Nruhari Viswanath S

200101101

DSA Individual Assignment

Submitted to :

Prof. (Dr.) Sridhar Vaithyanathan,

Associate Professor (Analytics).

IMT - Hyderabad

Date of submission: 18.12.2020

Table of Contents:

R codes

Nruhari Viswanath

12/18/2020

#1. R introduction

```
x = 10
x

## [1] 10

a = 10
a

## [1] 10

class(a)

## [1] "numeric"

a = "Hello World"

class(a)

## [1] "character"

a = TRUE
class(a)

## [1] "logical"

a = FALSE
class (a)

## [1] "logical"

# Logical TRUE and logical FALSE are equivalent to 1 and 0 respectively.
a= FALSE + TRUE
#basic calculations
a

## [1] 1

factorial(x)

## [1] 3628800

a^x
```

```
## [1] 1
```

```r
x*a^x
```

```
## [1] 10
```

#2. Data types

```r
a= 5
class(a)
```

```
## [1] "numeric"
```

```r
# To check is a is of numeric type (the below command returns TRUE or FALSE)
is.numeric(a)
```

```
## [1] TRUE
```

```r
x = "R is great"
# To check if x is of character type
is.character(x)
```

```
## [1] TRUE
```

```r
# In R, date is also a data type
#as.Date command converts character or numeric type to Date type
class("1996-08-31")
```

```
## [1] "character"
```

```r
date1 = as.Date("1996-08-31")
date1
```

```
## [1] "1996-08-31"
```

```r
class (date1)
```

```
## [1] "Date"
```

```r
as.numeric(date1)
```

```
## [1] 9739
```

```r
#To convert numeric type to character type
a= 5
class(a)
```

```
## [1] "numeric"
```

```r
a
```

```
## [1] 5
```

```r
as.character(a)
```

```
## [1] "5"

#POSIXct data type stores both Date and Time
#In R, the reference date is 01 Jan 1970.
date2 = as.POSIXct("1996-08-31 07:31")
date2

## [1] "1996-08-31 07:31:00 IST"

class(date2)

## [1] "POSIXct" "POSIXt"

#Below command gives the number of seconds from reference date and time to
date2
as.numeric(date2)

## [1] 841456860
```

#3. Vectors

```
# A vector is collection of elements of same data type.
# ':' operator can be used to create a vector
# 'c' stands for combine.
a <- c(1:7,99,76,44)
b <- 7:15
a

##  [1]  1  2  3  4  5  6  7 99 76 44

b

## [1]  7  8  9 10 11 12 13 14 15

#R is vectorized language. So any operation that can be performed on a
particular element of a vector can be performed for the entire vector. R
automatically performs the operation for the entire vector.

(b/2)+5

## [1]  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5

b^2

## [1]  49  64  81 100 121 144 169 196 225

#Length function returns the length of the vector
length(b)

## [1] 9

x= 1:10
y=1:5
```

```r
#To execute x+y, R converts the shorter vector (y) to the same length of the
longer vector(x) by recycling y.
x+y

##  [1]  2  4  6  8 10  7  9 11 13 15

#Comparing 2 vector
any(x<y)

## [1] FALSE

all(x>y)

## [1] FALSE

#Subsetting means accessing individual elements of an object.[] is used to
subscript a vector. The number inside the [] represents the position to be
subsetted.
z= x+y
z[3]

## [1] 6

z[c(3:5,9)]

## [1]  6  8 10 13

#Assigning names to a vector using names function
names(y) <- c("a","b","c","d","e")
y

## a b c d e
## 1 2 3 4 5

#Names can also be used for subsetted
y["a"]

## a
## 1
```

#4. Data structures #4.1 List

```r
#List
#List is a collection of different elements which can be of different data
types.
list1 <- list(a=1:5, b="Nruhari",c= c("R","is","interesting"),
d=matrix(1:6,3))
list1

## $a
## [1] 1 2 3 4 5
##
## $b
```

```
## [1] "Nruhari"
##
## $c
## [1] "R"           "is"          "interesting"
##
## $d
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6

#Subsetting lists
list1[1] #Accessing 1st element of list1

## $a
## [1] 1 2 3 4 5

list1[[1]][4] #Accessing 4th element of 1st element of list1

## [1] 4

list1[[3]][3] #Accessing 3rd element of 3rd element of list1

## [1] "interesting"

names(list1)

## [1] "a" "b" "c" "d"

list2 <- list(1:5, c("Good","Morning"),c("Hello","India"))
list2

## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "Good"    "Morning"
##
## [[3]]
## [1] "Hello" "India"

names(list2) <- c("vector","string1","string2")
list2

## $vector
## [1] 1 2 3 4 5
##
## $string1
## [1] "Good"    "Morning"
##
## $string2
## [1] "Hello" "India"
```

```r
length(list2)

## [1] 3
```

#4.2 Matrix

```r
#Matrix is table of 2D rows and columns containing elements of same data type
b= matrix(1:10,5,2)
b

##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10

A = matrix(1:10,5)# Create a 5x2 matrix
B = matrix(21:30,5)#Create another 5x2 matrix

#Addition of A and B (ELement to element addition)
A+B

##      [,1] [,2]
## [1,]   22   32
## [2,]   24   34
## [3,]   26   36
## [4,]   28   38
## [5,]   30   40

#Matrix Multiplication.
A %*% t(B) #t(B) transposes B so that matrix multiplication is possible
between A and B

##      [,1] [,2] [,3] [,4] [,5]
## [1,]  177  184  191  198  205
## [2,]  224  233  242  251  260
## [3,]  271  282  293  304  315
## [4,]  318  331  344  357  370
## [5,]  365  380  395  410  425
```

