Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East – 400098.

A Practical Journal Submitted in fulfillment

of the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

YEAR 2023-24

Part II

Semester-3

Subject code – 90987 R

Subject Name - Business Intelligence and

Big Data Analytics II

BY

Mr. Mohammed Maaz Shaikh

Application ID - 41775

Seat No - 4100058

Institute of Distance and Open Learning
(IDOL)

University of Mumbai

Certificate

This is to certify that **Mr. Mohammed Maaz Shaikh** student of Masters of Computer Science, Part 2, Semester 3 has completed the specified term work in the subject of **Business Intelligence and Big Data Analytics II** in satisfactorily manner within this institute as laid down by University of Mumbai during the academic year 2024 to 2025.

M.Sc. - CS Coordinator   Examiner

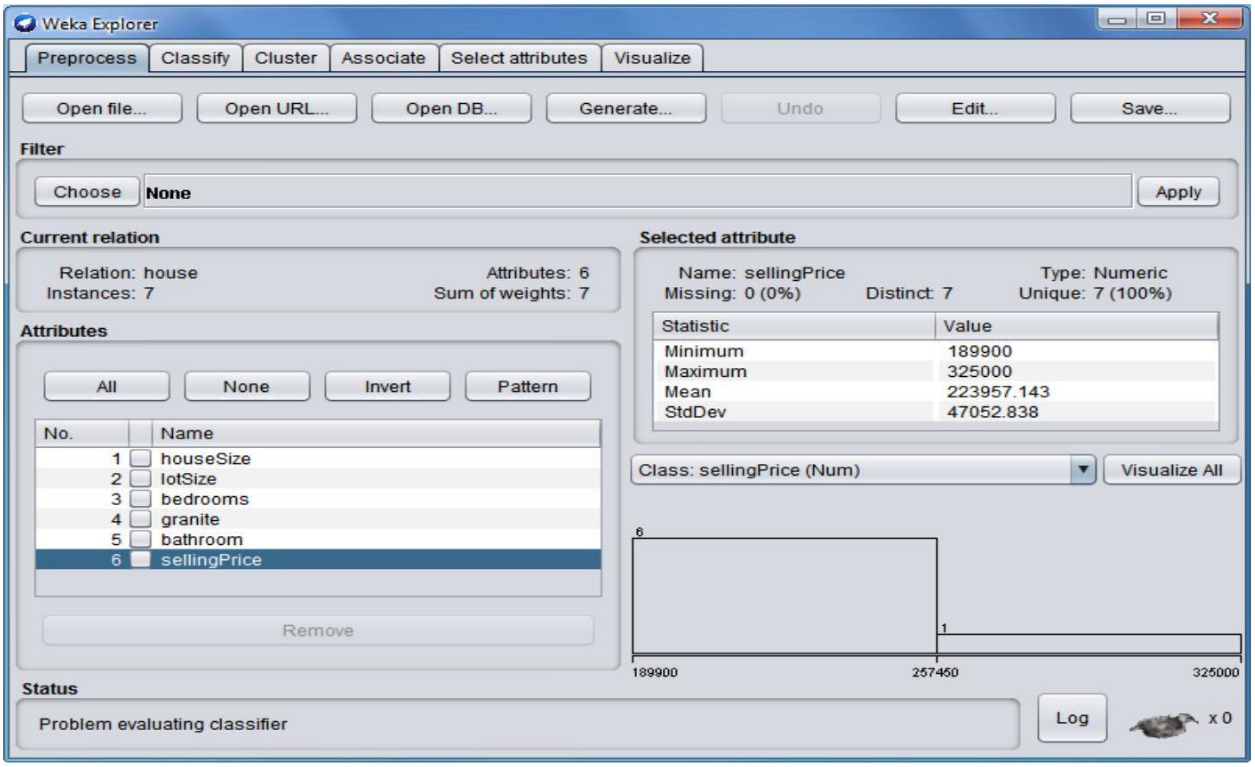
Date: Guide

INDEX

Practical No. 1

Aim : **Generate Regression model and interpret the result for a given data set.**

Step 1: Open Weka then open file h.arff in Weka Explorer.



Filter

None

Current relation

Relation: house
Instances: 7
Attributes: 6
Sum of weights: 7

Attributes

No.		Name
1	□	houseSize
2	□	lotSize
3	□	bedrooms
4	□	granite
5	□	bathroom
6	□	sellingPrice

Selected attribute

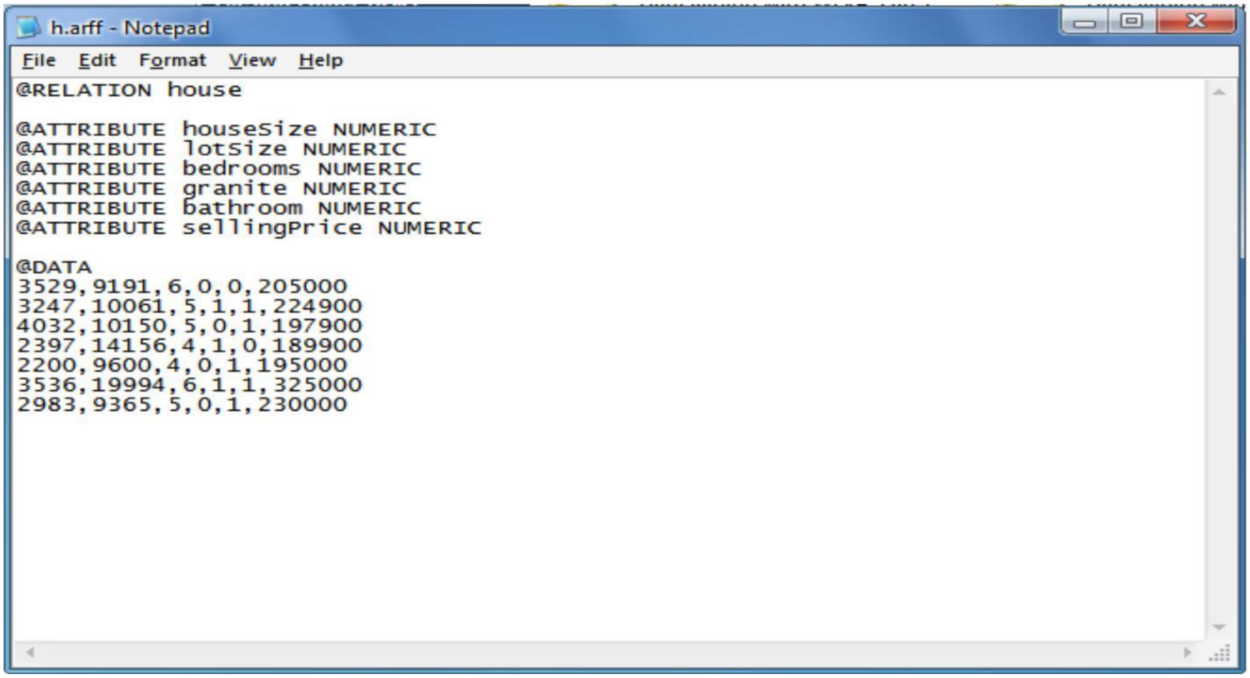
Name: sellingPrice
Type: Numeric
Missing: 0 (0%)
Distinct: 7
Unique: 7 (100%)

Statistic	Value
Minimum	189900
Maximum	325000
Mean	223957.143
StdDev	47052.838

Class: sellingPrice (Num)

