**Practical No. 1**

**Aim :-** Practical of Data collection, Data curation and management for Unstructured data (NoSQL) (**CouchDB**)

http://localhost:5984/\_utils/#login

**Code: -**

install.packages('sofa')

library('sofa')

x <- Cushion$new(user="ckt",pwd="ckt")

x$ping()

db\_create(x,dbname='ty')

db\_list(x)

doc1 <- '{"RollNo.":"01","Name":"ABC","GRADE":"A"}'

doc\_create(x,doc1,dbname="ty",docid="a\_1")

doc2 <- '{"RollNo.":"02","Name":"PQR","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")

db\_changes(x,"ty")

db\_query(x,dbname="ty",selector=list('\_id'=list('$gt'=NULL)))$docs

db\_query(x,dbname="ty",selector=list(GRADE="A"))$docs

db\_query(x,dbname="ty",selector=list(REMARK="PASS"))$docs

db\_query(x,dbname="ty",selector=list(rollno=list('$gt'='02')),fields=c("Name","GRADE"))$docs

library("jsonlite")

res <- db\_query(x,dbname="ty",selector=list('\_id'=list('$gt'=NULL)),fields=c("Name","RollNo.","GRADE","REMARK"),as="json")

fromJSON(res)$docs

doc\_delete(x,dbname = "ty",docid="a\_2")

doc\_get(x,dbname="ty",docid = "a\_2")

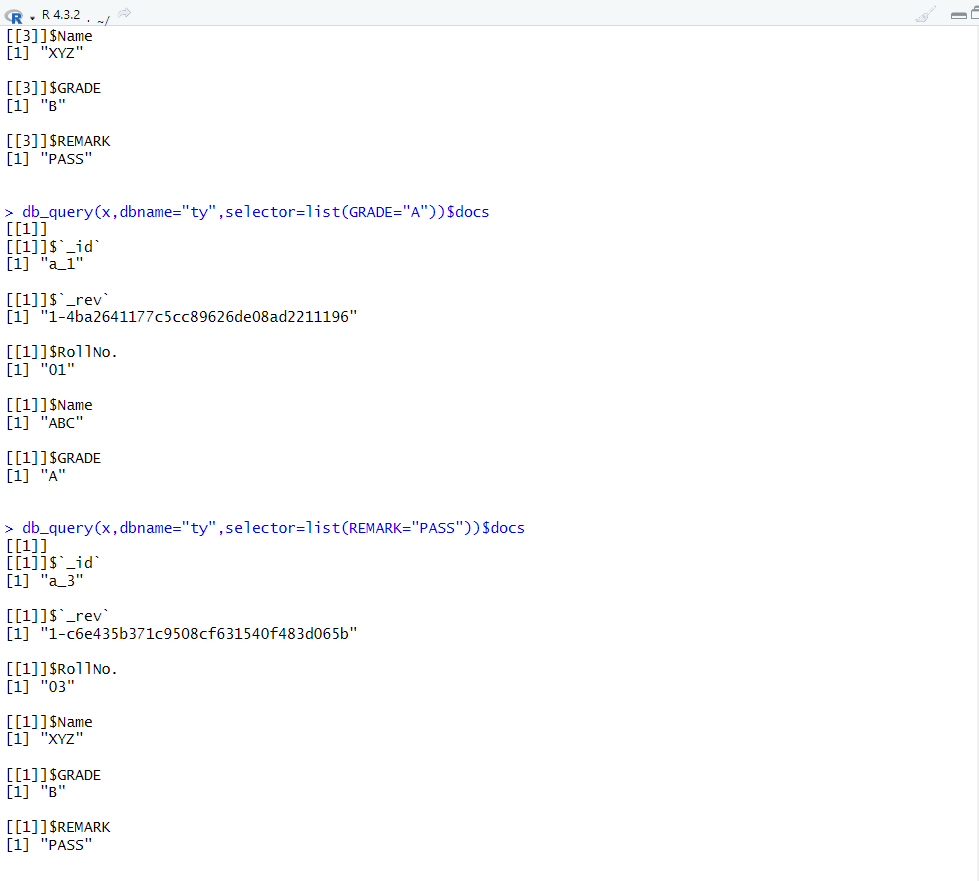
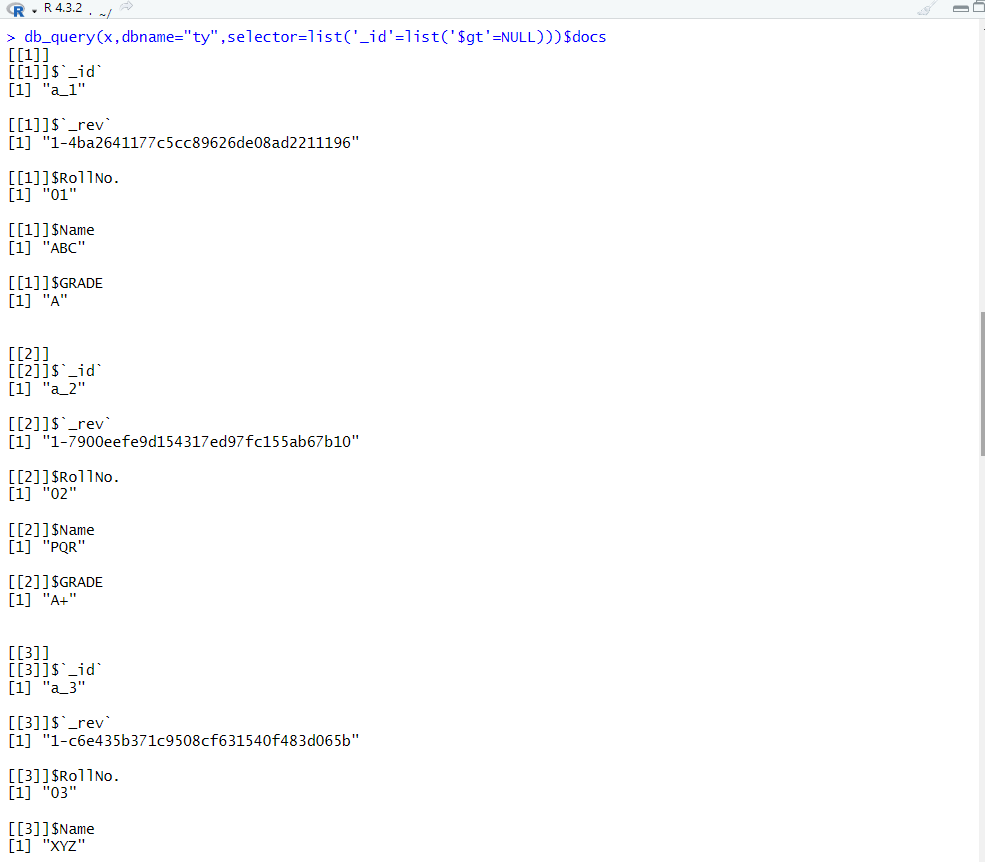
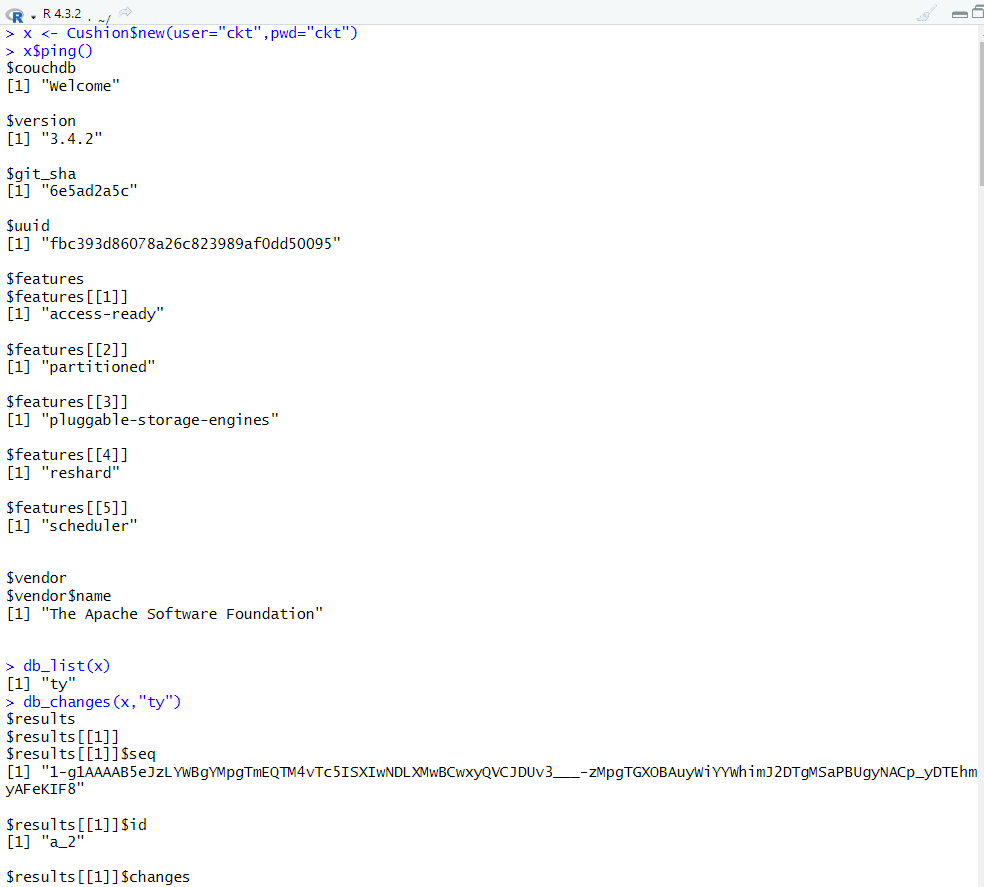
doc2 <- '{"Name":"SDrink","beer":"TEST","note":"yummy","note2:"yay"}'

doc\_update(x,dbname = "ty",doc=doc2,docid = "a\_3",rev = "1-c6e435b371c9508cf631540f483d065b")

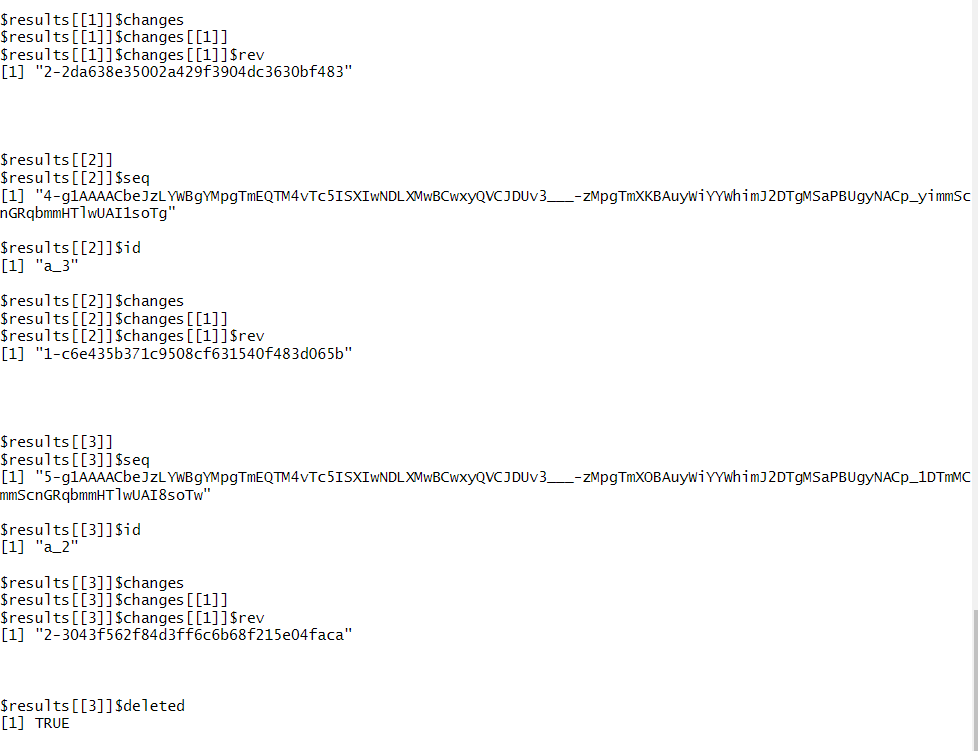
doc3 <- '{"RollNo.":"01","Name":"Pranav","GRADE":"0"}'

doc\_update(x,dbname="ty",doc=doc3,docid = "a\_1",rev="1-4ba2641177c5cc89626de08ad2211196")

**Output :-**







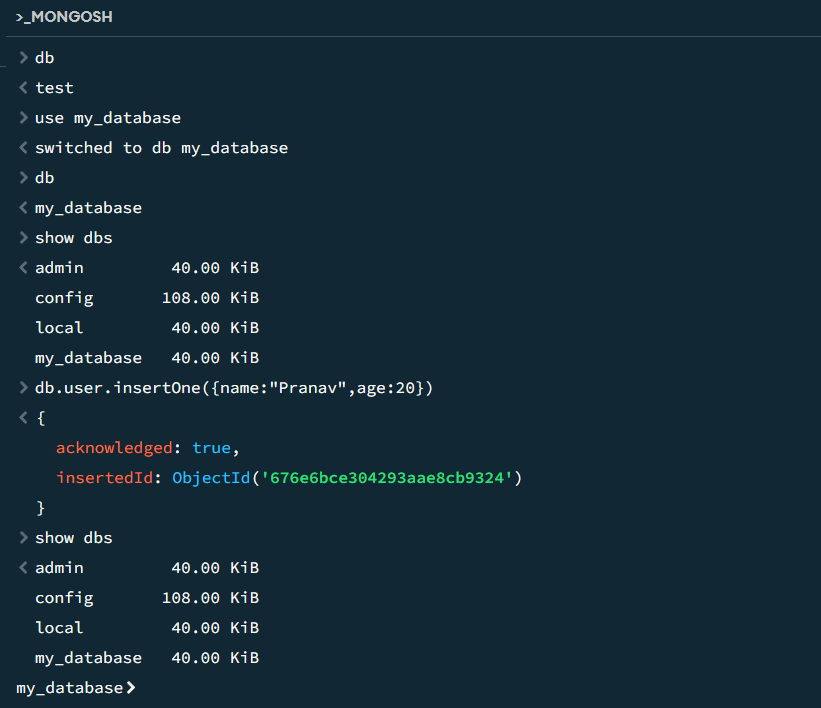
**Practical No. 2**

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

**Code: -**

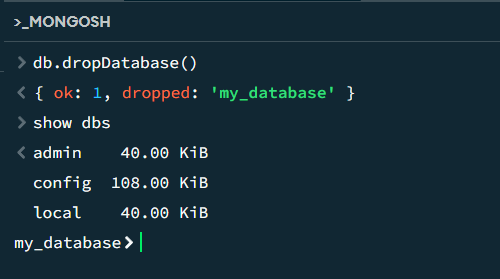
1. **Create Database:**

Command: use database\_name

****

1. **MongoDB Drop Database:**

Command : db.dropDatabase()



