



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

(Affiliated to Osmania University & Approved by AICTE, New Delhi)



LABORATORY MANUAL

DATA MINING LABORATORY

BE VII Semester (CBCS): 2020-21

NAME: _____

ROLL NO: _____

BRANCH: _____ SEM: _____

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Empower youth- Architects of Future World



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

VISION

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

MISSION

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

**DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING**

LABORATORY MANUAL

DATA MINING LAB

Prepared

By

Mr. T. PRAVEEN KUMAR,

Assistant Professor.



METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION & MISSION

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

- To offer flexible programs of study with collaborations to suit industry needs.
- To provide quality education and training through novel pedagogical practices.
- To expedite high performance of excellence in teaching, research and innovations.
- To impart moral, ethical values and education with social responsibility.



METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES

After 3-5 years of graduation, the graduates will be able to

PEO1: Apply technical concepts, Analyze, Synthesize data to Design and create novel products and solutions for the real life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.



METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

Engineering graduates will be able to:

P01: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

P02: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

P03: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

P04: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

P05: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

P06: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

P07: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

P08: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

P09: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

P010: Communication: Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

P011: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

P012: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

At the end of 4 years, Computer Science and Engineering graduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various domains like networking and data mining to manage projects in multidisciplinary environments.

PSO2: Develop software applications with open-ended programming environments.

PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

Course Code	Course Title					Core / Elective	
PC 753 CS	Data Mining Lab					Core	
Prerequisite	Contact Hours per Week				CIE	SEE	Credits
	L	T	D	P			
-	-	-	-	2	25	50	1
Course Objectives <ul style="list-style-type: none"> ➤ To introduce the basic concepts of data Mining and its applications ➤ To understand different data mining like classification, clustering and Frequent Pattern mining ➤ To introduce current trends in data mining Course Outcomes <p>After completing this course, the student will be able to</p> <ol style="list-style-type: none"> 1. Organize and Prepare the data needed for data mining using preprocessing techniques 2. Implement the appropriate data mining methods like classification, clustering or Frequent Pattern mining on a given data set 3. Define and apply metrics to measure the performance of various data mining algorithms 							

List of Experiments to be performed

1. Implement the following Multidimensional Data Models
 - a. Star Schema
 - b. Snowflake Schema
 - c. Fact Constellation
2. Implement Apriori algorithm to generate frequent item sets.
3. Implement the following clustering algorithms
 - a. K-means
 - b. K-medians
4. Implement the following classification algorithms
 - a. Decision Tree Induction
 - b. KNN
5. Perform data preprocessing using WEKA
6. Perform discretization using WEKA
7. Classification of algorithms using WEKA
8. Apriori algorithm using WEKA
9. Perform data transformations using an ETL Tool
10. A small case study involving all stages of KDD (Datasets are available online like UCI Repository etc.)



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Outcomes (CO's):

SUBJECT NAME : DATA MINING LAB

CODE : PC 753 CS

SEMESTER : VII

CO No.	Course Outcome	Taxonomy Level
PC 753 CS.1	Apply data preprocessing techniques	Applying
PC 753 CS.2	Apply Frequent Item-set Mining methods to generate association rules	Applying
PC 753 CS.3	Identify and perform appropriate classification for given dataset.	Applying
PC 753 CS.4	Categorize and apply appropriate clustering for given dataset.	Analyzing
PC 753 CS.5	Evaluate models/algorithms with respect to their accuracy.	Evaluating
PC 753 CS.6	Construct a data mining solution to a practical problem	Creating



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the program / experiment details.
3. Student should enter into the laboratory with:
 - a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
 - b. Laboratory Record updated up to the last session experiments.
 - c. Formal dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out. If anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department

Principal



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CODE OF CONDUCT FOR THE LABORATORY

- All students must observe the dress code while in the laboratory
- Footwear is NOT allowed
- Foods, drinks and smoking are NOT allowed
- All bags must be left at the indicated place
- The lab timetable must be strictly followed
- Be PUNCTUAL for your laboratory session
- All programs must be completed within the given time
- Noise must be kept to a minimum
- Workspace must be kept clean and tidy at all time
- All students are liable for any damage to system due to their own negligence
- Students are strictly PROHIBITED from taking out any items from the laboratory
- Report immediately to the lab programmer if any damages to equipment

BEFORE LEAVING LAB:

- Arrange all the equipment and chairs properly.
- Turn off / shut down the systems before leaving.
- Please check the laboratory notice board regularly for updates.

Lab In – charge



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF EXPERIMENTS

SI. No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1.	Implement the following Multidimensional Data Models i. Star Schema ii. Snowflake Schema iii. Fact Constellation			1	
2.	Implement Apriori algorithm to generate frequent Item Sets			5	
3.	Implement the following clustering algorithms i. K-means ii. K-medoids			11	
4.	Implement the following classification algorithms i. Decision Tree Induction ii. KNN			17	
5.	Perform data Preprocessing using WEKA			25	
6.	Perform Discretization of data using WEKA			29	
7.	Classification algorithms using WEKA			46	
8.	Apriori algorithm using WEKA			58	
9.	Perform data transformations using an ETL Tool			60	
10.	A small case study involving all stages of KDD. (Datasets are available online like UCI Repository etc.)			66	

ADDITIONAL EXPERIMENTS

Sl. No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1	Data Manipulation in R			78	
2.	Classification in R			81	

PROGRAM 1.

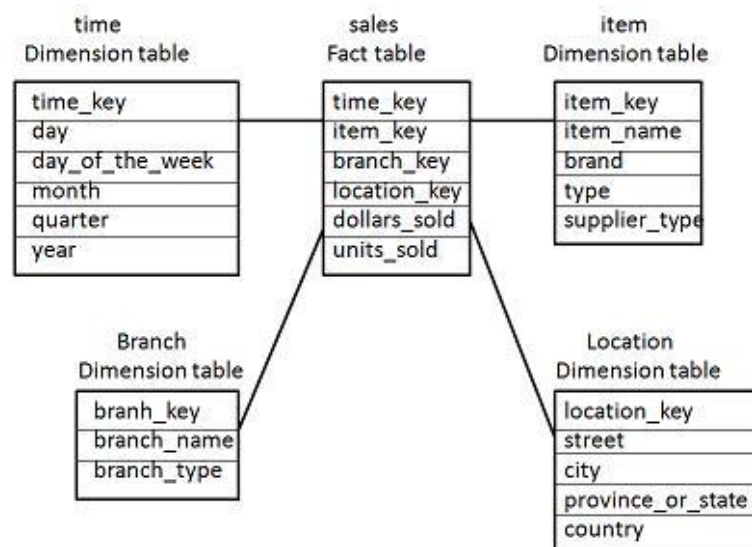
Aim : Implement the following Multidimensional Data Models

- i. Star Schema**
- ii. Snowflake Schema**
- iii. Fact Constellation**

Schema is a logical description of the entire database. It includes the name and description of records of all record types including all associated data-items and aggregates. Much like a database, a data warehouse also requires to maintain a schema. A database uses relational model, while a data warehouse uses Star, Snowflake, and Fact Constellation schema. In this chapter, we will discuss the schemas used in a data warehouse.

Star Schema

- Each dimension in a star schema is represented with only one-dimension table.
- This dimension table contains the set of attributes.
- The following diagram shows the sales data of a company with respect to the four dimensions, namely time, item, branch, and location.



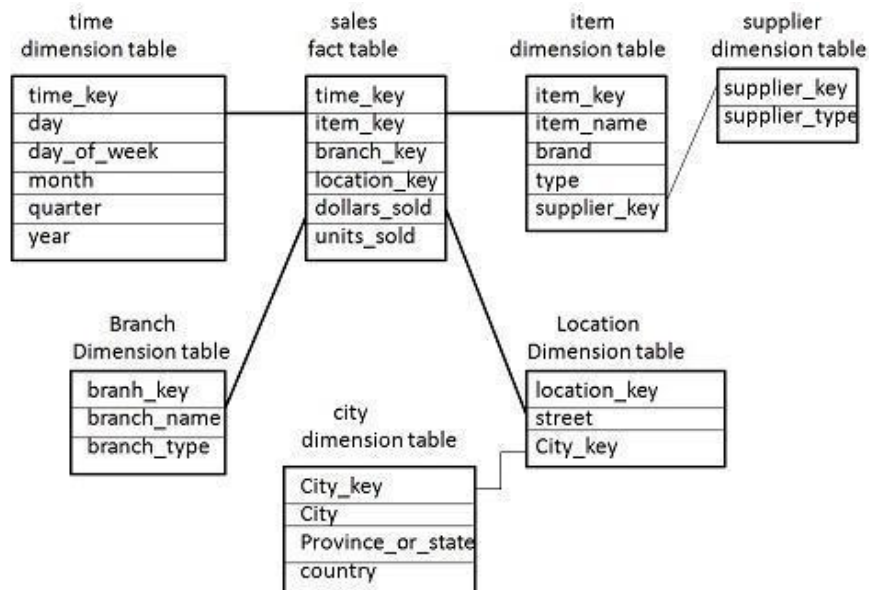
- There is a fact table at the center. It contains the keys to each of four dimensions.
- The fact table also contains the attributes, namely dollars sold and units sold.

Note: Each dimension has only one dimension table and each table holds a set of attributes. For example, the location dimension table contains the attribute set {location_key, street, city, province_or_state, country}. This constraint may cause data redundancy. For example, "Vancouver" and "Victoria" both the cities are in the Canadian province of British Columbia. The entries for such cities may cause data redundancy along the attributes province_or_state and country.

Snowflake Schema

- Some dimension tables in the Snowflake schema are normalized.
- The normalization splits up the data into additional tables.

- Unlike Star schema, the dimensions table in a snowflake schema are normalized. For example, the item dimension table in star schema is normalized and split into two dimension tables, namely item and supplier table.

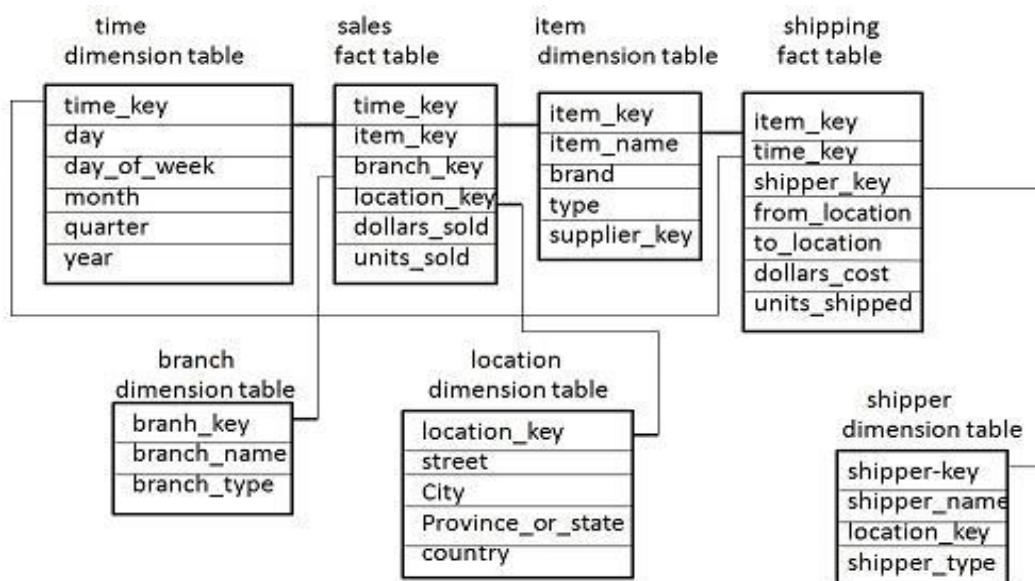


- Now the item dimension table contains the attributes item_key, item_name, type, brand, and supplier-key.
- The supplier key is linked to the supplier dimension table. The supplier dimension table contains the attributes supplier_key and supplier_type.

Note: Due to normalization in the Snowflake schema, the redundancy is reduced and therefore, it becomes easy to maintain and the save storage space.

Fact Constellation Schema

- A fact constellation has multiple fact tables. It is also known as galaxy schema.
- The following diagram shows two fact tables, namely sales and shipping.



- The sales fact table is same as that in the star schema.
- The shipping fact table has the five dimensions, namely item_key, time_key, shipper_key, from_location, to_location.
- The shipping fact table also contains two measures, namely dollars sold and units sold.
- It is also possible to share dimension tables between fact tables. For example, time, item, and location dimension tables are shared between the sales and shipping fact table.

Schema Definition

Multidimensional schema is defined using Data Mining Query Language (DMQL). The two primitives, cube definition and dimension definition, can be used for defining the data warehouses and data marts.

Syntax for Cube Definition

```
define cube < cube_name > [ < dimension-list > ]: < measure_list >
```

Syntax for Dimension Definition

```
define dimension < dimension_name > as ( < attribute_or_dimension_list > )
```

Star Schema Definition

The star schema can be defined using Data Mining Query Language (DMQL) as follows:

```
define cube sales star [time, item, branch, location]:
```

```
dollars sold = sum(sales in dollars), units sold = count(*)
```

```
define dimension time as (time key, day, day of week, month, quarter, year)
```

```
define dimension item as (item key, item name, brand, type, supplier type)
```

```
define dimension branch as (branch key, branch name, branch type)
```

```
define dimension location as (location key, street, city, province or state, country)
```

Snowflake Schema Definition

Snowflake schema can be defined using DMQL as follows:

```
define cube sales snowflake [time, item, branch, location]:
```

```
dollars sold = sum(sales in dollars), units sold = count(*)
```

```
define dimension time as (time key, day, day of week, month, quarter, year)
```

```
define dimension item as (item key, item name, brand, type, supplier (supplier key, supplier type))
```

```
define dimension branch as (branch key, branch name, branch type)
```

```
define dimension location as (location key, street, city (city key, city, province or state, country))
```

Fact Constellation Schema Definition

Fact constellation schema can be defined using DMQL as follows:

define cube sales [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)

define dimension item as (item key, item name, brand, type, supplier type)

define dimension branch as (branch key, branch name, branch type)

define dimension location as (location key, street, city, province or state, country)

define cube shipping [time, item, shipper, from location, to location]:

dollars cost = sum(cost in dollars), units shipped = count(*)

define dimension time as time in cube sales

define dimension item as item in cube sales

define dimension shipper as (shipper key, shipper name, location as location in cube sales, shipper type)

define dimension from location as location in cube sales

define dimension to location as location in cube sales

Viva Voce

- 1) Define Schema
- 2) Define dimension table and fact table
- 3) What is a star schema?
- 4) Define fact-less fact.
- 5) What is a data cube?
- 6) What is a snow flake schema?
- 7) What is the language that is used for schema definition?
- 8) What is a data warehouse?
- 9) What is a data mart?
- 10) What is a Fact Constellation schema?

PROGRAM 2.**Aim : Implement Apriori algorithm to generate frequent Item Sets****Introduction:**

Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

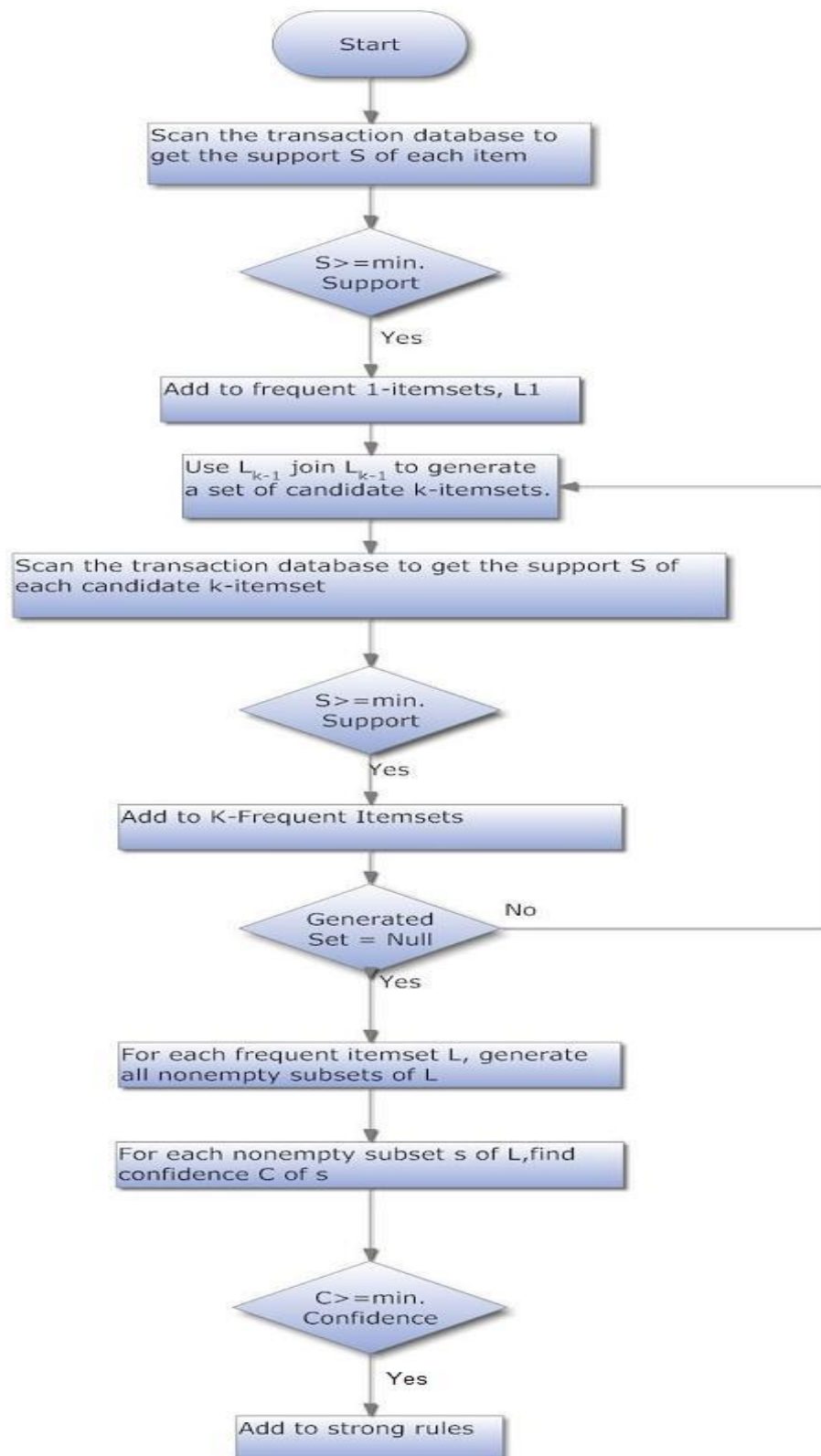
Working:

Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing). Each transaction is seen as a set of items (an itemset). Given a threshold C , the Apriori algorithm identifies the item sets which are subsets of at least C transactions in the database.

Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k - 1$. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k -length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

Low Level Design



Program:

```

import java.io.*;
class apriori
{
public static void main(String []arg)throws IOException
{
int i,j,m=0;
int t1=0;
BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the number of transaction :");
int n=Integer.parseInt(b.readLine());
System.out.println("items :1--Milk 2--Bread 3--Coffee 4--Juice 5--Cookies 6--Jam");
int item[][]=new int[n][6];
for(i=0;i<n;i++)
for(j=0;j<6;j++)
item[i][j]=0;
String[] itemlist={"MILK","BREAD","COFFEE","JUICE","COOKIES","JAM"};
int nt[]=new int[6];
int q[]=new int[6];
for(i=0;i<n;i++)
{ System.out.println("Transaction "+(i+1)+" :");
for(j=0;j<6;j++)
{ //System.out.println(itemlist[j]);
System.out.println("Is Item "+itemlist[j]+" present in this transaction(1/0)? :");
item[i][j]=Integer.parseInt(b.readLine());
}
}
for(j=0;j<6;j++)
{ for(i=0;i<n;i++)
{ if(item[i][j]==1)
nt[j]=nt[j]+1;
}
System.out.println("Number of Item "+itemlist[j]+" :"+nt[j]);
}

for(j=0;j<6;j++)
{ if(((nt[j]/(float)n)*100)>=50)
q[j]=1;
else
q[j]=0;

if(q[j]==1)
{t1++;
System.out.println("Item "+itemlist[j]+" is selected ");

}
}
for(j=0;j<6;j++)
{ for(i=0;i<n;i++)
{

```

```

        if(q[j]==0)
        {
            item[i][j]=0;
        }
    }
}

int nt1[][]=new int[6][6];
for(j=0;j<6;j++)
    { for(m=j+1;m<6;m++)
      { for(i=0;i<n;i++)
        { if(item[i][j]==1 &&item[i][m]==1)
          { nt1[j][m]=nt1[j][m]+1;
            }
          }
        }
      if(nt1[j][m]!=0)
        System.out.println("Number of Items of "+itemlist[j]+"& "+itemlist[m]+" :"+nt1[j][m]);
    }
}

for(j=0;j<6;j++)
    { for(m=j+1;m<6;m++)
      {
          if(((nt1[j][m]/(float)n)*100)>=50)
              q[j]=1;
          else
              q[j]=0;

          if(q[j]==1)
          {
              System.out.println("Item "+itemlist[j]+"& "+itemlist[m]+" is selected ");
          }
      }
    }
}
}
}
}

```

Expected Output:

C:\>javac apriori.java

C:\>java apriori

Enter the number of transaction : 4

items :1--Milk 2--Bread 3--Coffee 4--Juice 5--Cookies 6--Jam

Transaction 1 :

Is Item MILK present in this transaction(1/0)? :1

Is Item BREAD present in this transaction(1/0)? :1

Is Item COFFEE present in this transaction(1/0)? :0

Is Item JUICE present in this transaction(1/0)? :1

Is Item COOKIES present in this transaction(1/0)? :1

Is Item JAM present in this transaction(1/0)? :0

Transaction 2 :

Is Item MILK present in this transaction(1/0)? :1
Is Item BREAD present in this transaction(1/0)? :0
Is Item COFFEE present in this transaction(1/0)? :0
Is Item JUICE present in this transaction(1/0)? :1
Is Item COOKIES present in this transaction(1/0)? :0
Is Item JAM present in this transaction(1/0)? :0

Transaction 3 :

Is Item MILK present in this transaction(1/0)? :1
Is Item BREAD present in this transaction(1/0)? :0
Is Item COFFEE present in this transaction(1/0)? :0
Is Item JUICE present in this transaction(1/0)? :0
Is Item COOKIES present in this transaction(1/0)? :0
Is Item JAM present in this transaction(1/0)? :1

Transaction 4 :

Is Item MILK present in this transaction(1/0)? :0
Is Item BREAD present in this transaction(1/0)? :1
Is Item COFFEE present in this transaction(1/0)? :1
Is Item JUICE present in this transaction(1/0)? :0
Is Item COOKIES present in this transaction(1/0)? :1
Is Item JAM present in this transaction(1/0)? :0

Number of Item MILK :3

Number of Item BREAD :2

Number of Item COFFEE :1

Number of Item JUICE :2

Number of Item COOKIES :2

Number of Item JAM :1

Item MILK is selected

Item BREAD is selected

Item JUICE is selected

Item COOKIES is selected

Number of Items of MILK& BREAD :1

Number of Items of MILK& COFFEE :0

Number of Items of MILK& JUICE :2

Number of Items of MILK& COOKIES :1

Number of Items of MILK& JAM :0

Number of Items of BREAD& COFFEE :0

Number of Items of BREAD& JUICE :1

Number of Items of BREAD& COOKIES :2

Number of Items of BREAD& JAM :0

Number of Items of COFFEE& JUICE :0

Number of Items of COFFEE& COOKIES :0

Number of Items of COFFEE& JAM :0

Number of Items of JUICE& COOKIES :1

Number of Items of JUICE& JAM :0

Number of Items of COOKIES& JAM :0

Item MILK& JUICE is selected

Item BREAD& COOKIES is selected

Viva Voce

- 1) Define Apriori property
- 2) What are the two steps of Apriori algorithm?
- 3) Define support and confidence in Association rule mining.
- 4) What is Association rule?
- 5) What are the Applications of Association rule mining?
- 6) How are association rules mined from large databases?
- 7) What is pruning
- 8) How is apriori property used in algorithm
- 9) Define frequent item set
- 10) What are the limitations of Apriori Algorithm

PROGRAM 3:

Aim : Implement the following clustering algorithms

- i. K-means**
- ii. K-medoids**

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The working of the **K-Means algorithm** is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Program:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
import java.util.StringTokenizer;

public class Kmeans {

    public static class Instance {
        ArrayList<Double> mAttributes = new ArrayList<>();
        String mClass;

        @Override
        public String toString() {
            return mAttributes.toString() + "\t" + mClass;
        }
    }

    public static class Cluster {
        String name;
        ArrayList<Double> oldCentroid = new ArrayList<>();
        ArrayList<Double> newCentroid = new ArrayList<>();

        public boolean isStillMoving() {
            double difference = 0.0;
            for (int i = 0; i < oldCentroid.size(); i++) {
                difference += Math.abs(oldCentroid.get(i) - newCentroid.get(i));
            }
            return (difference > 0.1);
        }
    }

    private static String mInputFilename;
    private static String mOutputFilename;

    private static ArrayList<Instance> instances = new ArrayList<>();
    private static ArrayList<Cluster> clusters = new ArrayList<>();
```

```
private static int clusterCount;
private static int iteration = 0;

public static void main(String[] args) throws IOException {
    if (args.length < 3) {
        System.out.println("Arguments are incorrect. Try Application #inputfilename
#outputfilename #clusterCount");
        System.exit(0);
    }
    else {
        mInputFilename = args[0];
        mOutputFilename = args[1];
        clusterCount = Integer.parseInt(args[2]);
        readData();
        cluster();

        ArrayList<String> output = new ArrayList<>();
        output.add("ITERATIONS : " + iteration);
        for (Instance i : instances) {
            output.add(i.toString());
        }
        PutFileData(output);
    }
}

