Normalization

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

public class Normalization {
  public static void main(String[] args) throws Exception {
    String input = args.length>0? args[0] : "normalization_input.csv";
    List<Double> values = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader(input))) {
      String line;
      // skip header if non-numeric first line
      while ((line = br.readLine()) != null) {
        line = line.trim();
        if (line.isEmpty()) continue;
        try {
          double v = Double.parseDouble(line);
          values.add(v);
        } catch (NumberFormatException e) { /* assume header, skip */ }
      }
    }
    if (values.isEmpty()) {
      System.out.println("No numeric data found in " + input);
      return;
    }
    double min = values.stream().mapToDouble(Double::doubleValue).min().getAsDouble();
    double max = values.stream().mapToDouble(Double::doubleValue).max().getAsDouble();
    double mean = values.stream().mapToDouble(Double::doubleValue).average().getAsDouble();
    double sd = Math.sqrt(values.stream().mapToDouble(v ->
(v-mean)*(v-mean)).sum()/values.size());
    System.out.println("Min-Max Normalization:");
    for (double v : values) {
      double norm = (max==min)? 0.0 : (v-min)/(max-min);
      System.out.printf(Locale.US, "%.5f%n", norm);
    }
    System.out.println("\nZ-score Normalization:");
    for (double v : values) {
      double z = (sd==0)? 0.0 : (v-mean)/sd;
      System.out.printf(Locale.US, "%.5f%n", z);
    }
    System.out.println("\nDecimal Scaling Normalization:");
    double maxAbs = values.stream().mapToDouble(Math::abs).max().getAsDouble();
    int j = 0; while (maxAbs >= 1) { maxAbs /= 10.0; j++; }
    for (double v : values) {
      double ds = v / Math.pow(10, j);
      System.out.printf(Locale.US, "%.5f%n", ds);
```

```
    }
  }
}
```

	Standard
1	Num of items sold in last five months
2	25
3	30
4	22
5	24

```
Min-Max Normalization:
0.37500
1.00000
0.00000
0.25000
```

```
Z-score Normalization:
1.58339
-0.34421
-1.17033
-0.06884

Decimal Scaling Normalization:
0.22000
0.15000
0.12000
0.16000
```

Knime

Minmax

Rows: 4 | Columns: 1

#	RowID	Num of items sold in last five months Number (Float)
1	Row0	1
2	Row1	0.3
3	Row2	0
4	Row3	0.4

Z-score

#	RowID	Num of items sold in last five months Number (Float)
1	Row0	1.371
2	Row1	-0.298
3	Row2	-1.014
4	Row3	-0.06

Decimal Scaling

#	RowID	Num of items sold in last five months Number (Float)
1	Row0	0.22
2	Row1	0.15
3	Row2	0.12
4	Row3	0.16

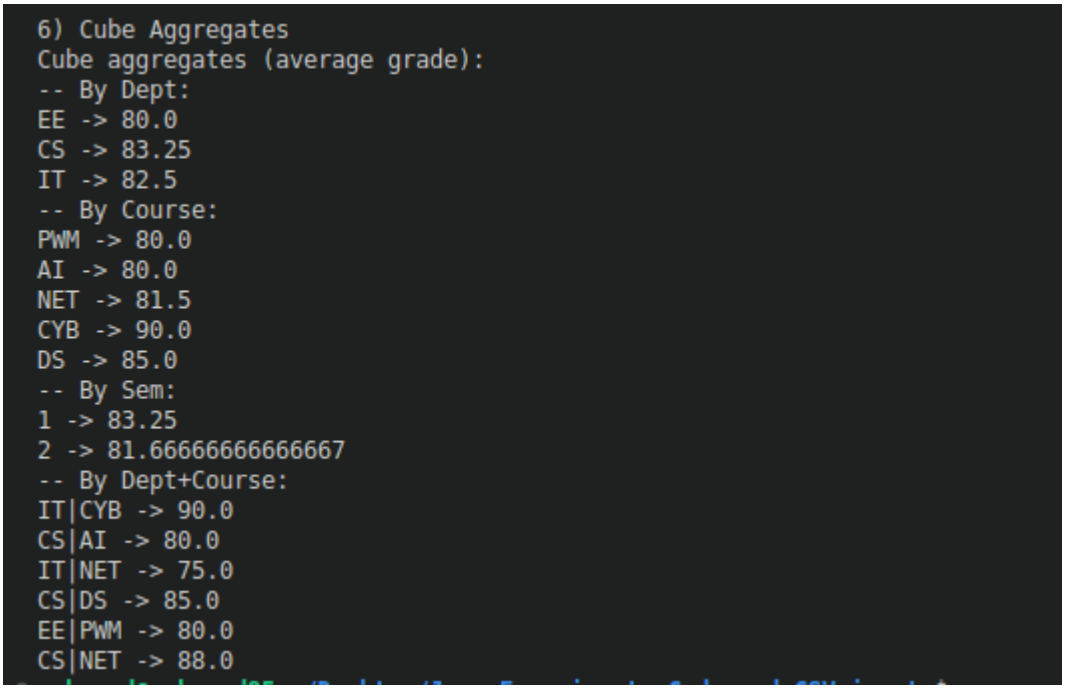
OLAP operations

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

public class OLAPDemo {
  static class Rec {
    String student, dept, course;
    int sem;
    double grade;
    Rec(String s, String d, String c, int sem, double g){
      student=s; dept=d; course=c; this.sem=sem; grade=g;
    }
    public String toString(){
      return student+" | "+dept+" | "+course+" | "+sem+" | "+grade;
    }
  }

  public static List<Rec> readCsv(String path) throws IOException {
    List<Rec> list = new ArrayList<>();
    try(BufferedReader br = new BufferedReader(new FileReader(path))){
      String line = br.readLine(); // skip header
      while((line=br.readLine())!=null){
        if(line.trim().isEmpty()) continue;
        String[] t = line.split(",");
        if(t.length < 5) continue;
        String student = t[0], dept = t[1], course = t[2];
        int sem = Integer.parseInt(t[3]);
        double grade = Double.parseDouble(t[4]);
        list.add(new Rec(student, dept, course, sem, grade));
      }
    }
    return list;
  }

  // Roll-up: average grade per department
  public static Map<String, Double> rollupDeptAvg(List<Rec> rows){
    return rows.stream().collect(
      Collectors.groupingBy(r -> r.dept, Collectors.averagingDouble(r -> r.grade))
    );
  }

  // Drill-down: within a department, show avg per course
  public static Map<String, Double> drilldownDeptToCourseAvg(List<Rec> rows, String dept){
    return rows.stream()
      .filter(r -> r.dept.equalsIgnoreCase(dept))
```

```
        .collect(Collectors.groupingBy(r -> r.course, Collectors.averagingDouble(r -> r.grade)));
  }

  // Slice: filter by department
  public static List<Rec> sliceByDept(List<Rec> rows, String dept){
     return rows.stream()
        .filter(r -> r.dept.equalsIgnoreCase(dept))
        .collect(Collectors.toList());
  }

  // Dice: filter by multiple conditions
  public static List<Rec> dice(List<Rec> rows, String dept, Integer sem){
     return rows.stream()
        .filter(r -> (dept==null || r.dept.equalsIgnoreCase(dept)))
        .filter(r -> (sem==null || r.sem==sem))
        .collect(Collectors.toList());
  }

  // Pivot: dept x sem average
  public static Map<String, Map<Integer, Double>> pivotDeptSem(List<Rec> rows){
     return rows.stream().collect(
        Collectors.groupingBy(r -> r.dept,
           Collectors.groupingBy(r -> r.sem, Collectors.averagingDouble(r -> r.grade)))
     );
  }

  // Cube: all possible aggregates
  public static void cubeAggregates(List<Rec> rows){
     System.out.println("Cube aggregates (average grade):");

     Map<String, Double> byDept = rows.stream()
        .collect(Collectors.groupingBy(r->r.dept, Collectors.averagingDouble(r->r.grade)));
     System.out.println("-- By Dept:");
     byDept.forEach((k,v)->System.out.println(k+" -> "+v));

     Map<String, Double> byCourse = rows.stream()
        .collect(Collectors.groupingBy(r->r.course, Collectors.averagingDouble(r->r.grade)));
     System.out.println("-- By Course:");
     byCourse.forEach((k,v)->System.out.println(k+" -> "+v));

     Map<Integer, Double> bySem = rows.stream()
        .collect(Collectors.groupingBy(r->r.sem, Collectors.averagingDouble(r->r.grade)));
     System.out.println("-- By Sem:");
     bySem.forEach((k,v)->System.out.println(k+" -> "+v));

     Map<String, Double> byDeptCourse = rows.stream()
        .collect(Collectors.groupingBy(r -> r.dept+"|"+r.course,
Collectors.averagingDouble(r->r.grade)));
```

```
        System.out.println("-- By Dept+Course:");
        byDeptCourse.forEach((k,v)->System.out.println(k+" -> "+v));
    }

    public static void main(String[] args) throws Exception{
        String csv = "olap_input.csv";
        if(args.length>0) csv = args[0];
        List<Rec> rows = readCsv(csv);

        System.out.println("All rows:");
        rows.forEach(System.out::println);
        System.out.println();

        System.out.println("1) Roll-up (Avg grade per Department)");
        rollupDeptAvg(rows).forEach((k,v)->System.out.println(k+" -> "+v));
        System.out.println();

        System.out.println("2) Drill-down (Dept CS -> Avg per course)");
        drilldownDeptToCourseAvg(rows, "CS").forEach((k,v)->System.out.println(k+" -> "+v));
        System.out.println();

        System.out.println("3) Slice (Dept=CS)");
        sliceByDept(rows, "CS").forEach(System.out::println);
        System.out.println();

        System.out.println("4) Dice (Dept=CS, Sem=2)");
        dice(rows, "CS", 2).forEach(System.out::println);
        System.out.println();

        System.out.println("5) Pivot (Dept x Sem Avg)");
        pivotDeptSem(rows).forEach((d,m)->{
            System.out.println(d);
            m.forEach((s,a)->System.out.println("  Sem "+s+" -> "+a));
        });
        System.out.println();

        System.out.println("6) Cube Aggregates");
        cubeAggregates(rows);
    }
}
```

```
All rows:
s1 | CS | DS | 1 | 85.0
s2 | CS | AI | 1 | 78.0
s3 | CS | NET | 2 | 88.0
s4 | IT | CYB | 1 | 90.0
s5 | IT | NET | 2 | 75.0
s6 | CS | AI | 2 | 82.0
s7 | EE | PWM | 1 | 80.0

1) Roll-up (Avg grade per Department)
EE -> 80.0
CS -> 83.25
IT -> 82.5

2) Drill-down (Dept CS -> Avg per course)
AI -> 80.0
NET -> 88.0
DS -> 85.0

3) Slice (Dept=CS)
s1 | CS | DS | 1 | 85.0
s2 | CS | AI | 1 | 78.0
s3 | CS | NET | 2 | 88.0
s6 | CS | AI | 2 | 82.0

4) Dice (Dept=CS, Sem=2)
s3 | CS | NET | 2 | 88.0
s6 | CS | AI | 2 | 82.0

5) Pivot (Dept x Sem Avg)
EE
  Sem 1 -> 80.0
CS
  Sem 1 -> 81.5
  Sem 2 -> 85.0
IT
  Sem 1 -> 90.0
  Sem 2 -> 75.0
```


6) Cube Aggregates
Cube aggregates (average grade):
-- By Dept:
EE -> 80.0
CS -> 83.25
IT -> 82.5
-- By Course:
PWM -> 80.0
AI -> 80.0
NET -> 81.5
CYB -> 90.0
DS -> 85.0
-- By Sem:
1 -> 83.25
2 -> 81.66666666666667
-- By Dept+Course:
IT|CYB -> 90.0
CS|AI -> 80.0
IT|NET -> 75.0
CS|DS -> 85.0
EE|PWM -> 80.0
CS|NET -> 88.0

