Competitive Programming Lab - 4

Academic year: 2020-2021 **Semester:** Long Sem

Faculty Name: Dr. Ajith Jublison sir **Date:** 23 / 6 / 2022

Student name: Taran Mamidala **Reg. no.:** 19BCE7346

Q1.) Highest and lowest array sum and product with removing duplicates.

A[ ]= [a0, a1, a2, a3....an]

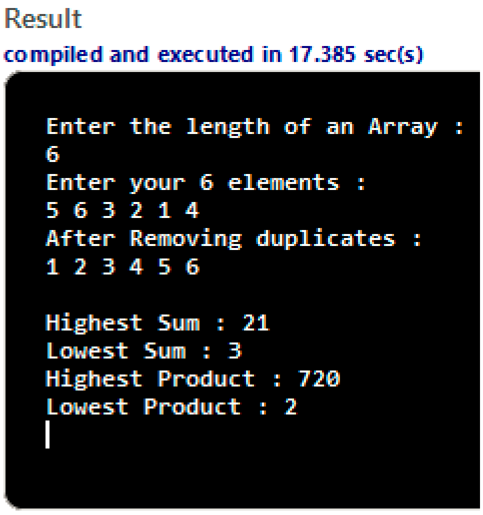
Range 2>=n

Sample I/P= [5,6,3,2,1,4]

Sample O/P:

Highest Product=720 Highest Sum=21

Lowest Product =2 Lowest Sum=3



Result

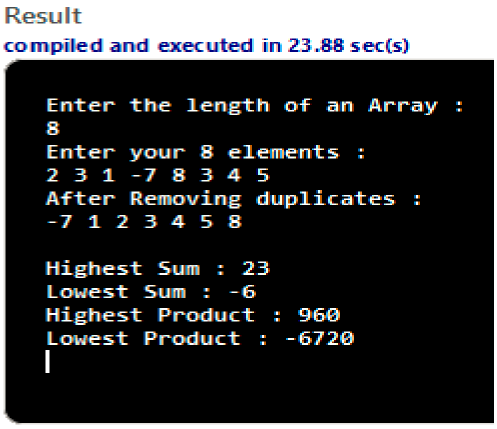
compiled and executed in 17.385 sec(s)

```
Enter the length of an Array :
6
Enter your 6 elements :
5 6 3 2 1 4
After Removing duplicates :
1 2 3 4 5 6

Highest Sum : 21
Lowest Sum : 3
Highest Product : 720
Lowest Product : 2
```

CODE:

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

public class FindHiLowOfSumPro{

    static ArrayList<Integer> sumlist = new ArrayList<Integer>(5);
    static ArrayList<Integer> prolist = new ArrayList<Integer>(5);
    static void storeSum(int sum){
    sumlist.add(sum);
    }
    static void storePro(int pro){
      prolist.add(pro);
      }

    static void allcombi(int arr[], int data[], int start,
                                           int end, int index, int r)
    {
       int sum = 0;
       int pro = 1;
       if (index == r)
       {
             for (int j=0; j<r; j++){
//                 System.out.print(data[j]+" ");
             sum = sum + data[j];
             pro = pro * data[j];
             }
//                 System.out.println(" ");
             storeSum(sum);
             storePro(pro);
             return;
       }

       for (int i=start; i<=end && end-i+1 >= r-index; i++)
       {
             data[index] = arr[i];
             allcombi(arr, data, i+1, end, index+1, r);
       }
    }
      public static int removeDuplicates(int arr[], int n){
      if (n==0 || n==1){
            return n;
      }
      int[] temp = new int[n];
      int j = 0;
      for (int i=0; i<n-1; i++){
            if (arr[i] != arr[i+1]){
                  temp[j++] = arr[i];
            }
      }
      temp[j++] = arr[n-1];
      // Here i am Changing original array
      for (int i=0; i<j; i++){
            arr[i] = temp[i];
      }
      return j;
      }
    static void findHiLowOfSumPro(int arr[], int n, int range)
    {

       int data[]=new int[range];
       allcombi(arr, data, 0, n-1, 0, range);
    }

    public static void main (String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the length of an Array : ");
      int n = sc.nextInt();
      int arr[] = new int[n];
      System.out.println("Enter your "+n+" elements :");
      for(int i=0;i<n;i++){
            arr[i] = sc.nextInt();
      }

      Arrays.sort(arr);
      n = removeDuplicates(arr, n);
            System.out.println("\nAfter Removing duplicates : ");
      for (int i=0; i<n; i++)
            System.out.print(+arr[i]+" ");