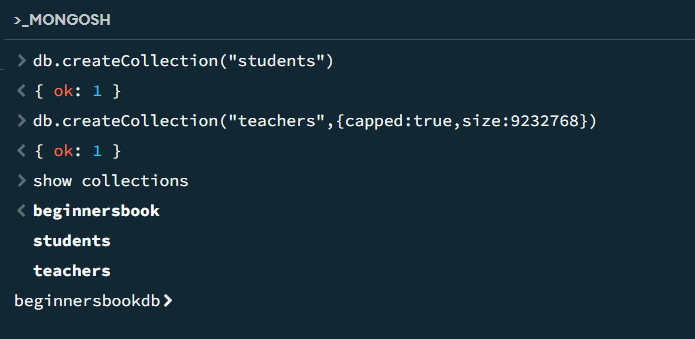
1. **MongoDB Create collection:**
2. Method 1: Creating the Collection in MongoDB on the fly :

Syntax: db.collection\_name.insert({key:value, key:value…})



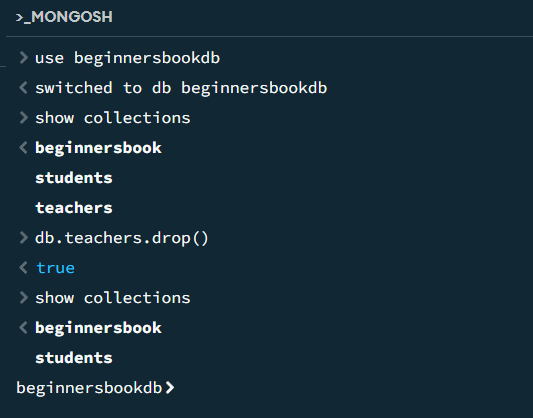
1. Method 2: Creating collection with options before inserting the documents:

Syntax: db.createCollection(name, options)



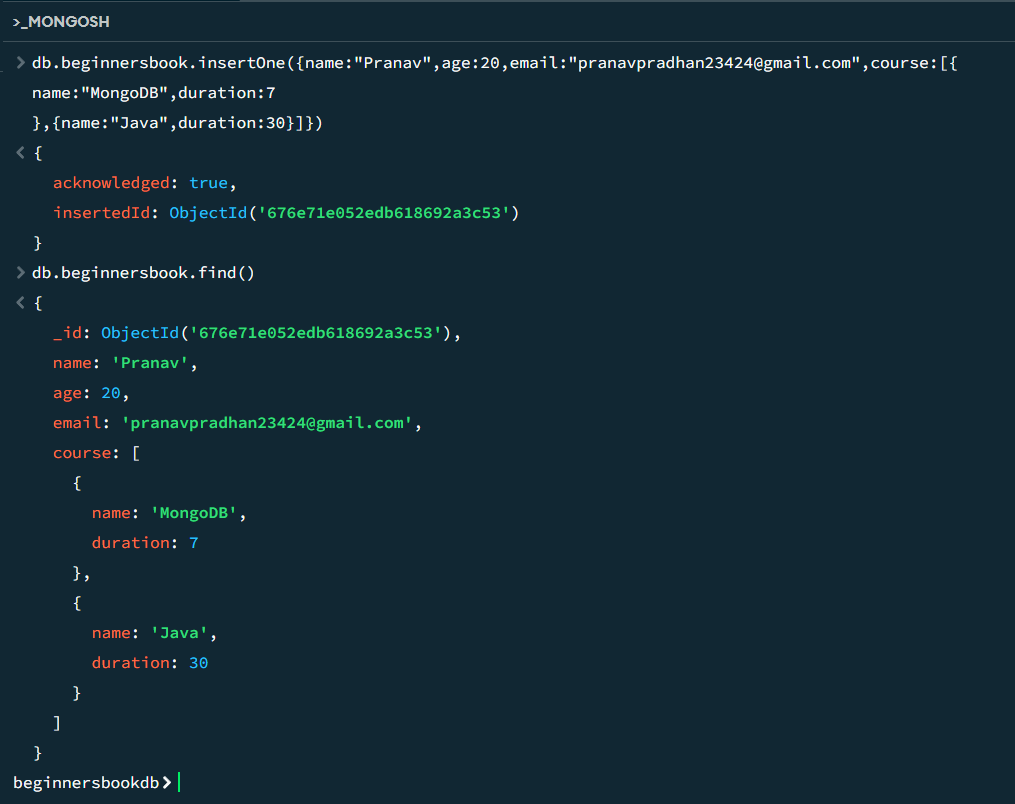
**MongoDB Drop collection:**

Syntax: db.collection\_name.drop()



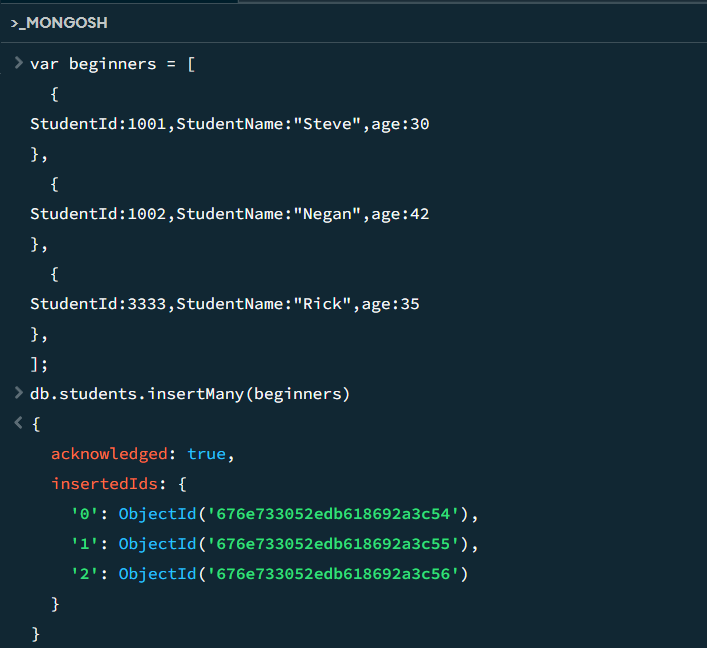
1. **MongoDB Insert Document:**
2. **Single document insertion:**

Syntax: db.collection\_name.insertOne()



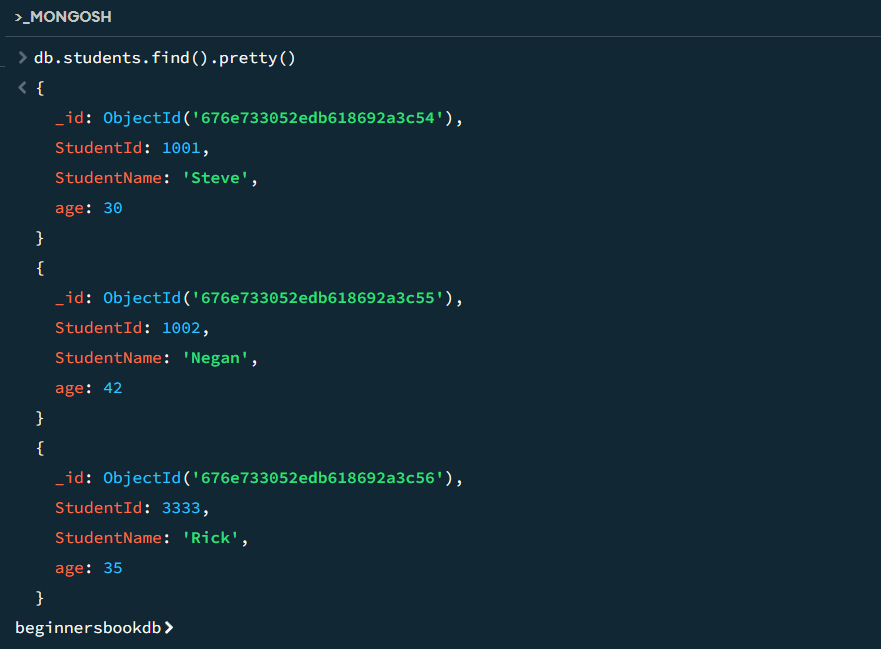
1. **Multiple document insertion:**

Syntax: db.collection\_name.insertMany()



1. **MongoDB Query Document:**
2. **Querying all the documents in JSON format:**

Syntax: db.collection\_name.find().pretty()



1. **Querying Document based on the criteria:**

* Equality Criteria:



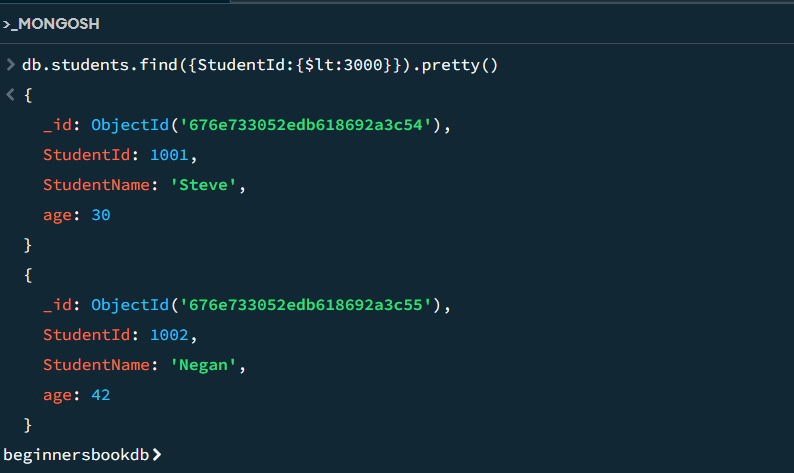
* Greater than criteria:

**Syntax:** db.collection\_name.find({&quot;field\_name&quot;:{$gt:criteria\_value}}).pretty



* Lesser than criteria:

**Syntax:** db.collection\_name.find({&quot;field\_name&quot;:{$lt:criteria\_value}}).pretty()



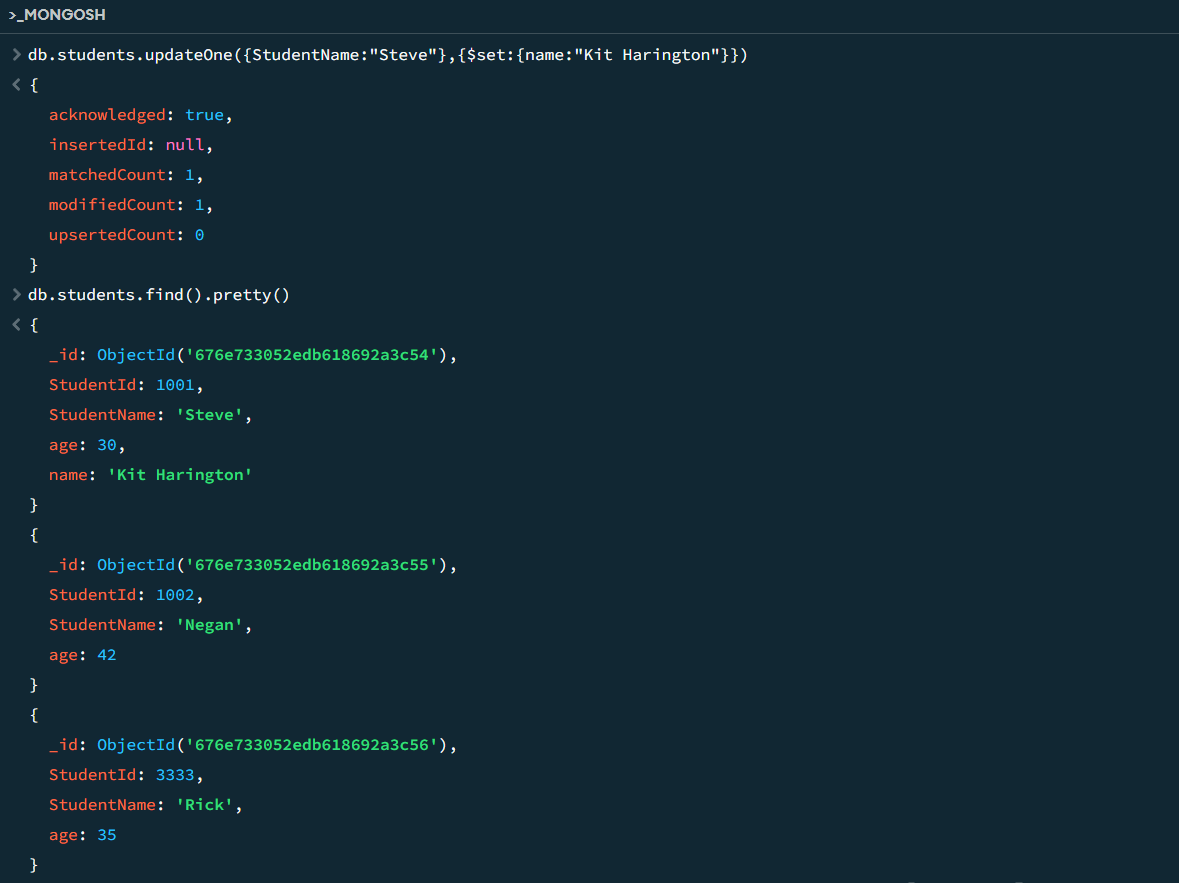
* Not equals to:

**Syntax:** db.collection\_name.find({&quot;field\_name&quot;:{$ne:criteria\_value}}).pretty()



1. **MongoDB Update Document:**
2. **Updating Document using update() method:**

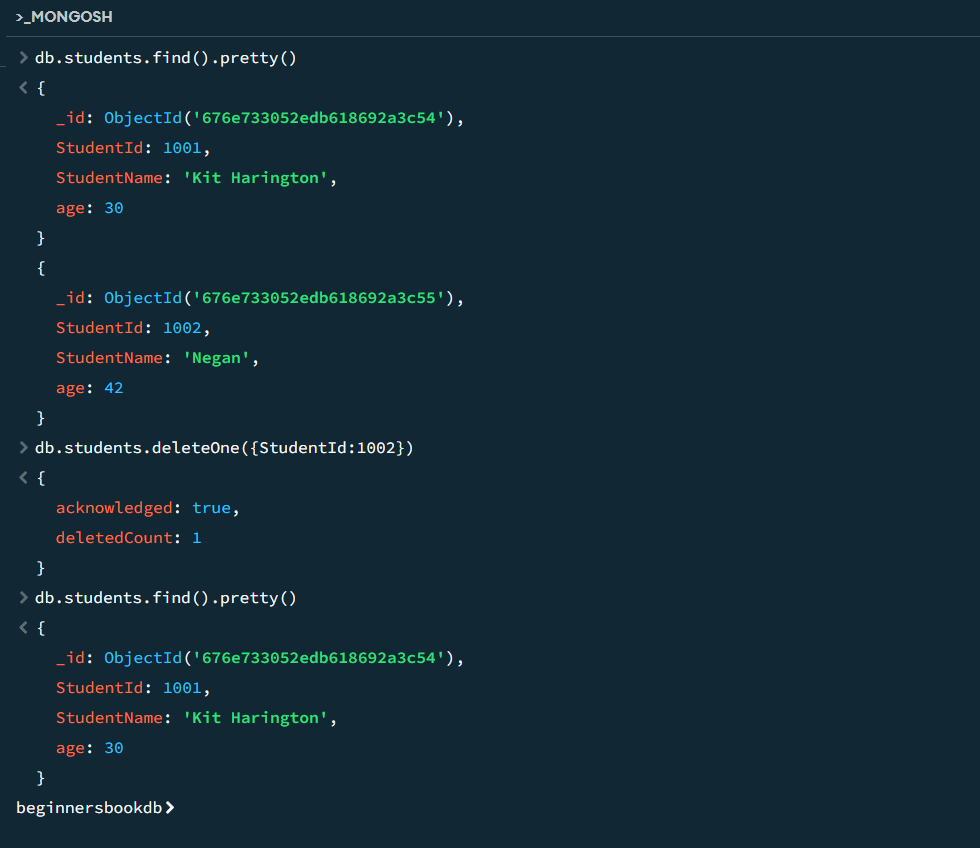
Syntax: db.collection\_name.updateOne(criteria, update\_data)