T-weight D-weight

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

public class TWeightDWeight {
   public static void main(String[] args) throws Exception {
      String input = args.length>0? args[0] : "tweight_input.csv";
      List<String[]> rows = new ArrayList<>();
      try (BufferedReader br = new BufferedReader(new FileReader(input))) {
         String line; boolean headerSkipped=false;
         while ((line = br.readLine()) != null) {
            if (!headerSkipped) { headerSkipped=true; continue; }
            if (line.trim().isEmpty()) continue;
            String[] parts = line.split("\");".replace("\\","\\\\").split(",");
         }
      } catch (IOException e) {
         // fallback manual parse
      }
      // We'll read generically
      rows.clear();
      try (BufferedReader br = new BufferedReader(new FileReader(input))) {
         String line = br.readLine(); // header
         while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty()) continue;
            String[] p = line.split(",");
            if (p.length < 3) continue;
            rows.add(new String[] {p[0].trim(), p[1].trim(), p[2].trim()});
         }
      }
      // pivot computation
      LinkedHashMap<String, LinkedHashMap<String, Integer>> pivot = new LinkedHashMap<>();
      LinkedHashMap<String, Integer> rowSum = new LinkedHashMap<>();
      LinkedHashMap<String, Integer> colSum = new LinkedHashMap<>();
      int grand=0;
      for (String[] r: rows) {
         String cls=r[0], place=r[1]; int cnt=Integer.parseInt(r[2]);
         pivot.putIfAbsent(cls, new LinkedHashMap<>());
         pivot.get(cls).put(place, cnt);
         rowSum.put(cls, rowSum.getOrDefault(cls,0)+cnt);
         colSum.put(place, colSum.getOrDefault(place,0)+cnt);
         grand += cnt;
      }
      System.out.printf("%-15s %-10s %-10s %-10s\n","Class","Count","T-weight%","D-weight%");
      for (String cls: pivot.keySet()) {
```

```
            for (String place: pivot.get(cls).keySet()) {
                int val = pivot.get(cls).get(place);
                double t = (rowSum.get(cls)>0)? 100.0*val/rowSum.get(cls):0;
                double d = (colSum.get(place)>0)? 100.0*val/colSum.get(place):0;
                System.out.printf("%-15s %-10d %-9.2f%% %-9.2f%%\n", cls+"->"+place, val, t, d);
            }
        }
        System.out.println("Grand total: " + grand);
    }
}
```

	Standard	Standard	Standard	Standard	Standard
1	City	Temp	Rainfall	Humidity	CO2
2	Delhi	92.12	154.44	49.69	1.77
3	Mumbai	88.45	200.32	78.22	1.92
4	Chennai	95.21	120.54	65.78	1.85
5	Kolkata	89.67	180.22	70.45	1.88
6	Bangalore	85.33	210.11	60.14	1.73

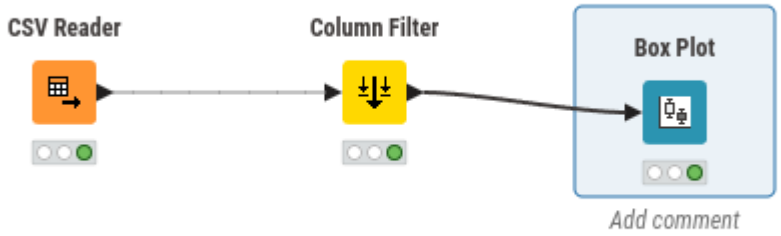
```
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$ cd "/ho
City-wise T-weight and D-weight:
City          Temp%        Rainfall%    Humidity%    CO2%
Delhi         30.91        51.82        16.67        0.59
Mumbai        23.98        54.30        21.20        0.52
Chennai       33.60        42.54        23.21        0.65
Kolkata       26.20        52.66        20.59        0.55
Bangalore     23.88        58.80        16.83        0.48

D-weight (column percentage contribution):
City          Temp%        Rainfall%    Humidity%    CO2%
Delhi         20.44        17.84        15.32        19.34
Mumbai        19.62        23.14        24.12        20.98
Chennai       21.12        13.93        20.28        20.22
Kolkata       19.89        20.82        21.73        20.55
Bangalore     18.93        24.27        18.55        18.91

Grand Total of all values: 1649.8400000000004
```

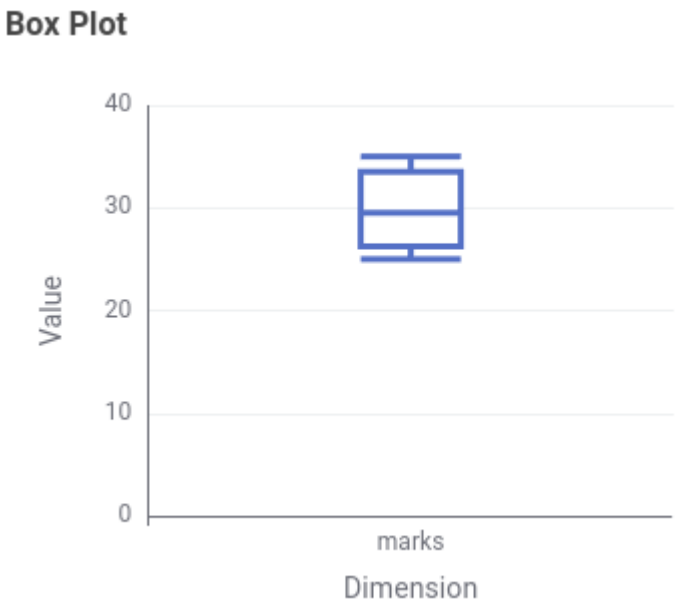
5 Number Summary

```
import java.io.*;import java.util.*;
public class FiveNumSummary {
   public static void main(String[] args) throws Exception {
      String input = args.length>0? args[0] : "fivenum_input.csv";
      List<Double> a = new ArrayList<>();
      try (BufferedReader br = new BufferedReader(new FileReader(input))) {
         String line; br.readLine(); // header
         while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty()) continue;
            a.add(Double.parseDouble(line.trim()));
         }
      }
      Collections.sort(a);
      int n = a.size();
      double min = a.get(0), max=a.get(n-1);
      double median = median(a,0,n-1);
      double q1 = median(a,0,(n-1)/2);
      double q3 = median(a,(n+1)/2,n-1);
      double iqr = q3-q1;
      double lower = q1 - 1.5*iqr, upper = q3 + 1.5*iqr;
      System.out.printf(Locale.US, "Min=%.2f Q1=%.2f Median=%.2f Q3=%.2f Max=%.2f\n",
min,q1,median,q3,max);
      System.out.println("IQR="+iqr+" Lower="+lower+" Upper="+upper);
      System.out.print("Outliers: ");
      boolean any=false;
      for (double v: a) if (v<lower || v>upper) { System.out.print(v+" "); any=true; }
      if (!any) System.out.print("None");
      System.out.println();
   }
   static double median(List<Double> a,int s,int e){
      int len = e - s + 1;
      int mid = s + len/2;
      if (len%2==0) return (a.get(mid-1)+a.get(mid))/2.0;
      else return a.get(mid);
   }
}
```

	Standard
1	marks
2	25
3	26
4	27
5	29
6	30
7	32
8	34
9	35

```
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$ cd "/h
Min=25.00 Q1=26.50 Median=29.50 Q3=33.00 Max=35.00
IQR=6.5 Lower=16.75 Upper=42.75
Outliers: None
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$
```



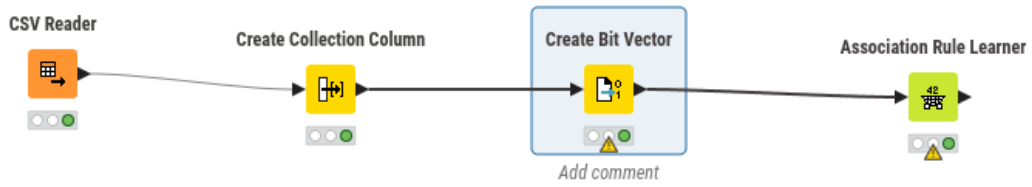


6 Apriori

```
import java.io.*;import java.util.*;
public class Apriori {
   static int minSupport = 2;
   public static void main(String[] args) throws Exception {
      String input = args.length>0? args[0] : "apriori_input.csv";
      if (args.length>1) minSupport = Integer.parseInt(args[1]);
      List<Set<String>> transactions = new ArrayList<>();
      try (BufferedReader br = new BufferedReader(new FileReader(input))) {
         String line;
         while ((line = br.readLine()) != null) {
            line=line.trim();
            if (line.isEmpty()) continue;
            String[] items = line.split(",");
            Set<String> set = new HashSet<>();
            for (String it: items) if (!it.trim().isEmpty()) set.add(it.trim());
            if (!set.isEmpty()) transactions.add(set);
         }
      }
      Map<Set<String>, Integer> freq = new HashMap<>();
      // frequent 1-itemsets
      Map<String,Integer> counts = new HashMap<>();
      for (Set<String> t: transactions) for (String it: t) counts.put(it, counts.getOrDefault(it,0)+1);
      List<Set<String>> Lk = new ArrayList<>();
      for (var e: counts.entrySet()) if (e.getValue()>=minSupport) Lk.add(Set.of(e.getKey()));
      System.out.println("Frequent 1-itemsets: " + Lk);
      // naive generation of 2-itemsets
      Map<Set<String>,Integer> two = new HashMap<>();
      for (Set<String> t: transactions) {
         List<String> list = new ArrayList<>(t);
         for (int i=0;i<list.size();i++) for (int j=i+1;j<list.size();j++) {
            Set<String> s = new TreeSet<>(Arrays.asList(list.get(i), list.get(j)));
            two.put(s, two.getOrDefault(s,0)+1);
         }
      }
      System.out.println("Frequent 2-itemsets (minSupport="+minSupport+"):");
      for (var e: two.entrySet()) if (e.getValue()>=minSupport) System.out.println(e.getKey()+" ->
"+e.getValue());
   }
}
```

```
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$ cd "
riments_Code_and_CSV_inputs/" && javac Apriori.java && java Apriori
Frequent 1-itemsets: [[bread], [butter], [milk], [juice]]
Frequent 2-itemsets (minSupport=2):
[bread, butter] -> 2
[bread, milk] -> 3
[butter, milk] -> 3
[bread, juice] -> 2
```

```
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$ cd "/hom
Total Transactions: 9
Minimum Support: 30.0% (3 transactions)

For k = 1:
Itemset         Frequency       Support
[Egg]           3               33%
[Butter]                6               67%
[Cheese]                3               33%
[Milk]          6               67%
[Bread]         6               67%

For k = 2:
Itemset         Frequency       Support
[Butter, Milk]          4               44%
[Bread, Butter]         4               44%
[Bread, Cheese]         3               33%
[Bread, Milk]           3               33%

mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$
```



▶ 1: Frequent itemsets/Association rules Flow Variables

Rows: 2 | Columns: 2

#	RowID	Support(0-1): Number (Float)	Items Collection (Set)
1	item set 0	1	[1]
2	item set 1	1	[0]

Assignment 7: Association Rule

