Psychological Research in R

An R Intro for people whose dog regularly ate their statistics homework.

Anni Tave Overlander

2024-08-29

Contents

About...

This book is still a work in progress! Most chapters are currently placeholders and will be filled over the next couple of weeks!

...this book

It's actually more of a course but book sounds really fancy.

Welcome! Chances are, you are a psychology student and either starting to learn R or looking to refresh your memory. Or perhaps you need to look up a specific step that you *can never quite remember* (or is that just me?). Maybe you also came across this resource by pure chance - lucky you!

In any case I am glad you are here and hope you find both what you were and weren't looking for. This book is based on an in-person introductory R course from the University of Konstanz. I tried my very best to cover all the basics on working with R from the ground up. Following the whole course should enable you to write your very own R Markdown report, taking full advantage of some of the most important and common features of R.

That being said, this book is quite *opinionated*, meaning I included all the lovely things that *I* like to work with. You might prefer other packages and that is completely okay - I still appreciate you reading my suggestions. As with anything in life, a lot can be learned from other peoples approaches to things.

...the author

I am Tave and I am currently working on my Ph.D. in Psychological Methods. This online book is a little side project that is quite near and dear to my heart.

In my experience, statistics and - goddess forbid - statistics programs can easily induce panic-like states in psychology students. And also in my experience, that can change over the course of one semester tops. Programming in R is a lot less scary than many may think and most of all, it can and *should be fun*! It is a weird, powerful language and can assist you with many everyday tasks.

I hope to alleviate some of the aveRsion over the course of this course and help you see R's advantages.

Please feel free to contact me if you have any questions or comments! You can reach out via e-mail: annika-tave.overlander@uni.kn or you can submit a GitHub issue over at the repository for this book: https://github.com/the-tave/psych_research_in_r.

Chapter 1

Intro and Installation

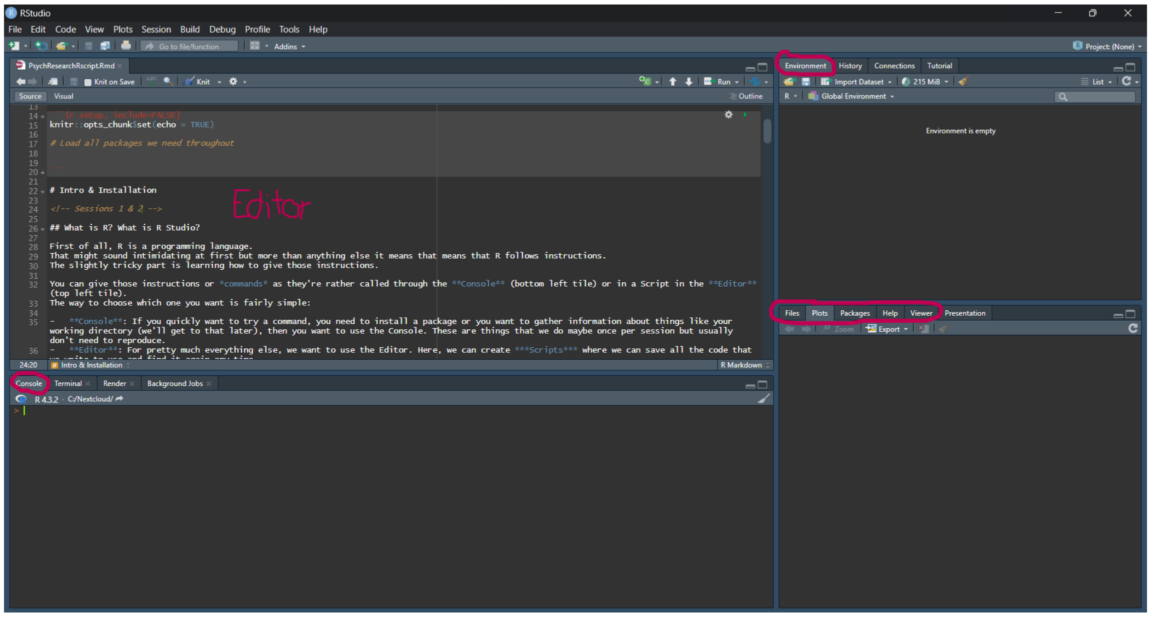
If you have no prior experience with R, the different names can get a little bit confusing. There is R, but there is also R Studio and the two serve different purposes in your workflow. So in the following I will try to make a clear distinction and guide you through the installation of both R and R Studio.

