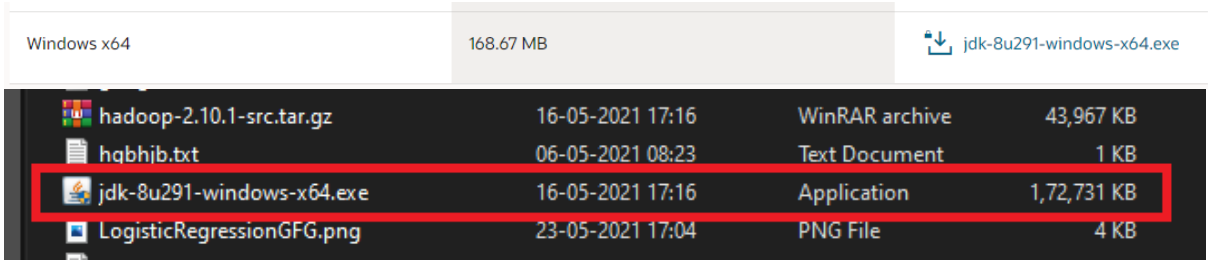
INDEX

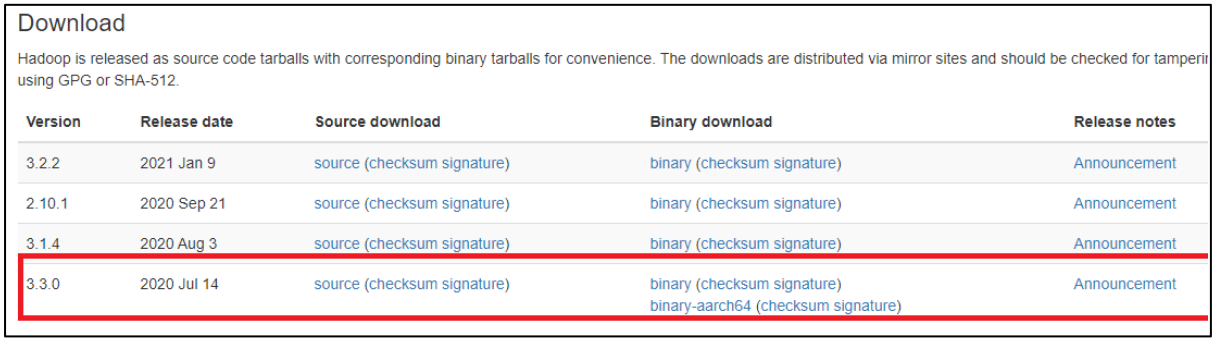
Sr No.	Practical	Date	Sign
1	Install, configure and run Hadoop and HDFS		
2	Implementing distinct word count problem using Map-Reduce.		
3	Implement a MapReduce program that processes a weather dataset.		
4	Implement an application that stores big data in HBase / MongoDB and manipulate it using R / Python.		
5	Configure the Hive and implement the application in Hive.		
6	Write a program to illustrate the working of JAQL.		
7	Implement Decision tree classification techniques.		
8	Implement SVM classification techniques.		
9	Write a Program showing implementation of Regression model.		
10	Write a Program showing Clustering.		

PRACTICAL NO: 1

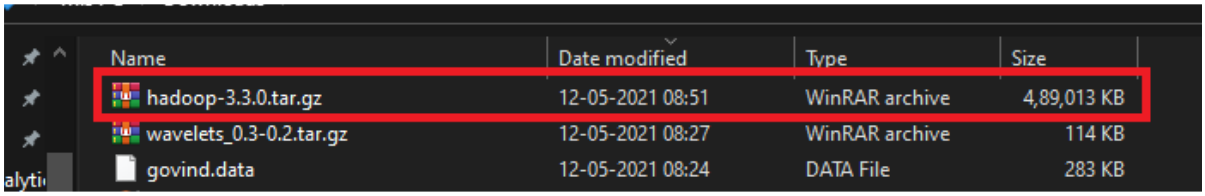
Aim: Install, configure and run Hadoop and HDFS
Description: Hadoop Installation.
Step 1: download java jdk first .the package size 168.67MB



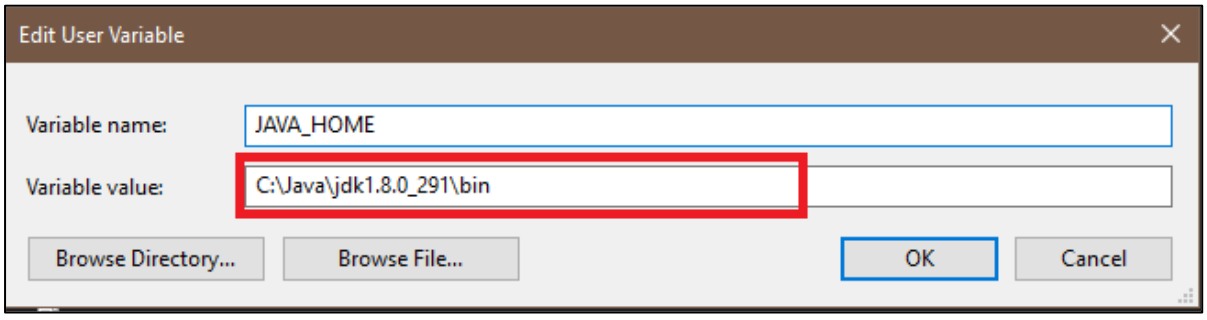
Step 2: download Hadoop binaries from the official website. The binary package size is about 342 MB.



Step 3: After finishing the file download, we should unpack the package using 7zip int two steps. First, we should extract the hadoop-3.2.1.tar.gz library, and then, we should unpack the extracted tar file:



Step 4: When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.

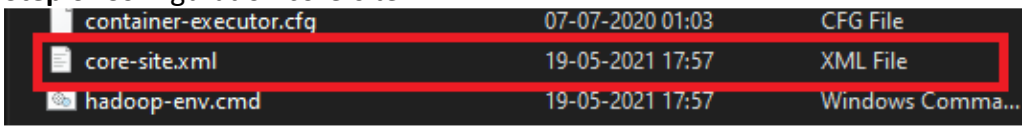


Step 5: Check the version of java.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19041.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files and annotation process
  -cp <path>                 Specify where to find user class files and annotation process
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compilation is
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; by
ss
```

```
C:\Users\hp>java -version
java version "1.8.0_291"
Java(TM) SE Runtime Environment (build 1.8.0_291-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.291-b10, mixed mode)
```

Step 6: Configuration core-site.xml.

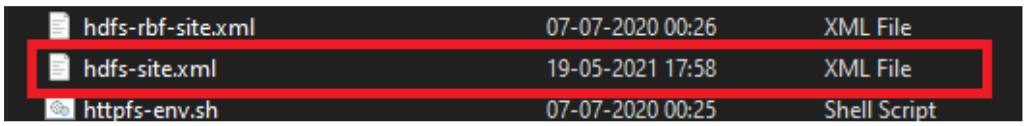


core-site.xml

C: > hadoop > etc > hadoop > core-site.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3
4   <configuration>
5
6   <property>
7       <name>fs.deafultFS</name>
8       <value>hdfs://localhost:9000</value>
9   <property>
10  </configuration>
```

Step 7: Configuration core-site.xml



core-site.xml ● hdfs-site.xml ●

C: > hadoop > etc > hadoop > hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
<property>
        <name>dfs.namenode.name.dir</name>
        <value>C:\hadoop\data\namenode</value>

</property>
<property>
        <name>dfs.namenode.data.dir</name>
        <value>C:\hadoop\data\datanode</value>
</property>
</configuration>
```

Step 8: Configuration core-site.xml.

mapred-queues.xml.template 07-07-2020 01:04 TEMPLATE File

mapred-site.xml 19-05-2021 17:58 XML File

ssl-client.xml.example 07-07-2020 00:16 EXAMPLE File

File Edit Selection View Go Run Terminal Help • mapre

core-site.xml ● hdfs-site.xml ● mapred-site.xml ●

C: > hadoop > etc > hadoop > mapred-site.xml

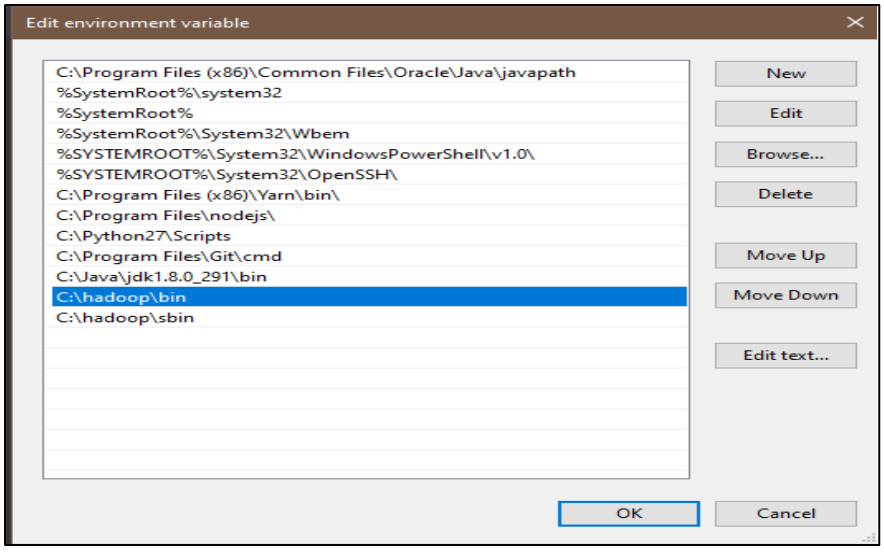
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
</configuration>
```

Step 9: Configuration core-site.xml.

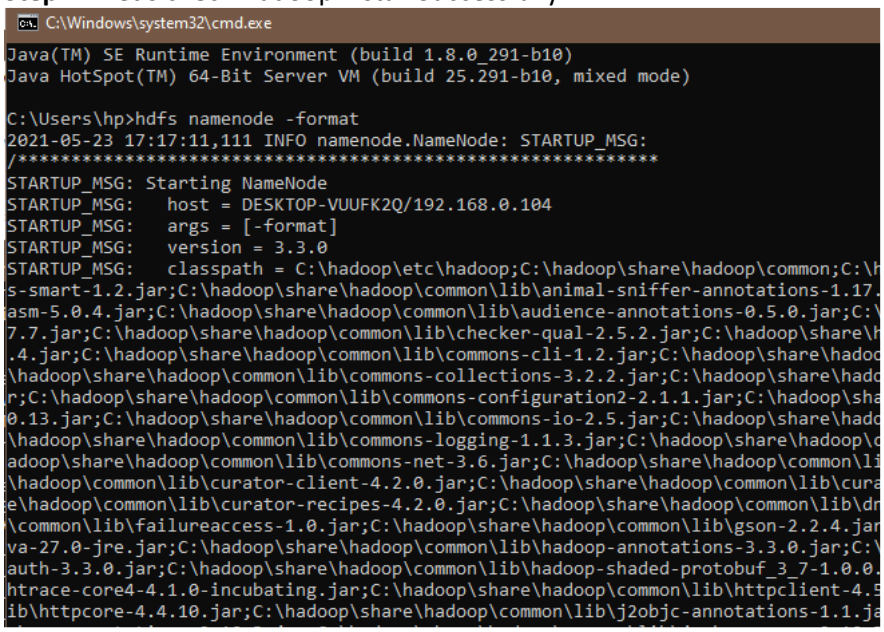
yarnservice-log4j.properties 07-07-2020 01:03 PROPERTIES File

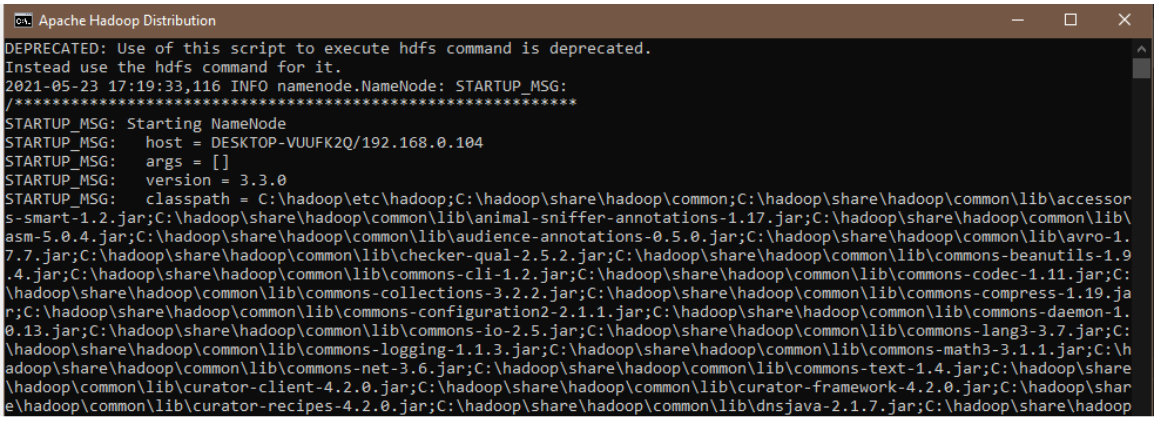
yarn-site.xml 19-05-2021 17:58 XML File

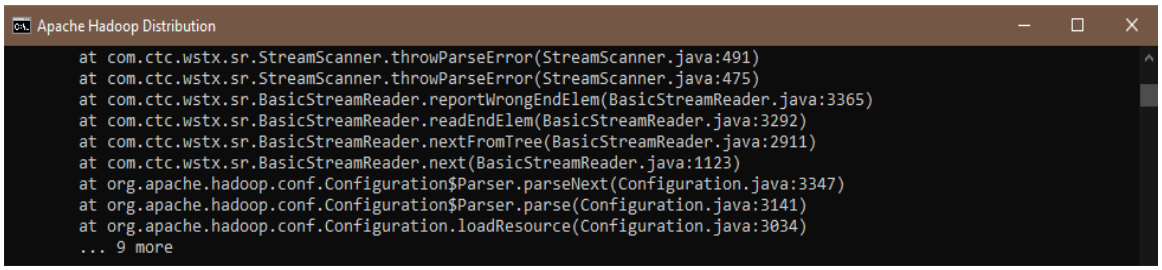
Step 10: When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.