```
import java.io.*;import java.util.*;
public class AssociationRule {
   public static void main(String[] args) throws Exception {
      String input = args.length>0? args[0] : "assoc_input.csv";
      List<Set<String>> transactions = new ArrayList<>();
      try (BufferedReader br = new BufferedReader(new FileReader(input))) {
         String line;
         while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty()) continue;
            Set<String> s = new HashSet<>();
            for (String it: line.split(",")) if (!it.trim().isEmpty()) s.add(it.trim());
            transactions.add(s);
         }
      }
      int N = transactions.size();
      Map<Set<String>, Integer> support = new HashMap<>();
      // all itemsets of size 1 and 2
      for (Set<String> t: transactions) {
         for (String a: t) support.put(Set.of(a), support.getOrDefault(Set.of(a),0)+1);
         List<String> list = new ArrayList<>(t);
         for (int i=0;i<list.size();i++) for (int j=i+1;j<list.size();j++) {
            Set<String> s = new TreeSet<>(Arrays.asList(list.get(i), list.get(j)));
            support.put(s, support.getOrDefault(s,0)+1);
         }
      }
      // generate rules from frequent 2-itemsets
      System.out.println("Association rules (support%, confidence%):");
      for (var e: support.entrySet()) if (e.getKey().size()==2) {
         Set<String> itemset = e.getKey();
         int sup = e.getValue();
         List<String> items = new ArrayList<>(itemset);
         Set<String> A = Set.of(items.get(0));
         Set<String> B = Set.of(items.get(1));
         int supA = support.getOrDefault(A,0);
         int supB = support.getOrDefault(B,0);
         double confAtoB = supA==0?0: (double)sup/supA;
         double confBtoA = supB==0?0: (double)sup/supB;
         System.out.printf("%s => %s  (%.1f%%, %.1f%%)%n", A, B, 100.0*sup/N,
100.0*confAtoB);
         System.out.printf("%s => %s  (%.1f%%, %.1f%%)%n", B, A, 100.0*sup/N,
100.0*confBtoA);
      }
   }
}
```

	Standard Milk	Standard Bread	Standard Butter	Standard Egg	Standard Cheese
1					
2	1	1	0	0	1
3	0	1	1	0	0
4	0	1	1	1	0
5	1	0	1	1	0
6	1	0	1	0	0
7	0	1	0	0	1
8	1	0	0	1	0
9	1	1	1	0	1
10	1	1	1	0	0

```
Total Transactions: 9
Minimum Support: 30.0% (3 transactions)
Minimum Confidence: 60.0%

For k = 1:
Itemset         Frequency       Support
[Egg]           3               33%
[Butter]                6               67%
[Cheese]                3               33%
[Milk]          6               67%
[Bread]         6               67%

For k = 2:
Itemset         Frequency       Support
[Butter, Milk]          4               44%
[Bread, Butter]         4               44%
[Bread, Cheese]         3               33%
[Bread, Milk]           3               33%

=== Association Rules (Confidence >= 60.0%) ===
[Butter] => [Milk] (Support: 44%, Confidence: 67%)
[Milk] => [Butter] (Support: 44%, Confidence: 67%)
[Bread] => [Butter] (Support: 44%, Confidence: 67%)
[Butter] => [Bread] (Support: 44%, Confidence: 67%)
[Cheese] => [Bread] (Support: 33%, Confidence: 100%)
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$
```

Rows: 2 | Columns: 2

#	RowID	Support(0-1): Number (Float)	Items Collection (Set)
1	item se	1	[1]
2	item se	1	[0]

Assignment 8: Correlation

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

public class Correlation {
  public static void main(String[] args) throws Exception {
    String input = args.length > 0 ? args[0] : "correlation_input.csv";

    List<Double> studyHours = new ArrayList<>();
    List<Double> marks = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(input))) {
      String header = br.readLine(); // skip header
      String line;
      while ((line = br.readLine()) != null) {
        if (line.trim().isEmpty()) continue;
        String[] parts = line.split(",");
        if (parts.length < 3) continue;
        studyHours.add(Double.parseDouble(parts[1].trim()));
        marks.add(Double.parseDouble(parts[2].trim()));
      }
    }

    int n = Math.min(studyHours.size(), marks.size());
    if (n == 0) {
      System.out.println("No valid data found.");
      return;
    }

    double meanX =
studyHours.stream().mapToDouble(Double::doubleValue).average().getAsDouble();
    double meanY = marks.stream().mapToDouble(Double::doubleValue).average().getAsDouble();

    double numerator = 0, denomX = 0, denomY = 0;
    for (int i = 0; i < n; i++) {
      numerator += (studyHours.get(i) - meanX) * (marks.get(i) - meanY);
      denomX += Math.pow(studyHours.get(i) - meanX, 2);
      denomY += Math.pow(marks.get(i) - meanY, 2);
    }

    double r = numerator / Math.sqrt(denomX * denomY);

    System.out.printf(Locale.US, "Mean of Study Hours = %.2f\n", meanX);
    System.out.printf(Locale.US, "Mean of Marks = %.2f\n", meanY);
    System.out.printf(Locale.US, "Pearson Correlation (r) = %.4f\n", r);

    if (r > 0.7)
```

```
        System.out.println("Strong positive correlation: More study hours → higher marks.");
    else if (r > 0.3)
        System.out.println("Moderate positive correlation.");
    else if (r > 0)
        System.out.println("Weak positive correlation.");
    else
        System.out.println("No positive correlation.");
  }
}
```

```
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$ cd "/home
Mean of Study Hours = 5.00
Mean of Marks = 62.00
Pearson Correlation (r) = 0.9882
Strong positive correlation: More study hours → higher marks.
mahemud@mahemud95:~/Desktop/Java_Experiments_Code_and_CSV_inputs$
```

: 1 | Columns: 5

#	RowID	First column name String	Second column name String	Correlation value Number (Float)
1	Row0	study_hours	marks	0.988

Table | Statistics

p value Number (Float)	Degrees of freedom Number (Integer)
0.002	3

Assignment 9: K means Clustering

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

public class KMeans {
  public static void main(String[] args) throws Exception {
    String input = args.length > 0 ? args[0] : "kmeans_input.csv";

    // Read data
    List<String> points = new ArrayList<>();
    List<Double> values = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader(input))) {
      String header = br.readLine(); // skip header
      String line;
      while ((line = br.readLine()) != null) {
        String[] parts = line.split(",");
        points.add(parts[0].trim());
        values.add(Double.parseDouble(parts[1].trim()));
      }
    }

    // Define K and initial centroids
    int k = 2;
    double[] centroids = {0.35, 0.95};
    double[] newCentroids = new double[k];
    int[] cluster = new int[values.size()];

    boolean changed = true;
    int iteration = 1;

    while (changed) {
      System.out.println("\nIteration " + iteration + ":");
      changed = false;

      // Assign points to nearest centroid
      for (int i = 0; i < values.size(); i++) {
        double minDist = Double.MAX_VALUE;
        int assignedCluster = -1;
        for (int j = 0; j < k; j++) {
          double dist = Math.abs(values.get(i) - centroids[j]);
          if (dist < minDist) {
            minDist = dist;
            assignedCluster = j;
          }
        }
        if (cluster[i] != assignedCluster) {
```

```
                cluster[i] = assignedCluster;
                changed = true;
            }
        }

        // Display assignments
        for (int j = 0; j < k; j++) {
            System.out.print("Cluster " + (j + 1) + ": ");
            for (int i = 0; i < values.size(); i++) {
                if (cluster[i] == j)
                    System.out.print(points.get(i) + "(" + values.get(i) + ") ");
            }
            System.out.println();
        }

        // Recalculate centroids
        for (int j = 0; j < k; j++) {
            double sum = 0;
            int count = 0;
            for (int i = 0; i < values.size(); i++) {
                if (cluster[i] == j) {
                    sum += values.get(i);
                    count++;
                }
            }
            newCentroids[j] = count > 0 ? sum / count : centroids[j];
        }

        System.out.println("New centroids: " + Arrays.toString(newCentroids));

        // Update centroids
        for (int j = 0; j < k; j++) {
            if (centroids[j] != newCentroids[j]) {
                centroids[j] = newCentroids[j];
            }
        }

        iteration++;
    }

    System.out.println("\nFinal Cluster Assignments:");
    for (int i = 0; i < points.size(); i++) {
        System.out.println(points.get(i) + " (" + values.get(i) + ") → Cluster " + (cluster[i] + 1));
    }
  }
}
```

kmeans_input.csv

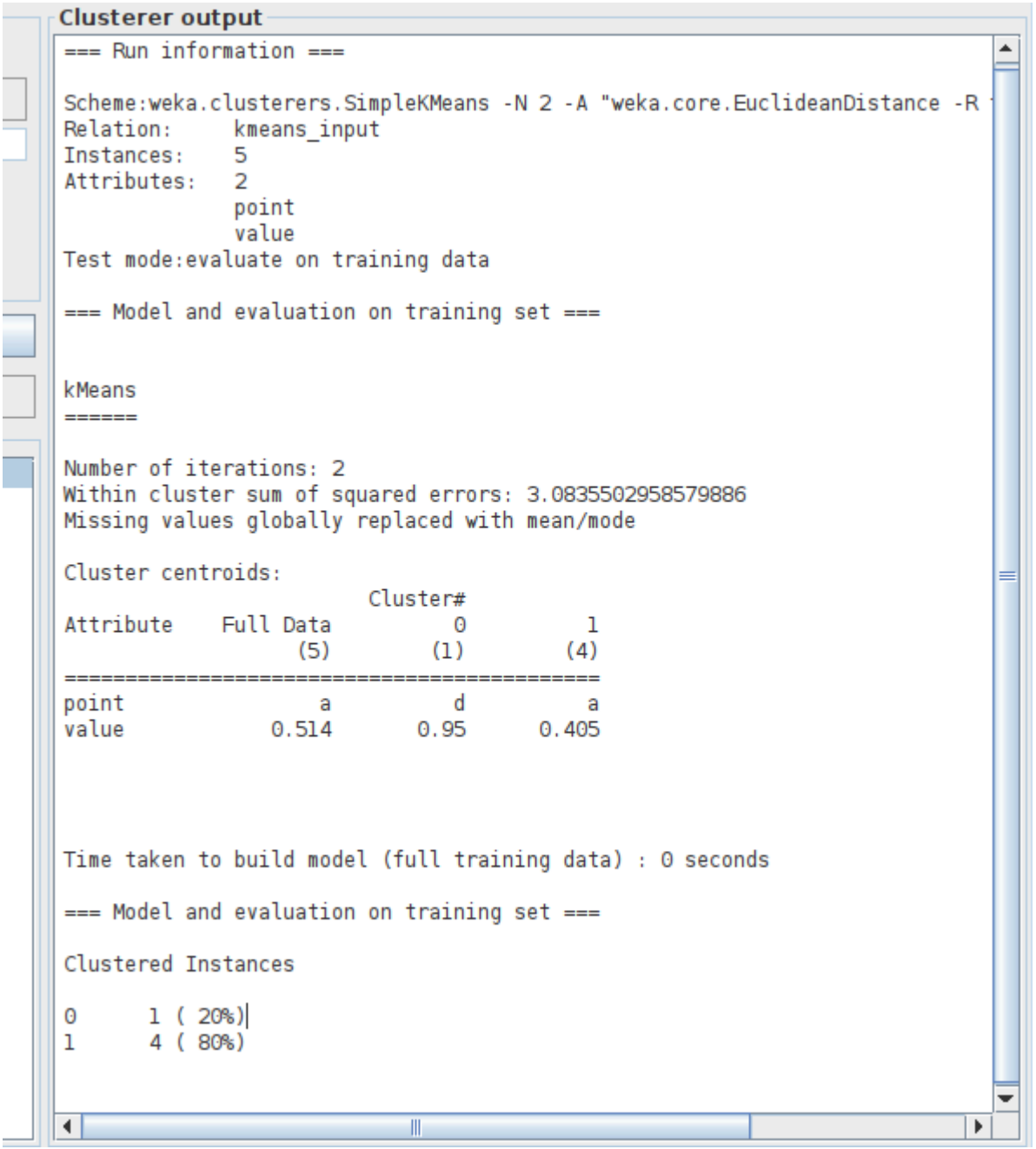
```
point,value
a,0.42
b,0.25
c,0.30
d,0.85
e,0.55
```

```
Iteration 1:
Cluster 1: a(0.42) b(0.35) c(0.3) e(0.55)
Cluster 2: d(0.95)
New centroids: [0.405, 0.95]

