lab1

Irlanda Ayon-Moreno

2/16/2018

Lab 1: Getting started with R and RStudio

Gaston Sanchez

Learning Objectives:

- Get started with R as a scientific calculator
- Understand pane layout of RStudio
- Understand the help documentation in R
- How to install packages
- Difference between .R and .Rmd files
- Get to know markdown syntax

R and RStudio

- Make sure everybody has installed **R**
  - R for Mac: https://cran.r-project.org/bin/macosx/ (https://cran.r-project.org/bin/macosx/)
  - R for windows: https://cran.r-project.org/bin/windows/base/ (https://cran.r-project.org/bin/windows/base/)
- Make sure everybody has installed **RStudio**
  - RStudio download: https://www.rstudio.com/products/rstudio/download/ (https://www.rstudio.com/products/rstudio/download/)
- Difference between R-GUI and RStudio
  - R-GUI is a simply graphical user interface
  - RStudio is an *Integrated Development Environment* (IDE)
    - It is much more than a simple GUI
    - It provides a nice working environment and development framework
- We are going to use mainly RStudio

R as a scientific calculator

Launch RStudio and notice the default position of the panes:

- Console (entire left)
- Environment/History (tabbed in upper right)
- Files/Plots/Packages/Help (tabbed in lower right)

FYI: you can change the default location of the panes, among many other things: Customizing RStudio (https://support.rstudio.com/hc/en-us/articles/200549016-Customizing-RStudio). If have no experience working with R/RStudio, you don't have to customize right now. It's better if you wait some days until you get a better feeling of the working environment. You will probably be experimenting (trial and error) with the customizing options until you find what works for you.

First contact with the R console

Let's start typing basic things in the *console*, using R as a scientific calculator.

For instance, consider the monthly bills of Leia (a stats undergrad student): cell phone $80, transportation $20, groceries $527, gym $10, rent $1500, other $83. You can use R to find Leia's total expenses:

```
# total expenses
80 + 20 + 527 + 10 + 1500 + 83
```

```
## [1] 2220
```

Often, it will be more convenient to create **objects** or **variables** that store one or more values. To do this, type the name of the variable, followed by the assignment operator `<-`, followed by the assigned value. For example, you can create an object `phone` for the cell phone bill, and then inspect the object by typing its name:

```
phone <- 80
phone
```

```
## [1] 80
```

```
## [1] 80
```

All R statements where you create objects, "assignments", have this form:

```
object <- value
```

this means you assign a `value` to a given `object`; you can read the previous assignment as "phone gets 80".

RStudio has a keyboard shortcut for the arrow operator `<-`: `Alt + -` (the minus sign).

Notice that RStudio automagically surrounds `<-` with spaces, which demonstrates a useful code formatting practice. So do yourself (and others) a favor by ALWAYS surrounding an assignment operator with spaces.

Make more assignments to create variables `transportation`, `groceries`, `gym`, `rent`, and `other` with their corresponding amounts:

```
# variables
transportation <- 20
groceries <- 527
gym <- 10
rent <- 1500
other <- 83
```

Now that you have all the variables, create a `total` object with the sum of the expenses:

```
# total expenses
total <- transportation + groceries + gym + rent + other
```

Assuming that Leia has the same expenses every month, how much would she spend during a school "semester"? (assume the semester involves five months)

```
# semester expenses
5*total
```

```
## [1] 10700
```

Maintaining the same assumption about the monthly expenses, how much would Leia spend during a school "year"? (assume the academic year is 10 months)

```
# year expenses
10*total
```

```
## [1] 21400
```

Object Names

There are certain rules you have to follow when creating objects and variables. Object names cannot start with a digit and cannot contain certain other characters such as a comma or a space. You will be wise to adopt a convention for demarcating words in names.

```
i_use_snake_case
other.people.use.periods
evenOthersUseCamelCase
```

The following are invalid names (and invalid assignments)

```
# cannot start with a number
5variable <- 5

# cannot start with an underscore
_invalid <- 10

# cannot contain comma
my,variable <- 3

# cannot contain spaces
my variable <- 1
```