1.1 What is R? What is R Studio?

First of all, R is a programming language for statistical and data analysis.

It is open source, which means anyone can contribute - an many many many people do. Like a lot of other programming languages, R can be used with the functions that it understands without any further instructions - we call this base R - or you can use *packages* - think of them as new tricks that you can teach your R.

That might all sound intimidating at first but more than anything else it means that means that R follows instructions. The slightly tricky part is learning how to give those instructions.



~~~
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

# Load all packages we need throughout


```

# Intro & Installation

<!-- Sessions 1 & 2 -->

## What is R? What is R Studio?

First of all, R is a programming language.
That might sound intimidating at first but more than anything else it means that means that R follows instructions.
The slightly tricky part is learning how to give those instructions.

You can give those instructions or *commands* as they're rather called through the **Console** (bottom left tile) or in a Script in the **Editor**
(top left tile).
The way to choose which one you want is fairly simple:

-   **Console**: If you quickly want to try a command, you need to install a package or you want to gather information about things like your
working directory (we'll get to that later), then you want to use the Console. These are things that we do maybe once per session but usually
don't need to reproduce.
-   **Editor**: For pretty much everything else, we want to use the Editor. Here, we can create ***Scripts*** where we can save all the code that
~~~

Figure 1.1: R Studio - typical layout

1.2 Installing R and R Studio

- Download & Install the newest R version 4.3.3 (2024-02-29 ucrt) at https://cloud.r-project.org/
- Download and install R Studio at https://posit.co/download/rstudio-desktop/

R as a language can be used on its own. However, it is not very modern, quite hard to use and frankly just no fun. That's why we use R Studio as a user interface to run R. Think of it as dipping your fingers in a pot of ink to write versus using a pen and paper - you will still write with the same ink, but the whole process is just nicer.

1.2.1 Workflow

You can give instructions or *commands* as they're rather called through the **Console** (bottom left tile) or in a Script in the **Editor** (top left tile). The way to choose which one you want is fairly simple:

- **Console**: If you quickly want to try a command, you need to install a package or you want to gather information about things like your working directory (we'll get to that later), then you want to use the Console. These are things that we do maybe once per session but usually don't need to reproduce.

- **Editor**: For pretty much everything else, we want to use the Editor. Here, we can create ***Scripts*** where we can save all the code that we write to use and find again any time.

For most things - especially during the learning phase - it makes sense to write a Script in the Editor in order to be able to save and access the work. To do so, simply click the button and choose 'R Script'.

Once we have a new script created, we can learn some basic things that R is capable of and save it to check out later. Math

```
# Basic Math
1+2
```

```
## [1] 3
```

```
2-3
```

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

```
3*4
```

```
## [1] 12
```

```
4/3
```

```
## [1] 1.333333
```

```
5^2
```

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

```
# Assigning Variables
a <- 3
b <- 4
a-b
```

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

```
a*b+5
```

```
## [1] 17
```

In this `code chunk`, R is essentially being used as a calculator to perform basic math. While we want to make use of all the more powerful functions of R, it is important to grasp the basics and be able to use arithmetic for our purposes. Next to the basic mathematical operators, there are many nifty mathematical functions, such as `sqrt()` - *square root*, `sum()` or `pi`.

We also added **variables** containing values (here a and b hold values 3 and 4 respectively) that can be used in the calculations just like the values they contain. As they only contain numbers, the variables a and b are called **numeric**. We can check this property of a variable, e.g. `a` using the function `class(a)`, which gives us "numeric" as output.

Usually when we fire up R, we don't just want to work with single values numeric but with **data frames** or **vectors** that can contain different classes of variables and several values respectively.

