

A REPORT OF

“Data Mining Lab” Code: 5IT451

Submitted by

**Mr. Padmanabhan Deokar 2020BTEIT00024 Email:** Padmanabhan.deokar@walchandsanli.ac.in

**Mobile:** 7066329685

DEPARTMENT OF INFORMATION TECHNOLOGY

**WALCHAND COLLEGE OF ENGINEERING, SANGLI**

**(An Autonomous Institute) 2023-2024**

**CERTIFICATE**



This is to certify that the report entitled **“Data Mining Lab 5IT451***”* submitted by

**Mr. Padmanabhan Deokar (2020BTEIT00024)**

is a record of the student’s own work carried out by him during the academic year 2023-2024, as per the curriculum/syllabus laid down for OSS lab at Final year B. Tech IT Sem-I.

**Dr R.R.Rathod (Course Teacher)**

# Declaration

I, the undersigned, at this moment, declare that the BTech report entitled “Data Mining Lab 5IT451” submitted by me to Data Mining Lab report at Final year BTech IT Sem-I is my original/experimented/experience work. I further declare that, to the best of my knowledge and belief, this report has not been previously submitted or copied by me.

I declare that this report reflects my thoughts about the subject in my own words. I have sufficiently cited and referenced the original sources, referred, or considered in this work. I have not misinterpreted, fabricated, or falsified any idea/data/fact/source in this my submission. I understand that any violation of the above will be cause for disciplinary action by the course teacher/institute.

Date: 22-11-2023 **Mr. Padmanabhan Deokar**

Place: WCE Sangli

# Acknowledgement

I am pleased to submit the report entitled “Data Mining Laboratory (DM Lab) 5IT451”. I am thankful to our guide Dr. R.R.Rathod for their valuable guidance and kind help during implementing the DM Lab.

Acknowledged by, Mr. Padmanabhan Deokar

# Data Mining Lab Book

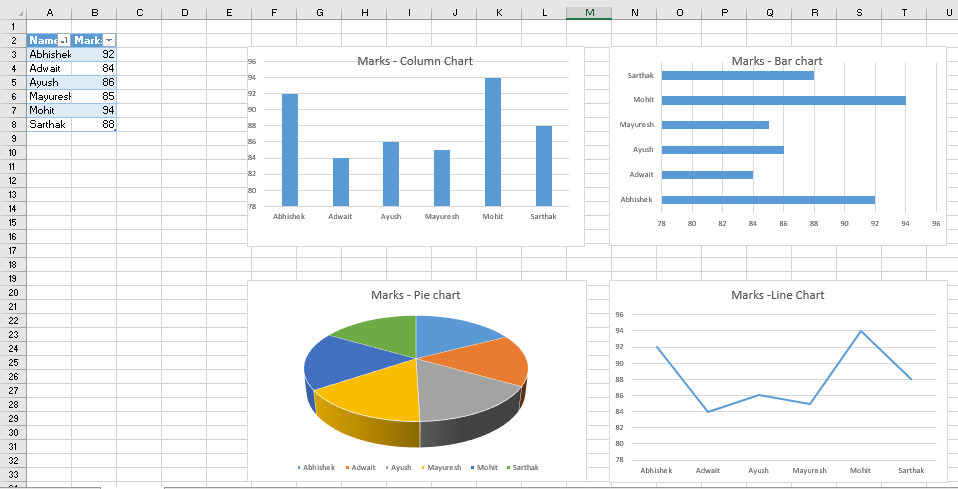
**Name: Padmanabhan Deokar**

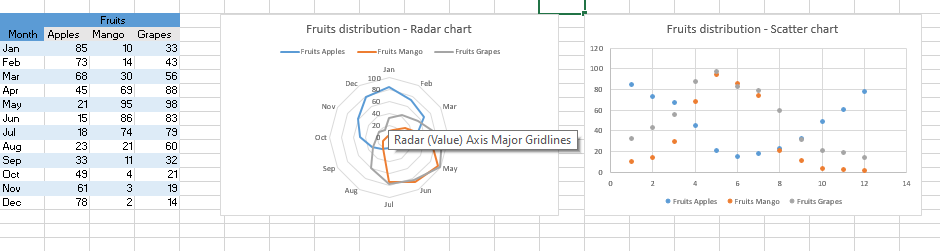
**PRN: 2020BTEIT00024**

**Class: Final Year-IT-Sem I (2023-2024)**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Title** | **Page No.** |
| 1. | Study and use of different types of graphs and charts (use MS-XLS). | 6 |
| 2. | Perform Normalization of data (Min-Max and Z- score ). | 7 |
| 3. | Perform Binning of data. | 9 |
| 4. | Find the Info Gain of an attribute from given data. | 11 |
| 5. | Find the t and d weight of the data. | 13 |
| 6. | Find 5 no summary of a dataset. | 15 |
| 7. | Find frequent item sets from given transaction data. | 17 |
| 8 | Extend program 7, to find association rule. | 20 |
| 9 | Find correlation between items/entities. | 22 |
| 10. | Distance and cluster. | 23 |
| 11. | Agglomerative Hierarchical Single Linkage Clustering. | 25 |
| 12. | Attribute for classification A. Gain B. Gini index. | 27 |
| 13. | WAP for Bayes classification. | 30 |

**Experiment No.1 Output:**





**Experiment No. 2 Program:**

#include <bits/stdc++.h>

#include <fstream>

using namespace std;

void calculateMinMax(ifstream &inputFile, double &minValue, double &maxValue)

{

  double currentValue;

  inputFile >> currentValue;

  while (inputFile)

  {

    if (currentValue > maxValue)

      maxValue = currentValue;

    if (currentValue < minValue)

      minValue = currentValue;

    inputFile >> currentValue;

  }

}

void performMinMaxNormalization(ifstream &inputFile, ofstream &outputFile, double oldMin, double oldMax, double newMin, double newMax)

{

  outputFile << "Original Data,"

        << "Normalized Data"

        << "\n";

  double currentValue;

  inputFile >> currentValue;

  while (inputFile)

  {

    double previousValue = currentValue;

    currentValue = (((currentValue - oldMin) / (oldMax - oldMin)) \* (newMax - newMin)) + newMin;

    outputFile << previousValue << "," << currentValue << "\n";

    inputFile >> currentValue;

  }

}

void performZScoreNormalization(ifstream &inputFile, ofstream &outputFile)

{

  double sum = 0.0, count = 0.0, squareSum = 0.0, mean, standardDeviation;

  double currentValue;

  // Calculate mean

  while (inputFile)

  {

    sum += currentValue;

    count++;

    inputFile >> currentValue;

  }

  mean = sum / count;

  // Calculate standard deviation

  inputFile.clear();

  inputFile.seekg(0, ios::beg);

  while (inputFile)

  {

    squareSum += (currentValue - mean) \* (currentValue - mean);

    inputFile >> currentValue;

  }

  inputFile.clear();

  inputFile.seekg(0, ios::beg);

  standardDeviation = sqrt(squareSum / count);

  // Perform z-score normalization

  outputFile << "Original Data,"

        << "Normalized Data"

        << "\n";

  while (inputFile)

  {

    double prev = currentValue;

    currentValue = (currentValue - mean) / standardDeviation;

    outputFile << prev << "," << currentValue << endl;

    inputFile >> currentValue;

  }

}

int main()

{

  double currentValue, minValue, maxValue, newMinValue, newMaxValue;

  double sum, count, squareSum, mean, standardDeviation;

  ifstream inputFileMinMax("input\_MinMax.csv");

  ifstream inputFileMinMax\_2("input\_MinMax.csv");

  ifstream inputFileZScore("input\_Zscore.csv");

  int option;

  cout << "\nEnter an option: \n1. Min-Max Normalization \n2. Z-Score Normalization\nOption: ";

  cin >> option;

  ofstream outputFileMinMax("output\_MinMax.csv", ios::app);

  ofstream outputFileZScore("output\_ZScore.csv", ios::app);

  switch (option)

  {

  case 1: // Min-Max Normalization

    if (!inputFileMinMax)

    {

      cout << "Error opening file, please try again.";

      exit(0);

    }

    calculateMinMax(inputFileMinMax, minValue, maxValue);

    cout << "Enter new minimum value: ";

    cin >> newMinValue;

    cout << "\nEnter new maximum value: ";

    cin >> newMaxValue;

    performMinMaxNormalization(inputFileMinMax\_2, outputFileMinMax, minValue, maxValue, newMinValue, newMaxValue);

    break;

  case 2: // Z-Score Normalization

    if (!inputFileZScore)

    {

      cout << "Error opening file, please try again.";

      exit(0);

    }

    performZScoreNormalization(inputFileZScore, outputFileZScore);

    break;

  default:

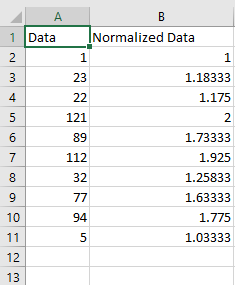
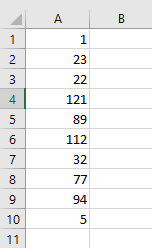
    cout << "Invalid option";

