

# Introduction to R Programming

## Getting Started

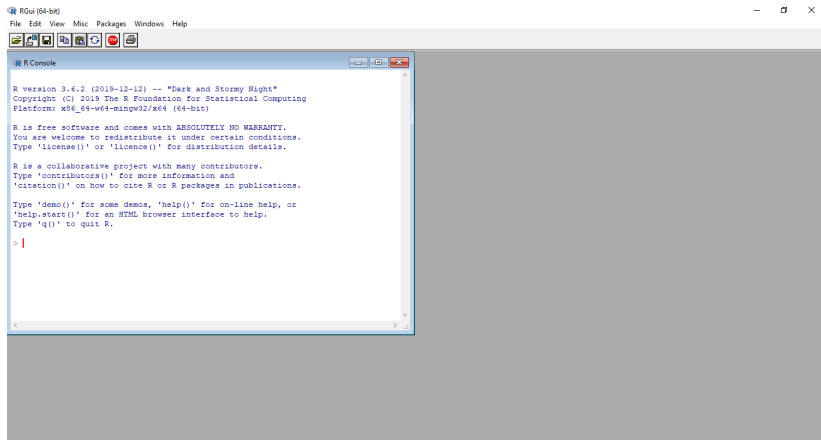
Pedro Fonseca

16 January 2022

# R and Rstudio

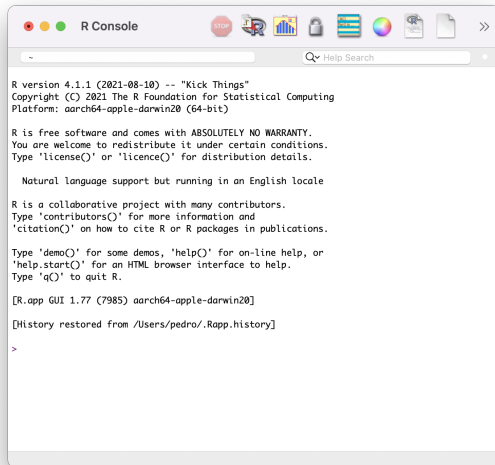
- ▶ R is a programming language and a free and open source software environment for statistical computing and graphics
- ▶ RStudio is an integrated development environment (IDE) for R with free/open source and commercial versions
- ▶ You can use R without using RStudio, but you cannot use Rstudio without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

