hs-fresenius.de)

Last compiled on 04 May, 2023

Contents

Preface

About the notes

- This script aims to support my lecture at the HS Fresenius. It is incomplete and no substitute for taking actively part in class.
- The old version of the notes can be found here PDF.
- I appreciate you reading it, and I appreciate any comments.
- This is work in progress so please check for updates regularly.
- The lecture notes are available online or you can download it as a pdf file here
- These notes are published under a Creative Commons BY-SA license (CC BY-SA) version 4.0. This means it can be reused, remixed, retained, revised and redistributed as long as appropriate credit is given to the authors. If you remix, or modify the original version of this open textbook, you must redistribute all versions of this open textbook under the same license—CC BY-SA. This script is build on the work of Navarro (2020), Muschelli and Jaffe (2022), Thulin (2021), and Ismay and Kim (2022) which is also published under CC BY-SA.

About the author

Figure 1: Prof. Dr. Stephan Huber[1]

Prof. Dr. Stephan Huber is Professor of International Economics and Data Science at *HS Fresenius* and holds a Diploma in Economics from the *University of Regensburg*

[1] Picture is taken from https://sites.google.com/view/stephanhuber

and a Doctoral Degree (summa cum laude) from the University of Trier. He completed postgraduate studies at the *Interdisciplinary Graduate Center of Excellence at the Institute for Labor Law and Industrial Relations in the European Union (IAAEU)* in Trier. He was a research assistant to Prof. Dr. Dr. h.c. Joachim Möller at the *University of Regensburg*, post-doc at the *Leibniz Institute for East and Southeast European Studies (IOS)* in Regensburg and freelancer at *Charles University* in Prague.

He has worked as a lecturer at various institutions including the *TU Munich*, the *University of Regensburg*, *Saarland University*, and the *Universities of Applied Sciences in Frankfurt and Augsburg*. He has also taught abroad for the *University of Cordoba* in Spain and the *University of Perugia*. Professor Huber has published his work in international journals such as the *Canadian Journal of Economics* and the *Stata Journal*. More on his work can be found on his private homepage www.t1p.de/stephanhuber.

Contact

Hochschule Fresenius für Wirtschaft & Medien GmbH
Im MediaPark 4c
50670 Cologne

Office: 4b OG-1 Bü01 (Office hour: Thursday 1-2 p.m.)
Telefon: +49 221 973199-523
Mail: stephan.huber@hs-fresenius.de
Private homepage: www.t1p.de/stephanhuber
Github: https://github.com/hubchev

Chapter 1

Getting Started with R

Before we can start exploring data in R, there are some key concepts to understand first:

1. Why R?
2. How to learn R?
3. What are R and RStudio?
4. How to use R and RStudio without installation
5. How to install R and RStudio
6. How to write and run code in R
7. What are R packages?

1.1 Why R?

R is a free and open-source programming language that provides a wide range of advanced statistics capabilities, state-of-the-art graphics, and powerful data manipulation capabilities. It supports larger data sets, reads any type of data, and runs on multiple platforms. R makes it easier to automate tasks, organize projects, ensure reproducibility, and find and fix errors, and anyone can contribute packages to improve its functionality. Moreover, the following points are worth to emphasize:

- **R is an artist!** Check out:
  - https://www.r-graph-gallery.com/
  - http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html
  - https://www.r-bloggers.com/2020/05/7-useful-interactive-charts-in-r/
- **R is an employment insurance.** If you are good in R programming or if you are good in writing programming code in general, you have plenty of opportunities to

earn a decent salary.

- **R uses the computer and computers are great!** Doing statistics on a computer is faster, easier and more powerful than doing it by hand. Computers excel at mindless repetitive tasks. For most people, the only reason to ever do statistical calculations with pencil and paper is for learning purposes.
- **Excel is bad!** Doing statistics in a spreadsheet (e.g., Microsoft Excel) is often a bad idea. Although many people are likely feel more familiar with them, spreadsheets are very limited in terms of what analyses they allow you to do. You can easily lose the overview and it is hard to keep track of what you have done and in comparison with command line driven programs. In particular, the ability to make your analysis *replicable* is limited.
- **R is good, proprietary software is bad!** Avoiding proprietary software is a very good idea because it costly, support is exclusively provided by the owner of the software (if they stop supporting your version you are lost), security issues cannot be checked as the source code is not available, and possibilities for customization are limited.
- **R is big!** Something that you might not appreciate now, but will love later on if you do anything involving data analysis, is the fact that R is highly extensible. When you download and install R, you get all the basic packages, and those are very powerful on their own. However, because R is so open and so widely used, it's become something of a standard tool in statistics, and so lots of people write their own packages that extend the system. And these are freely available too. One of the consequences of this, I've noticed, is that if you open up an advanced textbook (a recent one, that is) rather than introductory textbooks, is that a *lot* of them use R. In other words, if you learn how to do your basic statistics in R, then you're a lot closer to being able to use the state of the art methods than you would be if you'd started out with a "simpler" system: so if you want to become a genuine expert in data analysis, learning R is a very good use of your time.
- **R is the future!** Programming is a core skill in research, economics, and business. R is one of the most widely used programming languages in the world today. It is used in almost every industry such as finance, banking, medicine or manufacturing. R is used for portfolio management, risk analytics in finance and banking industries.

1.2 How to learn R

There are many different approaches to learning R. It pretty much depends on your preferences, needs, goals, prerequisites and limitations. It is up to you to search and find a suitable way to achieve the learning goals. However, I offer these notes and if you are in one of my classes, you can ask at any time for help.

The notes should walk you through many of the things that are important when working in R, and it should help you dig deeper and learn more if you want to. For beginners, I recommend starting with my swirl courses, see section 2. However, there are thousands of other resources for learning R: textbooks, online courses, videos, guided tutorials, etc.. I give some recommendations about learning resources in section 1.3.

Below, I'll give you a list of resources that are worth a look. You might find what you're looking for there. If not, just keep reading this book. Above all, those who have personally taken one of my courses are welcome to contact me if they think I can help them.

Warning: R is not without its weaknesses: It's not easy to learn, it has some very annoying quirks that we all have to deal with, it's slower than other languages (Phyton, MATLAB), and R's algorithms and sources are spread across many packages (since there's no big company behind it that wants you to buy it). This sometimes makes it very difficult for beginners to find what they are looking for. In simple words: you can get lost!

Tips on learning to code: Learning to code/program is quite similar to learning a foreign language. It can be daunting and frustrating at first. Such frustrations are common and it is normal to feel discouraged as you learn. However, just as with learning a foreign language, if you put in the effort and are not afraid to make mistakes, anybody can learn and improve.

Here are a few useful tips to keep in mind as you learn to program:

- **Remember that computers are not actually that smart**: You may think your computer or smartphone is "smart," but really people spent a lot of time and energy designing them to appear "smart." In reality, you have to tell a computer everything it needs to do. Furthermore, the instructions you give your computer can't have any mistakes in them, nor can they be ambiguous in any way.
- **Take the "copy, paste, and tweak" approach**: Especially when you learn your first programming language or you need to understand particularly complicated code, it is often much easier to take existing code that you know works and modify it to suit your ends. This is as opposed to trying to type out the code from scratch. We call this the *"copy, paste, and tweak"* approach. So early on, we suggest not trying to write code from memory, but rather take existing examples we have provided you, then copy, paste, and tweak them to suit your goals. After you start feeling more confident, you can slowly move away from this approach and write code from scratch. Think of the "copy, paste, and tweak" approach as training wheels for a child learning to ride a bike. After getting comfortable, they won't need them anymore.
- **Have a purpose when coding**: Rather than learning to code for its own sake, it

goes much smoother when you have a goal in mind or when you are working on a particular project, like analyzing data that you are interested in, replicating a paper to understand it, or do something with R that is important to you.

- **Practice is key**: Just as the only method to improve your foreign language skills is through lots of practice and speaking, the only method to improving your coding skills is through lots of practice. Don't worry, however, we'll give you plenty of opportunities to do so!