  }

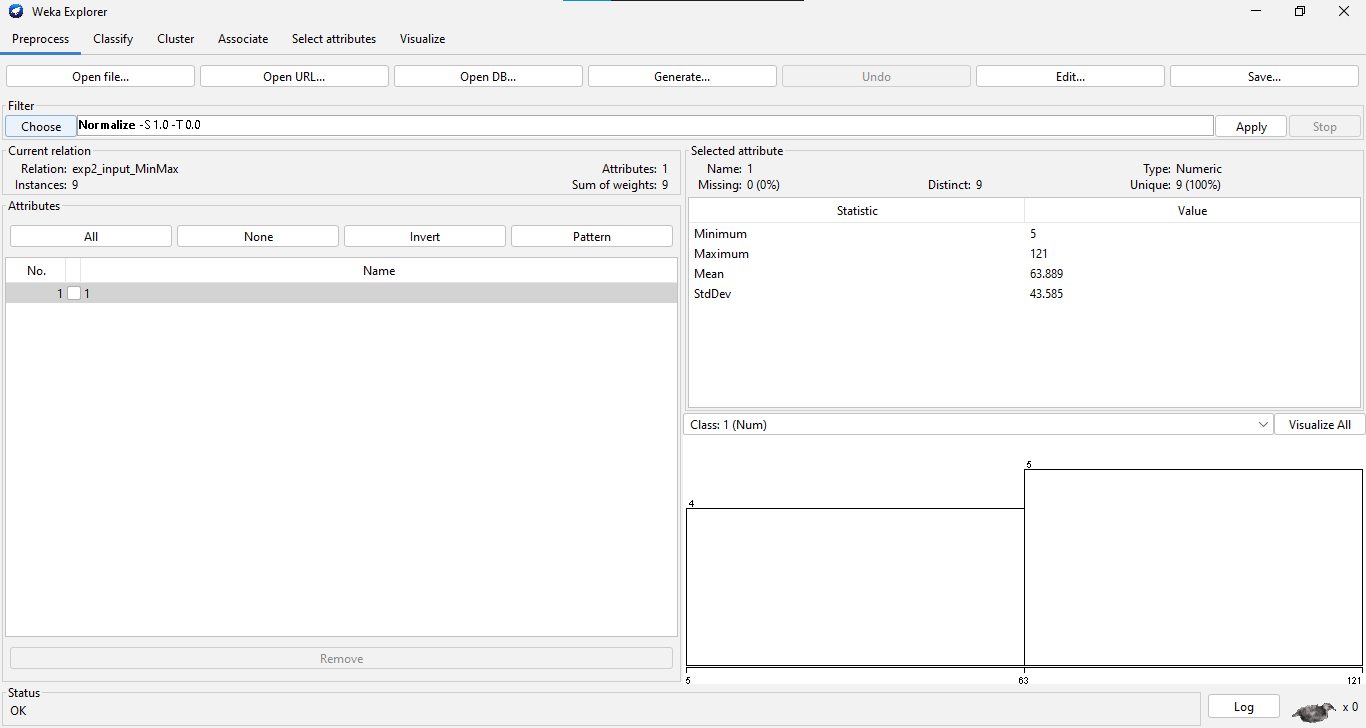
  return 0;

}

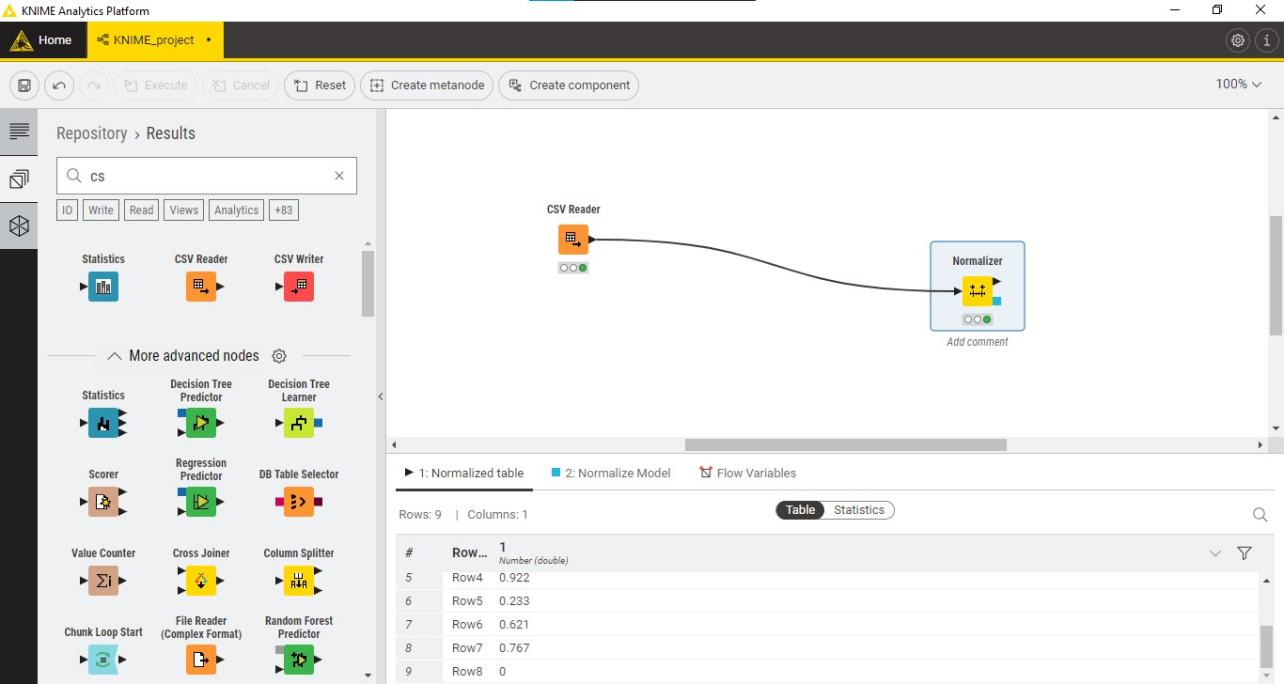
**INPUT OUTPUT**



## WEKA



**KNIME**



**Experiment No. 3 Program**

#include <iostream>

#include <fstream>

#include <vector>

#include <sstream>

#include <climits>

using namespace std;

// Equal Frequency Binning

vector<vector<int>> equifreq(vector<int> arr1, int m)

{

 int a = arr1.size();

 int n = a / m;

 vector<vector<int>> bins;

 for (int i = 0; i < m; i++)

 {

  vector<int> bin;

  for (int j = i \* n; j < (i + 1) \* n; j++)

  {

   if (j >= a)

   {

    break;

   }

   bin.push\_back(arr1[j]);

  }

  bins.push\_back(bin);

 }

 return bins;

}

// Equal Width Binning

vector<vector<int>> equiwidth(vector<int> arr1, int m)

{

 int a = arr1.size();

 int max\_ele = INT\_MIN;

 int min\_ele = INT\_MAX;

 for (int i = 0; i < arr1.size(); i++)

 {

  max\_ele = max(max\_ele, arr1[i]);

  min\_ele = min(min\_ele, arr1[i]);

 }

 int w = (max\_ele - min\_ele) / m;

 int min1 = min\_ele;

 vector<int> arr;

 for (int i = 0; i < m + 1; i++)

 {

  arr.push\_back(min1 + w \* i);

 }

 vector<vector<int>> arri;

 for (int i = 0; i < m; i++)

 {

  vector<int> temp;

  for (int j : arr1)

  {

   if (j >= arr[i] && j <= arr[i + 1])

   {

    temp.push\_back(j);

   }

  }

  arri.push\_back(temp);

 }

 return arri;

}

// Read data from CSV

vector<int> readCSV(string filename)

{

 ifstream inputFile(filename);

 vector<int> data;

 string line, value;

 while (getline(inputFile, line))

 {

  stringstream ss(line);

  while (getline(ss, value, ','))

  {

   data.push\_back(stoi(value));

  }

 }

 inputFile.close();

 return data;

}

// Write binning outputs to CSV

void writeCSV(string filename, vector<vector<int>> bins)

{

 ofstream outputFile(filename);

 for (int i = 0; i < bins.size(); i++)

 {

  outputFile << "Bin " << i + 1 << ",";

  for (int num : bins[i])

  {

   outputFile << num << ",";

  }

  outputFile << "\n";

 }

 outputFile.close();

}

int main()

{

 vector<int> data = readCSV("input.csv");

 int m;

 int method;

 cout << "Choose binning method: " << endl;

 cout << "1. Equal Frequency Binning" << endl;

 cout << "2. Equal Width Binning" << endl;

 cout << "\nEnter method number: ";

 cin >> method;

 cout << "\nEnter number of bins: ";

 cin >> m;

 if (method == 1)

 {

  vector<vector<int>> freqBins = equifreq(data, m);

  writeCSV("output\_equifreq.csv", freqBins);

 }

 else if (method == 2)

 {

  vector<vector<int>> widthBins = equiwidth(data, m);

  writeCSV("output\_equiwidth.csv", widthBins);

 }

 else

 {

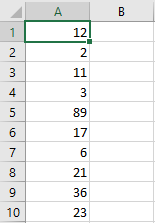
  cout << "Invalid method choice." << endl;

