Introduction to R Programming

Getting Started

Pedro Fonseca

08 May 2020

R and Rstudio

- R is a programming language and a free and open source software environment for statistical computing and graphics
- RStudio is an integrated development environment (IDE) for R with free/open source and commercial versions
- You can use R without using RStudio, but you can't use Rstudio without using R

This is how R looks like

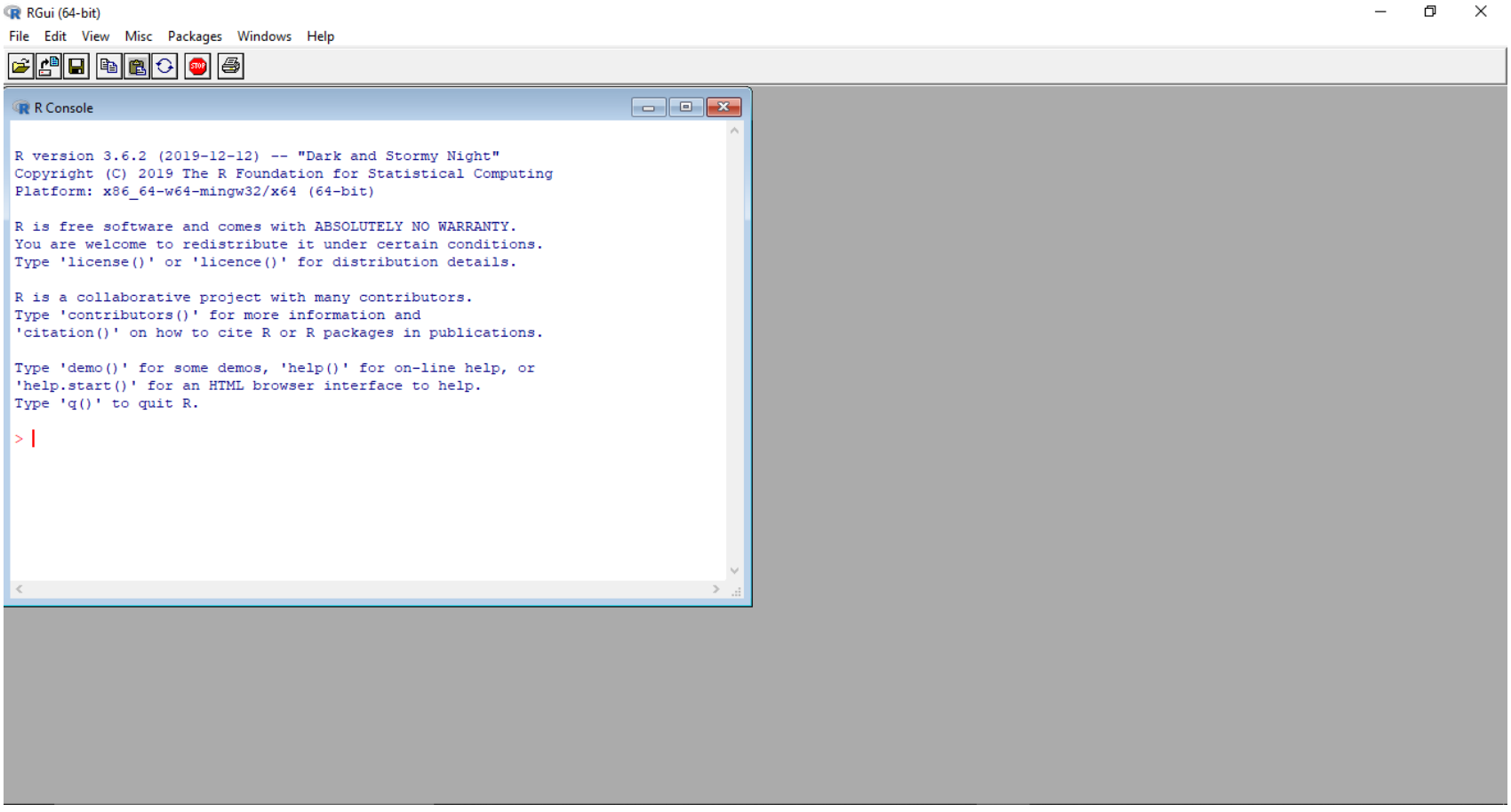


Figure 1: R on Windows

This is how R looks like

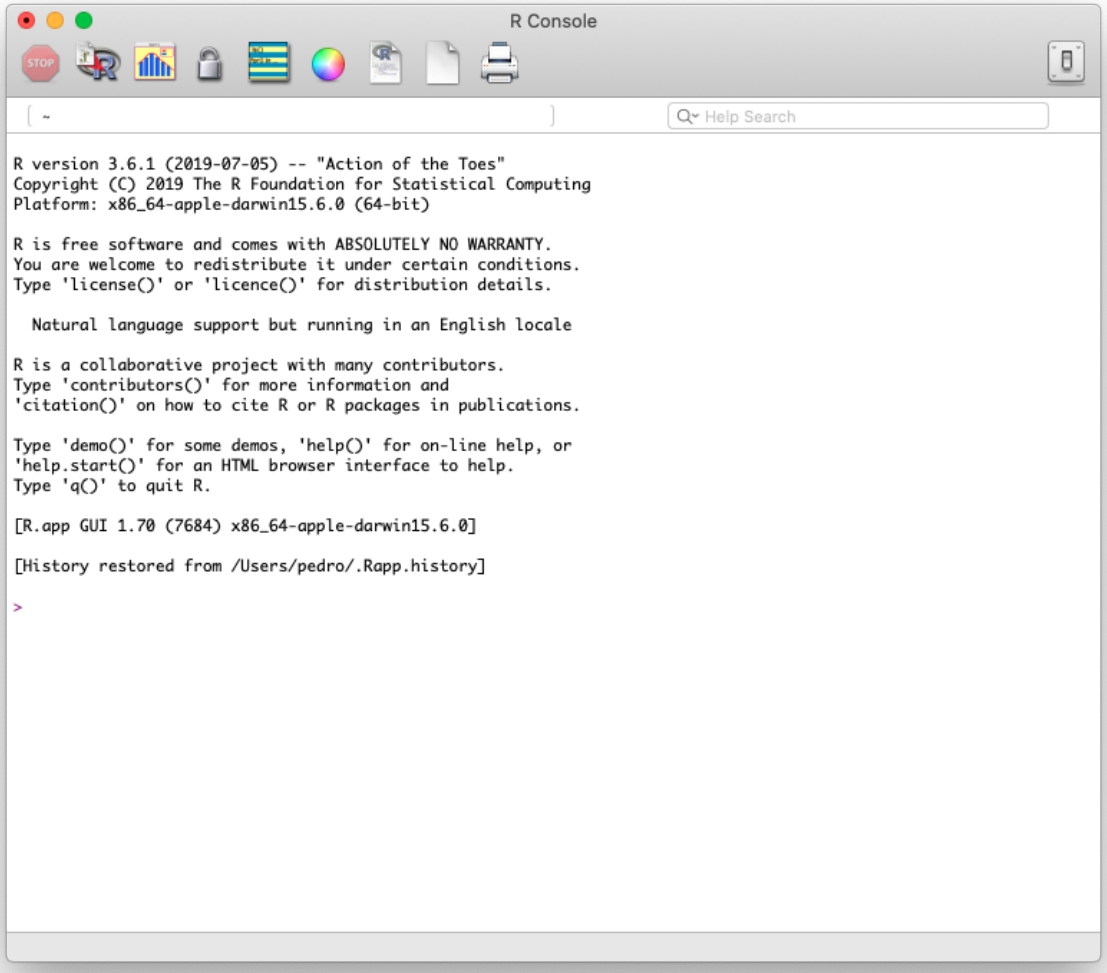


Figure 2: R on macOS

This is how R looks like

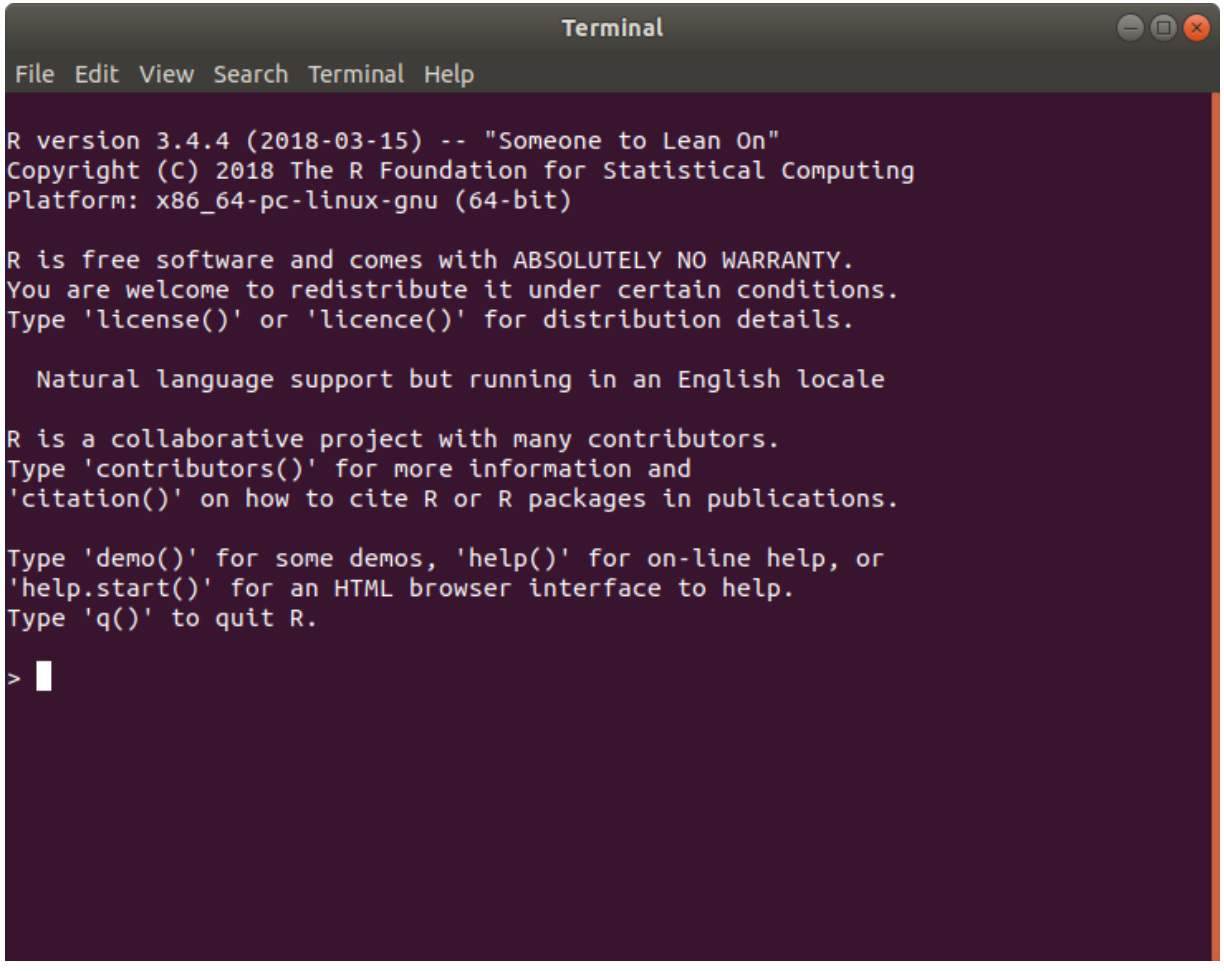


Figure 3: R on Ubuntu

This is how Rstudio looks like

