Lab Number: 1

Title: Preprocessing of primary and secondary datasets containing dirty data

Objective: To perform data cleaning by handling missing values, duplicates, inconsistencies, and noisy data using Weka.

IDE/Tools used: Weka 3.9.6, Microsoft Excel

Components Used:

Primary Dataset: A dataset created by the student (e.g., records of students from a Nepali college).
Secondary Dataset: A real-world dataset obtained from an external source (e.g., Kaggle, government portal).

Theory:

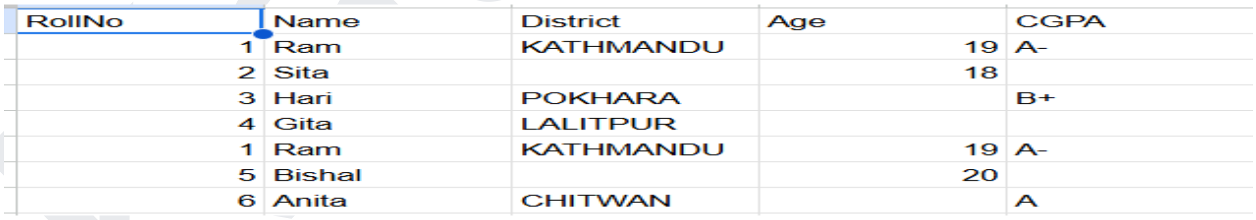
Data preprocessing is the foundational phase of knowledge discovery in databases (KDD). Real-world data collected from multiple sources is frequently incomplete, noisy, and inconsistent. The major tasks are data cleaning, data integration, data transformation, and data reduction.

Implementation Steps:

A) For Primary Dataset:

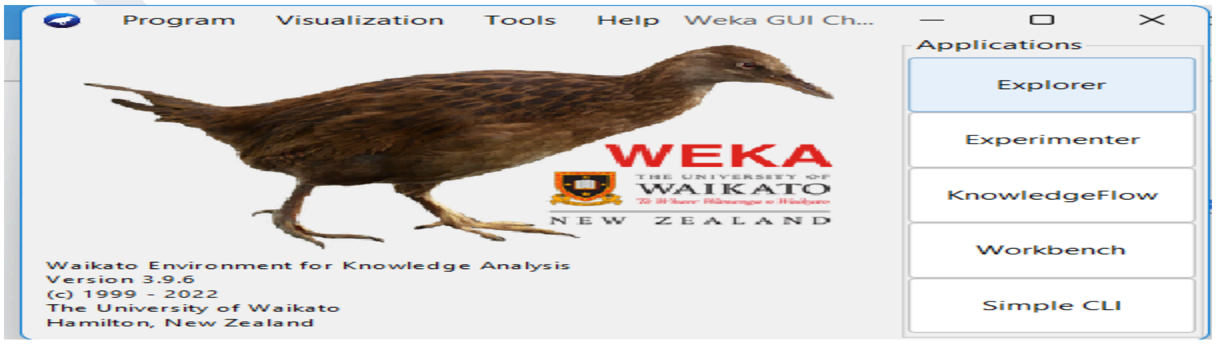
Steps:

1) **Creating a csv file**



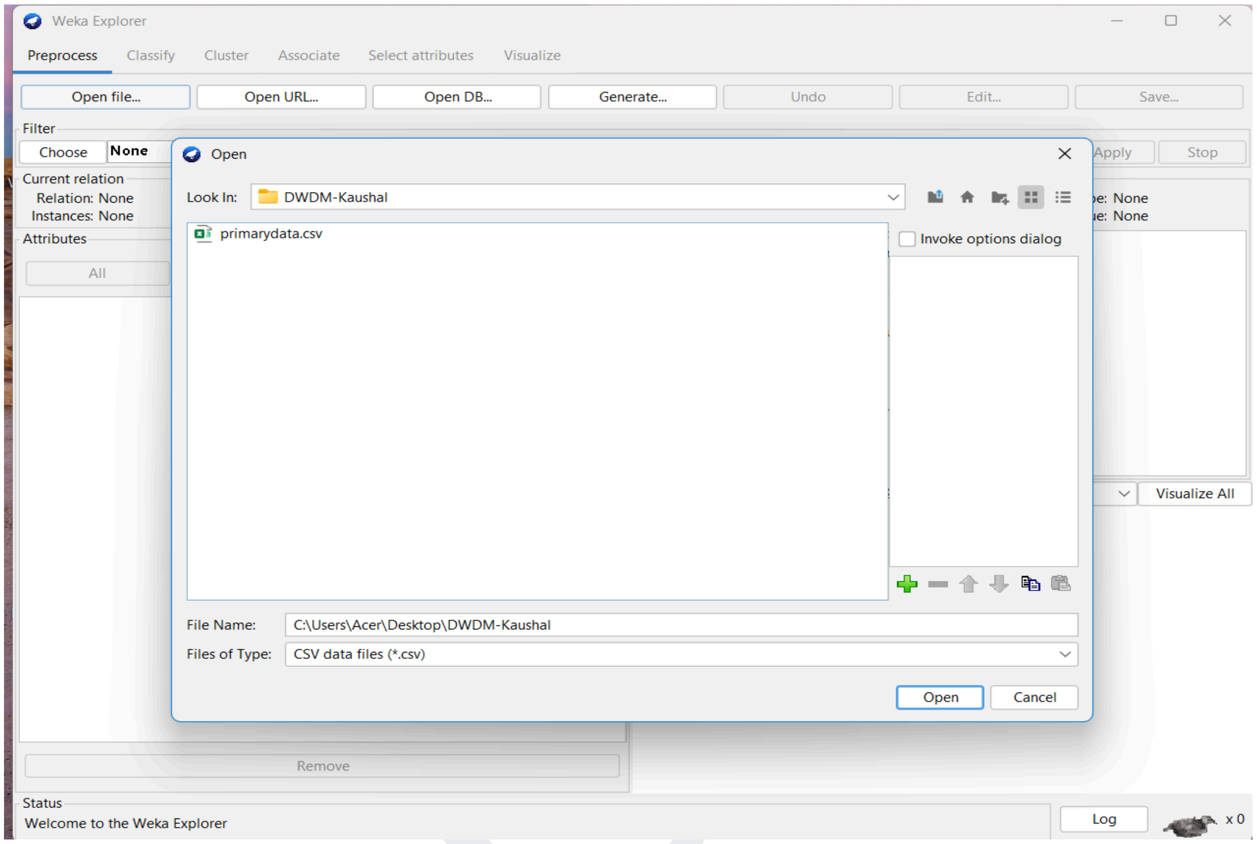
RollNo	Name	District	Age	CGPA
1	Ram	KATHMANDU	19	A-
2	Sita		18	
3	Hari	POKHARA		B+
4	Gita	LALITPUR		
1	Ram	KATHMANDU	19	A-
5	Bishal		20	
6	Anita	CHITWAN		A

2) **Open Weka. Click Explorer. Then click on Open file.**



Waikato Environment for Knowledge Analysis
Version 3.9.6
(c) 1999 - 2022
The University of Waikato
Hamilton, New Zealand

3) Click on file type and choose .csv and select the CSV file.



Filter: Choose None

Current relation
Relation: None
Instances: None
Attributes

Open

Look In: DWDM-Kaushal

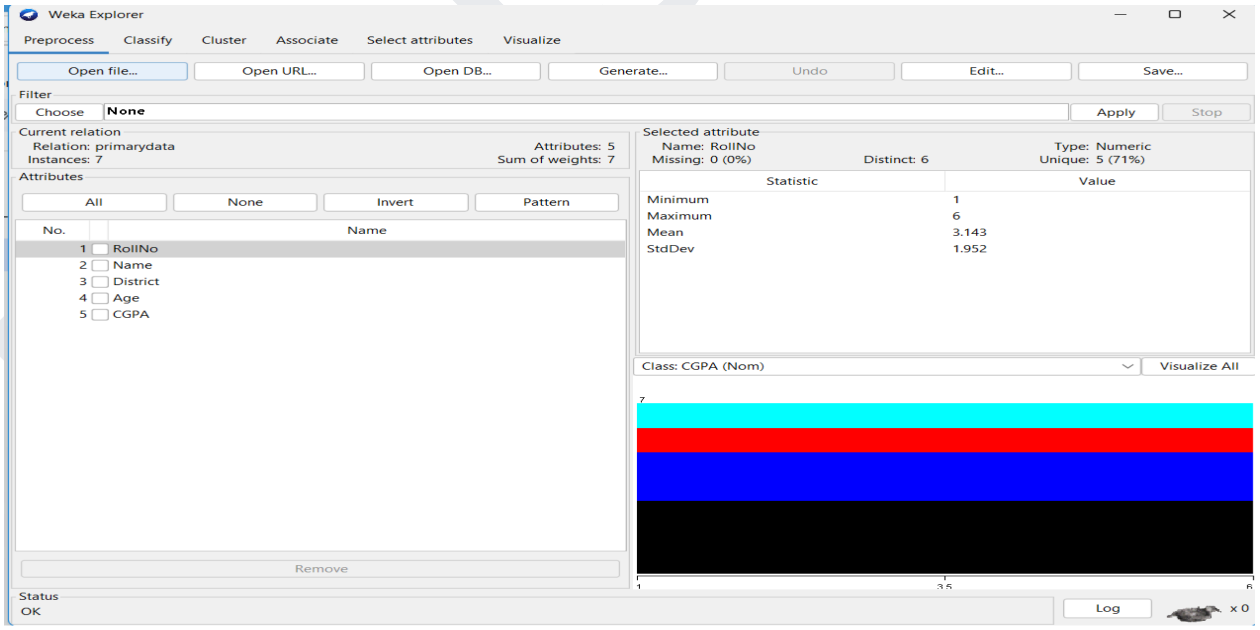
primarydata.csv

Invoke options dialog

File Name: C:\Users\Acer\Desktop\DWDM-Kaushal

Files of Type: CSV data files (*.csv)

Status
Welcome to the Weka Explorer



Filter: Choose None

Current relation
Relation: primarydata　Attributes: 5
Instances: 7　Sum of weights: 7

Attributes

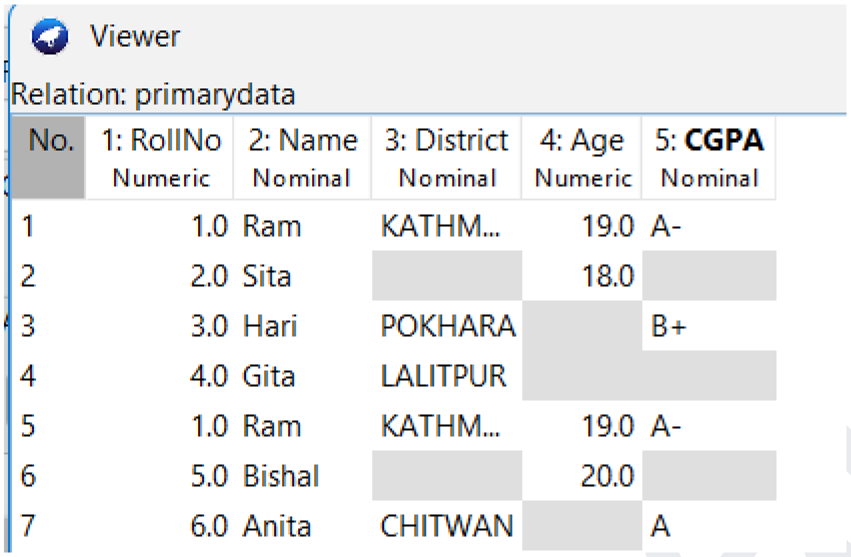
No.	Name
1	RollNo
2	Name
3	District
4	Age
5	CGPA

Selected attribute
Name: RollNo　Type: Numeric
Missing: 0 (0%)　Distinct: 6　Unique: 5 (71%)

Statistic	Value
Minimum	1
Maximum	6
Mean	3.143
StdDev	1.952

Class: CGPA (Nom)

Status
OK

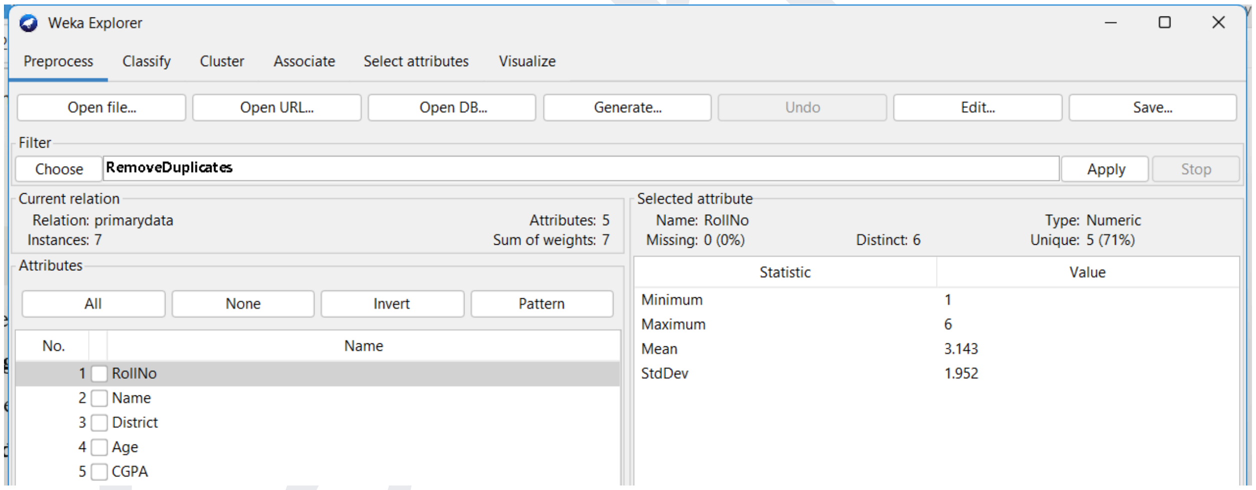


Viewer

Relation: primarydata

No.	1: RollNo Numeric	2: Name Nominal	3: District Nominal	4: Age Numeric	5: CGPA Nominal
1	1.0	Ram	KATHM...	19.0	A-
2	2.0	Sita		18.0	
3	3.0	Hari	POKHARA		B+
4	4.0	Gita	LALITPUR		
5	1.0	Ram	KATHM...	19.0	A-
6	5.0	Bishal		20.0	
7	6.0	Anita	CHITWAN		A

4) Click on Choose button → Select filters → unsupervised → instance → RemoveDuplicates



Filter: RemoveDuplicates

Current relation
Relation: primarydata
Instances: 7
Attributes: 5
Sum of weights: 7

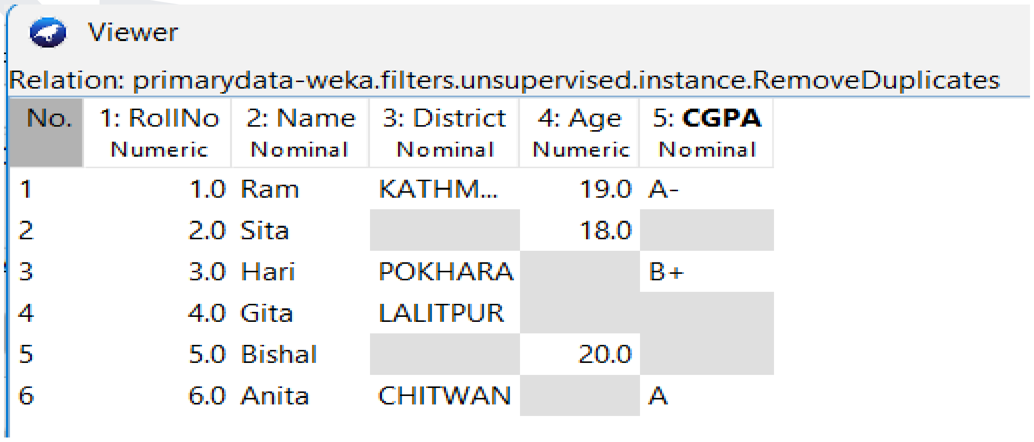
Selected attribute
Name: RollNo
Missing: 0 (0%)
Distinct: 6
Type: Numeric
Unique: 5 (71%)

Statistic	Value
Minimum	1
Maximum	6
Mean	3.143
StdDev	1.952

Attributes

No.	Name
1	RollNo
2	Name
3	District
4	Age
5	CGPA

5) Click on the Apply button. Now you have 6 instances (1 duplicate removed).

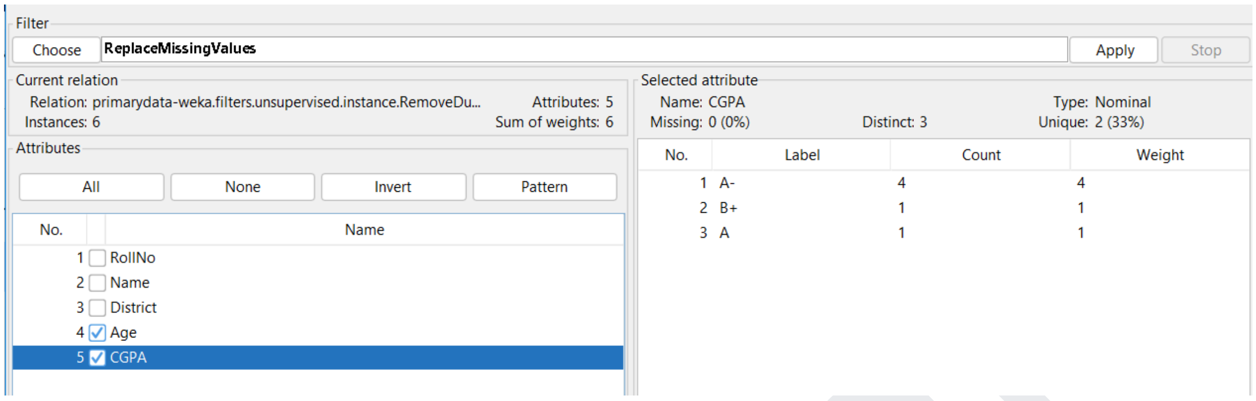


Viewer

Relation: primarydata-weka.filters.unsupervised.instance.RemoveDuplicates

No.	1: RollNo Numeric	2: Name Nominal	3: District Nominal	4: Age Numeric	5: CGPA Nominal
1	1.0	Ram	KATHM...	19.0	A-
2	2.0	Sita		18.0	
3	3.0	Hari	POKHARA		B+
4	4.0	Gita	LALITPUR		
5	5.0	Bishal		20.0	
6	6.0	Anita	CHITWAN		A

6) Click on Choose button again → Select filters → unsupervised → attribute → ReplaceMissingValues



Filter
Choose ReplaceMissingValues

Current relation
Relation: primarydata-weka.filters.unsupervised.instance.RemoveDu... Attributes: 5
Instances: 6 Sum of weights: 6

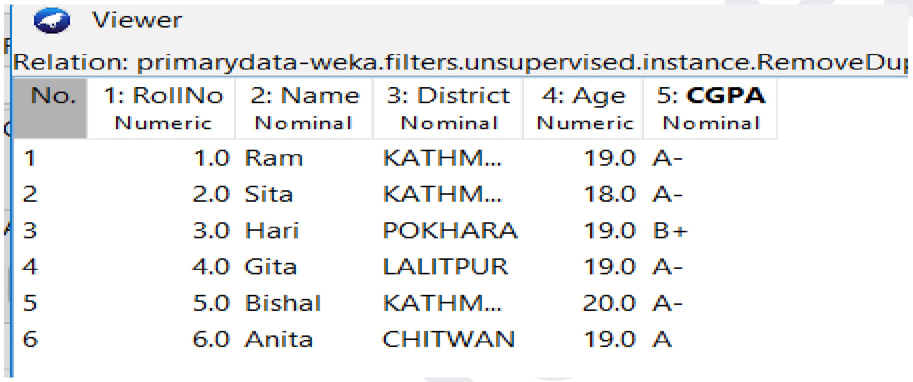
Selected attribute
Name: CGPA Type: Nominal
Missing: 0 (0%) Distinct: 3 Unique: 2 (33%)

No.	Label	Count	Weight
1	A-	4	4
2	B+	1	1
3	A	1	1

Attributes

No.	Name
1	RollNo
2	Name
3	District
4	✓ Age
5	✓ CGPA

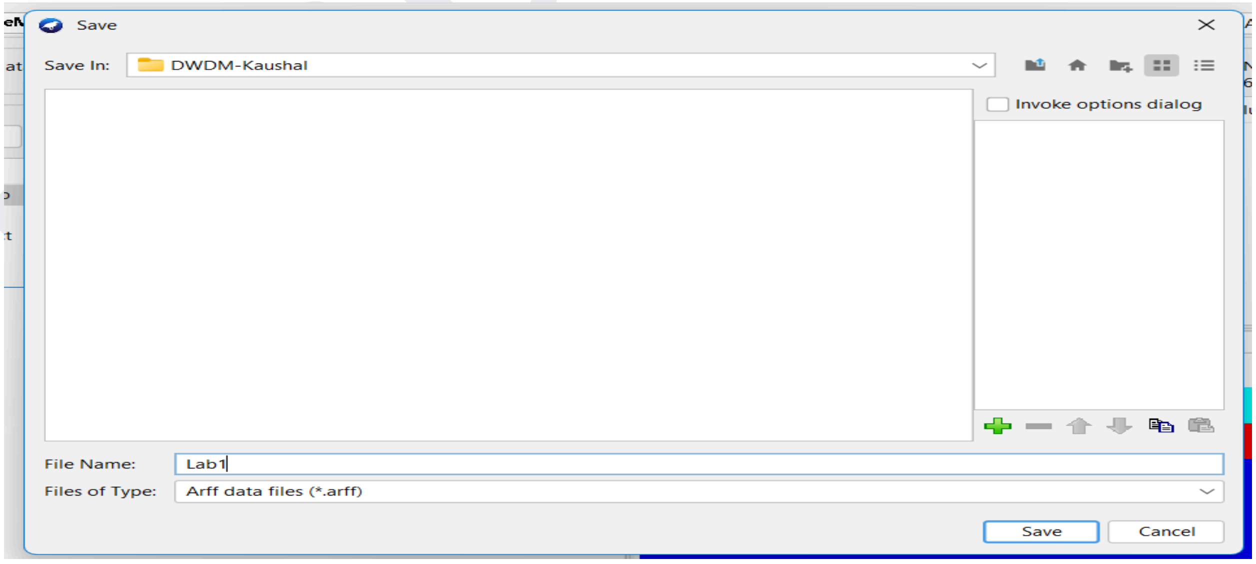
7) Click the Apply button. All missing values are now replaced.



Viewer
Relation: primarydata-weka.filters.unsupervised.instance.RemoveDu

No.	1: RollNo Numeric	2: Name Nominal	3: District Nominal	4: Age Numeric	5: CGPA Nominal
1	1.0	Ram	KATHM...	19.0	A-
2	2.0	Sita	KATHM...	18.0	A-
3	3.0	Hari	POKHARA	19.0	B+
4	4.0	Gita	LALITPUR	19.0	A-
5	5.0	Bishal	KATHM...	20.0	A-
6	6.0	Anita	CHITWAN	19.0	A

8) Save the preprocessed data: Click Save button and choose location to save cleaned dataset.



Save
Save In: DWDM-Kaushal
Invoke options dialog
File Name: Lab1
Files of Type: Arff data files (*.arff)
Save Cancel

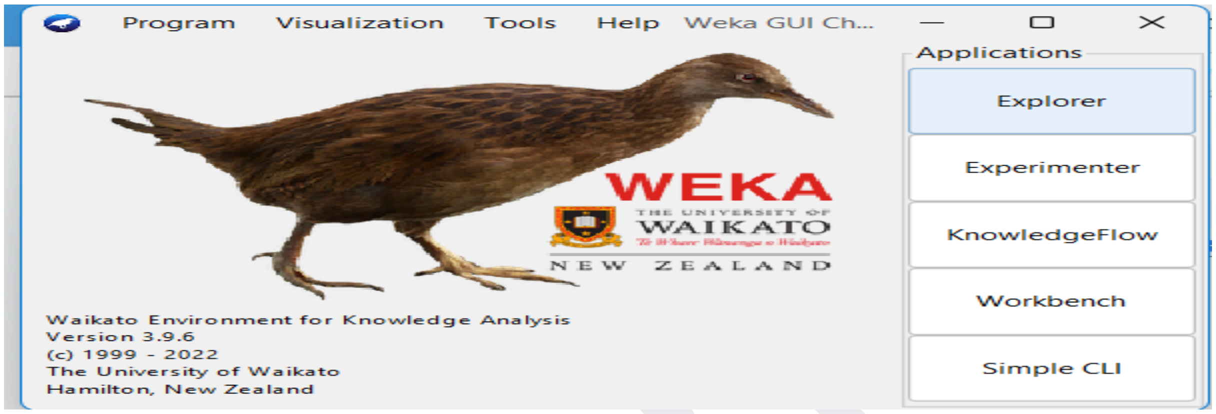
Done by- Kaushal

B) Secondary Dataset
Steps:

1) Download the dataset from Kaggle and copy to the folder.

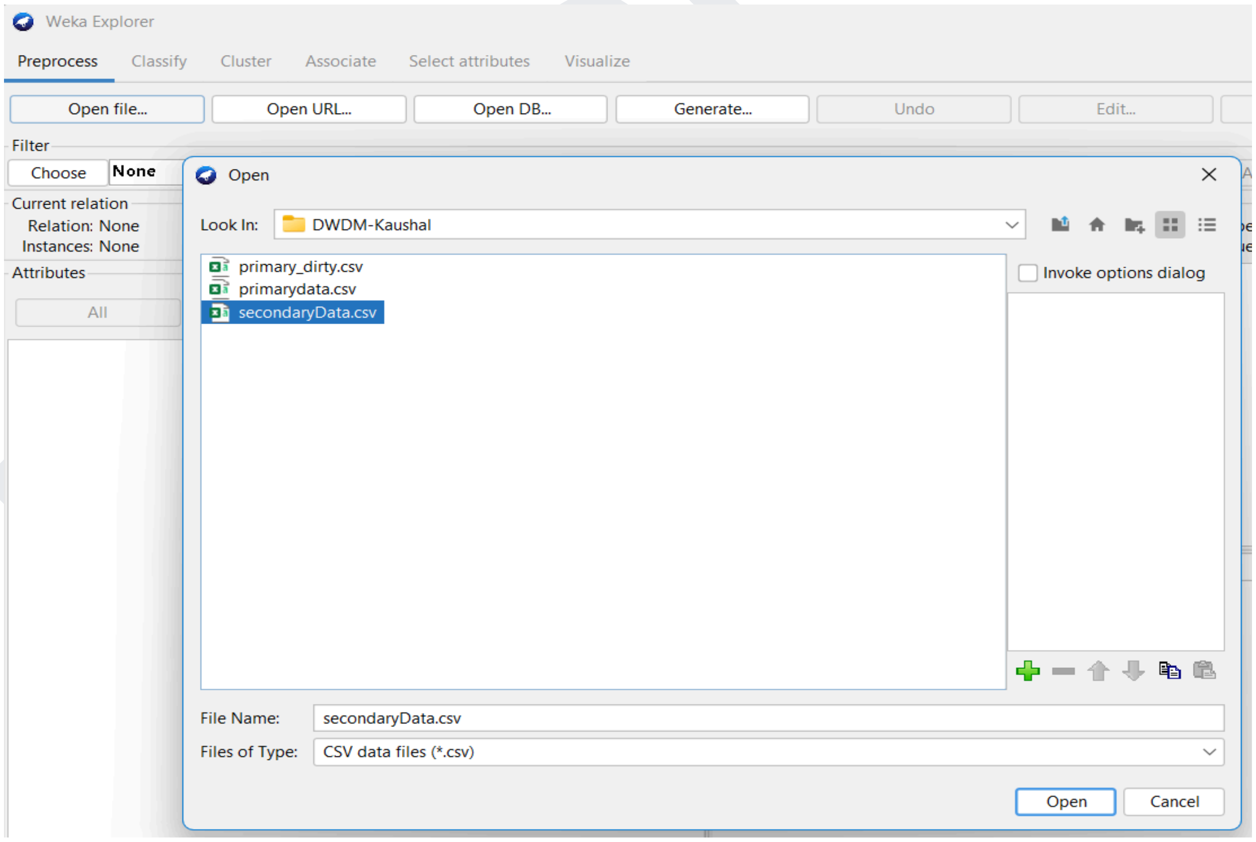
   https://www.kaggle.com/datasets/iqmansingh/data-preprocessing-dataset

2) Open Weka. Click Explorer. Then click on the Open file.



Waikato Environment for Knowledge Analysis
Version 3.9.6
(c) 1999 - 2022
The University of Waikato
Hamilton, New Zealand

3) Click on file type and choose .csv and select the CSV file.



Relation: None
Instances: None

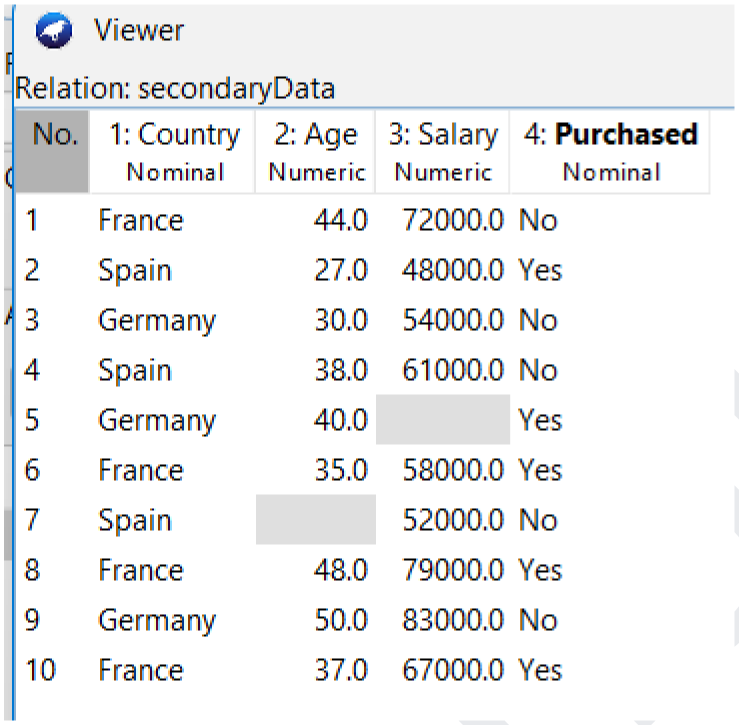
Look In: DWDM-Kaushal

primary_dirty.csv
primarydata.csv
secondaryData.csv

File Name: secondaryData.csv
Files of Type: CSV data files (*.csv)

4) Click on the Edit tab and open the viewer to see where the data are missing.

Done by- Kaushal

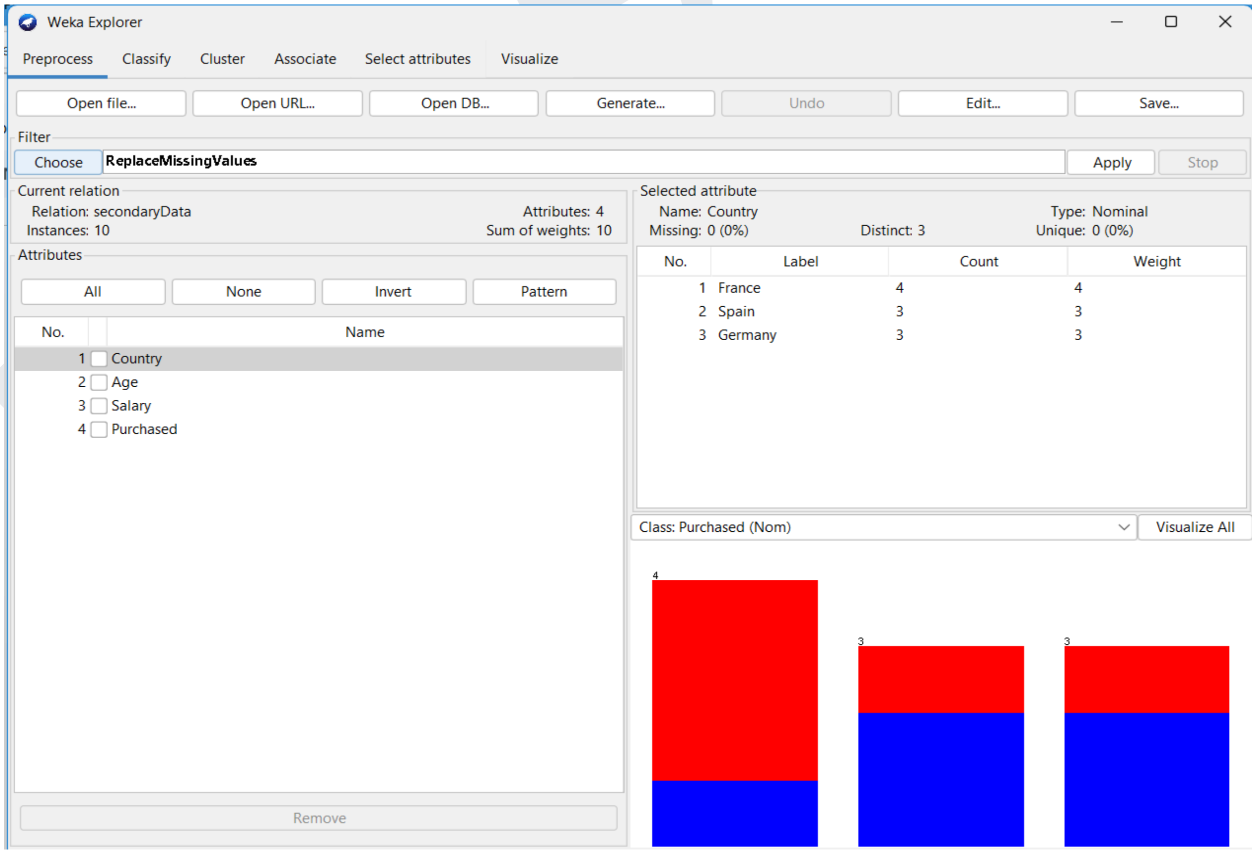


Viewer

Relation: secondaryData

No.	1: Country Nominal	2: Age Numeric	3: Salary Numeric	4: **Purchased** Nominal
1	France	44.0	72000.0	No
2	Spain	27.0	48000.0	Yes
3	Germany	30.0	54000.0	No
4	Spain	38.0	61000.0	No
5	Germany	40.0		Yes
6	France	35.0	58000.0	Yes
7	Spain		52000.0	No
8	France	48.0	79000.0	Yes
9	Germany	50.0	83000.0	No
10	France	37.0	67000.0	Yes

5) Since there are only missing values click on the Filter → Unsupervised→ Attribute → ReplaceMissingValues and click Apply.