This is fine but a little bit too much:

```
this_is_a_really_long_name <- 3.5
```

Functions

R has many functions. To use a function type its name followed by parenthesis. Inside the parenthesis you pass an input. Most functions will produce some type of output:

```
# absolute value
abs(10)
abs(-4)

# square root
sqrt(9)

# natural logarithm
log(2)
```

Comments in R

All programming languages use a set of characters to indicate that a specifc part or lines of code are **comments**, that is, things that are not to be executed. R uses the hash or pound symbol `#` to specify comments. Any code to the right of `#` will not be executed by R.

```
# this is a comment
# this is another comment
2 * 9

4 + 5  # you can place comments like this
```

Case Sensitive

R is case sensitive. This means that `phone` is not the same as `Phone` or `PHONE`

```
# case sensitive
phone <- 80
Phone <- -80
PHONE <- 8000

phone + Phone
```

```
## [1] 0
```

```
PHONE - phone
```

```
## [1] 7920
```

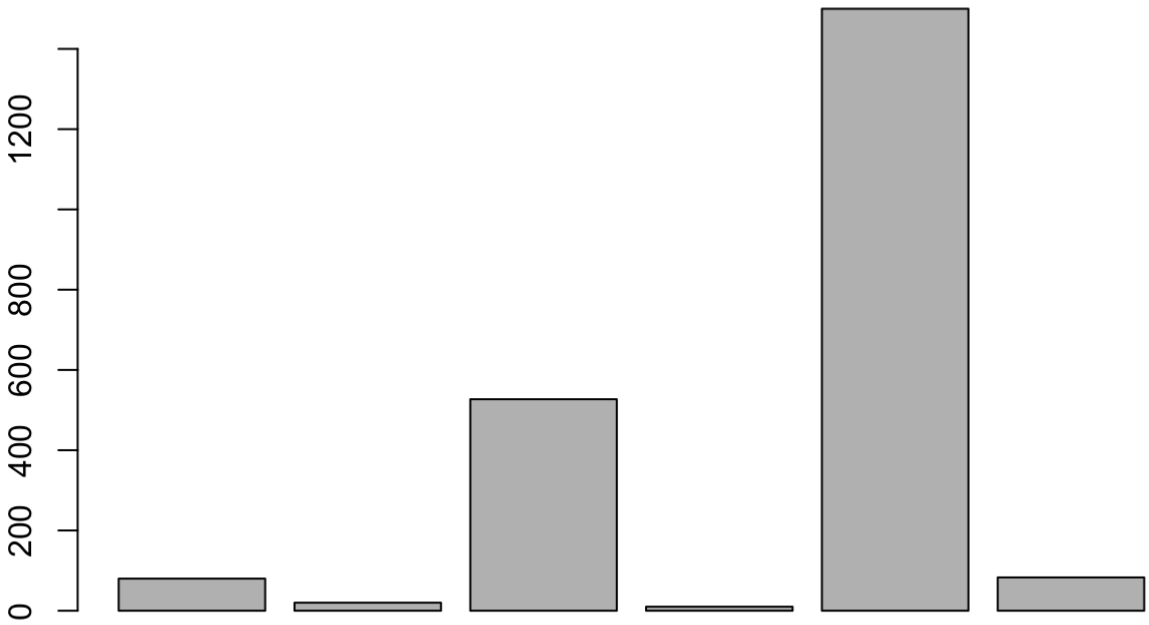
Your turn

Take your objects (i.e. variables) `phone`, `transportation`, `groceries`, `gym`, `rent`, and `other` and pass them inside the *combine* function `c()` to create a vector `expenses`

```
# your vector expenses
expenses <- c(phone, transportation, groceries, gym, rent, other)
```