Rstudio provides a user-friendly and interactive interface to R:

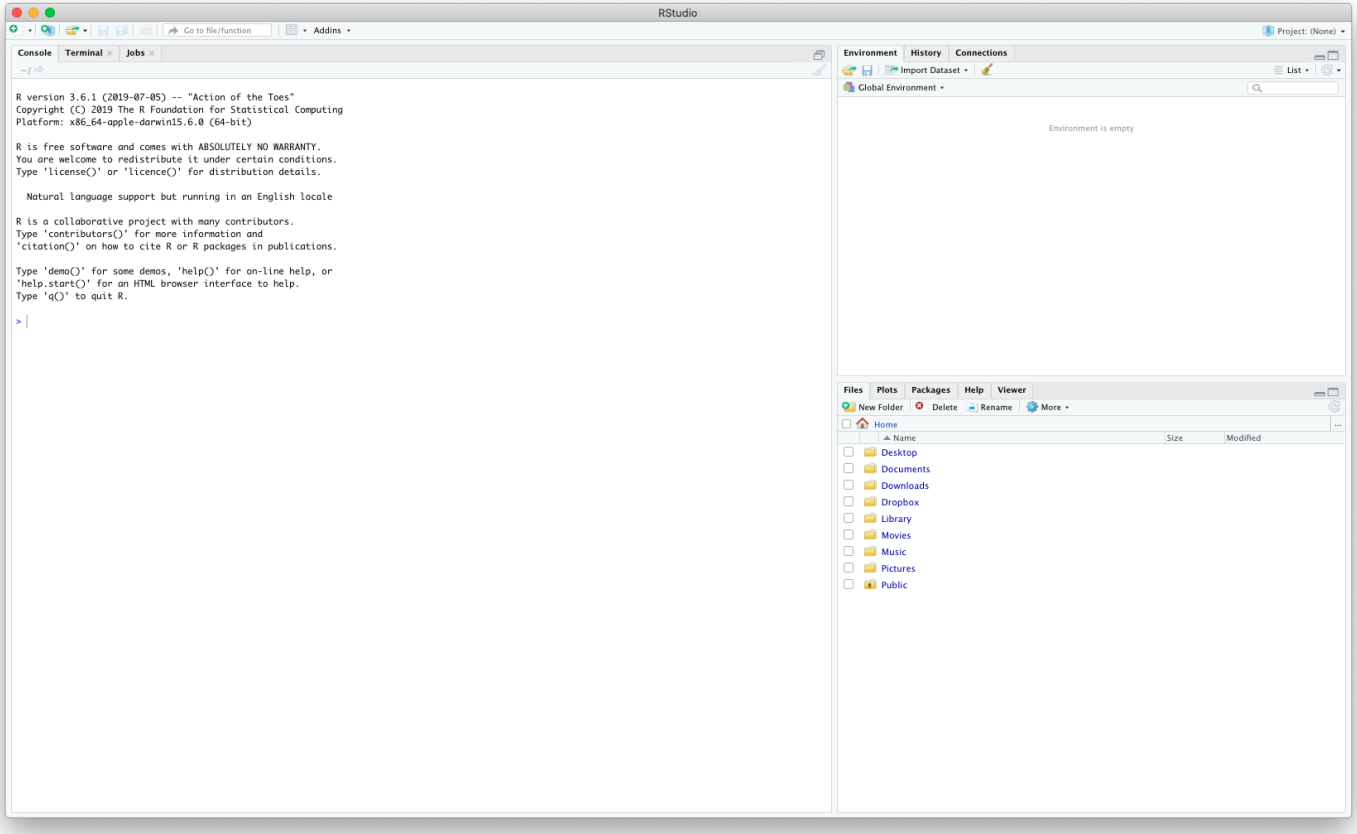


Figure 4: RStudio

The console

The pane on the left is the console. It can be used as a calculator:

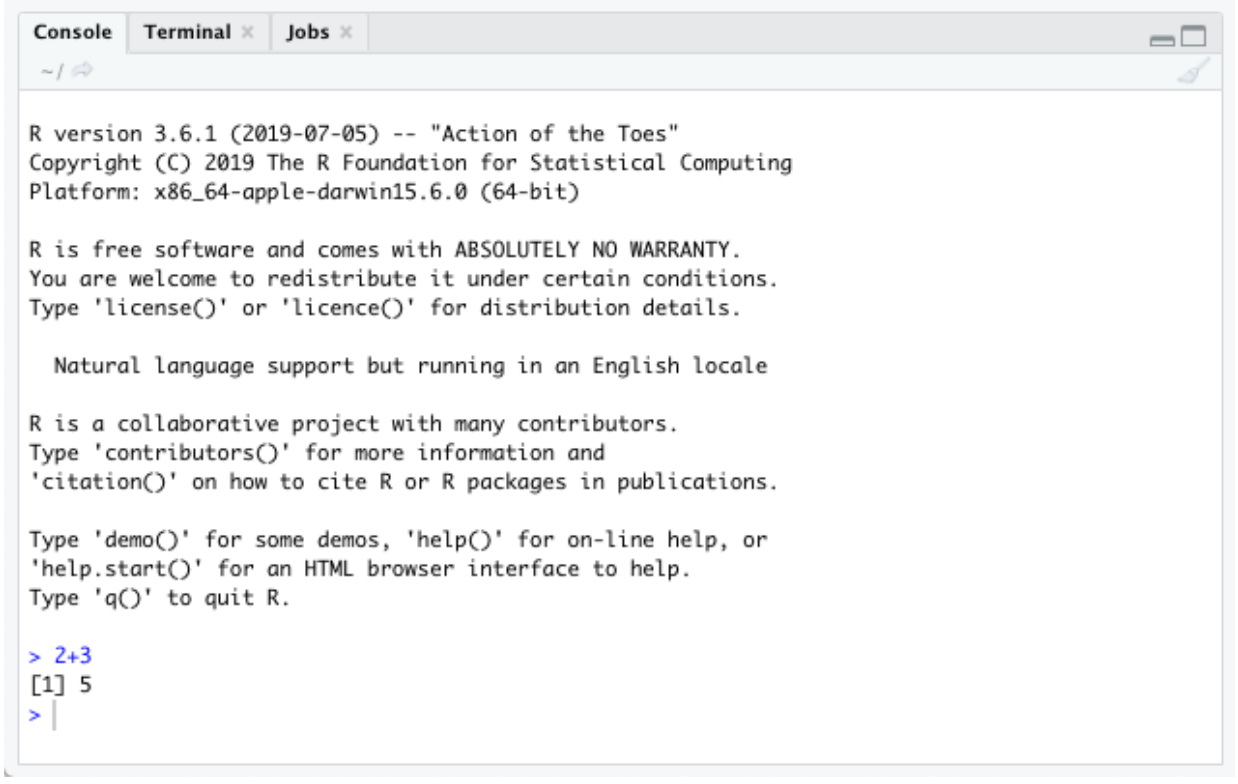


Figure 5: R's console as a calculator

R as a calculator

```
2 + 3
```

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

```
3 * 5
```

```
## [1] 15
```

```
14.5 / 6
```

```
## [1] 2.416667
```

```
3 ^ 2
```

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

R as a calculator

We can chain as many operations as we want. But be carefull with the parentheses!

```
(3 ^ 2) + 14 / (6 + 5)
```

```
## [1] 10.27273
```

```
(3 ^ 2) + 14 / 6 + 5
```

```
## [1] 16.33333
```

