RStudio - 01

Jean-Yves Sgro

May 2, 2017

Contents

1 First Steps with RStudio

RStudio (RStudio Team 2015) is an integrated graphical interface layer over the `R` program (R Core Team 2017).

When RStudio is launched, an `R` session is started and shown within the bottom quadrant of the RStdusio window.

Typically an RStudio session splits the screen in 4 main quadrants. Some quadrants may be split into Tabs:

- Left Top: Scripts
- Left Bottom: `R` console
- Right Top: Environment | History Tabs.
- Right Bottom: Files | Plots | Packages | Help | Viewer

Table 1: **The Quadrants in RStudio**

	Left	Right
Top	Scripts	`Environment` | `History`
Bottom	`R console` | `R Markdown`	`Files` | `Plots` | `Packages` | `Help` | `Viewer`

In the the Script area different file types can be open, the most common would be an `R Script` for recording commands that can be passed onto the `R` console easily.

2 Left - Bottom Quadrant

2.1 R Console

This is the `R` console that is "attached" to this RStudio session and that will execute the scripts with send.

The commands can be written within a script file (see below) or typed directly within the `R` console.

3 Left - Top Quadrant

3.1 RScript

You can start a new `R Script` file with the following menu cascade:

File > New File > R Script

Within the script area we can type commands, one per line.

For example type the following within the 'R Script: area:

```
print('Hello World')
```

Then, while the cursor is within that line, **press together** `control` and `return` (on a Mac `command return` *also* works.)

This action will transfer the command to `R` which will run it. Therefore within the `R` console you will see:

```
> print('Hello World')
[1] "Hello World"
```

i.e. the command and its output

An alternate method to activate the command (and passing it on to `R`) is to click the `Run` button at the top right of the Script quadrant.

In both cases, to activates **multiple lines** simply select them with the mouse first before pressing `control` and `return` or clicking on `Run`.

3.2 Comments

When writing code it is useful to document what the commands mean or what we are trying to do. A "gift" of information to your "future self" or colleagues.

The symbol **#** can be used to write comment lines that are ignored by `R` but that might prove useful in the future. For example:

```
# Example:
# This is an example of a line of code to print the words
# that are between the quotes. The words will be printed
# on the screen for all to see!
# Blank lines wihtin the code do not matter.
# These comment lines are not printed, they are here to
# remind me what the purpose of this comand is.

print('Hello World')

# The line above will execute, and then I can continue with
# more commands and make the world a better place...
```

3.3 R markdown

This type of script will be seen in a whole section later on.

4 Right - Top Quadrant

4.1 History

Now that we have issued at least one command, it will be recorded in the `history` list. You can see the list of issued command by pressing the `History` tab at the Top Right quadrant.

4.2 Environment

The environment variable will record and update all the `R objects` that we create along with their type, size and number of records; in other word the object *structure*.

For example, let's create a vector `x` containing a series of 100 ramdom numbers:

```
x <- rnorm(100)
```

Note: The function `rnorm()` samples random numbers from the `Normal distribution` with mean **zero** (μ=0) and standard deviation of **one** ($\sigma = 1$) if not otherwise specified.

If you now click on the `Environment` Tab at the top right you will see information about `x` that would be identical or similar to that obtained within the console with the `str` command to show the *structure* of an object, for example with the command:

```
str(x)
```

```
 num [1:100] 1.359 0.382 0.183 0.897 1.774 ...
```

5 Right - Bottom Quadrant

The tabs are: **Files** | **Plots** | **Packages** | **Help** | **Viewer**

5.1 Files

The `Files` tab displays the content of the current working directory. If we have not yet specified an area to save file, the typical default would be the default "Home" directory such as `/Users/name` on a Mac or Linux system or `C:\Users\yourname` on Windows.

The list of files and folders shown within the window could also be obtained with the command:

```
dir()
```

5.2 Plots

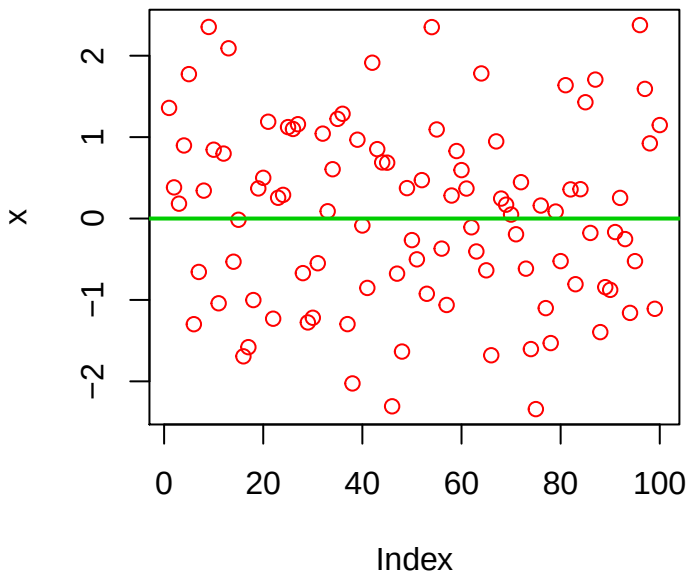
The `Plots` tab should be blank as we have not made a single plot yet.