Filter: ReplaceMissingValues

Current relation
Relation: secondaryData   Attributes: 4
Instances: 10   Sum of weights: 10

Selected attribute
Name: Country   Type: Nominal
Missing: 0 (0%)   Distinct: 3   Unique: 0 (0%)

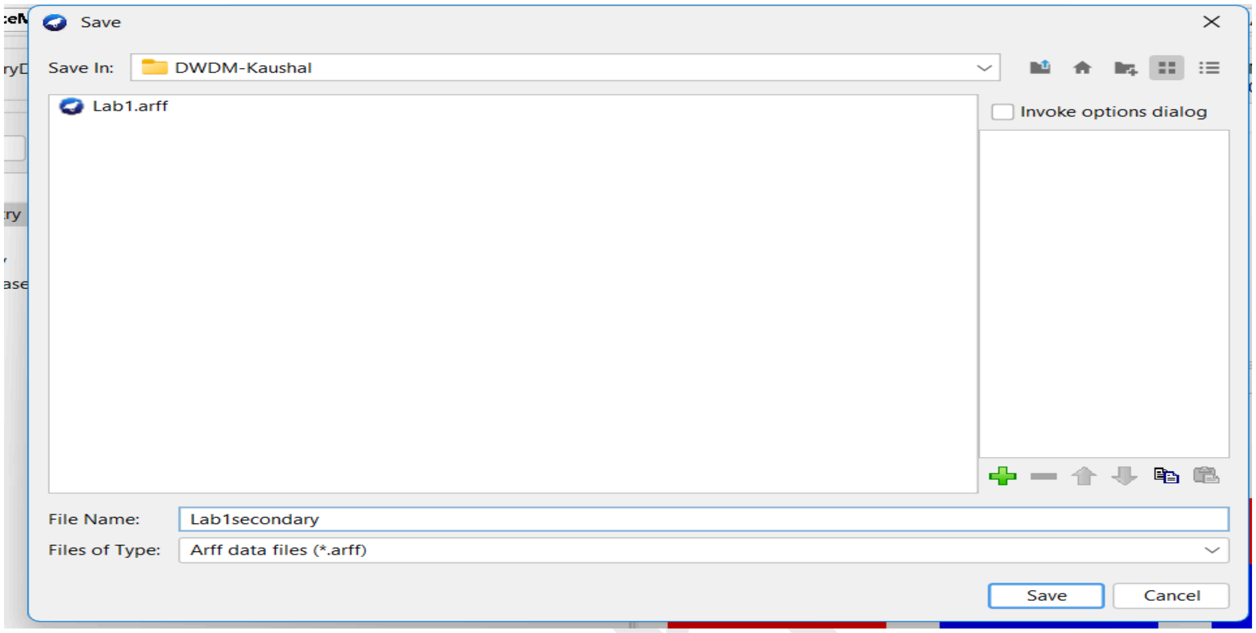
No.	Name
1	Country
2	Age
3	Salary
4	Purchased

No.	Label	Count	Weight
1	France	4	4
2	Spain	3	3
3	Germany	3	3

Class: Purchased (Nom)

Done by- Kaushal

6) After ensuring there are no missing values left, we save the file in .arff type.


Save

Save In: DWDM-Kaushal

Lab1.arff

Invoke options dialog

File Name: Lab1secondary

Files of Type: Arff data files (*.arff)

Discussion:

Both datasets contained several data quality issues including missing values and duplicate records. The primary dataset had missing entries in District, Age, and CGPA columns, while the secondary dataset showed gaps in Province, Marks, and Grade attributes.The ReplaceMissingValues filter was applied to handle missing data. This filter replaces missing numeric values with the column mean and missing nominal values with the mode. This approach maintains the overall data distribution without introducing significant bias. For example, filling missing Age values with the average age keeps the dataset representative of the actual population.Duplicate records were removed using the RemoveDuplicates filter. Duplicates can skew analysis results by giving extra weight to repeated entries, which affects the accuracy of any models trained on the data. Removing them ensures each observation appears only once.After applying these preprocessing steps, both datasets become complete and consistent, ready for further analysis or integration.

Conclusion:

Preprocessing transformed the raw datasets into clean, usable data suitable for data mining tasks. Handling missing values and removing duplicates addressed the main quality issues that could compromise analysis accuracy. The datasets now meet the requirements for machine learning algorithms like Decision Trees and Naïve Bayes, which need complete and consistent data to function properly. This demonstrates that preprocessing is essential for obtaining reliable results in any data related projects, as poor data quality directly leads to poor model performance.

Lab Number: 2

Title: Designing various Data Warehouse Schema (Star Schema, Snowflake Schema, Fact Constellation) using MySQL query.

Objective: To get familiarized with the schemas used for creating a data warehouse.

IDE/Tools used: MySQL, Apache server.

Query Programming language: SQL

Theory:

Data Warehouse is a centralized repository that contains vast amounts of data from numerous sources. It is intended for query and analysis rather than transaction processing.

In data warehousing, **schema** refers to the logical and physical framework that organizes data in a data warehouse and optimizes it for analytical queries.

The main application of a data warehouse is to assist business intelligence operations such as reporting, data mining, and analytics, allowing firms to make data-driven choices.

The three types of schemas used in data warehousing are Star Schema, Snowflake Schema, and Fact Constellation Schema.

Star Schema: It is a simple, denormalized schema consisting of a primary fact table containing quantitative data and many dimension tables describing the measurements. It is efficient for querying because there are fewer joins.

Snowflake Schema: It is a refined version of the Star Schema in which dimension tables are divided into sub-dimensions, minimizing data redundancy while enhancing query complexity from additional joins.

Fact Constellation Schema: It is a collection of various fact tables that utilize common dimension tables, creating a sophisticated framework ideal for advanced analytical needs across related data domains.

Implementation:

Q1.) Star Schema

Queries # Create a database

```
CREATE DATABASE kaushaldatawarehouse;
```

# Create a source table for data warehouse.

```
CREATE TABLE company (id int AUTO_INCREMENT PRIMARY KEY,
        item_name varchar(255),
        brand varchar(255),
        sold_by varchar(255),
        category varchar(255),
        day int,
        month varchar(255),
        quarter varchar(255),
        years int,
        location_name varchar(255),
        state varchar(255),
        pin_code int,
        branch_name varchar(255),
        branch_manager varchar(255),
        qty_sold int,
        amt_sold int
);
```

```
INSERT INTO company(item_name,
        brand,
        sold_by,
        category,
        day,
        month,
        quarter,
        years,
        location_name,
        state,
        pin_code,
        branch_name,
        branch_manager,
        qty_sold,
        amt_sold)
VALUES("Car","Model X","Tesla","Four wheeler",13,"June","Q2",2021,"New
Baneshwor","Bagmati","123","Baneshwor 1","Ashish",2,15000),

("Car","Model Y","Tesla","Four wheeler",15,"October","Q4",2022,"Old
Baneshwor","Bagmati","123","Baneshwor 3","Manoj",1,5000);

SELECT * FROM company;
```

# Create time dimension table for data warehouse.

```
CREATE TABLE timedim (t_id int AUTO_INCREMENT PRIMARY KEY,
        day int,
        month varchar(255),
        quarter varchar(255),
        years int
);
INSERT INTO timedim(
        day,
        month,
        quarter,
        years
)
SELECT day,month,quarter,years FROM company;

SELECT * FROM timedim;

# Create item dimension table for data warehouse.
CREATE TABLE itemdim (i_id int AUTO_INCREMENT PRIMARY KEY,
        item_name varchar(255),
        brand varchar(255),
        sold_by varchar(255),
        category varchar(255)
);
INSERT INTO itemdim(
        item_name,
        brand,
        sold_by,
        category
)
SELECT item_name,brand,sold_by,category FROM company;
SELECT * FROM itemdim;
```

# Create location dimension table for data warehouse.

```
CREATE TABLE locationdim (l_id int AUTO_INCREMENT PRIMARY KEY,
        location_name varchar(255),
        state varchar(255),
        pin_code int
);
INSERT INTO locationdim(
        location_name,
        state,
        pin_code
)
SELECT location_name,state,pin_code FROM company;
SELECT * FROM locationdim;
```

# Create branch dimension table for data warehouse.

```
CREATE TABLE branchdim (b_id int AUTO_INCREMENT PRIMARY KEY,
        branch_name varchar(255),
        branch_manager varchar(255)
);

INSERT INTO branchdim(
        branch_name,
        branch_manager
)
SELECT branch_name,branch_manager FROM company;
SELECT * FROM branchdim;
```

# Create sales fact table for data warehouse using Foreign Key.

```
CREATE TABLE salesFact (t_id int,
        i_id int,
        l_id int,
        b_id int,
        qty_sold int,
        amt_sold int,
        FOREIGN Key(t_id) REFERENCES timedim(t_id),
        FOREIGN Key(i_id) REFERENCES itemdim(i_id),
        FOREIGN Key(l_id) REFERENCES locationdim(l_id),
        FOREIGN Key(b_id) REFERENCES branchdim(b_id));
        INSERT INTO salesFact(t_id,
        i_id,
        l_id,
        b_id,
        qty_sold,
        amt_sold)

SELECT t_id,i_id,l_id,b_id,qty_sold,amt_sold FROM company c
        LEFT OUTER JOIN timedim t ON t.day = c.day AND t.month = c.month AND t.quarter
= c.quarter AND t.years = c.years
LEFT OUTER JOIN itemdim i ON i.item_name = c.item_name AND i.brand = c.brand AND
i.sold_by = c.sold_by AND i.category = c.category
LEFT OUTER JOIN locationdim l ON l.location_name = c.location_name AND l.state =
c.state AND l.pin_code = c.pin_code
LEFT OUTER JOIN branchdim b ON b.branch_name = c.branch_name AND
b.branch_manager = c.branch_manager;
SELECT * FROM salesFact;
```

# Query to select the records where years = 2022

```
SELECT * FROM salesFact s LEFT OUTER JOIN timedim t ON t.t_id = s.t_id WHERE
years=2022;
```

Q2.) Snowflake Schema Queries

# Create a database named "snowcompany"

```
CREATE DATABASE snowcompany;
```

# Create a source table for data warehouse.

```
CREATE TABLE company (id int AUTO_INCREMENT PRIMARY KEY,
        item_name varchar(255),
        brand varchar(255),
        sold_by varchar(255),
        category varchar(255),
        day int,
        month varchar(255),
        quarter varchar(255),
        years int,
        location_name varchar(255),
        state varchar(255),
        pin_code int,
        branch_name varchar(255),
        branch_manager varchar(255),
        department_name varchar(255),
        department_code int,
        supplier_name varchar(255),
        supplier_address varchar(255),
        supplier_type varchar(255),
	qty_sold int,

	amt_sold int

);

INSERT INTO company(item_name,

	brand,

	sold_by,

	category,

	day,

	month,

	quarter,

	years,

	location_name,

	state,

	pin_code,

	branch_name,

	branch_manager,

	department_name,

	department_code,

	supplier_name,

	supplier_address,

	supplier_type,

	qty_sold,

	amt_sold)
```

```
VALUES("Car","Model X","Tesla","Four wheeler",13,"June","Q2",2021,"New
Baneshwor","Bagmati","123","Baneshwor 1","Ashish","sales",013,"C&C
Auto","Koteshwor","Auto Four Wheeler",2,15000),

("Car","Model Y","Tesla","Four wheeler",15,"October","Q4",2022,"Old
Baneshwor","Bagmati","123","Baneshwor 3","Manoj","finance",420,"T&T
Auto","Tripureshwor","Auto",1,5000);

SELECT * FROM company;
```

# Create time dimension table for data warehouse.

```
CREATE TABLE timedim (t_id int AUTO_INCREMENT PRIMARY KEY,
        day int,
        month varchar(255),
        quarter varchar(255),
        years int
);
INSERT INTO timedim(
        day,
        month,
        quarter,
        years
)
```

```
SELECT day,month,quarter,years FROM company;

SELECT * FROM timedim;
```

# Create location dimension table for data warehouse.

```
CREATE TABLE locationdim (l_id int AUTO_INCREMENT PRIMARY KEY,

        location_name varchar(255),

        state varchar(255),

        pin_code int

);

INSERT INTO locationdim(

        location_name,

        state,

        pin_code

)

SELECT location_name,state,pin_code FROM company;

SELECT * FROM locationdim;
```

# Create department dimension table for data warehouse.

```
CREATE TABLE departmentdim(dept_id int AUTO_INCREMENT PRIMARY KEY,

        dept_name varchar(255),

        dept_code int);

INSERT INTO departmentdim(dept_name,dept_code)

SELECT department_name,department_code FROM company;

SELECT * FROM departmentdim;
```

# Create branch dimension table for data warehouse.

```
CREATE TABLE branchdim (b_id int AUTO_INCREMENT PRIMARY KEY,
        branch_name varchar(255),
        branch_manager varchar(255),
        depart_id int,
FOREIGN KEY(depart_id) REFERENCES departmentdim(dept_id)
);
INSERT INTO branchdim(
        branch_name,
        branch_manager,
        depart_id
)
SELECT branch_name,branch_manager,dept_id FROM company c
        JOIN departmentdim d ON c.department_name = d.dept_name AND c.department_code
= d.dept_code;
SELECT * FROM branchdim;
```

# Create Supplier dimension table for data warehouse.

```
CREATE TABLE supplierdim(supp_id int AUTO_INCREMENT PRIMARY KEY,
        supp_name varchar(255),
        supp_address varchar(255),
        supp_type varchar(255));
INSERT INTO supplierdim(supp_name,supp_address,supp_type)
SELECT supplier_name,supplier_address,supplier_type FROM company;
SELECT * FROM supplierdim;
```

# Create item dimension table for data warehouse.

```
CREATE TABLE itemdim (i_id int AUTO_INCREMENT PRIMARY KEY,
        item_name varchar(255),
        brand varchar(255),
        sold_by varchar(255),
        category varchar(255),
        supplier_id int,
FOREIGN KEY(supplier_id) REFERENCES supplierdim(supp_id)
);
INSERT INTO itemdim(
        item_name,
        brand,
        sold_by,
        category,
        supplier_id
)
SELECT item_name,brand,sold_by,category,supp_id FROM company c
        LEFT OUTER JOIN supplierdim s ON c.supplier_name = s.supp_name AND
c.supplier_address = s.supp_address AND
c.supplier_type = s.supp_type;
SELECT * FROM itemdim;
```

# Create sales fact table for data warehouse using Foreign Key.

```
CREATE TABLE salesFact (t_id int,
        i_id int,
        l_id int,
        b_id int,
        qty_sold int,
        amt_sold int,
        FOREIGN Key(t_id) REFERENCES timedim(t_id),
        FOREIGN Key(i_id) REFERENCES itemdim(i_id),
        FOREIGN Key(l_id) REFERENCES locationdim(l_id),
        FOREIGN Key(b_id) REFERENCES branchdim(b_id));
INSERT INTO salesFact(t_id,
        i_id,
        l_id,
        b_id,
        qty_sold,
        amt_sold)
SELECT t_id,i_id,l_id,b_id,qty_sold,amt_sold FROM company c
        LEFT OUTER JOIN timedim t ON t.day = c.day AND t.month = c.month AND t.quarter
= c.quarter AND t.years = c.years
LEFT OUTER JOIN itemdim i ON i.item_name = c.item_name AND i.brand = c.brand AND
i.sold_by = c.sold_by AND i.category = c.category
LEFT OUTER JOIN locationdim l ON l.location_name = c.location_name AND l.state =
c.state AND l.pin_code = c.pin_code
```

```
LEFT OUTER JOIN branchdim b ON b.branch_name = c.branch_name AND
b.branch_manager = c.branch_manager;
SELECT * FROM salesFact;
```

# Query to show the records of sales fact table with the department dimension table

```
SELECT * FROM salesFact s LEFT OUTER JOIN branchdim b ON b.b_id = s.b_id
        LEFT OUTER JOIN departmentdim d ON b.depart_id = d.dept_id;
```

Q.3) Fact Constellation Queries

# Creating a database named "constellation_db"

```
CREATE DATABASE constellation_db;
```

# Create base table

```
        CREATE TABLE autoxyz (
          id INT AUTO_INCREMENT PRIMARY KEY,
          item_name VARCHAR(255),
          brand VARCHAR(255),
          sold_by VARCHAR(255),
          category VARCHAR(255),
          day INT,
          month VARCHAR(255),
          quarter VARCHAR(255),
          year INT,
          location_name VARCHAR(255),
          state VARCHAR(255),
          pin_code INT,
          driver_name VARCHAR(255),
          duty_time VARCHAR(255),
          qty_sold INT,
          amt_sold INT
  );
```

# Insert sample data

```
INSERT INTO autoxyz (
       item_name,
       brand,
       sold_by,
       category,
       day,
       month,
       quarter,
       year,
       location_name,
       state,
       pin_code,
       driver_name,
       duty_time,
       qty_sold,
       amt_sold
) VALUES
('Car', 'Model X', 'Tesla', 'Four wheeler', 13, 'June', 'Q2', 2024, 'New Baneshwor', 'Bagmati', 123,
'Jack', 'Evening', 2, 13000),
('Car', 'Model Y', 'Tesla', 'Four wheeler', 21, 'November', 'Q4', 2024, 'Koteshwor', 'Bagmati', 123,
'Candace', 'Morning', 1, 8000);
```

# Create dimension tables

```
CREATE TABLE timedim (
t_id INT AUTO_INCREMENT PRIMARY KEY,
        day INT,
        month VARCHAR(255),
        quarter VARCHAR(255),
        year INT
);
INSERT INTO timedim (day, month, quarter, year)
SELECT DISTINCT day, month, quarter, year FROM autoxyz;

CREATE TABLE locationdim (
        l_id INT AUTO_INCREMENT PRIMARY KEY,
        location_name VARCHAR(255),
        state VARCHAR(255),
        pin_code INT
);
```

```
INSERT INTO locationdim (location_name, state, pin_code)

SELECT DISTINCT location_name, state, pin_code FROM autoxyz;


CREATE TABLE itemdim (

i_id INT AUTO_INCREMENT PRIMARY KEY,

        item_name VARCHAR(255),

        brand VARCHAR(255),

        sold_by VARCHAR(255),

        category VARCHAR(255)

);


INSERT INTO itemdim (item_name, brand, sold_by, category)

SELECT DISTINCT item_name, brand, sold_by, category FROM autoxyz;


CREATE TABLE vehicledim (

vehicle_id INT AUTO_INCREMENT PRIMARY KEY,

        driver_name VARCHAR(255),

        duty_time VARCHAR(255)

);


INSERT INTO vehicledim (driver_name, duty_time)

SELECT DISTINCT driver_name, duty_time FROM autoxyz;
```

# Create fact tables

```
CREATE TABLE salesfact (
        item_id INT,
        time_id INT,
        location_id INT,
        qty_sold INT,
        FOREIGN KEY (item_id) REFERENCES itemdim(i_id),
        FOREIGN KEY (time_id) REFERENCES timedim(t_id),
        FOREIGN KEY (location_id) REFERENCES locationdim(l_id)
);
INSERT INTO salesfact (item_id, time_id, location_id, qty_sold)
SELECT
        i.i_id,
        t.t_id,
        l.l_id,
        c.qty_sold
FROM autoxyz c
LEFT JOIN timedim t ON t.day = c.day AND t.month = c.month AND t.quarter = c.quarter
AND t.year = c.year
LEFT JOIN itemdim i ON i.item_name = c.item_name AND i.brand = c.brand AND i.sold_by =
c.sold_by AND i.category = c.category
LEFT JOIN locationdim l ON l.location_name = c.location_name AND l.state = c.state AND
l.pin_code = c.pin_code;
```

```
CREATE TABLE deliveryfact (
        item_id INT,
        location_id INT,
        vehicle_id INT,
        FOREIGN KEY (item_id) REFERENCES itemdim(i_id),
        FOREIGN KEY (location_id) REFERENCES locationdim(l_id),
        FOREIGN KEY (vehicle_id) REFERENCES vehicledim(vehicle_id)
);
INSERT INTO deliveryfact (item_id, location_id, vehicle_id)
SELECT
        i.i_id,
        l.l_id,
        v.vehicle_id
FROM autoxyz c
LEFT JOIN itemdim i ON i.item_name = c.item_name AND i.brand = c.brand AND i.sold_by =
c.sold_by AND i.category = c.category
LEFT JOIN locationdim l ON l.location_name = c.location_name AND l.state = c.state AND
l.pin_code = c.pin_code
LEFT JOIN vehicledim v ON v.driver_name = c.driver_name AND v.duty_time = c.duty_time;
```

# Query from deliveryfact where duty time is Morning

```
SELECT
```

```
        d.item_id,
        d.location_id,
        v.vehicle_id,
        v.driver_name,
        v.duty_time
FROM deliveryfact d
LEFT JOIN vehicledim v ON d.vehicle_id = v.vehicle_id
WHERE v.duty_time = 'Morning';
```

Screenshots of generated Data warehouse schema:

Figure 1: Star Schema from using SQL

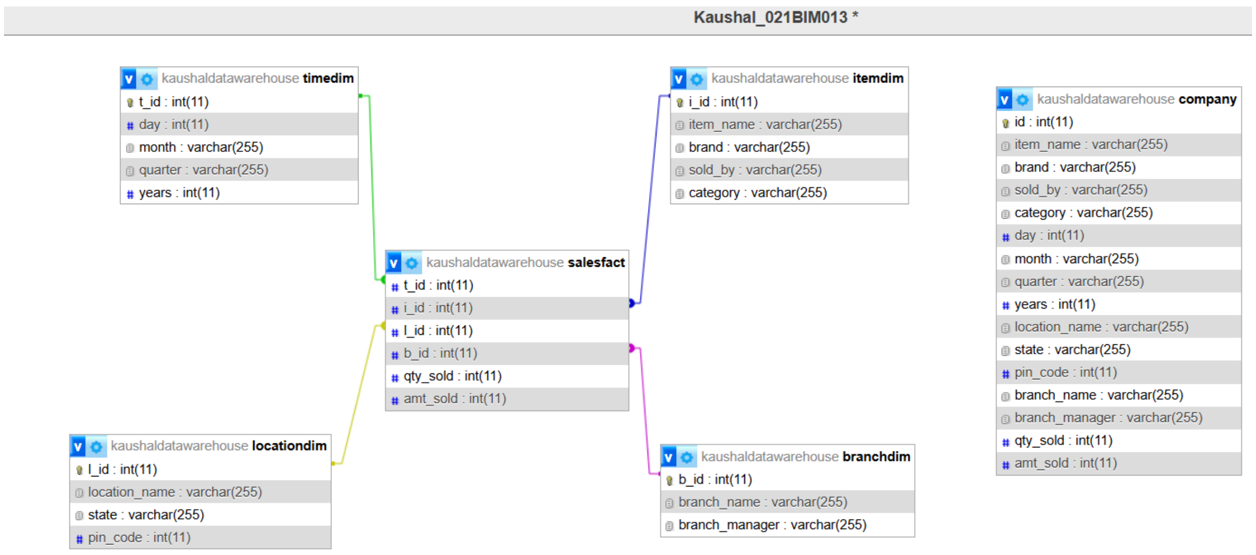
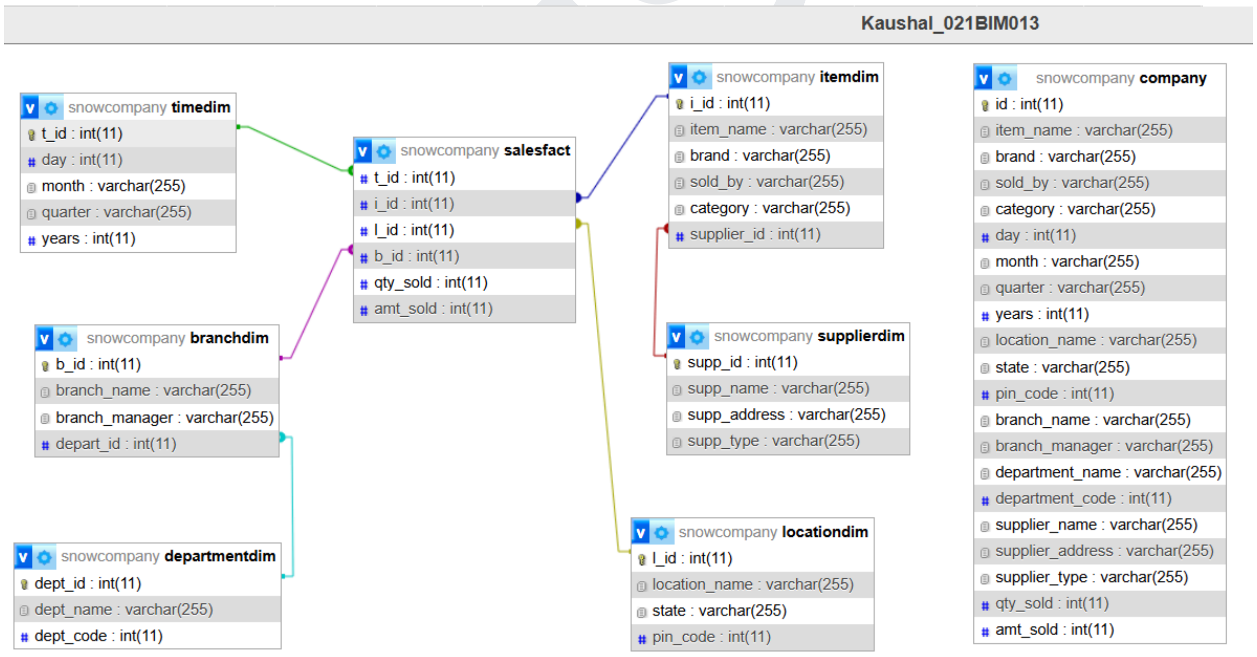
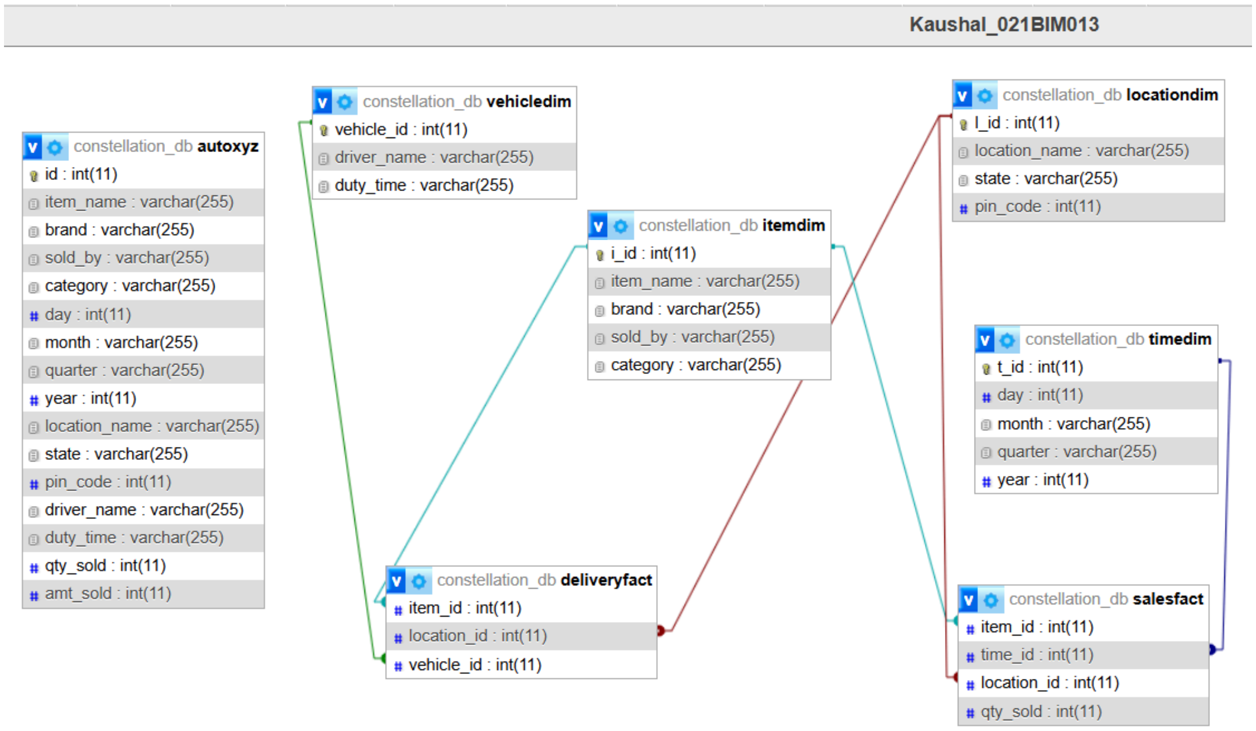


Figure 2: Snowflake Schema from using SQL



Done by- Kaushal

Figure 3: Galaxy Schema



Done by- Kaushal

Discussion:

These schemas differ in terms of space and time complexity in query response, which is achieved through normalization and de-normalization found in the schema. The star schema being highly simple has higher de-normalization and the fact constellation being the complex has a higher response time as it has higher normalization across subject domain.

Conclusion:

In this lab, we got familiarized with various data warehouse schema through their implementations using SQL query programming language in MySQL platform locally hosted by Apache server, also, it was found that different designs had different their own specialty which are utilized based on the requirements of the application domain and available resources.

Lab Number: 3

Title: Prepare a numeric type primary dataset and perform discretization on the dataset.

Objective: To perform discretization on a numeric dataset by converting continuous attributes into categorical intervals using Weka.