1.3 Learning resources

Thousand of freely available books and resources exist. On (https://bookdown.org/) and in the Big Book of R is a big collection of links to R books that verifies my claim. Another nice collection of learning ressources can be found here: AWESOME R Learning-Resources

In Rstudio you find in the left panel at the bottom a panel that is called *Help*. There you find a lot of links, manuals, and references that offer you tons of resources to learn R for free including: (https://education.rstudio.com/) and (https://support.rstudio.com/hc/en-us/articles/200552336-Getting-Help-with-R)

Since you may feel overwhelmed by the number of resources, I would like to highlight four books:

1. Timbers, Campbell, and Lee (2022): **Data Science: A First Introduction** is a free and up to date book that comes with exercises with worksheets that are available on UBC-DSCI GitHub repository
2. Wickham and Grolemund (2023): **R for Data Science: Import, Tidy, Transform, Visualize, and Model Data** is the most popular source to learn R. It focuses on introducing the tidyverse package and is freely available online.
3. Irizarry (2022): **Introduction to Data Science: Data Analysis and Prediction Algorithms With R** is a complete, up to date, and applied introduction.
4. Venables, Smith, and R Core Team (2022) **An Introduction to R: Notes on R: A Programming Environment for Data Analysis and Graphics** is a manual from the R Core Development Team that shows how to use R without having to install and load additional packages.

Some other sources that are worth mentioning are these:

- The search engine www.rseek.org is R specific and often better than www.google.com as it only searches for content that has to do with the programming language R.
- On www.rdocumentation.org you can find the complete documentation of all R packages.
- Many find these cheatsheets helpful.

1.4 What are R and RStudio?

Throughout this book, we will assume that you are using R via RStudio. First time users often confuse the two. At its simplest, R is like a car's engine while RStudio is like a car's dashboard as illustrated in Figure 1.1.

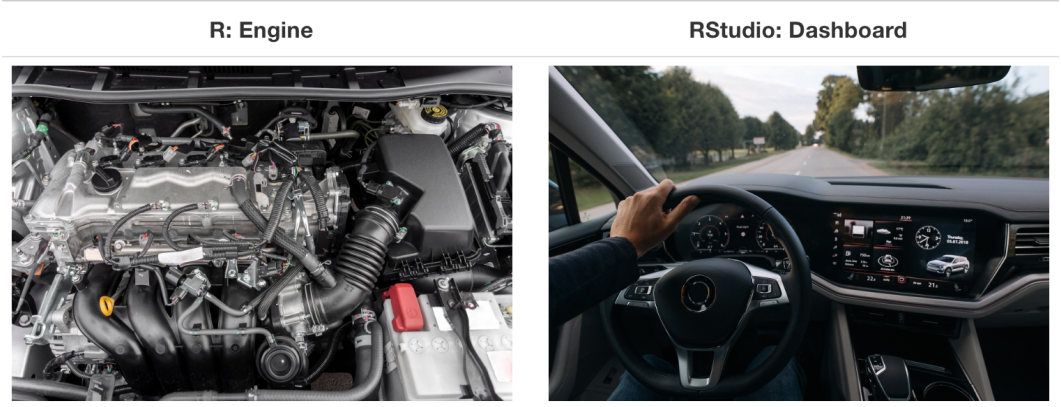

Figure 1.1: Analogy of difference between R and RStudio.

More precisely, R is a programming language that runs computations, while RStudio is an *integrated development environment (IDE)* that provides an interface by adding many convenient features and tools. So just as the way of having access to a speedometer, rearview mirrors, and a navigation system makes driving much easier, using RStudio's interface makes using R much easier as well.

Much as we don't drive a car by interacting directly with the engine but rather by interacting with elements on the car's dashboard, we won't be using R directly but rather we will use RStudio's interface. After you install R and RStudio on your computer, you'll have two new *programs* (also called *applications*) you can open. We'll always work in RStudio and not in the R application. Figure 1.2 shows what icon you should be clicking on your computer.

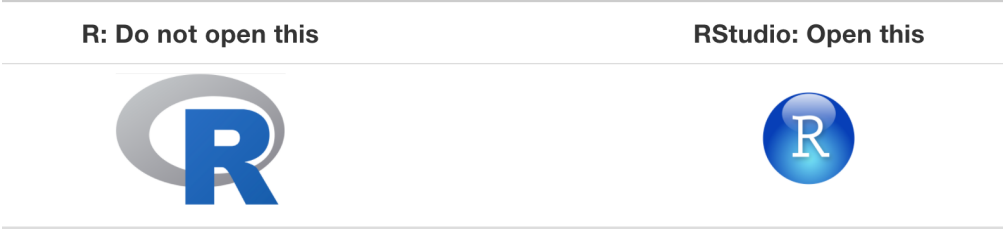

Figure 1.2: Icons of R versus RStudio on your computer.

After you open RStudio, you should see something similar to Figure 1.3 where three or four panels dividing the screen.

1. The *Environment* panel, where a list of the data you have imported and created can be found.

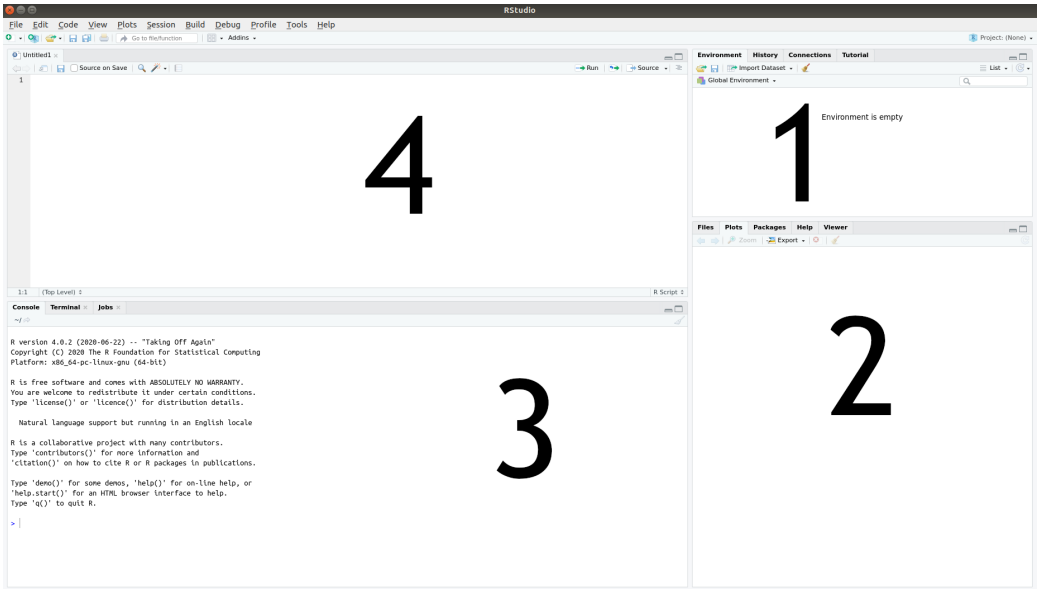

Figure 1.3: RStudio interface to R.

2. The *Files*, *Plots* and *Help* panel, where you can see a list of available files, will be able to view graphs that you produce, and can find help documents for different parts of R.
3. The *Console* panel, used for running code. This is where we'll start with the first few examples.
4. The *Script* panel, used for writing code. This is where you'll spend most of your time working.

The *Console* panel will contain R's startup message, which shows information about which version of R you're running. My startup message at the time of writing was as follows:

> R version 4.1.2 (2021-11-01) – "Bird Hippie" Copyright (C) 2021 The R Foundation for Statistical Computing Platform: x86_64-pc-linux-gnu (64-bit)
>
> R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details.
>
> Natural language support but running in an English locale
>
> R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

> Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R.

If you don't have panel number 4, open it by opening an existing R-script or creating a new one. You can create a new on by clicking *Ctrl+Shift+N* (alternatively, you can use the menu: File→New File→R Script).

You can resize the panels as you like, either by clicking and dragging their borders or using the minimise/maximise buttons in the upper right corner of each panel. Clicking *Ctrl++* and *Ctrl+-* allows to make the fonts larger or smaller.

When you exit RStudio, you will be asked if you wish to *save your workspace*, meaning that the data that you've worked with will be stored so that it is available the next time you run R. That might sound like a good idea, but in general, I recommend that you don't save your workspace, as that often turns out to cause problems down the line. It is almost invariably a much better idea to simply rerun the code you worked with in your next R session.

