

A REPORT OF

**“Data Mining Lab”**

**Code: 5IT451**

Submitted by

**Mr. Mohit Khairnar 2020BTEIT00038**

**Email:** mohitkhairnar9486@gmail.com

**Mobile:** 9028261668

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. Mohit Khairnar (2020BTEIT00038)**

is a record of the student’s own work carried out by him during the academic

year 2023-2024, as per the curriculum/syllabus laid down for OSS lab at Final

year B. Tech IT Sem-I.

**Dr R.R.Rathod**

**(Course Teacher)**

**Declaration**

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

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

Date: 23-11-2023 **Mr. Mohit Khairnar**

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. Mohit Khairnar

**Data Mining Lab Book**

**Name: Mohit Prakash Khairnar**

**PRN NO: 2020BTEIT00038**

**Subject: Data Mining**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Title** | **Page No.** |
| 1. | Study and use of different types of graphs and charts (use MS-XLS). | 6 |
| 2. | Perform Normalization of data (Min-max and Z-score). | 15 |
| 3. | Perform Binning of data. | 18 |
| 4. | Find the Info Gain of an attribute from given data. | 21 |
| 5. | Find the t and d weight of the data. | 24 |
| 6. | Find 5 no summary of a dataset. | 27 |
| 7. | Find frequent item sets from given transaction data. | 29 |
| 8. | Extend program 7, to find association rules. | 32 |
| 9. | Find correlation between items/entities. | 35 |
| 10. | Distance and cluster. | 37 |
| 11. | Agglomerative Hierarchical Single Linkage Clustering. | 40 |
| 12. | Attribute for classification A. Gain B. Gini index. | 42 |
| 13. | WAP for Bayes classification. | 45 |

Experiment No. 1

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

**Line chart:** A line chart, also known as a line graph or curve chart, is a graphical representation used to display data points over a continuous interval or time period. It is particularly effective in showing trends and patterns in data by connecting individual data points with straight lines. Line charts are widely used in various fields such as economics, science, finance, and statistics to visualize and interpret data.

# Key features of a line chart:

* X and Y Axes
* Data Points and Lines
* Trend Analysis
* Data Comparison
* Annotations and Labels
* Interpolation

**Example:** Stock market price trends.

|  |  |
| --- | --- |
| **Day (Deutsche Bank)** | **Price** |
| Monday | 886 |
| Tuesday | 890 |
| Wednesday | 850 |
| Thursday | 910 |
| Friday | 920 |
| Saturday | 930 |

**Column chart:** A column chart, also known as a vertical bar chart, is a graphical representation used to display data points using vertical bars. It is a popular chart type for comparing and contrasting categorical data or discrete values.

Deutsche Bank Stock

49000

48000

47000

46000

45000

44000

43000

42000

41000

40000

39000

01-08-2023 02-08-2023 03-08-2023 04-08-2023 05-08-2023 06-08-2023 07-08-2023

Column charts are widely used in various fields to visualize and communicate data patterns, distributions, and comparisons.

# Key features of a column chart:

* Vertical Bars
* Categories and Values
* Comparison and Contrast
* Data Labels
* Bar Grouping
* Visualizing Distribution

**Example:** Products sales comparison

|  |  |  |  |
| --- | --- | --- | --- |
| **Month** | **Product A sales** | **Product B sales** | **Product C sales** |
| January | 10000 | 12500 | 8000 |
| February | 11200 | 13800 | 9500 |
| March | 12500 | 14200 | 10200 |
| April | 11800 | 12900 | 9800 |
| May | 10300 | 13000 | 8500 |
| June | 11700 | 14500 | 9900 |
| July | 12800 | 15200 | 10800 |
| August | 13500 | 16000 | 11500 |
| September | 12000 | 14800 | 10000 |
| October | 11200 | 13700 | 9300 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | November |  | 10500 | 12300 |  | 8800 |  |
| December |  | 12300 | 14000 |  | 10100 |
|  |  |  | Products sales |  |  |  | |
| 60000 |  |  |  |  |  |  | |
| 50000 |  |  |  |  |  |  | |
| 40000 |  |  |  |  |  |  | |
| 30000 |  |  |  |  |  |  | |
| 20000 |  |  |  |  |  |  | |
| 10000 |  |  |  |  |  |  | |
| 0 |  |  |  |  |  |  | |
|  | 01-08-2023 | 02-08-2023 | 03-08-2023 04-08-2023 | 05-08-2023 | 06-08-2023 | 07-08-2023 | |
|  |  |  | Open High Low |  |  |  | |

**Pie chart:** A pie chart is a circular graphical representation used to display the distribution of data as a "pie" divided into slices, with each slice representing a proportion of the whole. Pie charts are commonly used to show how individual parts contribute to a whole and are effective for visualizing percentages or

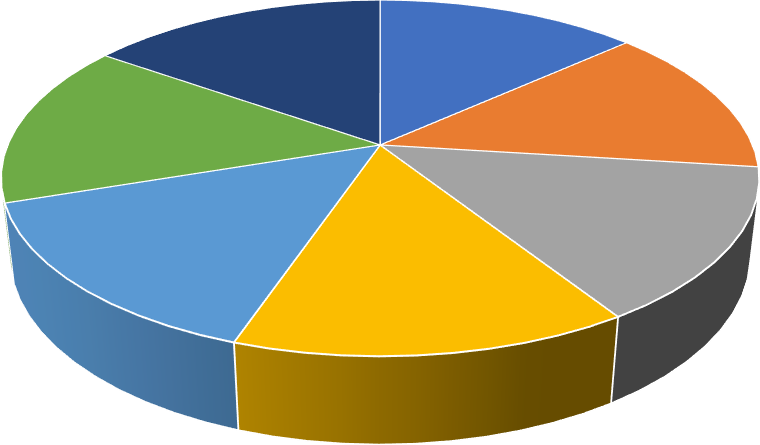
relative proportions of different categories within a dataset.

# Key features of a pie chart:

* Circular Layout
* Categories and Percentages
* Proportional Representation
* Limited Categories
* Data Labels and Legend

**Example:** Demographic distribution

|  |  |
| --- | --- |
| **Age Group** | **Population Percentage** |
| 0-14 | 25% |
| 15-24 | 15% |
| 25-34 | 20% |
| 35-44 | 18% |
| 45-54 | 12% |
| 55-64 | 6% |
| 65 and over | 4% |



Demographic Distribution

01-08-2023 02-08-2023 03-08-2023 04-08-2023 05-08-2023 06-08-2023 07-08-2023

**Bar chart:** A bar chart, also known as a bar graph, is a graphical representation used to display categorical data using rectangular bars. It is an effective way to compare and contrast different categories or groups by representing their values visually. Bar charts are widely used in various fields to convey

information, analyze patterns, and make data-driven decisions.

# Key features of a bar chart:

* Vertical or Horizontal Bars
* Categories and Values
* Comparison and Ranking
* Data Labels and Annotations
* Bar Grouping and Stacking
* Visualizing Distribution

**Example:** Student grades

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Subject** | **A** | **B** | **C** | **D** | **F** |
| Mathematics | 15% | 30% | 25% | 20% | 10% |
| Science | 20% | 25% | 30% | 15% | 10% |
| History | 10% | 35% | 30% | 15% | 10% |
| English | 25% | 30% | 20% | 15% | 10% |
| Art | 30% | 20% | 15% | 20% | 15% |

Student grades

05-08-2023

04-08-2023

03-08-2023

02-08-2023

01-08-2023

0

200000 400000 600000 800000 1000000 1200000 1400000 1600000

Volume Close Low High Open

**Scatter chart:** A scatter chart, also known as a scatter plot, is a graphical

representation used to display the relationship between two variables or sets of data. It uses a collection of individual data points, each representing a specific value for both the X-axis and the Y-axis. Scatter charts are particularly useful for identifying patterns, correlations, and outliers within data sets.

# Key features of a scatter chart:

* Data Points
* X and Y Axes
* Patterns and Relationships
* Outliers
* Data Spread and Clusters
* Data Labels and Annotations
* Regression Lines

**Example:** Hour study and score obtained.

|  |  |
| --- | --- |
| **Study Hours (hrs)** | **Exam Score (%)** |
| 2 | 65 |
| 3 | 70.00 |
| 4 | 75.00 |
| 5 | 80.00 |
| 6 | 85.00 |
| 7 | 88.00 |
| 8 | 92.00 |
| 9 | 95.00 |
| 10 | 98.00 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Radar chart:** A radar chart, also known as a spider chart or star plot, is a



Open

49000

48000

47000

46000

45000

44000

43000

42000

41000

31-07-2023 01-08-2023 02-08-2023 03-08-2023 04-08-2023 05-08-2023 06-08-2023 07-08-2023 08-08-2023

graphical representation used to display multivariate data in a two-dimensional format. It is particularly effective for comparing multiple variables or categories across a set of data points. A radar chart uses a central point as the origin and plots data values along radiating spokes, creating a web-like appearance.

# Key features of a radar chart:

* Spokes and Axes
* Data Points and Lines
* Comparison and Patterns
* Normalization
* Multiple Data Sets

**Example:** Skills of employee

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Employee** | **Communication** | **Problem**  **Solving** | **Leadership** | **Time**  **Management** | **Technical**  **Proficiency** |
| Mohit  Khairnar | 8 | 7 | 6 | 9 | 9 |
| Kshitish  Savkar | 9.00 | 8.00 | 7.00 | 8.00 | 7.00 |
| Abhishek  Deokar | 7.00 | 9.00 | 8.00 | 7.00 | 8.00 |
| Ayush  Wadalkar | 9.00 | 6.00 | 5.00 | 9.00 | 6.00 |
| Adwait  Samak | 6.00 | 8.00 | 9.00 | 6.00 | 9.00 |



Skills of employee

Open

High

Low

Close

Volume

01-08-2023

1500000

1000000

05-08-2023

500000

02-08-2023

0

04-08-2023

03-08-2023

**Histogram chart:** A histogram chart is a graphical representation used to

display the distribution of continuous data or a large number of data points

within specified intervals, known as bins. It provides insights into the frequency or count of data points falling within each bin, allowing for a visualization of the data's underlying distribution.

