Getting Started In R

Gregory S Gilbert

11/06/2016

Goals: Getting Started with R

(1) get R installed and running,
(2) be comfortable with R studio and the R framework (console, editor, quartz, and help panes)
(3) understand basic R notation and operators to be able to do simple functions,
(5) import your own data into R from a spreadsheet

(1) Install R and R Studio on your computer

R is provided free by the R project (https://www.r-project.org/).

You can run it straight out of the box, but most users like to run R within RStudio, which is a free program that helps you keep your R work organized.

INSTALL R on your computer

(1) Go to the R website at http://www.r-project.org/.
(2) Click on the link in the first paragraph **download R** or on the left navigation bar for **CRAN**
(3) Choose a nearby mirror site from the list (e.g., http://cran.cnr.Berkeley.edu)
(4) From the Download and Install R box, choose Linux, (Mac) OS X, or Windows, as appropriate for your operating system.

(5) Download and install the most recent package that your system can handle.

INSTALL RStudio on your computer

(1) Go to https://www.rstudio.com/
(2) From the Products menu, choose RStudio
(3) Select the Desktop option.
(4) Click on the Download RStudio Desktop button to install.

Open RStudio as you would any other application.

....

(2) FOUR KEY WINDOWS in the R framework

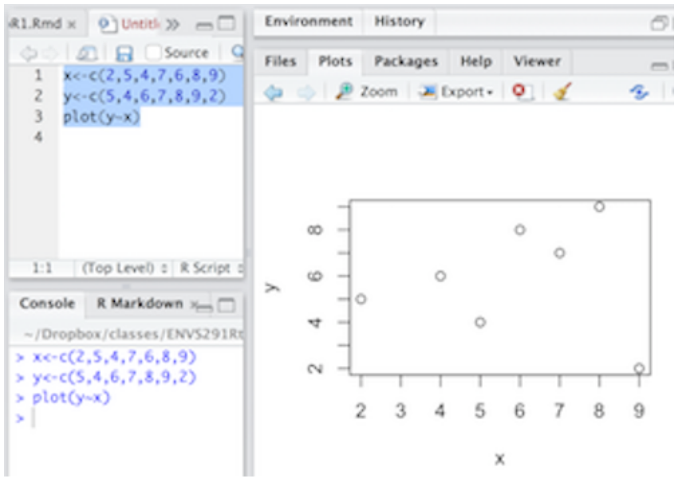
The R interface has four key windows that you will use. RStudio adds a few more, but we'll deal with them as they are needed

Script (Console): This is where you enter and execute commands, and where textual output of your analyses appear.

Source (Editor): This where you write your code, or open it from a saved file. Copy and paste from here into the console window, or better yet, put your cursor on the line of code (or highlight a block of code) and then select Run Selected Lines from the tool bar (or command-return [mac] Control-R [windows]).

Writing code in the Source window provides automatic helpful hints to the formatting of functions, and is easier to fix, re-run, and save than code directly in the Console.

Plots (Quartz device): This is where graphical output goes by default. On a mac, use right and left arrows to scroll through the plots you make in a session. On a PC you need to (1) create a graphic, (2) with the graphic window active, go to the History tab, next to File. (3) Click on Recording and make sure it is checked. (4) Now each graph in the session is recorded; use PgUp dn PgDn to scroll.



```
x<-c(2,5,4,7,6,8,9)
y<-c(5,4,6,7,8,9,2)
plot(y~x)
```

```
> x<-c(2,5,4,7,6,8,9)
> y<-c(5,4,6,7,8,9,2)
> plot(y~x)
>
```

Help: Finally, R has very useful help functions. For instance, if you what help for the function plot, either help(plot) or

```
?plot
```

will open up an R Help Window (in RStudio, in the same place as Plots), with detailed, standardized-format instructions on how to use that function. In RStudio, you can also just click on the Help tab, then put the term you want in the search bar with the magnifying glass. The format for R help at first is a bit cryptic, but all the functions take the same format. Once you get it, you can quickly learn what you need for any function in the R universe.

