R basics for Research

Marcelino Guerra

8/16/2022

Table of contents

Preface

This book is a gentle introduction on how to program in R. Hands-on examples show challenges that researchers frequently face and how to overcome common issues associated to data wrangling and visualization.

1 Setting things up

1.1 Installing R

To install R, go to this link. At the top of the web page, you have three links for downloading R, depending on your operating system. If you are using Windows, follow “Download R for Windows” -> “base.” To install R on a Mac, click “Download R for Mac.” Then, choose the latest release depending on which Mac you have: Intel or Apple silicon.

1.2 Installing RStudio

RStudio is an integrated development environment for R and is highly recommended - it makes using R much more accessible. Download RStudio for free here. Follow the default instructions.

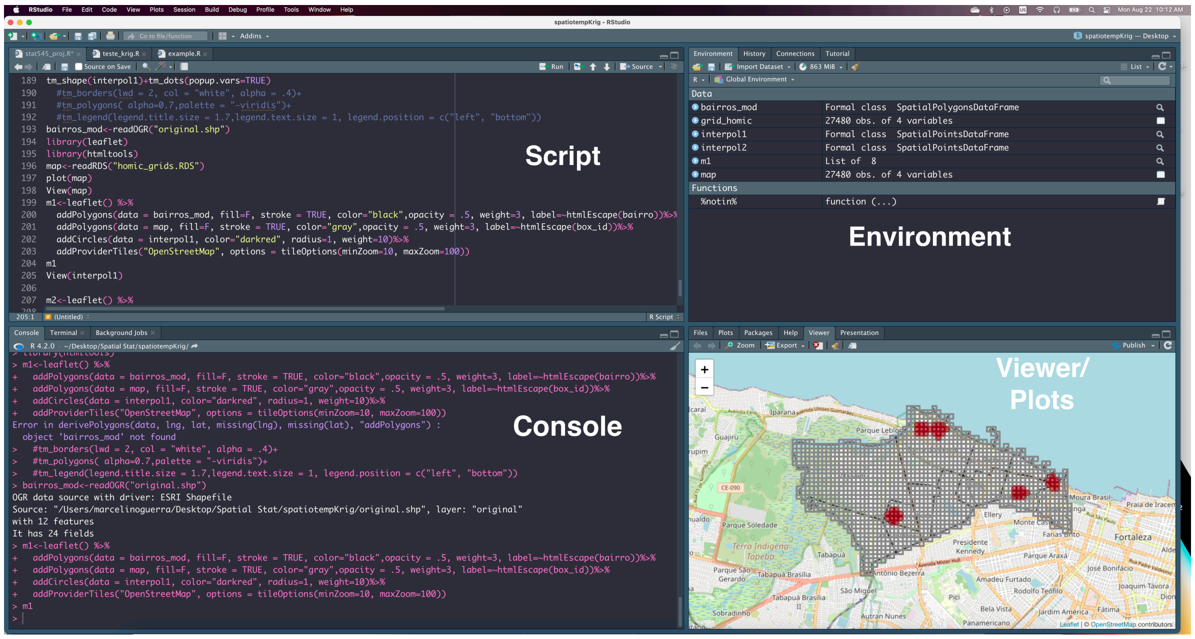
1.3 Rstudio tour

The standard RStudio set-up consists of four panes. On the top left, you have scripts where you write code and save it. On the bottom left, you have the console. The console waits for you to run coding lines, process them, and show the results of what you did (it might also output an error message). The environment shows stored information on the top right and might also report the session’s memory usage. Finally, at the bottom right, you have plots and interactive views.

It is possible to customize your RStudio. For instance, you can change the appearance and pane layout. Go to `Tools -> Global Options -> Appearance` to change font size, and editor theme.

