**MODULE#5 ASSIGNMENT**

The focus of this assignment is to learn and work with matrices in RStudio.

**The matrices ‘A’ from 1 to 100 with 10 rows and ‘B’ from 1 to 1000 with 10 rows is created using the following syntax** ‘y <- matrix(c(1,2,3,4),nrow=2,ncol=2)’

> A = matrix(1:100, nrow=10)

> B = matrix(1:1000, nrow=10)

**In order to get the transpose of a matrix, transpose function ‘**t()**’is implemented.** Transpose function will turn rows into columns and columns into rows which will be useful when Term Document Matrix is needed from the Document Term Matrix. In addition, it can also be useful in more advanced matrix algebra.

*#Transpose A and B*

> t(A)

> t(B)

**Two vectors ‘a and b’ are created using the colon operator to generate integers sequence.**

*#Create two vectors (a and b)*

> a = c(1:5)

> b = c(1:10)

**The created two matrices ‘A and B’ are multiplied with vectors ‘a and b’ respectively.**

*#Multiply matrices by vectors*

> X = a\*A

> Y = b\*B

**Vectors can be re-assigned to equal the number of rows of the column for the corresponding matrix.**

*#re-assign the vectors a and b*

> a=c(1:10)

> b=c(1:100)

**If only multiplication operator ‘\*’ is used, R will only multiply the corresponding elements of the two matrices if they have same dimensions.**So, in order to multiply matrices of different dimensions we can use **‘%\*%’** provided the **number of *columns* in the first matrix must match the number of *rows* in the second matrix in order for matrix multiplication to be possible.** If the dimensions do not match, an error will be produced.

*#Multiply the matrix by a matrix*

> A%\*%B

> A %\*% a

> B %\*% b

**In order to inverse of a matrix, following syntax ‘solve()’ can be used, provided the matrix is singular which is not invertible and gives us an error message.**

*#Inverse a matrix*

> S=matrix(1:4, nrow=2)

> S

[,1] [,2]

[1,] 1 3

[2,] 2 4

> inv\_S=solve(S)

> inv\_S

[,1] [,2]

[1,] -2 1.5

[2,] 1 -0.5

**When inverse of a matrix multiplied with the original matrix, it results in an identity matrix.**

> inv\_S%\*%S

[,1] [,2]

[1,] 1 0

[2,] 0 1

**This error in finding inverse of A results as its determinant is 0 which means there is no inverse for this dataset. The error in finding inverse of B resulted due to it being rectangular matrix and not square matrix.**

> solve(A)

Error in solve.default(A) :

Lapack routine dgesv: system is exactly singular: U[6,6] = 0

> solve(B)

Error in solve.default(B) : 'a' (10 x 100) must be square

**The determinant of a matrix can be obtained using determinant function ‘det()’ provided it is a square matrix, otherwise it results in error message.**

*#check det()*

> det(S)

[1] -2

> det(A)

[1] 0

> det(B)

Error in determinant.matrix(x, logarithm = TRUE, ...) :

'x' must be a square matrix

In this way, we can implement different codes and work with huge matrices and vectors in order to multiply them, find inverse and determinants for matrices.