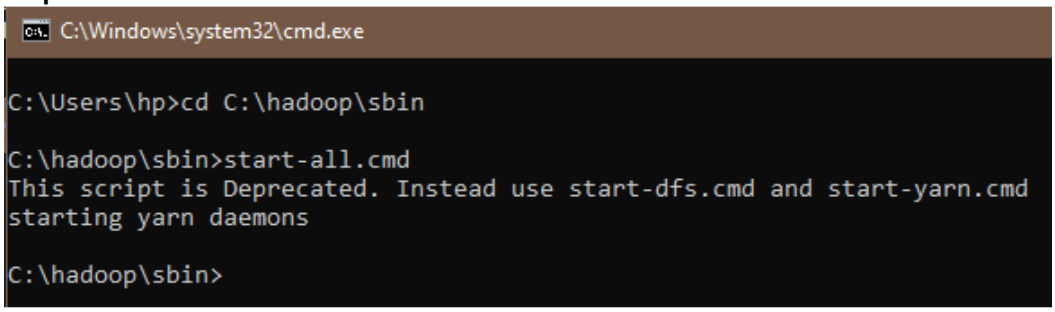
Step 11: let's check Hadoop install Successfully.







Step 12: Let check bin



PRACTICAL NO: 2

Aim: Implementing distinct word count problem using Map-Reduce.

The function of the mapper is as follows:

- Create a Int Writable variable 'one' with value as 1.
- Convert the input line in Text type to a String.
- Use a tokenizer to split the line into words.
- Iterate through each word and a form key value pairs as Assign each work from the tokenizer (of String type) to a Text 'word'.
- Form key value pairs for each word as < word,one > and push it to the output collector.

The function of Sort and Group:

After this, "aggregation" and "Shuffling and Sorting" done by framework. Then Reducers task these final pair to produce output.

The function of the reducer is as follows

- Initialize a variable 'sum' as 0.
- Iterate through all the values with respect to a key and sum up all of them.
- Push to the output collector the Key and the obtained sum as value.

For Example:

For the given sample input1 data file (input1.txt : Hello World Bye World) mapper emits:
<Hello, 1>
<World, 1>
<Bye, 1>
<World, 1>
The second input2 data file (input2.txt : Hello Hadoop Goodbye Hadoop) mapper emits:
<Hello, 1>
<Hadoop, 1>
<Goodbye, 1>
<Hadoop, 1>
WordCount also specifies a combiner. Hence, the output of each map is passed through the local combiner (which is same as the Reducer as per the job configuration) for local aggregation, after being sorted on the keys.
The output of the first map:
<Bye, 1>
<Hello, 1>
<World, 2>
The output of the second map:
<Goodbye, 1>
<Hadoop, 2>
<Hello, 1>
The Reducer implementation via the reduce method just sums up the values, which are the occurence counts for each key (i.e. words in this example). Thus the output of the job is:
<Bye, 1>
<Goodbye, 1>
<Hadoop, 2>
<Hello, 2>
<World, 2>

PRACTICAL NO: 3

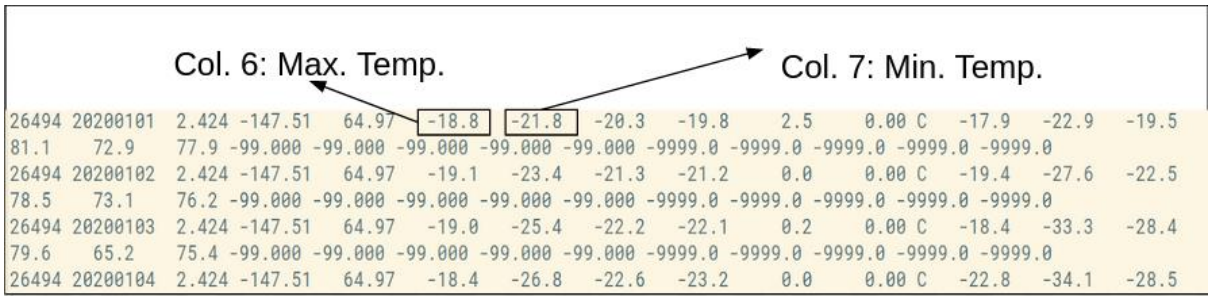
Aim: Implement an MapReduce program that processes a weather dataset.

Step 1:

I have selected *CRND0103-2020-AK_Fairbanks_11_NE.txt* dataset for analysis of hot and cold days in Fairbanks, Alaska.

Step 2:

Below is the example of our dataset where column 6 and column 7 is showing Maximum and Minimum temperature, respectively.




```
26494 20200101  2.424 -147.51   64.97   -18.8   -21.8   -20.3   -19.8     2.5     0.00 C   -17.9   -22.9   -19.5
81.1    72.9    77.9 -99.000 -99.000 -99.000 -99.000 -99.000 -9999.0 -9999.0 -9999.0 -9999.0 -9999.0
26494 20200102  2.424 -147.51   64.97   -19.1   -23.4   -21.3   -21.2     0.0     0.00 C   -19.4   -27.6   -22.5
78.5    73.1    76.2 -99.000 -99.000 -99.000 -99.000 -99.000 -9999.0 -9999.0 -9999.0 -9999.0 -9999.0
26494 20200103  2.424 -147.51   64.97   -19.0   -25.4   -22.2   -22.1     0.2     0.00 C   -18.4   -33.3   -28.4
79.6    65.2    75.4 -99.000 -99.000 -99.000 -99.000 -99.000 -9999.0 -9999.0 -9999.0 -9999.0 -9999.0
26494 20200104  2.424 -147.51   64.97   -18.4   -26.8   -22.6   -23.2     0.0     0.00 C   -22.8   -34.1   -28.5
```

Step 3:

First Open **Eclipse** -> then select **File -> New -> Java Project** ->Name it **MyProject** -> then select **use an execution environment** -> choose **JavaSE-1.8** then **next** -> **Finish**.

Step 4:

In this Project Create Java class with name **MyMaxMin** -> then click **Finish**.

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;

public class MyMaxMin {
  // Mapper

  public static class MaxTemperatureMapper extends
       Mapper<LongWritable, Text, Text, Text> {
  // the data in our data set with
  // this value is inconsistent data
  public static final int MISSING = 9999;
```

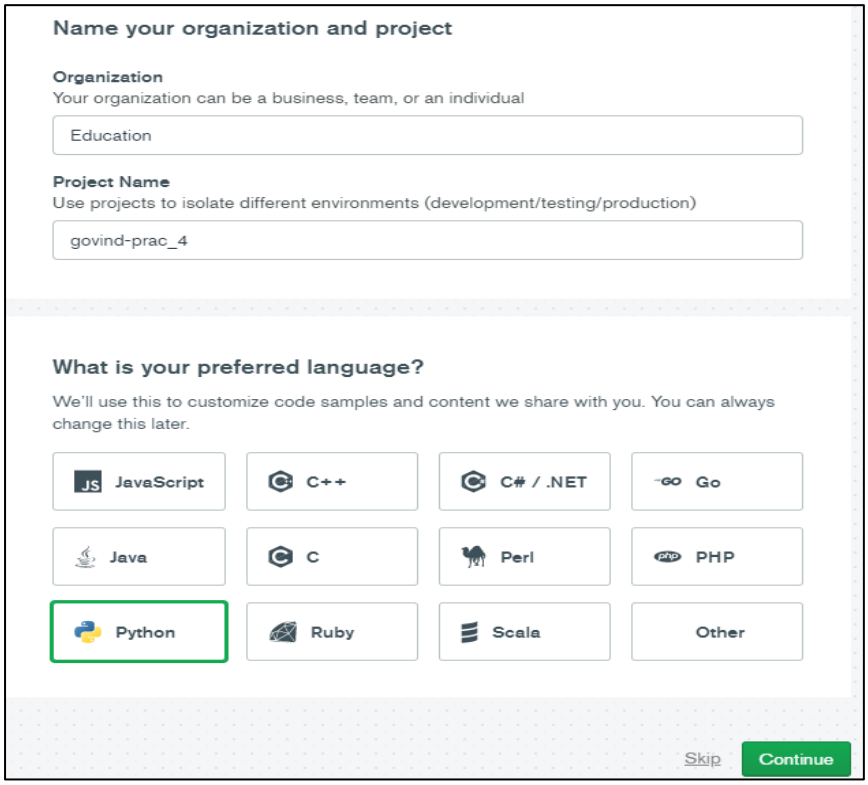
```
    @Override
    public void map(LongWritable arg0, Text Value, Context context) throws IOException,
InterruptedException {

    // Convert the single row(Record) to
    // String and store it in String
    // variable name line
    String line = Value.toString();

      // Check for the empty line
      if (!(line.length() == 0)) {
        // from character 6 to 14 we have
        // the date in our dataset
        String date = line.substring(6, 14);
        // similarly we have taken the maximum
        // temperature from 39 to 45 characters
        float temp_Max = Float.parseFloat(line.substring(39, 45).trim());
        // similarly we have taken the minimum
        // temperature from 47 to 53 characters
        float temp_Min = Float.parseFloat(line.substring(47, 53).trim());
        // if maximum temperature is
        // greater than 30, it is a hot day
        if (temp_Max > 30.0) {
          // Hot day
          context.write(new Text("The Day is Hot Day :" + date),
                        new Text(String.valueOf(temp_Max)));
        }
        // if the minimum temperature is
        // less than 15, it is a cold day
        if (temp_Min < 15) {
          // Cold day
          context.write(new Text("The Day is Cold Day :" + date),
              new Text(String.valueOf(temp_Min)));
    } } } }

// Reducer
  public static class MaxTemperatureReducer extends
      Reducer<Text, Text, Text, Text> {

    public void reduce(Text Key, Iterator<Text> Values, Context context)
        throws IOException, InterruptedException {
      // putting all the values in
      // temperature variable of type String
      String temperature = Values.next().toString();
      context.write(Key, new Text(temperature));
    }
  }
```

```
    public static void main(String[] args) throws Exception {
        // reads the default configuration of the
        // cluster from the configuration XML files
        Configuration conf = new Configuration();
        // Initializing the job with the
        // default configuration of the cluster
        Job job = new Job(conf, "weather example");
        // Assigning the driver class name
        job.setJarByClass(MyMaxMin.class);
        // Key type coming out of mapper
        job.setMapOutputKeyClass(Text.class);
        // value type coming out of mapper
        job.setMapOutputValueClass(Text.class);
        // Defining the mapper class name
        job.setMapperClass(MaxTemperatureMapper.class);
        // Defining the reducer class name
        job.setReducerClass(MaxTemperatureReducer.class);
        // Defining input Format class which is
        // responsible to parse the dataset
        // into a key value pair
        job.setInputFormatClass(TextInputFormat.class);
        // Defining output Format class which is
        // responsible to parse the dataset
        // into a key value pair
        job.setOutputFormatClass(TextOutputFormat.class);
        // setting the second argument
        // as a path in a path variable
        Path OutputPath = new Path(args[1]);
        // Configuring the input path
        // from the filesystem into the job
        FileInputFormat.addInputPath(job, new Path(args[0]));
        // Configuring the output path from
        // the filesystem into the job
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // deleting the context path automatically
        // from hdfs so that we don't have
        // to delete it explicitly
        OutputPath.getFileSystem(conf).delete(OutputPath);
        // exiting the job only if the
        // flag value becomes false
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

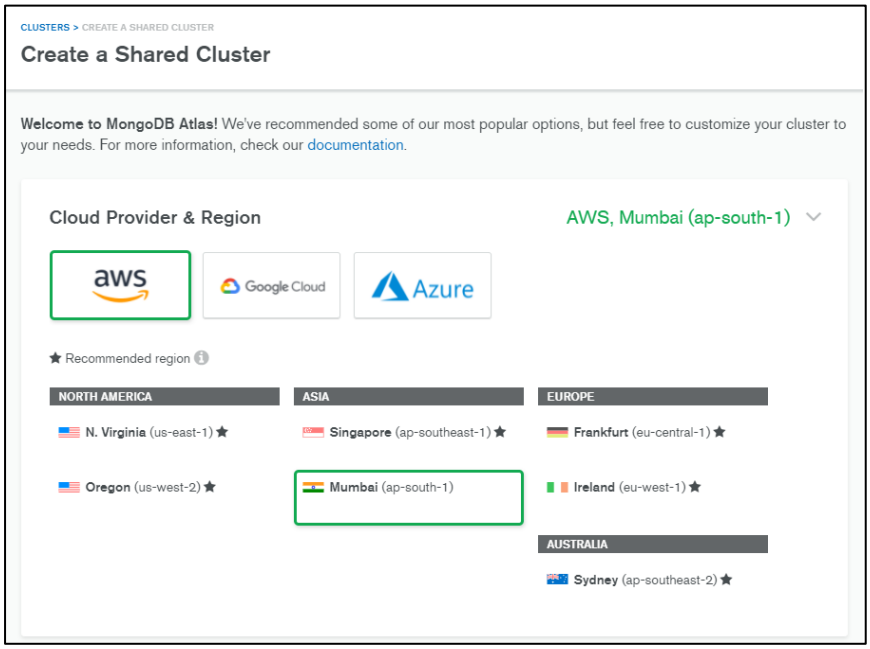
PRACTICAL NO: 4

Aim: Implement an application that stores big data in HBase / MongoDB and manipulate it using R / Python.