            System.out.println("");
      for(int i=2;i<n+1;i++){
            int range=i;

      findHiLowOfSumPro(arr, n, range);
      }
       System.out.println("Highest Sum : "+Collections.max(sumlist));
       System.out.println("Lowest Sum : "+Collections.min(sumlist));
       System.out.println("Highest Product :
"+Collections.max(prolist));
       System.out.println("Lowest Product : "+Collections.min(prolist));
    }
}
```

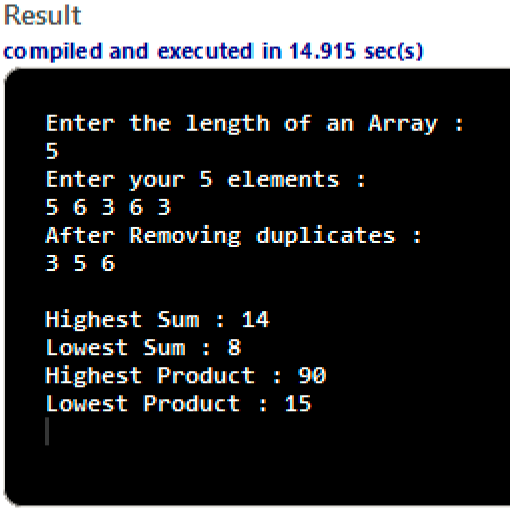
Output:



Result

compiled and executed in 23.88 sec(s)

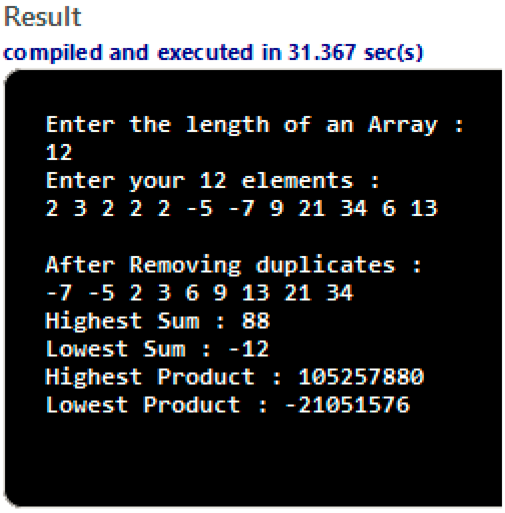
```
Enter the length of an Array :
8
Enter your 8 elements :
2 3 1 -7 8 3 4 5
After Removing duplicates :
-7 1 2 3 4 5 8

Highest Sum : 23
Lowest Sum : -6
Highest Product : 960
Lowest Product : -6720
```



Result

compiled and executed in 14.915 sec(s)

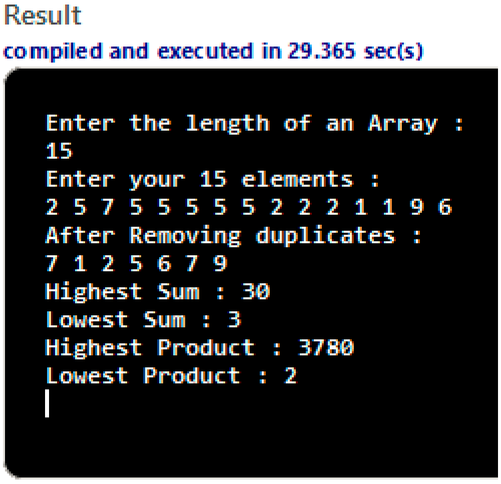
```
Enter the length of an Array :
5
Enter your 5 elements :
5 6 3 6 3
After Removing duplicates :
3 5 6

Highest Sum : 14
Lowest Sum : 8
Highest Product : 90
Lowest Product : 15
```



Result

compiled and executed in 31.367 sec(s)

```
Enter the length of an Array :
12
Enter your 12 elements :
2 3 2 2 2 -5 -7 9 21 34 6 13

After Removing duplicates :
-7 -5 2 3 6 9 13 21 34
Highest Sum : 88
Lowest Sum : -12
Highest Product : 105257880
Lowest Product : -21051576
```



Result

compiled and executed in 29.365 sec(s)

