



A REPORT OF

## **“Data Mining Laboratory”**

**Code: 5IT451**

Submitted by

**Miss. Sonali Kaingade (21620002)**

**Email: [sonali.kaingade@walchandsangli.ac.in](mailto:sonali.kaingade@walchandsangli.ac.in)**

**Mobile: 8459557584**

**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 Laboratory (DM Lab) 5IT451 ”**  
submitted by

**MISS. SONALI KAINGADE (21620002)**

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

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

## **Declaration**

I, the undersigned, hereby declare that the BTech report entitled “Data Mining Laboratory (DM Lab) 5IT451” submitted by me to the DM 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.

(Sign)

Date:

**Miss. Sonali Kaingade**

Place: WCE Sangli

## **Acknowledgement**

I feel immense pleasure in submitting 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,

**Miss. Sonali Kaingade**

## Data Mining Lab Book

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**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).	
2	Perform Normalization of data (Min-Max and Z-score).	
3	Perform Binning of data.	
4	Find the Info Gain of an attribute from the given data.	
5	Find the t and d weight of the data.	
6	Find 5 no summary of a dataset.	
7	Find frequent item sets from given transaction data.	
8	Extend program 6, to find association rules.	
9	Find correlation between items/entities.	
10	Distance and cluster.	
11	Agglomerative Hierarchical Single Linkage Clustering.	
12	Attribute for classification A. Gain B. Gini Index	
13	WAP for Bayes classification.	
14	WAP to implement any DM concept on complex data type.	

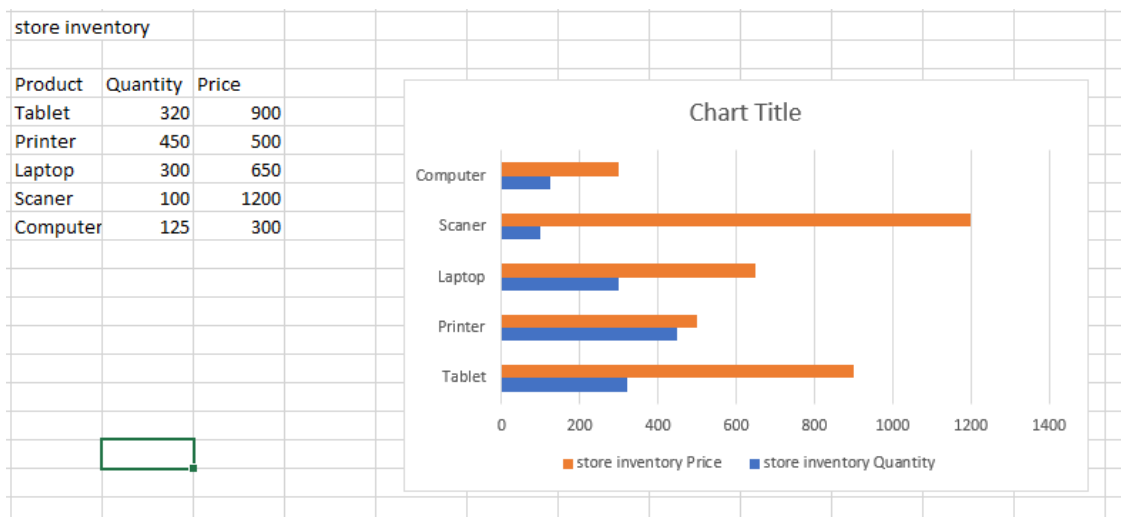
## Experiment No. 1

Name: Sonali Dattatray Kaingade

PRN: 21620002

Title: Study and use of different types of graphs and charts(use MS-XLS).

