Class to introduce RStudio & R
Luc Hens, 14 February 2019

[Tip: If you get stuck because R keeps expecting more input (showing the + sign instead of a prompt), press Escape]

Background

- R is a statistical programming environment.
- R is widely by statisticians and data scientists (e.g. to make the diagrams used by the BBC)
- R is, together with Python, the most used tool by data scientists.
- R is free, open source.

RStudio puts a much nicer user interface on top of R.

RStudio & R are installed on all computers in the computer labs of B building, ground floor (level 0)

Getting started

Start RStudio by double-clicking the RStudio icon. The standard startup screen looks as follows:
- left pane: Console
- upper right pane with tabs: Environment | History | Connections
- lower right pane with tabs: Files | Plots | Packages | Help | Viewer

(check that most students have this screen; if some students don't, do not waste time trying to fix it. Those students should sit next to someone who has RStudio running)

The left pane (Console) is where you directly interact with R.
Shows a prompt (>) and a blinking cursor: R is waiting for your input.
After the prompt, type:

```
2+3
```

and press return. The Console window shows the result, preceded by [1]:

```
[1] 5
```

The [1] shows that this is the first line of the output (if the output is long, it runs over more than 1 line: [1], [2], ...)

The arithmetic operators are: +, −, ∗ (times), /, ^ (to the power). Try

```
3^2
```

Objects

R works with "objects". Values are assigned to objects using the assignment operator <- (first type <, then type a minus sign -). <- is read as: ... is assigned the value...
For instance, to define an object called my.weight.in.kg, and assign it the value 64 :

```
my.weight.in.kg <- 64
```

and press Return.

Names of objects in R cannot contain blank spaces. Use a period (.) instead.
The object and its value have appeared in the Environment pane (top right).

You can display the value of an object by typing its name in the Console and pressing Return:

```
my.weight.in.kg
```

The Console window returns:

```
[1] 64
```

Objects can also be strings (enclosed in quotes " "):

```
my.name <- "Freddy"
```

To display the value of the object my.name type

```
my.name
```

and press return.

An object can also be a list. Take the frequency table for the survey on social media (table 2.2 p. 53) as an example. The categorical variable Social Media had three categories: "No" , "Yes", "No Access".

We can create an object called category to store the list of values:

```
category <- c("No", "Yes", "No Access")
```

Enclose strings in quotes (""). The concatenate function c() creates lists. To display the value of the object "category" type its name and press return:

```
category
```

Now create a list with the counts that corresponded to the categories:

```
count  <- c(1249, 2175, 1615 )
```

The order matters!

We can do computations with objects:

```
sum(count)
```

Check what happens if you try

```
sum(category)
```

R has built-in documentation with help for each of its functions. Just type a question mark followed by the name of the function:

```
? sum
```

The explanation appears in the Help tab in the lower right pane.

Plots

Let us make a bar chart of the frequency table (with the heights of the bars showing the counts and the labels of the bars showing the categories). The function is barplot(). The function has two arguments: what you want to put on the vertical axis, that is, what determines the heights of the bars (in our case: count), and the list of labels you want to put below the bars, identified with the argument names.arg=... :

```
barplot(count,names.arg=category)
```

The plot appears in the Plots tab in the lower right pane.

Click on Zoom to zoom the plot. Click on Export to export the plot (pdf is a convenient format; save the plot to your Statistics folder on your hard drive).

The labels on the vertical axis are not oriented in a reader-friendly way. Let us change that. No need to retype the code. In the upper right pane go to the History tab and select the line

```
barplot(count,names.arg=category)
```

Then click on the green arrow pointing to the left that says "To Console". The line of code re-appears in the Console after the prompt. Add the option las=1:

```
barplot(count,names.arg=category, las=1)
```

and press Return. The new plot appears in the bottom right pane.

You can use the left-right arrow in the Plots pane to go back to previous plots.