```
Enter the length of an Array :
15
Enter your 15 elements :
2 5 7 5 5 5 5 5 2 2 2 1 1 9 6
After Removing duplicates :
7 1 2 5 6 7 9
Highest Sum : 30
Lowest Sum : 3
Highest Product : 3780
Lowest Product : 2
```

Part-2

if m=3

Sample O/P:

Highest Product= 120 Highest Sum=15

Lowest Product = 6 Lowest Sum=6

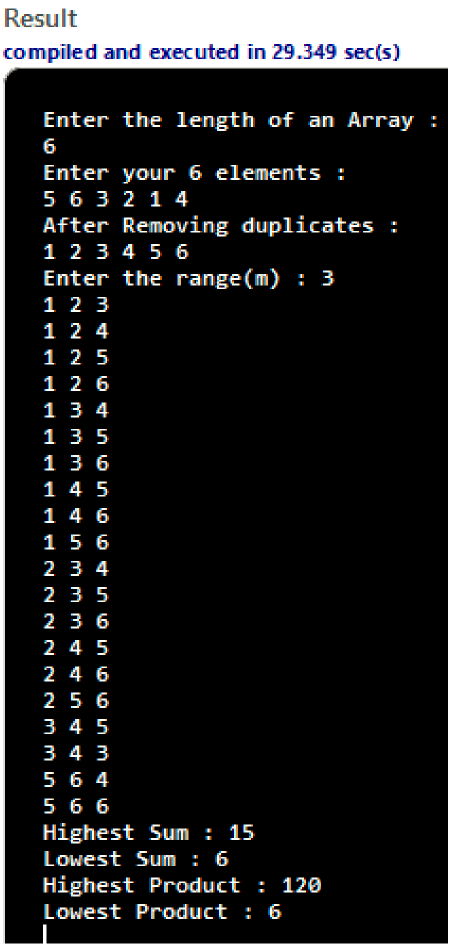
CODE:

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

public class FindingHiLowOfSumPro{

      static ArrayList<Integer> sumlist = new ArrayList<Integer>(5);
      static ArrayList<Integer> prolist = new ArrayList<Integer>(5);
      static void storeSum(int sum){
      sumlist.add(sum);
      }
      static void storePro(int pro){
      prolist.add(pro);
      }

   static void allcombi(int arr[], int data[], int start,
                                  int end, int index, int r)
    {
       int sum = 0;
       int pro = 1;
       if (index == r)
       {
            for (int j=0; j<r; j++){
            System.out.print(data[j]+" ");
            sum = sum + data[j];
            pro = pro * data[j];
            }
            System.out.println(" ");
            storeSum(sum);
            storePro(pro);
            return;
       }

       for (int i=start; i<=end && end-i+1 >= r-index; i++)
       {
            data[index] = arr[i];
            allcombi(arr, data, i+1, end, index+1, r);
       }
    }
      public static int removeDuplicates(int arr[], int n){
      if (n==0 || n==1){
           return n;
      }
```

```
  int[] temp = new int[n];
  int j = 0;
  for (int i=0; i<n-1; i++){
       if (arr[i] != arr[i+1]){
            temp[j++] = arr[i];
       }
  }
  temp[j++] = arr[n-1];
  // Changing original array
  for (int i=0; i<j; i++){
       arr[i] = temp[i];
  }
  return j;
  }
static void findHiLowOfSumPro(int arr[], int n, int range)
{
   int data[]=new int[range];
   allcombi(arr, data, 0, n-1, 0, range);
}

public static void main (String[] args) {
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter the length of an Array : ");
  int n = sc.nextInt();
  int arr[] = new int[n];
  System.out.println("Enter your "+n+" elements :");
  for(int i=0;i<n;i++){
       arr[i] = sc.nextInt();
  }
   Arrays.sort(arr);
  n = removeDuplicates(arr, n);

  System.out.println("\nAfter Removing duplicates : ");
  for (int i=0; i<n; i++)
       System.out.print(+arr[i]+" ");
       System.out.print("Enter the range(m) :");
  int range = sc.nextInt();

  findHiLowOfSumPro(arr, n, range);
  System.out.println("");
   System.out.println("Highest Sum : "+Collections.max(sumlist))
   System.out.println("Lowest Sum : "+Collections.min(sumlist));
```