#4.3 Data frame

```r
#Data frame is a 2D table of rows and columns which can contain elements of
different data types.
#The difference between matrix and data frame is that in matrix all elements
have to be of same data type.
#So appending row or column names to a matrix would coerce all the data
elements to character data type. #That is why data frame is preferred to
matrix.
c = data.frame(1:5,-1:3,-4:0)
c
```

```
##   X1.5 X.1.3 X.4.0
## 1    1    -1    -4
## 2    2     0    -3
## 3    3     1    -2
## 4    4     2    -1
## 5    5     3     0
```

```r
colnames(c) = c("a","b","c")
c
```

```
##   a  b  c
## 1 1 -1 -4
## 2 2  0 -3
## 3 3  1 -2
## 4 4  2 -1
## 5 5  3  0
```

```r
# Checking the dimensions of the data frame c.
nrow(c)
```

```
## [1] 5
```

```r
ncol(c)
```

```
## [1] 3
```

```r
dim(c) #dimensions of c
```

```
## [1] 5 3
```

```r
names(c)
```

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

```r
names(c)[3] #Name of the 3rd column
```

```
## [1] "c"
```

```r
#printing the heads and tails of c
head(c,3)
```

```
##   a  b  c
## 1 1 -1 -4
## 2 2  0 -3
## 3 3  1 -2
```

```r
tail(c,3)
```

```
##   a b  c
## 3 3 1 -2
## 4 4 2 -1
## 5 5 3  0
```

```r
#Subsetting a dataframe (Similar for Matrix)
#2nd column of c
c[ ,2]
```

```
## [1] -1  0  1  2  3
```

```r
#Or
c[ ,"b"]
```

```
## [1] -1  0  1  2  3
```

```r
#2nd and 3rd column of c
c[ ,2:3]
```

```
##    b  c
## 1 -1 -4
## 2  0 -3
## 3  1 -2
## 4  2 -1
## 5  3  0
```

```r
#Element at 3rd row and 2nd column
c[3,2]
```

```
## [1] 1
```

#4.4 Arrays

```r
#Arrays are multidimensional vectors. SInce it is a vector, all elements of
an array must be of same data type. Subsetting elements are done using [].
Dimensions apart from rows and columns are called outer dimensions.
myArray = array(1:16, dim=c(2,4,4))# Total Elements product of all dimensions
= 2x4x4=16.
myArray
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    9   11   13   15
## [2,]   10   12   14   16
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

```
##
## , , 4
##
##      [,1] [,2] [,3] [,4]
## [1,]    9   11   13   15
## [2,]   10   12   14   16
```

```
myArray [1, ,]# Accessing all elements from Row 1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    9    1    9
## [2,]    3   11    3   11
## [3,]    5   13    5   13
## [4,]    7   15    7   15
```

```
myArray[1,2,3]# Accessing all elements from Row 1, column 2 and 3rd outer
dimension.
```

```
## [1] 3
```

```
myArray[, ,4]# Accessing all elements of 4th outer dimension
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    9   11   13   15
## [2,]   10   12   14   16
```

#5. Factors

```
#Factors are ordinal variables.
a <- c("Water","Air","Earth","Water","Air","Fire")
as.factor(a)
```

```
## [1] Water Air   Earth Water Air   Fire
## Levels: Air Earth Fire Water
```

```
# This will return only the unique values in the vector a. These unique value
are called levels.

factor(x=c("Water","Air","Earth","Water","Air","Fire"),
       levels = c("Water","Air","Earth","Fire"),
       ordered = TRUE)
```

```
## [1] Water Air   Earth Water Air   Fire
## Levels: Water < Air < Earth < Fire
```

#6. Missing values

```
# There are 2 kinds of missing data in R.
# NA
# NA stands for Not available. When an element that R is searching turns out
to be missing, R simply remembers that as NA
x <- c(1,5,3,7,5,8,NA,NA,7,3,NA)
length(x)
```

```
## [1] 11

#is.na returns a logical vector
is.na(x)

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE

!is.na(x)

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE

#To remove NAs manually
x[!is.na(x)]

## [1] 1 5 3 7 5 8 7 3

#NULL
#NULL represents an element which is present but stores no value. A NULL
value cannot exist as a part of an object
y <- c(1,3,5,NULL)
length(y)

## [1] 3

# is.null checks if a particular element is NULL or not
m = NULL
is.null(m)

## [1] TRUE
```

#7. Reading data

```
#read.csv and read.table can be used to read data into R console.
getwd()

## [1] "D:/Documents/MBA course materials/Term 2/DSA"

head(read.csv("bowens.csv")) #R finds the mentioned file in my current
working directory and reads the data into R console.

##                place east north
## 1          Abingdon   50    97
## 2      Admoor Copse   60    70
## 3      AERE Harwell   48    87
## 4     Agates Meadow   70    73
## 5       Aldermaston   59    65
## 6 Aldermaston Court   60    65

#read.csv(file.choose()) #Allows the user to chose the file to be read.

read.table("D:\\Documents\\MBA course materials\\viz.txt")# Reads txt file
from the specified location
```

```
##      V1   V2
## 1 S.No Size
## 2    1    5
## 3    2   78
## 4    3    3
## 5    4   34
## 6    5   76
## 7    6   12
## 8    6  343
```

#8. Functions

```
#Function is a data structure in R. The arguments of the function are
sepcified within the parenthesis.
Concat <- function(a,b) #Concat is a function to concatenate 2 words
{

  print(c(a,b))#Body of the function

}
Concat("Nruhari","Viswanath")

## [1] "Nruhari"   "Viswanath"

factors <-function(n) #Function to find out the factors of an integer and
print it.
{
  j <-0 #Counter variable to keep count of the no. of factors
  for(i in 1:n)#For loop construct to determine factors
  {
    if(n%%i==0)#Criteria for a factor of any number
      {
      print(paste("Factor is",i))
    j <-j+1#Updation of counter
  }

  }
  print(paste("No of factors is ", j))
}
factors(120)#This line is the function call. Here 120 is matched with the
argument n defined above in the function declaration.

