An R Tutorial

1. Starting Out

R is an interactive environment for statistical computing and graphics, and it can be freely downloaded at

http:\\r-project.org\

*I highly recommend downloading RStudio (you first need to install R), which has a more convenient user interface than the standard R software.* http://www.rstudio.com/

A useful R resource for introductory statistics is Verzani’s book *simpleR*, which can be downloaded for free at <http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf>.

Once you enter R, you will receive the following prompt:

>

**Note**: In order to keep a record of your work, you should always type your code in a **separate** file, which we call a script. R has its own basic editor from which you can directly execute code. Under the File menu in R you will see the option to open a New Document. When you save this document (script), make sure it has the **file extension .r** (or .R). This way you can open the script from a folder by double-clicking on it and your computer will know to open it with R.

You can execute code from the script either line-by-line or by highlighting multiple lines of it and executing them all at once.

**Command to execute code from script:**

Windows: ctrl + r

Mac: command + return

2. Basics

**Preliminaries**

When R is waiting for a new command the prompt line will begin with a >. If R is waiting for you to finish a command (if you forgot to close a parenthesis for example) the prompt will be a + and you can finish your command line as though you had just typed a space. Press enter when you have typed in your command. To force a new prompt (if you are prompted with a + and can’t figure out why for example) hit Esc.

> cos(pi

+ )

[1] -1

The ‘#’ sign is used for writing *comments*, and R will ignore everything to the right of a # when executing commands.

**TIP: From the command line, you can press the up arrow key to see previous commands that were executed.**

For most functions, R has useful **help information**. For example, type

> help(cos)

to see help on the cosine function. From this help, you can read that the angle is in radians and that the function for arccos is called acos() in R.

**Arithmetic**

To compute something type it after the prompt > and the solution is returned after the [1].

> 5

[1] 5

R follows the rules for order of operations and ignores spaces between numbers (or objects)

> 4^ 3-2\* 7+9 /2

[1] 54.5

The equation above is computed as 43− (2 • 7) + 9/2 .

**Objects**

Often you want to save a value so that you can refer to it again. You can think of objects as being variables. To do this, pick a name that makes sense (case sensitive), and use the equal sign:

> junk=11/2 # notice that there is nothing returned by R when you do this.

To recall what you just saved use it like you would use any other number.

> junk

[1] 5.5

> junk+2

[1] 7.5

To see the list of objects that you have created so far in your R session, use the list function ls().

> ls()

[1] "junk"

> remove(junk) # Removes an object from your directory

The number pi is already in R like an object, but since you did not create it in R it does not show up in the ls() list. For Euler’s *e*, use exp(1) (which gives you *e1*).

> pi

[1] 3.141593

**Note**: In other R references, you will also see arrows (<- or ->) used to assign variables. In early versions of R, arrows were used instead of =. Current versions of R will accept either.

**Vectors**

The simplest way to save data in R is as vectors or matrices. A vector is an object in R that is a list of numbers. For example, consider the following list of heights (in inches): 68, 69, 72, 60. To write this list in R as a vector you use the *concatenate* function (c is for concatenate).

> heights=c(68,69,72,60) # make sure to give your vectors logical names

> heights

[1] 68 69 72 60

Arithmetic works the on vectors as though each element in the vector were entered in the equation separately.

> heights\*2.54 # to convert inches to cm

[1] 172.72 175.26 182.88 152.40

If you only want to return a specific element of the vector (say the second) then use square brackets [ ] to access the value in that spot.

> heights[2]

[1] 69

Note that the number 69 is from the original vector of heights was returned even though in the previous prompt you multiplied this by 2.54. If you wanted to save the heights in cm then make a new object for the cm heights:

> heights.cm=heights\*2.54

**Note**: In general, square brackets [ ] are used to access an element from a vector or matrix and parentheses are used in functions such as c(...), ls(), or remove(...).

To get several elements of a vector make a list of the ones you want inside the brackets.

> heights[c(1,3)] # To view the first and third elements of x.

[1] 68 72

Note that heights[1,3] gives an error since heights only has one dimension.

