College of Arts, Science & Commerce (Empowered Autonomous)

SIES

RISE WITH EDUCATION

NAAC REACCREDITED "A" GRADE

SIES College of Arts, Science and Commerce(Empowered Autonomous),

Sion (W), Mumbai – 400 022.

Department of Information Technology

CERTIFICATE

This is to certify that Mr. / Miss. Nadar Jeba Sona Mariya Selvan of MSc [Information Technology] Semester II, Seat No. FMIT2526176 has successfully completed the practicals for the subject of Big Data Analytics as a partial fulfilment of the degree M.Sc(IT) during the academic year 2025-26.

Faculty-in-Charge

Anita Gupta

Course Co-ordinator Date : External Examiner

Sudha Bhagavatheeswaran

Index

Practical 1

Aim: To implement CRUD operation in mongo db

Software: MongoDB Server, Command line, Mongo shell service

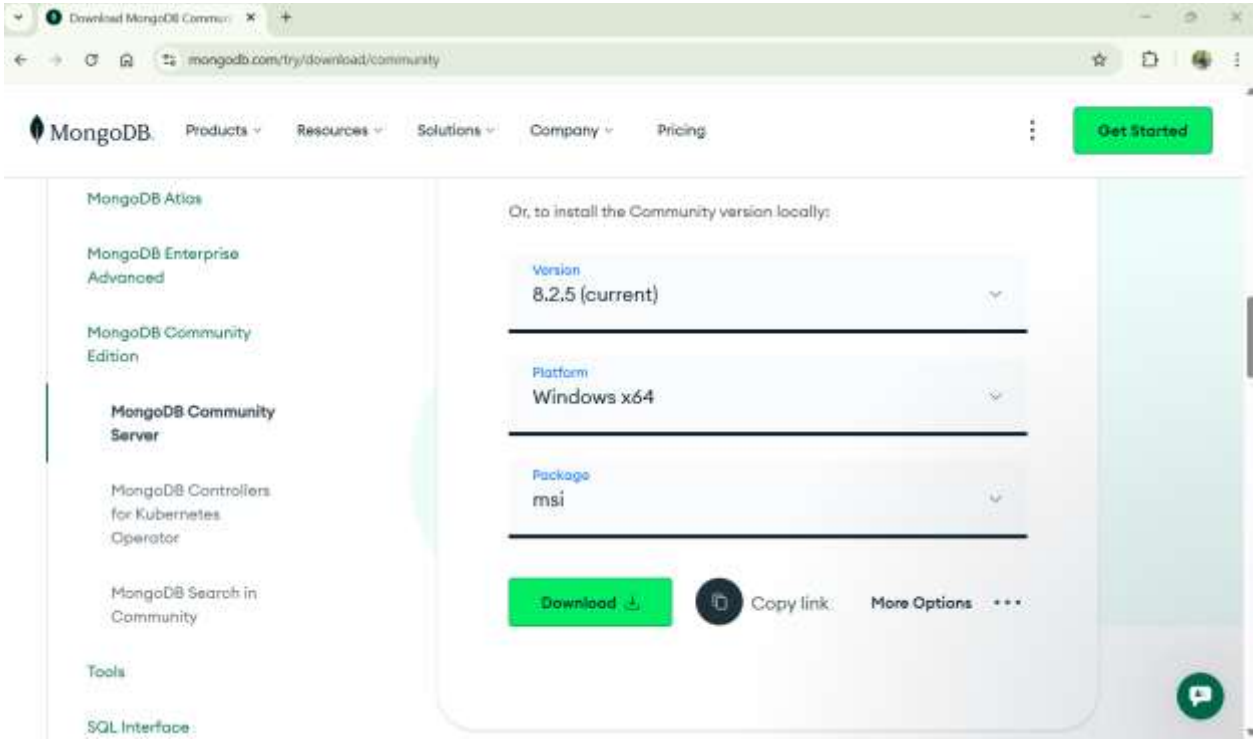
Description:

MongoDB for various things like building an application (including web and mobile), or analysis of data, or an administrator of a MongoDB database, in all these cases we need to interact with the MongoDB server to perform certain operations like entering new data into the application, updating data into the application, deleting data from the application, and reading the data of the application. MongoDB provides a set of some basic but most essential operations that will help you to easily interact with the MongoDB server and these operations are known as CRUD operations.

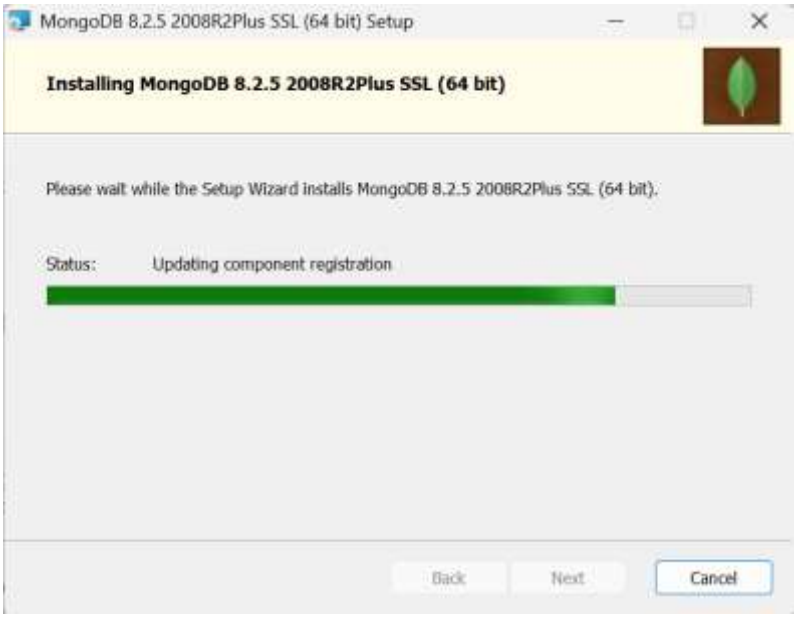
Steps:

Download MongoDB Server from the below given link as it is necessary in order to run the MongoDB Shell.

