

Introduction to R and RStudio

Data analysis and visualization in R UC Merced

Andrea Sánchez-Tapia

Rio de Janeiro Botanical Garden - ¡liibre! - RLadies+

2021-03-23

outline

- Overview to R and RStudio
- Introduction to R
- Starting with Data
- Exploratory data analysis and basic statistics with R
- Manipulating Data Frames with **dplyr**
- Data visualisation with **ggplot2**

overview of R and RStudio

why learn R?

- the language of choice for academic statisticians
- **libre software**: free and free-to-be-used-and-modified for any means -> one of the pillars of open science
- **script-based**: reproducibility, easier to scale up your analyses, transparency (track errors), great way to learn about methods.
- **interdisciplinary and modular**: lots of code written by area specialists. At the core of its philosophy is a smooth transition from user to programmer.
- **communication** with other tools: manuscripts, presentations, apps and dashboards

why learn R?

- communication with **other programming languages** (ex. **reticulate** to run python scripts)
- great **graphic capabilities!**
- **official support**: help in documentation, mailing lists
- **an active and welcoming community**: email lists, Stack Overflow, RStudio community, useR groups, R-Ladies+ chapters, Slack communities,
`icon::fontawesome("twitter") #rstats`

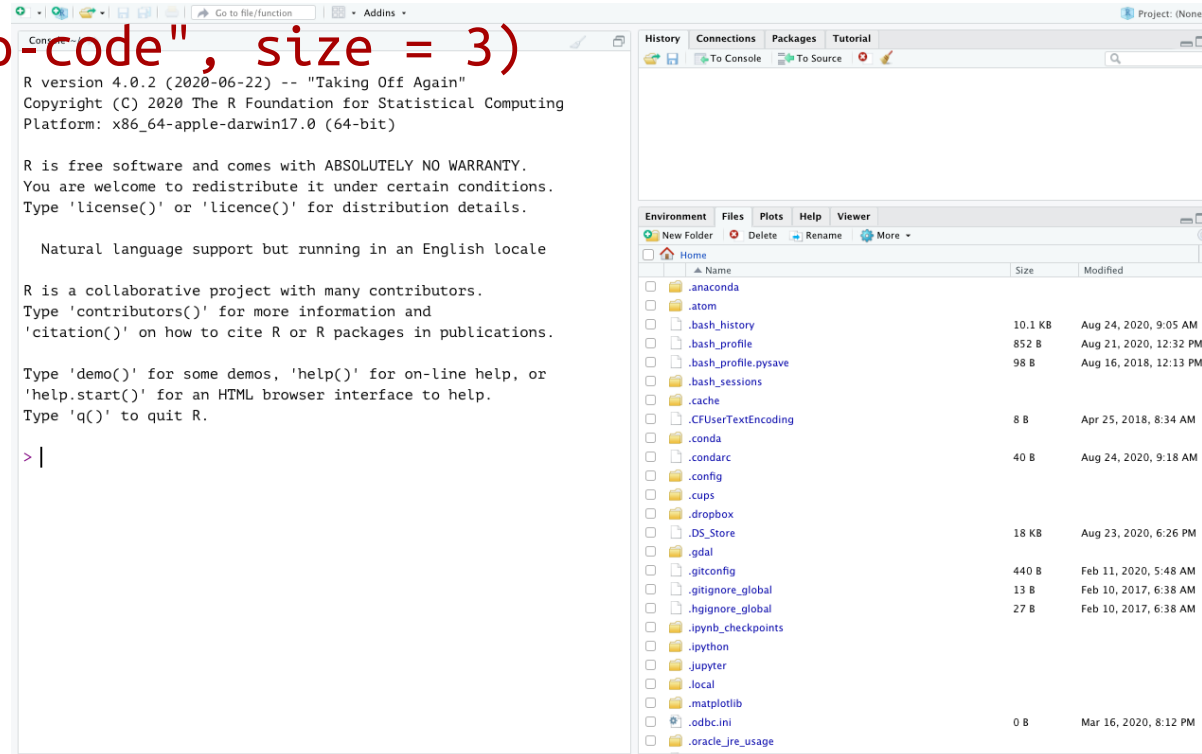


R has a modular structure: packages

- **R base** installation includes base packages developed and maintained by the **R Core Development Team**
- other packages are created by the community and hosted in **CRAN** (The Comprehensive R Archive Network) or Bioconductor, GitHub, rOpenSci.org
- to install packages from CRAN: `install.packages("tidyverse")`

Running R in RStudio

icon::fa("laptop-code", size = 3)