If you want to view all but one element you can use the brackets with a minus sign.

> heights[-1] # All but the 1st element of heights

[1] 69 72 60

You can also use a list here such as heights[-c(1,3,4)] to see just the 2nd element.

**Matrices**

A matrix is a collection of vectors (of the same length). An easy way to make a matrix is to bind vectors together. To do this use either cbind (“column bind”: each vector is a row of the matrix) or rbind (“row bind”).

> heights=c(68,69,72,60)

> weights=c(130,147,219,105)

> data.matrix=cbind(heights,weights)

> data.matrix

heights weights

[1,] 68 130

[2,] 69 147

[3,] 72 219

[4,] 60 105

Like with vectors, you can pull out one or more entries from the matrix. In a matrix, you must specify which row and column (separated by a comma) the entry is in. If you leave either one of these blank, it will return all of the entries of the one you left blank.

> data.matrix[,2] # To view the second column of data.matrix

[1] 130 147 219 105

> data.matrix[c(1,3),] # To view the first and third row of data.matrix

heights weights

[1,] 68 130

[2,] 72 219

> data.matrix[4,2] # To view the 4th row, 2nd column entry

[1] 105

Note that if you leave out the comma to separate the columns from rows then it will give you the element counting by columns until it gets to specified element.

> data.matrix[6]

[1] 147

3. Some Useful Features and Functions

**Comparisons in Objects**

Once you have data to work with, there are lots of things you can have R do with it.

> x = c(3,4,7)

> x>=5 # This lists which elements in x are greater than or equal to 5

[1] FALSE FALSE TRUE

You can use < <= == >= > to compare values.

> x[x >=5] # this will produce all numbers in x that are greater than or equal to 5

[1] 7

Recall that square brackets are used to access a sublist of a vector or matrix. Thus x[x >=5] is taking the sublist of x that consists of those values that are greater than or equal to 5.

Note that a **TRUE is equivalent to 1** and a **FALSE is equivalent to 0**. So if you take the sum of elements in a vector you can find the number of TRUEs.

> sum(x>3)

[1] 2

**Functions to Create Vectors**

There are functions in R to create vectors that follow a specific pattern (such as counting numbers), instead of using concatenate c(...).

> 1:3 # A shortcut for creating a sequence that increases by 1

[1] 1 2 3

> seq(3,1,by=-.5) # Creates a sequence from 3 to 1 counting down by -.5

[1] 3.0 2.5 2.0 1.5 1.0

> rep(1,3) # repeat 1 three times

[1] 1 1 1

> rep(1:3,2) # repeat a sequence from 1 to 3 two times

[1] 1 2 3 1 2 3

> sort(rep(seq(1,2,length=3),2)) # what is this doing?

[1] 1.0 1.0 1.5 1.5 2.0 2.0

Many functions have different parameters that you can enter. Use help(function) to learn more about them. For example, help(seq).

Example: Try to read this function and understand how it produces the vector below. Remember that object[-x] removes the *x*th element from object.

> c(sort(c(seq(1,2,by=.2),1.7))[-c(2,3)],.7)

[1] 1.0 1.6 1.7 1.8 2.0 0.7

Note: if you are making a complicated vector that consists of only a few elements, it is easier to just concatenate them together as in the vectors section.

**More Useful Functions**

For additional information on any of these functions use help(function) i.e. help(sin).

For functions with a \* see the help for useful additional features.

All of these are followed by ( ), to apply the function to some value(s).

sin, cos, tan, asin, acos, atan: trig functions

log\*, log10, exp: log and exponential functions

round\*, signif\*: rounds, gives significant digits

length: gives the number of elements

sort, rank, rev: sorts, gives the rank, reverses the order of a vector

mean, sum, prod: mean, su

m, product of elements in x

min, median, max, quantile\*: order statistics for a vector

var, sd, cov, cor: variance, standard deviation, covariance, correlation

union, intersect: union, intersect of sets

4. Reading Data into R

Typically you already have data in a file that you want to import into R. No matter where the file is saved (whether on your computer or the internet), R reads it in the same way.

