ADITYA

COLLEGE OF ENGINEERING & TECHNOLOGY

An AUTONOMOUS Institution

Approved by AICTE, Permanently Affiliated to JNTUK,
Accredited by NBA & NAAC with A+ Grade
Recognized by UGC under Section 2(f) and 12(B) of UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Kakinada District - 533437, A.P.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY RECORD

NAME	:	
ROLL NO	:	
YEAR	:	
SEMESTER	:	
LAB	:	

ADITYA

COLLEGE OF ENGINEERING & TECHNOLOGY

An AUTONOMOUS Institution

Approved by AICTE, Permanently Affiliated to JNTUK,
Accredited by NBA & NAAC with A+ Grade
Recognized by UGC under Section 2(f) and 12(B) of UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Kakinada District - 533437, A.P.

Department of

COMPUTER SCIENCE AND ENGINEERING

Name : Roll No. : | | | | | | | | | | |

Certified that this is the bonafide record of practical work done by

Mr. /Ms. .....

a student of .....with PIN No. .....

in the ..... Laboratory during the year .....

No. of Practicals Conducted : | | No. of Practicals Attended : | |

Signature - Faculty Incharge **Signature - Head of the Department**

Submitted for the Practical examination held on .....

EXAMINER - 1 **EXAMINER - 2**

ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY

INSTITUTE VISION AND MISSION

VISION:

To induce higher planes of learning by imparting technical education with

- ✓ International standards
- ✓ Applied research
- ✓ Creative Ability
- ✓ Value based instruction and to emerge as a premiere institute

MISSION:

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- ✓ Innovative Research And development
- ✓ Industry Institute Interaction
- ✓ Empowered Manpower

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT VISION AND MISSION

VISION:

To become a center for excellence in Computer Science and Engineering education and innovation.

MISSION:

- Provide state of art infrastructure
- Adapt skill-based learner centric teaching methodology
- Organize socio cultural events for better society
- Undertake collaborative works with academia and industry
- Encourage students and staff self-motivated, problem-solving individuals using Artificial Intelligence
- Encourage entrepreneurship in young minds.

Pointer

S No	Date	Name of the Experiment	Page No	Remark

Pointer

S No	Date	Name of the Experiment	Page No	Remark

Pointer

S No	Date	Name of the Experiment	Page No	Remark

EXPERIMENT-1

Aim: Creation of a Data Warehouse:

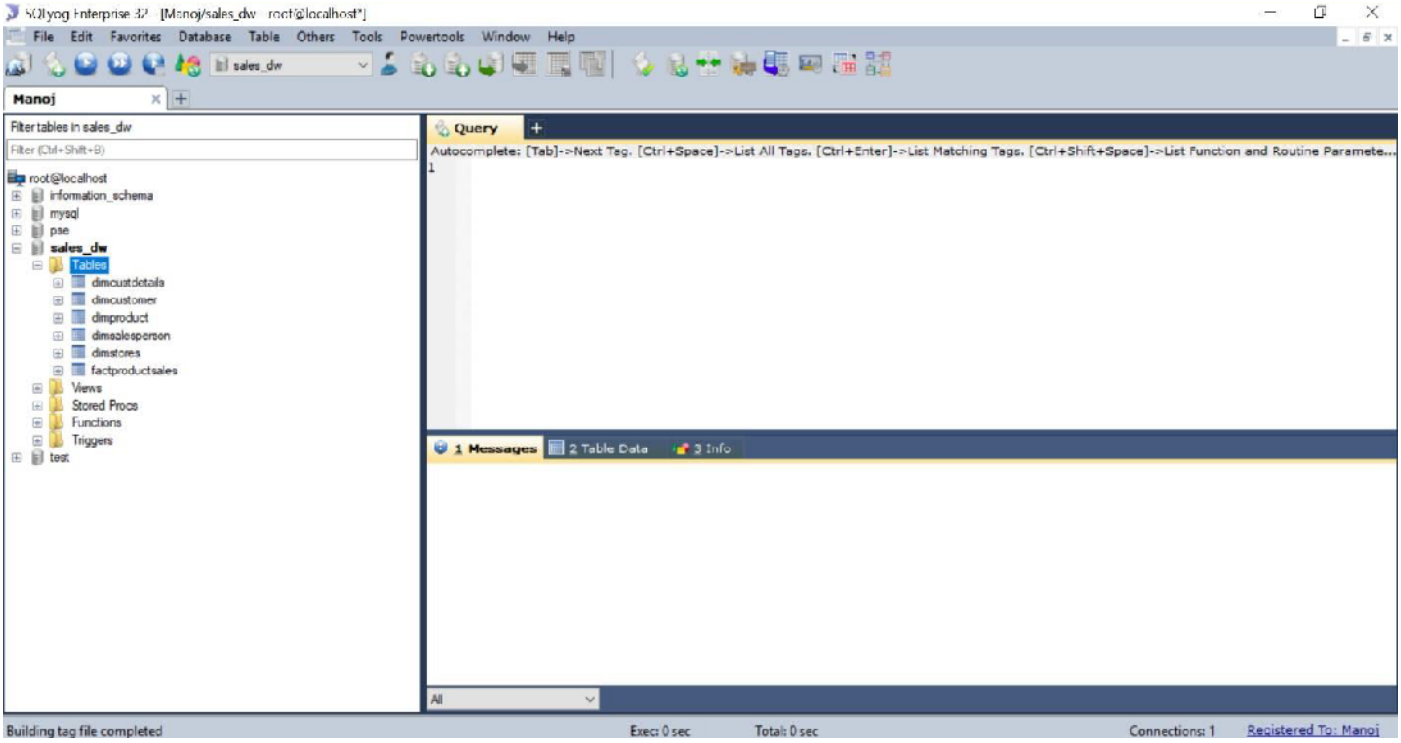
a) Build Data Warehouse/Data Mart (using open source tools like Pentaho Data Integration Tool, Pentaho Business Analytics; or other data warehouse tools like Microsoft-SSIS, Informatica, Business Objects,etc.,)
b) Design multi-dimensional data models namely Star, Snowflake and Fact Constellation schemas for any one enterprise (ex. Banking, Insurance, Finance, Healthcare, manufacturing, Automobiles, sales etc).
c) Write ETL scripts and implement using data warehouse tools.
d) Perform Various OLAP operations such slice, dice, roll up, drill up and pivot .

a) Build Data Warehouse/Data Mart

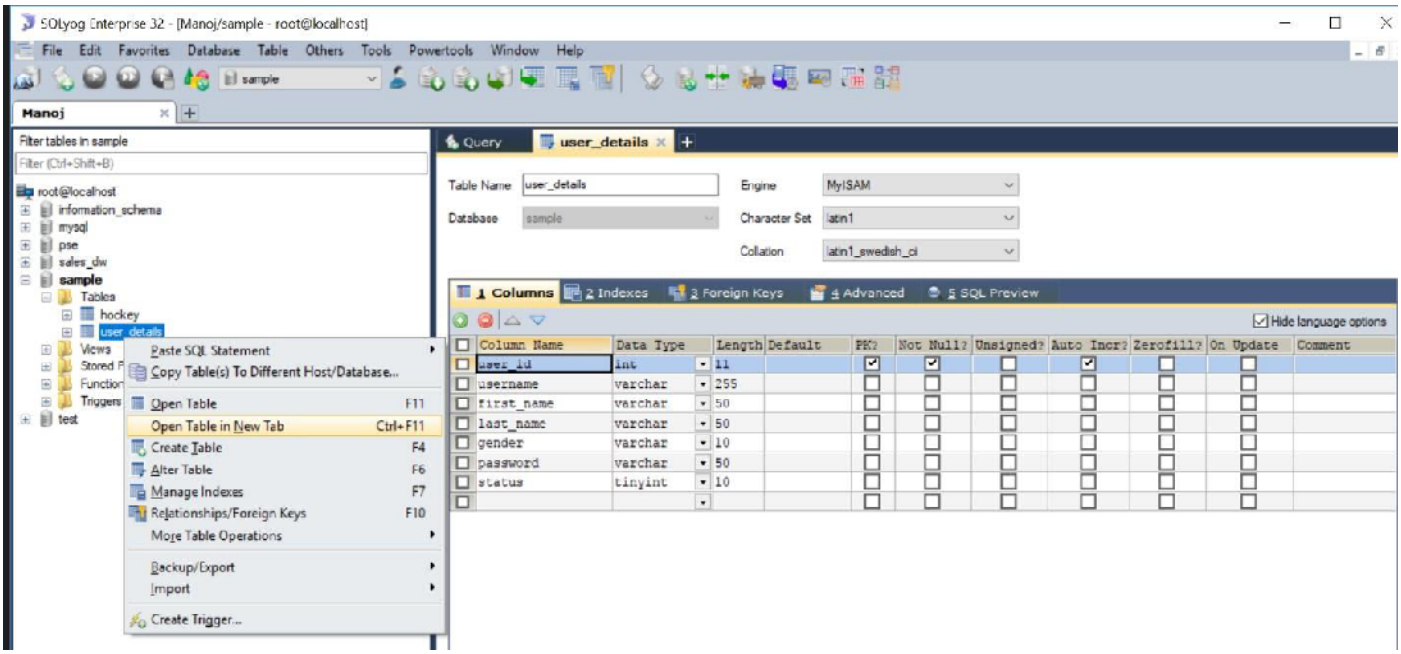
In this task, we are going to use **MySQL administrator, SQLyog Enterprise tools** for building & identifying tables in database & also for populating (filling) the sample data in those tables of a database. A data warehouse is constructed by integrating data from multiple heterogeneous sources. It supports analytical reporting, structured and/or ad hoc queries and decision making. We are building a data warehouse by integrating all the tables in database & analyzing those data. In the below figure we represented MySQL Administrator connection establishment.

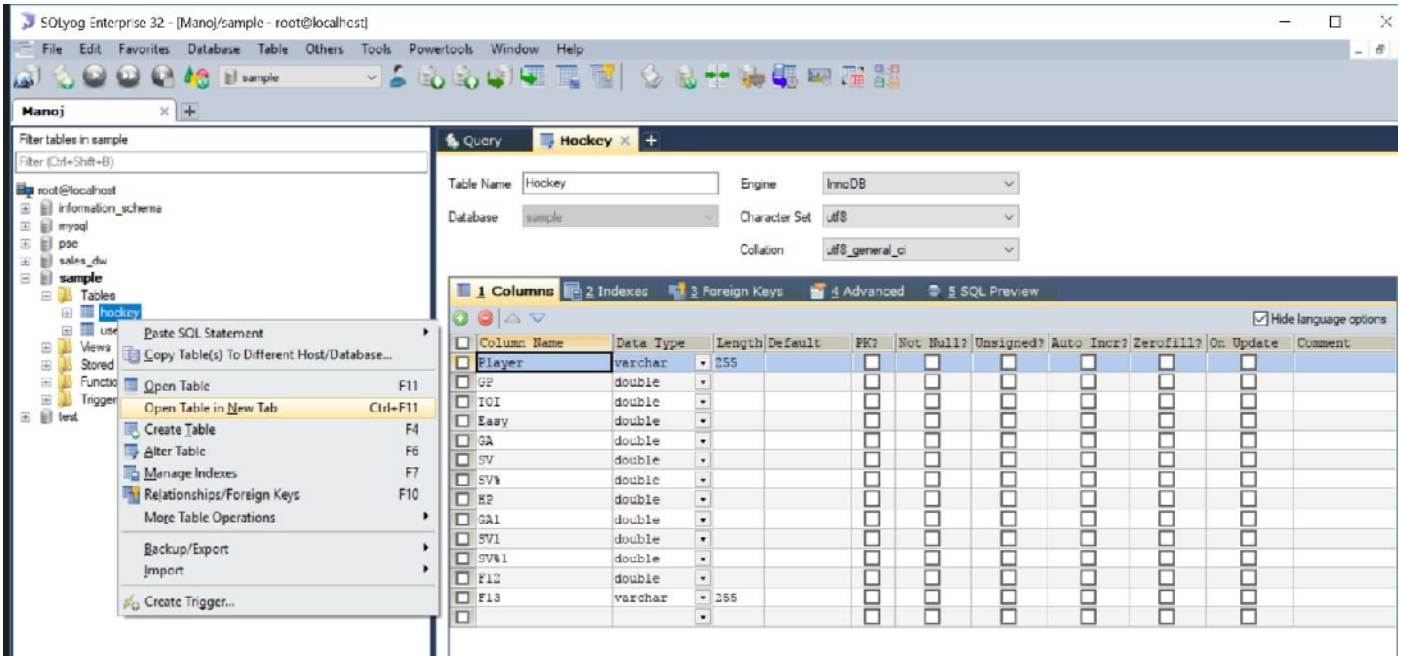
After successful login, it will open new window as shown below

There are different options available in MySQL administrator. Another tool SQLyog Enterprise, we are using for building & identifying tables in a database after successful connection establishment through MySQL Administrator. Below we can see the window of SQLyog Enterprise.



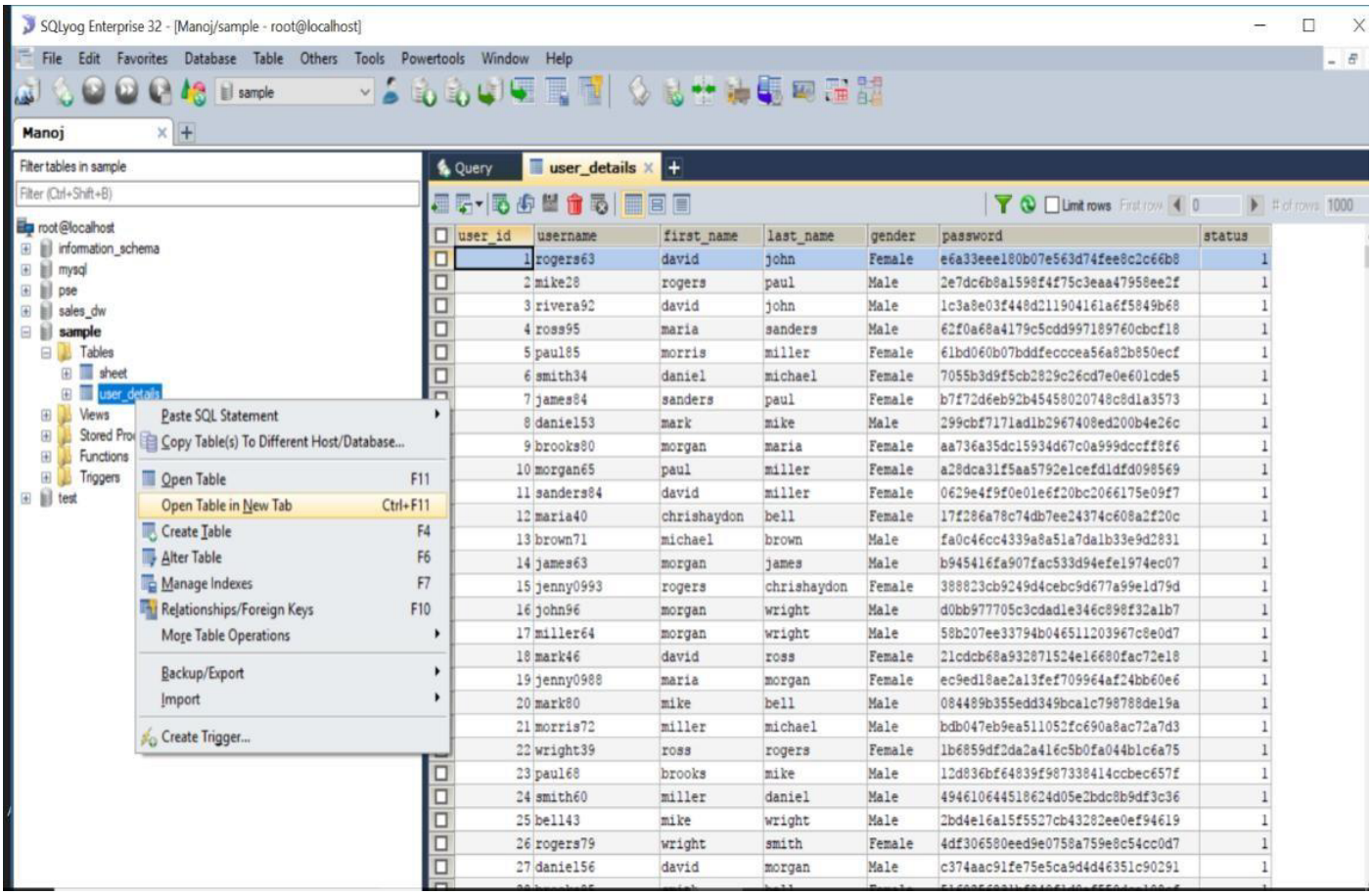
On left-side navigation, we can see different databases & it‘s related tables. Now we are going to build tables & populate table‘s data in database through SQL queries. These tables in database can be used further for building data warehouse.

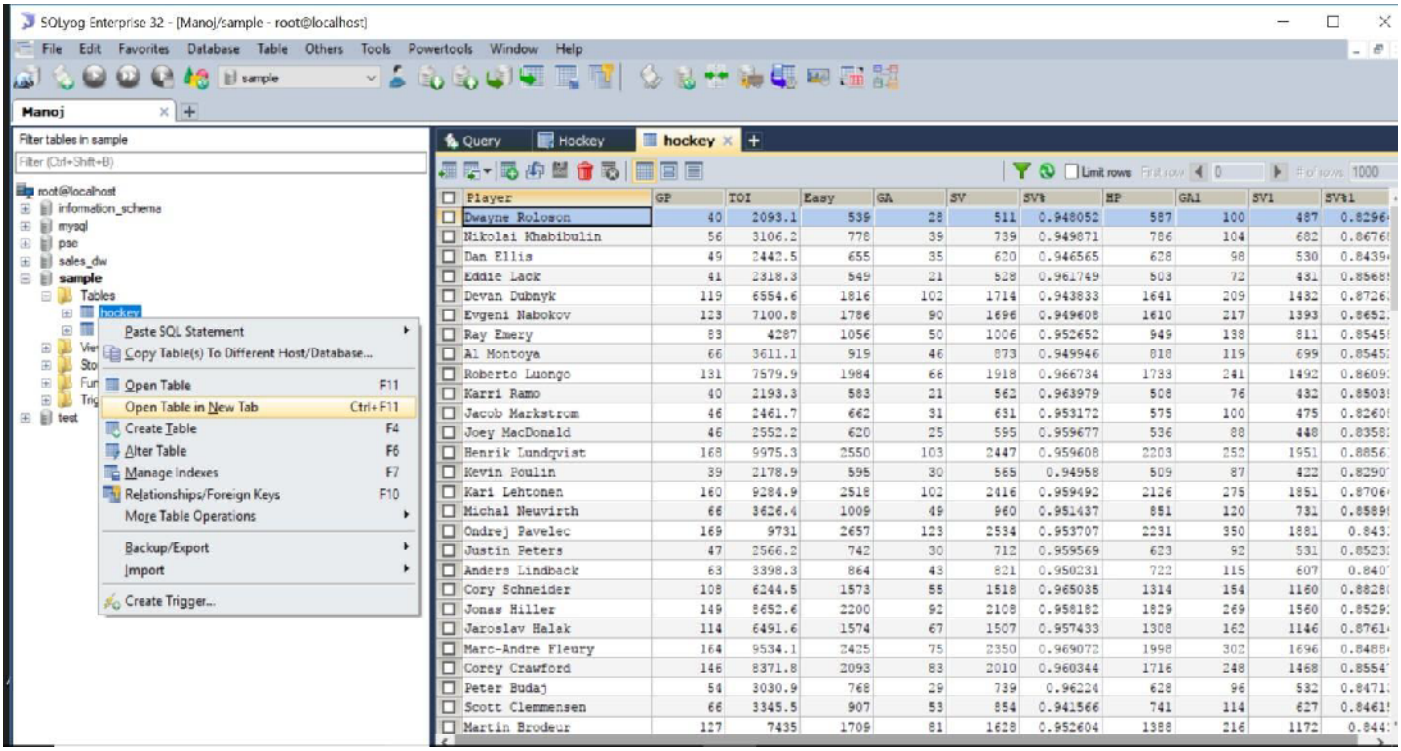




In the above two windows, we created a database named “sample”& in that database we created two tables named as “user_details”& “hockey”through SQL queries.

Now, we are going to populate (filling) sample data through SQL queries in those two created tables as represented in below windows.





Through MySQL administrator & SQLyog, we can import databases from other sources (.XLS,.CSV, .sql) & also we can export our databases as backup for further processing. We can connect MySQL to other applications for data analysis & reporting.

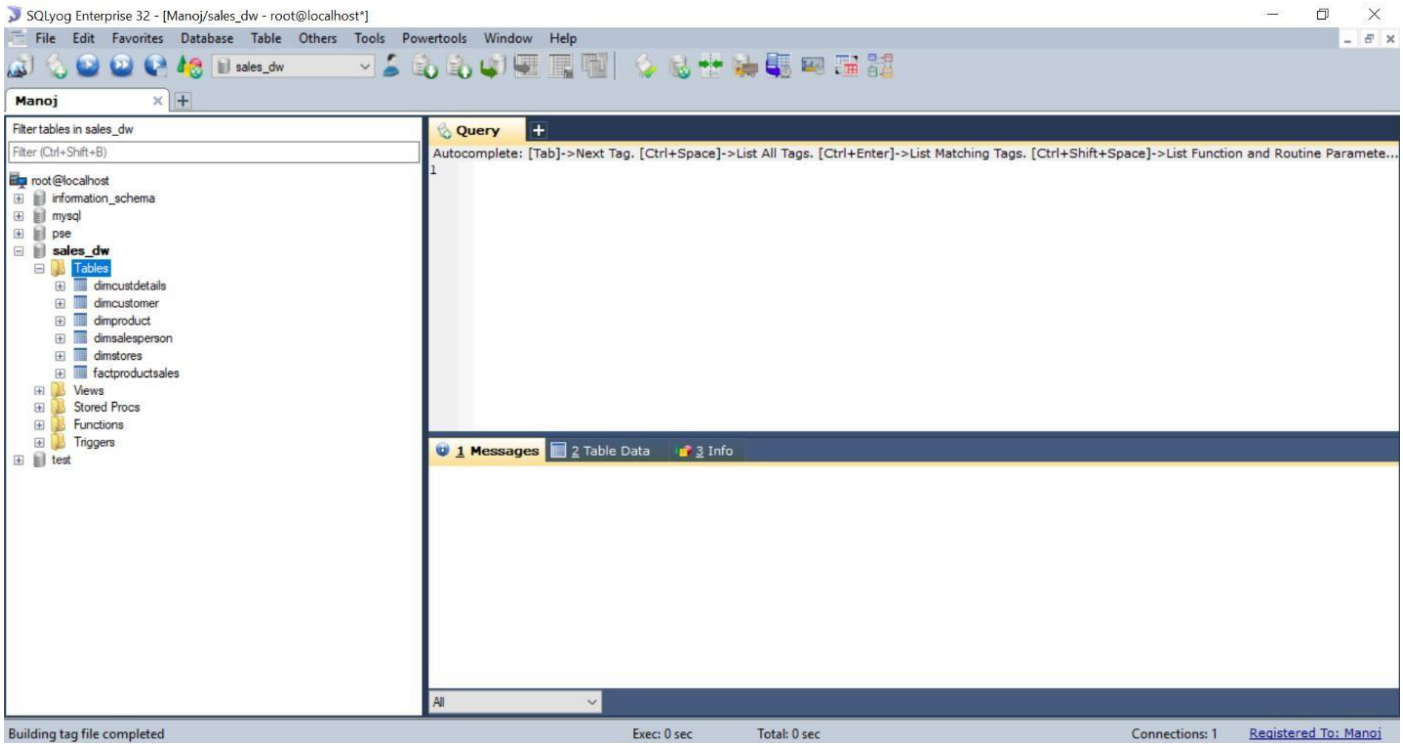
b) Design multi-dimensional data models namely Star, Snowflake and Fact Constellation schemas for any one enterprise (ex. Banking, Insurance, Finance, Healthcare, manufacturing, Automobiles, sales etc).

Multi-Dimensional model was developed for implementing data warehouses & it provides both a mechanism to store data and a way for business analysis. The primary components of dimensional model are dimensions & facts. There are different of types of multi-dimensional data models. They are:

1. Star Schema Model
2. Snow Flake Schema Model
3. Fact Constellation Model.

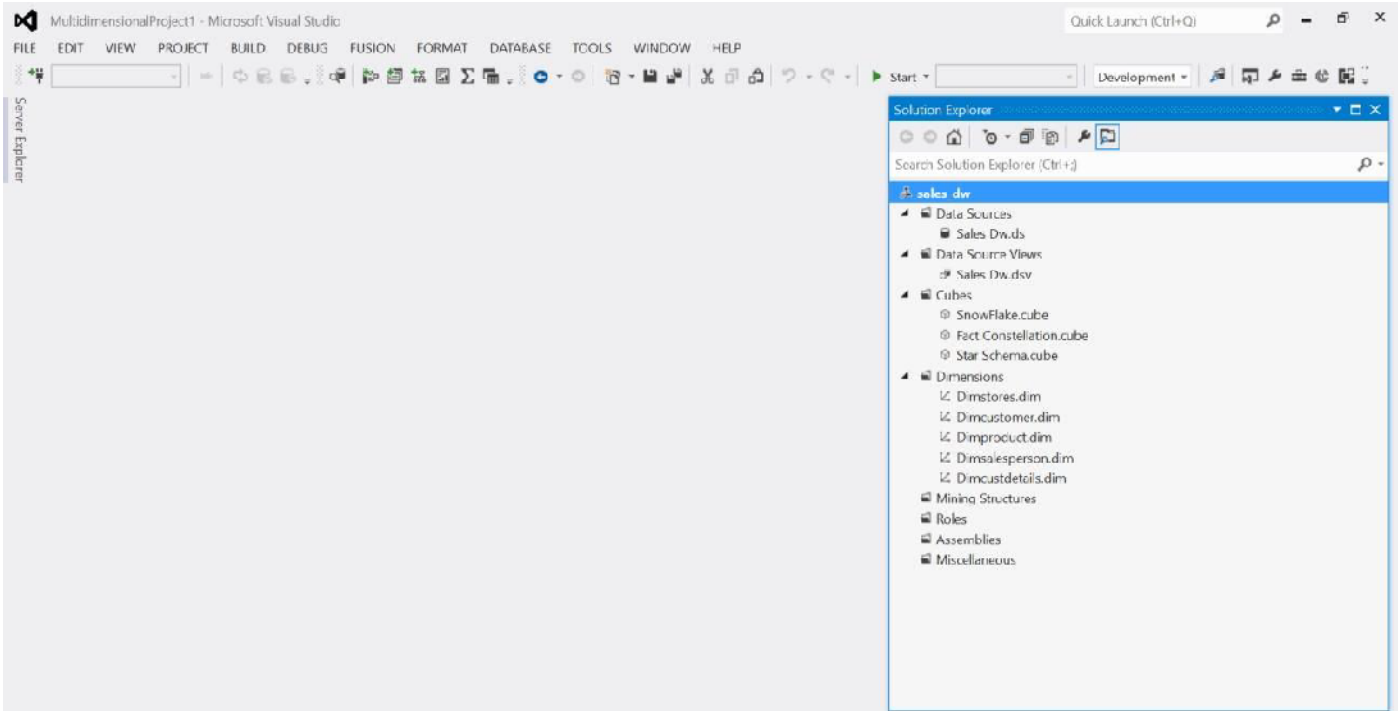
Now, we are going to design these multi-dimensional models for the Marketing enterprise.

First, we need to built the tables in a database through SQLyog as shown below.



In the above window, left side navigation bar consists of a database named as ―sales_dw‖ in which there are six different tables (dimcustdetails, dimcustomer, dimproduct, dimsalesperson, dimstores, factproductsales) has been created.

After creating tables in database, here we are going to use a tool called as **"Microsoft Visual Studio 2012 for Business Intelligence"** for building multi-dimensional models.



In the above window, we are seeing Microsoft Visual Studio before creating a project In which right side navigation bar contains different options like Data Sources, Data Source Views, Cubes, Dimensions etc.

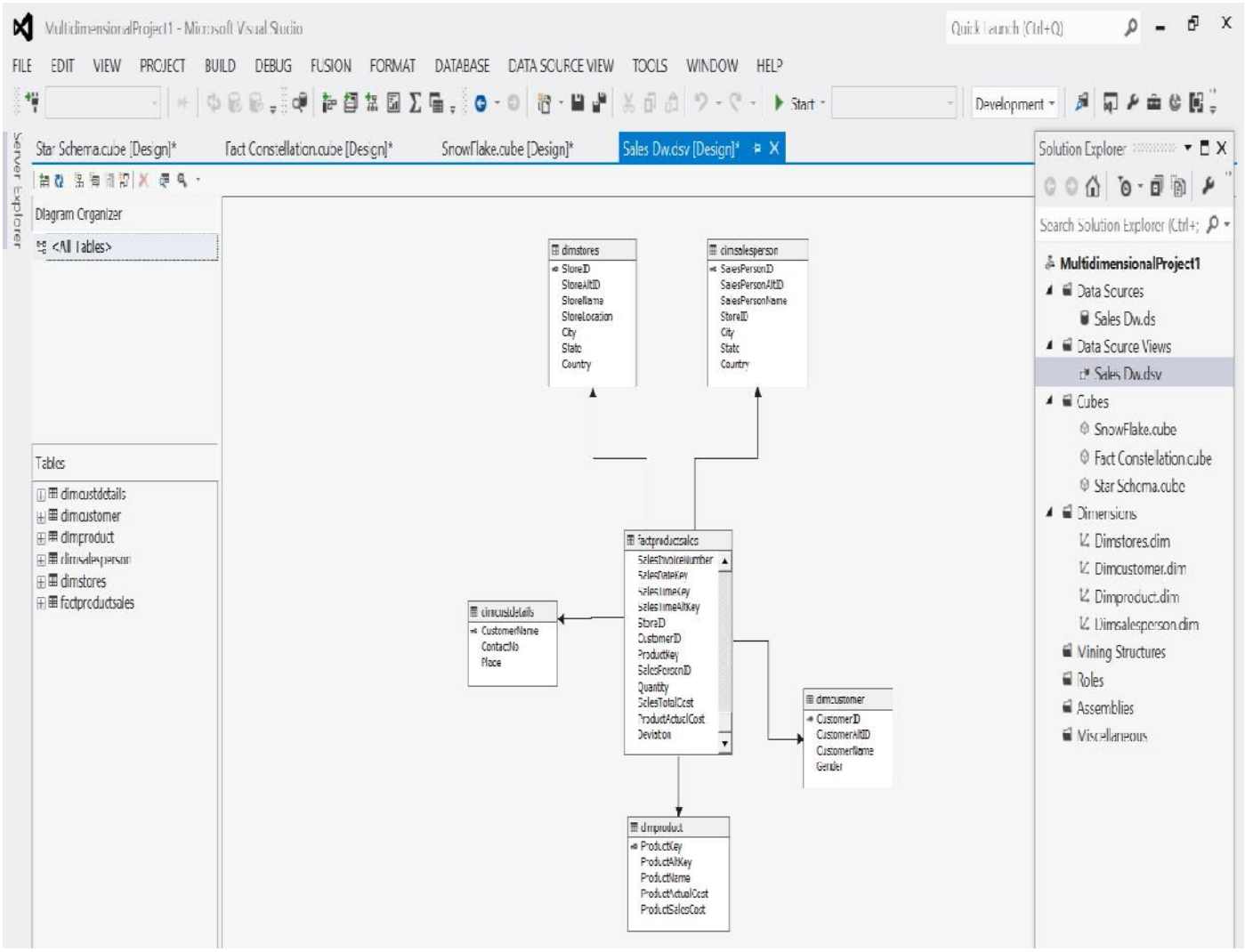
Through Data Sources, we can connect to our MySQL database named as **"sales_dw".**
Then, automatically all the tables in that database will be retrieved to this tool for creating multi-dimensional models.

By data source views & cubes, we can see our retrieved tables in multi-dimensional models. We need to add dimensions also through dimensions option. In general, Multi- dimensional models consists of dimension tables & fact tables.

Star Schema Model:

A Star schema model is a join between a fact table and a no. of dimension tables. Each dimensional table are joined to the fact table using primary key to foreign key join but dimensional tables are not joined to each other. It is the simplest style of dataware house schema.

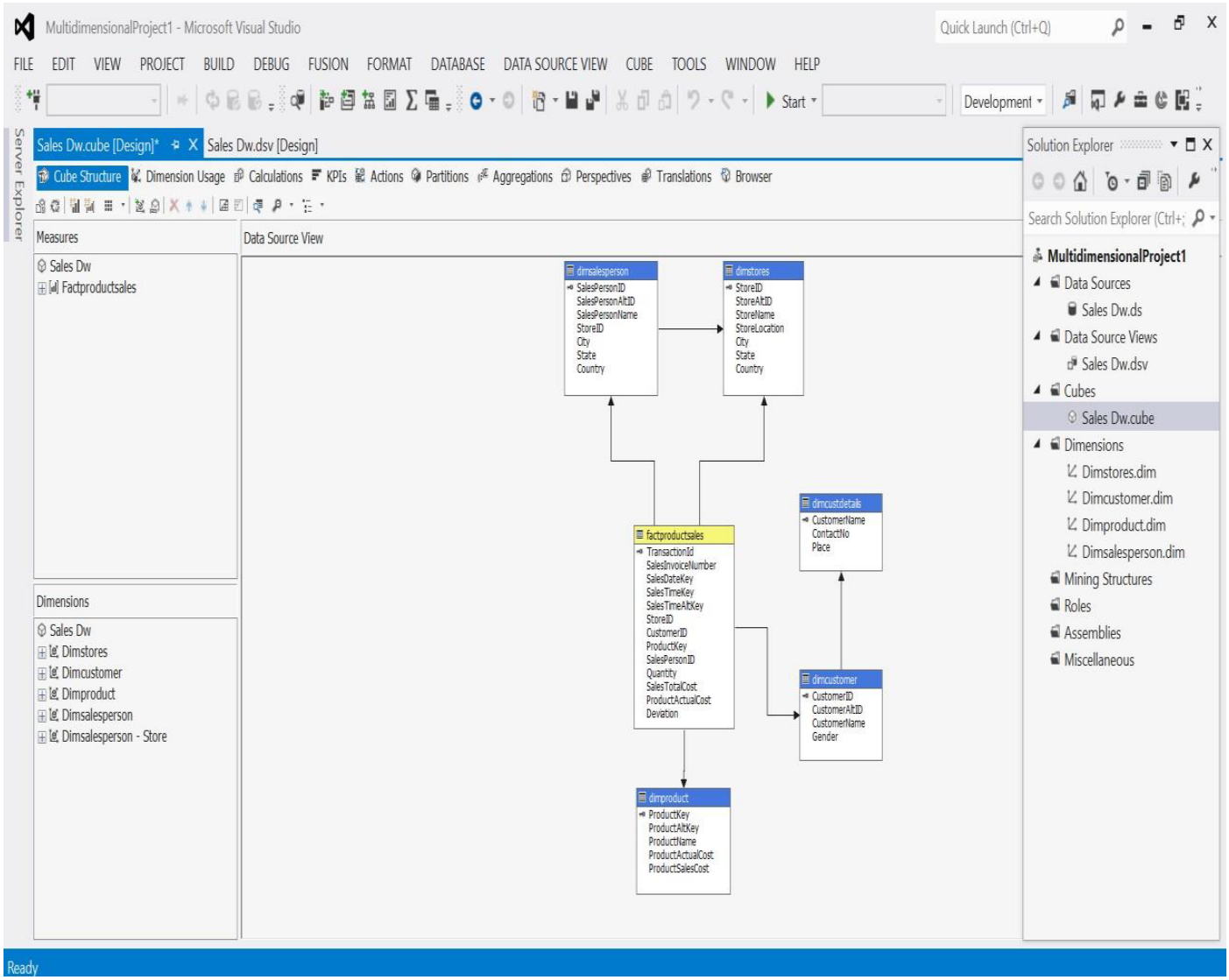
Star schema is a entity relationship diagram of this schema resembles a star with point radiating from central table as we seen in the below implemented window in visual studio.



Snow Flake Schema:

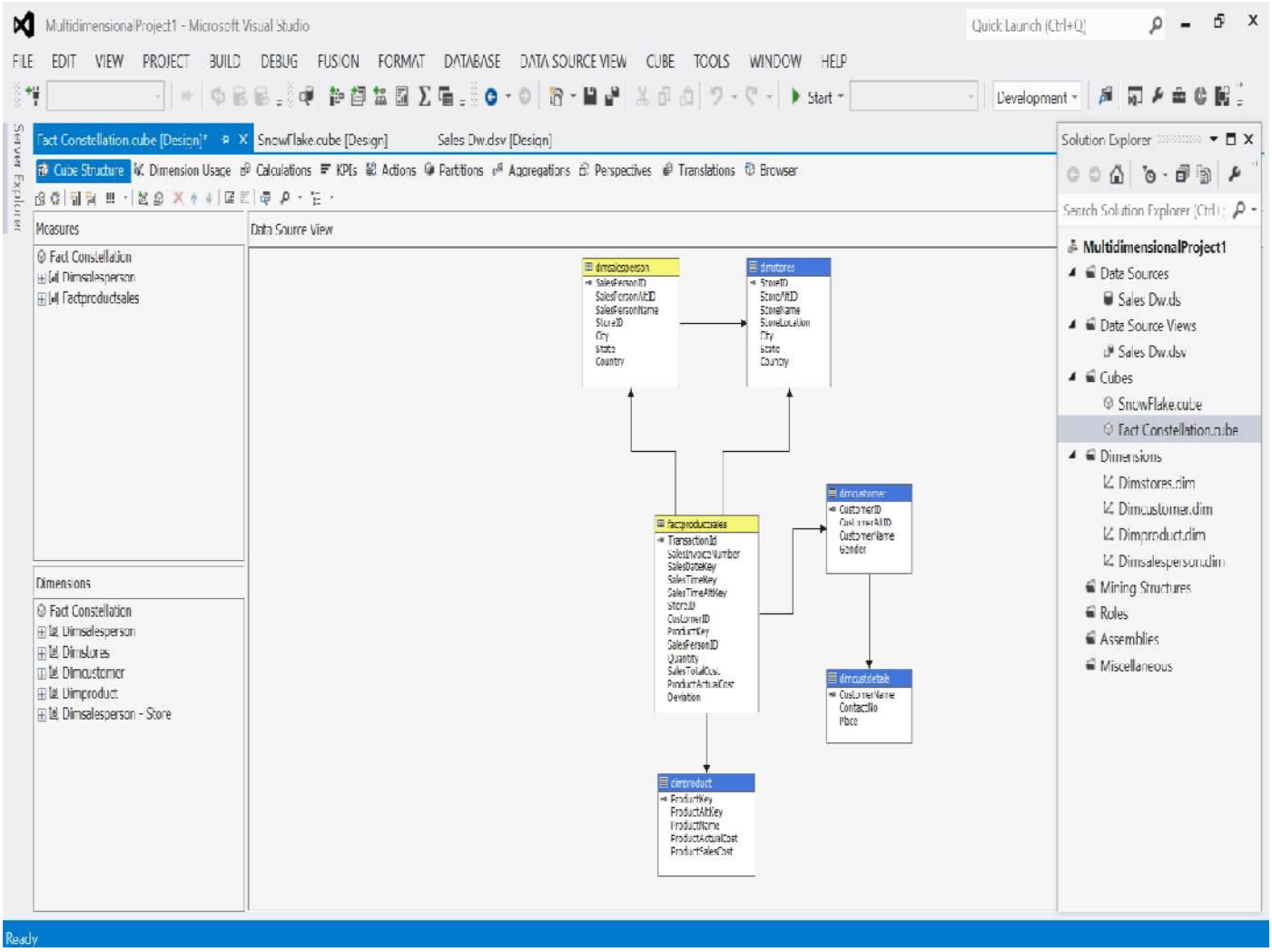
It is slightly different from star schema in which dimensional tables from a star schema are organized into a hierarchy by normalizing them.

Snow flake schema is represented by centralized fact table which are connected to multiple dimension tables. Snow flake effects only dimension tables not fact tables. we developed a snowflake schema for sales_dw database by visual studio tool as shown below.



Fact Constellation Schema:

Fact Constellation is a set of fact tables that share some dimension tables. In this schema there are two or more fact tables. We developed fact constellation in visual studio as shown below. Fact tables are labelled in yellow color.



c) Write ETL scripts and implement using data warehouse tools.

ETL (Extract-Transform-Load):
ETL comes from Data Warehousing and stands for Extract-Transform-Load. ETL covers a process of how the data are loaded from the source system to the data warehouse. Currently, the ETL encompasses a cleaning step as a separate step. The sequence is then Extract-Clean- Transform-Load. Let us briefly describe each step of the ETL process.

Process
Extract:
The Extract step covers the data extraction from the source system and makes it accessible for further processing. The main objective of the extract step is to retrieve all the required data from the source system with as little resources as possible. The extract step should be designed in a way that it does not negatively affect the source system in terms or performance, response time or any kind of locking.
There are several ways to perform the extract:

- Update notification - if the source system is able to provide a notification that a record has been changed and describe the change, this is the easiest way to get the data.
- Incremental extract - some systems may not be able to provide notification that an update has occurred, but they are able to identify which records have been modified and provide an extract of such records. During further ETL steps, the system needs to identify changes and propagate it down. Note, that by using daily extract, we may not be able to handle deleted records properly.
- Full extract - some systems are not able to identify which data has been changed at all, so a full extract is the only way one can get the data out of the system. The full extract requires keeping a copy of the last extract in the same format in order to be able to identify changes. Full extract handles deletions as well.

When using Incremental or Full extracts, the extract frequency is extremely important. Particularly for full extracts; the data volumes can be in tens of gigabytes.

Clean:
The cleaning step is one of the most important as it ensures the quality of the data in the data warehouse. Cleaning should perform basic data unification rules, such as:

- Making identifiers unique (sex categories Male/Female/Unknown, M/F/null, Man/Woman/Not Available are translated to standard Male/Female/Unknown)
- Convert null values into standardized Not Available/Not Provided value
- Convert phone numbers, ZIP codes to a standardized form
- Validate address fields, convert them into proper naming, e.g. Street/St/St./Str./Str
- Validate address fields against each other (State/Country, City/State, City/ZIP code, City/Street).

Transform:
The transform step applies a set of rules to transform the data from the source to the target. This includes converting any measured data to the same dimension (i.e. conformed dimension) using the same units so that they can later be joined. The transformation step also requires joining data from several sources, generating aggregates, generating surrogate keys, sorting, deriving new calculated values, and applying advanced validation rules.

Load:
During the load step, it is necessary to ensure that the load is performed correctly and with as little resources as possible. The target of the Load process is often a database. In order to make the load process efficient, it is helpful to disable any constraints and indexes before the load and enable them back only after the load completes. The referential integrity needs to be maintained by ETL tool to ensure consistency.

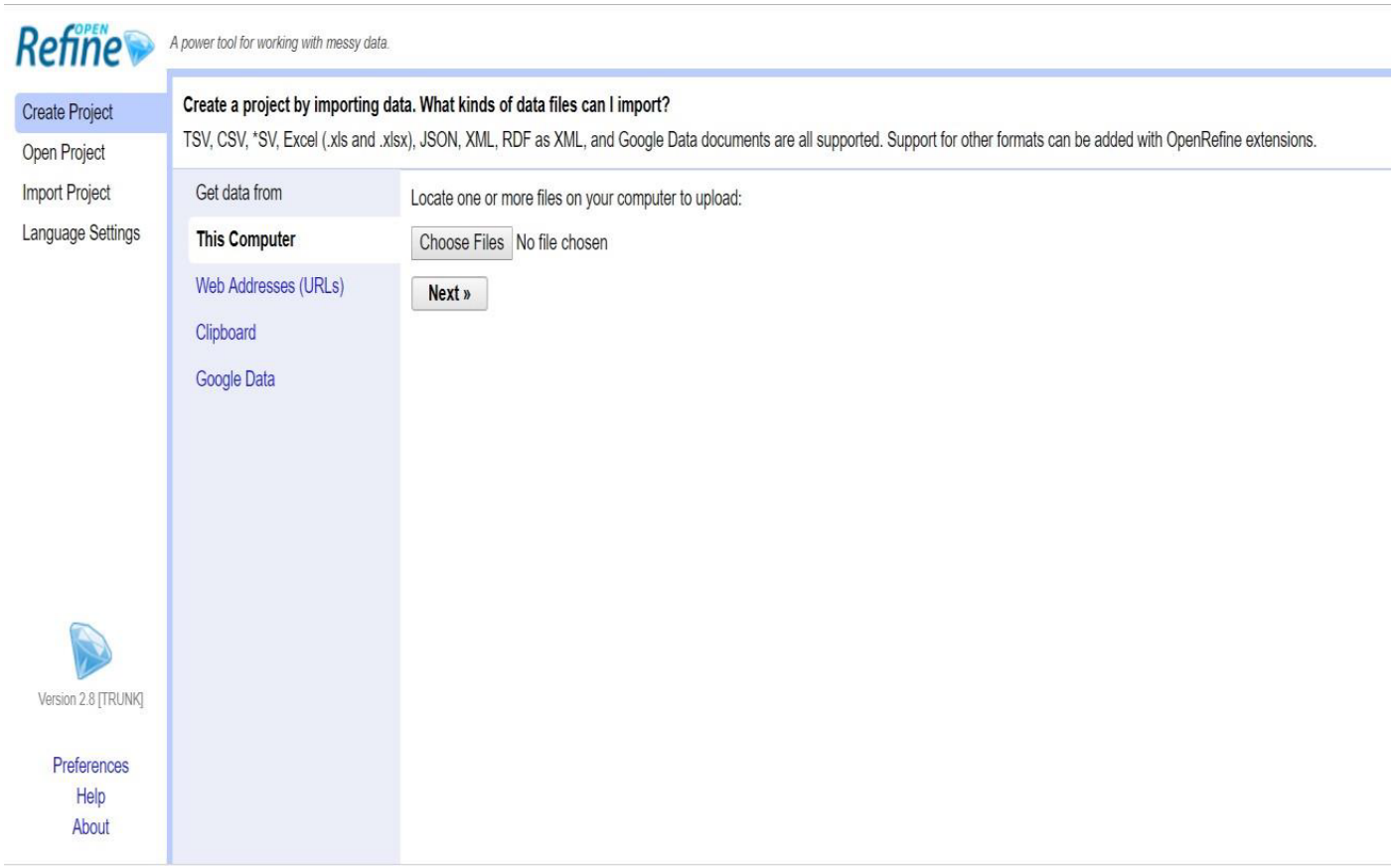
Managing ETL Process:
The ETL process seems quite straight forward. As with every application, there is a possibility that the ETL process fails. This can be caused by missing extracts from one of the systems, missing values in one of the reference tables, or simply a connection or power outage. Therefore, it is necessary to design the ETL process keeping fail-recovery in mind.

Staging:
It should be possible to restart, at least, some of the phases independently from the others. For example, if the transformation step fails, it should not be necessary to restart the Extract step. We can ensure this by implementing proper staging. Staging means that the data is simply dumped to the location (called the Staging Area) so that it can then be read by the next processing phase. The staging area is also used during ETL process to store intermediate results of processing. This is ok for the ETL process which uses for this purpose. However, tThe staging area should is be accessed by the load ETL process only. It should never be available to anyone else; particularly not to end users as it is not intended for data presentation to the end-user.may contain incomplete or in-the-middle-of-the-processing data.

ETL Tool Implementation:
When you are about to use an ETL tool, there is a fundamental decision to be made: will the company build its own data transformation tool or will it use an existing tool?
Building your own data transformation tool (usually a set of shell scripts) is the preferred approach for a small number of data sources which reside in storage of the same type. The reason for that is the effort to implement the necessary transformation is little due to similar data structure and common system architecture. Also, this approach saves licensing cost and there is no need to train the staff in a new tool. This approach, however, is dangerous from the TOC point of view. If the transformations become more sophisticated during the time or there is a need to integrate other systems, the complexity of such an ETL system grows but the manageability drops significantly. Similarly, the implementation of your own tool often resembles re-inventing the wheel.
There are many ready-to-use ETL tools on the market. The main benefit of using off-the-shelf ETL tools is the fact that they are optimized for the ETL process by providing connectors to common data sources like databases, flat files, mainframe systems, xml, etc. They provide a means to implement data transformations easily and consistently across various data sources. This includes filtering, reformatting, sorting, joining, merging, aggregation and other operations ready to use. The tools also support transformation scheduling, version control, monitoring and unified metadata management. Some of the ETL tools are even integrated with BI tools.

Some of the Well Known ETL Tools:
The most well-known commercial tools are Ab Initio, IBM InfoSphere DataStage, Informatica, Oracle Data Integrator, and SAP Data Integrator.
There are several open source ETL tools are **OpenRefine**, Apatar, CloverETL, Pentaho and Talend.

In these above tools, we are going to use **OpenRefine 2.8 ETL tool**to different sample datasets forextracting, data cleaning, transforming & loading.


OpenRefine
A power tool for working with messy data.
Create Project
Open Project
Import Project
Language Settings
Create a project by importing data. What kinds of data files can I import?
TSV, CSV, *SV, Excel (.xls and .xlsx), JSON, XML, RDF as XML, and Google Data documents are all supported. Support for other formats can be added with OpenRefine extensions.
Get data from
This Computer
Web Addresses (URLs)
Clipboard
Google Data
Locate one or more files on your computer to upload:
Choose Files
No file chosen
Next »
Version 2.8 [TRUNK]
Preferences
Help
About

d) Perform Various OLAP operations such slice, dice, roll up, drill up and pivot .

OLAP Operations:
Since OLAP servers are based on multidimensional view of data, we will discuss OLAP operations in multidimensional data.
Here is the list of OLAP operations

- Roll-up (Drill-up)
- Drill-down
- Slice and dice
- Pivot (rotate)

Roll-up (Drill-up):
Roll-up performs aggregation on a data cube in any of the following ways

- By climbing up a concept hierarchy for a dimension
- By dimension reduction
- Roll-up is performed by climbing up a concept hierarchy for the dimension location.
- Initially the concept hierarchy was "street < city < province < country".
- On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
- The data is grouped into cities rather than countries.
- When roll-up is performed, one or more dimensions from the data cube are removed

Drill-down:
Drill-down is the reverse operation of roll-up. It is performed by either of the following ways

- By stepping down a concept hierarchy for a dimension
- By introducing a new dimension.
- Drill-down is performed by stepping down a concept hierarchy for the dimension time.
- Initially the concept hierarchy was "day < month < quarter < year."
- On drilling down, the time dimension is descended from the level of quarter to the level of month.
- When drill-down is performed, one or more dimensions from the data cube are added.
- It navigates the data from less detailed data to highly detailed data.

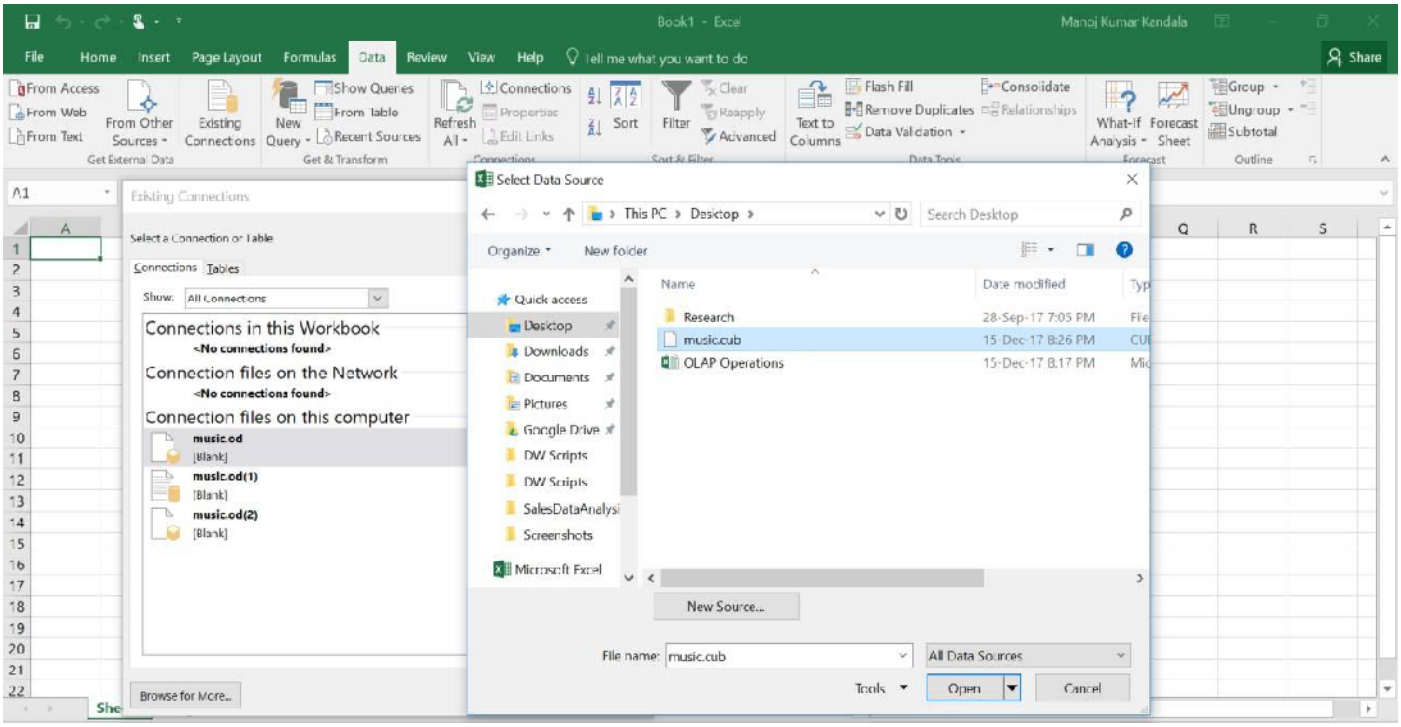
Slice:
The slice operation selects one particular dimension from a given cube and provides a new sub-cube.