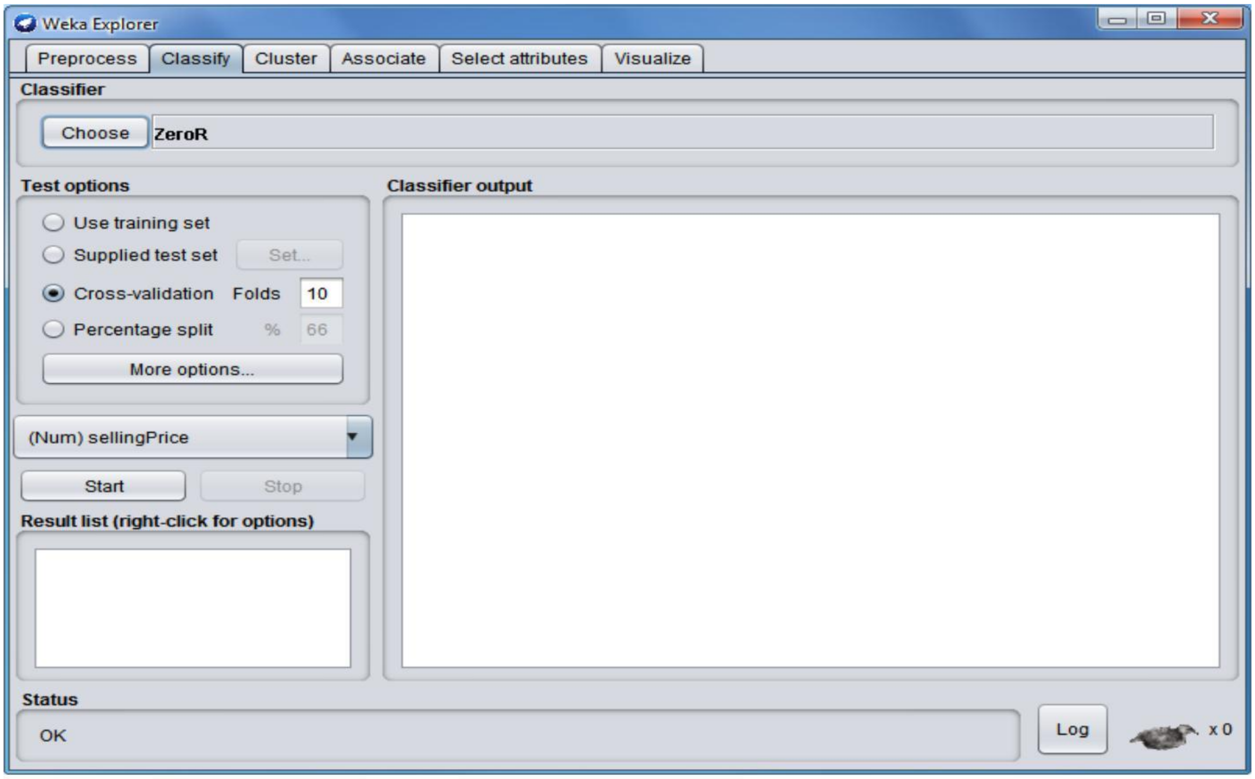
Status

Problem evaluating classifier



```
@RELATION house

@ATTRIBUTE houseSize NUMERIC
@ATTRIBUTE lotSize NUMERIC
@ATTRIBUTE bedrooms NUMERIC
@ATTRIBUTE granite NUMERIC
@ATTRIBUTE bathroom NUMERIC
@ATTRIBUTE sellingPrice NUMERIC

@DATA
3529,9191,6,0,0,205000
3247,10061,5,1,1,224900
4032,10150,5,0,1,197900
2397,14156,4,1,0,189900
2200,9600,4,0,1,195000
3536,19994,6,1,1,325000
2983,9365,5,0,1,230000
```

Step 2: Click on Classify, choose weka classifier function LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4.

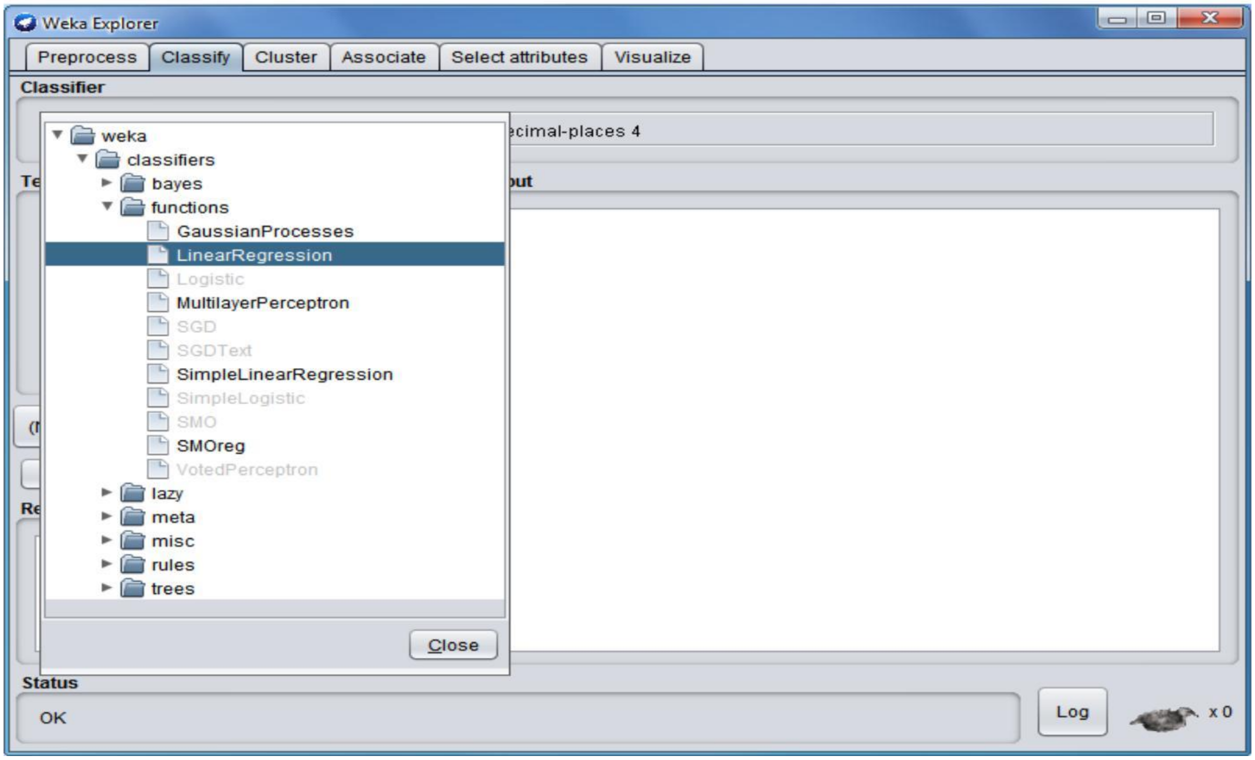


ZeroR

○ Use training set
○ Supplied test set
● Cross-validation Folds 10
○ Percentage split % 66