1.5 How to use R and RStudio without installation

If you don't want to install R on your PC or you don't have admin rights to do so, you can use RStudio online doing *cloud computing* on https://posit.cloud/. Posit Cloud (formerly RStudio Cloud) is a cloud-based solution that allows anyone to do, share, teach and learn data science online. It is free for individuals with some restrictions and limited capacities.

1.6 Installing R and RStudio

You will first need to download and install both R and RStudio (Desktop version) on your computer. It is important that you install R first and then install RStudio.

1. **You must do this first:** Download and install R by going to https://cloud.r-project.org/.

- If you are a Windows user: Click on "Download R for Windows", then click on "base", then click on the Download link.
- If you are macOS user: Click on "Download R for (Mac) OS X", then under "Latest release:" click on R-X.X.X.pkg, where R-X.X.X is the version number. For example, the latest version of R as of November 25, 2019 was R-3.6.1.
- If you are a Linux user: Click on "Download R for Linux" and choose your distribution for more information on installing R for your setup.

1. **You must do this second:** Download and install RStudio at https://www.rstudio.com/products/rstudio/download/.

- Scroll down to "Installers for Supported Platforms" near the bottom of the page.
- Click on the download link corresponding to your computer's operating system.

1.7 What are R packages?

A package is basically just a big collection of functions, data sets and other R objects that are all grouped together under a common name. Some packages are already installed when you put R on your computer, but the vast majority of them of R packages are out there on the internet, waiting for you to download, install and use them. R packages are collections of functions and data sets developed by the community. They increase the power of R by improving existing base R functionalities, or by adding new ones. For example, if you are usually working with data frames, probably you will have heard about *dplyr* or *data.table*, two of the most popular R packages. More than 10,000 packages are available at the official repository (CRAN) and many more are publicly available through the internet.

In this section, I'll describe how to work with packages using the Rstudio tools. Along the way, you'll see that whenever you get Rstudio to do something (e.g., install a package), you'll actually see the R commands that get created.

However, before we get started, there's a critical distinction that you need to understand, which is the difference between having a package **installed** on your computer, and having a package **loaded** in R. When you install R on your computer only a small number of packages come bundled with the basic R installation. The installed packages are on your computer. The critical thing to remember is that just because something is on your computer doesn't mean R can use it. In order for R to be able to *use* one of your installed packages, that package must also be *loaded*. Generally, when you open up R, only a few of these packages (about 7 or 8) are actually loaded. Basically what it boils down to is this:

1. A package must be installed before it can be loaded.
2. A package must be loaded before it can be used.

We only need to install a package once on our computer. However, to use the package, we need to load the library every time we start a new R environment. You can think of this as installing a bulb versus turning on the light.

The two step process might seem a little odd at first, but the designers of R had very good reasons to do it this way. That is, there are more than 10.000 packages, and probably about 8000 authors of packages, and no-one really knows what all of them do. Keeping

Figure 1.4: Installing packages

the installation separate from the loading minimizes the chances that two packages will interact with each other in a nasty way. Moreover having installed all available packages would probably blow your hard disk.

Another good analogy for R packages is they are like apps you can download onto a mobile phone:

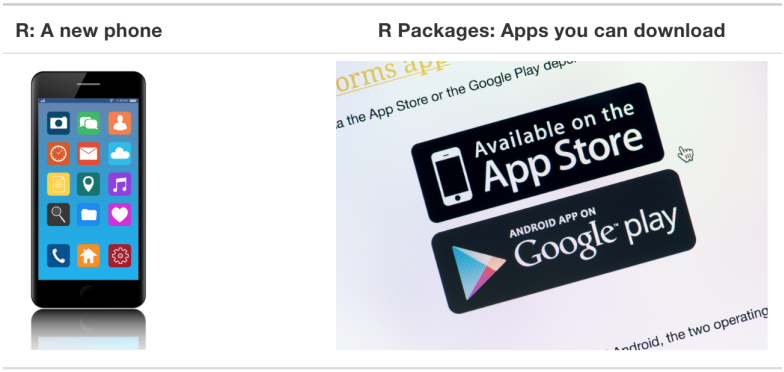

Figure 1.5: Analogy of R versus R packages.

So R is like a new mobile phone: while it has a certain amount of features when you use it for the first time, it doesn't have everything. R packages are like the apps you can download onto your phone from Apple's App Store or Android's Google Play.

1.7.1 Package installation

There are two ways to install an R package: an easy way and a very easy way. Let's install the `ggplot2` package the easy way first as shown in Figure 1.6. In the Files pane of RStudio:

a) Click on the "Packages" tab.
b) Click on "Install" next to Update.

c) Type the name of the package under "Packages (separate multiple with space or comma):" In this case, type `ggplot2`.
d) Click "Install."

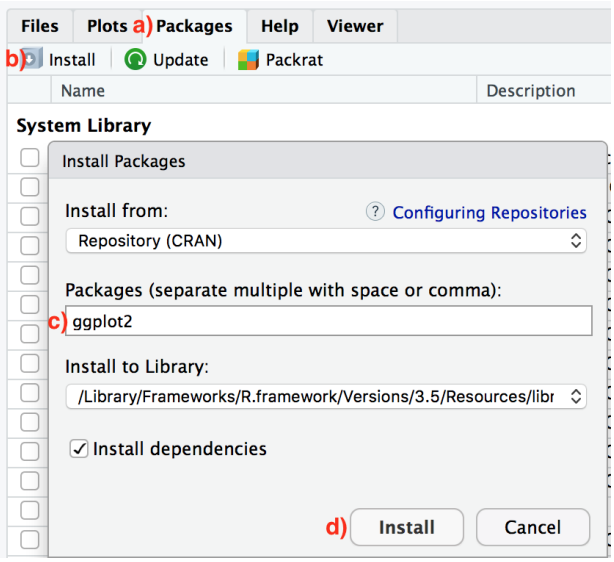

Figure 1.6: Installing packages in R the easy way.

An alternative way to install a package is by typing

```
install.packages("ggplot2")
```

in the console pane of RStudio and pressing Return/Enter on your keyboard. Note you must include the quotation marks around the name of the package.

Much like an app on your phone, you only have to install a package once. However, if you want to update a previously installed package to a newer version, you need to reinstall it by repeating the earlier steps.

1.7.2 Package loading

Recall that after you've installed a package, you need to *load it*. In other words, you need to *open it*. We do this by using the `library()` command.

For example, to load the `ggplot2` package, run the following code in the console pane. What do we mean by "run the following code"? Either type or copy-and-paste the following code into the console pane and then hit the Enter key.

```
library("ggplot2")
```

If after running the earlier code, a blinking cursor returns next to the > "prompt" sign, it means you were successful and the `ggplot2` package is now loaded and ready to use. If, however, you get a red "error message" that reads

```
Error in library(ggplot2) : there is no package called 'ggplot2'
```

It means that you didn't successfully install it. If you get this error message, go back to section 1.7.1 on R package installation and make sure to install the `ggplot2` package before proceeding.

One very common mistake new R users make when wanting to use particular packages is they forget to *load* them first by using the `library()` command we just saw. Remember: *you have to load each package you want to use every time you start RStudio.* If you don't first *load* a package, but attempt to use one of its features, you'll see an error message similar to:

```
Error: could not find function
```

This is a different error message than the one you just saw on a package not having been installed yet. R is telling you that you are trying to use a function in a package that has not yet been *loaded*. R doesn't know where to find the function you are using. Almost all new users forget to do this when starting out, and it is a little annoying to get used to doing it. However, you'll remember with practice and after some time it will become second nature for you.

1.8 Key guidelines on working with R

To avoid running into issues with R, it's important to be aware of the some conventions, rules, and best practices that apply to the language. While it can be tedious to go through all of the do's and don'ts in detail, following them can make your life with R much easier. Trust me, the benefits of adhering to these guidelines will become clear over time. Here is a non-exhaustive list:

1. Do remember that R programming language is case sensitive.
2. Do start names of objects such as vectors, numbers, variables, and data frames with a letter, not a number.
3. Do avoid using dots in names of objects.
4. Do avoid using certain keywords in naming objects, such as if, else, repeat, while, function, for, in, next, break, TRUE, FALSE, NULL, Inf, NaN, and NA.
5. Do use front slash / instead of backslash \ for navigating the file system.
6. Do not use whitespace and indentation for naming files, directories, or objects.

