****

**A REPORT OF**

**“Data Mining Lab”**

**Code: 5IT451**

**Submitted by**

**Mr. Shatayu Nagdeve 2020BTEIT00026**

**Email: shatayu.nagdeve@walchandsangli.ac.in**

**Mobile: 8421337102**

**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. Shatayu Nagdeve (2020BTEIT00026)**

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 DM 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. Shatayu Nagdeve**

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. Shatayu Nagdeve

**Data Mining Lab Book**

Name:- Nagdeve Shatayu Bhagyoday

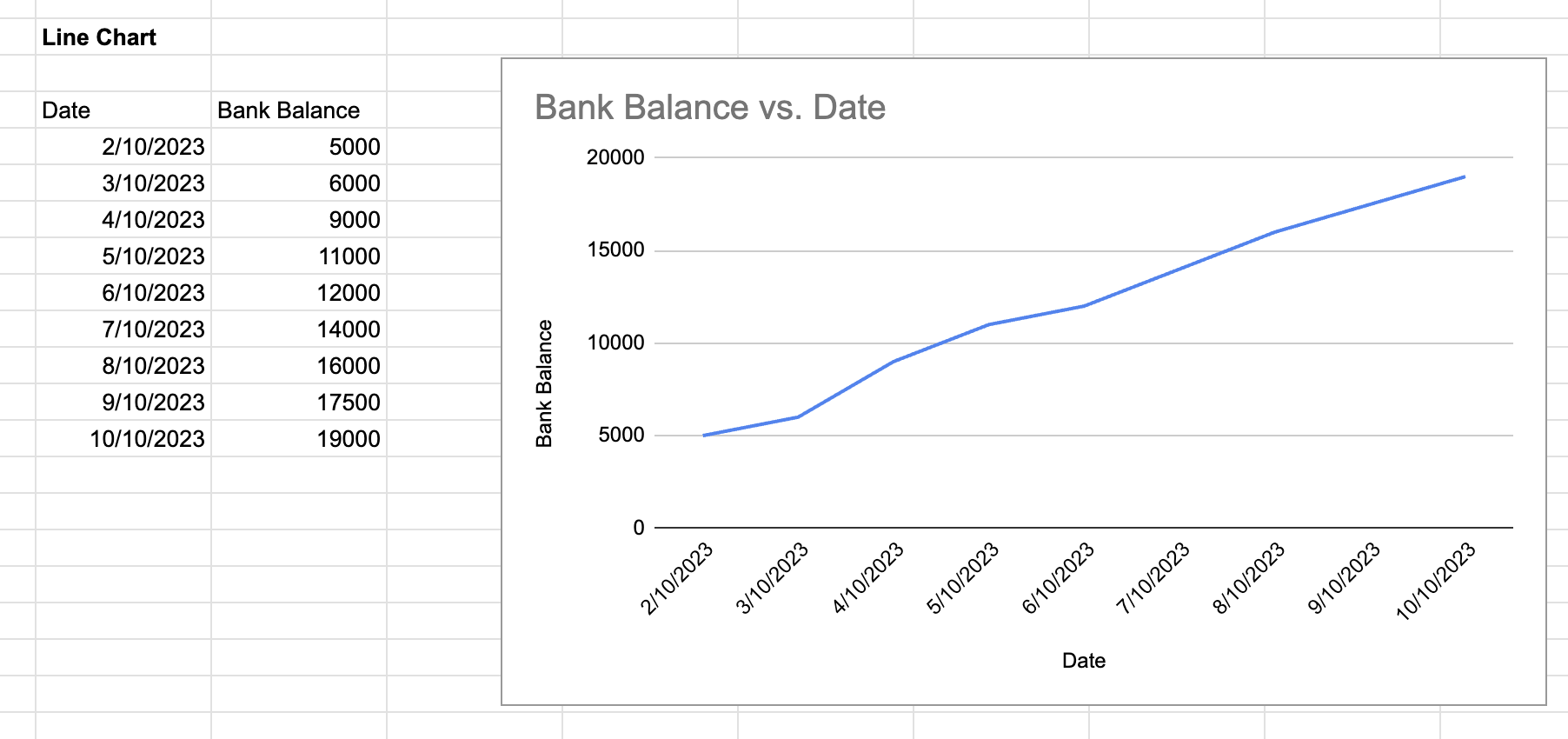
PRN:- 2020BTEIT00026

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 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. | DM concept of Complex Data Type. |  |

**Experiment No. 1**

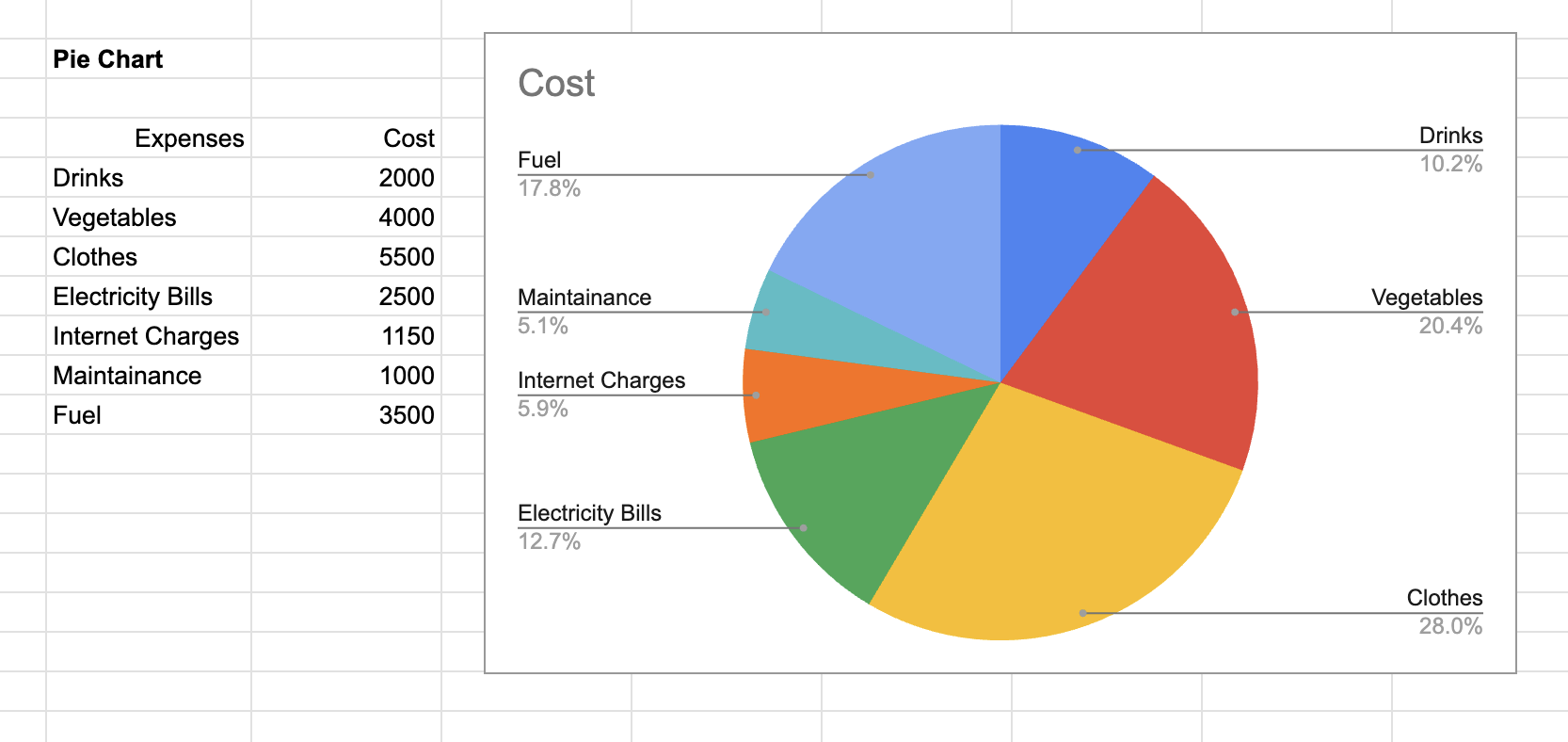
**Title :-** Study and use of different types of graphs and charts (use MS-XLS).

**Line chart- **

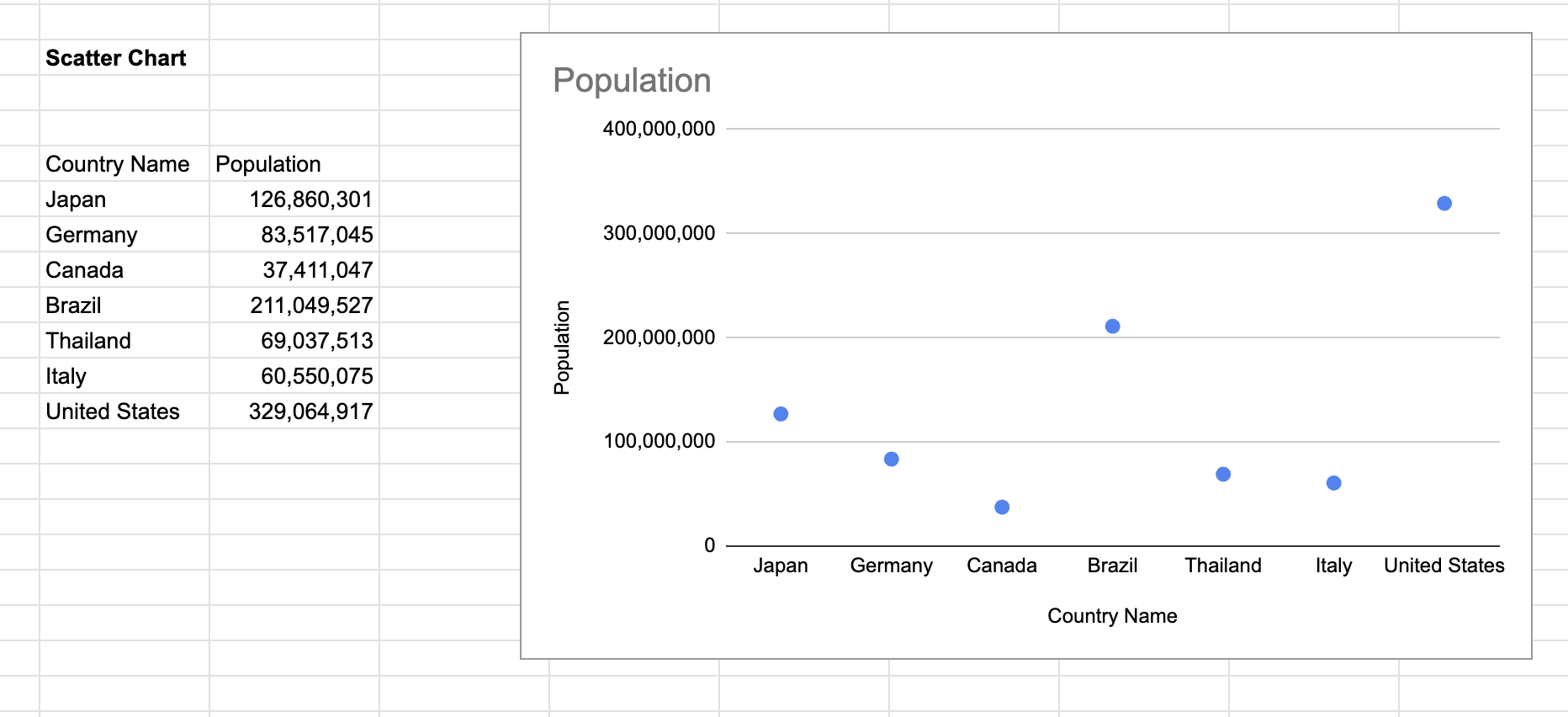
**Column chart-**



**Pie chart :-**



**Scatter chart:-**



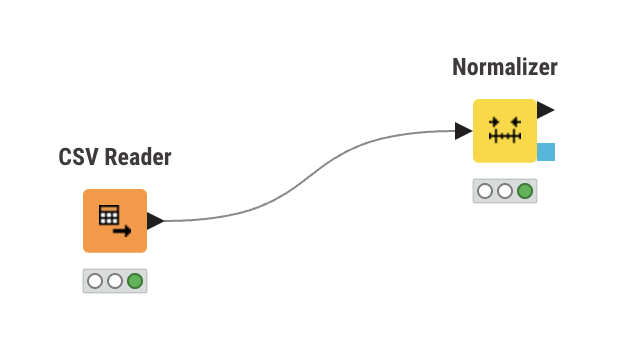
**Candlestick chart:-**

****

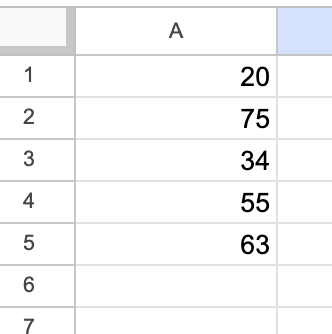
**Experiment No. 2**

**Title:-** To perform Normalization of data (Min-max and Z-score).

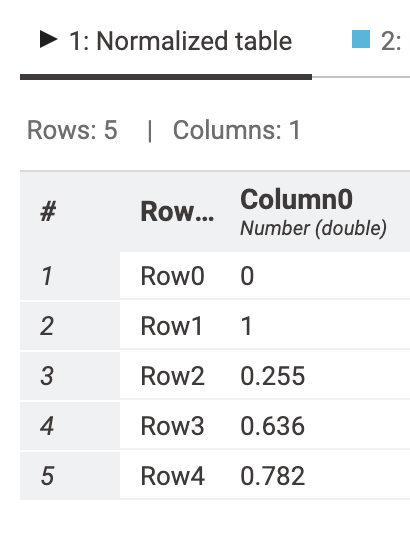
**Knime :**



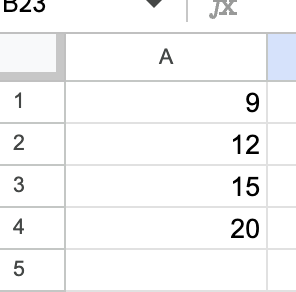
**Input :**

****

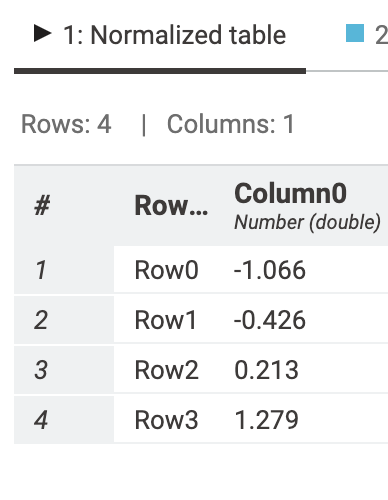
**Min - Max Normalization :**

****

**Input :**

****

**Z-Score Normalization**

****

**Code :**

import csv

import math

def min\_max\_normalization(data, new\_min, new\_max):