# This is how R looks like



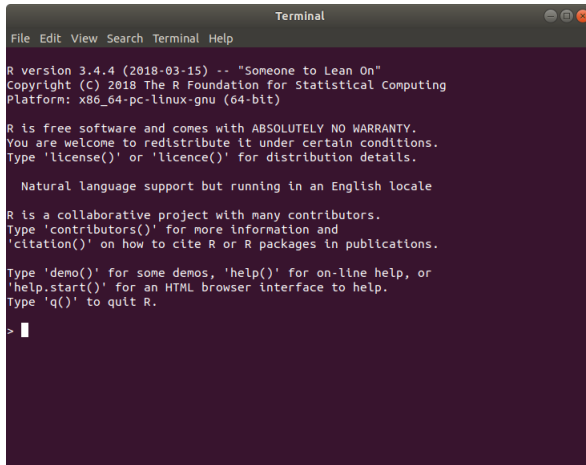
**Figure 1:** R on Windows

# This is how R looks like



**Figure 2:** R on macOS

# This is how R looks like

A screenshot of a terminal window titled "Terminal" with standard Ubuntu window controls. The terminal displays the R version 3.4.4 startup message, including copyright information for 2018, the platform (x86\_64-pc-linux-gnu), and instructions on how to use R, such as typing 'license()' for distribution details or 'demo()' for demos. The prompt is currently ">".

```
Terminal
File Edit View Search Terminal Help

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 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.

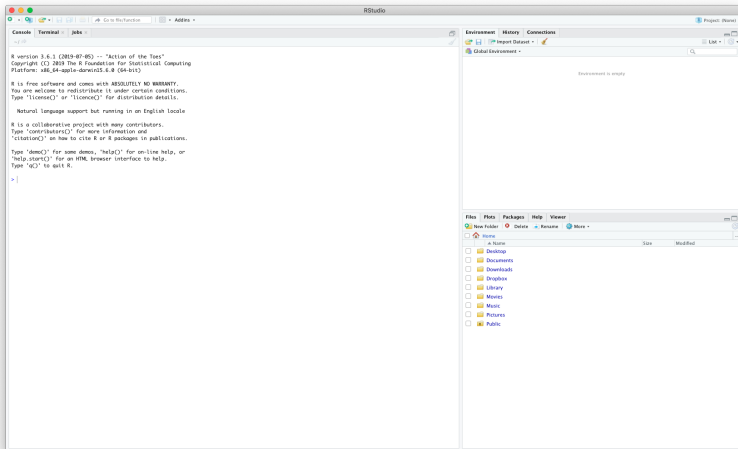
> 
```

**Figure 3:** R on Ubuntu

# Why use RStudio?

- ▶ R is command line driven
- ▶ RStudio provides more a user-friendly and interactive interface to R

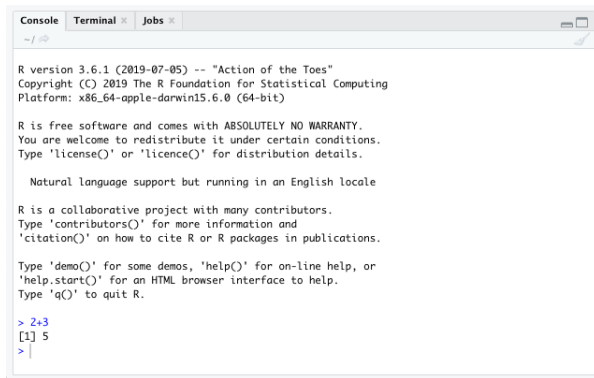
# This is how Rstudio looks like



**Figure 4:** RStudio

# The console

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



```
Console Terminal x Jobs x
~/
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (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.

> 2+3
[1] 5
> |
```

**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 careful 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
```

Common 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 includes 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

The `log()` function 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 = 2)
```

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

# Functions

The `log10()` function only has one argument, `x`, and it is mandatory:

```
log10(x = 243)
```

```
## [1] 2.385606
```

```
log(x = 243, base = 10)
```

```
## [1] 2.385606
```

# Functions

The `round()` function 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

Argument 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 the names of the arguments is safe in functions with only 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 we must pay attention to the ordering of the arguments:

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

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

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

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

# Functions

If we provide the names of the arguments, the ordering is irrelevant:

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

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

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

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



# Functions

It is usually 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
```

# Functions

Help pages can be very useful:

