Exp no. : 1 **INSTALLATION OF R AND R-STUDIO**
Date: 04-01-23

AIM:-

To install R and R-Studio

PROCEDURE:-

INSTALLATION OF R

Step 1: Go to CRAN R project website - https://cran.r-project.org/

Step 2: Click on the **Download R for Windows** link.

Step 3: Click on the **base** subdirectory link or **install R for the first time** link.

Step 4: Click **Download R 4.2.1 for Windows** and save the executable .exe file.

Step 5: Run the **.exe** file and follow the installation instructions.

Step 6: Select the desired language and then click **Next**.

Step 7: Read the license agreement and click **Next**.

Step 8: Select the components you wish to install . Click **Next**.

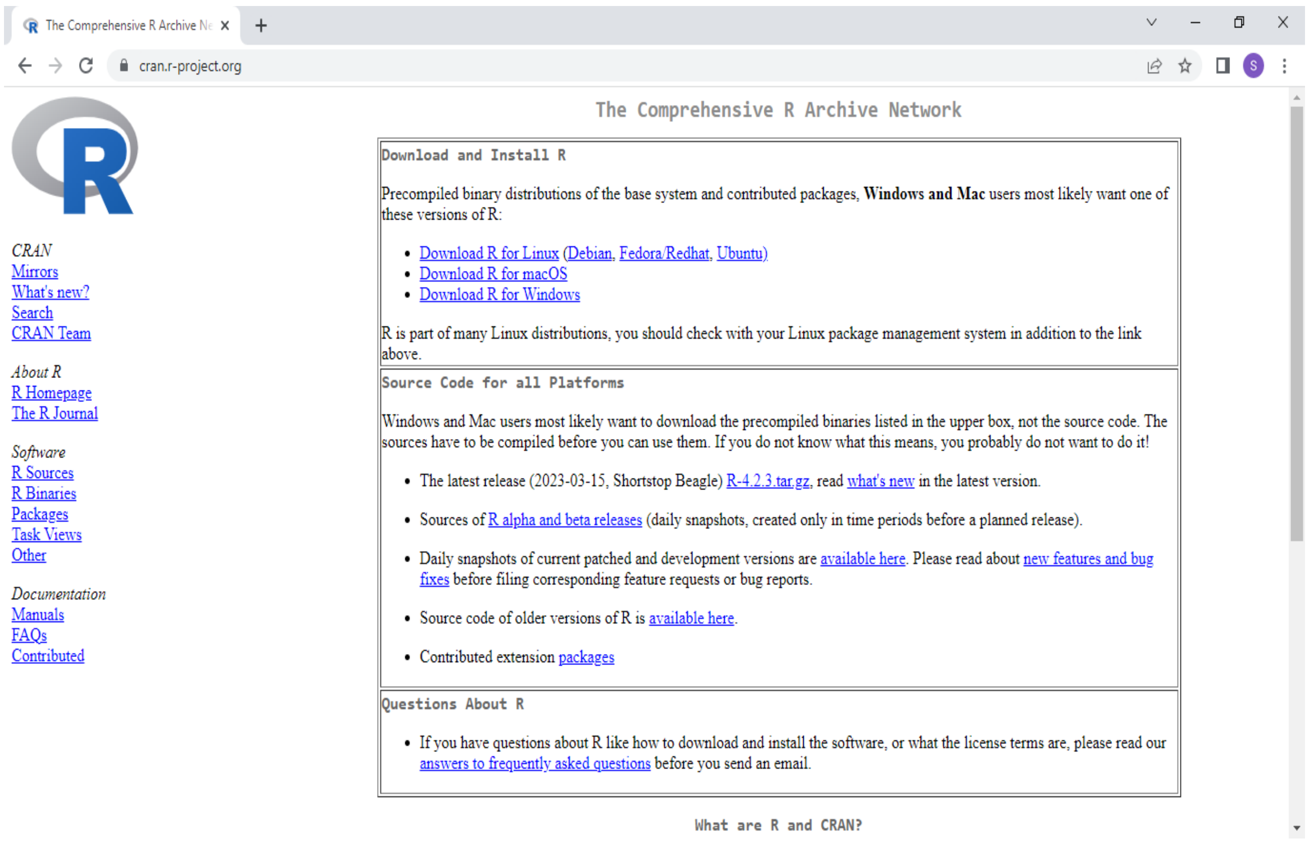
Step 9: Browse the folder/path you wish to install R into and then confirm by clicking **Next**.

Step 10: Select additional tasks like creating desktop shortcuts etc. then click **Next**.

Step 11: Wait for the installation process to complete.

Step 12: Click on **Finish** to complete the installation.

OUTPUT:-



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Search
CRAN Team

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Task Views
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

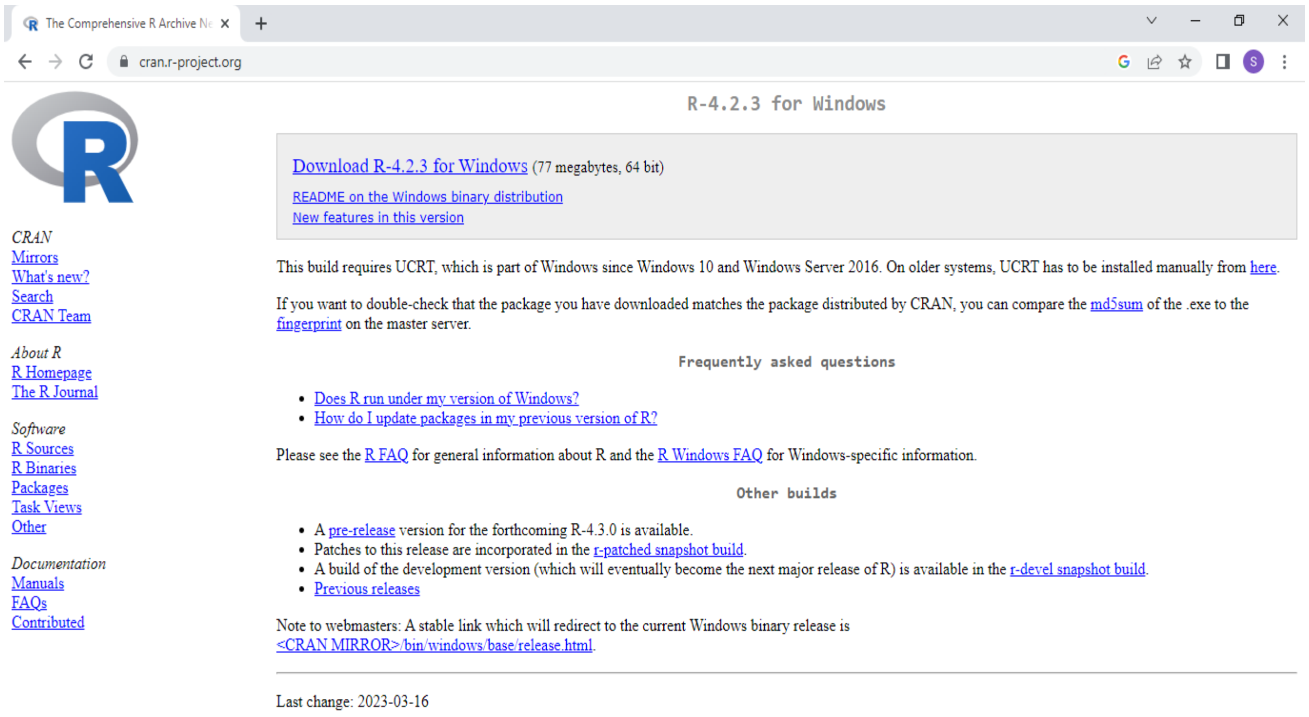
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2023-03-15, Shortstop Beagle) R-4.2.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

What are R and CRAN?



R-4.2.3 for Windows

CRAN
Mirrors
What's new?
Search
CRAN Team

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Task Views
Other

Documentation
Manuals
FAQs
Contributed

Download R-4.2.3 for Windows (77 megabytes, 64 bit)
README on the Windows binary distribution
New features in this version

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from here.

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- A pre-release version for the forthcoming R-4.3.0 is available.
- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.html.

Last change: 2023-03-16

INSTALLATION OF R-STUDIO

Step 1: With R-base installed, let's move on to installing R-Studio. To begin, go to download RStudio - https://posit.co/downloads and click on the download button for **R-Studio desktop**.

Step 2: Click on the link for the windows version of R-Studio and save the .exe file.

Step 3: Run the .exe and follow the installation instructions.

Step 4: Click **Next** on the welcome window.

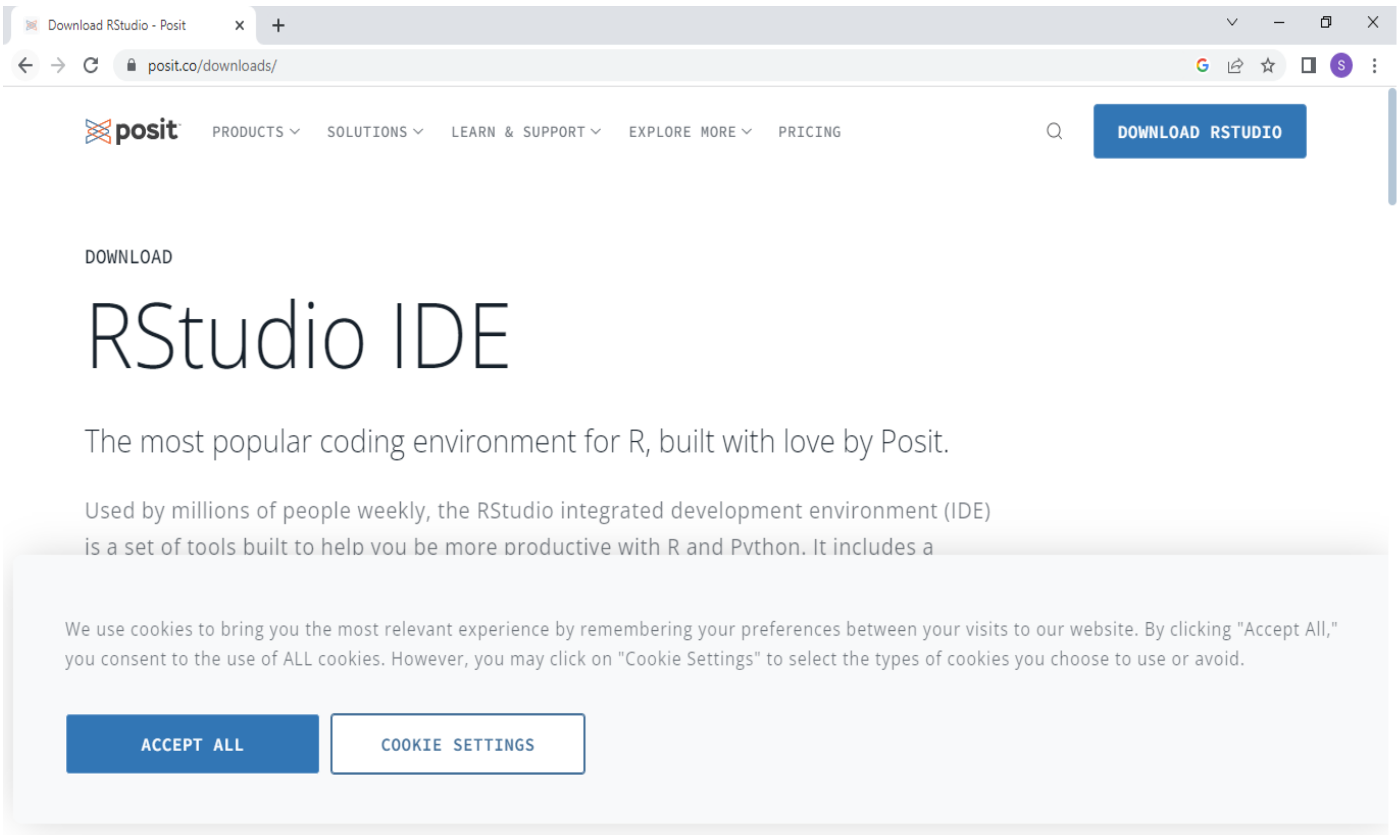
Step 5: Enter/browse the path to the installation folder and click **Next** to proceed.

Step 6: Select the folder for the start menu shortcut and then click **Next**.

Step 7: Wait for the installation process to complete.

Step 8: Click on Finish to complete the installation.

OUTPUT:-



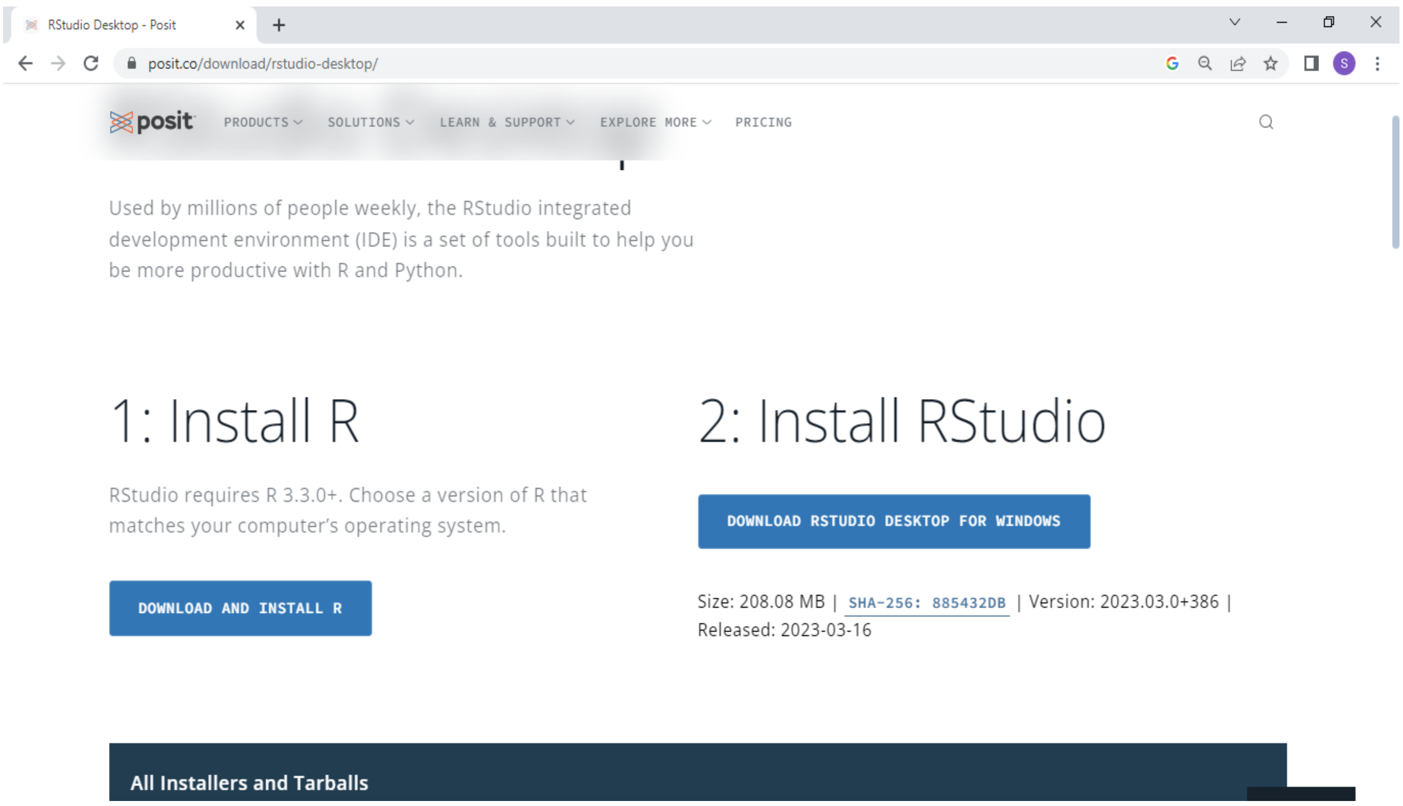
DOWNLOAD

RStudio IDE

The most popular coding environment for R, built with love by Posit.

Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python. It includes a

We use cookies to bring you the most relevant experience by remembering your preferences between your visits to our website. By clicking "Accept All," you consent to the use of ALL cookies. However, you may click on "Cookie Settings" to select the types of cookies you choose to use or avoid.



Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python.

1: Install R

RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

2: Install RStudio

Size: 208.08 MB | SHA-256: 885432DB | Version: 2023.03.0+386 | Released: 2023-03-16

All Installers and Tarballs

Exp no. : 2 WORKING WITH R PACKAGES
Date: 11-01-23

AIM:-

To install, load and work with packages

PACKAGES

A package is basically just a big collection of functions, data sets and other R objects that are all grouped together under a common name. A package must be installed before it can be loaded. A package must be loaded before it can be used.

Install R Packages from CRAN

install.packages ( ) - This function is used to install a required package in the R programming language.

Example :- To install ggplot2 package

```
install.packages(“ggplot2”)
```

Uninstall R Packages

remove.packages ( ) - This function is used to uninstall apackage in the R programming language.

Example :- To uninstall ggplot2 package

```
remove.packages(“ggplot2”)
```

Loading of R Packages

library ( ) - It is used to load and list all the packages in the R Programming language.

Example :- To load ggplot2 package

```
library ( ggplot2 )
```

To list all the packages installed

```
library ( )
```

Updating R Packages

old.packages ( ) - It is used to check which packages need an update in R.

Example :- To check an update

```
old.packages( )
```

update.packages ( ) - It is used to update all the packages in R .

Example :- To update Packages

```
update.packages( )
```

Listing the Packages that are Installed

install.packages ( ) - It is used to list out all packages which are installed in computer .

Example :- To list out installed packages

```
installed.packages( )
```

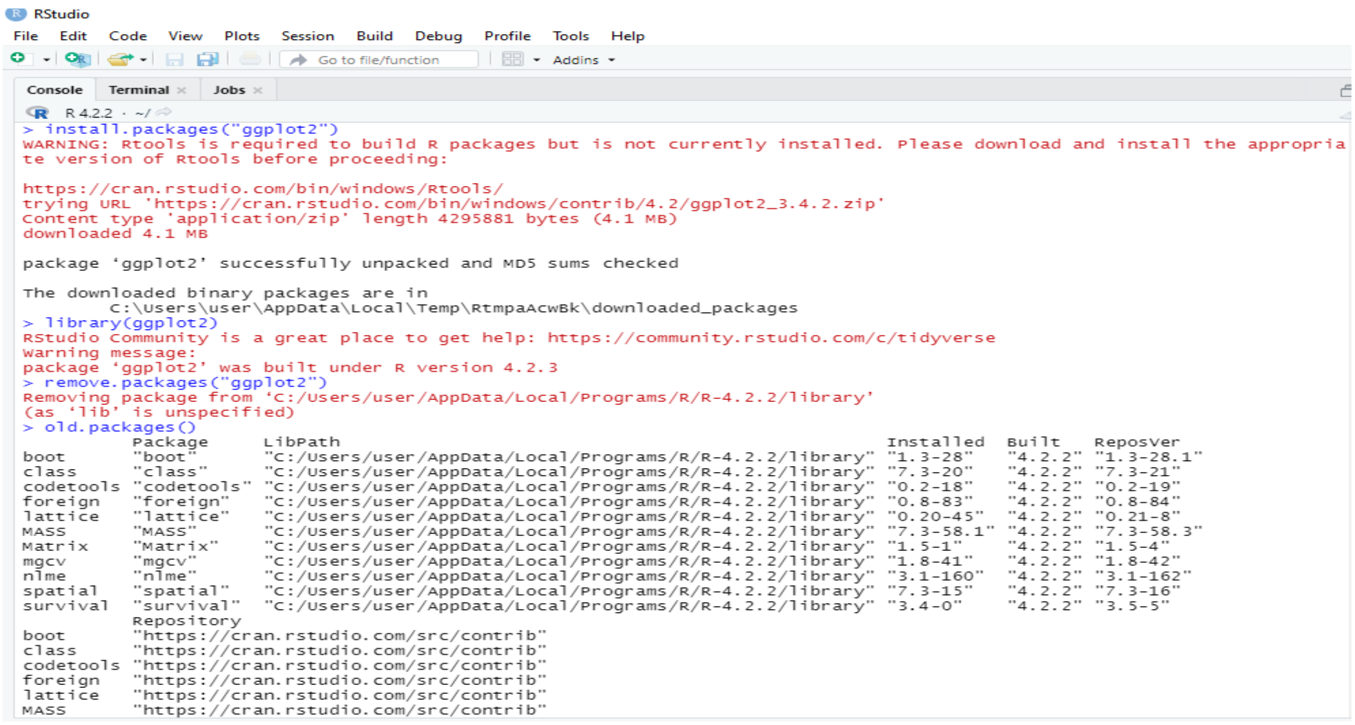
Get help pages about Packages

Help ( ) and (?)- They provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.

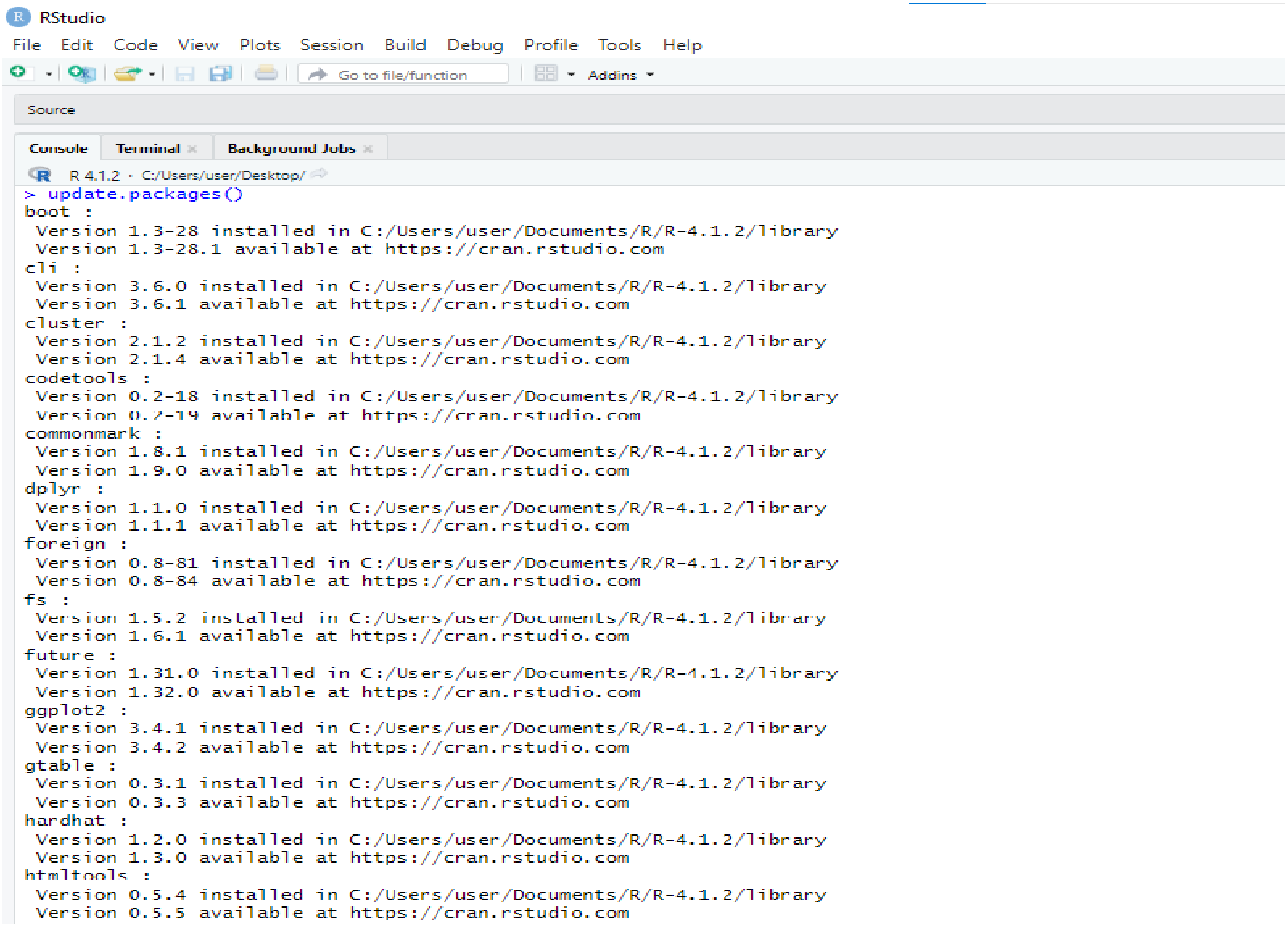
Example :- To get more description about ggplot2

```
help("ggplot2")  and  ?ggplot2
```

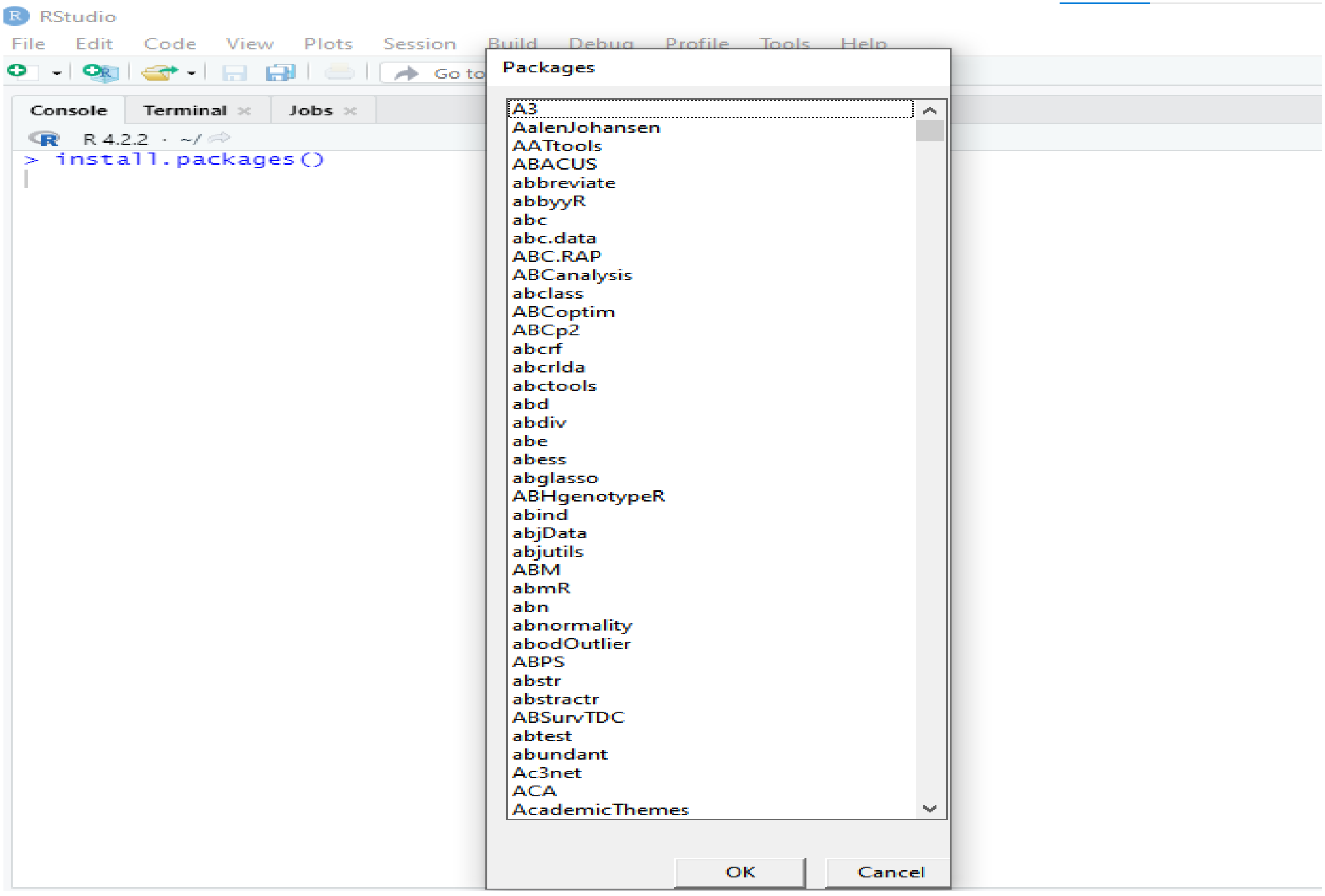
OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console
Terminal
Jobs
R 4.2.2 · ~/

```
> install.packages("ggplot2")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropria
te version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/ggplot2_3.4.2.zip'
Content type 'application/zip' length 4295881 bytes (4.1 MB)
downloaded 4.1 MB

package ‘ggplot2’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\user\AppData\Local\Temp\RtmpaAcwBk\downloaded_packages
> library(ggplot2)
RStudio Community is a great place to get help: https://community.rstudio.com/c/tidyverse
Warning message:
package ‘ggplot2’ was built under R version 4.2.3
> remove.packages("ggplot2")
Removing package from ‘C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library’
(as ‘lib’ is unspecified)
> old.packages()
          Package     LibPath                                          Installed  Built   ReposVer
boot      "boot"      "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "1.3-28"   "4.2.2" "1.3-28.1"
class     "class"     "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "7.3-20"   "4.2.2" "7.3-21"
codetools "codetools" "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "0.2-18"   "4.2.2" "0.2-19"
foreign   "foreign"   "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "0.8-83"   "4.2.2" "0.8-84"
lattice   "lattice"   "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "0.20-45"  "4.2.2" "0.21-8"
MASS      "MASS"      "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "7.3-58.1" "4.2.2" "7.3-58.3"
Matrix    "Matrix"    "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "1.5-1"    "4.2.2" "1.5-4"
mgcv      "mgcv"      "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "1.8-41"   "4.2.2" "1.8-42"
nlme      "nlme"      "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "3.1-160"  "4.2.2" "3.1-162"
spatial   "spatial"   "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "7.3-15"   "4.2.2" "7.3-16"
survival  "survival"  "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library" "3.4-0"    "4.2.2" "3.5-5"
          Repository
boot      "https://cran.rstudio.com/src/contrib"
class     "https://cran.rstudio.com/src/contrib"
codetools "https://cran.rstudio.com/src/contrib"
foreign   "https://cran.rstudio.com/src/contrib"
lattice   "https://cran.rstudio.com/src/contrib"
MASS      "https://cran.rstudio.com/src/contrib"
```