7. Do not ignore warnings or errors unless you know what they mean.
8. Do define objects to represent hard-coded values instead of using them directly in code.
9. Do remember to (install and) load packages that contain functions you want to use.
10. Do remember to set your working directory. (Tip: Use R Studio projects, see 7.1.4)
11. Do remember to comment your code.
12. Do use <- instead of = for assignment.

An exhaustive style guide on how to write code can be found here: https://style.tidyverse.org

Chapter 2

Learn interactively with swirl

The R package *swirl* makes it fun and easy to learn R programming and data science. *swirl* teaches you R programming and data science interactively, at your own pace, and right in the R console! You get immediate feedback on your progress. If you are new to R, have no fear. *swirl* will walk you through each of the steps required to employ Rstudio and R for your purpose. To start it, please follow my instructions precisely!

Open Rstudio and type in the console the following:

```
install.packages("swirl")
library("swirl")
ls()
rm(list=ls())
install_course_github("hubchev", "swirl-it")
swirl()
```

The above lines of code do the following:

- Install the *swirl* package.
- Load the *swirl* package.
- List the content of the environment.
- Remove everything from the environment.
- Install my *swirl* course that is hosted on GitHub.
- With *swirl* you start *swirl* and your learning experience.

If the course has failed to install, you can try to download the file `swirl-it.swc` from https://github.com/hubchev/swirl-it and install the course with loading the swirl package and typing `install_course()` into the console.

Please choose the course *swirl-it* and the learning module *huber-intro-1*. You can exit

swirl at any time by typing `bye()` or by clicking the *Esc* on your keyboard.

After you have successfully finished learning module *huber-intro-1* please go ahead with the learning module *huber-intro-2* that is also part of my swirl course *swirl-it*.

swirl modules on data analytical basics

In my swirl modules *huber-data-1*, *huber-data-2*, and *huber-data-3* I introduce some very basic statistical principles on how to analyse data.

swirl module on the `tidyverse` package

I compiled a short *swirl* module to introduce the *tidyverse* universe. This is a powerful collection of packages which I discuss later on. The learning module is also part of my *swirl-it* course.

Other *swirl* modules

You can also install some other courses. You find a list of courses here http://swirlstats.com/scn/index.html or here https://github.com/swirldev/swirl_courses.

I recommend this one as it gives a general overview on very basic principles of R:

```
library(swirl)
install_course_github("swirldev", "R_Programming_E")
swirl()
```

Chapter 3

Work with R scripts

3.1 R Scripts: Why they are useful

Typing functions into the console to run code may seem simple, but this interactive style has limitations:

- Typing commands one at a time can be cumbersome and time-consuming.
- It's hard to save your work effectively.
- Going back to the beginning when you make a mistake is annoying.
- You can't leave notes for yourself.
- Reusing and adapting analyses can be difficult.
- It's hard to do anything except the basics.
- Sharing your work with others can be challenging.

That's where having a transcript of all the code, which can be re-run and edited at any time, becomes useful. An R script is precisely that - a plain text file that contains code and comments and this comes with advantages:

- Scripts provide a record of everything you did during your data analysis.
- You can easily edit and re-run code in a script.
- Scripts allow you to leave notes for yourself.
- Scripts make it easy to reuse and adapt analyses.
- Scripts allow you to do more complex analyses.
- Scripts make it easy to share your work with others.

3.2 Generate, write and run R scripts

To **generate a script** you can

1. Go to the *File* menu, select *New File* and then choose *R Script* or
2. Use the keyboard shortcut *Ctrl+Shift+N* (Windows) or *Cmd+Shift+N* (Mac) or
3. Type the following command in the Console:

```
file.create("hello.R")
```

In the first two ways, a new R script window will open which can be edited and should be saved either by clicking on the *File* menu and selecting *Save*, clicking the disk icon, or by using the shortcut *Ctrl+S* (Windows) or *Cmd+S* (Mac). If you go for the third way, you need to open it manually.

Regardless of your preferred way of generating a script, we can now start **writing** our first script:

```
setwd("/home/sthu/Dropbox/hsf/23-ss/ds/")
x <- "hello world"
print(x)
```

```
## [1] "hello world"
```

Then save the script using the menus (File > Save) as hello.R.

The above lines of code do the following:

- `setwd()` allows to set the working directory. If you are not familiar with file systems, please read section 7.1 in the appendix.
- With the assignment operator <- we create an object that stores the words "hello world" in an object entitled `x`. In the next section 3.3 the assignment operator is further explained.
- With the third input we print the content of the object `x`.

So how do we **run the script**? Assuming that the `hello.R` file has been saved to your working directory, then you can run the script using the following command:

```
source( "hello.R" )
```

Suppose you saved the script in a sub-folder called `scripts` of your working directory, then you need to run the script using the following command:

```
source("./scripts/hello.R")
```

Just note that the dot, `.`, means the current folder. Instead of using the `source` function, you can click on the `source` button in Rstudio.

With the character # you can write a comment in a script and R will simply ignore everything that follows in that line onwards.

3.3 The assignment operator: <-

Suppose I'm trying to calculate how much money I'm going to make from this book. I agree, it is an unrealistic example but it will help you to understand R. Let's assume I'm only going to sell 350 copies. To create a variable called `sales` and assigns a value to it, we need to use the *assignment operator* of R, which is <- as follows:

```
sales <- 350
```

When you hit enter, R doesn't print out any output. If you are using Rstudio, and the *environment panel* you can see that something happened there, can you? It just gives you another command prompt. However, behind the scenes R has created a variable called `sales` and given it a value of `350`. You can check that this has happened by asking R to print the variable on screen. And the simplest way to do that is to type the name of the variable and hit enter.

```
sales
```

```
## [1] 350
```

Worth a mentioning is the curious features of R that there are several different ways of making assignments. In addition to the <- operator, we can also use -> and =. If you want to use ->, you might expect from just looking at the symbol you should write it like this:

```
350 -> sales
```

However, it is common practice to use <- and I recommend only to use this one because it is easier to read in scripts.

3.4 Doing calculation in scripts

R can do any kind of arithmetic calculation with the arithmetic operators given in the table below. Using the assignment operator, R functions, and the features of a R script is easy and gives an idea how R works and how you should embrace the power of the programming language.

operation	operator	example input	example output
addition	+	10+2	12
subtraction	-	9-3	6
multiplication	*	5*5	25
division	/	10/3	3
power	^	5^2	25

So please copy and past the following lines of code into a R script of yours, try to run it on your PC, and try to understand it. Of course, you have to tweak the script a bit to make it run on your PC. For example, I doubt you have the same working directory that I decided to use.

```
# Set working directory
setwd("~/Dropbox/hsf/23-ss/ds")
# Create a vector that contains the sales data
sales_by_month <- c(0, 100, 200, 50, 3, 4, 8, 0, 0, 0, 0, 0)
sales_by_month
sales_by_month[2]
sales_by_month[4]
february_sales <- sales_by_month[2]
february_sales
sales_by_month[5] <- 25 # added May sales data
sales_by_month
# Do I have 12 month?
length( x = sales_by_month )
# Assume each unit costs 7 Euro, then the revenue is
price <- 7
revenue <- sales_by_month*price
revenue
# To get statistics for daily revenue we define the number of days:
days_per_month <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
# Calculate the daily revenue
revenue_per_day <- revenue/days_per_month
revenue_per_day
# round number
round(revenue_per_day)
```

Use the "?" to search for the documentation of all functions used. In particular, do you understand how the function `round()` works? What arguments does the function contain? How can you manipulate the pre-defined arguments. For example, can you calculate the rounded revenue per day with two or four digits? Try it out!

```
?round()
```

3.5 User-defined functions

One of the great strengths of R is the user's ability to add functions. Sometimes there is a small task (or series of tasks) you need done and you find yourself having to repeat it

multiple times. In these types of situations it can be helpful to create your own custom function. The structure of a function is given below:

```
name_of_function <- function(argument1, argument2) {
    statements or code that does something
    return(something)
}
```

First you give your function a name. Then you assign value to it, where the value is the function. When defining the function you will want to provide the list of arguments required (inputs and/or options to modify behavior of the function), and wrapped between curly brackets place the tasks that are being executed on/using those arguments. The argument(s) can be any type of object (like a scalar, a matrix, a dataframe, a vector, a logical, etc), and it's not necessary to define what it is in any way. Finally, you can return the value of the object from the function, meaning pass the value of it into the global environment. The important idea behind functions is that objects that are created within the function are local to the environment of the function – they don't exist outside of the function. Note, a function doesn't require any arguments.