(**For multiple documents:** db.students.updateMany({StudentName:"Steve"},{$set:{name:"Kit Harington"}},{multi:true})**)**

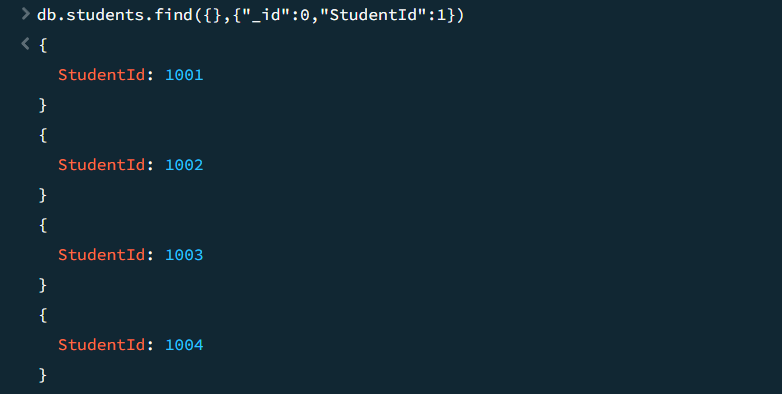
1. **Delete document in MongoDB:**

Syntax: db.collection\_name.deleteOne(delete\_criteria)



1. **MongoDB Projection:**

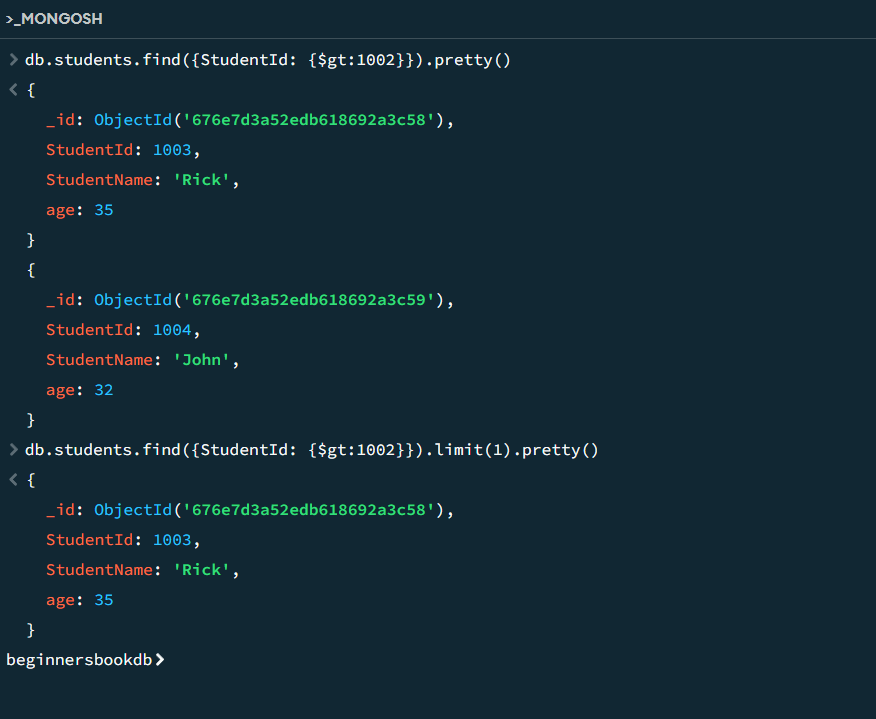
Syntax: db.collection\_name.find({},{field\_key:1 or 0})





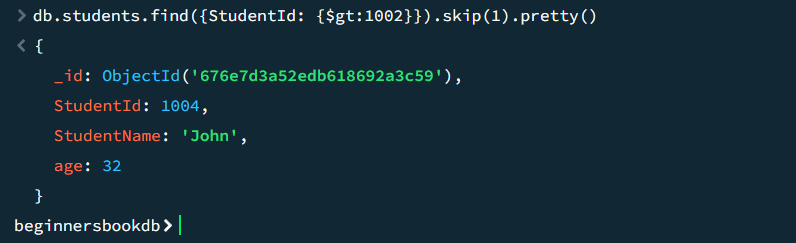
1. **limit() and skip() method in MongoDB:**
2. **limit():**

Syntax: db.collection\_name.find().limit(number\_of\_documents)



1. **skip():**

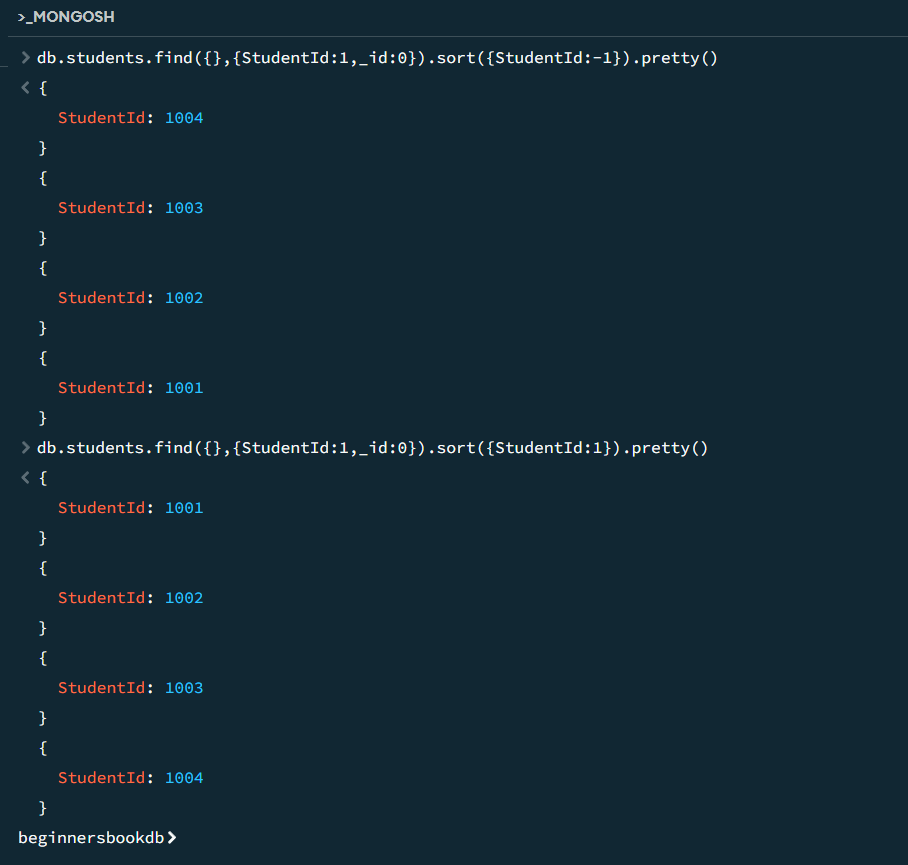
Syntax: db.collection\_name.find().skip(number\_of\_documents)



1. **Sorting of Documents in MongoDB:**

**Syntax of sort() method:** db.collecttion\_name.find().sort({field\_key:1 or -1})

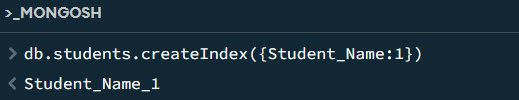
1 is for ascending order and -1 is for descending order. The default value is 1.



1. **MongoDB Indexing:**
2. **Create Index in MongoDB:**

Syntax: db.collection\_name.createIndex({field\_name: 1 or -1})

The value 1 is for ascending order and -1 is for descending order.



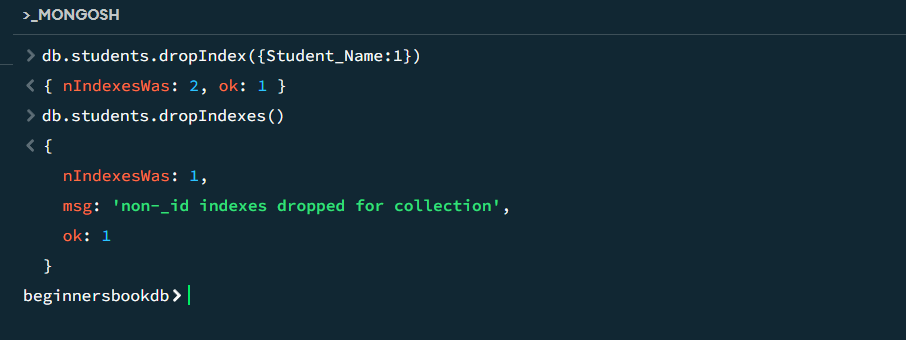
1. **Finding index :**

Syntax: db.collection\_name.getIndexes()



1. **Dropping specific index:**

Syntax: db.collection\_name.dropIndex({index\_name: 1})

****

**Practical No. 3**

**Aim:** Practical of principle components analysis.

**Code:**

install.packages('ggplot2')

library(ggplot2)

data(iris)

X <- iris[, 1:4]

pca\_result <- prcomp(X, center = TRUE, scale. = TRUE)

summary(pca\_result)

biplot(pca\_result, main = "PCA Biplot of Iris Dataset")

pca\_data <- data.frame(pca\_result$x, Species = iris $Species)

ggplot(pca\_data, aes (x = PC1, y = PC2, color = Species)) +

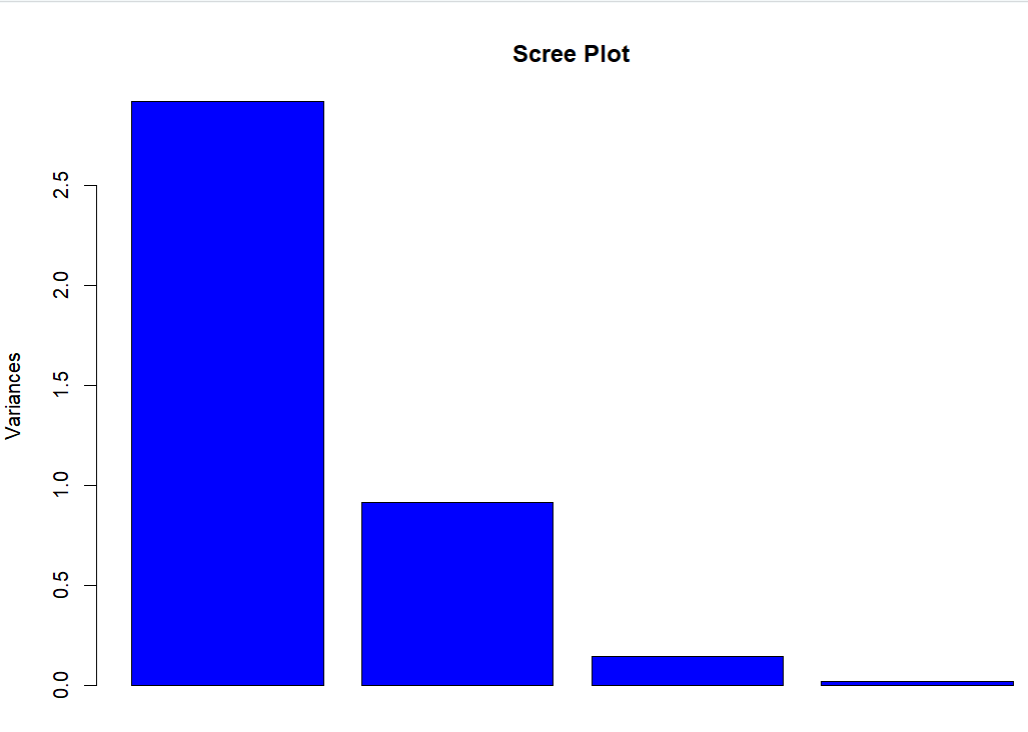
geom\_point() +

labs(title = "PCA of Iris Dataset", x = "Principal Component 1", y = "Principal

Component 2")

screeplot(pca\_result, main = "Screen Plot",col="blue")

**Output:**



**Practical No. 4**

**Aim: -** Practical of k means Clustering

**Code: -**

"k-means clustering "

data("iris")

names(iris)

new\_data<-subset(iris,select = c(-Species))

new\_data

cl<-kmeans(new\_data,3)

cl

data <- new\_data

wss <- sapply(1:15,

function(k){kmeans(data, k )$tot.withinss})

wss

plot(1:15, wss,

type="b", pch = 19, frame = FALSE,

xlab="Number of clusters K",

ylab="Total within-clusters sum of squares")

# install.packages("cluster")

library(cluster)

# install.packages("ggplot2")

library(ggplot2)

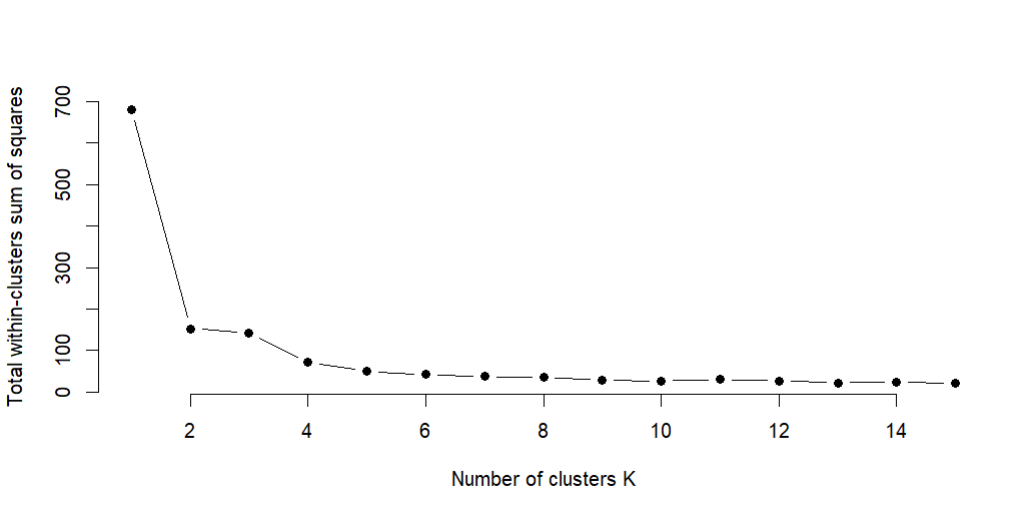
clusplot(new\_data, cl$cluster, color=TRUE, shade=TRUE,