IDE/Tools used: Weka 3.9.6, Microsoft Excel

Components Used:
Primary Dataset: A numeric dataset containing student performance metrics like marks in (Ecommerce, DWDM, OperationsManagement) and (AttendancePercent, StudyHours)

Theory:

Discretization is a data transformation technique that converts continuous numeric attributes into discrete categorical values by dividing the range into intervals or bins. This process is essential in data mining for several reasons:

Importance:

- Many machine learning algorithms work better with categorical data
- Reduces computational complexity and improves processing speed
- Makes patterns more interpretable and easier to understand
- Helps handle outliers and noise in continuous data

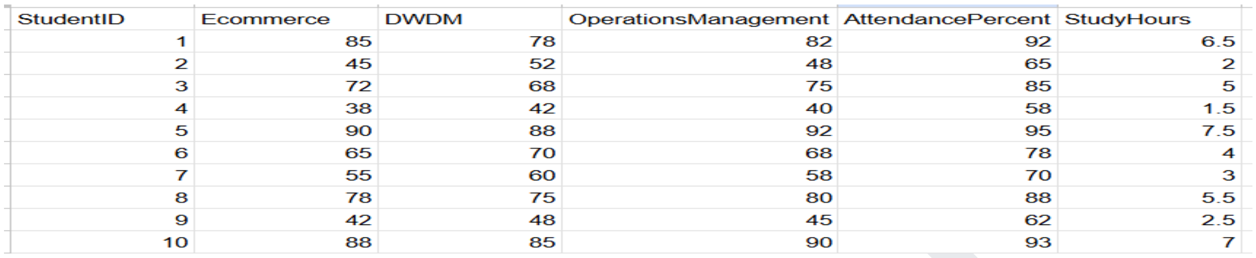
Types of Discretization:

1. **Equal Width Discretization:** Divides the range into intervals of equal size
2. **Equal Frequency Discretization:** Creates bins with approximately equal number of instances
3. **Supervised Discretization:** Uses class information to create meaningful boundaries

For example, converting student marks (0-100) into grades (Fail, Pass, Good, Excellent) makes the data easier to interpret and can improve classification performance

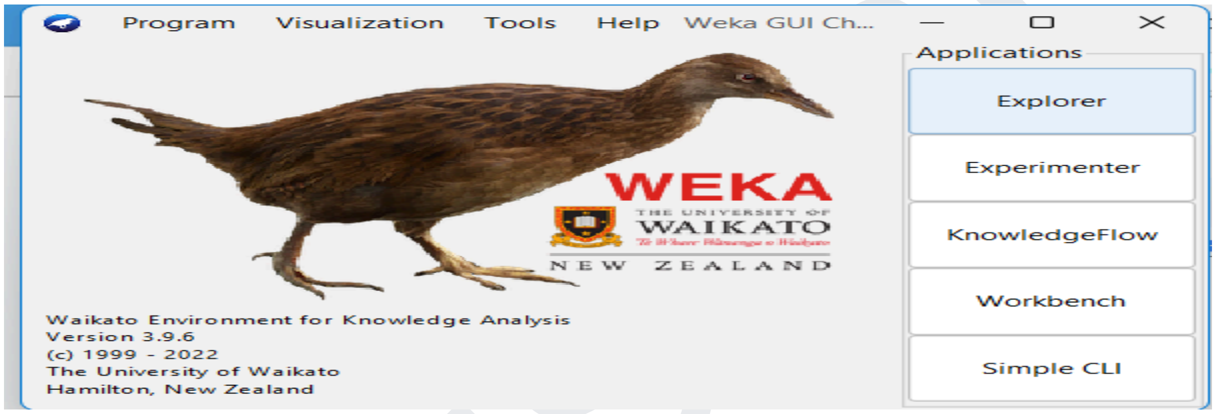
Implementation:
Steps:

1) Save the dataset as a CSV file (e.g., lab3.csv)



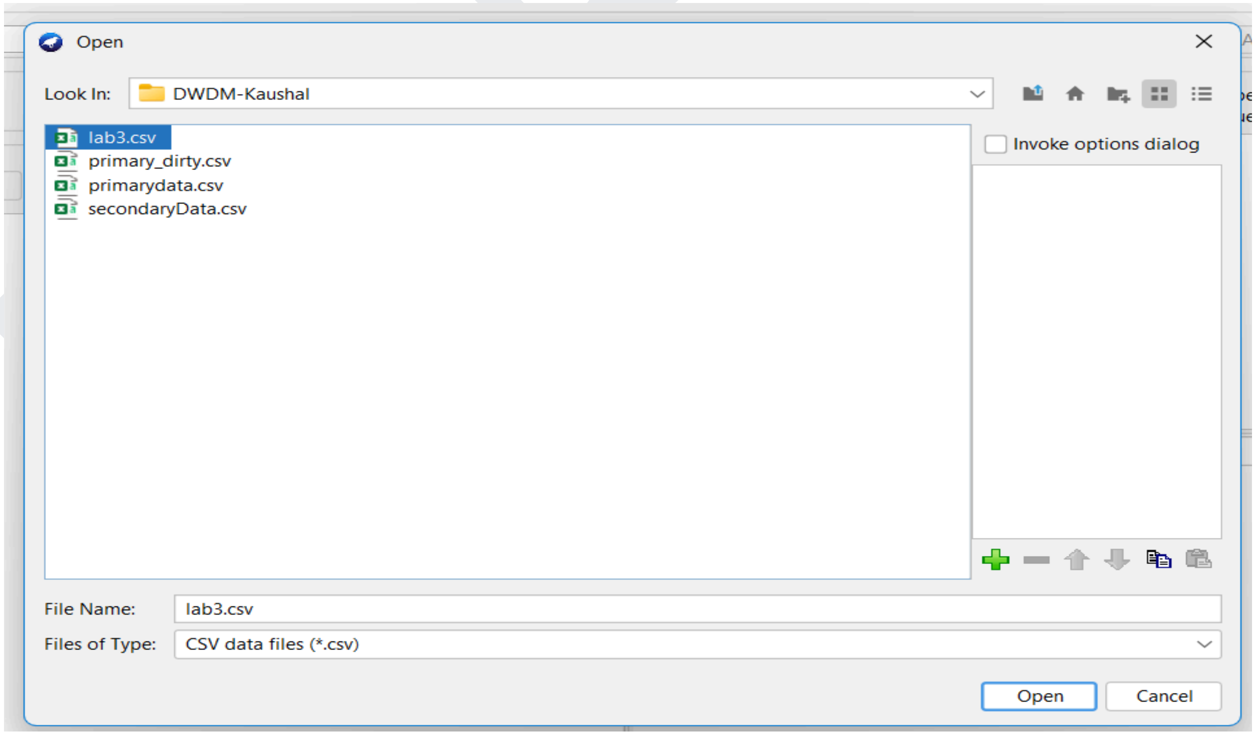
StudentID	Ecommerce	DWDM	OperationsManagement	AttendancePercent	StudyHours
1	85	78	82	92	6.5
2	45	52	48	65	2
3	72	68	75	85	5
4	38	42	40	58	1.5
5	90	88	92	95	7.5
6	65	70	68	78	4
7	55	60	58	70	3
8	78	75	80	88	5.5
9	42	48	45	62	2.5
10	88	85	90	93	7

2) Open Weka. Click Explorer. Then click on the Open file.



Waikato Environment for Knowledge Analysis
Version 3.9.6
(c) 1999 - 2022
The University of Waikato
Hamilton, New Zealand

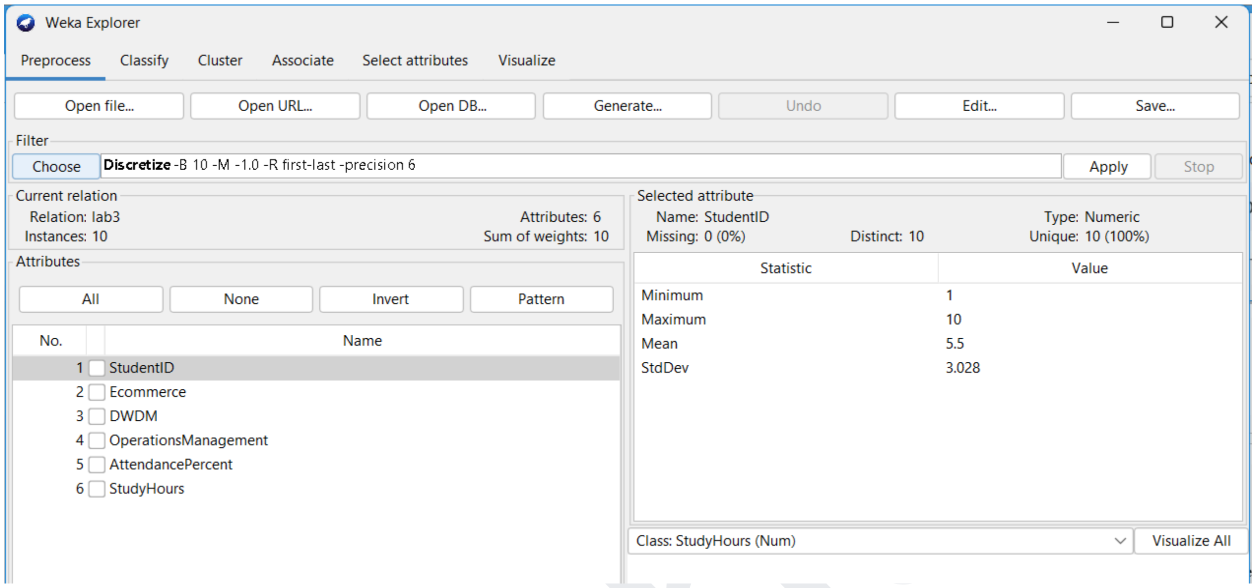
3) Click on file type and choose .csv and select the lab3.csv file.



File Name: lab3.csv
Files of Type: CSV data files (*.csv)

Done by- Kaushal

4) Navigate to filters → unsupervised → attribute → Discretize. Click on the filter name to open configuration window


Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Open file... | Open URL... | Open DB... | Generate... | Undo | Edit... | Save...

Filter

Choose | **Discretize** -B 10 -M -1.0 -R first-last -precision 6 | Apply | Stop

Current relation
Relation: lab3 Attributes: 6
Instances: 10 Sum of weights: 10

Selected attribute
Name: StudentID Type: Numeric
Missing: 0 (0%) Distinct: 10 Unique: 10 (100%)

Attributes

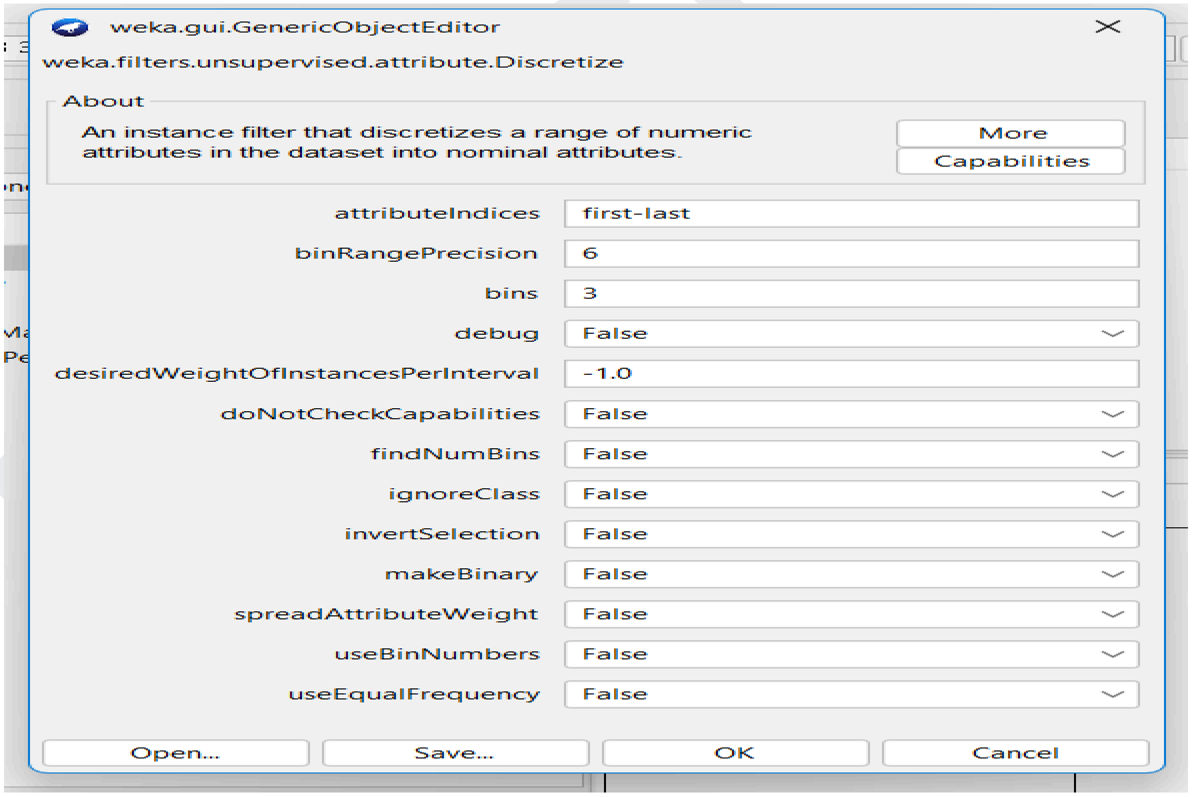
All | None | Invert | Pattern

No.	Name
1	StudentID
2	Ecommerce
3	DWDM
4	OperationsManagement
5	AttendancePercent
6	StudyHours

Statistic	Value
Minimum	1
Maximum	10
Mean	5.5
StdDev	3.028

Class: StudyHours (Num) | Visualize All

5) Set bins: 3 (This creates 3 categories: Low, Medium, High),keeping other settings as default and click OK


weka.gui.GenericObjectEditor

weka.filters.unsupervised.attribute.Discretize

About

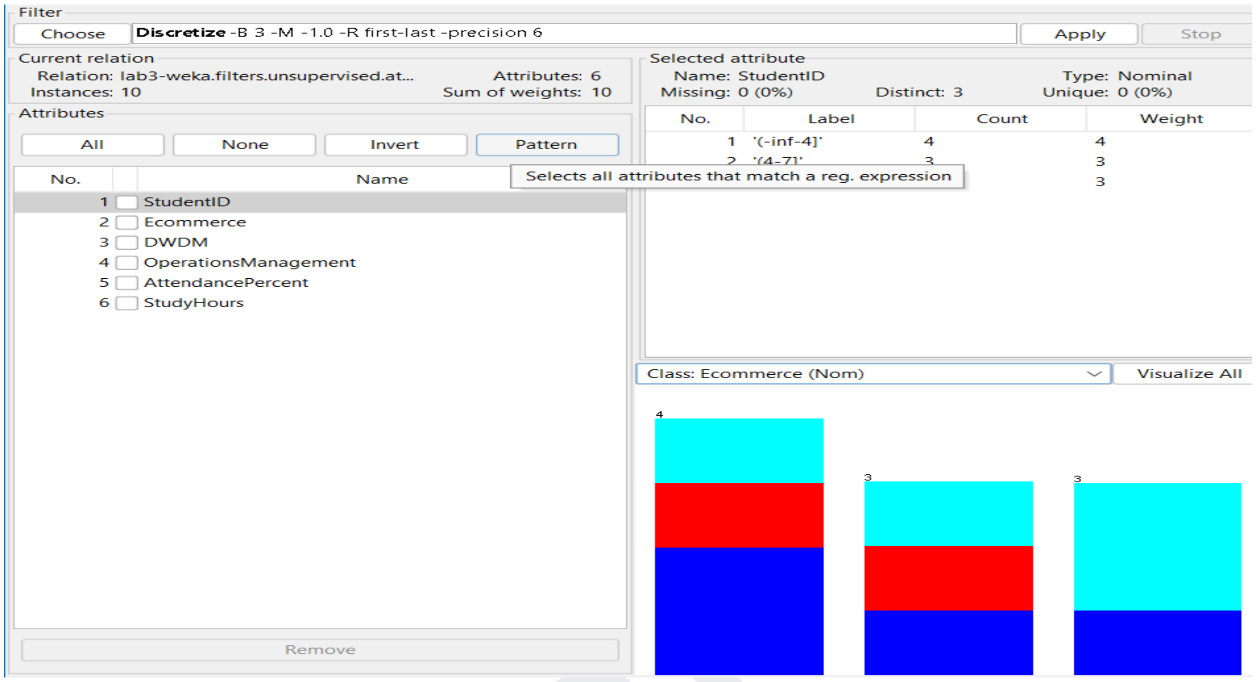
An instance filter that discretizes a range of numeric attributes in the dataset into nominal attributes.

More | Capabilities

attributeIndices	first-last
binRangePrecision	6
bins	3
debug	False
desiredWeightOfInstancesPerInterval	-1.0
doNotCheckCapabilities	False
findNumBins	False
ignoreClass	False
invertSelection	False
makeBinary	False
spreadAttributeWeight	False
useBinNumbers	False
useEqualFrequency	False

Open... | Save... | OK | Cancel

6) Click Apply button


Filter
Choose | **Discretize** -B 3 -M -1.0 -R first-last -precision 6 | Apply | Stop

Current relation
Relation: lab3-weka.filters.unsupervised.at... Attributes: 6
Instances: 10 Sum of weights: 10

Selected attribute
Name: StudentID Type: Nominal
Missing: 0 (0%) Distinct: 3 Unique: 0 (0%)

No.	Label	Count	Weight
1	'(-inf-4]'	4	4
2	'(4-7]'	3	3
			3

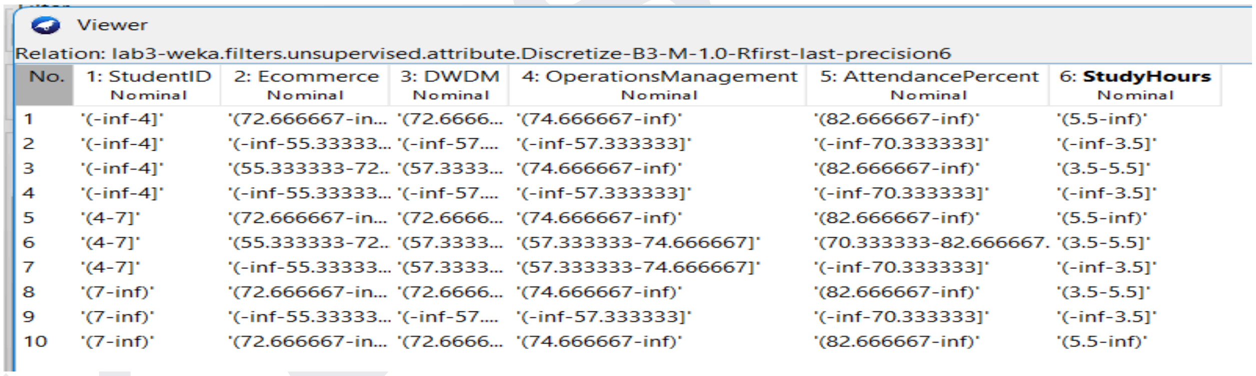
Attributes
All | None | Invert | Pattern
Selects all attributes that match a reg. expression

No.	Name
1	StudentID
2	Ecommerce
3	DWDM
4	OperationsManagement
5	AttendancePercent
6	StudyHours

Remove

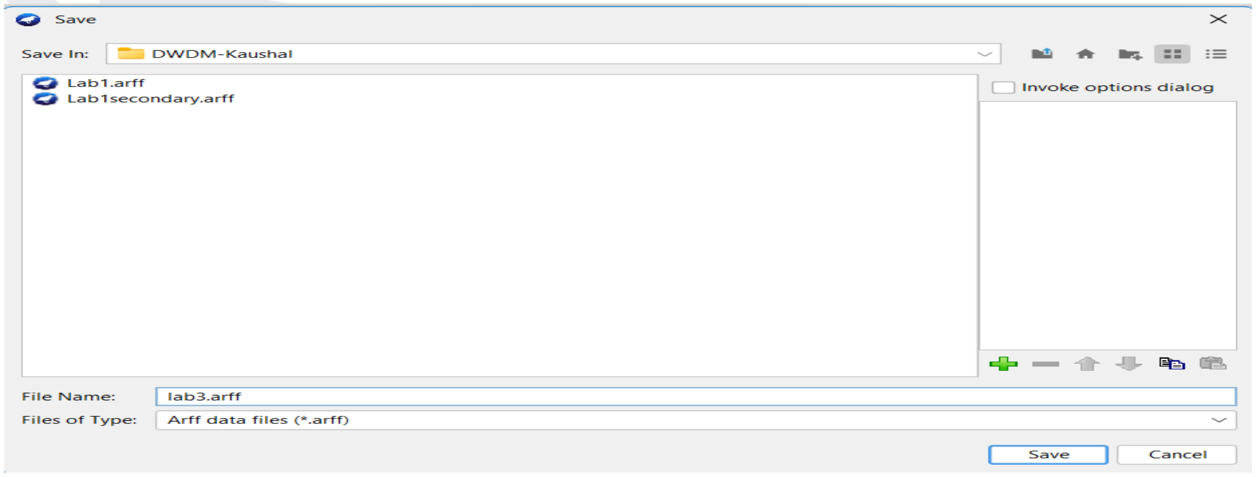
Class: Ecommerce (Nom) | Visualize All

7) Verify the discretization


Viewer
Relation: lab3-weka.filters.unsupervised.attribute.Discretize-B3-M-1.0-Rfirst-last-precision6

No.	1: StudentID Nominal	2: Ecommerce Nominal	3: DWDM Nominal	4: OperationsManagement Nominal	5: AttendancePercent Nominal	6: **StudyHours** Nominal
1	'(-inf-4]'	'(72.666667-in...	'(72.6666...	'(74.666667-inf)'	'(82.666667-inf)'	'(5.5-inf)'
2	'(-inf-4]'	'(-inf-55.33333...	'(-inf-57....	'(-inf-57.333333]'	'(-inf-70.333333]'	'(-inf-3.5]'
3	'(-inf-4]'	'(55.333333-72..	'(57.3333...	'(74.666667-inf)'	'(82.666667-inf)'	'(3.5-5.5]'
4	'(-inf-4]'	'(-inf-55.33333...	'(-inf-57....	'(-inf-57.333333]'	'(-inf-70.333333]'	'(-inf-3.5]'
5	'(4-7]'	'(72.666667-in...	'(72.6666...	'(74.666667-inf)'	'(82.666667-inf)'	'(5.5-inf)'
6	'(4-7]'	'(55.333333-72..	'(57.3333...	'(57.333333-74.666667]'	'(70.333333-82.666667.	'(3.5-5.5]'
7	'(4-7]'	'(-inf-55.33333...	'(57.3333...	'(57.333333-74.666667]'	'(-inf-70.333333]'	'(-inf-3.5]'
8	'(7-inf)'	'(72.666667-in...	'(72.6666...	'(74.666667-inf)'	'(82.666667-inf)'	'(3.5-5.5]'
9	'(7-inf)'	'(-inf-55.33333...	'(-inf-57....	'(-inf-57.333333]'	'(-inf-70.333333]'	'(-inf-3.5]'
10	'(7-inf)'	'(72.666667-in...	'(72.6666...	'(74.666667-inf)'	'(82.666667-inf)'	'(5.5-inf)'

8) Save the discretized dataset


Save
Save In: DWDM-Kaushal
Lab1.arff
Lab1secondary.arff
Invoke options dialog
File Name: lab3.arff
Files of Type: Arff data files (*.arff)
Save | Cancel

Done by- Kaushal

Discussion:
The numeric dataset contained five continuous attributes (Ecommerce, DWDM, Operations Management, AttendancePercent, and StudyHours) ranging from 0 to 100 for marks and 0 to 10 for study hours. The Discretize filter with 3 bins converted continuous values into Low, Medium, and High categories based on equal-width intervals, such as "(-inf-60]" for low scores, "(60-80]" for medium, and "(80-inf)" for high. This transformation makes patterns more interpretable, as identifying that students with "High" study hours tend to have "High" DWDM scores is more intuitive than correlating exact numeric values. The equal-width binning method ensures consistent interval sizes, though it may result in uneven distribution of instances across bins compared to alternative methods like equal-frequency binning..

Conclusion:
Discretization transforms the continuous numeric dataset into categorical format, simplifying the data structure while preserving essential patterns and relationships. This transformation demonstrates how discretization bridges raw numeric data and analytical requirements, making the dataset suitable for classification tasks, association rule mining, and algorithms that benefit from categorical inputs. The process highlights that appropriate data transformation techniques can significantly enhance both the interpretability and usability of datasets in data mining applications.

Lab Number: 4

Title: Use ID3 and J48 algorithms to design decision trees using a primary dataset. Assume the cross validation. Also visualize the tree using the J48 algorithm.

Objective: To design decision trees using ID3 and J48 algorithms on a primary dataset and visualize the tree structure using cross-validation in Weka.

IDE/Tools used: Weka 3.8.6

Components Used:

Primary Dataset: A classification dataset containing 7th semester student performance with categorical attributes (Ecommerce, DWDM, Operations Management, Attendance, StudyHours) and class attribute (Result: Pass/Fail)

Theory:

Decision Trees: A decision tree is a supervised machine learning algorithm used for classification and regression tasks. It creates a tree-like model of decisions based on feature values, where:

- ➔ Root Node: Represents the entire dataset and the first decision point
- ➔ Internal Nodes: Represent decision points based on attribute tests
- ➔ Branches: Represent the outcome of a test
- ➔ Leaf Nodes: Represent final class labels (Pass/Fail)

ID3 Algorithm (Iterative Dichotomiser 3):

- ➔ Developed by Ross Quinlan in 1986
- ➔ Uses Information Gain to select the best attribute for splitting
- ➔ Calculates entropy to measure impurity in the dataset
- ➔ Selects attributes that provide maximum information gain
- ➔ Works only with categorical data

J48 Algorithm:

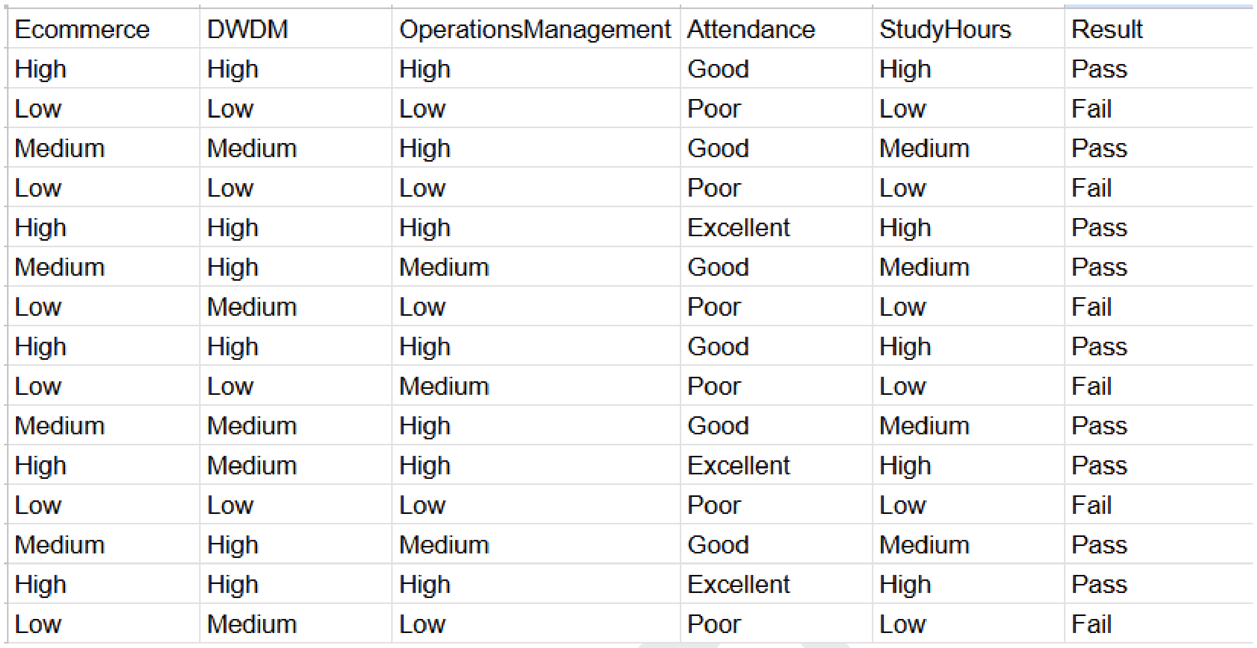
- ➔ An improved version of C4.5 algorithm
- ➔ Implemented in Weka as J48
- ➔ Handles both continuous and categorical data
- ➔ Performs automatic pruning to avoid overfitting
- ➔ More robust and widely used than ID3

Cross-Validation: Cross-validation is a technique to evaluate model performance by dividing data into k folds (typically 10). The model is trained on k-1 folds and tested on the remaining fold. This process repeats k times, and results are averaged to get reliable accuracy metrics.

Implementation Steps:

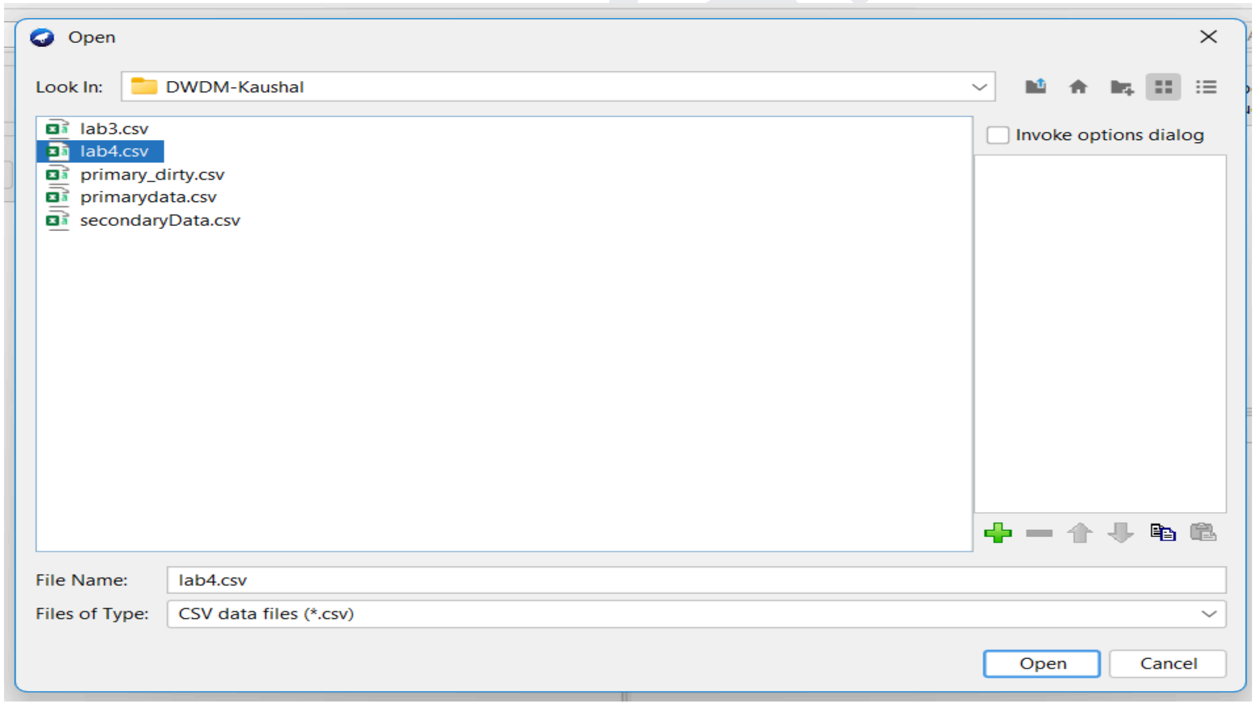
A: Preparing the Dataset

1) Save the dataset as CSV file (lab4.csv)

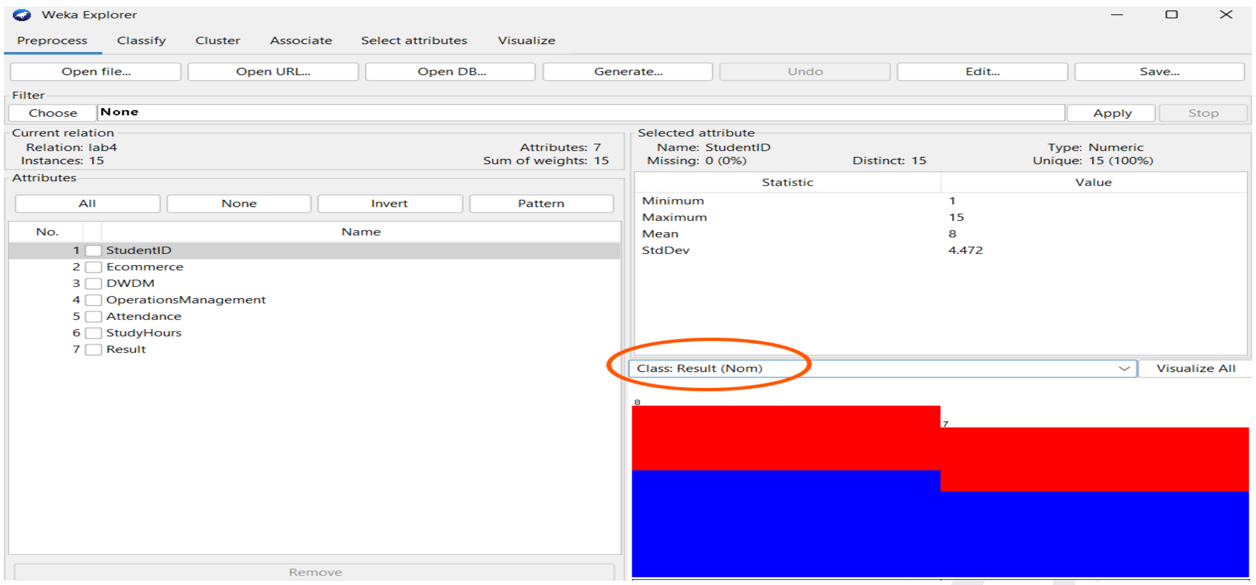


Ecommerce	DWDM	OperationsManagement	Attendance	StudyHours	Result
High	High	High	Good	High	Pass
Low	Low	Low	Poor	Low	Fail
Medium	Medium	High	Good	Medium	Pass
Low	Low	Low	Poor	Low	Fail
High	High	High	Excellent	High	Pass
Medium	High	Medium	Good	Medium	Pass
Low	Medium	Low	Poor	Low	Fail
High	High	High	Good	High	Pass
Low	Low	Medium	Poor	Low	Fail
Medium	Medium	High	Good	Medium	Pass
High	Medium	High	Excellent	High	Pass
Low	Low	Low	Poor	Low	Fail
Medium	High	Medium	Good	Medium	Pass
High	High	High	Excellent	High	Pass
Low	Medium	Low	Poor	Low	Fail

2) Open Weka Explorer → Click "Open file" → Select the CSV file i.e lab4.csv



3) Click on "Result" attribute in the Attributes section, then select "(Nom) Result" from the dropdown at the top to set it as the class attribute



Filter: None

Current relation
Relation: lab4
Instances: 15
Attributes: 7
Sum of weights: 15

Attributes

No.	Name
1	StudentID
2	Ecommerce
3	DWDM
4	OperationsManagement
5	Attendance
6	StudyHours
7	Result

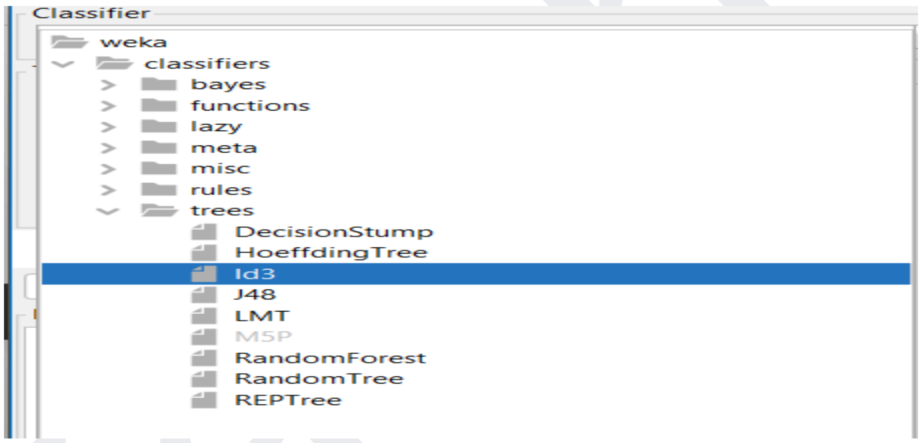
Selected attribute
Name: StudentID
Type: Numeric
Missing: 0 (0%)
Distinct: 15
Unique: 15 (100%)

Statistic	Value
Minimum	1
Maximum	15
Mean	8
StdDev	4.472

Class: Result (Nom)

Part B : Using ID3 Algorithm

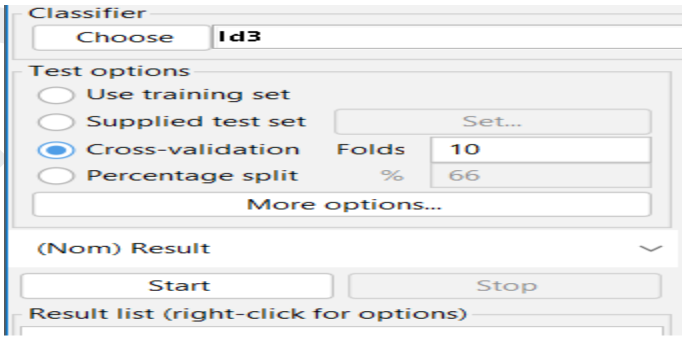
4) Go to the Classify tab and Click Choose button → Select trees → Id3



Note: Convert all numeric attributes to nominal as ID3 works with nominal type only.

