

Introduction to R for STAT 549

D. Patterson, September 2010

R is available as a free download from www.r-project.org for Unix, Windows and Mac. It is essentially a freeware version of the commercial product S-Plus, but without as fancy a user interface and without some of the data input capabilities. There is no difference in the commands for the three versions of R, although the user interface varies a little. R is available in the Math 206 PC computer lab.

The following shows you just a few of the capabilities of R. There are many graphics and modeling features which are not discussed here; see the Help menu or consult one of the many online manuals.

Throughout this document, text following a # is a comment and need not be typed.

1. Startup

When you start R for Windows (currently version 2.11.1), the R Console will appear. All commands can be typed in the Console window at the “>” prompt. Commands are executed one at a time. You can recall past commands (and edit them) by using the up and down arrows. However, you will find it to your advantage to use the advanced “scripting” feature of R which allows you to type several commands and then execute them all at once, then go back and edit the commands and re-execute them, etc. This is very convenient, particularly if you are doing repetitive analyses.

To open a Script window, go to **File...New Script**. You type a set of commands in the window; no “>” prompt appears; you simply type the commands in the order you would like them executed. When you are ready to execute the commands in the Script window, go to **Edit...Run All**. You can also run a subset of the commands in a window by highlighting them and hitting **Ctrl-R** (or selecting **Edit...Run line or selection**). You can also run a single line by putting the cursor anywhere on that line and hitting **Ctrl-R** (the line does not need to be highlighted).

To save a script, select **File...Save** or **Save as**. R scripts should be saved with a .R extension (you will have to type this explicitly; R will not automatically append it to the script name).

To change the default directory where script files are saved or looked for, select **File...Change dir**. This will also be the directory where data files, graphs, etc. are looked for and saved. You can see what the current working directory is with

```
> getwd()
```

You can add comments to a script by preceding the comment with the # sign, e.g.,

```
> median(x)      # computes median
```

All named objects created in an R session (see section 3 of this document) are stored and available throughout the session. At the end of the session, they will be deleted unless you save them in a file by using **File...Save Workspace**. The next time you run R, you can use **File...Load Workspace** to load the saved workspace.

2. Arithmetic

You can perform arithmetic calculations in R using the operators +, -, *, /, ^ as follows:

```
> (11+5)/(2^3)   # ^ indicates raising to a power
[1] 2            # the answer is 2; ignore the [1]; R generally represents
                # any set of numbers (even one number) as a vector and the
```

```
# [1] simply indicates that 2 is the first (and only)
# element of a vector.
```

Use parentheses to indicate the order of operations. \wedge is done first, then $*$ and $/$, then $+$ and $-$.

```
> 11+5/2^3
[1] 11.625      # this is 11 + (5/8)
```

You can also use any of the many built-in functions for computations:

```
> sqrt(11+5)
[1] 4
```

Other functions include: `log()`, `exp()`, `log10()`, `abs()`.

3. Variables

You can store numbers by assigning a name using the assignment operator “=” or the operator “<-” (two characters: “less than” followed by “minus”). This document uses “<-” for clarity, but “=” is perfectly acceptable. R is case sensitive (A is different from a).

```
> a <- 5 # could also type a = 5
> a      # if you type the name of an object, R will give its value
[1] 5
> aa <- sqrt(13*3)
> aa
[1] 6.244998
> b      # Nothing has been stored in b yet
Error: object "b" not found
```

Valid names in R can be of any length, must start with a letter, but can contain any combination of upper and lower-case letters, numbers, and periods. The following are all valid names:

```
a
N
n203
var.pop1
very.long.name.containing.many.characters.12345
```

You can use named variables in calculations, as in the following sample size calculation for the size of a simple random sample to estimate the population mean to within 2 units with probability .95 for a finite population of size 100 with standard deviation 5 (see p. 36 of Thompson, Sampling, 2nd ed.):

```
> N <- 100
> d <- 2
> z <- qnorm(.975) # the .975 quantile of N(0,1)
> s <- 5
> n0 <- z^2*s^2/d^2
> n0
[1] 24.01 # assuming infinite population
> n <- (1/n0 + 1/N)
> n
[1] 19.36134 # for finite population
```

A better way than typing the above commands in the Console would be to type the commands in a script window (without the `>` prompt). You can then execute them as a group and easily edit them if there are

errors or if you want to change the values. You could also make the above “program” into a function: see section 8 below.