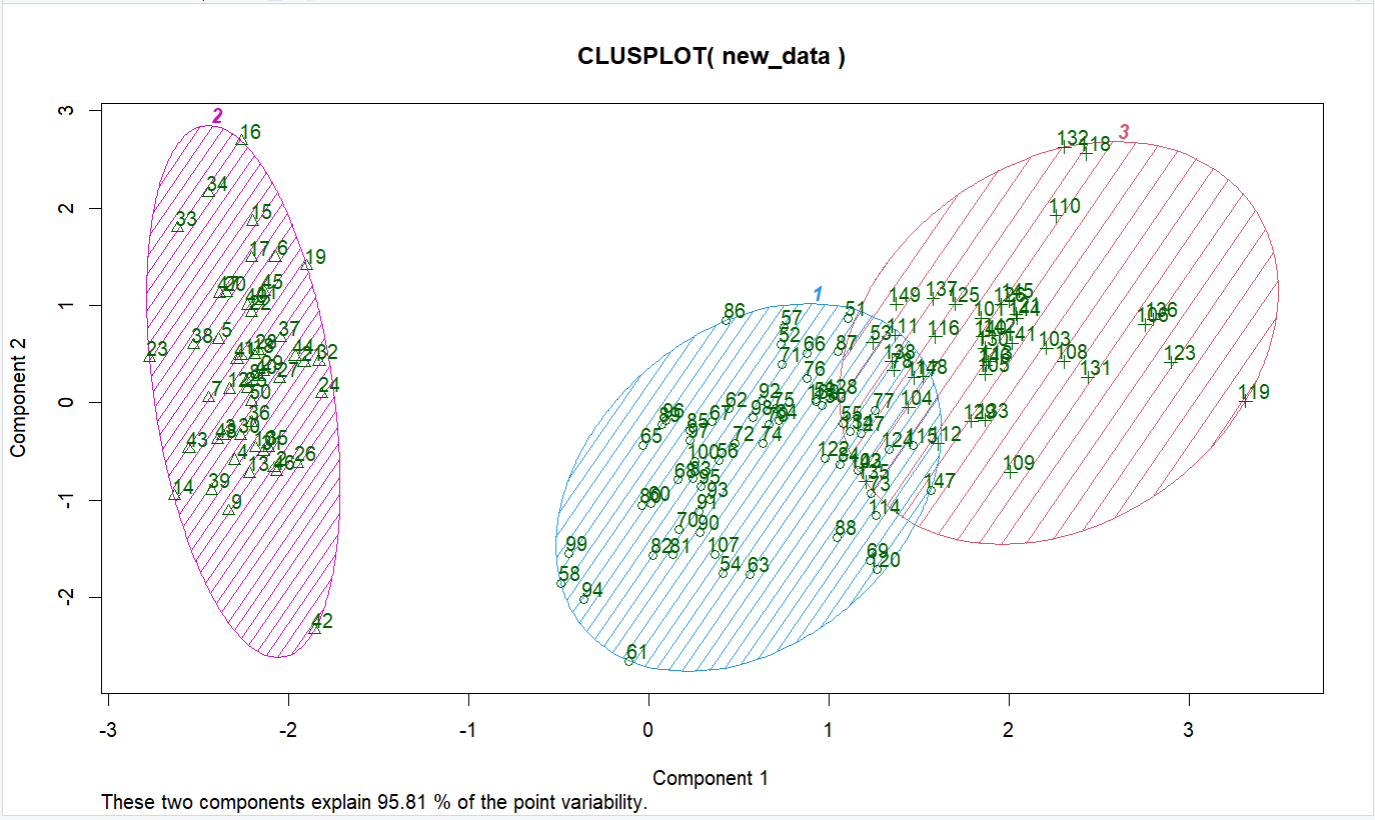
labels=2, lines=0)

cl$cluster

cl$centers

**Output:**





**Practical No. 5**

**Aim: -** Practical of Hierarchical Clustering

**Code: -**

# install.packages("ggplot2")

library(ggplot2)

"agglomarative clustering "

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

plot(clusters)

clusterCut <- cutree(clusters, 3)

table(clusterCut, iris$Species)

ggplot(iris, aes(Petal.Length, Petal.Width, color = iris$Species)) +

geom\_point(alpha = 0.4, size = 3.5) + geom\_point(col = clusterCut) +

scale\_color\_manual(values = c('black', 'red', 'green'))

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

clusterCut1 <- cutree(clusters, 3)

table(clusterCut1, iris$Species)

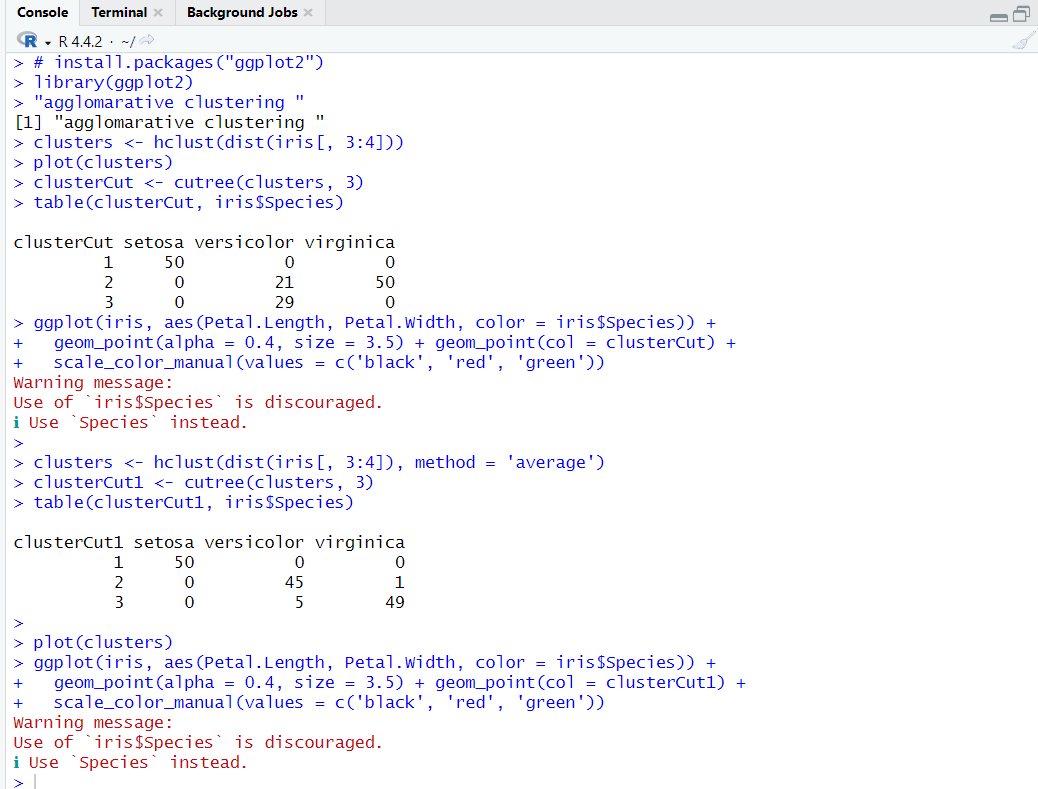
plot(clusters)

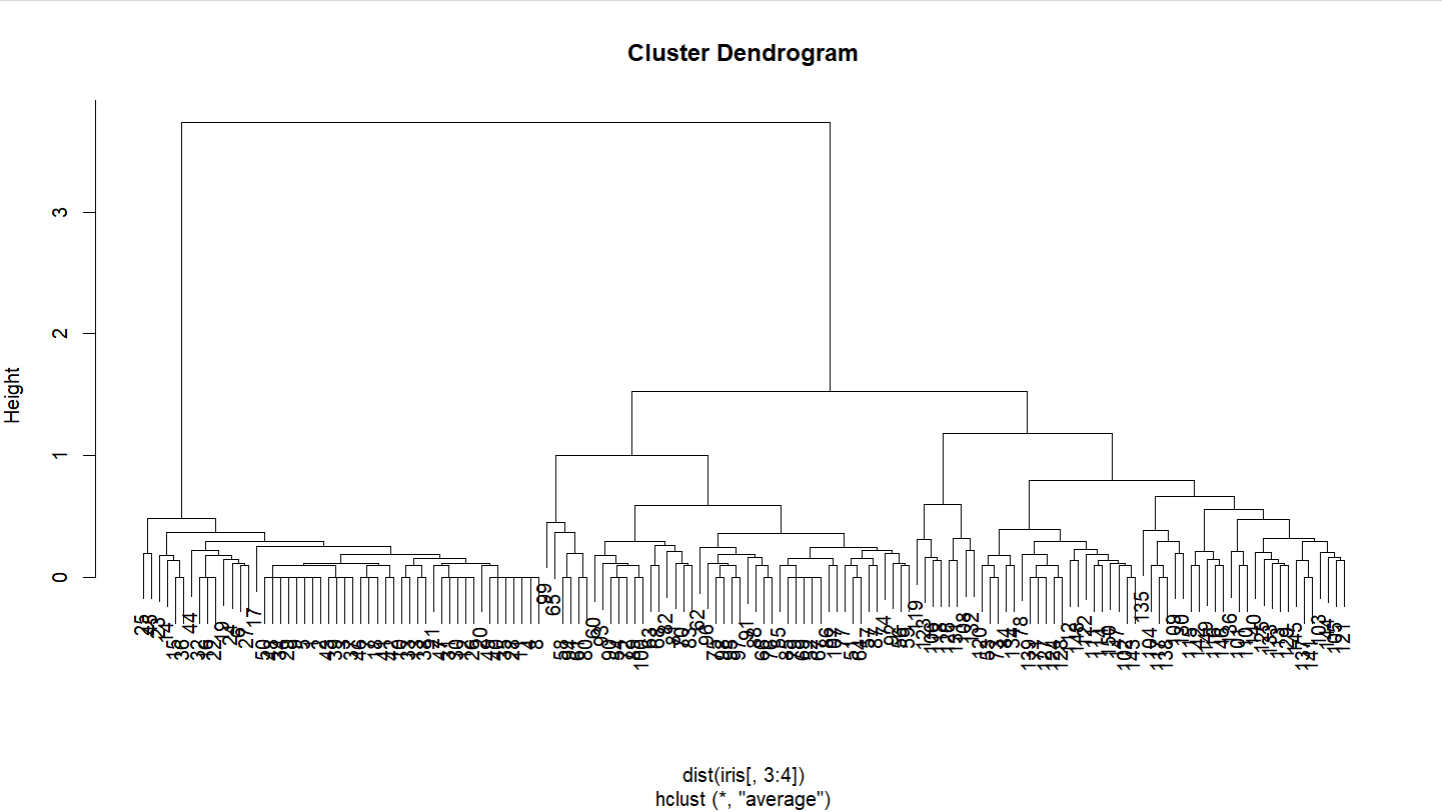
ggplot(iris, aes(Petal.Length, Petal.Width, color = iris$Species)) +

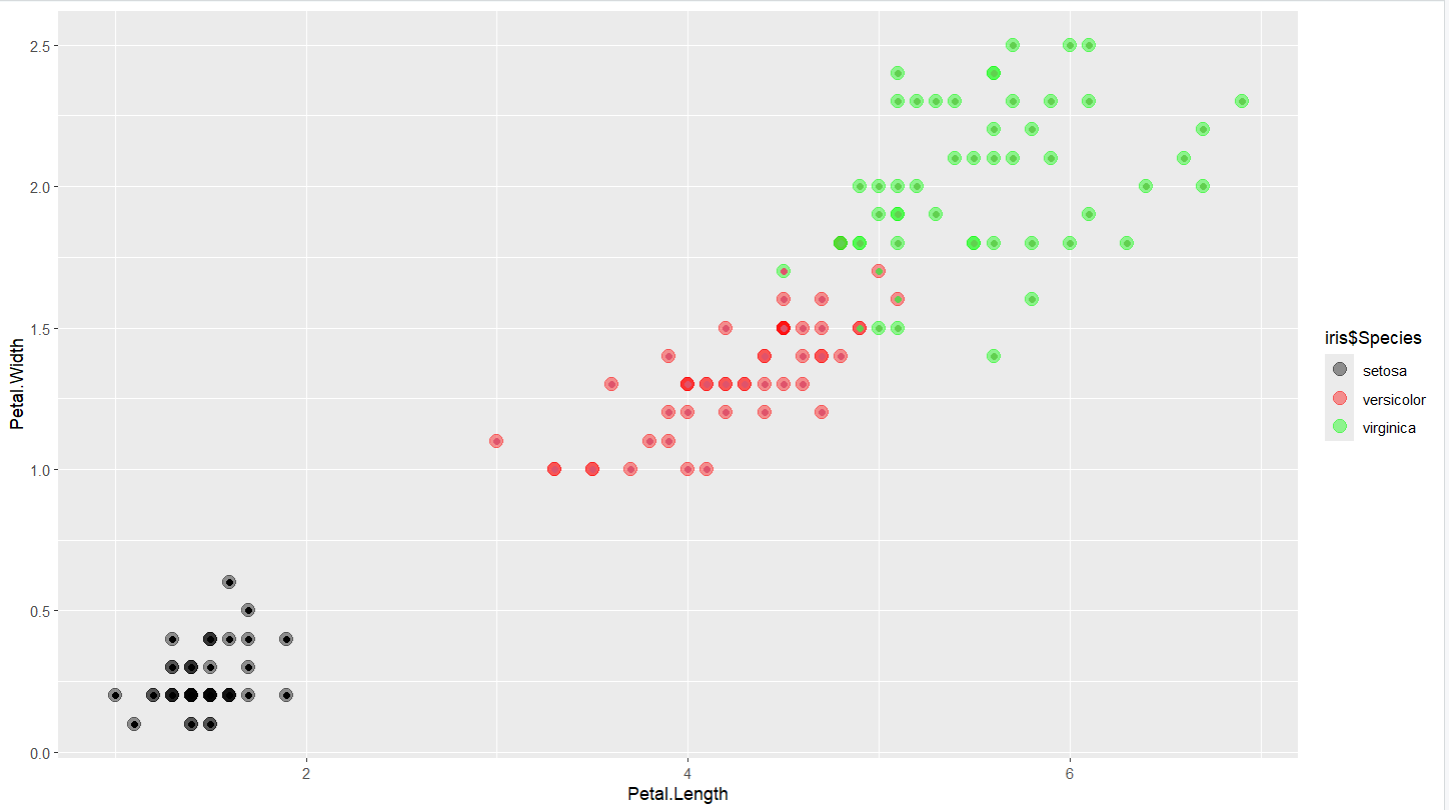
geom\_point(alpha = 0.4, size = 3.5) + geom\_point(col = clusterCut1) +

scale\_color\_manual(values = c('black', 'red', 'green'))

**Output:**







**Practical No. 6**

**Aim :-** Practical of Time-series forecasting

**Code :-**

**1)**

#consider the inbuilt data set Air Passengers

data("AirPassengers")

#to know the format of data set here ts will tell that the

#data set belongs to time series format

class(AirPassengers)

#To know the start of time series

start(AirPassengers)

#To know the end of time series

end(AirPassengers)

