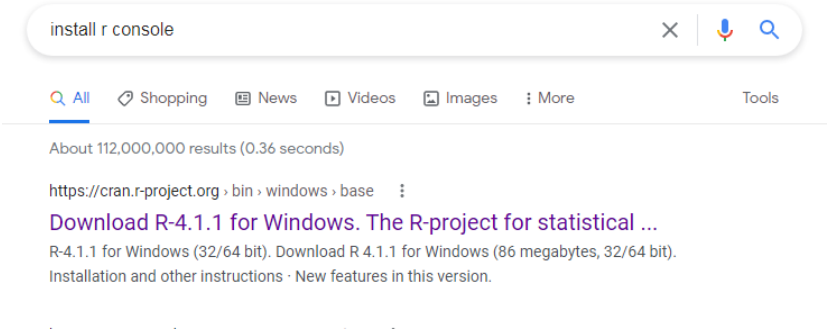
1. **Write down the installation of R**.

Ans. To install R Console in your PC follow the steps given below.

1. Search for "Install r console" in your browser.

2. Click on the first link that appears in the search result.
   i.e., https://cran.r-project.org/bin/windows/base/



3. Click on Download R 4.1.1. For windows

4. Now install the R console and click next.

2. Perform the following operations on X <- c(2,3,4) y <- c(5,6,2)

i. y^x ii. (1:10)^3 iii. sqrt(1:6) iv. sort(c(9,7,6,3,2))

Ans.

i. y^x

Code:

```
>  x <- c(2,3,4)   #Creates dataframe x
> y <- c(5,6,2)    #Creates dataframe y
> print(y^x)       #Prints y^x
```

Output:

```
[1]  25 216  16
```

ii. (1:10)^3

Code:

```
> a <- (1:10)     #Creates a dataframe that contains 1 to 10 numbers
> b <- 3          #Creates a dataframe b with value 3 in it.
> print(a^b)      #Prints the value of (1:10)^3
```

(OR)

```
> (1:10)^3
```

Output:

```
 [1]    1    8   27   64  125  216  343  512  729 1000
```

iii. sqrt(1:6)

Code:

```
> sqrt(1:6)
```

(OR)

```
> x <- sqrt(1:6)
> print(x)
```

Output:

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
```

iv. sort(c(9,7,6,3,2))

Code:

```
> sort(c(9,7,6,3,2))
```

Output:

```
[1] 2 3 6 7 9
```

3. Perform the following with

A<-matrix(c(9,7,6,5),nrow=2,ncol=2) & B<-matrix(c(1,1,0,1),nrow=2,ncol=2)

i. A%*%B ii. A*B iii. (A.inv<-solve(A))

Ans.

i. A%*%B

Code:

```
> A<-matrix(c(9,7,6,5),nrow=2,ncol=2)        #creates a 2x2 matrix A
> B<-matrix(c(1,1,0,1),nrow=2,ncol=2)        #creates a 2x2 matrix B
> print(A%*%B)
```

Output:

```
     [,1] [,2]
[1,]   15    6
[2,]   12    5
```

ii. A*B

Code:

```
> print(A*B)
```

Output:

```
     [,1] [,2]
[1,]    9    0
[2,]    7    5
```

iii. (A.inv<-solve(A))

Code:

```
> A.inv <- solve(A)
> print(A.inv)
```

Output:

```
          [,1] [,2]
[1,]  1.666667   -2
[2,] -2.333333    3
```

4. Calculate Fibonacci number up to 50.

Code:

```
> a <- 0
> b <- 1
> while(a<50){
+ print(a)
+ c=a+b
+ a=b
+  b=c
>  }
```

Note: + in above program lines at beginning indicates Shit+Enter(To type in next line)

Output:

```
[1] 0
[1] 1
[1] 1
[1] 2
[1] 3
[1] 5
[1] 8
[1] 13
[1] 21
[1] 34
```

5. Calculate n! of a number 6 by performing calculations.

Code:

```
n <- as.numeric(readline())

fact = 1

while(n != 1){

  fact = fact*n

  n = n-1

}

print(fact)
```

Output:

```
5

[1] 120
```

6. Perform matrix addition,subtraction and multiplication.

Code:

a) Addition of Matrix:

