B.K. BIRLA COLLEGE (Autonomous) OF

ARTS, SCIENCE & COMMERCE, KALYAN

(Department of Computer Science)

CERTIFICATE

This is to certify that

Mr./Miss ______________________________

Roll no. __________ *Exam Seat No.* __________ *has satisfactorily completed the practical in* ______________________________ *as laid down in the regulation of University of Mumbai for the purpose of* ***Semester VI*** *Examination* ***2022-23***

Date:

Place:

Professor In Charge Computer Science	Head Dept. of Computer Science	Signature of Examiner

INDEX

Practical No:1

Aim:Practical of Data collection,Data curation and management for unstructured data

(NoSql)

Theory : CouchDB is an open source database developed by Apache software foundation. The focus is on the ease of use, embracing the web. It is a NoSQL document store database.

It uses JSON, to store data (documents), java script as its query language to transform the documents, http protocol for api to access the documents, query the indices with the web browser. It is a multi master application released in 2005 and it became an apache project in 2008.

Installation:

- Get the latest Windows binaries from the CouchDB web site. Old releases are available at archive.
- Follow the installation wizard steps. Be sure to install CouchDB to a path with no spaces, such as C:\CouchDB.
- Your installation is not complete. Be sure to complete the Setup steps for a single node or clustered installation.

database ---

rscript

couchdb

first

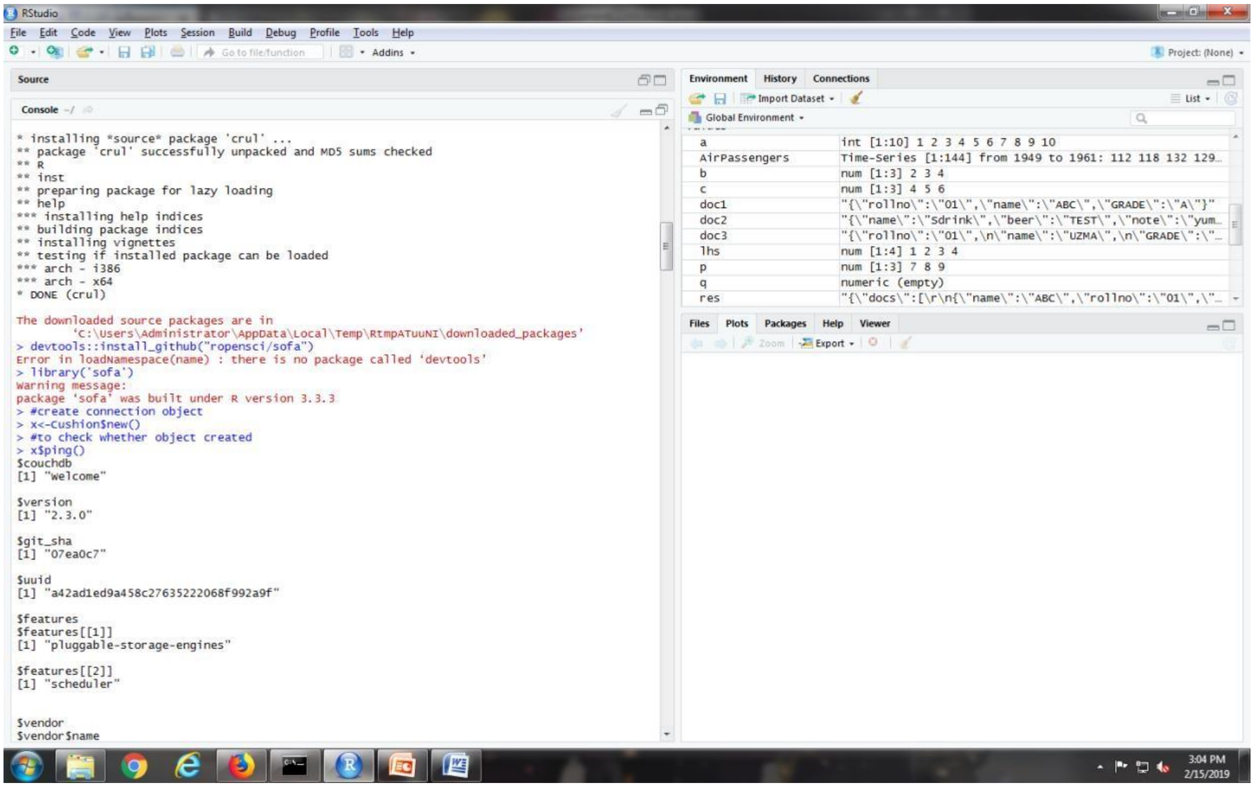
Rscript code

```
install.packages('sofa')

#devtools::install_github("

ropensci/sofa")

library('sofa')
```

```
#create
connectio
n object
x<-
Cushion$
new()
#to check
whether object
created
x$ping()
#create
database ty
db_create(
x,dbname
= 'ty')
db_list(x)
#create json doc
doc1<-'{"rollno":"01","name":"ABC","GRADE":"A"}'
doc_create(x,doc1,dbname
= "ty",docid = "a_1")
doc2<-
'{"rollno":"02","name":"P
QR","GRADE":"A"}'

doc_create(x,doc2,dbname = "ty",docid = "a_2")

doc3<-'{"rollno":"03","name":"xyz","GRADE":"B","REMARK":"PASS"}'

doc_create(x,doc3,dbname = "ty",docid = "a_3")


#CHANGES FEED

db_changes(x,"ty")


#search for id > null

so all docs will

display

db_query(x,dbname

= "ty",

   selector = list('_id'=list('$gt'=NULL)))$docs


#search for students with grade is A

db_query(x,dbname = "ty",selector = list(GRADE="A"))$docs
```

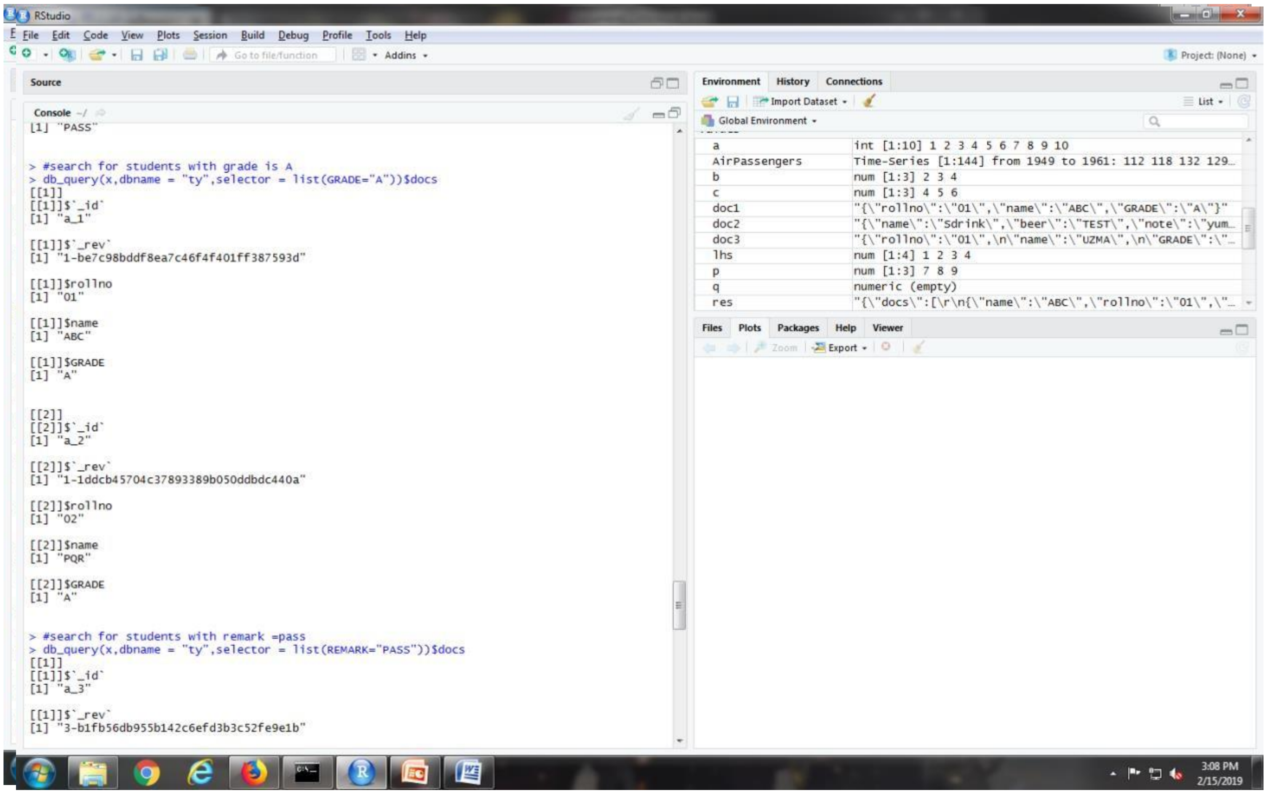
```
#search for students with remark =pass

        db_query(x,dbname = "ty",selector = list(REMARK="PASS"))$docs


        #return only certain fields where rollno>2

        db_query(x,dbname = "ty",selector =
        list(rollno=list('$gt'='02')),fields=c("name","GRADE"))$docs


        #convert the result of a query into a data frame

        using jsonlite library("jsonlite")

        res<-db_query(x,dbname = "ty",selector =
        list('_id'=list('$gt'=NULL)),fields=c("name","rollno","GRADE","REMARK"),as="json")


        #display json doc

        fromJSON(res)$do

        cs


        #doc_delete(cushion,dbnam

        e,docid)

        doc_delete(x,dbname =

        "ty",docid = "a_2")

        doc_get(x,dbname =

        "ty",docid = "a_2")


        doc2<-'{"name":"Sdrink","beer":"TEST","note":"yummy","note2":"yay"}'
```

```
doc_update(x,dbname = "ty",doc=doc2,docid="a_3",rev = "3-
b1fb56db955b142c6efd3b3c52fe9e1b")


doc3<-

'{"rollno":"01"

,

"name":"UZM

A",

"GRADE":"A"}'


doc_update(x,dbname = "ty",doc=doc3,docid = "a_1",rev = "1-be7c98bddf8ea7c46f4f401ff387593d")
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Source
Console ~/
Environment History Connections
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
Zoom
Export
3:04 PM
2/15/2019

```
* installing *source* package 'crul' ...
** package 'crul' successfully unpacked and MD5 sums checked
** R
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** installing vignettes
** testing if installed package can be loaded
*** arch - i386
*** arch - x64
* DONE (crul)

The downloaded source packages are in
        'C:\Users\Administrator\AppData\Local\Temp\RtmpATuuNI\downloaded_packages'
> devtools::install_github("ropensci/sofa")
Error in loadNamespace(name) : there is no package called 'devtools'
> library('sofa')
Warning message:
package 'sofa' was built under R version 3.3.3
> #create connection object
> x<-Cushion$new()
> #to check whether object created
> x$ping()
$couchdb
[1] "Welcome"

$version
[1] "2.3.0"

$git_sha
[1] "07ea0c7"

$uuid
[1] "a42ad1ed9a458c27635222068f992a9f"

$features
$features[[1]]
[1] "pluggable-storage-engines"

$features[[2]]
[1] "scheduler"


