Seminar 3

Matrix, Data Frame, and Functions

Matrix and Data Frame

In the last class, we only worked with vectors, which are simple lists of values. What if we need data in rows and columns? For example, a customer transaction table includes multiple arrays (e.g., ID, Names, Item, and Quantity).

ID	Name	Item	Quantity
C001	Gun	Pencil	3
C002	Tom	Paper	6
C003	Bob	Eraser	2
…	…	…	…

A "matrix" and a "data frame" enable us to store multiple arrays/columns/variables in a single storage. Today, we will learn the basic of working with matrices and data frames, from creating them to accessing them.

Matrix

A matrix is just a fancy term of a two-dimensional array. All columns in a matrix must have the same mode (numeric, character, etc.). Let's make a matrix, 2 rows by 3 columns, with all its fields set to 1.

> matrix (1, 2, 3) # matrix(initial value, #rows, #columns)

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

We can also use a vector (a list of values) to initialize a matrix's value. To fill a 2 X 3 matrix, we will need a vector containing 6 items. We will make it now:

> a <- seq(1,6)

Now call matrix with the vector we have just created, the number of rows and the number of columns:

> matrix (a, 2, 3)

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

The vector's values are copied into the new matrix, one by one. You can also re-shape the vector itself into a matrix. Create 12-item vector:

```
> b <- 1:12
```

The dim function sets dimensions for a matrix. It accepts a vector with the number of rows and the number of columns to assign.

Assign new dimensions to a vector, b, by passing a vector specifying 3 rows and 4 columns (i.e., c(3,4)):

```
> dim(b) <- c(3, 4)
```

If you print 'b' now, you will see that the values have shifted to form a 3 X 4 matrix:

```
> print(b)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

The vector is no longer one-dimensional. It has been converted into a matrix with rows and columns.

Getting values from matrices is not that different from vectors; you just have to provide two indices instead of one.

Try getting the value from the second row in the fourth column of 'b'

```
> b[2, 4]        # [row index, column index]
```

```
[1] 11
```

What about getting the value from the third row in the first column of 'b'?

```
> b[3, 1]
```

```
[1] 3
```

As with vectors, to set a single value, just assign to it, set the value to 7:

```
> b[3,1] <- 7
```

```
> b
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    7    6    9   12
```

You can get an entire row of the matrix by omitting the column index (but keep the comma). Try retrieving the third row of 'b':

> b[3,]

```
[1] 20 6 9 12
```

Try retrieving the first column of 'b':

> b[,1]

```
[1] 1 2 20
```

We can also read multiple rows or columns by providing a vector (or sequence) with their indices. Try retrieving columns 2 through 4:

> b[, 2:4]

```
     [,1] [,2] [,3]
[1,]    4    7   10
[2,]    5    8   11
[3,]    6    9   12
```

Data Frame

In R, a data frame is the typical storage of data. A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

Let's create the four variables containing different types of values (No., Names, Item, and Quantity) in the customer transaction table on the first page:

> id <− c('C001', 'C002', 'C003')

> name <− c('Gun', 'Tom', 'Bob')

> item <− c('Pencil', 'Paper', 'Eraser')

> qty <− c(3, 6, 2)

Now we can put the four vectors together into a single storage by using the **data.frame** function:

> sales <− data.frame(ID=id, Name=name, Item=item, Quantity=qty)

> sales

```
    ID Name   Item Quanity
1 C001  Gun Pencil       3
2 C002  Tom  Paper       6
3 C003  Bob Eraser       2
```

As you can see, the data frame, sales, has four variables (ID, Names, Item, and Quantity) in columns and three observations of each in rows. This is a two-dimensional data structure. It means we can refer to rows and columns.

Like a matrix, we can retrieve a specific value from a data frame by providing two indices of row and column.

Try getting the value from the second row in the third column of 'sales':

> sales[2,3]

```
[1] Paper
```

We can also refer to multiple rows and columns at the same time.

> sales[1:2, 3:4]

```
    Item Quanity
1 Pencil       3
2  Paper       6
```

We can use column names to refer to columns. Try getting the value in the "Name" column.

> sales[, "Name"]

```
[1] Gun Tom Bob
```

Alternatively, we can use the $ sign to call a specific column (variable):

> sales$Name

```
[1] Gun Tom Bob
```

Now think about a situation, where the first column is the customer ID and you want to all transactions of a specific customer. We can use the logic operators we learned about earlier to obtain just that. Let us extract the observations relating to customer, C002:

> sales[sales$ID == 'C002',]

```
    ID Name  Item Quanity
2 C002  Tom Paper       6
```

Functions

So now we know about assignment operators (<- and =), logical operates (==, !=, <, >, …), variables, vectors, matrices, and data frames. Learning about functions will conclude our introduction to R console, and enable us to explore more exciting area of analytics with R.

A basic function is just a word followed by parentheses. Between the brackets are the options (parameters) of the function.

Some functions (the most useful ones) take parameters. Here we are asking R to calculate the mean of numbers in vector D. D serves as a parameter to tell the mean function over which numbers to calculate the mean.

> D <- 1:10

> mean(D)

```
[1] 5.5
```

You can directly use the output of one function with another. Here we wan to calculate the mean by first summing the D vector then dividing the sum by number of elements.

> sum(D) / length(D)

```
[1] 5.5
```

Functions can take more than one parameter. Here we are asking R to calculate mean, after trimming the extreme values (10% of the data).

> mean(D, trim = 0.1)

```
[1] 5.5
```

If you want to learn what a function does, or what parameters are available, use the help function.

> help(mean)

If you want to see the examples of a function, use the example function.

> example(mean)

```
mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

Finally, we can also create our own function by saving them in a variable. Below I declare a 'hello()' function that takes a string, or number and prints out a hello message.

```
# Declare a Function
> hello <− function(x) {print(paste("hello", x))}

#Use a Function
> hello('BC2406')

[1] "Hello BC2406"
```