```
> A<-matrix(c(9,7,6,5,2,8,1,3,0),nrow=3,ncol=3)
> B<-matrix(c(3,4,5,1,2,3,6,9,1),nrow=3,ncol=3)
> cat("matrix addtion")
matrix addtion> A+B
     [,1] [,2] [,3]
[1,]   12    6    7
[2,]   11    4   12
[3,]   11   11    1
```

b) Subtraction of matrix:

```
> cat("matrix subtraction")
matrix subtraction> A-B
     [,1] [,2] [,3]
[1,]    6    4   -5
[2,]    3    0   -6
[3,]    1    5   -1
```

c) Multiplication of Matrix:

```
> A<-matrix(c(9,7,6,5,2,8,1,3,0),nrow=3,ncol=3)
> B<-matrix(c(3,4,5,1,2,3,6,9,1),nrow=3,ncol=3)
> cat("matrix multiplication")
matrix multiplication> A*B
     [,1] [,2] [,3]
[1,]   27    5    6
[2,]   28    4   27
[3,]   30   24    0
```

7. What is mapply function? Uses of mapply function. Create 4*4 matrix using a mapply function repeatedly four times.

Description:-

Mapply is a multivariate version of Sapply**. Mapply applies** FUN **to the first elements of each … argument, the second elements, the third elements, and so on. Arguments are recycled if necessary.**

USAGE:

mapply(FUN,...,MoreArgs=NULL,SIMPLIFY=TRUE,USE.NAMES=TRUE)

Arguments:

FUN : function to apply, found via match.fun**.**

… : arguments to vectorize over (vectors or lists of strictly positive length, or all of zero length)

.

MoreArgs : a list of other arguments to FUN**.**

SIMPLIFY : logical or character string; attempt to reduce the result to a vector, matrix or higher dimensional array;see the simplify **argument of** sapply**.**

USE.NAMES : logical; use names if the first … argument has names, or if it is a character vector, use that character vector as the names.

Code:

```
> mapply(rep,1:4,4)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    1    2    3    4
[3,]    1    2    3    4
[4,]    1    2    3    4
```

8. What is apply function? Generate the output by using apply function

1 6 1 6 1 6

2 7 2 7 2 7

3 8 3 8 3 8

4 9 4 9 4 9

5 10 5 10 5 10

DESCRIPTION:

Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

USAGE:

apply(X, MARGIN, FUN, …).

Arguments:

X : an array, including a matrix.

MARGIN : a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 **indicates rows,** 2 **indicates columns,** c(1, 2) **indicates rows and columns. Where** X **has named dimnames, it can be a character vector selecting dimension names.**

FUN : the function to be applied: see 'Details'. In the case of functions like +, %*%, etc., the function name must be backquoted or quoted.

… : optional arguments to FUN**.**

```
> b<-matrix(c(rep(c(1,6),3),rep(c(2,7),3),rep(c(3,8),3),rep(c(4,9),3),rep(c(5,10),3)),ncol=5,nrow=6)
> b
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]    1    2    3    4    5
[4,]    6    7    8    9   10
[5,]    1    2    3    4    5
[6,]    6    7    8    9   10
> t(b)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    6    1    6    1    6
[2,]    2    7    2    7    2    7
[3,]    3    8    3    8    3    8
[4,]    4    9    4    9    4    9
[5,]    5   10    5   10    5   10
> apply(t(b),2,sum)
[1] 15 40 15 40 15 40
```

9. Create 4*4 matrix using a call to repetition function repeatedly four times.

Description:

"rep" **replicates the values in** X**. It is a generic function, and the (internal) default method is described here.**

"rep.int" **and "***rep_len"* **are faster simplified versions for two common cases. Internally, they are generic, so methods can be defined for them.**