Iteration 2:
Cluster 1: a(0.42) b(0.35) c(0.3) e(0.55)
Cluster 2: d(0.95)
New centroids: [0.405, 0.95]

Final Cluster Assignments:
a (0.42) → Cluster 1
b (0.35) → Cluster 1
c (0.3) → Cluster 1
d (0.95) → Cluster 2
e (0.55) → Cluster 1
```

Rows: 5 | Columns: 3

#	RowID	point String	value Number (Float)	Cluster String
1	Row0	a	0.42	cluster_1
2	Row1	b	0.35	cluster_1
3	Row2	c	0.3	cluster_1
4	Row3	d	0.95	cluster_0
5	Row4	e	0.55	cluster_1

Weka


Clusterer output
=== Run information ===
Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R
Relation: kmeans_input
Instances: 5
Attributes: 2
point
value
Test mode:evaluate on training data
=== Model and evaluation on training set ===
kMeans
======
Number of iterations: 2
Within cluster sum of squared errors: 3.0835502958579886
Missing values globally replaced with mean/mode
Cluster centroids:
Cluster#
Attribute Full Data 0 1
(5) (1) (4)
==========
point a d a
value 0.514 0.95 0.405
Time taken to build model (full training data) : 0 seconds
=== Model and evaluation on training set ===
Clustered Instances
0 1 (20%)
1 4 (80%)

Assignment 10: Hierarchical clustering

```
#include <bits/stdc++.h>
using namespace std;

vector<double> splitCSVLine(const string& line) {
    vector<double> values;
    stringstream ss(line);
    string token;
    while (getline(ss, token, ',')) {
        if (!token.empty())
            values.push_back(stod(token));
    }
    return values;
}

vector<vector<double>> readCSV(const string& filename) {
    vector<vector<double>> data;
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Error: Cannot open file " << filename << endl;
        exit(1);
    }

    string line;
    while (getline(file, line)) {
        if (line.empty()) continue;
        data.push_back(splitCSVLine(line));
    }

    file.close();
    return data;
}

double euclideanDistance(const vector<double>& a, const vector<double>& b) {
    double sum = 0.0;
    for (size_t i = 0; i < a.size(); ++i)
        sum += (a[i] - b[i]) * (a[i] - b[i]);
    return sqrt(sum);
}

vector<vector<double>> computeDistanceMatrix(const vector<vector<double>>& data) {
    int n = data.size();
    vector<vector<double>> dist(n, vector<double>(n, 0.0));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            dist[i][j] = euclideanDistance(data[i], data[j]);
        }
    }
    return dist;
}
```

```
double clusterDistance(const vector<int>& c1, const vector<int>& c2,
                       const vector<vector<double>>& dist, int method) {
    vector<double> dists;
    for (int i : c1) {
        for (int j : c2) {
            if (i != j)
                dists.push_back(dist[max(i, j)][min(i, j)]);
        }
    }

    if (method == 1)
        return *min_element(dists.begin(), dists.end());
    else if (method == 2)
        return *max_element(dists.begin(), dists.end());
    else
        return accumulate(dists.begin(), dists.end(), 0.0) / dists.size();
}

int main() {
    string filename;
    cout << "Enter CSV file name (e.g. data.csv): ";
    cin >> filename;

    vector<vector<double>> data = readCSV(filename);
    int n = data.size();
    if (n == 0) {
        cerr << "Error: No data found in file." << endl;
        return 1;
    }

    cout << "\nChoose Linkage Method:\n";
    cout << "1. Single Linkage\n";
    cout << "2. Complete Linkage\n";
    cout << "3. Average Linkage\n";
    cout << "Enter choice (1/2/3): ";
    int choice;
    cin >> choice;

    vector<vector<double>> dist = computeDistanceMatrix(data);

    vector<vector<int>> clusters;
    vector<string> names;
    for (int i = 0; i < n; ++i) {
        clusters.push_back({i});
        names.push_back(string(1, 'A' + i));
    }

    cout << "\nInitial Distance Matrix:\n";
    cout << fixed << setprecision(2);
    for (int i = 0; i < n; i++) {
        cout << setw(6) << names[i];
        for (int j = 0; j < i; j++) cout << setw(8) << dist[i][j];
        cout << endl;
```

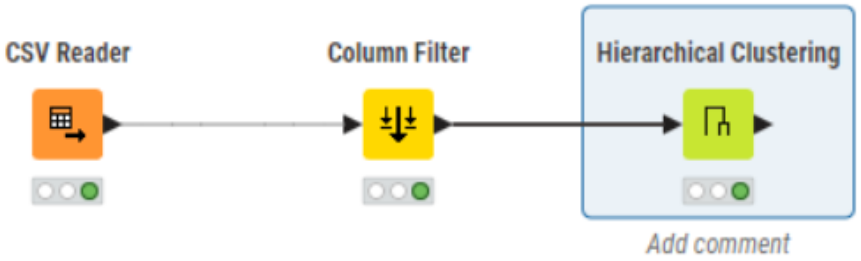
```
    }

    cout << "\n--- Agglomerative Hierarchical Clustering ---\n";
    cout << "(Linkage method: ";
    if (choice == 1) cout << "Single";
    else if (choice == 2) cout << "Complete";
    else cout << "Average";
    cout << ")\n\n";

    int step = 1;

    while (clusters.size() > 1) {
        double minDist = DBL_MAX;
        int x = -1, y = -1;

        for (int i = 0; i < clusters.size(); ++i) {
            for (int j = i + 1; j < clusters.size(); ++j) {
                double d = clusterDistance(clusters[i], clusters[j], dist, choice);
                if (d < minDist) {
                    minDist = d;
                    x = i;
                    y = j;
                }
            }
        }

        cout << "Step " << step++ << ": Merge (" << names[x] << ") and (" << names[y]
             << ") --> Distance = " << minDist << endl;

        clusters[x].insert(clusters[x].end(), clusters[y].begin(), clusters[y].end());
        clusters.erase(clusters.begin() + y);

        names[x] = "(" + names[x] + names[y] + ")";
        names.erase(names.begin() + y);

        cout << "Current Clusters: ";
        for (auto& nm : names) cout << nm << "  ";
        cout << "\n-------------------------------------------\n";
    }

    cout << "\n Final Cluster: " << names[0] << endl;
    return 0;
}
```

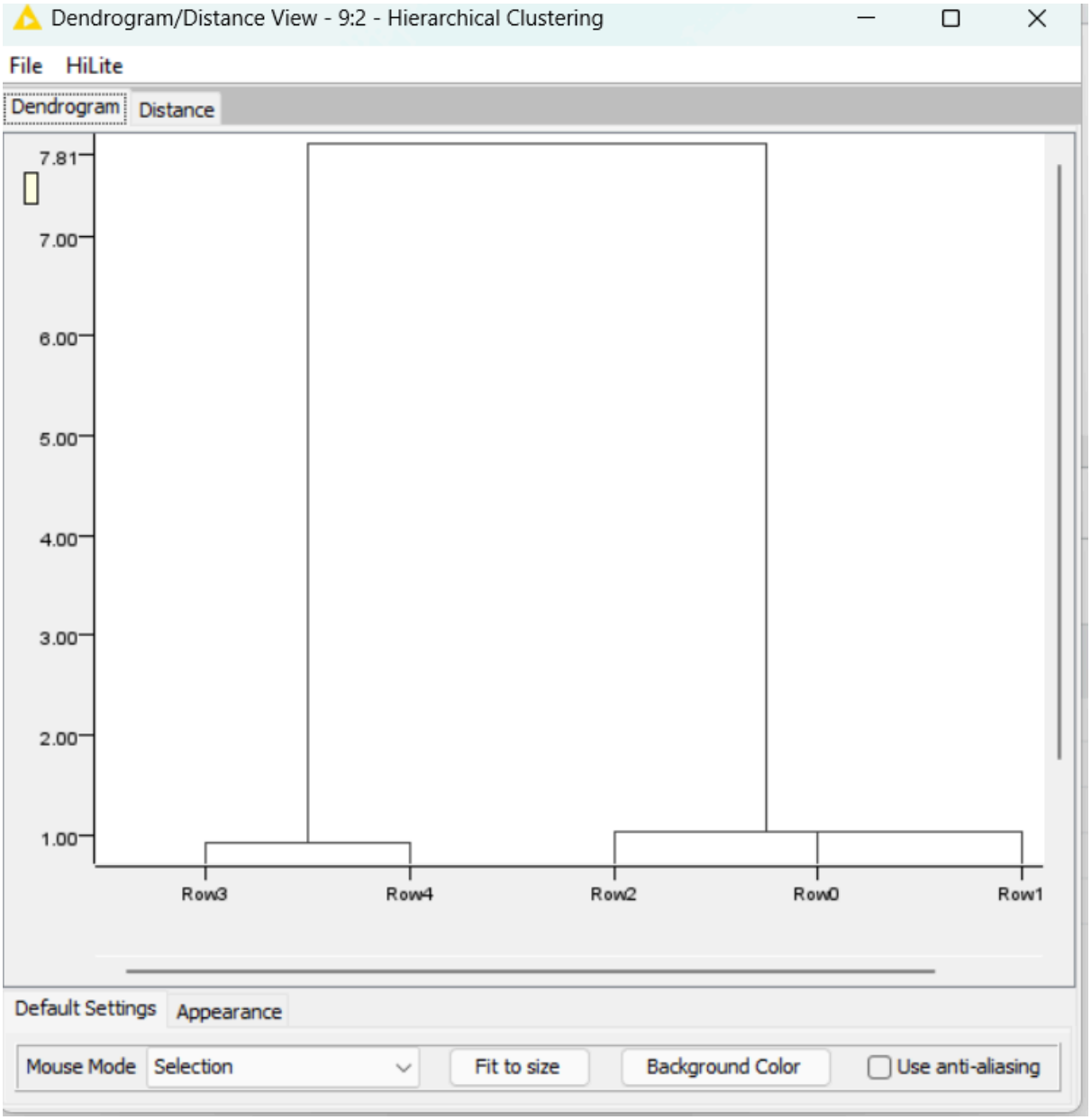
1	1
2	1.5
3	2
8	8
8	9

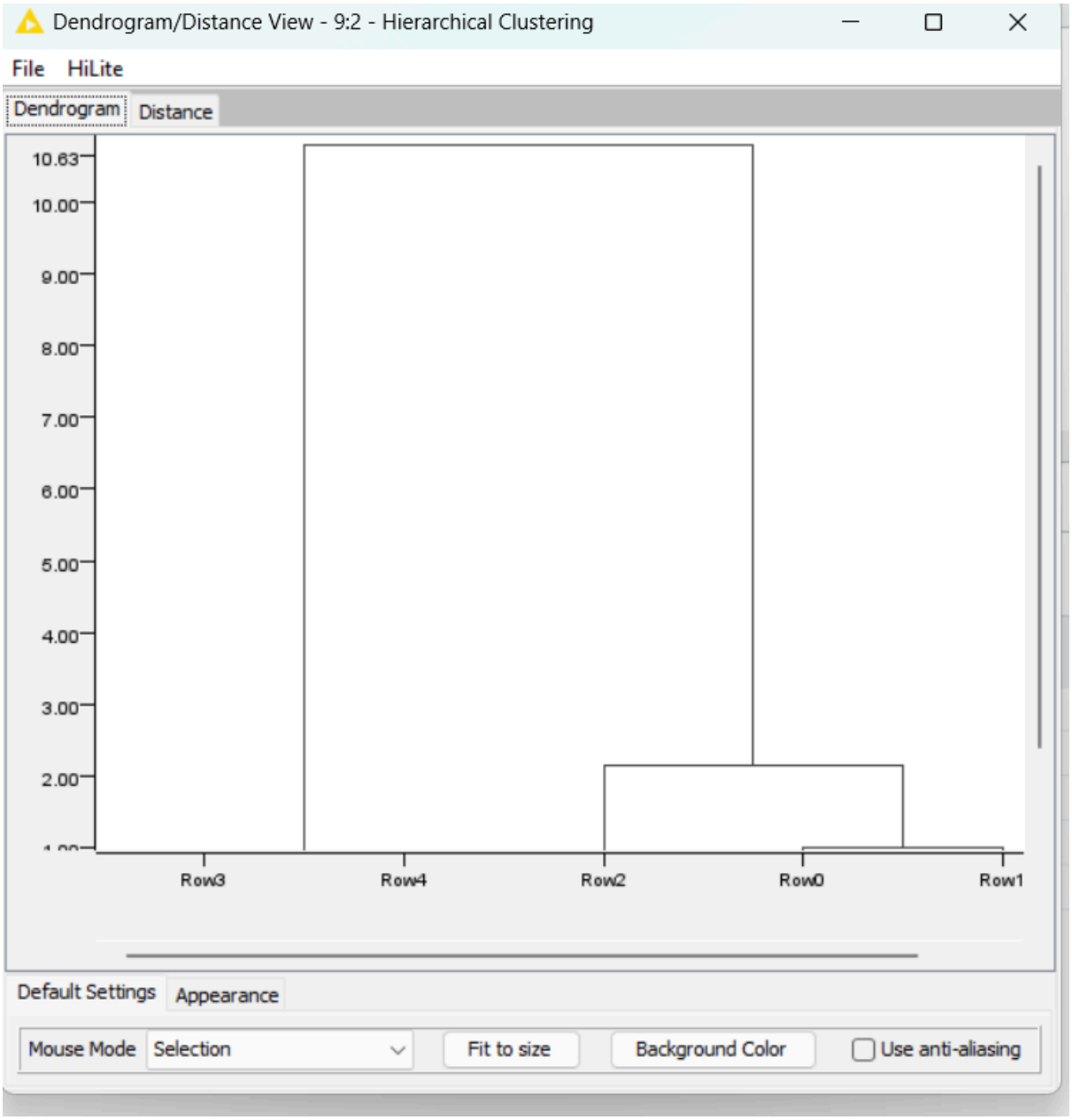
```
Enter CSV file name (e.g. data.csv): single.csv