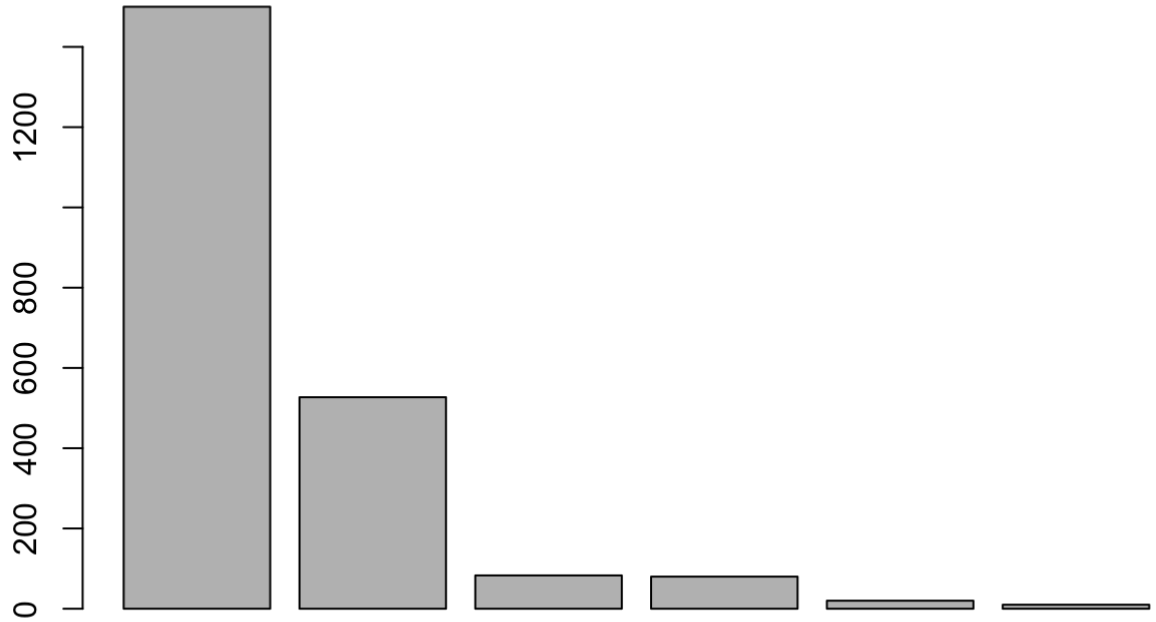
Now, use the graphing function `barplot()` to produce a barchart of `expenses`:

```
barplot(expenses)
```



Find out how to use `sort()` to sort the elements in `expenses`, in order to produce a bar-chart with bars in decreasing order. Also, see if you can figure out how to display the names of the variables below each of the bars. Also optional, see if you can find out how to display the values of each variable at the top of each bar.

```
barplot(sort(expenses, decreasing = TRUE))
```



More about RStudio

You will be working with RStudio a lot, and you will have time to learn most of the bells and whistles RStudio provides. Think about RStudio as your “workbench”. Keep in mind that RStudio is NOT R. RStudio is an environment that makes it easier to work with R, while taking care of many of the little tasks than can be a hassle.

Using an R script file

Most of the times you won’t be working directly on the console. Instead, you will be typing your commands in some source file. The basic type of source files are known as *R script files*. Open a new script file in the *source* pane, and rewrite the previous commands.

You can copy the commands in your source file and paste them in the console. But that’s not very efficient. Find out how to run (execute) the commands (in your source file) and pass them to the console pane.

Getting help

Because we work with functions all the time, it’s important to know certain details about how to use them, what input(s) is required, and what is the returned output.

There are several ways to get help.

If you know the name of a function you are interested in knowing more, you can use the function `help()` and pass it the name of the function you are looking for:

```
# documentation about the 'abs' function
help(abs)

# documentation about the 'mean' function
help(mean)
```

Alternatively, you can use a shortcut using the question mark `?` followed by the name of the function:

```
# documentation about the 'abs' function
?abs

# documentation about the 'mean' function
?mean
```

- How to read the manual documentation
  - Title
  - Description
  - Usage of function
  - Arguments
  - Details
  - See Also
  - Examples!!!