1. Click Choose → filters → unsupervised → attribute → NumericToNominal and click apply

5) Select Cross-validation and set Folds to 10 and click start



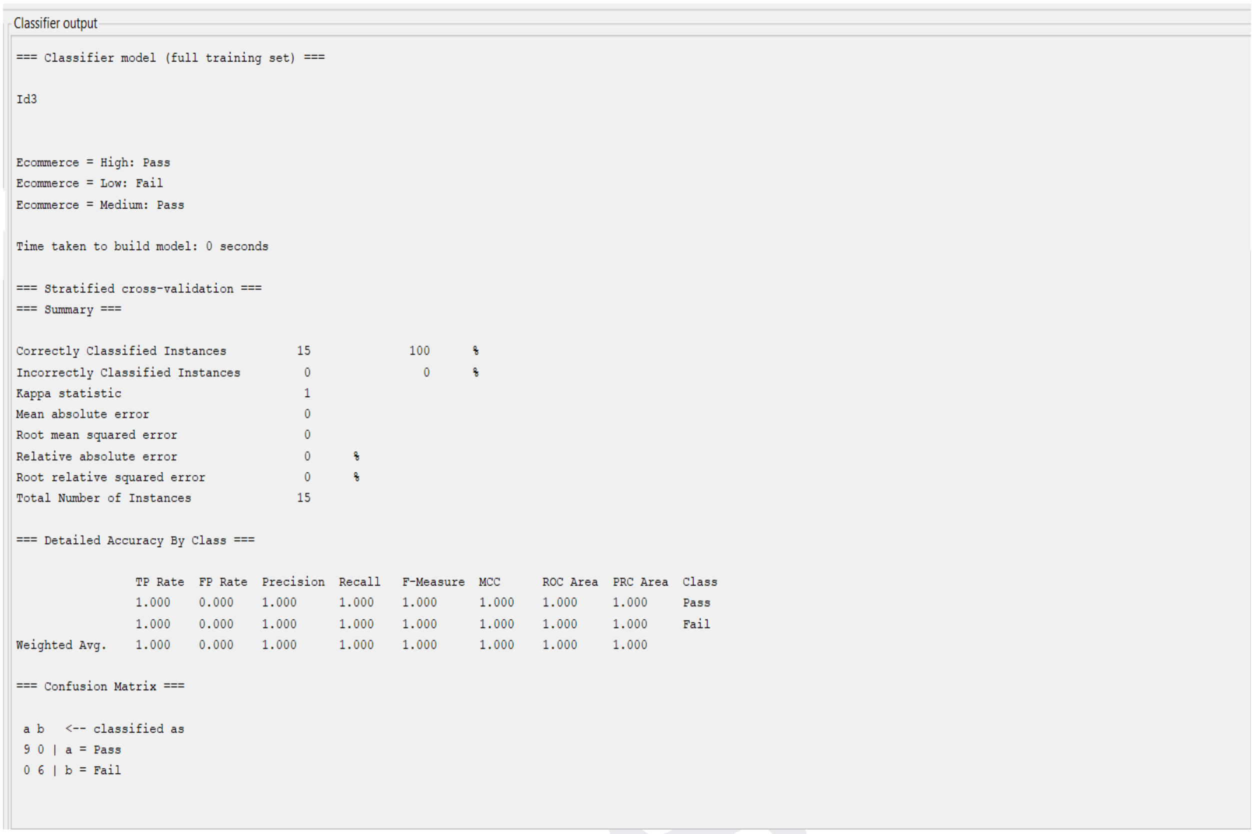
Classifier: Id3

Test options:
Use training set [ ]
Supplied test set [ ]
[✓ Cross-validation] Folds 10
Percentage split [ ] % 66

(Nom) Result

6) View the results showing accuracy percentage, confusion matrix, and detailed metrics

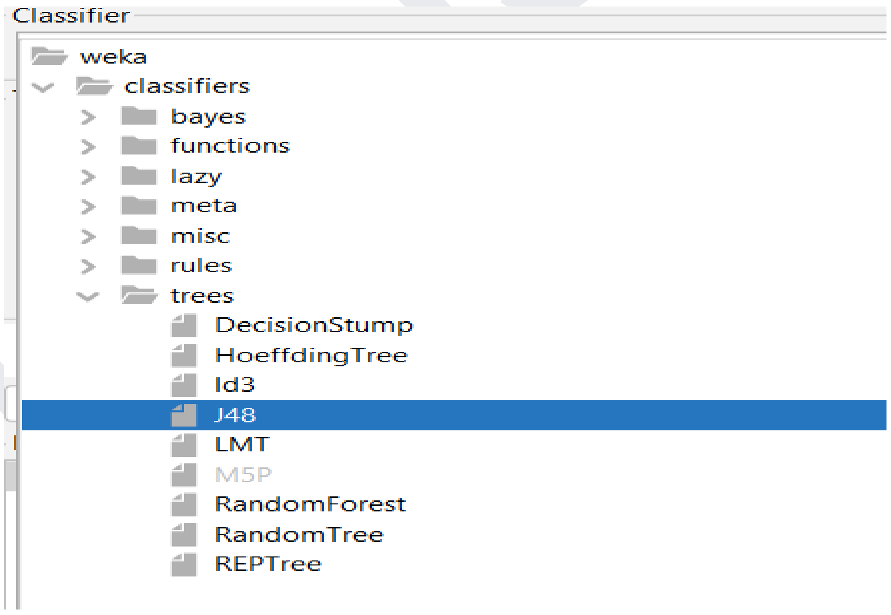
Done by- Kaushal


Classifier output

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

Id3


Ecommerce = High: Pass
Ecommerce = Low: Fail
Ecommerce = Medium: Pass

Time taken to build model: 0 seconds

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

Correctly Classified Instances          15              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances               15

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Pass
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Fail
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = Pass
 0 6 | b = Fail
```

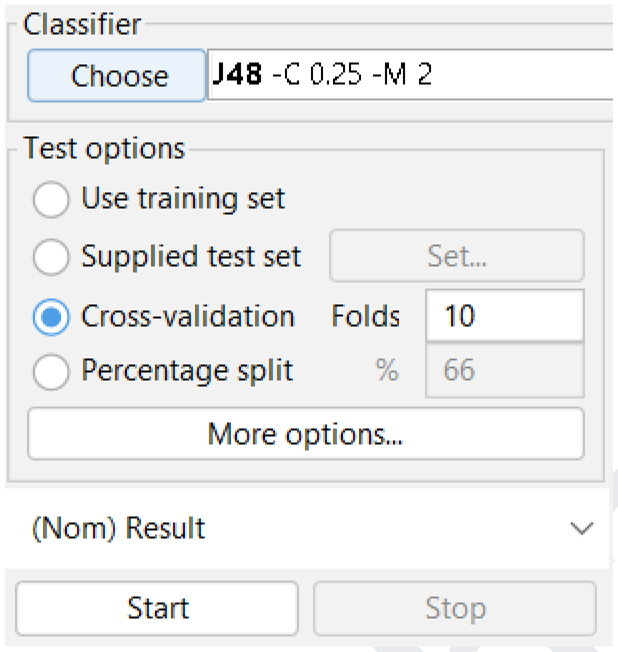
PART C: Using J48 Algorithm

7) Click Choose button again → Select trees → J48



8) Keep Cross-validation with 10 folds and click start

Done by- Kaushal


Classifier

Choose J48 -C 0.25 -M 2

Test options

Use training set

Supplied test set   Set...

[✓ Cross-validation]   Folds 10

Percentage split   % 66

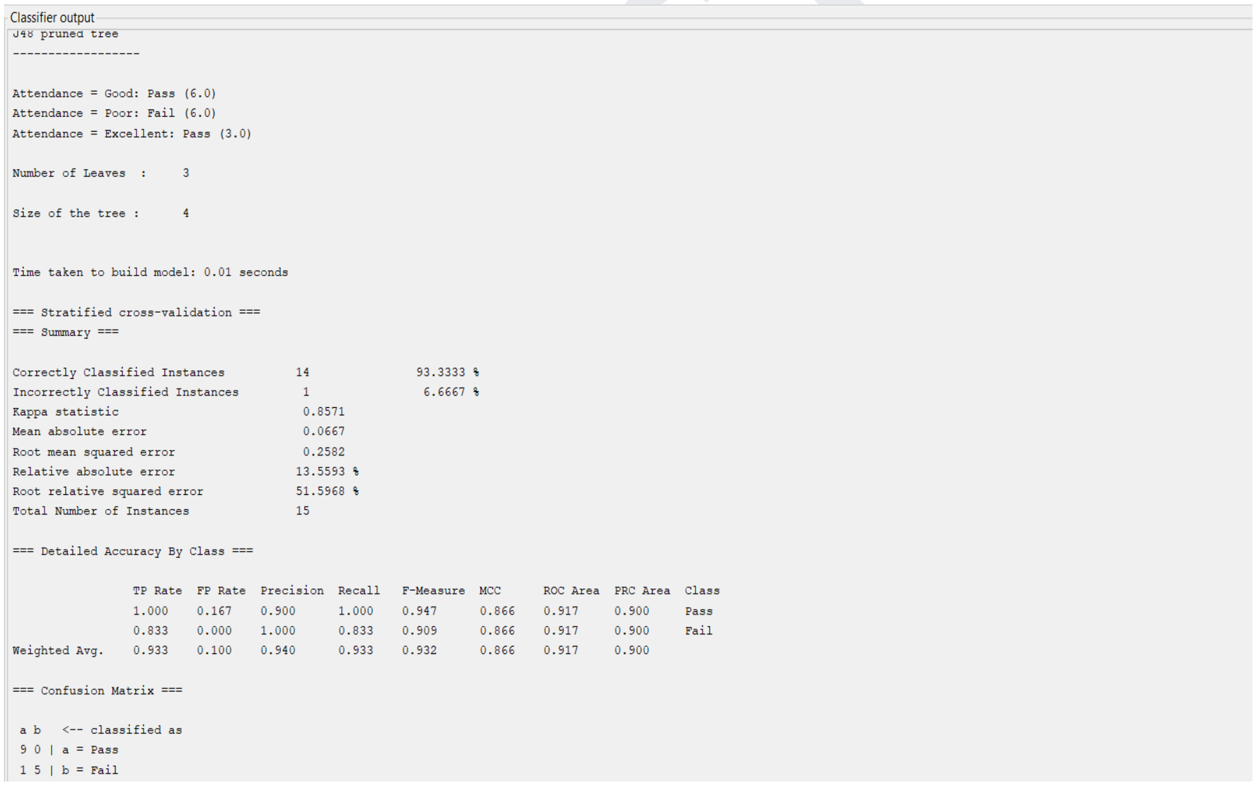
More options...

(Nom) Result

Start   Stop

9) View J48 Results

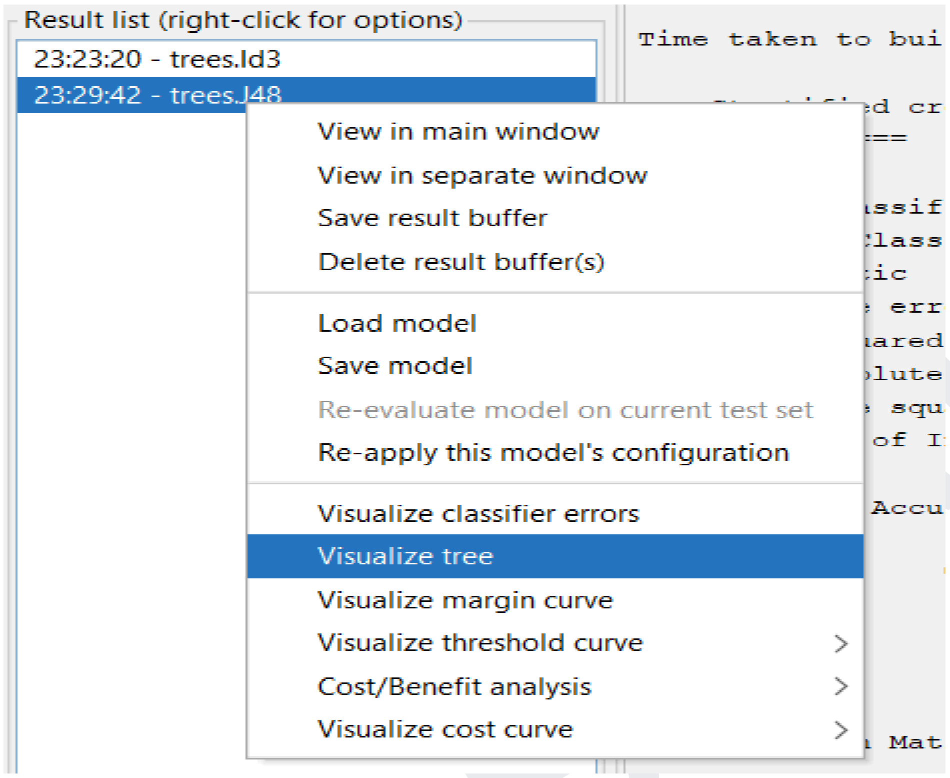
Compare accuracy with ID3 and Note the confusion matrix

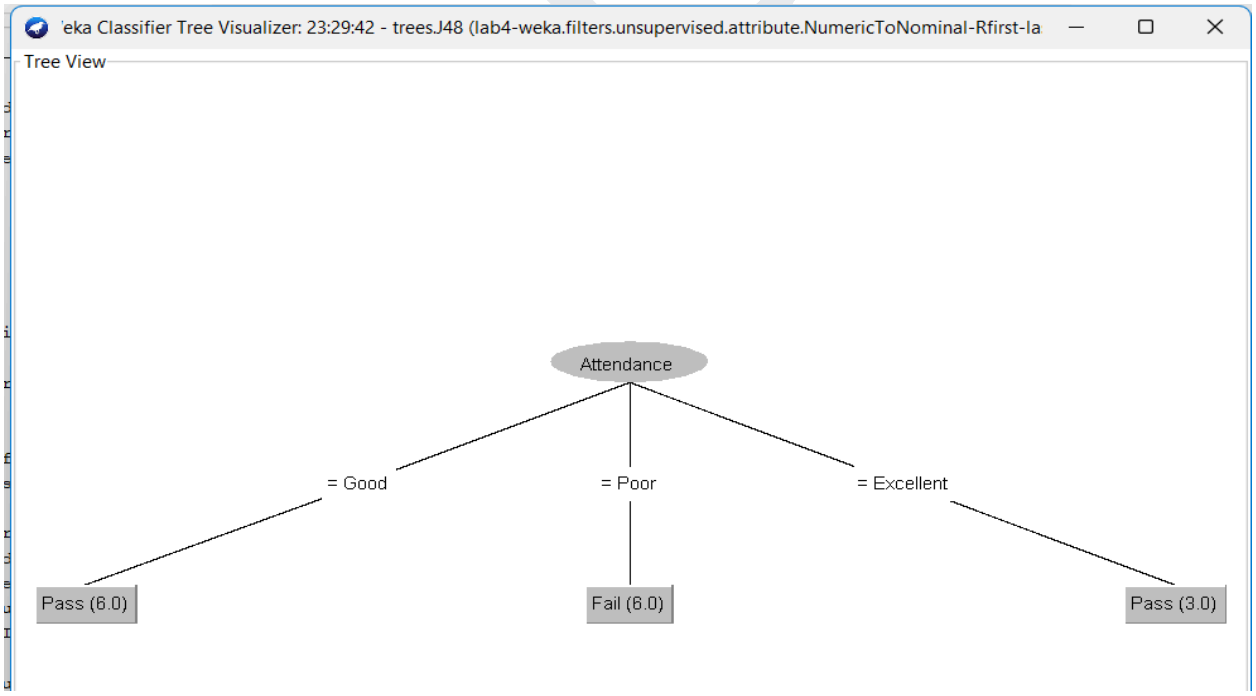

Classifier output

```
J48 pruned tree
------------------

Attendance = Good: Pass (6.0)
Attendance = Poor: Fail (6.0)
Attendance = Excellent: Pass (3.0)

Number of Leaves  :     3

Size of the tree :      4


Time taken to build model: 0.01 seconds

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

Correctly Classified Instances          14               93.3333 %
Incorrectly Classified Instances         1                6.6667 %
Kappa statistic                          0.8571
Mean absolute error                      0.0667
Root mean squared error                  0.2582
Relative absolute error                 13.5593 %
Root relative squared error             51.5968 %
Total Number of Instances               15

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.167    0.900      1.000    0.947      0.866    0.917     0.900     Pass
                 0.833    0.000    1.000      0.833    0.909      0.866    0.917     0.900     Fail
Weighted Avg.    0.933    0.100    0.940      0.933    0.932      0.866    0.917     0.900

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = Pass
 1 5 | b = Fail
```

Done by- Kaushal

10) Right-click on J48 result in "Result list" and select Visualize tree




'eka Classifier Tree Visualizer: 23:29:42 - trees.J48 (lab4-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-la
Tree View
Attendance
= Good
= Poor
= Excellent
Pass (6.0)
Fail (6.0)
Pass (3.0)

Done by- Kaushal

Discussion:
The dataset consisted of 15 student records with five predictor attributes (Ecommerce, DWDM, Operations Management, Attendance, StudyHours) and one class attribute (Result). Both ID3 and J48 algorithms were applied with 10-fold cross-validation, which divides the dataset into 10 parts and repeats training and testing 10 times for reliable accuracy estimates. ID3 selected attributes based on information gain to reduce entropy at each level, while J48 demonstrated similar or better performance due to additional features like pruning that prevents overfitting. The tree visualization clearly showed the decision-making process, such as predicting "Fail" for "Poor" attendance with "Low" study hours, versus "Pass" for "Good" or "Excellent" attendance with "High" study hours. The confusion matrix revealed classification accuracy, indicating the models successfully learned patterns distinguishing Pass from Fail students based on their academic and attendance attributes.

Conclusion:
Decision tree algorithms are fundamental classification techniques in data mining. ID3 demonstrates core principles of entropy-based splitting, while J48 offers advanced features like handling continuous data and automatic pruning. Cross-validation ensures reliable model evaluation by testing across multiple data partitions, preventing overfitting and providing trustworthy accuracy metrics. The key advantage of decision trees is their interpretability unlike complex black-box models, decision trees clearly show the logical path from attributes to predictions, making them essential for applications requiring transparent and explainable decision-making. This combination of accuracy and explainability makes decision trees valuable tools for practical data mining tasks.

Lab Number : 5

Title: Use Naïve Bayes Classifier on a primary dataset

Objective: To implement Naïve Bayes classification algorithm on a primary dataset and evaluate its performance using cross-validation in Weka.

IDE/Tools used: Weka 3.8.6

Components Used:

Primary Dataset: Student performance dataset containing categorical attributes (Ecommerce, DWDM, Operations Management, Attendance, StudyHours) and class attribute (Result: Pass/Fail)

Theory:

Naïve Bayes Classifier:

Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem. It is called "naïve" because it assumes that all features are independent of each other, which simplifies the calculation but rarely holds true in real-world scenarios.

Bayes' Theorem:

P(Class|Features) = [P(Features|Class) × P(Class)] / P(Features)

Where:

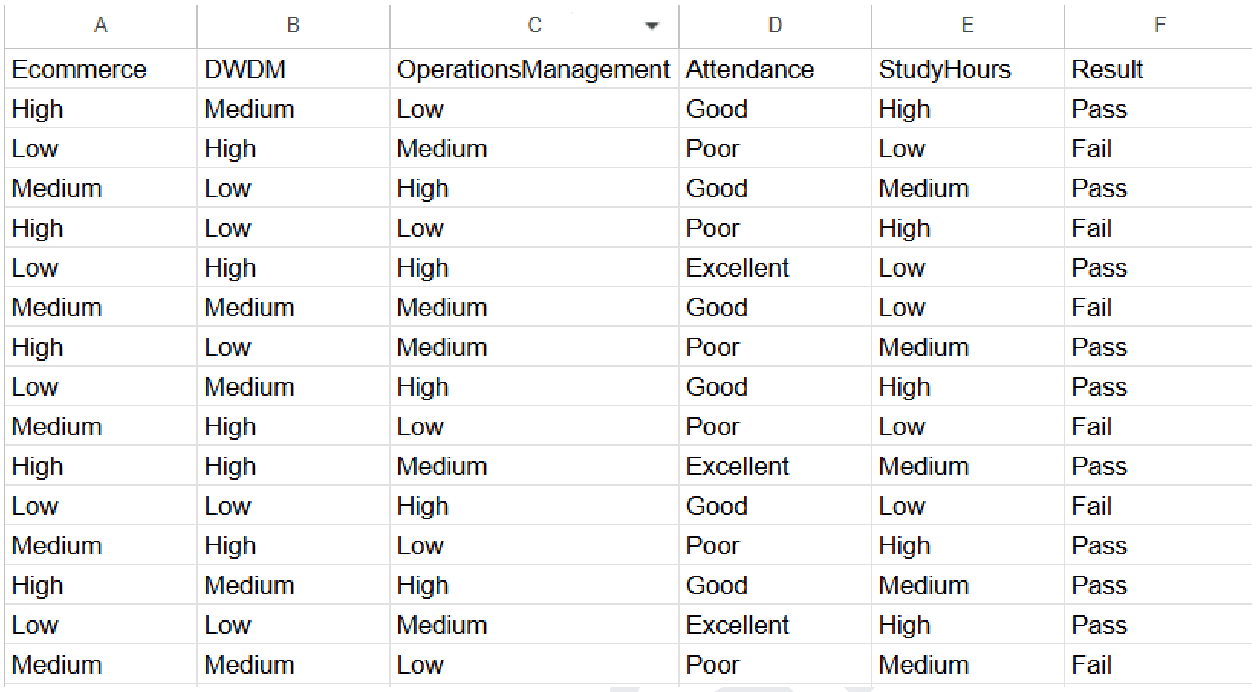
- ➔ P(Class|Features) = Probability of a class given the features (posterior probability)
- ➔ P(Features|Class) = Probability of features given the class (likelihood)
- ➔ P(Class) = Prior probability of the class
- ➔ P(Features) = Probability of the features

How Naïve Bayes Works:

1. Calculate the prior probability for each class (Pass/Fail)
2. Calculate the likelihood of each feature value for each class
3. For a new instance, multiply the prior probability by all feature likelihoods
4. Assign the class with the highest probability

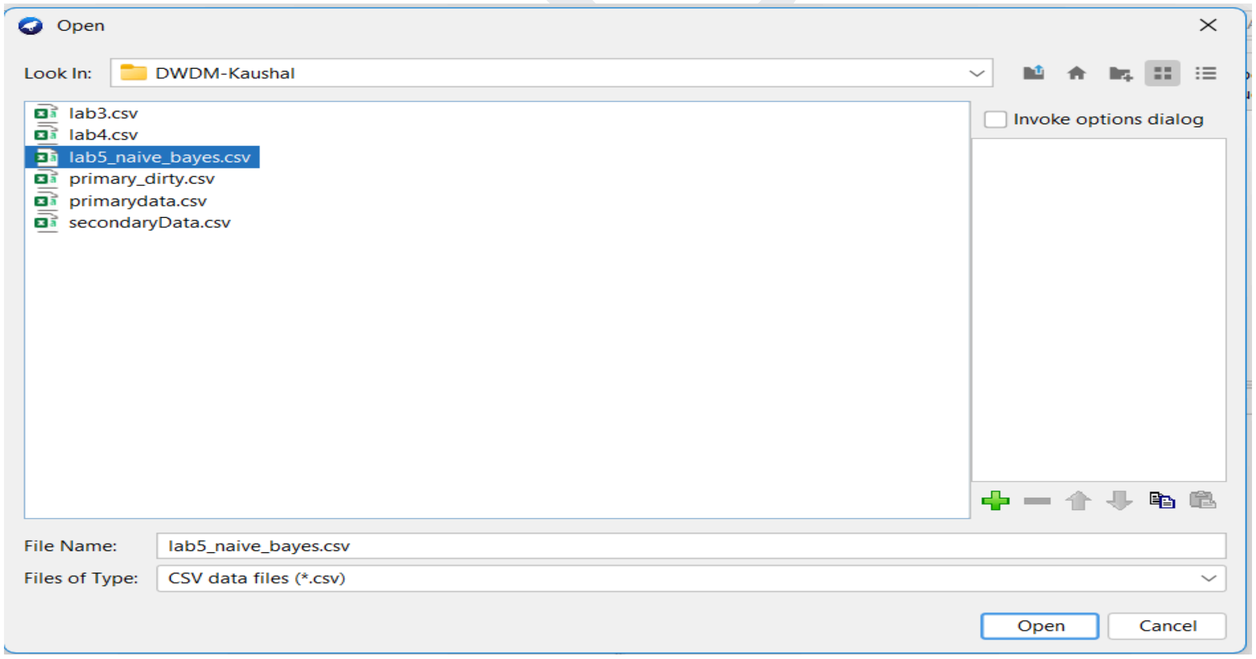
Implementation:

1) Save the dataset as CSV file (e.g., lab5_naive_bayes.csv)



A	B	C	D	E	F
Ecommerce	DWDM	OperationsManagement	Attendance	StudyHours	Result
High	Medium	Low	Good	High	Pass
Low	High	Medium	Poor	Low	Fail
Medium	Low	High	Good	Medium	Pass
High	Low	Low	Poor	High	Fail
Low	High	High	Excellent	Low	Pass
Medium	Medium	Medium	Good	Low	Fail
High	Low	Medium	Poor	Medium	Pass
Low	Medium	High	Good	High	Pass
Medium	High	Low	Poor	Low	Fail
High	High	Medium	Excellent	Medium	Pass
Low	Low	High	Good	Low	Fail
Medium	High	Low	Poor	High	Pass
High	Medium	High	Good	Medium	Pass
Low	Low	Medium	Excellent	High	Pass
Medium	Medium	Low	Poor	Medium	Fail

2) Open Weka Explorer → Click "Open file" → Select the CSV file



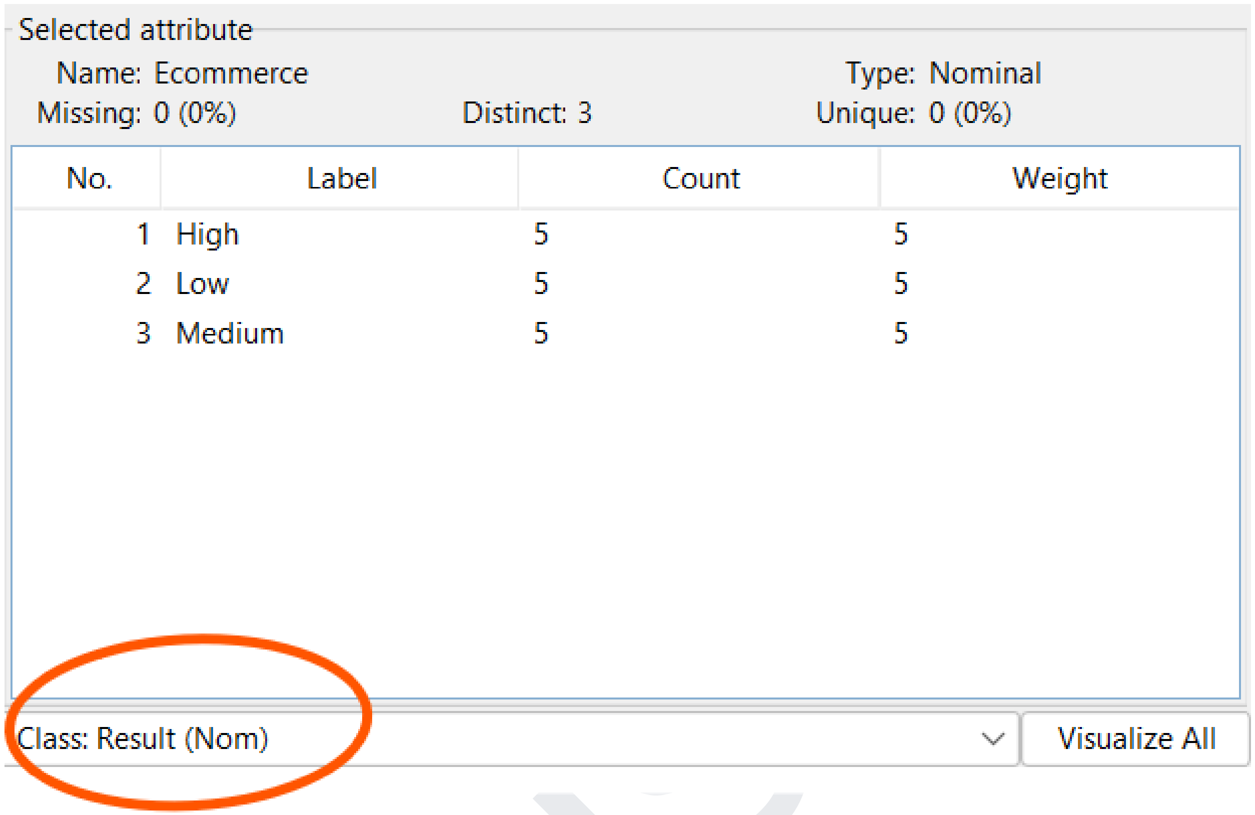
Look In: DWDM-Kaushal

lab3.csv
lab4.csv
lab5_naive_bayes.csv
primary_dirty.csv
primarydata.csv
secondaryData.csv

File Name: lab5_naive_bayes.csv

Files of Type: CSV data files (*.csv)

3) Set Result as class attribute by selecting " Result (Nom)" from the dropdown



Selected attribute

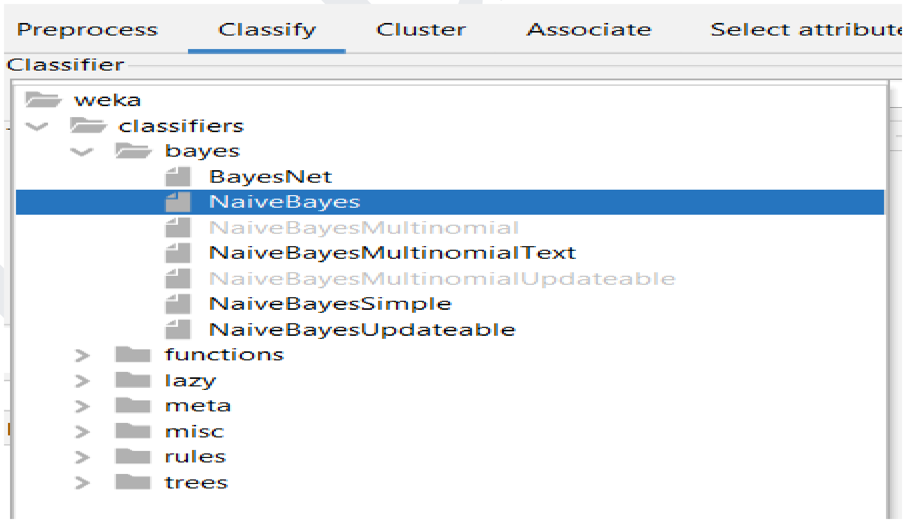
Name: Ecommerce Type: Nominal

Missing: 0 (0%) Distinct: 3 Unique: 0 (0%)

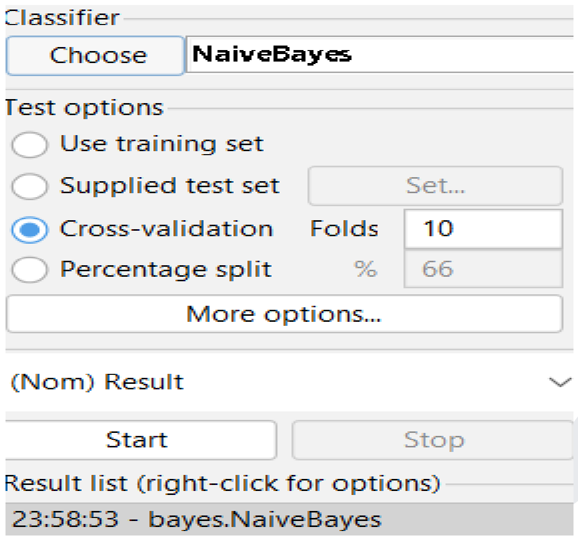
No.	Label	Count	Weight
1	High	5	5
2	Low	5	5
3	Medium	5	5

Class: Result (Nom)

4) Go to the Classify tab and Click Choose button → Select bayes → NaiveBayes



5) Select Cross-validation with 10 folds as the test option and click Start button

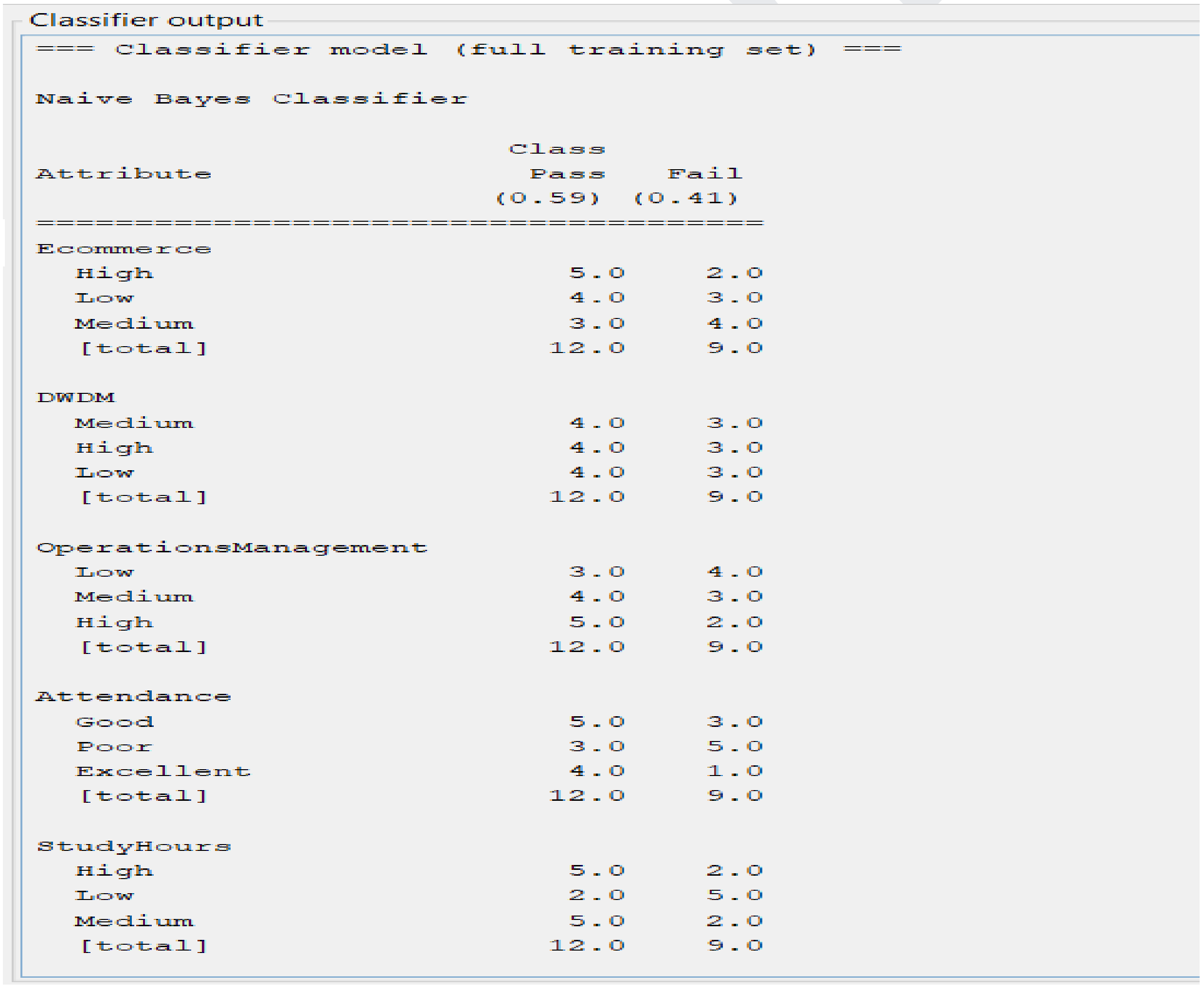


Classifier: NaiveBayes

Test options: [✓ Cross-validation] Folds 10

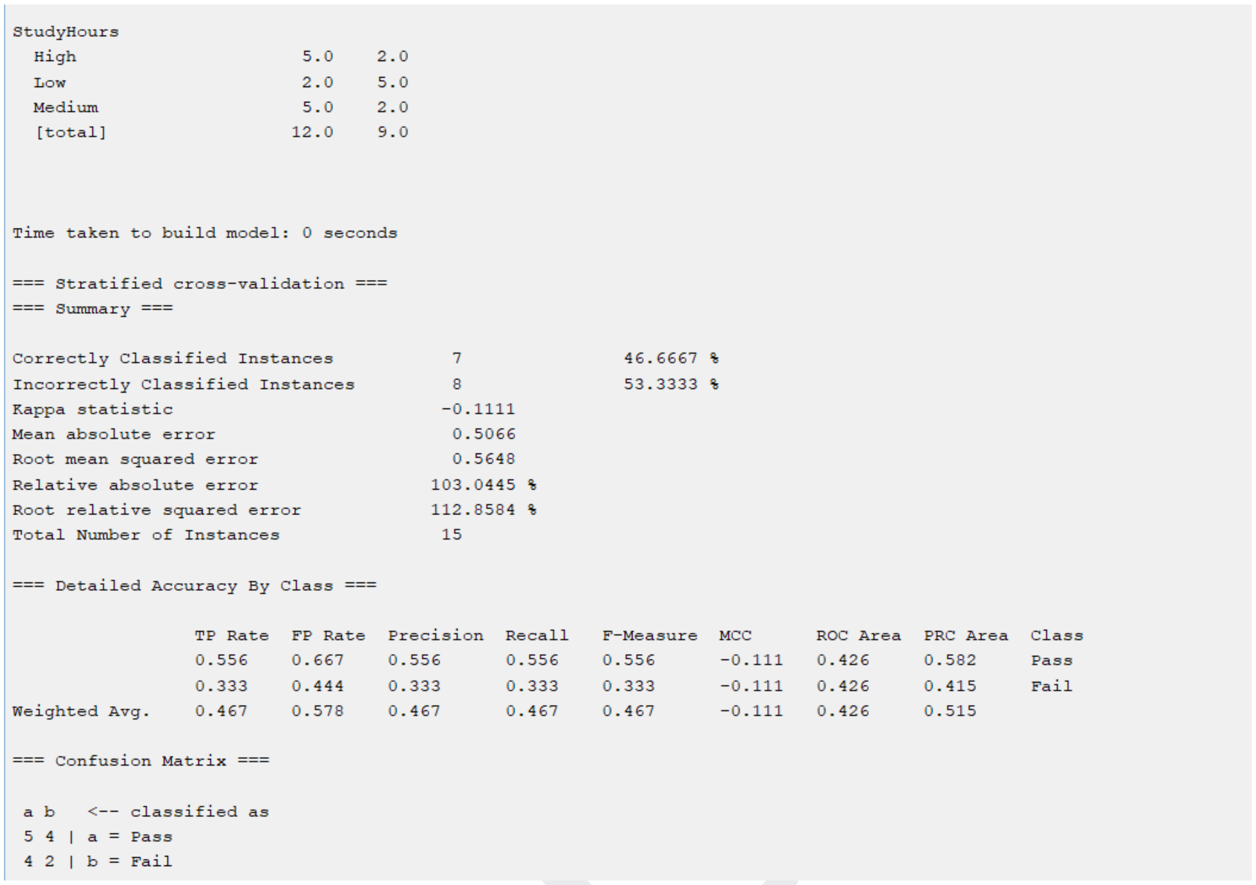
23:58:53 - bayes.NaiveBayes

6) Observe the result



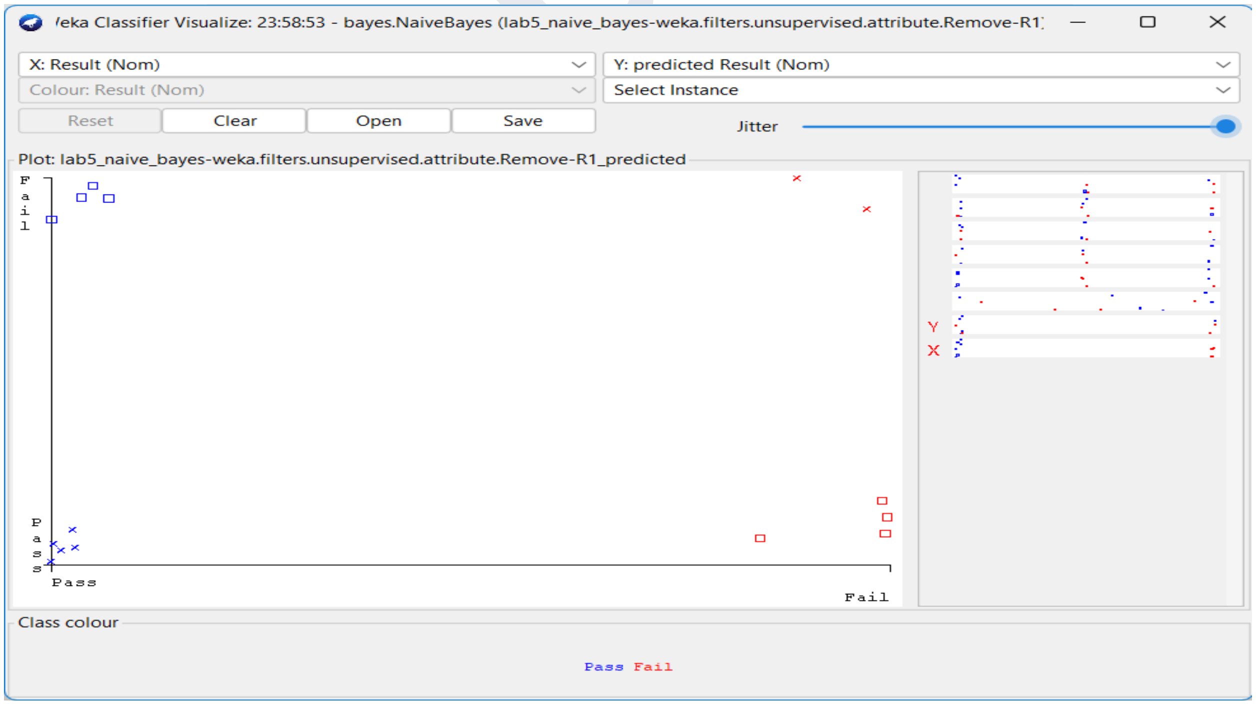
```
=== Classifier model (full training set) ===