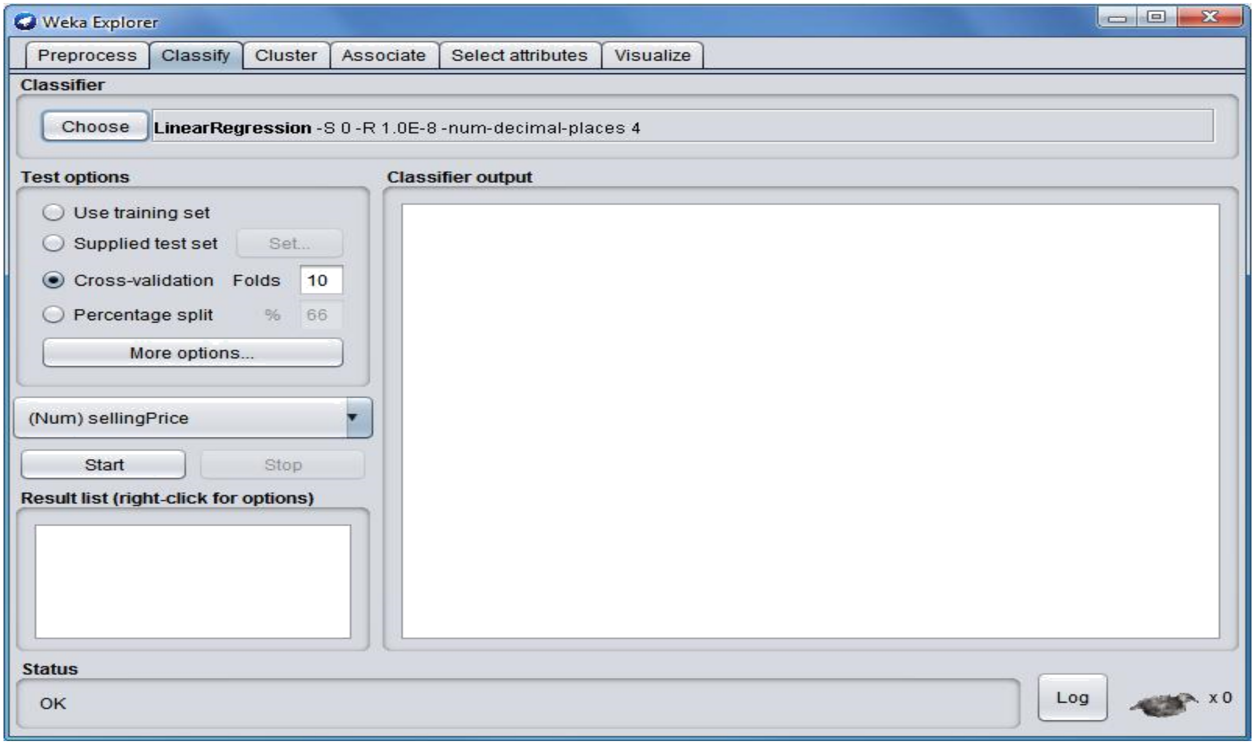
(Num) sellingPrice

Status
OK



ecimal-places 4

Status
OK



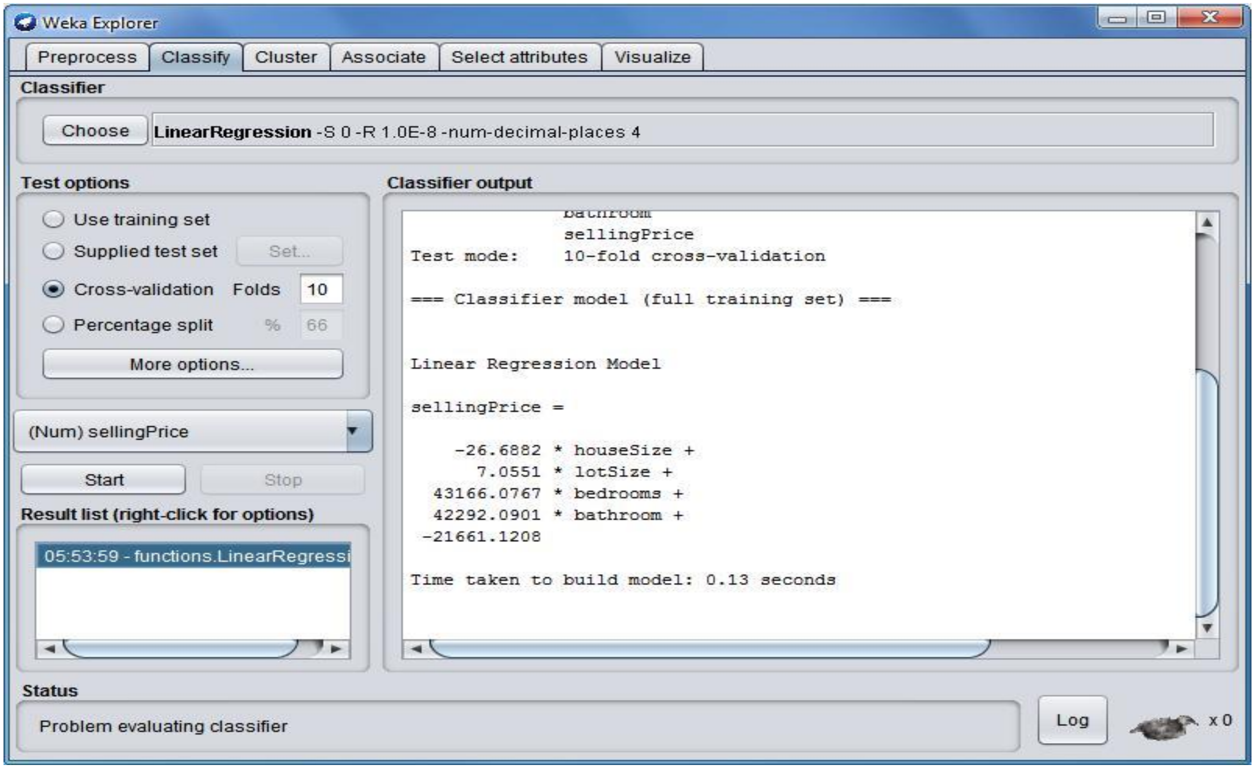
Classifier: LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4

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

(Num) sellingPrice

Status: OK

Step 3: Click on Start. You can see the linear regression on the input file.



Classifier: LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4

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

(Num) sellingPrice

Result list: 05:53:59 - functions.LinearRegressi

Classifier output:

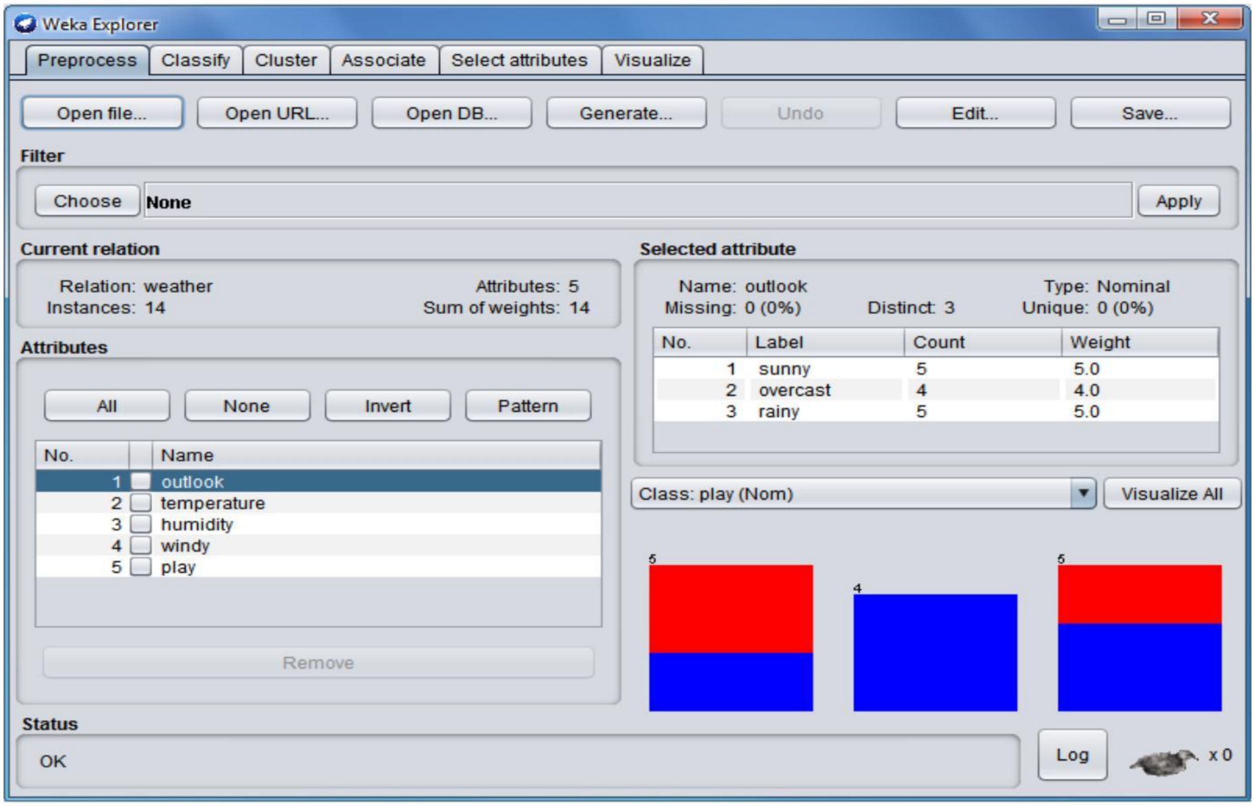
```
                bathroom
                sellingPrice
Test mode:     10-fold cross-validation

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


Linear Regression Model

sellingPrice =

    -26.6882 * houseSize +
      7.0551 * lotSize +
  43166.0767 * bedrooms +
  42292.0901 * bathroom +
 -21661.1208

Time taken to build model: 0.13 seconds
```

Status: Problem evaluating classifier

Practical No. 2

Aim : **Generate forecasting model and interpret the result for a given data set.**

Step 1: Open Weka then open file Weather.arff in Weka Explorer.