```
# Vectors
c <- c(6, 7, 8)
class(c)
```

```
## [1] "numeric"
```

```
d <- c("sunny", "rainy", "foggy")
class(d)
```

```
## [1] "character"
```

```
# Data Frame
e <- data.frame(c,d)
class(e)
```

```
## [1] "data.frame"
```

Notice that we did not just add all values one after the other, but followed a certain notation that begins with the function `c()`. The c stands for "combine" or "concatenate" and tells R that all following values belong to the same variable. There are several ways of adding variables to a data frame, but the `data.frame()` command is the simplest.

All values in a variable should have the same class. Go ahead and try out `hm <- c(3, "sunny", 5.2)` and check the class. What happened - did you expect that?

When we have our data in a neat data frame, what we usually want to do is access either certain **rows** or **columns**. To do so, base R uses square brackets `[]` behind the name of a data frame to indicate "take this data, but only certain rows/columns/cells". Remember: The brackets understand the first input as rows and the second as columns - *rows right away*.

```
e
```

```
##   c     d
## 1 6 sunny
## 2 7 rainy
## 3 8 foggy
```

```
e[1, ]
```

```
##   c     d
## 1 6 sunny
```

```
e[ , 1]
```

```
## [1] 6 7 8
```

```
weather <- e[ , 2]
weather
```

```
## [1] "sunny" "rainy" "foggy"
```

```
e[1, 1]*e[2, 1]
```

```
## [1] 42
```

Our data set `e` contains three numbers in column c and different strings in column d. We can select just the first row, or just the first column or either of the other rows and columns that we have in the data. We can also re-assign the values, e.g. to a new variable named "weather" or perform calculations on single cells in the data frame - only if they contain numerics, of course.

With square brackets we can only index rows and columns that are present in the data.

That maybe sounds obvious, but can easily lead to confusion because of error messages! What happens when you try to index `e[,3]`?

While it is common practice to work with data frames and edit them according to our tasks and needs, using square brackets and base R can get a bit humdrum. Luckily, many many people have developed many many packages that contain different functions, helping us in most tasks that we will need to tackle!

1.3 Installing Packages

Figure 1.2: Hexagon Package logos

Most packages are available on CRAN - the Comprehensive R Archive Network. That being the case you can easily install the package you want to have with the command `install.packages("packagename")`. It is important to note that the package name must be in quotes for the installation, while loading it into a script works without quotes using `library(packagename)`.

Packages usually serve quite specific purposes, e.g. ones we will later get to know are `dplyr`, with which data handling is made a lot easier and more intuitive and `ggplot2`, which allows us to create beautiful, publication-ready plots and visualizations. What is special about these two, among some others, is that they were developed by the same person (Hadley Wickham), and are made available in a sort of “meta package” - the `tidyverse`. Installing and loading the `tidyverse` makes the functions from many different packages available at once. This is quite convenient when we want to use a lot of those packages in the same session or script, but does also take a longer time to load and is sometimes not actually necessary.

To use this package of packages, please install the `tidyverse`, using `install.packages("tidyverse")` in the console and then load it into your script (or, again, directly in the console, bottom-left) with `library(tidyverse)`. You can test whether it works by running `iris %>% pull(Sepal.Length)`

`%>% mean()` in your console. We will get to know the syntax in depth another time, but just so you know what's going on: This line of code takes the data set `iris`, which is included in R by default, "pulls" the variable `Sepal.Length` out of the data and runs the `mean()` function on it, to calculate the average sepal length, which should be 5.8433333.

- R is a powerful language, R Studio is the user interface we use with it
- R can be a calculator and perform basic and advanced math
- We mostly work with variables and data frames
- Packages make working with R easier and more fun!

- Tadaa Data: R für Psychos (german)
- Intro to R
- Tidyverse

Chapter 2

R Basics and how to read error messages

2.1 Basics

```r
library(dplyr)
```

R is an object-based language. The great advantage of that is we can assign values, vectors, text, matrices or almost anything else to variables and access them more easily later. To do that, we use the ← like so:
`x <- c(1, 2, 3)` Go ahead and try it out! If you had too much fun assigning variables, you can remove them again from your work environment using the `rm()` command in the Console. So, if you assign the values 1, 2 and 3 to `x`, you can type `rm(x)` into the console, hit enter and x has disappeared from your environment!

- Execute a line of code with `ctrl + enter` in the editor
  - In console you just need to press `enter`
- Text aka *strings* need to be put in quotes so it can be recognized as such
  - `some_text <- c("amazing", "wow")`

Figure 2.1: Iris Flower

- Comments can and should be added to your code using the #

2.1.1 Basic Functions I

