R Tutorial

Dirk Moore

2023-09-20

This is a short R session that serves a tutorial for those just getting started in R. The best way to run R is via the R Studio interface, since it provides a convenient arrangement of panes, one for the R command prompt, one for an editor, and two more for plots and other displays. For this introductory tutorial, you can just enter these commands directly at the R command prompt.

## variables, vectors, and simple operations

To get a feel for how R works, enter a numerical expressions, such as these:

2+3

[1] 5

2^3

[1] 8

2\*\*3 # same as the previous expression

[1] 8

Note that the # character indicates a comment, which of course is not executed. The [1], which is perhaps superfluous here, just indicates that the printed result is the first (and in this simple case only) element of the result. (The purpose of the outpur format will become clear when you look at long vectors that stretch out over more than one line.)

It is possible to assign values to variables using the <- symbol:

x <- 3  
y <- 2  
y^x

[1] 8

In addtion to constants, R can work with vectors, as shown in the following. Here, the c function is used to combine numbers into a vector.

x.vec <- c(1, 3.5, 7)  
y.vec <- c(2, 7, 8.6)  
x.vec

[1] 1.0 3.5 7.0

y.vec

[1] 2.0 7.0 8.6

Any legal variable name may be used for a vector; the .vec notation is just a readability aid. Note that, unlike in some other systems, variable names are case sensitive; lower and upper case letters result in distinct variables.

One may perform operations on vectors, as long as they are the same length:

x.vec + y.vec

[1] 3.0 10.5 15.6

x.vec + 2\*y.vec

[1] 5.0 17.5 24.2

The c function may be used to combine two vectors:

z.vec <- c(x.vec, y.vec)  
z.vec

[1] 1.0 3.5 7.0 2.0 7.0 8.6

Brackets may be used to access individual components of a vector. For example, to access the second, third, and fourth elements of z.vec, use this:

z.vec[2:4]

[1] 3.5 7.0 2.0

The notation 2:4 is shorthand for c(2, 3, 4).

Vectors may also contain character variables:

abc.vec <- c("This", "is", "a", "vector", "of", "character", "variables")  
abc.vec

[1] "This" "is" "a" "vector" "of" "character" "variables"

To end a session, you may enter the quit function with no arguments, q().

## Working with data in data frames

Data frames are the principal way that data sets are stored in the R environment. Suppose you have a data set phCavities.csv stored on your computer in the directory C:\temp. To read this data into R, you need to set your working directory to the location of the data. Here, the backslash character needs to be doubled for R to interpret it correctly. (The first \ is and escape character that tells R to interpret the next \ as an actual backslash character.)

Next, we may check that R is pointing there by using the getwd command.

Finally, we read in the data. This data set has as its first row the names of the variables and, in succeeding rows, the values of those variables for all of the subjects. The data are “comma separated values”, hence the file extension .csv. To read in the date, use the “read.csv” function.

setwd("C:\\RGit\\RU\_BIST0625")  
getwd()

[1] "c:/RGit/RU\_BIST0625"

phCavities <- read.csv("phCavities.csv", header=T)

We may use the str command to show the structure of the data set, and the head command to examine the first few rows

str(phCavities)

'data.frame': 27 obs. of 2 variables:  
 $ pH : num 7.14 7.11 7.61 7.98 7.21 7.16 7.89 7.24 7.86 7.47 ...  
 $ grp: chr "none" "none" "none" "none" ...

head(phCavities)

pH grp  
1 7.14 none  
2 7.11 none  
3 7.61 none  
4 7.98 none  
5 7.21 none  
6 7.16 none

Be aware that in R, variables are case-sensitive. Thus, phCavities and phcavities would denote different variables.

One way to access the different varialbes in phCavities is via the $ operator:

phCavities$pH

[1] 7.14 7.11 7.61 7.98 7.21 7.16 7.89 7.24 7.86 7.47 7.82 7.37 7.66 7.62 7.65 7.36 7.04 7.19 7.41 7.10 7.15 7.36 7.57 7.64 7.00 7.25 7.19

phCavities$grp

[1] "none" "none" "none" "none" "none" "none" "none" "none" "none" "none" "none" "none" "none"   
[14] "none" "none" "cavities" "cavities" "cavities" "cavities" "cavities" "cavities" "cavities" "cavities" "cavities" "cavities" "cavities"  
[27] "cavities"

Alternatively, one may attach the data frame to allow direct access to the variables:

attach(phCavities)  
pH

[1] 7.14 7.11 7.61 7.98 7.21 7.16 7.89 7.24 7.86 7.47 7.82 7.37 7.66 7.62 7.65 7.36 7.04 7.19 7.41 7.10 7.15 7.36 7.57 7.64 7.00 7.25 7.19

Be very careful when using the attach command, as it can lead to confusion. Suppose, for example, you issue the following command:

pH <- 5  
pH

[1] 5

Now you have defined the variable pH in the workspace to have a different value than in the phCavities data frame, and the workspace assignment takes precedence. To see the precedence, use the search() command:

search()

[1] ".GlobalEnv" "phCavities" "package:stats" "package:graphics" "package:grDevices" "package:utils" "package:datasets"   
 [8] "package:methods" "Autoloads" "package:base"

When you type the variable name ‘pH’ R first searches the workspace (.GlobalEnv) and it finds pH there, which has the value 5. But if you remove this variable via the rm command and do the same, this is what you get:

rm(pH)  
pH

[1] 7.14 7.11 7.61 7.98 7.21 7.16 7.89 7.24 7.86 7.47 7.82 7.37 7.66 7.62 7.65 7.36 7.04 7.19 7.41 7.10 7.15 7.36 7.57 7.64 7.00 7.25 7.19

Now when you type pH R searches .GlobalEnv and does not find pH there (since you just removed it). Next it searches the next object in the search path, which is pHCavities, and finds it there.

Understanding the search path will help the user to avoid confusion on what variable is being accessed.