

## Index

Sr. No.	Practical Name	Page No	Date
1	Generate regression model and interpret the result for a given data set.	3-5	11 <sup>th</sup> July 2022
2	Generate forecasting model and interpret the result for a given data set.	6-9	18 <sup>th</sup> July 2022
3	Write a map-reduce program to count the number of occurrences of each alphabetic character in the given dataset. The count for each letter should be case-insensitive (i.e., include both upper-case and lower-case versions of the letter; Ignore non-alphabetic characters).	10-13	22 <sup>nd</sup> August 2022
4	De 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)	14-16	29 <sup>th</sup> August 2022
5	Write a program to construct different types of k-shingles for given document.	17-19	04 <sup>th</sup> August 2022
6	Write a program for measuring similarity among documents and detecting passages which have been reused.	20-21	12 <sup>th</sup> September 2022
7	Write a program to compute the n-moment for a given stream where n is given.	22-23	12 <sup>th</sup> September 2022
8	Write a program to demonstrate the Alon-Matias-Szegedy Algorithm for second moments.	24-25	19 <sup>th</sup> September 2022



## CERTIFICATE

This is here to certify that Mr. **Zeeshan Ali Sayyad**, Seat Number **051** M.Sc. II Computer Science, has satisfactorily completed the required number of experiments prescribed by the syllabus during the academic year 2022 – 2023.

Date:

Place: Mumbai

Teacher In-Charge

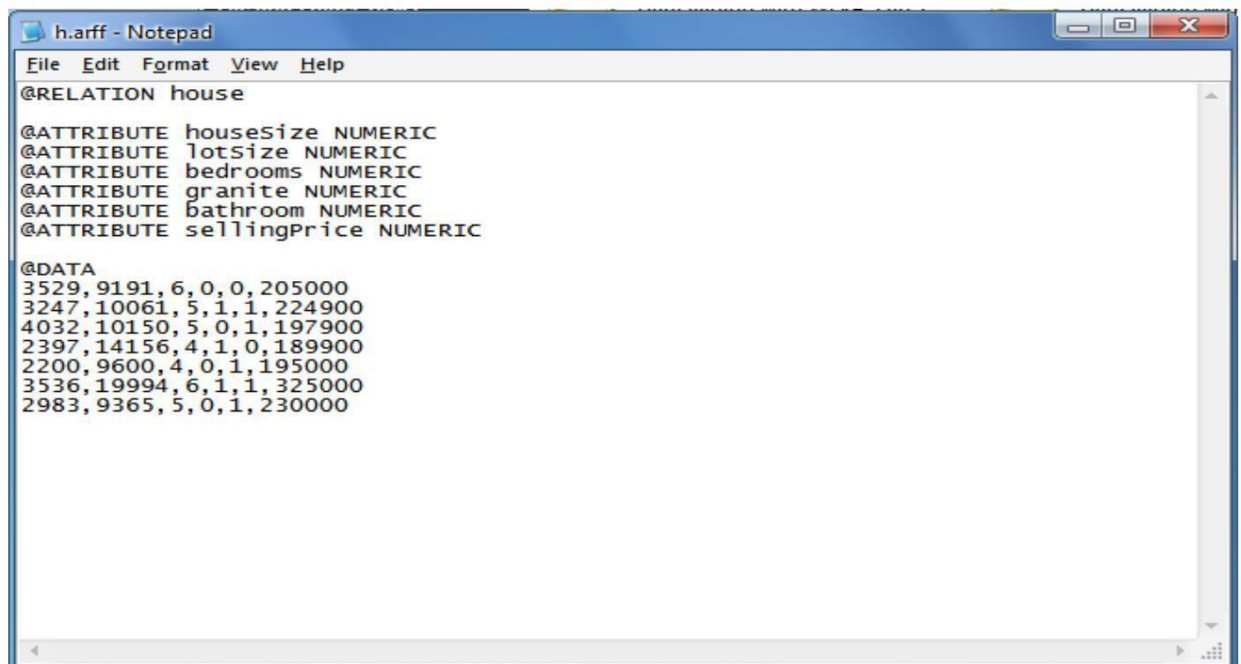
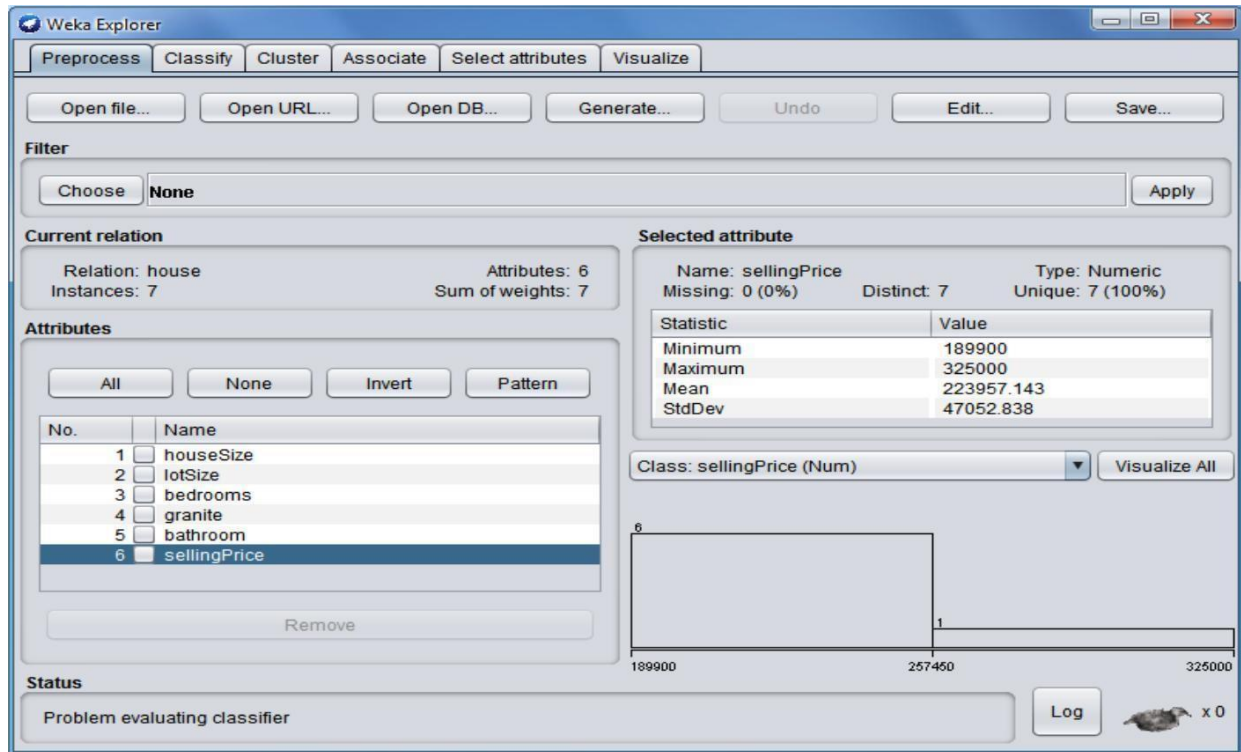
Head of Department