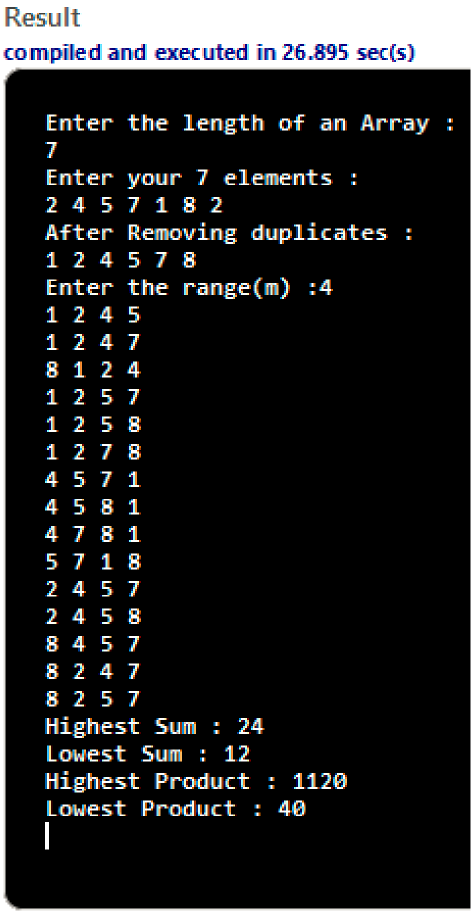
```
        System.out.println("Highest Product :
"+Collections.max(prolist));
        System.out.println("Lowest Product :
"+Collections.min(prolist));
    }
}
```

Output:



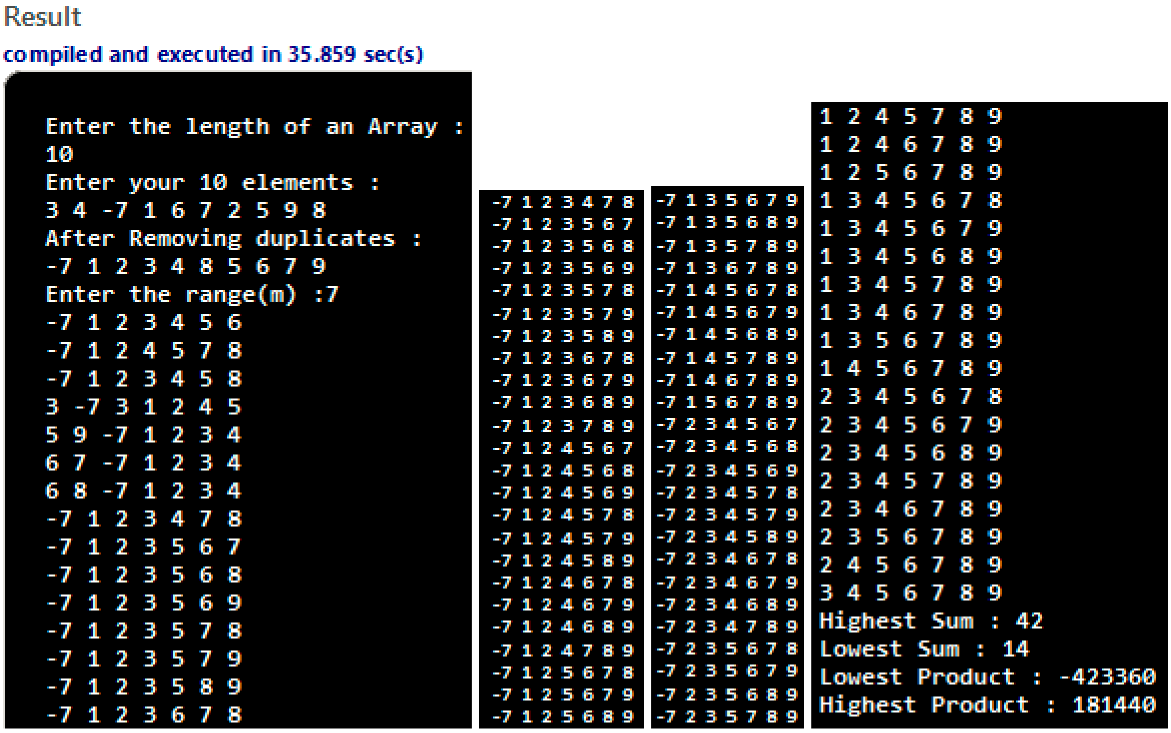
Result
compiled and executed in 29.349 sec(s)

```
Enter the length of an Array :
6
Enter your 6 elements :
5 6 3 2 1 4
After Removing duplicates :
1 2 3 4 5 6
Enter the range(m) : 3
1 2 3
1 2 4
1 2 5
1 2 6
1 3 4
1 3 5
1 3 6
1 4 5
1 4 6
1 5 6
2 3 4
2 3 5
2 3 6
2 4 5
2 4 6
2 5 6
3 4 5
3 4 3
5 6 4
5 6 6
Highest Sum : 15
Lowest Sum : 6
Highest Product : 120
Lowest Product : 6
```



