```r
####--------------------------------------####
##---- ECON690 COMPUTATIONAL ECONOMICS ----##

## by Bahar Zafer

### 1 Class Assignment: Introduction to R ###
#---------------------------------------------#

## 1.1 INTRODUCTION
#---------------------------------------------#
install.packages("BB")
library(BB)

##------ Exercise 1: Introduction ------##
#----------------------------------------#

# 1. Create a directory for this class and store your script "a0.R"
dir.create("C:\\Users\\bahar\\Desktop\\Econ690_Computational_Economics\\W_dir_EC690")

# 2. Install the packages, Hmisc, gdata,boot,xtable,MASS,moments,snow,mvtnorm
packages = c("Hmisc", "gdata", "boot", "xtable", "MASS", "moments", "snow", "mvtnorm")
install.packages(packages)

# 3. Set your working directory
setwd("C:\\Users\\bahar\\Desktop\\Econ690_Computational_Economics\\W_dir_EC690")

# 4. List the content of your directory and the content of your environment.
dir()
ls()

# 5. Check whether 678 is a multiple of 9.
678%%9  # 678 is not a multiple of 9.

# 6. Save your environment
save.image("C:/Users/bahar/Desktop/Econ690_Computational_Economics/W_dir_EC690/ENV0.RData")

# 7. Find help on the function mean, cut
help(mean)
help(cut)

# 8. Find an operation that returns NaN (Not A Number)
0*Inf


##------ Exercise 2: Object Manipulation 2 ------##
#-------------------------------------------------#

# 1. Print Titanic, and write the code to answer these questions (one function (sum), one operation)
print(Titanic)

      # (a) Total population
      sum(Titanic)

      # (b) Total adults
      sum(Titanic[, , "Adult", ])

      # (c) Total crew
      sum(Titanic["Crew", , , ])

      # (d) 3rd class children
      sum(Titanic["3rd", ,"Child", ])

      # (e) 2nd class adult female
      sum(Titanic["2nd", "Female", "Adult", ])

      # (f) 1st class children male
      sum(Titanic["1st", "Male", "Child", ])
```

```r

      # (g) Female Crew survivor
      sum(Titanic["Crew", "Female", ,"Yes" ])

      # (h) 1st class adult male survivor
      sum(Titanic["1st", "Male", "Adult", "Yes"])

# 2. Using the function prop.table, find

      # (a) The proportion of survivors among first class, male, adult
      prop.table(Titanic["1st", "Male", "Adult", ])

      # (b) The proportion of survivors among first class, female, adult
      prop.table(Titanic["1st", "Female", "Adult", ])

      # (c) The proportion of survivors among first class, male, children
      prop.table(Titanic["1st", "Male", "Child", ])

      # (d) The proportion of survivors among third class, female, adult
      prop.table(Titanic["3rd", "Female", "Adult", ])


##------ Exercise 3: VECTORS - INTRODUCTION ------##
#-----------------------------------------------------#

# 1. Use three different ways, to create the vectors

      # a) a = 1,2,...,50
      a = 1:50
      a = c(1:20, 21:50)
      a = seq(1, 50, by=1)

      # b) b = 50,49,...,1
      b = rev(a)
      b = c(50, seq(49, 1))
      b = list(50:1)

# 2. Create the vectors

      # a) a = 10,19,7,10,19,7,...,10,19,7  with 15 occurrences of 10,19,7
      a = rep(c(10,19,7), 15)

      # b) b = 1,2,5,6,...,1,2,5,6   with 8 occurrences of 1,2,5,6
      b = rep(c(1,2,5,6), 8)

# 3. Create a vector of the values of log(x)sin(x) at x = 3.1, 3.2, ... , 6
vecA033 = vector(mode = "numeric")
a = 1
for (i in seq(3.1, 6, by = 0.1)) {
   vecA033[a] = log(i)*sin(i)
   a = a + 1
}
vecA033  # is a vector of the values of log(x)sin(x) at x = 3.1, 3.2, ... , 6.

# 4. Using the function sample, draw 90 values between (0,100) and calculate the mean.
# Re-do the same operation allowing for replacement.
x = sample(0:100, 90, replace = F)
y = sample(0:100, 90, replace = T)
f = function(x) {log(x)*sin(x)}
mean(f(x))
mean(f(y))

# 5. Calculate

# a)
c = 0
funA = for (a in 1:15){
      for (b in 1:20) {
   c = c + exp(a^(1/2))*log(a^5) / (5+cos(a)*sin(b))
```

```r
        }
}
c # the answer is 14091.51.