```
> v<-matrix(rep(1:4,4),nrow=4,ncol=4)
> v
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    2    2    2    2
[3,]    3    3    3    3
[4,]    4    4    4    4
```

10. Write about sequence function their syntax and some examples and execute them.

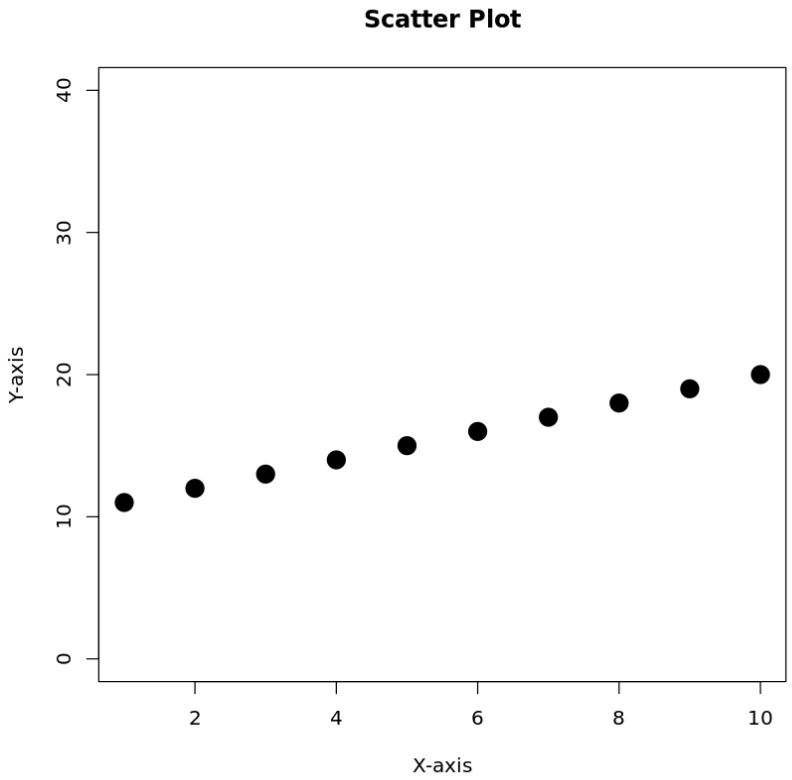
Description Generate regular sequences. seq is a standard generic with a default method. seq.int is a primitive which can be much faster but has a few restrictions. seq_along and seq_len are very fast primitives for two common cases. Arguments ... --- arguments passed to or from methods. from, to --- the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument. by --- number: increment of the sequence. length.out --- desired length of the sequence. A non-negative number, which for seq and seq.int will be rounded up if fractional. along.with --- take the length from the length of this argument.