Filter: None

Relation: weather
Instances: 14
Attributes: 5
Sum of weights: 14

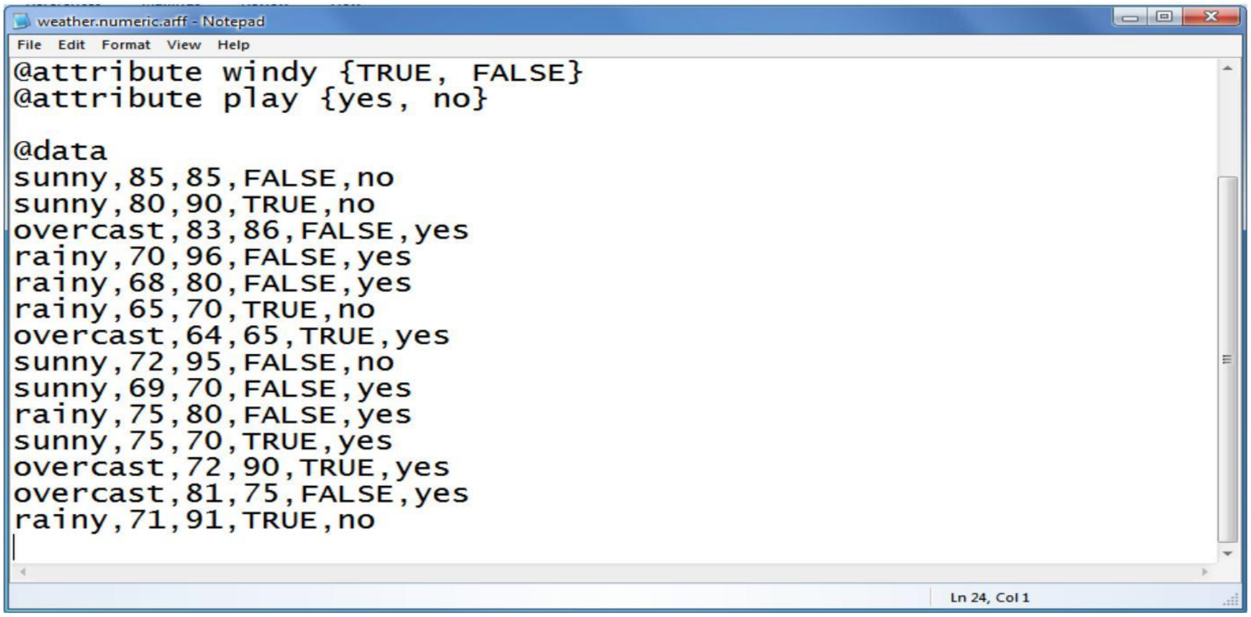
No.	Name
1	outlook
2	temperature
3	humidity
4	windy
5	play

Name: outlook
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	sunny	5	5.0
2	overcast	4	4.0
3	rainy	5	5.0

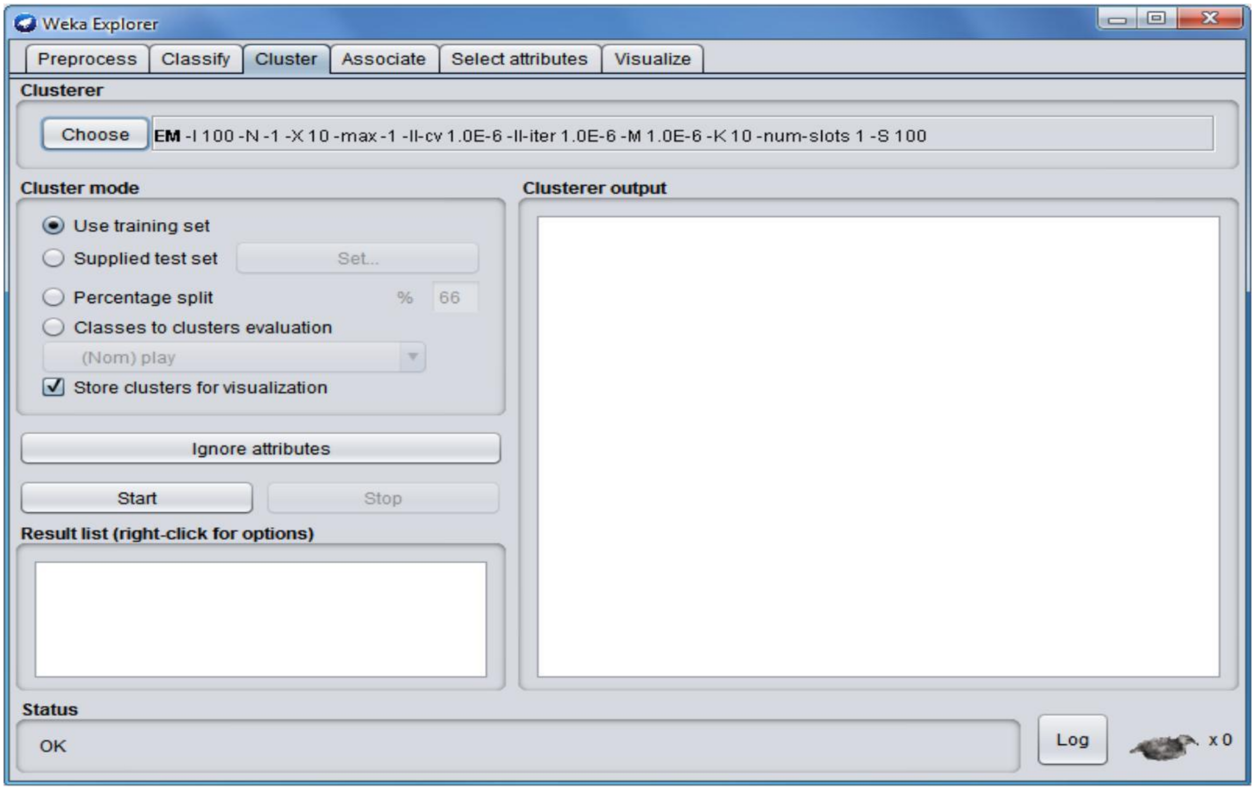
Class: play (Nom)

