SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR CAMPUS, MODINAGAR

(FACULTY OF SCIENCE AND HUMANITIES)

DEPARTMENT OF COMPUTER APPLICATIONS

PRACTICAL FILE

DATA ANALYSIS USING R

(PCA20S02J)

MCA-1st Year, 2nd Semester

Session: Dec 2023 - May 2024

Submitted to:	**Submitted by:**
Dr. Lalit Kishore Arora	**Kirti Garg**
(Assisstant Professor)	**RA2332241030047**

SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR CAMPUS, MODINAGAR

(FACULTY OF SCIENCE AND HUMANITIES) DEPARTMENT OF COMPUTER APPLICATIONS

Registration No. :- RA2332241030047

BONAFIDE CERTIFICATE

Certified to be the bonafide record of the work done by **KIRTI GARG** of MCA-First year, Second Semester (Section A) for the award of **Master's** degree course in DEPARTMENT OF COMPUTER APPLICATIONS in the FACULTY OF SCIENCE & HUMANITIES in DATA ANALYSIS USING R **[PCA20S02J]** laboratory during the Academic year- 2023-24.

SUBJECT -INCHARGE **HEAD OF THE DEPARTMENT**

Submitted for the university examination held on ___________________

INTERNAL EXAMINER 1 **INTERNAL EXAMINER 2**

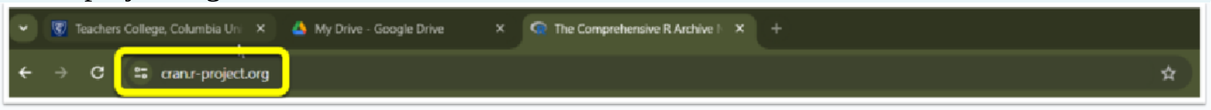
INDEX

PRACTICAL:1

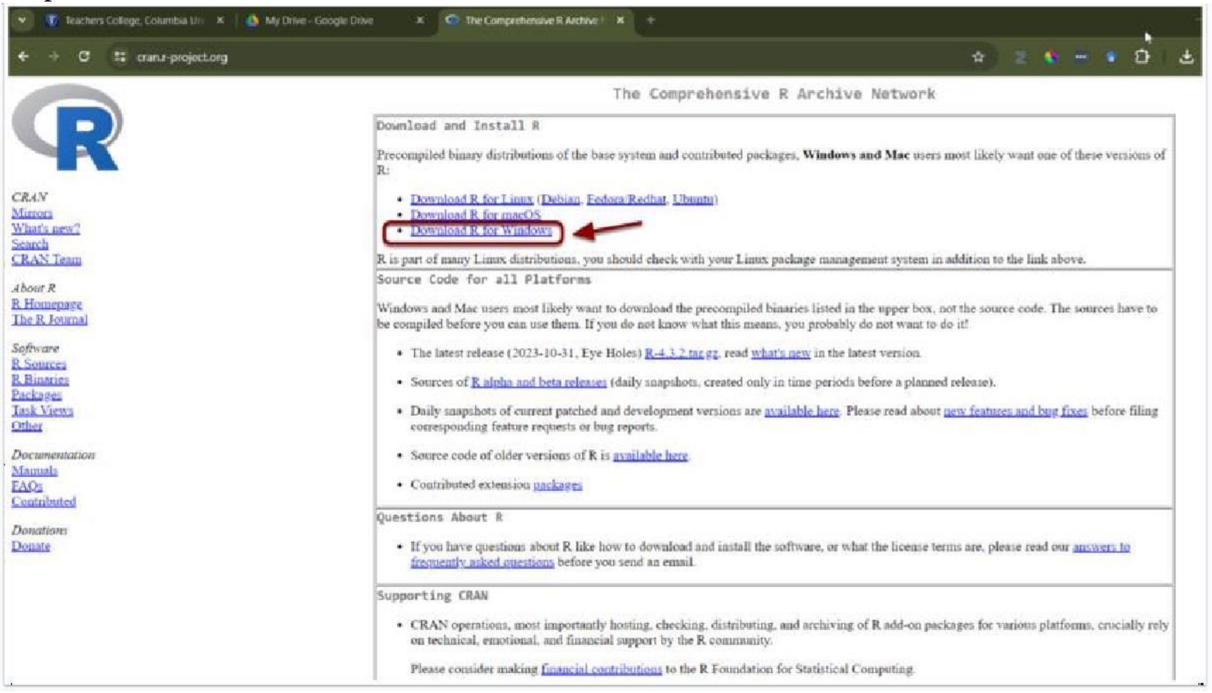
AIM: Write steps of installation of R and R Studio.

Procedure:

Step 1. To install R, go to cran.r-project.org
cran.r-project.org



Step 2. Click Download R for Windows.



The Comprehensive R Archive Network

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 (2023-10-31, Eye Holes) R-4.3.2.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

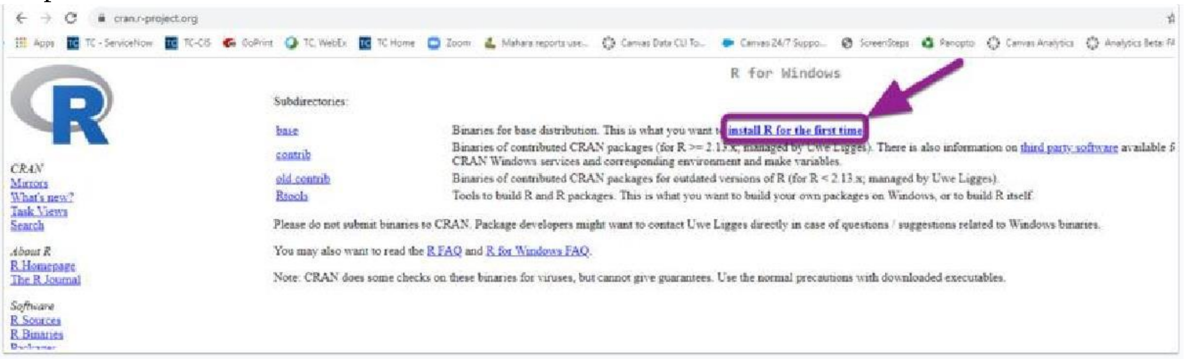
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

Supporting CRAN

- CRAN operations, most importantly hosting, checking, distributing, and archiving of R add-on packages for various platforms, crucially rely on technical, emotional, and financial support by the R community.

  Please consider making financial contributions to the R Foundation for Statistical Computing.

Step 3. Install R Click on install R for the first time.



R for Windows

Subdirectories:

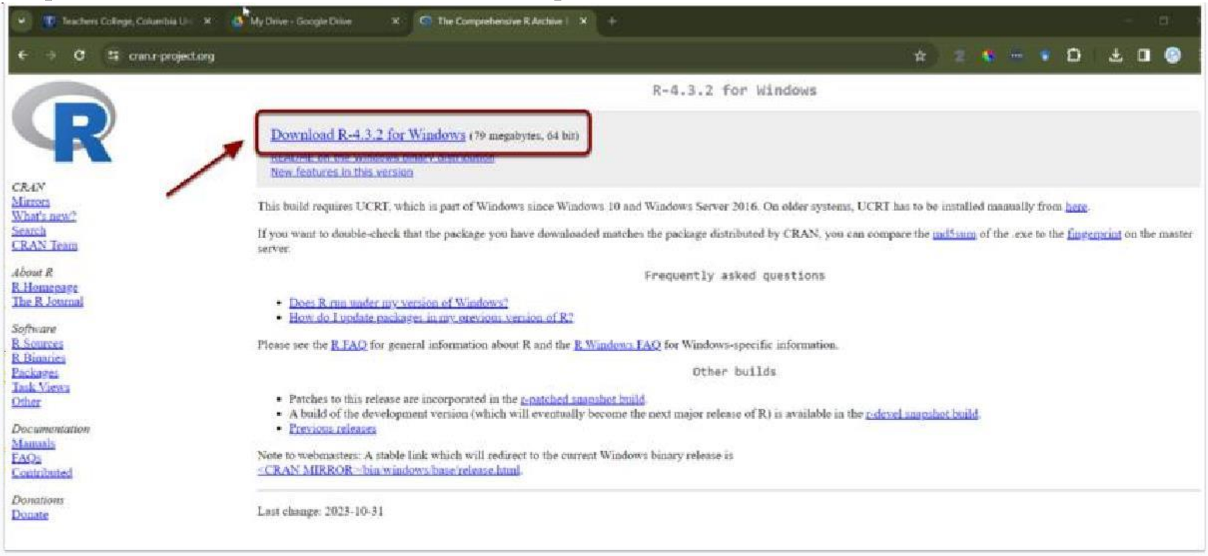
base	Binaries for base distribution. This is what you want to install R for the first time.
contrib	Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available f CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Step 4. Click Download R for Windows. Open the downloaded file.



R-4.3.2 for Windows

Download R-4.3.2 for Windows (79 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.

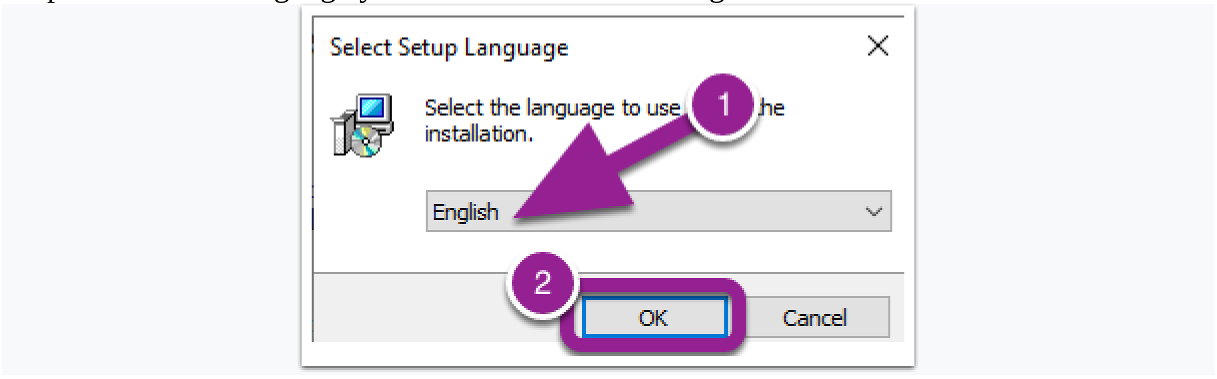
Other builds

- 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.

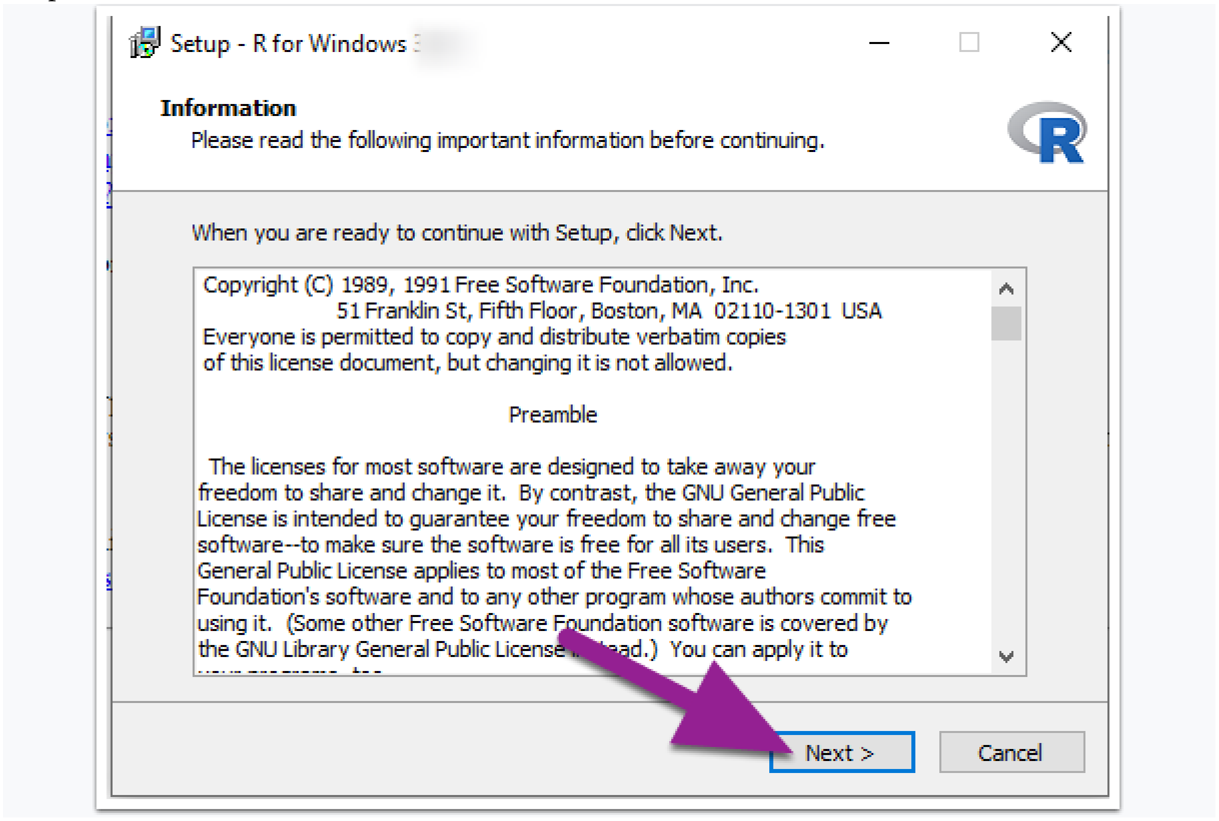
Last change: 2023-10-31

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


Select Setup Language

Select the language to use during the installation.

Step 6. Click Next.



Information

Please read the following important information before continuing.

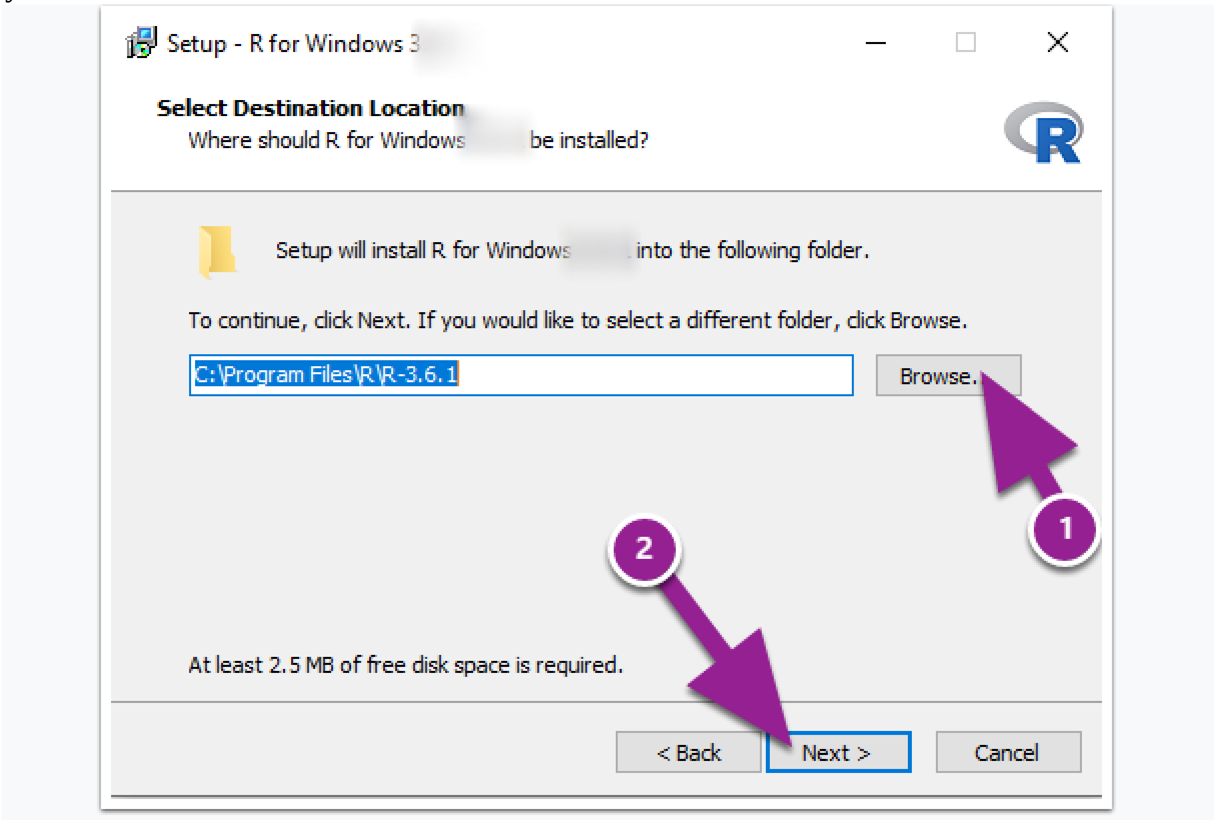
When you are ready to continue with Setup, click Next.

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users. This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it. (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.) You can apply it to

Step 7. Select where you would like R to be installed. It will default to your Program Files on your C Drive. Click Next.



Select Destination Location

Where should R for Windows be installed?

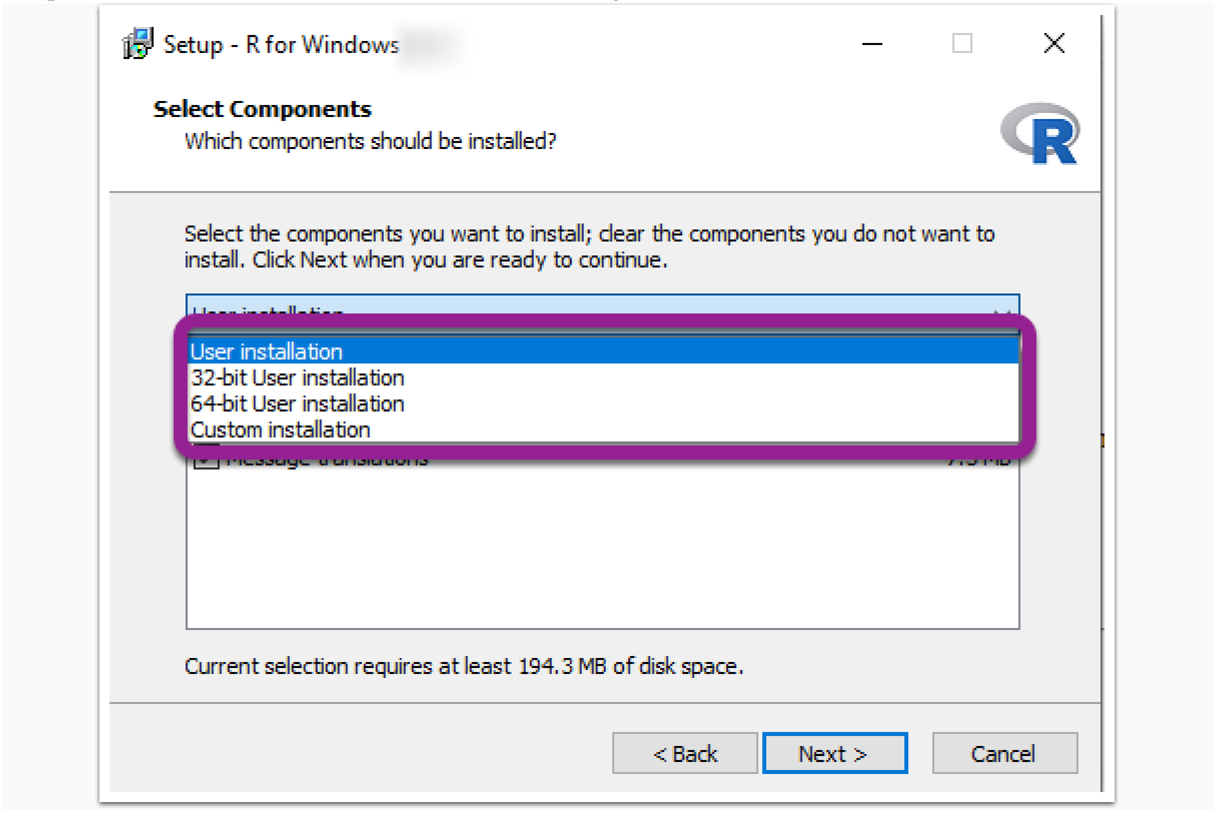
Setup will install R for Windows into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\R\R-3.6.1

At least 2.5 MB of free disk space is required.

Step 8. You can then choose which installation you would like.


Setup - R for Windows

Select Components

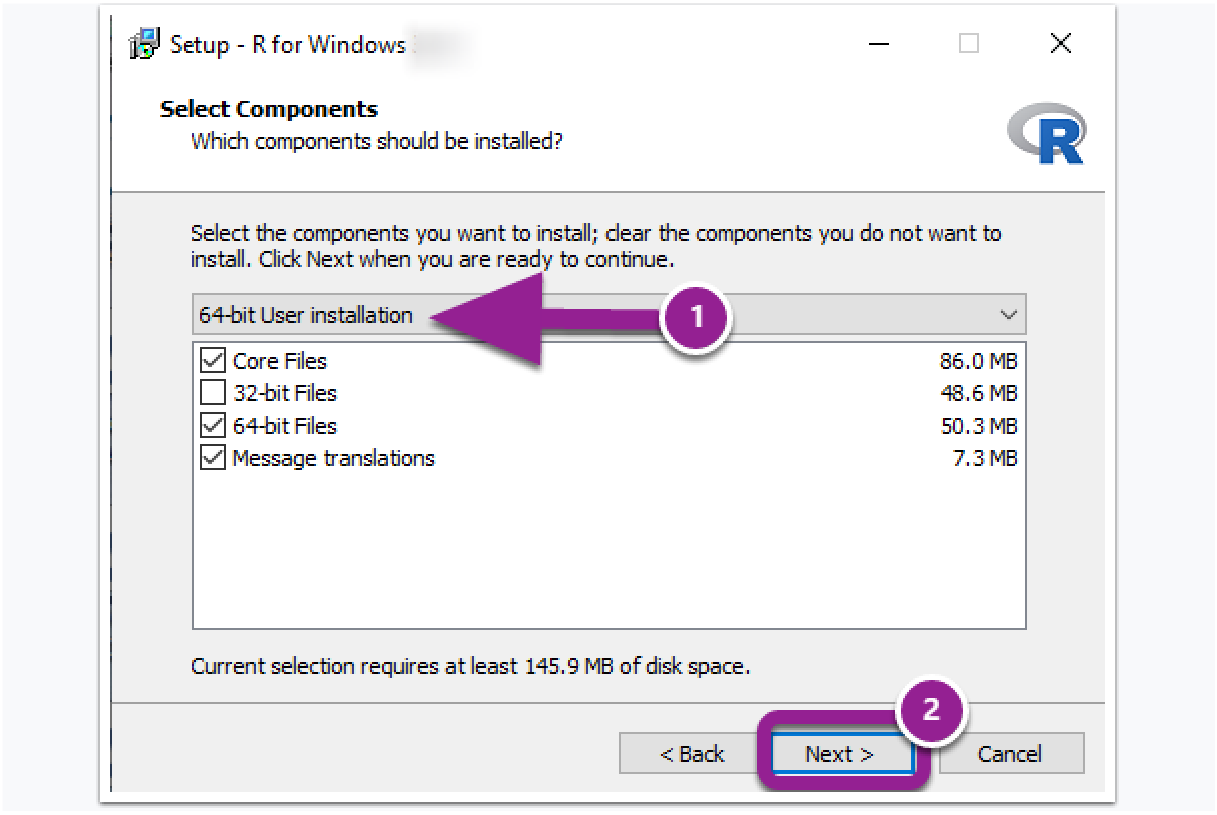
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

User installation
32-bit User installation
64-bit User installation
Custom installation

Current selection requires at least 194.3 MB of disk space.

