GURU NANAK DEV ENGINEERING COLLEGE,LUDHIANA

Department of Information Technology

LPEIT-101

Business Intelligence & Its Applications

LABORATORY MANUAL

Subject Code: LPEIT-101
Subject Name: Business Intelligence & its Applications Laboratory

Programme: B. Tech.	**L:**0 **T:**0 **P:**2
Semester:5	**Teaching Hours:** 24
Theory/Practical : Practical	**Credits:**1
Internal Marks:30	**Percentage of Numerical/Design Problems:**100%
External Marks:20	**Duration of End Semester Exam(ESE):**1.5 Hours
Total Marks:50	**Course Type:** Professional Elective-I

On Completion of the course, the student will have the ability to:

CO#	Course Outcomes
1.	Conduct Investigation on real world problems using BI tools like QlikView, Google analytics etc.
2.	Exemplify the implementation of data mart.
3.	Apply data mining algorithms like, Apriori etc. for analysis and prediction of data for health , social , cultural issues etc.
4.	Develop solutions for multi-disciplinary dataset by applying different classification and clustering methods using different Data Mining tools
5.	Identify different Business intelligence tools for different applications
6.	Function effectively as individual or as team in multidisciplinary area of engineering practices.

Prerequisites: Fundamentals of Computer with any basic programming language and knowledge of Database Management Systems.

Detailed Contents:

1. Case Study and Design of a Data Mart Application
2. To study different Data Mining tools
3. To Perform Data Cleaning on Data Sets
4. To Perform association rule mining (AprioriAlgorithm) using any Data Mining tool.
5. To perform classification using Bayes Method using any Data Mining tool.
6. To Perform K-means clustering techniques for data mining on data set using any Data Mining tool.
7. To interpret and visualize the output of data mining using any Data mining tool.

Mini Project: - Student has to do a project assigned from course contents in a group of two or three students. The group of students must submit a project report of 8 to 10 pages (approximately) and the team will have to demonstrate as well as have to give a presentation of the same.
Note: It is recommended that mini project allocation to students be done within two-three weeks of the start of the semester. This is only the suggested list of Practical’s. Instructor may also frame additional Practical’s relevant to the course contents (if required).

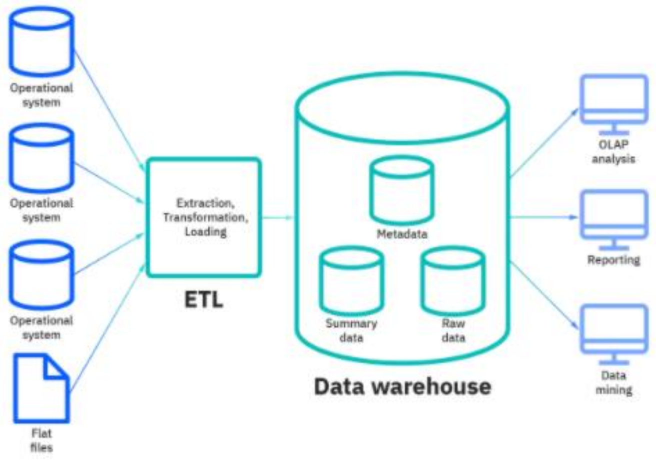
Practical-1 Case Study and Design of a Data Mart Application

Data Warehouse

A core component of business intelligence, a data warehouse pulls together data from many different sources into a single data repository for sophisticated analytics and decision support.

Data warehouse architecture

- **Bottom tier:** The bottom tier consists of a data warehouse server, usually a relational database system, which collects, cleanses, and transforms data from multiple data sources through a process known as Extract, Transform, and Load (ETL) or a process known as Extract, Load, and Transform (ELT).
- **Middle tier:** The middle tier consists of an OLAP (i.e. online analytical processing) server which enables fast query speeds. Three types of OLAP models can be used in this tier, which are known as ROLAP, MOLAP and HOLAP. The type of OLAP model used is dependent on the type of database system that exists.
- **Top tier:** The top tier is represented by some kind of front-end user interface or reporting tool, which enables end users to conduct ad-hoc data analysis on their business data.



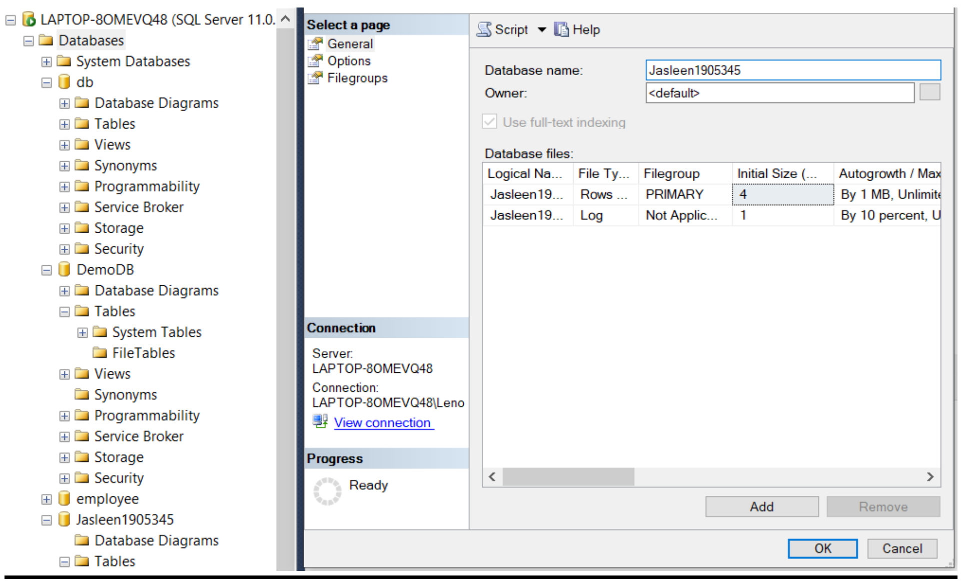
How to Create a Table in SQL Server

KEYS

A **Primary Key** *uniquely* identifies each record (i.e., row) in your table, while an **Identity Column** ensures that an auto-increment is applied to your column whenever a new record is inserted into the table.
Let’s say that we want to create a table with a Primary Key and Identity Column as given below.

Step 1: Create a database

we may use the following query to create a database called the Jasleen1905345**:**



Database name: Jasleen1905345
Owner: <default>

Server: LAPTOP-8OMEVQ48
Connection: LAPTOP-8OMEVQ48\Leno

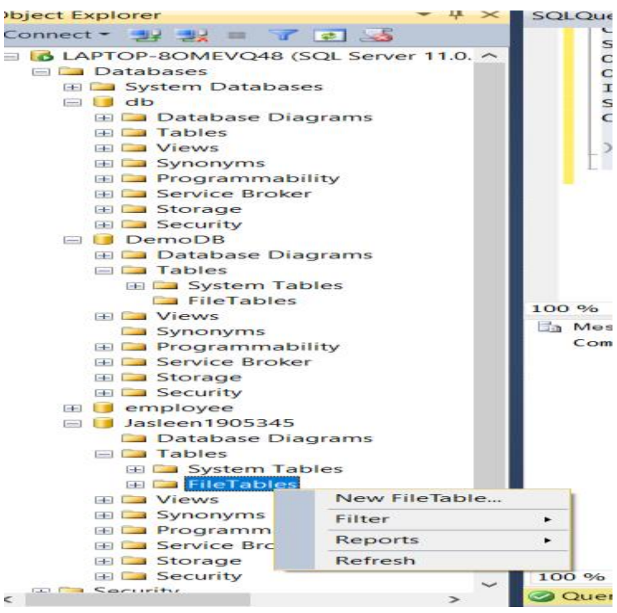
Database files:

Logical Na...	File Ty...	Filegroup	Initial Size (...	Autogrowth / Max
Jasleen19...	Rows ...	PRIMARY	4	By 1 MB, Unlimite
Jasleen19...	Log	Not Applic...	1	By 10 percent, U

Step 2: Create a table

Next, create a table under your database.

- ➔ CLICK ON FILE TABLES IN TABLES . SELECT NEW FILE TABLE .
- ➔ WRITE DOWN THE FOLLOWING COMMANDS IN THAT.



- ➔ NOW USE THE DATABASE IN WHICH YOU LIKE TO MAKE YOUR TABLES

```
use Jasleen1905345;
GO
```

➔ CREATE DIMENTION TABLES

A Dimension Table is present in the star or snowflake schema. Dimension tables' help to describe dimensions i.e. dimension values, attributes and keys. It is generally small in size. Size can range from several to thousand rows. It describes the objects present in the fact table. Dimension Table refers to the collection or group of information related to any measurable event. They form a core for dimensional modelling. It contains a column that can be considered as a primary key column which helps to uniquely identify every dimension row or record. It is being joined with the fact tables through this key. When it is created a key called surrogate key that is system generated is used to uniquely identify the rows in the dimension.

You can then create the table using the following CREATE TABLE query under your database:

```
Create Table DimProduct
( Productkey int identity Not Null PRIMARY KEY NONCLUSTERED,
ProductAltKey nvarchar(10) Not Null,
ProductName nvarchar(50) Null,
ProductFDescription nvarchar(100) Null,
ProductCategoryName nvarchar(50))
GO

Create Table DimCustomer
( CustomerKey int identity Not Null PRIMARY KEY NONCLUSTERED,
CustomerAltKey nvarchar(10) Not Null,
CustomerName nvarchar(50) Null,
CustoemrEmail nvarchar(50)Null,
CustomerGeographyKey int Null)
GO

Create Table DimSalesperson
(
SalespersonKey int identity Not Null PRIMARY KEY NONCLUSTERED,
SalesPersonAltKey nvarchar(10) Not Null,
SalesPersonName nvarchar(50) Null,
StoreName varchar(50)Null,
StoreGeographyKey int Null)
GO

Create Table DimDate
(
DateKey int identity Not Null PRIMARY KEY NONCLUSTERED,
DateAltKey nvarchar(10) Not Null,
CalenderYear int Not Null,
CalenderQuater int Not Null,
MonthOfYear int NOT Null,

[DayOfMonth] int Not Null)
GO
```

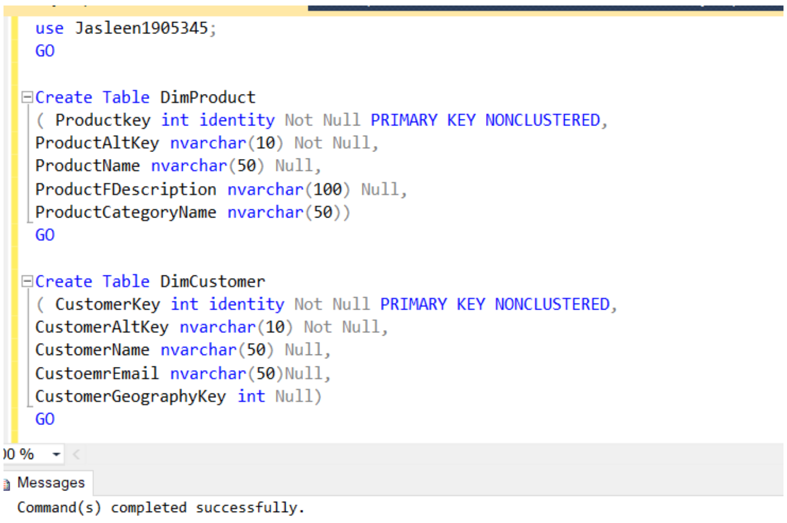
➔ CREATE A FACT TABLE

A fact table is the central table in star schema of a data warehouse. A fact table stores quantitative information for analysis and is often denormalized.

A fact table works with dimension tables. A fact table holds the data to be analyzed, and a dimension table stores data about the ways in which the data in the fact table can be analyzed. Thus, the fact table consists of two types of columns. The foreign keys column allows joins with dimension tables, and the measures columns contain the data that is being analyzed.

```
Create Table FactSalesOrder
(Productkey int Not Null REFERENCES DimProduct(ProductKey),
CustomerKey int Not Null REFERENCES DimCustomer(CustomerKey),
SalespersonKey int Not Null REFERENCES DimSalesperson(SalespersonKey),
OrderDateKey int Not Null REFERENCES DimDate(DateKey),
OrderNo int Not Null,
ItemNo int Not  Null,
SalesAmount money Not Null,
Cost money Not Null,

)
```

➔ EXECUTE IT



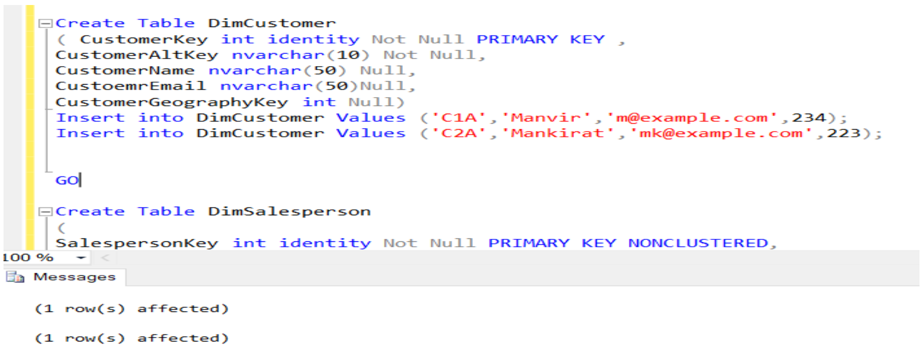
```
use Jasleen1905345;
GO

Create Table DimProduct
( Productkey int identity Not Null PRIMARY KEY NONCLUSTERED,
ProductAltKey nvarchar(10) Not Null,
ProductName nvarchar(50) Null,
ProductFDescription nvarchar(100) Null,
ProductCategoryName nvarchar(50))
GO

Create Table DimCustomer
( CustomerKey int identity Not Null PRIMARY KEY NONCLUSTERED,
CustomerAltKey nvarchar(10) Not Null,
CustomerName nvarchar(50) Null,
CustoemrEmail nvarchar(50)Null,
CustomerGeographyKey int Null)
GO
```

```
Command(s) completed successfully.
```

Step 3: Insert values into the table

You can insert values into the table using an INSERT INTO query:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```



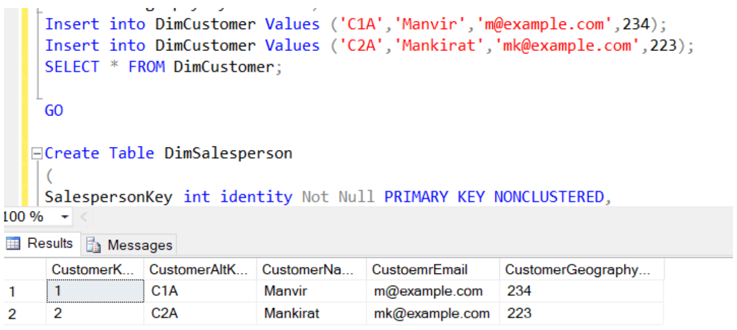
```
Create Table DimCustomer
( CustomerKey int identity Not Null PRIMARY KEY ,
CustomerAltKey nvarchar(10) Not Null,
CustomerName nvarchar(50) Null,
CustoemrEmail nvarchar(50)Null,
CustomerGeographyKey int Null)
Insert into DimCustomer Values ('C1A','Manvir','m@example.com',234);
Insert into DimCustomer Values ('C2A','Mankirat','mk@example.com',223);


GO

Create Table DimSalesperson
(
SalespersonKey int identity Not Null PRIMARY KEY NONCLUSTERED,
```

```
(1 row(s) affected)

(1 row(s) affected)
```

Step 4: Verify that the values were inserted into the table

Finally, run the following SELECT query to verify that the values were inserted into the table:

```
SELECT * FROM table_Name;
```



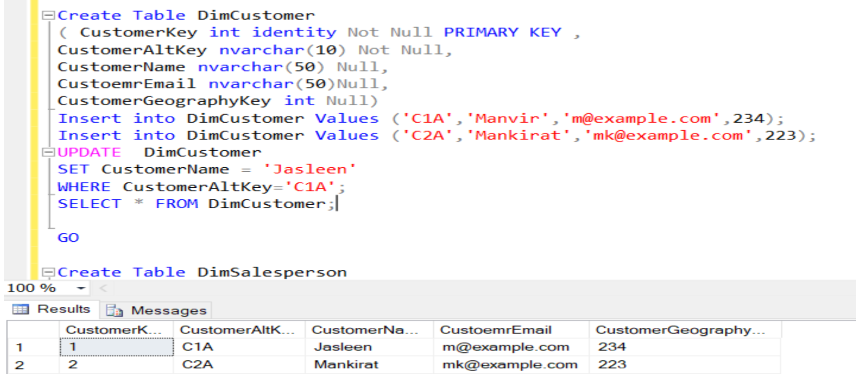
```
Insert into DimCustomer Values ('C1A','Manvir','m@example.com',234);
Insert into DimCustomer Values ('C2A','Mankirat','mk@example.com',223);
SELECT * FROM DimCustomer;

GO

Create Table DimSalesperson
(
SalespersonKey int identity Not Null PRIMARY KEY NONCLUSTERED,
```