# Key features of a histogram chart:

* Bins and Intervals
* Frequency or Count
* Continuous Data
* Shape Analysis
* Skewness and Outliers
* Normalization
* Bar Width

**Example:** Height distribution

|  |  |
| --- | --- |
| **Height (inches)** | **Frequency** |
| 60-62 | 12 |
| 63-65 | 28.00 |
| 66-68 | 45.00 |
| 69-71 | 55.00 |
| 72-74 | 40.00 |
| 75-77 | 20.00 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |  |  |
|  | | | | | | | | |  |  |  |
|  | | | | | | |  |  |  |  |
|  | | | | |  |  |  |  |  |
|  | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Candlestick chart:** A candlestick chart is a graphical representation used to display financial data, particularly stock prices, over a specific time period. It

Height distribution

48000

47000

46000

45000

44000

43000

42000

41000

40000

39000

01-08-2023

02-08-2023

03-08-2023

04-08-2023

05-08-2023

06-08-2023

provides a comprehensive view of price movements within a chosen interval, offering insights into opening, closing, high, and low prices for a given asset or security. Candlestick charts are widely used in technical analysis to assess market trends and make informed trading decisions.

# Key features of a candlestick chart:

* Candles
* Body
* Wicks
* Time Intervals
* Patterns and Trends
* Technical Analysis

**Example:** Cryptocurrency analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Open** | **High** | **Low** | **Close** | **Volume** |
| 42000 | 41000 | 41000 | 43000 | 1200000.00 |
| 43000 | 43000.00 | 42000.00 | 44000.00 | 1350000.00 |
| 44000 | 45000.00 | 43000.00 | 45000.00 | 1180000.00 |
| 45000 | 47000.00 | 44000.00 | 46000.00 | 1250000.00 |
| 46000 | 48000.00 | 45000.00 | 47000.00 | 1300000.00 |
| 47000 | 49000.00 | 46000.00 | 48000.00 | 1220000.00 |
| 48000 | 50000.00 | 47000.00 | 49000.00 | 1180000.00 |

Open-high-low-close-volume

50000

48000

46000

44000

42000

40000

38000

1600000

1400000

1200000

1000000

800000

600000

400000

200000

0

1 2 3 4 5 6 7

Open High Low Close Volume

**Experiment No. 2**

**Program:**

import csv

import os

#Reads content from csv file

def read\_csv(file\_path):

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

reader = csv.reader(file)

header = next(reader)

data = [list(map(float, row)) for row in reader]

return header, data

#Performs min-max normalization

def min\_max\_scaling(data, min\_val, max\_val):

for i in range(len(data[0])):

col = [row[i] for row in data]

min\_col, max\_col = min(col), max(col)

for j in range(len(data)):

data[j][i] = (data[j][i] - min\_col) / (max\_col - min\_col) \* (max\_val - min\_val) + min\_val

#Performs z-score normalization

def z\_score\_normalization(data):

for i in range(len(data[0])):

col = [row[i] for row in data]

mean\_col = sum(col) / len(col)

std\_col = (sum((x - mean\_col) \*\* 2 for x in col) / len(col)) \*\* 0.5

for j in range(len(data)):

data[j][i] = (data[j][i] - mean\_col) / std\_col

def write\_to\_csv(header, original\_data, min\_max\_normalized, z\_score\_normalized, output\_file):

with open(output\_file, 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(header + ['Min-Max Normalization', 'Z-Score Normalization'])

for row in zip(original\_data, min\_max\_normalized, z\_score\_normalized):

# Remove square brackets from each entry

row = [str(item).strip("[]") for item in row]

writer.writerow(row)

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

current\_dir = os.getcwd()

file\_path = os.path.join(current\_dir, 'input.csv')

header, data = read\_csv(file\_path)

# Min-Max scaling

min\_val, max\_val = 0, 1

min\_max\_normalized = [row[:] for row in data]

min\_max\_scaling(min\_max\_normalized, min\_val, max\_val)

# z-score

z\_score\_normalized = [row[:] for row in data]

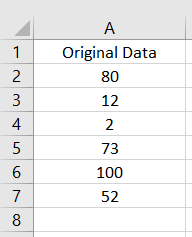
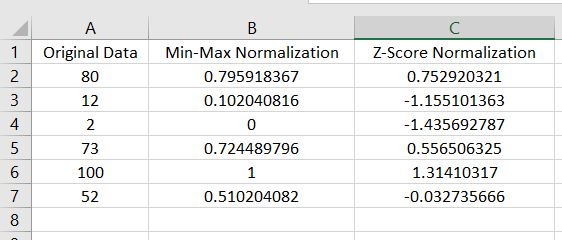
z\_score\_normalization(z\_score\_normalized)

output\_file\_path = os.path.join(current\_dir, 'normalized\_output.csv')

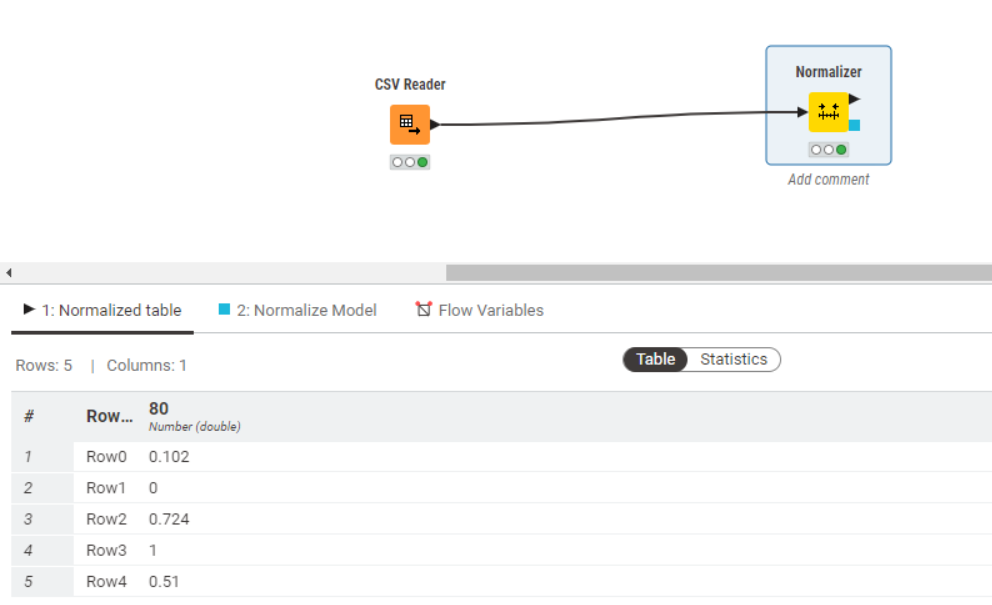
write\_to\_csv(header, data, min\_max\_normalized, z\_score\_normalized, output\_file\_path)

print(f"\nNormalized data written to {output\_file\_path}")

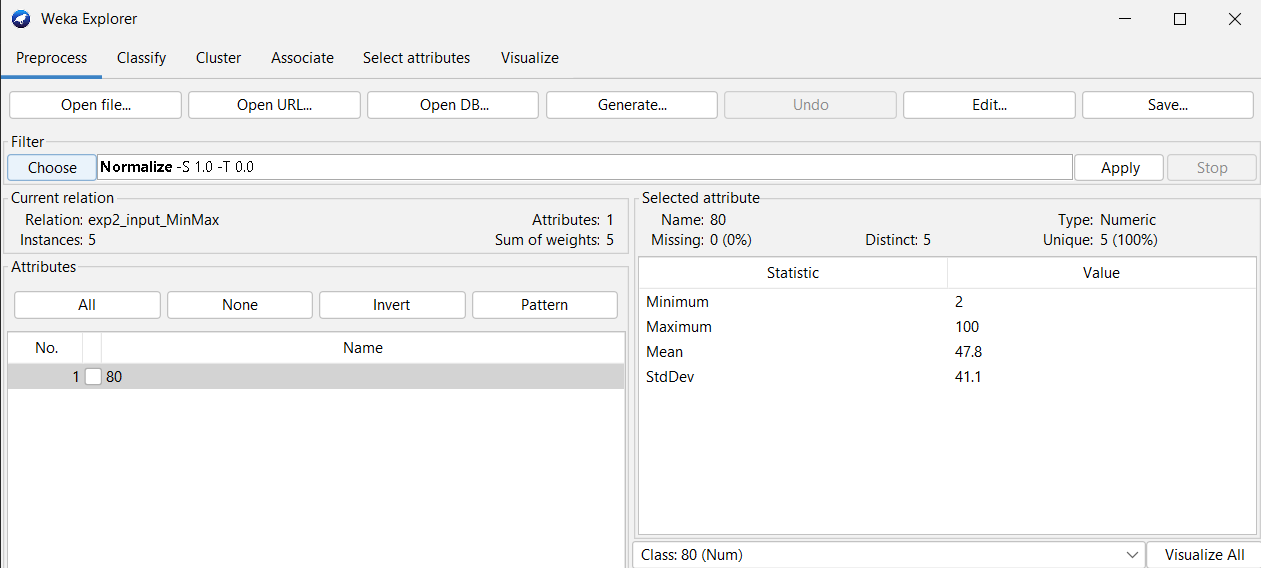
**Input: Output:**

**** ****

**Knime output:**

****

**Weka output:**

****

**Experiment No. 3**

**Program:**

import pandas as pd

def perform\_binning(data, column\_to\_bin, num\_bins, bin\_labels, method):

df = data.copy()

bin\_column\_name = f'binned\_{method}'

if method == 'width':

df[bin\_column\_name] = pd.cut(df[column\_to\_bin], bins=num\_bins, labels=bin\_labels, include\_lowest=True)

elif method == 'quantile':

df[bin\_column\_name] = pd.qcut(df[column\_to\_bin], q=num\_bins, labels=bin\_labels)

return df

def display\_and\_save\_binned\_data(df, bin\_column\_name, output\_file\_path):

print(f"Binned Data ({bin\_column\_name}):")

for label, group in df.groupby(bin\_column\_name):

print(f"{label}: {list(group[column\_to\_bin])}")

df.to\_csv(output\_file\_path, index=False)

print(f"\nBinned data saved to {output\_file\_path}\n")

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

# Read the CSV file

input\_file\_path = 'input\_data.csv'

output\_file\_path\_width = 'equal-width.csv'

output\_file\_path\_quantile = 'equal-frequency.csv'

# Load the data into a DataFrame

df = pd.read\_csv(input\_file\_path)

# Extract the column to be binned

column\_to\_bin = 'value' # Change this to the column you want to bin

# Get user input for binning parameters

num\_bins\_width = int(input("Enter the number of bins for equal-width binning: "))

bin\_labels\_width = [f'Bin{i+1}' for i in range(num\_bins\_width)] # You can customize bin labels

num\_bins\_quantile = int(input("Enter the number of bins for equal-frequency binning: "))

bin\_labels\_quantile = [f'Bin{i+1}' for i in range(num\_bins\_quantile)] # You can customize bin labels

# Perform equal-width binning

df\_width = perform\_binning(df, column\_to\_bin, num\_bins\_width, bin\_labels\_width, 'width')

