An Introductory Tutorial:
Learning R for Quantitative Thinking in the Life Sciences

Scott C Merrill

September 5th, 2012

Chapter 2

Additional help tools

Last week you asked about getting help on packages. I have spent some time looking around. At the prompt ">" you can type in

```
>  ? function.name()
```

```
>  ? package.name
```

and get nice html help about that function or package.

```
> ?? function.name()
```

Or

```
?? package.name
```

which will provide help on a list of functions or related to "function.name()"

If you want help on a package, one of the best resources is your help tab on the main menu. This tab also links to many of the nice R resources such as the manual *An Introduction to R* by Venables and Smith.

Another help tool is the function example(). This function compiles an example or multiple examples of the function that you are interested in. I have found this to be useful only when I have good pre-existing knowledge of the workings of the function. E.g.,

```
> example(histogram)
```

These examples are fairly helpful to me but likely that is only because I know what I am looking for.

Macs apparently also have some nice functionality in the form of Vignettes. From Hobbs:

3.7 Getting Help: Mac OS

As with most Mac OS programs, help is close at hand in R. Simply click on the Help button and a screen opens up with a variety of choices[8]. You have two choices, R Help and Vignettes. R Help takes you a screen with several choices; the ones you will use most are to An Introduction to R, Packages, and Search Engine and Keywords. See the Windows Help (above) for a description of these resources. Vignettes is a menu choice unique to Mac OS. Vignettes offer helpful overviews of R packages that supplement the regular package documentation, which tends to be a bit terse. Vignettes are not available for all packages.

Another great feature available only on the Mac is "Function Hints." To illustrate, type the function for doing linear regression, lm(), in the console and look at the bottom of the console window where you will see a template for the syntax for the function. This works for all R functions.

Back to the basics…

Below are some of the basic (calculator) functions in R:

Table 2.1: Basic mathematical operations in R

Operation	**Expression**	**R statement**
Addition	$a = b + c$	a = b + c
Subtraction	$a = b - c$	a = b - c
Division	$a = b/c$	a = b/c
Power	$a = b^c$	a = b^c
Square root	$a = \sqrt{b}$	a = sqrt(b) or a = b^0.5
Root	$a = \sqrt[c]{b}$	a = b^(1/c)
Exponential	$a = e^b$	a = exp(b)
Natural log	$a = ln\, b$	a = log(b)
Log base 10	$a = log_{10} b$	a = log10(b)

Datasets in R base package

We are going to use some of R's internal datasets to practice manipulating data. To let R know that you want to use their datasets compile:

```
> library(datasets)
```

Note that you do not need to download this package/library. It comes with the base R package. There are numerous data files available in this library ranging from files with 14 bits of information "BOD" to more complex data files such as the topographic data file "Volcano". Examples on the web frequently will use one of these files to illustrate the functionality of their package or function. You can compile `> ? datasets` for more information. We are going to start

with the data file “ChickWeight”, which are data examining chicken weights under different diets over time. Compile:

> ChickWeight

Your console should have filled up with the ChickWeight data (remember the R is case sensitive, and thus, compiling Chickweight will give you an error!) More about this file later.

Reading/writing data from/to an Excel csv file:

Unlike how you will likely work with your own data, we are going to work in the reverse of the natural order by writing a file to our computers and then reading the file back into R.

Writing CSV (comma separated values) files is fairly easy. There are a couple of options:

```
write.csv(ChickWeight,”ChickWeight.csv”) # This writes to your working directory
```

Alternatively, you could direct R to write to the path and of your choosing:

```
write.csv(ChickWeight,"C:\\Research\\R_Course_Directory\\DataFiles\\ChickWeight_version2.csv")
```

The write.csv function saves your data as a CSV file which can be read by almost any data management program such as Excel, or even programs such as ArcGIS (e.g, with x and y data) or Notepad.

Reading in a CSV file is equally as easy. The first step is saving your data as a CSV file. In excel this is as easy as using the save as function and scrolling down the file type until you see CSV. However, if you save an excel document as a CSV file, note that excel will only save the worksheet that you are on at that moment (other sheets will be ignored).

There are numerous methods for reading in CSV files. The one I have found to the most useful is the read.csv() function. The read.table() function is a good one as well.

You need the path to the CSV file (remember setting the working directory?). There are a couple of ways of figuring out the path.