	CustomerK...	CustomerAltK...	CustomerNa...	CustoemrEmail	CustomerGeography...
1	1	C1A	Manvir	m@example.com	234
2	2	C2A	Mankirat	mk@example.com	223

Step 4: Update values into the table

If you want to update any entry/row in table then use UPDATE command:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```



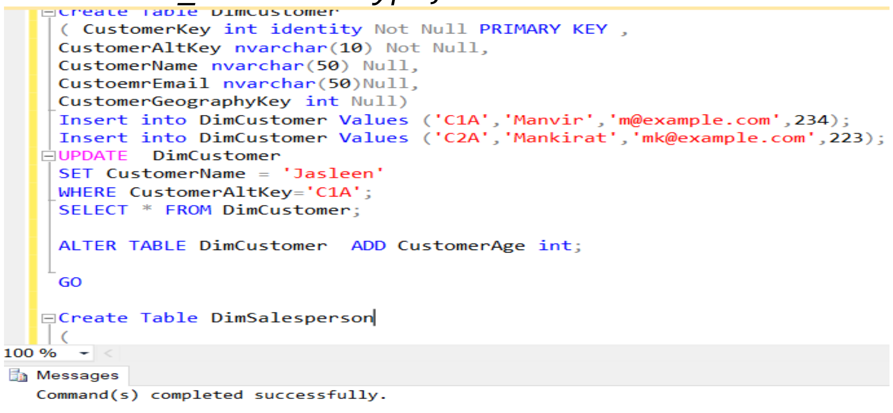
```
Create Table DimCustomer
( CustomerKey int identity Not Null PRIMARY KEY ,
CustomerAltKey nvarchar(10) Not Null,
CustomerName nvarchar(50) Null,
CustoemrEmail nvarchar(50)Null,
CustomerGeographyKey int Null)
Insert into DimCustomer Values ('C1A','Manvir','m@example.com',234);
Insert into DimCustomer Values ('C2A','Mankirat','mk@example.com',223);
UPDATE  DimCustomer
SET CustomerName = 'Jasleen'
WHERE CustomerAltKey='C1A';
SELECT * FROM DimCustomer;

GO

Create Table DimSalesperson
```

	CustomerK...	CustomerAltK...	CustomerNa...	CustoemrEmail	CustomerGeography...
1	1	C1A	Jasleen	m@example.com	234
2	2	C2A	Mankirat	mk@example.com	223

Step 5: Alter columns into the table

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

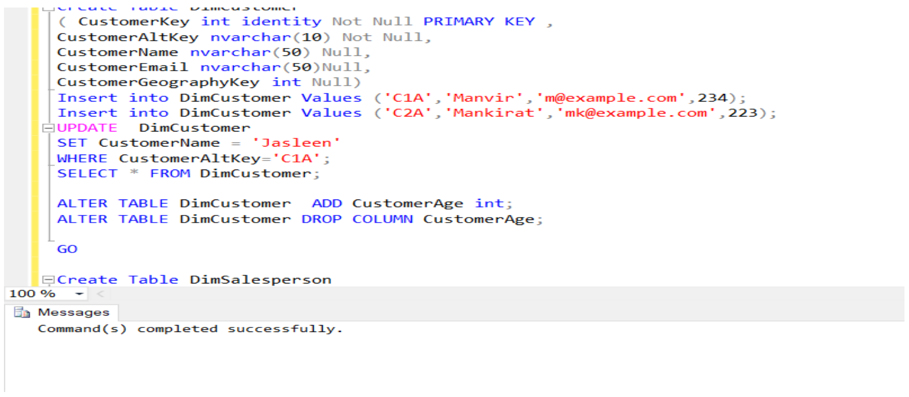


```
Create Table DimCustomer
( CustomerKey int identity Not Null PRIMARY KEY ,
CustomerAltKey nvarchar(10) Not Null,
CustomerName nvarchar(50) Null,
CustoemrEmail nvarchar(50)Null,
CustomerGeographyKey int Null)
Insert into DimCustomer Values ('C1A','Manvir','m@example.com',234);
Insert into DimCustomer Values ('C2A','Mankirat','mk@example.com',223);
UPDATE  DimCustomer
SET CustomerName = 'Jasleen'
WHERE CustomerAltKey='C1A';
SELECT * FROM DimCustomer;

ALTER TABLE DimCustomer  ADD CustomerAge int;

GO

Create Table DimSalesperson
(
```

```
Command(s) completed successfully.
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```



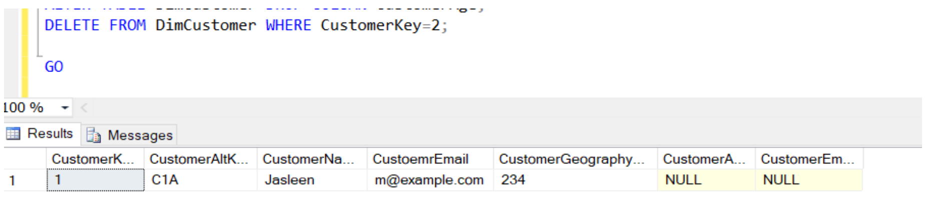
```
( CustomerKey int identity Not Null PRIMARY KEY ,
CustomerAltKey nvarchar(10) Not Null,
CustomerName nvarchar(50) Null,
CustomerEmail nvarchar(50)Null,
CustomerGeographyKey int Null)
Insert into DimCustomer Values ('C1A','Manvir','m@example.com',234);
Insert into DimCustomer Values ('C2A','Mankirat','mk@example.com',223);
UPDATE  DimCustomer
SET CustomerName = 'Jasleen'
WHERE CustomerAltKey='C1A';
SELECT * FROM DimCustomer;

ALTER TABLE DimCustomer  ADD CustomerAge int;
ALTER TABLE DimCustomer DROP COLUMN CustomerAge;

GO

Create Table DimSalesperson
```

```
Messages
Command(s) completed successfully.
```

Step 5: Delete rows of the table

If you want to delete particular entry in a table then use DELETE command:

`DELETE FROM table_name WHERE condition;`



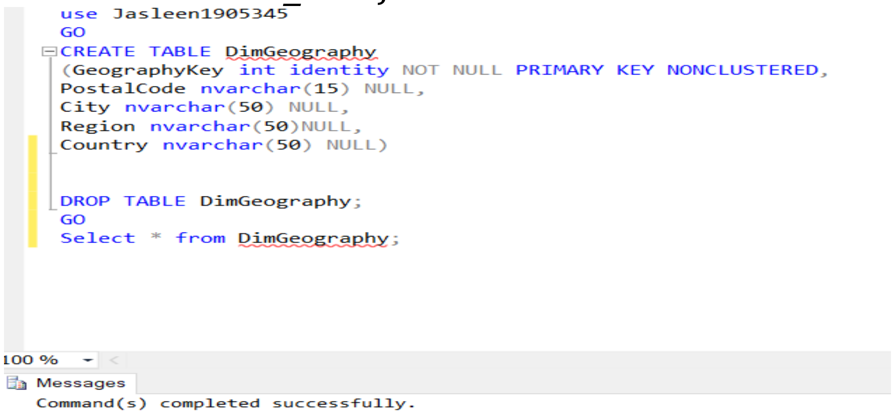
```
DELETE FROM DimCustomer WHERE CustomerKey=2;

GO
```

	CustomerK...	CustomerAltK...	CustomerNa...	CustoemrEmail	CustomerGeography...	CustomerA...	CustomerEm...
1	1	C1A	Jasleen	m@example.com	234	NULL	NULL

Step 6: Drop the table

The DROP TABLE statement is used to drop an existing table in a database.

`DROP TABLE table_name;`

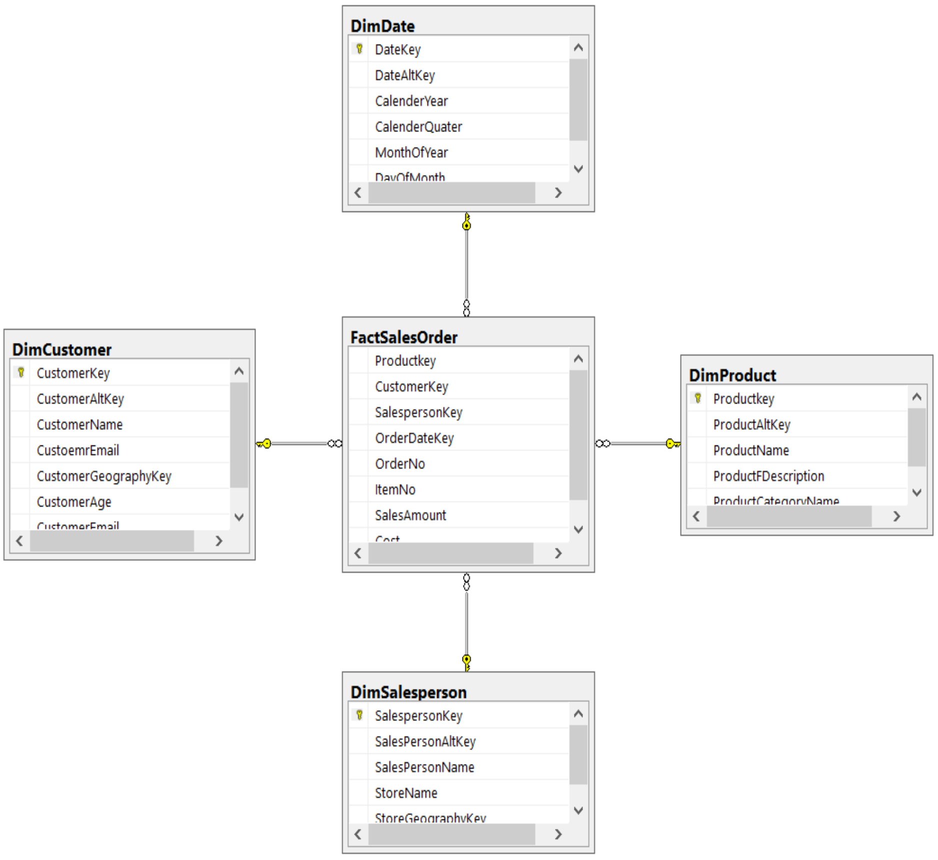


```
use Jasleen1905345
GO
CREATE TABLE DimGeography
(GeographyKey int identity NOT NULL PRIMARY KEY NONCLUSTERED,
PostalCode nvarchar(15) NULL,
City nvarchar(50) NULL,
Region nvarchar(50)NULL,
Country nvarchar(50) NULL)


DROP TABLE DimGeography;
GO
Select * from DimGeography;
```

```
Messages
Command(s) completed successfully.
```

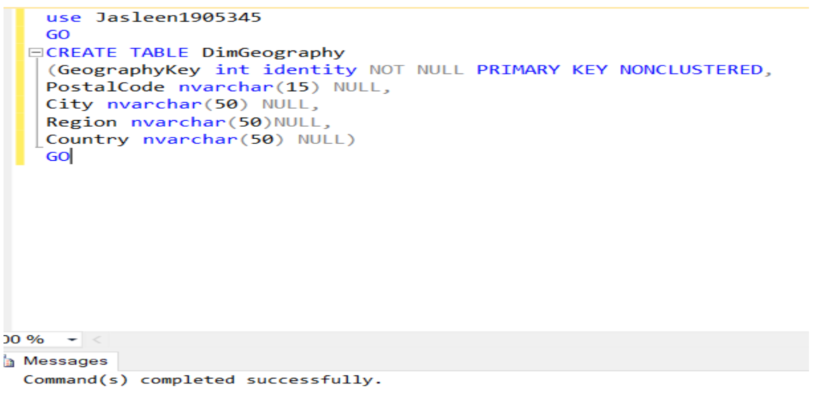
STAR SCHEMA

A Star Schema is a schema Architectural structure used to create and implement the Data Warehouse systems, where there is only one fact table and multiple dimension tables connected to it. It is structured like a star in the shape of appearance. This is one of the efficient data warehouse schema types, which can use simple querying to access the data from the system to derive logical contents for analytical and report generation purposes.



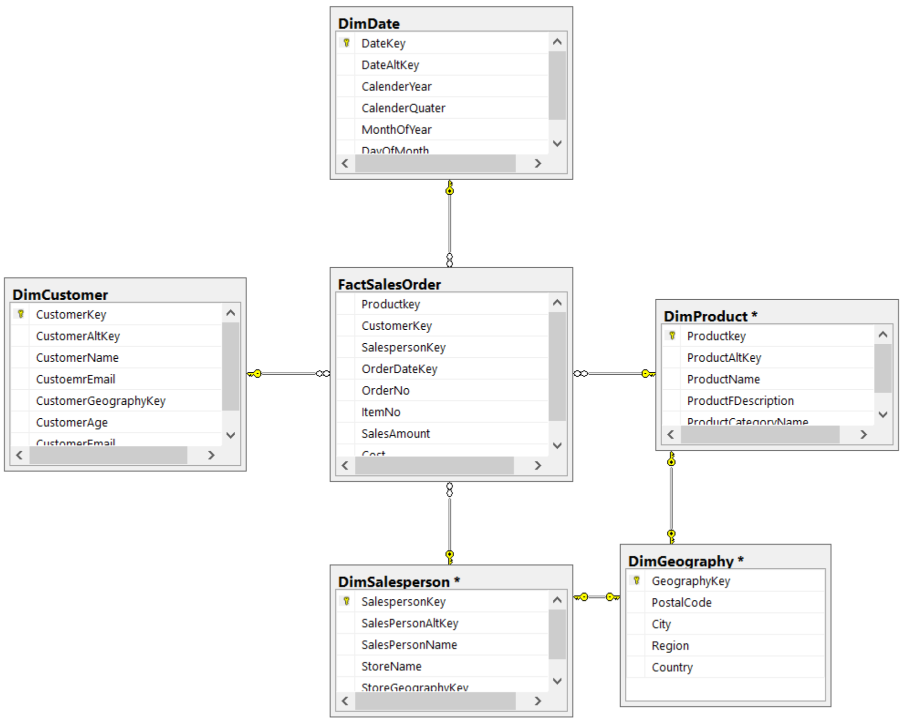
SnowFlake Schema

The snowflake schema is a variant of the star schema. Here, the centralized fact table is connected to multiple dimensions. In the snowflake schema, dimensions are present in a normalized form in multiple related tables. The snowflake structure materialized when the dimensions of a star schema are detailed and highly structured, having several levels of relationship, and the child tables have multiple parent tables. The snowflake effect affects only the dimension tables and does not affect the fact tables.



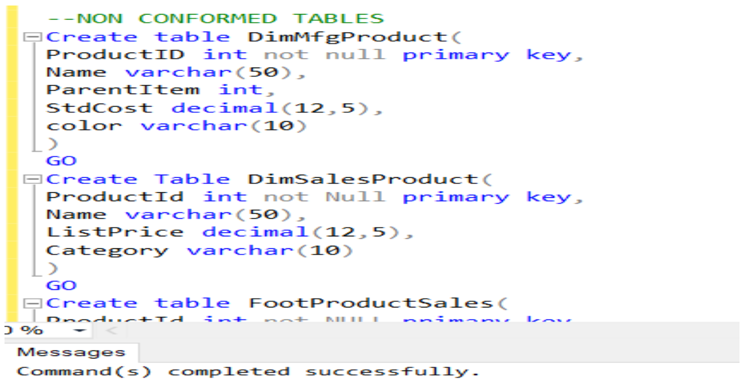
```
use Jasleen1905345
GO
CREATE TABLE DimGeography
(GeographyKey int identity NOT NULL PRIMARY KEY NONCLUSTERED,
PostalCode nvarchar(15) NULL,
City nvarchar(50) NULL,
Region nvarchar(50)NULL,
Country nvarchar(50) NULL)
GO
```

```
Command(s) completed successfully.
```



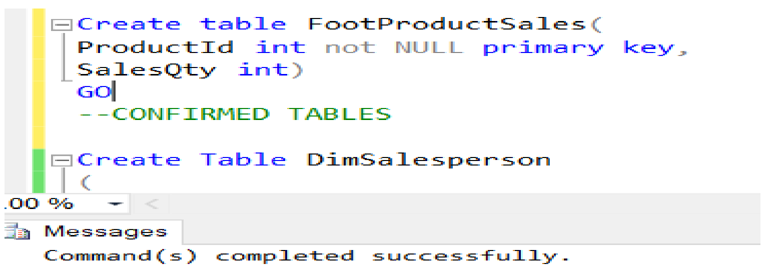
NON CONFORMED TABLES

A non-conformed dimension is where you add an attribute to an analysis which does not logically relate to all the facts.



