**Assignment R Programming**

**Write the following functions:**

1. **makeCacheMatrix: This function creates a special "matrix" object that can cache its inverse.**

*#makeCacheMatrix creates a special "vector", which is really a list containing a function to*

*# create a matrix "makeCacheMatrix"*

*# the matrix is invertible*

*# set the value of the matrix using x <<- y which is invertible*

*# get the value of the x matrix*

*# set the inverse function*

*# get the value of inverse*

*# create a list*

makeCacheMatrix <- function(x = matrix()) {

inv <- NULL

set <- function(y){

x <<- y

inv <<- NULL

}

get <- function() (x)

setInverse <- function(inverse) (inv <<- inverse)

getInverse <- function() (inv)

list(set = set, get = get, setInverse = setInverse, getInverse = getInverse)

}

1. **cacheSolve: This function computes the inverse of the special "matrix" returned by makeCacheMatrix above. If the inverse has already been calculated (and the matrix has not changed), then the cachesolve should retrieve the inverse from the cache.**

*##cacheSolve is a function which computes the inverse of the special "matrix" returned by makeCacheMatrix above. If the inverse has already been calculated (and the matrix has not changed), then the cachesolve should retrieve the inverse from the cache*

*## Return a matrix that is the inverse of 'x'*

*## To see if the inverse has already been calculated, it will get it from the cache and skip the computation*

*## If inverse has already been calculated it will display message "getting cached data"*

*## return the inverse from the cache*

cacheSolve <- function(x, ...) {

inv <- x$getInverse()

if(!is.null(inv)){

message("getting cached data")

return(inv)

}

*## if the inverse is not in the cache, compute the inverse*

mat <- x$get()

inv <- solve(mat, ...)

x$setInverse(inv)

inv

}