Status: OK



```
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

Step 2: click on Cluster, choose weka forcasting function SimpleKMean.



Clusterer

Choose **EM** -I 100 -N -1 -X 10 -max -1 -ll-cv 1.0E-6 -ll-iter 1.0E-6 -M 1.0E-6 -K 10 -num-slots 1 -S 100

Cluster mode

● Use training set

○ Supplied test set

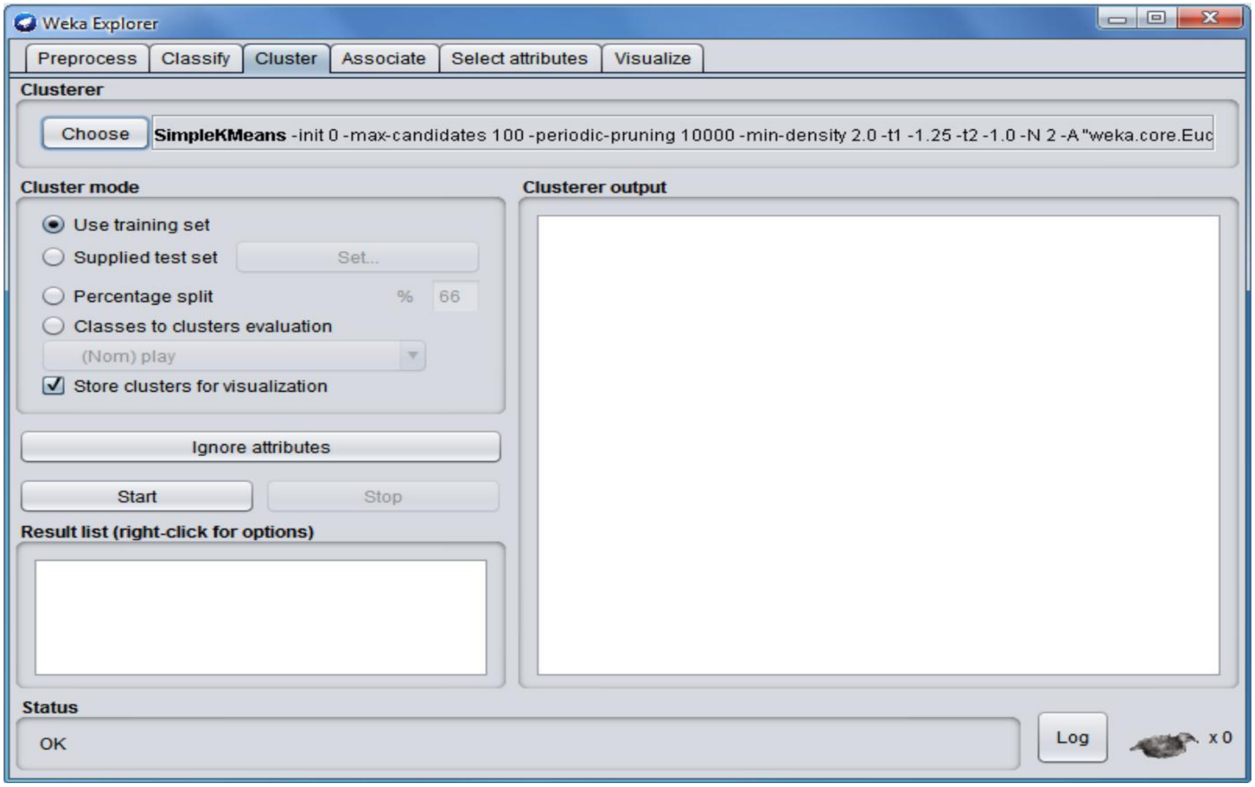
○ Percentage split % 66

○ Classes to clusters evaluation (Nom) play

☑ Store clusters for visualization

Status

OK



Clusterer

Choose **SimpleKMeans** -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.Euc

Cluster mode

● Use training set

○ Supplied test set

○ Percentage split % 66

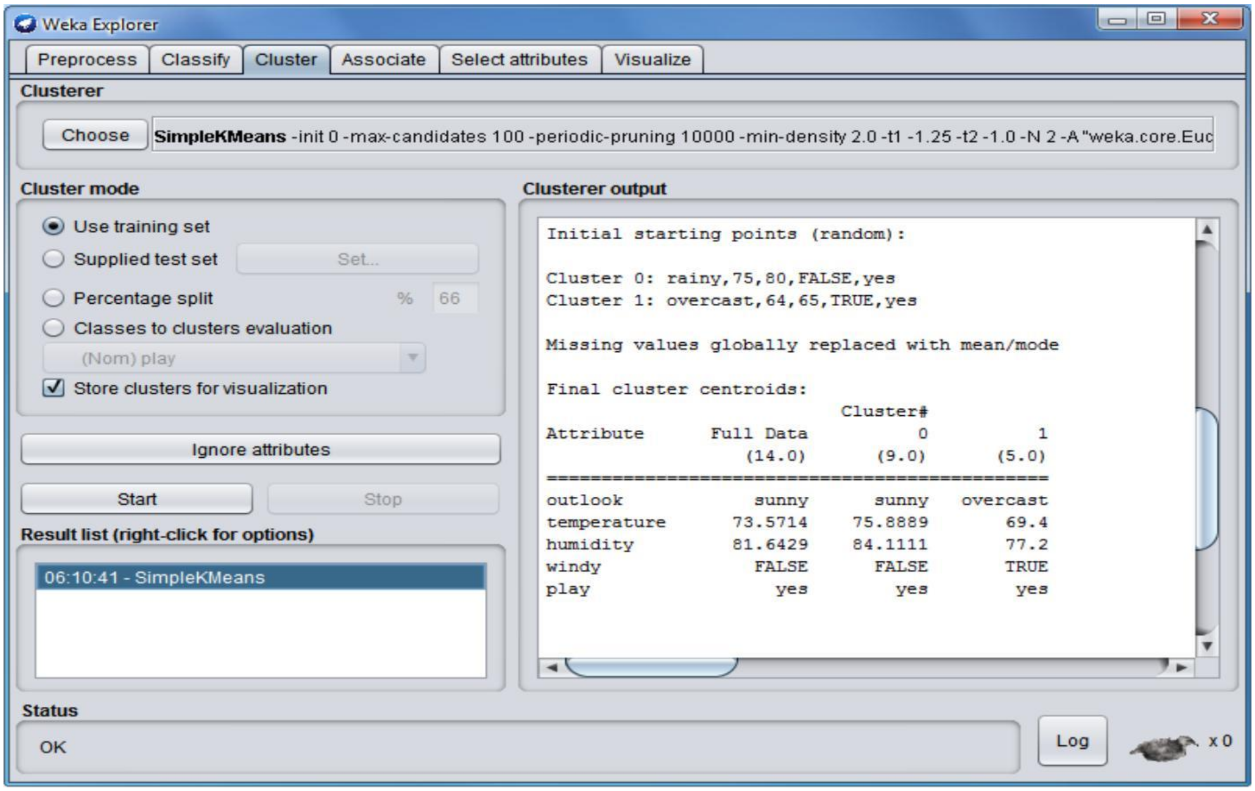
○ Classes to clusters evaluation (Nom) play

☑ Store clusters for visualization

Status

OK

Step 3: Click on Start. You can see the SimpleKMean on the input file.



Clusterer: **SimpleKMeans** -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.Euc

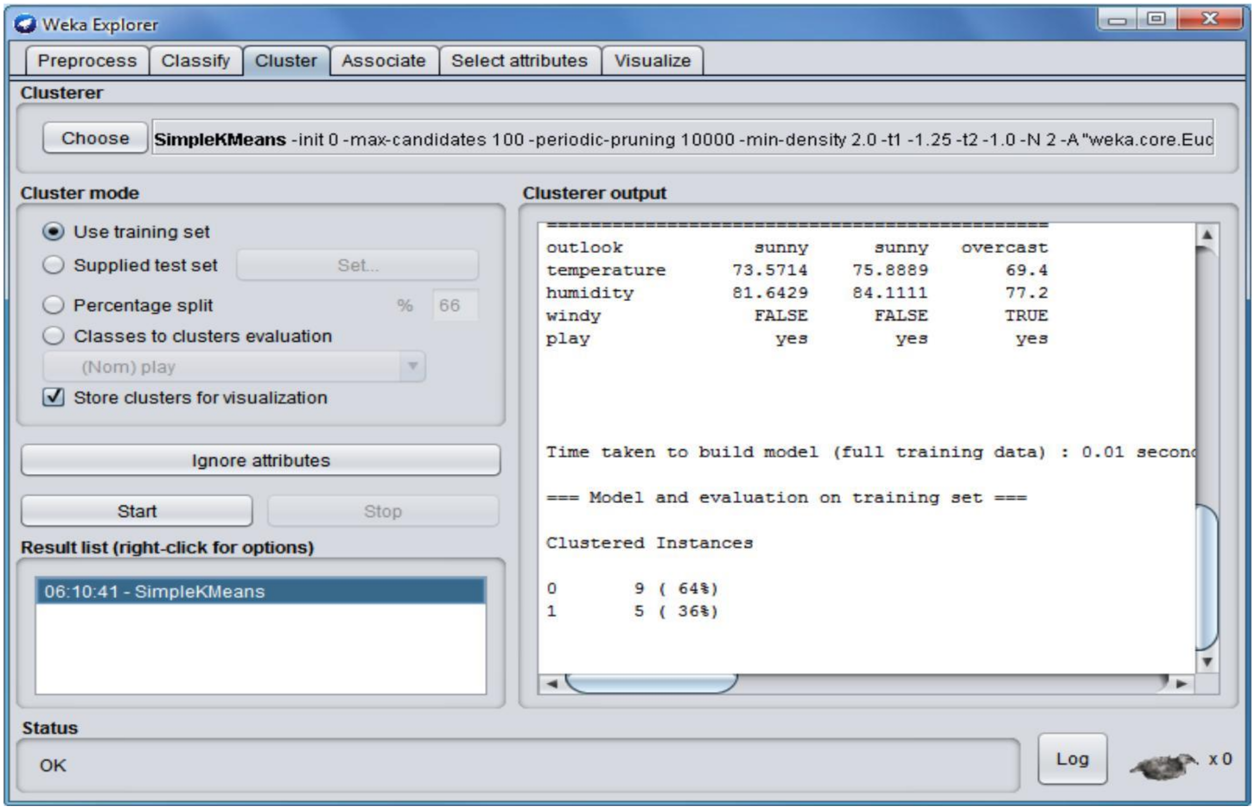
Cluster mode:

- ● Use training set
- ○ Supplied test set
- ○ Percentage split % 66
- ○ Classes to clusters evaluation
- (Nom) play
- ☑ Store clusters for visualization

Result list: 06:10:41 - SimpleKMeans

```
Initial starting points (random):