// random number generator in a range, upper bound not included
public static int randInt(int min, int max) {
    Random random = new Random();
    return random.nextInt((max - min) + min);
}

public static void cluster() {
    do {
        System.out.println(++iteration);

        if (iteration == 1) {
            // initialize clusters
            for (int i = 0; i < clusterCount; i++) {
                Cluster c = new Cluster();
                c.name = i + "";
                c.oldCentroid.addAll(instances.get(i).mAttributes);
                clusters.add(c);
            }
        } else {
            // copy new mean to old
            for (int i = 0; i < clusterCount; i++) {
                clusters.get(i).oldCentroid.clear();
                clusters.get(i).oldCentroid.addAll(clusters.get(i).newCentroid);
            }
        }
        // assign instances
        for (Instance i : instances) {
```

```

        ArrayList<Double> distances = new ArrayList<>();
        for (Cluster c : clusters) {
            distances.add(getDistance(i.mAttributes, c.oldCentroid));
        }
        i.mClass = clusters.get(distances.indexOf(Collections.min(distances))).name;
    }

    // calc new mean
    for (int j = 0; j < clusterCount; j++) {

        ArrayList<Double> centroid = new ArrayList<>();
        for (int i = 0; i < clusters.get(j).oldCentroid.size(); i++) {
            double sum = 0.0;
            int count = 0;
            for (Instance inst : instances) {
                if (inst.mClass.equals(clusters.get(j).name)) {
                    sum += inst.mAttributes.get(i);
                    count++;
                }
            }
            centroid.add(sum / count);
        }
        clusters.get(j).newCentroid.clear();
        clusters.get(j).newCentroid.addAll(centroid);
    }
} while (isStillMoving() && iteration < 10);
}

public static boolean isStillMoving(){

    for(int i=0;i<clusterCount;i++){
        if(clusters.get(i).isStillMoving()){
            return true;
        }
    }
    return false;
}

public static double getDistance(ArrayList<Double> a, ArrayList<Double> b) {

    double distance = 0;
    for (int i = 0; i < a.size(); i++) {
        double x = a.get(i);
        double y = b.get(i);
        distance += (x - y) * (x - y);
    }
    return Math.sqrt(distance);
}

// get data
public static void readData() throws IOException {
    StringTokenizer st;
    int size;

```

```
ArrayList<String> line;
line = GetFileData();

for (String str : line) {
    Instance value = new Instance();
    st = new StringTokenizer(str, ",");
    size = st.countTokens();
    for (int i = 0; i < size; i++) {
        value.mAttributes.add(Double.parseDouble(st.nextToken()));
    }
    instances.add(value);
}
}

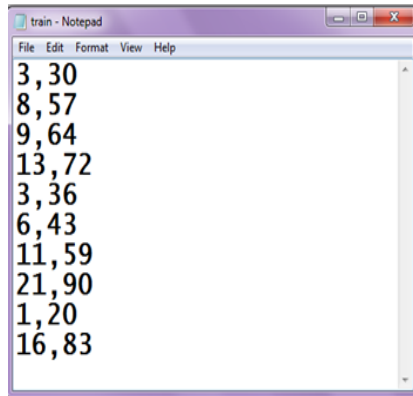
// File read write functions
public static ArrayList<String> GetFileData() throws IOException {
    String line;
    ArrayList<String> fileData = new ArrayList<String>();
    BufferedReader bufferedReader = new BufferedReader(new FileReader(
        mInputFilename));
    while ((line = bufferedReader.readLine()) != null) {
        fileData.add(line);
    }
    bufferedReader.close();
    return fileData;
}

public static void PutFileData(ArrayList<String> output) throws IOException {

    File f = new File(mOutputFilename);
    if (!f.exists()) {
        f.createNewFile();
    }
    BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(
        mOutputFilename, true));
    bufferedWriter.newLine();
    for (String line : output) {
        bufferedWriter.write(line);
        bufferedWriter.newLine();
    }
    bufferedWriter.close();
}
}
```

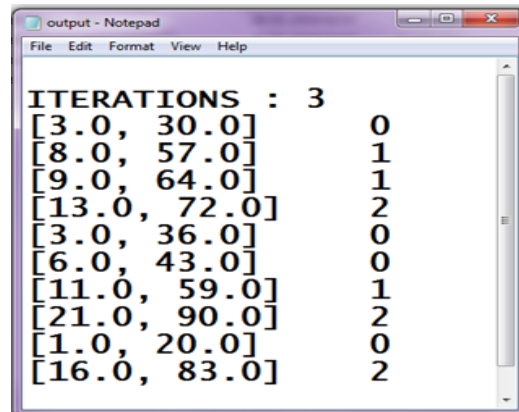
Expected Output:

Training data



```
3,30
8,57
9,64
13,72
3,36
6,43
11,59
21,90
1,20
16,83
```

output



```
ITERATIONS : 3
[3.0, 30.0]      0
[8.0, 57.0]      1
[9.0, 64.0]      1
[13.0, 72.0]     2
[3.0, 36.0]      0
[6.0, 43.0]      0
[11.0, 59.0]     1
[21.0, 90.0]    2
[1.0, 20.0]      0
[16.0, 83.0]    2
```

Viva Voce

- 1) What is Clustering?
- 2) What are different types of clustering?
- 3) How does the K mean algorithm work
- 4) Define unsupervised Learning
- 5) How to determine the number of clusters in k-means clustering algorithm?
- 6) What are the fields in which clustering techniques are used?
- 7) What are the different types of data used for cluster analysis?
- 8) What are the requirements of cluster analysis?
- 9) What do you mean by partitioning method?
- 10) What are the requirements of cluster analysis?

PROGRAM 4.**Aim : Implement the following classification algorithms****i. Decision Tree Induction****ii. KNN****i. Decision Tree Induction**

Decision tree learning, used in data mining and machine learning, uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. More descriptive names for such tree models are classification trees or regression trees. In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making.

- Basic steps
- Building tree
- Applying the tree to database

Internal node-test on attribute

Branch-outcome of test

Leaf node-class

Topmost-root node

Algorithm: Generate decision tree. Generate a decision tree from the training tuples of data partition, D .

Input:

- Data partition, D , which is a set of training tuples and their associated class labels;
- *attribute list*, the set of candidate attributes;
- *Attribute selection method*, a procedure to determine the splitting criterion that “best” partitions the data tuples into individual classes. This criterion consists of a *splitting attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.**Method / Algorithm:**

- (1) create a node N ;
- (2) **if** tuples in D are all of the same class, C , **then**
- (3) return N as a leaf node labeled with the class C ;
- (4) **if** *attribute list* is empty **then**

```

(5) return  $N$  as a leaf node labeled with the majority class in  $D$ ; // majority voting
(6) apply Attribute selection method( $D$ , attribute list) to find the “best” splitting criterion;
(7) label node  $N$  with splitting criterion;
(8) if splitting attribute is discrete-valued and multiway splits allowed then // not restricted to
binary trees
(9) attribute list = attribute list  $\square$  splitting attribute; // remove splitting attribute
(10) for each outcome  $j$  of splitting criterion
// partition the tuples and grow subtrees for each partition
(11) let  $D_j$  be the set of data tuples in  $D$  satisfying outcome  $j$ ; // a partition
(12) if  $D_j$  is empty then
(13) attach a leaf labeled with the majority class in  $D$  to node  $N$ ;
(14) else attach the node returned by Generate decision tree( $D_j$ , attribute list) to node  $N$ ;
Endfor
(15) return  $N$ ;

```

ii. KNN

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well –

- **Lazy learning algorithm** – KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** – KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Working of KNN Algorithm

K-nearest neighbors (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 – For each point in the test data do the following –

- **3.1** – Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- **3.2** – Now, based on the distance value, sort them in ascending order.
- **3.3** – Next, it will choose the top K rows from the sorted array.
- **3.4** – Now, it will assign a class to the test point based on most frequent class of **these rows**.

Step 4 – End

Program:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.StringTokenizer;

public class KNNClassifier {
    private static class Instance {
        ArrayList<Double> mAttributes = new ArrayList<>();
        String mClass;
        @Override
        public String toString() {
            return mAttributes.toString() + "\t" + mClass;
        }
    }
    private static class PredictInstance extends Instance {
        ArrayList<Double> mDistances = new ArrayList<>();
        ArrayList<Integer> mNeighbours = new ArrayList<>();
        ArrayList<Double> mVote = new ArrayList<>();
    }
    private static String mInputFilename;
```



```
private static String mInputTestFilename;
private static String mOutputFilename;
private static int K;
private static String[] classNames = { "iris-setosa", "iris-versicolor", "iris-virginica" };
private static List<Instance> mTrainData;
private static List<PredictInstance> mPredictData;

public static void main(String[] args) throws IOException {
    if (args.length < 4) {
        System.out.println("Arguments are incorrect. Try Application #inputfilename
            #testfilename #outputfilename #k-value");
        System.exit(0);
    } else {
        mInputFilename = args[0];
        mInputTestFilename = args[1];
        mOutputFilename = args[2];
        K = Integer.parseInt(args[3]);
        readData();
        classify();
        ArrayList<String> output = new ArrayList<>();
        for (PredictInstance t : mPredictData) {
            output.add(t.toString());
        }
        PutFileData(output);
    }
}

public static void classify() {
    // compute distances for all test instances
    for (PredictInstance t : mPredictData) {
        for (Instance i : mTrainData) {
            t.mDistances.add(getDistance(t.mAttributes, i.mAttributes));
        }
    }
    // sorting the distances and getting k-nearest class frequency
    for (PredictInstance t : mPredictData) {
```

```

        t.mNeighbours.addAll(getNearestNeighbours(t.mDistances));
    for (Integer i : t.mNeighbours) {
        if(mTrainData.get(i).mClass.equalsIgnoreCase(classNames[0])) {
            t.mVote.set(0, t.mVote.get(0) + (1 / t.mDistances.get(i)));
        } else if (mTrainData.get(i).mClass.equalsIgnoreCase(classNames[1])) {
            t.mVote.set(1, t.mVote.get(1) + (1 / t.mDistances.get(i)));
        } else {
            t.mVote.set(2, t.mVote.get(2) + (1 / t.mDistances.get(i)));
        }
    }
}

// comparing weights and selecting a class
for (PredictInstance t : mPredictData) {
    t.mClass = classNames[t.mVote.indexOf(Collections.max(t.mVote))];
}
}

public static ArrayList<Integer> getNearestNeighbours(ArrayList<Double> distances) {
    ArrayList<Integer> indexes = new ArrayList<>(K);
    ArrayList<Double> values = new ArrayList<>();
    values.addAll(distances);
    Collections.sort(values);
    for (int i = 0; i < K; i++) {
        indexes.add(distances.indexOf(values.get(i)));
    }
    return indexes;
}

public static double getDistance(ArrayList<Double> a, ArrayList<Double> b) {
    double distance = 0;
    for (int i = 0; i < a.size(); i++) {
        double x = a.get(i);
        double y = b.get(i);
        distance += (x - y) * (x - y);
    }
    return Math.sqrt(distance);
}
}

```

```
// get data
public static void readData() throws IOException {
    StringTokenizer st;
    mTrainData = new ArrayList<>();
    mPredictData = new ArrayList<>();
    int size;
    ArrayList<String> line;
    line = GetFileData();
    for (String str : line) {
        Instance value = new Instance();
        st = new StringTokenizer(str, ",");
        size = st.countTokens();
        for (int i = 0; i < size - 1; i++) {
            value.mAttributes.add(Double.parseDouble(st.nextToken()));
        }
        value.mClass = st.nextToken();
        mTrainData.add(value);
    }
    line = GetTestFileData();
    for (String str : line) {
        PredictInstance value = new PredictInstance();
        st = new StringTokenizer(str, ",");
        size = st.countTokens();
        for (int i = 0; i < size; i++) {
            value.mAttributes.add(Double.parseDouble(st.nextToken()));
        }
        value.mClass = "unknown";
        for (int j = 0; j < classNames.length; j++) {
            value.mVote.add(0.0);
        }
        mPredictData.add(value);
    }
}

// File read write functions
public static ArrayList<String> GetFileData() throws IOException {
```

```
String line;
ArrayList<String> fileData = new ArrayList<String>();
BufferedReader bufferedReader = new BufferedReader(new FileReader(
    mInputFilename));
while ((line = bufferedReader.readLine()) != null) {
    fileData.add(line);
}
bufferedReader.close();
return fileData;
}

public static ArrayList<String> GetTestFileData() throws IOException {
    String line;
    ArrayList<String> fileData = new ArrayList<String>();
    BufferedReader bufferedReader = new BufferedReader(new FileReader(
        mInputTestFilename));
    while ((line = bufferedReader.readLine()) != null) {
        fileData.add(line);
    }
    bufferedReader.close();
    return fileData;
}

public static void PutFileData(ArrayList<String> output) throws IOException {
    File f = new File(mOutputFilename);
    if (!f.exists()) {
        f.createNewFile();
    }
    BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(
        mOutputFilename, true));
    bufferedWriter.newLine();
    for (String line : output) {
        bufferedWriter.write(line);
        bufferedWriter.newLine();
    }
    bufferedWriter.close();
}
```

}

Expected Output:**Training Data**

```

train.txt - Notepad
File Edit Format View Help
5.1,3.5,1.4,0.2,Iris-setosa
4.8,3.1,1.4,0.1,Iris-setosa
5.8,4.1,2.0,0.2,Iris-setosa
4.7,3.2,1.6,0.2,Iris-setosa
5.5,3.5,1.3,0.2,Iris-setosa
6.4,3.2,4.5,1.5,Iris-versicolor
6.3,2.5,4.9,1.5,Iris-versicolor
6.8,2.8,4.8,1.4,Iris-versicolor
5.8,2.6,4.1,1.2,Iris-versicolor
5.7,3.4,2.1,2,Iris-versicolor
6.8,3.5,5.2,1,Iris-virginica
6.2,2.5,1.5,Iris-virginica
5.6,2.8,4.9,2,Iris-virginica
7.7,3.6,1.2,3,Iris-virginica
6.7,3,5.2,2.3,Iris-virginica

```

Testing Data

```

test.txt - Notepad
File Edit Format View Help
5.3,2.1,2,0.2
4.9,3.1,1.5,0.1
6.7,3.1,4.7,1.5
5.6,2.7,4.2,1.3
6.7,3.3,5.7,2.1
6.4,3.1,5.5,1.8

```

Output.txt

```

output.txt - Notepad
File Edit Format View Help
[5.0, 3.2, 1.2, 0.2] iris-setosa
[4.9, 3.1, 1.5, 0.1] iris-setosa
[6.7, 3.1, 4.7, 1.5] iris-versicolor
[5.6, 2.7, 4.2, 1.3] iris-versicolor
[6.7, 3.3, 5.7, 2.1] iris-virginica
[6.4, 3.1, 5.5, 1.8] iris-virginica

```

Viva Voce

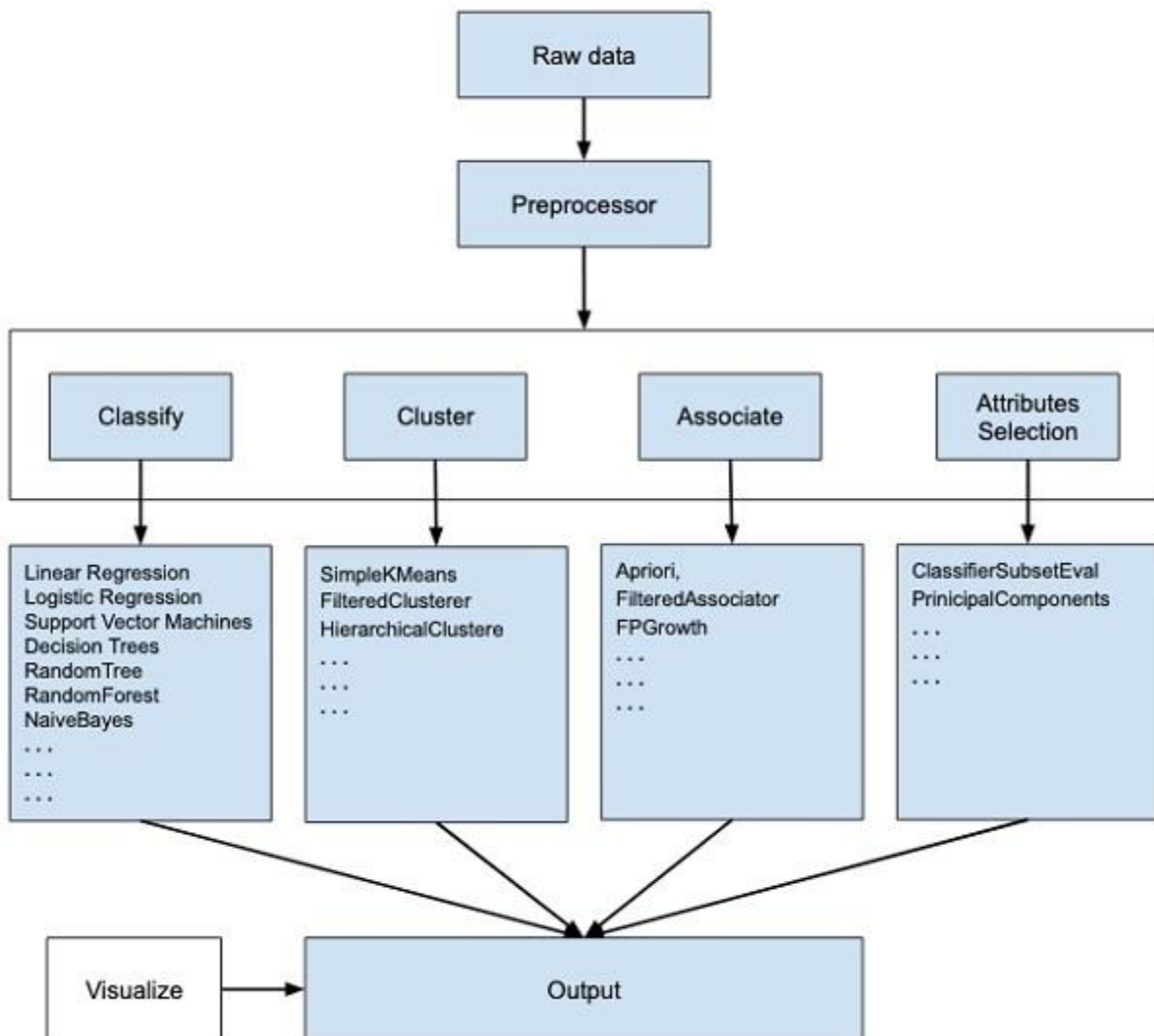
- 1) Define classifier
- 2) How does classification work
- 3) When do we classify the data
- 4) what are the applications of KNN Classifier
- 5) How to select the value of K in the K-NN Algorithm?
- 6) what are the advantages of KNN Algorithm ?
- 7) what are the design issues of Decision Tree Induction
- 8) How to build a decision Tree
- 9) How should the training records be split
- 10) How should the splitting procedure stops
- 11) Define classification

PROGRAM 5 & 6.

Aim : Perform data preprocessing and discretization using WEKA.

Description (for Programs 5, 6, 7 and 8):

WEKA - an open source software provides tools for data preprocessing, implementation of several Data Mining and Machine Learning algorithms, and visualization tools so that you can develop Data Mining and Machine Learning techniques and apply them to real-world data mining problems. What WEKA offers is summarized in the following diagram –



If you observe the beginning of the flow of the image, you will understand that there are many stages in dealing with Data to make it suitable for Data Mining and Machine Learning

First, you will start with the raw data collected from the field. This data may contain several null values and irrelevant fields. You use the data preprocessing tools provided in WEKA to cleanse the data.

Then, you would save the preprocessed data in your local storage for applying Data Mining and Machine Learning algorithms.

Next, depending on the kind of Data Mining that you are trying to develop you would select one of the options such as **Classify**, **Cluster**, or **Associate**. The **Attributes Selection** allows the automatic selection of features to create a reduced dataset.

Note that under each category, WEKA provides the implementation of several algorithms. You would select an algorithm of your choice, set the desired parameters and run it on the dataset.

Then, WEKA would give you the statistical output of the model processing. It provides you a visualization tool to inspect the data.

The various models can be applied on the same dataset. You can then compare the outputs of different models and select the best that meets your purpose.

Thus, the use of WEKA results in a quicker development of Data Mining and Machine Learning models on the whole.

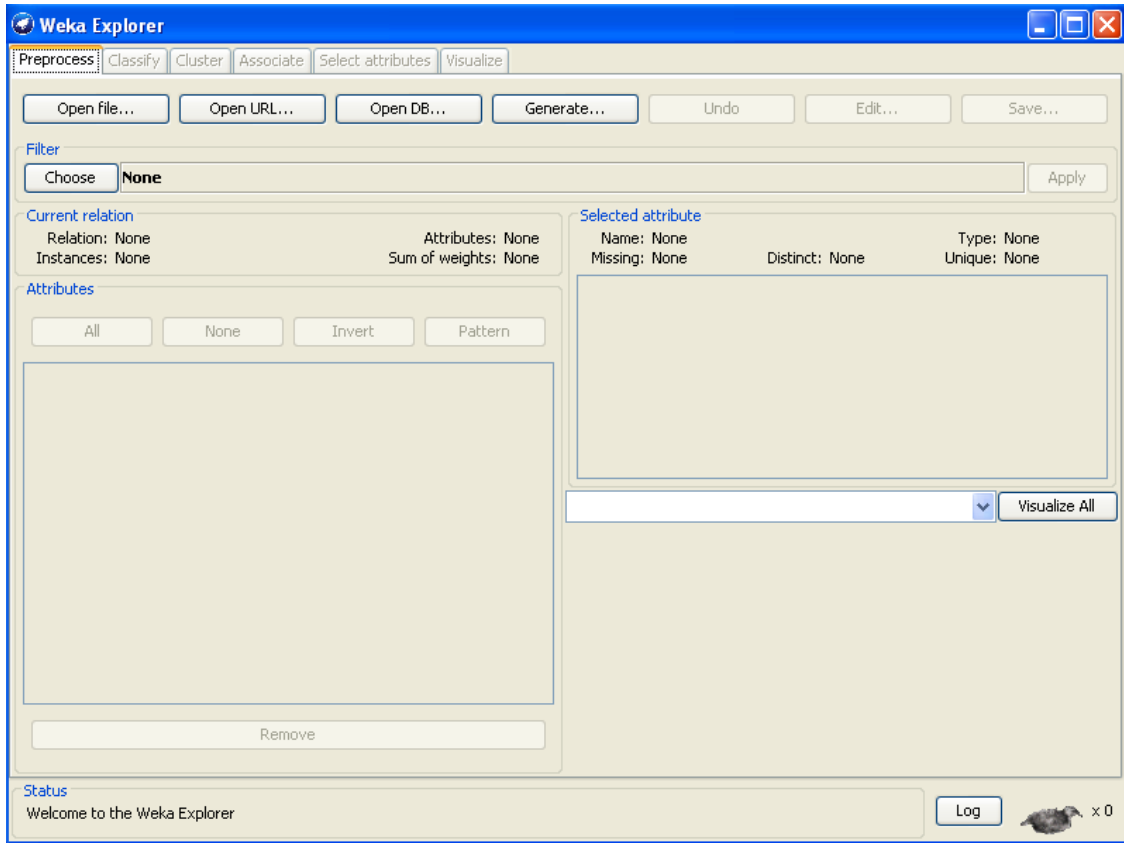
Program 5 & 6:

1. ADD

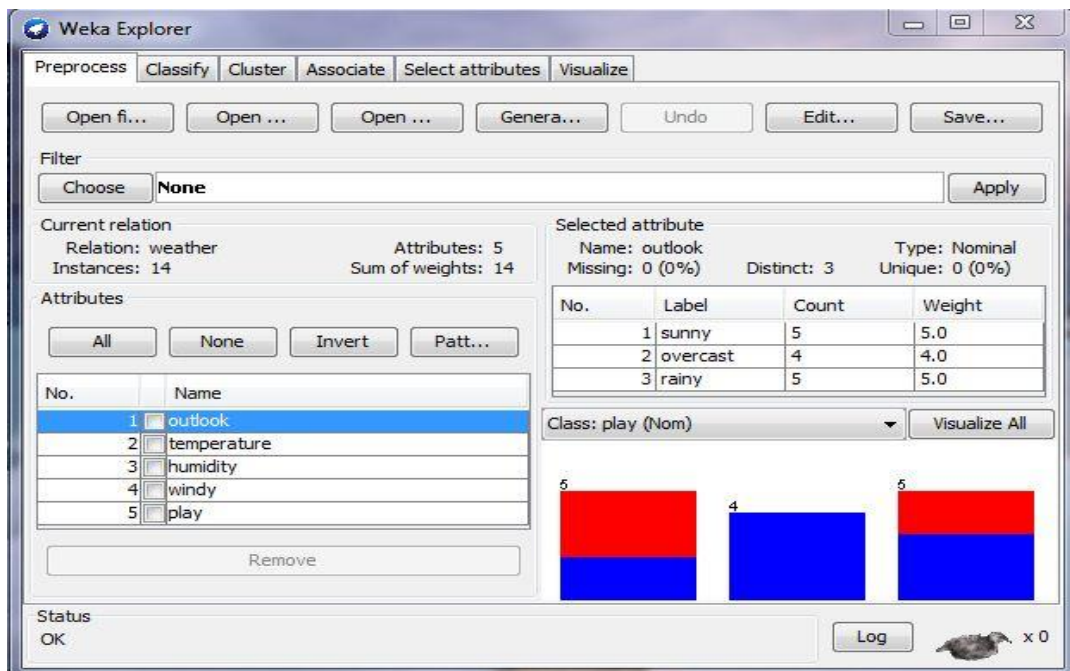
1. Start Weka – you get the Weka GUI chooser window.