1.4 Creating a Script

To create a **script**, you can either click on the top left icon and then R script or press `Shift + Control/Command + N`. Scripts are handy to save your work and organize tasks. For instance,



```
tm_shape(interpol1)+tm_dots(popup.vars=TRUE)
  #tm_borders(lwd = 2, col = "white", alpha = .4)+
  #tm_polygons( alpha=0.7,palette = "-viridis")+
  #tm_legend(legend.title.size = 1.7,legend.text.size = 1, legend.position = c("left", "bottom"))
bairros_mod<-readOGR("original.shp")
library(leaflet)
library(htmltools)
map<-readRDS("homic_grids.RDS")
plot(map)
View(map)
m1<-leaflet() %>%
  addPolygons(data = bairros_mod, fill=F, stroke = TRUE, color="black",opacity = .5, weight=3, label=~htmlEscape(bairro))%>%
  addPolygons(data = map, fill=F, stroke = TRUE, color="gray",opacity = .5, weight=3, label=~htmlEscape(box_id))%>%
  addCircles(data = interpol1, color="darkred", radius=1, weight=10)%>%
  addProviderTiles("OpenStreetMap", options = tileOptions(minZoom=10, maxZoom=100))
m1
View(interpol1)

m2<-leaflet() %>%
```

```
205:1   (Untitled)                                   R Script
```

Data	
bairros_mod	Formal class SpatialPolygonsDataFrame
grid_homic	27480 obs. of 4 variables
interpol1	Formal class SpatialPointsDataFrame
interpol2	Formal class SpatialPointsDataFrame
m1	List of 8
map	27480 obs. of 4 variables
Functions	
%notin%	function (...)

```
R 4.2.0 · ~/Desktop/Spatial Stat/spatiotempKrig/
> m1<-leaflet() %>%
+   addPolygons(data = bairros_mod, fill=F, stroke = TRUE, color="black",opacity = .5, weight=3, label=~htmlEscape(bairro))%>%
+   addPolygons(data = map, fill=F, stroke = TRUE, color="gray",opacity = .5, weight=3, label=~htmlEscape(box_id))%>%
+   addCircles(data = interpol1, color="darkred", radius=1, weight=10)%>%
+   addProviderTiles("OpenStreetMap", options = tileOptions(minZoom=10, maxZoom=100))
Error in derivePolygons(data, lng, lat, missing(lng), missing(lat), "addPolygons") :
  object 'bairros_mod' not found
>   #tm_borders(lwd = 2, col = "white", alpha = .4)+
>   #tm_polygons( alpha=0.7,palette = "-viridis")+
>   #tm_legend(legend.title.size = 1.7,legend.text.size = 1, legend.position = c("left", "bottom"))
> bairros_mod<-readOGR("original.shp")
OGR data source with driver: ESRI Shapefile
Source: "/Users/marcelinoguerra/Desktop/Spatial Stat/spatiotempKrig/original.shp", layer: "original"
with 12 features
It has 24 fields
> m1<-leaflet() %>%
+   addPolygons(data = bairros_mod, fill=F, stroke = TRUE, color="black",opacity = .5, weight=3, label=~htmlEscape(bairro))%>%
+   addPolygons(data = map, fill=F, stroke = TRUE, color="gray",opacity = .5, weight=3, label=~htmlEscape(box_id))%>%
+   addCircles(data = interpol1, color="darkred", radius=1, weight=10)%>%
+   addProviderTiles("OpenStreetMap", options = tileOptions(minZoom=10, maxZoom=100))
> m1
>
```

Figure 1.1: RStudio

if you have a project that demands data cleaning, visualization, and model estimation, you might want to create different scripts to deal with various tasks.

1.5 Dealing with Packages

Although the base system contains many built-in tools, you will need to install packages to perform many tasks. For instance, to create beautiful visualizations, you might want to use `ggplot2`, included in the `tidyverse` collection of R packages. To run regressions with many fixed effects, I suggest `fixest`. At this point, you know the drill. In the console, let's try to install `tidyverse` typping `install.packages("tidyverse")` (**yes, between quotation marks**).