Choose Linkage Method:
1. Single Linkage
2. Complete Linkage
3. Average Linkage
Enter choice (1/2/3): 1

Initial Distance Matrix:
     A
     B    1.12
     C    2.24    1.12
     D    9.90    8.85    7.81
     E   10.63    9.60    8.60    1.00

--- Agglomerative Hierarchical Clustering ---
(Linkage method: Single)

Step 1: Merge (D) and (E) --> Distance = 1.00
Current Clusters: A  B  C  (DE)
-------------------------------------------
Step 2: Merge (A) and (B) --> Distance = 1.12
Current Clusters: (AB)  C  (DE)
-------------------------------------------
Step 3: Merge ((AB)) and (C) --> Distance = 1.12
Current Clusters: ((AB)C)  (DE)
-------------------------------------------
Step 4: Merge (((AB)C)) and ((DE)) --> Distance = 7.81
Current Clusters: (((AB)C)(DE))
-------------------------------------------

 Final Cluster: (((AB)C)(DE))
```

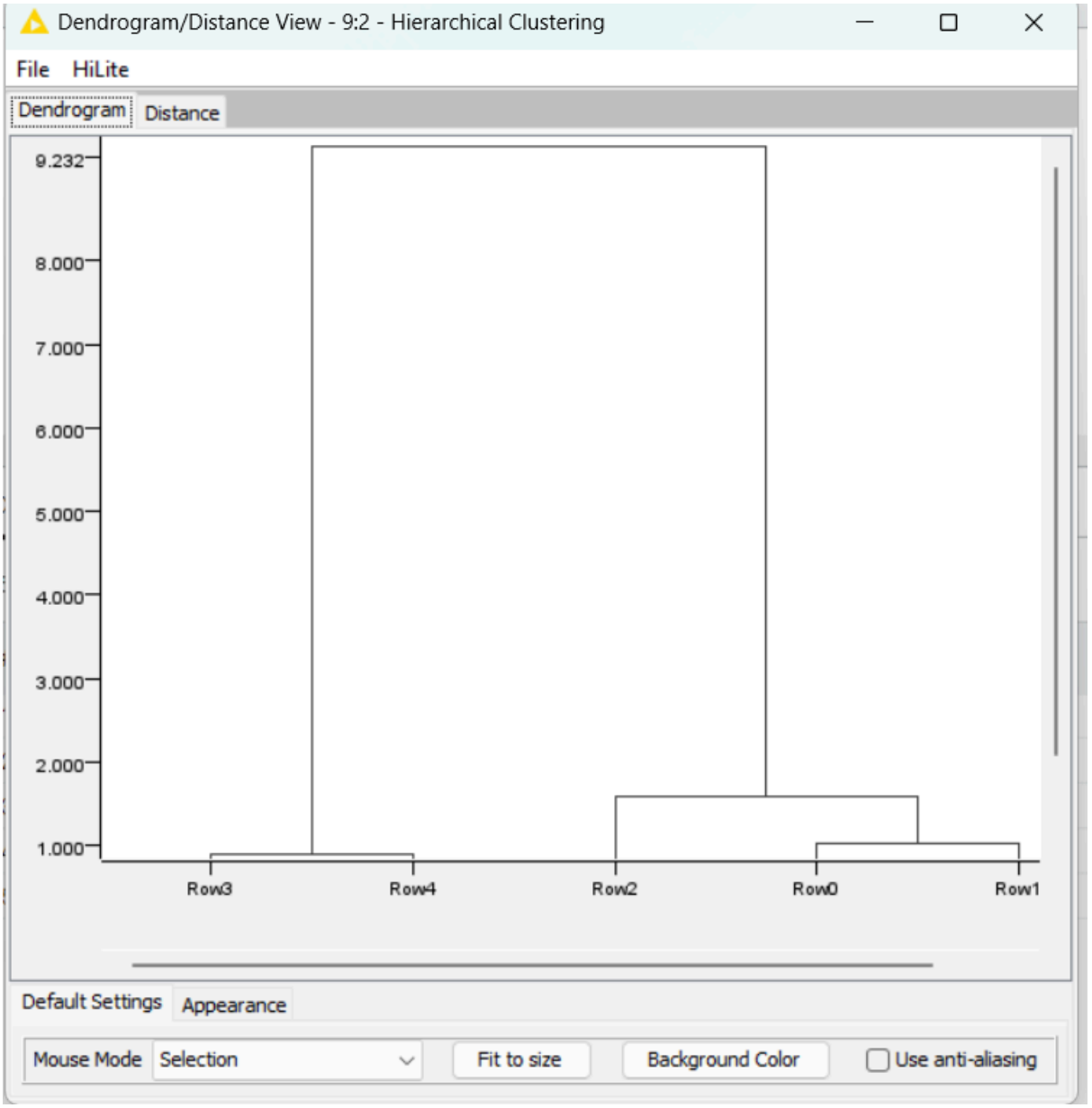
```
Choose Linkage Method:
1. Single Linkage
2. Complete Linkage
3. Average Linkage
Enter choice (1/2/3): 2

Initial Distance Matrix:
     A
     B    1.12
     C    2.24    1.12
     D    9.90    8.85    7.81
     E   10.63    9.60    8.60    1.00

--- Agglomerative Hierarchical Clustering ---
(Linkage method: Complete)

Step 1: Merge (D) and (E) --> Distance = 1.00
Current Clusters: A  B  C  (DE)
------------------------------------------
Step 2: Merge (A) and (B) --> Distance = 1.12
Current Clusters: (AB)  C  (DE)
------------------------------------------
Step 3: Merge ((AB)) and (C) --> Distance = 2.24
Current Clusters: ((AB)C)  (DE)
------------------------------------------
Step 4: Merge (((AB)C)) and ((DE)) --> Distance = 10.63
Current Clusters: (((AB)C)(DE))
------------------------------------------

 Final Cluster: (((AB)C)(DE))
```

```
Enter CSV file name (e.g. data.csv): single.csv

Choose Linkage Method:
1. Single Linkage
2. Complete Linkage
3. Average Linkage
Enter choice (1/2/3): 3

Initial Distance Matrix:
     A
     B    1.12
     C    2.24    1.12
     D    9.90    8.85    7.81
     E   10.63    9.60    8.60    1.00

--- Agglomerative Hierarchical Clustering ---
(Linkage method: Average)

Step 1: Merge (D) and (E) --> Distance = 1.00
Current Clusters: A  B  C  (DE)
------------------------------------------
Step 2: Merge (A) and (B) --> Distance = 1.12
Current Clusters: (AB)  C  (DE)
------------------------------------------
Step 3: Merge ((AB)) and (C) --> Distance = 1.68
Current Clusters: ((AB)C)  (DE)
------------------------------------------
Step 4: Merge (((AB)C)) and ((DE)) --> Distance = 9.23
Current Clusters: (((AB)C)(DE))
------------------------------------------

 Final Cluster: (((AB)C)(DE))
```


CSV Reader
Column Filter
Hierarchical Clustering
Add comment


Dendrogram/Distance View - 9:2 - Hierarchical Clustering
File
HiLite
Dendrogram
Distance
7.81
7.00
6.00
5.00
4.00
3.00
2.00
1.00
Row3
Row4
Row2
Row0
Row1
Default Settings
Appearance
Mouse Mode
Selection
Fit to size
Background Color
Use anti-aliasing

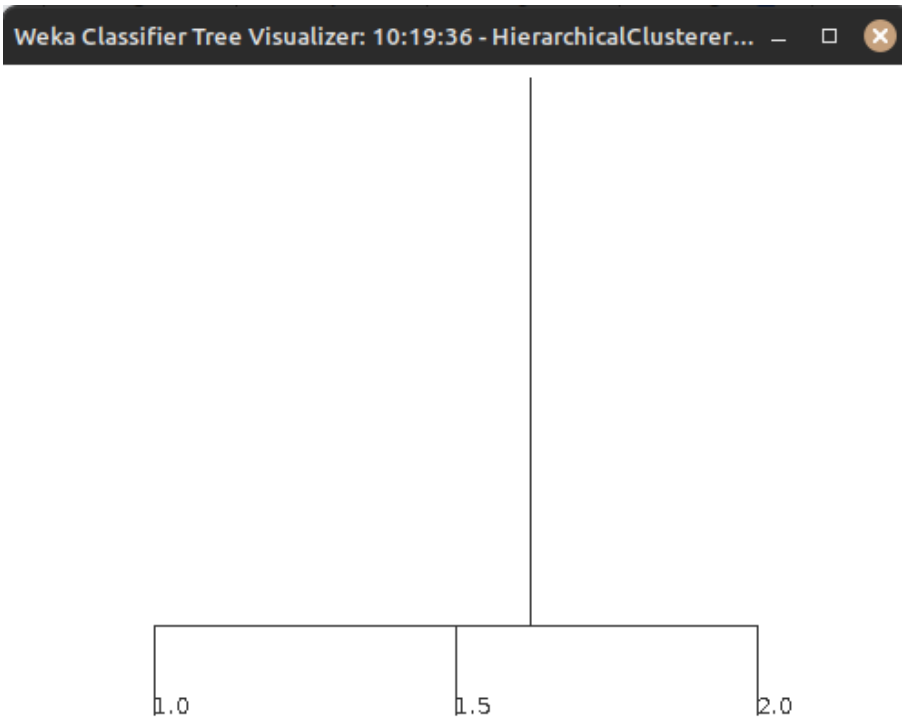

Dendrogram/Distance View - 9:2 - Hierarchical Clustering
File
HiLite
Dendrogram
Distance
10.63
10.00
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
Row3
Row4
Row2
Row0
Row1
Default Settings
Appearance
Mouse Mode
Selection
Fit to size
Background Color
Use anti-aliasing


Dendrogram/Distance View - 9:2 - Hierarchical Clustering
File
HiLite
Dendrogram
Distance
9.232
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
Row3
Row4
Row2
Row0
Row1
Default Settings
Appearance
Mouse Mode
Selection
Fit to size
Background Color
Use anti-aliasing

Weka

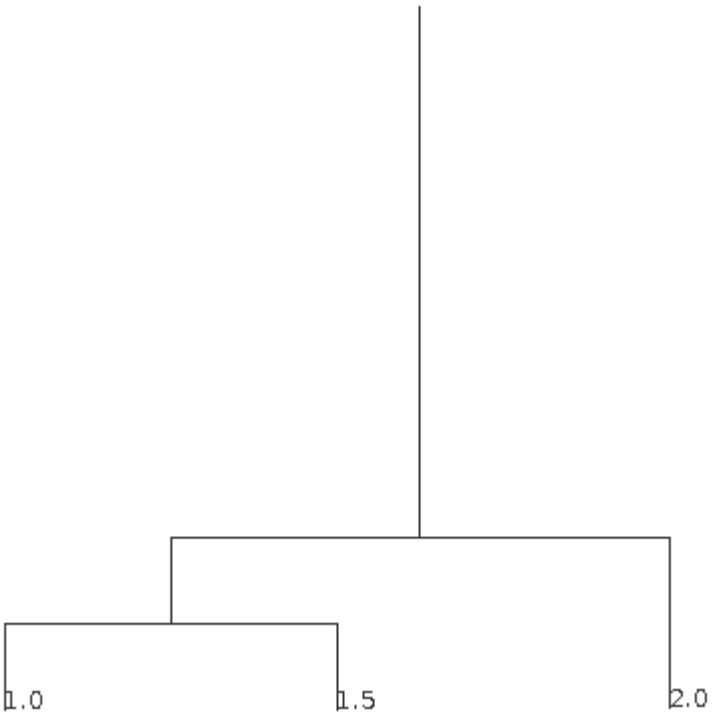
Single