External Examiner

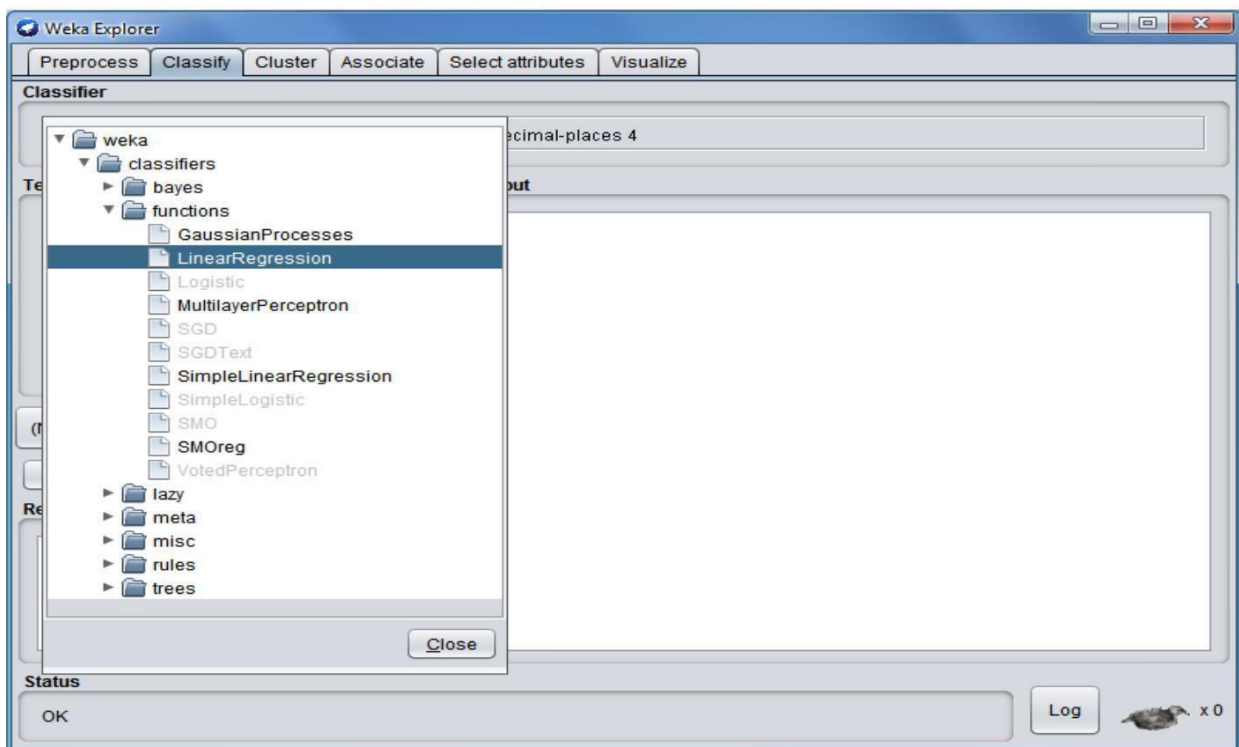
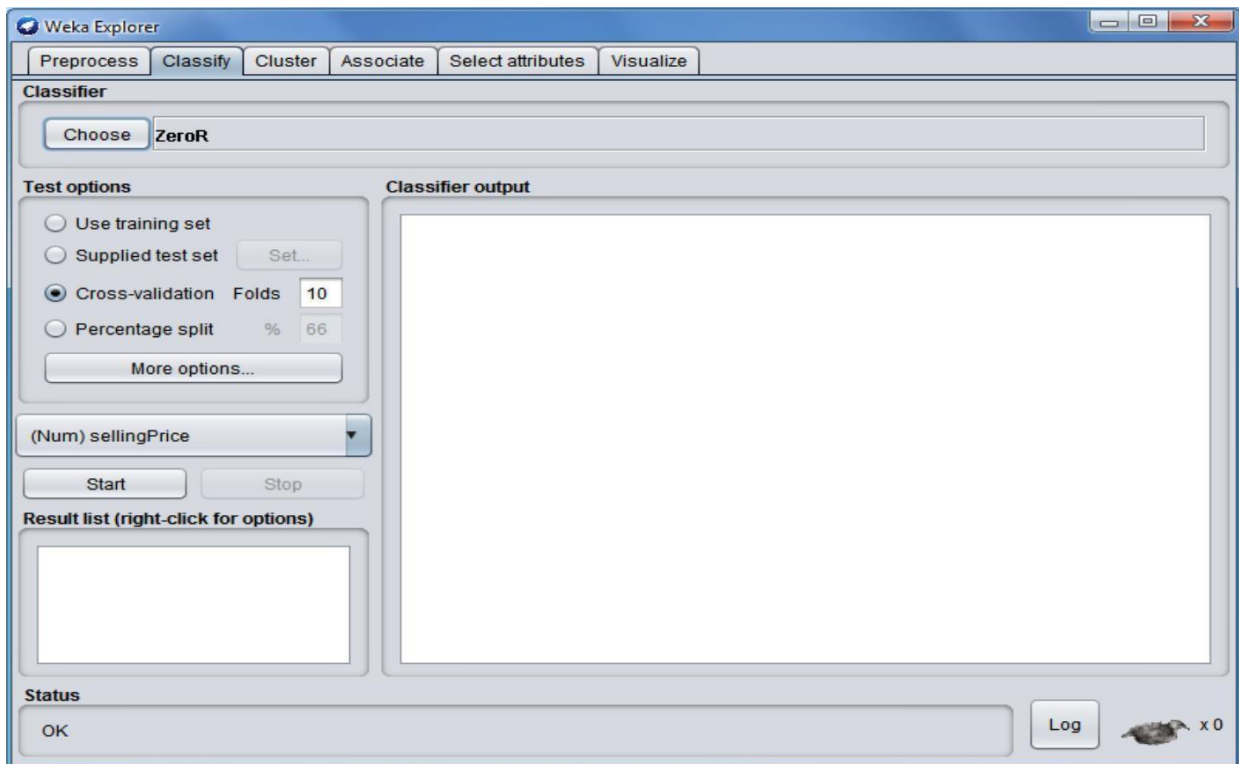
## Practical No. 1