 }

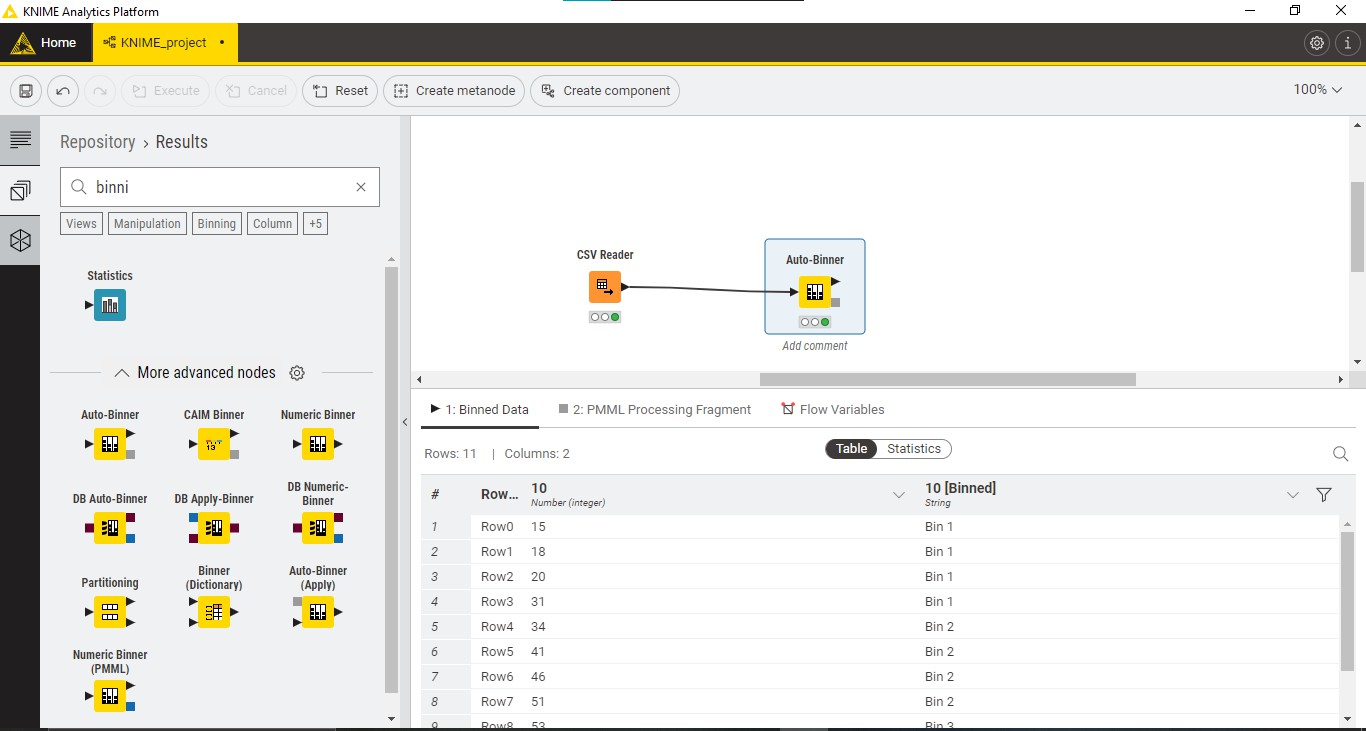
 return 0;

}

**Input Output**



## KNIME



**Experiment No.4 Program:**

#include <iostream>

#include <fstream>

#include <sstream>

#include <map>

#include <cmath>

using namespace std;

// Function to calculate entropy

double calculateEntropy(int positive, int negative)

{

  double total = positive + negative;

  double entropy = 0.0;

  if (total > 0)

  {

    double positiveProbability = positive / total;

    double negativeProbability = negative / total;

    if (positiveProbability > 0)

    {

      entropy -= positiveProbability \* log2(positiveProbability);

    }

    if (negativeProbability > 0)

    {

      entropy -= negativeProbability \* log2(negativeProbability);

    }

  }

  return entropy;

}

// Function to compute information gain

double computeInformationGain(map<string, int> &parentCounts, map<string, map<string, int>> &childCounts)

{

  double positiveParent = parentCounts["Yes"];

  double negativeParent = parentCounts["No"];

  double totalParent = positiveParent + negativeParent;

  double parentEntropy = calculateEntropy(positiveParent, negativeParent);

  cout << "Parent Entropy: " << parentEntropy << "\n";

  double childEntropy = 0;

  for (auto it = childCounts.begin(); it != childCounts.end(); ++it)

  {

    string childName = it->first;

    double positiveChild = it->second["Yes"];

    double negativeChild = it->second["No"];

    double totalChild = positiveChild + negativeChild;

    double childEntropyPart = calculateEntropy(positiveChild, negativeChild);

    childEntropy += (totalChild / totalParent) \* childEntropyPart;

  }

  cout << "Weighted Child Entropy: " << childEntropy << "\n";

  double informationGain = parentEntropy - childEntropy;

  cout << "Information Gain: " << informationGain << "\n";

  return informationGain;

}

int main()

{

  ifstream file("info-gain.csv");

  string line, day, level, routine, playGame, value;

  map<string, int> parentCounts;

  map<string, map<string, int>> childCounts;

  if (!file.is\_open())

  {

    cerr << "Error opening input file." << endl;

    return -1;

  }

  int i = 0;

  string childName;

  int choice;

  while (getline(file, line))

  {

    stringstream str(line);

    getline(str, day, ',');

    getline(str, level, ',');

    getline(str, routine, ',');

    getline(str, playGame, ',');

    getline(str, value, ',');

    if (i == 0)

    {

      i++;

      cout << "Enter Child Column Number: ";

      cin >> choice;

      continue;

    }

    switch (choice)

    {

    case 1:

      childName = day;

      break;

    case 2:

      childName = level;

      break;

    case 3:

      childName = routine;

      break;

    case 4:

      childName = value;

      break;

    default:

      childName = routine;

      break;

    }

    parentCounts[playGame]++;

    childCounts[childName][playGame]++;

  }

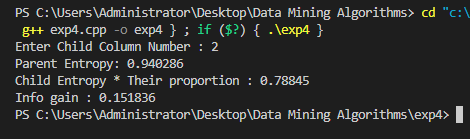
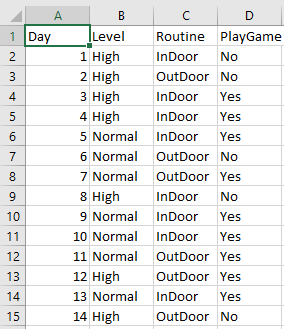
  double informationGain = computeInformationGain(parentCounts, childCounts);

  cout << "Overall Information Gain: " << informationGain << "\n";

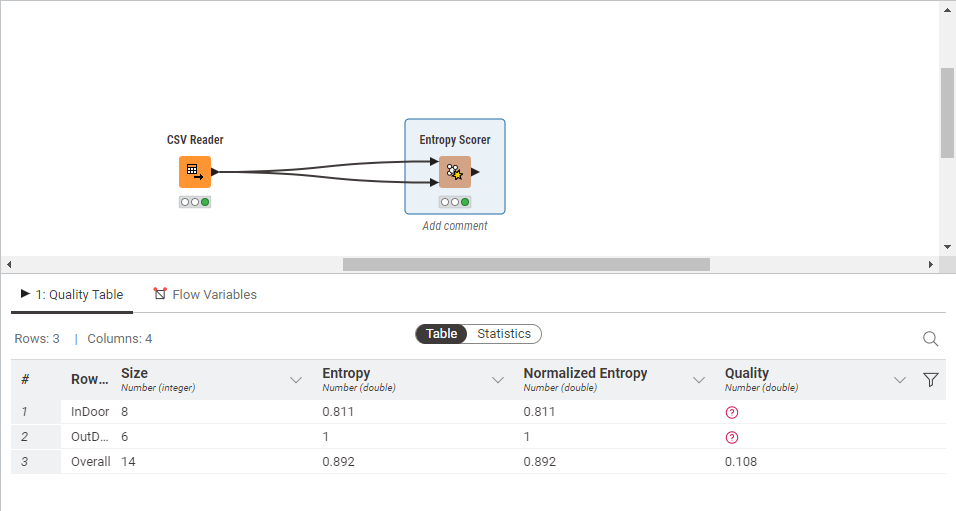
  return 0;

}

**Input Output**



## KNIME



**Experiment No.5 Program:**

#include <iostream>

#include <fstream>

#include <sstream>

#include <map>

using namespace std;

// A struct to represent a cell's data in the CSV file

struct CellData

{

  int count;

  int tWeight;

  int dWeight;

};

// Function to read data from the input CSV file into the provided data structures

void readData(const string &filename, map<string, map<string, CellData>> &cellData,

       map<string, int> &columnTotal, map<string, int> &rowTotal)

{

  fstream file(filename, ios::in);

  if (!file.is\_open())

  {

    cout << "Couldn't open file: " << filename << endl;

    return;

  }

  string line, row, col, count;

  int val;

  int lineNumber = 0;

  while (getline(file, line))

  {

    stringstream str(line);

    if (lineNumber == 0)

    {

      lineNumber++;

      continue; // Skip the header line

    }

    getline(str, row, ',');

    getline(str, col, ',');

    getline(str, count, ',');

    val = stoi(count);

    cellData[row][col].count += val;

    columnTotal[col] += val;

    rowTotal[row] += val;