Let's try creating a simple example function. This function will take in a numeric value as input, and return the squared value.

```
square_it <- function(x) {
   square <- x * x
   return(square)
}
```

Now, we can use the function as we would any other function. We type out the name of the function, and inside the parentheses we provide a numeric value `x`:

```
square_it(5)
```

```
## [1] 25
```

Let us get back to script with sales and try to calculate the monthly growth rates of revenue using a self-written function.

The formula of a growth rate is clear:

$$g = \left(\frac{y_t - y_{t-1}}{y_{t-1}}\right) \cdot 100 = \left(\frac{y_t}{y_{t-1}} - 1\right) \cdot 100$$

So the challenge is to divide the value of `revenue` with the value of the previous period, a.k.a. the lagged value. Let us assume that the function `lag()` can give you exactly that

value of a vector. Lets try it out:

```
lag(revenue)
```

```
##  [1]   NA    0  700 1400  350  175   28   56    0    0    0    0
```

```
(revenue/lag(revenue)-1)*100
```

```
##  [1]   NA  Inf  100  -75  -50  -84  100 -100  NaN  NaN  NaN  NaN
```

Unfortunately, this does not work out. The `lag()` function does not work as we think it should. Well, the reason is simply that we are using the wrong function. The current `lag()` function is part of the stats package which is part of R base and is always loaded. The `lag()` function we aim to use stems from the dplyr package which we must install and load to be able to use it. So let's do it:

```
# check if the package is installed
find.package("dplyr")
```

```
## [1] "/usr/local/lib/R/site-library/dplyr"
```

```
# I already installed the package so I can just load it
# install.packages("dplyr")
library("dplyr")
```

Now, we was informed that among other functions the `lag()` function is *masked*. That means that now the function of the newly loaded package is active. So, let's try again:

```
lag(revenue)
```

```
##  [1]   NA    0  700 1400  350  175   28   56    0    0    0    0
```

```
(revenue/lag(revenue)-1)*100
```

```
##  [1]   NA  Inf  100  -75  -50  -84  100 -100  NaN  NaN  NaN  NaN
```

That looks good now. And here is a way to calculate growth rates with a self-written function:

```
growth_rate <- function(x)(x/lag(x)-1)*100
growth_rate(revenue)
```

```
##  [1]   NA  Inf  100  -75  -50  -84  100 -100  NaN  NaN  NaN  NaN
```

```
sales_gr_rate <- growth_rate(revenue)
sales_gr_rate
```

```
##  [1]   NA  Inf  100  -75  -50  -84  100 -100  NaN  NaN  NaN  NaN
```

You know, all functions in R are user-written and sometimes it happens that two people had the idea to name functions identically. In such a case we have to deal with that conflict as we cannot use both functions at the same time. Also, it is a very realistic case that you find a solution somewhere that should work out your problem but you fail to make it run. In such a case it is often a forgotten package that needs to be loaded.

Exercise 3.1. All roads lead to R(ome)

If you ask ten programmers to solve a particular problem, you will probably receive ten different soltuions that are all valid. R is no exception here. This can be very confussing when just started to learn R.

Below you find two more ways to calculate the a growth rate. Do you understand them?

```
c(NA, diff(revenue)/head(revenue, -1))*100
```

```
##  [1]   NA  Inf  100  -75  -50  -84  100 -100  NaN  NaN  NaN  NaN
```

```
(revenue/c(NA,revenue[-length(revenue)])-1)*100
```

```
##  [1]   NA  Inf  100  -75  -50  -84  100 -100  NaN  NaN  NaN  NaN
```

Chapter 4

Visualizing data

Data visualization is an art. The purposes of visualizing data are manifold. You can emphasize facts, get known to data, detect anomalies, and communicate a large amount of infirmation simply and intuitive. Whatever your goal is, thousand of appropiate ways exist to visualize data. Many decisions to take are simply a matter of taste. However, there are some conventions and guidelines that help you to make on average better decisions when designing a visualization:

- Good graphs are easy to understand and eye catching.
- Graphs can be misleading and manipulative and that is opposing to the ideas of science. Thus, be responsible and honest.
- Minimize colors and other attention-grabbing elements that are not directly related to the data of interest. Worldwide, there are approximately 300 million color blind people. In particular, red, green or blue light are problematic to color blind people. Thus, better rely on color schemes that are designed for colorblind people.
- Don't truncate an axis or change the scaling within an axis just to make you your story more appealing. Show the full scale of the graph, then zoom to show the data of interest, if necessary.
- Label and describe your chart sufficiently so that everybody can fully understand the content of the shown data set and statistics without having to study the notes of the graph for too long.
- Don't do pie charts. They may look simple, but they're tricky to get right and there are usually better alternatives. Humans are not very good at comparing the size of angles and as there's no scale in pie plots, reading accurate values is difficult. Figure 4.1 may proof this.
- For more tips, see:
  - Data Visualization: Chart Dos and Don'ts (by Duke University)
- Graphs and Visualising Data by Oliver Kirchkamp. In particular, I highly

recommend his handout (Kirchkamp, 2018). It discusses many pitfalls of visualizing data, instructs how to do good graphs, and he shows the corresponding R code of all graphs.

- The R Graph Gallery offers shows graphs and the corresponding R code to replicate the graphs
- The work of Edward Tufte and his book *The Visual Display of Quantitative Information* (Tufte, 2022) are classical readings.

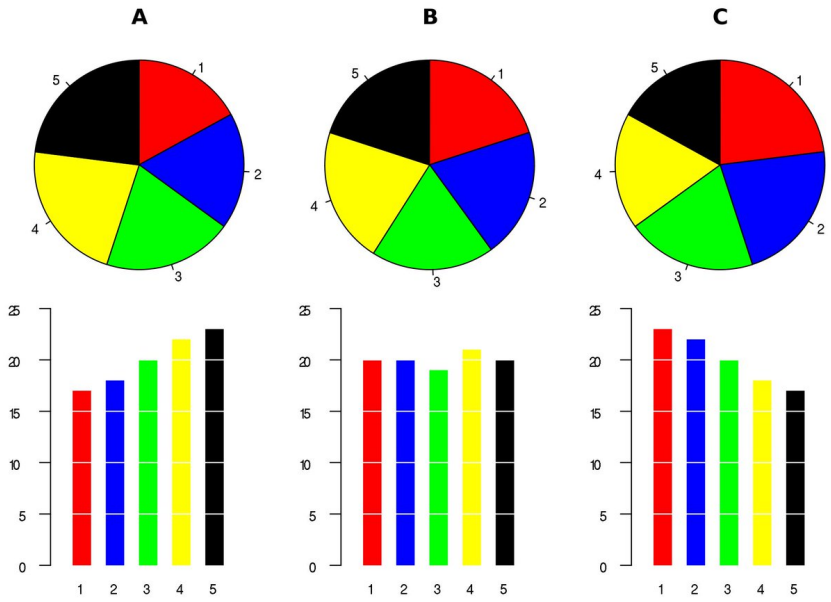

Figure 4.1: Pie charts are problematic[1]

When it comes to data visualization, there's no better resource than the book by Wickham and Grolemund (2023). It introduces the ggplot function which is part of the ggplot2 package which, in turn, is part of the tidyverse package. Thus, if you've installed and loaded tidyverse, you automatically have access to ggplot. Creating beautiful and informative graphs is easy with ggplot. To proof that claim, study chapters 1 and 2 of Wickham and Grolemund (2023). To reap the best benefits from studying, I recommend to copy all the code that is shown in the book into a R script and try to run it on your PC. That is the best way to learn, understand, and create your own notes that may guide you later on. Whenever you see interesting code somewhere, try to run it on your PC. Moreover, I recommend the exercises of the book, they are challenging sometimes but to really understand code you need to run code yourself.