Step 9. (Optional) If your computer is a 64-bit, you can choose the 64-bit User Installation. Then click Next.


Setup - R for Windows

Select Components

Which components should be installed?

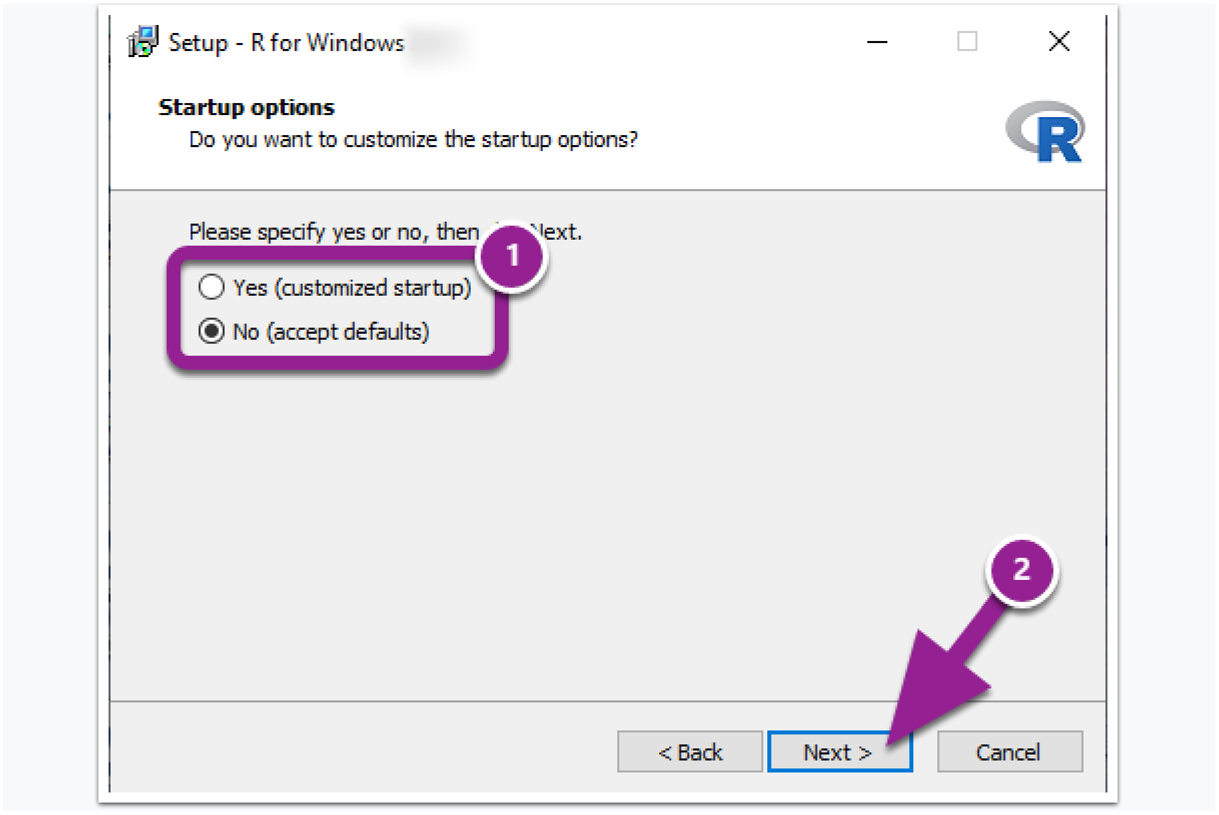
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

64-bit User installation

✓ Core Files	86.0 MB
☐ 32-bit Files	48.6 MB
✓ 64-bit Files	50.3 MB
✓ Message translations	7.3 MB

Current selection requires at least 145.9 MB of disk space.

Step 10. Then specify if you want to customized your startup or just use the defaults. Then click Next.


Setup - R for Windows

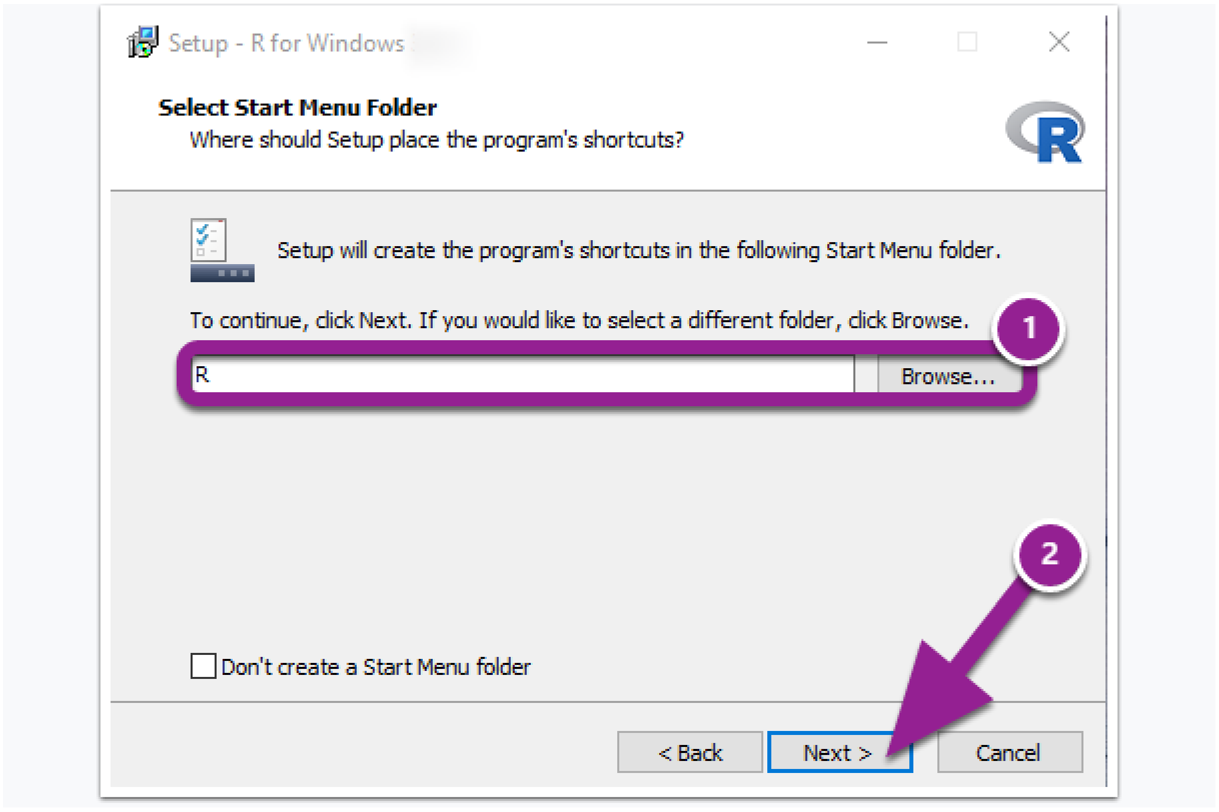
Startup options

Do you want to customize the startup options?

Please specify yes or no, then click Next.

Yes (customized startup) [✓ No (accept defaults)]

Step 11. 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.
You can also choose if you do not want a Start Menu folder at the bottom.


Setup - R for Windows

Select Start Menu Folder

Where should Setup place the program's shortcuts?

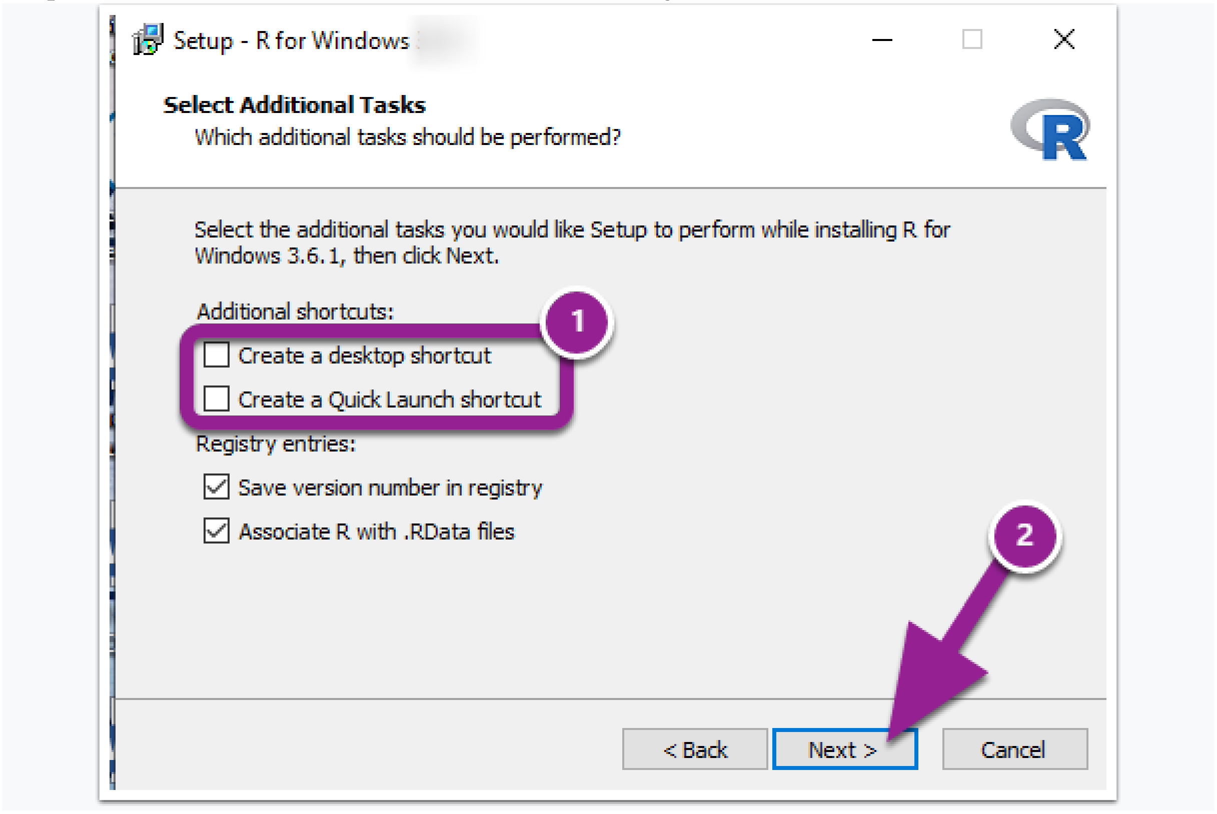
Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

R

☐ Don't create a Start Menu folder

Step 12. You can then select additional shortcuts if you would like. Click Next.



Setup - R for Windows

Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing R for Windows 3.6.1, then click Next.

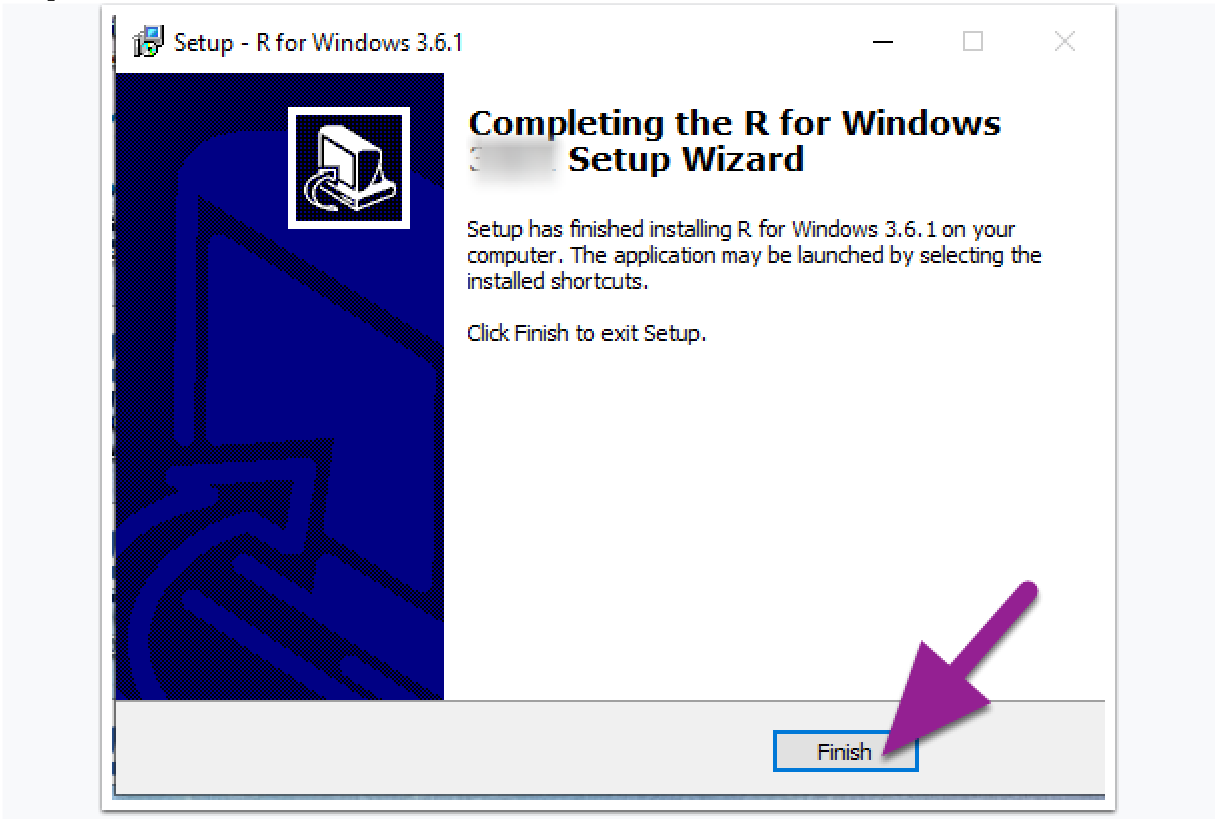
Additional shortcuts:

- [ ] Create a desktop shortcut
- [ ] Create a Quick Launch shortcut

Registry entries:

- [x] Save version number in registry
- [x] Associate R with .RData files

Step 13. Click Finish.



Setup - R for Windows 3.6.1

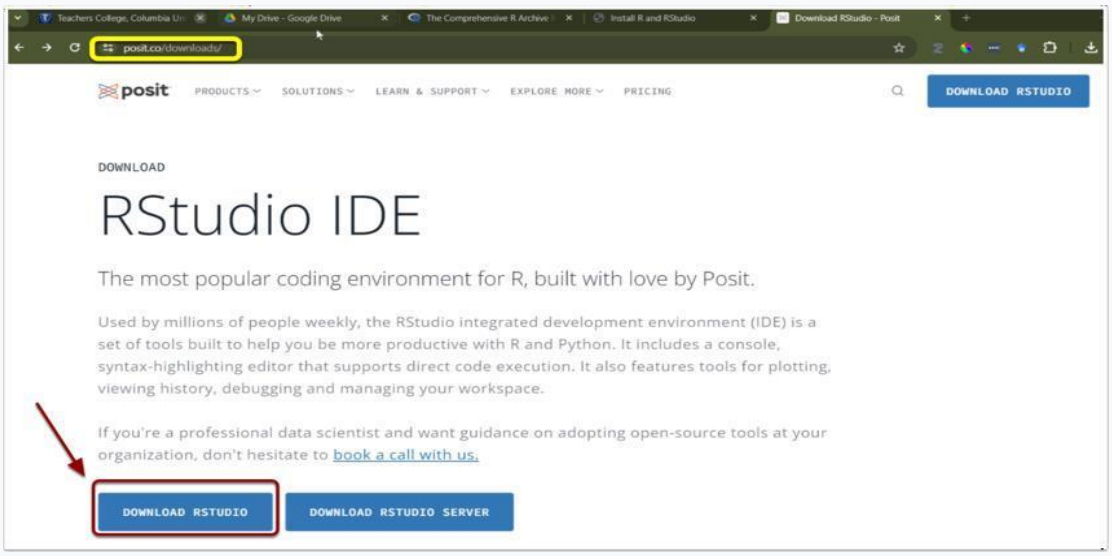
Completing the R for Windows Setup Wizard

Setup has finished installing R for Windows 3.6.1 on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

STEPS OF INSTALLATION OF R STUDIO

Step1:. Next, download RStudio. Go to https://posit.co/downloads/
https://posit.co/downloads/



DOWNLOAD

RStudio IDE

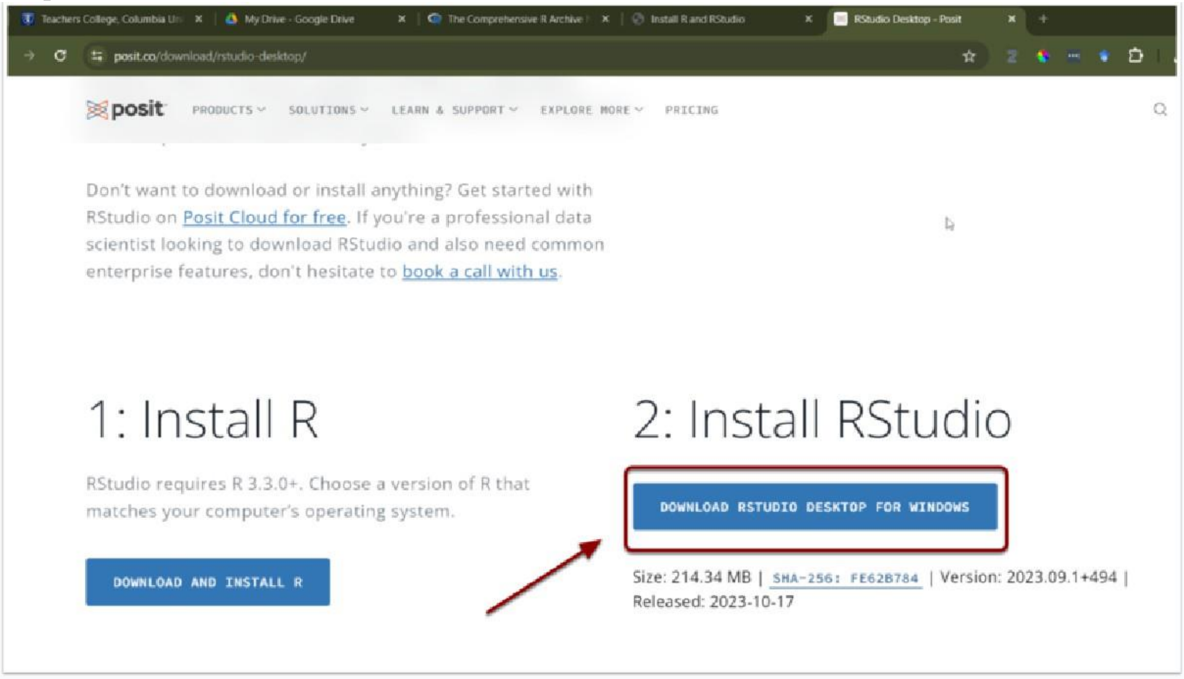
The most popular coding environment for R, built with love by Posit.

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 console, syntax-highlighting editor that supports direct code execution. It also features tools for plotting, viewing history, debugging and managing your workspace.

If you're a professional data scientist and want guidance on adopting open-source tools at your organization, don't hesitate to book a call with us.

DOWNLOAD RSTUDIO   DOWNLOAD RSTUDIO SERVER

Step2. Click Download RStudio.



Don't want to download or install anything? Get started with RStudio on Posit Cloud for free. If you're a professional data scientist looking to download RStudio and also need common enterprise features, don't hesitate to book a call with us.

1: Install R

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

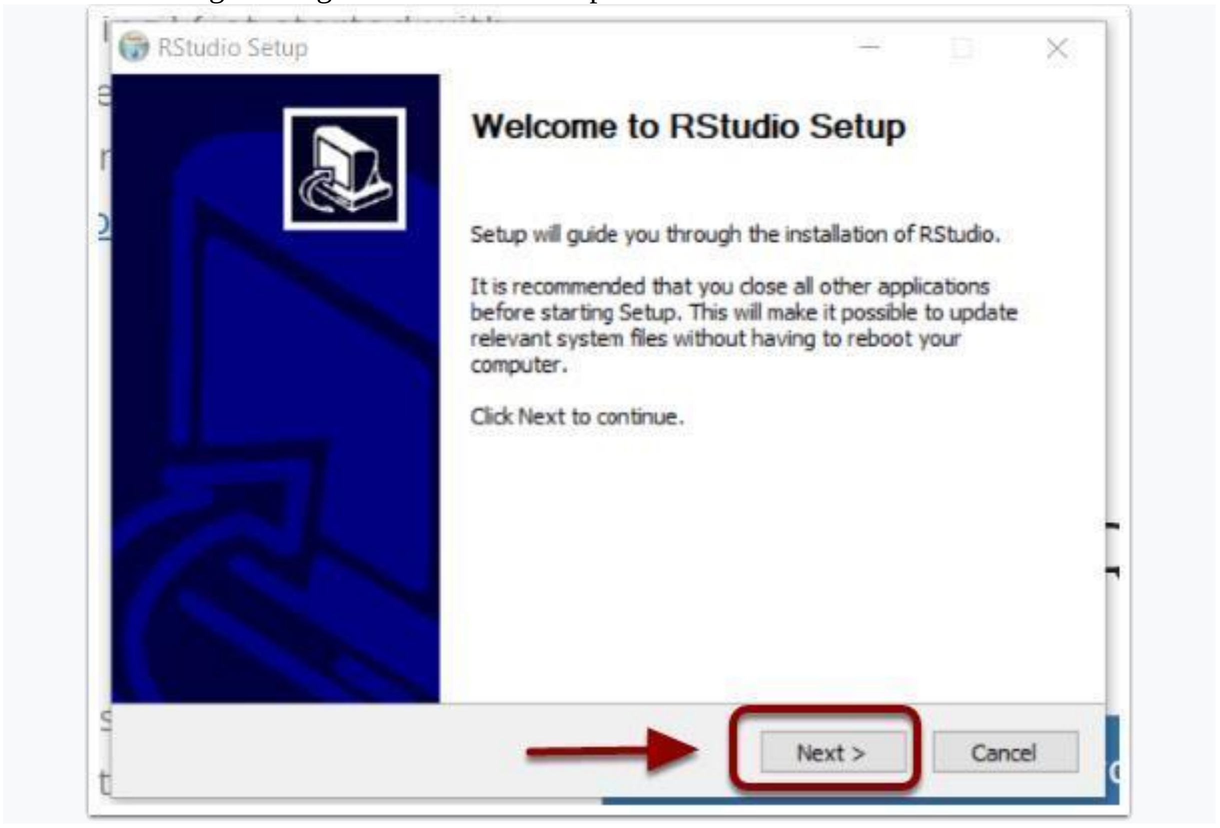
DOWNLOAD AND INSTALL R

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

Size: 214.34 MB | SHA-256: FE62B784 | Version: 2023.09.1+494 | Released: 2023-10-17

Step3. Once the packet has downloaded, the Welcome to RStudio Setup Wizard will open. Click Next and go through the installation steps.