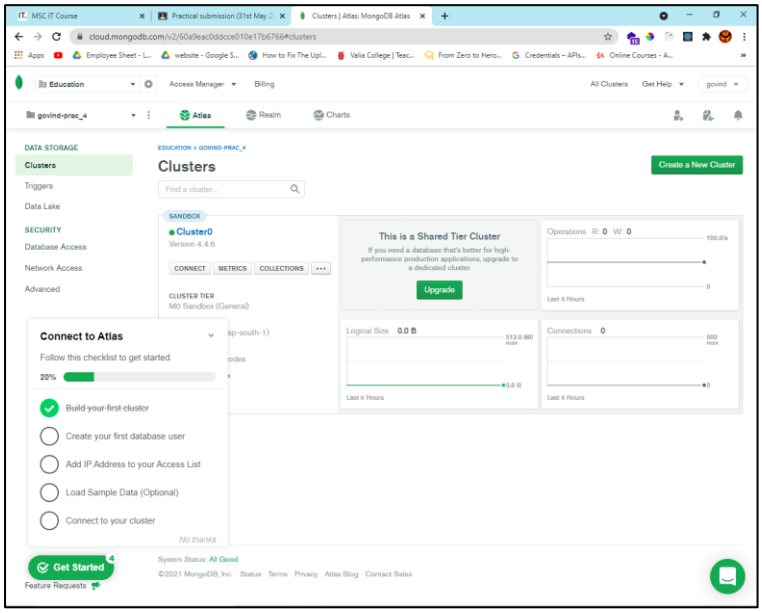
Description: MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License



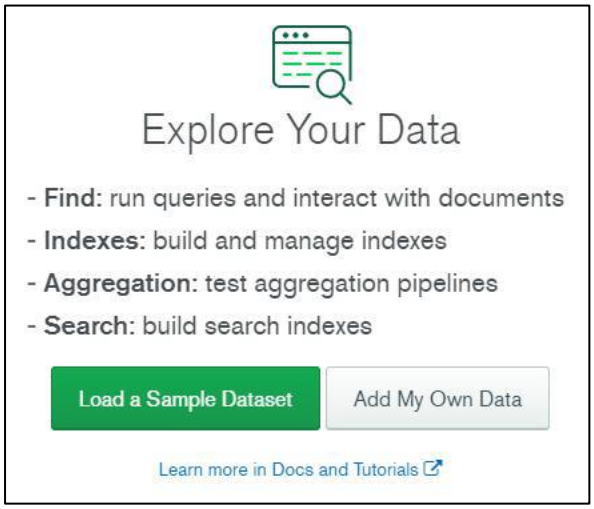
Step 1: Sign up and create a cluster.



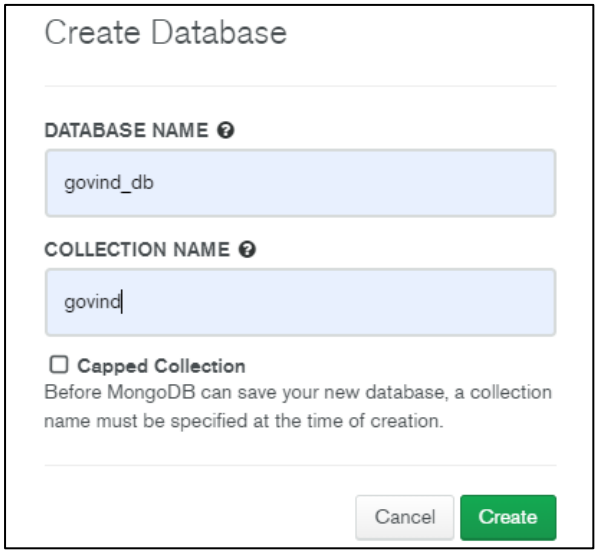
This is the home page of MongoDB Atlas.



Step 2: Click on collections to create and view existing databases.

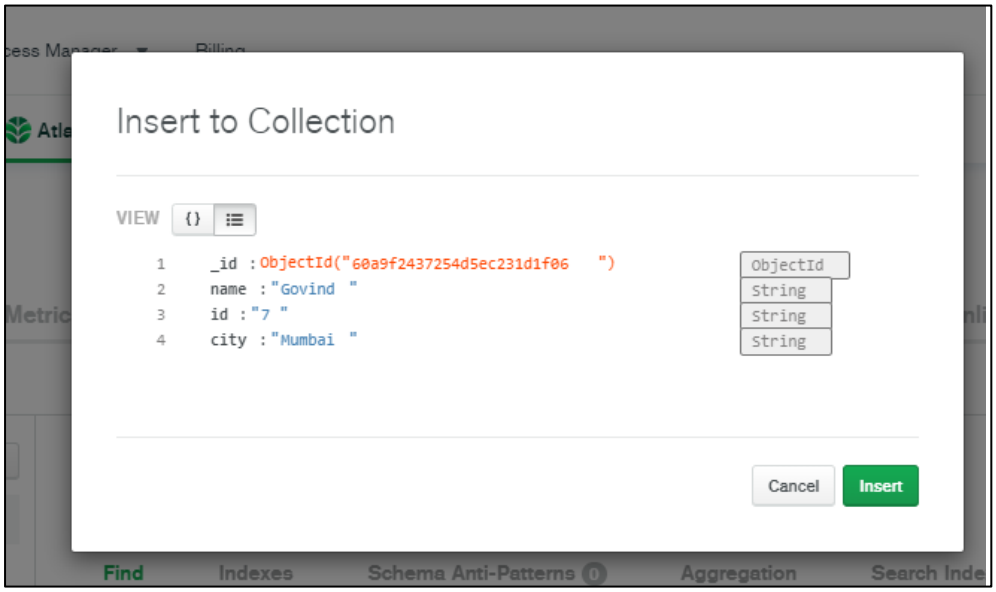


Step 3: Click on 'Add My Own Data' to create a database.



Step 4: Click on insert document to add records.

Since MongoDB is a No-SQL database, so you can add 'n' number of columns for any row/record.



Perform updating data

Performing deleting data.

```
_id: ObjectId("60a9f2437254d5ec231d1f06")
name: "Govind Saini"
id: "7"
city: "Mumbai"

_id: ObjectId("60a9f4917254d5ec231d1f07")
name: "Sayali Mam"
id: "8"
city: "Mumbai"
```

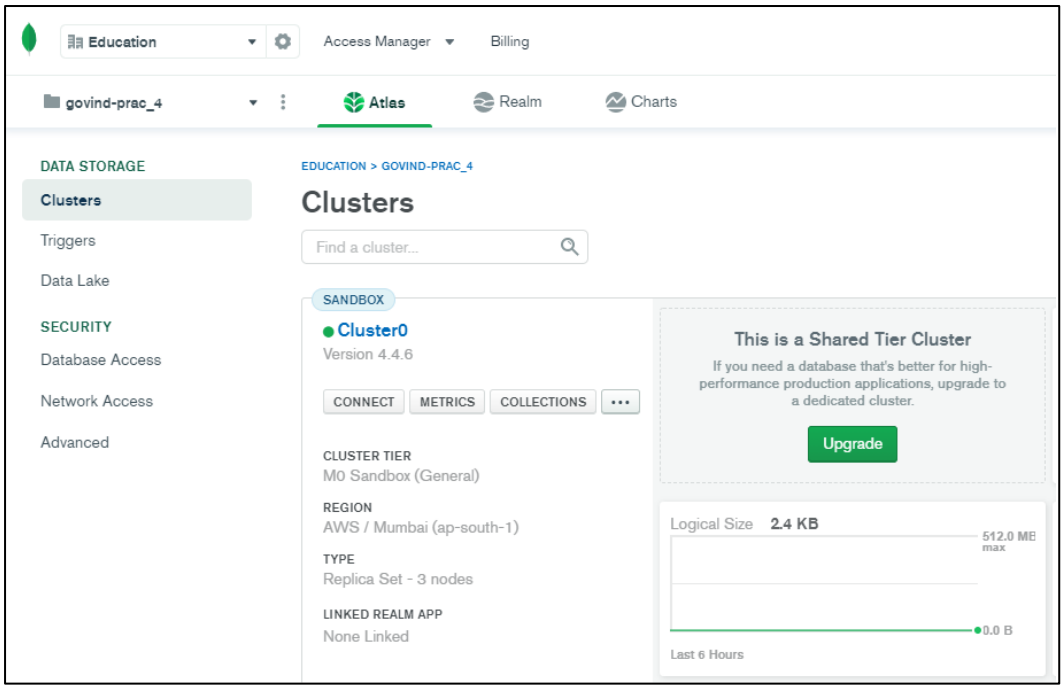
Deleting Document.

Performing Insert data

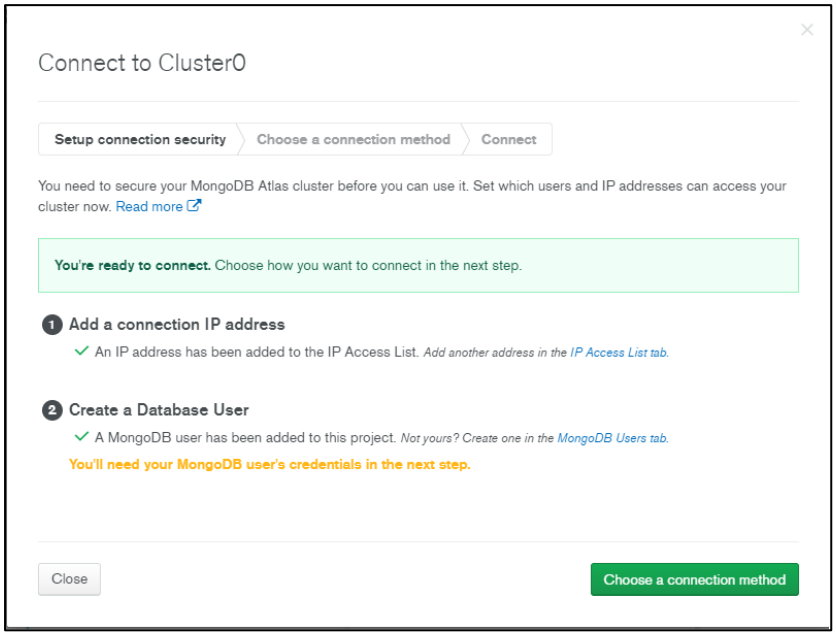
QUERY RESULTS 1-2 OF 2

```
_id: ObjectId("60a9ff027254d5ec231d1f0b")
name: "Govind Saini"
id: " 7"
city: "Mumbai"

_id: ObjectId("60a9ff3a7254d5ec231d1f0c")
name: "Sohrab Sir"
id: " 5"
city: "Mumbai"
```

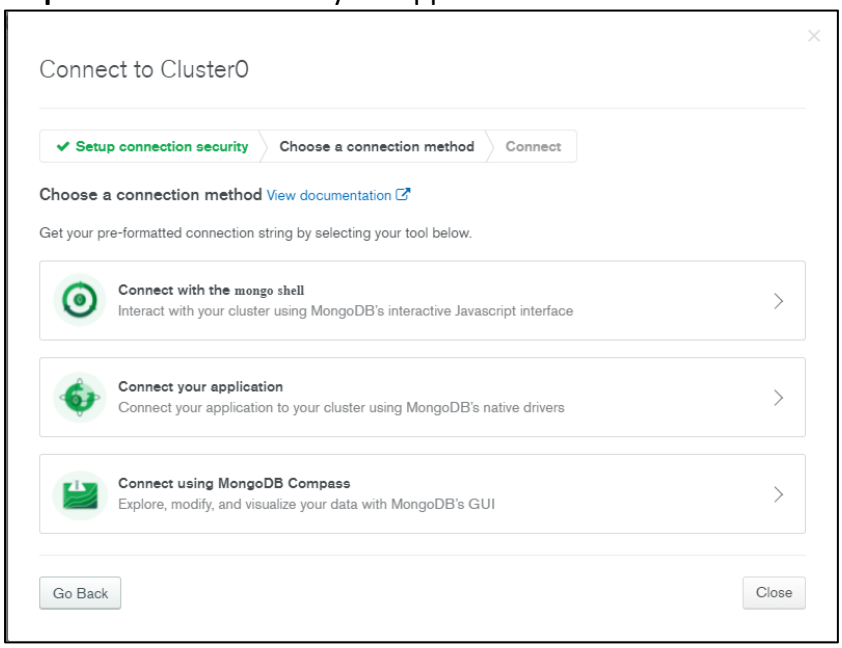
Step 5: To start with the connection click on Overview, and then click on Connect.