```
?log
```

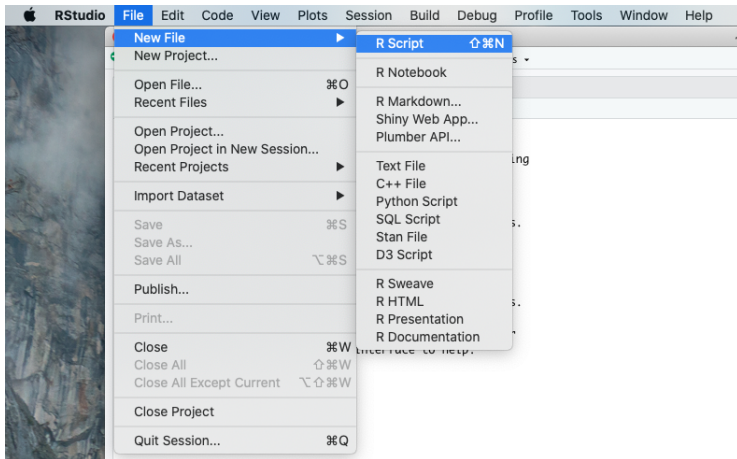
In the help page of a function you can find:

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

# R scripts

- ▶ So far we have only 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 that we can use to write code

# Your first R script

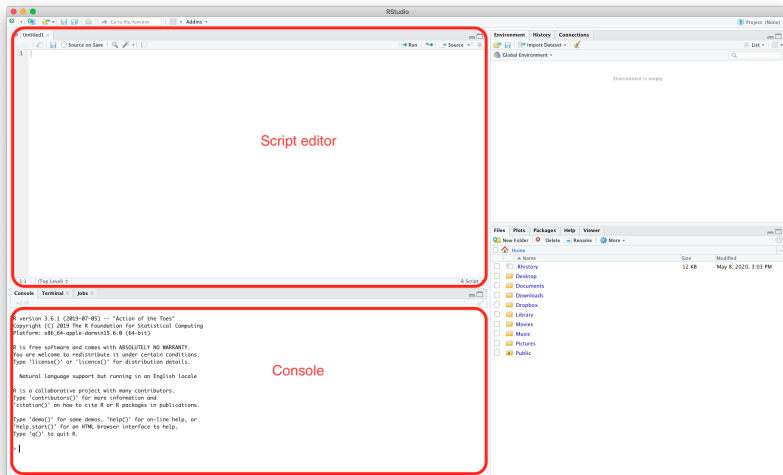


**Figure 6:** Creating a new script

# Editor

- ▶ R opens scripts in the editor pane (top left)
- ▶ This is where you should write your code
- ▶ In the editor you can modify, rerun and save your code at any time
- ▶ Scripts are the holly grail of reproducible data analysis

# 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 assignment operator (<-):

```
x <- 5
```

```
x
```

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



# The assignment operator

- ▶ The assignment operator is a better option
- ▶ The equal sign should be reserved to provide arguments to functions
- ▶ Rstudio's shortcut to the assignment operator is "Alt/Option" + "—"

# The assignment operator



# The assignment operator

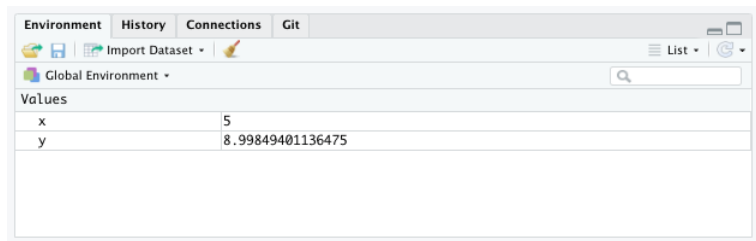
Values stored in objects can be used in calculations:

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

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

# The assignment operator

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



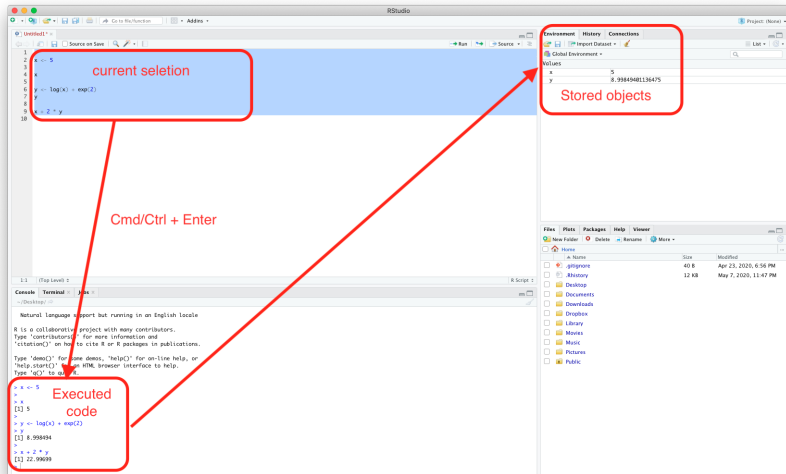
**Figure 8:** Our session's global environment

# Workflow

Our workflow so far:

- ▶ Write code in the editor
- ▶ Send code to the console
- ▶ The code is executed 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

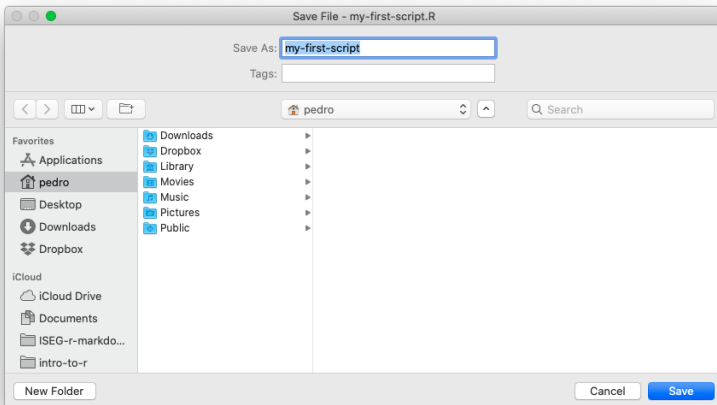
# Now let's print x
x

## [1] 5
```