# b)
c = 0
funB = for (a in 1:15){
   for (b in 1:a) {
      c = c + exp(a^(1/2))*log(a^5) / (5+exp(a*b)*cos(a)*sin(b))
   }
}
c  # the answer is  3.462342

# 6.  Create a vector of the values of exp(x)*cos(x) at x = 3, 3.1, ...6.
vecA036 = vector(mode = "numeric")
a = 1
for (i in seq(3, 6, by = 0.1)) {
   vecA036[a] = exp(i)*cos(i)
   a = a + 1
}


##------ Exercise 4 Vectors - Advanced ------##
#--------------------------------------------#

# 1. Create two vectors xVec and yVec by sampling 1000 values between 0 and 999.
xVec = sample(0:999, 1000, replace = T)
yVec = sample(0:999, 1000, replace = T)

# 2. Suppose xVec = (x1, , ,xn) and yVec = (y1,,,yn)

      # (a) Create the vector (y2 - x1, ..., yn - xn-1) denoted by zVec.
      yyVec = yVec[-1]
      xxVec = xVec[-1000]
      zVec = yyVec - xxVec

      # (b)
      yyVec2 = yVec[-1000]
      xxVec2 = xVec[-1]
      zVec = sin(yyVec2)/cos(xxVec2)

      # (c) Create a vector subX which consists of the values of X which are >= 200.
      subX = xVec[xVec>=200]

      # (d) What are the index positions in yVec of the values which are >= 600.
      for (i in 1:length(yVec)) {
         if (yVec[i] >= 600) {print(i)}
      }


##------ Exercise 5 Matrix ------##
#--------------------------------#

# 1.
A = matrix(c(1,5,-2,1,2,-1,3,6,-3), ncol = 3)

   # a) Check that A^3=0 (matrix 0).
   A^3==0

   # b) Bind a fourth column as the sum of the first and third column.
   A = cbind(A, A[,1]+A[,3])

   # c) Replace the third row by the sum of the first and second row.
   A[3, ] = A[1, ] + A[2, ]

   # d) Calculate the average by row and column.
   colMeans(A)    # Values of column means are 4, 2, 6, and 10.
   rowMeans(A)    # Values of row means are 2.25, 6, and 8.25
```

```r
# 2.
B = matrix(c(2, 1, 3, 1, 1, 1, 1, 3, 2), nrow = 3, byrow = TRUE)
S = matrix(c(10, 6, 13))

# 3.
solve(B, S)  # x = 2, y = 3 and z = 1.


##------ Exercise 6 Functions ------##
#------------------------------------#

# 1. Write a function fun1 which takes two arguments (a,n) where (a) is a scalar
# and n is a positive integer, and returns a + a^2/2 + ... + a^n/n.

fun1 = function(a, n){
   n1 <- 1:n
   sum(a^n1/n1)
}

#2. Evaluate the following function at -3, 0 and 3.
fun2 = function(x){if (x<0) {x^2+2*x+abs(x)}
   else if (x>=0 & x<2) {x^2+3+log10(1+x)}
   else {x^2+4*x-14}}
fun2(-3)     # equals 6.
fun2(0)      # equals 3.
fun2(3)      # equals 7.


###------ Exercise 7 Indexes ------###
#------------------------------------#

# 1. Sample 36 values between 1 and 20 and name it v1.
v1 = sample(1:20, 36, replace = T)

#2. Use two different ways, to create the subvector of elements that are not in the first row.


#3. Create a logical element (TRUE or FALSE), v2, which is true if v1 > 5.
# Can you convert this logical element into a dummy 1 (TRUE) and 0 (FALSE)?
v2 = if (v1>5) {T} else {F}
v2 = as.numeric(v2)

# 4. Create a matrix m1 [6 x 6] which is filled by row using the vector v1.
m1 = matrix(v1, nrow = 6, byrow = T)

#5.
x = c(rnorm(10), NA, paste("d", 1:16), NA, log(rnorm(10)))

#6. Test for the position of missing values, and non-finite values.
# Return a subvector free of missing and non-finite values.
x = as.numeric(x)
is.na(x)
is.finite(x)
subx = vector()
for (i in 1:length(x)) {
   if (as.numeric(is.na(x[i])==F) + as.numeric(is.finite(x[i])==T) == 2)  {subx[i] = x[i]}
   else {next}
   subx = subx[!is.na(subx)]
}
subx   # subx is a subvextor of x free of missing and non-finite values.


###------ Exercise 8 Data Manipulation ------###
#----------------------------------------------#

#1. Load the library AER, and the dataset (data("GSOEP9402")) to be named dat.
library(AER)
```

```r
data("GSOEP9402")
dat = GSOEP9402

#2. What type of object is it? Find the number of rows and column?