$vendor
$vendor$name
```

```
a               int [1:10] 1 2 3 4 5 6 7 8 9 10
AirPassengers   Time-Series [1:144] from 1949 to 1961: 112 118 132 129…
b               num [1:3] 2 3 4
c               num [1:3] 4 5 6
doc1            "{\"rollno\":\"01\",\"name\":\"ABC\",\"GRADE\":\"A\"}"
doc2            "{\"name\":\"Sdrink\",\"beer\":\"TEST\",\"note\":\"yum…
doc3            "{\"rollno\":\"01\",\n\"name\":\"UZMA\",\n\"GRADE\":\"…
lhs             num [1:4] 1 2 3 4
p               num [1:3] 7 8 9
q               numeric (empty)
res             "{\"docs\":[\r\n{\"name\":\"ABC\",\"rollno\":\"01\",\"…
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Source
Console ~/
Environment History Connections
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
Zoom
Export
3:08 PM
2/15/2019

```
[1] "PASS"


> #search for students with grade is A
> db_query(x,dbname = "ty",selector = list(GRADE="A"))$docs
[[1]]
[[1]]$`_id`
[1] "a_1"

[[1]]$`_rev`
[1] "1-be7c98bddf8ea7c46f4f401ff387593d"

[[1]]$rollno
[1] "01"

[[1]]$name
[1] "ABC"

[[1]]$GRADE
[1] "A"


[[2]]
[[2]]$`_id`
[1] "a_2"

[[2]]$`_rev`
[1] "1-1ddcb45704c37893389b050ddbdc440a"

[[2]]$rollno
[1] "02"

[[2]]$name
[1] "PQR"

[[2]]$GRADE
[1] "A"


> #search for students with remark =pass
> db_query(x,dbname = "ty",selector = list(REMARK="PASS"))$docs
[[1]]
[[1]]$`_id`
[1] "a_3"

[[1]]$`_rev`
[1] "3-b1fb56db955b142c6efd3b3c52fe9e1b"
```

```
a               int [1:10] 1 2 3 4 5 6 7 8 9 10
AirPassengers   Time-Series [1:144] from 1949 to 1961: 112 118 132 129…
b               num [1:3] 2 3 4
c               num [1:3] 4 5 6
doc1            "{\"rollno\":\"01\",\"name\":\"ABC\",\"GRADE\":\"A\"}"
doc2            "{\"name\":\"Sdrink\",\"beer\":\"TEST\",\"note\":\"yum…
doc3            "{\"rollno\":\"01\",\n\"name\":\"UZMA\",\n\"GRADE\":\"…
lhs             num [1:4] 1 2 3 4
p               num [1:3] 7 8 9
q               numeric (empty)
res             "{\"docs\":[\r\n{\"name\":\"ABC\",\"rollno\":\"01\",\"…
```

Practical No.2

Aim: Practical of data collection, Data curation and management for Large-scale Data-System(such as MongoDB)

Theory:

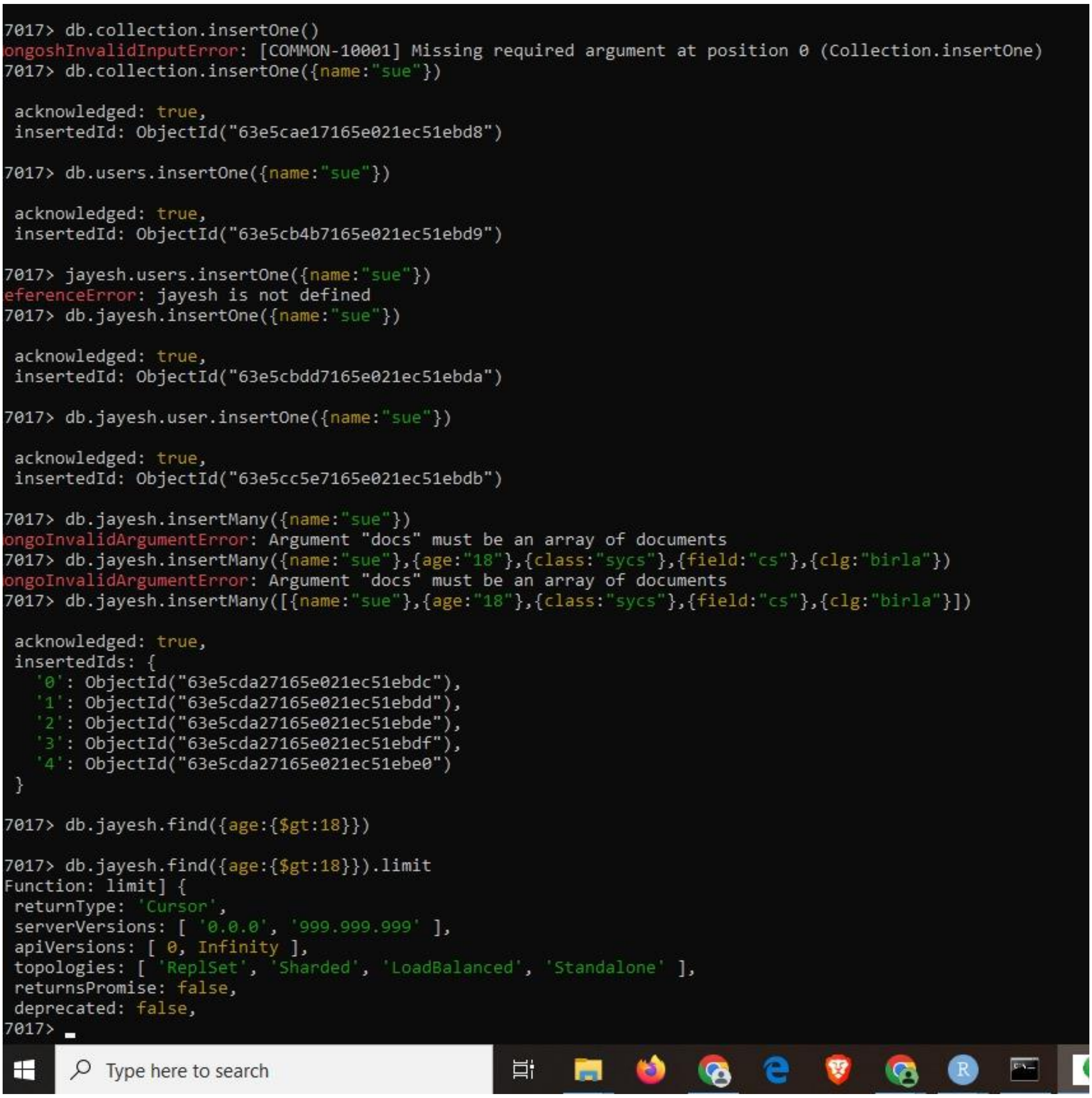
MongoDB is an open-source document-oriented database that is designed to store a large scale of data and allows you to work with that data very efficiently. It is categorized under the NoSQL(Not only SQL) database because the storage and retrieval of data in MongoDB are not in the form of tables

Installation process:

- In the Version dropdown, select the version of MongoDB to download.
- In the Platform dropdown, select Windows.
- In the Package dropdown, select msi.
- Click Download

```
7017> db.collection.insertone()

longoshInvalidInputError: [COMMON-10001] Missing required argument at position
(Collection.insertOne)

7017> db.collection.insertone ({name:"sue"})

acknowledged: true,

insertedId: ObjectId("63e5cae17165e021ec51ebd8")

7017> db.users.insertOne({name:"sue"})

acknowledged: true,

insertedId: ObjectId("63e5cb4b7165e021ec51ebd9")

7017> jayesh.users.insertOne({name:"sue"})

eferenceError: jayesh is not defined

7017> db.jayesh.insertOne({name:"sue"})

acknowledged: true,

insertedId: ObjectId("63e5cbdd7165e021ec51ebda")

7017> db.jayesh.user.insertone ({name:"sue"})

acknowledged: true,

insertedId: ObjectId("63e5cc5e7165e021ec51ebdb")
```

```
7017> db.jayesh.insertMany ({name:"sue"})

MongoInvalidArgumentError: Argument "docs" must be an array of documents

7017> db.jayesh.insertMany ({name:"sue"},{age:"18"}, {class:"sycs"},{field: "cs"},
{clg:"birla"}) MongoInvalidArgumentError: Argument "docs" must be an array of documents
7017> db.jayesh.insertMany ([{name:"sue"},{age: "18"}, {class: "sycs"},{field:
"cs"},{clg:"birla"}])

acknowledged: true, insertedIds: {

0: ObjectId("63e5cda27165e021ec51ebdc"),

'1': ObjectId("63e5cda27165e021ec51ebdd")

'2': ObjectId("63e5cda27165e021ec51ebde"

3: ObjectId("63e5cda27165e021ec51ebdf") '4': ObjectId("63e5cda27165e021ec51ebe0")

}

7017> db.jayesh.find({age: {$gt:18}})

7017> db.jayesh.find({age: {$gt:18}}).limit Function: limit] {

returnType: Cursor,

serverVersions: [ 0.0.0", "999.999.999' ],

apiVersions: [0, Infinity ],

topologies: [ReplSet, Sharded', 'LoadBalanced', 'Standalone' ], 7017>

returnsPromise: false, deprecated: false
```



```
7017> db.collection.insertOne()
ongoshInvalidInputError: [COMMON-10001] Missing required argument at position 0 (Collection.insertOne)
7017> db.collection.insertOne({name:"sue"})

 acknowledged: true,
 insertedId: ObjectId("63e5cae17165e021ec51ebd8")

7017> db.users.insertOne({name:"sue"})

 acknowledged: true,
 insertedId: ObjectId("63e5cb4b7165e021ec51ebd9")

7017> jayesh.users.insertOne({name:"sue"})
eferenceError: jayesh is not defined
7017> db.jayesh.insertOne({name:"sue"})

 acknowledged: true,
 insertedId: ObjectId("63e5cbdd7165e021ec51ebda")

7017> db.jayesh.user.insertOne({name:"sue"})

 acknowledged: true,
 insertedId: ObjectId("63e5cc5e7165e021ec51ebdb")

7017> db.jayesh.insertMany({name:"sue"})
ongoInvalidArgumentError: Argument "docs" must be an array of documents
7017> db.jayesh.insertMany({name:"sue"},{age:"18"},{class:"sycs"},{field:"cs"},{clg:"birla"})
ongoInvalidArgumentError: Argument "docs" must be an array of documents
7017> db.jayesh.insertMany([{name:"sue"},{age:"18"},{class:"sycs"},{field:"cs"},{clg:"birla"}])

 acknowledged: true,
 insertedIds: {
   '0': ObjectId("63e5cda27165e021ec51ebdc"),
   '1': ObjectId("63e5cda27165e021ec51ebdd"),
   '2': ObjectId("63e5cda27165e021ec51ebde"),
   '3': ObjectId("63e5cda27165e021ec51ebdf"),
   '4': ObjectId("63e5cda27165e021ec51ebe0")
 }

7017> db.jayesh.find({age:{$gt:18}})

7017> db.jayesh.find({age:{$gt:18}}).limit
Function: limit] {
 returnType: 'Cursor',
 serverVersions: [ '0.0.0', '999.999.999' ],
 apiVersions: [ 0, Infinity ],
 topologies: [ 'ReplSet', 'Sharded', 'LoadBalanced', 'Standalone' ],
 returnsPromise: false,
 deprecated: false,
7017> _
```

Practical No.3

AIM: Practical of Principal Component Analysis(PCA).
Theory:

PCA is a very popular method of dimensionality reduction because it provides a way to easily reduce the dimensions and is easy to understand. For this reason, PCA has been used in various applications from image compression to complex gene comparison. While using PCA, one should keep in mind its limitations well.

PCA is very sensitive to the scale of the data. It will create an initial basis in the direction of the largest variance in the data. Moreover, PCA applies a transformation over the data where all new components are orthogonal. The new features may not be interpretable in business.

Another limitation of PCA is the reliance on the mean and variance of the data. If the data has a relationship in higher dimensions such as kurtosis and skewness then PCA may not be the right technique to use on the data. In situations when the features are already orthogonal to each other and are uncorrelated, PCA will not produce any useful results except ordering the features in decreasing order of their variances.

PCA is very useful in situations when the data at hand is very large. Example, in case of image compression, PCA can be used to store the image in the first few hundred components and use less number of pixels.