## [1] "Factor is 1"
## [1] "Factor is 2"
## [1] "Factor is 3"
## [1] "Factor is 4"
## [1] "Factor is 5"
## [1] "Factor is 6"
## [1] "Factor is 8"
## [1] "Factor is 10"
## [1] "Factor is 12"
```

```
## [1] "Factor is 15"
## [1] "Factor is 20"
## [1] "Factor is 24"
## [1] "Factor is 30"
## [1] "Factor is 40"
## [1] "Factor is 60"
## [1] "Factor is 120"
## [1] "No of factors is  16"
```

#9. Builtin datasets

```
data(mtcars) #Loads mtcars dataset
tail(mtcars,5) #Prints last 5 rows of mtcars dataset

##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

#10. STatistics #10.1 Summary statistics

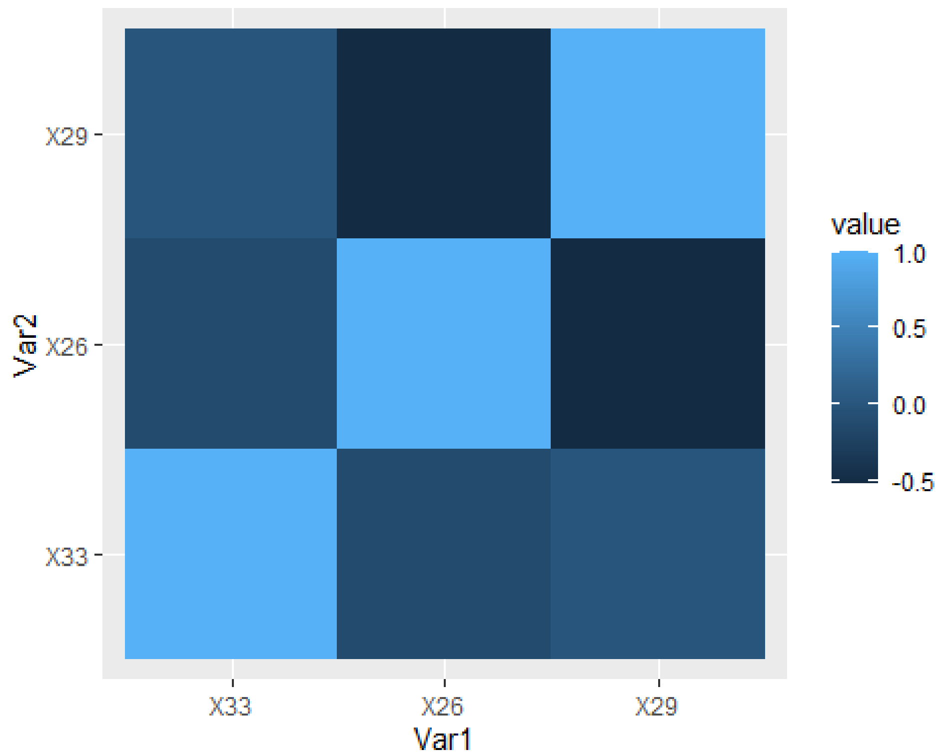
```
pkg <- c("ggplot2","reshape2","UsingR")
install.packages(pkg,repo="http://cran.us.r-project.org")

## Installing packages into 'C:/Users/Shwanath-Pc/Documents/R/win-
library/3.6'
## (as 'lib' is unspecified)

## package 'ggplot2' successfully unpacked and MD5 sums checked
## package 'reshape2' successfully unpacked and MD5 sums checked
## package 'UsingR' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Shwanath-Pc\AppData\Local\Temp\RtmpCETjT5\downloaded_packages

library(ggplot2) #Load ggplot2

library(reshape2) #Load reshape2

library(UsingR) #Loading usingR package

## Loading required package: MASS

## Loading required package: HistData

## Loading required package: Hmisc

## Loading required package: lattice

## Loading required package: survival
```

```
## Loading required package: Formula

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, units

##
## Attaching package: 'UsingR'

## The following object is masked from 'package:survival':
##
##     cancer

# To Generate a random sample of 10 numbers between 1 and 500 with
replacement
x = sample(x=1:500,20, replace = TRUE)#Numbers in x will repeat now because
replace=TRUE.
# The output x is a vector of 20 random numbers
x

##  [1] 103 222 313 419  69 357 158 358  13 413 151 336  21  41 479  19 428
261 390
## [20] 162

# To Generate a random sample of 10 numbers between 1 and 500 without
replacement
x = sample(x=1:500,20, replace = FALSE) #No number in x will repeat now.
# Simple Arithmetic Mean
mean(x)

## [1] 241.55

y = sample(c(x,rep(NA,10)),10)#Random sample of 10 numbers from x and a
repetition vector of 10 NA's.
y

##  [1]  NA  70 297 265 219 292 454 176 119  23

# y contains NAs so mean(y) will return NA. So NA's need to be removed while
computing mean.
mean(y, na.rm=TRUE) #Mean value computed because NA's are removed

## [1] 212.7778

# Weighted Mean
Concentration = c(30,20,25,50)
Weights = c(.25,.25,.3,.2)
weighted.mean(Concentration,Weights)# Weighted average of Concentrations

## [1] 30
```

```r
#Variance
var(x)

## [1] 16949.1

# Standard Deviation
sqrt(var(x))

## [1] 130.1887

sd(x)

## [1] 130.1887

sd(y)

## [1] NA

sd(y, na.rm=TRUE)

## [1] 132.6987

# Other Functions
min(x)

## [1] 23

max(x)

## [1] 477

median(x)

## [1] 229.5

median(y) #Will return NA

## [1] NA

median(y, na.rm=TRUE)

## [1] 219

# Summary Statistics
summary(x)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    23.0   163.0   229.5   241.6   302.5   477.0

summary(y)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    23.0   119.0   219.0   212.8   292.0   454.0       1
```