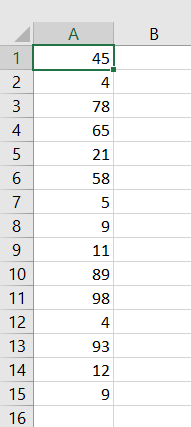
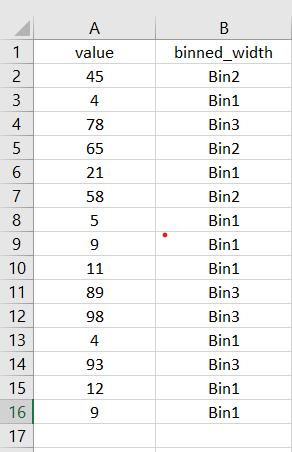
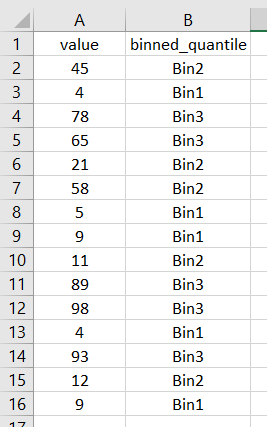
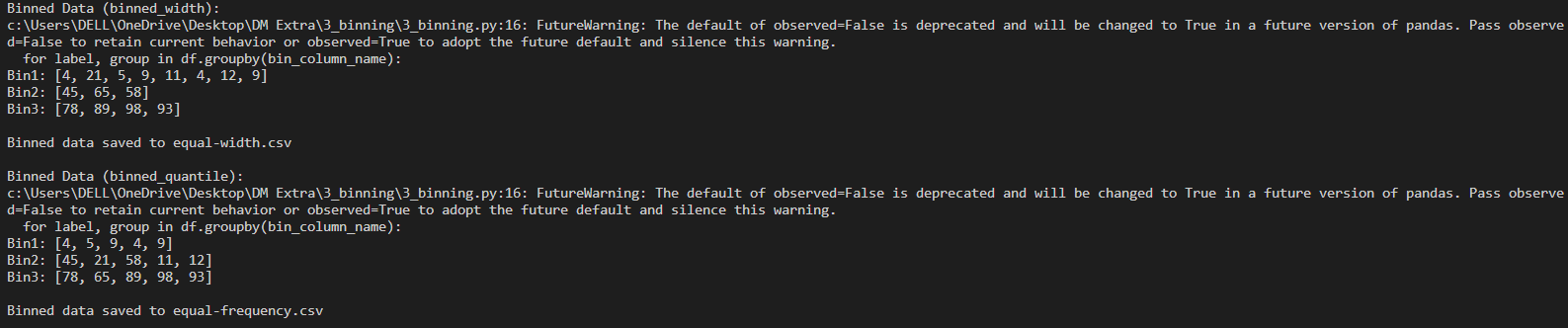
display\_and\_save\_binned\_data(df\_width, 'binned\_width', output\_file\_path\_width)

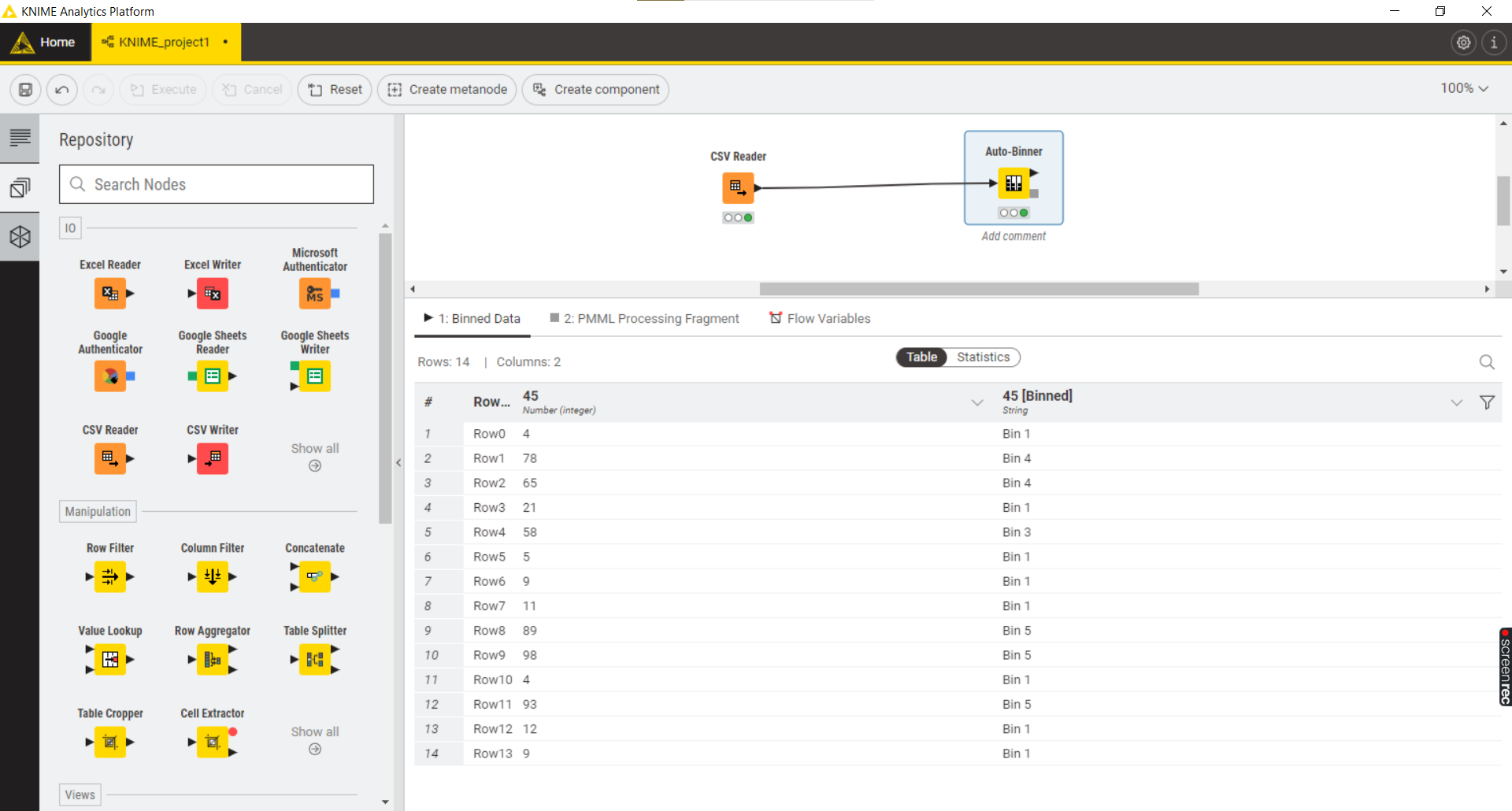
# Perform equal-frequency binning (quantile binning)

df\_quantile = perform\_binning(df, column\_to\_bin, num\_bins\_quantile, bin\_labels\_quantile, 'quantile')

display\_and\_save\_binned\_data(df\_quantile, 'binned\_quantile', output\_file\_path\_quantile)

**Input: Output:**

**** ****  

**Knime output:**

**Experiment No. 4**

**Program:**

import pandas as pd

import math

def calculate\_entropy(data, class\_column):

class\_counts = data[class\_column].value\_counts()

entropy = 0

for count in class\_counts:

probability = count / len(data)

entropy -= probability \* math.log2(probability)

return entropy

def calculate\_information\_gain(data, attribute, class\_column):

total\_entropy = calculate\_entropy(data, class\_column)

attribute\_entropy = 0

attribute\_values = data[attribute].unique()

for value in attribute\_values:

subset = data[data[attribute] == value]

subset\_entropy = calculate\_entropy(subset, class\_column)

weight = len(subset) / len(data)

attribute\_entropy += weight \* subset\_entropy

information\_gain = total\_entropy - attribute\_entropy

return information\_gain

def get\_user\_input():

attribute = input("Enter the attribute for which you want to calculate Information Gain: ")

return attribute

def display\_result(attribute, info\_gain):

print(f"\nInformation Gain for '{attribute}': {info\_gain}\n")

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

# Load data from CSV

file\_path = 'input\_data.csv' # Replace with your CSV file path

data = pd.read\_csv(file\_path)

# Get user input for attribute

attribute\_name = get\_user\_input()

# Replace 'PlayTennis' with the actual column name representing the class

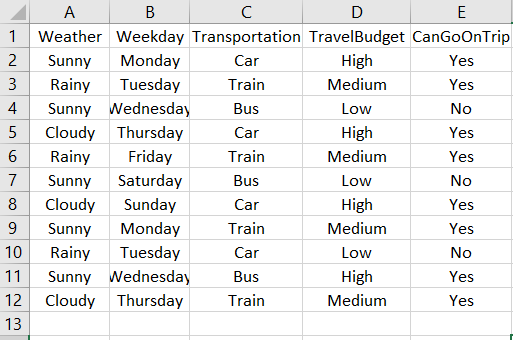
class\_column\_name = 'CanGoOnTrip'

info\_gain = calculate\_information\_gain(data, attribute\_name, class\_column\_name)

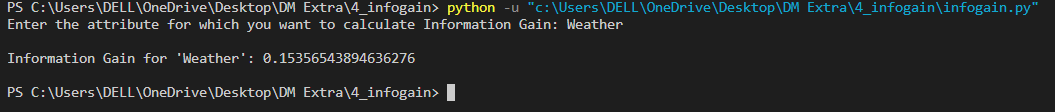
# Display the result

display\_result(attribute\_name, info\_gain)