We can implement the same in R programming language.

The **princomp()** function in R calculates the principal components of any data. We will also compare our results by calculating eigenvectors and eigenvalues separately. Let's use the **IRIS dataset.**

Let's start by loading the dataset.
# Taking the numeric part of the IRIS data

> data_iris <- iris[1:4]
The iris dataset having 150 observations (rows) with 4 features.
Let's use the **cov()** function to calculate the covariance matrix of the loaded iris data set.

# Calculating the covariance matrix
> Cov_data <- cov(data_iris)
The next step is to calculate the eigenvalues and eigenvectors.
We can use the **eigen()** function to do this automatically for us.

# Find out the eigenvectors and eigenvalues using the covariance matrix
> Eigen_data <- eigen(Cov_data)

We have calculated the Eigen values from the data. We will now look at the PCAfunction **princomp()** which automatically calculates these values.
Let’s calculate the components and compare the values.

# Using the inbuilt function

```
> PCA_data <- princomp(data_iris ,cor="False")
```

```
# Let’s now compare the output variances
> Eigen_data$values
```

Output:

```
[1] 4.22824171 0.24267075 0.07820950 0.02383509

> PCA_data$sdev^2
   Comp.1     Comp.2     Comp.3     Comp.4
4.20005343 0.24105294 0.07768810 0.02367619
```

There is a slight difference due to squaring in PCA_data but the outputs are more or less similar. We can also compare the eigenvectors of both models.

```
> PCA_data$loadings[,1:4]
                 Comp.1      Comp.2      Comp.3     Comp.4
Sepal.Length  0.36138659  0.65658877  0.58202985  0.3154872
Sepal.Width  -0.08452251  0.73016143 -0.59791083 -0.3197231
Petal.Length  0.85667061 -0.17337266 -0.07623608 -0.4798390
Petal.Width   0.35828920 -0.07548102 -0.54583143  0.7536574

> Eigen_data$vectors
            [,1]        [,2]        [,3]       [,4]
[1,]  0.36138659 -0.65658877 -0.58202985  0.3154872
[2,] -0.08452251 -0.73016143  0.59791083 -0.3197231
[3,]  0.85667061  0.17337266  0.07623608 -0.4798390
[4,]  0.35828920  0.07548102  0.54583143  0.7536574
```

This time the eigenvectors calculated are same and there is no difference.

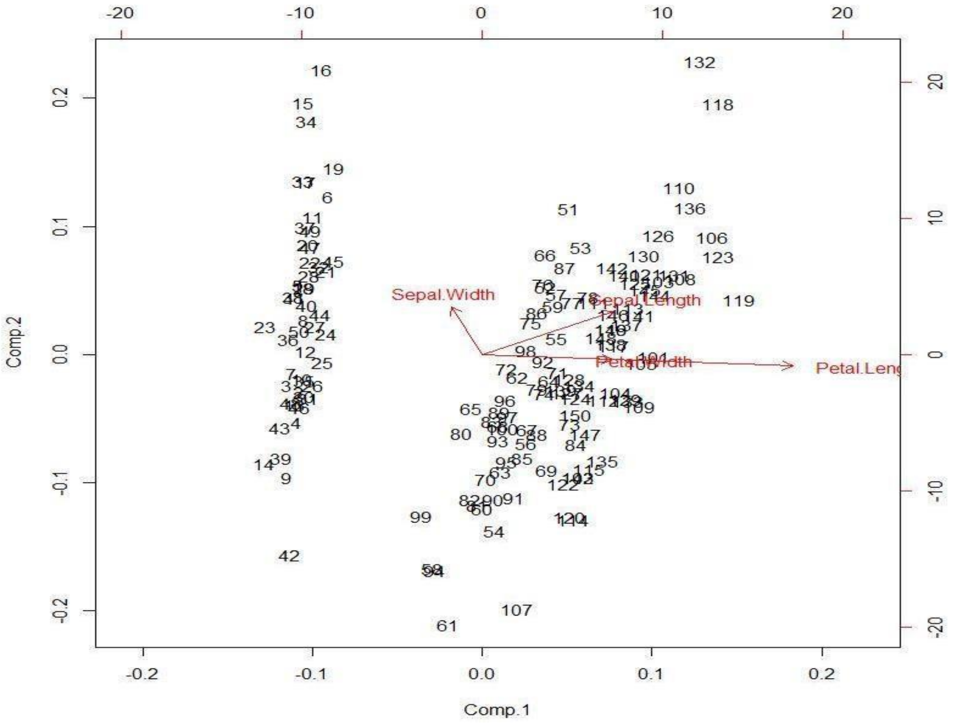
Let us now understand our model. We transformed our 4 features into 4 new orthogonal components. To know the importance of the first component, we can view the summary of the model.

```
> summary(PCA_data)
  Importance of components:
                          Comp.1     Comp.2     Comp.3      Comp.4
  Standard deviation     2.0494032 0.49097143 0.27872586 0.153870700
  Proportion of Variance 0.9246187 0.05306648 0.01710261 0.005212184
  Cumulative Proportion 0.9246187 0.97768521 0.99478782 1.000000000
```

From the Proportion of Variance, we see that the first component has an importanceof **92.5%** in predicting the class while the second principal component has an importanceof **5.3%** and so on. This means that using just the first component instead of all the 4 features will make our model accuracy to be about **92.5%** while we use only one-fourth of the entire set of features.

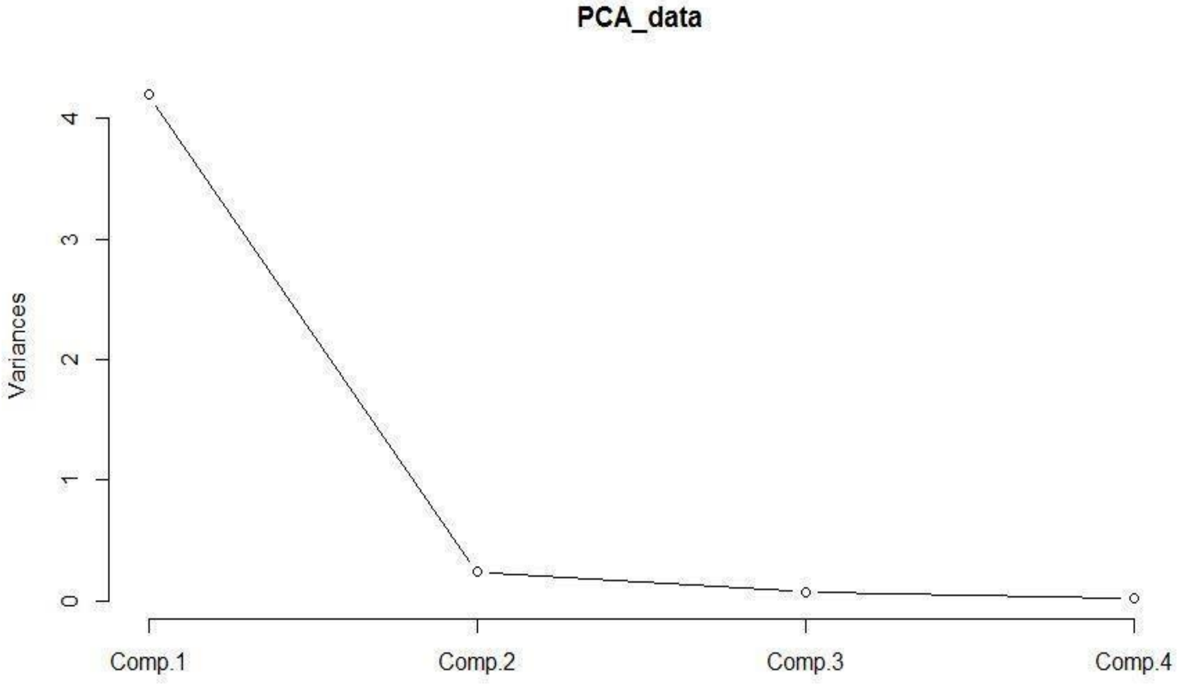
If we want the higher accuracy, we can take the first two components together and obtain a cumulative accuracy of up to **97.7%.** We can also understand how our features are transformed by using the biplot function on our model.

> biplot (PCA_data)



PCA feature transformation

> screeplot(PCA_data, type="lines")



principle components

This plot shows the bend at the second principal component.
Let us now fit two naive Bayes models.

1. one over the entire data.
2. The second on the first principal component.

We will calculate the difference in accuracy between these two models.

```
#Select the first principal component for the second model
> model2 = PCA_data$loadings[,1]


#For the second model, we need to calculate scores by multiplying our loadings with the data
> model2_scores <- as.matrix(data_iris) %*% model2

#Loading libraries for naiveBayes model
> library(class)
> install.packages("e1071")
> library(e1071)


#Fitting the first model over the entire data
> mod1<-naiveBayes(iris[,1:4], iris[,5])

#Fitting the second model using the first principal component
> mod2<-naiveBayes(model2_scores, iris[,5])

# Accuracy for the first model
>table(predict(mod1, iris[,1:4]), iris[,5])

            setosa versicolor virginica
  setosa        50          0         0
  versicolor     0         47         3
  virginica      0          3        47



# Accuracy for the second model
>table(predict(mod2, model2_scores), iris[,5])

            setosa versicolor virginica
  setosa        50          0         0
  versicolor     0         46         5
  virginica      0          4        45
```

Practical No.4

Aim: Practical of Clustering.

Theory: This dataset is very commonly used for Overview of data, Data Visualization and Clustering model. It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The given columns in this dataset are:
i> Id
ii> SepalLength (Cm)
iii>SepalWidth (Cm)
iv> PetalLength (Cm)
v> PetalWidth (Cm)
vi> Species