Result
compiled and executed in 26.895 sec(s)

```
Enter the length of an Array :
7
Enter your 7 elements :
2 4 5 7 1 8 2
After Removing duplicates :
1 2 4 5 7 8
Enter the range(m) :4
1 2 4 5
1 2 4 7
8 1 2 4
1 2 5 7
1 2 5 8
1 2 7 8
4 5 7 1
4 5 8 1
4 7 8 1
5 7 1 8
2 4 5 7
2 4 5 8
8 4 5 7
8 2 4 7
8 2 5 7
Highest Sum : 24
Lowest Sum : 12
Highest Product : 1120
Lowest Product : 40
```



Result

compiled and executed in 35.859 sec(s)

```
Enter the length of an Array :
10
Enter your 10 elements :
3 4 -7 1 6 7 2 5 9 8
After Removing duplicates :
-7 1 2 3 4 8 5 6 7 9
Enter the range(m) :7
-7 1 2 3 4 5 6
-7 1 2 4 5 7 8
-7 1 2 3 4 5 8
3 -7 3 1 2 4 5
5 9 -7 1 2 3 4
6 7 -7 1 2 3 4
6 8 -7 1 2 3 4
-7 1 2 3 4 7 8
-7 1 2 3 5 6 7
-7 1 2 3 5 6 8
-7 1 2 3 5 6 9
-7 1 2 3 5 7 8
-7 1 2 3 5 7 9
-7 1 2 3 5 8 9
-7 1 2 3 6 7 8
```

```
-7 1 2 3 4 7 8
-7 1 2 3 5 6 7
-7 1 2 3 5 6 8
-7 1 2 3 5 6 9
-7 1 2 3 5 7 8
-7 1 2 3 5 7 9
-7 1 2 3 5 8 9
-7 1 2 3 6 7 8
-7 1 2 3 6 7 9
-7 1 2 3 6 8 9
-7 1 2 3 7 8 9
-7 1 2 4 5 6 7
-7 1 2 4 5 6 8
-7 1 2 4 5 6 9
-7 1 2 4 5 7 8
-7 1 2 4 5 7 9
-7 1 2 4 5 8 9
-7 1 2 4 6 7 8
-7 1 2 4 6 7 9
-7 1 2 4 6 8 9
-7 1 2 4 7 8 9
-7 1 2 5 6 7 8
-7 1 2 5 6 7 9
-7 1 2 5 6 8 9
```

```
-7 1 3 5 6 7 9
-7 1 3 5 6 8 9
-7 1 3 5 7 8 9
-7 1 3 6 7 8 9
-7 1 4 5 6 7 8
-7 1 4 5 6 7 9
-7 1 4 5 6 8 9
-7 1 4 5 7 8 9
-7 1 4 6 7 8 9
-7 1 5 6 7 8 9
-7 2 3 4 5 6 7
-7 2 3 4 5 6 8
-7 2 3 4 5 6 9
-7 2 3 4 5 7 8
-7 2 3 4 5 7 9
-7 2 3 4 5 8 9
-7 2 3 4 6 7 8
-7 2 3 4 6 7 9
-7 2 3 4 6 8 9
-7 2 3 4 7 8 9
-7 2 3 5 6 7 8
-7 2 3 5 6 7 9
-7 2 3 5 6 8 9
-7 2 3 5 7 8 9
```

```
1 2 4 5 7 8 9
1 2 4 6 7 8 9
1 2 5 6 7 8 9
1 3 4 5 6 7 8
1 3 4 5 6 7 9
1 3 4 5 6 8 9
1 3 4 5 7 8 9
1 3 4 6 7 8 9
1 3 5 6 7 8 9
1 4 5 6 7 8 9
2 3 4 5 6 7 8
2 3 4 5 6 7 9
2 3 4 5 6 8 9
2 3 4 5 7 8 9
2 3 4 6 7 8 9
2 3 5 6 7 8 9
2 4 5 6 7 8 9
3 4 5 6 7 8 9
Highest Sum : 42
Lowest Sum : 14
Lowest Product : -423360
Highest Product : 181440
```