# Can you provide the names of the variables?
typeof(dat)       # dat is a list.
nrow(dat)         # there are 675 rows in the dataset.
ncol(dat)         # there are 12 columns in the dataset.
var_names = colnames(dat)    # Names of the variables are school, birthyear, gender, kids, parity,
                              # income, size, state, marital, meducation, memployment, year.

# 3. Evaluate and plot the average annual income by year.
avg_inc_year = tapply(dat$income, dat$year, mean)
avg_inc_year = as.matrix(avg_inc_year)
years = row.names(avg_inc_year)
plot(years, avg_inc_year)

# 4. Create an array that illustrates simultaneously the income differences (mean)
# by gender, school and memployment.
gend = tapply(dat$income, dat$gender, mean)
mem = tapply(dat$income, dat$memployment, mean)
s = tapply(dat$income, dat$school, mean)
arr = array(c(gend, mem, s))


###------ Exercise 9 First Regerssion ------###
#--------------------------------------------#

# 1.
library(AER)
data("CASchools")
data = CASchools

# 2.
reg1 = lm(data$read ~ data$district + data$school +
            data$county + data$grades + data$students +
             data$teachers + data$calworks + data$lunch +
             data$computer + data$expenditure + data$income + data$english)

# 3.



###------ Exercise 10 Advanced Indexing ------###
#----------------------------------------------#

# 1. Create a vector lu of 200 draws from a pareto distribution (1,1). How many values are higher than 10.
# Replace these values by draws from a logistic distribution (6.5,0.5).
library(actuar)
lu = rpareto(200, 1, 1)
sum(lu>10)  # 22 values are higher than 10.
logist = rlogis(22, 6.5, 0.5)
a=1
for(i in 1:200) {
   if(lu[i]>10) {
      lu[i] = logist[a]
      a = a + 1}
}

# 2.
de = rnorm(200, 1, 2)
de = log(de)
sum(is.na(de))    # there are 58 missing values.
install.packages("truncnorm")
library(truncnorm)
t = rtruncnorm(58, 0, 1)
```

```
a = 1
for (i in 1:200) {
   if(is.na(de[i]) == 1) { de[i] = t[a]
   a = a + 1}
}
sum(de<0)          # there are 38 negative values.
tt = rtruncnorm(38, 0, 1)
a = 1
for (i in 1:200) {
   if(de[i]<0) {de[i] = tt[a]
   a = a + 1}
}
sum(de<0)  # sum equals 0. All negative values are replaced.

# 3.
orig = runif(200, 0, 1)
dest = runif(200, 0, 1)

# 4.
histt = matrix(runif(200*200, 0, 1), ncol = 200, nrow = 200)
dist = matrix(runif(200*200, 0, 1), ncol = 200, nrow = 200)

# 5.
# 6.
# 7.
# 8.
# 9.


###------ Exercise 11 Testing and Idexing ------###
#-----------------------------------------------------#

# 1. Test if c(1,2,3) is an array? a vector? a matrix?
is.array(c(1,2,3))
is.vector(c(1,2,3))
is.matrix(c(1,2,3))
# c(1, 2, 3) is a vector.

# 2. x0 = rnorm(1000);
# Using the function table() count the number of occurrences of
# x0 > 0, x0 > 1, x0 > 2, x0 > 0.5, x0 < 1 and x0 > -1
x0 = rnorm(1000)
table(x0 > 0)     # 509 False, 491 True
table(x0 > 1)     # 832 False, 168 True
table(x0 > 2)     # 976 False, 24 True
table(x0 > 0.5)   # 684 False, 316 True
table(x0 < 1)     # 168 False, 832 True
table(x0 > -1)    # 170 False, 830 True

# 3.
library(Hmisc)
x1 = cut2(runif(100, 0, 1), g=10)
levels(x1) = paste("q", 1:10, sep = "")

# 4. Test whether or not x1 is a factor.
is.factor(x1)     # x1 is a factor.

# 5. Verify that "q1" has 10 occurences.
a = 0
for (i in 1:length(x1)) {
   if(x1[i]=="q1") a = a + 1
}
print(a)

# 6. Convert x1 into a numeric variable. What happens to the levels?
x1 = as.numeric(x1)
levels(x1)  # levels(x1) returns NULL.