....

(3) Trying it out - operators, objects, and funcitons

Let's start of playing directly in the Console window. You face the dreaded > prompt, with no indication of what to do. Don't panic!

R as a calculator

In the Console window, after the ">"" prompt, just type some arithmetic expressions, using familiar **operators** +-*/(), each followed by a return

```
6+6

## [1] 12

43*2+(7+2)/14

## [1] 86.64286
```

In the console, if you want to repeat, or copy and modify something from a previous line, just **use the up arrow to move through previous lines** until the one you want appears at the active prompt. Then edit it as you want, and hit return again! Try arrowing up to the 6+6 line, then change it to 6+6*3 and return

Now let's use the Script window

In the Script window, type 7+5. With the cursor still on that line, click the Run button (or command-return (mac) or control-R (windows)). Look what appears in the Console!

Assigning values to objects

Data are bundled together in **objects** of different types, and then we use **functions** to manipulate the objects. We **assign** values to an object using "<-"

```
a<-64  #assign the value 64 to the object "a"
b<-(a+23)/4 #assigns the value of the expression to "b"
a #just typing the name of the object shows its value

## [1] 64

b*a  #this returns the product of a and b, but does not keep it

## [1] 1392

d<-b*a # this assigns the product to a new object "d"
d

## [1] 1392
```

Names for objects

Names for objects should start with a letter and avoid special characters or spaces. Two special characters that are permitted and useful are "_" and ".".

names OK	May cause problems
hair_color	hair color
dbh.mm	dbh(mm)
sample1	sample$4

Everything in R is case sensitive:

dbh.mm ≠ DBH.mm

sample1 ≠ Sample1

Functions do things with objects

Functions are always followed by their object, included in parentheses. Many functions can also take options inside the parenthesis, and separated by commas from the object

function(object,option1, option2, option3)

```
sqrt(25)

## [1] 5

a<-125
sqrt(a)

## [1] 11.18034

round(a/7,digits=2)  #round a/7 to 2 digits

## [1] 17.86

round(sqrt(a/7),digits=3) #you can nest functions

## [1] 4.226

aover7<-round(a/7,digits=2) #assign the output to an object
aover7 #but you don't see the value until you request it

## [1] 17.86

v1<-c(3,4,6,8,7,3,4,9,8) #make vector of numbers
mean(v1) #calculate the mean of the numbers

## [1] 5.777778

length(v1) #find the number of elements in the vector

## [1] 9
```

....

Relational operators

symbol	meaning	example	read as
<	less than	x < y	x is less than y
>	greater than	x >y	x is greater than y
==	equal to	x ==y	x is equal to y
<=	less than or equal to	x <=y	x is less than or equal to y
>=	greater than or equal to	x >=y	x is less than or equal to y
!=	not equal to	x!=y	x is not equal to y
%in%	included in the set	var %in% c(1,3,5)	var is in the set

.
.
.

Basic R Notation

symbol	meaning
#	Anything following the "#" and before a carriage return is treated as a comment, and is not executed. Use it to annotate code. #annotate!
?	Help function, where ?x looks up R help for x. Also help(x)
<-	Assign. Sets an object to be equal to whatever is on the right side of the "<-" symbol. a<-9. Note equivalent to a=9, BUT = can only be used at top level, so avoid using = to mean assign.
()	Round brackets. Functions act on whatever is inside the round brackets.log10(100) returns the base10 log of 100, that is, 2
c()	Concatenate function. Creates a vector of multiple elements. d<-c(1,2,4,6) # sets d to be a 4-element vector with elements 1 2 4 6
[]	Square brackets. Indicates the position inside a vector or matrix. d[3] has the value 4 for the vector d<-c(1,2,4,6). f[1:5,2:3] returns rows 1 to 5 of columns 2 and 3 of matrix f
{}	Curly brackets. Marks the start and stop of a loop or complex function. for (i in 1:5){b<-b+a[i]}
;	functions as a carriage return, and indicates the end of a function. You can use it to string together several simple functions on the same line. c<-3;d<-c*2;c;d|

.
.
.
.
.
.