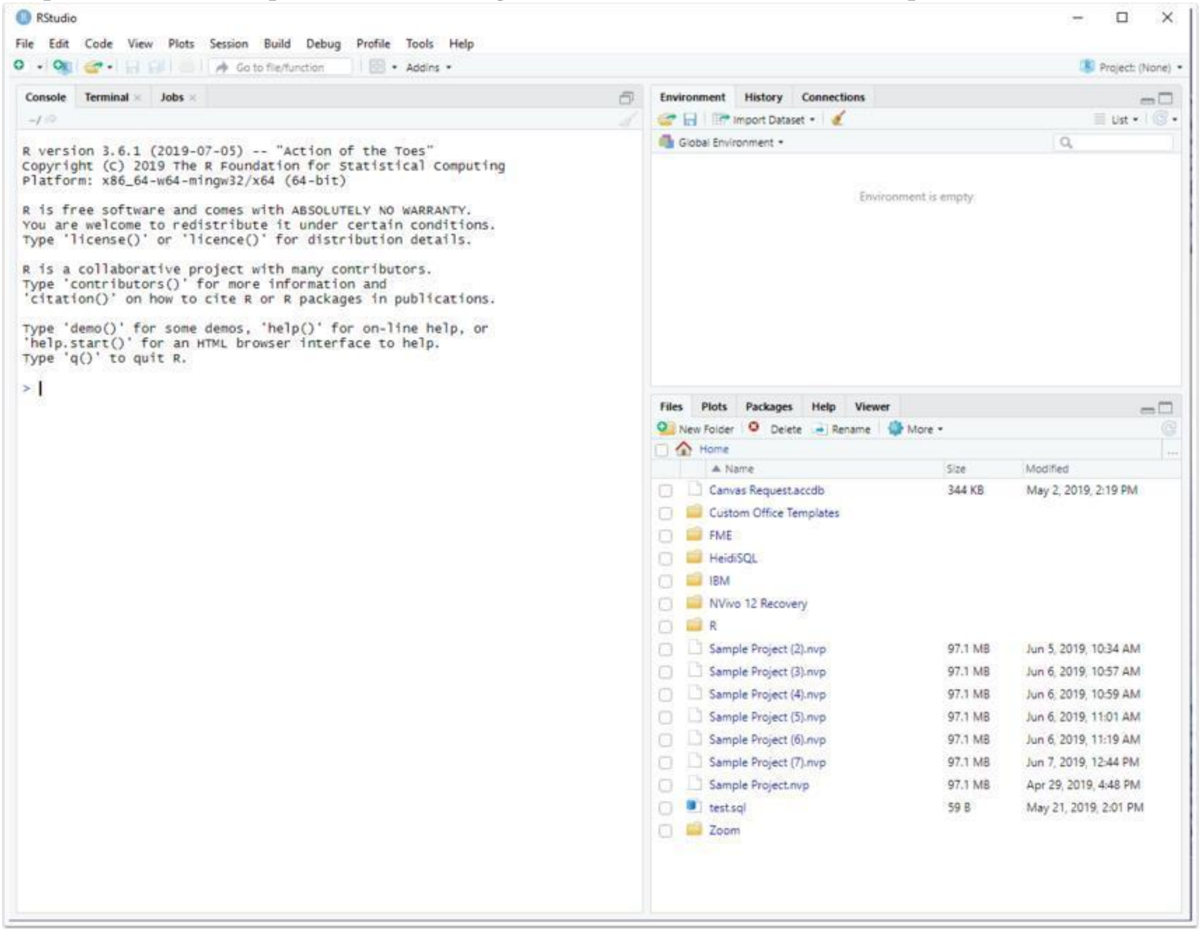
Welcome to RStudio Setup

Setup will guide you through the installation of RStudio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Step4. After the Setup Wizard finishing the installation, RStudio will open.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console Terminal Jobs
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home

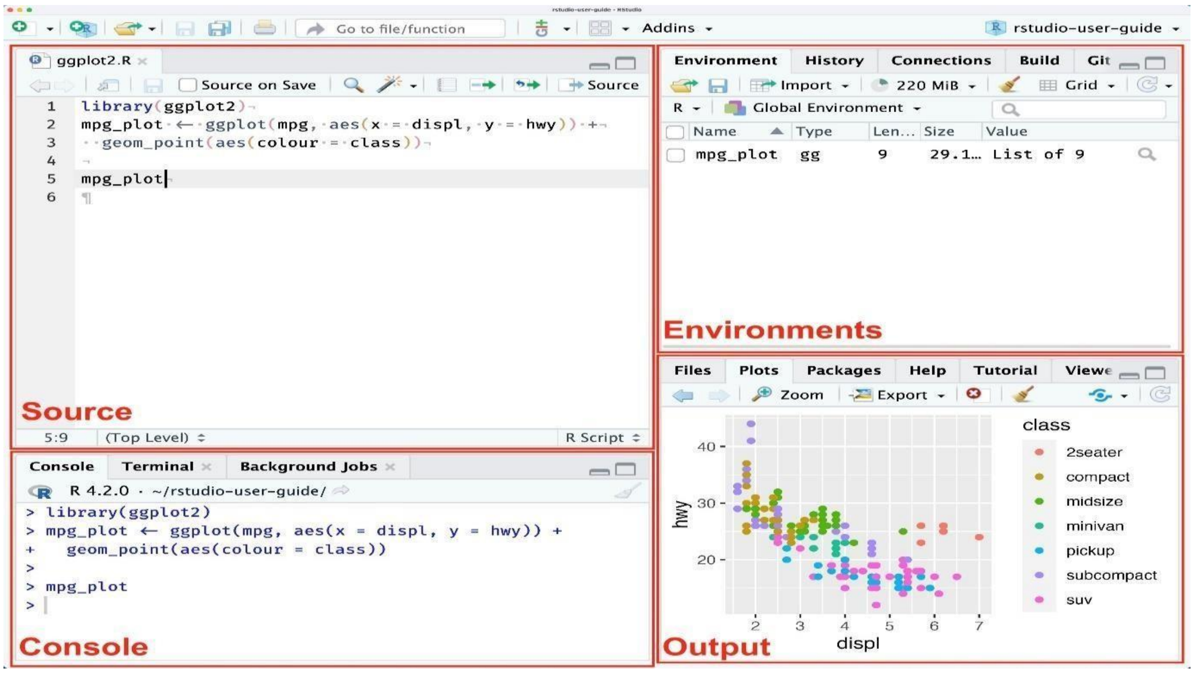
```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Name	Size	Modified
Canvas Request.accdb	344 KB	May 2, 2019, 2:19 PM
Custom Office Templates		
FME		
HeidiSQL		
IBM		
NVivo 12 Recovery		
R		
Sample Project (2).nvp	97.1 MB	Jun 5, 2019, 10:34 AM
Sample Project (3).nvp	97.1 MB	Jun 6, 2019, 10:57 AM
Sample Project (4).nvp	97.1 MB	Jun 6, 2019, 10:59 AM
Sample Project (5).nvp	97.1 MB	Jun 6, 2019, 11:01 AM
Sample Project (6).nvp	97.1 MB	Jun 6, 2019, 11:19 AM
Sample Project (7).nvp	97.1 MB	Jun 7, 2019, 12:44 PM
Sample Project.nvp	97.1 MB	Apr 29, 2019, 4:48 PM
test.sql	59 B	May 21, 2019, 2:01 PM
Zoom		

Pane Layout
The RStudio user interface has 4 primary panes:

- **Source pane**
- **Console pane**
- **Environment pane**, containing the **Environment**, **History**, **Connections**, **Build**, **VCS** , and **Tutorial** tabs
- **Output pane**, containing the **Files**, **Plots**, **Packages**, **Help**, **Viewer**, and **Presentation** tabs

Each pane can be minimized or maximized within the column by clicking the minimize/maximize buttons .



```
library(ggplot2)
mpg_plot <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = class))

mpg_plot
```

Name	Type	Len...	Size	Value
mpg_plot	gg	9	29.1…	List of 9

```
R 4.2.0 · ~/rstudio-user-guide/
> library(ggplot2)
> mpg_plot <- ggplot(mpg, aes(x = displ, y = hwy)) +
+   geom_point(aes(colour = class))
>
> mpg_plot
>
```

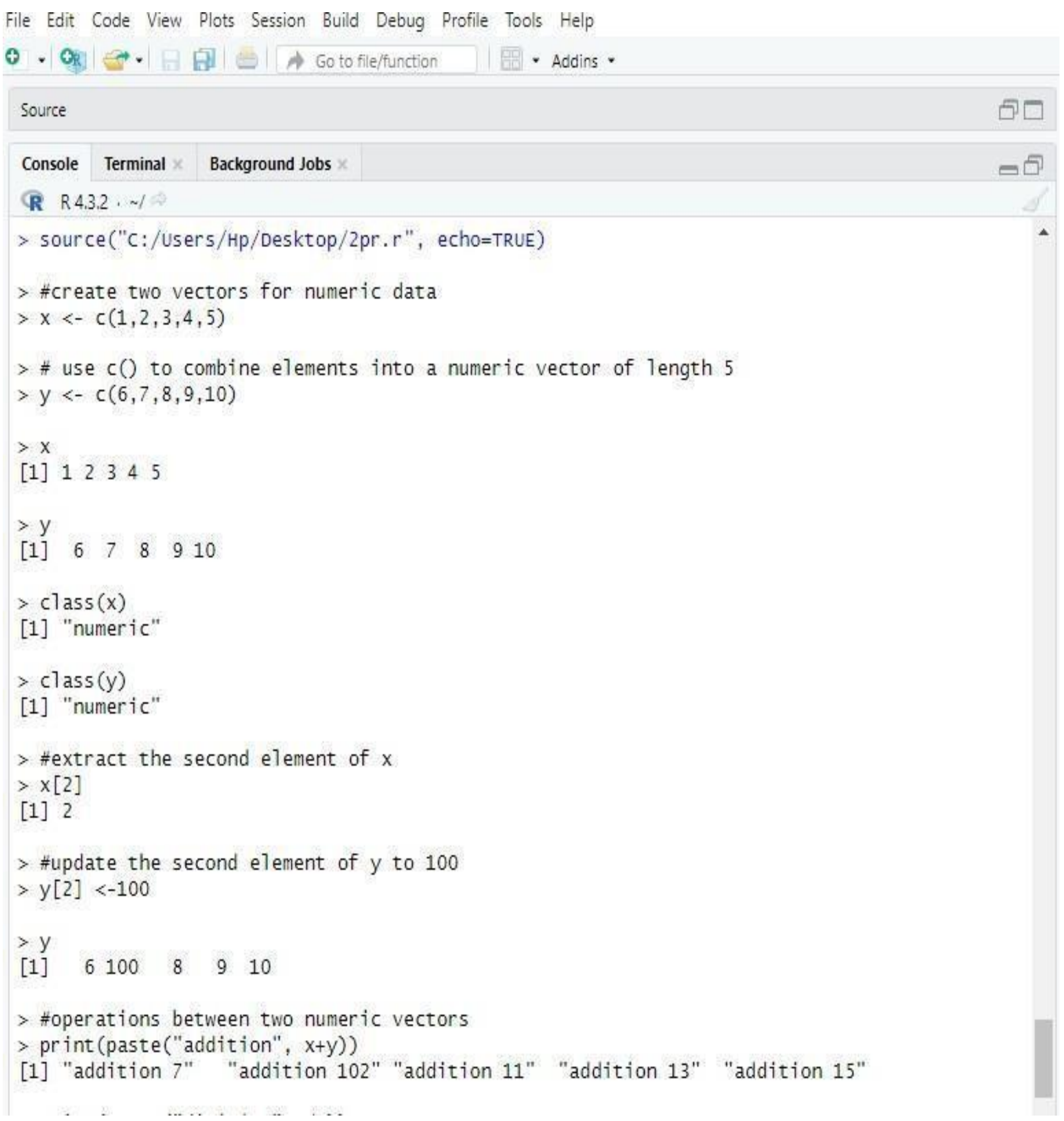
PRACTICAL-2

AIM: Create two vectors in R for Numeric Data.

PROCEDURE:

```
#create two vectors for numeric data
x <- c(1,2,3,4,5)
# use c() to combine elements into a numeric vector of length 5
y <- c(6,7,8,9,10)
x
y
class(x)
class(y)
#extract the second element of x
x[2]
#update the second element of y to 100
y[2] <-100
y
#operations between two numeric vectors
print(paste("addition", x+y))
print(paste("division",y/5))
print(paste("subtraction",y-x)) #create two vectors for numeric data
x <- c(1,2,3,4,5)
# use c() to combine elements into a numeric vector of length 5
y <- c(6,7,8,9,10)
x
y
class(x)
class(y)
#extract the second element of x
x[2]
#update the second element of y to 100
y[2] <-100
y
#operations between two numeric vectors
print(paste("addition", x+y))
```

INPUT/OUTPUT:



```
> source("C:/Users/Hp/Desktop/2pr.r", echo=TRUE)

> #create two vectors for numeric data
> x <- c(1,2,3,4,5)

> # use c() to combine elements into a numeric vector of length 5
> y <- c(6,7,8,9,10)

> x
[1] 1 2 3 4 5

> y
[1]  6  7  8  9 10

> class(x)
[1] "numeric"

> class(y)
[1] "numeric"

> #extract the second element of x
> x[2]
[1] 2

> #update the second element of y to 100
> y[2] <-100

> y
[1]   6 100   8   9  10

> #operations between two numeric vectors
> print(paste("addition", x+y))
[1] "addition 7"   "addition 102" "addition 11"  "addition 13"  "addition 15"
```

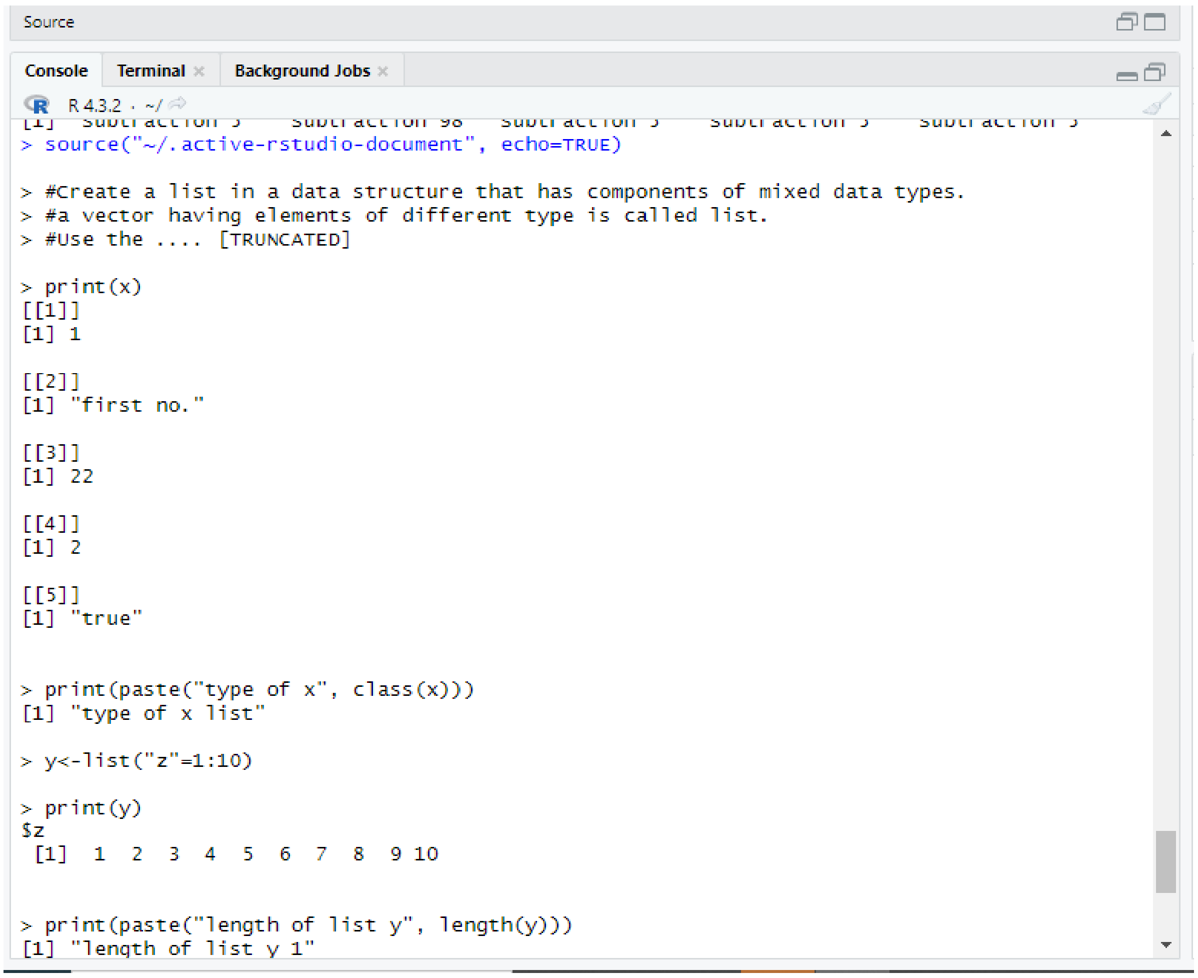
PRACTICAL-3

AIM: Create a list in a data structure that has components of mixed data types.

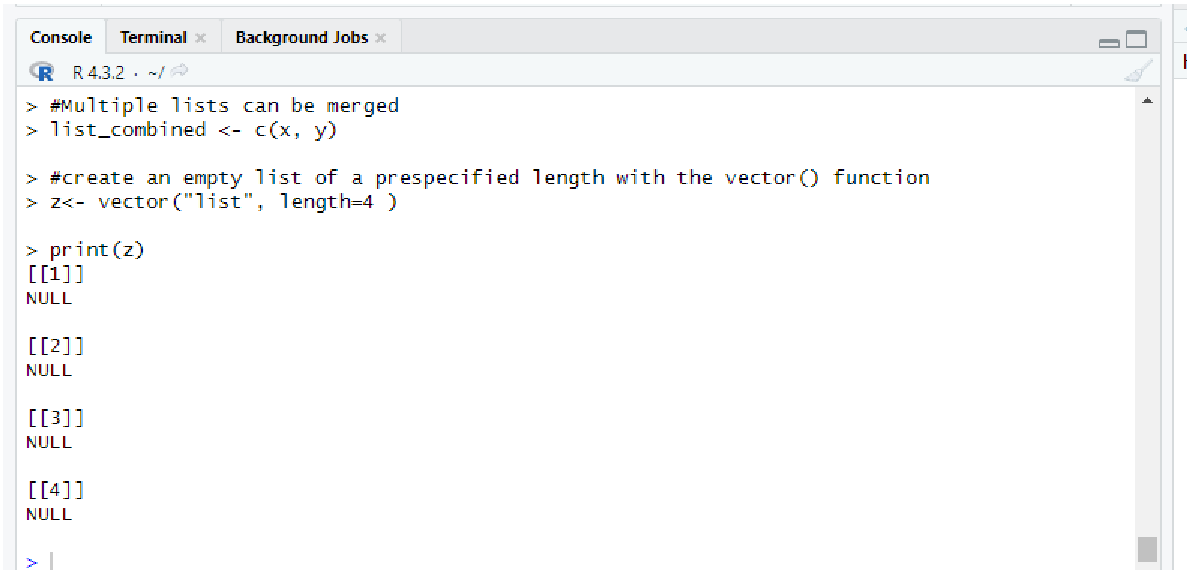
PROCEDURE:

```
#a vector having elements of different type is called list.

#Use the list() function to create a list.


x <- list(1,"first no.",22, 2, "true")

print(x)

print(paste("type of x", class(x)))


y<-list("z"=1:10)

print(y)

print(paste("length of list y", length(y)))


#Multiple lists can be merged

list_combined <- c(x, y)


#create an empty list of a prespecified length with the vector() function

z<- vector("list", length=4 )

print(z)
```

INPUT/OUTPUT:



```
[1] subtraction 5    subtraction 98    subtraction 5    subtraction 5    subtraction 5
> source("~/.active-rstudio-document", echo=TRUE)

> #Create a list in a data structure that has components of mixed data types.
> #a vector having elements of different type is called list.
> #Use the .... [TRUNCATED]

> print(x)
[[1]]
[1] 1

[[2]]
[1] "first no."

[[3]]
[1] 22

[[4]]
[1] 2

[[5]]
[1] "true"


> print(paste("type of x", class(x)))
[1] "type of x list"

> y<-list("z"=1:10)

> print(y)
$z
 [1]  1  2  3  4  5  6  7  8  9 10


> print(paste("length of list y", length(y)))
[1] "length of list y 1"
```



```
> #Multiple lists can be merged
> list_combined <- c(x, y)

> #create an empty list of a prespecified length with the vector() function
> z<- vector("list", length=4 )

> print(z)
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

> |
```

PRACTICAL-4

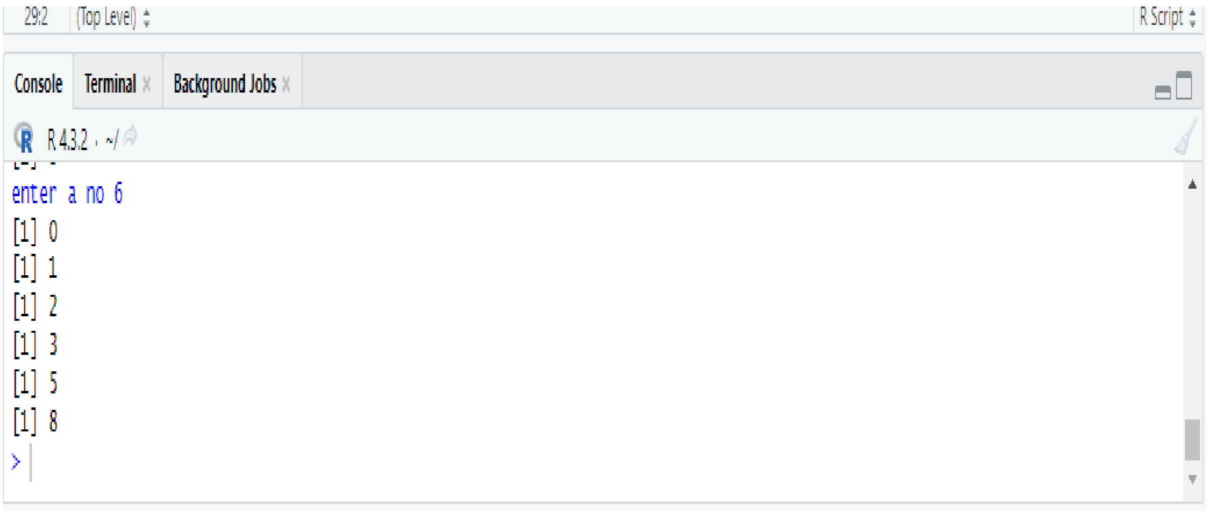
AIM: Create a code to display a Fibonacci

series

PROCEDURE:

```
x=readline("enter a no")
x=as.integer(x)
a=0
b=1
c=0
while(x>0){
  print(c)
  c=a+b
  a=b
  b=c
  x=x-1
}
```

OUTPUT



```
enter a no 6
[1] 0
[1] 1
[1] 2
[1] 3
[1] 5
[1] 8
>
```

PRACTICAL-5

AIM: Implement decision tree on credit card issue dataset(import from kaggle).

PROCEDURE:

Algorithm :

1. Import the dataset: Import the dataset which contains the input variables (or the independent variables) and the output variable (or the dependent variable).

2. Data Pre-processing: Check for missing values and handle them, handle outliers if any, and remove or transform variables that do not contribute to the model.

3. Split the dataset: Split the dataset into a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate the performance of the model.

4. Feature Scaling: If the input variables (or the independent variables) have different ranges, scale them to have the same range to ensure that no variable dominates the others in the modeling process.

5. Model Training: Train the decision tree model using the training set. The decision tree model recursively partitions the input space into subsets, each defined by a set of conditions on the input variables. The partitioning is done in a way that the subsets are as homogeneous as possible with respect to the output variable.

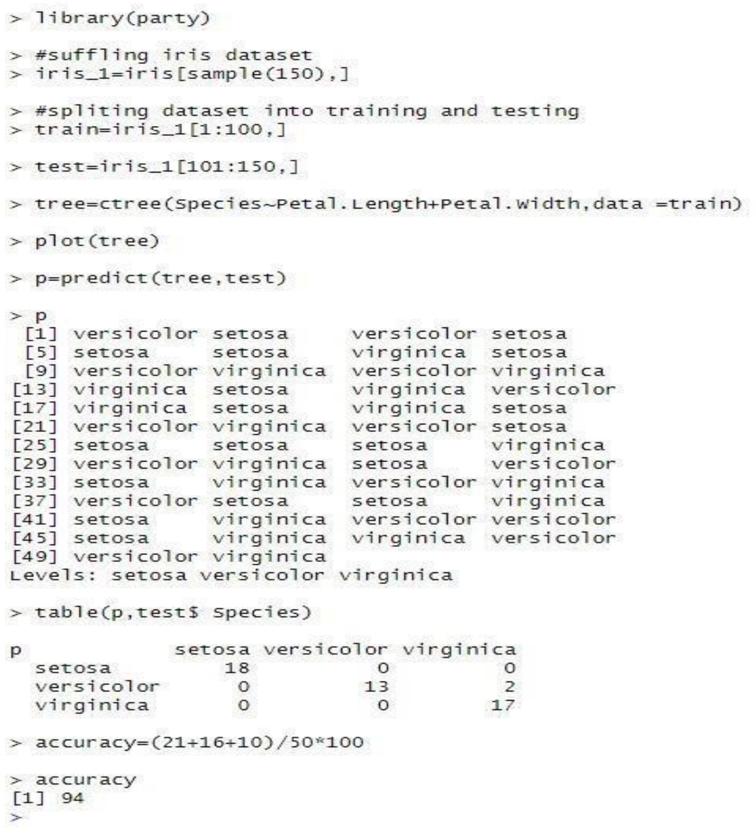
6. Model Evaluation: Evaluate the performance of the model using the testing set. There are several metrics to evaluate the performance of the model, such as accuracy, precision, recall, Fl score, and area under the ROC curve.

7. Model Improvement: If the model does not perform well, try to improve the model by changing the hyperparameters, selecting the relevant features, or using other modeling
techniques.

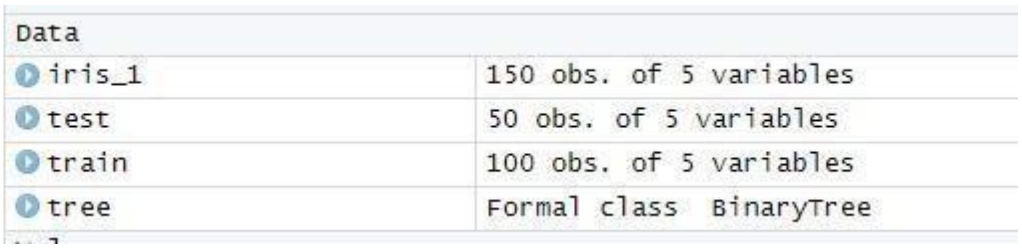
8. Model Deployment: Once the model is finalized and performs well on the testing set, deploy it to the production environment for making predictions on new data.