(Link: https://www.mongodb.com/try/download/community)



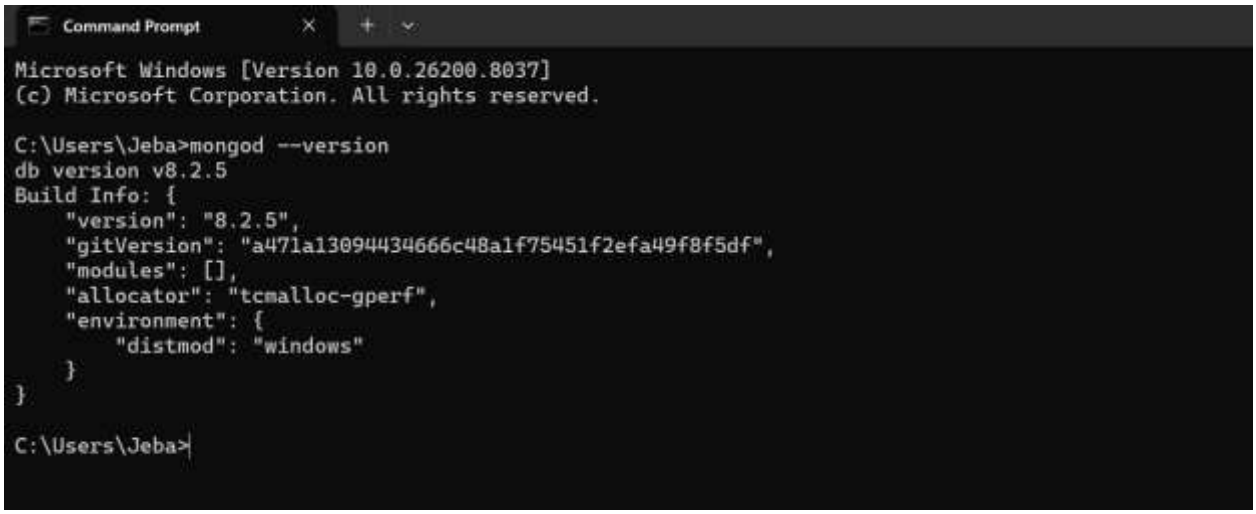
Once downloaded open the location and click on the downloaded file. Give all the necessary permissions and install the complete version of mongodb server.

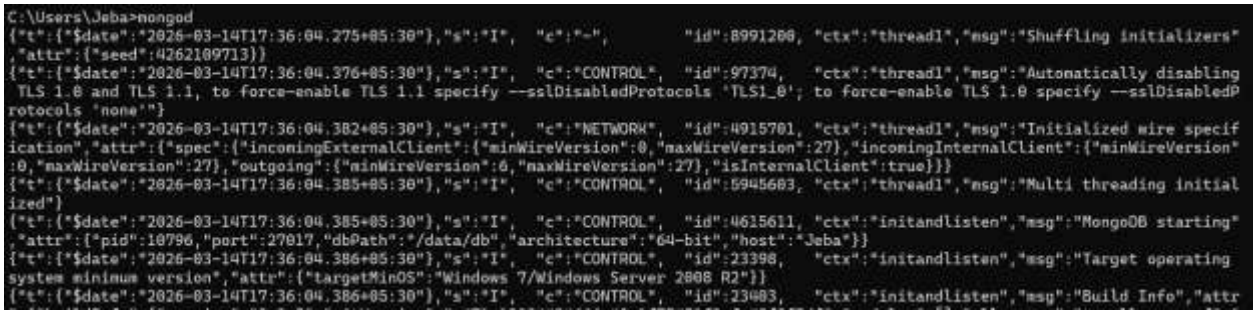


After complete installation add the directory to environment variable path by copying the mongodb bin path (C:\Program Files\MongoDB\Server\7.0\bin).

Now open the command prompt and type '**mongod –version'** to check if mongodb server is been setup properly or not.