Change the color of the bars:

```
        barplot(count,names.arg=category, las=1, col="LightBlue")
```

Do you want to know which colors you can pick? Type:

```
        colors()
```

Let us clear the Console window. In the RStudio menu select: Edit > Clear Console

Add labels to the x-axis and y-axis:

```
        barplot(count,names.arg=category, las=1, col="LightBlue",
xlab="Social Media", ylab="# of Respondents")
```

Find the relative frequencies expressed as a fraction:

```
        count/sum(count)
```

Store the relative frequencies, expressed as a percentage, in a new object:

```
        relative.frequency <- c(100*count/sum(count))
```

Display the values:

```
        relative.frequency
```

Make a bar chart of the relative frequencies:

```
    barplot(relative.frequency,names.arg=category, las=1,
col="LightBlue", xlab="Social Media", ylab="% of Respondents")
```

Scripts
=======
It is often a good idea to save code in a script file that you can re-use later. In the RStudio menu, select File > New File > R Script On the top left a new pane appears, with a tab Untitled1. Go to that tab, and move the code you want to save from the History tab at the right to the script tab. You do this by selecting a line of code and clicking the To Source button. Put the following lines of code in the script:

```
        category  <- c("No"  , "Yes", "No Access"   )
        count     <- c(1249, 2175, 1615 )
        barplot(count,names.arg=category, las=1, col="LightBlue",
xlab="Social Media", ylab="# of Respondents")
        relative.frequency <- c(100* count/sum(count))
        barplot(relative.frequency,names.arg=category,las=1,
col="LightBlue", xlab="Social Media",ylab = "Relative frequency
(%)")
```

It is a good idea to add some comments (preceded by a hashtag, #) that document the script:

```
# This script makes bar charts of table 2.2 on p. 53
category  <- c("No"  , "Yes", "No Access"   )
....
```

Save the script as table-2-2.R (R scripts use the extension .R) by selecting in the RStudio menu: File > Save, or by clicking on the Save icon (showing a floppy disk; ask your parents what a floppy disk is). Save the script to the Statistics folder on your hard drive.

To execute one line of the script, put the cursor on that line and click the Run icon in the pane menu. To execute several lines of a script (a block of code), select the lines and click the Run icon.

After you saved the script, close all tabs from the upper left pane.
Clear the Console by selecting in the RStudio menu: Edit > Clear Console.
Clear the History by clicking on the broom icon in the History tab.
Clear all Plots from the lower right pane by clicking on the broom icon in the Plots tab.
Clear the Environment by clicking on the broom icon in the Environment tab.
All panes and tabs are now empty.

Now let us import the script and run it. In the RStudio menu, select File > Open File ...
Navigate to the file you just saved (table-2-2.R) and select it. The script appears in the top left pane. Select all lines of the script. Click the Run icon. R executes the script. Congratulations: you have written your first R program.

Importing Data Sets

Before we continue, let us start with a clean slate: close all tabs from the upper left pane. Clear the Console by selecting in the RStudio menu: Edit > Clear Console.
Clear the History by clicking on the broom icon in the History tab.
Remove all Plots from the lower right pane by clicking on the broom icon in the Plots tab.
Clear the Environment by clicking on the broom icon in the Environment tab.
All panes and tabs are now empty.

I assume you downloaded the data files in text format from the textbook. (If not, do it now. The link is on Canvas. Don't forget to unzip the zip file.)

Select in the RStudio menu: File > Import Dataset > From text (base)...
Navigate to the folder (directory) where you stored the data files from the textbook. Select the file

```
AIG Stock series.txt
```

A dialogue window opens. RStudio makes a guess about the formatting of the data on the Input file (top panel) and shows how it will read the data in the bottom panel (Data Frame). If the Data Frame panel does not look right, you may have to change some of the settings in the left part of the window:

Heading: Yes means that RStudio assumes that the first line of the file contains the variable names
Separator: Tab means that RStudio assumes that tabs were used to separate columns (the tabs are shown in light gray in the top panel). Tip: If you create your own data files, I recommend to use semicolons (;) to separate columns.
Decimal: Period means that RStudio assumes that decimals are separated using periods (.) (many languages other than English use commas to separate decimals.) Tip: If you create your own data files, I recommend to use a period (.) to separate decimals.
na.strings: NA means that RStudio assumes that values that were not available are coded as NA in the data set.

In this case, you don't have to make any changes to the defaults. Click the Import button. The data set is displayed in the upper left pane.