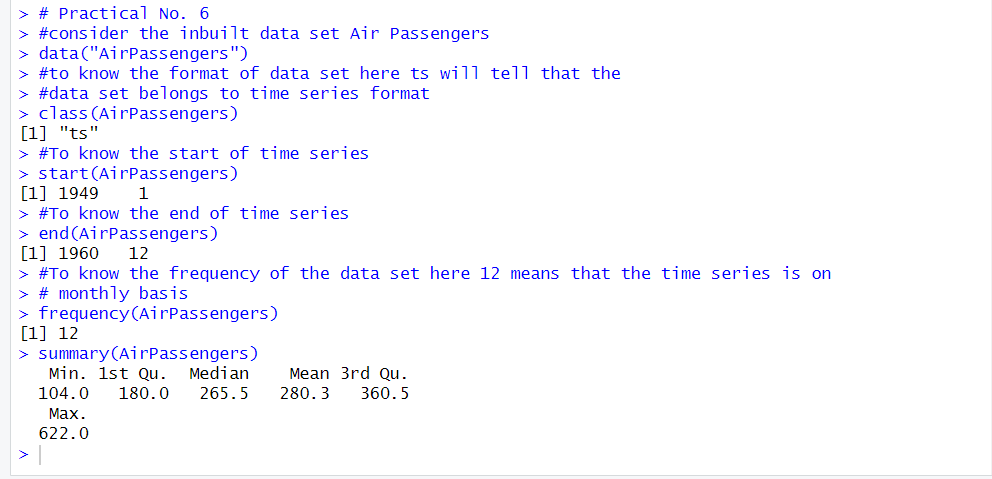
#To know the frequency of the data set here 12 means that the time series is on

# monthly basis

frequency(AirPassengers)

summary(AirPassengers)

**Output :-**

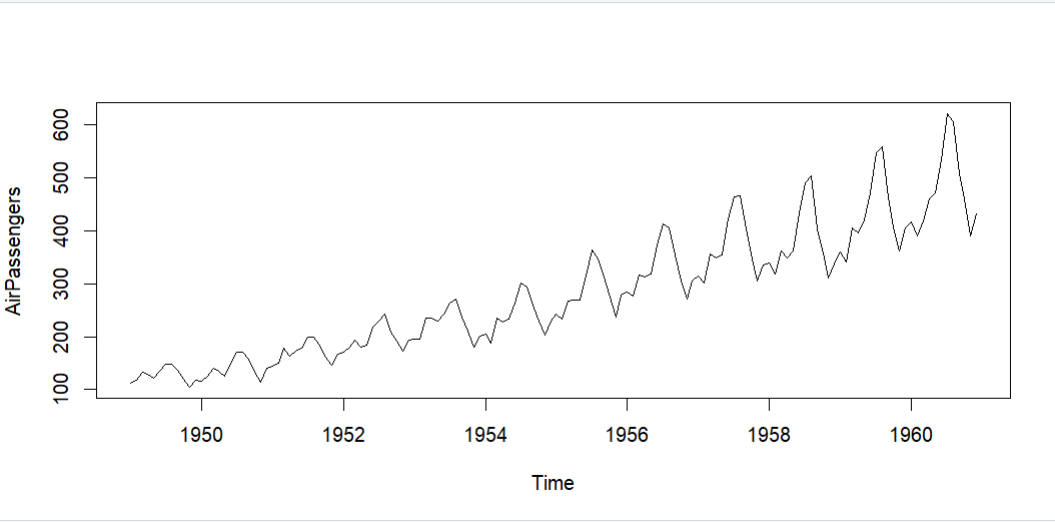


**2) Plot the time series model:**

# To Plot the time series model

plot(AirPassengers)

**Output (2): -**

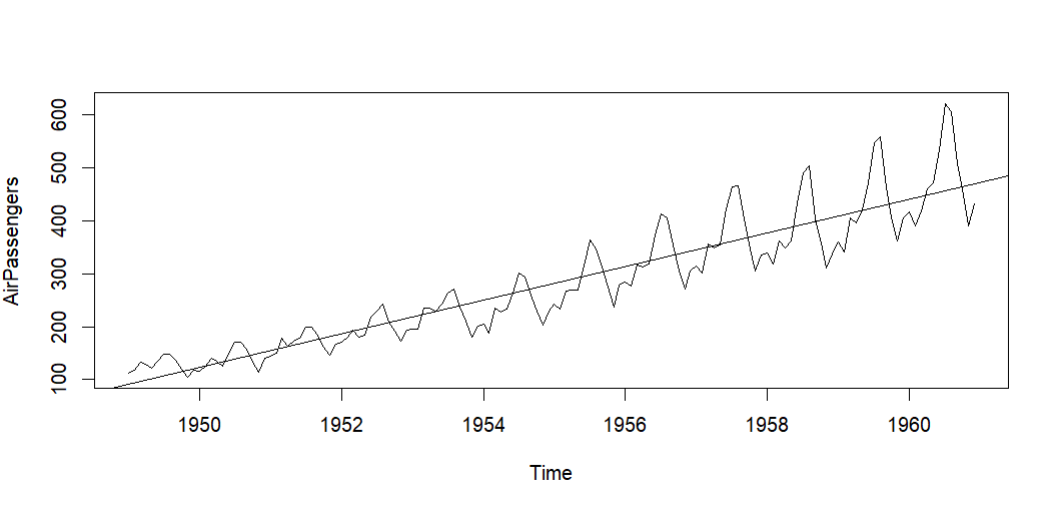


**3) To plot a best fit line which can be used for regression:**

# To plot a best fit line which can be used for regression

abline(reg=lm(AirPassengers~time(AirPassengers)))

**Output (3): -**

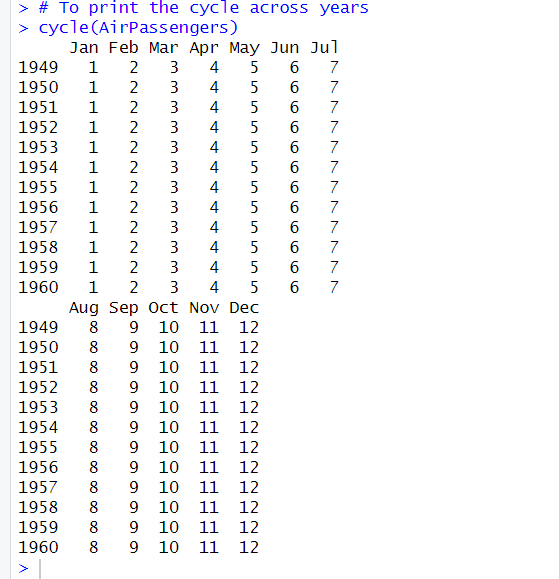


**4) To print the cycle across years: -**

# To print the cycle across years

cycle(AirPassengers)

**Output (4): -**

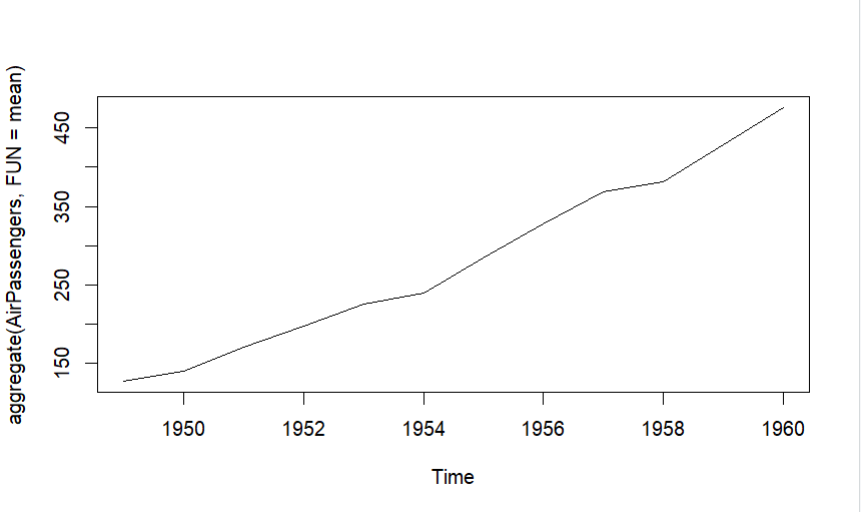


**5) To aggregate the cycle and display its trend per year : -**

# To aggregate the cycle and display its trend per year

plot(aggregate(AirPassengers,FUN = mean))

**Output (5): -**

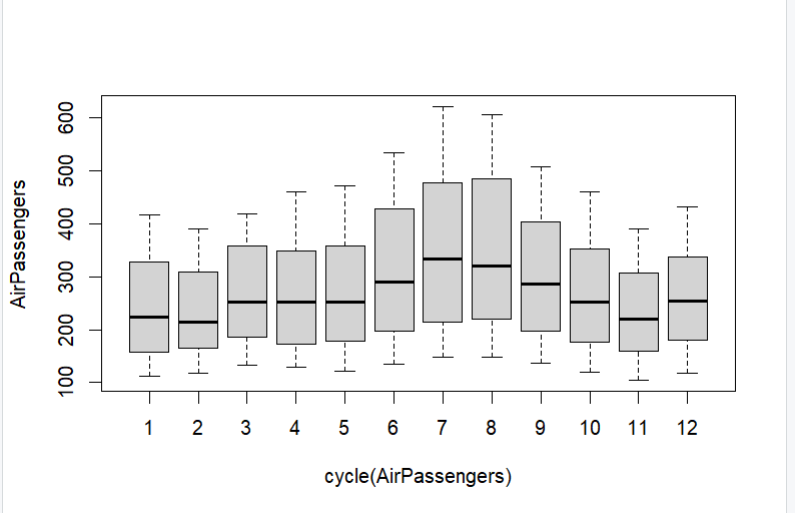


**6) To get the Box Plot : -**

# To get the Box Plot

boxplot(AirPassengers~cycle(AirPassengers))

**Output (6): -**



**Practical No. 7**

**Aim: -** Practical to implement regression line in R

**Code: -**

#Simple plot by considering in built data set mtcars

attach(mtcars)

plot(wt,mpg)

# plot is the function and wt and mpg are the attributes available

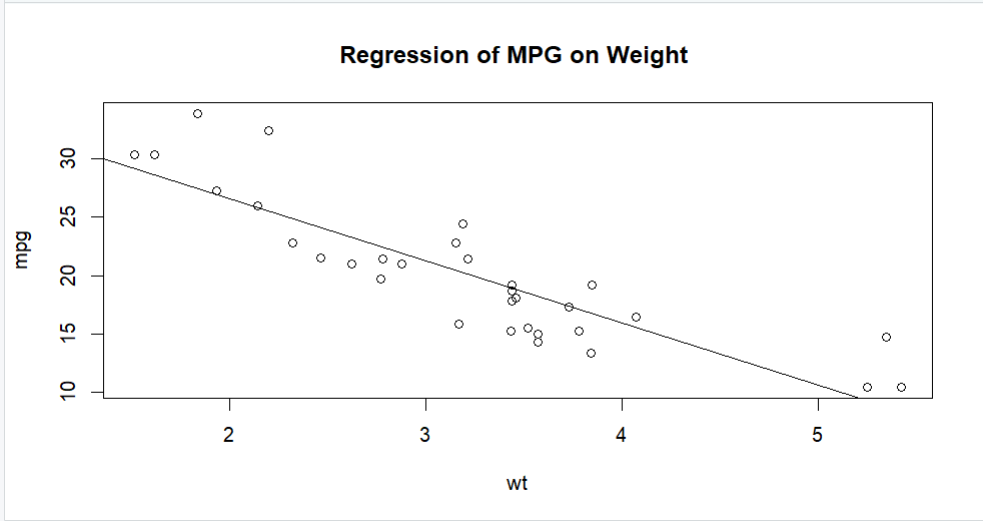
# in mtcars dataset

abline(lm(mpg ~ wt))

# abline is used to draw line on plot and mpg (~) versus wt

title("Regression of MPG on Weight ")

**Output:**



**Practical No. 8**

**Aim: -** Practical of Simple/Multiple Linear Regression

**Code: -**

1. **Simple Linear Regression:**

#Consider some dataset

#The predictor vector (independent variable)

height <- c(151,174,138,186,128,136,179,163,152,131)

#The response vector (dependent variable)

weight <- c(63,81,56,91,47,57,76,72,62,48)

#apply lm function for linear regression

student <- lm(weight ~ height)

student

#Find the weight of a person with height 170

a <- data.frame(height = 170)

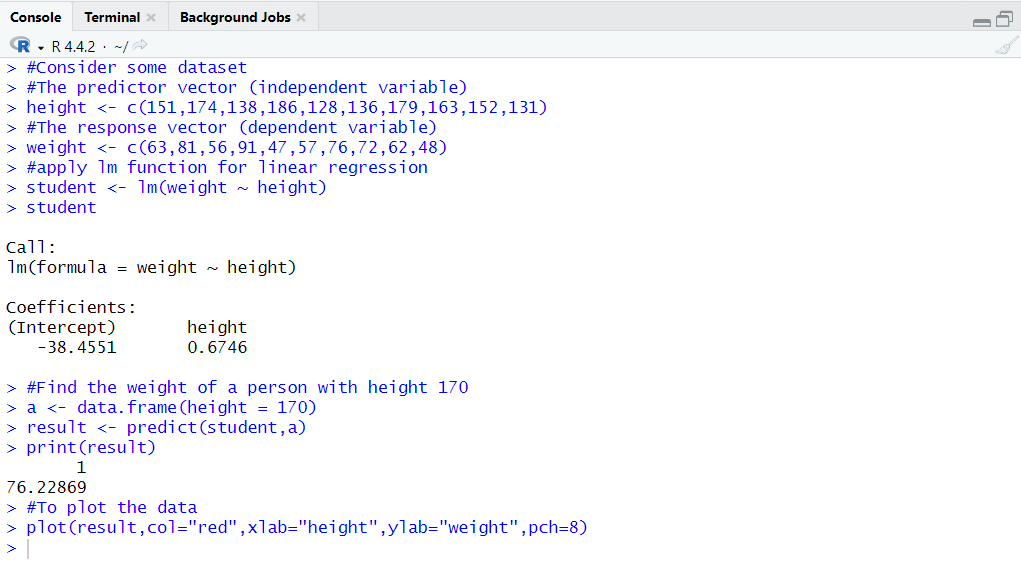
result <- predict(student,a)

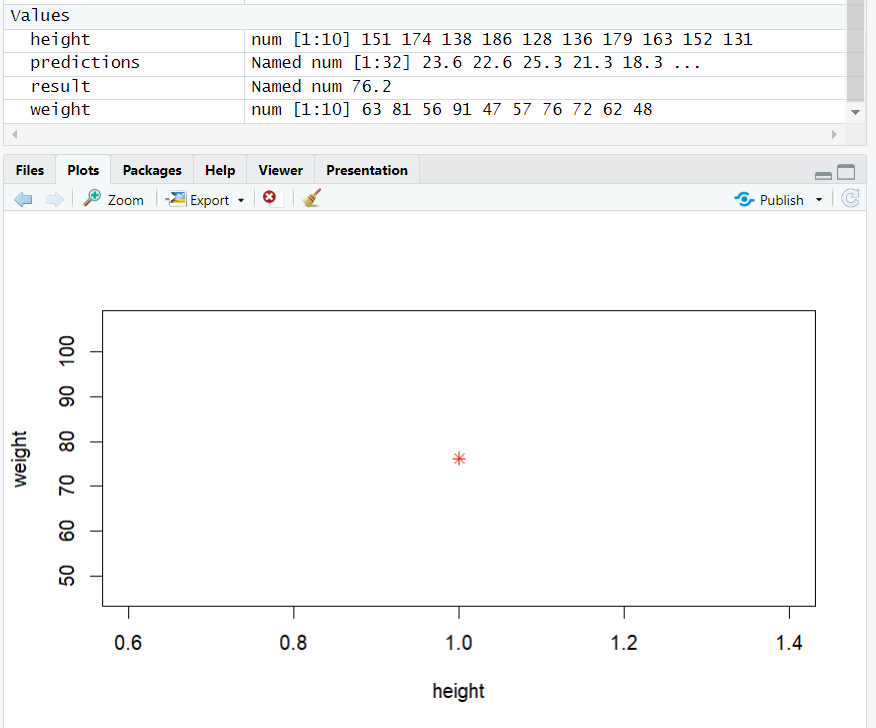
print(result)