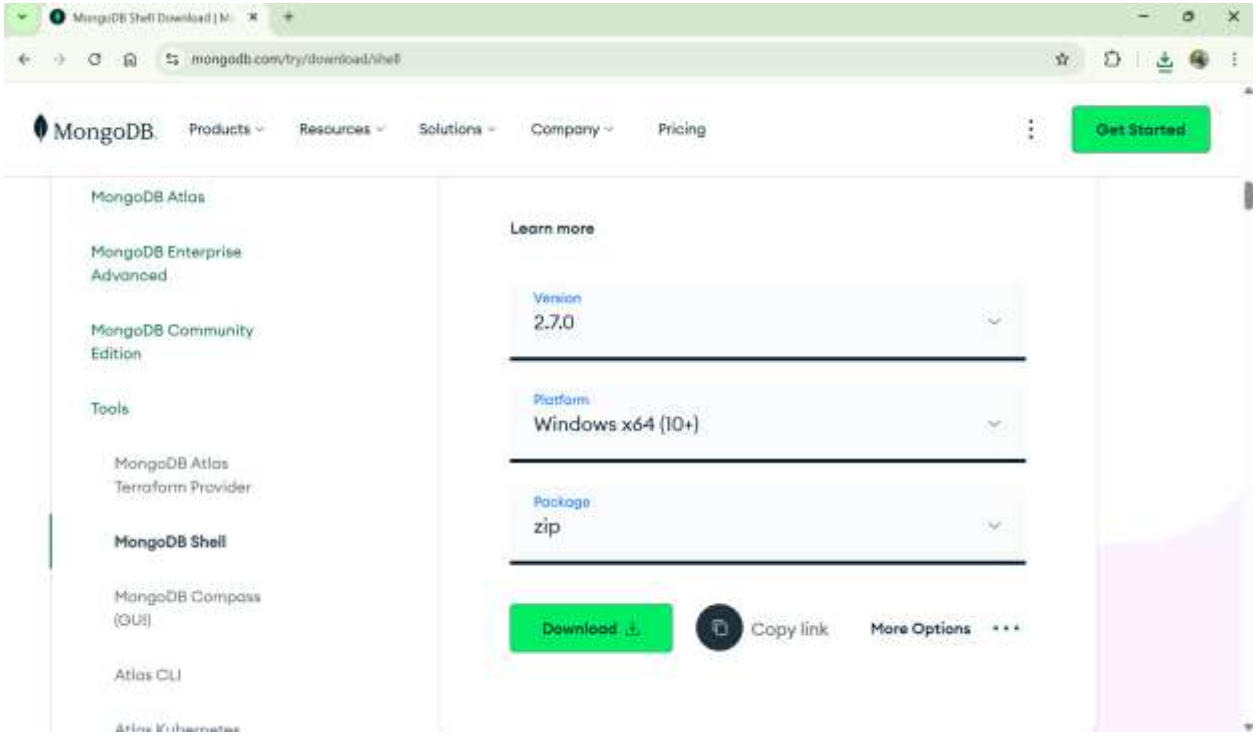
Note:- Before running mongodb shell always run the **'mongod'** command in the command prompt.





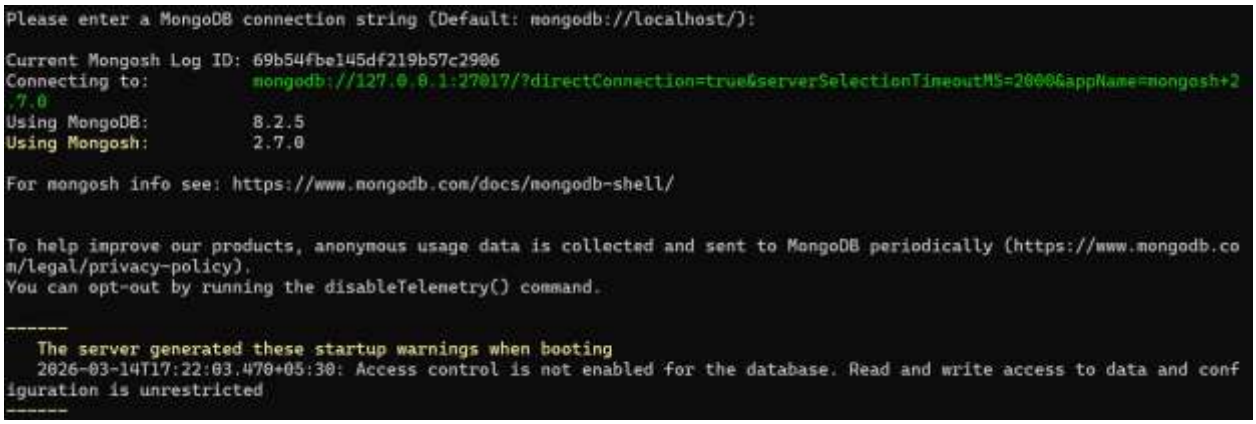
Step 1: Download MongoDB Shell

(Link: https://www.mongodb.com/try/download/shell)



Step 2: Extract it where it is been downloaded and install the application. Click and open the shell application

MongoDB Shell……



Note:- Sometime it will ask you to create database to store the data in that situation you need to create a data folder in the C: drive and inside data folder create a db folder.

Step 4: Create Operations –

The create or insert operations is used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database. You can perform, create operations using the following methods provided by the MongoDB:

It is used to insert a single document in the collection.

Code:-

use BDA

db.collection.insertOne()

Example 1: In this example, we are inserting details of a single student in the form of document in the student collection using db.collection.insertOne() method.

Output:

```
test> use BDA
switched to db BDA
BDA> db.student.insertOne({
| name: "Aloysius",
| age: 23,
| course: "MSc IT",
| year: 2
| })
{
  acknowledged: true,
  insertedId: ObjectId('69b54fd9145df219b57c2907')
}
```

Step 5: **Insert Many -**

It is used to insert multiple documents in the collection

Code:

db.collection.insertMany().

Example 2:

In this example, we are inserting details of the multiple students in the form of documents in the student collection using db.collection.insertMany() method.

```
BDA> db.student.insertMany([
| { name: "Rahul", age: 22, course: "MSc IT", year: 2 },
| { name: "Neha", age: 24, course: "MSc IT", year: 1 },
| { name: "Sumit", age: 23, course: "MSc IT", year: 2 }
| ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('69b54fdc145df219b57c2908'),
    '1': ObjectId('69b54fdc145df219b57c2909'),
    '2': ObjectId('69b54fdc145df219b57c290a')
  }
}
BDA>
```

Step 6: Read Operations –

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document. You can perform read operation using the following method provided by the MongoDB:

It is used to retrieve documents from the collection.