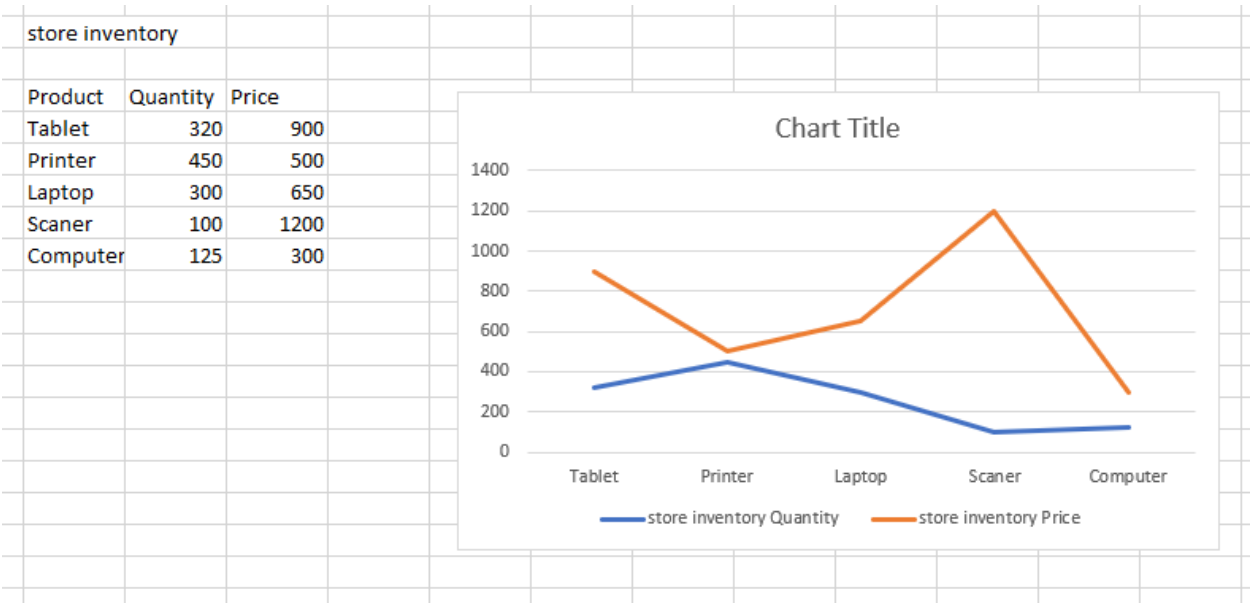
### 1. Bar graph



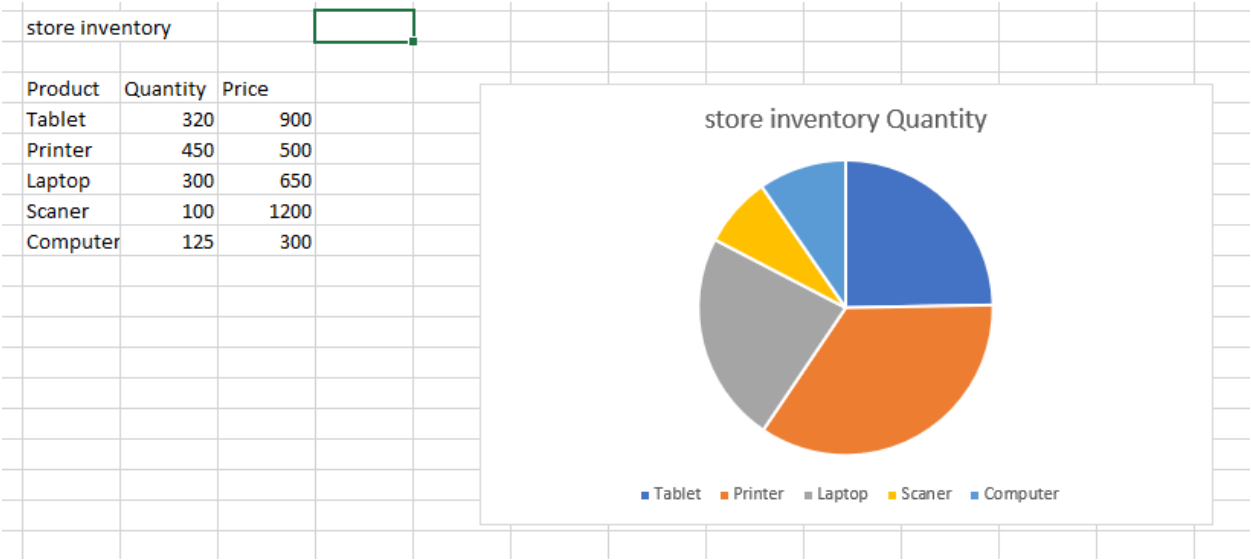
### 2. Column graph



3. Line chart



4. Pie chart



5. Scatter chart





## Experiment no. 2

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** To perform normalization of data (Min-max and z-score).

### 1. Min max normalization

**Code:**

```
#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

using namespace std;

double min_max_scaling(double x, double x_min, double x_max, double x_newmin, double x_newmax)
{
    return ((x - x_min) / (x_max - x_min)) * (x_newmax - x_newmin) + x_newmin;
}

int main() {

    ifstream input_file("input.txt");

    ofstream output_file("output_minmax.txt");

    vector<double> data;

    double value;

    while (input_file >> value) {
```

```
        data.push_back(value);
    }

    double x_min = *min_element(data.begin(), data.end());
    double x_max = *max_element(data.begin(), data.end());

    double x_newmin = 1.0; // New minimum value for scaled data
    double x_newmax = 10.0; // New maximum value for scaled data

    for (const double &x : data) {
        double normalized_value = min_max_scaling(x, x_min, x_max, x_newmin, x_newmax);
        output_file << normalized_value << endl;
    }

    input_file.close();
    output_file.close();

    cout << "output is generated in output_minmax file" << endl;

    return 0;
}
```

**Output:**

**input.txt**

```
minmax.cpp  input.txt  output_minmax.txt
input.txt
1 5
2 10
3 15
4 20
5 25
```

**Output\_minmax.txt**

```
minmax.cpp  input.txt  output_minmax.txt
output_minmax.txt
1 1
2 3.25
3 5.5
4 7.75
5 10
6
```

## 2. Z-score normalization

### Code:

```
#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

#include <cmath>


using namespace std;


double z_score(double x, double mean, double std_dev) {

    return (x - mean) / std_dev;

}


int main() {

    ifstream input_file("input.txt");

    ofstream output_file("output_zscore.txt");


    vector<double> data;

    double value;


    while (input_file >> value) {

        data.push_back(value);

    }
```

```
double sum = 0.0;

for (const double &x : data) {

    sum += x;

}

double mean = sum / data.size();


double squared_diff_sum = 0.0;

for (const double &x : data) {

    squared_diff_sum += pow(x - mean, 2);

}

double std_dev = sqrt(squared_diff_sum / data.size());


for (const double &x : data) {

    double normalized_value = z_score(x, mean, std_dev);

    output_file << normalized_value << endl;

}


input_file.close();

output_file.close();


cout << "output is generated in output_zscore file" << endl;


return 0;

}
```

**Output:**

**input.txt:**

```
minmax.cpp  input.txt  output_minmax.txt
input.txt
1 5
2 10
3 15
4 20
5 25
```

**Output\_zscore.txt:**

```
input.txt  zscore.cpp  output_zscore.txt
output_zscore.txt
1 -1.41421
2 -0.707107
3 0
4 0.707107
5 1.41421
6
```

## Knime:

## Min-Max Normalization:

Repository > Results

Q CSV

IO Write Read Analytics Mining +83

CSV Reader CSV Writer

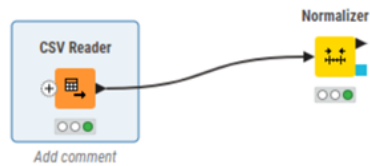
More advanced nodes

Statistics Decision Tree Predictor Decision Tree Learner

Scorer Regression Predictor DB Table Selector

Value Counter Cross Joiner Column Splitter

File Reader Random Forest



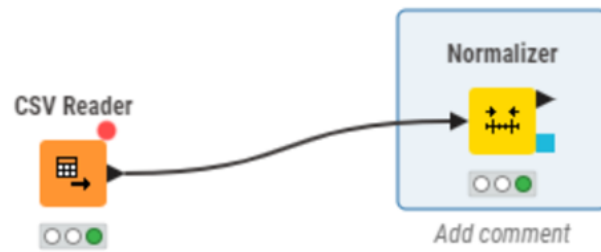
1: File Table

Flow Variables

Rows: 4 | Columns: 1

Table Statistics

#	Row...	12
		Number (integer)
1	Row0	25
2	Row1	18
3	Row2	30
4	Row3	15



► 1: Normalized table    ■ 2: Normalize Model    Flow Variables

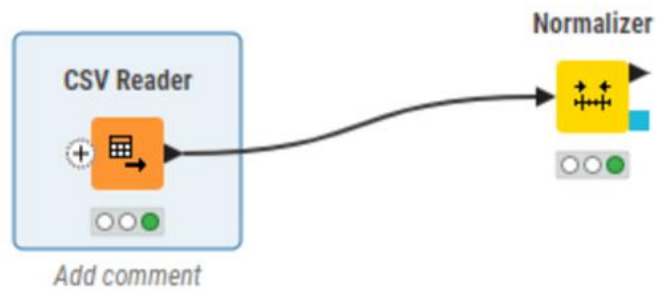
Rows: 4 | Columns: 1

Table Statistics

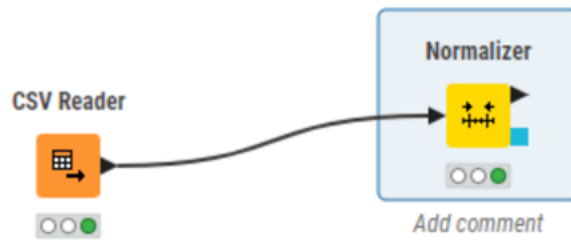
#	Row...	12 <i>Number (double)</i>
1	Row0	16.667
2	Row1	12
3	Row2	20
4	Row3	10