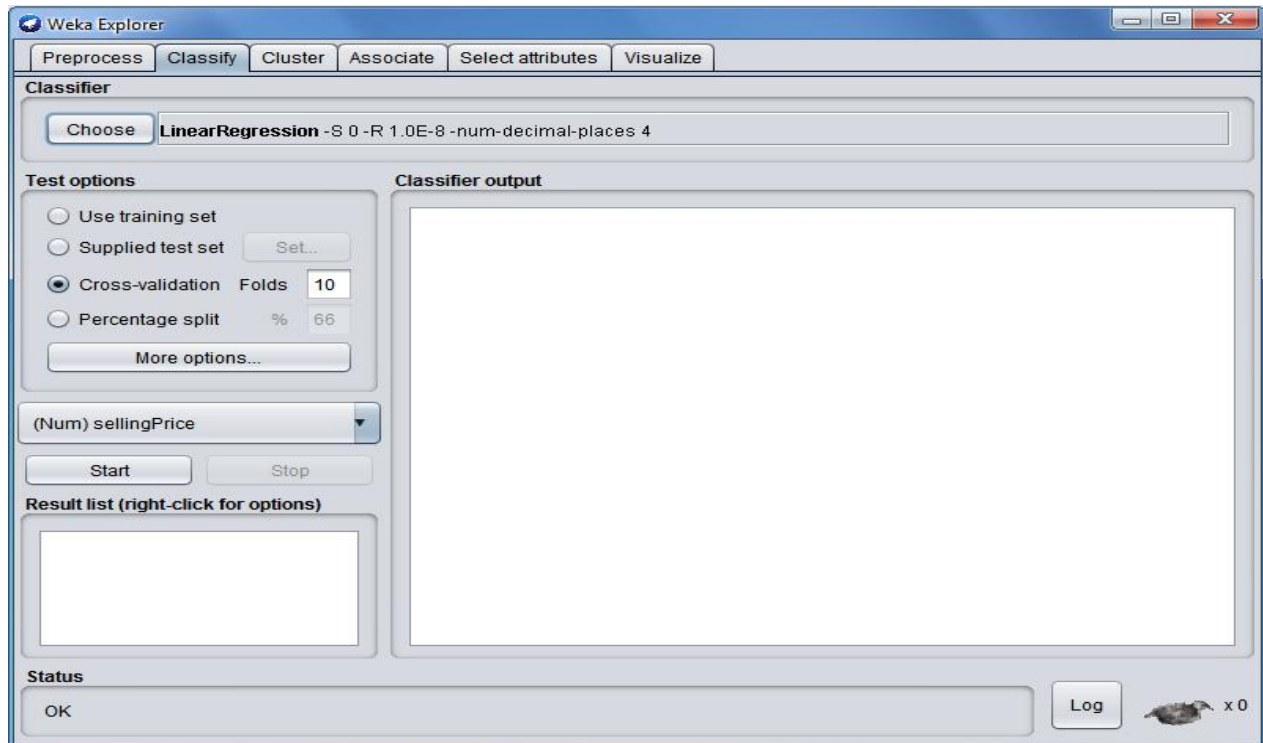
**Aim:** Generate Regression model and interpret the result for a given data set.