R as a calculator

Square root:

```
sqrt(x = 25)
```

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

Natural logarithm:

```
log(x = 5)
```

```
## [1] 1.609438
```

Base 10 logarithm:

```
log10(x = 5)
```

```
## [1] 0.69897
```

R as a calculator

Exponential function:

```
exp(x = 1)
```

```
## [1] 2.718282
```

```
exp(x = 3)
```

```
## [1] 20.08554
```

R as a calculator

Nested operations:

```
10^log10(x = 5)
```

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

```
log(x = exp(x = 4))
```

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

```
sqrt(x = 25)^2 + log(x = exp(x = 5))
```

```
## [1] 30
```

R as a calculator

Trigonometric functions:

```
pi
```

```
## [1] 3.141593
```

```
cos(x = 2 * pi)
```

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

```
tan(x = 0.6)
```

```
## [1] 0.6841368
```

```
sin(x = 0.6) / cos(x = 0.6)
```

```
## [1] 0.6841368
```

Functions

- R has a large collection of built-in functions
- We already used `log`, `log10`, `sqrt`, `exp`, `sin`, `cos` and `tan`

Functions

Most functions have more than one argument:

- Some arguments are mandatory
- Some arguments are optional and have default values

Functions

`log` has two arguments:

- `x` is mandatory
- `base` is optional. The default value of `base` is `exp(1)`

```
log(x = 243)
```

```
## [1] 5.493061
```

```
log(x = 243, base = exp(1))
```

```
## [1] 5.493061
```

```
log(x = 243, base = 6)
```

```
## [1] 3.065736
```

Functions

`round` rounds numbers to a specified number of decimal places. It has two arguments:

- `x` is mandatory
- `digits` is optional. The default value of `digits` is 0

```
round(x = 5.23452)
```

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

```
round(x = 5.23452, digits = 2)
```

```
## [1] 5.23
```

```
round(x = 5.23452, digits = 3)
```

```
## [1] 5.235
```

Functions

`sqrt`, `log10()`, `exp`, `sin`, `cos` and `tan` have only one argument, `x`, and it is mandatory:

```
sqrt(x = 5)
```

```
## [1] 2.236068
```

```
log10(x = 5)
```

```
## [1] 0.69897
```

```
exp(x = 5)
```

```
## [1] 148.4132
```

Functions

Arguments names are not mandatory:

```
log(x = 5, base = 10)
```

```
## [1] 0.69897
```

```
log(x = 5, 10)
```

```
## [1] 0.69897
```

```
log(5, base = 10)
```

```
## [1] 0.69897
```

```
log(5, 10)
```

```
## [1] 0.69897
```

Functions

Dropping argument names is safe in functions of one argument:

```
sqrt(x = 25)
```

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

```
sqrt(25)
```

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

Functions

R does positional matching for unnamed arguments. Therefore, in functions with more than one argument if you don't provide the names of the arguments you must input them in the correct order:

```
log(243, 2)
```

```
## [1] 7.924813
```

```
log(2, 243)
```

```
## [1] 0.126186
```

Functions

If you provide argument names, the order is irrelevant:

```
log(x = 243, base = 2)
```

```
## [1] 7.924813
```

```
log(base = 2, x = 243)
```

```
## [1] 7.924813
```

Functions

Help pages can be useful:

```
?log
```

In the help page of a function you can find:

- An ordered list of arguments
- Details about the arguments and their admissible values
- The interpretation of the output of the function
- Examples
- Related functions

Functions

Most of the times it is safe to drop the name of the first argument. Providing the names of the remaining arguments is usually a good idea: it avoids mistakes and improves readability.

```
log(4, base = 3)
```

```
## [1] 1.26186
```

```
round(pi, digits = 2)
```

```
## [1] 3.14
```

```
round(sqrt(2), digits = 4)
```

```
## [1] 1.4142
```

R scripts

- So far we've been using Rstudio's console
- Code sent directly to the console is executed but you won't be able to modify it or reuse it later
- Using scripts is a better option
- A script is just a text file we can use to write code

Your first R script

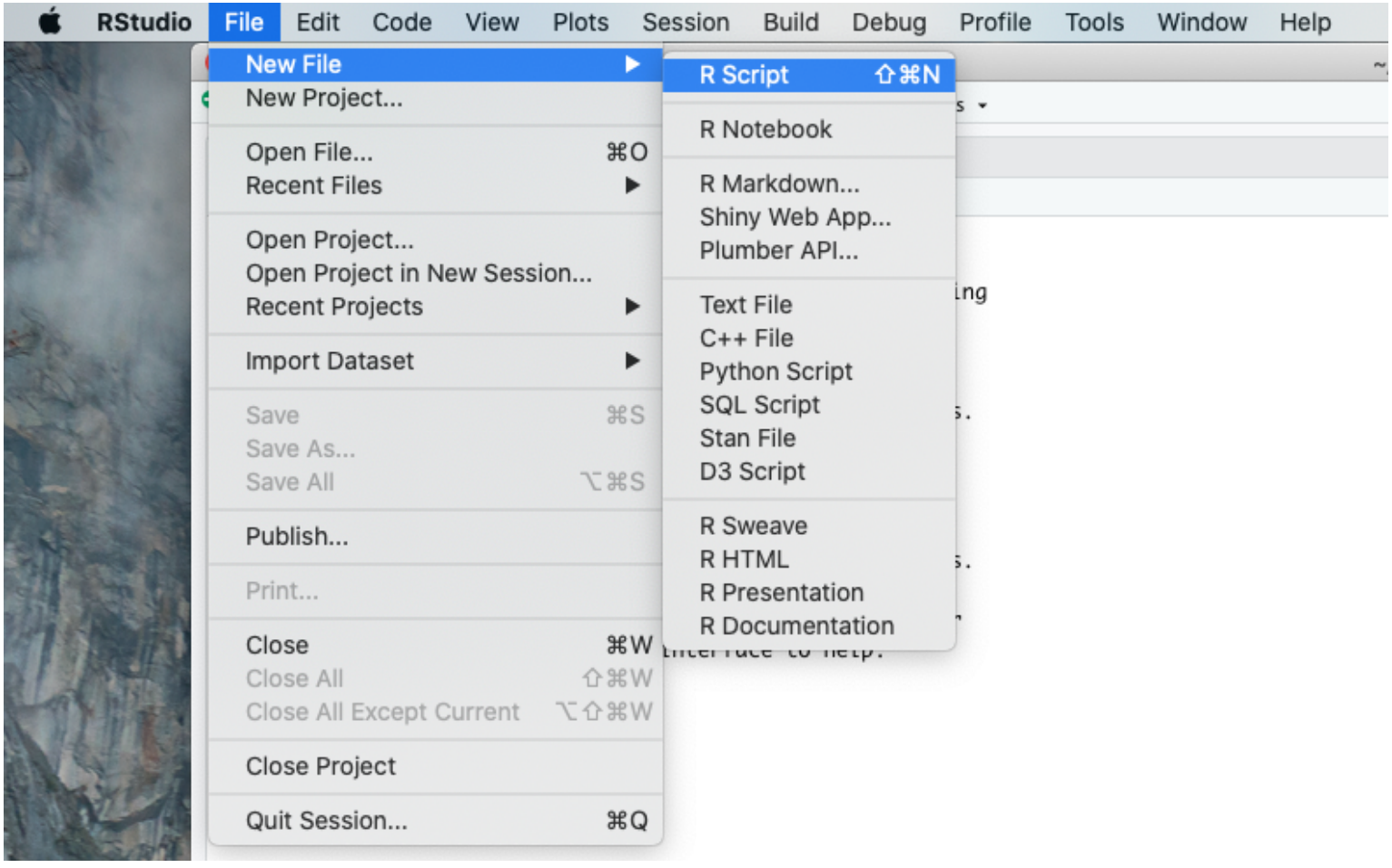


Figure 6: Creating a new script

Editor

- R opens scripts in the editor pane
- This is where you should write your code
- In the editor you can modify, rerun and save your code at any time