# 7.
```

```r
rand = rnorm(1000)

#8. Using the function which() find the indexes of positive values.
positive_rand_i = which(rand>0)
positive_rand_i

#9. Create the object w of positive values of x using:
   # a) Which
   w = rand[which(rand>0)]
   # b) Subset
   w = subset(rand, rand>0)
   # c) By indexing directly the values that respect a condition.
   w = vector()
   a = 1
   for (i in 1:length(rand)) {
      if(rand[i] > 0) {
         w[a] = rand[i]
         a = a + 1
      }
   }

###------ Exercise 12 Programming ------###
#-----------------------------------------#

# Write a program that asks the user to type an integer N and compute u(N) defined with:
# u(0)=1, u(1)=1, u(n+1)=u(n)+u(n-1)




# 1. Evaluate 1^2 + 2^2 + ... + 400^2.
s = 0
f = function(x) {
   for(i in 1:x) {
      s = s + i^2
   }
   print(s)
}
f(400)   # f(400) = 1^2 + 2^2 + ... + 400^2 = 21413400

# 2. Evaluate 1 x 2 + 2 x 3 + 3 x 4 + ... + 249 x 250.
s = 0
g = function(x) {
   for (i in 1:(x-1)){
    s = s + i*(1+i)
   }
   print(s)
}
g(250)   # g(250) =  1 x 2 + 2 x 3 + 3 x 4 + ... + 249 x 250 = 5208250.

# 3. Create a function "crra" with two arguments (c, theta) that returns c^(1-theta)/(1-theta).
# Add. if condition such that the utility is guven by the log when theta is in [0.97, 1,03].
crra = function(c, theta) {if(0.97 <= theta & theta <= 1.03) {log(c)} else {c^(1-theta)/(1-theta)}}

# 4. Create a function "fact" that returns the factorial of a number.
fact = function(x) {factorial(x)}

#---------------------------------------------#
###------ Exercise 13 Apply Functions ------###
#---------------------------------------------#

# 1. Using this object
m = matrix(c(rnorm(20,0,10), rnorm(20,-1,10)), nrow = 20, ncol = 2)
# Calculate the mean, median, min, max and standard deviation by row and column.
rowMeans(m)    # calculates means for each row
colMeans(m)    # calculates means for each column
```

```r
for (i in 1:nrow(m)) {print(c(i, "median", median(m[i, ])))} # calculates median for each row.
for (i in 1:nrow(m)) {print(c(i, "max", max(m[i, ])))}  # returns maximum value at each row.
for (i in 1:nrow(m)) {print(c(i, "min", min(m[i, ])))}  # returns minimum value at each row.
for (i in 1:nrow(m)) {print(c(i, "sd", sd(m[i, ])))} # calculates standard deviation for each row.
for (i in 1:ncol(m)) {print(c(i, "median", median(m[,i])))} # calculates median for each column.
for (i in 1:ncol(m)) {print(c(i, "max", max(m[,i])))}  # returns maximum value at each column.
for (i in 1:ncol(m)) {print(c(i, "min", min(m[,i])))}  # returns minimum value at each column.
for (i in 1:ncol(m)) {print(c(i, "sd", sd(m[,i])))} # calculates standard deviation for each column.

# 2. Using the dataset iris in the package "datasets", calculate the average Sepal.Length by Species.
# Evaluate the sum log of Sepal.Width by Species.
library(datasets)
data(iris)

species = levels(iris$Species)

s1 = 0
s2 = 0
s3 = 0
d1 = 0
d2 = 0
d3 = 0
w1 = 0
w2 = 0
w3 = 0
if(iris$Species[i] == species[1]) {
      s1 = s1 + iris$Sepal.Length[i]
      d1 = d1 + 1
      m1 = s1/d1
      w1 = w1 + log(iris$Sepal.Width[i])
} else if (iris$Species[i] == species[2]) {
      s2 = s2 + iris$Sepal.Length[i]
      d2 = d2 + 1
      m2 = s2/d2
      w2 = w2 + log(iris$Sepal.Width[i])
} else {
      s3 = s3 + iris$Sepal.Length[i]
      d3 = d3 + 1
      m3 = s3/d3
      w3 = w3 + log(iris$Sepal.Width[i])
}
MeanSepalLength = c(m1, m2, m3)
SumLogSepalWidth = c(w1, w2, w3)
M = rbind(MeanSepalLength, SumLogSepalWidth)
colnames(M) = species
print(M)

# 3.
y1 = NULL; for (i in 1:100) y1[i]=exp(i)
y2 = exp(1:100)
y3 = sapply(1:100,exp)

   # a. Check the outcomes.
   y1
   y2
   y3    # All three give the same vector.

   # b. Using proc.time() or system.time(), compare the execution time of these three equivalents commands.
   system.time(y1)
```

```r
   system.time(y2)
   system.time(y3)      # All values are 0.