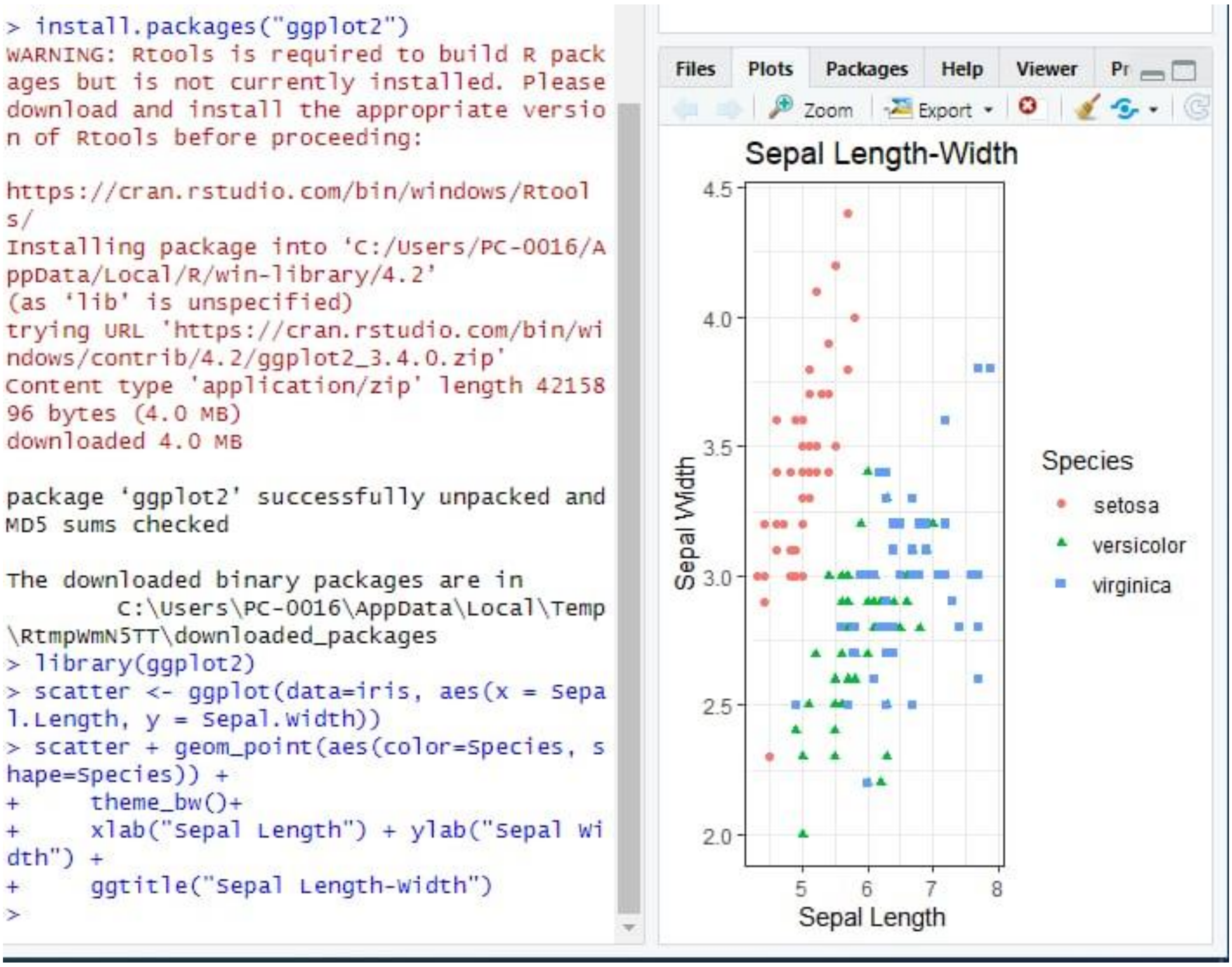
Lets visualize this dataSet and Cluster with kmeans
Solution approach –
IRIS Data, Basic Visualization before Clustering

```
> install.packages("ggplot2")
> library(ggplot2)

> scatter <- ggplot(data=iris, aes(x = Sepal.Length, y = Sepal.Width))


> scatter + geom_point(aes(color=Species, shape=Species)) +
+   theme_bw()+
+   xlab("Sepal Length") +  ylab("Sepal Width") +
+   ggtitle("Sepal Length-Width")


> ggplot(data=iris, aes(Sepal.Length, fill = Species))+
+   theme_bw()+
+   geom_density(alpha=0.25)+
+   labs(x = "Sepal.Length", title="Species vs Sepal Length")


> vol <- ggplot(data=iris, aes(x = Sepal.Length))

> vol + stat_density(aes(ymax = ..density..,  ymin = -..density..,
+                           fill = Species, color = Species),
+                       geom = "ribbon", position = "identity") +
+   facet_grid(. ~ Species) + coord_flip() + theme_bw()+labs(x = "Sepal Length", title="Spec ies vs Sepal Length")
```

```
> vol <- ggplot(data=iris, aes(x = Sepal.Width))
> vol + stat_density(aes(ymax = ..density..,  ymin = -..density..,
+                          fill = Species, color = Species),
+                      geom = "ribbon", position = "identity") +
+   facet_grid(. ~ Species) + coord_flip() + theme_bw()+labs(x = "Sepal Width", title="Speci es vs Sepal Width")
```

Clustering Data :: Method-1

```
> irisData <- iris[,1:4]
> totalwSS<-c()

# kmeans clustering for 15 times in a loop
> for (i in 1:15)
+ {
+   clusterIRIS <- kmeans(irisData, centers=i)
+   totalwSS[i]<-clusterIRIS$tot.withinss
+ }

# Scree plot - Use plot function to plot values of tot_wss against no-of-clusters
> plot(x=1:15,                    # x= No of clusters, 1 to 15
+       y=totalwSS,                # tot_wss for each
+       type="b",                  # Draw both points as also connect them
+       xlab="Number of Clusters",
+       ylab="Within groups sum-of-squares")




# Draw a histogram denoting how various indices have voted for number of clusters.
# Out of 26 indicies, most (10) voted for 2 clusters, eight voted
# for 3 clusters and remaining eight (26-10-8) for other no of clusters
# Histogram, breaks =15 as our algorithm checks from 2 to 15
clusters

> hist(nb$Best.nc[1,], breaks = 15, main="Histogram for Number of Clusters")
```



```
> install.packages("ggplot2")
WARNING: Rtools is required to build R pack
ages but is not currently installed. Please
download and install the appropriate versio
n of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtool
s/
Installing package into ‘C:/Users/PC-0016/A
ppData/Local/R/win-library/4.2’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/wi
ndows/contrib/4.2/ggplot2_3.4.0.zip'
Content type 'application/zip' length 42158
96 bytes (4.0 MB)
downloaded 4.0 MB

package ‘ggplot2’ successfully unpacked and
MD5 sums checked

The downloaded binary packages are in
        C:\Users\PC-0016\AppData\Local\Temp
\RtmpWmN5TT\downloaded_packages
> library(ggplot2)
> scatter <- ggplot(data=iris, aes(x = Sepa
l.Length, y = Sepal.Width))
> scatter + geom_point(aes(color=Species, s
hape=Species)) +
+      theme_bw()+
+      xlab("Sepal Length") + ylab("Sepal Wi
dth") +
+      ggtitle("Sepal Length-Width")
>
```

Practical No.5

Aim: Practical of Time-series forecasting.

Theory:

Making predictions about the future is called extrapolation in the classical statistical handling of time series data.

More modern fields focus on the topic and refer to it as time series forecasting.

Forecasting involves taking models fit on historical data and using them to predict future observations.

Descriptive models can borrow for the future (i.e. to smooth or remove noise), they only seek to best describe the data.

An important distinction in forecasting is that the future is completely unavailable and must only be estimated from what has already happened.

The skill of a time series forecasting model is determined by its performance at predicting the future. This is often at the expense of being able to explain why a specific prediction was made, confidence intervals and even better understanding the underlying causes behind the problem.

Exploration of Time Series Data in R:

Here we'll learn to handle time series data on R. Our scope will be restricted to data exploring in a time series type of data set and not go to building time series models.

I have used an inbuilt data set of R called AirPassengers. The dataset consists of monthly totals of international airline passengers, 1949 to 1960.

Loading the Data Set

Following is the code which will help you load the data set and spill out a few top level metrics.

```
> data(AirPassengers)

> class(AirPassengers)
[1] "ts"
#This tells you that the data series is in a time series format

> start(AirPassengers)
[1] 1949    1
#This is the start of the time series

> end(AirPassengers)
[1] 1960 12
#This is the end of the time series

> frequency(AirPassengers)
[1] 12
#The cycle of this time series is 12months in a year
> summary(AirPassengers)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  104.0 180.0 265.5 280.3 360.5 622.0
   #The number of passengers are distributed
  across the spectrum
> plot(AirPassengers)

#This will plot the time series
> abline(reg=lm(AirPassengers~time(AirPassengers)))
# This will fit in a line

> cycle(AirPassengers)



     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949   1   2   3   4   5   6   7   8   9  10  11  12
1950   1   2   3   4   5   6   7   8   9  10  11  12
1951   1   2   3   4   5   6   7   8   9  10  11  12
1952   1   2   3   4   5   6   7   8   9  10  11  12
1953   1   2   3   4   5   6   7   8   9  10  11  12
1954   1   2   3   4   5   6   7   8   9  10  11  12
1955   1   2   3   4   5   6   7   8   9  10  11  12
1956   1   2   3   4   5   6   7   8   9  10  11  12
1957   1   2   3   4   5   6   7   8   9  10  11  12
1958   1   2   3   4   5   6   7   8   9  10  11  12
1959   1   2   3   4   5   6   7   8   9  10  11  12
1960   1   2   3   4   5   6   7   8   9  10  11  12
#This will print the cycle across years.
> plot(aggregate(AirPassengers,FUN=mean))

#This will aggregate the cycles and display a year on year trend


> boxplot(AirPassengers~cycle(AirPassengers))
#Box plot across months will give us a sense on seasonal effect
```

Important Inferences

1. The year on year trend clearly shows that the #passengers have been increasing without fail.

2. The variance and the mean value in July and August is much higher than rest of the months.

3. Even though the mean value of each month is quite different their variance is small. Hence, we have strong seasonal effect with a cycle of 12 months or less.

Exploring data becomes most important in a time series model – without this exploration, you will not know whether a series is stationary or not. As in this case we already know many details about the kind of model we are looking out for.Let's now take up a few time series models and their characteristics. We will also takethis problem forward and make a few predictions.

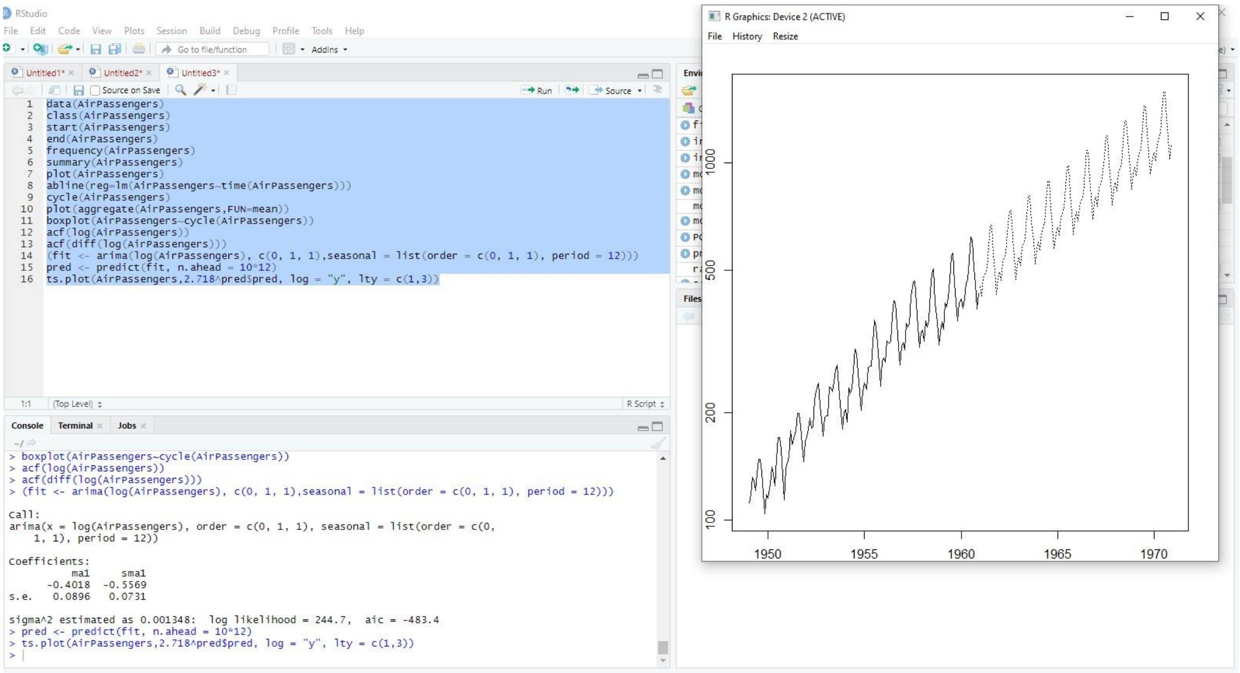
Auto – correlation Function(ACF): ACF is a plot of total correlation between different lag functions.
Following are the ACF plots for the series :

```
> acf(log(AirPassengers))
```