Code:-

db.collection.find().pretty()

Example :

In this example, we are retrieving the details of students from the student collection using db.collection.find() method.

```
BDA> db.student.find().pretty()
[
  {
    _id: ObjectId('69b54fd9145df219b57c2907'),
    name: 'Aloysius',
    age: 23,
    course: 'MSc IT',
    year: 2
  },
  {
    _id: ObjectId('69b54fdc145df219b57c2908'),
    name: 'Rahul',
    age: 22,
    course: 'MSc IT',
    year: 2
  },
  {
    _id: ObjectId('69b54fdc145df219b57c2909'),
    name: 'Neha',
    age: 24,
    course: 'MSc IT',
    year: 1
  },
  {
    _id: ObjectId('69b54fdc145df219b57c290a'),
    name: 'Sumit',
    age: 23,
    course: 'MSc IT',
    year: 2
  }
]
BDA>
```

Step 7: Update Operations –

The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB:

It is used to update a single document in the collection that satisfy the given criteria.

Code:

db.collection.updateOne()

Example 1:

In this example, we are updating the age of Sumit in the student collection using db.collection.updateOne() method.

```
BDA> db.student.updateOne(
| { name: "Sumit" },
| { $set: { age: 25 } }
| )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
BDA> db.student.find().pretty()
[
  {
    _id: ObjectId('69b54fd9145df219b57c2907'),
    name: 'Aloysius',
    age: 23,
    course: 'MSc IT',
    year: 2
  },
  {
    _id: ObjectId('69b54fdc145df219b57c2908'),
    name: 'Rahul',
    age: 22,
    course: 'MSc IT',
    year: 2
  },
  {
    _id: ObjectId('69b54fdc145df219b57c2909'),
    name: 'Neha',
    age: 24,
    course: 'MSc IT',
    year: 1
  },
  {
    _id: ObjectId('69b54fdc145df219b57c290a'),
    name: 'Sumit',
    age: 25,
    course: 'MSc IT',
    year: 2
  }
]
BDA>
```

It is used to update multiple documents in the collection that satisfy the given criteria.

Code:

db.collection.updateMany()

Example 2:

In this example, we are updating the year of course in all the documents in the student collection using db.collection.updateMany() method.

```
BDA> db.student.updateMany({},{$set:{year:2024}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
BDA> db.student.find().pretty()
[
  {
    _id: ObjectId('660d5cc431728aa2899f990a'),
    name: 'Neeraj',
    age: 30,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  },
  {
    _id: ObjectId('660d5de931728aa2899f990b'),
    name: 'Gautam',
    age: 21,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  },
  {
    _id: ObjectId('660d5de931728aa2899f990c'),
    name: 'Kiran',
    age: 22,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  }
]
BDA>
```

Step 8: Delete Operations –

The delete operation are used to delete or remove the documents from a collection. You can perform delete operations using the following methods provided by the MongoDB:

It is used to delete a single document from the collection that satisfy the given criteria.

Code:

db.collection.deleteOne()

Example 1:

In this example, we are deleting a document from the student collection using db.collection.deleteOne() method.

```
BDA> db.student.deleteOne({name : "Neeraj"})
{ acknowledged: true, deletedCount: 1 }
BDA> db.student.find().pretty()
[
  {
    _id: ObjectId('660d5de931728aa2899f990b'),
    name: 'Gautam',
    age: 21,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  },
  {
    _id: ObjectId('660d5de931728aa2899f990c'),
    name: 'Kiran',
    age: 22,
    branch: 'IT',
    course: 'MSC',
    mode: 'offline',
    paid: true,
    amount: 52000,
    year: 2024
  }
]
BDA>
```

It is used to delete multiple documents from the collection that satisfy the given criteria.

Code:

db.collection.deleteMany()

Example 2:

In this example, we are deleting all the documents from the student collection using db.collection.deleteMany() method.

```
BDA> db.student.deleteMany({})
{ acknowledged: true, deletedCount: 2 }
BDA> db.student.find().pretty()

BDA>
```

Practical 2

Aim: Implement Clustering and Associated algorithms

Software: R Studio

Download R Studio 4.3.0 from the official website (https://posit.co/download/rstudio-desktop/ Desktop - Posit) and install it

Description:

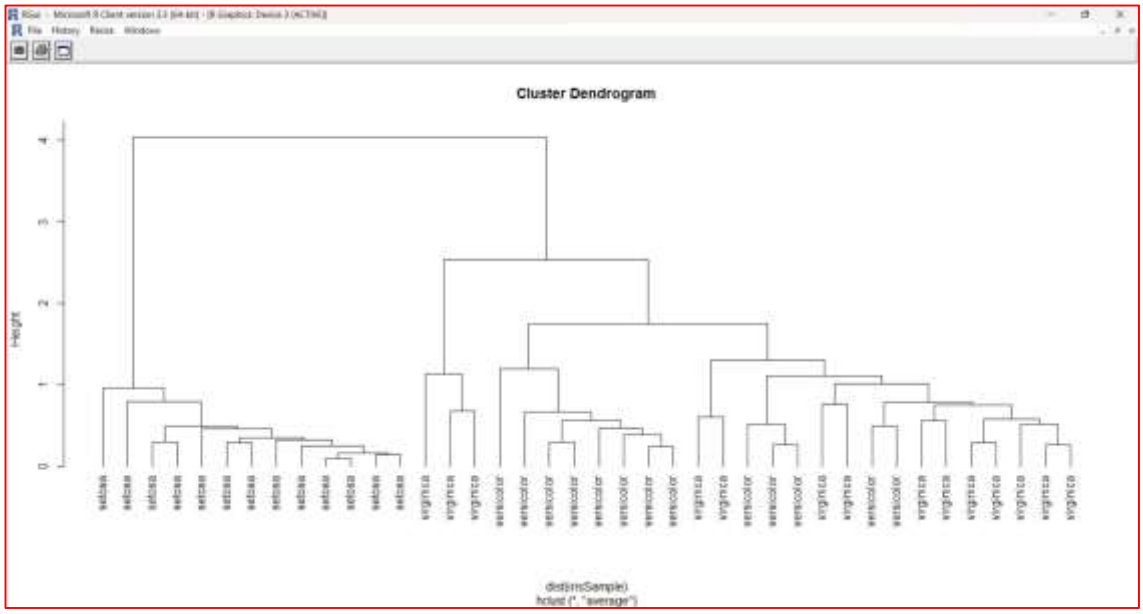
1. Data preparation
2. Assessing clustering tendency (i.e., the cluster ability of the data)
3. Defining the optimal number of clusters
4. Computing partitioning cluster analyses (e.g.: K-means, pam) or hierarchical clustering
5. Validating Clustering analyses

