V.E.S. Institute of Technology, Collector Colony,
Chembur, Mumbai, Maharashtra 400047
Department of M.C.A

INDEX

S. No.	Contents	Date Of Preparation	Date Of Submission	Marks	Faculty Sign
1.	To install R & implement basic commands of R	20-09-2024	30-09-2024		
2.	Data preprocessing techniques	27-09-2024	3-10-2024		
3.	To implements different types of charts and graphs like histogram, boxplots, bar graph, line graphs, Scatterplots, Pie chart	04-10-2024	10-10-2024		
4.	Implement data Visualization with Ggplot2	11-10-2024	17-10-2024		
5.	Implementation and analysis of Apriori Algorithm using Market Basket Analysis	25-10-2024	31-10-2024		
6.	Implementation and analysis of Linear regression through graphical methods.	25-10-2024	12-11-2024		
7.	Implementation and analysis of Classification algorithms like Naïve Bayesian, K-Nearest Neighbour, ID3, C4.5	25-10-2024	14-11-2024		
8.	Implementation and analysis of clustering algorithms like K- means and hierarchical clustering.	18-11-2024	22-11-2024		
9.	Implementation of various data visualization techniques using Tableau	18-11-2024	25-11-2024		
10.	Implementation and analysis of Reports & dashboard using Tableau	18-11-2024	28-11-2024		

Final Grade	Instructor Signature

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 1**	
Title of LAB Assignment: To Install R and Implement Basic commands in R.			
DOP: 20 – 09 – 2024		**DOS: 30 – 09 – 2024**	
CO Mapped: CO1	**PO Mapped: PO1**	**Faculty Signature:**	**Marks:**

Practical No: 1

AIM: To install R and Implement Basic Commands in R.

Theory:

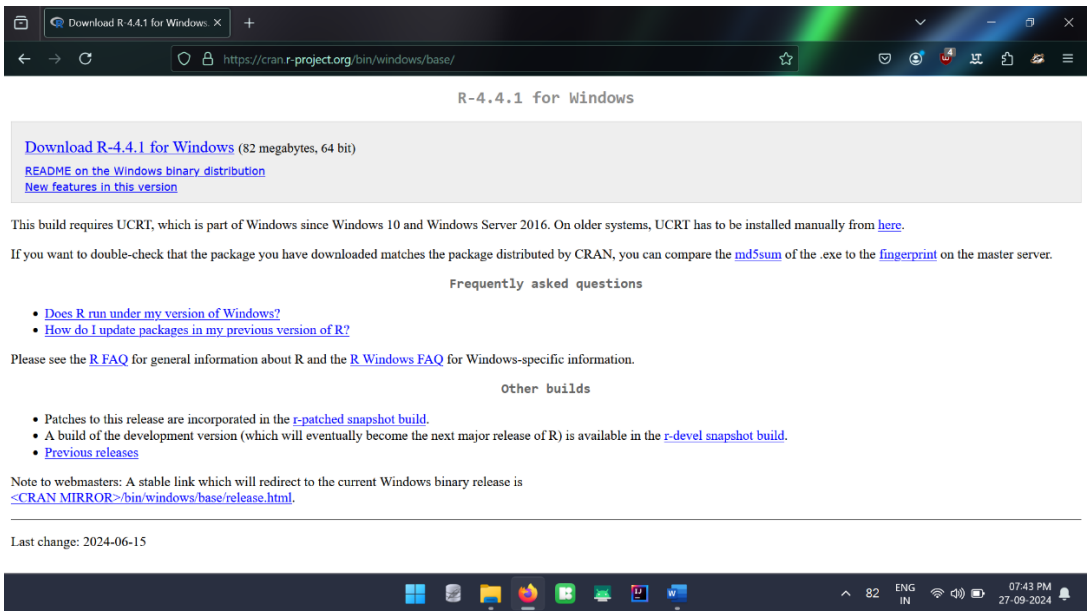
Introduction to R:

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity. One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Steps to Install R:

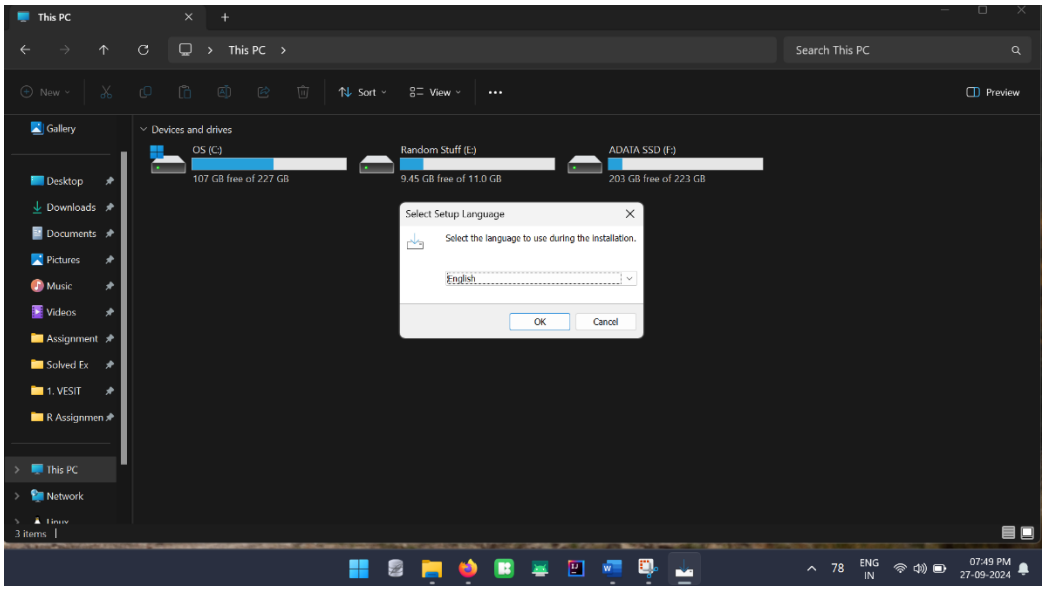
1. To install R, visit cran.r-project.org



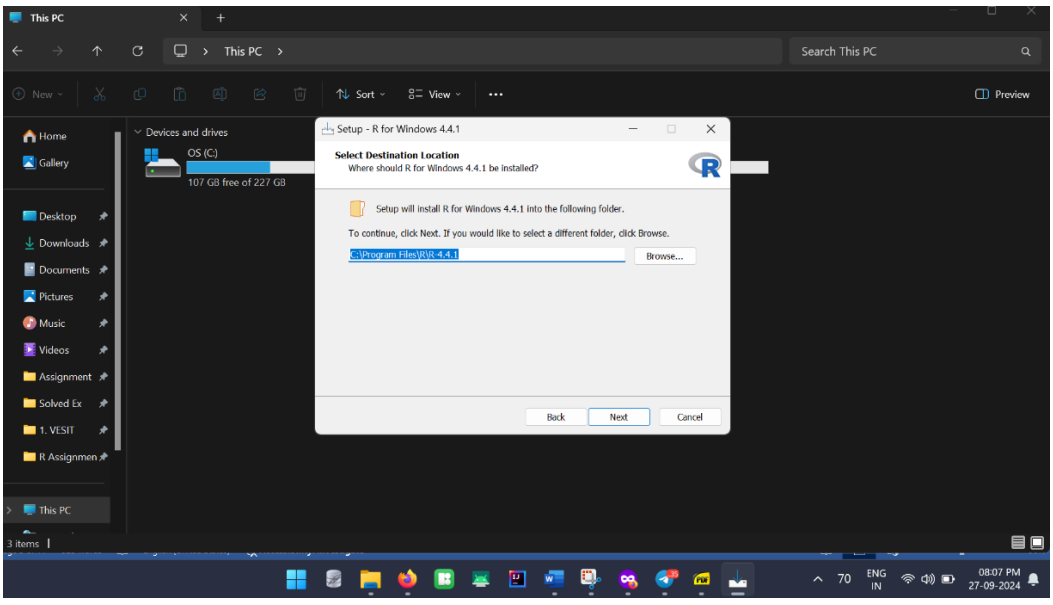
2. Click Download R for Windows.

3. Install R Click on install R for the first time.
4. Click Download R for Windows. Open the downloaded file.

5. Select the language you would like to use during the installation. Then click OK

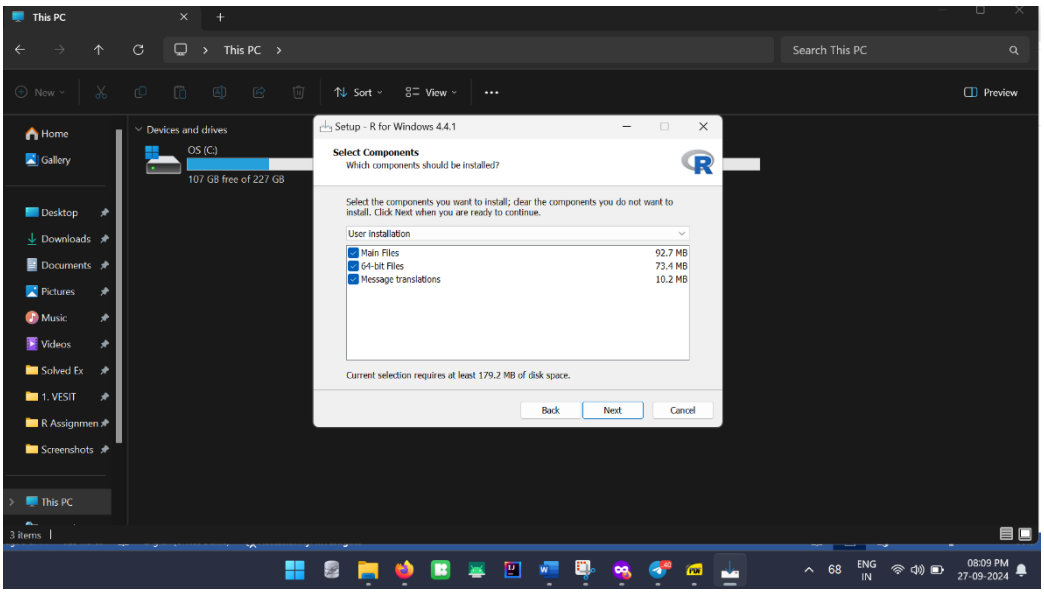


6. Select where you would like R to be installed. It will default to your Program Files on your C Drive. Click Next.



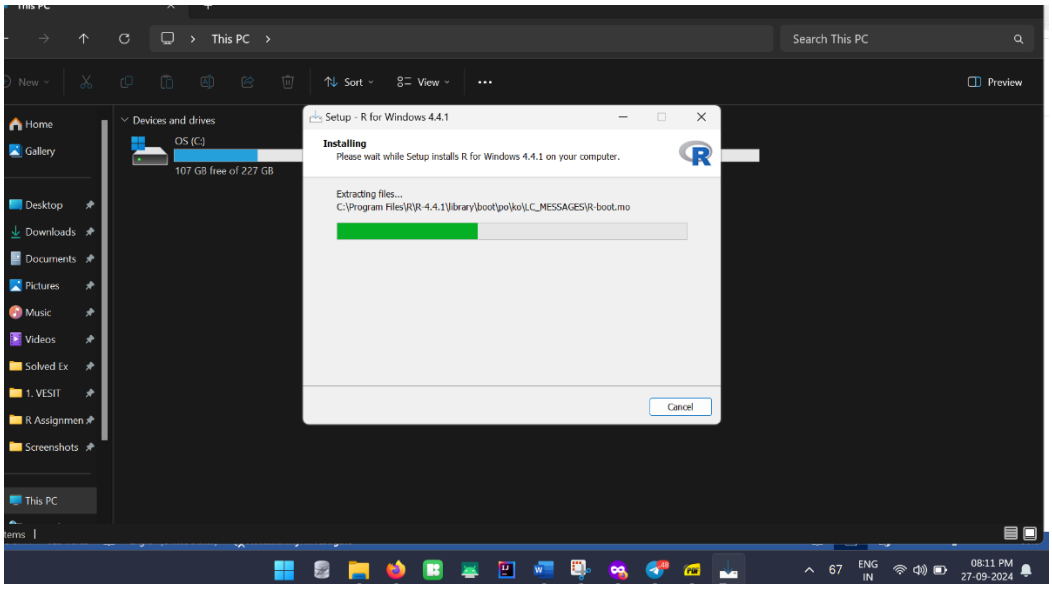
7. You can then choose which installation you would like.

8. If your computer is a 64-bit, you can choose the 64-bit User Installation. Then click Next.



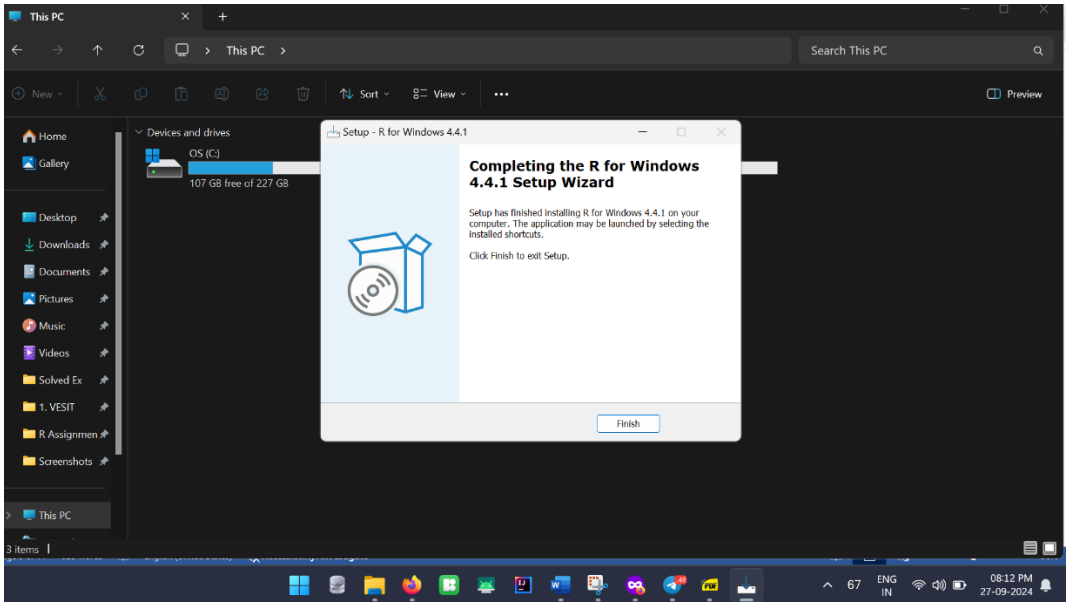
9. Then specify if you want to customized your startup or just use the defaults

10. Then you can choose the folder that you want R to be saved within or the default if the R folder that was created. Once you have finished, click Next.



11. You can then select additional shortcuts if you would like. Click Next.

12. Click Finish to complete the installation.



Introduction to RStudio:

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

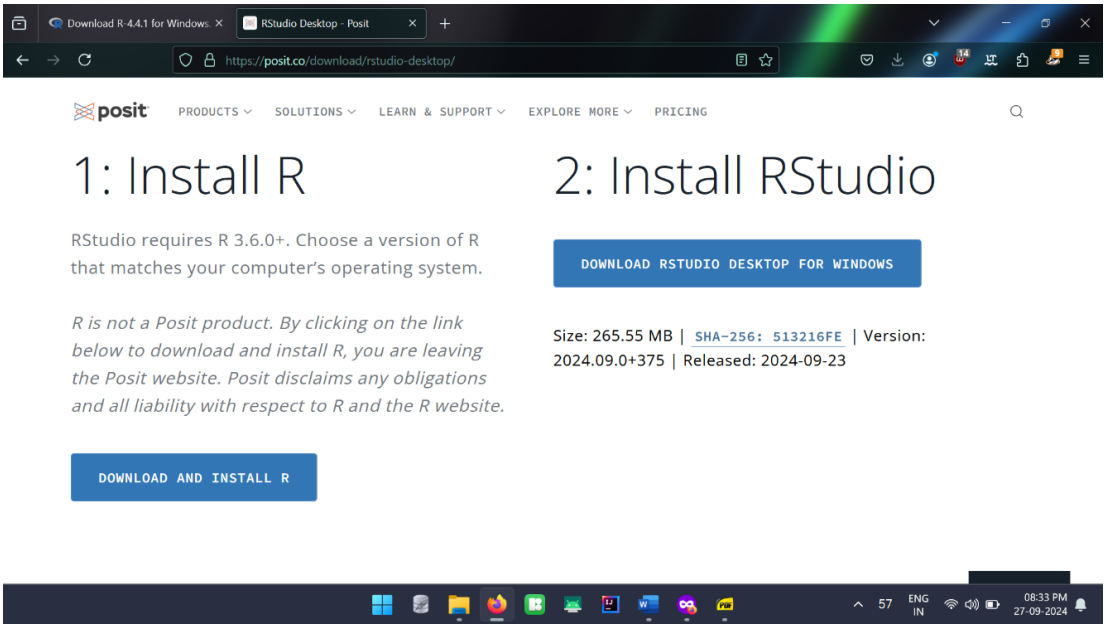
RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux). RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. JJ Allaire, creator of the programming language ColdFusion, founded RStudio. Hadley Wickham is the Chief Scientist at RStudio.

RStudio is available in two editions: RStudio Desktop, where the program is run locally as a regular desktop application; and RStudio Server, which allows accessing RStudio using a web browser while it is running on a remote Linux server

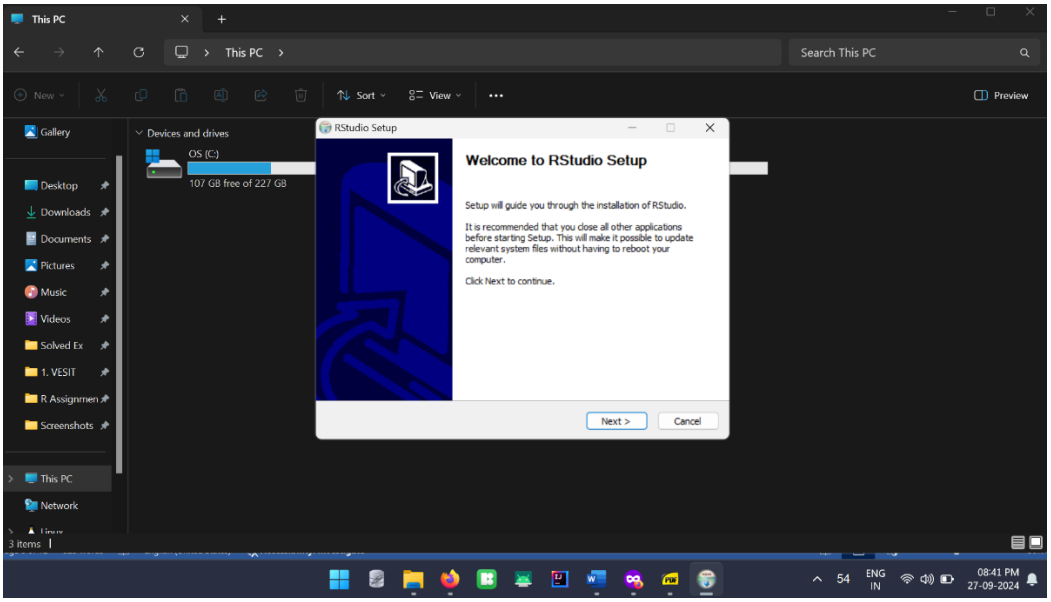
Steps to Install RStudio:

1. Visit to

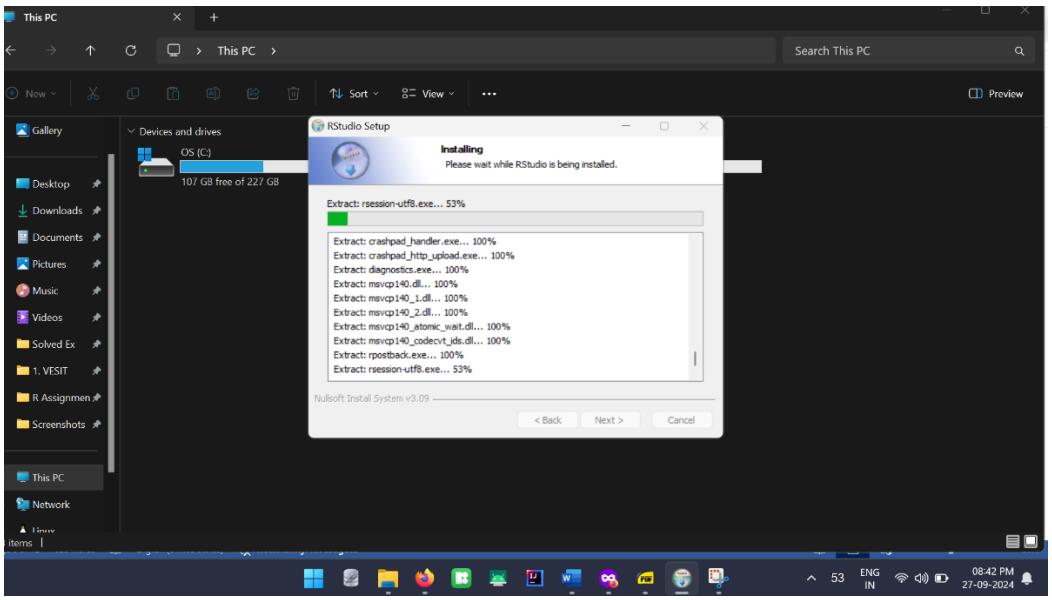
2. Click Download RStudio.



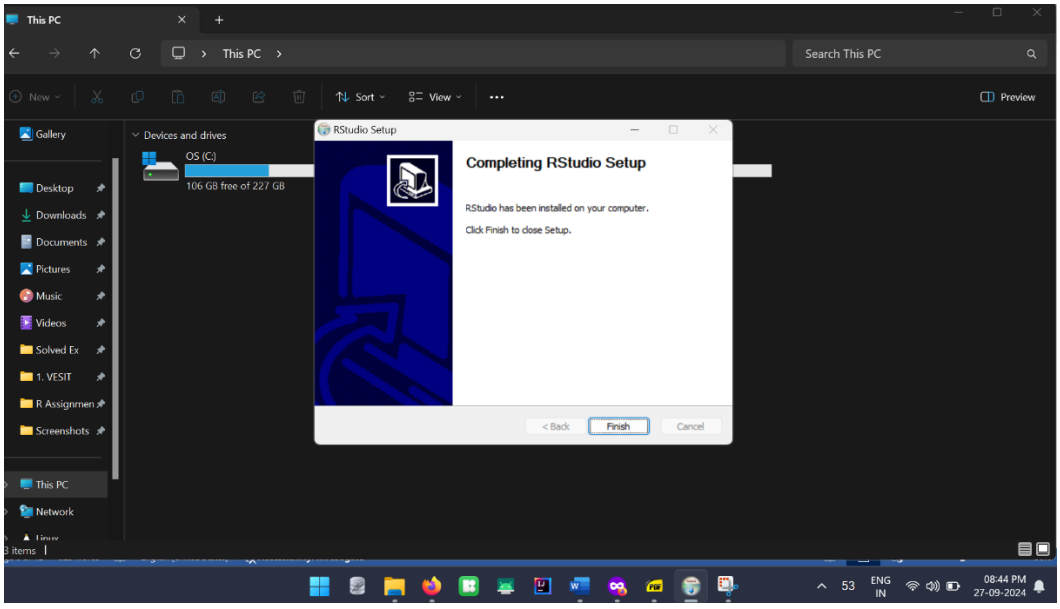
3. Once the packet has downloaded, the Welcome to RStudio Setup Wizard will open. Click Next and go through the installation steps.



4. After the Setup Wizard finishing the installation, RStudio will be installed.



5. RStudio is now successfully installed in your system.



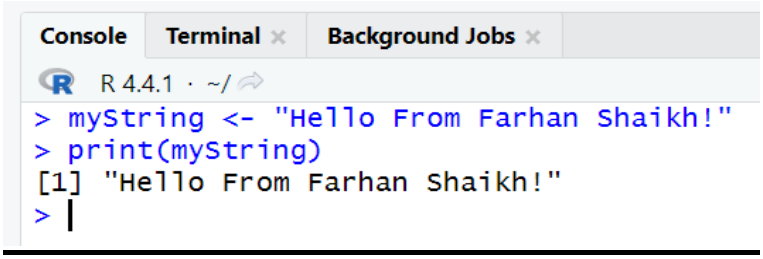
Solved R Reference Material

1. Printing String:

Source Code:

```
myString <- “Hello From Farhan Shaikh”

print(myString)
```

Output:



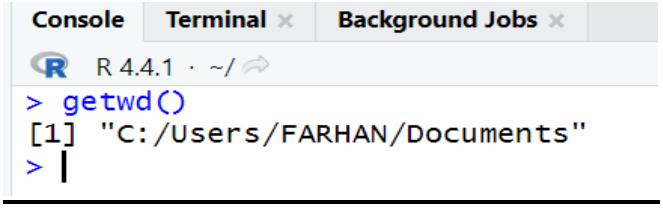
---

2. Get Current Working Directory:

Source Code:

```
getwd()
```

Output:



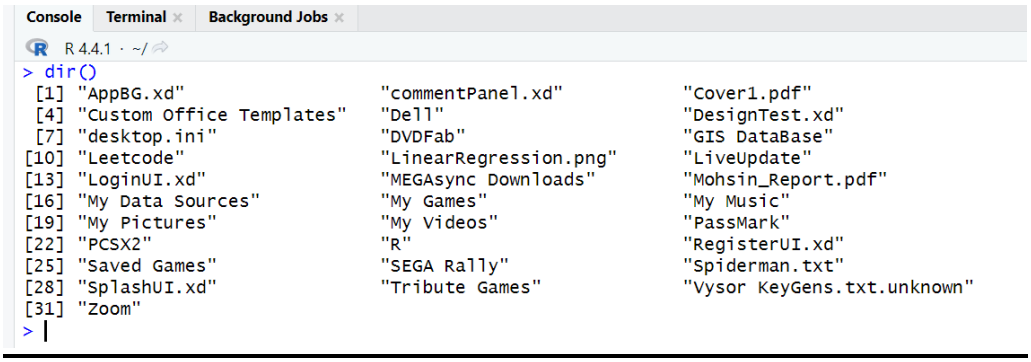
---

3. Get List of Directories:

Source Code:

```
dir()
```

Output:



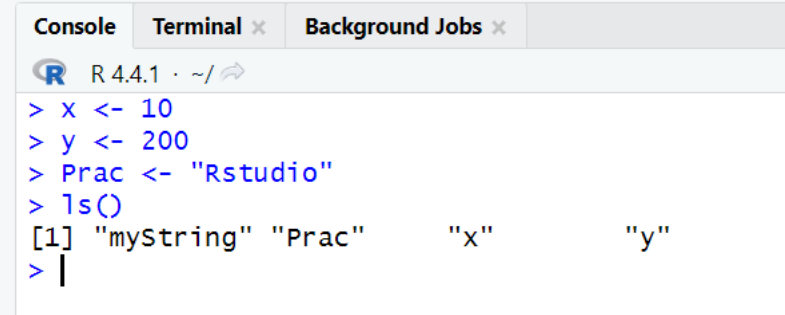
---

4. Get List names of Objects in R Environment:

Source Code:

```
ls()
```

Output:

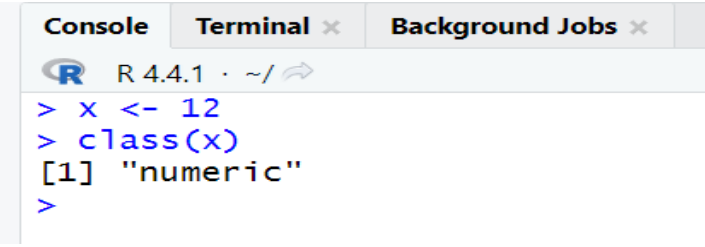


5. Checking Type of Object:

Source Code:

```
x <- 12

class(x)
```

Output:

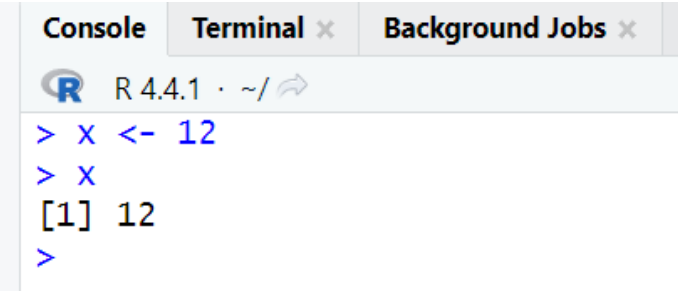


6. Example of Auto-Printing:

Source Code:

```
x <- 12

x
```

Output:

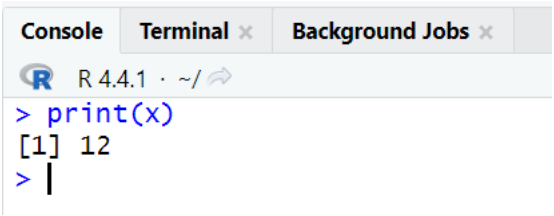


7. Example of Explicit-Printing:

Source Code:

```
print(x)
```

Output:



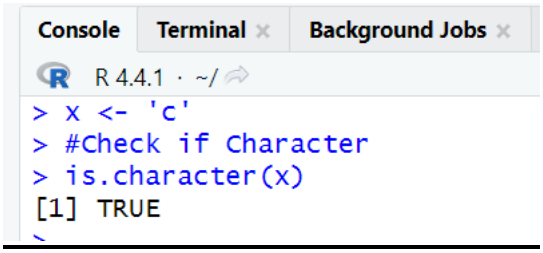
8. Check Data Type:

1. Character:

Source Code:

```
x <- 'c'

#Check if character

is.character(x)
```

Output:

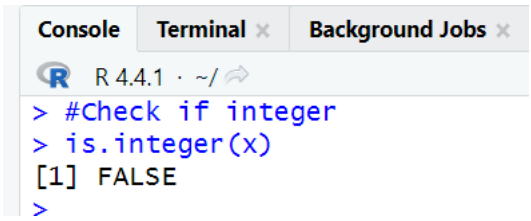


2. Integer

Source Code:

```
#Check if integer

is.integer(x)
```

Output:

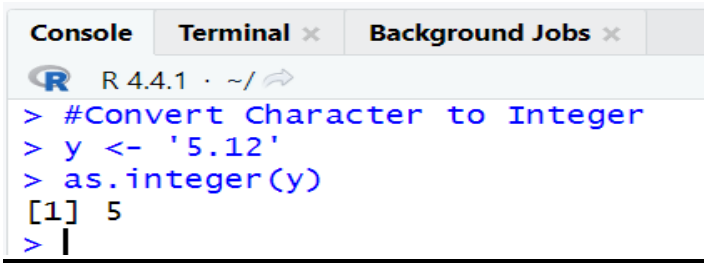


9. Convert Character to Integer:

Source Code:

```
y <- '5.12'
as.integer(y)
```

Output:



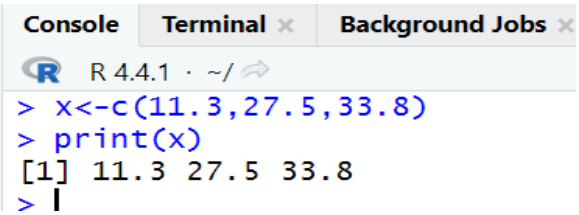
10. Create Vector:

1. Using c() function

Source Code:

```
x <- c(11.3,27.5,33.8)
print(x)
```

Output:

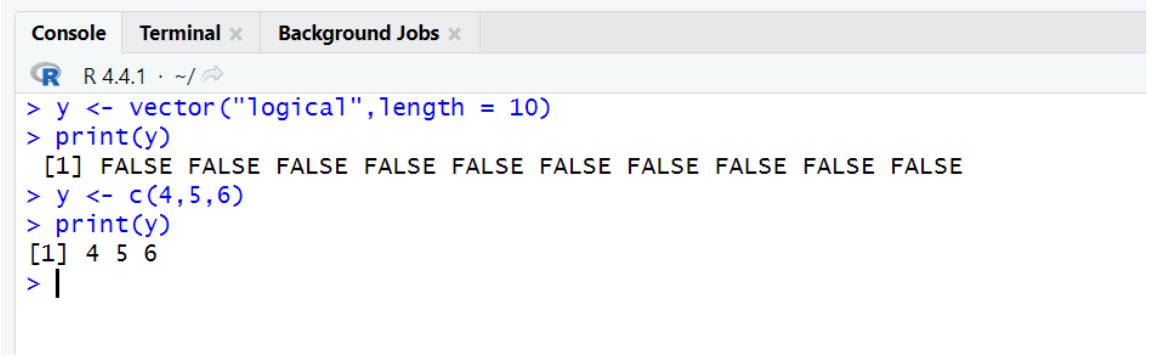


2. Using vector() function

Source Code:

```
y <- vector("logical",length=10)
print(y)
y <- c(4,5,6)
print(y)
```

Output:

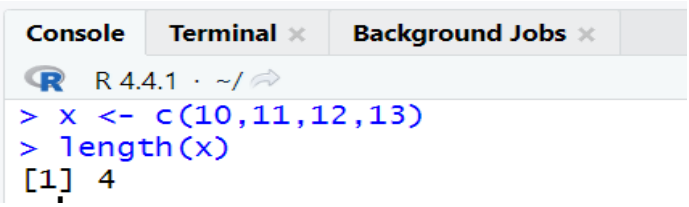


11. Find Length of Vector:

Source Code:

```
x <- c(10,11,12,13)

length(x)
```

Output:



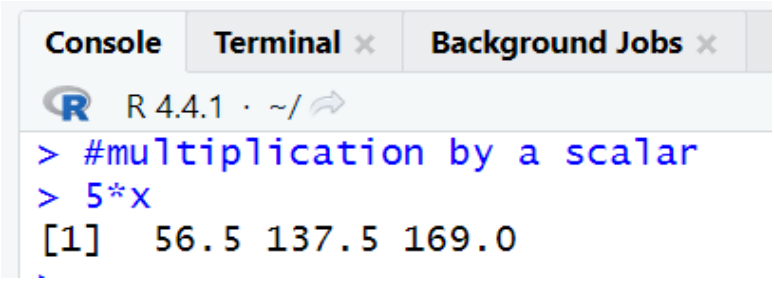
12. Arithmetic Operations on Vectors

1. Multiplication of Scalar

Source Code:

```
#multiplication by a scalar

5*x
```

Output:

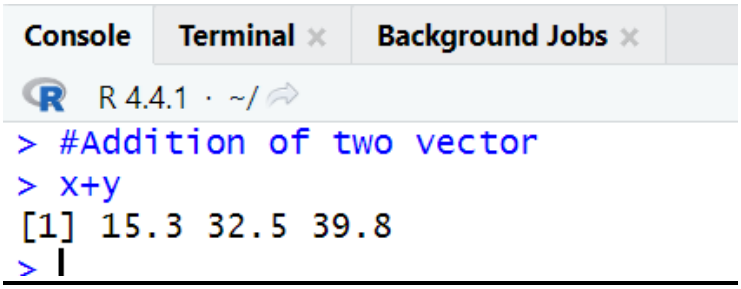


2. Addition of Vectors:

Source Code:

```
#Addition of two vectors

x+y
```

Output:

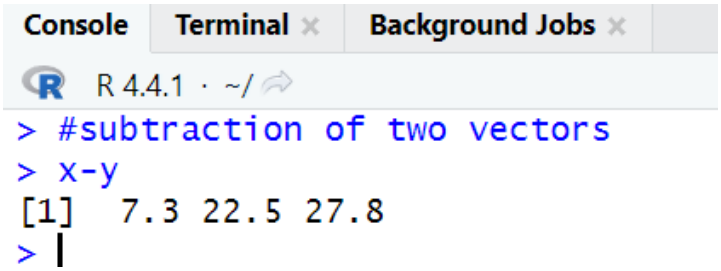


3. Subtraction of Vectors:

Source Code:

```
#Subtraction of two vectors

x-y
```

Output:

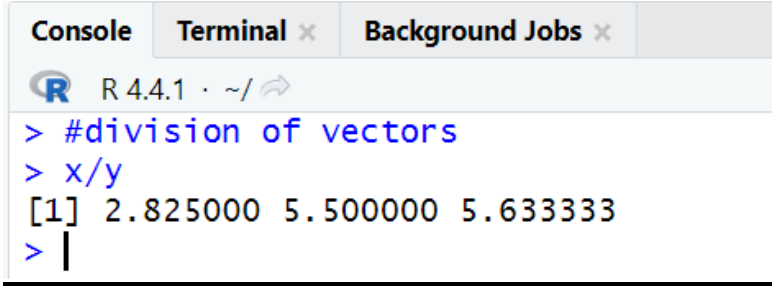


4. Division of Vectors:

Source Code:

```
#division of vectors

x/y
```

Output:



5. Power of Vectors:

Source Code:

```
#x to the power y
x^y
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · ~/
> #x to the power y
> x^y
[1] 1.630474e+04 1.572764e+07 1.491077e+09
>
```

13. Creation of Matrix:

Source Code:

```
###Creating Matrix: Two-dimensional array having elements of same class.
#using matrix() function
m<-matrix(c(11,12,13,55,60,65,66,72,78),nrow=3,ncol=3)
m
#By default, elements in matrix are filled by column. "byrow" attribute
of matrix() can be used to
fillm<-matrix(c(11,12,13,55,60,65,66,72,78),nrow=3,ncol=3,byrow = TRUE)
fillm
```

Output:

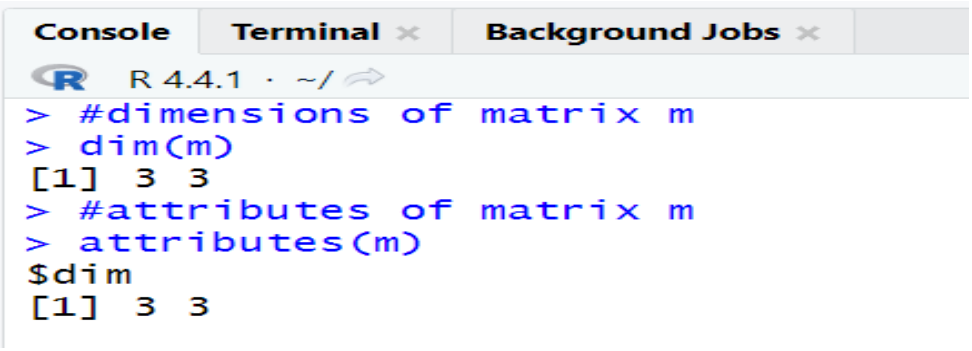
```
Console   Terminal   Background Jobs
R 4.4.1 · ~/
> ###Creating Matrix: Two-dimensional array having elements of same class.
> #using matrix() function
> m<-matrix(c(11,12,13,55,60,65,66,72,78),nrow=3,ncol=3)
> m
     [,1] [,2] [,3]
[1,]   11   55   66
[2,]   12   60   72
[3,]   13   65   78
> #By default, elements in matrix are filled by column. "byrow" attribute of matrix() can be used to
> fillm<-matrix(c(11,12,13,55,60,65,66,72,78),nrow=3,ncol=3,byrow = TRUE)
> fillm
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   55   60   65
[3,]   66   72   78
>
```

14. Find Dimensions & Attribute of a Matrix:

Source Code:

```
#dimensions of matrix m
dim(m)
#attributes of matrix m
attributes(m)
```

Output:

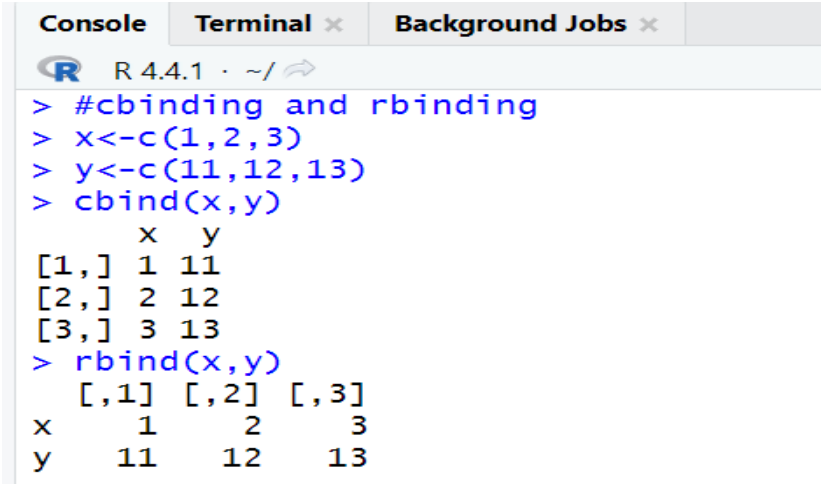


15. CBIND() & RBIND():

Source Code:

```
#cbinding and rbinding
x<-c(1,2,3)
y<-c(11,12,13)
cbind(x,y)
rbind(x,y)
```

Output:



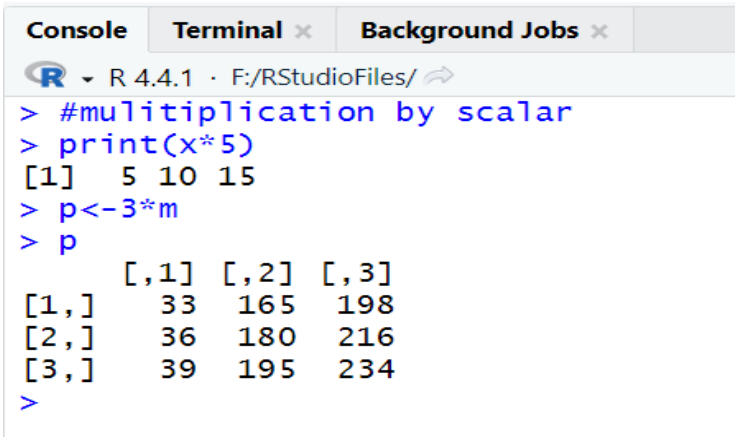
16. Operations on Matrix:

1. Multiplication by a scalar

Source Code:

```
#multiplication by a scalar

print(x*5)

p<-3*m

p
```

Output:

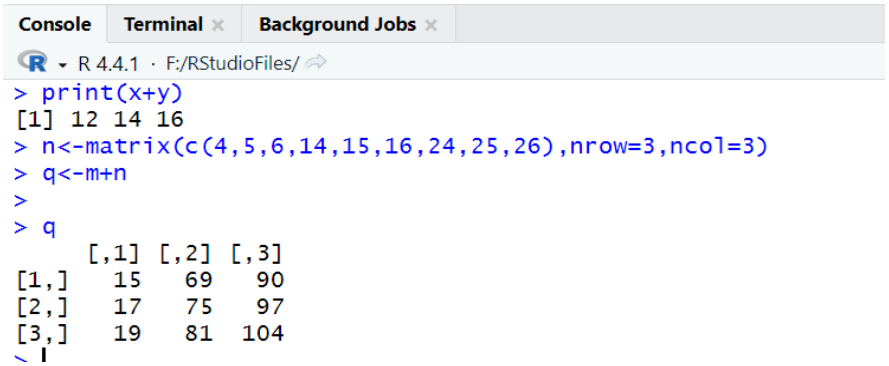


2. Addition of Matrices:

Source Code:

```
print(x+y)

n<-matrix(c(4,5,6,14,15,16,24,25,26),nrow=3,ncol=3)

q<-m+n

q
```

Output:

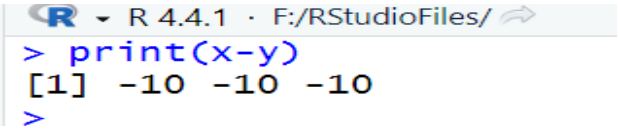


3. Subtraction of Matrices:

Source Code:

```
Print(x-y)
```

Output:

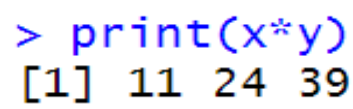


4. Multiplication of Matrices

Source Code:

```
Print(x*y)
```

Output:

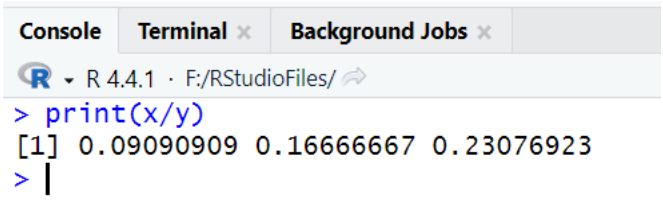


5. Division of Matrices

Source Code:

```
Print(x/y)
```

Output:

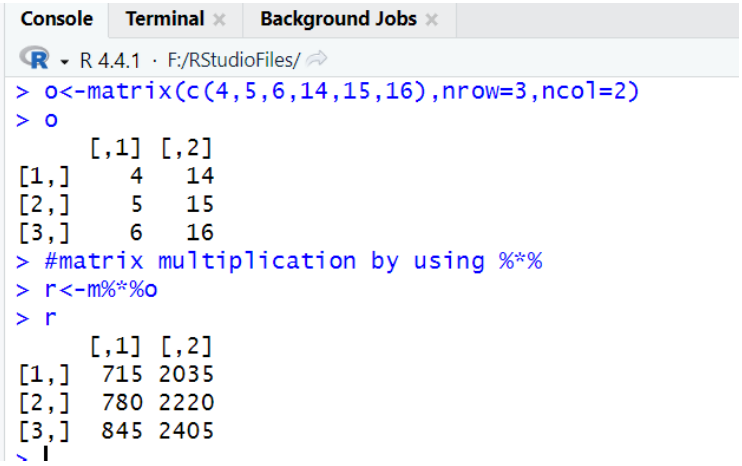


6. Matrix Multiplication by using %*%:

Source Code:

```
o<-matrix(c(4,5,6,14,15,16), nrow=3,ncol=2)

o

#matrix multiplication by using %*%

r<-m%*%o

r
```

Output:

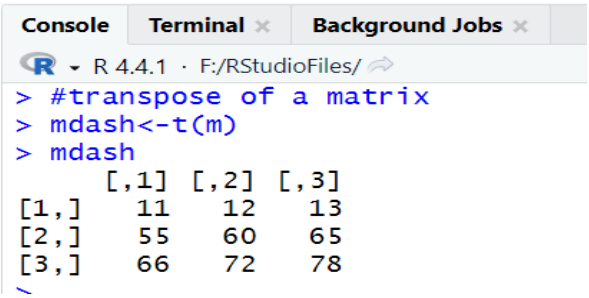


7. Transpose of Matrix

Source Code:

```
#transpose of a matrix

mdash <- t(m)

mdash
```

Output:

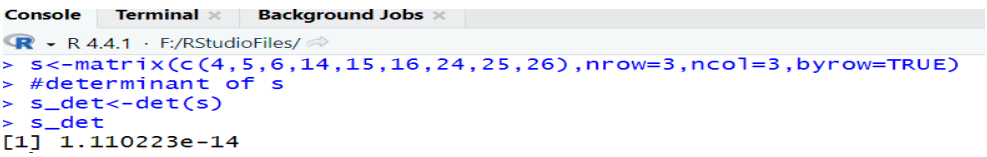


8. Find Determinant for Matrix:

Source Code:

```
s<-matrix(c(4,5,6,14,15,16,24,25,26),nrow=3,ncol=3,byrow=TRUE)

#determinant of s

s_det<-det(s)

s_det
```

Output:

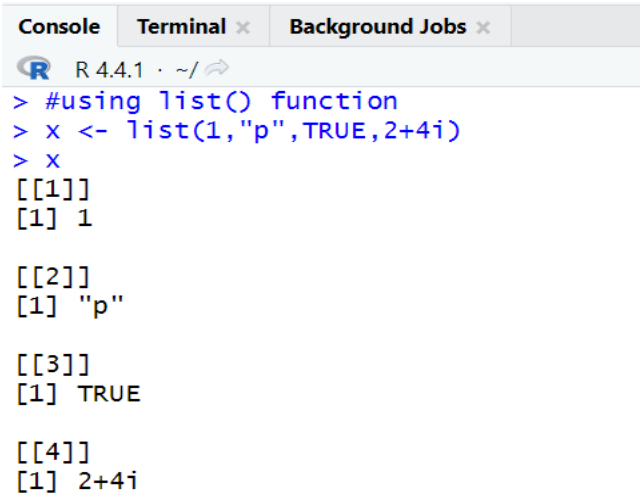


17. Example of List:

Source Code:

```
#using list() function

x<-list(1,”p”,TRUE,2+4i)

x
```

Output:

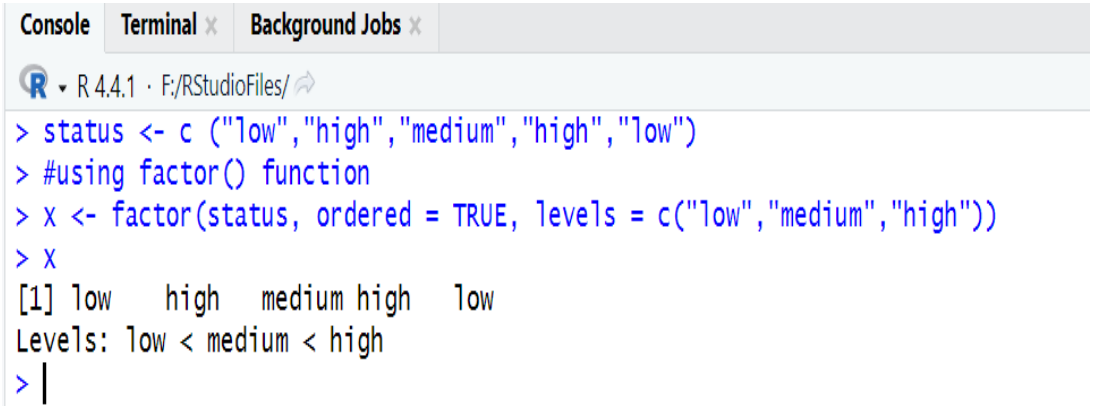


18. Example of Factor & Levels:

Source Code:

```
status <- c ("low","high","medium","high","low")

#using factor() function

x <- factor(status, ordered = TRUE, levels = c("low","medium","high"))

x
```

Output:



19. Example of Data Frame:

Source Code:

```
student_id<-c(1,2,3)
student_names<-c("Ram","Shyam","Laxman")
position<-c("First","Second","Third")
#using data.frame() function
data<-data.frame(student_id,student_names,position)
data
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> student_id<-c(1,2,3)
> student_names<-c("Ram","Shyam","Laxman")
> position<-c("First","Second","Third")
> #using data.frame() function
> data<-data.frame(student_id,student_names,position)
> data
  student_id student_names position
1          1           Ram    First
2          2         Shyam   Second
3          3        Laxman    Third
>
```

---

20. Functions of Data frame:

1. Accessing a Particular Column:

Source Code:

```
#accessing a particular column
data$student_id
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> #accessing a particular column
> data$student_id
[1] 1 2 3
>
```

2. Number of Rows in Data frame:

Source Code:

```
#no. of rows in data
nrow(data)
```

Output:

```
> #no. of rows in data
> nrow(data)
[1] 3
>
```

3. Number of Columns in Data frame:

Source Code:

```
#no. of columns in data

ncol(data)
```

Output:

```
> #no. of columns in data
> ncol(data)
[1] 3
```

4. Get Column Names of a Data frame:

Source Code:

```
#column names of data. For a dataframe, colnames() can also be used.

names(data)
```

Output:

```
> #column names of data. For a dataframe, colnames() can also be used.
> names(data)
[1] "student_id"    "student_names" "position"
>
```

21. Create a Two-Dimensional Table:

Source Code:

```
smoke <- matrix(c(51,43,22,92,28,21,68,22,9),ncol=3,byrow=TRUE)

colnames(smoke) <- c("High","Low","Middle")

rownames(smoke) <- c("current","former","never")

smoke <- as.table(smoke)

smoke
```

Output:

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/
> smoke <- matrix(c(51,43,22,92,28,21,68,22,9),ncol=3,byrow=TRUE)
> colnames(smoke) <- c("High","Low","Middle")
> rownames(smoke) <- c("current","former","never")
> smoke <- as.table(smoke)
> smoke
        High Low Middle
current   51  43     22
former    92  28     21
never     68  22      9
>
```

22. Install Packages:

Source Code:

```
install.packages("XLConnect")
library(XLConnect)
install.packages("readxl")
library(readxl)
install.packages("writexl")
library(writexl)
```

Output:

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/
> install.packages("XLConnect")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropri
ate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/FARHAN/AppData/Local/R/win-library/4.4'
(as 'lib' is unspecified)
also installing the dependency 'rJava'

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.4/rJava_1.0-11.zip'
Content type 'application/zip' length 835913 bytes (816 KB)
downloaded 816 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.4/XLConnect_1.1.0.zip'
Content type 'application/zip' length 23803250 bytes (22.7 MB)
downloaded 22.7 MB

package 'rJava' successfully unpacked and MD5 sums checked
package 'XLConnect' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\FARHAN\AppData\Local\Temp\RtmpklwEJh\downloaded_packages
> library(XLConnect)
XLConnect 1.1.0 by Mirai Solutions GmbH [aut],
  Martin Studer [cre],
  The Apache Software Foundation [ctb, cph] (Apache POI),
  Graph Builder [ctb, cph] (Curvesapi Java library),
  Brett Woolridge [ctb, cph] (SparseBitSet Java library)
https://mirai-solutions.ch
https://github.com/miraisolutions/xlconnect
>
```

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/
downloaded 86 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.4/readxl_1.4.3.zip'
Content type 'application/zip' length 1204095 bytes (1.1 MB)
downloaded 1.1 MB

package 'cli' successfully unpacked and MD5 sums checked
package 'glue' successfully unpacked and MD5 sums checked
package 'utf8' successfully unpacked and MD5 sums checked
package 'rematch' successfully unpacked and MD5 sums checked
package 'fansi' successfully unpacked and MD5 sums checked
package 'lifecycle' successfully unpacked and MD5 sums checked
package 'magrittr' successfully unpacked and MD5 sums checked
package 'pillar' successfully unpacked and MD5 sums checked
package 'pkgconfig' successfully unpacked and MD5 sums checked
package 'rlang' successfully unpacked and MD5 sums checked
package 'vctrs' successfully unpacked and MD5 sums checked
package 'crayon' successfully unpacked and MD5 sums checked
package 'hms' successfully unpacked and MD5 sums checked
package 'prettyunits' successfully unpacked and MD5 sums checked
package 'R6' successfully unpacked and MD5 sums checked
package 'cellranger' successfully unpacked and MD5 sums checked
package 'tibble' successfully unpacked and MD5 sums checked
package 'cpp11' successfully unpacked and MD5 sums checked
package 'progress' successfully unpacked and MD5 sums checked
package 'readxl' successfully unpacked and MD5 sums checked

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

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/
> install.packages("writexl")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropr
ate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/FARHAN/AppData/Local/R/win-library/4.4'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.4/writexl_1.5.0.zip'
Content type 'application/zip' length 203850 bytes (199 KB)
downloaded 199 KB

package 'writexl' successfully unpacked and MD5 sums checked

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

23. Examples of CSV:

1. Reading Data from CSV.

Source Code:

```
dataT <- read.table("input.csv", sep =",", header = T)

dataT
```

Output:

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/
> dataT <- read.table("input.csv", sep =",", header = T)
> dataT
  id          name  hp
1  1  Toyota Camry 203
2  2  Honda Accord 192
3  3  Ford Mustang 450
4  4       Chemra   230
5  5 Nissan Altima 188
```

2. Get Dimensions of CSV File:

Source Code:

```
#dimension

Dim(dataT)
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> #dimension
> dim(dataT)
[1] 5 3
```

3. Head & Tail:

Source Code:

```
#Load just few lines at the top or bottom

head(dataT,2)

tail(dataT,2)
```

Output:

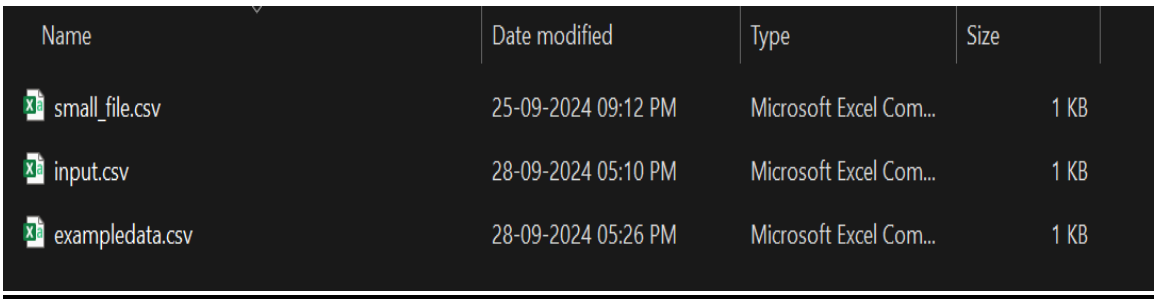
```
> #Load just few lines at the top or bottom
> head(dataT,2)
  id         name  hp
1  1 Toyota Camry 203
2  2 Honda Accord 192
> tail(dataT,2)
  id          name  hp
4  4      Chemra   230
5  5 Nissan Altima 188
>
```

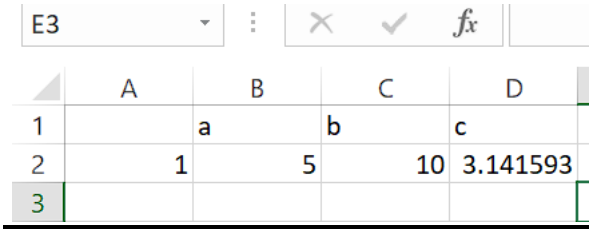
4. Writing Data to CSV File:

Source Code:

```
> z<-data.frame(a=5,b=10,c=pi)

> write.csv(z,file = "exampledata.csv")
```

Output:





	A	B	C	D
1		a	b	c
2	1	5	10	3.141593
3				

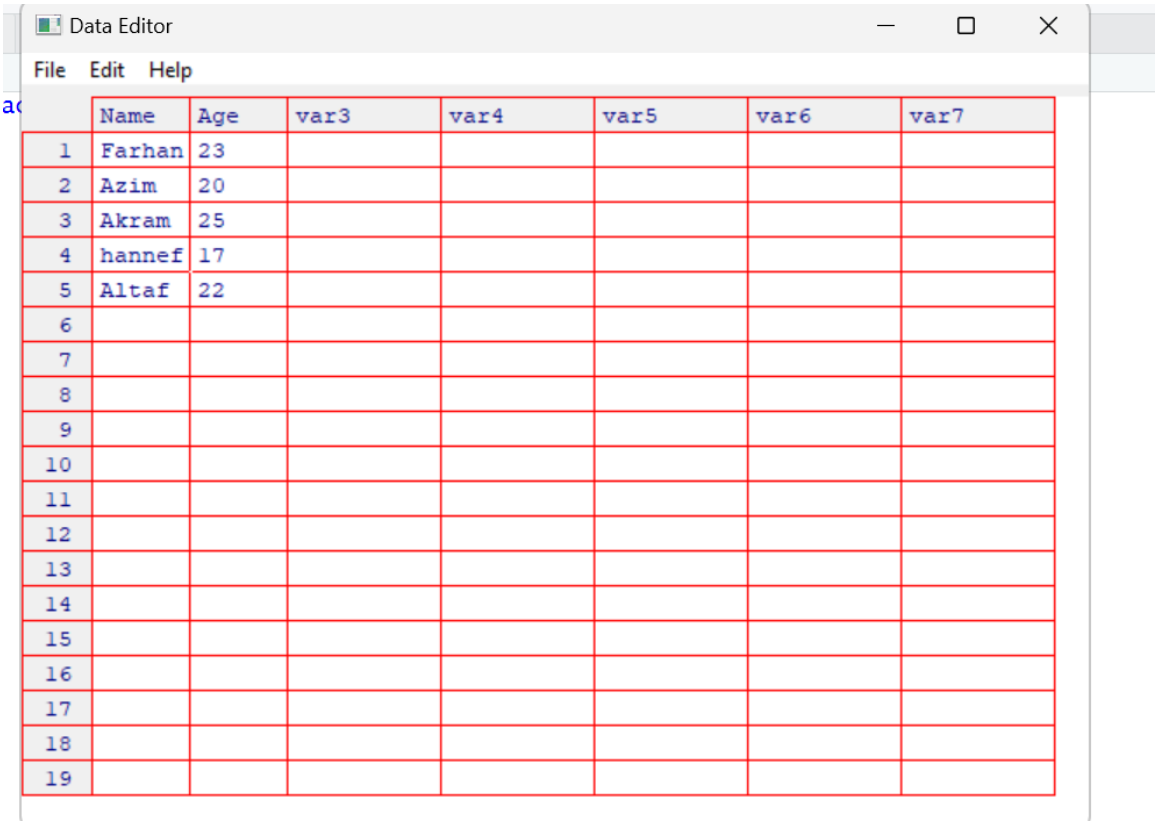
24. Reading and Writing Data from Excel using XL Connect:

Source Code:

```
install.packages('Rcpp')

library(Rcpp)

dataX <- XLConnect::
readWorksheetFromFile("Employee_Info.xlsx",sheet=1)

dataX

#Following is called Subsetting - It will print rows from 1 to 2 and all columns

dataY<- dataX[1:2,]

dataY

data <- data.frame(Name=character(), Age=numeric())

data <- edit(data)

print(data)
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> dataX <- XLConnect:: readWorksheetFromFile("Employee_Info.xlsx",sheet=1)
> dataX
  id    name department salary
1  1  Farhan         IT  60000
2  2 Zeeshan         HR  45000
3  3   Samad  Marketing  38000
4  4  Rizwan      Sales  50000
5  5  Anurag    Finance  43000
6  6  Tushar  Marketing  29000
> # Following is called Subsetting - It will print rows from 1 to 2 and all columns
> dataY<- dataX[1:2,]
> dataY
  id    name department salary
1  1  Farhan         IT  60000
2  2 Zeeshan         HR  45000
>
```



	Name	Age	var3	var4	var5	var6	var7
1	Farhan	23					
2	Azim	20					
3	Akram	25					
4	hannef	17					
5	Altaf	22					
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							

Console Terminal Background Jobs

R 4.4.1 · F:/RStudioFiles/

```
> data <- data.frame(Name=character(), Age=numeric())
> data <- edit(data)
>
> print(data)
    Name Age
1 Farhan  23
2   Azim  20
3  Akram  25
4 hannef  17
5  Altaf  22
```

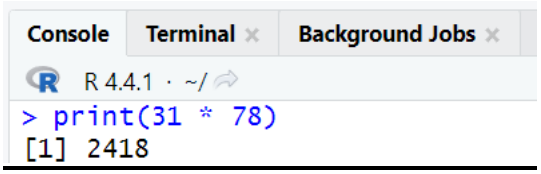
Assignment Questions

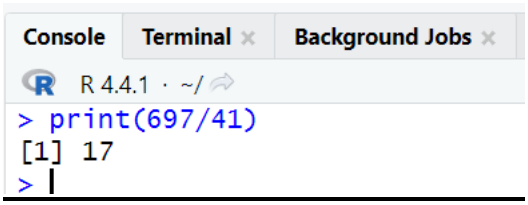
1. Use R to Calculate:
   - 31 * 78
   - 697/41

Source Code:

```
print(31 * 78)

print(697 / 41)
```

Output:



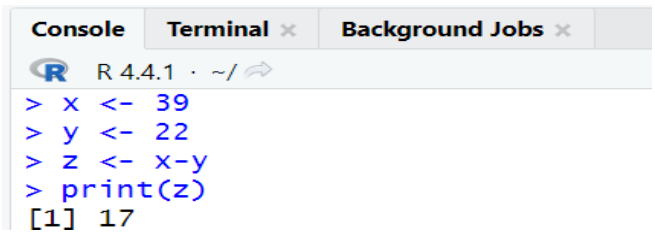


2. **Assign the value of 39 to x**
   - Assign the value off 22 to y
   - Make z the value of x-y
   - Display z in the console

Source Code:

```
x <- 39

y <- 22

z <- x-y

print(z)
```

Output:

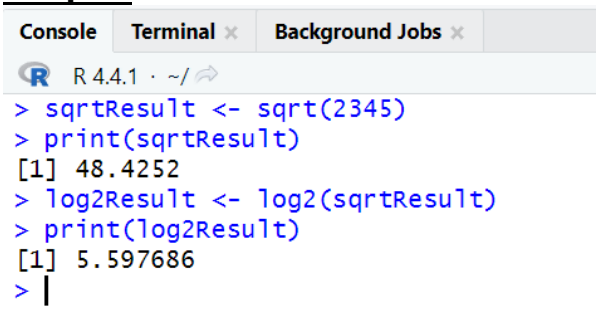


3. **Calculate the square root of 2345 and perform a log 2 transformation on the result.**

Source Code:

```
sqrtResult <- sqrt(2345)
print(sqrtResult)
log2Result <- log2(sqrtResult)
print(log2Result)
```

Output:

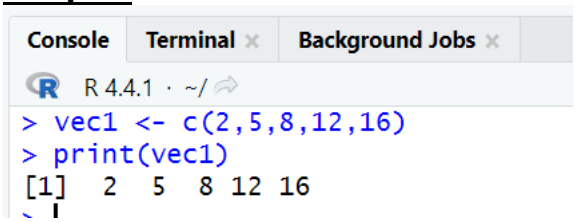


4. **Create a vector called vec1 containing the number 2 5 8 12 16.**

Source Code:

```
vec1 <- c(2,5,8,12,16)
print(vec1)
```

Output:

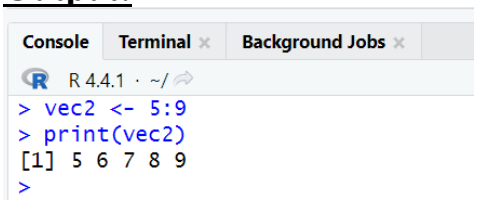


5. **Use x:y notation to make a second vector called vec2 containing the numbers 5 to 9.**

Source Code:

```
vec2 <- 5:9
print(vec2)
```

Output:



6. **Subtract vec2 from vec1 and look at the result.**

Source Code:

```
print(vec2 - vec1)
```

Output:

```
> #Subtracting Vectors
> print(vec2 - vec1)
[1]  3  1 -1 -4 -7
```

7. **Use seq() to make a vector of 100 values starting at 2 and increasing by 3 each time.**

Source Code:

```
sequenceVector <- seq(from=2,by=3,length.out=100)
print(sequenceVector)
```

Output:

```
> sequenceVector <- seq(from=2,by=3,length.out=100)
> print(sequenceVector)
  [1]   2   5   8  11  14  17  20  23  26  29  32  35  38  41  44  47  50  53  56  59  62  65  68  71  74  77  80
 [28]  83  86  89  92  95  98 101 104 107 110 113 116 119 122 125 128 131 134 137 140 143 146 149 152 155 158 161
 [55] 164 167 170 173 176 179 182 185 188 191 194 197 200 203 206 209 212 215 218 221 224 227 230 233 236 239 242
 [82] 245 248 251 254 257 260 263 266 269 272 275 278 281 284 287 290 293 296 299
```

8. **Extract the values at positions 5,10,15 and 20 in the vector of values you just created.**

Source Code:

```
sequenceVector[c(5,10,15,20)]
```

Output:

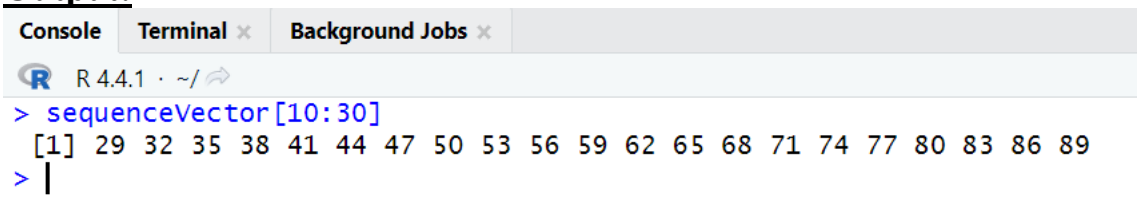
```
> sequenceVector[c(5,10,15,20)]
[1] 14 29 44 59
```

9. **Extract the values at the positions 10 to 30.**

Source Code:

```
sequenceVector[10:30]
```

Output:

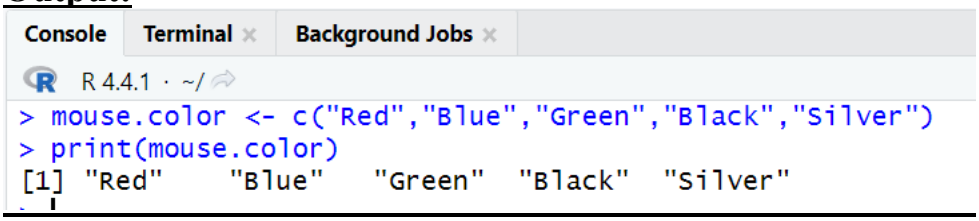


10. Enter a list of colors into a vector called mouse.color

Source Code:

```
mouse.color <- c("Red","Blue","Green","Black","Silver")
print(mouse.color)
```

Output:

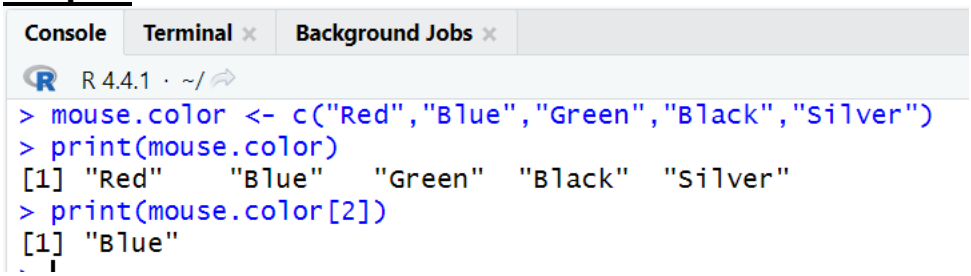


11. Display the second element in the vector.

Source Code:

```
print(mouse.color[2])
```

Output:

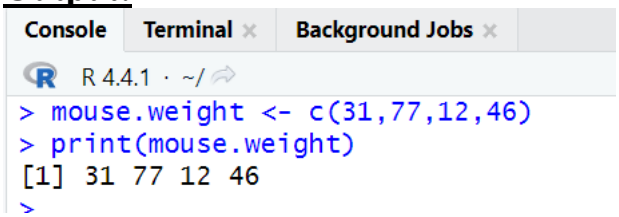


12. Enter some numerical weight data (supplied in the question) into a vector called mouse.weight.

Source Code:

```
mouse.weight <- c(31,77,12,46)
print(mouse.weight)
```

Output:

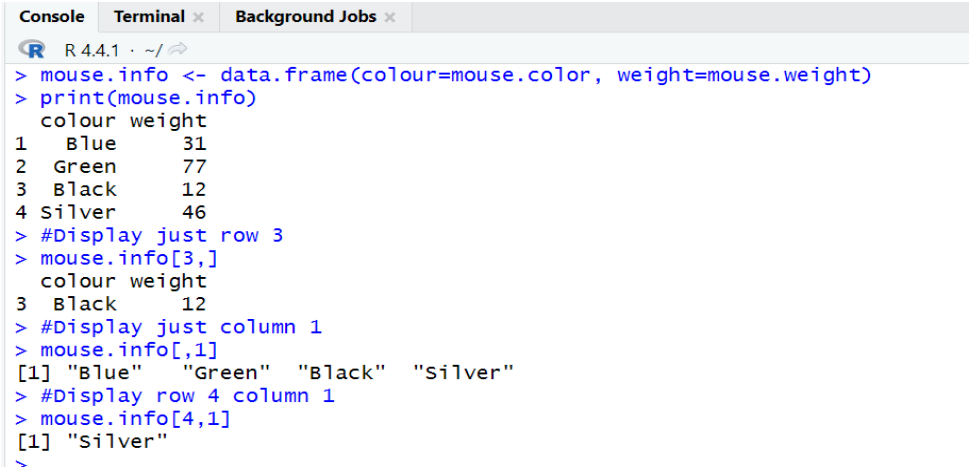


13. **Join the two vectors into a data frame called mouse.info containing 2 columns and 4 rows. Call the first column colour and the second one weight.**
    - Display just row 3
    - Display just column 1
    - Display row 4 column 1

Source Code:

```
mouse.info <- data.frame(colour=mouse.color, weight=mouse.weight)

print(mouse.info)

#Display just row 3

mouse.info[3,]

Display just column 1

mouse.info[,1]

Display row 4 column 1

mouse.info[4,1]
```

Output:

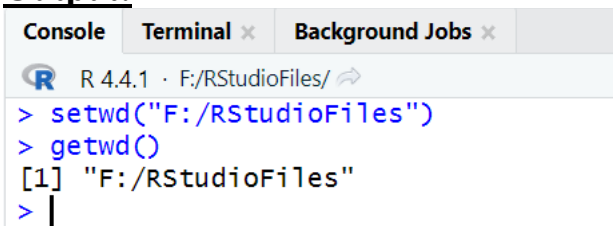


14. Set your working directory to where your data is stored

Source Code:

```
setwd("F:/RStudioFiles")
getwd()
```

Output:



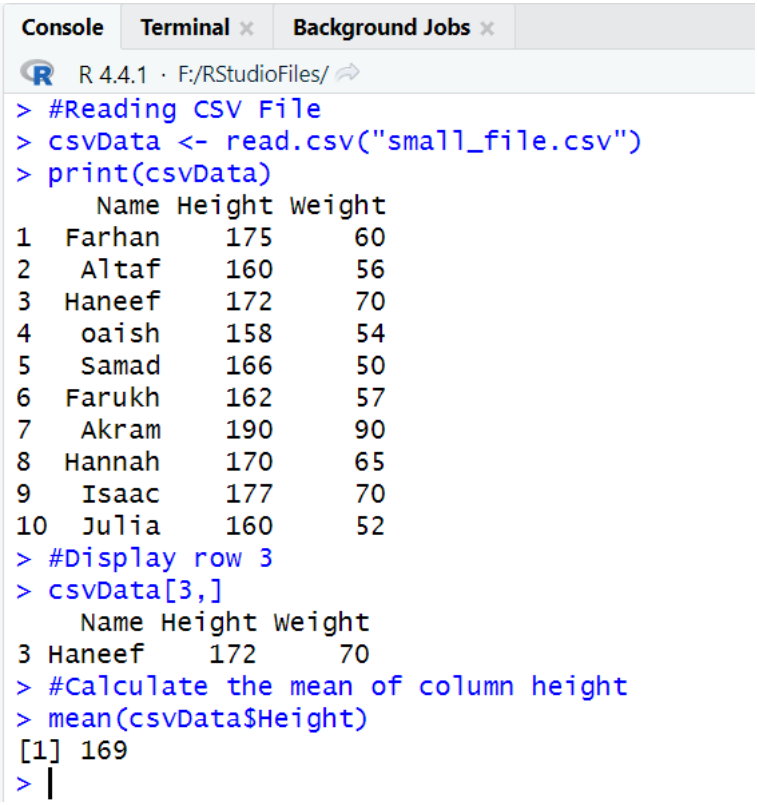
15. Read the file "small_file.csv" into a new structure. (create this csv with random data with attributes Name (character), Height (numeric in cm), weight (in kg)

- Display row 3
- Calculate the mean of column named height

Source Code:

```
#Reading CSV File

csvData <- read.csv("small_file.csv")

print(csvData)

#Display row 3

csvData[3,]

#Calculate the mean of column height

mean(csvData$Height)
```

Output:



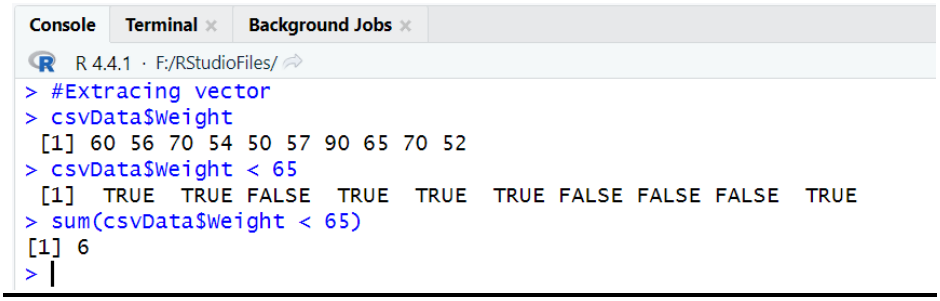
16. Extract the vector of values you want to filter against (in this case small_file$weight)

- Apply the logical test (in this case < 65)
- Either use the logical vector produced to filter the data using a selction, or use sum() to get the count of the hits.

Source Code:

```
#Extracing vector

csvData$Weight

csvData$Weight < 65

sum(csvData$Weight < 65)
```

Output:

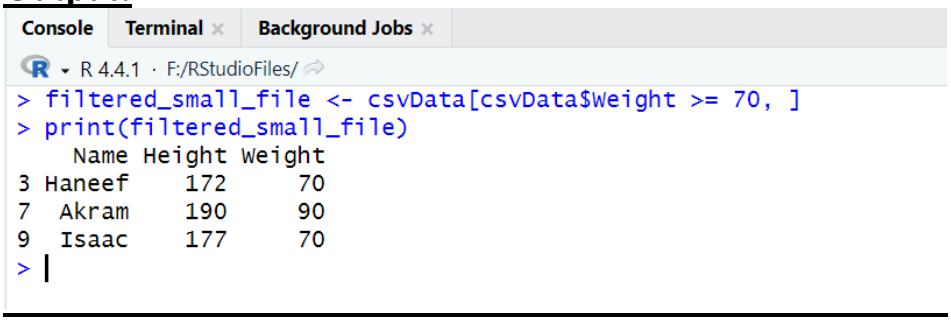


17. Create a filtered version of the small_file.csv variants data which includes only rows where weight >= 70

Source Code:

```
filtered_small_file <- csvData[csvData$Weight >= 70, ]
print(filtered_small_file
```

Output:



18. Given two matrices A and B, where:

- A is 2x3 matrix with the values 1,2,3,4,5 and 6.
- B is 2x3 matrix with the values 7,8,9,10,11 and 12.
- Combine these matrices column wise using the cbind() function.
- Combine these matrices row wise using the rbind() function.

Source Code:

```
#Creating two matrix

A <- matrix(c(1,2,3,4,5,6), nrow = 2,ncol = 3)

B <- matrix(c(7,8,9,10,11,12), nrow = 2,ncol = 3)

print(A)

print(B)
```

#cbind() function

c_bind_matrix <- cbind(A,B)

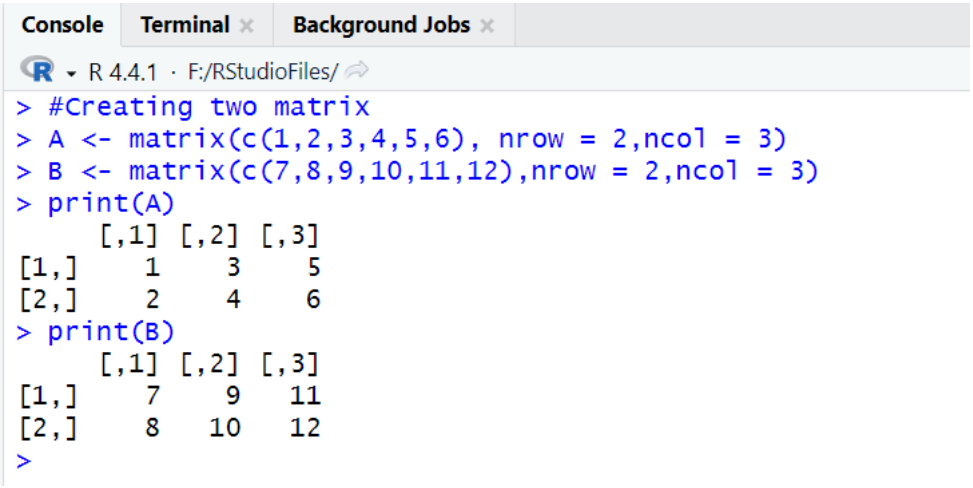
print(c_bind_matrix)

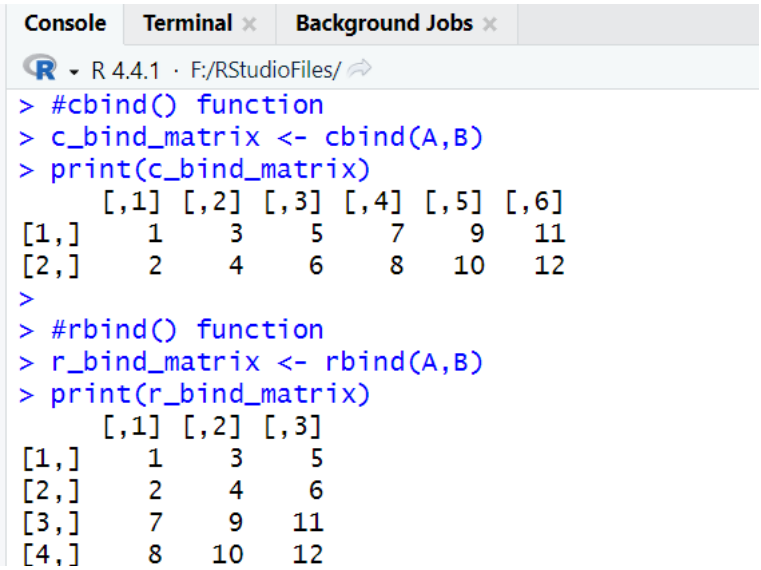
#rbind() function

r_bind_matrix <- rbind(A,B)

print(r_bind_matrix)

Output:





19. Consider two data frames df1 and df2:

- df1 contains the columns Name and Score, with two rows: (Alice, 85) and (Bob, 90).
- df2 also contains the columns Name and Score, with two rows: (Charlie, 88) and (David, 95).
- Combine these two data frames row-wise using the rbind() function.
- Create a new data frame by adding a column Age (values: 23, 22) to df1 using cbind().

Source Code:

```
#Creating two data frames
df1 <- data.frame(Name = c("Alice","Bob"),Score = c(85,90))
print(df1)
df2 <- data.frame(Name = c("Charlie","David"),Score = c(88,95))
print(df2)

#Combine two data frame
r_bind_df <- rbind(df1,df2)
print(r_bind_df)

#Age dataframe
 Age <- c(23,22)
df3 <- cbind(df1,Age)
print(df3)
```

Output:

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/
> #Creating two data frames
>     df1 <- data.frame(Name = c("Alice","Bob"),Score = c(85,90))
>     print(df1)
   Name Score
1 Alice    85
2   Bob    90
> df2 <- data.frame(Name = c("Charlie","David"),Score = c(88,95))
> print(df2)
     Name Score
1 Charlie    88
2   David    95
>
```

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/
> #Combine two data frame
> r_bind_df <- rbind(df1,df2)
> print(r_bind_df)
     Name Score
1   Alice    85
2     Bob    90
3 Charlie    88
4   David    95
> Age <- c(23,22)
> df3 <- cbind(df1,Age)
> print(df3)
   Name Score Age
1 Alice    85  23
2   Bob    90  22
>
```

20. Write the steps to:

- Read a CSV file named "students.csv" from your working directory into a data frame.
- Display the first six rows of this data frame.
- Read a tab-separated text file "data.txt" using the read.table() function.

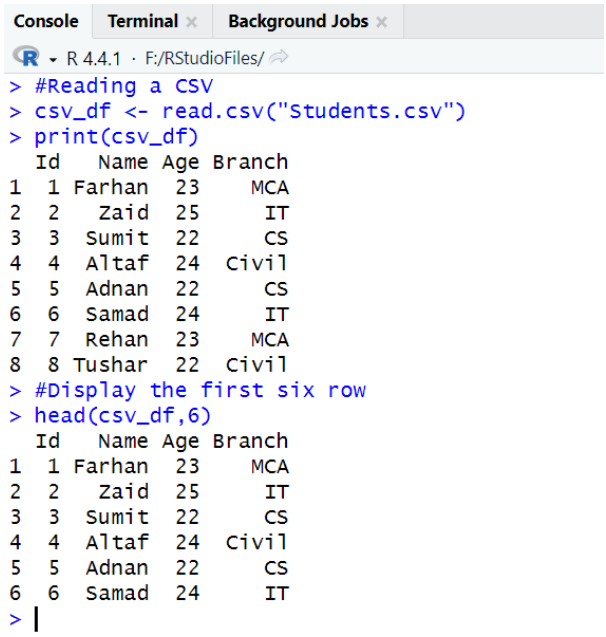
Source Code:

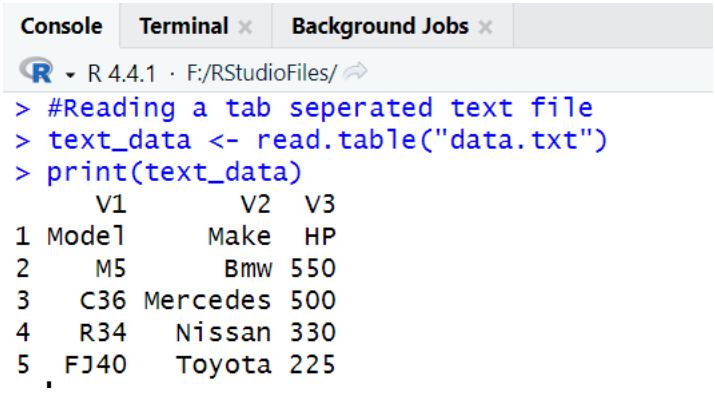
```
#Reading a CSV

csv_df <- read.csv("Students.csv")

print(csv_df)
```

```
#Display the first six row
 head(csv_df,6)
```

```
#Reading a tab seperated text file
 text_data <- read.table("data.txt")
 print(text_data)
```

Output:





21. You are asked to write data from a data frame to a CSV file:

Source Code:

```
write.csv(csv_df,"Farhan.csv")
```

Output:

1. Explain the significance of the row.names = FALSE argument when writing data to a CSV file.

Explanation:

When you set row.names = FALSE: This argument removes the row names when writing to CSV file, so that the actual data from the data frame is written to the file without including an extra column of row names.

Source Code:

```
#With the row names
write.csv(csv_df,"farhan_df.csv",row.names = TRUE)
read.csv("farhan_df.csv")
#Without the row names
write.csv(csv_df,"farhan_df.csv",row.names = FALSE)
read.csv("farhan_df.csv")
```

Output

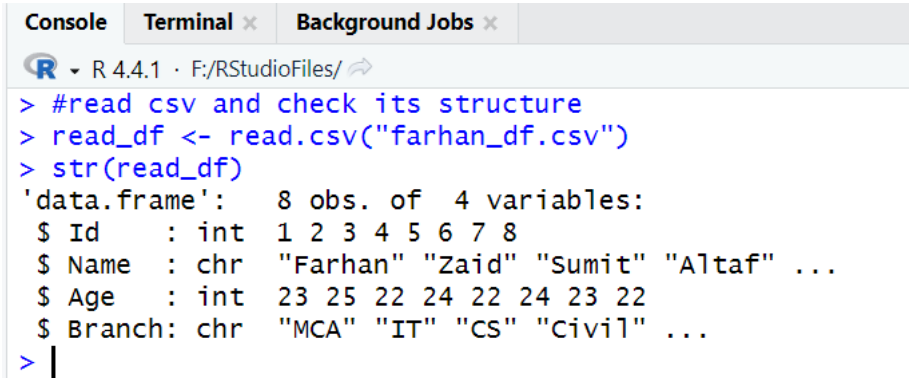
```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> #With the row names
> write.csv(csv_df,"farhan_df.csv",row.names = TRUE)
> read.csv("farhan_df.csv")
  X Id    Name Age Branch
1 1  1  Farhan  23    MCA
2 2  2    Zaid  25     IT
3 3  3   Sumit  22     CS
4 4  4   Altaf  24  Civil
5 5  5   Adnan  22     CS
6 6  6   Samad  24     IT
7 7  7   Rehan  23    MCA
8 8  8  Tushar  22  Civil
> #Without the row names
> write.csv(csv_df,"farhan_df.csv",row.names = FALSE)
> read.csv("farhan_df.csv")
  Id    Name Age Branch
1  1  Farhan  23    MCA
2  2    Zaid  25     IT
3  3   Sumit  22     CS
4  4   Altaf  24  Civil
5  5   Adnan  22     CS
6  6   Samad  24     IT
7  7   Rehan  23    MCA
8  8  Tushar  22  Civil
: >
```

2. After writing a CSV file, describe how to read the CSV file back into R and check its structure.

Source Code:

```
#read csv and check its structure
read_df <- read.csv("farhan_df.csv")
str(read_df)
```

Output:



22. You are provided a data frame containing StudentID, Name, and Marks of three students:

- Save this data frame to a file named "marks.csv".
- Save this data frame to a tab-delimited text file named "marks.txt".

Source Code:

```
provide_df <- data.frame(StudentID = c(1,2,3),
Name = c("Farhan","Zeeshan","Rizwan"),Marks = c(99,81,88))
#Writing to a csv file
write.csv(provide_df,"Marks.csv",row.names = FALSE)
#Writing to a tab delimited text file
write.table(provide_df,"Marks.txt",row.names = FALSE)
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> provide_df <- data.frame(StudentID = c(1,2,3),Name = c("Farhan","Zeeshan","Rizwan"),Marks = c(99,81,88))
> #Writing to a csv file
> write.csv(provide_df,"Marks.csv",row.names = FALSE)
> #Writing to a tab delimited text file
> write.table(provide_df,"Marks.txt",row.names = FALSE)
>
```

Marks.csv 29-09-2024 04:30 PM

	A	B	C
1	StudentID	Name	Marks
2	1	Farhan	99
3	2	Zeeshan	81
4	3	Rizwan	88
5			

Marks.txt 29-09-2024 04:31 PM

File Edit View

```
"StudentID" "Name" "Marks"
1 "Farhan" 99
2 "Zeeshan" 81
3 "Rizwan" 88
```

23. You are given a matrix results with 3 rows and 3 columns.

- Describe how you would save this matrix as a CSV file.
- Explain how you would save the matrix as an RData file and later load it back into R

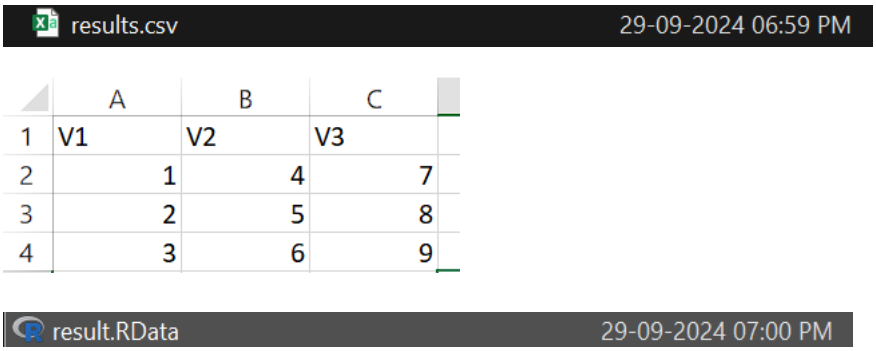
Explanation:

1. To save matrix as a CSV file in R, you have to first convert the matrix into a dataframe using **as.data.frame()** and then use the **write.csv()** to save it into the CSV file.
2. To save the matrix to an RData file using the **save()** function and the extension of the new file will be **.RData** and later load it back into R using **load()** function.

Source Code:

```
#Saving data into a csv file.
 results <- matrix(c(1,2,3,4,5,6,7,8,9),nrow = 3,ncol = 3)
 write.csv(as.data.frame(results),"results.csv",row.names = FALSE)
#Saving data into a RData File.
 save(results,file = "result.RData")
 load("result.RData")
```

Output:



	A	B	C
1	V1	V2	V3
2	1	4	7
3	2	5	8
4	3	6	9

24. You have two CSV files "part1.csv" and "part2.csv", which contain different sections of the same dataset.

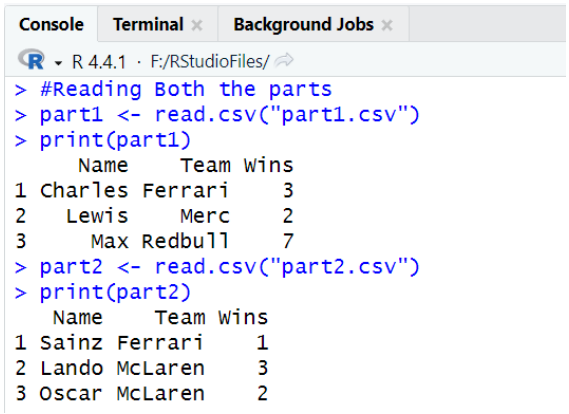
- Write the steps to read both files into R.
- Combine these datasets using rbind().
- Save the combined dataset as a new CSV file named "combined_data.csv".

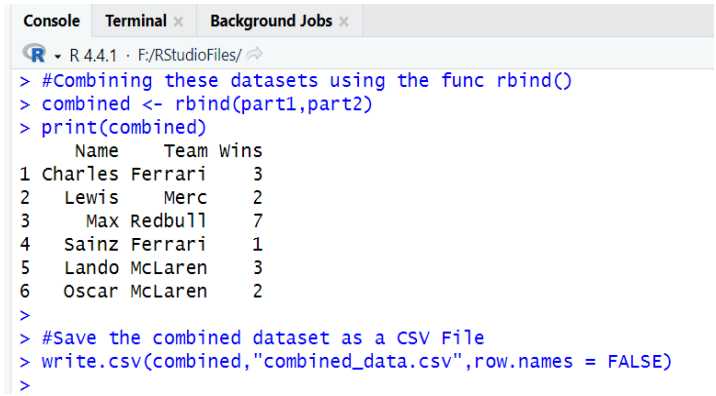
Source Code:

```
#Reading Both the parts
part1 <- read.csv("part1.csv")
print(part1)
part2 <- read.csv("part2.csv")
print(part2)
```

```
#Combining these datasets using the func rbind()
combined <- rbind(part1,part2)
print(combined)

#Save the combined dataset as a CSV File
write.csv(combined,"combined_data.csv",row.names = FALSE)
```

Output:





	A	B	C
1	Name	Team	Wins
2	Charles	Ferrari	3
3	Lewis	Merc	2
4	Max	Redbull	7
5	Sainz	Ferrari	1
6	Lando	McLaren	3
7	Oscar	McLaren	2

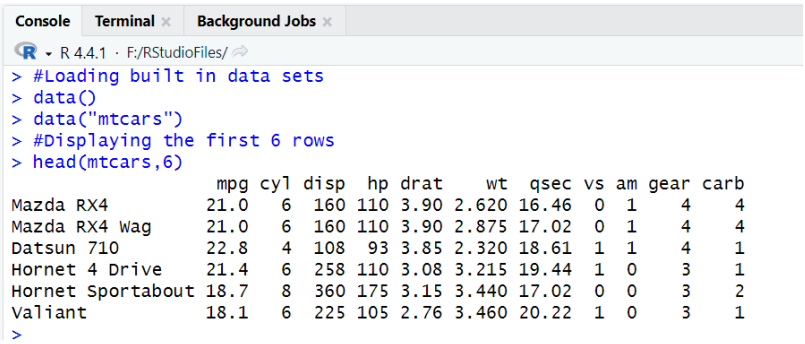
25. R has several built-in datasets. Using data(), list all the built-in datasets available in R.

- Choose one dataset (e.g., mtcars) and load it into your R session.
- Display the first 6 rows of the dataset.

Source Code:

```
#Loading built in data sets
data()
data("mtcars")
#Displaying the first 6 rows
head(mtcars,6)
```

Output:



```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> #Loading built in data sets
> data()
> data("mtcars")
> #Displaying the first 6 rows
> head(mtcars,6)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
>
```

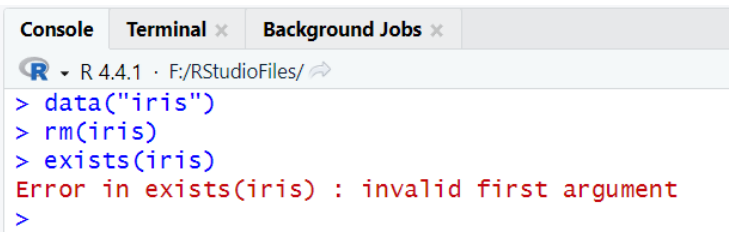
26. After loading a dataset (e.g., iris), you want to free up memory by removing objects that are no longer needed.

- Use rm() to remove the dataset iris from the environment.
- Verify that the dataset has been removed

Source Code:

```
data("iris")
rm("iris")
exists("iris")
```

Output:



```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/
> data("iris")
> rm(iris)
> exists(iris)
Error in exists(iris) : invalid first argument
>
```

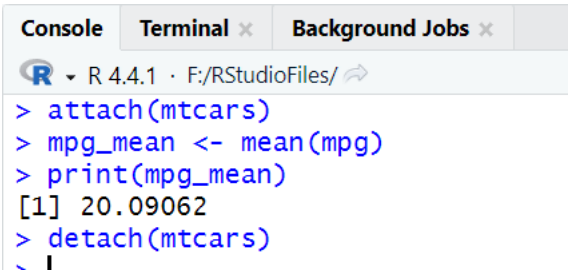
27. Consider the built-in dataset mtcars. Perform the following steps:

- Attach the mtcars dataset to the search path so that its variables can be referred to by their names directly.
- Use the variable mpg (Miles per Gallon) to calculate the mean without referring to the dataset.
- Detach the mtcars dataset after completing your calculations.

Source Code:

```
attach(mtcars)

mpg_mean <- mean(mpg)

print(mpg_mean)

detach(mtcars)
```

Output:



28. Explain the potential issues that might arise from attaching a dataset and working with it in R. Why is it generally recommended to avoid using attach() in large projects?

Explanation:

Using attach() in R can lead to several potential issues, especially in larger projects. This includes namespace conflicts, unintentional variable overwriting, ambiguity, performance and code readability issues.

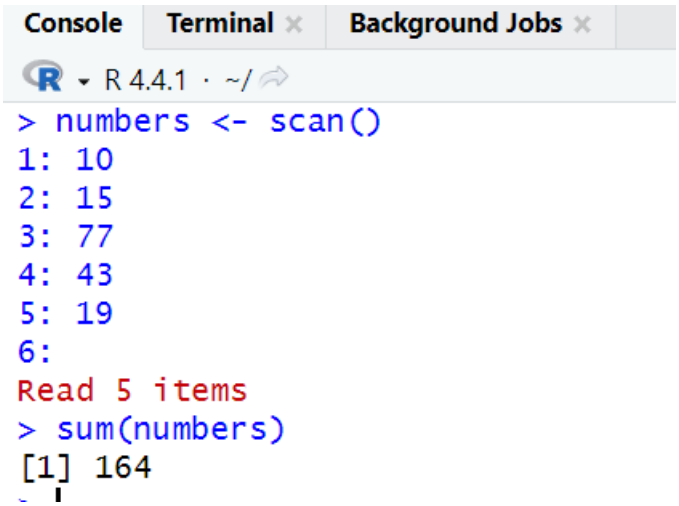
29. You are required to enter data manually from the console:

- Write the steps to create a small vector of numbers (e.g., 5 numbers) by reading input from the console using scan().
- After reading the data, calculate the sum of the numbers entered.

Source Code:

```
numbers <- scan()
10
15
77
43
19
sum(numbers)
```

Output:

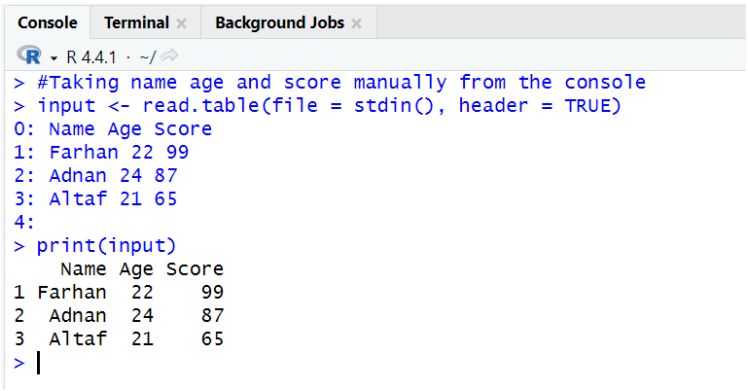


30. Describe the process of entering a small dataset directly into R as a data frame using the read.table() function with the option to input data manually from the console. Use an example with 3 columns: Name, Age, and Score.

Source Code:

```
#Taking name age and score manually from the console
input <- read.table(file = stdin(), header = TRUE)
0: Name Age Score
1: Farhan 22 99
2: Adnan 24 87
3: Altaf 21 65
print(input)
```

Output:



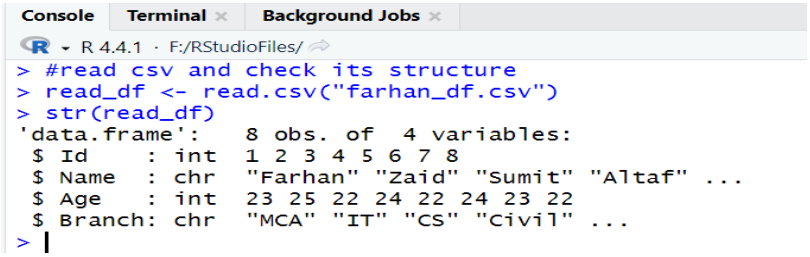
31. You are provided a CSV file "students.csv" in your working directory:

- Describe the steps to load this file into R using read.csv().
- After loading the data, show how you would display the structure of the data frame.

Source Code:

```
#read csv and check its structure
read_csv <- read.csv("farhan_df.csv")
str(read_csv)
```

Output:



32. You have been given an Excel file "grades.xlsx" containing student grades in multiple sheets. Write the steps to:

- Install and load the required package to read Excel files in R.
- Load data from the first sheet of the file "grades.xlsx".

Source Code:

```
#Loading data from excel

 library(readxl)

 grades <- read_excel("grades.xlsx")

 head(grades)
```

Output:

```
Console   Terminal ×   Background Jobs ×
R 4.4.1 · F:/RStudioFiles/
> #Loading data from excel
> library(readxl)
> grades <- read_excel("grades.xlsx")
> head(grades)
# A tibble: 6 × 3
  Name    Percentage Grade
  <chr>        <dbl> <chr>
1 Farhan          85 A
2 Altaf           50 C
3 Adnan           65 B
4 Zeeshan         76 B+
5 Sumit           95 A+
6 Haneef          34 F
```

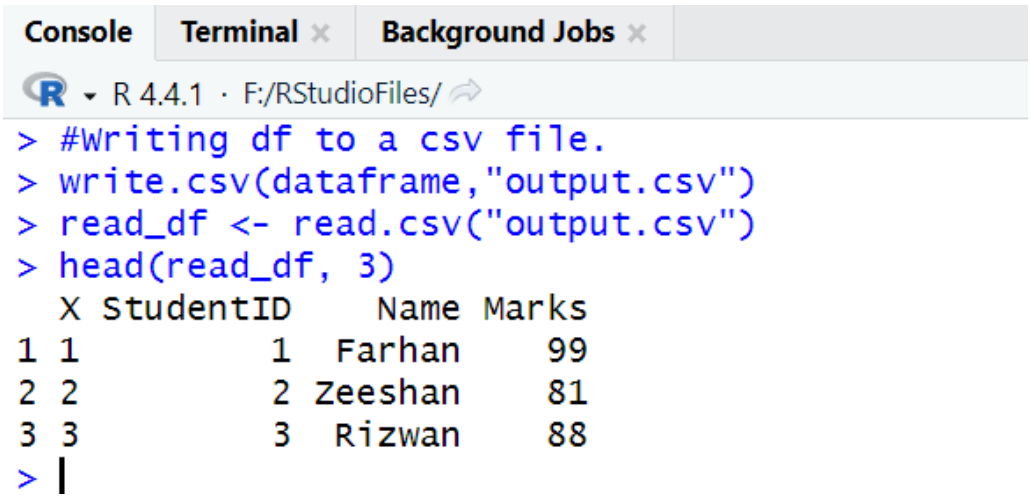
33. You are asked to export a data frame to a CSV file:

- Write the steps to save the data frame df to a CSV file named "output.csv".
- Explain how to read the CSV file back into R.

Source Code:

```
#Writing df to a csv file.
 write.csv(dataframe,"output.csv")
 read_df <- read.csv("output.csv")
 head(read_df, 3)
```

Output:



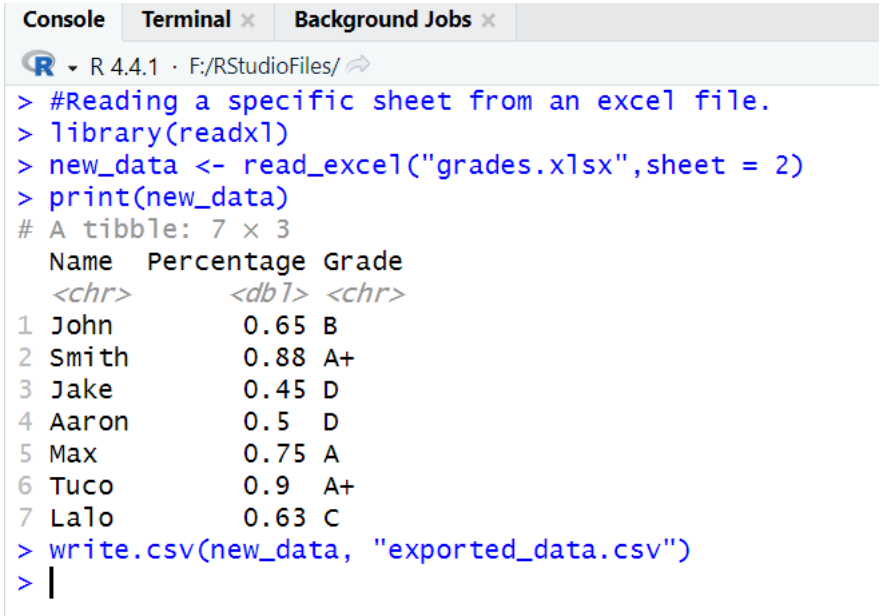
34. You have an Excel file that contains a large dataset with multiple sheets.

- Explain how to read a specific sheet (e.g., Sheet 2) from the file into R using the appropriate functions.
- Describe how you would export this data to a new CSV file named "exported_data.csv".

Source Code:

```
#Reading a specific sheet from an excel file.
 library(readxl)
 new_data <- read_excel("grades.xlsx",sheet = 2)
 print(new_data)
 write.csv(new_data , “exported_data.csv”)
```

Output:



CONCLUSION:

From this Practical, I have Learned the basics of R Programming.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 2**	
Title of LAB Assignment: Data Preprocessing in R.			
DOP: 27 – 09 – 2024		**DOS: 03 – 10 – 2024**	
CO Mapped:	**PO Mapped:**	**Faculty Signature:**	**Marks:**

Practical No: 2

AIM: Data Preprocessing in R.

Theory:

Data Preprocessing Technique:

1. **Data Cleaning:** The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.
   a. **Missing Data:** This situation arises when some data is missing in the data. It can be handled in various ways. Some of them are:
      i. **Ignore the tuples:** This approach is suitable only when the dataset we have is quite large and multiple values are missing within a tuple.
      ii. **Fill the Missing values:** There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.

   b. **Noisy Data:** Noisy data is a meaningless data that cannot be interpreted by machines. It can be generated due to faulty data collection, data entry errors etc. It can be handled in following ways:
      i. **Binning Method:** This method works on sorted data in order to smooth it. The whole data is divided into segments of equal size and then various methods are performed to complete the task. Each segmented is handled separately. One can replace all data in a segment by its mean or boundary values can be used to complete the task.
      ii. **Regression:** Here data can be made smooth by fitting it to a regression function. The regression used may be linear (having one independent variable) or multiple (having multiple independent variables).
      iii. **Clustering:** This approach groups the similar data in a cluster. The outliers may be undetected or it will fall outside the clusters.

2. **Data Transformation:** This step is taken in order to transform the data appropriate forms suitable for mining process. This involves following ways:

i. **Normalization: -** It is done in order to scale the data values in a specified range (-1.0 to 1.0 or 0.0 to 1.0)

ii. **Attribute Selection: -** In this strategy, new attributes are constructed from the given set of attributes to help the mining process.

iii. **Discretization: -** This is done to replace the raw values of numeric attribute by interval levels or conceptual levels.

iv. **Concept Hierarchy Generation: -** Here attributes are converted from level to higher level in hierarchy. For Example-The attribute “city” can be converted to “country”

1. **Print head of MTCARS Dataset (Predefined in R):**

Source Code:

```
my_data<-mtcars
head(my_data,5)
```

Output:

```
> my_data<-mtcars
> head(my_data,5)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
>
```

2. **Reading data of specific rows and columns:**

Source Code:

```
my_data1<-my_data[1:6,1:5]
my_data1
```

Output:

```
> my_data1<-my_data[1:6,1:5]
> my_data1
                   mpg cyl disp  hp drat
Mazda RX4         21.0   6  160 110 3.90
Mazda RX4 Wag     21.0   6  160 110 3.90
Datsun 710        22.8   4  108  93 3.85
Hornet 4 Drive    21.4   6  258 110 3.08
Hornet Sportabout 18.7   8  360 175 3.15
Valiant           18.1   6  225 105 2.76
>
```

3. **Rename column name using DPLYR**

Source Code:

```
install.packages("dplyr")
library(dplyr,warn.conflicts=FALSE)
```

```
my_data1=dplyr::rename(my_data1,"horse_power"="hp")
my_data1
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · ~/
> require(dplyr)
Loading required package: dplyr

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> my_data1 = rename(my_data1, horse_power = hp)
> my_data1
                   mpg cyl disp horse_power drat
Mazda RX4         21.0   6  160         110 3.90
Mazda RX4 Wag     21.0   6  160         110 3.90
Datsun 710        22.8   4  108          93 3.85
Hornet 4 Drive    21.4   6  258         110 3.08
Hornet Sportabout 18.7   8  360         175 3.15
Valiant           18.1   6  225         105 2.76
>
```

4. **Adding new column:**

Source Code:

