Introduction to R

Rachael Smyth and Andrew Johnson

Preamble

R is an open-source statistical programming language. Unlike statistical analysis software like SPSS, R is more like a programming language than software - and although this makes R more complicated to use than SPSS (at least initially), it offers a considerably more personalized and flexible analysis platform. And… there are a number of free archives for R syntax, and new syntax libraries ("packages") are being created and updated every day.

But best of all… R is totally free. You will be able to use the analytic strategies and syntax files that you learn about in this and other statistics courses, long after you graduate from your degree. This is probably one of the reasons that R is one of the fastest growing analytic platforms for statistical analysis. Thus, the time that you spend now, developing analytic routines and procedures, is a long-term investment in transferable skills.

Downloading R

Your first step in learning to use R is downloading it for your own computer. There are versions of R that have been compiled for Mac, Windows, and Linux, and you can find them all here:

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

Click on your operating system, and then select the proper version of R, based on your system's characteristics. You should check back here every few months to see if there is a new version of R available for download. The labs in this module were created and tested in **R version 3.2.2 (2015-08-14)**, which was nicknamed **Fire Safety**.

Downloading RStudio

You can carry out all of the labs and assignments in this course using just the R Console that you have now downloaded. You will, however, probably enjoy using R much more if you download RStudio. This is not a different version of R - it is an environment for running R. It provides you with a single application window in which you can type syntax, view data objects and loaded functions, view plots, loaded packages, and manual pages for your functions, and even manipulate files on your computer.

To download RStudio, go here:

http://www.rstudio.com

You probably want the desktop version of RStudio. When you click on **Download RStudio Desktop** you will be directed to a list of possible downloads. Select the one that matches your operating system. Although the labs in this course focus mainly on commands that can be typed directly into the console, you should spend some time acquainting yourself with RStudio - it is much more than a console. For example, all of the handouts in this graduate module were entirely created in R Markdown language…

Using the R Console

You can use the R console in a number of different ways. You can use it as a calculator, for example. To illustrate this, try typing `6*10` into the console (and then pressing enter).

You should see the following:

```
6*10
```

```
## [1] 60
```

The answer is 60.

You can also create variables within the R console. For example, consider the following:

```
x <- 4
```

We have now set `x` equal to 4. If you are using RStudio, you will see the value `x` come up in the environment window (in the upper right-hand corner of your screen), and it will have the number 4 beside it.

We can set `y` equal to 15:

```
y <- 15
```

and we can then use these two variables (x and y) to create a new variable, `z`:

```
z <- x * y
```

Finally, if you want to display the value of your newly created variable, you would simply type it into the console:

```
z
```

```
## [1] 60
```

Creating Numeric Vectors

Objects in R can (obviously) contain more than a single value. If you have more than one number in a variable, we can create an object that creates a vector. For example, if you had BMI values for 20 different people you could combine them into one object. Let's call that object `BMI`.

```
BMI <- c(36.8, 28.1, 18.1, 31.0, 17.2, 31.9, 19.6, 18.6, 21.9, 35.7,
         32.7, 25.2, 25.2, 38.4, 35.2, 34.2, 26.5, 22.1, 39.2, 16.0)
```

If you are entering this directly into the console, you should not press enter until you've entered all of the information within the parentheses. To combine these values, we used the function `c`, which (intuitively enough) stands for "combine". Individual values inside the parentheses are separated by commas, and all of these become values within the `BMI` object. Note that R is case-sensitive, and so the object is `BMI`, not `bmi`. In other words, if you typed `bmi` into the console, R would not assume that you meant `BMI`.