Code:

```
idx<-sample(1:dim(iris)[1],40)

irisSample<-iris[idx,]

irisSample$Species<-NULL

hc<-hclust(dist(irisSample),method="ave")

plot(hc,hang=-1,labels=iris$Species[idx])
```

```
> idx<-sample(1:dim(iris)[1],40)
> irisSample<-iris[idx,]
> irisSample$Species<-NULL
> hc<-hclust(dist(irisSample),method="ave")
> plot(hc,hang=-1,labels=iris$Species[idx])
>
```

Output:



Association Rule Mining:

Code:

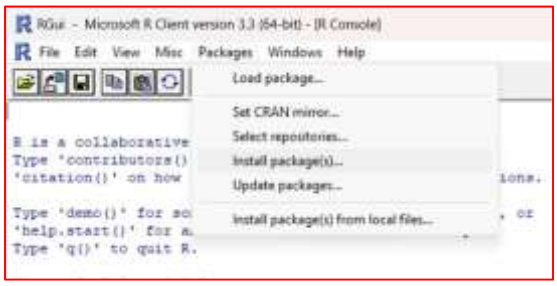
>#load data

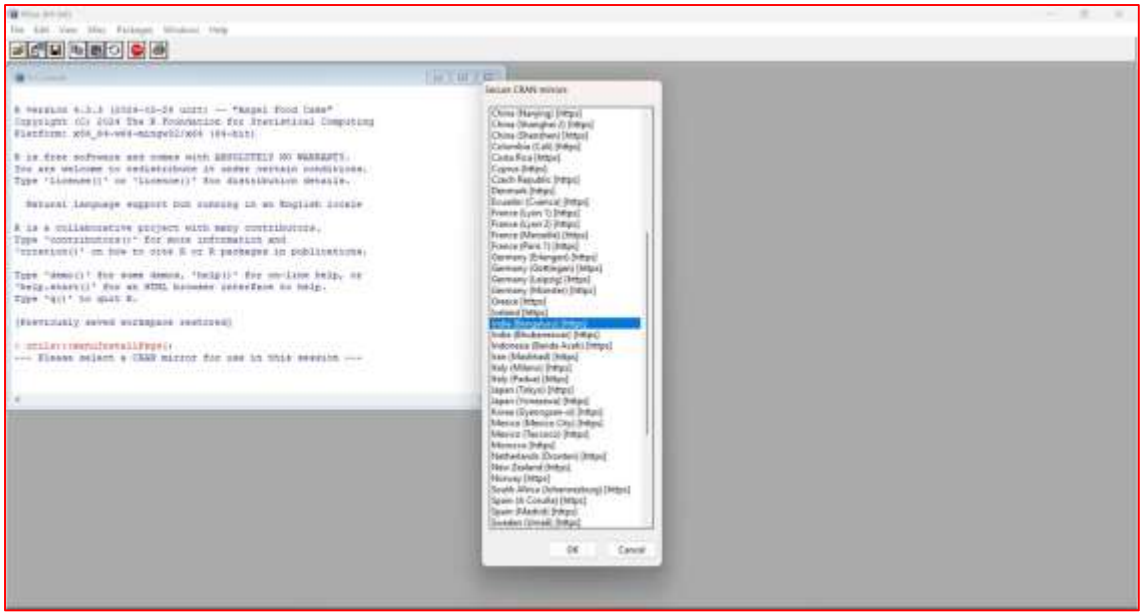
>load("D:/titanic.raw.rdata")