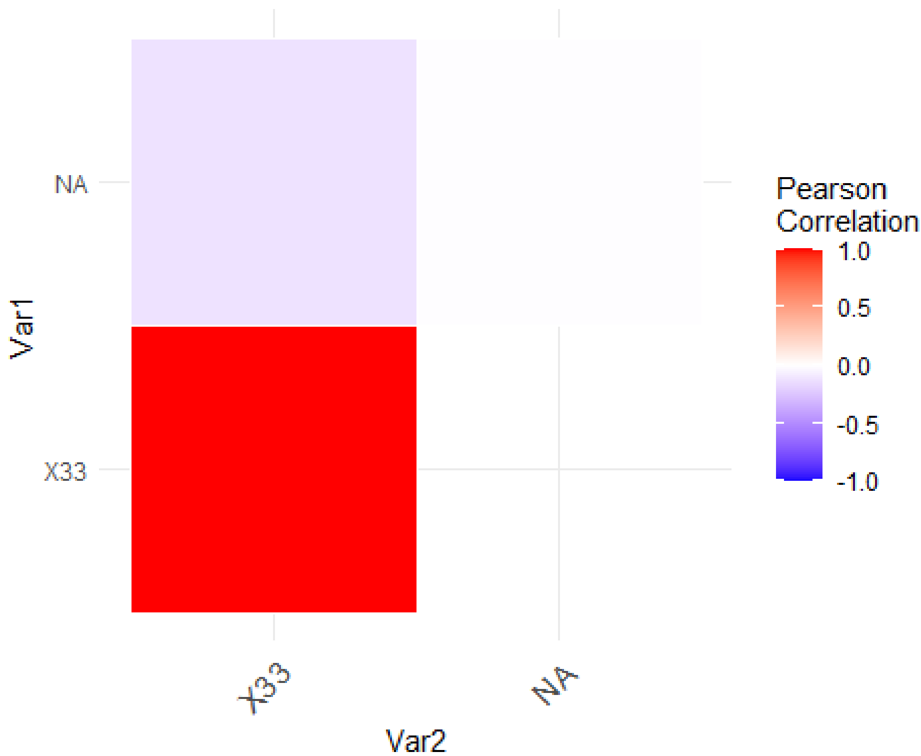
```r
# Quantiles
quantile(x, c(0.25, 0.75)) # Calculates 25th and 75th Quantile of x

##   25%   75% 
## 163.0 302.5

quantile(x, c(0.2,0.28,0.59, 0.75,0.99))

##    20%    28%    59%    75%    99% 
## 139.80 171.24 268.99 302.50 472.63

quantile(y, c(0.25, 0.75), na.rm = TRUE)

## 25% 75% 
## 119 292

#Correlation
data <- read.csv("New.csv")
cor(data[ ,2],data[ ,3])#Calculates the correlation coefficient between the 2nd and 3rd columns

## [1] -0.1237765

#Correlation between multiple variables
ecor = cor(data[ ,c(2,3,4)])# Returns a matrix with 3 rows and 3 columns and the correlation coefficients as the table elements
ecor

##              X33        X26          X29
## X33  1.000000000 -0.1237765 -0.007762953
## X26 -0.123776540  1.0000000 -0.523479265
## X29 -0.007762953 -0.5234793  1.000000000

# We can use the melt function to change this format.
emelt = melt(ecor)# This will return a long table with first 2 columns as x and y(variables) and 3rd column as the correlation coefficient

# Display the molten data frame
emelt

##   Var1 Var2        value
## 1  X33  X33  1.000000000
## 2  X26  X33 -0.123776540
## 3  X29  X33 -0.007762953
## 4  X33  X26 -0.123776540
## 5  X26  X26  1.000000000
## 6  X29  X26 -0.523479265
## 7  X33  X29 -0.007762953
## 8  X26  X29 -0.523479265
## 9  X29  X29  1.000000000
```

```r
#Correlation heatmap
#This returns the heat map of the molten table
ggplot(data = emelt, aes(Var1, Var2, fill=value)) +
  geom_tile()
```



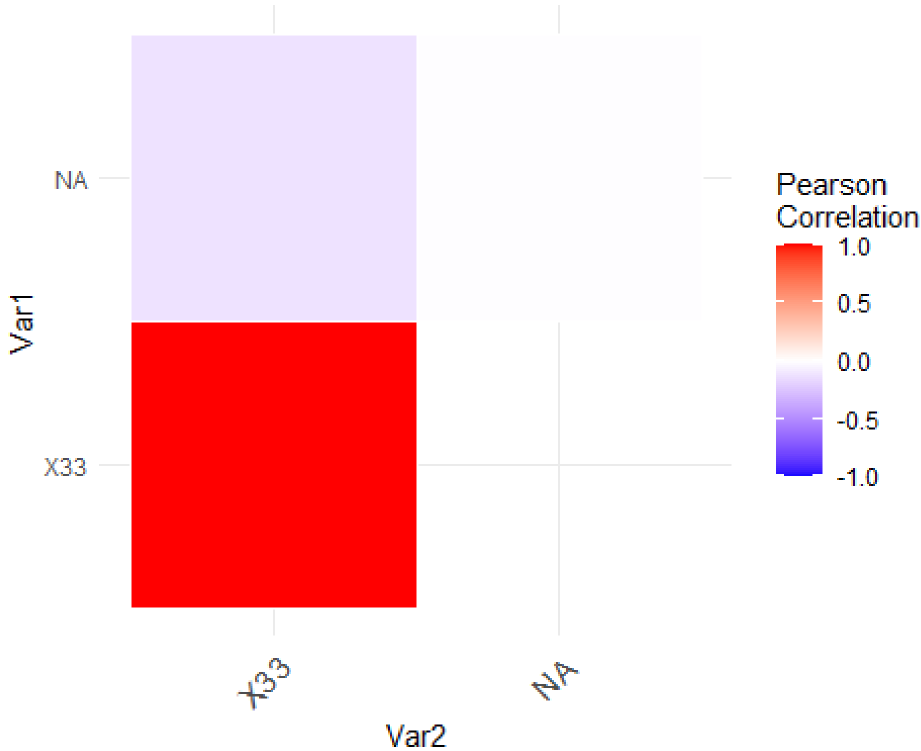

```r
# Get lower triangle of the correlation matrix
get_lower_tri<-function(emelt){
  emelt[upper.tri(emelt)] <- NA
  return(emelt)
}
# Get upper triangle of the correlation matrix
get_upper_tri <- function(emelt){
  emelt[lower.tri(emelt)]<- NA
  return(emelt)
}

upper_tri <- get_upper_tri(emelt)
upper_tri

##   Var1 Var2        value
## 1  X33  X33  1.000000000
## 2 <NA>  X33 -0.123776540
## 3 <NA> <NA> -0.007762953
## 4 <NA> <NA>           NA
## 5 <NA> <NA>           NA
## 6 <NA> <NA>           NA
## 7 <NA> <NA>           NA
```

```
## 8 <NA> <NA>           NA
## 9 <NA> <NA>           NA

#  Finished correlation matrix heatmap
# Melt the correlation data and drop the rows with NA values
# Melt the correlation matrix
melted_cormat <- melt(upper_tri, na.rm = TRUE)

## Using Var1, Var2 as id variables

# Heatmap

ggplot(data = melted_cormat, aes(Var2, Var1, fill = value))+
  geom_tile(color = "white")+
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab",
                       name="Pearson\nCorrelation") +
  theme_minimal()+
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                   size = 12, hjust = 1))+
  coord_fixed()
```

