

This lab guide will give a first look at using R statistical software. It includes:

- hints on setting up a new project in R
- simple arithmetic
- basic arithmetic and statistical functions built in to R, including `sum()`, `mean()` and `sd()`
- using vectors, including data types, named vectors, and indexing
- data frames and factors
- the use of scripts in R
- tidying up and saving your work, and closing R at the end of session.

You should have already downloaded R and installed it on your computer before starting this session. Go to [http://www.wcsmalaysia.org/stats/Software\\_summary.htm](http://www.wcsmalaysia.org/stats/Software_summary.htm) for information and links on this.

## ***Managing separate projects***

You will probably want to use R with a range of projects. A good way of keeping track of the different projects is to create a separate folder for each and to put a copy of the “.Rdata” file in each. Then you can navigate to the appropriate folder (via My Computer) and double-click on “.Rdata” to start R.

Let’s start off on the right foot by setting up a new folder for this work.

Create a new folder and name it something like “Starting R”. Now find the file “.Rdata” or a file with no name<sup>1</sup> but a blue “R” icon, which will probably be in the folder “C:\Program Files\R\<version>\bin”, where <version> is the current version of R, something like “R-2.8.1”; if it’s not there, try using the Search facility in Windows. Copy the “.Rdata” file and paste into your new “Starting R” folder.

Start R by double-clicking on “.Rdata” or the blue “R” icon in your “Starting R” folder. “Starting R” becomes the default folder for this R session, and the results will be saved there when you exit R.

## ***Simple arithmetic***

Inside the main window (named “RGui”, Gui = graphical user interface) is a smaller one called “R Console”, which is where you do your work in R. It has a copyright notice and some hints. Then there is a red wedge (**>**), the prompt, and a vertical red line (**|**), which is the cursor. The cursor changes to a red rectangle (**█**) when you switch from insert mode to overwrite mode.

R is a rather fancy calculator, and it will do all the simple calculator-type things too.

Start with some simple arithmetic: type “2 + 3” and press enter:

```
> 2 + 3  
[1] 5  
> |
```

Notice that your input is red, R’s output is blue (in this document your input is bold).

Don’t worry about the “[ 1 ]” for the moment, that will come clear later.

---

<sup>1</sup> If the name isn’t visible, go to ‘Tools > Folder Options’ in My Computer, select the ‘View’ tab, and uncheck “Hide extensions for known file types”.

Try some other sums:  $2 * 3$ ,  $2 / 3$ , etc, etc. The symbol  $\wedge$  means “to the power”, so  $3^2$  means “3 to the power 2” or “3 squared” – try it. R recognizes brackets: try  $2 + 3 * 5$  and  $(2 + 3) * 5$  and you’ll get different answers.

Try using the up-arrow and down-arrow ( $\uparrow$  and  $\downarrow$ ) on your keyboard. Go up until “ $2 + 3 * 5$ ” appears on the bottom line, then put in extra brackets to make “ $2 + (3 * 5)$ ” (you’ll have to use the left and right arrows,  $\leftarrow$  and  $\rightarrow$ , to move along the line, you can’t use the mouse).

I use the up-arrow a lot to avoid retyping complicated formulae or object names.

## Creating objects

R can store numbers in named *objects*.

To store “3” in the object “my”, type “`my <- 3`”. “`<-`” (a less-than sign followed by a hyphen or minus sign) is called the *assignment* operator, but it’s easier to read it as ‘gets’: “my gets 3”. Now type “my”:

```
> my <- 3
> my
[1] 3
> |
```

Object names must start with a letter and can include letters, numbers and dots. R is case sensitive: “my”, “My”, “MY” and “mY” are all different objects.

R treats objects just like numbers: try typing “`my + 5`”, “`my * 5`”, etc

You can also assign the result of a calculation to an object, like this:

```
> total <- 3 + 7 + 9
> total
[1] 19
> |
```

To see a list of the objects you have created, select “Misc > List objects” from the pull-down menus.

The list which appears will include the objects you just created – my, total, prn and consec.

To remove an object, use ‘rm’:

```
> rm(total)
> total
Error: object "total" not found
```

## Creating vectors

A neat feature of R is that an object can store more than one number. To assign them, you need to use the *concatenation* function, `c()`.

Try this

```
> prn <- c(2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, 41)
> prn
[1] 2 3 5 7 11 13 17 23 29 31 37 41
```

An object which contains a row of numbers like this is called a *vector*. You can now do arithmetic with all the numbers in a vector at the same time:

```
> prn/3
[1] 0.6666667 1.0000000 1.6666667 2.3333333 3.6666667 4.3333333
[7] 5.6666667 7.6666667 9.6666667 10.3333333 12.3333333 13.6666667
```

Notice that the second line of output begins with the 7<sup>th</sup> number in the vector, as indicated by the [ 7 ] at the beginning of the line.

If you want a series of consecutive numbers, eg 3, 4, 5, 6 and 7, you can't use "3-7" as R will understand that to mean "3 minus 7"; instead you use "3:7" :

```
> consec <- 3:7
> consec * 2
[1] 6 8 10 12 14
```

R has many *functions* which use vectors: you can add up the numbers in `prm` with `sum(prm)`. Also try `mean(prm)`, `max(prm)` and `min(prm)`. `length(prm)` tells you how many numbers (or *elements*) the vector contains.

You can get help on a particular function by typing `?` and the function name at the prompt:

```
> ?max
```

If you don't know the name of the function you want, use 'Help > Search help...' from the pull-down menus.