Rstudio Panes

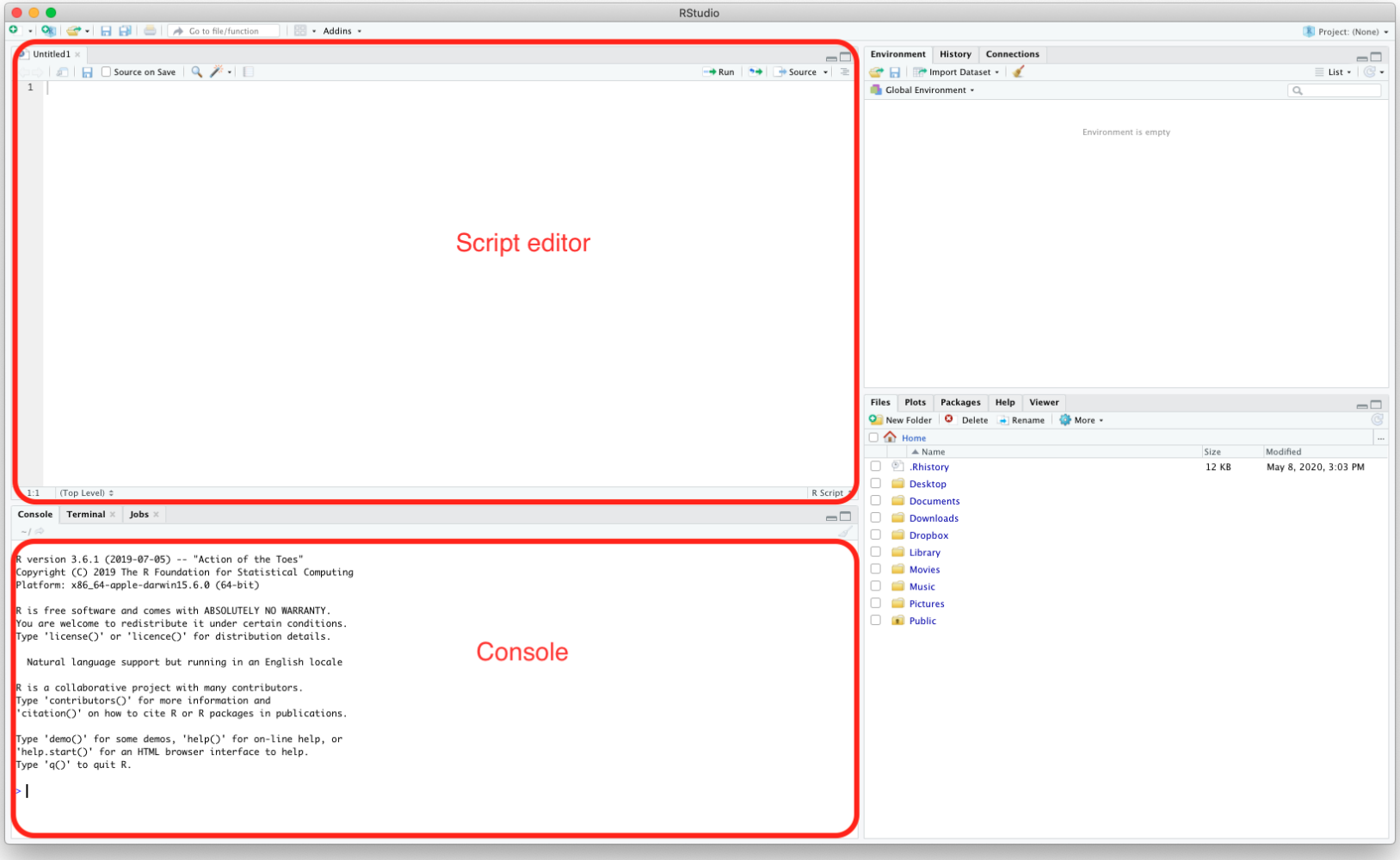


Figure 7: R studio's console and editor

Some useful shortcuts

- New script: Cmd/Ctrl + Shift + N
- Save the script: Cmd/Ctrl + S
- Send code from the script to the console:
  - Cmd/Ctrl + Enter (current line or current selection)
  - Cmd/Ctrl + Shift + S (entire script)

The assignment operator

To store values in R's memory you need to assign them to objects. You can use the equal sign (=) or the assign operator (<-)

```
x <- 5
x
```

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

The assignment operator

- The assignment operator is recommended
- The equal sign should be reserved to provide arguments to functions
- Rstudio's shortcut to the assignment operator: "Alt/Option" + "_"

The assignment operator

The assignment operator

Values stored in objects can be used in calculations:

```
y <- log(x) + exp(2)
y
```

```
## [1] 8.998494
```

```
x + 2 * y
```

```
## [1] 22.99699
```

The assignment operator

Stored objects are visible in the upper-right pane, under the "Environment" tab:

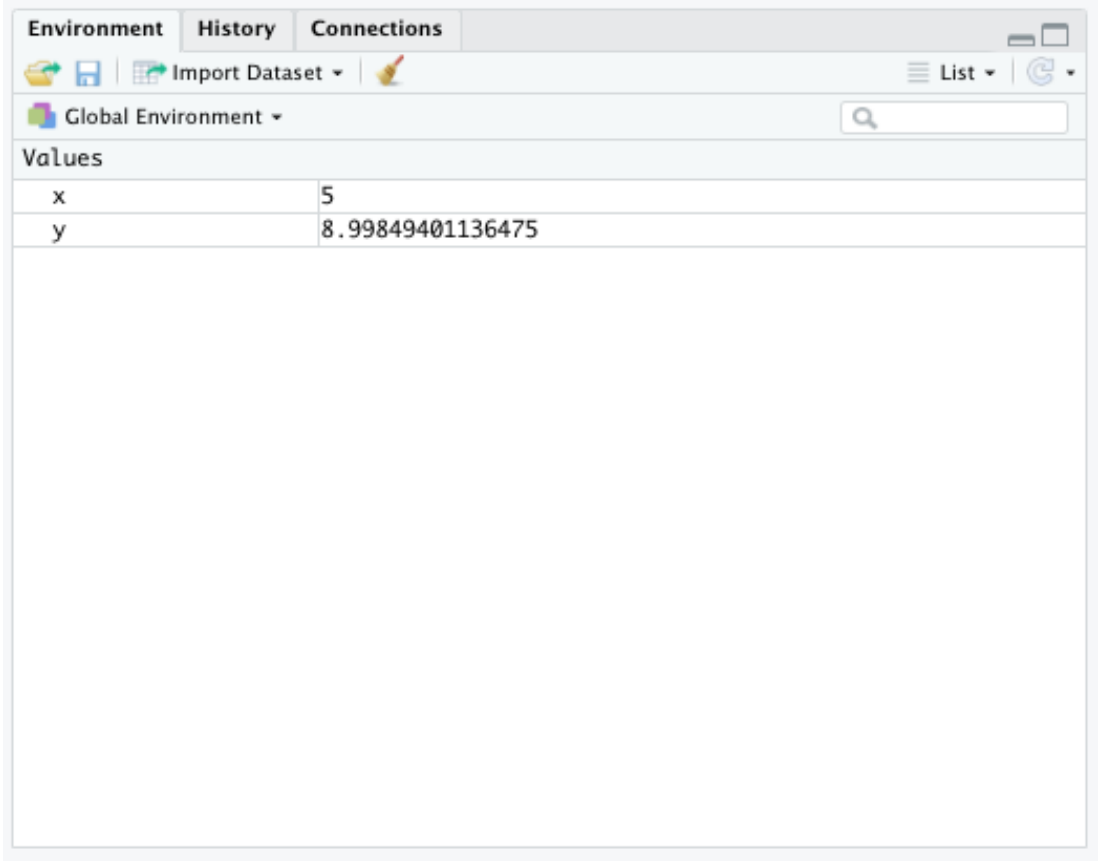


Figure 8: Our session's global envoronment

Workflow

Our workflow so far:

- Write code in the editor
- Send code to the console
- The code is exectued and the results are printed in the console
- The objects we created are listed in the environment tab

Workflow

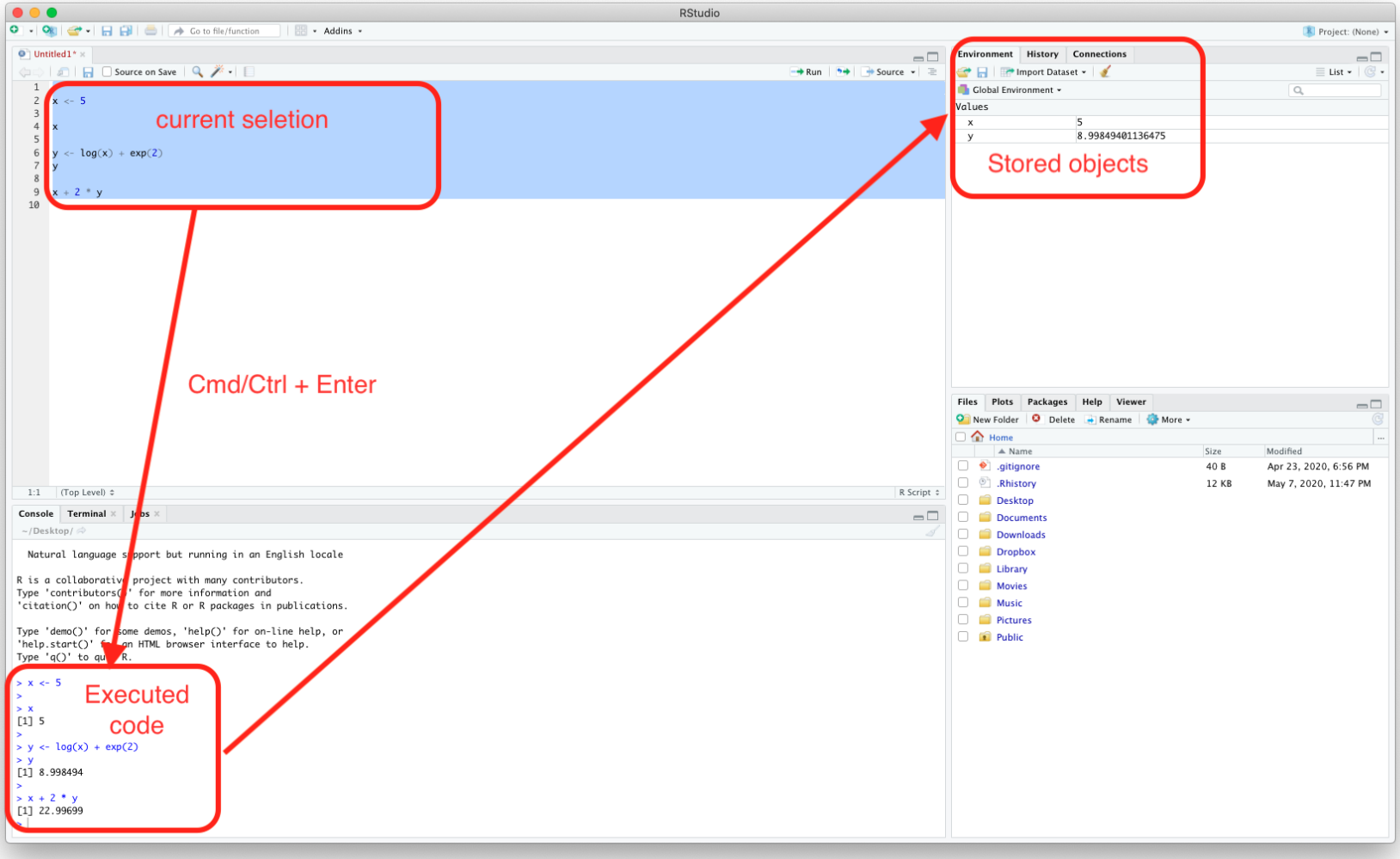


Figure 9: Workflow

Commenting

- We can make comments in our code using #
- Lines starting with # are printed in the console but are not executed