```
Scheme:weka.clusterers.HierarchicalClusterer -N 2 -L SINGLE -P
Relation:     hierarchical
Instances:    5
Attributes:   2
              X
              Y
Test mode:evaluate on training data

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

Cluster 0
((1.0:0.15593,1.5:0.15593):0,2.0:0.15593)

Cluster 1
(8.0:0.125,9.0:0.125)


Time taken to build model (full training data) : 0 seconds

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

Clustered Instances

0      3 ( 60%)
1      2 ( 40%)
```



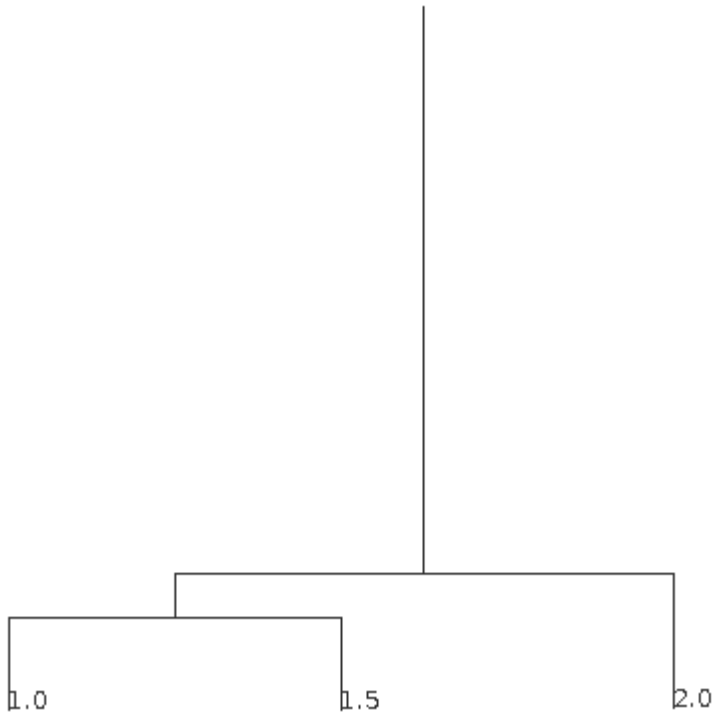
Complete

```
Scheme:weka.clusterers.HierarchicalClusterer -N 2 -L COMPLETE -P -A
Relation:     hierarchical
Instances:    5
Attributes:   2
              X
              Y
Test mode:evaluate on training data

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

Cluster 0
((1.0:0.15593,1.5:0.15593):0.15593,2.0:0.31186)

Cluster 1
(8.0:0.125,9.0:0.125)



Time taken to build model (full training data) : 0 seconds

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

Clustered Instances

0      3 ( 60%)
1      2 ( 40%)
```



Average

```
Scheme:weka.clusterers.HierarchicalClusterer -N 2 -L AVERAGE -P -
Relation:     hierarchical
Instances:    5
Attributes:   2
              X
              Y
Test mode:evaluate on training data

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

Cluster 0
((1.0:0.15593,1.5:0.15593):0.07797,2.0:0.2339)

Cluster 1
(8.0:0.125,9.0:0.125)



Time taken to build model (full training data) : 0 seconds

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

Clustered Instances

0      3 ( 60%)
1      2 ( 40%)
```



Assignment 11: Density Based Clustering

```
#include <bits/stdc++.h>
using namespace std;

struct Point {
   double x;
   int cluster;
   string type;
};

double distance(Point a, Point b) {
   return fabs(a.x - b.x);
}

vector<Point> loadCSV(string filename) {
   vector<Point> data;
   ifstream file(filename);
   string line;
   while (getline(file, line)) {
      if (line.empty()) continue;
      data.push_back({stod(line), -1, ""});
   }
   return data;
}

vector<int> regionQuery(const vector<Point> &points, int idx, double eps) {
   vector<int> neighbors;
   for (int i = 0; i < points.size(); i++) {
      if (distance(points[idx], points[i]) <= eps)
         neighbors.push_back(i);
   }
   return neighbors;
}

void expandCluster(vector<Point> &points, int idx, int clusterId, double eps, int minPts) {
   vector<int> neighbors = regionQuery(points, idx, eps);
   points[idx].cluster = clusterId;

   for (int i = 0; i < neighbors.size(); i++) {
      int nIdx = neighbors[i];
      if (points[nIdx].cluster == -1) {
         points[nIdx].cluster = clusterId;
         vector<int> newNeighbors = regionQuery(points, nIdx, eps);
         if (newNeighbors.size() >= minPts)
            neighbors.insert(neighbors.end(), newNeighbors.begin(), newNeighbors.end());
      } else if (points[nIdx].cluster == 0) {
         points[nIdx].cluster = clusterId;
      }
   }
}
```

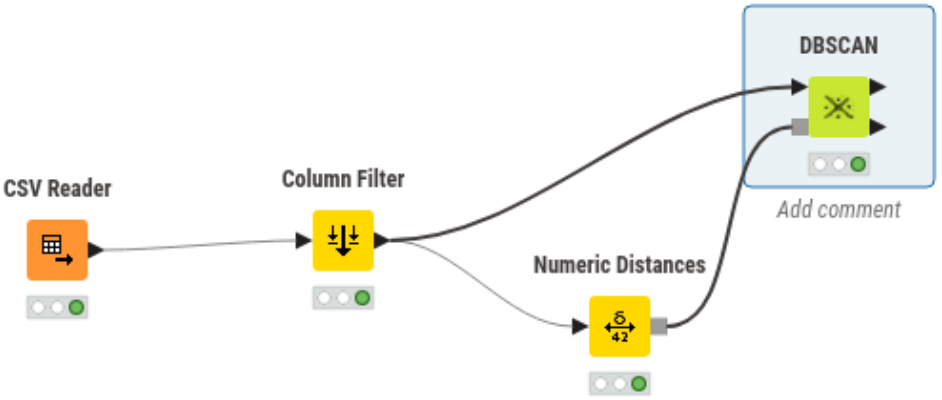
```
void DBSCAN(vector<Point> &points, double eps, int minPts) {
    int clusterId = 0;

    cout << "\n--- Distance Matrix ---\n";
    for (int i = 0; i < points.size(); i++) {
        for (int j = 0; j < points.size(); j++) {
            cout << fixed << setprecision(2) << distance(points[i], points[j]) << "\t";
        }
        cout << endl;
    }

    for (int i = 0; i < points.size(); i++) {
        if (points[i].cluster != -1) continue;

        vector<int> neighbors = regionQuery(points, i, eps);

        if (neighbors.size() < minPts) {
            points[i].cluster = 0; // noise
        } else {
            clusterId++;
            expandCluster(points, i, clusterId, eps, minPts);
        }
    }

    for (int i = 0; i < points.size(); i++) {
        vector<int> neighbors = regionQuery(points, i, eps);
        if (neighbors.size() >= minPts)
            points[i].type = "Core";
        else if (points[i].cluster == 0)
            points[i].type = "Noise";
        else
            points[i].type = "Border";
    }
}

int main() {
    string filename;
    double eps;
    int minPts;

    cout << "Enter CSV filename: ";
    cin >> filename;
    cout << "Enter eps (neighborhood radius): ";
    cin >> eps;
    cout << "Enter minPts (minimum points): ";
    cin >> minPts;

    vector<Point> data = loadCSV(filename);

    DBSCAN(data, eps, minPts);

    cout << "\n--- Clustering Result ---\n";
    cout << "Point\tCluster\tType\n";
    for (auto &p : data)
```

```
        cout << p.x << "\t" << p.cluster << "\t" << p.type << endl;

    return 0;
}
```

Input

Points
1
1.2
1.3
5
5.1
9
9.5
9.7

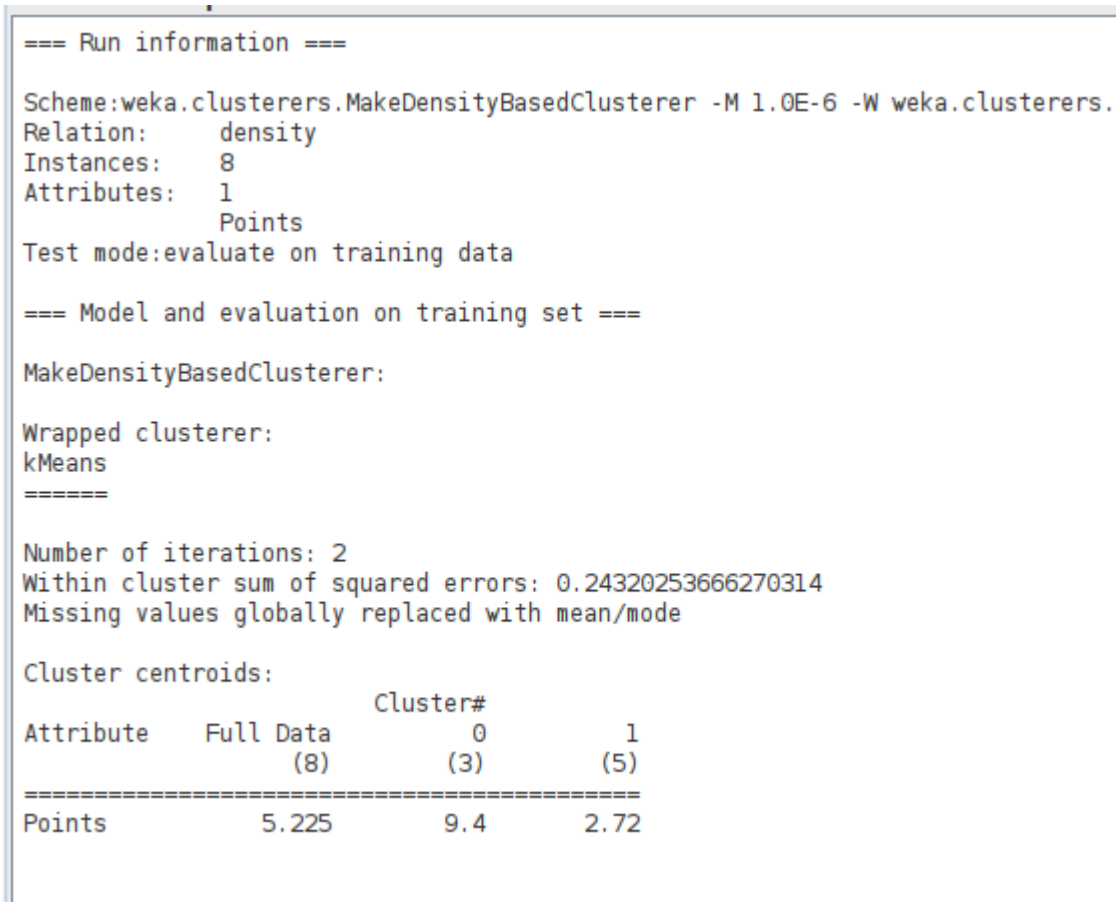
```
Enter CSV filename: dbscan.csv
Enter eps (neighborhood radius): 0.5
Enter minPts (minimum points): 3