First a warning: Don't do this! I only include this code because I have seen it a lot in other people’s code. It works but provides no record of which file you used.

```
csv.data.name <- read.csv(file.choose())
```

Below works and will provide you with a record in your script of the steps that you took. Specifically, just like when you figured out your working directory, file.choose() allows you to browse for the file you want and then copy it into the read.csv() function

file.choose()

Copy the file path from the R console

```
csv.data.name <- read.csv("C:\\Research\\R_Course_Directory\\DataFiles\\ChickWeight.csv")
```

or if it is in your working directory

```
csv.data.name <- read.csv("ChickWeight.csv")
```

As an aside, if you had copied and pasted the above line into your R editor and then compiled it, you would have received an error. Why? Because Word's smart quotes are considered different characters than standard quotes.

Being able to read and write quickly and easily to excel (or similar) is both a blessing and a curse. The problem with excel and other programs is there is no record/script of what has been done (but obviously there is some excellent functionality to excel).

Learning about data in R

Run through Tom Hobb's primer from 5 to 5.5 (stopping at 5.6 sub-setting matricies).

5 Data in R
Before you build your first model, you need to know about how R stores data. In this section you will learn about four simple structures for data: scalars, vectors, matrices, and arrays. Later, we will cover more advanced structures: lists and data frames.

There is potential for confusion here about what we mean by "data", particular for those who have some modeling background. Usually, we think of data as observations from experiments, samples, surveys, etc. R includes such observations as data, but has a somewhat broader definition. Data include values that are assigned to variables represented by its data types (i.e, scalars, vectors, and arrays). So, simulation results stored in an array, are, in R's view, data. They are not observations. I will probably jump back and forth between these views in the course—there is simply no way to be totally consistent without inventing a new word.

I presume you are sufficiently clever to juggle these uses of the term.

5.1 Scalars
The variables we have been working with above are scalars. Scalars are variables that contain a single numeric value.

[e.g., x = 2]

5.2 Strings
Strings are like scalars, except they contain character values. So, to assign a string to a variable, you will use syntax like a = "Fred".

5.3 Vectors

5.3.1 Vector basics

Unlike scalars, vectors are objects that contain more than one value. They are analogous to a row of data in Excel. So, for example consider the following in Excel:

	A	B	C	D
1	234	17	42.5	64

To create and display an analogous vector (named v1) in R enter:

```
> v1 = c(234, 17, 42.5, 64)
> v1
```

R responds:

```
[1] 234.0 17.0 42.5 64.0
```

The c( ) is a concatenate function .

In Excel you access the value of a cell using an indexing system based on letters (for columns) and numbers for rows. So, 234 is the value contained in Excel's cell A1. The indexing system in R is subtly different, but much more flexible as you will see shortly. Because vectors are like "rows" in R , to get the value of an element, you simply given the position of the element brackets [ ] next to the vector name, e.g.:

```
> v1[3]
```

will output the third element of the array v1.

It is also possible for vectors to include characters. For example:

```
study.areas = c("Maybell", "Poudre", "Gunnison")
```

Exercise: Vectors 1. What is the value of v1[2]? Create a vector called "ages" that contains the ages of everyone in your workgroup [SCM: or create a vector of the estimated ages of your introduction group]. Create a vector called names that contains the names of everyone in your workgroup. Output the value of the third element of v1 to the console. Multiply the second element of v1 by 2. Enter v1[1:3] in the console. What does the : operator do?

Table 2.2: Logical operations in R

Symbol	**Meaning**
<	less than
>	greater than

<=	less than or equal to
>=	greater than or equal to
==	equal to
! =	not equal to
!	logical not
&	logical and
|	logical or

5.3.2 Mathematical operations on vectors

Vectors can be used in arithmetic expressions. Operations are performed element by element. Try a variety of arithmetic and mathematical operations on the vector v1. For example, compile:

```
>  v2 = v1*2
>  v2
```

5.3.3 Logical operations on vectors

Here, I introduce logical operations in the context of vectors, but they have wide application in R–for example in use with if() statements, which will be covered later. Logical expressions differ from arithmetic expressions. Where arithmetic expressions evaluate to some number (the expression 2* 2 evaluates to 4), logical expressions evaluate to TRUE or FALSE. So if the value of a is 2 then a < 4 evaluates to TRUE while a > 4 evaluates to FALSE. R does logical as well as arithmetic evaluation. The symbols for frequently used logical operations are shown in Table 2.

Exercise: logical operations on vectors. Compile the following

```
> v1; v1 < 200; v1[v1 < 200]
```

Which should be the equivalent to:

```
> v1
> v1 < 200
> v1[v1 < 200]
```

What is going on here. Using logical statements like the one above, form a new vector from the from elements of v1 containing elements that are a) greater than twenty b) less than 200 and greater than 20 c) not equal to 17 d) equal to 17 or equal to 42.5

5.3.4 Vector functions

R has a rich set of functions that operate on vectors (Table 3).