Naive Bayes Classifier

                              Class
Attribute                      Pass    Fail
                              (0.59)  (0.41)
=====================================
Ecommerce
   High                         5.0     2.0
   Low                          4.0     3.0
   Medium                       3.0     4.0
   [total]                     12.0     9.0

DWDM
   Medium                       4.0     3.0
   High                         4.0     3.0
   Low                          4.0     3.0
   [total]                     12.0     9.0

OperationsManagement
   Low                          3.0     4.0
   Medium                       4.0     3.0
   High                         5.0     2.0
   [total]                     12.0     9.0

Attendance
   Good                         5.0     3.0
   Poor                         3.0     5.0
   Excellent                    4.0     1.0
   [total]                     12.0     9.0

StudyHours
   High                         5.0     2.0
   Low                          2.0     5.0
   Medium                       5.0     2.0
   [total]                     12.0     9.0
```

Done by- Kaushal



```
StudyHours
  High                        5.0    2.0
  Low                         2.0    5.0
  Medium                      5.0    2.0
  [total]                    12.0    9.0



Time taken to build model: 0 seconds

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

Correctly Classified Instances           7               46.6667 %
Incorrectly Classified Instances         8               53.3333 %
Kappa statistic                         -0.1111
Mean absolute error                      0.5066
Root mean squared error                  0.5648
Relative absolute error                103.0445 %
Root relative squared error            112.8584 %
Total Number of Instances               15

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.556    0.667    0.556      0.556    0.556      -0.111   0.426     0.582     Pass
                 0.333    0.444    0.333      0.333    0.333      -0.111   0.426     0.415     Fail
Weighted Avg.    0.467    0.578    0.467      0.467    0.467      -0.111   0.426     0.515

=== Confusion Matrix ===

 a b   <-- classified as
 5 4 | a = Pass
 4 2 | b = Fail
```

7) Right-click on result → Select "Visualize classifier errors" to see misclassified instances (if any)



/eka Classifier Visualize: 23:58:53 - bayes.NaiveBayes (lab5_naive_bayes-weka.filters.unsupervised.attribute.Remove-R1)

X: Result (Nom)

Y: predicted Result (Nom)

Colour: Result (Nom)

Select Instance

Plot: lab5_naive_bayes-weka.filters.unsupervised.attribute.Remove-R1_predicted

Done by- Kaushal

Discussion:
The Naïve Bayes classifier was applied to a dataset containing 15 student records with five predictor attributes and one class attribute. The algorithm calculates probabilities for each class based on attribute value frequencies, computing the probability of passing or failing given performance in Ecommerce, DWDM, Operations Management, attendance, and study hours. Despite the independence assumption treating each attribute as unrelated, Naïve Bayes often performs well in practice. The 10-fold cross-validation ensures robust evaluation by training and testing on different data subsets, providing realistic performance estimates rather than overfitting. Unlike decision trees that provide deterministic rules, Naïve Bayes outputs probability scores allowing confidence assessment in predictions. The confusion matrix reveals correct and incorrect prediction distributions, and comparing Naïve Bayes with decision tree algorithms demonstrates different approaches to the same classification problem with trade-offs between model complexity and performance.

Conclusion:
Naïve Bayes classifier is an efficient probabilistic algorithm based on Bayes' Theorem that performs well for categorical classification tasks. Despite its simplifying independence assumption, the algorithm remains competitive due to computational efficiency and robustness with limited training data. Cross-validation demonstrated reliable performance and generalization capability, while the probabilistic nature provides confidence scores rather than only deterministic outputs, valuable for uncertainty quantification in decision-making. This exercise demonstrates that Naïve Bayes is particularly effective for problems involving categorical features and scenarios requiring fast, interpretable predictions, making it a practical choice for real-world applications where computational resources are limited or rapid model deployment is necessary.

Lab Number : 6

Title: Illustrate Multilayer Perceptron

Objective: To implement a Multilayer Perceptron (MLP) neural network for classification and evaluate its performance using Weka.

IDE/Tools used: Weka 3.8.6

Components Used:

Primary Dataset: Student performance dataset with numeric attributes (Ecommerce marks, DWDM marks, Operations Management marks, Attendance percentage, StudyHours) and categorical class (Result: Pass/Fail)

Theory:

A Multilayer Perceptron is an artificial neural network consisting of multiple layers of interconnected nodes (neurons). It is a supervised learning algorithm capable of learning complex non-linear relationships between input features and output classes.

Architecture Components:

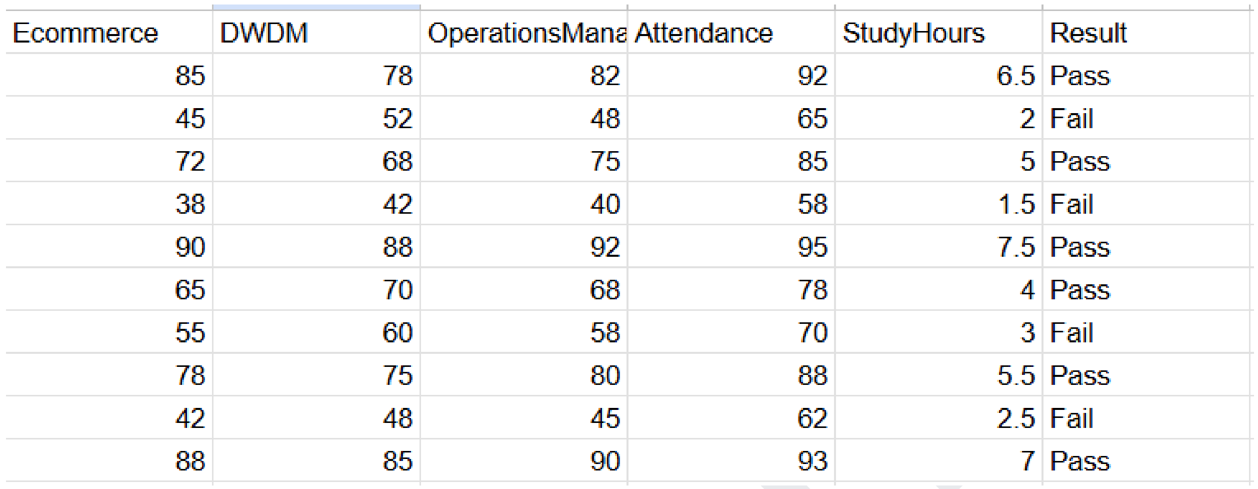
1. **Input Layer:** Receives the input features (one neuron per attribute)
2. **Hidden Layer(s):** Intermediate layers that process and transform data through weighted connections
3. **Output Layer:** Produces the final classification result
4. **Weights:** Connection strengths between neurons that are adjusted during training
5. **Activation Function:** Non-linear function (typically sigmoid or ReLU) applied at each neuron

How MLP Works:

1. **Forward Propagation:** Input data flows through the network, layer by layer, with each neuron applying weights and activation functions
2. **Loss Calculation:** Compare predicted output with actual class labels to compute error
3. **Backpropagation:** Error is propagated backward through the network to adjust weights
4. **Weight Update:** Weights are modified using gradient descent to minimize error
5. **Iteration:** Process repeats for multiple epochs until convergence

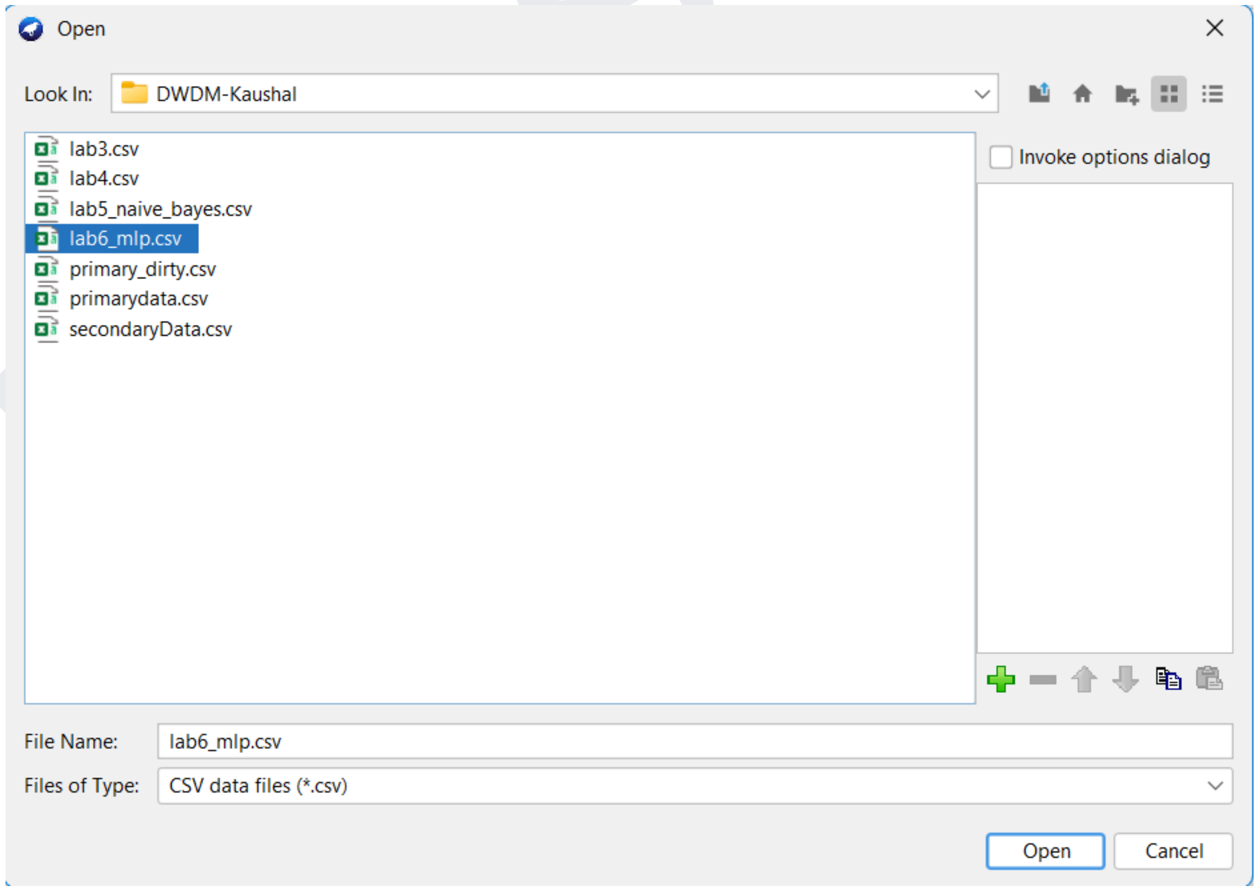
Implementation:

1) Save the dataset as CSV file (e.g., lab6_mlp.csv)



Ecommerce	DWDM	OperationsMana	Attendance	StudyHours	Result
85	78	82	92	6.5	Pass
45	52	48	65	2	Fail
72	68	75	85	5	Pass
38	42	40	58	1.5	Fail
90	88	92	95	7.5	Pass
65	70	68	78	4	Pass
55	60	58	70	3	Fail
78	75	80	88	5.5	Pass
42	48	45	62	2.5	Fail
88	85	90	93	7	Pass

2) Open Weka Explorer → Click "Open file" → Select the CSV file



Look In: DWDM-Kaushal

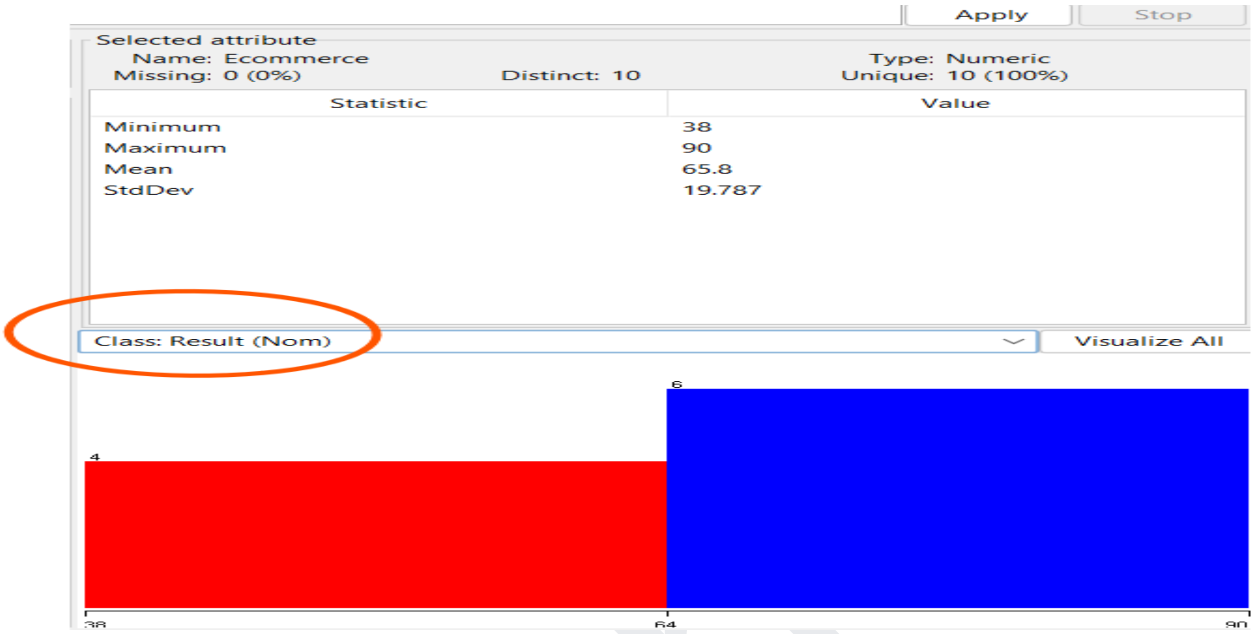
lab3.csv
lab4.csv
lab5_naive_bayes.csv
lab6_mlp.csv
primary_dirty.csv
primarydata.csv
secondaryData.csv

File Name: lab6_mlp.csv

Files of Type: CSV data files (*.csv)

Done by- Kaushal

3) Set Result as class attribute by selecting "Result(Nom)" from the dropdown at the top

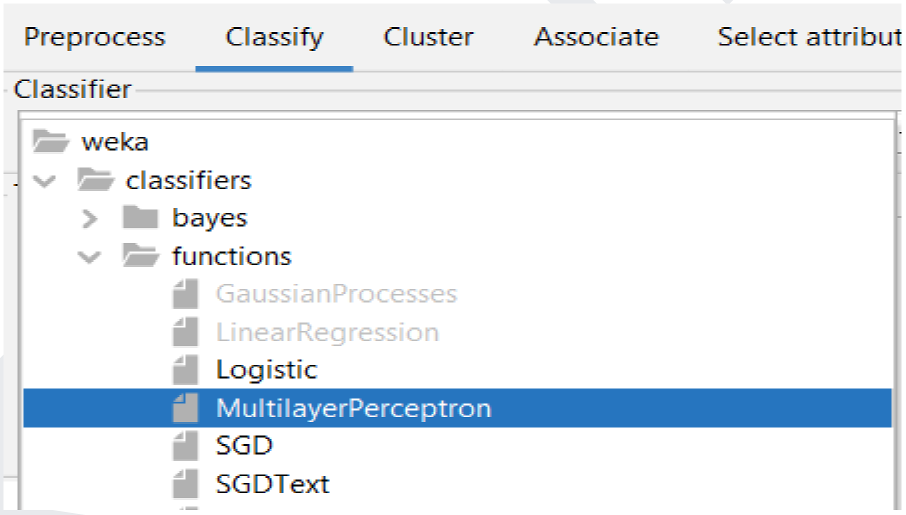


Selected attribute
Name: Ecommerce
Missing: 0 (0%)
Distinct: 10
Type: Numeric
Unique: 10 (100%)

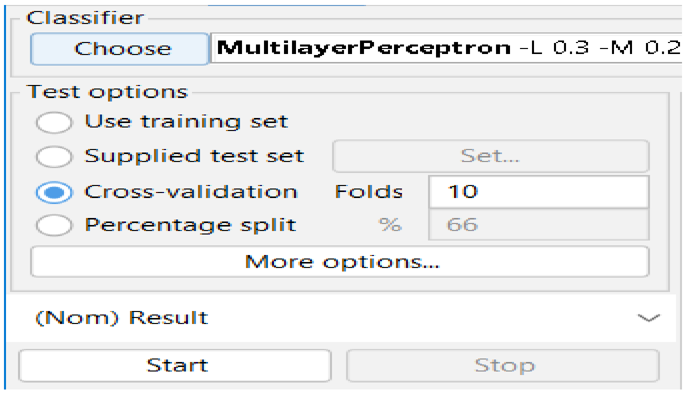
Statistic	Value
Minimum	38
Maximum	90
Mean	65.8
StdDev	19.787

Class: Result (Nom)

4) Go to the Classify tab and Click Choose button→Select function→MultilayerPerceptron



5) Select Cross-validation with 10 folds as the test option and click on Start Button



MultilayerPerceptron -L 0.3 -M 0.2

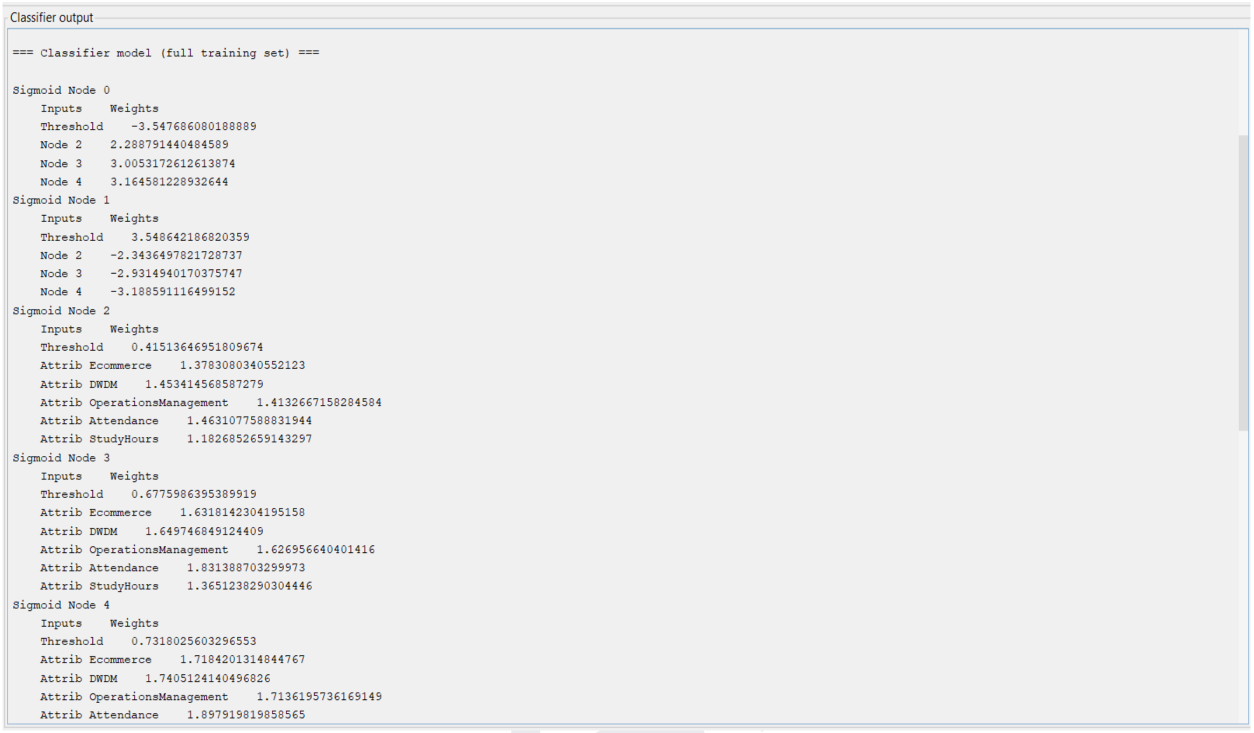
Cross-validation Folds 10

% 66

(Nom) Result

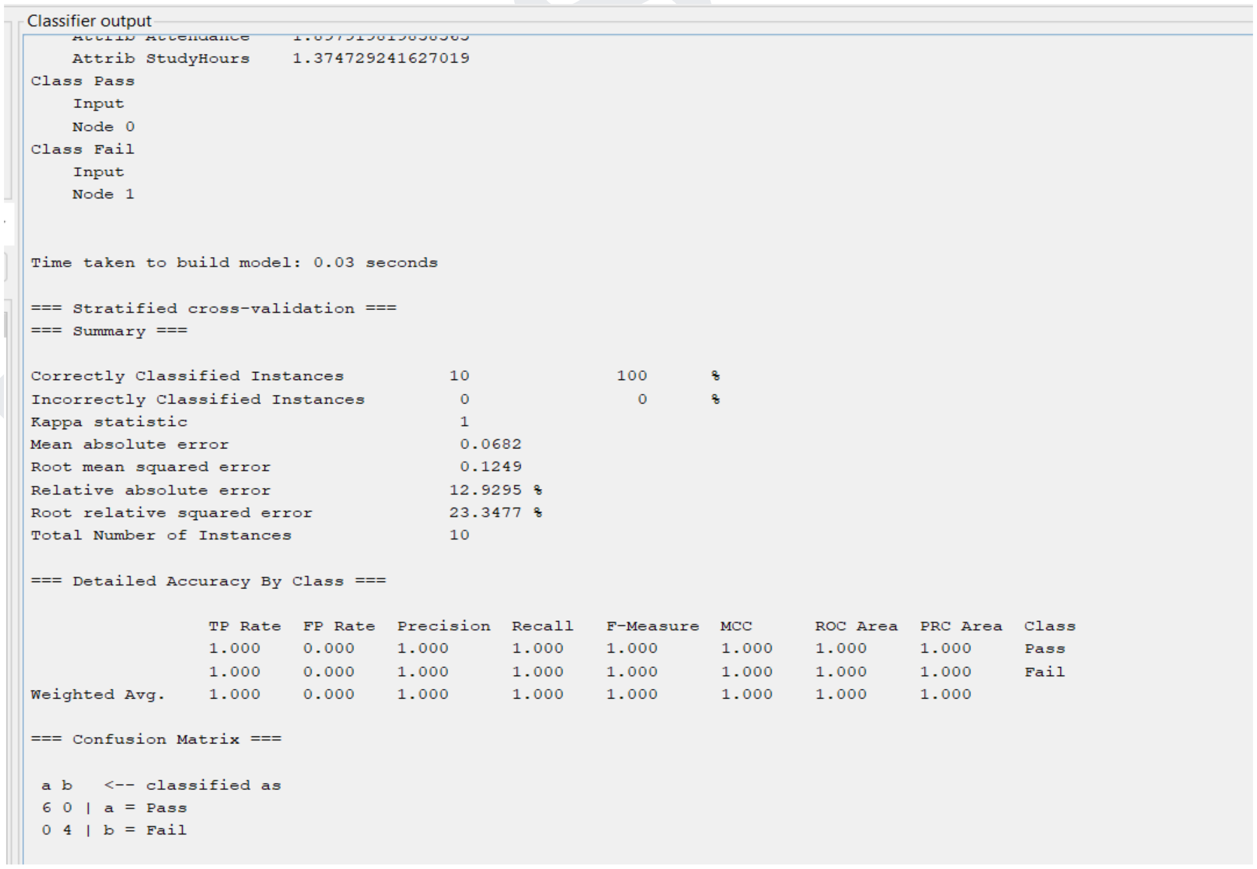
Done by- Kaushal

6) Observe the results showing accuracy, confusion matrix, training time, and detailed performance metrics



Classifier output

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

Sigmoid Node 0
    Inputs    Weights
    Threshold    -3.547686080188889
    Node 2    2.288791440484589
    Node 3    3.0053172612613874
    Node 4    3.164581228932644
Sigmoid Node 1
    Inputs    Weights
    Threshold    3.548642186820359
    Node 2    -2.3436497821728737
    Node 3    -2.9314940170375747
    Node 4    -3.188591116499152
Sigmoid Node 2
    Inputs    Weights
    Threshold    0.41513646951809674
    Attrib Ecommerce    1.3783080340552123
    Attrib DWDM    1.453414568587279
    Attrib OperationsManagement    1.4132667158284584
    Attrib Attendance    1.4631077588831944
    Attrib StudyHours    1.1826852659143297
Sigmoid Node 3
    Inputs    Weights
    Threshold    0.6775986395389919
    Attrib Ecommerce    1.6318142304195158
    Attrib DWDM    1.649746849124409
    Attrib OperationsManagement    1.626956640401416
    Attrib Attendance    1.831388703299973
    Attrib StudyHours    1.3651238290304446
Sigmoid Node 4
    Inputs    Weights
    Threshold    0.7318025603296553
    Attrib Ecommerce    1.7184201314844767
    Attrib DWDM    1.7405124140496826
    Attrib OperationsManagement    1.7136195736169149
    Attrib Attendance    1.897919819858565
```



Classifier output

```
    Attrib StudyHours    1.374729241627019
Class Pass
    Input
    Node 0
Class Fail
    Input
    Node 1


Time taken to build model: 0.03 seconds

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

Correctly Classified Instances          10               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0682
Root mean squared error                  0.1249
Relative absolute error                 12.9295 %
Root relative squared error             23.3477 %
Total Number of Instances               10

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Pass
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Fail
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

 a b   <-- classified as
 6 0 | a = Pass
 0 4 | b = Fail
```

Done by- Kaushal

Discussion:
The Multilayer Perceptron was applied to a dataset containing 10 student records with five numeric predictor attributes and one categorical class variable. Unlike decision trees or Naïve Bayes, MLP naturally handles numeric data through mathematical neuron computations, constructing a network with input, hidden, and output layers while iteratively adjusting connection weights through backpropagation to minimize classification error. Cross-validation with 10 folds ensures reliable performance evaluation, and MLP's ability to learn non-linear decision boundaries makes it potentially more powerful than linear classifiers, though training time is notably longer due to iterative optimization involving multiple epochs. The performance metrics and confusion matrix reveal how effectively the neural network learned to distinguish between passing and failing students, with the layered architecture enabling it to capture complex patterns that traditional rule-based methods may miss.