What do you see in the chart shown above?
Clearly, the decay of ACF chart is very slow, which means that the population is not stationary. We have already discussed above that we now intend to regress on the difference of logs rather than log directly. Let's see how ACF curve come out after regressing on the difference.

```
> acf(diff(log(AirPassengers)))
```

```
> (fit <- arima(log(AirPassengers), c(0, 1, 1),seasonal = list(order = c(0, 1, 1), period = 12)))

Call:
arima(x = log(AirPassengers), order = c(0, 1, 1), seasonal = list(order = c(0,
    1, 1), period = 12))

Coefficients:
          ma1     sma1
      -0.4018  -0.5569
s.e.   0.0896   0.0731

sigma^2 estimated as 0.001348:  log likelihood = 244.7,  aic = -483.4

> pred <- predict(fit, n.ahead = 10*12)
> ts.plot(AirPassengers,2.718^pred$pred, log = "y", lty = c(1,3))
```



```r
data(AirPassengers)
class(AirPassengers)
start(AirPassengers)
end(AirPassengers)
frequency(AirPassengers)
summary(AirPassengers)
plot(AirPassengers)
abline(reg=lm(AirPassengers~time(AirPassengers)))
cycle(AirPassengers)
plot(aggregate(AirPassengers,FUN=mean))
boxplot(AirPassengers~cycle(AirPassengers))
acf(log(AirPassengers))
acf(diff(log(AirPassengers)))
(fit <- arima(log(AirPassengers), c(0, 1, 1),seasonal = list(order = c(0, 1, 1), period = 12)))
pred <- predict(fit, n.ahead = 10*12)
ts.plot(AirPassengers,2.718^pred$pred, log = "y", lty = c(1,3))
```

```
> boxplot(AirPassengers~cycle(AirPassengers))
> acf(log(AirPassengers))
> acf(diff(log(AirPassengers)))
> (fit <- arima(log(AirPassengers), c(0, 1, 1),seasonal = list(order = c(0, 1, 1), period = 12)))

Call:
arima(x = log(AirPassengers), order = c(0, 1, 1), seasonal = list(order = c(0,
    1, 1), period = 12))

Coefficients:
          ma1     sma1
      -0.4018  -0.5569
s.e.   0.0896   0.0731

sigma^2 estimated as 0.001348:  log likelihood = 244.7,  aic = -483.4
> pred <- predict(fit, n.ahead = 10*12)
> ts.plot(AirPassengers,2.718^pred$pred, log = "y", lty = c(1,3))
> 
```

Practical No. 6

AIM: Practical of Simple/Multiple Linear Regression

Theory:

In statistics, **linear regression** is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables).

The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called **multiple linear regression**. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quintile is used.

Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis

You will have noticed on the iris dataset, that petal length and petal width are highly correlated over all species. How about running a linear regression? First of all, using the "least squares fit" function lsfit gives this:

```
  > lsfit(iris$Petal.Length, iris$Petal.Width)$coefficients
   Intercept          X
-0.3630755  0.4157554

  > plot(iris$Petal.Length, iris$Petal.Width, pch=21, bg=c("red","green3","blue")[unclass(iris$
  Species)], main="Iris Data", xlab="Petal length", ylab="Petal width")

  > abline(lsfit(iris$Petal.Length, iris$Petal.Width)$coefficients, col="black")
```

The function lsfit is a bit of a "one trick pony" and its a lot more flexible to use a linear model instead (function lm). For this example you get exactly the same thing when we model petal width depending on petal length (written as Petal.Width ~ Petal.Length in R's model syntax):

```
  > lm(Petal.Width ~ Petal.Length, data=iris)$coefficients
   (Intercept) Petal.Length
-0.3630755        0.4157554

  > plot(iris$Petal.Length, iris$Petal.Width, pch=21, bg=c("red","green3","blue")[unclass(iris$
  Species)], main="Iris Data", xlab="Petal length", ylab="Petal width")
  > abline(lm(Petal.Width ~ Petal.Length, data=iris)$coefficients, col="black")
```

(same graph again)

You get more than just that with a linear model:

```
  > summary(lm(Petal.Width ~ Petal.Length, data=iris))
```

```
  Call:
lm(formula = Petal.Width ~ Petal.Length, data = iris)
Residuals:

Min            1Q   Median      3Q      Max
-0.56515 -0.12358 -0.01898  0.13288  0.64272
Coefficients:
Estimate Std. Error t value Pr(>|t|) (Intercept)  -
0.363076   0.039762  -9.131  4.7e-16 ***
Petal.Length  0.415755   0.009582  43.387  < 2e-16 ***
---Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.2065 on 148 degrees of freedom Multiple
R-squared:  0.9271,                          Adjusted R-squared:  0.9266
F-statistic:  1882 on 1 and 148 DF,  p-value: < 2.2e-16
```

The main point about using a linear model is we can consider more complicated examples. What about the sepal length as a function of the sepal width?

```
  > plot(iris$Sepal.Width, iris$Sepal.Length, pch=21, bg=c("red","green3","blue")[unclass(iris
$Species)], main="Iris Data", xlab="Sepal Width", ylab="Sepal Length")
```

```
  > abline(lm(Sepal.Length ~ Sepal.Width, data=iris)$coefficients, col="black")
```

It very clear that the linear model Sepal.Length ~ Sepal.Width (black line) is not doing a very good job, even without looking at the statistics:

```
  > summary(lm(Sepal.Length ~ Sepal.Width, data=iris))
```

```
  Call:

lm(formula = Sepal.Length ~ Sepal.Width, data = iris)

Residuals:
Min          1Q  Median      3Q     Max
-1.5561 -0.6333 -0.1120  0.5579  2.2226


Coefficients:
```

```
Estimate Std. Error t value Pr(>|t|) (Intercept)
6.5262              0.4789  13.63  <2e-16 ***

Sepal.Width  -0.2234      0.1551   -1.44    0.152
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.8251 on 148 degrees of freedom
Multiple R-squared:  0.01382,          Adjusted R-squared:  0.007159
F-statistic: 2.074 on 1 and 148 DF,  p-value: 0.1519
```

What happens if we divide the data up by species, and run three separate linear regressions?

```
 > plot(iris$Sepal.Width, iris$Sepal.Length, pch=21, bg=c("red","green3","blue")[unclass(iris
$Species)], main="Iris Data", xlab="Petal length", ylab="Sepal length")
 > abline(lm(Sepal.Length ~ Sepal.Width, data=iris)$coefficients, col="black")
 > abline(lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="setosa"),])$coeffici
 ents, col="red")
 > abline(lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="versicolor"),])$coe
 fficients, col="green3")
 > abline(lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="virginica"),])$coeff
 icients, col="blue")
```

The coefficients doing separate per species regressions of Sepal.Length ~ Sepal.Width are:

```
 > lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="setosa"),])$coefficients
 (Intercept) Sepal.Width
2.6390012   0.6904897
 > lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="versicolor"),])$coefficient
 s
(Intercept) Sepal.Width 3.5397347
0.8650777
 > lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="virginica"),])$coefficients
 (Intercept) Sepal.Width
3.9068365   0.9015345
```

The equivalent linear model would be something like Sepal.Length ~ Petal.Length:Species + Species- 1, which gives identical coefcients (see later for why I did this):

```
 > lm(Sepal.Length ~ Sepal.Width:Species + Species - 1, data=iris)$coefficients
             Speciessetosa          Speciesversicolor          Speciesvirginica
2.6390012                            3.5397347                 3.9068365
Sepal.Width:Speciessetosa Sepal.Width:Speciesversicolor  Sepal.Width:Speciesvirginica
0.6904897                            0.8650777                 0.9015345
```

What are these new terms? Because Species is a categorical input variable (a factor in R's terminology) it can't be used directly in a linear model as they need actual numbers (a linear model is basically a matrix equation). So,the following "dummy variables" have been invented for each data point (which *are* just numbers)

Speciessetosa = 1 if Species is "setosa", 0 otherwise
Speciesversicolor = 1 if Species is "versicolor", 0 otherwise
Speciesvirginica = 1 if Species is "virginica", 0 otherwise
Sepal.Width:Speciessetosa = Sepal.Width if Species is "setosa", 0 otherwise
Sepal.Width:Speciesversicolor = Sepal.Width if Species is "versicolor", 0 otherwise
Sepal.Width:Speciesvirginica = Sepal.Width if Species is "virginica", 0 otherwiseUsing the summary command on the linear model object gives:

```
  > summary(lm(Sepal.Length ~ Sepal.Width:Species + Species - 1, data=iris))

  Call:
lm(formula = Sepal.Length ~ Sepal.Width:Species + Species - 1,
data = iris)


Residuals:
Min            1Q   Median      3Q      Max
-1.26067 -0.25861 -0.03305  0.18929  1.44917
Coefficients:

Estimate Std. Error t value Pr(>|t|) Speciessetosa       2.6390
                                0.5715   4.618 8.53e-06 ***
Speciesversicolor                3.5397     0.5580   6.343 2.74e-09 ***
Speciesvirginica                 3.9068     0.5827   6.705 4.25e-10 ***
Sepal.Width:Speciessetosa           0.6905     0.1657   4.166 5.31e-05 ***
Sepal.Width:Speciesversicolor   0.8651       0.2002   4.321 2.88e-05 ***
Sepal.Width:Speciesvirginica       0.9015     0.1948   4.628 8.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Residual standard error: 0.4397 on 144 degrees of freedom Multiple
R-squared:  0.9947,                          Adjusted R-squared:  0.9944
F-statistic:  4478 on 6 and 144 DF,  p-value: < 2.2e-16
```

Just look at those p-values! Every single term has an excellent p-value, as does the model as a whole. And the residual standard error has also been halved.
In this case, the Sepal.Length ~ Sepal.Width:Species + Species - 1 model is clearly much better than just Sepal.Length ~ Sepal.Width.

Simplify with AIC
On the other hand, what about this choice instead: Sepal.Length ~ Sepal.Width + Species. In fact, this is what the AIC (Akaike Information Criterion) step function gives you if you start with all possible interactions between sepal width and species, which is written Sepal.Length ~ Sepal.Width * Species (using a asterix instead of a plus or colon) in R:

```
  > summary(step(lm(Sepal.Length ~ Sepal.Width * Species, data=iris)))
  Start:  AIC=-240.59
Sepal.Length ~ Sepal.Width * Species
```

```
Df Sum of Sq                    RSS     AIC
- Sepal.Width:Species  2   0.15719 28.004 -243.75
<none>                            27.846 -240.59

Step:  AIC=-243.74
Sepal.Length ~ Sepal.Width + SpeciesDf Sum of Sq RSS     AIC
<none>                       28.004 -243.75
- Sepal.Width  1      10.953  38.956 -196.23
- Species        2    72.752 100.756  -55.69


Call:
lm(formula = Sepal.Length ~ Sepal.Width + Species, data = iris)


Residuals:
Min              1Q   Median       3Q      Max
-1.30711 -0.25713 -0.05325  0.19542  1.41253

Coefficients:
Estimate Std. Error t value Pr(>|t|) (Intercept)
  2.2514      0.3698 6.089 9.57e-09 *** Sepal.Width
  0.8036      0.1063 7.557 4.19e-12 *** Speciesversicolor
  1.4587      0.1121 13.012  < 2e-16 *** Speciesvirginica
  1.9468      0.1000 19.465  < 2e-16 ***
  ---

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.438 on 146 degrees of freedom
Multiple R-squared:  0.7259,                Adjusted R-squared:  0.7203
F-statistic: 128.9 on 3 and 146 DF,  p-value: < 2.2e-16
```