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

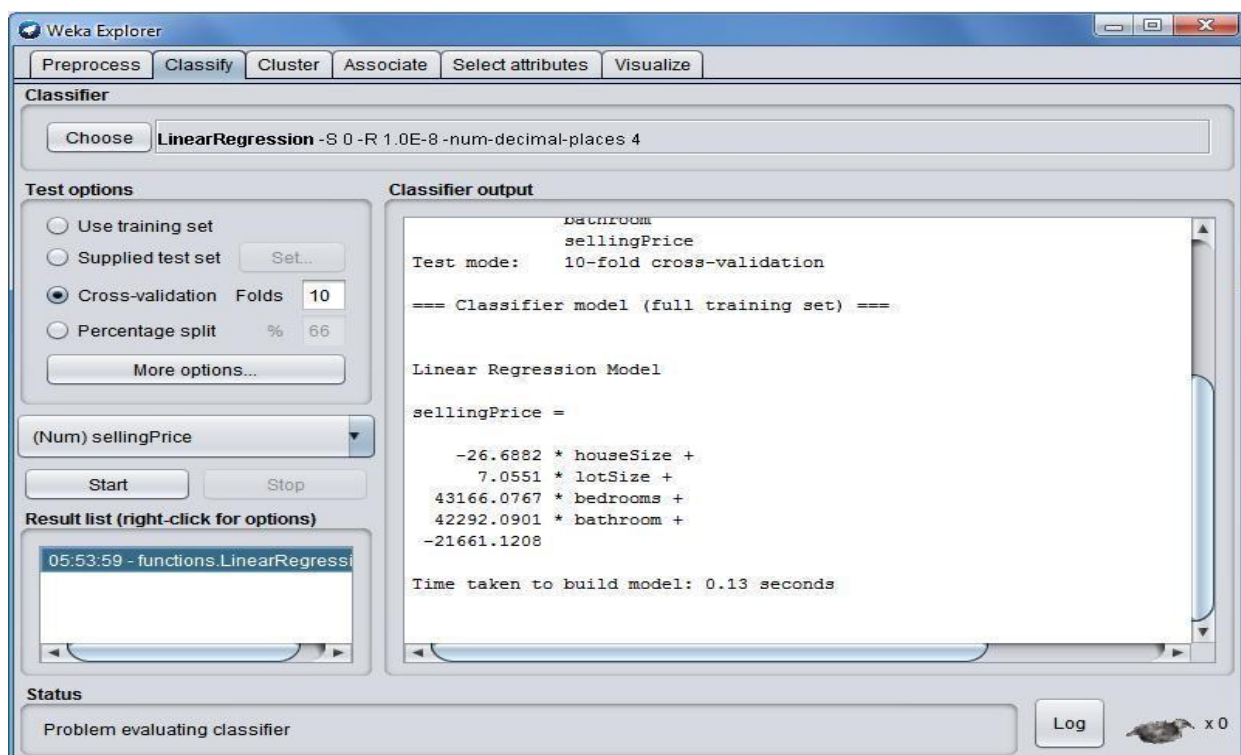


**Step 2:** Click on Classify, choose weka classifier function LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4.





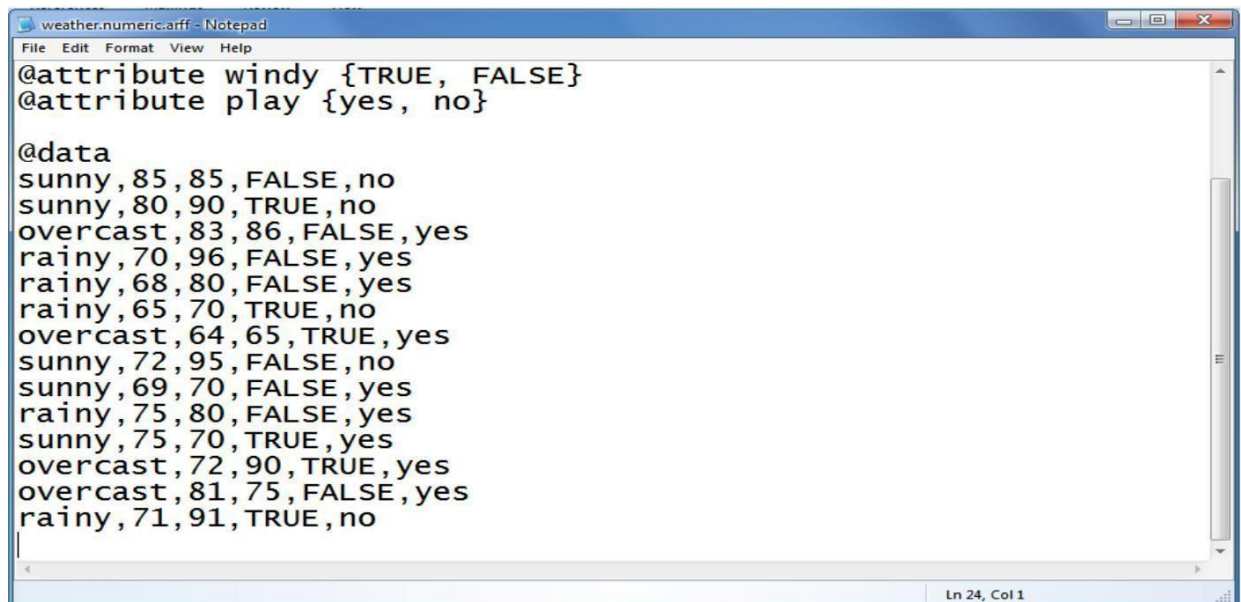
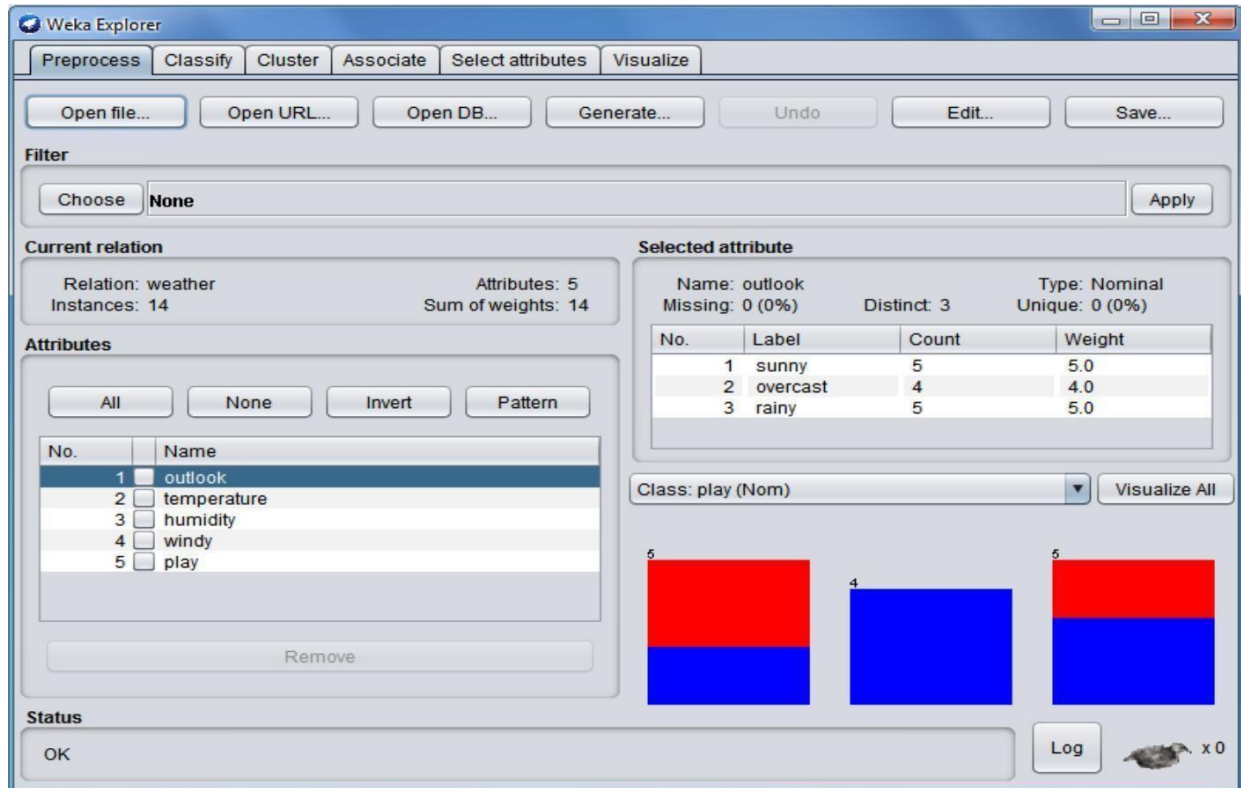
**Step 3:** Click on Start. You can see the linear regression on the input file.