## Z - Score Normalization:



1: File Table			Flow Variables	
Rows: 4   Columns: 1			Table	Statistics
#	Row...	12	Number (integer)	
1	Row0	25		
2	Row1	18		
3	Row2	30		
4	Row3	15		



▶ 1: Normalized table    ■ 2: Normalize Model    📄 Flow Variables

Rows: 4   |   Columns: 1

Table   Statistics

#	Row...	12 <i>Number (double)</i>
1	Row0	0.442
2	Row1	-0.59
3	Row2	1.18
4	Row3	-1.032

### Experiment no. 3

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** To perform Binning of data

**Code:**

```
#include <iostream>

#include <fstream>

#include <vector>

#include<algorithm>

#include <climits>

#include<cmath>

using namespace std;

//equal frequency

vector<vector<int>> equi_frequency(vector<int> data,double m)

{

    double n=data.size();

    double ele=ceil(n/m);

    vector<vector<int>> totalbins;

    for(int i=0;i<m;i++)

    {

        vector<int> bin;

        for(int j=i*ele;j<(i+1)*ele;j++)
```

```

{
    if(j>=n)
    {
        break;
    }
    bin.push_back(data[j]);
}
totalbins.push_back(bin);
}
return totalbins;
}

```

//equal width

```
vector<vector<int>>equi_width(vector<int> data,int m)
```

```

{
    int n=data.size();

    int min_ele=INT_MAX;
    int max_ele=INT_MIN;

    for(int i=0;i<data.size();i++)
    {
        min_ele= min(min_ele,data[i]);
        max_ele= max(max_ele,data[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 k:data)

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

        {
            temp.push_back(k);

        }

    }

    arri.push_back(temp);

}

return arri;

}

```

```

// 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()
{

    ifstream inputf("input.csv");

    vector<int> data;

    int val;

    while(inputf>>val)
    {

```

```

        data.push_back(val);
    }

    sort(data.begin(),data.end());

    int method,m;

    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 = equi_frequency(data, m);
        writeCSV("output_equi_frequency.csv", freqBins);
    }

    else if (method == 2)
    {
        vector<vector<int>> widthBins = equi_width(data, m);
        writeCSV("output_equi_width.csv", widthBins);
    }

    else
    {
        cout << "Invalid method choice." << endl;
    }

```

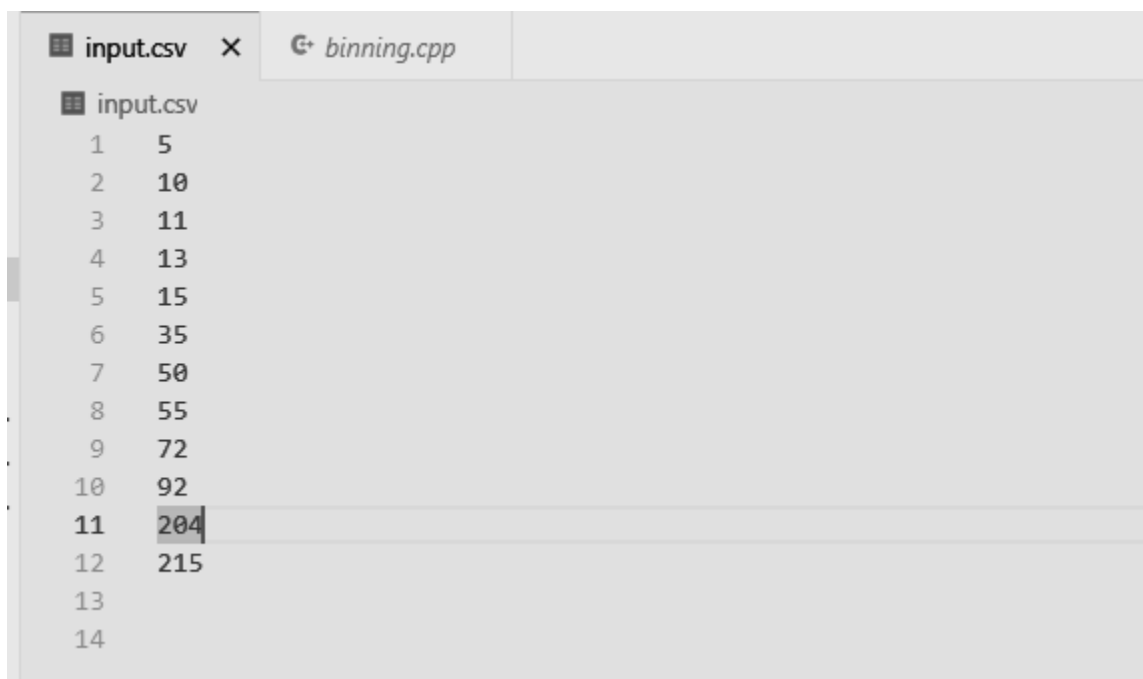
```
}
```

```
return 0;
```

```
}
```

**Output:**

**Input.csv**



The image shows a code editor window with two tabs: 'input.csv' and 'binning.cpp'. The 'input.csv' tab is active, displaying a list of 14 rows. The first column contains integers from 1 to 14, and the second column contains corresponding values. The value '204' in row 11 is highlighted with a mouse cursor.

Row	Value
1	5
2	10
3	11
4	13
5	15
6	35
7	50
8	55
9	72
10	92
11	204
12	215
13	
14	



## Output.csv

### 1.Equal frequency

input.csv	output_equi_frequency.csv X
output_equi_frequency.csv	
1	Bin 1: 5,10,11,13,
2	Bin 2: 15,35,50,55,
3	Bin 3: 72,92,204,215,
4	

### 2. Equal width

input.csv	output_equi_width.csv X
output_equi_width.csv	
1	Bin 1: 5,10,11,13,15,35,50,55,72,
2	Bin 2: 92,
3	Bin 3: 204,215,
4	

Knime:

### Binning methods:

1. Equi- Frequency:

Dialog - 3:1 - Auto-Binner

File

Auto Binner Settings | Number Format Settings | Flow Variables | Job Manager Selection | Memory Policy

☒ Manual Selection ☐ Wildcard/Regex Selection

Exclude

Filter

No columns in this list

☒ Enforce exclusion

>

>>

<

<<

Include

Filter

12

☐ Enforce inclusion

Binning Method

☒ Fixed number of bins

Number of bins: 3

Equal: frequency

☐ Sample quantiles

Quantiles (comma separated): 0.0, 0.25, 0.5, 0.75, 1.0

Bin Naming

☒ Numbered e.g.: Bin 1, Bin 2, Bin 3

☐ Borders e.g.: [-10,0], (0,10], (10,20]

☐ Midpoints e.g.: -5, 5, 15

OK Apply Cancel ?