```
--NON CONFORMED TABLES
Create table DimMfgProduct(
ProductID int not null primary key,
Name varchar(50),
ParentItem int,
StdCost decimal(12,5),
color varchar(10)
)
GO
Create Table DimSalesProduct(
ProductId int not Null primary key,
Name varchar(50),
ListPrice decimal(12,5),
Category varchar(10)
)
GO
Create table FootProductSales(
```

```
Messages
Command(s) completed successfully.
```

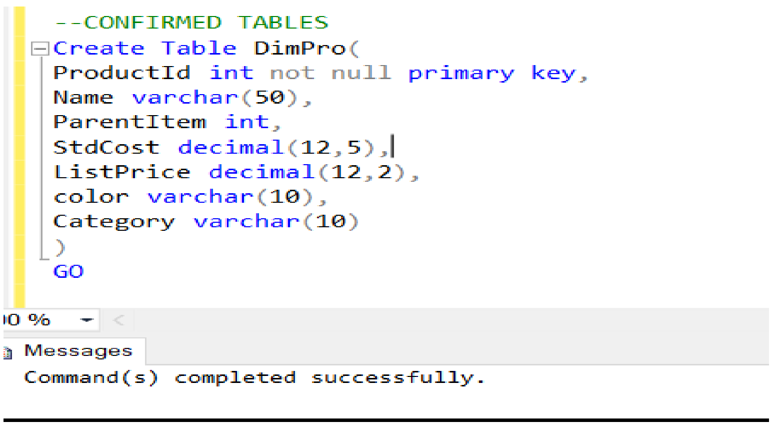


```
Create table FootProductSales(
ProductId int not NULL primary key,
SalesQty int)
GO
--CONFIRMED TABLES

Create Table DimSalesperson
(
```

```
Messages
Command(s) completed successfully.
```

CONFORMED DIMENSION

Dimension tables ***conform*** when attributes in separate dimension tables have the same column names and domain contents. Information from separate fact tables can be combined in a single report by using conformed dimension attributes that are associated with each fact table. When a conformed attribute is used as the row header (that is, the grouping column in the SQL query), the results from the separate fact tables can be aligned on the same rows in a drill-across report. This is the essence of integration in an enterprise DW/ BI system. ***Conformed dimensions***, defined once in collaboration with the business's data governance representatives, are reused across fact tables; they deliver both analytic consistency and reduced future development costs because the wheel is not repeatedly re-created.



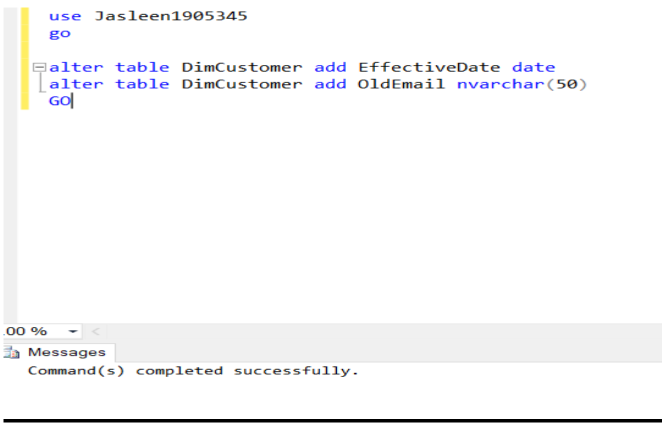
```
--CONFIRMED TABLES
Create Table DimPro(
ProductId int not null primary key,
Name varchar(50),
ParentItem int,
StdCost decimal(12,5),
ListPrice decimal(12,2),
color varchar(10),
Category varchar(10)
)
GO
```

```
Messages
Command(s) completed successfully.
```

Slowly changing dimensions

The term slowly changing dimensions encompasses the following three different methods for handling changes to columns in a data warehouse dimension table:

- Type 1 - update the columns in the dimension row without preserving any change history.
- Type 2 - preserve the change history in the dimension table and create a new row when there are changes.
- Type 3 - some combination of Type 1 and Type 2, usually maintaining multiple instances of a column in the dimension row; e.g. a current value and one or more previous values.



```
use Jasleen1905345
go

alter table DimCustomer add EffectiveDate date
alter table DimCustomer add OldEmail nvarchar(50)
GO
```

```
Messages
Command(s) completed successfully.
```

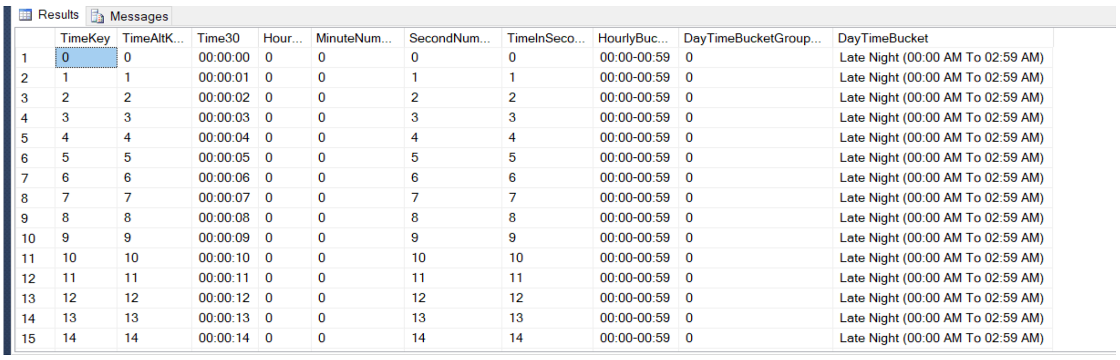
POPULATE THE TIME DIMENSION TABLE

```
use Jasleen1905345
GO

--POPULATE THE TIME DIMENSION TABLE
```

```sql
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [Jasleen1905345].[DimTime](
[TimeKey] [int] NOT NULL,
[TimeAltKey] [int] NOT NULL,
[Time30] [varchar](8) NOT NULL,
[Hour30] [tinyint] NOT NULL,
[MinuteNumber] [tinyint] NOT NULL,
[SecondNumber] [tinyint] NOT NULL,
[TimeInSecond] [int] NOT NULL,
[HourlyBucket] varchar(15)not null,
[DayTimeBucketGroupKey] int not null,
[DayTimeBucket] varchar(100) not null
CONSTRAINT [PK_DimTime] PRIMARY KEY CLUSTERED
(
[TimeKey] ASC
)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
)
ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/***** Create Stored procedure In Test_DW and Run SP To Fill Time Dimension with Values****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [Jasleen1905345].[FillDimTime]
as
BEGIN
--Specify Total Number of Hours You need to fill in Time Dimension
DECLARE @Size INTEGER
--iF @Size=32 THEN This will Fill values Upto 32:59 hr in Time Dimension
Set @Size=23
DECLARE @hour INTEGER
DECLARE @minute INTEGER
DECLARE @second INTEGER
DECLARE @k INTEGER
DECLARE @TimeAltKey INTEGER
DECLARE @TimeInSeconds INTEGER
DECLARE @Time30 varchar(25)
DECLARE @Hour30 varchar(4)
DECLARE @Minute30 varchar(4)
DECLARE @Second30 varchar(4)
DECLARE @HourBucket varchar(15)
DECLARE @HourBucketGroupKey int
DECLARE @DayTimeBucket varchar(100)
DECLARE @DayTimeBucketGroupKey int
```

```
SET @hour = 0
SET @minute = 0
SET @second = 0
SET @k = 0
SET @TimeAltKey = 0
WHILE(@hour<= @Size )
BEGIN
if (@hour <10 )
begin
set @Hour30 = '0' + cast( @hour as varchar(10))
end
else
begin
set @Hour30 = @hour
end
--Create Hour Bucket Value
set @HourBucket= @Hour30+':00' +'-' +@Hour30+':59'
WHILE(@minute <= 59)
BEGIN
WHILE(@second <= 59)
BEGIN
set @TimeAltKey = @hour *10000 +@minute*100 +@second
set @TimeInSeconds =@hour * 3600 + @minute *60 +@second
If @minute <10
begin
set @Minute30 = '0' + cast ( @minute as varchar(10) )
end
else
begin
set @Minute30 = @minute
end
if @second <10
begin
set @Second30 = '0' + cast ( @second as varchar(10) )
end
else
begin
set @Second30 = @second
end
--Concatenate values for Time30
set @Time30 = @Hour30 +':'+@Minute30 +':'+@Second30
--DayTimeBucketGroupKey can be used in Sorting of DayTime Bucket In proper Order
SELECT @DayTimeBucketGroupKey =
CASE
WHEN (@TimeAltKey >= 00000 AND @TimeAltKey <= 25959) THEN 0
WHEN (@TimeAltKey >= 30000 AND @TimeAltKey <= 65959) THEN 1
WHEN (@TimeAltKey >= 70000 AND @TimeAltKey <= 85959) THEN 2
WHEN (@TimeAltKey >= 90000 AND @TimeAltKey <= 115959) THEN 3
WHEN (@TimeAltKey >= 120000 AND @TimeAltKey <= 135959)THEN 4
WHEN (@TimeAltKey >= 140000 AND @TimeAltKey <= 155959)THEN 5
WHEN (@TimeAltKey >= 50000 AND @TimeAltKey <= 175959) THEN 6
WHEN (@TimeAltKey >= 180000 AND @TimeAltKey <= 235959)THEN 7
WHEN (@TimeAltKey >= 240000) THEN 8
END
--print @DayTimeBucketGroupKey
```

```
-- DayTimeBucket Time Divided in Specific Time Zone
-- So Data can Be Grouped as per Bucket for Analyzing as per time of day
SELECT @DayTimeBucket =
CASE
WHEN (@TimeAltKey >= 00000 AND @TimeAltKey <= 25959)
THEN 'Late Night (00:00 AM To 02:59 AM)'
WHEN (@TimeAltKey >= 30000 AND @TimeAltKey <= 65959)
THEN 'Early Morning(03:00 AM To 6:59 AM)'
WHEN (@TimeAltKey >= 70000 AND @TimeAltKey <= 85959)
THEN 'AM Peak (7:00 AM To 8:59 AM)'
WHEN (@TimeAltKey >= 90000 AND @TimeAltKey <= 115959)
THEN 'Mid Morning (9:00 AM To 11:59 AM)'
WHEN (@TimeAltKey >= 120000 AND @TimeAltKey <= 135959)
THEN 'Lunch (12:00 PM To 13:59 PM)'
WHEN (@TimeAltKey >= 140000 AND @TimeAltKey <= 155959)
THEN 'Mid Afternoon (14:00 PM To 15:59 PM)'
WHEN (@TimeAltKey >= 50000 AND @TimeAltKey <= 175959)
THEN 'PM Peak (16:00 PM To 17:59 PM)'
WHEN (@TimeAltKey >= 180000 AND @TimeAltKey <= 235959)
THEN 'Evening (18:00 PM To 23:59 PM)'
WHEN (@TimeAltKey >= 240000) THEN 'Previous Day Late Night
(24:00 PM to '+cast( @Size as varchar(10)) +':00 PM )'
END
-- print @DayTimeBucket
INSERT into DimTime (TimeKey,TimeAltKey,[Time30] ,[Hour30] ,
[MinuteNumber],[SecondNumber],[TimeInSecond],[HourlyBucket],
DayTimeBucketGroupKey,DayTimeBucket)
VALUES (@k,@TimeAltKey ,@Time30 ,@hour ,@minute,@Second ,
@TimeInSeconds,@HourBucket,@DayTimeBucketGroupKey,@DayTimeBucket )
SET @second = @second + 1
SET @k = @k + 1
END
SET @minute = @minute + 1
SET @second = 0
END
SET @hour = @hour + 1
SET @minute =0
END
END
Go
Exec [FillDimTime]
go
select * from DimTime;
```



Results | Messages

	TimeKey	TimeAltK...	Time30	Hour...	MinuteNum...	SecondNum...	TimeInSeco...	HourlyBuc...	DayTimeBucketGroup...	DayTimeBucket
1	0	0	00:00:00	0	0	0	0	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
2	1	1	00:00:01	0	0	1	1	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
3	2	2	00:00:02	0	0	2	2	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
4	3	3	00:00:03	0	0	3	3	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
5	4	4	00:00:04	0	0	4	4	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
6	5	5	00:00:05	0	0	5	5	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
7	6	6	00:00:06	0	0	6	6	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
8	7	7	00:00:07	0	0	7	7	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
9	8	8	00:00:08	0	0	8	8	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
10	9	9	00:00:09	0	0	9	9	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
11	10	10	00:00:10	0	0	10	10	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
12	11	11	00:00:11	0	0	11	11	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
13	12	12	00:00:12	0	0	12	12	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
14	13	13	00:00:13	0	0	13	13	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)
15	14	14	00:00:14	0	0	14	14	00:00-00:59	0	Late Night (00:00 AM To 02:59 AM)

<u>**Practical-2 To study different Data Mining tools**</u>

Data Mining Tools

Weka

Weka is the tool most commonly used due to its vast functionality and supported features. This java based data mining tool provides user with both GUI and simple CLI for performing and managing tasks to be performed. It supports all data mining tasks from preprocessing, classification, and clustering to visualization and feature selection.

RapidMiner

*R*apidMiner is a free open-source data science platform that features hundreds of algorithms for data preparation, machine learning, deep learning, text mining, and predictive analytics.

Its drag-and-drop interface and pre-built models allow non-programmers to intuitively create predictive workflows for specific use cases, like fraud detection and customer churn. Meanwhile, programmers can take advantage of RapidMiner's R and Python extensions to tailor their data mining.

Once you've created your workflows and analyzed your data, visualize your results in RapidMiner Studio, to help you spot patterns, outliers, and trends in your data.

Oracle Data Mining

Oracle Data Mining is a component of Oracle Advanced Analytics that enables data analysts to build and implement predictive models. It contains several data mining algorithms for tasks like classification, regression, anomaly detection, prediction, and more.

With Oracle Data Mining, you can build models that help you predict customer behavior, segment customer profiles, detect fraud, and identify the best prospects to target. Developers can use a Java API to integrate these models into business intelligence applications to help them discover new trends and patterns.

Knime

KNIME (Konstanz Information Miner) is an open API workflow based data mining tool that provides easy accessibility to new nodes to be added into the workflow. It provides its user with the GUI which aid with the simplification of workflow generation by the user. It also provides with features to modify a particular node accordingly and execution of partial data flow

Tanagra

This extension of SIPINA provides the users an easy to use interface for the analysis of either real or artificial data. It allows the researchers to easily add their own data mining research methodology or any newly identified data mining processing technique and also supports by providing them with architecture and a means to compare their methodology performances. It provides the beginners or naives with a platform where they can carry out their experimental procedures.

Orange

Orange is a free, open-source data science toolbox for developing, testing, and visualizing data mining workflows.

It is a component-based software, with a large collection of pre-built machine learning algorithms and text mining add-ons. It also has extended functionalities for bioinformaticians and molecular biologists.

Orange also allows for interactive data visualization, offering numerous graphics like silhouette plots and sieve diagrams, and non-programmers can perform data mining tasks through visual programming in a drag-and-drop interface. Developers, meanwhile, can opt to mine data in Python.

Practical 3 To Perform Data Cleaning on Data Sets

Aim: This experiment illustrates some of the basic data preprocessing operations that can be performed using WEKA-Explorer. The sample dataset used for this example is the student data available in arff format.

Step1: Loading the 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 above figure 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

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.

Step 6: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-7”.

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.(student.arff)

<u>Discretization</u>

1) 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 age attribute.

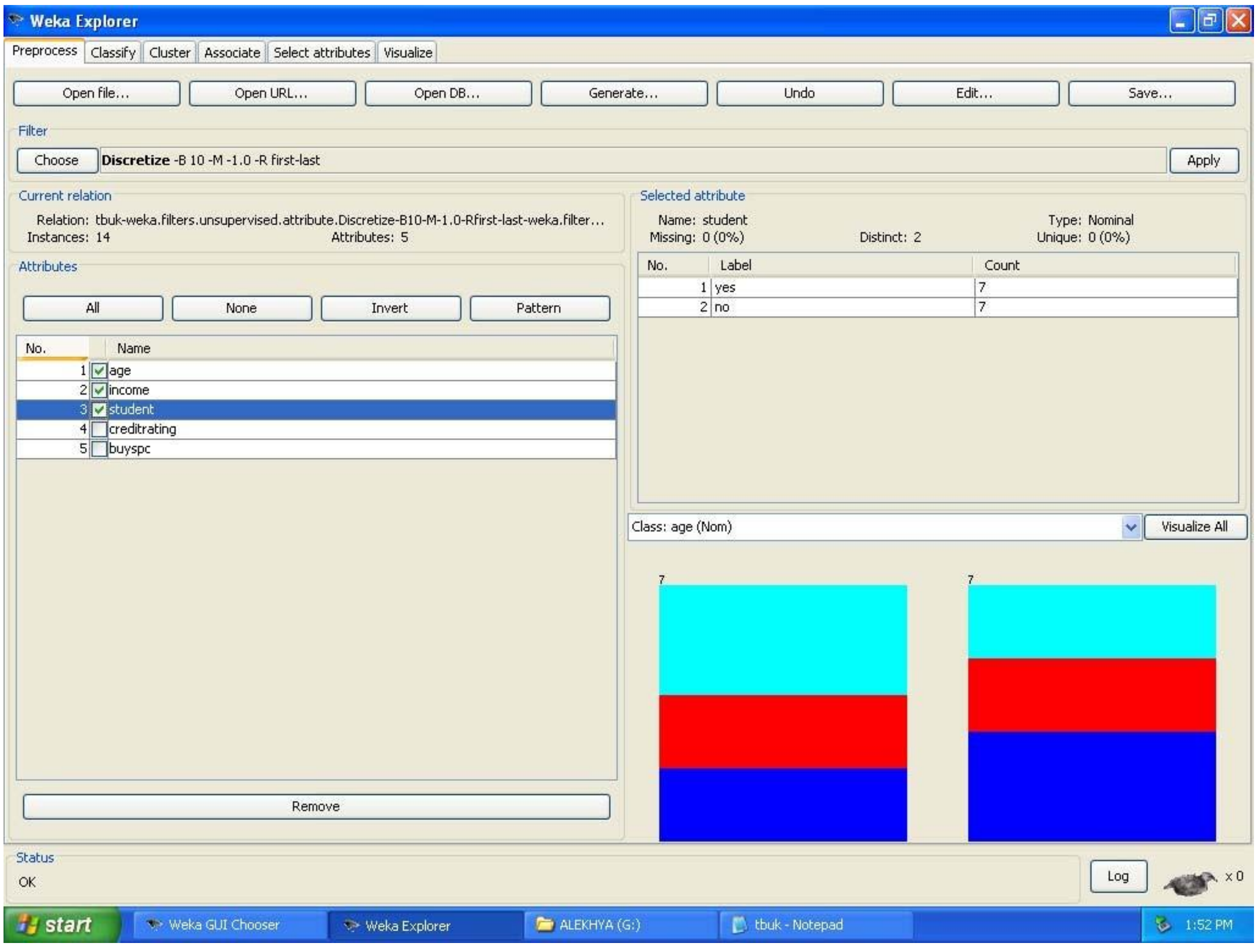
- Let us divide the values of age attribute into three bins(intervals).
- First load the dataset into weka(student.arff)
- Select the age attribute.
- Activate filter-dialog box and select "WEKA.filters.unsupervised.attribute.discretize"from the list.
- To change the defaults for the filters,click on 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 age.So we must enter '1' corresponding to the age attribute.
- Enter '3' 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 3 bins.
- Save the new working relation in a file called student-data-discretized.arff

Dataset student .arff