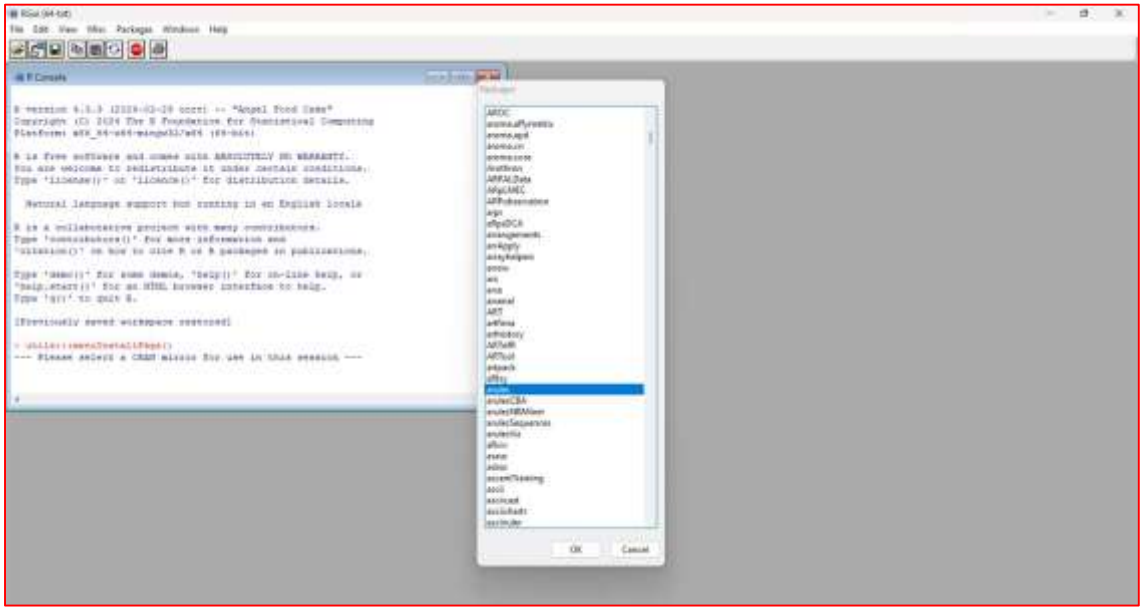
>str(titanic.raw)

```
> load("D:/titanic.raw.rdata")
> str(titanic.raw)
'data.frame':   2201 obs. of  4 variables:
 $ Class   : Factor w/ 4 levels "1st","2nd","3rd",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ Sex     : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ Age     : Factor w/ 2 levels "Adult","Child": 2 2 2 2 2 2 2 2 2 2 ...
 $ Survived: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Go to “Packages” select “install packages” select India (Bengaluru) [https] and select “arules”. Click on yes and then it will install the packages.







Code:

```
library(arules)

# find association rules with default settings

rules <- apriori(titanic.raw)
```

inspect(rules)

```
> library(arules)
Loading required package: Matrix

Attaching package: 'arules'

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

    abbreviate, write
```

```
> rules <- apriori(titanic.raw)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules TRUE

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

Absolute minimum support count: 220

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[10 item(s), 2201 transaction(s)] done [0.00s].
sorting and recoding items ... [9 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [27 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```
> inspect(rules)
     lhs                                   rhs           support   confidence coverage  lift     count
[1]  {}                                 => {Age=Adult}   0.9504771 0.9504771  1.0000000 1.0000000 2092
[2]  {Class=2nd}                        => {Age=Adult}   0.1185825 0.9157895  0.1294866 0.9635051  261
[3]  {Class=1st}                        => {Age=Adult}   0.1449341 0.9815385  0.1476602 1.0326798  319
[4]  {Sex=Female}                       => {Age=Adult}   0.1930940 0.9042553  0.2135393 0.9513700  425
[5]  {Class=3rd}                        => {Age=Adult}   0.2848705 0.8881020  0.3207633 0.9343750  627
[6]  {Survived=Yes}                     => {Age=Adult}   0.2971377 0.9198312  0.3230350 0.9677574  654
[7]  {Class=Crew}                       => {Sex=Male}    0.3916402 0.9740113  0.4020900 1.2384742  862
[8]  {Class=Crew}                       => {Age=Adult}   0.4020900 1.0000000  0.4020900 1.0521033  885
[9]  {Survived=No}                      => {Sex=Male}    0.6197183 0.9154362  0.6769650 1.1639949 1364
[10] {Survived=No}                      => {Age=Adult}   0.6533394 0.9651007  0.6769650 1.0153856 1438
[11] {Sex=Male}                         => {Age=Adult}   0.7573830 0.9630272  0.7864607 1.0132040 1667
[12] {Sex=Female, Survived=Yes}         => {Age=Adult}   0.1435711 0.9186047  0.1562926 0.9664669  316
[13] {Class=3rd, Sex=Male}              => {Survived=No} 0.1917310 0.8274510  0.2317129 1.2222950  422
[14] {Class=3rd, Survived=No}           => {Age=Adult}   0.2162653 0.9015152  0.2398910 0.9484870  476
[15] {Class=3rd, Sex=Male}              => {Age=Adult}   0.2099046 0.9058824  0.2317129 0.9530818  462
[16] {Sex=Male, Survived=Yes}           => {Age=Adult}   0.1535666 0.9209809  0.1667424 0.9689670  338
[17] {Class=Crew, Survived=No}          => {Sex=Male}    0.3044071 0.9955423  0.3057701 1.2658514  670
[18] {Class=Crew, Survived=No}          => {Age=Adult}   0.3057701 1.0000000  0.3057701 1.0521033  673
[19] {Class=Crew, Sex=Male}             => {Age=Adult}   0.3916402 1.0000000  0.3916402 1.0521033  862
[20] {Class=Crew, Age=Adult}            => {Sex=Male}    0.3916402 0.9740113  0.4020900 1.2384742  862
[21] {Sex=Male, Survived=No}            => {Age=Adult}   0.6038164 0.9743402  0.6197183 1.0251065 1329
[22] {Age=Adult, Survived=No}           => {Sex=Male}    0.6038164 0.9242003  0.6533394 1.1751385 1329
[23] {Class=3rd, Sex=Male, Survived=No} => {Age=Adult}   0.1758292 0.9170616  0.1917310 0.9648435  387
[24] {Class=3rd, Age=Adult, Survived=No}=> {Sex=Male}    0.1758292 0.8130252  0.2162653 1.0337773  387
[25] {Class=3rd, Sex=Male, Age=Adult}   => {Survived=No} 0.1758292 0.8376623  0.2099046 1.2373791  387
[26] {Class=Crew, Sex=Male, Survived=No}=> {Age=Adult}   0.3044071 1.0000000  0.3044071 1.0521033  670
[27] {Class=Crew, Age=Adult, Survived=No}=> {Sex=Male}   0.3044071 0.9955423  0.3057701 1.2658514  670
>
```

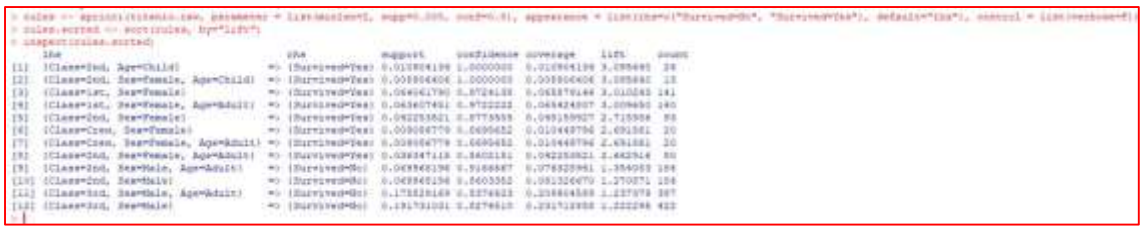
Code:

>\# rules with rhs containing "Survived" only

>rules <- apriori(titanic.raw, parameter = list(minlen=2, supp=0.005, conf=0.8), appearance = list(rhs=c("Survived=No", "Survived=Yes"), default="lhs"), control = list(verbose=F))

>rules.sorted <- sort(rules, by="lift")

>inspect(rules.sorted)



Pruning Redundant Rules

In the above result, rule 2 provides no extra knowledge in addition to rule 1, since rules 1 tells us that all 2nd-class children survived. Generally speaking, when a rule (such as rule 2) is a super rule of another rule (such as rule 1) and the former has the same or a lower lift, the former rule (rule 2) is considered to be redundant. Below we prune redundant rules.

Code:

>\# find redundant rules

>subset.matrix <- is.subset(rules.sorted, rules.sorted)

>subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA

>redundant <- colSums(subset.matrix, na.rm=T) >= 1

>which(redundant)

>\# remove redundant rules

>rules.pruned <- rules.sorted[!redundant]

>inspect(rules.pruned)

Output:

Practical 3

Aim: To implement Linear Regression.

Software: Google Colab

Datasets: data.csv

Step 1: Import libraries and dataset.

Import the important libraries and the dataset we are using to perform Polynomial Regression.

Step 2: Dividing the dataset into 2 components.

Divide dataset into two components that is X and y. X will contain the Column between 1 and 2. y will contain the 2 column.

Step 3: Fitting Linear Regression to the dataset

Fitting the linear Regression model On two components.

Step 4: Fitting Polynomial Regression to the dataset

Fitting the Polynomial Regression model on two components X and y.

Step 5: In this step we are Visualising the Linear Regression results using scatter plot.

	A	B	C
1	sno	Temperat	Pressure
2	1	0	0.0002
3	2	20	0.0012
4	3	40	0.006
5	4	60	0.03
6	5	80	0.09
7	6	100	0.27

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
df= pd.read_csv('/content/data.csv')
display(df.head())
```