```
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(23)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
[22] 22 23
> seq(1,20,2)
 [1]  1  3  5  7  9 11 13 15 17 19
> seq(1,10,5)
[1] 1 6
```

11. Why scatter plots are used? Draw a graph covering Cex & Pch, rep arguments and describe them.

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis. The simple scatterplot is created using the plot() function. Syntax The basic syntax for creating scatterplot in R is − plot(x, y, main, xlab, ylab, xlim, ylim, axes) Following is the description of the parameters used − x is the data set whose values are the horizontal coordinates. y is the data set whose values are the vertical coordinates. main is the tile of the graph. xlab is the label in the horizontal axis. ylab is the label in the vertical axis. xlim is the limits of the values of x used for plotting. ylim is the limits of the values of y used for plotting. axes indicates whether both axes should be drawn on the plot.

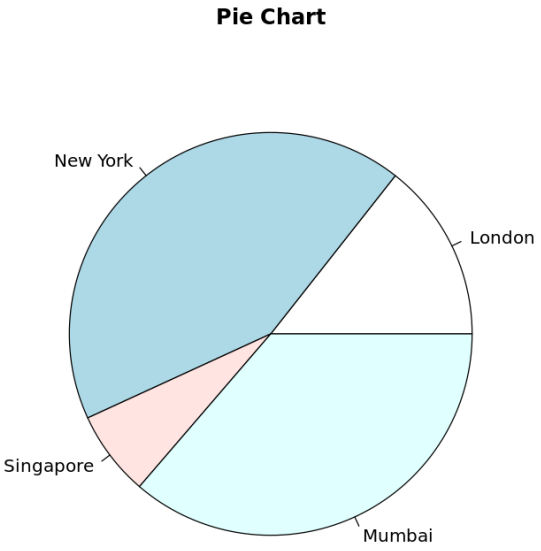
```
x<-rep(1:10)
y<-rep(11:20)
plot(x, y, main="Scatter Plot", xlab="X-axis", ylab="Y-axis",ylim=c(0,40),pch=19,cex=2)
```



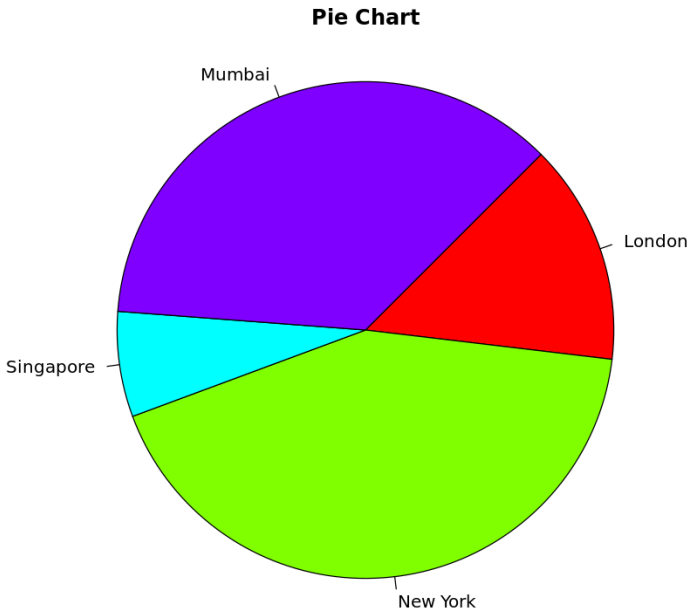
12. What is pie chart? Use the arguments clockwise, init.angle, Label and color arguments and describe them.

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart. In R the pie chart is created using the pie() function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc. Syntax: The basic syntax for creating a pie-chart using the R is − pie(x, labels, radius, main, col, clockwise) Following is the description of the parameters used − x is a vector containing the numeric values used in the pie chart. labels is used to give description to the slices. radius indicates the radius of the circle of the pie chart.(value between −1 and +1). main indicates the title of the chart. col indicates the color palette. clockwise is a logical value indicating if the slices are drawn clockwise or anti clockwise.