4. Vectors

You can create vectors in R using the function “c”:

```
> x <- c(2,3,5,1,4,4)
> x
[1] 2 3 5 1 4 4
```

There are many functions which operate on vectors including `mean()`, `var()`, `sd()`, `median()`, `sum()`, `max()`, `min()`. So, for example, to get the sample standard deviation of the numbers in x:

```
> sd(x)
[1] 1.47196
> sqrt(var(x)) # another way to get the std. dev.
[1] 1.47196
```

Most functions which operate on a single number will also operate on a vector and will yield a vector as the result. For example:

```
> sqrt(c(1,4,9,16))
[1] 1 2 3 4
```

This is convenient for doing repeated calculations. For the sample size example in Section 3 of this document, we could calculate the sample sizes for several different values of `d`. The commands below were typed in a script. When the script is executed (using Edit...Run all), the commands are transferred to the Console and executed.

```
d <- c(2,5,10)
N <- 100
s <- 5
z <- qnorm(.975) # .975 quantile of N(0,1)
n0 <- (z^2*s^2)/(d^2)
1/(1/n0 + 1/N)
```

The output is just as if you had typed the commands one by one in the Console:

```
> d <- c(2,5,10)
> N <- 100
> s <- 5
> z <- qnorm(.975) # .975 quantile of N(0,1)
> n0 <- (z^2*s^2)/(d^2)
> 1/(1/n0 + 1/N)
[1] 19.3607681 3.6993498 0.9512294
```

The three sample sizes at the end correspond to the values 2, 5, and 10, respectively, for `d`.

5. Probability Distributions

R has many probability distributions built in. One can compute probabilities, quantiles (e.g., z-scores and t-scores), and generate random numbers.

Standard normal distribution:

```

> pnorm(1.43) # P(Z < 1.43)
[1] 0.9236415
> qnorm(.995) # the .995 quantile of the standard normal.
[1] 2.575829
> rnorm(25) # generates a vector of 25 standard normal random numbers.

```

Any normal distribution:

```

> pnorm(1.43,1,2) # P(X < 1.43) where X is normal with mean 1 and
# standard deviation (not variance) 2
[1] 0.5851163

```

You can also add a mean and standard deviation to qnorm and rnorm.

t-distribution:

```

> pt(1.43,10) # P(t < 1.43) for a t-distribution with 10 degrees of freedom.
[1] 0.9083956
> qt(.995,10) # the .995 quantile of a t-distribution with 10 degrees of freedom.
[1] 3.169273

```

Uniform distribution:

```

> x <- runif(100) # generates a vector of 100 random values from a uniform
# distribution over (0,1).
> x <- runif(100,0,10) # generates a vector of 100 random values from a uniform
# distribution over (0,10)

```

Example in confidence interval calculation:

```

> x <- c(1,4,3,3,4,2,7,3) # Hypothetical set of 8 data values.
> mean(x) - qt(.975,7)*sd(x)/sqrt(8) # lower limit of 95% CI for the mean.
> mean(x) + qt(.975,7)*sd(x)/sqrt(8) # upper limit of 95% CI for the mean.

```

6. Random samples with and without replacement

```

> x <- c(4,3,1,1,2,7,12,4,6,9)
> sample(x,4) # generates a simple random sample (i.e., without
[1] 3 4 9 1 # replacement) of size 4 from x.
> sample(x,4) # generates a different random sample
[1] 6 4 9 3
> sample(x,4,replace=T) # generates a random sample with replacement
[1] 4 2 6 6

```

7. A List of Common Commands:

Miscellaneous:

<code>ls()</code>	Lists all objects created
<code>args(fun)</code>	This lists the arguments of the function ‘‘fun’’ where ‘‘fun’’ is the name of a function, either a built-in R function or one you’ve created.
<code>rm(x,y)</code>	Deletes the objects x and y (can list any number of objects).
<code>seq(0,10,.1)</code>	Generates the vector of numbers from 0 to 10 in steps of .1:

	0.0, 0.1, 0.2, 0.3, ... 9.9, 10.0
<code>seq(0,10,n=50)</code>	Generates a vector of 50 evenly-spaced numbers running from 0 to 10.
<code>3:20</code>	Generates the vector of integers from 3 through 20.
<code>abs(x)</code>	Absolute value
<code>log(x)</code>	Log to base e (natural logarithm)
<code>log10(x)</code>	Log to base 10
<code>exp(x)</code>	e to the power x
<code>sqrt(x)</code>	Square root