```
@relation student

@attribute age {<30,30-40,>40}

@attribute income {low, medium, high}

@attribute student {yes, no}

@attribute credit-rating {fair, excellent}

@attribute buyspc {yes, no}

@data

%
```

```
<30, high, no, fair, no
<30, high, no, excellent, no
30-40, high, no, fair, yes
>40, medium, no, fair, yes
>40, low, yes, fair, yes
>40, low, yes, excellent, no
30-40, low, yes, excellent, yes
<30, medium, no, fair, no
<30, low, yes, fair, no
>40, medium, yes, fair, yes
<30, medium, yes, excellent, yes
30-40, medium, no, excellent, yes
30-40, high, yes, fair, yes
>40, medium, no, excellent, no
%
```

The following screenshot shows the effect of discretization.



Filter: Discretize -B 10 -M -1.0 -R first-last

Current relation
Relation: tbuk-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-weka.filter...
Instances: 14   Attributes: 5

Attributes

No.		Name
1	✓	age
2	✓	income
3	✓	student
4		creditrating
5		buyspc

Selected attribute
Name: student   Type: Nominal
Missing: 0 (0%)   Distinct: 2   Unique: 0 (0%)

No.	Label	Count
1	yes	7
2	no	7

Class: age (Nom)

Status
OK

Practical 4 To Perform association rule mining (AprioriAlgorithm) using any Data Mining tool.

Aim: This experiment illustrates some of the basic elements of asscociation rule mining using WEKA. The sample dataset used for this example is test.arff

Step1: Open the data file in Weka Explorer. It is presumed that the required data fields have been discretized. In this example it is age attribute.

Step2: Clicking on the associate tab will bring up the interface for association rule algorithm.

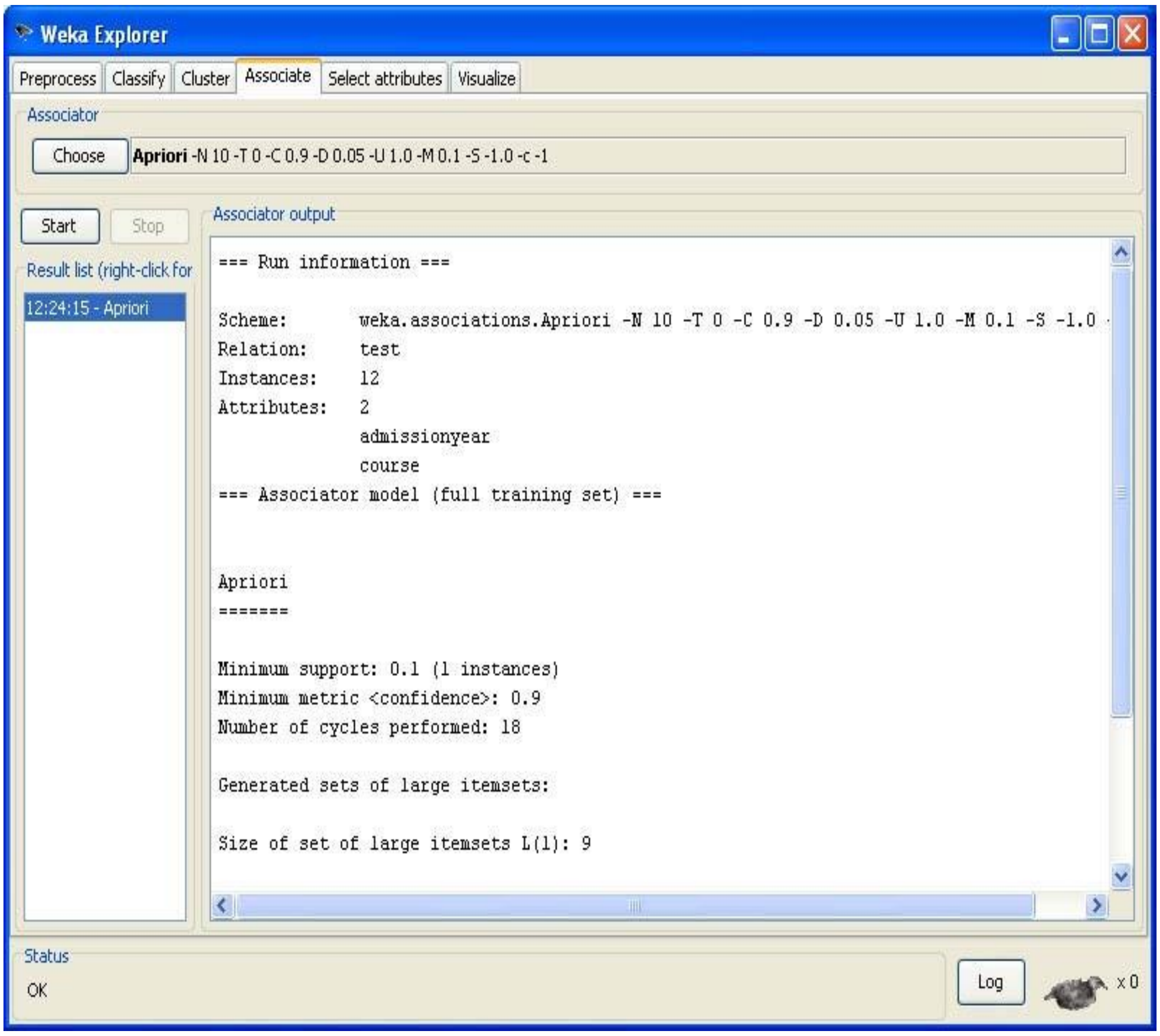
Step3: We will use apriori algorithm. This is the default algorithm.

Step4: Inorder to change the parameters for the run (example support, confidence etc) we click on the text box immediately to the right of the choose button.

Dataset test.arff

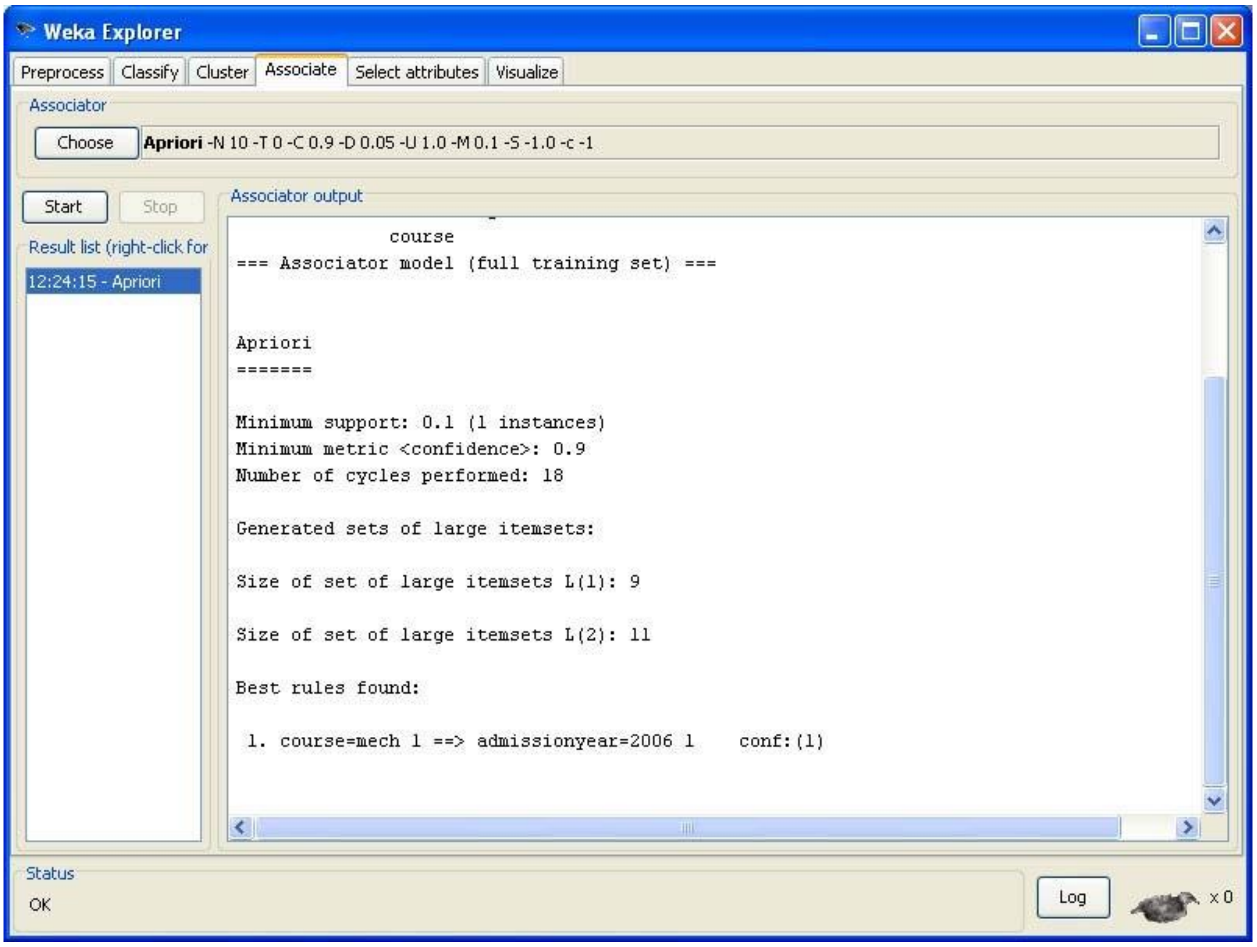
```
@relation test

@attribute admissionyear {2005,2006,2007,2008,2009,2010}

@attribute course {cse,mech,it,ece}

@data

%

2005, cse

2005, it

2005, cse

2006, mech

2006, it

2006, ece

2007, it

2007, cse

2008, it

2008, cse

2009, it

2009, ece

%
```

The following screenshot shows the association rules that were generated when apriori algorithm is applied on the given dataset.



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0
Relation:     test
Instances:    12
Attributes:   2
              admissionyear
              course
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.1 (1 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9
```



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
                course
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.1 (1 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 11

Best rules found:

 1. course=mech 1 ==> admissionyear=2006 1    conf:(1)
```

Practical 5 To perform classification using Bayes Method using any Data Mining tool.

Aim: This experiment illustrates the use of naïve bayes classifier in weka. The sample data set used in this experiment is “employee”data available at arff format. This document assumes that appropriate data pre processing has been performed.

Steps involved in this experiment:

1. We begin the experiment by loading the data (employee.arff) into weka.

Step2: next we select the “classify” tab and click “choose” button to select the “id3”classifier.

Step3: 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 his default version does perform some pruning but does not perform error pruning.

Step4: 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.

Step-5: 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.

Step-6: note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)

Step-7: 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.

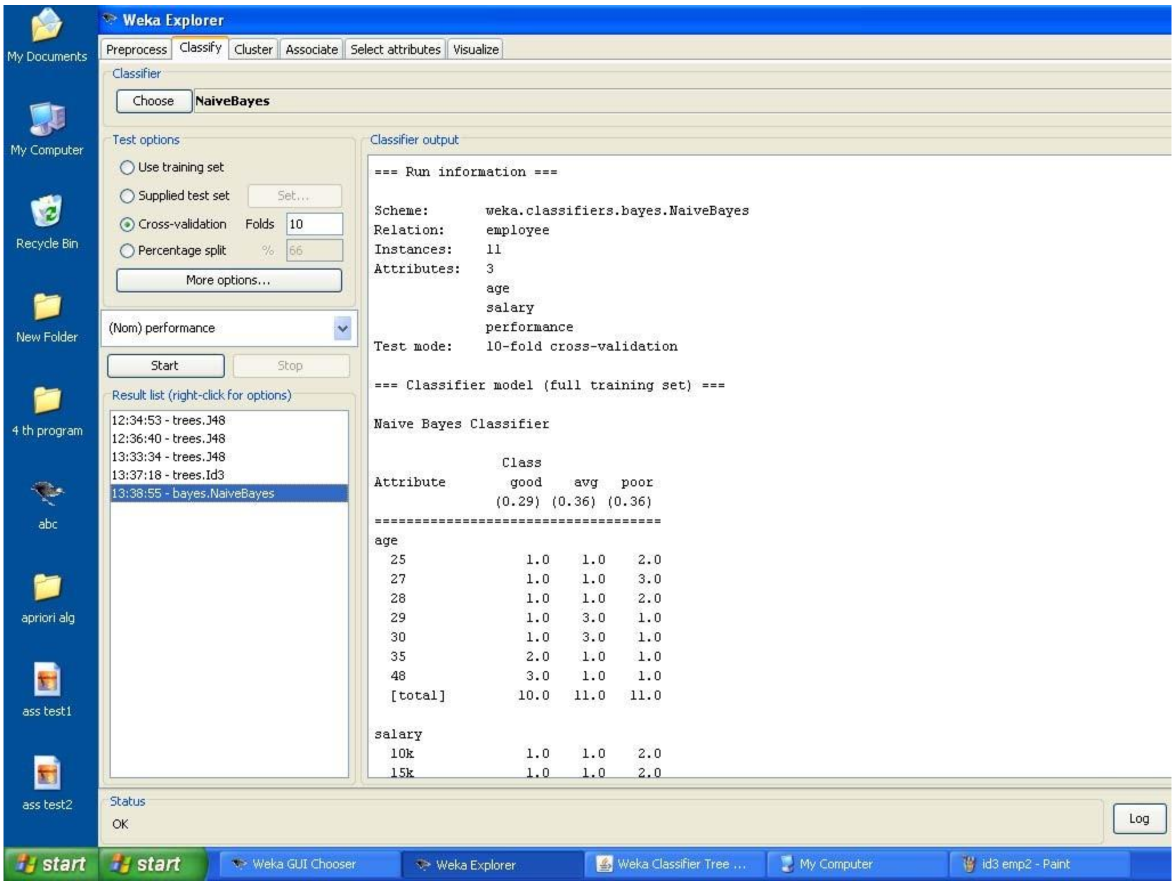
Step-8: we will use our model to classify the new instances.

Step-9: In the main panel under “text “options click the “supplied test set” radio button and then click the “set” button. This will show pop-up window which will allow you to open the file containing test instances.

Data set employee.arff:

```
@relation employee

@attribute age {25, 27, 28, 29, 30, 35, 48}

@attribute salary{10k,15k,17k,20k,25k,30k,35k,32k}

@attribute performance {good, avg, poor}

@data

%

25, 10k, poor

27, 15k, poor

27, 17k, poor

28, 17k, poor

29, 20k, avg

30, 25k, avg

29, 25k, avg

30, 20k, avg

35, 32k, good

48, 34k, good

48, 32k, good

%
```

The following screenshot shows the classification rules that were generated when naive bayes algorithm is applied on the given dataset.



Classifier: Choose NaiveBayes

Test options:
- Use training set
- Supplied test set
- [✓ Cross-validation] Folds 10
- Percentage split % 66

(Nom) performance

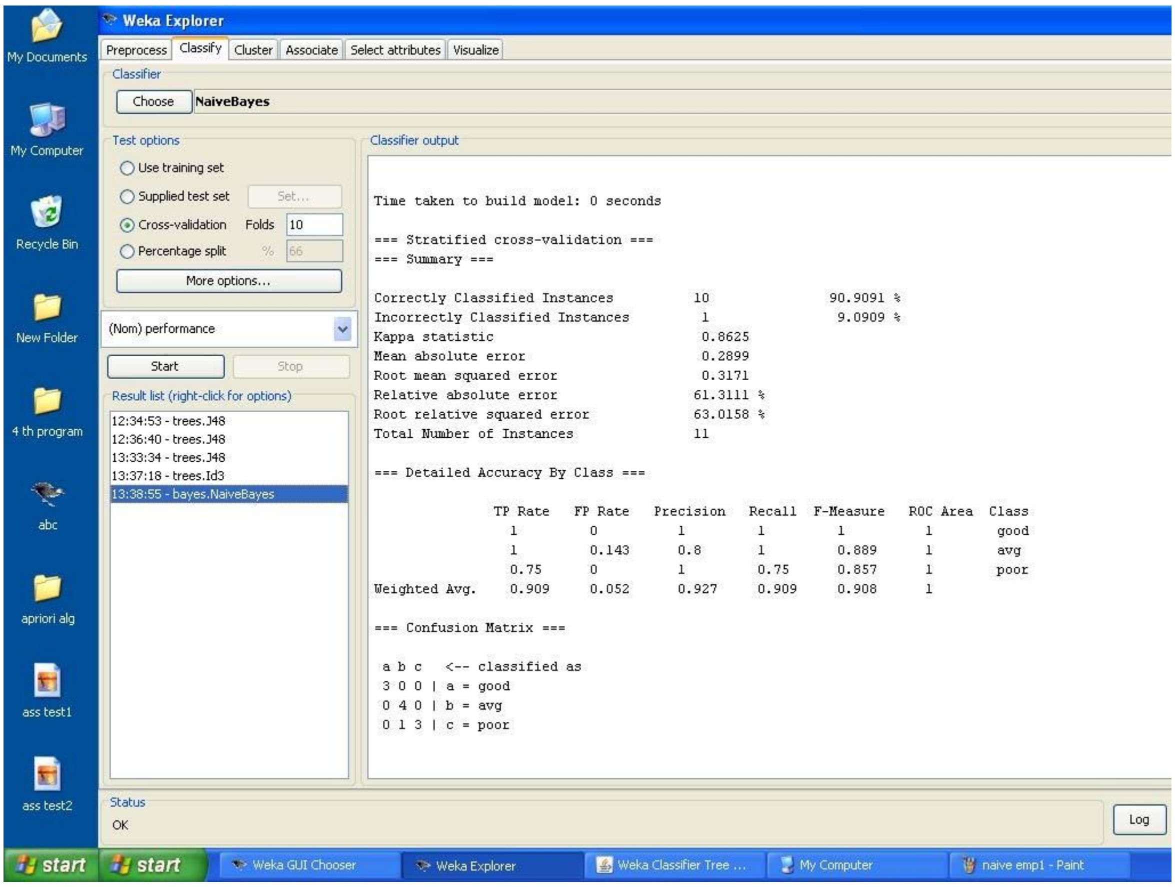
Result list (right-click for options)

```
12:34:53 - trees.J48
12:36:40 - trees.J48
13:33:34 - trees.J48
13:37:18 - trees.Id3
13:38:55 - bayes.NaiveBayes
```

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     employee
Instances:    11
Attributes:   3
              age
              salary
              performance
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Naive Bayes Classifier

                    Class
Attribute           good    avg   poor
                  (0.29) (0.36) (0.36)
====================================
age
  25                 1.0    1.0    2.0
  27                 1.0    1.0    3.0
  28                 1.0    1.0    2.0
  29                 1.0    3.0    1.0
  30                 1.0    3.0    1.0
  35                 2.0    1.0    1.0
  48                 3.0    1.0    1.0
  [total]           10.0   11.0   11.0

salary
  10k                1.0    1.0    2.0
  15k                1.0    1.0    2.0
```

Status
OK



Classifier: NaiveBayes

Test options: Use training set; Supplied test set; [✓ Cross-validation] Folds 10; Percentage split % 66; More options...

(Nom) performance

```
12:34:53 - trees.J48
12:36:40 - trees.J48
13:33:34 - trees.J48
13:37:18 - trees.Id3
13:38:55 - bayes.NaiveBayes
```

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          10               90.9091 %
Incorrectly Classified Instances         1                9.0909 %
Kappa statistic                          0.8625
Mean absolute error                      0.2899
Root mean squared error                  0.3171
Relative absolute error                 61.3111 %
Root relative squared error             63.0158 %
Total Number of Instances               11

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0          1         1         1          1        good
                 1         0.143      0.8       1         0.889      1        avg
                 0.75      0          1         0.75      0.857      1        poor
Weighted Avg.    0.909     0.052      0.927     0.909     0.908      1

=== Confusion Matrix ===

 a b c   <-- classified as
 3 0 0 | a = good
 0 4 0 | b = avg
 0 1 3 | c = poor
```

Practical 6 To Perform K-means clustering techniques for data mining on data set using any Data Mining tool.

Aim: This experiment illustrates the use of simple k-mean clustering with Weka explorer. The sample data set used for this example is based on the iris data available in ARFF format. This document assumes that appropriate preprocessing has been performed. This iris dataset includes 150 instances.

Steps involved in this Experiment

Step 1: Run the Weka explorer and load the data file iris.arff in preprocessing interface.

Step 2: Inorder to perform clustering select the ‘cluster’ tab in the explorer and click on the choose button. This step results in a dropdown list of available clustering algorithms.

Step 3 : In this case we select ‘simple k-means’.

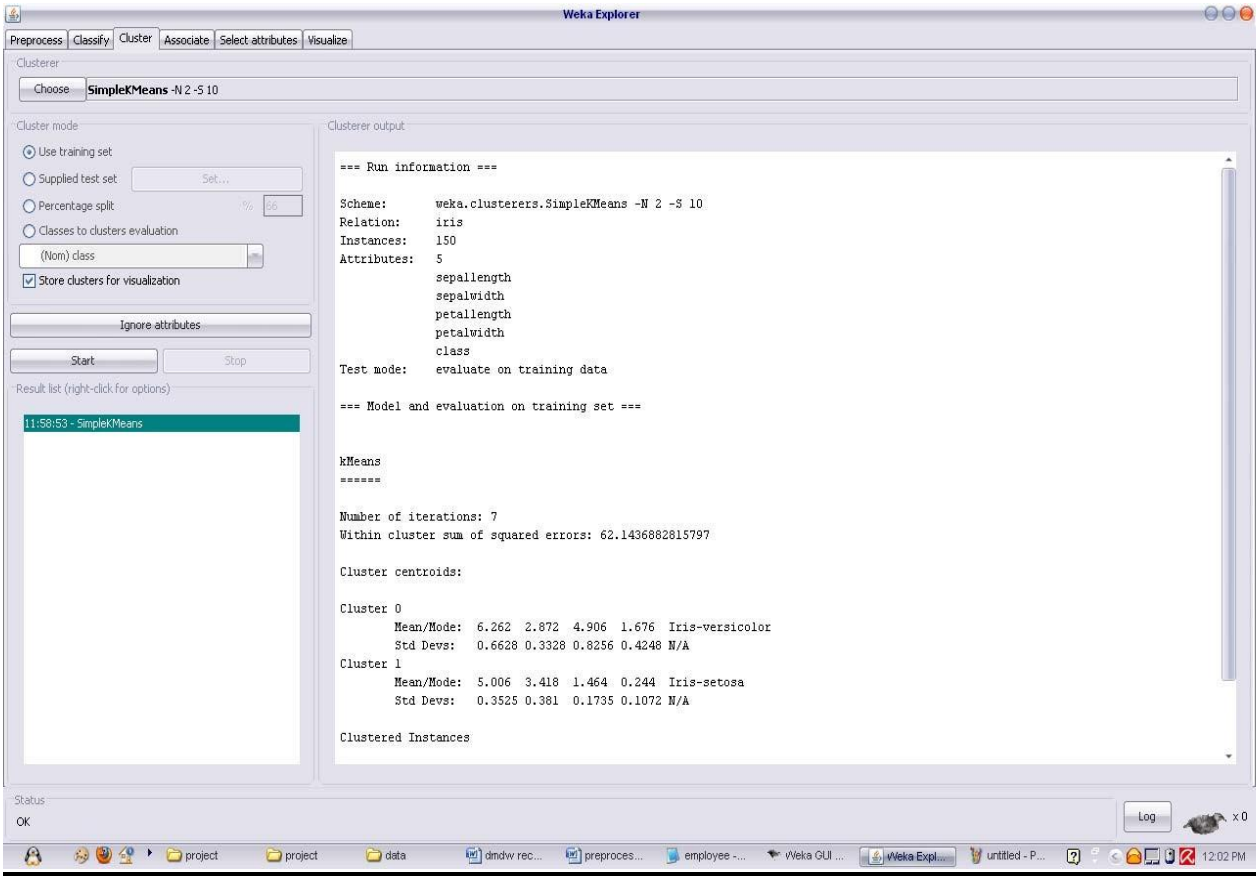
Step 4: Next click in text button to the right of the choose button to get popup window shown in the screenshots. In this window we enter six on the number of clusters and we leave the value of the seed on as it is. The seed value is used in generating a random number which is used for making the internal assignments of instances of clusters.

Step 5 : 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 training set option is selected and then we click ‘start’ button. This process and resulting window are shown in the following screenshots.

Step 6 : 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. Here clusters centroid are means vectors for each clusters. This clusters can be used to characterized the cluster.For eg, the centroid of cluster1 shows the class iris.versicolor mean value of the sepal length is 5.4706, sepal width 2.4765, petal width 1.1294, petal length 3.7941.

Step 7: 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.

The following screenshot shows the clustering rules that were generated when simple k means algorithm is applied on the given dataset.



Clusterer: SimpleKMeans -N 2 -S 10

Cluster mode: [✓ Use training set]

[✓ Store clusters for visualization]

Result list: 11:58:53 - SimpleKMeans

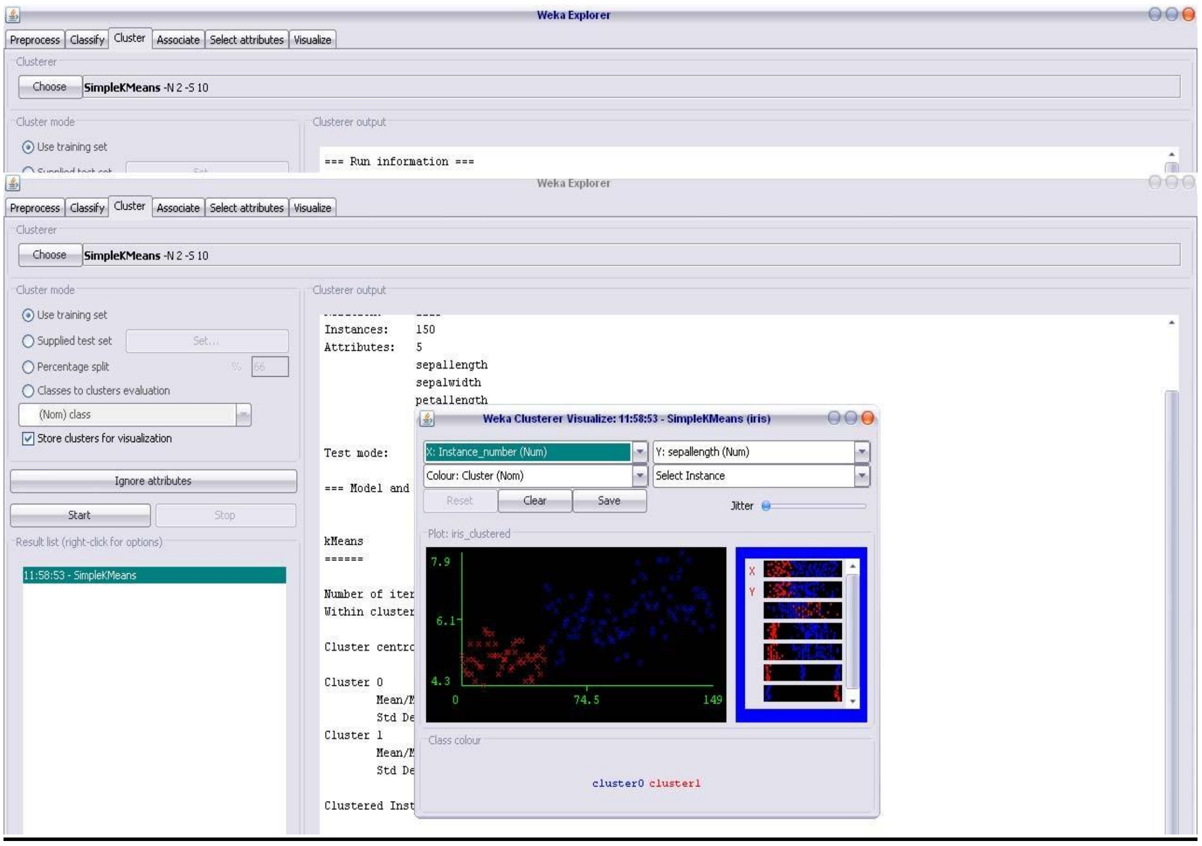
Status: OK

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -N 2 -S 10
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    evaluate on training data

=== Model and evaluation on training set ===


kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797

Cluster centroids:

Cluster 0
        Mean/Mode:  6.262  2.872  4.906  1.676  Iris-versicolor
        Std Devs:   0.6628 0.3328 0.8256 0.4248 N/A
Cluster 1
        Mean/Mode:  5.006  3.418  1.464  0.244  Iris-setosa
        Std Devs:   0.3525 0.381  0.1735 0.1072 N/A

Clustered Instances
```

Interpretation of the above visualization

From the above visualization, we can understand the distribution of sepal length and petal length in each cluster. For instance, for each cluster is dominated by petal length. In this case by changing the color dimension to other attributes we can see their distribution with in each of the cluster.

Step 8: We can assure that resulting dataset which included each instance along with its assign cluster. To do so we click the save button in the visualization window and save the result iris k-mean .The top portion of this file is shown in the following figure.



Clusterer: SimpleKMeans -N 2 -S 10

Cluster mode: [✓ Use training set] / Supplied test set / Percentage split % 66 / Classes to clusters evaluation (Nom) class / [✓ Store clusters for visualization]

Result list (right-click for options): 11:58:53 - SimpleKMeans

X: Instance_number (Num) | Y: sepallength (Num) | Colour: Cluster (Nom) | Select Instance

```
=== Run information ===

Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength

Test mode:

=== Model and

kMeans
======

Number of iter
Within cluster

Cluster centro

Cluster 0
        Mean/M
        Std De
Cluster 1
        Mean/M
        Std De

Clustered Inst
```

Clustering rule process on dataset student.arff using simple k- means

Aim: This experiment illustrates the use of simple k-mean clustering with Weka explorer. The sample data set used for this example is based on the student data available in ARFF format. This document assumes that appropriate preprocessing has been performed. This istudent dataset includes 14 instances.