```
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

pie(x,labels,main="Pie Chart")
```



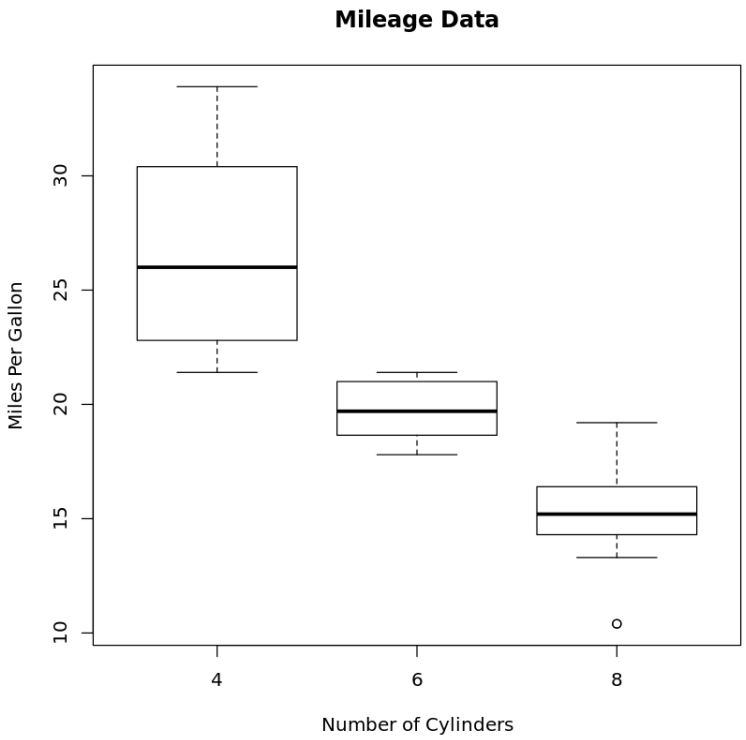

```
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

pie(x,labels,main="Pie Chart",clockwise=TRUE,radius=1,,init.angle=45,col = rainbow(length(x)))
```


Pie Chart
Mumbai
London
Singapore
New York

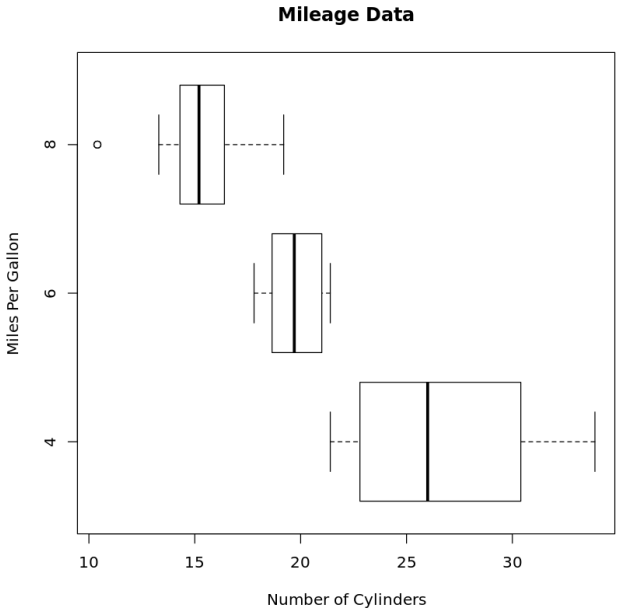
13. What is Box plot? What are the usage of it? Display the Boxplot for Car dataset. Use the title() and mention the arguments present in the Boxplot.

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them. Boxplots are created in R by using the boxplot() function Syntax: The basic syntax to create a boxplot in R is − boxplot(x, data, notch, varwidth, names, main) Following is the description of the parameters used − x -- is a vector or a formula. data -- is the data frame. notch -- is a logical value. Set as TRUE to draw a notch. varwidth -- is a logical value. Set as true to draw width of the box proportionate to the sample size. names -- are the group labels which will be printed under each boxplot. main -- is used to give a title to the graph.

```
input <- mtcars[,c('mpg','cyl')]
boxplot(mpg ~ cyl, data = input, xlab = "Number of Cylinders", ylab = "Miles Per Gallon", main = "Mileage Data")
```




```
boxplot(mpg ~ cyl, data = input, xlab = "Number of Cylinders", ylab = "Miles Per Gallon", main = "Mileage Data", horizontal=TRUE)
```


Mileage Data
Miles Per Gallon
Number of Cylinders
8
6
4
10
15
20
25
30

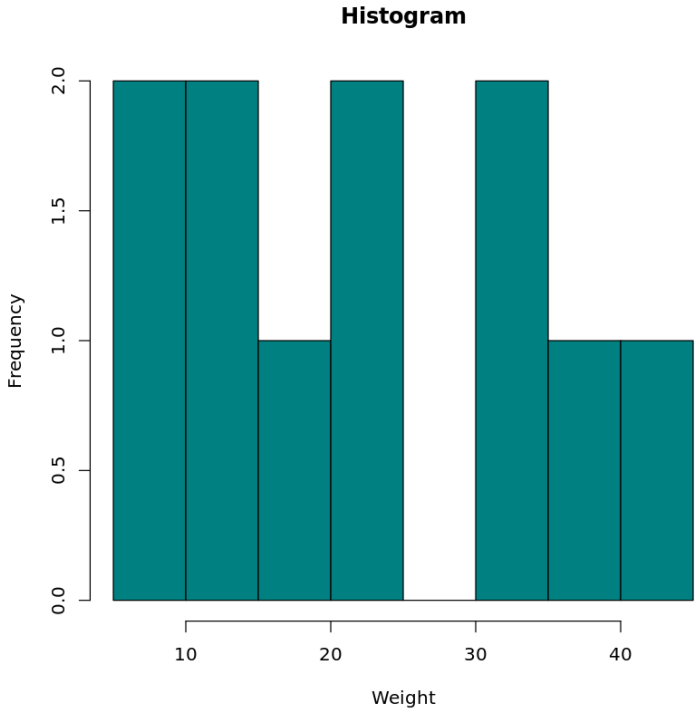
14. What is Histogram? Plot the histogram by using various arguments and describe those arguments.

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using hist() function. This function takes a vector as an input and uses some more parameters to plot histograms.

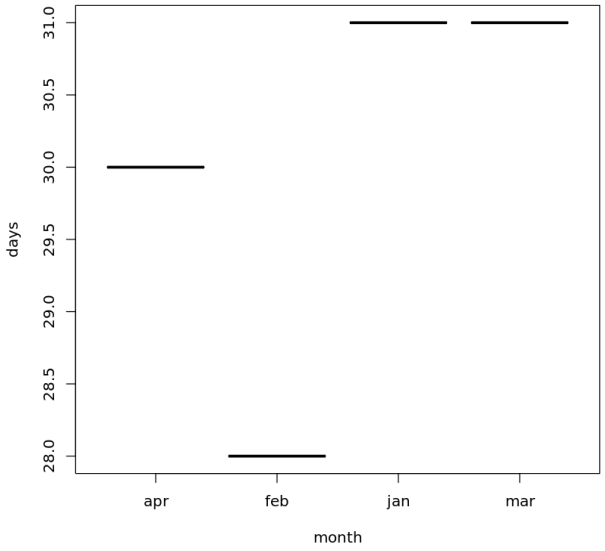
Syntax: The basic syntax for creating a histogram using R is − hist(v,main,xlab,xlim,ylim,breaks,col,border) Following is the description of the parameters used − v -- is a vector containing numeric values used in histogram. main -- indicates title of the chart. col -- is used to set color of the bars. border -- is used to set border color of each bar. xlab -- is used to give description of x-axis. xlim -- is used to specify the range of values on the x-axis. ylim -- is used to specify the range of values on the y-axis. breaks -- is used to mention the width of each bar.