Once you have the package installed, there is no need to install it again. However, to use the package, you need to call it first using `library()`:

```
#install.packages("tidyverse")
library(tidyverse)
# tidyverse is a collection of super useful packages like ggplot2, dplyr, readr, etc...
```

Another detail here is the use of `#` in the script. If you use `#` before your coding lines, R won't run them. `#` is also helpful if you want to make comments within your code.

1.6 Installing R Markdown

To install Rmarkdown, **write in the console**:

```r
# Install from CRAN
install.packages('rmarkdown')
```

To generate PDF output, you will need to install LaTeX. Your machine might already have MikTeX, but TinyTex is highly recommended. Again in the console:

```r
install.packages('tinytex')
tinytex::install_tinytex()
```

1.7 Creating an R Markdown Document

R Markdown documents are fully reproducible and allow the use of multiple languages (R, Python, SQL). If you are teaching a class that demands to code, R Markdown will make grading much easier since the PDF would display the R coding chunk and the output right after.

To create an R Markdown document, click on the top left and `R Markdown… -> Document`. Then, choose the output format. If you want to know more about R Markdown, check Xie, Allaire, and Grolemund (2018).

1.8 Creating a Project

A Project helps you to organize all the files related to a specific task:

- A paper you are writing.
- A replication you are doing.
- Maybe a homework assignment you have.

In that sense, a Project is a folder where you keep all the scripts, data, tables, and results of whatever the task is. Important to mention that keeping everything in one place avoids trouble with the `working directory`.

On the top left (second icon), you have the option to create a new Project. Then, `New Directory -> New Project`. You need to give it a name and locate it (for instance, on the Desktop). Note that you need to open the Project before opening its scripts.

1.8.1 Exercise

Create an R Project and scripts to use during the workshop.

2 Getting Started with R

2.1 Messing around

Math expressions are generally accepted in R. For instance if you type `2+2` the console will output `4`.

```
2+2
```

```
[1] 4
```

Now, try `-`, `*`, `/`, and `^` (for raising to a power). Besides that, there are many built-in math functions - check some of them here.

What you will mostly do is to create objects. For example:

```
odd<-c(1,3,5,7,9,11)
odd
```

```
[1]  1  3  5  7  9 11
```

`odd` is a vector containing some odd numbers. A few details: `c()` concatenates its arguments (odd numbers from 1 to 11) to form a vector named `odd`. Another way to do it is using `seq()` :

```
odd<-seq(from=1, to=11, by=2)
odd
```

```
[1]  1  3  5  7  9 11
```

You can easily apply functions to objects:

```
mean(odd)
```

```
[1] 6
```

```
sum(odd)
```

```
[1] 36
```

```
max(odd)
```

```
[1] 11
```

```
min(odd)
```

```
[1] 1
```

Logical tests are common when dealing with data and now is a good time to get some practice. Test equality with `==` and inequality with `<=`, `<`, `!=`, `>`, or `>=`.

```
4/2==2 # Is 4 divided by 2 equal to 2?
```

```
[1] TRUE
```

```
2!=3 # Is 2 different than 3?
```

```
[1] TRUE
```

```
2>10/5 # Is two greater than 10 divided by 5?
```

```
[1] FALSE
```

It is very common to check whether something belongs to a group, and `%in%` is very helpful in this case:

```
2 %in% odd # 2 does not belongs to odd
```

```
[1] FALSE
```

Finally, we need to talk about `&` (and) and `|` (or):

```
2|3 %in% odd # does 2 or 3 belong to odd?
```

```
[1] TRUE
```

```
2 & 3 %in% odd # does 2 and 3 belong to odd?
```

```
[1] TRUE
```

Throughout your research, you will constantly work with strings. Any value written within a pair of single or double quotes in R is treated as a string. Below you have stored `Hi` and `Marcelino`.

```
hi<-"Hi"
name<-"Marcelino"
```