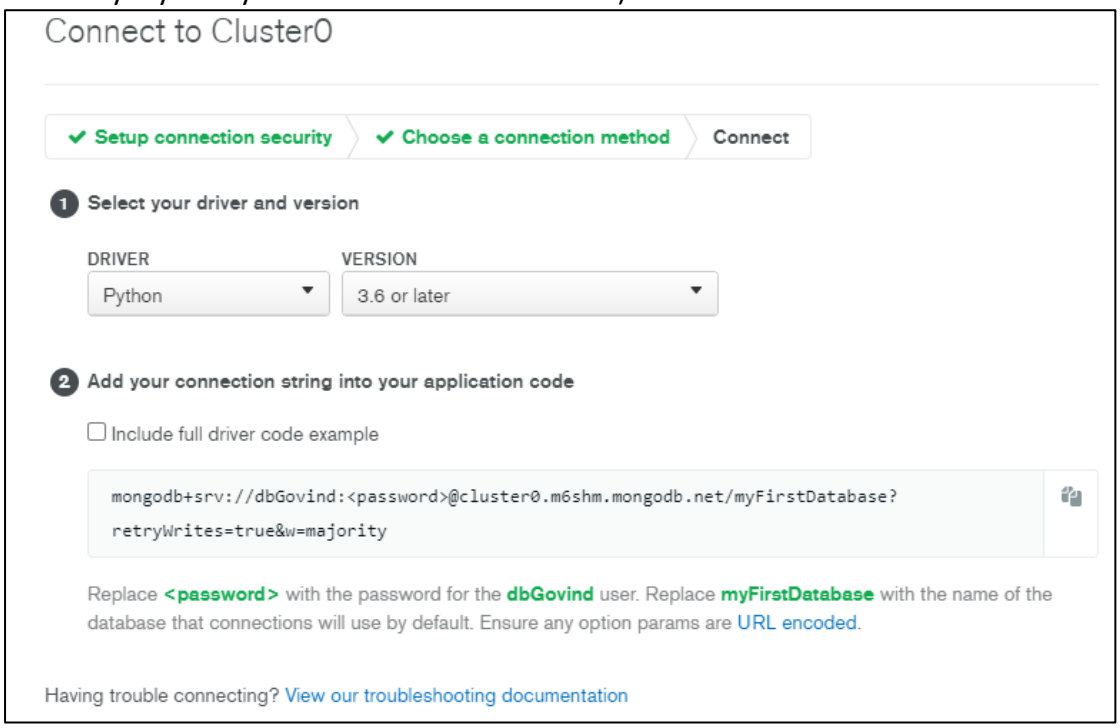
Step 6: Select on add your current IP and create a MongoDB user.



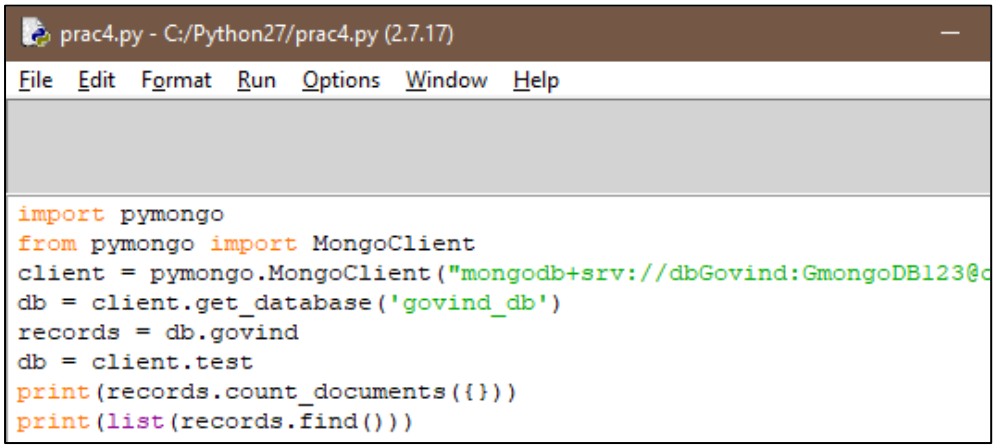
Step 7: Click on 'Connect your application'.



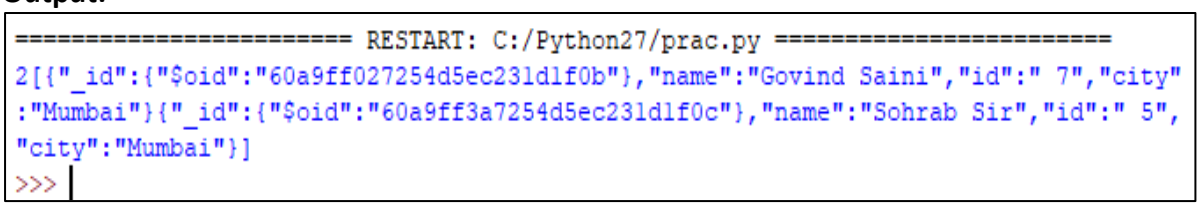
Step 8: Select the driver as 'Python' and version as '3.6 or later'. (Select the version as 3.6 or later only if your Python's version is 3.6 or later.)



Step 9: Write the code given below in a Python file.



Output:



PRACTICAL NO: 5

Aim: Configure the Hive and implement the application in Hive.

Configuring the Hive:

Step 1: Pre-requisites:

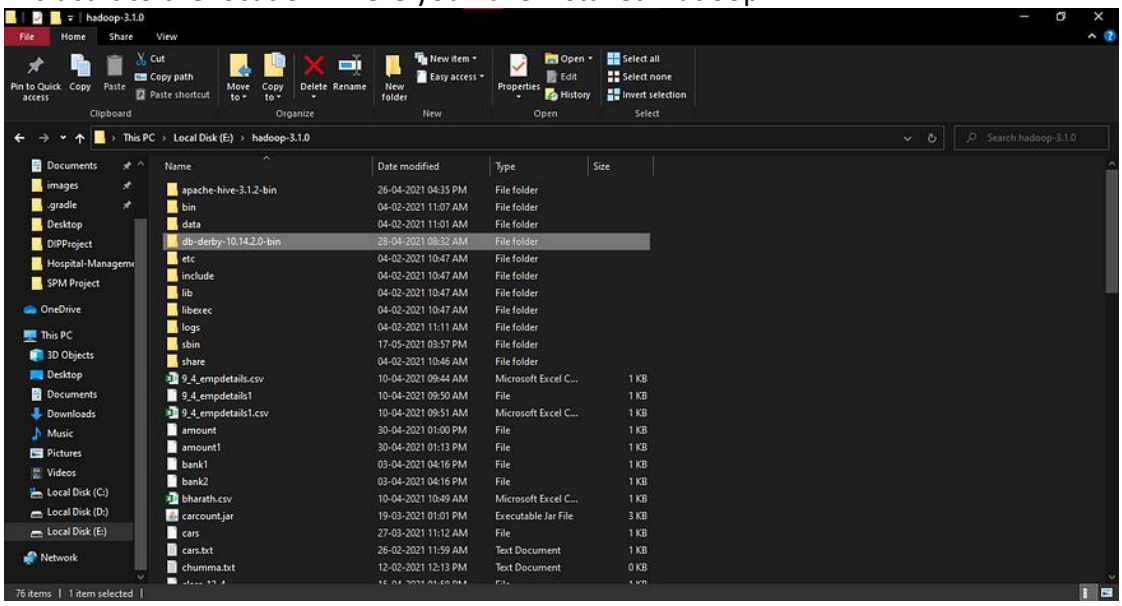
Download Apache Derby Binaries:

Hive requires a relational database like Apache Derby to create a Metastore and store all metadata

Download the derby tar file from the following link:

https://downloads.apache.org//db/derby/db-derby-10.14.2.0/db-derby-10.14.2.0-bin.tar.gz

Extract it to the location where you have installed Hadoop

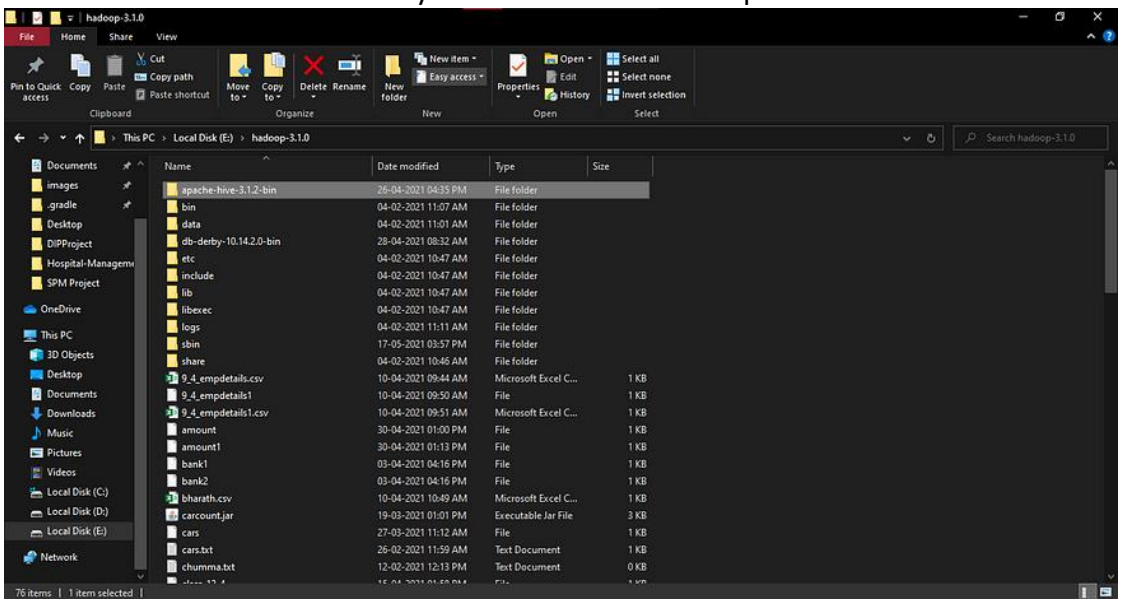


Step 2: Download Hive binaries:

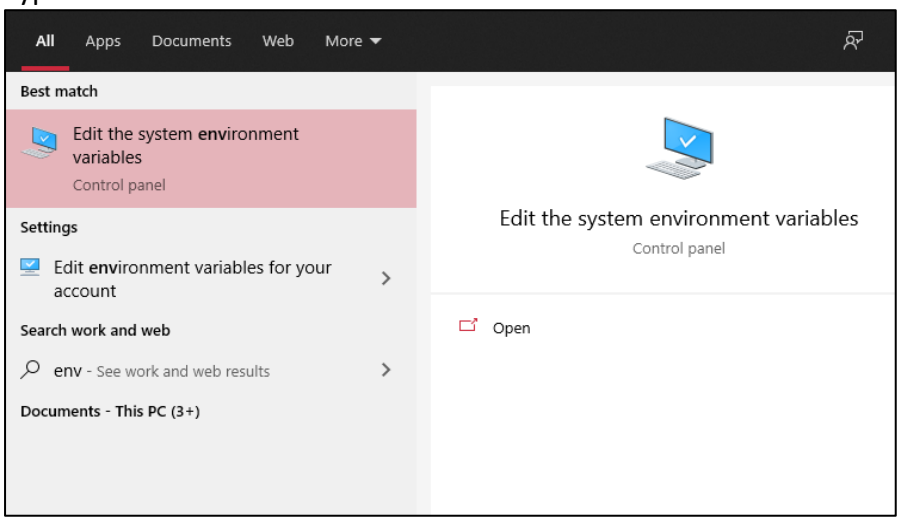
Download Hive binaries from the following link:

https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz

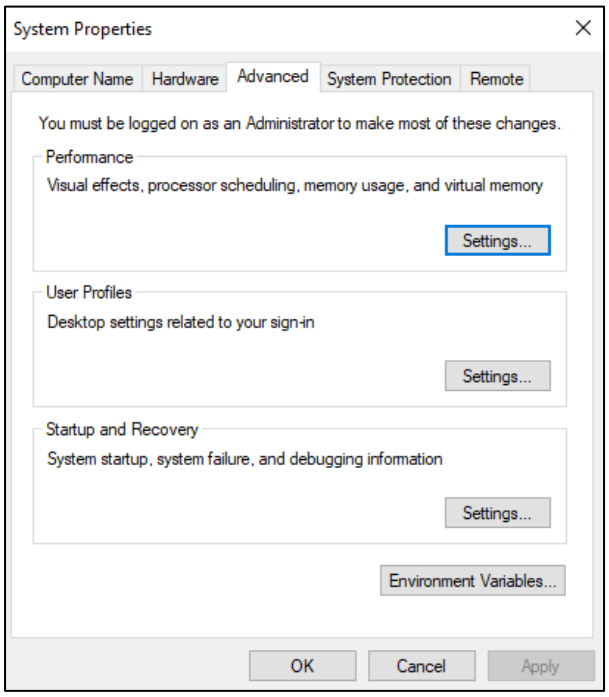
Extract it to the location where you have installed Hadoop