mini = min(data)

maxi = max(data)

normalized\_data = []

for value in data:

normalized\_value = ((value - mini) / (maxi - mini)) \* (new\_max - new\_min) + new\_min

normalized\_data.append(normalized\_value)

return normalized\_data

def z\_score\_normalization(data):

mean = sum(data) / len(data)

standard\_deviation = math.sqrt(sum((value - mean) \*\* 2 for value in data) / len(data))

normalized\_data = []

for value in data:

normalized\_value = (value - mean) / standard\_deviation

normalized\_data.append(normalized\_value)

return normalized\_data

def main():

data = []

with open('2/2input.csv', 'r') as file:

reader = csv.reader(file)

for row in reader:

data.append(float(row[0]))

opt = int(input("Enter option:\n1. Min-Max Normalization\n2. Z-Score Normalization\nOption: "))

if opt == 1:

new\_min = float(input("Enter new min: "))

new\_max = float(input("Enter new max: "))

normalized\_data = min\_max\_normalization(data, new\_min, new\_max)

with open('2/exp2\_output\_MinMax.csv', 'w') as file:

writer = csv.writer(file)

writer.writerow(['Data', 'Normalized Data'])

for i in range(len(data)):

writer.writerow([data[i], normalized\_data[i]])

elif opt == 2:

normalized\_data = z\_score\_normalization(data)

with open('2/exp2\_output\_Zscore.csv', 'w') as file:

writer = csv.writer(file)

writer.writerow(['Data', 'Normalized Data'])

for i in range(len(data)):

writer.writerow([data[i], normalized\_data[i]])

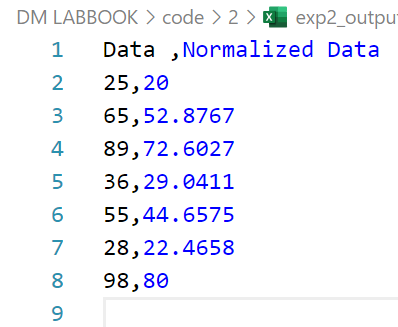
else:

print("Wrong Option")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output :**

****

**Experiment No. 3**

**Title:-** ToPerform Binning of Data.

**Code:**y

def equifreq(arr1, m):

a = len(arr1)

n = int(a / m)

for i in range(0, m):

arr = []

for j in range(i \* n, (i + 1) \* n):

if j >= a:

break

arr = arr + [arr1[j]]

print(arr)

#equal width

def equiwidth(arr1, m):

a = len(arr1)

w = int((max(arr1) - min(arr1)) / m)

min1 = min(arr1)

arr = []

for i in range(0, m + 1):

arr = arr + [min1 + w \* i]

arri=[]

for i in range(0, m):

temp = []

for j in arr1:

if j >= arr[i] and j <= arr[i+1]:

temp += [j]

arri += [temp]

print(arri)

#data to be binned

data = [5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215]

#no of bins

m = 3

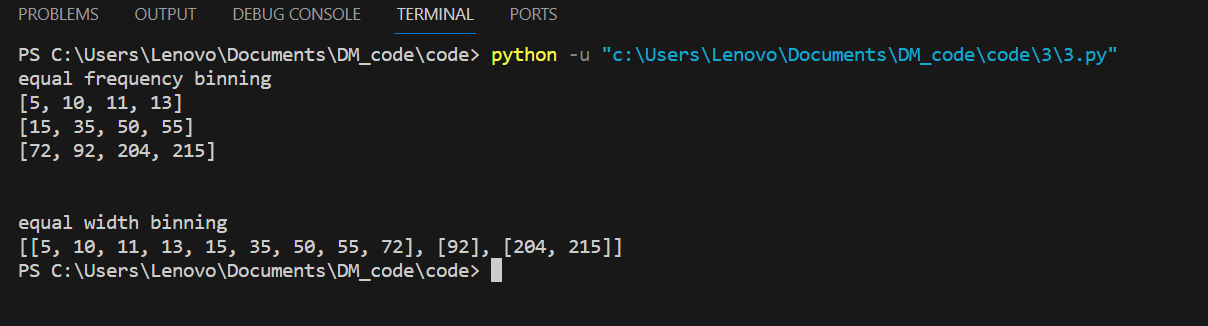
print("equal frequency binning")

equifreq(data, m)

print("\n\nequal width binning")

equiwidth(data, 3)

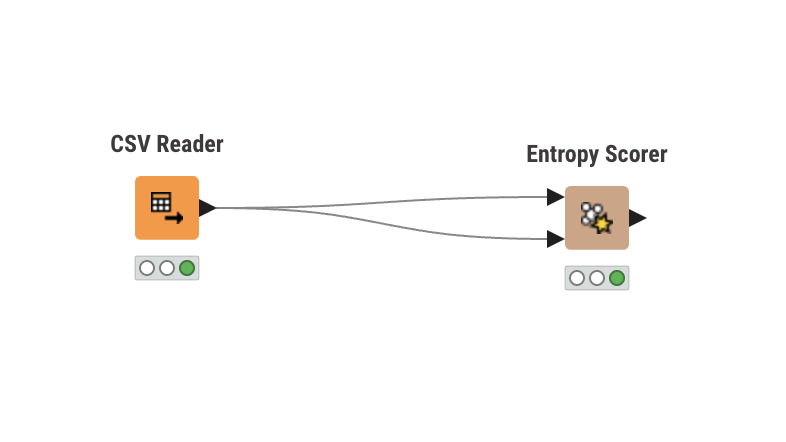
**Output :**

****

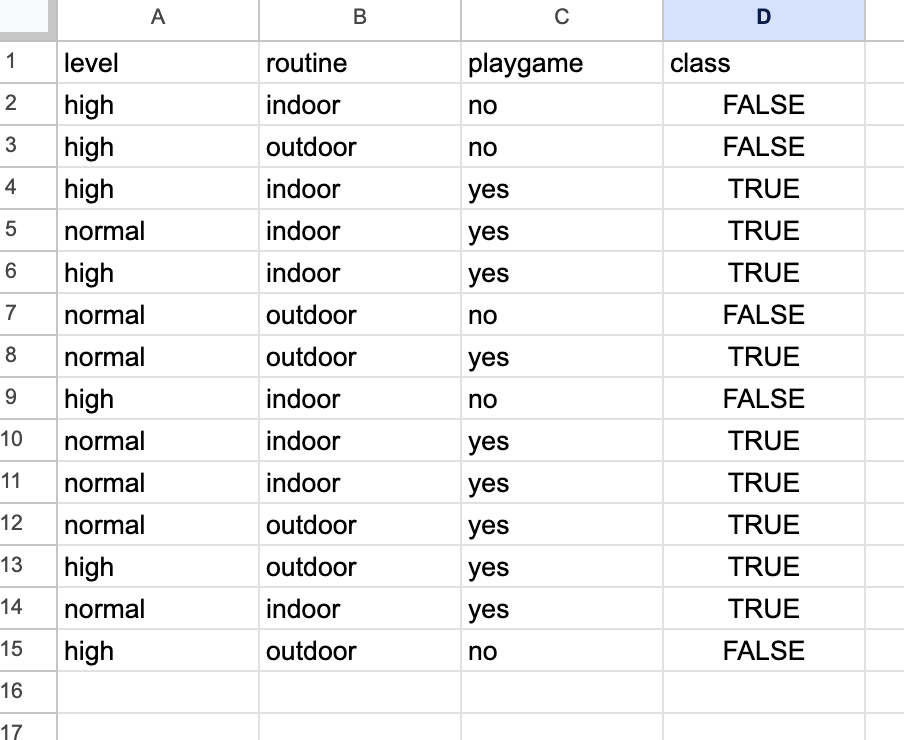
**Experiment No. 4**

**Title:-** Find Info Gain of an attribute from given data

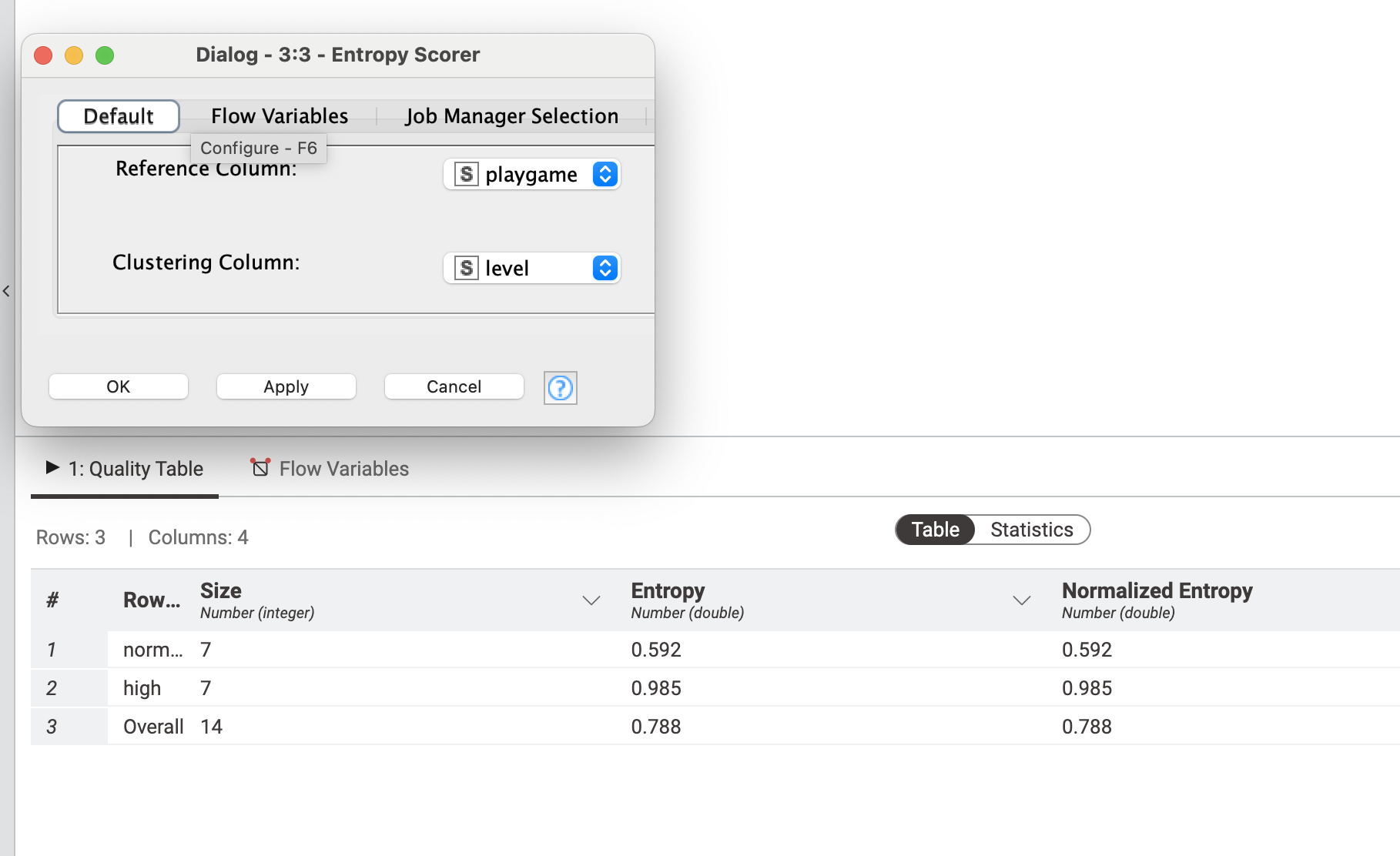
**Knime :**



**Input :**

****

**Output :**

****

**Code:**

import csv

import math

def main():

with open('4/4input.csv', 'r') as file:

reader = csv.reader(file)

next(reader) # Skip header row

choice = int(input("Enter Child Column Number : "))

parent = {}

child = {}

for row in reader:

day = row[0]

level = row[1]

routine = row[2]

playGame = row[3]

value = row[4]

if choice == 1:

childName = day

elif choice == 2:

childName = level

elif choice == 3:

childName = routine

elif choice == 4:

childName = value

else:

childName = routine

if playGame in parent:

parent[playGame] += 1

else:

parent[playGame] = 1

if childName in child:

if playGame in child[childName]:

child[childName][playGame] += 1

else:

child[childName][playGame] = 1

else:

child[childName] = {playGame: 1}

pos = parent.get("Yes", 0)

neg = parent.get("No", 0)

total = pos + neg

parent\_entropy = -((pos / total) \* math.log2(pos / total) +

(neg / total) \* math.log2(neg / total))

print("Parent Entropy:", parent\_entropy)

child\_entropy = 0

for childName, games in child.items():

pR = games.get("Yes", 0)

nR = games.get("No", 0)

tR = pR + nR

child\_entropy += -((pR + nR) / total) \* ((pR / tR) \* math.log2(pR / tR) +

(nR / tR) \* math.log2(nR / tR))