[1] Picture is taken from `https://en.wikipedia.org/wiki/Pie_chart`

Chapter 5

Manage data

5.1 The tidyverse universe

The R package *tidyverse* is a collection of R packages. All packages share an underlying design philosophy, grammar, and data structures. The most popular packages are ggplot2, dplyr, tidyr, readr, purrr, tibble, stringr, and forcats. They provide functionality to model, transform, and visualize data. Tidyverse is extremely popular and many individual packages of tidyverse are regularly in the top 10 most downloaded R packages on CRAN[1]. How to do data science with tidyverse is the subject of multiple books and tutorials. The popular book *R for Data Science* by Wickham and Grolemund (2023) is all about the tidyverse universe. Thus, I highly recommend reading sections 3 (Workflow: basics) and 4 (Data transformation) of Wickham and Grolemund (2023). Addidionally, visit www.tidyverse.org and, if you still haven't done so, do the tidyverse module of my swirl package, called *swirl-it*, see section 2.

To install and load tidyverse run the following lines of code:

```
install.packages("tidyverse")
library("tidyverse")
```

5.2 The pipe operator

The pipe operator, %>%, comes from the magrittr package which is also a part of the tidyverse package. The *base* pipe operator, |>, is since R.4.1.0 part of base R. For most cases these two are identical. The pipe is to help you write code in a way that is easier

[1]CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R, see: https://cran.r-project.org.

to read and understand. As R is a functional language, code often contains a lot of parenthesis, (and). Nesting those parentheses together is complex and you easily get lost. This makes your R code hard to read and understand. Here's where %>% comes in to the rescue! Consider the following chunk of code to explain the usage of the pipe:

```
# create some data `x`
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)
x
```

```
## [1] 0.109 0.359 0.630 0.996 0.515 0.142 0.017 0.829 0.907
```

```
# take the logarithm of `x`,
x2 <- log(x)
x2
```

```
## [1] -2.216407397 -1.024432890 -0.462035460 -0.004008021 -0.663588378 -1.951928
## [9] -0.097612829
```

```
# compute the lagged and iterated differences (see `diff()`)
x3 <- diff(x2)
x3
```

```
## [1]  1.19197451  0.56239743  0.45802744 -0.65958036 -1.28833984 -2.12261371  3
```

```
#  Make yourself familiar with the functions round() and round the result (1 digi
x4 <- round(x3, 1)
x4
```

```
## [1]  1.2  0.6  0.5 -0.7 -1.3 -2.1  3.9  0.1
```

That is rather long and we actually don't need objects x2, x3, and x4. Well, then let us write that in a nested function:

```
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)

round(diff(log(x)), 1)
```

```
## [1]  1.2  0.6  0.5 -0.7 -1.3 -2.1  3.9  0.1
```

This is short but you easily loose overview. The solution is the *pipe*:

```
# load one of these packages: `magrittr` or `tidyverse`
library(tidyverse)

# Perform the same computations on `x` as above
x %>% log() %>%
```

```
    diff() %>%
    round(1)
```

```
## [1]  1.2  0.6  0.5 -0.7 -1.3 -2.1  3.9  0.1
```

You can read the %>% with "and then" because it takes the results of some function "and then" does something with that in the next. Another example can be found in this short clip: Using the pipe operator in R

Read out loud the following code:

```
library("datasets")
iris %>%
  subset(Sepal.Length > 5) %>%
  aggregate( . ~ Species , . , mean)
```

```
##      Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     setosa     5.313636    3.713636     1.509091   0.2772727
## 2 versicolor     5.997872    2.804255     4.317021   1.3468085
## 3  virginica     6.622449    2.983673     5.573469   2.0326531
```

A solution may be the following: "you take the Iris data *and then* you subset the data *and then* you aggregate the data and show the mean".

5.3 Import data

5.3.1 Vectors and Matrices

We already got known to the c() function which allows to combine multiple values into a vector or list. Here are some examples how you can use this function to create vectors and matrices:

```
# defining multiple vectors
v_a <- c(1:3)
v_b <- c(10:12)

# creating matrix
m_ab <- matrix(c(v_a, v_b), ncol = 2)
m_cbind <- cbind(v_a, v_b)
m_rbind <- rbind(v_a, v_b)

# print matrix
print(m_ab)
```

```
##      [,1] [,2]
## [1,]    1   10
## [2,]    2   11
## [3,]    3   12
```

```
print(m_cbind)
```

```
##      v_a v_b
## [1,]   1  10
## [2,]   2  11
## [3,]   3  12
```

```
print(m_rbind)
```

```
##     [,1] [,2] [,3]
## v_a    1    2    3
## v_b   10   11   12
```

```
# defining row names and column names
rown <- c("row_1", "row_2")
coln <- c("col_1", "col_2", "col_3")

# creating matrix
m_ab_label <- matrix(m_ab, ncol = 3, byrow = TRUE,
            dimnames = list(rown, coln))

# print matrix
print(m_ab_label)
```

```
##       col_1 col_2 col_3
## row_1     1     2     3
## row_2    10    11    12
```

The two most common formats to store and work with data in R are dataframe and tibble. Both formats store table-like structures of data in rows and columns. We will learn more on that in section 5.6

```
# convert the matrix into dataframe
df_ab=as.data.frame(m_ab_label)
tbl_ab=as.tibble(m_ab_label)
```

5.4 RData files

You can save some of your objects (`save()`) or all (`save.image()`) and load data stored in the `.RData` (`load()`) format easily. With `rm()` you remove objects from your workspace. and with `rm(list = ls())` you clear all objects from the workspace. When you delete an object in R, you cannot recover it by clicking some *Undo button*.

5.5 Import other file formats

RStudio provides convenient data import tools that can be accessed by clicking *File > Import Dataset*. In addition, tidyverse offers packages for importing data in various formats. This cheatsheet, for example, is about the packages `readr`, `readxl` and `googlesheets4`. The first allows you to read data in various file formats, including fixed-width files like .csv and .tsv. The package `readxl` can read in Excel files, i.e., .xls and .xlsx file formats and `googlesheets4` allows to read and write data from Google Sheets directly from R.

For more information, I recommend once again the second version book *R for Data Science* by Wickham and Grolemund (2023). In particular, check out the "Data tidying" section for importing CSV and TSV files, the "Spreadsheets" section for Excel files, the "Databases" section for retrieving data with SQL, the "Arrow" section for working with large datasets, and the "Web scraping" section for extracting data from web pages.

5.6 Dataframes and tidy data (tibbles)

```
df_ab
```

```
##       col_1 col_2 col_3
## row_1     1     2     3
## row_2    10    11    12
```

```
tbl_ab
```

```
## # A tibble: 2 x 3
##   col_1 col_2 col_3
##   <int> <int> <int>
## 1     1     2     3
## 2    10    11    12
```

```
df_test <- as.data.frame(as.table(m_ab_label))
tbl_test=as.tibble(df_test)
```

```
test <- mtcars |>
    rownames_to_column( var = "car") |>
    as.tibble()
```

Both data frames and tibbles are two of the most commonly used data structures in R for handling tabular data. A tibble actually is a data frame and you can use all functions that work with a data frame also with a tibble. However, a tibble has some additional features. Data frames are provided by base R while tibbles are provided by the tidyverse package. This means that if you want to use tibbles you should load tidyverse.

When printed to the console, both display the first 10 rows but only data frames display all columns. In contrast, a tibble displays only as many columns as fit within the console width. That makes it easier to view and work with large datasets.

Most importantly, tibbles should follow three interrelated rules that make a dataset *tidy*:

1. Each variable is a column; each column is a variable.
2. Each observation is a row; each row is an observation.
3. Each value is a cell; each cell is a single value.