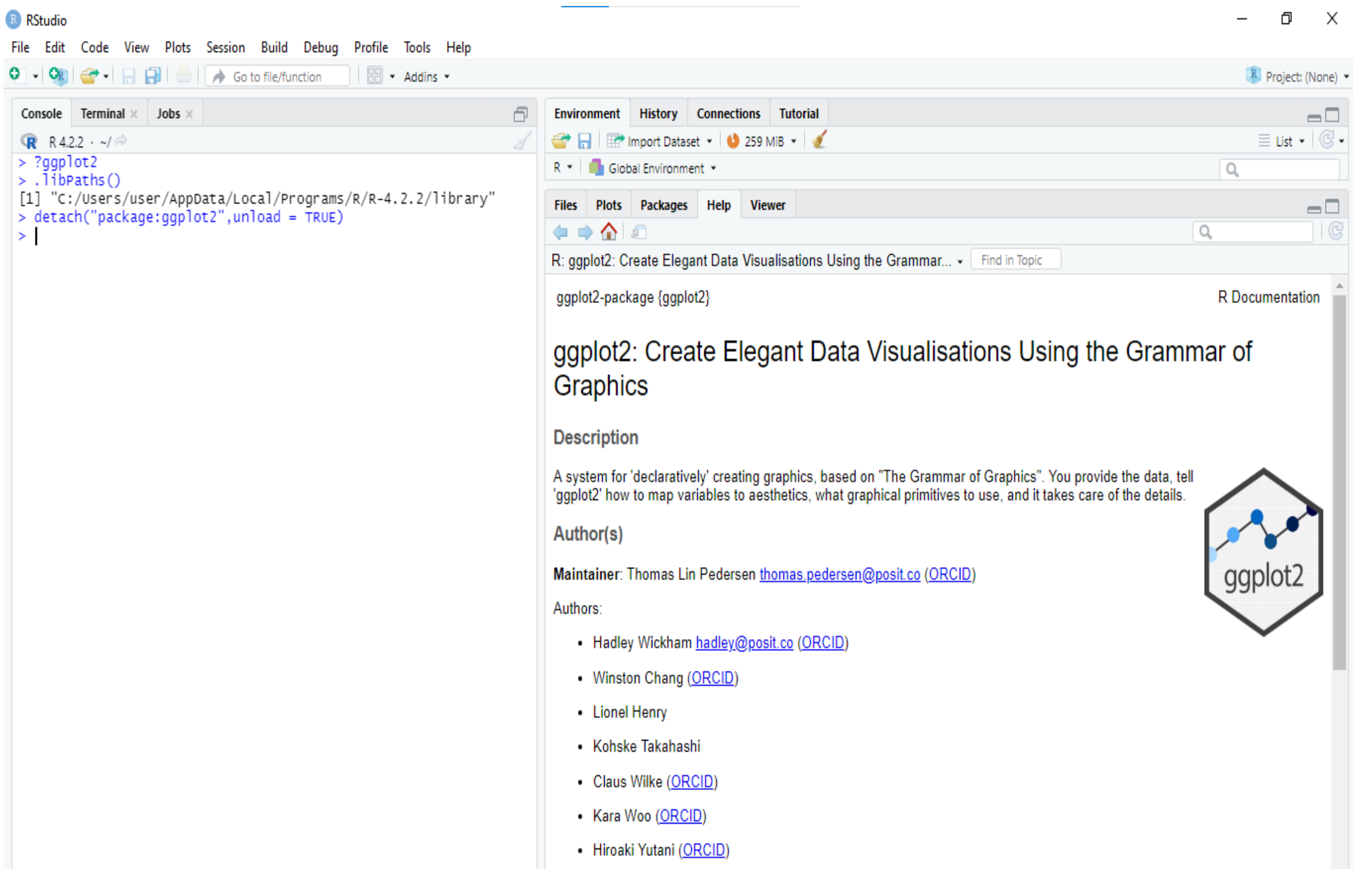

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
Console
Terminal
Background Jobs
R 4.1.2 · C:/Users/user/Desktop/

```
> update.packages()
boot :
 Version 1.3-28 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 1.3-28.1 available at https://cran.rstudio.com
cli :
 Version 3.6.0 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 3.6.1 available at https://cran.rstudio.com
cluster :
 Version 2.1.2 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 2.1.4 available at https://cran.rstudio.com
codetools :
 Version 0.2-18 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 0.2-19 available at https://cran.rstudio.com
commonmark :
 Version 1.8.1 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 1.9.0 available at https://cran.rstudio.com
dplyr :
 Version 1.1.0 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 1.1.1 available at https://cran.rstudio.com
foreign :
 Version 0.8-81 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 0.8-84 available at https://cran.rstudio.com
fs :
 Version 1.5.2 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 1.6.1 available at https://cran.rstudio.com
future :
 Version 1.31.0 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 1.32.0 available at https://cran.rstudio.com
ggplot2 :
 Version 3.4.1 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 3.4.2 available at https://cran.rstudio.com
gtable :
 Version 0.3.1 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 0.3.3 available at https://cran.rstudio.com
hardhat :
 Version 1.2.0 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 1.3.0 available at https://cran.rstudio.com
htmltools :
 Version 0.5.4 installed in C:/Users/user/Documents/R/R-4.1.2/library
 Version 0.5.5 available at https://cran.rstudio.com
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Console
Terminal
Jobs
R 4.2.2 · ~/
Packages
OK
Cancel

```
> install.packages()
```

A3
AalenJohansen
AATtools
ABACUS
abbreviate
abbyyR
abc
abc.data
ABC.RAP
ABCanalysis
abclass
ABCoptim
ABCp2
abcrf
abcrlda
abctools
abd
abdiv
abe
abess
abglasso
ABHgenotypeR
abind
abjData
abjutils
ABM
abmR
abn
abnormality
abodOutlier
ABPS
abstr
abstractr
ABSurvTDC
abtest
abundant
Ac3net
ACA
AcademicThemes



```
> ?ggplot2
> .libPaths()
[1] "C:/Users/user/AppData/Local/Programs/R/R-4.2.2/library"
> detach("package:ggplot2",unload = TRUE)
> 
```

ggplot2-package {ggplot2} R Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

Description

A system for 'declaratively' creating graphics, based on "The Grammar of Graphics". You provide the data, tell 'ggplot2' how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Author(s)

Maintainer: Thomas Lin Pedersen thomas.pedersen@posit.co (ORCID)

Authors:

- Hadley Wickham hadley@posit.co (ORCID)
- Winston Chang (ORCID)
- Lionel Henry
- Kohske Takahashi
- Claus Wilke (ORCID)
- Kara Woo (ORCID)
- Hiroaki Yutani (ORCID)

Exp no. : 3 **BUILT - IN FUNCTION**

Date : 20-01-23

AIM :-

To show the working of built-in function.

BUILT - IN FUNCTIONS:-

FUNCTIONS	DESCRIPTION
sum ( )	It returns the sum of all the input vector.
prod ( )	It returns the multiplication result of all the input vector.
max ( )	It returns the maximum value of input vector.
min ( )	It returns the minimum value of input vector.
unique ( )	It returns the unique value.
sort ( )	It returns sorted value from input vector.
rev ( )	It returns reversed order of input vector.
rbind ( )	It combines vector and matrix row-wise.
cbind ( )	It combines vector and matrix column-wise.
setdiff ( )	It returns differences between two vectors.
cumsum ( )	It returns the cumulative sum of two vector.
abs ( )	It returns the absolute value of input vector.
sqrt ( )	It returns the square root of input vector.
ceiling ( )	It returns the smallest integer which is larger than or equal to input vector.
floor ( )	It returns the largest integer, which is smaller than or equal to input vector.
trunc ( )	It returns the truncate value of input vector.
round ( )	It returns round value of input vector.
cos ( ), sin ( ), tan ( )	It returns cos( ), sin( ), tan( ) value of input vector.
log ( )	It returns natural logarithm of input vector.
log10 ( )	It returns common logarithm of input vector.
exp ( )	It returns exponent of input vector.

1. **Create a Vector 'v' with the values 1,5,8,10,4,5,3,9,8,10,12 and do the following**

   a. **Find Sum, Mean and Product of the Vector**

   Sol :-

   ```
   v  =c(1,5,8,10,4,5,3,9,8,10,12)
   sum(v)
   mean(v)
   prod(v)
   ```

   b. **Find the minimum and the maximum of the Vector.**

   Sol :-

   ```
   min(v)
   max(v)
   ```

   c. **Sort the Vector in ascending and descending order.**

   Sol : -

   ```
   sort(v)
   sort(v,decreasing=TRUE)
   ```

   d. **List the distinct values in the vector**

   Sol :-

   ```
    unique(v)
   ```

   e. **Reverse the order of the vector.**

   Sol :-

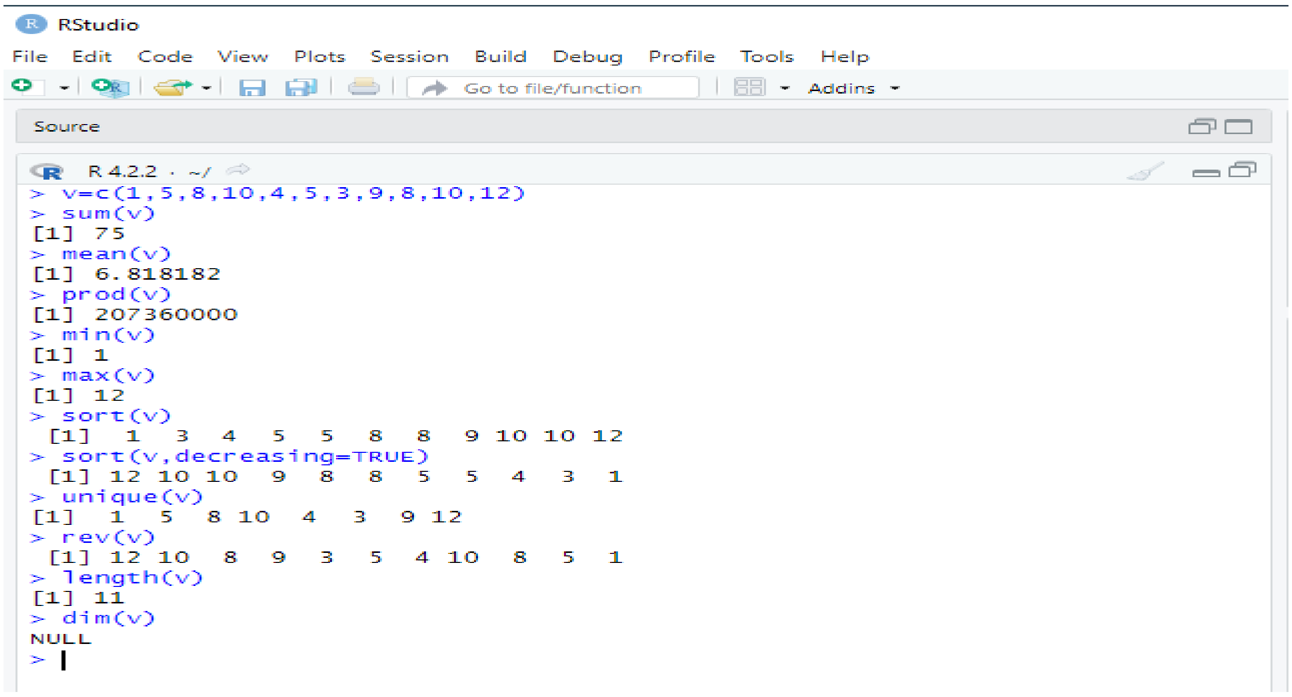
   ```
    rev(v)
   ```

   f. **Find the length and the dimension of the vector.**

   Sol :-

   ```
   length(v)
   dim(v)
   ```

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
R 4.2.2 · ~/

```
> v=c(1,5,8,10,4,5,3,9,8,10,12)
> sum(v)
[1] 75
> mean(v)
[1] 6.818182
> prod(v)
[1] 207360000
> min(v)
[1] 1
> max(v)
[1] 12
> sort(v)
 [1]  1  3  4  5  5  8  8  9 10 10 12
> sort(v,decreasing=TRUE)
 [1] 12 10 10  9  8  8  5  5  4  3  1
> unique(v)
[1]  1  5  8 10  4  3  9 12
> rev(v)
 [1] 12 10  8  9  3  5  4 10  8  5  1
> length(v)
[1] 11
> dim(v)
NULL
> 
```

2. Create two vectors as below :-

A = 0,2,4,15
B = 3,12,4,11

a. Combines these two vectors by column wise and row wise.

Sol :-

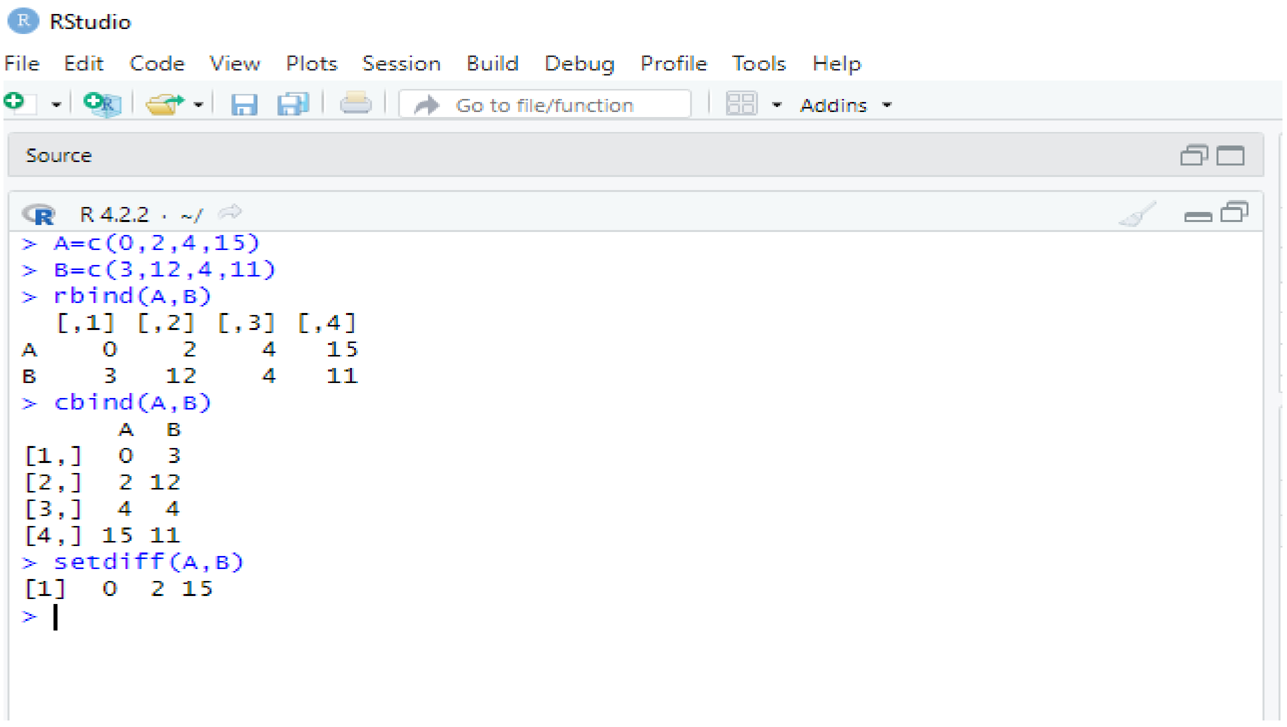
```
A = c(0,2,4,15)
B = c(3,12,4,11)
rbind(A,B)
Cbind(A,B)
```

a. Find the elements of a 'A' vector that are not in 'B' vector.

Sol :-

```
Setdiff(A,B)
```

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
R 4.2.2 · ~/

```
> A=c(0,2,4,15)
> B=c(3,12,4,11)
> rbind(A,B)
  [,1] [,2] [,3] [,4]
A    0    2    4   15
B    3   12    4   11
> cbind(A,B)
      A  B
[1,]  0  3
[2,]  2 12
[3,]  4  4
[4,] 15 11
> setdiff(A,B)
[1]  0  2 15
> 
```

3.Your cell phone bill varies from month to month.

46 33 39 37 46 30 48 32 49 35 30 48

a. **Represent the above information in a vector 'bill'**

Sol –

Bill = c(46,33,39,37,46,30,48,32,49,35,30,48)

b. **Find the total amount you spent this year on the cell phone**

Sol –

sum(Bill)

c. **Display the smallest amount you spent in a month**

Sol-

min(Bill)

d. **Display the largest amount you spent in a month**

Sol –

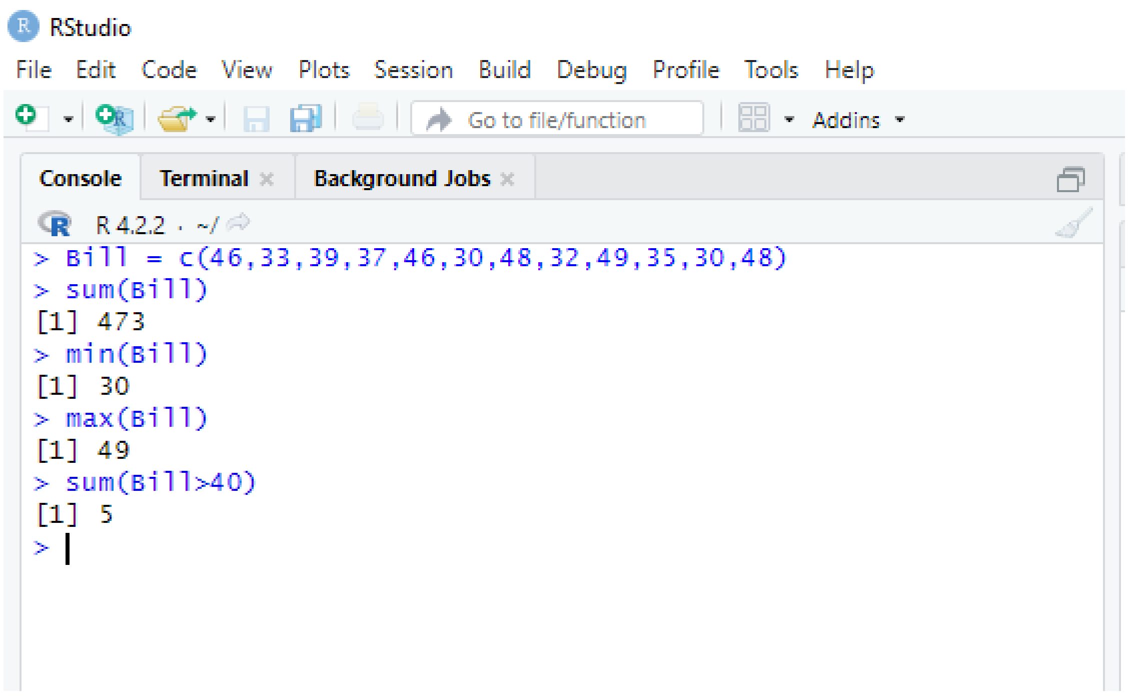
max(Bill)

e. **How many months was the amount greater than $40**

Sol-

sum(Bill>40)

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console Terminal Background Jobs
R 4.2.2 · ~/

```
> Bill = c(46,33,39,37,46,30,48,32,49,35,30,48)
> sum(Bill)
[1] 473
> min(Bill)
[1] 30
> max(Bill)
[1] 49
> sum(Bill>40)
[1] 5
> 
```

3. A survey asks people if they smoke or not. The data is

yes, no, no, yes, yes

a. Represent the above information in a vector

Sol –

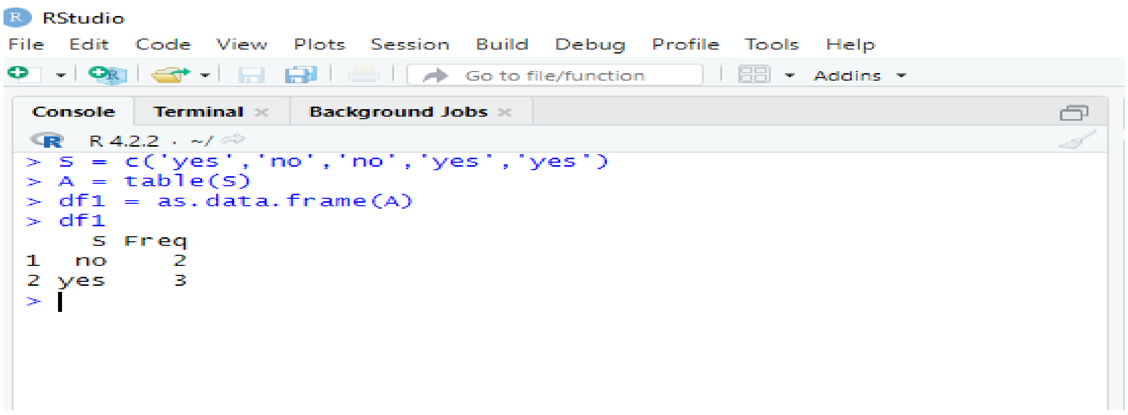
S=c("yes","no","no","yes","yes")

b) Display frequency table for above information

Sol-

A =table(S)
df1 = as.data.frame(A)

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console Terminal Background Jobs
R 4.2.2 · ~/

```
> S = c('yes','no','no','yes','yes')
> A = table(S)
> df1 = as.data.frame(A)
> df1
    S Freq
1  no    2
2 yes    3
> 
```

Exp no. : 4 **USER - DEFINED FUNCTION**
Date : 20-01-23

AIM :-

To create and use user defined function.

USER-DEFINED FUNCTION

To declare a user-defined function in R, we use the keyword function. The syntax is as follows:

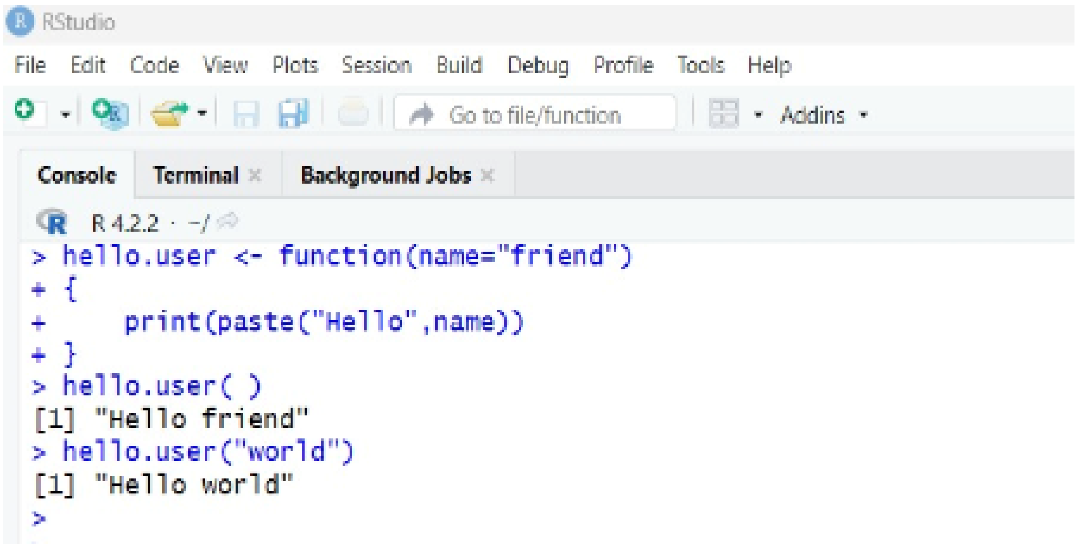
```
function_name <- function(parameters)
{
        function body
}
```

1. Write a function 'hello.user' in R with an argument for name that displays "hello username". If no argument passed, display "hello friend".

Sol -

```
hello.user <- function(name=”Friend”)
{
        print(paste(“Hello”,name))
 }
 hello.user ( )
 hello.user (“World”)
```

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console Terminal Background Jobs
R 4.2.2 · ~/

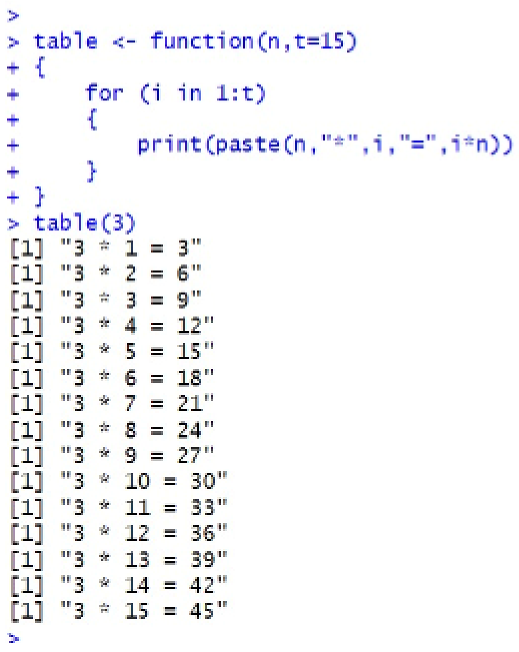
```
> hello.user <- function(name="friend")
+ {
+     print(paste("Hello",name))
+ }
> hello.user( )
[1] "Hello friend"
> hello.user("world")
[1] "Hello world"
>
```

2. Write a function 'tables' in R with argument for table number and times to display the respective multiplication table. Use default value 15 for times.

Sol -

```
table <- function(n,t=15)
   {
     for ( i in 1 : t )
       {
         print(paste(n , ”*” , i , ”=”, i * n ))
       }
   }
table (3)
```

OUTPUT:



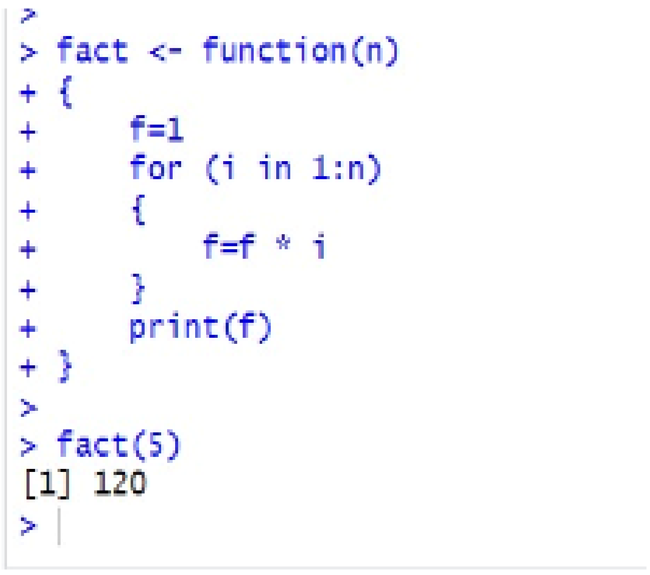
```
>
> table <- function(n,t=15)
+ {
+     for (i in 1:t)
+     {
+         print(paste(n,"*",i,"=",i*n))
+     }
+ }
> table(3)
[1] "3 * 1 = 3"
[1] "3 * 2 = 6"
[1] "3 * 3 = 9"
[1] "3 * 4 = 12"
[1] "3 * 5 = 15"
[1] "3 * 6 = 18"
[1] "3 * 7 = 21"
[1] "3 * 8 = 24"
[1] "3 * 9 = 27"
[1] "3 * 10 = 30"
[1] "3 * 11 = 33"
[1] "3 * 12 = 36"
[1] "3 * 13 = 39"
[1] "3 * 14 = 42"
[1] "3 * 15 = 45"
>
```

3. Write a function ‘fact’ in R to display the factorial value of the passed argument .

Sol -