Setup and project organization

working directory

- you have to tell R where you will be working, so that it understands where to read tables, where to save outputs etc: **working directory**
- **getwd()** in the console
- the default is **"home"**: check general options and the "File" tab
- you can tell R and RStudio where you want to work with **setwd()**
- even better: instead of opening RStudio open an **R script** or a **RStudio project** (just as you would in MS Word **`icon::fa("file-word")`**)

project organization and best practices

- projects are better organized if we use **one folder per project** and **subfolders** within our working directory
- we shouldn't modify **raw data files** but **save processed data** (and the corresponding scripts)
- instead of **absolute paths** we should use **relative paths**:
 - **.** "here"
 - **./figs** a subfolder called **figs**
 - the upper level **..**
- avoid **C:users/your_name/your_file_structure/your_working_directory**

In this and the following sessions

```
project/
├── data/
│   ├── raw
│   └── processed
├── docs/
├── figs/
├── R/
├── output/
└── README.md
```

- unzip the .zip file into a folder of your preference

icon :: *fa*(laptop-code, *size* = 3)

RStudio projects

RStudio projects create a **.Rproj** file in your folder that acts as a shortcut for your projects



introduction to R

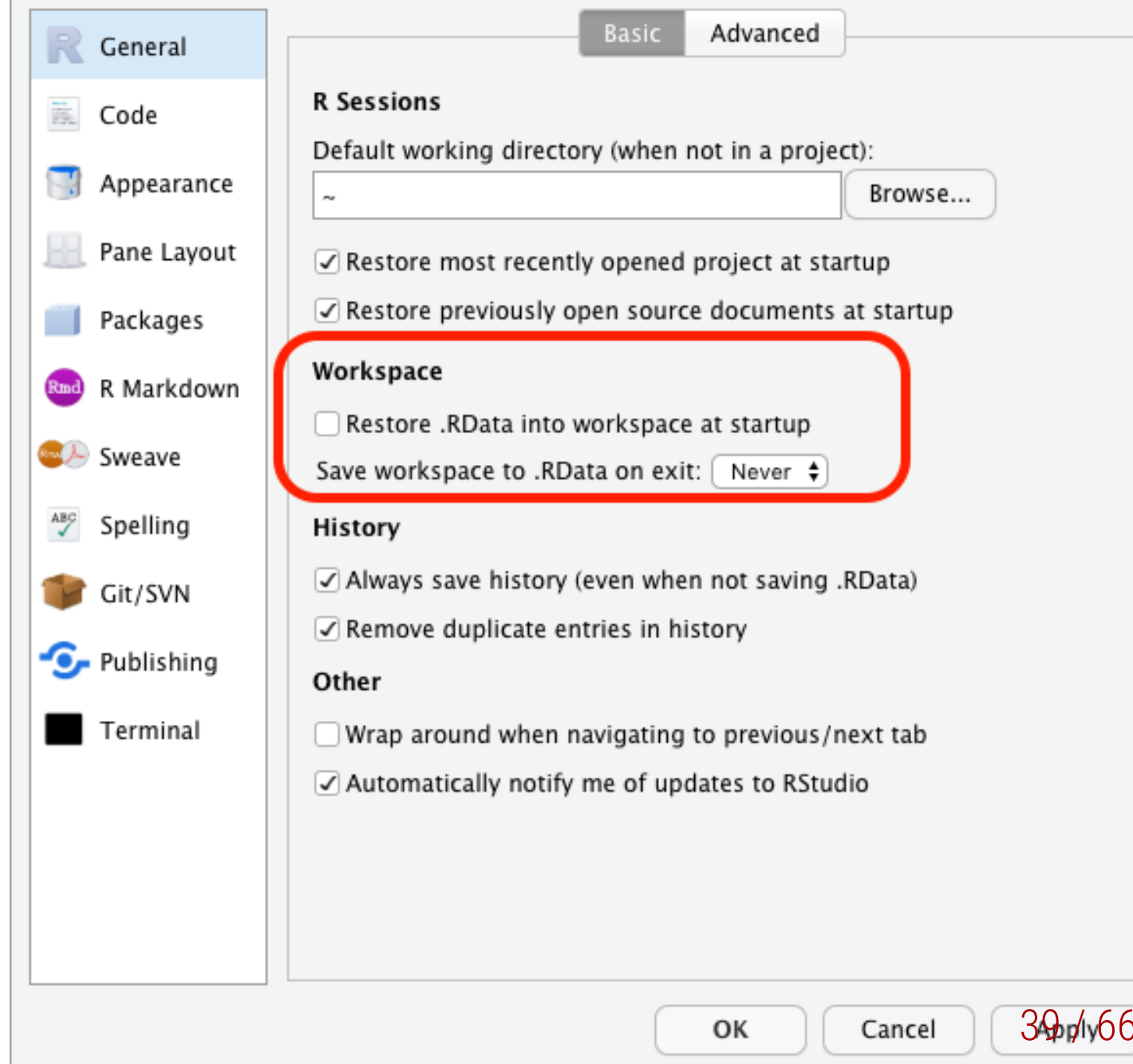
introduction to R

- `<-` is the assignment operation in R and it does not return output it **creates objects** that are saved in the workspace (**Alt + -**)
- overwriting objects **does not affect other objects**
- **naming object tips:**
 - don't begin with a number or symbol.
 - there are forbidden words
 - be consistent with your **coding style!**
 - avoid dots
 - **name functions as verbs and objects as nouns**
- you can see which objects are saved in the workspace by using **ls()**

about the workspace

- R creates **objects** that occupy RAM memory: the **workspace**
- the **workspace** can be saved and loaded between sessions BUT
- **you can lose track of how you created the objects in the workspace**
- **#goodpractices** don't save the workspace

in the general options



functions, arguments and understanding the help

functions and arguments

```
weight_kg <- sqrt(10)
```

```
round(3.14159)
```

```
args(round)
```

if you know the name of the function

```
help(function)  
?function  
args(function)
```

- select the name of the function and click **F1**

Check the structure of the help file:

- Description
- Usage
- Arguments
- Details

if you don't know the name of the function

??kruskal

(or search it - google it - duckduckgo it)

the structure of a function help

`args(function)`

- The arguments of a function are coded:
 - in order
 - with or without default settings

You can either

- use the arguments in order, without naming them
- use the first arguments without naming them and then some optional arguments, with name

data types in R

```
animals <- c("mouse", "rat", "dog")  
weight_g <- c(50, 60, 65, 82)
```

```