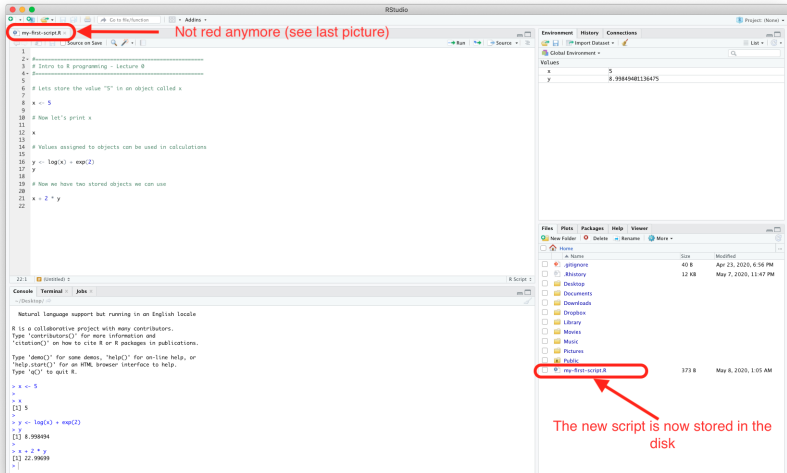
# Saving scripts

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())
```

## Overwriting 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 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)

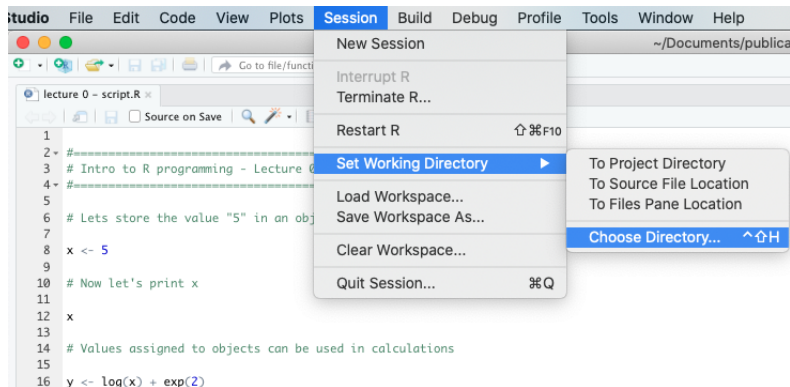
To check your working directory:

```
getwd()
```



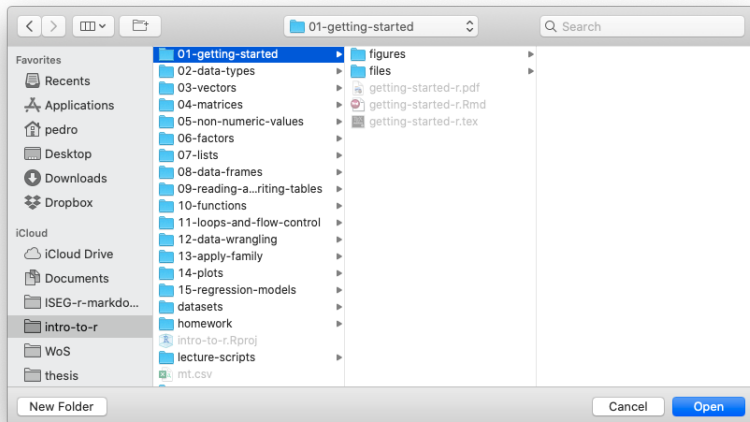
# Setting the working directory

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



**Figure 12:** Setting the working directory

# Setting the working directory



**Figure 13:** Setting the working directory

## Setting the working directory

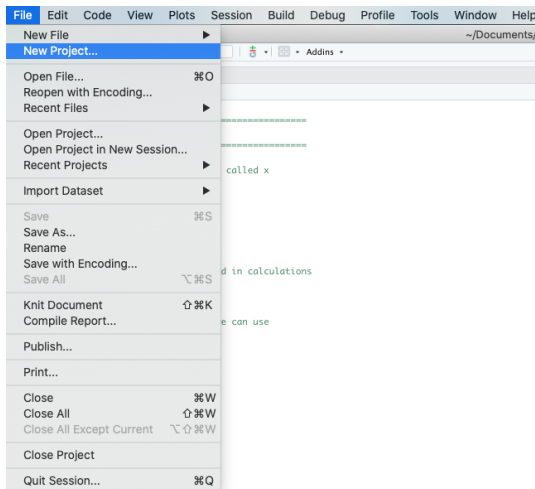
You can also change the working directory in R's console:

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

## Setting the working directory

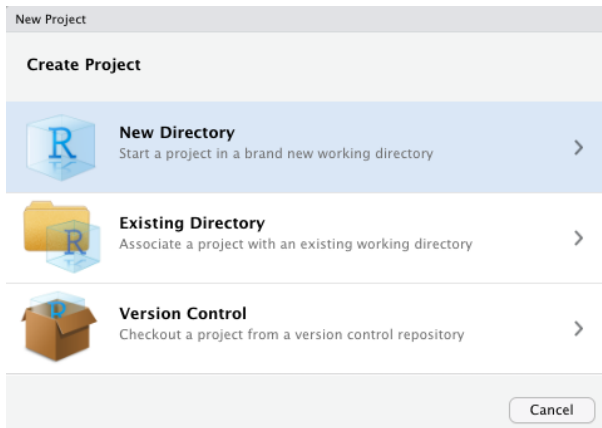
- ▶ The problem with absolute paths like the one in the last slide is that they only exist in my computer
- ▶ This makes it more difficult to share and reproduce scripts