```
v <-  c(9,13,21,8,36,22,12,41,31,33,19)
hist(v,main="Histogram",xlab = "Weight",col = "#008080",border = "black",breaks=10)
```

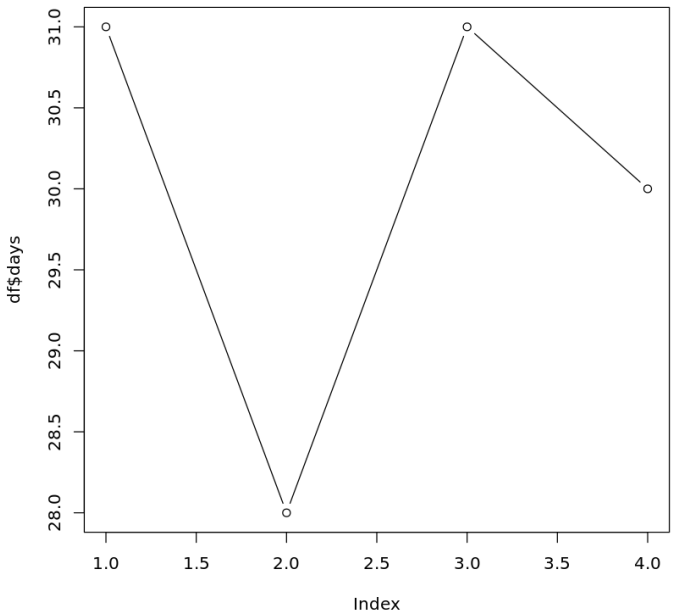


15. Draw a line chart using categorical data. Explain the syntax and several arguments present in them

```
df<-data.frame(month=c("jan","feb","mar","apr"),days=c(31,28,31,30))
plot(df,type="b")
```

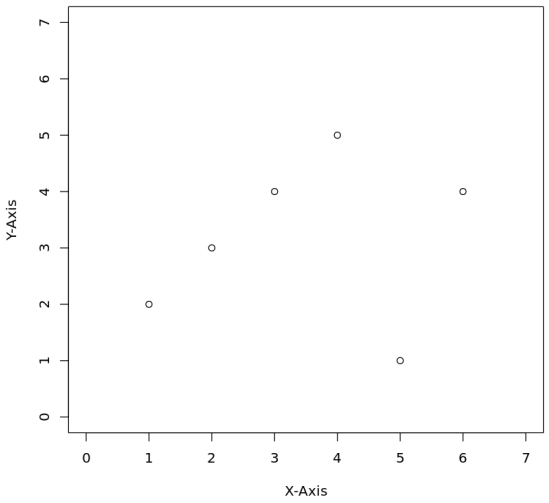