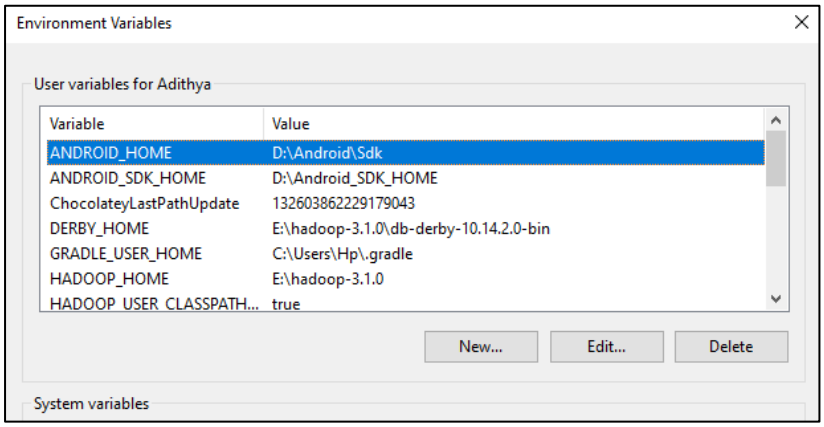
Step 3: Setting up Environment variables:
Type ‘environment’ in Windows Search Bar



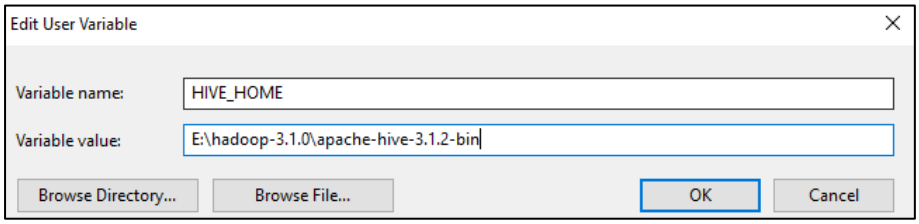
Click on Environment Variables



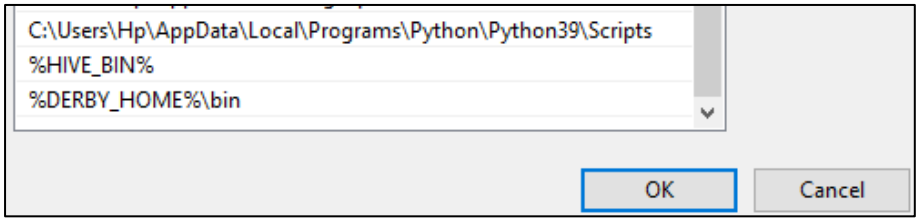
Click on New



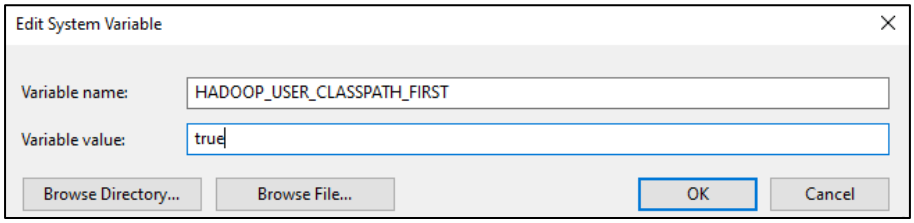
Add the following variables:
HIVE_HOME: E:\hadoop-3.1.0\apache-hive-3.1.2-bin
DERBY_HOME: E:\hadoop-3.1.0\db-derby-10.14.2.0-bin
HIVE_LIB: E:\hadoop-3.1.0\apache-hive-3.1.2-bin\lib
HIVE_BIN: E:\hadoop-3.1.0\apache-hive-3.1.2-bin\bin
HADOOP_USER_CLASSPATH_FIRST: true



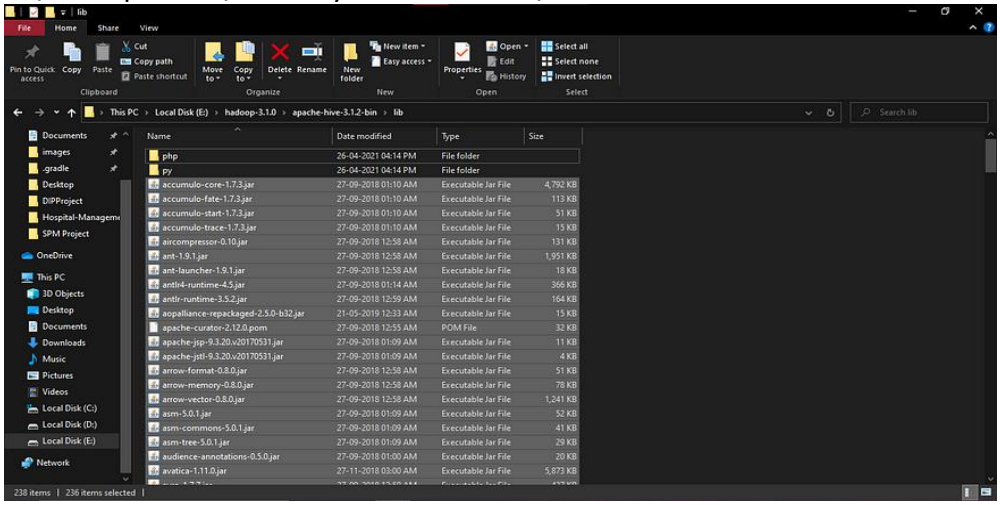
In Path Variable in User Variables add the following paths:
%HIVE_BIN%
%DERBY_HOME%\bin



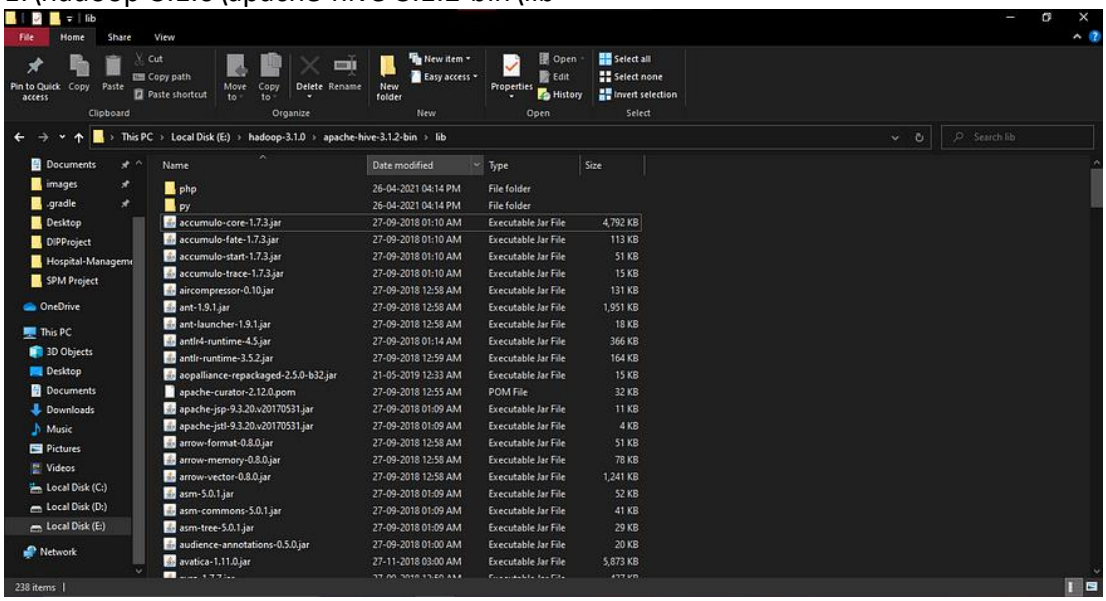
Now in System Variables add the following:
HADOOP_USER_CLASSPATH_FIRST: true



Step 4: Configuring Hive
Copy Derby Libraries:
Copy all the jar files stored in Derby library files stored in:
E:\hadoop-3.1.0\db-derby-10.14.2.0-bin\lib



And paste them in Hive libraries directory:
E:\hadoop-3.1.0\apache-hive-3.1.2-bin\lib



Step 5: Configuring Hive-site.xml
Create a new file with the name hive-site.xml in E:\hadoop-3.1.0\apache-hive-3.1.2-bin\conf
Add the following lines in the file

```
<?xml version=”1.0"?>
<?xml-stylesheet type=”text/xsl” href=”configuration.xsl”?>
<configuration><property> <name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:derby://localhost:1527/metastore_db;create=true</value>
<description>JDBC connect string for a JDBC metastore</description>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>org.apache.derby.jdbc.ClientDriver</value>
<description>Driver class name for a JDBC metastore</description></property>
<property><name>hive.server2.enable.doAs</name>
<description>Enable user impersonation for HiveServer2</description>
<value>true</value></property>
<property><name>hive.server2.authentication</name><value>NONE</value>
<description> Client authentication types. NONE: no authentication check LDAP: LDAP/AD
based authentication KERBEROS: Kerberos/GSSAPI authentication CUSTOM: Custom
authentication provider (Use with property hive.server2.custom.authentication.class)
</description></property>
<property>
<name>datanucleus.autoCreateTables</name>
<value>True</value></property>
<property>
<name>hive.server2.active.passive.ha.enable</name>
<value>true</value> # change false to true
</property>
</configuration>
```

Step 6: Starting Services
Start Hadoop Services:
Change the directory in terminal to the location where Hadoop is stored and give the following command:
start-all.cmd
Start Derby Network Server:
Start the Derby Network Server with the following command:
StartNetworkServer -h 0.0.0.0

Initialize Hive Metastore:
Give the following command to initialize Hive Metastore:
hive --service schematool -dbType derby -initSchema
Start Hive Server:
hive --service hiveserver2 start
Start Hive:
Start hive by giving the following command:
hive

Implement the application in Hive.
Step 1: Creating a Hive Database
Start the Hive CLI or Beeline (depending on your setup). You can use the Hive CLI by running hive in your terminal.
hive

Once inside the Hive CLI, create a database:
CREATE DATABASE my_database;
USE my_database;

Step 2: Creating a Table
Let's create a simple table to store user information. For this example, we'll create a table named users.

```
CREATE TABLE users (
user_id INT,
user_name STRING,
user_email STRING
)
ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Step 3: Loading Data into the Table.
Assume you have a file named users.csv with the following content:

```
1,John Doe,john.doe@example.com
2,Jane Smith,jane.smith@example.com
3,Jim Brown,jim.brown@example.com
```

You can load this data into the Hive table using the following command:
LOAD DATA LOCAL INPATH '/path/to/users.csv' INTO TABLE users;

Step 4: Querying the Data
SELECT * FROM users;

Application Code (Java)
Add Hive JDBC dependency to your pom.xml if you're using Maven:

```
<dependency>
   <groupId>org.apache.hive</groupId>
   <artifactId>hive-jdbc</artifactId>
   <version>3.1.2</version>
</dependency>
```

Java code to interact with Hive:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class HiveExample {
   public static void main(String[] args) {
      // JDBC driver and Hive connection URL
      String jdbcDriver = "org.apache.hive.jdbc.HiveDriver";
      String jdbcUrl = "jdbc:hive2://localhost:10000/my_database";
      try {
         // Register JDBC driver
         Class.forName(jdbcDriver);
         // Open a connection
         Connection conn = DriverManager.getConnection(jdbcUrl, "", "");
         Statement stmt = conn.createStatement();
         // Execute a query
         String sql = "SELECT * FROM users";
         ResultSet rs = stmt.executeQuery(sql);
         // Process the result set
         while (rs.next()) {
            int userId = rs.getInt("user_id");
            String userName = rs.getString("user_name");
            String userEmail = rs.getString("user_email");
            System.out.println("ID: " + userId + ", Name: " + userName + ", Email: " +
userEmail);
         }
         // Clean up
         rs.close();
         stmt.close();
         conn.close();
      } catch (Exception e) {
         e.printStackTrace();
      } } }
```

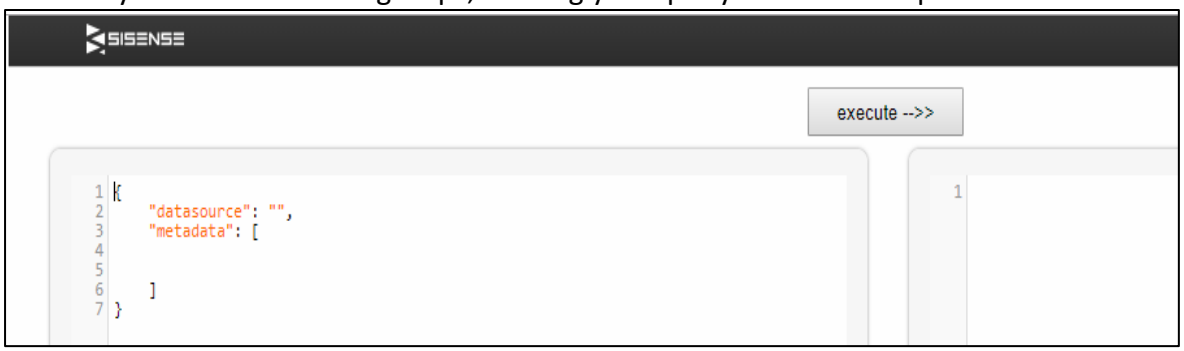
PRACTICAL NO: 6

Aim: Write a program to illustrate the working of JAQL.