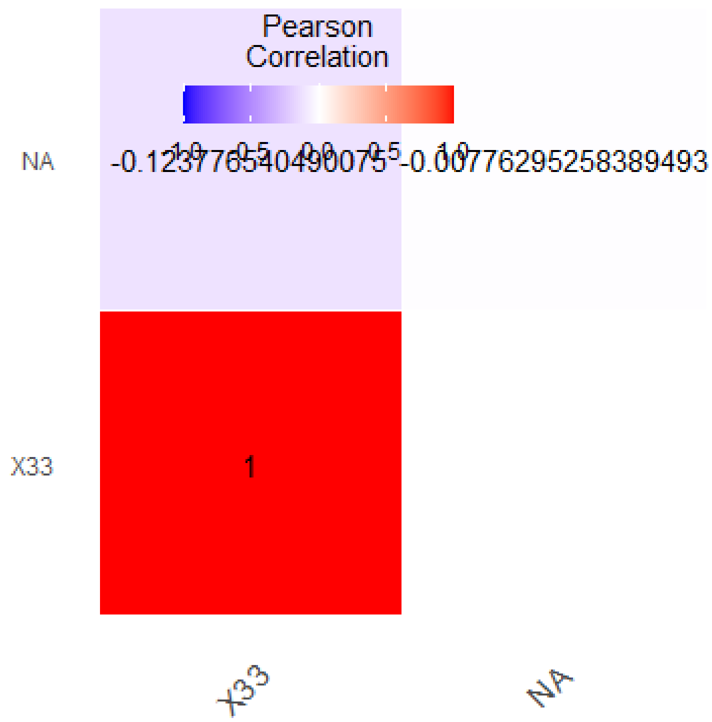



```
# negative correlations are in blue color and positive correlations in red.
# The function scale_fill_gradient2 is used with the argument limit = c(-1,1)
as correlation coefficients range from -1 to 1.
# coord_fixed() : this function ensures that one unit on the x-axis is the
same length as one unit on the y-axis.
```

```r
# Reorder the correlation matrix

# This section describes how to reorder the correlation matrix according to
the correlation coefficient.
# This is useful to identify the hidden pattern in the matrix.

reorder_cormat <- function(emelt){
  # Use correlation between variables as distance
  dd <- as.dist((1-emelt)/2)
  hc <- hclust(dd)
  emelt <-emelt[hc$order, hc$order]
}

# Reorder the correlation matrix

upper_tri <- get_upper_tri(emelt)
# Melt the correlation matrix
melted_cormat <- melt(upper_tri, na.rm = TRUE)

## Using Var1, Var2 as id variables

# Create a ggheatmap
ggheatmap <- ggplot(melted_cormat, aes(Var2, Var1, fill = value))+
  geom_tile(color = "white")+
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab",
                       name="Pearson\nCorrelation") +
  theme_minimal()+ # minimal theme
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                   size = 12, hjust = 1))+
  coord_fixed()
# Print the heatmap
print(ggheatmap)
```




```
#Add correlation coefficients on the heatmap

## Use geom_text() to add the correlation coefficients on the graph
## Use a blank theme (remove axis labels, panel grids and background, and
axis ticks)

ggheatmap +
  geom_text(aes(Var2, Var1, label = value), color = "black", size = 4) +
  theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    panel.grid.major = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.ticks = element_blank(),
    legend.justification = c(1, 0),
    legend.position = c(0.6, 0.7),
    legend.direction = "horizontal")+
  guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
                               title.position = "top", title.hjust = 0.5))
```