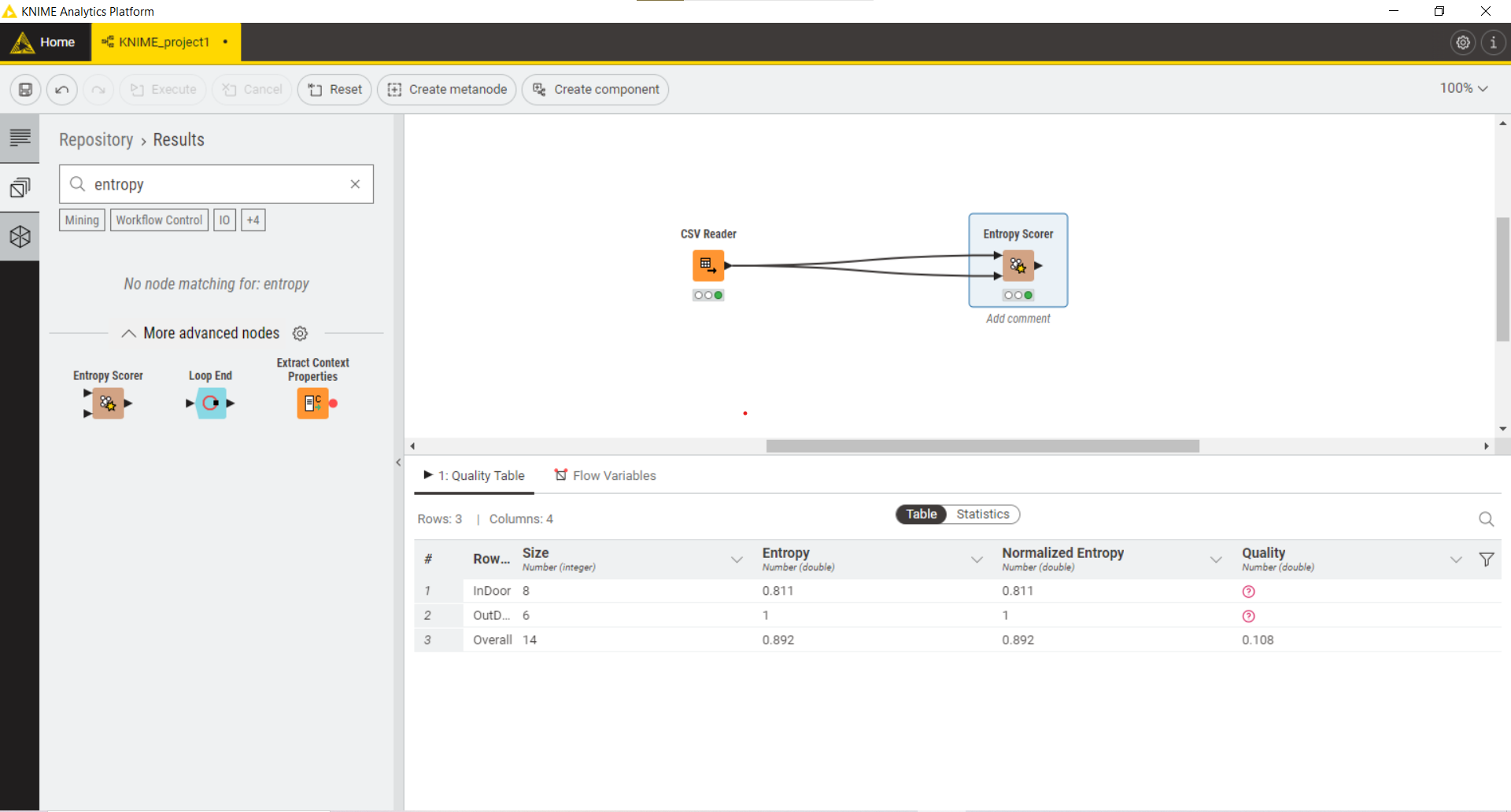
**Input:**



**Output:**

****

**Knime output:**

****

**Experiment No. 5**

**Program:**

import csv

classrowcolMap = {}

colMap = {}

rowMap = {}

with open("input.csv", mode="r") as file:

reader = csv.reader(file)

i = 0

for row in reader:

if i == 0:

i += 1

continue

row\_name, col\_name, count = row

val = int(count)

if row\_name not in classrowcolMap:

classrowcolMap[row\_name] = {}

classrowcolMap[row\_name][col\_name] = val

colMap[col\_name] = colMap.get(col\_name, 0) + val

rowMap[row\_name] = rowMap.get(row\_name, 0) + val

for r in rowMap:

for c in colMap:

print(f"{r}-{c}: {classrowcolMap[r][c]}")

for r in rowMap:

print(f"{r} -> {rowMap[r]}")

for c in colMap:

print(f"{c} -> {colMap[c]}")

colSum = sum(colMap.values())

rowSum = sum(rowMap.values())

print(f"colSum: {colSum}")

print(f"rowSum: {rowSum}")

with open("output.csv", mode="w", newline='') as fw:

writer = csv.writer(fw)

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

writer.writerow(["", "Count", "t-weight", "d-weight", "Count", "t-weight", "d-weight", "Count", "t-weight", "d-weight"])

for r in rowMap:

row = r

row\_data = [row]

for c in colMap:

col = c

row\_data.extend([

classrowcolMap[row][col],

(classrowcolMap[row][col] / rowMap[row]) \* 100,

(classrowcolMap[row][col] / colMap[col]) \* 100

])

row\_data.extend([rowMap[row], (rowMap[row] / rowMap[row]) \* 100, (rowMap[row] / colSum) \* 100])

writer.writerow(row\_data)

total\_row = ["Total"]

for c in colMap:

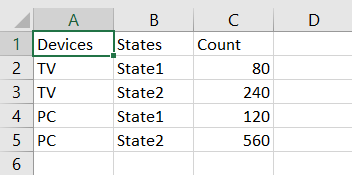
col = c

total\_row.extend([colMap[col], (colMap[col] / colSum) \* 100, (colMap[col] / colMap[col]) \* 100])

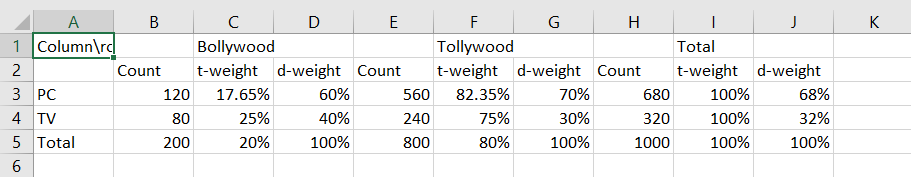
total\_row.extend([colSum, 100, 100])

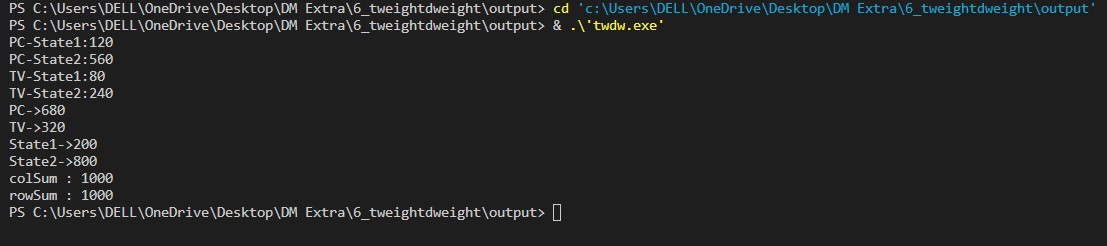
writer.writerow(total\_row)

**Input:**

****

**Output:**

****

****

**Experiment No. 6**

**Problem:**

import pandas as pd

def calculate\_five\_number\_summary(data):

summary = data.describe().transpose()[['min', '25%', '50%', '75%', 'max']]

summary.columns = ['Min', 'Q1', 'Median', 'Q3', 'Max']

return summary

def main(input\_csv, output\_csv):

# Read CSV file

data = pd.read\_csv(input\_csv)

# Calculate 5-number summary

summary = calculate\_five\_number\_summary(data)

# Write results to a new CSV file

summary.to\_csv(output\_csv)

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

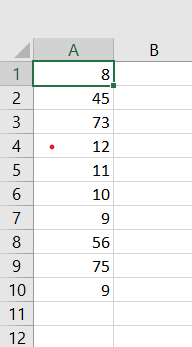
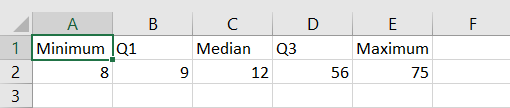
input\_csv = "input\_data.csv" # Replace with the path to your input CSV file

output\_csv = "output\_summary.csv" # Replace with the desired output CSV file path

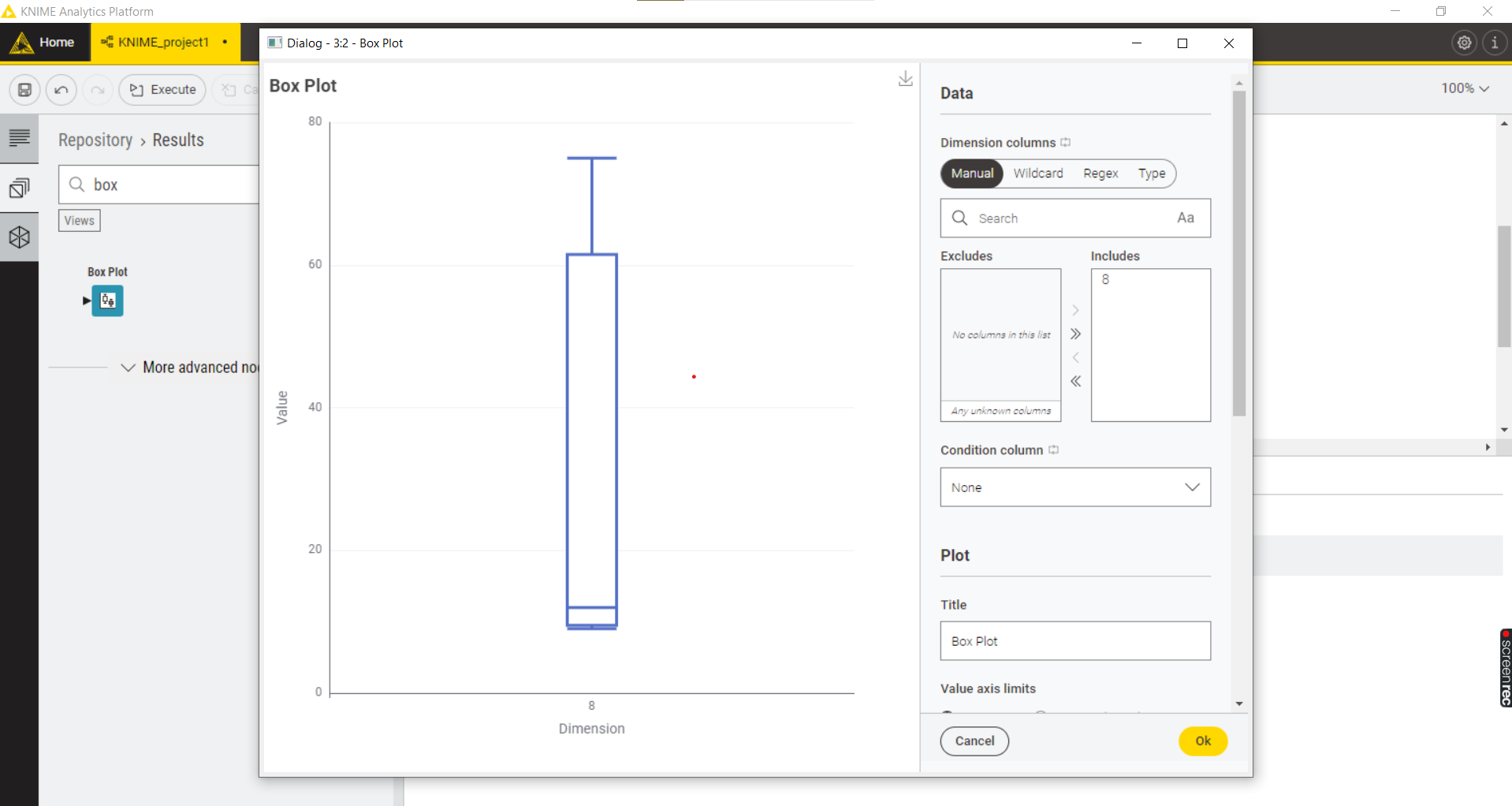
main(input\_csv, output\_csv)