```
install.packages("party"')
install.packages("'Rtools")
library(party)
#suffling iris dataset
iris_1=iris[sample(150),]
#spliting dataset into training and testing
train=iris_1[1:100,]
test=iris_1[101:150,]
tree=ctree(Species~Petal.Length+Petal.Width,data =train)
plot(tree)
p=predict(tree,test)
p
table(p,test$ Species)
accuracy=(21+16+10)/50*100
accuracy
```

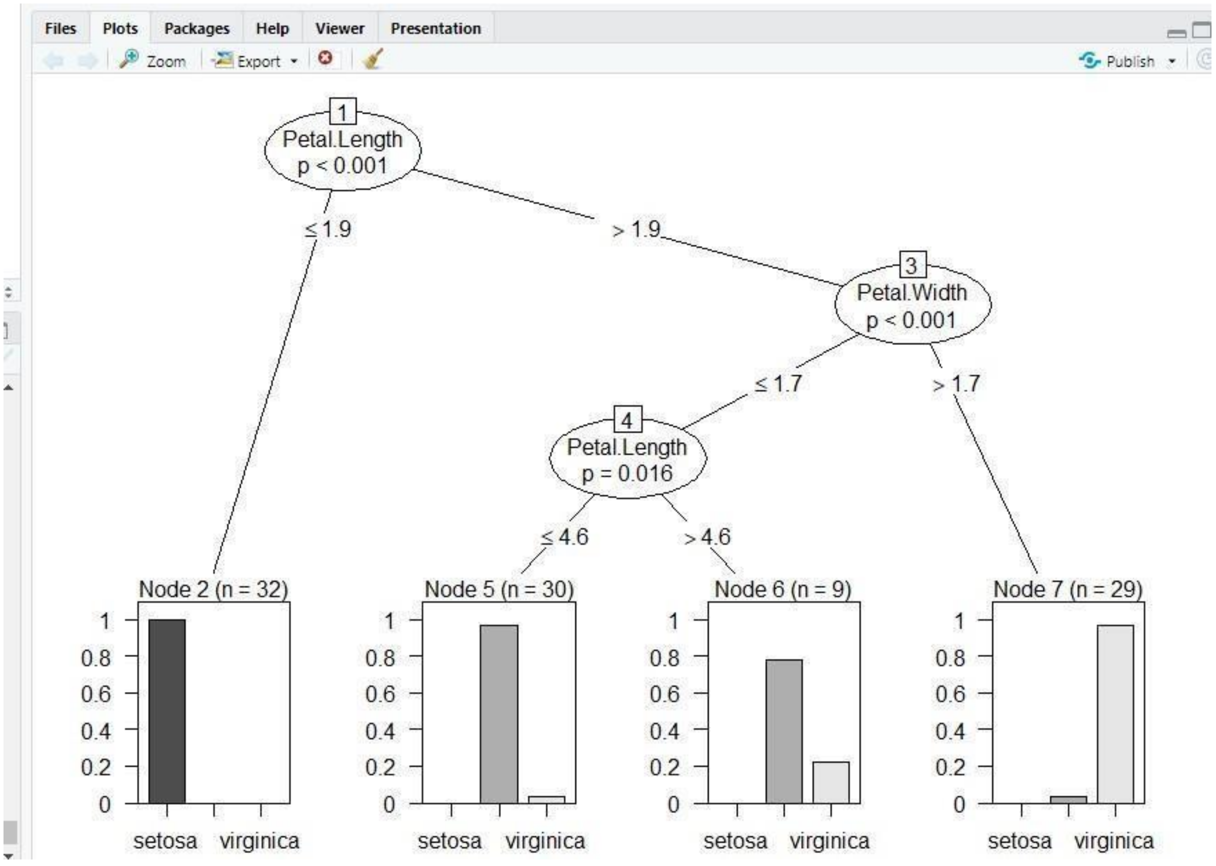
OUTPUT :



```
> library(party)

> #suffling iris dataset
> iris_1=iris[sample(150),]

> #spliting dataset into training and testing
> train=iris_1[1:100,]

> test=iris_1[101:150,]

> tree=ctree(Species~Petal.Length+Petal.Width,data =train)

> plot(tree)

> p=predict(tree,test)

> p
 [1] versicolor setosa     versicolor setosa
 [5] setosa     setosa     virginica  setosa
 [9] versicolor virginica  versicolor virginica
[13] virginica  setosa     virginica  versicolor
[17] virginica  setosa     virginica  setosa
[21] versicolor virginica  versicolor setosa
[25] setosa     setosa     setosa     virginica
[29] versicolor virginica  setosa     versicolor
[33] setosa     virginica  versicolor virginica
[37] versicolor setosa     setosa     virginica
[41] setosa     virginica  versicolor versicolor
[45] setosa     virginica  virginica  versicolor
[49] versicolor virginica
Levels: setosa versicolor virginica

> table(p,test$ Species)

p            setosa versicolor virginica
  setosa         18          0         0
  versicolor      0         13         2
  virginica       0          0        17

> accuracy=(21+16+10)/50*100

> accuracy
[1] 94
>
```



```
Data
iris_1          150 obs. of 5 variables
test            50 obs. of 5 variables
train           100 obs. of 5 variables
tree            Formal class  BinaryTree
```



RESULT:
The Decision tree have been implemented successfully.

PRACTICA-6

AIM:- Implement the KNN algorithm on the Brest cancer dataset.

PROCEDURE:-

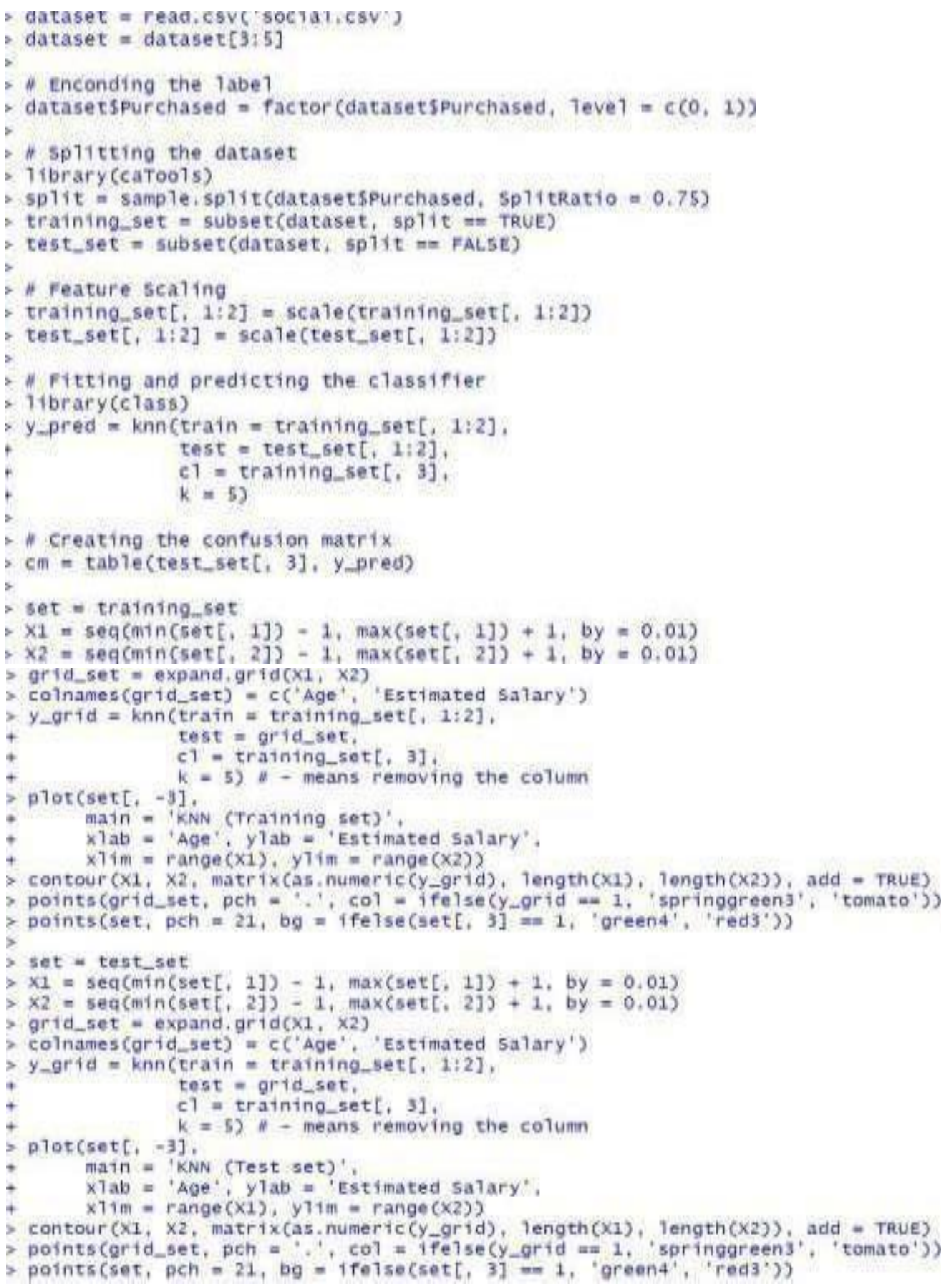
To implement the k-Nearest Neighbors (KNN) algorithm on the Breast Cancer dataset, you can use the caret and class packages in R. The Breast Cancer dataset is often available through the datasets package in R. Here's an example**.**

```
# Install and load necessary libraries
install.packages("caret") install.packages("class")
library(caret)
library(class)

# Load the Breast Cancer dataset data("BreastCancer")

# Explore the structure of the dataset str(BreastCancer)

# Split the data into training and testing sets set.seed(123) # Set
seed for reproducibility
sample_index <- createDataPartition(BreastCancer$Class, p = 0.8, list = FALSE) train_data <-
BreastCancer[sample_index, ] test_data <- BreastCancer[-sample_index, ]

# Preprocess the data
# In this example, we'll scale the features
preprocess_params <- preProcess(train_data[, -1], method = c("center", "scale"))

train_data_scaled <- predict(preprocess_params, train_data[, -1]) test_data_scaled <-
predict(preprocess_params, test_data[, -1])

# Train the KNN model
knn_model <- knn(train_data_scaled, test_data_scaled, train_data$Class, k = 5)

# Evaluate the model
conf_matrix <- table(knn_model, test_data$Class)
print("Confusion Matrix:")
print(conf_matrix)

accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix) print(paste("Accuracy:",
round(accuracy, 4)))
```

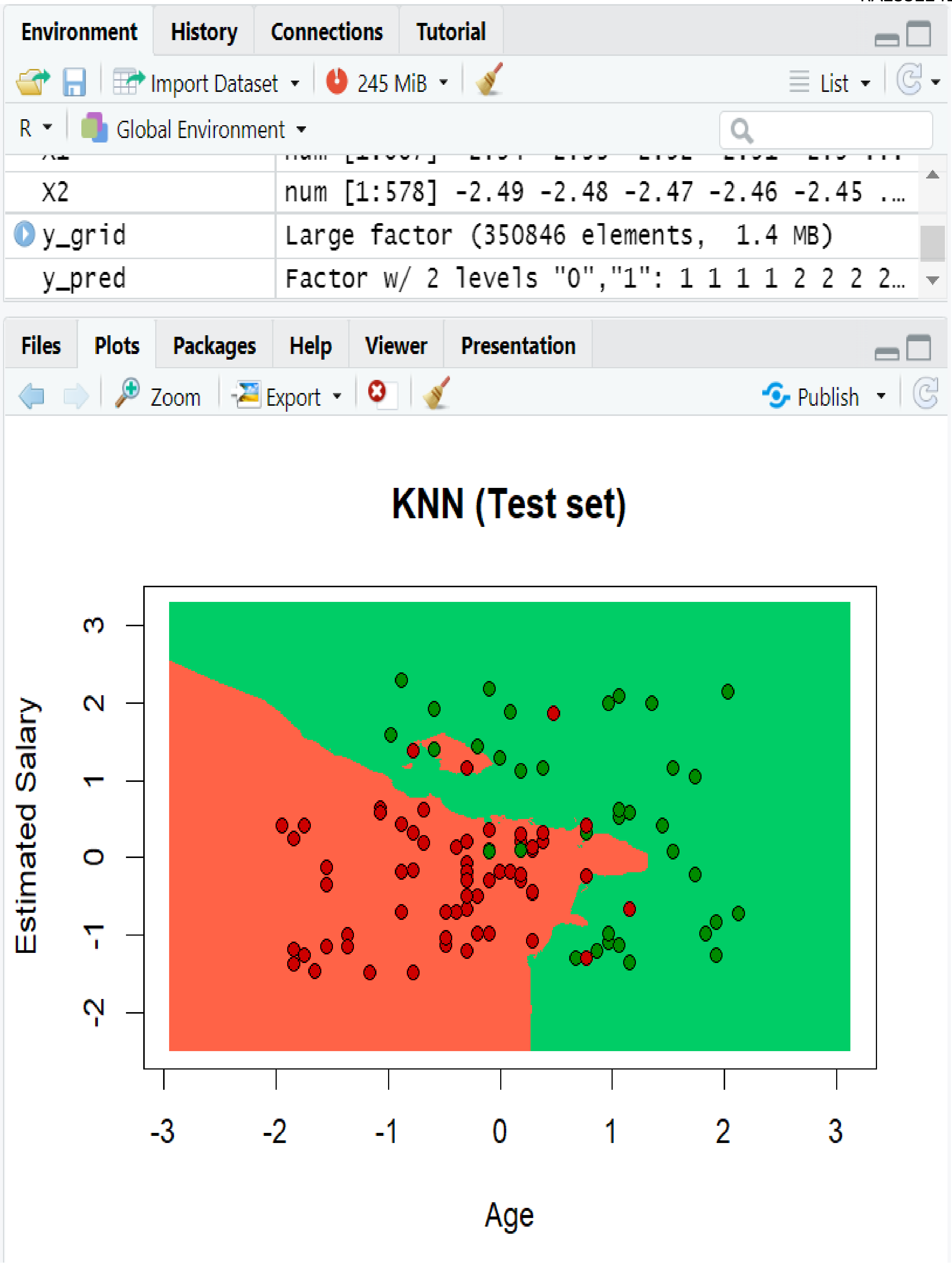
In this example, the Breast Cancer dataset is split into training and testing sets, and the features are scaled. The KNN model is then trained using the k n function from the class package, and the accuracy ofthe model is evaluated.

Remember to replace Breast Cancer $Class with the actual column name representing the target variablein your dataset. Additionally, consider experimenting with different values of k to find the optimal number of neighbors for your specific dataset.

OUTPUT:-



```
> dataset = read.csv('social.csv')
> dataset = dataset[3:5]
>
> # Enconding the label
> dataset$Purchased = factor(dataset$Purchased, level = c(0, 1))
>
> # Splitting the dataset
> library(caTools)
> split = sample.split(dataset$Purchased, SplitRatio = 0.75)
> training_set = subset(dataset, split == TRUE)
> test_set = subset(dataset, split == FALSE)
>
> # Feature Scaling
> training_set[, 1:2] = scale(training_set[, 1:2])
> test_set[, 1:2] = scale(test_set[, 1:2])
>
> # Fitting and predicting the classifier
> library(class)
> y_pred = knn(train = training_set[, 1:2],
+              test = test_set[, 1:2],
+              cl = training_set[, 3],
+              k = 5)
>
> # Creating the confusion matrix
> cm = table(test_set[, 3], y_pred)
>
> set = training_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
> grid_set = expand.grid(X1, X2)
> colnames(grid_set) = c('Age', 'Estimated Salary')
> y_grid = knn(train = training_set[, 1:2],
+              test = grid_set,
+              cl = training_set[, 3],
+              k = 5) # - means removing the column
> plot(set[, -3],
+      main = 'KNN (Training set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
>
> set = test_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
> grid_set = expand.grid(X1, X2)
> colnames(grid_set) = c('Age', 'Estimated Salary')
> y_grid = knn(train = training_set[, 1:2],
+              test = grid_set,
+              cl = training_set[, 3],
+              k = 5) # - means removing the column
> plot(set[, -3],
+      main = 'KNN (Test set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```



```
X2       num [1:578] -2.49 -2.48 -2.47 -2.46 -2.45 ...
y_grid   Large factor (350846 elements,  1.4 MB)
y_pred   Factor w/ 2 levels "0","1": 1 1 1 1 2 2 2 2...
```

RESULT:-

The code has been executed successfully.

PRACTICAL :7

AIM: Implement Naive Bayes algorithm.

PROCEDURE:

1. Convert the given dataset into frequency tables.

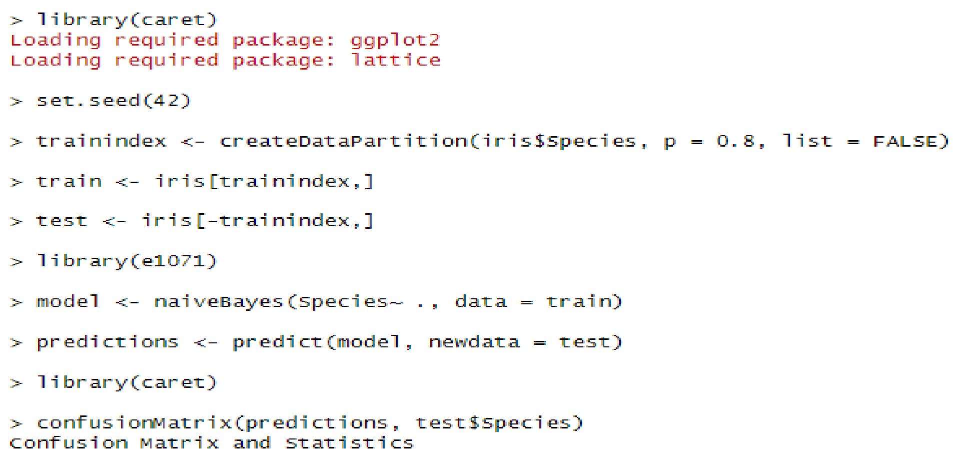
2. Generate Likelihood table by finding the probabilities of given features.