- ▶ Solution: Rstudio projects

# Your first Rstudio project



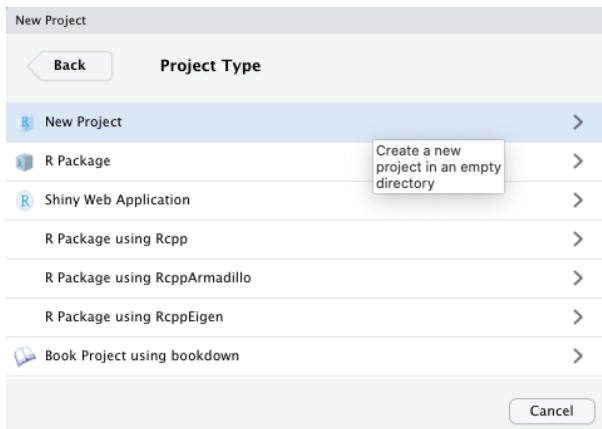
**Figure 14:** Creating a new project

# Your first Rstudio project



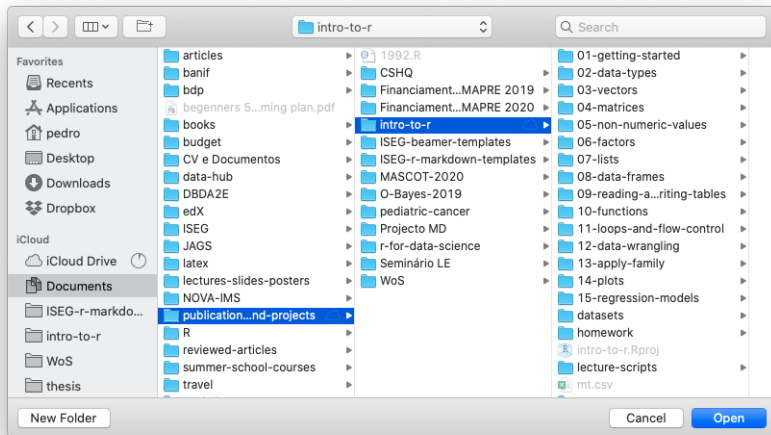
**Figure 15:** Creating a new project

# Your first Rstudio project



**Figure 16:** Creating a new project

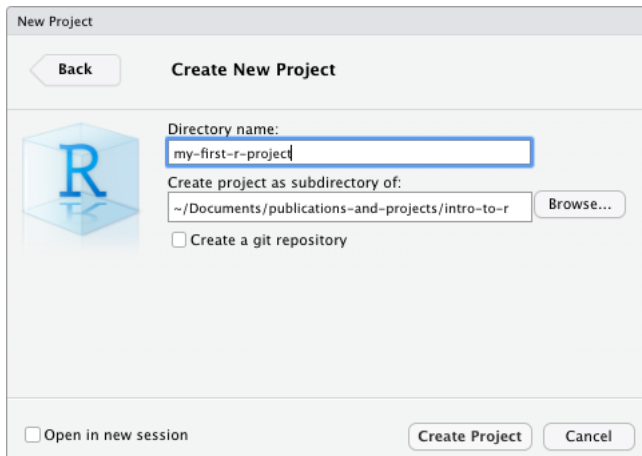
# Your first Rstudio project



**Figure 17:** Creating a new project

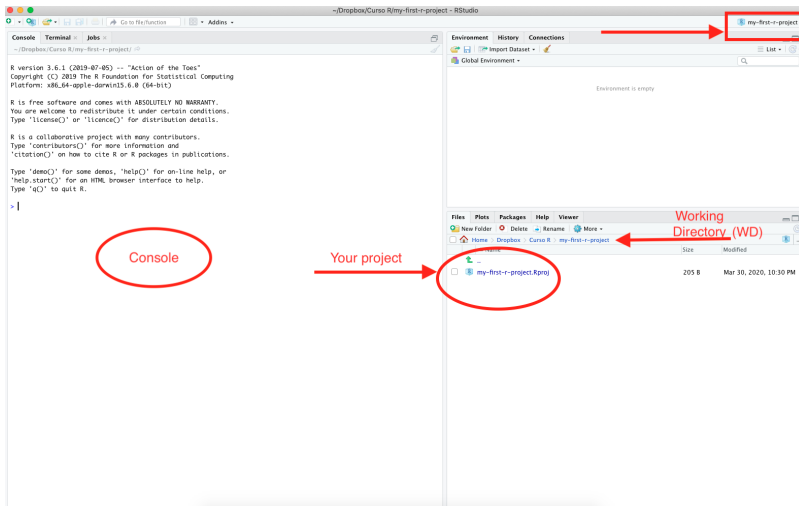


# Your first Rstudio project



**Figure 18:** Creating a new project

# Your first Rstudio project



**Figure 19:** Creating a new 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!