# 1 2 3 4 5 6 7 8

**Input: Output:**

****

**Knime output:**

****

**Experiment No. 7**

**Program:**

import csv

from collections import defaultdict

import pandas as pd

from mlxtend.frequent\_patterns import apriori

from mlxtend.preprocessing import TransactionEncoder

def read\_transactions(file\_path):

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

reader = csv.reader(csvfile)

return defaultdict(list, {i + 1: [item.strip() for item in row if item.strip()] for i, row in enumerate(reader) if any(row)})

def mine\_frequent\_itemsets(transactions, min\_support):

te = TransactionEncoder()

transformed\_df = te.fit\_transform(list(transactions.values()))

frequent\_itemsets = apriori(pd.DataFrame(transformed\_df, columns=te.columns\_), min\_support=min\_support, use\_colnames=True)

frequent\_itemsets['itemsets'] = frequent\_itemsets['itemsets'].apply(set)

return frequent\_itemsets

def main():

input\_file\_path = 'input.csv'

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

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

frequent\_itemsets = mine\_frequent\_itemsets(transactions, min\_support)

print("\nFrequent Itemsets:")

print(frequent\_itemsets)

output\_file\_path = 'output\_frequent\_itemsets.csv'

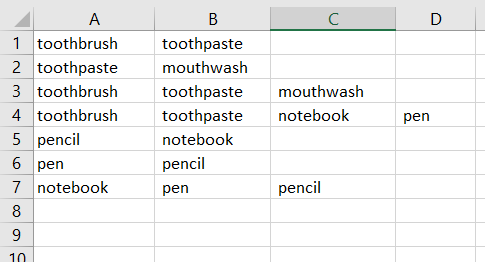
frequent\_itemsets.to\_csv(output\_file\_path, index=False)

print(f"\nFrequent itemsets saved to {output\_file\_path}")

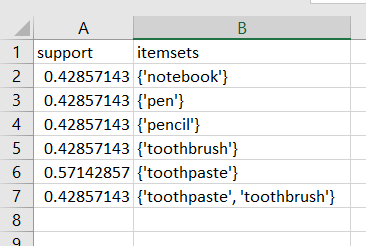
if \_\_name\_\_ == "\_\_main\_\_":

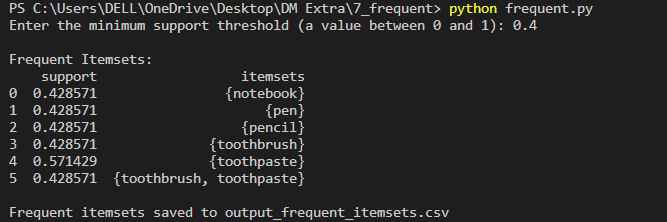
main()

**Input:**

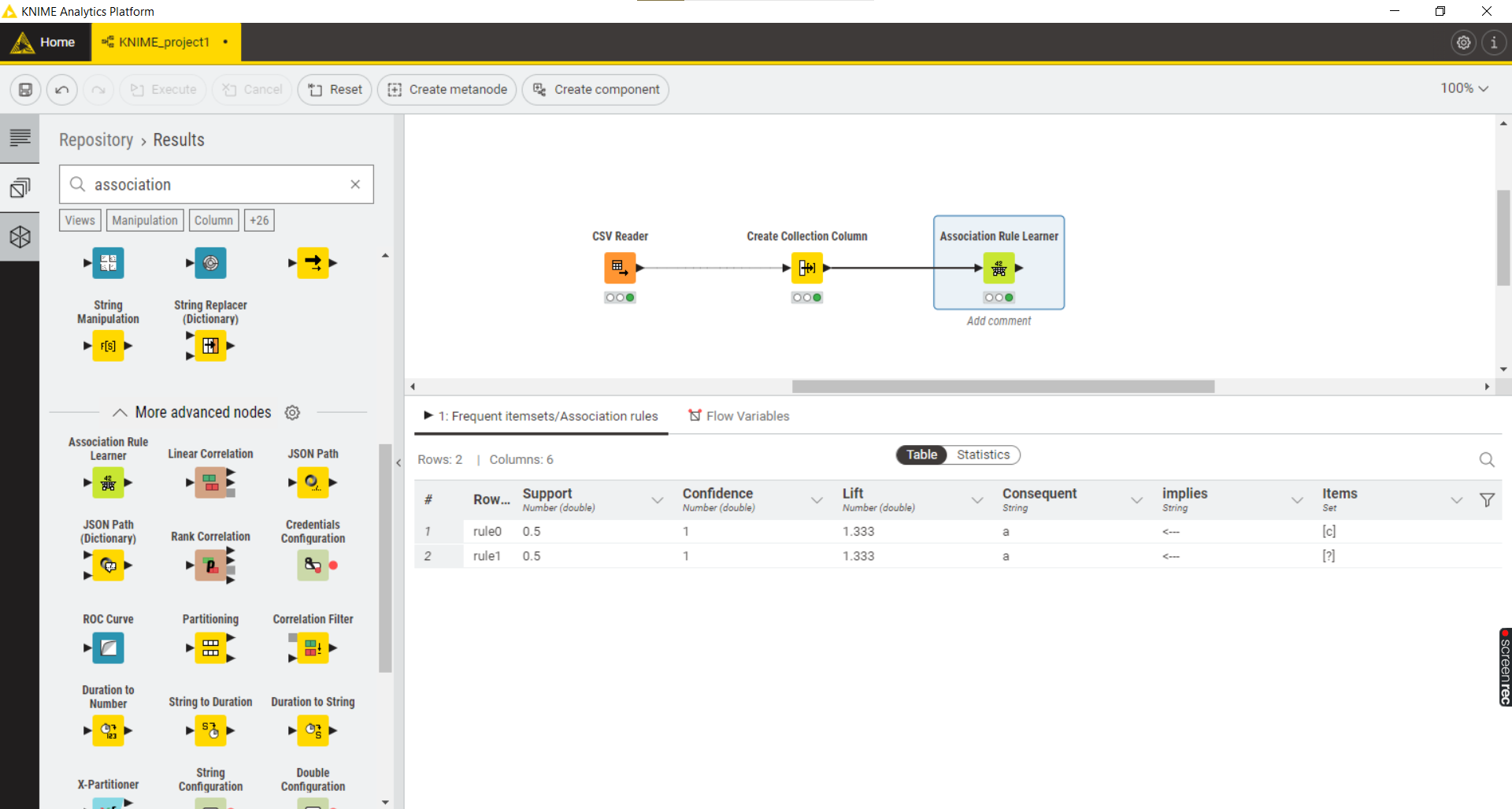
****

**Output:**

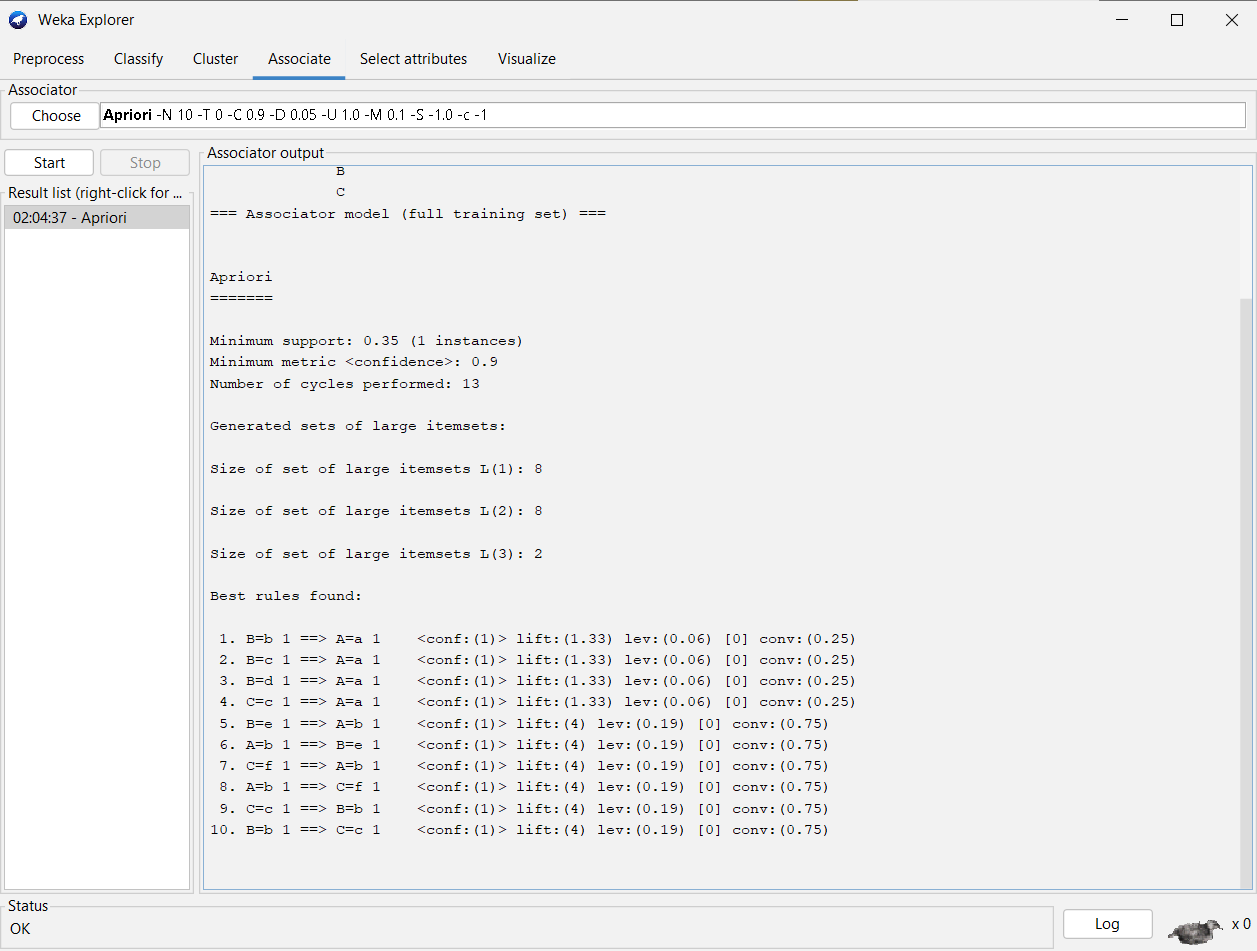
****

****

**Knime output:**

****

**Weka output:**

****

**Experiment No. 8**

**Program:**

import csv

from collections import defaultdict

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

from mlxtend.preprocessing import TransactionEncoder

def read\_transactions(file\_path):