```
df<-data.frame(month=c("jan","feb","mar","apr"),days=c(31,28,31,30))
plot(df$days,type="b")
```

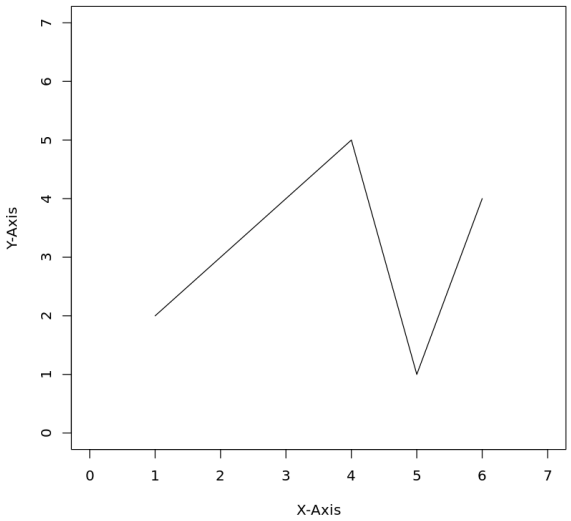


16. Define the point? Draw different points and mention the syntax of it.

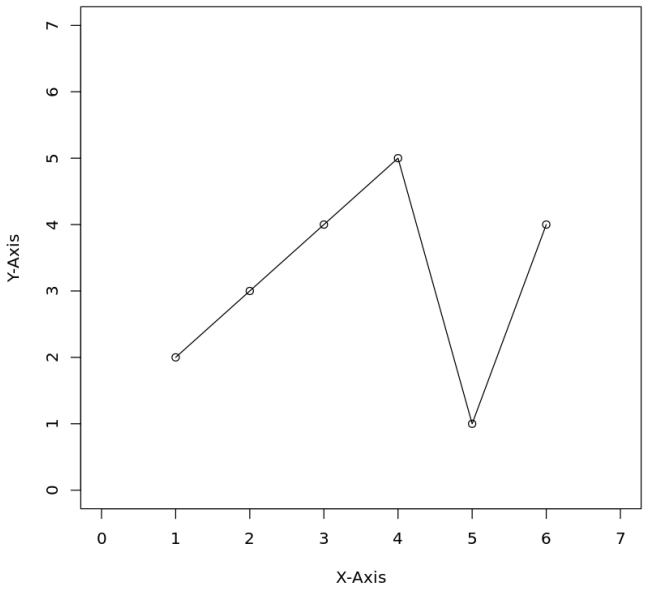
```
plot(0:7,0:7,type="n",xlab="X-Axis",ylab="Y-Axis")
x=c(1,2,3,4,5,6)
y=c(2,3,4,5,1,4)
points(x,y,type="p")
```



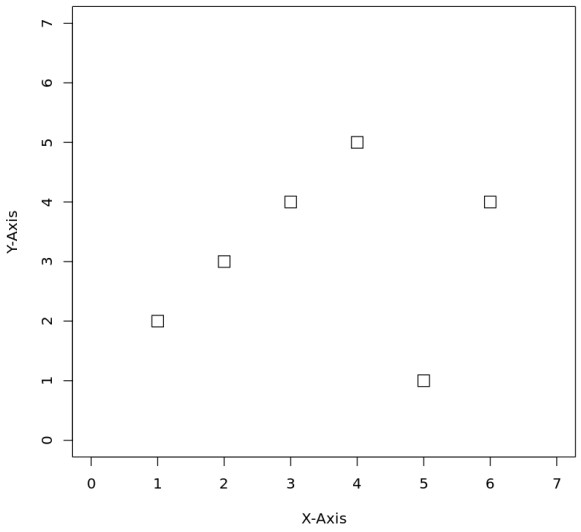