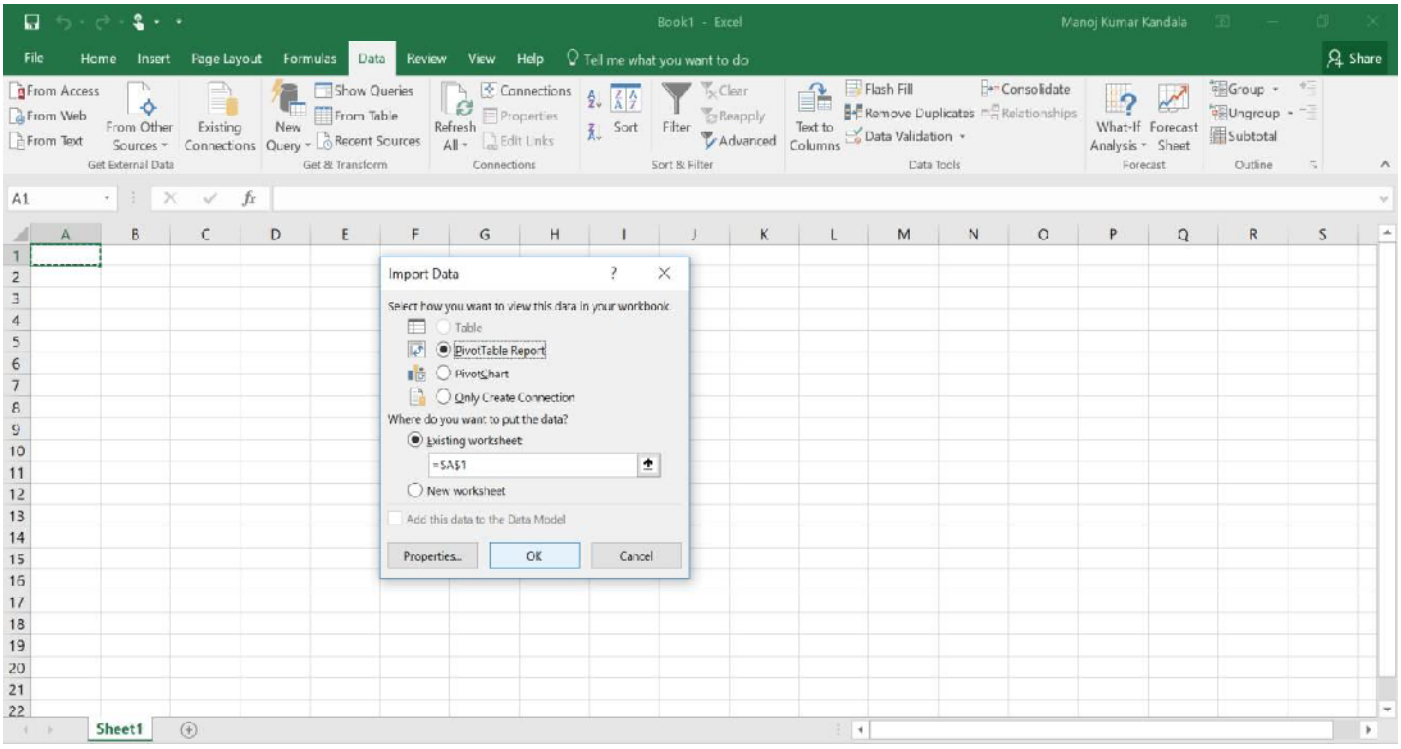
Dice:
Dice selects two or more dimensions from a given cube and provides a new sub-cube.

Pivot (rotate):
The pivot operation is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data.

Now, we are practically implementing all these OLAP Operations using **Microsoft Excel.**

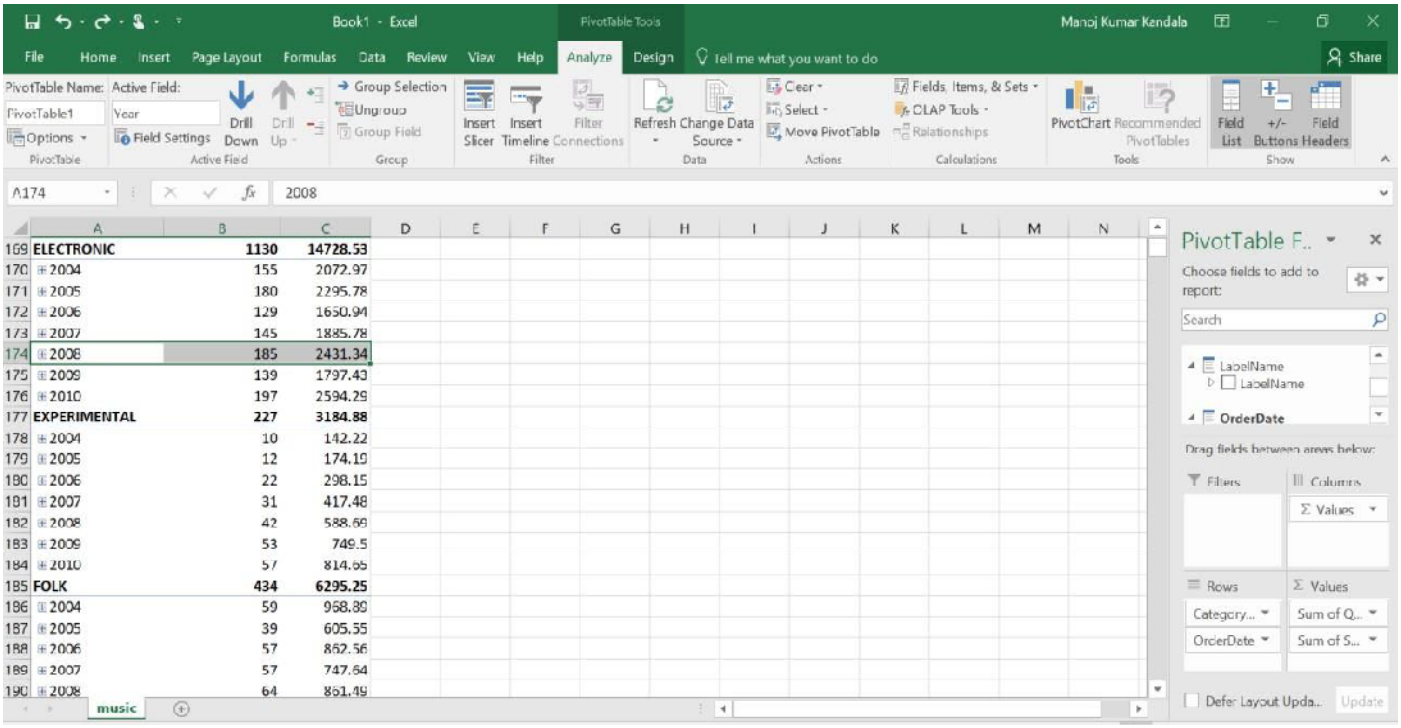
Procedure for OLAP Operations:
1. Open Microsoft Excel, go to**Data** tab in top & click on **—Existing Connections”**.
2. Existing Connections window will be opened, there **“Browse for more”**option should be clicked for importing **.cub extension file** for performing OLAP Operations. For sample, I took **music.cub** file.



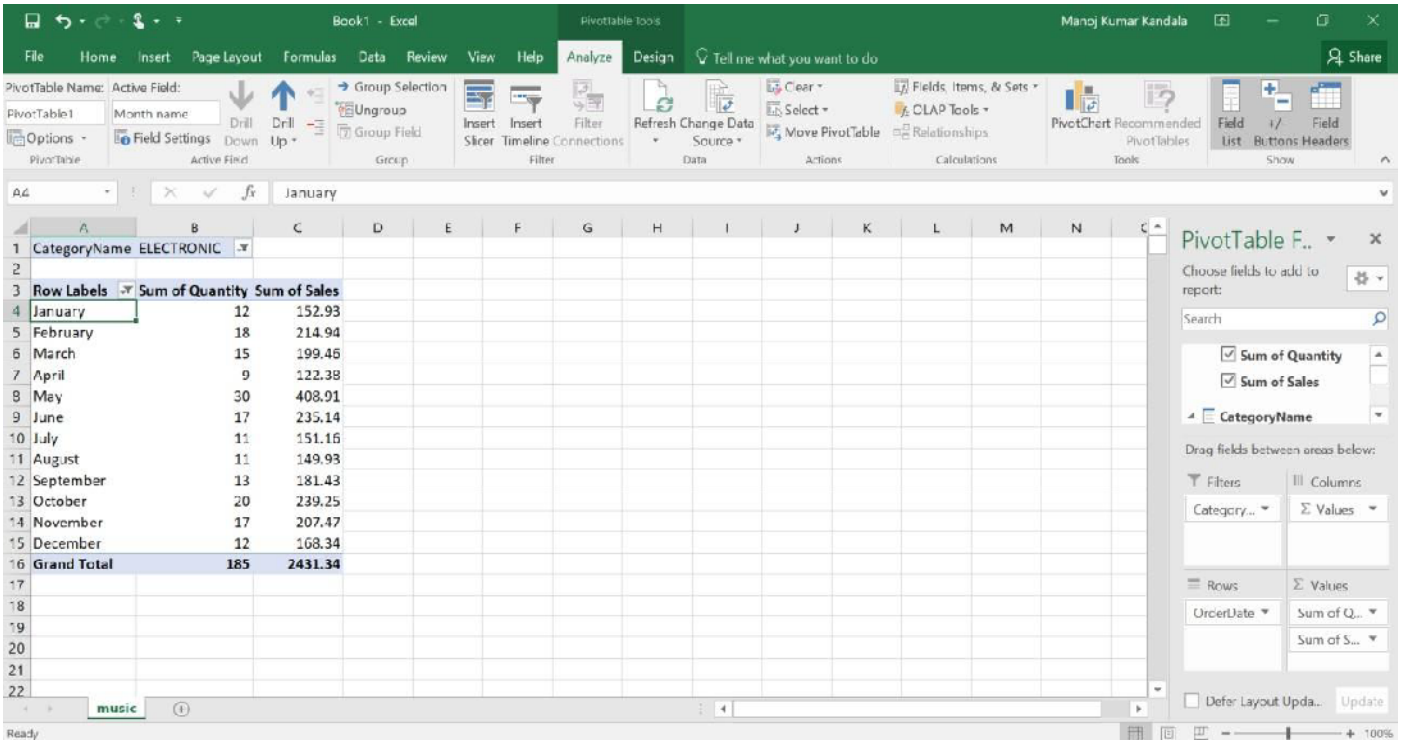


3. As shown in above window, select **—PivotTable Report"** and click **"OK".**

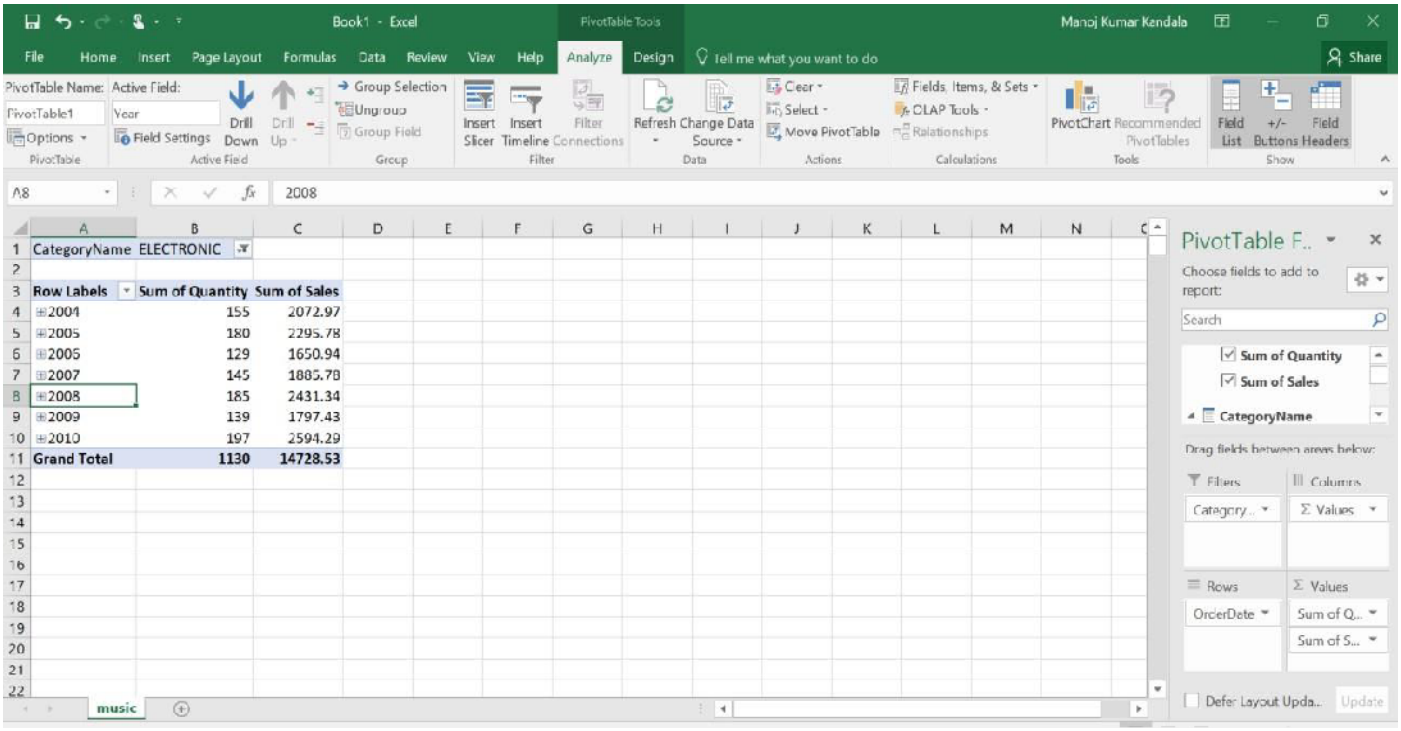
4. We got all the **music.cub** data for analyzing different OLAP Operations.Firstly, we performed **drill-down operation** as shown below.



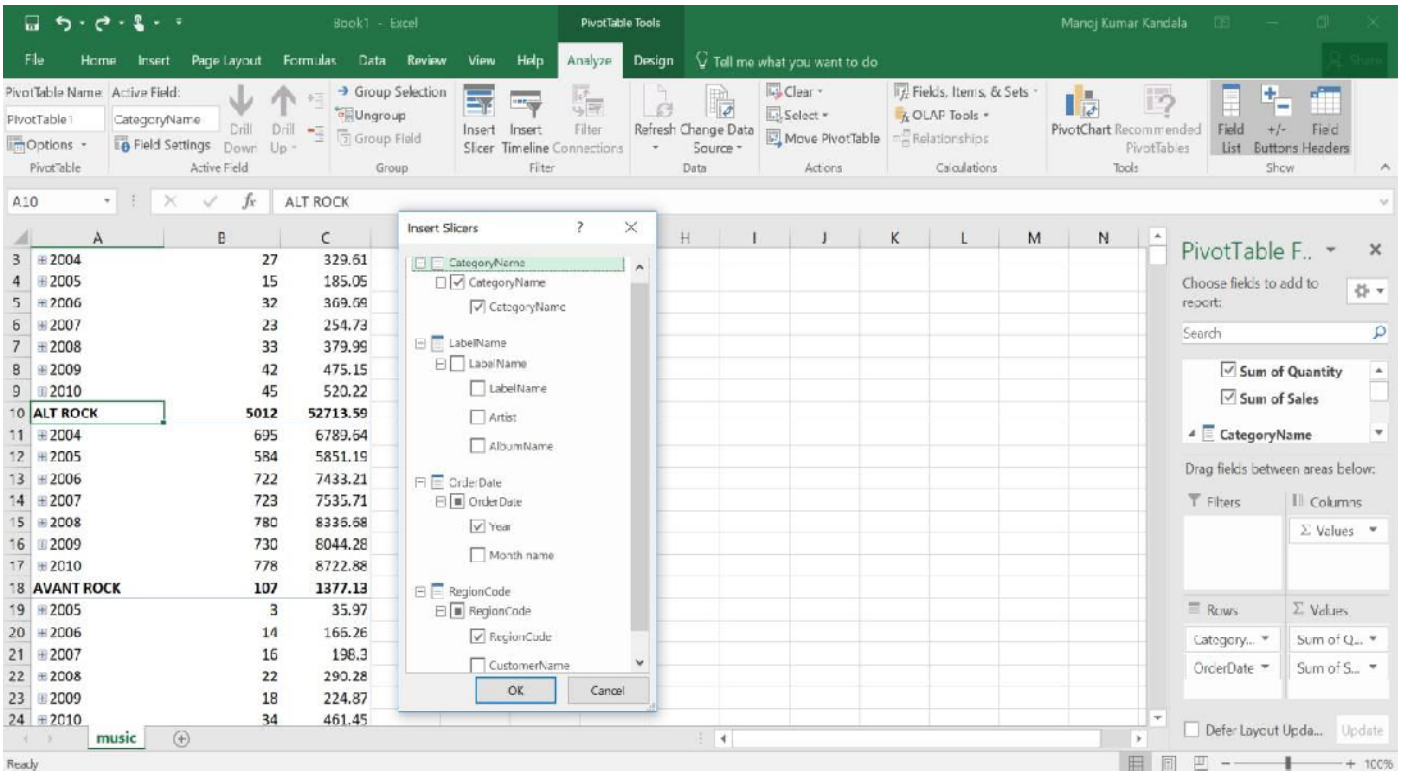
In the above window, we selected year **„2008"** in **„Electronic"** Category, then automatically the Drill-Down option is enabled on top navigation options. We will click on **„Drill-Down"** option, then the below window will be displayed.



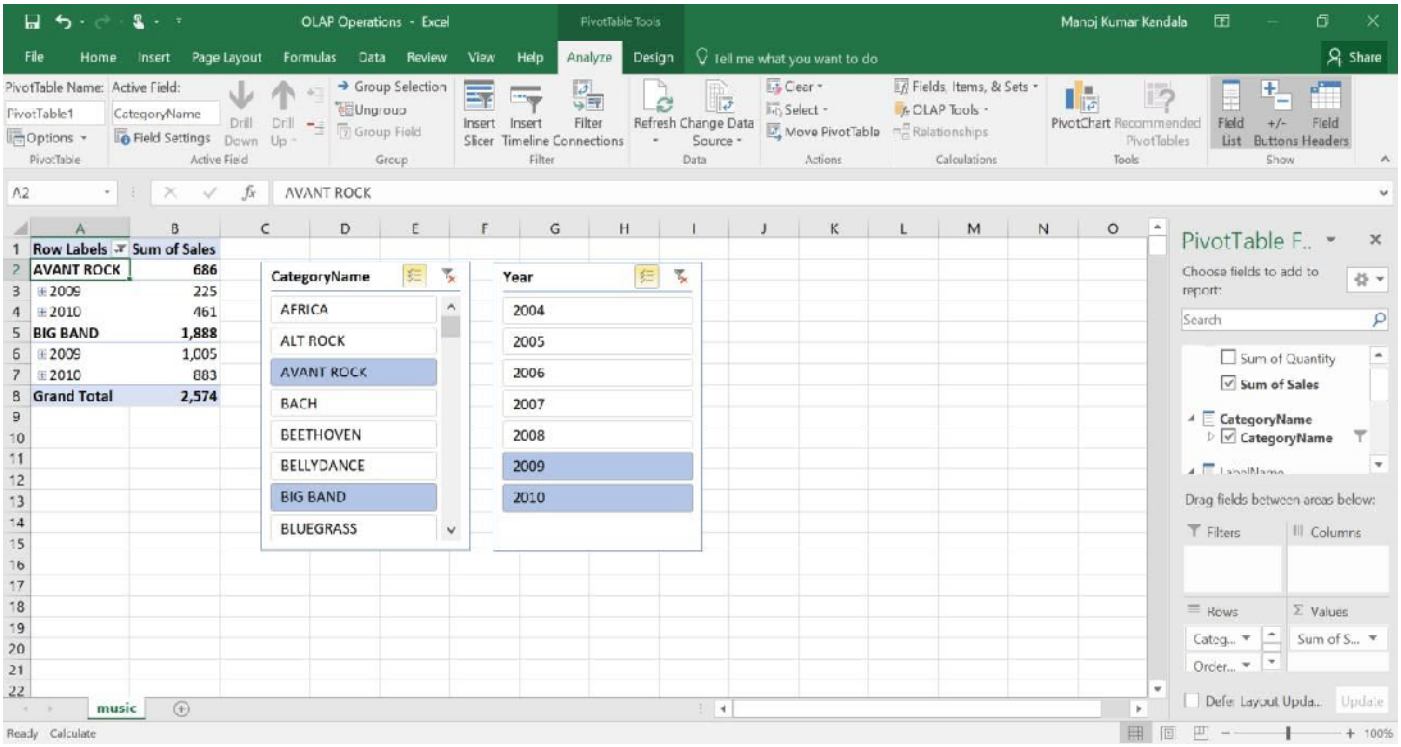
5. Now we are going to perform **roll-up (drill-up) operation**, in the above window I selected January month then automatically **Drill-up option** is enabled on top. We will click on **Drill-up** option, then the below window will be displayed.



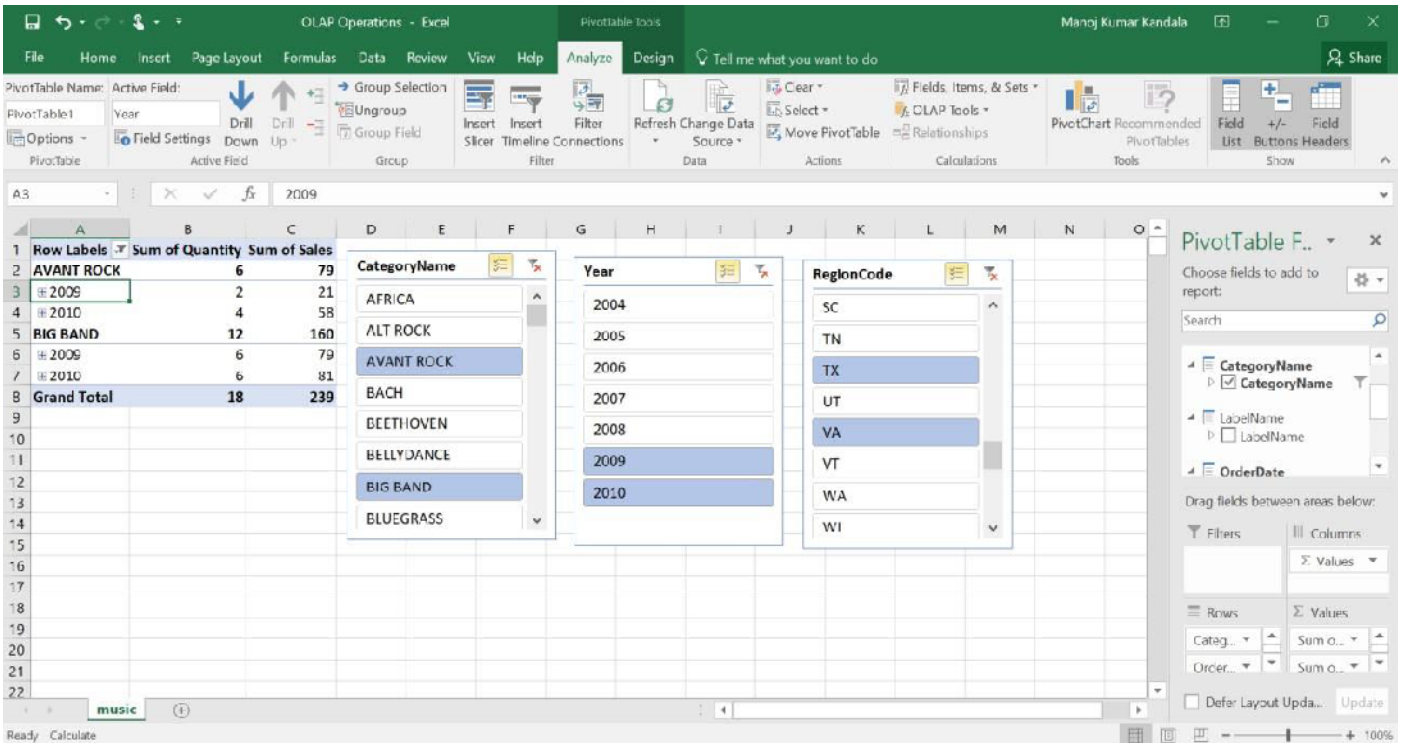
6. Next OLAP operation **Slicing** is performed by inserting slicer as shown in top navigation options.



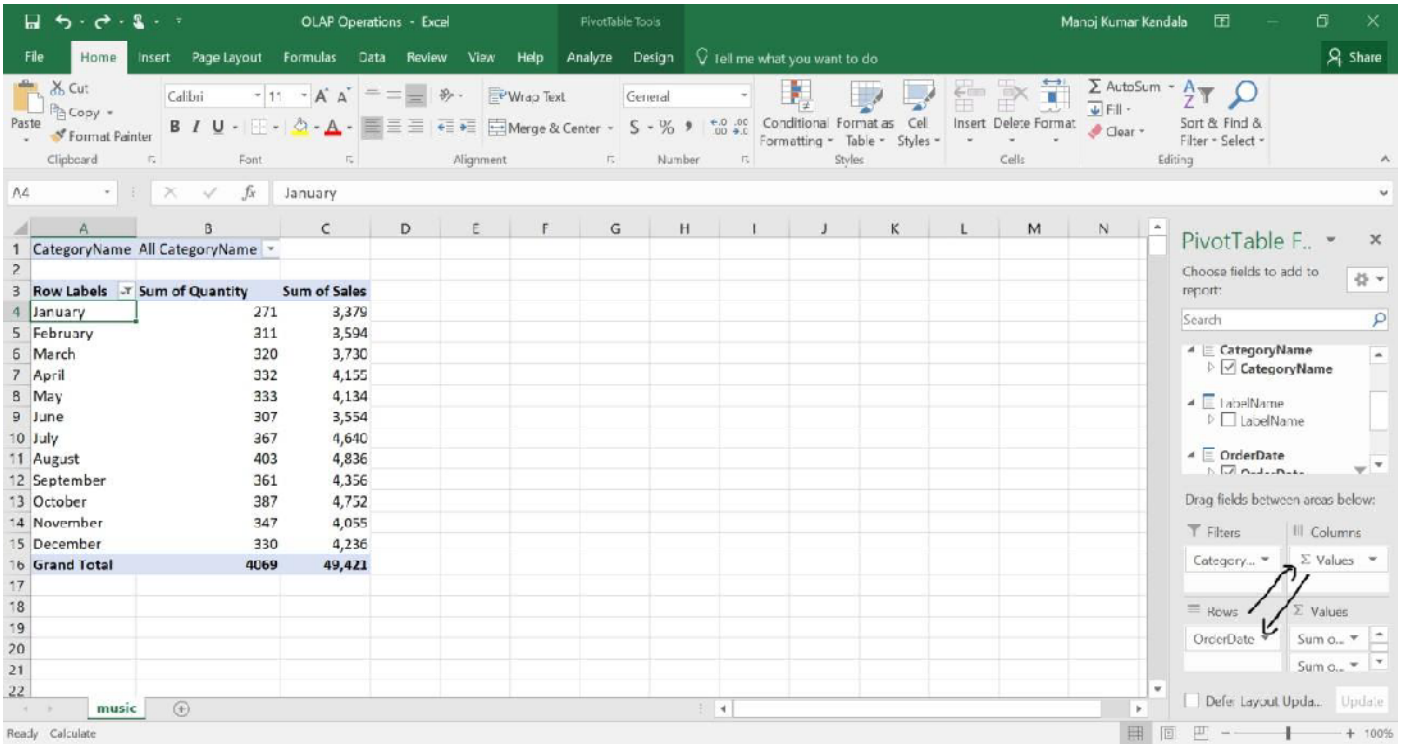
While inserting slicers for slicing operation, we select 2 Dimensions (for e.g. CategoryName & Year) only with one Measure (for e.g. Sum of sales).After inserting a slice& adding a filter (CategoryName: AVANT ROCK & BIG BAND; Year: 2009 & 2010), we will get table as shown below.



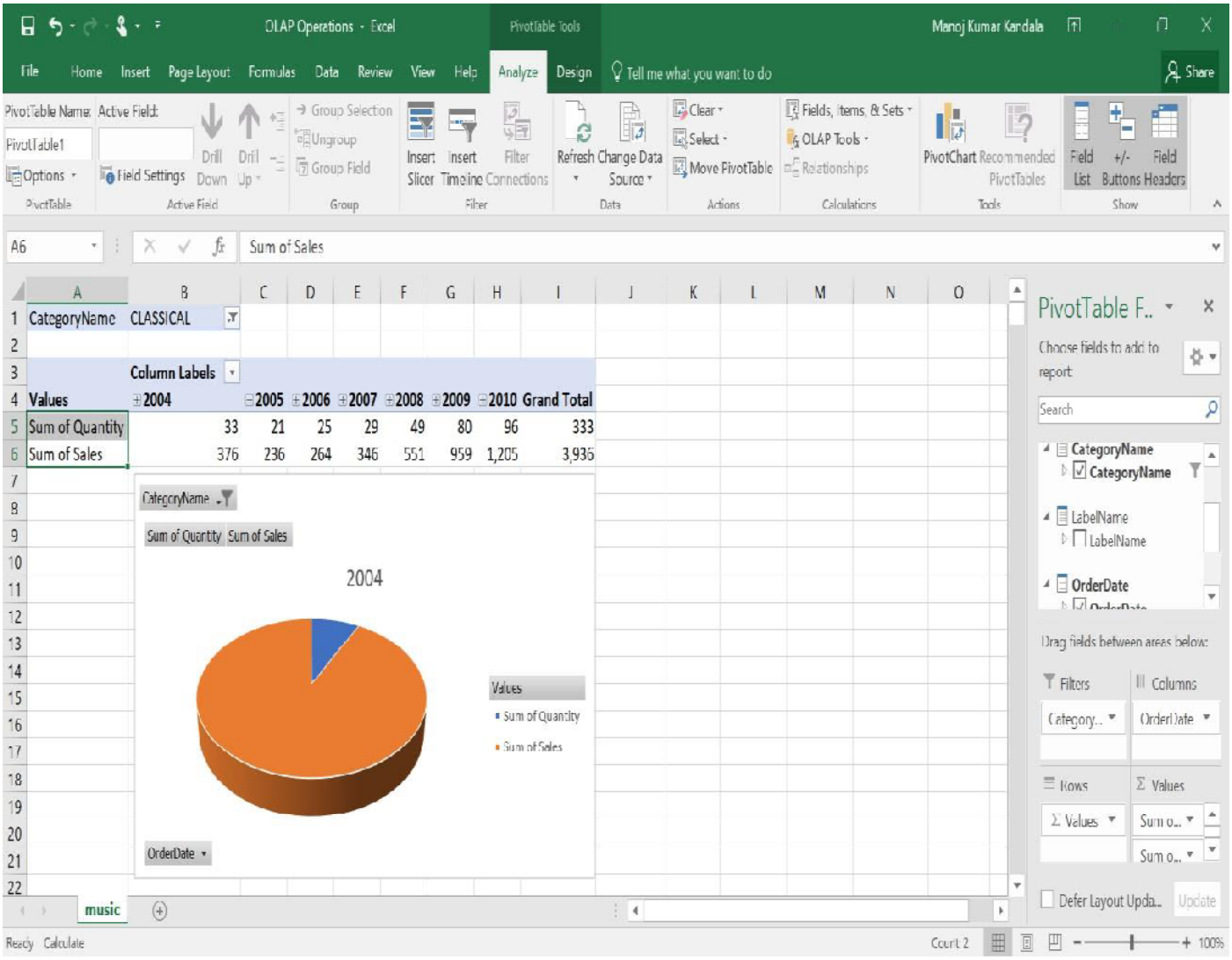
7. Dicing operation is similar to Slicing operation. Here we are selecting 3 dimensions (CategoryName, Year, RegionCode)& 2 Measures (Sum of Quantity, Sum of Sales) through **„insert slicer"** option. After that adding a filter for CategoryName, Year & RegionCode as shown below.



8. Finally, the **Pivot (rotate)** OLAP operation is performed by swapping rows (Order Date-Year) & columns (Values-Sum of Quantity & Sum of Sales) through right side bottom navigation bar as shown below.



After Swapping (rotating), we will get resultant as represented below with a pie-chart for Category-Classical& Year Wise data.



EXPERIMENT-2

Aim: Explore machine learning tool "WEKA"

a) Explore WEKA Data Mining/Machine Learning Toolkit.
b) Downloading and/or installation of WEKA data mining toolkit.
c) Understand the features of WEKA toolkit such as Explorer, Knowledge Flow interface, Experimenter, command-line interface.
d) Navigate the options available in the WEKA (ex. Select attributes panel, Preprocess panel, Classify panel, Cluster panel, Associate panel and Visualize panel)
e) Study the arff file format Explore the available data sets in WEKA. Load a data set (ex. Weather dataset, Iris dataset, etc.)
f) Load each dataset and observe the following:
    1. List the attribute names and they types
    2. Number of records in each dataset
    3. Identify the class attribute (if any)
    4. Plot Histogram
    5. Determine the number of records for each class.
    6. Visualize the data in various dimensions

a) Explore WEKA Data Mining/Machine Learning Toolkit.

WEKA:

- ***"Weka"*** *Named after a flightless New Zealand bird,*
- Weka is data mining software that uses a collection of machine learning algorithms.
- These algorithms can be applied directly to the data or called from the Java code.
- Weka is a collection of tools for Data pre-processing, Classification, Association, Clustering, Visualization, Regression
- Weka was developed at the University of Waikato in New Zealand; the name stands for "***Waikato Environment for Knowledge Analysis".***
- The system is written in Java and distributed under the terms of the GNU General Public License.
- It runs on almost any platform and has been tested under Linux, Windows, and Macintosh operating systems.

Machine learning is nothing but a type of artificial intelligence which enables computers to learn the data without help of any explicit programs. Machine learning systems crawl through the data to find the patterns and, when these are found, adjust the program's actions accordingly.

Data mining analyses the data from different perspectives and summaries it into parcels of useful information. The machine learning method is similar to data mining. The difference is that data mining systems extract the data for human comprehension. Data mining uses machine language to find valuable information from large volumes of data

Key Features responsible for Weka's success are:

- it provides many different algorithms for data mining and machine learning
- is is open source and freely available

- it is platform-independent
- it is easily useable by people who are not data mining specialists
- it provides flexible facilities for scripting experiments
- it has kept up-to-date, with new algorithms being added as they appear in the research literature.

b) Downloading and/or installation of WEKA data mining toolkit.

Steps to install weka on windows machine are:

1. Search "Download Weka". As of today, the URL is .
http://www.cs.waikato.ac.nz/ml/weka/downloading.html

2. Now, it'll have options to download the Weka. Here, based on your
- Machine configuration (i.e 32 bit or 64 bit)
- Java version and the corresponding Weka version

3. To check the Java version installed on your computer, open up **command prompt** and type **Java –version**
C:\>java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) Client VM (build 25.144-b01, mixed mode, sharing)
Note that we've java version 1.8

4.Check the operation system type (32bit or 64bit) from **system properties** and download the corresponding version of weka.
our systems are 32 bit , so download weka for 32 bit windows

5.Go to weka website then you find to links for 32 bit windows and jvm 1.8
Click **here** to download a self-extracting executable for 32-bit Windows that includes Oracle's 32-bit Java VM 1.8
(weka-3-8-3jre.exe; 113.4 MB)
Click **here** to download a self-extracting executable for 32-bit Windows without a Java VM
(weka-3-8-3.exe; 51 MB)

As you can see, the version of weka that we'll be installing requires Java 1.8 and we already have that – so we selected the option:
Click **here** to download a self-extracting executable for 32-bit Windows without a Java VM
(weka-3-8-3.exe; 51 MB

6. After downloading, install it. left all the options default.

7. After successful installation, launched weka by going to:

start > all programs >weka 3.8.3 >weka 3.8

c) Understand the features of WEKA toolkit such as Explorer, Knowledge Flow interface, Experimenter, command-line interface

WEKA GUI CHOOSER

Figure-1 Weka GUI Chooser screen

WEKA GUI CHOOSER :

When we open Weka, it will start the *Weka GUI Chooser* screen from where we can open the Weka application interface.There are totally **Five Application Interfaces** available for Weka.
The Weka GUI screen and the available application interfaces are seen in Figure 1.

Weka application interface:

- **Explorer** An environment for exploring data with WEKA
- **Experimenter** An environment for performing experiments and conducting statistical tests between learning schemes.
- **KnowledgeFlow** This environment supports essentially the same functions as the Explorer but with a drag-and-drop interface. One advantage is that it supports incremental learning.
- **Workbench** An all-in-one application that combines all the others within user-selectable "perspectives".
- **SimpleCLI** Provides a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface

Weka Menu Consists Of Four Sections:

1. Program

- Log Window Opens a log window that captures all that is printed to stdout or stderr. Useful for environments like MS Windows, where WEKA is normally not started from a terminal.
- Exit Closes WEKA.

2. Tools Other useful applications.

- Package manager A graphical interface to Weka's package management system.
- ArffViewerAn MDI application for viewing ARFF files in spread-sheet format.
- SqlViewer Represents an SQL worksheet, for querying databases via JDBC.
- Bayes net editor An application for editing, visualizing and learning Bayes nets.

3. Visualization Ways of visualizing data with WEKA.

- Plot For plotting a 2D plot of a dataset.
- ROC Displays a previously saved ROC curve.
- TreeVisualizerFor displaying directed graphs, e.g., a decision tree.
- GraphVisualizer Visualizes XML BIF or DOT format graphs, e.g.,for Bayesian networks.
- BoundaryVisualizer Allows the visualization of classifier decision boundaries in two dimensions.

4. Help Online resources for WEKA can be found here.

- Weka homepage Opens a browser window with WEKA's home-page.
- HOWTOs, code snippets, etc. The general WekaWiki [2], containing lots of examples and HOWTOs around the development and use of WEKA.
- Weka on SourceforgeWEKA's project homepage on Sourceforge.net.
- SystemInfoLists some internals about the Java/WEKA environment, e.g., the CLASSPATH.

d)Navigate the options available in the WEKA (ex. Select attributes panel, Preprocess panel, Classify panel, Cluster panel, Associate panel and Visualize panel)

WEKA EXPLORER

Weka Explorer:It is a user interface which contains a group of tabs just below the title bar. The tabs are as follows:

1. Preprocess
2. Classify
3. Cluster
4. Associate
5. Select Attributes
6. Visualize

The bottom of the window contains status box, log and WEKA bird

The Weka Explorer is illustrated in Figure

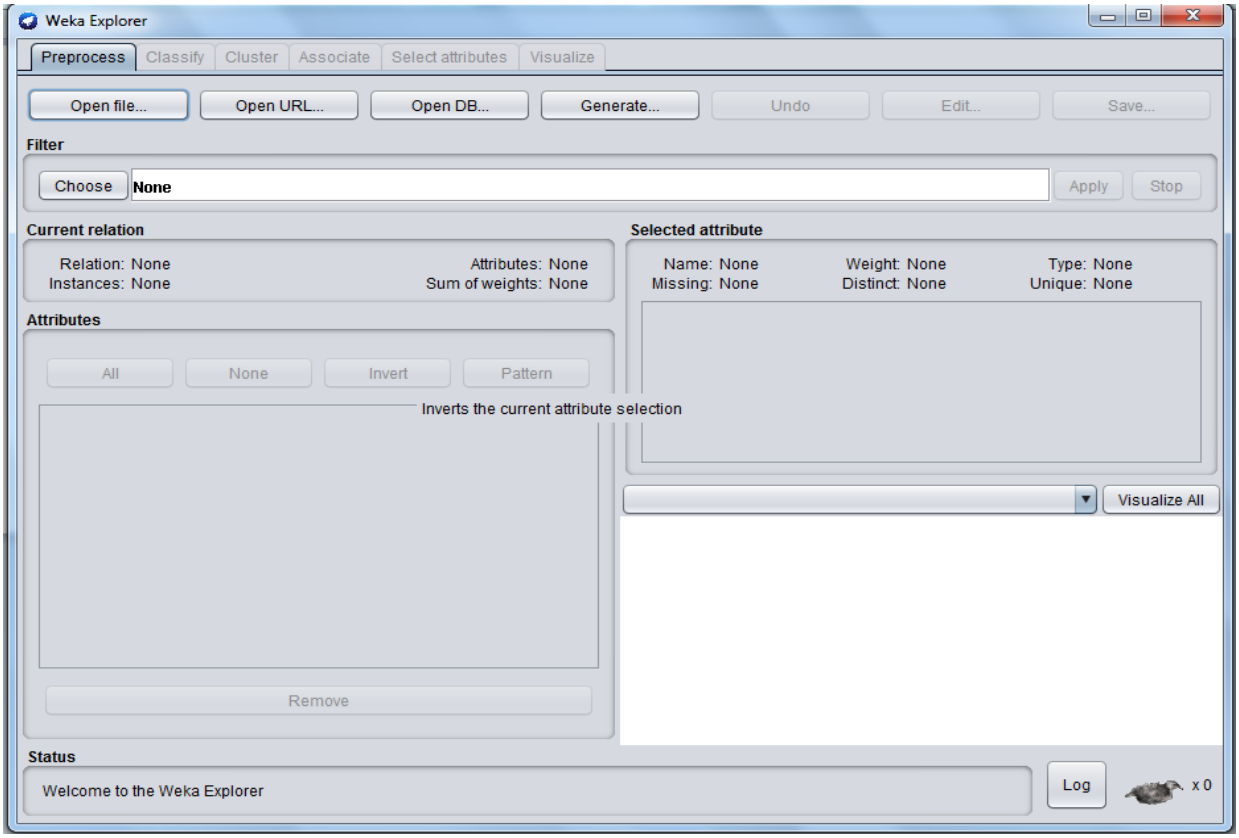
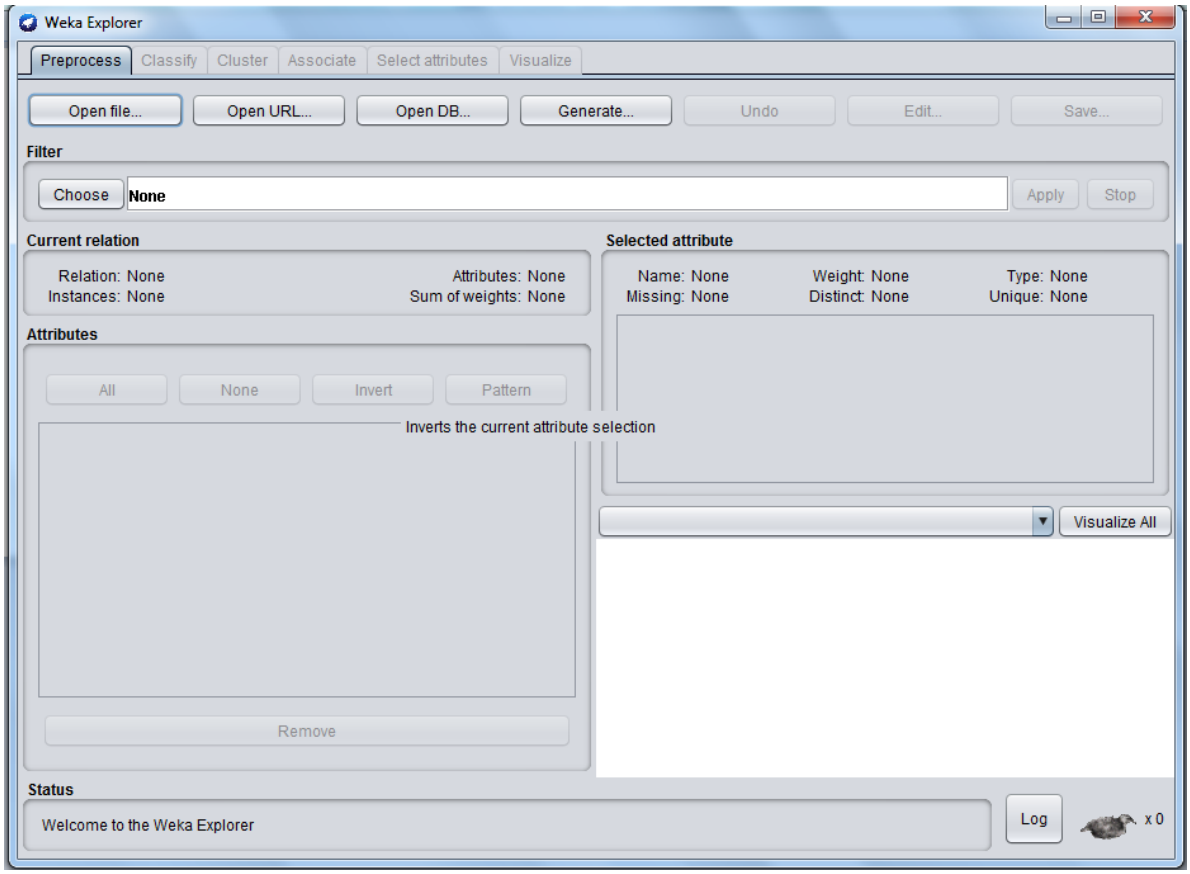


Figure 2: Weka Explorer Screen

1. PREPROCESSING:



LOADING DATA

The first four buttons at the top of the preprocess section enable you to load

Data into WEKA:

1. **Open file**: It shows a dialog box allowing you to browse for the data file on the local file system.
2. **Open URL**: Asks for a Uniform Resource Locator address for where the data is stored.
3. **Open DB**: Reads data from a database.
4. **Generate**: It is used to generate artificial data from a variety of Data Generators.

Using the Open file button we can read files in a variety of formats like WEKA's ARFF format, CSV format. Typically ARFF files have .arff extension and CSV files .csv extension.

THE CURRENT RELATION

The Current relation box contains the currently loaded data i.e. interpreted as a single relational table in database terminology, which has three entries:

1. **Relation**: It provides the name of the relation in the file from which it was loaded. Filters are used modify the name of a relation.
2. **Instances**: The number of instances (data points/records) in the data.
3. **Attributes**: The number of attributes (features) in the data.

ATTRIBUTES

It is located below the current relation box which contains four buttons, they are:

1) **All** is used to tick all boxes 2) **None** is used to clear all boxes 3) **Invert** is used make ticked boxes unticked. 4) **Pattern** is used to select attributes by representing an expression. E.g. a.* is used to select all the attributes that begins with a.

SELECTED ATTRIBUTE:

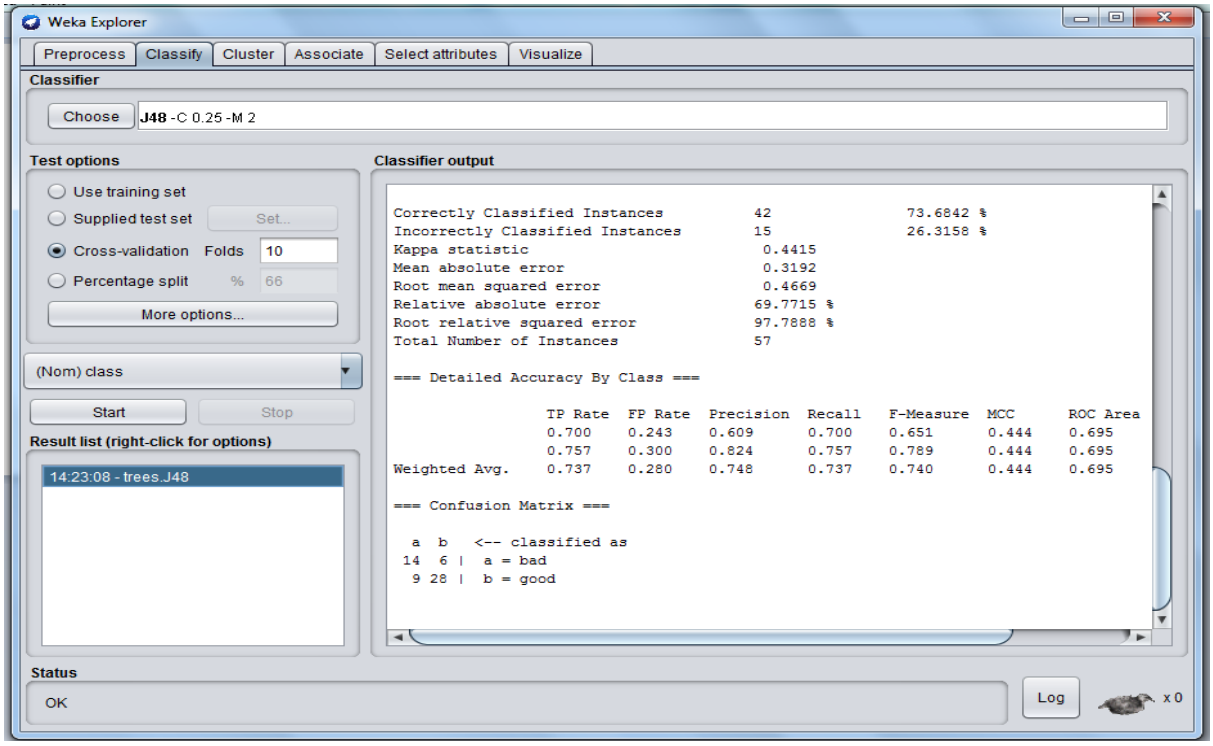
It is located beside the current relation box which contains the following:

1. **Name:** It specifies the name of the attribute i.e. same as in the attribute list.
2. **Type**: It specifies the type of attribute, most commonly Nominal or Numeric.
3. **Missing:** It provides a numeric value of instances in the data for which an attribute is missing.
4. **Distinct**: It provides the number of different values that the data contains for an attribute.
5. **Unique**: it provides the number of instances in the data having a value for an attribute that no other instances have.

FILTERS

By clicking the **Choose** button at the left of the Filter box, it is possible to select one of the filters in WEKA. Once a filter has been selected, its name and options are shown in the field next to the Choose button, by clicking on this box with the left mouse button it shows a Generic Object Editor dialog box which is used to configure the filter

2. CLASSIFICATION



Classification has a text box which gives the name of currently selected classifier, and its options. By clicking it with the left mouse button it shows a GenericObjectEditor dialog box, which is same as for filters i.e. used to configure the current classifier options.

TEST OPTIONS

The result of applying the chosen classifier will be tested according to the options that are set by clicking in the Test options box. There are four test modes:

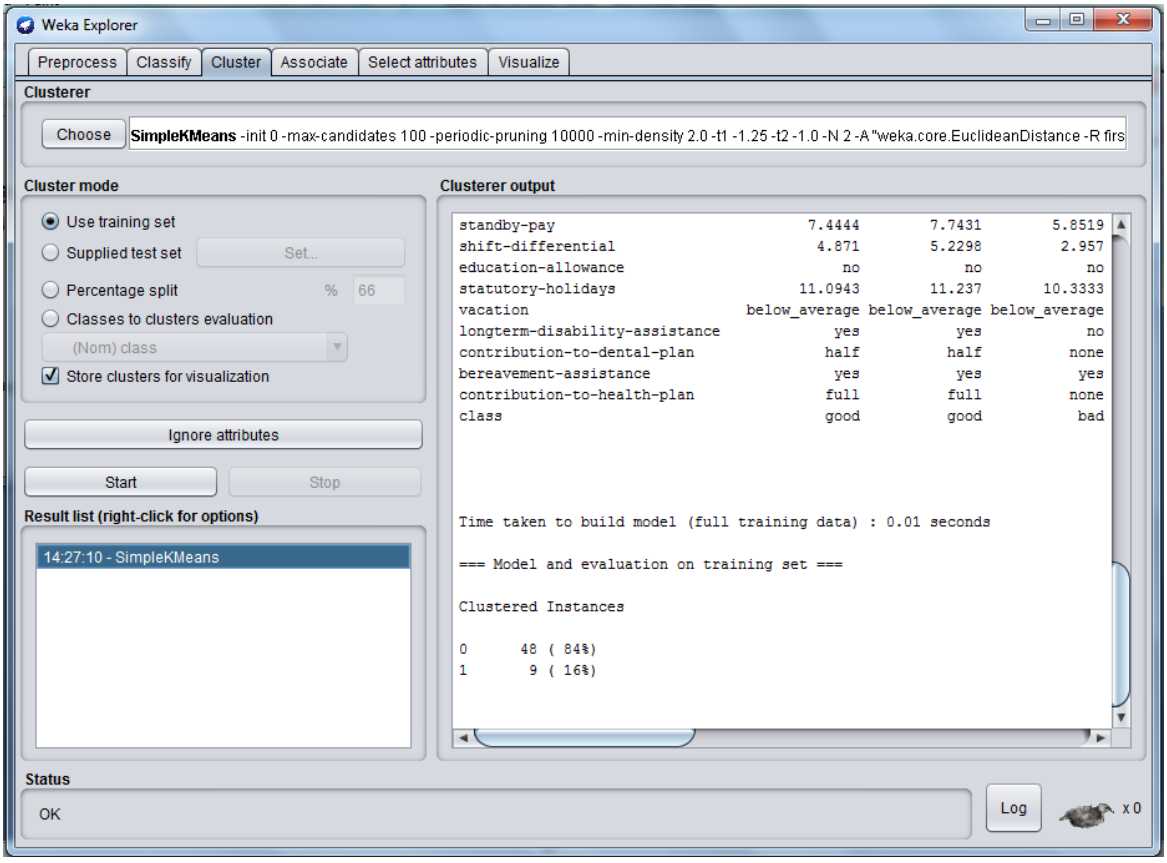
1. Use training set.
2. Supplied test set.
3. Cross-validation.
4. Percentage split.