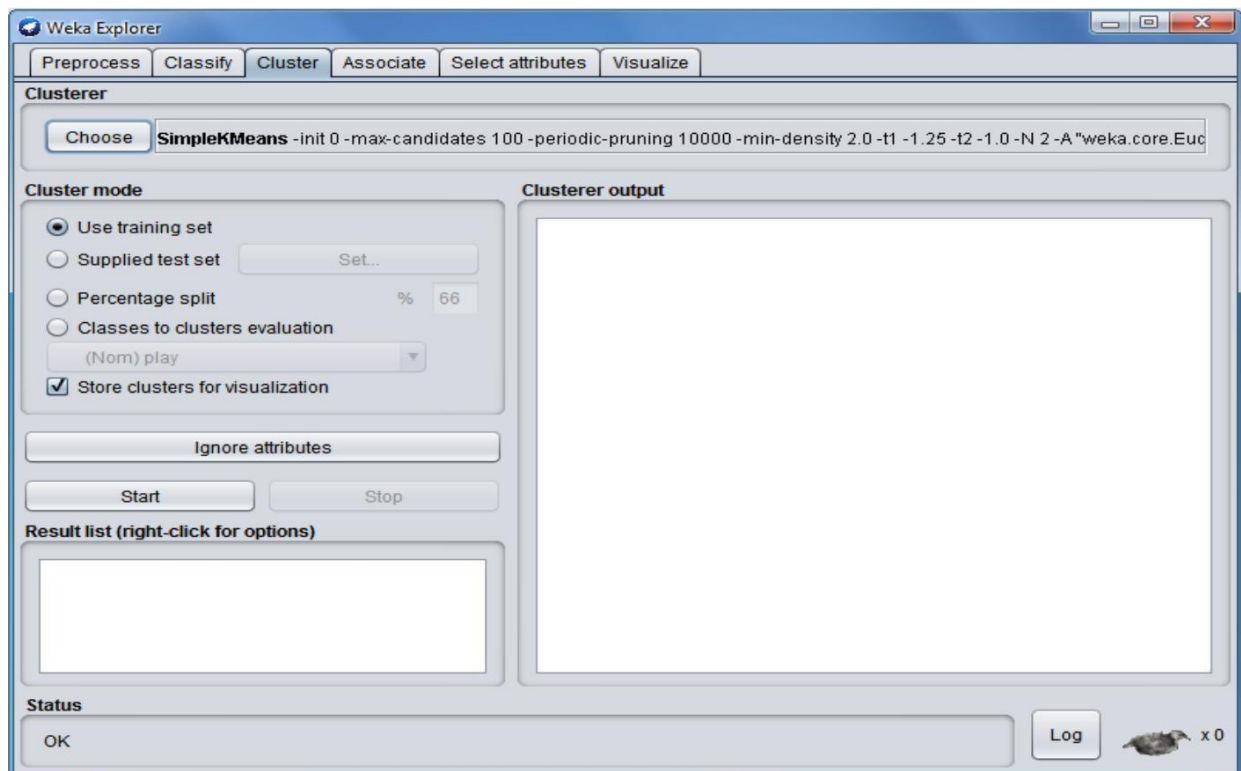
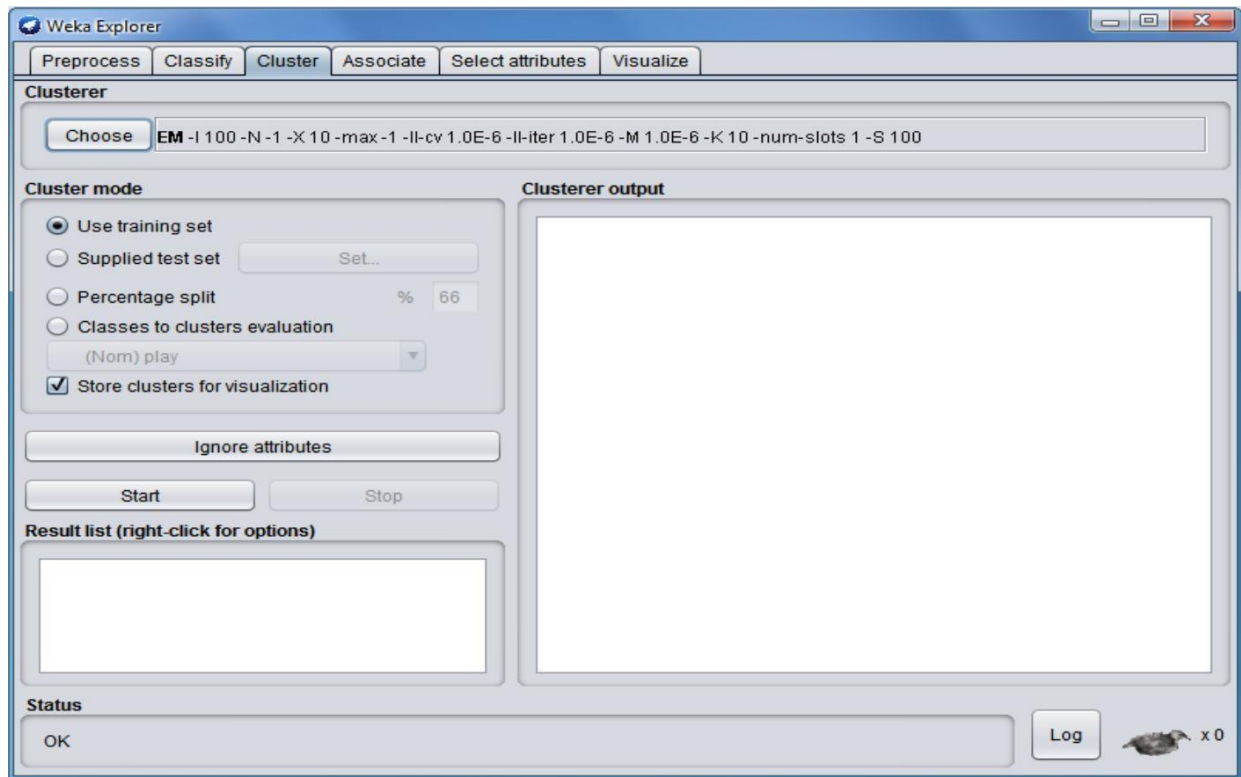
## Practical No. 2