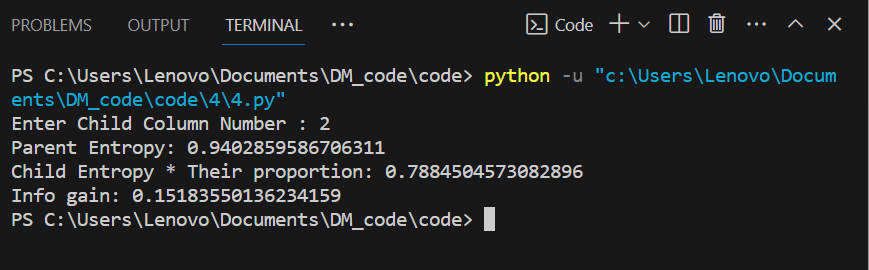
print("Child Entropy \* Their proportion:", child\_entropy)

print("Info gain:", parent\_entropy - child\_entropy)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

****

**Experiment No. 5**

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

Code:

import csv

def main():

with open("5/5input.csv", "r") as file:

csv\_reader = csv.reader(file)

next(csv\_reader) # Skip header row

classrowcolMap = {}

colMap = {}

rowMap = {}

for row in csv\_reader:

rowName = row[0]

colName = row[1]

count = int(row[2])

classrowcolMap.setdefault(rowName, {})

classrowcolMap[rowName][colName] = count

colMap.setdefault(colName, 0)

colMap[colName] += count

rowMap.setdefault(rowName, 0)

rowMap[rowName] += count

for row in rowMap:

for col in colMap:

print(row + "-" + col + ":", classrowcolMap[row][col])

for row in rowMap:

print(row + "->" + str(rowMap[row]))

for col in colMap:

print(col + "->" + str(colMap[col]))

colSum = sum(colMap.values())

print("colSum:", colSum)

rowSum = sum(rowMap.values())

print("rowSum:", rowSum)

with open("5/exp5\_output.csv", "w", newline="") as fw:

csv\_writer = csv.writer(fw)

csv\_writer.writerow(["Column\\row", "", "Bollywood", "", "Tollywood", "", "Total", "", ""])

csv\_writer.writerow(["", "Count", "t - weight", "d - weight", "Count", "t - weight", "d - weight", "Count", "t - weight", "d - weight"])

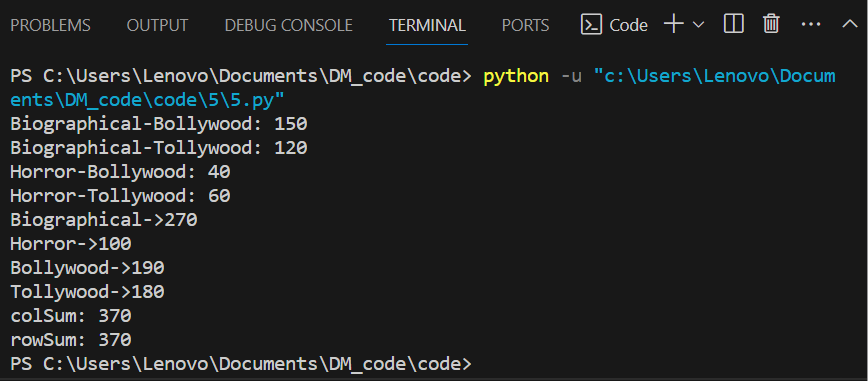
for row in rowMap:

csv\_writer.writerow([row] + [classrowcolMap[row][col] for col in colMap] + [rowMap[row], rowMap[row] / rowMap[row] \* 100, rowMap[row] / colSum \* 100])

csv\_writer.writerow(["Total"] + [colMap[col] for col in colMap] + [colSum, 100, 100])

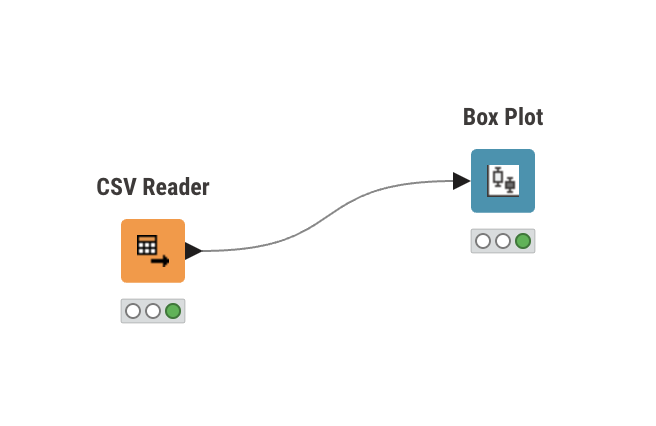
if \_\_name\_\_ == "\_\_main\_\_":

main()

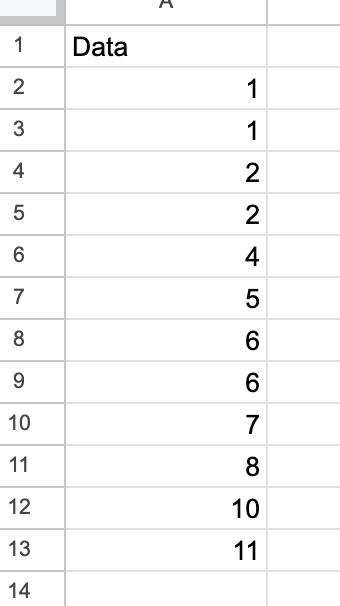
**Experiment No. 6**

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

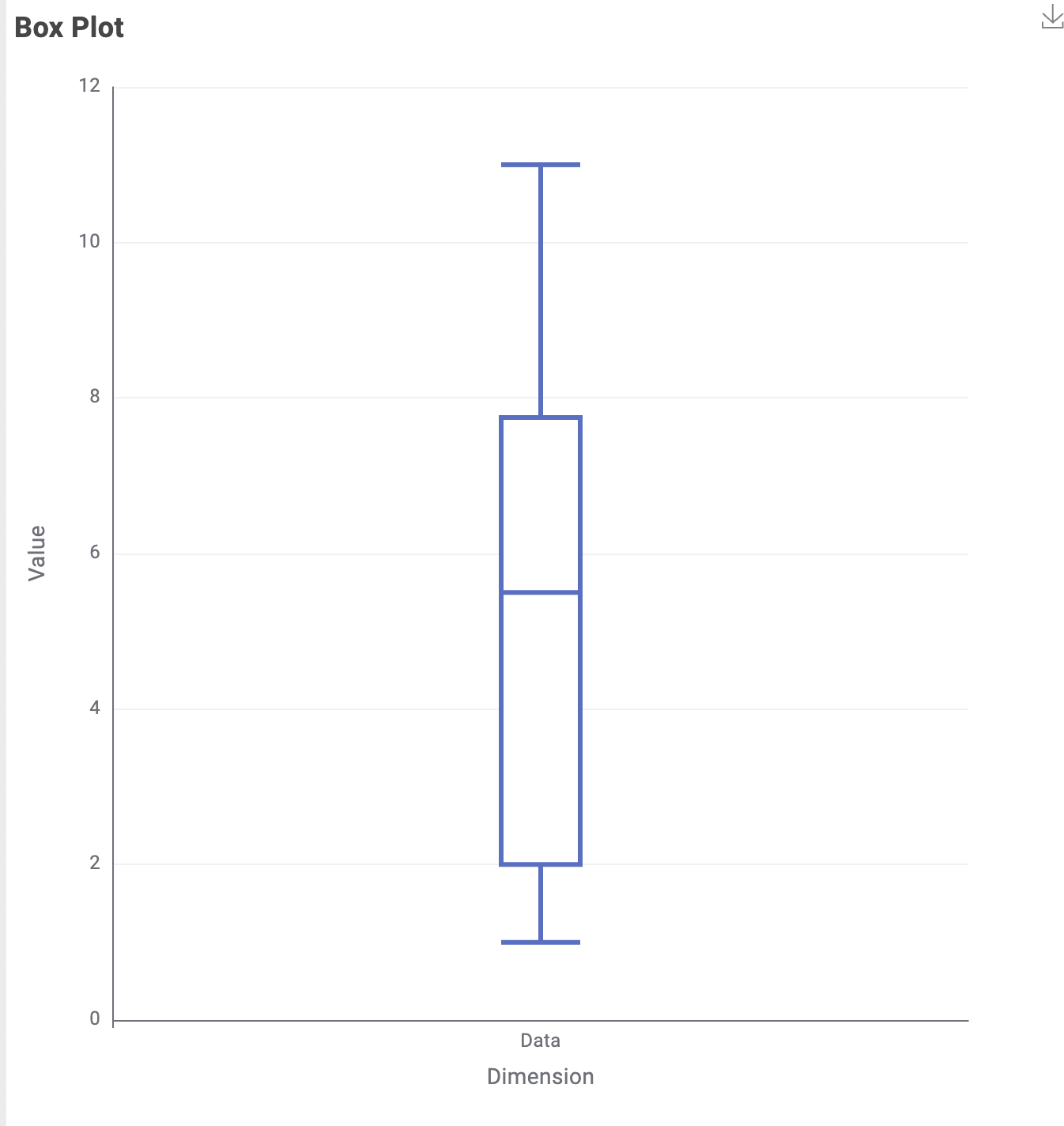
**Knime :**

****

**Input :**

****

**Output :**

****

**Code :**

import csv

def median(a):

size = len(a)

if size % 2 == 1:

return a[size//2]

else:

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

def quartile1(v):

n = len(v)

first = []

for i in range(n // 2):

first.append(v[i])

return median(first)

def quartile3(v):

n = len(v)

last = []

if n % 2 == 0:

for i in range(n // 2, n):

last.append(v[i])

else:

for i in range(n // 2 + 1, n):

last.append(v[i])

return median(last)

arr = []

with open('6/6input.csv', 'r') as in\_file:

csv\_reader = csv.reader(in\_file)

next(csv\_reader) # Skip header

for line in csv\_reader:

x = int(line[0])

arr.append(x)

n = len(arr)

arr.sort()

with open('6/exp6\_output.csv', 'w') as out\_file:

out\_file.write("Minimum value: ," + str(arr[0]) + "\n")

out\_file.write("Quartile1 value: ," + str(quartile1(arr)) + "\n")

out\_file.write("Median value: ," + str(median(arr)) + "\n")

out\_file.write("Quartile3 value: ," + str(quartile3(arr)) + "\n")

out\_file.write("Maximum value: ," + str(arr[n-1]) + "\n")

print("Minimum value is", arr[0])

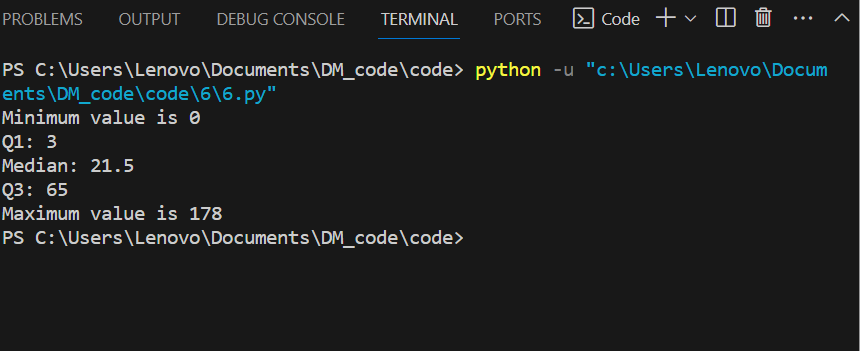
print("Q1:", quartile1(arr))

print("Median:", median(arr))

print("Q3:", quartile3(arr))

print("Maximum value is", arr[n - 1])