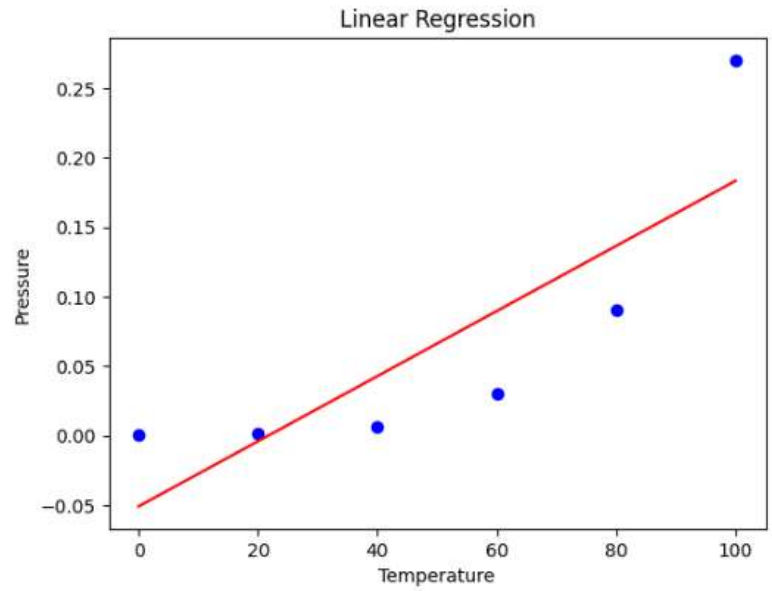
	sno	Temperature	Pressure
0	1	0	0.0002
1	2	20	0.0012
2	3	40	0.0060
3	4	60	0.0300
4	5	80	0.0900

```
x=df.iloc[:,1:2].values
y=df.iloc[:,2].values
```

```
from sklearn.linear_model import LinearRegression
lin=LinearRegression()
lin.fit(x,y)
```

```
LinearRegression
LinearRegression()
```

```
plt.scatter(x,y,color='blue')
plt.plot(x,lin.predict(x),color='red')
plt.title('Linear Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')
plt.show()
```



Practical 4

Aim: To Implement Time Series

Software: R Studio

Description:

Time Series

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame. The time series object is created by using the ts () function.