#10.2 Hypothesis testing

```
data(tips)
head(tips)

##   total_bill  tip    sex smoker day   time size
## 1      16.99 1.01 Female     No Sun Dinner    2
## 2      10.34 1.66   Male     No Sun Dinner    3
## 3      21.01 3.50   Male     No Sun Dinner    3
## 4      23.68 3.31   Male     No Sun Dinner    2
## 5      24.59 3.61 Female     No Sun Dinner    4
## 6      25.29 4.71   Male     No Sun Dinner    4

#'$' symbol can be used to subset a named column from a data frame.
unique(tips$sex) #Returns the unique values in the column 'sex' in tips
dataset

## [1] Female Male
## Levels: Female Male

unique(tips$day) #Returns the unique values in the column 'day' in tips
dataset

## [1] Sun  Sat  Thur Fri
## Levels: Fri Sat Sun Thur

#One Sample t-test, population standard deviation unknown (and hence t test)
#Only one group, two tailed test
```

```r
#Null hypothesis Ho: mu = 2.5
t.test(tips$total_bill, alternative = "two.sided", mu=2.5)

##
##  One Sample t-test
##
## data:  tips$total_bill
## t = 30.331, df = 243, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 2.5
## 95 percent confidence interval:
##  18.66333 20.90855
## sample estimates:
## mean of x
##  19.78594

#One Sample t-test
#Null hypothesis Ho: mu < 2.5
t.test(tips$total_bill, alternative = "greater", mu=2.5)

##
##  One Sample t-test
##
## data:  tips$total_bill
## t = 30.331, df = 243, p-value < 2.2e-16
## alternative hypothesis: true mean is greater than 2.5
## 95 percent confidence interval:
##  18.84492      Inf
## sample estimates:
## mean of x
##  19.78594

#Two Sample T-test
#2 columns of data, population variances of the 2 samples can be equal or unequal
t.test(tip ~ sex, data = tips, var.equal = TRUE)

##
##  Two Sample t-test
##
## data:  tip by sex
## t = -1.3879, df = 242, p-value = 0.1665
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.6197558  0.1074167
## sample estimates:
## mean in group Female   mean in group Male
##             2.833448             3.089618

t.test(tip ~ sex, data = tips, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  tip by sex
## t = -1.4895, df = 215.71, p-value = 0.1378
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.5951448  0.0828057
## sample estimates:
## mean in group Female   mean in group Male
##             2.833448             3.089618

#Paired Two-Sample T-Test

head(father.son) #Name od the dataset is father.son (a part of usingR
package)

##    fheight  sheight
## 1 65.04851 59.77827
## 2 63.25094 63.21404
## 3 64.95532 63.34242
## 4 65.75250 62.79238
## 5 61.13723 64.28113
## 6 63.02254 64.24221

#It contains the heights of father and son. Since both variables are are of
same type, we go for paired sample t test.
write.csv(father.son, "Sample.csv") #Creates a csv file named Sample and
Writes the father.son dataset to the file.

#ANOVA is used to compare the population means of multiple groups
head(tips)

##   total_bill  tip    sex smoker day   time size
## 1      16.99 1.01 Female     No Sun Dinner    2
## 2      10.34 1.66   Male     No Sun Dinner    3
## 3      21.01 3.50   Male     No Sun Dinner    3
## 4      23.68 3.31   Male     No Sun Dinner    2
## 5      24.59 3.61 Female     No Sun Dinner    4
## 6      25.29 4.71   Male     No Sun Dinner    4

anova = aov(tip~total_bill,tips)
summary(anova)

##              Df Sum Sq Mean Sq F value Pr(>F)
## total_bill    1  212.4  212.42   203.4 <2e-16 ***
## Residuals   242  252.8    1.04
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

#10.3 Linear Regression

```r
#R can generate a regression line from a scatter diagram
#The dataset to be used is father.son dataset which contains the heights of
father ans son( This dataset is from usingR package which is already
installed)

head(father.son)
```

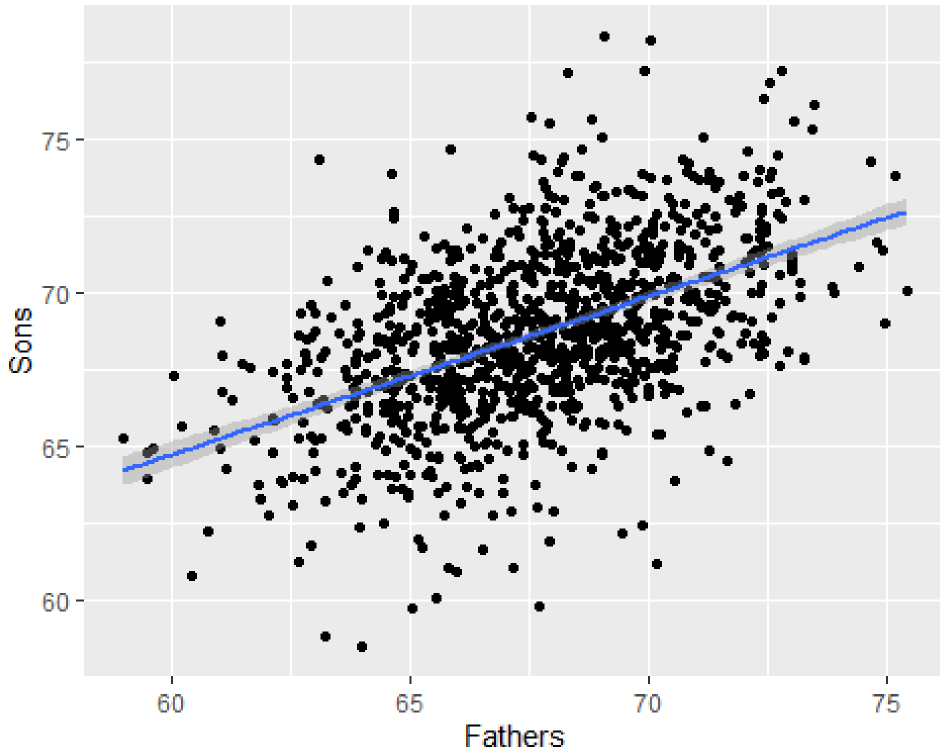
```
##    fheight  sheight
## 1 65.04851 59.77827
## 2 63.25094 63.21404
## 3 64.95532 63.34242
## 4 65.75250 62.79238
## 5 61.13723 64.28113
## 6 63.02254 64.24221
```

```r
#To regress fheight upon sheight variable
reg <- lm(fheight~sheight,father.son) #returns the intercept and slope of the
regression line
reg
```

```
##
## Call:
## lm(formula = fheight ~ sheight, data = father.son)
##
## Coefficients:
## (Intercept)      sheight
##     34.1075       0.4889
```