`help()` only works if you know the name of the function your are looking for. Sometimes, however, you don't know the name but you may know some keywords. To look for related functions associated to a keyword, use double `help.search()` or simply `??`

```
# search for 'absolute'
help.search("absolute")

# alternatively you can also search like this:
??absolute
```

Notice the use of quotes surrounding the input name inside `help.search()`

Your Turn

Pythagoras formula

The pythagoras formula is used to compute the length of the hypotenuse, *c*, of a right triangle with legs of length *a* and *b*.

$$c = \sqrt{a^2 + b^2}.$$

hypotenuse

Calculate the hypotenuse of a right triangle with legs of length 3 and 4. Use the `sqrt()` function, and create variables `a = 3` and `b = 4`. If you don't know what's the symbol to calculate exponents, search for the help documentation of the arithmetic operators: `?Arithmetic`.

```
a = 3
b = 4
sqrt(a^2 + b^2)
```

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

Binomial Formula

The formula for the binomial probability is:

$$Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

binomial probability

where:

- *n* is the number of (fixed) trials
- *p* is the probability of success on each trial
- 1 – *p* is the probability of failure on each trial
- *k* is a variable that represents the number of successes out of *n* trials
- the first term in parenthesis is not a fraction, it is the number of combinations in which *k* success can occur in *n* trials

R provides the `choose()` function to compute the number of combinations:

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1},$$

combinations

For instance, the number of combinations in which *k* = 2 success can occur in *n* = 5 trials is:

```
choose(n = 5, k = 2)
```

```
## [1] 10
```

Combinations are typically expressed in terms of factorials as:

$$\frac{n!}{k!(n-k)!}$$

combs

Conveniently, R also provides the function `factorial()` to calculate the factorial of an integer:

```
factorial(4)
```

```
## [1] 24
```

Let's consider a simple example. A fair coin is tossed 5 times. What is the probability of getting exactly 2 heads?

- Create the objects `n`, `k`, and `p` for the number of trials, the number of success, and the probability of success, respectively.

```
n = 5
k = 2
p = 0.5
```

- Use `factorial()` to compute the number of combinations "*n choose k*"

```
factorial(n)/factorial(k)*factorial(n-k)
```

```
## [1] 360
```

- Apply the binomial formula, using `factorial()`, to calculate the probability of getting exactly 2 heads out of 5 tosses.

```
(factorial(n)/(factorial(k)*factorial(n-k)))*p^k*(1-p)^(n-k)
```

```
## [1] 0.3125
```

- Recalculate the same probability but now using `choose()` (instead of `factorial()`)

```
choose(n, k)*p^k*(1-p)^(n-k)
```

```
## [1] 0.3125
```

- Consider rolling a fair die 10 times. What is the probability of getting exactly 3 sixes?

```
n2 = 10
k2 = 3
p2 = 1/6
choose(n2, k2)*p2^k2*(1-p2)^(n2-k2)
```

```
## [1] 0.1550454
```

- Now look for help documentation (e.g. `help.search()` or `??`) using the keyword binomial: `binomial`.
- You should get a list of topics related with the searched term `binomial`.
- Choose the one related with the *Binomial Distribution*, which is part of the R package `stats` (i.e. `stats::Binomial`).
- Read the documentation and figure out how to use the `dbinom()` function to obtain the above probabilities: 2 heads in 5 coin tosses, and 3 sixes in 3 rolls of a die.

```
dbinom(k, n, p)
```

```
## [1] 0.3125
```

```
dbinom(x = 3, size = 3, prob = 1/6)
```

```
## [1] 0.00462963
```

- How would you modify the previous binomial function to calculate the same probability (2 heads in 5 tosses) of a **biased** coin with a chance of heads of 35%?

```
dbinom(x = 2, size = 5, prob = 0.35)
```

```
## [1] 0.3364156
```

- Finally, obtain the probability of getting more than 3 heads in 5 tosses with a biased coin of 35% chance of heads.

```
heads4 <- dbinom(x = 4, size = 5, prob = 0.35)
heads5 <- dbinom(x = 5, size = 5, prob = 0.35)
heads4 + heads5
```

```
## [1] 0.0540225
```

Installing Packages

R comes with a large set of functions and packages. A package is a collection of functions that have been designed for a specific purpose. One of the great advantages of R is that many analysts, scientists, programmers, and users can create their own pacakages and make them available for everybody to use them. R packages can be shared in different ways. The most common way to share a package is to submit it to what is known as **CRAN**, the *Comprehensive R Archive Network*.