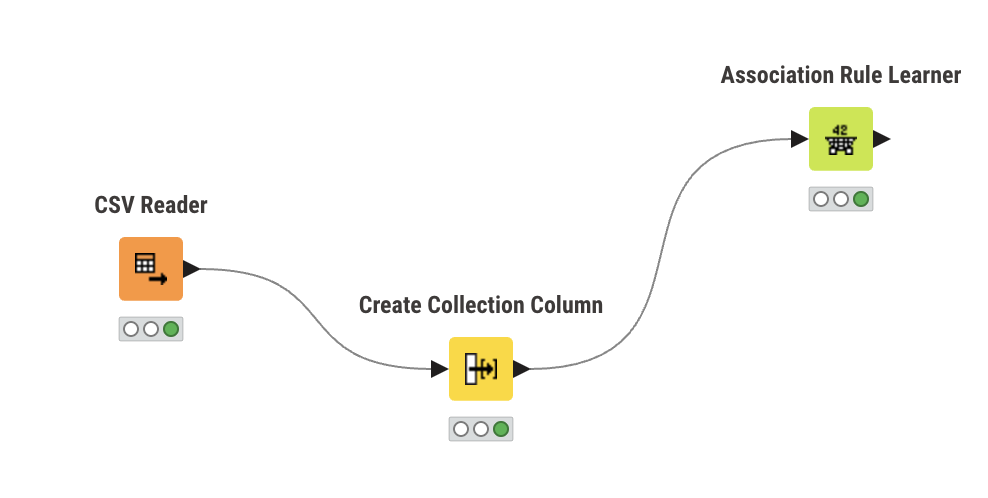
**Output :**

****

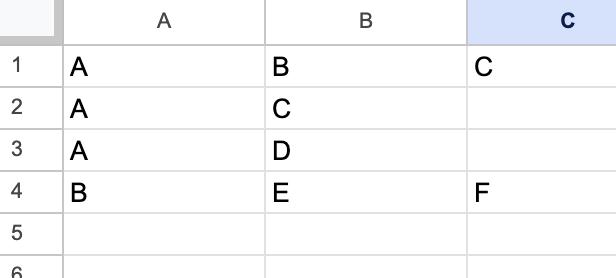
**Experiment No. 7**

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

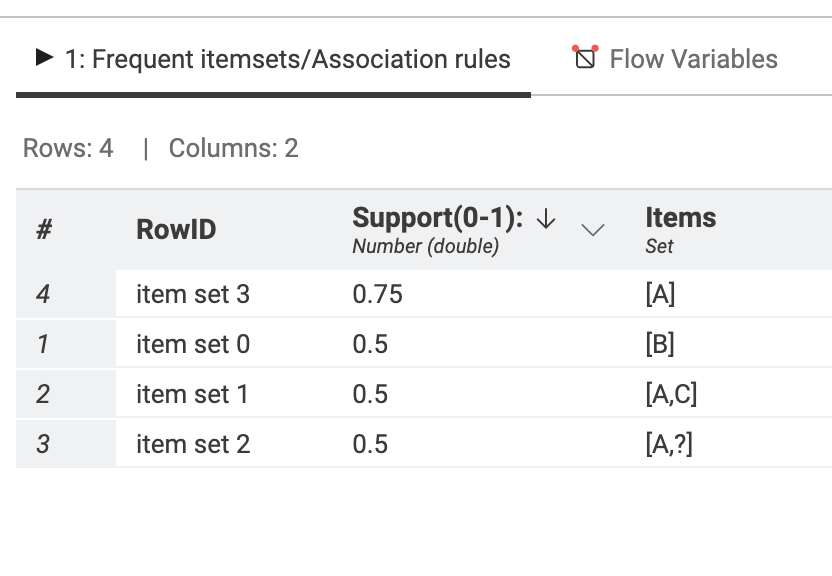
**Knime :**

****

**Input :**

****

**Output :**

****

**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;

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;

}

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;

}

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;

}

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);

// cout << "\nVector 1 :";

// for(auto s : v1){

// cout << s << " ";

// }

// cout << "\n";

// cout << "\nVector 2 :";

// for(auto s : v2){

// cout << s << " ";

// }

// cout << "\n";

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("7input.csv", ios::in);

if (!fin.is\_open())

{

perror("Error in opening file : ");

}

cout << "Frequency % :";

cin >> minfre;

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

// cout << datatable.size() << endl;

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

minfre = minfre \* 4 / 100;

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());

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;

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();

}

// cout << "Flag : " << flag << "\n";

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;

prev = cloneit(cur);

i++;

}

ofstream fw("ferqitem\_op.csv", ios::out);

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

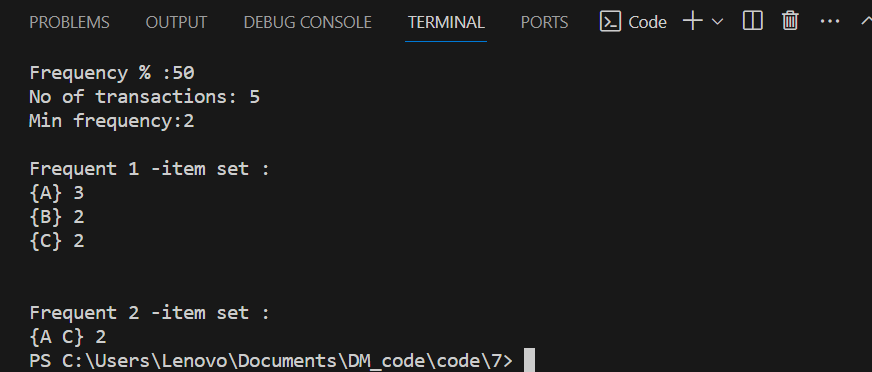
{

fw << "{" << \*it << "}" << endl;

}

return 1;

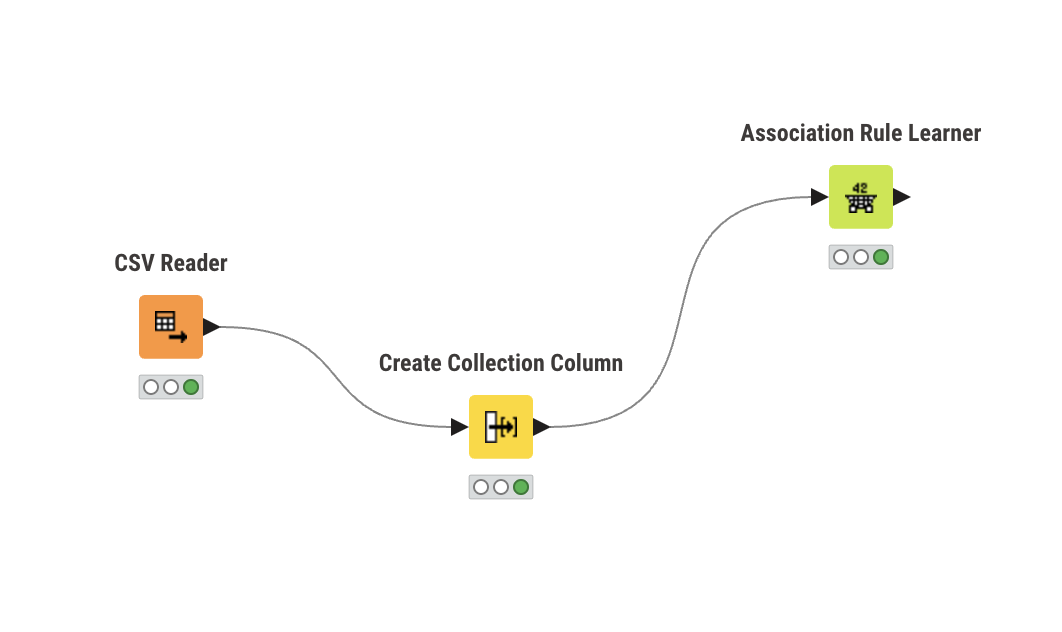
}

****

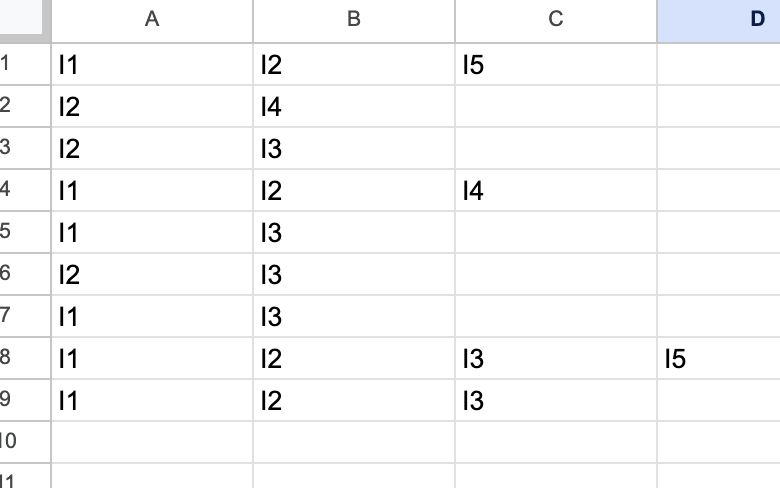
**Experiment No 8**

**Title:-** Extend program 6, to find association rules.

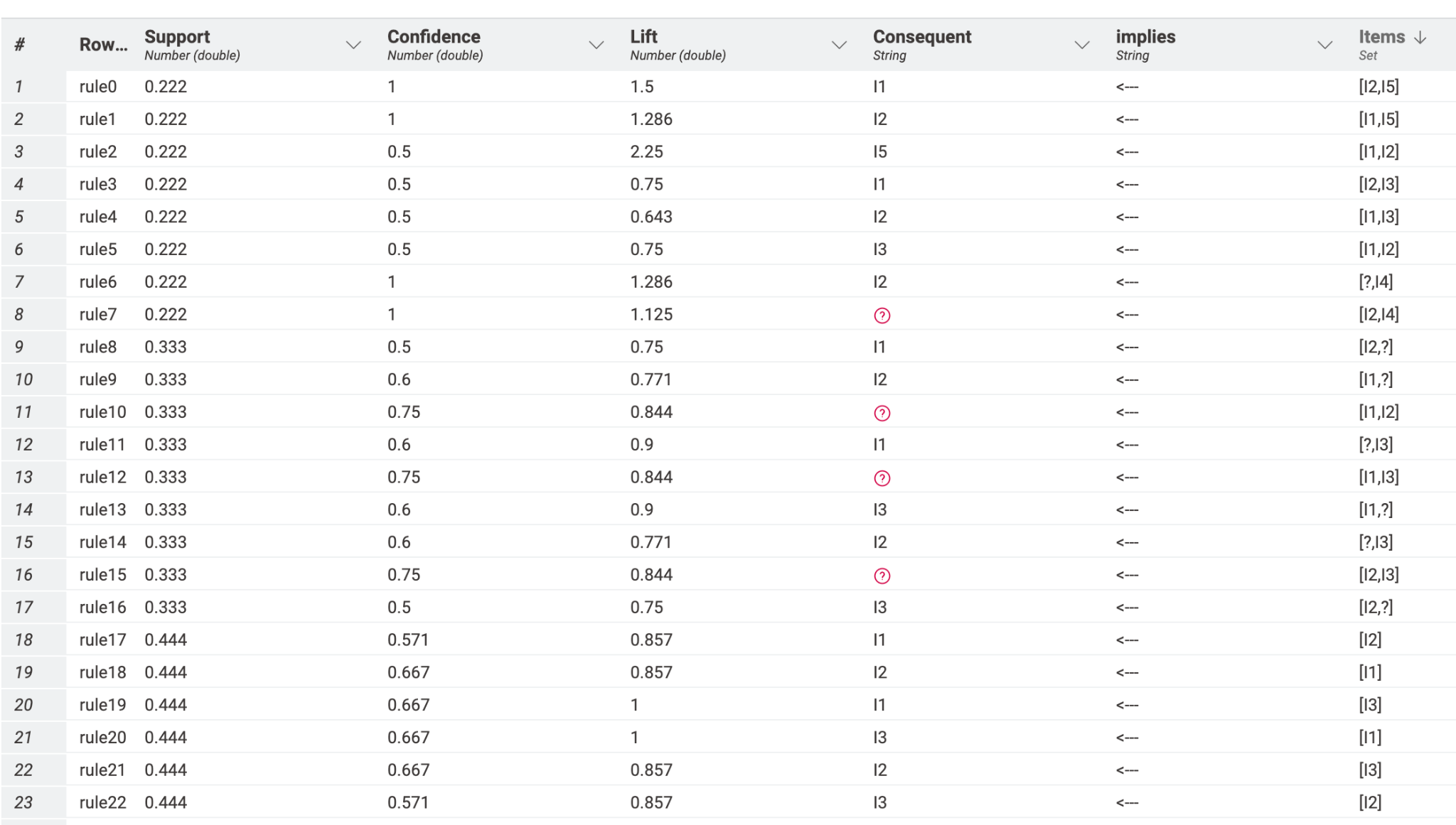
**Knime :**

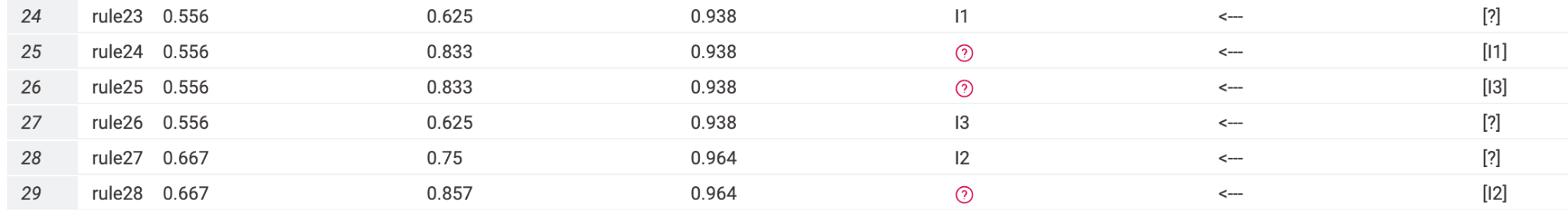


**Input :**



**Output :**

****

****

**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;

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;