--- Distance Matrix ---
0.00    0.20    0.30    4.00    4.10    8.00    8.50    8.70
0.20    0.00    0.10    3.80    3.90    7.80    8.30    8.50
0.30    0.10    0.00    3.70    3.80    7.70    8.20    8.40
4.00    3.80    3.70    0.00    0.10    4.00    4.50    4.70
4.10    3.90    3.80    0.10    0.00    3.90    4.40    4.60
8.00    7.80    7.70    4.00    3.90    0.00    0.50    0.70
8.50    8.30    8.20    4.50    4.40    0.50    0.00    0.20
8.70    8.50    8.40    4.70    4.60    0.70    0.20    0.00

--- Clustering Result ---
Point   Cluster Type
1.00    1       Core
1.20    1       Core
1.30    1       Core
5.00    0       Noise
5.10    0       Noise
9.00    2       Border
9.50    2       Core
9.70    2       Border
```



Rows: 8 | Columns: 2

#	RowID	Points *Number (Float)*	Cluster *String*
1	Row0	1	Cluster_0
2	Row1	1.2	Cluster_0
3	Row2	1.3	Cluster_0
4	Row3	5	Noise
5	Row4	5.1	Noise
6	Row5	9	Cluster_1
7	Row6	9.5	Cluster_1
8	Row7	9.7	Cluster_1

Weka



```
=== Run information ===

Scheme:weka.clusterers.MakeDensityBasedClusterer -M 1.0E-6 -W weka.clusterers.
Relation:     density
Instances:    8
Attributes:   1
              Points
Test mode:evaluate on training data

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

MakeDensityBasedClusterer:

Wrapped clusterer:
kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 0.24320253666270314
Missing values globally replaced with mean/mode

Cluster centroids:
                         Cluster#
Attribute    Full Data          0          1
                   (8)        (3)        (5)
=============================================
Points           5.225        9.4       2.72
```

```
Fitted estimators (with ML estimates of variance):

Cluster: 0 Prior probability: 0.4

Attribute: Points
Normal Distribution. Mean = 9.4 StdDev = 0.2944

Cluster: 1 Prior probability: 0.6

Attribute: Points
Normal Distribution. Mean = 2.72 StdDev = 1.9052


Time taken to build model (full training data) : 0 seconds

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

Clustered Instances

0      3 ( 38%)
1      5 ( 63%)


Log likelihood: -2.02567
```

-

Assignment 12 : Info Gain and Gini Index

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <map>
#include <cmath>
#include <algorithm>
#include <iomanip>

using namespace std;

class GainGiniCalculator
{
private:
  vector<vector<string>> dataset;
  vector<string> headers;
  int totalInstances;
  int playCount;
  int noPlayCount;

  double log2(double value)
  {
    if (value == 0)
      return 0;
    return log(value) / log(2);
  }

  double calculateEntropy(int yes, int total)
  {
    if (total == 0)
      return 0;
    double pYes = (double)yes / total;
    double pNo = 1 - pYes;

    double entropy = 0;
    if (pYes > 0)
      entropy -= pYes * log2(pYes);
    if (pNo > 0)
      entropy -= pNo * log2(pNo);
    return entropy;
  }

  double calculateGini(int yes, int total)
  {
    if (total == 0)
      return 0;
    double pYes = (double)yes / total;
    double pNo = 1 - pYes;
    return 1 - (pYes * pYes + pNo * pNo);
```

```
  }

  string categorizeTemperature(int temp)
  {
    if (temp > 80)
      return "High";
    else if (temp >= 70)
      return "Mild";
    else
      return "Low";
  }

  string categorizeHumidity(int humidity)
  {
    return (humidity > 80) ? "High" : "Normal";
  }

public:
  bool loadData(const string &filename)
  {
    ifstream file(filename);
    if (!file.is_open())
    {
      cerr << "Error: Cannot open file " << filename << endl;
      return false;
    }

    string line;
    // Read headers
    if (getline(file, line))
    {
      stringstream ss(line);
      string header;
      while (getline(ss, header, ','))
      {
        headers.push_back(header);
      }
    }

    // Read data
    while (getline(file, line))
    {
      if (line.empty())
        continue;

      vector<string> row;
      stringstream ss(line);
      string value;
      while (getline(ss, value, ','))
      {
        row.push_back(value);
      }
      dataset.push_back(row);
    }
```

```
  file.close();

  // Count Play/NoPlay
  playCount = 0;
  noPlayCount = 0;
  for (const auto &row : dataset)
  {
    if (row.size() > 4)
    {
      if (row[4] == "Play")
        playCount++;
      else if (row[4] == "Don't Play" || row[4] == "NoPlay")
        noPlayCount++;
    }
  }
  totalInstances = playCount + noPlayCount;

  return true;
}

void calculateForAttribute(int attributeIndex)
{
  if (attributeIndex < 0 || attributeIndex >= 4)
  {
    cerr << "Invalid attribute index" << endl;
    return;
  }

  // Calculate parent metrics
  double parentEntropy = calculateEntropy(playCount, totalInstances);
  double parentGini = calculateGini(playCount, totalInstances);

  cout << "\n=== Results for " << headers[attributeIndex] << " ===" << endl;
  cout << "Total Instances: " << totalInstances << endl;
  cout << "Instances with 'Play': " << playCount << endl;
  cout << "Instances with 'Don't Play': " << noPlayCount << endl;
  cout << fixed << setprecision(4);
  cout << "Parent Entropy: " << parentEntropy << endl;
  cout << "Parent Gini Index: " << parentGini << endl;

  // Count for each attribute value
  map<string, vector<int>> attributeCounts; // [playCount, noPlayCount]

  for (const auto &row : dataset)
  {
    if (row.size() <= 4)
      continue;

    string attributeValue = row[attributeIndex];

    // Convert numerical values to categorical if needed
    if (attributeIndex == 1)
    { // Temperature
      try
```

```
        {
          int temp = stoi(attributeValue);
          attributeValue = categorizeTemperature(temp);
        }
        catch (...)
        {
          // Keep original if not numeric
        }
      }
      else if (attributeIndex == 2)
      { // Humidity
        try
        {
          int humidity = stoi(attributeValue);
          attributeValue = categorizeHumidity(humidity);
        }
        catch (...)
        {
          // Keep original if not numeric
        }
      }

      if (attributeCounts.find(attributeValue) == attributeCounts.end())
      {
        attributeCounts[attributeValue] = {0, 0};
      }

      if (row[4] == "Play")
      {
        attributeCounts[attributeValue][0]++;
      }
      else
      {
        attributeCounts[attributeValue][1]++;
      }
    }

    // Print table header
    cout << "\n"
         << string(70, '-') << endl;
    cout << left << setw(15) << "Attribute Value"
         << setw(8) << "Play"
         << setw(12) << "Don't Play"
         << setw(8) << "Total"
         << setw(12) << "Entropy"
         << setw(12) << "Gini Index" << endl;
    cout << string(70, '-') << endl;

    // Calculate weighted averages
    double childEntropy = 0;
    double childGini = 0;

    for (const auto &pair : attributeCounts)
    {
```

```
    const string &value = pair.first;
    int play = pair.second[0];
    int noPlay = pair.second[1];
    int total = play + noPlay;

    double entropy = calculateEntropy(play, total);
    double gini = calculateGini(play, total);
    double proportion = (double)total / totalInstances;

    childEntropy += proportion * entropy;
    childGini += proportion * gini;

    cout << left << setw(15) << value
         << setw(8) << play
         << setw(12) << noPlay
         << setw(8) << total
         << setw(12) << entropy
         << setw(12) << gini << endl;
  }

  cout << string(70, '-') << endl;

  // Calculate gains
  double infoGain = parentEntropy - childEntropy;
  double giniGain = parentGini - childGini;

  cout << "\nChild Entropy: " << childEntropy << endl;
  cout << "Information Gain: " << infoGain << endl;
  cout << "Child Gini: " << childGini << endl;
  cout << "Gini Gain: " << giniGain << endl;
}

void compareAllAttributes()
{
  cout << "\n=== COMPARISON OF ALL ATTRIBUTES ===" << endl;
  cout << string(50, '-') << endl;
  cout << left << setw(15) << "Attribute"
       << setw(15) << "Info Gain"
       << setw(15) << "Gini Index" << endl;
  cout << string(50, '-') << endl;

  vector<pair<string, double>> infoGains;
  vector<pair<string, double>> giniIndices;

  for (int i = 0; i < 4; i++)
  {
    double parentEntropy = calculateEntropy(playCount, totalInstances);
    double parentGini = calculateGini(playCount, totalInstances);

    map<string, vector<int>> attributeCounts;

    for (const auto &row : dataset)
    {
      if (row.size() <= 4)
```

```
    continue;

  string attributeValue = row[i];

  if (i == 1)
  {
    try
    {
      int temp = stoi(attributeValue);
      attributeValue = categorizeTemperature(temp);
    }
    catch (...)
    {
    }
  }
  else if (i == 2)
  {
    try
    {
      int humidity = stoi(attributeValue);
      attributeValue = categorizeHumidity(humidity);
    }
    catch (...)
    {
    }
  }

  if (attributeCounts.find(attributeValue) == attributeCounts.end())
  {
    attributeCounts[attributeValue] = {0, 0};
  }

  if (row[4] == "Play")
  {
    attributeCounts[attributeValue][0]++;
  }
  else
  {
    attributeCounts[attributeValue][1]++;
  }
}

double childEntropy = 0;
double childGini = 0;

for (const auto &pair : attributeCounts)
{
  int play = pair.second[0];
  int noPlay = pair.second[1];
  int total = play + noPlay;

  double entropy = calculateEntropy(play, total);
  double gini = calculateGini(play, total);
  double proportion = (double)total / totalInstances;
```

```
      childEntropy += proportion * entropy;
      childGini += proportion * gini;
    }

    double infoGain = parentEntropy - childEntropy;

    cout << left << setw(15) << headers[i]
         << setw(15) << infoGain
         << setw(15) << childGini << endl;

    infoGains.push_back({headers[i], infoGain});
    giniIndices.push_back({headers[i], childGini});
  }

  // Find best attribute
  auto bestInfo = *max_element(infoGains.begin(), infoGains.end(),
                               [](const pair<string, double> &a, const pair<string, double> &b)
                               {
                                 return a.second < b.second;
                               });

  auto bestGini = *min_element(giniIndices.begin(), giniIndices.end(),
                               [](const pair<string, double> &a, const pair<string, double> &b)
                               {
                                 return a.second < b.second;
                               });

  cout << string(50, '-') << endl;
  cout << "\nBest attribute by Information Gain: " << bestInfo.first
       << " (Gain: " << bestInfo.second << ")" << endl;
  cout << "Best attribute by Gini Index: " << bestGini.first
       << " (Index: " << bestGini.second << ")" << endl;
 }
};

int main(int argc, char *argv[])
{
 if (argc != 2)
 {
  cerr << "Usage: " << argv[0] << " <csv_file>" << endl;
  cerr << "Example: " << argv[0] << " play_dataset.csv" << endl;
  return 1;
 }

 string filename = argv[1];
 GainGiniCalculator calculator;

 if (!calculator.loadData(filename))
 {
  return 1;
 }

 cout << "=== Gini Index and Information Gain Calculator ===" << endl;
```