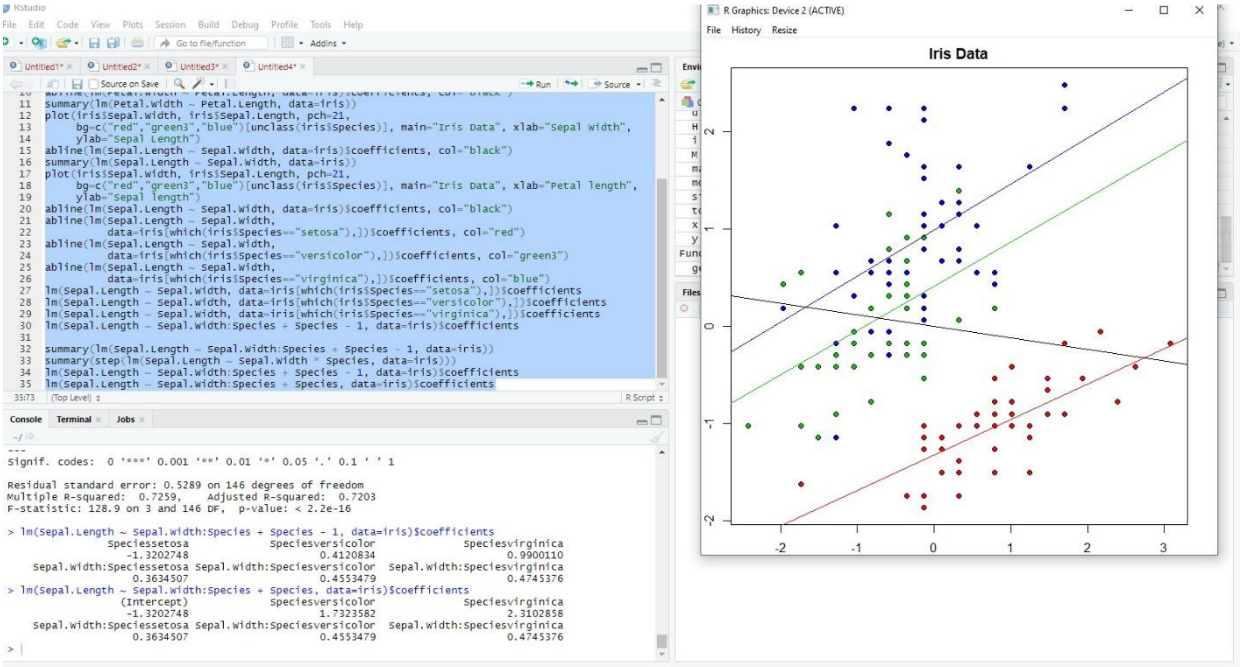
I just introduced a model of the form Sepal.Length ~ Sepal.Width:Species + Species - 1, which gave identical coefficients to those found doing species specific regressions:

```
  > lm(Sepal.Length ~ Sepal.Width:Species + Species - 1, data=iris)$coefficients
               Speciessetosa           Speciesversicolor             Speciesvirginica
2.6390012                                  3.5397347                  3.9068365
Sepal.Width:Speciessetosa Sepal.Width:Speciesversicolor  Sepal.Width:Speciesvirginica
0.6904897                                  0.8650777                  0.9015345
```

The use of the "- 1" in the model above told R not to automatically include a default intercept term. The alternative is the following:

```
  > lm(Sepal.Length ~ Sepal.Width:Species + Species, data=iris)$coefficients
                (Intercept)           Speciesversicolor             Speciesvirginica
```

```
                 2.6390012                     0.9007335                    1.2678352
Sepal.Width:Speciessetosa Sepal.Width:Speciesversicolor  Sepal.Width:Speciesvirginica
0.6904897                                        0.8650777                    0.9015345
```



```
summary(lm(Petal.Width ~ Petal.Length, data=iris))
plot(iris$Sepal.Width, iris$Sepal.Length, pch=21,
     bg=c("red","green3","blue")[unclass(iris$Species)], main="Iris Data", xlab="Sepal Width",
     ylab="Sepal Length")
abline(lm(Sepal.Length ~ Sepal.Width, data=iris)$coefficients, col="black")
summary(lm(Sepal.Length ~ Sepal.Width, data=iris))
plot(iris$Sepal.Width, iris$Sepal.Length, pch=21,
     bg=c("red","green3","blue")[unclass(iris$Species)], main="Iris Data", xlab="Petal length",
     ylab="Sepal length")
abline(lm(Sepal.Length ~ Sepal.Width, data=iris)$coefficients, col="black")
abline(lm(Sepal.Length ~ Sepal.Width,
          data=iris[which(iris$Species=="setosa"),])$coefficients, col="red")
abline(lm(Sepal.Length ~ Sepal.Width,
          data=iris[which(iris$Species=="versicolor"),])$coefficients, col="green3")
abline(lm(Sepal.Length ~ Sepal.Width,
          data=iris[which(iris$Species=="virginica"),])$coefficients, col="blue")
lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="setosa"),])$coefficients
lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="versicolor"),])$coefficients
lm(Sepal.Length ~ Sepal.Width, data=iris[which(iris$Species=="virginica"),])$coefficients
lm(Sepal.Length ~ Sepal.Width:Species + Species - 1, data=iris)$coefficients

summary(lm(Sepal.Length ~ Sepal.Width:Species + Species - 1, data=iris))
summary(step(lm(Sepal.Length ~ Sepal.Width * Species, data=iris)))
lm(Sepal.Length ~ Sepal.Width:Species + Species - 1, data=iris)$coefficients
lm(Sepal.Length ~ Sepal.Width:Species + Species, data=iris)$coefficients
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5289 on 146 degrees of freedom
Multiple R-squared:  0.7259,    Adjusted R-squared:  0.7203
F-statistic: 128.9 on 3 and 146 DF,  p-value: < 2.2e-16

> lm(Sepal.Length ~ Sepal.Width:Species + Species - 1, data=iris)$coefficients
             Speciessetosa             Speciesversicolor              Speciesvirginica
                -1.3202748                     0.4120834                     0.9900110
 Sepal.Width:Speciessetosa Sepal.Width:Speciesversicolor  Sepal.Width:Speciesvirginica
                 0.3634507                     0.4553479                     0.4745376
> lm(Sepal.Length ~ Sepal.Width:Species + Species, data=iris)$coefficients
               (Intercept)             Speciesversicolor              Speciesvirginica
                -1.3202748                     1.7323582                     2.3102858
 Sepal.Width:Speciessetosa Sepal.Width:Speciesversicolor  Sepal.Width:Speciesvirginica
                 0.3634507                     0.4553479                     0.4745376
> |
```

\

Practical No.7

Aim: Practical of Logistics Regression.

Theory:

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Data Exploration:

```
> library(datasets)
> ir_data<- iris
> head(ir_data)
  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
> str(ir_data)
'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 ...
```

So, we have a dataframe with 150 observations of 5 variables. The first 4 variables give information about plant attributes in centimeters and the last one give us the name of plant species. Species are given as Factor variable with 3 levels:

```
> levels(ir_data$Species)
[1] "setosa"              "versicolor" "virginica"
```

We should check whether we have any NA values in our dataset:

```
> sum(is.na(ir_data))
[1] 0
```

So, we are dealing with a complete dataset here. As we want to use Logistic Regression in this post, let’s subset the data so that we have to deal with 2 species of plants rather than 3 (because logistic regression will be built on binary outcomes)

```
> ir_data<-ir_data[1:100,]
```

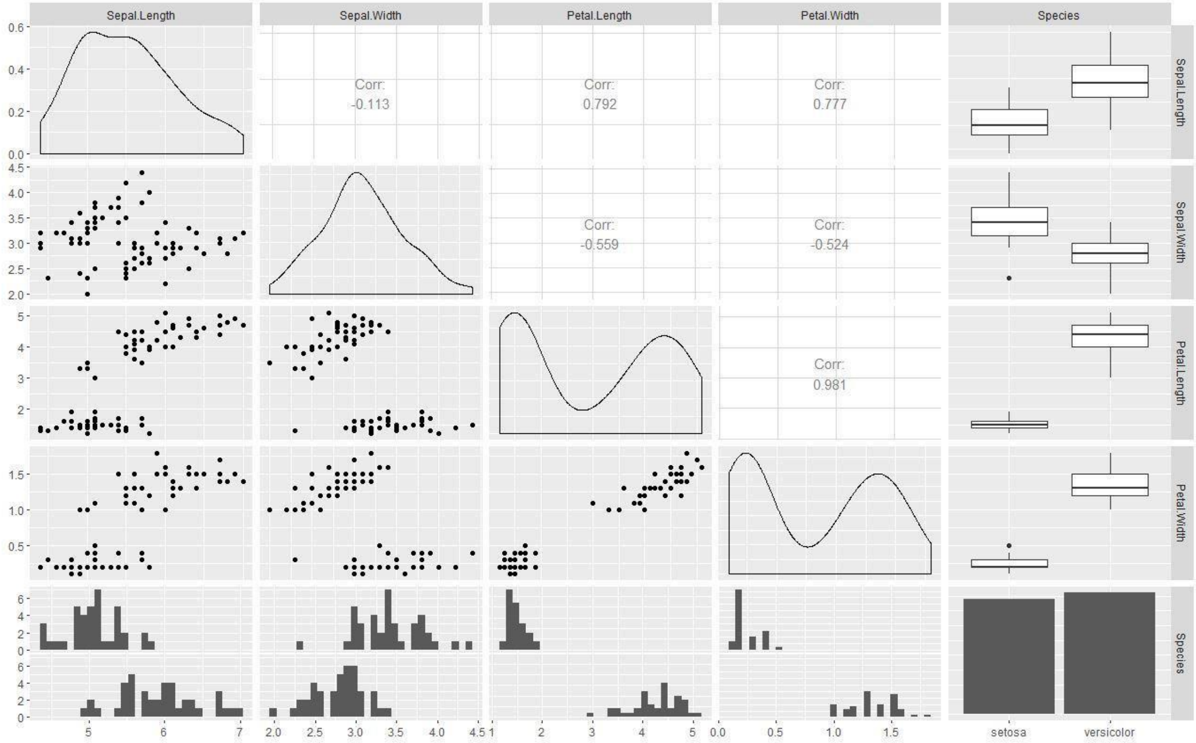
Also we will randomly define our Test and Control groups:

```
> set.seed(100)
> samp<-sample(1:100,80)

> ir_test<-ir_data[samp,]

> ir_ctrl<-ir_data[-samp,]
```

We will use the test set to create our model and control set to check our model. Now, lets explore the dataset a little bit more with the help of plots:

```
> install.packages("ggplot2")
> library(ggplot2)
> install.packages("GGally")
> library(GGally)

> ggpairs(ir_test)
```

```
    plot:  [5,1] [==========================================================
==============================================>-------------------------------------------------------------------------------- ] 84% est
: 2s `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
    plot: [5,2] [===========================================================
=====================================================>---------------] 88%
est: 2s `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
    plot: [5,3] [===========================================================
===========================================================>----------] 9
2% est: 1s `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
    plot:  [5,4] [==========================================================
==================================================================>-----]
96% est: 1s `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Model Fitting

Now, we will try to model this data using Logistic Regression. Here, we will keep it simple and will use only a single variable:

```
> y<-ir_test$Species; x<-ir_test$Sepal.Length
> glfit<-glm(y~x, family = 'binomial')
```

The default link function for above model is ‘logit’, which is what we want. We can use this simple model to get the probability of whether a given plant is ‘satosa’ or ‘versicolor’ based on its ‘sepal length’. Before jumping to predictions using, let us have a look at the model itself.