3. Now, use Bayes theorem to calculate the posterior probability.

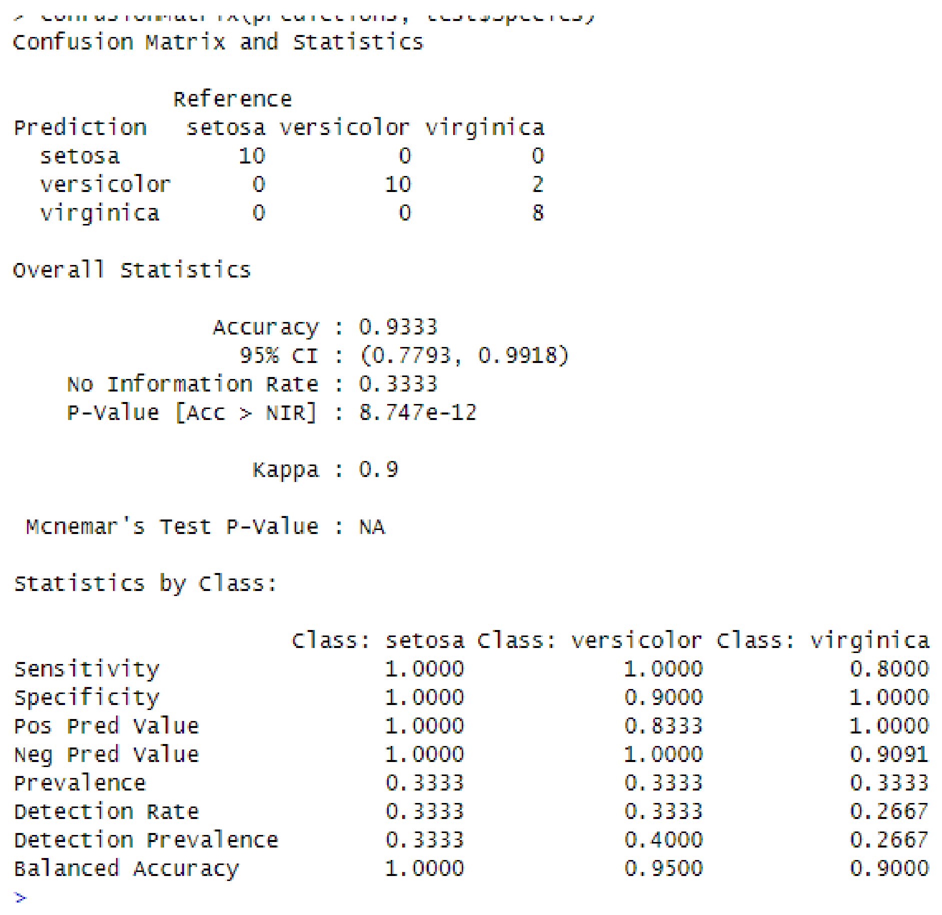
CODE:

```
library(datasets)
data(iris)
install.packages("caret")
library(caret)
set.seed(42)
trainindex <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
train <- iris[trainindex,]
test <- iris[-trainindex,]
library(e1071)
model <- naiveBayes(Species~ ., data = train)
predictions <- predict(model, newdata = test)
library(caret)
```

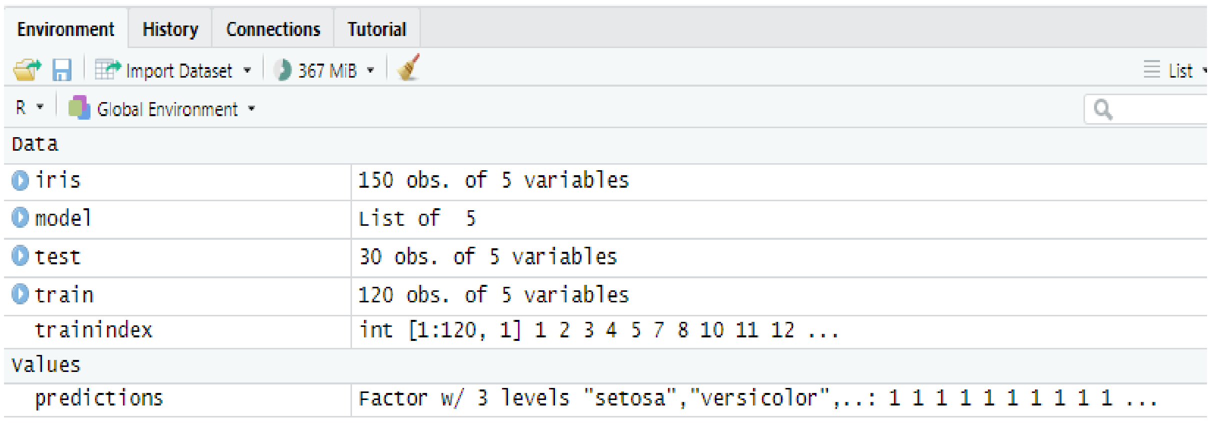
OUTPUT:



```
> library(caret)
Loading required package: ggplot2
Loading required package: lattice

> set.seed(42)

> trainindex <- createDataPartition(iris$Species, p = 0.8, list = FALSE)

> train <- iris[trainindex,]

> test <- iris[-trainindex,]

> library(e1071)

> model <- naiveBayes(Species~ ., data = train)

> predictions <- predict(model, newdata = test)

> library(caret)

> confusionMatrix(predictions, test$Species)
Confusion Matrix and Statistics
```



```
> confusionMatrix(predictions, test$Species)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8

Overall Statistics
                                          
               Accuracy : 0.9333          
                 95% CI : (0.7793, 0.9918)
    No Information Rate : 0.3333          
    P-Value [Acc > NIR] : 8.747e-12       
                                          
                  Kappa : 0.9             
                                          
 Mcnemar's Test P-Value : NA              

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            1.0000           0.8000
Specificity                 1.0000            0.9000           1.0000
Pos Pred Value              1.0000            0.8333           1.0000
Neg Pred Value              1.0000            1.0000           0.9091
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3333            0.3333           0.2667
Detection Prevalence        0.3333            0.4000           0.2667
Balanced Accuracy           1.0000            0.9500           0.9000
>
```



```
Data
  iris          150 obs. of 5 variables
  model         List of  5
  test          30 obs. of 5 variables
  train         120 obs. of 5 variables
  trainindex    int [1:120, 1] 1 2 3 4 5 7 8 10 11 12 ...
Values
  predictions   Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

RESULT:

The Naïve Bayes Algorithm has been implemented successfully.

PRACTICAL 8

AIM- Implement Random Forrest Algorithm.

PROCEDURE:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

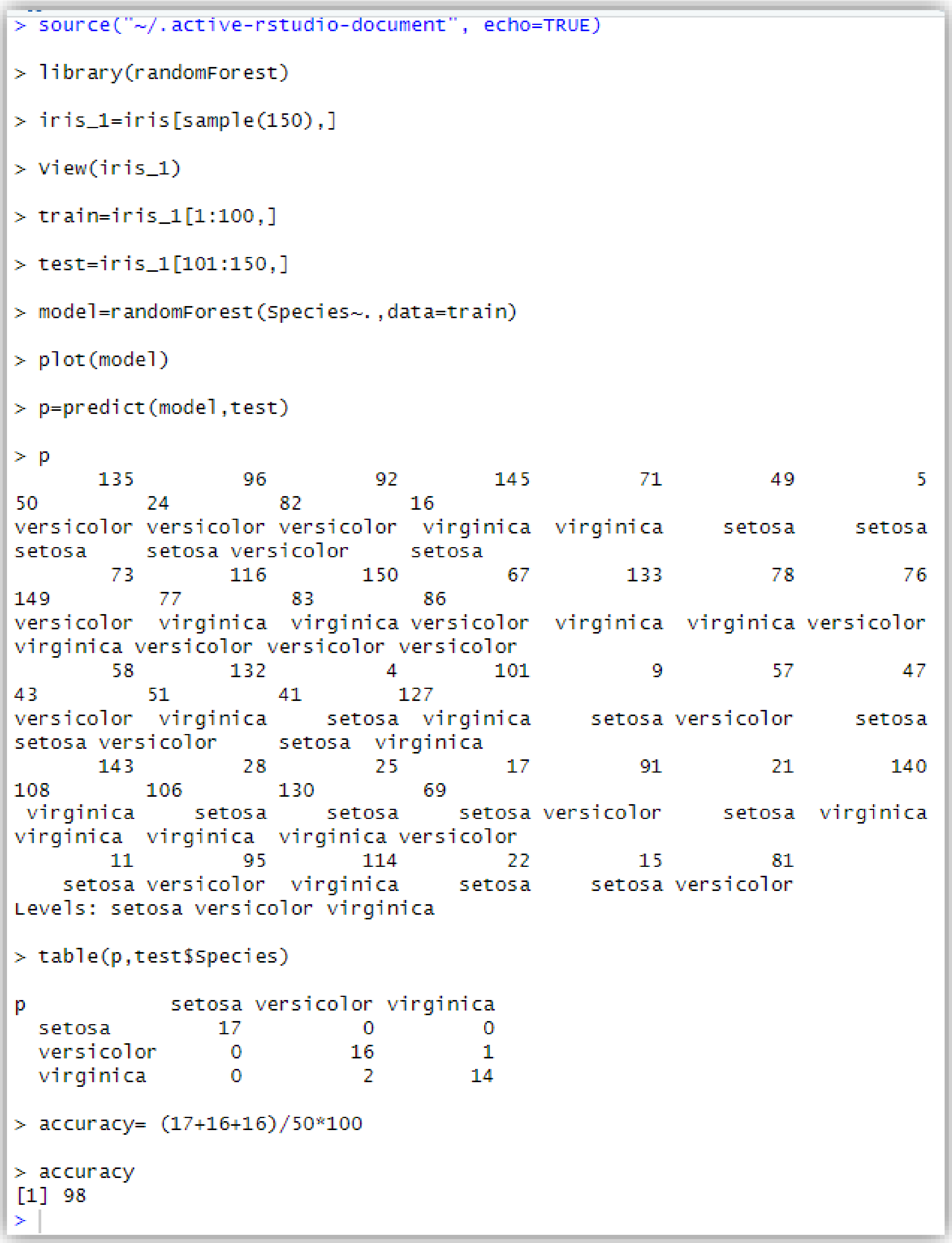
Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

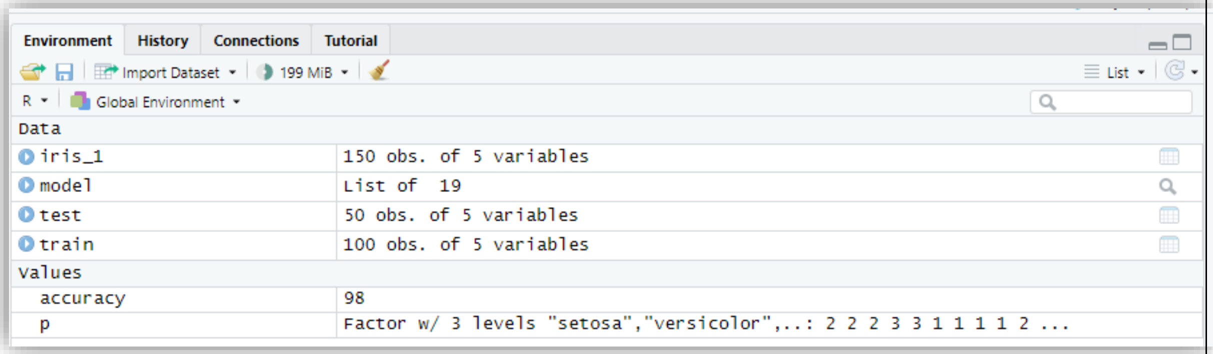
CODE:

```
install.packages("randomForest")

library(randomForest)

iris_1=iris[sample(150),]

View(iris_1)

train=iris_1[1:100,]

test=iris_1[101:150,]

model=randomForest(Species~.,data=train)

plot(model)

p=predict(model,test)

p

table(p,test$Species)

accuracy= (17+16+16)/50*100

accuracy
```

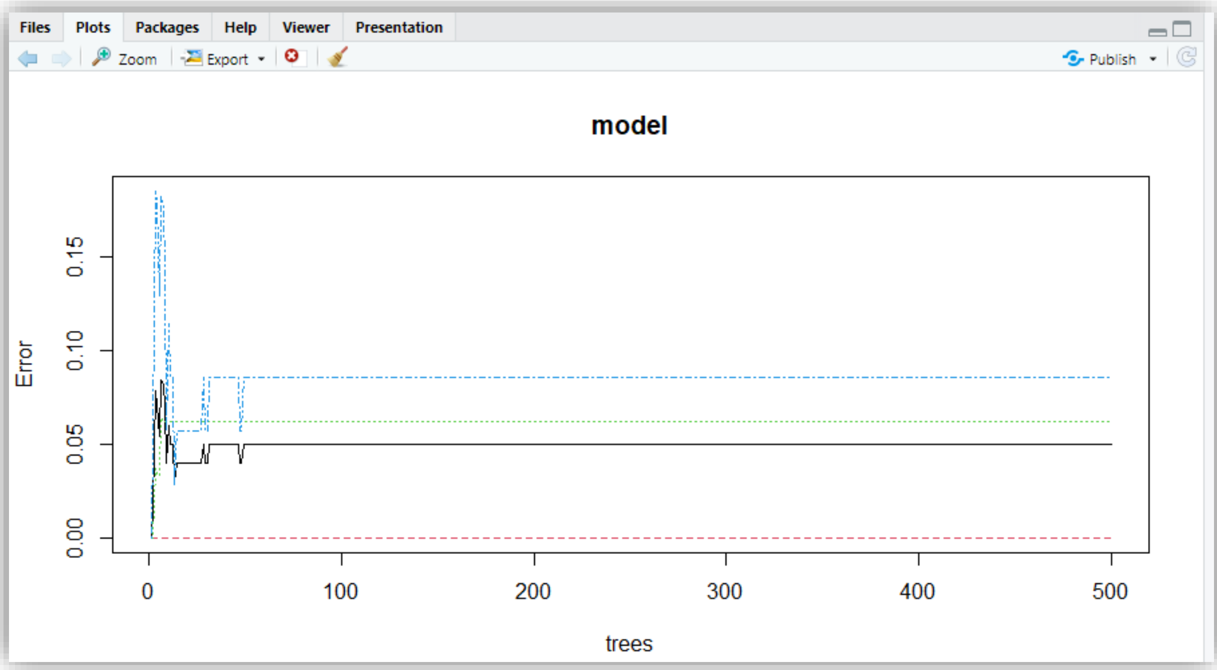
OUTPUT:



```
> source("~/.active-rstudio-document", echo=TRUE)

> library(randomForest)

> iris_1=iris[sample(150),]

> View(iris_1)

> train=iris_1[1:100,]

> test=iris_1[101:150,]

> model=randomForest(Species~.,data=train)

> plot(model)

> p=predict(model,test)

> p
       135        96        92       145        71        49         5 
50        24        82        16 
versicolor versicolor versicolor  virginica  virginica     setosa     setosa 
setosa     setosa versicolor     setosa 
        73       116       150        67       133        78        76 
149         77        83        86 
versicolor  virginica  virginica versicolor  virginica  virginica versicolor 
virginica versicolor versicolor versicolor 
        58       132         4       101         9        57        47 
43        51        41       127 
versicolor  virginica     setosa  virginica     setosa versicolor     setosa 
setosa versicolor     setosa  virginica 
       143        28        25        17        91        21       140 
108       106       130        69 
 virginica     setosa     setosa     setosa versicolor     setosa  virginica 
virginica  virginica  virginica versicolor 
        11        95       114        22        15        81 
    setosa versicolor  virginica     setosa     setosa versicolor 
Levels: setosa versicolor virginica

> table(p,test$Species)

p            setosa versicolor virginica
  setosa         17          0         0
  versicolor      0         16         1
  virginica       0          2        14

> accuracy= (17+16+16)/50*100

> accuracy
[1] 98
> 
```



```
Data
iris_1      150 obs. of 5 variables
model       List of  19
test        50 obs. of 5 variables
train       100 obs. of 5 variables
Values
accuracy    98
p           Factor w/ 3 levels "setosa","versicolor",..: 2 2 2 3 3 1 1 1 1 2 ...
```



RESULT : The Random Forest has been implemented successfully.

PRACTICAL 9

AIM : Implement K -Mean clustering in your own dataset create database using vector

PROCEDURE:

Step 1: Choose the number **K** clusters.

Step 2: Select at random K points, the centroids(Not necessarily from the given data).

Step 3: Assign each data point to closest centroid that forms K clusters.

Step 4: Compute and place the new centroid of each centroid.

Step 5: Reassign each data point to new cluster.

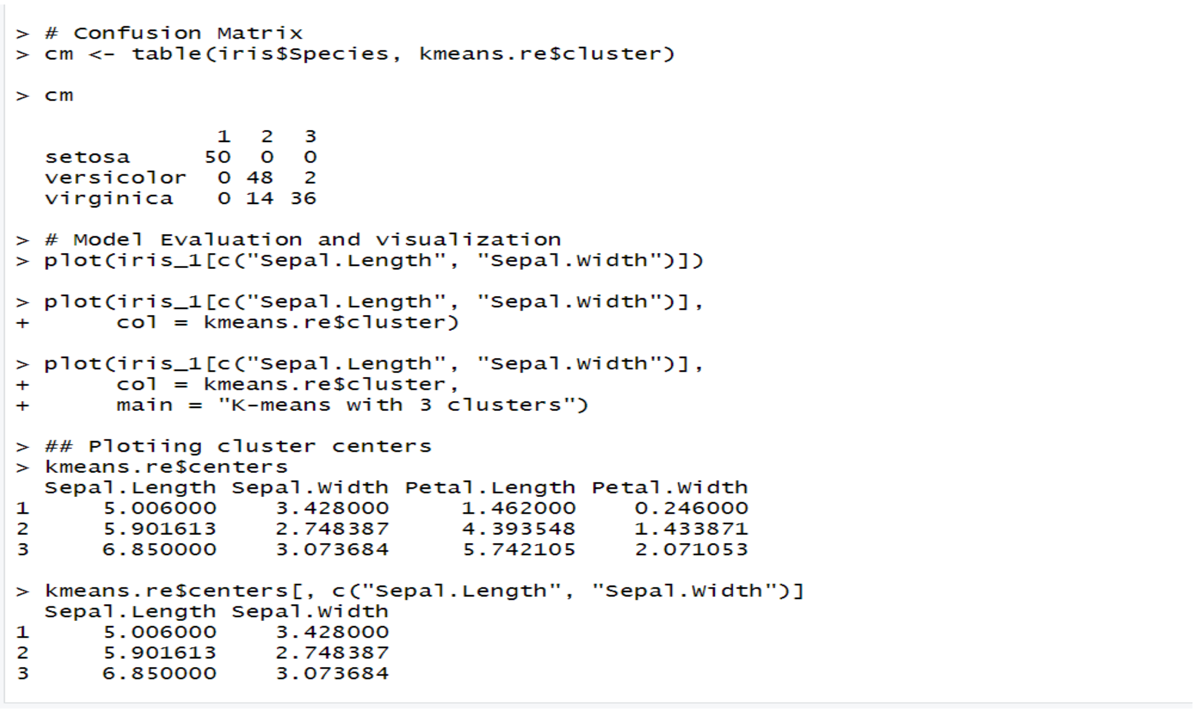
CODE :

```
install.packages("ClusterR")
install.packages("cluster")

# Loading package
library(ClusterR)
library(cluster)

# Removing initial label of
# Species from original dataset
iris_1 <- iris[, -5]

# Fitting K-Means clustering Model
# to training dataset
set.seed(240) # Setting seed
kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)
kmeans.re

# Cluster identification for
# each observation
kmeans.re$cluster

# Confusion Matrix
cm <- table(iris$Species, kmeans.re$cluster)
cm

# Model Evaluation and visualization
plot(iris_1[c("Sepal.Length", "Sepal.Width")])
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
col = kmeans.re$cluster)
```

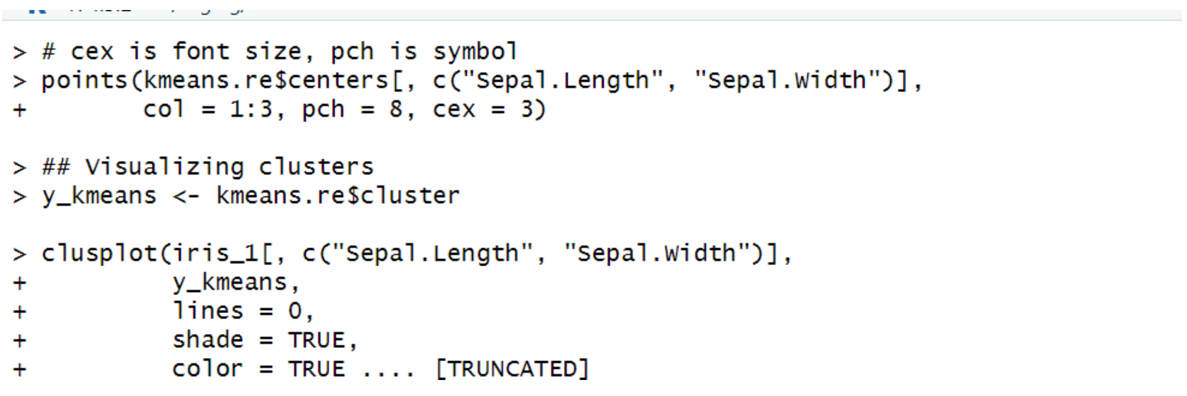
```
 plot(iris_1[c("Sepal.Length", "Sepal.Width")],
 col = kmeans.re$cluster,
main = "K-means with 3 clusters")
```

output;

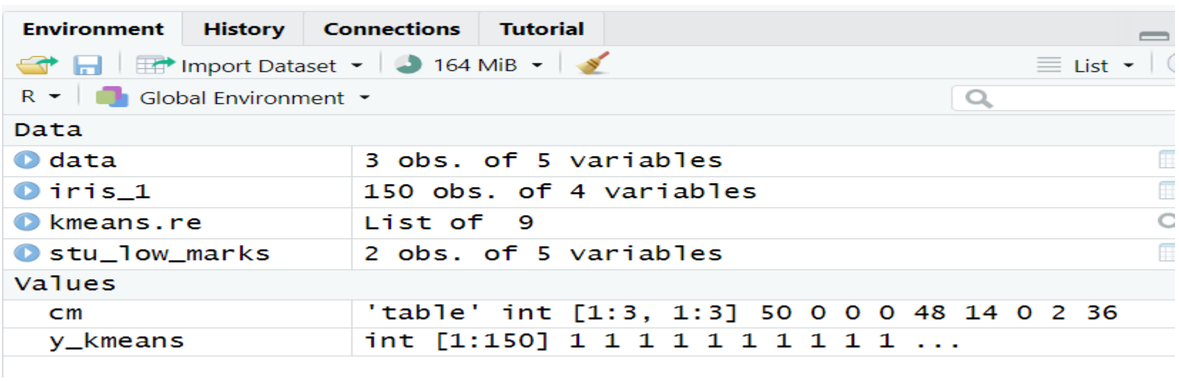
```
The downloaded binary packages are in
        C:\Users\Admin\AppData\Local\Temp\RtmpySWcnC\downloaded_packages

> # Loading package
> library(ClusterR)

> library(cluster)

> # Removing initial label of
> # Species from original dataset
> iris_1 <- iris[, -5]

> # Fitting K-Means clustering Model
> # to training dataset
> set.seed(240) # Setting seed

> kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)

> kmeans.re
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [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
 [38] 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
 [75] 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 3 2 3 3 3 3 2 3 3 3 3 3
[112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3
[149] 3 2
```



```
> # Confusion Matrix
> cm <- table(iris$Species, kmeans.re$cluster)

> cm

              1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0 14 36

> # Model Evaluation and visualization
> plot(iris_1[c("Sepal.Length", "Sepal.Width")])

> plot(iris_1[c("Sepal.Length", "Sepal.Width")],
+      col = kmeans.re$cluster)

> plot(iris_1[c("Sepal.Length", "Sepal.Width")],
+      col = kmeans.re$cluster,
+      main = "K-means with 3 clusters")

> ## Plotiing cluster centers
> kmeans.re$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

> kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]
  Sepal.Length Sepal.Width
1     5.006000    3.428000
2     5.901613    2.748387
3     6.850000    3.073684
```



```
> # cex is font size, pch is symbol
> points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")],
+        col = 1:3, pch = 8, cex = 3)

> ## Visualizing clusters
> y_kmeans <- kmeans.re$cluster

> clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],
+          y_kmeans,
+          lines = 0,
+          shade = TRUE,
+          color = TRUE .... [TRUNCATED]
```



```
Data
 data           3 obs. of 5 variables
 iris_1         150 obs. of 4 variables
 kmeans.re      List of  9
 stu_low_marks  2 obs. of 5 variables
Values
 cm             'table' int [1:3, 1:3] 50 0 0 0 48 14 0 2 36
 y_kmeans       int [1:150] 1 1 1 1 1 1 1 1 1 1 ...
```

PRACTICAL 9

AIM : Implement K -Mean clustering in your own dataset create database using vector

PROCEDURE:

Step 1: Choose the number **K** clusters.

Step 2: Select at random K points, the centroids(Not necessarily from the given data).

Step 3: Assign each data point to closest centroid that forms K clusters.

Step 4: Compute and place the new centroid of each centroid.

Step 5: Reassign each data point to new cluster.

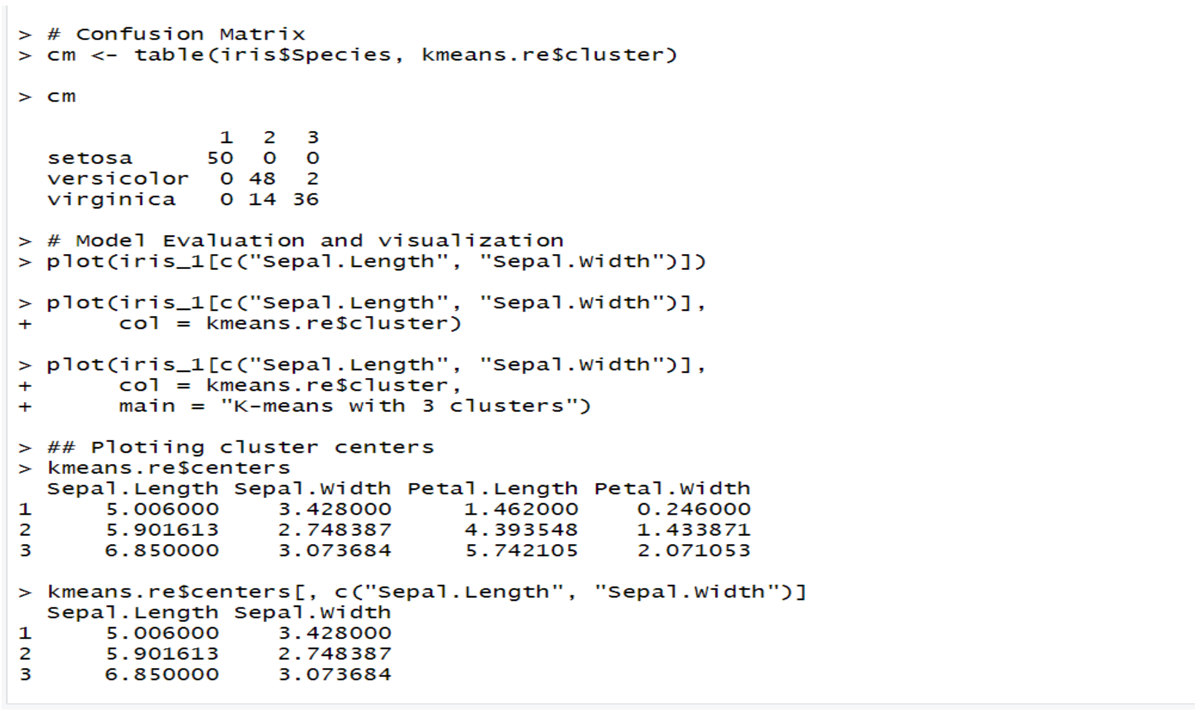
CODE :