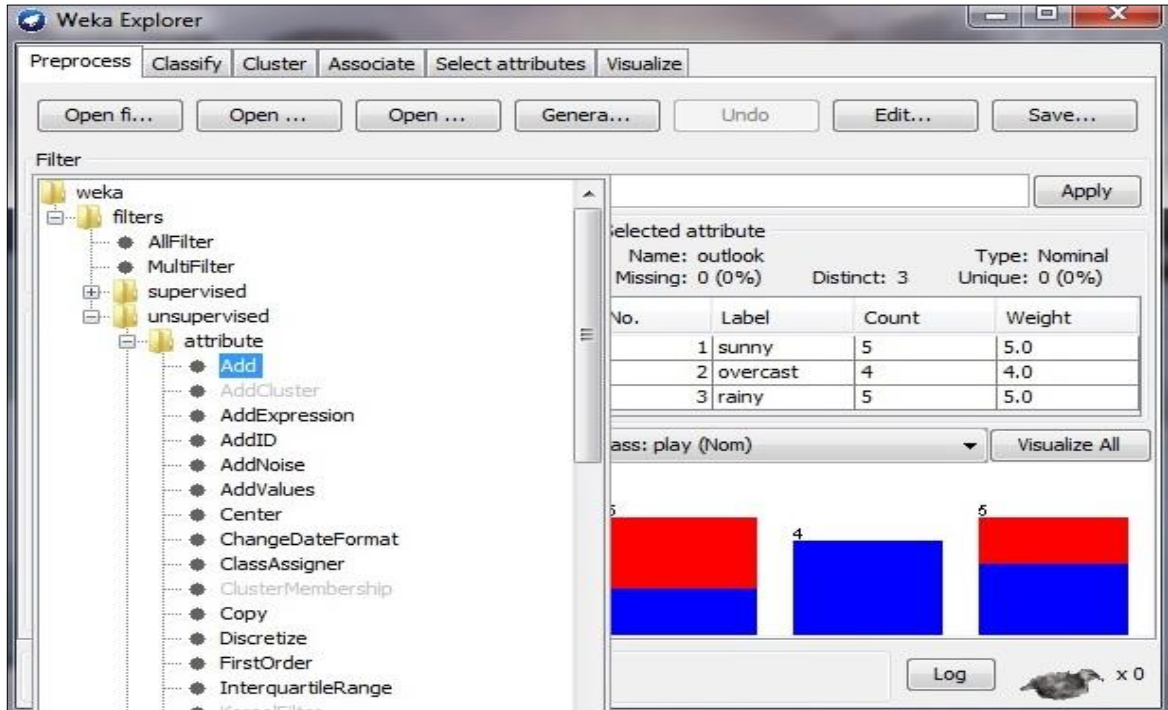
- Click on the Explorer button and you get the Weka Knowledge Explorer window.



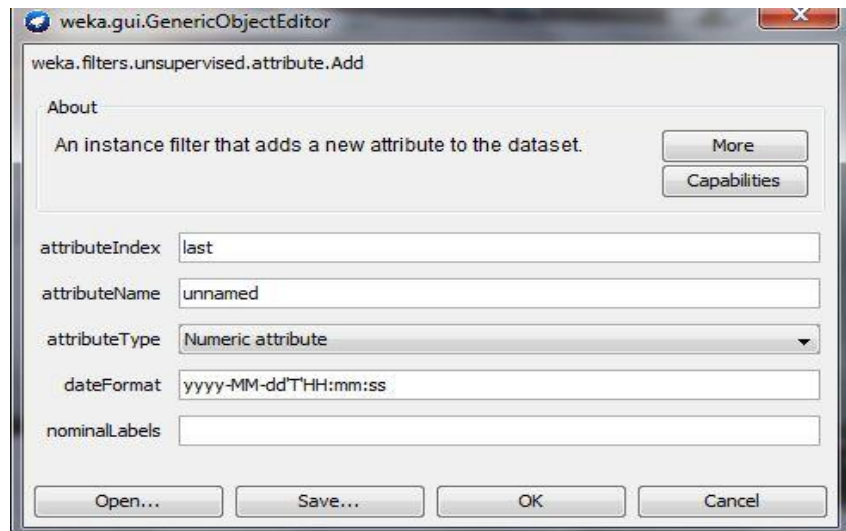
- Click on the “Open File.” button and open an ARFF file (try it first with an example supplied in Weka-3-6/data, e.g. Weather.arff). You get the following:



4. **VALIDATION :** Click on Choose and select filters/unsupervised/attribute/Add.

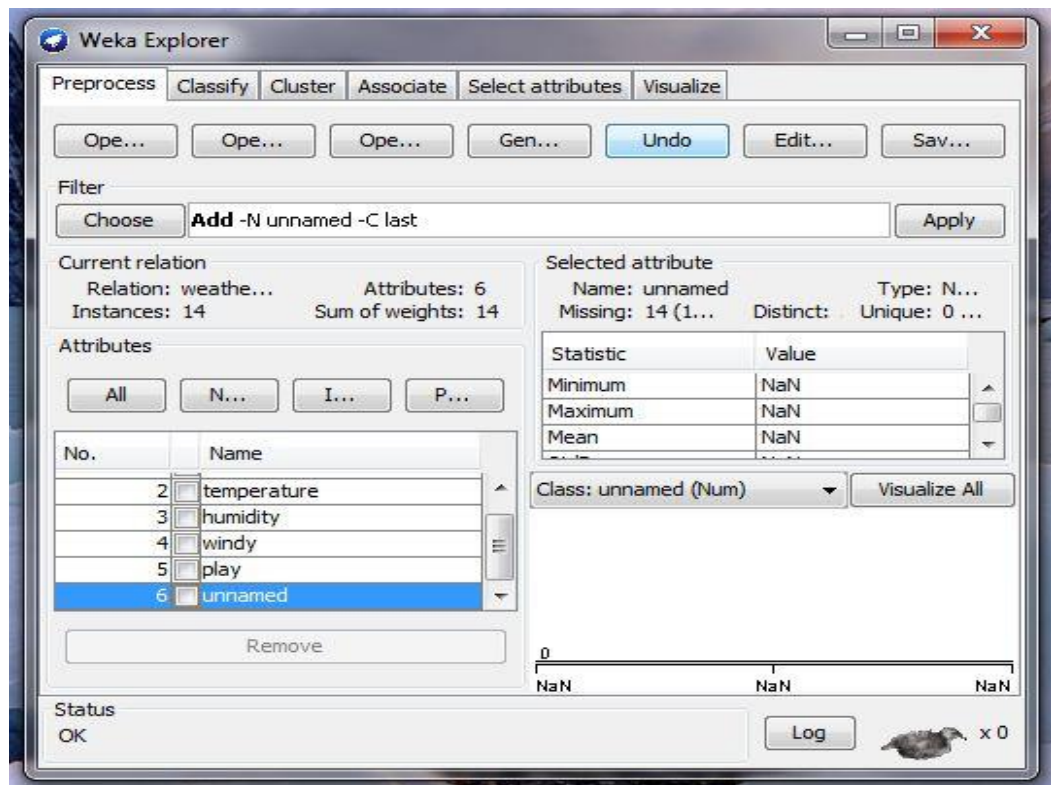


Then click on the area right of the Choose button. You get the following:



You see here the default parameters of this filter. Click on More to get more information about these parameters.

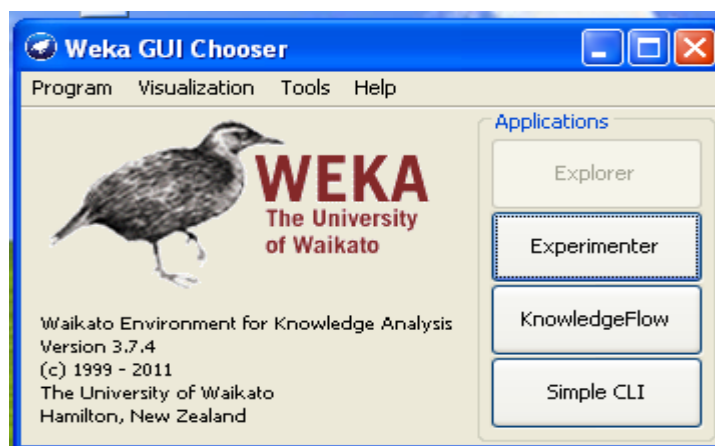
5. Click on the Apply button to do the Addition and see how it is Added in the Selected attribute window.



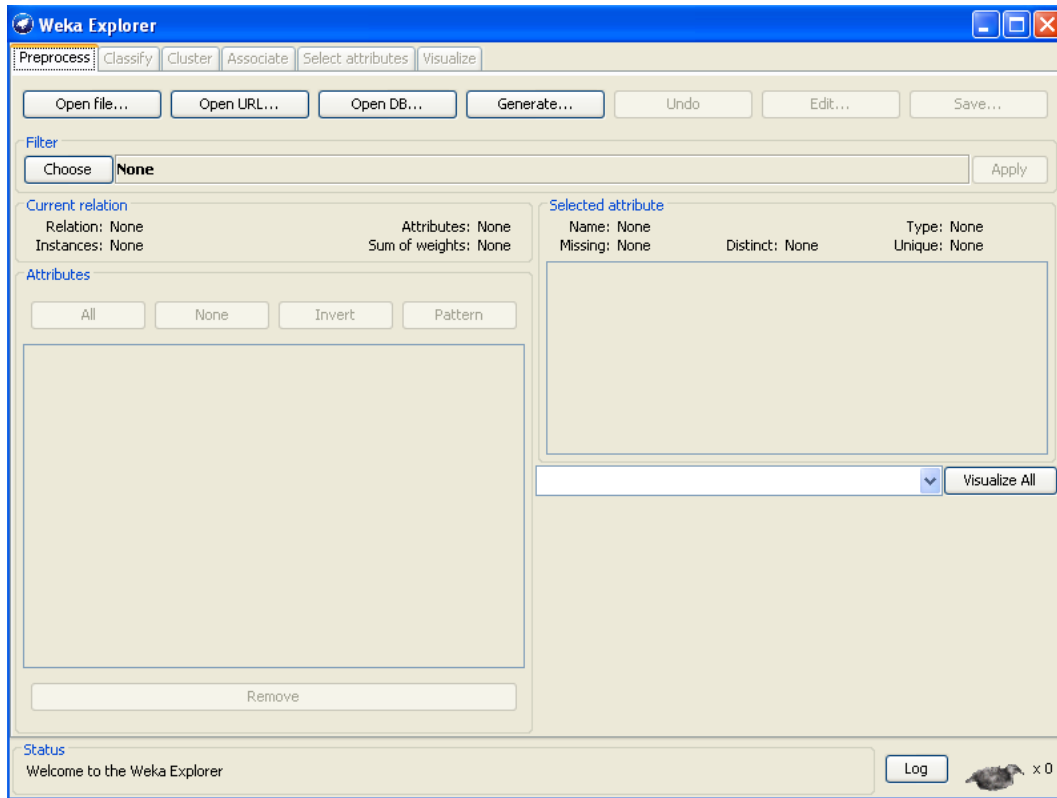
6. Try other parameters for the filter and see how the Addition changes. Don't forget to reload the original (numeric) relation or Undo the Addition before applying another one.

2. Discretization

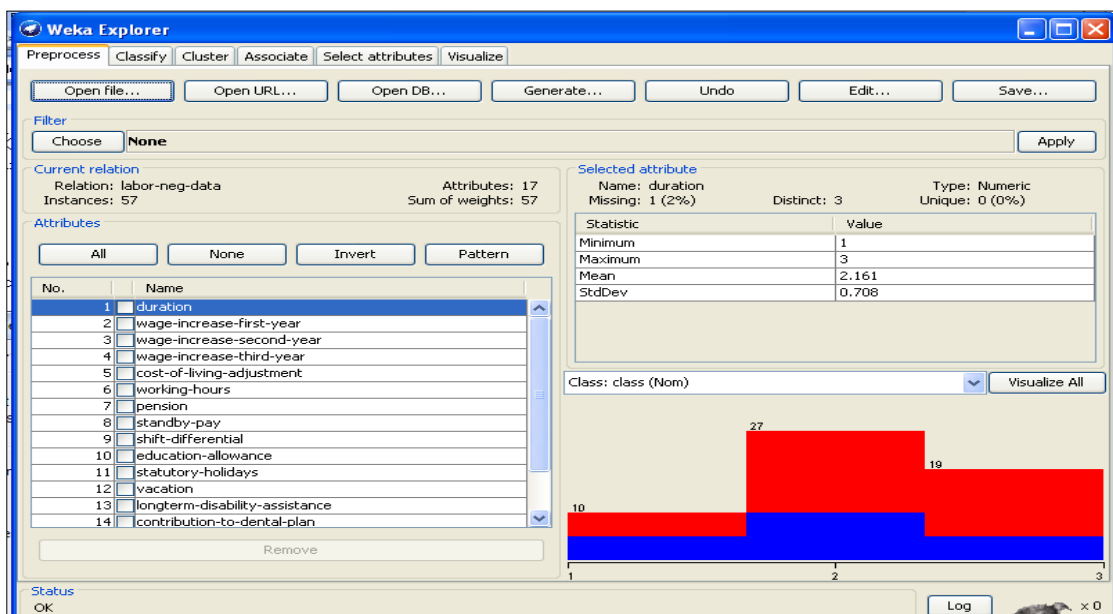
1. Start Weka – you get the Weka GUI chooser window.



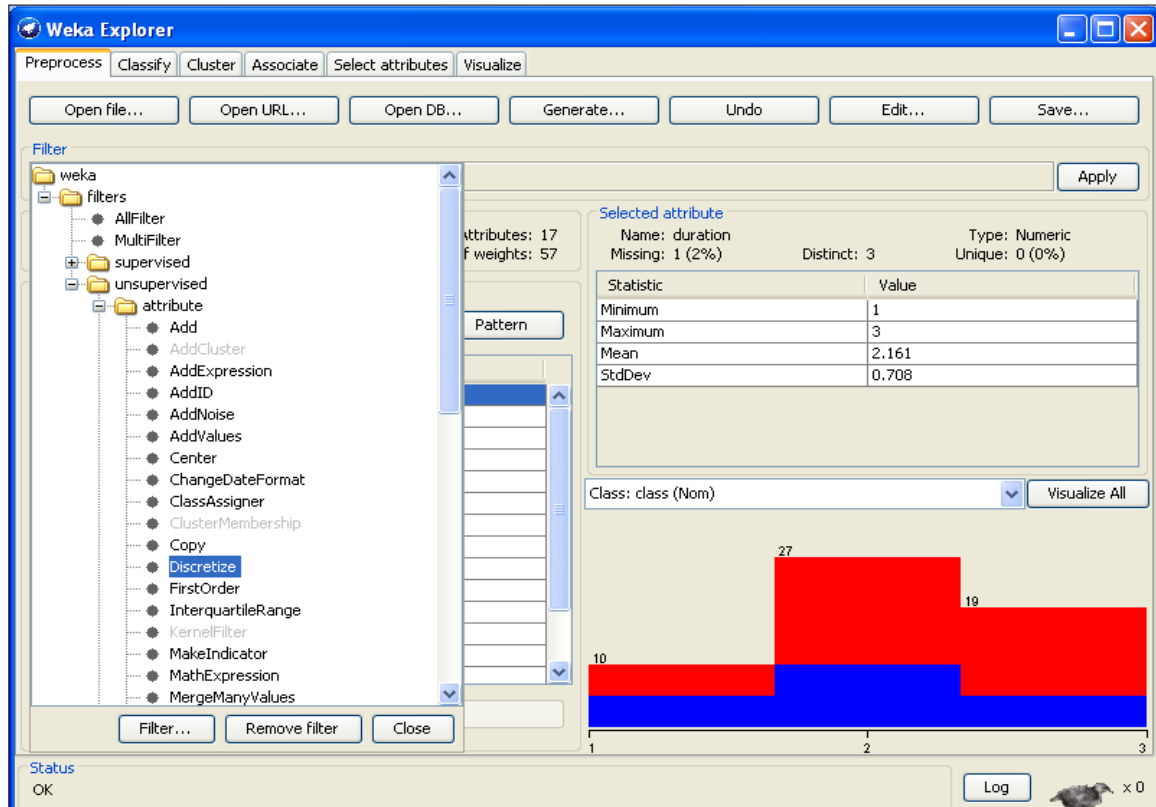
2. Click on the Explorer button and you get the Weka Knowledge Explorer window.



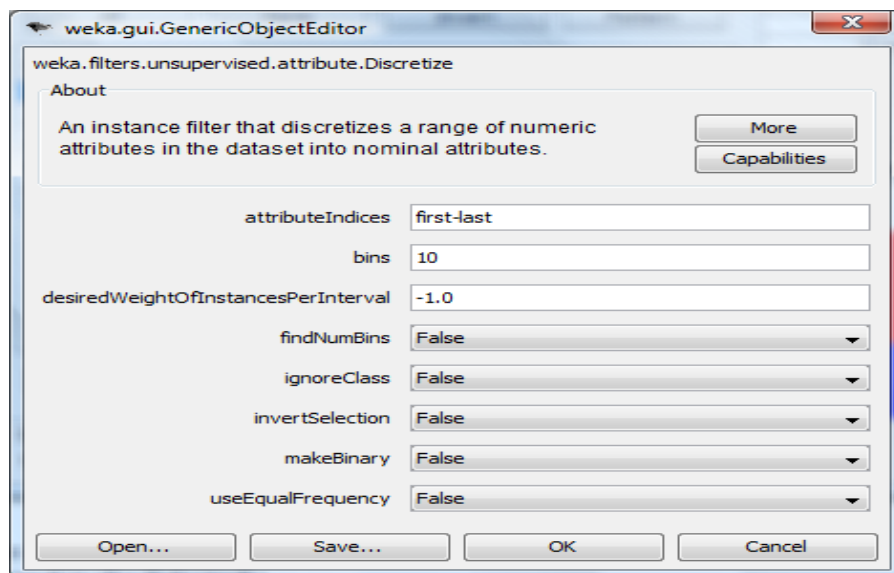
3. **VALIDATION :** on the “Open File.” button and open an ARFF file (try it first with an example supplied in Weka-3-6/data, e.g. Wether.arff). You get the following:



4. **VALIDATION :** Click on Choose and select filters/unsupervised/attribute/Discretize.



5. Then click on the area right of the Choose button. You get the following:



You see here the default parameters of this filter. Click on More to get more information about these parameters.

6. **VALIDATION :** Click on the Apply button to do the discretization. Then select one of the original numeric attributes (e.g. temperature) and see how it is discretized in the Selected attribute window.

The screenshot shows the Weka Explorer interface with the 'Discretize' filter applied to the 'duration' attribute. The 'Selected attribute' window displays the following table:

No.	Label	Count	Weight
1	'(-inf-1.2]'	10	10.0
2	'(1.2-1.4]'	0	0.0
3	'(1.4-1.6]'	0	0.0
4	'(1.6-1.8]'	0	0.0
5	'(1.8-2]'	27	27.0
6	'(2-2.2]'	0	0.0
7	'(2.2-2.4]'	0	0.0
8	'(2.4-2.6]'	0	0.0

The bar chart below the table shows the distribution of the 'duration' attribute across the bins. The y-axis represents the count, and the x-axis represents the bins. The bars are colored red and blue, with the total count for each bin shown above the bar.

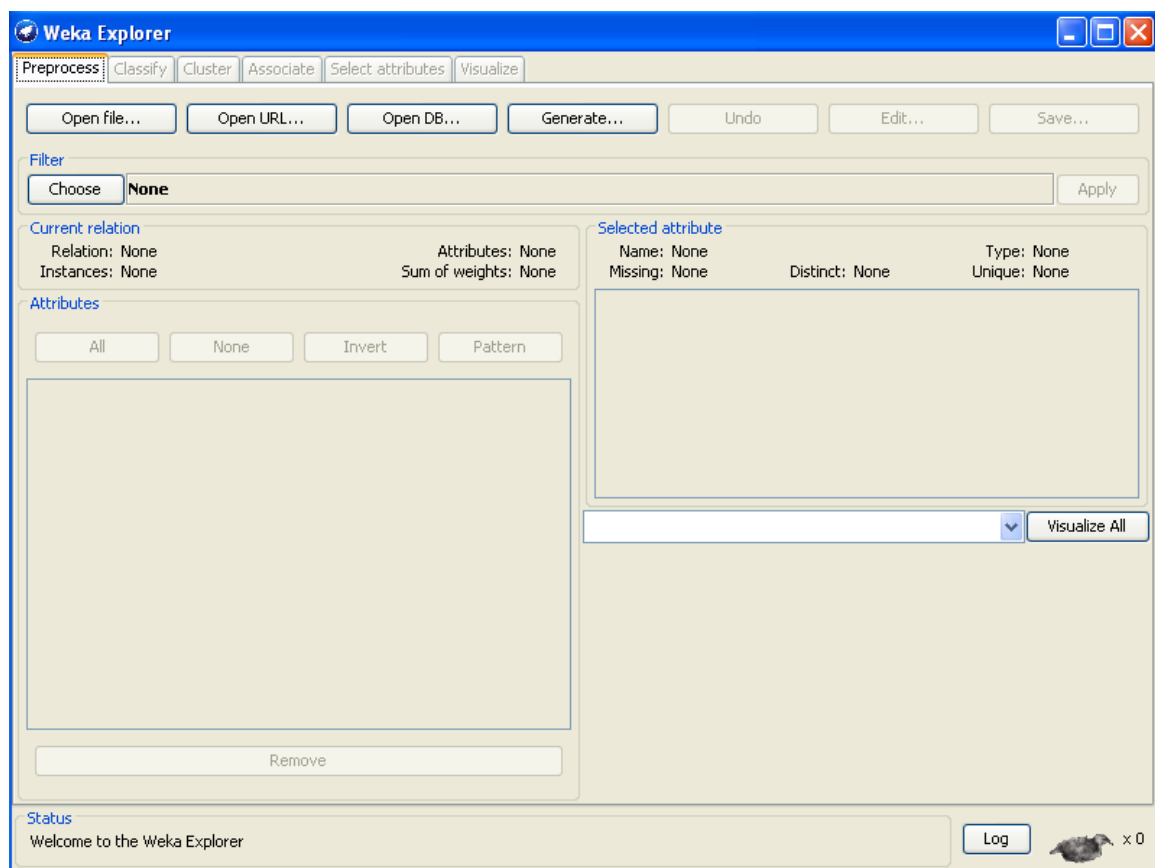
7. Try other parameters for the filter and see how the discretization changes. Don't forget to reload the original (numeric) relation or Undo the discretization before applying another one.

3. Normalize

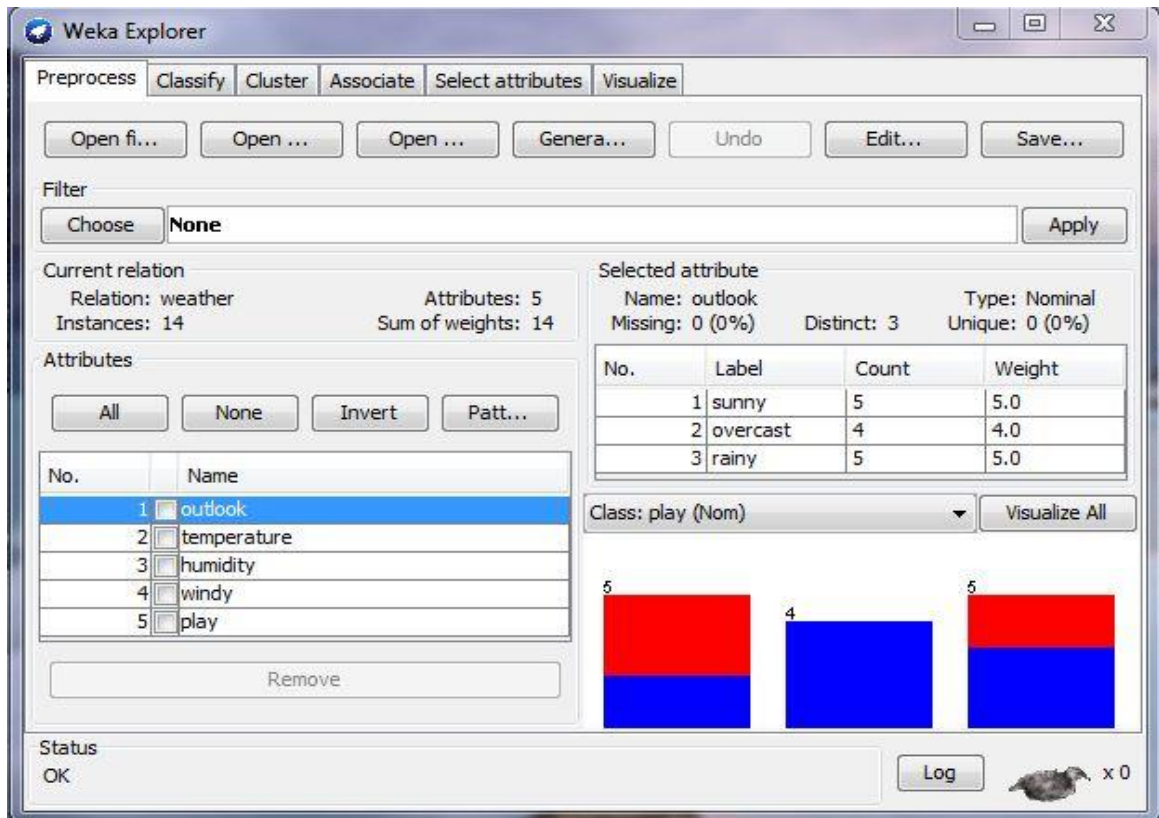
1. Start Weka – you get the Weka GUI chooser window.