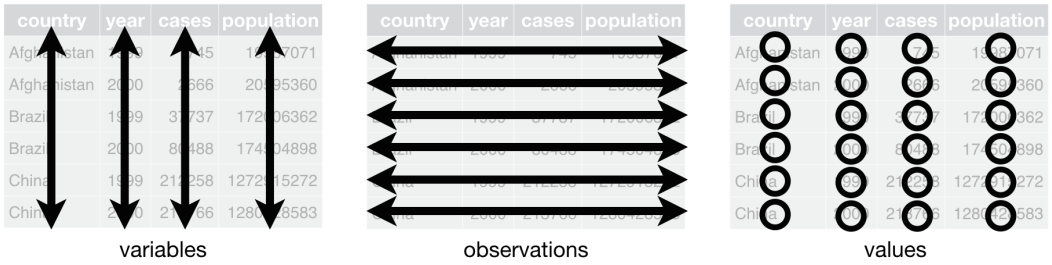

Figure 5.1: Features of a tidy dataset: variables are columns, observations are rows, and values are cells[3]

Figure 5.1 illustrates the characteristics of a tidy dataset, which, as described by Wickham and Grolemund (2023), offers two key benefits:

1. A tidy dataset with a consistent data structure is easier to work with because the underlying uniformity facilitates learning and using data manipulation tools.
2. Representing variables in columns is advantageous because it enables R's vectorized operations to be applied, leveraging the fact that most R functions operate on vectors of values.

[3]The figure stems from Wickham and Grolemund (2023)

5.7 Data manipulation with dplyr

The dplyr package, which is part of tidyverse, make data manipulation easy:

- Pick observations by their values (`filter()`).
- Reorder the rows (`arrange()`).
- Pick variables by their names (`select()`).
- Create new variables with functions of existing variables (`mutate()`).
- Collapse many values down to a single summary (`summarise()`).

These functions can all be used in conjunction with `group_by()` , which changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

All functions work similarly: 1. The first argument is a data frame. 2. The subsequent arguments describe what to do with the data frame. 3. The result is a new data frame.

Exercise 5.1. Subsetting

1. Check to see if you have the mtcars dataset by entering the command mtcars.
2. Save the mtcars dataset in an object named cars.
3. What class is cars?
4. How many observations (rows) and variables (columns) are in the mtcars dataset?
5. Rename mpg in cars to MPG. Use rename().
6. Convert the column names of cars to all upper case. Use rename_all, and the toupper command.
7. Convert the rownames of cars to a column called car using rownames_to_column.
8. Subset the columns from cars that end in "p" and call it pvars using ends_with().
9. Create a subset cars that only contains the columns: wt, qsec, and hp and assign this object to carsSub. (Use select().)
10. What are the dimensions of carsSub? (Use dim().)
11. Convert the column names of carsSub to all upper case. Use rename_all(), and toupper() (or colnames()).
12. Subset the rows of cars that get more than 20 miles per gallon (mpg) of fuel efficiency. How many are there? (Use filter().)
13. Subset the rows that get less than 16 miles per gallon (mpg) of fuel efficiency and have more than 100 horsepower (hp). How many are there? (Use filter() and the pipe operator.)
14. Create a subset of the cars data that only contains the columns: wt, qsec, and hp for cars with 8 cylinders (cyl) and reassign this object to carsSub. What are the dimensions of this dataset? Do not use the pipe operator.
15. Create a subset of the cars data that only contains the columns: wt, qsec, and hp for cars with 8 cylinders (cyl) and reassign this object to carsSub2. Use the pipe

operator.

16. Re-order the rows of carsSub by weight (wt) in increasing order. (Use arrange().)
17. Create a new variable in carsSub called wt2, which is equal to wt^2, using mutate() and piping %>%.

Please find solutions here.

Chapter 6

Write with R Markdown

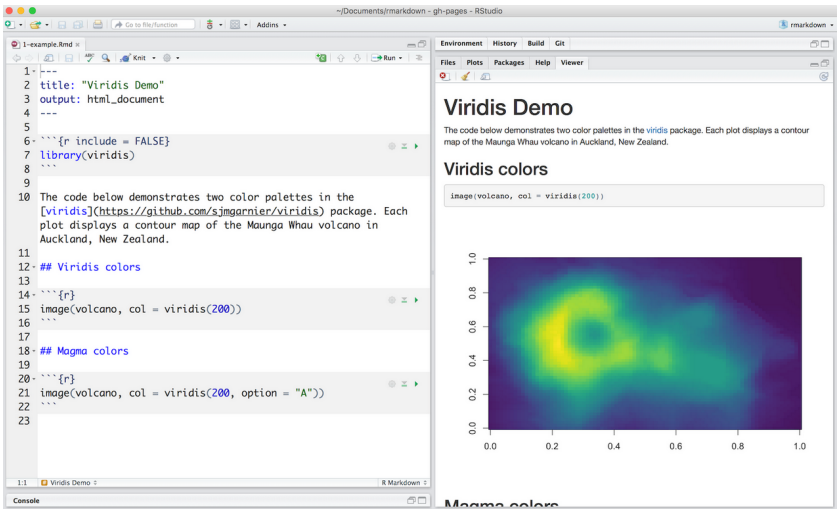

Figure 6.1: Example of an R Markdown file

R Markdown provides an authoring framework for data science. You can use a single R Markdown file to transcript your work, run code, and generate high quality reports, books, websites, articles, theses, blogs, and many more.

In contrast to Quarto (see: `https://quarto.org` and `https://quarto.org/docs/get-started/hello/rstudio.html`), which is the more recent format, R Markdown is around for some time and hence there are uncountable resources to learn it. For example:

- The R Markdown Cheatsheet from posit offers an overview on the most important features of R Markdown.

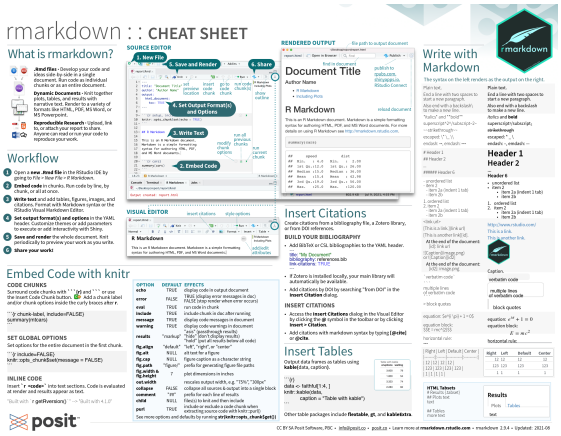

Figure 6.2: R Markdown Cheatsheet from posit

- The book *R Markdown Cookbook* by Xie, Dervieux, and Riederer (2020) offers an introduction. The online version of the book is regularly updated and free of costs.

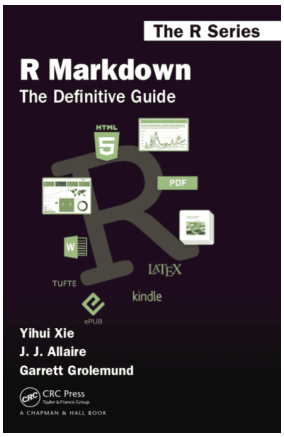

Figure 6.3: Xie et al. (2020): R Markdown Cookbook

- The book *R Markdown: The Definitive Guide* by Xie, Allaire, and Grolemund (2018) offers a comprehensive introduction. The online version of the book is regularly updated and free of costs.

Please watch the video What is R Markdown? and then study the R Markdown tutorial from RStudio.

Exercise 6.1. Start Markdown and R Markdown

a) You can learn Markdown (not R Markdown!) in 10 minutes. Just go to `https://www.markdowntutorial.com` and work throught the interactive lessons.

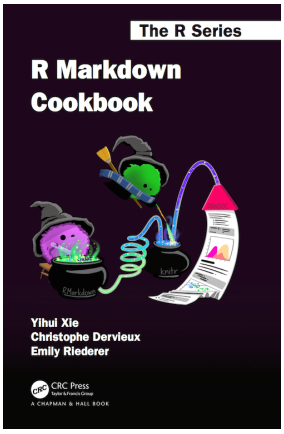

Figure 6.4: Xie et al. (2018): R Markdown: The Definitive Guide

b) Now create your first R Markdown file in 3 minutes by doing the following:

- click in RStudio on *File > New File > R Markdown*
- click *OK*
- look for a button entitled *Knit* and click it
- save your file (it will be saved with .Rmd file extension)

c) Play around with the file. For example, change the output format can you create a word file or a presentation. Play around with the code chunks. Add a picture that you find somewhere online.

d) Set your working directory to the folder where you have saved your first Rmd-file. Can you come up with a way to generate different output format with just one function.