}

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;

}

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;

}

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 countOccurences(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("exp8\_output.csv", ios::out);

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 = countOccurences(items); // Total support

int count2 = countOccurences(v1);

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

if (conf >= confidence)

{

fw1 << "{ ";

for (auto s : v1)

{

fw1 << s << " ";

}

fw1 << "} ,"

<< "-> "

<< ", {";

for (auto s : v2)

{

fw1 << s << " ";

}

fw1 << "} , " << 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();

}

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("8input.csv", ios::in);

if (!fin.is\_open())

{

perror("Error in opening file : ");

}

cout << "Enter Support % :";

cin >> minfre;

cout << "Enter 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<<datatable.size()<<endl;

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

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

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());

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;

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();

}

// cout << "Flag : " << flag << "\n";

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;

prev = cloneit(cur);

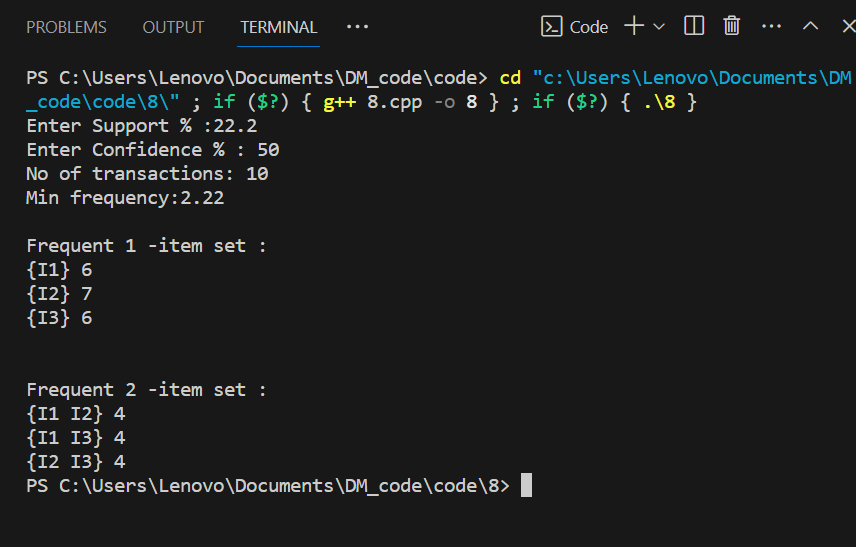
i++;

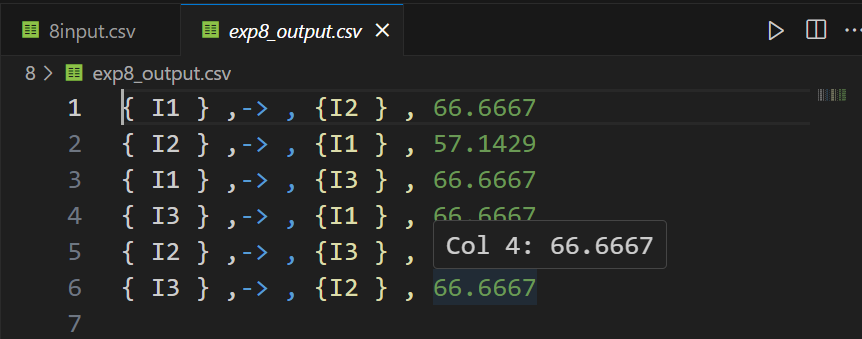
}

generateAssociationRules(prev);

return 1;

}

****

****

**Experiment No 9**

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

**Code :**

import csv

from collections import defaultdict

def read\_csv(file\_path):

transactions = defaultdict(list)

with open(file\_path, 'r', newline='') as csvfile:

reader = csv.reader(csvfile)

for row in reader:

items = [item.strip() for item in row if item.strip()]

if items:

transactions[len(transactions) + 1] = items

return list(transactions.values())

def calculate\_support(itemset, transactions):

count = 0

for transaction in transactions:

if set(itemset).issubset(set(transaction)):

count += 1

return count / len(transactions)

def apriori(transactions, min\_support):

frequent\_itemsets = []

unique\_items = set(item for transaction in transactions for item in transaction)

# Generate 1-itemsets

candidates = [[item] for item in unique\_items]

k = 1

while candidates:

next\_candidates = []

for candidate in candidates:

support = calculate\_support(candidate, transactions)

if support >= min\_support:

frequent\_itemsets.append((candidate, support))

next\_candidates.extend([candidate + [item] for item in unique\_items if item not in candidate])

candidates = next\_candidates

k += 1

return frequent\_itemsets

def generate\_association\_rules(frequent\_itemsets, min\_confidence):

association\_rules = []

for itemset, support in frequent\_itemsets:

if len(itemset) > 1:

for i in range(1, len(itemset)):

antecedent = itemset[:i]

consequent = itemset[i:]

confidence = support / calculate\_support(antecedent, transactions)

if confidence >= min\_confidence:

association\_rules.append((antecedent, consequent, confidence))

return association\_rules

# Read the CSV file containing transaction data

input\_file\_path = '9/9input.csv'

transactions = read\_csv(input\_file\_path)

# Get user input for the minimum support and confidence thresholds

min\_support = float(input("Enter the minimum support threshold (a value between 0 and 1): "))

min\_confidence = float(input("Enter the minimum confidence threshold (a value between 0 and 1): "))

# Find frequent itemsets using Apriori

frequent\_itemsets = apriori(transactions, min\_support)

# Print the frequent itemsets

print("\nFrequent Itemsets:")

for itemset, support in frequent\_itemsets:

print(f"{set(itemset)} - Support: {support}")

# Find association rules

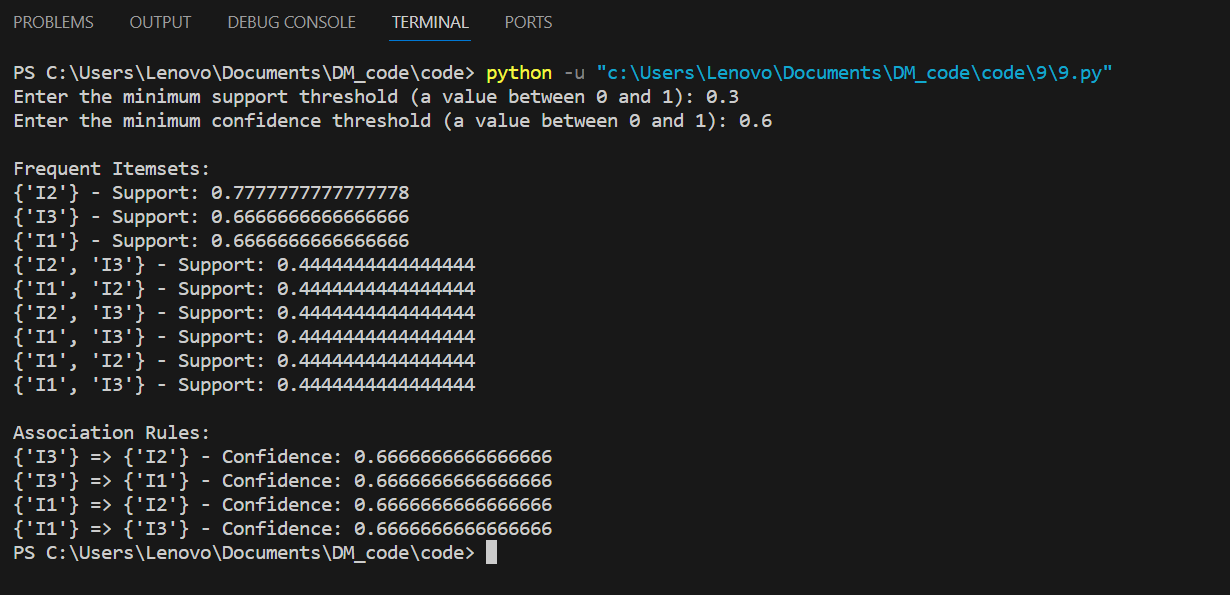
association\_rules = generate\_association\_rules(frequent\_itemsets, min\_confidence)

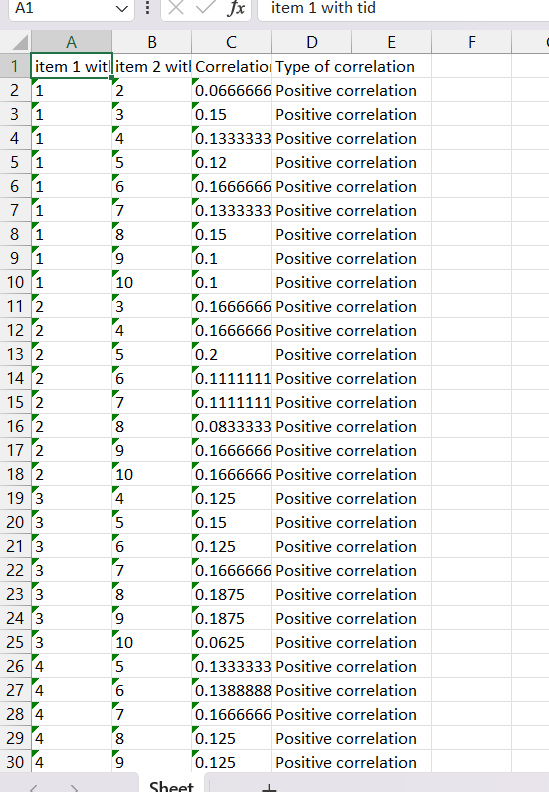
# Print the association rules

print("\nAssociation Rules:")

for antecedent, consequent, confidence in association\_rules:

print(f"{set(antecedent)} => {set(consequent)} - Confidence: {confidence}")