  }

}

// Function to write the result to an output CSV file

void writeResult(const string &filename, const map<string, map<string, CellData>> &cellData,

         const map<string, int> &columnTotal, const map<string, int> &rowTotal)

{

  ofstream fw(filename, ios::out);

  fw << "Column\\Row, Count, T-Weight, D-Weight, Count, T-Weight, D-Weight, Count, T-Weight, D-Weight" << endl;

  for (const auto &rowEntry : rowTotal)

  {

    const string &row = rowEntry.first;

    fw << row << ",";

    for (const auto &colEntry : columnTotal)

    {

      const string &col = colEntry.first;

      const CellData &cell = cellData.at(row).at(col);

      fw << cell.count << ",";

      fw << ((float)cell.count / rowTotal.at(row)) \* 100 << "%,";

      fw << ((float)cell.count / colEntry.second) \* 100 << "%,";

    }

    fw << rowTotal.at(row) << ","

      << "100%, " << ((float)rowTotal.at(row) / rowTotal.at(rowTotal.begin()->first)) \* 100 << "%" << endl;

  }

  fw << "Total,";

  for (const auto &colEntry : columnTotal)

  {

    fw << colEntry.second << ",";

    fw << ((float)colEntry.second / columnTotal.at(columnTotal.begin()->first)) \* 100 << "%,";

    fw << "100%,";

  }

  fw << columnTotal.at(columnTotal.begin()->first) << ",100%, 100%" << endl;

}

int main()

{

  map<string, map<string, CellData>> cellData;

  map<string, int> columnTotal;

  map<string, int> rowTotal;

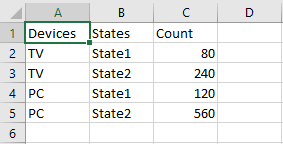
  readData("t-d-weight-input.csv", cellData, columnTotal, rowTotal);

  writeResult("t-d-weight-output.csv", cellData, columnTotal, rowTotal);

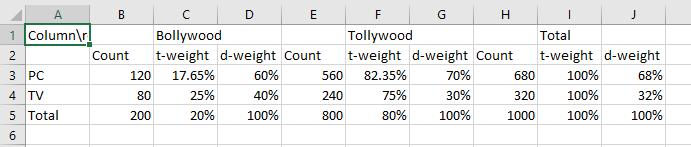
  cout << "Processing complete. Results saved to 't-d-weight-output.csv'." << endl;

  return 0;

}

**Input**

**Output**



**Experiment No.6 Program:**

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <algorithm>

using namespace std;

// Function to calculate the median of a vector

float calculateMedian(vector<int> a)

{

  int size = a.size();

  if (size % 2 == 1)

    return a[size / 2];

  else

    return (a[(size / 2) - 1] + a[size / 2]) / 2.0;

}

// Function to calculate the first quartile (Q1)

float calculateQuartile1(vector<int> v)

{

  int n = v.size();

  vector<int> first;

  for (int i = 0; i < n / 2; i++)

  {

    first.push\_back(v[i]);

  }

  return calculateMedian(first);

}

// Function to calculate the third quartile (Q3)

float calculateQuartile3(vector<int> v)

{

  int n = v.size();

  vector<int> last;

  if (n % 2 == 0)

  {

    for (int i = n / 2; i < n; i++)

    {

      last.push\_back(v[i]);

    }

  }

  else

  {

    for (int i = n / 2 + 1; i < n; i++)

    {

      last.push\_back(v[i]);

    }

  }

  return calculateMedian(last);

}

int main()

{

  ifstream in("five\_number\_input.csv");

  if (!in.is\_open())

  {

    cout << "Error: Unable to open the input file." << endl;

    exit(0);

  }

  ofstream out("five\_number\_output.csv");

  int i = 0;

  string line, mark;

  vector<int> arr;

  // Read data from the input file

  while (getline(in, line))

  {

    if (i == 0)

    {

      i++;

      continue;

    }

    stringstream str(line);

    getline(str, mark, ',');

    int x = stoi(mark);

    arr.push\_back(x);

  }

  int n = arr.size();

  sort(arr.begin(), arr.end());

  // Write results to the output file and console

  out << "Minimum value: "

    << "," << arr[0] << "\n";

  out << "First Quartile (Q1) value: "

    << "," << calculateQuartile1(arr) << "\n";

  out << "Median value: "

    << "," << calculateMedian(arr) << "\n";

  out << "Third Quartile (Q3) value: "

    << "," << calculateQuartile3(arr) << "\n";

  out << "Maximum value: "

    << "," << arr[n - 1] << "\n";

  cout << "The minimum value is " << arr[0] << endl;

  cout << "The First Quartile (Q1) is " << calculateQuartile1(arr) << endl;

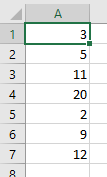
  cout << "The median is " << calculateMedian(arr) << endl;

  cout << "The Third Quartile (Q3) is " << calculateQuartile3(arr) << endl;

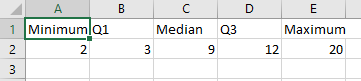
  cout << "The maximum value is " << arr[n - 1] << endl;

  return 0;

}

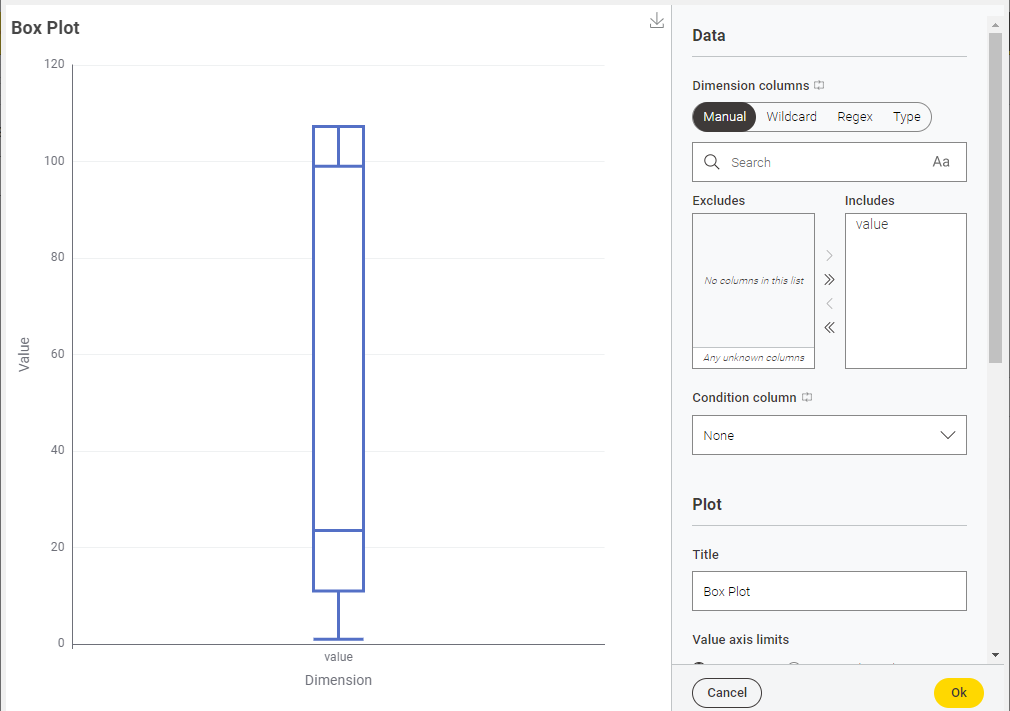


**Output:**



## KNIME:





**Experiment No.7 Program**

#include <bits/stdc++.h>

#include <map>

using namespace std;

ifstream fin;

double minfre;

vector<set<string>> datatable;

set<string> products;

map<string, int> freq;

// Function to split a string into words based on alphanumeric characters.

vector<string> wordsof(string str)

{

  vector<string> tmpset;

  string tmp = "";

  int i = 0;

  while (str[i])

  {

    if (isalnum(str[i]))

      tmp += str[i];

    else

    {

      if (tmp.size() > 0)

        tmpset.push\_back(tmp);

      tmp = "";

    }

    i++;

  }

  if (tmp.size() > 0)

    tmpset.push\_back(tmp);

  return tmpset;

}

// Function to combine elements in a vector into a string, excluding the one at 'miss' index.

string combine(vector<string> &arr, int miss)

{

  string str;

  for (int i = 0; i < arr.size(); i++)

    if (i != miss)

      str += arr[i] + " ";

  str = str.substr(0, str.size() - 1);

  return str;

}

// Function to clone a set and return a copy.

set<string> cloneit(set<string> &arr)

{

  set<string> dup;

  for (set<string>::iterator it = arr.begin(); it != arr.end(); it++)

    dup.insert(\*it);

  return dup;

}

// Function to generate frequent itemsets of size k based on candidate itemsets of size k-1.

set<string> apriori\_gen(set<string> &sets, int k)

{

  set<string> set2;

  for (set<string>::iterator it1 = sets.begin(); it1 != sets.end(); it1++)

  {

    set<string>::iterator it2 = it1;

    it2++;

    for (; it2 != sets.end(); it2++)

    {

      vector<string> v1 = wordsof(\*it1);

      vector<string> v2 = wordsof(\*it2);

      bool alleq = true;

      for (int i = 0; i < k - 1 && alleq; i++)

        if (v1[i] != v2[i])

          alleq = false;

      v1.push\_back(v2[k - 1]);

      if (v1[v1.size() - 1] < v1[v1.size() - 2])

        swap(v1[v1.size() - 1], v1[v1.size() - 2]);

      for (int i = 0; i < v1.size() && alleq; i++)

      {

        string tmp = combine(v1, i);

        if (sets.find(tmp) == sets.end())

          alleq = false;

      }

      if (alleq)

        set2.insert(combine(v1, -1));

    }

  }

  return set2;

}

int main()

{

  fin.open("item\_set\_input.csv", ios::in); // Open the input file for reading.

  if (!fin.is\_open())

  {

    perror("Error in opening file : "); // Print an error message if the file cannot be opened.