Exercise: vector functions. Find the mean, max, minimum, range, and variance of v1. Find the average of the maximum and the minimum of v1. Compose another vector v2 containing 4 numeric values of your choice. Now, execute the commands min(v1,v2)

Table 3: Frequently used functions that operate on vectors in R. In the examples below, z is a vector.

Operation	**Result**
length(z)	number of elements in z
max(z)	maximum value of z
min(z)	minimum value of z
sum(z)	sum of values in z
mean(z)	arithmetic average of values in z
median(z)	median of z
range(z)	vector including minimum and maximum of z
var(z)	variance of z
sd(z)	standard deviation of z
cor(z,y)	correlation between vectors z and y
sort(z)	vector sorted
rank(z)	vector containing the ranks of elements of z

and pmin(v1,v2). Discuss with your lab mates [or imaginary friends] what the two commands are doing. The difference between min() and pmin() is really important because failing to understand it can cause mistakes that are very hard to diagnose. There is an analogous version of max() and pmax().

5.3.5 Declaring vectors

In the examples above, R figured out how long a vector is and what kind of data it contains by the assignment you gave it, that is by what was on the right hand side of the assignment operator. It is also possible to create a vector of zeros, which can be very useful if you want to create a data structure that will be assigned values latter. Here are some ways to do that.

Most often, we will use something like

```
> v = numeric(10)
```

where 10 is simply the number of elements/cells in the vector, its length. (Remember, I am using v as a variable name here, it could be anything you want to name the vector.) This statement says "create a vector that can hold 10 numbers."

It can be useful to find out the length of a vector, which is accomplished with the statement:

```
> length(v)
```

It is also possible to create a vector of integers 11 using:

```
> v = integer(10)
```

[SCM] Why would you do this? Storing integers (i.e., not fractions) requires less memory and can make the program run faster. Also, this is nice if you want to make sure a number doesn't somehow get designated as a fraction.

Exercise: Put the following code in a script and run it line by line. Explain what is going on to your lab mates.

```
> d = numeric(5)
> d
> (d + 1) /3
> length(d)
```

What do you do if you know you want to declare a vector, but you don't know how long it should be? You can let R know that it should recognize v as a vector and allow you to assign values to any element in the vector. For example, compile:

```
> v=NULL
> v[8]=254
> v[2]=10
```

The symbol NA is used to indicate missing data, more about this soon. Explain what is going on here. Now enter v[12]=13.4 and explain the result.

5.4 Matrices and arrays

Vectors are analogous to rows in a spreadsheet. Matrices are analogous to sheets; three dimensional arrays are analogous to workbooks. First we will work on matrices and then turn to arrays of higher dimension. The matrix will be a true workhorse in this course.
Matrices are special types of two dimensional arrays; more about arrays later. They differ from arrays because certain mathematical operations, for example, transforms, are valid only for matrices.

5.5 Creating matrices

Consider the following bit of data contained in Excel:

	A	B
1	1995	10.0
2	1996	12.5
3	1997	15.6
4	1998	19.5
5	1999	24.4

In R, as in Excel, specifying the value in a "cell" now requires both rows and columns. In R, this is done by the order of the indices. The first index gives the row number, the second index gives the column. One way to remember the order is to think Row Column, RC, Roman Catholic.

To create matrix analogous to the rows and columns in the sheet, begin by using syntax like this:

```
> A = matrix(0,nrow=5, ncol=2)
```

This syntax says "Create an matrix named A with 5 rows and 2 columns containing the value 0 for all elements." (It doesn't have to be named A, you could call it Bookshelves if you want to.)

Exercise: Creating a matrix

Create and output a matrix named B that contains seven columns and 5 rows with all elements containing the value 7.

Working with rows and columns of a matrix R has exceptionally powerful ways to work with matrices using commas to designate entire rows and columns, effectively making them vectors.

For example, the syntax:

```
> A[,2]
```

specifies all of the elements in the second column of the matrix A, while

```
> A[4,]
```

specifies all of the elements of the fourth row of the matrix A. Remember, rows and columns treated this way create vectors. You will use this capability a lot in this course, so it is truly critical that you understand it.

Exercise: matrices. Enter the data in the Excel screenshot above into an matrix called M. Write and execute a script that

1. Sets up the matrix M and fills it with zeros using the matrix() statement.

2. Enters the appropriate values in the matrix row by row Hint: to enter the first row use: `A[1,] = c(1995,10.0)`

3. Outputs the entire matrix

4. Outputs each row

5. Outputs each column.

6. Finds the largest value in the matrix

7. Finds the average of the second column