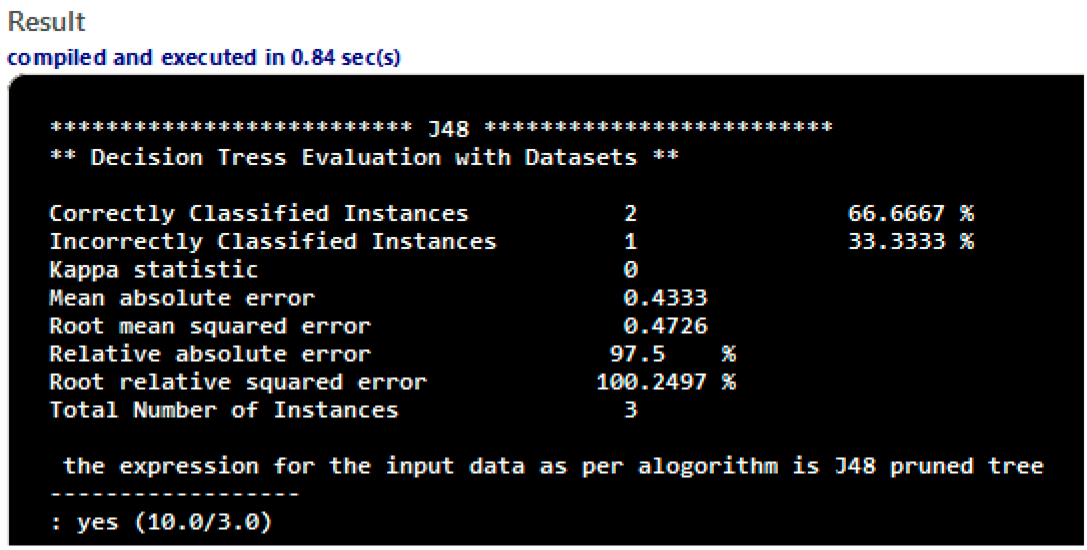
Q2.) Implement Supervised and Unsupervised algorithm

CODE:(Decision Tree)

```
import java.io.IOException;

public class DecisionTree {


    public static final String
TRAINING_DATA_SET_FILENAME="decision-train.arff";
    public static final String
TESTING_DATA_SET_FILENAME="decision-test.arff";


    public static Instances getDataSet(String fileName) throws
IOException {

        int classIdx = 1;
        /** the arffloader to load the arff file */
        ArffLoader loader = new ArffLoader();
        /** load the traing data */
        loader.setSource(DecisionTree.class.getResourceAsStream("/"
+ fileName));

        Instances dataSet = loader.getDataSet();
        /** set the index based on the data given in the arff files
*/
        dataSet.setClassIndex(classIdx);
        return dataSet;
     }

     public static void process() throws Exception {

        Instances trainingDataSet =
getDataSet(TRAINING_DATA_SET_FILENAME);
        Instances testingDataSet =
getDataSet(TESTING_DATA_SET_FILENAME);

        System.out.println("************************** J48
*************************");
        /** Classifier here is Linear Regression */
        Classifier classifier = new J48();

        classifier.buildClassifier(trainingDataSet);

        Evaluation eval = new Evaluation(trainingDataSet);
        eval.evaluateModel(classifier, testingDataSet);
        /** Print the algorithm summary */
        System.out.println("** Decision Tress Evaluation with
Datasets **");
        System.out.println(eval.toSummaryString());
        System.out.print(" the expression for the input data as per
alogorithm is ");
        System.out.println(classifier);
        System.out.println(eval.toMatrixString());
        System.out.println(eval.toClassDetailsString());

        System.out.println("************************** ID3
*************************");
        /** Classifier here is Linear Regression */
        Classifier id3Classifier = new Id3();

        id3Classifier.buildClassifier(trainingDataSet);
        /**
```

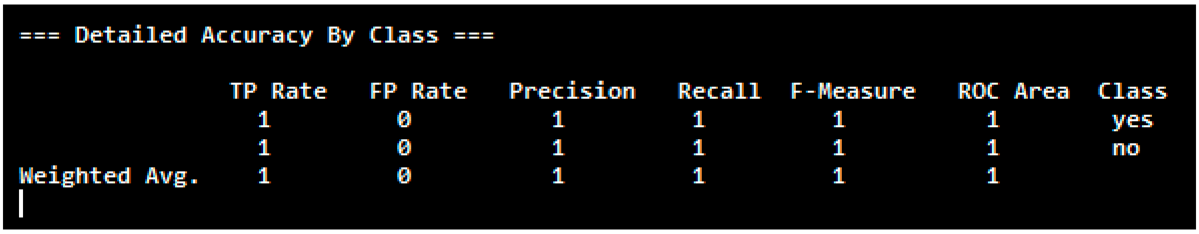
```
          * train the alogorithm with the training data and evaluate
 the
          * algorithm with testing data
          */
         Evaluation evalId3 = new Evaluation(trainingDataSet);
         evalId3.evaluateModel(id3Classifier, testingDataSet);
         /** Print the algorithm summary */
         System.out.println("** Decision Tress Evaluation with
 Datasets **");
         System.out.println(evalId3.toSummaryString());
         System.out.print(" the expression for the input data as per
 alogorithm is ");
         System.out.println(id3Classifier);
         System.out.println(evalId3.toMatrixString());
         System.out.println(evalId3.toClassDetailsString());

      }

 }
```