#To plot the data

plot(result,col="red",xlab="height",ylab="weight",pch=8)

**Output :-**





1. **Multiple Linear Regression:**

data(mtcars) #Load the built-in mtcars dataset

head(mtcars) #View the first few rows of the dataset

summary(mtcars) #Summary Statistics

model <- lm(mpg ~ wt + hp,data = mtcars) #Fit a multiple linear

# regression model

summary(model) # view the model summary

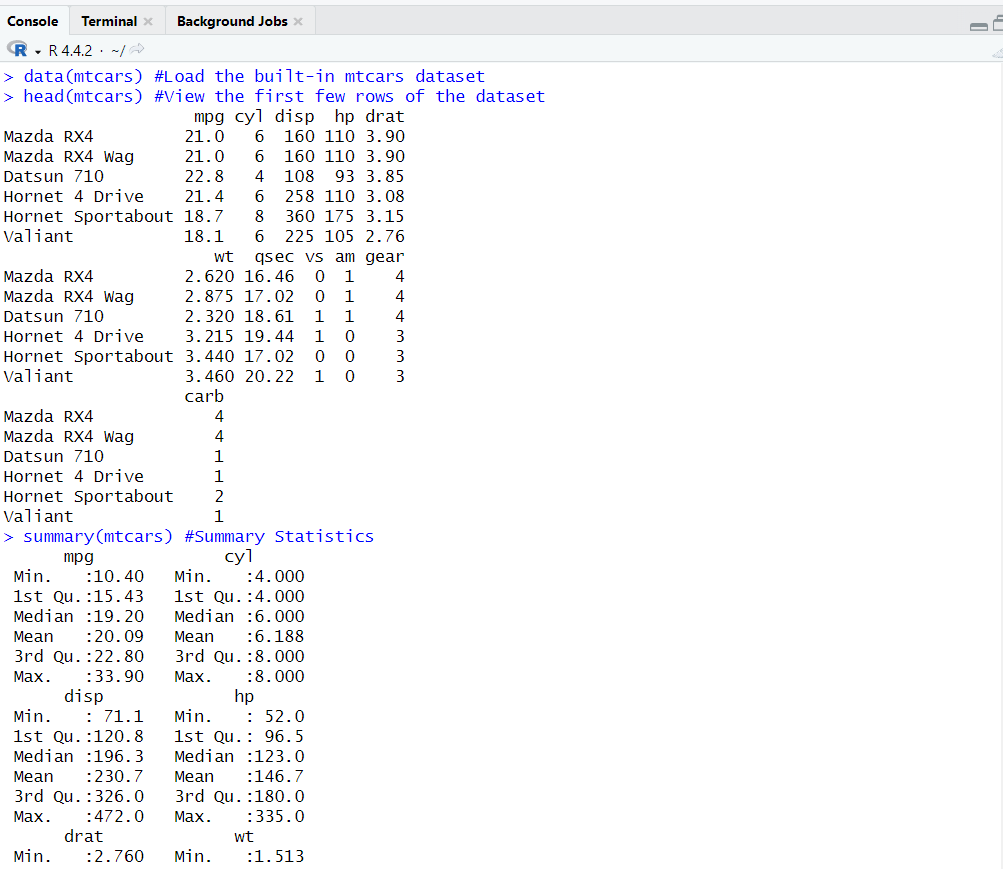
new\_data <- data.frame(wt = c(3,2.5),hp= c(110,150))

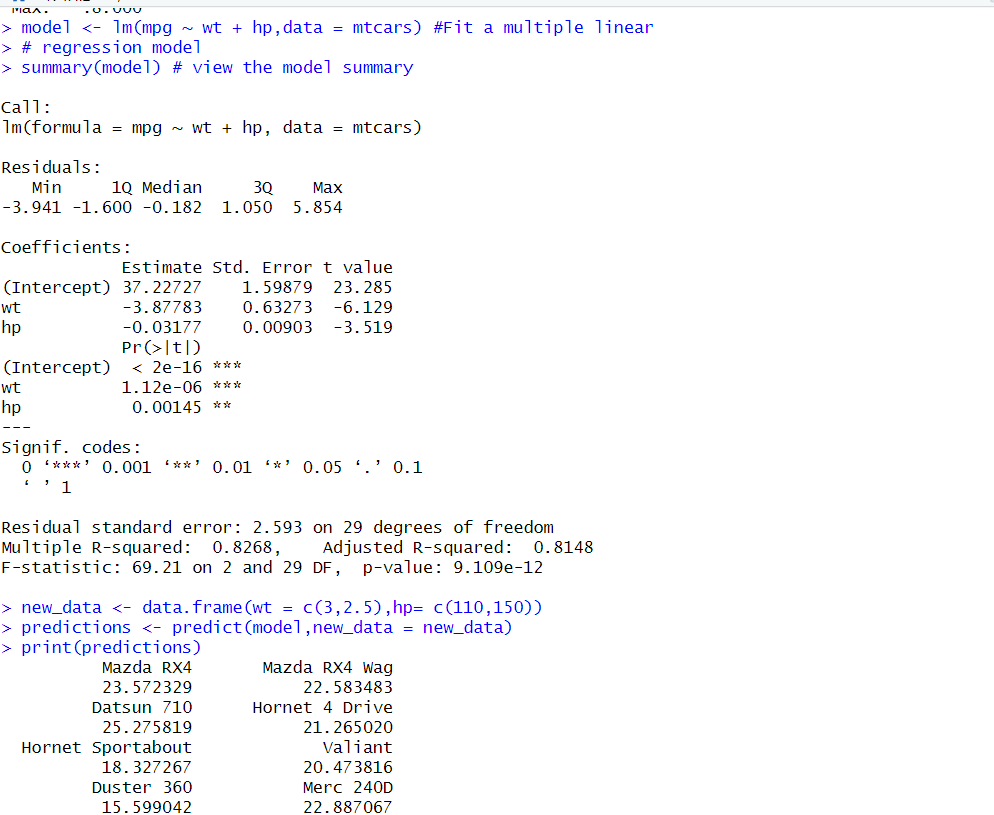
predictions <- predict(model,new\_data = new\_data)

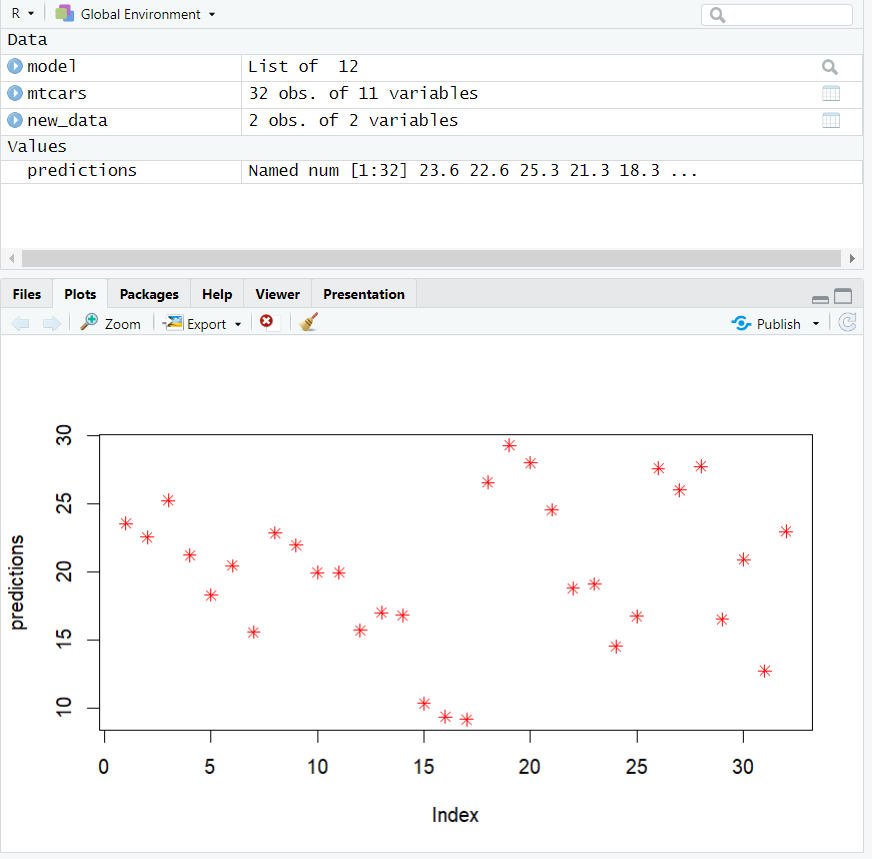
print(predictions)

plot(predictions,col="red",pch=8)

**Output :-**







**Practical No. 9**

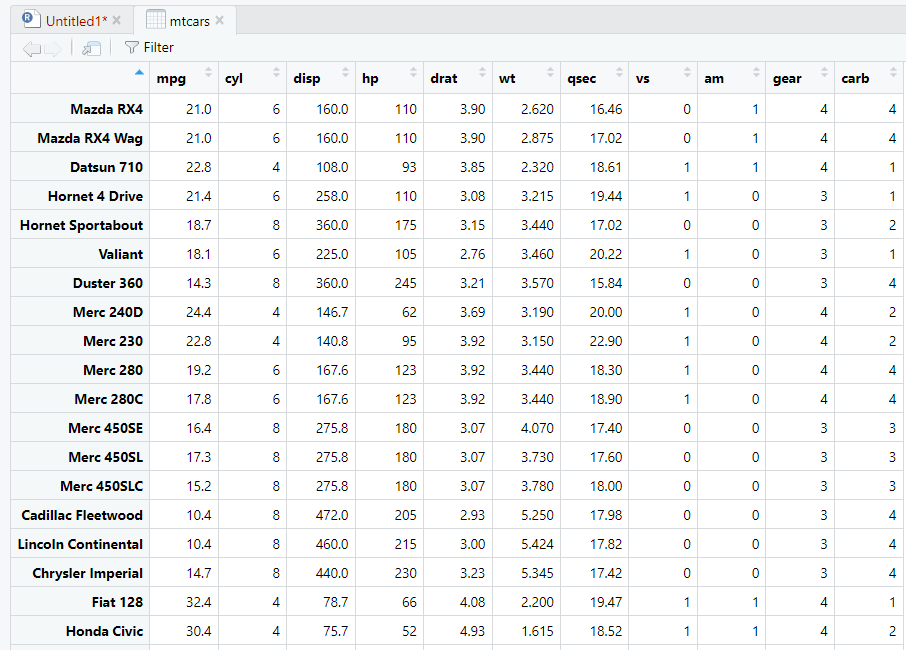
**Aim:** Practical to Implement Logistics Regression.

**Code:**

# Load the 'mtcars' dataset and display it for exploration

View(mtcars)

**Output:**



# The variable 'vs' in the 'mtcars' dataset indicates the engine type:

# vs = 0 (V-shaped engine), vs = 1 (Straight engine)

#===============####=================

# Building the first logistic regression model based on 'mpg'

model1 <- glm(formula = vs ~ mpg, data = mtcars, family = "binomial")

summary(model1)

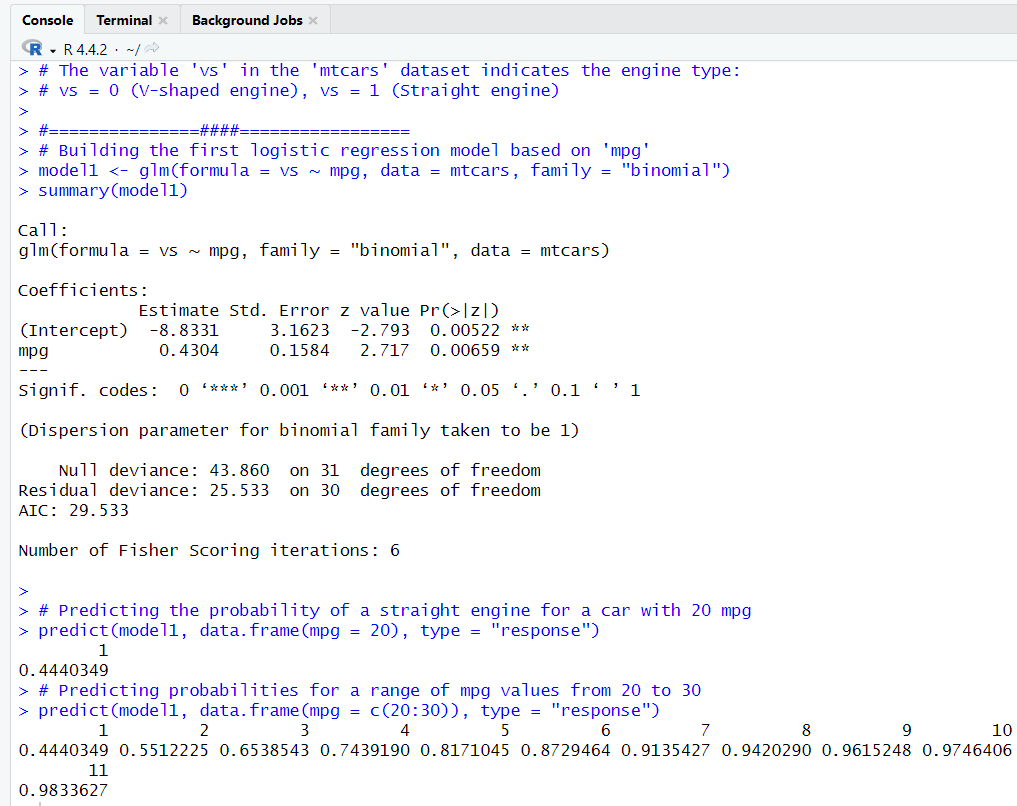
# Predicting the probability of a straight engine for a car with 20 mpg

predict(model1, data.frame(mpg = 20), type = "response")

# Predicting probabilities for a range of mpg values from 20 to 30

predict(model1, data.frame(mpg = c(20:30)), type = "response")

**Output:**

****

# Building the second logistic regression model based on 'hp'

model2 <- glm(formula = vs ~ hp, data = mtcars, family = "binomial")

summary(model2)