Cluster 0: rainy,75,80,FALSE,yes
Cluster 1: overcast,64,65,TRUE,yes

Missing values globally replaced with mean/mode

Final cluster centroids:
                          Cluster#
Attribute      Full Data         0          1
                  (14.0)     (9.0)      (5.0)
==============================================
outlook            sunny     sunny   overcast
temperature      73.5714   75.8889       69.4
humidity         81.6429   84.1111       77.2
windy              FALSE     FALSE       TRUE
play                 yes       yes        yes
```

Status: OK



Clusterer: **SimpleKMeans** -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.Euc

Cluster mode:

- ● Use training set
- ○ Supplied test set
- ○ Percentage split % 66
- ○ Classes to clusters evaluation
- (Nom) play
- ☑ Store clusters for visualization

Result list: 06:10:41 - SimpleKMeans

```
==============================================
outlook            sunny     sunny   overcast
temperature      73.5714   75.8889       69.4
humidity         81.6429   84.1111       77.2
windy              FALSE     FALSE       TRUE
play                 yes       yes        yes



Time taken to build model (full training data) : 0.01 second

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

Clustered Instances

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

Status: OK

Practical No. 3

Aim : **Write a map-reduce program to count the number of occurrences of each alphabetic character in the given dataset. The count for each should be case-insensitive(i.e include both upper-case and lower-case versions of the letter, ignore non-alphabetic characters).**

Source Code:

Charcount.java(Driver Class)

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path; import
org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.input.TextInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Charcount {

   public static void main(String[] args) throws Exception {
      //  TODO Auto-generated method stub Configuration conf
      = new Configuration(); Job job = new Job(conf,
      "Charcount"); job.setJarByClass(Charcount.class);
      job.setMapperClass(Charmap.class);
      job.setReducerClass(Charreduce.class);
      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);
      job.setMapOutputKeyClass(Text.class);
      job.setMapOutputValueClass(IntWritable.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);

   }

}
```

Charmap.java(Mapper Class)

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class Charmap extends Mapper<LongWritable, Text, Text, IntWritable> {
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        String line = value.toString();
        char[] carr = line.toCharArray();
        for (char c : carr) {
            System.out.println(c);
            context.write(new Text(String.valueOf(c)), new IntWritable(1));
        }

    }

}
```

Charreduce.java(Reducer Class)

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class Charreduce extends Reducer<Text, IntWritable, Text, IntWritable> { public
    void reduce(Text key,Iterable<IntWritable> values,Context context)throws
IOException,InterruptedException{
        int count = 0;
        IntWritable result = new IntWritable();
        for (IntWritable val : values) {
            count +=val.get();
            result.set(count);
        }
        String found = key.toString();
        if (found.equals("a") || found.equals("t") || found.equals("c") || found.equals("g"))
{
            context.write(key, result);
        }

    }
}
```

Practical No. 4

Aim : **Write a map-reduce program to count the number of occurrences of each word in the given dataset.(A word is defined as any string of alphabetic characters appearing between non-alphabetic characters like nature’s is two words. The count should be case-insensitive. If a word occurs multiple times in a line, all should be counted).**

WordCount.java(Driver Class)

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;


public class WordCount extends Configured implements Tool{ public
    int run(String[] args) throws Exception
    {
        //creating a JobConf object and assigning a job name for identification purposes
        JobConf conf = new JobConf(getConf(), WordCount.class);
        conf.setJobName("WordCount");

        //Setting configuration object with the Data Type of output Key and Value
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        //Providing the mapper and reducer class names
        conf.setMapperClass(WordCountMapper.class);
        conf.setReducerClass(WordCountReducer.class);
        //We wil give 2 arguments at the run time, one in input path and other is output
path
        Path inp = new Path(args[0]);
        Path out = new Path(args[1]);
        //the hdfs input and output directory to be fetched from the command line
        FileInputFormat.addInputPath(conf, inp); FileOutputFormat.setOutputPath(conf, out);
```

```
            JobClient.runJob(conf);
            return 0;
      }

      public static void main(String[] args) throws Exception
      {
            // this main function will call run method defined above.
        int res = ToolRunner.run(new Configuration(), new WordCount(),args);
            System.exit(res);
      }
}
```

WordCountMapper.java(Mapper Class)

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class WordCountMapper extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {

      //hadoop supported data types
      private final static IntWritable one = new IntWritable(1); private
      Text word = new Text();

      //map method that performs the tokenizer job and framing the initial key value pairs
      // after all lines are converted into key-value pairs, reducer is called.
      public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException
      {
            //taking one line at a time from input file and tokenizing the same String
            line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

          //iterating through all the words available in that line and forming the key value
pair
            while (tokenizer.hasMoreTokens())
            {
               word.set(tokenizer.nextToken());
               //sending to output collector which inturn passes the same to reducer
                 output.collect(word, one);
            }
       }
}
```

WordCountReducer.java(Reducer Class)

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class WordCountReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable>
{
      //reduce method accepts the Key Value pairs from mappers, do the aggregation based on keys
and produce the final out put
      public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException
      {
            int sum = 0;
            /*iterates through all the values available with a key and add them together and give the
            final result as the key and sum of its values*/ while
          (values.hasNext()) {

               sum += values.next().get();
          }
          output.collect(key, new IntWritable(sum));
      }
}
```

Practical No. 7

Aim :**Write a program to construct different types of k-shingles for given document.**
Installation of required packages before executing program:-

```
install.packages("tm")
require("tm")
install.packages("devtools")
```

```
readinteger <- function()
{
        n <- readline(prompt="Enter value of k-1: ")
        k<-as.integer(n)
        u1 <- readLines(E:/BA/Hadoop.txt")
        Shingle<-0
        i <-0 while(i<nchar(u1)-
        k+1)
        {
                Shingle[i] <- substr(u1, start=i, stop=i+k)
                print(Shingle[i])
                i=i+1
        }
}
if(interactive()) readinteger()
```

OutPut:-

```
> if(interactive()) readinteger() Enter
value of k-1: 2 character(0)
[1] "thi"
[1] "his"
[1] "is "
[1] "s i"
[1] " is"
[1] "is "
[1] "s a"
[1] " a "
[1] "a t"
[1] " te"
[1] "tex"
[1] "ext"
[1] "xt."
```

OutPut:-

```
> if(interactive()) readinteger() Enter
value of k-1: 3 character(0)
[1] "this"
[1] "his "
[1] "is i"
[1] "s is"
[1] " is "
[1] "is a"
[1] "s a "
[1] " a t"
[1] "a te"
```

OutPut:-

```
> if(interactive()) readinteger() Enter
value of k-1: 4 character(0)
[1] "this "
[1] "his i"
[1] "is is"
[1] "s is "
[1] " is a"
[1] "is a "
[1] "s a t"
[1] " a te"
[1] "a tex"
[1] " text"
[1] "text."
[1] "ext. "
```

Practical No. 8

Aim : **Write a program for measuring similarity among documents and detecting passages which have been reused.**

Installation of required packages before executing program:-

```
install.packages("tm")
require("tm")
install.packages("ggplot2")
install.packages("textreuse")
install.packages("devtools")
```