Once the classifier, test options and class have all been set, the learning process is started by clicking on the **Start** button. We can stop the training process at any time by clicking on the **Stop** button.

The **Classifier output** area to the right of the display is filled with text describing the results of training and testing.

After training several classifiers, the **Result List** will contain several entries using which we can move over various results that have been generated. By pressing Delete we can remove a selected entry from the results.

3. CLUSTERING



By clicking the text box beside the choose button in the **Clusterer box**, it shows a dialog box used to choose a new clustering scheme.

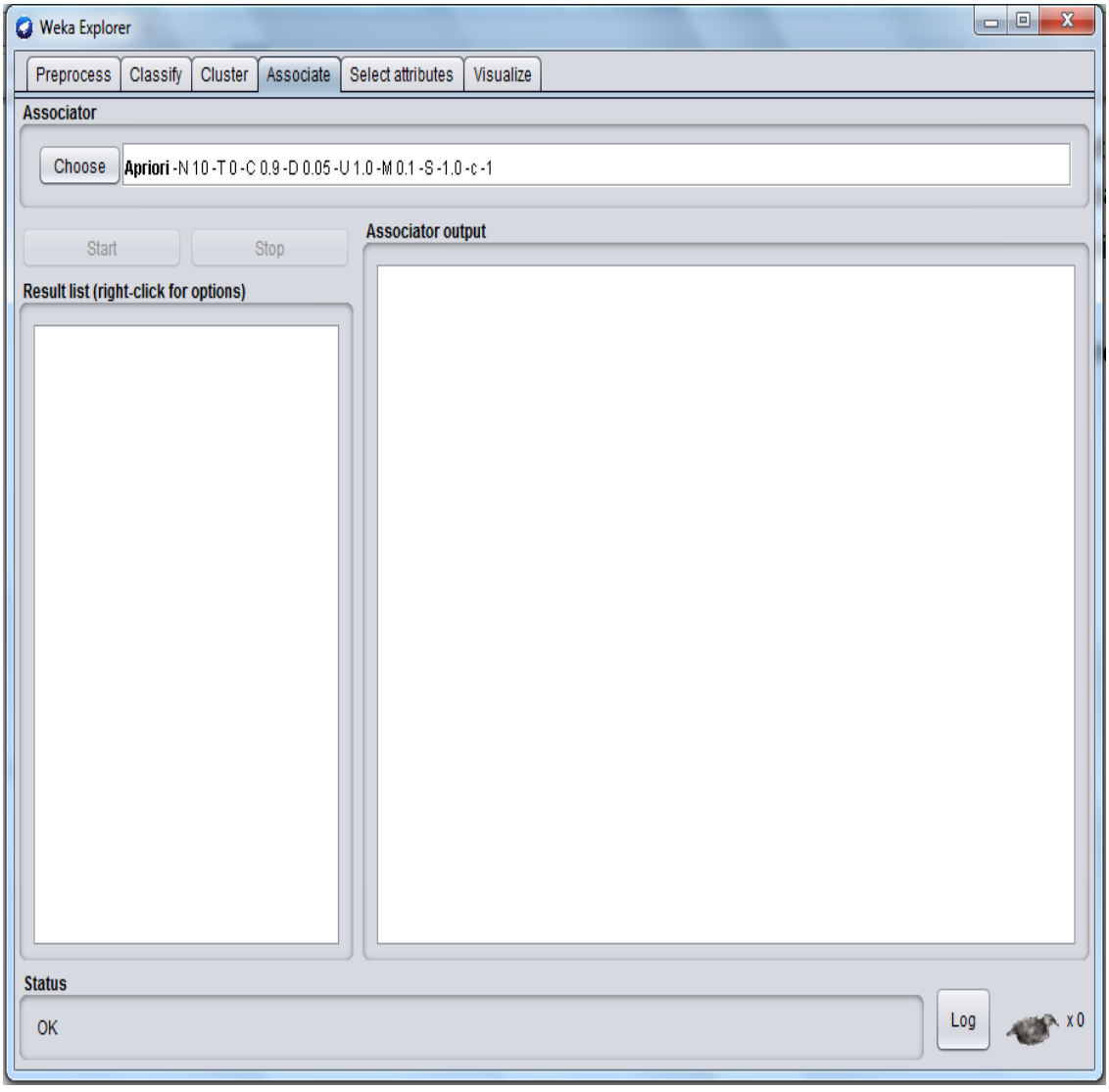
The **Cluster mode** box is used to choose what to cluster and how to evaluate the results. The first three options in it are same as in classification like **Use training set, Supplied test set** and **Percentage split.**

The fourth option is **classes to clusters evaluation**

An additional option in the Cluster mode box is the **Store clusters for visualization** which finds whether or not it will be possible to visualize the clusters once training is complete.

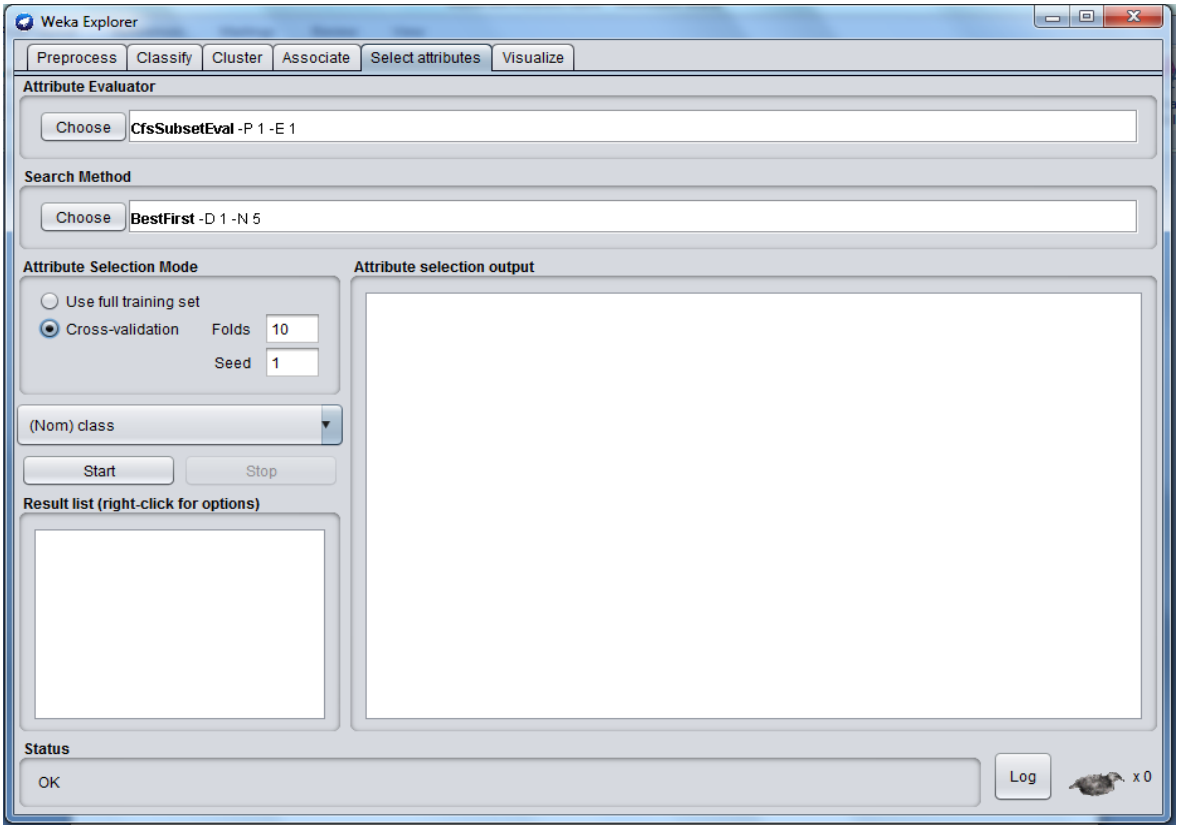
Ignore Attributes: when clustering, some attributes in the data should be ignored. It shows a small window that allows you to select which attributes are ignored.

4. ASSOCIATING



It contains schemes for learning association rules, and the learners are chosen and configured in the same way as the cluster, filters, and classifiers in the other panels.

5. SELECTING ATTRIBUTES



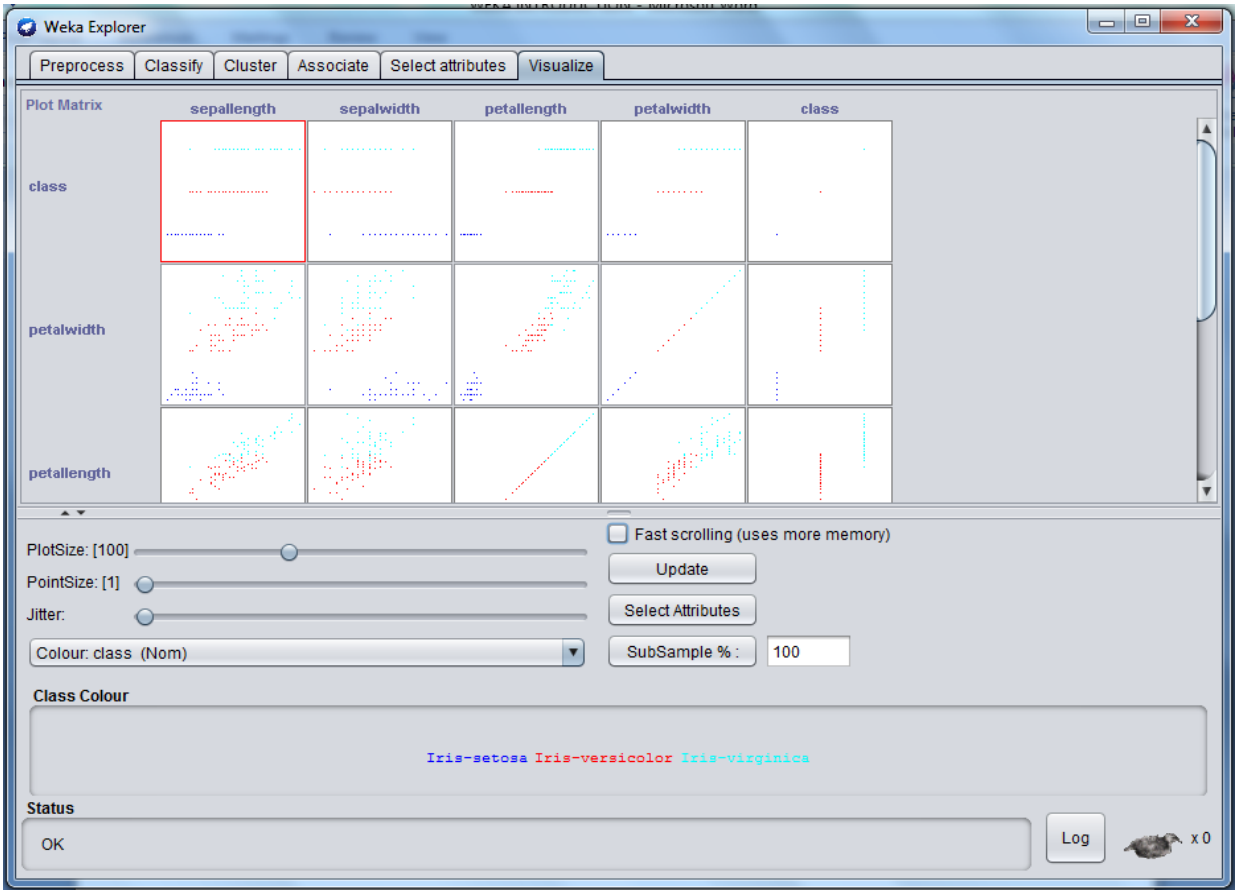
Attribute selection involves searching through all possible combinations of attributes in the data to find which subset of attributes works best for prediction. To do this, two objects must be set up: an attribute evaluator and a search method. The evaluator determines what method is used to assign a worth to each subset of attributes. The search method determines what style of search is performed.

The **Attribute Selection Mode** box has two options:

1. **Use full training set**: The worth of the attribute subset is determined using the full set of training data.
2. **Cross-validation**: The worth of the attribute subset is determined by a process of cross-validation. The Fold and Seed fields set the number of folds to use and the random seed used when shuffling the data.

6. VISUALIZING

WEKA's visualization section allows you to visualize 2D plots of the current relation.



The *Visualize* panel helps you visualize a dataset—not the result of a classification or clustering model, but the dataset itself.

It displays a matrix of two-dimensional scatter plots of every pair of attributes.

You can select an attribute—normally the class—for coloring the data points using the controls at the bottom. If it is nominal, the coloring is discrete; if it is numeric, the color spectrum ranges continuously from blue (low values) to orange (high values). Data points with no class value are shown in black. You can change the size of each plot, the size of the points, and the amount of jitter, which is a random displacement applied to X and Y values to separate points that lie on top of one another.

e) Study the arff file format Explore the available data sets in WEKA.

ARFF (Attribute-Relation File Format):

- An **ARFF** file is an ASCII text file that describes a list of instances sharing a set of attributes.
- ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information.
- The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types.
- An example header on the standard IRIS dataset looks like this:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
% (a) Creator: R.A. Fisher
% (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
% (c) Date: July, 1988
%
@RELATION iris
@ATTRIBUTE sepallength NUMERIC
@ATTRIBUTE sepalwidth NUMERIC
@ATTRIBUTE petallength NUMERIC
@ATTRIBUTE petalwidth NUMERIC
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The Data of the ARFF file looks like the following:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
```

- Lines that begin with a **%** are comments. The **@RELATION**, **@ATTRIBUTE** and **@DATA** declarations are case insensitive.

The ARFF Header Section

- The ARFF Header section of the file contains the **relation** declaration and **attribute** declarations.

The @relation Declaration

- The relation name is defined as the first line in the ARFF file. The format is:

  @relation <relation-name>

- where<relation-name> is a string. The string must be quoted if the name includes spaces. Furthermore, relation names or attribute names (see below) cannot begin with
  - a character below \\u0021
  - '{', '}', ',', or '%'

- Moreover, it can only begin with a single or double quote if there is a corresponding quote at the end of the name.

The @attribute Declarations

- Attribute declarations take the form of an ordered sequence of @attribute statements.
- Each attribute in the data set has its own @attribute statement which uniquely defines the name of that attribute and it's data type.
- The order the attributes are declared indicates the column position in the data sectionof the file.
- For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found in the third comma delimited column.
- The format for the @attribute statement is:

  @attribute <attribute-name><datatype>
- where the <attribute-name> must adhere to the constraints specified in the above section on the @relation declaration.
- The <datatype> can be any of the four types supported by Weka:
  - numeric
  - integer is treated as numeric
  - real is treated as numeric
  - <nominal-specification>
  - string
  - date [<date-format>]
- where<nominal-specification> and <date-format> are defined below. The keywords numeric, real, integer, string and date are case insensitive.

Numeric attributes

- Numeric attributes can be real or integer numbers.

Nominal attributes

- Nominal values are defined by providing an <nominal-specification> listing the possible values: <nominal-name1>, <nominal-name2>, <nominal-name3>,...
- For example, the class value of the Iris dataset can be defined as follows:

  @ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
- Values that contain spaces must be quoted.

String attributes

- String attributes allow us to create attributes containing arbitrary textual values.
- This is very useful in text-mining applications, as we can create datasets with string attributes, then writeWeka Filters to manipulate strings (likeStringToWordVectorFilter).
- String attributes are declared as follows:

  @ATTRIBUTE LCC string

Date attributes

- Date attribute declarations take the form:

  @attribute <name> date [<date-format>]
- where<name> is the name for the attribute and <date-format> is an optional string specifying how date values should be parsed and printed (this is the same format used by SimpleDateFormat). The default format string accepts the ISO-8601 combined date and time format**: yyyy-MM-dd'T'HH:mm:ss.**

- Dates must be specified in the data section as the corresponding string representations of the date/time

The ARFF Data Section

- The ARFF Data section of the file contains the data declaration line and the actual instance lines.

The @data Declaration

- The @data declaration is a single line denoting the start of the data segment in the file. The format is:

@data

The instance data

- Each instance is represented on a single line, with carriage returns denoting the end of the instance.
- A percent sign (%) introduces a comment, which continues to the end of the line.
- Attribute values for each instance are delimited by commas.
- They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth @attribute declaration is always the nth field of the attribute).
- Missing values are represented by a single question mark, as in:

```
@data
4.4,?,1.5,?,Iris-setosa
```

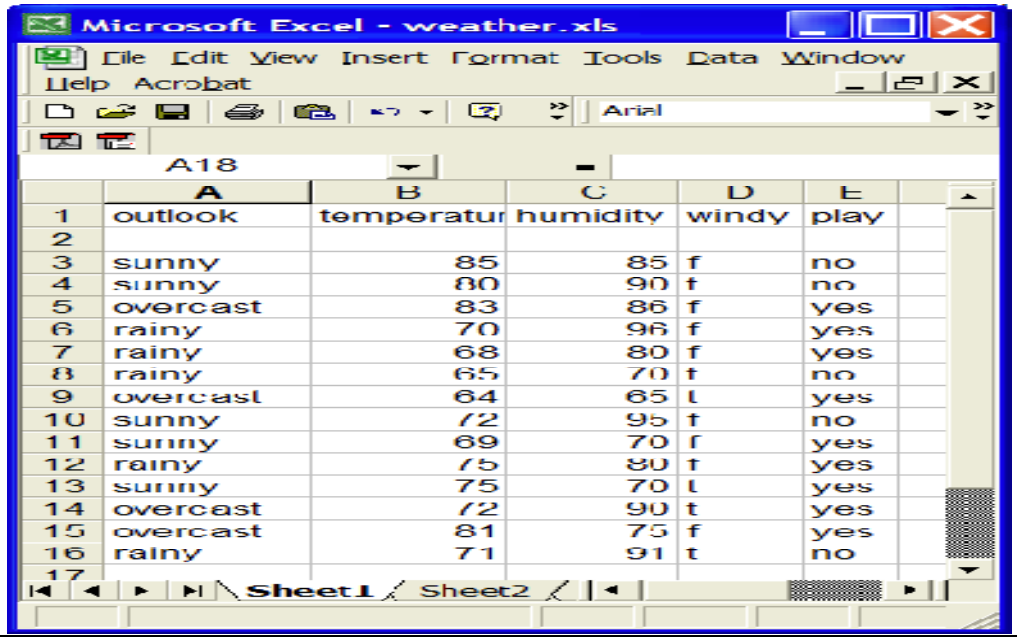
- Values of string and nominal attributes are case sensitive, and any that contain space or the comment-delimiter character % must be quoted. (The code suggests that double-quotes are acceptable and that a backslash will escape individual characters.)

CREATING AN *ARFF* FILE

- The easiest and the most common way of getting the data into WEKA is to store it as Attribute-Relation File Format (ARFF) file.
- We can create ARFF file manual from excel sheet(without using weka)
- We can create ARFF file with using weka

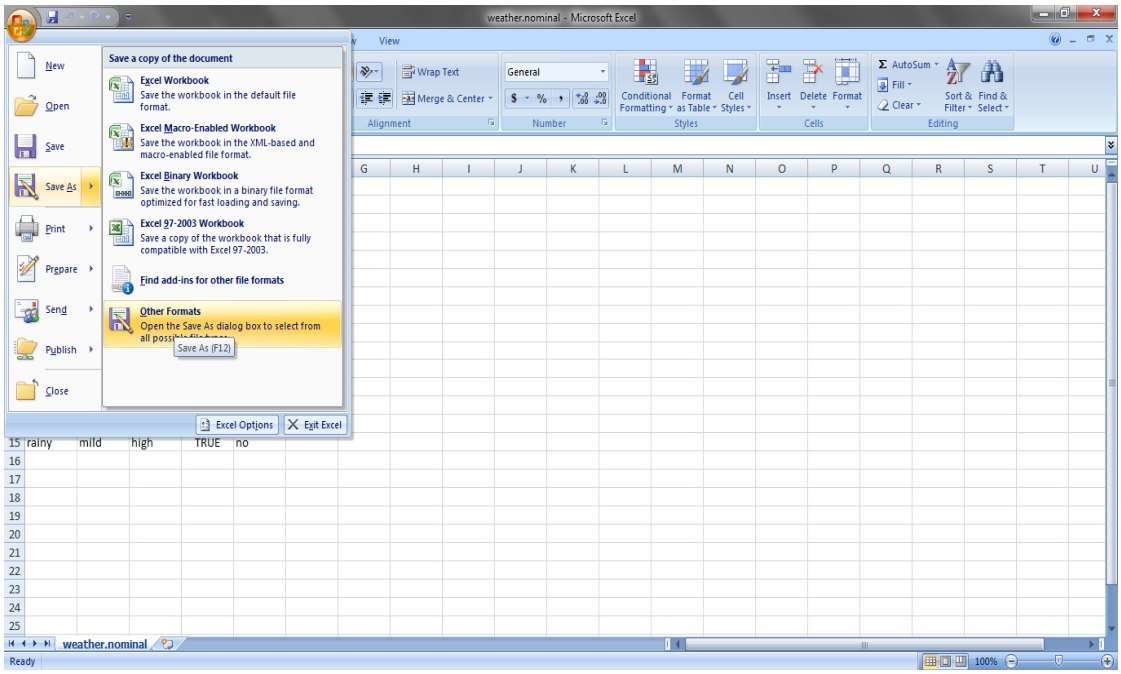
Creating ARFF File Without using Weka:

- We assume that all our data stored in a Microsoft Excel spreadsheet “weather.xls”.

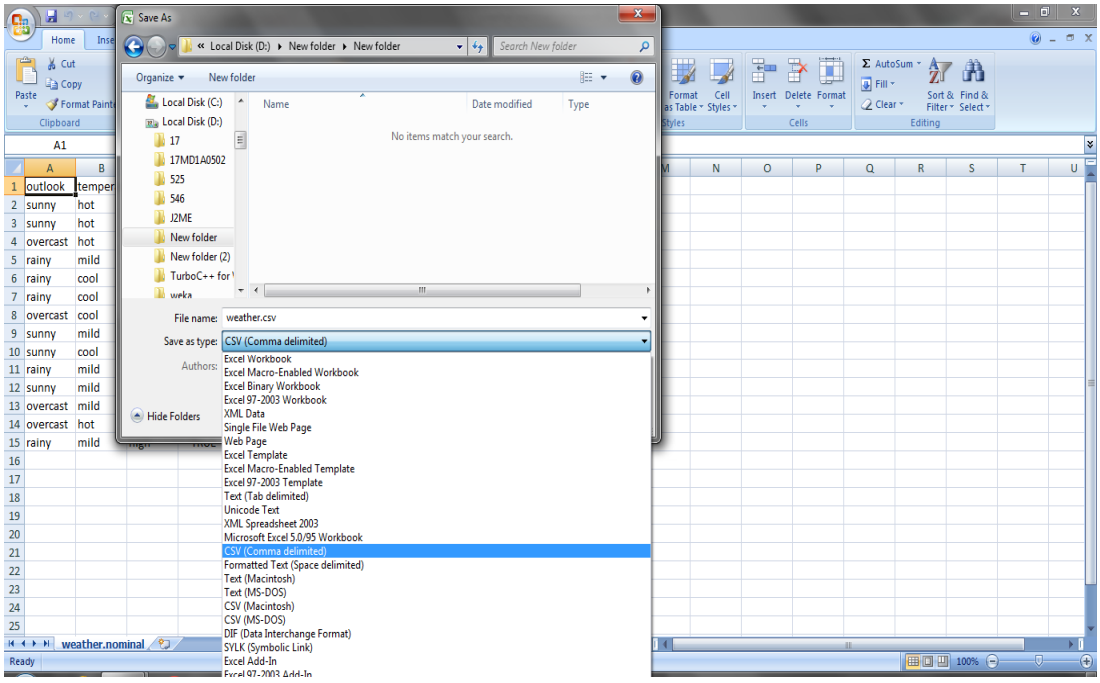


	A	B	C	D	E
1	outlook	temperatur	humidity	windy	play
2					
3	sunny	85	85	f	no
4	sunny	80	90	t	no
5	overcast	83	86	f	yes
6	rainy	70	96	f	yes
7	rainy	68	80	f	yes
8	rainy	65	70	t	no
9	overcast	64	65	t	yes
10	sunny	72	95	f	no
11	sunny	69	70	f	yes
12	rainy	75	80	f	yes
13	sunny	75	70	t	yes
14	overcast	72	90	t	yes
15	overcast	81	75	f	yes
16	rainy	71	91	t	no
17					

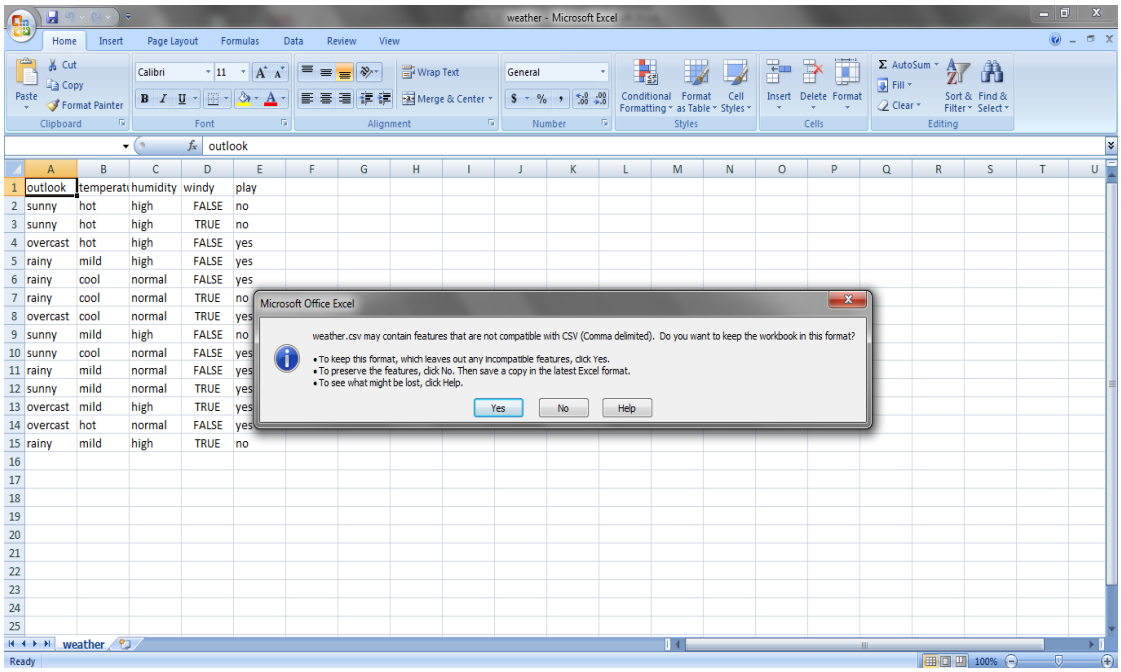
- We have to convert our data into **comma-separated file (.csv)**
- To save our data in comma-separated format, select the **'Save As**…' menu item from Excel '**File**' pull-down menu.



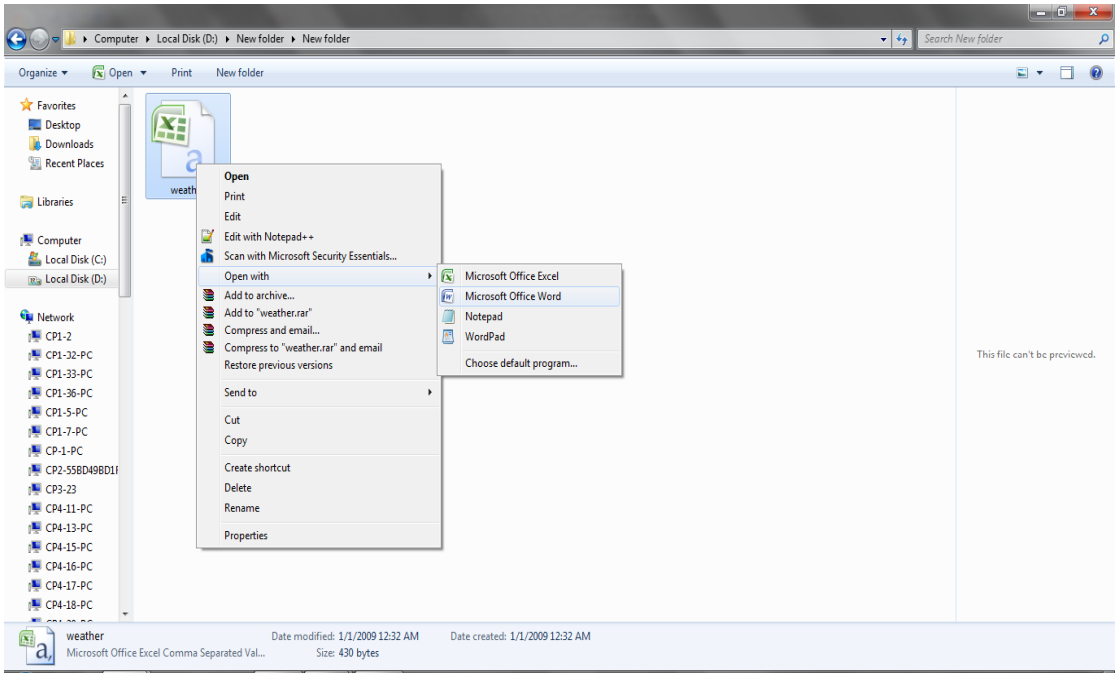
- In the ensuing dialog box select **'CSV (Comma Delimited)'**from the file type pop-up menu, enter a **name** of the file, and click **'Save'** button.



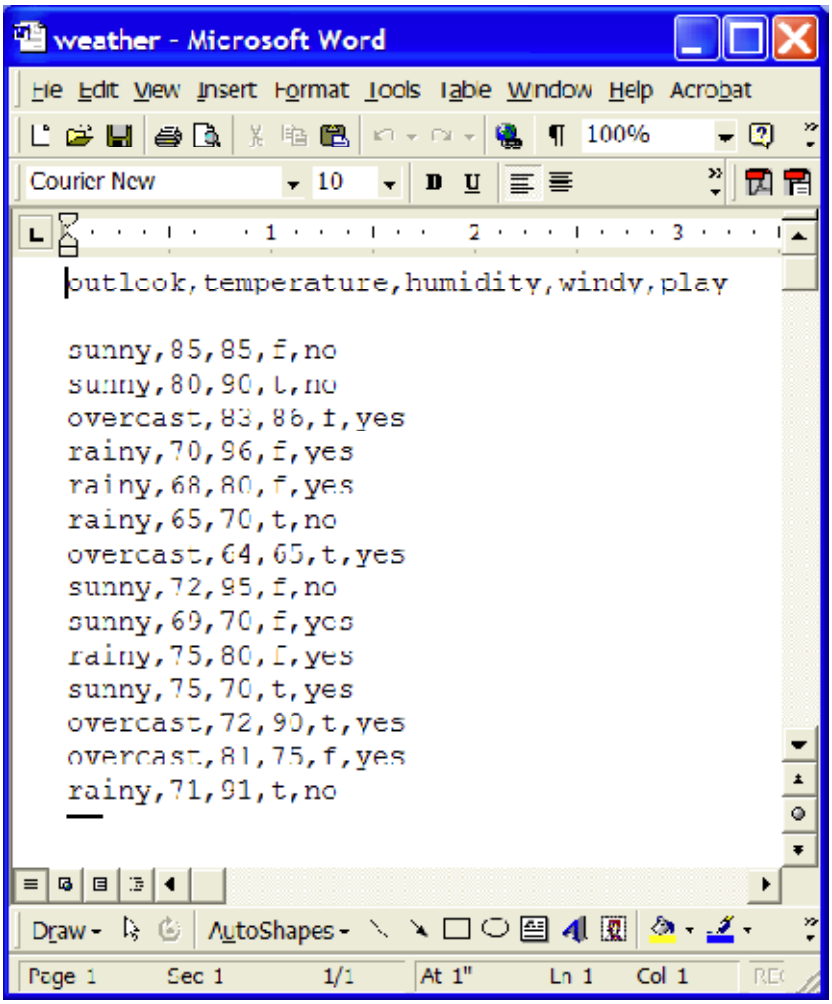
❖ Ignore all messages that appear by clicking **'OK' or 'Yes'**



❖ Open this file with MicrosoftWord.



- Your screen will look like the screen below.

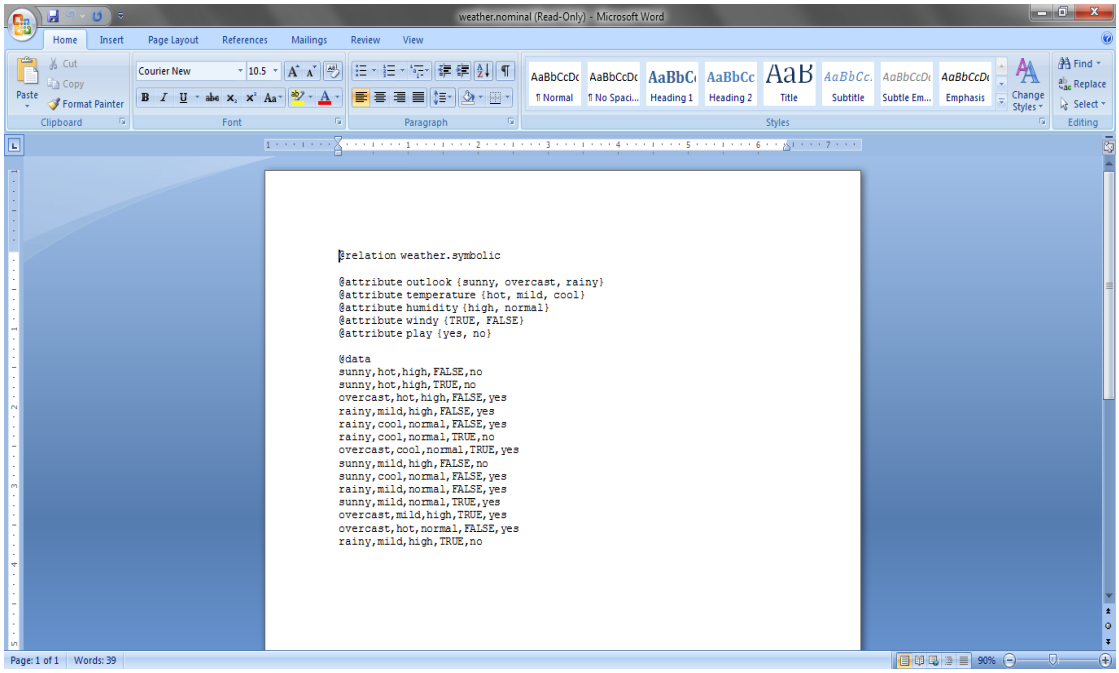


CSV

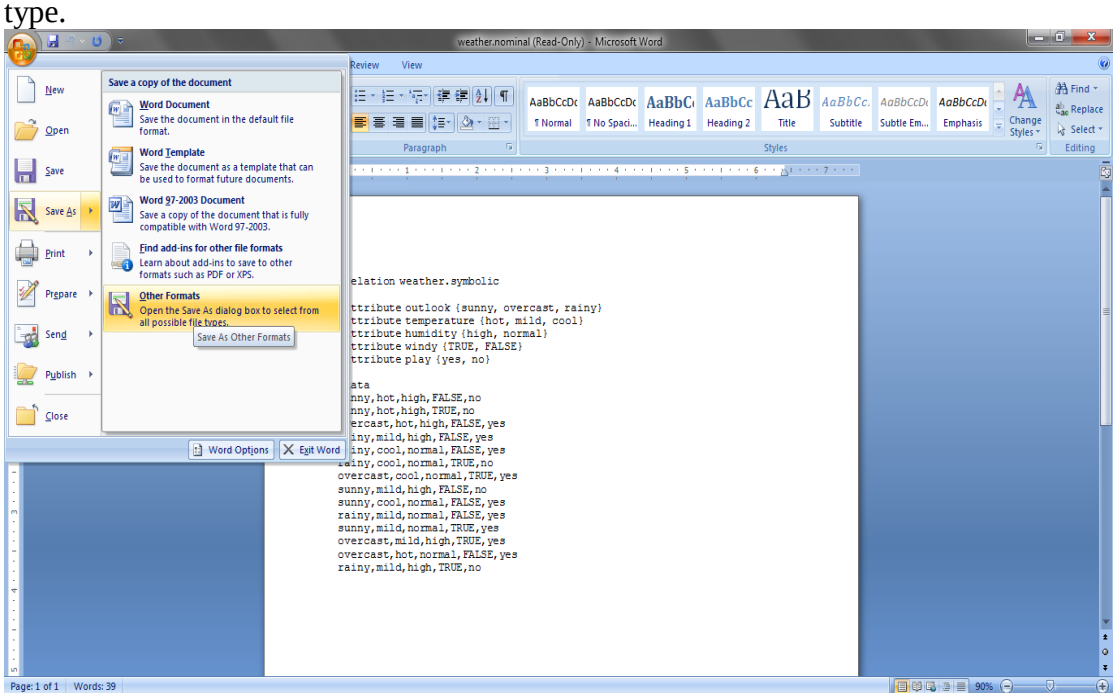
- The rows of the original spreadsheet are converted into lines of text where the elements are separated from each other by commas.

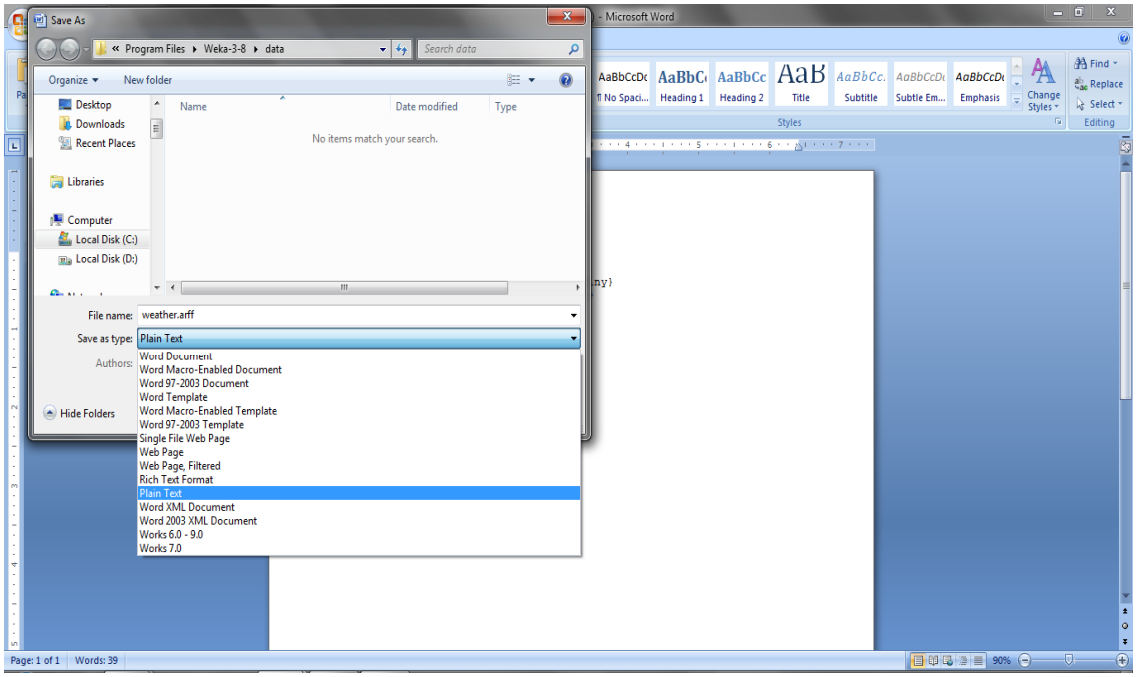
- In this file we need to change the first line, which holds the attribute names, into the header structure that makes up the beginning of an ARFF file.

- Adda **@relation tag** with the **dataset's name**, an **@attribute tag** with the **attribute information**, and a **@data tag** as shown below.

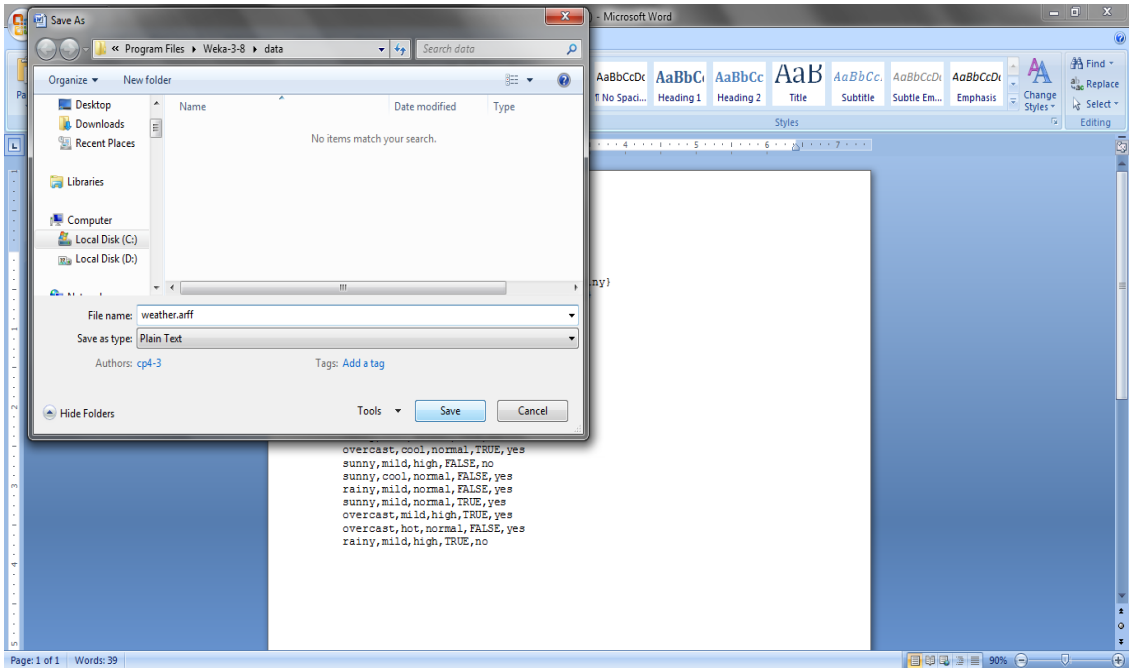


❖ Choose **'Save As…'** from the **'File'** menu and specify **'Text Only with Line Breaks'** as the file type.





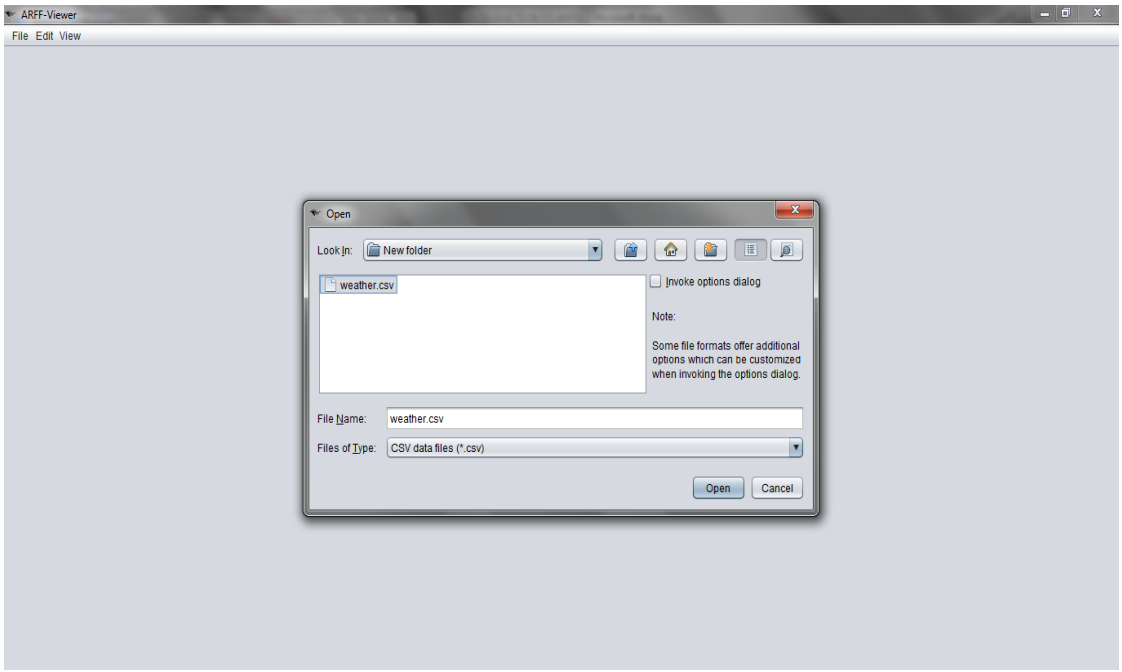
❖ Enter a **file name** and click **'Save'** button. Rename the file to the file with extension **.arff** to indicate that it is in ARFF format



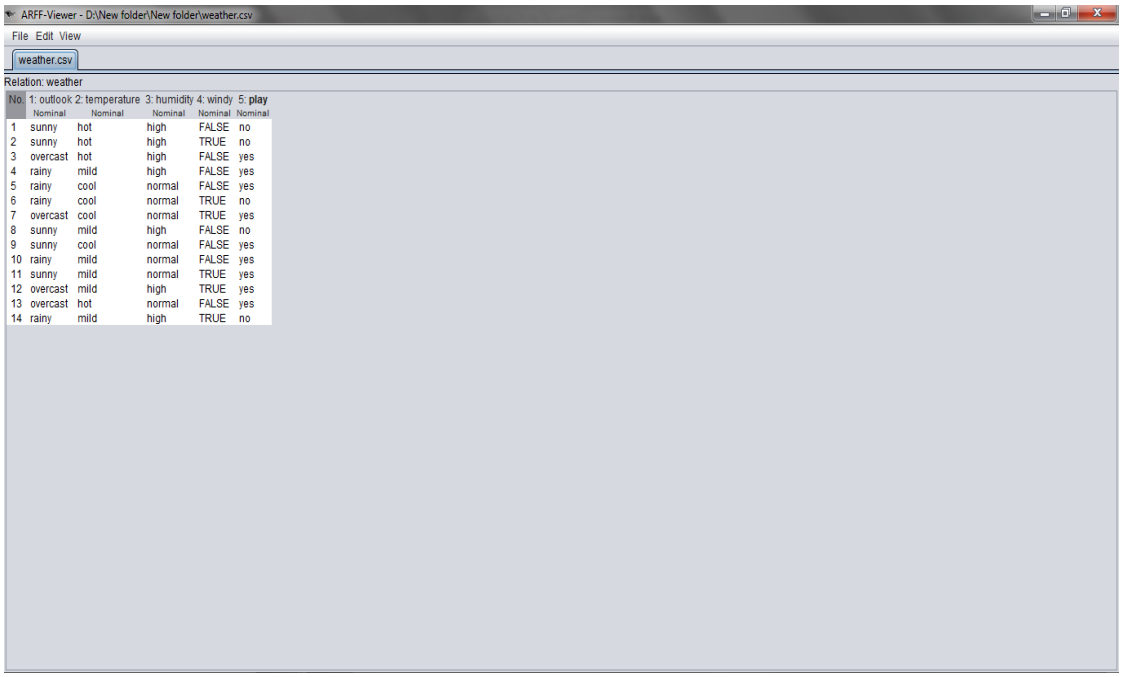
Creating ARFF File using Weka:

- We can create .arff file by using Weka tool also.
- To create .arff file , First we have to convert our **excel file** to **csv file** (use above method)
- After converting excel file to csv file
- Open created csv file by using Weka tool in the below process
- Step 1: start weka tool
- Step 2: Go to **tools** in toolbar and select **Arffviewer**

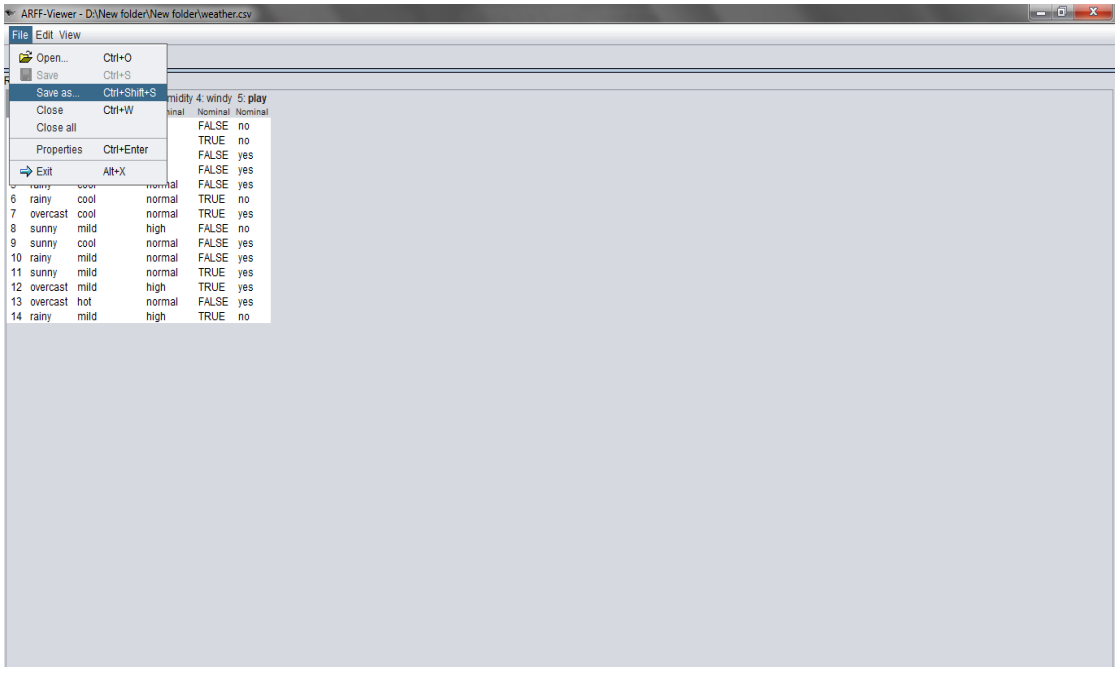
- Step 3: Then we get a window, in that window go to **file** ---> **open** and select our csv file and click **open**



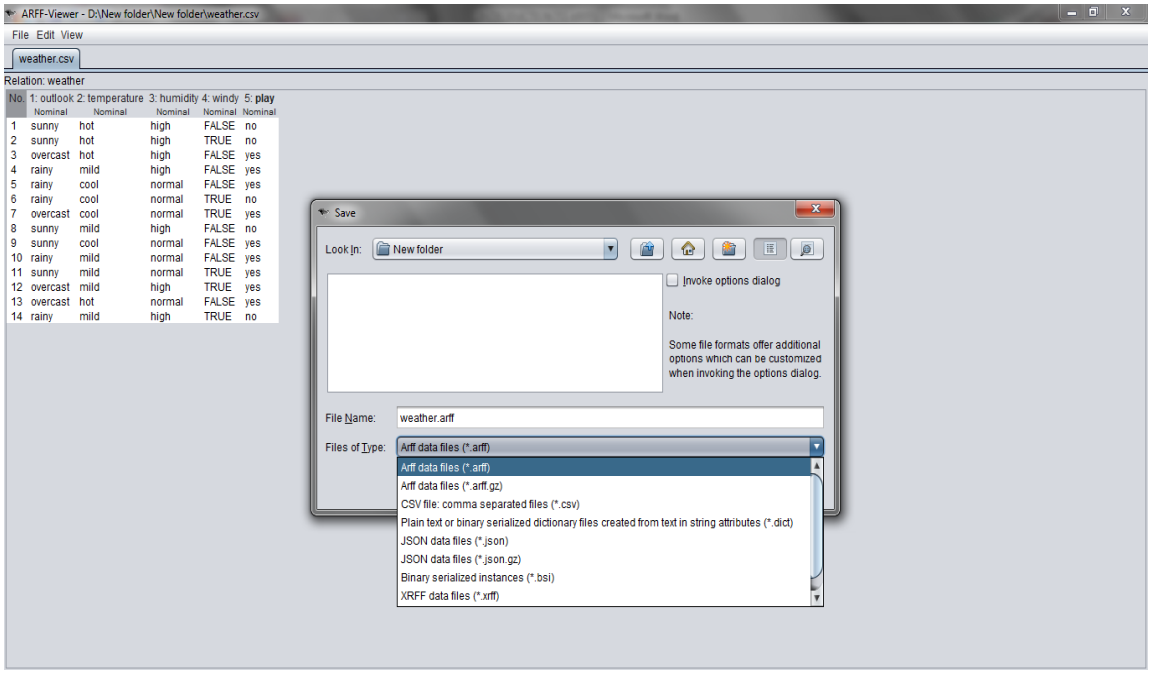
❖ Step 4:Then our screen look like this



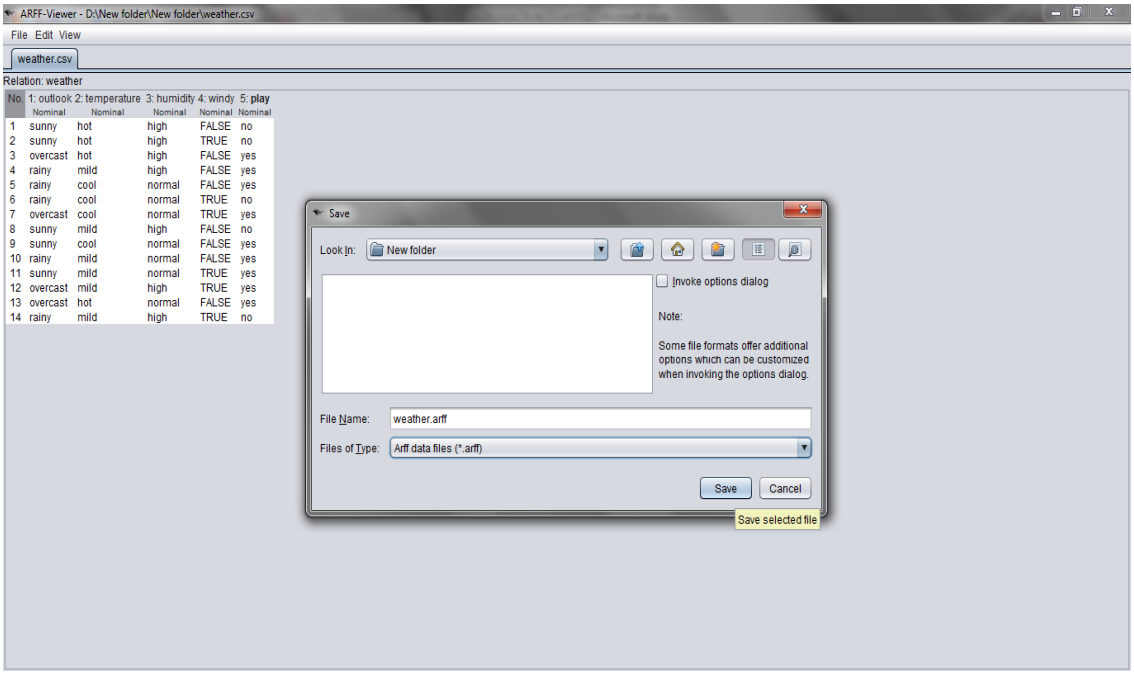
❖ Step 5:Now again go to **file ---->save as**



❖ Step 6:Then select **file type** as "**Arff data files**" and type the **file name** with **.arff** extension

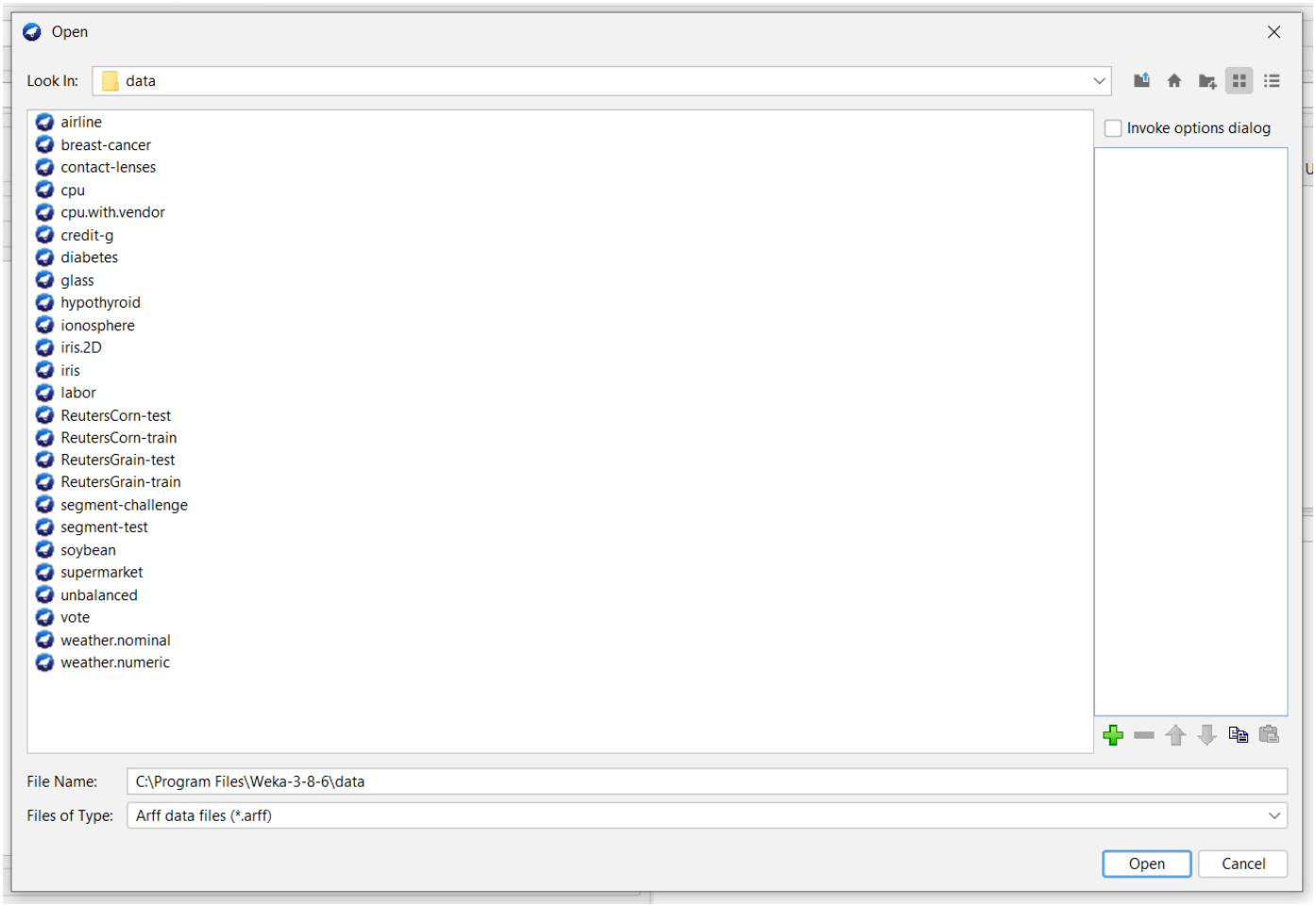


❖ Step 7:Then click **save** button to save our csv file as arff file.



Explore the available data sets in WEKA

There are 25 different datasets are available in weka (C:\Program Files\Weka-3-8\) by default for testing purpose. All the datasets are available in. arff format. Those datasets are listed below.

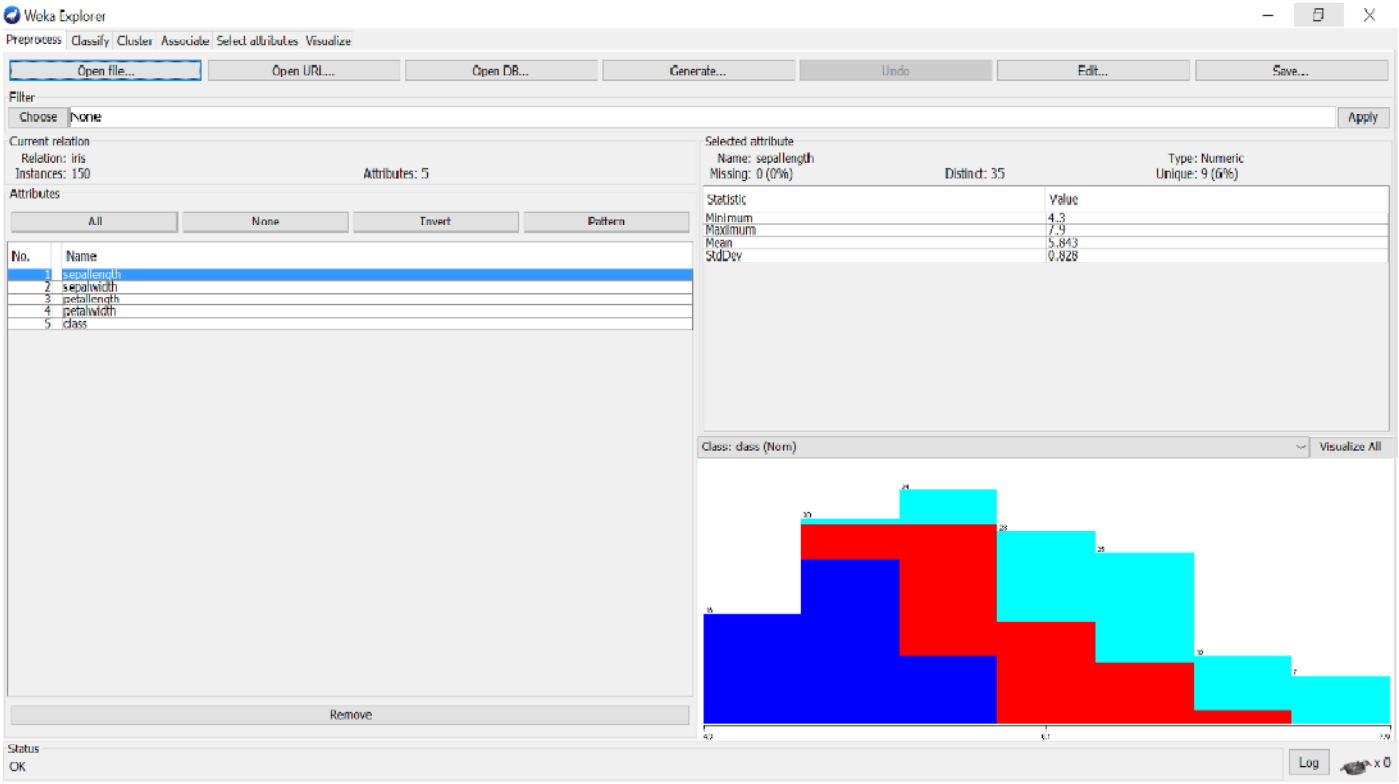


f) Load each dataset and observe the following:

1. **List the attribute names and they types**
2. **Number of records in each dataset**
3. **Identify the class attribute (if any)**
4. **Plot Histogram**
5. **Determine the number of records for each class.**
6. **Visualize the data in various dimensions**

Loading dataset Procedure:

1. Open the weka tool and select the explorer option.
2. New window will be opened which consists of different options (Preprocess, Association etc.)
3. In the preprocess, click the ―open file‖ option.
4. Go to C:\Program Files\Weka-3-8\data for finding different existing. arff datasets.
5. Click on any dataset for loading the data then the data will be displayed as shown below.



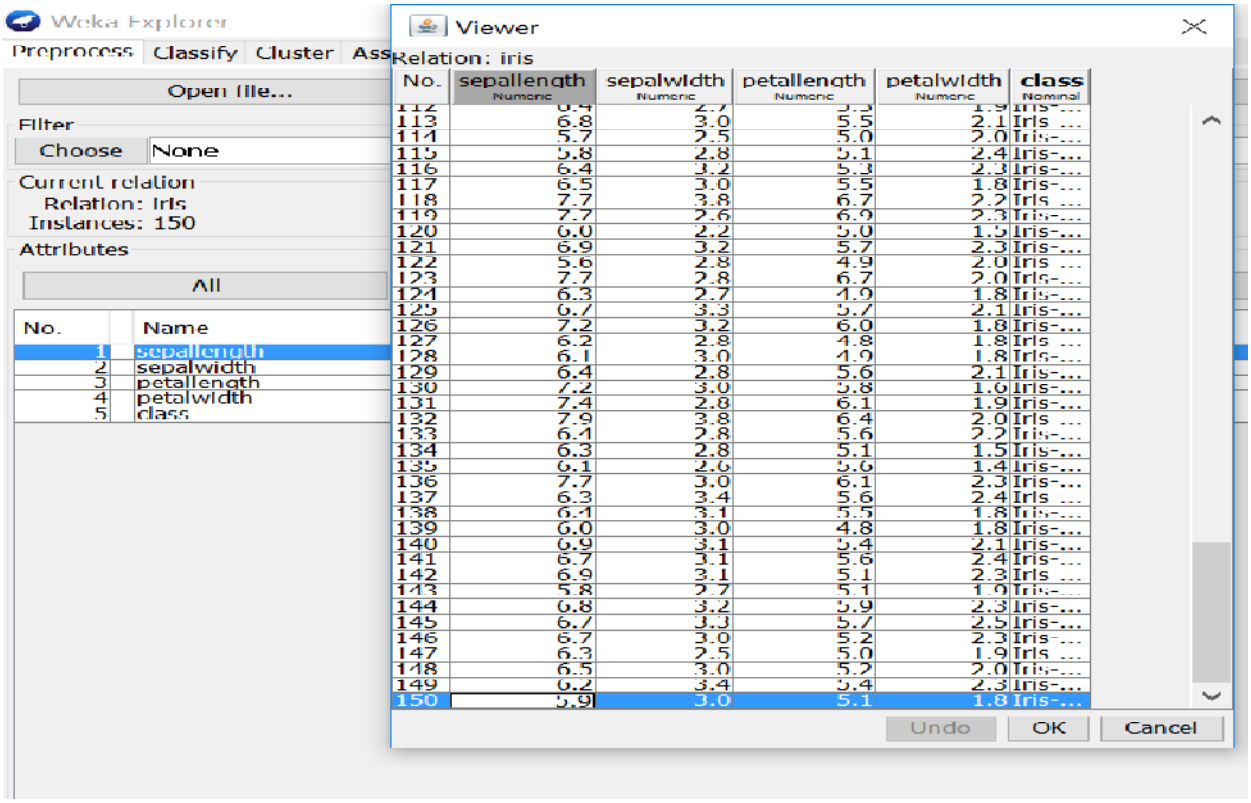
Here we have taken **IRIS.arff** dataset as sample for observing all the below things.

i. List the attribute names and they types

There are 5 attributes& its datatype present in the above loaded dataset (IRIS.arff)
sepallength – Numeric
sepalwidth – Numeric
petallength – Numeric
petallength – Numeric
Class – Nominal

ii. Number of records in each dataset

There are total 150 records (Instances) in dataset (IRIS.arff).

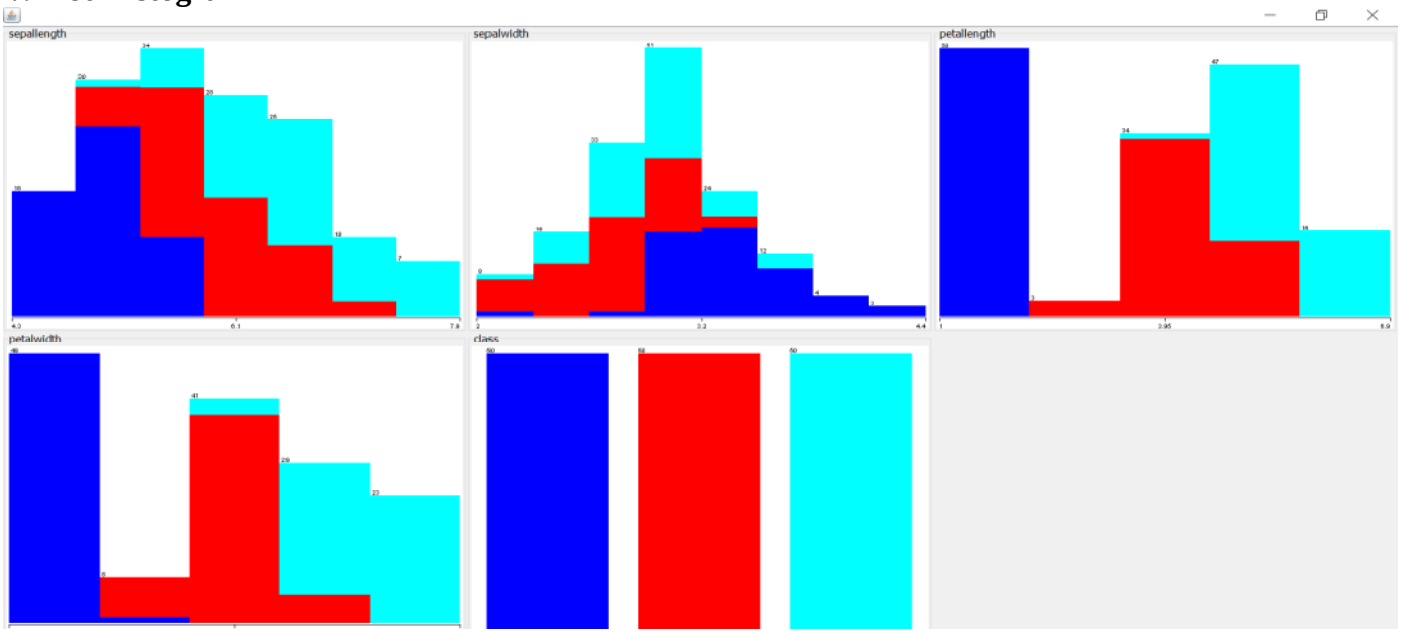


iii. Identify the class attribute (if any)

There is one class attribute which consists of 3 labels. They are:

1. Iris-setosa
2. Iris-versicolor
3. Iris-virginica

iv. Plot Histogram

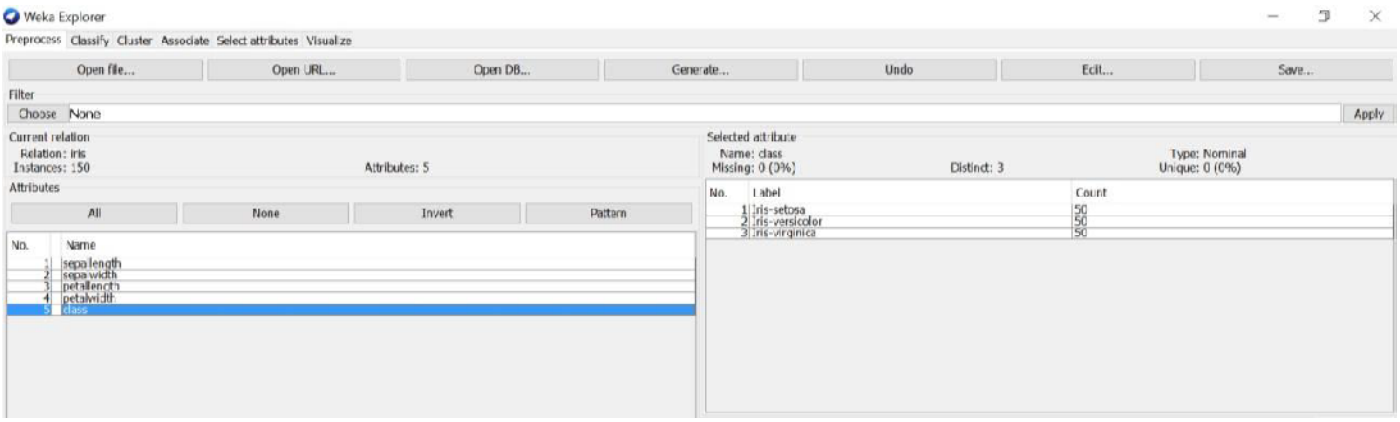


v. Determine the number of records for each class.

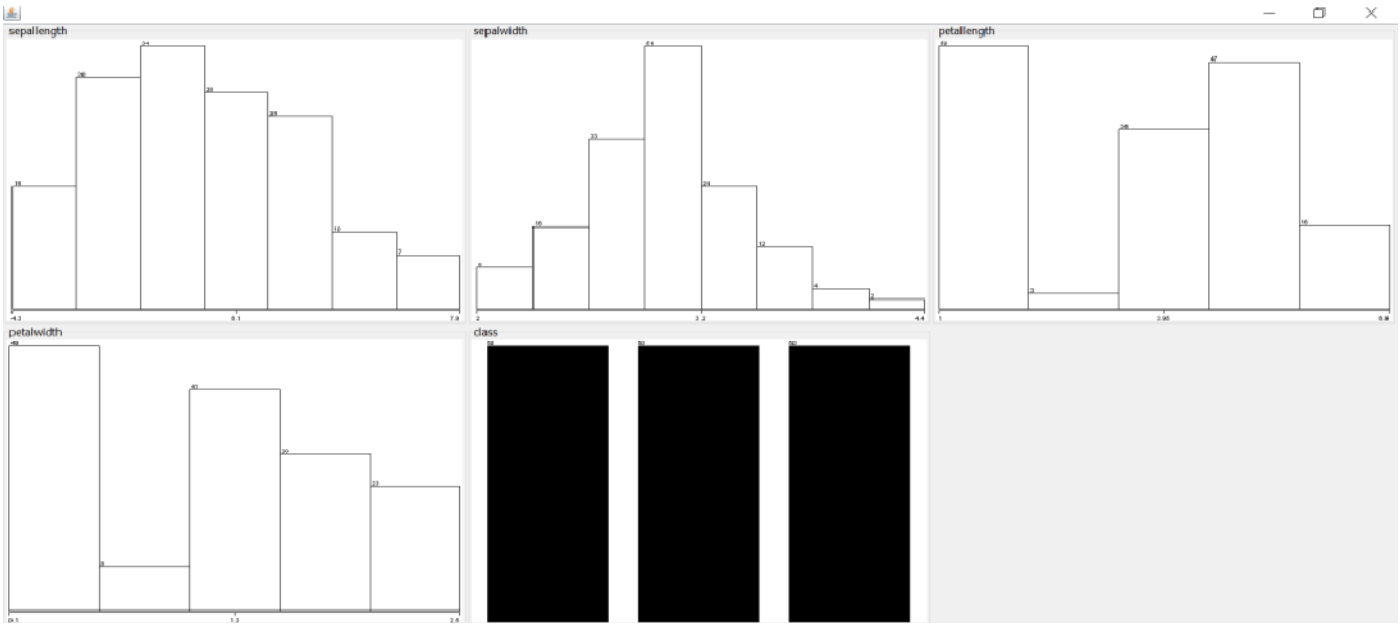
There is one class attribute (150 records) which consists of 3 labels. They are shown below

1. Iris-setosa - 50 records
2. Iris-versicolor – 50 records

3. Iris-virginica – 50 records



vi. Visualize the data in various dimensions



EXPERIMENT-3

Aim: Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets

a) Explore various options available in Weka for preprocessing data and apply Unsupervised filters like Discretization, Resample filter, etc. on each dataset
b) Load weather. nominal, Iris, Glass datasets into Weka and run Apriori Algorithm with different support and confidence values.
c) Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.

a) Explore various options available in Weka for preprocessing data and apply Unsupervised filters like Discretization, Resample filter, etc. on each dataset

Procedure:

Step1:Loading the data labor.arff from **C:\Program Files\Weka-3-8\data**. We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step2:Once the data is loaded, weka will recognize the attributes and during the scan of the data weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step3:Clicking on an attribute in the left panel will show the basic statistics on the attributes for the categorical attributes the frequency of each attribute value is shown, while for continuous attributes we can obtain min, max, mean, standard deviation and deviation etc.,

Step4:The visualization in the right button panel in the form of cross-tabulation across two attributes.

Note:we can select another attribute using the dropdown list.

Step5:Selecting or filtering attributes

Discretization

Sometimes association rule mining can only be performed on categorical data. This requires performing discretization on numeric or continuous attributes. In the following example let us discretize duration attribute.

- Let us divide the values of duration attribute into three bins(intervals).
- First load the dataset into weka(labor.arff)
- Select the duration attribute.
- Activate filter-dialog box and select "WEKA.filters.unsupervised.attribute .discretize" from the list.
- To change the defaults for the filters, click on the box immediately to the right of the choose button.

- We enter the index for the attribute to be discretized. In this case the attribute is duration So we must enter '1' corresponding to the duration attribute.
- Enter '10' as the number of bins. Leave the remaining field values as they are.
- Click OK button.
- Click apply in the filter panel. This will result in a new working relation with the selected attribute partition into 10 bin.
- Save the new working relation in a file called labor-data-discretized.arff

Resample filter: We use this filter when we want to Produces a random subsample of a dataset using either sampling with replacement or without replacement. The original dataset must fit entirely in memory. The number of instances in the generated dataset may be specified. When used in batch mode, subsequent batches are NOT resampled.

Steps to apply resample filter:

- First load the dataset into weka(labor.arff)
- Activate filter-dialog box and select "weka.filters.unsupervised.instance.Resample" from the list.
- To change the defaults for the filters, click on the box immediately to the right of the choose button.
- We change the value of sampleSizePercent from 100 to required value.
- Let change the value to 50
- Click OK button.
- Click apply in the filter panel. This will result in a new working relation with 50% of instances

Removing an attribute-When we need to remove an attribute, we can do this by using the attribute filters in weka. In the filter model panel, click on choose button, This will show a popup window with a list of available filters.

Scroll down the list and select the "weka.filters.unsupervised.attribute.remove" filters.

a)Next click the textbox immediately to the right of the choose button. In the resulting dialog box enter the index of the attribute to be filtered out.

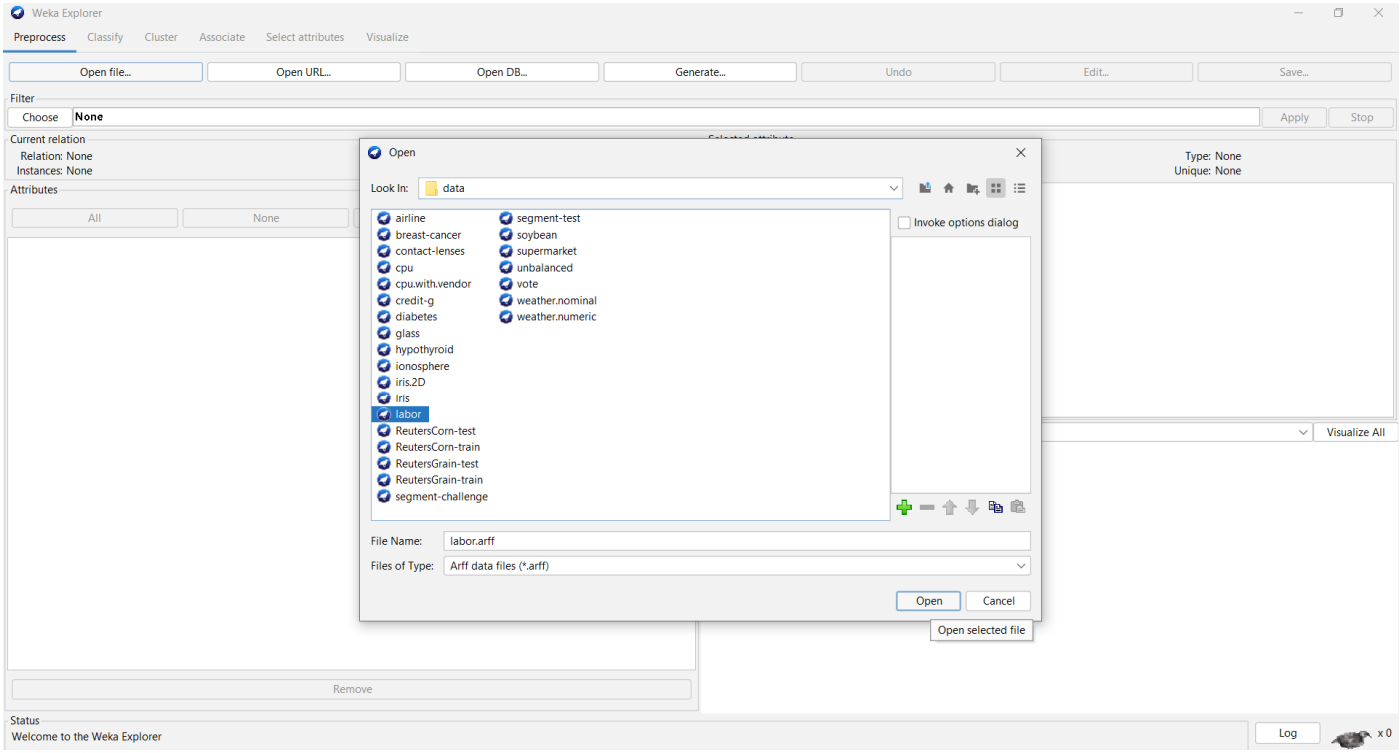
b) Make sure that invert selection option is set to false. The click OK now in the filter box. you will see "Remove-R-1".

c) Click the apply button to apply filter to this data. This will remove the attribute and create new working relation.

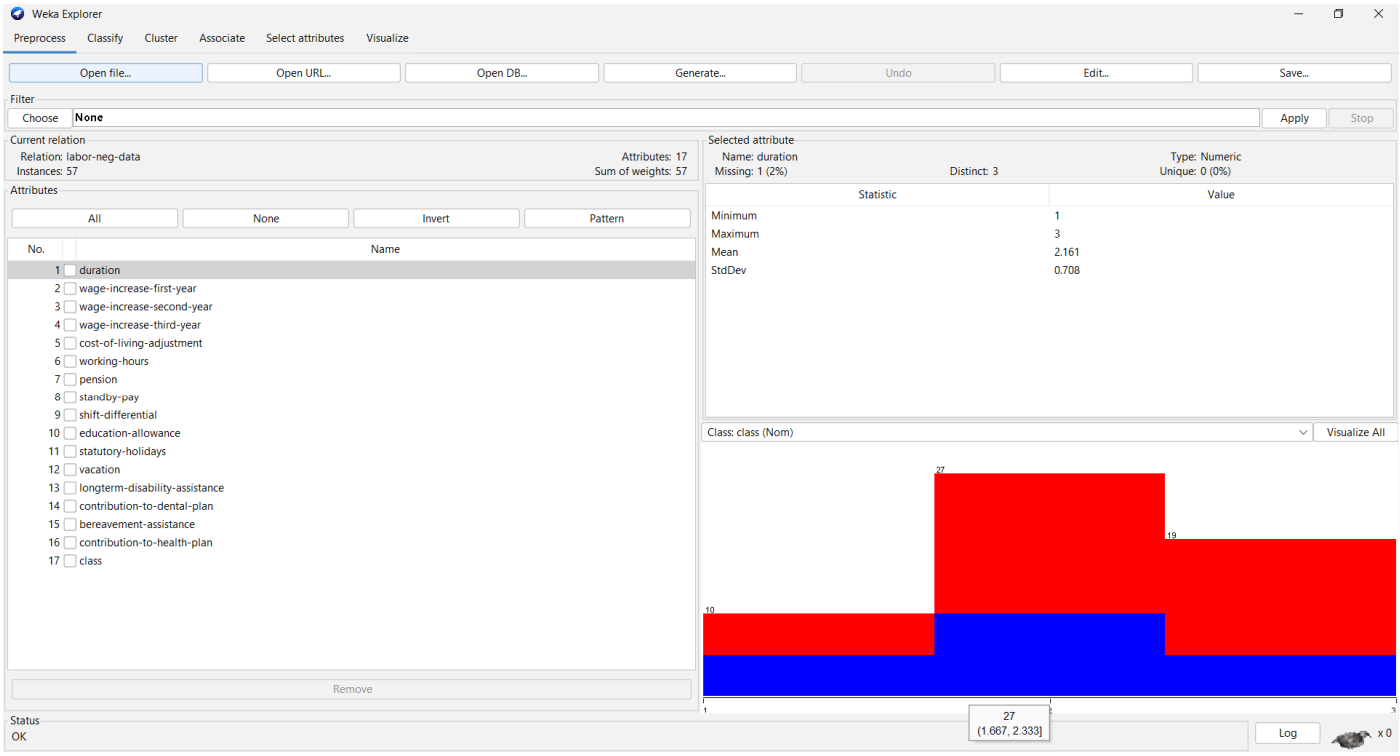
d) Save the new working relation as an arff file by clicking save button on the top(button)panel.(labor.arff)

Results:

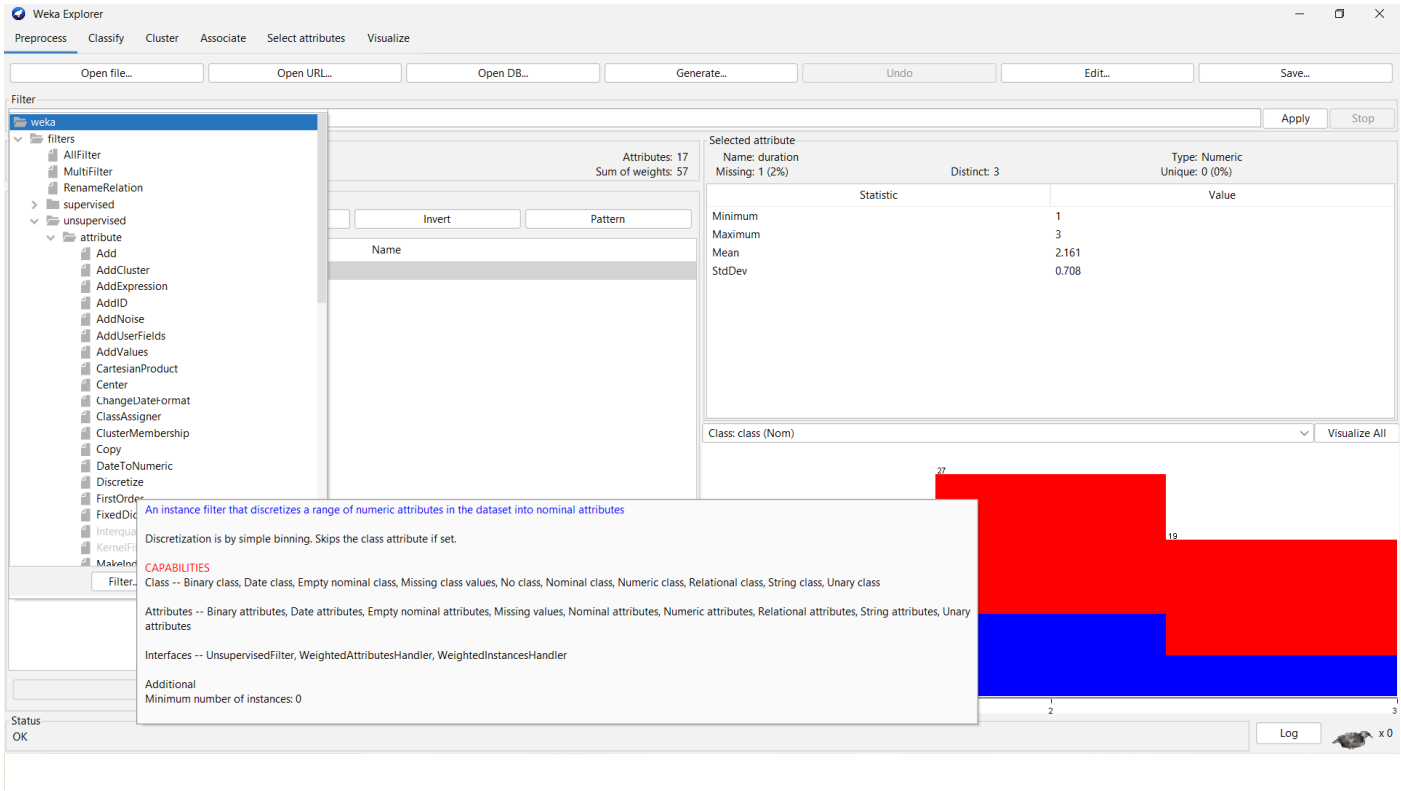
The following Screen shows the process of loading a file



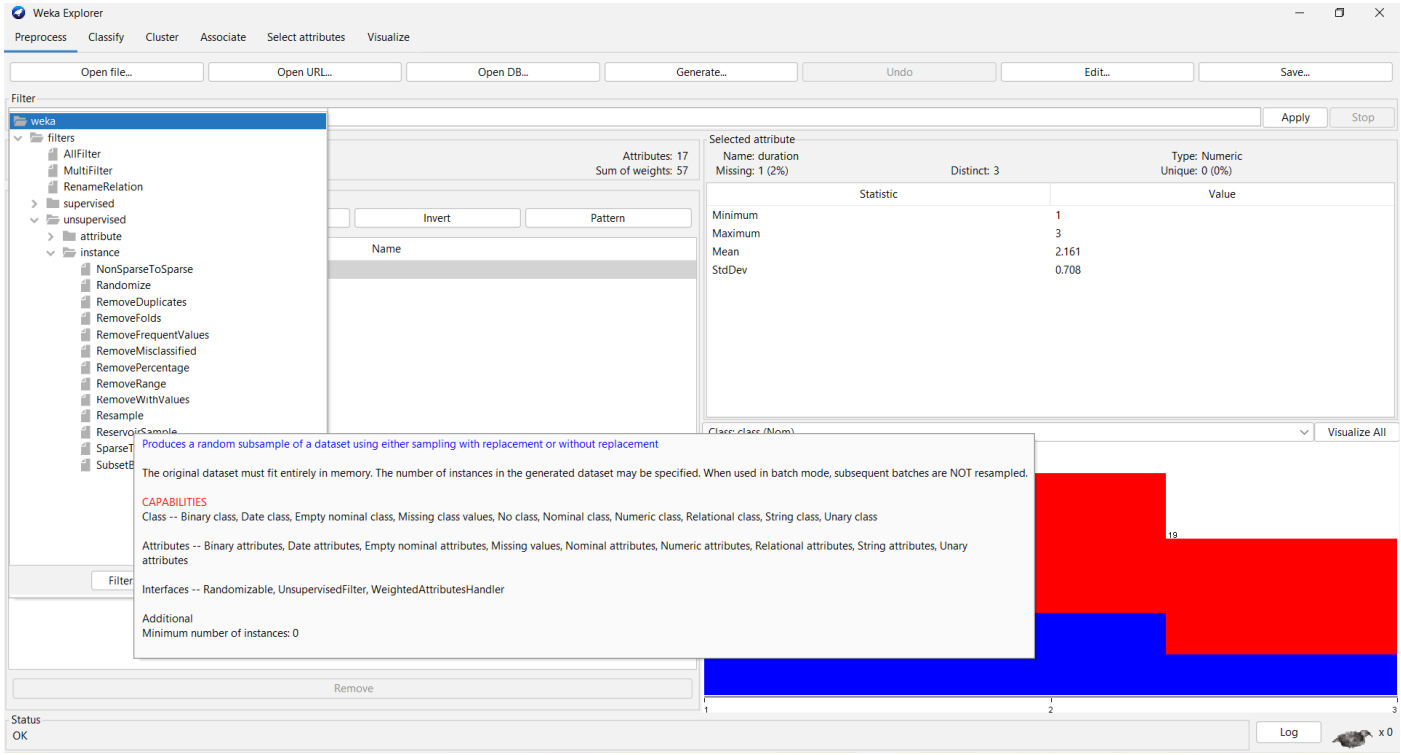
The following Screen shows the view of preprocessing window when data loaded



The following Screen shows the selection of Discretization



The following Screen shows the selection of resample filter



b) Load weather. nominal, Iris, Glass datasets into Weka and run Apriori Algorithm with different support and confidence values.

Procedure:

Step1:Open weka and then go to Explorer interface

Step2:Loading the data weather.nominal.arff from **C:\Program Files\Weka-3.8\data**. We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4:Now check whether all attributes are nominal or not (because apriori algorithm works on only nominal, binary, unary attributes) by clicking on an attribute in the left panel will show the basic statistics on the attributes . if the attribute is not nominal or binary or unary then apply discretization on that attribute to make that attribute as nominal.

NOTE: In weather.nominal file all attributes are nominal so we can directly apply apriori

Step5:Now click on Association tab and then select apriori algorithm by using choose button .After that set the different options in apriori algorithm

Options in apriori-

- **minMetric** -- Minimum metric score. Consider only rules with scores higher than this value.
- **verbose** -- If enabled the algorithm will be run in verbose mode.
- **numRule**s -- Number of rules to find.
- **lowerBoundMinSupport** -- Lower bound for minimum support.
- **classIndex** -- Index of the class attribute. If set to -1, the last attribute is taken as class attribute.
- **outputItemSets** -- If enabled the itemsets are output as well.
- **car** -- If enabled class association rules are mined instead of (general) association rules.
- **doNotCheckCapabilities** -- If set, associator capabilities are not checked before associator is built (Use with caution to reduce runtime).
- **removeAllMissingCols** -- Remove columns with all missing values.
- **significanceLevel** -- Significance level. Significance test (confidence metric only).
- **treatZeroAsMissing** -- If enabled, zero (that is, the first value of a nominal) is treated in the same way as a missing value.
- **delta** -- Iteratively decrease support by this factor. Reduces support until min support is reached or required number of rules has been generated.

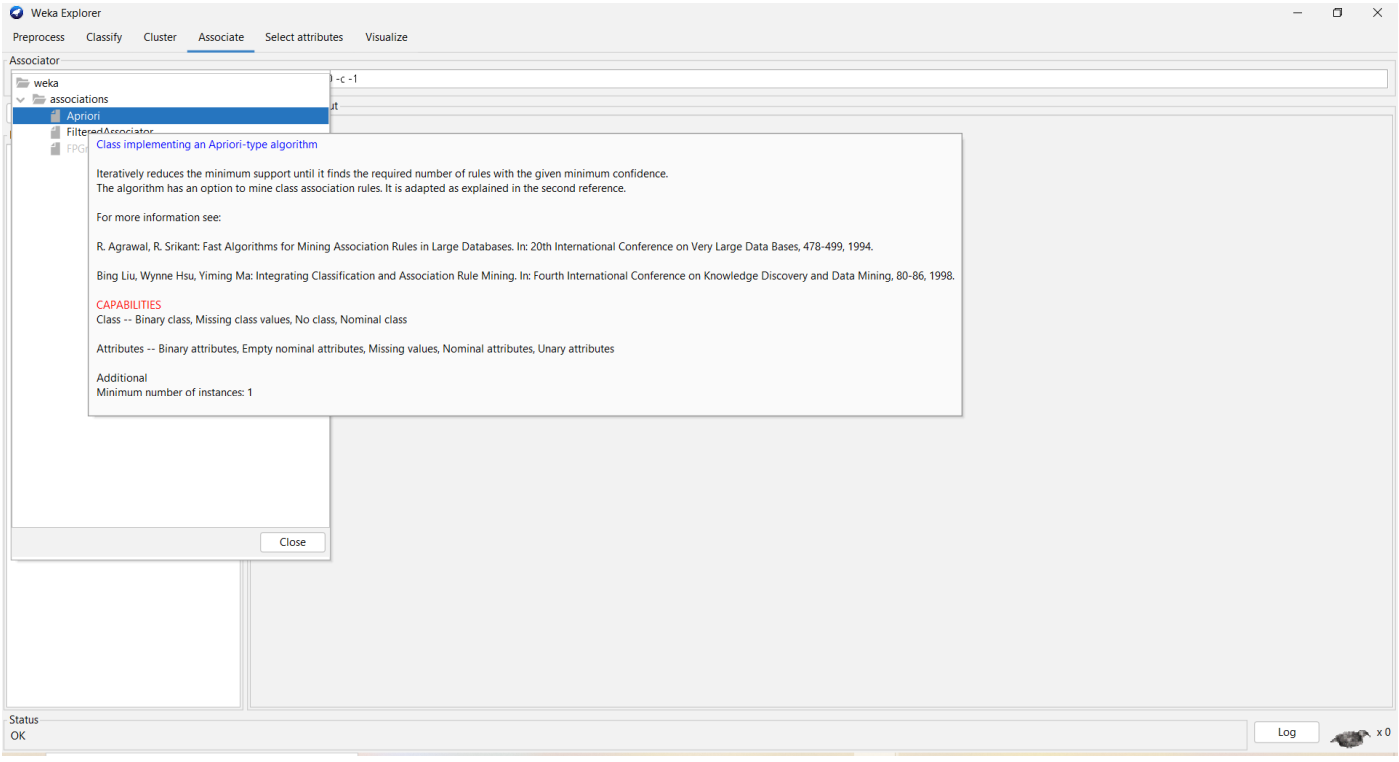
- **metricType** -- Set the type of metric by which to rank rules. Confidence is the proportion of the examples covered by the premise that are also covered by the consequence (Class association rules can only be mined using confidence). Lift is confidence divided by the proportion of all examples that are covered by the consequence. This is a measure of the importance of the association that is independent of support. Leverage is the proportion of additional examples covered by both the premise and consequence above those expected if the premise and consequence were independent of each other. The total number of examples that this represents is presented in brackets following the leverage. Conviction is another measure of departure from independence. Conviction is given by P(premise)P(!consequence) / P(premise, !consequence).

- **upperBoundMinSupport** -- Upper bound for minimum support. Start iteratively decreasing minimum support from this value.

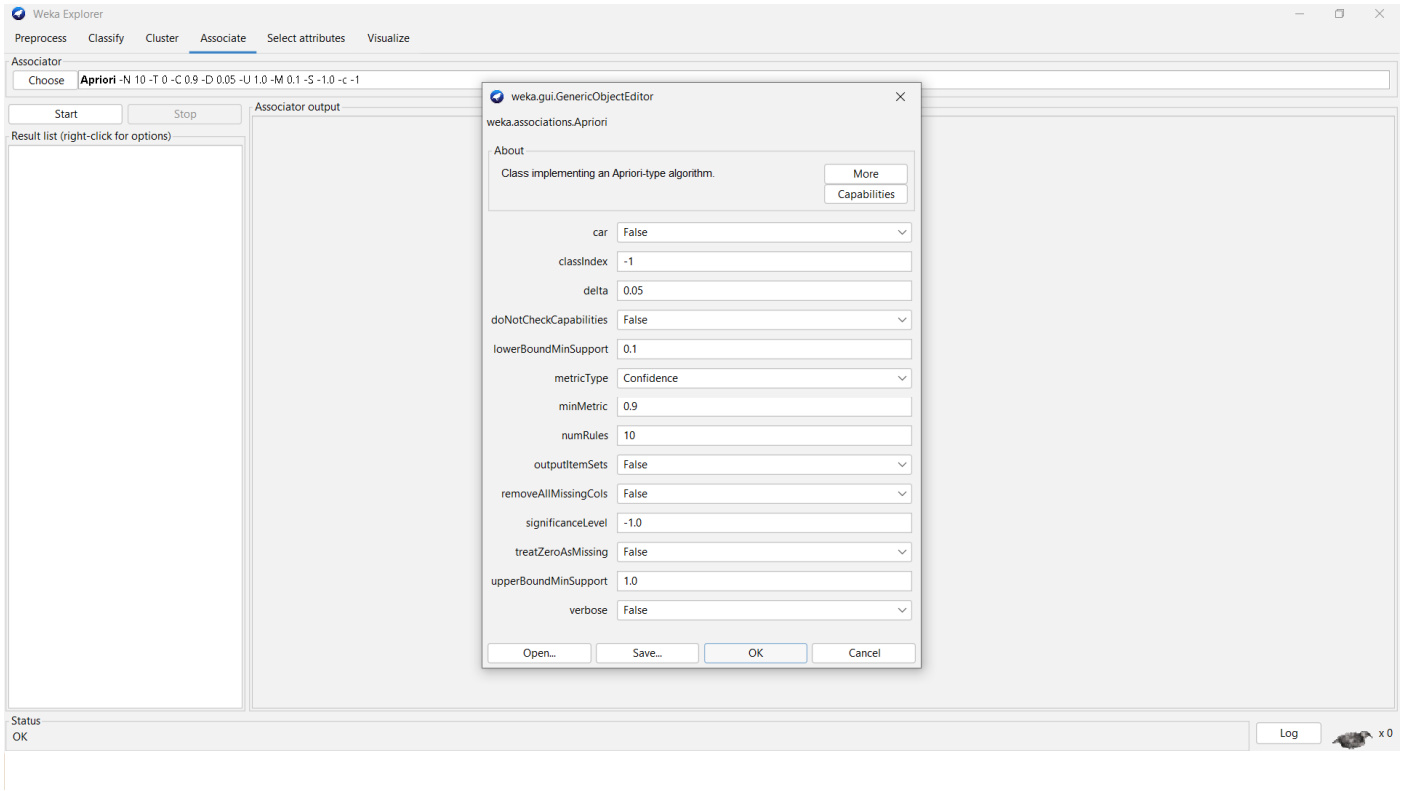
Step6:After setting the options then click on start button to generate the output(i.e association rules)

Results:

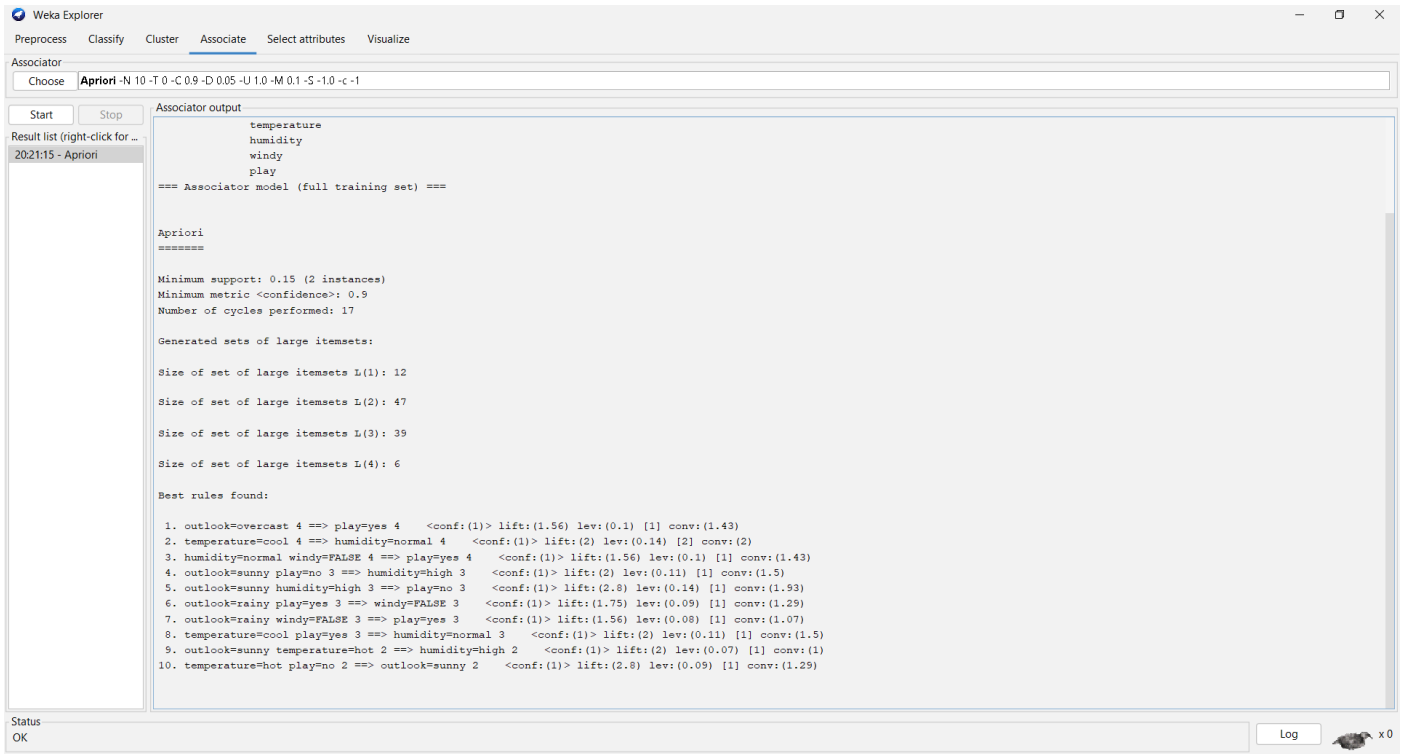
The following Screen shows selection of 'apriori algorithm'.



The following Screen shows how to set the options for 'apriori algorithm'.



The following Screen shows the result after applying 'apriori algorithm'



c) Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.

Procedure:

Step1:Open weka and then go to Explorer interface

Step2:Loading the data iris.arff from **C:\Program Files\Weka-3.8\data**. We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4:Now check whether all attributes are nominal or not (because apriori algorithm works on only nominal, binary, unary attributes) by clicking on an attribute in the left panel will show the basic statistics on the attributes . if the attribute is not nominal or binary or unary then apply discretization on that attribute to make that attribute as nominal.

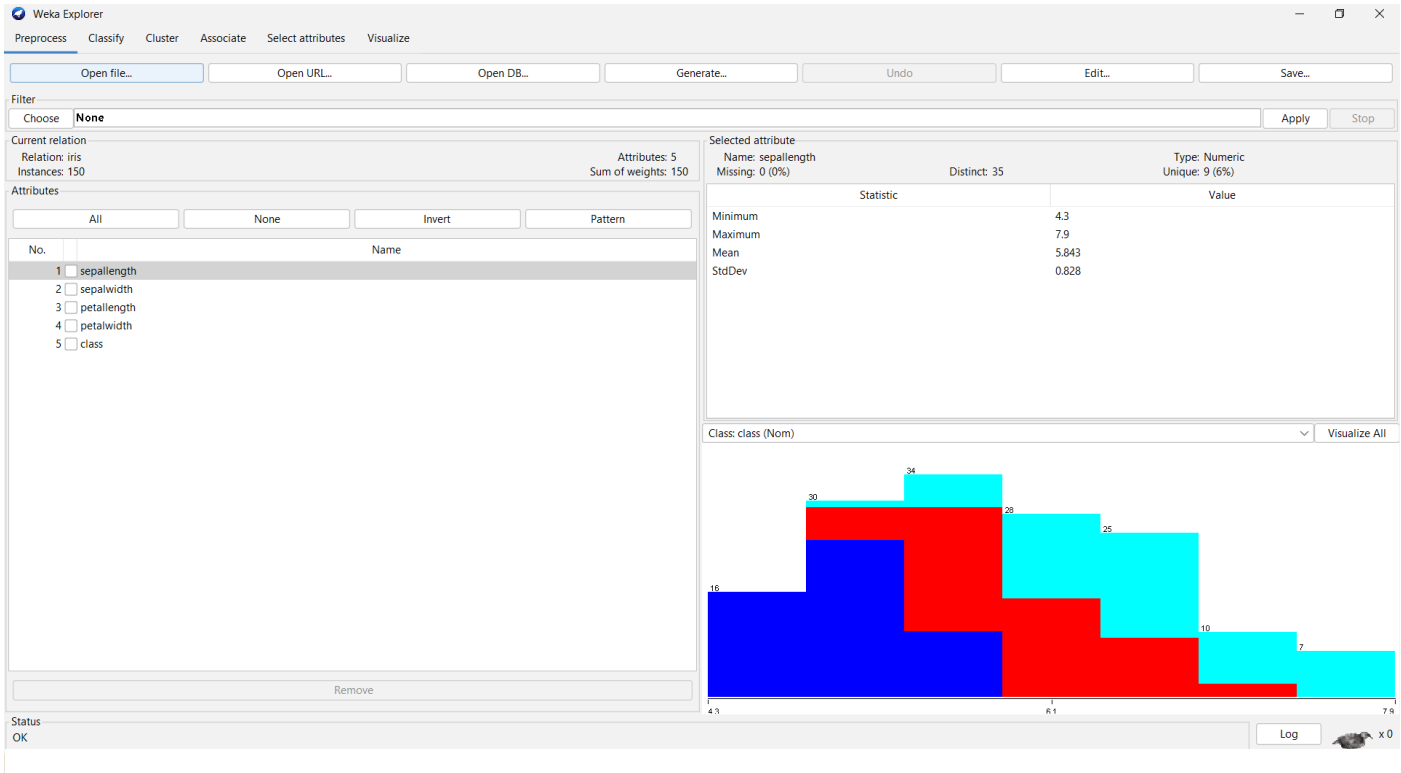
NOTE: In iris file all attributes are numeric so we can not directly apply apriori so first we have to apply discretization on all attributes then only we can apply apriori

Step5:Now click on Association tab and then select apriori algorithm by using choose button .After that set the different options in apriori algorithm

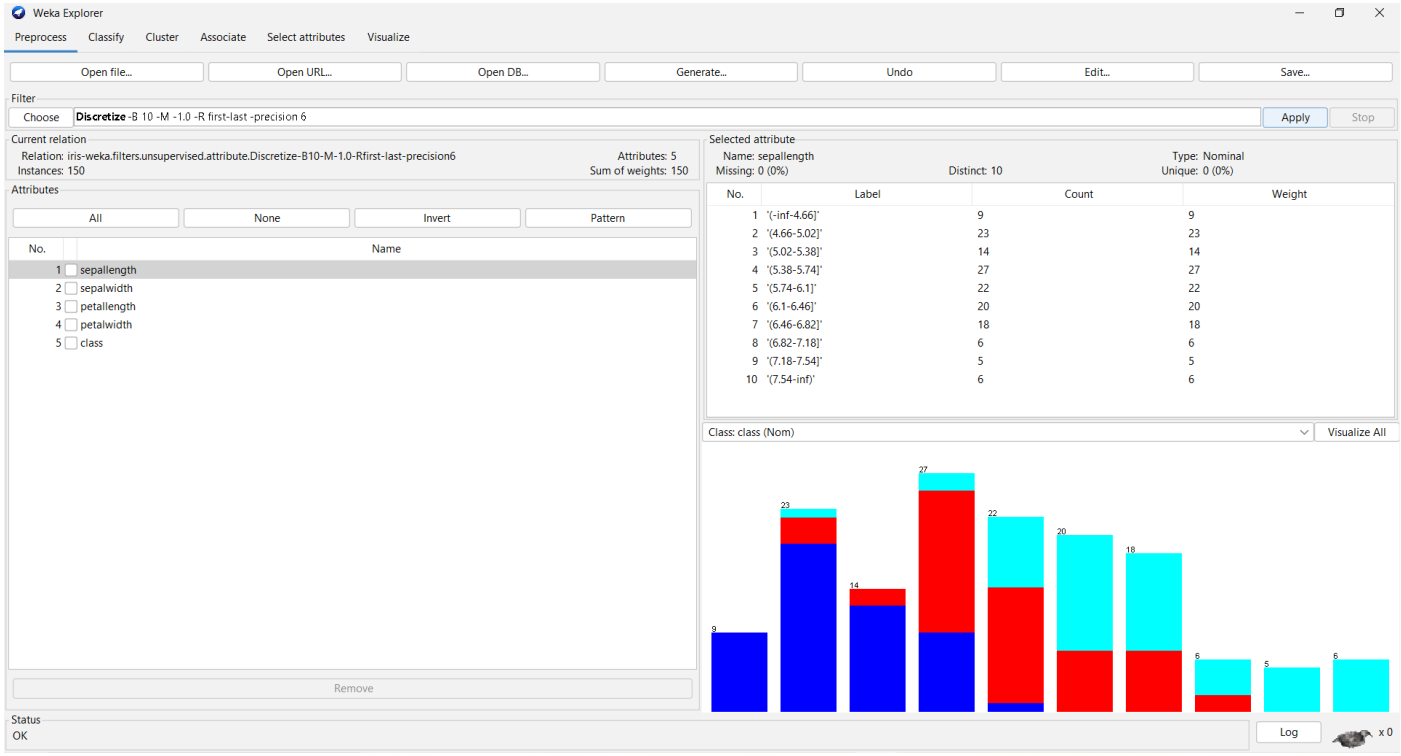
Step6:After setting the options then click on start button to generate the output(i.e association rules)

Results:

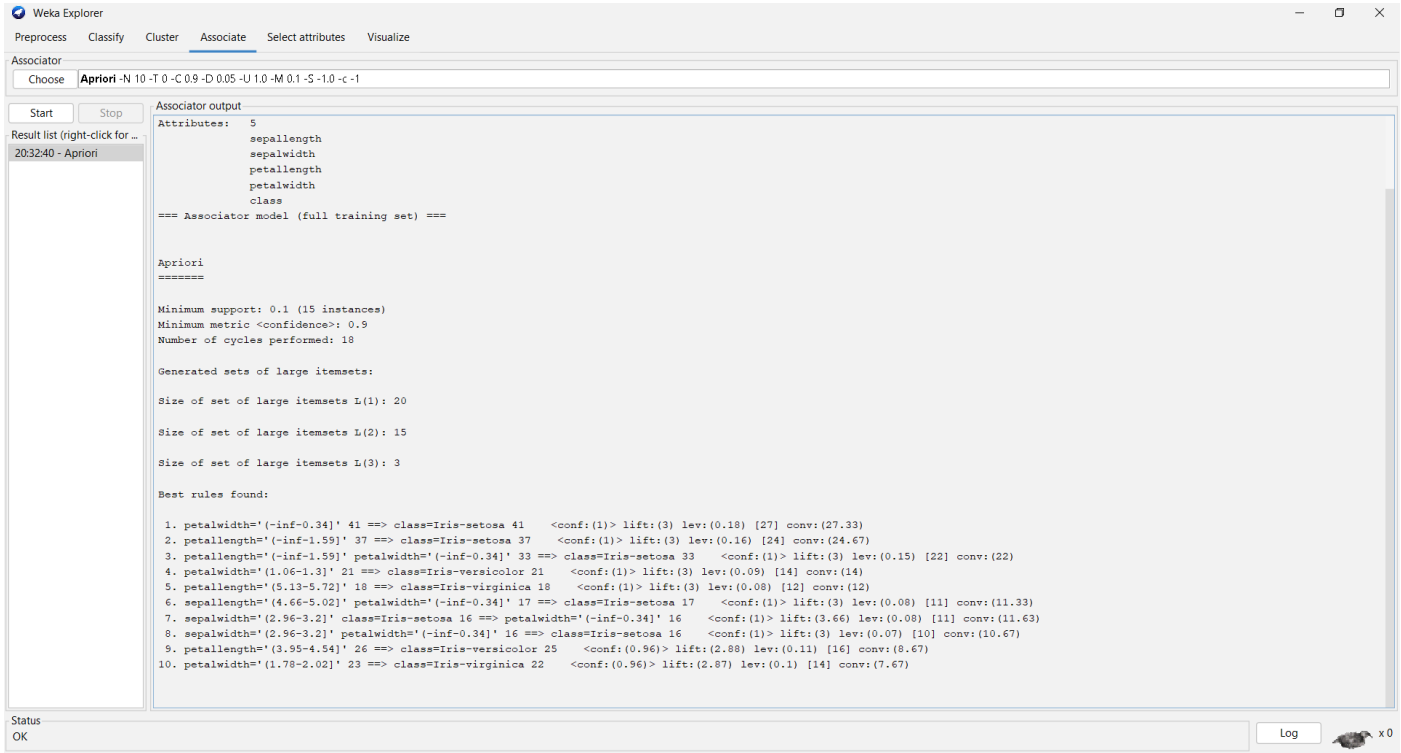
The following Screen shows iris data with numerical attributes.



The following Screen shows iris data after applying discretization.



The following Screen shows the result after applying 'apriori algorithm'



EXPERIMENT-4

Aim: Demonstrate performing classification on data sets

a) Load each dataset into Weka and run 1d3, J48 classification algorithm. Study the classifier output. Compute entropy values, Kappa statistic.
b) Extract if-then rules from the decision tree generated by the classifier, Observe the confusion matrix.
c) Load each dataset into Weka and perform Naïve-bayes classification and k-Nearest Neighbour classification. Interpret the results obtained.
d) Plot RoC Curves
e) Compare classification results of ID3, J48, Naïve-Bayes and k-NN classifiers for each dataset, and deduce which classifier is performing best and poor for each dataset and justify.

a) Load each dataset into Weka and run 1d3, J48 classification algorithm. Study the classifier output. Compute entropy values, Kappa statistic.

Procedure: j48 classification algorithm

Step1:Open weka and then go to Explorer interface

Step2:Loading the data iris.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4: Next we select the "classify" tab and click "choose" button to select the "j48"classifier

Step5: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values. The default version does perform some pruning but does not perform error pruning.

Options in j48:

- **seed** -- The seed used for randomizing the data when reduced-error pruning is used.
- **unpruned** -- Whether pruning is performed.
- **confidenceFactor** -- The confidence factor used for pruning (smaller values incur more pruning).
- **numFolds** -- Determines the amount of data used for reduced-error pruning. One fold is used for pruning, the rest for growing the tree.