Steps involved in this Experiment

Step 1: Run the Weka explorer and load the data file student.arff in preprocessing interface.

Step 2: Inorder to perform clustering select the 'cluster' tab in the explorer and click on the choose button. This step results in a dropdown list of available clustering algorithms.

Step 3 : In this case we select 'simple k-means'.

Step 4: Next click in text button to the right of the choose button to get popup window shown in the screenshots. In this window we enter six on the number of clusters and we leave the value of the seed on as it is. The seed value is used in generating a random number which is used for making the internal assignments of instances of clusters.

Step 5 : 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 training set option is selected and then we click 'start' button. This process and resulting window are shown in the following screenshots.

Step 6 : 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. Here clusters centroid are means vectors for each clusters. This clusters can be used to characterized the cluster.

Step 7: 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.

Interpretation of the above visualization

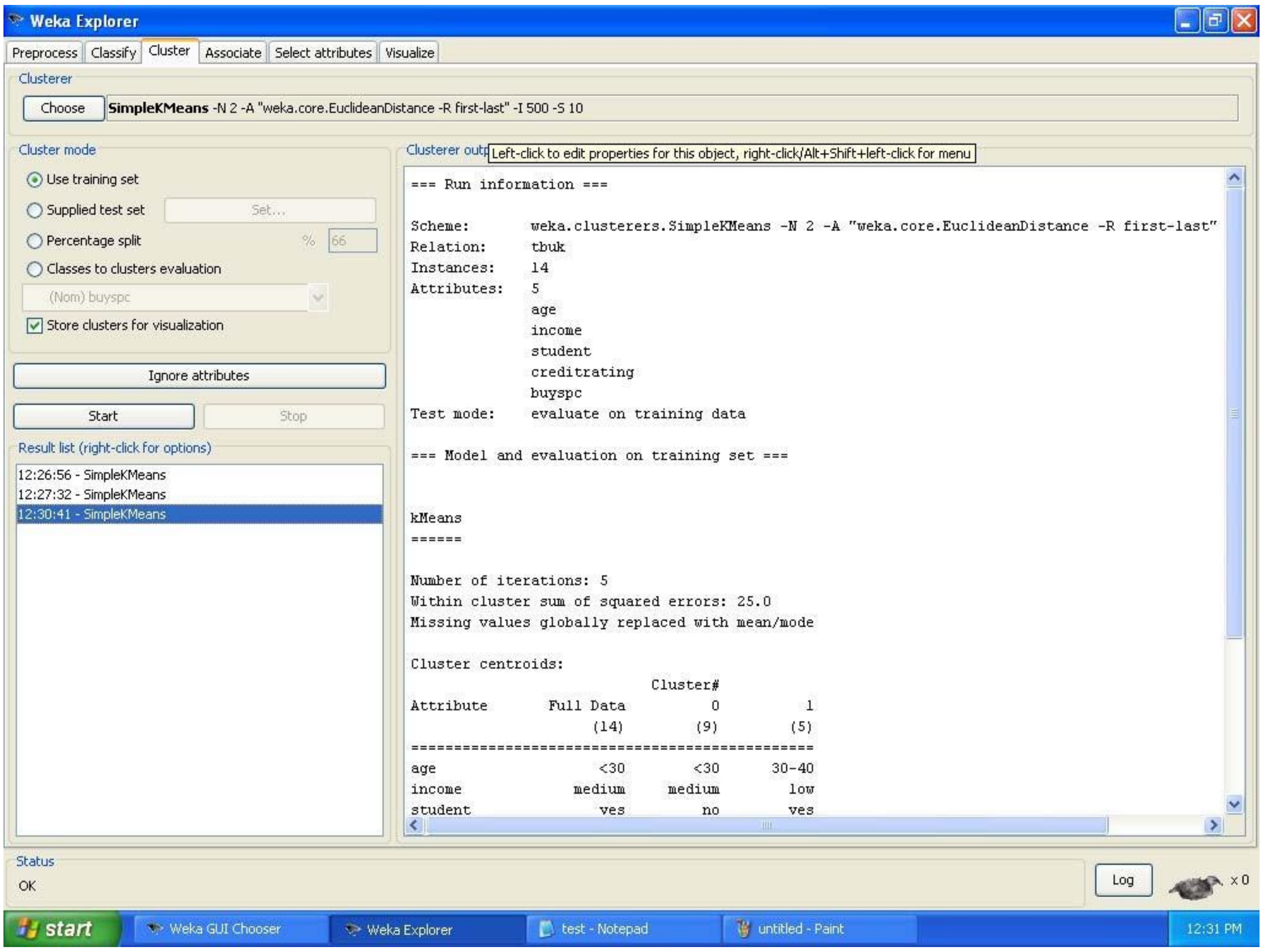
From the above visualization, we can understand the distribution of age and instance number in each cluster. For instance, for each cluster is dominated by age. In this case by changing the color dimension to other attributes we can see their distribution with in each of the cluster.

Step 8: We can assure that resulting dataset which included each instance along with its assign cluster. To do so we click the save button in the visualization window and save the result student k-mean .The top portion of this file is shown in the following figure.

Dataset student .arff

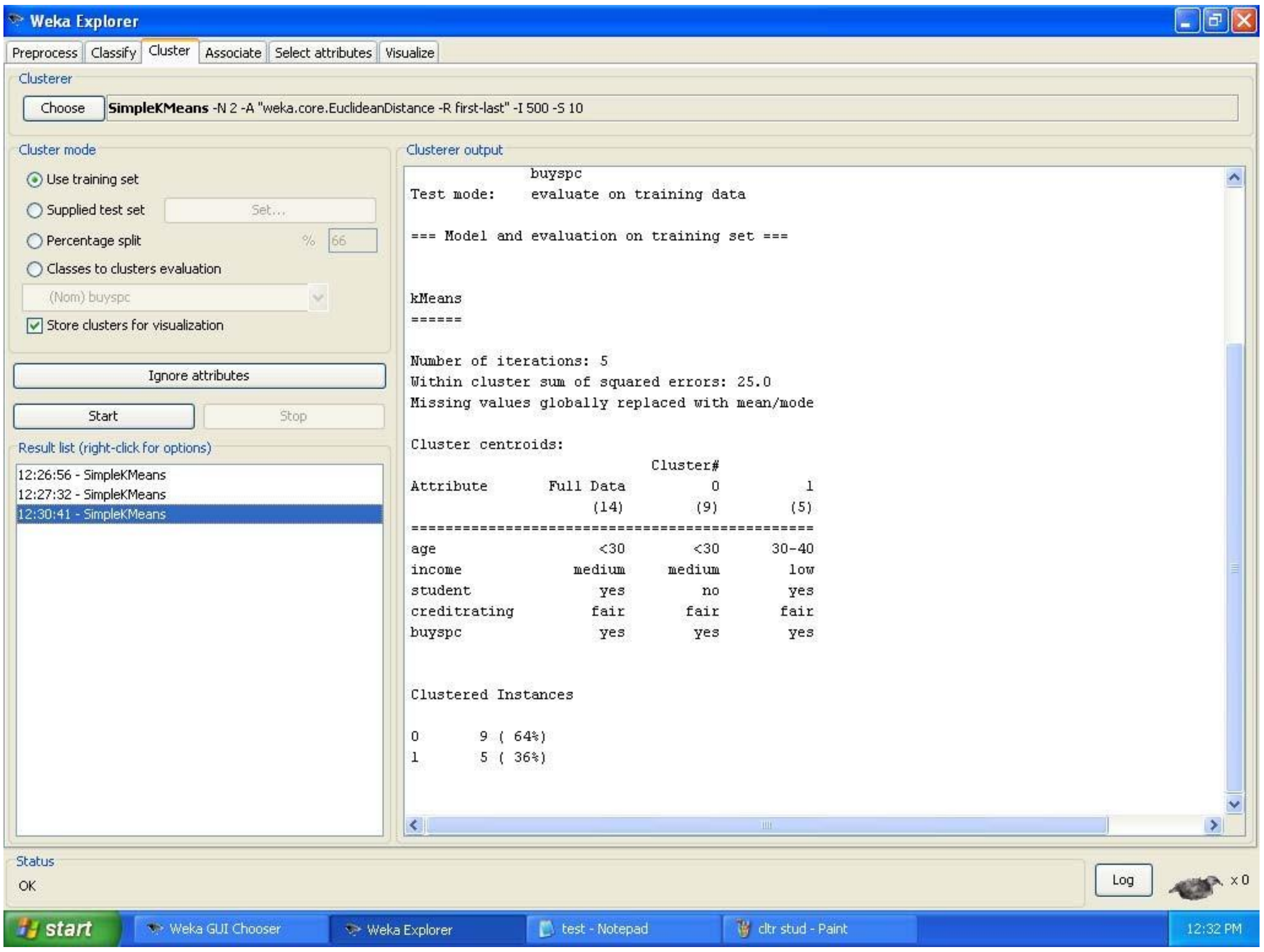
```
@relation student

@attribute age {<30,30-40,>40}

@attribute income {low,medium,high}

@attribute student {yes,no}

@attribute credit-rating {fair,excellent}

@attribute buyspc {yes,no}

@data

%

<30, high, no, fair, no

<30, high, no, excellent, no

30-40, high, no, fair, yes

>40, medium, no, fair, yes

>40, low, yes, fair, yes

>40, low, yes, excellent, no

30-40, low, yes, excellent, yes

<30, medium, no, fair, no

<30, low, yes, fair, no

>40, medium, yes, fair, yes

<30, medium, yes, excellent, yes

30-40, medium, no, excellent, yes

30-40, high, yes, fair,  yes

>40, medium, no, excellent, no

%
```

The following screenshot shows the clustering rules that were generated when simple k-means algorithm is applied on the given dataset.



SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last"
Relation:     tbuk
Instances:    14
Attributes:   5
              age
              income
              student
              creditrating
              buyspc
Test mode:    evaluate on training data

=== Model and evaluation on training set ===


kMeans
======

Number of iterations: 5
Within cluster sum of squared errors: 25.0
Missing values globally replaced with mean/mode

Cluster centroids:
                            Cluster#
Attribute       Full Data          0          1
                     (14)        (9)        (5)
===============================================
age                   <30        <30      30-40
income             medium     medium        low
student               yes         no        yes
```



SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

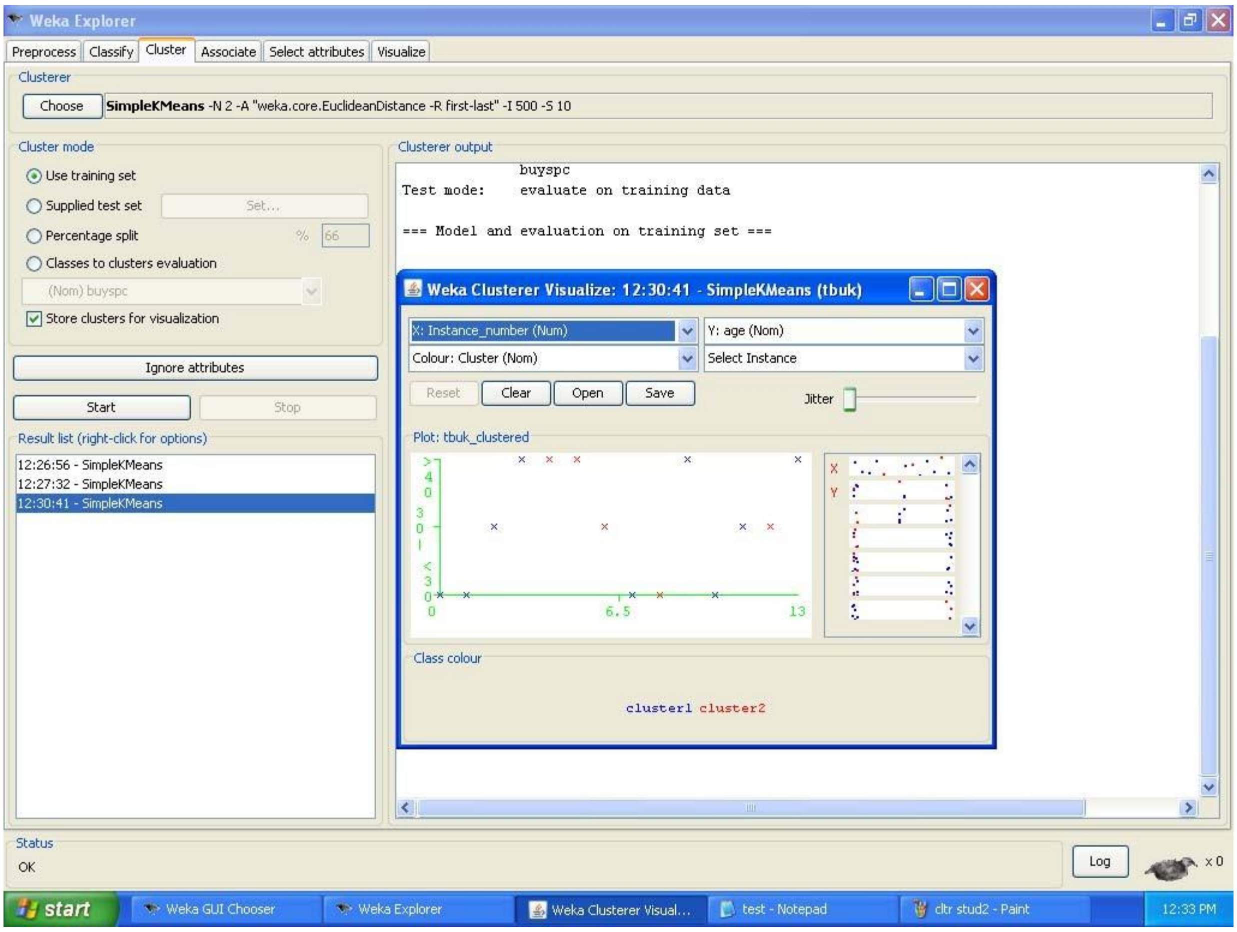
(●) Use training set
( ) Supplied test set
( ) Percentage split % 66
( ) Classes to clusters evaluation
(Nom) buyspc
[✓] Store clusters for visualization

Result list (right-click for options)
12:26:56 - SimpleKMeans
12:27:32 - SimpleKMeans
12:30:41 - SimpleKMeans

Clusterer output

```
              buyspc
Test mode:    evaluate on training data

=== Model and evaluation on training set ===


kMeans
======

Number of iterations: 5
Within cluster sum of squared errors: 25.0
Missing values globally replaced with mean/mode

Cluster centroids:
                            Cluster#
Attribute       Full Data          0          1
                     (14)        (9)        (5)
===============================================
age                   <30        <30      30-40
income             medium     medium        low
student               yes         no        yes
creditrating         fair       fair       fair
buyspc                yes        yes        yes


Clustered Instances

0       9 ( 64%)
1       5 ( 36%)
```

Status
OK



SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