► 1: Binned Data    ■ 2: PMML Processing Fragment    Flow Variables

Rows: 9 | Columns: 2

Table    Statistics

#	Row...	12 <i>Number (integer)</i>	12 [Binned] <i>String</i>
1	Row0	15	Bin 1
2	Row1	18	Bin 1
3	Row2	21	Bin 1
4	Row3	22	Bin 2
5	Row4	25	Bin 2
6	Row5	27	Bin 2
7	Row6	30	Bin 3
8	Row7	32	Bin 3
9	Row8	38	Bin 3

2. Equi width:

Dialog - 3:1 - Auto-Binner

File

Auto Binner Settings | Number Format Settings | Flow Variables | Job Manager Selection | Memory Policy

☒ Manual Selection ☐ Wildcard/Regex Selection

Exclude

No columns in this list

☒ Enforce exclusion

>

>>

<

<<

Include

12

☐ Enforce inclusion

Binning Method

☒ Fixed number of bins

Number of bins:

Equal:

☐ Sample quantiles

Quantiles (comma separated):

Bin Naming

☒ Numbered e.g.: Bin 1, Bin 2, Bin 3

☐ Borders e.g.: [-10,0], (0,10], (10,20]

☐ Midpoints e.g.: -5, 5, 15

OK Apply Cancel ?

CSV Reader



Auto-Binner



Add comment

► 1: Binned Data

■ 2: PMML Processing Fragment

📄 Flow Variables

Rows: 9 | Columns: 2

Table

Statistics

#	Row...	12 <i>Number (integer)</i>	12 [Binned] <i>String</i>
1	Row0	15	Bin 1
2	Row1	18	Bin 1
3	Row2	21	Bin 1
4	Row3	22	Bin 1
5	Row4	25	Bin 2
6	Row5	27	Bin 2
7	Row6	30	Bin 2
8	Row7	32	Bin 3
9	Row8	38	Bin 3

## Experiment no. 4

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Find info gain of an attribute from given data.

**Code:**

```
#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 file:**

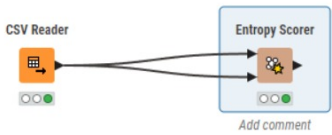
	A	B	C	D	E	F	
1	Day	Level	Routine	PlayGame	Value		
2		1 High	InDoor	No	FALSE		
3		2 High	OutDoor	No	FALSE		
4		3 High	InDoor	Yes	TRUE		
5		4 High	InDoor	Yes	TRUE		
6		5 Normal	InDoor	Yes	TRUE		
7		6 Normal	OutDoor	No	FALSE		
8		7 Normal	OutDoor	Yes	TRUE		
9		8 High	InDoor	No	FALSE		
10		9 Normal	InDoor	Yes	TRUE		
11		10 Normal	InDoor	Yes	TRUE		
12		11 Normal	OutDoor	Yes	TRUE		
13		12 High	OutDoor	Yes	TRUE		
14		13 Normal	InDoor	Yes	TRUE		
15		14 High	OutDoor	No	FALSE		
16							
17							


## Output:


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS




PS E:\second year engg IT\7th sem\dm lab\13th experiment> cd ..
PS E:\second year engg IT\7th sem\dm lab> cd '.\4th experiment\'
PS E:\second year engg IT\7th sem\dm lab\4th experiment> g++ infogain.cpp -o infog
PS E:\second year engg IT\7th sem\dm lab\4th experiment> ./infog
Enter Child Column Number: 2
Parent Entropy: 0.940286
Weighted Child Entropy: 0.78845
Information Gain: 0.151836
Overall Information Gain: 0.151836
PS E:\second year engg IT\7th sem\dm lab\4th experiment> █
```

## knime:



1: Quality Table  Flow Variables

Rows: 4 | Columns: 4 Table Statistics 

#	Row...	Size Number (integer)	Entropy Number (double)	Normalized Entropy Number (double)	Quality Number (double)
1	Over...	4	0	0	
2	Sunny	5	0.971	0.971	
3	Rain	5	0.971	0.971	
4	Overall	14	0.694	0.694	0.306

## Experiment no. 5

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Find t and d weight of data.

**Code:**

```
#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;

    int total1 = 0;

    for (const auto &rowEntry : rowTotal)
    {
        total1 += rowEntry.second;
    }

    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) / total1) * 100 << "%" << endl;

}

fw << "Total,";

int total = 0;

for (const auto &colEntry : columnTotal)
{
    total += colEntry.second;
}

for (const auto &colEntry : columnTotal)
{
    fw << colEntry.second << ",";

```



```

        fw << ((float)colEntry.second / total) * 100 << "%,";

        fw << "100%,";

    }

    fw << total << ",100%, 100%" << endl;

}

int main()

{

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

    map<string, int> columnTotal;

    map<string, int> rowTotal;


    readData("inputdata.csv", cellData, columnTotal, rowTotal);

    writeResult("outputdata.csv", cellData, columnTotal, rowTotal);


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


    return 0;

}

```

**Result:**

**Input dataset:**

	A	B	C	D
1	class	workplace	count	
2	musician	india	180	
3	musician	USA	120	
4	dancer	india	20	
5	dancer	USA	80	
6				
7				
8				

**Output:**

[illegible]

## Experiment no. 6

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Find 5 no. summary of a dataset.

**Code:**

```
#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;

}

```

**Result:**

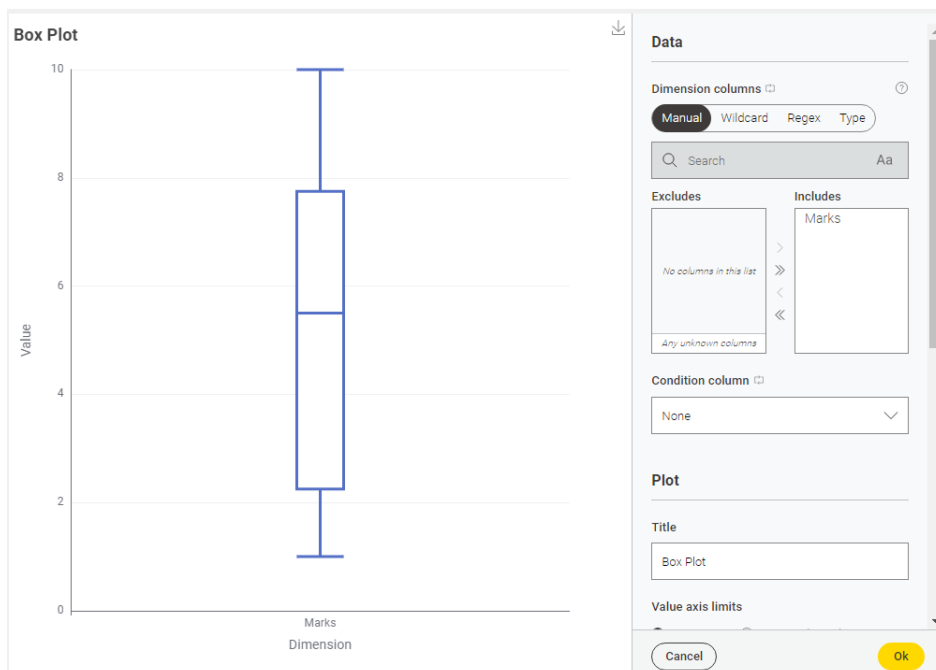
**Input:**

	A	B	C	
1	Marks			
2	2			
3	4			
4	5			
5	8			
6	10			
7	9			
8	1			
9	1			
10	3			
11	6			
12	6			
13	7			
14				

## Output:

	A	B	C
1	Minimum value:	1	
2	First Quartile (Q1) value:	2.5	
3	Median value:	5.5	
4	Third Quartile (Q3) value:	7.5	
5	Maximum value:	10	
6			
7			
8			

knime:





## Experiment no. 7

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Find frequent itemset from given transaction data.