```
a <- c(1, 2, 3, 4)
b <- c(5, 7, 9, 11)
mean(a)
```

```
## [1] 2.5
```

```
sd(b)
```

```
## [1] 2.581989
```

```
min(b)
```

```
## [1] 5
```

2.1.2 Basic Functions II

```
max(b)
```

```
## [1] 11
```

```
a+b
```

```
## [1]  6  9 12 15
```

```
sum(b)
```

```
## [1] 32
```

```
length(a)
```

```
## [1] 4
```

2.1.3 Basic Functions III

```r
c <- 3:9
c # look at the variable
```

```
## [1] 3 4 5 6 7 8 9
```

```r
range(c)
```

```
## [1] 3 9
```

```r
d <- c(10:15, 20:25)
```

2.2 Basic Functions IV

```r
d # look at the variable
```

```
##  [1] 10 11 12 13 14 15 20 21 22 23 24 25
```

```r
d[1] # index
```

```
## [1] 10
```

```r
d[d==10]
```

```
## [1] 10
```

```r
which(d==12)
```

```
## [1] 3
```

2.3 Exercise

Create a vector x with the numbers from 50 to 100 and 150 to 200.

Find its mean, standard deviation and its range.

What is the 77th number of vector x?

2.3.1 Solution

- `x <- c(50:100, 150:200)`

- `mean(x), sd(x), range(x) %>% round(2)`:
  125, 52.38, 50, 200
  - `%>% round(2)` takes the numbers and rounds them up to two decimal points
- `x[77]`: 175

2.3.2 Brainteaser

We define two variables in the following way:

```
e <- 1:5
f <- 2:5
```

Let's say we want to access the number 5 in both variables. How come `e[5]` works but `f[5]` does not?

2.4 Logic

- next to numeric and string variables: *boolean*
  - either hold the value **TRUE** or **FALSE** (also 1 or 0 respectively)
- useful for filtering data
  - also for conditional operations and recoding

2.4.1 Logical Examples I

```
a <- 4; b <- 5
a < b # smaller than
```

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

```
a <= 4 # smaller/ equal
```

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

```r
a > b # greater than
```

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

```r
a >= 3 # greater/ equal
```

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

2.4.2 Logical Examples II

```r
a == b # double equal sign!!
```

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

```r
a != b
```

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

```r
c <- TRUE
c
```

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

```r
!c
```

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

2.4.3 Putting it all together...

2.4.3.1 Exercise

We want to find out whether the average Petal Length and Sepal Length of iris flowers are different from each other.

Use what you know to "ask R" if those values are unequal!

Hint: You will need logic, the mean()-function and square brackets. Use **names(iris)** to find all variable names in that data set.

2.4.4 Solution

- Square bracket indexing with the variable name in quotes

  `iris[ , "Sepal.Length"]`

  `iris[ , "Petal.Length"]`
- now we add the mean function around both like this: `mean(iris[ , "Sepal.Length"])`

  `mean(iris[ , "Sepal.Length"])`
- Finally we compare the two with the != operator: `mean(iris[ , "Sepal.Length"]) != mean(iris[ , "Petal.Length"])`

  TRUE

2.5 Reading Error Messages

Even the most advanced R coder will encounter the occasional error message. While they are quite helpful and usually lead to being able to solve problems, it can be challenging to learn how to properly read the message in order to actually understand what the problem is. Therefore, we will look at some of the most common wordings to decipher what R needs so it can understand what we want to do.

Think of error messages not as discouraging faults of your program but rather as invites to help you help R understand what you are trying to achieve.

Error: unexpected 'X' in "Y"

This message is fairly straightforward: R expected a certain kind of symbol at the place where we put 'X' and is now confused because our input did not match the expectation.

For most Europeans, for example, the most commonly used decimal separator will be a comma. Since R is an american-built language, however, it expects decimals to be separated with a decimal point while commas indicate separate inputs to a function. Therefore, when we accidentally input a decimal as '3,141', we will receive the error message *Error: unexpected ',' in "3,"* because R expected the input to be something like '3.141'.

Fix: Replace the 'x' in "Y" with something that R can understand. Or alternatively: Make sure that we meant the input like that.