```
                buyspc
Test mode:      evaluate on training data

=== Model and evaluation on training set ===
```

Result list (right-click for options)

12:26:56 - SimpleKMeans
12:27:32 - SimpleKMeans
12:30:41 - SimpleKMeans

Weka Clusterer Visualize: 12:30:41 - SimpleKMeans (tbuk)

X: Instance_number (Num)
Y: age (Nom)
Colour: Cluster (Nom)

Plot: tbuk_clustered

Class colour

cluster1 cluster2

Status

OK

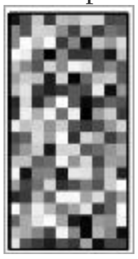
Practical 7 To interpret and visualize the output of data mining using any Data mining tool.

Data Visualization

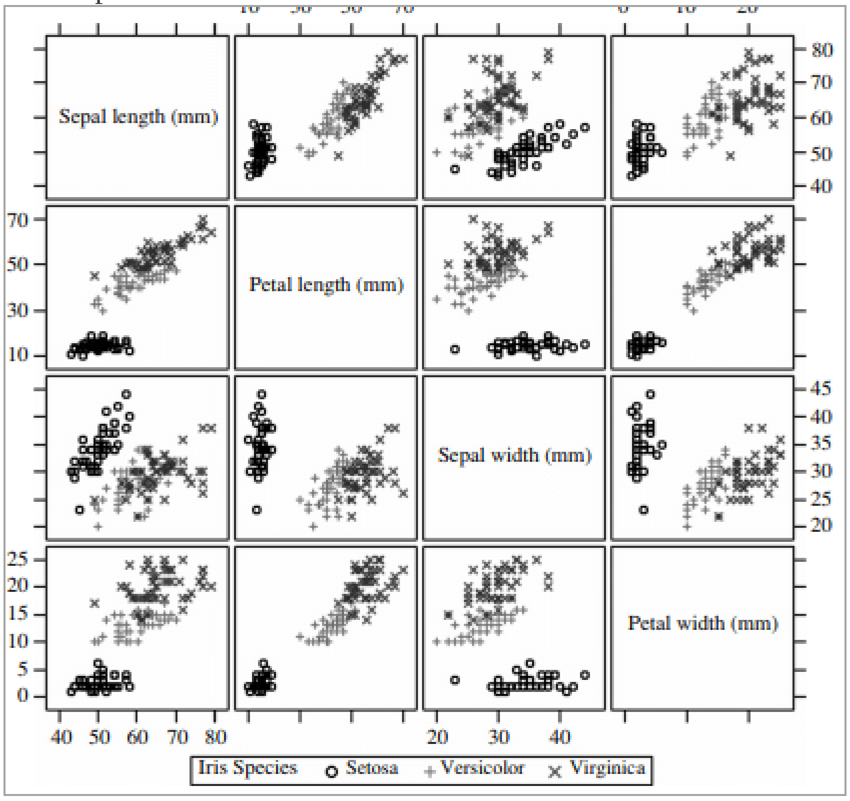
The method of representing data through graphs and plots with the aim to understand data clearly is data visualization.

There are many ways to represent data. Some of them are as follows:

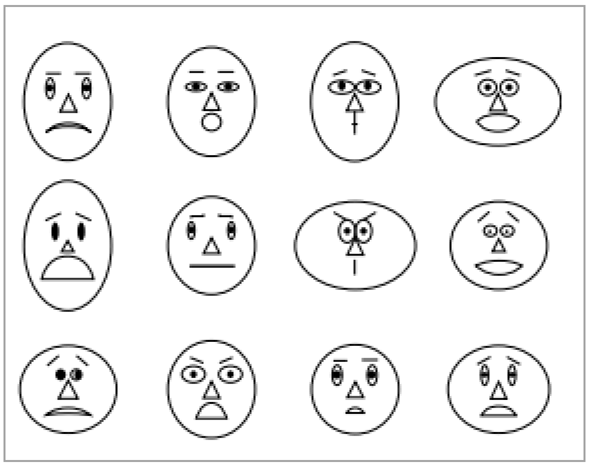
#1) Pixel Oriented Visualization: Here the color of the pixel represents the dimension value. The color of the pixel represents the corresponding values.



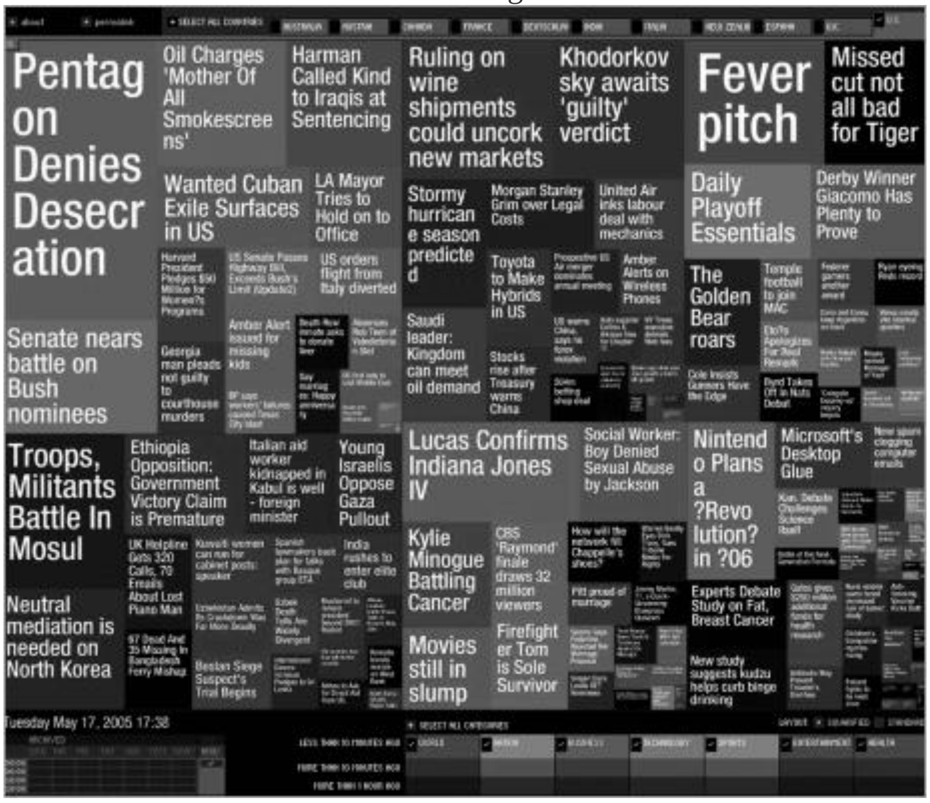
#2) Geometric Representation: The multidimensional datasets are represented in 2D, 3D, and 4D scatter plots.



#3) Icon Based Visualization: The data is represented using Chernoff's faces and stick figures. Chernoff's faces use the human mind's ability to recognize facial characteristics and differences between them. The stick figure uses 5 stick figures to represent multidimensional data.



#4) Hierarchical Data Visualization: The datasets are represented using treemaps. It represents hierarchical data as a set of nested triangles.


Pentagon Denies Desecration
Oil Charges 'Mother Of All Smokescreens'
Harman Called Kind to Iraqis at Sentencing
Ruling on wine shipments could uncork new markets
Khodorkovsky awaits 'guilty' verdict
Fever pitch
Missed cut not all bad for Tiger
Wanted Cuban Exile Surfaces in US
LA Mayor Tries to Hold on to Office
Stormy hurricane season predicted
Daily Playoff Essentials
Derby Winner Giacomo Has Plenty to Prove
Senate nears battle on Bush nominees
The Golden Bear roars
Troops, Militants Battle In Mosul
Ethiopia Opposition: Government Victory Claim is Premature
Young Israelis Oppose Gaza Pullout
Lucas Confirms Indiana Jones IV
Social Worker: Boy Denied Sexual Abuse by Jackson
Nintendo Plans a ?Revolution? in ?06
Microsoft's Desktop Glue
Neutral mediation is needed on North Korea
Kylie Minogue Battling Cancer
Movies still in slump
Firefighter Tom is Sole Survivor
Experts Debate Study on Fat, Breast Cancer
Tuesday May 17, 2005 17:38

Data Visualization Using WEKA Explorer

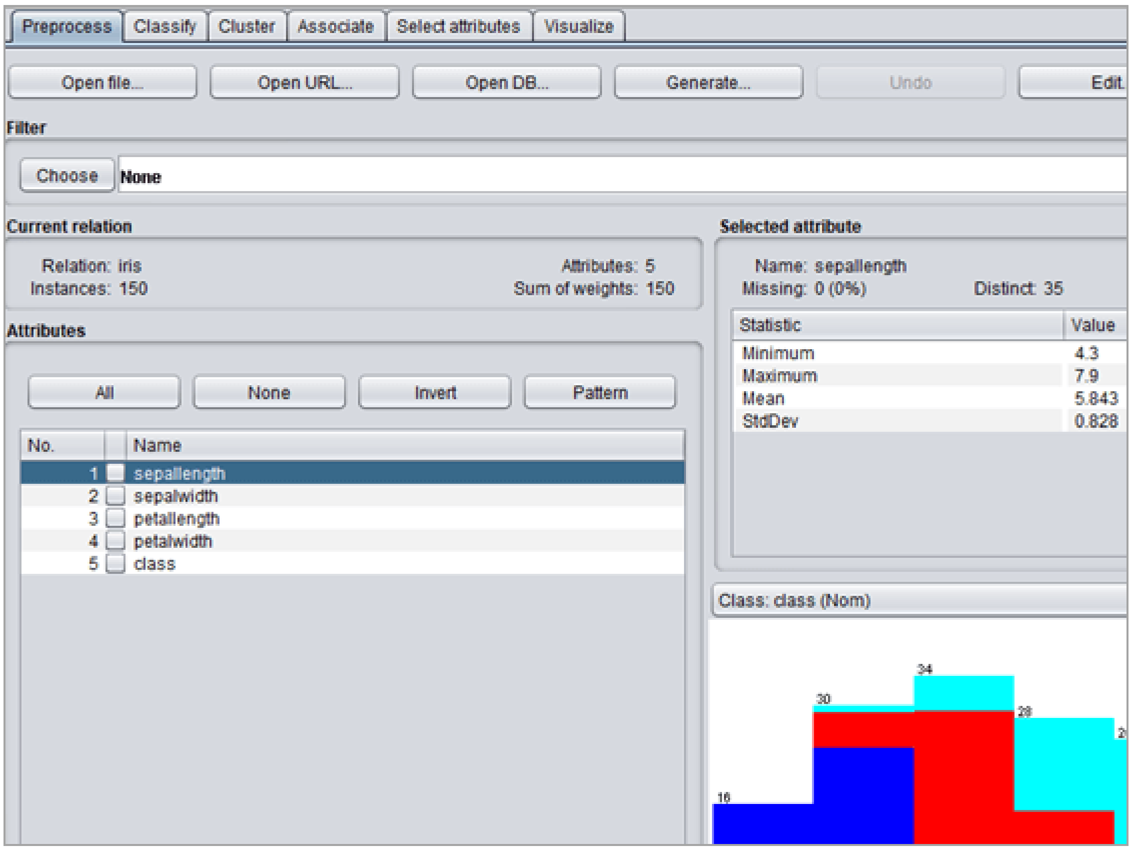
Data Visualization using WEKA is done on the IRIS.arff dataset.

Steps involved are as follows:

#1) Go to the Preprocess tab and open IRIS.arff dataset.

#2) The dataset has 4 attributes and 1 class label. **The attributes in this dataset are:**

- **Sepallength :** Type -numeric
- **Sepalwidth:** Type- numeric
- **Petalength:** Type-numeric
- **Petalwidth:** Type-numeric
- **Class:** Type-nominal



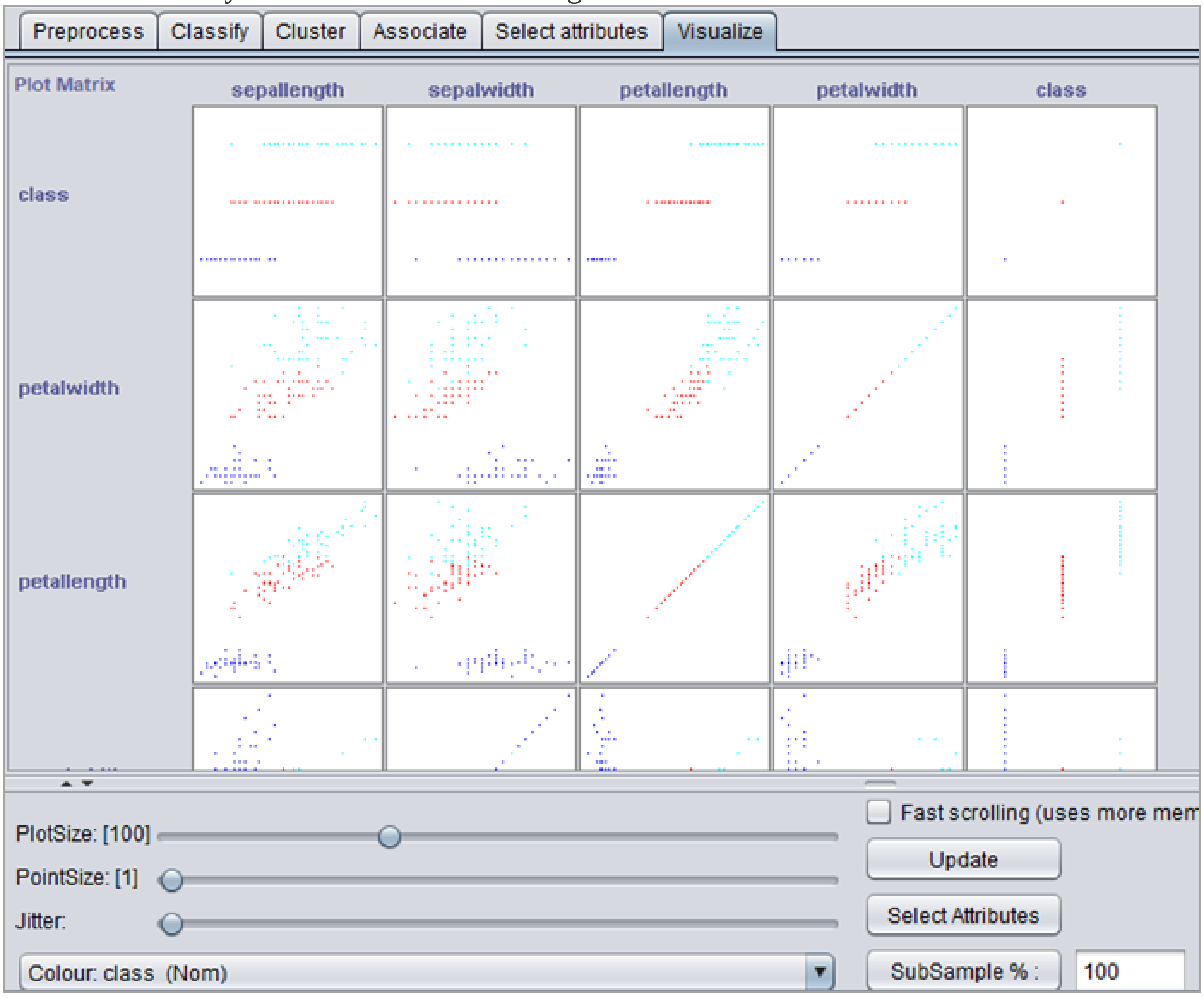
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150

Name: sepallength
Missing: 0 (0%)
Distinct: 35

Statistic	Value
Minimum	4.3
Maximum	7.9
Mean	5.843
StdDev	0.828

No.	Name
1	sepallength
2	sepalwidth
3	petallength
4	petalwidth
5	class

#3) To visualize the dataset, go to the Visualize tab. The tab shows the attributes plot matrix. The dataset attributes are marked on the x-axis and y-axis while the instances are plotted. The box with x-axis attribute and y-axis attribute can be enlarged.

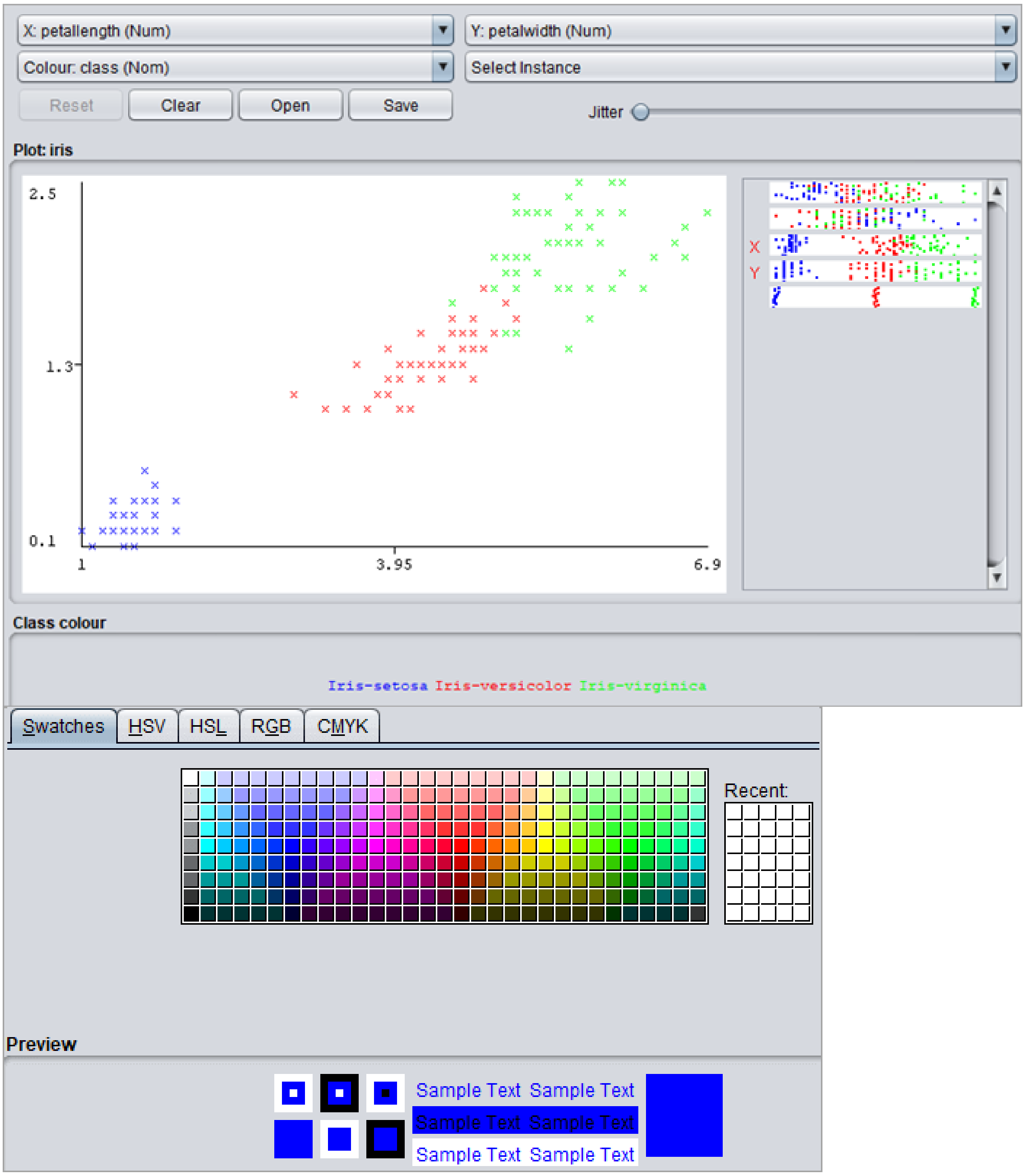


PlotSize: [100]
PointSize: [1]
Jitter:
Colour: class (Nom)
SubSample % : 100

#4) Click on the box of the plot to enlarge. **For example,** x: petallength and y:petalwidth. The class labels are represented in different colors.

- Class label- Iris-setosa: blue color
- Class label- Iris-versicolor: red
- Class label-Iris-virginica-green

These colors can be changed. To change the color, click on the class label at the bottom, a color window will appear.



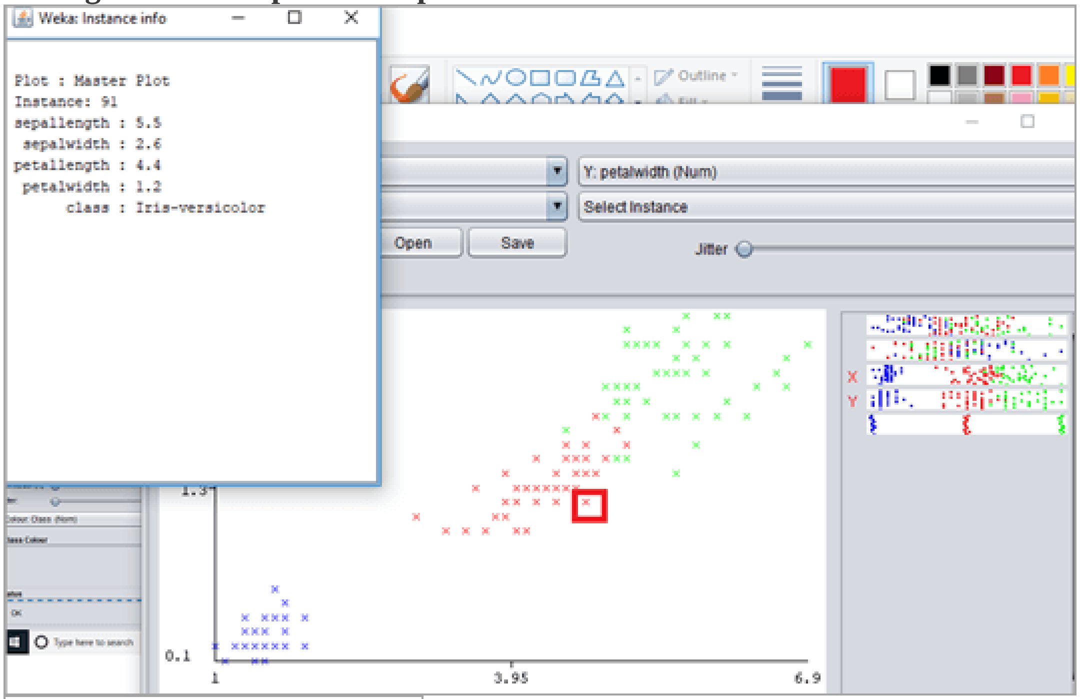
#5) Click on the instance represented by 'x' in the plot. It will give the instance details. **For example:**

- **Instance number:** 91

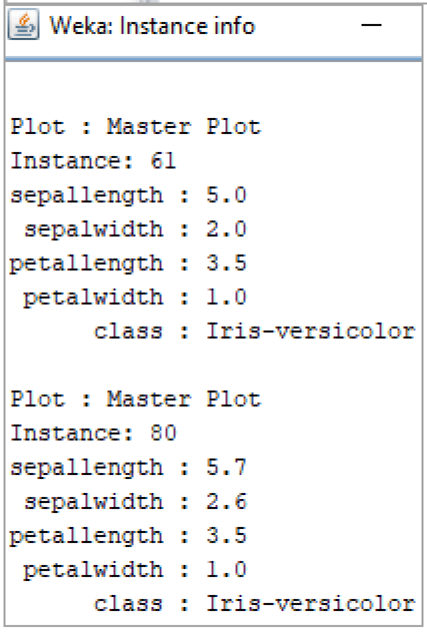
- **Sepalength:** 5.5
- **Sepalwidth:** 2.6
- **Petalength:** 4.4
- **Petalwidth:** 1.2
- **Class:** Iris-versicolor

Some of the points in the plot appear darker than other points. These points represent 2 or more instances with the same class label and the same value of attributes plotted on the graph such as petalwidth and petallength.

The figure below represents a point with 2 instance information.

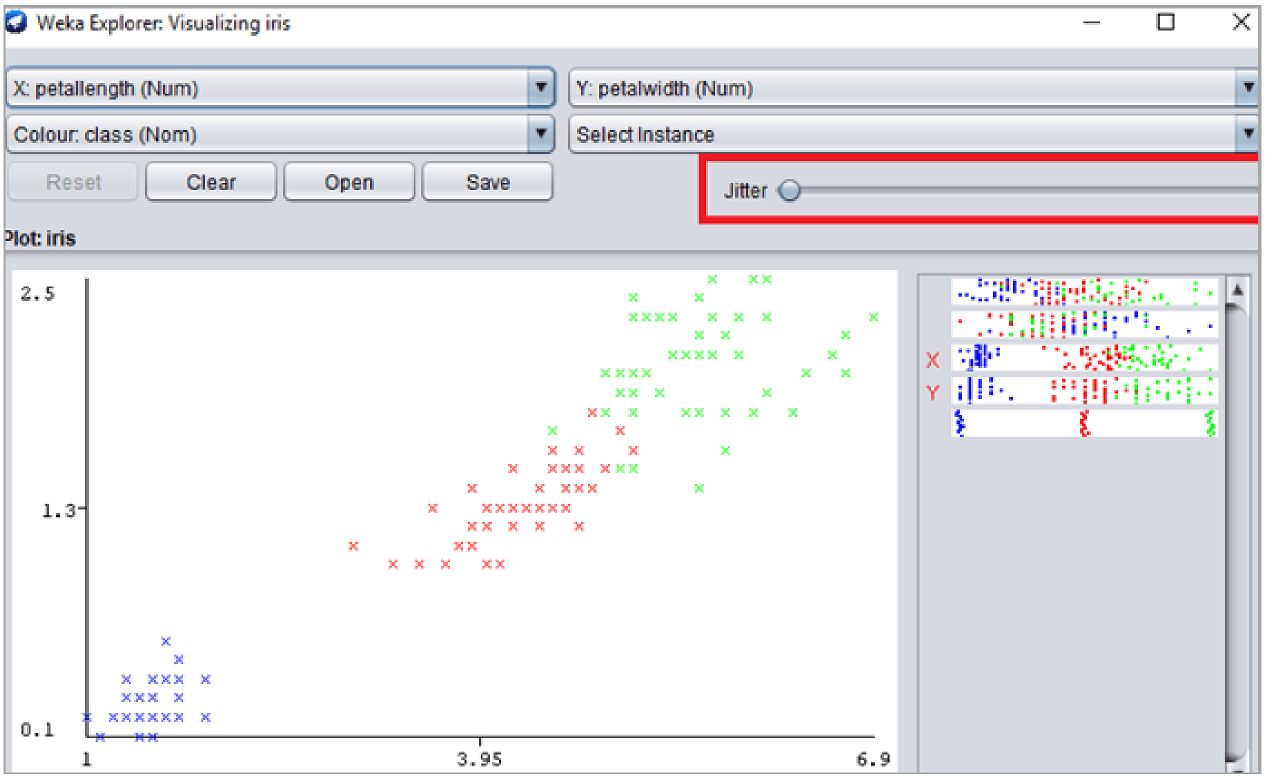