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

reader = csv.reader(csvfile)

return defaultdict(list, {i + 1: [item.strip() for item in row if item.strip()] for i, row in enumerate(reader) if any(row)})

def mine\_frequent\_itemsets(transactions, min\_support):

te = TransactionEncoder()

transformed\_df = te.fit\_transform(list(transactions.values()))

frequent\_itemsets = apriori(pd.DataFrame(transformed\_df, columns=te.columns\_), min\_support=min\_support, use\_colnames=True)

frequent\_itemsets['itemsets'] = frequent\_itemsets['itemsets'].apply(set)

return frequent\_itemsets

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

rules\_df = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=min\_confidence)

columns\_to\_exclude = ['leverage', 'conviction', 'zhangs\_metric', 'lift']

rules\_df = rules\_df.drop(columns=columns\_to\_exclude, errors='ignore')

rules\_df['antecedents'] = rules\_df['antecedents'].apply(set)

rules\_df['consequents'] = rules\_df['consequents'].apply(set)

return rules\_df

def main():

input\_file\_path = 'input.csv'

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

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

frequent\_itemsets = mine\_frequent\_itemsets(transactions, min\_support)

print("\nFrequent Itemsets:")

print(frequent\_itemsets)

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

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

print("\nAssociation Rules:")

print(association\_rules\_df)

output\_rules\_file\_path = 'output\_association\_rules.csv'

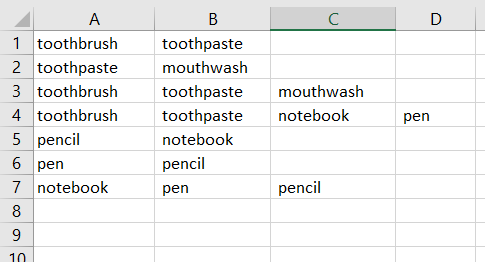
association\_rules\_df.to\_csv(output\_rules\_file\_path, index=False)

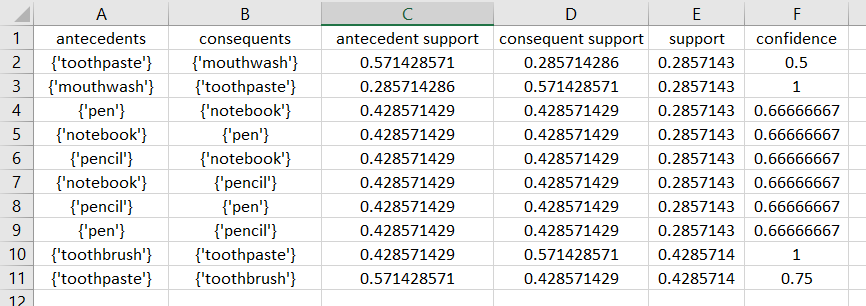
print(f"\nAssociation rules saved to {output\_rules\_file\_path}")

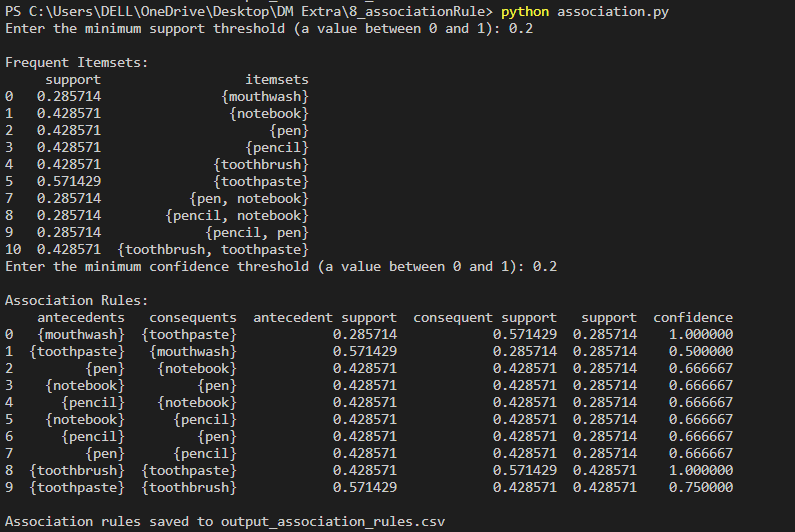
if \_\_name\_\_ == "\_\_main\_\_":

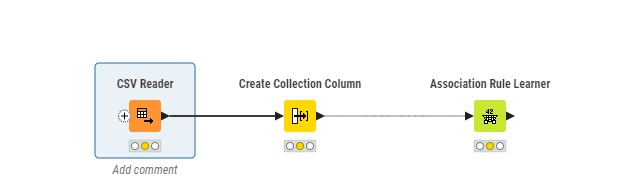
main()

**Input:**

****

**Output:** ****

****

**Knime output:**

**Experiment No. 9**

**Program:**

import csv

import math

def load\_data(file\_path):

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

data = list(csv.reader(file))

return data

def calculate\_mean(data):

return sum(data) / len(data)

def calculate\_correlation(x, y):

n = len(x)

mean\_x = calculate\_mean(x)

mean\_y = calculate\_mean(y)

numerator = sum((xi - mean\_x) \* (yi - mean\_y) for xi, yi in zip(x, y))

denominator\_x = math.sqrt(sum((xi - mean\_x) \*\* 2 for xi in x))

denominator\_y = math.sqrt(sum((yi - mean\_y) \*\* 2 for yi in y))

correlation = numerator / (denominator\_x \* denominator\_y)

return correlation

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

file\_path = 'input\_data.csv'

# Load data from CSV file

data = load\_data(file\_path)

# Extract columns for X and Y variables

x\_column = [float(row[1]) for row in data[1:]] # Assuming the first column is X

y\_column = [float(row[2]) for row in data[1:]] # Assuming the second column is Y

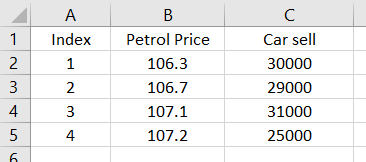
# Calculate correlation

correlation\_coefficient = calculate\_correlation(x\_column, y\_column)

# Print the result

print(f"Correlation Coefficient: {correlation\_coefficient}")

**Input:**

****

**Output:** 

**Experiment No. 10**

**Program:**

import pandas as pd

from scipy.spatial.distance import cdist

def compute\_cluster\_center(points):

# Assuming 'points' is a DataFrame with columns representing multidimensional points

center = points.mean(axis=0)

return center

def compute\_distances(points, cluster\_center):

# Compute distances between each point and the cluster center

distances = cdist(points, [cluster\_center], metric='euclidean')

return distances

# Load data from CSV

file\_path = 'input\_data.csv' # Replace with your CSV file path

data = pd.read\_csv(file\_path)

# Assuming columns 'X' and 'Y' represent the coordinates of the points

points = data[['X', 'Y']]

# Compute the initial center of the cluster

initial\_cluster\_center = compute\_cluster\_center(points)

# Print the initial cluster center

print("Initial Cluster Center:")

print(initial\_cluster\_center)

# Print distances from each point to the initial cluster center

initial\_distances = compute\_distances(points, initial\_cluster\_center)

print("\nDistances from Initial Cluster Center:")

print(initial\_distances)

# Find the nearest point as the updated cluster center

nearest\_point\_index = initial\_distances.argmin()

updated\_cluster\_center = points.iloc[nearest\_point\_index]

# Print the updated cluster center

print("\nUpdated Cluster Center:")

print(updated\_cluster\_center)

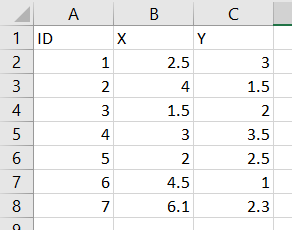
# Compute distances from the updated cluster center

updated\_distances = compute\_distances(points, updated\_cluster\_center)

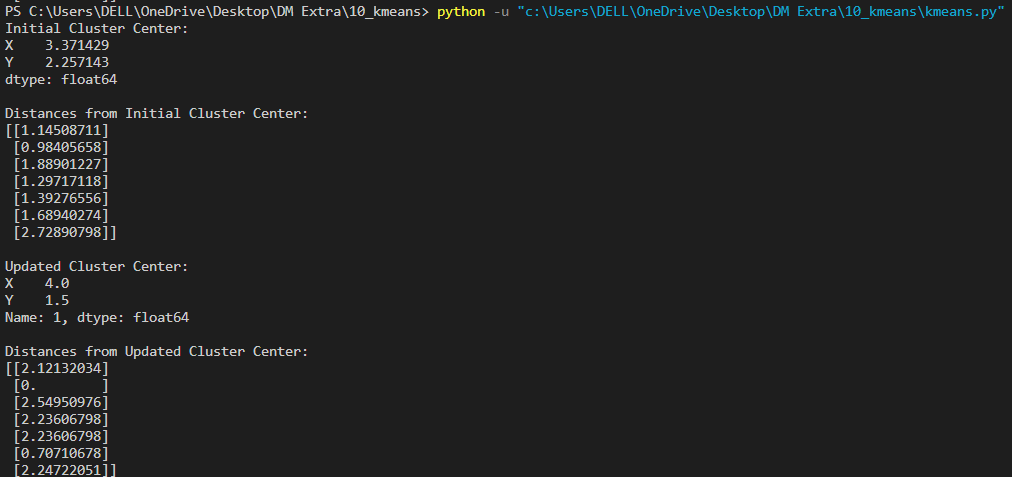
print("\nDistances from Updated Cluster Center:")

print(updated\_distances)