```
install.packages("ClusterR")
install.packages("cluster")

# Loading package
library(ClusterR)
library(cluster)

# Removing initial label of
# Species from original dataset
iris_1 <- iris[, -5]

# Fitting K-Means clustering Model
# to training dataset
set.seed(240) # Setting seed
kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)
kmeans.re

# Cluster identification for
# each observation
kmeans.re$cluster

# Confusion Matrix
cm <- table(iris$Species, kmeans.re$cluster)
cm

# Model Evaluation and visualization
plot(iris_1[c("Sepal.Length", "Sepal.Width")])
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
col = kmeans.re$cluster)
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
col = kmeans.re$cluster,
```

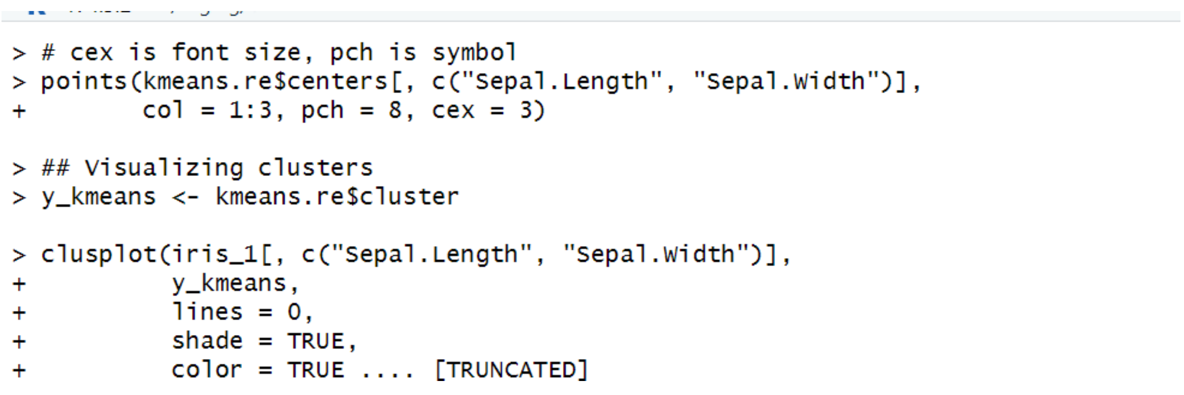
main = "K-means with 3 clusters")

OUTPUT:

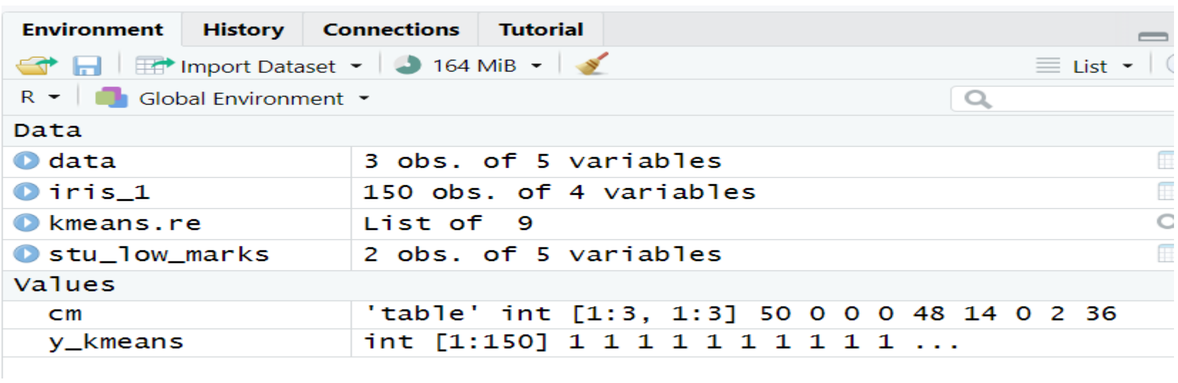
```
The downloaded binary packages are in
        C:\Users\Admin\AppData\Local\Temp\RtmpySWcnC\downloaded_packages

> # Loading package
> library(ClusterR)

> library(cluster)

> # Removing initial label of
> # Species from original dataset
> iris_1 <- iris[, -5]

> # Fitting K-Means clustering Model
> # to training dataset
> set.seed(240) # Setting seed

> kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)

> kmeans.re
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [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
 [38] 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
 [75] 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 3 2 3 3 3 3 2 3 3 3 3
[112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3
[149] 3 2
```



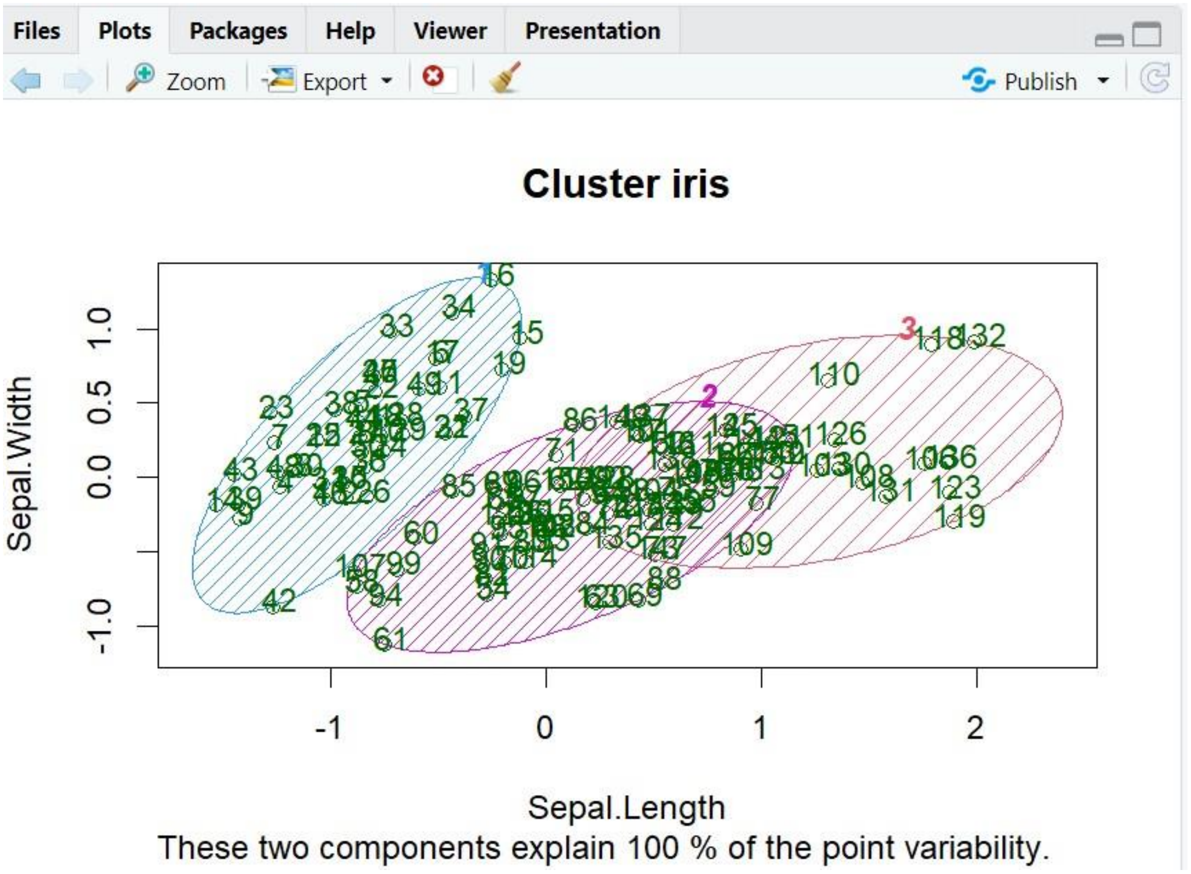
```
> # Confusion Matrix
> cm <- table(iris$Species, kmeans.re$cluster)

> cm

              1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0 14 36

> # Model Evaluation and visualization
> plot(iris_1[c("Sepal.Length", "Sepal.Width")])

> plot(iris_1[c("Sepal.Length", "Sepal.Width")],
+      col = kmeans.re$cluster)

> plot(iris_1[c("Sepal.Length", "Sepal.Width")],
+      col = kmeans.re$cluster,
+      main = "K-means with 3 clusters")

> ## Plotiing cluster centers
> kmeans.re$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

> kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]
  Sepal.Length Sepal.Width
1     5.006000    3.428000
2     5.901613    2.748387
3     6.850000    3.073684
```



```
> # cex is font size, pch is symbol
> points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")],
+        col = 1:3, pch = 8, cex = 3)

> ## Visualizing clusters
> y_kmeans <- kmeans.re$cluster

> clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],
+          y_kmeans,
+          lines = 0,
+          shade = TRUE,
+          color = TRUE .... [TRUNCATED]
```



```
Data
  data          3 obs. of 5 variables
  iris_1        150 obs. of 4 variables
  kmeans.re     List of  9
  stu_low_marks 2 obs. of 5 variables
Values
  cm            'table' int [1:3, 1:3] 50 0 0 0 48 14 0 2 36
  y_kmeans      int [1:150] 1 1 1 1 1 1 1 1 1 1 ...
```



RESULT: The K-Mean Clustering Algorithm has been implemented successfully.

PRACTICAL-10

AIM: Implement Linear Regression on iris dataset.

PROCEDURE:

Step 1: Load the data: Load the dataset that you want to use for predicting the target variable. The dataset should have at least one dependent variable and one independent variable.

Step 2. Split the data: Split the dataset into training and testing datasets. The training dataset is used to fit the model, and the testing dataset is used to evaluate the model's performance.

Step 3. Prepare the data: Prepare the data by cleaning and preprocessing it. This includes removing missing values, handling categorical variables, and scaling the data if necessary.

Step 4. Choose the type of Linear Regression: Choose the type of Linear Regression based on the problem you are trying to solve. There are two types of Linear Regression: Simple Linear Regression and Multiple Linear Regression.

Step 5. Fit the model: Fit the Linear Regression model on the training dataset using the chosen algorithm. In the case of Simple Linear Regression, the model is a straight line that best fits the data. In the case of Multiple Linear Regression, the model is a hyperplane that best fits the data.

Step 6. Evaluate the model: Evaluate the performance of the model on the testing dataset. This can be done using metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R-squared value, or Adjusted R squared value.

Step 7. Make predictions: Use the trained Linear Regression model to make predictions on new data. This can be done using the predict() function in R.

Step 8. Fine-tune the model: Fine-tune the Linear Regression model by tweaking the hyperparameters and optimizing the model using techniques such as regularization, feature selection, and cross-validation.

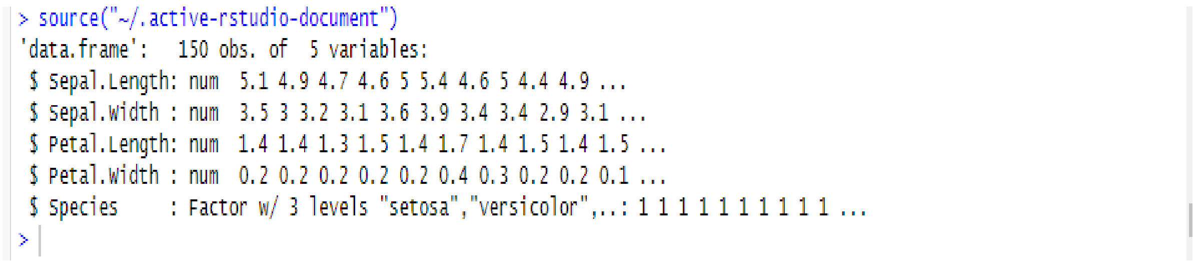
Step 9. Deploy the model: Deploy the trained Linear Regression model to make predictions on new data in production environments

CODE:

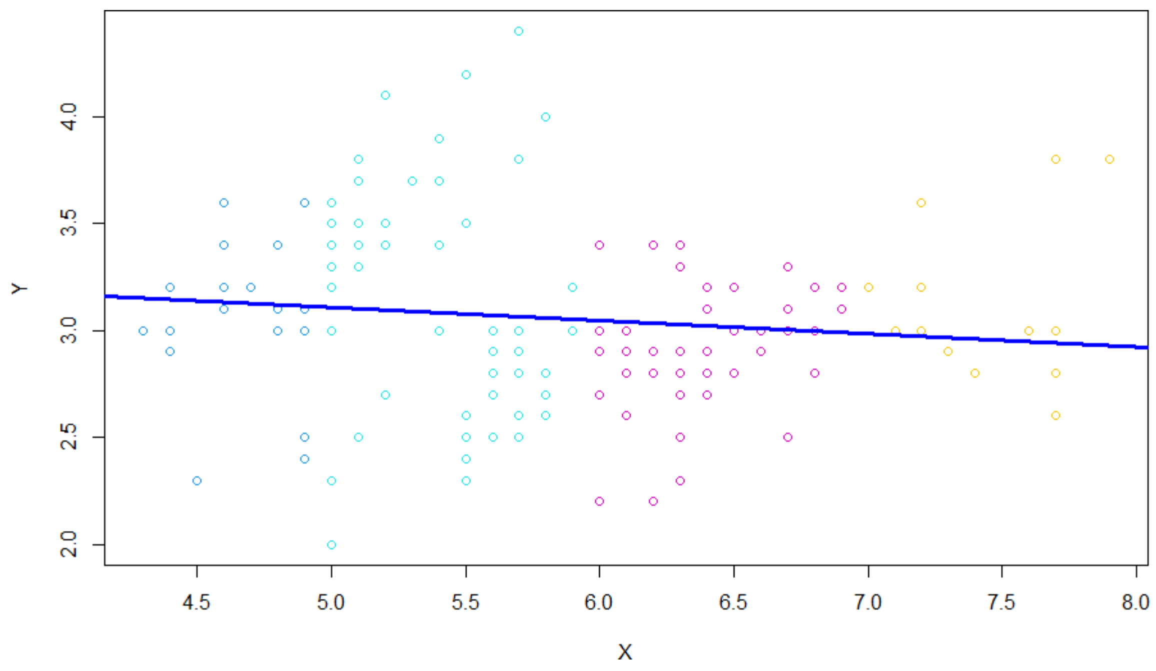
```
require("datasets")
data("iris")  str(iris)
head(iris)
Y<- iris[,"Sepal.Width"] # select Target attribute
```

X<- iris[,"Sepal.Length"] # select Predictor attribute head(X)
xycorr<- cor(Y,X, method="pearson") # find pearson correlation coefficient
xycorr # a value near 1 implies high correlation and that near 0 shows low correlation plot(Y~X, col=X)
model1<- lm(Y~X)
model1 # provides regression line coefficients i.e. slope and y-intercept plot(Y~X, col=X) # scatter plot between X and Y
abline(model1, col="blue", lwd=3)

OUTPUT:



```
> source("~/.active-rstudio-document")
'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 ...
>
```



RESULT : The linear Regression has been implemented successfully.

PRACTICAL 11

AIM- Implement SVM algorithm using WDBC datasets.

PROCEDURE:

Here are the steps for implementing the SVM algorithm:

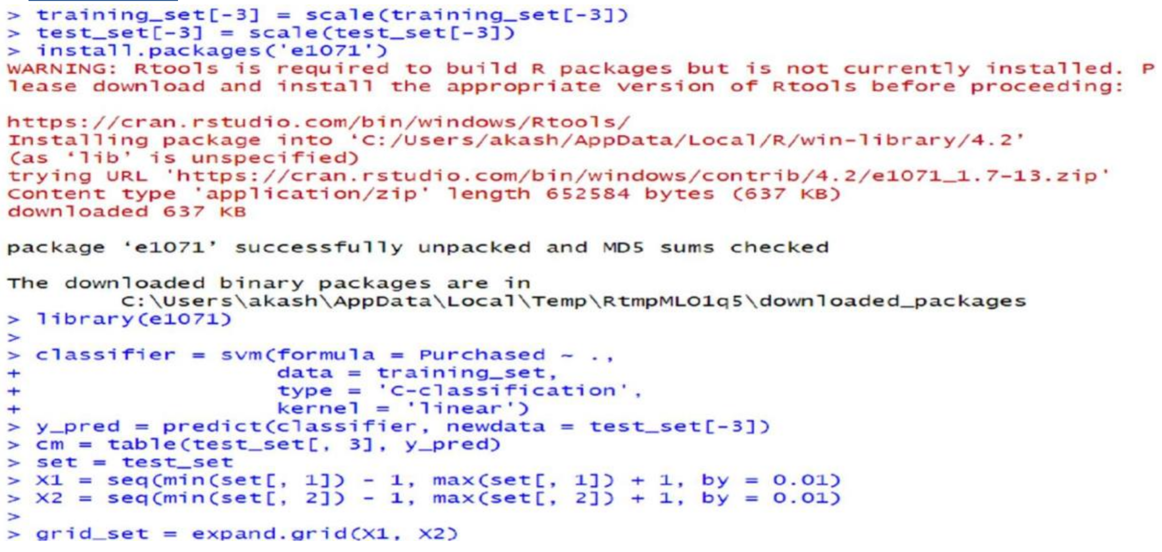
1. Load the data: Load the dataset that you want to classify or predict. The dataset should be split into a training set and a test set.
2. Normalize the data: Normalize the data by scaling the features to have zero mean and unit variance.
3. Choose the kernel function: Choose a kernel function that maps the data to a higherdimensional space. The most commonly used kernel functions are the linear kernel, the polynomial kernel, and the radial basis function (RBF) kernel.
4. Train the model: Train the SVM model on the training data using the chosen kernel function.
5. Choose the regularization parameter: Choose the regularization parameter (C) that controls the trade-off between maximizing the margin and minimizing the classification error.
6. Test the model: Test the SVM model on the test data to evaluate its performance.
7. Tune the hyperparameters: Tune the hyperparameters of the SVM model using techniques such as grid search or random search to find the best combination of kernel function, regularization parameter, and other hyperparameters.
8. Deploy the model: Deploy the trained SVM model to make predictions on new data.

CODE:

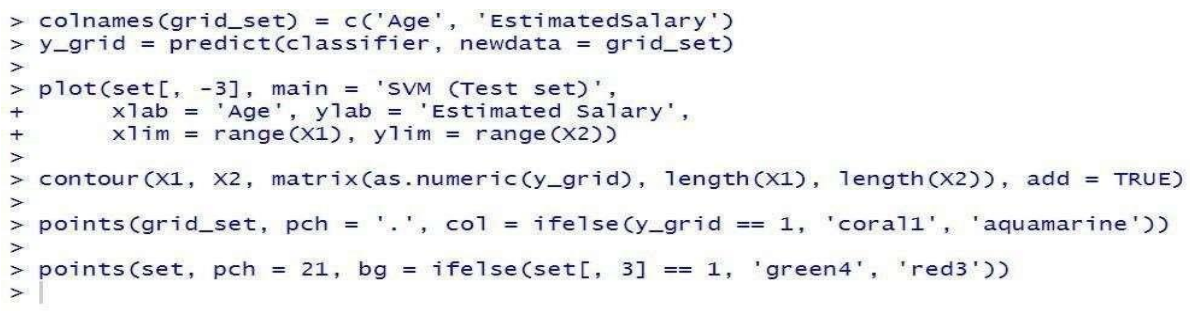
```
dataset =

read.csv('social.csv')dataset

= dataset[3:5]

dataset$Purchased = factor(dataset$Purchased, levels =
c(0, 1))install.packages('caTools')

 library(caTools) set.seed(123)

split = sample.split(dataset$Purchased, SplitRatio = 0.75)
```

```
training_set = subset(dataset, split == TRUE)
 test_set = subset(dataset, split == FALSE training_set[-3]

= scale(training_set[-3]) test_set[-3] =

scale(test_set[-3])install.packages('e1071')

library(e1071)
classifier = svm(formula =
Purchased ~ .,data = training_set,
type = 'C- classification'  kernel =
'linear')

y_pred = predict(classifier,
newdata =test_set[-3]) cm =
table(test_set[, 3], y_pred) set =
test_set

X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01) X2

= seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by =

0.01)grid_set = expand.grid(X1, X2)


colnames(grid_set) = c('Age',

'EstimatedSalary') y_grid =

predict(classifier, newdata = grid_set)

plot(set[, -3], main = 'SVM (Test set)',

                                          xlab  =

'Age', ylab = 'Estimated Salary',xlim =

range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add =

TRUE) points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1',

'aquamarine')) points(set,pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

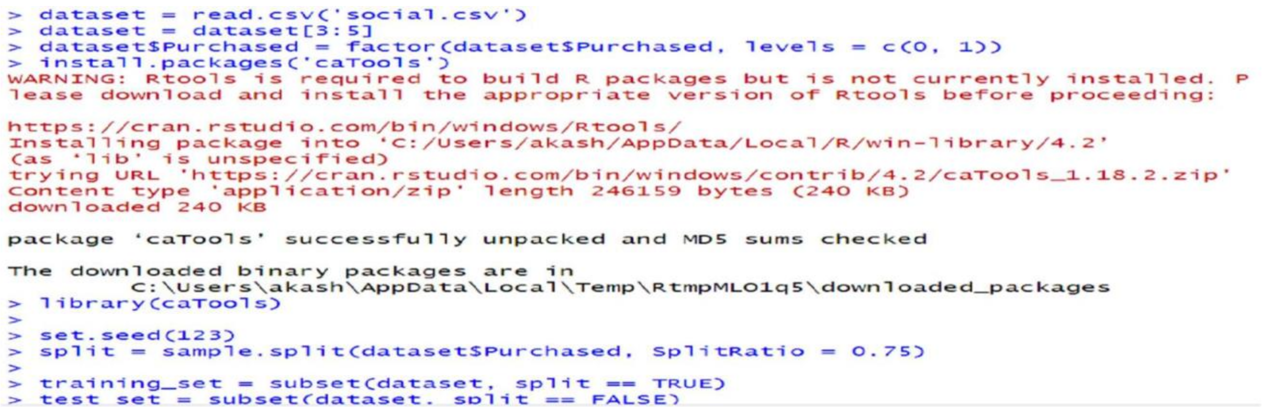
OUTPUT:-



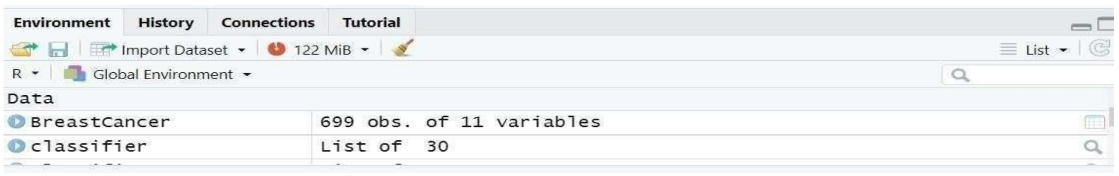
```
> training_set[-3] = scale(training_set[-3])
> test_set[-3] = scale(test_set[-3])
> install.packages('e1071')
WARNING: Rtools is required to build R packages but is not currently installed. P
lease download and install the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/akash/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/e1071_1.7-13.zip'
Content type 'application/zip' length 652584 bytes (637 KB)
downloaded 637 KB

package 'e1071' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\akash\AppData\Local\Temp\RtmpMLO1q5\downloaded_packages
> library(e1071)
>
> classifier = svm(formula = Purchased ~ .,
+                  data = training_set,
+                  type = 'C-classification',
+                  kernel = 'linear')
> y_pred = predict(classifier, newdata = test_set[-3])
> cm = table(test_set[, 3], y_pred)
> set = test_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
>
> grid_set = expand.grid(X1, X2)
```