Syntax

The basic syntax for ts() function in time series analysis is – timeseries.object.name <- ts(data, start, end, frequency) Following is the description of the parameters used –

- data is a vector or matrix containing the values used in the time series.
- start specifies the start time for the first observation in time series.
- end specifies the end time for the last observation in time series.
- frequency specifies the number of observations per unit time. Except the parameter "data" all other parameters are optional.

Example:

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.

Code:

```
# Get the data points in form of a R vector.

rainfall <-

c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

# Convert it to a time series object.

rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)

# Print the timeseries data.

print(rainfall.timeseries)
```

Give the chart file a name.

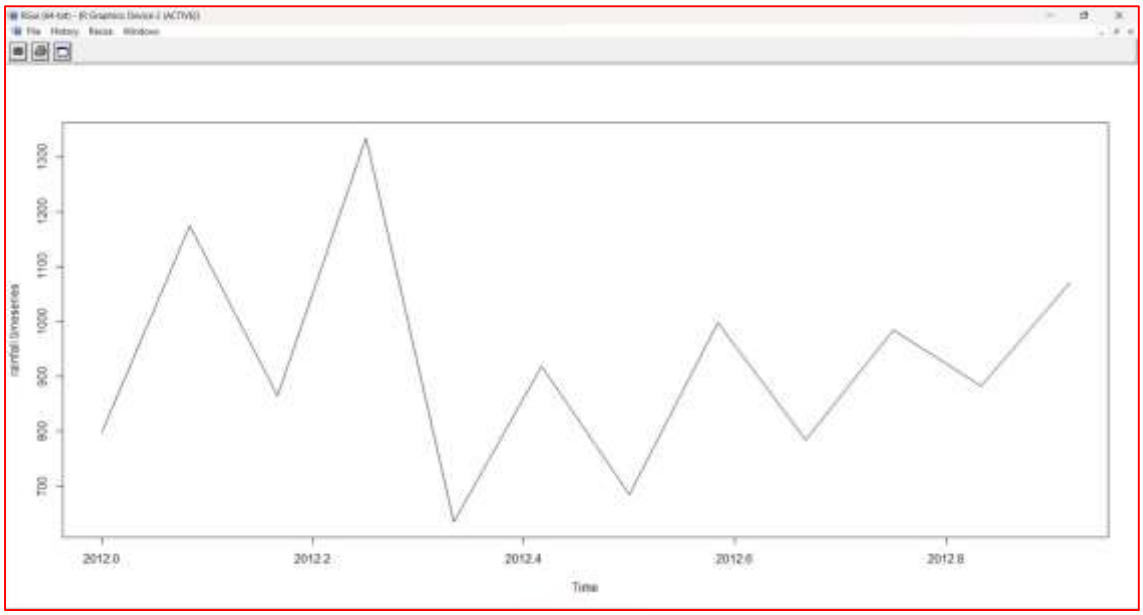
png(file = "rainfall.png")

Plot a graph of the time series.

plot(rainfall.timeseries) dev.off() plot(rainfall.timeseries)

Output:

```
> rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
> rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
> print(rainfall.timeseries)
       Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct    Nov    Dec
2012 799.0 1174.8  865.1 1334.6  635.4  918.5  685.5  998.6  784.2  985.0  882.8 1071.0
> png(file = "rainfall.png")
> plot(rainfall.timeseries)
> dev.off()
null device
          1
> plot(rainfall.timeseries)
>
```



Different Time Intervals

The value of the frequency parameter in the ts() function decides the time intervals at which the data points are measured. A value of 12 indicates that the time series is for 12 months. Other values and its meaning is as below –

- frequency = 12 pegs the data points for every month of a year.
- frequency = 4 pegs the data points for every quarter of a year.
- frequency = 6 pegs the data points for every 10 minutes of an hour.
- frequency = 24*6 pegs the data points for every 10 minutes of a day. Multiple Time Series

We can plot multiple time series in one chart by combining both the series into a matrix.

Code:

```
# Get the data points in form of a R vector.
rainfall1 <-
c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
rainfall2 <-
c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337. 8)
# Convert them to a matrix.
combined.rainfall <- matrix(c(rainfall1,rainfall2),nrow = 12)
# Convert it to a time series object.
rainfall.timeseries <- ts(combined.rainfall,start = c(2012,1),frequency = 12)
# Print the timeseries data.
print(rainfall.timeseries)
# Give the chart file a name.
png(file = "rainfall_combined.png")
# Plot a graph of the time series.
plot(rainfall.timeseries, main = "Multiple Time Series")
# Save the file.
dev.off()
plot(rainfall.timeseries, main = "Multiple Time Series")
```

Output:

```
> rainfall1 <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
> rainfall2 <- c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337.8)
> combined.rainfall <- matrix(c(rainfall1,rainfall2),nrow = 12)
> rainfall.timeseries <- ts(combined.rainfall,start = c(2012,1),frequency = 12)
> print(rainfall.timeseries)
         Series 1 Series 2
Jan 2012    799.0    655.0
Feb 2012   1174.8   1306.9
Mar 2012    865.1   1323.4
Apr 2012   1334.6   1172.2
May 2012    635.4    562.2
Jun 2012    918.5    824.0
Jul 2012    685.5    822.4
Aug 2012    998.6   1265.5
Sep 2012    784.2    799.6
Oct 2012    985.0   1105.6
Nov 2012    882.8   1106.7
Dec 2012   1071.0   1337.8
> png(file = "rainfall_combined.png")
> plot(rainfall.timeseries, main = "Multiple Time Series")
> dev.off()
windows
      2
> plot(rainfall.timeseries, main = "Multiple Time Series")
>
```