  }

  cout << "Frequency % :";

  cin >> minfre; // Read the minimum frequency from the user.

  string str;

  while (!fin.eof())

  {

    getline(fin, str);

    vector<string> arr = wordsof(str);

    set<string> tmpset;

    for (int i = 0; i < arr.size(); i++)

      tmpset.insert(arr[i]);

    datatable.push\_back(tmpset); // Store the transaction data in the 'datatable' vector.

    for (set<string>::iterator it = tmpset.begin(); it != tmpset.end(); it++)

    {

      products.insert(\*it); // Store unique products in the 'products' set.

      freq[\*it]++;      // Increment the frequency of each product in the 'freq' map.

    }

  }

  fin.close(); // Close the input file.

  cout << "No of transactions: " << datatable.size() << endl;

  minfre = minfre \* datatable.size() / 100; // Calculate the minimum frequency threshold.

  cout << "Min frequency:" << minfre << endl;

  queue<set<string>::iterator> q;

  for (set<string>::iterator it = products.begin(); it != products.end(); it++)

    if (freq[\*it] < minfre)

      q.push(it);

  while (q.size() > 0)

  {

    products.erase(\*q.front()); // Remove infrequent products from the 'products' set.

    q.pop();

  }

  int pass = 1;

  cout << "\nFrequent " << pass++ << " -item set : \n";

  for (set<string>::iterator it = products.begin(); it != products.end(); it++)

    cout << "{" << \*it << "} " << freq[\*it] << endl; // Display frequent 1-itemsets.

  int i = 2;

  set<string> prev = cloneit(products);

  while (i)

  {

    set<string> cur = apriori\_gen(prev, i - 1); // Generate candidate itemsets of size 'i'.

    if (cur.size() < 1)

    {

      break;

    }

    for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)

    {

      vector<string> arr = wordsof(\*it);

      int tot = 0;

      for (int j = 0; j < datatable.size(); j++)

      {

        bool pres = true;

        for (int k = 0; k < arr.size() && pres; k++)

          if (datatable[j].find(arr[k]) == datatable[j].end())

            pres = false;

        if (pres)

          tot++;

      }

      if (tot >= minfre)

        freq[\*it] += tot;

      else

        q.push(it);

    }

    while (q.size() > 0)

    {

      cur.erase(\*q.front());

      q.pop();

    }

    bool flag = true;

    for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)

    {

      vector<string> arr = wordsof(\*it);

      if (freq[\*it] < minfre)

        flag = false;

    }

    if (cur.size() == 0)

      break;

    cout << "\n\nFrequent " << pass++ << " -item set : \n";

    for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)

      cout << "{" << \*it << "} " << freq[\*it] << endl; // Display frequent k-itemsets.

    prev = cloneit(cur);

    i++;

  }

  ofstream fw("item\_set\_output.csv", ios::out); // Open an output file for writing.

  for (auto it = prev.begin(); it != prev.end(); it++)

  {

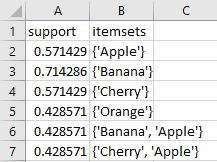
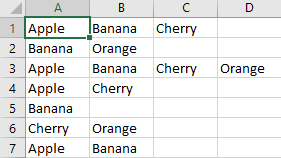
    fw << "{" << \*it << "}" << endl; // Write frequent itemsets to the output file.

  }

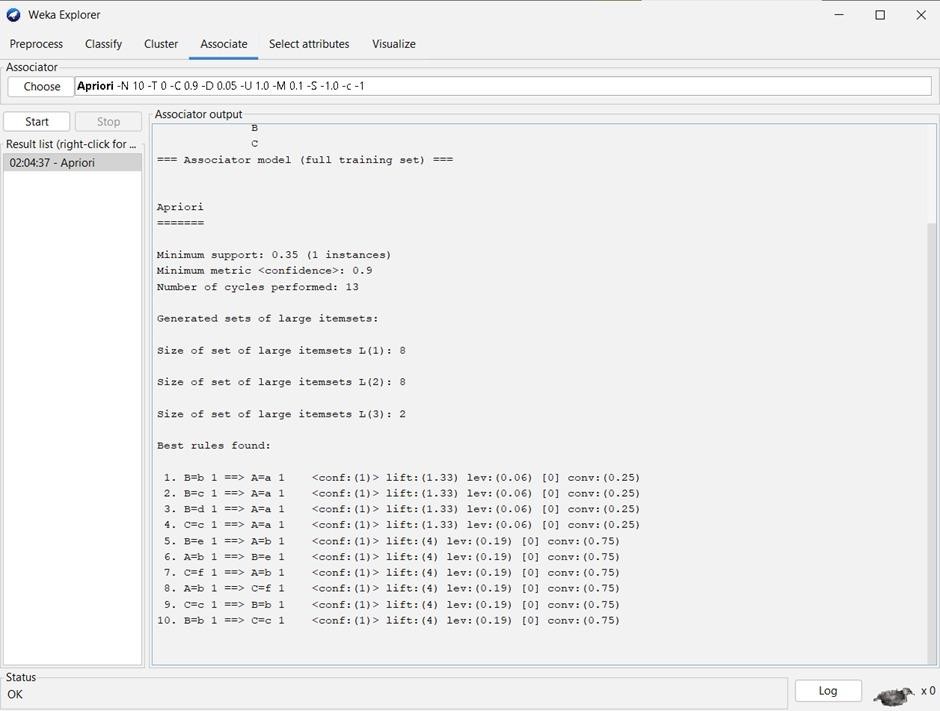
  return 1;

}

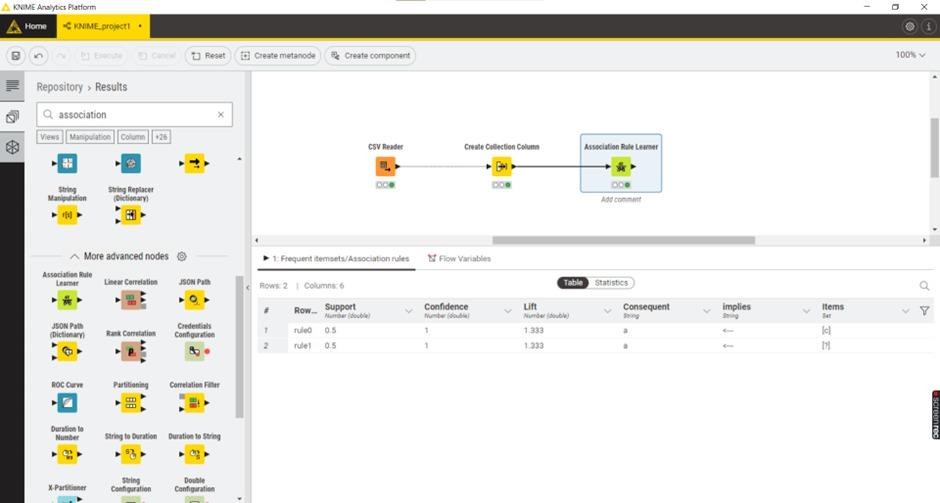
**Input Output**



## WEKA



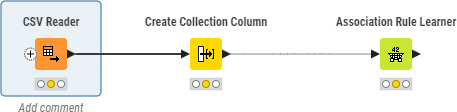
**KNIME**



**Experiment No.8**

**Aim:-** Extend program 7 to find Association rules.

## KNIME



**Experiment No.9 Program:**

#include <iostream>

#include <fstream>

#include <vector>

#include <sstream>

using namespace std;

vector<int> readData(const string &filename)

{

  vector<int> data;

  fstream file(filename, ios::in);

  if (!file.is\_open())

  {

    cerr << "Error in opening input file: " << filename << endl;

    exit(1);

  }

  string line, value;

  int lineCount = 0;

  while (getline(file, line))

  {

    if (lineCount > 0)

    {

      stringstream ss(line);

      getline(ss, value, ',');

      data.push\_back(stoi(value));

    }

    lineCount++;

  }

  file.close();

  return data;

}

float calculateCorrelationCoefficient(const vector<int> &a, const vector<int> &b)

{

  int n = a.size();

  int a\_plus = 0, b\_plus = 0, ab\_plus = 0;

  for (int i = 0; i < n; i++)

  {

    a\_plus += a[i] == 1 ? 1 : 0;

    b\_plus += b[i] == 1 ? 1 : 0;

    ab\_plus += (a[i] == 1 && b[i] == 1) ? 1 : 0;