Descriptive Statistics for a Data Vector x

<code>length(x)</code>	Number of elements in the vector x
<code>mean(x)</code>	
<code>var(x)</code>	
<code>median(x)</code>	
<code>min(x)</code>	
<code>max(x)</code>	
<code>quantile(x)</code>	5-number summary consisting of min, Q1, Q2, Q3, max
<code>cor(x,y)</code>	Correlation between vectors x and y (must be of same length)
<code>lsfit(x,y)</code>	Least squares fit of y to x

Graphics

<code>plot(x,y)</code>	Scatterplot of the vector x vs. the vector y
<code>hist(x)</code>	Histogram
<code>boxplot(x)</code>	Boxplot
<code>boxplot(x,y)</code>	Side-by-side boxplots of x and y
<code>par(mfrow=c(2,3))</code>	Splits the graphics window into a 2x3 frame

There are many options to the above commands (type “?plot”, for example, to see many plotting options) and many other graphics commands. See the manuals for more information.

8. Writing Your Own Functions

One of the most powerful features of R is the ability to create your own functions. You can create a function in a script by using the format

```
fun1 <- function(...){
  ...
}
```

where “fun1” is the name of the function, the first “...” is where you put the arguments to the function (if any) and the second “...” is where you put the commands to execute the function. A simple example of a function is given below.

Example:

Suppose we wanted to put the sample size calculation in section 3 into a function with arguments N, d, s, and prob.

Type the following in a script file:

```
sampsize <- function(N,d,s,prob=.95){
```

```

z <- qnorm(prob+(1-prob)/2)
n0 <- z^2*s^2/d^2 #intermediate calculation
1/(1/n0 + 1/N) # final result (no assignment to a variable name)
}

```

The argument `prob=.95` creates a default value for `prob` if it is not specified. Execute the script to create the function. It can then be invoked from the console:

```

> sampsize(100,2,5) # uses default probability of .95
[1] 19.36077
> sampsize(100,2,5,.99)
[1] 29.31269

```

To edit the function, edit the script which created it. You can view the function by just typing its name without parentheses:

```
> sampsize
```

9. Reading Data from a File

To read data for a single variable from a file into a vector, you can use the `scan` command. Create a text file, say `data.txt`, with the data values separated by commas, then type

```
> x <- scan('data.txt',sep=",")
```

Note that the quotes around the file name are necessary. R assumes the file is in the working directory. If it isn't, you can do one of three things:

1. you can give the complete path name to the file, e.g.,

```
> scan("g:\\data.txt")
```

Note that you must use double backslashes instead of single backslashes in the path name.

2. you can find out the working directory and put the file there. Use

```
> getwd()
```

to find out the working directory.

3. you can change the working directory by using **File...Change dir...** from the menus.

Data files with multiple variables are best read in using the `read.table` command which has options to tell how the data are formatted. There is a version of this command called `read.csv` which is designed for `.csv` (comma separated variables) files and expects data in the following format: variable names in the first row, one case per row, and commas as the separators. Excel will export data in the `.csv` format. As an example, suppose the following data on 6 farms is entered into a text file called "farm.txt" in the following format:

```

Size,Class,Workers
6,A,2
2,B,1
12,C,4
4,A,1
6,C,3
6,D,2

```

The R command to read these data in is:

```
> farm <- read.csv("farm.txt")
```

The file will then be read into a data frame. A data frame is a spreadsheet where the rows are cases and the columns are variables. You can look at the data frame by simply typing its name.

```
> farm
```

You can edit the data frame by typing

```
> fix(farm)
```

This will bring up a spreadsheet and you can make changes in the spreadsheet. You must X out of the spreadsheet before you can continue working in R.

To refer to a specific column (variable) in the data frame, you give the name of the data frame, then the variable name with a \$ sign in between:

```
> farm$Size # remember that R is case sensitive
```

If you didn't want to keep typing `farm$Size` every time you used it, you could store the values in a vector:

```
> size <- farm$Size
```

10. Obtaining Help

To obtain help with a particular function in R (say "sample"), just type:

```
> ?sample
```

For general help, look under the Help menu or refer to the many online resources.