- **numDecimalPlaces** -- The number of decimal places to be used for the output of numbers in the model.
- **batchSize** -- The preferred number of instances to process if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size.

- **reducedErrorPruning** -- Whether reduced-error pruning is used instead of C.4.5 pruning.
- **useLaplace** -- Whether counts at leaves are smoothed based on Laplace.
- **doNotMakeSplitPointActualValue** -- If true, the split point is not relocated to an actual data value. This can yield substantial speed-ups for large datasets with numeric attributes.
- **debug** -- If set to true, classifier may output additional info to the console.
- **subtreeRaising** -- Whether to consider the subtree raising operation when pruning.
- **saveInstanceData** -- Whether to save the training data for visualization.
- **binarySplits** -- Whether to use binary splits on nominal attributes when building the trees.
- **doNotCheckCapabilities** -- If set, classifier capabilities are not checked before classifier is built (Use with caution to reduce runtime).
- **minNumObj** -- The minimum number of instances per leaf.
- **useMDLcorrection** -- Whether MDL correction is used when finding splits on numeric attributes.
- **collapseTree** -- Whether parts are removed that do not reduce training error**.**

Step6: Under the "text" options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don't have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

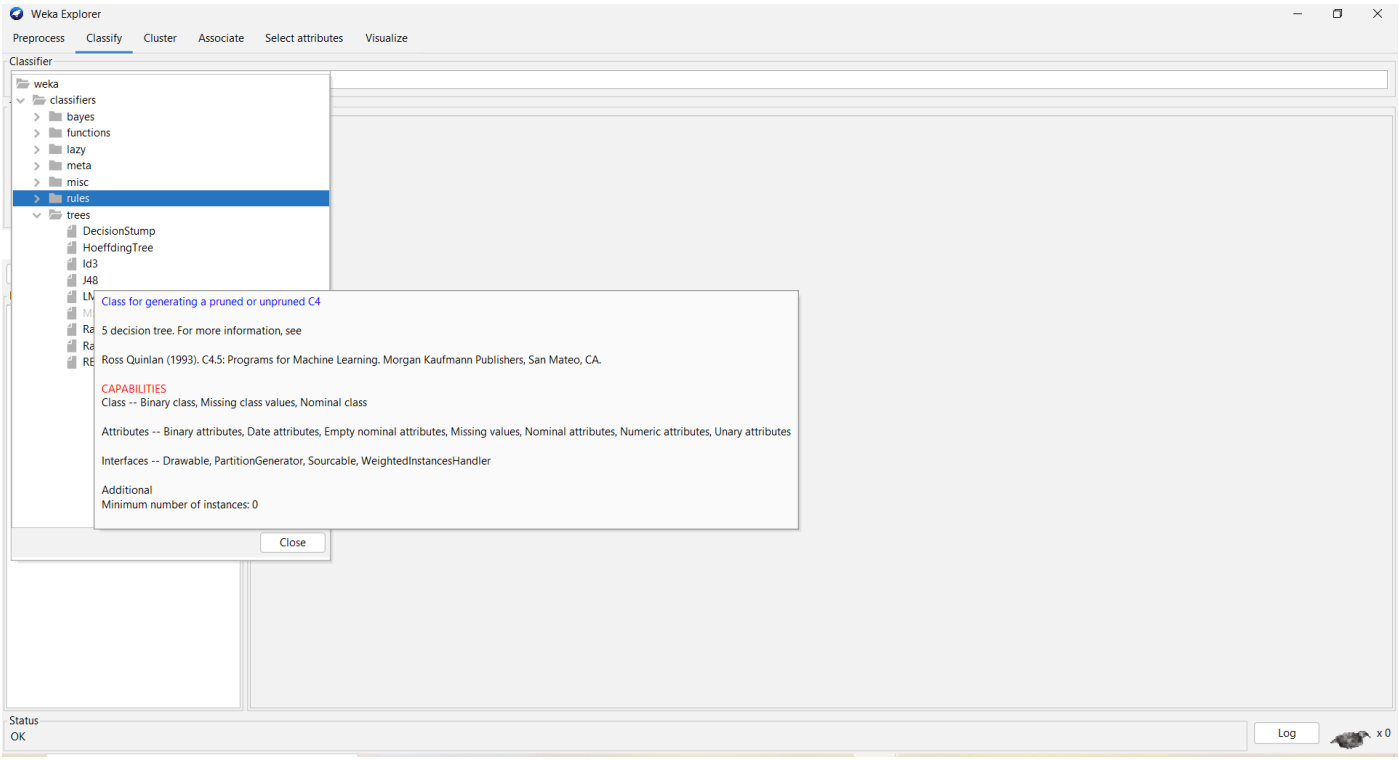
Step7: We now click "start" to generate the model .the Ascii version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

Step8: Now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting "visualize tree" from the pop-up menu.

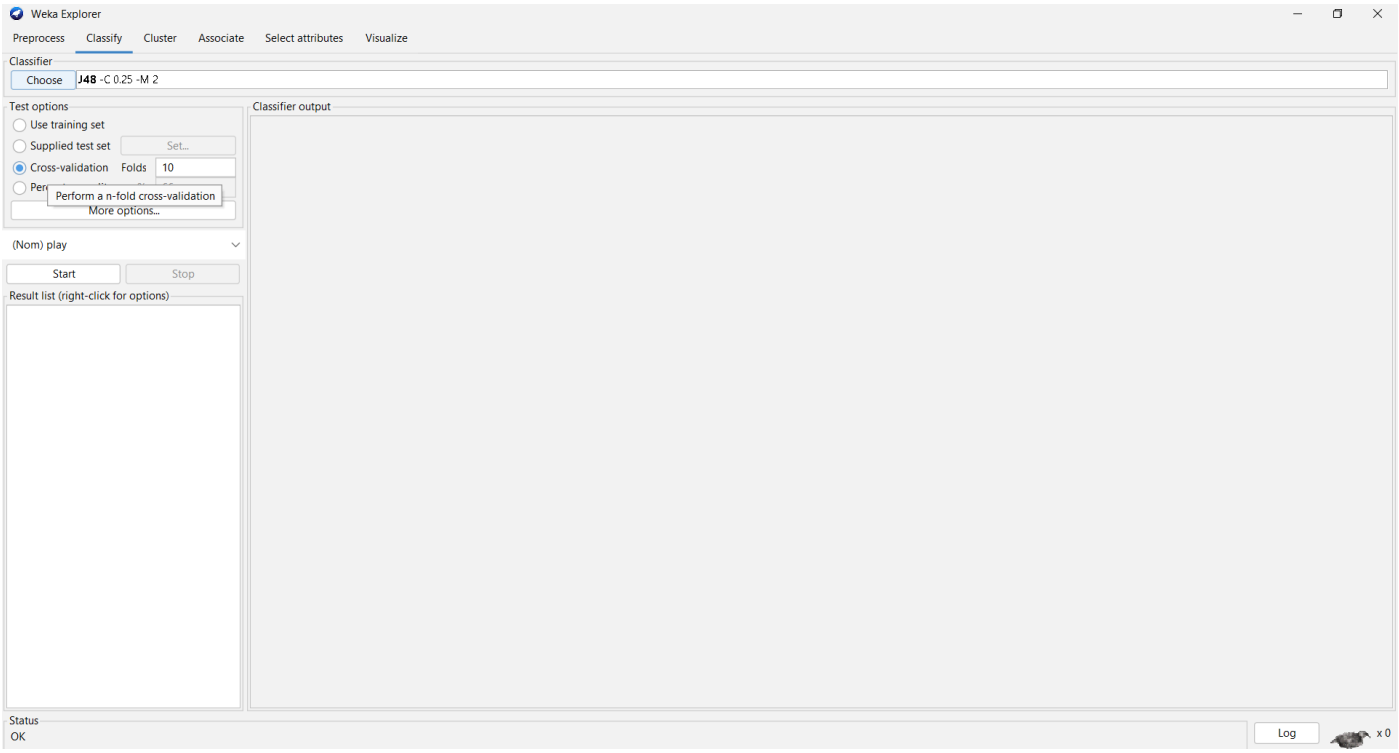
Step9: We can use our model to classify the new instances. In the main panel under "text" options click the "supplied test set" radio button and then click the "set" button. This wills pop-up a window which will allow you to open the file containing test instances.

Results:

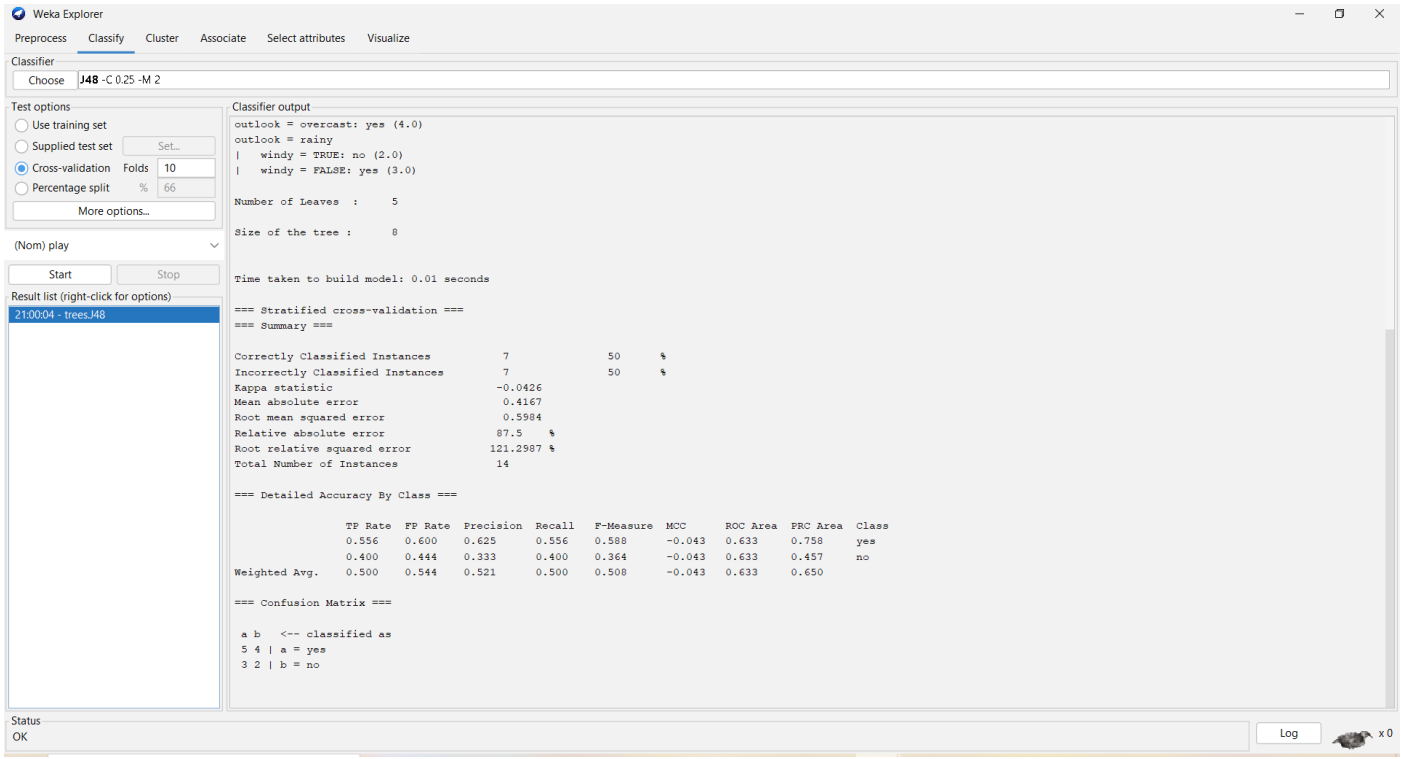
The following Screen shows selection of ‘j48’.



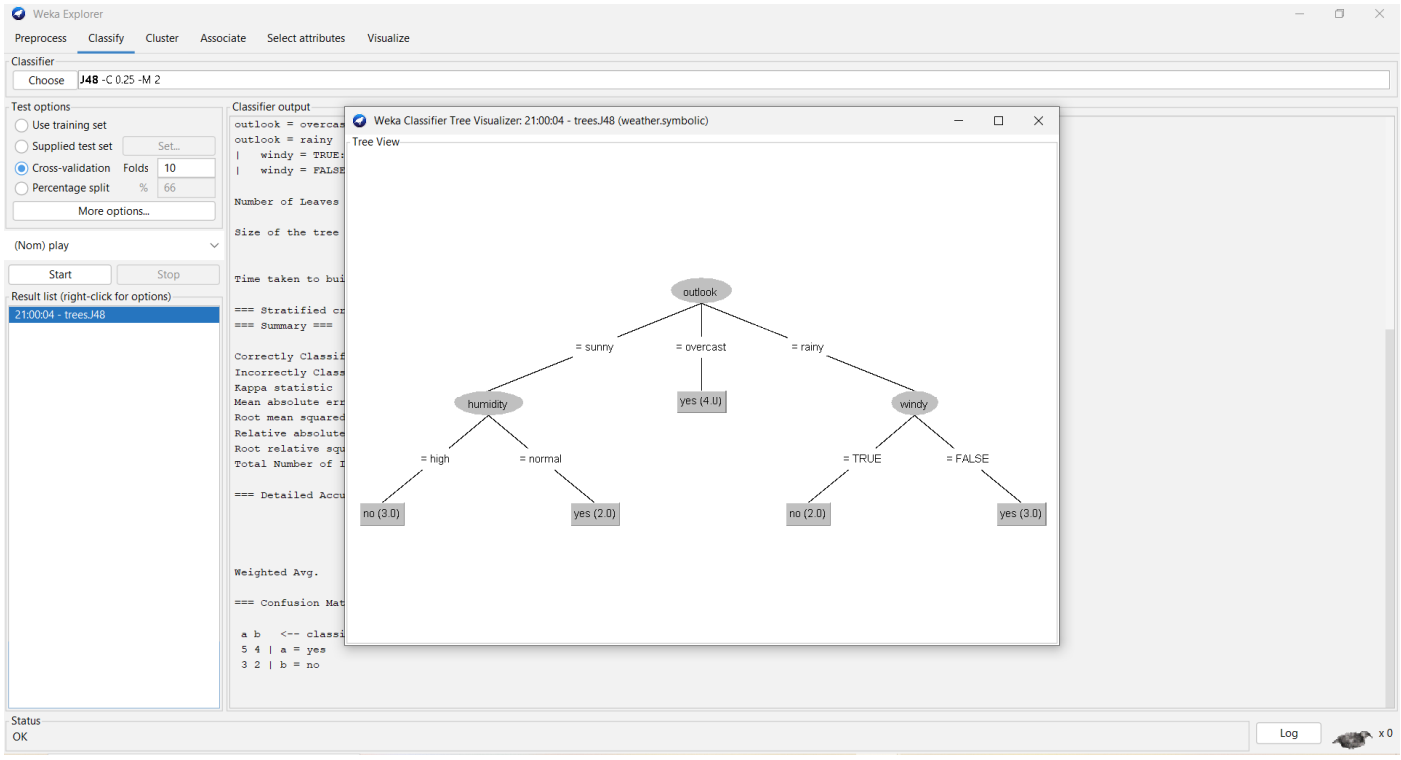
The following Screen shows selection of ‘crossfold option’.



The following Screen shows the result after applying 'j48'



The following Screen shows the tree visualization



Procedure: id3 classification algorithm

Step1:Open weka and then go to Explorer interface

Step2:Loading the data weather.nominal.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4:Now we have to check whether all the attributes are nominal or not and is there any missing values in data set (because id3 algorithm works for only nominal attributes without any missing values) preprocess the data if any changes needed.

Step5: Next we select the "classify" tab and click "choose" button to select the "id3"classifier

Step6: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values. The default version does perform some pruning but does not perform error pruning.

Options in id3:

- **numDecimalPlaces** -- The number of decimal places to be used for the output of numbers in the model.
- **batchSize** -- The preferred number of instances to process if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size.
- **debug** -- If set to true, classifier may output additional info to the console.
- **doNotCheckCapabilities** -- If set, classifier capabilities are not checked before classifier is built (Use with caution to reduce runtime).

Step7: Under the "text" options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don't have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

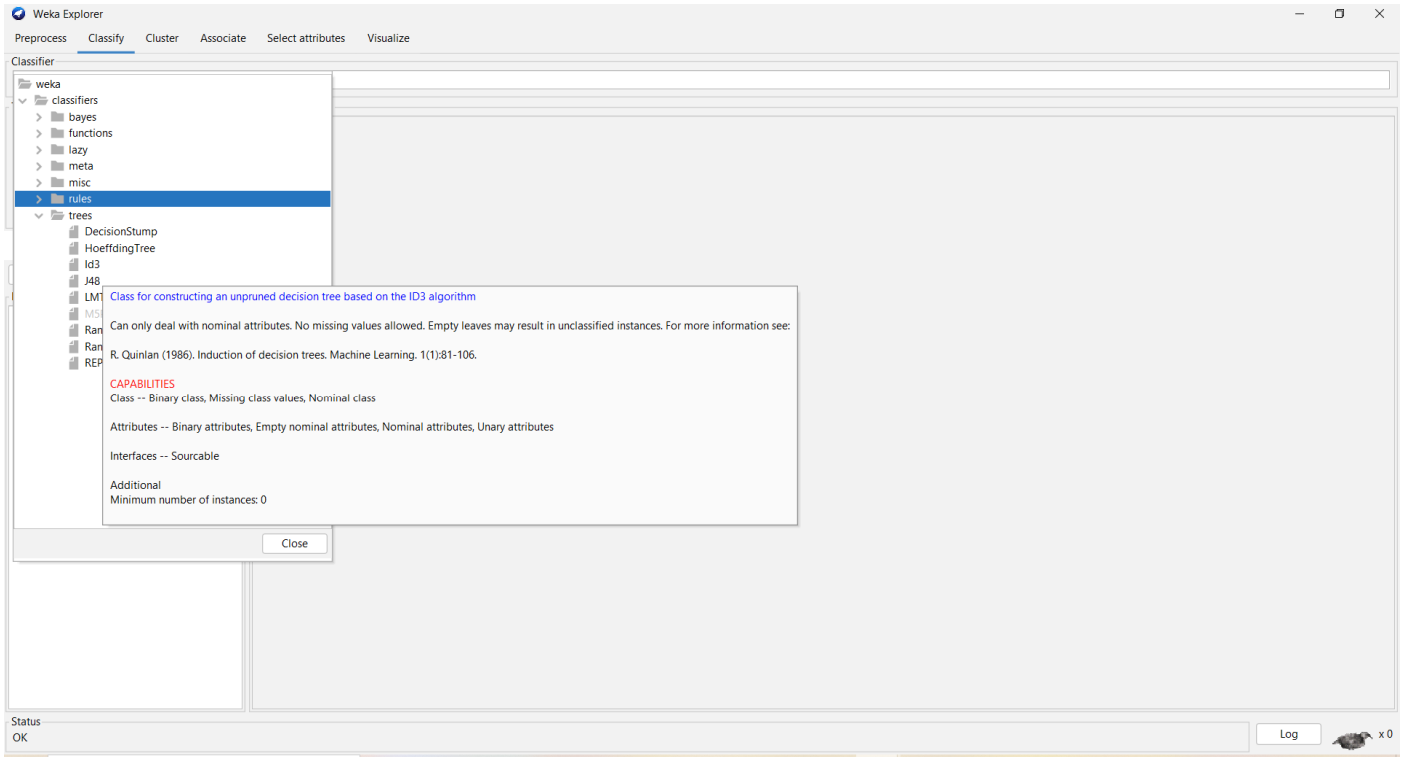
Step8: We now click "start" to generate the model .the Ascii version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

Step9: Now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting "visualize tree" from the pop-up menu.

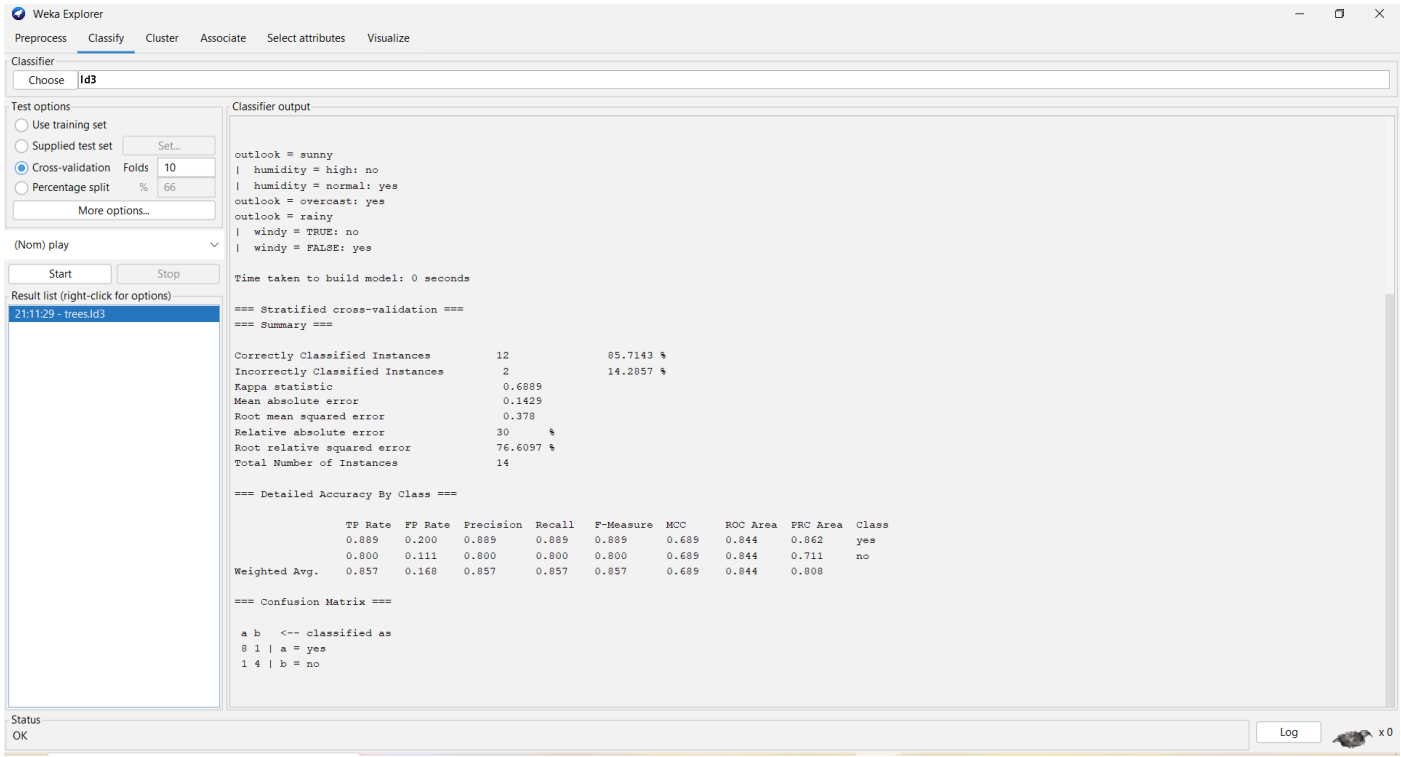
Step10: We can use our model to classify the new instances. In the main panel under "text" options click the "supplied test set" radio button and then click the "set" button. This wills pop-up a window which will allow you to open the file containing test instances.

Results:

The following Screen shows selection of 'id3'



The following Screen shows the result after applying 'id3'



b) Extract if-then rules from the decision tree generated by the classifier, Observe the confusion matrix.

Procedure:

Step1:Open weka and then go to Explorer interface

Step2:Loading the data iris.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4: Next we select the "classify" tab and click "choose" button to select the "DecisionTable" classifier

Step5: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values.

Options in decision table:

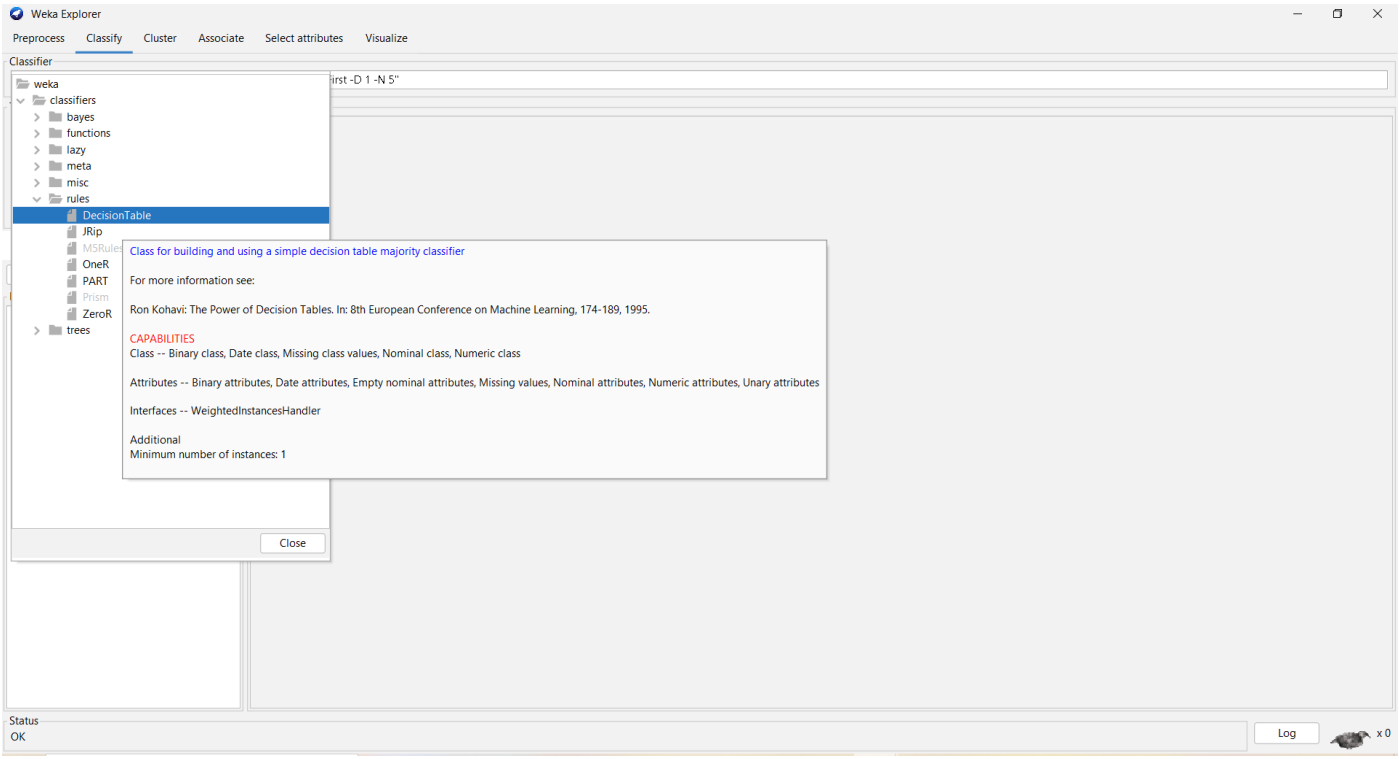
- numDecimalPlaces -- The number of decimal places to be used for the output of numbers in the model.
- batchSize -- The preferred number of instances to process if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size.
- debug -- If set to true, classifier may output additional info to the console.
- doNotCheckCapabilities -- If set, classifier capabilities are not checked before classifier is built (Use with caution to reduce runtime).
- evaluationMeasure -- The measure used to evaluate the performance of attribute combinations used in the decision table.
- search -- The search method used to find good attribute combinations for the decision table.
- displayRules -- Sets whether rules are to be printed.
- useIBk -- Sets whether IBk should be used instead of the majority class.
- crossVal -- Sets the number of folds for cross validation (1 = leave one out).

Step6: Under the "text" options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don't have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

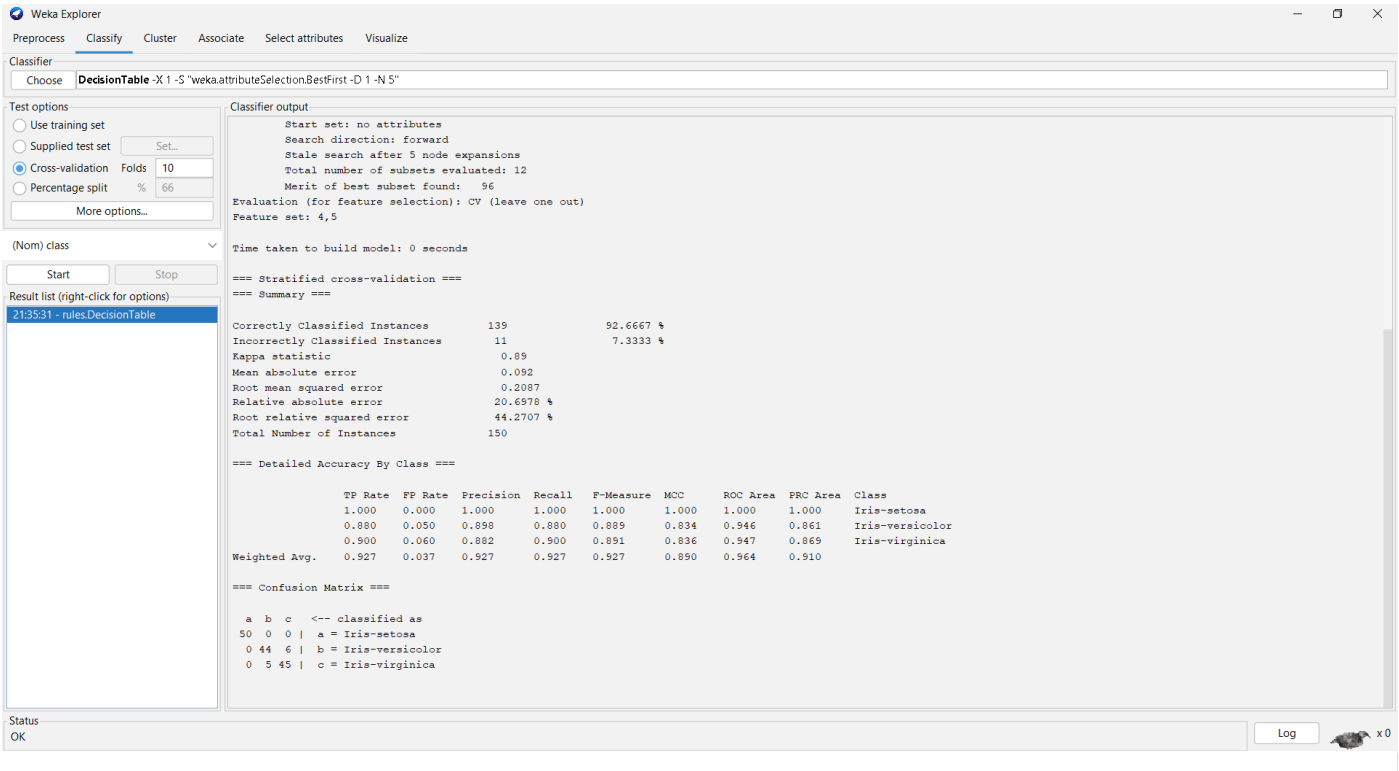
Step7: We now click "start" to generate the model .the evaluation statistic will appear in the right panel when the model construction is complete.

Results:

The following Screen shows selection of ‘decision table’



The following Screen shows the result after applying ‘decision table’



c) Load each dataset into Weka and perform Naïve-bayes classification and k-Nearest Neighbour classification. Interpret the results obtained.

Procedure: Naïve-bayes classification

Step1:Open weka and then go to Explorer interface

Step2:Loading the data iris.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4: Next we select the "classify" tab and click "choose" button to select the "**Naïve-bayes**" classifier

Step5: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values.

Options in Naïve-bayes:

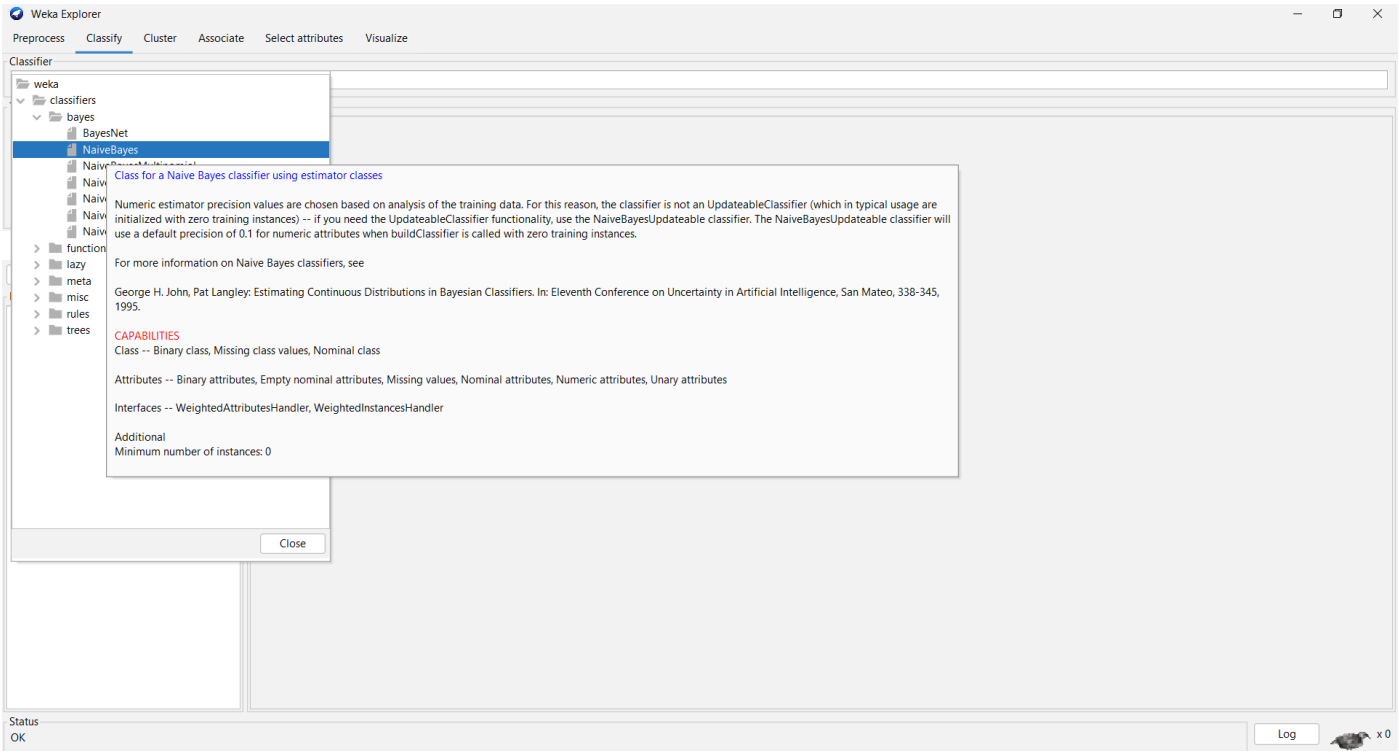
- useKernelEstimator -- Use a kernel estimator for numeric attributes rather than a normal distribution.
- numDecimalPlaces -- The number of decimal places to be used for the output of numbers in the model.
- batchSize -- The preferred number of instances to process if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size.
- debug -- If set to true, classifier may output additional info to the console.
- displayModelInOldFormat -- Use old format for model output. The old format is better when there are many class values. The new format is better when there are fewer classes and many attributes.
- doNotCheckCapabilities -- If set, classifier capabilities are not checked before classifier is built (Use with caution to reduce runtime).
- useSupervisedDiscretization -- Use supervised discretization to convert numeric attributes to nominal ones.

Step6: Under the "text" options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don't have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

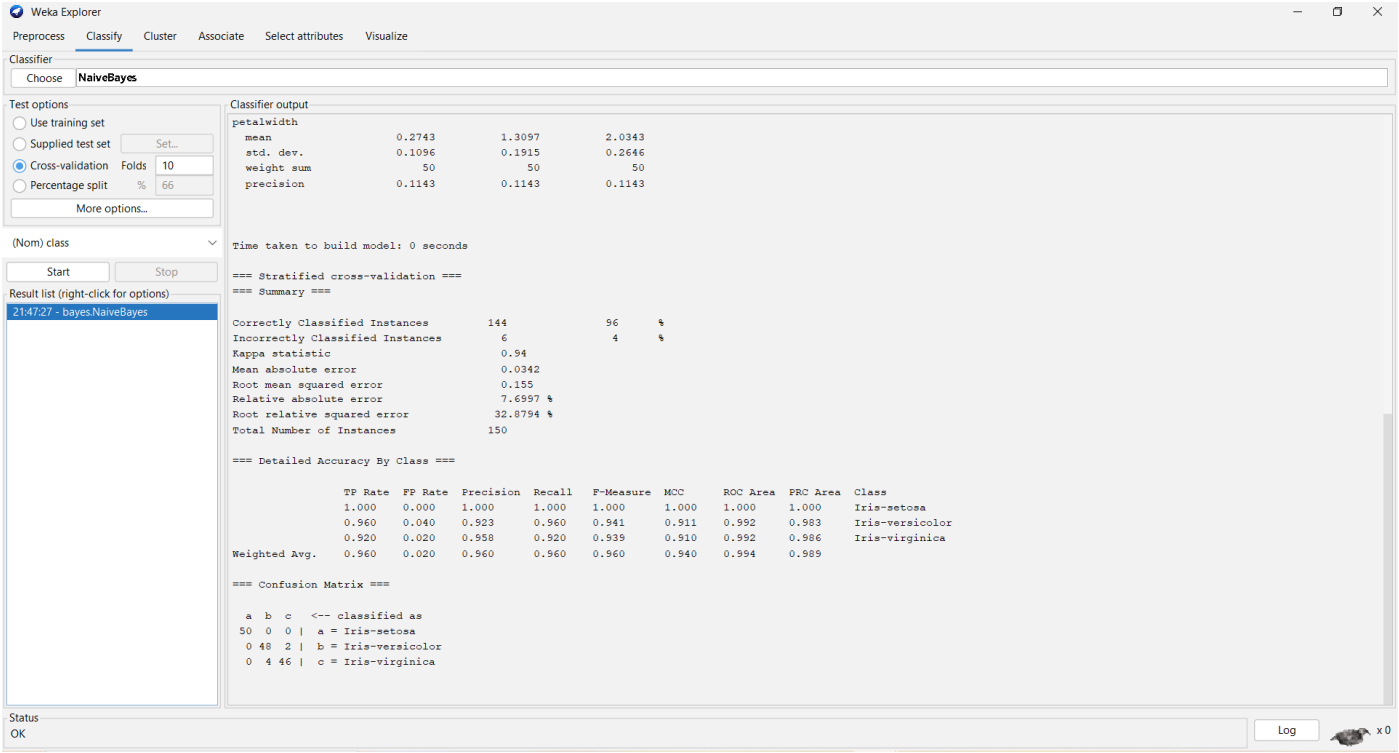
Step7: We now click "start" to generate the model .the evaluation statistic will appear in the right panel when the model construction is complete.

Results:

The following Screen shows selection of 'Naïve-bayes':



The following Screen shows the result after applying 'Naïve-bayes'



Procedure: k-Nearest Neighbour classification.

Step1:Open weka and then go to Explorer interface

Step2:Loading the data iris.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4: Next we select the "classify" tab and click "choose" button to select the "**IBK**" classifier

Step5: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values.

Options in IBK:

- numDecimalPlaces -- The number of decimal places to be used for the output of numbers in the model.
- batchSize -- The preferred number of instances to process if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size.
- KNN -- The number of neighbours to use.
- distanceWeighting -- Gets the distance weighting method used.
- nearestNeighbourSearchAlgorithm -- The nearest neighbour search algorithm to use (Default: weka.core.neighboursearch.LinearNNSearch).
- debug -- If set to true, classifier may output additional info to the console.
- windowSize -- Gets the maximum number of instances allowed in the training pool. The addition of new instances above this value will result in old instances being removed. A value of 0 signifies no limit to the number of training instances.
- doNotCheckCapabilities -- If set, classifier capabilities are not checked before classifier is built (Use with caution to reduce runtime).
- meanSquared -- Whether the mean squared error is used rather than mean absolute error when doing cross-validation for regression problems.
- crossValidate -- Whether hold-one-out cross-validation will be used to select the best k value between 1 and the value specified as the KNN parameter.

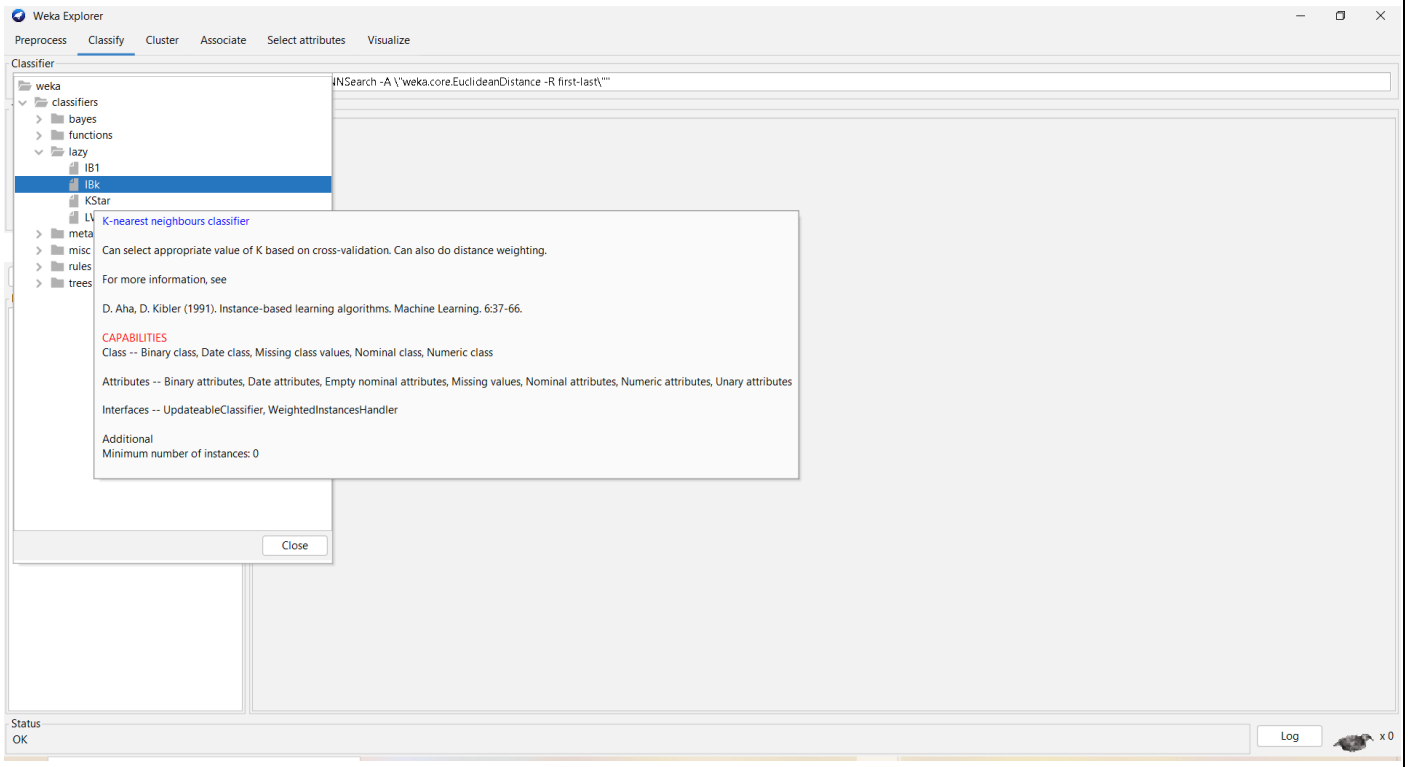
Step6: Under the "text" options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don't have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

Step7: We now click "start" to generate the model .the evaluation statistic will appear in the

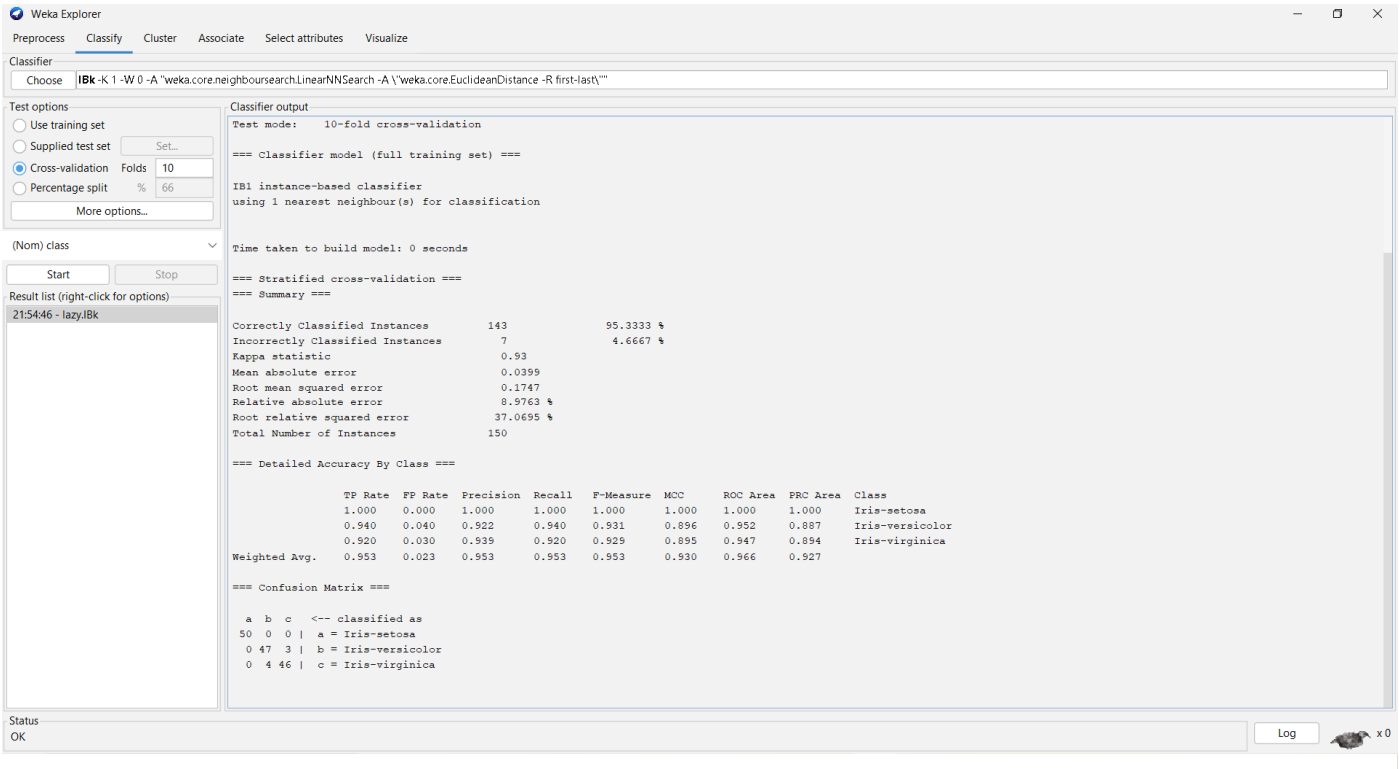
right panel when the model construction is complete.

Results:

The following Screen shows selection of 'k-Nearest Neighbour classification':



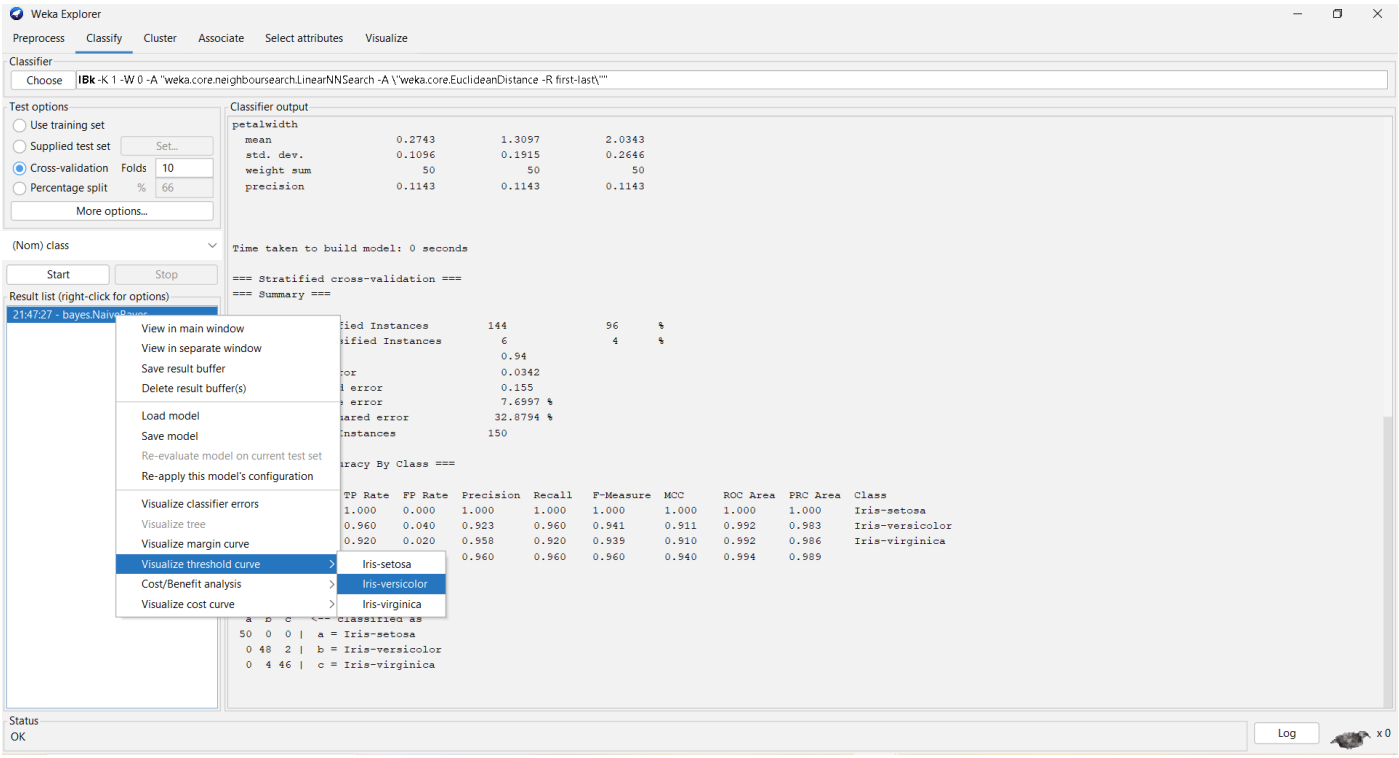
The following Screen shows the result after applying 'k-Nearest Neighbour classification'

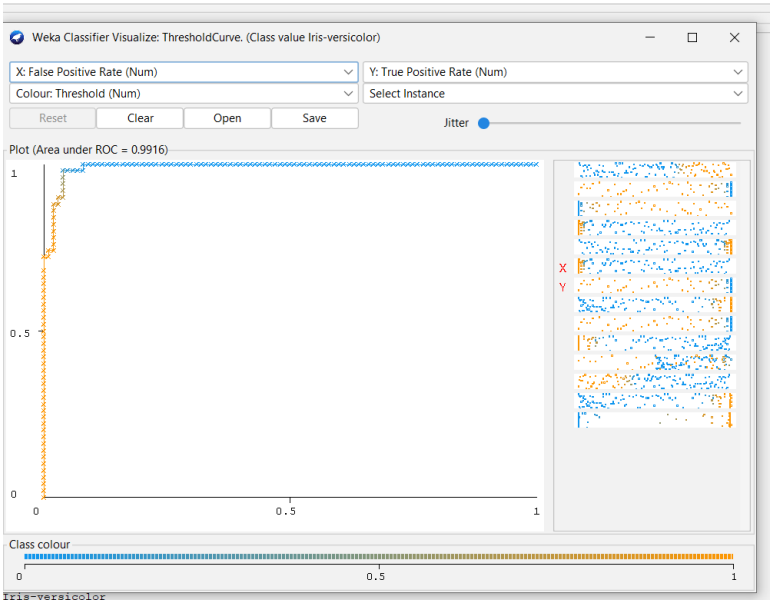


d) Plot RoC Curves

Procedure:

1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see different classification algorithms under bayes section.
3. In which we selected Naïve-Bayes algorithm & click on start option with ―use training set‖ test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix.
5. For plotting RoC Curves, we need to right click on ―bayes.NaiveBayes‖ for getting more options, In which we will select the ―Visualize Threshold Curve‖ & go to any class (Iris-setosa, Iris-versicolor, Iris-Virgincia) as shown in below snapshot.
6. After selecting an class, RoC (Receiver Operating Characteristic) Curve plot will be displayed which has X-Axis –False Positive (FP) rate and Y-Axis – True Positive (TP) rate.





EXPERIMENT-5

Aim: Demonstrate performing clustering of data sets

a) Load each dataset into Weka and run simple k-means clustering algorithm with different values of k (number of desired clusters). Study the clusters formed. Observe the sum of squared errors and centroids, and derive insights.
b) Explore other clustering techniques available in Weka.
c) Explore visualization features of Weka to visualize the clusters. Derive interesting insights and explain.

a) Load each dataset into Weka and run simple k-means clustering algorithm with different values of k (number of desired clusters). Study the clusters formed. Observe the sum of squared errors and centroids, and derive insights.

Procedure:

Step1: Open weka and then go to Explorer interface

Step2: Loading the data iris.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4: Next Inorder to perform clustering select the 'cluster' tab in the explorer

Step5: Now click on the choose button. This step results in a dropdown list of available clustering algorithms. In this case we select 'simple k-means'.

Step6: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button.

Options in Simple K mean:

- **seed** -- The random number seed to be used.
- **displayStdDevs** -- Display std deviations of numeric attributes and counts of nominal attributes.
- **numExecutionSlots** -- The number of execution slots (threads) to use. Set equal to the number of available cpu/cores
- **dontReplaceMissingValues** -- Replace missing values globally with mean/mode.
- **canopyMinimumCanopyDensity** -- If using canopy clustering for initialization and/or speedup this is the minimum T2-based density below which a canopy will be pruned during periodic pruning
- **debug** -- If set to true, clusterer may output additional info to the console.

- **canopyT2** -- The T2 distance to use when using canopy clustering. Values < 0 indicate that this should be set using a heuristic based on attribute standard deviation

- **numClusters** -- set number of clusters

- **doNotCheckCapabilities** -- If set, clusterer capabilities are not checked before clusterer is built (Use with caution to reduce runtime).

- **preserveInstancesOrder** -- Preserve order of instances.

- **maxIterations** -- set maximum number of iterations

- **canopyPeriodicPruningRate** -- If using canopy clustering for initialization and/or speedup this is how often to prune low density canopies during training

- **canopyMaxNumCanopiesToHoldInMemory** -- If using canopy clustering for initialization and/or speedup this is the maximum number of candidate canopies to retain in main memory during training of the canopy clusterer. T2 distance and data characteristics determine how many candidate canopies are formed before periodic and final pruning are performed. There may not be enough memory available if T2 is set too low.

- **initializationMethod** -- The initialization method to use. Random, k- means++, Canopy or farthest first

- **distanceFunction** -- The distance function to use for instances comparison (default: weka.core.EuclideanDistance).

- **canopyT1** -- The T1 distance to use when using canopy clustering. Values < 0 are taken as a positive multiplier for the T2 distance

- **fastDistanceCalc** -- Uses cut-off values for speeding up distance calculation, but suppresses also the calculation and output of the within cluster sum of squared errors/sum of distances.

- **reduceNumberOfDistanceCalcsViaCanopies** -- Use canopy clustering to reduce the number of distance calculations performed by k-means

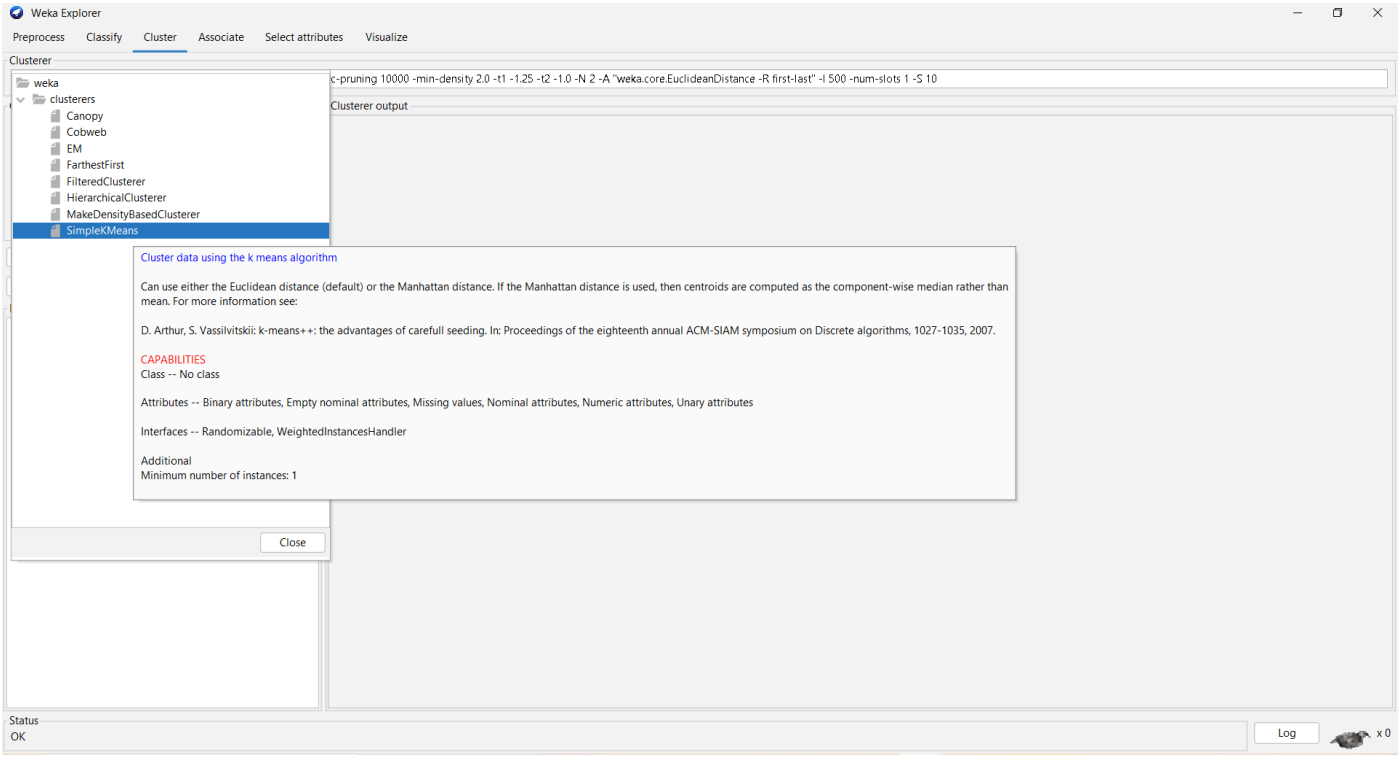
Step7: Once of the option have been specified. We run the clustering algorithm there we must make sure that they are in the 'cluster mode' panel. The use of "classes to clusters evaluation" option is selected.

Step8: Now we click 'start' button. The result window shows the centroid of each cluster as well as statistics on the number and the percent of instances assigned to different clusters.

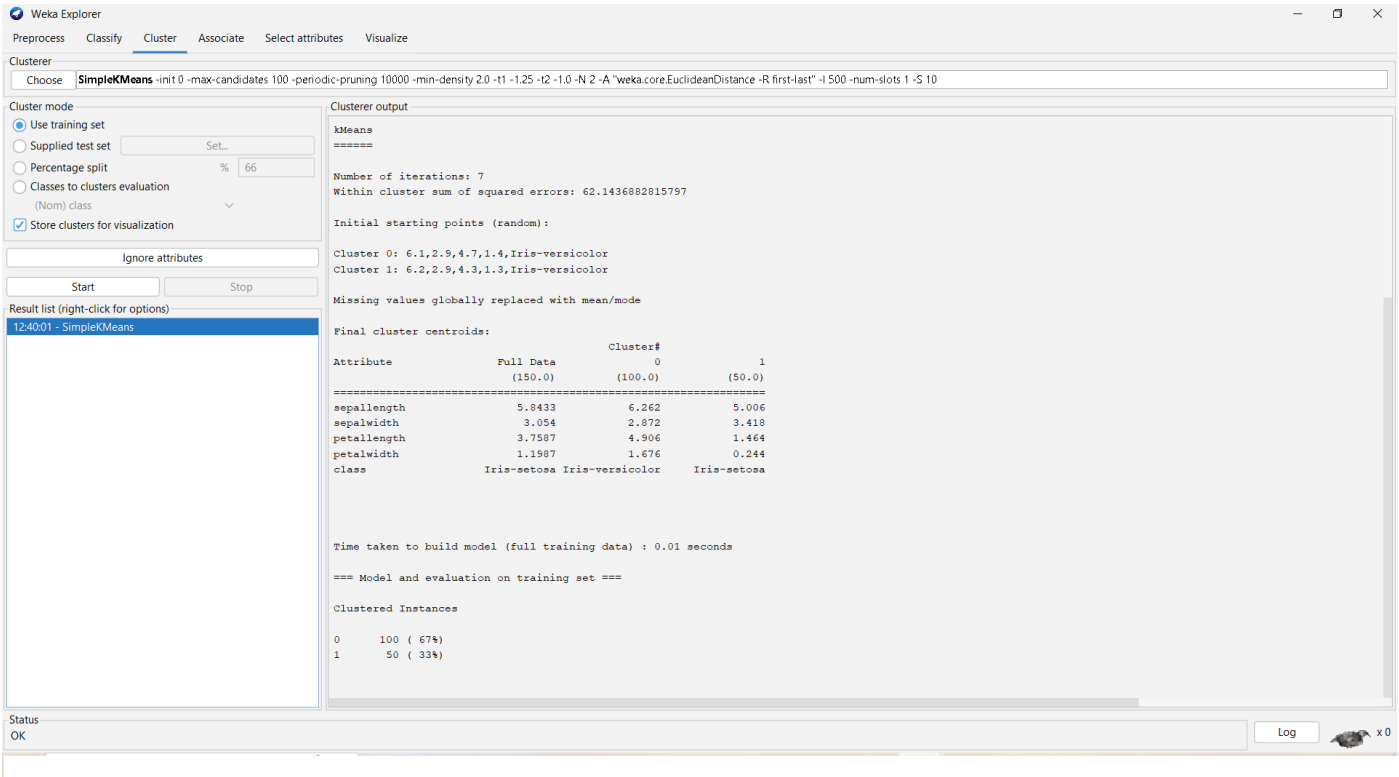
Step9: Another way of understanding characterstics of each cluster through visualization ,we can do this, try right clicking the result set on the result. List panel and selecting the visualize cluster assignments.

Results:

The following Screen shows selection of 'simple k-means clustering algorithm':



The following Screen shows the result after applying 'simple k-means clustering algorithm



b) Explore other clustering techniques available in Weka.

Procedure: HierarchicalCluster

Step1: Open weka and then go to Explorer interface

Step2: Loading the data iris.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4: Next In order to perform clustering select the 'cluster' tab in the explorer

Step5: Now click on the choose button. This step results in a dropdown list of available clustering algorithms. In this case we select 'weka.clusterers.HierarchicalClusterer'.

Step6: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button.

Options in HierarchicalCluster:

- printNewick -- Flag to indicate whether the cluster should be print in Newick format. This can be useful for display in other programs. However, for large datasets a lot of text may be produced, which may not be a nuisance when the Newick format is not required

- debug -- If set to true, clusterer may output additional info to the console.

- numClusters -- Sets the number of clusters. If a single hierarchy is desired, set this to 1.

- doNotCheckCapabilities -- If set, clusterer capabilities are not checked before clusterer is built (Use with caution to reduce runtime).