We can create a simple plot from the 100 random numbers stored within the `x` object created above with the `plot()` function:

```
# This plot not shown
plot(x)
```

We can add a little fantasy by coloring the plotted points and by adding a horizontal line at zero (since the random numbers are ditributed around zero,) thickened with the linewidth command `lwd=2` and colored.

```
plot(x, col=2)
abline(h=0, col=3, lwd=2)
```



If you have not clicked the `Plots` tab yet do so now and you'll see the plot.

This plot will remain visible until another plot command is issued.

Note: Here the color was given by a number where `col=2` made *red* points and `col=3` made a *green* line. There are only `8` colors in this default which can be listed with:

```
palette()
```

```
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

R does have a longer list of 657 colors which can be listed exhaustively with:

```
colors()
```

The first 8 colors in the list are:

```
[1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
[5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
```

The last 8 colors in the list are:

```
[1] "wheat4"      "whitesmoke"  "yellow"      "yellow1"     "yellow2"
[6] "yellow3"     "yellow4"     "yellowgreen"
```

5.3 Packages

The RStudio tab makes installing new packages very easy.

R packages are hosted on the "The Comprehensive R Archive Network"or CRAN web site and its mirrors. Packages provide new functionalities to R and can "depend" on other packages which will be installed together.

We will now install a package that we'll use a little later, and we may add new packages as we go along.

There are currently about 7,000 packages available for R which is great but can also create challenges.

We will install the **knitr** (Xie 2015) package:

1. Click the Packages tab.
2. Click Install button just below
3. Enter **knitr** within the "Packages" text entry
4. Do not change any of the other options.
5. Press the Install button.

The equivallent R command:

```
install.packages("knitr")
```

will be issued within the R console and the package, as well as all its dependencies will be installed.

We will make use of this functionbality of this package in another section.

5.4 Help

The Help tab will present R help when information on commands are requeted. For example:

```
help(print)

# which can also be written as:

?print
```

The help pages are presented from the HTML formatted help pages.

5.5 Viewer

This tab is used to present special output in web format from more advanced RStudio features.

We will not use this tab today.

6 Data and Data frames

Data is generically often avaiable as a table. For example a spreadsheet table.

Dataframes are useful `R` tables that can contain *mixed data* (numbers, words, vectors, etc.)

Dataframes are a "higher order" structure "above" a simple vector or a matrix. However, it is possible to "coerce" these "lower" structures into a data frame for output purposes.

This will be useful later on...

6.1 Simple vector data

Let's start with a simple, small vector example: a vector of 5 random numbers

```
# a vector of random numbers
x2 <- rnorm(5)
```

There are default oputput presentations to `print` these objects onto the screen:

```
# print x2
x2
```

```
[1]  0.5316320 -0.7866012 -2.0469383 -0.1883031  0.4753727
```

We can use the function `as.data.frame()` to "coerce" vector `x2` (later we'll apply this to matrix `z`.)

```
# x2 as dataframe:
as.data.frame(x2)
```

```
          x2
1  0.5316320
2 -0.7866012
3 -2.0469383
4 -0.1883031
5  0.4753727
```

The numbers `1` throuh `5` on the left columns are the `rownames` or `index` of the values of `x2`, *i.e.* the order in which they appear in `x2`:

```
rownames(as.data.frame(x2))
```

```
[1] "1" "2" "3" "4" "5"
```

6.2 Coerce into table format

Let's say that now we want to create a table where we show the value of `x` in one column and the value of `x * 100` and `x * 1000` in adjacent columns. We can use the `cbind()` function that "binds columns" to accomplish that:

```
as.data.frame(cbind(x2, x2*100, x2*1000))
```

```
          x2         V2         V3
1  0.5316320   53.16320   531.6320
2 -0.7866012  -78.66012  -786.6012
3 -2.0469383 -204.69383 -2046.9383
4 -0.1883031  -18.83031  -188.3031
5  0.4753727   47.53727   475.3727
```

However, note that the column names is correct for `x2` but for the other 2 columns `R` simply give a generic vector name: `V2` and `V3` in this case.

Here is a simple method to remedy that: create a new `R` object `Y` containing the new data frame and then change the name of the column names:

