Seminar 2

Basics in R

Today, we will go through a brief introduction to R console: (1) expressions, (2) variables, and (3) vector. Generally, we will be using R console for this purpose. The command driven interface looks a bit intimidating; however there are good reasons to stick to it.

First of all, command driven interface is very flexible and it is better suited for data manipulation and analysis tasks as a graphical user interface with similar functionality would be hard to navigate and harder to use.

Second, a command driven interface allows scripting. In any serious data analysis task, reproducibility is a key concern. A command based analysis environment makes replication of results easier, as it is far easier to record every detail of the configuration used in scripts. Thus, if you are worried about R after seeing R console; it is time to lay your worries to rest and learn to love the R console.

Expressions

R console is quite flexible. You can use it for a number of purposes. First of all, try some simple math, and R will evaluate them and print the answers.

```
> 6 + 3

> 6 – 3

> 6 * 3

> 6 / 2

> log(6) + exp(2)
```

Type the string “Business Analytics” on the console

```
> “Business Analytics”
```

Some expressions return a "logical values" (either TRUE or FALSE)

```
> 6 < 3          # Is 6 is smaller than 3?

> 6 + 3 == 10    # Is 6 + 3 equal to 10?
```

 # This is a comment, any line starting with a # sign won't be evaluated by R.

Note that you need a double-equals sign (==) to check whether two values are equal - a single-equal sign (=) won't work

As you can see, R can serve as a calculator. You can also use R for handling vector and matrix operations, which will discuss later

At his point you may be wondering about the [1] pretended to all the results. This is the index of the first item displayed in a row of results.

Let's ask R to return numbers from 1 to 30.

```
> 1:30
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30
```

As you can see, in the second row the index shifts to [26] indicating the 26th number in the series

Variables

If we want to store anything in the environment (memory) to use it later, we can use the assignment operator ' <–"

```
> A <– 6
```

Notice there is no ouput. R merely saved 6 into memory named as 'A'. Simply, A is a shortcut (variable) for 5. We can verify if 6 is assigned to 'A'

```
> A
[1] 6
```

Let's so some math with 'A'

```
> A + 3
> A * 2
> log(A)
```

Notice that R is case sensitive!

> a

```
Error: object 'a' not found
```

We can also assign strings and logical values to variables

> B <− "Business Analytics"

> C <− "FALSE"

Vectors

A vector is simply a list (array) of values. A vector's values can be numbers, strings, logical values, or any other type. Try creating a vector of numbers.

> c(1,2,3)

```
[1] 1 2 3
```

The c function creates a new vector by combining a list of values.

Now try creating a vector with strings:

> c ('a', 'b', 'c')

```
[1] "a" "b" "c"
```

If you need a vector with a sequence of numbers you can create it with start:end notation. Let's make a vector with values from 1 through 5 (you may also use the 'seq' function to make sequences, seq(1, 5))

> 1:5

```
[1] 1 2 3 4 5
```

We can also store a vector (i.e., an array of numbers) in a variable. Here we are saving numbers 1 to 5 in D.

> D <− 1:5

You can carry out arithmetic with scalars (i.e., single number) and other vectors.

> D

```
[1] 1 2 3 4 5
```

```
> D+A
[1] 7 8 9 10 11
```

```
> D*A
[1] 6 12 18 24 30
```

```
> E <- D*A      # Let us save D*A into another variable E
> E
[1] 6 12 18 24 30
```

Nice thing about vectors is you can access variables in a vector through indexes. We know E has numbers from 6 to 30. If we want to access the 3rd element of the vector we can do this as follows:

```
> E[3]
[1] 18
```

Here the number between the square braket "[]" is the index. It tells R to extract only the third element. **Indexing is a key point which we will use quite a bit later.

You can also refer to more than one element at once. Below I refer to elements from the first to third.

```
> E[1:3]
[1] 6 12 18
```

Similarly, we can refer to just 3rd and 5th elements.

```
> E[c(3,5)]
[1] 18 30
```

Now, the third through fifth values are retrieved.

We can assign new values within an existing vector. Try changing the third value to 100:

```
> E[3] <- 100
```

```
> E
[1] 6 12 100 24 30
```

We can also set ranges of values; just provide the values in a vector.

```
> E[4:5] <- c(14, 20)
> E
[1] 6 12 100 14 20
```

Finally, we can expand the current vector by adding new values into the vector.

```
> E[6:8] <- c(11, 40, 35)
> E
[1] 6 12 100 14 20 11 40 35
```

In addition, we can use logic operators to refer to specific elements. This will come in handy later when you are trying to extract all information before a specific date or all transactions of a customer.

We can see if elements of C are smaller than 15.

```
> E < 15
[1] TRUE TRUE FALSE TRUE FALSE TRUE FALSE FALSE
```

We can use this information to extract only elements of E less than 15.

```
> E[E<15]
[1]  6 12 14 11
```

Now, we can assign names to a vector's elements by passing a second vector filled with names to the **names** assignment function, like this:

```
> names(E) <- c("a", "b", "c", "d", "e", "f", "g", "h")
```

Assigning names for a vector can act as useful labels for the data. Below, you can see what our vector looks like now.

> E

```
 a b  c  d  e  f  g  h
6 12 100 14 20 11 40 35
```

You can also use the names to access the vector's values. Try getting the value for "c" :

> E [“c”]

```
  c
100
```

Finally, set the currenct value for the label “c” to a different value using the name rather than the position.

> E[“c”] <− 5

> E

```
   a b c  d e  f  g  h
   6 12 5 14 20 11 40 35
```