**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.

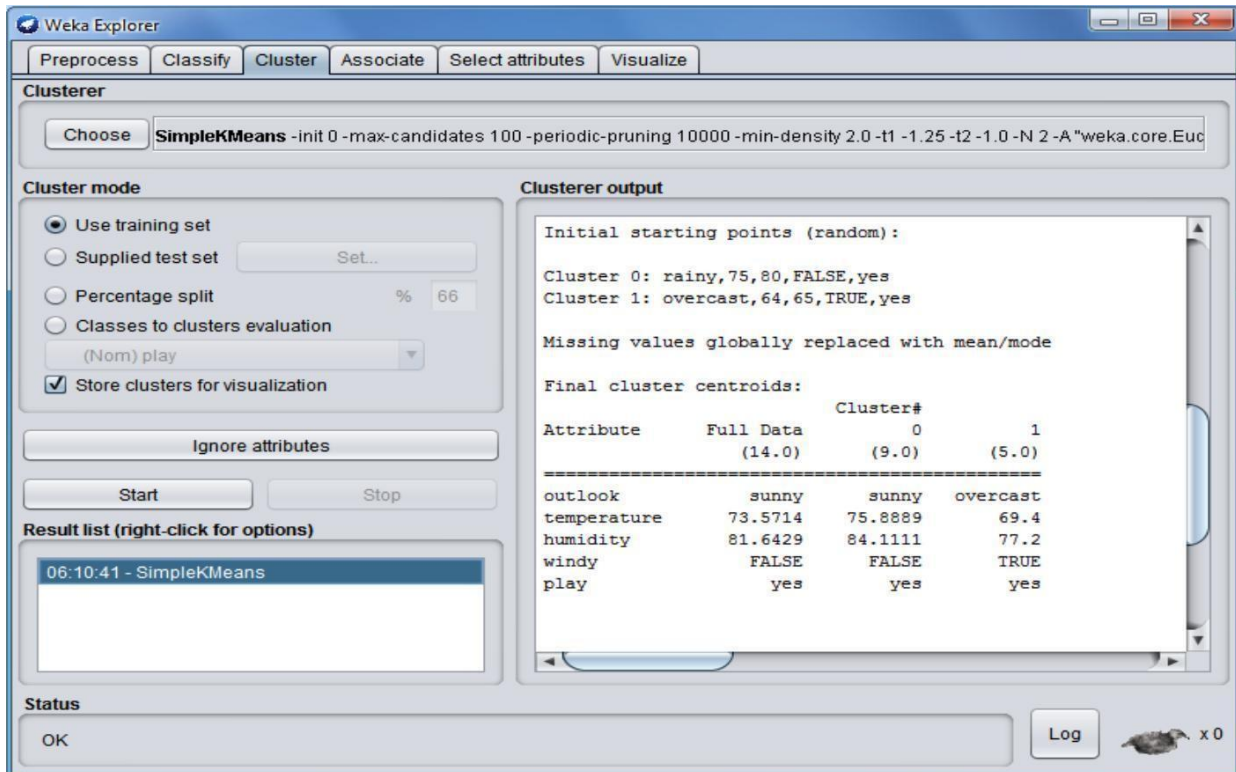


**Step 2:** click on Cluster, choose weka forecasting function SimpleKMean.





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



**Weka Explorer**

Preprocess Classify **Cluster** Associate Select attributes Visualize

**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 (Norm) play
- ☒ Store clusters for visualization

**Result list (right-click for options)**

06:10:41 - SimpleKMeans

**Clusterer output**

Initial starting points (random):