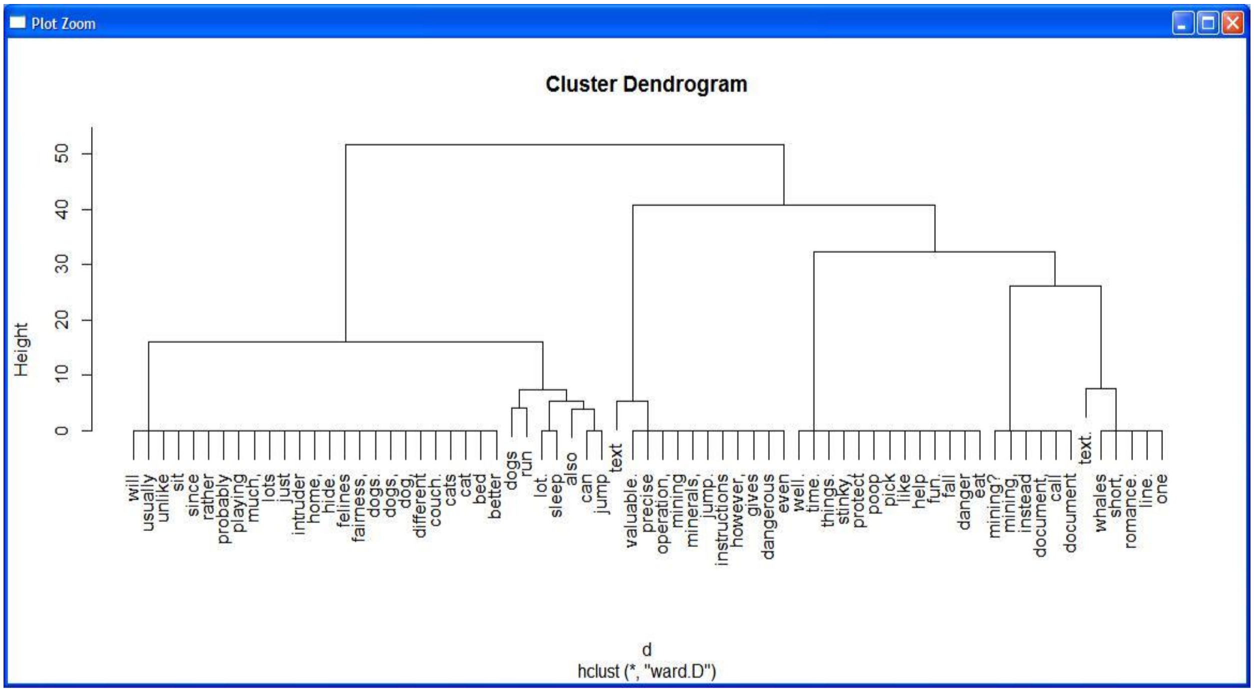
Source Code 1:-

```
my.corpus <- Corpus(DirSource("c:/msc/r-corpus"))
my.corpus <- tm_map(my.corpus, removeWords, stopwords("english"))
my.tdm <- TermDocumentMatrix(my.corpus)
#inspect(my.tdm)
my.dtm <- DocumentTermMatrix(my.corpus, control = list(weighting =
weightTfIdf, stopwords = TRUE))
#inspect(my.dtm)
my.df <- as.data.frame(inspect(my.tdm))
my.df.scale <- scale(my.df)
d <- dist(my.df.scale,method="euclidean") fit
<- hclust(d, method="ward") plot(fit)
```

OutPut:-

```
<<TermDocumentMatrix (terms: 69, documents: 6)>>
Non-/sparse entries: 97/317
Sparsity : 77%
Maximal term length: 12
Weighting : term frequency (tf)
Docs
Terms                   File1.txt File2.txt File3.txt File4.txt File5.txt File6.txt
also        0      1        1        1       0        0
bed         0      0        0        1       0        0
better      0      0        0         1       0        0
call        0      1         0       0       0        0
can         0      0        1        1       0        0
cat          0     0        0        1       0        0
cats         0     0        0        1       0        0
couch.       0     0        0        1       0        0
```

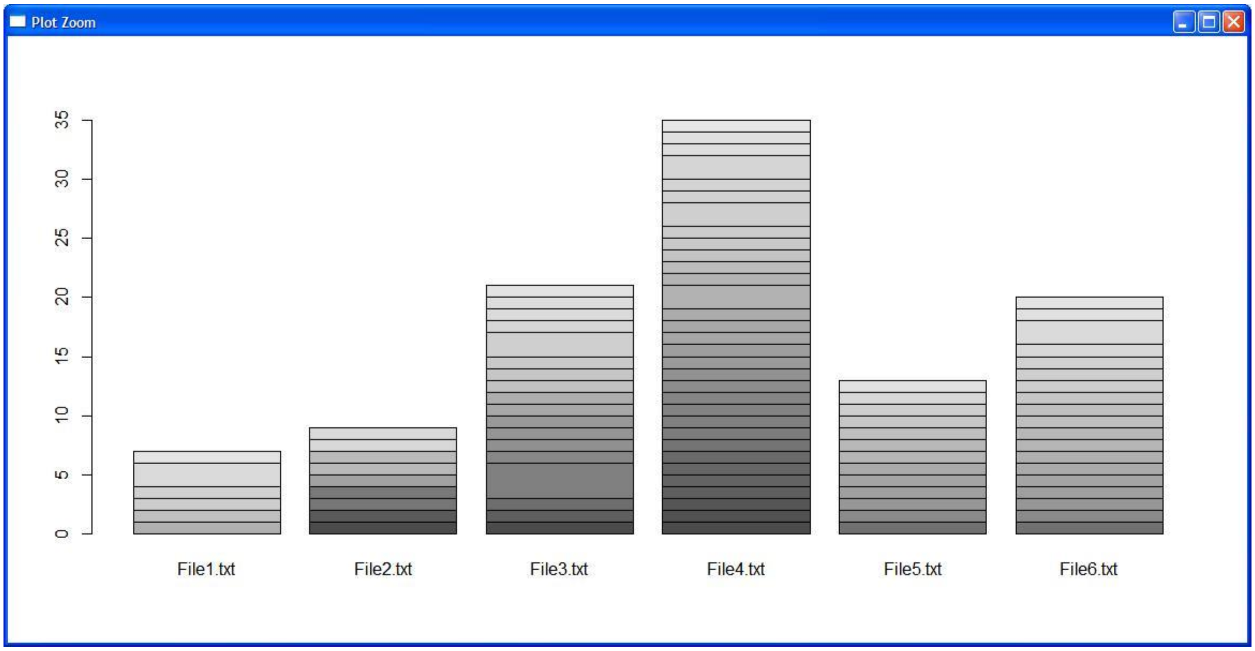
```
>   barplot(as.matrix(my.tdm))
> my.df.scale <- scale(my.df)
> d <- dist(my.df.scale,method="euclidean")
> fit <- hclust(d, method="ward")
The "ward" method has been renamed to "ward.D"; note new "ward.D2" >
plot(fit)
```



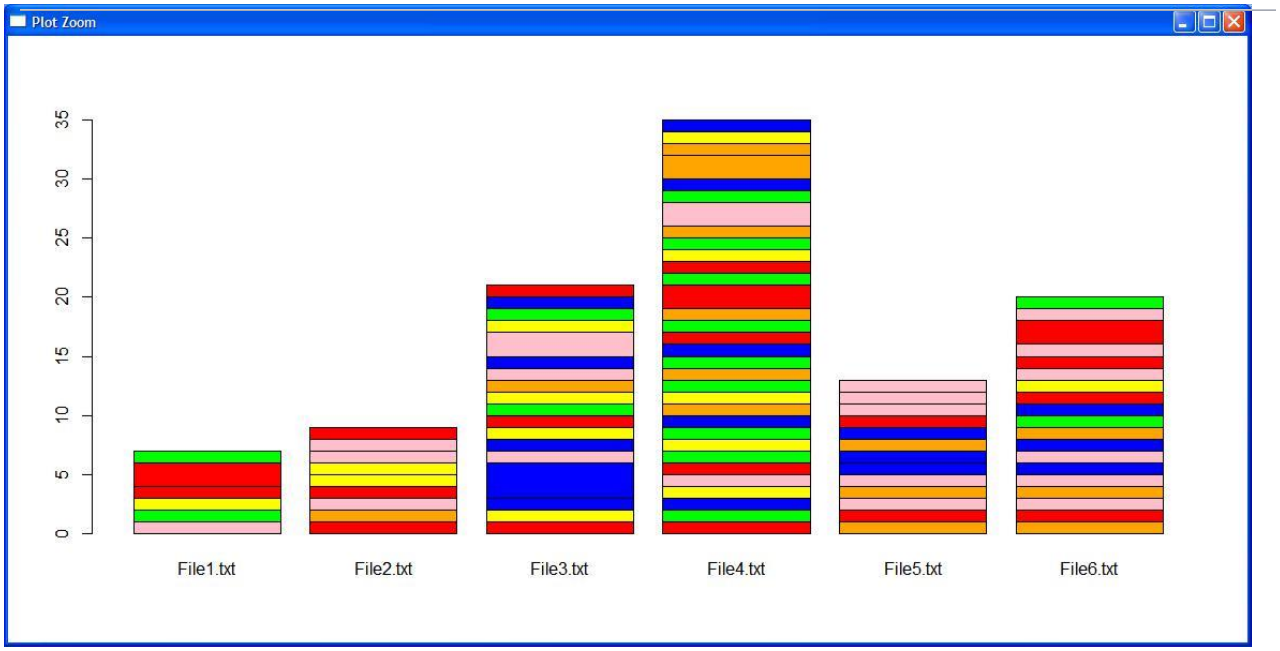
Sourec code 2 (using bar plot with and without color):-

```
my.corpus <- Corpus(DirSource("c:/msc/r-corpus"))
my.corpus <- tm_map(my.corpus, removeWords, stopwords("english"))
my.tdm <- TermDocumentMatrix(my.corpus)
inspect(my.tdm)
my.df <- as.data.frame(inspect(my.tdm))
barplot(as.matrix(my.tdm))
#barplot(as.matrix(my.tdm),col = color)
```