```
> summary(glfit)
```

```
Call:
glm(formula = y ~ x, family = "binomial")

Deviance Residuals:
      Min          1Q      Median           3Q         Max
-1.94538 -0.50121 0.04079 0.45923  2.26238

Coefficients:
            Estimate Std. Error z value Pr(>|z|) (Intercept)  -25.386 5.517 -
4.601 4.20e-06 ***
x                    4.675           1.017 4.596 4.31e-06 ***
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1(Dispersion parameter for

binomial family taken to be 1)

    Null deviance: 110.854 on 79 degrees of freedomResidual deviance: 56.716
on 78 degrees of freedomAIC: 60.716

Number of Fisher Scoring iterations: 6
```

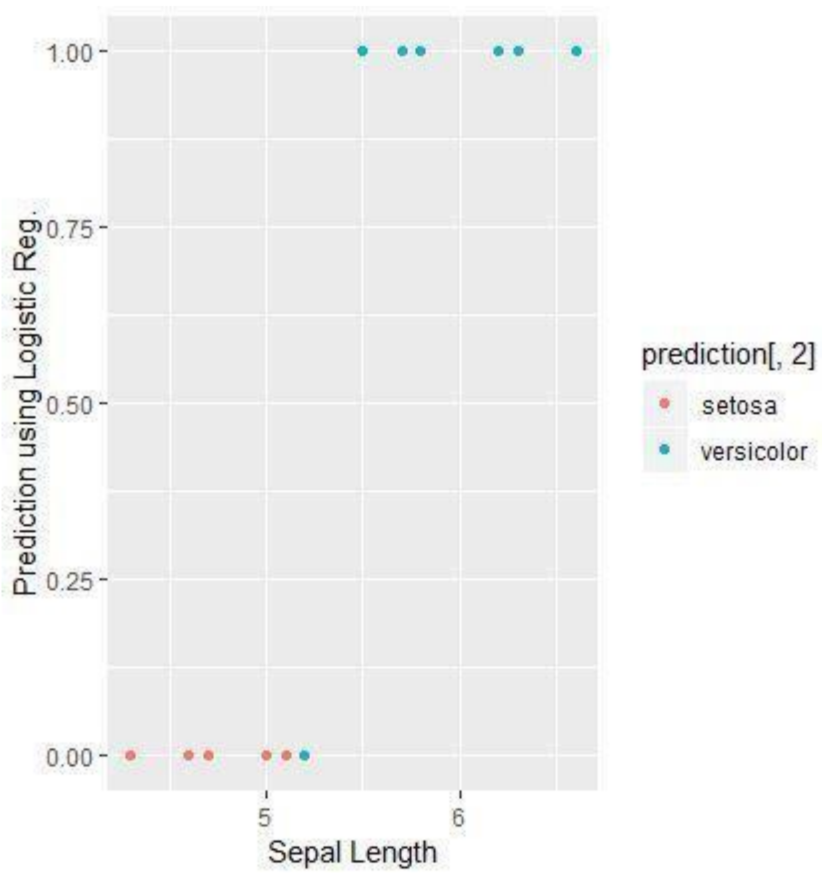
We can see that the P-Values indicate highly significant results for this model. Although, we should check any model deeply, but right now we will move to the prediction part. ###Checking Model’s Predictions Let us use our Control set which we defined earlier to predict using this model:

```
> newdata<- data.frame(x=ir_ctrl$Sepal.Length)
> predicted_val<-predict(glfit, newdata, type="response")
> prediction<-data.frame(ir_ctrl$Sepal.Length, ir_ctrl$Species,predicted_val)
> prediction
  ir_ctrl.Sepal.Length ir_ctrl.Species predicted_val
1       5.1          setosa      0.176005274
2       4.7          setosa      0.031871367
3       4.6          setosa      0.020210042
4       5.0          setosa      0.118037011
5       4.6          setosa      0.020210042
6       4.3          setosa      0.005048194
7       4.6          setosa      0.020210042
```

```
8        5.2           setosa  0.254235573
9        5.2           setosa  0.254235573
10       5.0            setosa  0.118037011
11       5.0            setosa  0.118037011
12       6.6         versicolor  0.995801728
13       5.2         versicolor  0.254235573
14       5.8         versicolor  0.849266756
15       6.2         versicolor  0.973373695
16       6.6         versicolor  0.995801728
17       5.5         versicolor  0.580872616
18       6.3         versicolor  0.983149322
19       5.7         versicolor  0.779260130
20       5.7         versicolor  0.779260130
```

We can see in the table above that what probability our model is predicting for a given plant to be ‘versicolor’ based on its ‘sepal length’. Let’s visualize this result using a simple plot. Let’s say that we will consider any plant to be ‘versicolor’ if its probability for the same is more than 0.5:

```
  > qplot(prediction[,1], round(prediction[,3]), col=prediction[,2], xlab = 'Sepal Length', ylab
+            = 'Prediction using Logistic Reg.')
```



So, from the above plot, we can see that our simple model is doing a fairly good prediction for plant species. We can also see a blue dot in the bottom cluster. This blue dot is showing that although correct specie of this plant is ‘versicolor’ but our model is predicting it as ‘setosa’.

Practical No.8

Aim: Practical of Hypothesis testing.

Theory:

Hypothesis Tests, or Statistical Hypothesis Testing, is a technique used to compare two datasets, or a sample from a dataset. It is a statistical inference method so, in the end of the test, you'll draw a conclusion—you'll infer something—about the characteristics of what you're comparing.

A hypothesis test is usually composed by

- *Null Hypothesis* (H0, read "H zero"): states that all things remain equal. No phenomena is observed or there is not relationship between what you are comparing;
- *Alternative Hypothesis* (H1, read "H one"): states the opposite of the Null Hypothesis. That there was some change, or observed relationship between what you are comparing.

Hypothesis Testing with R:

Hypothesis tests for population means are done in R using the command "t.test".

One-sample hypothesis test :

Let x represents a sample collected from a normal population with unknown mean and standard deviation. We want to test if the population mean is equal to 9, at significance level 5%.

The hypotheses are:

```
> x= c(6.2, 6.6, 7.1, 7.4, 7.6, 7.9, 8, 8.3, 8.4, 8.5, 8.6,
+         8.8, 8.8, 9.1, 9.2, 9.4, 9.4, 9.7, 9.9, 10.2, 10.4, 10.8,
+         11.3, 11.9)                              #Entering the data
```

```
> t.test(x-9,alternative="two.sided",conf.level=0.95)
#Performing the t-test
```

```
            One Sample t-test

data: x - 9
t = -0.35687, df = 23, p-value = 0.7244
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  -0.7079827 0.4996494
sample estimates:
  mean of x
-0.1041667
```

Interpretation of the result:

The P-value (0.3622) is greater than the significance level 5% (1-0.95), so we conclude that the null hypothesis that the mean of this population is 9 is plausible.

Two-sample hypothesis test:

If we are interested in finding the confidence interval for the difference of two population means, the R-command "t.test" is also to be used.

We are interested in testing observations middle range and higher viscosity are from populations with different means, at significance level 5%.

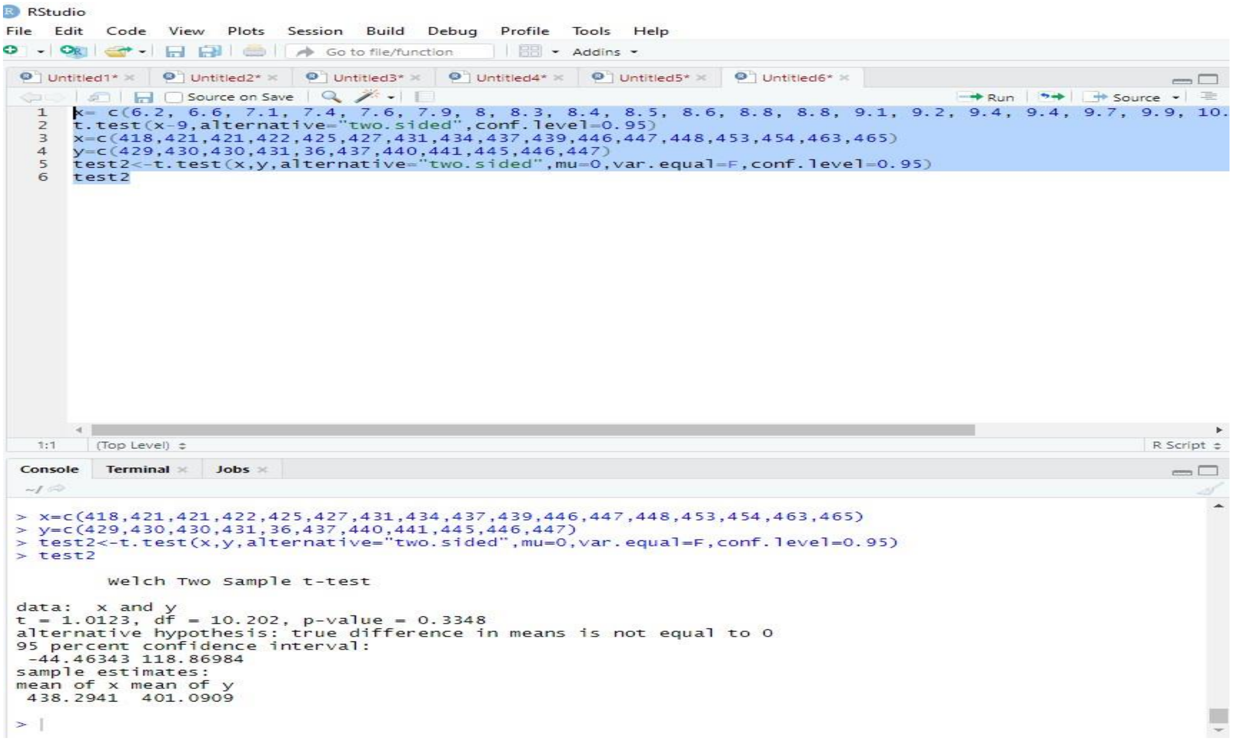
The hypotheses are:

```
> x=c(418,421,421,422,425,427,431,434,437,439,446,447,448,453,454,463,465)
  # Entering the data into the R-workspace
> y=c(429,430,430,431,36,437,440,441,445,446,447)
 > test2<-t.test(x,y,alternative="two.sided",mu=0,var.equal=F,conf.level=0.95)
 > test2                    #performing a t-test procedure, containing a confidence
                             interval computation
```

```
                Welch Two Sample t-test

data: x and y
t = 1.0123, df = 10.202, p-value = 0.3348
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  -44.46343 118.86984
sample estimates:
 mean of x mean of y
  438.2941 401.0909
```

Interpretation of the result:

The P-value (0.3348) is greater than the significance level 5% (1-0.95), so we conclude that the null hypothesis that the population means are equal is plausible.



```
x= c(6.2, 6.6, 7.1, 7.4, 7.6, 7.9, 8, 8.3, 8.4, 8.5, 8.6, 8.8, 8.8, 9.1, 9.2, 9.4, 9.4, 9.7, 9.9, 10.
t.test(x-9,alternative="two.sided",conf.level=0.95)
x=c(418,421,421,422,425,427,431,434,437,439,446,447,448,453,454,463,465)
y=c(429,430,430,431,36,437,440,441,445,446,447)
test2<-t.test(x,y,alternative="two.sided",mu=0,var.equal=F,conf.level=0.95)
test2
```

```
> x=c(418,421,421,422,425,427,431,434,437,439,446,447,448,453,454,463,465)
> y=c(429,430,430,431,36,437,440,441,445,446,447)
> test2<-t.test(x,y,alternative="two.sided",mu=0,var.equal=F,conf.level=0.95)
> test2

        Welch Two Sample t-test

data:  x and y
t = 1.0123, df = 10.202, p-value = 0.3348
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -44.46343 118.86984
sample estimates:
mean of x mean of y
 438.2941  401.0909

> |
```

Practical No.9

AIM: Practical of Analysis of Variance.

Theory:

Analysis of variance (ANOVA) is a collection of statistical models and their associated estimation procedures (such as the "variation" among and between groups) used to analyse the differences among group means in a sample.

ANOVA was developed by statistician and evolutionary biologist Ronald Fisher. In the ANOVA setting, the observed variance in a particular variable is partitioned into components attributable to different sources of variation. In its simplest form, ANOVA provides a statistical test of whether the population means of several groups are equal, and therefore generalizes the t-test to more than two groups.

ANOVA is useful for comparing (testing) three or more group means for statistical significance. It is conceptually similar to multiple two-sample t-tests, but is more conservative, resulting in fewer type I errors, and is therefore suited to a wide range of practical problems.

An Example of ANOVA using R:

There are three groups with seven observations per group. We denote group i values by yi:

```
> y1 = c(18.2, 20.1, 17.6, 16.8, 18.8, 19.7, 19.1)
> y2 = c(17.4, 18.7, 19.1, 16.4, 15.9, 18.4, 17.7)
> y3 = c(15.2, 18.8, 17.7, 16.5, 15.9, 17.1, 16.7)
```

Now we combine them into one long vector, with a second vector, group, identifying group membership:

```
> y = c(y1, y2, y3)
> n = rep(7, 3)
  > n
[1] 7 7 7
  > group = rep(1:3, n)
  > group
   [1] 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3
```

Here are summaries by group and for the combined data. First we show stem-leaf diagrams.