Commenting

```
#=================================================
# Intro to R programming - Lecture 0
#=================================================

# Lets store the value "5" in an object called x
x <- 5

# Now let's print x
x
```

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

Saving scripts

Since we already edited our script, let's save it:

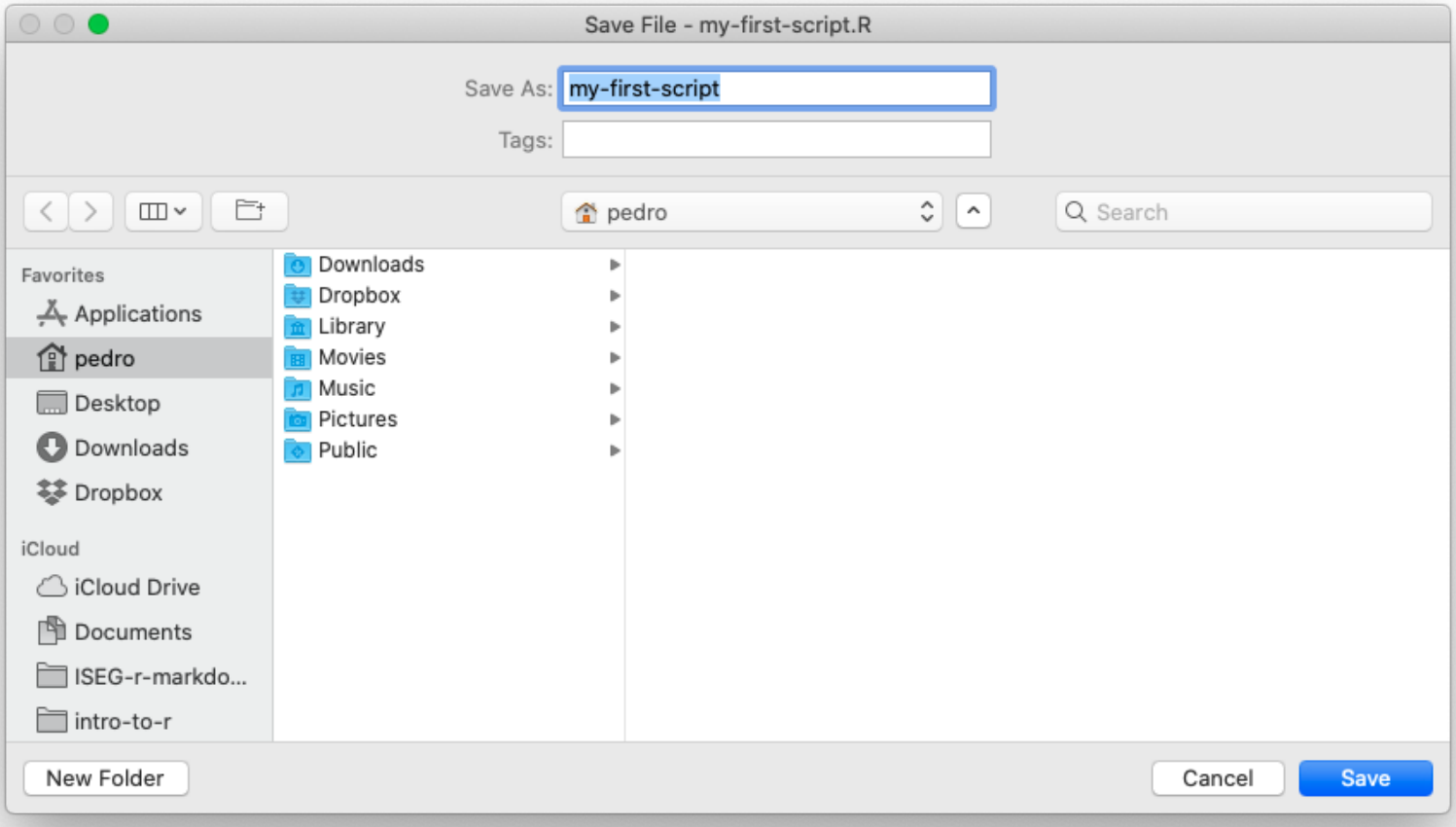


Figure 10: Saving a script

Saving scripts

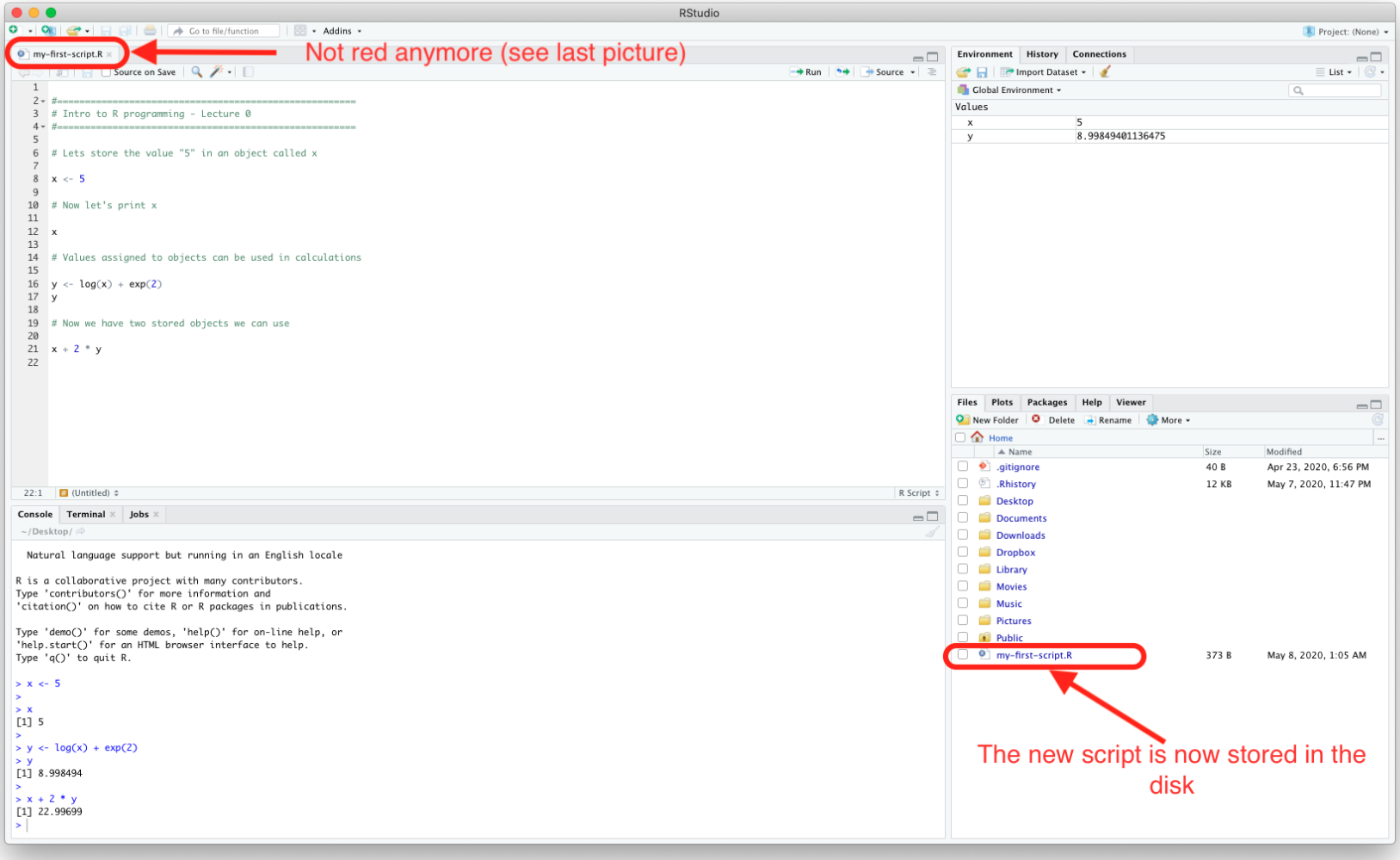


Figure 11: The script is now saved

Naming Objects

- Object names must start with a letter and can only contain letters, numbers, underscores and dots
- Ideally, one should follow a convention
- Object names should to be short, descriptive and consistent

Case matters

```
r_rocks <- 2
r_rocks
```

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

```
r_Rocks
```

```
## Error: object 'r_Rocks' not found
```

How to delete objects

To delete stored objects use the `rm` function:

```
r_rocks
```

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

```
rm(r_rocks)
```

```
r_rocks
```

```
## Error: object 'r_rocks' not found
```