The attach() command will make our lives easier: it will allow us to access variables from the data set by simply giving their names. In the Console window, type:

```
attach(AIG.Stock.series)
```

If you now type the name of a variable in the Console, the Console will display the values of that variable. Type in the Console window:

```
Close
```

and press return. (The variable Close is the daily closing price in $ of AIG stock at the New York Stock Exchange.)

Describing quantitative data

Obtaining the mean, median, standard deviation, or five-number summary is straightforward:

```
mean(Close)
median(Close)
sd(Close)
summary(Close)

max(Close)
min(Close)
```

Find a summary of percentiles:

```
quantile(Close)
```

Find the 10th percentile:

```
quantile(Close, 0.10)
```

Find the coefficient of correlation between the variables Open and Close:

```
cor(Open, Close)
```

Displaying a data set
==========
To get a histogram of the variable Close:

```
hist(Close)
```

We can make the histogram prettier by changing the graphical parameters (see above):

```
hist(Close, las=1, col="LightBlue", xlab="Daily closing price ($)", ylab="# of Days")
```

You can omit the title by adding main="":

```
hist(Close, las=1, col="LightBlue", xlab="Daily closing price ($)", ylab="# of Days", main="")
```

To make a scatter plot of the variables Open (on the x-axis) and Close (on the y-axis):

```
plot(Open, Close)
```

Note that in this data set the variable

```
year.     (no capital and a period after year)
```

represents the date as a real number. To make a time series diagram of the variable Close:

```
plot(year., Close)
```

To plot the time series diagram as a line diagram:

```
plot(year., Close, type="l")
```

We can make the time series diagram prettier by changing the graphical parameters (see above):

```
plot(year., Close, type="l",las=1, xlab="Date", ylab="Closing Price ($)", col="DarkRed", lwd=2, frame.plot=FALSE)
```

The option lwd (line width) controls the width of the line: a bigger value means a fatter line.

Describing and displaying categorical data

Import the data set:

```
Global.txt
```

(see above on how to import data in RStudio).

Inspect the data set: which variables are categorical? Which variables are quantitative?

Attach the data set. In the Console window, type:

```
attach(Global)
```

and press Return.

To get a frequency table with the values (categories) for the variable Education and the counts per category, type in the Console window:

```
table(Education)
```

To sort the table:

```
sort(table(Education))
```

To make a bar chart of the frequency table:

```
barplot(table(Education))
```

The font size on the horizontal axis is too big show all categories. We fix this by changing the expansion factor for axis names (bar labels) using cex.names:

```
barplot(table(Education), cex.names=0.5)
```

(You may have to experiment a bit with the value of cex.names.)

Again, we make the plot prettier by changing some parameters:

```
barplot(sort(table(Education)), cex.names=0.5, las=1,
xlab="Education", ylab="# of respondents", col="DarkBlue")
```

To construct a contingency table of the variables Education and Age.Class:

```
table(Education,Age.Class)
```

Let us store this table in a new object:

```
my.table <- table(Education,Age.Class)
```

To get the marginal distributions apply the function margin.table()

to the table. To get the marginal distribution over the rows use margin.table(...,1). To get the marginal distribution over the columns use margin.table(...,2):

```
margin.table(my.table,1)

margin.table(my.table,2)
```

What does

```
margin.table(my.table)
```

show?

To get the proportions and percentages use prop.table(). (Compare to table 2.4 p. 55.)

To get the proportions of the table total:

```
prop.table(my.table)
```

Note that the sum of all values is 1.:

```
sum(prop.table(my.table))
```

To convert the proportions of the table total to percentages:

```
100*prop.table(my.table)
```

To get the conditional distributions use prop.table(...,1) and prop.table(...,1).

To get the proportions of the row total:

```
prop.table(my.table,1)
```

Note that the sum of the values in each row is 1.

To convert the proportions of the row total into percentages:

```
100*prop.table(my.table,1)
```

To get the proportions of the column total:

```
prop.table(my.table,2)
```

Note that the sum of the values in each column is 1.

To convert the proportions of the row total into percentages:

```
100*prop.table(my.table,2)
```