  }

  if (a\_plus == 0 || b\_plus == 0)

  {

    return 0.0; // To handle cases where division by zero may occur

  }

  return static\_cast<float>(ab\_plus) / (a\_plus \* b\_plus);

}

void writeCorrelationCoefficient(const string &filename, float corr\_coeff)

{

  ofstream file(filename, ios::out);

  if (!file.is\_open())

  {

    cerr << "Error in opening output file: " << filename << endl;

    exit(1);

  }

  file << "Pearson Correlation Coefficient"

     << "," << corr\_coeff << endl;

  file.close();

}

int main()

{

  string inputFileName = "correlation\_input.csv";

  string outputFileName = "correlation\_output.csv";

  vector<int> a = readData(inputFileName);

  vector<int> b = readData(inputFileName);

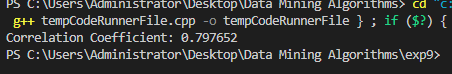
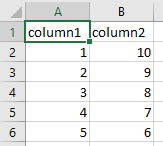
  float corr\_coeff = calculateCorrelationCoefficient(a, b);

  writeCorrelationCoefficient(outputFileName, corr\_coeff);

  cout << "Correlation coefficient calculated and saved in '" << outputFileName << "'." << endl;

  return 0;

}

**Input Output**

**Experiment 10 Program:**

#include <bits/stdc++.h>

#include <limits>

using namespace std;

// Function to calculate the Euclidean distance between two points (x1, y1) and (x2, y2)

float distance(float x1, float y1, int x2, int y2)

{

  return sqrt(((float)x2 - x1) \* ((float)x2 - x1) + ((float)y2 - y1) \* ((float)y2 - y1));

}

int main()

{

  string line;

  int mid\_point;

  string point, x, y;

  int i = 0;

  int val1;

  int val2;

  vector<pair<int, int>> v;

  // Open the input file for reading

  fstream in("cluster\_input.csv", ios::in);

  // Check if the file was opened successfully

  if (!in.is\_open())

  {

    cout << "Error: Unable to open the input file 'cluster\_input.csv'." << endl;

    return -1;

  }

  // Read data from the CSV file line by line

  while (getline(in, line))

  {

    stringstream str(line);

    if (i == 0)

    {

      i++;

      continue; // Skip the first line (possibly a header).

    }

    // Parse the CSV line into variables point, x, and y

    getline(str, point, ',');

    getline(str, x, ',');

    getline(str, y, ',');

    val1 = stoi(x); // Convert x to an integer

    val2 = stoi(y); // Convert y to an integer

    v.push\_back({val1, val2}); // Add the pair (val1, val2) to the vector v

  }

  int n = v.size();

  int x\_sum = 0, y\_sum = 0;

  // Calculate the sum of x and y coordinates

  for (int i = 0; i < v.size(); i++)

  {

    int first = v[i].first;

    int second = v[i].second;

    x\_sum += first;

    y\_sum += second;

  }

  // Calculate the coordinates of the midpoint (average)

  float mid\_x = (float)x\_sum / n;

  float mid\_y = (float)y\_sum / n;

  cout << "Midpoint of the data: (" << mid\_x << "," << mid\_y << ")" << endl;

  // Open the output file for writing

  ofstream out("cluster\_output.csv");

  // Write the header for the output CSV file

  out << " , p1 ,p2 ,p3 ,p4,C";

  out << "\n";

  // Calculate the distances between all pairs of points

  for (int i = 0; i < v.size(); i++)

  {

    if (i < v.size())

      out << "p" << i + 1 << ",";

    for (int j = 0; j <= i; j++)

    {

      int f\_x1 = v[i].first;

      int s\_y1 = v[i].second;

      int f\_x2 = v[j].first;

      int s\_y2 = v[j].second;

      if (f\_x1 == f\_x2 && s\_y1 == s\_y2)

      {

        out << "0"

          << ",";

        break;

      }

      float dis = distance(f\_x1, s\_y1, f\_x2, s\_y2);

      out << dis << ",";

    }

    out << "\n";

  }

  out << "C"

    << ",";

  // Variables for finding the nearest point to the center

  pair<int, int> p;

  int nearestPoint = 0;

  float x\_new;

  float y\_new;

  float nearestDistance = INT\_MAX;

  // Calculate the distances of each point from the calculated midpoint

  for (int i = 0; i < v.size(); i++)

  {

    int first = v[i].first;

    int second = v[i].second;

    float d = distance(mid\_x, mid\_y, first, second);

    cout << "Distance of p" << i + 1 << " from the center: " << d << " units." << endl;

    if (nearestDistance > d)

    {

      nearestDistance = d;

      nearestPoint = i + 1;

      x\_new = first;

      y\_new = second;

    }

    out << d << ",";

    if (i == v.size() - 1)

      out << "0"

        << ",";

  }

  cout << "The nearest distance from the center is: " << nearestDistance << " units." << endl;

  cout << "The nearest point from the center is: "

     << "p" << nearestPoint << endl;

  out << ",";

  out << "\n";

  // New Center Calculation

  out << " , p1 ,p2 ,p3 ,p4";

  out << "\n";

  for (int i = 0; i < v.size(); i++)

  {

    if (i < v.size())

      out << "p" << i + 1 << ",";

    for (int j = 0; j <= i; j++)

    {

      int f\_x1 = v[i].first;

      int s\_y1 = v[i].second;

      int f\_x2 = v[j].first;

      int s\_y2 = v[j].second;

      if (f\_x1 == f\_x2 && s\_y1 == s\_y2)

      {

        out << "0"

          << ",";

        break;

      }

      float dis = distance(f\_x1, s\_y1, f\_x2, s\_y2);

      out << dis << ",";

    }

    out << "\n";

  }

  out << "p" << nearestPoint << " (New Center)"

    << ",";

  // Calculate the distances of each point from the new center

  for (int i = 0; i < v.size(); i++)

  {

    int first = v[i].first;

    int second = v[i].second;

    float d = distance(x\_new, y\_new, first, second);

    cout << "Distance of p" << i + 1 << " from the new center (p" << nearestPoint << "): " << d << " units." << endl;

    out << d << ",";

    if (i == v.size() - 1)

      out << "0"

        << ",";

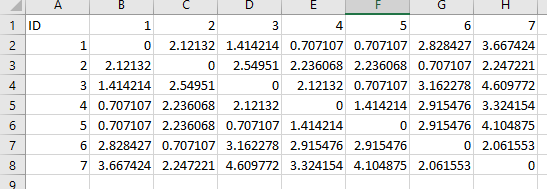
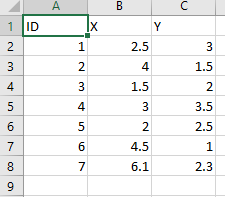
  }

  cout << "Results have been written to 'cluster\_output.csv'." << endl;

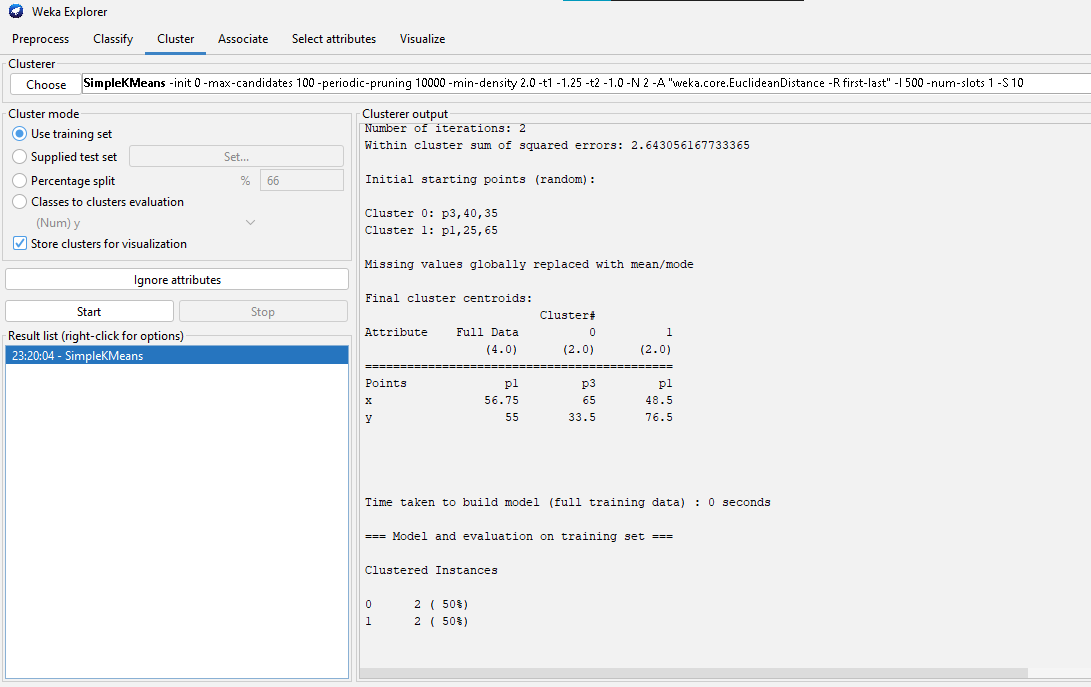
  return 0;