```r
ggplot(father.son, aes(x=fheight, y=sheight))+geom_point()+
  geom_smooth(method="lm")+labs(x="Fathers", y="Sons")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
#This command plots the scatter points of the entire father.son dataset and along with it plots he regression line (method="lm")
summary(reg) # Returns summary of all residuals, adjusted R, t statistic and p value of the intercept and slope

##
## Call:
## lm(formula = fheight ~ sheight, data = father.son)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.3590 -1.6406  0.0761  1.6095  7.1044
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.10745    1.76826   19.29   <2e-16 ***
## sheight      0.48890    0.02572   19.01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.376 on 1076 degrees of freedom
## Multiple R-squared:  0.2513, Adjusted R-squared:  0.2506
## F-statistic: 361.2 on 1 and 1076 DF,  p-value: < 2.2e-16
```

#10.4 Multiple linear regression

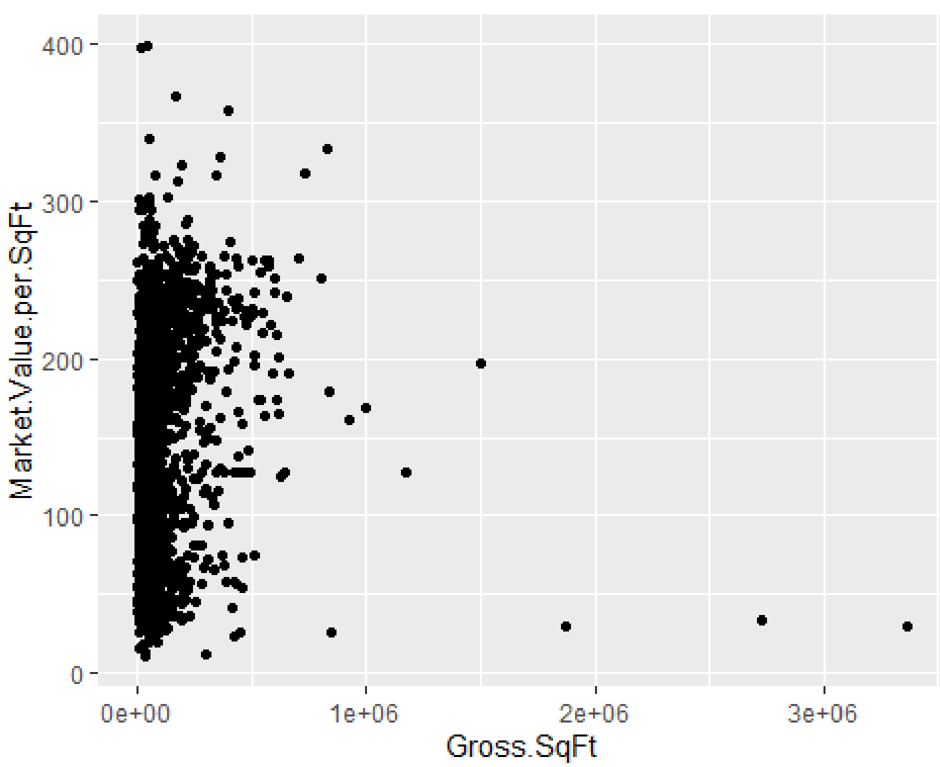
```r
housing <- read.csv("housing.csv")
head(housing)
```

```
##   Neighborhood Building.Classification Total.Units Year.Built Gross.SqFt
## 1    FINANCIAL          R9-CONDOMINIUM          42       1920      36500
## 2    FINANCIAL          R4-CONDOMINIUM          78       1985     126420
## 3    FINANCIAL          RR-CONDOMINIUM         500         NA     554174
## 4    FINANCIAL          R4-CONDOMINIUM         282       1930     249076
## 5      TRIBECA          R4-CONDOMINIUM         239       1985     219495
## 6      TRIBECA          R4-CONDOMINIUM         133       1986     139719
##   Estimated.Gross.Income Gross.Income.per.SqFt Estimated.Expense
## 1                1332615                 36.51            342005
## 2                6633257                 52.47           1762295
## 3               17310000                 31.24           3543000
## 4               11776313                 47.28           2784670
## 5               10004582                 45.58           2783197
## 6                5127687                 36.70           1497788
##   Expense.per.SqFt Net.Operating.Income Full.Market.Value
Market.Value.per.SqFt
## 1             9.37               990610           7300000
200.00
## 2            13.94              4870962          30690000
242.76
## 3             6.39             13767000          90970000
164.15
## 4            11.18              8991643          67556006
271.23
## 5            12.68              7221385          54320996
247.48
## 6            10.72              3629899          26737996
191.37
##        Boro
## 1 Manhattan
## 2 Manhattan
## 3 Manhattan
## 4 Manhattan
## 5 Manhattan
## 6 Manhattan
```

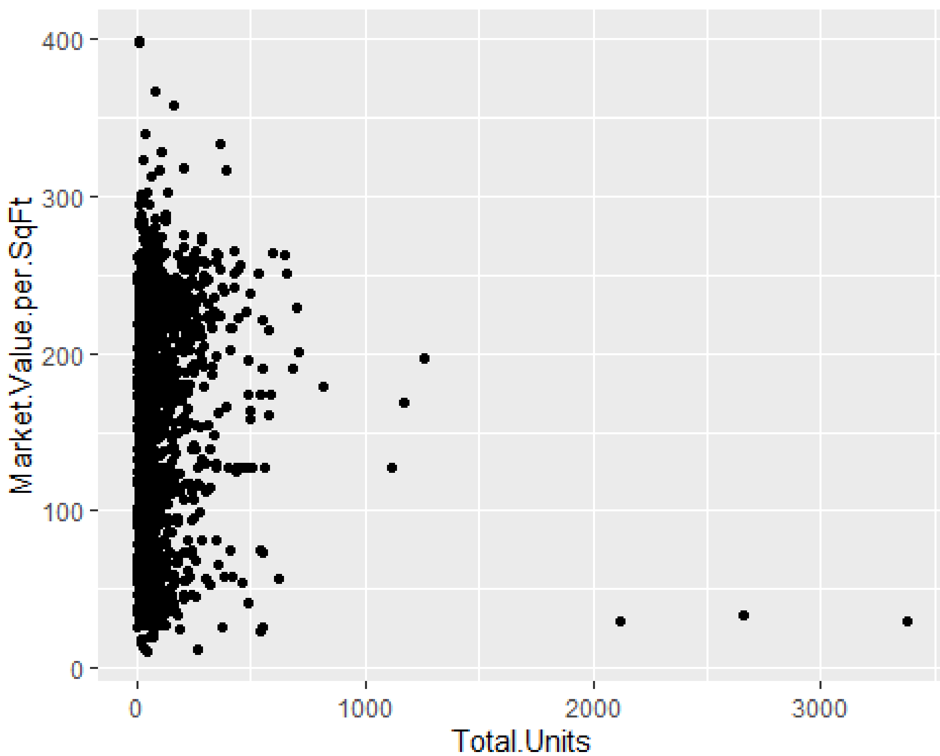
```r
str(housing) #GIves a sense of the nature and category of Total.Units
Gross.SqFtall the variables in the dataset
```