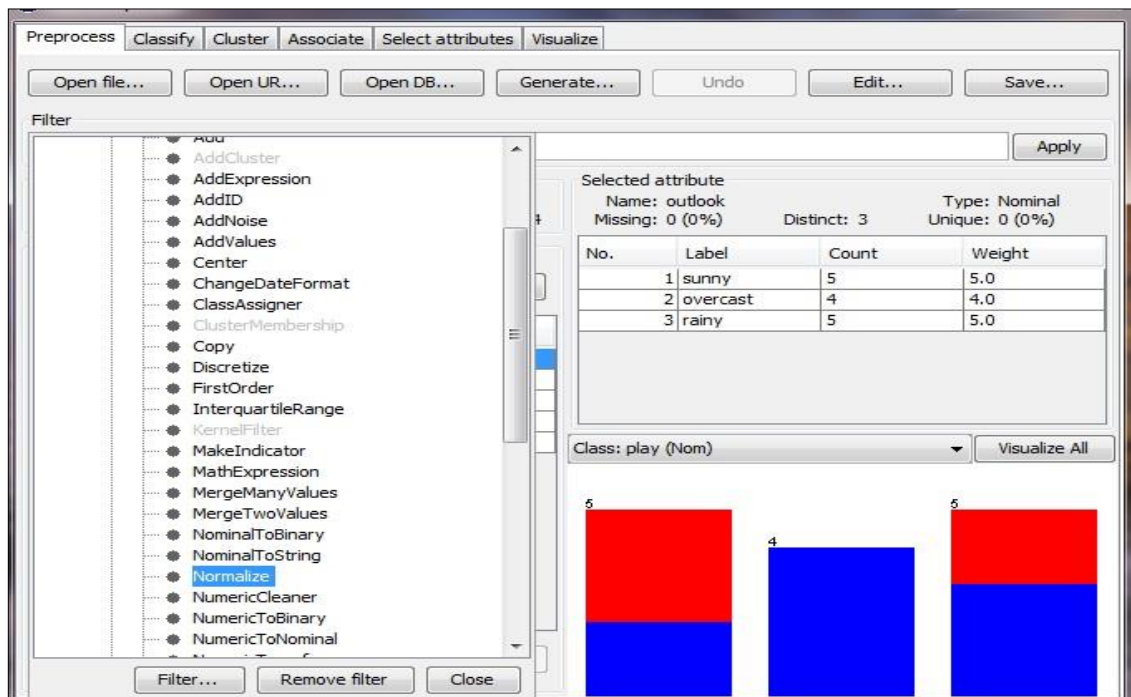
2. Click on the Explorer button and you get the Weka Knowledge Explorer window.

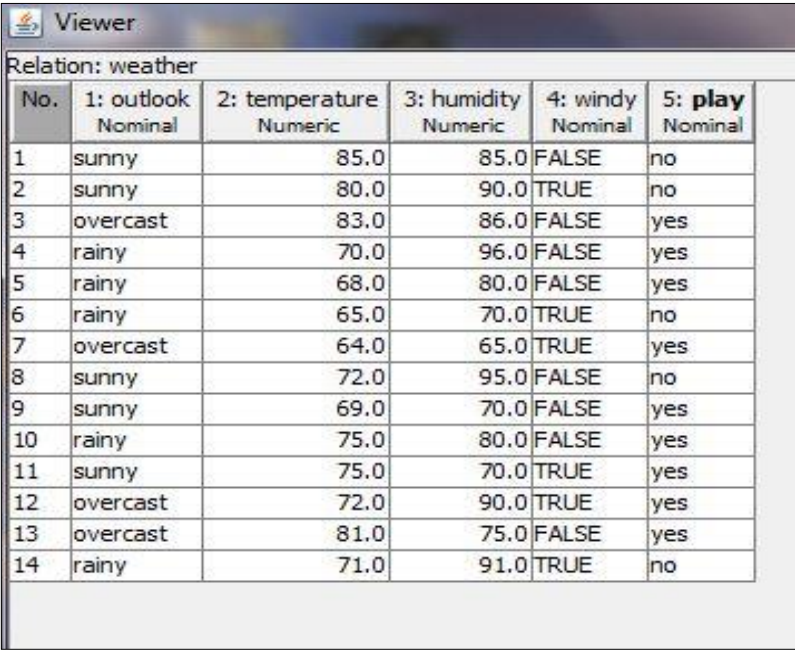


- Click on the “Open File.” button and open an ARFF file (try it first with an example supplied in Weka-3-6/data, e.g. whether.arff). You get the following:



- Click on Choose and select filters/unsupervised/attribute/Normalize.

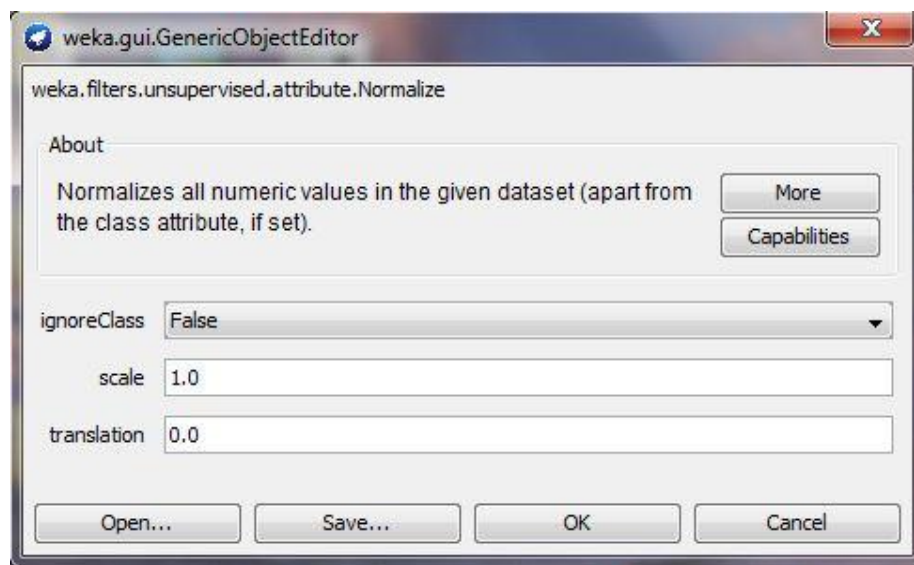




Relation: weather

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	80.0	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	70.0	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	sunny	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	TRUE	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

5. Then click on the area right of the Choose button. You get the following:



You see here the default parameters of this filter. Click on More to get more information about these parameters.

6. **VALIDATION :** Click on the Apply button to do the normalization. Then select edit tab to view data and see how it is normalized in the data window.

Viewer

Relation: weather-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	1.0	0.645161...	FALSE	no
2	sunny	0.761904761...	0.806451...	TRUE	no
3	overcast	0.904761904...	0.677419...	FALSE	yes
4	rainy	0.285714285...	1.0	FALSE	yes
5	rainy	0.190476190...	0.483870...	FALSE	yes
6	rainy	0.047619047...	0.161290...	TRUE	no
7	overcast	0.0	0.0	TRUE	yes
8	sunny	0.380952380...	0.967741...	FALSE	no
9	sunny	0.238095238...	0.161290...	FALSE	yes
10	rainy	0.523809523...	0.483870...	FALSE	yes
11	sunny	0.523809523...	0.161290...	TRUE	yes
12	overcast	0.380952380...	0.806451...	TRUE	yes
13	overcast	0.809523809...	0.322580...	FALSE	yes
14	rainy	0.333333333...	0.838709...	TRUE	no

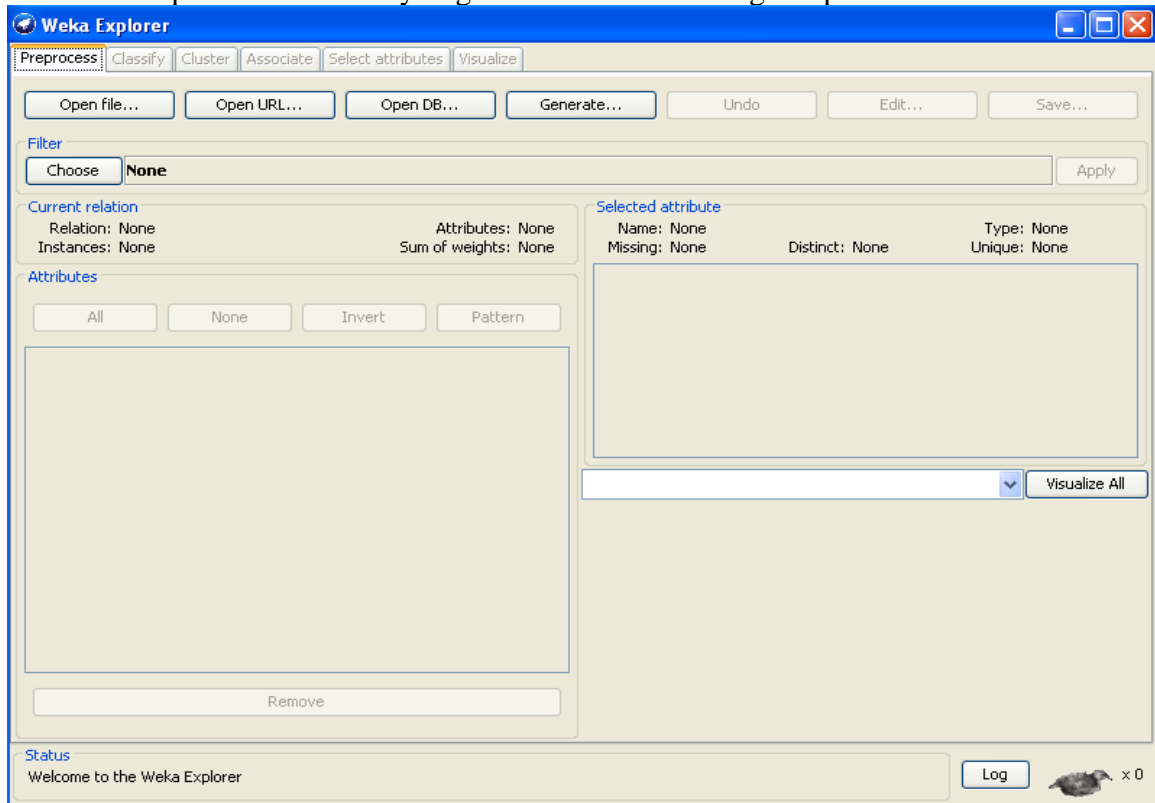
- Try other parameters for the filter and see how the normalization changes. Don't forget to reload the original (numeric) relation or Undo the normalization before applying another one.

4. Remove

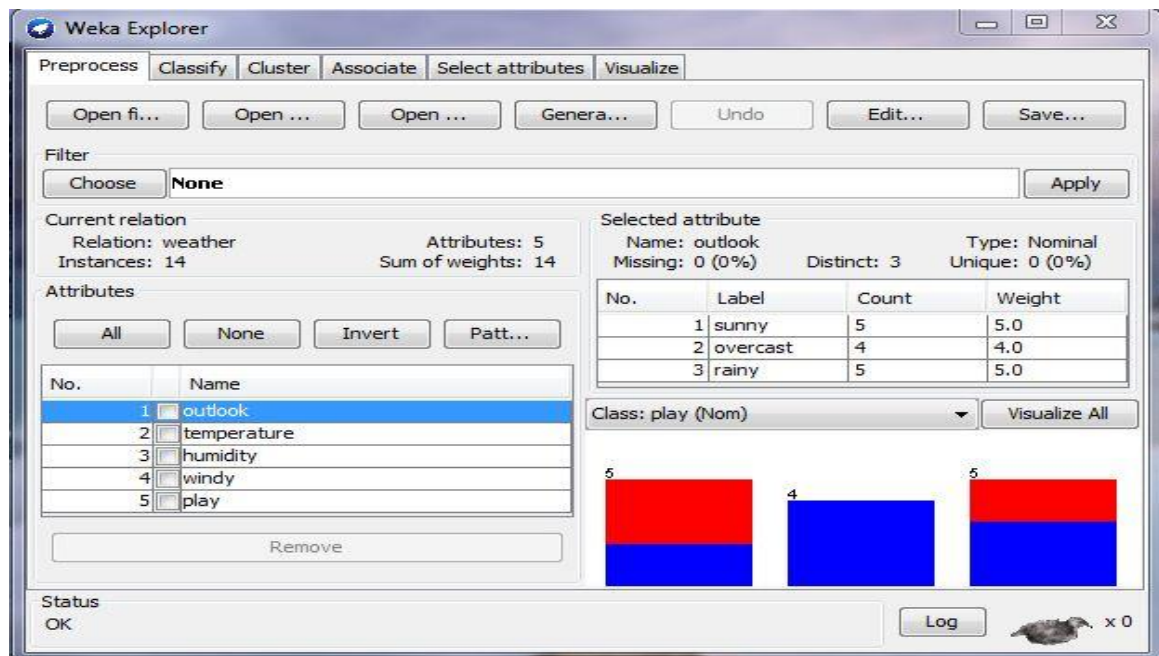
- Start Weka – you get the Weka GUI chooser window.



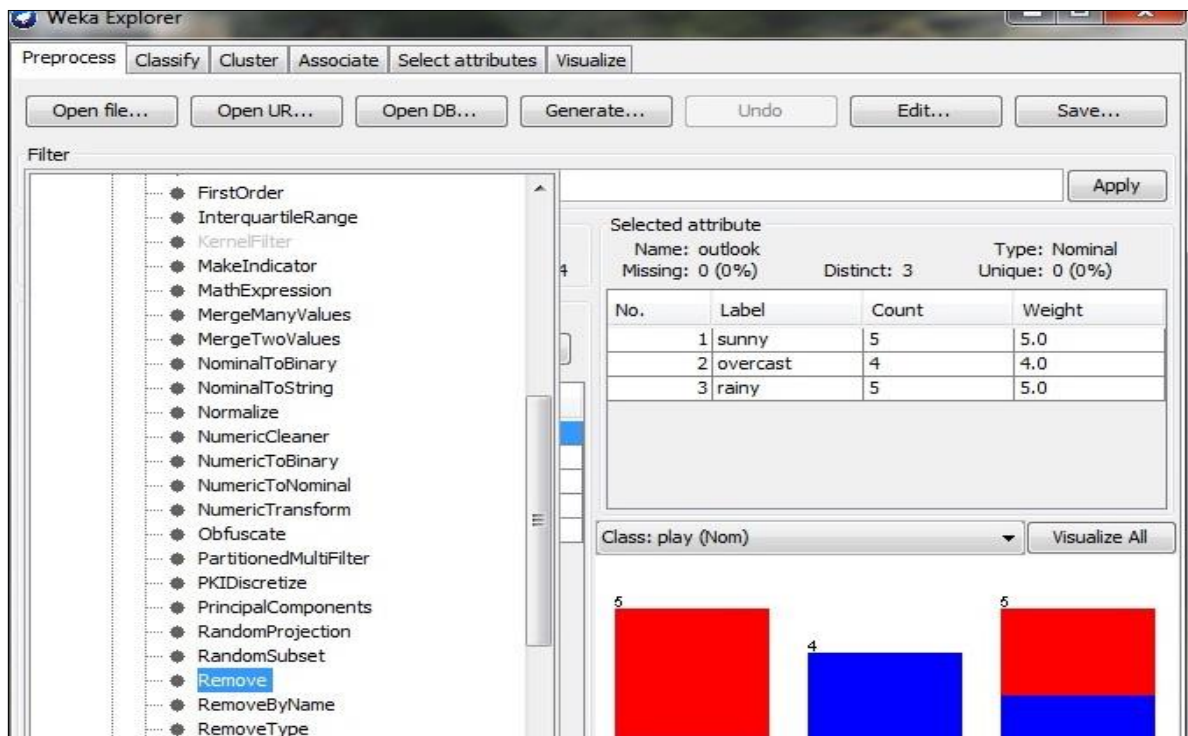
2. Click on the Explorer button and you get the Weka Knowledge Explorer window.



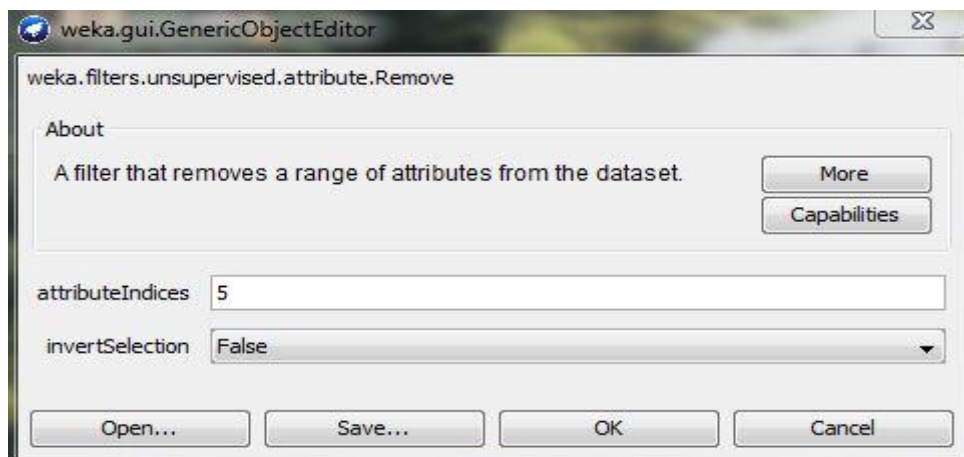
3. Click on the “Open File.” button and open an ARFF file (try it first with an example supplied in Weka-3-6/data, e.g. weather.arff). You get the following:



4. Click on Choose and select filters/unsupervised/attribute/Remove.

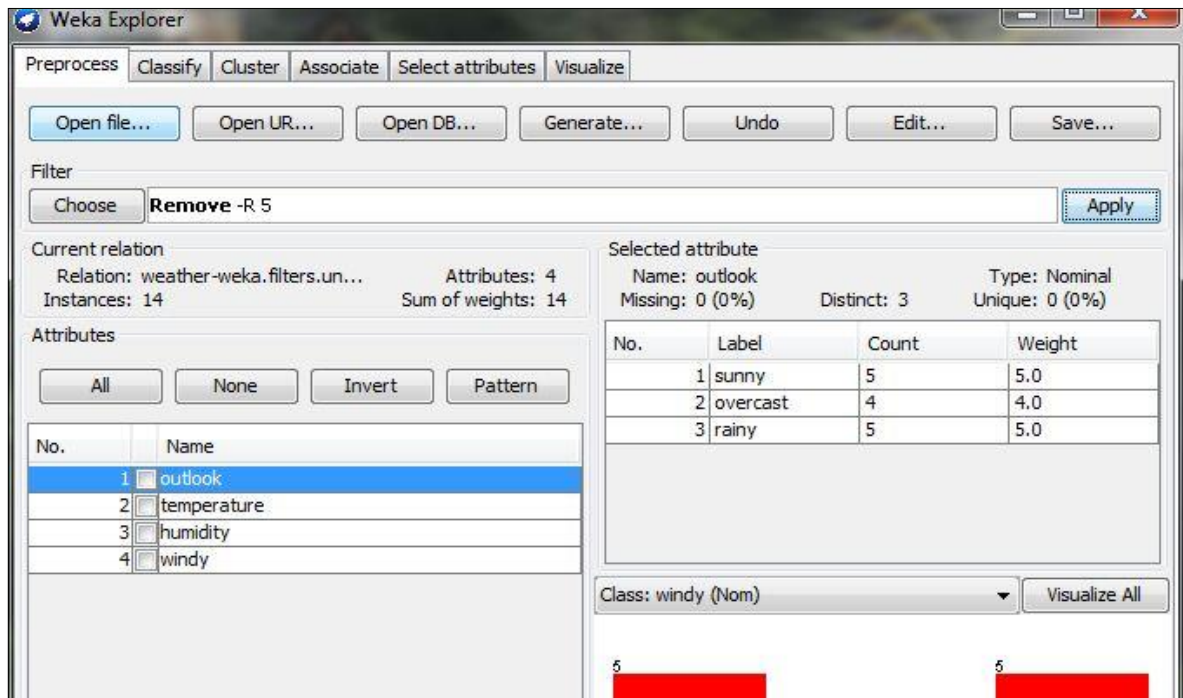


5. Then click on the area right of the Choose button. You get the following:



You see here the default parameters of this filter. Enter the Indices of attribute to be remove Click on more to get more information about these parameters.

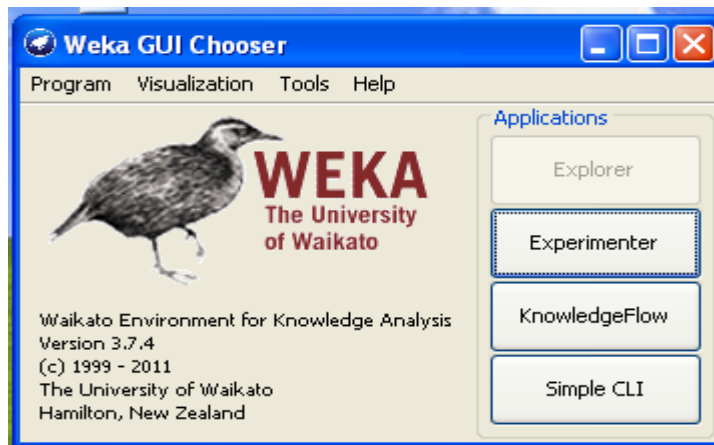
6. **VALIDATION** : Click on the Apply button to do the remove.



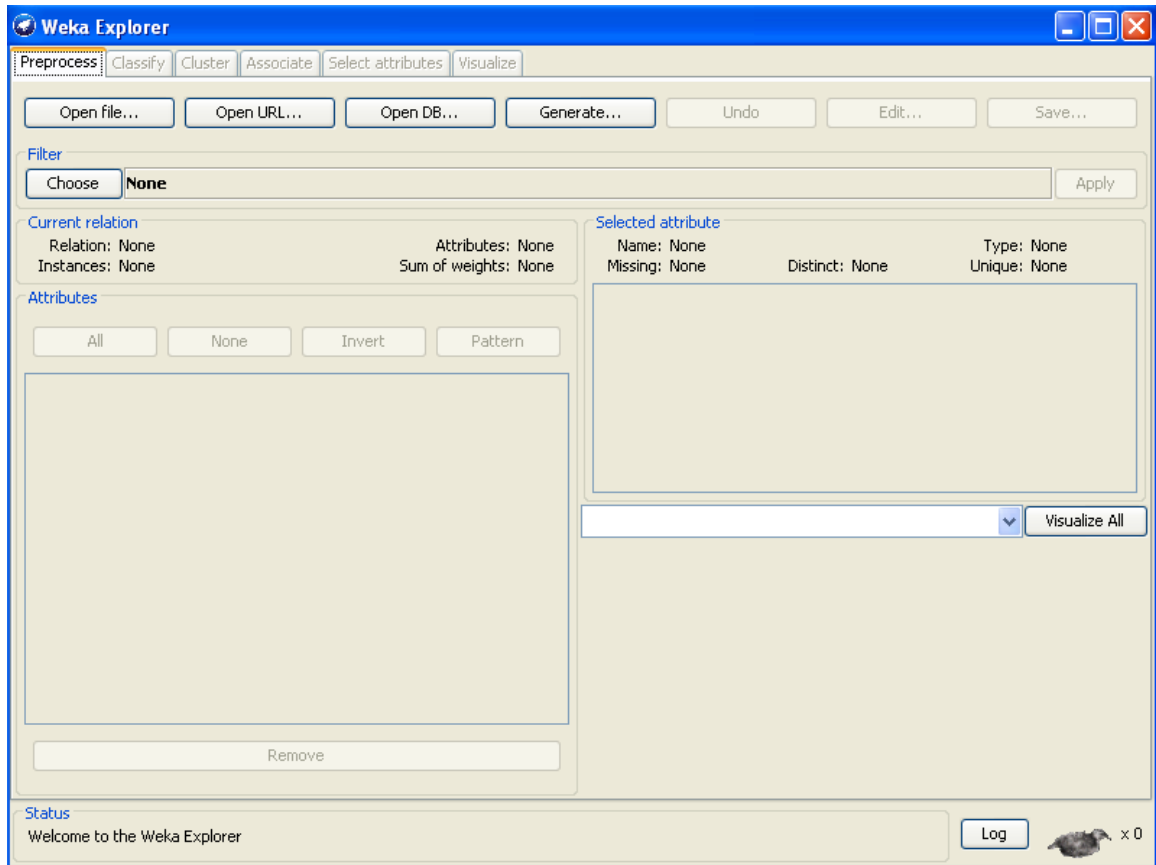
7. Try other parameters for the filter and see how the remove changes. Don't forget to reload the original (numeric) relation or Undo the remove before applying another one.

5. Replace Missing Values

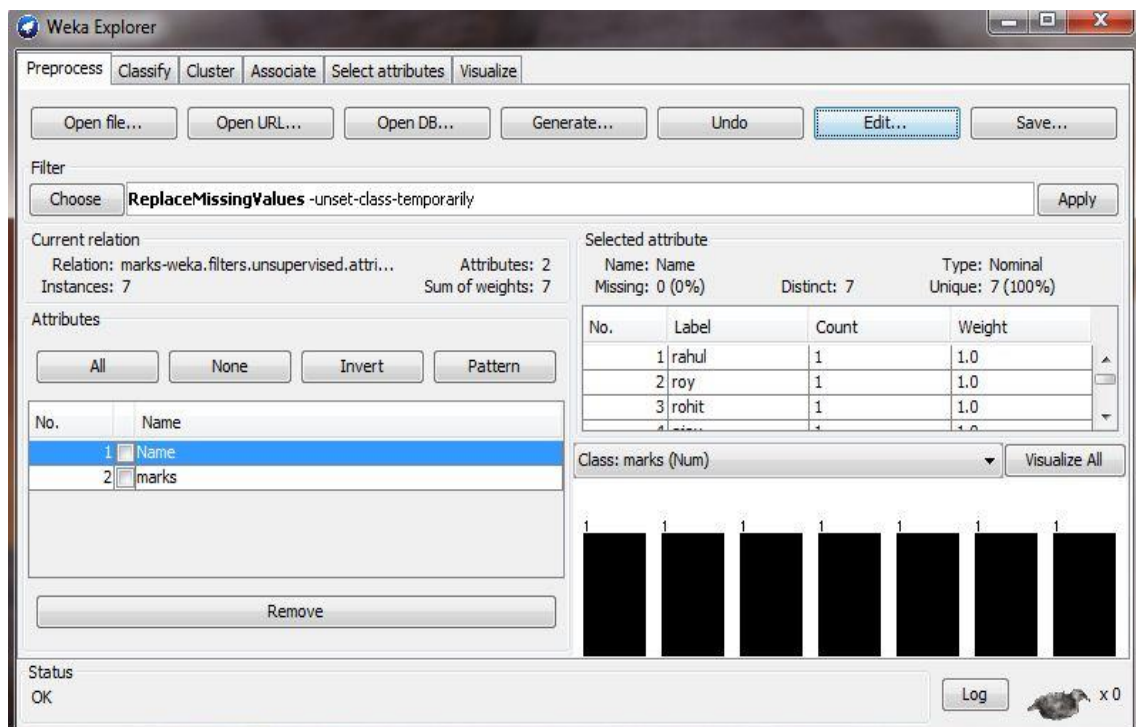
1. Start Weka – you get the Weka GUI chooser window.