}

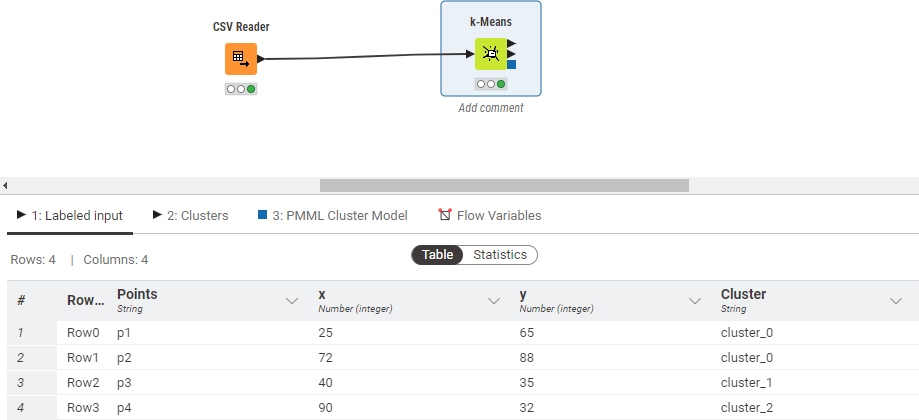
**Input Output**



## WEKA



**KNIME**



**Experiment No.11 Program :**

#include <bits/stdc++.h>

using namespace std;

int op = 1;

ofstream fwtr("linkage\_output.csv", ios::out);

// Function to perform agglomerative clustering and return the name of the resulting cluster

string agglomerative(string input)

{

  map<string, map<string, int>> dm;

  fstream file(input, ios::in);

  string line;

  getline(file, line);

  int pt = 0;

  stringstream st(line);

  int i = 0;

  string point;

  vector<string> points;

  // Read the point names from the first line of the input file

  while (getline(st, point, ','))

  {

    if (i == 0)

    {

      i++;

      continue;

    }

    points.push\_back(point);

  }

  // Populate the distance matrix from the input file

  while (getline(file, line))

  {

    stringstream str(line);

    getline(str, point, ',');

    string dist;

    int idx = 0;

    while (getline(str, dist, ','))

    {

      if (dist.length() != 0)

        dm[point][points[idx]] = stoi(dist);

      idx++;

    }

  }

  string pt1, pt2;

  int min\_dist = INT\_MAX;

  // Find the two points with the minimum distance

  for (auto p : dm)

  {

    for (auto pp : p.second)

    {

      string p1 = p.first, p2 = pp.first;

      int dist = pp.second;

      if (p1 != p2 && dist < min\_dist)

      {

        pt1 = p1;

        pt2 = p2;

        min\_dist = dist;

      }

    }

  }

  cout << "Clusters Chosen: " << pt1 << " & " << pt2 << endl;

  string up, down;

  // Determine the order of the two points based on their names

  if (pt1[0] > pt2[0])

  {

    up = pt2;

    down = pt1;

  }

  else

  {

    up = pt1;

    down = pt2;

  }

  string newPt = down + up;

  // Update distances and remove old points from the matrix

  for (auto p : dm)

  {

    point = p.first;

    if (point[0] > newPt[0])

    {

      dm[point][newPt] = min(dm[point][up], dm[point][down]);

    }

  }

  for (auto p : dm[down])

  {

    point = p.first;

    int d1 = p.second;

    if (point[0] < up[0])

      d1 = min(d1, dm[up][point]);

    else

      d1 = min(d1, dm[point][up]);

    dm[newPt][point] = d1;

  }

  for (auto p : dm)

  {

    point = p.first;

    auto mtemp = p.second;

    if (point[0] >= up[0])

    {

      int d1 = dm[point][up];

      if (down[0] > point[0])

        d1 = min(d1, dm[down][point]);

      else

        d1 = min(d1, dm[point][down]);

      dm[point][newPt] = d1;

      dm[point].erase(up);

      if (point[0] >= down[0])

        dm[point].erase(down);

    }

  }

  dm.erase(up);

  dm.erase(down);

  // Create an output file with updated cluster data

  string output = "output" + to\_string(op++) + ".csv";

  ofstream fw(output, ios::out);

  fw << ",";

  for (auto p : dm)

  {

    fw << p.first << ",";

  }

  fw << "\n";

  for (auto p : dm)

  {

    fw << p.first << ",";

    for (auto pp : p.second)

    {

      fw << pp.second << ",";

    }

    fw << "\n";

  }

  fw.close();

  fwtr << down << " & " << up << "\n";

  return output;

}

int main()

{

  string input = "linkage\_input.csv";

  fstream file1(input, ios::in);

  string line;

  getline(file1, line);

  int pt = 0;

  stringstream st(line);

  int j = 0, len = 0;

  string point;

  // Determine the number of points in the dataset

  while (getline(st, point, ','))

  {

    if (j == 0)

    {

      j++;

      continue;

    }

    len++;

  }

  // Repeatedly perform agglomerative clustering to create clusters

  for (int i = 1; i <= len - 2; i++)

  {

    string output = agglomerative(input);

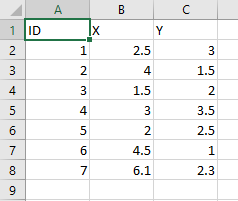
    input = output;

  }

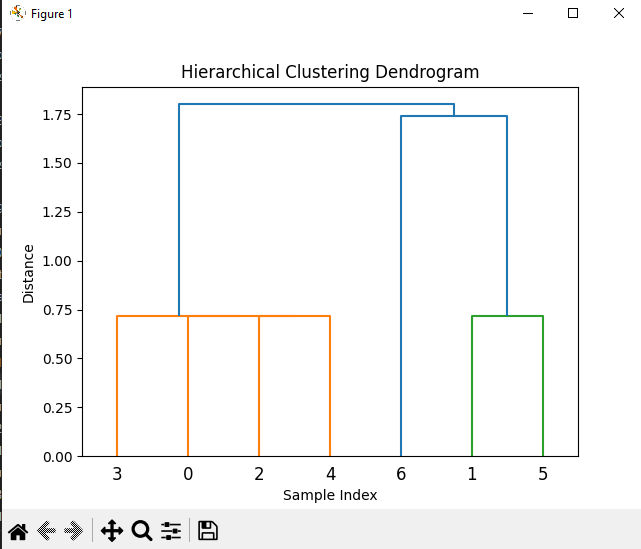
  return 0;

}

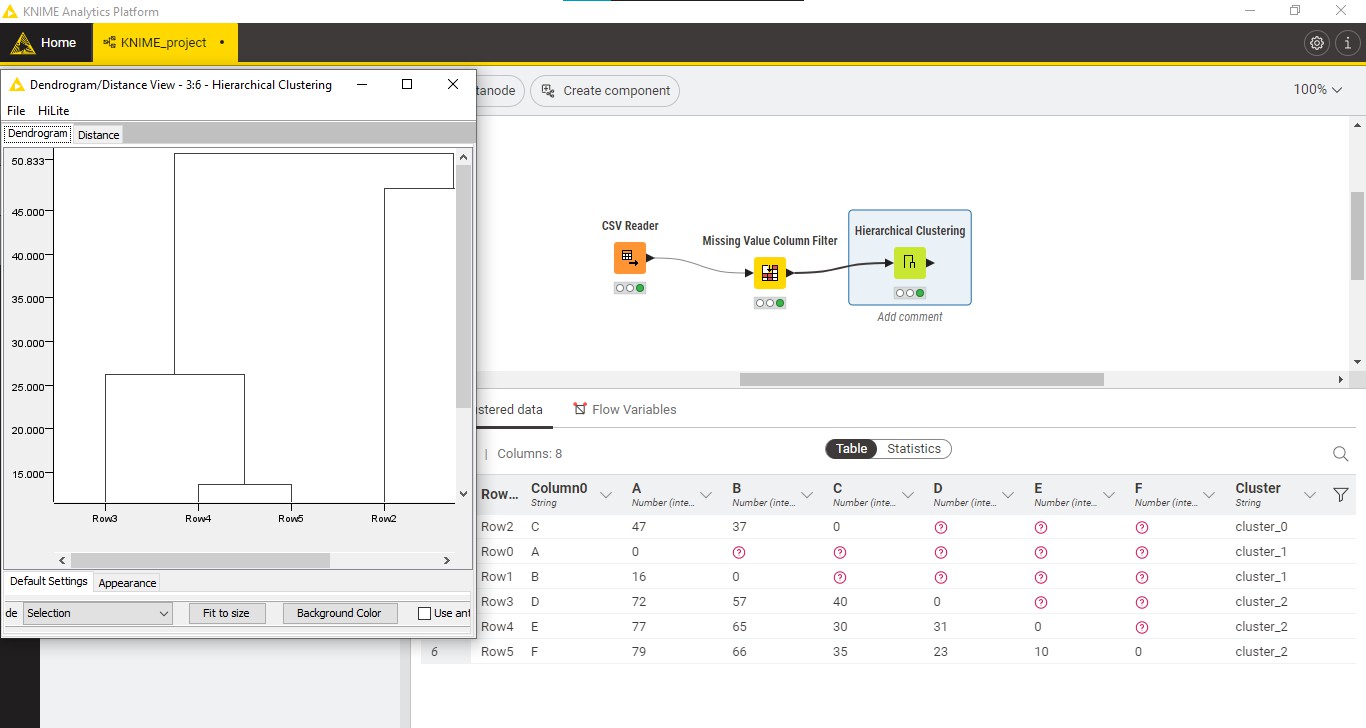
**Input**



**Output**



## KNIME



**Experiment No. 12 Program:**

#include <bits/stdc++.h>

using namespace std;

