Roith

Introduction to R: 1

What is R?

R is a statistical software that allows users to perform quick computations and statistical tests. It also allows users to write and create their own tests or unique programs and share them within the R community.

When you open RStudio, there are four main windows that appear, here is a brief description of each window.

Bottom Left: Console

This is the main user interface for R, where you type or paste the commands that you would like to be run. If code is not written in a way that is recognized in the R language, an error message will be returned, usually with some indication of where the problem is. Numerical and other non-graphical output is also displayed in this window.

Top Left: R Script

By default this window is usually named Untitled. It is a notepad where you can write code out before having it run. This is helpful when writing functions and checking code for bugs before submitting it. You can run the whole script at once, or you can highlight what you would like to use. You can also save code in an R Script and use it again in another session.

Top Right: Environment/History

The Environment tab lets you know what objects or data are currently in R. So if you try to import a data set called TestScores, if done correctly it will be listed in this window. The History tab is straightforward, it will display all the previous code run in R.

Bottom Right: Files/Plots/Packages/Help/Viewer

For the most part we will only use the Plots, Packages, and Help tab in this window.

Plots: Any graphical output (scatterplots, histograms, barcharts, etc.) from code you run will show in this window. You can copy using Export and paste a graph into a word file or other document.

Packages: Sometimes there are functions or datasets we would like to use that are not in the basic R program. You can install these as packages as long as you know the name of the package. This allows you to use code that someone else has written instead of starting from scratch and developing your own every time.

Help: If you ever want to know what a function does or see some examples of it being used, you can search for it by name in the Help tab. Also, in the Console window you can write the name of a function with a question mark in from of it to view the help page. (Ex. ?mean)

Using R

At its core R is simply a calculator. We can have it perform basic arithmetic like $6 + 5 \cdot 4$

```
> 6+5*4
[1] 26
```

As a rule, R will always follow the order of operations (Parenthesis, Exponents, Multiply/Divide, then Add/Subtract). So it is important that mathematical expressions be entered into R as you actually intend them to be. In the expression above, if I intended to perform the addition first, I need to include a parenthesis.

```
> (6+5)*4
[1] 44
```

Arithmetic Operators	**Symbol in R**
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^ or **
Square Root	sqrt( )

Exercise 1: I would like to calculate the value for the following mathematical expression:

$$-8 \cdot 7^{\frac{2}{3}} + \sqrt{4(9 + 6.2)}$$

Find the value for the following expressions in R and choose the one that would be correct for the above statement:

```
-8 * 7^2/3 + sqrt(4 * 9+6.2) =          _____________
-8 * 7^2/3 + sqrt(4 * (9 + 6.2)) =      _____________
-8 * 7^(2/3) + sqrt(4 * (9 + 6.2)) =    _____________
(-8 * 7)^2/3 + sqrt(4 * (9 + 6.2)) =    _____________
-8 * 7^(2/3) + sqrt(4(9 + 6.2)) =       _____________
```

Could this also be considered an acceptable way to find the correct value?

```
(4 * (9 + 6.2))^0.5 + 7**(2/3) * (-8)
```

So you can see that even using R as a calculator can become very complicated quickly, keep track of your parentheses and remember the order of operations.

Objects

R is considered an object-oriented programming language. This is because of its ability to *assign* values to **objects**. For instance, if there is a specific value from a calculation that we want to use multiple times, we can assign it to a unique letter/word/shorthand of our choosing. Then we can refer to it later and/or use it in a different statement. Use "<-" or "=" to assign a value to an object.

```
> value <- 6+5*4

> value
[1] 26

> value/2 - 3
[1] 10
```

There are rules to naming objects in R:

1. object names:

Can contain	**Cannot contain**
Letters: a-z, A-Z	Operators: + - / * ^ ( )
Numbers: 0-9	Logical Operators: < > = ! | & ?
Some symbols: . _	Most other symbols: [ ] { } \ , $ # @ % or spaces

2. objects cannot start with a number.
3. names are case sensitive (as is everything else in R).
    a. so "Time" and "time" are two different things.
4. an object will stay in the *environment* (and be available) unless you rename it or remove it.
    a. renaming involves assigning an already existing object a new value

```
> one <- 1
> one <- 2
> one
[1] 2
```

    b. you can remove an object by using the rm() function

```
> rm(one)
> one
Error: object 'one' not found
```

Exercise 2: State whether the following object names would allowed in R or not.

first_quarter	_____	Fourth.qtr	_____
second quarter	_____	freethrow%	_____
3rdquarter	_____	.	_____

There is a certain etiquette and style to writing object names in R, try to make things short and simple but still clear so other people can look at your code and know to what you are referring.

Vectors

R is not limited to dealing with one value or operation at a time. It is very versatile and can handle thousands of computations at the same time relatively easily. One common way to do this is to use **vectors**. A vector is a set of numbers or words, all assigned to one object. The simplest type of vector is just a sequence of numbers, the ":" between two numbers indicates you would like to create a set of values starting at the first number and increasing/decreasing to the second by increments of 1.

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1

> e <- 2:-3
> e
[1]  2  1  0 -1 -2 -3
```

Other ways to generate numbers are by using the repeat and sequence functions, rep() and seq(). These need a few *arguments* included in order to work properly. For the rep() function, you must specify the value that you want to be repeated, and then how many times it should appear. Using the seq() function, you must state the starting point (from =), the ending point (to =), and the increment (by =).

```
> rep(2, times= 6)
[1] 2 2 2 2 2 2

> seq(from= 4, to= 5, by= 0.2)
[1] 4.0 4.2 4.4 4.6 4.8 5.0
```

When you get more comfortable with the code language, you can eliminate some of the extra words, as long as you know the order of the arguments.

```
> rep(2, 6)
[1] 2 2 2 2 2 2
> seq(4, 5, 0.2)
[1] 4.0 4.2 4.4 4.6 4.8 5.0
```

The most common way to create a vector, especially when the numbers are not in a sequence or repeated is to use the concatenation function c(). This will combine any set of numbers, letters, words, or other vectors together.

```
> c(23, 14, 41.5, 0.8)
[1] 23.0 14.0 41.5  0.8

> m <- c(23, 14, 41.5, 0.8)
> n <- 1000

> c(m, n)
[1]   23.0   14.0   41.5    0.8 1000.0
```

Exercise 3: These functions can be used together to create vectors, type the following code in R and write down the output you get:

```
rep(1:3, times= 4)
```

```
rep(1:3, times= 1:3)
```

```
rep(seq(from= 0, to= 100, by= 25), times= 2)
```

```
c(rep(1,4), seq(7, 4, by= -0.5))
```

```
rep(c(45, 50), times= c(3, 5))
```

Exercise 4: Write code that will produce the following vectors in R:

30 29 28 27 26 25 24

-10 -8 -6 -4 -2 0 2 4

3.14 1.56 -74 15000

5 4 3 2 5 4 3 2 5 4 3 2