Introduction to R: The Basics

Tom Barber

The goal of this first document is to gain familiarity with `R` and produce your first R Markdown document. We will start with simple R Code and provide an explanation of what the it does. In R Studio you have the ability to take this code and "Knit" it into a well formatted document. Make sure to select "Knit to PDF". I recommend modifying any of the code, experiment and see what changes. You might need to look up the documentation of some of the functions (type `help(fn)` into the console, where `fn` is the name of the function).

```
1 + 0.0000001
```

The above code adds 1 and 0.0000001 together.

```
1 + 0.0000001 == 1
```

The code adds 1 and 0.0000001 together and then uses the "==" to check if the answer is equal to 1. It is not, and so we should get a FALSE output.

```
1 + (0.0000001 == 1)
```

We see above that the code checks if 0.0000001 is equal to 1. Since it isn't we get a FALSE value, which in R is represented with a 0 value. We then add 1 to it and should have an answer of 1.

```
print(1 + 0.0000001, digits = 8)
```

This code prints the answer of 1 added to 0.0000001 and uses the digits input to print the answer to the 8th digit.

```
any(c(TRUE, FALSE, TRUE, TRUE))
```

We use the any function to check if the input contains any values corresponding to values of TRUE. In this case the input contains at least one true value and so we get an output of TRUE.

```
all(c(TRUE, FALSE, TRUE, TRUE))
```

In a similar sense to the code above the all function takes the input and looks for TRUE values. In this case the all function needs every value to return a true, but sense we see there are FALSE values then this is not satisfied. Thus the output is false.

```
sum(c(TRUE, FALSE, TRUE, TRUE))
```

The sum function takes all values of the array/vector created and adds them together. Since R encodes False as 0 and True as 1 we get a value of 3.

```r
x <- c(6, 8, 9)
c(x[1], 7, x[2:3])
```

The code first creates a vector and assigns it to the variable x. The second line then creates another vector made from the first value of x, then 7, and finally the second and third values of x. Essentially this appends 7 in between 6 and 8 from x and we get 6, 7, 8, 9.

```r
x <- c(6, 8, 9)
length(x)
```

Since we already know what the first line does from the explanation above the only other pertinent information here is that the length function then finds the number of elements in x, so 3.

```r
x <- c(6, 8, 9)
length(x[0])
```

In this case the length function should return a 0 value since R does not start indexing at 0 there will be no corresponding value and so the number of elements is 0.

```r
x <- c()
length(x)
```

This problem reassigns a vector with nothing in it to the variable x. Thus when finding a length we get 0 because there are no elements.

```r
x <- c(2,3,4)
print(x^2)
```

This code reassigns a vector with new inputs to x and then prints the square of each element of x, so we get 4, 9, 16.

```r
x <- c(2,3,4)
x * x
```

This code is just another way to write the same problem as above, multiplying x to itself. Thus we get 4, 9, 16 again.

```r
x * x
```

This is the same as the code above in terms of multiplying x by itself, essentially getting x squared. The only difference is the initial assignment of x isn't written. This is fine because its been assigned already and stored in the global environment.

```r
x <- matrix(c(2,10,0,1,1,0), nrow = 2)
y <- matrix(c(1,0,0,1), nrow = 2)
t(x) %*% y
```

This code defines two matrices x and y, both of which are defined as having two rows. It means x is a 2x3 matrix and y is 2x2. It then takes the transpose of x to allow for inner dimension alignment and then performs matrix multiplication.

```
y <- matrix(c(1,0,0,1), nrow = 2)
solve(y)
```

Above we define a 2x2 matrix and use the solve function on it. This will return the inverse of the matrix.

```
y <- matrix(c(1,2,2,1), nrow = 2)
solve(y)
```

Similar to the above code, except we redefine the matrix with different values. Doing this allows us to use the solve function to find a less obvious inverse matrix.

```
y <- matrix(c(1,2,2,1), nrow = 2)
z <- solve(y)
print(y %*% z)
```

This code takes a matrix y, solves it, and then stores it as the variable z. The code then prints the answer of the two matrices that have been matrix multiplied. Since this is a matrix multiplied by its inverse we should get the identity matrix of the 2x2 matrix.

While I am confident in the code and following explanations, I understand that mistakes are always possible. Regardless, this document will give a basic understanding into the beginnings of R coding, R Studio, R Markdown, and statistical analysis in code.