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

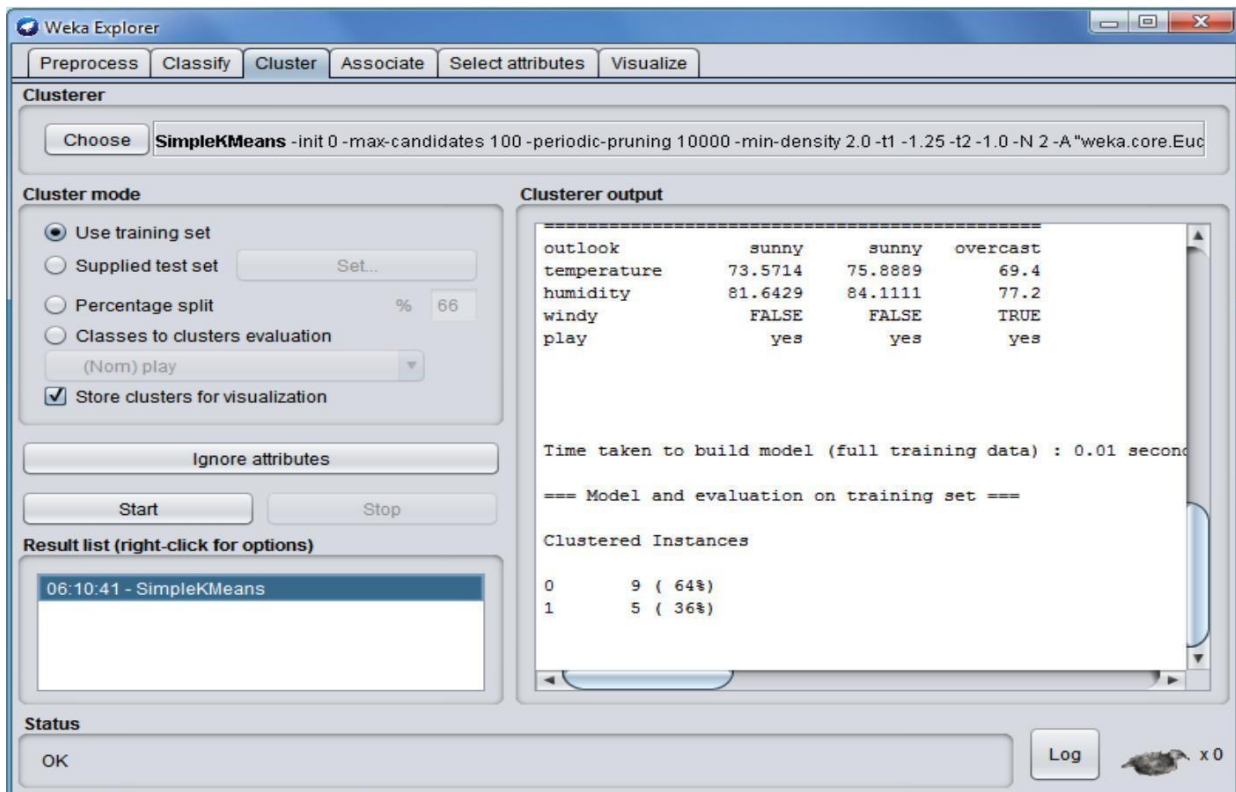
Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Full Data (14.0)	Cluster# 0 (9.0)	Cluster# 1 (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  x0



**Weka Explorer**

Preprocess Classify **Cluster** Associate Select attributes Visualize

**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 (Norm) play
- ☒ Store clusters for visualization

**Result list (right-click for options)**

06:10:41 - SimpleKMeans

**Clusterer output**

	sunny	sunny	overcast
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 seconds

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

Clustered Instances

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

**Status**

OK  x0



### **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. 5**

**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. 6

**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

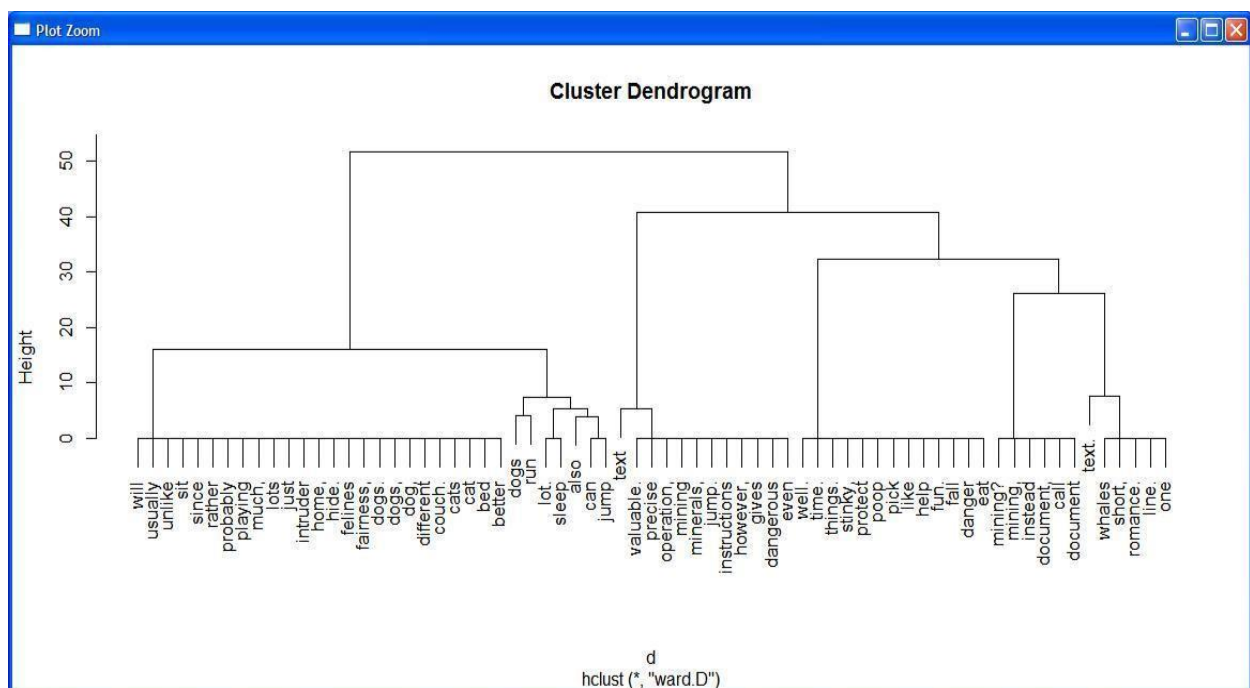
Weighting : term frequency (tf)

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)