Building your First JAQL Query.

Step 1: Using the JAQL Runner Utility

Sisense comes bundled with a utility that lets you write and run JAQL queries. You will use this utility for all the following steps, running your query after each step to test it.



Step 2: Retrieving a Dimension.

```
{
  "datasource": "Training",
  "metadata": [ {
        "dim": "[Customers.Country]"
    } ] }
```

Step 3: Adding a Simple Aggregation

Add the following object to your query's metadata array:

```
{
  "dim": "[Customers.CustomerID]",
  "agg": "count"
}
```

Execute the query, and you should see a result similar to this:


```
{
    "headers": [
        "Country",
        "count Customers.CustomerID"
    ],
    "metadata": [
        {
            "dim": "[Customers.Country]"
        },
        {
            "dim": "[Customers.CustomerID]",
            "agg": "count"
        }
    ],
    "datasource": {
        "fullname": "LocalHost/Training",
        "revisionId": "f07d89ab-1313-4fff-8f77-52b921f2de76"
    },
    "values": [
        [
            {
                "data": "Argentina",
                "text": "Argentina"
            },
            {
                "data": 3,
                "text": "3"
            }
        ],
        [
            {
                "data": "Austria",
                "text": "Austria"
            },
            {
                "data": 2,
                "text": "2"
            }
        ],
```

Step 4: Adding a Filter
Add the following property to your "country" metadata object:

```
"filter": {
   "members": [
      "Argentina", "Canada", "USA",
      "Mexico", "Venezuela", "Brazil" ] }
```

Step 5: Adding a Background Filter

```
{
  "dim": "[Products.ProductName]",
  "filter": {
     "members": ["Tofu"]
  }
}
```

Step 6: Adding a Measure Filter
Add the following filter object to your measure:

```
"filter": {
  ">": 1
}
```

Step 7: Adding a Formula
Add the following object to your query's metadata collection:

```
{
 "formula": "AVG([OrderDateYears], [CountOrderID])",
 "context": {
    "[OrderDateYears]": {
       "dim": "[Orders.OrderDate (Calendar)]",
       "level": "years"
    },
    "[CountOrderID]": {
        "dim": "[Orders.OrderID]",
        "agg": "count"
     } } }
```

Step 8: Using Additional Properties
At this stage, your query should look like this:

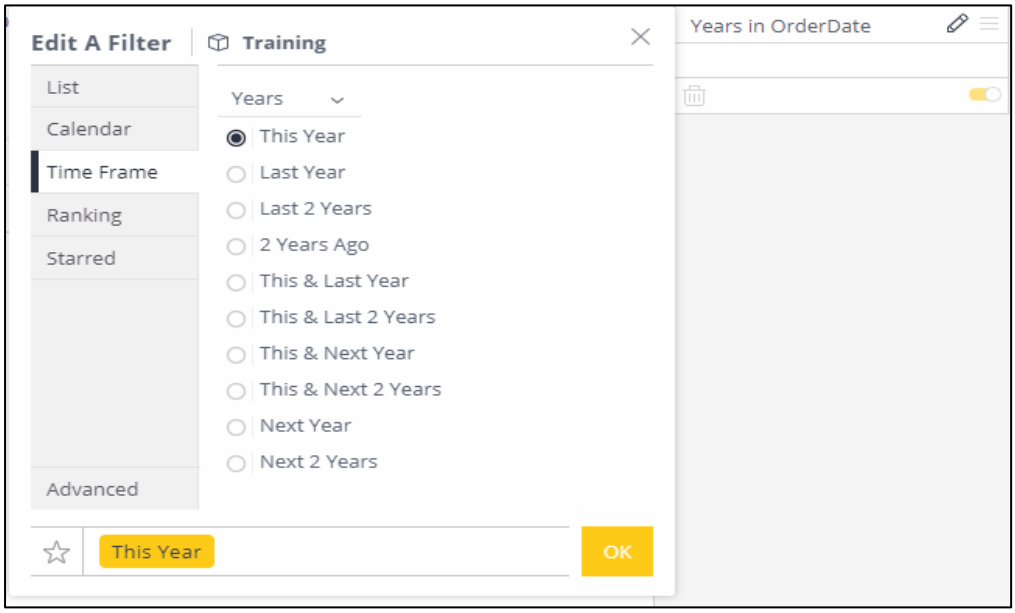
```
{
  "datasource": "Training",
  "metadata": [
     {
        "dim": "[Customers.Country]",
        "filter": {
           "members": [
              "Argentina", "Canada", "USA",
              "Mexico", "Venezuela", "Brazil"
```

```
        ] }
    },{
      "dim": "[Customers.CustomerID]",
      "agg": "count",
      "filter": {
        ">":1
      }
    },
    {
      "dim": "[Products.ProductName]",
      "filter": {
        "members": ["Tofu"]
      },
      "panel": "scope"
    }, {
      "formula": "AVG([OrderDateYears], [CountOrderID])",
      "context": {
        "[OrderDateYears]": {
          "dim": "[Orders.OrderDate (Calendar)]",
          "level": "years"
        },
        "[CountOrderID]": {
          "dim": "[Orders.OrderID]",
          "agg": "count"
        }
      } }
] }
```

Using JAQL for Custom Filters:

Step 1: Viewing the JAQL of a Filter

Click the Edit button on the OrderDate dashboard filter:



Step 2: Constructing a Custom Filter:
To construct a custom filter, you only need to modify the JAQL extracted from the time frame filter - by changing the "count" property to 8, you will retrieve data from the past 8 years.
However, in some cases you will need to perform a more elaborate modification of the JAQL. For this purpose, the JAQL Reference will be helpful. You can use some capabilities that aren't present in the UI, such as composite filters using the "and" and "or" keywords.

Step 3: Applying Custom Filters:
Use the "Test" button to execute a simple 1-dimensional query using the filter in the "Advanced" tab's left textbox.
Note that since the filter is a custom one, it has no UI to represent it, and will appear on your filter pane as an empty panel, like so:

Using JAQL Queries in Scripts:
In this part of the tutorial, you will learn how to extract the JAQL queries behind various widgets on your dashboard, and a simple method of running JAQL queries from a script and parsing the result.

Step 1: Extracting JAQL from a widget.
Open the attached dashboard called "JAQL-Training-2". You will find an Indicator from which you will extract the underlying JAQL query. Follow these steps:

- Edit the widget.
- Open your browser's developer console (usually by pressing F12).
- Type in the following code in the console: prism.debugging.GetJaql(prism.activeWidget).

Step 2: Setting Up and Authentication:
Note: This example is written in Node.js. For other languages such as Python, implementation will vary slightly.

```
// Import Modules
const rp = require('request-promise');
const querystring = require('querystring');
const authenticate = (username, password) => {
  const data = querystring.stringify({
    username,
    password
  });
  const options = {
    url: "https://example.com/api/v1/authentication/login",
    method: "POST",
```

```
    headers: {
      "content-type":"application/x-www-form-urlencoded",
      "Content-Length": Buffer.byteLength(data)
    },
    body: data
  };
  return rp(options).then((res) =>{
    const response = JSON.parse(res);
    token = response.access_token;
    return token;
  }).catch((err) => {
    console.error("An error has occurred attempting API call to the authentication/login
endpoint.");
    throw err;
  });
}
```

Step 3: Executing the Query

Now that you have extracted a JAQL from an existing widget, you can simply execute it by sending it as the payload. Don't forget to include the API token retrieved in the previous step. The Node.js code below is a simple example of a JAQL request to a Sisense for Windows server:

```
const runJaql = (jaql, cube, token) => {
  const options = {
    url: "https://example.com/api/elasticubes/"+cube+"/jaql",
    method: 'POST',
    headers: {
      "Content-Type": "application/json",
      "Authorization": 'Bearer ' + token
    },
    body: JSON.stringify(jaql);
  };
  }).catch((err) => {
    console.error("An error has occurred attempting API call to JAQL endpoint.");
    throw err;
  });
}
```

Step 4: Parsing the Result.

Running the runJaql function from the previous step will return a JavaScript Promise which, if the HTTP call is successful, resolves to a response JSON object which should look like this:

```
{
  "headers": [
    "Average Orders Per Customer"
  ],
  "datasource": {
    "fullname": "LocalHost/Training",
    "revisionId": "f07d89ab-1313-4fff-8f77-52b921f2de76"
```

```
  },
  "metadata": [
    {
      "jaql": {
        "type": "measure",
        "formula": "AVG([99CFE-E7A], [AD2EA-8D0])",
        "context": {
          "[AD2EA-8D0]": {
            "table": "Orders",
            "column": "OrderID",
            "dim": "[Orders.OrderID]",
            "datatype": "numeric",
            "merged": true,
            "agg": "count",
            "title": "# of unique OrderID"
          },
          "[99CFE-E7A]": {
            "table": "Customers",
            "column": "CustomerID",
            "dim": "[Customers.CustomerID]",
            "datatype": "text",
            "merged": true,
            "title": "CustomerID"
          }
        },
        "title": "Average Orders Per Customer"
      },
      "format": {
        "mask": {
          "type": "number",
          "abbreviations": { t": true,  "b": true,  m": true,  k": false
          },
          "separated": true,  "decimals": "auto", "isdefault": true
        },
        "color": {"color": "#00cee6", "type": "color"  }
      },
      "source": "value",
      "handlers": [
        {},
        {}
      ] } ],
  "values": [
    {
      "data": 4.744186046511628,
      "text": "4.74418604651163"
    } ]
}
```

PRACTICAL NO: 7

Aim: Implement Decision tree classification techniques.
Description: Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**

Step 1: The package "party" has the function ctree() which is used to create and analyse decision tree.

```
> install.packages("party")
```

Step 2: Load the party package. It will automatically load other# dependent packages Print some records from data set reading Skills.

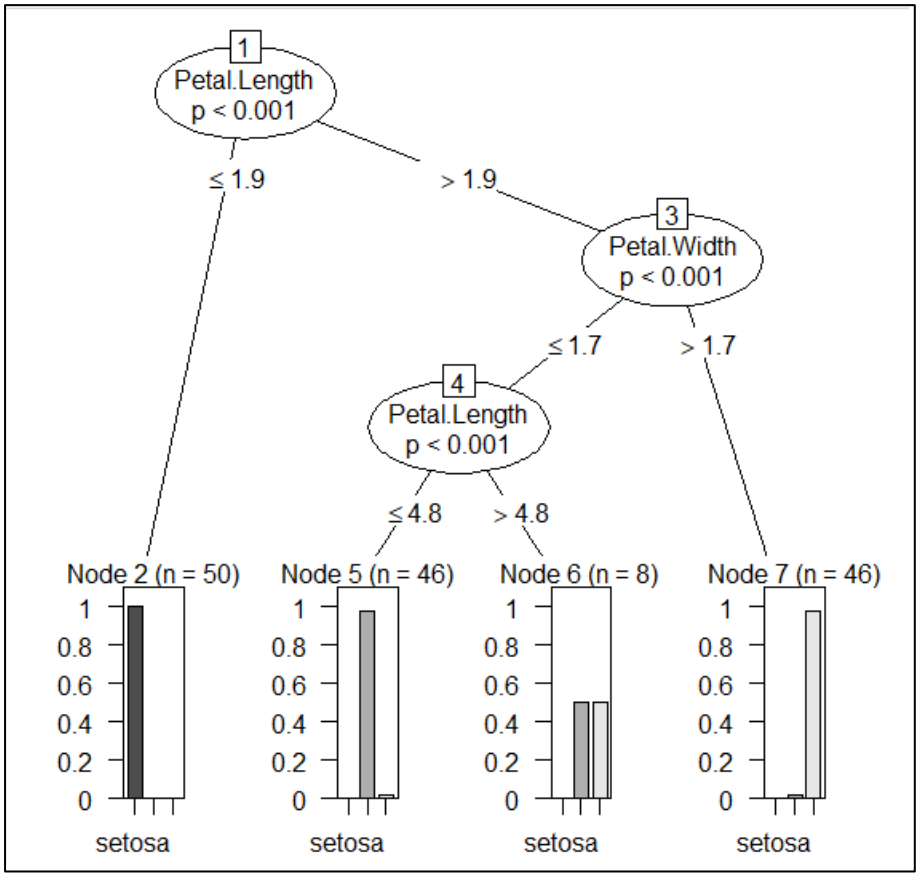
```
> library("party")
> print(head(readingSkills))
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3            no  11 30.42170 49.60593
4           yes   7 28.66450 40.28456
5           yes  11 31.88207 55.46085
6           yes  10 30.07843 52.83124
>
```

Step 3: Call function ctree to build a decision tree. The first parameter is a formula, which defines a target variable and a list of independent variables.