**Code:**

```
#include <bits/stdc++.h>

#include <map>

using namespace std;

double minfre;          // Minimum frequency for itemsets to be considered frequent.

vector<set<string>> datatable; // A vector to store transaction data.

set<string> products;    // Set to store unique products/items in the transactions.

map<string, int> freq;    // A map to store the frequency of each product/item.

// 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);

            // mergig v1 and v2 like lattice i.e permutations and combinations type

            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()
{
    ifstream fin("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 << "Enter min 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;
}

```

**Result:**

**Input:**

	A	B	C	D	
1	A	B	C		
2	A	D			
3	B	C			
4	A	B	D		
5	A	B	C	D	
6					
7					
8					

## Output:

```
PS E:\second year engg IT\7th sem\dm lab\7th experiment> g++ frequentitem.cpp -o freit
PS E:\second year engg IT\7th sem\dm lab\7th experiment> ./freit
Frequency % :60
No of transactions: 5
Min frequency:3

Frequent 1 -item set :
{A} 4
{B} 4
{C} 3
{D} 3

Frequent 2 -item set :
{A B} 3
{A D} 3
{B C} 3
PS E:\second year engg IT\7th sem\dm lab\7th experiment>
```

## Frequent-itemsets are:

	A	B
1	{A B}	
2	{A D}	
3	{B C}	
4		
5		

## Experiment no. 8

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Extend program 7, to find association rule.

**code:**

```
#include <bits/stdc++.h>

#include <map>

using namespace std;

ifstream fin;

double minfre;

vector<set<string>> datatable;

set<string> products;

map<string, int> freq;

double confidence;

// Function to split a string into words
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 a vector of strings, excluding the one at a given 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 of strings
```

```
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;
```

```
}
```

```
// Generate candidate itemsets for Apriori
```

```
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;
}

// Count occurrences of a set of items in the dataset
int countOccurrences(vector<string> v)
{
    int count = 0;

```

```

for (auto s : datatable)
{
    bool present = true;

    for (auto x : v)
    {
        if (s.find(x) == s.end())
        {
            present = false;
            break;
        }
    }

    if (present)
        count++;
}

return count;
}

ofstream fw1("association_output.csv", ios::out);

// Generate subsets of items for association rule generation
void subsets(vector<string> items, vector<string> v1, vector<string> v2, int idx)

```

```

{
    if (idx == items.size())
    {
        if (v1.size() == 0 || v2.size() == 0)
            return;

        int count1 = countOccurrences(items); // Total support
        int count2 = countOccurrences(v1);

        double conf = (((double)count1) / count2) * 100;

        if (conf >= confidence)
        {
            fw1 << "Association Rule: { ";
            for (auto s : v1)
            {
                fw1 << s << " ";
            }
            fw1 << "}->{";
            for (auto s : v2)
            {
                fw1 << s << " ";
            }
            fw1 << "}, Confidence: " << conf << "%" << endl;
        }
    }
}

```



```

        return;
    }

    v1.push_back(items[idx]);
    subsets(items, v1, v2, idx + 1);

    v1.pop_back();
    v2.push_back(items[idx]);
    subsets(items, v1, v2, idx + 1);
    v2.pop_back();
}

// Generate association rules from frequent itemsets
void generateAssociationRules(set<string> freqItems)
{
    for (auto it = freqItems.begin(); it != freqItems.end(); it++)
    {
        vector<string> items = wordsof(*it);

        subsets(items, {}, {}, 0);
    }
}

int main()

```

```

{
    fin.open("association_input.csv", ios::in);

    if (!fin.is_open())
    {
        cerr << "Error in opening file." << endl;
        return 1;
    }

    cout << "Enter Minimum Support (%): ";
    cin >> minfre;

    cout << "Enter Minimum Confidence (%): ";
    cin >> confidence;

    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);
    }
}

```

```

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

        freq[*it]++;
    }
}

fin.close();

```

```

cout << "Number of transactions: " << datatable.size() << endl;

minfre = minfre * datatable.size() / 100;

cout << "Minimum Frequency Threshold: " << 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());

    q.pop();
}

```

```

int pass = 1;

cout << "Frequent " << pass++ << "-item set: " << endl;

```

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

```
    cout << "{" << *it << "}" - Support: " << freq[*it] << endl;
```

```
int i = 2;
```

```
set<string> prev = cloneit(products);
```

```
while (i)
```

```
{
```

```
    set<string> cur = apriori_gen(prev, i - 1);
```

```
    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 << "\nFrequent " << pass++ << "-item set: " << endl;
        for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)
            cout << "{" << *it << "} - Support: " << freq[*it] << endl;

        prev = cloneit(cur);
        i++;
    }

    generateAssociationRules(prev);

    cout << "Association rules generated successfully." << endl;

    return 0;
}

```

**Output:**

**input.csv:**

association\_input.csv

```
1 I1,I2,I5
2 I2,I4
3 I2,I3
4 I1,I2,I4
5 I1,I3
6 I2,I3
7 I1,I3
8 I1,I2,I3,I5
9 I1,I2,I3
```

output.csv:

association\_output.csv

```
1 Association Rule: { I1 I2 I3 } -> {I5 } , Confidence: 50%
2 Association Rule: { I1 I2 I5 } -> {I3 } , Confidence: 50%
3 Association Rule: { I1 I2 } -> {I3 I5 } , Confidence: 25%
4 Association Rule: { I1 I3 I5 } -> {I2 } , Confidence: 100%
5 Association Rule: { I1 I3 } -> {I2 I5 } , Confidence: 25%
6 Association Rule: { I1 I5 } -> {I2 I3 } , Confidence: 50%
7 Association Rule: { I1 } -> {I2 I3 I5 } , Confidence: 16.6667%
8 Association Rule: { I2 I3 I5 } -> {I1 } , Confidence: 100%
9 Association Rule: { I2 I3 } -> {I1 I5 } , Confidence: 25%
10 Association Rule: { I2 I5 } -> {I1 I3 } , Confidence: 50%
11 Association Rule: { I2 } -> {I1 I3 I5 } , Confidence: 14.2857%
12 Association Rule: { I3 I5 } -> {I1 I2 } , Confidence: 100%
13 Association Rule: { I3 } -> {I1 I2 I5 } , Confidence: 16.6667%
14 Association Rule: { I5 } -> {I1 I2 I3 } , Confidence: 50%
15
```

```

PS C:\Users\USER\Desktop\dm lab\8th experiment> g++ exp8.cpp -o e
PS C:\Users\USER\Desktop\dm lab\8th experiment> ./e
Enter Minimum Support (%): 3
Enter Minimum Confidence (%): 2
Number of transactions: 9
Minimum Frequency Threshold: 0.27
Frequent 1-item set:
{A} - Support: 6
{B} - Support: 7
{C} - Support: 6
{D} - Support: 2
{E} - Support: 2

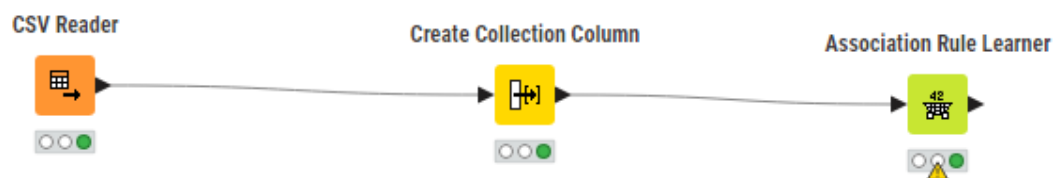
Frequent 2-item set:
{A B} - Support: 4
{A C} - Support: 4
{A D} - Support: 1
{A E} - Support: 2
{B C} - Support: 4
{B D} - Support: 2
{B E} - Support: 2
{C E} - Support: 1

Frequent 3-item set:
{A B C} - Support: 2
{A B D} - Support: 1
{A B E} - Support: 2
{A C E} - Support: 1
{B C E} - Support: 1

Frequent 4-item set:
{A B C E} - Support: 1
Association rules generated successfully.
PS C:\Users\USER\Desktop\dm lab\8th experiment> █

```

**knime:**





## Experiment no. 9

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Find correlation between items/entities.

**code:**

```
#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;
}

```

**Output:**

**Input.csv:**

Tid	M	T	W	Th	F	S		
1	1	1	1	0	0	1	0	
2	0	1	1	1	0	0	1	
3	1	1	1	1	0	1	1	
4	0	0	0	0	1	1	1	

Output.csv:

outputfile.csv X

outputfile.csv

```
1 vector 1 and vector 2 are negatively correlated: 0.583333
2 vector 1 and vector 3 are positively correlated: 1.05
3 vector 1 and vector 4 are negatively correlated: 0.583333
4 vector 2 and vector 3 are positively correlated: 1.4
5 vector 2 and vector 4 are negatively correlated: 0.777778
6 vector 3 and vector 4 are negatively correlated: 0.933333
7
```

## Experiment no. 10

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Distance and cluster

- Assume some points (multidimensional)

Compute centre of cluster assuming all points belonging to one cluster.

Find distance of all points with obtained cluster centre using suitable distance function

Display results in the form of upper or lower triangular matrix

**code:**

```
#include <bits/stdc++.h>

using namespace std;

double finddist(pair<double,double>p1, pair<double,double>p2){

    double x1 = p1.first;

    double x2 = p2.first;

    double y1 = p1.second;

    double y2 = p2.second;

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

}

int main(){

    string line;

    ifstream input("cluster_input.csv");

    vector<pair<double, double>>points;

    int j = 0;
```

```
while(getline(input,line)){  
    stringstream str(line);  
  
    if(j==0){  
        j++;  
        continue;  
    }  
  
    string name,x,y;  
    getline(str,name,',');  
    getline(str,x,',');  
    getline(str,y);  
  
    points.push_back({stoi(x), stoi(y)});  
}  
input.close();  
  
double x_sum, y_sum, x_mean, y_mean;  
int n = points.size();  
for(auto it : points){  
    x_sum += it.first;  
    y_sum += it.second;  
}  
x_mean = x_sum/n;  
y_mean = y_sum/n;
```

```

//calculate distance of each point from every point

ofstream output("output.csv");

output<<" "<<"p1," <<"p2,"<<"p3,"<<"p4,"<<"p5"<<endl;

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

    output<<"p"<<i+1<<" ";

    for(int j = 0; j<=i;j++){

        if(i==j){

            output<<"0,";

        }

        else{

            double ans = finddist(points[i],points[j]);

            output<<ans<<" ";

        }

    }

    output<<endl;

}

output<<"centroid,";

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

    double ans = finddist(points[i],{x_mean,y_mean});

    output<<ans<<" ";

}

cout << endl;

output.close();

```



```
}
```

**Output:**

**Input.csv:**

cluster_input.csv	
cluster_input.csv	
1	Points,x,y
2	p1,10,40
3	p2,20,10
4	p3,15,20
5	p4,25,30
6	p5,15,5
7	

**Output.csv:**

output.csv	
output.csv	
1	,p1,p2,p3,p4,p5
2	p1,0,
3	p2,31.6228,0,
4	p3,20.6155,11.1803,0,
5	p4,18.0278,20.6155,14.1421,0,
6	p5,35.3553,7.07107,15,26.9258,0,
7	centroid,20.2485,11.4018,2.23607,12.0416,16.1245,