How to delete objects

You can input as many objects as you want to `rm()`:

```
rm(x, y)
```

To remove all stored objects all once, use the following command:

```
rm(list = ls())
```

Overwritting stored values

```
x <- -5
x
```

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

```
x <- x + 1
x
```

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

```
x <- round(log(3)/2, digits = 2)
x
```

```
## [1] 0.55
```

Working directory

An active R session always has an associated working directory. R will use the working directory by default to:

- Search for files
- Save files
- Save outputs (tables, plots, etc)

Getting the working directory

To check your workig directory:

```
getwd()
```

Setting the working directory

You can change your working directory using RStudio's menus:

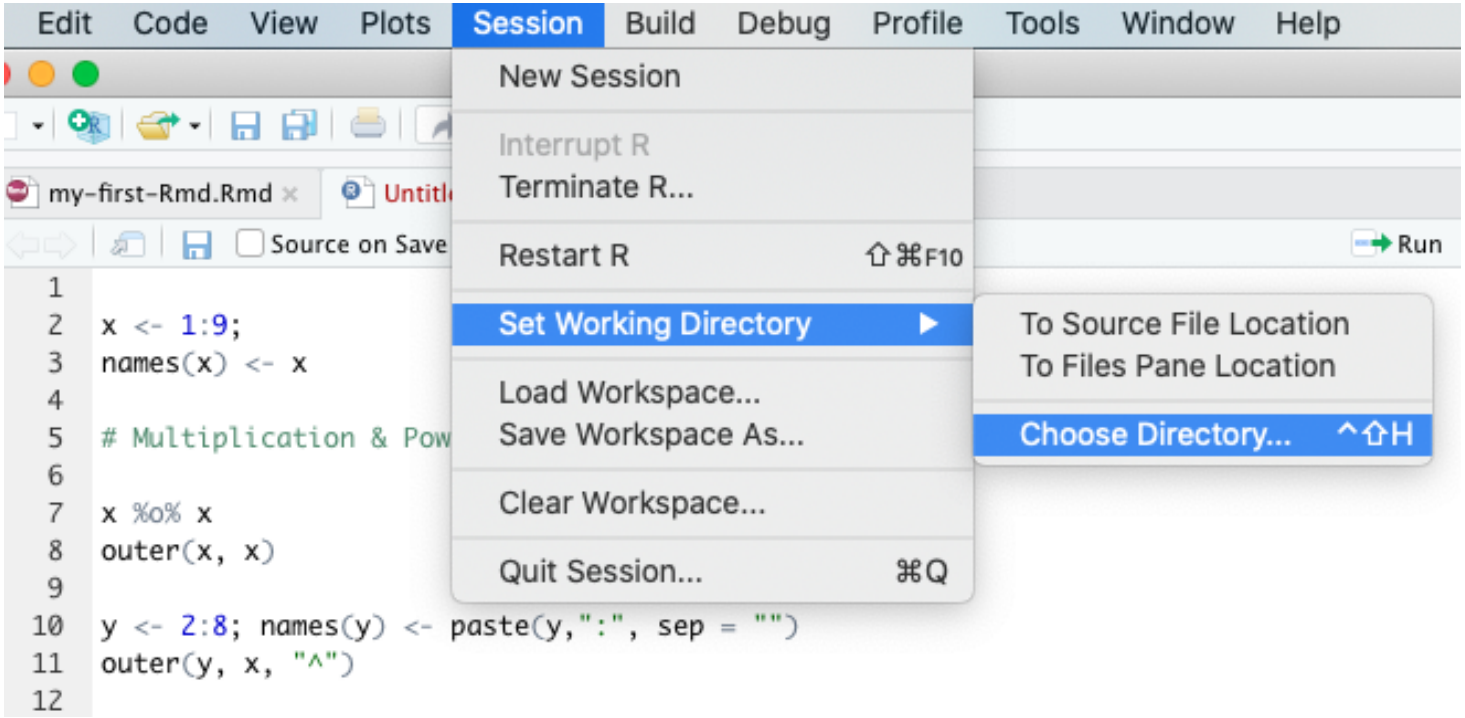


Figure 12: Setting the working directory

Setting and setting the working directory

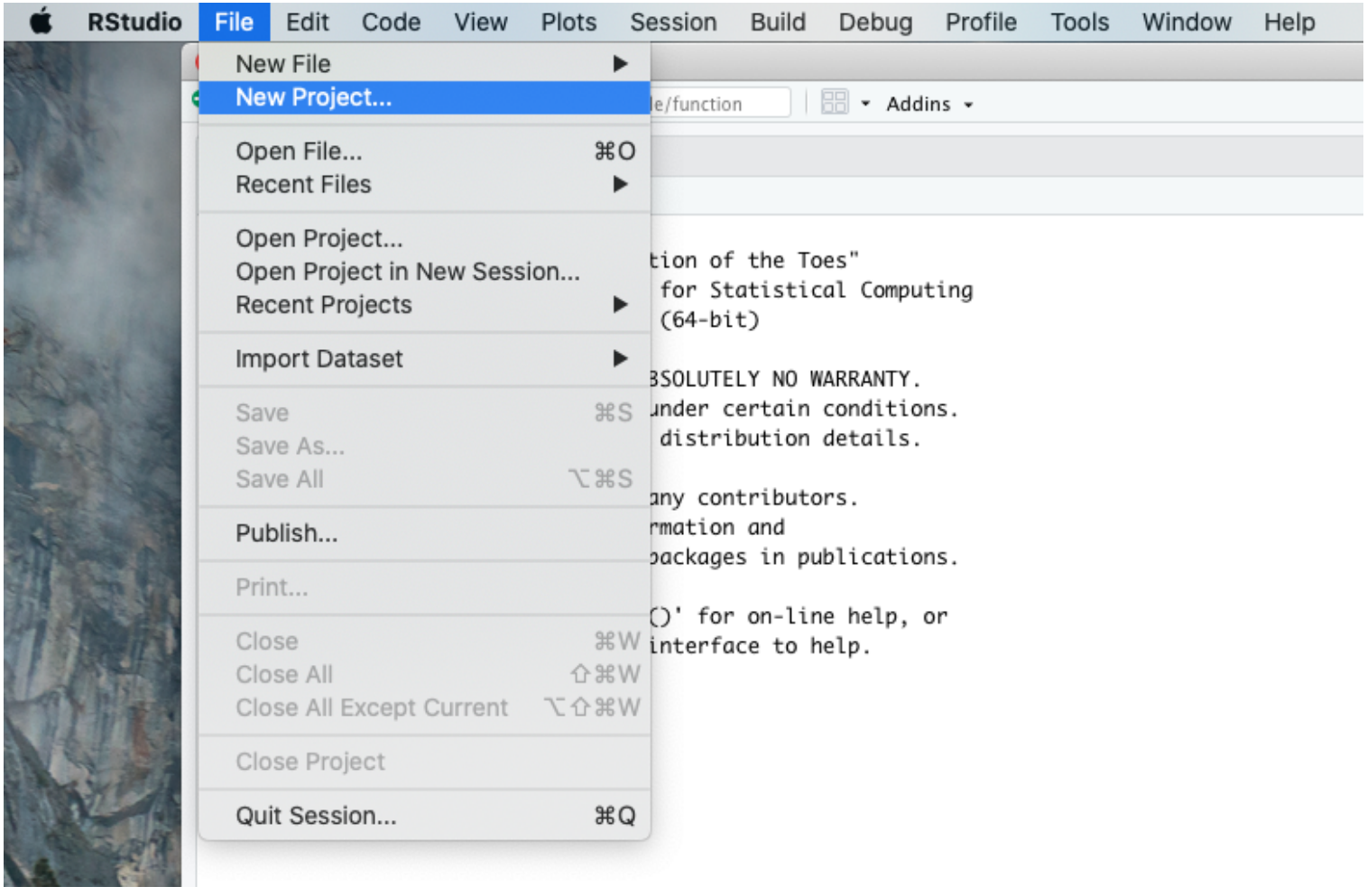
You can also change the working directory in R's console:

```
setwd("/Users/pedro/Documents/intro-to-r")
```