Creating Non-numeric Vectors and Factors

Objects in R can hold either text or numbers. Let's create a vector of participant names, to go along with the BMI data that we just entered:

```r
subjectnames <- c("Bob", "Mary", "John", "Doug", "Lee", "Luke", "Leia", "Sarah",
                  "Toby", "Rachel", "Jack", "Katie", "Darth", "Jill", "Gwen",
                  "Peter", "Jane", "Liz", "Barb", "Penny")
```

And, because it could be useful when conducting group comparisons, let's create a variable that contains participant sexes:

```r
sex <- c("M", "F", "M", "M", "M", "M", "F", "F", "M", "F", "M", "F", "M",
         "F", "F", "M", "F", "F", "F", "F")
```

Both `subjectnames` and `sex` are now "character vectors", because they contain non-numeric information. For the `subjectnames` vector, we will probably only be using this information to describe individual participants (i.e., it is unlikely to be involved with any actual analyses). If we wanted to use the `sex` variable as a grouping variable in a statistical analysis (such as a t-test), we might want to turn it into a "factor" by using this command:

```r
sex <- factor(sex)
```

This object is now a "factor" with two levels ("F", and "M"), and so it will behave differently in analyses, and will display differently in the console:

```r
sex
```

```
##  [1] M F M M M M F F M F M F M F F M F F F F
## Levels: F M
```

Factors can be either numeric or non-numeric - but in most group-wise analyses, we use non-numeric factors, as it coerces R into labeling the output in more intuitive and easy to read ways. More on this as we move through the other R labs in this module.

Working with Objects in R

You should see the object `BMI` in your environment window, and it should indicate that this object is a numeric vector with 20 values in it (the first part of the description should be 'num [1:20]'). If you want to view the contents of this vector object in your console window, you can simply type `BMI`.

```r
BMI
```

```
##  [1] 36.8 28.1 18.1 31.0 17.2 31.9 19.6 18.6 21.9 35.7 32.7 25.2 25.2 38.4 35.2 34.2 26.5
## [18] 22.1 39.2 16.0
```

If you want this output to show up on multiple shorter lines, you can set the width of your display by typing `options(width=40)`. This sets the width of your display to be 40 columns.

```r
options(width=40)
```

Now if you type `BMI` again, your table of values will be more compact.

```r
BMI
```

```
##  [1] 36.8 28.1 18.1 31.0 17.2 31.9 19.6
##  [8] 18.6 21.9 35.7 32.7 25.2 25.2 38.4
## [15] 35.2 34.2 26.5 22.1 39.2 16.0
```

The square brackets on the left-hand side of the screen indicate the "index" of the first value on that line. tell you what position in the order it is that is at the start of the row. For example, you can see that the number 18.6 is the 8th value in the `BMI` object, and 35.2 is the 15th value.

If you want to pull out specific values from the object, you can do so by typing the position you want into square brackets after the name of the object. For example, if you wanted the 3rd value from the `BMI` object, you would type:

```r
BMI[3]
```

```
## [1] 18.1
```

and if you wanted values 3, 7, and 10 to 15, you would use the `c` function to combine all of those indices:

```r
BMI[c(3,7,10:15)]
```

```
## [1] 18.1 19.6 35.7 32.7 25.2 25.2 38.4
## [8] 35.2
```

You can easily perform basic descriptives calculations (e.g., mean, median, and standard deviation) on your vector, using the built-in statistical functions in R:

```r
mean(BMI)
```

```
## [1] 27.68
```

```r
median(BMI)
```

```
## [1] 27.3
```

```r
sd(BMI)
```

```
## [1] 7.651598
```

In each case, the "command" that you are using calls a function that is currently loaded by R. For example, when we type `mean(BMI)`, we are calling the function `mean` on the object `BMI`. R has many functions built into the packages that are loaded by default with every new installation - but as you will see, later on in this module, there are a number of specialized functions that can be installed with other packages that are freely available (most commonly within the CRAN archive from which you downloaded R at the outset of this lab). If you encounter a package that you need to use, but don't have installed on your machine, you can download it from the CRAN archives, directly within your console, using the `install.packages` function. For example, one of the packages that we will be using extensively within this module is the `psych` package, authored by William Revelle. You can download this package using the following syntax:

```
install.packages("psych")
```

You now have the most recent version of the `psych` package installed on your computer. We don't need this package for the demonstrations that we will be working through in this lab, but if you wanted to load the `psych` package now, you could do so using the `library` function:

```
library(psych)
```

Creating Datasets

We now have three objects in our environment that we can use for analyses. Although we can use most functions with the information left separate in this fashion, it is frequently desirable to combine variables to create a dataset. In this way, we can summarize individual cases across multiple variables using indices in the way that we described earlier.