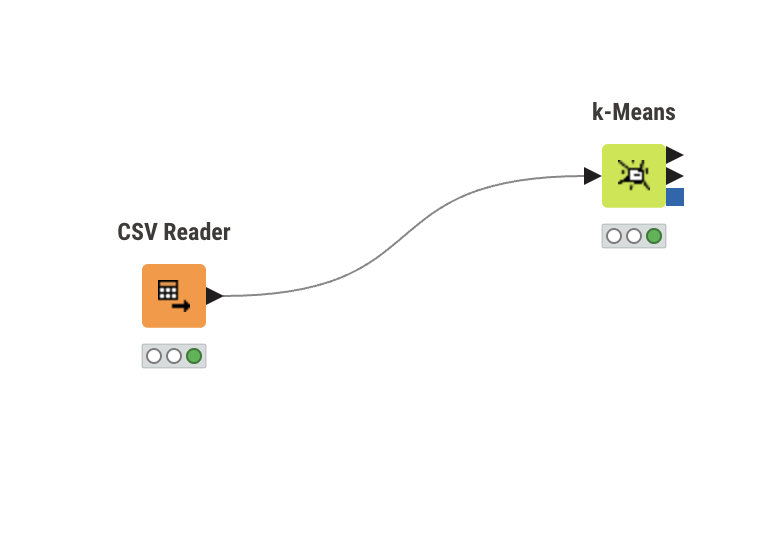




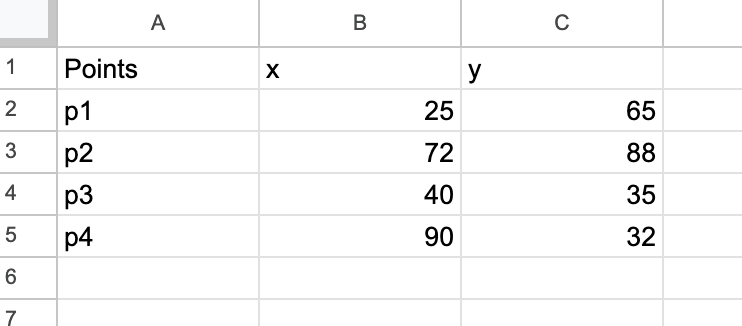
**Experiment No 10**

**Title:-** Distance and cluster

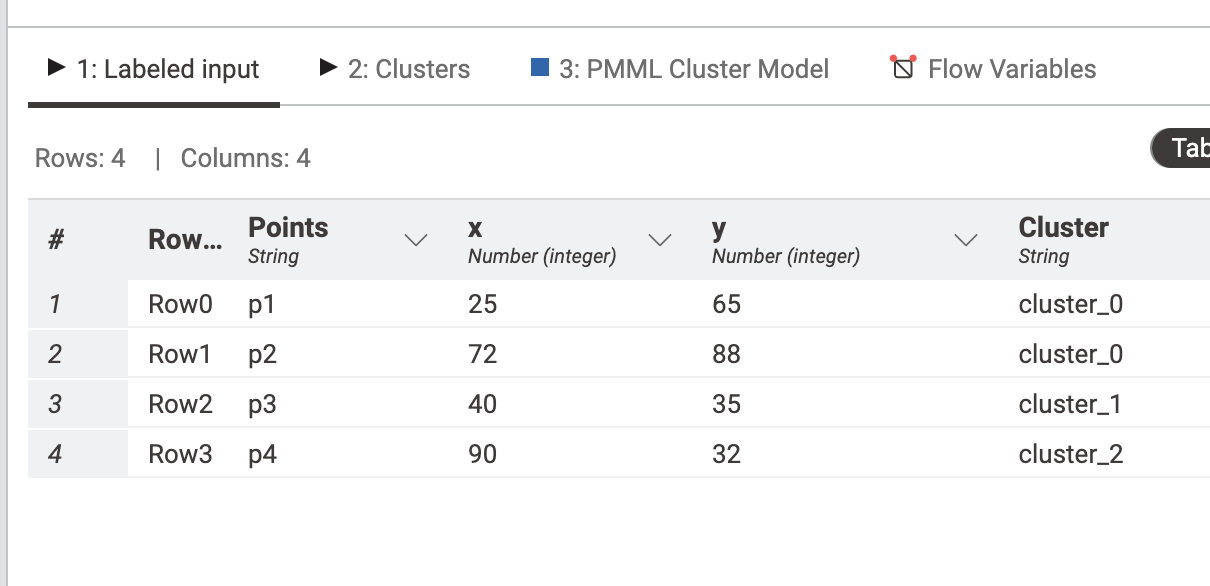
**Knime :**

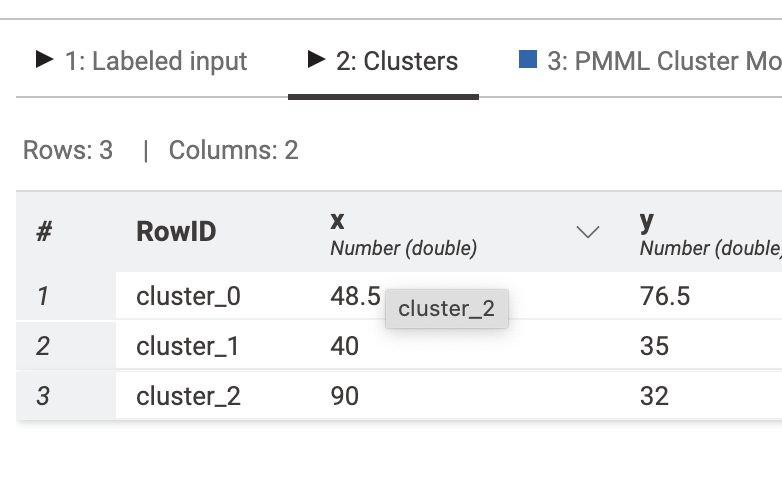
****

**Input :**



**Output :**

****

****

**Code :**

import csv

import math

def distance(x1, y1, x2, y2):

return math.sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2)

def main():

lines = []

with open("10/10input.csv", "r") as infile:

lines = infile.readlines()

points = []

for line in lines[1:]:

x, y = map(int, line.strip().split(",")[1:])

points.append((x, y))

n = len(points)

# Calculate mid point

x\_sum = sum(x for x, y in points)

y\_sum = sum(y for x, y in points)

mid\_x = x\_sum / n

mid\_y = y\_sum / n

print(f"Mid Point: ({mid\_x}, {mid\_y})")

# Write distance matrix to cluster\_output.csv

with open("10/cluster\_output.csv", "w", newline='') as outfile:

writer = csv.writer(outfile)

writer.writerow(["", "p1", "p2", "p3", "p4", "C"])

for i in range(n):

writer.writerow([f"p{i+1}"] + [distance(points[i][0], points[i][1], points[j][0], points[j][1]) for j in range(i + 1)] + [0])

# Calculate nearest point to the center

nearest\_dist = float('inf')

nearest\_point = 0

for i in range(n):

dist = distance(mid\_x, mid\_y, points[i][0], points[i][1])

print(f"Distance of p{i + 1} from centre: {dist}")

writer.writerow([dist] + [0] \* i + [dist] + [0] \* (n - i - 1))

if dist < nearest\_dist:

nearest\_dist = dist

nearest\_point = i + 1

print(f"Nearer Distance: {nearest\_dist}")

print(f"\nNearest point from Centre is: p{nearest\_point}")

# Calculate distance from new center to points

writer.writerow([""])

writer.writerow(["", "p1", "p2", "p3", "p4"])

for i in range(n):

writer.writerow([f"p{i+1}"] + [distance(points[i][0], points[i][1], points[j][0], points[j][1]) for j in range(i + 1)] + [0])

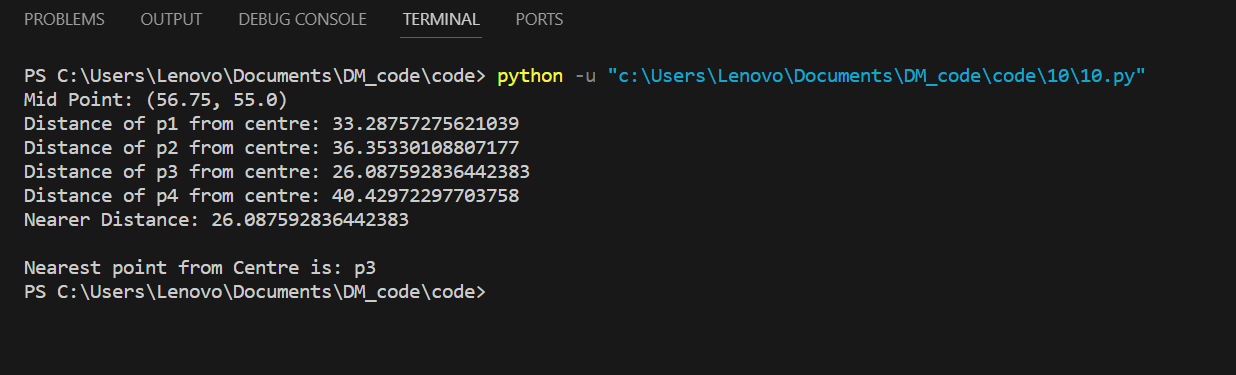
x\_new, y\_new = points[nearest\_point - 1]

writer.writerow([f"p{nearest\_point} (New Center)"] + [distance(x\_new, y\_new, points[j][0], points[j][1]) for j in range(n)])

if \_\_name\_\_ == "\_\_main\_\_":

main()

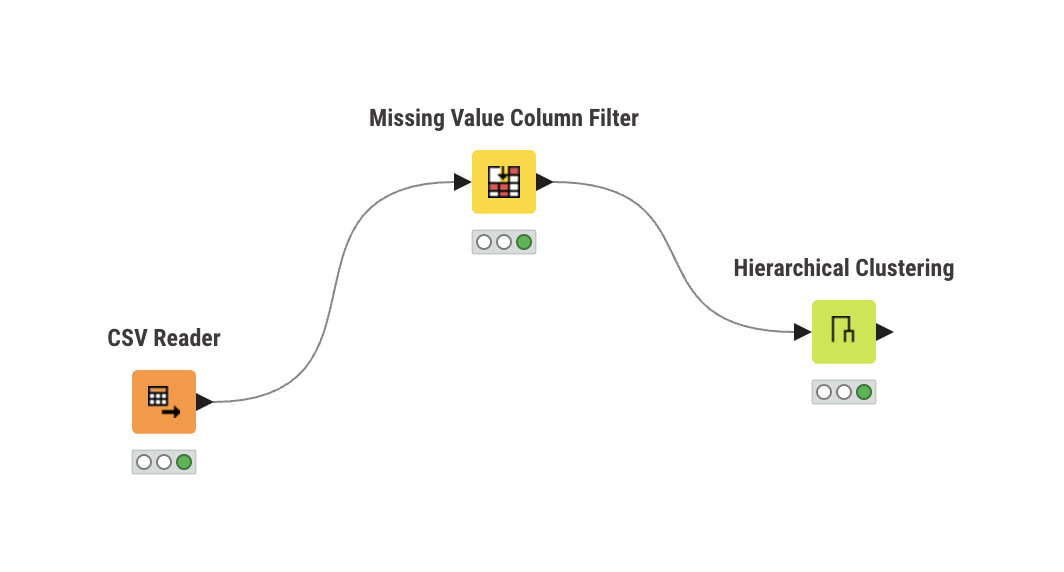
**Output:**

****

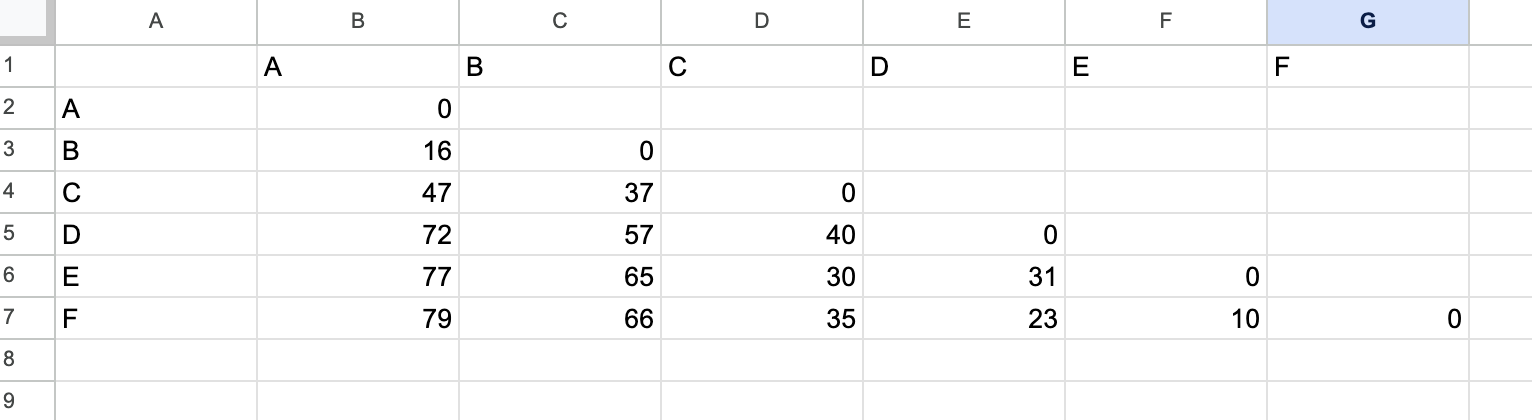
**Experiment No 11**

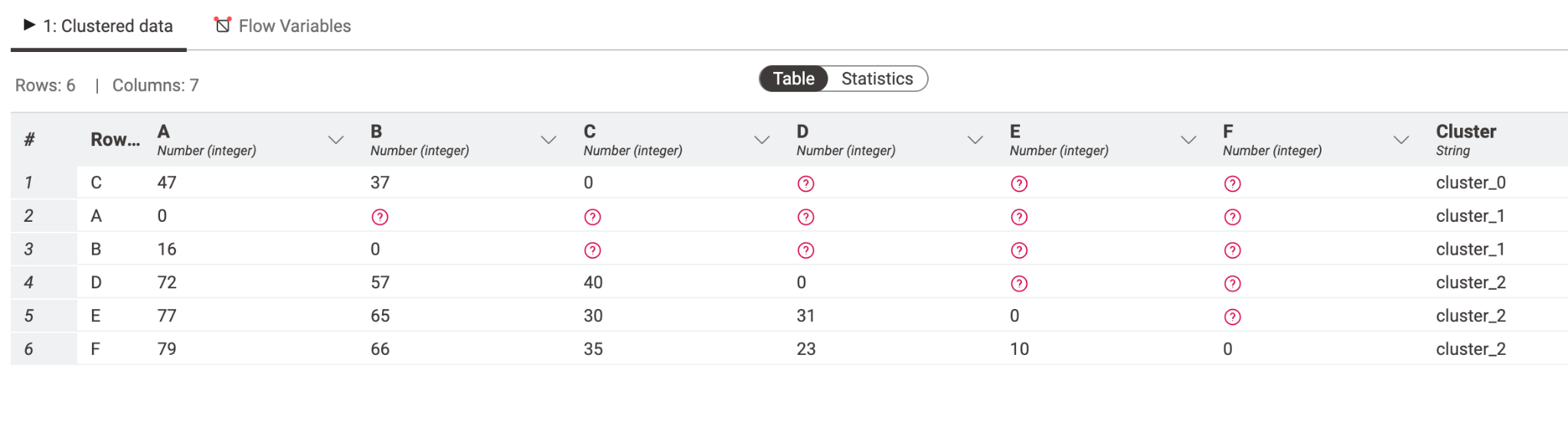
**Title :-** Agglomerative Hierarchical clustering using single linkage method