```
Y <- as.data.frame(cbind(x2, x2*100, x2*1000))
```

Then change the value of the column names:

```
colnames(Y) <- c("x2","x2 times 100", "x2 times 1000")
```

Now print-out `Y` with its new column names:

```
Y
```

```
          x2 x2 times 100 x2 times 1000
1  0.5316320     53.16320      531.6320
2 -0.7866012    -78.66012     -786.6012
3 -2.0469383   -204.69383    -2046.9383
4 -0.1883031    -18.83031     -188.3031
5  0.4753727     47.53727      475.3727
```

6.3 Apply method to matrix

A similar method can be applied to a matrix:

```
# a matrix of numbers with 4 rows and 5 colums
# containing numbers 1 through 20
z <- matrix(1:20, 4,5)
```

```
# print z
z
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

```
# z as dataframe:
as.data.frame(z)
```

```
  V1 V2 V3 V4 V5
1  1  5  9 13 17
2  2  6 10 14 18
3  3  7 11 15 19
4  4  8 12 16 20
```

We note that the column names again have the generic names of “V1” to “V5”.

With the same method this could be changed with our own column and row names. We’ll first convert matrix `z` into data frame `Z` and then change the row and column names:

```r
# Create an object to contain the dataframe
Z <- as.data.frame(z)

# change the column names:
colnames(Z) <- c("col1", "col2", "col3", "col4", "col5")

# change the row names:
row.names(Z) <- c("row1", "row2", "row3","row4")

# Print the "new Z" with its new column and row names:
print(Z)
```

```
     col1 col2 col3 col4 col5
row1    1    5    9   13   17
row2    2    6   10   14   18
row3    3    7   11   15   19
row4    4    8   12   16   20
```

Note: we created the column names as a vector with the combine `c()` function "on the fly" but the vectors could be created first as an `R` object separately and then assigned to `Z`.

Note also that the command is different for rows and columns, one of them has a `.` in the middle:

- change column name: `colnames()`
- change row names: `row.names()`

This is part of the "fun difficulties" of working with `R` !

6.4 Tables output

Earlier we installed package `knitr` so now we can use a function of `knitr` called `kable()` to format tabular data into a nice output:

```r
# First we load the knitr package
library(knitr)
```

6.4.1 Output `Z` as a table

Now we can write a table example with *e.g.* the matrix `z` transformed earlier in dataframe `Z` with updated row and column names:

```r
kable(Z, padding = 0)
```

	col1	col2	col3	col4	col5
row1	1	5	9	13	17
row2	2	6	10	14	18
row3	3	7	11	15	19
row4	4	8	12	16	20

View options for `kable()` with the help command `?kable` which would explain the following, different output from command:

Note that the output will only make sense for the type of format requested for final output. The HTML format will appear correct only on this document saved as HTML and will appear bady formatted on PDF or

DOCX versions of this document.

```
kable(Z, format="html", caption = "Table 1: This table was a matrix.", padding = 10)
```

Table 1: This table was a matrix.

col1

col2

col3

col4

col5

row1

1

5

9

13

17

row2

2

6

10

14

18

row3

3

7

11

15

19

row4

4

8

12

16

20

Below the "latex" format will appear only on PDF output and will remain blank on other outputs:

```
kable(Z, format="latex")
```

	col1	col2	col3	col4	col5
row1	1	5	9	13	17
row2	2	6	10	14	18
row3	3	7	11	15	19
row4	4	8	12	16	20

6.4.2 Output `Y` as a table:

`Y` is a 3-column dataframe representing `x2`, `100` times `x2` and `1000` times `x2` that we created earlier. We can limit the number of decimal digits shown with the extra command `digit=2` as an option:

```
kable(Y, digit=2)
```

Table 3: The values of `x2` within `Y`

x2	x2 times 100	x2 times 1000
0.53	53.16	531.63
-0.79	-78.66	-786.60
-2.05	-204.69	-2046.94
-0.19	-18.83	-188.30
0.48	47.54	475.37

You can try adding `padding=0` or `padding=1` to see the effect on the output.

In the next section we will refine our use of RStudio by creating an "dynamic" document.

REFERENCES

R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.

RStudio Team. 2015. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc. http://www.rstudio.com/.

Xie, Yihui. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. http://yihui.name/knitr/.