Exercise 6.2. Please download and open the file "23-04_ds-project-desc.Rmd" from my GitHub account here. Save the file in your working directory and try to run it using the knit function. It may not work properly at first, so you may need to troubleshoot some issues. However, do not worry, as it are usually simply problems that occur and often the error message provided can guide you.

For instance, I use BibTeX to cite literature. It is an excellent method to make the computer perform tedious tasks for you, such as citing papers and generating reference lists based on citation styles. This can save you a significant amount of time and reduce the likelihood of errors while citing literature. The literature I cite is in a separate file, which you can find on one of my GitHub repositories.

Moreover, the YAML header is quite sensitive to spacing and will break you code.

Chapter 7

Appendix

7.1 Navigating the file system

It is essential to know how R interacts with the file system on your computer. Modern operating systems are incredibly user-friendly and try to hide boring and annoying stuff from the customer. In the following, I will try to give a brief introduction on how to navigate around a computer using a DOS or UNIX shell. If you familiar with that, you can skip this part of the notes.

7.1.1 The file system

In this section, I describe the basic idea behind file locations and file paths. Regardless of whether you are using Windows, macOS, or Linux, every file on the computer is assigned a human-readable address, and every address has the same basic structure: it describes a path that starts from a root location, through a series of folders (or directories), and finally ends up at the file.

On a Windows computer, the root is the storage device on which the file is stored, and for many home computers, the name of the storage device that stores all your files is `C:`. After that comes the folders, and on Windows, the folder names are separated by a backslash symbol \. So, the complete path to this book on my Windows computer might be something like this:

```
C:\Users\huber\Rbook\rcourse-book.pdf
```

On Linux, Unix, and macOS systems, the addresses look a little different, but they are more or less identical in spirit. Instead of using the backslash, folders are separated using a forward slash, and unlike Windows, they do not treat the storage device as being the

root of the file system. So, the path on a Mac might be something like this:

`/Users/huber/Rbook/rcourse-book.pdf`

That is what we mean by the *path* to a file. The next concept to grasp is the idea of a working directory and how to change it. For those of you who have used command-line interfaces previously, this should be obvious already. But if not, here is what I mean. The working directory is just whatever folder I am currently looking at. Suppose that I am currently looking for files in Explorer (if you are using Windows) or using Finder (on a Mac). The folder I currently have open is my user directory (i.e., `C:\Users\huber` or `/Users/huber`). That is my current working directory.

7.1.2 Working directory

The next concept to grasp is the idea of a *working directory* and how to change it. For those of you who have used command line interfaces previously, this should be obvious already. But if not, here's what I mean. The working directory is just "whatever folder I'm currently looking at". Suppose that I'm currently looking for files in Explorer (if you're using Windows) or using Finder (on a Mac). The folder I currently have open is my user directory (i.e., `C:\Users\huber` or `/Users/huber`). That's my current working directory.

The fact that we can imagine that the program is "in" a particular directory means that we can talk about moving *from* our current location *to* a new one. What that means is that we might want to specify a new location in relation to our current location. To do so, we need to introduce two new conventions. Regardless of what operating system you're using, we use `.` to refer to the current working directory, and `..` to refer to the directory above it. This allows us to specify a path to a new location in relation to our current location, as the following examples illustrate. Let's assume that I'm using my Windows computer, and my working directory is `C:\Users\huber\Rbook`. The table below shows several addresses in relation to my current one:

Absolute path	Relative path
`C:\Users\huber`	`..`
`C:\Users`	`..\..`
`C:\Users\huber\Rbook\source`	`.\source`
`C:\Users\huber\nerdstuff`	`..\nerdstuff`

It is quite common on computers that have multiple users to define `~` to be the user's *home directory*. The home directory on a Mac for the 'huber'' user is/Users/huber/. And so, not surprisingly, it is possible

to define other directories in terms of their relationship to the home directory. For example, an alternative way to describe the location of thercourse-book.pdffile on a Mac would be ``` ~\Rbook\rcourse-book.pdf ``` You can find out your home directory with thepath.expand()‘ function:

```
path.expand("~")
```

```
## [1] "/home/sthu"
```

Thus, on my machine ~ is an abbreviation for the path /home/sthu.

```
getwd()
```

```
## [1] "/home/sthu/Dropbox/hsf/23-ss/ds"
```

7.1.3 Navigating the file system using the R console

When you want to load or save a file in R it's important to know what the working directory is. You can find out by using the `getwd()` command. For the moment, let's assume that I'm using Mac OS or Linux, since things are different on Windows, see section 7.1.5. Let's check the current active working directory:

```
getwd()
```

```
## [1] "/home/sthu/Dropbox/hsf/23-ss/ds"
```

The function `setwd()` allows to change the working directory:

```
setwd("/Users/huber/Rbook/data")
setwd("./Rbook/data")
```

The function `list.files()` lists all the files in that directory:

```
list.files()
```

7.1.4 R Studio projects

Setting the working directory repeatedly can be a cumbersome task. Fortunately, R Studio projects can automate this process for you. When you open an R Studio project, the working directory is automatically set to the project directory.

Creating a new project in R Studio is simple. Just click on *File > New Project...*. This will create a directory on your computer with a _*.Rproj_ file that can be used to open the saved project at a later date. The newly created directory contains your R code, data files, and other project-related files. By working within projects, all of your files and data

are organized in one place, making it easier to share your work with others, reproduce your analyses, and keep track of changes over time.

7.1.5 Why do the Windows paths use the back-slash?

Let's suppose I'm using a computer with Windows. As before, I can find out what my current working directory is like this:

```
getwd()
[1] "C:/Users/huber/
```

R is displaying a Windows path using the wrong type of slash, the back-slash. The answer has to do with the fact that R treats the \ character as *special*. If you're deeply wedded to the idea of specifying a path using the Windows style slashes, then what you need to use two back-slashes \\ whenever you mean \. In other words, if you want to specify the working directory on a Windows computer, you need to use one of the following commands:

```
setwd( "C:/Users/huber" )
setwd( "C:\\Users\\huber" )
```

References

Irizarry, R. A. (2022). *Introduction to data science: Data analysis and prediction algorithms with R*. CRC Press. Accessed January 30, 2023. Retrieved from https://rafalab.github.io/dsbook/

Ismay, C., & Kim, A. Y. (2022). *Statistical inference via data science: A moderndive into R and the tidyverse*. CRC Press. Accessed January 30, 2023. Retrieved from https://moderndive.com/

Kirchkamp, O. (2018). *Using graphs and visualising data* (Tech. Rep.). (https://www.kirchkamp.de/oekonometrie/pdf/gra-p.pdf (retrieved on 2022/05/20))

Muschelli, J., & Jaffe, A. (2022). *Introduction to r for public health researchers* (This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.) GitHub. (https://github.com/muschellij2/intro_to_r)

Navarro, D. (2020). *Learning statistics with r* (Version 0.6 ed.). Accessed January 30, 2023. Retrieved from https://learningstatisticswithr.com

Thulin, M. (2021). *Modern statistics with r: From wrangling and exploring data to inference and predictive modelling*. Eos Chasma Press. Accessed February 30, 2023. Retrieved from https://www.modernstatisticswithr.com/

Timbers, T., Campbell, T., & Lee, M. (2022). *Data science: A first introduction*. CRC Press. Accessed January 30, 2023. Retrieved from https://datasciencebook.ca/

Tufte, E. R. (2022). *The visual display of quantitative information* (2nd ed.). Graphics Press.

Venables, W. N., Smith, D. M., & R Core Team. (2022). *An introduction to R: Notes on R: A programming environment for data analysis and graphics* (This manual is for R, version 4.1.3 (2022-03-10) ed.). (http://cran.r-project.org/doc/manuals/R-intro.pdf (retrieved on 2022/04/06))

Wickham, H., & Grolemund, G. (2023). *R for data science (2e)*. Accessed January 30, 2023. Retrieved from https://r4ds.hadley.nz/

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). *R markdown: The definitive guide*. Chapman and Hall/CRC.

Xie, Y., Dervieux, C., & Riederer, E. (2020). *R markdown cookbook*. Chapman

and Hall/CRC. (available at https://bookdown.org/yihui/rmarkdown-cookbook)