```
## 'data.frame':    2626 obs. of  13 variables:
##  $ Neighborhood           : Factor w/ 151 levels "ALPHABET CITY",..: 45 45
45 45 132 132 132 132 132 132 ...
##  $ Building.Classification: Factor w/ 4 levels "R2-CONDOMINIUM",..: 3 2 4
2 2 2 2 2 2 2 ...
##  $ Total.Units            : int  42 78 500 282 239 133 109 107 247 121 ...
##  $ Year.Built             : int  1920 1985 NA 1930 1985 1986 1985 1986
1987 1985 ...
```

```
##  $ Gross.SqFt             : int  36500 126420 554174 249076 219495 139719
105000 87479 255845 106129 ...
##  $ Estimated.Gross.Income : int  1332615 6633257 17310000 11776313
10004582 5127687 4365900 3637377 11246946 4115683 ...
##  $ Gross.Income.per.SqFt  : num  36.5 52.5 31.2 47.3 45.6 ...
##  $ Estimated.Expense      : int  342005 1762295 3543000 2784670 2783197
1497788 1273650 1061120 2440761 1231096 ...
##  $ Expense.per.SqFt       : num  9.37 13.94 6.39 11.18 12.68 ...
##  $ Net.Operating.Income   : int  990610 4870962 13767000 8991643 7221385
3629899 3092250 2576257 8806185 2884587 ...
##  $ Full.Market.Value      : int  7300000 30690000 90970000 67556006
54320996 26737996 22210281 19449002 66316999 21821999 ...
##  $ Market.Value.per.SqFt  : num  200 243 164 271 247 ...
##  $ Boro                   : Factor w/ 5 levels "Bronx","Brooklyn",..: 3 3
3 3 3 3 3 3 3 3 ...

#In this regression, the response variable is Market.Value.per.SqFt and the
regressors(input variables) are Total.Units and Gross.SqFt.

#To Plot  Market.Value.per.SqFt against Gross.SqFt
ggplot(housing, aes(x=Gross.SqFt, y=Market.Value.per.SqFt))+geom_point()
```




```
#To Plot  Market.Value.per.SqFt against Total.Units
ggplot(housing, aes(x=Total.Units, y=Market.Value.per.SqFt))+geom_point()
```




```
mreg = lm(Market.Value.per.SqFt~Total.Units+Gross.SqFt,housing) #returns the
intercept, slope for total units and slope for gross sqft.
mreg #Returns both call function and coefficients

##
## Call:
## lm(formula = Market.Value.per.SqFt ~ Total.Units + Gross.SqFt,
##     data = housing)
##
## Coefficients:
## (Intercept)  Total.Units   Gross.SqFt
##   1.211e+02   -3.819e-01    4.454e-04

summary(mreg)

##
## Call:
## lm(formula = Market.Value.per.SqFt ~ Total.Units + Gross.SqFt,
##     data = housing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -328.88  -51.05  -15.39   54.27  273.72
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.211e+02  1.476e+00   82.05   <2e-16 ***
```

```
## Total.Units -3.819e-01  3.280e-02  -11.64   <2e-16 ***
## Gross.SqFt   4.454e-04  3.077e-05   14.47   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 65.3 on 2623 degrees of freedom
## Multiple R-squared:  0.09304,    Adjusted R-squared:  0.09235
## F-statistic: 134.5 on 2 and 2623 DF,  p-value: < 2.2e-16
```

```
mreg$coefficients #Returns only the coefficients
```

```
##   (Intercept)   Total.Units    Gross.SqFt
##  1.211249e+02 -3.818753e-01  4.454146e-04
```

```
coefficients(mreg) #This function also returns only the coefficients
```

```
##   (Intercept)   Total.Units    Gross.SqFt
##  1.211249e+02 -3.818753e-01  4.454146e-04
```

Learnings from DSA assignment:

- R is a vectorized language. Any operation that can be performed on one element can also be performed on an entire vector without the need of additional loop.
- R is dynamically typed language. This means that R requires a variable to be defined before being used anywhere. This also means that R does not require explicit variable declaration before using the variable.
- A function requires a function call outside the function declaration and body. In the function call statement, the values need to be passed to the arguments of the function. R also performs partial matching of values and arguments.
- Data frame is preferred over matrix because the former does not require its elements to be of same data type.
- str() command is the most useful command to understand the nature, number of variables, types of variables, number of entries of a dataset.
- While knitting an R markdown file which contains a chunk with package installation and loading commands, the source of the repository needs to be mentioned in the package installation statement.