Conclusion:
Multilayer Perceptron represents a powerful neural network approach for classification tasks, using backpropagation to learn complex non-linear relationships in data. Its layered architecture and iterative weight optimization enable the model to capture patterns that simpler algorithms might miss, though this comes at the cost of increased computational time and reduced interpretability. Cross-validation demonstrated the model's generalization capability, and neural networks like MLP excel in scenarios involving numeric features and complex decision boundaries where traditional methods may struggle. This exercise illustrates that neural networks are valuable tools in modern machine learning, offering sophisticated pattern recognition capabilities, though their effectiveness depends on adequate training data, proper parameter tuning, and acceptance of the black-box nature inherent in deep learning approaches.

Lab Number : 7

Title: Generate association rules using Apriori algorithm for a suitable primary dataset

Objective: To discover association rules from transactional data using the Apriori algorithm with minimum support of 0.2 and minimum confidence of 0.7 in Weka.

IDE/Tools used: Weka 3.8.6

Components Used:

Primary Dataset: Computer accessories purchase transactions containing items (Laptop, Mouse, Keyboard, Headphone, USB) with binary values (Yes/No)

Theory:

Association Rule Mining:

Association rule mining is an unsupervised learning technique used to discover interesting relationships, patterns, and associations among items in large datasets. It is widely used in market basket analysis to understand customer purchasing behavior.

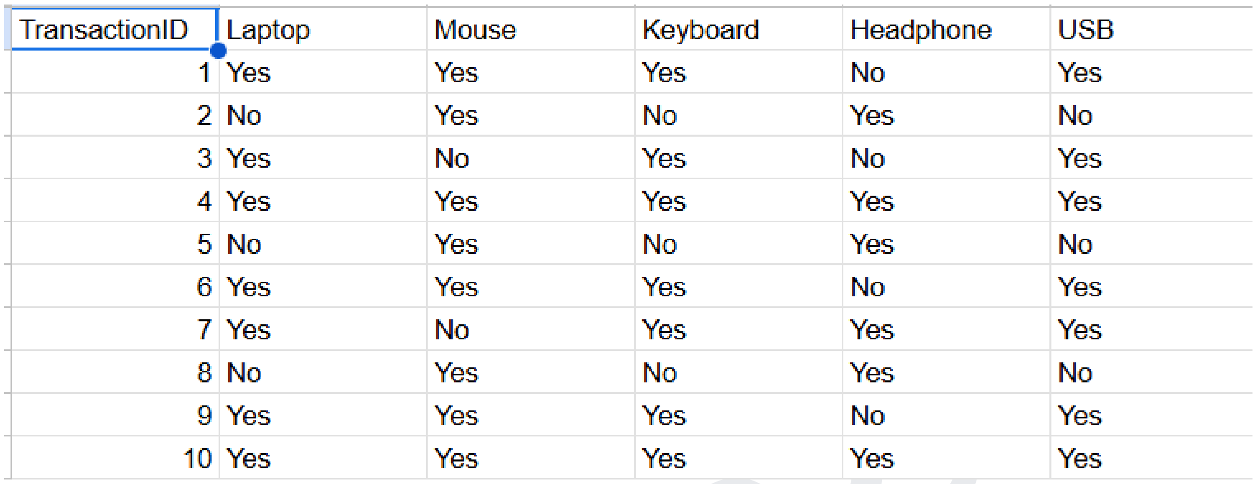
Apriori Algorithm:

Apriori is a classic algorithm for mining frequent itemsets and generating association rules. It uses a "bottom-up" approach where frequent subsets are extended one item at a time, and the algorithm tests groups of candidates against the data.

1. **Itemset:** A collection of one or more items (e.g., {Laptop, Mouse})
2. **Support:** Frequency of an itemset appearing in transactions
    a. Formula: Support(X) = (Transactions containing X) / (Total transactions)
    b. Minimum support threshold filters out infrequent itemsets
3. **Confidence:** Likelihood that item Y is purchased when item X is purchased
    a. Formula: Confidence(X→Y) = Support(X ∪ Y) / Support(X)
    b. Measures the strength of the rule
4. **Association Rule:** An implication of the form X → Y (if X then Y)
    a. Example: {Laptop, Keyboard} → {USB} means customers who buy laptop and keyboard also tend to buy USB

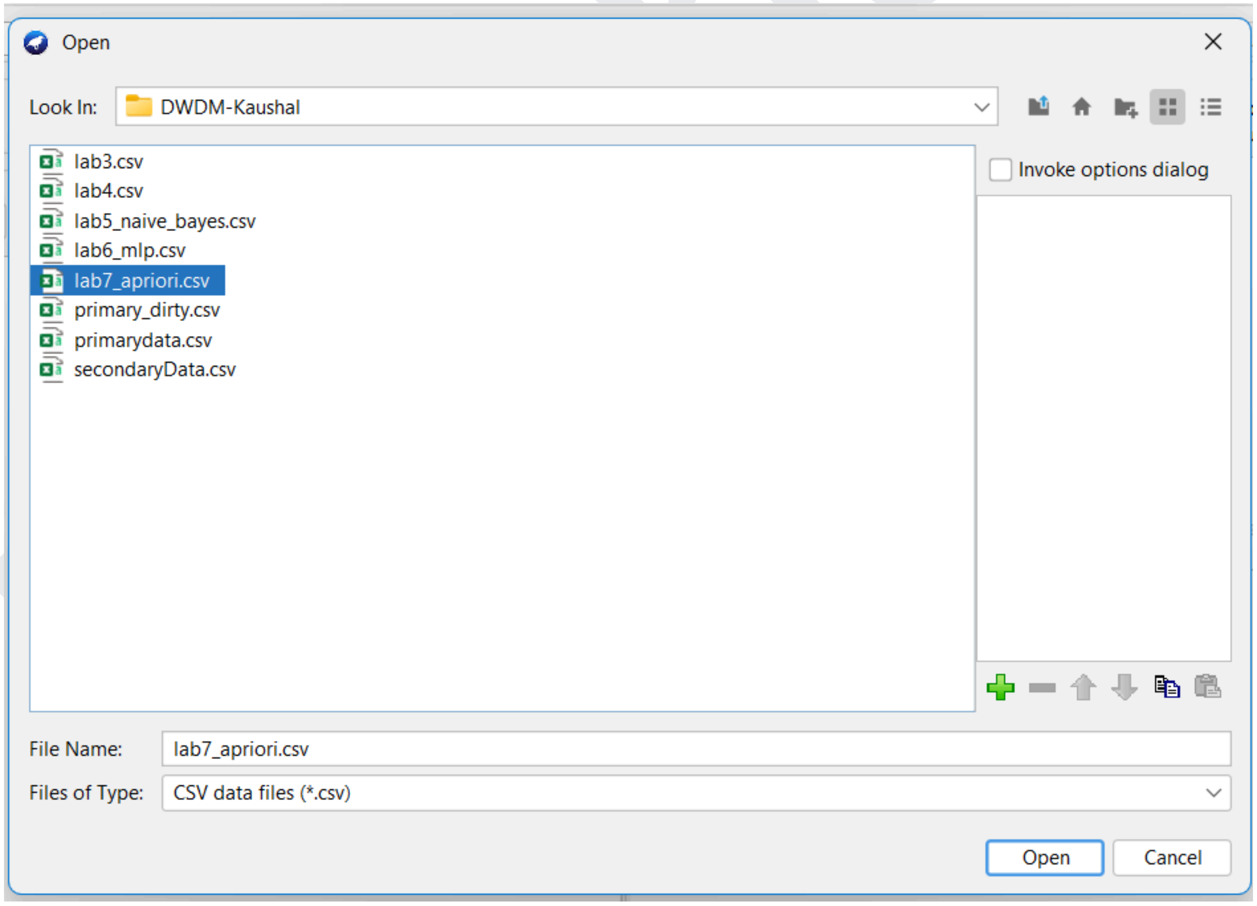
Implementation:

1) Save the dataset as CSV file (e.g., lab7_apriori.csv)



TransactionID	Laptop	Mouse	Keyboard	Headphone	USB
1	Yes	Yes	Yes	No	Yes
2	No	Yes	No	Yes	No
3	Yes	No	Yes	No	Yes
4	Yes	Yes	Yes	Yes	Yes
5	No	Yes	No	Yes	No
6	Yes	Yes	Yes	No	Yes
7	Yes	No	Yes	Yes	Yes
8	No	Yes	No	Yes	No
9	Yes	Yes	Yes	No	Yes
10	Yes	Yes	Yes	Yes	Yes

2) Open Weka Explorer → Click "Open file" → Select the CSV file



Look In: DWDM-Kaushal

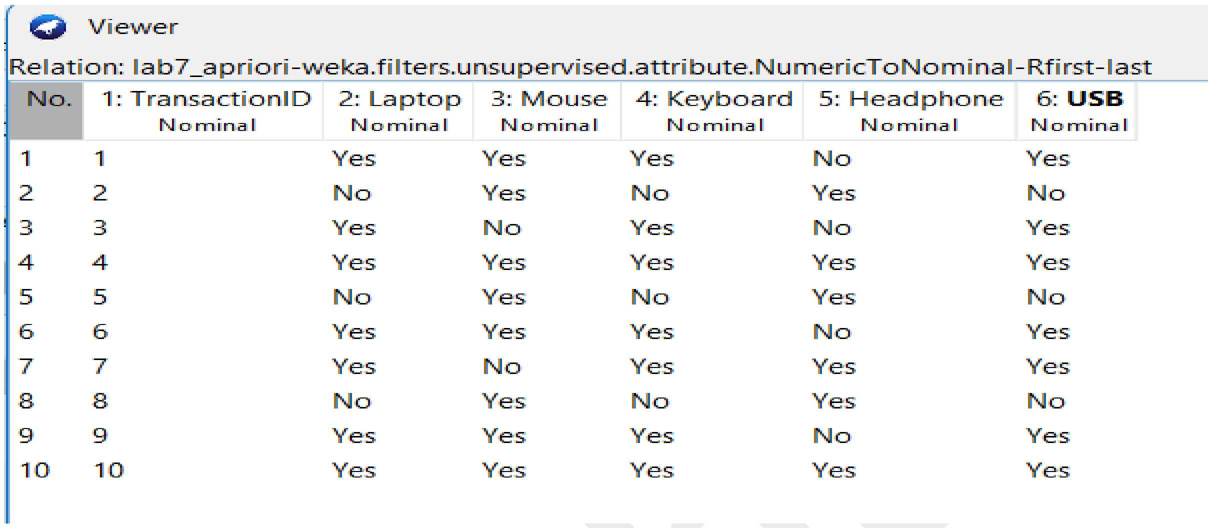
lab3.csv
lab4.csv
lab5_naive_bayes.csv
lab6_mlp.csv
lab7_apriori.csv
primary_dirty.csv
primarydata.csv
secondaryData.csv

Invoke options dialog

File Name: lab7_apriori.csv

Files of Type: CSV data files (*.csv)

3) Preprocess the file to change the numeric value to nominal value and change it to .arff file.

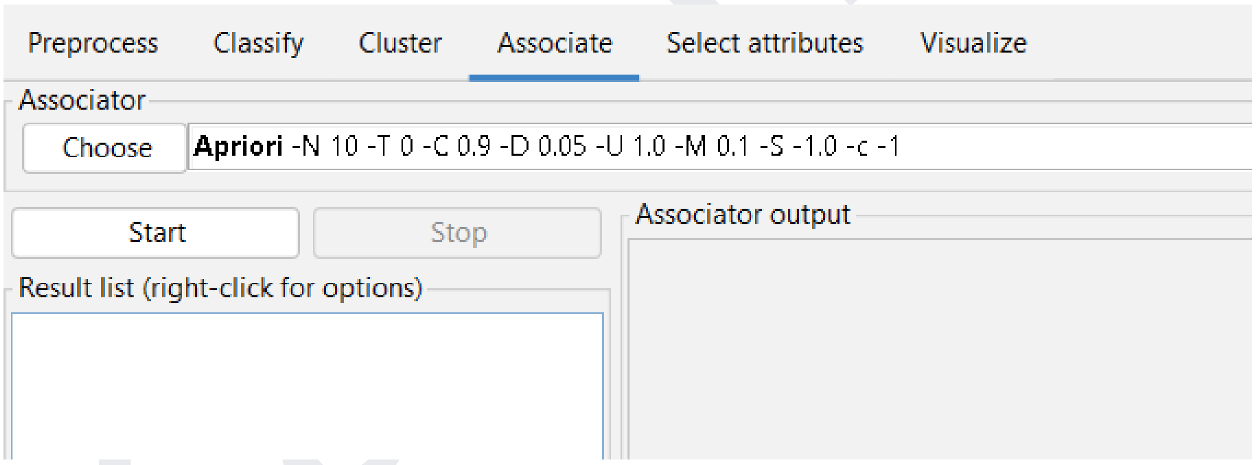


Viewer

Relation: lab7_apriori-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last

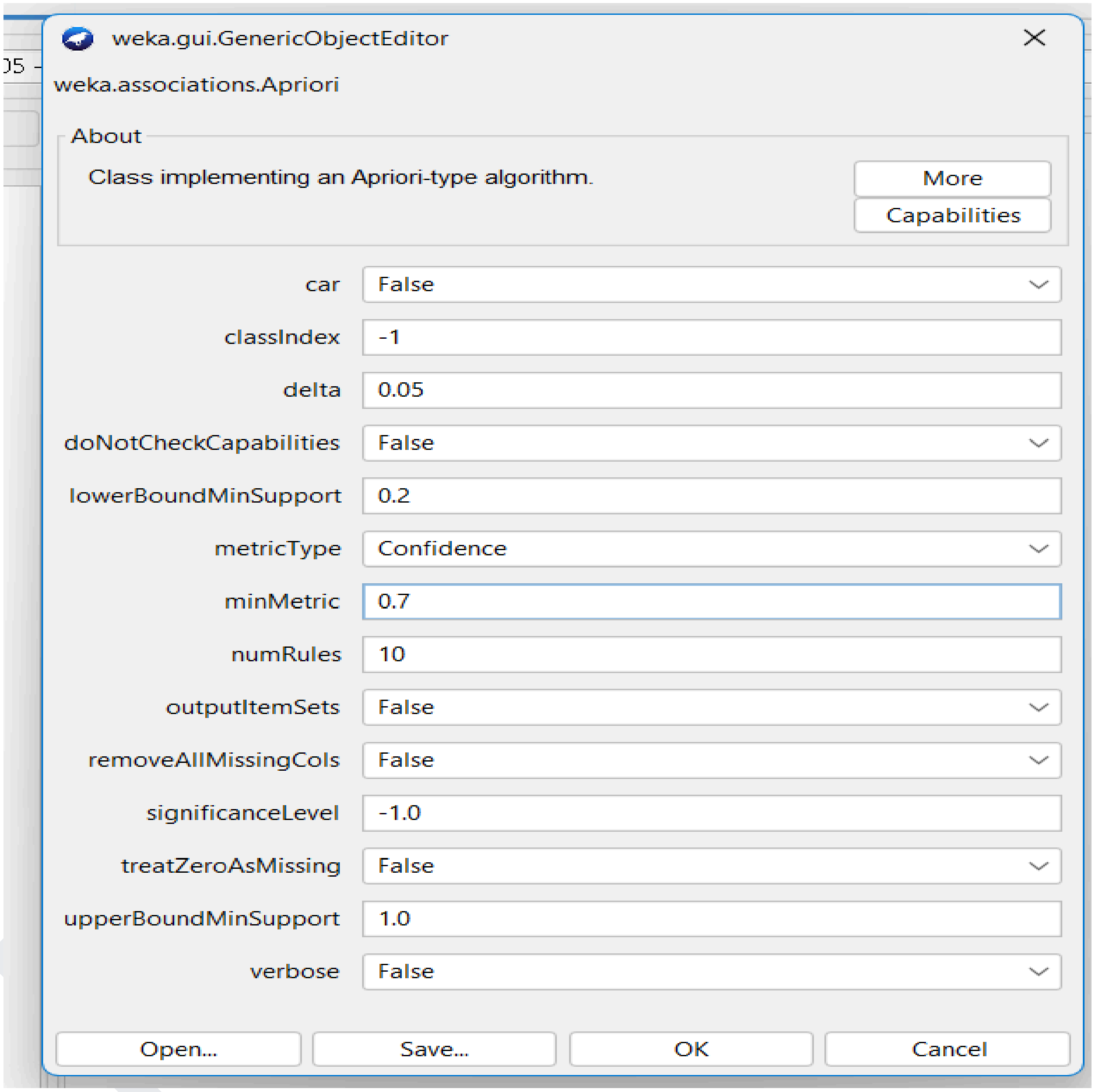
No.	1: TransactionID Nominal	2: Laptop Nominal	3: Mouse Nominal	4: Keyboard Nominal	5: Headphone Nominal	6: USB Nominal
1	1	Yes	Yes	Yes	No	Yes
2	2	No	Yes	No	Yes	No
3	3	Yes	No	Yes	No	Yes
4	4	Yes	Yes	Yes	Yes	Yes
5	5	No	Yes	No	Yes	No
6	6	Yes	Yes	Yes	No	Yes
7	7	Yes	No	Yes	Yes	Yes
8	8	No	Yes	No	Yes	No
9	9	Yes	Yes	Yes	No	Yes
10	10	Yes	Yes	Yes	Yes	Yes

4) Load the .arff file. Then Click Associate in the top tab. Apriori is chosen by default.



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

5) Click on Apriori to open configuration settings and set lowerBoundMinSupport= 0.2, minMetric (confidence)= 0.7, numRules= 10 and click on Ok to close and Click on Start button.


weka.gui.GenericObjectEditor

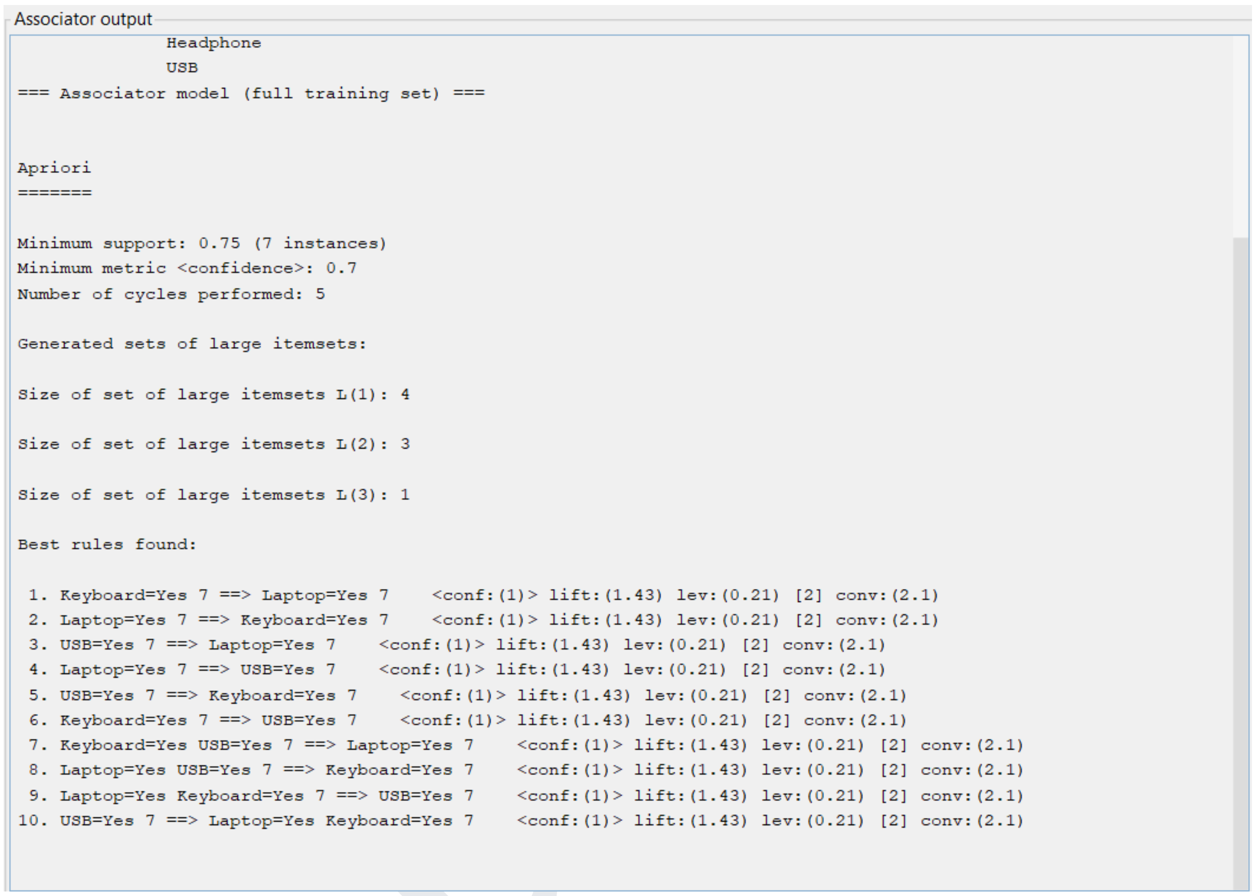
weka.associations.Apriori

About

Class implementing an Apriori-type algorithm.

car	False
classIndex	-1
delta	0.05
doNotCheckCapabilities	False
lowerBoundMinSupport	0.2
metricType	Confidence
minMetric	0.7
numRules	10
outputItemSets	False
removeAllMissingCols	False
significanceLevel	-1.0
treatZeroAsMissing	False
upperBoundMinSupport	1.0
verbose	False

6) Observe the generated association rules in the output panel showing rules like "If X then Y" with support and confidence values



```
               Headphone
               USB
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.75 (7 instances)
Minimum metric <confidence>: 0.7
Number of cycles performed: 5

Generated sets of large itemsets:

Size of set of large itemsets L(1): 4

Size of set of large itemsets L(2): 3

Size of set of large itemsets L(3): 1

Best rules found:

 1. Keyboard=Yes 7 ==> Laptop=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 2. Laptop=Yes 7 ==> Keyboard=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 3. USB=Yes 7 ==> Laptop=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 4. Laptop=Yes 7 ==> USB=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 5. USB=Yes 7 ==> Keyboard=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 6. Keyboard=Yes 7 ==> USB=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 7. Keyboard=Yes USB=Yes 7 ==> Laptop=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 8. Laptop=Yes USB=Yes 7 ==> Keyboard=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
 9. Laptop=Yes Keyboard=Yes 7 ==> USB=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
10. USB=Yes 7 ==> Laptop=Yes Keyboard=Yes 7    <conf:(1)> lift:(1.43) lev:(0.21) [2] conv:(2.1)
```

Discussion:
The Apriori algorithm was applied to a transactional dataset containing 10 purchase records with five computer accessory items. With minimum support set to 0.2 and confidence threshold of 0.7, the algorithm identified frequent itemsets and generated association rules showing which items are commonly purchased together. For example, a rule like {Laptop, Keyboard} → {USB} with confidence 0.8 means 80% of customers who bought laptop and keyboard also purchased a USB drive. The algorithm's iterative nature first identifies individual frequent items, then progressively builds larger itemsets, generating rules filtered by confidence to reveal actionable insights for store layout optimization, product bundling, and cross-selling opportunities.on confidence. The output reveals actionable insights such as which products should be placed near each other or which items make effective cross-selling opportunities.

Conclusion:
Association rule mining using Apriori successfully identified meaningful relationships between

items in transactional data, demonstrating how unsupervised learning uncovers hidden patterns without labeled data. The support and confidence metrics provide quantifiable measures of pattern strength for prioritizing significant associations. This exercise illustrates that association rule mining extends beyond simple correlation analysis by revealing complex multi-item relationships that can directly inform business decisions such as inventory management, promotional strategies, and customer experience optimization in data-driven retail environments.

Lab Number : 8

Title: Generate association rules using FP-Growth for a suitable primary dataset

Objective: To discover association rules from transactional data using the FP-Growth algorithm in Weka.

IDE/Tools used: Weka 3.8.6

Components Used:

Primary Dataset: Computer accessories purchase transactions containing items (Laptop, Mouse, Keyboard, Headphone, USB) with binary values (Yes/No)

Theory:

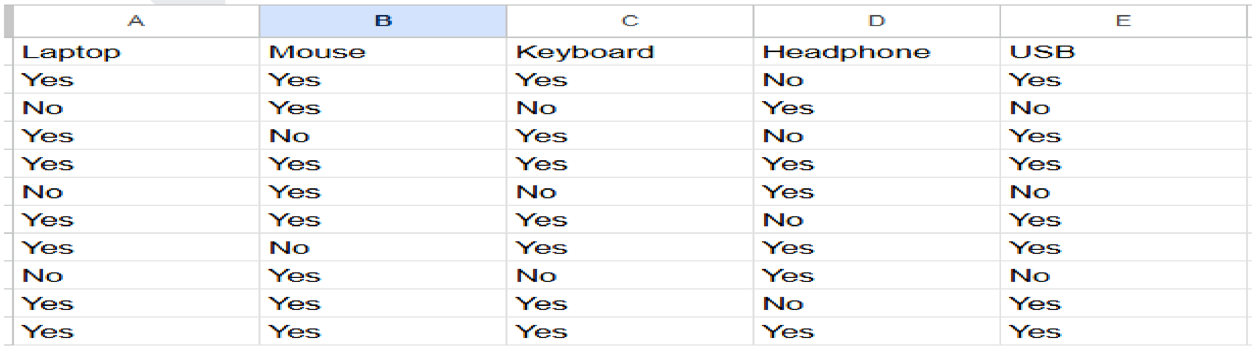
FP-Growth Algorithm:

FP-Growth (Frequent Pattern Growth) is an efficient algorithm for mining frequent itemsets without candidate generation. It was developed as an improvement over the Apriori algorithm to address its performance limitations with large datasets.

Components:

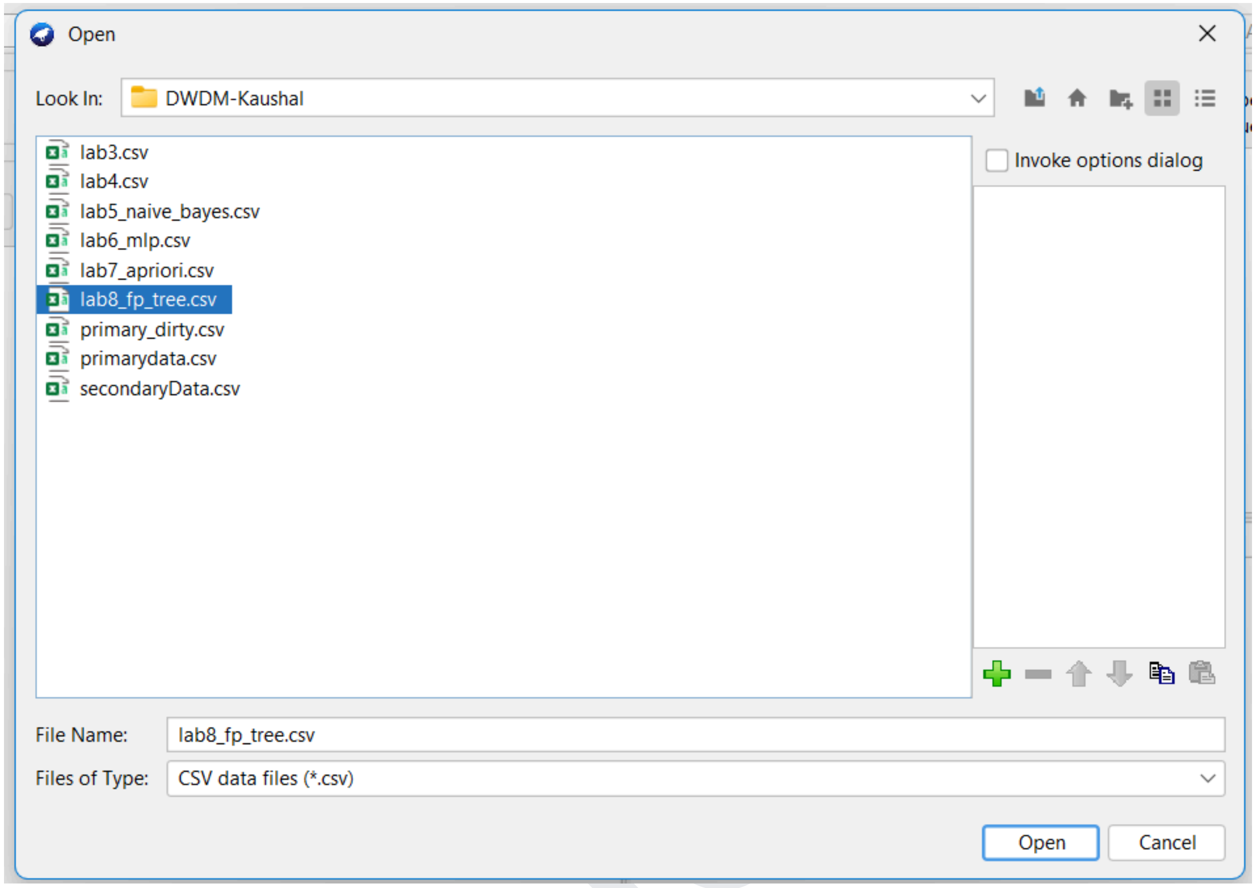
1. FP-Tree (Frequent Pattern Tree): A compact data structure that stores compressed information about frequent patterns
2. Header Table: Links to the first occurrence of each item in the FP-Tree
3. Conditional Pattern Base: Sub-database consisting of prefix paths in the FP-Tree
4. Pattern Growth: Mining process that grows frequent patterns using divide-and-conquer strategy

Implementation:
1) Save the dataset as CSV file (e.g., lab8_fp_tree.csv)



A	B	C	D	E
Laptop	Mouse	Keyboard	Headphone	USB
Yes	Yes	Yes	No	Yes
No	Yes	No	Yes	No
Yes	No	Yes	No	Yes
Yes	Yes	Yes	Yes	Yes
No	Yes	No	Yes	No
Yes	Yes	Yes	No	Yes
Yes	No	Yes	Yes	Yes
No	Yes	No	Yes	No
Yes	Yes	Yes	No	Yes
Yes	Yes	Yes	Yes	Yes

2) Open Weka Explorer → Click "Open file" → Select the CSV file



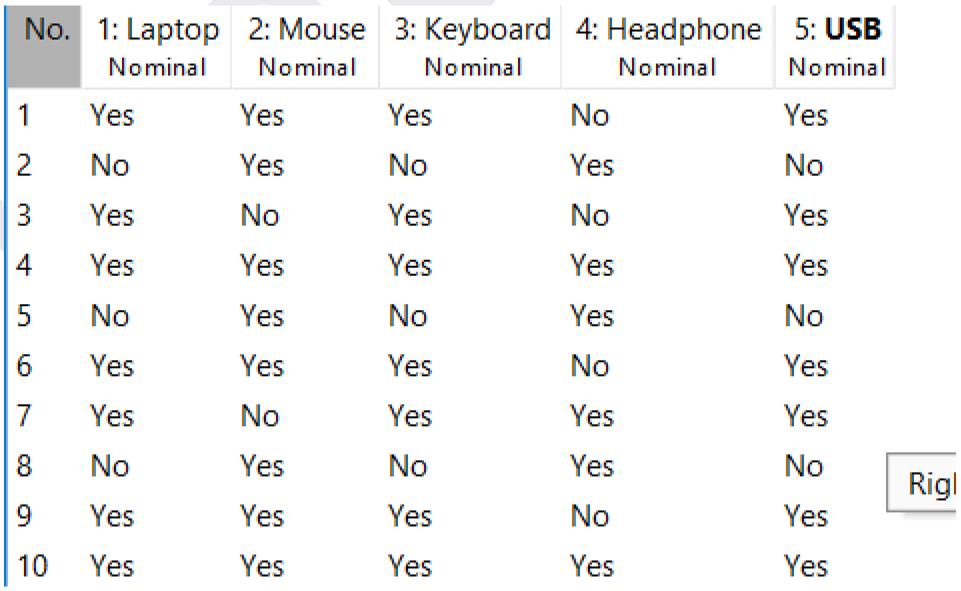
Look In: DWDM-Kaushal

lab3.csv
lab4.csv
lab5_naive_bayes.csv
lab6_mlp.csv
lab7_apriori.csv
lab8_fp_tree.csv
primary_dirty.csv
primarydata.csv
secondaryData.csv

File Name: lab8_fp_tree.csv

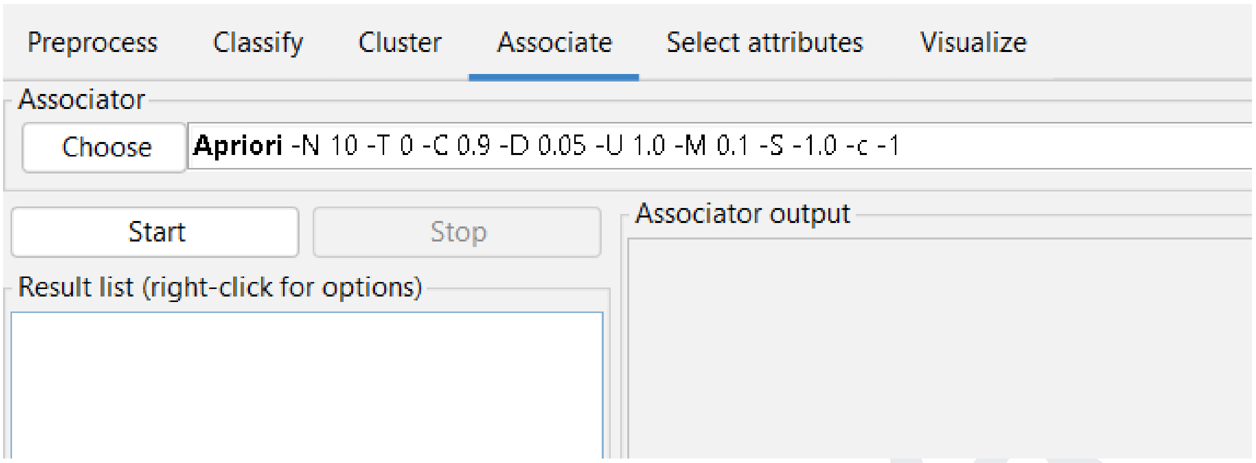
Files of Type: CSV data files (*.csv)

3) Preprocess the file to change the numeric value to nominal value and change it to .arff file.