You can get at individual elements of a vector by using an *index* placed in square brackets `[]` after the object name. So the third element in `prm` is `prm[3]`:

```
> prm[3]
[1] 5
```

Try typing `prm[-3]` : this gives all the elements except the 3<sup>rd</sup>. Try `prm[1:4]` : it gives the first 4 elements.

## Data types

Data can be in the form of numbers, character strings or true/false observations.

```
> vn <- c(3.4, 5.6, 7.8)
> vn
[1] 3.4 5.6 7.8
> vc <- c("Melvin", "Cynthia", "Jason")
> vc
[1] "Melvin" "Cynthia" "Jason"
> vl <- c(T, F, T)
> vl
[1] TRUE FALSE TRUE
```

Note that character strings need quotes (single or double) and you can use the abbreviations T and F for TRUE and FALSE.

You can't mix data types in the same vector; R will change the type of data to something that makes sense, so for the following input

```
> c(1, 2, "three", TRUE)
[1] "1" "2" "three" "TRUE"
```

only the character type makes sense.

You can give names to the elements of a vector. For example:

```
> names(vn) <- c("Jan", "Feb", "Mar")
> vn
Jan Feb Mar
3.4 5.6 7.8
```

This can be a useful way of formatting data: for example, the data set for the areas of large islands looks like this:

```
> islands
      Africa      Antarctica      Asia      Australia
      11506         5500      16988         2968
etc etc
```

Use `?islands` to see details of this data set.

## Data frames and factors

We usually want to arrange our data into a table. For example, data on girth, height and volume for black cherry trees looks like this (use `?trees` to see details of the data set):

```
> trees
  Girth Height Volume
1    8.3    70  10.3
2    8.6    65  10.3
3    8.8    63  10.2
... etc etc
```

These data are in a class of object called a data frame

```
> class(trees)
[1] "data.frame"
```

A useful feature of a data frame is that you can access the individual columns with the ‘\$’ operator:

```
> trees$Height
 [1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64 78 80 74 72 77
[26] 81 82 80 80 80 87
> mean(trees$Height)
[1] 76
```

In the ‘trees’ data frame, all the columns are numeric, but this is not necessary. We can combine numeric, character and logical data in the same data frame (though not in the same column!):

```
> data.frame(vl, vc, vn)
      vl      vc vn
Jan TRUE  Melvin 3.4
Feb FALSE Cynthia 5.6
Mar TRUE   Jason  7.8
```

In this case, ‘vn’ is a named vector, and the names are used for the row names of the data frame.

Categorical data can be encoded in R as factors, and this is especially useful in data frames. For example, look at the first few rows of the ‘chickwts’ data frame:

```
> head(chickwts)
  weight      feed
1   179 horsebean
2   160 horsebean
3   136 horsebean
4   227 horsebean
5   217 horsebean
6   168 horsebean
```

These are the growth rates of chicks with various feed supplements (use `?chickwts` to see details). We can get a summary of the data with:

```
> summary(chickwts)
  weight      feed
Min.   :108.0  casein   :12
1st Qu.:204.5  horsebean:10
Median :258.0  linseed  :12
Mean   :261.3  meatmeal :11
3rd Qu.:323.5  soybean  :14
Max.   :423.0  sunflower:12
```

The ‘feed’ column is a factor with 6 levels for the 6 types of feed:

```
> levels(chickwts$feed)
[1] "casein"      "horsebean"  "linseed"    "meatmeal"   "soybean"    "sunflower"
```

Factors are useful for variables such as male/female, infant/juvenile/adult, or control/treatment.

## Using scripts

A ‘script’ in R is simply a text file containing the same commands that you would type into the R Console. Look at the example in the file “example\_script.R”: on the main menu in R, use File > Open script... and browse to “example\_script.R”.

Place the cursor anywhere in a line and press Ctrl-r; the line appears in the R Console and the command is run. (The lines beginning with “#” are comments which do nothing in R.)

As you will discover, I use scripts a lot! I rarely type anything directly into the R Console; instead I have a script file open, type the commands into the script, then use Ctrl-r to run them. I save the script, so that I have an exact record of the calculations I did, and I can go back to it, review it, and change it if necessary. This is essential if you are doing an analysis of real data for a report or thesis.

## Getting help

As we’ve seen, you can get help on a specific function or data set in R with the question mark:

```
> ?mean
> ?islands
```

If you don’t know the name of the function or data set, you can use double question marks:

```
> ??average
> ??oats
```

which opens a window with a list of occurrences in the R help files.

And of course you can consult the manuals and FAQ files via the Help menu. You might find Help > Manuals (in PDF) > An Introduction to R useful.

## Getting finished


I rarely save the objects in the “workspace” when I exit R; instead I save the commands I used to create the objects, so that I can quickly recreate them next time.

If you do want to save the workspace, it’s a good idea to tidy up first.

Use ‘Misc > List objects’ from the pull-down menu bar to see a list of the objects in the workspace.

Use the remove function `rm()` to get rid of objects you don’t need to save (note the use of quotation marks):

```
> rm("my", "total", "prm", "conseq")
```

The easy way to exit is to click on the  in the top right corner of the RGui window (or you can select ‘File > Exit’ from the pull-down menus, or press Alt-F4). R will ask if you want to “Save the workspace image?” If you click on Yes, it will be saved in the .RData file and reloaded next time you start R by clicking on that file icon.

Most software packages allow you to choose the name for the file when you save it. **R doesn’t!** It always saves the data in “.Rdata”, and if a file called “.Rdata” already exists, it will be overwritten; R will not ask before overwriting. So you can’t keep data from different projects in R files with different names: that’s why I recommend that you use separate folders.