## Knime:

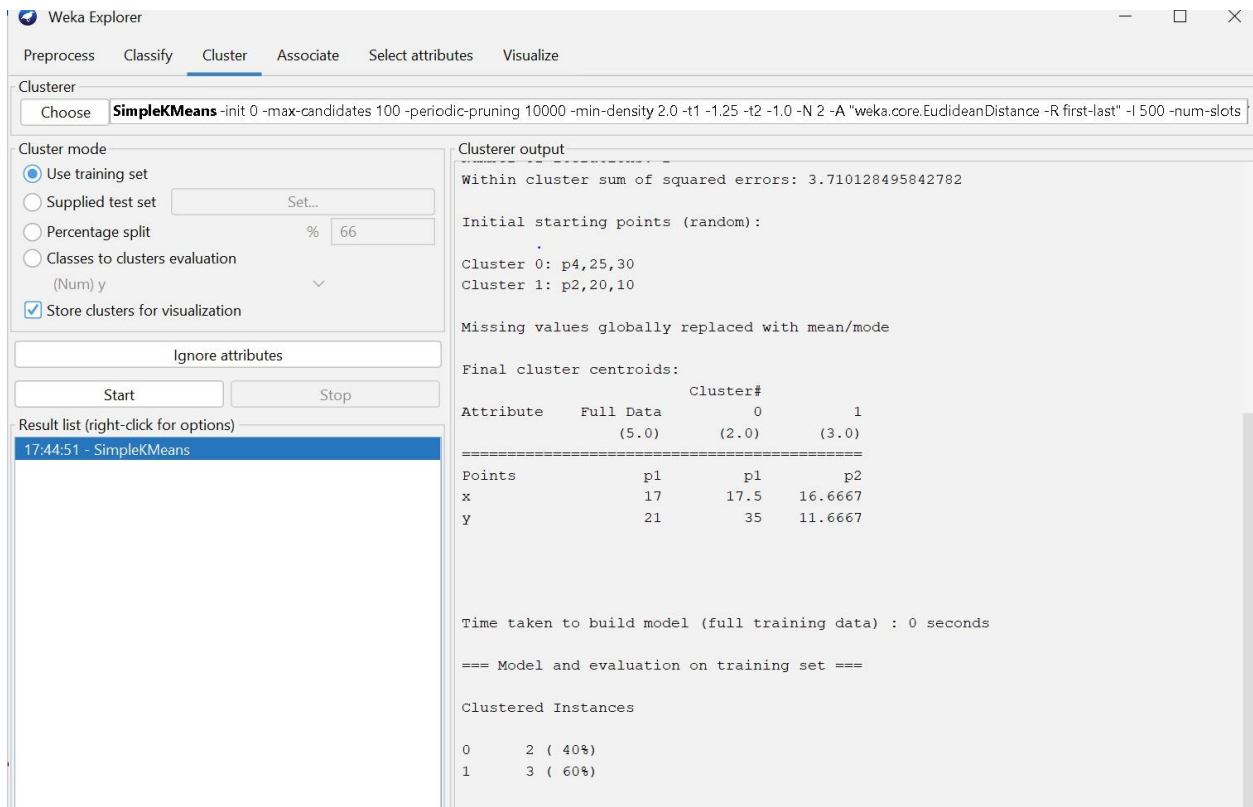


The Knime workflow consists of two nodes: a **CSV Reader** and a **k-Means** node. The CSV Reader is connected to the k-Means node. Below the k-Means node, there is a comment box that says "Add comment".

The k-Means node output is displayed in a table view with 5 rows and 4 columns. The columns are: #, Row..., points, x, y, and Cluster. The data is as follows:

#	Row...	points	x	y	Cluster
1	Row0	p1	10	40	cluster_0
2	Row1	p2	20	10	cluster_1
3	Row2	p3	15	20	cluster_2
4	Row3	p4	25	30	cluster_2
5	Row4	p5	15	5	cluster_1

## Weka:



The Weka Explorer interface shows the **SimpleKMeans** clustering algorithm settings. The **Clusterer** dropdown is set to **SimpleKMeans**. The **Cluster mode** is set to **Use training set**. The **Ignore attributes** field is empty. The **Start** button is visible.

The **Clusterer output** pane displays the following information:

Within cluster sum of squared errors: 3.710128495842782

Initial starting points (random):

Cluster 0: p4,25,30  
Cluster 1: p2,20,10

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Full Data	Cluster# 0	Cluster# 1
	(5.0)	(2.0)	(3.0)

Points

	p1	p1	p2
x	17	17.5	16.6667
y	21	35	11.6667

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

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

Clustered Instances

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

## Experiment no. 11

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Write a program for Agglomerative Hierarchical clustering using single linkage method

**code:**

```
#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;
}

```

**Result:**

**Input.csv:**

	A	B	C	D	E	F	
A	0						
B	16	0					
C	47	37	0				
D	72	57	40	0			
E	77	65	30	31	0		
F	79	66	35	23	10	0	

**Output.csv:**

**1.**

	A	B	C	D	FE	
A	0					
B	16	0				
C	47	37	0			
D	72	57	40	0		
FE	77	65	30	23	0	

2.

	BA	C	D	FE		
BA	0					
C	37	0				
D	57	40	0			
FE	65	30	23	0		

3.

	BA	C	FED			
BA	0					
C	37	0				
FED	57	30	0			

4.

	BA	FEDC	
BA	0		
FEDC	37	0	

final output:

F & E				
B & A				
FE & D				
FED & C				

knife:



► 1: Clustered data Flow Variables

Rows: 6 | Columns: 8

Table Statistics

#	Row...	Column0 String	A Number (inte...	B Number (inte...	C Number (inte...	D Number (inte...	E Number (inte...	F Number (inte...	Cluster String
1	Row2	C	47	37	0	?	?	?	cluster_0
2	Row0	A	0	?	?	?	?	?	cluster_1
3	Row1	B	16	0	?	?	?	?	cluster_1
4	Row3	D	72	57	40	0	?	?	cluster_2
5	Row4	E	77	65	30	31	0	?	cluster_2
6	Row5	F	79	66	35	23	10	0	cluster_2

## Experiment no. 12

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** Attribute for classification ,Write a program to find

A. Gain

B. Gini index

For categorical and numerical values.

**code:**

```
#include <bits/stdc++.h>

using namespace std;

vector<string>attributes;

map<string,int>class_attri_cnt;

map<string,unordered_set<string>>values_in_attributes;

map<string,int>attribute_cnt;

map<string,map<string,int>>attribute_yn_cnt;

double calculateGain(string attri_name, double entropy){

    double info = 0.0;

    unordered_set<string>sets = values_in_attributes[attri_name];

    for(auto it : sets){

        double total = class_attri_cnt["Yes"]+class_attri_cnt["No"];

        double cnt_attri = attribute_cnt[it];

        double cnt_attri_yes = attribute_yn_cnt[it]["Yes"];

        double cnt_attri_no = attribute_yn_cnt[it]["No"];
```

```

    if(cnt_attri_yes != 0){
        info += cnt_attri/total*(-cnt_attri_yes/cnt_attri * log2(cnt_attri_yes/cnt_attri));
    }

    if(cnt_attri_no != 0){
        info += cnt_attri/total*( -cnt_attri_no/cnt_attri * log2(cnt_attri_no/cnt_attri)) ;
    }

}

double gain = entropy - info;

return gain;
}

```

```

double calculategini(string attri_name){
    double gini = 0.0;
    unordered_set<string>sets = values_in_attributes[attri_name];
    //cout << sets.size() << endl;
    for(auto it : sets){
        double total = class_attri_cnt["Yes"]+class_attri_cnt["No"];
        double cnt_attri = attribute_cnt[it];
        double cnt_attri_yes = attribute_yn_cnt[it]["Yes"];
        double cnt_attri_no = attribute_yn_cnt[it]["No"];
    }
}

```

```
    gini += cnt_attri/total * (1-(cnt_attri_yes/cnt_attri)*(cnt_attri_yes/cnt_attri)-  
(cnt_attri_no/cnt_attri)*(cnt_attri_no/cnt_attri));
```

```
}
```

```
    return gini;
```

```
}
```

```
int main(){
```

```
    ifstream input("info-gain.csv");
```