To read a text file (.txt) from your computer:

> x=read.table("C:/Documents and Settings/username/Desktop/sample.txt", header=T)

Replace the file path with the one where you saved the file and notice that the slashes are forward slashes / instead of backslashes. Also note that the name of the file has **quotes** around it.

*Tip*: If you are using a Mac, click on the file in a Finder window and drag-and-drop it into R. You will then have the file path for the file.

To read a file from the internet you do pretty much the same thing:

> x=read.table("http://www. ... /sample.txt")

Again replace the path with the url of where the data are, and again make sure to use quotes around the url.

Although there care many ways to load data into R, the simplest way is to convert the data to a text (.txt) file (such as in Excel) and use read.table().

If data is already saved in a .csv (comma separated values) format then you can use read.csv the same way that you would use read.table. If the data have missing values and you want to convert to a .txt then you need to find all the missing values and replace them with NA which isn’t much fun. If the data is in a .csv format already then you don’t need to write in the NAs.

For example, try:

> data=read.csv("http://teaching.up.edu/MTH361/data/birth.csv")

**“Looking at” the data**

Once you have read in a data set, you want to look at it. If you do

> data

you will see a long list of what the data look like. This is not useful if the data set is very large. If you want to look at the data directly, it is best to open it in Excel.

You can see a summary of the data using the summary command though:

> summary(data)

This allows you to see what all of the variables are and a sense of what the values are for each of them. Notice that it also lets you know if there are missing values (NA).

To see what its dimensions are:

> dim(data)

[1] 800 14

Note that this means that there are 14 variables where each variable has 800 data points.

**IMPORTANT: Accessing the variables (attach and detach)**

If the data set has names for each column then you can look at those names by using:

> names(data)

[1] "plural" "sex" "mage" "weeks" "marital" "racemom" "hispmom"

[8] "gained" "smoke" "drink" "tounces" "tgrams" "low" "premie"

Note that the [8] just means that "gained" is the name of the elements in the 8th column of data.

To access one column at a time (the variables) when the data have names, you can use the dollar sign:

> mean(data$tounces)

[1] 116.3763

This returns that the mean of the column named tounces is 116.3763.

A shorter way to do this that we will usually be using is to attach the data set (a.k.a. the *data frame*) after loading it.

Note the following:

> mean(tounces)

Error in mean(plural) : object " tounces " not found

This happened because “tounces” is not a variable R recognizes (yet). To be able to use the variables directly without using the $, we attach the data:

> attach(data)

> mean(tounces)

[1] 116.3762

When you are done using the data set, detach it so that you do not run the risk of loading the same variable names from different data sets:

> detach(data)

The following is a technical explanation of attach.

What exactly does attaching do? When you ask R to use a value of a variable or a function it needs to find it. R searches through several \environments" for these variables. By attaching a data frame, you put the names into the second environment searched (the name of the dataframe is in the first). These are masked by any variables which already have the same name. There are consequences to this to be aware of. First, you might be confused about which variable you are using. And most importantly, you can't change the values of the variables in the data frame without referencing the data frame.[[1]](#footnote-1)

**Datasets with NAs**

Look at data$gained from what you read in above. There are NAs in there which means that some values are missing. Some functions don’t like NAs and you have to enter in a special parameter to ignore them (see the help section for the function mean).

> is.na(data)

produces a list of TRUEs and FALSEs for each element in the data set saying if the data is NA or not. If you ask data==NA, R will always return FALSE. Remember that TRUE is equivalent to 1 and FALSE to 0 so you can do

> sum(is.na(data))

to find how many points are missing. Note that if you use length on a vector, it will also count the NAs.

5. Problems

Show the R commands that you used for the following questions. Read in data from

http://teaching.up.edu/MTH361/data/birth.csv

(a) How many (non-NA) entries are there for the column “gained”.

(b) Find the number of entries in the first 100 observed “weeks” that are less than 35.

(c) Find the mean value of the 47 smallest elements in “mage”. (Hint: use the function sort.)

1. simpleR {Using R for Introductory Statistics, John Verzani, pg. 24 [↑](#footnote-ref-1)