Read data in from a spreadsheet

There are many ways to import data into R. Here we improt data from a spreadsheet format into a data frame called "df1".
Note 1 Save your data as comma-delimited (.csv) or tab-delimited (.txt) file. In Excel, choose SaveAs, File Format: Comma Separated Values (.csv)) or Tab Delimited Text (.txt)
Note 2 Except for one, all these variations use read.csv for simplicity. They can also all be used with read.table.
Note 3 The pathways given here are in mac OS X style ("/Users/Greg/Mydatafolder"). The Windows equivalent is ("C:"). **Note 4** To peek at the data after you read it in, use head(df1)

Read from a .csv file with headers with full path

```
df1<-
read.csv("/Users/greg/Dropbox/classes/ENVS291Rtransition/IntroToR_Fall2016/so
medata.csv")
```

Read from a .txt file with headers with full path

```
df1<-
read.table("/Users/greg/Dropbox/classes/ENVS291Rtransition/IntroToR_Fall2016/
somedata.txt", sep="\t", header=TRUE)
```

Read from a .csv file with headers using read.table, full path

```
df1<-
read.table("/Users/greg/Dropbox/classes/ENVS291Rtransition/IntroToR_Fall2016/
somedata.csv", sep=",", header=TRUE)
```

Read from a .csv file with headers, with set working directory

Wet the working directory (setwd) first, then you only need the file name. The working directory stays set until you change it.

```
setwd("/Users/greg/Dropbox/classes/ENVS291Rtransition/IntroToR_Fall2016/")
df1<-read.csv("somedata.csv")
```

You can also set the working directory from R Studio using the menu Session > Set Working Directory > Choose Directory.

```
getwd()  #tells you what your current working directory is

## [1] "/Users/greg/Dropbox/classes/ENVS291Rtransition/IntroToR_Fall2016"
```

Use a dialog box to read from a .csv file with headers

```
df1<-read.csv(file.choose())
```

Read a .csv file from web URL

```
df1<-read.csv("http://greggilbertlab.sites.ucsc.edu/wp-
content/uploads/sites/276/2015/09/RegressionDataset.csv")
```

Working directory:

The **working directory** is the default folder you want to use to read from and write to in a particular session. Specifying the path to the working directory allows you to use much shorter names to refer to specific files. You can set the working directory using the function setwd("/path/to/your/folder/here"), from the Session menu in RStudio, or from the Files:More panel in RStudio.

In Windows the pathway would be setwd("C:").

Libraries (packages):

Libraries , or **packages**, are sets of functions designed to do specific things, written by various people in the R community. The base installation of R has a number of useful functions, but the real value of R is the development of different libraries.

Install packages from the Packages Panel in RStudio. Choose Packages, then Install, type in the first letters of the name of the package, and click the one you want. Check the install dependencies box, and then click Install. You only need to install once (it writes the package to your computer for R to access it.)

To use a package you installed, you then need to load it into each session in which you use it. In the RStudio Packages panel just click on the check boxes for the installed packes you want.

You can do this in your script as well. I usually do this at the beginning of my analytical script. This can be done using either of two functions. To load the **ape** package for Analyses of Phylogenetic and Evolution:

```r
library(ape)
require(ape)
#both do the same thing
```

Or, you can include it directly in your script code

You can then use Package & Data: Package Manager to load the libraries you want. Or, use the command line library(xxx), where xxx is the name of the library you wish to load.

Workspace

Your **Workspace** includes the objects and libraries you have open in a particular session. When closing an R session, if you choose to Save workspace image, they will all still be there the next time you open, ready to go. You can save your workspace in RStudio from the Session menu > Save Workspace As. You can then restore the Workspace the next time you work on that projectd. Save different workspace files for different projects.