```
fact <- function (n)
   {
     f = 1
     for ( i in 1 : n )
       {
         f = f * i
       }
   print ( f )
   }
fact (5)
```

OUTPUT:



```
>
> fact <- function(n)
+ {
+     f=1
+     for (i in 1:n)
+     {
+         f=f * i
+     }
+     print(f)
+ }
>
> fact(5)
[1] 120
>
```

Exp no. : 5 DESCRIPTIVE STATISTICS

Date: 30-01-23

AIM:-

To explore the commands that will give an overview of data to be used.

DESCRIPTIVE STATISTICS:

Descriptive statistics is the branch of statistics that focuses on describing and gaining more insight into the data in its present state. It makes the data easier to understand and also gives us knowledge about the data which is necessary to perform further analysis.

Function	Description
mean()	It returns arithmetic average of a numeric input vector.
median()	It returns median of a numeric input vector.
var()	It returns variance of a numeric input vector.
sd()	It returns standard deviation of a numeric input vector.
range() .	It returns the maximum and minimum value of a numeric input vector.
diff()	It returns the difference between pairs of consecutive elements of a numeric vector.
IQR()	It returns the interquartile range of a numeric input vector.
quantile()	It returns the sample quantile of a numeric input vector.
summary()	It returns summary statistics such as mean, median, minimum, maximum, 1st quantile, 3rd quantile, etc. for each component in the object
str()	It displays the internal structure of an R object
table()	It performs a tabulation of categorical variable and gives its frequency as output.
prop.table()	It calculates the proportions of a table, with the result presented as a table with proportions

1. **Display the structure and summary statistics of the iris dataset**
   Sol :

```
str(iris)
summary(iris)
```

2. **Find the mean of Sepal.Length of iris dataset**
   Sol :

```
mean(iris$Sepal.Length)
```

3. **Display variance of Sepal.Length of iris dataset**
   Sol :

```
var(iris$Setal.Length)
```

4. **Display median of Petal.Length of iris dataset**
   Sol :

```
median(iris$Petal.Length)
```

5. **Find Standard Deviation of Setal.Length of iris dataset**
   Sol :

```
sd(iris$Sepal.Length)
```

6. **Display top 10 records from the dataset iris**
   Sol :

```
head(iris, n = 10)
```

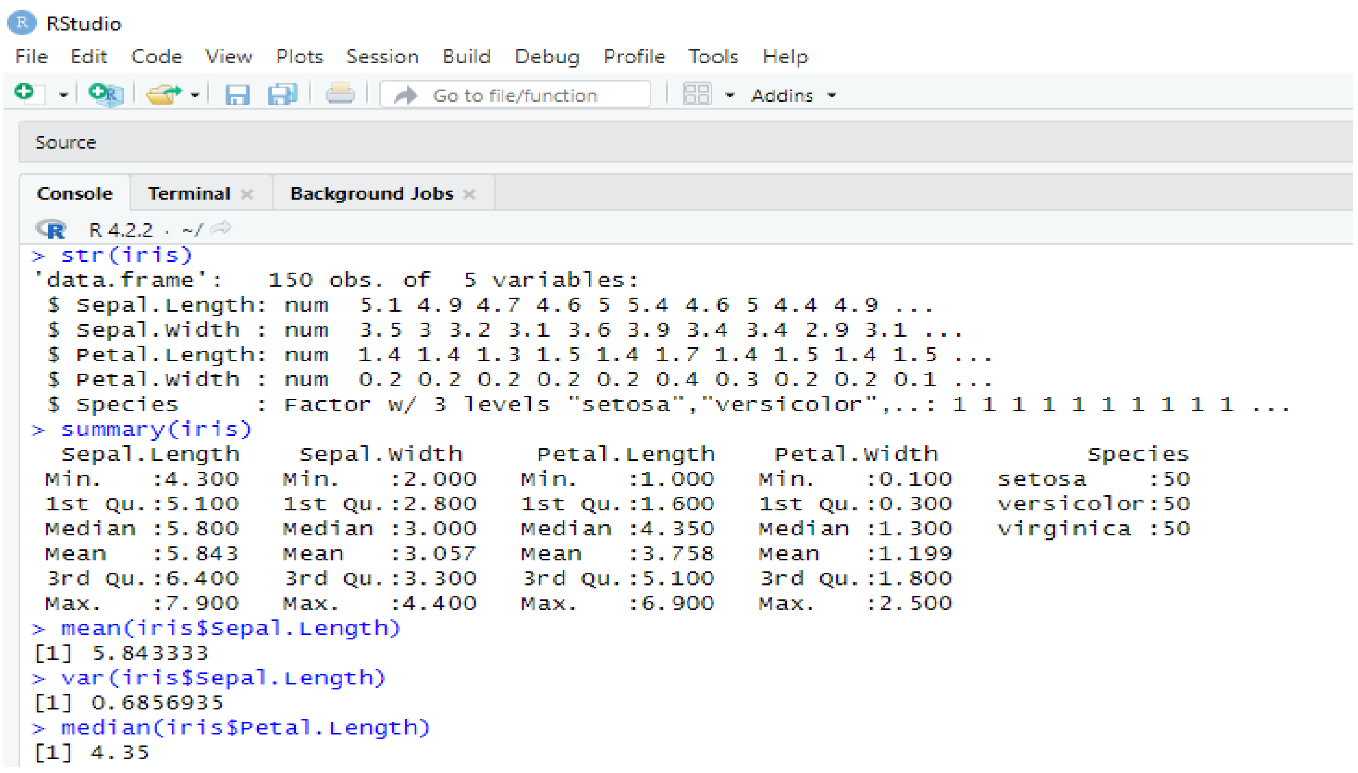
7. **Display bottom 10 records from the dataset iris**
   Sol :

```
tail(iris, n=10)
```

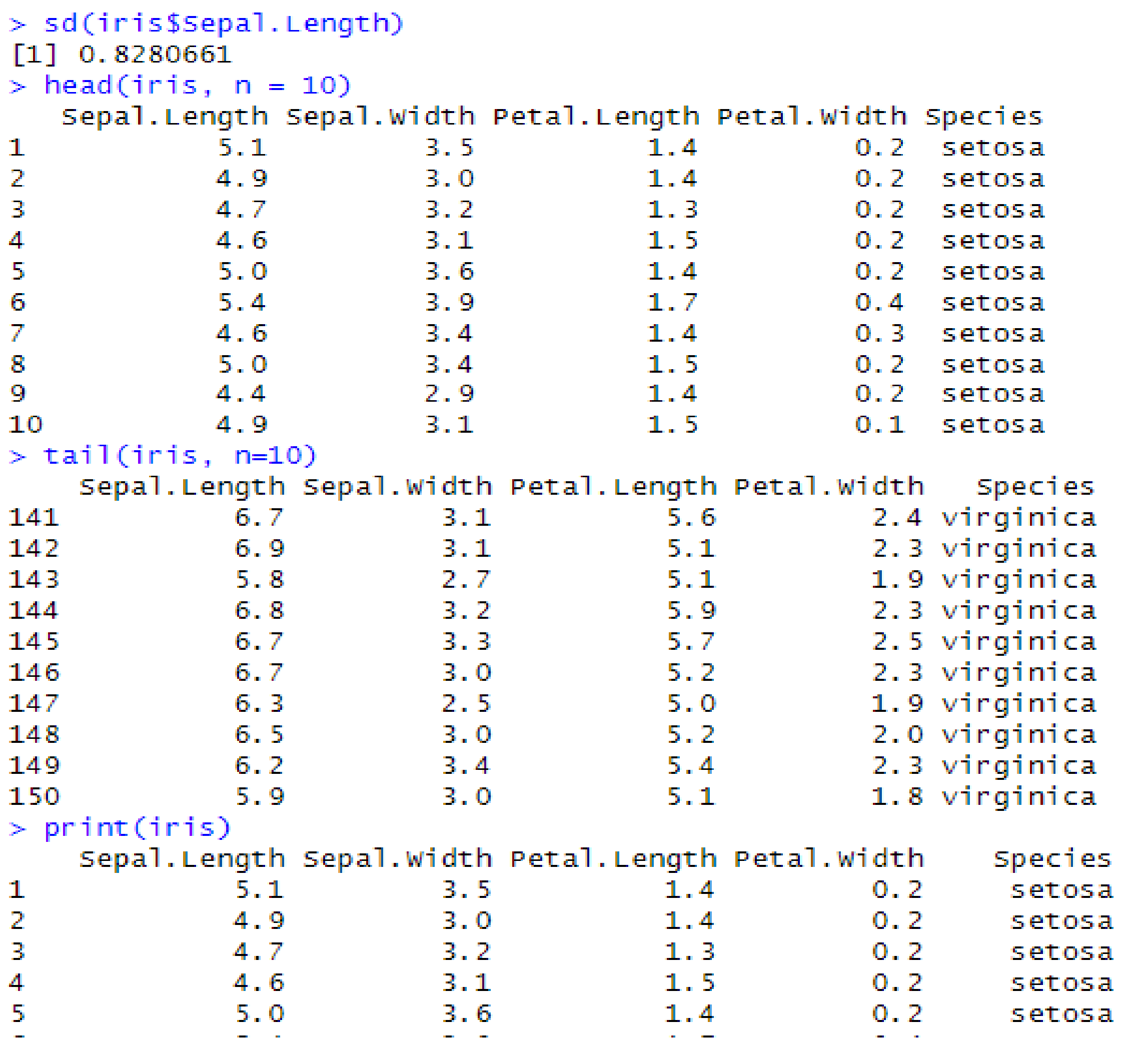
8. **Print contents of the dataframe iris**
   Sol :

```
print(iris)
```

OUTPUT:



```
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199                  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800                  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500                  
> mean(iris$Sepal.Length)
[1] 5.843333
> var(iris$Sepal.Length)
[1] 0.6856935
> median(iris$Petal.Length)
[1] 4.35
```



```
> sd(iris$Sepal.Length)
[1] 0.8280661
> head(iris, n = 10)
   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
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
> tail(iris, n=10)
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
141          6.7         3.1          5.6         2.4 virginica
142          6.9         3.1          5.1         2.3 virginica
143          5.8         2.7          5.1         1.9 virginica
144          6.8         3.2          5.9         2.3 virginica
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
> print(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
```

Exp no. : 6 VECTOR MANIPULATION

Date: 30-01-23

AIM:-

To create and manipulate vector in R.

VECTOR:-

A vector is asequence of data elements of the same basic type. Data types can be numeric, integer,character, complex or logical. It is created using the c() function. Since, a vector must have elements of thesame type, this function will try and coerce elements to the same type, if they are different.Coercion is from lower to higher types from logical to integer to double to character.

1. Create the following vectors:

Note: use: operator, seq() and rep()

a. (1, 2, 3, . . . , 19, 20)

Sol -

```
seq(1,20)
```

a. (20, 19, . . . , 2, 1)

Sol -

```
seq(20,1)
```

b. (1, 2, 3, . . . , 19, 20, 19, 18, . . . , 2, 1)

Sol -

```
c(1:20,19:1)
```

c. (5,10,15, ... , 100)

Sol -

```
seq(5,100,by=5)
```

d. (4, 6, 3, 4, 6, 3, . . . , 4, 6, 3) where there are 10 occurrences of 4,6,3.

Sol -

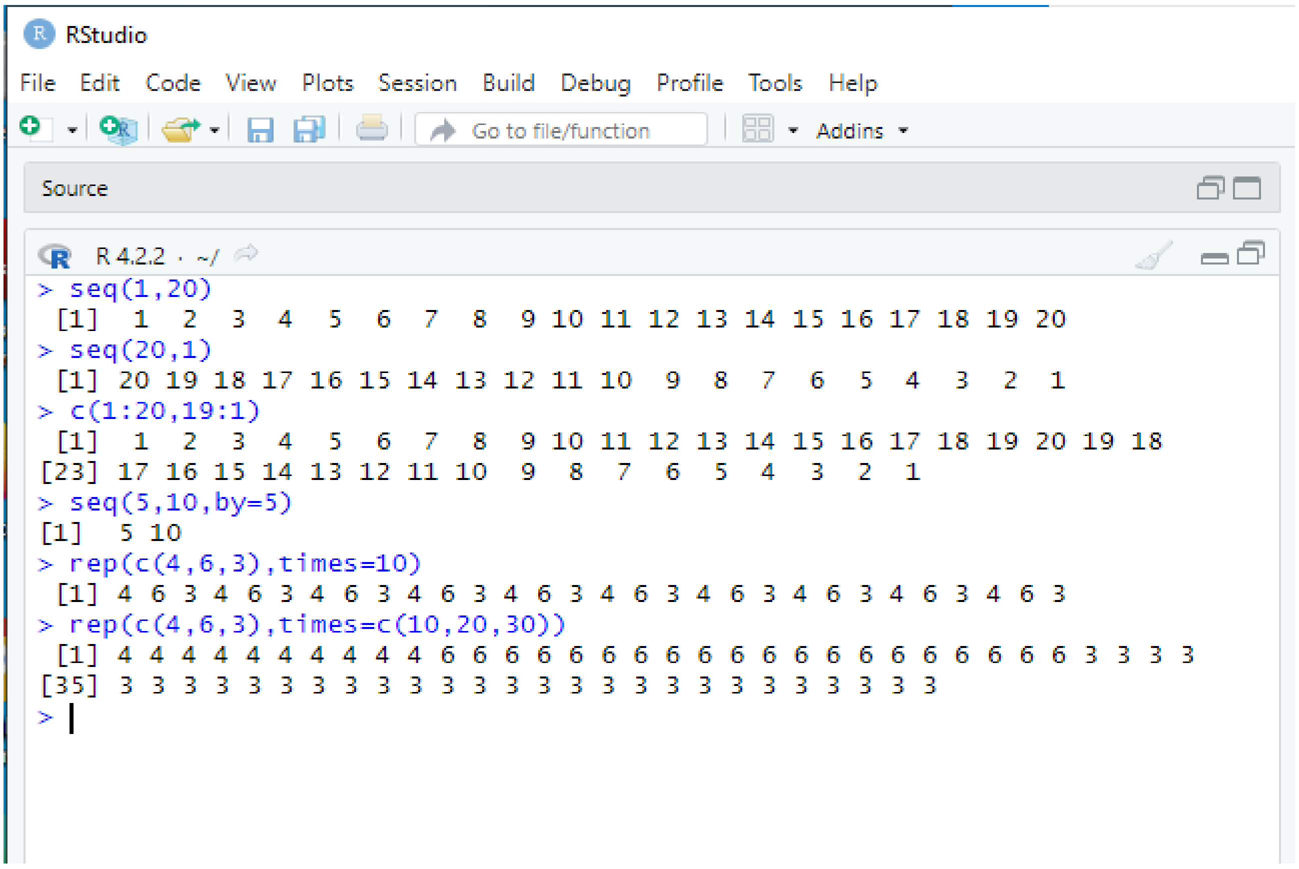
```
rep(c(4,6,3),times=10)
```

e. (4, 4, . . . , 4, 6, 6, . . . , 6, 3, 3, . . . , 3) where there are 10 occurrences of 4, 20 occurrences of 6 and 30 occurrences of 3.

Sol -

```
rep(c(4,6,3),times=c(10,20,30))
```

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
R 4.2.2 · ~/

```
> seq(1,20)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> seq(20,1)
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> c(1:20,19:1)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 18
[23] 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> seq(5,10,by=5)
[1]  5 10
> rep(c(4,6,3),times=10)
 [1] 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3
> rep(c(4,6,3),times=c(10,20,30))
 [1] 4 4 4 4 4 4 4 4 4 4 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 3 3 3 3
[35] 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
> 
```

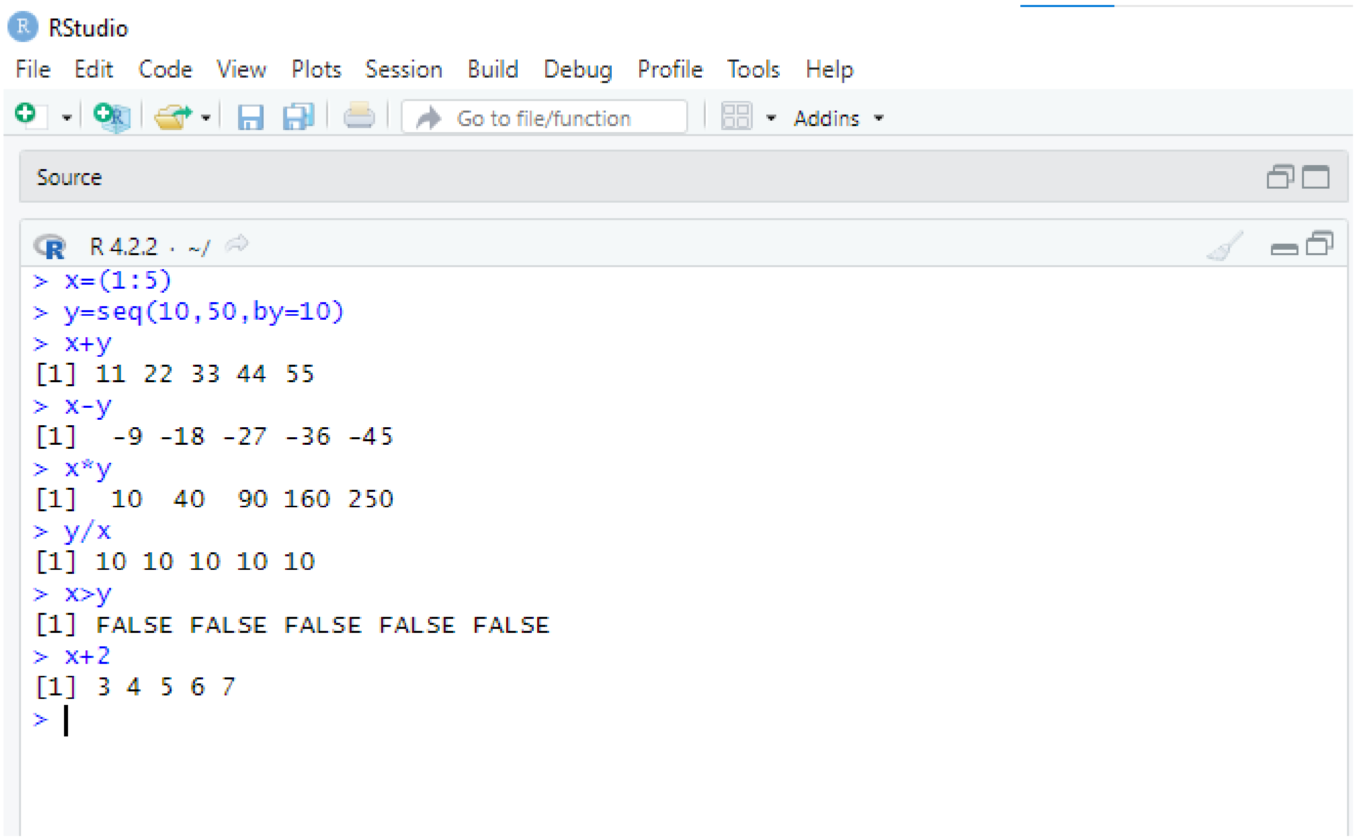
2. Create a vector with elements 1, 2, 3, 4, 5 and call it x.
Create another vector with elements 10 20 30 40 50 and call it y
What is the value of

```
x + y
x – y
x * y
y /x
x > y
x+2
```

Sol -

```
 x=c(1:5)
 y=seq(10,50,by=10)
x + y
x – y
x * y
y /x
x > y
x+2
```

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
R 4.2.2 · ~/

```
> x=(1:5)
> y=seq(10,50,by=10)
> x+y
[1] 11 22 33 44 55
> x-y
[1]  -9 -18 -27 -36 -45
> x*y
[1]  10  40  90 160 250
> y/x
[1] 10 10 10 10 10
> x>y
[1] FALSE FALSE FALSE FALSE FALSE
> x+2
[1] 3 4 5 6 7
> 
```

3. Create a Vector 'v' with the values 10, 20, 30, 10, 40, 50, 30, 40, 80, 90, 100 and do the following exercises

v = c(10, 20, 30, 10, 40, 50, 30, 40, 80, 90, 100)

a. List elements of the vector that are greater than 10 and less than 80 .

Sol -

```
v[v>10 & v<80]
```

b. List elements of the vector that are multiples of 4

Sol -

```
v[v %% 4 ==0]
```

c. How many times the element 40 occurred in the vector

Sol -

```
sum (v == 40)
```

d. List the last value in the given vector.

Sol -

```
v[ length(v) ]
```

e. Find second highest value in the given vector.

Sol -

```
sort(v , decreasing = TRUE) [2]
```

f. Find nth highest value in the given vector

Sol -

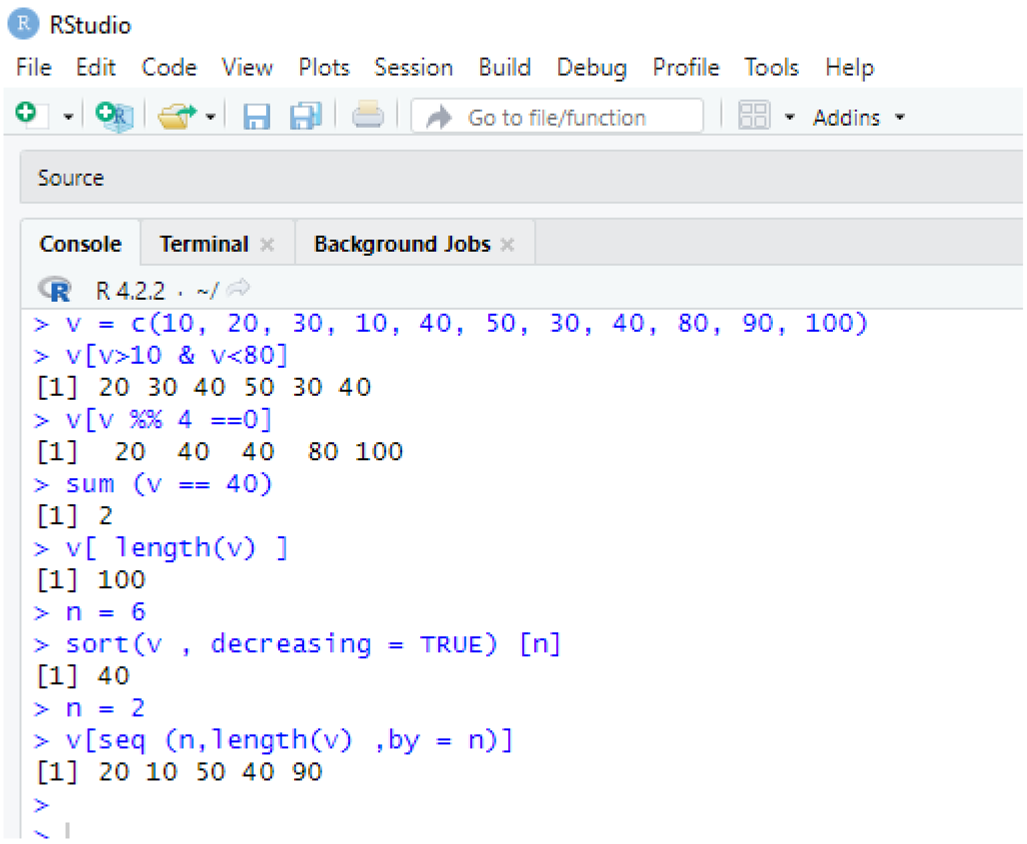
```
n = 6
sort(v , decreasing = TRUE) [n]
```

g. Extract every nth element of a given vector.

Sol -

```
n = 2
 v[seq (n,length(v) ,by = n)]
```

OUTPUT:



```
> v = c(10, 20, 30, 10, 40, 50, 30, 40, 80, 90, 100)
> v[v>10 & v<80]
[1] 20 30 40 50 30 40
> v[v %% 4 ==0]
[1]  20  40  40  80 100
> sum (v == 40)
[1] 2
> v[ length(v) ]
[1] 100
> n = 6
> sort(v , decreasing = TRUE) [n]
[1] 40
> n = 2
> v[seq (n,length(v) ,by = n)]
[1] 20 10 50 40 90
>
```

DATA FRAMES IN R

AIM:

To create data frame and perform various operations on data frame in R

PROCEDURE:

Data frame is a two dimensional data structure in R. It is a special case of a list which has each component of equal length. Each component forms the column and contents of the component form the rows.

1.Create data frame to display voting details.

```
dataframe1 <- data.frame ( Name = c("Juan", "Alcaraz", "Simantha"),
                              Age = c(22, 15, 9),
                              Vote = c(TRUE, FALSE, TRUE))
                  print(dataframe1)


print(dataframe1)
```

2.Access the name column from data frame.

```
                 # pass index number inside [ ]
                   print(dataframe1[1])

                   # pass column name inside [[  ]]
                        print(dataframe1[["Name"]])
                    # use $ operator and column name
                        print(dataframe1$Name)
```

3.Use rbind() in data frame.

```
dataframe2 <- data.frame (Name = c("Yiruma", "Bach"),
                             Age = c(46, 89))
print(dataframe2)
```

4.Use cbind() in data frame.

```
                # create a data frame
             dataframe1 <- data.frame ( Name = c("Juan", "Alcaraz"),
                                          Age = c(22, 15))
```

```
# create another data frame
dataframe2 <- data.frame (
Hobby = c("Tennis", "Piano"))

# combine two data frames horizontally
updated <- cbind(dataframe1, dataframe2)
print(updated)
```

4. Find the length of Data frame.