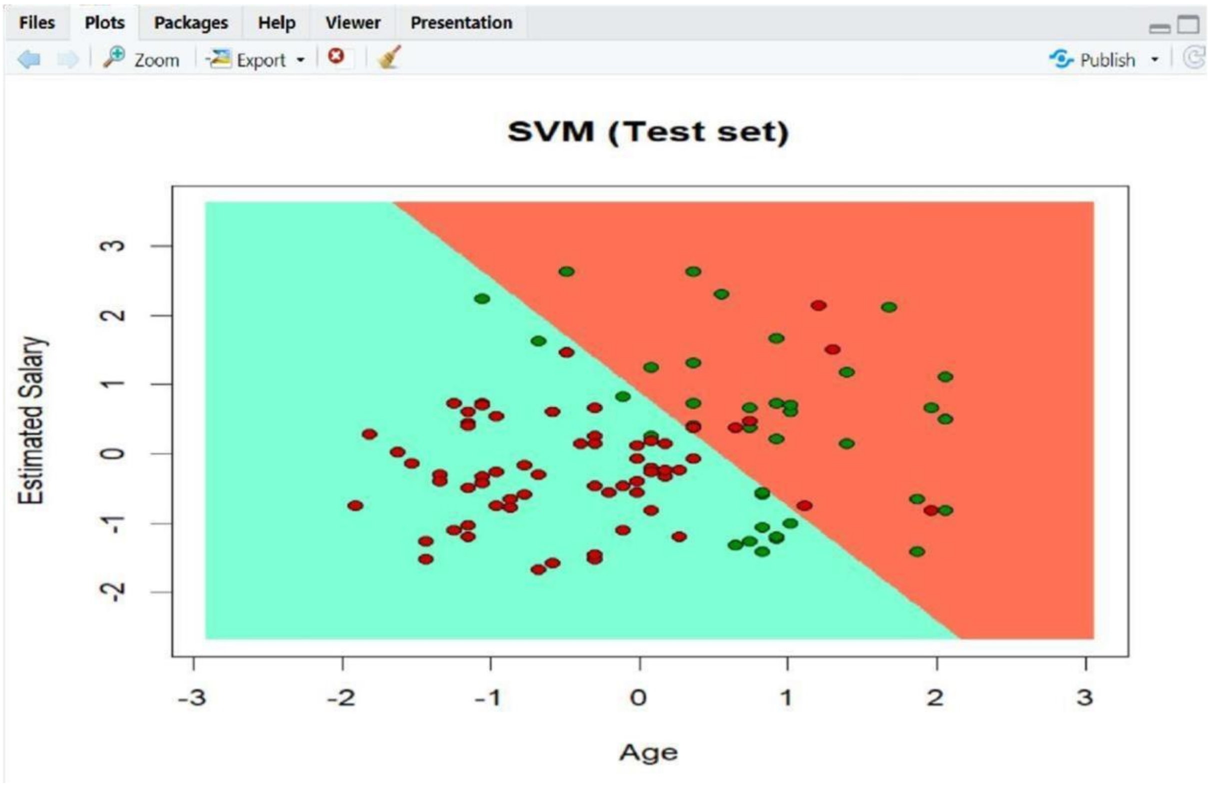
```
> colnames(grid_set) = c('Age', 'EstimatedSalary')
> y_grid = predict(classifier, newdata = grid_set)
>
> plot(set[, -3], main = 'SVM (Test set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
>
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
>
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))
>
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
> |
```



```
> dataset = read.csv('social.csv')
> dataset = dataset[3:5]
> dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
> install.packages('caTools')
WARNING: Rtools is required to build R packages but is not currently installed. P
lease download and install the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/akash/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/caTools_1.18.2.zip'
Content type 'application/zip' length 246159 bytes (240 KB)
downloaded 240 KB

package 'caTools' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\akash\AppData\Local\Temp\RtmpMLO1q5\downloaded_packages
> library(caTools)
>
> set.seed(123)
> split = sample.split(dataset$Purchased, SplitRatio = 0.75)
>
> training_set = subset(dataset, split == TRUE)
> test_set = subset(dataset, split == FALSE)
```



```
Data
BreastCancer      699 obs. of 11 variables
classifier        List of  30
```



Result:- The SVM Algorithm have been implemented successfully.

PRACTICAL 12

Aim: Implement Logistic regression on IRIS datasets.

PROCEDURE:

1. Import the dataset: Import the dataset which contains the input variables (or the independent variables) and the output variable (or the dependent variable).

2. Data Pre-processing: Check for missing values and handle them, handle outliers if any, and remove or transform variables that do not contribute to the model.

3. Split the dataset: Split the dataset into a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate the performance of the model.

4. Feature Scaling: Scale the input variables (or the independent variables) to have the same range to ensure that no variable dominates the others in the modeling process.

5. Model Training: Train the logistic regression model using the training set. The logistic regression model calculates the probability of the output variable taking a particular value given the input variables (or the independent variables).

6. Model Evaluation: Evaluate the performance of the model using the testing set. There are several metrics to evaluate the performance of the model, such as accuracy, precision, recall, F1 score, and area under the ROC curve.

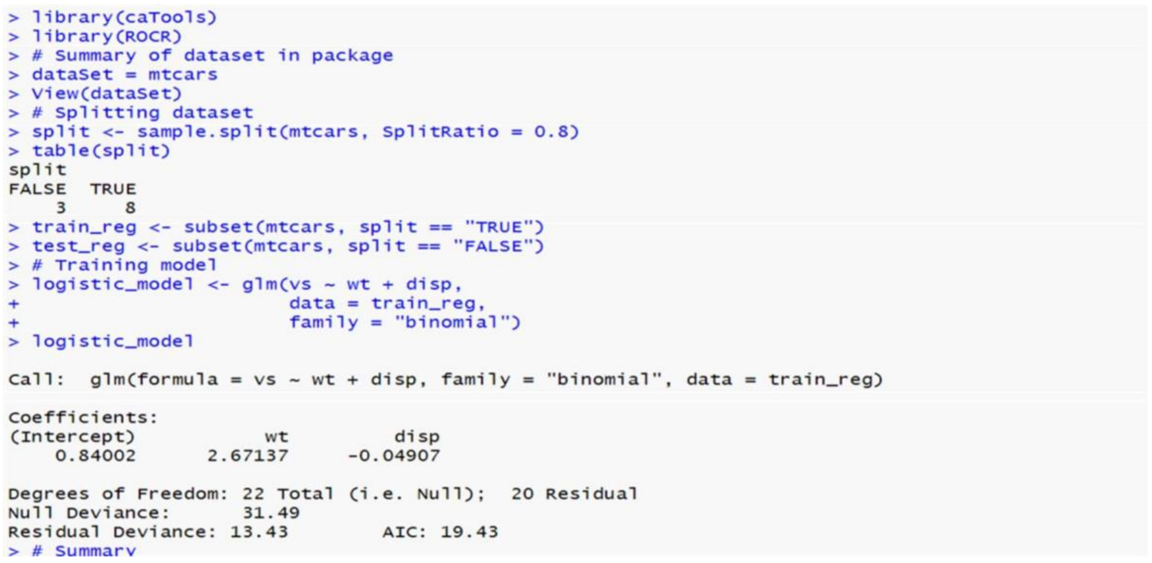
7. Model Improvement: If the model does not perform well, try to improve the model by changing the hyperparameters, selecting the relevant features, or using other modeling techniques.

8. Model Deployment: Once the model is finalized and performs well on the testing set, deploy it to the production environment for making predictions on new data.

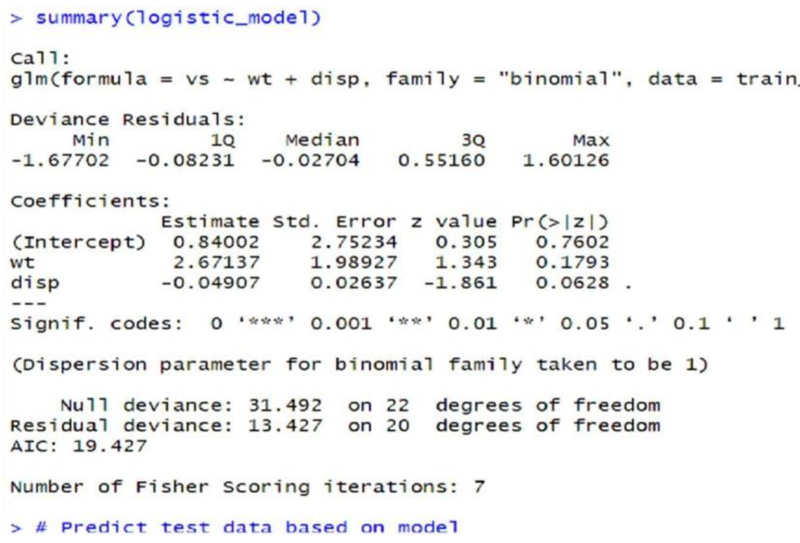
CODE:

```
install.packages("dplyr")
install.packages("caTools"
)install.packages("ROCR")

 # loading packages

library(dplyr)
library(caTools
)
```

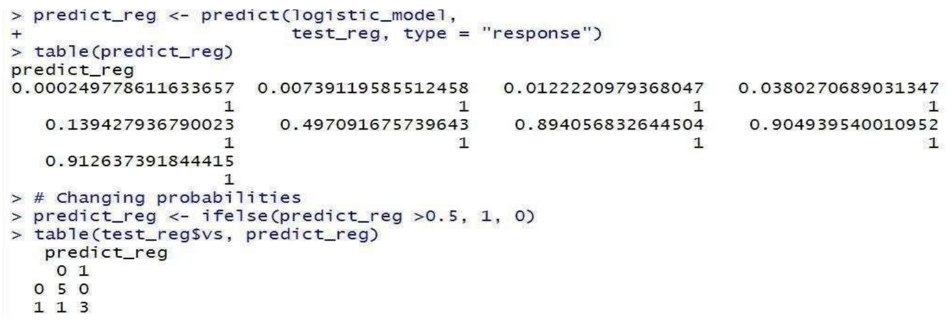
```
library(ROCR)    dataSet    =
mtcars View(dataSet)

split <- sample.split(mtcars, SplitRatio = 0.8)

table(split)

train_reg    <-    subset(mtcars,    split    ==

"TRUE")test_reg <- subset(mtcars, split ==

"FALSE") # Training model

 logistic_model  <-  glm(vs  ~  wt  +
disp,data = train_reg,

 family = "binomial")

 logistic_model
summary(logistic_model)
predict_reg
                                             <
-
predict(logistic_model,

 test_reg, type = "response") table(predict_reg)

predict_reg <- ifelse(predict_reg >0.5, 1, 0)

table(test_reg$vs, predict_reg)

missing_classerr <- mean(predict_reg != test_reg$vs)

print(paste('Accuracy =', 1 - missing_classerr))

ROCPred <- prediction(predict_reg, test_reg$vs)

ROCPer <- performance(ROCPred, measure = "tpr", x.measure  = "fpr")

auc <- performance(ROCPred, measure = "auc")
auc <- auc@y.values[[1]]

auc plot(ROCPer) plot(ROCPer, colorize = TRUE,
print.cutoffs.at = seq(0.1, by = 0.1),

  main = "ROC CURVE")

abline(a = 0, b = 1) auc <- round(auc, 4)

legend(.6, .4, auc, title = "AUC", cex = 1)
```



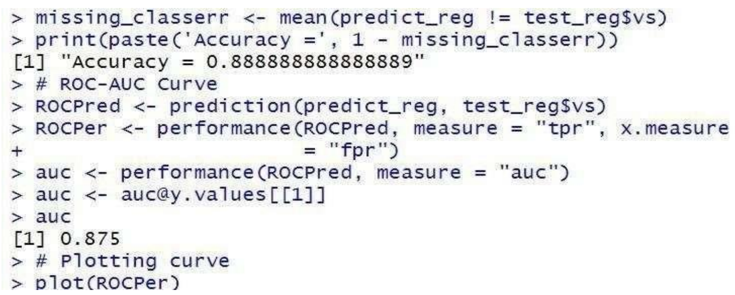
```
> library(caTools)
> library(ROCR)
> # Summary of dataset in package
> dataSet = mtcars
> View(dataSet)
> # Splitting dataset
> split <- sample.split(mtcars, SplitRatio = 0.8)
> table(split)
split
FALSE  TRUE
    3     8
> train_reg <- subset(mtcars, split == "TRUE")
> test_reg <- subset(mtcars, split == "FALSE")
> # Training model
> logistic_model <- glm(vs ~ wt + disp,
+                       data = train_reg,
+                       family = "binomial")
> logistic_model

Call:  glm(formula = vs ~ wt + disp, family = "binomial", data = train_reg)

Coefficients:
(Intercept)           wt         disp
    0.84002      2.67137     -0.04907

Degrees of Freedom: 22 Total (i.e. Null);  20 Residual
Null Deviance:      31.49
Residual Deviance: 13.43        AIC: 19.43
> # Summary
```



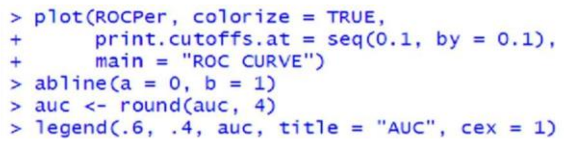
```
> summary(logistic_model)

Call:
glm(formula = vs ~ wt + disp, family = "binomial", data = train_reg)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.67702  -0.08231  -0.02704   0.55160   1.60126

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.84002    2.75234   0.305   0.7602
wt           2.67137    1.98927   1.343   0.1793
disp        -0.04907    0.02637  -1.861   0.0628 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 31.492  on 22  degrees of freedom
Residual deviance: 13.427  on 20  degrees of freedom
AIC: 19.427

Number of Fisher Scoring iterations: 7

> # Predict test data based on model
```



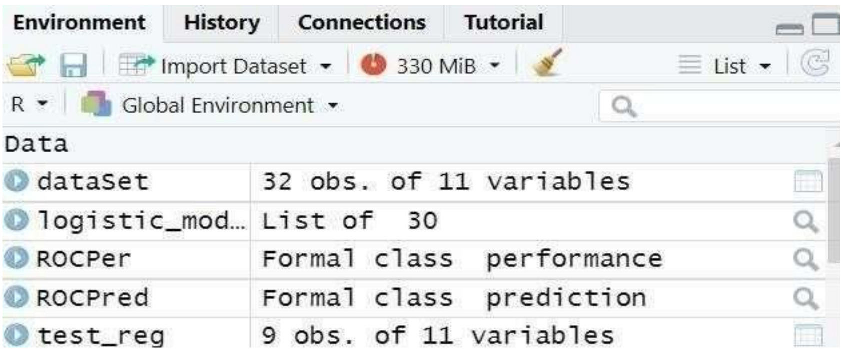
```
> predict_reg <- predict(logistic_model,
+                        test_reg, type = "response")
> table(predict_reg)
predict_reg
0.000249778611633657  0.00739119585512458   0.0122220979368047   0.0380270689031347
                   1                    1                    1                    1
   0.139427936790023    0.497091675739643    0.894056832644504    0.904939540010952
                   1                    1                    1                    1
   0.912637391844415
                   1
> # Changing probabilities
> predict_reg <- ifelse(predict_reg >0.5, 1, 0)
> table(test_reg$vs, predict_reg)
   predict_reg
    0 1
  0 5 0
  1 1 3
```



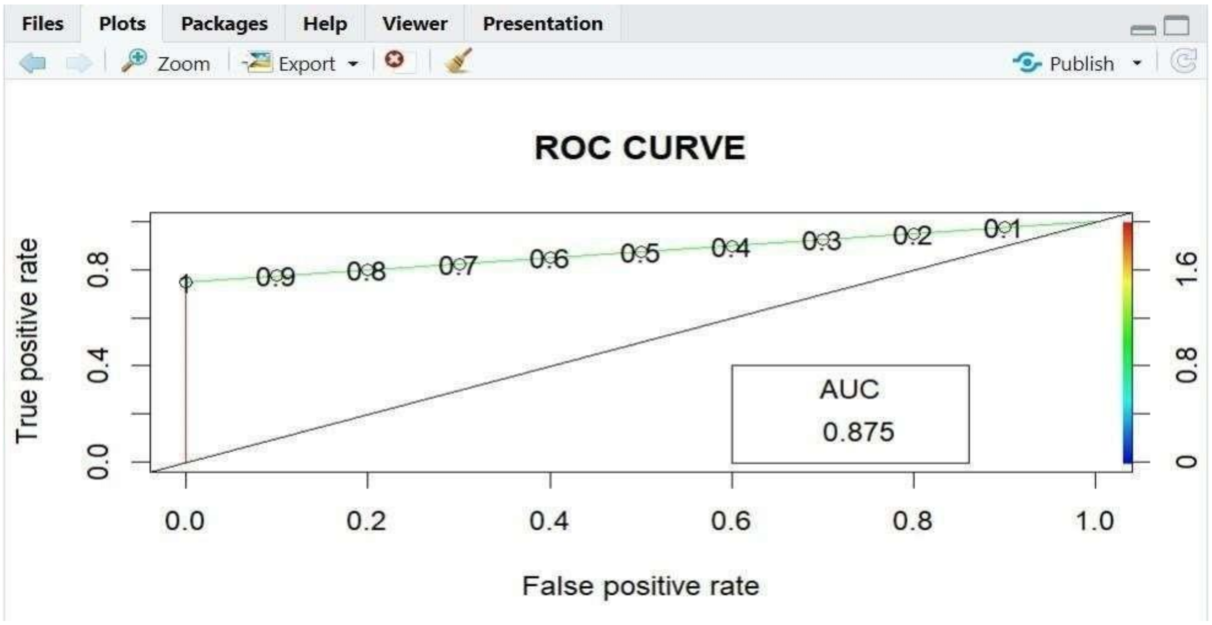
```
> missing_classerr <- mean(predict_reg != test_reg$vs)
> print(paste('Accuracy =', 1 - missing_classerr))
[1] "Accuracy = 0.888888888888889"
> # ROC-AUC Curve
> ROCPred <- prediction(predict_reg, test_reg$vs)
> ROCPer <- performance(ROCPred, measure = "tpr", x.measure
+                       = "fpr")
> auc <- performance(ROCPred, measure = "auc")
> auc <- auc@y.values[[1]]
> auc
[1] 0.875
> # Plotting curve
> plot(ROCPer)
```



```
> plot(ROCPer, colorize = TRUE,
+      print.cutoffs.at = seq(0.1, by = 0.1),
+      main = "ROC CURVE")
> abline(a = 0, b = 1)
> auc <- round(auc, 4)
> legend(.6, .4, auc, title = "AUC", cex = 1)
```



```
Data
dataSet          32 obs. of 11 variables
logistic_mod...  List of  30
ROCPer           Formal class  performance
ROCPred          Formal class  prediction
test_reg         9 obs. of 11 variables
```



<u>Result:</u> The logistic regression have been implemented successfully

PRACTICAL 13

AIM: Implement Apriori algorithm.

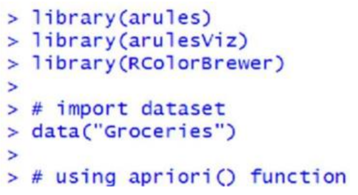
PROCEDURE:

1. Read each item in the transaction.
2. Calculate the support of every item.
3. If support is less than minimum support, discard the item. Else, insert it into frequent itemset.
4. Calculate confidence for each non- empty subset.
5. If confidence is less than minimum confidence, discard the subset. Else, it into strong rules.

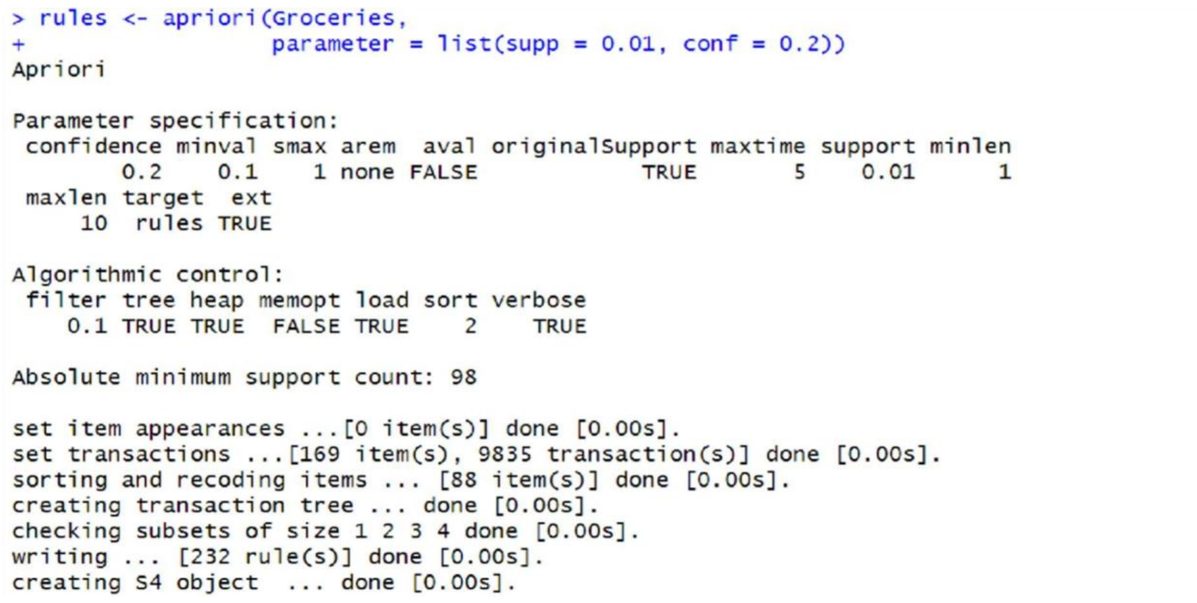
CODE:

```
install.packages("arules")
install.packages("arulesViz")
install.packages("RColorBrewer"
)library(arules) library(arulesViz)
library(RColorBrewer)
data("Groceries") rules <-
apriori(Groceries, parameter =
list(supp = 0.01, conf = 0.2))
inspect(rules[1:10])
arules::itemFrequencyPlot

(Groceries, topN = 20,

 col = brewer.pal(8, 'Pastel2'),

 main = 'Relative Item Frequency Plot',
type = "relative" ylab = "Item Frequency (Relative)")
```