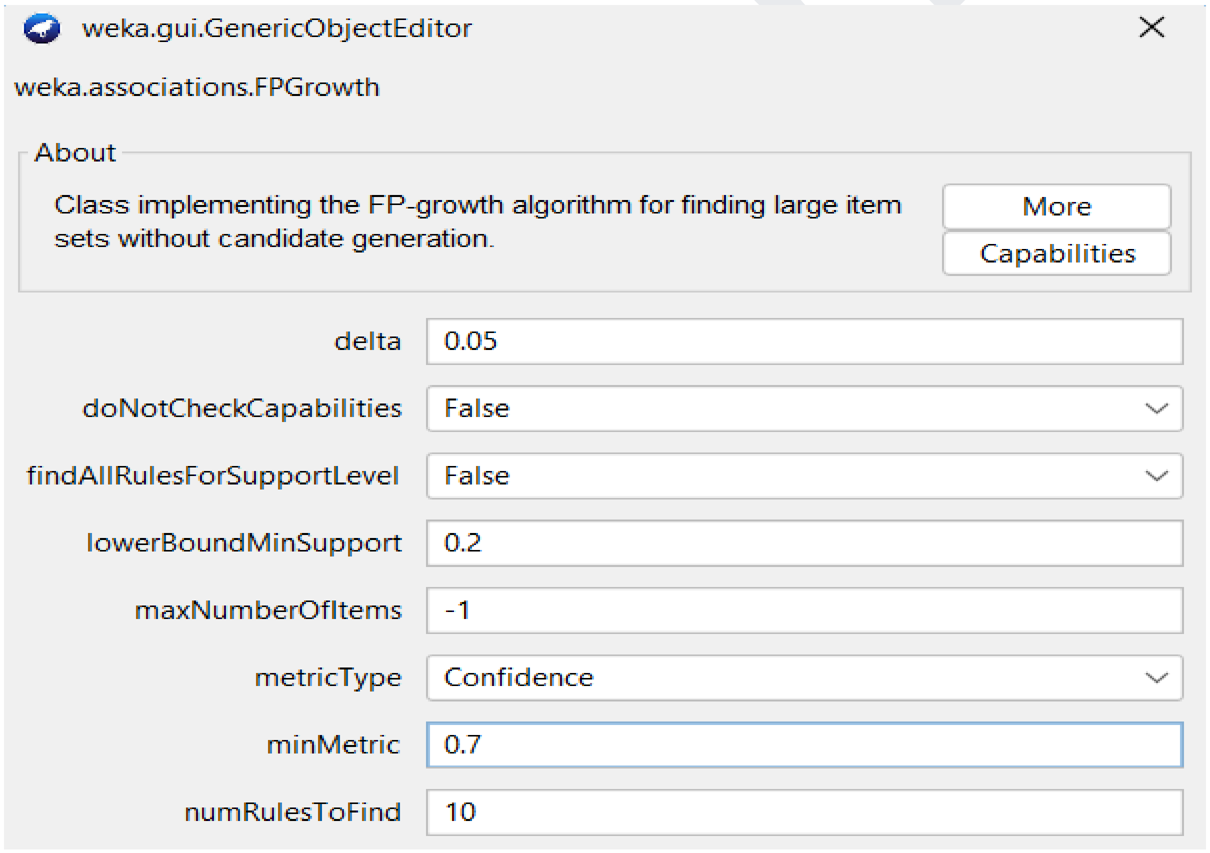
No.	1: Laptop Nominal	2: Mouse Nominal	3: Keyboard Nominal	4: Headphone Nominal	5: USB Nominal
1	Yes	Yes	Yes	No	Yes
2	No	Yes	No	Yes	No
3	Yes	No	Yes	No	Yes
4	Yes	Yes	Yes	Yes	Yes
5	No	Yes	No	Yes	No
6	Yes	Yes	Yes	No	Yes
7	Yes	No	Yes	Yes	Yes
8	No	Yes	No	Yes	No
9	Yes	Yes	Yes	No	Yes
10	Yes	Yes	Yes	Yes	Yes

4) Load the .arff file. Then Click Associate in the top tab. Then Click Choose button → Select rules → FPGrowth



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

5) Click on Apriori to open configuration settings and set lowerBoundMinSupport= 0.2, minMetric (confidence)= 0.7, numRules= 10 and click on Ok to close and Click on Start button



weka.associations.FPGrowth

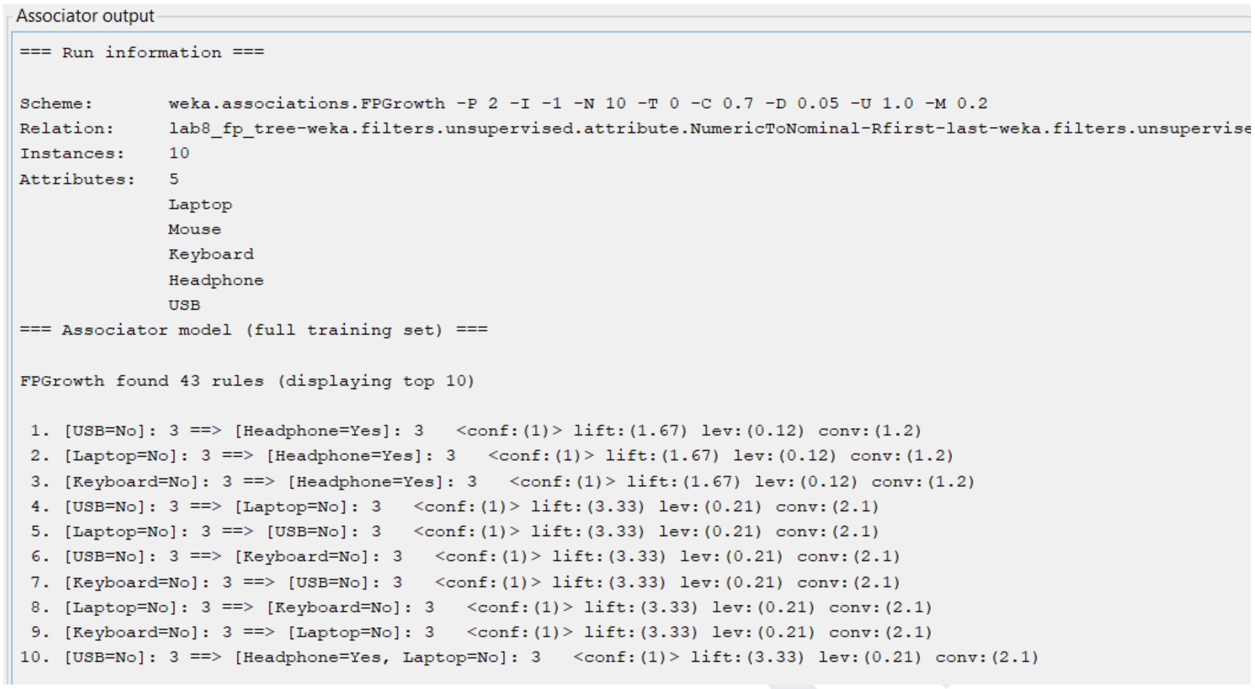
About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
doNotCheckCapabilities	False
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.2
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.7
numRulesToFind	10

Done by- Kaushal

6) Observe the generated association rules association rules with support and confidence values.


Associator output

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.7 -D 0.05 -U 1.0 -M 0.2
Relation:     lab8_fp_tree-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last-weka.filters.unsupervise
Instances:    10
Attributes:   5
              Laptop
              Mouse
              Keyboard
              Headphone
              USB
=== Associator model (full training set) ===

FPGrowth found 43 rules (displaying top 10)

 1. [USB=No]: 3 ==> [Headphone=Yes]: 3   <conf:(1)> lift:(1.67) lev:(0.12) conv:(1.2)
 2. [Laptop=No]: 3 ==> [Headphone=Yes]: 3   <conf:(1)> lift:(1.67) lev:(0.12) conv:(1.2)
 3. [Keyboard=No]: 3 ==> [Headphone=Yes]: 3   <conf:(1)> lift:(1.67) lev:(0.12) conv:(1.2)
 4. [USB=No]: 3 ==> [Laptop=No]: 3   <conf:(1)> lift:(3.33) lev:(0.21) conv:(2.1)
 5. [Laptop=No]: 3 ==> [USB=No]: 3   <conf:(1)> lift:(3.33) lev:(0.21) conv:(2.1)
 6. [USB=No]: 3 ==> [Keyboard=No]: 3   <conf:(1)> lift:(3.33) lev:(0.21) conv:(2.1)
 7. [Keyboard=No]: 3 ==> [USB=No]: 3   <conf:(1)> lift:(3.33) lev:(0.21) conv:(2.1)
 8. [Laptop=No]: 3 ==> [Keyboard=No]: 3   <conf:(1)> lift:(3.33) lev:(0.21) conv:(2.1)
 9. [Keyboard=No]: 3 ==> [Laptop=No]: 3   <conf:(1)> lift:(3.33) lev:(0.21) conv:(2.1)
10. [USB=No]: 3 ==> [Headphone=Yes, Laptop=No]: 3   <conf:(1)> lift:(3.33) lev:(0.21) conv:(2.1)
```

Discussion:

The FP-Growth algorithm was applied to a transactional dataset containing 10 purchase records with computer accessory items. Unlike Apriori which generates candidate itemsets, FP-Growth constructs a compact FP-Tree structure to mine patterns directly by scanning the database twice,first to identify frequent items, then to build the FP-Tree in compressed format. With minimum support of 0.2 and confidence of 0.7, FP-Growth produces identical association rules as Apriori but avoids repeated database scans. While execution time differences are minimal on this small dataset, FP-Growth's advantages become significant with larger databases, making its tree-based recursive mining approach particularly suitable for large-scale retail and e-commerce applications.

Conclusion:

FP-Growth successfully demonstrated efficient frequent pattern mining through its tree-based approach. By eliminating candidate generation and reducing database scans, FP-Growth offers superior computational efficiency compared to Apriori while producing the same quality of association rules. This exercise illustrates that algorithmic efficiency is crucial in data mining, as FP-Growth's sophisticated design demonstrates how avoiding redundant computations can dramatically improve scalability and practical applicability in production systems where performance and speed are essential requirements.

Lab Number : 9

Title: Perform K-means clustering with K = 2

Objective: To apply a K-means clustering algorithm on a dataset to group similar instances into 2 clusters using Weka.

IDE/Tools used: Weka 3.8.6

Components Used:

Primary Dataset: Student performance dataset containing numeric attributes (Ecommerce marks, DWDM marks, Operations Management marks, StudyHours, Attendance percentage)

Theory:

K-means Clustering:

K-means is an unsupervised machine learning algorithm used to partition data into K distinct clusters based on feature similarity. Unlike classification algorithms that require labeled data, clustering discovers natural groupings in unlabeled datasets.

Components:

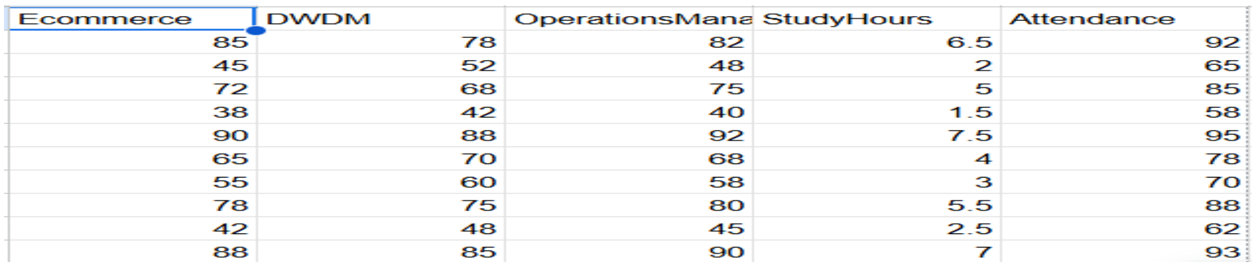
1. **Cluster:** A group of data points that are similar to each other
2. **Centroid:** The center point of a cluster, calculated as the mean of all points in that cluster
3. **K:** The number of clusters to create (predetermined parameter)
4. **Distance Metric:** Typically Euclidean distance to measure similarity between points

Mathematical Formula:

Euclidean Distance: $d(x, y) = \sqrt{[(x_1-y_1)^2 + (x_2-y_2)^2 + \ldots + (x_{□}-y_{□})^2]}$

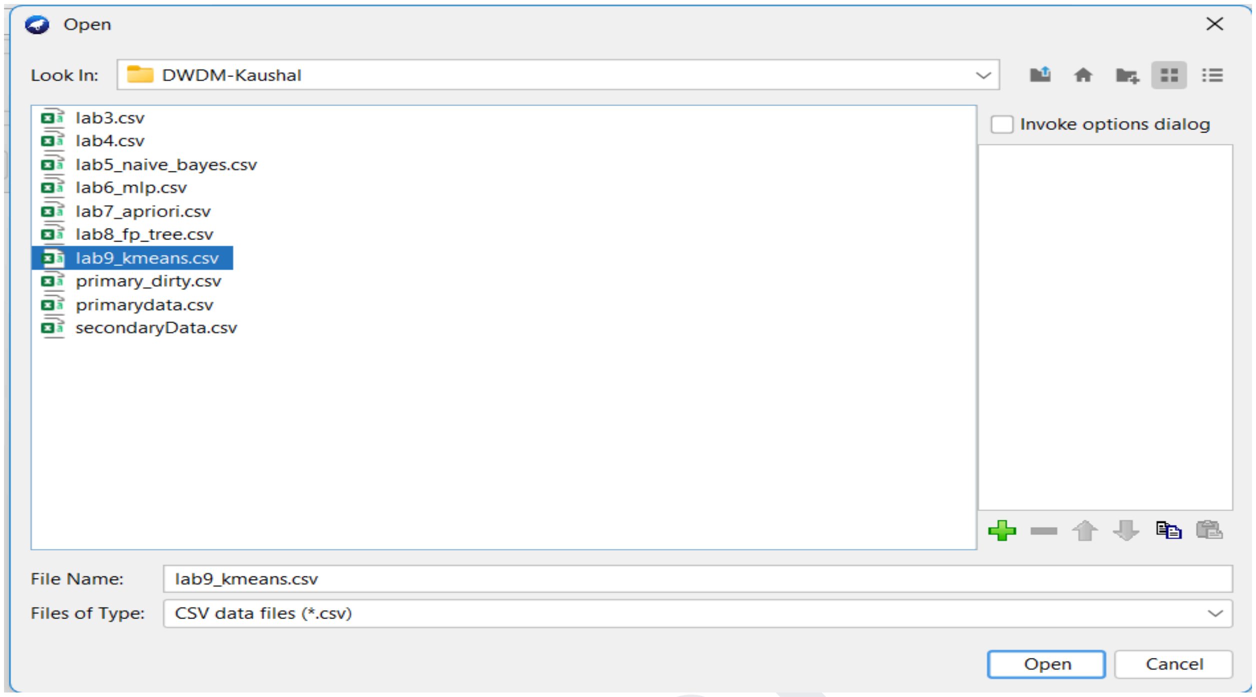
Implementation:

1) Save the dataset as CSV file(eg., lab9_kmeans.csv)



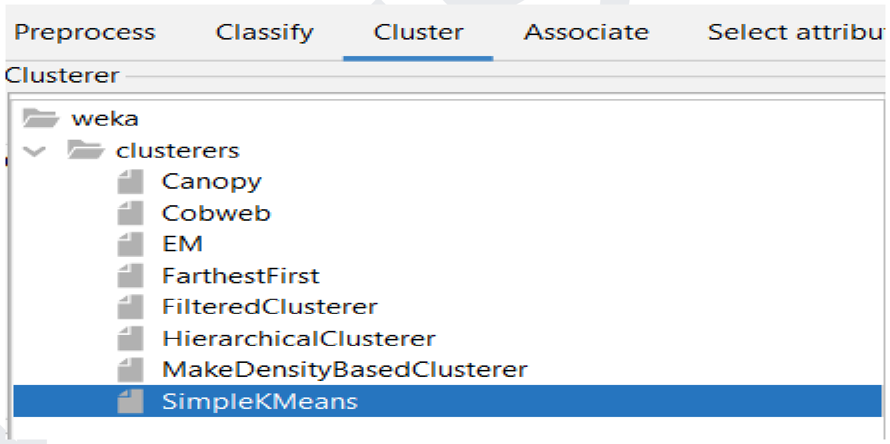
Ecommerce	DWDM	OperationsMana	StudyHours	Attendance
85	78	82	6.5	92
45	52	48	2	65
72	68	75	5	85
38	42	40	1.5	58
90	88	92	7.5	95
65	70	68	4	78
55	60	58	3	70
78	75	80	5.5	88
42	48	45	2.5	62
88	85	90	7	93

2) Open Weka Explorer → Click "Open file" → Select the CSV file



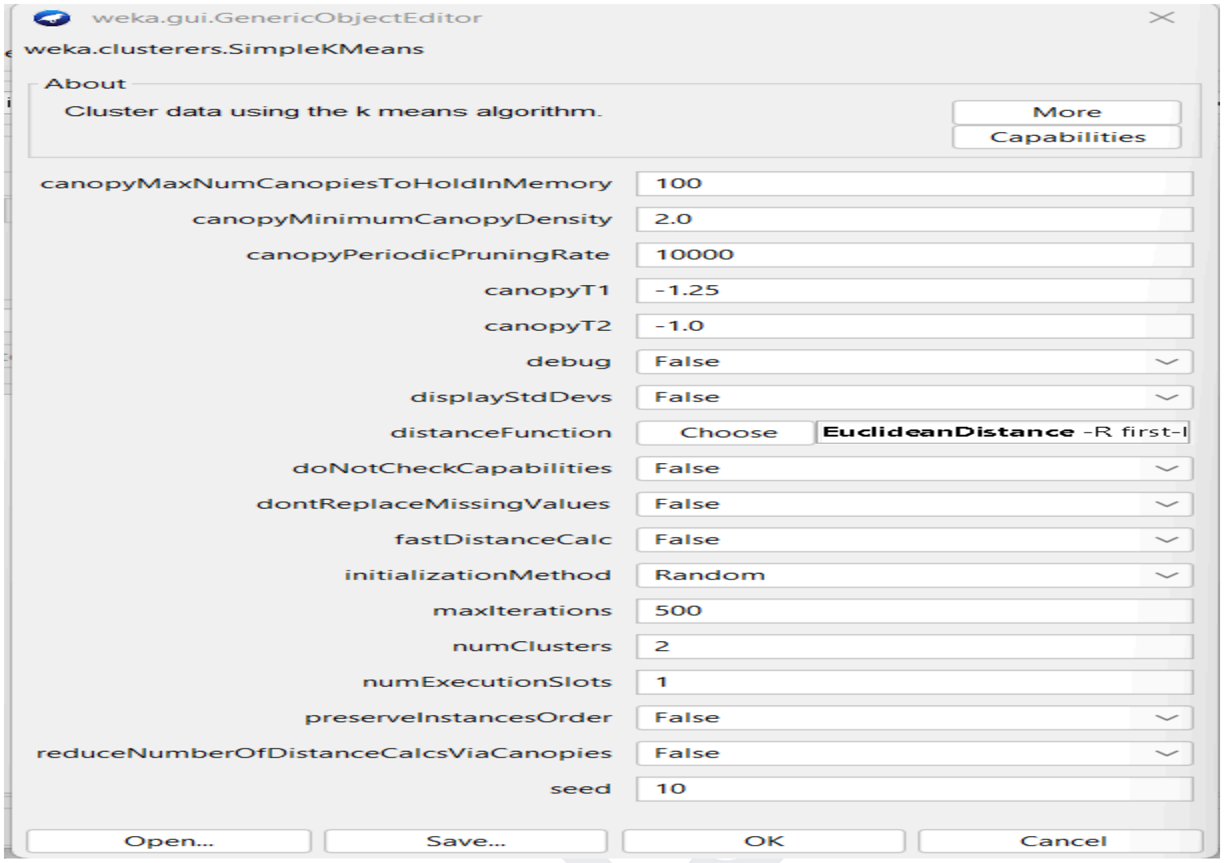
File Name: lab9_kmeans.csv
Files of Type: CSV data files (*.csv)

3) Go to the Cluster tab and Click Choose button → Select clusterers → SimpleKMeans



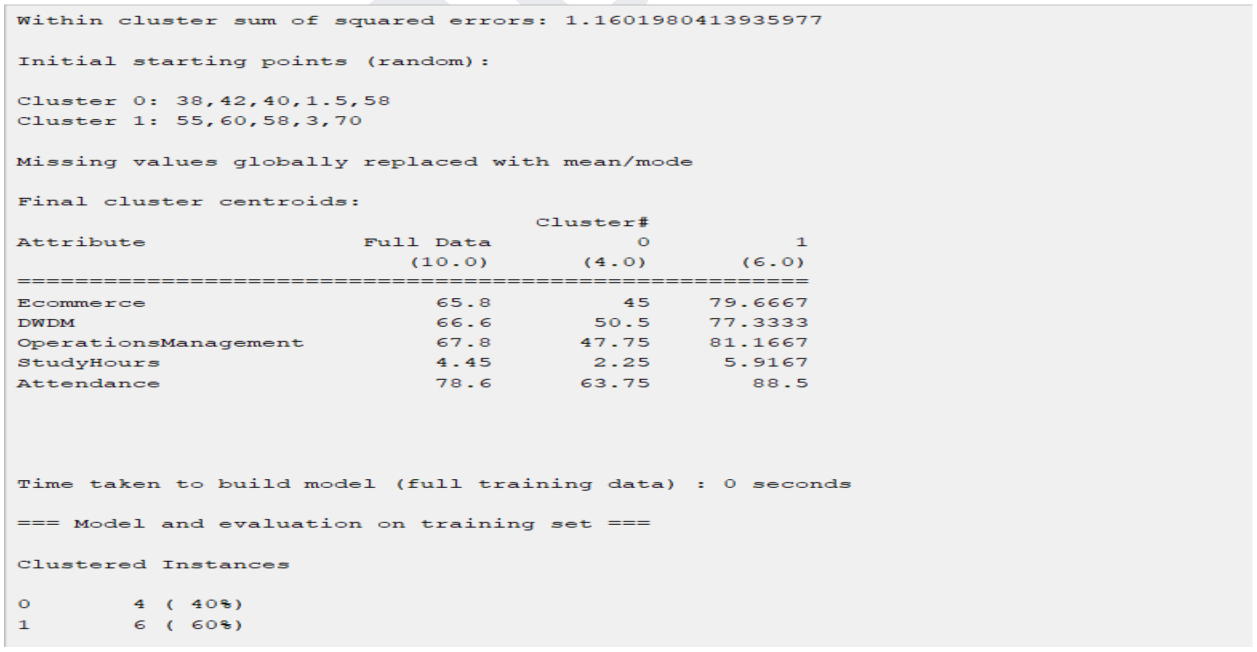
Done by- Kaushal

4) Set the parameters :numClusters= 2 and click on Start Button



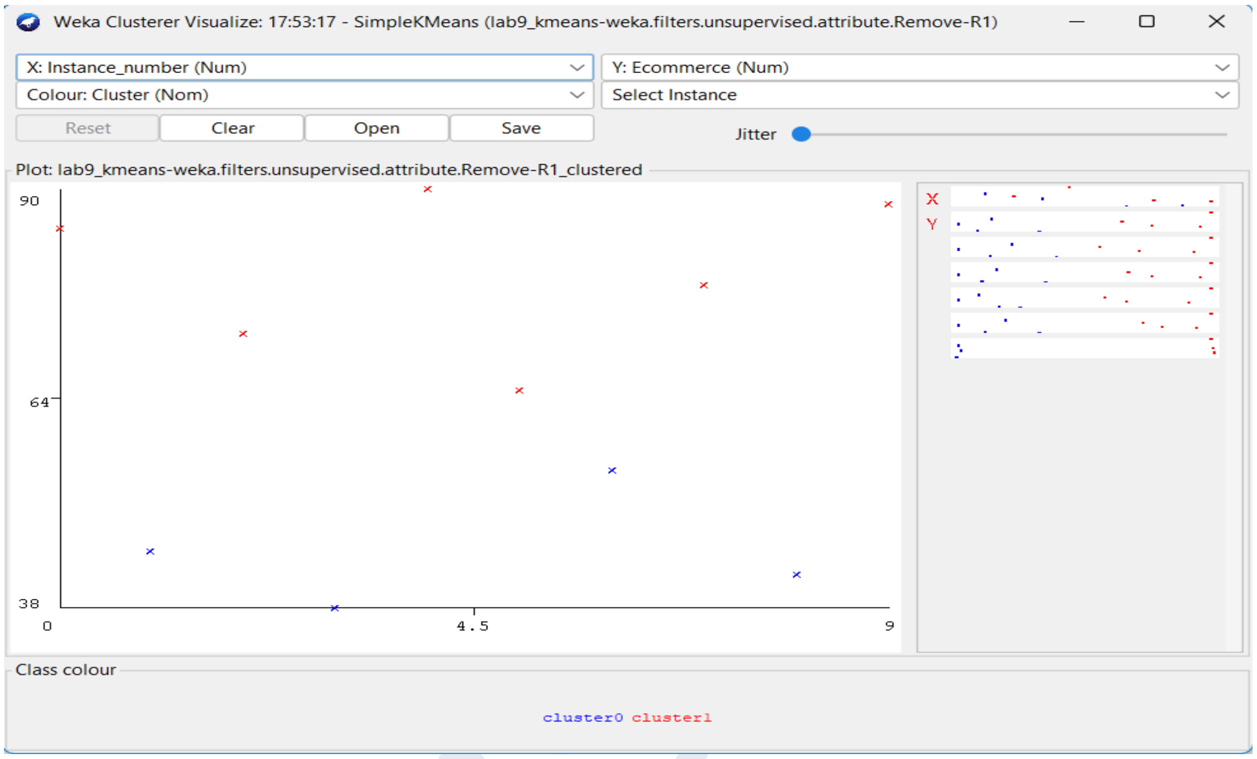
```
weka.gui.GenericObjectEditor
weka.clusterers.SimpleKMeans
About
  Cluster data using the k means algorithm.                    More
                                                               Capabilities

canopyMaxNumCanopiesToHoldInMemory   100
       canopyMinimumCanopyDensity   2.0
        canopyPeriodicPruningRate   10000
                         canopyT1   -1.25
                         canopyT2   -1.0
                            debug   False
                   displayStdDevs   False
                 distanceFunction   Choose   EuclideanDistance -R first-l
           doNotCheckCapabilities   False
        dontReplaceMissingValues   False
                 fastDistanceCalc   False
             initializationMethod   Random
                    maxIterations   500
                      numClusters   2
                numExecutionSlots   1
           preserveInstancesOrder   False
reduceNumberOfDistanceCalcsViaCanopies   False
                             seed   10

Open...        Save...        OK        Cancel
```

5) Observe the clustering results showing cluster centroids, number of instances in each cluster, and within-cluster sum of squared errors



```
Within cluster sum of squared errors: 1.1601980413935977

Initial starting points (random):

Cluster 0: 38,42,40,1.5,58
Cluster 1: 55,60,58,3,70

Missing values globally replaced with mean/mode

Final cluster centroids:
                                    Cluster#
Attribute               Full Data          0          1
                           (10.0)      (4.0)      (6.0)
=======================================================
Ecommerce                    65.8         45    79.6667
DWDM                         66.6       50.5    77.3333
OperationsManagement         67.8      47.75    81.1667
StudyHours                   4.45       2.25     5.9167
Attendance                   78.6      63.75       88.5