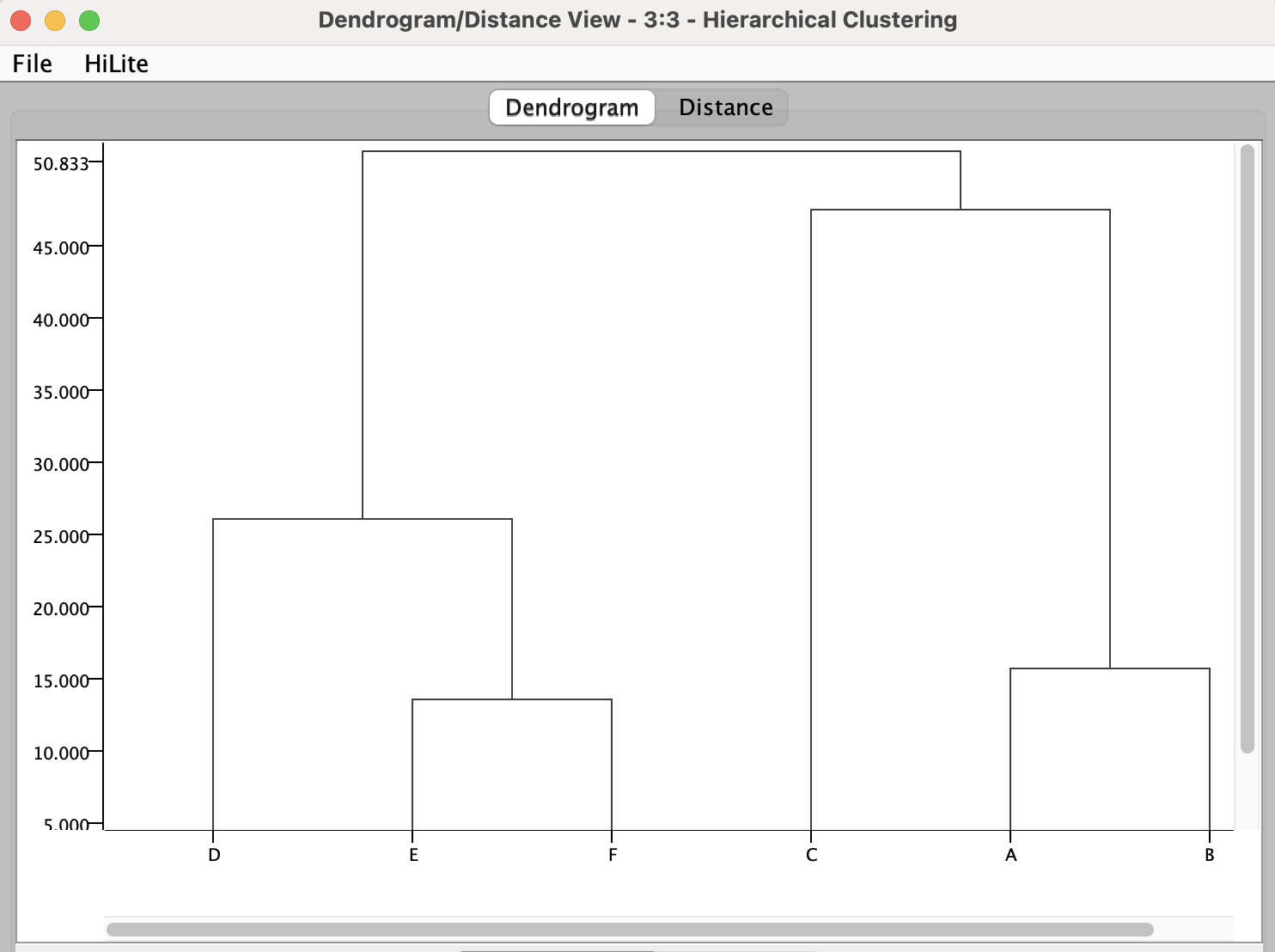
**Knime :**



**Input :**

****

**Output : **

****

**Code :**

#include <bits/stdc++.h>

using namespace std;

int op = 1;

ofstream fwtr("exp11\_output.csv", ios::out);

string algomerative(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;

while (getline(st, point, ','))

{

if (i == 0)

{

i++;

continue;

}

points.push\_back(point);

}

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;

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 Choosen : " << pt1 << " " << pt2 << endl;

string up, down;

if (pt1[0] > pt2[0])

{

up = pt2;

down = pt1;

}

else

{

up = pt1;

down = pt2;

}

string newPt = down + up;

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);

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 = "11input.csv";

fstream file1(input, ios::in);

string line;

getline(file1, line);

int pt = 0;

stringstream st(line);

int j = 0, len = 0;

string point;

while (getline(st, point, ','))

{

if (j == 0)

{

j++;

continue;

}

len++;

}

for (int i = 1; i <= len - 2; i++)

{

string output = algomerative(input);

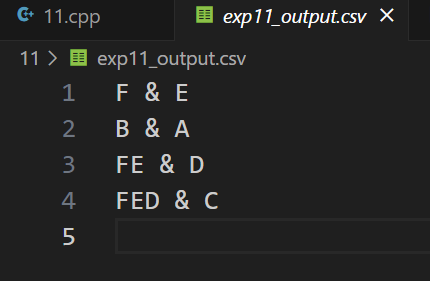
input = output;

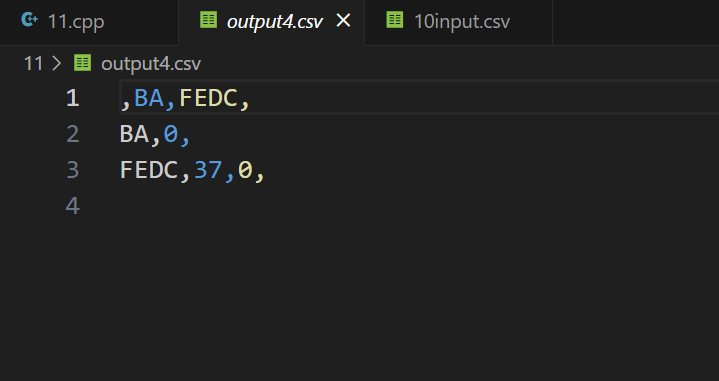
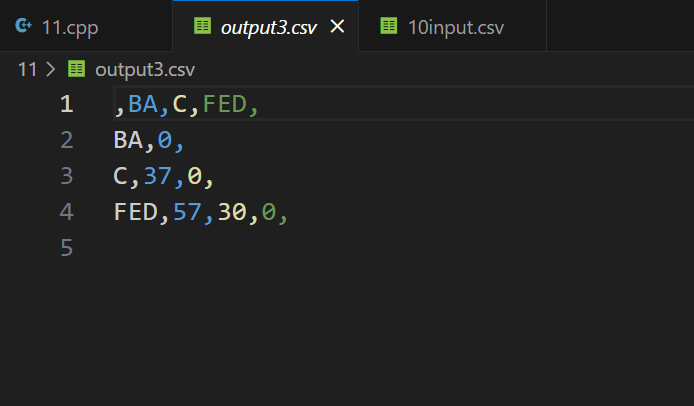
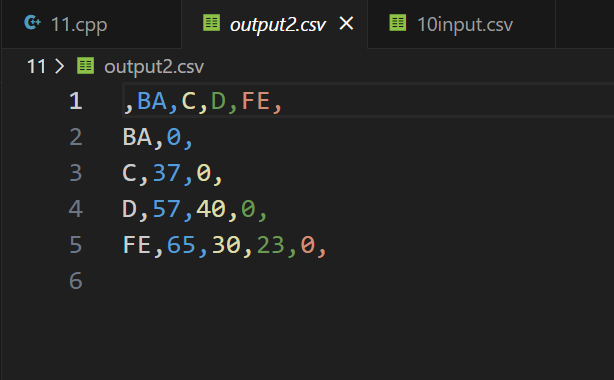
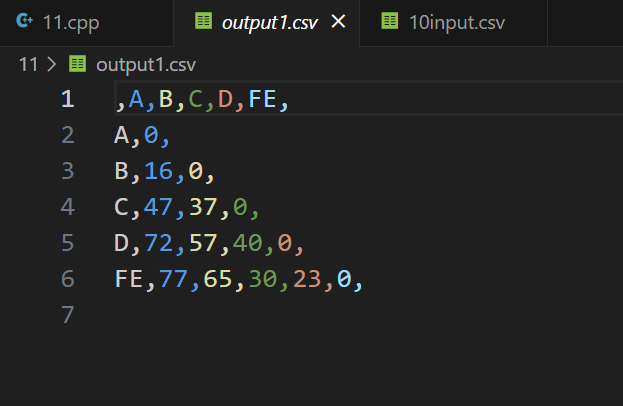
}

return 0;

}

**Output:**

****

****

**Experiment No 12**

**Title:-**

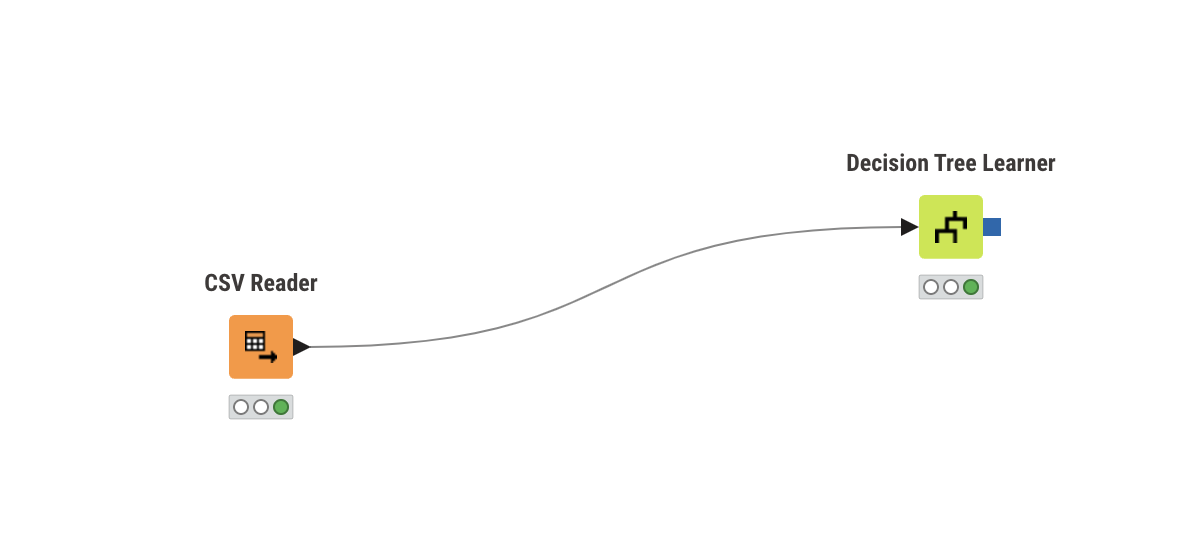
Attribute for classification ,Write a program to find

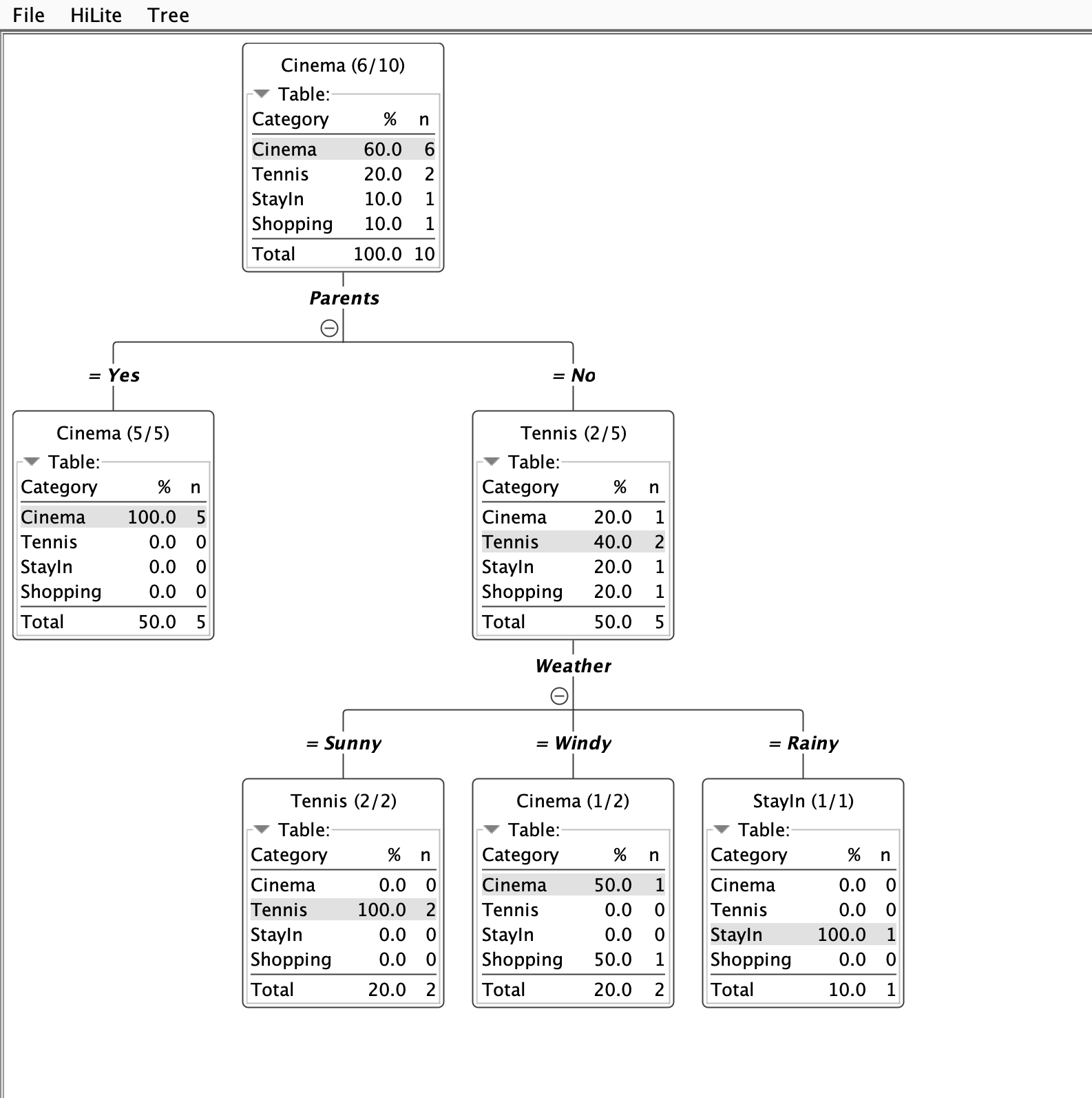
A. Gain

B. Gini index

For categorical and numerical values

**Knime :**

****

****

**Code :**

**Gain.cpp**

#include <bits/stdc++.h>

using namespace std;

vector<string> sub\_classes;

map<string, int> mainClass;

map<string, unordered\_set<string>> dist\_val;

map<string, int> dist\_val\_count;

map<string, map<string, int>> val\_count;

double maxGain = DBL\_MIN;

string root = "null";

ofstream fw("exp12\_op\_gain.csv", ios::out);

void calculateGain(string subClass, double mainC\_gain)