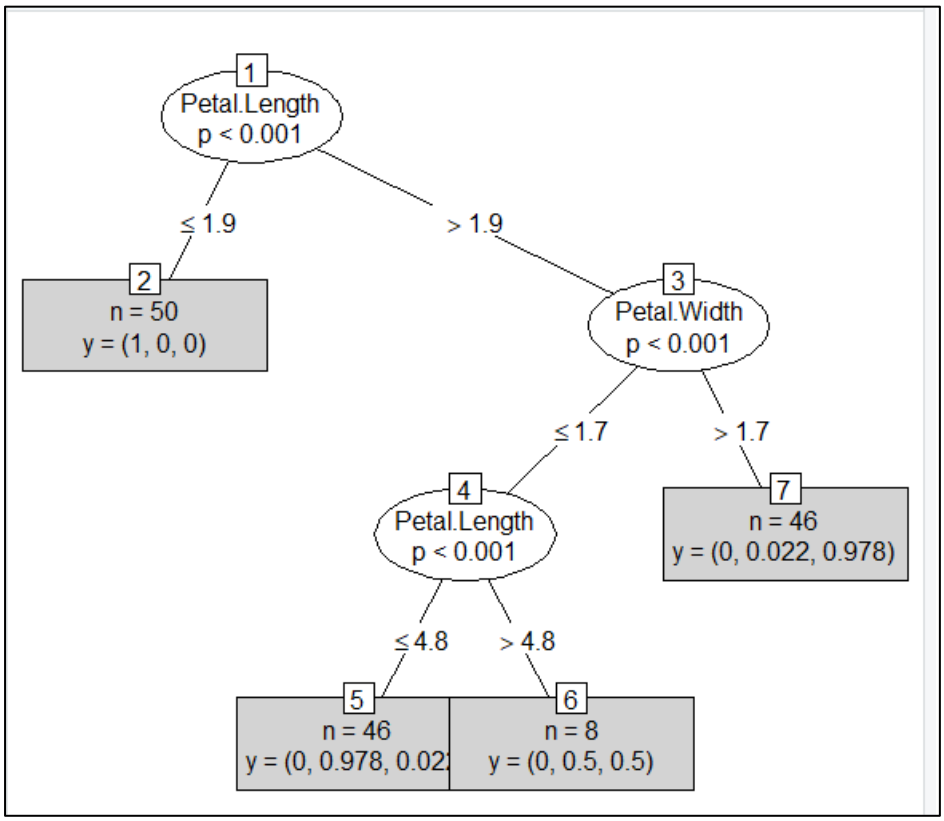
```
> library("party")
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1
```

```
> iris_ctree <- ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Peta
l.Width, data=iris)
> print(iris_ctree)

         Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  150

1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264
  2)*  weights = 50
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 67.894
    4) Petal.Length <= 4.8; criterion = 0.999, statistic = 13.865
      5)*  weights = 46
    4) Petal.Length > 4.8
      6)*  weights = 8
  3) Petal.Width > 1.7
    7)*  weights = 46
> plot(iris_ctree)
```

Output:



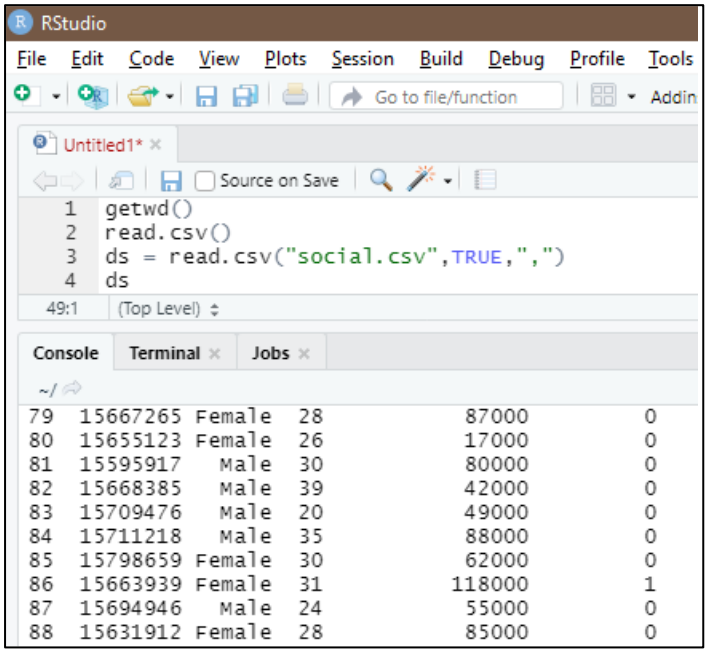

```
> plot(iris_ctree, type="simple")
```



PRACTICAL NO: 8

Aim: Implement SVM classification techniques.
Description: A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labelled training data for each category, they're able to categorize new text

The implementation is explained in the following steps:
Step 1: Importing the dataset



Step 2: Selecting columns 3-5.

```
> ds = ds[3:5]
> ds[3:5]
Error in `[.data.frame`(ds, 3:5) : undefined
> ds
    Age EstimatedSalary Purchased
1    19           19000         0
2    35           20000         0
3    26           43000         0
4    27           57000         0
5    19           76000         0
6    27           58000         0
7    27           84000         0
8    32          150000         1
9    25           33000         0
10   35           65000         0
11   26           80000         0
12   26           52000         0
```

Step 3: install package

```
> install.packages("caTools")
```

Step 4: Splitting the dataset.

```
> library(caTools)
> set.seed(123)
> split = sample.split(ds$Purchased, SplitRatio = 0.75)
> training_set = subset(ds, split == TRUE)
> test_set = subset(ds, split == FALSE)
> ds
    Age EstimatedSalary Purchased
1    19           19000         0
2    35           20000         0
3    26           43000         0
4    27           57000         0
5    19           76000         0
6    27           58000         0
7    27           84000         0
8    32          150000         1
9    25           33000         0
10   35           65000         0
```

Step 5: Feature Scaling.

```
332  48          119000         1
333  42           65000         0
 [ reached 'max' / getOption("max.print") -- omitted 67 rows ]
> test_set[-3] = scale(test_set[-3])
> training_set[-3] = scale(training_set[-3])
> test_set[-3] = scale(test_set[-3])
> test_set[-3]
           Age EstimatedSalary
2   -0.30419063     -1.51354339
4   -1.05994374     -0.32456026
5   -1.81569686      0.28599864
9   -1.24888202     -1.09579256
12  -1.15441288     -0.48523366
18   0.64050076     -1.32073531
19   0.73496990     -1.25646596
20   0.92390818     -1.22433128
22   0.82943904     -0.58163769
29  -0.87100546     -0.77444577
32  -1.05994374      2.24621408
34  -0.96547460     -0.74231109
35  -1.05994374      0.73588415
38  -0.77653633     -0.58163769
45  -0.96547460      0.54307608
46  -1.43782030     -1.51354339
```

Step 6: Fitting SVM to the training set.

```
> install.packages('e1071')
```

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

Call:
svm(formula = Purchased ~ ., data = training_set, type = "C-classification",
    kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  116
```

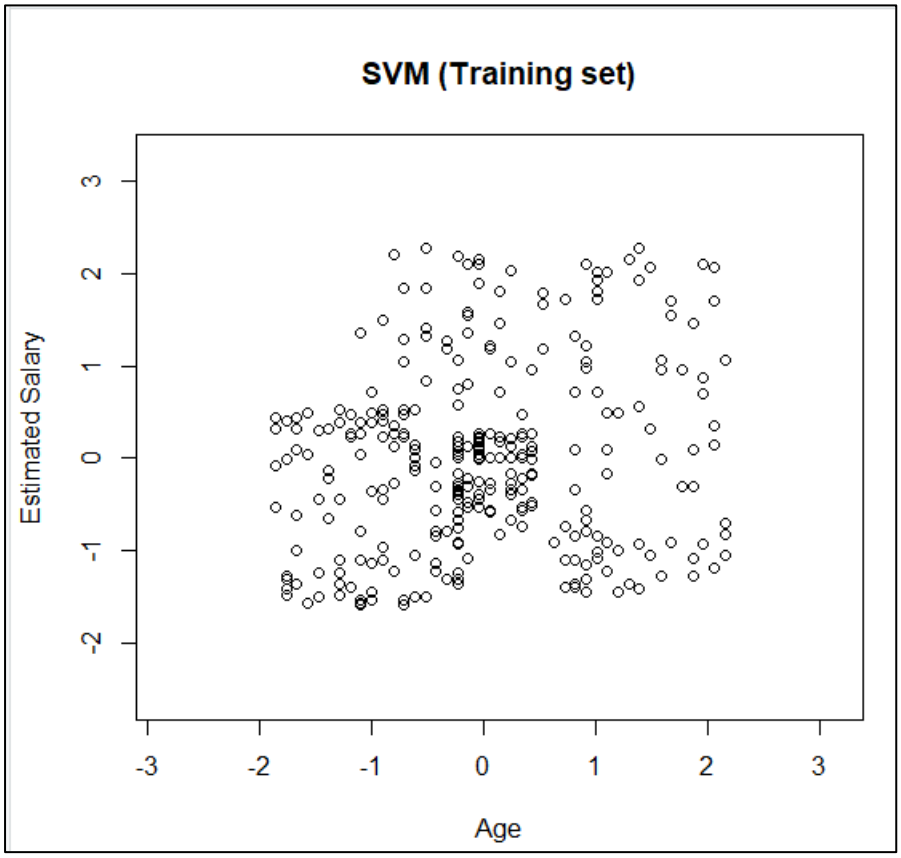
Step 7: Predicting the test set result.

```
> y_pred = predict(classifier, newdata = test_set[-3])
> y_pred
  2   4   5   9  12  18  19  20  22  29  32  34  35  38  45  46  48  52  66
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 69  74  75  82  84  85  86  87  89 103 104 107 108 109 117 124 126 127 131
  0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
134 139 148 154 156 159 162 163 170 175 176 193 199 200 208 213 224 226 228
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   0   1
229 230 234 236 237 239 241 255 264 265 266 273 274 281 286 292 299 302 305
  0   1   1   1   0   1   1   1   0   1   1   1   1   1   0   1   1   1   0
307 310 316 324 326 332 339 341 343 347 353 363 364 367 368 369 372 373 380
  1   0   0   0   0   1   0   1   0   1   1   0   1   1   1   0   1   0   1
383 389 392 395 400
  1   0   0   0   0
Levels: 0 1
```

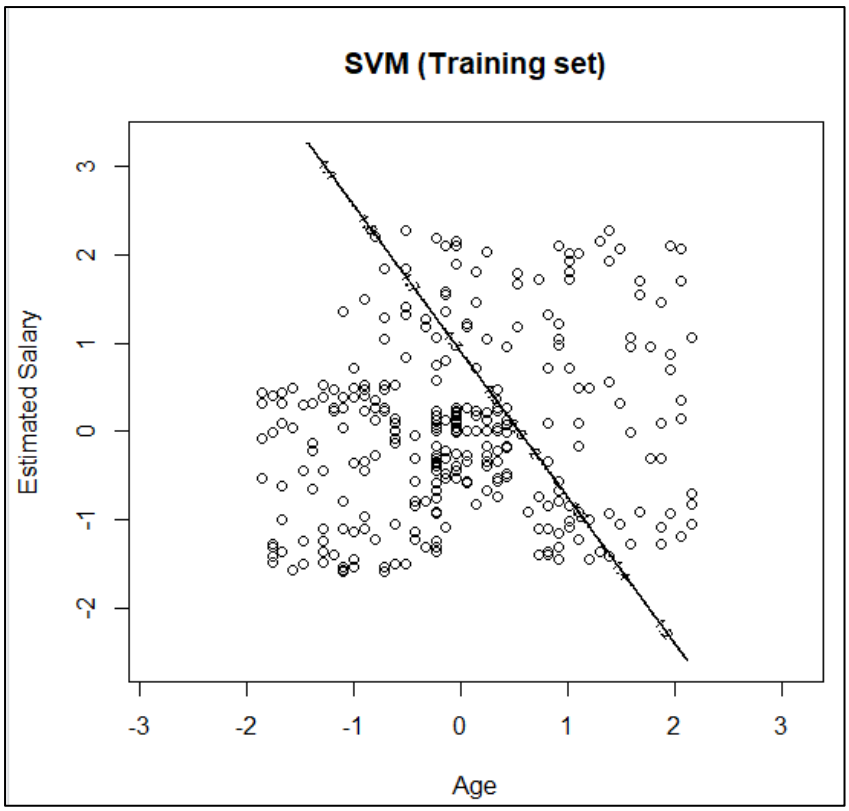
```
> cm = table(test_set[, 3], y_pred)
> cm
   y_pred
     0  1
  0 57  7
  1 13 23
```

Step 8: Visualizing the Training set results.