#------------------------------------------------------#
###------ Exercise 14 Simulating and Computing ------###
#------------------------------------------------------#

# 1. Simulate a vector x of 10,000 draws from a normal distribution.
# Use the function summary to provide basic characteristics of x.
x = rnorm(10000)
summary(x)

# 2. Create a function dsummary that returns, the minimum, the 1st decile, the 1st quartile,
# the median, the mean, the standard deviation, the 3rd quartile, the 9th decile, and the maximum.
dsummary = function(x) {
   a = summary(x)
   a = as.matrix(a)
   q = quantile(x, probs = 0:10/10)
   q = as.matrix(q)
   M = rbind(a, q[2,], q[10,])
   row.names(M) = c("Min", "1st Qu", "Median", "Mean", "3rd Qu", "Max", "1st dec", "9th dec")
   return(M)
}

# 3. Suppose X ~ N(2, 0.25). Evaluate f(0.5), F(2.5), F^(-1)(0.95)
dnorm(0.5, mean=2, sd=sqrt(0.25)) # evaluates the density function at 0.5. f(0.5) = 0.008863697
pnorm(2.5, mean=2, sd=sqrt(0.25)) # evaluates the cumulative probability. F(2.5) = 0.8413447
qnorm(0.95, mean=2, sd=sqrt(0.25)) # F^(-1)(0.95) = 2.822427

# 4. Repeat if X has t-distribution with 5 degrees of freedom.
dt(0.5, df=5)      # f(0.5) = 0.3279185
pt(2.5, df=5)      # F(2.5) = 0.972755
qt(0.95, df=5)     # F(-1)(0.95) = 2.015048

# 5. Suppose X ~ P(3, 1), where P is the pareto distribution. Evaluate f(0.5), F(2.5), F^(-1)(0.95)
install.packages("actuar")
library(actuar)
dpareto(0.5, 3, 1)      # f(0.5) = 0.5925926
ppareto(2.5, 4, 1)      # F(2.5) = 0.9766764
qpareto(0.95, 3, 1)     # F^(-1) = 1.714418

#------------------------------------#
###------ Exercise 15 Moments ------###
#------------------------------------#

# Consider a vector
V = rnorm(100, -2, 5)

# 1. Evaluate n as the length of V.
n = length(V)    # n = 100

# 2. Compute the mean
m = mean(V)   # m = -2.378689

# 3. Compute the variance
ss = 0
for (i in 1:n) {
   ss = ss + (V[i] - m)^2
   variance = ss/(n-1)
}
variance  # s^2 = 22.90643
```

```
# 4. Compute the skewness
sk = 0
for (i in 1:n) {
   sk = sk + (V[i] - m)^3
   skewness = sk/(n*variance^(3/2))
}
skewness  # Skewness = 0.1808976

# 5. Compute the kurtosis
k = 0
for (i in 1:n) {
   k = k + (V[i] - m)^4
   kurtosis = k/(n*variance^2) - 3
}
kurtosis    # kurtosis = 0.1909011


#---------------------------------#
###------ Exercise 16 OLS ------###
#---------------------------------#

# 1. Create a matrix X of dimension (1000,10).
# Fill it with draws from a beta distribution with shape1 parameter 2, and shape 2 parameter 1.
# Make sure that there is no negative.
install.packages("matlib")
library(matlib)
X = matrix(rbeta(10000, 2, 1), ncol = 10, nrow = 1000)
sum(X > 0) # equals 10000. Then, all elements in X are greater than 0.

# 2.
sigma_sq = 0.5
beta = vector(length = 10)
beta = rgamma(10, 2, 1)

# 3.
e = rnorm(1000)

# 4.
Y = X%*%beta + sigma_sq^(1/2)*e

# 5.
beta_hat = inv(t(X)%*%X)%*%t(X)%*%Y

# 6.
e_hat = X%*%beta_hat - Y
hist(e_hat, col = "grey")
plot(density(e_hat))

# 7.
sigma_sq_est = (t(e_hat)%*%e_hat)/(1000-10-1)
Var_beta_hat = sigma_sq*inv(t(X)%*%X)

# 8.
OLS = lm(Y ~ X)
coefficients(OLS)

# 9.
conf_int_beta = confint(OLS, level = 0.95)

# 10.
X = matrix(rbeta(10000, 2, 1), ncol = 10, nrow = 1000)
sigma_sq = 0.01
beta = vector(length = 10)
beta = rgamma(10, 2, 1)
e = rnorm(1000)
Y = X%*%beta + sigma_sq^(1/2)*e
OLS = lm(Y ~ X)
conf_int_beta = confint(OLS, level = 0.95)
```

```
coefficients(OLS)
# Confident intervals for beta are smaller.
```