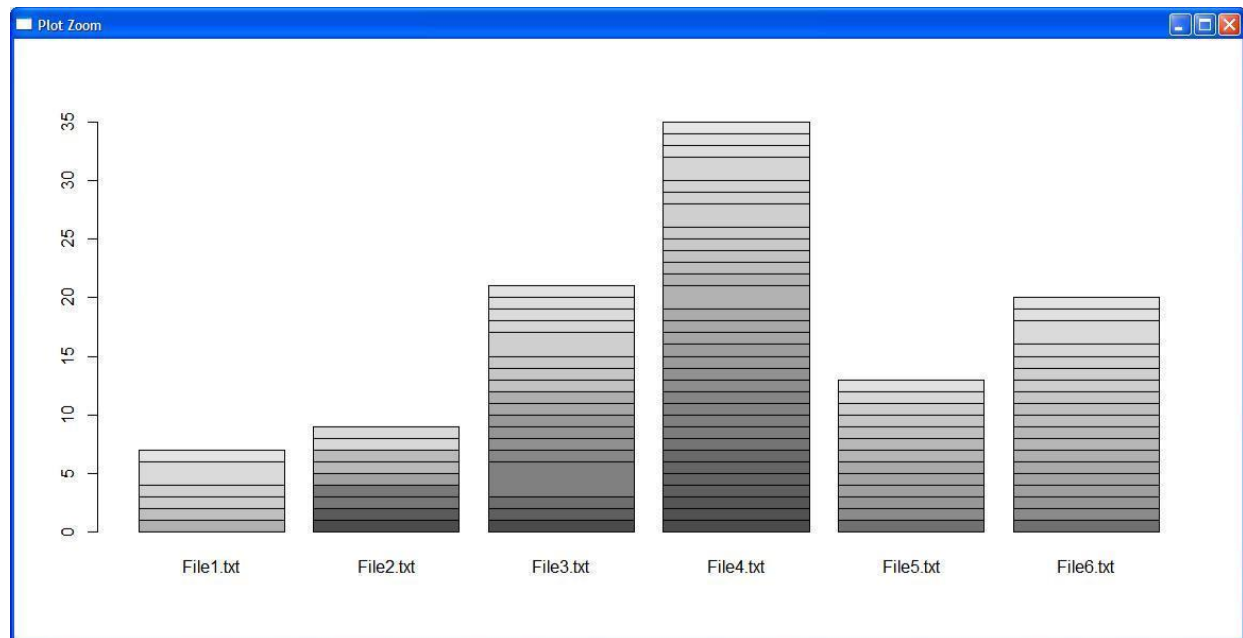
Source code 2 (using bar plot with and without color):-

```
my.corpus <- Corpus(DirSource("c:/msc/r-corpus"))

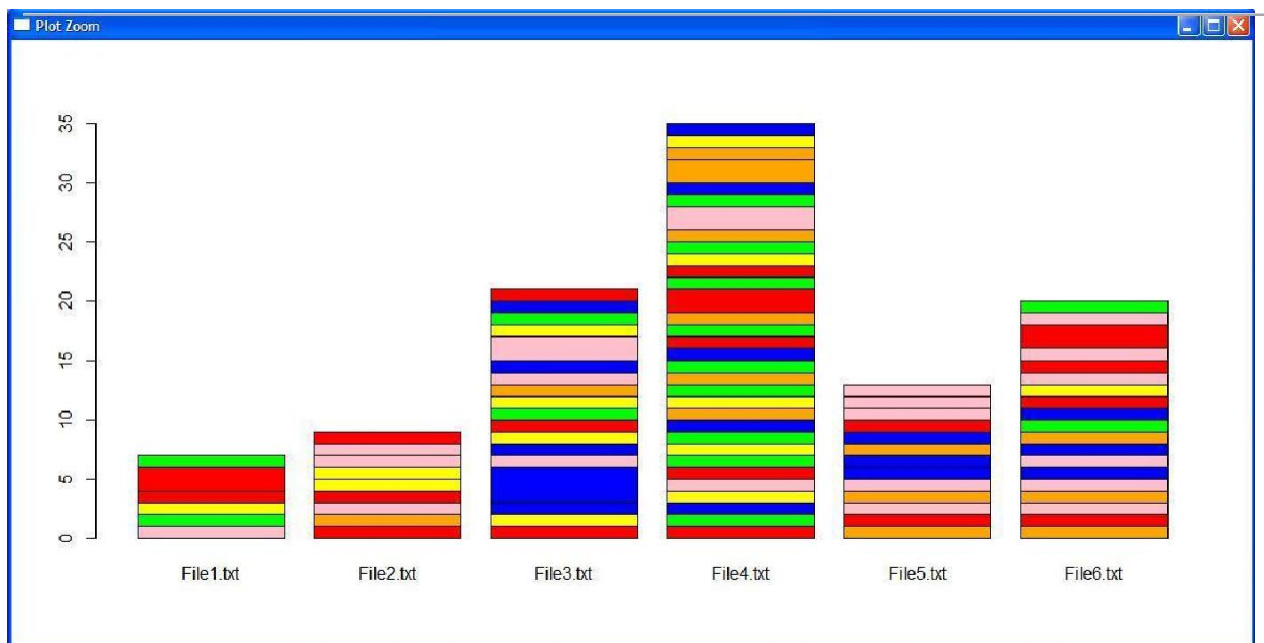
my.corpus <- tm_map(my.corpus, removeWords, stopwords("english"))
my.tdm <- TermDocumentMatrix(my.corpus)

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

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

		<b>A</b>	
		0	1
<b>B</b>	0	$M_{00}$	$M_{10}$
	1	$M_{01}$	$M_{11}$

Source 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)
```

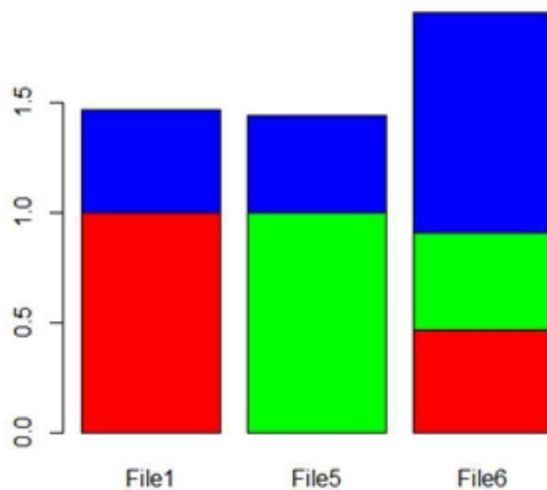
```
scores <- lsh_compare(candidates, ats, jaccard_similarity, progress = FALSE)  
scores
```

```
color <- c("red", "green", "blue", "orange", "yellow", "pink")
```

```
barplot(as.matrix(scores), col = color)
```

### Output:

```
   a      b      score  
<chr> <chr> <dbl>  
1 File 1 File 6 0.4651163  
2 File 5 File 6 0.4418605
```





## Practical No. 7

**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\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. 8

**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