2. Click on the Explorer button and you get the Weka Knowledge Explorer window.

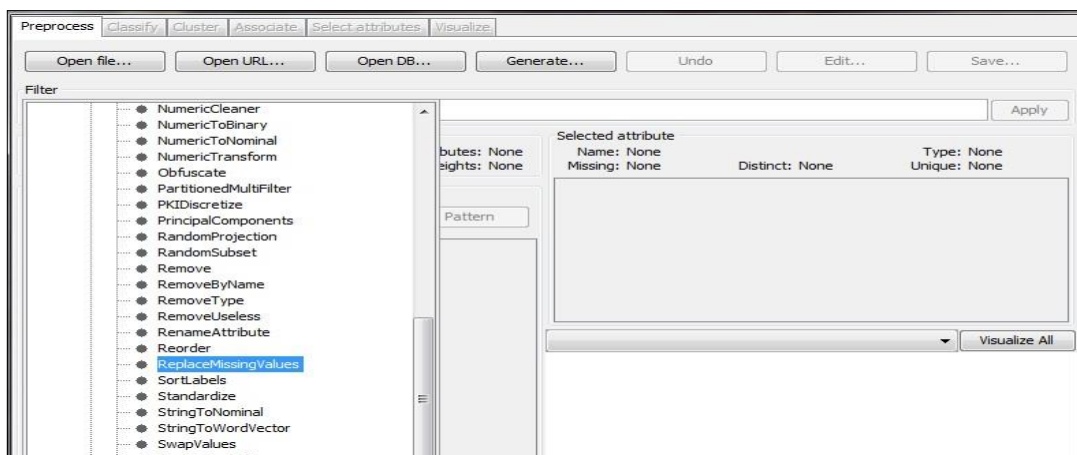


3. Click on the “Open File.” button and open an ARFF file (try it first with an example supplied in Weka-3-6/data, e.g. weather.arff). You get the following:

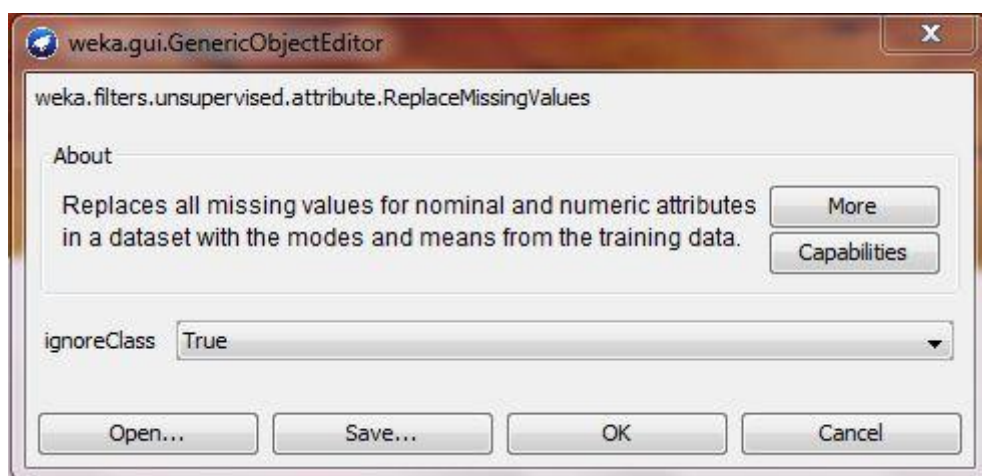


No.	1: Name Nominal	2: marks Numeric
1	rahul	
2	roy	55.0
3	rohit	22.0
4	ajay	87.0
5	asad	
6	asha	65.0
7	seema	

4. Click on Choose and select filters/unsupervised/attribute/ReplaceMissingValues.

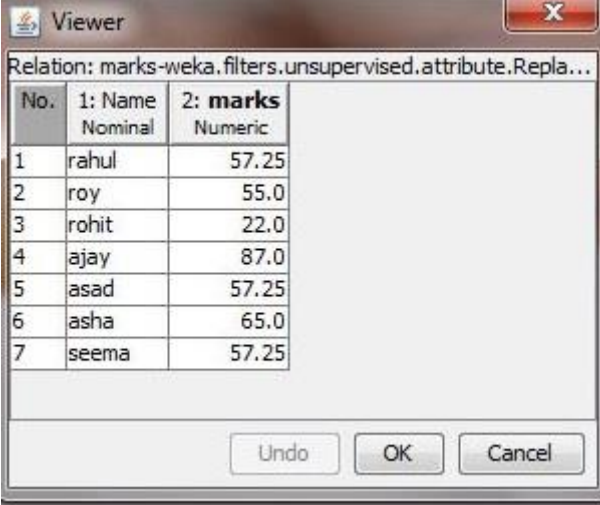


5. Then click on the area right of the Choose button. You get the following:



You see here the default parameters of this filter. Click on more to get more information about these parameters.

6. **VALIDATION :** Click on the Apply button to do the Replace Missing Values. Then select edit tab to view data and see how it Replaced missing values in the data window.



Relation: marks-weka.filters.unsupervised.attribute.Repla...

No.	1: Name Nominal	2: marks Numeric
1	rahul	57.25
2	roy	55.0
3	rohit	22.0
4	ajay	87.0
5	asad	57.25
6	asha	65.0
7	seema	57.25

Buttons: Undo, OK, Cancel

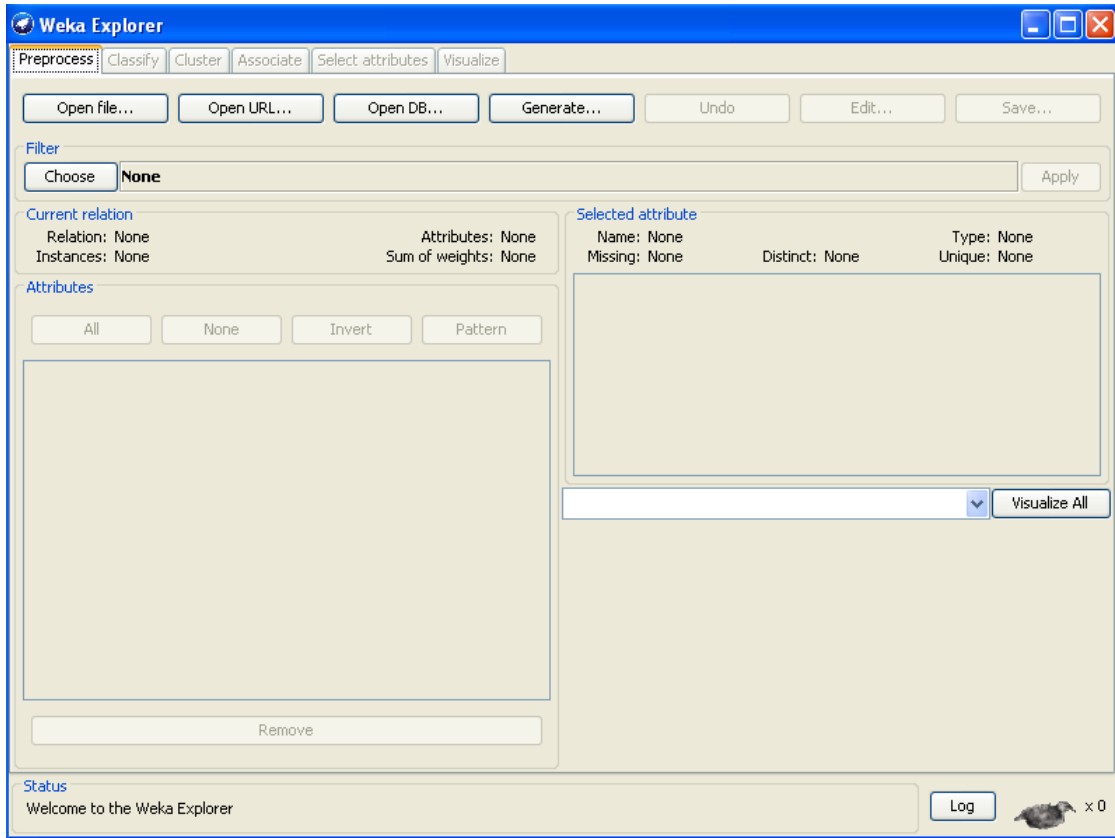
7. Try other parameters for the filter and see how the replace values changes. Don't forget to reload the original (numeric) relation or Undo the replaced before applying another one.

6. Standardize

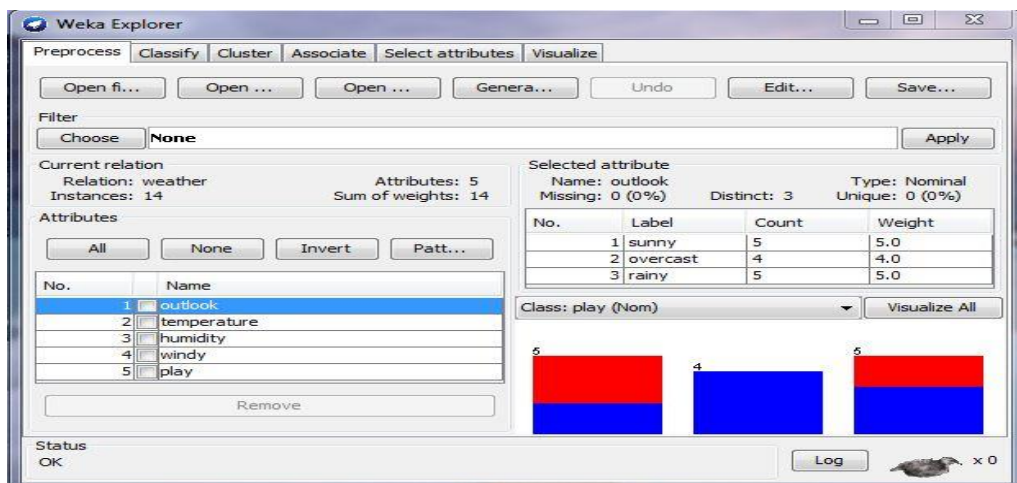
1. Start Weka – you get the Weka GUI chooser window.



- Click on the Explorer button and you get the Weka Knowledge Explorer window.

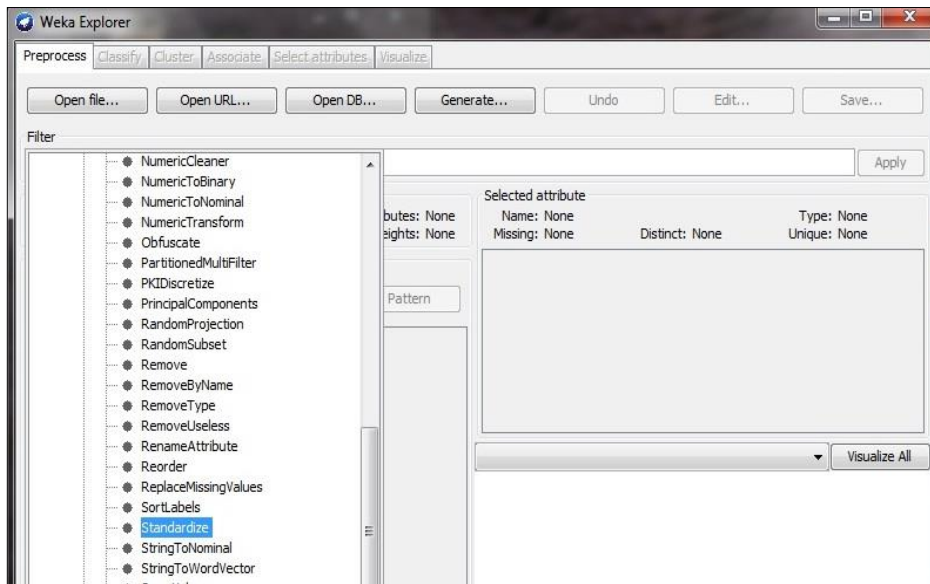


- Click on the “Open File.” button and open an ARFF file (try it first with an example supplied in Weka-3-6/data, e.g. weather.arff). You get the following:

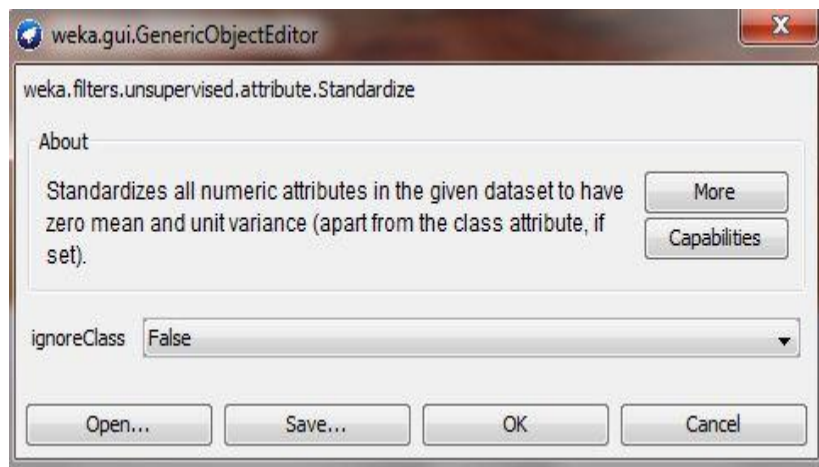


No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	80.0	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	70.0	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	sunny	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	TRUE	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

4. Click on Choose and select filters/unsupervised/attribute/Standardize.

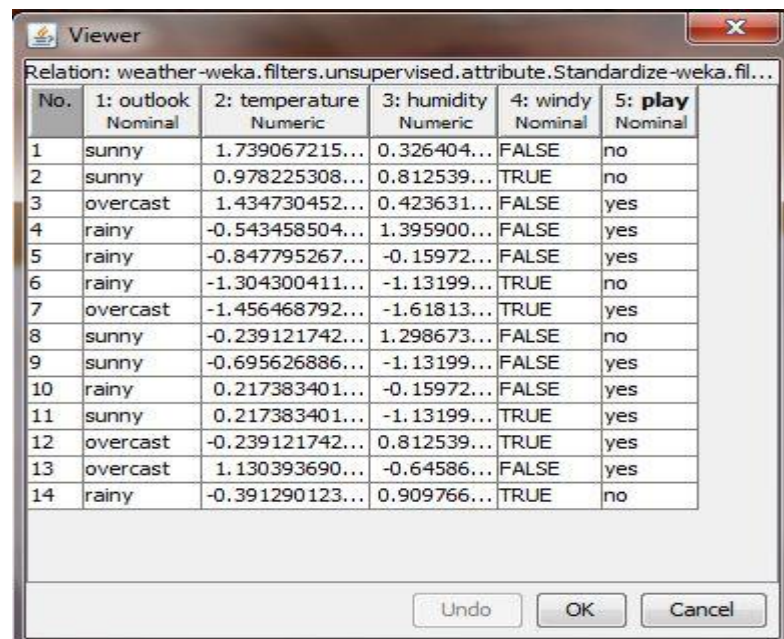


5. Then click on the area right of the Choose button. You get the following:



You see here the default parameters of this filter. Click on more to get more information about these parameters.

6. **VALIDATION :** Click on the Apply button to do the Standardization. Then select edit tab to view data and see how it standard the values in the data window.

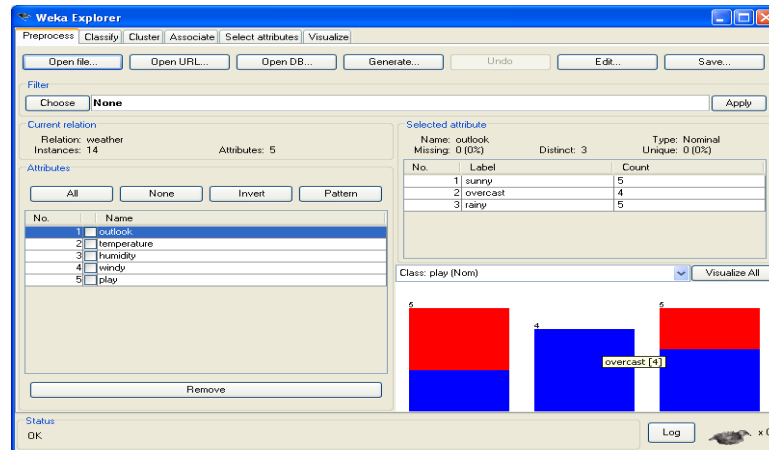


No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	1.739067215...	0.326404...	FALSE	no
2	sunny	0.978225308...	0.812539...	TRUE	no
3	overcast	1.434730452...	0.423631...	FALSE	yes
4	rainy	-0.543458504...	1.395900...	FALSE	yes
5	rainy	-0.847795267...	-0.15972...	FALSE	yes
6	rainy	-1.304300411...	-1.13199...	TRUE	no
7	overcast	-1.456468792...	-1.61813...	TRUE	yes
8	sunny	-0.239121742...	1.298673...	FALSE	no
9	sunny	-0.695626886...	-1.13199...	FALSE	yes
10	rainy	0.217383401...	-0.15972...	FALSE	yes
11	sunny	0.217383401...	-1.13199...	TRUE	yes
12	overcast	-0.239121742...	0.812539...	TRUE	yes
13	overcast	1.130393690...	-0.64586...	FALSE	yes
14	rainy	-0.391290123...	0.909766...	TRUE	no

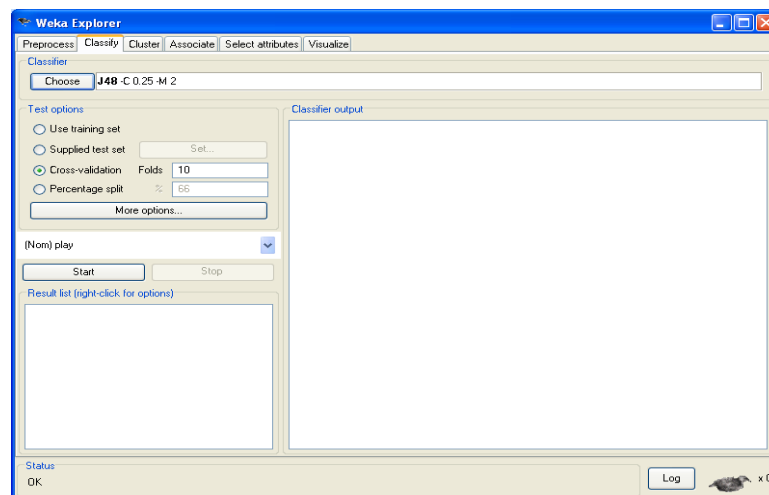
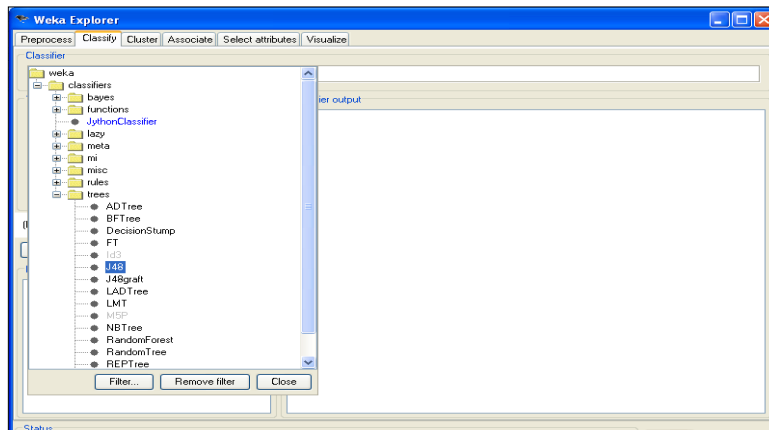
7. Try other parameters for the filter and see how the standardize changes. Don't forget to reload the original (numeric) relation or Undo the standardize before applying another one.

PROGRAM 7. Classification of algorithms using WEKA

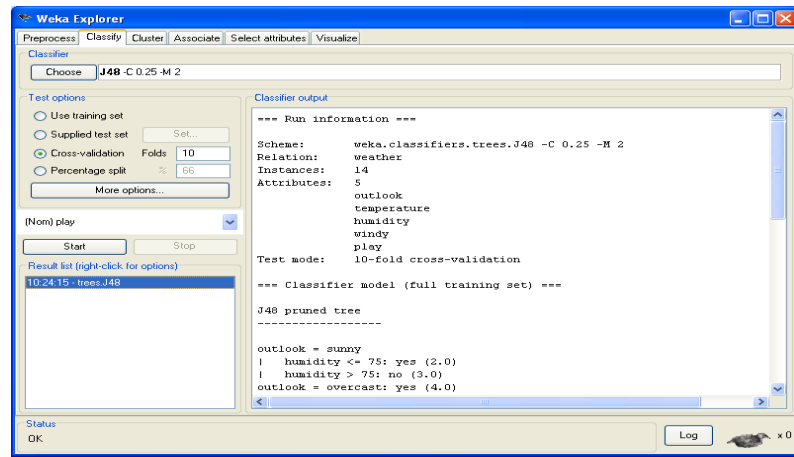
7.1. Aim : Obtain decision tree for different data sets using WEKA



Click on classify, then choose

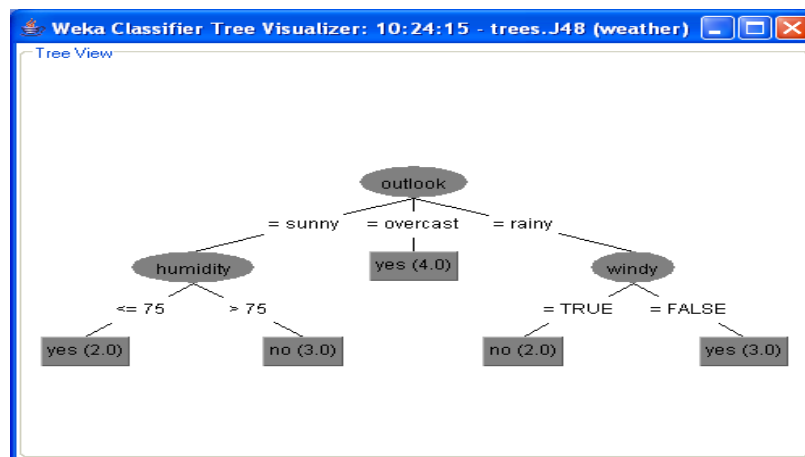
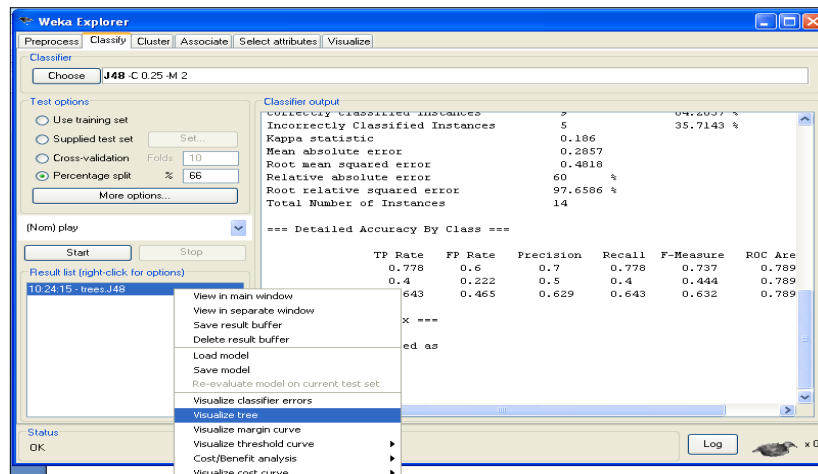


Click on start



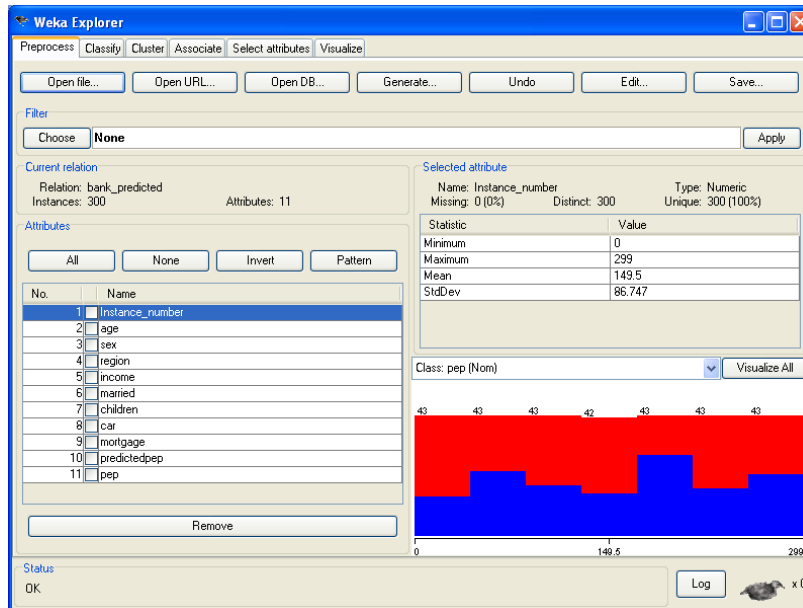
Click on percentage split

Click on 102415-treesJ48



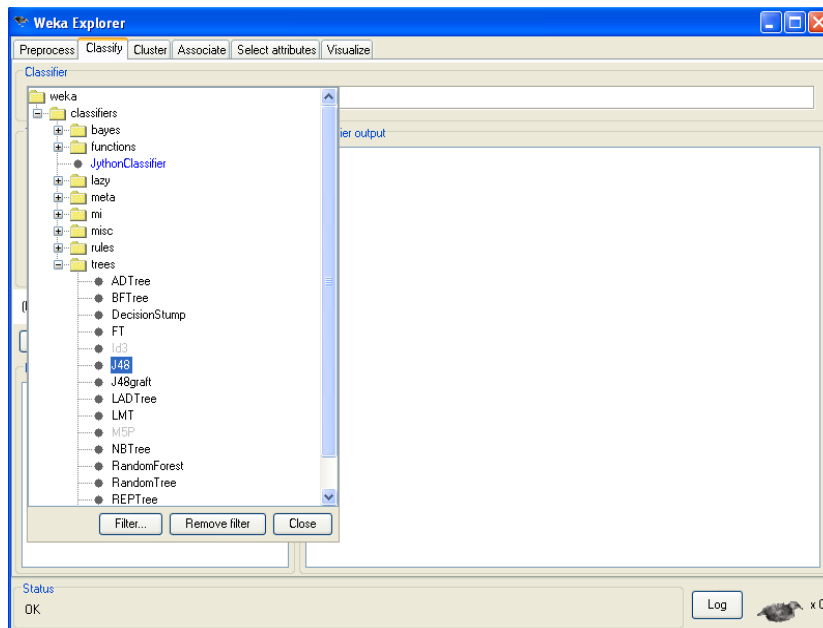
Second data set:

Bankprediction data set

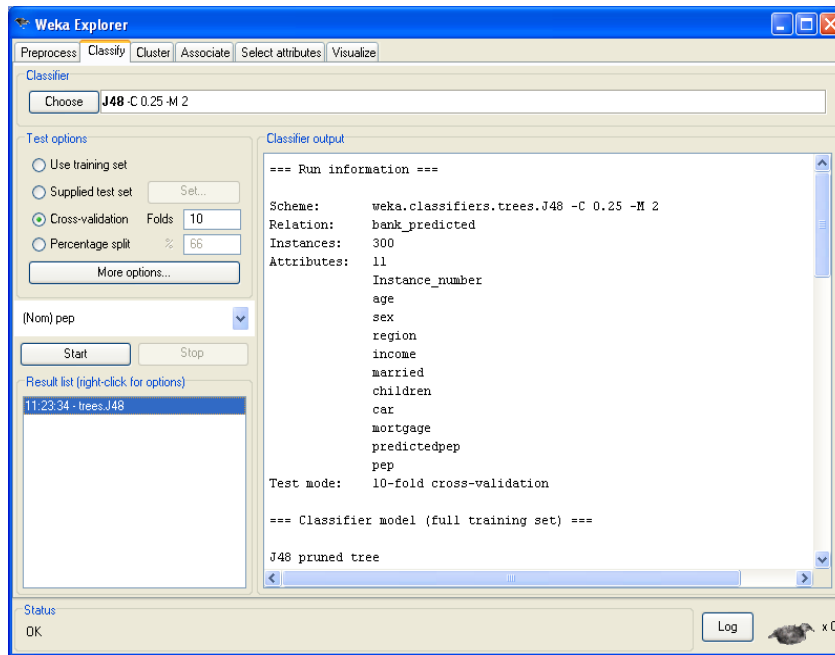


Click on classify

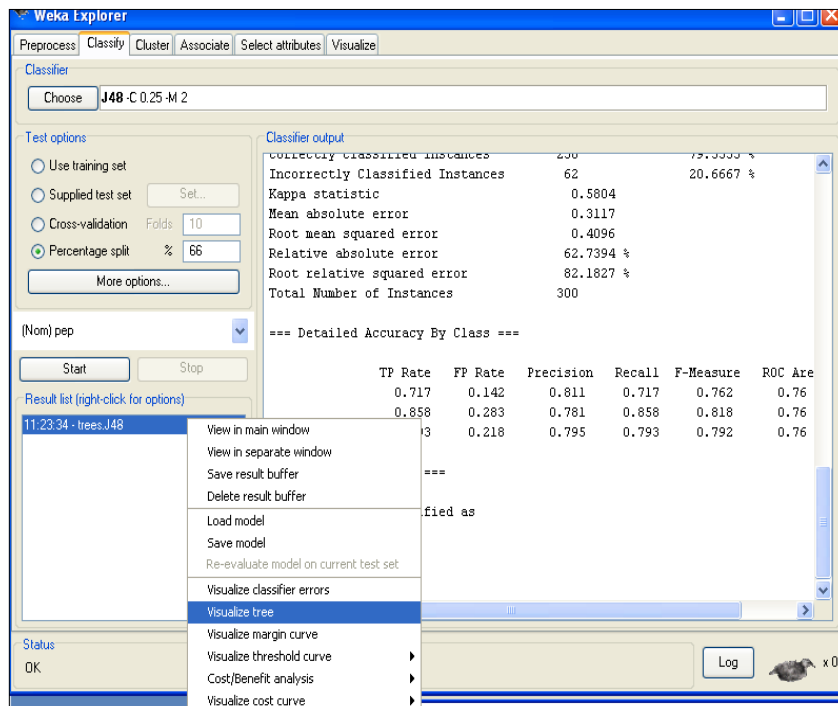
Then click on choose and in trees choose j48

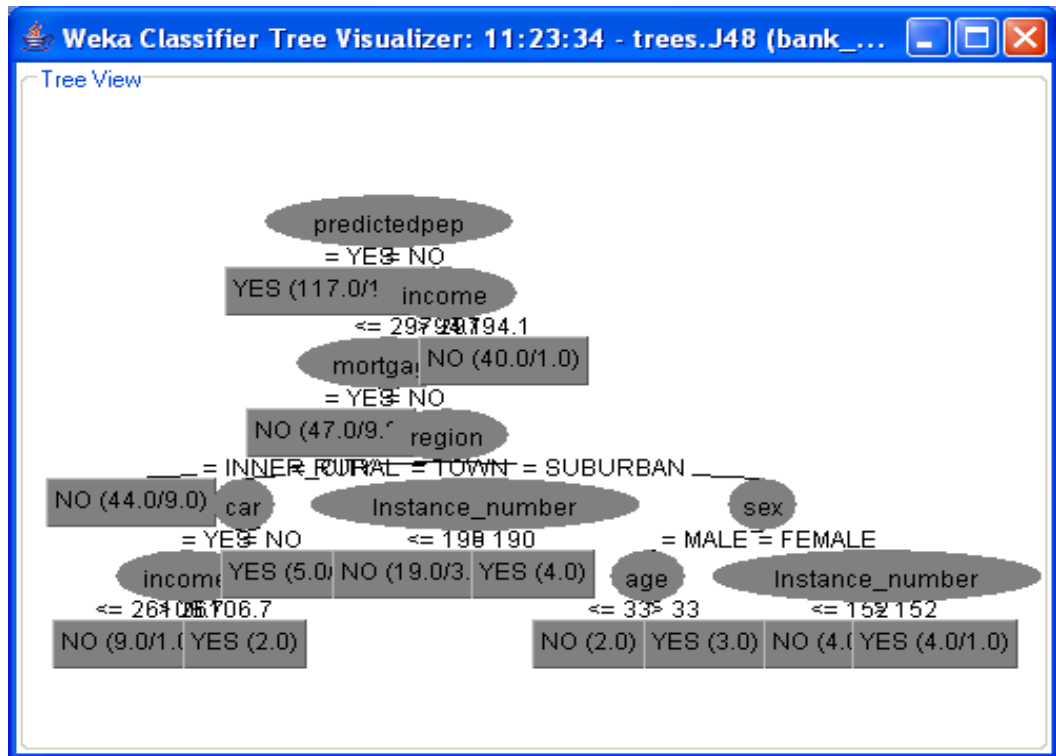


Click on start



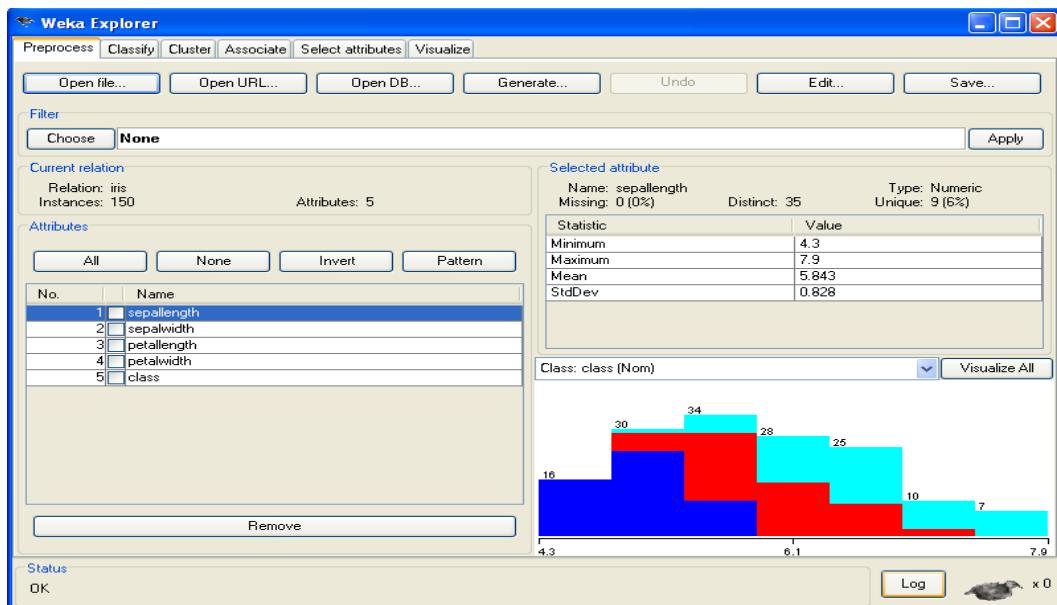
Click on percentage split



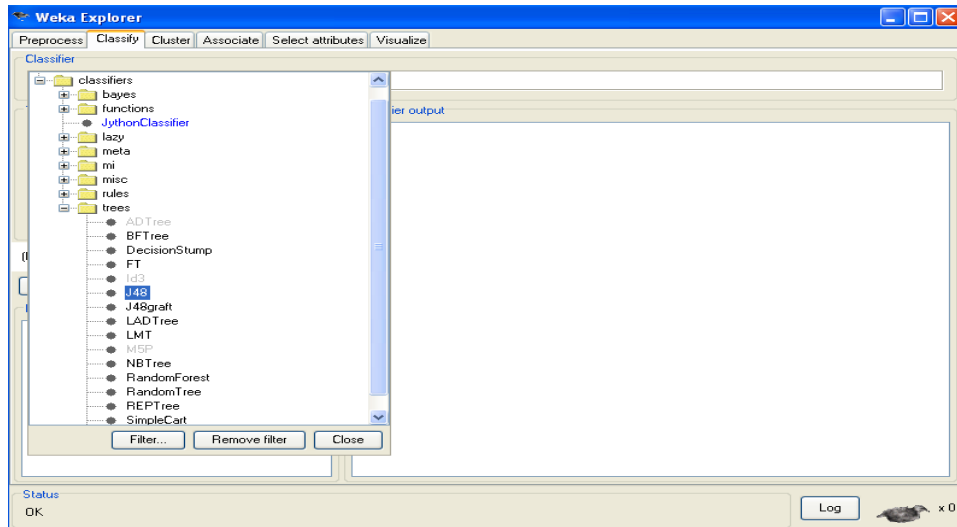


Third data set:

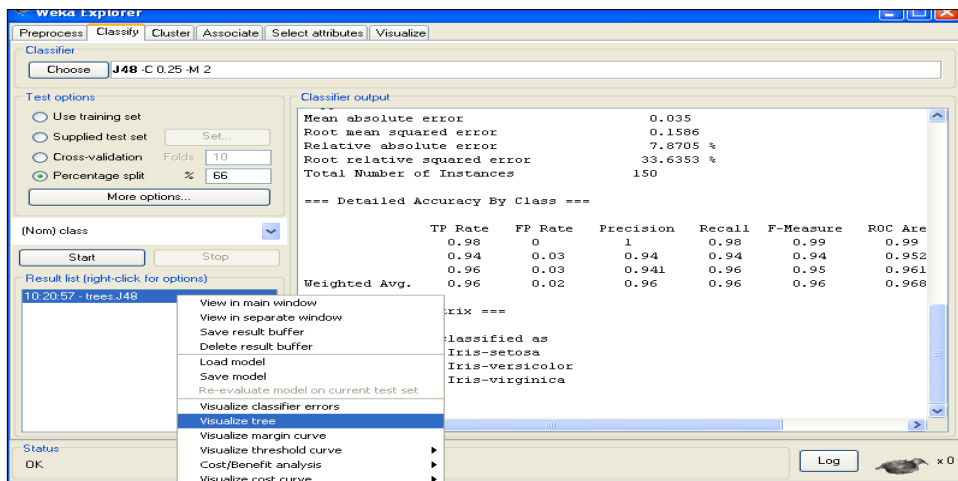
Sleep data set



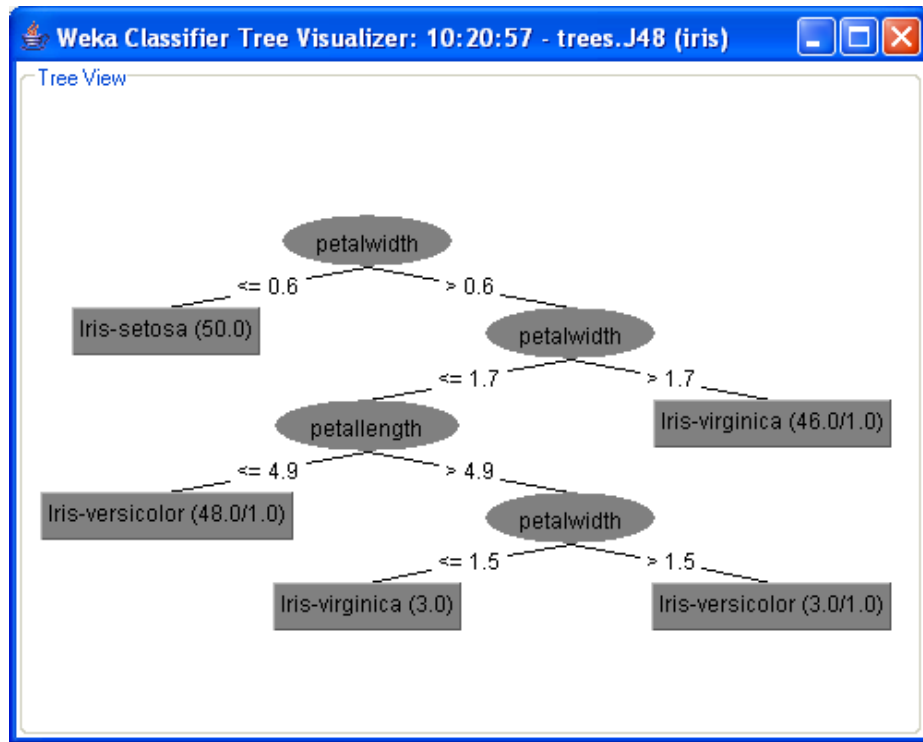
Click on classify and choose from trees choose j48 algorithm



Click on start and then click on percentage split

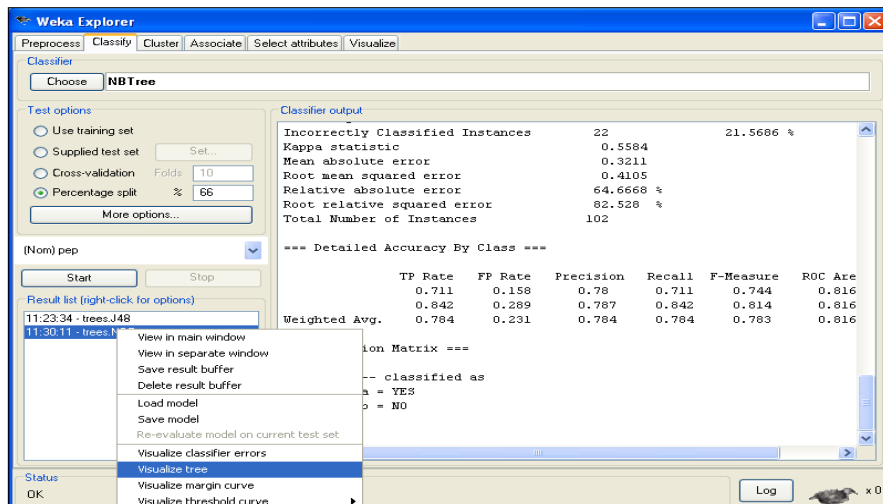
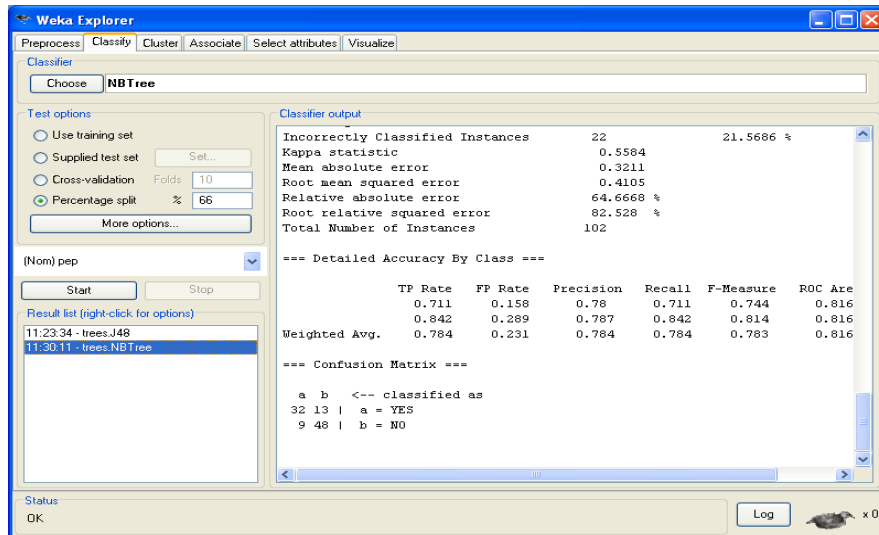
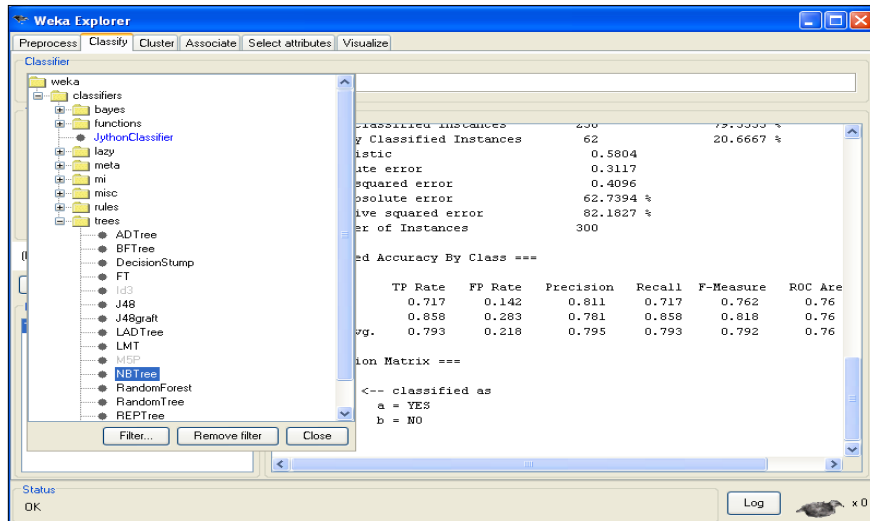


VALIDATION :



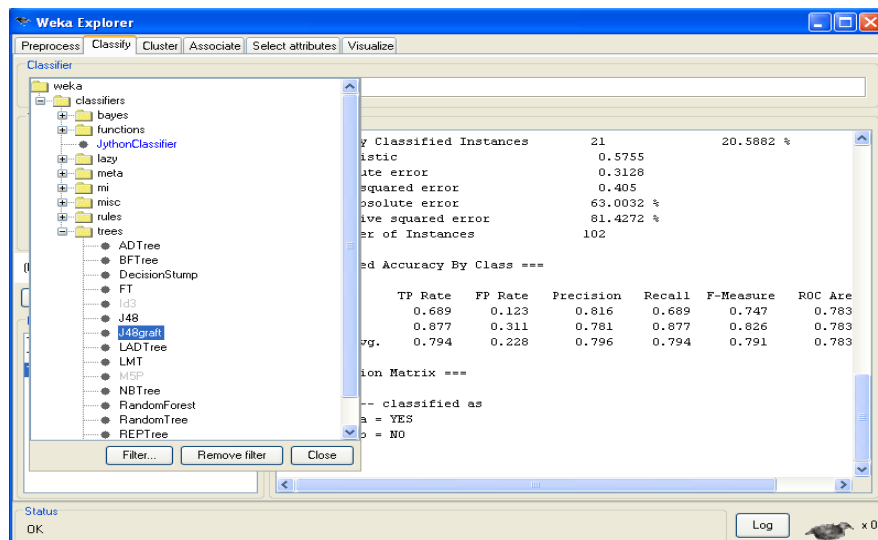
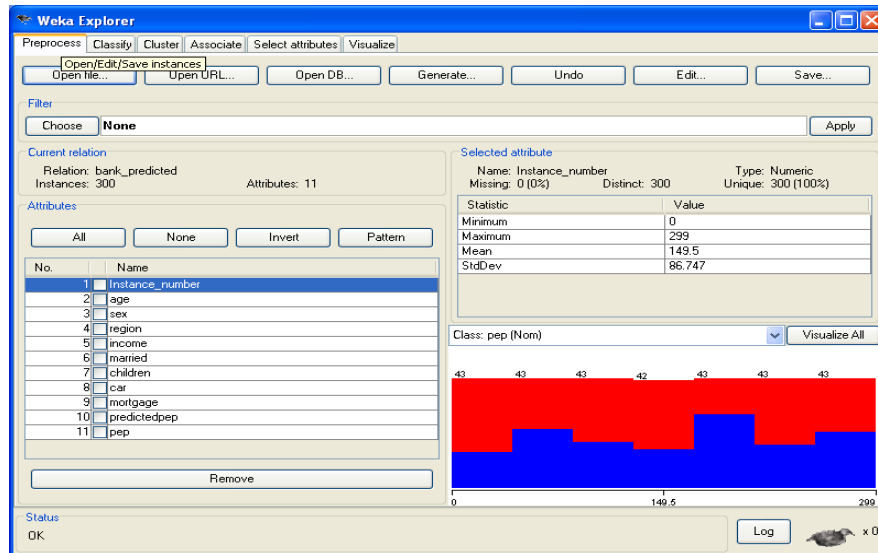
7.2. Aim: Classification algorithms using WEKA.

Bank prediction data set using nb tree:





Bank prediction data set using j48graft:



Weka Explorer

Preprocess | **Classify** | Cluster | Associate | Select attributes | Visualize

Classifier: Choose **J48graft-C 0.25-M 2**

Test options:
 Use training set
 Supplied test set (Set...)
 Cross-validation (Folds: 10)
 Percentage split (%: 66)
 More options...

Classifier output:

```

Incorrectly Classified Instances    21          20.5882 %
Kappa statistic                    0.5755
Mean absolute error                 0.3128
Root mean squared error            0.405
Relative absolute error             63.0032 %
Root relative squared error        81.4272 %
Total Number of Instances         102

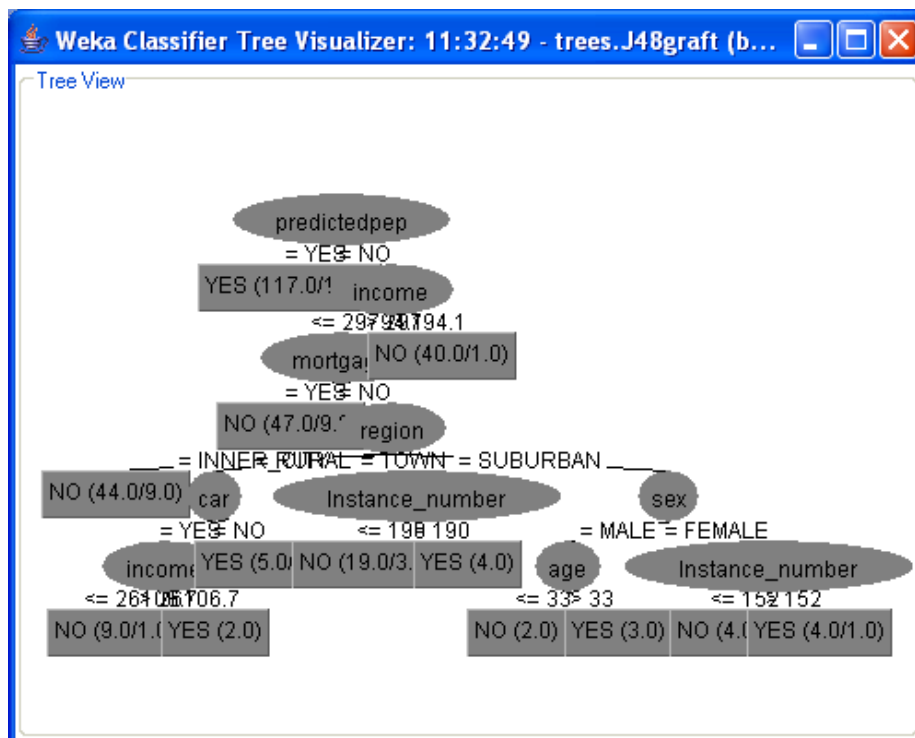
=== Detailed Accuracy By Class ===
                TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area
Weighted Avg.   0.794   0.228    0.796     0.794   0.791     0.783

=== Confusion Matrix ===
-- classified as
a = YES
b = NO
    
```

Result list (right-click for options):
 11:23:34 - trees.J48
 11:30:11 - trees.NBTree
 11:31:30 - trees.J48graft
 11:32:49 - trees.J48graft

Context menu for 11:32:49 - trees.J48graft:
 View in main window
 View in separate window
 Save result buffer
 Delete result buffer
 Load model
 Save model
 Re-evaluate model on current test set
 Visualize classifier errors
 Visualize tree

Status: OK



Bank prediction data set using nb tree:

Bankpred dataset: using NBtree

Classifier
Choose **NBTree**

Test options
 Use training set
 Supplied test set (Set...)
 Cross-validation Folds
 Percentage split %
 More options...

(Nom) pep

Start Stop

Result list (right-click for options)
11:17:49 - trees.NBTree

Classifier output

```

Mean absolute error          0.0478
Root mean squared error      0.1621
Relative absolute error      10.7474 %
Root relative squared error  34.3807 %
Total Number of Instances    150

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area
          1         0         1           1         1           1
          0.92     0.05     0.902     0.92     0.911     0.991
          0.9         0.04     0.918     0.9         0.909     0.983
Weighted Avg.  0.94     0.03     0.94     0.94     0.94     0.991

=== Confusion Matrix ===

 a b c <-- classified as
50 0 0 | a = Iris-setosa
 0 46 4 | b = Iris-versicolor
 0 5 45 | c = Iris-virginica
    
```

Status
OK

Log x 0

Classifier
Choose **NBTree**

Test options
 Use training set
 Supplied test set (Set...)
 Cross-validation Folds
 Percentage split %
 More options...

