Tutorial 1: Getting Started with R

Jason Pienaar and Tom Miller

Why R ?

R is an independent, open source statistical computing environment, incorporating implementations of a older commercial program called "S" by an international team of statisticians. Most other statistics packages are principally orientated towards applying standard methods of data analysis, which is great, but it is difficult to apply non-standard methods or to add to the capabilities of existing methods. This is primarily due to their limited programming capabilities. In addition they are pretty expensive and require new licensing agreements etc. One of the great strengths of R (other than that it is free) is the relative ease with which new capabilities can be added (the R language is really easy to use). Thus the user can easily create new functions or combine existing functions or data BUT in order to use R we need to learn the R language hence the simple tutorials.

Obtaining your own copy of R and further resources

Both PC and Macintosh versions of R can be downloaded from the R home page:

http://www.r-project.org/

It is easy to get there by simply typing r in your search engine, and selecting "the R project for statistical computing". Download and install R as follows:

1. Find the heading **download** (on the left hand side of the page).
2. Click on **CRAN.**
3. Find **USA** cran mirrors.
4. Click on the University of North Carolina's mirror (I think this is the closest).
5. Choose the version you want (i.e. mac or PC) PC = "**windows 95 or later**". For Macs, realize that you may need to update to system 10.4 or higher.
6. Follow the instructions and set-up wizard appropriate for your system.

The R home page also contains user manuals etc. The manual is very useful once you know a bit about R, but is not really that helpful as an introduction. A really easy introduction to using R for statistics is "An R and S-PLUS Companion to Applied Regression" by John Fox. Another book that is useful and has more on programming in S is "Modern Applied Statistics with S" by W.N. Venables & B.D. Ripley. Finally, several of us have found that Michael Crawley's "Statistics: An Introduction using R" is a good guide for beginners. There are also many webpages with tutorials and instructions on particular aspects of R.

By the way, the PC and Mac versions of R work identically about 99% of the time. We will try to highlight the minor differences where we spot them.

The Simplest Tricks with R:

Basic arithmetic

Open the R program by double clicking on the R icon. Under the opening message, you will find the ">" prompt, waiting for you to ask R to do something. Data analyses in R proceeds as an interactive dialogue. We type an S statement at the > prompt, press *Enter*, and the interpreter executes the statement, i.e. by returning a result, producing graphical output or sending output to a file or device. Try typing in the following simple arithmetic examples (just type what follows the prompt > and then *Enter*).

```
>2+3
>2-3
>2*3
>2/3
>2^3
>4^2-3*2
>(2-3)*(2*3)
```

The usual precedence for mathematical operators is followed (multiplication and division first, then addition and substration). In general, R ignores spaces and so they are not necessary, but for bigger expressions spaces may improve the readability.

R functions

R is a functional programming language meaning that pretty much everything we do in R is in terms of functions. R includes hundreds of built-in functions for mathematical calculations (including matrix algebra, which is extremely useful in statistics), data analyses, graphing, etc. Values passed to functions are specified within parenthesis after the function name. Here are some simple examples to try:

```
>log(100)
>log(100, base=10)
>seq(1, 4)
>seq(2, 8, by=2)
>seq(0, 1, length=11)
```

To obtain help or additional info on a function, type *?* before the function name or *help*(function name) and press *Enter*. (Note: the function *log* returns the natural log).

Variables and Vectors

Most R functions (including the simple arithmetic ones from above) can operate on more complex data structures that individual numbers. The simplest data structure (and one that we will use often) is a vector. To construct a vector use the *c* function

```
>c(1, 2, 3, 4)
```

The "*c*" function combines all the numbers you provide into a vector or a list. From now on, we will give these vectors a variable name so that we can re-use them. To do this simply assign a name to the vector using "=" symbol and press *Enter* (some may prefer the old assignment symbol of "<-", but "=" is simpler and more intuitive). Note that R is case sensitive so Vector1 and vector1 are not the same variable.

```
>Vector1 = c(1, 2, 3, 4)
```

To see what is stored in the variable simply type the variable name and press *Enter* (that is, now type "Vector1", hit return, and R will show you Vector1). The sequence operator from above (*seq*) also returns a vector. Functions applied to vectors operate on an element-wise basis. Thus:

```
>Vector2 = log(Vector1)
```

returns the natural log values of the elements stored in Vector1 and then stores them in the variable Vector2 (note: you will probably use this at some point to log-transform a variable). The rules for naming variables in S are simple: variable names are composed of letters (a-z, A-Z), numerals (0-9) and periods (.) and can be of any length (the first character cannot be a number, and spaces are not allowed). Consequently, variables can be given descriptive names so that you don't forget what the variable is. For example: "this.is.a.variable.containing.log.values.for.vector1" is a valid variable name. Stupid, maybe, but valid.

```
>this.is.a.variable.containing.log.values.for.vector1 = log(Vector1)
```

However, remember that you will have to type it out again to recall the variable. Unlike in many programming languages, variables in S are dynamically defined and redefined so we do not need to tell the interpreter how many values, what type (real, integer etc) or whether it is numeric or a character. We can also redefine a variable simply by assigning it to a different function e.g.

```
>Vector2 = rnorm(100)
```

Here the previous values in Vector2 are replaced by 100 standard-normal random numbers (the default mean = 0 and standard deviation = 1, we could easily change these defaults eg: *rnorm*(20, mean=25, sd=17) returns a vector containing 20 numbers drawn randomly from a set of normally distributed numbers with mean 25 and standard deviation 17). If we wish to print only one of the elements of a vector we index the element using square brackets as in the following example.

```
>Vector2[21]
```

returns the 21st element of the vector.

R is much more than just a calculator. It can manipulate data, conduct much of the same statistics covered in SAS, JMP, Canoco, etc., and has excellent graphic capabilities. It can also serve as a programming language. We will cover examples of these in subsequent tutorials.

Quick Exercises (Answers at the end).

1. Create a variable "random.set" containing 300 elements, randomly drawn from a normal distribution of elements with a mean of 2 and standard deviation 0.5.
2. Create a second variable that contains the natural log values of the above elements
3. Use the function *mean* to return the means of the two variables
4. Use the function *var* to return the variances of the two variables
5. The function "plot" will create a separate window on your screen with a standard labeled plot. Type *plot*(variable) to create a scatter plot of your variables against their indices, substituting your variable name into the brackets, and also *plot*(variable1, variable2) to plot your variables against each other.

Answers or Hints for selected problems:

1.
```
>random.set = rnorm(300,mean=2,sd=0.5)
```
2.
```
>log.random.set = log(random.set)
```
3.
```
>mean(random.set)
>mean(log.random.set)
```
4.
```
>var(random.set)
>var(log.random.set)
```
5.
```
>plot(random.set)
>plot(log.random.set)
>plot(random.set, log.random.set)
```