The function `paste()` puts things together with any separator:

```
paste(hi, name, sep=" ") # separating strings with space
```

```
[1] "Hi Marcelino"
```

Another useful function is `sample()`. It takes a sample of the specified `size` from the elements of a vector using either `with` or `without replacement`. Before using `sample()`, to make sure we get the same results, lets start the code chunk with `set.seed(123)`.

```
set.seed(123)
numbers<-seq(1:1000)
sample(numbers, size=2, replace = TRUE) # a random sample size 2 of numbers from 1 to 1000
```

```
[1] 415 463
```

```
sample(numbers, size=10, replace=FALSE) # a random sample size 10 of numbers from 1 to 100
```

```
 [1] 179 526 195 938 818 118 299 229 244  14
```

You can also use a sample with strings:

```
fruits<-c("apple", "orange", "lime")
sample(fruits, size=2) ## replace is False by default
```

```
[1] "orange" "apple"
```

2.1.1 Exercises

1. Use `sample()` to simulate a fair coin toss 6 and 1,000 times. Does it look like a fair coin?

Hint: create a vector c("H", "T") and use ***`sample()`*** *with different sizes. Should you use replace = False or replace = True?*

2. Let's play dice! When you roll a fair die, you expect to get 1,2,3,4,5, and 6 with the same probability $\frac{1}{6}$. Hence, the mathematical expectation of that process is:

$$E[X] = \frac{1}{6}[1 + 2 + 3 + 4 + 5 + 6] = 3.5$$

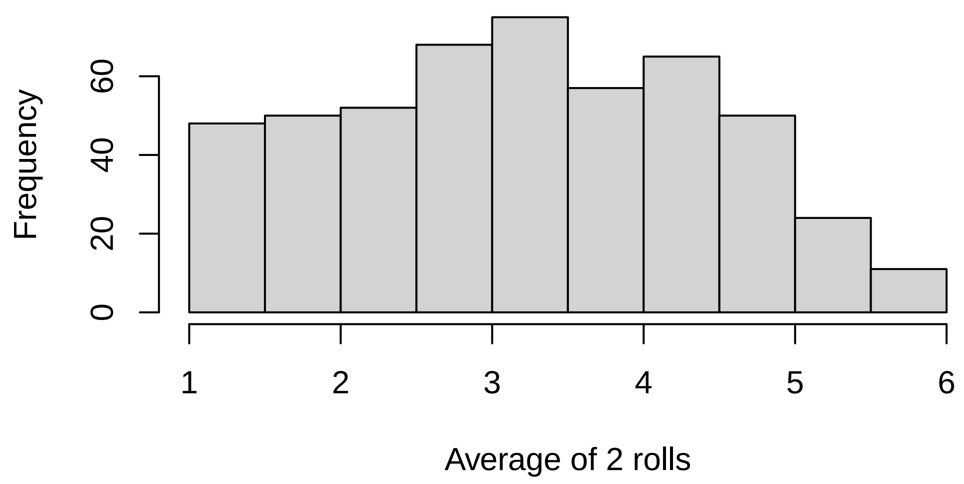
According to the Law of Large Numbers, if you play long enough, the sample average will get close to 3.5. Now, let's look at the histograms for each set of averages according to the sample size (number of rolls). We start rolling a dice two times, and we repeat this process 500 times. The resulting distribution is called "the sampling distribution of the sample mean."

```
roll<-sample(1:6, size=2, replace=TRUE)
mean(roll)
```

```
[1] 2.5
```

Using the function `replicate()` to repeat this process 500 times and plotting the histogram using `hist()`:

```
hist(replicate(500,mean(sample(1:6, size=2, replace=TRUE ))), main=" ", xlab = "Average of
```



Your turn! Roll dice **100 times** and repeat the process again, plotting the histogram. Does this distribution look familiar?

2.2 Data with R

TBD

3 Data Wrangling with Tidyverse

TBD

References

Xie, Yihui, Joseph J Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Chapman; Hall/CRC.