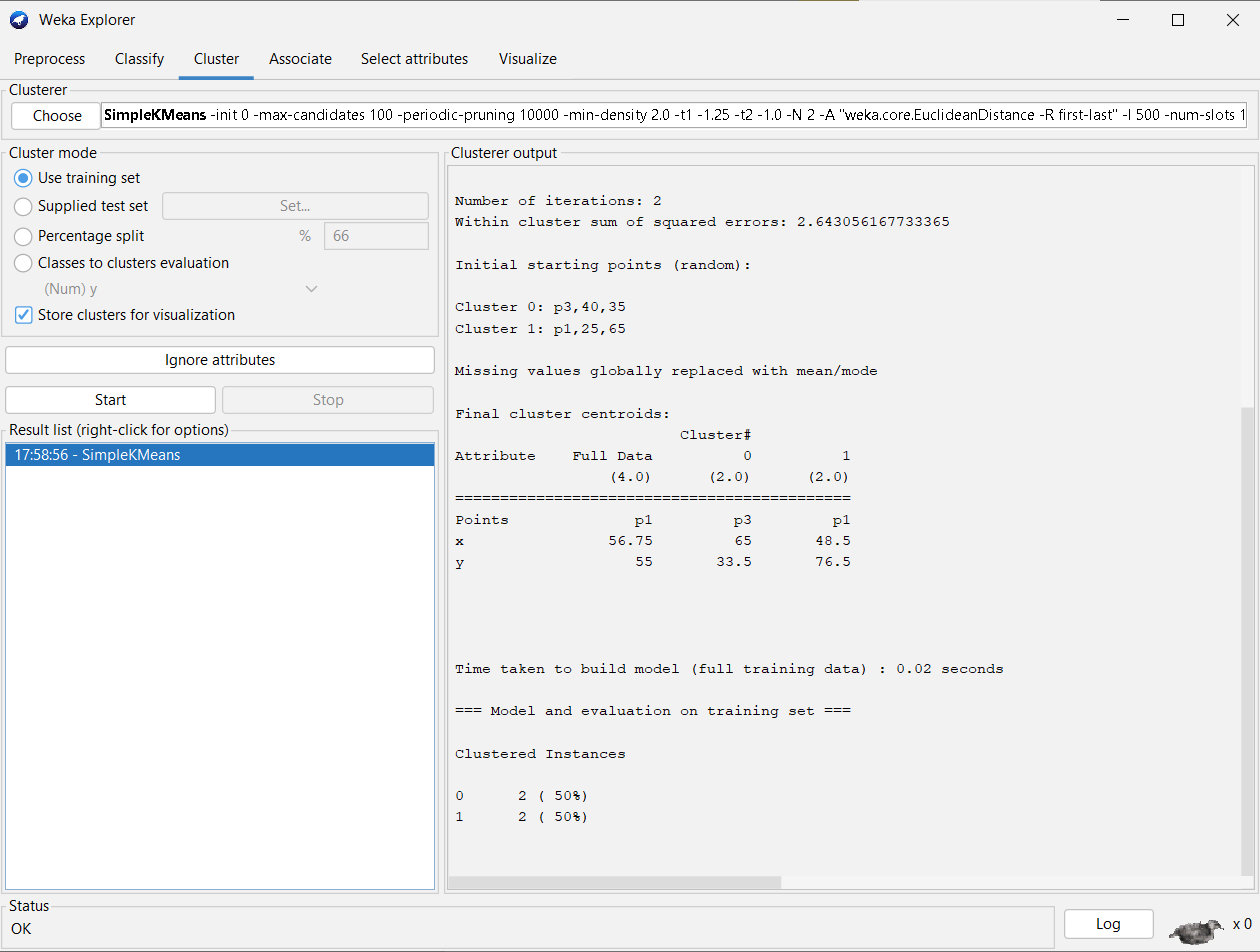
**Input:**

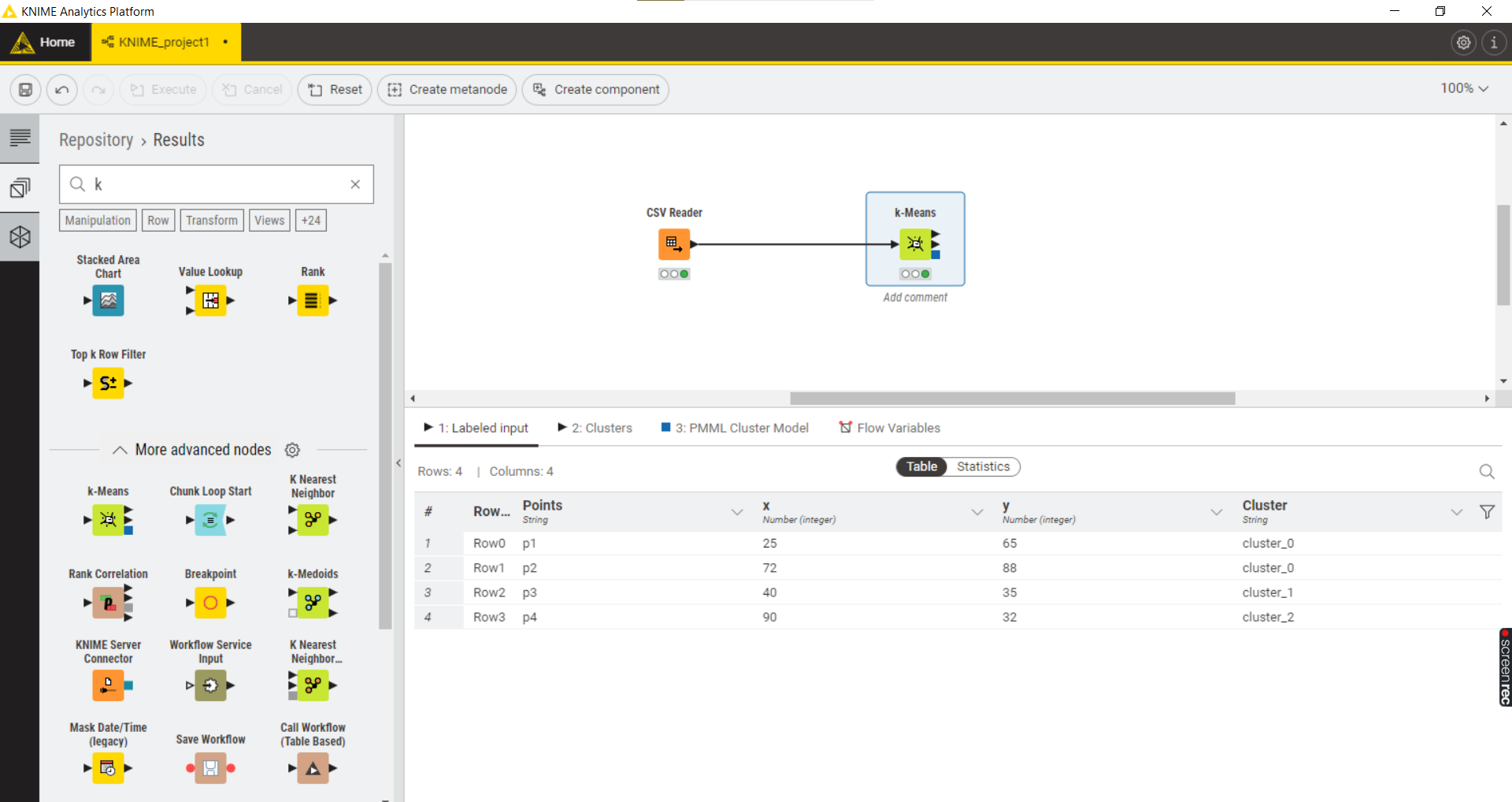
****

**Output:**

****

**Weka output:**

****

**Knime output: **

**Experiment No. 11**

**Program:**

import pandas as pd

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

def load\_data(file\_path):

return pd.read\_csv(file\_path, index\_col=0)

def hierarchical\_clustering(data, linkage\_method='single'):

# Perform hierarchical clustering

clusters = linkage(data, method=linkage\_method)

# Plot the dendrogram (optional)

dendrogram(clusters, labels=data.index, leaf\_rotation=90)

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('Data Points')

plt.ylabel('Distance')

plt.show()

return clusters

def save\_clusters\_to\_csv(clusters, output\_file):

pd.DataFrame(clusters).to\_csv(output\_file, header=False, index=False)

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

file\_path = 'input\_data.csv'

output\_file = 'hierarchical\_clusters.csv'

# Load data from CSV file

data = load\_data(file\_path)

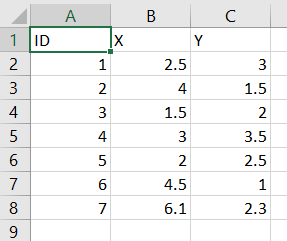
# Perform hierarchical clustering

clusters = hierarchical\_clustering(data, linkage\_method='single')

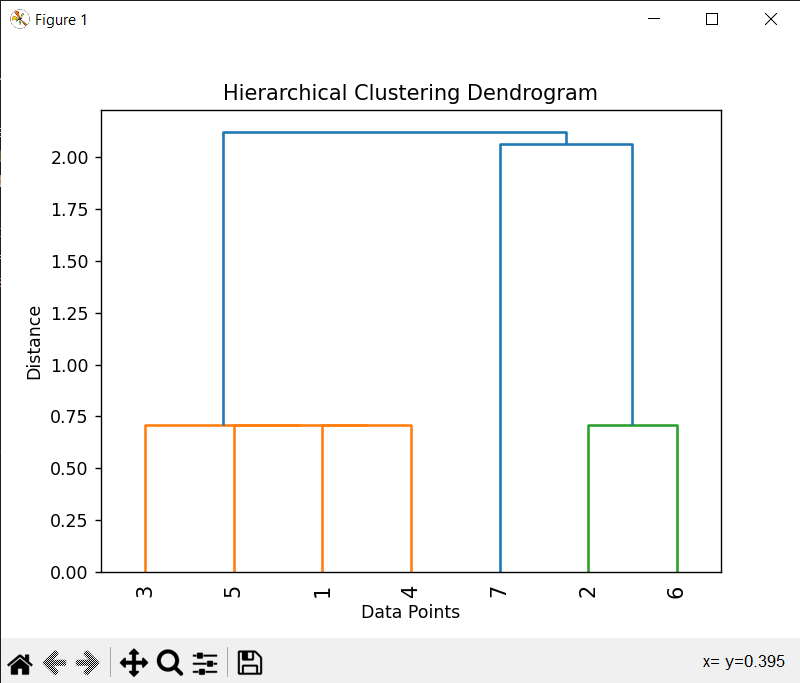
# Save clusters to CSV file

save\_clusters\_to\_csv(clusters, output\_file)