R calls datasets "dataframes", and we can combine our variables into a dataframe using the following code:

```r
BMIdata <- data.frame(subjectnames, sex, BMI)
```

You will now see an object called `BMIdata` in your environment window, that indicates a dataset with 20 observations of 3 variables. The `head` function can be used to get a snapshot of the data (i.e., the variable names and the first 6 rows of data):

```r
head(BMIdata)
```

```
##   subjectnames sex  BMI
## 1          Bob   M 36.8
## 2         Mary   F 28.1
## 3         John   M 18.1
## 4         Doug   M 31.0
## 5          Lee   M 17.2
## 6         Luke   M 31.9
```

Note that each line of the dataframe is a single subject in the data, and we have now linked participant sex and BMI within a single object.

To access specific values within this dataframe, we will again use indices to specify the position of the value we are looking for. Because our object now has both multiple rows and multiple columns, we need a second parameter to specify the coordinates of the value we are looking for. Within the square brackets, the first value indicates the row within the dataframe, and the second value indicates the column. Thus, if we wanted to look at the BMI (values for which are found in the third column) of the fourth participant, we would use the indices `[4,3]`:

```
BMIdata[4,3]
```

```
## [1] 31
```

Or, if we wanted all of the data on "Luke" (the 6th participant), we could do the following:

```
BMIdata[6,]
```

```
##   subjectnames sex  BMI
## 6         Luke   M 31.9
```

Because we didn't specify any columns after the comma in our index specification, R provides us with all of the columns. Likewise, we could generate a vector with all of the BMI data in it (in other words, exactly the same as the `BMI` vector that you used to create the `BMIdata` dataframe) as follows:

```
BMIdata[,3]
```

```
##  [1] 36.8 28.1 18.1 31.0 17.2 31.9 19.6
##  [8] 18.6 21.9 35.7 32.7 25.2 25.2 38.4
## [15] 35.2 34.2 26.5 22.1 39.2 16.0
```

Or, because we have named the individual variables within our dataframe, we could pull out the BMI data as follows:

```
BMIdata$BMI
```

```
##  [1] 36.8 28.1 18.1 31.0 17.2 31.9 19.6
##  [8] 18.6 21.9 35.7 32.7 25.2 25.2 38.4
## [15] 35.2 34.2 26.5 22.1 39.2 16.0
```

We can also pull out the BMI values for all of the women within the data, using this command syntax:

```
subset(BMIdata, sex=="F")
```

```
##    subjectnames sex  BMI
## 2          Mary   F 28.1
## 7          Leia   F 19.6
## 8         Sarah   F 18.6
## 10       Rachel   F 35.7
## 12        Katie   F 25.2
## 14         Jill   F 38.4
## 15         Gwen   F 35.2
## 17         Jane   F 26.5
## 18          Liz   F 22.1
## 19         Barb   F 39.2
## 20        Penny   F 16.0
```

Note that we are using TWO equal signs in this function, as we are effectively telling R that we want a subset of `BMIdata` in which "Sex is exactly equal to F".

Closing Thoughts

We are really just scratching the surface with these examples - and you will only get better at using R by actually using R. As you develop in skill, and begin to use more complicated syntax and function calls, you should start working in script files. This allows you to tweak your analysis over multiple lines of syntax, and also provides you with an audit trail so that you can re-trace your steps to determine what you have actually done within your analysis (thus ensuring that you can replicate your results!). You don't need RStudio in order to run scripts, but you will probably find that it helps, given the extent to which R is integrated within RStudio. You can, for example, run your code in the console, directly from the script window in RStudio.

And, of course, you can save all of your script files so that you can begin accumulating a library of analysis workflows. If you do this throughout your work in this course, you should have accumulated a library of R script files that will facilitate your own psychometric evaluation of tests and measures, within your own research.