You can install a package using the `install.packages()` function. Just give it the name of a package, surrounded by qoutes, and R will look for it in CRAN, and if it finds it, R will download it to your computer.

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

You can also install a bunch of packages at once:

```
install.packages(c("readr", "ggplot2"))
```

Once you installed a package, you can start using its functions by *loading* the package with the function `library()`

```
library(knitr)
```

Your turn

- Install packages `"stringr"`, `"RColorBrewer"`, and “ `XML` ”

```
#install.packages(c("stringer", "RColorBrewer", "XML"))
```

- Calculate: $3x^2 + 4x + 8$ when $x = 2$

```
x <- 2
(3 * x^2) + (4 * x) + 8
```

```
## [1] 28
```

- Calculate: $3x^2 + 4x + 8$ but now with a numeric sequence for x using `x <- -3:3`

```
x <- -3:3
(3 * x^2) + (4 * x) + 8
```

```
## [1] 23 12  7  8 15 28 47
```

- Find out how to look for information about math binary operators like `+` or `^` (without using `?Arithmetic`).
- There are several tabs in the pane `Files, Plots, Packages, Help, Viewer`. Find what does the tab **Files** is good for?
- What about the tab **Help**?
- In the tab **Help**, what happens when you click the button with a House icon?
- Now go to the tab **History**. What is it good for? and what about the buttons of its associated menu bar?
- Likewise, what can you say about the tab **Environment**?

Introduction to Markdown

Besides using R script files to write source code, you will be using other type of source files known as *R markdown* files. These files use a special syntax called **markdown**.

Get to know the `Rmd` files

In the menu bar of RStudio, click on **File**, then **New File**, and choose **R Markdown**. Select the default option (Document), and click **Ok**.

Rmd files are a special type of file, referred to as a *dynamic document*, that allows to combine narrative (text) with R code. Because you will be turning in most homework assignments as `Rmd` files, it is important that you quickly become familiar with this resource.

Locate the button **Knit HTML** (the one with a knitting icon) and click on it so you can see how `Rmd` files are renderer and displayed as HTML documents.

Yet Another Syntax to Learn

R markdown (`Rmd`) files use markdown (https://daringfireball.net/projects/markdown/) as the main syntax to write content. It is a very lightweight type of markup language, and it is relatively easy to learn.

Your turn

In RStudio's menu bar select the `Help` tab. Then click on the option `Markdown Quick Reference`.

Work through the markdown tutorial: www.markdown-tutorial.com (http://www.markdown-tutorial.com)

Your turn: After lab discussion, find some time to go through this additional markdown tutorial www.markdowntutorial.com/ (http://www.markdowntutorial.com/)

RStudio has a very comprehensive R Markdown tutorial: Rstudio markdown tutorial (http://rmarkdown.rstudio.com/)

Review Questions

Take a look at the following commands. Notice the difficulty of reading code when assignment operators are not surrounded by spaces (Don't do this!). Without typing the code on the console, try to guess what will be the output:

```
# example 1
var<-3
Var*2


# example 2
x<-2
2x<-2*x


# example 3
sqrt4 <- sqrt(4)
sqrt4


# example 4
a number <- 16


# example 5
"one number" <- 16
`one number`
one number
```

RStudio working environment

Understand the **pane layout** (i.e. windows) of RStudio. What is the purpose of the following panes?

- Source
- Console
- Environment, History, etc
- Files, Plots, Packages, Help, Viewer

Play with the customizing options of RStudio (appearance of source pane, etc)

- font
- size
- background