```
my_data1$new_hp1<-my_data1$horse_power*0.5
colnames(my_data1)
my_data1
```

Output:

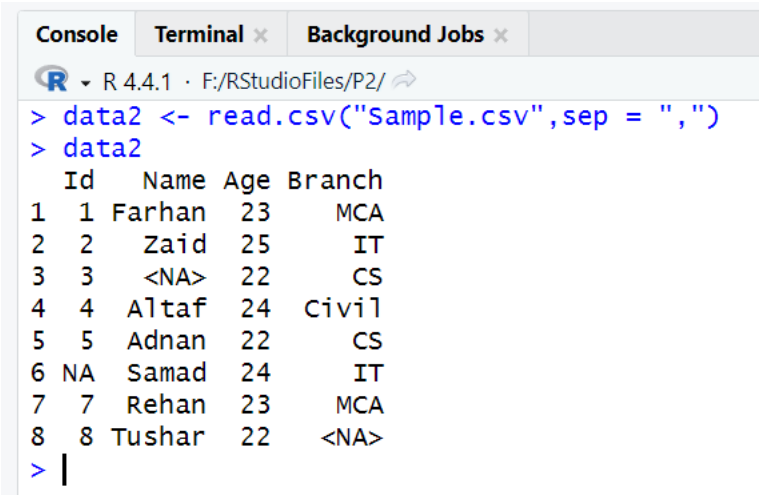
```
Console   Terminal   Background Jobs
R 4.4.1 · ~/
> my_data1$new_hp1 <- my_data1$horse_power * 0.5
> colnames(my_data1)
[1] "mpg"        "cyl"        "disp"       "horse_power" "drat"       "new_hp1"
> my_data1
                   mpg cyl disp horse_power drat new_hp1
Mazda RX4         21.0   6  160         110 3.90    55.0
Mazda RX4 Wag     21.0   6  160         110 3.90    55.0
Datsun 710        22.8   4  108          93 3.85    46.5
Hornet 4 Drive    21.4   6  258         110 3.08    55.0
Hornet Sportabout 18.7   8  360         175 3.15    87.5
Valiant           18.1   6  225         105 2.76    52.5
>
```

5. Reading data from CSV file:

Source Code:

```
data2 <- read.csv("Sample.csv", sep = ",")
data2
```

Output:

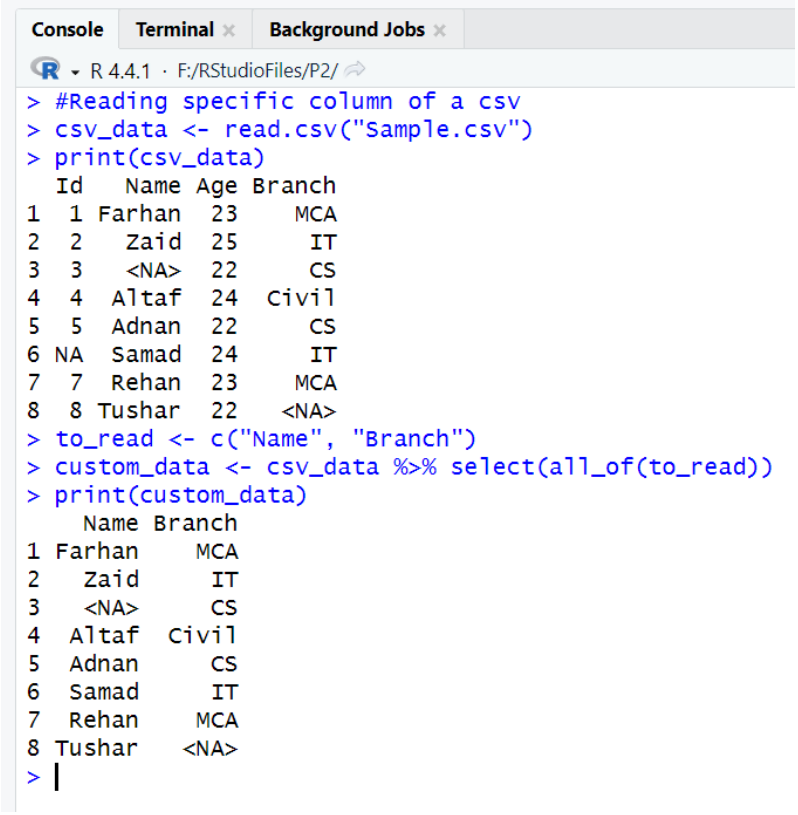


6. Reading data from specific columns of CSV file:

Source Code:

```
#Reading specific column of a csv

csv_data <- read.csv("Sample.csv")

print(csv_data)


to_read <- c("Name", "Branch")

custom_data <- csv_data %>% select(all_of(to_read))

print(custom_data)
```

Output:

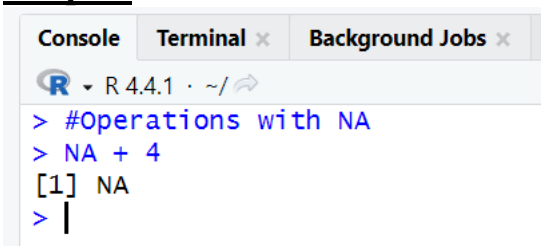


7. Operation with NA:

Source Code:

```
NA+4
```

Output:

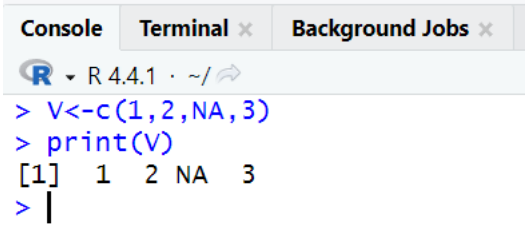


8. Create a vector V with NA values:

Source Code:

```
V<-c (1, 2, NA, 3)
V
```

Output:

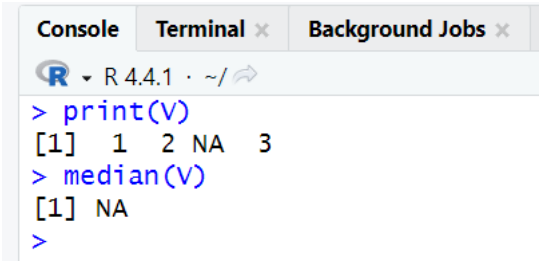


9. **Find median:**
   i. **With NA:**

Source Code:

```
median(V)
```

Output:

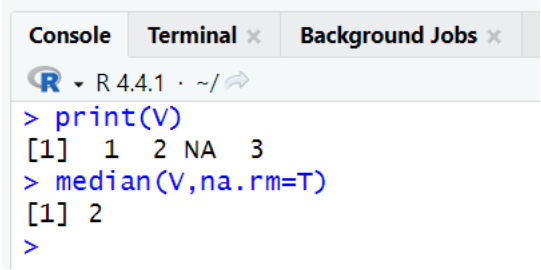


   ii. **Without NA:**

Source Code:

```
median(V, na.rm=T)
```

Output:

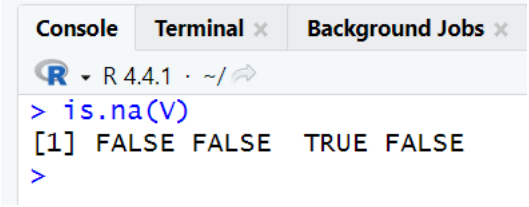


10. **Check whether it is NA or not:**

Source Code:

```
is.na(V)
```

Output:

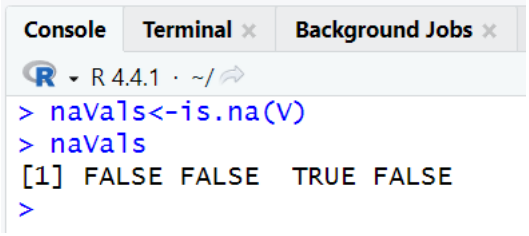


11. Removing the NA values by using logical indexing:

Source Code:

```
naVals<-is.na(V)
naVals
```

Output:

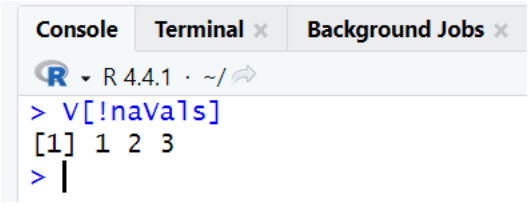


12. Get the values that are not NA:

Source Code:

```
V[!naVals]
```

Output:

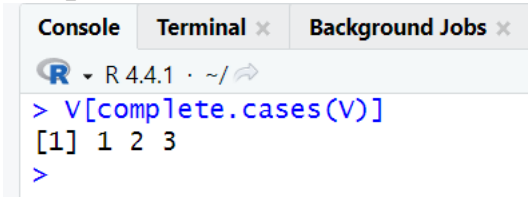


13. Sub-setting with complete cases- values that are not NA:

Source Code:

```
V[complete.cases(V)]
```

Output:

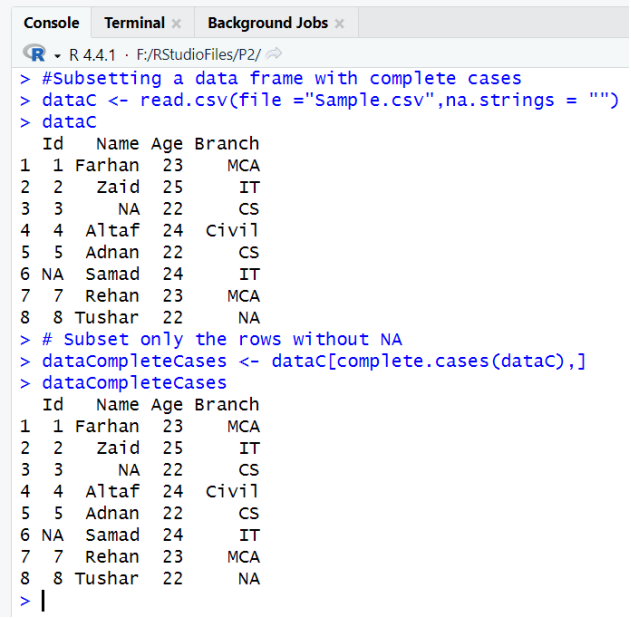


14. Sub-setting a data frame with complete cases:

Source Code:

```
#Subsetting a data frame with complete cases
dataC <- read.csv(file ="Sample.csv",na.strings = "")
dataC
# Subset only the rows without NA
dataCompleteCases <- dataC[complete.cases(dataC),]
dataCompleteCases
```

Output:

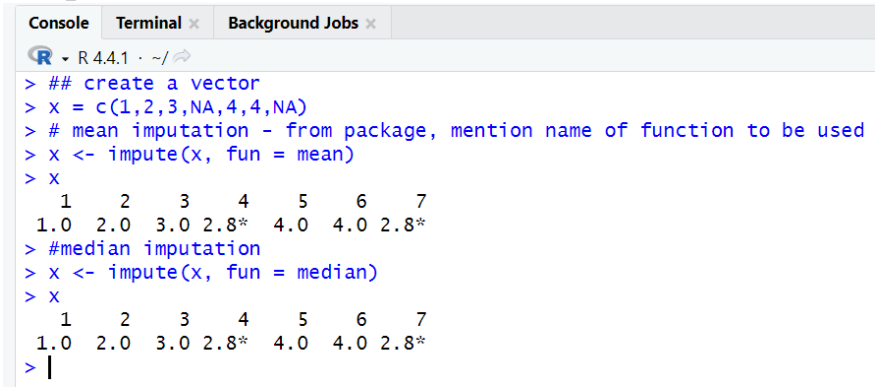


15. Mean imputation and Median imputation:

Source Code:

```
library(Hmisc)
## create a vector
x = c(1,2,3,NA,4,4,NA)
# mean imputation - from package, mention name of function to be used
x <- impute(x, fun = mean)
x
#median imputation
x <- impute(x, fun = median)
x
```

Output:

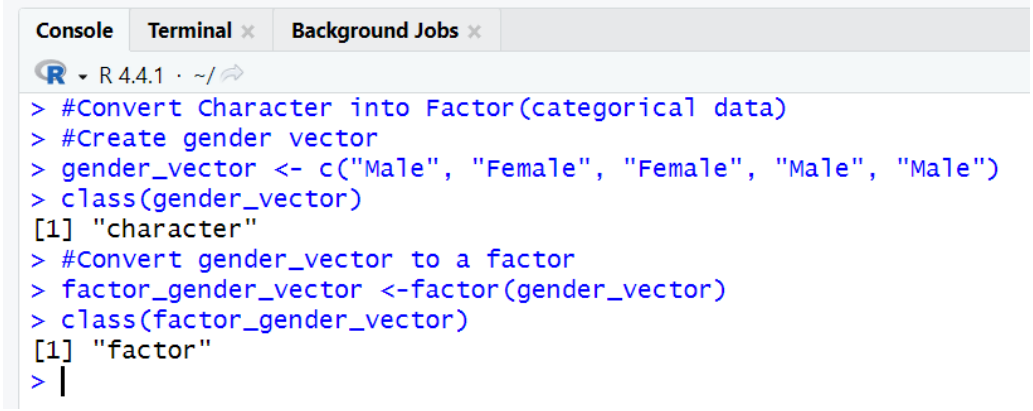


16. Convert Character into Factor (categorical data):

Source Code:

```
#Convert Character into Factor(categorical data)
#Create gender vector
gender_vector <- c("Male", "Female", "Female", "Male", "Male")
class(gender_vector)
#Convert gender_vector to a factor
factor_gender_vector <-factor(gender_vector)
class(factor_gender_vector)
```

Output:

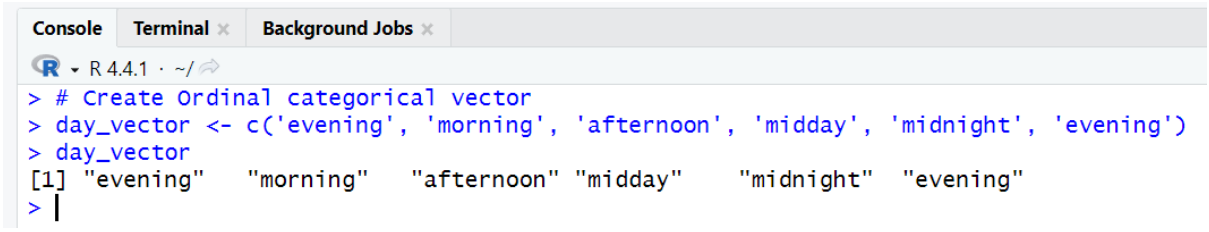


17. Create ordinal categorical vector:

Source Code:

```
#Create ordinal categorical vector
day_vector<-
c('evening','morning','afternoon','midday','midnight','evening')
day_vector
```

Output:

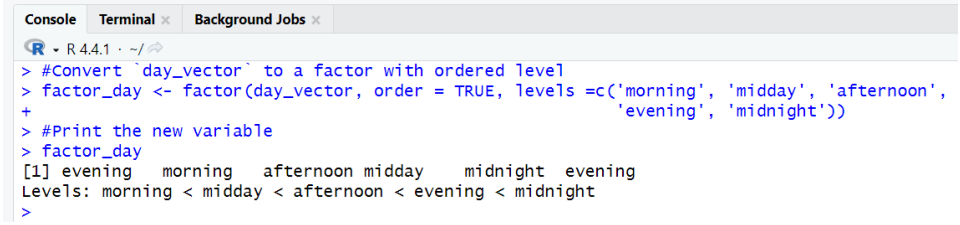


18.Convert vector into a factor with ordered level:

Source Code:

```
#Convert `day_vector` to a factor with ordered level
factor_day <- factor(day_vector, order = TRUE, levels =c('morning',
'midday', 'afternoon',
'evening', 'midnight'))
#Print the new variable
factor_day
```

Output:

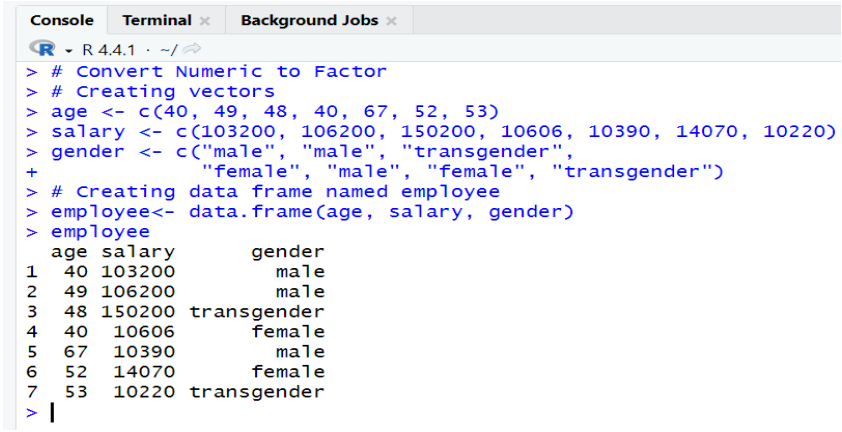


19. Create Data frame from vector:

Source Code:

```
# Creating vectors
age <- c(40, 49, 48, 40, 67, 52, 53)
salary <- c(103200, 106200, 150200, 10606, 10390, 14070, 10220)
gender <- c("male", "male", "transgender","female", "male", "female",
"transgender")
# Creating data frame named employee
employee<- data.frame(age, salary, gender)
employee
```

Output:

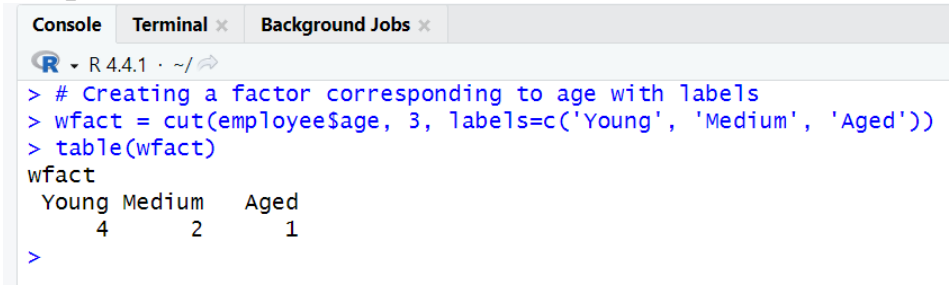


20. Create factor with labels:

Source Code:

```
# Creating a factor corresponding to age with labels
wfact = cut(employee$age, 3, labels=c('Young', 'Medium', 'Aged'))
table(wfact)
```

Output:



21. Naming and Renaming Variables

Question 1.

You are given a data frame employee_df with the following columns: emp_id, emp_name, age, and salary. The company decides to rename the column emp_name to name and emp_id to ID.

- Write the steps to rename these columns.
- Verify the new names of the columns in the data frame.

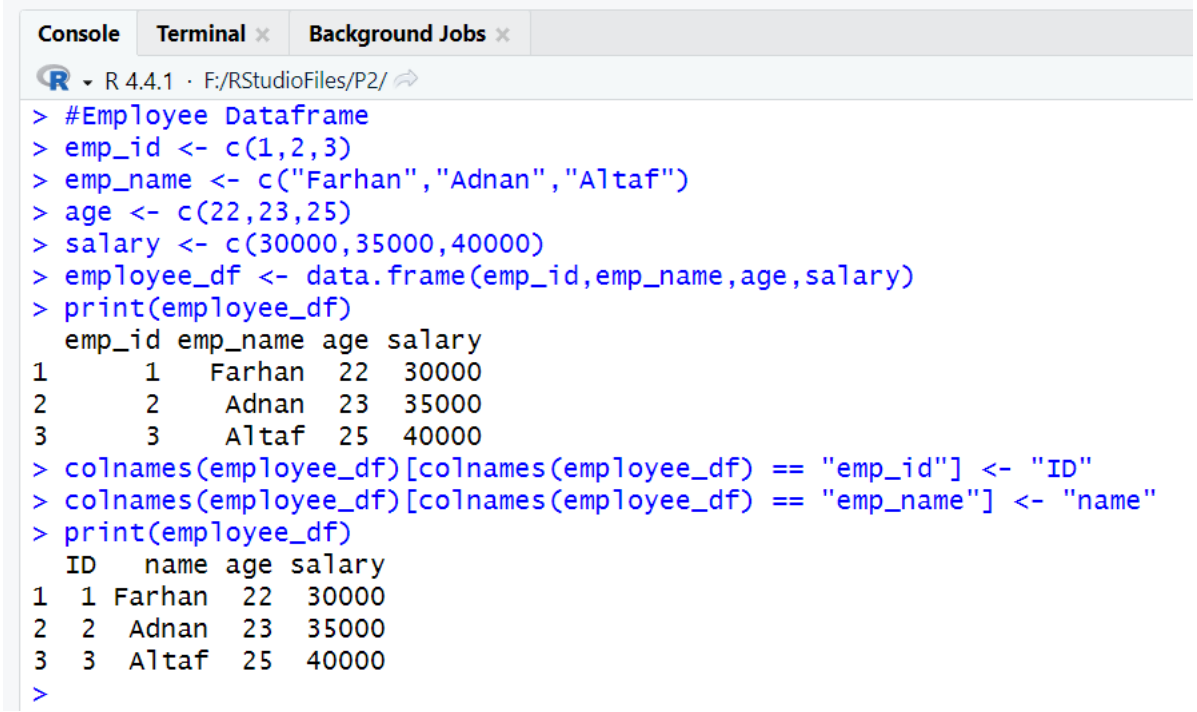
Explanation:

colnames() method in R is used to rename and replace the column names in the dataframe.

Source Code:

```
#Employee Dataframe
emp_id <- c(1,2,3)
emp_name <- c("Farhan","Adnan","Altaf")
age <- c(22,23,25)
salary <- c(30000,35000,40000)
employee_df <- data.frame(emp_id,emp_name,age,salary)
print(employee_df)

colnames(employee_df)[colnames(employee_df) == "emp_id"] <- "ID"
colnames(employee_df)[colnames(employee_df) == "emp_name"] <-
"name"
print(employee_df)
```

Output:



Question 2.

Consider a data frame student_df with the columns StudentID, Name and Marks.

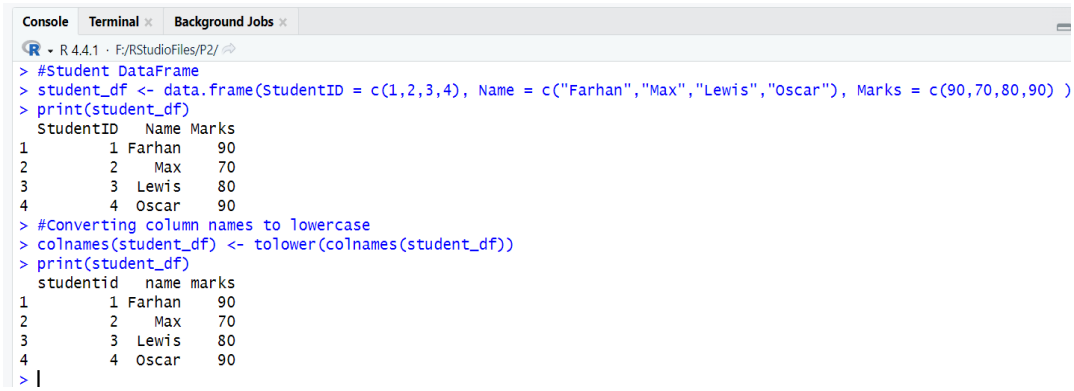
- Write the steps to rename all the columns to lowercase (studentid, name, and marks).
- Add a new column named Grade that assigns an "A" grade for students with marks greater than 90, and "B" for others.

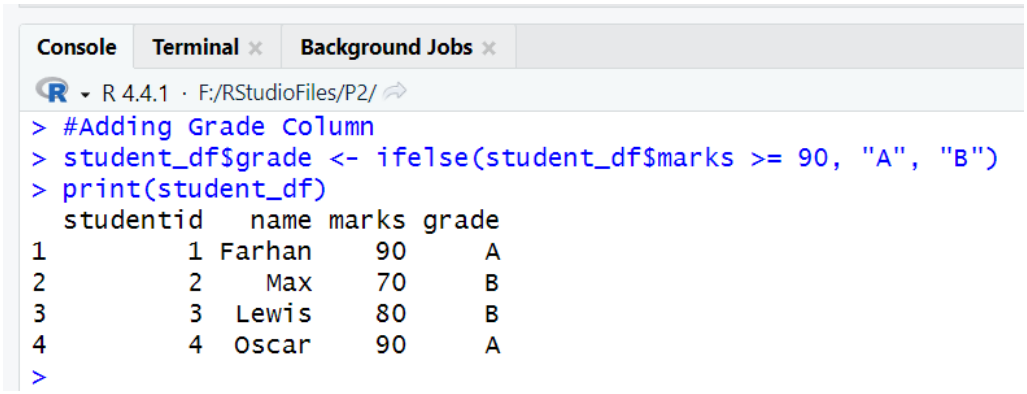
Source Code:

```
#Student DataFrame

student_df <- data.frame(

 StudentID = c(1,2,3,4),

Name = c("Farhan","Max","Lewis","Oscar"),

Marks = c(90 ,70,80,90) )

print(student_df)
```

```
#Converting column names to lowercase
 colnames(student_df) <- tolower(colnames(student_df))
 print(student_df)
#Adding Grade Column
 student_df$grade <- ifelse(student_df$marks >= 90, "A", "B")
 print(student_df)
```

Output:





22. Adding a New Variable

Question 3.

You are analyzing the mtcars dataset and want to create a new variable performance, where:

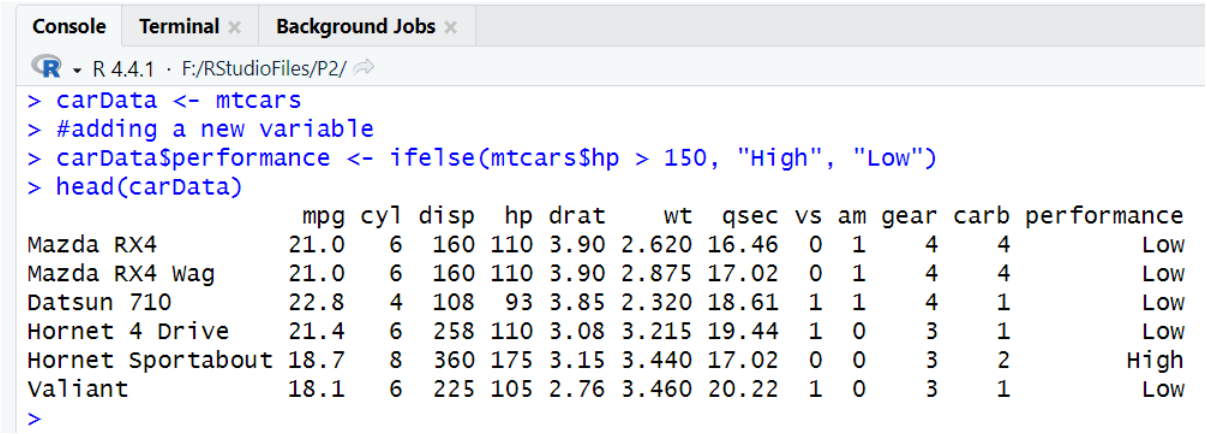
- performance = "High" if hp (horsepower) is greater than 150.
- performance = "Low" if hp is less than or equal to 150.

- Write the steps to add this new variable to the mtcars dataset.
- Display the first 6 rows of the updated dataset with the new column.

Source Code:

```
carData <- mtcars
#adding a new variable
carData$performance <- ifelse(mtcars$hp > 150, "High", "Low")
#first 6 rows of data
head(carData)
```

Output:



```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/P2/
> carData <- mtcars
> #adding a new variable
> carData$performance <- ifelse(mtcars$hp > 150, "High", "Low")
> head(carData)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb performance
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4         Low
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4         Low
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1         Low
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1         Low
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2        High
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1         Low
>
```

Question 4.

You have a dataset sales_data containing Product, Sales, and Region.

Add a new variable Sales_Category to categorize the sales into "High" (Sales > 1000) and "Low" (Sales <= 1000). Write the steps for this task.

Source Code:

```
#Sales Dataset
Product <- c("s24","s24Ultra","A55","A35","s23")
Sales <- c(1100,1500,900,800,1000)
Region <- c("Asia","Europe","North America","South America","Africa")
```

```
sales_data <- data.frame(Product,Sales,Region)
print(sales_data)

sales_data$Sales_Category <- ifelse(sales_data$Sales <= 1000, "Low",
"High")
print(sales_data)
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/P2/
> #Sales Dataset
> Product <- c("s24","s24Ultra","A55","A35","s23")
> Sales <- c(1100,1500,900,800,1000)
> Region <- c("Asia","Europe","North America","South America","Africa")
> sales_data <- data.frame(Product,Sales,Region)
> print(sales_data)
   Product Sales        Region
1      s24  1100          Asia
2 s24Ultra  1500        Europe
3      A55   900 North America
4      A35   800 South America
5      s23  1000        Africa
> sales_data$Sales_Category <- ifelse(sales_data$Sales <= 1000, "Low", "High")
> print(sales_data)
   Product Sales        Region Sales_Category
1      s24  1100          Asia           High
2 s24Ultra  1500        Europe           High
3      A55   900 North America            Low
4      A35   800 South America            Low
5      s23  1000        Africa            Low
>
```

23. Dealing with Missing Data

Question 5.

You are working with a dataset survey_data that contains missing values. Perform the following tasks:

- Identify which columns have missing values.
- Write the steps to replace all missing values in the column age with the mean age of the dataset.
- Remove any rows where more than 2 columns have missing values.

Source Code

```
#Loading the dataset
dataset <- read.csv("survey_data.csv")

#Identify the columns that have missing values
na_data <- colSums(is.na(survey_data)) > 0
na_columns <- names(survey_data)[na_data]
print(na_columns)

mean_age <- mean(survey_data$age, na.rm = TRUE)
survey_data$age[is.na(survey_data$age)] <- mean_age
print(survey_data)

survey_data <- survey_data[rowSums(is.na(survey_data)) <= 2, ]
print(survey_data)
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/P2/
> #Loading the dataset
> dataset <- read.csv("survey_data.csv")
>
> #Identify the columns that have missing values
> na_data <- colSums(is.na(survey_data)) > 0
> na_columns <- names(survey_data)[na_data]
> print(na_columns)
[1] "age"
>
> mean_age <- mean(survey_data$age, na.rm = TRUE)
> survey_data$age[is.na(survey_data$age)] <- mean_age
>
> print(survey_data)
   id       age gender income
1   1 25.00000   Male  36558
2   2 30.00000   Male  23594
3   3 32.14286   Male  45691
4   4 22.00000 Female  43148
5   5 45.00000   Male  41192
6   6 32.14286   Male  41455
7   7 34.00000   Male  32341
8   8 32.14286 Female  43206
9   9 29.00000 Female  40452
10 10 40.00000 Female  38055
```

```
> survey_data <- survey_data[rowSums(is.na(survey_data)) <= 2, ]
> print(survey_data)
   id       age gender income
1   1 25.00000   Male  36558
2   2 30.00000   Male  23594
3   3 32.14286   Male  45691
4   4 22.00000 Female  43148
5   5 45.00000   Male  41192
6   6 32.14286   Male  41455
7   7 34.00000   Male  32341
8   8 32.14286 Female  43206
9   9 29.00000 Female  40452
10 10 40.00000 Female  38055
>
```

Question 6.

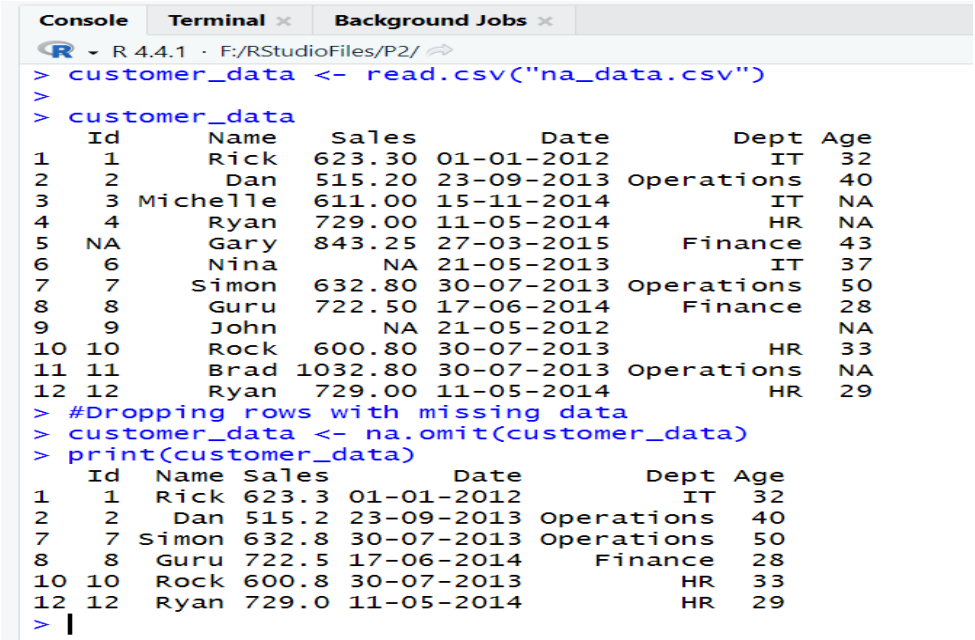
Consider a dataset customer_data with missing values in multiple columns:

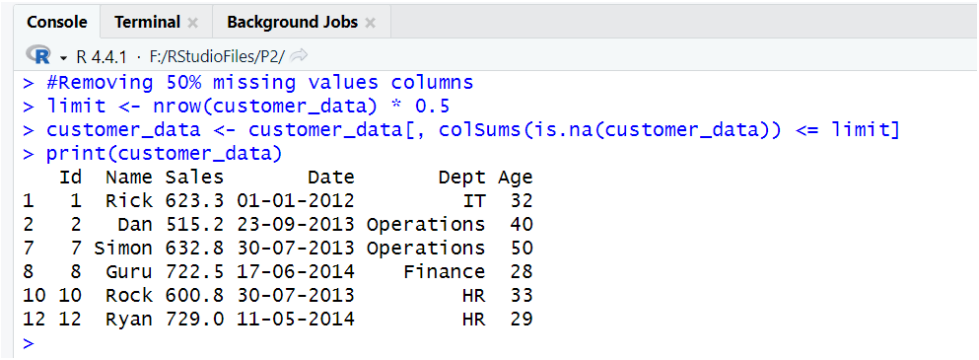
- Write the steps to drop rows that contain any missing values.
- If a column has more than 50% missing values, drop that column from the dataset.

Source Code:

```
customer_data <- read.csv("na_data.csv")
customer_data

#Dropping rows with missing data
customer_data <- na.omit(customer_data)
print(customer_data)

#Removing 50% missing values columns
limit <- nrow(customer_data) * 0.5
customer_data <- customer_data[,
colSums(is.na(customer_data)) <= limit]
print(customer_data)
```

Output:





24. Dealing with Categorical Data.

Question 7.

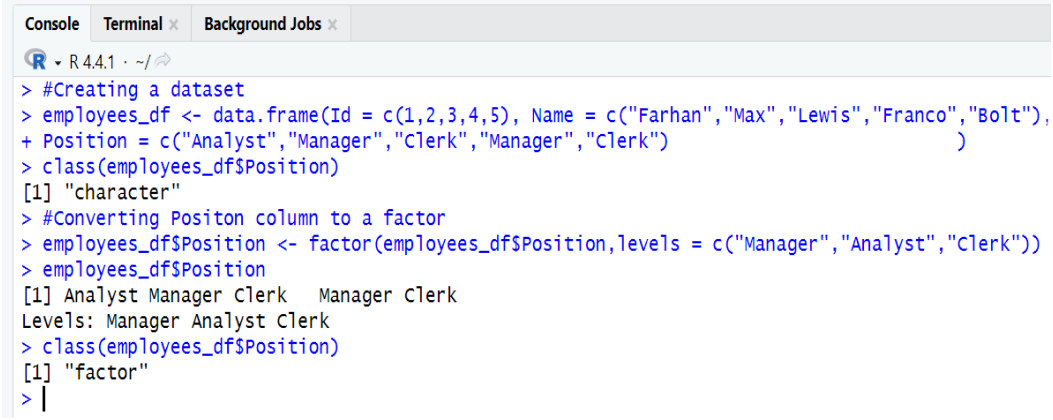
You are given a dataset employee's with a column Position that contains categorical data such as "Manager", "Analyst", and "Clerk".

- Convert the Position column to a factor.
- Write the steps to reorder the levels of the factor such that "Manager" comes first, followed by "Analyst", and "Clerk".

Source Code:

```
#Creating a dataset
 employees_df <- data.frame(Id = c(1,2,3,4,5),
 Name = c("Farhan","Max","Lewis","Franco","Bolt"),
 Position = c("Analyst","Manager","Clerk","Manager","Clerk"))

 class(employees_df$Position)
 #Converting Positon column to a factor
 employees_df$Position <- factor(
 employees_df$Position,levels = c("Manager","Analyst","Clerk"))
 employees_df$Position

 class(employees_df$Position)
```

Output:



Question 8.

You are analyzing a survey dataset where one column Gender contains categorical data ("Male", "Female"). Perform the following tasks:

- Convert the Gender column into a factor.
- Create dummy variables (binary variables) for the Gender column where 1 represents "Male" and 0 represents "Female".

Source Code:

```
#Creating the dataset
survey_data <- data.frame(
 id = 1:6,
 Name = c("Farhan","Jane","Max","Mary","Adnan","Amber"),
Gender = c("Male","Female","Male","Female","Male","Female"))
print(survey_data)


#Converting the Gender Column to a factor
 survey_data$Gender <- as.factor(survey_data$Gender)
 str(survey_data)

#Creating dummy Variables for the Gender Column.
 survey_data$Gender <- ifelse(survey_data$Gender == "Male", 1, 0)
#Updated Dataset
 print(survey_data)
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · ~/
> #Creating the dataset
> survey_data <- data.frame(
+     id = 1:6,
+     Name = c("Farhan","Jane","Max","Mary","Adnan","Amber"),
+     Gender = c("Male","Female","Male","Female","Male","Female")
+ )
> print(survey_data)
  id   Name Gender
1  1 Farhan   Male
2  2   Jane Female
3  3    Max   Male
4  4   Mary Female
5  5  Adnan   Male
6  6  Amber Female
> #Converting the Gender Column to a factor
> survey_data$Gender <- as.factor(survey_data$Gender)
> str(survey_data)
'data.frame':   6 obs. of  3 variables:
 $ id    : int  1 2 3 4 5 6
 $ Name  : chr  "Farhan" "Jane" "Max" "Mary" ...
 $ Gender: Factor w/ 2 levels "Female","Male": 2 1 2 1 2 1
>
```

```
Console   Terminal   Background Jobs
R 4.4.1 · ~/
> #Creating dummy Variables for the Gender Column.
> survey_data$Gender <- ifelse(survey_data$Gender == "Male", 1, 0)
> #Updated Dataset
> print(survey_data)
  id   Name Gender
1  1 Farhan      1
2  2   Jane      0
3  3    Max      1
4  4   Mary      0
5  5  Adnan      1
6  6  Amber      0
>
```

25. Data Reduction Using Sub-setting

Question 9.

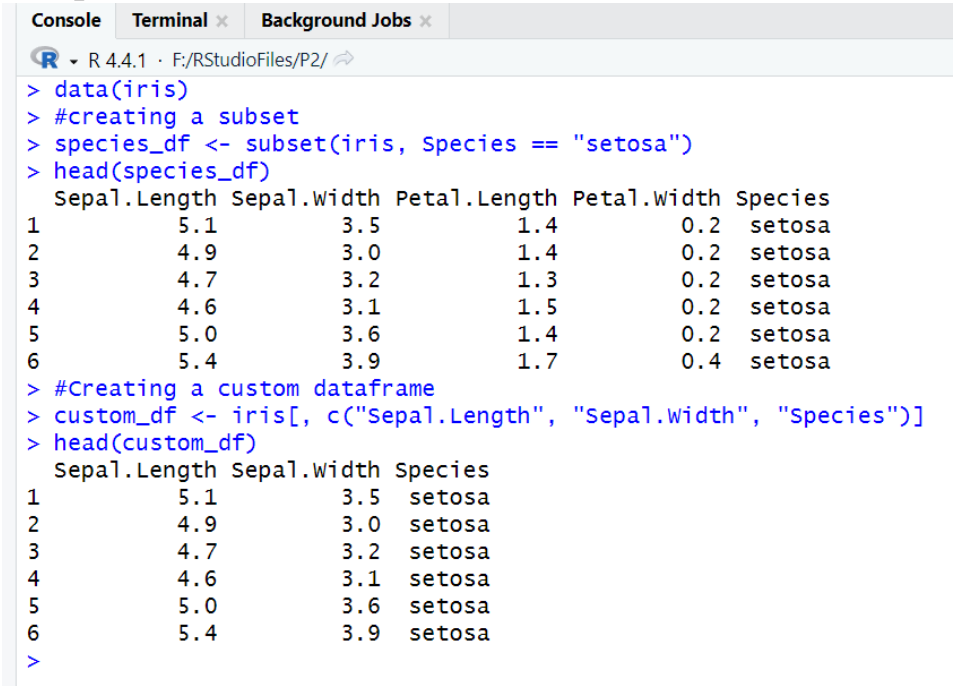
You are given the iris dataset, which contains 150 rows and 5 columns (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species). Perform the following tasks:

- Subset the dataset to include only the rows where Species is "setosa".
- Create a new data frame with only the columns Sepal.Length, Sepal.Width, and Species.

Source Code:

```
data(iris)
#creating a subset
species_df <- subset(iris, Species == "setosa")
head(species_df)

#Creating a custom dataframe
 custom_df <- iris[, c("Sepal.Length", "Sepal.Width", "Species")]
 head(custom_df)
```

Output:



Question 10.

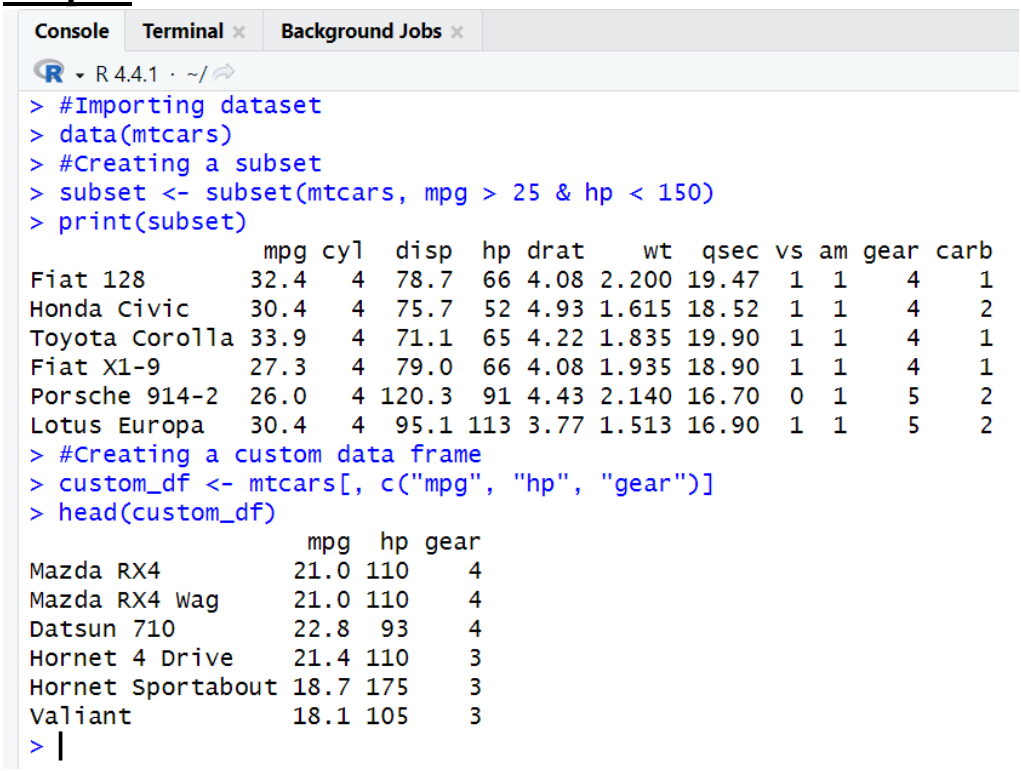
You are working with the mtcars dataset. Write the steps to:

- Subset the data to include only cars with mpg greater than 25 and hp less than 150.
- Create a new data frame that includes only the variables mpg, hp, and gear from the original dataset.

Source Code:

```
#Importing dataset
 data(mtcars)
#Creating a subset
  subset <- subset(mtcars, mpg > 25 & hp < 150)
  print(subset)

#Creating a custom data frame
 custom_df <- mtcars[, c("mpg", "hp", "gear")]
 head(custom_df)
```

Output:



<table>
<tr><td colspan="4">Name of Student: Shaikh Farhan Ahmed</td></tr>
<tr><td colspan="2">Roll Number: 50</td><td colspan="2">LAB Assignment Number: 3</td></tr>
<tr><td colspan="4">Title of LAB Assignment: To implement different types of Charts and Graphs like Histograms, Box Plots, Bar Charts, Line Graphs, Scatterplots, Pie Charts.</td></tr>
<tr><td colspan="2">DOP: 04 – 10 – 2024</td><td colspan="2">DOS: 10 – 10 – 2024</td></tr>
<tr><td>CO Mapped:
CO2</td><td>PO Mapped:
PO1, PO2, PO3
PO5, PO7,
PSO1, PSO2</td><td>Faculty Signature:</td><td>Marks:</td></tr>
</table>

Practical No: 3

Aim: To implement different types of Charts and Graphs like Histograms, Box Plots, Bar Charts, Line Graphs, Scatterplots, Pie Charts.

Description:

1. **Histogram**

a. A histogram is a chart that plots the distribution of a numeric variable's values as a series of bars.

b. Example:

```
data <- c (22, 30, 35, 40, 42, 45, 50, 55, 60, 65)
# Create a histogram
hist(data, main = "Histogram Example",
xlab = "Values",
col = "blue",
border = "black", breaks = 5) # You can customize the number of bins
```

c. Here,

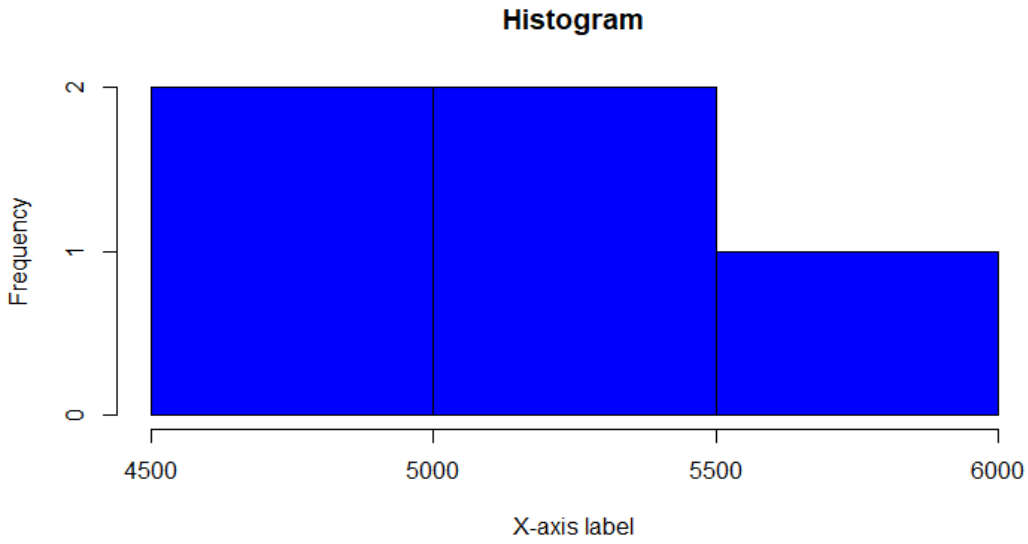
- `data`: The data you want to create a histogram for.
- `main`: The title of the histogram.
- `xlab`: Label for the x-axis.
- `col`: Color of the bars.
- `border`: Color of the border of the bars.
- `breaks`: Number of bins.

Source Code:

```
#Setting Directory
setwd("F:/RStudioFiles/P3")
data <- read.csv("SalesData1.csv")
data
```

```
#Create a histogram
 hist(data$toothpaste,main = "Histogram",
 xlab = "X-axis label",col =    "blue",border = "black")
```

Output:



2. Boxplots

a. Boxplots are used to visualize the distribution and spread of a dataset. You can create a boxplot using the `boxplot()` function:

b. Example:

```
data <- c (22, 30, 35, 40, 42, 45, 50, 55, 60, 65)
# Create a boxplot
boxplot(data,
main = "Boxplot Example",
col = "lightblue",
horizontal = TRUE) # Create a horizontal boxplot
```

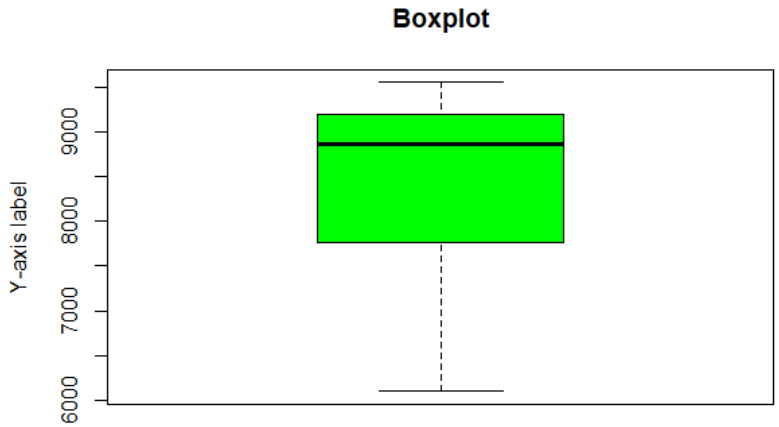
c. Here,

- `data`: The data for which you want to create a boxplot.
- `main`: The title of the boxplot.
- `col`: Color of the boxes.
- `horizontal`: Set to `TRUE` for a horizontal boxplot.

Source Code:

```
#Create a boxplot
boxplot(data$bathingsoap, main = "Boxplot", ylab = "Y-axis label",
col = "green")
```

Output:



3. Bar Charts

a. Bar charts are used to display categorical data. You can create bar charts using the `barplot()` function or the `ggplot2` package. Here's a basic example using the`barplot()` function:

b. Example:

```
categories <- c("Category A", "Category B", "Category C")
values <- c(10, 20, 30)
# Create a bar chart
  barplot(values,
  names.arg = categories,
  main = "Bar Chart Example",
  col = "green")
```

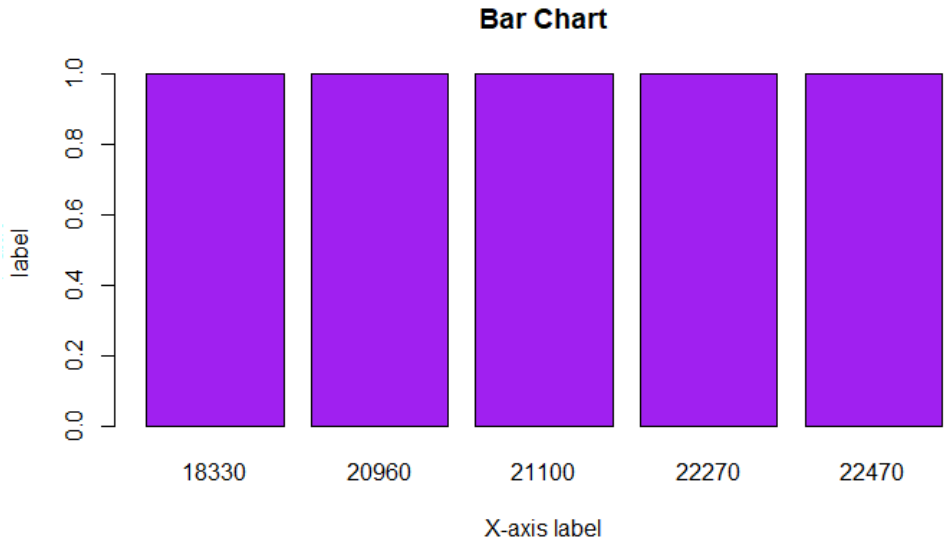
c. Here,

- `values`: Numeric values for the bars.
- `names.arg`: Names for the categories.
- `main`: The title of the bar chart.
- `col`: Color of the bars

Source Code:

```
#Create a bar chart

barplot(table(data$total_units), main = "Bar Chart",

xlab = "X-axis label", ylab = "Y-axis label", col = "purple")
```

Output:



4. Line Graphs:

a. Line graphs are used to visualize trends and relationships between data points over time or a continuous variable. You can create line graphs using the `plot()` function.

b. Example:
```
x <- 1:10
y <- x^2
# Create a line graph
plot(x, y, type = "l", # "l" for lines
main = "Line Graph Example",
xlab = "Time",
ylab = "Value", col = "red")
```

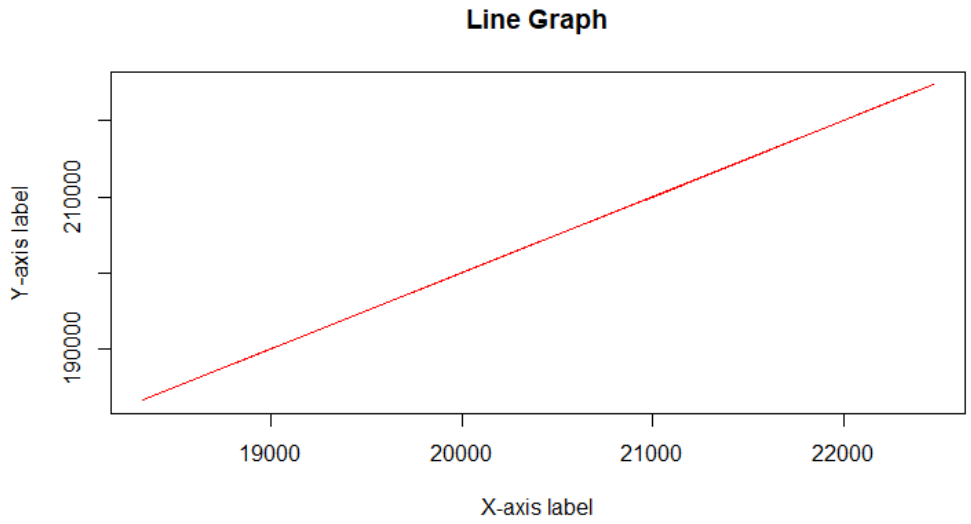
c. Here,
- `x` and `y`: The data for the x and y axes.

- `type`: "l" for a line graph.
- `main`: The title of the line graph.
- `xlab` and `ylab`: Labels for the x and y axes.
- `col`: Color of the line.

Source Code:

```
# Create a line graph

plot(data$total_units, data$total_profit, type = "l", col = "red", main = "Line Graph", xlab = "X-axis label",  ylab = "Y-axis label")
```

Output:



5. **Scatterplots:**

a. Scatterplots are used to show relationships between two variables. You can create scatterplots using the `plot()` function.

b. Example:

```
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 6, 8, 10)
# Create a scatterplot
plot(x, y,
main = "Scatterplot Example",
xlab = "X-Axis",
```

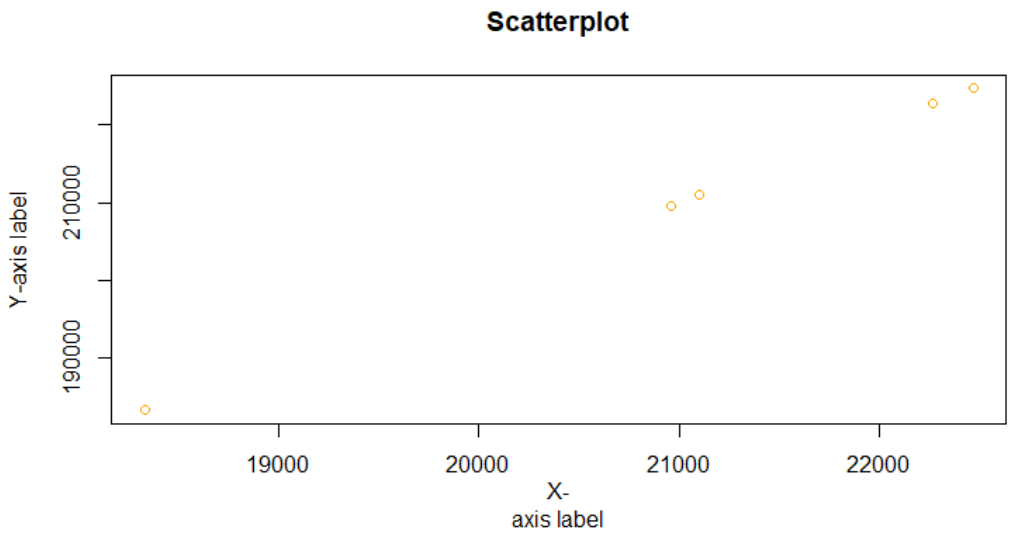
ylab = "Y-Axis", col = "blue")

c. Here,
   - `x` and `y`: The data for the x and y axes.
   - `main`: The title of the scatterplot.
   - `xlab` and `ylab`: Labels for the x and y axes.
   - `col`: Color of the points.

Source Code:

```
#Create a scatterplot

plot(data$total_units, data$total_profit, col = "orange", main = "Scatterplot",

xlab = "X-axis label", ylab = "Y-axis label")
```

Output:



6. Pie Charts:

a. Pie charts are used to represent parts of a whole. You can create pie charts using the `pie()` function.

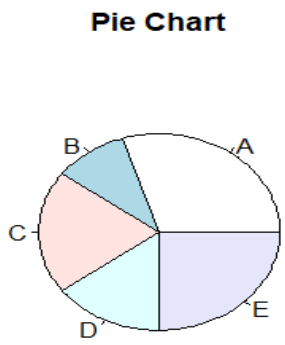
b. Example:
```
data <- c(10, 20, 30)
labels <- c("Category A", "Category B", "Category C")
# Create a pie chart
pie(data,
labels = labels,
main = "Pie Chart Example",
col = rainbow(length(data)))
```

c. Here,
- `data`: A vector of values for each segment.
- `labels`: Labels for each segment.
- `main`: The title of the pie chart.
- `col`: Color palette for the segments

Source Code:

```
# Create a pie chart

slices <- c(30, 10, 20, 15, 25)

lbls <- c("A", "B", "C", "D", "E")

pie(slices, labels = lbls, main = "Pie Chart")
```

Output:



Histogram

Problem Statement:

1. **You are provided with a dataset that contains the test scores of students from various schools. Your task is to:**
    - Load the dataset into R.
    - Generate a histogram for one of the numerical columns.
    - Customize the histogram by adjusting the labels, colors, and number of bins.
    - Analyze and interpret the resulting histogram to understand the distribution of the data.

Dataset:

You will be using the dataset named student_scores.csv, which contains the following columns:

- student_id: Unique ID of each student.
- name: Name of the student.
- age: Age of the student.
- school: School attended by the student.
- score: Test score of the student (range: 0-100).

Steps to Complete:

1. **Data Loading:**

- Download the student_scores.csv file
- Use the read.csv() function to load the dataset into R.

2. **Basic Histogram:**

- Generate a histogram for the score column using the hist() function.
- Set the title of the histogram to "Distribution of Student Scores".
- Label the x-axis as "Test Scores" and the y-axis as "Frequency".

3. **Customizing the Histogram:**

- Change the color of the histogram bars (e.g., "lightgreen").
- Add a border to the bars (e.g., "black").
- Adjust the number of bins to 10 using the breaks parameter.
- Optionally, set limits on the x-axis (e.g., from 0 to 100) using xlim().

4. **Analysis and Interpretation:**

- Based on the histogram, analyze the distribution of student scores. Is the data normally distributed, skewed, or does it have any outliers?
- Write a short paragraph interpreting the histogram. Identify patterns in the
- distribution of scores, such as whether the scores are clustered around a specific range, whether there is a skew, or if there are any unusual gaps.

Expected Output:

- A histogram showing the distribution of student test scores.
- Customization with labels, colors, and a specified number of bins.
- A brief interpretation of the distribution based on the histogram.

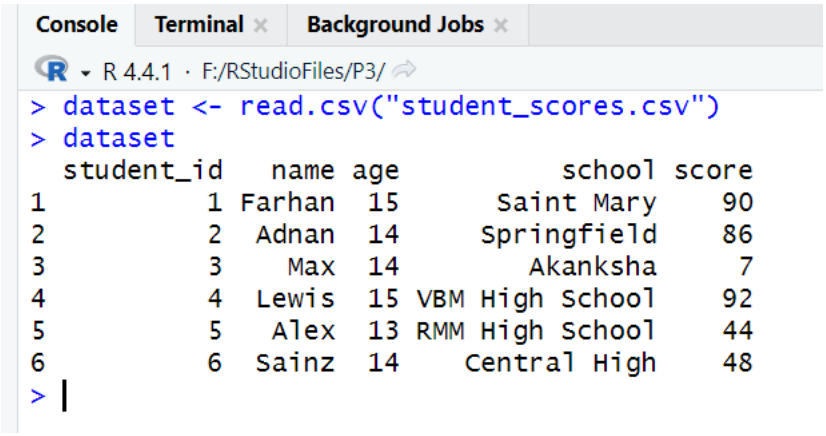
Solution:

1. **Loading Data into R.**

Source Code:

```
dataset <- read.csv("student_scores.csv")
dataset
```

Output:

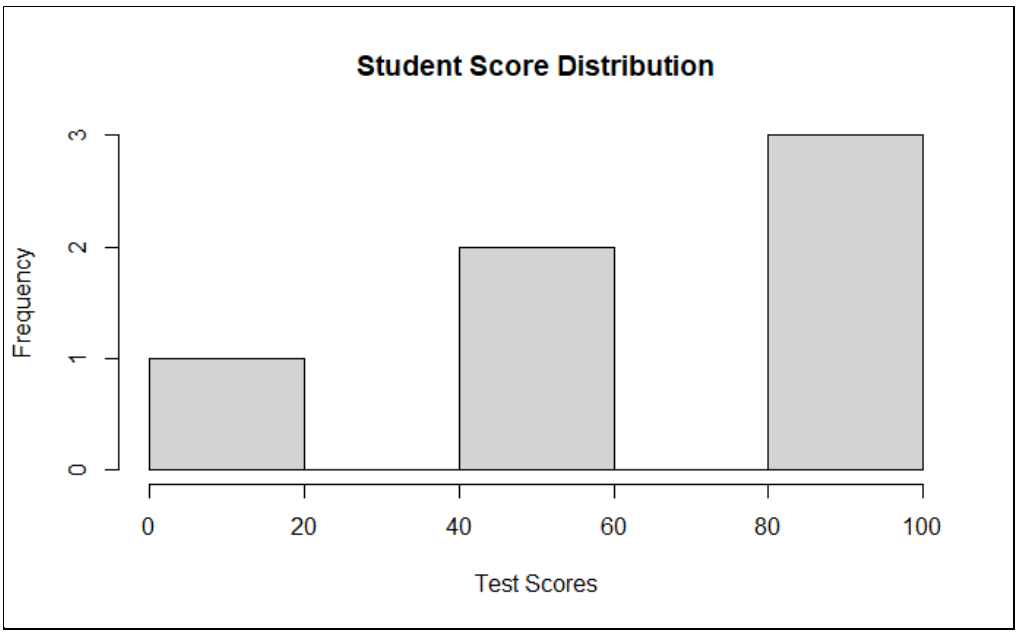


2. Plotting Basic Histogram.

Source Code:

```
hist(dataset$score, breaks = 4, main = “Student Score Distribution”,
xlab = “Test Scores”, ylab = “Frequency”)
```

Output:

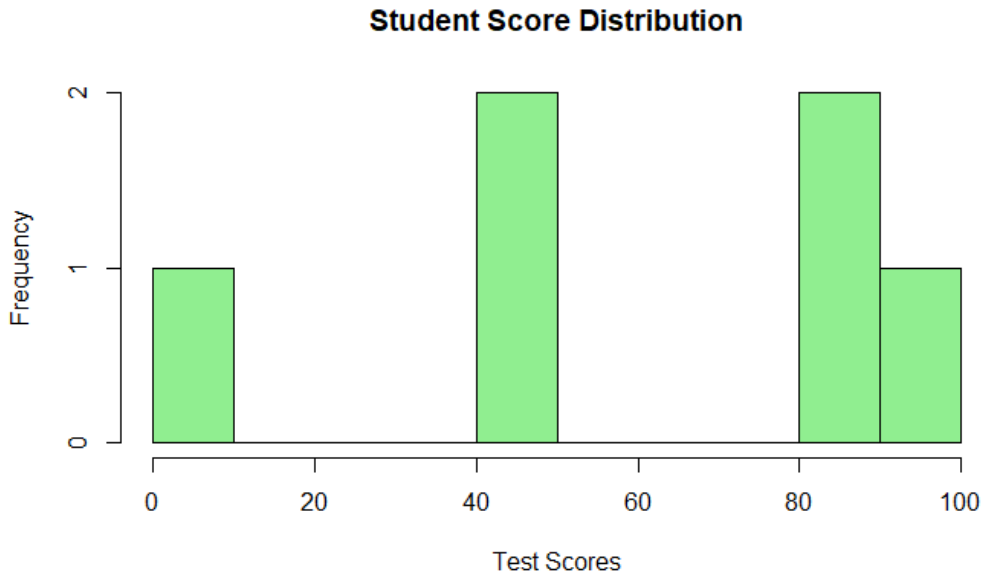


3. Customizing the Histogram:

Source Code:

```
#Customizing the histogram
hist(
    dataset$score,
    breaks = 8,
    main = "Student Score Distribution",
    xlab = "Test Scores",
    ylab = "Frequency",
    col = "lightgreen",
    border = "black",
    xlim = c(0,100)
)
```

Output:



4. **Analysis:**
   The histogram of the scores shows a clear clustering around the higher end, with most values concentrated between 40 and 100. Notably, there is a significant outlier at 7, which skews the distribution towards the lower end.

Boxplot

Problem Statement:

1. **You are provided with a dataset containing the test scores of students across different schools Your task is to:**
   - Load the dataset into R.
   - Generate a boxplot to visualize the distribution of the scores.
   - Customize the boxplot by adding labels and color.
   - Analyze and interpret the boxplot, focusing on the spread, outliers, and other statistical features.

Dataset:

You will be using the dataset named student_scores.csv, which contains the following columns:

- student_id: Unique ID of each student.
- name: Name of the student.
- age: Age of the student.
- school: School attended by the student.
- score: Test score of the student (range: 0-100).

Steps to Complete:

1. **Data Loading:**
- Download the student_scores.csv file
- Use the read.csv() function to load the dataset into R.

2. **Basic Boxplot:**
- Generate a basic boxplot of the score column to visualize the distribution of student scores.
- Set the title of the histogram to " Boxplot of Student Scores".
- Label the y-axis as "Test Scores".

3. **Customizing the Boxplot:**
- Add color to the box (e.g., col = "lightblue")
- Draw a horizontal boxplot using the horizontal = TRUE parameter.
- Use different color schemes for each school by plotting scores by school and adding the col parameter to differentiate.

4. **Analysis and Interpretation:**
- Analyze the boxplot to identify the range, interquartile range (IQR), and any outliers in the data.
- Comment on whether the scores are skewed and which schools have the widest or narrowest range of scores.
- Write a short interpretation, identifying key features such as the median, spread, and presence of any outliers in each school.

Expected Output:

- A boxplot showing the distribution of student test scores, either overall or grouped by school.
- Customization with labels and colors.
- A brief interpretation of the boxplot, focusing on the distribution and presence of outliers.

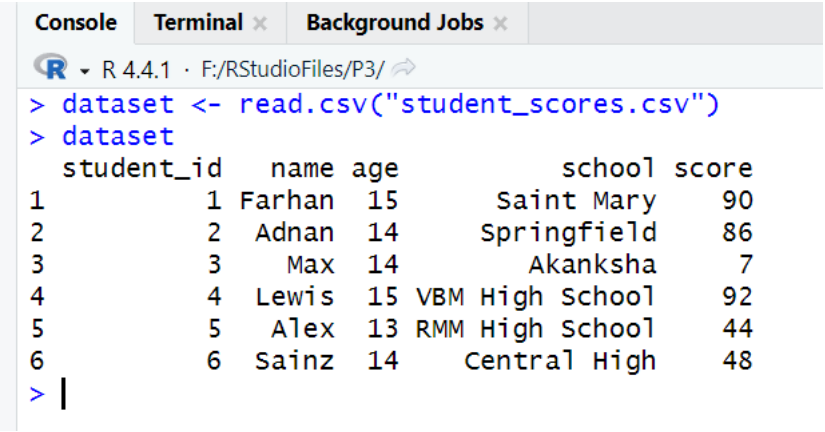
Solution:

1. **Loading Data into R.**

Source Code:

```
dataset <- read.csv("student_scores.csv")
dataset
```

Output:

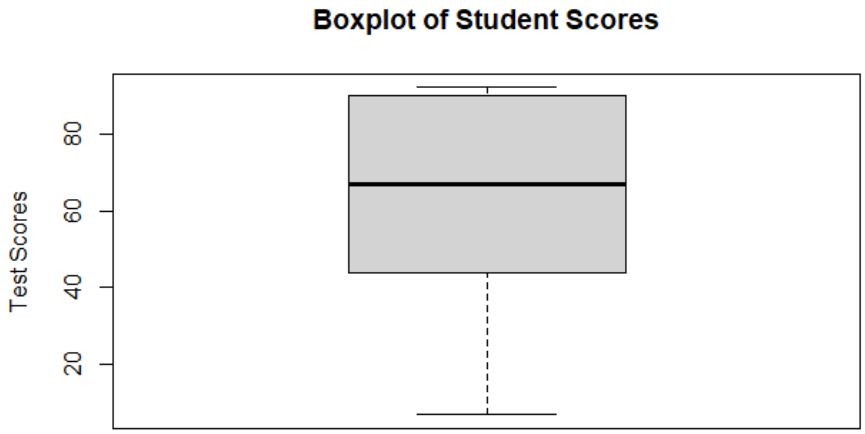


2. **Plot Basic Boxplot:**

Source Code:

```
boxplot(dataset$score, main = "Boxplot of Student Scores",
ylab = "Test Scores")
```

Output:

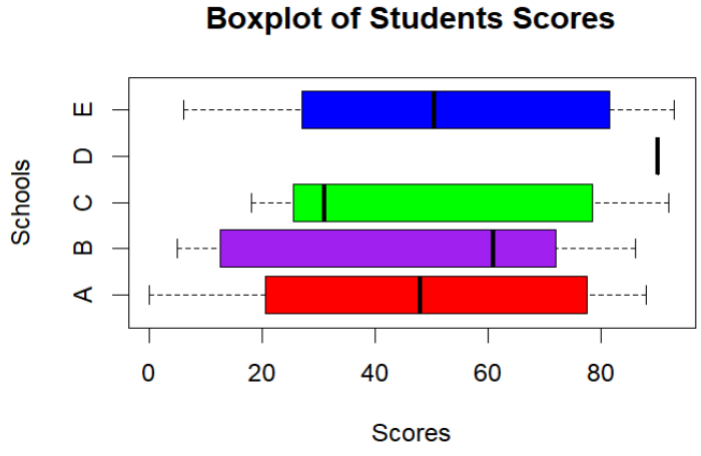


3. Customizing the Box Plot:

Source Code:

```
boxplot(
   dataset$score ~ dataset$school,
   main = "Boxplot of Students Scores",
   ylab = "Schools",
   xlab = "Scores",
   col = c("red","purple","green","yellow","blue"),
   horizontal = TRUE,
   names = c("A","B","C","D","E")
)
```

Output:



4. **Analysis:**

   For School A, the range of scores is from 0 to 90, with the first quartile (Q1) at 20 and the third quartile (Q3) at 78. This gives an interquartile range (IQR) of 20 to 78, indicating that 50% of students scored between these values. The median score is approximately 47, and there are no outliers present in the data.

   In School B, the range is from 5 to 87, with Q1 around 11 and Q3 around 72, resulting in an IQR of 11 to 72. Here, 50% of students scored within this range, and the median score is approximately 60. There are also no outliers in this dataset.

   For School C, the scores range from 18 to 95, with Q1 around 23 and Q3 around 79, leading to an IQR of 23 to 79. There are no outliers in the data for this school.

   School D has only one score of around 90 in its dataset.

   Lastly, School E's range is from 8 to 98, with Q1 around 23 and Q3 around 82, giving an IQR of 23 to 82. Fifty percent of students scored between these quartiles, and the median score is approximately 50. Similar to the other schools, there are no outliers in this dataset.

2. Bar chart

Problem Statement:

1. **You are provided with a set of data representing the number of students participating in different extracurricular activities at a school. Your task is to:**
   - Load the dataset into R.
   - Generate a bar chart to visualize the participation in each activity.
   - Customize the bar chart with labels, colors, and additional features.
   - Analyze and interpret the results.

Data:

The data you will use for this assignment is as follows:

Activity	Number of Students
Soccer	25
Basketball	18
Drama Club	12
Debate Club	10
Chess Club	8
Music Band	22

Steps to Complete:

1. **Data Input:**
- Manually input the data into R by creating two vectors: one for the activity names and one for the corresponding number of students.

2. **Basic Bar Chart:**
- Create a bar chart using the barplot() function to visualize the number of students
- Set the title of the chart to "Participation in Extracurricular Activities".
- Label the x-axis as "Activities" and the y-axis as "Number of Students"

3. **Customizing the Bar Chart:**
    - Add colors to the bars (e.g., col = "lightblue").
    - Rotate the labels on the x-axis for better readability using the las parameter.
    - Display data labels above each bar showing the exact number of students.
    - Optionally, change the orientation of the bars to horizontal by adding the horiz =TRUE parameter.

4. **Analysis and Interpretation:**

- Based on the bar chart, analyze the distribution of students across different activities.
- Identify which activity has the highest participation and which has the lowest.
- Write a short paragraph interpreting the results, highlighting any key insights or patterns

Expected Output:

- A bar chart showing the number of students participating in each extracurricular activity.
- Customization with labels, colors, and proper formatting.
- A brief interpretation of the chart, discussing the distribution of students across activities.

Solution:

1. **Loading Data into R.**

Source Code:

```
#Creating dataset
activity <- c("Soccer","Basketball","Drama Club",
"Debate Club","Chess Club","Music Band")

no_students <- c(25,18,12,10,8,22)

sports_df <- data.frame(activity,no_students)
print(sports_df)
```

Output:

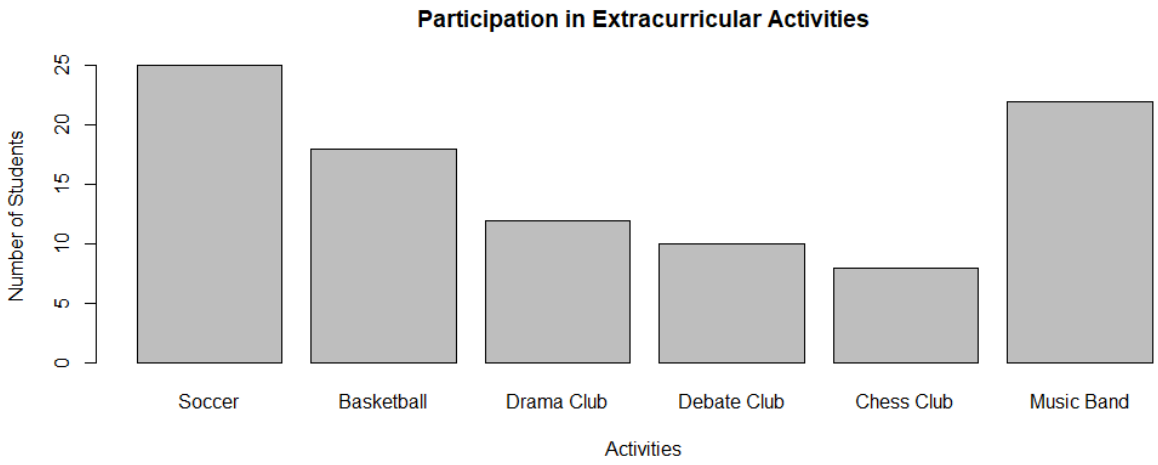
```
Console  Terminal  Background Jobs
R 4.4.1 · ~/
> #Creating dataset
> activity <- c("Soccer","Basketball","Drama Club","Debate Club","Chess Club","Music Band")
> no_students <- c(25,18,12,10,8,22)
>
> sports_df <- data.frame(activity,no_students)
> print(sports_df)
     activity no_students
1      Soccer          25
2  Basketball          18
3  Drama Club          12
4 Debate Club          10
5  Chess Club           8
6  Music Band          22
>
```

2. Plotting Bar Chart:

Source Code:

```
barplot(sports_df$no_students,
        main = "Participation in Extracurricular Activities",
        names.arg = activity,
        xlab = "Activities",
        ylab = "Number of Students"
)
```

Output:

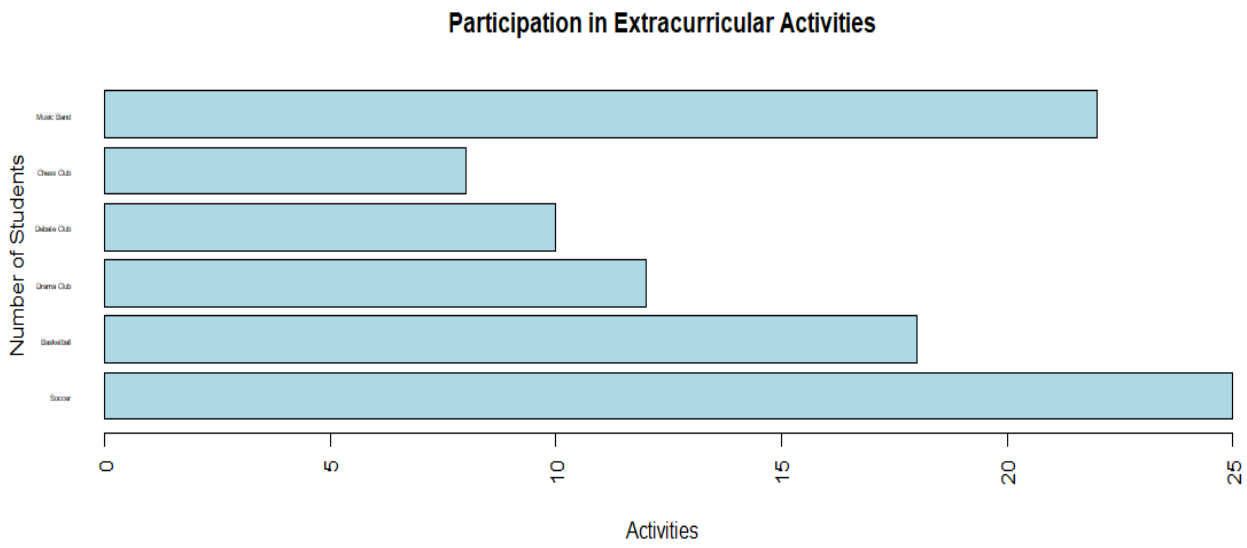


3. Customizing the Bar Chart:

Source Code:

```
barplot(sports_df$no_students,
        main = "Participation in Extracurricular Activities",
        names.arg = activity,
        xlab = "Activities",
        las = 2,
        col = "lightblue",
        cex.names = 0.4,
        horiz = T,
        ylab = "Number of Students"
)
```

Output:



4. Analysis:

The sport in which the most students participated was soccer. Twenty-five students engaged in soccer.

The second most popular activity was the music band activity. Twenty-two students engaged in music band.

The club with the fewest members was chess with only 8 students.

Line chart

Problem Statement:

1. **Repeat above steps for Line Chart and Scatter plot but with different dataset. Assume values as per requirement**

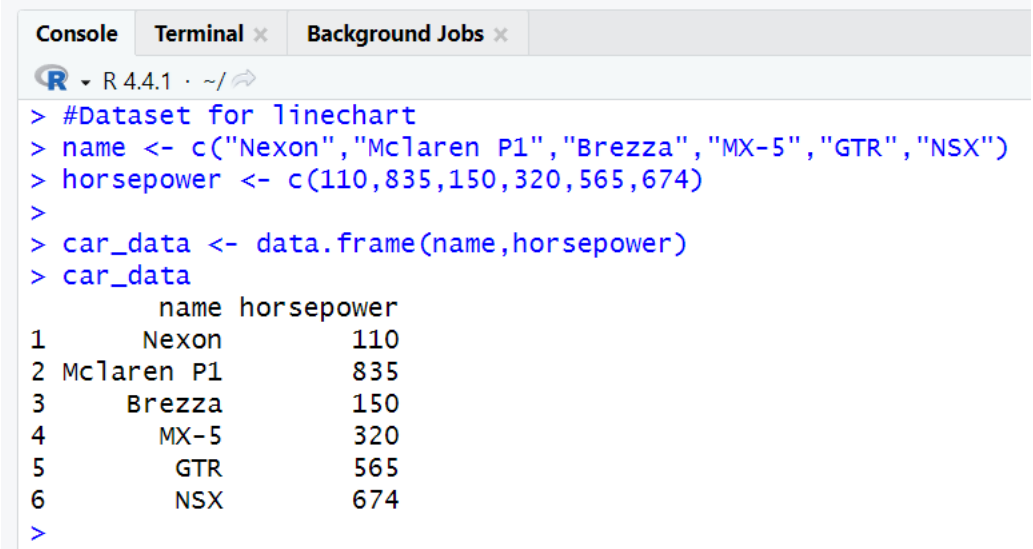
Solution:

1. **Loading Data into R.**

Source Code:

```
#Dataset for linechart
name <- c("Nexon","Mclaren P1","Brezza","MX-5","GTR","NSX")
horsepower <- c(110,835,150,320,565,674)
car_data <- data.frame(name,horsepower)
car_data
```

Output:

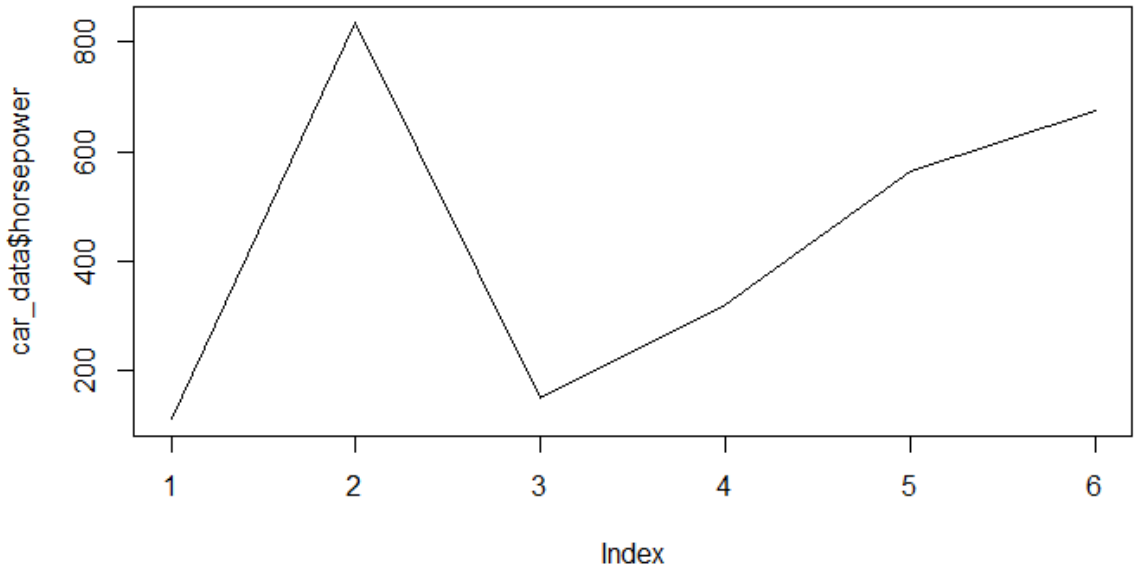


2. Plotting Line Chart:

Source Code:

```
plot(car_data$horsepower, type =”l”)
```

Output:

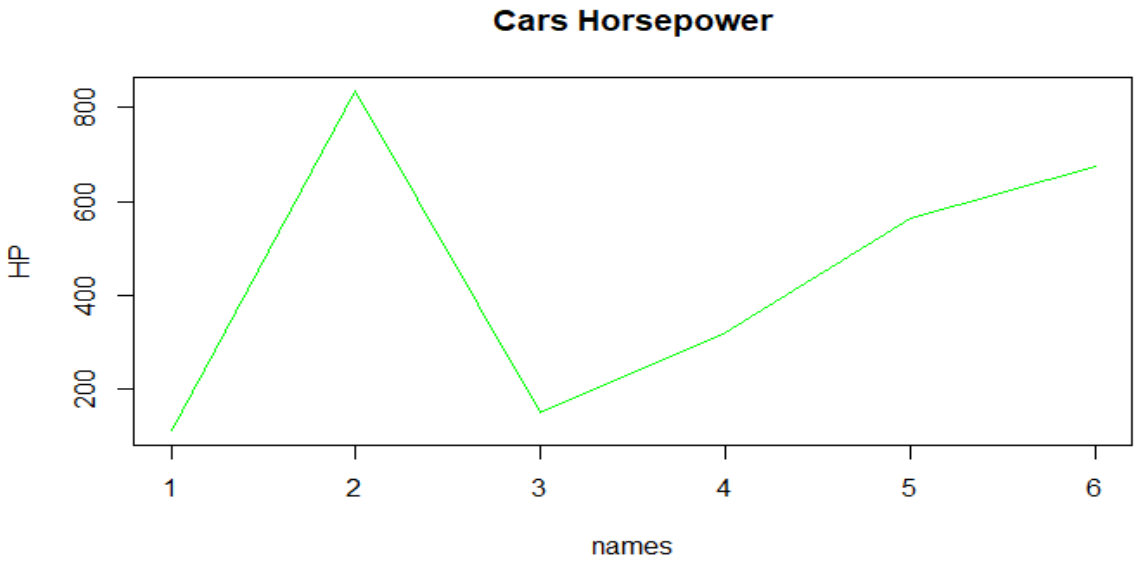


3. Customizing the Line Chart:

Source Code:

```
 plot(car_data$horsepower,
 main = "Cars Horsepower",
 type = "l",
   xlab = "names",
   ylab = "HP",
   col = "green"
)
```

Output:



4. **Analysis:**
   Highest Horsepower: You can expect the McLaren P1 to have the highest horsepower, which reflects its status as a high-performance supercar.
   Ideal Horsepower: The MX-5 has the ideal horsepower, which means it's the best between performance and compactness.
   Lowest Horsepower: The Nexon will likely have the lowest horsepower, as it is a compact car.

Pie chart

Problem Statement:

1. **You are provided with a set of data representing the percentage of students participating in different extracurricular activities at a school. Your task is to:**

- Input the data into R manually.
- Generate a pie chart to visualize the distribution of students across activities.
- Customize the pie chart by adding labels, colors, and other features.
- Analyze and interpret the pie chart to explain the distribution of students in various activities

Data:

The data you will use for this assignment is as follows:

Activity	Percentage of Students
Soccer	30%
Basketball	20%
Drama Club	15%
Debate Club	10%
Chess Club	5%
Music Band	20%

Steps to Complete:

1. Data Input:

- Manually input the data into R by creating two vectors: one for the activity names and one for the corresponding percentages.

Basic Pie Chart:

- Create a pie chart using the pie() function to visualize the distribution of students across activities.
- Set the title of the chart to "Participation in Extracurricular Activities".
- Ensure that the labels for each section of the pie chart show the activity names

Customizing the Pie Chart:

- Add colors to each slice of the pie (e.g., using rainbow() or specific colors).
- Display the percentage values for each activity on the pie chart.
- Optionally, create a 3D pie chart using the pie3D() function from the plotrix package (if installed).

Analysis and Interpretation:

- Analyze the pie chart and describe which activities have the highest and lowest participation rates.
- Write a short paragraph interpreting the distribution of students in the different activities, noting any significant differences or patterns in participation.

Expected Output:

- A pie chart showing the percentage of students participating in each extracurricular activity.
- Customization with labels, colors, and other features.
- A brief interpretation of the chart, discussing the distribution of students across the different activities.

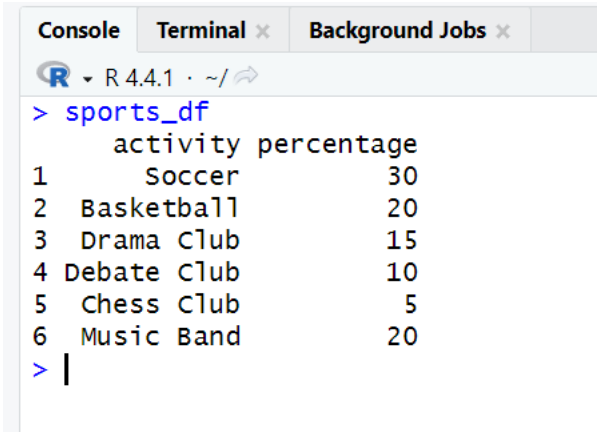
Solution:

1. **Loading Data into R.**

Source Code:

```
activity <- c("Soccer","Basketball","Drama Club","Debate Club","Chess Club","Music Band")
percentage <- c(30,20,15,10,5,20)
sports_df <- data.frame(activity,percentage)
sports_df
```

Output:

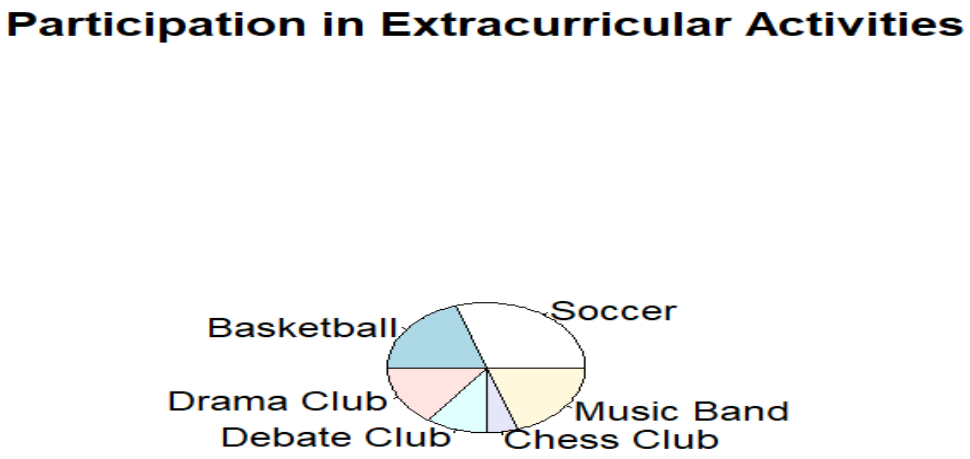


2. **Plotting Pie Chart:**

Source Code:

```
pie(sports_df$percentage,
   main = "Participation in Extracurricular Activities",
   labels = sports_df$activity)
```

Output:

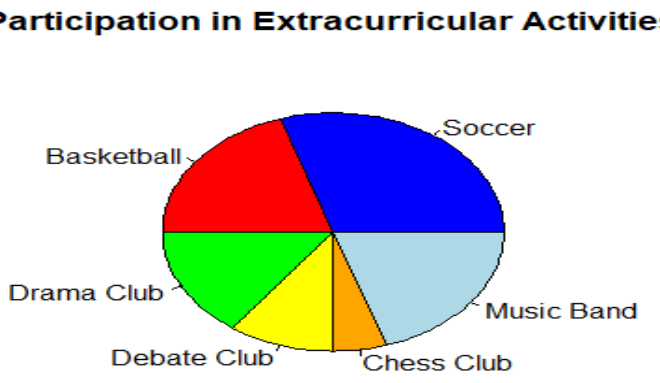


3. Customizing the Pie Chart:

Source Code:

```
#Customizing the piechart
pie(sports_df$percentage,
    main = "Participation in Extracurricular Activities",
    labels = sports_df$activity,
    col = c("blue","red","green","yellow","orange","lightblue")
)
```

Output:

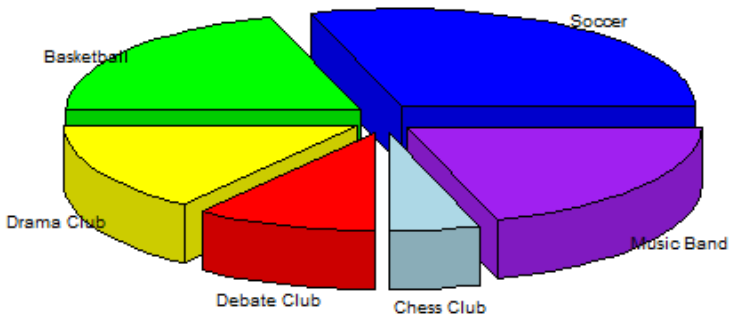


4. Customizing the 3D Pie Chart using pie3D():

Source Code:

```
#Creating a 3D PieChart
install.packages('plotrix')
library('plotrix')

pie3D(sports_df$percentage,
      col = c("blue","green","yellow","red","lightblue","purple"),
      labels = sports_df$activity,
      labelcex = 0.65,
      border = "black",
      explode = 0.1,
      mar = rep(1.75,4)
)
```

Output:



5. **Analysis:**
   Soccer sees the highest student participation, followed by basketball, while the chess club has the lowest turnout.
   Overall, students show less interest in intellectual activities such as debate and chess, and prefer outdoor physical activities like soccer and basketball.
   The music band stands out as the only indoor activity with a significant number of participants.

Conclusion:

From this practical I learned to implement different types of Charts and Graphs like Histograms, Box Plots, Bar Charts, Line Graphs, Scatterplots, Pie Charts and use the descriptive data visualization for data analysis.

<table>
<tr><td colspan="4">Name of Student: Shaikh Farhan Ahmed</td></tr>
<tr><td colspan="2">Roll Number: 50</td><td colspan="2">LAB Assignment Number: 4</td></tr>
<tr><td colspan="4">Title of LAB Assignment: To implement data Visualization with ggplot2</td></tr>
<tr><td colspan="2">DOP: 11 – 10 – 2024</td><td colspan="2">DOS: 17 – 10 – 2024</td></tr>
<tr><td>CO Mapped:
CO2</td><td>PO Mapped:
PO1, PO2, PO3, PO5, PO7,
PSO1, PSO2</td><td>Faculty Signature :</td><td>Marks:</td></tr>
</table>

Practical No: 4

AIM : Implement data visualization with ggplot2

Theory :
Data visualization is a key technique for extracting insights from data and effectively communicating those insights to others. In R, ggplot2 is a powerful package designed for creating a wide range of data visualizations based on a theoretical framework called the Grammar of Graphics. Developed by Leland Wilkinson, the Grammar of Graphics provides structured and coherent way to think about and construct statistical graphics.ggplot2, created by Hadley Wickham, implements this theory and simplifies theprocess of creating complex visualizations.

The Grammar of Graphics: A Layered Approach
The Grammar of Graphics provides a declarative approach to creating visualizations by treating each plot as a composition of distinct, modular components (layers). It moves beyond simple charts by abstracting the fundamental principles underlying all types of data visualization. Each plot is viewed as a combination of several elements: data, aesthetics, geoms (geometrical objects), scales, statistical transformations, coordinates, and faceting.

The basic components of the Grammar of Graphics that are used in ggplot2 are:

1. Data: The foundation of any visualization is the data itself. Data is usually provided as a dataframe or a tibble in R. Every visualization begins with defining the dataset that will be visualized.

2. Aesthetics (aes): Aesthetics are the "mappings" that associate variables in the dataset to visual properties of the plot, such as position (x and y coordinates), color, size, shape, etc. The aesthetics layer maps data variables to visual properties of graphical objects.

Common aesthetic mappings include:

- x and y: Cartesian coordinates
- color: The color of points, lines, or other geoms
- size: The size of points or lines
- shape: The shape of points

3. Geometries (Geoms): Geoms represent the graphical objects used to display the data. They determine the type of plot that is drawn, such as points for scatter plots (geom_point), lines for line plots (geom_line), bars for bar charts (geom_bar), and so on.

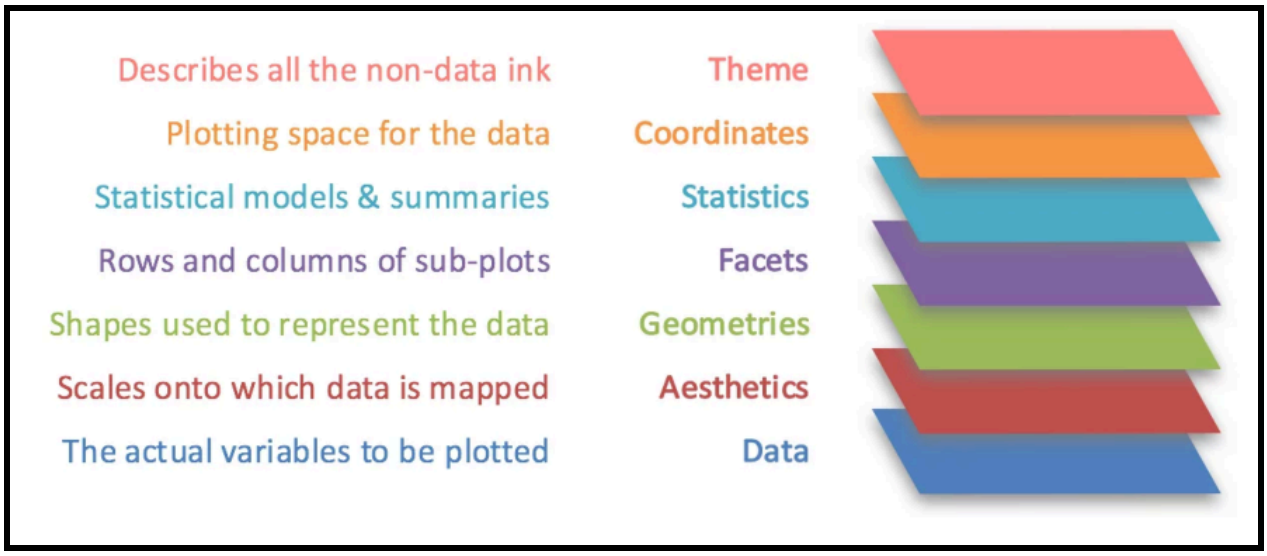
4. Scales: Scales control how data values are mapped to aesthetic properties like position, size, and color. They allow customization of axes and legends, as well as the transformation of data values (e.g., log scales, continuous vs categorical scales).

5. Facets: Faceting allows the creation of multiple plots, split by the values of one or more categorical variables, each plot representing a subset of the data. Faceting can be achieved using facet_wrap (one variable) or facet_grid (two variables).

6. Statistical Transformations: Many visualizations involve some kind of statistical transformation, such as summarizing the data, calculating means, or fitting a model. ggplot2 allows built-in statistical transformations like smoothing (fitting lines) and binning for histograms.

7. Coordinate Systems: Coordinate systems define how data is positioned on the plot. The default is Cartesian coordinates (coord_cartesian), but ggplot2 also supports other types, such as polar coordinates (coord_polar) for pie charts.

8. Themes: Themes control the non-data elements of the plot, such as titles, labels,legends, background, and grids. ggplot2 allows you to fully customize these elements using functions like theme(), and provides built-in themes (e.g.,theme_bw(), theme_minimal()).



Histogram with ggplot2

A histogram is a graphical representation used to display the distribution of a continuous variable by dividing the data into bins or intervals and counting the number of observations within each bin. It provides insights into the frequency and spread of the data, highlighting aspects like central tendency, variability, skewness, and the presence of outliers.
In ggplot2, the function geom_histogram() is used to create histograms. This function automatically divides the data into bins and plots the frequency or density of data points in each bin. The binwidth parameter allows you to control the width of the bins, influencing how the data distribution is visualized.

Syntax:

```
library(ggplot2)
ggplot(data = your_data, aes(x = your_variable)) +
geom_histogram(
binwidth = your_bin_width,
color = "outline_color",
fill = "fill_color",
alpha = transparency,
```

```
position = "position"
) +
labs(
title = "Plot Title",
x = "X-axis Label",
y = "Y-axis Label"
)
```

Explanation :
1. library(ggplot2): This line loads the ggplot2 package.

2. ggplot(data = your_data, aes(x = your_variable)): This initiates a ggplot object and specifies your dataset (your_data) and the variable you want to create a histogram of (your_variable). You can use the aes() function to map aesthetics.

3. geom_histogram(): This function adds a layer to the plot to create the histogram.
You can customize it with several arguments:

- binwidth: This controls the width of the bins in the histogram. You can set it to a specific value or let ggplot2 calculate it.
- color: This specifies the outline color of the bars in the histogram.
- fill: This specifies the fill color of the bars in the histogram.
- alpha: This controls the transparency of the bars.
- position: This argument determines how bars are positioned in the plot.The default is "stack," but you can change it to "dodge," "fill," or other options.

Boxplot with ggplot2

Creating a boxplot in ggplot2 involves specifying your data,aesthetics, and then using the geom_boxplot() function to represent the data as a boxplot. Here's the syntax and an explanation for creating a boxplot using ggplot2:

Syntax:

```
library(ggplot2)
ggplot(data = your_data, aes(x = your_x_variable, y =
your_y_variable))
+
geom_boxplot(
color = "outline_color",
fill = "fill_color",
notch = FALSE,
width = your_width,
outlier.shape = "shape"
) +
labs(
title = "Plot Title",
x = "X-axis Label",
y = "Y-axis Label"
)
```

Explanation:

1. library(ggplot2): This line loads the ggplot2 package if it's not already loaded.

2. ggplot(data = your_data, aes(x = your_x_variable, y =your_y_variable)): This initiates a ggplot object and specifiesyour dataset (your_data) and the variables you want to use forthe x-axis (your_x_variable) and y-axis (your_y_variable). Youcan use the aes() function to map aesthetics.

3. geom_boxplot():This function adds a layer to the plot to create the boxplot. You
can customize it with several arguments:

- color: This specifies the outline color of the boxes.
- fill: This specifies the fill color of the boxes.
- notch: A logical value that determines whether notches should be included in the boxes to show the confidence interval around the median.

- width: This controls the width of the boxes.
- outlier.shape: This determines the shape of the outlier points.

4. labs(title = "Plot Title", x = "X-axis Label", y = "Y-axis Label"): This function is used to add labels to the plot. You can provide a title, x-axis label, and y-axis label for your boxplot.

Line Graph with ggplot2

A line graph is used to visualize the relationship between two continuous variables, particularly when one variable represents time or an ordered sequence. Line graphs are particularly useful for showing trends over time or ordered categories. In ggplot2, the geom_line() function is used to create line graphs.

Syntax:

```
library(ggplot2)
ggplot(data = your_data, aes(x = your_x_variable, y =
your_y_variable)) +
geom_line(
color = "line_color",
size = your_line_size,
linetype = "line_type"
) +
labs(
title = "Plot Title",
x = "X-axis Label",
y = "Y-axis Label" )
```

Explanation:

- library(ggplot2): Loads the ggplot2 package.
- ggplot(data = your_data, aes(x = your_x_variable, y =
- your_y_variable)): Initializes the plot object and specifies the data and variables for the x and y axes.

- geom_line(): Adds a line graph layer. You can customize:
  color: Specifies the line color.
  size: Adjusts the thickness of the line.
  linetype: Changes the type of line (e.g., "solid", "dashed").
- labs(title = "Plot Title", x = "X-axis Label", y = "Y-axis Label"): Adds labels and a title to the graph.

Scatter Plot with ggplot2

A scatter plot is used to visualize the relationship between two continuous variables by representing them as points in a Cartesian plane. In ggplot2, geom_point() is used to create scatter plots.

Syntax:

```
library(ggplot2)
ggplot(data = your_data, aes(x = your_x_variable, y =
your_y_variable)) +
geom_point(
color = "point_color",
size = your_point_size,
shape = "point_shape"
) +
labs(
title = "Plot Title",
x = "X-axis Label",
y = "Y-axis Label")
```

Explanation:

- library(ggplot2): Loads the ggplot2 package.
- ggplot(data = your_data, aes(x = your_x_variable, y =
- your_y_variable)): Initiates the plot object and maps the variables for the x
- and y axes.
- geom_point(): Adds points to the plot. You can customize:
  color: Changes the point color.
  size: Adjusts the size of points.
  shape: Specifies the shape of the points (e.g., circles, triangles).

- labs(title = "Plot Title", x = "X-axis Label", y = "Y-axis Label"): Adds labels and a title to the plot.

Pie Chart with ggplot2

A pie chart is used to represent proportions or percentages of a whole. Although ggplot2 doesn't directly support pie charts, they can be created by converting the data into polar coordinates using coord_polar().

Syntax:

```
library(ggplot2)
ggplot(data = your_data, aes(x = "", y = your_value_variable,
fill = your_category_variable)) +
geom_bar(width = 1, stat = "identity") +
coord_polar("y") +
labs(
title = "Pie Chart Title",
fill = "Legend Title" )
```

Explanation:

- library(ggplot2): Loads the ggplot2 package.
- ggplot(data = your_data, aes(x = "", y = your_value_variable, fill = your_category_variable)): Initializes a bar plot with categories in fill and their proportions in y.
- geom_bar(): Adds bars with stat = "identity" to plot the raw values.
- coord_polar("y"): Converts the bar plot into a pie chart by transforming to polar coordinates.
- labs(title = "Pie Chart Title", fill = "Legend Title"): Adds a title and legend label.

Density Plot with ggplot2

A density plot shows the distribution of a continuous variable, providing a smoothed estimate of the variable's density. The geom_density() function is used in ggplot2 to create these plots.

Syntax:

```
library(ggplot2)
ggplot(data = your_data, aes(x = your_variable)) +
geom_density(
fill = "fill_color",
color = "line_color",
alpha = your_transparency_value
) +
labs(
title = "Density Plot Title",
x = "X-axis Label",
y = "Y-axis Label" )
```

Explanation:

- library(ggplot2): Loads the ggplot2 package.
- ggplot(data = your_data, aes(x = your_variable)): Initializes the
- plot with the variable for which you want to estimate the density.
- geom_density(): Creates a density plot. You can customize:
  fill: Specifies the fill color under the density curve.
  color: Defines the outline color of the curve.
  alpha: Controls the transparency of the fill.
- labs(title = "Density Plot Title", x = "X-axis Label", y = "Y-axis Label"): Adds labels and a title to the plot.

Heat Map with ggplot2

A heat map visualizes data in a matrix format, using color to indicate the intensity or concentration of values in the matrix. In ggplot2, geom_tile() is used to create heat maps.

Syntax:

```
library(ggplot2)
ggplot(data = your_data, aes(x = your_x_variable, y =
your_y_variable, fill = your_fill_variable)) +
geom_tile(color = "tile_outline_color") +
scale_fill_gradient(low = "low_color", high = "high_color") +
labs(
title = "Heat Map Title",
x = "X-axis Label",
y = "Y-axis Label",
fill = "Legend Title" )
```

Explanation:

- library(ggplot2): Loads the ggplot2 package.
- ggplot(data = your_data, aes(x = your_x_variable, y = your_y_variable, fill = your_fill_variable)): Maps x and y variables to the axes and a third variable to fill color.
- geom_tile(): Creates a heat map by filling the grid cells with color based on data values.
  color: Sets the outline color of the tiles.
  scale_fill_gradient(): Defines the color gradient to show the intensity from a low value to a high value.
- labs(title = "Heat Map Title", x = "X-axis Label", y = "Y-axis Label", fill = "Legend Title") :
  Adds labels, a title, and a legend title for the heat map.

Plots:

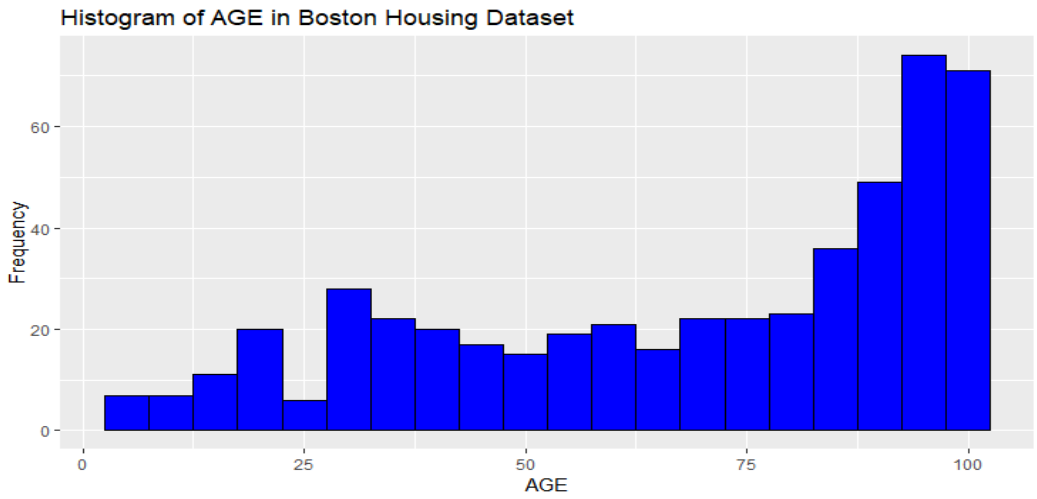
1)Histogram

Source Code:

```
#Loading dataset
dataset <- read.csv("housing.csv")
head(dataset,8)

ggplot(data = dataset , aes(x= AGE)) +
geom_histogram(binwidth = 5,fill="blue",color='black')+
labs(title="Histogram of AGE in Boston Housing Dataset",
x= "AGE",
y= "Frequency")
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/P4/
> #Loading dataset
> dataset <- read.csv("housing.csv")
> head(dataset,8)
     CRIM    ZN INDUS CHAS    NOX     RM  AGE    DIS RAD TAX PTRATIO      B LSTAT MEDV
1 0.00632 18.0  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98 24.0
2 0.02731  0.0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14 21.6
3 0.02729  0.0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03 34.7
4 0.03237  0.0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94 33.4
5 0.06905  0.0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33 36.2
6 0.02985  0.0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21 28.7
7 0.08829 12.5  7.87    0 0.524 6.012 66.6 5.5605   5 311    15.2 395.60 12.43 22.9
8 0.14455 12.5  7.87    0 0.524 6.172 96.1 5.9505   5 311    15.2 396.90 19.15 27.1
>
```



Explanation :

- Import library ggplot2 using library() function
- Create a ggplot using ggplot() function pass two arguments data= housing_data the dataset which we are visualizing and mapping = aes(x=AGE). aes() is used to map the aesthetics that is specify that “AGE” should be on x-axis
- Adding the histogram layer with geom_histogram(). Customizing it with bindwidth,fill color,outline color.
- Use labs() to add the title to the plot and label x and y axis

Analysis :

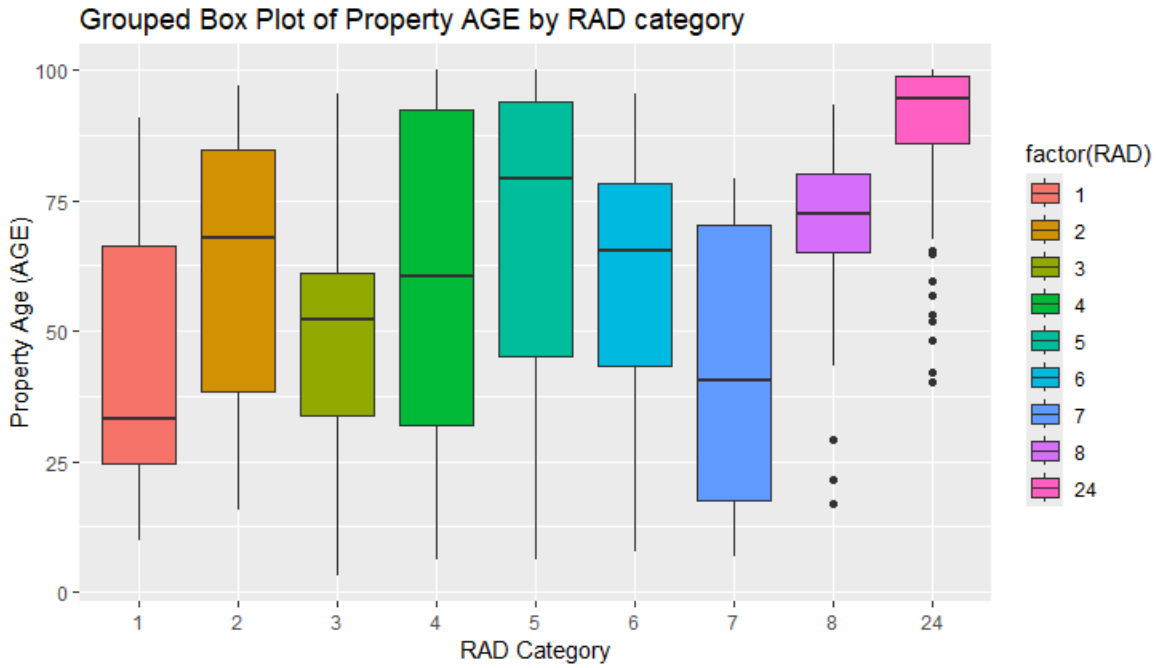
- The distribution is bimodal, with two noticeable peaks.
- A significant portion of the houses has an AGE value close to 100, indicating that many homes in the dataset are relatively old (near or over 100 years).
- There is also a smaller peak around 20-30, indicating some newer homes as well.
- The frequency declines from 0 to 25 years, then becomes relatively uniform between 25 and 75 before a steep rise after 75 years.
- Right-Skewed: The distribution is skewed toward older homes, with a notable concentration of homes aged 75 to 100.

2) Boxplot

Source Code:

```
#boxplot
ggplot(data = housing_data, aes(x=factor(RAD), y = AGE,fill=factor(RAD)))+
geom_boxplot() +
labs(
title = "Grouped Box Plot of Property AGE by RAD category",
x = "RAD Category",
y = "Property Age (AGE)" )
```

Output:



Explanation :

- ggplot(data = housing_data, aes(x = factor(RAD), y = AGE, fill = factor(RAD))):
- This initializes a ggplot object with the dataset.
- aes() (aesthetics) defines the variables to be plotted:
  x = factor(RAD): The x-axis is assigned to the RAD variable, which is converted into a factor (categorical variable). RAD likely represents some categorical variable, like the "index of accessibility to radial highways." y = AGE: The y-axis represents the AGE variable, presumably the age of properties.
- fill = factor(RAD): The fill color of the boxplots is determined by the
- RAD variable, allowing different categories of RAD to be visually distinguished.

Analysis :

1. **RAD Category 24:**
   - This category has the highest median property age, close to 100 years, and the distribution of ages is tightly clustered near the upper end.
   - Several outliers appear in this category, indicating a few properties with ages significantly lower than the median.
2. **RAD Categories 1 to 8:**
   - These categories show more spread in the age distribution.
   - RAD 1, 2, and 3: The age distribution is relatively wide, with medians around 50-75 years, but there is a greater spread and some younger properties (outliers in RAD 8).
   - RAD 4 and 5: These categories have slightly higher medians and a narrower spread.
   - RAD 8: Shows more spread with multiple outliers on the lower end, indicating properties that are much younger than the majority in that Category.

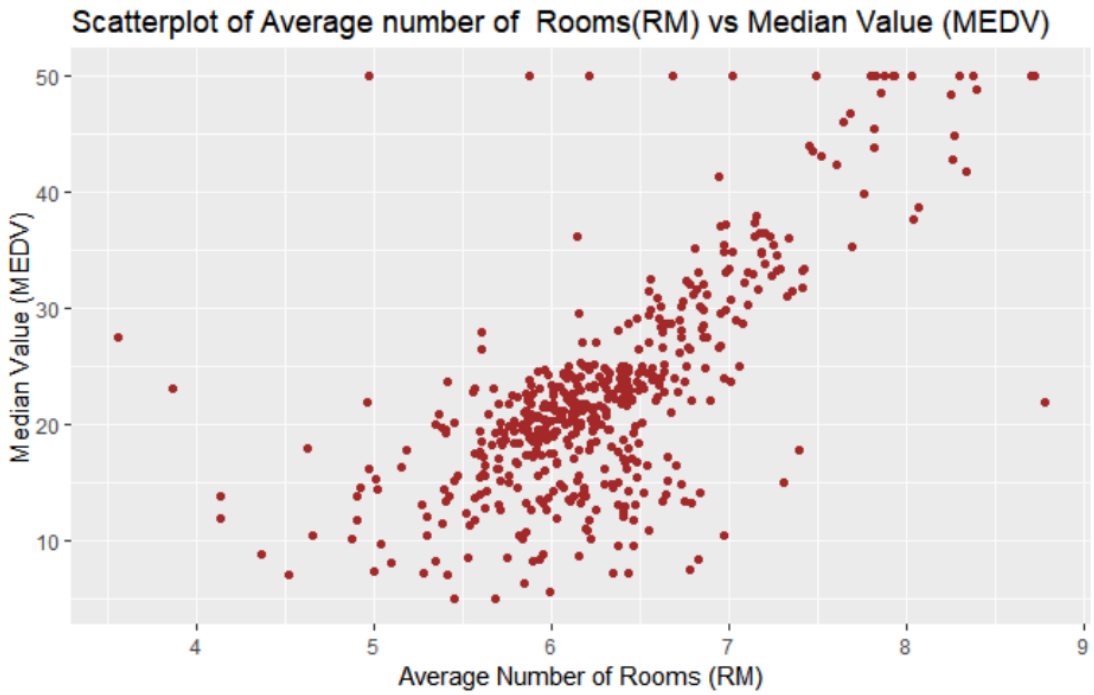
3. **General Trend:**
   - The distribution of property ages varies by RAD category, but most categories show properties that are quite old, with the median property age often around 50 to 75 years.
   - RAD Category 24 stands out for having the oldest properties.

3) Scatter Plot

Source Code:

```
#scatterplot
#create a scatterplot for
"RM" and "MEDV"
ggplot(data = dataset ,aes(x = RM,y=MEDV)) +
geom_point(color = "brown")+
labs(title = "Scatterplot of Average number of
Rooms(RM) vs Median Value
(MEDV)",
x="Average Number of Rooms (RM)",
y="Median Value (MEDV)")
```

Output:



Explanation :

1. Create a scatter plot using ggplot(). Map the “RM” variable to the x axis using x = RM and the “MEDV” variable to the y axis using y = MEDV
2. Use geom_plot() to add the data points to the plot. You can customize the color of the points to your liking.
3. Use labs() to add a title to the plot and label the x axis and y axis.

Analysis :

1. **Positive Correlation:**
There is a clear positive trend: as the average number of rooms (RM) increases, the median home value (MEDV) also tends to increase.
This suggests that houses with more rooms are generally more expensive, which is expected in real estate markets where larger homes often have higher values.

2. **Concentration of Data:**
The majority of data points cluster between 5 and 7 rooms on the x-axis, and 15 to 35 (in $1000's) on the y-axis. This indicates that most homes in the dataset have between 5 and 7 rooms, and their median values range between $15,000 and $35,000.

3. **Outliers:**
There are some outliers with median values above 40 (towards the top of the graph), which suggests that a few homes have significantly higher values, even with an average number of rooms.
Similarly, there is one point around 9 rooms, but its median value is less than $20,000, which could be an anomaly or special case (such as location or condition affecting the price).

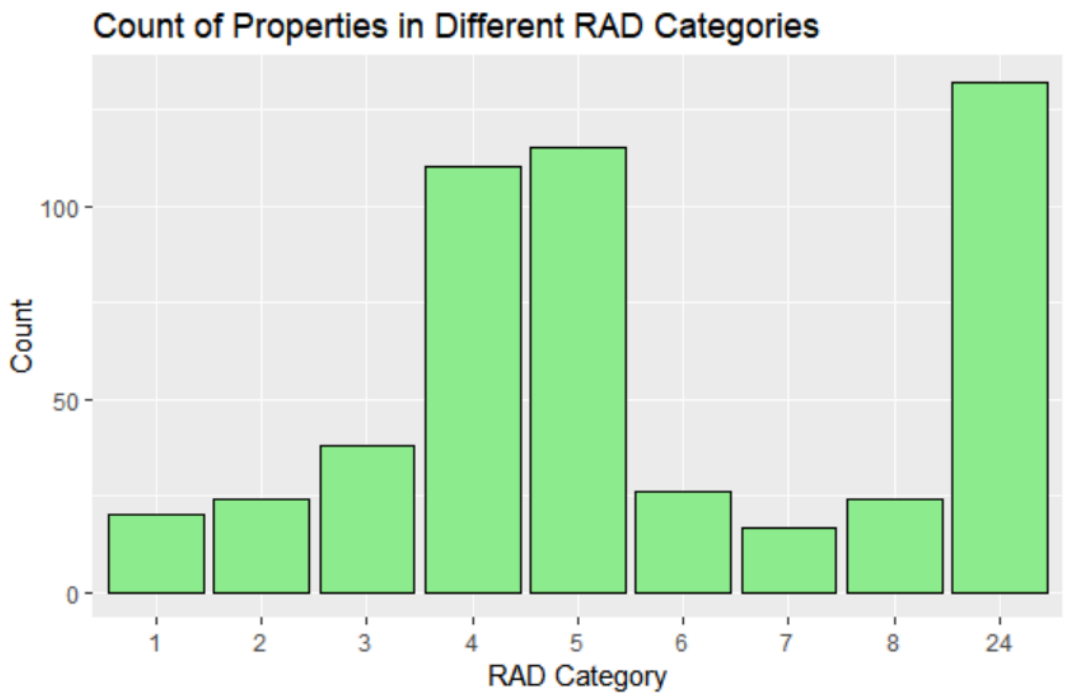
4. **Non-linear Behavior:**
For homes with an average number of rooms greater than 7, there seems to be a plateau in median value around $50,000. While the trend is generally upward, the increase in house prices for these larger homes is not as steep, indicating that the relationship may not be perfectly linear at the high end.

4) Bar Chart

Source Code:

```
#bar chart
ggplot(data = housing_data , aes(x = factor(RAD))) +
geom_bar(fill = "lightgreen",color='black')+
labs(title="Count of Properties in Different RAD Categories",
x = "RAD Category",
y = "Count")
```

Output:



Explanation:

1) **ggplot**(data = Boston, aes(x = factor(RAD))): This line initiates a ggplot object using the Boston dataset. It maps the "RAD" variable to the x-axis, specifying that "RAD" should be treated as a factor. This is important for creating a bar chart where categories are represented on the x-axis.

2) **geom_bar**(fill = "blue", color = "black"): This adds the bar chart layer to the plot. The geom_bar() function creates the bars for each "RAD" category. The fill parameter sets the fill color of the bars to blue, and the color parameter specifies the outline color of the bars as black.

3) **labs**(title = "Count of Properties in Different RAD Categories", x = "RAD Category", y = "Count"): This line is used to add labels to the plot. It sets the title of the chart to "Count of Properties in Different RAD Categories" and labels the x-axis as "RAD Category" and the y-axis as "Count."

Analysis :

- RAD Category 24 has the highest number of properties, followed by RAD Category 5.
- RAD Categories 1, 2, 7, and 8 have the lowest number of properties.
- The overall distribution of properties across RAD categories is skewed to the right, with a larger number of properties in higher RAD categories.
- There is a significant gap between the number of properties in RAD Category 24 and the next highest category (RAD Category 5).

Overall, the graph suggests that properties are concentrated in higher RAD categories, with a small number of properties in lower categories.

Data Visualization with ggplot2

Assignment Questions

1. **You are provided with the following dataset of exam scores for 100 students:**

```
exam_scores <- c (55, 67, 72, 80, 68, 90, 74, 61, 77, 85, 95, 88, 76, 54, 92, 79, 83, 70, 60, 73,81, 65, 78, 93, 56, 82, 75, 66, 64, 89, 84, 87, 71, 69, 59, 62, 63, 57, 53, 58,91, 86, 50, 52, 48, 47, 46, 94, 51, 49, 44, 45, 100, 99, 97, 96, 98, 43, 42, 41,39, 40, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21,20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
```

- Use the ggplot2 package to create a **histogram** for the given exam_scores data.
- Set the bin width to 5.
- Customize the histogram by:

➔ Setting the bar fill color to "blue" and the border color to "black".

➔ Adding a title, and labeling the x-axis as "Exam Scores" and the

➔ y-axis as "Frequency".

➔ Applying a theme of your choice (e.g., theme_minimal()).

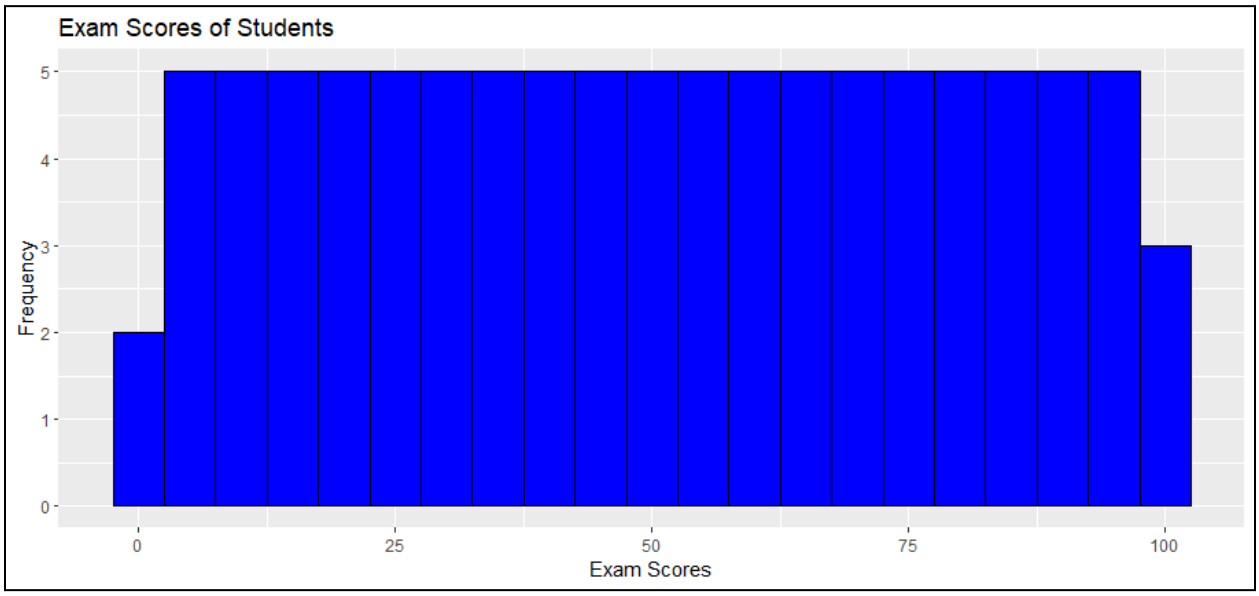
- Based on the histogram, interpret the distribution of exam scores.

Source code:

```
#Creating dataset
exam_scores <- c (55, 67, 72, 80, 68, 90, 74, 61, 77, 85, 95, 88, 76, 54, 92, 79, 83, 70, 60, 73,81, 65, 78, 93, 56, 82, 75, 66, 64, 89, 84, 87, 71, 69, 59, 62, 63, 57, 53, 58,91, 86, 50, 52, 48, 47, 46, 94, 51, 49, 44, 45, 100, 99, 97, 96, 98, 43, 42, 41,39, 40, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21,20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
```

```
exam_scores_df = data.frame(exam_scores)

#Creating Histogram
library("ggplot2")
ggplot(data= exam_scores_df,aes(x = exam_scores)) +
     geom_histogram(binwidth = 5, fill = 'blue',color='black')+
     labs(title="Exam Scores of Students",
          x="Exam Scores",y="Frequency")
```

Output:



Interpretation of the Graph :

- The graph display the distribution of exam scores.
- The range of the data is 0-100.
- Spread of data is nearly even across the board.

2. **You are provided with the following dataset that shows the sales of different products in a store:**

```
product_sales <- data.frame(
product = c("Apples", "Bananas", "Oranges", "Grapes",
"Pineapples"),
sales = c(150, 200, 120, 180, 90))
```

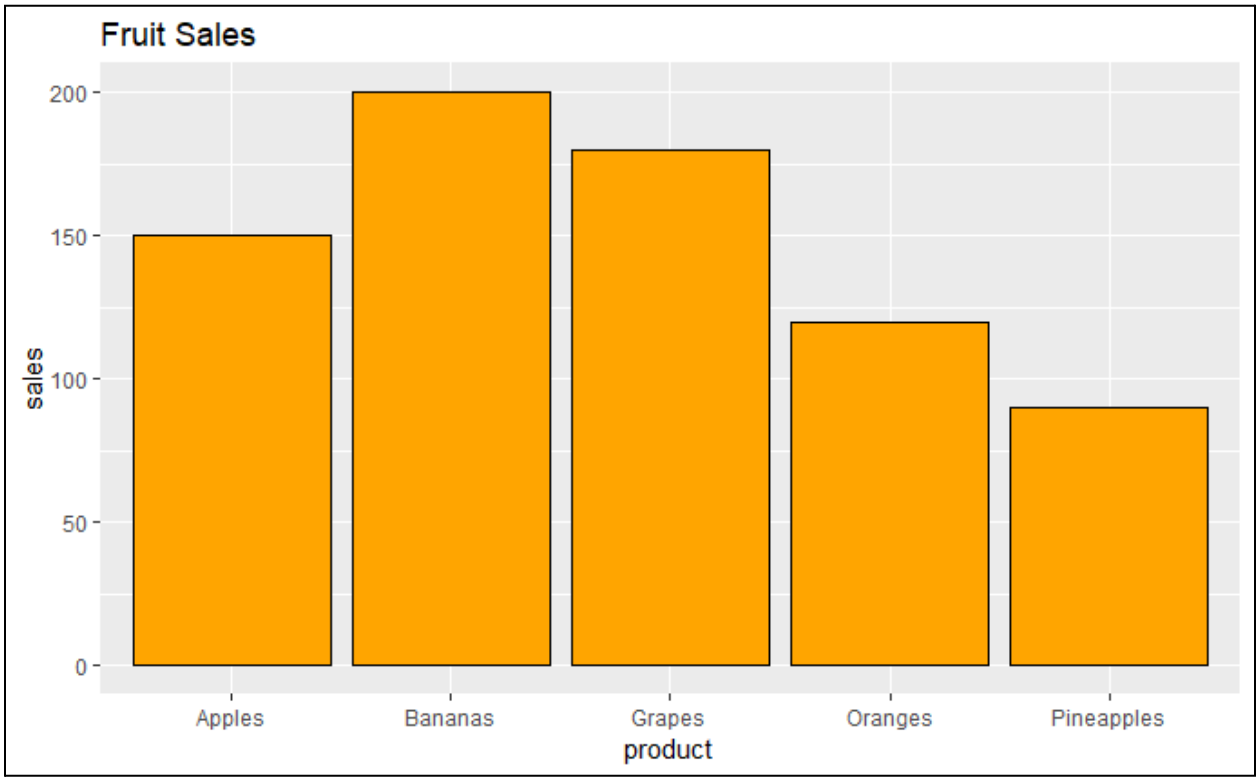
Task:

1. Use the ggplot2 package to create a bar chart showing the sales for each product.
2. Customize the bar chart by:
   - Setting the bar fill color to "orange" and the border color to "black".
   - Adding a title, and labeling the x-axis as "Product" and the y-axis as "Sales".
   - Adjusting the bar width or alignment if necessary.
   - Applying a theme of your choice (e.g., theme_classic(), theme_minimal(), etc.).
3. Based on the bar chart, identify which product has the highest and lowest sales.

Source Code:

```
product_sales <- data.frame(
product=c("Apples", "Bananas", "Oranges", "Grapes",
"Pineapples"), sales = c(150, 200, 120, 180, 90)
)

ggplot(data = product_sales ,mapping = aes(x=product,y=sales)) +
     geom_col(fill='orange',color='black')+
     labs(title = "Fruit Sales" , )+
     theme(theme_classic())
```

Output:



Analysis:

- Banana is the highest selling fruit in the data with 200 sales.
- Pineapples is the least selling fruit in the data.
- The distribution is skewed to left with banana and grapes having most sales.

3. **You are provided with a dataset showing the average temperatures (in degrees Celsius) over 12 months of the year. Download the dataset using the link below and create a line chart to visualize the temperature trends.**

   Dataset Download Link:
   Download Temperature Dataset
   (https://www.kaggle.com/datasets/sudalairajkumar/daily-temperature-of-major-cities)

   Dataset Columns:
   - month: The month of the year (January to December).
   - temperature: The average temperature in degrees Celsius for each month.

   Task:
   - Download and load the dataset into R.
   - Use the read.csv() function to load the dataset into R.
   - Create a line chart using the ggplot2 package to visualize the trend of average temperatures over the months.

   - Customize the line chart by:

   ➔ Changing the line color to "red" and adjusting the line width 2.

   ➔ Adding points to the line chart using geom_point() with the following options:
   Set the point size to 4.
   Use a point shape of your choice.

   - Adding labels and title:

   ➔ Title the chart as "Average Monthly Temperatures".

   ➔ Label the x-axis as "Month" and the y-axis as "Temperature (°C)".

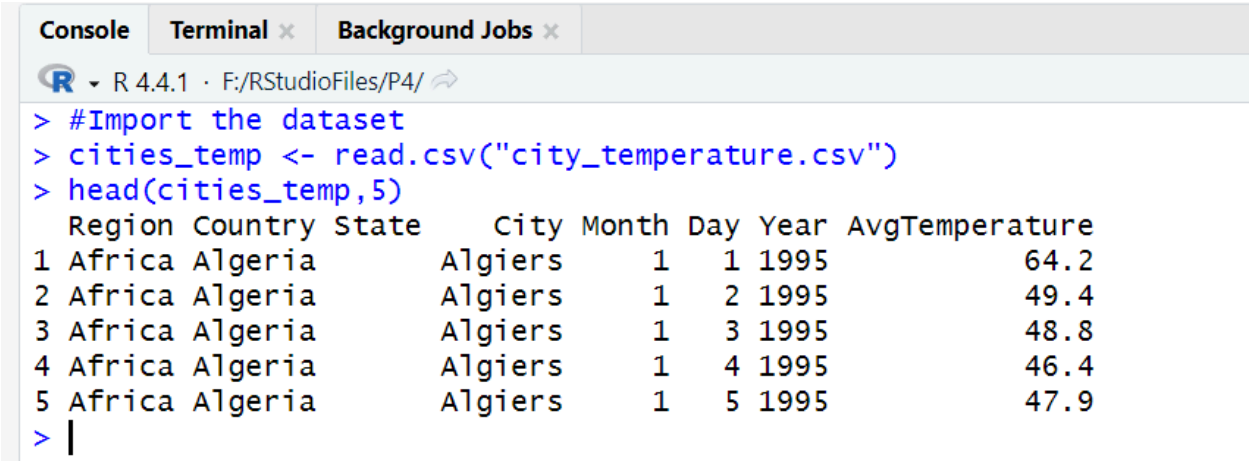
- Rotating the x-axis labels for better readability.
- Applying a suitable theme like theme_classic() or theme_minimal().
- Adding gridlines for better clarity.

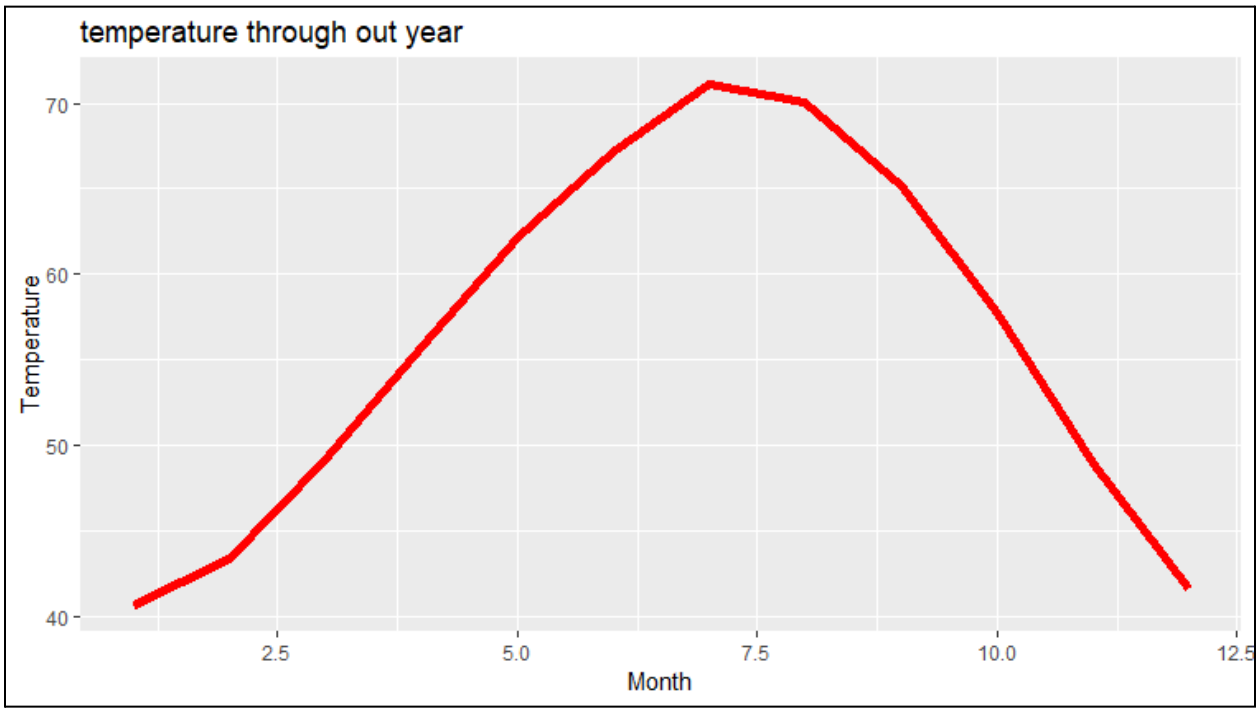
- Interpret the line chart by describing the temperature trends throughout the year.

Source Code:

```
#Import the dataset
cities_temp <- read.csv("city_temperature.csv")
head(cities_temp,5)

#grouping and averaging the temperature by month
city_temp      <-      aggregate(cities_temp$AvgTemperature      ~
cities_temp$Month ,data =cities_temp , FUN = mean)
print(city_temp)

colnames(city_temp)<- c("Month","Temperature")
city_temp

ggplot(data = city_temp , aes(x=Month,y=Temperature))+
geom_line(color = 'red' ,size = 2)+
labs(title = "temperature through out year")
```

Output:





Analysis :

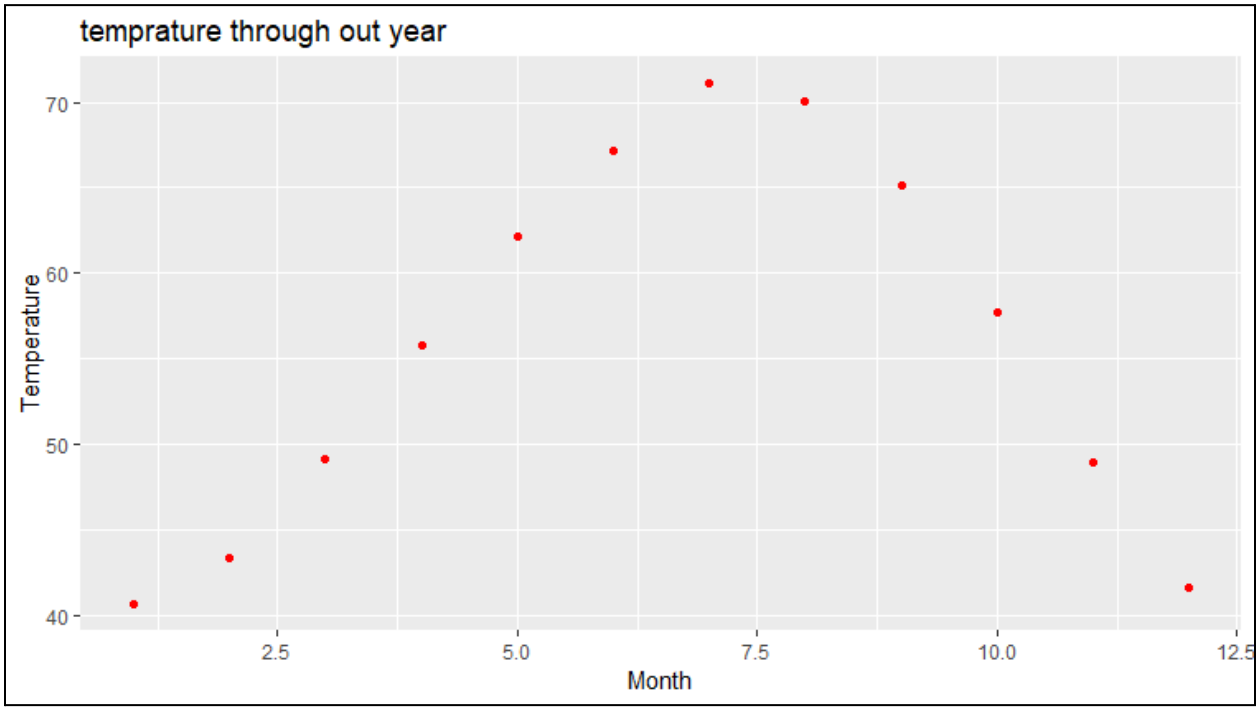
- **Seasonal Pattern:** The graph shows a clear seasonal pattern, with temperatures rising from the beginning of the year (around January) to a peak in the middle of the year (around July), and then declining towards the end of the year (around December).
- **Maximum Temperature:** July is when the maximum temperature is recorded, at roughly 72 degrees.
- **Lowest Temperature:** January typically records the lowest temperature, which is about 42 degrees.
- **Temperature Range:** The entire temperature range is roughly 30 degrees.
- **Variations in Temperature:** Throughout the year, there are comparatively mild variations in temperature, with no abrupt ups or downs.

4. Create Scatter plot of the same dataset as the one used for line chart.

Source Code:

```
#Plotting Scatterplot
ggplot(data = city_temp, mapping =aes(x = Month,y = Temperature)) +
geom_point(color = "red") +
labs(title = "temprature through out year")
```

Output:



Analysis :

- **Seasonal Pattern**: The graph shows a general seasonal pattern, with temperatures increasing from the beginning of the year (around January) to a peak in the middle of the year (around July), and then decreasing towards the end of the year (around December). However, the pattern is not as smooth as in the previous graph, with more fluctuations and variations.
- **Highest Temperature:** The highest temperature is recorded around July, with a value of approximately 72 degrees.
- **Lowest Temperature:** The lowest temperature is recorded around January, with a value of approximately 42 degrees.
- **Temperature Range:** The overall range of temperatures is approximately 30 degrees.
- **Temperature Fluctuations:** The temperature fluctuations throughout the year are more pronounced than in the previous graph, with several spikes and drops.

This suggests that the region may experience more variable weather conditions. Overall, the graph suggests that the region experiences a general seasonal pattern with hot summers and cold winters. However, the temperature fluctuations are more significant compared to the previous graph, indicating that the weather may be less predictable.

Conclusion:

From this practical I learned Implementation and use of data visualization using the ggplot2 library.

<table>
<tr><td colspan="4">Name of Student: Shaikh Farhan Ahmed</td></tr>
<tr><td colspan="2">Roll Number: 50</td><td colspan="2">LAB Assignment Number: 5</td></tr>
<tr><td colspan="4">Title of LAB Assignment: Implementation and analysis of Apriori Algorithm using Market Basket Analysis.</td></tr>
<tr><td colspan="2">DOP: 25 – 10 – 2024</td><td colspan="2">DOS: 30 – 10 – 2024</td></tr>
<tr><td>CO Mapped:
CO3</td><td>PO Mapped:
PO1, PO2, PO3, PO5, PO6, PO7, PSO1, PSO2</td><td>Faculty Signature :</td><td>Marks:</td></tr>
</table>

Practical No 5

Aim: Implementation and analysis of Apriori Algorithm using Market Basket Analysis.

Theory:

The Apriori algorithm is an unsupervised machine learning algorithm used for association rule learning. Association rule learning is a data mining technique that identifies frequent patterns, connections and dependencies among different groups of items called itemsets in data.

The Apriori algorithm uses the **support**, **confidence**, and lift metrics to define its operating criteria and improve performance efficiency.

Support

Support is defined as the ratio of the number of times an item occurs in the transactions to the total number of transactions. This metric thus defines the probability of the occurrence of each individual item in the transactions. The same logic can be extended to itemsets.

$$S(IA) = Occ(IA) / TotalTransactions$$

where I_A is item A, $Occ(I_A)$ is the number of occurrences of item A, and $S(I_A)$ = support of item A

Confidence

The confidence metric identifies the probability of items or itemsets occurring in the itemsets together. For example, if there are two items in a transaction, the existence of one item is assumed to lead to the other. The first item or itemset is the antecedent, and the second is the consequent.

$$C(A,B) = Occ(A \cap B) / Occ(A)$$

where *A* is the antecedent, *B* is the consequent, and *C(A,B)* is the confidence that *A* leads to *B.*

Source Code:

```
#setting the working directory

setwd("F:/RStudioFiles/P5")

#Loading and Displaying Data

dataset <- read.csv("data_apriori.csv")

transaction <- split(dataset$Products, dataset$Customer_Id,
"transactions")

head(transaction)

#installing the library required for apriori

install.packages('arules')

ibrary(arules)

rules = apriori(transaction, parameter = list(support = 0.5,
confidence = 0.9, maxlen = 3, minlen = 2))

#Visualizing

inspect(rules)
```

Output:

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P5/
Attaching package: 'arules'

The following objects are masked from 'package:base':

    abbreviate, write

> rules = apriori(transaction, parameter = list(support = 0.5, confidence = 0.9, maxlen = 3, minlen = 2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.9    0.1    1 none FALSE            TRUE       5     0.5      2      3  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 7

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[15 item(s), 15 transaction(s)] done [0.00s].
sorting and recoding items ... [4 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [11 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
Warning messages:
1: In asMethod(object) : removing duplicated items in transactions
2: In apriori(transaction, parameter = list(support = 0.5, confidence = 0.9,  :
  Mining stopped (maxlen reached). Only patterns up to a length of 3 returned!
>
```

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P5/
> #Visualizing
> inspect(rules)
     lhs                rhs       support   confidence coverage  lift     count
[1]  {eggs}          => {milk}    0.6000000 1          0.6000000 1.666667 9
[2]  {milk}          => {eggs}    0.6000000 1          0.6000000 1.666667 9
[3]  {butter}        => {bread}   0.6000000 1          0.6000000 1.250000 9
[4]  {butter, eggs}  => {milk}    0.5333333 1          0.5333333 1.666667 8
[5]  {butter, milk}  => {eggs}    0.5333333 1          0.5333333 1.666667 8
[6]  {bread, eggs}   => {milk}    0.5333333 1          0.5333333 1.666667 8
[7]  {bread, milk}   => {eggs}    0.5333333 1          0.5333333 1.666667 8
[8]  {butter, eggs}  => {bread}   0.5333333 1          0.5333333 1.250000 8
[9]  {bread, eggs}   => {butter}  0.5333333 1          0.5333333 1.666667 8
[10] {butter, milk}  => {bread}   0.5333333 1          0.5333333 1.250000 8
[11] {bread, milk}   => {butter}  0.5333333 1          0.5333333 1.666667 8
>
```

Apriori Algorithm Implementation on custom dataset.

#Dataset Link

https://github.com/okzapradhana/simple-apriori-algorithm/blob/master/dataset_apriori.csv

Source Code:

```
#setting the working directory

setwd("F:/RStudioFiles/P5")

#Loading and Displaying Data

apriori_ds <- read.csv("dataset_apriori.csv")

transaction <- split(apriori_ds$items,"transactions")

head(transaction)

#installing the library required for apriori

install.packages('arules')

library(arules)

rules = apriori(transaction, parameter = list(support = 0.5,
confidence = 0.8, maxlen = 3, minlen = 2 ))

#Visualizing the first 20 association rules.

inspect(rules[1:20])
```

Output:

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P5/
> rules = apriori(transaction, parameter = list(support = 0.5,confidence = 0.8,maxlen = 3, minlen = 2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.8    0.1    1 none FALSE            TRUE       5     0.5      2      3  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 0

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[10 item(s), 1 transaction(s)] done [0.00s].
sorting and recoding items ... [10 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [450 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
Warning message:
In apriori(transaction, parameter = list(support = 0.5, confidence = 0.8,  :
  Mining stopped (maxlen reached). Only patterns up to a length of 3 returned!
>
```

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P5/
> inspect(rules[1:20])
     lhs                                    rhs                                      support confidence coverage lift
[1]  {Charger,Handphone}                 => {Handphone,Charger,Laptop}               1       1          1        1
[2]  {Handphone,Charger,Laptop}          => {Charger,Handphone}                      1       1          1        1
[3]  {Charger,Handphone}                 => {Handphone,Charger,Tablet}               1       1          1        1
[4]  {Handphone,Charger,Tablet}          => {Charger,Handphone}                      1       1          1        1
[5]  {Charger,Handphone}                 => {Handphone,Laptop}                       1       1          1        1
[6]  {Handphone,Laptop}                  => {Charger,Handphone}                      1       1          1        1
[7]  {Charger,Handphone}                 => {Handphone,Laptop,Tablet,Charger}        1       1          1        1
[8]  {Handphone,Laptop,Tablet,Charger}   => {Charger,Handphone}                      1       1          1        1
[9]  {Charger,Handphone}                 => {Handphone,Powerbank}                    1       1          1        1
[10] {Handphone,Powerbank}               => {Charger,Handphone}                      1       1          1        1
[11] {Charger,Handphone}                 => {Laptop,Charger,Powerbank}               1       1          1        1
[12] {Laptop,Charger,Powerbank}          => {Charger,Handphone}                      1       1          1        1
[13] {Charger,Handphone}                 => {Powerbank,Laptop,Charger,Handphone}     1       1          1        1
[14] {Powerbank,Laptop,Charger,Handphone} => {Charger,Handphone}                     1       1          1        1
[15] {Charger,Handphone}                 => {Tablet,Laptop,Handphone}                1       1          1        1
[16] {Tablet,Laptop,Handphone}           => {Charger,Handphone}                      1       1          1        1
[17] {Charger,Handphone}                 => {Tablet,Powerbank}                       1       1          1        1
[18] {Tablet,Powerbank}                  => {Charger,Handphone}                      1       1          1        1
[19] {Handphone,Charger,Laptop}          => {Handphone,Charger,Tablet}               1       1          1        1
[20] {Handphone,Charger,Tablet}          => {Handphone,Charger,Laptop}               1       1          1        1
```

Conclusion:

From this practical, I have learned how to implement the apriori algorithm in R.

<table>
<tr><td colspan="4">Name of Student: Shaikh Farhan Ahmed</td></tr>
<tr><td colspan="2">Roll Number: 50</td><td colspan="2">LAB Assignment Number: 6</td></tr>
<tr><td colspan="4">Title of LAB Assignment: Implementation and analysis of Linear regression through graphical methods including plots.</td></tr>
<tr><td colspan="2">DOP: 25 – 10 – 2024</td><td colspan="2">DOS: 12 – 11 – 2024</td></tr>
<tr><td>CO Mapped:
CO4</td><td>PO Mapped:
PO3, PO4, PO5, PO6, PO7, PO8
PSO1, PSO2</td><td>Faculty Signature :</td><td>Marks:</td></tr>
</table>

Practical no 6

Aim: Implementation and analysis of Linear regression through graphical methods.

Theory:

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc. Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called linear regression.
Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable. Steps to Establish a Regression A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between the height and weight of a person.

The steps to create the relationship is

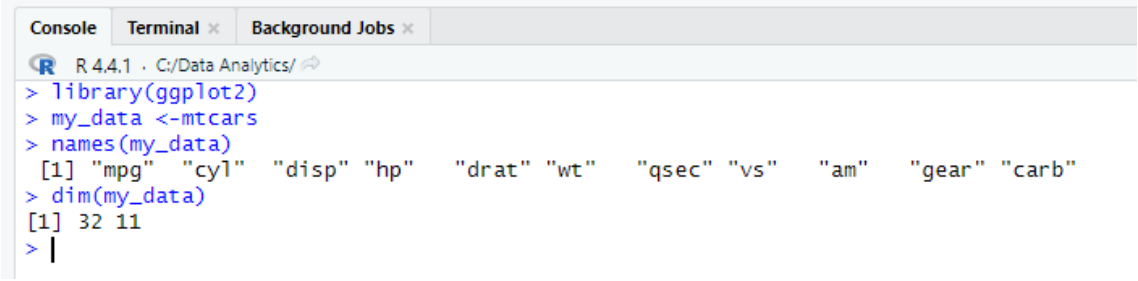
• Carry out the experiment of gathering a sample of observed values of height and corresponding weight. • Create a relationship model using the lm() functions in R.
• Find the coefficients from the model created and create the mathematical equation using these • Get a summary of the relationship model to know the average error in prediction. Also called residuals.
• To predict the weight of new persons, use the predict() function in R.

Source Code:

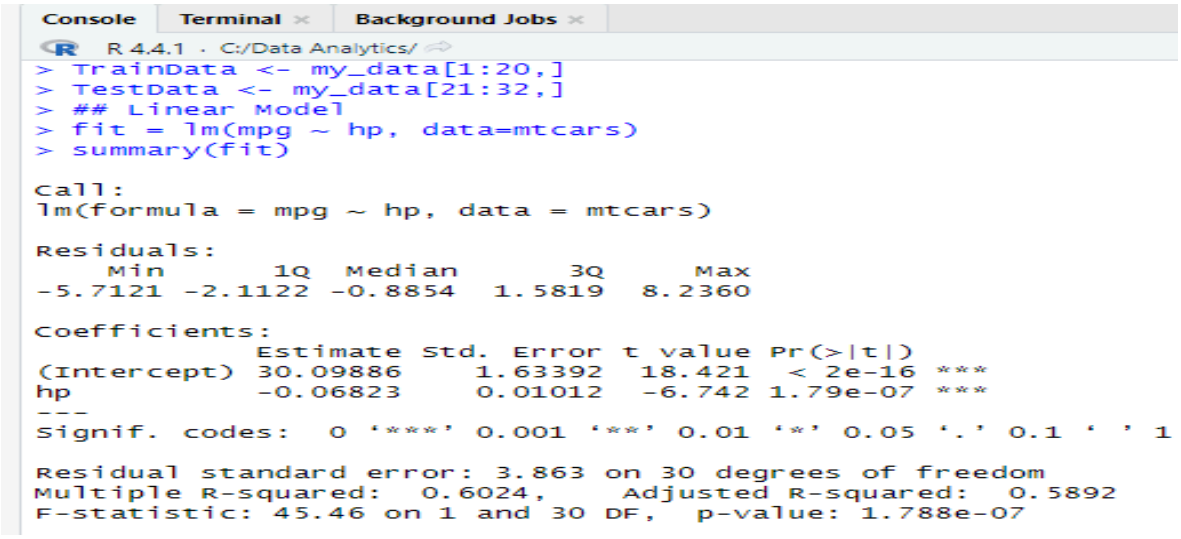
```
library(ggplot2)
my_data <-mtcars
names(my_data)

dim(my_data)

#randomize
my_data <- my_data[sample(nrow(my_data), ), ]
head(my_data)

#Training Data
TrainData <- my_data[1:20,]
TestData <- my_data[21:32,]
## Linear Model
fit = lm(mpg ~ hp, data=mtcars)
summary(fit)

preds <- predict(fit, newdata = TestData)
df1 <- data.frame(preds,TestData$mpg)
head(df1)

#correlation
cor(preds,TestData$mpg)

#plotting
plot(mtcars$hp, mtcars$mpg)

ggplot(fit, aes(hp, mpg)) +
geom_point() +
stat_smooth(method = lm, se = FALSE) +
geom_segment(aes(xend = hp, yend = .fitted), color = "red",
size = 0.3)

lmmodel1 <- lm(mpg ~ hp+cyl+gear+wt, data = TrainData)
summary(lmmodel1)

preds_new <- predict(lmmodel1, newdata = TestData)
df2 <- data.frame(preds_new,TestData$mpg)
```

```
head(df2)
#plotting
plot(mtcars$hp+mtcars$cyl+mtcars$gear+mtcars$wt, mtcars$mpg)

ggplot(fit, aes(mtcars$hp+mtcars$cyl+mtcars$gear+mtcars$wt,mpg))
+ geom_point() + stat_smooth(method = lm, se = FALSE) +
geom_segment(aes(xend = hp, yend = .fitted),
color = "red", size = 0.3)
```

Output:



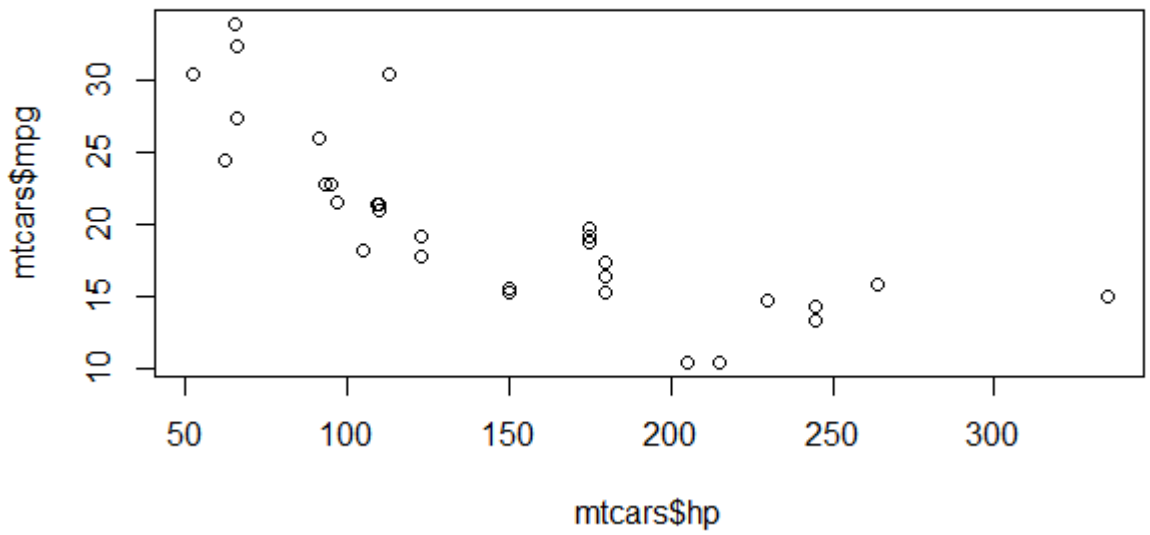

```
> #randomize
> my_data <- my_data[sample(nrow(my_data), ), ]
> head(my_data)
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 280         19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
Merc 450SL       17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Toyota Corona    21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Volvo 142E       21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
Merc 240D        24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
>
```

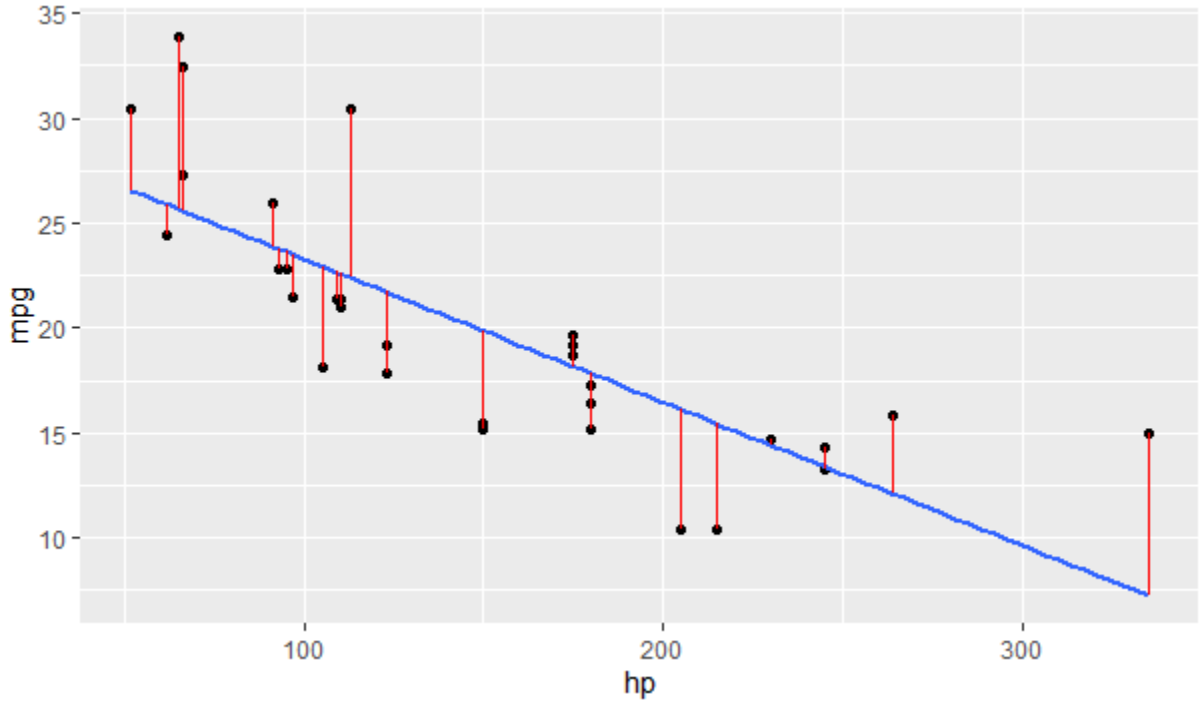


```
Console  Terminal  Background Jobs
R 4.4.1 · C:/Data Analytics/
> preds <- predict(fit, newdata = TestData)
> df1 <- data.frame(preds,TestData$mpg)
> head(df1)
                      preds TestData.mpg
Maserati Bora      7.242387          15.0
Mazda RX4         22.593750          21.0
Cadillac Fleetwood 16.112064          10.4
Merc 450SE        17.817770          16.4
Merc 450SLC       17.817770          15.2
Fiat 128          25.595794          32.4
>
```

```
Console  Terminal  Background Jobs
R 4.4.1 · C:/Data Analytics/
> #correlation
> cor(preds,TestData$mpg)
[1] 0.7694323
> plot(mtcars$hp, mtcars$mpg)
>
```

Plot:





Console Terminal Background Jobs

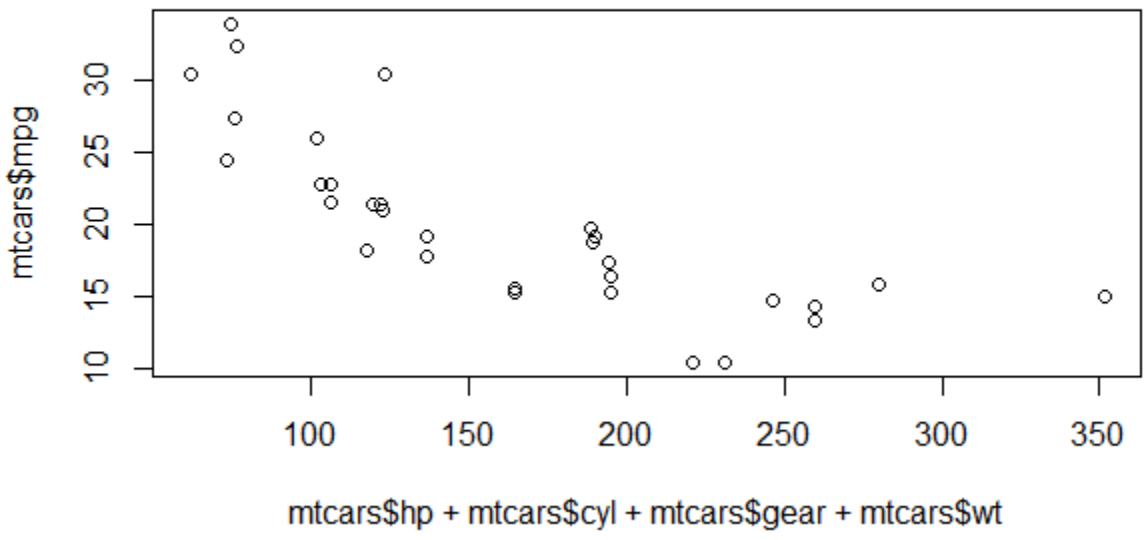
R 4.4.1 · C:/Data Analytics/

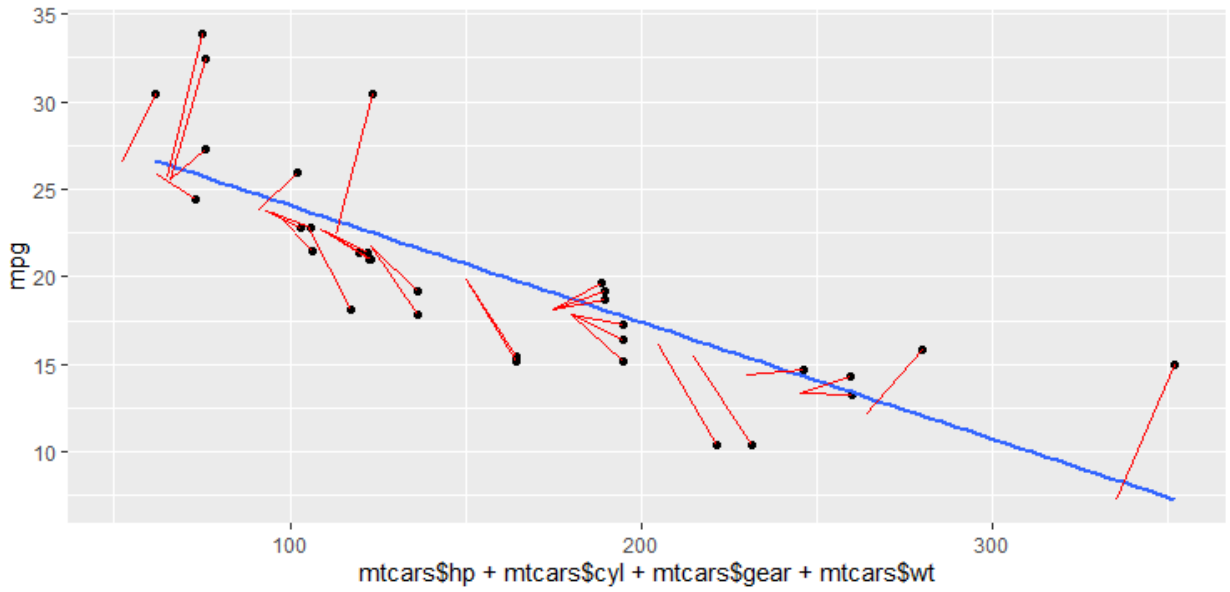
```
> # BetterModel ?
> lmmodel1 <- lm(mpg ~ hp+cyl+gear+wt, data = TrainData)
> summary(lmmodel1)

Call:
lm(formula = mpg ~ hp + cyl + gear + wt, data = TrainData)

Residuals:
    Min      1Q  Median      3Q     Max
-2.2011 -1.1954 -0.2617  1.4126  2.2302

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 34.00518    4.80565   7.076 3.77e-06 ***
hp          -0.03483    0.01498  -2.326  0.03447 *
cyl         -0.08802    0.55340  -0.159  0.87574
gear         0.85758    0.77223   1.111  0.28426
wt          -3.85672    0.97952  -3.937  0.00132 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.713 on 15 degrees of freedom
Multiple R-squared:  0.9005,    Adjusted R-squared:  0.874
F-statistic: 33.95 on 4 and 15 DF,  p-value: 2.354e-07