{

double totR = mainClass["Yes"] + mainClass["No"];

double ent = 0;

for (auto dv : dist\_val[subClass])

{

double tR = dist\_val\_count[dv];

double pR = val\_count[dv]["Yes"], nR = val\_count[dv]["No"];

if (pR != 0)

ent += -(tR / totR) \* ((pR / tR) \* log2(pR / tR));

if (nR != 0)

ent += -(tR / totR) \* ((nR / tR) \* log2(nR / tR));

}

cout << "InfoGain ( " << subClass << "|"

<< "playGame ) : " << ent << "\n";

fw << "InfoGain ( " << subClass << "|"

<< "playGame )," << ent << "\n";

double gain = mainC\_gain - ent;

cout << "Gain ( " << subClass << "|"

<< "playGame ) : " << gain << "\n\n";

fw << "Gain ( " << subClass << "|"

<< "playGame )," << gain << "\n";

if (gain > maxGain)

{

maxGain = gain;

root = subClass;

}

}

int main()

{

fstream file("12gain.csv", ios::in);

string line, word;

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

if (!file.is\_open())

{

perror("Error in opening input file : ");

return -1;

}

int j = 0;

string main\_class = "playgame";

while (getline(file, 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++;

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;

}

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]++;

dist\_val\_count[day]++;

dist\_val\_count[outlook]++;

dist\_val\_count[temp]++;

dist\_val\_count[humidity]++;

dist\_val\_count[wind]++;

val\_count[day][playGame]++;

val\_count[outlook][playGame]++;

val\_count[temp][playGame]++;

val\_count[humidity][playGame]++;

val\_count[wind][playGame]++;

}

double posR = mainClass["Yes"], negR = mainClass["No"];

double totR = posR + negR;

double mainC\_gain = -((posR / totR) \* log2(posR / totR) + (negR / totR) \*

log2(negR / totR));

cout << "Main Class Gain : " << mainC\_gain << "\n";

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

{

calculateGain(sub\_classes[i], mainC\_gain);

}

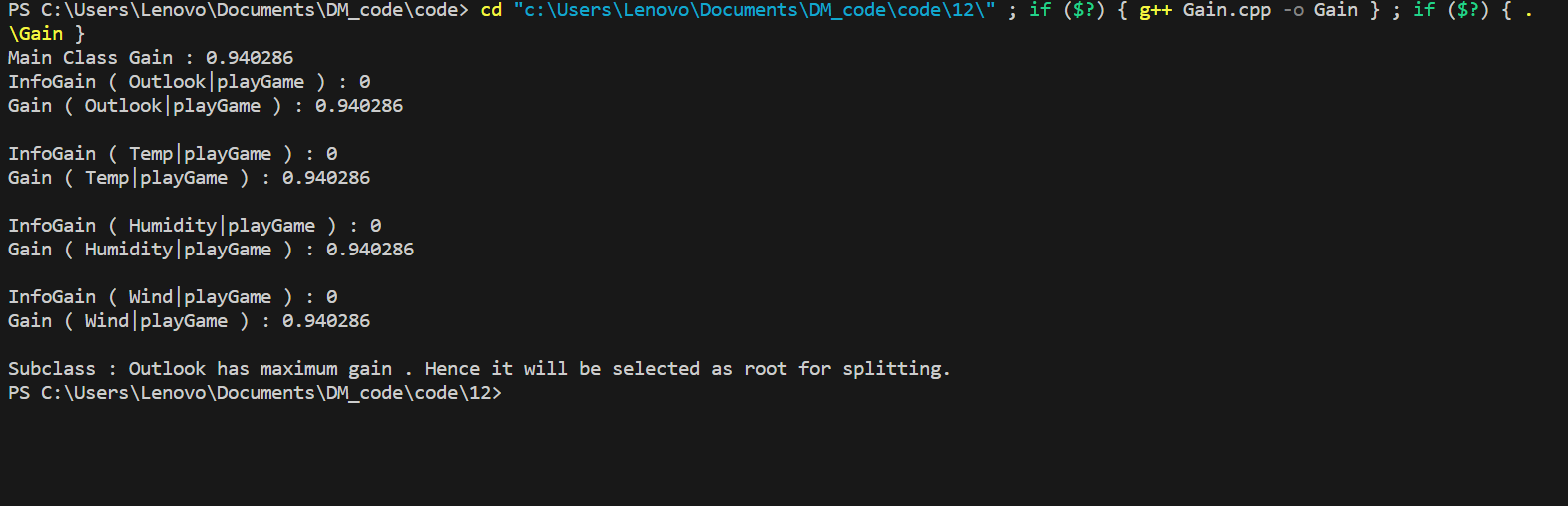
cout << "Subclass : " << root << " has maximum gain . Hence it will be selected as root for splitting.\n";

fw << "Subclass : " << root << "has maximum gain . Hence it will be selected as root for splitting.\n";

return 0;

}

**Output:**

****

**GiniIndex.cpp**

#include <bits/stdc++.h>

using namespace std;

double gini\_of\_class(double p1, double p2)

{

int tot = p1 + p2;

double tmp1 = (double)pow((p1 / tot), 2.0);

double tmp2 = (double)pow((p2 / tot), 2.0);

double ans = 1 - tmp1 - tmp2;

return ans;

}

double gini\_attribute(map<string, map<string, int>> attribute, double count)

{

double gini = 0.0;

for (auto i : attribute)

{

string val = i.first;

double play\_cnt = attribute[val]["Play"];

double NoPlay\_cnt = attribute[val]["NoPlay"];

double tot = play\_cnt + NoPlay\_cnt;

gini += (double)(tot / (double)count) \* (1 - (play\_cnt / tot) \* (play\_cnt / tot) - (NoPlay\_cnt / tot) \* (NoPlay\_cnt / tot));

}

return gini;

}

int main()

{

ifstream file("12gini.csv");

string line, word;

string outlook, temp, humidity, windy, mainclass;

map<string, int> parent;

map<string, map<string, int>> attribute;

int count = 0;

if (!file.is\_open())

{

perror("Error in opening input file : ");

return -1;

}

int i = 0;

string attributeName, name;

while (getline(file, line))

{

stringstream str(line);

getline(str, outlook, ',');

getline(str, temp, ',');

getline(str, humidity, ',');

getline(str, windy, ',');

getline(str, mainclass, ',');

int choice;

if (i == 0)

{

i++;

cout << "Enter Attribute Column Number : ";

cin >> choice;

cout << endl;

continue;

}

switch (choice)

{

case 1:

attributeName = outlook;

break;

case 2:

attributeName = temp;

break;

case 3:

attributeName = humidity;

break;

case 4:

attributeName = windy;

break;

// case 5:

// attributeName = mainclass;

// break;

default:

attributeName = outlook;

break;

}

parent[mainclass]++;

attribute[attributeName][mainclass]++;

count++;

}

int p1 = parent["Play"];

int p2 = parent["NoPlay"];

// cout << p1 << " " << p2 << endl;

double gini\_parent = gini\_of\_class(p1, p2);

cout << "Gini Index "

<< "(column class): " << gini\_parent << endl;

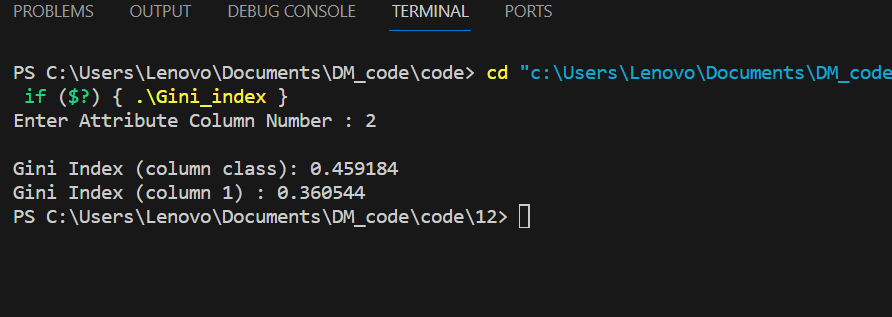
double gini = gini\_attribute(attribute, count);

cout << "Gini Index "

<< "(column " << i << ") : " << gini << endl;

}

**Output**

****

**Experiment No 13**

**Title:-** Bayes classification

Code :

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <sstream>

#include <ostream>

#include <bits/stdc++.h>

using namespace std;

int main()

{

string line, word;

ifstream file("13input.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;

if (file.is\_open())

{

int i = 0;

while (file >> line)

{

stringstream str(line);

if (i == 0)

{

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]]++;

for (int i = 1; i < n - 1; i++)

{

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

}

count++;

}

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 \n";

cin >> input;

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

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

{

cout << "no match\n";

goto again;

}

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

{

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;

}

}

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

{

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

}

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

cout << "sum " << sum << endl;

cout << "output---------" << endl;

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

{

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

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

}

}

else

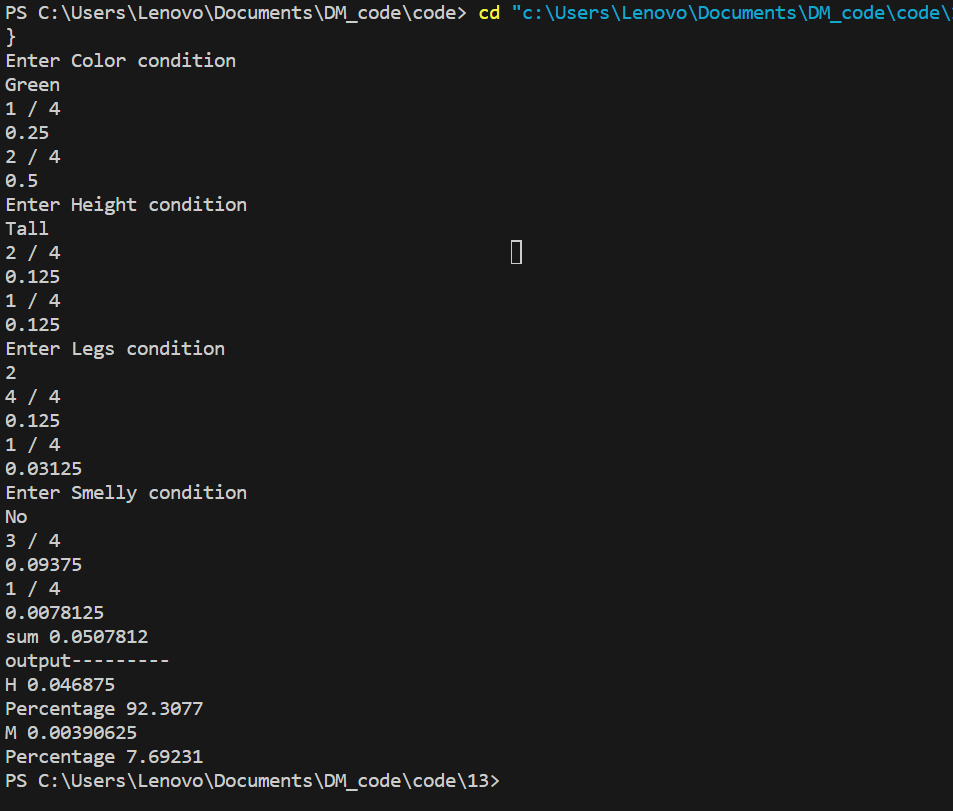
{

cout << "Could not open the file\n";

}

return 0;

}



Knime :