(Nom) pep

Start Stop

Result list (right-click for options)
11:17:49 - trees.NBTree

Classifier output

```

Mean absolute error          0.0478
Root mean squared error      0.1621
Relative absolute error      10.7474 %
Root relative squared error  34.3807 %
Total Number of Instances    150

=== Detailed Accuracy By Class ===

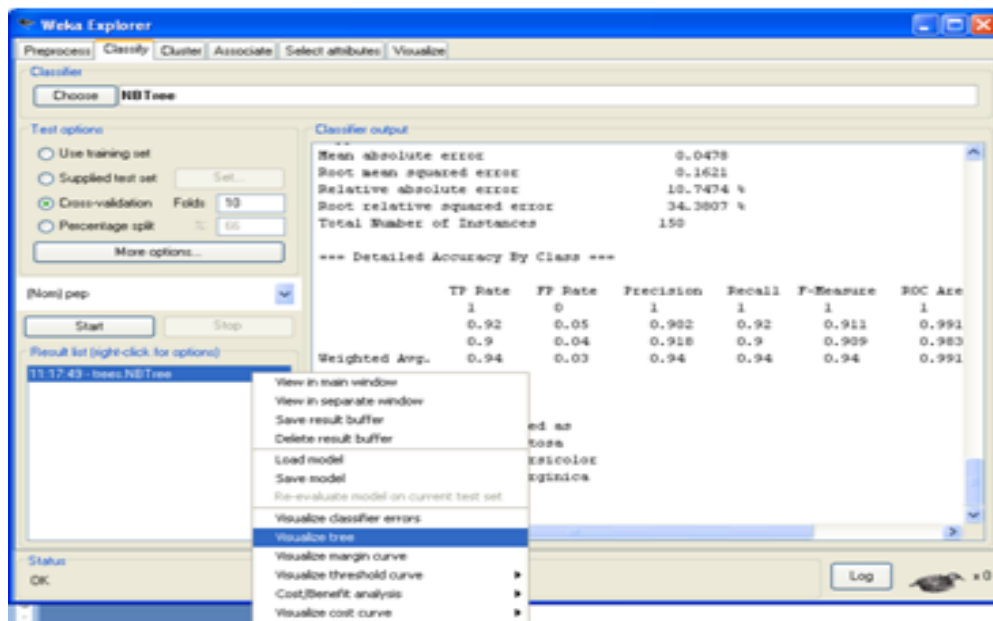
          TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area
          1         0         1           1         1           1
          0.92     0.05     0.902     0.92     0.911     0.991
          0.9         0.04     0.918     0.9         0.909     0.983
Weighted Avg.  0.94     0.03     0.94     0.94     0.94     0.991

=== Confusion Matrix ===

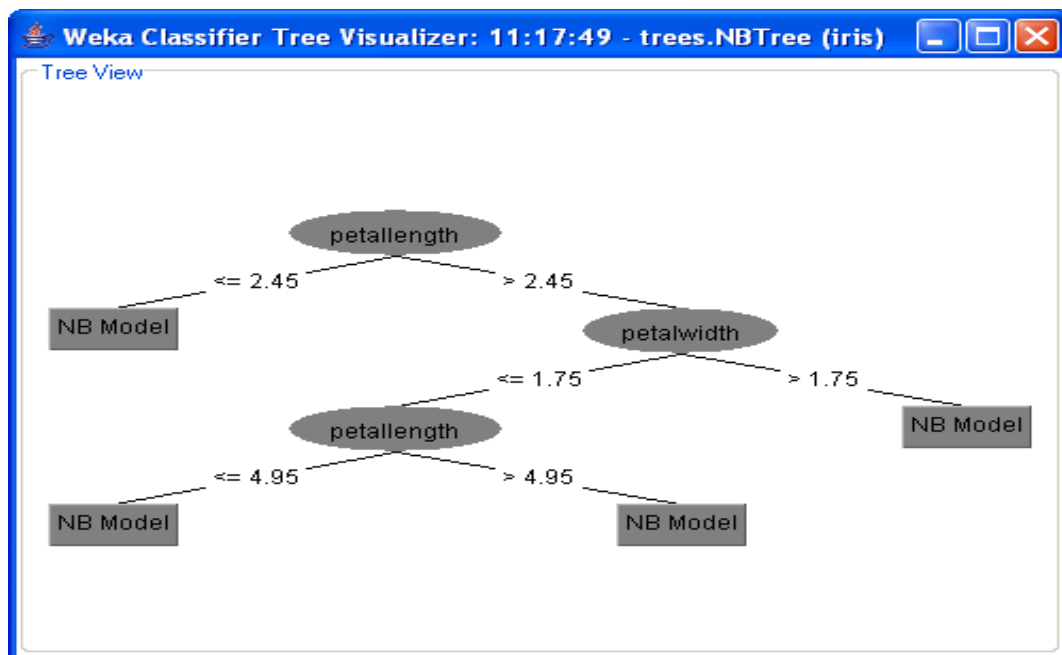
 a b c <-- classified as
50 0 0 | a = Iris-setosa
 0 46 4 | b = Iris-versicolor
 0 5 45 | c = Iris-virginica
    
```

Status
OK

Log x 0



VALIDATION :

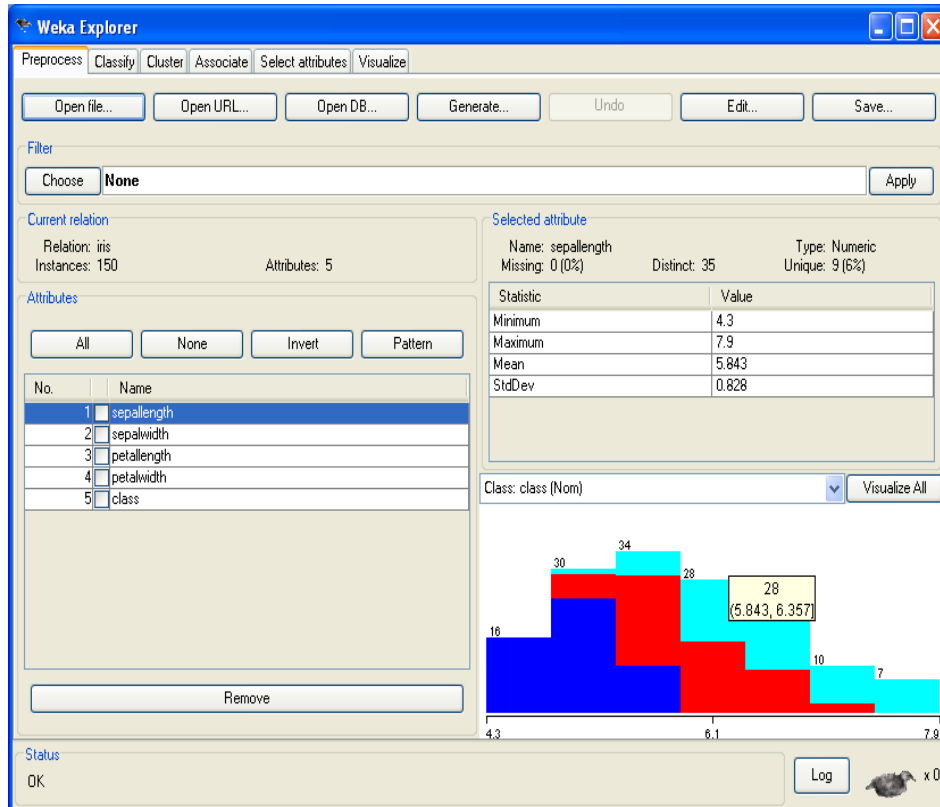


PROGRAM 8.

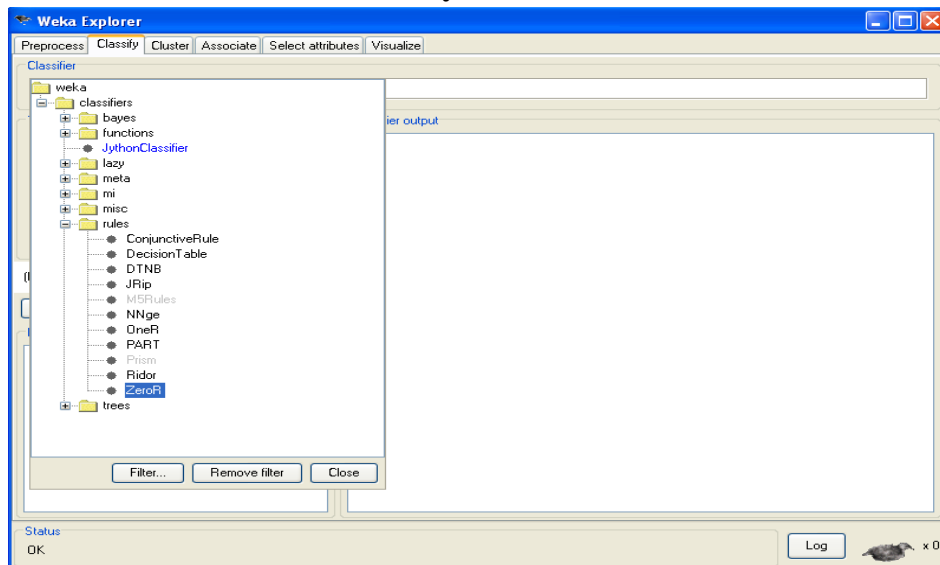
Aim: Obtain the association rules using WEKA

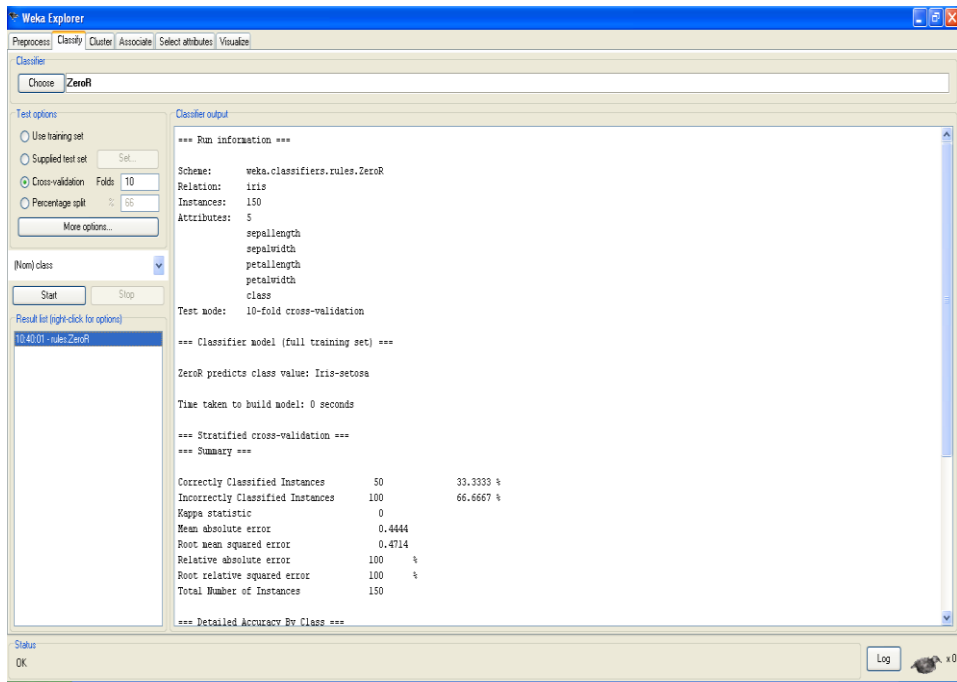
Choose a data set

Iris data set

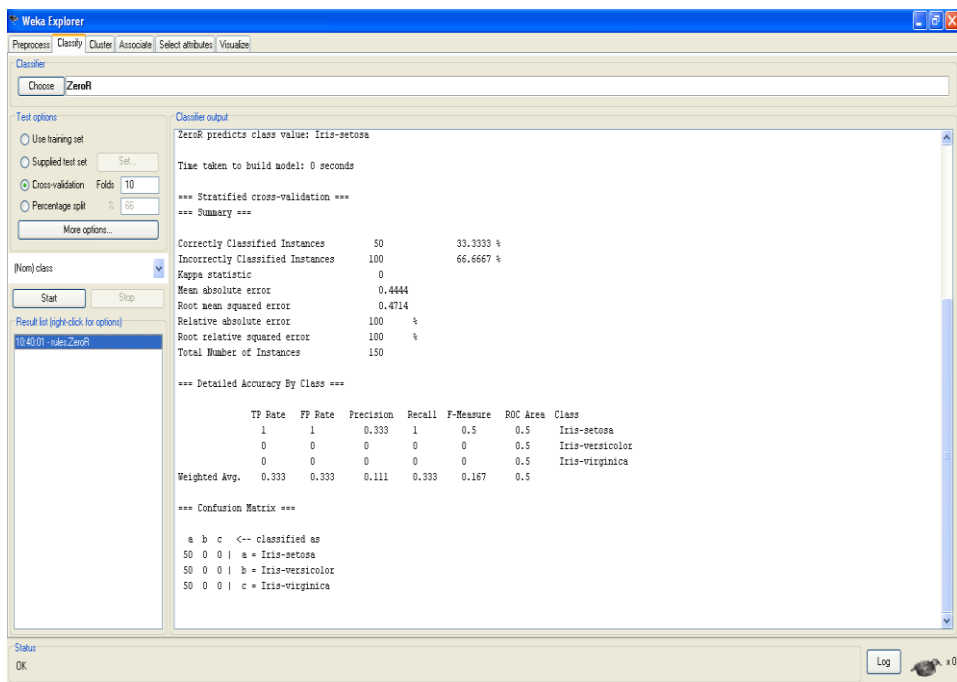


Click on classify and choose ZeroR





VALIDATION :



PROGRAM 9.

Aim: Perform Data Transformations using an ETL Tool.

Description: Overview of ETL in Data Warehouses

You need to load your data warehouse regularly so that it can serve its purpose of facilitating business analysis. To do this, data from one or more operational systems needs to be extracted and copied into the data warehouse. The challenge in data warehouse environments is to integrate, rearrange and consolidate large volumes of data over many systems, thereby providing a new unified information base for business intelligence.

The process of extracting data from source systems and bringing it into the data warehouse is commonly called **ETL**, which stands for extraction, transformation, and loading. Note that ETL refers to a broad process, and not three well-defined steps. The acronym ETL is perhaps too simplistic, because it omits the transportation phase and implies that each of the other phases of the process is distinct. Nevertheless, the entire process is known as ETL.

The methodology and tasks of ETL have been well known for many years, and are not necessarily unique to data warehouse environments: a wide variety of proprietary applications and database systems are the IT backbone of any enterprise. Data has to be shared between applications or systems, trying to integrate them, giving at least two applications the same picture of the world. This data sharing was mostly addressed by mechanisms similar to what we now call ETL.

ETL Tools for Data Warehouses

Designing and maintaining the ETL process is often considered one of the most difficult and resource-intensive portions of a data warehouse project. Many data warehousing projects use ETL tools to manage this process. Oracle Warehouse Builder, for example, provides ETL capabilities and takes advantage of inherent database abilities. Other data warehouse builders create their own ETL tools and processes, either inside or outside the database.

Besides the support of extraction, transformation, and loading, there are some other tasks that are important for a successful ETL implementation as part of the daily operations of the data warehouse and its support for further enhancements. Besides the support for designing a data warehouse and the data flow, these tasks are typically addressed by ETL tools such as Oracle Warehouse Builder.

Oracle is not an ETL tool and does not provide a complete solution for ETL. However, Oracle does provide a rich set of capabilities that can be used by both ETL tools and customized ETL solutions. Oracle offers techniques for transporting data between Oracle databases, for transforming large volumes of data, and for quickly loading new data into a data warehouse.

Introduction to Extraction Methods in Data Warehouses

The extraction method you should choose is highly dependent on the source system and also from the business needs in the target data warehouse environment. Very often, there is no possibility to add additional logic to the source systems to enhance an incremental extraction of data due to the performance or the increased workload of these systems. Sometimes even the customer is not allowed to add anything to an out-of-the-box application system.

Logical Extraction Methods

There are two types of logical extraction:

- Full Extraction
- Incremental Extraction

Full Extraction

The data is extracted completely from the source system. Because this extraction reflects all the data currently available on the source system, there's no need to keep track of changes to the data source since the last successful extraction. The source data will be provided as-is and no additional logical information (for example, timestamps) is necessary on the source site. An example for a full extraction may be an export file of a distinct table or a remote SQL statement scanning the complete source table.

Incremental Extraction

At a specific point in time, only the data that has changed since a well-defined event back in history will be extracted. This event may be the last time of extraction or a more complex business event like the last booking day of a fiscal period. To identify this delta change there must be a possibility to identify all the changed information since this specific time event. This information can be either provided by the source data itself such as an application column, reflecting the last-changed timestamp or a change table where an appropriate additional mechanism keeps track of the changes besides the originating transactions. In most cases, using the latter method means adding extraction logic to the source system.

Many data warehouses do not use any change-capture techniques as part of the extraction process. Instead, entire tables from the source systems are extracted to the data warehouse or staging area, and these tables are compared with a previous extract from the source system to identify the changed data. This approach may not have significant impact on the source systems, but it clearly can place a considerable burden on the data warehouse processes, particularly if the data volumes are large.

Physical Extraction Methods

Depending on the chosen logical extraction method and the capabilities and restrictions on the source side, the extracted data can be physically extracted by two mechanisms. The data can either be extracted online from the source system or from an offline structure. Such an offline structure might already exist or it might be generated by an extraction routine.

There are the following methods of physical extraction:

- Online Extraction
- Offline Extraction

Online Extraction

The data is extracted directly from the source system itself. The extraction process can connect directly to the source system to access the source tables themselves or to an intermediate system

that stores the data in a preconfigured manner (for example, snapshot logs or change tables). Note that the intermediate system is not necessarily physically different from the source system.

With online extractions, you need to consider whether the distributed transactions are using original source objects or prepared source objects.

Offline Extraction

The data is not extracted directly from the source system but is staged explicitly outside the original source system. The data already has an existing structure (for example, redo logs, archive logs or transportable tablespaces) or was created by an extraction routine.

You should consider the following structures:

- Flat files

Data in a defined, generic format. Additional information about the source object is necessary for further processing.

- Dump files

Oracle-specific format. Information about the containing objects may or may not be included, depending on the chosen utility.

- Redo and archive logs

Information is in a special, additional dump file.

- Transportable tablespaces

*Extracting into Flat Files Using SQL*Plus*

The most basic technique for extracting data is to execute a SQL query in SQL*Plus and direct the output of the query to a file. For example, to extract a flat file, `country_city.log`, with the pipe sign as delimiter between column values, containing a list of the cities in the US in the tables `countries` and `customers`, the following SQL script could be run:

```
SET echo off SET pagesize 0 SPOOL country_city.log
SELECT distinct t1.country_name ||'|'|| t2.cust_city
FROM countries t1, customers t2 WHERE t1.country_id = t2.country_id
AND t1.country_name= 'United States of America';
SPOOL off
```

The exact format of the output file can be specified using SQL*Plus system variables.

This extraction technique offers the advantage of storing the result in a customized format. Note that, using the external table data pump unload facility, you can also extract the result of an arbitrary SQL operation. The example previously extracts the results of a join.

This extraction technique can be parallelized by initiating multiple, concurrent SQL*Plus sessions, each session running a separate query representing a different portion of the data to be extracted. For example, suppose that you wish to extract data from an `orders` table, and that the `orders` table has been range partitioned by month, with partitions `orders_jan1998`, `orders_feb1998`, and so on. To extract a single year of data from the `orders` table, you could initiate 12 concurrent SQL*Plus sessions, each extracting a single partition. The SQL script for one such session could be:

```
SPOOL order_jan.dat  
  
SELECT * FROM orders PARTITION (orders_jan1998);  
  
SPOOL OFF
```

Even if the `orders` table is not partitioned, it is still possible to parallelize the extraction either based on logical or physical criteria. The logical method is based on logical ranges of column values, for example:

```
SELECT ... WHERE order_date  
BETWEEN TO_DATE('01-JAN-99') AND TO_DATE('31-JAN-99');
```

The physical method is based on a range of values. By viewing the data dictionary, it is possible to identify the Oracle Database data blocks that make up the `orders` table. Using this information, you could then derive a set of rowid-range queries for extracting data from the `orders` table:

```
SELECT * FROM orders WHERE rowid BETWEEN value1 and value2;
```

Parallelizing the extraction of complex SQL queries is sometimes possible, although the process of breaking a single complex query into multiple components can be challenging. In particular, the coordination of independent processes to guarantee a globally consistent view can be difficult. Unlike the SQL*Plus approach, using the external table data pump unload functionality provides transparent parallel capabilities.

Note that all parallel techniques can use considerably more CPU and I/O resources on the source system, and the impact on the source system should be evaluated before parallelizing any extraction technique.

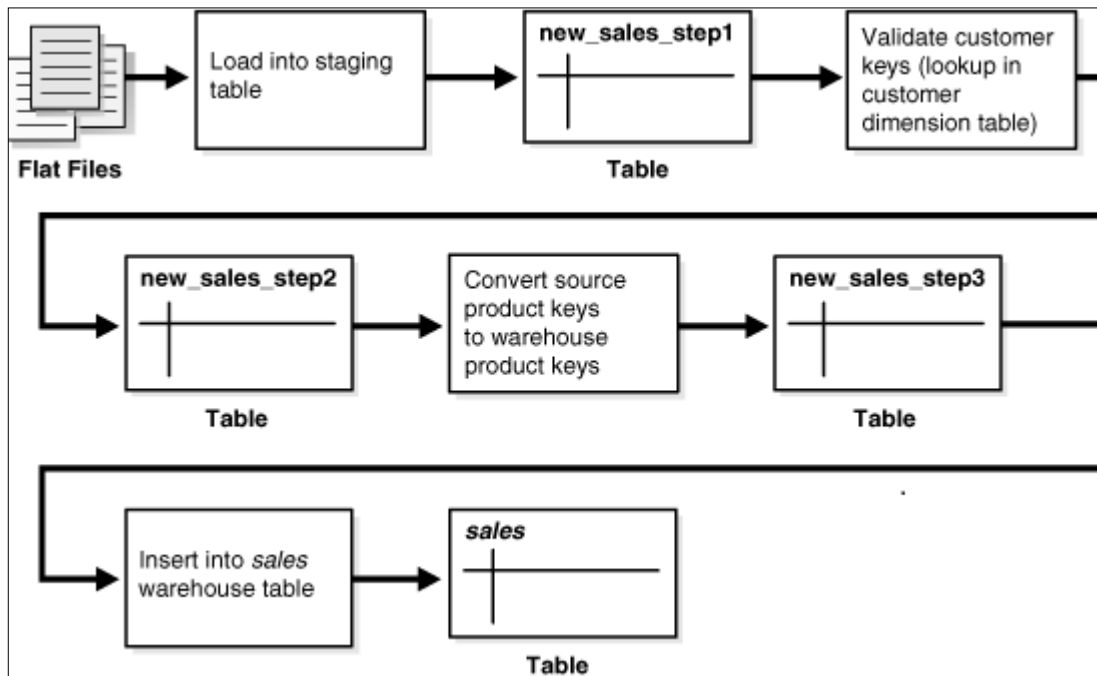
Transformation Flow

From an architectural perspective, you can transform your data in two ways:

- Multistage Data Transformation
- Pipelined Data Transformation

Multistage Data Transformation

The data transformation logic for most data warehouses consists of multiple steps. For example, in transforming new records to be inserted into a sales table, there may be separate logical transformation steps to validate each dimension key.



When using Oracle Database as a transformation engine, a common strategy is to implement each transformation as a separate SQL operation and to create a separate, temporary staging table (such as the tables `new_sales_step1` and `new_sales_step2` to store the incremental results for each step. This load-then-transform strategy also provides a natural checkpointing scheme to the entire transformation process, which enables the process to be more easily monitored and restarted. However, a disadvantage to multistaging is that the space and time requirements increase.

It may also be possible to combine many simple logical transformations into a single SQL statement or single PL/SQL procedure. Doing so may provide better performance than performing each step independently, but it

Loading a Data Warehouse with SQL*Loader

SQL*Loader is used to move data from flat files into an Oracle data warehouse. During this data load, SQL*Loader can also be used to implement basic data transformations. When using direct-path SQL*Loader, basic data manipulation, such as datatype conversion and simple NULL handling, can be automatically resolved during the data load. Most data warehouses use direct-path loading for performance reasons.

The conventional-path loader provides broader capabilities for data transformation than a direct-path loader: SQL functions can be applied to any column as those values are being loaded. This provides a rich capability for transformations during the data load. However, the conventional-path loader is slower than direct-path loader. For these reasons, the conventional-path loader should be considered primarily for loading and transforming smaller amounts of data.

The following is a simple example of a SQL*Loader controlfile to load data into the sales table of the sh sample schema from an external file sh_sales.dat. The external flat file sh_sales.dat consists of sales transaction data, aggregated on a daily level. Not all columns of this external file are loaded into sales. This external file will also be used as source for loading the second fact table of the sh sample schema, which is done using an external table:

The following shows the control file (sh_sales.ctl) loading the sales table:

```
LOAD DATA INFILE sh_sales.dat APPEND INTO TABLE sales
FIELDS TERMINATED BY "|"
(PROD_ID, CUST_ID, TIME_ID, CHANNEL_ID, PROMO_ID, QUANTITY_SOLD,
AMOUNT_SOLD)
```

It can be loaded with the following command:

```
$ sqlldr control=sh_sales.ctl direct=true
```

Username:

Password:

Viva Voce

- 1) Define ETL
- 2) Define Data Warehouse
- 3) Define Staging Area
- 4) Define tablespace
- 5) What is SQL * Loader

PROGRAM 10.