```
  > tmp = tapply(y, group, stem)
```

```
    The decimal point is at the |16 | 8
    17 | 6
    18 | 28
    19 | 17
    20 | 1


    The decimal point is at the |
    15 | 9
    16 | 4
    17 | 47
    18 | 47
    19 | 1


    The decimal point is at the |


    15 | 29
    16 | 57
    17 | 17
    18 | 8

  > stem(y)

    The decimal point is at the |15 | 299
    16 | 4578
    17 | 14677
    18 | 24788
    19 | 117
    20 | 1
```

Now we show summary statistics by group and overall. We locally define a temporary function, tmpfn, to make this easier.

```
  > tmpfn = function(x) c(sum = sum(x), mean = mean(x), var = var(x),
+                          n = length(x))
  > tapply(y, group, tmpfn)
$`1`
         sum          mean           var          n
  130.300000  18.614286  1.358095    7.000000

  $`2`
         sum       mean       var        n
  123.600000   17.657143 1.409524    7.000000

  $`3`
         sum       mean       var        n
  117.900000   16.842857 1.392857    7.000000

  > tmpfn(y)
         sum       mean       var        n
  371.800000   17.704762 1.798476   21.000000
```

While we could show you how to use R to mimic the computation of SS by hand, it is more natural to go directly to the ANOVA table.

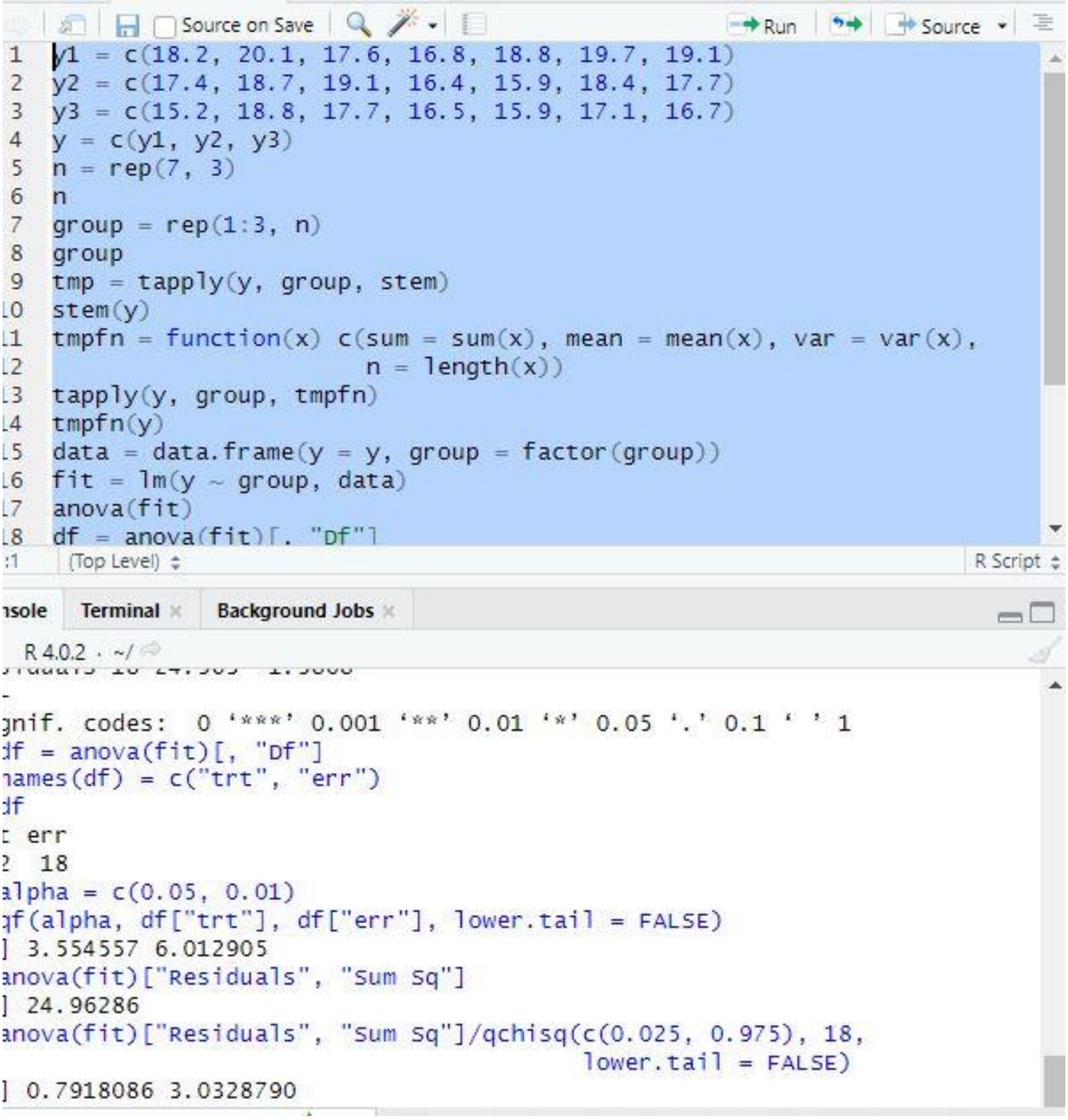
```
  > data = data.frame(y = y, group = factor(group))
  > fit = lm(y ~ group, data)
  > anova(fit)
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
group        2 11.007 5.5033 3.9683 0.03735 *
Residuals 18 24.963 1.3868
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The anova(fit) object can be used for other computations on the handout and in class. For instance, the tabled F values can be found by the following. First we extract the treatment and error degrees of freedom. Then we use qt to get the tabled F values.

```
  > df = anova(fit)[, "Df"]
  > names(df) = c("trt", "err")
  > df
  trt err
  2 18
  > alpha = c(0.05, 0.01)
  > qf(alpha, df["trt"], df["err"], lower.tail = FALSE)
  [1] 3.554557 6.012905
```

A confidence interval on the pooled variance can be computed as well using the anova(fit) object. First we get the residual sum of squares, SSTrt, then we divide by the appropriate chi-square tabled values.

```
  > anova(fit)["Residuals", "Sum Sq"]
  [1] 24.96286

  > anova(fit)["Residuals", "Sum Sq"]/qchisq(c(0.025, 0.975), 18,
+                                      lower.tail = FALSE)
[1] 0.7918086 3.0328790
```



```
y1 = c(18.2, 20.1, 17.6, 16.8, 18.8, 19.7, 19.1)
y2 = c(17.4, 18.7, 19.1, 16.4, 15.9, 18.4, 17.7)
y3 = c(15.2, 18.8, 17.7, 16.5, 15.9, 17.1, 16.7)
y = c(y1, y2, y3)
n = rep(7, 3)
n
group = rep(1:3, n)
group
tmp = tapply(y, group, stem)
stem(y)
tmpfn = function(x) c(sum = sum(x), mean = mean(x), var = var(x),
                      n = length(x))
tapply(y, group, tmpfn)
tmpfn(y)
data = data.frame(y = y, group = factor(group))
fit = lm(y ~ group, data)
anova(fit)
df = anova(fit)[, "Df"]
```

```
-
gnif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
df = anova(fit)[, "Df"]
names(df) = c("trt", "err")
df
t err
2  18
alpha = c(0.05, 0.01)
qf(alpha, df["trt"], df["err"], lower.tail = FALSE)
] 3.554557 6.012905
anova(fit)["Residuals", "Sum Sq"]
] 24.96286
anova(fit)["Residuals", "Sum Sq"]/qchisq(c(0.025, 0.975), 18,
                                        lower.tail = FALSE)
] 0.7918086 3.0328790
```

Practical No. 10

Aim: Practical of Decision Tree.

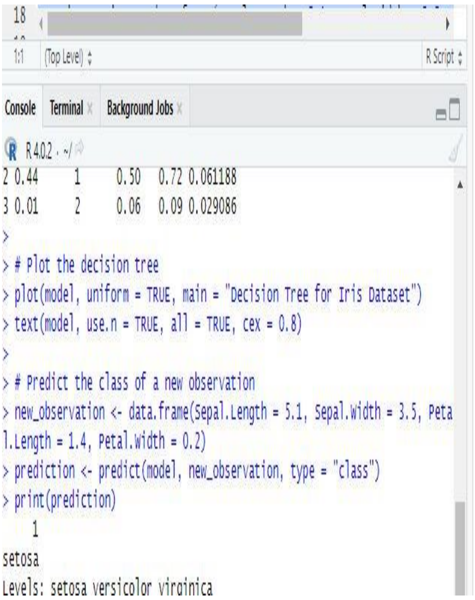
Theory: A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

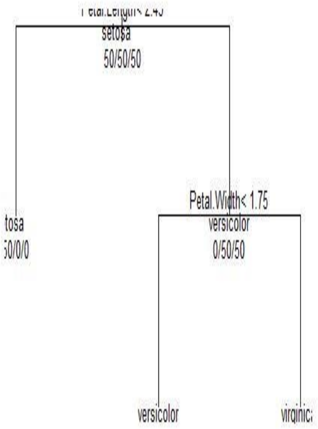
- Used for classifying data by partitioning attribute space.
- Tries to find axis-parallel decision boundaries for specified optimality criteria.
- Leaf nodes contain class labels, representing classification decisions.
- Keeps splitting nodes based on split criterion, such as GINI index, information gain orentropy.
- Pruning necessary to avoid over fitting.

Decision Tree using R:

```
# Load the required library
library(rpart)

# Load the iris dataset
data(iris)

# Fit a decision tree model to the data
model <- rpart(Species ~ ., data = iris, method = "class")

# Print a summary of the model
printcp(model)

# Plot the decision tree
plot(model, uniform = TRUE, main = "Decision Tree for Iris Dataset")
text(model, use.n = TRUE, all = TRUE, cex = 0.8)

# Predict the class of a new observation
new_observation <- data.frame(Sepal.Length = 5.1, Sepal.Width = 3.5, Petal.Length = 1.4,
Petal.Width = 0.2)
prediction <- predict(model, new_observation, type = "class")
print(prediction)
```



```
2 0.44      1      0.50   0.72 0.061188
3 0.01      2      0.06   0.09 0.029086
>
> # Plot the decision tree
> plot(model, uniform = TRUE, main = "Decision Tree for Iris Dataset")
> text(model, use.n = TRUE, all = TRUE, cex = 0.8)
>
> # Predict the class of a new observation
> new_observation <- data.frame(Sepal.Length = 5.1, Sepal.Width = 3.5, Peta
l.Length = 1.4, Petal.Width = 0.2)
> prediction <- predict(model, new_observation, type = "class")
> print(prediction)
     1
setosa
Levels: setosa versicolor virginica
```


Petal.Length< 2.45
setosa
50/50/50
tosa
50/0/0
Petal.Width< 1.75
versicolor
0/50/50
versicolor
virginica