```
length(dataframe1)
```

Exp no. : 7 **MATRIX MANIPULATION**

Date : 06-02-23

AIM :-

To create and manipulate matrix in R.

MATRIX

Matrix is a two dimensional data structure in R programming. It can be created using the matrix() function.Dimension of the matrix can be defined by passing appropriate value for arguments nrow andncol

1. Create the following matrix

1 4

2 5

3 6

Sol -
```
matrix1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
print(matrix1)
```

2. Create a vector v of size 12 with random value between 1 to 100

Sol -
```
v = sample(1:100,12)
print(v)
```

3. Convert the vector v to a 4*3 matrix A

Sol -
```
vec <- c(16, 5, 4, 3, 2, 1)
matrix1 <- matrix(vec,nrow = 2, ncol = 3)
print(matrix1)
```

4.Change the column names of matrix A to x, y, z and row names to a, b, c, d.

Sol -
```
rownames(matrix1) = c("a","b")
colnames(matrix1) = c("x","y","z")
print(matrix1)
```

Compute A+3, A-3, A*3 and A/3.

Sol -
```
matrix1+3
matrix-3
matrix*3
matrix/3
```

4. Obtain the transpose matrix of A

Sol -
```
t (a)
```

5. Display 2nd row of matrix

Sol -
```
A [2,  ]
```

6. Display the entire matrix leaving 2nd column

Sol -

```
A [  , -2]
```

7. Display only first three rows of matrix

Sol -

```
A [  c(1:3),   ]
```

8. Change the value of element at 2nd row and 3rd column to 300

Sol -

```
A [2:3] = 300
```

9. Replace all elements of matrix A that are greater than 50 with 200.

Sol -

```
A [ A>50]=200
```

10. Display column and row names of matrix A

Sol -

```
rownames (A )
colnames(A)
```

11. Display dimension and the no of elements in matrix A

Sol -

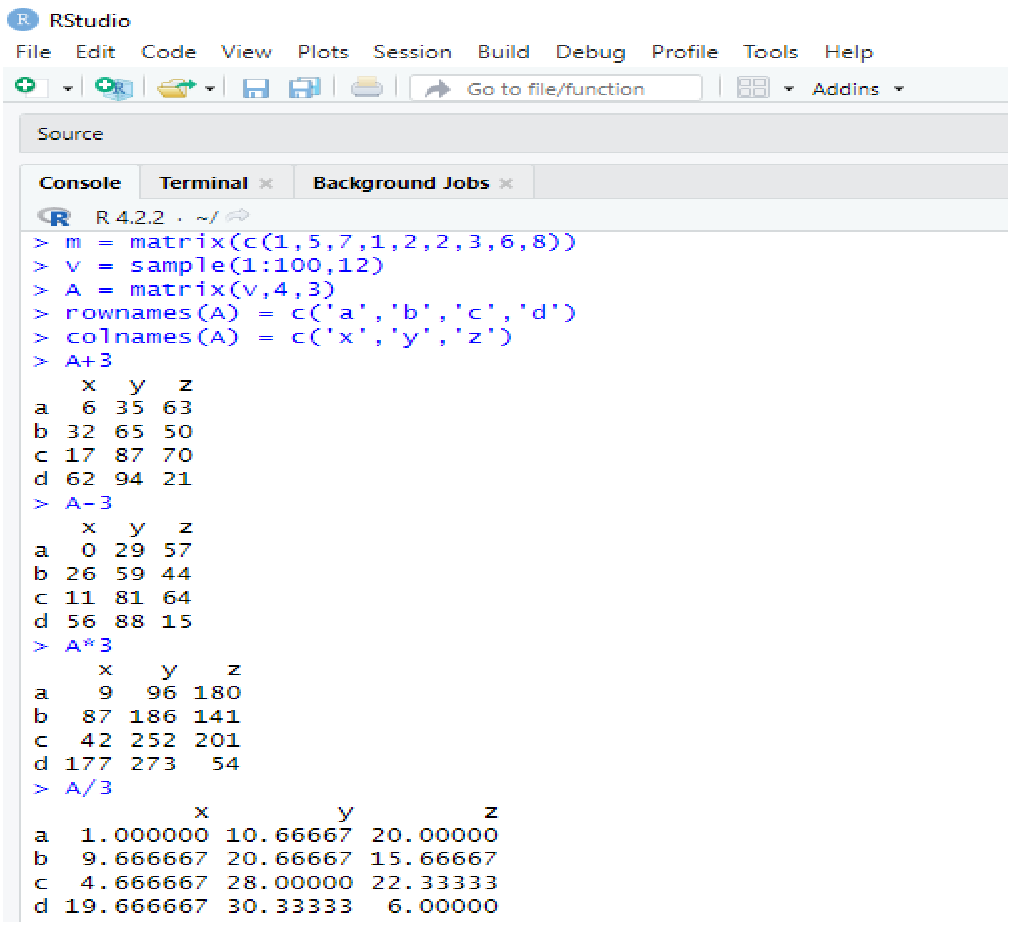
```
 dim(A)
length(A)
```

12. Multiply the matrix A with its transpose

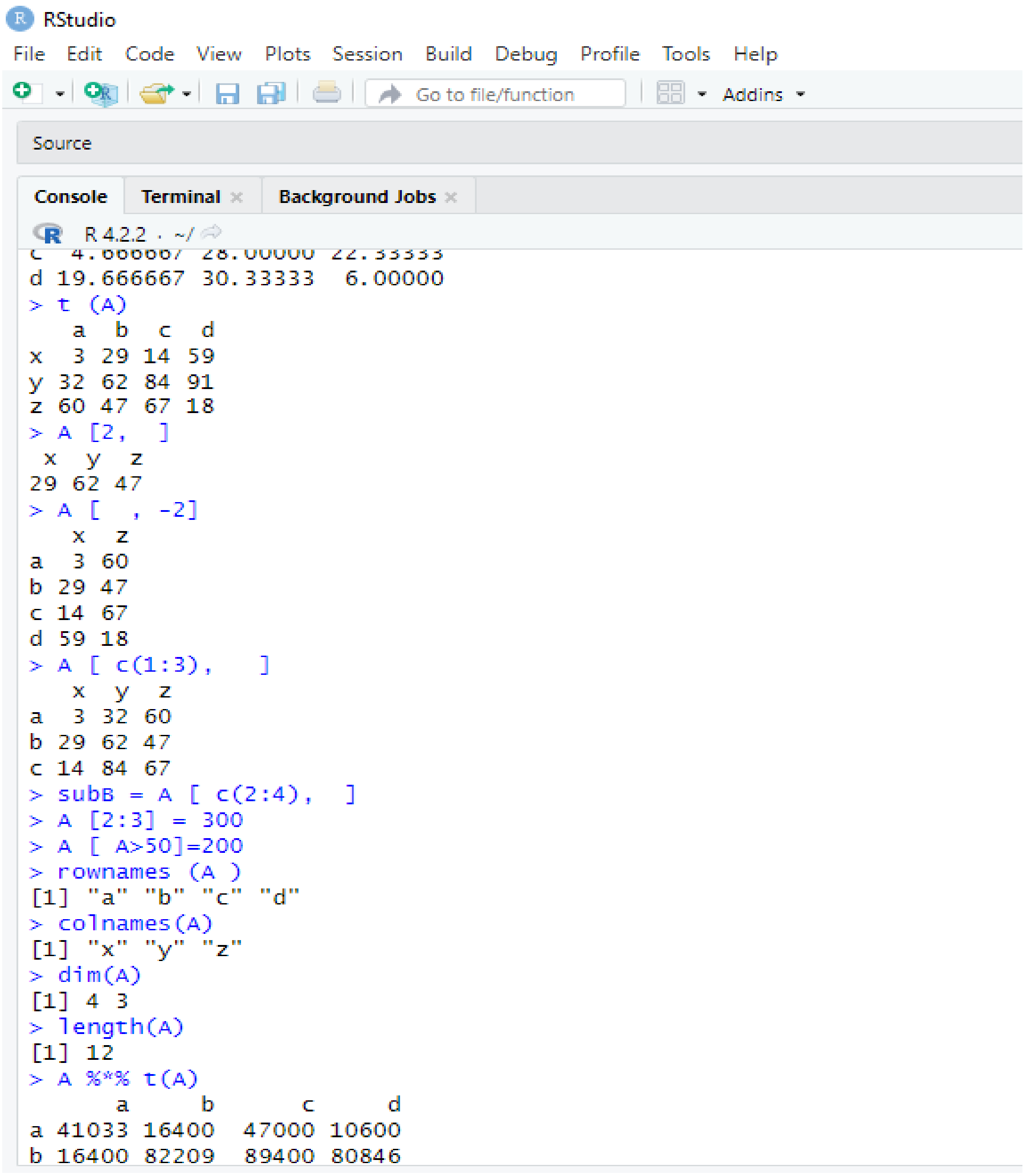
Sol -

```
A %*% t(A)
```

OUTPUT:



```
> m = matrix(c(1,5,7,1,2,2,3,6,8))
> v = sample(1:100,12)
> A = matrix(v,4,3)
> rownames(A) = c('a','b','c','d')
> colnames(A) = c('x','y','z')
> A+3
   x  y  z
a  6 35 63
b 32 65 50
c 17 87 70
d 62 94 21
> A-3
   x  y  z
a  0 29 57
b 26 59 44
c 11 81 64
d 56 88 15
> A*3
    x   y   z
a   9  96 180
b  87 186 141
c  42 252 201
d 177 273  54
> A/3
          x        y        z
a  1.000000 10.66667 20.00000
b  9.666667 20.66667 15.66667
c  4.666667 28.00000 22.33333
d 19.666667 30.33333  6.00000
```



```
c  4.666667 28.00000 22.33333
d 19.666667 30.33333  6.00000
> t (A)
   a  b  c  d
x  3 29 14 59
y 32 62 84 91
z 60 47 67 18
> A [2,  ]
 x  y  z
29 62 47
> A [  , -2]
   x  z
a  3 60
b 29 47
c 14 67
d 59 18
> A [ c(1:3),    ]
   x  y  z
a  3 32 60
b 29 62 47
c 14 84 67
> subB = A [ c(2:4),  ]
> A [2:3] = 300
> A [ A>50]=200
> rownames (A )
[1] "a" "b" "c" "d"
> colnames(A)
[1] "x" "y" "z"
> dim(A)
[1] 4 3
> length(A)
[1] 12
> A %*% t(A)
      a     b      c     d
a 41033 16400  47000 10600
b 16400 82209  89400 80846
```

Exp no. : 8 **LIST MANIPULATION**
Date: 06-02-23

AIM:-

To create and manipulate list in R.

LIST:

List is a data structure having components of mixed data types. A vector having all elements of the same type is called atomic vector but a vector havingelements of different type is called list. It can be created using the list() function.

1. Create a list called my_list from the 3 vectors provided below and do the following

a <- "My first list&quot"
b <- c("R", "is", "Fun!")
c <- matrix(1:9,3,3)

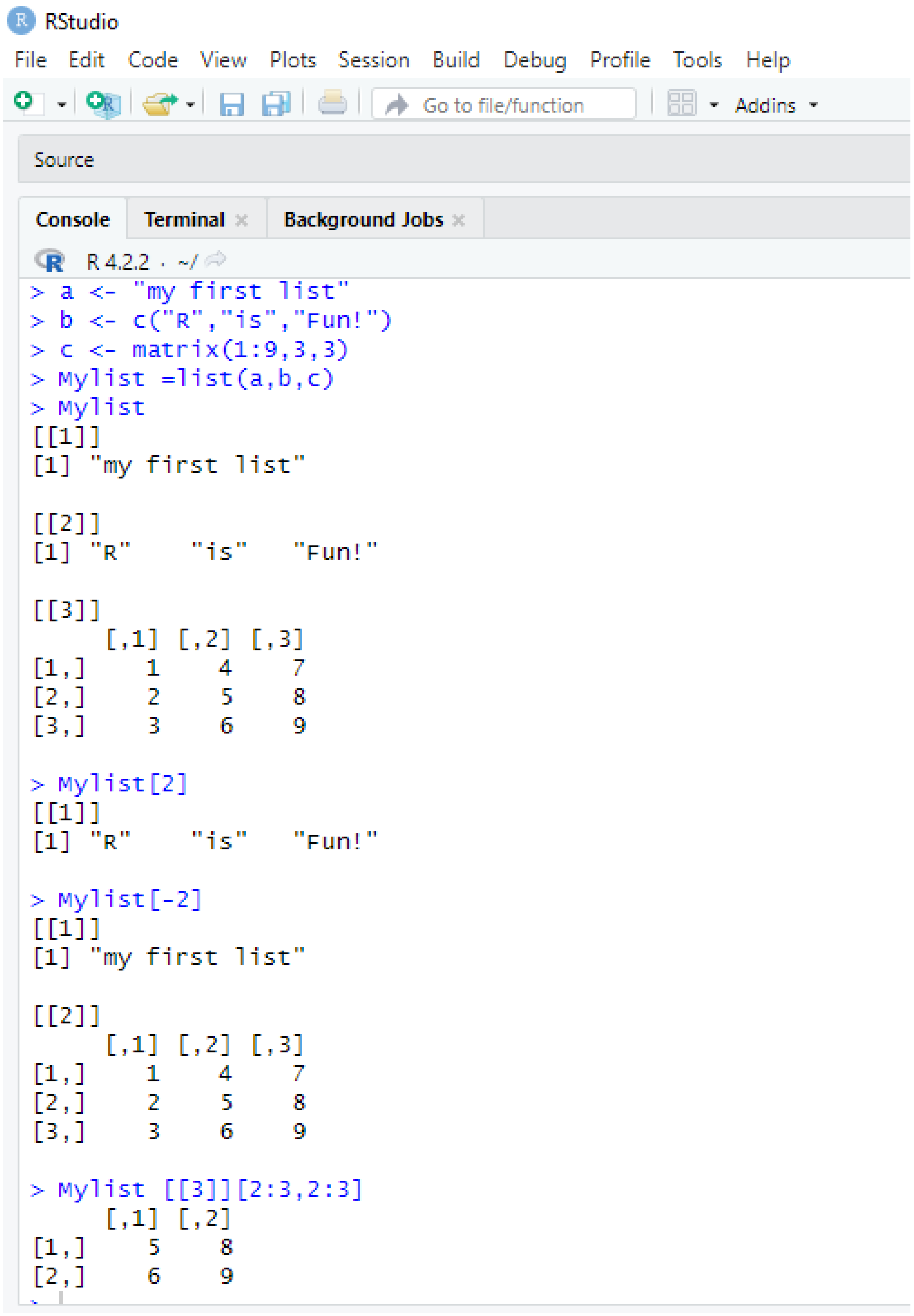
Sol :

```
Mylist = list(a,b,c)
```

a) **Access second component of my_list.**
Sol :
```
Mylist[2]
```
b) **Display all components of my_list leaving second component**
Sol :
```
Mylist[-2]
```
c) **Remove the first and third items from my_list.**

Sol :
```
mylist [-2] = null
```

OUTPUT:



```
> a <- "my first list"
> b <- c("R","is","Fun!")
> c <- matrix(1:9,3,3)
> Mylist =list(a,b,c)
> Mylist
[[1]]
[1] "my first list"

[[2]]
[1] "R"    "is"   "Fun!"

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

> Mylist[2]
[[1]]
[1] "R"    "is"   "Fun!"

> Mylist[-2]
[[1]]
[1] "my first list"

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

> Mylist [[3]][2:3,2:3]
     [,1] [,2]
[1,]    5    8
[2,]    6    9
```

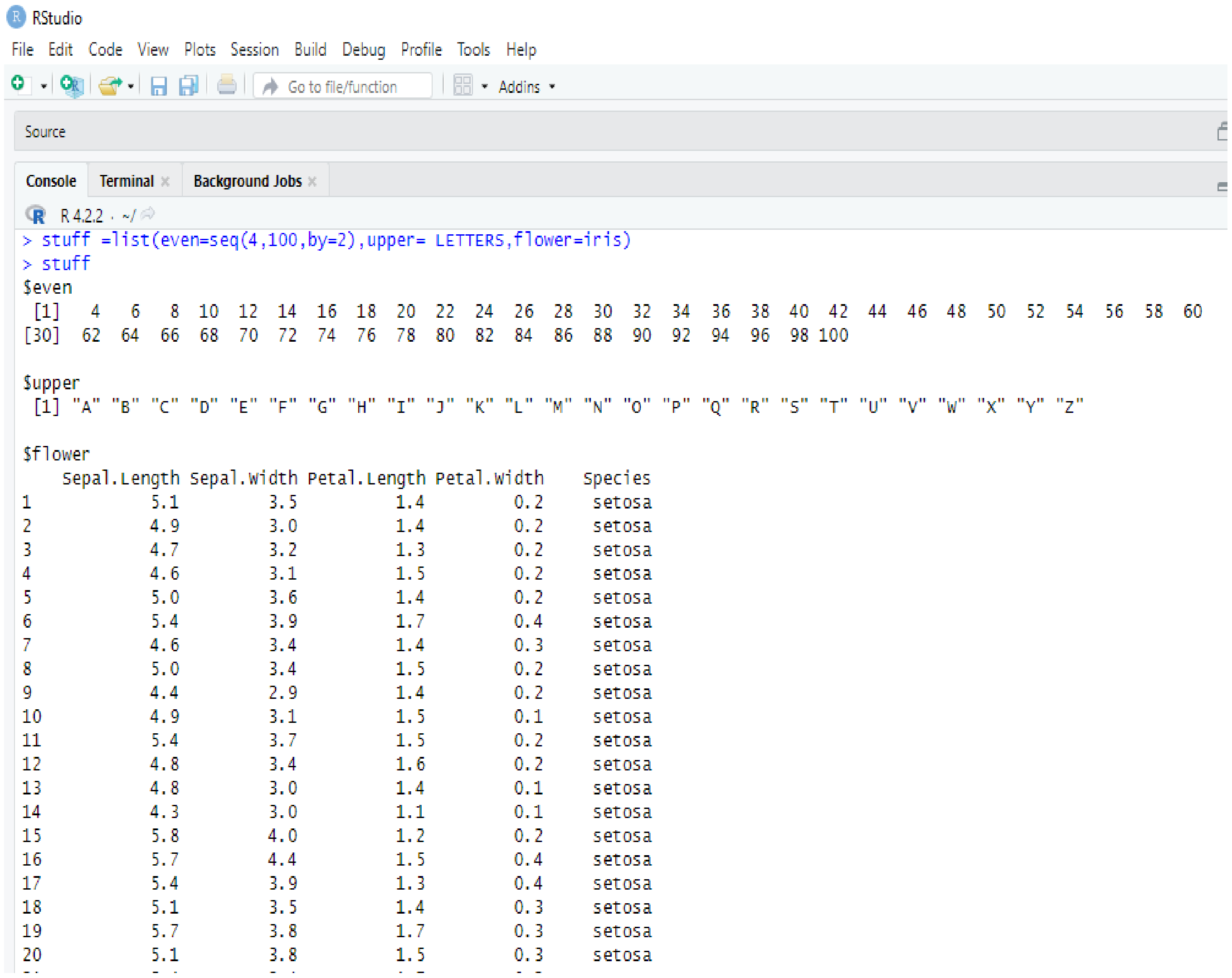
2. Make a list called Stuff and display its structure. The list should contain three elements

a) The sequence of even numbers from 4 to 100. Its name should be even.
b) The uppercase letters of the alphabet. Its name should be upper.
c) The data frame iris. Its name should be flower.

Sol :

```
stuff= list(even= seq(4,100,by=2),upper = LETTERS,flower =iris)
```

OUTPUT :



```
> stuff =list(even=seq(4,100,by=2),upper= LETTERS,flower=iris)
> stuff
$even
 [1]   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  50  52  54  56  58  60
[30]  62  64  66  68  70  72  74  76  78  80  82  84  86  88  90  92  94  96  98 100

$upper
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

$flower
    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
7            4.6         3.4          1.4         0.3     setosa
8            5.0         3.4          1.5         0.2     setosa
9            4.4         2.9          1.4         0.2     setosa
10           4.9         3.1          1.5         0.1     setosa
11           5.4         3.7          1.5         0.2     setosa
12           4.8         3.4          1.6         0.2     setosa
13           4.8         3.0          1.4         0.1     setosa
14           4.3         3.0          1.1         0.1     setosa
15           5.8         4.0          1.2         0.2     setosa
16           5.7         4.4          1.5         0.4     setosa
17           5.4         3.9          1.3         0.4     setosa
18           5.1         3.5          1.4         0.3     setosa
19           5.7         3.8          1.7         0.3     setosa
20           5.1         3.8          1.5         0.3     setosa
```

Exp no. : 9 **IMPORTING FILES IN R**
Date: 13-02-23

AIM:-

To import different types of file such as csv,xml, excel and text files in R.

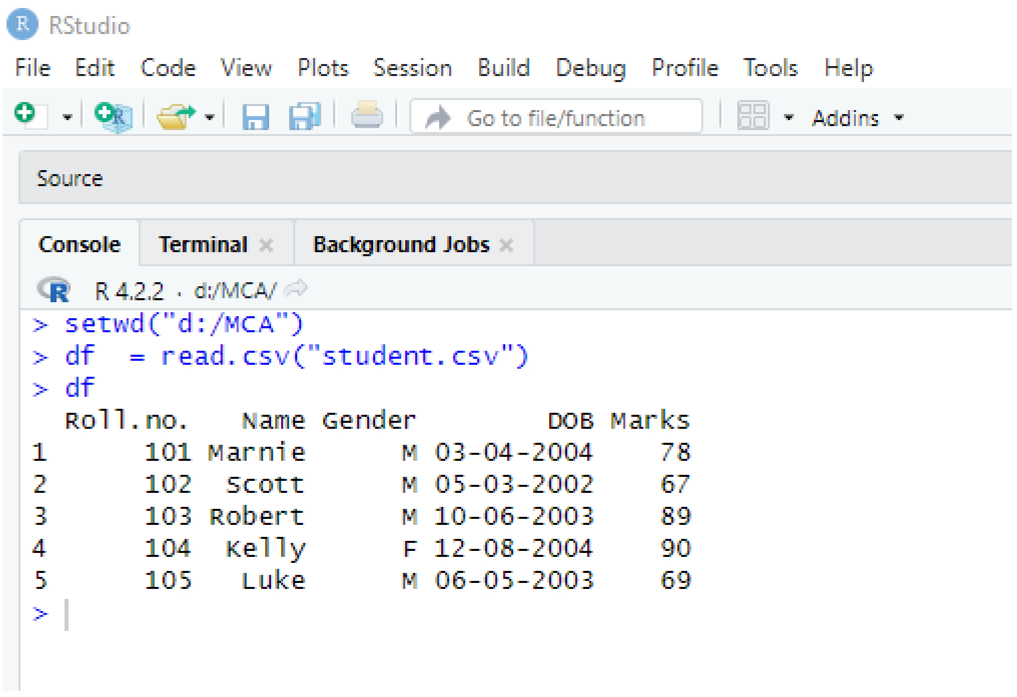
1. Importing a CSV File :

Create a csv file with the following student data and save it as student.csv:Import this file in r studio:

Roll no.	Name	Gender	DOB	Marks
101	Marnie	M	3-4-2004	78
102	Scott	M	5-3-2002	67
103	Robert	M	10-6-2003	89
104	Kelly	F	12-8-2004	90
105	Luke	M	6-5-2003	69

```
setwd(“D:/MCA”)
df  = read.csv(“student.csv”)
df
```

OUTPUT:



```
> setwd("d:/MCA")
> df   = read.csv("student.csv")
> df
  Roll.no.    Name Gender        DOB Marks
1      101 Marnie      M 03-04-2004    78
2      102  Scott      M 05-03-2002    67
3      103 Robert      M 10-06-2003    89
4      104  Kelly      F 12-08-2004    90
5      105   Luke      M 06-05-2003    69
> 
```

2. Importing Excel file :

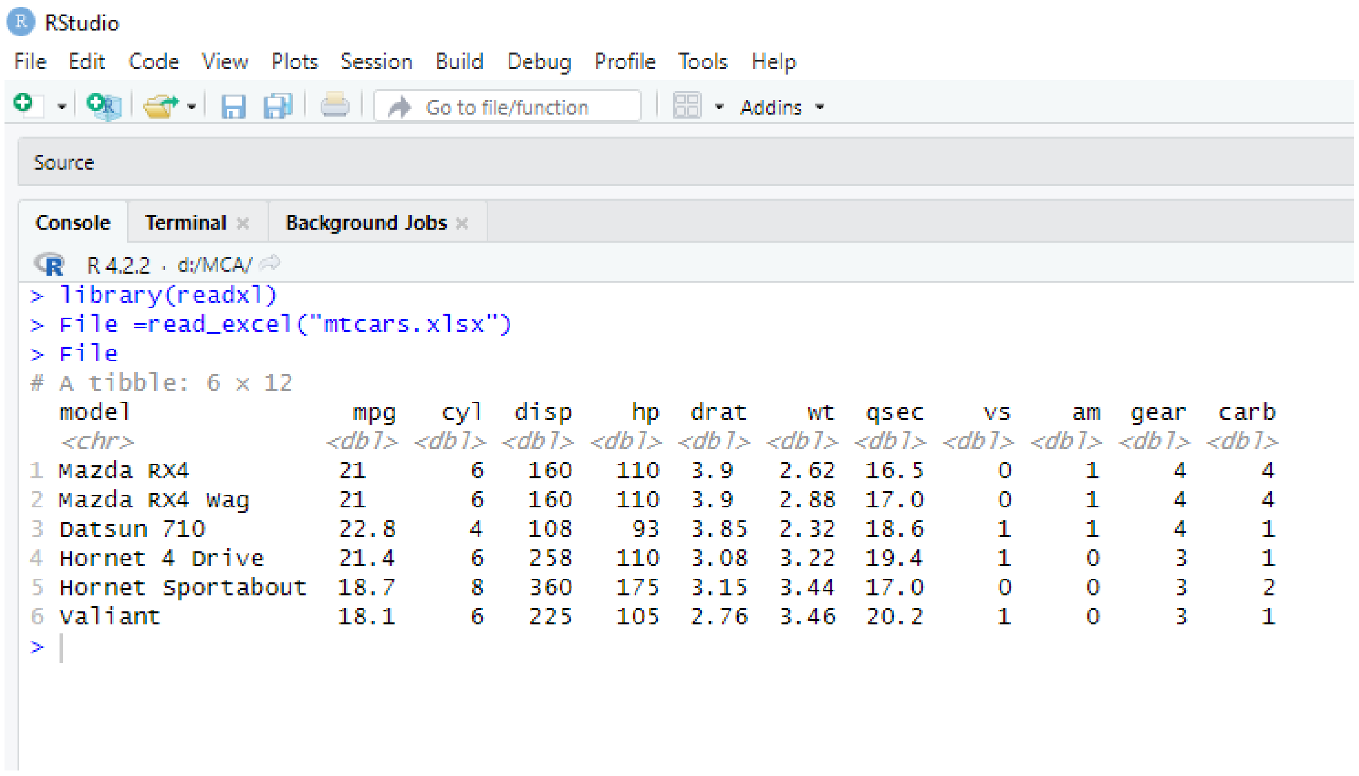
Create a excel file which contains the following data and save it as mtcars.xlsx :

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	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	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1

Import this file in r studio and display it:

```
install.packages("readxl")
library("readxl")
File =read_excel("mtcars.xlsx")
```

OUTPUT:



```
> library(readxl)
> File =read_excel("mtcars.xlsx")
> File
# A tibble: 6 × 12
  model               mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>             <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4          21       6   160   110  3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 Wag      21       6   160   110  3.9   2.88  17.0     0     1     4     4
3 Datsun 710         22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
4 Hornet 4 Drive     21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
5 Hornet Sportabout  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
6 Valiant            18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
> 
```

3. Importing a Text file :-

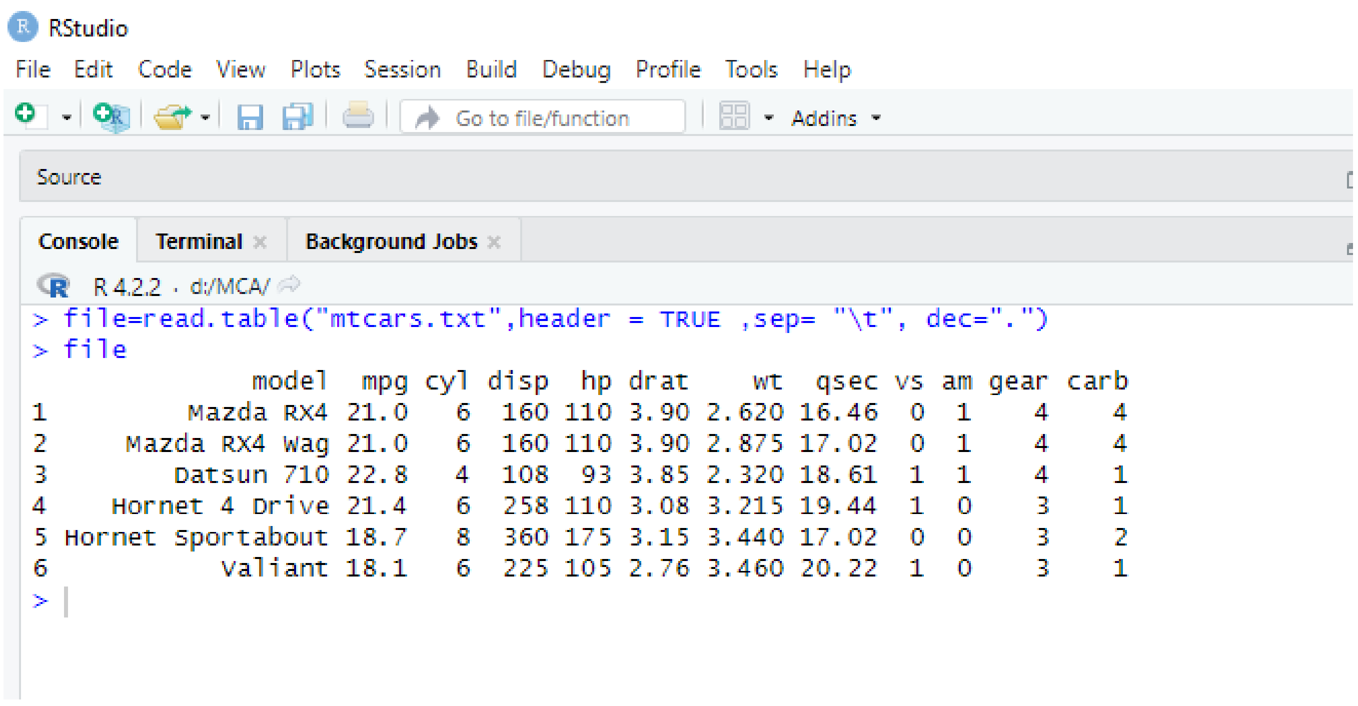
Create a text file using notepad with the following data and save it as ‘mtcars.txt’

```
model           mpg      cyl    disp   hp    drat   wt    qsec   vs     am    gear   carb
Mazda RX4        21       6     160    110   3.9    2.62  16.46  0      1     4      4
Mazda RX4 Wag          21       6      160   110    3.9   2.875  17.02  0     1      4
4
Datsun 710      22.8      4     108    93    3.85   2.32  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  3.44   17.02  0     0      3
2
Valiant               18.1      6      225   105    2.76  3.46   20.22  1     0      3
1
```

Import these file in r studio :

file=read.table(“mtcars.txt”)
file

OUTPUT:



```
> file=read.table("mtcars.txt",header = TRUE ,sep= "\t", dec=".")
> file
              model  mpg cyl disp  hp drat    wt  qsec vs am gear carb
1         Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
2     Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
3        Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
4    Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
5 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
6           Valiant 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> |
```

Exp no. : 10 **IMPLEMENTING NAVIE BAYES USING IRIS DATASET**
Date : 22-02-23

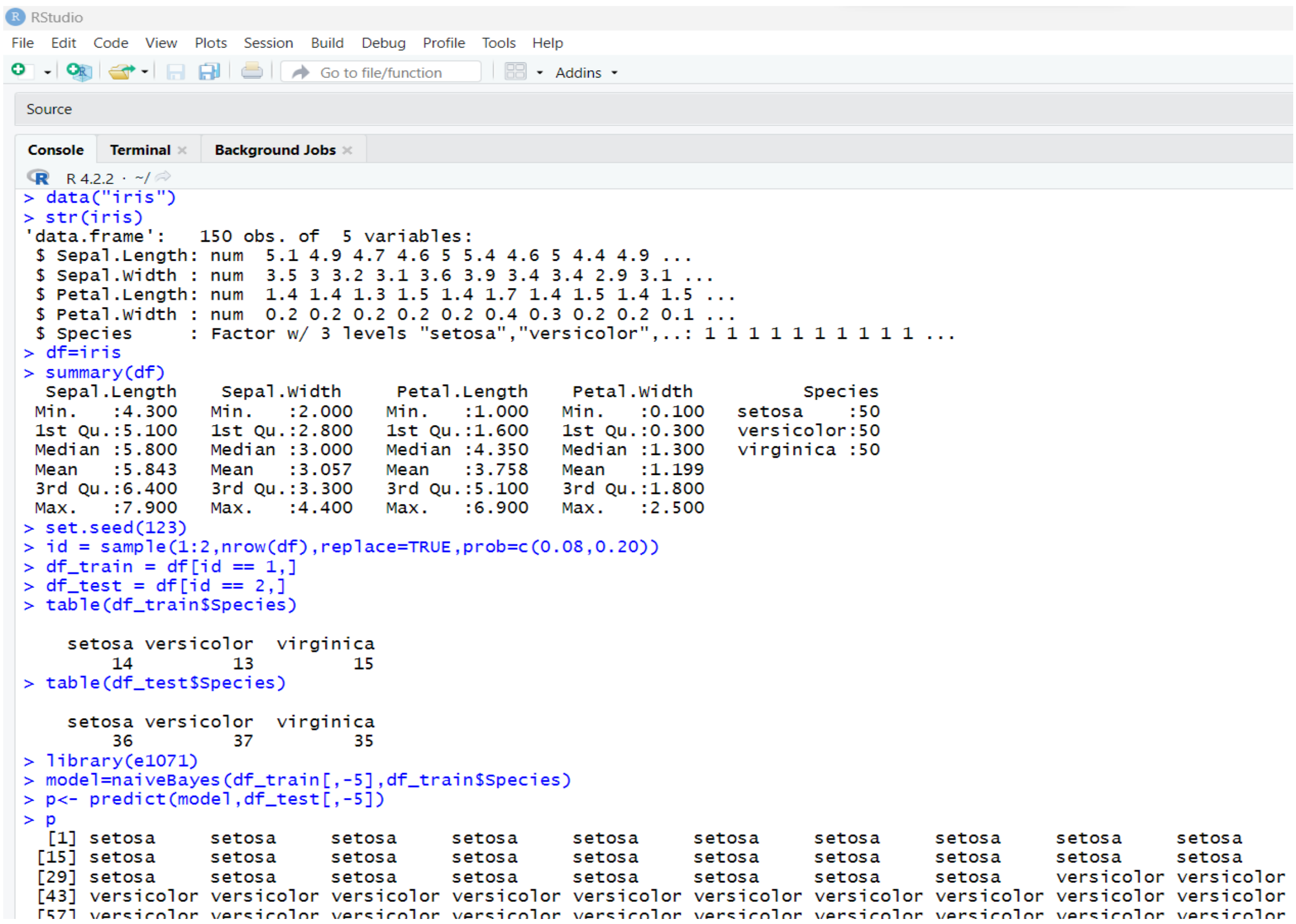
AIM :-
To show the implementation of Navie Bayes algorithm using Iris dataset.

NAVIE BAYES ALGORITHM:
Naive Bayes is a Supervised Machine Learning algorithm based on the Bayes Theorem that is used to solve classification problems by following a probabilistic approach. It is based on the idea that the predictor variables in a Machine Learning model are independent of each other. Meaning that the outcome of a model depends on a set of independent variables(Predictor) that have nothing to do with each other.
PROCEDURE:
1. Load the dataset .
2. Create train/test set.
    i. id <- sample(2,nrow(df),replace=TRUE,prob=c(0.80,0.20))
    ii. df_train <- df [id==1,]
    iii. df_test <- df [id==2,]
3. Build the model.
    i. Install Package e1071
    ii. Load the package
    iii. Model <- naviesBayes(df_train[,5] ,df_train$species)
4. Make prediction.
    p= predict(model,df_test[, -5])
5. Measure performance by confusion matrix.

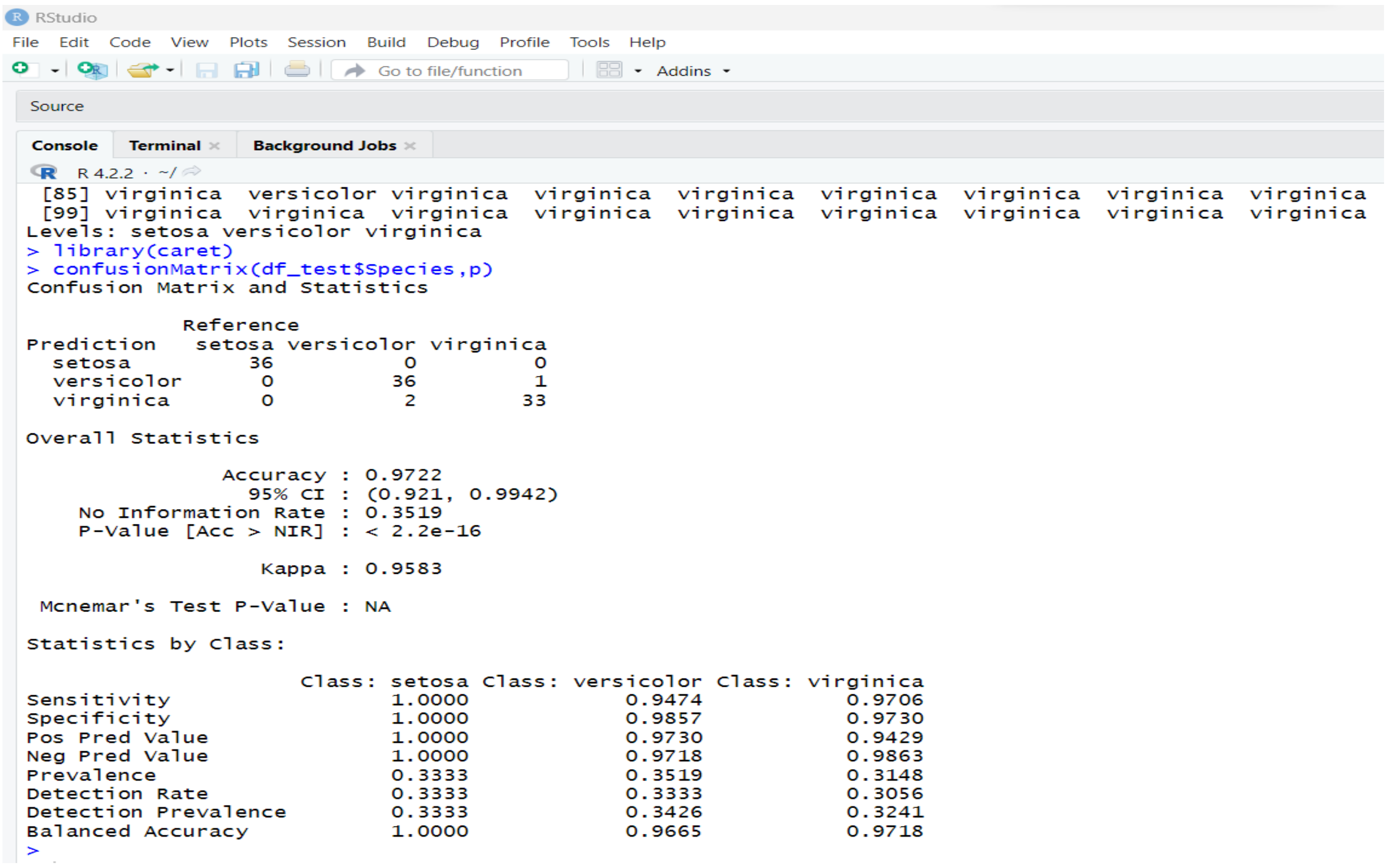
SOURCE CODE :
```
data("iris")
str(iris)
df=iris
summary(df)
set.seed(123)
id = sample(1:2,nrow(df),replace=TRUE,prob=c(0.08,0.20))
df_train = df[id == 1,]
df_test = df[id == 2,]
table(df_train$Species)
table(df_test$Species)
install.packages("e1071")
library(e1071)
model=naiveBayes(df_train[,-5],df_train$Species)
p<- predict(model,df_test[,-5])
p
```

```
install.packages("caret")
library(caret)
confusionMatrix(df_test$Species,p)
```

OUTPUT :



```
> data("iris")
> str(iris)
'data.frame':	150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> df=iris
> summary(df)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199                  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800                  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500                  
> set.seed(123)
> id = sample(1:2,nrow(df),replace=TRUE,prob=c(0.08,0.20))
> df_train = df[id == 1,]
> df_test = df[id == 2,]
> table(df_train$Species)

    setosa versicolor  virginica 
        14         13         15 
> table(df_test$Species)

    setosa versicolor  virginica 
        36         37         35 
> library(e1071)
> model=naiveBayes(df_train[,-5],df_train$Species)
> p<- predict(model,df_test[,-5])
> p
  [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa    
 [15] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa    
 [29] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa     versicolor versicolor
 [43] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versicolor versicolor versicolor
 [57] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versicolor versicolor versicolor
```



```
 [85] virginica  versicolor virginica  virginica  virginica  virginica  virginica  virginica  virginica
 [99] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
> library(caret)
> confusionMatrix(df_test$Species,p)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         36          0         0
  versicolor      0         36         1
  virginica       0          2        33

Overall Statistics

               Accuracy : 0.9722
                 95% CI : (0.921, 0.9942)
    No Information Rate : 0.3519
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9583

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            0.9474           0.9706
Specificity                 1.0000            0.9857           0.9730
Pos Pred Value              1.0000            0.9730           0.9429
Neg Pred Value              1.0000            0.9718           0.9863
Prevalence                  0.3333            0.3519           0.3148
Detection Rate              0.3333            0.3333           0.3056
Detection Prevalence        0.3333            0.3426           0.3241
Balanced Accuracy           1.0000            0.9665           0.9718
>
```

Exp no. : 11 **BREAST CANCER PREDICTION USING KNN**
Date : 27-02-23

AIM :-

To show Implementation of KNN using Breast Cancer dataset.

KNN ALGORITHM : -

K nearest neighbors or KNN Algorithm is a simple algorithm which uses the entire dataset in its training phase. Whenever a prediction is required for an unseen data instance, it searches through the entire training dataset for k-most similar instances and the data with the most similar instance is finally returned as the prediction.

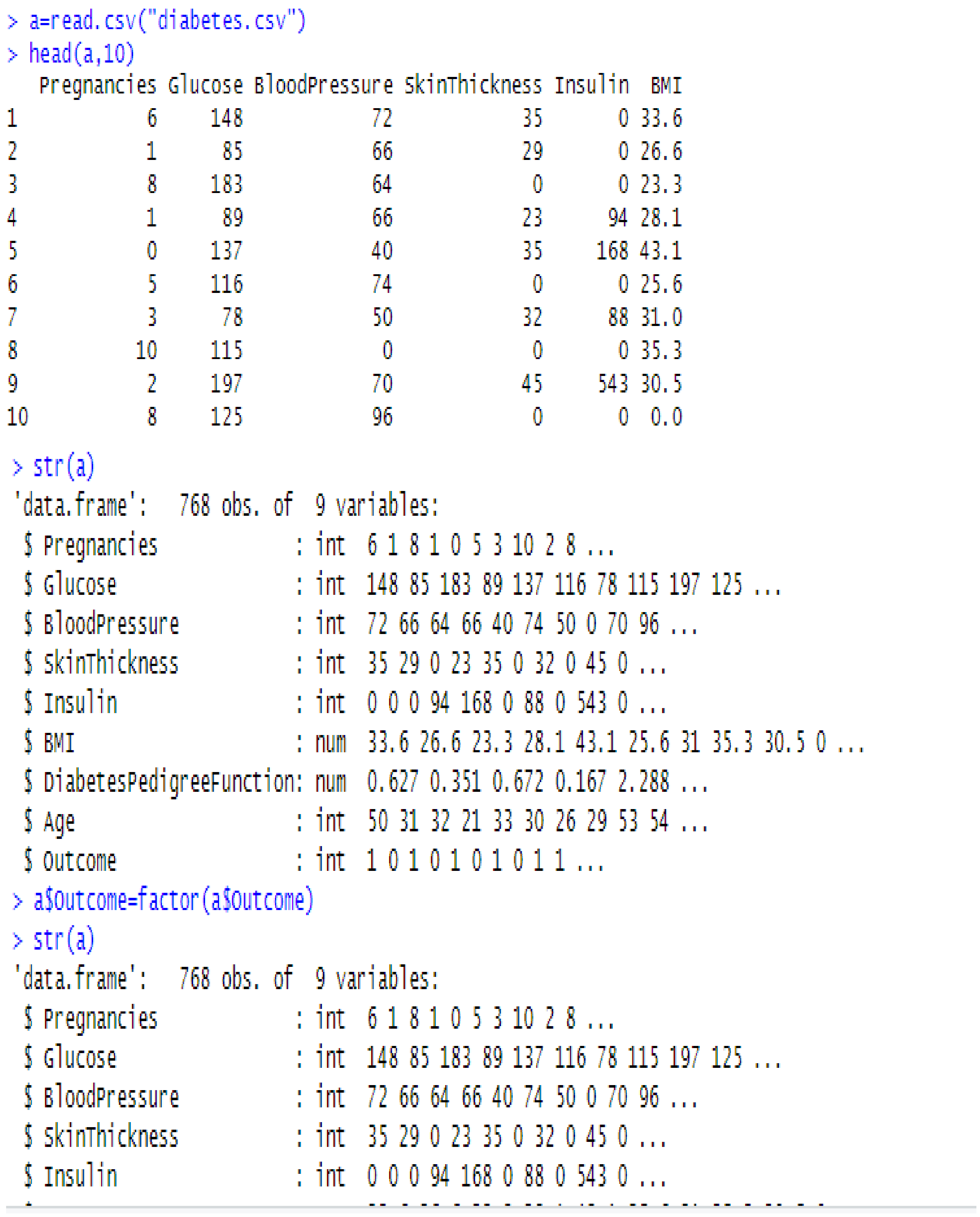
PROCEDURE:

1. Load the dataset
2. Initialize the value of k
3. For each sample in the training dataset.
   a) Calculate the distance between test data and each sample in the training data.
4. Based on the distance value, sort them in ascending order.
5. Get top k rows from the sorted array
6. Return the most frequent class of these rows.

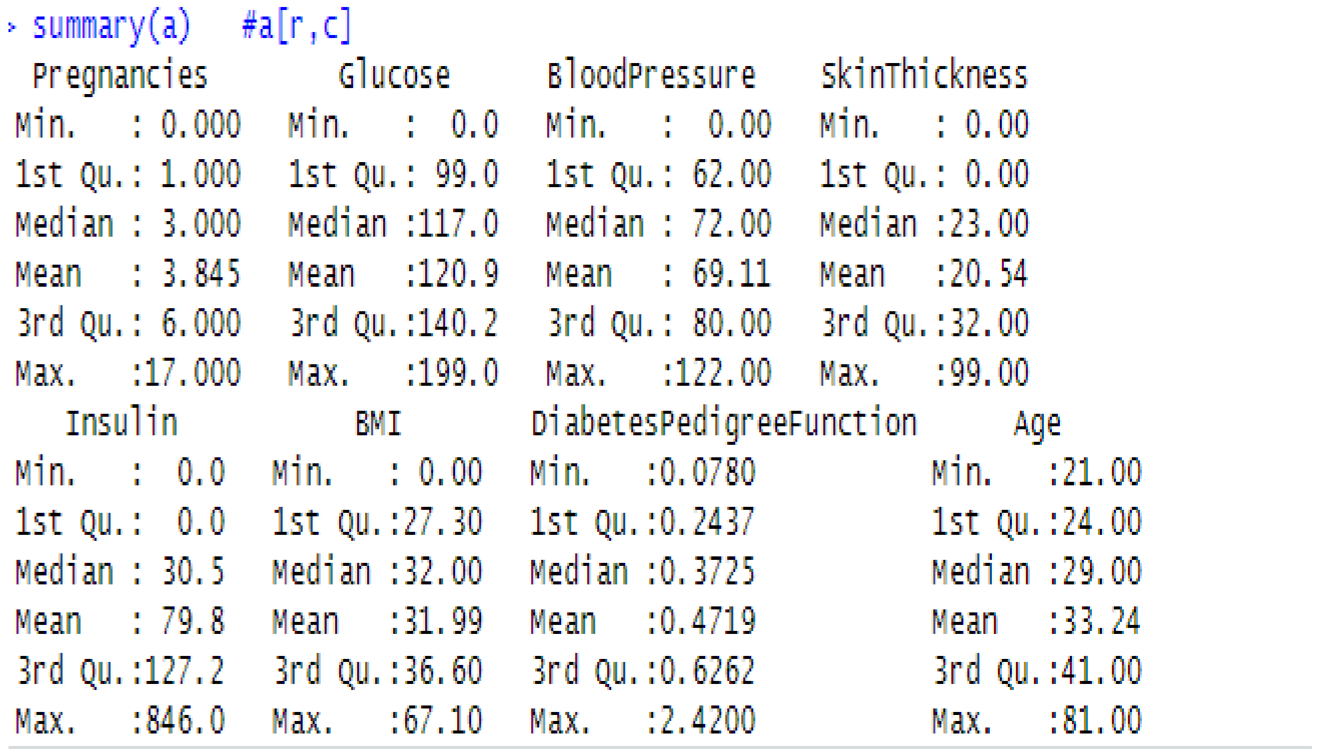
SOURCE CODE :

```
library(class)
library(caret)
a=read.csv("diabetes.csv")
head(a,10)
str(a)
a$Outcome=factor(a$Outcome)
str(a)
summary(a)   #a[r,c]
is.na(a)
colSums(is.na(a))
train=a[1:500,]
test=a[501:768,]
# knn(training,test )
pred_test=knn(train[,-9],test[,-9],train$Outcome,k=7)
cm=table(pred_test,test$Outcome)
cm
confusionMatrix(pred_test,test$Outcome)
```

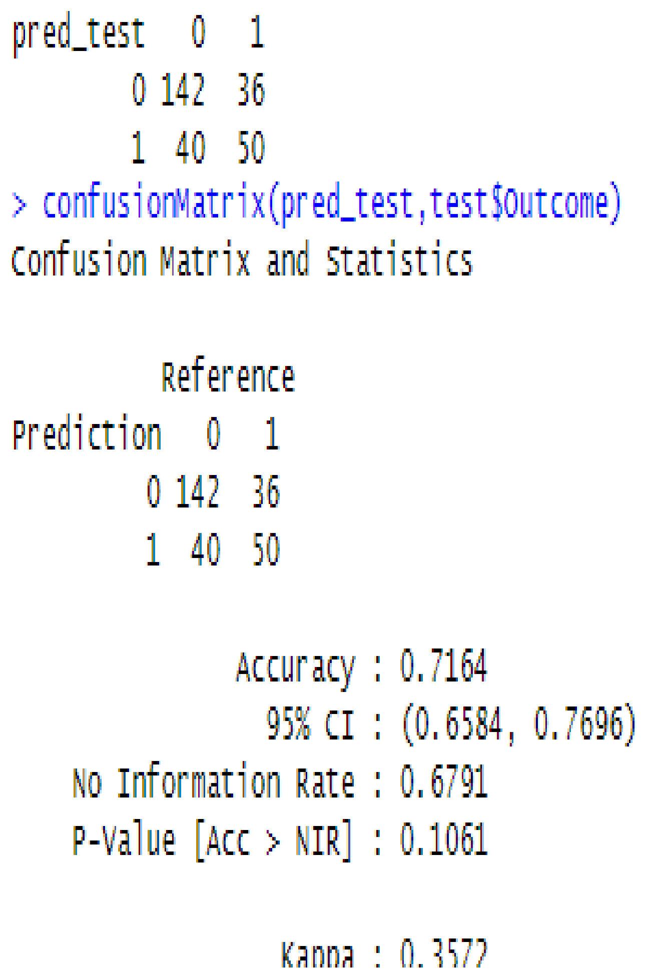
OUTPUT:



```
> a=read.csv("diabetes.csv")
> head(a,10)
   Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
1            6     148            72            35       0 33.6
2            1      85            66            29       0 26.6
3            8     183            64             0       0 23.3
4            1      89            66            23      94 28.1
5            0     137            40            35     168 43.1
6            5     116            74             0       0 25.6
7            3      78            50            32      88 31.0
8           10     115             0             0       0 35.3
9            2     197            70            45     543 30.5
10           8     125            96             0       0  0.0
> str(a)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
 $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
> a$Outcome=factor(a$Outcome)
> str(a)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
```



```
> summary(a)   #a[r,c]
  Pregnancies        Glucose      BloodPressure     SkinThickness
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
    Insulin           BMI        DiabetesPedigreeFunction      Age
 Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
 1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
 Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
 Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
 3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
 Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
```



```
pred_test   0   1
        0 142  36
        1  40  50
> confusionMatrix(pred_test,test$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 142  36
         1  40  50

               Accuracy : 0.7164
                 95% CI : (0.6584, 0.7696)
    No Information Rate : 0.6791
    P-Value [Acc > NIR] : 0.1061

                  Kappa : 0.3572
```

Exp no. : 12 **DIABETIES PREDICTION USING DECISION TREE**

Date : 06-03-23

AIM :-

To show the implementation of Decision Tree using Diabeties dataset.

DECISION TREE ALGORITHM:

Decision Tree is one of the most widely used and practical methods for supervised learning.They can be used to solve both regression and classification problems. It is robust to noisy data and capable of learning disjunctive expressions. The decision tree algorithms such as ID3, CART, C4.5 & ASSISTANT are very popular inductive inference algorithms, and they are successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

PROCEDURE:

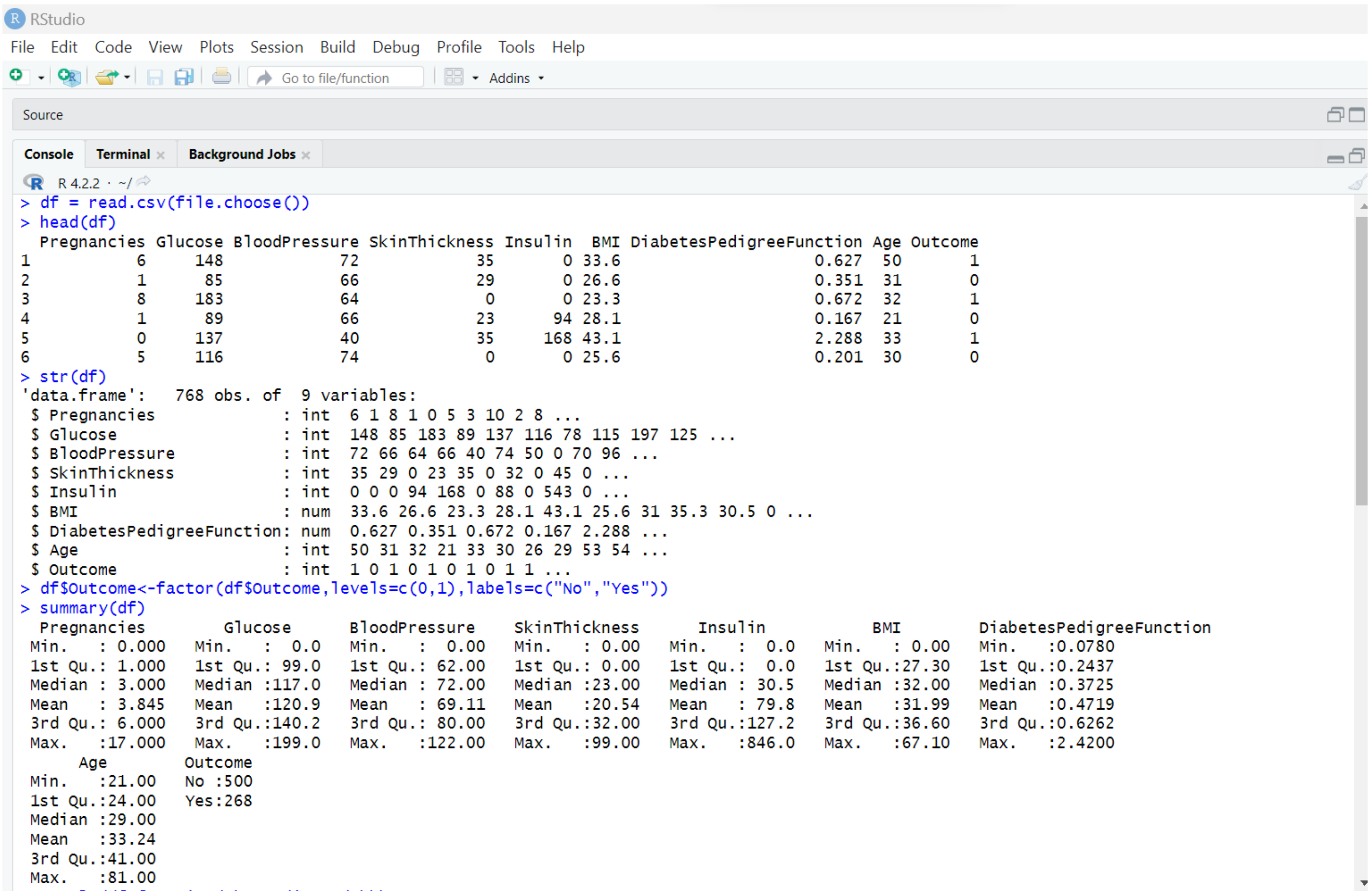
1. Load the dataset
2. Create train/test set.
   i. train <- sample(n,trunc(0.70*n))
   ii. df_train <- df [train,]
   iii. df_test <- df [- train,]
3. Build the model.
   i. Install Package **rpart**
   ii. Load the package .
   iii. Model <- rpart(Outcome~. ,data =df_train)
4. Make prediction.
   predict (model ,df_test,type=”class”)
5. Measure performance by confusion matrix.

SOURCE CODE:

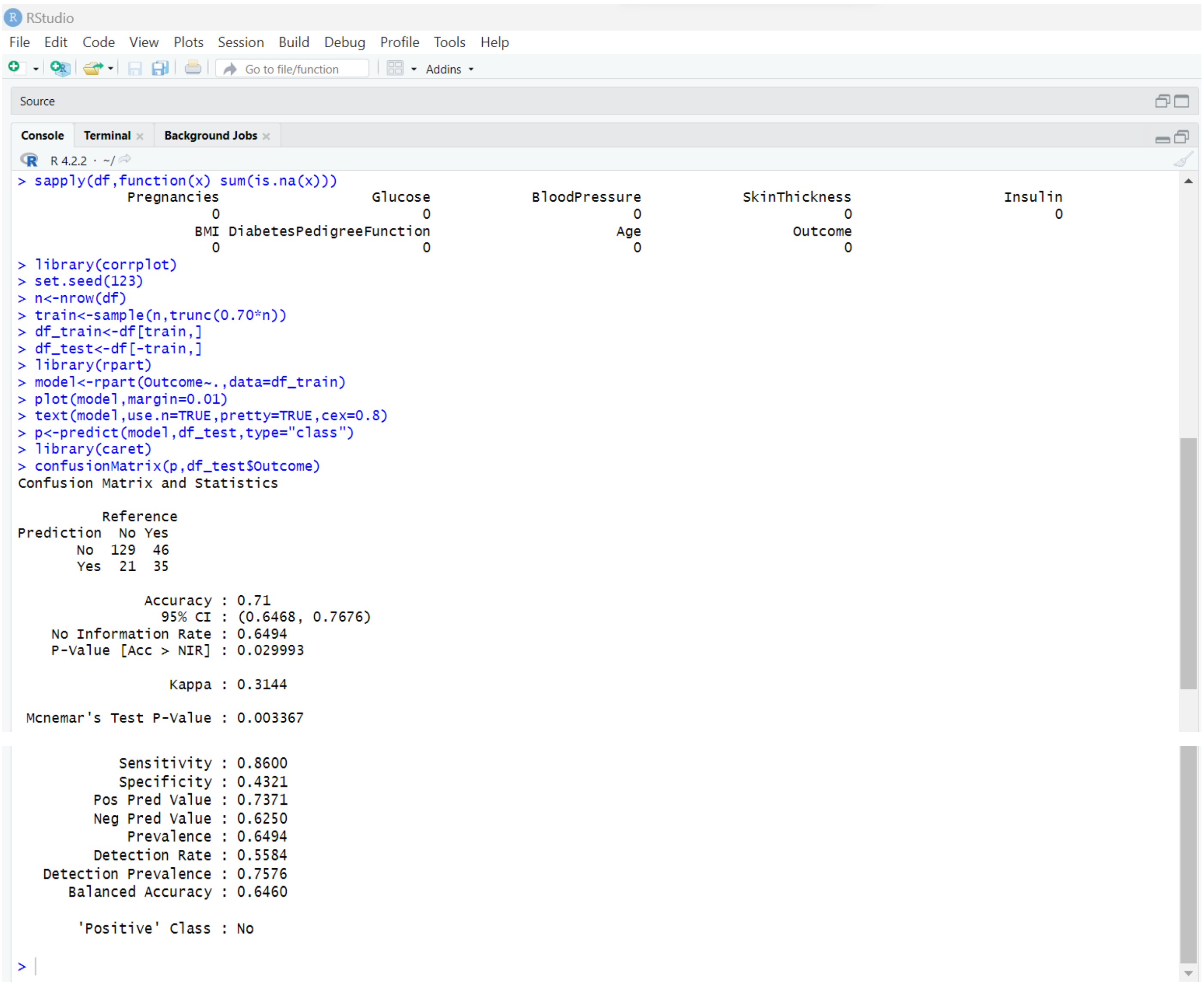
```
df = read.csv(file.choose())
head(df)
str(df)
df$Outcome<-factor(df$Outcome,levels=c(0,1),labels=c("No","Yes"))
summary(df)
sapply(df,function(x) sum(is.na(x)))
library(corrplot)
set.seed(123)
n<-nrow(df)
train<-sample(n,trunc(0.70*n))
df_train<-df[train,]
df_test<-df[-train,]
library(rpart)
model<-rpart(Outcome~.,data=df_train)
```