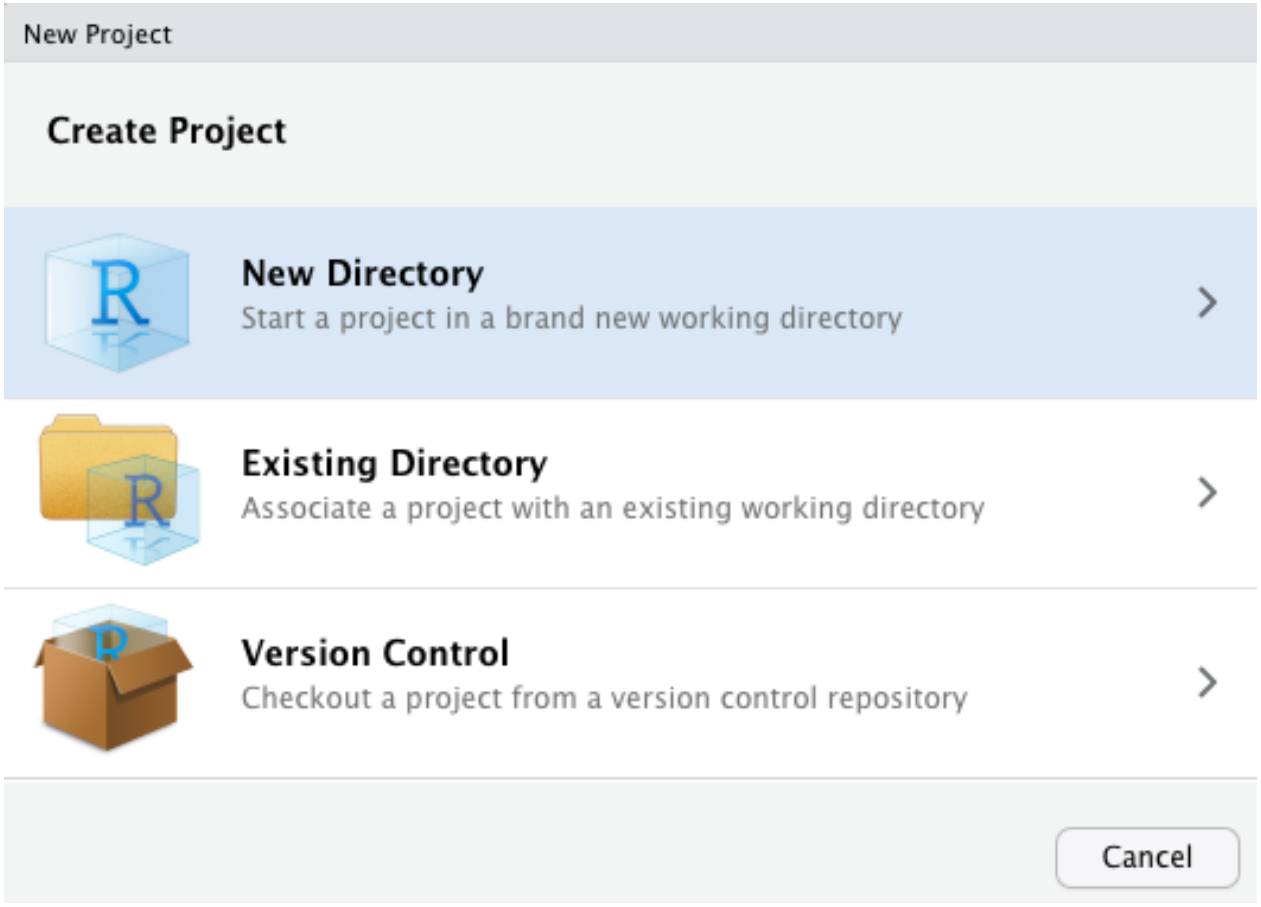
The problem with absolute paths is that they will only exist on your computer. Solution:

- Rstudio projects

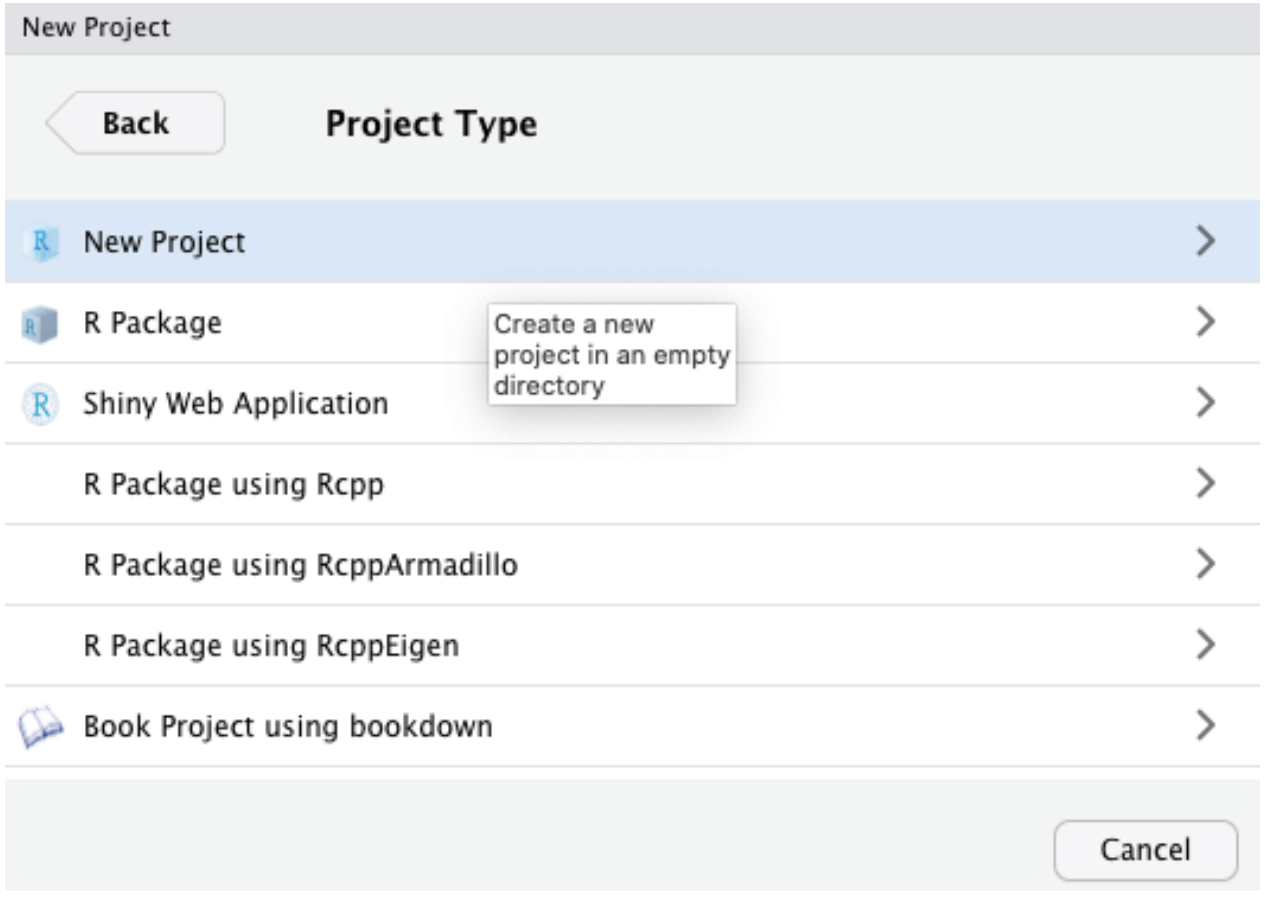
Your first Rstudio project



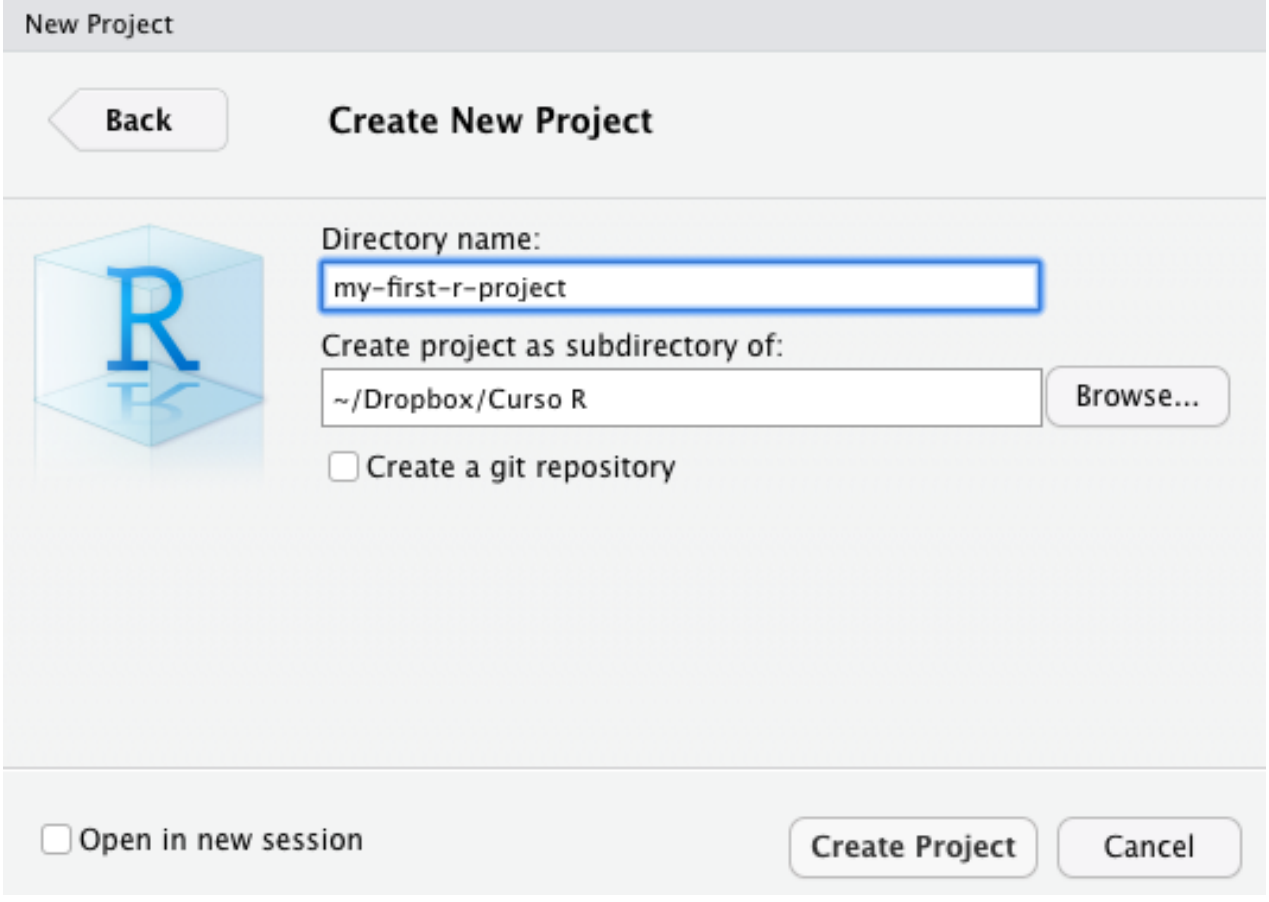
Your first Rstudio project



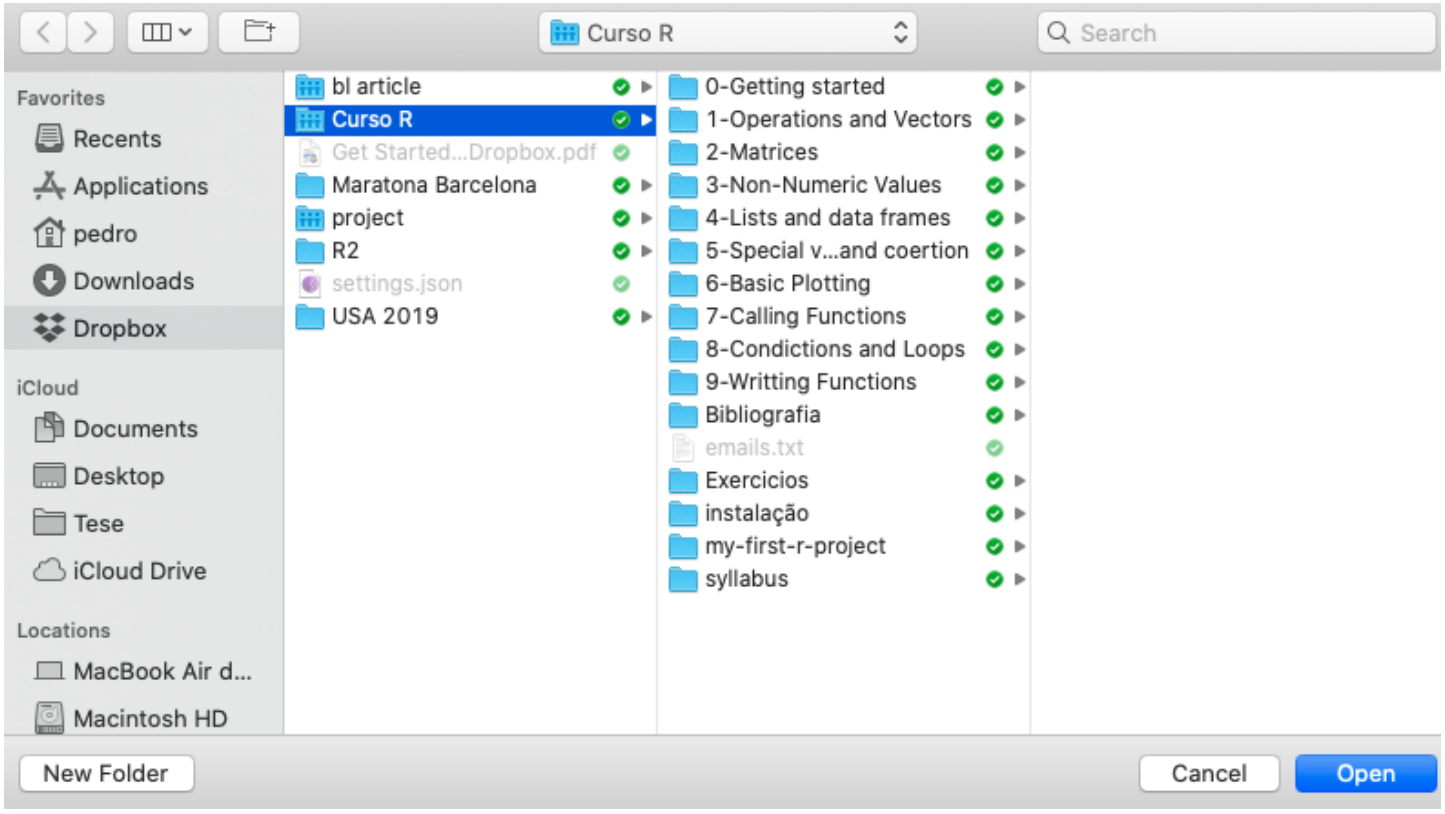
Your first Rstudio project



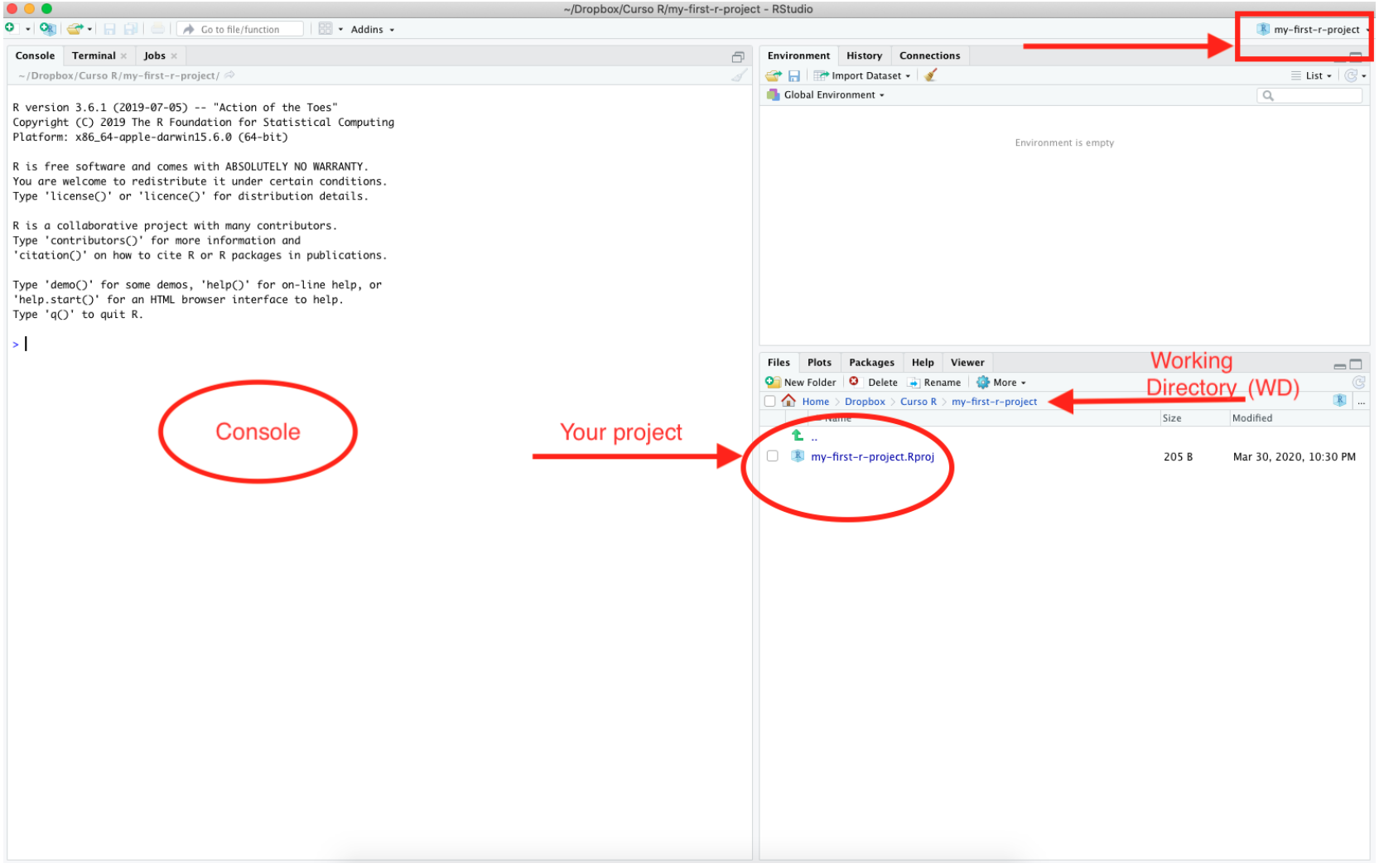
Your first Rstudio project



Your first Rstudio project



Your first Rstudio project



Advantages of Rstudio projects

- Rstudio projects are self-contained.
- They put together all the files that are relevant for a particular project (article, book, research project) in the same folder
- The project's working directory always points to that folder by default
- Rstudio projects can be moved around on your computer or onto other computers and will still "just work". No directory changes are needed.
- If you need to create additional folders or start moving around parts of you project around dont use the `setwd` function. It is safer to reference the full path.

Packages

- The more specialized functions are distributed on packages
- Packages are developed by the R core team and also by the community of R users
- You can develop your own packages and make them available to the community through CRAN (The Comprehensive R Archive Network)

Packages

Later in this course, we will use the `sqldf` package. Let's install it:

```
install.packages("sqldf")
```

If you want to use an installed package, you must load it first:

```
library("sqldf")
```

Update an installed package:

```
update.packages("sqldf")
```

Packages

- It is recommended that you start your scripts by loading the packages that will be used
- That way, if you share your code with others (even if that's future you), they can easily see what packages they need to install

Packages

- Note, however, that you should never include `install.packages` or `setwd` in a script that you share
- Use `library` instead
- It is very antisocial to change settings or install software on someone else's computer!