>
```

```
Console  Terminal  Background Jobs
R 4.4.1 · C:/Data Analytics/
> preds_new <- predict(lmmodel1, newdata = TestData)
> df2 <- data.frame(preds_new,TestData$mpg)
> head(df2)
                   preds_new TestData.mpg
Maserati Bora      12.151722         15.0
Mazda RX4          22.971251         21.0
Cadillac Fleetwood  8.485419         10.4
Merc 450SE         13.907143         16.4
Merc 450SLC        15.025591         15.2
Fiat 128           26.299727         32.4
>
```

```
Console  Terminal  Background Jobs
R 4.4.1 · C:/Data Analytics/
> cor(preds_new,TestData$mpg)
[1] 0.9175391
> plot(mtcars$hp+mtcars$cyl+mtcars$gear+mtcars$wt, mtcars$mpg)
>
```

Plot:




35
30
25
20
15
10
mpg
100
200
300
mtcars$hp + mtcars$cyl + mtcars$gear + mtcars$wt

Finding Linear Regression Between Student Study Hours and Student Scores.

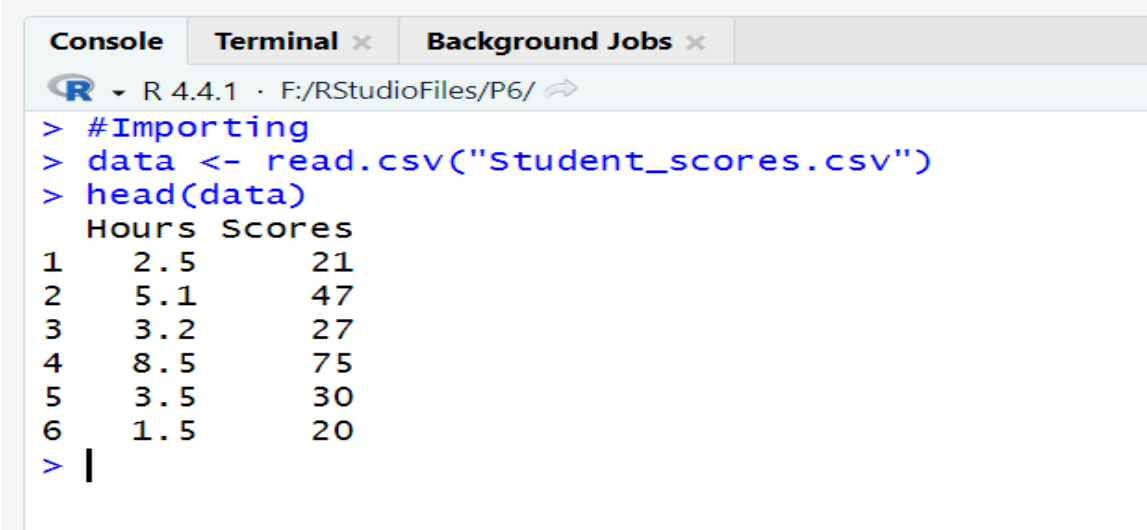
Dataset:
https://raw.githubusercontent.com/AdiPersonalWorks/Random/master/student_scores%20-%20student_scores.csv

Importing the dataset and loading ggplot2 library:

Source Code:

```
#Importing
data <- read.csv("Student_scores.csv")
head(data)
```

Output:

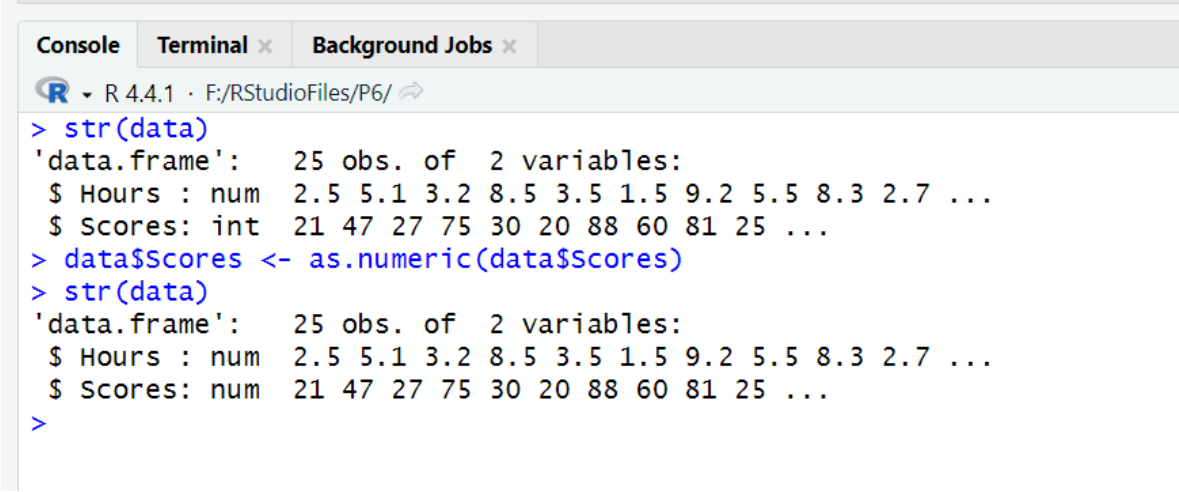


Checking the datatypes and converting them into numeric if needed.

Source Code:

```
str(data)
data$Scores <- as.numeric(data$Scores)
str(data)
```

Output:

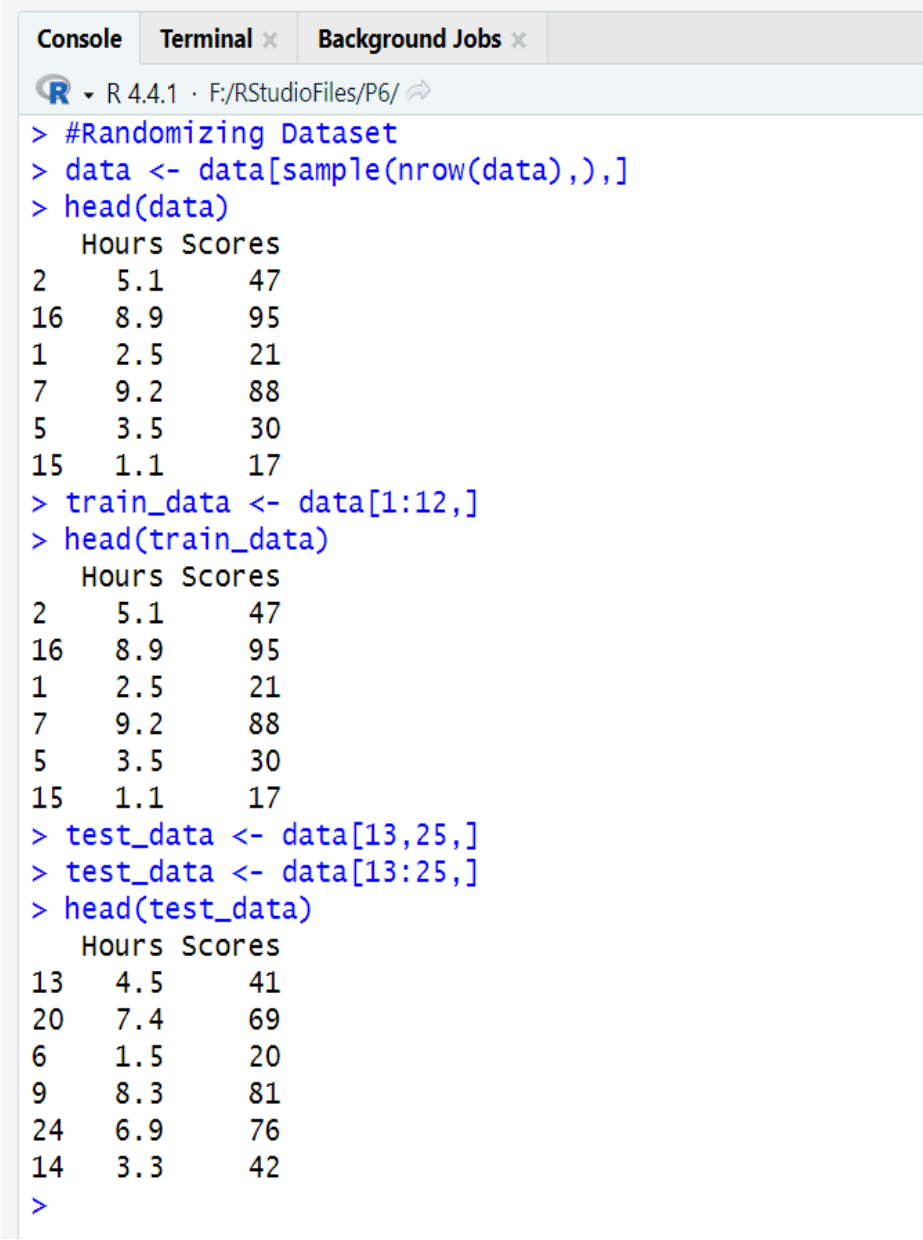


Randomizing the dataset and splitting it into Train data and Test data

Source Code:

```
#Randomizing Dataset
data <- data[sample(nrow(data),),]
head(data)

train_data <- data[1:12,]
head(train_data)

test_data <- data[1:12,]
head(test_data)
```

Output:


Console
Terminal
Background Jobs
R 4.4.1 · F:/RStudioFiles/P6/
> #Randomizing Dataset
> data <- data[sample(nrow(data),),]
> head(data)
Hours Scores
2 5.1 47
16 8.9 95
1 2.5 21
7 9.2 88
5 3.5 30
15 1.1 17
> train_data <- data[1:12,]
> head(train_data)
Hours Scores
2 5.1 47
16 8.9 95
1 2.5 21
7 9.2 88
5 3.5 30
15 1.1 17
> test_data <- data[13,25,]
> test_data <- data[13:25,]
> head(test_data)
Hours Scores
13 4.5 41
20 7.4 69
6 1.5 20
9 8.3 81
24 6.9 76
14 3.3 42
>

Creating Linear Model Using training data:

Source Code:

```
library(ggplot2)
fit <- lm(Hours ~ Scores, data = data)
summary(fit)
```

Output:

```
Console   Terminal   Background Jobs
R 4.4.1 · F:/RStudioFiles/P6/
> library(ggplot2)
> fit <- lm(Hours ~ Scores, data = data)
> summary(fit)

Call:
lm(formula = Hours ~ Scores, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-0.7879 -0.4433 -0.2181  0.5096  1.1953

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.006286   0.258038  -0.024    0.981
Scores       0.097480   0.004517  21.583   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5595 on 23 degrees of freedom
Multiple R-squared:  0.9529,    Adjusted R-squared:  0.9509
F-statistic: 465.8 on 1 and 23 DF,  p-value: < 2.2e-16

> |
```

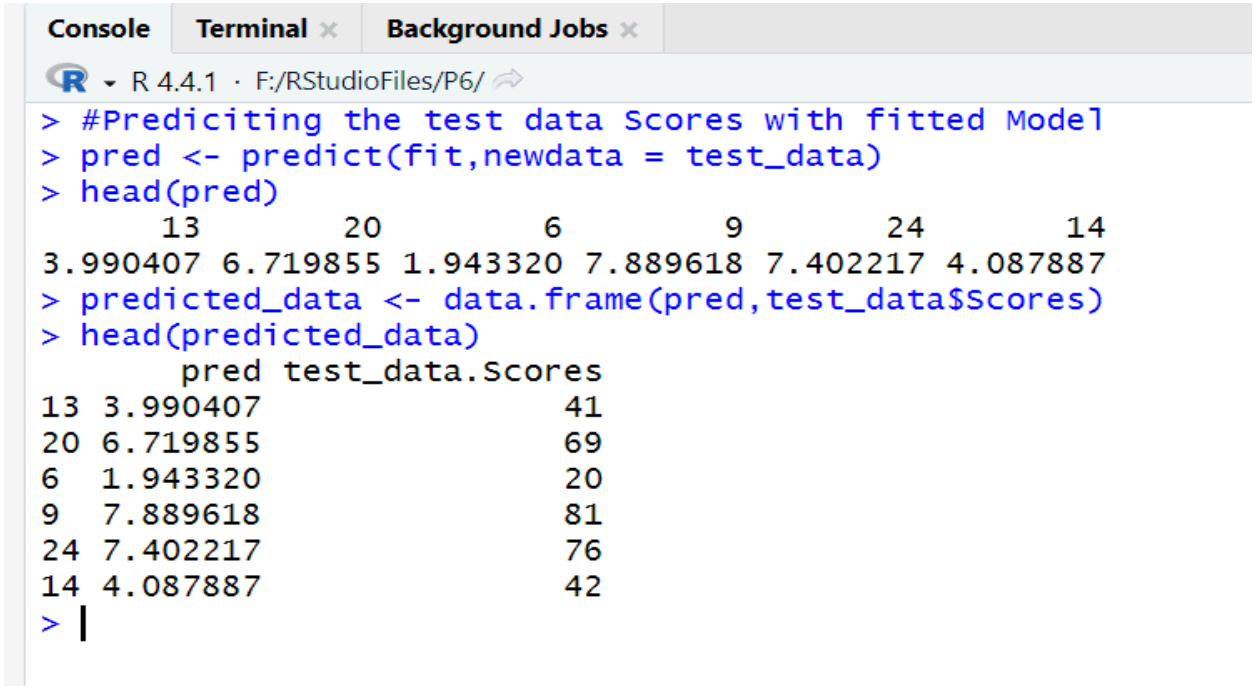
Predicting the test data with model:

Source Code:

```
#Prediciting the test data Scores with fitted Model
pred <- predict(fit,newdata = test_data)
head(pred)
```

```
predicted_data <- data.frame(pred,test_data$Scores)
head(predicted_data)
```

Output:

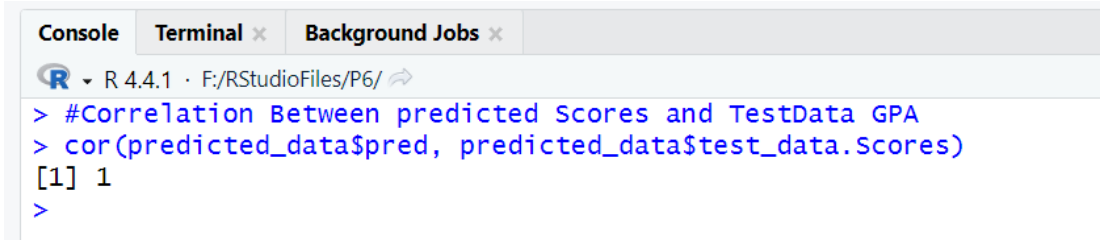


Correlation:

Source Code:

```
#Correlation Between predicted Scores and TestData GPA
cor(predicted_data$pred,
predicted_data$test_data.Scores)
```

Output:



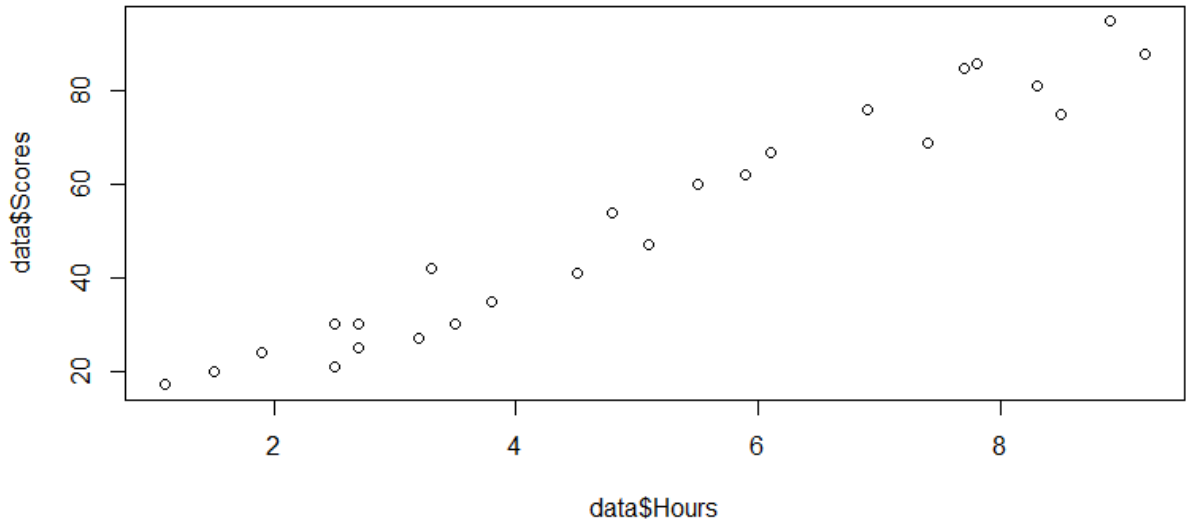
Plotting Graph and Regression Curve:

Graph:

Source Code:

```
#plotting the graph for model
 plot(data$Hours,data$Scores)
```

Output:

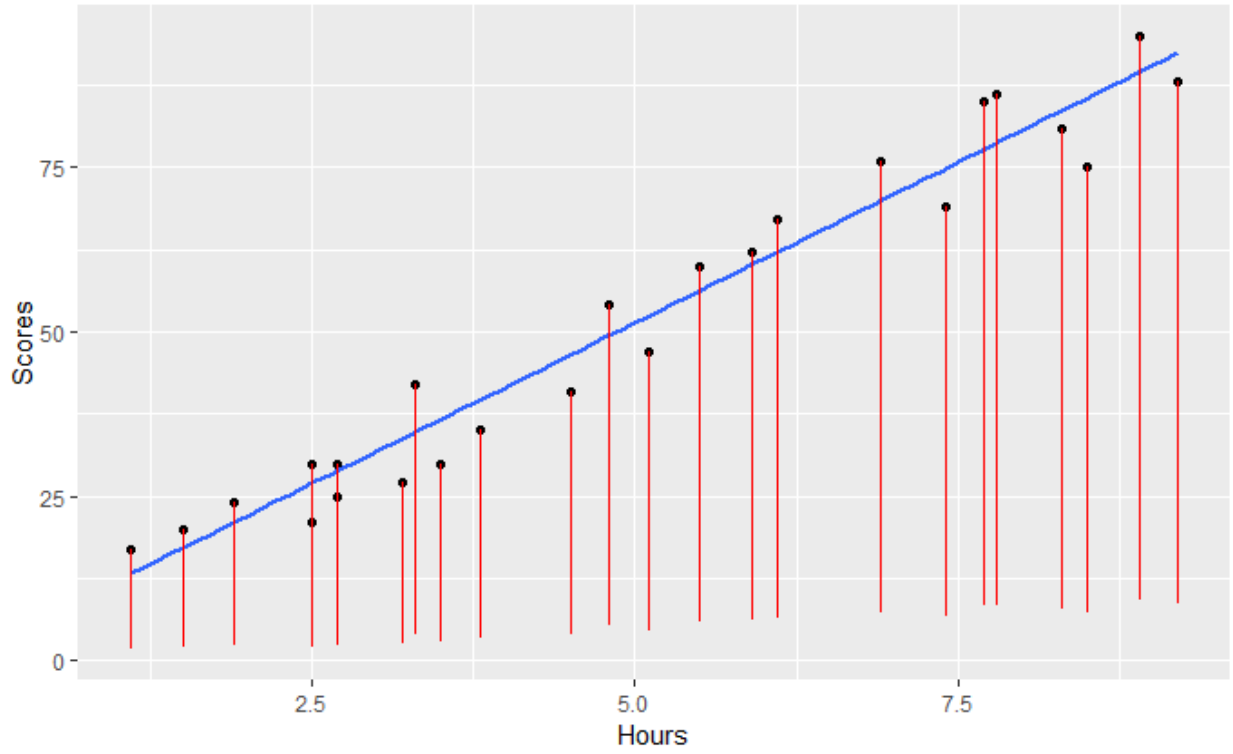


Regression Curve:

Source Code:

```
ggplot(fit, aes(x = Hours, y = Scores)) +
    geom_point() +  # Scatter plot points
    geom_smooth(method = "lm", se = FALSE) +
    geom_segment(aes(xend = Hours, yend = .fitted),
    color = "red", size = 0.3)
```

Output:



Conclusion:

Learned to implement linear regression through graphical methods.

<table>
<tr><td colspan="4">Name of Student: Shaikh Farhan Ahmed</td></tr>
<tr><td colspan="2">Roll Number: 50</td><td colspan="2">LAB Assignment Number: 7</td></tr>
<tr><td colspan="4">Title of LAB Assignment: Implementation and analysis of Classification algorithms like Naive Bayesian, K-Nearest Neighbor, ID3, C4.5.</td></tr>
<tr><td colspan="2">DOP: 25 – 10 – 2024</td><td colspan="2">DOS: 14 – 11 – 2024</td></tr>
<tr><td>CO Mapped:
CO4</td><td>PO Mapped:
PO1,PO2,PO3, PO4, PO5, PO6, PO7, PO8,
PSO1, PSO2</td><td>Faculty Signature :</td><td>Marks:</td></tr>
</table>

Practical No 7

Aim: Implementation and analysis of Classification algorithms like Naive Bayesian, K-Nearest Neighbor, ID3, C4.5.

Theory:

This article discusses the theory behind the Naive Bayes classifiers and their implementation.

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

To start with, let us consider a dataset.

Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for plaing golf.

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

P(A|B) = \frac{P(B|A) P(A)}{P(B)}

where A and B are events and P(B) ? 0.

Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.

P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen).

The evidence is an attribute value of an unknown instance(here, it is event B).

P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

Now, with regards to our dataset, we can apply Bayes' theorem in following way:

P(y|X) = \frac{P(X|y) P(y)}{P(X)}

K nearest:

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

R Packages:

1. tidyr: The word tidyr comes from the word tidy, which means clear. tidyr package is used to make the data' tidy'

2. ggplot2: R provides the ggplot package for creating graphics declaratively. This package is famous for its elegant and quality graphs which sets it apart from other visualization packages.

3. dplyr: R provides the dplyr library for performing data wrangling and data analysis.This library facilitates several functions for the data frame in R.

4. caret: R allows us to perform classification and regression tasks by providing the caret package. CaretEnsemble is a feature of caret which is used for the combination of different models.

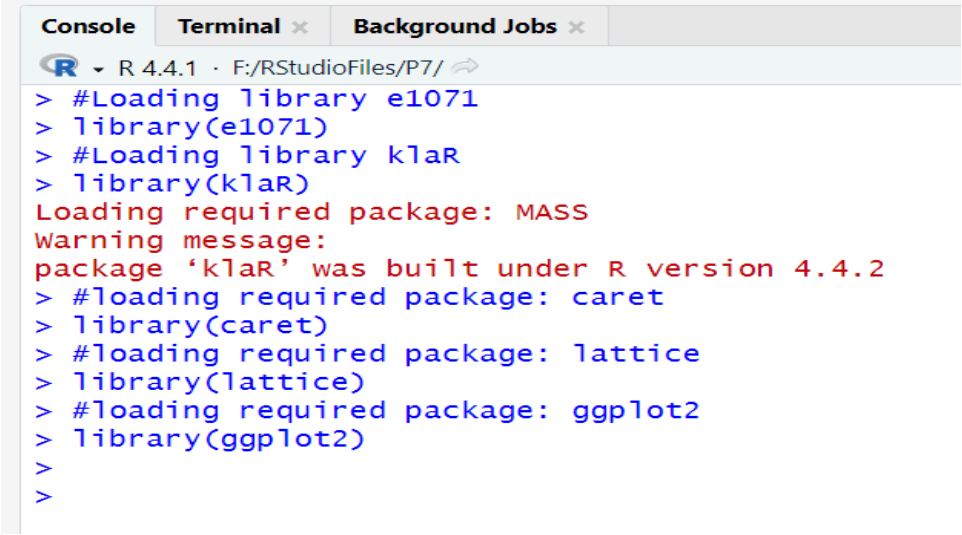
5. e1071: The e1071 library provides useful functionsessential for data analysis like Naive Bayes, Fourier Transforms, SVMs, Clustering, and other miscellaneous functions.

Installing and Loading Libraries:

Source Code:

```
install.packages("e1071")

install.packages("klaR")

install.packages("caret")

install.packages("lattice)

#Loading library e1071

library(e1071)

#Loading library klaR

library(klaR)

#loading required package: caret

library(caret)

#loading required package: lattice

library(lattice)

#loading required package: ggplot2

library(ggplot2)
```

Output:

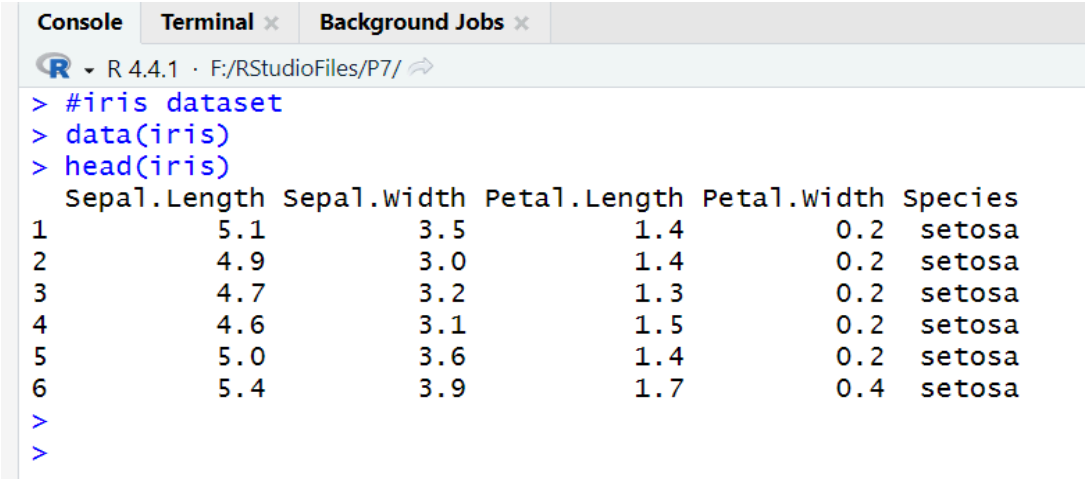


Printing Dataset:

Source Code:

```
#iris dataset

data(iris)

head(iris)
```

Output:

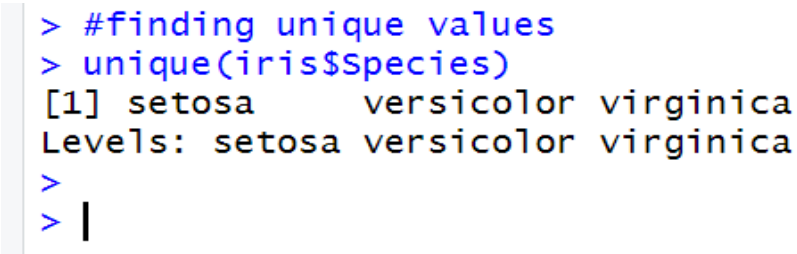


Printing Unique Elements:

Source Code:

```
#finding unique values

unique(iris$Species)
```

Output:

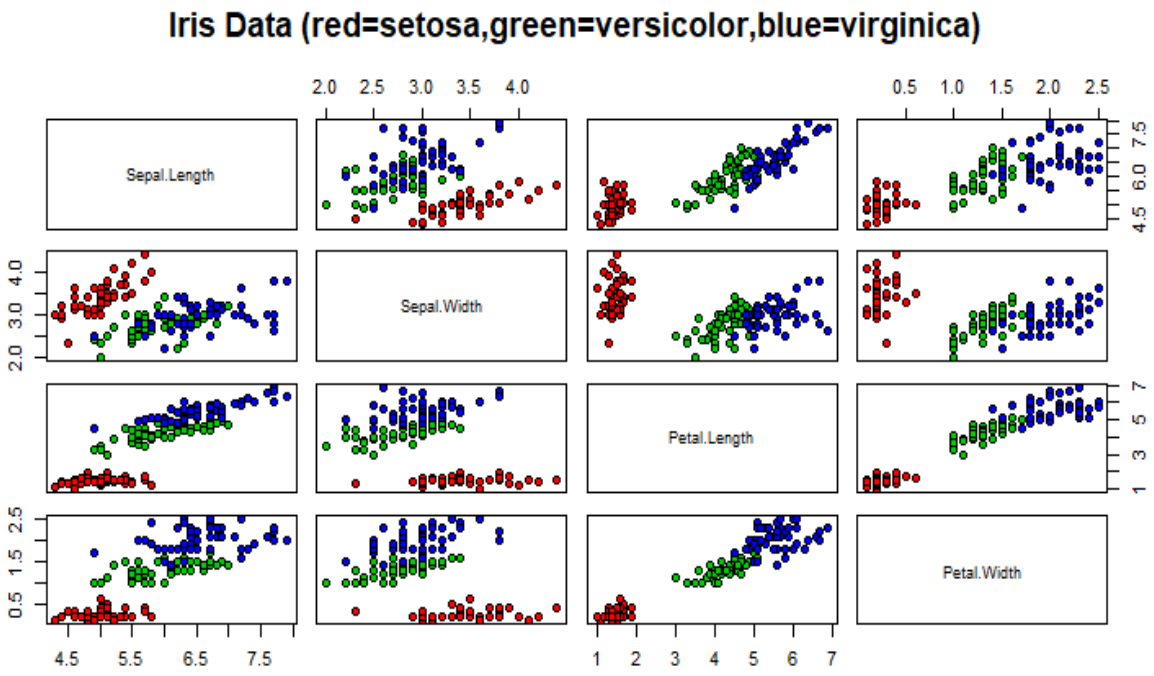


Plotting Graph:

Source Code:

```
#plot graph

pairs(iris[1:4], main="Iris Data
(red=setosa,green=versicolor,blue=virginica)",pch=21,bg=c("red",
"green3","blue")[unclass(iris$Species)])
```

Output:



Training Naive Bayes:

Source Code:

```
# training a naive Bayes model

index = sample(nrow(iris), floor(nrow(iris) * 0.7)) #70/30
split.

train = iris[index,]

test = iris[-index,]
```

```
xTrain = train[,-5] # removing y-outcome variable.

yTrain = train$Species # only y.

xTest = test[,-5]

yTest = test$Species

# nb - tells to use naive bayes

# cv - cross validation

model =
train(xTrain,yTrain,'nb',trControl=trainControl(method='cv',
number=10))

model

## table() gives frequency table, prop.table() gives freq%
table.

prop.table(table(predict(model$finalModel,xTest)$class,yTest))
```

Output:

```
Naive Bayes

105 samples
  4 predictor
  3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 95, 94, 95, 95, 93, 95, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.9616667  0.9420455
   TRUE      0.9516667  0.9268939

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust' was
 held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = FALSE and adjust = 1.
> ## table() gives frequency table, prop.table() gives freq% table.
> prop.table(table(predict(model$finalModel,xTest)$class,yTest))
            yTest
                 setosa versicolor  virginica
  setosa     0.22222222 0.00000000 0.00000000
  versicolor 0.00000000 0.37777778 0.04444444
  virginica  0.00000000 0.00000000 0.35555556
>
```

2)Implement K Nearest Neighbours:

Source Code:

Loading Dataset:

```
#K nearest neighbour

df<-data(iris) #load data

head(iris) #see the structure

#Generate a random number that is 90% of the total number of
rows in dataset.

ran<-sample(1:nrow(iris),0.9*nrow(iris))

ran
```

Output:

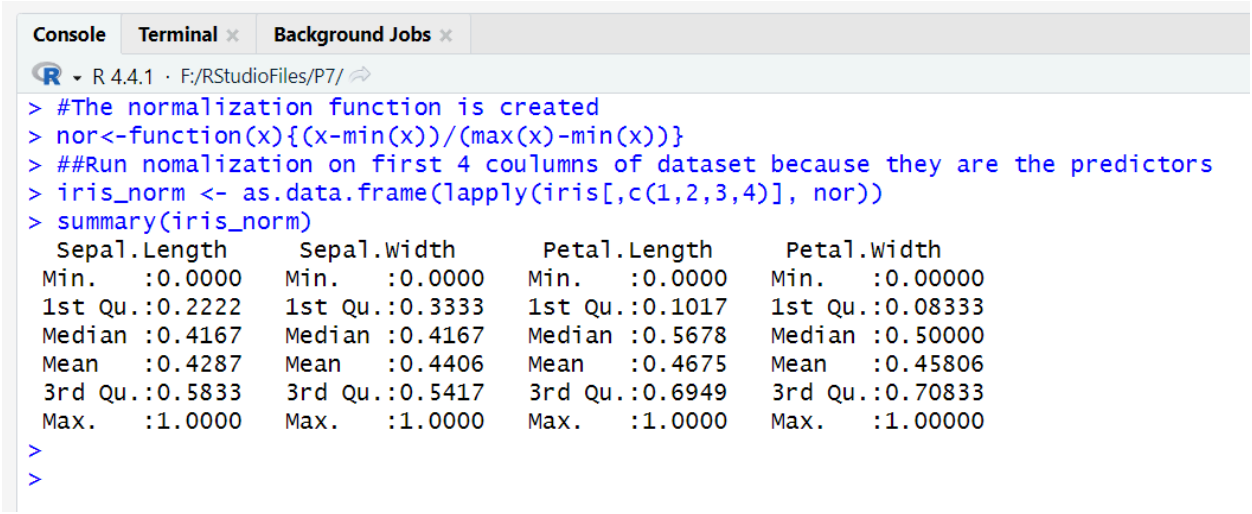
```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P7/
> #K nearest neighbour
> df<-data(iris) #load data
> head(iris) #see the structure
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> #Generate a random number that is 90% of the total number of rows in dataset.
> ran<-sample(1:nrow(iris),0.9*nrow(iris))
> ran
  [1]  99 144  84 139  76  19  11  37  62  73 124   4 130  31 103 111  17  83 102 126 138 147
 [23]  91 129  72  30  82  46 101 113 118 140  77 127 122  81 105 121  89 131  96 133  36  28
 [45]  57  18  79  92  90  10  51  68  43  45 134  21  33  80 141 112 104  34  15  85   5  88
 [67]  50  41  63  98  25 132 146  64  39  74 143  12 115  78  65  75 107  47 150   2 119 148
 [89] 149 100   9   3   8  40 116  86  35  58  93  16  49  38  94 128  95  13  14  23 136 109
[111]  97 137  56 106 145  70  52  71  26  54 117  67   6 125  59  60   1  27 110  66 123  20
[133] 120  29  53
>
```

Normalization of dataset:

Source Code:

```
#The normalization function is created

Nor<-function(x){(x-min(x))/(max(x)-min(x))}

##Run nomalization on first 4 coulumns of dataset because they
are the predictors

iris_norm <- as.data.frame(lapply(iris[,c(1,2,3,4)], nor))

summary(iris_norm)
```

Output:

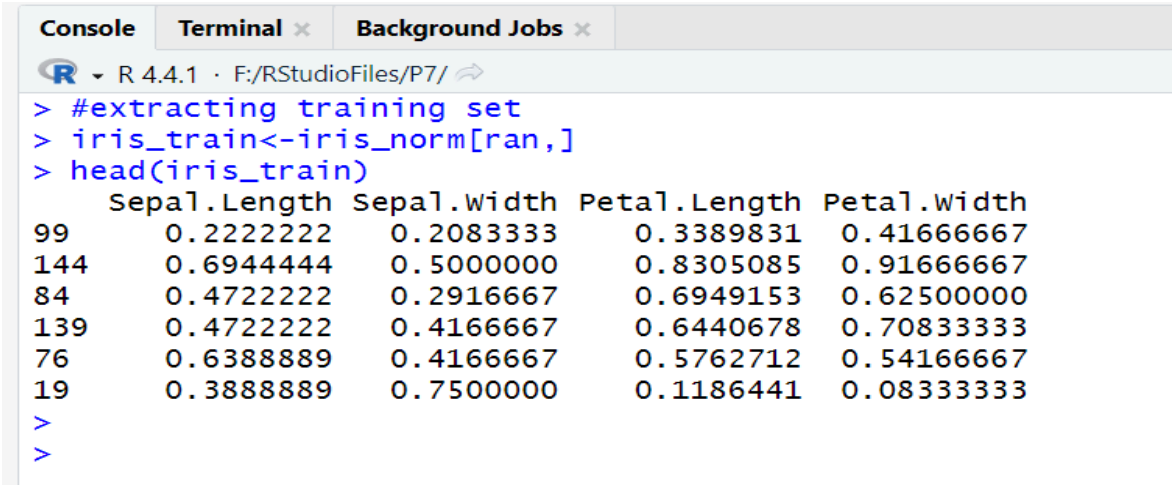


Extracting Training Set:

Source Code:

```
#extracting training set

iris_train<-iris_norm[ran,]

head(iris_train)
```

Output:



```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P7/
> #extracting training set
> iris_train<-iris_norm[ran,]
> head(iris_train)
    Sepal.Length Sepal.Width Petal.Length Petal.Width
99     0.2222222   0.2083333    0.3389831  0.41666667
144    0.6944444   0.5000000    0.8305085  0.91666667
84     0.4722222   0.2916667    0.6949153  0.62500000
139    0.4722222   0.4166667    0.6440678  0.70833333
76     0.6388889   0.4166667    0.5762712  0.54166667
19     0.3888889   0.7500000    0.1186441  0.08333333
>
>
```

Extracting Test Dataset:

Source Code:

```
#extracting test dataset

iris_test<-iris_norm[-ran,]

head(iris_test)
```

Output:

```
> #extracting test dataset
> iris_test<-iris_norm[-ran,]
> head(iris_test)
   Sepal.Length Sepal.Width Petal.Length Petal.Width
7    0.08333333   0.5833333   0.06779661  0.08333333
22   0.22222222   0.7083333   0.08474576  0.12500000
24   0.22222222   0.5416667   0.11864407  0.16666667
32   0.30555556   0.5833333   0.08474576  0.12500000
42   0.05555556   0.1250000   0.05084746  0.08333333
44   0.19444444   0.6250000   0.10169492  0.20833333
>
>
```

```
##extract 5th column of train dataset because it will be used as
'cl' argument in knn function.

iris_target_category <- iris[ran,5]

iris_target_category
```

Output:

```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P7/
> ##extract 5th column of train dataset because it will be used as 'cl' argument in knn function.
> iris_target_category <- iris[ran,5]
> iris_target_category
  [1] versicolor virginica  versicolor virginica  versicolor setosa     setosa     setosa
  [9] versicolor versicolor virginica  setosa     virginica  setosa     virginica  virginica
 [17] setosa     versicolor virginica  virginica  virginica  virginica  versicolor virginica
 [25] versicolor setosa     versicolor setosa     virginica  virginica  virginica  virginica
 [33] versicolor virginica  virginica  versicolor virginica  virginica  versicolor virginica
 [41] versicolor virginica  setosa     setosa     versicolor setosa     versicolor versicolor
 [49] versicolor setosa     versicolor versicolor setosa     setosa     virginica  setosa
 [57] setosa     versicolor virginica  virginica  virginica  setosa     setosa     versicolor
 [65] setosa     versicolor setosa     setosa     versicolor versicolor setosa     virginica
 [73] virginica  versicolor setosa     versicolor virginica  setosa     virginica  versicolor
 [81] versicolor versicolor virginica  setosa     virginica  setosa     virginica  virginica
 [89] virginica  versicolor setosa     setosa     setosa     setosa     virginica  versicolor
 [97] setosa     versicolor versicolor setosa     setosa     setosa     versicolor virginica
[105] versicolor setosa     setosa     setosa     virginica  virginica  versicolor virginica
[113] versicolor virginica  virginica  versicolor versicolor versicolor setosa     versicolor
[121] virginica  versicolor setosa     virginica  versicolor versicolor setosa     setosa
[129] virginica  versicolor virginica  setosa     virginica  setosa     versicolor
Levels: setosa versicolor virginica
```

```
##extract 5th column if test dataset to measure the accuracy

iris_test_category <- iris[-ran,5]

iris_test_category
```

Output:

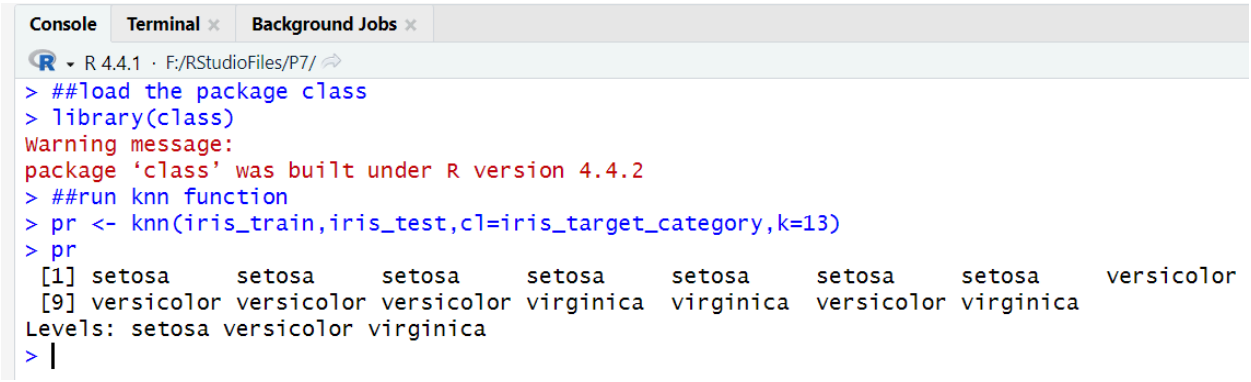
```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P7/
> ##extract 5th column if test dataset to measure the accuracy
> iris_test_category <- iris[-ran,5]
> iris_test_category
 [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa     versicolor
 [9] versicolor versicolor versicolor virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
>
```

Execute KNN Function:

Source Code:

```
##load the package class

library(class)

##run knn function

pr <- knn(iris_train,iris_test,cl=iris_target_category,k=13)

pr
```

Output:



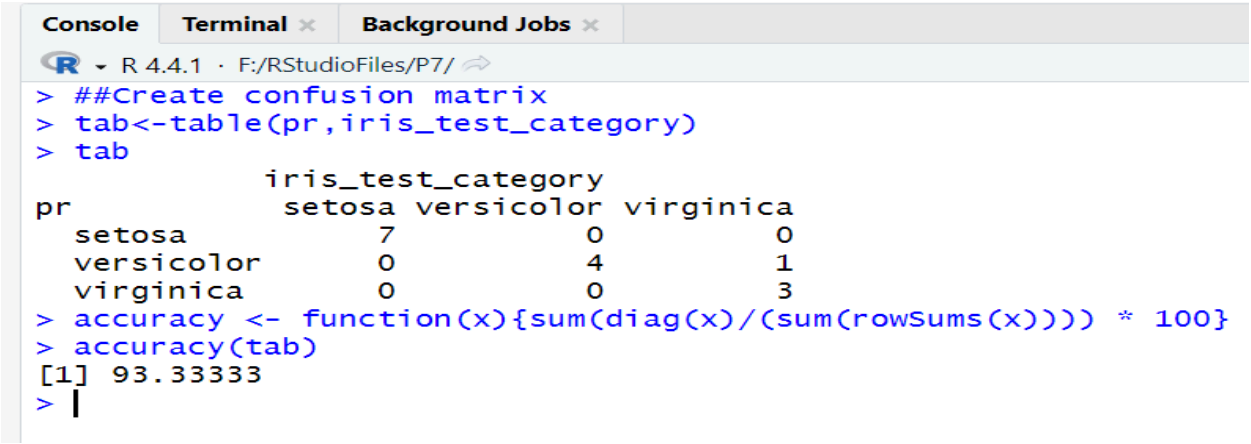
Create Confusion Matrix:

```
##Create confusion matrix

tab<-table(pr,iris_test_category)

tab
```

Checking Accuracy:

```
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}

accuracy(tab)
```

Output:



Conclusion:

From this practical, I have learned how to implement naive bayes,

KNN clustering in R.

<table>
<tr><td colspan="4">Name of Student: Shaikh Farhan Ahmed</td></tr>
<tr><td colspan="2">Roll Number: 50</td><td colspan="2">LAB Assignment Number: 08</td></tr>
<tr><td colspan="4">Title of LAB Assignment: Implementation and analysis of clustering algorithms like K-Means and hierarchical clustering.</td></tr>
<tr><td colspan="2">DOP: 18 – 11 – 2024</td><td colspan="2">DOS: 22 – 11 – 2024</td></tr>
<tr><td>CO Mapped:
CO4</td><td>PO Mapped:
PO1,PO2,PO3, PO4, PO5, PO6, PO7, PO8,
PSO1, PSO2</td><td>Faculty Signature:</td><td>Marks:</td></tr>
</table>

Practical No: 8

AIM: Implementation and analysis of clustering algorithms like K-Means and hierarchical clustering.

Theory:

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what K-means clustering algorithm, how the algorithm works, along with the Python implementation of k- means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs to only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

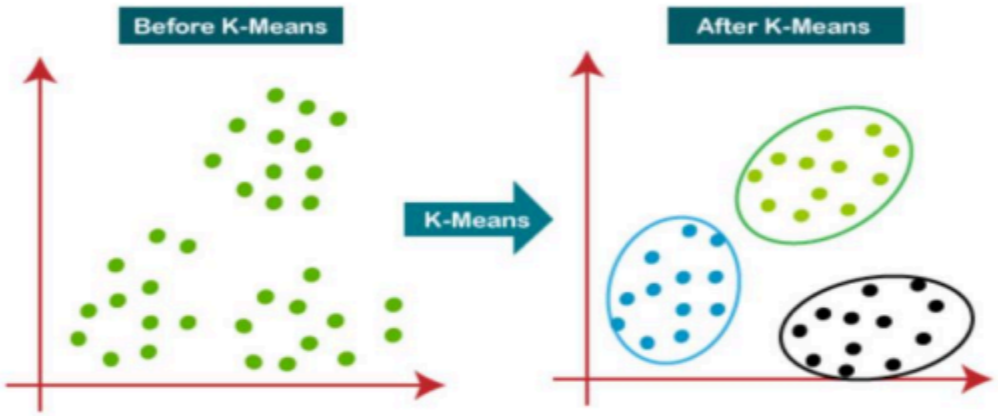
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has data points with some commonalities, and it is away from other clusters and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be different from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means re-assign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

1 IMPLEMENT K-MEANS CLUSTERING

LOADING DATASET:

```
#K-Means clustering
head(iris)
```

Output:

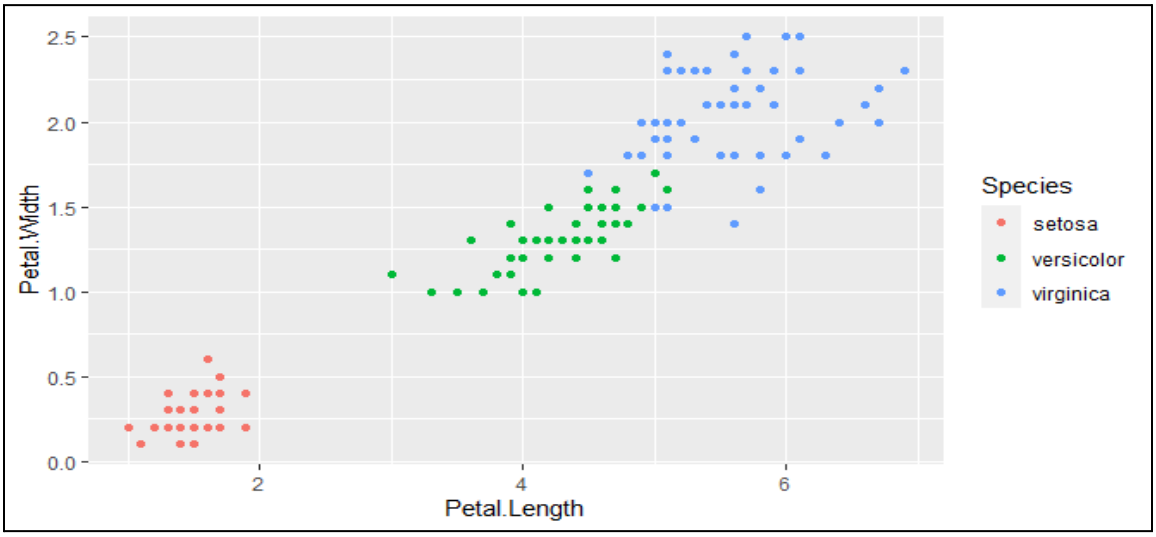
```
R 4.2.2 · D:/DAL/P8/
> setwd("D:/DAL/P8")
> #K-Means clustering
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
>
```

2. LOADING LIBRARY FOR PLOTTING GRAPH:

Source Code:

```
library(ggplot2)
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) +
geom_point()
```

Output:



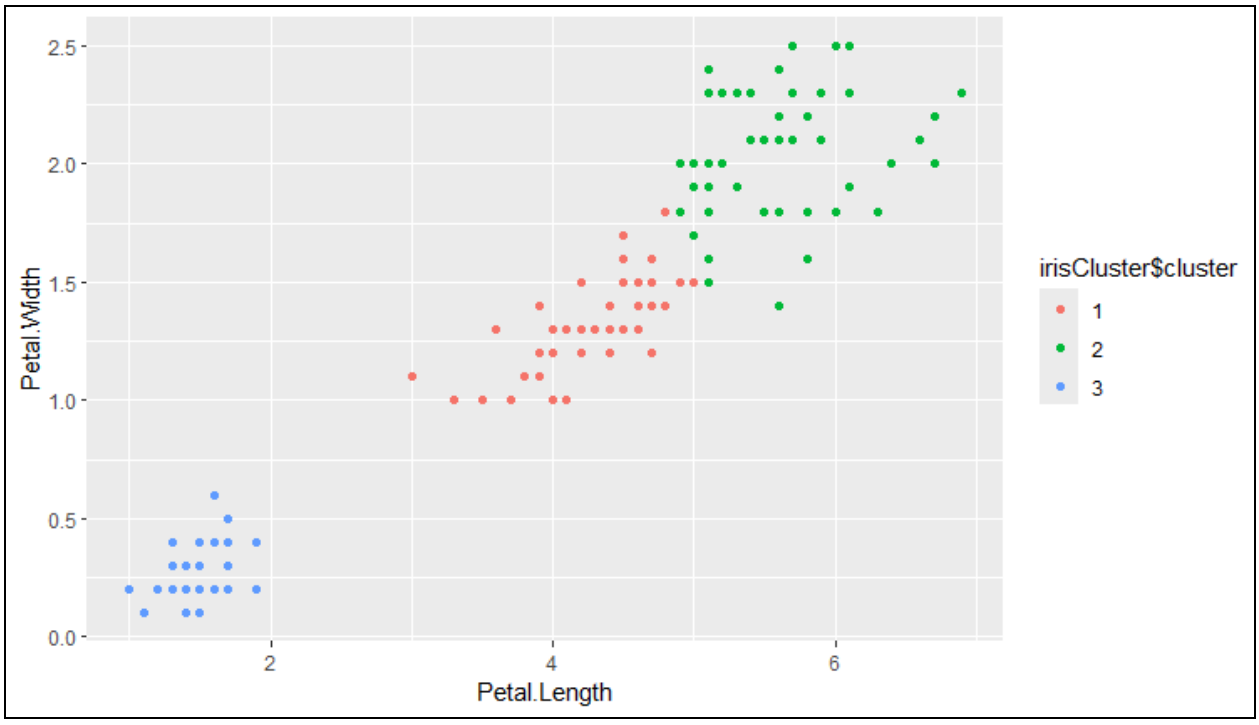
EXECUTING K-MEANS AND PLOTTING GRAPH:

Source Code:

```
#Setting a seed in R means to initialize a pseudorandom number
generator.
set.seed(20)
#Executing Kmeans
irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
irisCluster
table(irisCluster$cluster, iris$Species)
irisCluster$cluster <- as.factor(irisCluster$cluster)
ggplot(iris, aes(Petal.Length, Petal.Width, color =
irisCluster$cluster)) + geom_point()
```

Output:

```
K-means clustering with 3 clusters of sizes 52, 48, 50

Cluster means:
  Petal.Length Petal.Width
1     4.269231    1.342308
2     5.595833    2.037500
3     1.462000    0.246000

Clustering vector:
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [45] 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1
 [89] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2
[133] 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1] 13.05769 16.29167  2.02200
 (between_SS / total_SS =  94.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
> table(irisCluster$cluster, iris$Species)

    setosa versicolor virginica
  1      0         48         4
  2      0          2        46
  3     50          0         0
> irisCluster$cluster <- as.factor(irisCluster$cluster)
> ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()
```



2. K-Means Agglomerative:

```
LOADING DATASET
#K-Means Agglomerative
head(iris)
```

Output:

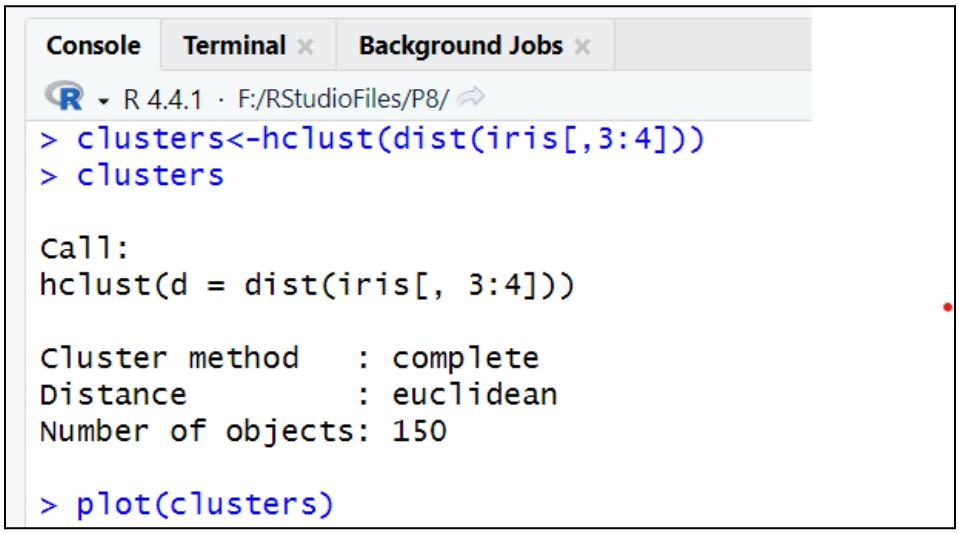
```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P8/
> #K-Means Agglomerative
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
>
```

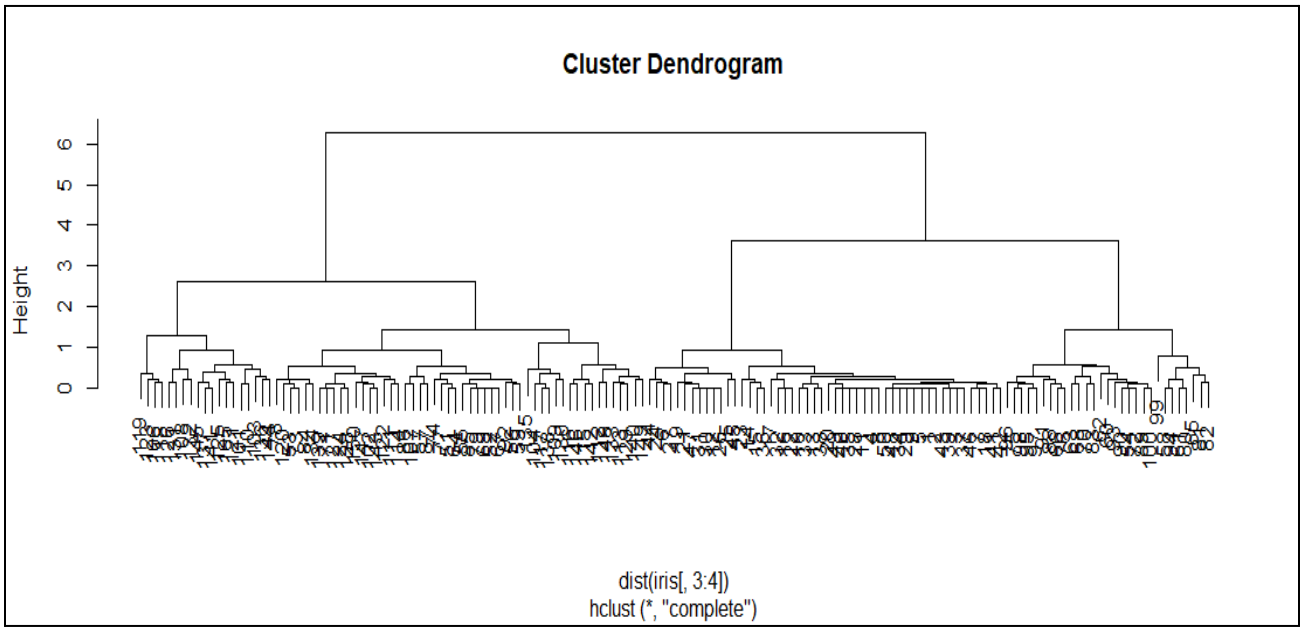
CREATING CLUSTERS:

Source Code:

```
clusters<-hclust(dist(iris[,3:4]))
clusters
plot(clusters)
```

Output:





CREATING GROUP OF CLUSTERS

Source Code:

```
clusterCut<-cutree(clusters,3)
clusterCut
table(clusterCut, iris$Species)
```

Output:

```
Console   Terminal ×   Background Jobs ×
R 4.4.1 · F:/RStudioFiles/P8/
> clusterCut<-cutree(clusters,3)
> clusterCut
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [33] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 2 2 3 2 3 3 3 3 2
 [65] 3 3 2 3 2 3 2 3 2 2 3 3 2 2 2 3 3 3 3 2 2 2 2 3 3 3 3 2 3 3 3 3
 [97] 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[129] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
> table(clusterCut, iris$Species)

clusterCut setosa versicolor virginica
         1     50          0         0
         2      0         21        50
         3      0         29         0
>
> |
```

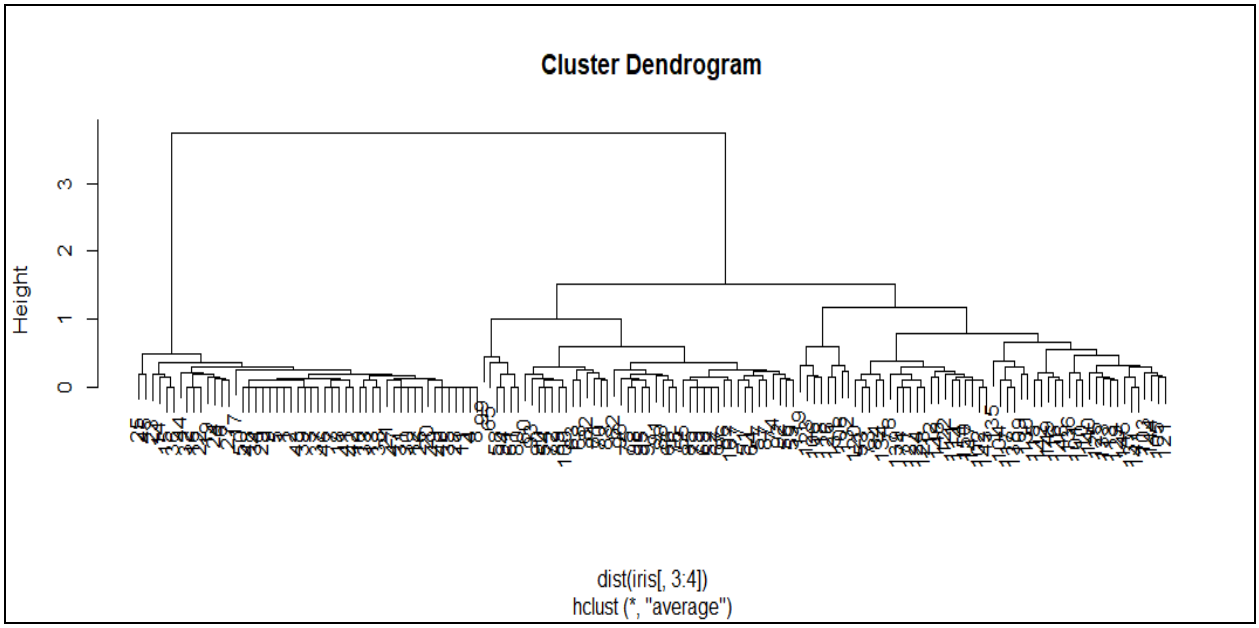
CREATING AVERAGE LINKAGE CLUSTERS:

Source Code:

```
clusters<-hclust(dist(iris[, 3:4]), method = 'average')
clusters
plot(clusters)
```

Output:

```
Console   Terminal ×   Background Jobs ×
R 4.4.1 · F:/RStudioFiles/P8/
> clusters<-hclust(dist(iris[, 3:4]), method = 'average')
> clusters

Call:
hclust(d = dist(iris[, 3:4]), method = "average")

Cluster method   : average
Distance         : euclidean
Number of objects: 150

> plot(clusters)
> |
```



CREATING GROUP CLUSTER FOR AVERAGE LINKAGE

Source Code:

```
clusterCut<-cutree(clusters,3)
clusterCut
table(clusterCut,iris$Species)
```

Output:

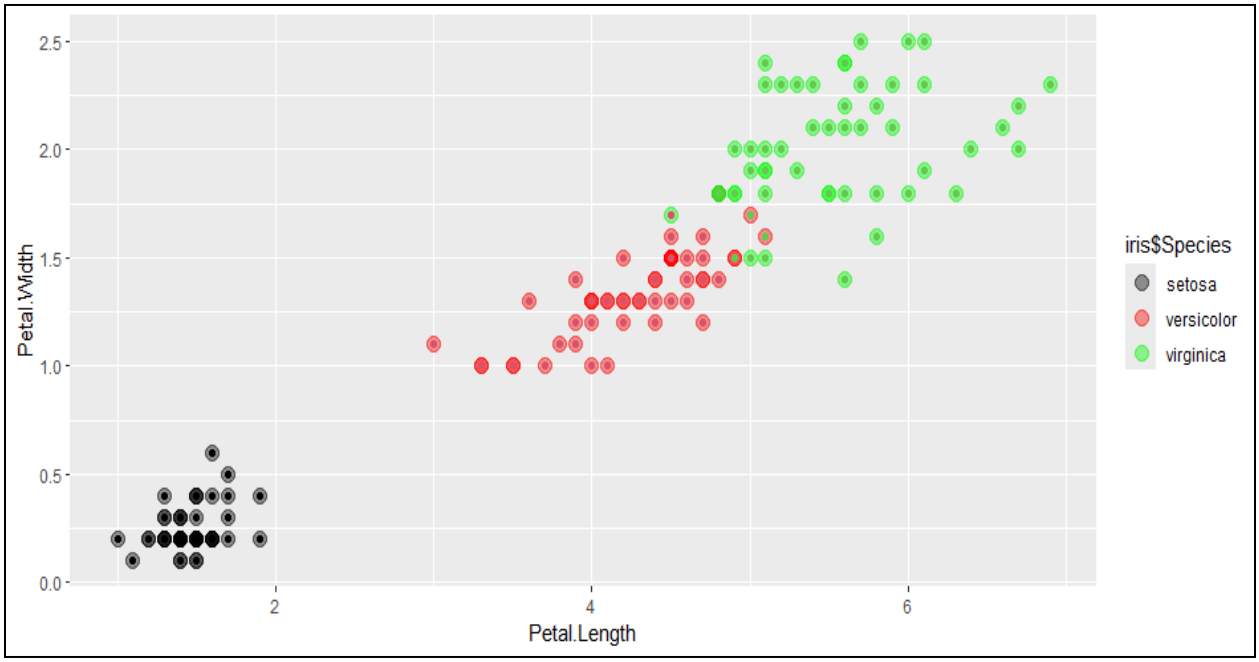
```
Console  Terminal  Background Jobs
R 4.4.1 · F:/RStudioFiles/P8/
> clusterCut<-cutree(clusters,3)
> clusterCut
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [33] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [65] 2 2 2 2 2 2 3 2 3 2 2 2 2 3 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [97] 2 2 2 2 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[129] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
> table(clusterCut,iris$Species)

clusterCut setosa versicolor virginica
         1     50          0         0
         2      0         45         1
         3      0          5        49
> |
```

PLOTTING CLUSTERED GRAPH

Source Code:

```
ggplot(iris,aes(Petal.Length, Petal.Width,
color=iris$Species))+geom_point(alpha=0.4,size=3.5)+
geom_point(col=clusterCut)+
scale_color_manual(values=c("black","red","green"))
```

Output:



CONCLUSION:

From this practical, I have learned to implement K-Means Clustering and agglomerative in R.

Name of Student: Shaikh Farhan Ahmed			
Roll Number: 50		**LAB Assignment Number: 09**	
Title of LAB Assignment: Implement various data visualization techniques using tableau.			
DOP: 18 – 11 – 2024		**DOS: 25 – 11 – 2024**	
CO Mapped: **CO5**	**PO Mapped:** **PO1, PO2, PO3, PO5, PO6, PO7, PO8, PO11, PSO1, PSO2**	**Faculty Signature:**	**Marks:**

Practical 9

AIM: Implement various data visualization techniques using tableau

Theory:

Tableau is a visual analytics platform that is revolutionizing the way we use data to solve problems by enabling individuals and organisations to make the most of their data.
Tableau is a great data visualization and business intelligence application that can be used to report and analyse massive amounts of data. Salesforce purchased Tableau in June 2019, an American firm founded in 2003. It enables users to build various charts, graphs, maps, dashboards, and stories for visualising and analysing data in order to aid in business choices. Tableau offers several unique and fascinating features that make it one of the most popular business intelligence (BI) applications.

Tableau Features

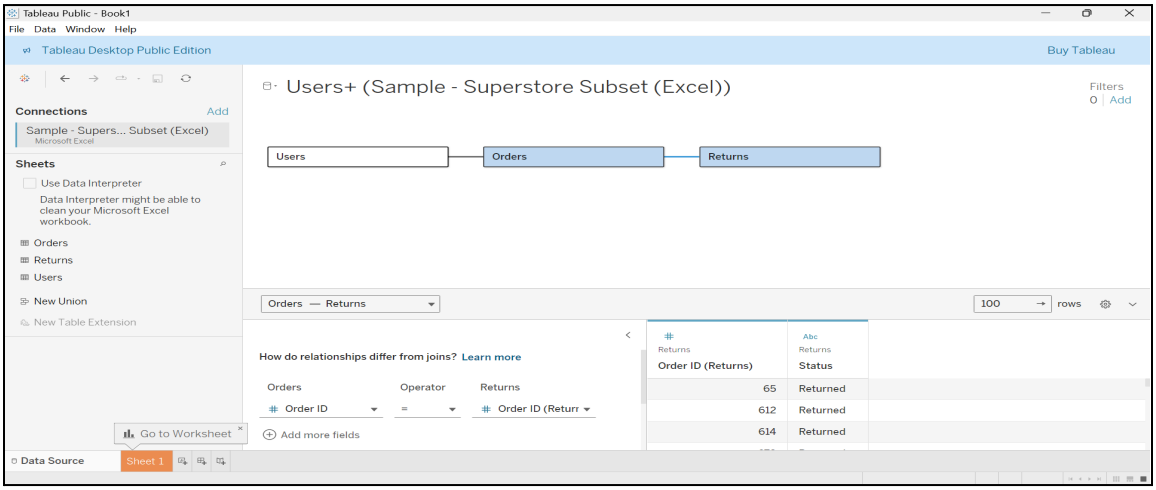
- Tableau supports powerful data discovery and exploration that enables users to answer important questions in seconds
- No prior programming knowledge is needed; users without relevant experience can start immediately with creating visualizations using Tableau
- It can connect to several data sources that other BI tools do not support. Tableau enables users to create reports by joining and blending different datasets
- Tableau Server supports a centralized location to manage all published data sources within an organization

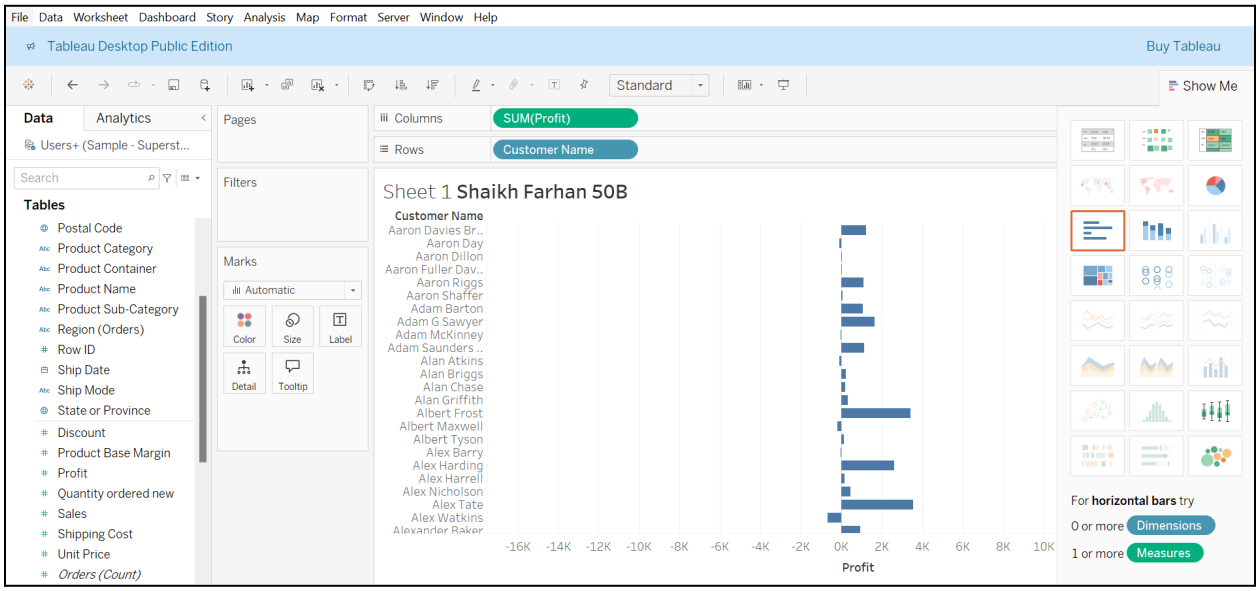
Assignment 1: Analysis Operations using tableau

Q 1: Find the customer with the highest overall profit. What is his/her profit ratio?

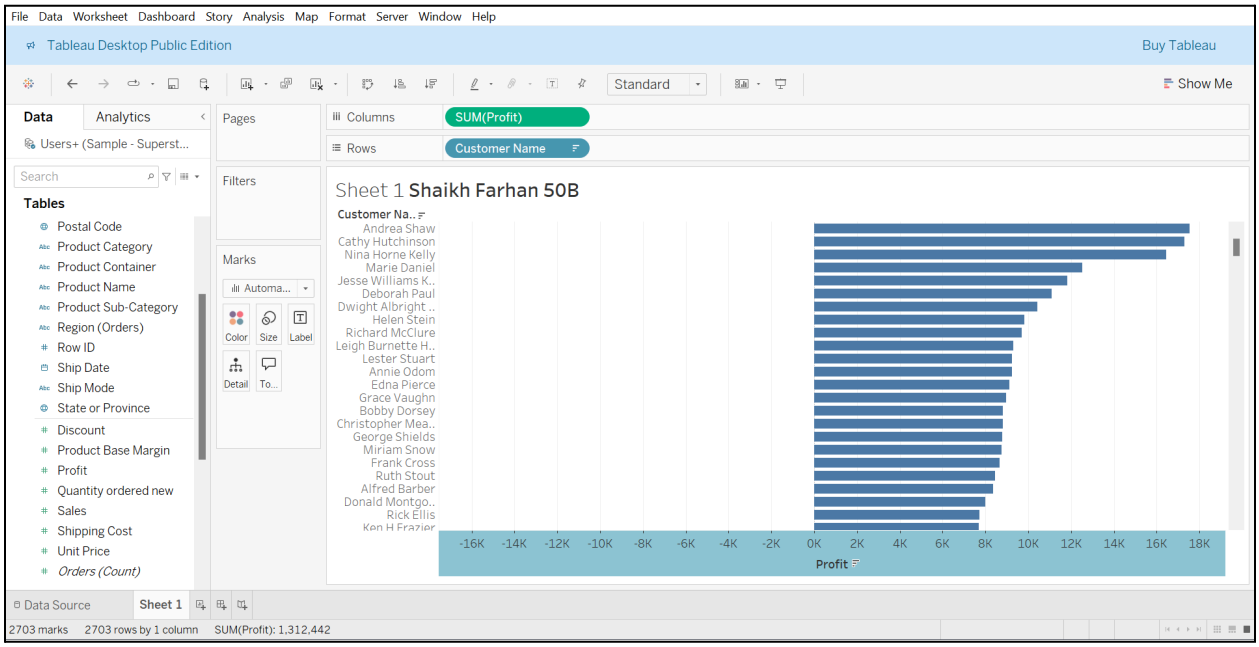
Ans:

Step 1: Open the superstore subset 2015 excel data set
Step 2: Drag Orders sheet to sheet area
Step 3: Go to sheet 1 and add Customer name as rows and profit as column

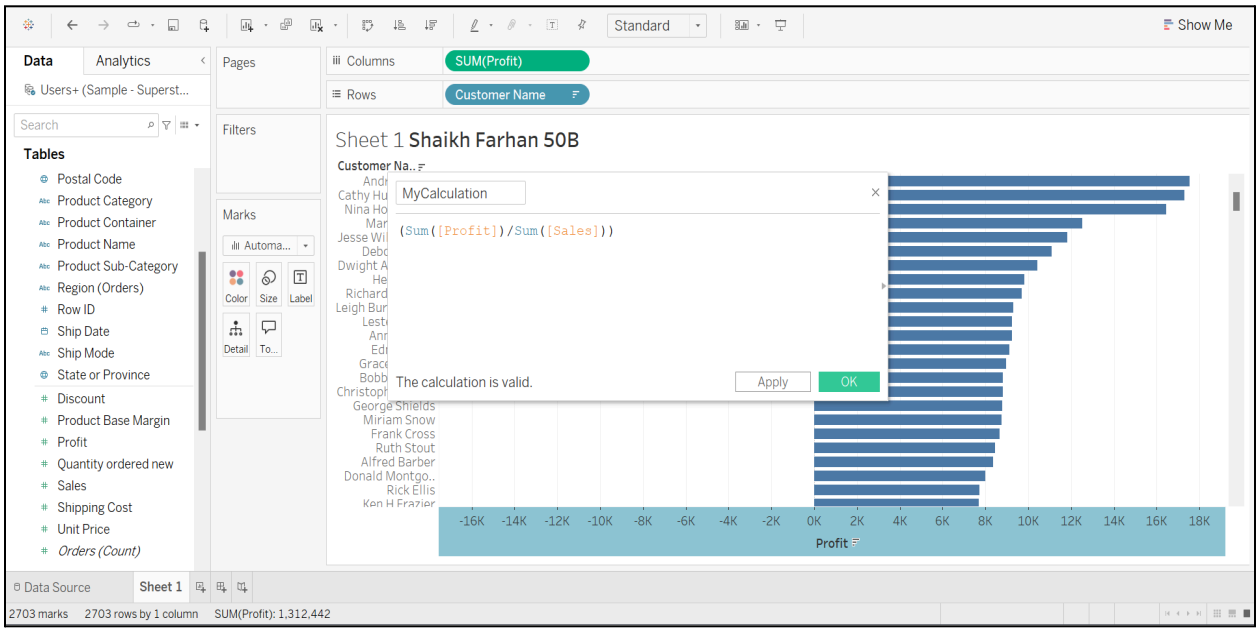


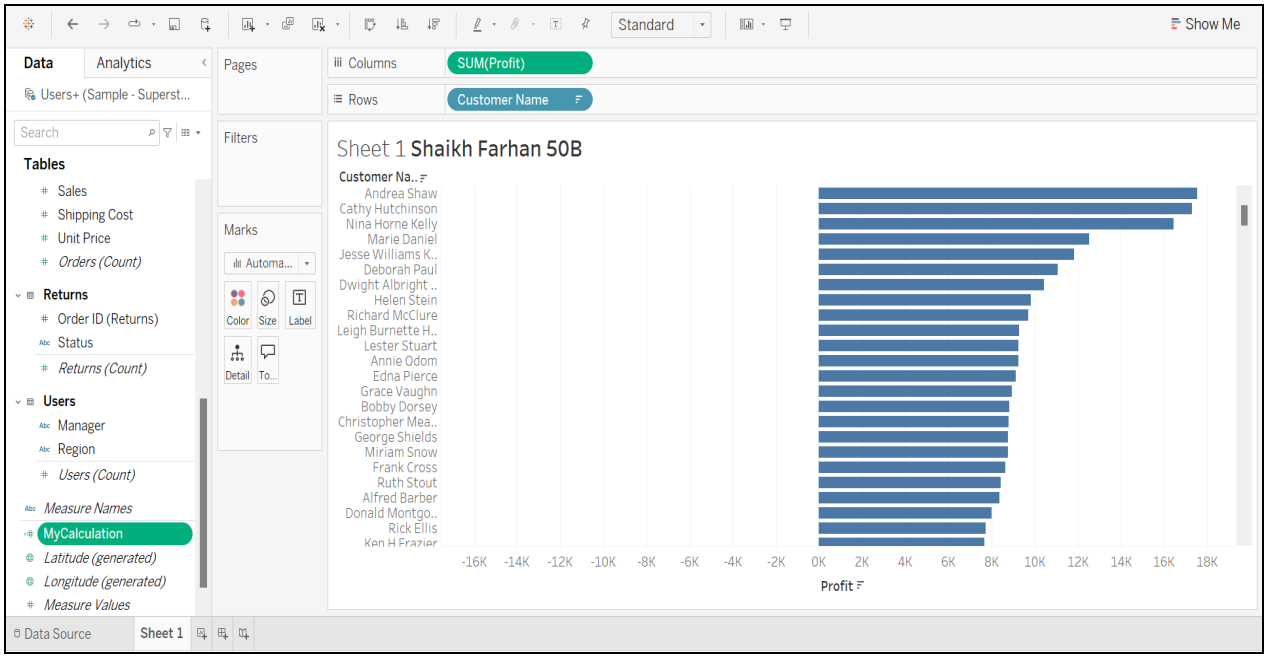


Step 4: Sort the data by clicking on Profit label on bottom

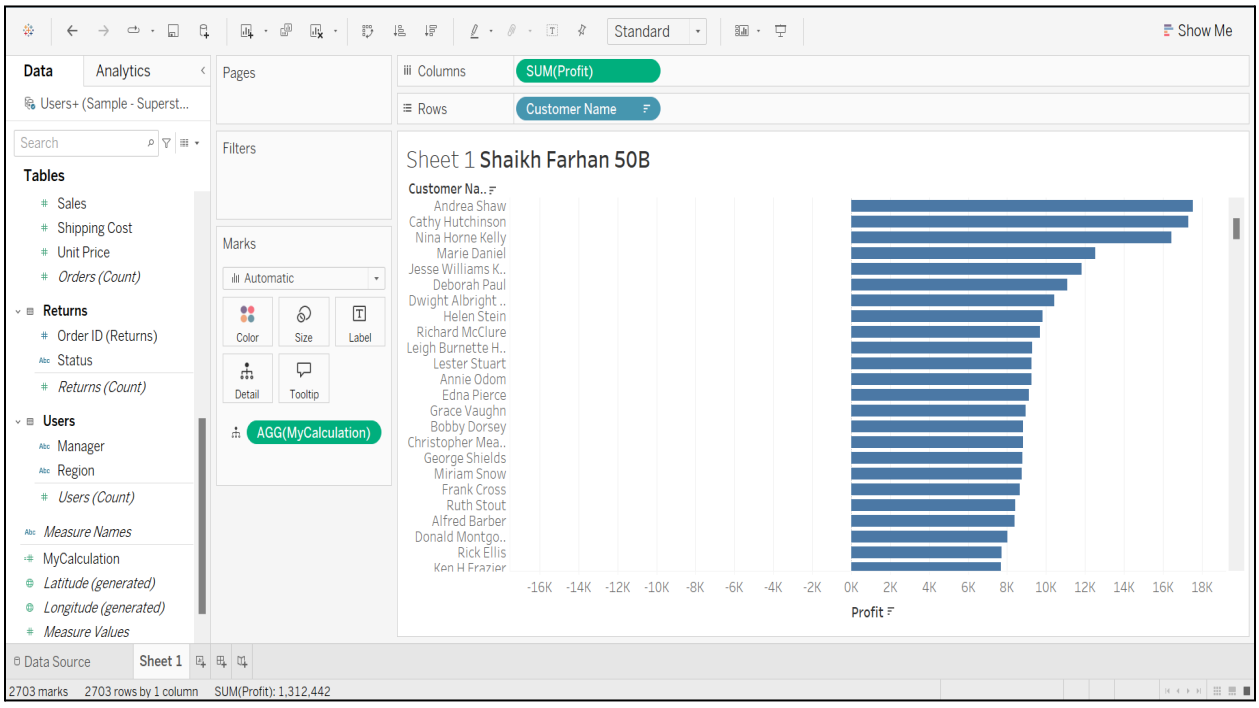


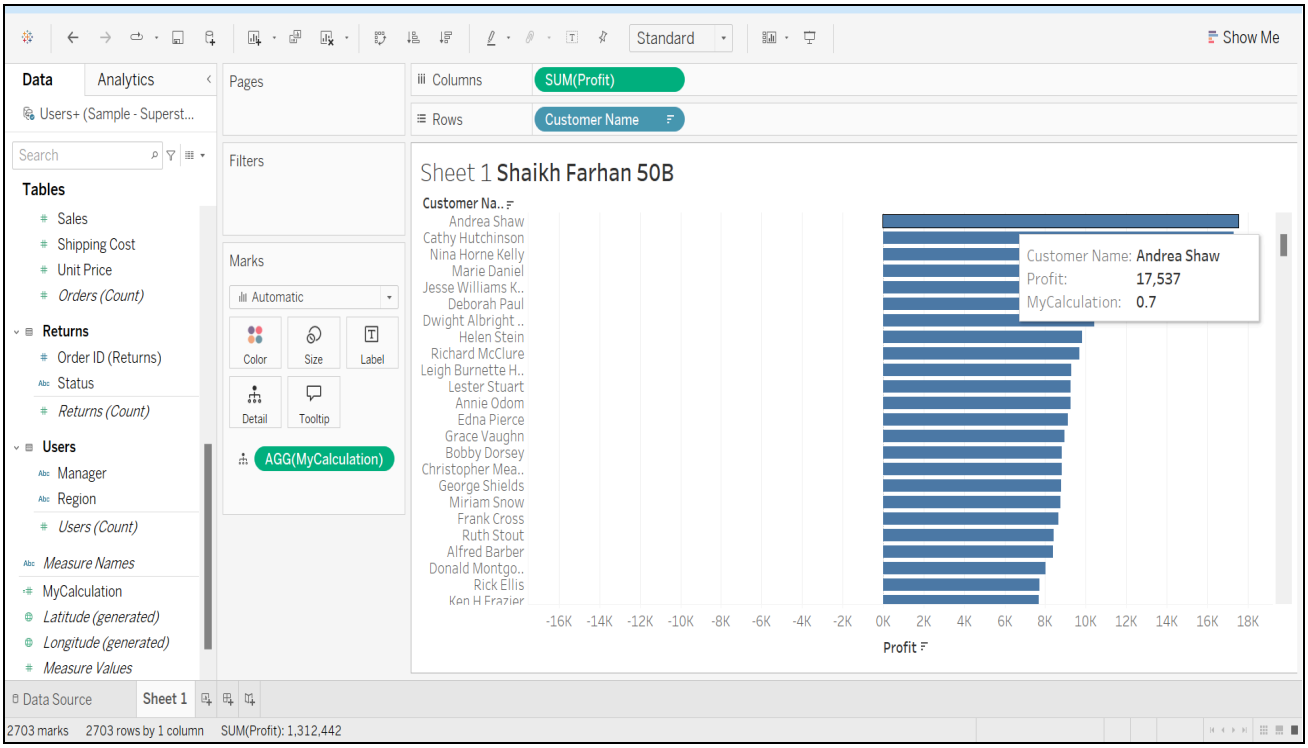
Step 5: To calculate profit Ratio
Profit Ratio= (Sum([Profit])/Sum([Sales]))
This formula needs to be entered as tooltip or label
Click on Analysis>Create Calculated Field and enter the formula





You can see **MyCalculation** in measures. Drag it to the Marks area.





Final Answer:

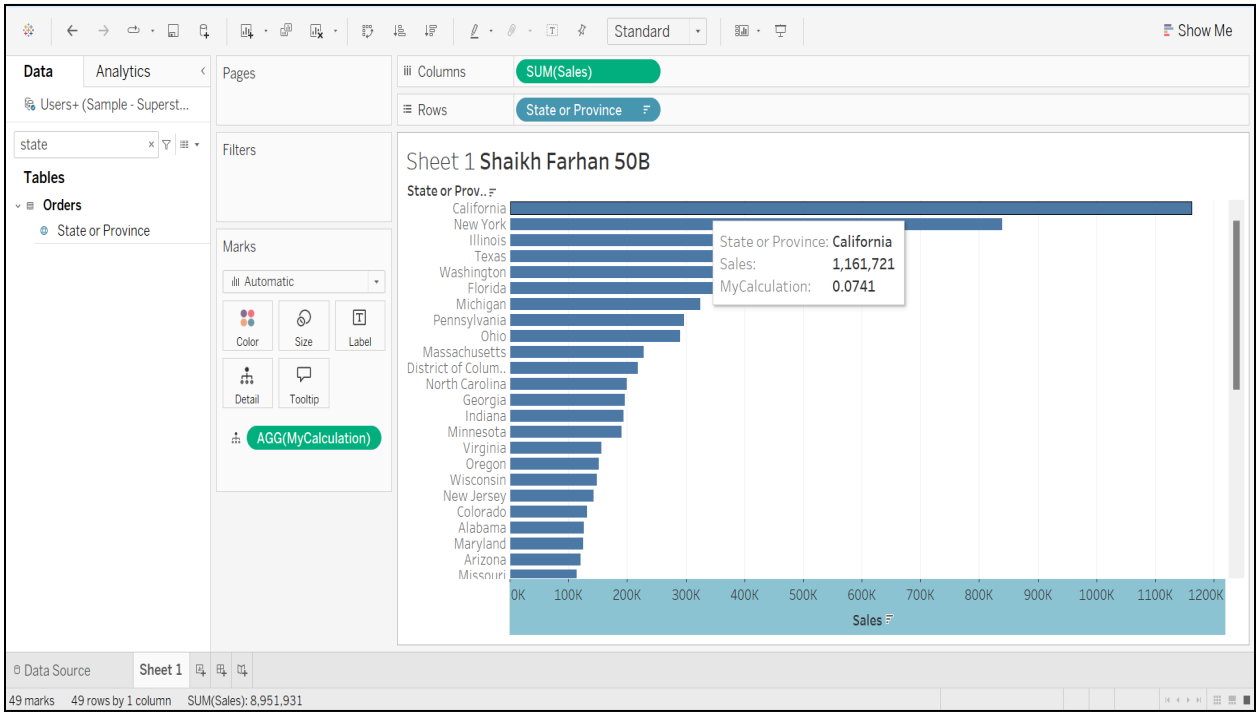
Q2: Which state has the highest Sales (Sum)? What is the total Sales for that state?

Ans:

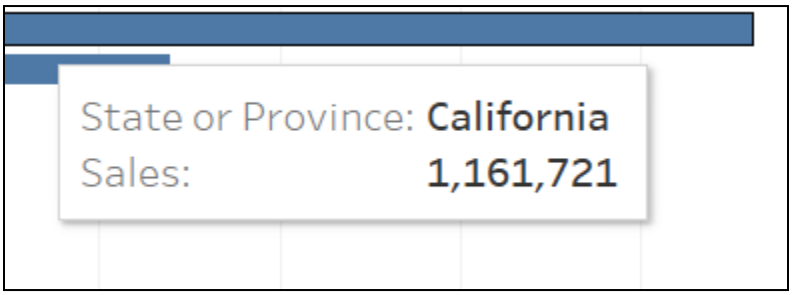
Step 1: Drag SUM as Columns

Step 2: Drag State or Province as Rows

Step 3: Sort the Sales from Highest to lowest.



Final Answer:

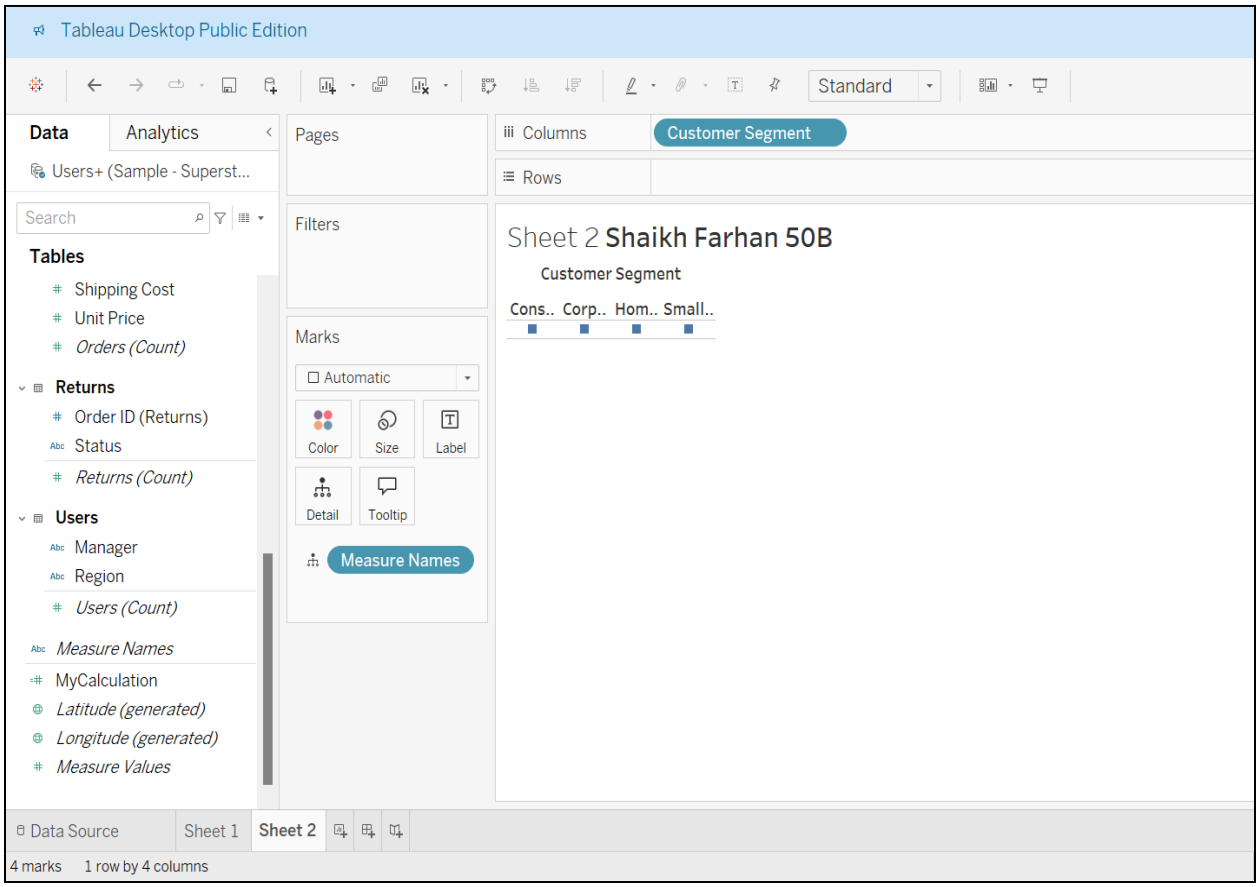


Q3: Which customer segment has both the highest order quantity and average discount rate? What is the order quantity and average discount rate for that state?

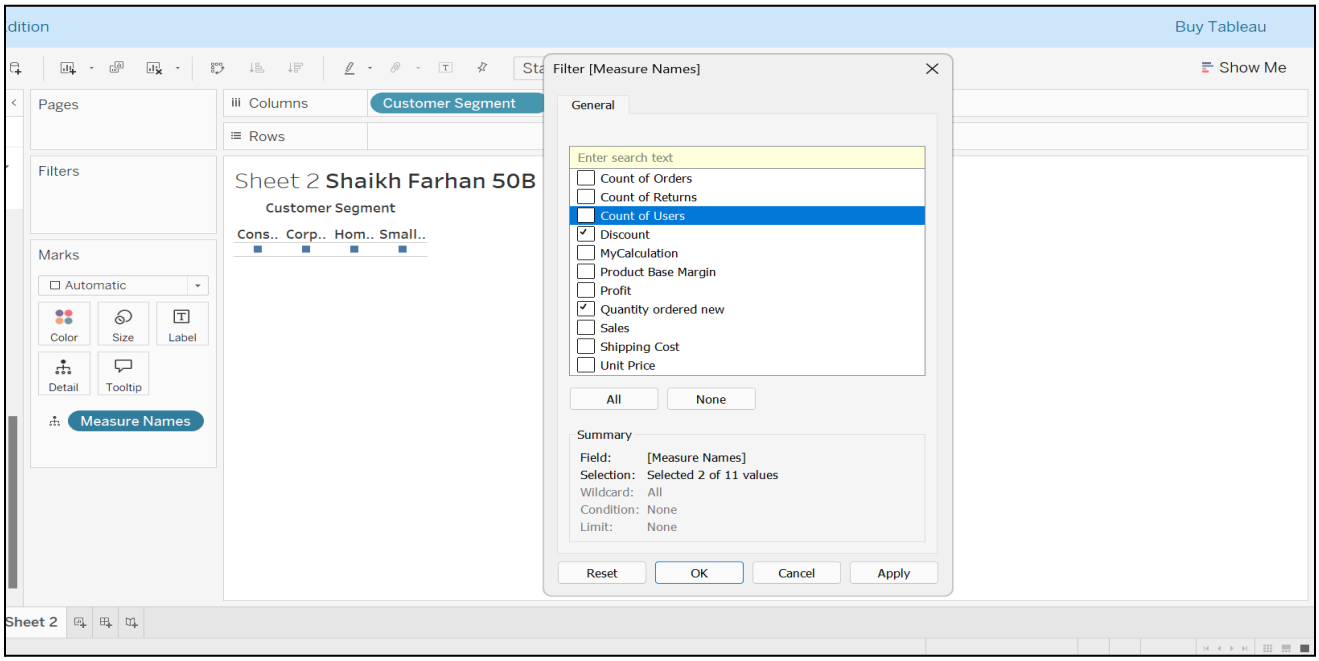
Ans:

Step 1: Drag Customer Segment to columns section

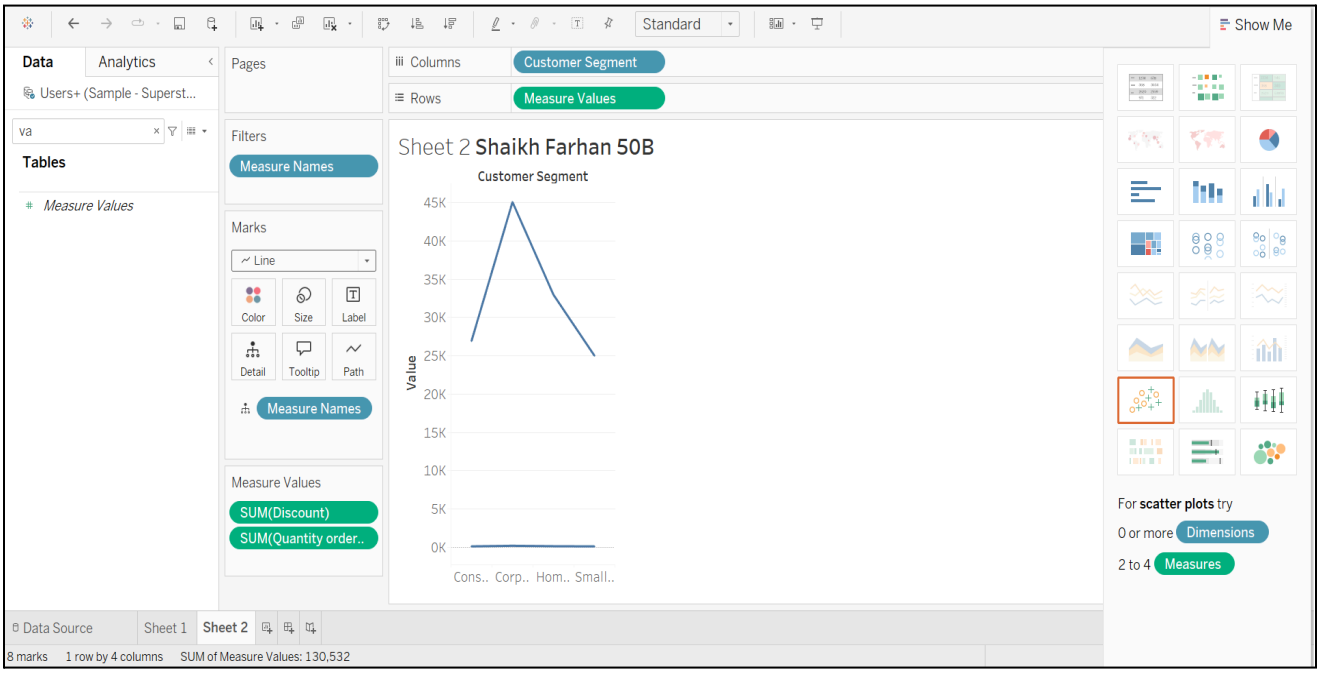
Step 2: Drag Measure Names into the Marks Section.



Step 3: Filter Measure Names as Discount and Quality Ordered New.



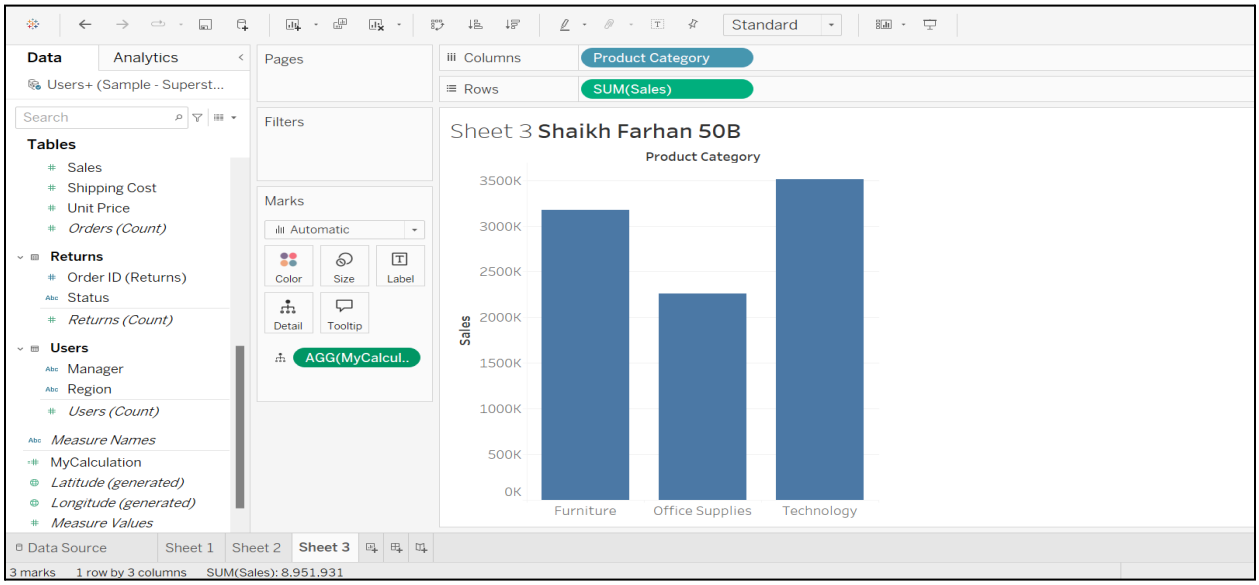
Step 4: Drag Measure Values into Rows section.
Step 5: Change Marks representation from automatic to line.



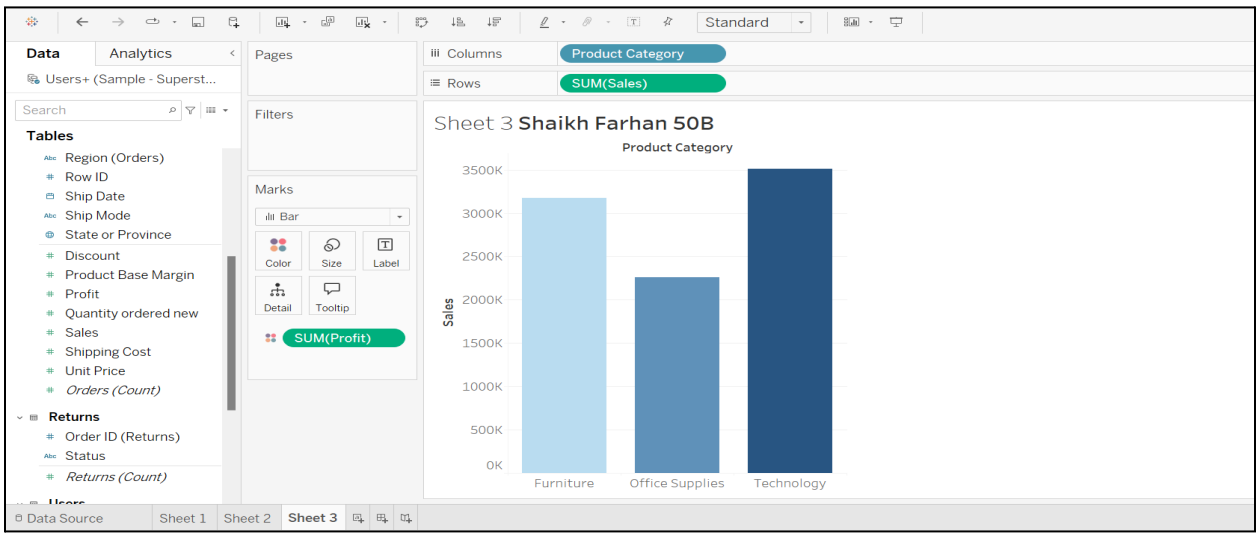
Q 4: Which Product Category has the highest total Sales? Which Product Category has the worst Profit? Name the Product Category and $ amount for each.

Ans:

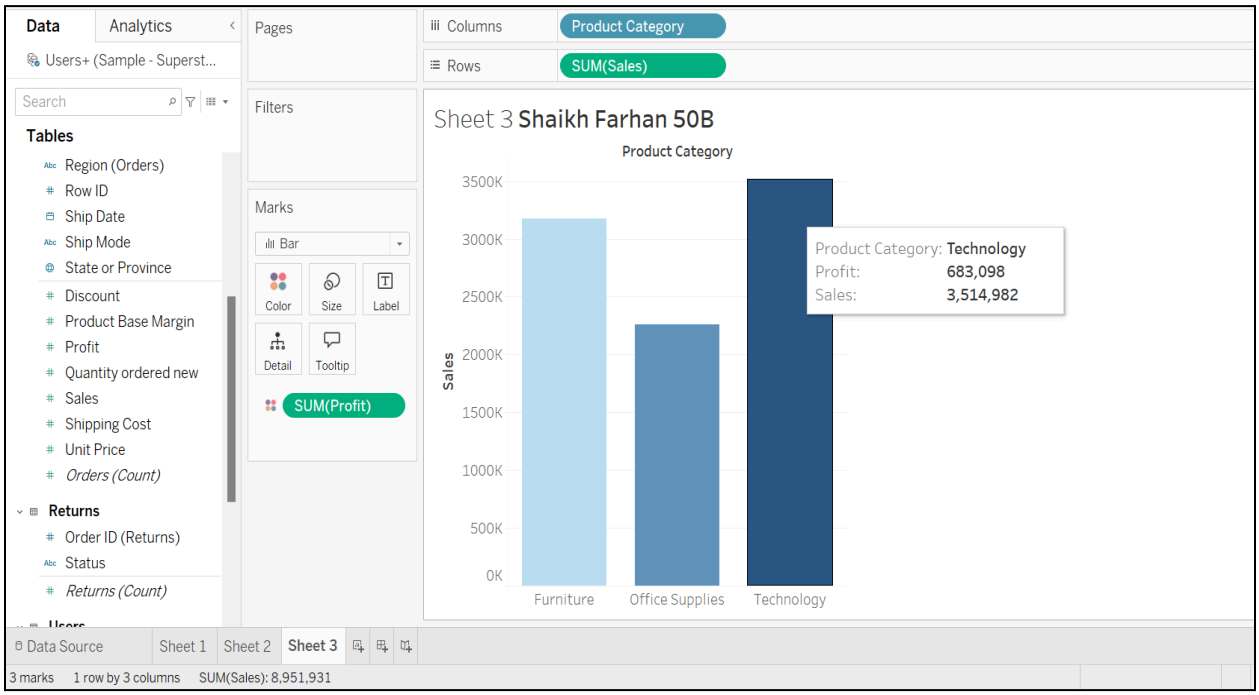
1. Bar Chart displaying total Sales for each Product Category



2. Add a color scale indicating Profit by dragging Profit into Marks section and Dragging it onto Color Option.



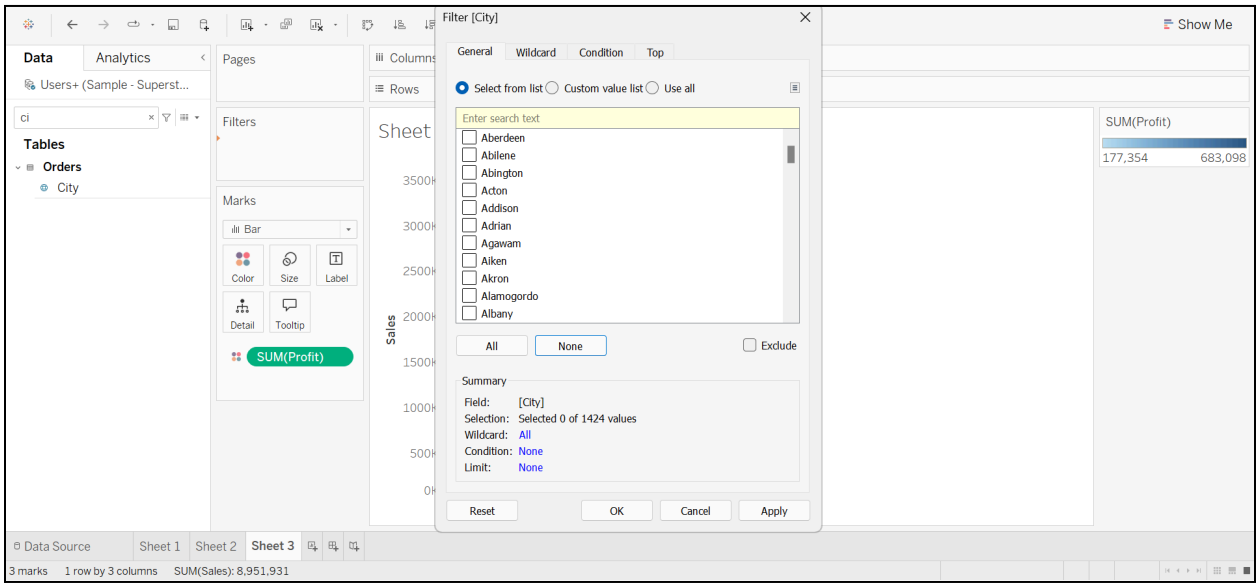
3. Each Product Category labeled with total Sales and Each Product Category labeled with Profit as well.

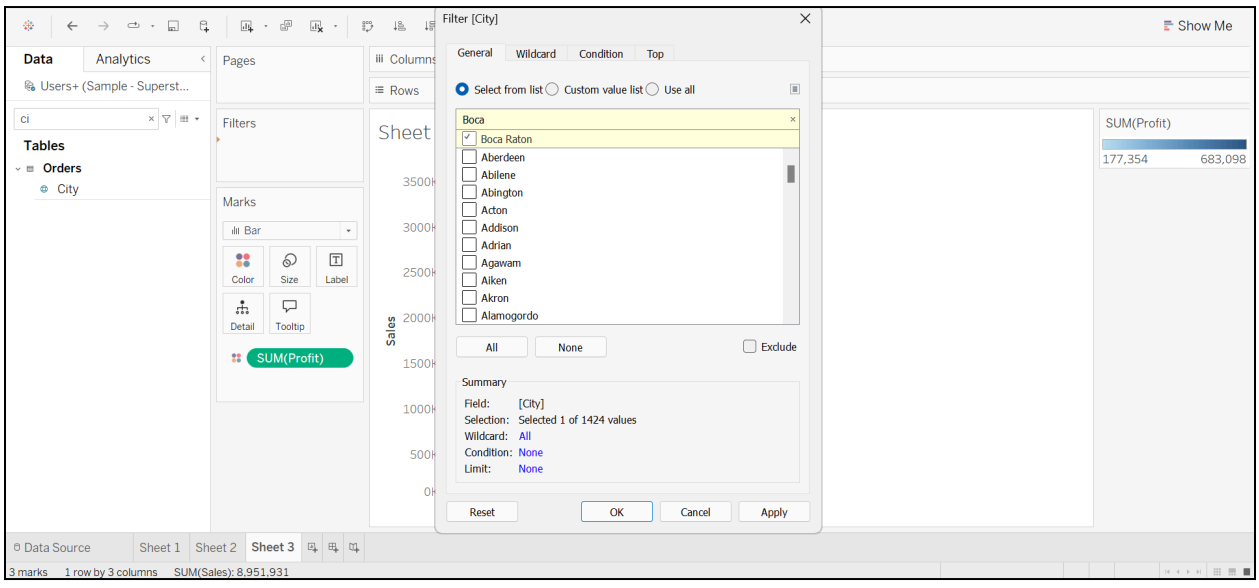


Q 5: Use the same visualization created for Question #4.What was the Profit on Technology (Product Category) in Boca Raton (City)?

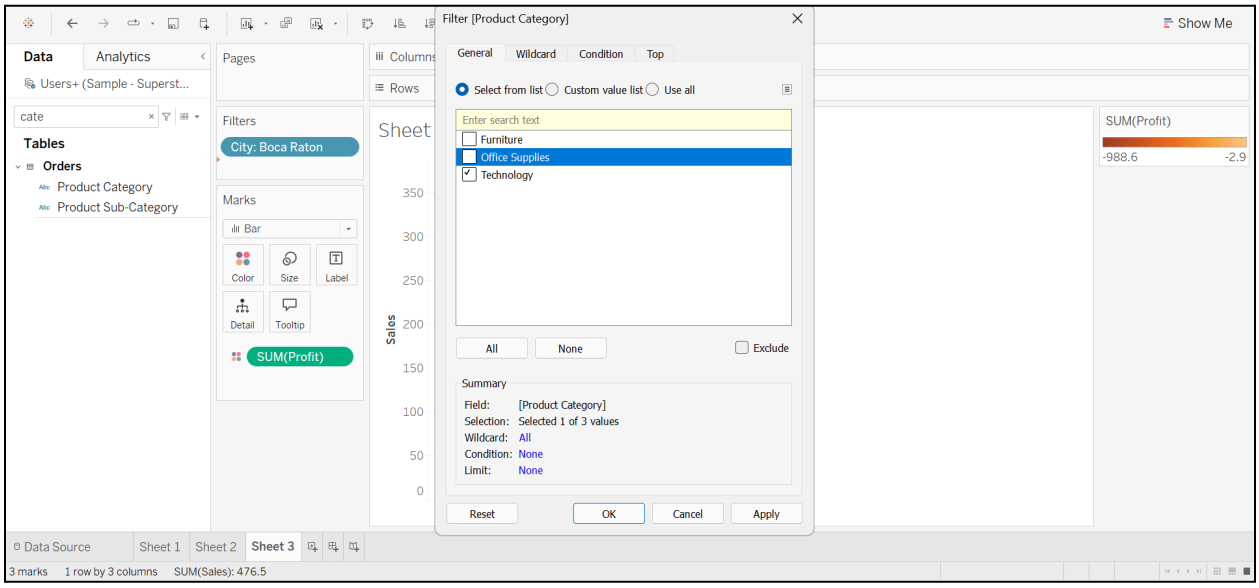
Ans:

Step 1: Add Filter City
Step 2: Search and Select Boca Raton City.

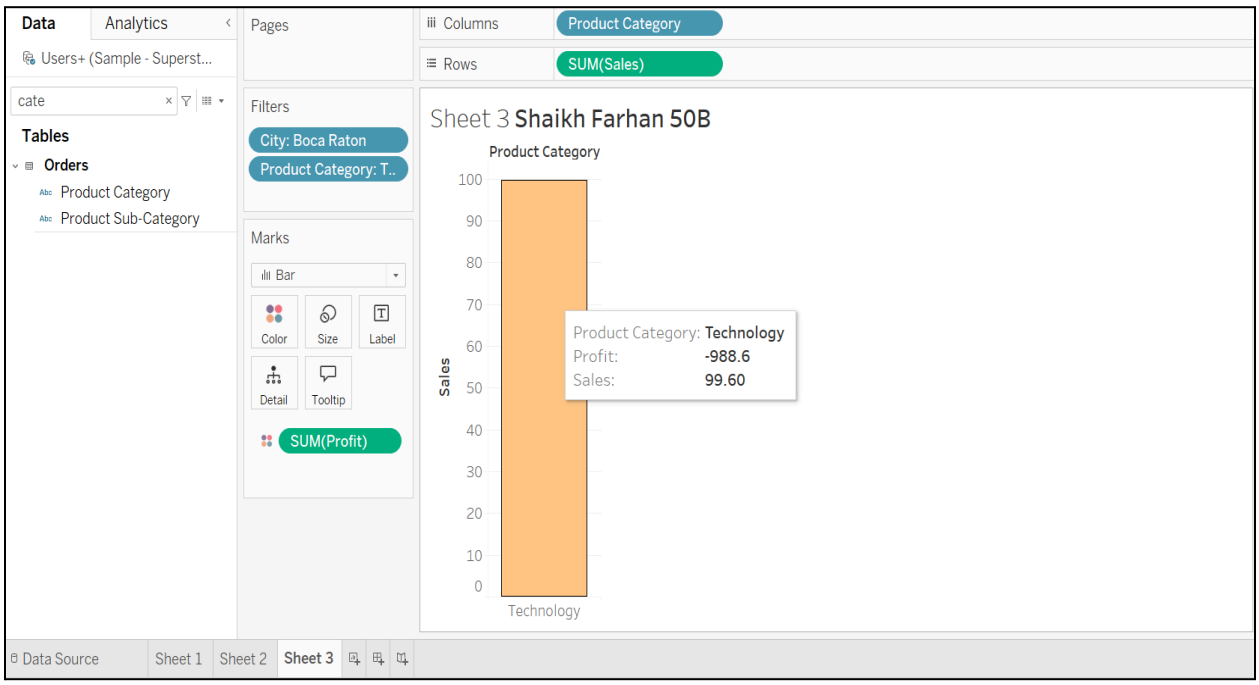




Step 3: Add Product Category as a filter and select Technology.



Step 4: Apply filter to get the result.



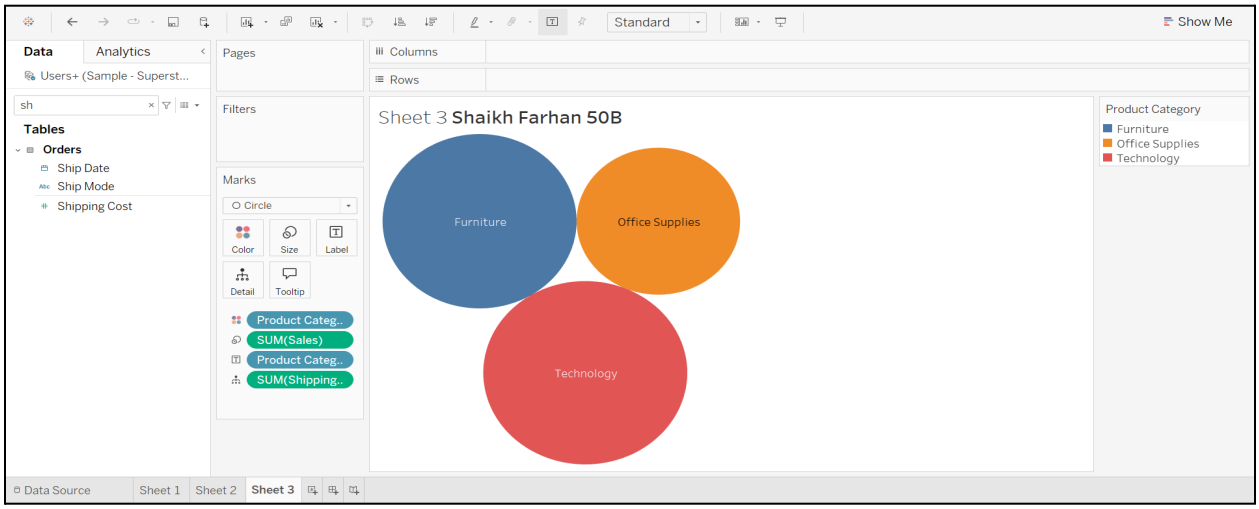
Q6: Which Product Department has the highest Shipping Costs? Name the Department and cost.

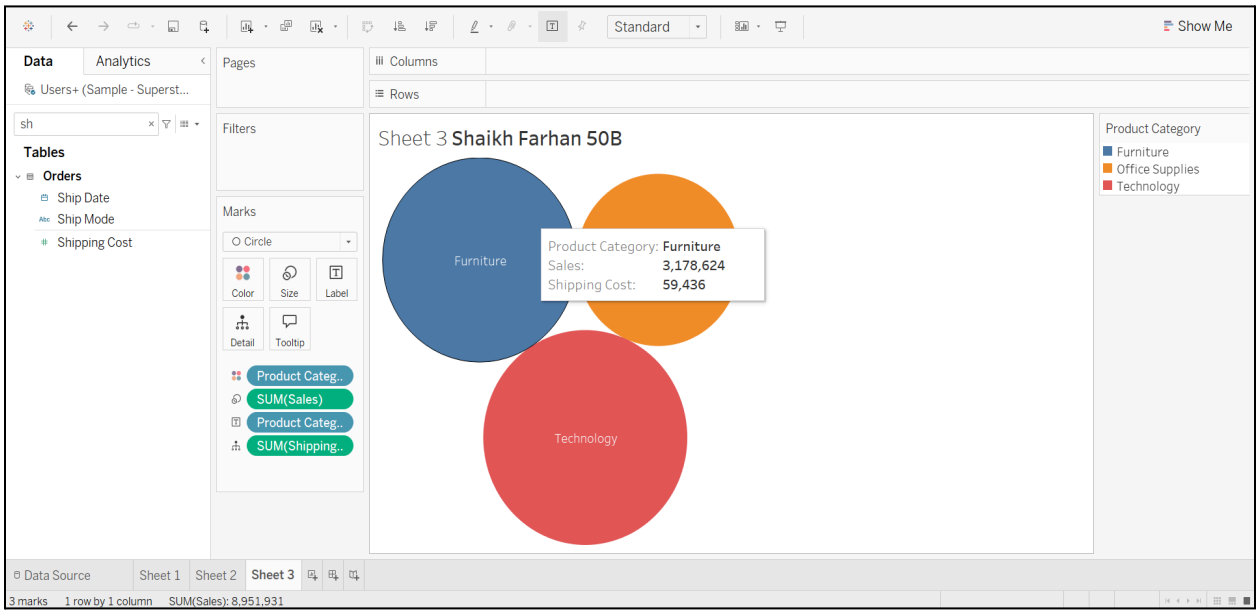
Ans:

Packed bubble chart showing each Product Department as a colored bubble

Step 1: Drag Product Category into Label and Color in Marks Section.

Step 2: Drag Shipping Cost into the Marks section to get information about every department.

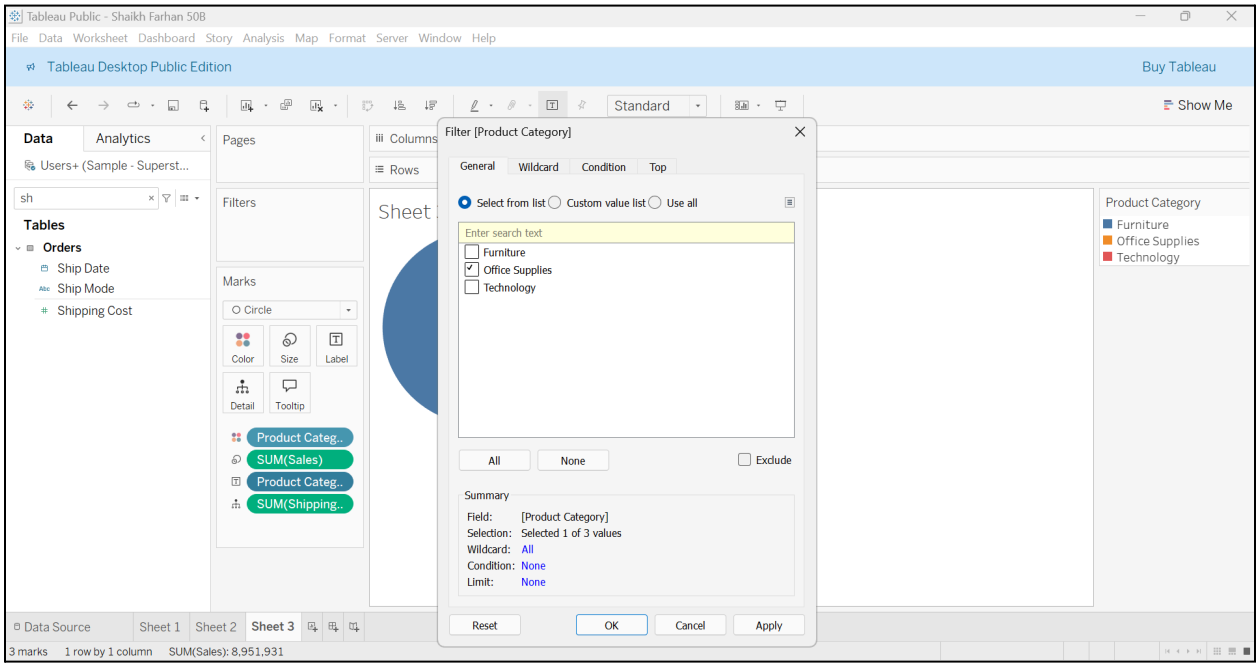


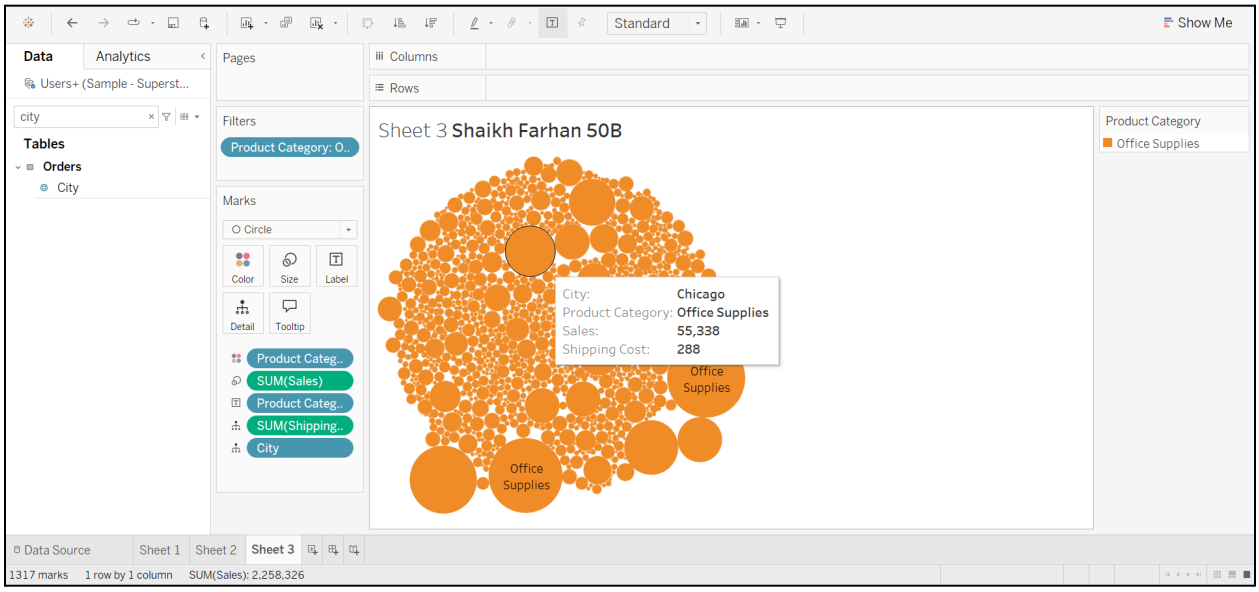


Q7: Use the same visualization created for Question #6. What was the shipping cost of Office Supplies for Xerox 1905 in the Home Customer Segment in Cambridge?

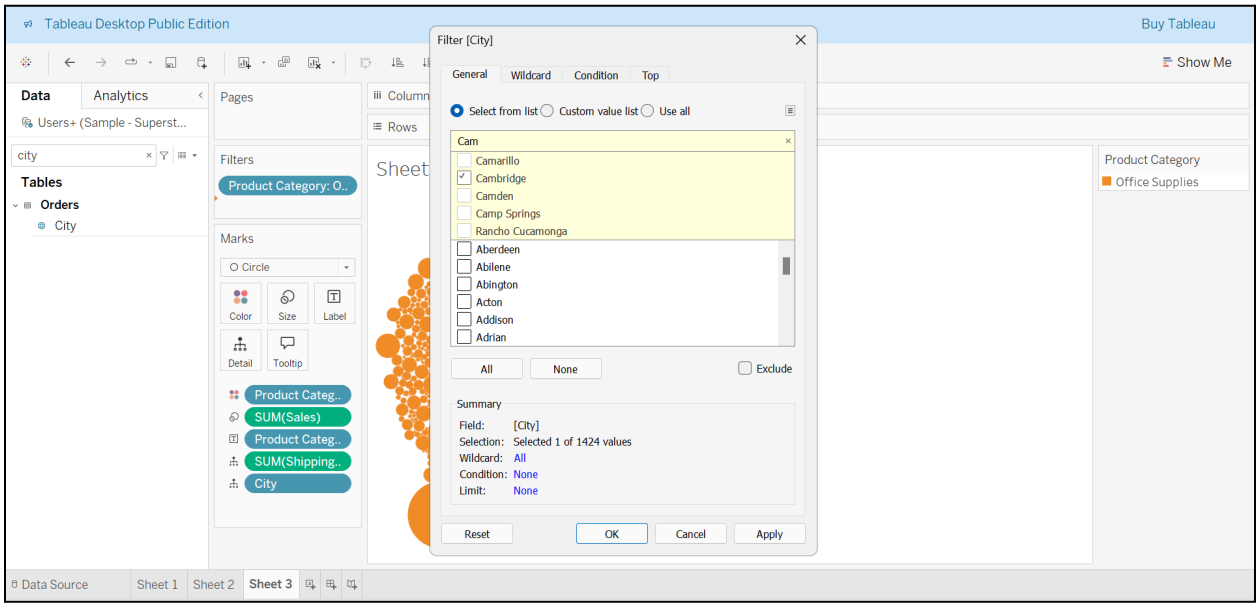
Ans:

Apply Product Category Filter as Office Supplies.

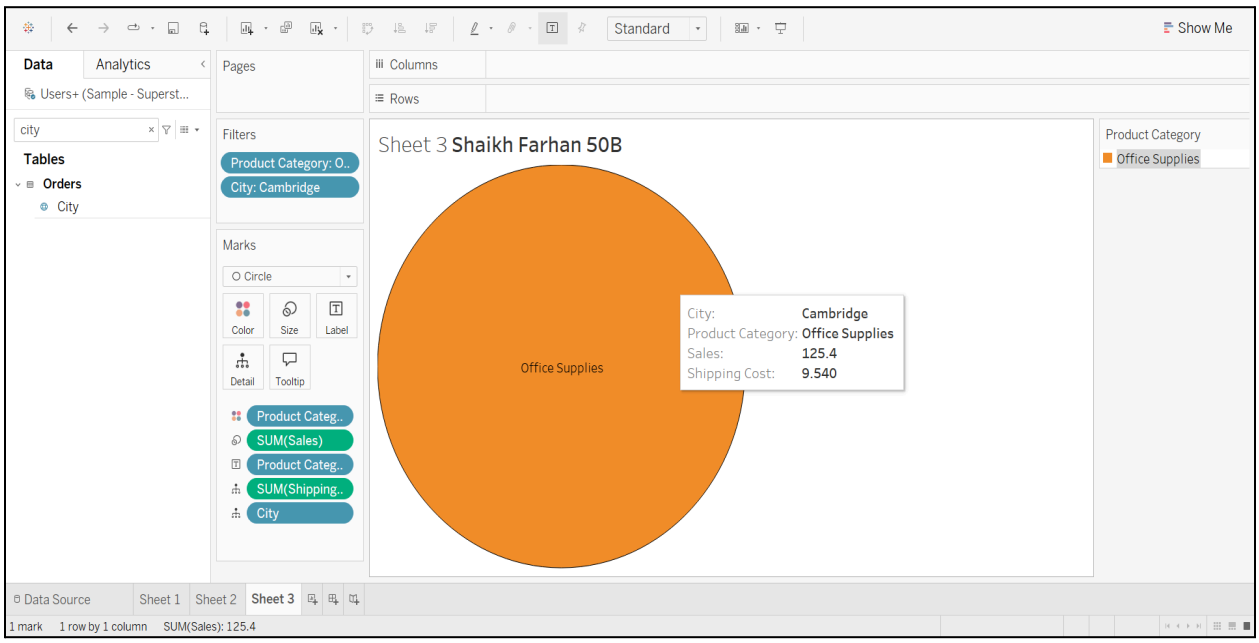




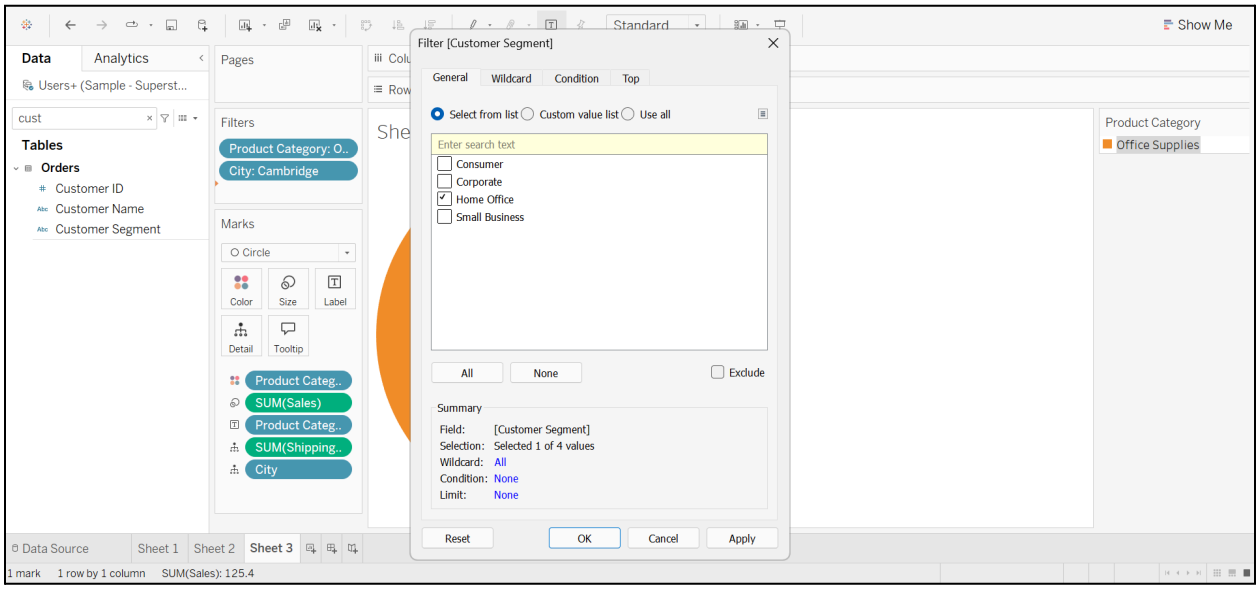
Select City as a Filter and select Cambridge.



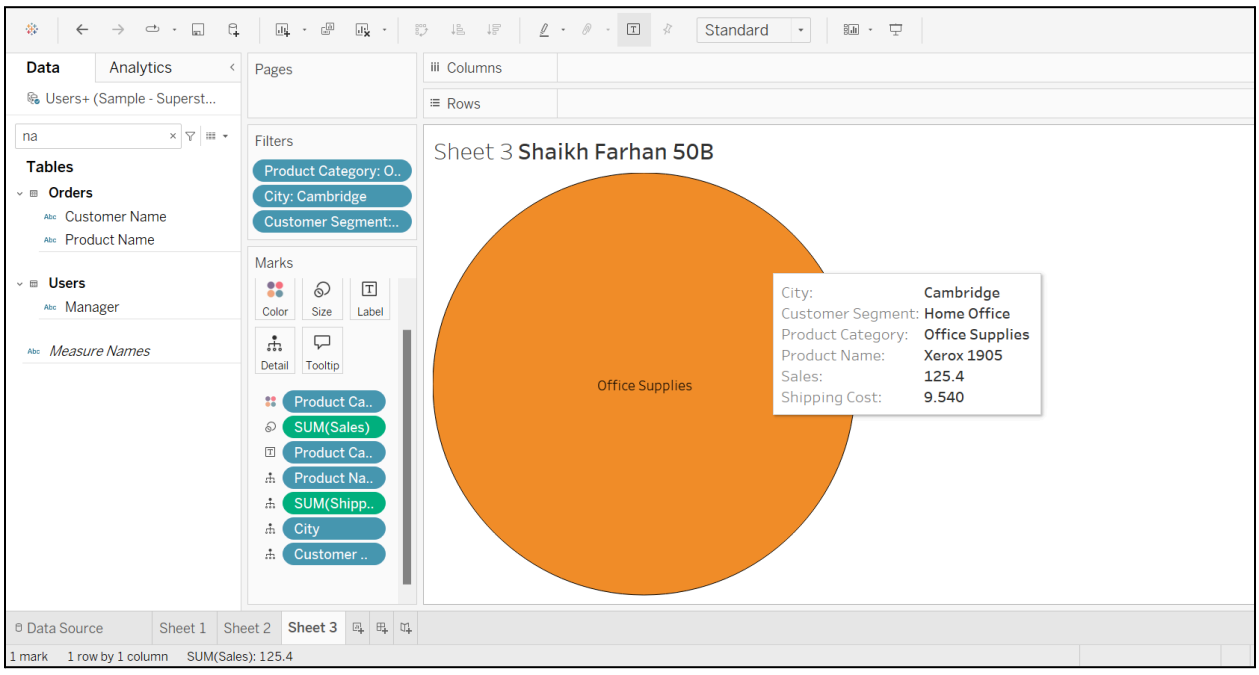
Apply Filter.



Select Customer Segment as a filter and select Home Office.



Apply filter to get the result.



Conclusion:

From this practical I have learned various data visualization techniques using tableau.

<table>
<tr><td colspan="4">Name of Student: Shaikh Farhan Ahmed</td></tr>
<tr><td colspan="2">Roll Number: 50</td><td colspan="2">LAB Assignment Number: 10</td></tr>
<tr><td colspan="4">Title of LAB Assignment: Implementation and analysis of Reports & Dashboards using tableau</td></tr>
<tr><td colspan="2">DOP: 18 – 11 – 2024</td><td colspan="2">DOS: 28 – 11 – 2024</td></tr>
<tr><td>CO Mapped:
CO5</td><td>PO Mapped:
PO1, PO2, PO3, PO5, PO6, PO7, PO8, PO11,
PSO1, PSO2</td><td>Faculty Signature:</td><td>Marks:</td></tr>
</table>

Practical No 10

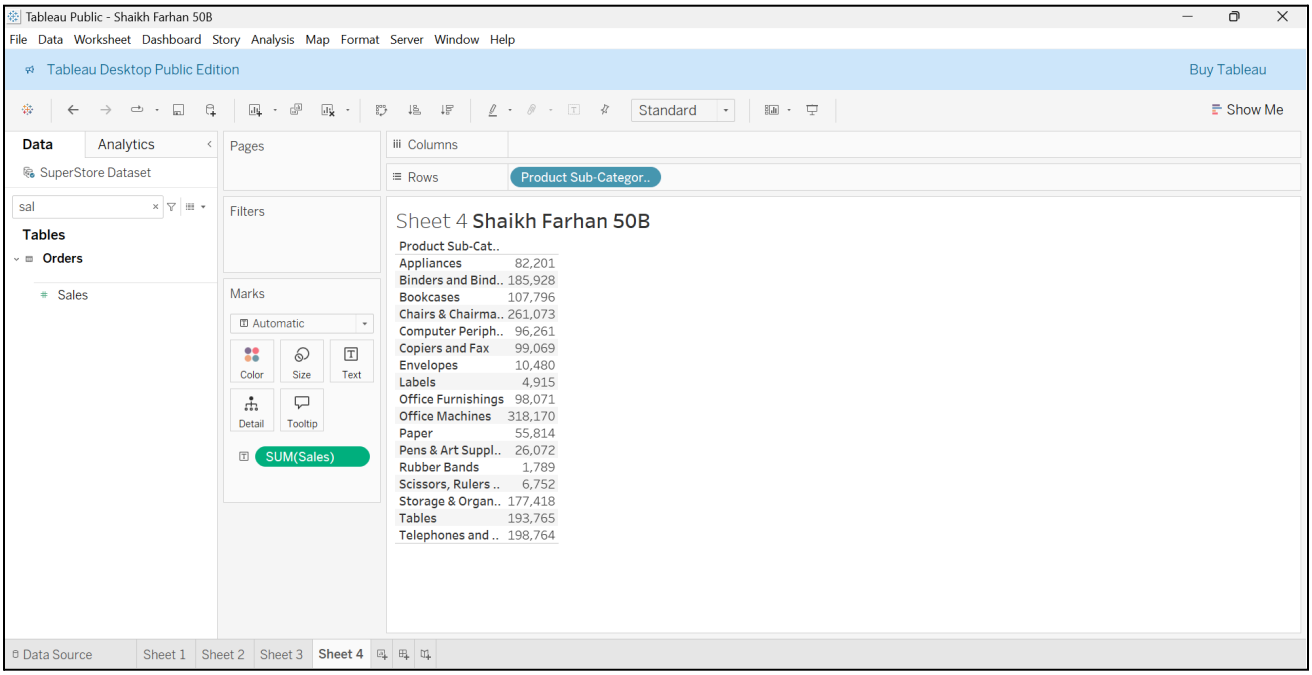
AIM: Implementation and analysis of Reports & Dashboards using tableau

Add SuperStore Dataset in Tableau.

1) Prepare a report showing product category sales wise.

Step 1: Drag Product Sub Category in rows shelf
Step 2: Drag Sales into Marks Section change it to Text.

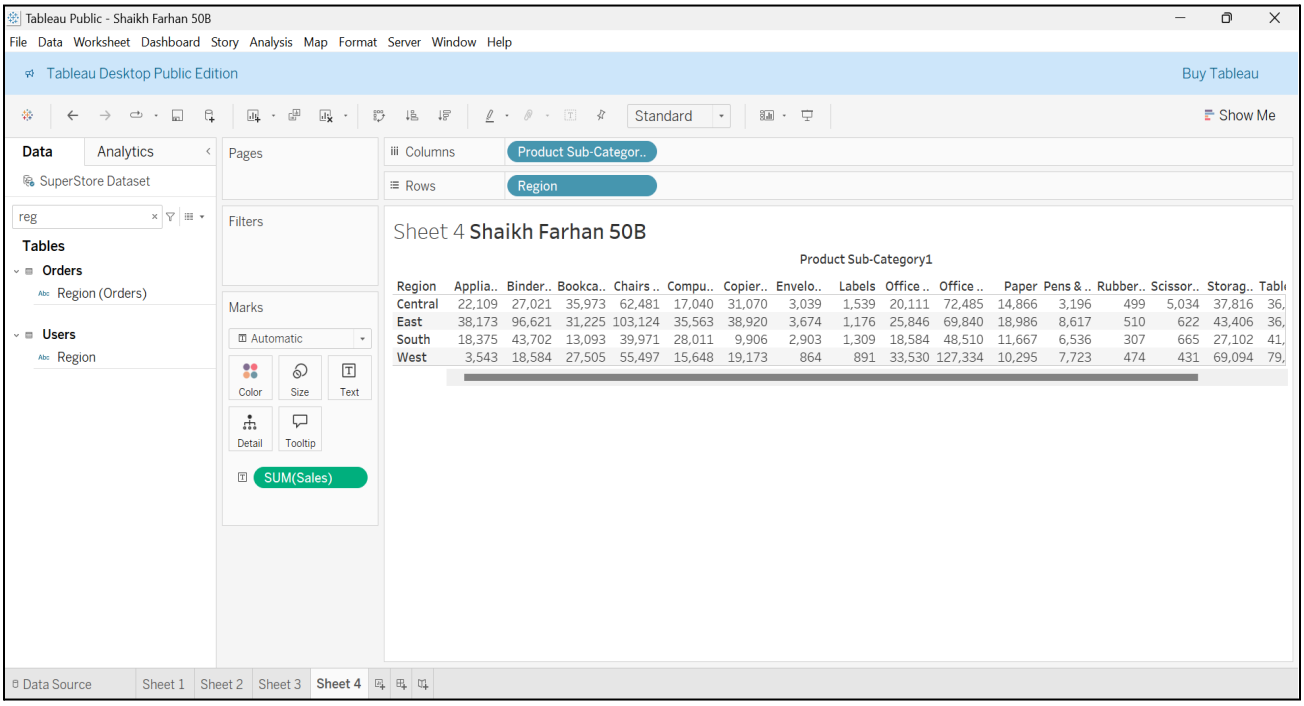
Output:



2) Report showing region wise and product wise sales.

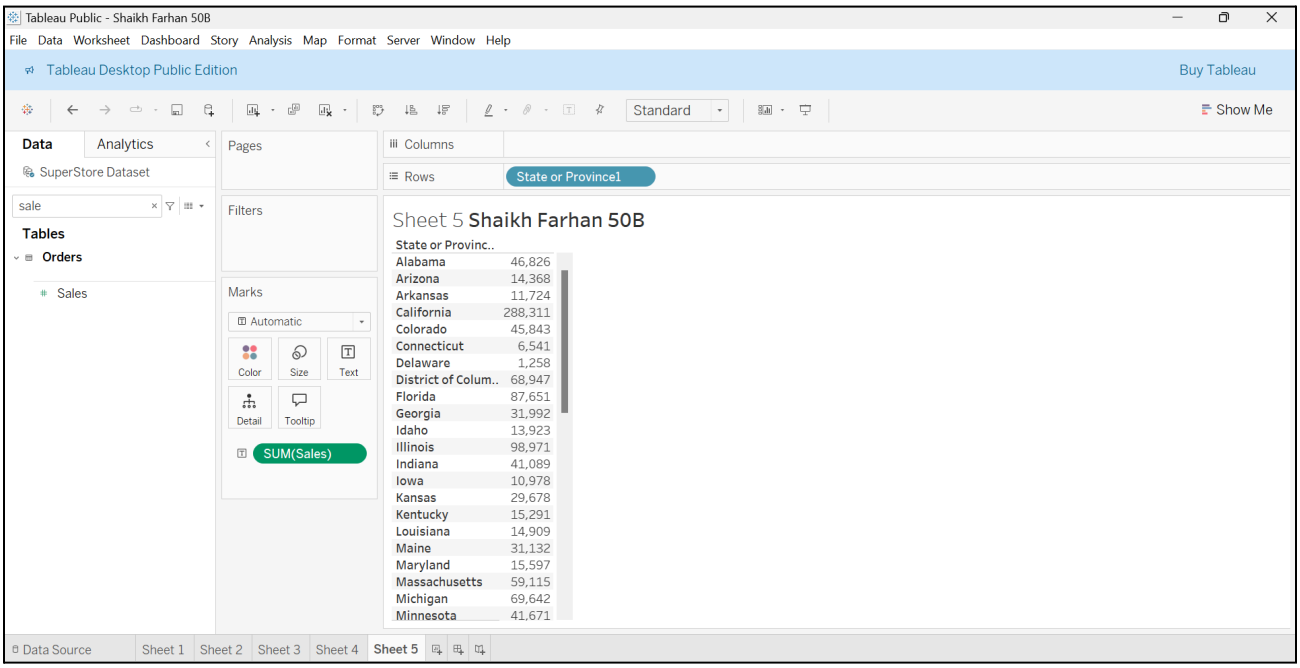
Step 1: Drag Product Sub Category in Columns shelf
Step 2: Drag Region in Rows shelf
Step 3: Drag Sales into Marks Section change it to Text.

Output:

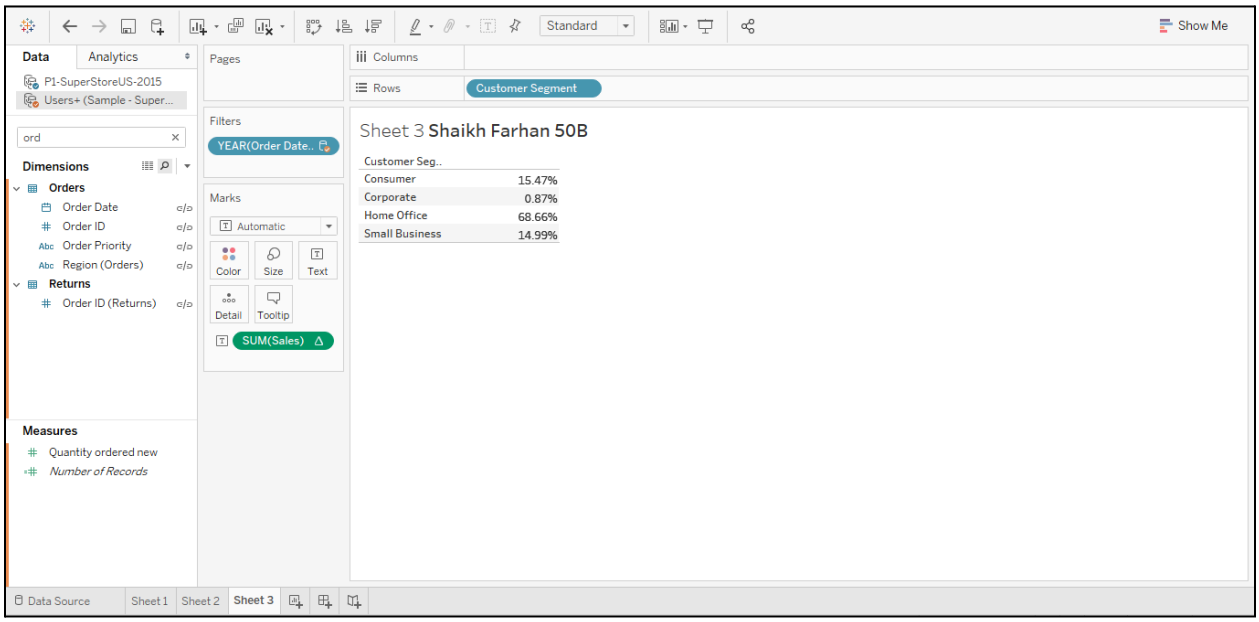


3) Report showing state wise sales.

Step 1: Drag State or Province in Rows Shelf
Step 2: Step 3: Drag Sales into Marks Section change it to Text.



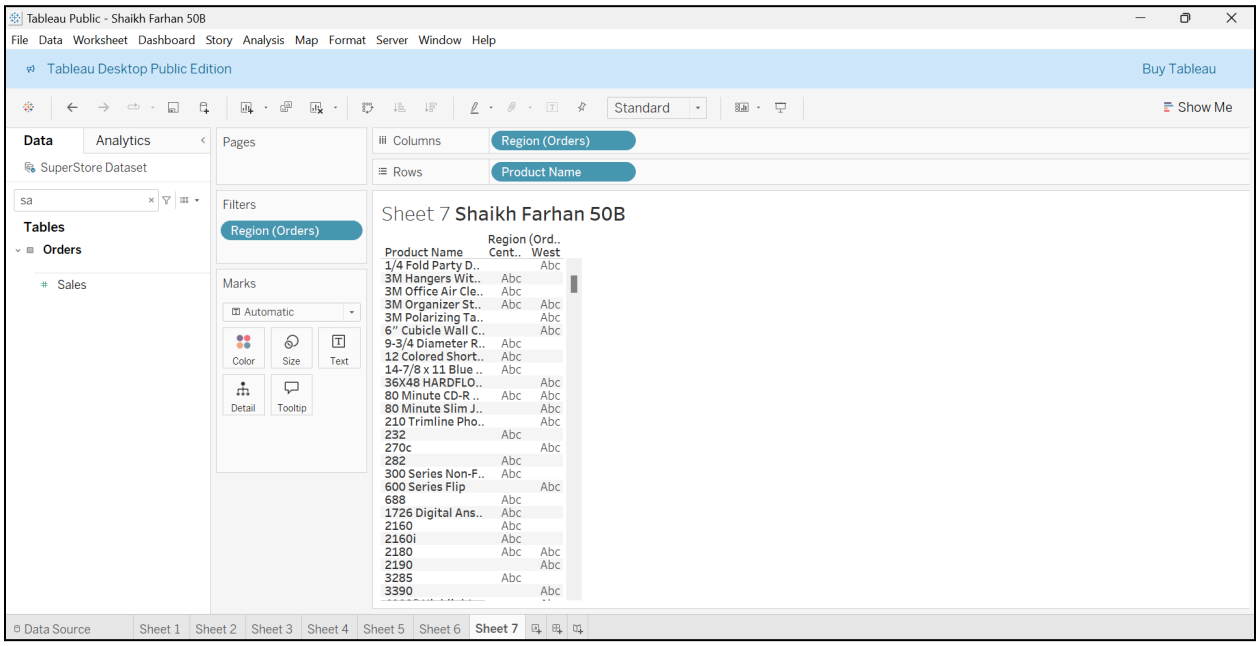
4) What is the percent of total Sales for the ‘Home Office’ Customer Segment in July of 2014?

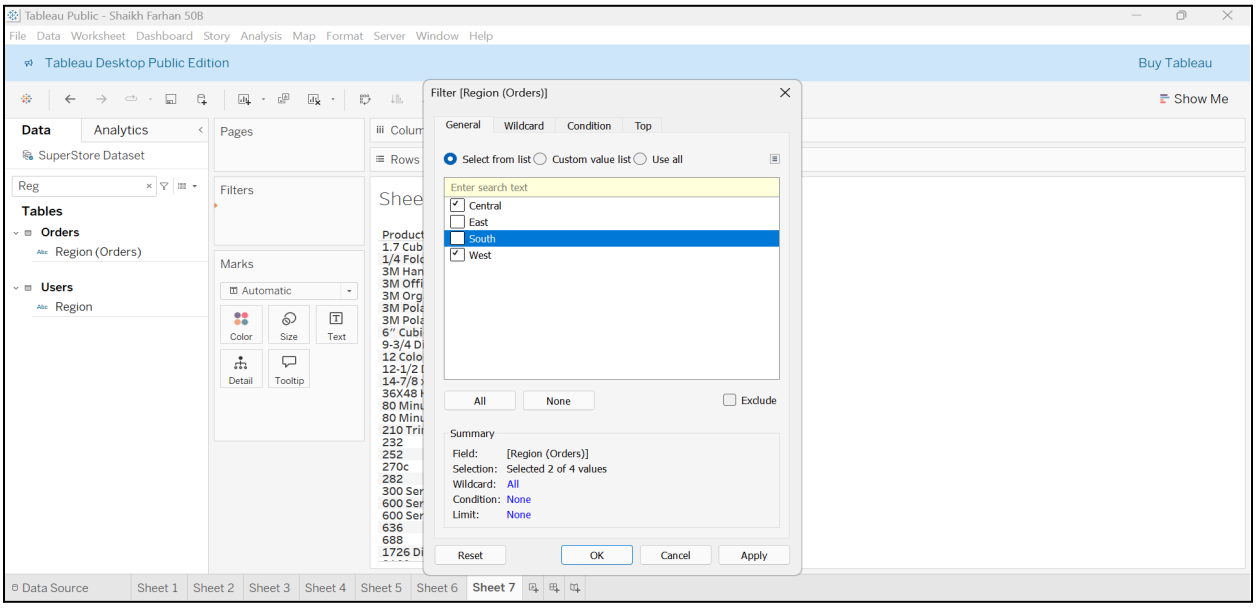


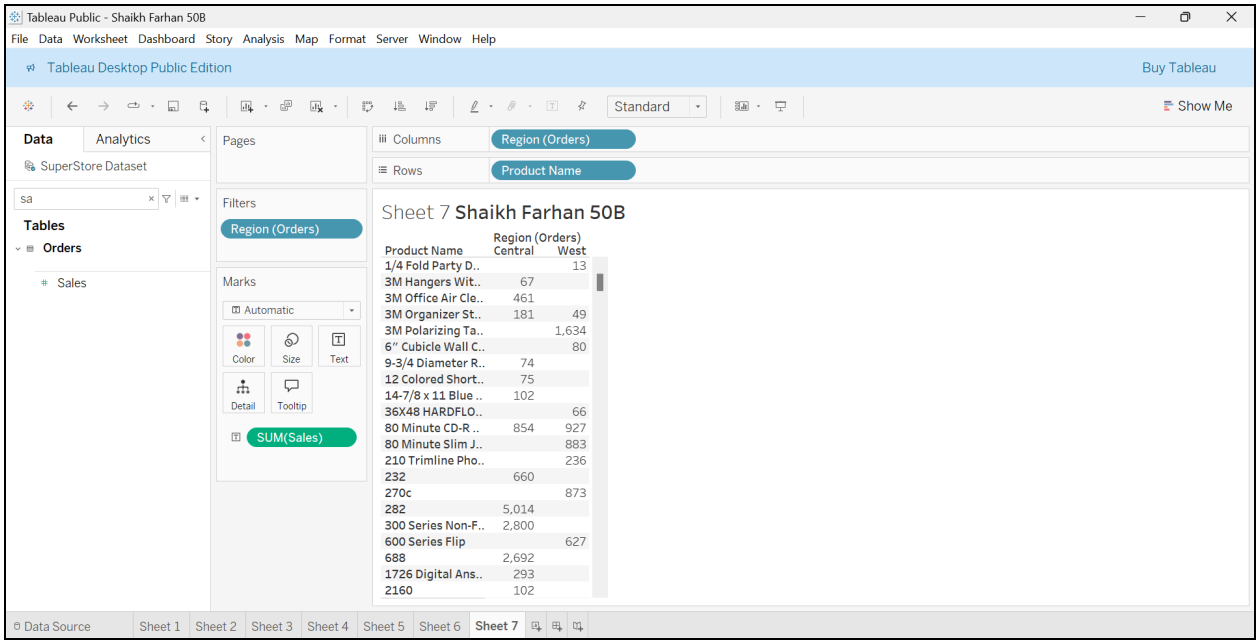
5. Find the top 10 Product Names by Sales within each region. Which product is ranked #2 in both the Central & West regions in 2015?

Drag “Product Name” dimension from data panel window to Row Shelf and then add an “Order Date” on the Filter shelf and select “Year” of Order date as 2015.After that, put the region onFilter shelf and select “Central” and “West” checkbox. Also, put a copy of the region to the Column shelf as well.

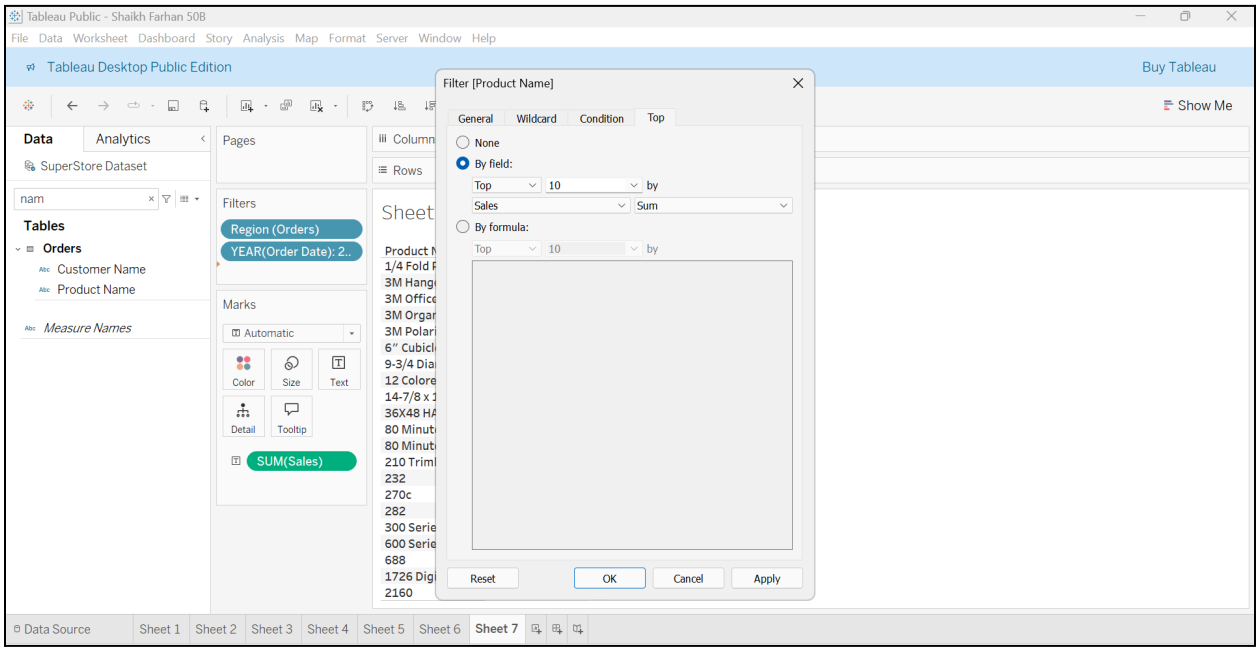
Output:

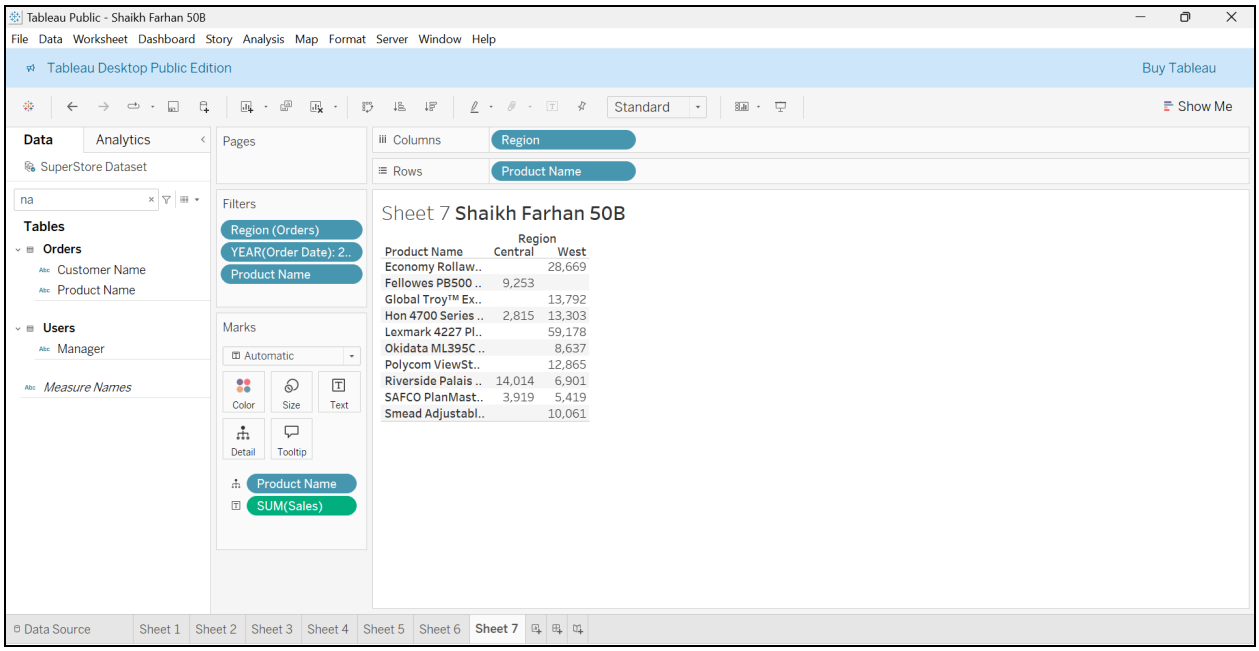




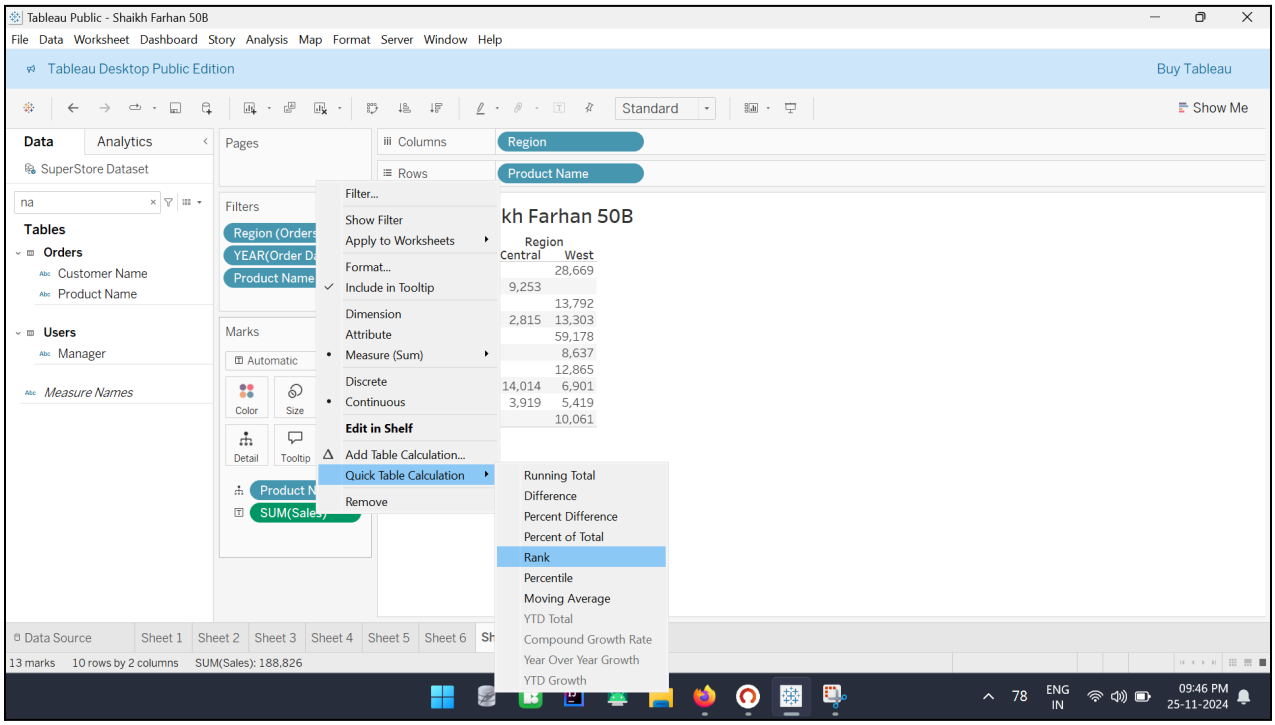


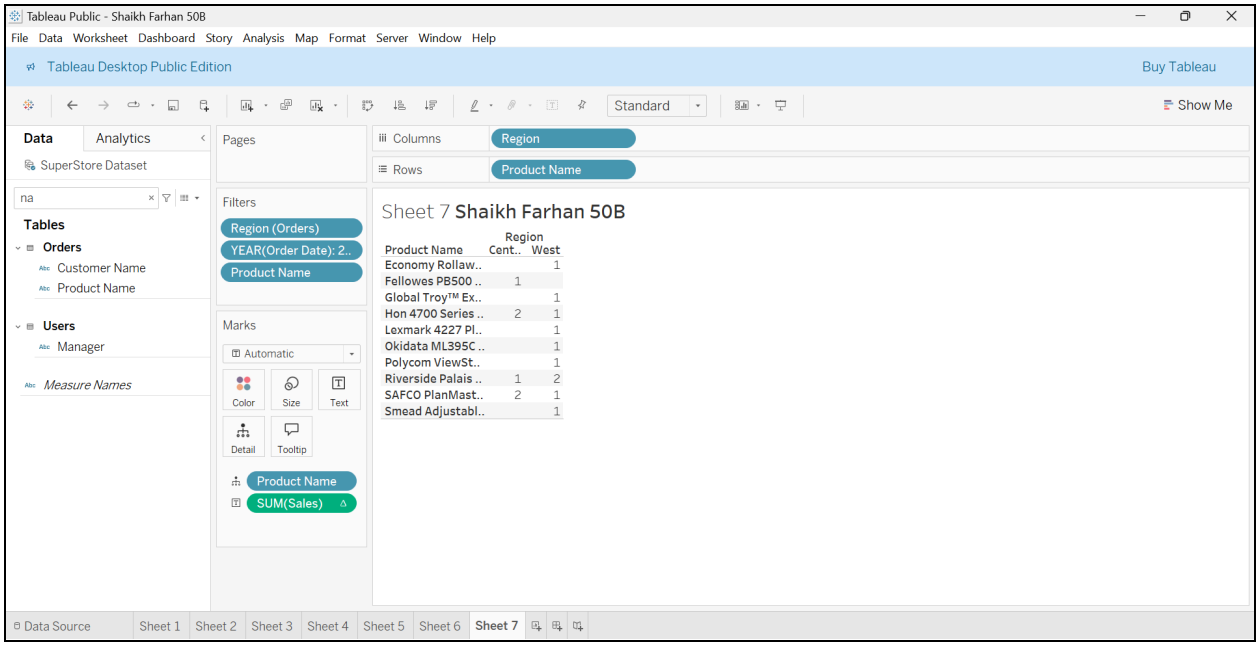
Drag a Sales measure to the text label. So for getting the Top 10 “product name” by sales, we need to add the “Product name” on Filter shelf. Once the Filter Pop up is open, Select “TOP” tab >By Field > Top 10 by Sum (Sales).





Right click on the aggregated Sales measure and click on arrow sign then select Quick Table Calculation > Rank. As the default addressing is Table across, please change it into Table Down (Compute using -> Table Down).





Conclusion:

From this practical I have learned Implementation and analysis of Reports and Dashboards using tableau.