Aim: A small case study on involving all stages of KDD

Description:

KDD Process is the process of using data mining methods (algorithms) to extract (identify) what is deemed knowledge according to the specifications of measures and thresholds, using database F along with any required preprocessing, subsampling, and transformation of F.”

KDD:

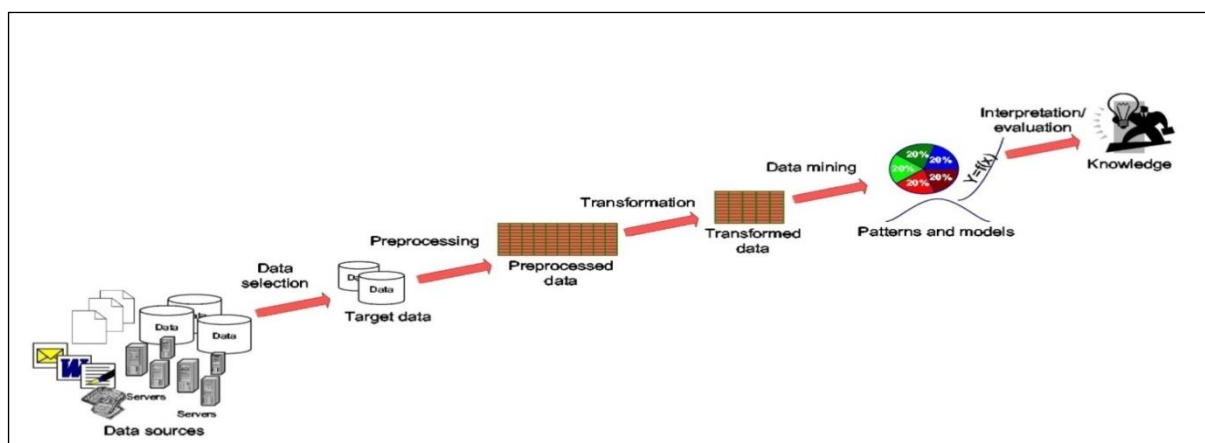
- In a multistep process many decisions are made by the user (domain expert):
- Iterative and interactive – loops between any two steps are possible
- Usually the most focus is on the DM step, but other steps are of considerable importance for the successful application of KDD in practice

GOALS:

- Verification of user’s hypothesis (this against the EDA principle...)
- Autonomous discovery of new patterns and models
- Prediction of future behavior of some entities
- Description of interesting patterns and models

STEPS OF DM:

1. Domain understanding and goal setting
2. Creating a target data set
3. Data cleaning and preprocessing
4. Data reduction and projection
5. Data mining
 - Choosing the data mining task
 - Choosing the data mining algorithm(s)
 - Use of data mining algorithms
6. Interpretation of mined patterns
7. Utilization of discovered knowledge



1) Domain analysis

- Development of domain understanding
- Discovery of relevant prior knowledge
- Definition of the goal of the knowledge discovery

2) Data selection

- Selection and integration of the target data from possibly many different and heterogeneous sources
- Interesting data may exist, e.g., in relational databases, document collections, e-mails, photographs, video clips, process database, customer transaction database, web logs etc.
- Focus on the correct subset of variables and data samples
 - E.g., customer behavior in a certain country, relationship between items purchased and customer income and age

3) Data cleaning and preprocessing

- Dirty data can confuse the mining procedures and lead to unreliable and invalid outputs
- Complex analysis and mining on a huge amount of data may take a very long time
- Preprocessing and cleaning should improve the quality of data and mining results by enhancing the actual mining process
- The actions to be taken includes
 - Removal of noise or outliers
 - Collecting necessary information to model or account for noise
 - Using prior domain knowledge to remove the inconsistencies and duplicates from the data
 - Choice or usage of strategies for handling missing data fields

4) Data reduction and projection

- Data transformation techniques
 - Smoothing (binning, clustering, regression etc.)
 - Aggregation (use of summary operations (e.g., averaging) on data)
 - Generalization (primitive data objects can be replaced by higher-level concepts)
 - Normalization (min-max-scaling, z-score)
 - Feature construction from the existing attributes (PCA, MDS)
- Data reduction techniques are applied to produce reduced representation of the data (smaller volume that closely maintains the integrity of the original data)
- Aggregation
 - Dimension reduction (Attribute subset selection, PCA, MDS,...)
 - Compression (e.g., wavelets, PCA, clustering,...)
 - Numerosity reduction
 - parametric models: regression and log-linear models
 - non-parametric models: histograms, clustering, sampling...
 - Discretization (e.g., binning, histograms, cluster analysis,...)
 - Concept hierarchy generation (numeric value of "age" to a higher level concept "young, middle-aged, senior")

5) Choice of data mining task

- Define the task for data mining
 - Exploration/summarization
 - Summarizing statistics (mean, median, mode, std,...)
 - Class/concept description
 - Explorative data analysis
 - Graphical techniques, low-dimensional plots,...
 - Predictive
 - Classification or regression
 - Descriptive
 - Cluster analysis, dependency modelling, change and outlier detection

6) Choosing the DM algorithm(s)

- Select the most appropriate methods to be used for the model and pattern search
- Matching the chosen method with the overall goal of the KDD process (necessitates communication between the end user and method specialists)
- Note that this step requires understanding in many fields, such as computer science, statistics, machine learning, optimization, etc.

7) Use of data mining algorithms

- Application of the chosen DM algorithms to the target data set
- Search for the patterns and models of interest in a particular representational form or a set of such representations
- Classification rules or trees, regression models, clusters, mixture models...
- Should be relatively automatic
- Generally DM involves:
 - Establish the structural form (model/pattern) one is interested
 - Estimate the parameters from the available data
 - Interpret the fitted models

8) Interpretation/evaluation

- The mined patterns and models are interpreted
- The results should be presented in understandable form
- Visualization techniques are important for making the results useful – mathematical models or text type descriptions may be difficult for domain experts
- Possible return to any of the previous step

ADDITIONAL PROGRAMS

Introduction and Basics of R Programming:

How to install R Packages?

The sheer power of R lies in its incredible packages. In R, most data handling tasks can be performed in 2 ways: Using R packages and R base functions. In this tutorial, I'll also introduce you with the most handy and powerful R packages. To install a package, simply type:

```
install.packages("package name")
```

As a first time user, a pop might appear to select your CRAN mirror (country server), choose accordingly and press OK.

Note: You can type this either in console directly and press 'Enter' or in R script and click 'Run'.

Basic Computations in R

Let's begin with basics. To get familiar with R coding environment, start with some basic calculations. R console can be used as an interactive calculator too. Type the following in your console:

```
> 2 + 3  
> 5
```

```
> 6 / 3  
> 2
```

```
> (3*8)/(2*3)  
> 4
```

```
> log(12)  
> 1.07
```

```
> sqrt(121)  
> 11
```

Similarly, you can experiment various combinations of calculations and get the results. In case, you want to obtain the previous calculation, this can be done in two ways. First, click in R console, and press 'Up / Down Arrow' key on your keyboard. This will activate the previously executed commands. Press Enter.

But, what if you have done too many calculations ? It would be too painful to scroll through every command and find it out. In such situations, creating variable is a helpful way.

In R, you can create a variable using <- or = sign. Let's say I want to create a variable x to compute the sum of 7 and 8. I'll write it as:

```
> x <- 8 + 7
> x
> 15
```

Once we create a variable, you no longer get the output directly (like calculator), unless you call the variable in the next line. Remember, variables can be alphabets, alphanumeric but not numeric. You can't create numeric variables.

2. Essentials of R Programming

Understand and practice this section thoroughly. This is the building block of your R programming knowledge. If you get this right, you would face less trouble in debugging.

R has five basic or 'atomic' classes of objects. Wait, what is an object ?

Everything you see or create in R is an object. A vector, matrix, data frame, even a variable is an object. R treats it that way. So, R has 5 basic classes of objects. This includes:

1. Character
2. Numeric (Real Numbers)
3. Integer (Whole Numbers)
4. Complex
5. Logical (True / False)

Since these classes are self-explanatory by names, I wouldn't elaborate on that. These classes have attributes. Think of attributes as their 'identifier', a name or number which aptly identifies them. An object can have following attributes:

1. names, dimension names
2. dimensions
3. class
4. length

Attributes of an object can be accessed using *attributes()* function. More on this coming in following section.

Let's understand the concept of object and attributes practically. The most basic object in R is known as vector. You can create an empty vector using *vector()*. Remember, a vector contains object of same class.

For example: Let's create vectors of different classes. We can create vector using *c()* or concatenate command also.

```
> a <- c(1.8, 4.5) #numeric
> b <- c(1 + 2i, 3 - 6i) #complex
> d <- c(23, 44) #integer
> e <- vector("logical", length = 5)
```

Similarly, you can create vector of various classes.

Data Types in R

R has various type of ‘data types’ which includes vector (numeric, integer etc), matrices, data frames and list. Let’s understand them one by one.

Vector: As mentioned above, a vector contains object of same class. But, you can mix objects of different classes too. When objects of different classes are mixed in a list, coercion occurs. This effect causes the objects of different types to ‘convert’ into one class. For example:

```
> qt <- c("Time", 24, "October", TRUE, 3.33) #character
> ab <- c(TRUE, 24) #numeric
> cd <- c(2.5, "May") #character
```

To check the class of any object, use `class("vector name")` function.

```
> class(qt)
"character"
```

To convert the class of a vector, you can use `as.` command.

```
> bar <- 0:5
> class(bar)
"integer"
> as.numeric(bar)
> class(bar)
"numeric"
> as.character(bar)
> class(bar)
"character"
```

Similarly, you can change the class of any vector. But, you should pay attention here. If you try to convert a “character” vector to “numeric”, NAs will be introduced. Hence, you should be careful to use this command.

List: A list is a special type of vector which contain elements of different data types. For example:

```
> my_list <- list(22, "ab", TRUE, 1 + 2i)
> my_list
[[1]]
[1] 22
[[2]]
[1] "ab"
[[3]]
[1] TRUE
[[4]]
[1] 1+2i
```

As you can see, the output of a list is different from a vector. This is because, all the objects are of different types. The double bracket `[[1]]` shows the index of first element and so on. Hence, you can easily extract the element of lists depending on their index. Like this:

```
> my_list[[3]]
> [1] TRUE
```

You can use [] single bracket too. But, that would return the list element with its index number, instead of the result above. Like this:

```
> my_list[3]
> [[1]]
[1] TRUE
```

Matrices: When a vector is introduced with *row* and *column* i.e. a dimension attribute, it becomes a matrix. A matrix is represented by set of rows and columns. It is a 2 dimensional data structure. It consist of elements of same class. Let's create a matrix of 3 rows and 2 columns:

```
> my_matrix <- matrix(1:6, nrow=3, ncol=2)
> my_matrix
[,1] [,2]
[1,] 1 4
[2,] 2 5
[3,] 3 6
> dim(my_matrix)
[1] 3 2
> attributes(my_matrix)
$dim
[1] 3 2
```

As you can see, the dimensions of a matrix can be obtained using either *dim()* or *attributes()* command. To extract a particular element from a matrix, simply use the index shown above. For example(try this at your end):

```
> my_matrix[,2] #extracts second column
> my_matrix[,1] #extracts first column
> my_matrix[2,] #extracts second row
> my_matrix[1,] #extracts first row
```

As an interesting fact, you can also create a matrix from a vector. All you need to do is, assign dimension *dim()* later. Like this:

```
> age <- c(23, 44, 15, 12, 31, 16)
> age
[1] 23 44 15 12 31 16

> dim(age) <- c(2,3)
> age
[,1] [,2] [,3]
```

```
[1,] 23 15 31
[2,] 44 12 16
> class(age)
[1] "matrix"
```

You can also join two vectors using *cbind()* and *rbind()* functions. But, make sure that both vectors have same number of elements. If not, it will return NA values.

```
> x <- c(1, 2, 3, 4, 5, 6)
> y <- c(20, 30, 40, 50, 60)
> cbind(x, y)
> cbind(x, y)
x y
[1,] 1 20
[2,] 2 30
[3,] 3 40
[4,] 4 50
[5,] 5 60
[6,] 6 70
> class(cbind(x, y))
[1] "matrix"
```

Data Frame: This is the most commonly used member of data types family. It is used to store tabular data. It is different from matrix. In a matrix, every element must have same class. But, in a data frame, you can put list of vectors containing different classes. This means, every column of a data frame acts like a list. Every time you will read data in R, it will be stored in the form of a data frame. Hence, it is important to understand the majorly used commands on data frame:

```
> df <- data.frame(name = c("ash", "jane", "paul", "mark"), score = c(67, 56, 87, 91))
> df
name score
1 ash 67
2 jane 56
3 paul 87
4 mark 91

> dim(df)
[1] 4 2
```

```
> str(df)
'data.frame': 4 obs. of 2 variables:
 $ name : Factor w/ 4 levels "ash","jane","mark",...: 1 2 4 3
 $ score: num 67 56 87 91
> nrow(df)
[1] 4
> ncol(df)
[1] 2
```

Let's understand the code above. *df* is the name of data frame. *dim()* returns the dimension of data frame as 4 rows and 2 columns. *str()* returns the structure of a data frame i.e. the list of variables stored in the data frame. *nrow()* and *ncol()* return the number of rows and number of columns in a data set respectively.

Here you see “name” is a factor variable and “score” is numeric. In data science, a variable can be categorized into two types: Continuous and Categorical.

Continuous variables are those which can take any form such as 1, 2, 3.5, 4.66 etc. **Categorical variables** are those which takes only discrete values such as 2, 5, 11, 15 etc. In R, categorical values are represented by factors. In *df*, name is a factor variable having 4 unique levels. Factor or categorical variable are specially treated in a data set. Similarly, you can find techniques to deal with continuous variables here.

Let's now understand the concept of **missing values** in R. This is one of the most painful yet crucial part of predictive modelling. You must be aware of all techniques to deal with them. The complete explanation on such techniques is provided here.

Missing values in R are represented by *NA* and *NaN*. Now we'll check if a data set has missing values (using the same data frame *df*).

```
> df[1:2,2] <- NA #injecting NA at 1st, 2nd row and 2nd column of df
> df
  name score
1 ash NA
2 jane NA
3 paul 87
4 mark 91
> is.na(df) #checks the entire data set for NAs and return logical output
  name score
[1,] FALSE TRUE
[2,] FALSE TRUE
```

```
[3,] FALSE FALSE
[4,] FALSE FALSE
> table(is.na(df)) #returns a table of logical output
FALSE TRUE
6 2
> df[!complete.cases(df),] #returns the list of rows having missing values
name score
1 ash NA
2 jane NA
```

Missing values hinder normal calculations in a data set. For example, let's say, we want to compute the mean of score. Since there are two missing values, it can't be done directly. Let's see:

```
mean(df$score)
[1] NA
> mean(df$score, na.rm = TRUE)
[1] 89
```

The use of *na.rm = TRUE* parameter tells R to ignore the NAs and compute the mean of remaining values in the selected column (score). To remove rows with NA values in a data frame, you can use *na.omit*:

```
> new_df <- na.omit(df)
> new_df
name score
3 paul 87
4 mark 91
```

Control Structures in R

As the name suggest, a control structure 'controls' the flow of code / commands written inside a function. A function is a set of multiple commands written to automate a repetitive coding task. For example: You have 10 data sets. You want to find the mean of 'Age' column present in every data set. This can be done in 2 ways: either you write the code to compute mean 10 times or you simply create a function and pass the data set to it.

Let's understand the control structures in R with simple examples:

if, else – This structure is used to test a condition. Below is the syntax:

```
if (<condition>){
  ##do something
} else {
```



```
##do something  
}
```

Example

```
#initialize a variable  
N <- 10  
#check if this variable * 5 is > 40  
if (N * 5 > 40){  
  print("This is easy!")  
} else {  
  print ("It's not easy!")  
}  
[1] "This is easy!"
```

for – This structure is used when a loop is to be executed fixed number of times. It is commonly used for iterating over the elements of an object (list, vector). Below is the syntax:

```
for (<search condition>){  
  #do something  
}
```

Example

```
#initialize a vector  
y <- c(99,45,34,65,76,23)  
#print the first 4 numbers of this vector  
for(i in 1:4){  
  print (y[i])  
}  
[1] 99  
[1] 45  
[1] 34  
[1] 65
```

while – It begins by testing a condition, and executes only if the condition is found to be true. Once the loop is executed, the condition is tested again. Hence, it's necessary to alter the condition such that the loop doesn't go infinity. Below is the syntax:

```
#initialize a condition  
Age <- 12  
#check if age is less than 17  
while(Age < 17){
```

```
print(Age)
Age <- Age + 1 #Once the loop is executed, this code breaks the loop
}
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
```

There are other control structures as well but are less frequently used than explained above. Those structures are:

1. `repeat` – It executes an infinite loop
2. `break` – It breaks the execution of a loop
3. `next` – It allows to skip an iteration in a loop
4. `return` – It help to exit a function

Note: If you find the section ‘control structures’ difficult to understand, not to worry. R is supported by various packages to compliment the work done by control structures.

Useful R Packages

Out of ~7800 packages listed on [CRAN](#), I’ve listed some of the most powerful and commonly used packages in predictive modeling in this article. Since, I’ve already explained the method of installing packages, you can go ahead and install them now. Sooner or later you’ll need them.

Importing Data: R offers wide range of packages for importing data available in any format such as `.txt`, `.csv`, `.json`, `.sql` etc. To import large files of data quickly, it is advisable to install and use `data.table`, `readr`, `RMySQL`, `sqldf`, `jsonlite`.

Data Visualization: R has in built plotting commands as well. They are good to create simple graphs. But, becomes complex when it comes to creating advanced graphics. Hence, you should install `ggplot2`.

Data Manipulation: R has a fantastic collection of packages for data manipulation. These packages allows you to do basic & advanced computations quickly. These packages are `dplyr`, `plyr`, `tidyr`, `lubridate`, `stringr`. **Modeling / Machine Learning:** For modeling, `caret` package in R is powerful enough to cater to every need for creating machine learning model. However, you can install packages algorithms wise such as `randomForest`, `rpart`, `gbm` etc

PROGRAM 11.

Aim : Data Manipulation in R.

Description:

Let's call it as, the advanced level of data exploration. In this section we'll practically learn about feature engineering and other useful aspects.

Feature Engineering: This component separates an intelligent data scientist from a technically enabled data scientist. You might have access to large machines to run heavy computations and algorithms, but the power delivered by new features, just can't be matched. We create new variables to extract and provide as much 'new' information to the model, to help it make accurate predictions.

If you have been thinking all this time, great. But now is the time to think deeper. Look at the data set and ask yourself, what else (factor) could influence `Item_Outlet_Sales` ? Anyhow, the answer is below.

1. **Count of Outlet Identifiers** – There are 10 unique outlets in this data. This variable will give us information on count of outlets in the data set. More the number of counts of an outlet, chances are more will be the sales contributed by it.

```
> library(dplyr)
> a <- combi %>%
  group_by(Outlet_Identifier) %>%
  tally()

> head(a)
Source: local data frame [6 x 2]
Outlet_Identifier n
(fctr)          (int)
1 OUT010         925
2 OUT013        1553
3 OUT017        1543
4 OUT018        1546
5 OUT019         880
6 OUT027        1559

> names(a)[2] <- "Outlet_Count"
> combi <- full_join(a, combi, by = "Outlet_Identifier")
```

As you can see, `dplyr` package makes data manipulation quite effortless. You no longer need to write long function. In the code above, I've simply stored the new data frame in a variable `a`. Later, the new column `Outlet_Count` is added in our original 'combi' data set..

2. **Count of Item Identifiers** – Similarly, we can compute count of item identifiers too. It's a good practice to fetch more information from unique ID variables using their count. This will help us to understand, which outlet has maximum frequency.

```
> b <- combi%>%
  group_by(Item_Identifier)%>%
  tally()
```

```
> names(b)[2] <- "Item_Count"
> head(b)
Item_Identifier Item_Count
(fctr)          (int)
1 DRA12         9
2 DRA24        10
3 DRA59        10
4 DRB01         8
5 DRB13         9
6 DRB24         8
```

```
> combi <- merge(b, combi, by = "Item_Identifier")
```

3. **Outlet Years** – This variable represent the information of existence of a particular outlet since year 2013. Why just 2013? You'll find the answer in problem statement [here](#). My hypothesis is, older the outlet, more footfall, large base of loyal customers and larger the outlet sales.

```
> c <- combi%>%
  select(Outlet_Establishment_Year)%>%
  mutate(Outlet_Year = 2013 - combi$Outlet_Establishment_Year)
> head(c)
```

```
Outlet_Establishment_Year Outlet_Year
1 1999                    14
2 2009                    4
3 1999                    14
4 1998                    15
5 1987                    26
6 2009                    4
```

```
> combi <- full_join(c, combi)
```

This suggests that outlets established in 1999 were 14 years old in 2013 and so on.

4. **Item Type New** – Now, pay attention to *Item_Identifier*. We are about to discover a new trend. Look carefully, there is a pattern in the identifiers starting with “FD”, “DR”, “NC”. Now, check the corresponding *Item_Types* to these identifiers in the data set. You'll discover, items corresponding to “DR”, are mostly eatables. Items corresponding to “FD”, are drinks. And, item corresponding to “NC”, are products which can't be consumed, let's call them non-consumable. Let's extract these variables into a new variable representing their counts.

Here I'll use *substr()*, *gsub()* function to extract and rename the variables respectively.

```
> q <- substr(combi$Item_Identifier,1,2)
> q <- gsub("FD", "Food", q)
> q <- gsub("DR", "Drinks", q)
> q <- gsub("NC", "Non-Consumable", q)
```

```
> table(q)
Drinks Food Non-Consumable
1317  10201 2686
```

Let's now add this information in our data set with a variable name 'Item_Type_New'.

```
> combi$Item_Type_New <- q
```

I'll leave the rest of feature engineering intuition to you. You can think of more variables which could add more information to the model. But make sure, the variables aren't correlated. Since, they are emanating from a same set of variables, there is a high chance for them to be correlated. You can check the same in R using `cor()` function.

PROGRAM 12. Classification in R.

Aim: Decision Trees

Description:

Before you start, I'd recommend you to glance through the basics of decision tree algorithms. To understand what makes it superior than linear regression, In R, decision tree algorithm can be implemented using *rpart package*. In addition, we'll use *caret package* for doing cross validation.

Cross validation is a technique to build robust models which are not prone to overfitting.

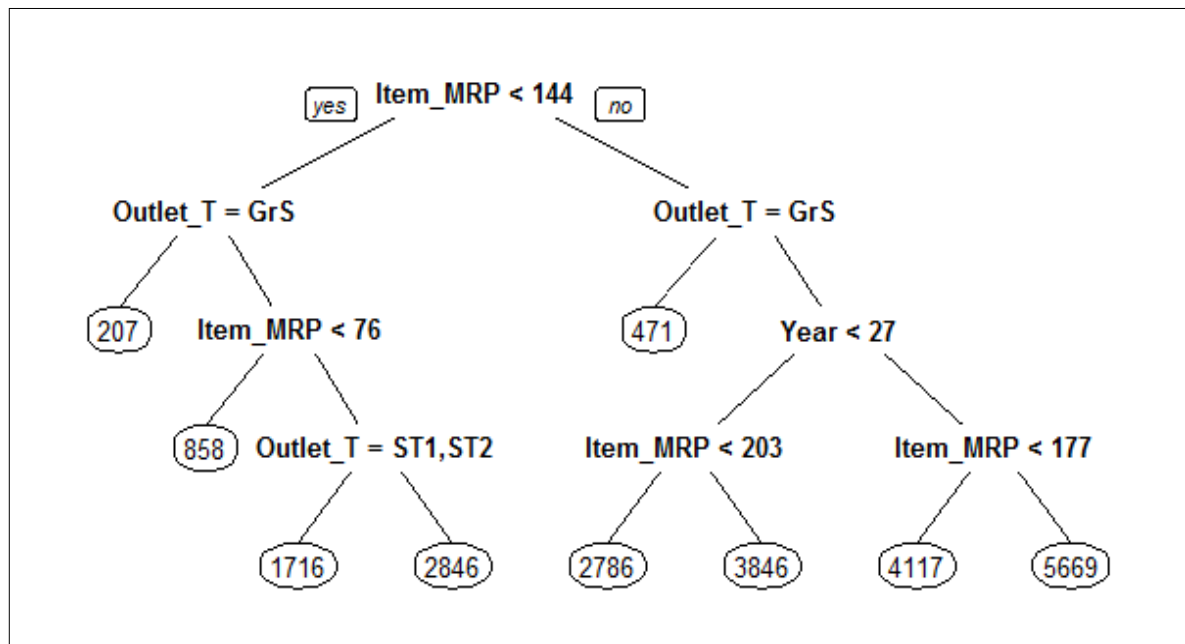
In R, decision tree uses a complexity parameter (*cp*). It measures the tradeoff between model complexity and accuracy on training set. A smaller *cp* will lead to a bigger tree, which might overfit the model. Conversely, a large *cp* value might underfit the model. Underfitting occurs when the model does not capture underlying trends properly. Let's find out the optimum *cp* value for our model with 5 fold cross validation.

Program:

```
#loading required libraries
> library(rpart)
> library(e1071)
> library(rpart.plot)
> library(caret)
#setting the tree control parameters
> fitControl <- trainControl(method = "cv", number = 5)
> cartGrid <- expand.grid(.cp=(1:50)*0.01)
#decision tree
> tree_model <- train(Item_Outlet_Sales ~ ., data = new_train, method = "rpart", trControl =
fitControl, tuneGrid = cartGrid)
> print(tree_model)
```

The final value for *cp* = 0.01. You can also check the table populated in console for more information. The model with *cp* = 0.01 has the least RMSE. Let's now build a decision tree with 0.01 as complexity parameter.

```
> main_tree <- rpart(Item_Outlet_Sales ~ ., data = new_train, control = rpart.control(cp=0.01))
> prp(main_tree)
```



Here is the tree structure of our model. If you have gone through the basics, you would now understand that this algorithm has marked `Item_MRP` as the most important variable (being the root node).

Let's check the RMSE of this model and see if this is any better than regression.

```
> pre_score <- predict(main_tree, type = "vector")
> rmse(new_train$Item_Outlet_Sales, pre_score)
[1] 1102.774
```