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

```
    int j = 0;
```

```
    while(getline(input,line)){
```

```
        stringstream str(line);
```

```
        getline(str,day,',');
```

```
        getline(str,outlook,',');
```

```
        getline(str,temp,',');
```

```
        getline(str,humidity,',');
```

```
        getline(str,wind,',');
```

```
        getline(str,playGame,'.');
```

```
        if(j == 0){
```

```
            j++;
```



```
attributes.push_back(day);  
attributes.push_back(outlook);  
attributes.push_back(temp);  
attributes.push_back(humidity);  
attributes.push_back(wind);  
attributes.push_back(playGame);  
continue;  
}
```

```
class_attri_cnt[playGame]++;
```

```
values_in_attributes["outlook"].insert(outlook);  
values_in_attributes["temp"].insert(temp);  
values_in_attributes["humidity"].insert(humidity);  
values_in_attributes["wind"].insert(wind);
```

```
attribute_cnt[outlook]++;  
attribute_cnt[temp]++;  
attribute_cnt[humidity]++;  
attribute_cnt[wind]++;
```

```
attribute_yn_cnt[outlook][playGame]++;  
attribute_yn_cnt[temp][playGame]++;  
attribute_yn_cnt[humidity][playGame]++;  
attribute_yn_cnt[wind][playGame]++;
```

```

}

double yes_cnt = class_attri_cnt["Yes"];

double no_cnt = class_attri_cnt["No"];

double total = yes_cnt + no_cnt;

//cout << yes_cnt << no_cnt << total << endl;

double entropy_ca = (-yes_cnt/total * log2(yes_cnt/total)-no_cnt/total *
log2(no_cnt/total)); //entropy of class attribute

cout << "Entropy of class attribute is: " << entropy_ca << endl;

ofstream output("gain_output.csv");

output << "Attributes"<<"," << "Gain" << "," << "Gini"<< endl;

for(int i = 1; i<5; i++){

    double gain = calculateGain(attributes[i],entropy_ca);

    double gini = calculategini(attributes[i]);

    output << attributes[i]<<"," << gain << "," << gini<< endl;

}

}

```

**Result:**

Input.csv:

day	outlook	temp	humidity	wind	playGame		
1	Sunny	Hot	High	Weak	No.		
2	Sunny	Hot	High	Strong	No.		
3	Overcast	Hot	High	Weak	Yes.		
4	Rain	Mild	High	Weak	Yes.		
5	Rain	Cool	Normal	Weak	Yes.		
6	Rain	Cool	Normal	Strong	No.		
7	Overcast	Cool	Normal	Strong	Yes.		
8	Sunny	Mild	High	Weak	No.		
9	Sunny	Cool	Normal	Weak	Yes.		
10	Rain	Mild	Normal	Weak	Yes.		
11	Sunny	Mild	Normal	Strong	Yes.		
12	Overcast	Mild	High	Strong	Yes.		
13	Overcast	Hot	Normal	Weak	Yes.		
14	Rain	Mild	High	Strong	No.		

Output.csv:

Attributes	Gain	Gini		
outlook	0.24675	0.342857		
temp	0.029223	0.440476		
humidity	0.151836	0.367347		
wind	0.048127	0.428571		

## Experiment no. 13

**Name:** Sonali Dattatray Kaingade

**PRN:** 21620002

**Title:** WAP for Baye's classification

**code:**

```
#include <bits/stdc++.h>

using namespace std;

map<string,int>class_attri_cnt;

vector<string>attributes;

map<string,map<string,map<string,int>>>>attribute_yn_cnt;

double calculateProbability(string Outlook,string Temp,string Humidity,string Wind,string playgame){

    for(auto it: attributes){

        // cout << it << " ";

    }

    double yes_cnt = class_attri_cnt["Yes"];

    double no_cnt = class_attri_cnt["No"];

    double total = yes_cnt + no_cnt;

    double ans = class_attri_cnt[playgame]/total *

(attribute_yn_cnt[attributes[0]][Outlook][playgame])/(class_attri_cnt[playgame])

*(attribute_yn_cnt[attributes[1]][Temp][playgame])/(class_attri_cnt[playgame])*(attribute_yn_cnt[attributes[2]][Humidity][playgame])/(class_attri_cnt[playgame])*

(attribute_yn_cnt[attributes[3]][Wind][playgame])/(class_attri_cnt[playgame]);

    return ans;
```

```
};
```

```
int main(){  
    ifstream input("info-gain.csv");  
  
    string line, day, outlook, temp, humidity, wind, playGame;  
    int j = 0;  
    while(getline(input,line)){  
  
        stringstream str(line);  
  
        getline(str,day,',');  
        getline(str,outlook,',');  
        getline(str,temp,',');  
        getline(str,humidity,',');  
        getline(str,wind,',');  
        getline(str,playGame,'.');
```

```
        if(j==0){  
            j++;  
            attributes.push_back(outlook);  
            attributes.push_back(temp);  
            attributes.push_back(humidity);  
            attributes.push_back(wind);
```

```

        continue;
    }

    class_attri_cnt[playGame]++;

    attribute_yn_cnt[attributes[0]][outlook][playGame]++;
    attribute_yn_cnt[attributes[1]][temp][playGame]++;
    attribute_yn_cnt[attributes[2]][humidity][playGame]++;
    attribute_yn_cnt[attributes[3]][wind][playGame]++;

}

double yes_cnt = class_attri_cnt["Yes"];
double no_cnt = class_attri_cnt["No"];
double total = yes_cnt + no_cnt;

cout << "Enter the unknown case" << endl;
string Outlook, Temp, Humidity, Wind;
cin >> Outlook >> Temp >> Humidity >> Wind;

double p_yes = calculateProbability(Outlook, Temp, Humidity, Wind, "Yes");
double p_no = calculateProbability(Outlook, Temp, Humidity, Wind, "No");

//cout << p_yes << p_no << endl;

```

```

if(p_yes > p_no ) {

    cout << "The unknown case is classified as Yes" ;

}

else cout<< "The unknown case is classified as No";

}

```

**Result:**

**Input.csv:**

A	B	C	D	E	F
day	outlook	temp	humidity	wind	playGame
1	Sunny	Hot	High	Weak	No.
2	Sunny	Hot	High	Strong	No.
3	Overcast	Hot	High	Weak	Yes.
4	Rain	Mild	High	Weak	Yes.
5	Rain	Cool	Normal	Weak	Yes.
6	Rain	Cool	Normal	Strong	No.
7	Overcast	Cool	Normal	Strong	Yes.
8	Sunny	Mild	High	Weak	No.
9	Sunny	Cool	Normal	Weak	Yes.
10	Rain	Mild	Normal	Weak	Yes.
11	Sunny	Mild	Normal	Strong	Yes.
12	Overcast	Mild	High	Strong	Yes.
13	Overcast	Hot	Normal	Weak	Yes.
14	Rain	Mild	High	Strong	No.

### Output.csv:

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\USER\Desktop\dm lab\13th experiment> g++ bayescla.cpp -o b
PS C:\Users\USER\Desktop\dm lab\13th experiment> ./b
Enter the unknown case
sunny
hot
normal
weak
The unknown case is classified as No
PS C:\Users\USER\Desktop\dm lab\13th experiment> █
```