// Global variables to store data and results

vector<string> sub\_classes;          // Subclasses for analysis

map<string, int> mainClass;          // Count of main class values (e.g., "Yes" and "No")

map<string, unordered\_set<string>> dist\_val; // Distinct values for each subclass

map<string, int> dist\_val\_count;       // Count of distinct values

map<string, map<string, int>> val\_count;   // Count of values for each subclass

// Variables to keep track of minimum Gini index and selected root

double minGini = DBL\_MAX;

string root = "null";

// Output file stream

ofstream fw("gini\_output.csv", ios::out);

// Function to calculate the Gini index and select the root

void calculateGini(string subClass, double mainC\_gini)

{

  double totR = mainClass["Yes"] + mainClass["No"]; // Total count of main class values

  double ent = 0; // Initialize Gini entropy

  // Calculate Gini index for the given subclass

  for (auto dv : dist\_val[subClass])

  {

    double tR = dist\_val\_count[dv];     // Total count for the distinct value

    double pR = val\_count[subClass]["Yes"]; // Count of "Yes" in the subclass

    double nR = val\_count[subClass]["No"];  // Count of "No" in the subclass

    ent += (tR / totR) \* (1 - (pR / tR) \* (pR / tR) - (nR / tR) \* (nR / tR));

  }

  double gini = ent; // Final Gini index

  // Output the Gini index for the subclass

  cout << fixed << setprecision(4); // Set precision to 4 decimal places

  cout << "Gini Index ( " << subClass << "| playGame ) : " << gini << "\n\n";

  fw << fixed << setprecision(4); // Set precision to 4 decimal places in the output file

  fw << "Gini Index ( " << subClass << "| playGame )," << gini << "\n";

  // Check if the current Gini index is the minimum

  if (gini < minGini)

  {

    minGini = gini;

    root = subClass; // Update the root with the minimum Gini index

  }

}

int main()

{

  // Open and read the input file

  ifstream file("gini\_input.csv");

  string line;

  if (!file.is\_open())

  {

    cerr << "Error in opening the input file." << endl;

    return -1;

  }

  int j = 0;            // Line counter

  string main\_class = "playgame"; // Main class attribute name

  while (getline(file, line))

  {

    stringstream str(line);

    string day, outlook, temp, humidity, wind, playGame;

    getline(str, day, ',');

    getline(str, outlook, ',');

    getline(str, temp, ',');

    getline(str, humidity, ',');

    getline(str, wind, ',');

    getline(str, playGame, ',');

    if (j == 0)

    {

      j++;

      sub\_classes.push\_back(day);

      sub\_classes.push\_back(outlook);

      sub\_classes.push\_back(temp);

      sub\_classes.push\_back(humidity);

      sub\_classes.push\_back(wind);

      continue;

    }

    // Store data for calculations

    dist\_val["day"].insert(day);

    dist\_val["outlook"].insert(outlook);

    dist\_val["temp"].insert(temp);

    dist\_val["humidity"].insert(humidity);

    dist\_val["wind"].insert(wind);

    mainClass[playGame]++; // Update the count for the main class value

    dist\_val\_count[day]++; // Update the count for the distinct values

    dist\_val\_count[outlook]++;

    dist\_val\_count[temp]++;

    dist\_val\_count[humidity]++;

    dist\_val\_count[wind]++;

    val\_count[day][playGame]++; // Update the count for values in the subclasses

    val\_count[outlook][playGame]++;

    val\_count[temp][playGame]++;

    val\_count[humidity][playGame]++;

    val\_count[wind][playGame]++;

  }

  // Calculate Gini index for the main class

  double posR = mainClass["Yes"];

  double negR = mainClass["No"];

  double totR = posR + negR;

  double mainC\_gini = 1 - ((posR / totR) \* (posR / totR) + (negR / totR) \* (negR / totR));

  // Output the Gini index for the main class

  cout << fixed << setprecision(4); // Set precision to 4 decimal places

  cout << "Main Class Gini Index : " << mainC\_gini << "\n";

  // Calculate and output the Gini index for each subclass

  for (int i = 1; i < 5; i++)

  {

    calculateGini(sub\_classes[i], mainC\_gini);

  }

  // Output the selected root with the minimum Gini index

  cout << "The selected root for splitting is: " << root << " (Minimum Gini Index)"

     << "\n";

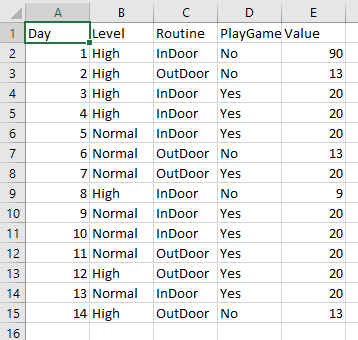
  fw << fixed << setprecision(4); // Set precision to 4 decimal places in the output file

  fw << "The selected root for splitting is: " << root << " (Minimum Gini Index)"

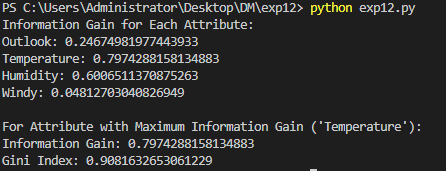
    << "\n";

  return 0;

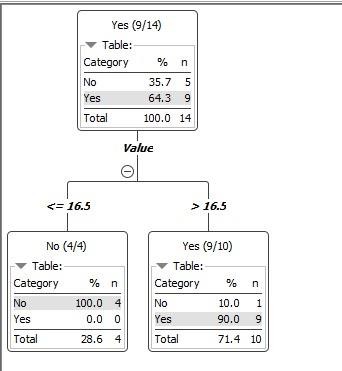
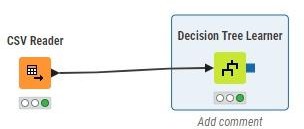
}



**Output:**



## KNIME



**Experiment 13 Program:**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <sstream>

#include <map>

#include <numeric>

using namespace std;

int main()

{

  // Variables to store data

  string line, word;

  ifstream file("bayes.csv");

  string day, outlook, three, four, five, six;

  map<string, double> parent;

  map<string, map<string, map<string, double>>> child;

  int count = 0;

  vector<string> title;

  // Check if the file is open

  if (file.is\_open())

  {

    int i = 0;

    while (file >> line)

    {

      stringstream str(line);

      if (i == 0)

      {

        // Read the column headings

        string heading;

        while (getline(str, heading, ','))

        {

          title.push\_back(heading);

        }

        i++;

        continue;

      }

      vector<string> columns;

      while (getline(str, day, ','))

      {

        columns.push\_back(day);

      }

      int n = columns.size();

      parent[columns[n - 1]]++; // Count the occurrences of the result class

      for (int i = 1; i < n - 1; i++)

      {

        // Count occurrences of attributes given the result class

        child[title[i]][columns[i]][columns[n - 1]]++;

      }

      count++; // Count total instances

    }

    vector<string> resultclass;

    for (auto it : parent)

    {

      resultclass.push\_back(it.first);

    }

    vector<double> output(resultclass.size(), 1);

    for (auto it : child)

    {

      string input;

    again:

      cout << "Enter " << it.first << " condition: ";

      cin >> input;

      auto curr = child[it.first].find(input);

      if (curr == child[it.first].end())

      {

        cout << "No match. Please enter a valid condition.\n";

        goto again;

      }

      for (int i = 0; i < resultclass.size(); i++)

      {

        // Calculate conditional probabilities

        cout << child[it.first][input][resultclass[i]] << " / " << parent[resultclass[i]] << endl;

        double val = child[it.first][input][resultclass[i]] / parent[resultclass[i]];

        output[i] \*= val;

        cout << output[i] << endl;

        cout << "Updated output: " << output[i] << endl;

      }

    }

    for (int i = 0; i < resultclass.size(); i++)

    {

      // Multiply by prior probabilities

      output[i] \*= parent[resultclass[i]] / count;

    }

    double sum = accumulate(output.begin(), output.end(), 0.0);

    // Output results

    cout << "Sum of probabilities: " << sum << endl;

    cout << "Output probabilities:\n";

    for (int i = 0; i < resultclass.size(); i++)

    {

      cout << resultclass[i] << ": " << output[i] << endl;

      cout << "Percentage: " << (output[i] / sum) \* 100 << "%" << endl;

    }

  }

  else

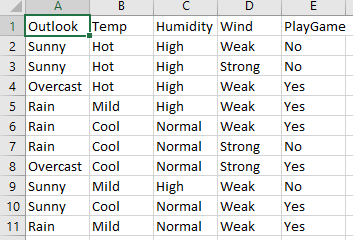
  {

    cout << "Could not open the file." << endl;

  }

  return 0;

}

**Input**

**Output**