Output:



Result
compiled and executed in 0.84 sec(s)

```
************************** J48 *************************
** Decision Tress Evaluation with Datasets **

Correctly Classified Instances          2               66.6667 %
Incorrectly Classified Instances        1               33.3333 %
Kappa statistic                         0
Mean absolute error                     0.4333
Root mean squared error                 0.4726
Relative absolute error                97.5     %
Root relative squared error           100.2497 %
Total Number of Instances               3

 the expression for the input data as per alogorithm is J48 pruned tree
------------------
: yes (10.0/3.0)
```



```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0          1         1         1          1        yes
                 1         0          1         1         1          1        no
Weighted Avg.    1         0          1         1         1          1
```

CODE:(KNN)

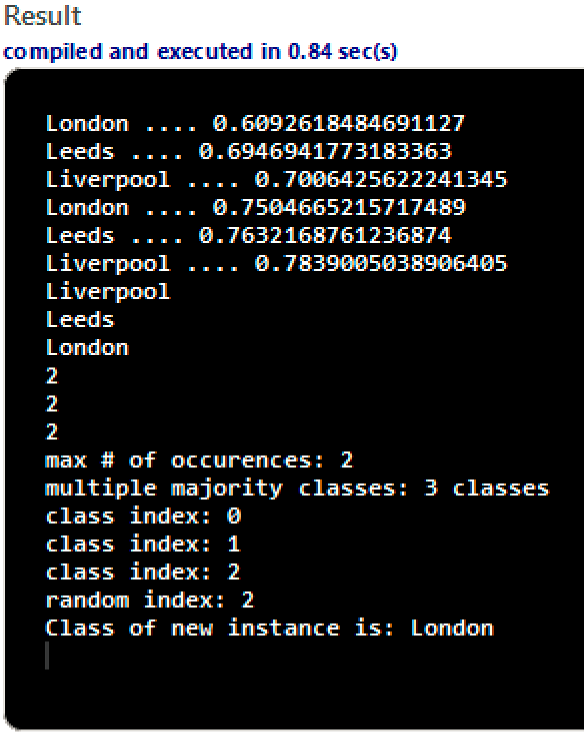
```
import java.util.*;

public class KNN
{
    // the data
    static double[][] instances = {
       {0.35,0.91,0.86,0.42,0.71},
       {0.21,0.12,0.76,0.22,0.92},
       {0.41,0.58,0.73,0.21,0.09},
       {0.71,0.34,0.55,0.19,0.80},
       {0.79,0.45,0.79,0.21,0.44},
       {0.61,0.37,0.34,0.81,0.42},
       {0.78,0.12,0.31,0.83,0.87},
       {0.52,0.23,0.73,0.45,0.78},
       {0.53,0.17,0.63,0.29,0.72},
    };


    private static String findMajorityClass(String[] array)
    {
       //add the String array to a HashSet to get unique String values
       Set<String> h = new HashSet<String>(Arrays.asList(array));
       //convert the HashSet back to array
       String[] uniqueValues = h.toArray(new String[0]);
       //counts for unique strings
       int[] counts = new int[uniqueValues.length];
       // loop thru unique strings and count how many times they appear
in origianl array
       for (int i = 0; i < uniqueValues.length; i++) {
             for (int j = 0; j < array.length; j++) {
                   if(array[j].equals(uniqueValues[i])){
                         counts[i]++;
                   }
             }
       }

       for (int i = 0; i < uniqueValues.length; i++)
             System.out.println(uniqueValues[i]);
```

```
        for (int i = 0; i < counts.length; i++)
            System.out.println(counts[i]);


        int max = counts[0];
        for (int counter = 1; counter < counts.length; counter++) {
            if (counts[counter] > max) {
                max = counts[counter];
            }
        }
        System.out.println("max # of occurences: "+max);

        int freq = 0;
        for (int counter = 0; counter < counts.length; counter++) {
            if (counts[counter] == max) {
                freq++;
            }
        }

        //index of most freq value if we have only one mode
        int index = -1;
        if(freq==1){
            for (int counter = 0; counter < counts.length; counter++) {
                if (counts[counter] == max) {
                    index = counter;
                    break;
                }
            }

            return uniqueValues[index];
        } else{//we have multiple modes
            int[] ix = new int[freq];//array of indices of modes
            System.out.println("multiple majority classes: "+freq+"
classes");
            int ixi = 0;
            for (int counter = 0; counter < counts.length; counter++) {
                if (counts[counter] == max) {
                    ix[ixi] = counter;//save index of each max count
value
                    ixi++; // increase index of ix array
                }
            }

            for (int counter = 0; counter < ix.length; counter++)