class(animals)
```

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

```
class(weight_g)
```

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

character and **numeric** but also **logical** and **integer** ("whole" numbers, with no decimal component, in N), **complex**, and others.

```
icon::fa("laptop-code", size = 3)
```

subsetting vectors

- R is **1-indexed** and intervals are closed (not half-open)

```
animals <- c("mouse", "rat", "dog", "cat")  
animals[2]
```

```
## [1] "rat"
```

- Subsetting is done with brackets **[]**

```
animals[c(3, 2)]
```

```
## [1] "dog" "rat"
```


conditional subsetting

```
weight_g <- c(21, 34, 39, 54, 55)  
weight_g[c(TRUE, FALSE, FALSE, TRUE, TRUE)]
```

```
## [1] 21 54 55
```

Nobody works like this, instead we use **logical clauses** to **generate** these logical vectors

logical clauses

- equality or not: `==`, `!=`
- inequalities: `<`, `>`, `<=`, `>=`
- union (OR) `|`
- intersection (AND) `&`
- belonging `%in%`
- differences between sets: `setdiff()`
- negation works `!:` "not in" `!a %in% b`

comparing vectors

```
animals      <- c("mouse", "rat", "dog", "cat")
more_animals <- c("rat", "cat", "dog", "duck", "goat")

animals %in% more_animals
```

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

comparing vectors

```
animals      <- c("mouse", "rat", "dog", "cat")
more_animals <- c("rat", "cat", "dog", "duck", "goat")

animals == more_animals
```

```
## Warning in animals == more_animals: longer object length is not a multiple
## shorter object length
```

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

- Vectors are compared **one by one AND recycled** when one of them is shorter, so use **%in%** when you want to check **belonging to a set**

missing data

```
heights <- c(2, 4, 4, NA, 6)  
mean(heights)
```

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

```
max(heights)
```

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

```
mean(heights, na.rm = TRUE)
```

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

```
max(heights, na.rm = TRUE)
```

data structures

- **vector**: lineal arrays (one dimension: only length)
- **factors**: vectors (one-dimensional) representing **categorical variables** and thus having **levels**
- **matrices**: arrays of vectors -> the same type (all numeric or all character, for instance) (two dimensions: width and length)
- **data frames**: two-dimensional arrays but might be of combined types (i.e., column 1 with names, column 2 with numbers)
- **arrays** are similar to matrices and dataframes but may be three-dimensional ("layered" data frames)
- **list**: literally a list of anything (a list of data frames, or different objects)

Getting help in R

Other sources of help

- Taskviews

<https://cran.r-project.org/web/views/>

¡Thanks!

`icon::fa("paper-plane", colour = "#562457") andreasancheztapia@gmail.com`

`icon::fa("twitter", colour = "#562457") @SanchezTapiaA`

`icon::fa("github", colour = "#562457") icon::fa("gitlab", colour =
"#562457") icon::fa("bitbucket", colour = "#562457") andreasancheztapia`