```
plot(model,margin=0.01)
text(model,use.n=TRUE,pretty=TRUE,cex=0.8)
p<-predict(model,df_test,type="class")
library(caret)
confusionMatrix(p,df_test$Outcome)
```

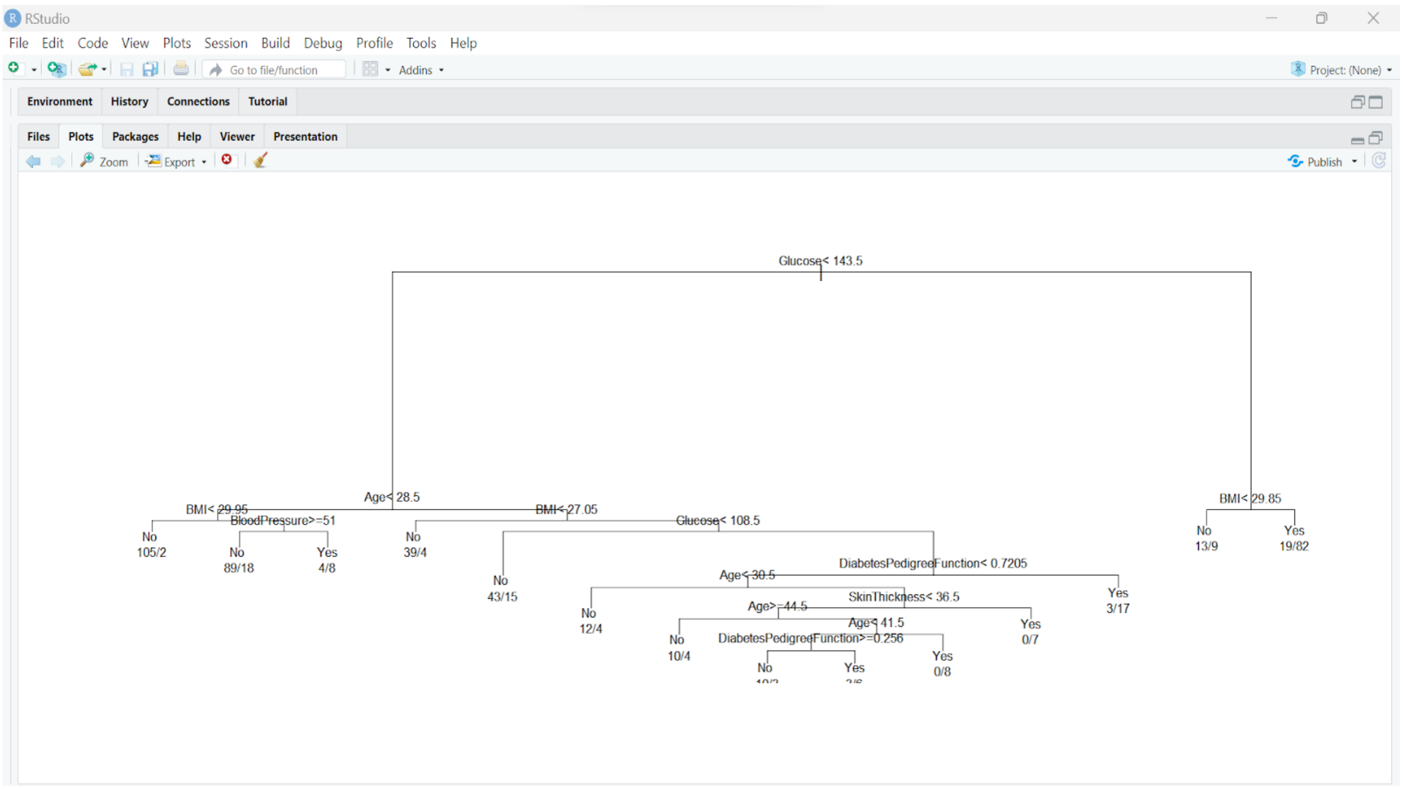
OUTPUT:



```
> df = read.csv(file.choose())
> head(df)
  Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI DiabetesPedigreeFunction Age Outcome
1           6     148            72            35       0 33.6                    0.627  50       1
2           1      85            66            29       0 26.6                    0.351  31       0
3           8     183            64             0       0 23.3                    0.672  32       1
4           1      89            66            23      94 28.1                    0.167  21       0
5           0     137            40            35     168 43.1                    2.288  33       1
6           5     116            74             0       0 25.6                    0.201  30       0
> str(df)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
 $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
> df$Outcome<-factor(df$Outcome,levels=c(0,1),labels=c("No","Yes"))
> summary(df)
  Pregnancies        Glucose      BloodPressure     SkinThickness      Insulin           BMI        DiabetesPedigreeFunction
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00   Min.   :  0.0   Min.   : 0.00   Min.   :0.0780
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00   1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00   Median : 30.5   Median :32.00   Median :0.3725
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54   Mean   : 79.8   Mean   :31.99   Mean   :0.4719
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00   3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00   Max.   :846.0   Max.   :67.10   Max.   :2.4200
      Age        Outcome
 Min.   :21.00   No :500
 1st Qu.:24.00   Yes:268
 Median :29.00
 Mean   :33.24
 3rd Qu.:41.00
 Max.   :81.00
```



```
> sapply(df,function(x) sum(is.na(x)))
             Pregnancies                  Glucose            BloodPressure            SkinThickness                  Insulin
                       0                        0                        0                        0                        0
                     BMI DiabetesPedigreeFunction                      Age                  Outcome
                       0                        0                        0                        0
> library(corrplot)
> set.seed(123)
> n<-nrow(df)
> train<-sample(n,trunc(0.70*n))
> df_train<-df[train,]
> df_test<-df[-train,]
> library(rpart)
> model<-rpart(Outcome~.,data=df_train)
> plot(model,margin=0.01)
> text(model,use.n=TRUE,pretty=TRUE,cex=0.8)
> p<-predict(model,df_test,type="class")
> library(caret)
> confusionMatrix(p,df_test$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  129  46
       Yes  21  35
                                          
               Accuracy : 0.71            
                 95% CI : (0.6468, 0.7676)
    No Information Rate : 0.6494          
    P-Value [Acc > NIR] : 0.029993        
                                          
                  Kappa : 0.3144          
                                          
 Mcnemar's Test P-Value : 0.003367        

            Sensitivity : 0.8600          
            Specificity : 0.4321          
         Pos Pred Value : 0.7371          
         Neg Pred Value : 0.6250          
             Prevalence : 0.6494          
         Detection Rate : 0.5584          
   Detection Prevalence : 0.7576          
      Balanced Accuracy : 0.6460          
                                          
       'Positive' Class : No              
                                          
> 
```



Exp no.: 13 WINE QUALITY PREDICTION USING RANDOM FOREST

Date: 13-03-23

AIM :-

To show the Implementation of Random Forest using Wine Quality Prediction dataset.

RANDOM FOREST ALGORITHM:

Random forest is a popular supervised machine learning algorithm—used for both classification and regression problems. It is based on the concept of ensemble learning, which enables users to combine multiple classifiers to solve a complex problem and to also improve the performance of the model. The random forest algorithm relies on multiple decision trees and accepts the results of the predictions from each tree. Based on the majority votes of predictions, it determines the final result.

PROCEDURE :

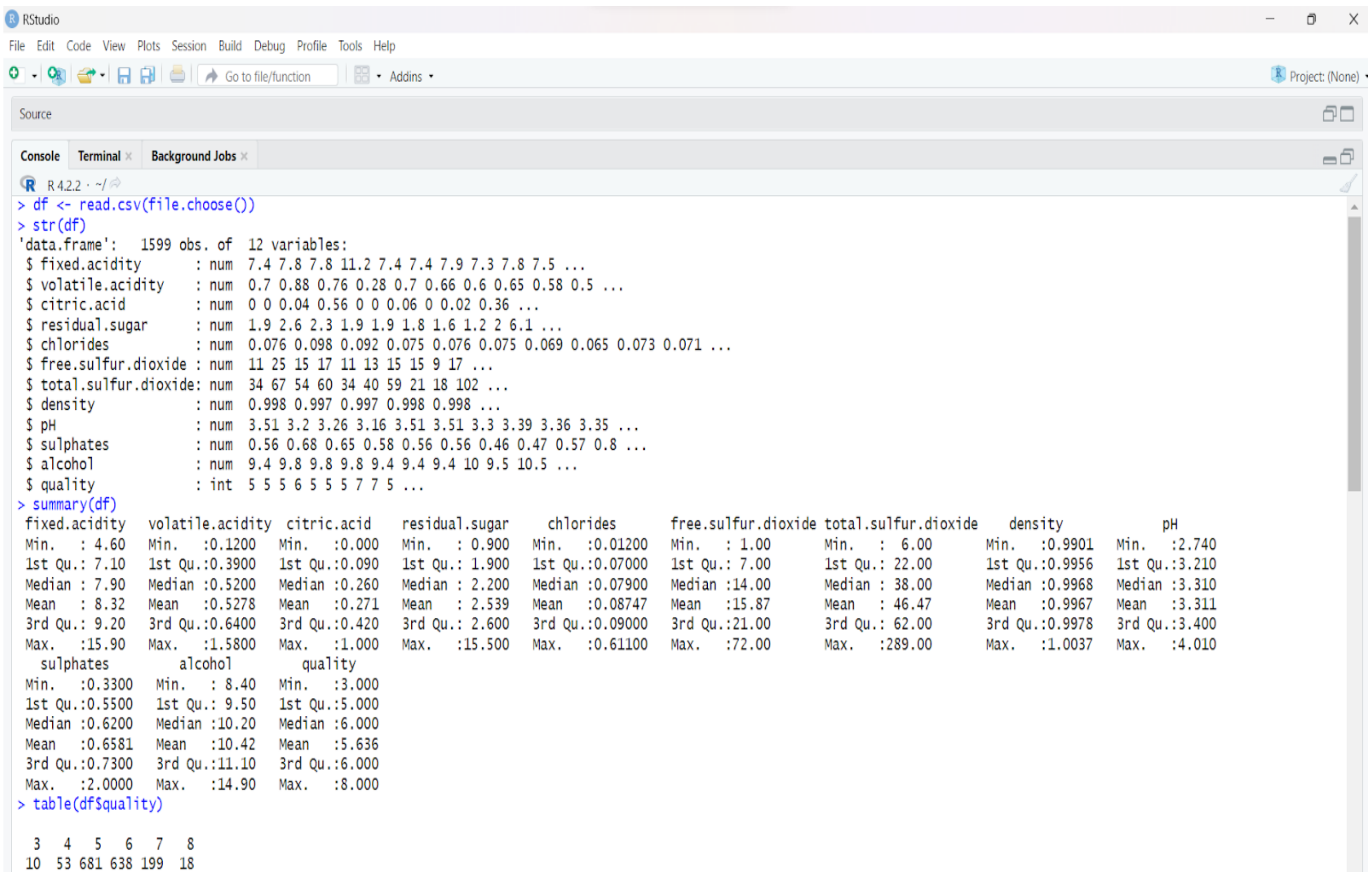
1. Load the dataset .
2. Create train/test set.
   i. id <- sample(2,nrow(df),replace=TRUE,prob=c(0.70,0.30))
   ii. df_train <- df [id==1,]
   iii. df_test <- df [id==2,]
3. Build the model.
   i. Install Package randomForest
   ii. Load the package .
   iii. Model <- randomForest(taste~. ,data =df_train,ntree=1000,ntry=5)
4. Make prediction.
   p= predict(model,df_test)
5. Calculate the votes for each of the predicted targets.
6. The most highly voted predicted target is the final prediction.

SOURCE CODE :

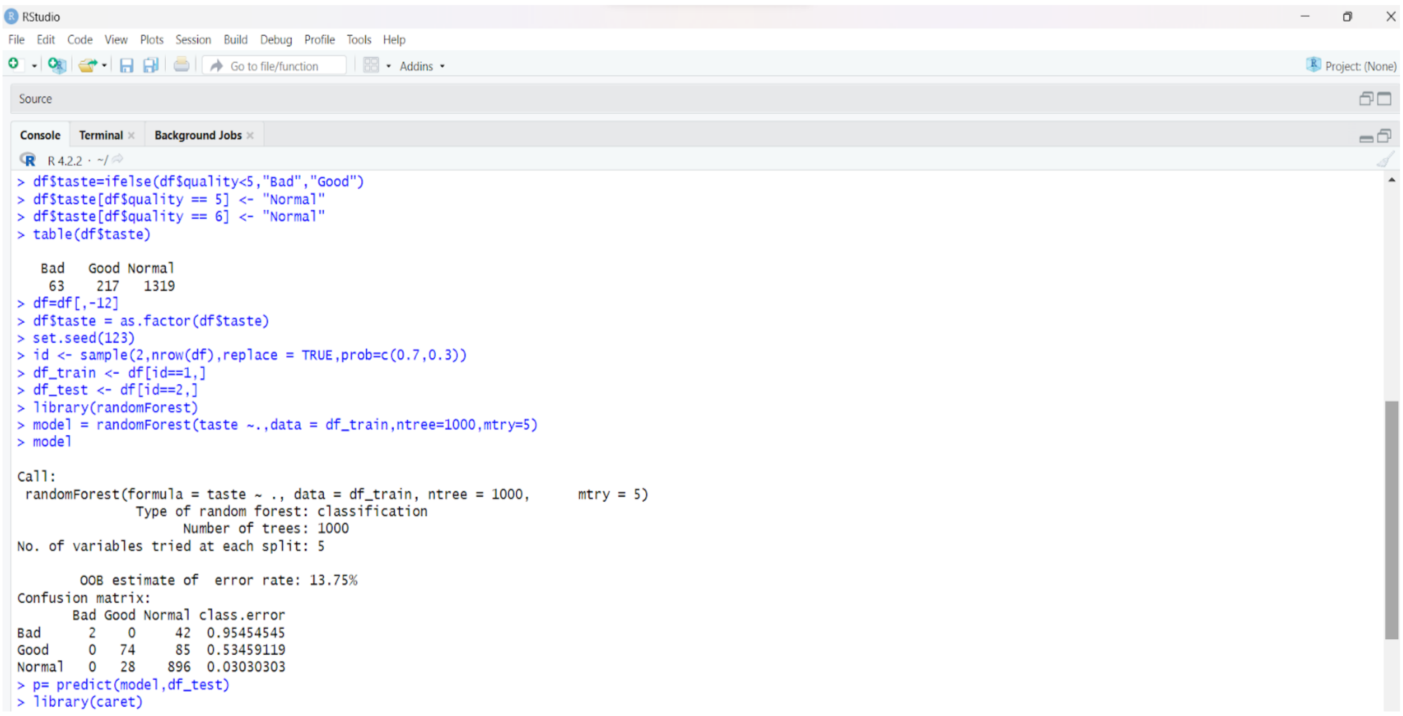
```
df <- read.csv(file.choose())
str(df)
summary(df)
table(df$quality)
df$taste=ifelse(df$quality<5,"Bad","Good")
df$taste[df$quality == 5] <- "Normal"
df$taste[df$quality == 6] <- "Normal"
table(df$taste)
df=df[,-12]
df$taste = as.factor(df$taste)
set.seed(123)
id <- sample(2,nrow(df),replace = TRUE,prob=c(0.7,0.3))
df_train <- df[id==1,]
```

```
df_test <- df[id==2,]
install.packages("randomForest")
library(randomForest)
model = randomForest(taste ~.,data = df_train,ntree=1000,mtry=5)
model
p= predict(model,df_test)
library(caret)
confusionMatrix(p,df_test$taste)
```

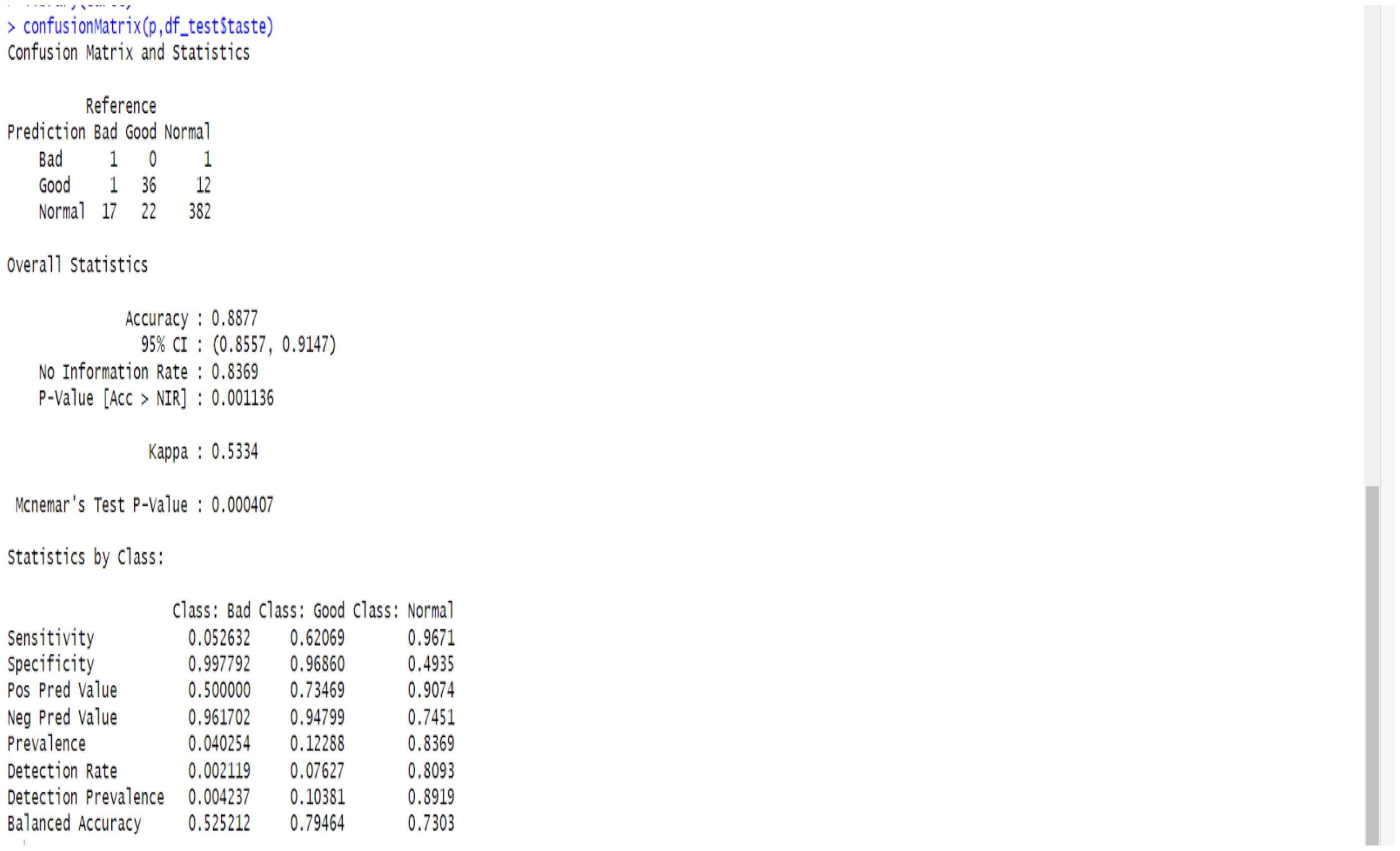
OUTPUT :



```
> df <- read.csv(file.choose())
> str(df)
'data.frame':	1599 obs. of  12 variables:
 $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
 $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
 $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
 $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
 $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
 $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
 $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
 $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
 $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
 $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
 $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
 $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
> summary(df)
 fixed.acidity   volatile.acidity  citric.acid    residual.sugar     chlorides       free.sulfur.dioxide total.sulfur.dioxide    density             pH
 Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900   Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901   Min.   :2.740
 1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900   1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956   1st Qu.:3.210
 Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200   Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968   Median :3.310
 Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539   Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967   Mean   :3.311
 3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600   3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978   3rd Qu.:3.400
 Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500   Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037   Max.   :4.010
   sulphates         alcohol          quality
 Min.   :0.3300   Min.   : 8.40   Min.   :3.000
 1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
 Median :0.6200   Median :10.20   Median :6.000
 Mean   :0.6581   Mean   :10.42   Mean   :5.636
 3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
 Max.   :2.0000   Max.   :14.90   Max.   :8.000
> table(df$quality)

  3   4   5   6   7   8 
 10  53 681 638 199  18 
```



```
> df$taste=ifelse(df$quality<5,"Bad","Good")
> df$taste[df$quality == 5] <- "Normal"
> df$taste[df$quality == 6] <- "Normal"
> table(df$taste)

   Bad   Good Normal 
    63    217   1319 
> df=df[,-12]
> df$taste = as.factor(df$taste)
> set.seed(123)
> id <- sample(2,nrow(df),replace = TRUE,prob=c(0.7,0.3))
> df_train <- df[id==1,]
> df_test <- df[id==2,]
> library(randomForest)
> model = randomForest(taste ~.,data = df_train,ntree=1000,mtry=5)
> model

Call:
 randomForest(formula = taste ~ ., data = df_train, ntree = 1000,      mtry = 5) 
              Type of random forest: classification
                    Number of trees: 1000
No. of variables tried at each split: 5

        OOB estimate of  error rate: 13.75%
Confusion matrix:
       Bad Good Normal class.error
Bad      2    0     42  0.95454545
Good     0   74     85  0.53459119
Normal   0   28    896  0.03030303
> p= predict(model,df_test)
> library(caret)
```



```
> confusionMatrix(p,df_test$taste)
Confusion Matrix and Statistics

          Reference
Prediction Bad Good Normal
    Bad      1    0      1
    Good     1   36     12
    Normal  17   22    382

Overall Statistics
                                          
               Accuracy : 0.8877          
                 95% CI : (0.8557, 0.9147)
    No Information Rate : 0.8369          
    P-Value [Acc > NIR] : 0.001136        
                                          
                  Kappa : 0.5334          
                                          
 Mcnemar's Test P-Value : 0.000407        

Statistics by Class:

                     Class: Bad Class: Good Class: Normal
Sensitivity            0.052632     0.62069        0.9671
Specificity            0.997792     0.96860        0.4935
Pos Pred Value         0.500000     0.73469        0.9074
Neg Pred Value         0.961702     0.94799        0.7451
Prevalence             0.040254     0.12288        0.8369
Detection Rate         0.002119     0.07627        0.8093
Detection Prevalence   0.004237     0.10381        0.8919
Balanced Accuracy      0.525212     0.79464        0.7303
```

Exp no. : 14 **Clustering USArrest Dataset using KMeans Algorithm**
Date : 20-03-23
AIM :-

To show the implementation of KMeans Algorithm using USArrest Dataset

KMEANS ALGORITHM:

Kmeans clustering is one of the simplest and popular unsupervised machine learning algorithms. A cluster refers to a collection of data points aggregated together because of certain similarities. The number k refers to the number of centroids. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to the nearest cluster.

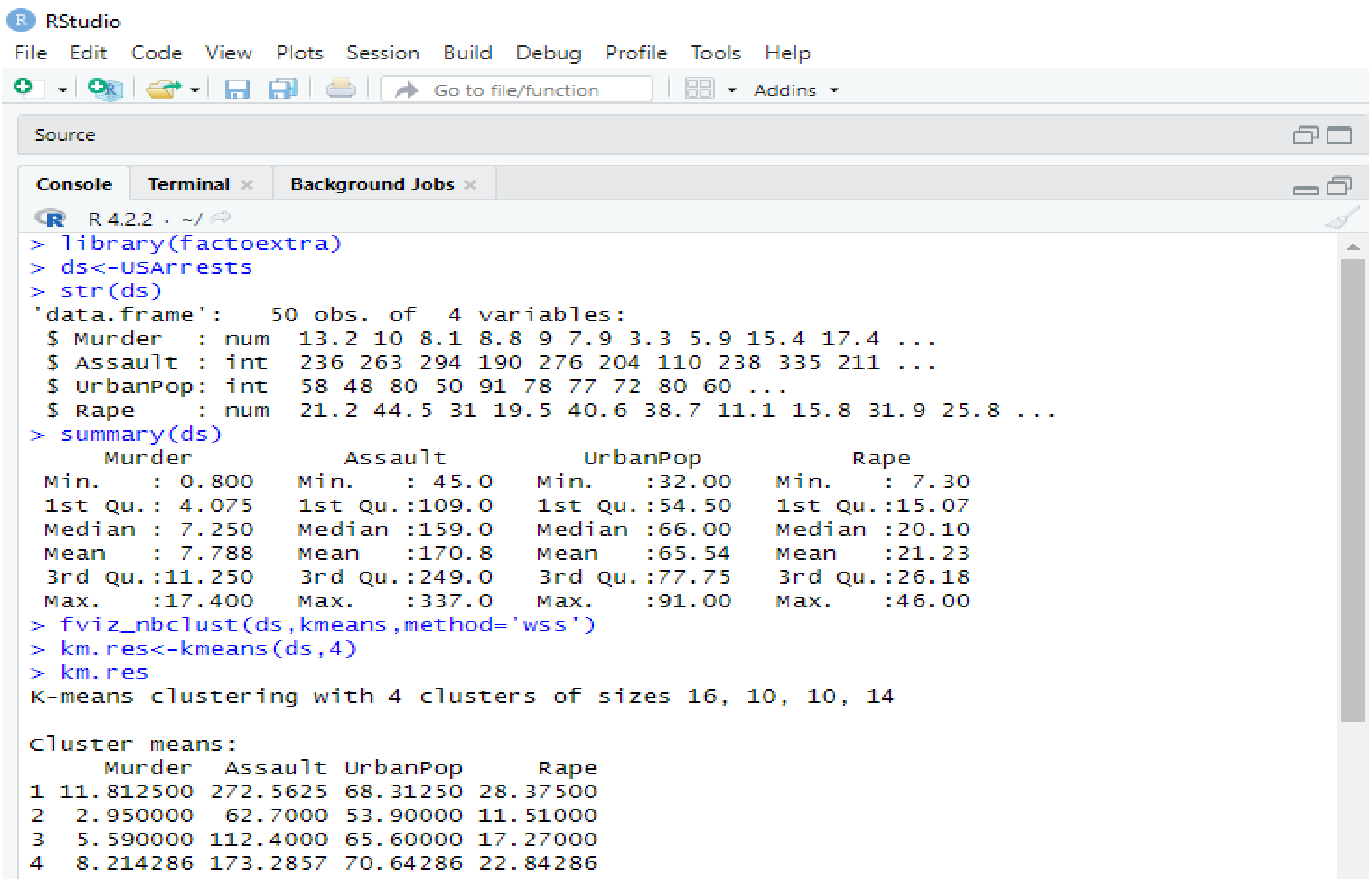
PROCEDURE :

1. Load the dataset .
2. Choose the number K clusters.
3. Select at random K points
4. Assign each data point to closest centroid that forms K clusters.
5. Compute and place the new centroid of each centroid.
6. Reassign each data point to new cluster.

SOURCE CODE :

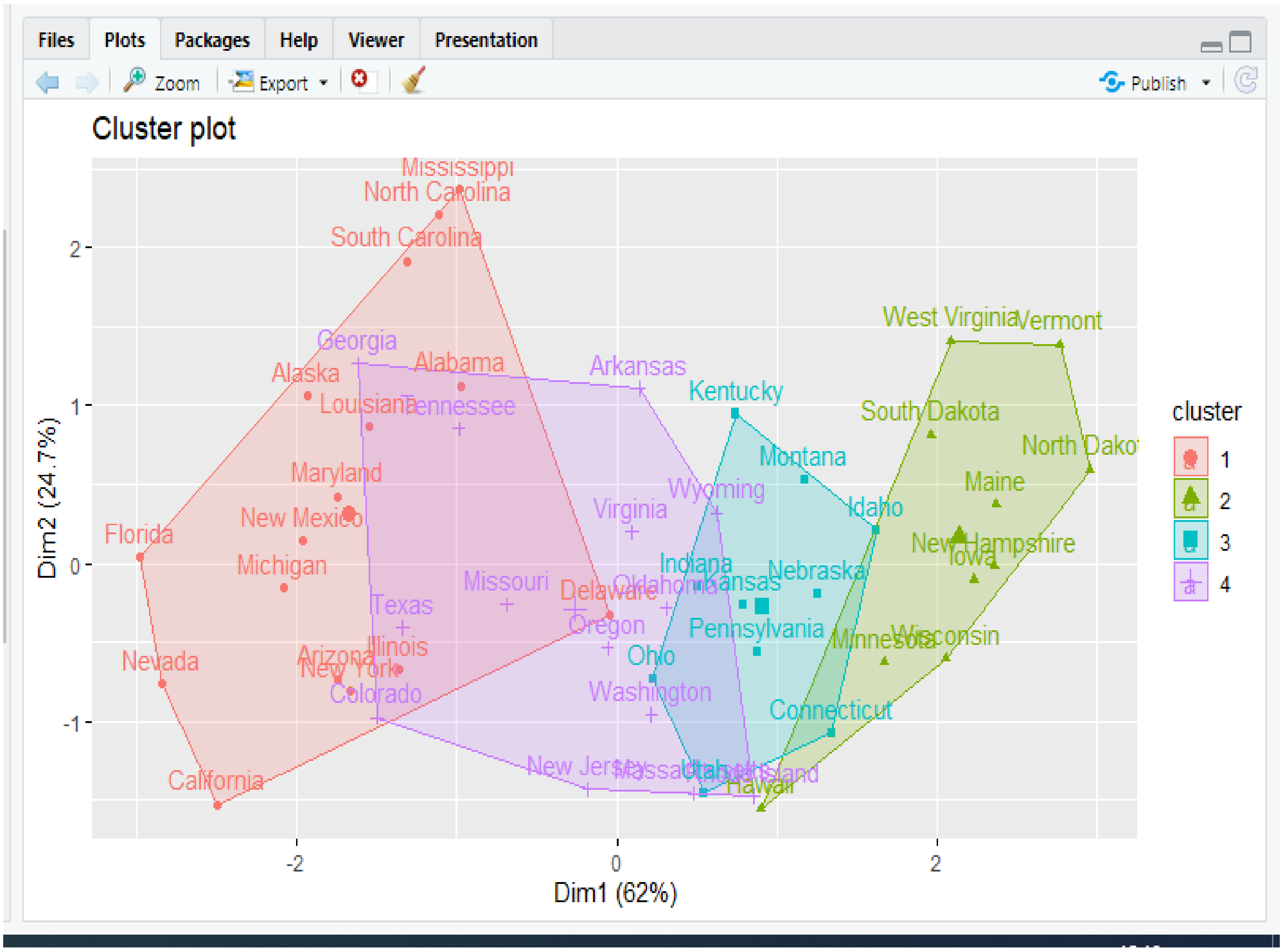
```
install.packages(“factoextra”)
library(factoextra)
ds <- USArrests
str(ds)
summary(ds)
fviz_nbclust(ds, kmeans, method =”wss”)
km.res <- kmeans(ds, 4)
km.res
fviz_cluster(km.res, data = ds)
```

OUTPUT:



```
> library(factoextra)
> ds<-USArrests
> str(ds)
'data.frame':	50 obs. of  4 variables:
 $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
 $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
 $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
 $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
> summary(ds)
     Murder          Assault         UrbanPop          Rape      
 Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30  
 1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07  
 Median : 7.250   Median :159.0   Median :66.00   Median :20.10  
 Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23  
 3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18  
 Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00  
> fviz_nbclust(ds,kmeans,method='wss')
> km.res<-kmeans(ds,4)
> km.res
K-means clustering with 4 clusters of sizes 16, 10, 10, 14

Cluster means:
     Murder  Assault UrbanPop     Rape
1 11.812500 272.5625 68.31250 28.37500
2  2.950000  62.7000 53.90000 11.51000
3  5.590000 112.4000 65.60000 17.27000
4  8.214286 173.2857 70.64286 22.84286
```

```
Clustering vector:
       Alabama         Alaska        Arizona       Arkansas     California 
             1              1              1              4              1 
      Colorado    Connecticut       Delaware        Florida        Georgia 
             4              3              1              1              4 
        Hawaii          Idaho       Illinois        Indiana           Iowa 
             2              3              1              3              2 
        Kansas       Kentucky      Louisiana          Maine       Maryland 
             3              3              1              2              1 
 Massachusetts       Michigan      Minnesota    Mississippi       Missouri 
             4              1              2              1              4 
       Montana       Nebraska         Nevada  New Hampshire     New Jersey 
             3              3              1              2              4 
    New Mexico       New York North Carolina   North Dakota           Ohio 
             1              1              1              2              3 
      Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina 
             4              4              3              4              1 
  South Dakota      Tennessee          Texas           Utah        Vermont 
             2              4              4              3              2 
      Virginia     Washington  West Virginia      Wisconsin        Wyoming 
             4              4              2              2              4 

Within cluster sum of squares by cluster:
[1] 19563.863  4547.914  1480.210  9136.643
 (between_SS / total_SS =  90.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"      
> fviz_cluster(km.res,data=ds)
> |
```



Files
Plots
Packages
Help
Viewer
Presentation
Zoom
Export
Publish
Cluster plot
Dim1 (62%)
Dim2 (24.7%)
cluster
1
2
3
4
Mississippi
North Carolina
South Carolina
Georgia
Alabama
Alaska
Louisiana
Tennessee
Arkansas
Kentucky
West Virginia
Vermont
South Dakota
North Dakota
Montana
Maine
Wyoming
Idaho
Maryland
New Mexico
Virginia
Florida
Michigan
Indiana
Kansas
Nebraska
New Hampshire
Iowa
Missouri
Delaware
Oklahoma
Texas
Oregon
Pennsylvania
Minnesota
Wisconsin
Nevada
Arizona
Illinois
New York
Ohio
Colorado
Washington
Connecticut
California
New Jersey
Massachusetts
Utah
Rhode Island
Hawaii

Exp no :15 Clustering USArrests using Agglomerative Hierarchical Clustering
Date : 28-03-23

AIM :-

To show the implementation of Agglomerative Hierarchical Clustering using USArrest Dataset .

AGGLOMERATIVE HIERARCHICAL CLUSTERING :

The Agglomerative Hierarchical Clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It's a “bottom-up” approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

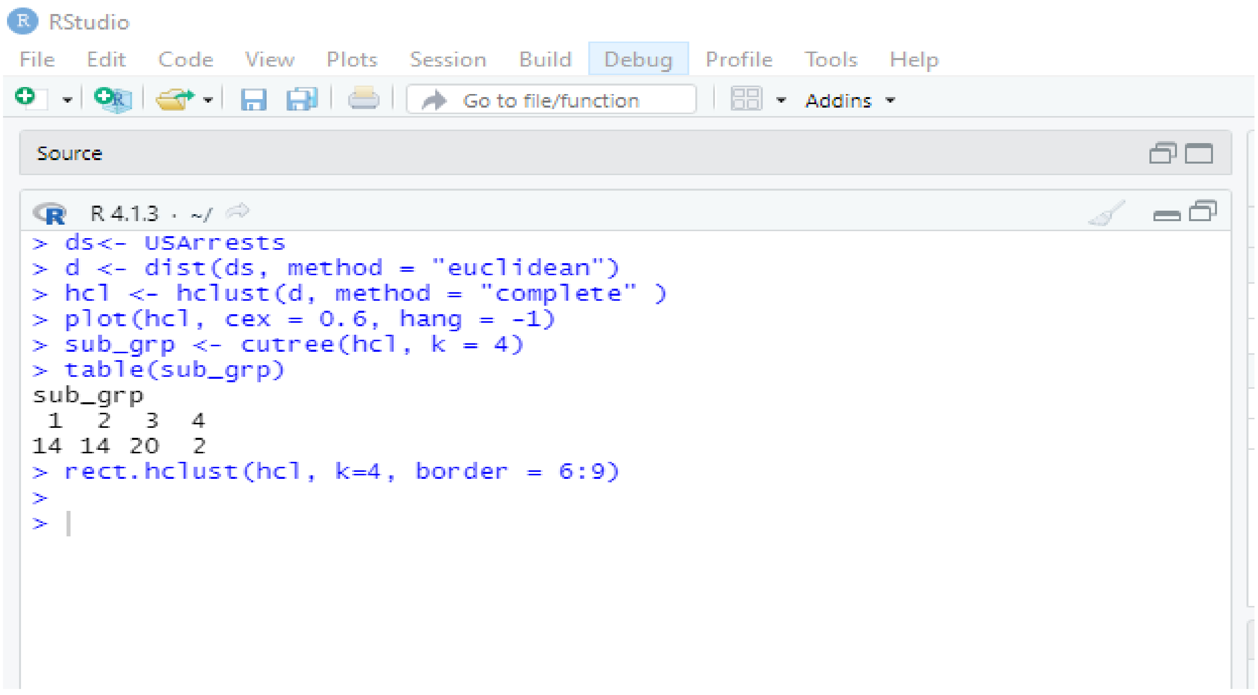
PROCEDURE:

1. Make each data point a single-point cluster → forms N clusters
2. Take the two closest clusters and make them one cluster → Forms N-1 clusters.
3. Repeat step-3 until you are left with only one cluster.

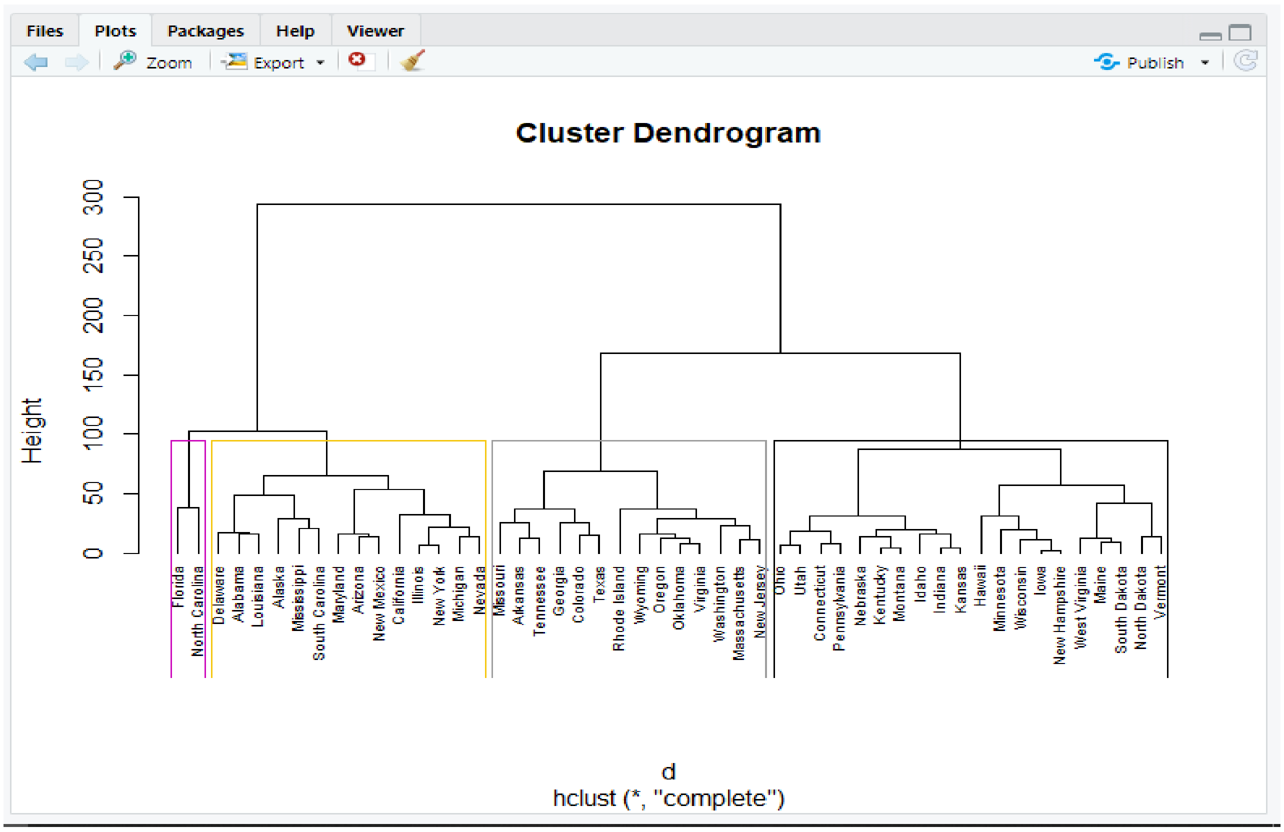
SOURCE CODE:

```
ds<- USArrests
d <- dist(ds, method = "euclidean")
hcl <- hclust(d, method = "complete" )
plot(hcl, cex = 0.6, hang = -1)
sub_grp <- cutree(hcl, k = 4)
table(sub_grp)
rect.hclust(hcl, k=4, border = 6:9)
```

OUTPUT :


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
R 4.1.3 · ~/

```
> ds<- USArrests
> d <- dist(ds, method = "euclidean")
> hcl <- hclust(d, method = "complete" )
> plot(hcl, cex = 0.6, hang = -1)
> sub_grp <- cutree(hcl, k = 4)
> table(sub_grp)
sub_grp
 1  2  3  4
14 14 20  2
> rect.hclust(hcl, k=4, border = 6:9)
>
> |
```


Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
Cluster Dendrogram
Height
0
50
100
150
200
250
300
Florida
North Carolina
Delaware
Alabama
Louisiana
Alaska
Mississippi
South Carolina
Maryland
Arizona
New Mexico
California
Illinois
New York
Michigan
Nevada
Missouri
Arkansas
Tennessee
Georgia
Colorado
Texas
Rhode Island
Wyoming
Oregon
Oklahoma
Virginia
Washington
Massachusetts
New Jersey
Ohio
Utah
Connecticut
Pennsylvania
Nebraska
Kentucky
Montana
Idaho
Indiana
Kansas
Hawaii
Minnesota
Wisconsin
Iowa
New Hampshire
West Virginia
Maine
South Dakota
North Dakota
Vermont
d
hclust (*, "complete")

Exp no : 16 CLUSTERING USARRESTS USING DIVISIVE HIERARCHICAL CLUSTERING
Date: 05-04-23

AIM :-

To show the implementation of Divisive Hierarchical Clustering using USArrest Dataset .

DIVISIVE HIERARCHICAL CLUSTERING:

Divisive Hierarchical Clustering is a top-down clustering method where we assign all of the observations to a single cluster and then partition the cluster to two least similar clusters.This is exactly opposite to Agglomerative clustering.Finally, we proceed recursively on each cluster until there is one cluster for each observation.

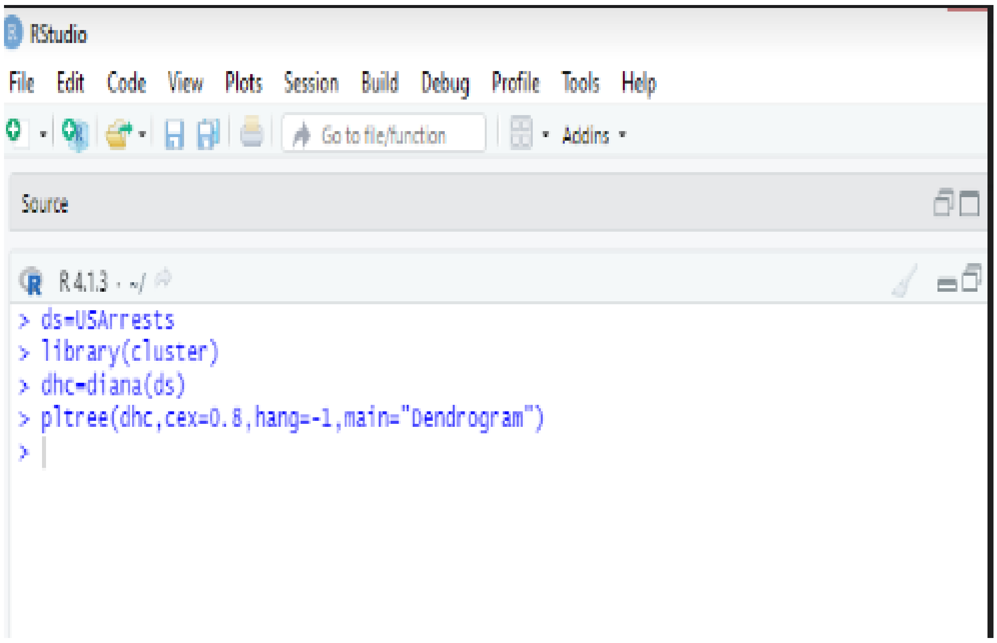
PROCEDURE:

1. Make each data point a single-point cluster → forms N clusters
2. Take the two closest clusters and make them one cluster → Forms N-1 clusters.
3. Repeat step-3 until you are left with only one cluster.

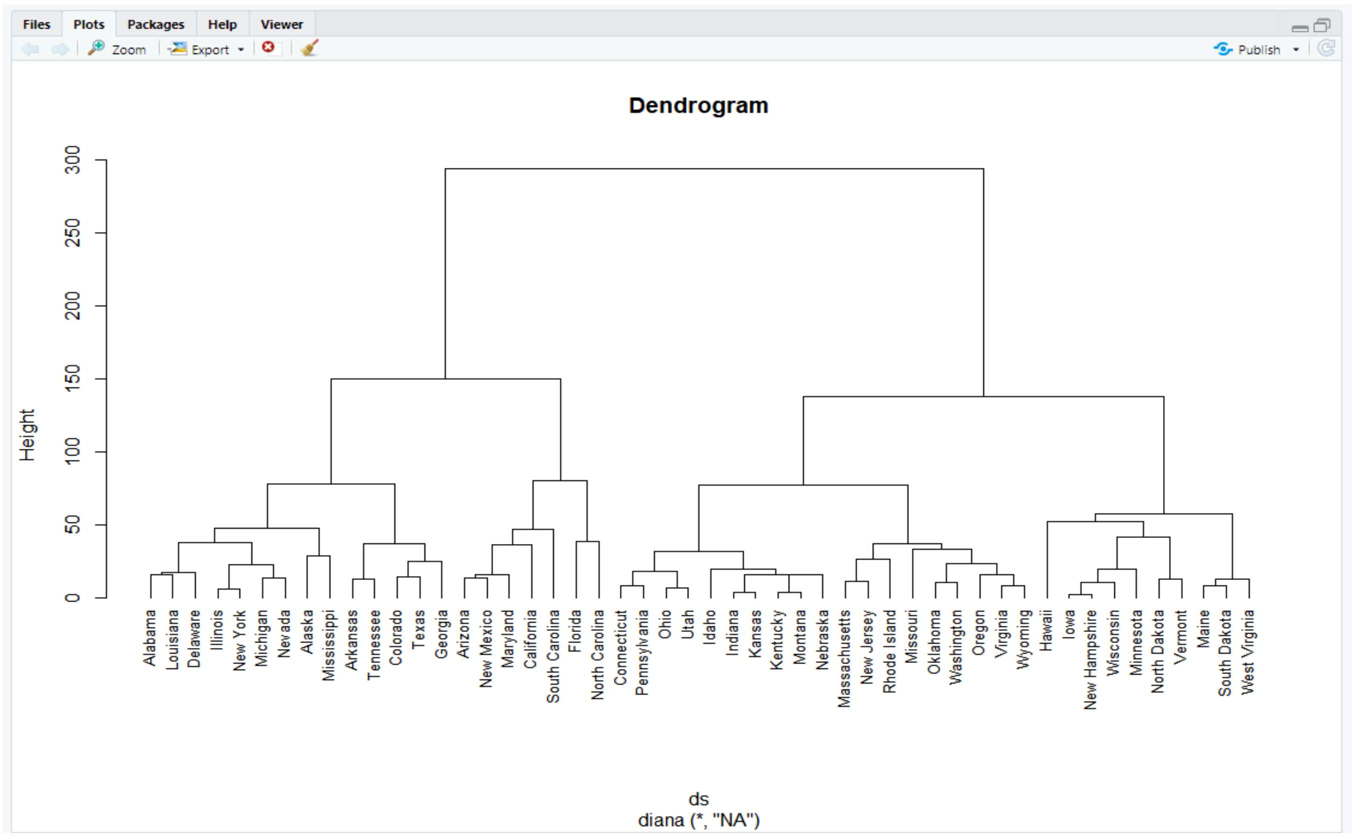
SOURCE CODE:

```
library("cluster")
ds<- USArrests
dhc <- diana(ds)
pltree(dhc, cex = 0.6, hang = -1, main = "Dendrogram")
```

OUTPUT:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
R 4.1.3 · ~/

```
> ds=USArrests
> library(cluster)
> dhc=diana(ds)
> pltree(dhc,cex=0.8,hang=-1,main="Dendrogram")
>
```


Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
Dendrogram
Height
0
50
100
150
200
250
300
Alabama
Louisiana
Delaware
Illinois
New York
Michigan
Nevada
Alaska
Mississippi
Arkansas
Tennessee
Colorado
Texas
Georgia
Arizona
New Mexico
Maryland
California
South Carolina
Florida
North Carolina
Connecticut
Pennsylvania
Ohio
Utah
Idaho
Indiana
Kansas
Kentucky
Montana
Nebraska
Massachusetts
New Jersey
Rhode Island
Missouri
Oklahoma
Washington
Oregon
Virginia
Wyoming
Hawaii
Iowa
New Hampshire
Wisconsin
Minnesota
North Dakota
Vermont
Maine
South Dakota
West Virginia
ds
diana (*, "NA")

Exp no. : 17 **CLUSTERING IRIS DATASET USING DBSCAN**
Date: 05-04-23

AIM:-

To show the implementation of DBSCAN using Iris dataset.

DBSCAN:

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. It is a density-based clustering algorithm that works on the assumption that clusters are dense regions in space separated by regions of lower density.It groups 'densely grouped' data points into a single cluster. It can identify clusters in large spatial datasets by looking at the local density of the data points. DBSCAN requires only two parameters: epsilon and minPoints. Epsilon is the radius of the circle to be created around each data point to check the density and minPoints is the minimum number of data points required inside that circle for that data point to be classified as a Core point.

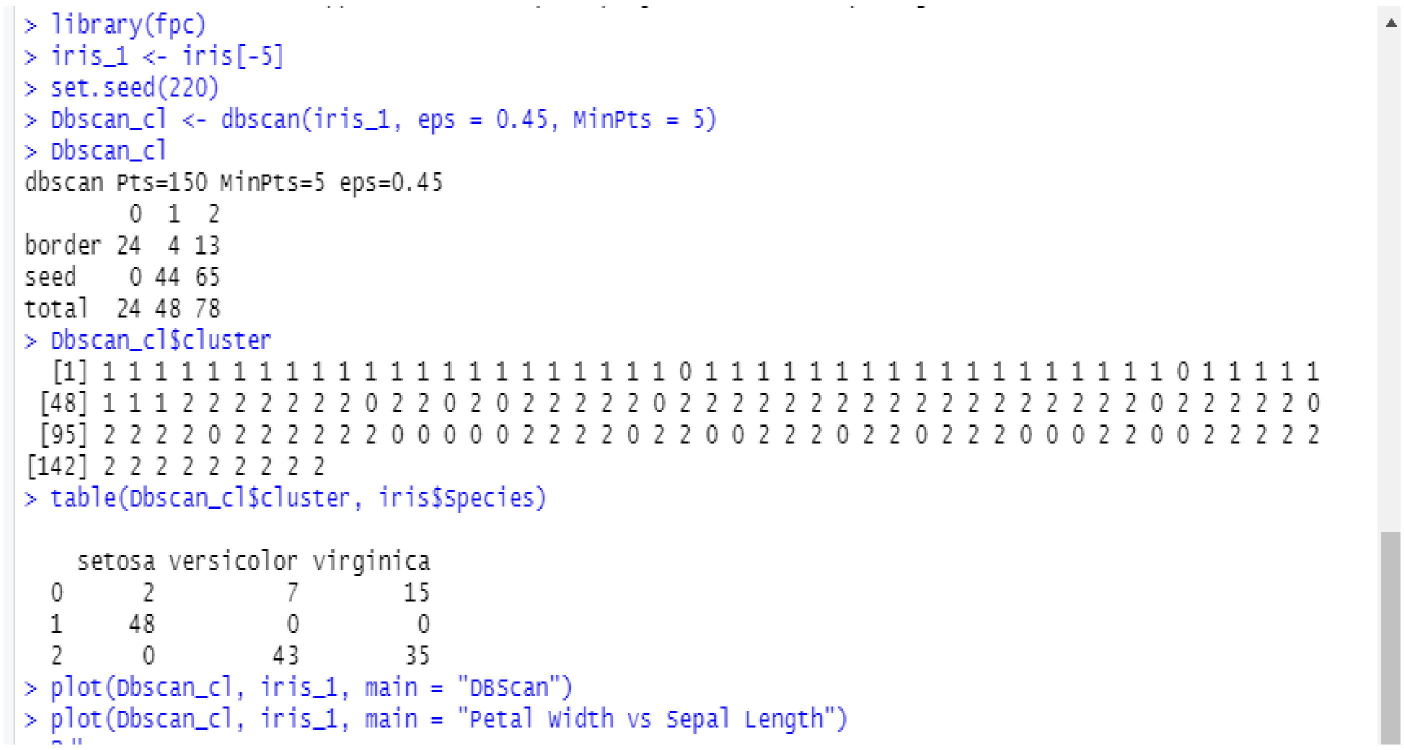
PROCEDURE:

1. Load the dataset.
2. Randomly select a point p.
3. Retrieve all the points that are density reachable from p with regard to Maximum radius of the neighbourhood(eps) and minimum number of points within eps neighborhood(Min pts).
4. If the number of points in the neighborhood is more than Min pts then p is a core point.
5. For p core points, a cluster is formed. If p is not a core point, then mark it as a noise/outlier and move to the next point.
6. Continue the process until all the points have been processed.

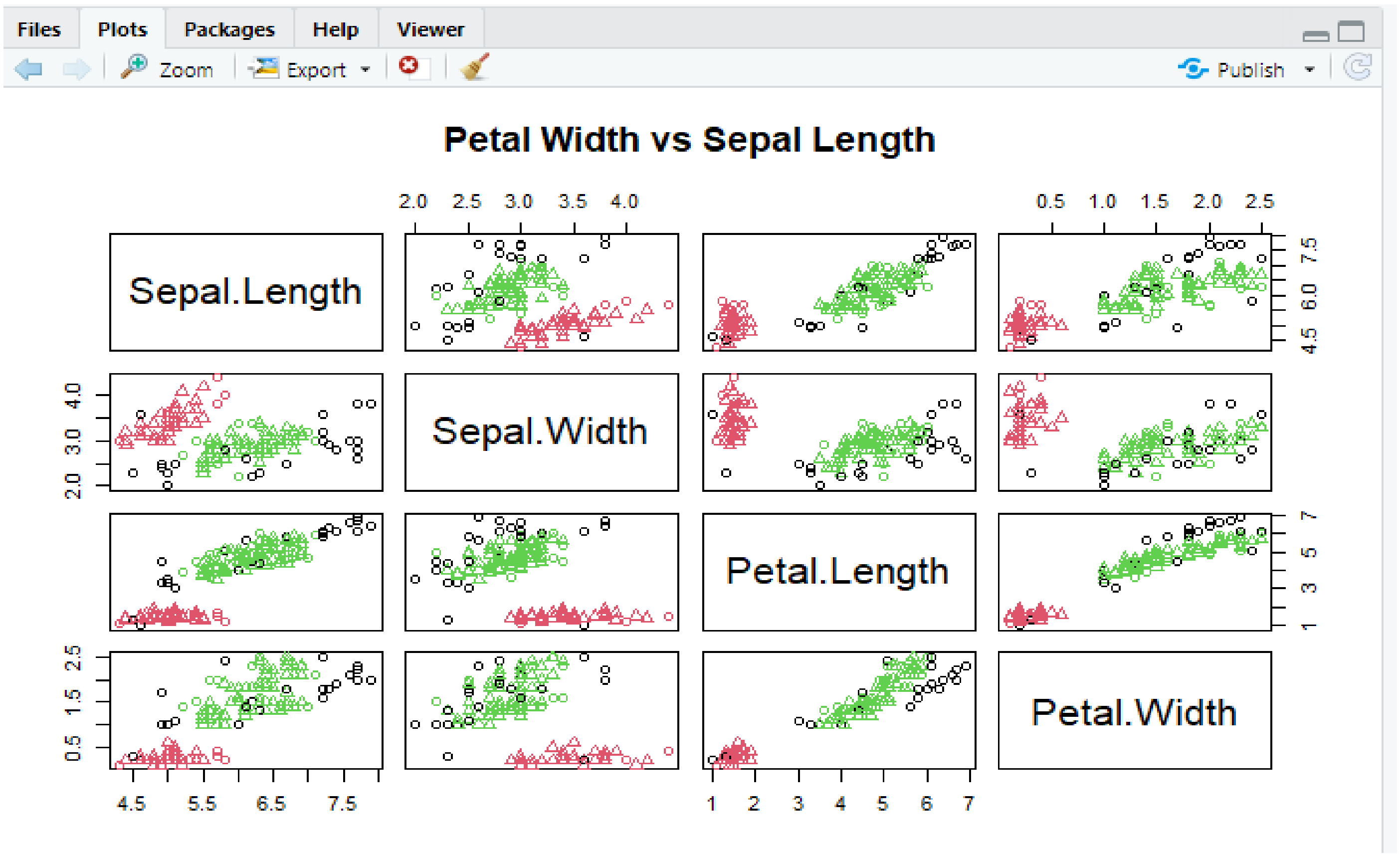
SOURCE CODE:

```
install.packages("fpc")
library(fpc)
iris_1 <- iris[-5]
set.seed(220)
Dbscan_cl <- dbscan(iris_1, eps = 0.45, MinPts = 5)
Dbscan_cl
Dbscan_cl$cluster
table(Dbscan_cl$cluster, iris$Species)
plot(Dbscan_cl, iris_1, main = "DBScan")
plot(Dbscan_cl, iris_1, main = "Petal Width vs Sepal Length")
```

OUTPUT:



```
> library(fpc)
> iris_1 <- iris[-5]
> set.seed(220)
> Dbscan_cl <- dbscan(iris_1, eps = 0.45, MinPts = 5)
> Dbscan_cl
dbscan Pts=150 MinPts=5 eps=0.45
        0  1  2
border 24  4 13
seed    0 44 65
total  24 48 78
> Dbscan_cl$cluster
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 [48] 1 1 1 2 2 2 2 2 2 2 0 2 2 0 2 0 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 0
 [95] 2 2 2 2 0 2 2 2 2 2 2 0 0 0 0 0 2 2 2 2 0 2 2 0 0 2 2 2 0 2 2 0 2 2 2 0 0 0 2 2 0 0 2 2 2 2 2
[142] 2 2 2 2 2 2 2 2 2
> table(Dbscan_cl$cluster, iris$Species)

    setosa versicolor virginica
  0      2          7        15
  1     48          0         0
  2      0         43        35
> plot(Dbscan_cl, iris_1, main = "DBScan")
> plot(Dbscan_cl, iris_1, main = "Petal Width vs Sepal Length")
```



Exp no. :18 **VISUALIZATIONS**
Date : 13-04-23

AIM :-

To illustrate the concept of visualization in R.

VISUALIZATIONS:

Data visualization is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

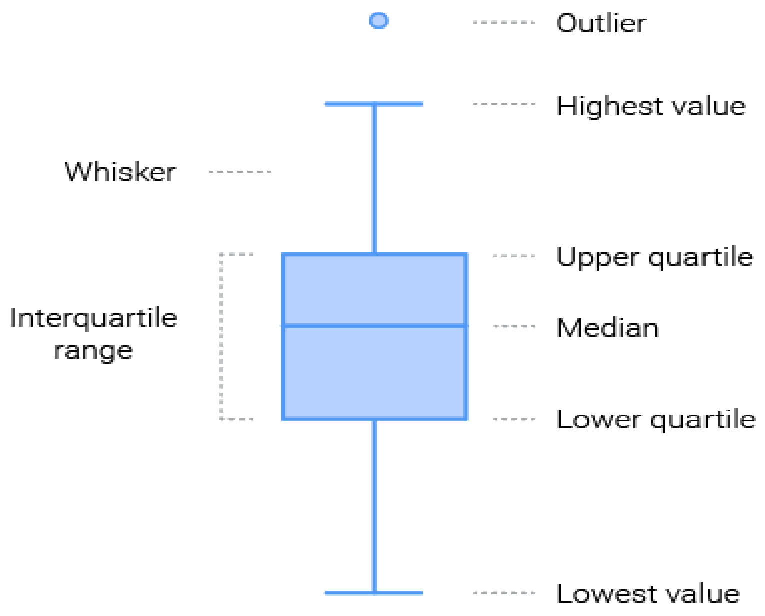
1) **Pie Chart :** A Pie Chart is a special chart that shows relative sizes of data using pie slices.

2) **Bar plot :** A bar plot represents data in rectangular bars with length of the bar proportional to the value of the variable.

3) **Scatter plot :** Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

4) **Histogram:** A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges.

5) **Box Plot :** The box-whisker plot (or a boxplot) is a quick and easy way to visualize complex data where you have multiple samples.



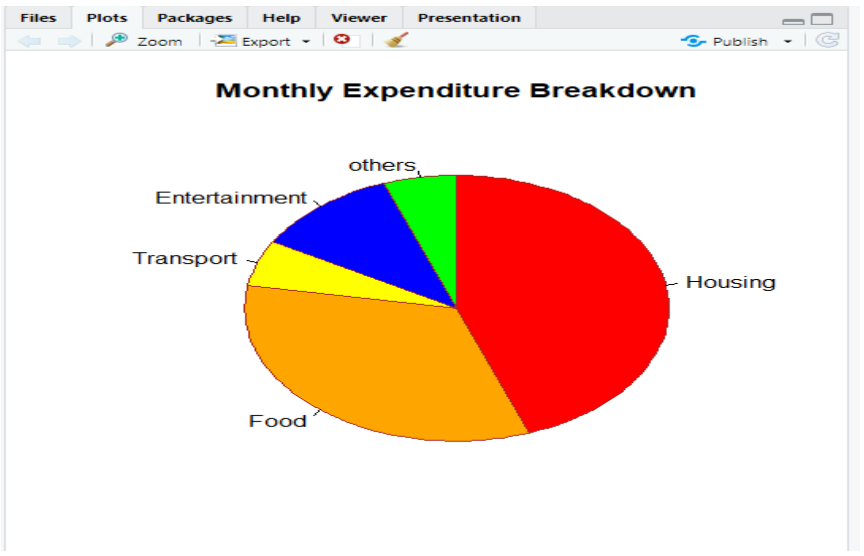
.

Q1) Create a pie chart which shows the monthly expenditure ?

Sol :

```
expenditure<-c("Housing"=8000, "Food"=6000, "Transport"=1000,
"Entertainment"=2000, "others" = 1000)
#Draw Pie Chart
pie(expenditure, main="Monthly Expenditure Breakdown",
col=c("red","orange","yellow","blue","green"),
border="brown",
clockwise=TRUE
)
```

OUTPUT:

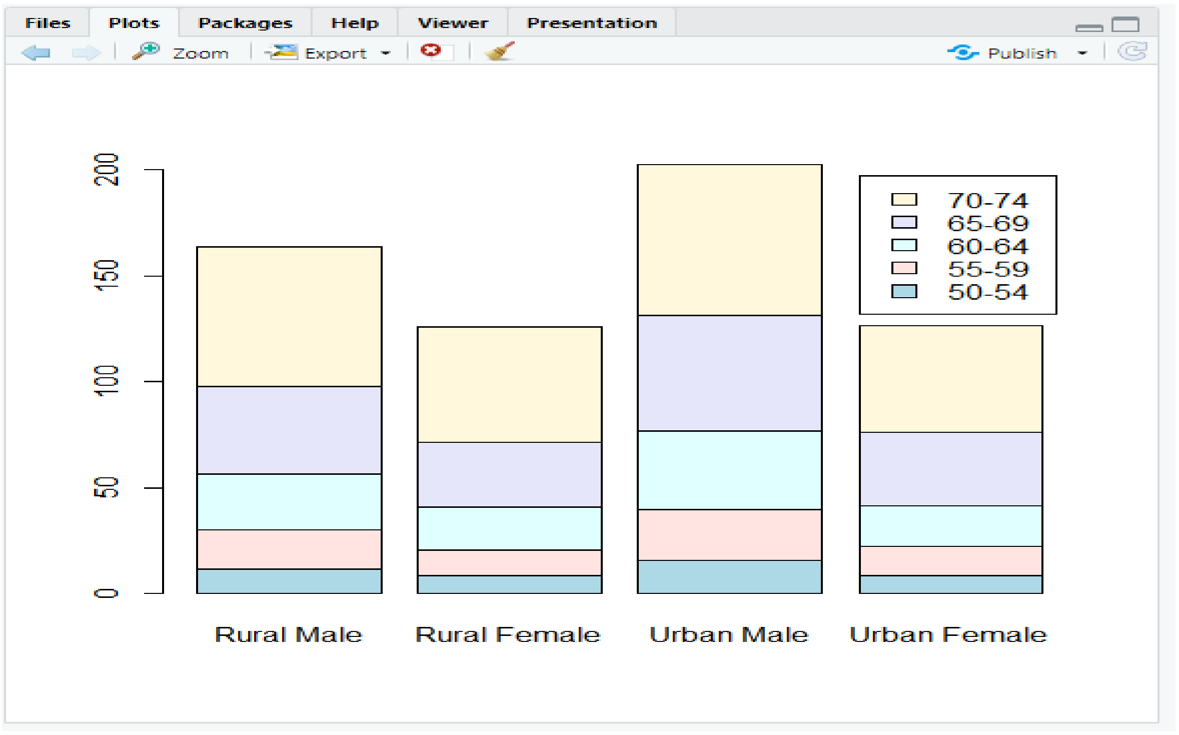


Q2) Create a Bar Plot which will plot the data present in VADeaths dataset.

Sol :

```
barplot(VADeaths, col=c("lightblue", "mistyrose", "lightcyan", "lavender",
"cornsilk"), legend=rownames(VADeaths))
```

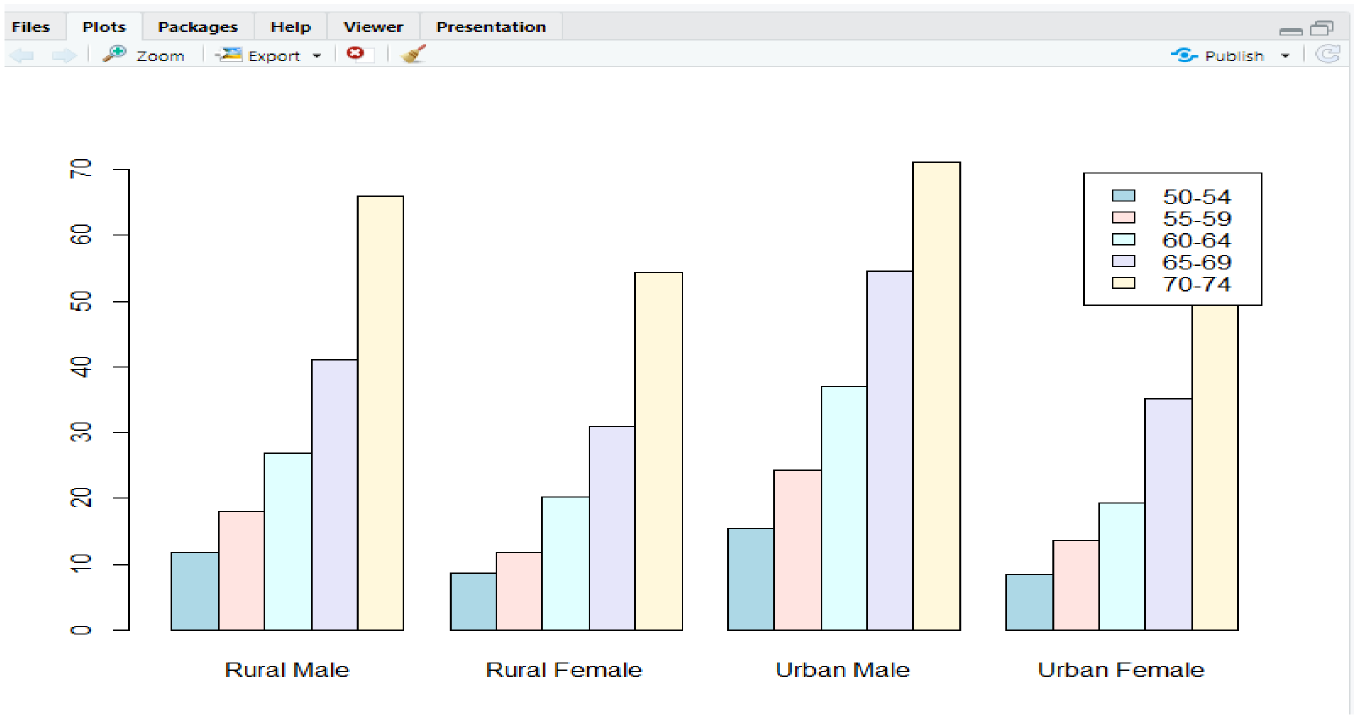
OUTPUT:



Q3) Create a Bar Plot which will plot the data present in VADeaths dataset.

Sol :
barplot(VADeaths, col=c("lightblue", "mistyrose", "lightcyan", "lavender", "cornsilk"), legend=rownames(VADeaths), beside=TRUE)

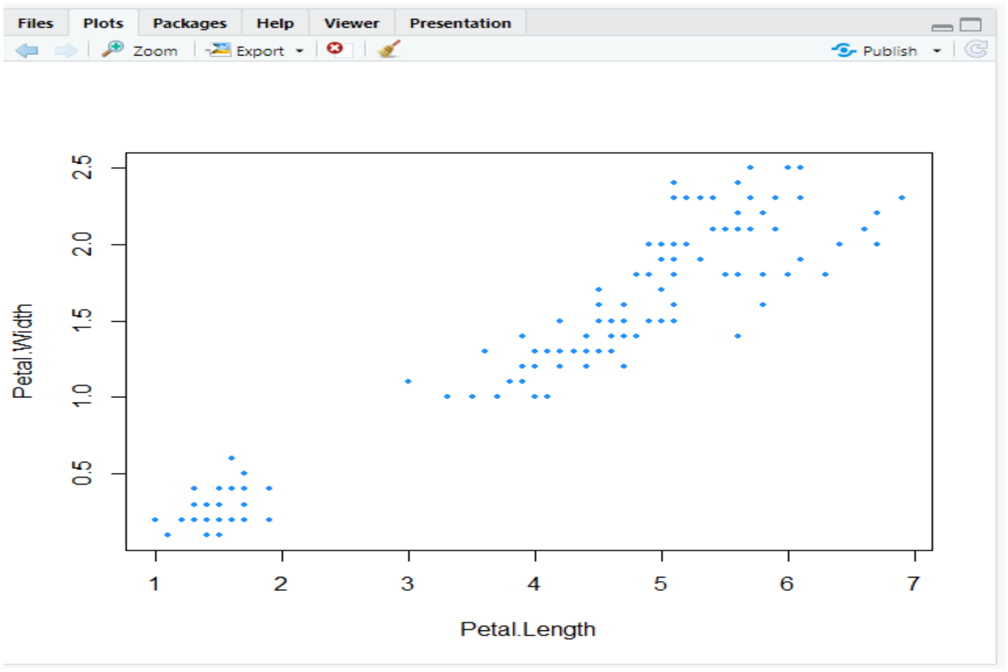
Output:



Q4) Create a Scatter Plot which will plot the data present in iris dataset.
Sol :

```
plot(Petal.Width ~ Petal.Length, data=iris,
     pch=20,
     cex=0.8,
col="dodgerblue1")
```

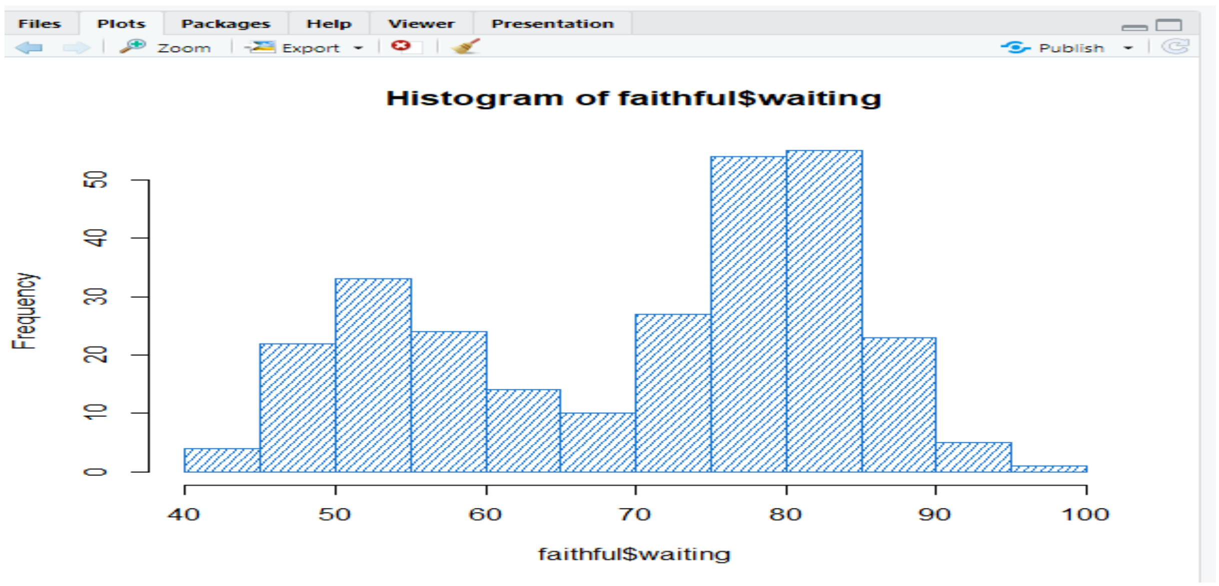
OUTPUT:



Q5) Create a histogram which will plot the data present in faithful dataset.
Sol :

```
hist(faithful$waiting,
     col="dodgerblue3",
     density=25,
     angle=60)
```

OUTPUT:

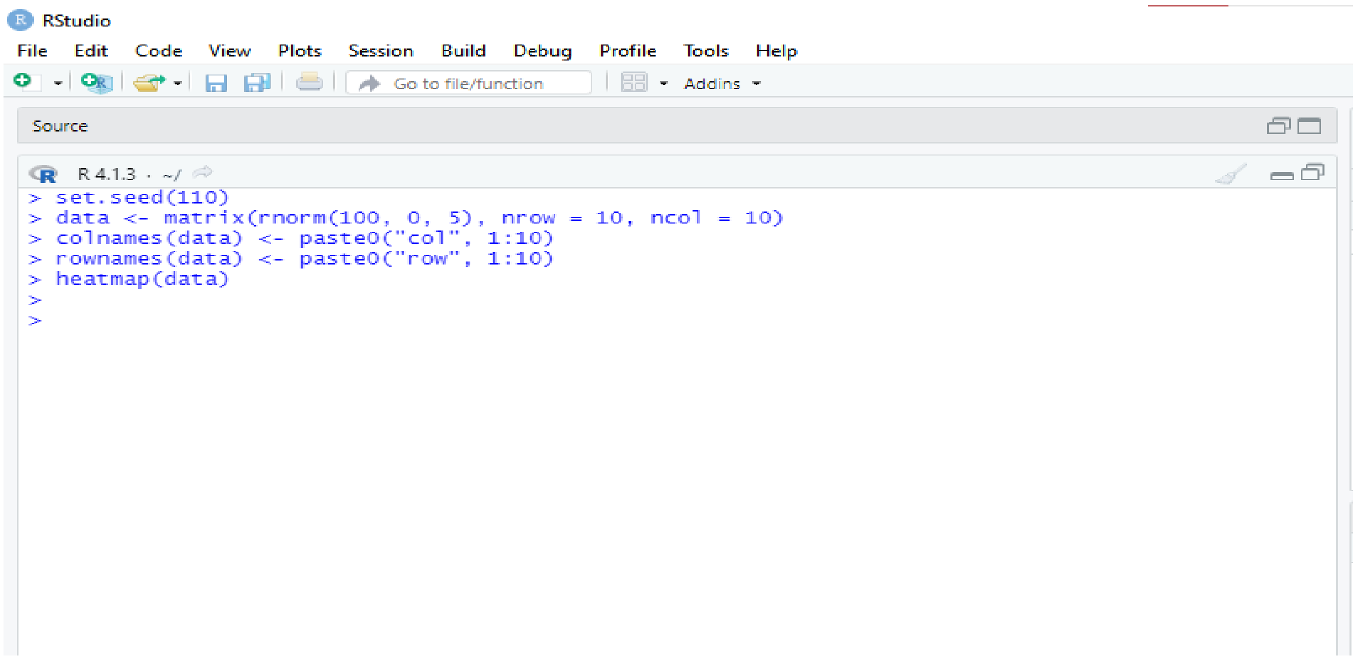


Q6) Create a heatmap in r programming language?

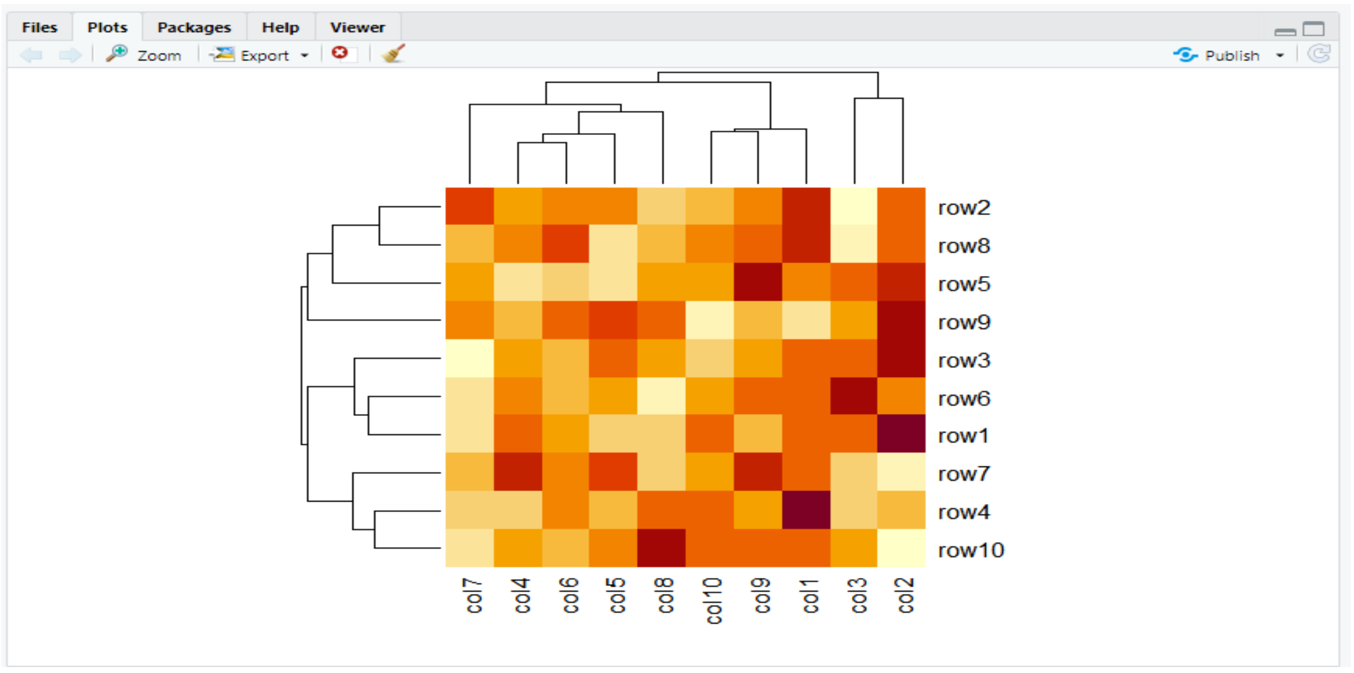
Sol :

```
 set.seed(110)
 data <- matrix(rnorm(100, 0, 5), nrow = 10, ncol = 10)
colnames(data) <- paste0("col", 1:10)
rownames(data) <- paste0("row", 1:10)
heatmap(data)
```

OUTPUT :


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
R 4.1.3 · ~/

```
> set.seed(110)
> data <- matrix(rnorm(100, 0, 5), nrow = 10, ncol = 10)
> colnames(data) <- paste0("col", 1:10)
> rownames(data) <- paste0("row", 1:10)
> heatmap(data)
>
>
```

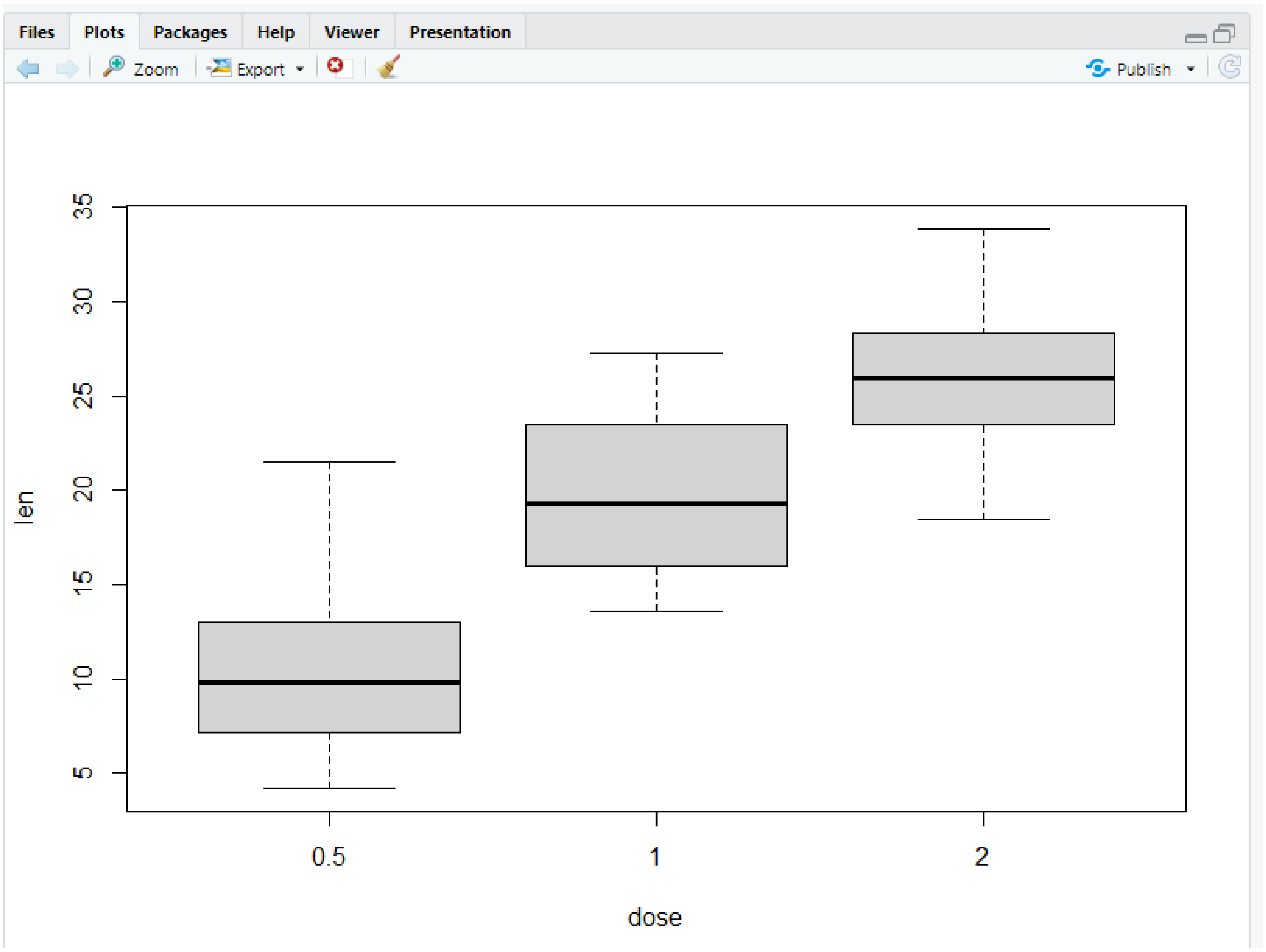


Q7) Create a box plot using R programming language with ToothGrowth dataset?

Sol:

```
boxplot(len ~ dose, data = ToothGrowth)
```

OUTPUT :


Files
Plots
Packages
Help
Viewer
Presentation
Zoom
Export
Publish
len
dose