                System.out.println("class index: "+ix[counter]);
```

```
         //now choose one at random
         Random generator = new Random();
         //get random number 0 <= rIndex < size of ix
         int rIndex = generator.nextInt(ix.length);
         System.out.println("random index: "+rIndex);
         int nIndex = ix[rIndex];
         //return unique value at that index
         return uniqueValues[nIndex];
   }

}


private static double meanOfArray(double[] m) {
   double sum = 0.0;
   for (int j = 0; j < m.length; j++){
         sum += m[j];
   }
   return sum/m.length;
}



public static void main(String args[]){

   int k = 6;// # of neighbours
   //list to save city data
   List<City> cityList = new ArrayList<City>();

   List<Result> resultList = new ArrayList<Result>();
   // add city data to cityList
   cityList.add(new City(instances[0],"London"));
   cityList.add(new City(instances[1],"Leeds"));
   cityList.add(new City(instances[2],"Liverpool"));
   cityList.add(new City(instances[3],"London"));
   cityList.add(new City(instances[4],"Liverpool"));
   cityList.add(new City(instances[5],"Leeds"));
   cityList.add(new City(instances[6],"London"));
   cityList.add(new City(instances[7],"Liverpool"));
   cityList.add(new City(instances[8],"Leeds"));
   //data about unknown city
   double[] query = {0.65,0.78,0.21,0.29,0.58};
   //find disnaces
   for(City city : cityList){
```

```
             double dist = 0.0;
             for(int j = 0; j < city.cityAttributes.length; j++){

                   dist += Math.pow(city.cityAttributes[j] - query[j],
2) ;
                   //System.out.print(city.cityAttributes[j]+" ");

             }
             double distance = Math.sqrt( dist );
             resultList.add(new Result(distance,city.cityName));
             //System.out.println(distance);
       }

       //System.out.println(resultList);
       Collections.sort(resultList, new DistanceComparator());
       String[] ss = new String[k];
       for(int x = 0; x < k; x++){
             System.out.println(resultList.get(x).cityName+ " .... " +
resultList.get(x).distance);
             //get classes of k nearest instances (city names) from the
list into an array
             ss[x] = resultList.get(x).cityName;
       }
       String majClass = findMajorityClass(ss);
       System.out.println("Class of new instance is: "+majClass);

    }//end main

    //simple class to model instances (features + class)
    static class City {
       double[] cityAttributes;
       String cityName;
       public City(double[] cityAttributes, String cityName){
             this.cityName = cityName;
             this.cityAttributes = cityAttributes;
       }
    }
    //simple class to model results (distance + class)
    static class Result {
       double distance;
       String cityName;
       public Result(double distance, String cityName){
             this.cityName = cityName;
             this.distance = distance;
       }
    }
    //simple comparator class used to compare results via distances
    static class DistanceComparator implements Comparator<Result> {
       @Override
       public int compare(Result a, Result b) {
             return a.distance < b.distance ? -1 : a.distance ==
b.distance ? 0 : 1;
       }
    }

}
```

Output:



Result

compiled and executed in 0.84 sec(s)

```
London .... 0.6092618484691127
Leeds .... 0.6946941773183363
Liverpool .... 0.7006425622241345
London .... 0.7504665215717489
Leeds .... 0.7632168761236874
Liverpool .... 0.7839005038906405
Liverpool
Leeds
London
2
2
2
max # of occurences: 2
multiple majority classes: 3 classes
class index: 0
class index: 1
class index: 2
random index: 2
Class of new instance is: London
```