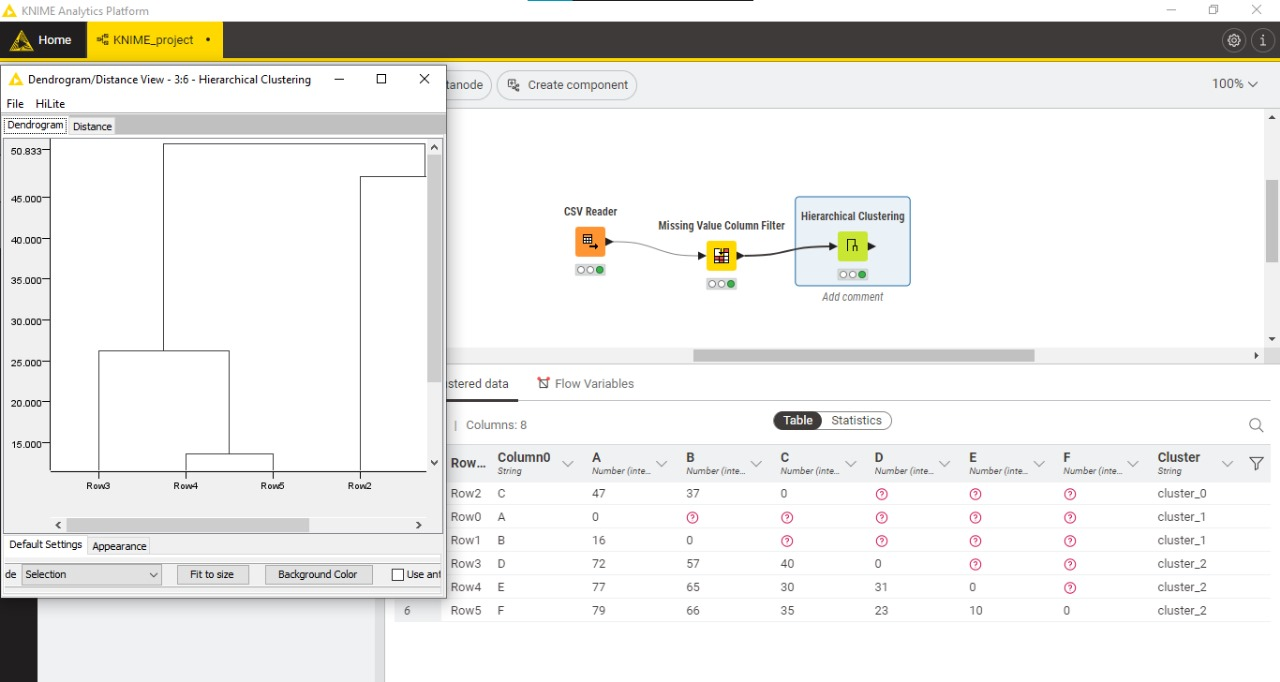
**Input:**

****

**Output:**

****

**Knime output:**

****

**Experiment No. 12**

**Program:**

import pandas as pd

import math

def calculate\_entropy(data, class\_column):

class\_counts = data[class\_column].value\_counts()

entropy = sum((-count / len(data)) \* math.log2(count / len(data)) for count in class\_counts)

return entropy

def calculate\_gini\_index(data, class\_column):

class\_counts = data[class\_column].value\_counts()

gini\_index = 1 - sum((count / len(data))\*\*2 for count in class\_counts)

return gini\_index

def calculate\_gain(data, attribute, class\_column):

total\_entropy = calculate\_entropy(data, class\_column)

attribute\_entropy = sum((len(subset) / len(data)) \* calculate\_entropy(subset, class\_column)

for \_, subset in data.groupby(attribute))

information\_gain = total\_entropy - attribute\_entropy

return information\_gain

def calculate\_gini\_gain(data, attribute, class\_column):

total\_gini = calculate\_gini\_index(data, class\_column)

attribute\_gini = sum((len(subset) / len(data)) \* calculate\_gini\_index(subset, class\_column)

for \_, subset in data.groupby(attribute))

gini\_gain = total\_gini - attribute\_gini

return gini\_gain

# Load data from CSV

file\_path = 'C:\\Users\\DELL\\OneDrive\\Desktop\\DM Extra\\12\_giniIndex\\data.csv' # Replace with your CSV file path

data = pd.read\_csv(file\_path)

class\_column\_name = 'CanGoOnTrip'

# Calculate information gain for each attribute

information\_gains = {attribute: calculate\_gain(data, attribute, class\_column\_name)

for attribute in data.columns if attribute != class\_column\_name}

# Identify the attribute with the maximum information gain

max\_gain\_attribute = max(information\_gains, key=information\_gains.get)

# Calculate Gini index for the attribute with the maximum information gain

max\_gain\_gini\_index = calculate\_gini\_index(data, max\_gain\_attribute)

# Print information gain of each attribute

print("Information Gain for Each Attribute:")

for attribute, gain in information\_gains.items():

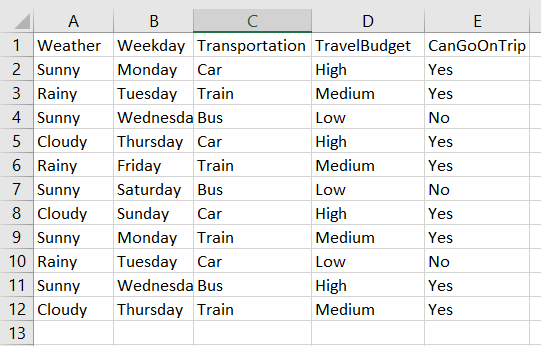
print(f"{attribute}: {gain}")

print(f"\nFor Attribute with Maximum Information Gain ('{max\_gain\_attribute}'):")

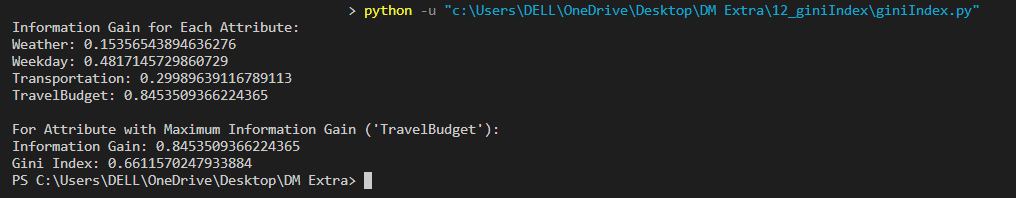
print(f"Information Gain: {information\_gains[max\_gain\_attribute]}")

print(f"Gini Index: {max\_gain\_gini\_index}")

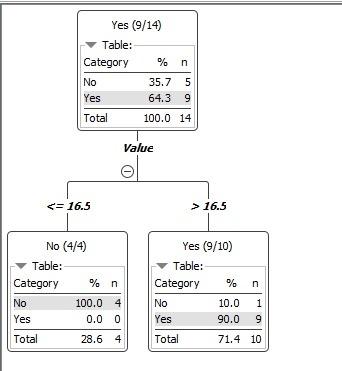
**Input:**

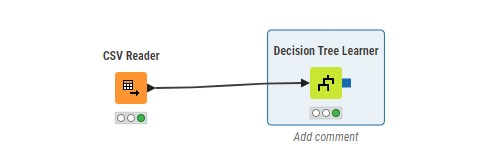
****

**Output:**

****

**Knime output:**

****

****

**Experiment No. 13**

**Program:**

import csv

from collections import defaultdict

def train(dataset):

class\_counts = defaultdict(int)

attribute\_counts = defaultdict(lambda: defaultdict(int))

total\_samples = 0

for row in dataset:

if len(row) < 2: # Ensure the row has at least two elements (attributes and class label)

continue

class\_val = row[-1].strip("'").strip() # last column is the class label

class\_counts[class\_val] += 1

total\_samples += 1

for i, value in enumerate(row[:-1]):

attribute\_counts[i][value.strip("'").strip(), class\_val] += 1

return class\_counts, attribute\_counts, total\_samples

def predict(class\_counts, attribute\_counts, total\_samples, instance):

class\_probs = defaultdict(float)

for class\_val, class\_count in class\_counts.items():

prob = 1.0

for i, value in enumerate(instance):

count = attribute\_counts[i][value.strip("'").strip(), class\_val]

prob \*= (count + 1) / (class\_count + len(attribute\_counts[i]))

class\_probs[class\_val] = prob \* class\_count / total\_samples

prob\_yes = class\_probs['Yes'] / (class\_probs['Yes'] + class\_probs['No'])

prob\_no = class\_probs['No'] / (class\_probs['Yes'] + class\_probs['No'])

return max(class\_probs, key=class\_probs.get), prob\_yes, prob\_no

def read\_csv(filename):

dataset = []

with open(filename, 'r') as file:

csv\_reader = csv.reader(file)

for row in csv\_reader:

dataset.append(row)

return dataset

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

filename = 'exp13\_input.csv'

dataset = read\_csv(filename)

class\_counts, attribute\_counts, total\_samples = train(dataset)

# Take test\_instance as input from the user

test\_instance = input("Enter the test sample attributes (comma-separated values): ").split(',')

test\_instance = [value.strip().strip("'") for value in test\_instance]

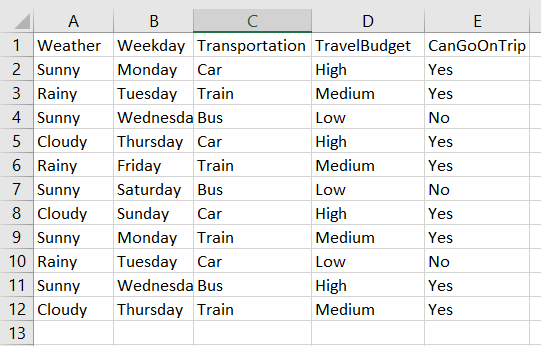
predicted\_class, prob\_yes, prob\_no = predict(class\_counts, attribute\_counts, total\_samples, test\_instance)

print(f"Predicted class: {predicted\_class}")

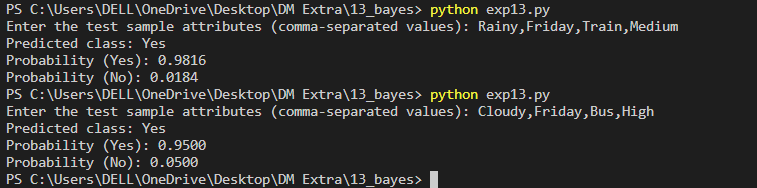
print(f"Probability (Yes): {prob\_yes:.4f}")

print(f"Probability (No): {prob\_no:.4f}")

**Input:**

****

**Output:**

****