```
> set = training_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
```

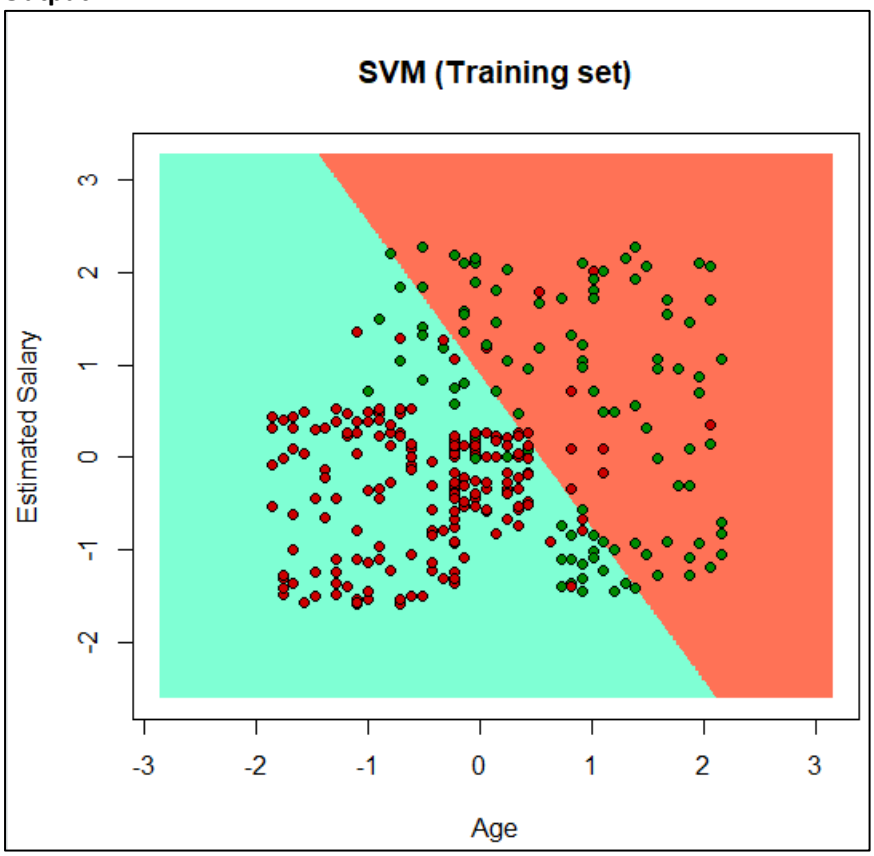



```
> grid_set = expand.grid(X1, X2)
> colnames(grid_set) = c('Age', 'EstimatedSalary')
> y_grid = predict(classifier, newdata = grid_set)
> plot(set[, -3],
+      main = 'SVM (Training set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
```




```
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Output:



PRACTICAL NO: 9

Aim: Write a Program showing implementation of Regression model.

Description: Regression is a method to mathematically formulate relationship between variables that in due course can be used to estimate, interpolate and extrapolate. Suppose we want to estimate the weight of individuals, which is influenced by height, diet, workout, etc. Here, *Weight* is the **predicted** variable.

Let's implementation of Regression Model some Example:

```
> IQ <- rnorm(40, 30, 2)
```

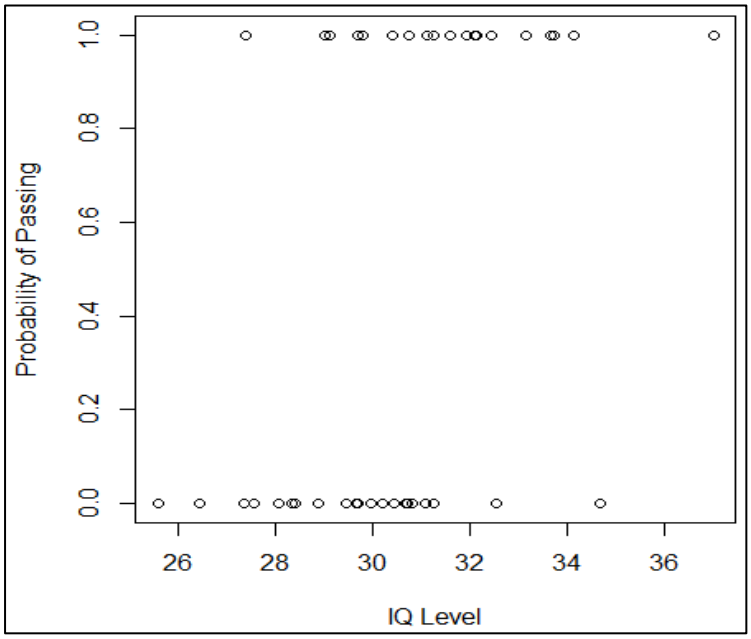
```
> IQ <- sort(IQ)
```

```
> result <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
+ 1, 0, 0, 0, 1, 1, 0, 0, 1, 0,
+ 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
+ 1, 1, 1, 0, 1, 1, 1, 1, 0, 1)
```

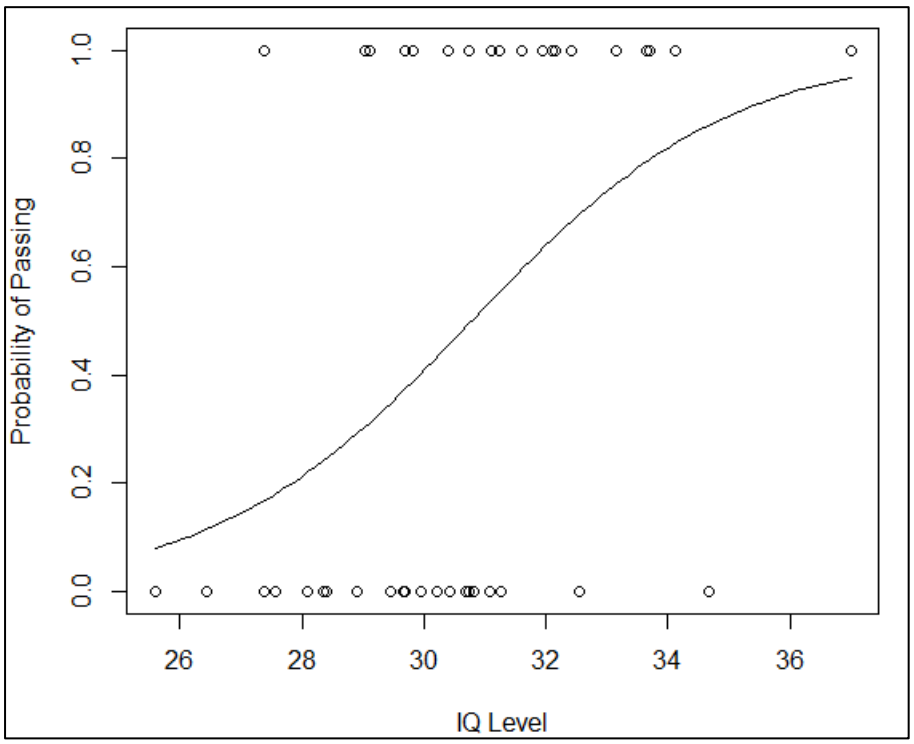
```
> df <- as.data.frame(cbind(IQ, result))
> print(df)
          IQ result
1   25.58824      0
2   26.43200      0
3   27.37083      0
4   27.37898      1
5   27.56671      0
6   28.08275      0
7   28.35637      0
8   28.41538      0
9   28.89752      0
10  29.03158      1
11  29.12386      1
12  29.46181      0
13  29.66945      0
14  29.68934      0
15  29.69886      1
16  29.80735      1
17  29.95326      0
18  30.21428      0
19  30.39298      1
20  30.43421      0
21  30.67802      0
22  30.72653      0
23  30.74974      1
24  30.82265      0
25  31.07116      0
26  31.11633      1
27  31.24740      1
28  31.25662      0
29  31.60194      1
30  31.93038      1
```

```
> png(file="LogisticRegressionGFG.png")
```

```
> plot(IQ, result, xlab = "IQ Level",
+ ylab = "Probability of Passing")
> g = glm(result~IQ, family=binomial, df)
```



```
> curve(predict(g, data.frame(IQ=x), type="resp"), add=TRUE)
> points(IQ, fitted(g), pch=30)
```




```
> summary(g)

Call:
glm(formula = result ~ IQ, family = binomial, data = df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9877  -0.9804  -0.4502   0.9731   1.8898

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -14.4934     5.8835  -2.463   0.0138 *
IQ            0.4708     0.1922   2.450   0.0143 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 55.352  on 39  degrees of freedom
Residual deviance: 47.090  on 38  degrees of freedom
AIC: 51.09

Number of Fisher Scoring iterations: 4
```

```
> dev.off()
null device
          1
```

PRACTICAL NO: 10

Aim: Write a Program showing Clustering.

Description:

In this Program we understand about K-Mean Clustering

What Does K-Means Clustering Mean?

- K-means clustering is a simple unsupervised learning algorithm that is used to solve clustering problems.
- It follows a simple procedure of classifying a given data set into a number of clusters, defined by the letter "k," which is fixed beforehand.
- The clusters are then positioned as points and all observations or data points are associated with the nearest cluster, computed, adjusted and then the process starts over using the new adjustments until a desired result is reached.

We Understand in different Steps:

Step 1: Apply k-means to *new iris*, and store the clustering result in *kc*. The cluster number is set to 3.

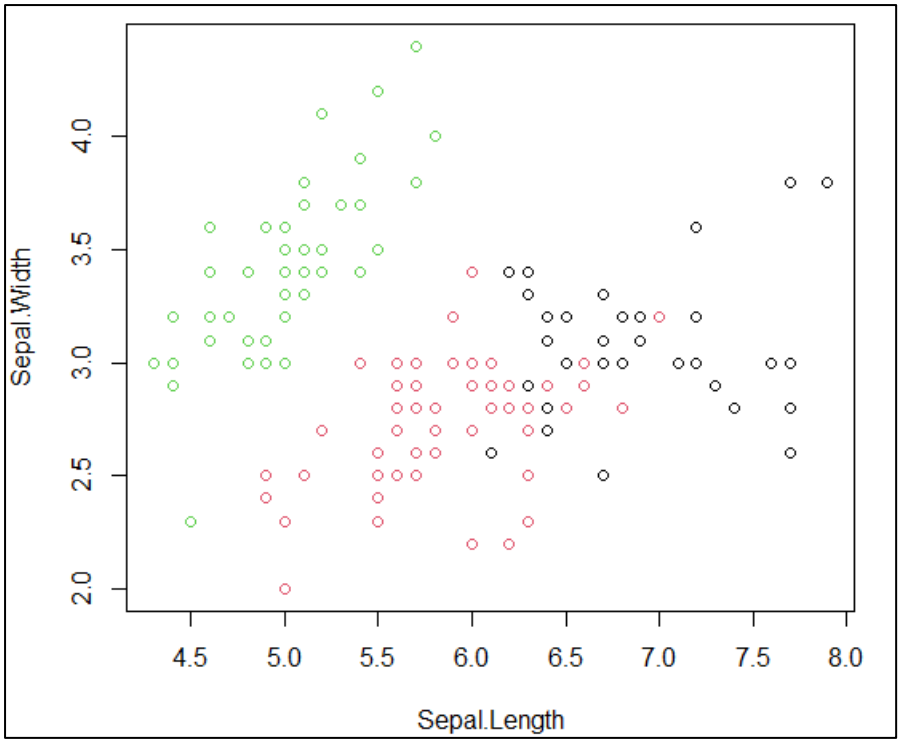
```
> newiris <- iris
> newiris$Species <- NULL
> (kc <- kmeans(newiris, 3))
K-means clustering with 3 clusters of sizes 38, 62, 50

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

Clustering vector:
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [35] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [69] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2
[103] 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 2 1
[137] 1 1 2 1 1 1 2 1 1 1 2 1 1 2

Within cluster sum of squares by cluster:
[1] 23.87947 39.82097 15.15100
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

Step 2: Compare the Species label with the clustering result.

```
> table(iris$Species, kc$cluster)

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

Step 3: Plot the clusters and their centres. Note that there are four dimensions in the data and that only the first two dimensions are used to draw the plot below.

```
> plot(newiris[c("Sepal.Length", "Sepal.Width")], col=kc$cluster)
```



Step 4: Some black points close to the green centre (asterisk) are actually closer to the black centre in the four-dimensional space.

```
> points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=8, cex=2)
```