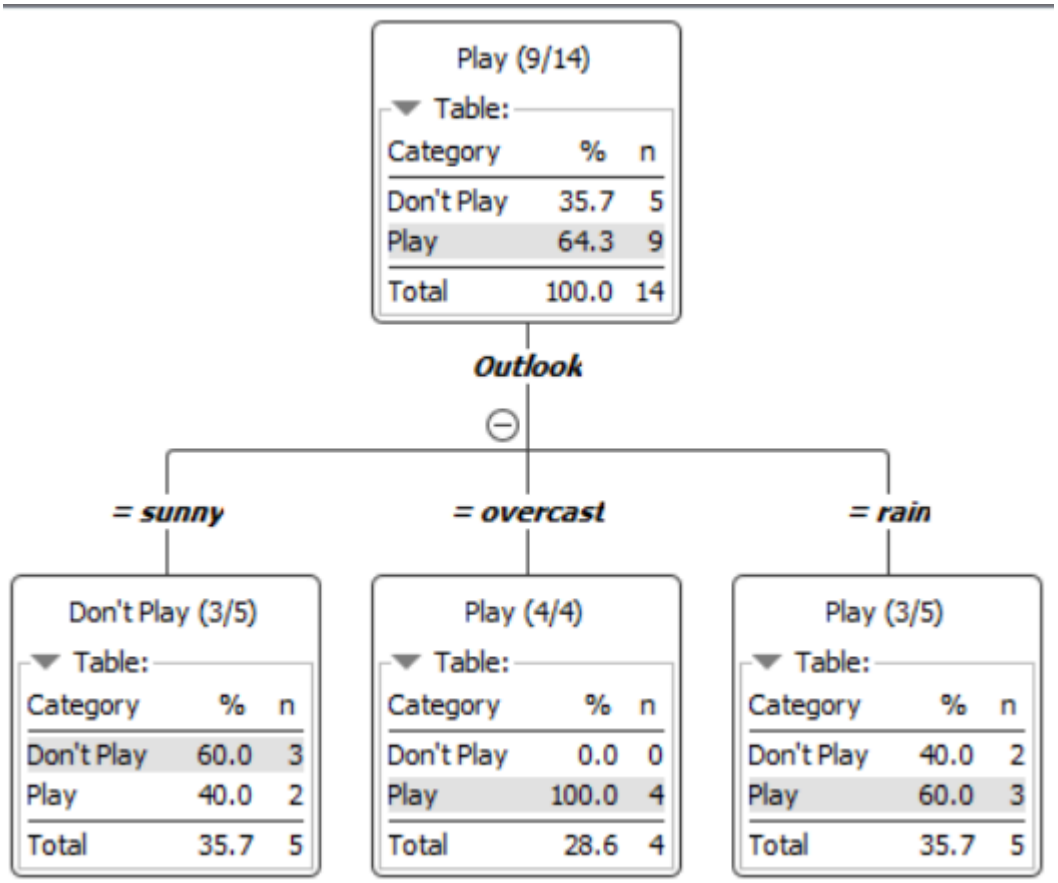
```
  while (true)
  {
    cout << "\nSelect Attribute to Analyze:" << endl;
    cout << "0 - Outlook" << endl;
    cout << "1 - Temperature" << endl;
    cout << "2 - Humidity" << endl;
    cout << "3 - Wind" << endl;
    cout << "4 - Compare All Attributes" << endl;
    cout << "5 - Exit" << endl;
    cout << "Enter your choice (0-5): ";

    int choice;
    cin >> choice;

    if (choice == 5)
      break;

    if (choice == 4)
    {
      calculator.compareAllAttributes();
    }
    else if (choice >= 0 && choice <= 3)
    {
      calculator.calculateForAttribute(choice);
    }
    else
    {
      cout << "Invalid choice!" << endl;
    }
  }

  return 0;
}
```

Input

Outlook	Temperature	Humidity	Wind	Play
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	86	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	91	TRUE	Don't Play

```
=== Gini Index and Information Gain Calculator ===

Select Attribute to Analyze:
0 - Outlook
1 - Temperature
2 - Humidity
3 - Wind
4 - Compare All Attributes
5 - Exit
Enter your choice (0-5): 4

=== COMPARISON OF ALL ATTRIBUTES ===
--------------------------------------------------
Attribute      Info Gain      Gini Index
--------------------------------------------------
Outlook        0.24675        0.342857
Temperature    0.0191005      0.447279
Humidity       0.151836       0.367347
Wind           0.048127       0.428571
--------------------------------------------------

Best attribute by Information Gain: Outlook (Gain: 0.24675)
Best attribute by Gini Index: Outlook (Index: 0.342857)
```


CSV Reader
Decision Tree Learner


Play (9/14)
Table:
Category % n
Don't Play 35.7 5
Play 64.3 9
Total 100.0 14
Outlook
= sunny
= overcast
= rain
Don't Play (3/5)
Table:
Category % n
Don't Play 60.0 3
Play 40.0 2
Total 35.7 5
Play (4/4)
Table:
Category % n
Don't Play 0.0 0
Play 100.0 4
Total 28.6 4
Play (3/5)
Table:
Category % n
Don't Play 40.0 2
Play 60.0 3
Total 35.7 5

Assignment 13: Naïve Bayes

Code

```
#include <bits/stdc++.h>
using namespace std;

// Function to trim spaces
string trim(const string &str) {
    size_t first = str.find_first_not_of(" \t\n\r");
    if (first == string::npos) return "";
    size_t last = str.find_last_not_of(" \t\n\r");
    return str.substr(first, (last - first + 1));
}

// Split CSV line
vector<string> split(string line, char delimiter = ',') {
    vector<string> tokens;
    string token;
    stringstream ss(line);
    while (getline(ss, token, delimiter)) {
        tokens.push_back(trim(token));
    }
    return tokens;
}

// Count frequency of attribute values given class
int countConditional(vector<vector<string>> &data, int colIndex, string value, string classLabel, int classCol) {
    int count = 0;
    for (auto &row : data) {
        if (row[classCol] == classLabel && row[colIndex] == value)
            count++;
    }
    return count;
}

// Count total rows of a class
int countClass(vector<vector<string>> &data, string classLabel, int classCol) {
    int count = 0;
    for (auto &row : data) {
        if (row[classCol] == classLabel)
            count++;
    }
    return count;
}

int main() {
    string fileName;
    cout << "Enter CSV file name (with .csv extension): ";
    cin >> fileName;

    ifstream file(fileName);
    if (!file.is_open()) {
        cerr << "Error opening file!" << endl;
```

```
    return 1;
}

string line;
vector<vector<string>> data;
vector<string> headers;

bool firstLine = true;
while (getline(file, line)) {
    vector<string> row = split(line);
    if (firstLine) {
        headers = row;
        firstLine = false;
    } else {
        data.push_back(row);
    }
}
file.close();

int classCol = headers.size() - 1; // Last column is the class

// Get unique classes
set<string> classes;
for (auto &row : data) classes.insert(row[classCol]);

// Take new case input
map<string, string> newCase;
cout << "\nEnter new case details:\n";
for (int i = 0; i < classCol; i++) {
    cout << headers[i] << ": ";
    string val;
    cin >> val;
    newCase[headers[i]] = val;
}

// Calculate prior and conditional probabilities
map<string, double> classProb;

for (auto &cls : classes) {
    int classCount = countClass(data, cls, classCol);
    double prior = (double)classCount / data.size();

    double likelihood = 1.0;
    for (int i = 0; i < classCol; i++) {
        int condCount = countConditional(data, i, newCase[headers[i]], cls, classCol);
        double condProb = (double)condCount / classCount;
        if (condProb == 0) condProb = 1e-6; // Laplace smoothing
        likelihood *= condProb;
    }
    classProb[cls] = prior * likelihood;
}

// Normalize probabilities
double sum = 0;
```

```
    for (auto &p : classProb) sum += p.second;
    for (auto &p : classProb) p.second /= sum;

    // Predict class with max probability
    string predictedClass;
    double maxProb = -1;
    for (auto &p : classProb) {
        if (p.second > maxProb) {
            maxProb = p.second;
            predictedClass = p.first;
        }
    }

    cout << "\nClass Probabilities:\n";
    for (auto &p : classProb)
        cout << p.first << ": " << fixed << setprecision(4) << p.second << endl;

    cout << "\nPredicted Class: " << predictedClass << endl;

    return 0;
}
```

Input

Outlook	Temperature	Humidity	Wind	Play
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	86	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	91	TRUE	Don't Play

Output

```
Outlook: sunny
Temperature: 72
Humidity: 85
Wind: FALSE

Class Probabilities:
Don't Play: 1.0000
Play: 0.0000

Predicted Class: Don't Play
```

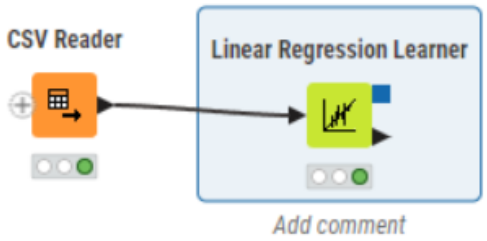
Weka

```
Scheme:weka.classifiers.bayes.NaiveBayes
Relation:     gaingini
Instances:    14
Attributes:   5
              Outlook
              Temperature
              Humidity
              Wind
              Play
Test mode:10-fold cross-validation

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

Naive Bayes Classifier
```

```
                Class
Attribute           No     Yes
                 (0.38)  (0.63)
================================
Outlook
  sunny             4.0     3.0
  overcast          1.0     5.0
  rain              3.0     4.0
  [total]           8.0    12.0

Temperature
  mean          74.8364 72.9697
  std. dev.       7.384  5.2304
  weight sum          5       9
  precision      1.9091  1.9091

Humidity
  mean          86.1111 78.8395
  std. dev.      9.2424  9.8023
  weight sum          5       9
  precision      3.4444  3.4444

Wind
  FALSE             3.0     7.0
  TRUE              4.0     4.0
  [total]           7.0    11.0



Time taken to build model: 0 seconds

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

Correctly Classified Instances           9               64.2857 %
Incorrectly Classified Instances         5               35.7143 %
Kappa statistic                          0.1026
Mean absolute error                      0.4649
Root mean squared error                  0.543
Relative absolute error                 97.6254 %
Root relative squared error            110.051  %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.2       0.111      0.5       0.2       0.286      0.444    No
                 0.889     0.8        0.667     0.889     0.762      0.444    Yes
Weighted Avg.    0.643     0.554      0.607     0.643     0.592      0.444

=== Confusion Matrix ===

 a b   <-- classified as
 1 4 | a = No
 1 8 | b = Yes
```

14 Linear Regression

	A	B	C
1	Hours_Studied	Exam_Score	XY
2	1	2	2
3	2	4	8
4	3	5	15
5	4	4	16
6	5	5	25
7	6	7	42
8	7	8	56
9	8	9	72
10			



1: Model for Predictor | 2: Coefficients and Statistics | Flow Variables

Rows: 3 | Columns: 5 — Table | Statistics

#	RowID	Variable (String)	Coeff. (Number (Float))	Std. Err. (Number (Float))
1	Row1	Hours_Studied	4.5	2.926
2	Row2	Exam_Score	5.816	3.076
3	Row3	Intercept	-22.737	6.332

t-value (Number (Float))	P>|t| (Number (Float))
1.538	0.185
1.891	0.117
-3.591	0.016