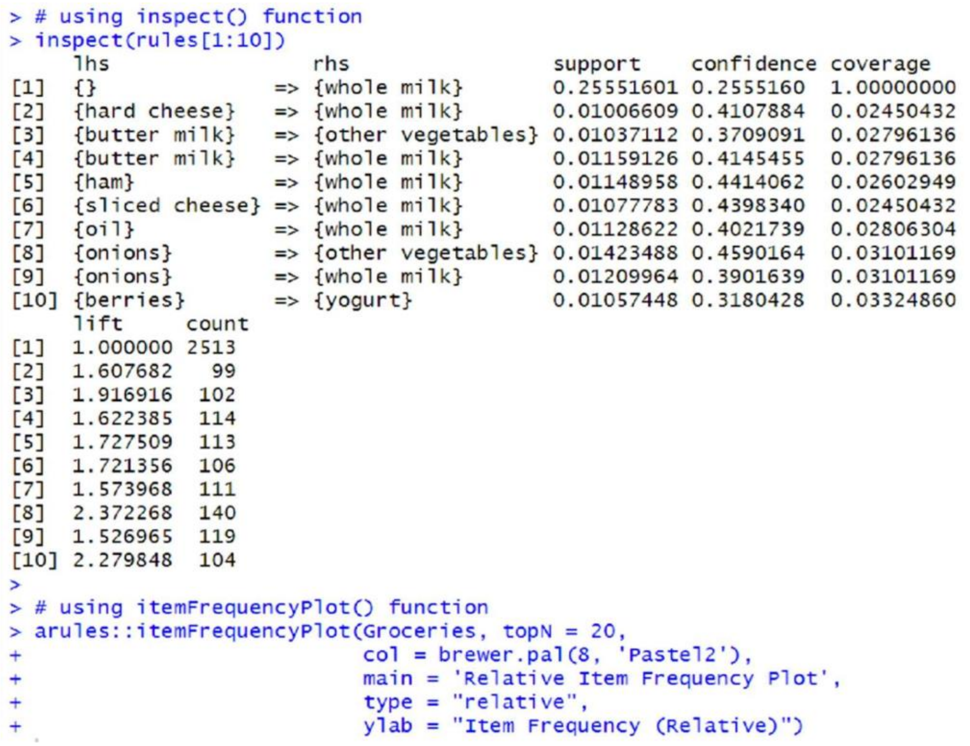
Output:



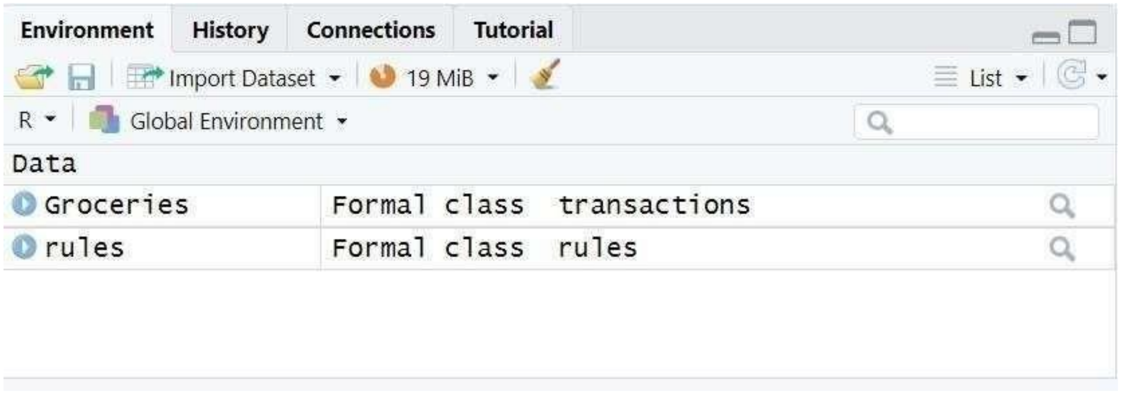
```
> library(arules)
> library(arulesViz)
> library(RColorBrewer)
>
> # import dataset
> data("Groceries")
>
> # using apriori() function
```



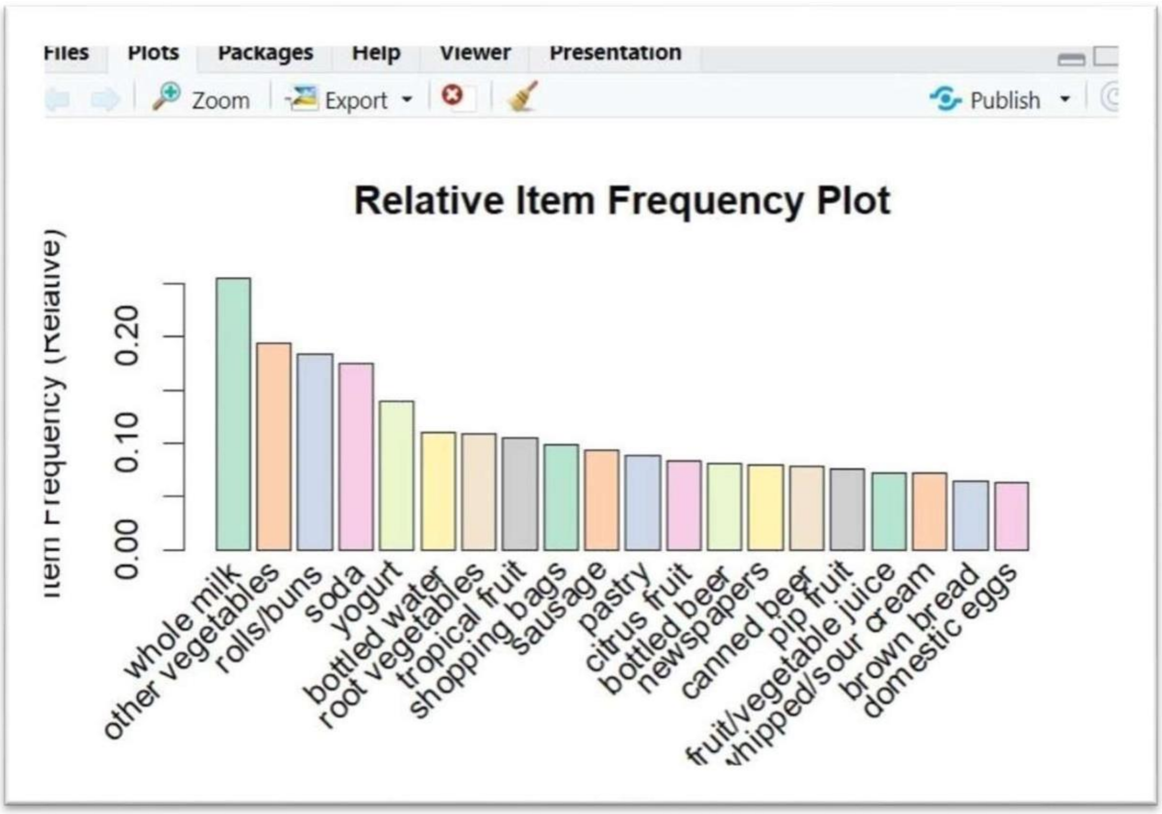
```
> rules <- apriori(Groceries,
+                  parameter = list(supp = 0.01, conf = 0.2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.2    0.1    1 none FALSE            TRUE       5    0.01      1
 maxlen target  ext
     10  rules TRUE

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

Absolute minimum support count: 98

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [232 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```



```
> # using inspect() function
> inspect(rules[1:10])
     lhs                rhs                support    confidence coverage
[1]  {}              => {whole milk}       0.25551601 0.2555160  1.00000000
[2]  {hard cheese}   => {whole milk}       0.01006609 0.4107884  0.02450432
[3]  {butter milk}   => {other vegetables} 0.01037112 0.3709091  0.02796136
[4]  {butter milk}   => {whole milk}       0.01159126 0.4145455  0.02796136
[5]  {ham}           => {whole milk}       0.01148958 0.4414062  0.02602949
[6]  {sliced cheese} => {whole milk}       0.01077783 0.4398340  0.02450432
[7]  {oil}           => {whole milk}       0.01128622 0.4021739  0.02806304
[8]  {onions}        => {other vegetables} 0.01423488 0.4590164  0.03101169
[9]  {onions}        => {whole milk}       0.01209964 0.3901639  0.03101169
[10] {berries}       => {yogurt}           0.01057448 0.3180428  0.03324860
     lift     count
[1]  1.000000 2513
[2]  1.607682   99
[3]  1.916916  102
[4]  1.622385  114
[5]  1.727509  113
[6]  1.721356  106
[7]  1.573968  111
[8]  2.372268  140
[9]  1.526965  119
[10] 2.279848  104
>
> # using itemFrequencyPlot() function
> arules::itemFrequencyPlot(Groceries, topN = 20,
+                           col = brewer.pal(8, 'Pastel2'),
+                           main = 'Relative Item Frequency Plot',
+                           type = "relative",
+                           ylab = "Item Frequency (Relative)")
+
```



```
Data
Groceries    Formal class  transactions
rules        Formal class  rules
```



<u>**Result:**</u> The Apriori algorithm have been implemented successfully

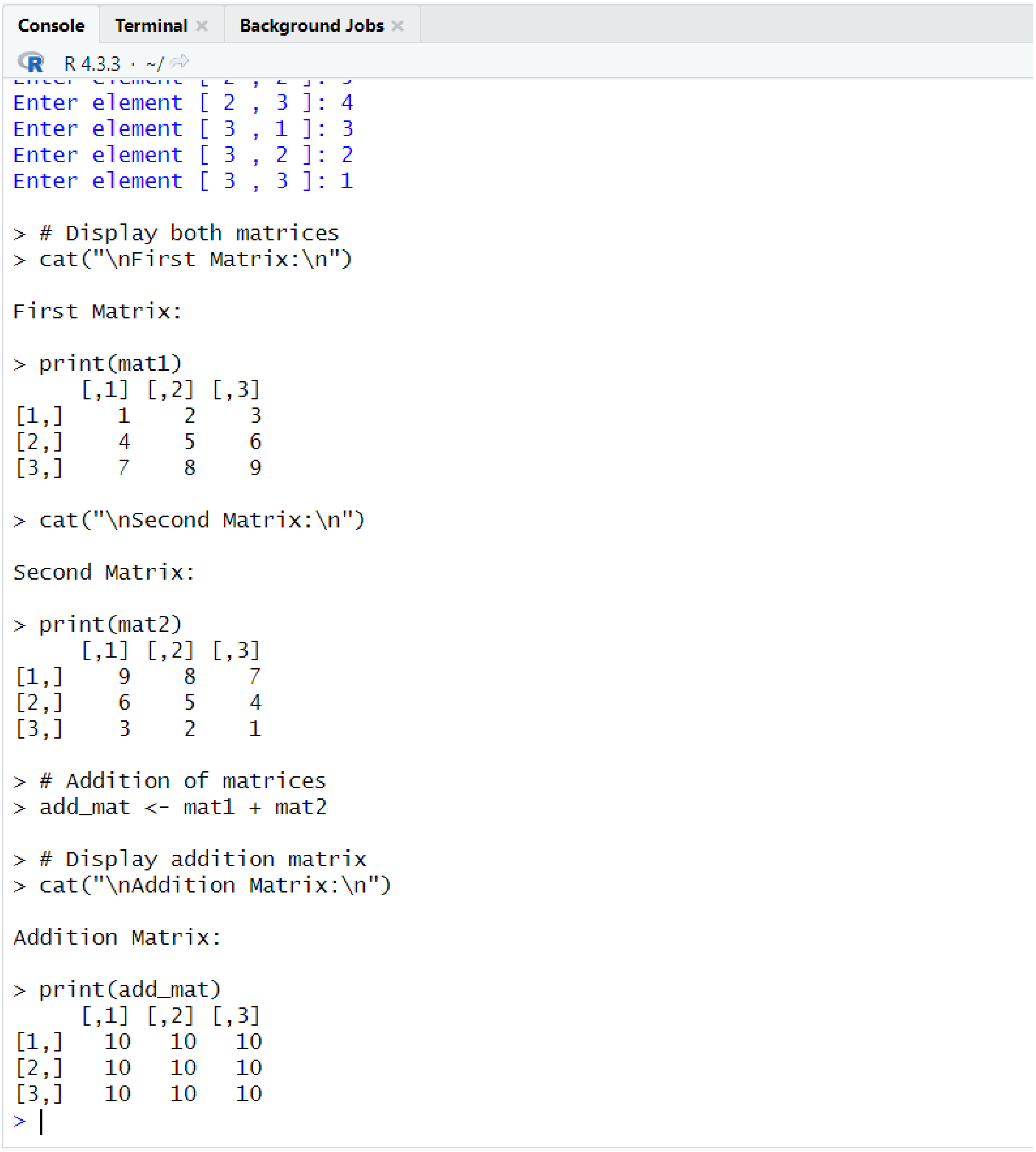
PRACTICAL-14

AIM: Input two 3x3 Matrices and display both and their addition matrix.

PROCEDURE:

```
# Input first 3x3 matrix
mat1 <- matrix(numeric(), nrow = 3, ncol = 3)
cat("Enter elements of the first 3x3 matrix:\n")
for (i in 1:3) {
for (j in 1:3) {
mat1[i, j] <- as.numeric(readline(prompt = paste("Enter element [", i, ",", j, "]: ")))
}
}
# Input second 3x3 matrix
mat2 <- matrix(numeric(), nrow = 3, ncol = 3)
cat("Enter elements of the second 3x3 matrix:\n")
for (i in 1:3) {
for (j in 1:3) {
mat2[i, j] <- as.numeric(readline(prompt = paste("Enter element [", i, ",", j, "]: ")))
}
}
# Display both matrices
cat("\nFirst Matrix:\n")
print(mat1)
cat("\nSecond Matrix:\n")
print(mat2)

# Addition of matrices
add_mat <- mat1 + mat2
# Display addition matrix
cat("\nAddition Matrix:\n")
print(add_mat)
```

INPUT/OUTPUT:



```
Enter element [ 2 , 3 ]: 4
Enter element [ 3 , 1 ]: 3
Enter element [ 3 , 2 ]: 2
Enter element [ 3 , 3 ]: 1

> # Display both matrices
> cat("\nFirst Matrix:\n")

First Matrix:

> print(mat1)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> cat("\nSecond Matrix:\n")

Second Matrix:

> print(mat2)
     [,1] [,2] [,3]
[1,]    9    8    7
[2,]    6    5    4
[3,]    3    2    1

> # Addition of matrices
> add_mat <- mat1 + mat2

> # Display addition matrix
> cat("\nAddition Matrix:\n")

Addition Matrix:

> print(add_mat)
     [,1] [,2] [,3]
[1,]   10   10   10
[2,]   10   10   10
[3,]   10   10   10
> |
```

RESULT: The Code Has Been Executed Successfully.

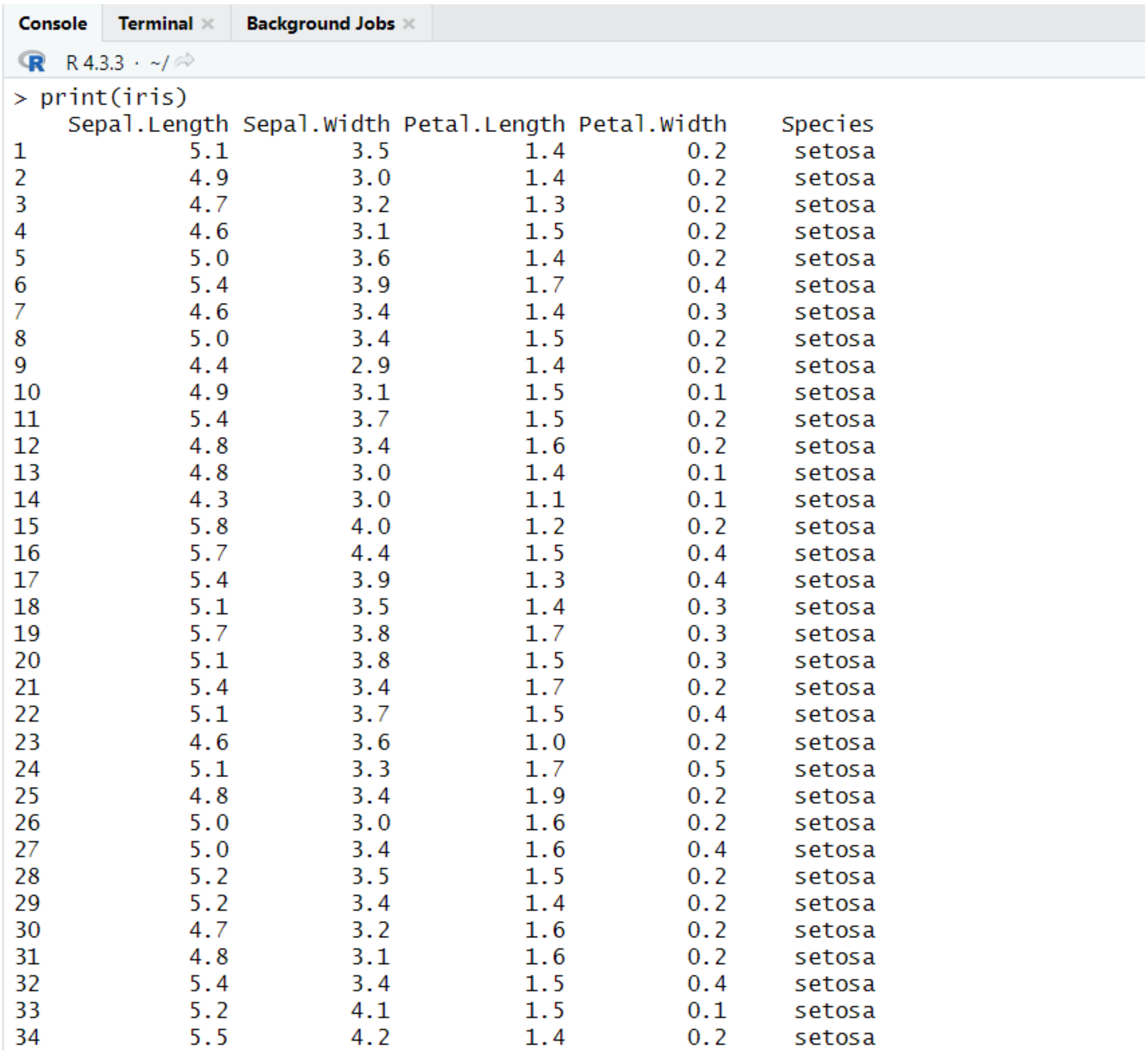
PRACTICAL -15

AIM: Import IRIS dataset and display First, 3rd and last columns only.

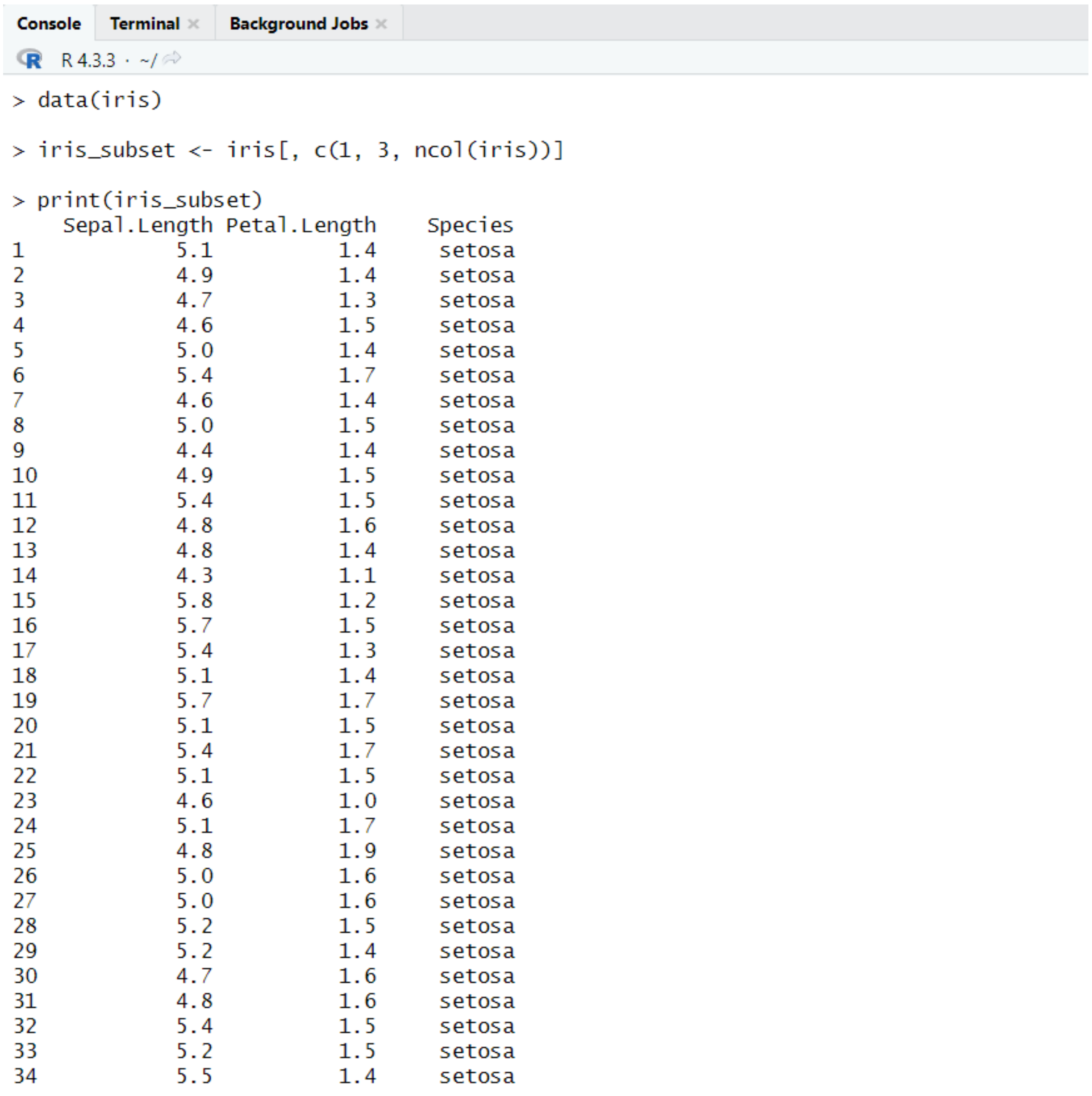
PROCEDURE:

```
# Import IRIS dataset
data(iris)
# Display all data
print(iris)
# Display first, third and last columns
iris_subset <- iris[, c(1, 3, ncol(iris))]
print(iris_subset)
```

INPUT/OUTPUT:



```
> 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
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
21           5.4         3.4          1.7         0.2     setosa
22           5.1         3.7          1.5         0.4     setosa
23           4.6         3.6          1.0         0.2     setosa
24           5.1         3.3          1.7         0.5     setosa
25           4.8         3.4          1.9         0.2     setosa
26           5.0         3.0          1.6         0.2     setosa
27           5.0         3.4          1.6         0.4     setosa
28           5.2         3.5          1.5         0.2     setosa
29           5.2         3.4          1.4         0.2     setosa
30           4.7         3.2          1.6         0.2     setosa
31           4.8         3.1          1.6         0.2     setosa
32           5.4         3.4          1.5         0.4     setosa
33           5.2         4.1          1.5         0.1     setosa
34           5.5         4.2          1.4         0.2     setosa
```



```
> data(iris)

> iris_subset <- iris[, c(1, 3, ncol(iris))]

> print(iris_subset)
    Sepal.Length Petal.Length    Species
1            5.1          1.4     setosa
2            4.9          1.4     setosa
3            4.7          1.3     setosa
4            4.6          1.5     setosa
5            5.0          1.4     setosa
6            5.4          1.7     setosa
7            4.6          1.4     setosa
8            5.0          1.5     setosa
9            4.4          1.4     setosa
10           4.9          1.5     setosa
11           5.4          1.5     setosa
12           4.8          1.6     setosa
13           4.8          1.4     setosa
14           4.3          1.1     setosa
15           5.8          1.2     setosa
16           5.7          1.5     setosa
17           5.4          1.3     setosa
18           5.1          1.4     setosa
19           5.7          1.7     setosa
20           5.1          1.5     setosa
21           5.4          1.7     setosa
22           5.1          1.5     setosa
23           4.6          1.0     setosa
24           5.1          1.7     setosa
25           4.8          1.9     setosa
26           5.0          1.6     setosa
27           5.0          1.6     setosa
28           5.2          1.5     setosa
29           5.2          1.4     setosa
30           4.7          1.6     setosa
31           4.8          1.6     setosa
32           5.4          1.5     setosa
33           5.2          1.5     setosa
34           5.5          1.4     setosa
```

<u>RESULT:</u> The Code Has Been Executed Successfully.