OutPut:-



barplot(as.matrix(my.tdm),col = color)



Jaccard similarity

Similarity of asymmetric binary attributes[edit]

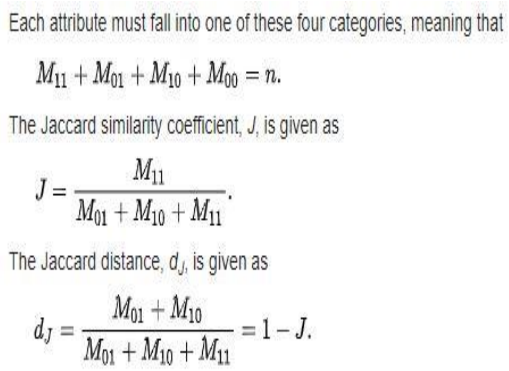
Given two objects, *A* and *B*, each with *n* binary attributes, the Jaccard coefficient is a useful measure of the overlap that *A* and *B*share with their attributes. Each attribute of *A* and *B* can either be 0 or 1. The total number of each combination of attributes for both *A* and *B* are specified as follows:

represents the total number of attributes where *A* and *B* both have a value of 1.

represents the total number of attributes where the attribute of *A* is 0 and the attribute of *B* is 1.

represents the total number of attributes where the attribute of *A* is 1 and the attribute of *B* is 0.

represents the total number of attributes where *A* and *B* both have a value of 0.



Each attribute must fall into one of these four categories, meaning that

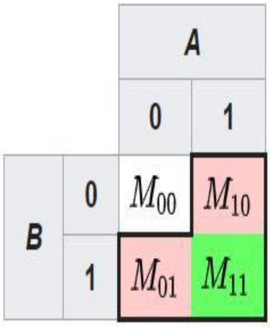
$$M_{11}+M_{01}+M_{10}+M_{00}=n.$$

The Jaccard similarity coefficient, *J*, is given as

$$J=\frac{M_{11}}{M_{01}+M_{10}+M_{11}}.$$

The Jaccard distance, d_J, is given as

$$d_J=\frac{M_{01}+M_{10}}{M_{01}+M_{10}+M_{11}}=1-J.$$



		A	
		0	1
B	0	M_{00}	M_{10}
	1	M_{01}	M_{11}

Sourec code 3 (using minhash and jaccard similarity):-

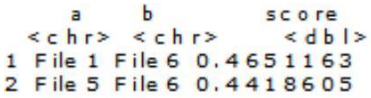
```
library(textreuse)
```

Source Code:-

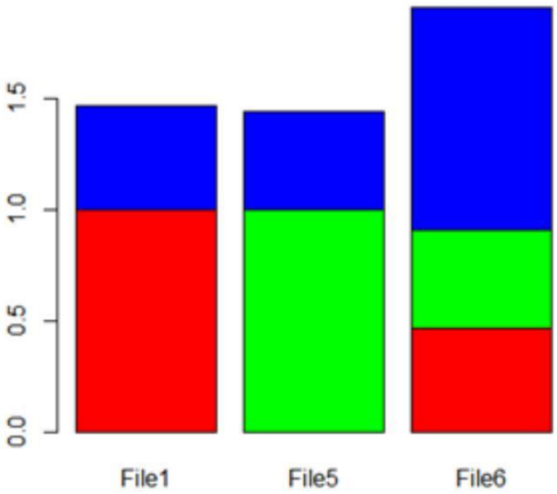
```
minhash <- minhash_generator(200, seed = 235)
ats <- TextReuseCorpus(dir = "c:/msc/r-corpus", tokenizer = tokenize_ngrams, n = 5,
minhash_func = minhash)
buckets <- lsh(ats, bands = 50, progress = interactive())
```

```
candidates <- lsh_candidates(buckets)
scores <- lsh_compare(candidates, ats, jaccard_similarity, progress = FALSE)
scores
color <- c("red","green","blue","orange","yellow","pink")
barplot(as.matrix(scores),col = color)
```

Output:



```
      a      b           score
  <chr>  <chr>           <dbl>
1 File1  File6  0.4651163
2 File5  File6  0.4418605
```



Practical No. 9

Aim : Write **a program to compute the n-moment for a given stream where n is given.**

Source Code:

```
import java.io.*;
import java.util.*;

class n_moment2
{
 public static void main(String args[])
 {

	 int n=15;
	String stream[]={"a","b","c","b","d","a","c","d","a","b","d","c","a","a","b"};

	 int zero_moment=0,first_moment=0,second_moment=0,count=1,flag=0;
	 ArrayList<Integer> arrlist=new ArrayList();;

	System.out.println("Arraylist elements are :: ");

	for(int i=0;i<15;i++)
	{
		System.out.print(stream[i]+" ");
	 }
	Arrays.sort(stream);

	//Calculate Zeroth moment(calculates unique elements-raised to zero)
	for(int i=1;i<n;i++)
	{
		 if(stream[i]==stream[i-1])
		{
			count++;
		 }
		else
		{
			arrlist.add(count);
			count=1;
		 }
	 }
	arrlist.add(count);

   zero_moment=arrlist.size();
   System.out.println("\n\n\nValue of Zeroth moment for given stream ::"+zero_moment);

	//Calculate First moment(Calculate length of the stream-raised to one)
	 for(int i=0;i<arrlist.size();i++)
```

```
             {
                    first_moment+=arrlist.get(i);
              }
             System.out.println("\n\nValue of First moment for given stream ::"+first_moment);

              //Calculate Second moment(raised to two)
             for(int i=0;i<arrlist.size();i++)
             {
                   int j=arrlist.get(i);
                   second_moment+=(j*j);
             }
    System.out.println("\n\nValue of Second moment for given stream ::"+second_moment);
  }
}
```

Output :

```
Arraylist elements are ::
a b c b d a c d a b d c a a b


Value of Zeroth moment for given stream ::4


Value of First moment for given stream ::15


Value of Second moment for given stream ::59
```

Practical No. 10

Aim : Write a program to demonstrate the Alon-Matias-Szegedy Algorithm for second moments.

Source Code:

```
import java.io.*;
import java.util.*;
class AMSA
{
        public static int findCharCount(String stream,char XE,int random,int n)
        {
                int countOccurance=0;

                 for(int i=random;i<n;i++)
                 {
                         if(stream.charAt(i)==XE)
                         {
                                 countOccurance++;//System.out.println(countOccurance+" "+i);
                         }
                }
                 return countOccurance;

        }

        public static int estimateValue(int XV1,int n)
        {
                int ExpValue;

                ExpValue=n*(2*XV1-1);
                return ExpValue;
         }



        public static void main(String args[])
        {
                int n=15;
                String stream="abcbdacdabdcaab";
                int random1=3,random2=8,random3=13;
                char XE1,XE2,XE3;
                int XV1,XV2,XV3;
                int ExpValuXE1, ExpValuXE2, ExpValuXE3;
                int apprSecondMomentValue;
                XE1=stream.charAt(random1-1);
                XE2=stream.charAt(random2-1);
                XE3=stream.charAt(random3-1);

                //System.out.println(XE1+" "+XE2+" "+XE3);

                 XV1=findCharCount(stream,XE1,random1-1,n);
                XV2=findCharCount(stream,XE2,random2-1,n);
                XV3=findCharCount(stream,XE3,random3-1,n);

                System.out.println(XE1+"="+XV1+" "+XE2+"="+XV2+" "+XE3+"="+XV3);

                ExpValuXE1=estimateValue(XV1,n);
                ExpValuXE2=estimateValue(XV2,n);
                ExpValuXE3=estimateValue(XV3,n);

                System.out.println("Expected value for "+XE1+" is :: "+ExpValuXE1);
                System.out.println("Expected value for "+XE2+" is :: "+ExpValuXE2);
                System.out.println("Expected value for "+XE3+" is :: "+ExpValuXE3);

                apprSecondMomentValue=(ExpValuXE1+ExpValuXE2+ExpValuXE3)/3;

                System.out.println("Approximate Second moment value using Alon-Matias-Szegedy is :: "+apprSecondMomentValue);
        }
}
```

Output:

```
c=3     d=2     a=2
Expected value for c is :: 75
Expected value for d is :: 45
Expected value for a is :: 45
Approximate Second moment value using Alon-Matias-Szegedy is :: 55
```