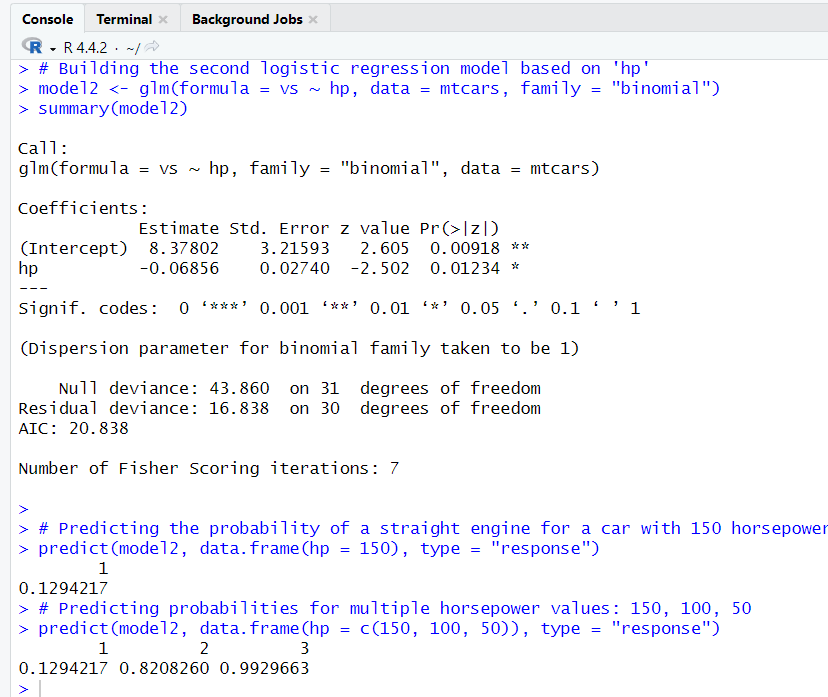
# Predicting the probability of a straight engine for a car with 150 horsepower

predict(model2, data.frame(hp = 150), type = "response")

# Predicting probabilities for multiple horsepower values: 150, 100, 50

predict(model2, data.frame(hp = c(150, 100, 50)), type = "response")

**Output:**

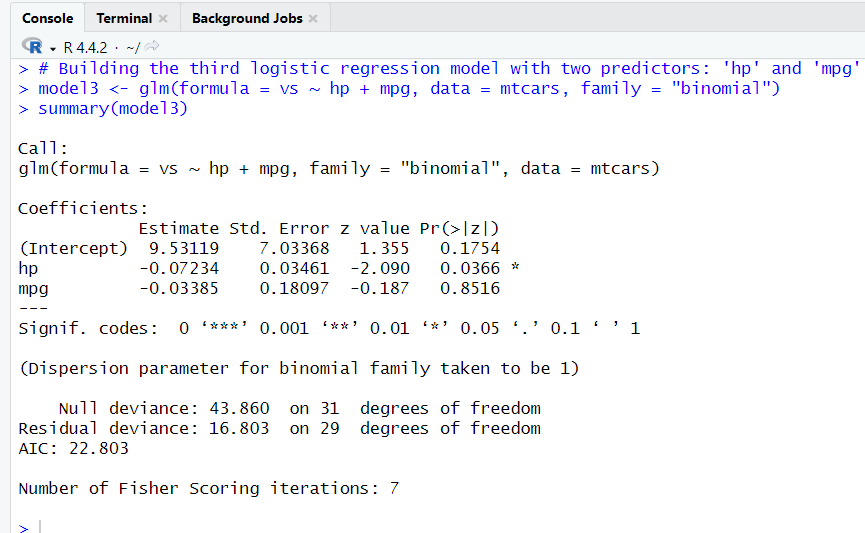
****

# Building the third logistic regression model with two predictors: 'hp' and 'mpg'

model3 <- glm(formula = vs ~ hp + mpg, data = mtcars, family = "binomial")

summary(model3)

**Output:**

****

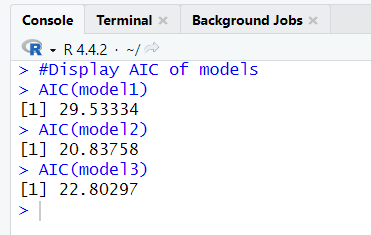
#Display AIC of models

AIC(model1)

AIC(model2)

AIC(model3)

**Output:**

****

**Practical No. 10**

**Aim:** Practical to Implement Hypothesis Testing.

**Hypothesis Testing in R Programming**

Hypothesis Testing is a process used to validate or reject an assumption (hypothesis) based on sample data. The four-step process involves stating the hypothesis, formulating a plan for analysis, analyzing sample data, and interpreting the decision based on the test statistic.

**Four-Step Process of Hypothesis Testing**

1. State the hypothesis
2. Formulate an analysis plan and set the criteria for decision
3. Analyze sample data
4. Interpret decision

**1. One Sample T-Testing**

Use: Compares the mean of a sample to a known value or population mean.

**Code:**

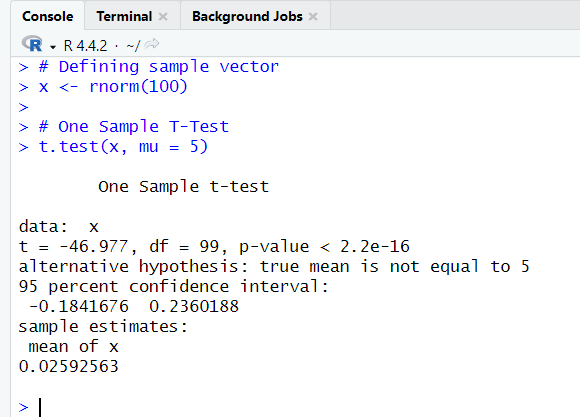
# Defining sample vector

x <- rnorm(100)

# One Sample T-Test

t.test(x, mu = 5)

**Output:**

****

**2. Two Sample T-Testing**

Use: Compares the means of two independent samples to determine if they are significantly different.

**Code:**

# Defining sample vectors

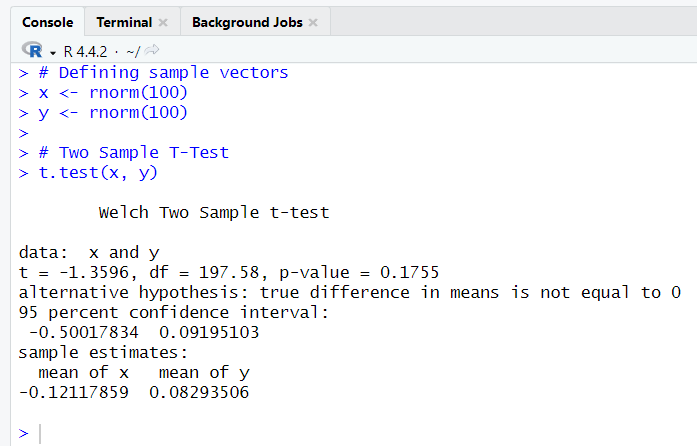
x <- rnorm(100)

y <- rnorm(100)

# Two Sample T-Test

t.test(x, y)

**Output**:



**3. Directional Hypothesis**

**Use**: Tests whether the sample mean is greater or smaller than a known population mean in a specified direction.

**Code:**

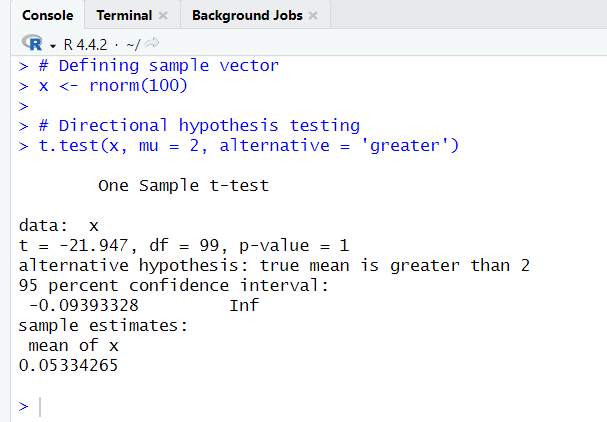
# Defining sample vector

x <- rnorm(100)

# Directional hypothesis testing

t.test(x, mu = 2, alternative = 'greater')

**Output:**

****

**4. One Sample μ-Test**

Use: Non-parametric test to compare the median of a sample to a known value.

**Code:**

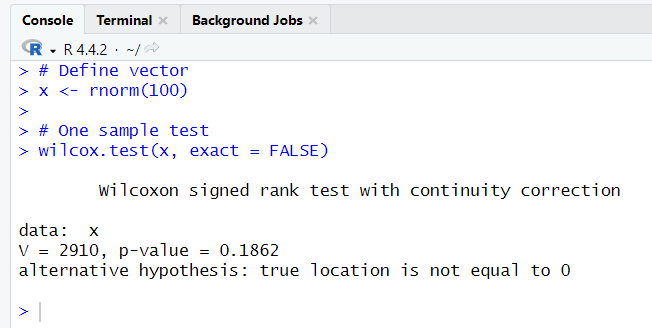
# Define vector

x <- rnorm(100)

# One sample test

wilcox.test(x, exact = FALSE)

**Output:**

****

**5. Two Sample μ-Test**

Use: Non-parametric test to compare the medians of two independent samples.

**Code:**

# Define vectors

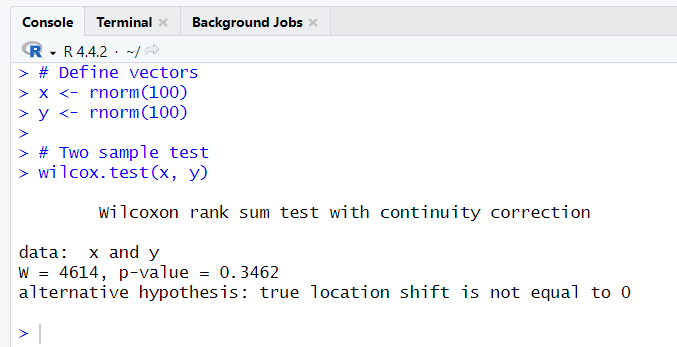
x <- rnorm(100)

y <- rnorm(100)

# Two sample test

wilcox.test(x, y)

**Output:**

****

**6. Correlation Test**

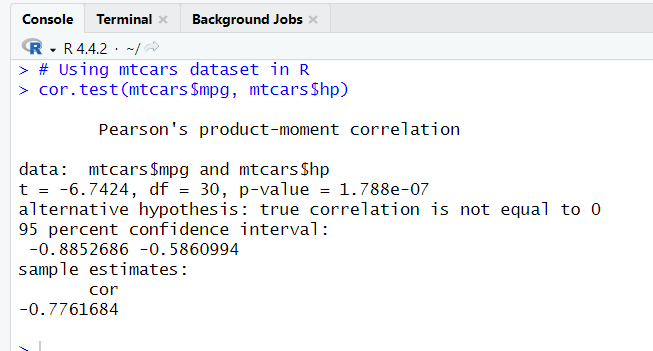
Use: Tests the strength and direction of the linear relationship between two variables.

**Code:**

# Using mtcars dataset in R

cor.test(mtcars$mpg, mtcars$hp)

**Output:**



**Practical No. 11**

**Aim: -** Practical of one way Anova

**Code: -**

# One way anova

data1 <- read.csv(file.choose(),sep = ",",header = T)

names(data1)

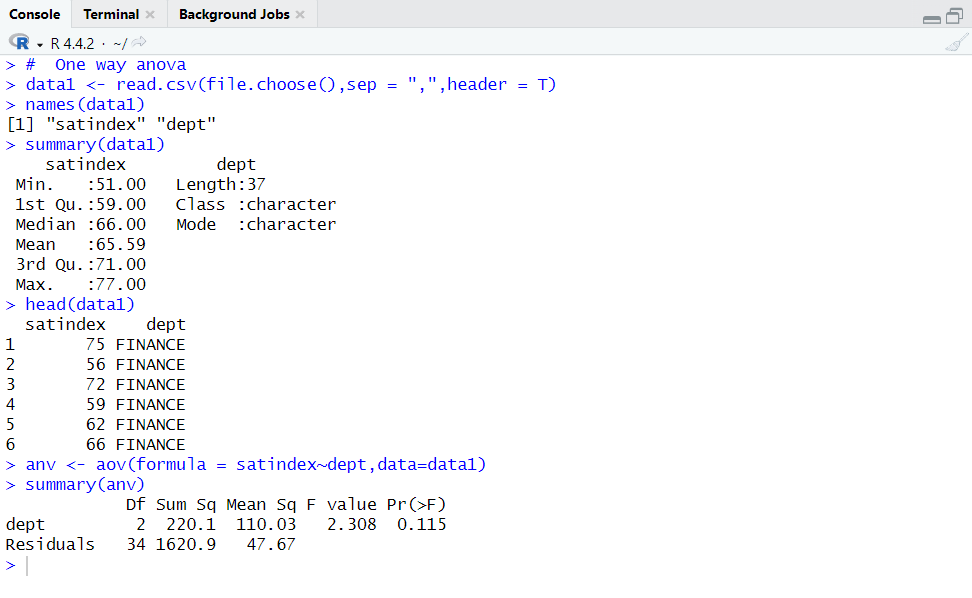
summary(data1)

head(data1)

anv <- aov(formula = satindex~dept,data=data1)

summary(anv)

**Output:**



**Practical No. 12**

**Aim: -** Practical of two way Anova

**Code: -**

"two way anova"

data2<-read.csv(file.choose(),sep=",",header = T)

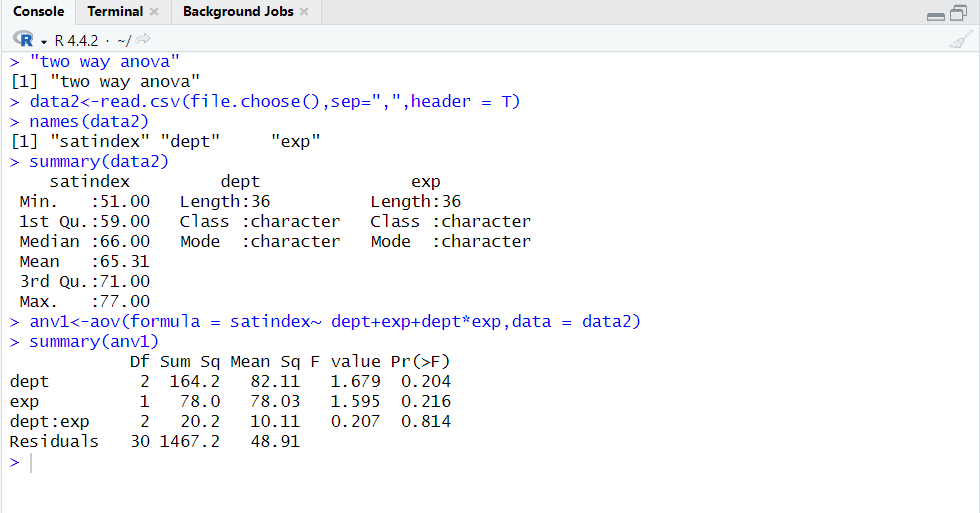
names(data2)

summary(data2)

anv1<-aov(formula = satindex~ dept+exp+dept\*exp,data = data2)

summary(anv1)

**Output:**



**Practical No. 13**

**Aim:** Practical to Implement Decision Tree.

**Code:**

# Load necessary libraries

library(tree)

library(caret)

library(dplyr)

# Load the dataset

df <- read.csv("C:\\Users\\sejalpatil1728\\Downloads\\student\_improved.csv")