```
Plot : Master Plot
Instance: 91
sepallength : 5.5
 sepalwidth : 2.6
petallength : 4.4
 petalwidth : 1.2
      class : Iris-versicolor
```

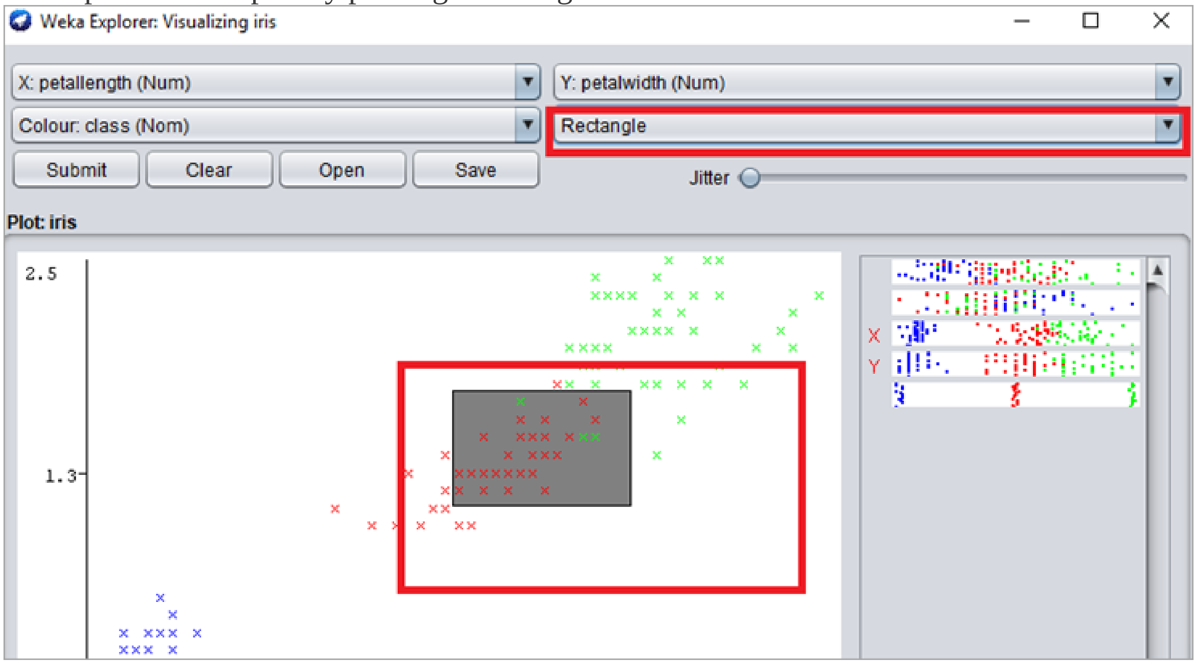


```
Plot : Master Plot
Instance: 61
sepallength : 5.0
 sepalwidth : 2.0
petallength : 3.5
 petalwidth : 1.0
      class : Iris-versicolor

Plot : Master Plot
Instance: 80
sepallength : 5.7
 sepalwidth : 2.6
petallength : 3.5
 petalwidth : 1.0
      class : Iris-versicolor
```

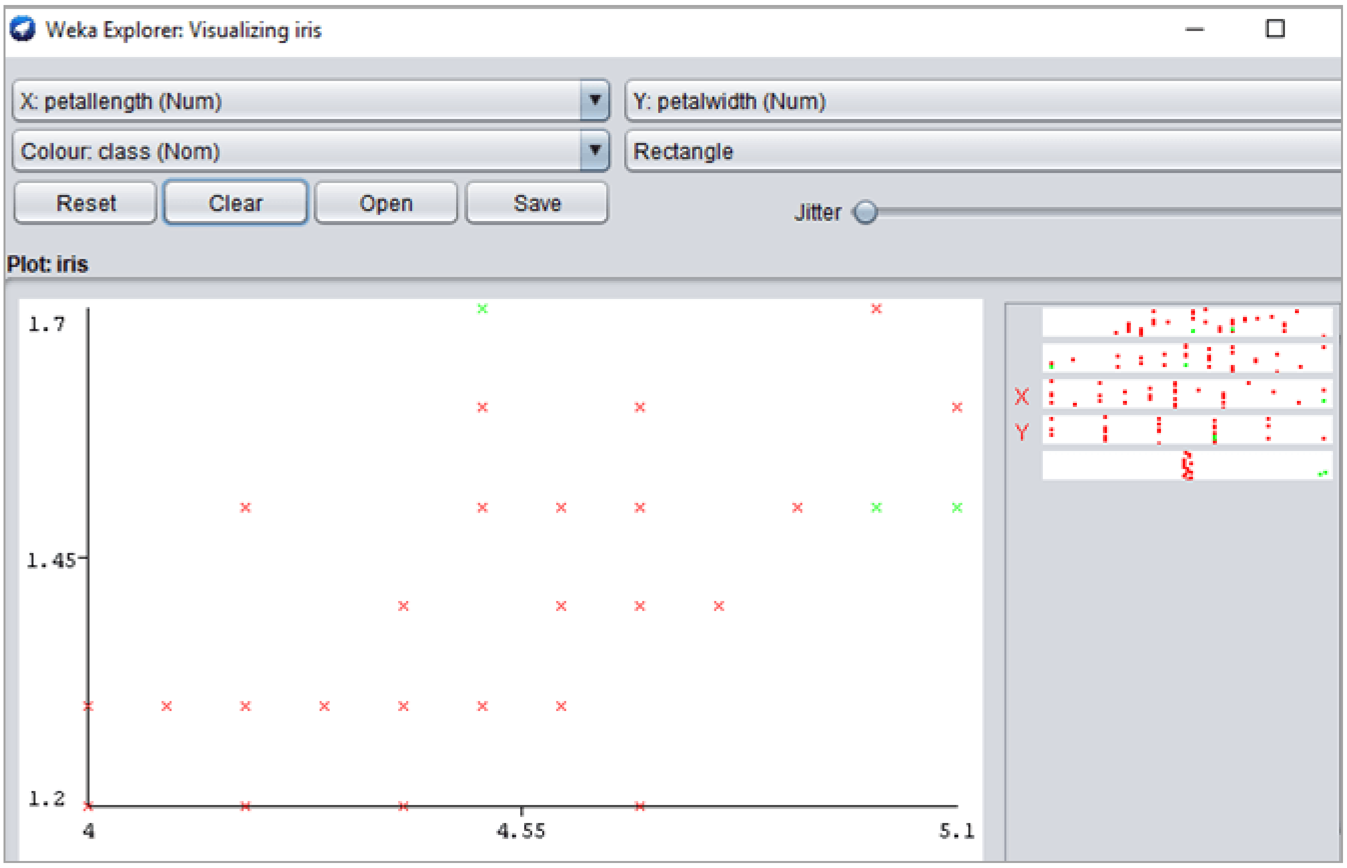
#6) The X and Y-axis attributes can be changed from the right panel in Visualize graph. The user can view different plots.
#7) The Jitter is used to add randomness to the plot. Sometimes the points overlap. With jitter, the darker spots represent multiple instances.



#8) To get a clearer view of the dataset and remove outliers, the user can select an instance from the dropdown. Click on "select instance" dropdown. Choose "Rectangle". With this, the user will be able to select points in the plot by plotting a rectangle.



#9) Click on "Submit". Only the selected dataset points will be displayed and the other points will be excluded from the graph.
The figure below shows the points from the selected rectangular shape. The plot represents points with only 3 class labels. The user can click on "Save" to save the dataset or "Reset" to select another instance. The dataset will be saved in a separate .ARFF file.



Output:
Data visualization using WEKA is simplified with the help of the box plot. The user can view any level of granularity. The attributes are plotted on X-axis and y-axis while the instances are plotted against the X and Y-axis. Some points represent multiple instances which are represented by points with dark color.