```
plot(0:7,0:7,type="n",xlab="X-Axis",ylab="Y-Axis")
x=c(1,2,3,4,5,6)
y=c(2,3,4,5,1,4)
points(x,y,type="l")
```



```r
plot(0:7,0:7,type="n",xlab="X-Axis",ylab="Y-Axis")
x=c(1,2,3,4,5,6)
y=c(2,3,4,5,1,4)
points(x,y,type="o")
```




```r
plot(0:7,0:7,type="n",xlab="X-Axis",ylab="Y-Axis")
x=c(1,2,3,4,5,6)
y=c(2,3,4,5,1,4)
points(x,y,pch=22,cex=2)  # pch=22 gives squares , cex=2 for size
```



17. By reading excel file perform mean and median for the data.

```
> install.packages("readxl")
Installing package into 'C:/Users/Nandu/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://mirror.niser.ac.in/cran/bin/windows/contrib/4.1/readxl_1.3.1.zip'
Content type 'application/zip' length 1717479 bytes (1.6 MB)
downloaded 1.6 MB

package 'readxl' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Nandu\AppData\Local\Temp\Rtmp0EUw9z\downloaded_packages
> library(readxl)
```

```
> library(readxl)
> library(Rcpp)
> data <- read_excel("C:/Users/Nandu/Desktop/employee_data.xlsx", sheet = "Sheet1")
> data
# A tibble: 10 x 3
   `Employ Id` Name     Salary
         <dbl> <chr>     <dbl>
 1           1 Jhon      20000
 2           2 Rose      30000
 3           3 Angela    35000
 4           4 Patrick   25000
 5           5 Lisa      34000
 6           6 Kimberly  23000
 7           7 Bonnie    20000
 8           8 Michael   50000
 9           9 Todd      24600
10          10 Joe       34500
> data$Salary
 [1] 20000 30000 35000 25000 34000 23000 20000 50000 24600 34500
> mean(data$Salary)
[1] 29610
> median(data$Salary)
[1] 27500
> |
```

Note: If above loaded packages were not installed in your computer please install the package by using command "install.packages(package_name)"

18. By using the mtcars dataset or Airquality dataset perform mean, median,quantiles and variance for the dataset.

```
summary(mtcars)
```

```
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

```
mean(mtcars$mpg)
median(mtcars$mpg)
var(mtcars$mpg)
quantile(mtcars$mpg)
```

20.090625

19.2

36.3241028225806

0%: 10.4 **25%:** 15.425 **50%:** 19.2 **75%:** 22.8 **100%:** 33.9

```
summary(airquality)
```

```
     Ozone           Solar.R          Wind             Temp      
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00  
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00  
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00  
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88  
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00  
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00  
 NA's   :37       NA's   :7                                       
     Month            Day       
 Min.   :5.000   Min.   : 1.0  
 1st Qu.:6.000   1st Qu.: 8.0  
 Median :7.000   Median :16.0  
 Mean   :6.993   Mean   :15.8  
 3rd Qu.:8.000   3rd Qu.:23.0  
 Max.   :9.000   Max.   :31.0  
```

```
mean(airquality$Ozone,na.rm=TRUE)
median(airquality$Ozone,na.rm=TRUE)
var(airquality$Ozone,na.rm=TRUE)
quantile(airquality$Ozone,na.rm=TRUE)
```

42.1293103448276

31.5

1088.20052473763

0%: 1 **25%:** 18 **50%:** 31.5 **75%:** 63.25 **100%:** 168

19.Define mapply function and perform the mapply function to generate the following output

a)	b)	c)
1 1 1 1	4	42
2 2 2	3 3	42 42
3 3	2 2 2	42 42 42
4	1 1 1 1	42 42 42 42 42

```
> mapply(rep,times=1:4,x=4:1)
[[1]]
[1] 4

[[2]]
[1] 3 3

[[3]]
[1] 2 2 2

[[4]]
[1] 1 1 1 1
```

```
> mapply(rep,1:4,4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4
```

```
> mapply(rep,times=1:4,MoreArgs=list(x=42))
[[1]]
[1] 42

[[2]]
[1] 42 42

[[3]]
[1] 42 42 42

[[4]]
[1] 42 42 42 42
```