- linkType -- Sets the method used to measure the distance between two clusters.
  - SINGLE: find single link distance aka minimum link, which is the closest distance between any item in cluster1 and any item in cluster2
  - COMPLETE: find complete link distance aka maximum link, which is the largest distance between any item in cluster1 and any item in cluster2
  - ADJCOMPLETE: as COMPLETE, but with adjustment, which is the largest within cluster distance
  - AVERAGE: finds average distance between the elements of the two clusters
  - MEAN: calculates the mean distance of a merged cluster (akak Group-average agglomerative clustering)
  - CENTROID: finds the distance of the centroids of the clusters
  - WARD: finds the distance of the change in caused by merging the cluster. The information of a cluster is calculated as the error sum of squares of the centroids of the cluster and its members.
  - NEIGHBOR_JOINING use neighbor joining algorithm.

- distanceIsBranchLength -- If set to false, the distance between clusters is interpreted as the height of the node linking the clusters. This is appropriate for example for single link clustering. However, for neighbor joining, the distance is better interpreted as branch length. Set this flag to get the latter interpretation.

- distanceFunction -- Sets the distance function, which measures the distance between two individual. instances (or possibly the distance between an instance and the centroid of a cluster depending on the Link type).

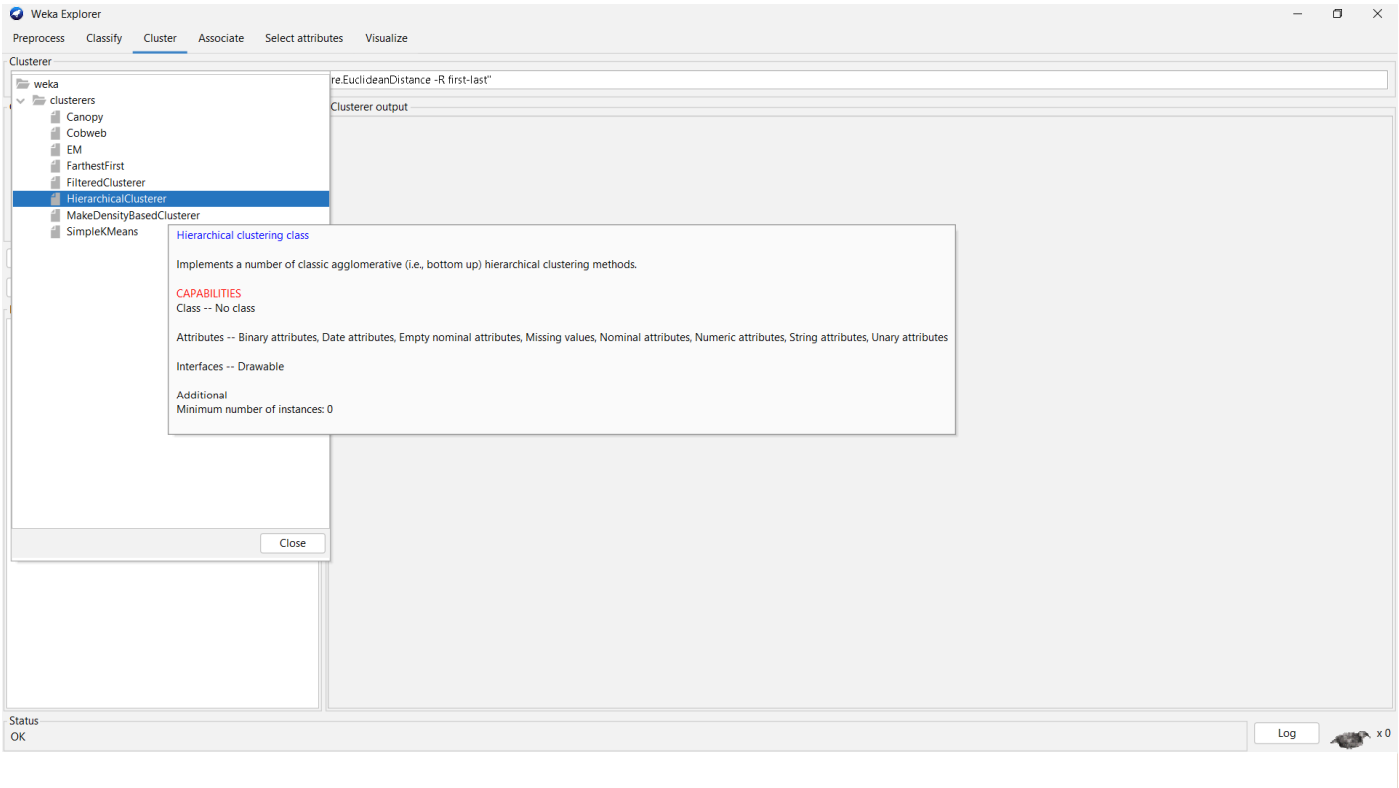
Step7: Once of the option have been specified. We run the clustering algorithm there we must make sure that they are in the 'cluster mode' panel. The use of "classes to clusters evaluation" option is selected.

Step8: Now we click 'start' button. The result window shows the centroid of each cluster as well as statistics on the number and the percent of instances assigned to different clusters.

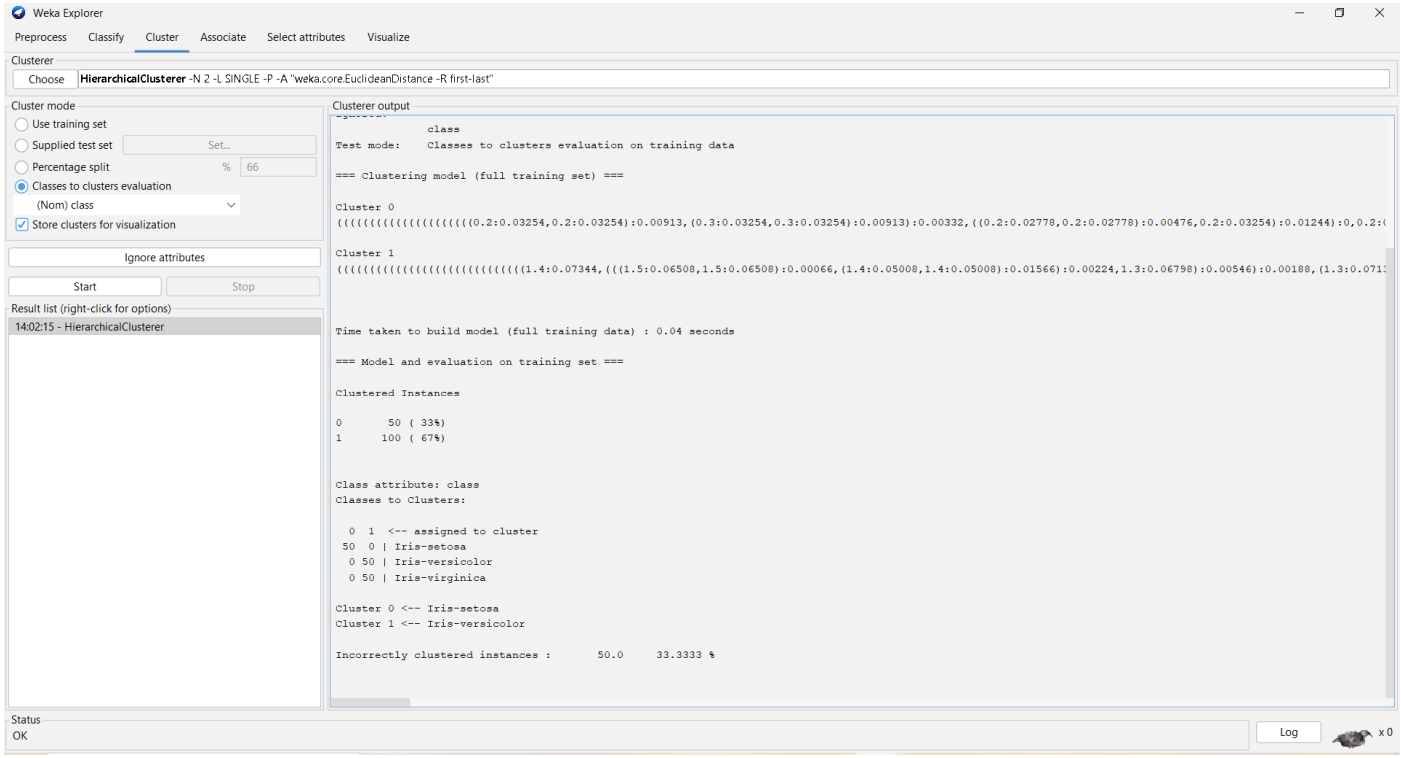
Step9: Another way of understanding characterstics of each cluster through visualization ,we can do this, try right clicking the result set on the result. List panel and selecting the visualize cluster assignments.

Results:

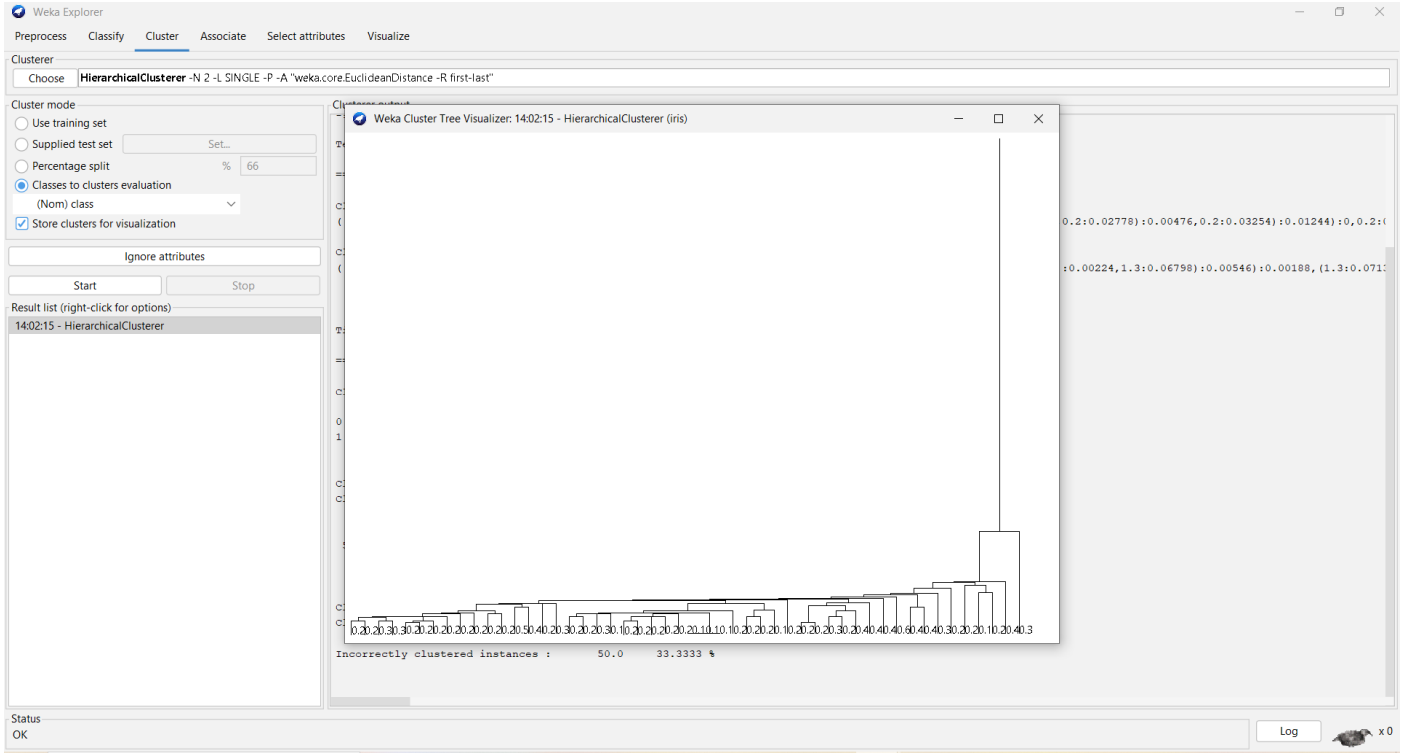
The following Screen shows selection of 'HierarchicalCluster':



The following Screen shows the result after applying 'HierarchicalCluster'



The following Screen shows the cluster tree

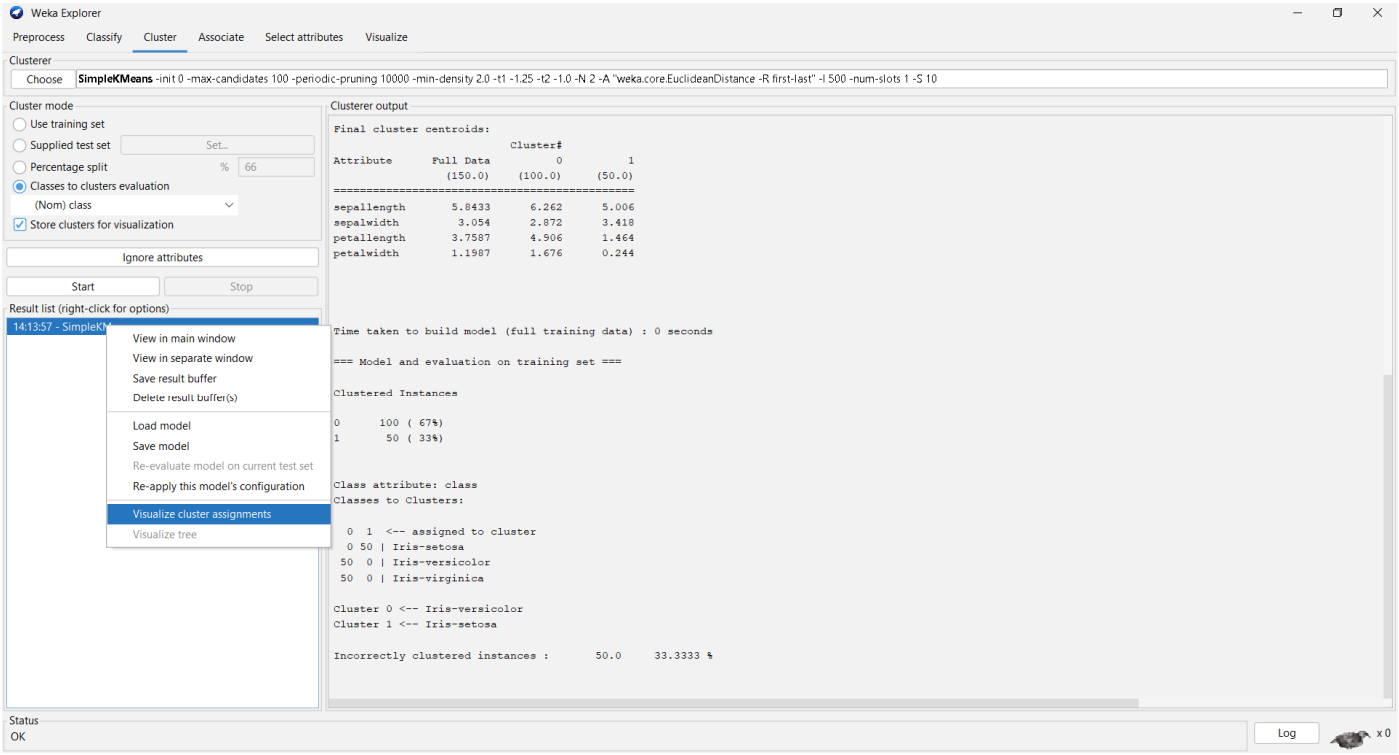


c) Explore visualization features of Weka to visualize the clusters. Derive interesting insights and explain.

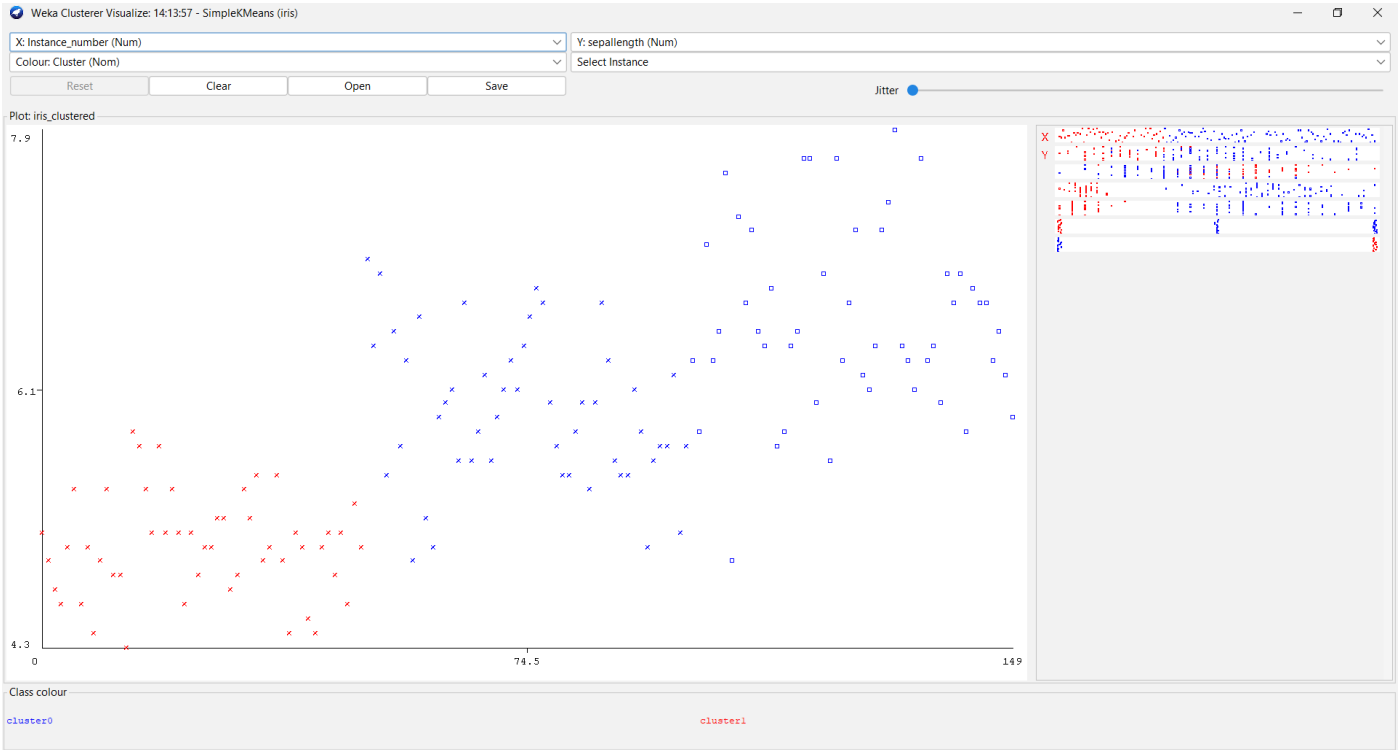
Procedure:

Another way of understanding characterstics of each cluster is through visualization ,
we can do this, try right clicking the result set on the result, List panel and selecting the visualize cluster assignments

The following Screen shows selecting visualize cluster assignments



The following Screen shows result of visualize cluster assignments



EXPERIMENT-6

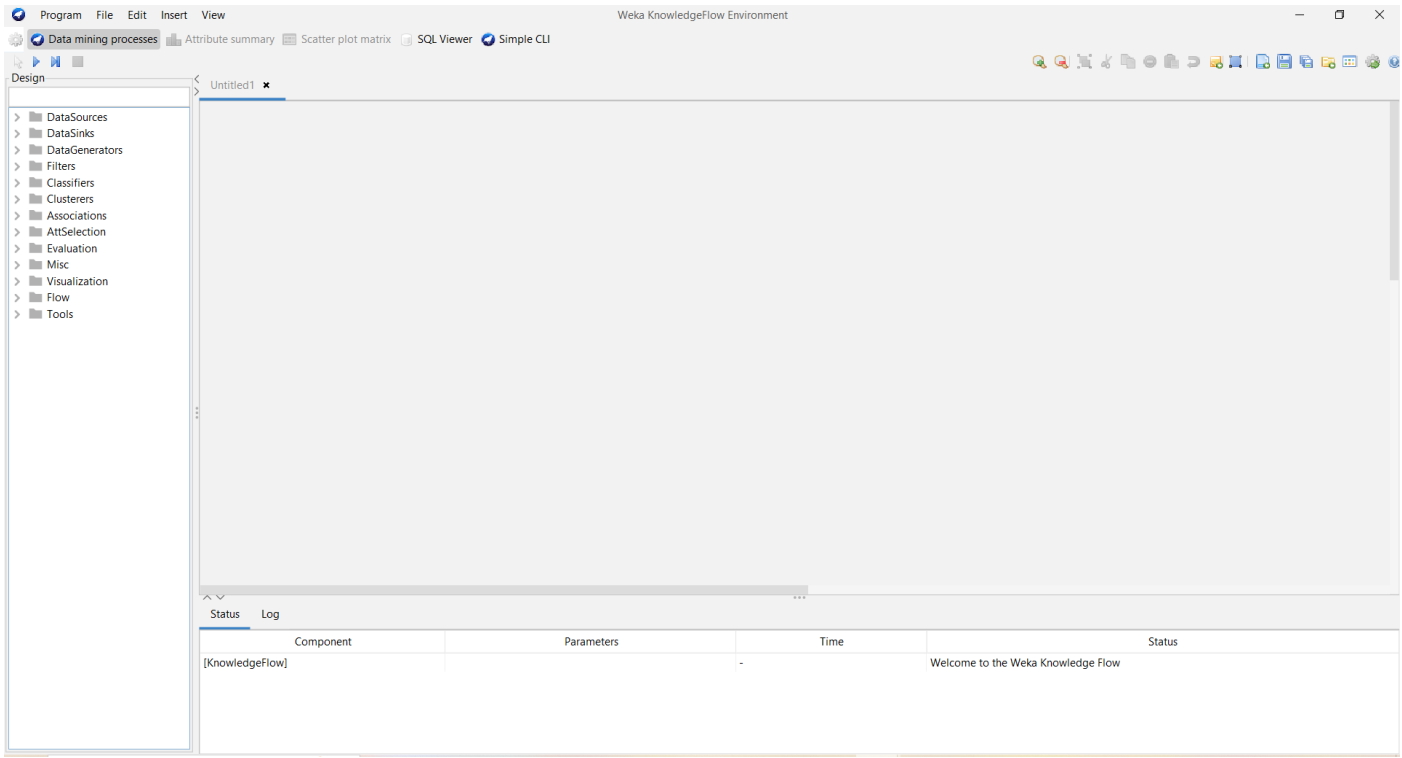
Aim: Demonstrate knowledge flow application on data sets

a) Develop a knowledge flow layout for finding strong association rules by using Apriori, FP Growth algorithms
b) Set up the knowledge flow to load an ARFF (batch mode) and perform a cross validation using J48 algorithm
c) Demonstrate plotting multiple ROC curves in the same plot window by using j48 and Random forest tree

KnowledgeFlow

- The KnowledgeFlow provides an alternative to the Explorer as a graphical front end to WEKA's core algorithms.
- The KnowledgeFlow presents a data-flow inspired interface to WEKA.
- The user can select WEKA steps from a palette, place them on a layout canvas and connect them together in order to form a knowledge flow for processing and analyzing data.
- At present, all of WEKA's classifiers, filters, clusterers, associators, loaders and savers are available in the KnowledgeFlow along with some extra tools.
- We can save the flows drawn with **.kf** extension

The following Screen shows knowledgeFlow window



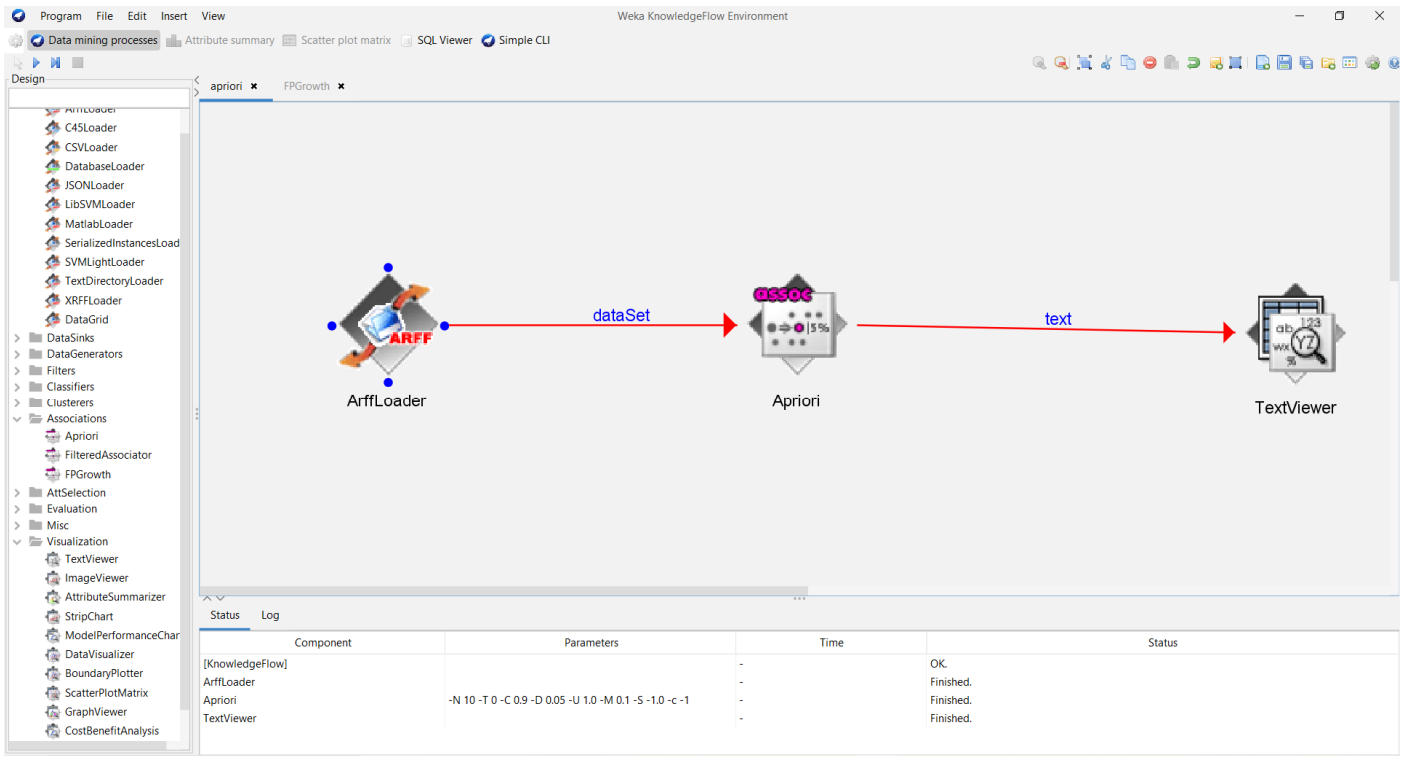
a) Develop a knowledge flow layout for finding strong association rules by using Apriori, FP Growth algorithms

Procedure: For finding strong association rules by using Apriori

- Expand the **DataSources** entry in the Design panel and choose **ArffLoader** (the mouse pointer will change to a cross hairs).
- Next place the **ArffLoader** step on the layout area by clicking somewhere on the layout (a copy of the ArffLoader icon will appear on the layout area).
- Next specify an ARFF file to load by first right clicking the mouse over the **ArffLoader icon** on the layout. A pop-up menu will appear. Select **Configure** under Edit in the list from this menu and browse to the location of your ARFF file.
- Next expand the **Association** entry and then the trees sub-entry in the Design panel and choose the **Apriori** step. Place a **Apriori** step on the layout.
- Now connect the **ArffLoader** to the **Apriori**: first right click over the ArffLoader and select the **dataSet** under Connections in the menu. A rubber band line will appear. Move the mouse over the Apriori step and left click - a red line labeled dataSet will connect the two steps.
- Next go to the **Visualization** entry and place a **TextViewer** step on the layout. Connect the **Apriori** to the **TextViewer** by selecting the **text** entry from the pop-up menu of **Apriori.**
- Now start the flow executing by pressing the **play button** on the toolbar at the top of the window. Progress information for each step in the flow will appear in the Status area and Log at the bottom of the window.
- When finished you can view the results by choosing **Show results** from the pop-up menu for the **TextViewer** step.

Results:

The following Screen shows knowledgeflow of apriori':



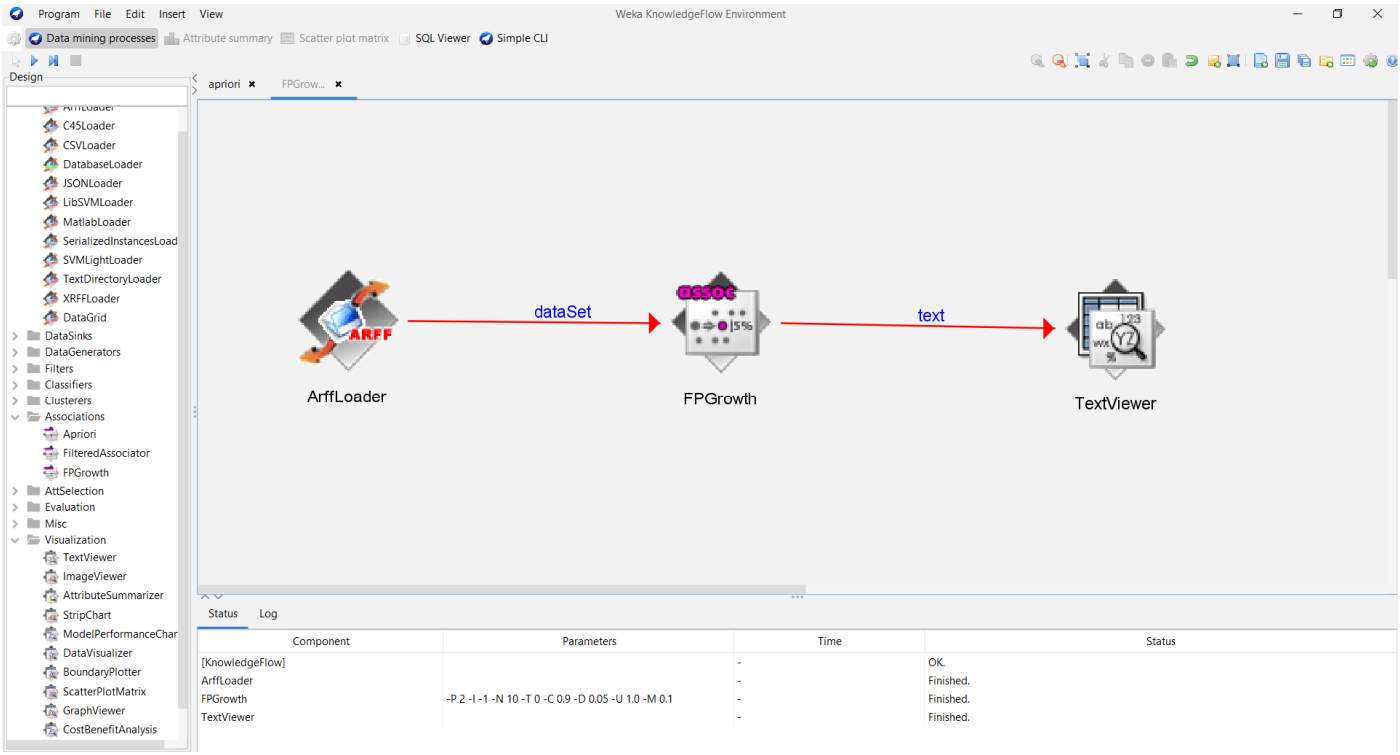
The following Screen shows result of textviewer in knowledgeflow of apriori':

Procedure: For finding strong association rules by using FP Growth algorithm

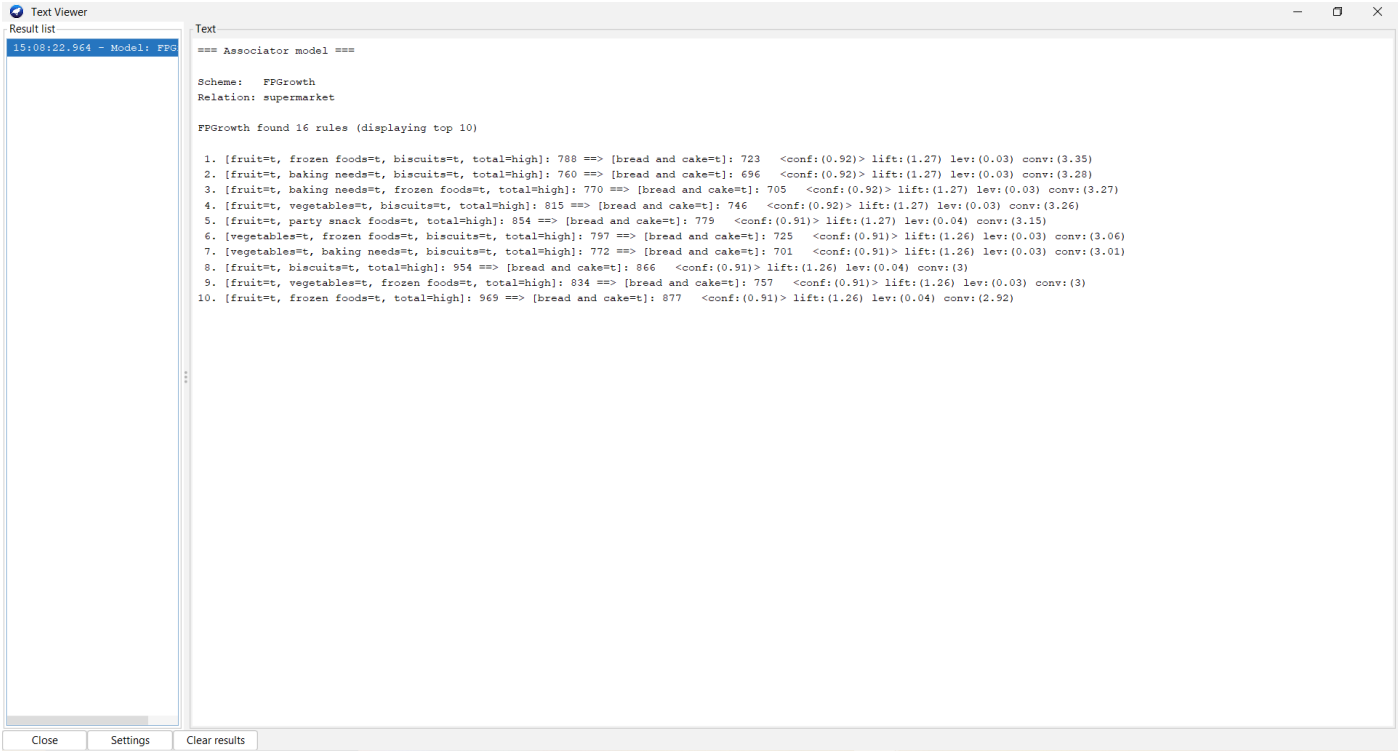
- Expand the **DataSources** entry in the Design panel and choose **ArffLoader** (the mouse pointer will change to a cross hairs).
- Next place the **ArffLoader** step on the layout area by clicking somewhere on the layout (a copy of the ArffLoader icon will appear on the layout area).
- Next specify an ARFF file to load by first right clicking the mouse over the **ArffLoader icon** on the layout. A pop-up menu will appear. Select **Configure** under Edit in the list from this menu and browse to the location of your ARFF file.
- Next expand the **Association** entry and then the trees sub-entry in the Design panel and choose the **FPGrowth** step. Place a **FPGrowth** step on the layout.
- Now connect the **ArffLoader** to the **FPGrowth**: first right click over the ArffLoader and select the **dataSet** under Connections in the menu. A rubber band line will appear. Move the mouse over the FPGrowth step and left click - a red line labeled dataSet will connect the two steps.
- Next go to the **Visualization** entry and place a **TextViewer** step on the layout. Connect the FPGrowth to the TextViewer by selecting the **text** entry from the pop-up menu of FPGrowth.
- Now start the flow executing by pressing the **play button** on the toolbar at the top of the window. Progress information for each step in the flow will appear in the Status area and Log at the bottom of the window.
- When finished you can view the results by choosing **Show results** from the pop-up menu for the TextViewer step.

Results:

The following Screen shows knowledgeflow of FPGrowth':



The following Screen shows result of textviewer in knowledgeflow of FPGrowth':



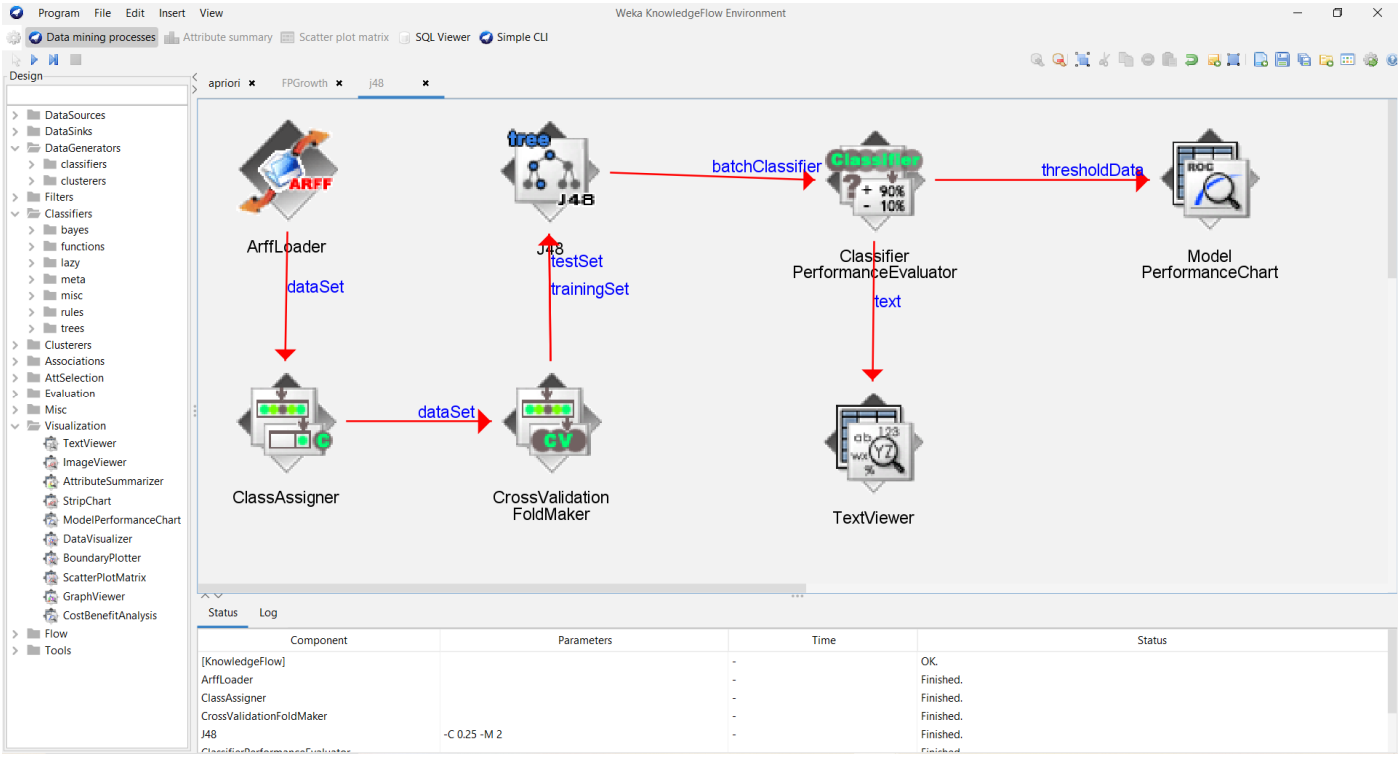
b) Set up the knowledge flow to load an ARFF (batch mode) and perform a cross validation using J48 algorithm

Procedure:

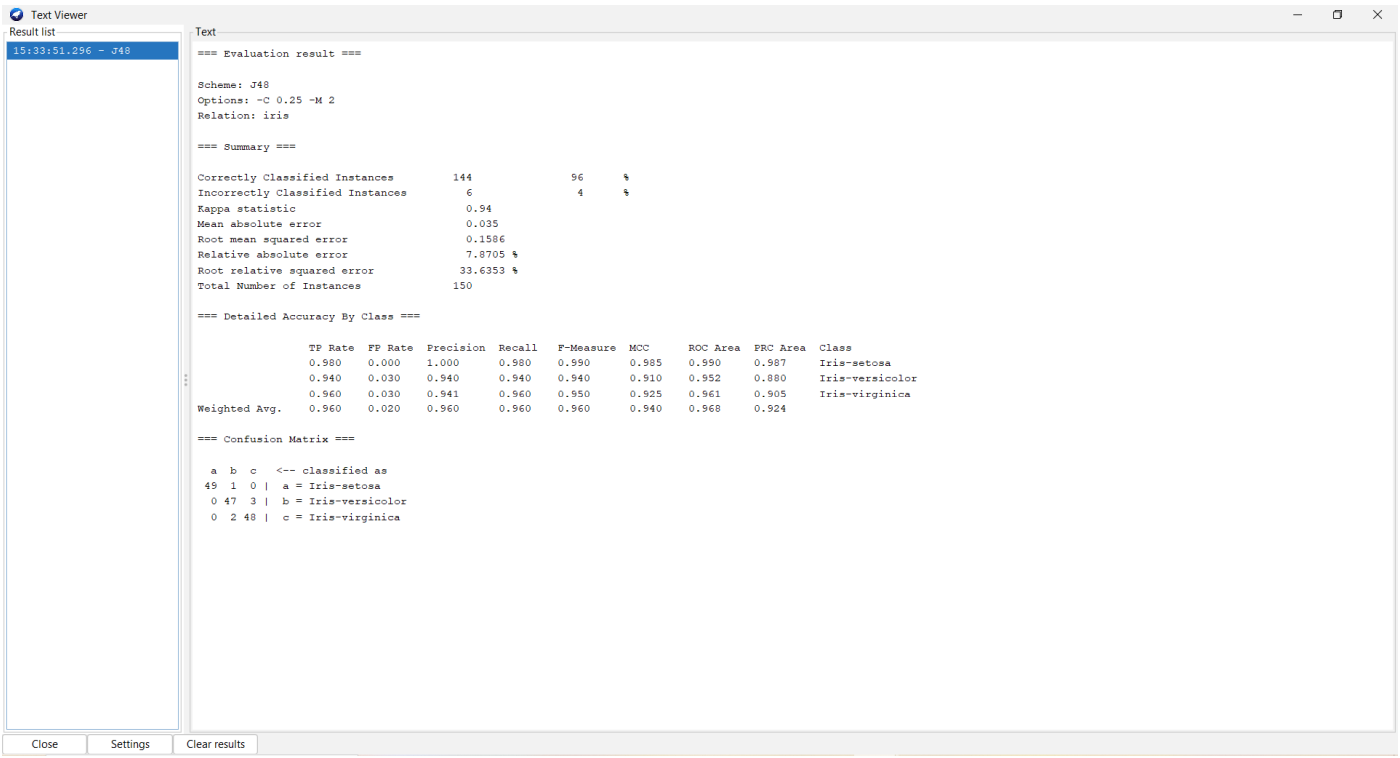
- Expand the **DataSources** entry in the Design panel and choose **ArffLoader** (the mouse pointer will change to a cross hairs).
- Next place the ArffLoader step on the layout area by clicking somewhere on the layout (a copy of the ArffLoader icon will appear on the layout area).
- Next specify an ARFF file to load by first right clicking the mouse over the **ArffLoader icon** on the layout. A pop-up menu will appear. Select **Configure** under Edit in the list from this menu and browse to the location of your ARFF file.
- Next click expand the **Evaluation** entry in the Design panel and choose the **ClassAssigner** (allows you to choose which column to be the class) step from the toolbar. Place this on the layout.
- Now connect the **ArffLoader** to the **ClassAssigner**: first right click over the ArffLoader and select the **dataSet** under Connections in the menu. A rubber band line will appear. Move the mouse over the ClassAssigner step and left click - a red line labeled dataSet will connect the two steps.
- Next right click over the ClassAssigner and choose Configure from the menu. This will pop up a window from which you can specify which column is the class in your data (last is the default).
- Next grab a **CrossValidationFoldMaker** step from the **Evaluation** entry in the Design panel and place it on the layout. Connect the ClassAssigner to the CrossValidationFoldMaker by right clicking over ClassAssigner and selecting **dataSet** from under Connections in the menu.
- Next expand the **Classifiers** entry and then the trees sub-entry in the Design panel and choose the **J48** step. Place a J48 step on the layout.
- Connect the **CrossValidationFoldMaker** to **J48** TWICE by first choosing **trainingSet** and then **testSet** from the pop-up menu for the CrossValidationFoldMaker.
- Next go back to the **Evaluation** entry and place a **ClassifierPerformanceEvaluator** step on the layout. Connect J48 to this step by selecting the **batchClassifier** entry from the pop-up menu for J48.
- Next go to the **Visualizatio**n entry and place a **TextViewer** step on the layout. Connect the ClassifierPerformanceEvaluator to the TextViewer by selecting the **text** entry from the pop-up menu for ClassifierPerformanceEvaluator.
- Now start the flow executing by pressing the **play button** on the toolbar at the top of the window. Progress information for each step in the flow will appear in the **Status area and Log** at the bottom of the window.
- When finished you can view the results by choosing **Show results** from the pop-up menu for the **TextViewer** step.
- Other cool things to add to this flow: connect a TextViewer and/or a GraphViewer to J48 in order to view the textual or graphical representations of the trees produced for each fold of the cross validation

Results:

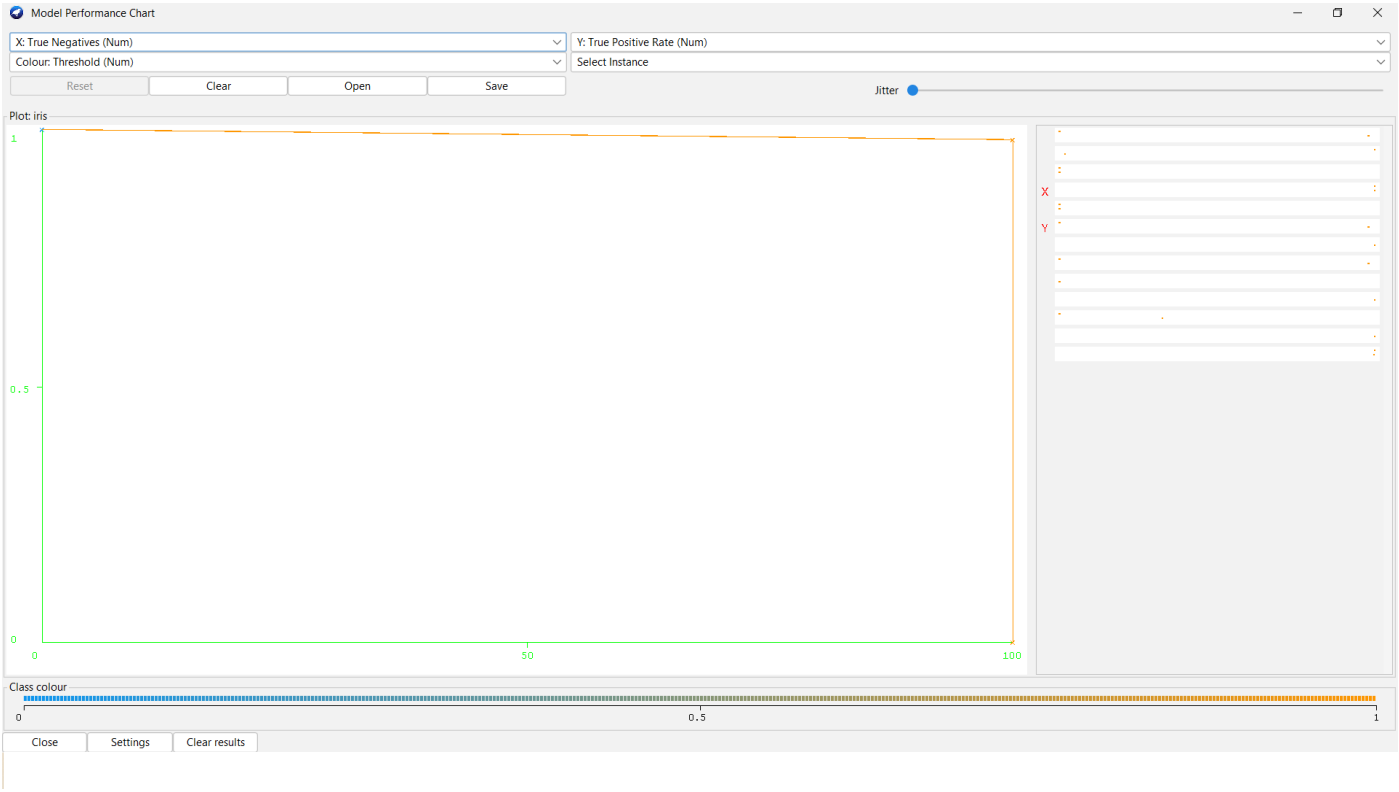
The following Screen shows knowledgeflow of J48':



The following Screen shows result of textviewer in knowledgeflow of J48':



The following Screen shows result of model performance chart in knowledgeflow of J48':



c) Demonstrate plotting multiple ROC curves in the same plot window by using j48 and Random forest tree

Procedure:

- Expand the **DataSources** entry in the Design panel and choose **ArffLoader** (the mouse pointer will change to a cross hairs).
- Next place the ArffLoader step on the layout area by clicking somewhere on the layout (a copy of the ArffLoader icon will appear on the layout area).
- Next specify an ARFF file to load by first right clicking the mouse over the **ArffLoader icon** on the layout. A pop-up menu will appear. Select **Configure** under Edit in the list from this menu and browse to the location of your ARFF file.
- Next click expand the **Evaluation** entry in the Design panel and choose the **ClassAssigne**r (allows you to choose which column to be the class) step from the toolbar. Place this on the layout.
- Now connect the **ArffLoader** to the **ClassAssigne**r: first right click over the ArffLoader and select the **dataSet** under Connections in the menu. A rubber band line will appear. Move the mouse over the ClassAssigner step and left click - a red line labeled dataSet will connect the two steps.
- Next right click over the **ClassAssigner** and choose **Configure** from the menu. This will pop up a window from which you can specify which column is the class in your data (last is the default).
- Next grab a **CrossValidationFoldMaker** step from the **Evaluation** entry in the Design panel and place it on the layout. Connect the ClassAssigner to the CrossValidationFoldMaker by right clicking over ClassAssigner and selecting **dataSet** from under Connections in the menu.
- Next expand the **Classifiers** entry and then the trees sub-entry in the Design panel and choose the **J48** step. Place a J48 step on the layout.
- Connect the **CrossValidationFoldMaker** to **J48** TWICE by first choosing **trainingSet** and then **testSet** from the pop-up menu for the CrossValidationFoldMaker.
- Next go back to the **Evaluation** entry and place a **ClassifierPerformanceEvaluator** step on the layout. Connect J48 to this step by selecting the **batchClassifier** entry from the pop-up menu for J48.
- Next go to the **Visualization** entry and place a **TextViewer** step on the layout. Connect the ClassifierPerformanceEvaluator to the TextViewer by selecting the **text** entry from the pop-up menu for ClassifierPerformanceEvaluator.
- Next expand the **Classifiers** entry and then the trees sub-entry in the Design panel and choose the **Random forest** step. Place a **Random forest** step on the layout.
- Connect the **CrossValidationFoldMaker** to **Random forest** TWICE by first choosing **trainingSet** and then **testSet** from the pop-up menu for the CrossValidationFoldMaker.
- Next go back to the Evaluation entry and place a ClassifierPerformanceEvaluator step on the layout. Connect **Random forest** to this step by selecting the **batchClassifier** entry from the pop-up menu for **Random forest**.

- Next go to the **Visualization** entry and place a **TextViewer** step on the layout. Connect the ClassifierPerformanceEvaluator to the TextViewer by selecting the **text** entry from the pop-up menu for ClassifierPerformanceEvaluator.

- Next go to the **Visualization** entry and place a **ModelPerformanceChart** step on the layout. Connect the ClassifierPerformanceEvaluator of both **J48** & **Random forest** to the ModelPerformanceChart by selecting the **Threshold data** entry from the pop-up menu for ClassifierPerformanceEvaluator.

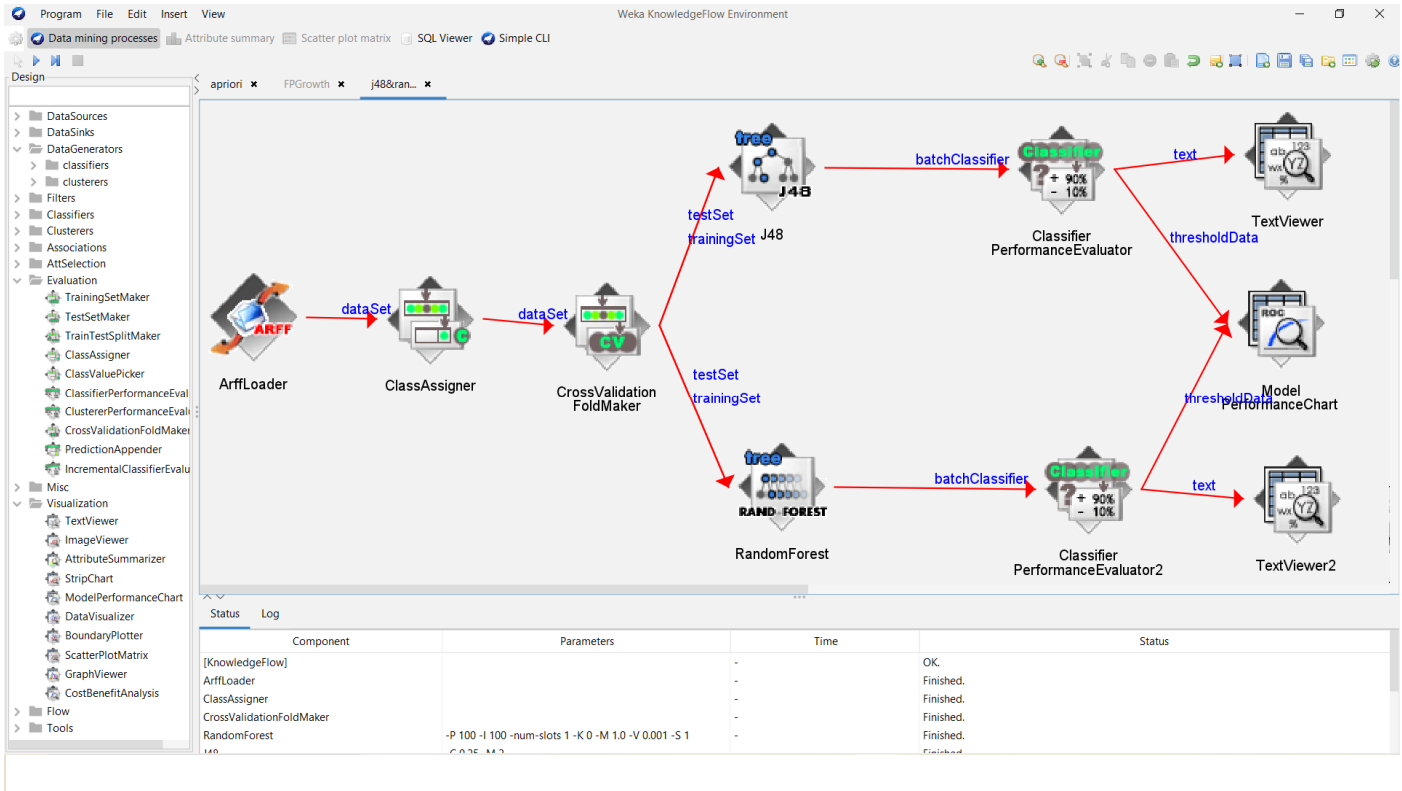
- Now start the flow executing by pressing the **play button** on the toolbar at the top of the window. Progress information for each step in the flow will appear in the **Status area** and **Log** at the bottom of the window.

- When finished you can view the results by choosing Show results from the pop-up menu for the TextViewer step.

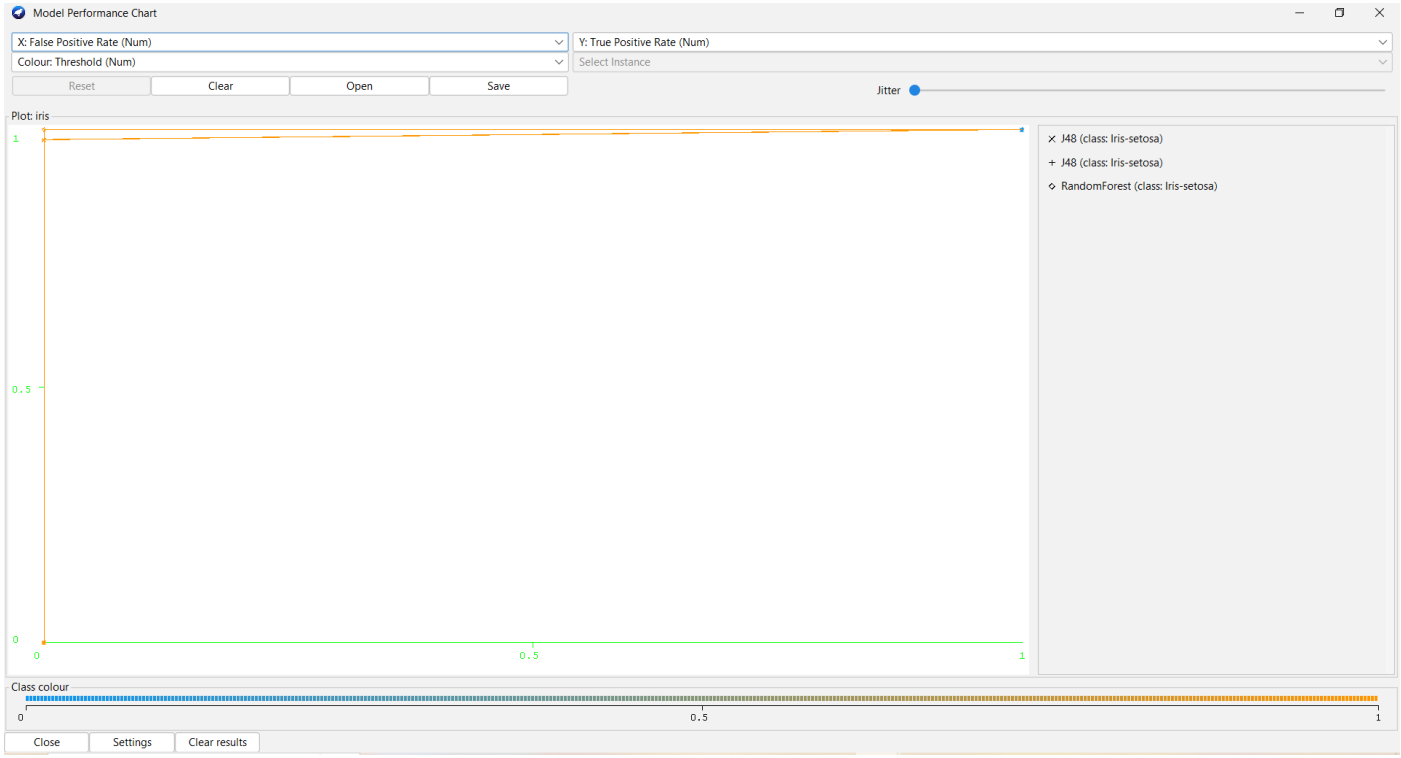
- When finished you can view the ROC curves by choosing Show chart from the pop-up menu for the ModelPerformanceChart step.

Results:

The following Screen shows knowledgeflow of J48 & Randomforest:



The following Screen shows ROC CURVES of J48 & Random Forest:



EXPERIMENT-7

Aim: Demonstrate ZeroR technique on Iris dataset (by using necessary preprocessing technique(s)) and share your observations

Procedure: ZeroR classification algorithm

Step1:Open weka and then go to Explorer interface

Step2:Loading the data iris.arff . We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step3:Once the data is loaded, weka will recognize the attributes and during the scan of the data ,weka will compute some basic strategies on each attribute. The left panel in the preprocessing window shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).

Step4: Next we select the "classify" tab and click "choose" button to select the "**ZeroR**"classifier

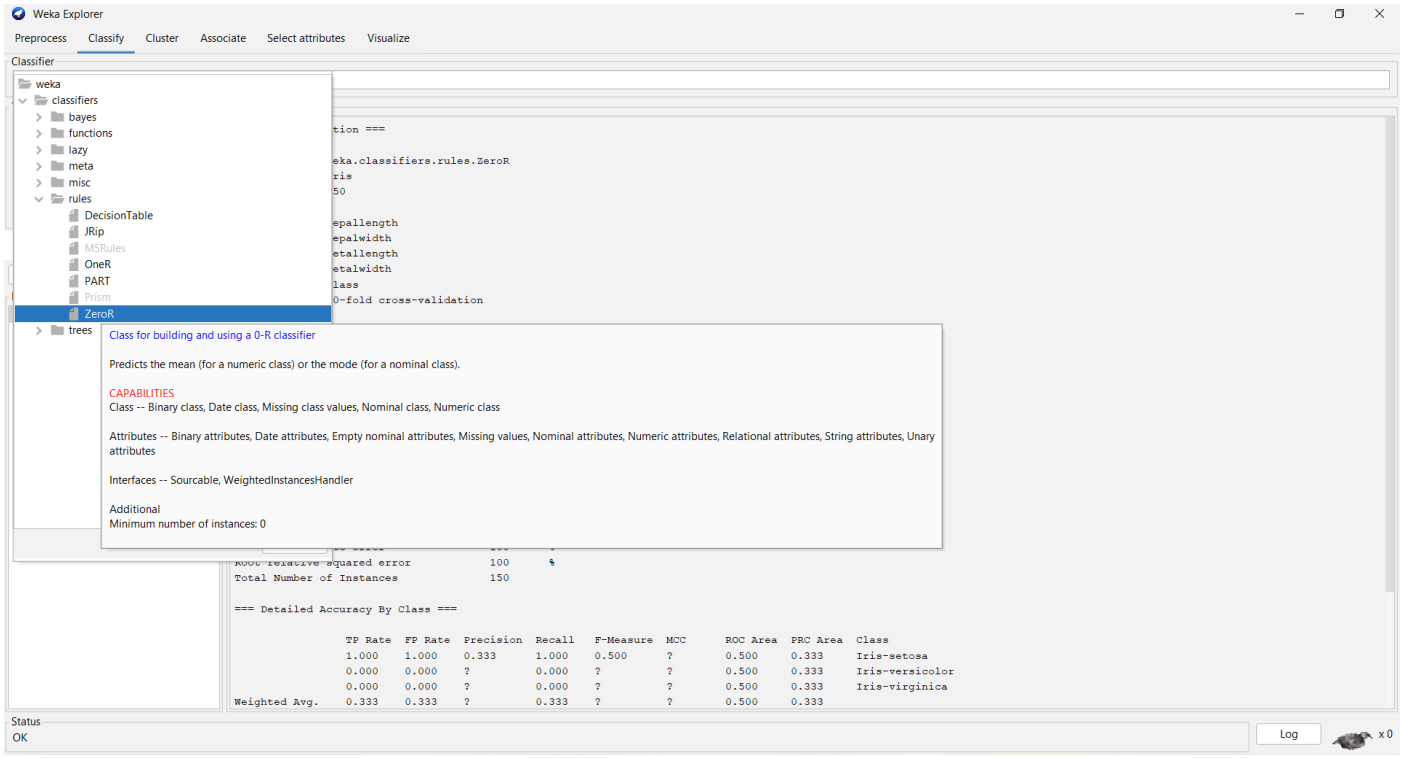
Step5: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values. The default version does perform some pruning but does not perform error pruning

Step6: Under the "text" options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don't have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

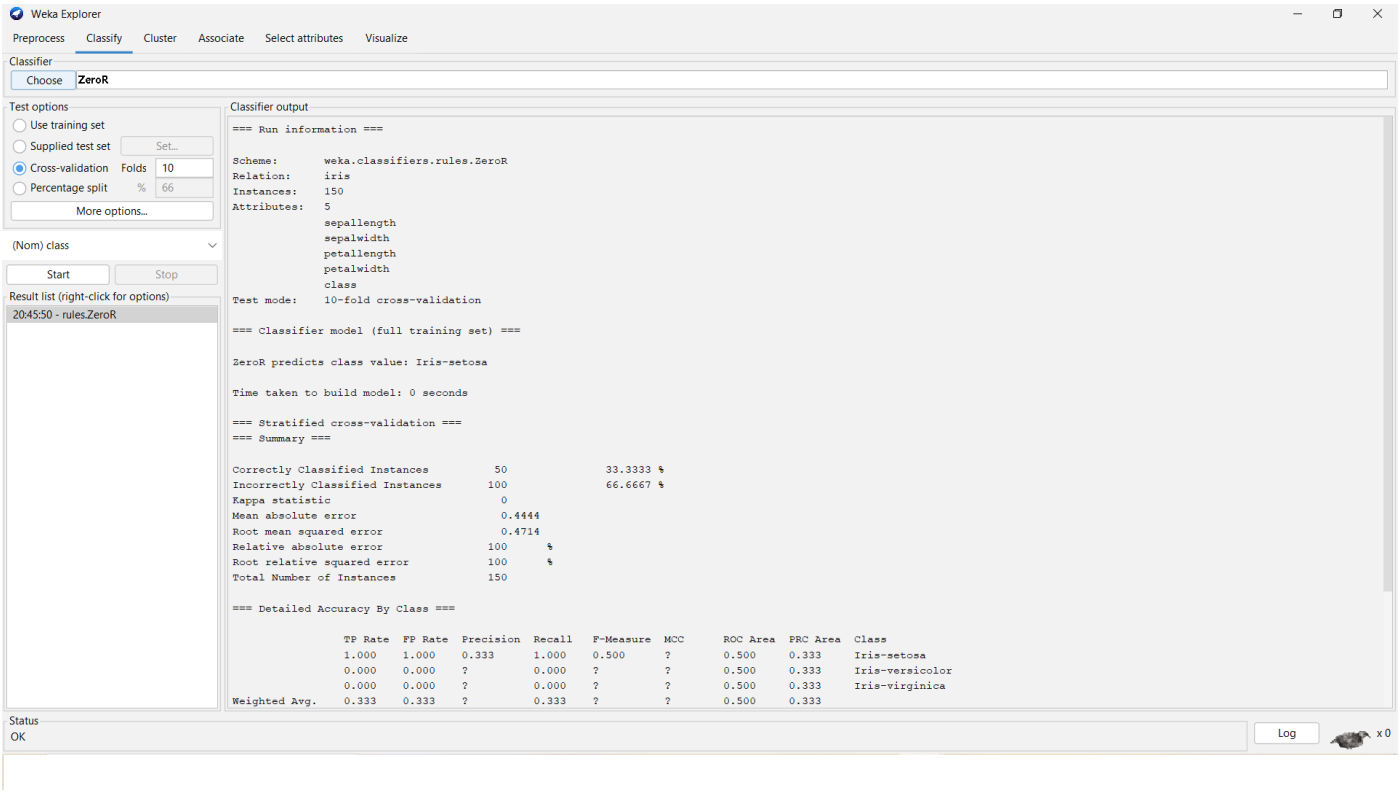
Step7: We now click "start" to generate the model .the Ascii version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

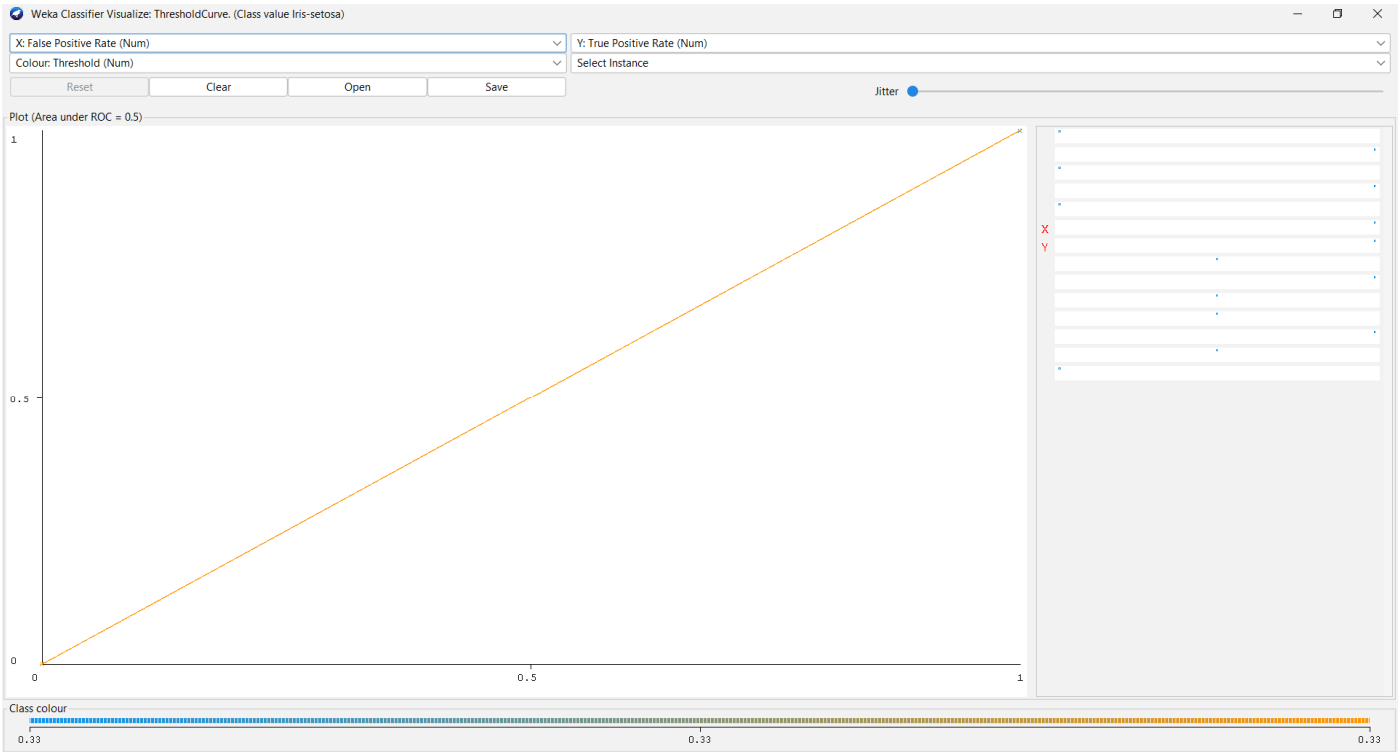
Results:

The following Screen shows selection of 'ZeroR classification':



The following Screen shows the result after applying 'ZeroR classification'





EXPERIMENT-8

Aim: Write a java program to prepare a simulated data set with unique instances.

Program:

```
import java.io.*;
public class Demo
{
	static void distinct_vals(int my_arr[], int len)
	{
		for (int i = 0; i < len; i++)
		{
			int j;
			for (j = 0; j < i; j++)
			if (my_arr[i] == my_arr[j])
				break;
			if (i == j)
				System.out.print( my_arr[i] + " ");
		}
	}
	public static void main (String[] args)
	{
		int my_arr[] = {55, 67, 99, 11, 54, 55, 88, 99, 1, 13, 45};
		int arr_len = my_arr.length;
		System.out.println("The distinct elements in the array are ");
		distinct_vals(my_arr, arr_len);
	}
}
```

Output:

```
The distinct elements in the array are
55 67 99 11 54 88 1 13 45
```

EXPERIMENT-9

Aim: Write a Python program to generate frequent item sets / association rules using Apriori algorithm

Program:

```
 from google.colab import drive
drive.mount("/content/drive/")
```

Upon executing our code, it leads us to a Google Authentication stage

```
!pip install apyori

# for importing packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from apyori import apriori


# for Read data and Display
store_data = pd.read_csv('/content/drive/MyDrive/store_data.csv', header=None)
display(store_data.head())
print(store_data.shape)
```

output:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	frozen smoothie	spinach	olive oil
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

(7501, 20)

```
# for Preprocessing on Data
#Here we need a data in form of list for Apriori Algorithm.
records = []
for i in range(1, 7501):
    records.append([str(store_data.values[i, j]) for j in range(0, 20)])
print(type(records))
```

output:

<class 'list'>

```
# for Apriori Algorithm
association_rules = apriori(records, min_support=0.0045, min_confidence=0.2,min_lift=3, min_length=2)
association_results = list(association_rules)
# for How many relation derived
print("There are {} Relation derived.".format(len(association_results)))
```

output:

There are 48 Relation derived.

```
# for Association Rules Derived
for i in range(0, len(association_results)):
    print(association_results[i][0])
```

output:

```
frozenset({'light cream', 'chicken'})
frozenset({'escalope', 'mushroom cream sauce'})
frozenset({'escalope', 'pasta'})
frozenset({'ground beef', 'herb & pepper'})
frozenset({'tomato sauce', 'ground beef'})
frozenset({'olive oil', 'whole wheat pasta'})
frozenset({'shrimp', 'pasta'})
frozenset({'light cream', 'chicken', 'nan'})
frozenset({'shrimp', 'chocolate', 'frozen vegetables'})
frozenset({'spaghetti', 'ground beef', 'cooking oil'})
frozenset({'escalope', 'nan', 'mushroom cream sauce'})
frozenset({'escalope', 'pasta', 'nan'})
frozenset({'spaghetti', 'frozen vegetables', 'ground beef'})
frozenset({'milk', 'frozen vegetables', 'olive oil'})
frozenset({'shrimp', 'mineral water', 'frozen vegetables'})
frozenset({'spaghetti', 'frozen vegetables', 'olive oil'})
frozenset({'shrimp', 'spaghetti', 'frozen vegetables'})
frozenset({'spaghetti', 'frozen vegetables', 'tomatoes'})
frozenset({'spaghetti', 'grated cheese', 'ground beef'})
frozenset({'mineral water', 'ground beef', 'herb & pepper'})
frozenset({'nan', 'ground beef', 'herb & pepper'})
frozenset({'spaghetti', 'ground beef', 'herb & pepper'})
frozenset({'milk', 'ground beef', 'olive oil'})
frozenset({'nan', 'tomato sauce', 'ground beef'})
frozenset({'shrimp', 'spaghetti', 'ground beef'})
frozenset({'milk', 'spaghetti', 'olive oil'})
frozenset({'soup', 'mineral water', 'olive oil'})
frozenset({'nan', 'olive oil', 'whole wheat pasta'})
frozenset({'shrimp', 'pasta', 'nan'})
frozenset({'pancakes', 'spaghetti', 'olive oil'})
frozenset({'shrimp', 'nan', 'chocolate', 'frozen vegetables'})
frozenset({'nan', 'ground beef', 'spaghetti', 'cooking oil'})
frozenset({'spaghetti', 'nan', 'frozen vegetables', 'ground beef'})
frozenset({'milk', 'spaghetti', 'mineral water', 'frozen vegetables'})
frozenset({'milk', 'nan', 'frozen vegetables', 'olive oil'})
```

```
frozenset({'shrimp', 'nan', 'mineral water', 'frozen vegetables'})
frozenset({'spaghetti', 'nan', 'frozen vegetables', 'olive oil'})
frozenset({'shrimp', 'nan', 'frozen vegetables', 'spaghetti'})
frozenset({'nan', 'frozen vegetables', 'spaghetti', 'tomatoes'})
frozenset({'spaghetti', 'nan', 'grated cheese', 'ground beef'})
frozenset({'nan', 'mineral water', 'ground beef', 'herb & pepper'})
frozenset({'spaghetti', 'nan', 'ground beef', 'herb & pepper'})
frozenset({'milk', 'nan', 'ground beef', 'olive oil'})
frozenset({'shrimp', 'nan', 'ground beef', 'spaghetti'})
frozenset({'milk', 'spaghetti', 'nan', 'olive oil'})
frozenset({'soup', 'mineral water', 'nan', 'olive oil'})
frozenset({'spaghetti', 'pancakes', 'nan', 'olive oil'})
frozenset({'milk', 'frozen vegetables', 'mineral water', 'nan', 'spaghetti'})
```

```
# for Rules Generated
for item in association_results:
    # first index of the inner list
    # Contains base item and add item
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])

    # second index of the inner list
    print("Support: " + str(item[1]))

    # third index of the list located at 0th
    # of the third index of the inner list

    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("====================================")
```

output:

```
Rule: light cream -> chicken
Support: 0.004533333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
====================================
Rule: escalope -> mushroom cream sauce
Support: 0.005733333333333333
Confidence: 0.30069930069930073
Lift: 3.7903273197390845
====================================
Rule: escalope -> pasta
Support: 0.005866666666666667
Confidence: 0.37288135593220345
Lift: 4.700185158809287
====================================
Rule: ground beef -> herb & pepper
Support: 0.016
Confidence: 0.3234501347708895
Lift: 3.2915549671393096
====================================
Rule: tomato sauce -> ground beef
Support: 0.005333333333333333
Confidence: 0.37735849056603776
Lift: 3.840147461662528
====================================
Rule: olive oil -> whole wheat pasta
Support: 0.008
Confidence: 0.2714932126696833
Lift: 4.130221288078346
====================================
Rule: shrimp -> pasta
Support: 0.005066666666666666
Confidence: 0.3220338983050848
```

```
Lift: 4.514493901473151
====================================
Rule: light cream -> chicken
Support: 0.004533333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
====================================
Rule: shrimp -> chocolate
Support: 0.005333333333333333
Confidence: 0.23255813953488372
Lift: 3.260160834601174
====================================
Rule: spaghetti -> ground beef
Support: 0.0048
Confidence: 0.5714285714285714
Lift: 3.281557646029315
====================================
Rule: escalope -> nan
Support: 0.005733333333333333
Confidence: 0.30069930069930073
Lift: 3.7903273197390845
====================================
Rule: escalope -> pasta
Support: 0.005866666666666667
Confidence: 0.37288135593220345
Lift: 4.700185158809287
====================================
Rule: spaghetti -> frozen vegetables
Support: 0.008666666666666666
Confidence: 0.3110047846889952
Lift: 3.164906221394116
====================================
Rule: milk -> frozen vegetables
Support: 0.0048
Confidence: 0.20338983050847456
Lift: 3.094165778526489
====================================
Rule: shrimp -> mineral water
Support: 0.0072
Confidence: 0.3068181818181818
Lift: 3.2183725365543547
====================================
Rule: spaghetti -> frozen vegetables
Support: 0.005733333333333333
Confidence: 0.20574162679425836
Lift: 3.1299436124887174
====================================
Rule: shrimp -> spaghetti
Support: 0.006
Confidence: 0.21531100478468898
Lift: 3.0183785717479763
====================================
Rule: spaghetti -> frozen vegetables
Support: 0.006666666666666667
Confidence: 0.23923444976076555
Lift: 3.497579674864993
====================================
Rule: spaghetti -> grated cheese
Support: 0.005333333333333333
Confidence: 0.3225806451612903
Lift: 3.282706701098612
====================================
Rule: mineral water -> ground beef
Support: 0.006666666666666667
Confidence: 0.390625
Lift: 3.975152645861601
====================================
Rule: nan -> ground beef
Support: 0.016
Confidence: 0.3234501347708895
Lift: 3.2915549671393096
====================================
Rule: spaghetti -> ground beef
Support: 0.0064
Confidence: 0.3934426229508197
Lift: 4.003825878061259
====================================
Rule: milk -> ground beef
Support: 0.004933333333333333
```

```
Confidence: 0.22424242424242424
Lift: 3.411395906324912
====================================
Rule: nan -> tomato sauce
Support: 0.005333333333333333
Confidence: 0.37735849056603776
Lift: 3.840147461662528
====================================
Rule: shrimp -> spaghetti
Support: 0.006
Confidence: 0.5232558139534884
Lift: 3.004914704939635
====================================
Rule: milk -> spaghetti
Support: 0.0072
Confidence: 0.20300751879699247
Lift: 3.0883496774390333
====================================
Rule: soup -> mineral water
Support: 0.0052
Confidence: 0.2254335260115607
Lift: 3.4295161157945335
====================================
Rule: nan -> olive oil
Support: 0.008
Confidence: 0.2714932126696833
Lift: 4.130221288078346
====================================
Rule: shrimp -> pasta
Support: 0.005066666666666666
Confidence: 0.3220338983050848
Lift: 4.514493901473151
====================================
Rule: pancakes -> spaghetti
Support: 0.005066666666666666
Confidence: 0.20105820105820105
Lift: 3.0586947422647217
====================================
Rule: shrimp -> nan
Support: 0.005333333333333333
Confidence: 0.23255813953488372
Lift: 3.260160834601174
====================================
Rule: nan -> ground beef
Support: 0.0048
Confidence: 0.5714285714285714
Lift: 3.281557646029315
====================================
Rule: spaghetti -> nan
Support: 0.008666666666666666
Confidence: 0.3110047846889952
Lift: 3.164906221394116
====================================
Rule: milk -> spaghetti
Support: 0.004533333333333334
Confidence: 0.28813559322033905
Lift: 3.0224013274860737
====================================
Rule: milk -> nan
Support: 0.0048
Confidence: 0.20338983050847456
Lift: 3.094165778526489
====================================
Rule: shrimp -> nan
Support: 0.0072
Confidence: 0.3068181818181818
Lift: 3.2183725365543547
====================================
Rule: spaghetti -> nan
Support: 0.005733333333333333
Confidence: 0.20574162679425836
Lift: 3.1299436124887174
====================================
Rule: shrimp -> nan
Support: 0.006
Confidence: 0.21531100478468898
Lift: 3.0183785717479763
====================================
Rule: nan -> frozen vegetables
```

```
Support: 0.006666666666666667
Confidence: 0.23923444976076555
Lift: 3.497579674864993
====================================
Rule: spaghetti -> nan
Support: 0.005333333333333333
Confidence: 0.3225806451612903
Lift: 3.282706701098612
====================================
Rule: nan -> mineral water
Support: 0.006666666666666667
Confidence: 0.390625
Lift: 3.975152645861601
====================================
Rule: spaghetti -> nan
Support: 0.0064
Confidence: 0.3934426229508197
Lift: 4.003825878061259
====================================
Rule: milk -> nan
Support: 0.004933333333333333
Confidence: 0.22424242424242424
Lift: 3.411395906324912
====================================
Rule: shrimp -> nan
Support: 0.006
Confidence: 0.5232558139534884
Lift: 3.004914704939635
====================================
Rule: milk -> spaghetti
Support: 0.0072
Confidence: 0.20300751879699247
Lift: 3.0883496774390333
====================================
Rule: soup -> mineral water
Support: 0.0052
Confidence: 0.2254335260115607
Lift: 3.4295161157945335
====================================
Rule: spaghetti -> pancakes
Support: 0.005066666666666666
Confidence: 0.20105820105820105
Lift: 3.0586947422647217
====================================
Rule: milk -> frozen vegetables
Support: 0.004533333333333334
Confidence: 0.28813559322033905
Lift: 3.0224013274860737
====================================
```

EXPERIMENT-10

Aim:Write a program to calculate chi-square value using Python. Report your observation.

Program:

```
from scipy.stats import chi2_contingency

# defining the table
data = [[207, 282, 241], [234, 242, 232]]
stat, p, dof, expected = chi2_contingency(data)

# interpret p-value
alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
  print('Dependent (reject H0)')
else:
  print('Independent (H0 holds true)')
```

Output:

```
p value is 0.1031971404730939
Independent (H0 holds true)
```

EXPERIMENT-11

Aim: Write a program of Naive Bayesian classification using Python programming language.

Program:

```
# For importing library and dataset
import pandas as pd
import numpy as np
from sklearn import datasets
wine=datasets.load_wine()
print(wine.feature_names)
```

Output:

```
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
'proanthocyanins', 'color_intensity', 'hue',
'od280/od315_of_diluted_wines', 'proline']
```

```
# For printing classes
print(wine.target_names)
```

Output:

```
['class_0' 'class_1' 'class_2']
```

```
# For viewing data
X=pd.DataFrame(wine['data'])
X.head()
```

Output:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

```
y=print(wine.target)
```

Output:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

```
# For applying naïve bayes
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(wine.data,wine.target,test_size=0.30,random_state=100)
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(X_train,y_train)
y_pred=gnb.predict(X_test)
print(y_pred)
```

Output:

```
[1 2 0 1 2 2 1 1 1 1 2 1 2 2 2 0 2 0 1 0 2 0 1 1 0 0 1 1 1 2 2 1 0 1 2 2 1
 1 2 2 0 2 2 2 0 2 2 2 0 0 0 1 0 1]
```

```
# For viewing accuracy and confusion matrix
from sklearn import metrics
print(metrics.accuracy_score(y_test,y_pred))
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
cm
```

Output:

```
1.0
array([[14,  0,  0],
       [ 0, 19,  0],
       [ 0,  0, 21]])
```

EXPERIMENT-12

Aim: Implement a Java program to perform Apriori algorithm

Program:

```
import java.util.*;
import java.sql.*;

class Tuple {
	Set<Integer> itemset;
	int support;

	Tuple() {
		itemset = new HashSet<>();
		support = -1;
	}

	Tuple(Set<Integer> s) {
		itemset = s;
		support = -1;
	}

	Tuple(Set<Integer> s, int i) {
		itemset = s;
		support = i;
	}
}

class Apriori {
	static Set<Tuple> c;
	static Set<Tuple> l;
```

```
static int d[][];
static int min_support;

public static void main(String args[]) throws Exception {
        getDatabase();
        c = new HashSet<>();
        l = new HashSet<>();
        Scanner scan = new Scanner(System.in);
        int i, j, m, n;
        System.out.println("Enter the minimum support (as an integer value):");
        min_support = scan.nextInt();
        Set<Integer> candidate_set = new HashSet<>();
        for(i=0 ; i < d.length ; i++) {
                System.out.println("Transaction Number: " + (i+1) + ":");
                for(j=0 ; j < d[i].length ; j++) {
                        System.out.print("Item number " + (j+1) + " = ");
                        System.out.println(d[i][j]);
                        candidate_set.add(d[i][j]);
                }
        }

        Iterator<Integer> iterator = candidate_set.iterator();
        while(iterator.hasNext()) {
                Set<Integer> s = new HashSet<>();
                s.add(iterator.next());
                Tuple t = new Tuple(s, count(s));
                c.add(t);
        }

        prune();
```

```
        generateFrequentItemsets();
    }

    static int count(Set<Integer> s) {
        int i, j, k;
        int support = 0;
        int count;
        boolean containsElement;
        for(i=0 ; i < d.length ; i++) {
            count = 0;
            Iterator<Integer> iterator = s.iterator();
            while(iterator.hasNext()) {
                int element = iterator.next();
                containsElement = false;
                for(k=0 ; k < d[i].length ; k++) {
                    if(element == d[i][k]) {
                        containsElement = true;
                        count++;
                        break;
                    }
                }
                if(!containsElement) {
                    break;
                }
            }
            if(count == s.size()) {
                support++;
            }
        }
        return support;
```

```
}

static void prune() {
        l.clear();
        Iterator<Tuple> iterator = c.iterator();
        while(iterator.hasNext()) {
                Tuple t = iterator.next();
                if(t.support >= min_support) {
                        l.add(t);
                }
        }
        System.out.println("-+- L -+-");
        for(Tuple t : l) {
                System.out.println(t.itemset + " : " + t.support);
        }
}

static void generateFrequentItemsets() {
        boolean toBeContinued = true;
        int element = 0;
        int size = 1;
        Set<Set> candidate_set = new HashSet<>();
        while(toBeContinued) {
                candidate_set.clear();
                c.clear();
                Iterator<Tuple> iterator = l.iterator();
                while(iterator.hasNext()) {
                        Tuple t1 = iterator.next();
                        Set<Integer> temp = t1.itemset;
                        Iterator<Tuple> it2 = l.iterator();
```

```
                while(it2.hasNext()) {
                        Tuple t2 = it2.next();
                        Iterator<Integer> it3 = t2.itemset.iterator();
                        while(it3.hasNext()) {
                                try {
                                        element = it3.next();
                                } catch(ConcurrentModificationException e) {
                                        // Sometimes this Exception gets thrown, so simply
break in that case.
                                        break;
                                }
                                temp.add(element);
                                if(temp.size() != size) {
                                        Integer[] int_arr = temp.toArray(new Integer[0]);
                                        Set<Integer> temp2 = new HashSet<>();
                                        for(Integer x : int_arr) {
                                                temp2.add(x);
                                        }
                                        candidate_set.add(temp2);
                                        temp.remove(element);
                                }
                        }
                }
        }
        Iterator<Set> candidate_set_iterator = candidate_set.iterator();
        while(candidate_set_iterator.hasNext()) {
                Set s = candidate_set_iterator.next();
                // These lines cause warnings, as the candidate_set Set stores a raw set.
                c.add(new Tuple(s, count(s)));
        }
```

```
                prune();
                if(l.size() <= 1) {
                        toBeContinued = false;
                }
                size++;
        }
        System.out.println("\n=+= FINAL LIST =+=");
        for(Tuple t : l) {
                System.out.println(t.itemset + " : " + t.support);
        }
}

static void getDatabase() throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection("jdbc:odbc:DWM");
        Statement s = con.createStatement();

        ResultSet rs = s.executeQuery("SELECT * FROM apriori;");
        Map<Integer, List <Integer>> m = new HashMap<>();
        List<Integer> temp;
        while(rs.next()) {
                int list_no = Integer.parseInt(rs.getString(1));
                int object = Integer.parseInt(rs.getString(2));
                temp = m.get(list_no);
                if(temp == null) {
                        temp = new LinkedList<>();
                }
                temp.add(object);
                m.put(list_no, temp);
        }
```

```
                Set<Integer> keyset = m.keySet();
                d = new int[keyset.size()][];
                Iterator<Integer> iterator = keyset.iterator();
                int count = 0;
                while(iterator.hasNext()) {
                        temp = m.get(iterator.next());
                        Integer[] int_arr = temp.toArray(new Integer[0]);
                        d[count] = new int[int_arr.length];
                        for(int i=0 ; i < d[count].length ; i++) {
                                d[count][i] = int_arr[i].intValue();
                        }
                        count++;
                }
        }
}
```

Output:

Enter the minimum support (as a floating point value, 0<x<1):

0.5

Transaction Number: 1:

Item number 1 = 1

Item number 2 = 3

Item number 3 = 4

Transaction Number: 2:

Item number 1 = 2

Item number 2 = 3

Item number 3 = 5

Transaction Number: 3:

Item number 1 = 1

Item number 2 = 2

Item number 3 = 3

Item number 4 = 5

Transaction Number: 4:

Item number 1 = 2

Item number 2 = 5

-+- L -+-

[1] : 2

[3] : 3

[2] : 3

[5] : 3

-+- L -+-

[2, 3] : 2

[3, 5] : 2

[1, 3] : 2

[2, 5] : 3

-+- L -+-

[2, 3, 5] : 2

=+= FINAL LIST =+=

[2, 3, 5] : 2

EXPERIMENT-13

Aim: Write a program to cluster your choice of data using simple k-means algorithm using JDK

Program:

```
/*
        Simple K means creating 2 partitions with 2-dimensional dataset in JAVA
*/


import java.util.*;


class KmeansJ {


        public static void main(String args[]) {


                int dataset[][] = {
                        {2,1},
                        {5,2},
                        {2,2},
                        {4,1},
                        {4,3},
                        {7,5},
                        {3,6},
                        {5,7},
                        {1,4},
                        {4,1}
                };


                int i,j,k=2;
                int part1[][] = new int[10][2];
                int part2[][] = new int[10][2];
```

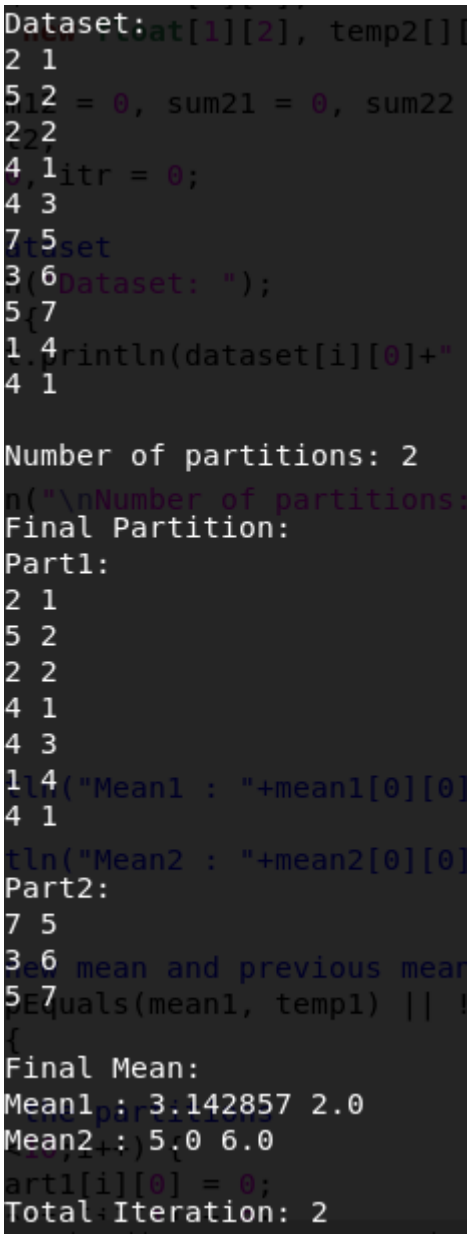
```
float mean1[][] = new float[1][2];
float mean2[][] = new float[1][2];
float temp1[][] = new float[1][2], temp2[][] = new float[1][2];
int sum11 = 0, sum12 = 0, sum21 = 0, sum22 = 0;
double dist1, dist2;
int i1 = 0, i2 = 0, itr = 0;

// Printing the dataset
System.out.println("Dataset: ");
for(i=0;i<10;i++) {
        System.out.println(dataset[i][0]+" "+dataset[i][1]);
}

System.out.println("\nNumber of partitions: "+k);

// Assuming (2,2) and (5,7) are random means
mean1[0][0] = 2;
mean1[0][1] = 2;
mean2[0][0] = 5;
mean2[0][1] = 7;

// Loop till the new mean and previous mean are same
while(!Arrays.deepEquals(mean1, temp1) || !Arrays.deepEquals(mean2, temp2)) {

        //Empting the partitions
        for(i=0;i<10;i++) {
                part1[i][0] = 0;
                part1[i][1] = 0;
                part2[i][0] = 0;
                part2[i][1] = 0;
```

```
                }

                i1 = 0; i2 = 0;

                //Finding distance between mean and data point and store the data point in the
corresponding partition
                for(i=0;i<10;i++) {
                        dist1 = Math.sqrt(Math.pow(dataset[i][0] - mean1[0][0],2) +
Math.pow(dataset[i][1] - mean1[0][1],2));
                        dist2 = Math.sqrt(Math.pow(dataset[i][0] - mean2[0][0],2) +
Math.pow(dataset[i][1] - mean2[0][1],2));

                        if(dist1 < dist2) {
                                part1[i1][0] = dataset[i][0];
                                part1[i1][1] = dataset[i][1];

                                i1++;
                        }
                        else {
                                part2[i2][0] = dataset[i][0];
                                part2[i2][1] = dataset[i][1];

                                i2++;
                        }
                }

                //Storing the previous mean
                temp1[0][0] = mean1[0][0];
                temp1[0][1] = mean1[0][1];
                temp2[0][0] = mean2[0][0];
                temp2[0][1] = mean2[0][1];
```

```
        //Finding new mean for new partitions
        sum11 = 0; sum12 = 0; sum21 = 0; sum22 = 0;

        for(i=0;i<i1;i++) {
                sum11 += part1[i][0];
                sum12 += part1[i][1];
        }
        for(i=0;i<i2;i++) {
                sum21 += part2[i][0];
                sum22 += part2[i][1];
        }
        mean1[0][0] = (float)sum11/i1;
        mean1[0][1] = (float)sum12/i1;
        mean2[0][0] = (float)sum21/i2;
        mean2[0][1] = (float)sum22/i2;

        itr++;
}

System.out.println("\nFinal Partition: ");
System.out.println("Part1:");
for(i=0;i<i1;i++) {
        System.out.println(part1[i][0]+" "+part1[i][1]);
}
System.out.println("\nPart2:");
for(i=0;i<i2;i++) {
        System.out.println(part2[i][0]+" "+part2[i][1]);
}
System.out.println("\nFinal Mean: ");
```

```
                System.out.println("Mean1 : "+mean1[0][0]+" "+mean1[0][1]);

                System.out.println("Mean2 : "+mean2[0][0]+" "+mean2[0][1]);

                System.out.println("\nTotal Iteration: "+itr);

        }

}
```

Output:



EXPERIMENT-14

Aim: Write a program of cluster analysis using simple k-means algorithm Python programming language.

Program:

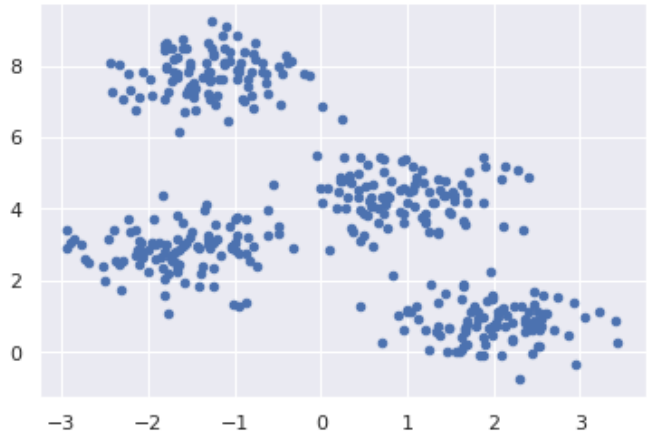
```
# For importing library and dataset
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4, cluster_std=0.60, random_state=0)
#print(X)
print(y_true)
```

Output:

```
[3 2 2 0 2 3 1 0 2 1 3 1 0 2 2 1 2 1 0 2 2 2 3 0 0 3 3 0 0 1 0 0 2 0 2 3 1
 0 1 0 2 3 1 0 1 3 1 1 1 3 3 3 2 3 3 2 0 1 1 2 1 1 1 1 0 2 1 0 2 3 2 0 2 3
 1 2 0 2 3 2 2 0 1 3 0 2 2 0 2 1 2 1 0 2 2 3 0 1 1 1 3 1 0 0 2 3 0 3 1 2 2
 2 0 2 3 1 3 2 3 3 1 2 1 3 0 3 1 1 1 0 3 3 3 2 0 2 3 0 3 2 0 2 2 3 2 1 0 2
 1 2 1 0 1 0 3 2 1 2 2 2 0 2 3 2 2 3 1 1 2 3 3 3 3 1 1 1 0 2 0 3 0 3 2 0 3
 2 3 0 0 3 0 0 3 2 3 3 3 1 1 0 0 2 1 1 3 0 2 3 0 3 0 0 1 1 3 1 0 3 1 3 2 0
 3 2 2 2 1 3 0 3 0 2 0 0 3 1 2 1 0 2 1 0 0 3 1 0 3 0 1 2 2 1 2 0 3 3 0 0 1
 3 0 2 2 2 3 0 0 0 0 0 1 0 0 3 2 1 3 1 1 1 3 1 1 3 3 3 1 0 1 0 1 2 0 1 2 0
 1 1 0 2 3 3 0 3 1 1 0 3 3 1 1 0 3 0 1 0 1 3 2 2 1 2 0 2 2 3 2 3 1 2 3 1 3
 0 2 0 2 1 0 1 3 1 1 1 1 3 1 2 1 2 1 0 3 2 1 1 0 2 2 3 3 3 2 0 1 0 2 1 2 3
 2 0 0 2 3 0 3 3 2 3 2 3 0 0 0 2 3 3 3 3 2 3 0 1 1 0 0 1 2 1]
```

```
# For view data
plt.scatter(X[:, 0], X[:, 1], s=20);
plt.show()
```

Output:



```
# For applying kmean
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
y_kmeans
```

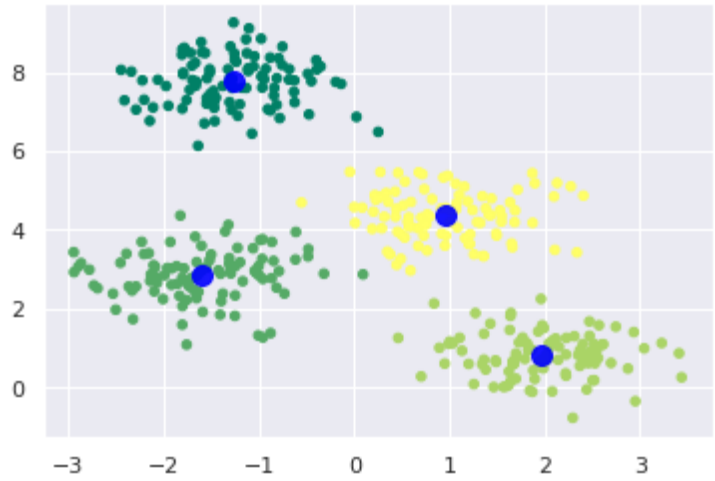
Output:

```
array([0, 1, 1, 3, 1, 0, 2, 3, 1, 2, 0, 2, 3, 1, 1, 2, 1, 2, 3, 1, 1, 1, 0, 3,
3, 0, 0, 3, 3, 2, 3, 3, 1, 3, 1, 0, 2, 3, 2, 3, 1, 0, 2, 3, 2, 0, 2, 2, 2, 0,
0, 0, 1, 0, 0, 1, 3, 2, 2, 1, 2, 2, 2, 2, 3, 1, 2, 3, 1, 0, 1, 3, 1, 0, 2, 1,
3, 1, 0, 1, 1, 3, 2, 0, 3, 1, 1, 3, 1, 2, 1, 2, 3, 1, 1, 0, 3, 2, 2, 2, 0, 2,
3, 3, 1, 0, 3, 0, 2, 1, 1, 1, 3, 1, 0, 2, 0, 1, 0, 0, 2, 1, 2, 0, 3, 0, 2, 2,
2, 3, 0, 0, 0, 1, 3, 1, 0, 3, 0, 1, 3, 1, 1, 0, 1, 2, 3, 1, 2, 1, 2, 3, 2, 3,
0, 1, 2, 1, 1, 1, 3, 1, 0, 1, 1, 0, 2, 2, 1, 0, 0, 0, 0, 2, 2, 2, 3, 1, 3, 0,
3, 0, 1, 3, 0, 1, 0, 3, 3, 0, 3, 3, 0, 1, 0, 0, 0, 2, 2, 3, 3, 1, 2, 2, 0, 3,
1, 0, 3, 0, 3, 3, 2, 2, 0, 2, 3, 0, 2, 0, 1, 3, 0, 1, 1, 1, 2, 0, 3, 0, 3, 1,
3, 3, 0, 2, 1, 2, 3, 1, 2, 3, 3, 0, 2, 3, 0, 3, 2, 1, 1, 2, 1, 3, 0, 0, 3, 3,
2, 0, 3, 1, 1, 1, 0, 3, 3, 3, 3, 3, 2, 3, 3, 0, 1, 2, 0, 2, 2, 2, 0, 2, 2, 0,
0, 0, 2, 3, 2, 3, 2, 1, 3, 2, 1, 3, 2, 2, 3, 1, 0, 0, 3, 0, 2, 2, 3, 0, 0, 2,
2, 3, 0, 3, 2, 3, 2, 0, 1, 1, 2, 1, 3, 1, 1, 0, 1, 0, 2, 1, 0, 2, 0, 3, 1, 3,
1, 2, 3, 2, 0, 2, 2, 2, 2, 0, 2, 1, 2, 1, 2, 3, 0, 1, 2, 2, 3, 1, 1, 0, 0, 0,
1, 3, 2, 3, 1, 2, 1, 0, 1, 3, 3, 1, 0, 3, 0, 0, 1, 0, 1, 0, 3, 3, 3, 1, 0, 0,
0, 0, 1, 0, 3, 2, 2, 3, 3, 2, 1, 2], dtype=int32)
```

```
# For viewing clusters with means
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=20,cmap="summer")
centers = kmeans.cluster_centers_
print(centers)
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);
plt.show()
```

Output:

```
[[-1.2689694   7.75608144]
 [-1.61366997  2.84849883]
 [ 1.95662677  0.83945671]
 [ 0.95041055  4.36874542]]
```



EXPERIMENT-15

Aim: Write a program to compute/display dissimilarity matrix (for your own dataset containing at least four instances with two attributes) using Python

Program:

```
import numpy as np
from scipy.spatial import distance_matrix
 # Create the matrices
x = np.array([[1,2],[2,1],[2,2]])
y = np.array([[5,0],[1,2],[2,0]])
# Display the matrices
print("matrix x:\n", x)
print("matrix y:\n", y)
 # compute the distance matrix
dist_mat = distance_matrix(x, y, p=2)
print("Distance Matrix:\n", dist_mat)
```

Output:

```
matrix x:
 [[1 2]
 [2 1]
 [2 2]]
matrix y:
 [[5 0]
 [1 2]
 [2 0]]

Distance Matrix:
 [[4.47213595 0.         2.23606798]
 [3.16227766 1.41421356 1.        ]
 [3.60555128 1.         2.        ]]
```

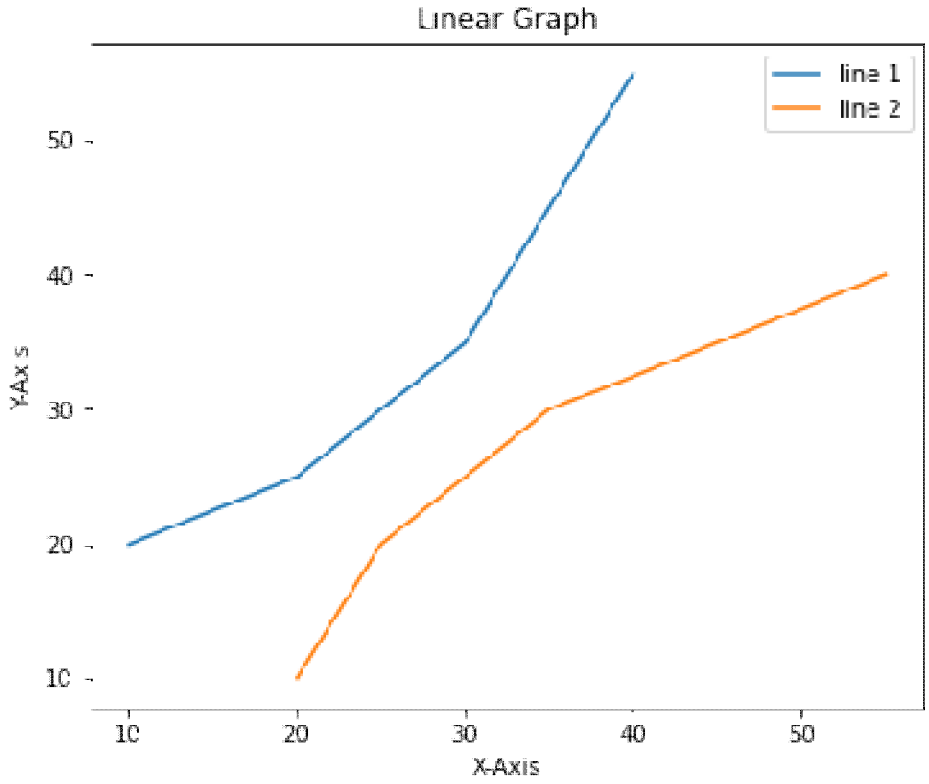
EXPERIMENT-16

Aim: Visualize the datasets using matplotlib in python.(Histogram, Box plot, Bar chart, Pie chart etc.,)

Program for Linear graph:

```
# Python program to show pyplot module
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
fig = plt.figure(figsize = (5, 4))
# Adding the axes to the figure
ax = fig.add_axes([1, 1, 1, 1])
# plotting 1st dataset to the figure
ax1 = ax.plot(x, y)
# plotting 2nd dataset to the figure
ax2 = ax.plot(y, x)
# Setting Title
ax.set_title("Linear Graph")
# Setting Label
ax.set_xlabel("X-Axis")
ax.set_ylabel("Y-Axis")
# Adding Legend
ax.legend(labels = ('line 1', 'line 2'))
plt.show()
```

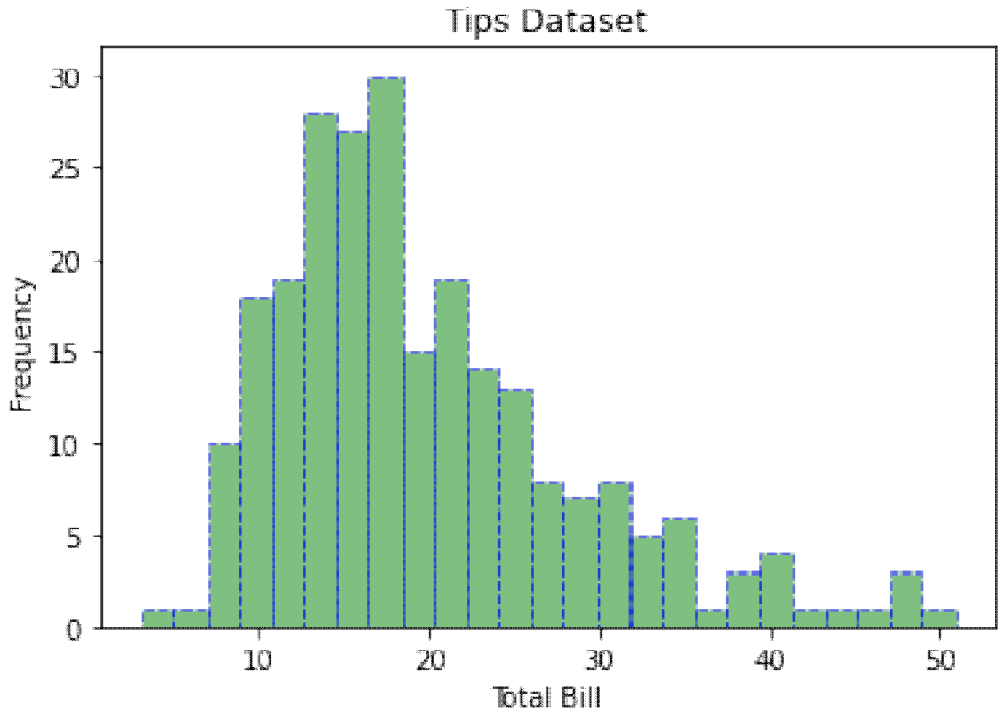
Output:



Program for Histograms:

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

# Reading the tips.csv file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/tips.csv')

# initializing the data
x = data['total_bill']

# plotting the data
plt.hist(x, bins=25, color='green', edgecolor='blue',
    linestyle='--', alpha=0.5)

# Adding title to the plot
plt.title("Tips Dataset")

# Adding label on the y-axis
plt.ylabel('Frequency')

# Adding label on the x-axis
plt.xlabel('Total Bill')

plt.show()
```

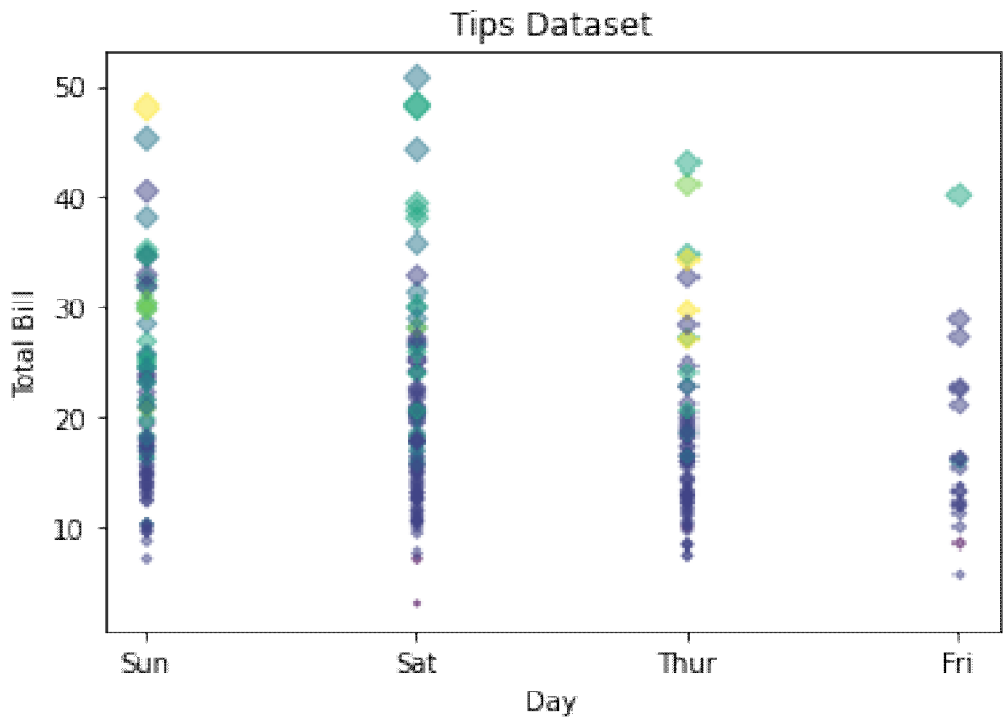
Output:



Program for Scatter Plot:

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

# Reading the tips.csv file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/tips.csv')

# initializing the data
x = data['day']
y = data['total_bill']

# plotting the data
plt.scatter(x, y, c=data['size'], s=data['total_bill'],
      marker='D', alpha=0.5)

# Adding title to the plot
plt.title("Tips Dataset")

# Adding label on the y-axis
plt.ylabel('Total Bill')

# Adding label on the x-axis
plt.xlabel('Day')

plt.show()
```

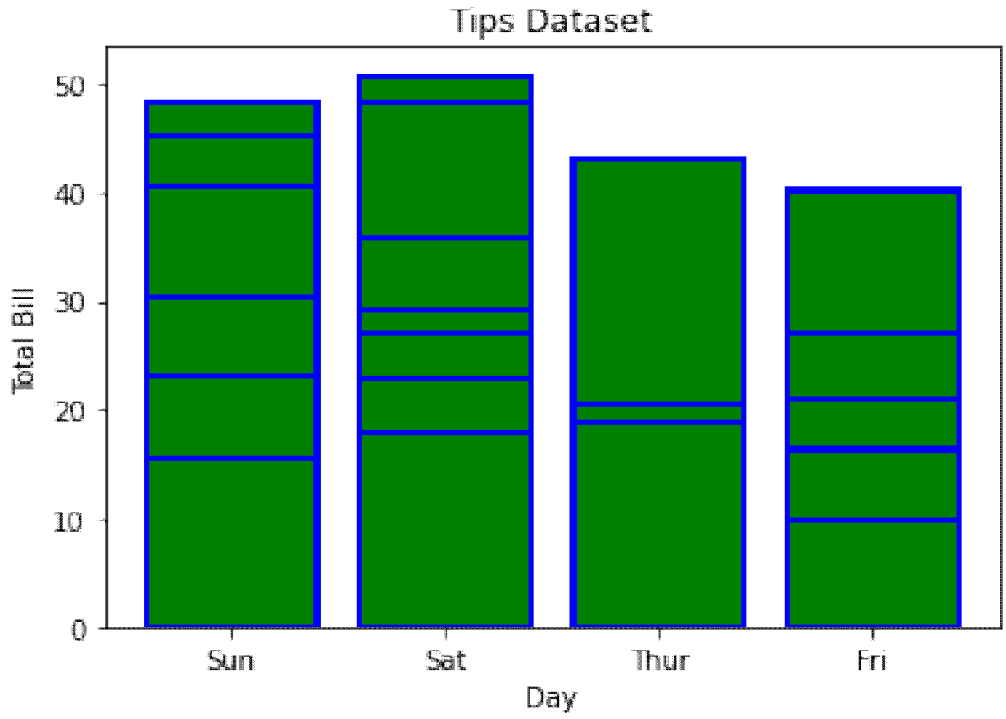
Output:



Program for Bar chart:

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

# Reading the tips.csv file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/tips.csv')

# initializing the data
x = data['day']
y = data['total_bill']

# plotting the data
plt.bar(x, y, color='green', edgecolor='blue',
    linewidth=2)

# Adding title to the plot
plt.title("Tips Dataset")

# Adding label on the y-axis
plt.ylabel('Total Bill')

# Adding label on the x-axis
plt.xlabel('Day')

plt.show()
```

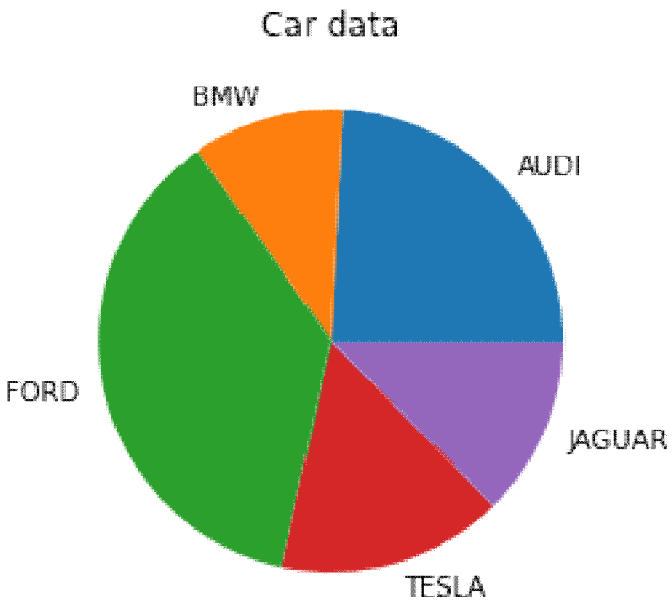
Output:



Program for pie chart:

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

# Reading the tips.csv file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/tips.csv')

# initializing the data
cars = ['AUDI', 'BMW', 'FORD',
   'TESLA', 'JAGUAR',]
data = [23, 10, 35, 15, 12]

# plotting the data
plt.pie(data, labels=cars)

# Adding title to the plot
plt.title("Car data")

plt.show()
```

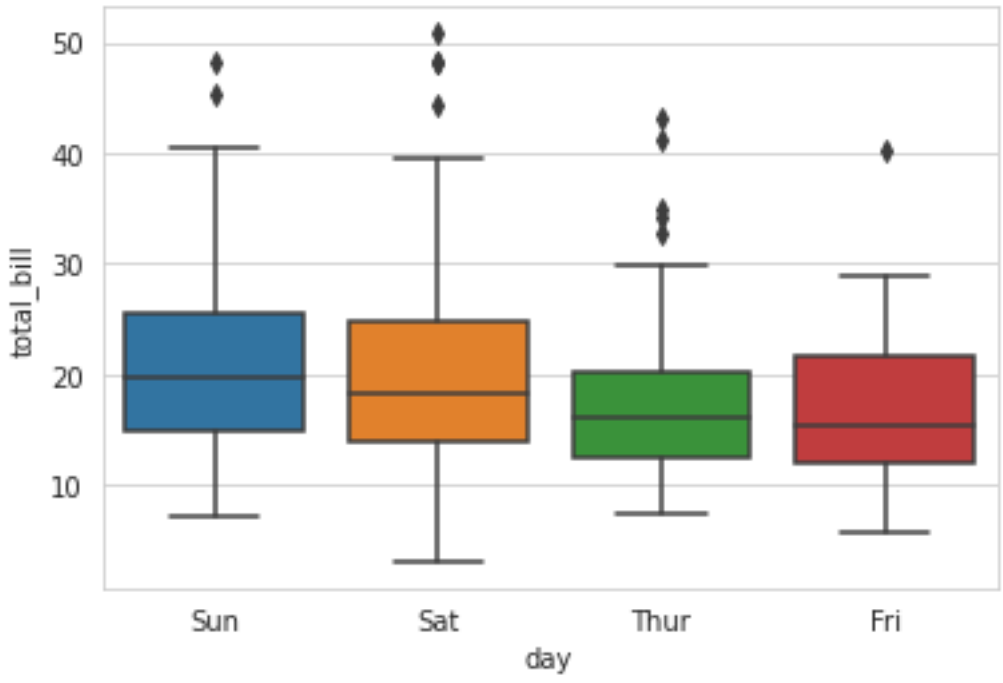
Output:



Program for Box plot:

```
# import the required library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


# load the dataset
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/tips.csv")

# display 5 rows of dataset
df.head()
# Boxplot of days with respect total_bill.
#Draw a vertical boxplot grouped
# by a categorical variable:
sns.set_style("whitegrid")

sns.boxplot(x = 'day', y = 'total_bill', data = df)
```

Output:



EXPERIMENT-17

Aim: Demonstrate performing Regression on data sets

a) **Load each dataset into Weka and build Linear Regression model. Study the clusters formed. Use Training set option. Interpret the regression model and derive patterns and conclusions from the regression results**.

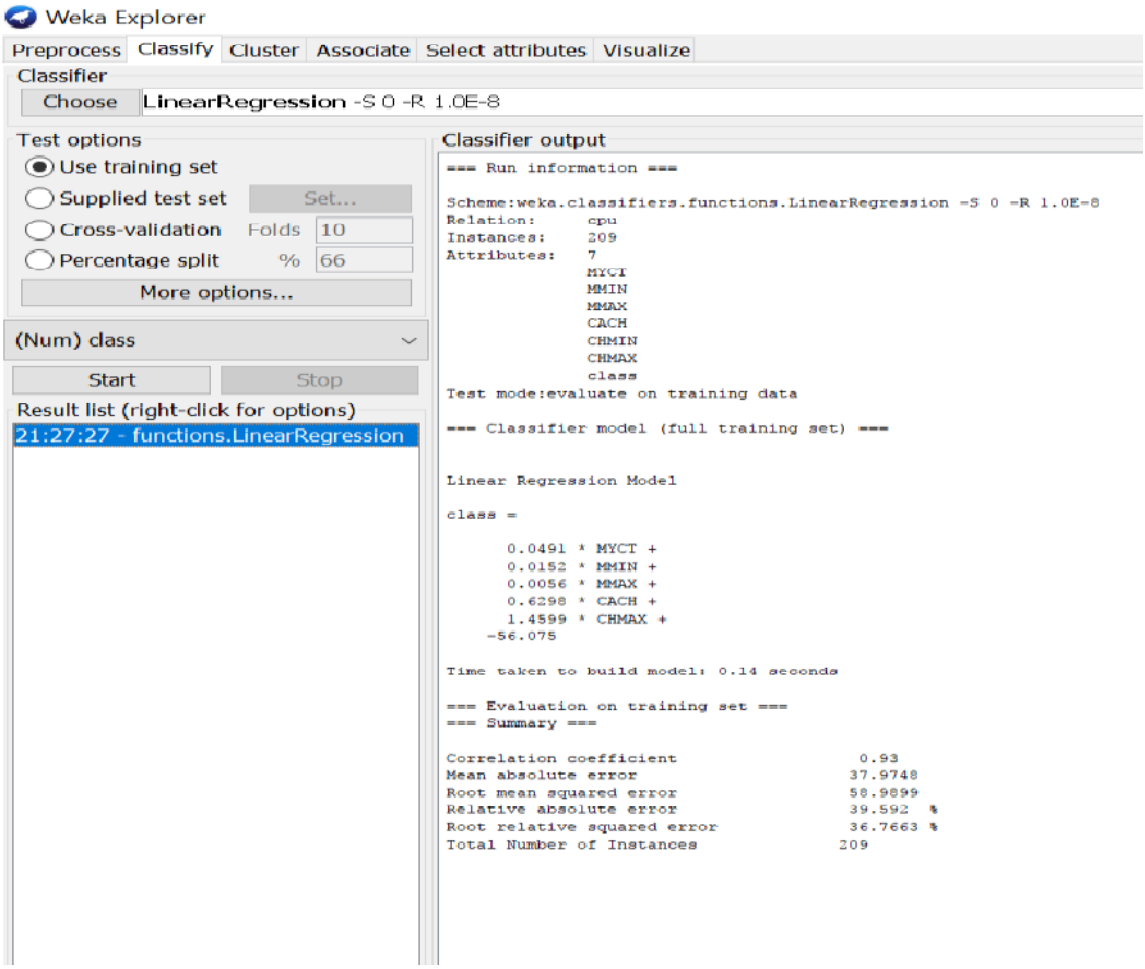
b) Use options cross-validation and percentage split and repeat running the Linear Regression Model. Observe the results and derive meaningful results.

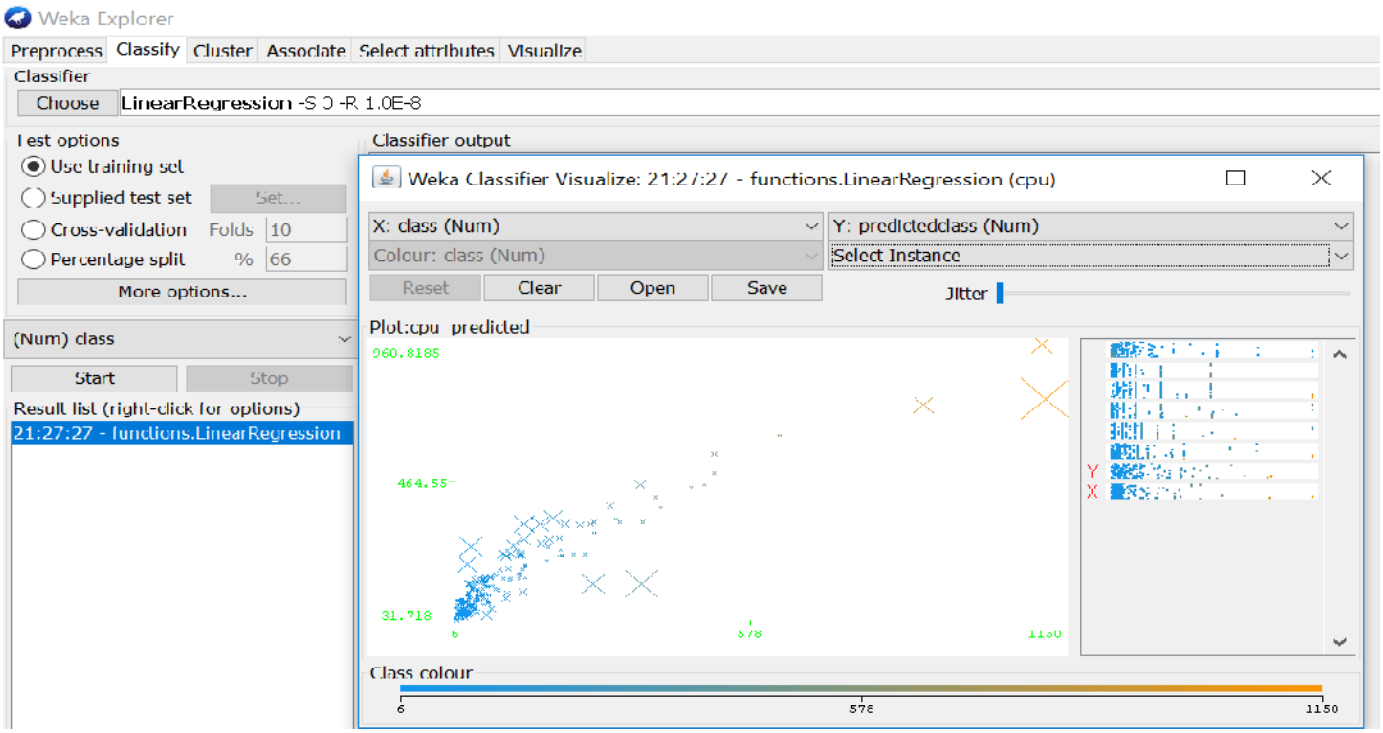
a) Load each dataset into Weka and build Linear Regression model. Study the clusters formed. Use Training set option. Interpret the regression model and derive patterns and conclusions from the regression results.

Procedure:

1. Load the dataset (Cpu.arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see different classification algorithms under functions section.
3. In which we selected Linear Regression algorithm & click on start option with use training set option.
4. Then we will get regression model & its result as shown below.
5. The patterns are visually mentioned below for regression model through visualize classifier errors option which is available in right click options.

Results:



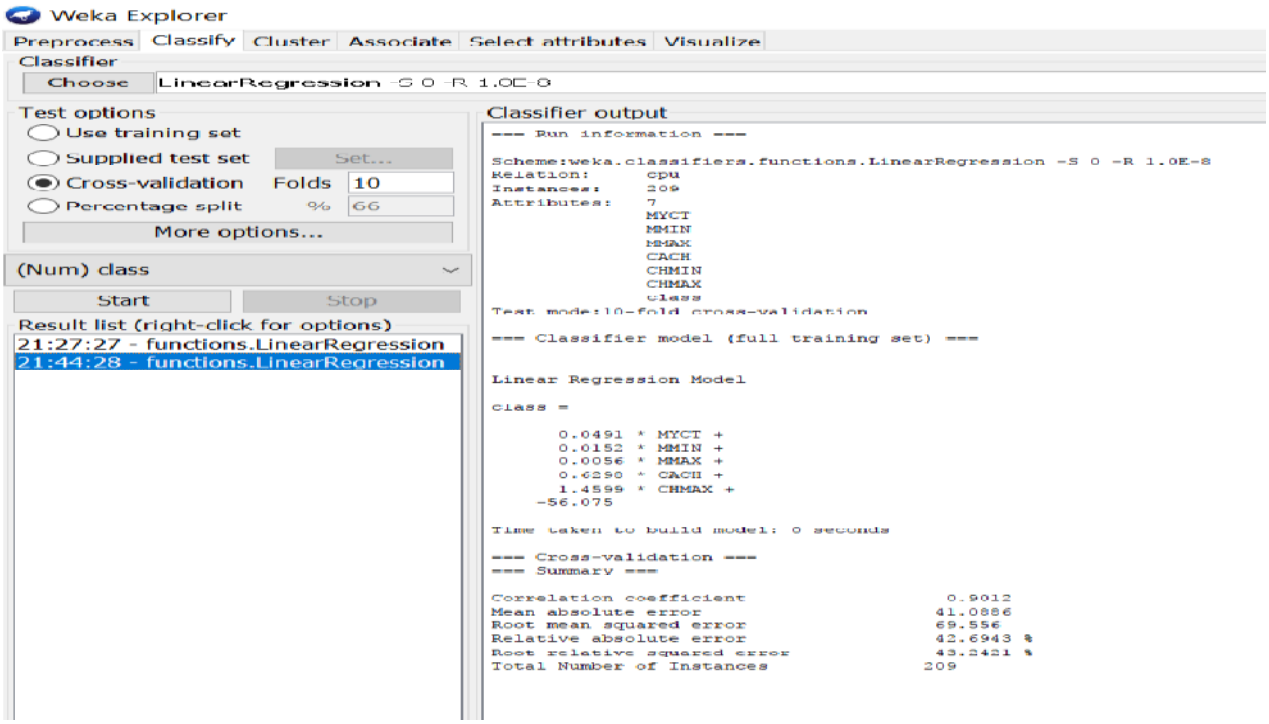


b) Use options cross-validation and percentage split and repeat running the Linear Regression Model. Observe the results and derive meaningful results.

Procedure for cross-validation:

1. Load the dataset (Cpu.arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see different classification algorithms under functions section.
3. In which we selected Linear Regression algorithm & click on start option with cross validation option with 10 folds.
4. Then we will get regression model & its result as shown below.

Result:



EXPERIMENT-18

Aim: Write a python program to perform transformation of data using Discretization (Binning) and normalization (MinMaxScaler or MaxAbsScaler) on given dataset.

Program: for Discretization

```
import numpy as np
import math
from sklearn.datasets import load_iris
from sklearn import datasets, linear_model, metrics


# load iris data set
dataset = load_iris()
a = dataset.data
b = np.zeros(150)

# take 1st column among 4 column of data set
for i in range (150):
        b[i]=a[i,1]

 b=np.sort(b) #sort the array

# create bins
bin1=np.zeros((30,5))
bin2=np.zeros((30,5))
bin3=np.zeros((30,5))


# Bin mean
for i in range (0,150,5):
        k=int(i/5)
        mean=(b[i] + b[i+1] + b[i+2] + b[i+3] + b[i+4])/5
        for j in range(5):
                bin1[k,j]=mean
print("Bin Mean: \n",bin1)


# Bin boundaries
for i in range (0,150,5):
        k=int(i/5)
        for j in range (5):
                if (b[i+j]-b[i]) < (b[i+4]-b[i+j]):
                        bin2[k,j]=b[i]
                else:
                        bin2[k,j]=b[i+4]
print("Bin Boundaries: \n",bin2)
```

```
# Bin median
for i in range (0,150,5):
        k=int(i/5)
        for j in range (5):
                bin3[k,j]=b[i+2]
print("Bin Median: \n",bin3)
```

OutPut:

Bin Mean:
```
[[2.18 2.18 2.18 2.18 2.18]
[2.34 2.34 2.34 2.34 2.34]
[2.48 2.48 2.48 2.48 2.48]
[2.52 2.52 2.52 2.52 2.52]
[2.62 2.62 2.62 2.62 2.62]
[2.7 2.7 2.7 2.7 2.7 ]
[2.74 2.74 2.74 2.74 2.74]
[2.8 2.8 2.8 2.8 2.8 ]
 [2.8 2.8 2.8 2.8 2.8 ]
[2.86 2.86 2.86 2.86 2.86]
[2.9 2.9 2.9 2.9 2.9 ]
[2.96 2.96 2.96 2.96 2.96]
[3. 3. 3. 3. 3. ]
[3. 3. 3. 3. 3. ]
[3. 3. 3. 3. 3. ]
[3. 3. 3. 3. 3. ]
[3.04 3.04 3.04 3.04 3.04]
[3.1 3.1 3.1 3.1 3.1 ]
 [3.12 3.12 3.12 3.12 3.12]
[3.2 3.2 3.2 3.2 3.2 ]
[3.2 3.2 3.2 3.2 3.2 ]
[3.26 3.26 3.26 3.26 3.26]
[3.34 3.34 3.34 3.34 3.34]
 [3.4 3.4 3.4 3.4 3.4 ]
[3.4 3.4 3.4 3.4 3.4 ]
[3.5 3.5 3.5 3.5 3.5 ]
[3.58 3.58 3.58 3.58 3.58]
[3.74 3.74 3.74 3.74 3.74]
[3.82 3.82 3.82 3.82 3.82]
[4.12 4.12 4.12 4.12 4.12]]
```

Bin Boundaries:
```
[[2. 2.3 2.3 2.3 2.3]
[2.3 2.3 2.3 2.4 2.4]
[2.4 2.5 2.5 2.5 2.5]
[2.5 2.5 2.5 2.5 2.6]
[2.6 2.6 2.6 2.6 2.7]
[2.7 2.7 2.7 2.7 2.7]
[2.7 2.7 2.7 2.8 2.8]
 [2.8 2.8 2.8 2.8 2.8]
[2.8 2.8 2.8 2.8 2.8]
[2.8 2.8 2.9 2.9 2.9]
[2.9 2.9 2.9 2.9 2.9]
[2.9 2.9 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. 3.1 3.1]
[3.1 3.1 3.1 3.1 3.1]
[3.1 3.1 3.1 3.1 3.2]
 [3.2 3.2 3.2 3.2 3.2]
[3.2 3.2 3.2 3.2 3.2]
[3.2 3.2 3.3 3.3 3.3]
[3.3 3.3 3.3 3.4 3.4]
 [3.4 3.4 3.4 3.4 3.4]
[3.4 3.4 3.4 3.4 3.4]
 [3.5 3.5 3.5 3.5 3.5]
[3.5 3.6 3.6 3.6 3.6]
 [3.7 3.7 3.7 3.8 3.8]
[3.8 3.8 3.8 3.8 3.9]
[3.9 3.9 3.9 4.4 4.4]]
```

Bin Median:
```
[[2.2 2.2 2.2 2.2 2.2]
[2.3 2.3 2.3 2.3 2.3]
[2.5 2.5 2.5 2.5 2.5]
[2.5 2.5 2.5 2.5 2.5]
[2.6 2.6 2.6 2.6 2.6]
[2.7 2.7 2.7 2.7 2.7]
[2.7 2.7 2.7 2.7 2.7]
[2.8 2.8 2.8 2.8 2.8]
[2.8 2.8 2.8 2.8 2.8]
 [2.9 2.9 2.9 2.9 2.9]
[2.9 2.9 2.9 2.9 2.9]
[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. ]
[3.1 3.1 3.1 3.1 3.1]
[3.1 3.1 3.1 3.1 3.1]
[3.2 3.2 3.2 3.2 3.2]
[3.2 3.2 3.2 3.2 3.2]
[3.3 3.3 3.3 3.3 3.3]
[3.3 3.3 3.3 3.3 3.3]
[3.4 3.4 3.4 3.4 3.4]
[3.4 3.4 3.4 3.4 3.4]
[3.5 3.5 3.5 3.5 3.5]
 [3.6 3.6 3.6 3.6 3.6]
[3.7 3.7 3.7 3.7 3.7]
 [3.8 3.8 3.8 3.8 3.8]
[4.1 4.1 4.1 4.1 4.1]]
```

Program: for transformation of data using normalization

#Example to scale a toy data matrix to the [0, 1] range:

```
from sklearn.preprocessing import MinMaxScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
print(scaler.fit(data))
MinMaxScaler()
print("data:\n",scaler.data_max_)
print("Transformed data:\n",scaler.transform(data))
```

OutPut:

```
MinMaxScaler(copy=True, feature_range=(0, 1))
data:
[ 1. 18.]

Transformed data:
[[0. 0. ]
[0.25 0.25]
[0.5 0.5 ]
[1. 1. ]]
```