# Convert categorical columns to factors

df$Gender <- as.factor(df$Gender)

df$ProgrammingLanguage <- as.factor(df$ProgrammingLanguage)

df$Grade <- as.factor(df$Grade)

# Split the data into training and testing sets (80% train, 20% test)

set.seed(42) # For reproducibility

trainIndex <- createDataPartition(df$Grade, p = 0.8, list = FALSE)

trainData <- df[trainIndex, ]

testData <- df[-trainIndex, ]

# Create the decision tree model using the tree package

model <- tree(Grade ~ Age + Gender + ProgrammingLanguage, data = trainData)

# Print the model summary

summary(model)

# Predict on the test data

predictions <- predict(model, testData, type = "class")

# Evaluate the model's accuracy

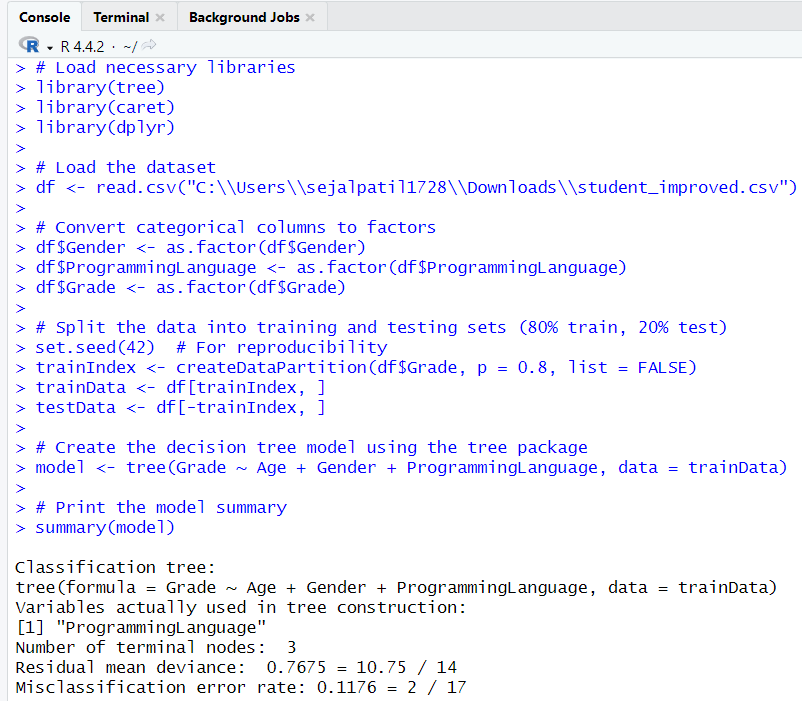
confusionMatrix(predictions, testData$Grade)

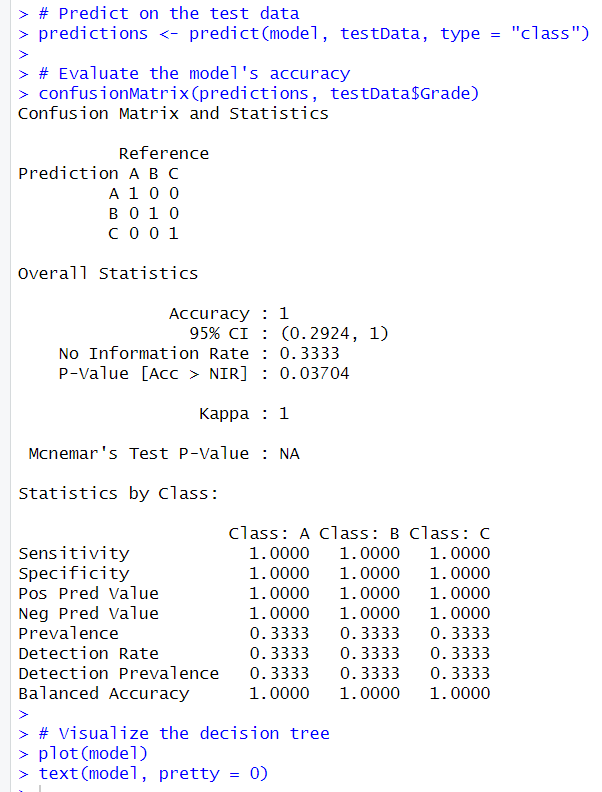
# Visualize the decision tree

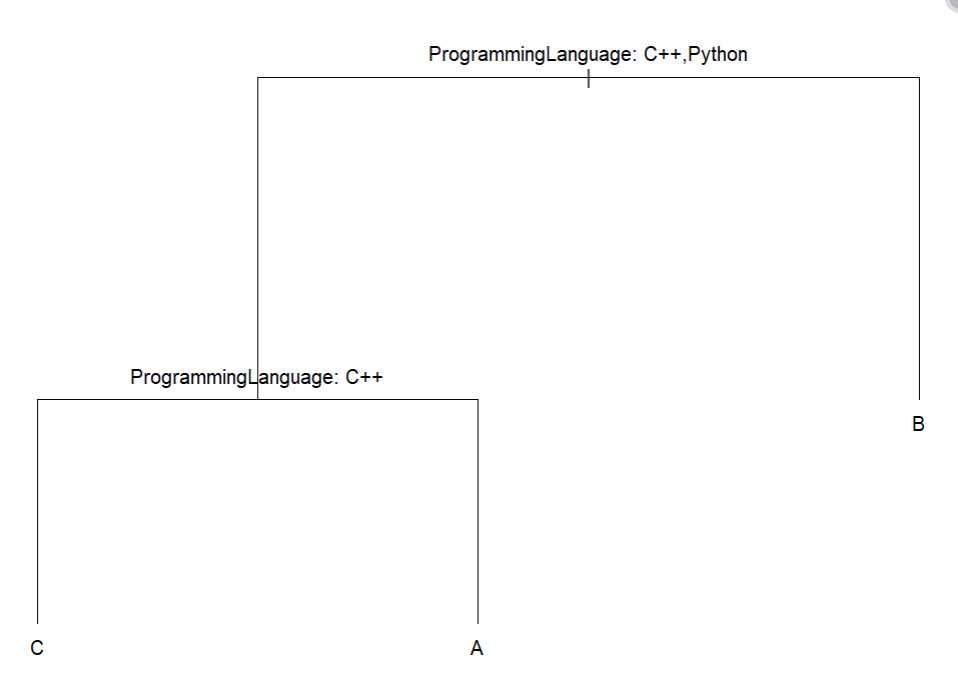
plot(model)

text(model, pretty = 0)

**Output:**

****

****

****

The decision tree first splits the data based on the Programming Language (C++ or Python).

C++ students: The tree checks features like Age and Gender to predict grades such as B, C, or A.

Python students: Similarly, it uses features like Age and Gender to predict grades, which could also be B, C, or A.

In short, it classifies students into different grades based on their Programming Language and other features.

**Practical No. 14**

**Aim: -** Practical to draw histogram, Line plot

**Code: -**

1. **Line Plot:**

#Line Plot

#create some data in x

x <- c(1:5)

y <- x

#plotting symbol and color

par(pch = 22 ,col = "darkblue")

#all plots on one page

par(mfrow = c(2,4))

opts = c("p","l","o","b","c","s","S","h")

for(i in 1:length(opts)){

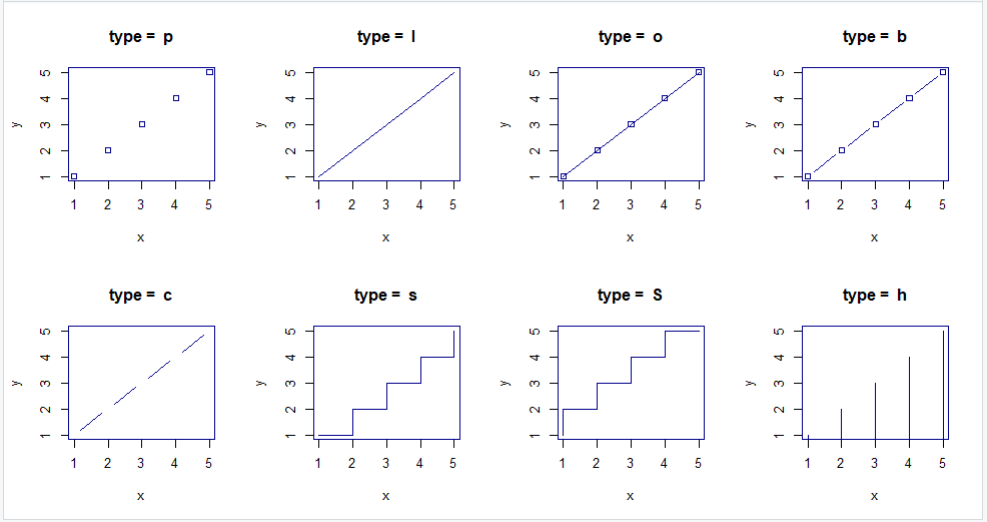
heading = paste("type = ",opts[i])

plot(x,y,type="n",main=heading)

lines(x,y,type=opts[i])

}

**Output:**



1. **Histogram:**

#Histogram in R using mtcars dataset

hist(mtcars$mpg)

#mtcars is the name of the dataset and mpg is attribute

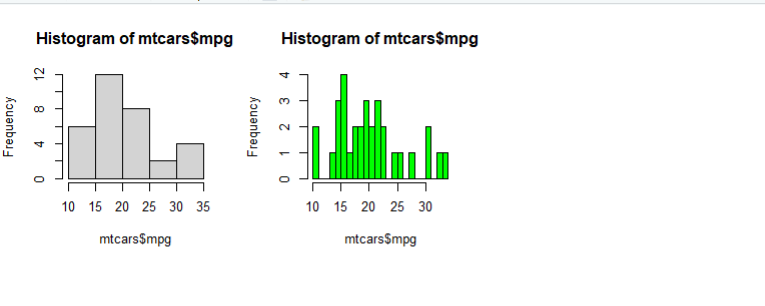
#$ is used to mention that mpg is in mtcars,hist is the command

#for histogram no of bars in the histogram can be controlled

# and bars can be colored

hist(mtcars$mpg,breaks=20,col="green")

**Output:**

****

**Practical No. 15**

**Aim: -** Practical to draw pie chart, box plot and scattered diagram

**Code: -**

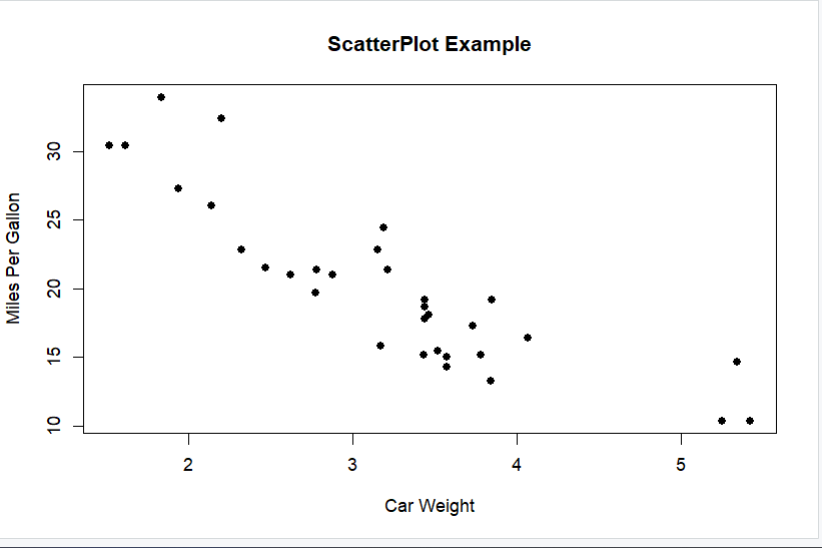
1. **Scattered Diagram (ScatterPlot):**

#simple scatterplot from mtcars dataset

attach(mtcars)

plot(wt,mpg,main="ScatterPlot Example",xlab="Car Weight",ylab="Miles Per Gallon",pch=19)

**Output :**



1. **Pie Chart:**

# Pie Chart with percentages

slices <- c(20,20,30,30)

label <- c('INDIA','US','DUBAI','FRANCE')

pct <- round(slices/sum(slices)\*100)

label <- paste(label,pct)

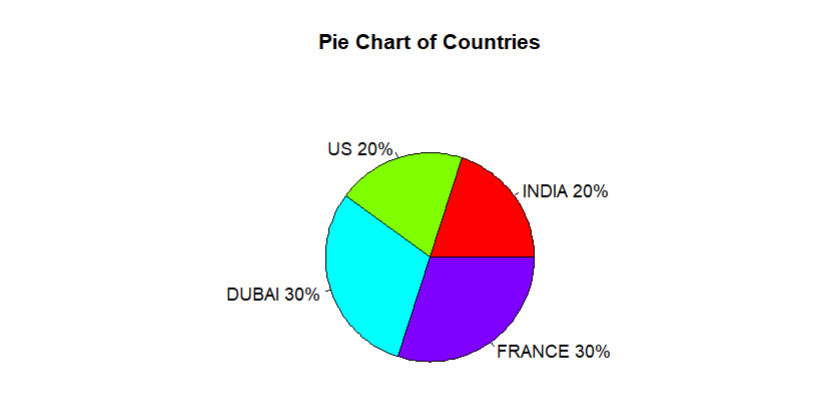
#add percents to labels

label <- paste(label,"%",sep="")

#ad % to labels

pie(slices,labels = label,col=rainbow(length(label)),main="Pie Chart of Countries")

**Output:**

****

1. **Box Plot:**

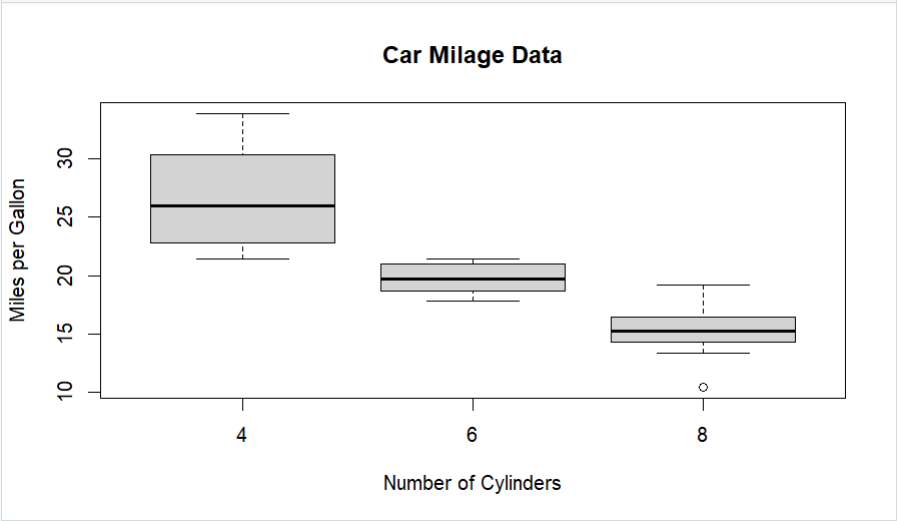
#Box Plot

#Boxplot of MPG by car cylinders

boxplot(mpg ~ cyl,data = mtcars,main="Car Milage Data",

xlab="Number of Cylinders",ylab="Miles per Gallon")

**Output:**

****