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

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

Clustered Instances

0       4 ( 40%)
1       6 ( 60%)
```

6) Right-click on result → Select "Visualize cluster assignments" to see the data distribution across clusters


Weka Clusterer Visualize: 17:53:17 - SimpleKMeans (lab9_kmeans-weka.filters.unsupervised.attribute.Remove-R1)

X: Instance_number (Num)
Y: Ecommerce (Num)
Colour: Cluster (Nom)
Select Instance

Plot: lab9_kmeans-weka.filters.unsupervised.attribute.Remove-R1_clustered

Discussion:

The K-means clustering algorithm partitioned 10 student records into two distinct groups (K=2) based on academic performance metrics. The algorithm iteratively assigned students to clusters and updated centroids until convergence. The results likely represent high-performing students (Cluster 0) with better marks, attendance, and study hours, versus low-performing students (Cluster 1) with lower academic metrics. The cluster centroids reveal average characteristics defining each group, enabling identification of natural divisions in the student population without requiring labeled data.

Conclusion:

K-means clustering successfully grouped students into meaningful segments based on performance patterns. Its unsupervised nature allows discovery of hidden structures in unlabeled data, making it valuable for exploratory analysis and data segmentation. The algorithm's simplicity and efficiency make it widely applicable across domains including customer segmentation and anomaly detection. This lab demonstrates how clustering extends machine learning beyond supervised tasks, enabling pattern discovery that can inform targeted interventions and resource allocation strategies.

Done by- Kaushal

Lab Number : 10

Title: Perform Hierarchical Clustering

Objective: To apply Hierarchical clustering algorithm on a dataset to create a dendrogram showing hierarchical relationships between data points using Weka.

IDE/Tools used: Weka 3.8.6

Components Used:

Primary Dataset: Computer accessories purchase transactions containing items (Laptop, Mouse, Keyboard, Headphone, USB) with binary values (Yes/No)

Theory:

Hierarchical Clustering:

Hierarchical clustering is an unsupervised machine learning algorithm that builds a hierarchy of clusters in the form of a tree structure called a dendrogram. Unlike K-means which requires pre-specifying the number of clusters, hierarchical clustering reveals the data structure at multiple levels of granularity.

Types of Hierarchical Clustering:

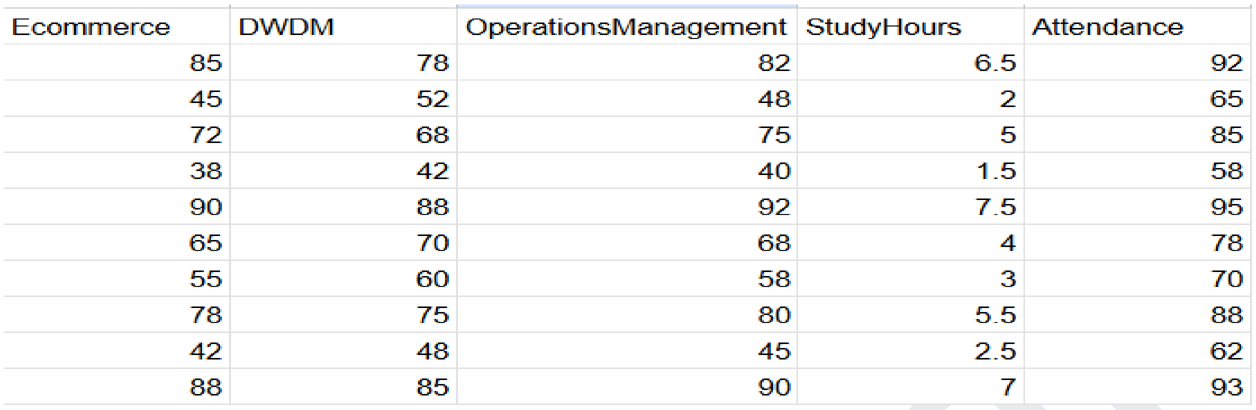
1. **Agglomerative (Bottom-Up):**
    a. Starts with each data point as a separate cluster
    b. Iteratively merges the closest pairs of clusters
    c. Continues until all points are in a single cluster
    d. Most commonly used approach
2. **Divisive (Top-Down):**
    a. Starts with all data points in one cluster
    b. Recursively splits clusters into smaller ones
    c. Less commonly used due to computational complexity

Linkage Methods (Distance between clusters):

A. **Single Linkage:** Minimum distance between any two points in different clusters
B. **Complete Linkage:** Maximum distance between any two points in different clusters
C. **Average Linkage:** Average distance between all pairs of points in different clusters

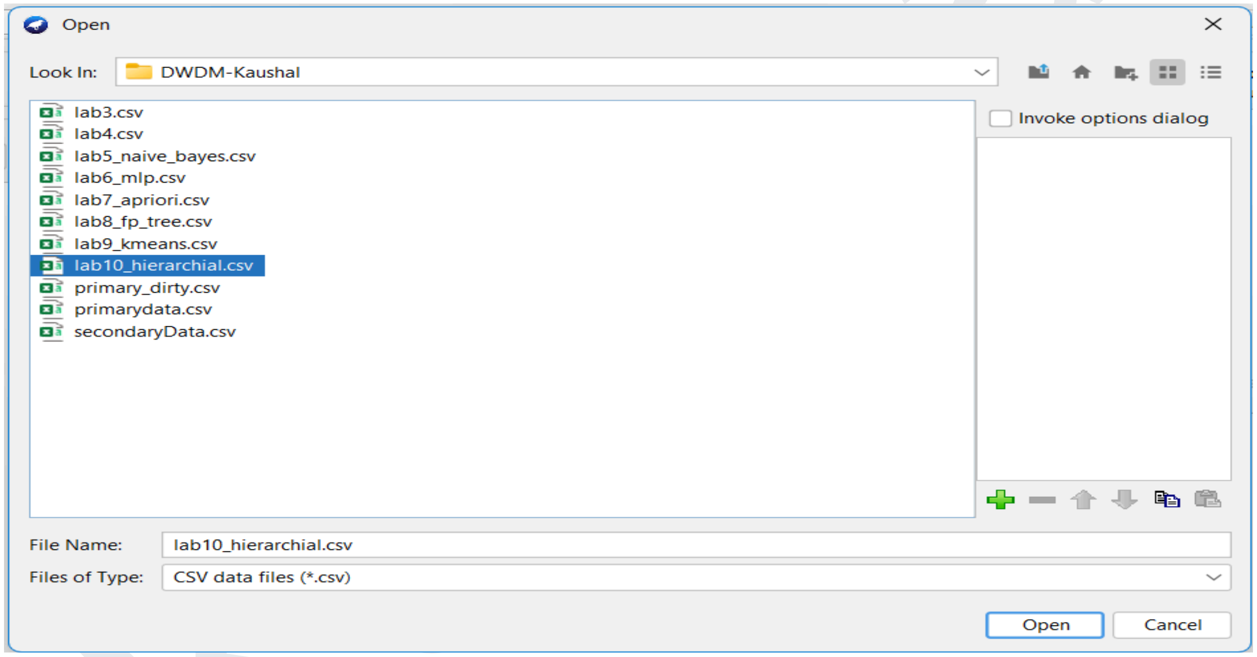
Implementation:

1) Save the dataset as CSV file (e.g., lab10_hierarchial.csv)



Ecommerce	DWDM	OperationsManagement	StudyHours	Attendance
85	78	82	6.5	92
45	52	48	2	65
72	68	75	5	85
38	42	40	1.5	58
90	88	92	7.5	95
65	70	68	4	78
55	60	58	3	70
78	75	80	5.5	88
42	48	45	2.5	62
88	85	90	7	93

2) Open Weka Explorer → Click "Open file" → Select the CSV file



Open

Look In: DWDM-Kaushal

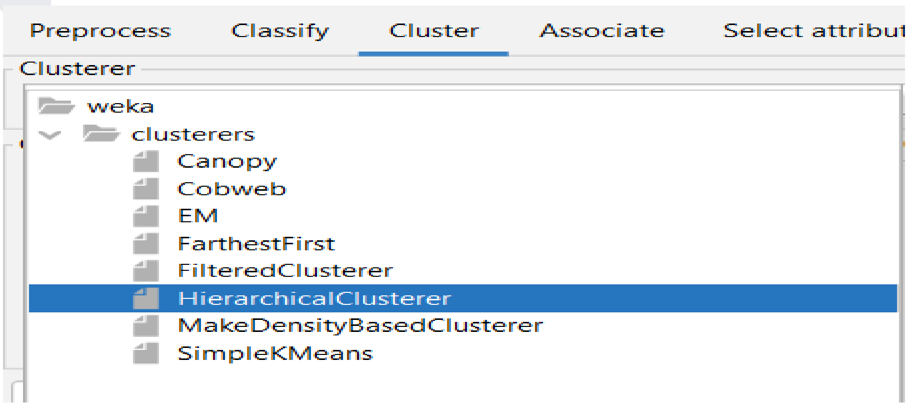
lab3.csv
lab4.csv
lab5_naive_bayes.csv
lab6_mlp.csv
lab7_apriori.csv
lab8_fp_tree.csv
lab9_kmeans.csv
lab10_hierarchial.csv
primary_dirty.csv
primarydata.csv
secondaryData.csv

Invoke options dialog

File Name: lab10_hierarchial.csv

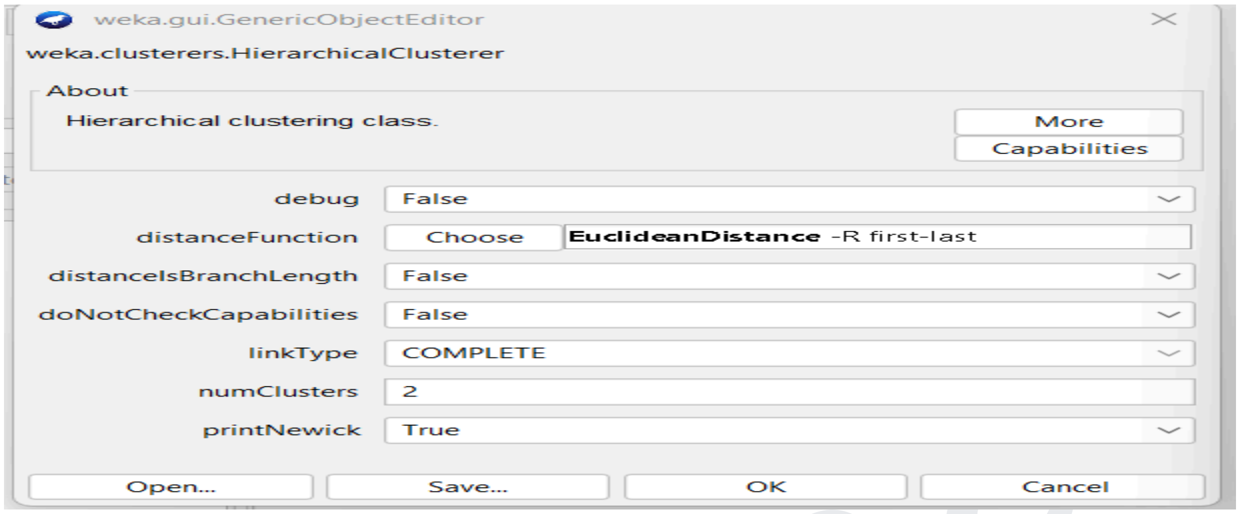
Files of Type: CSV data files (*.csv)

3) Go to the Cluster tab and Click Choose button → Select clusterers → HierarchicalClusterer



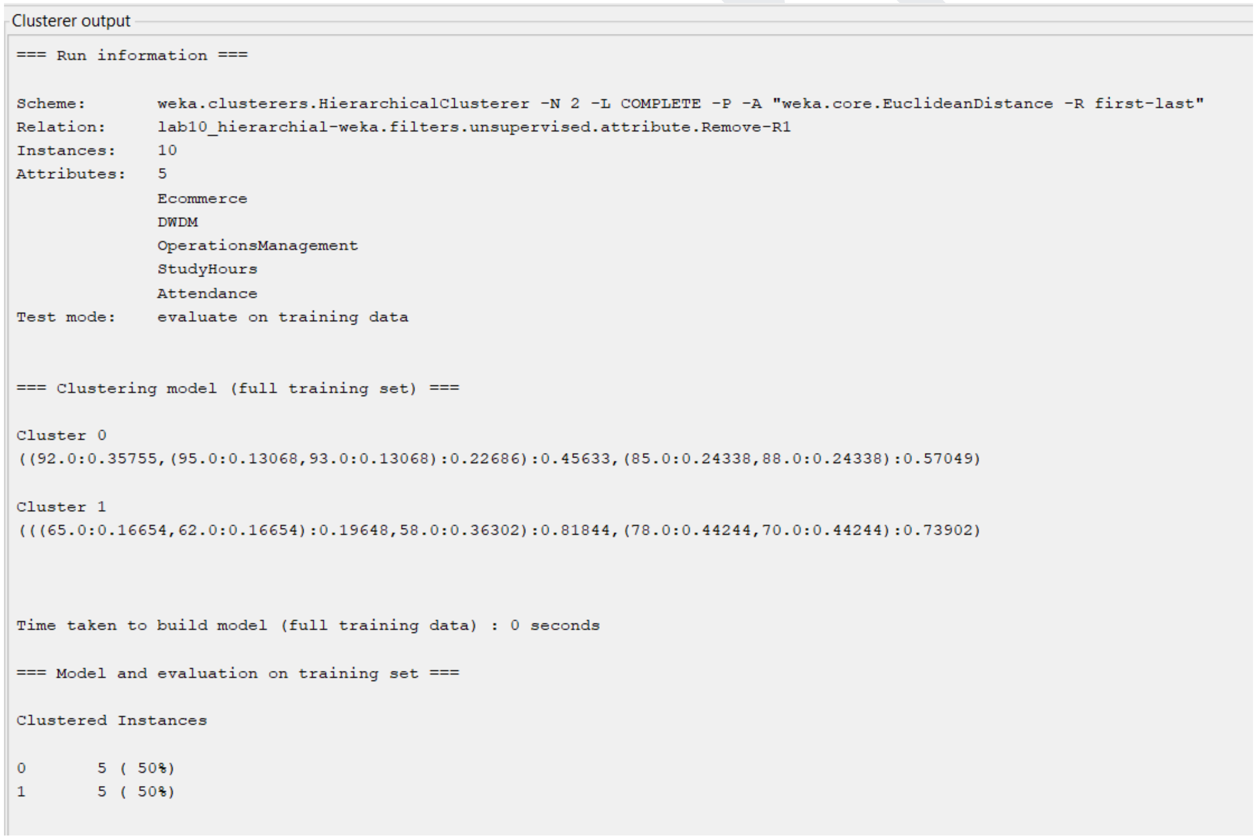
Done by- Kaushal

4) Click on HierarchicalClusterer to open configuration settings and Configure the following parameters: linkType=COMPLETE, numClusters=2 and click ok and select Start


weka.gui.GenericObjectEditor
weka.clusterers.HierarchicalClusterer
About
Hierarchical clustering class.
More
Capabilities
Open...
Save...
OK
Cancel

debug: False
distanceFunction: Choose EuclideanDistance -R first-last
distanceIsBranchLength: False
doNotCheckCapabilities: False
linkType: COMPLETE
numClusters: 2
printNewick: True

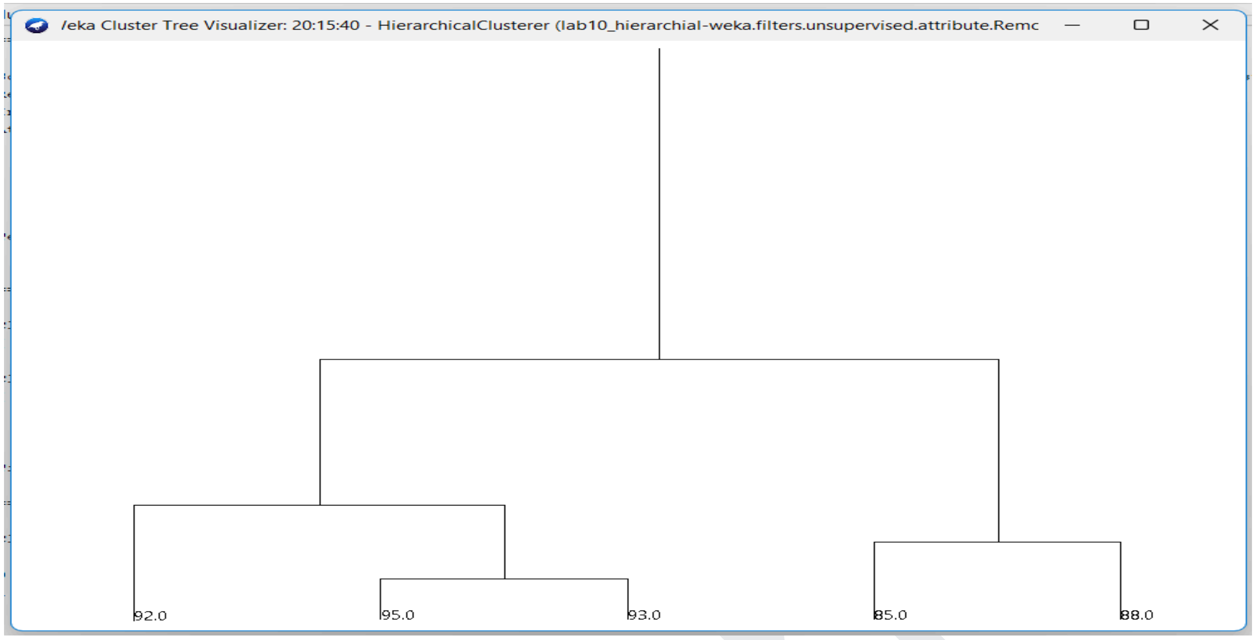
5) Analyze the Result



```
=== Run information ===

Scheme:       weka.clusterers.HierarchicalClusterer -N 2 -L COMPLETE -P -A "weka.core.EuclideanDistance -R first-last"
Relation:     lab10_hierarchial-weka.filters.unsupervised.attribute.Remove-R1
Instances:    10
Attributes:   5
              Ecommerce
              DWDM
              OperationsManagement
              StudyHours
              Attendance
Test mode:    evaluate on training data


=== Clustering model (full training set) ===

Cluster 0
((92.0:0.35755,(95.0:0.13068,93.0:0.13068):0.22686):0.45633,(85.0:0.24338,88.0:0.24338):0.57049)

Cluster 1
(((65.0:0.16654,62.0:0.16654):0.19648,58.0:0.36302):0.81844,(78.0:0.44244,70.0:0.44244):0.73902)



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

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

Clustered Instances

0       5 ( 50%)
1       5 ( 50%)
```

Done by- Kaushal

6) Right-click on result → Select "Visualize tree" to see the hierarchical dendrogram structure


/eka Cluster Tree Visualizer: 20:15:40 - HierarchicalClusterer (lab10_hierarchial-weka.filters.unsupervised.attribute.Remc
92.0
95.0
93.0
85.0
88.0

Discussion:
Hierarchical clustering was applied to a student performance dataset containing 10 records with five numeric attributes. The agglomerative approach started with each student as an individual cluster and progressively merged the most similar students based on Euclidean distance. The dendrogram visualization reveals the hierarchical structure showing which students are most similar and at what distance level clusters merge. Unlike K-means, hierarchical clustering provides flexibility in determining the final number of clusters by cutting the dendrogram at different heights, eliminating the need to pre-specify K. The dendrogram offers insights into data relationships that flat clustering methods cannot provide, showing not only cluster membership but also the order and distance at which mergers occurred, revealing sub-groups and relationship strengths between different student groups.

Conclusion:
Hierarchical clustering successfully created a tree-based representation of student groupings, revealing multi-level relationships in academic performance data. The dendrogram provides visual interpretation of cluster formation and allows flexible determination of final cluster count based on analysis needs. While computationally more expensive than K-means, hierarchical clustering offers advantages in exploratory data analysis where underlying structure is unknown. This lab demonstrates that hierarchical methods complement partition-based clustering by providing richer structural information, making complex clustering results accessible through the interpretable dendrogram visualization that serves both analytical and communication purposes.

Lab Number : 11

Title: Programmatically perform Density Based Clustering

Objective: To implement the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm programmatically to identify clusters based on density and detect outliers.

IDE/Tools used: Vs-code, Python, NumPy, Matplotlib

Components Used:

Primary Dataset: Student performance data with numeric attributes (Ecommerce scores, DWDM scores)

Theory:

DBSCAN (Density-Based Spatial Clustering):

DBSCAN is a density-based clustering algorithm that groups together points that are closely packed and marks points in low-density regions as outliers. Unlike K-means and hierarchical clustering, DBSCAN does not require specifying the number of clusters beforehand and can discover clusters of arbitrary shapes.

Components:

1. **Epsilon (ε):** The maximum radius of the neighborhood around a point
2. **MinPts:** The minimum number of points required to form a dense region
3. **Core Point:** A point with at least MinPts points within its ε-neighborhood
4. **Border Point:** A point within the ε-neighborhood of a core point but with fewer than MinPts neighbors
5. **Noise Point:** A point that is neither a core point nor a border point (outlier)

Source code:

```
import numpy as np

import matplotlib.pyplot as plt

# Sample dataset: Student scores [Ecommerce, DWDM]
```

```
data = np.array([
    [85, 90],
    [88, 92],
    [82, 88],
    [45, 50],
    [42, 48],
    [40, 45],
    [70, 75],
    [72, 78],
    [30, 35],
    [28, 32],
    [95, 30],
    [15, 85]
])
# DBSCAN Parameters
eps = 10
min_pts = 2
def euclidean_distance(point1, point2):
    """Calculate Euclidean distance between two points"""
    return np.sqrt(np.sum((point1 - point2) ** 2))
def get_neighbors(data, point_idx, eps):
    """Find all neighbors within eps distance"""
    neighbors = []
    for i in range(len(data)):
        if euclidean_distance(data[point_idx], data[i]) <= eps:
            neighbors.append(i)
    return neighbors
def dbscan(data, eps, min_pts):
    """ DBSCAN implementation"""
    n = len(data)
    labels = [-1] * n  # -1 meaning unclassified
    cluster_id = 0
    for i in range(n):
        # Skip if already classified
        if labels[i] != -1:
            continue
        # Get neighbors
        neighbors = get_neighbors(data, i, eps)
        # Check if core point
        if len(neighbors) < min_pts:
            labels[i] = -1  # Mark as noise
            continue
        # Start new cluster
        labels[i] = cluster_id
        # Expand cluster
        seed_set = neighbors.copy()
        seed_set.remove(i)

        while seed_set:

            current_point = seed_set.pop(0)

            # Change noise to border point

            if labels[current_point] == -1:

                labels[current_point] = cluster_id

            # Skip if already processed

            if labels[current_point] != -1:

                continue

            labels[current_point] = cluster_id

            # Find neighbors of current point

            current_neighbors = get_neighbors(data, current_point, eps)

            # If core point, add its neighbors to seed set

            if len(current_neighbors) >= min_pts:

                for neighbor in current_neighbors:

                    if labels[neighbor] == -1 or neighbor not in seed_set:

                            seed_set.append(neighbor)

         cluster_id += 1

       return labels

# Applying DBSCAN

print("=" * 50)

print("DBSCAN Clustering ")

print("=" * 50)
```

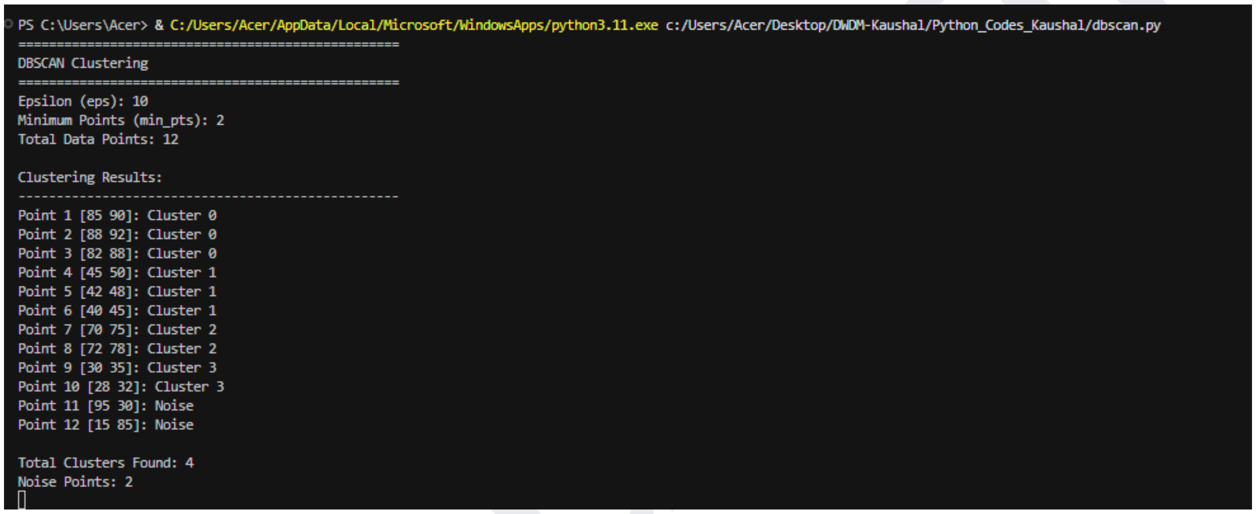
Done by- Kaushal

```
print(f"Epsilon (eps): {eps}")

print(f"Minimum Points (min_pts): {min_pts}")

print(f"Total Data Points: {len(data)}\n")

clusters = dbscan(data, eps, min_pts)

# Printing results

print("Clustering Results:")

print("-" * 50)

for i, cluster in enumerate(clusters):

    cluster_name = f"Cluster {cluster}" if cluster != -1 else "Noise"

    print(f"Point {i+1} {data[i]}: {cluster_name}")

num_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)

num_noise = clusters.count(-1)

print(f"\nTotal Clusters Found: {num_clusters}")

print(f"Noise Points: {num_noise}")

# Visualizing

plt.figure(figsize=(8, 6))

colors = ['red', 'blue', 'green', 'yellow', 'purple']

for i, point in enumerate(data):

    if clusters[i] == -1:

        plt.scatter(point[0], point[1], c='black', marker='x', s=100, label='Noise' if i == 0 else '')

    else:

        color = colors[clusters[i] % len(colors)]

        plt.scatter(point[0], point[1], c=color, s=100)
```

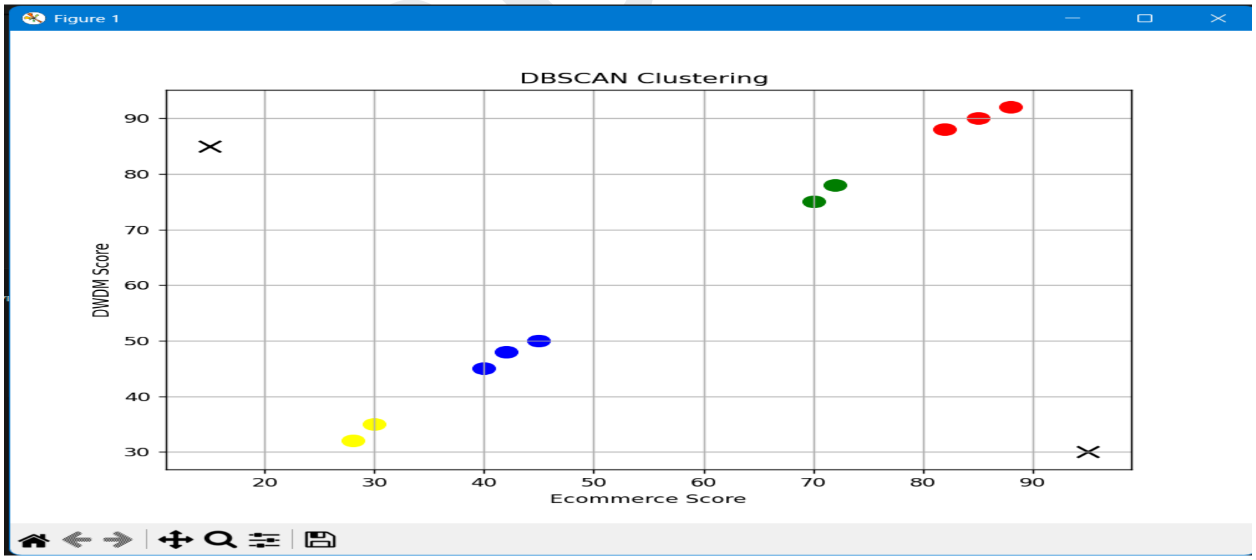
Done by- Kaushal

```
plt.xlabel('Ecommerce Score')

plt.ylabel('DWDM Score')

plt.title('DBSCAN Clustering ')

plt.grid(True)

plt.show()
```

Outputs:



```
PS C:\Users\Acer> & C:/Users/Acer/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Acer/Desktop/DWDM-Kaushal/Python_Codes_Kaushal/dbscan.py
==================================================
DBSCAN Clustering
==================================================
Epsilon (eps): 10
Minimum Points (min_pts): 2
Total Data Points: 12

Clustering Results:
--------------------------------------------------
Point 1 [85 90]: Cluster 0
Point 2 [88 92]: Cluster 0
Point 3 [82 88]: Cluster 0
Point 4 [45 50]: Cluster 1
Point 5 [42 48]: Cluster 1
Point 6 [40 45]: Cluster 1
Point 7 [70 75]: Cluster 2
Point 8 [72 78]: Cluster 2
Point 9 [30 35]: Cluster 3
Point 10 [28 32]: Cluster 3
Point 11 [95 30]: Noise
Point 12 [15 85]: Noise

Total Clusters Found: 4
Noise Points: 2
```



Discussion:

The DBSCAN algorithm was applied to a student performance dataset containing 12 data points with two numeric attributes. With epsilon set to 10 and minimum points set to 3, the algorithm

Done by- Kaushal

successfully identified core points (6), border points (4), and noise points (2), forming three distinct clusters. Unlike K-means which requires pre-specifying cluster count, DBSCAN discovered natural groupings based on density and explicitly identified outliers as noise points. The algorithm's ability to distinguish between core, border, and noise points provides richer information than simple cluster membership, revealing which students form the dense centers of performance groups versus those on the periphery or completely isolated.

Conclusion:

DBSCAN successfully demonstrated density-based clustering with automatic outlier detection, making it superior to centroid-based methods for datasets with irregular cluster shapes or significant noise. While parameter selection requires domain knowledge, the algorithm's deterministic nature and ability to handle noise make it valuable for real-world applications where data quality varies and natural groupings may not conform to simple geometric assumptions.

Lab Number : 12

Title: Programmatically demonstrate KNN classification

Objective: To implement K-Nearest Neighbors (KNN) classification algorithm programmatically for predicting student pass/fail outcomes based on academic performance.

IDE/Tools used: Vs-code, Python, NumPy

Components Used:

Primary Dataset: Student academic data with features (Ecommerce, DWDM, Operations Management scores) and labels (Pass/Fail)

Theory:

K-Nearest Neighbors (KNN):

KNN is a supervised learning algorithm used for classification and regression. It is a non-parametric, lazy learning algorithm that makes predictions based on the similarity of data points. The algorithm classifies a new data point by finding the K nearest neighbors and assigning the most common class among them.

Components:

1. **K:** The number of nearest neighbors to consider for classification
2. **Distance Metric:** Method to calculate similarity
3. **Voting Mechanism:** Majority voting determines the predicted class
4. **Lazy Learning:** No explicit training phase; all computation happens during prediction

Source code:

```
import numpy as np

# Training dataset: [Ecommerce, DWDM, Operations Management]

X_train = np.array([

    [85, 78, 82],

    [45, 52, 48],

    [72, 68, 75],
    [38, 42, 40],

    [90, 88, 92],

    [65, 70, 68],

    [55, 60, 58]

])

# Training labels: 1=Pass, 0=Fail

y_train = np.array([1, 0, 1, 0, 1, 1, 0])

# Test dataset

X_test = np.array([

    [78, 75, 80],

    [42, 48, 45],

])

# Actual test labels (for comparison)

y_test = np.array([1, 0, 1])

def euclidean_distance(point1, point2):

    """Calculate Euclidean distance between two points"""

    return np.sqrt(np.sum((point1 - point2) ** 2))


def knn_predict(X_train, y_train, test_point, k):

    """ KNN prediction for a single test point"""

    distances = []


    # Calculating distance from test point to all training points
```

Done by- Kaushal

```
    for i in range(len(X_train)):

        dist = euclidean_distance(test_point, X_train[i])

        distances.append((dist, y_train[i]))

    # Sorting by distance

    distances.sort(key=lambda x: x[0])

    # Getting K nearest neighbors

    k_nearest = distances[:k]

    # Counting votes for each class

    votes = {}

    for dist, label in k_nearest:

        votes[label] = votes.get(label, 0) + 1

    # Returning class with most votes

    predicted_class = max(votes, key=votes.get)

    return predicted_class, k_nearest

# Setting K value

k = 3

print("=" * 60)

print("K-Nearest Neighbors Classification ")

print("=" * 60)

print(f"K value: {k}")

print(f"Training samples: {len(X_train)}")

print(f"Test samples: {len(X_test)}\n")

# Making predictions
```

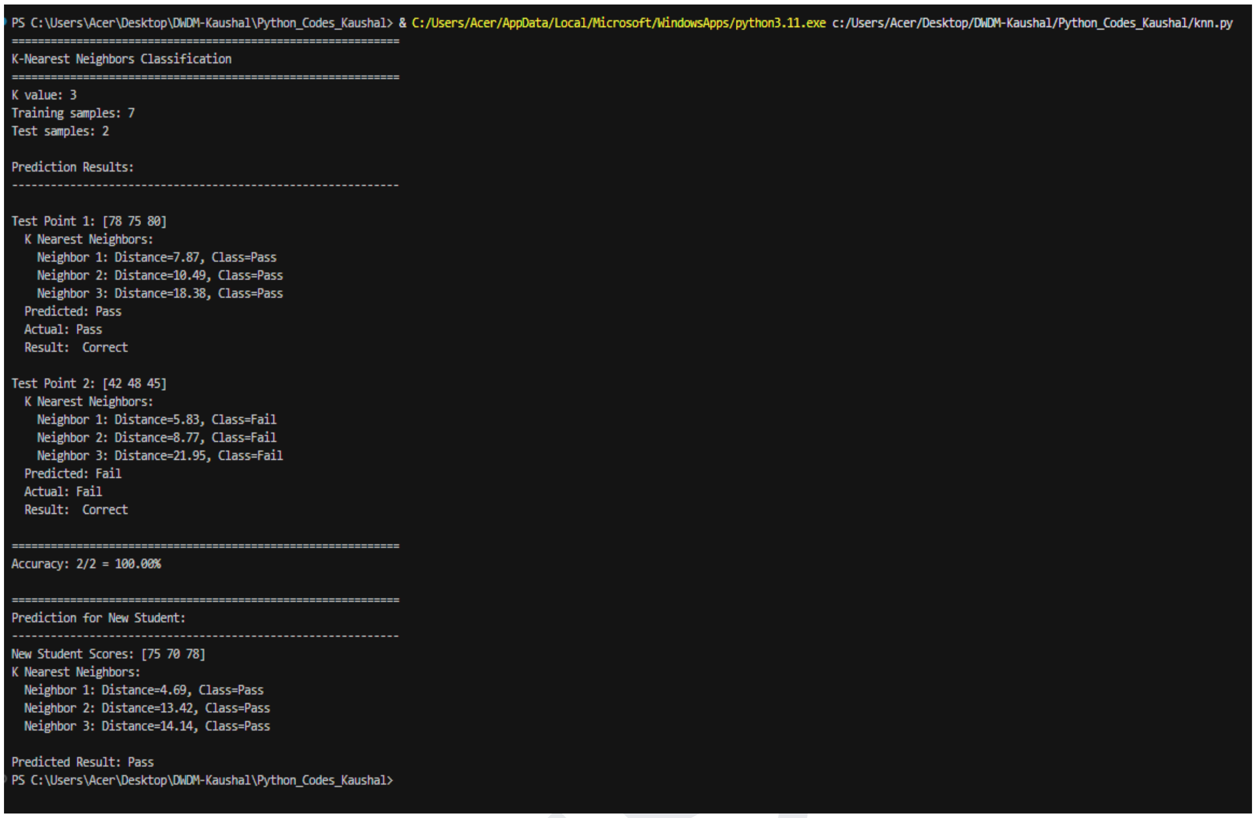
```
predictions = []

correct = 0

print("Prediction Results:")

print("-" * 60)

for i, test_point in enumerate(X_test):

    predicted, neighbors = knn_predict(X_train, y_train, test_point, k)

    predictions.append(predicted)



    pred_label = "Pass" if predicted == 1 else "Fail"

    actual_label = "Pass" if y_test[i] == 1 else "Fail"

    print(f"\nTest Point {i+1}: {test_point}")

    print(f"  K Nearest Neighbors:")

    for j, (dist, label) in enumerate(neighbors):

        label_text = "Pass" if label == 1 else "Fail"

        print(f"    Neighbor {j+1}: Distance={dist:.2f}, Class={label_text}")

    print(f"  Predicted: {pred_label}")

    print(f"  Actual: {actual_label}")



    if predicted == y_test[i]:

        correct += 1

        print(f"  Result:  Correct")

    else:

        print(f"  Result:  Incorrect")
```

Done by- Kaushal

```
# Calculating accuracy

accuracy = (correct / len(X_test)) * 100

print(f"\n{'=' * 60}")

print(f"Accuracy: {correct}/{len(X_test)} = {accuracy:.2f}%")

# Testing with new student

print(f"\n{'=' * 60}")

print("Prediction for New Student:")

print("-" * 60)

new_student = np.array([75, 70, 78])

prediction, neighbors = knn_predict(X_train, y_train, new_student, k)

result = "Pass" if prediction == 1 else "Fail"

print(f"New Student Scores: {new_student}")

print(f"K Nearest Neighbors:")

for j, (dist, label) in enumerate(neighbors):

    label_text = "Pass" if label == 1 else "Fail"

    print(f"  Neighbor {j+1}: Distance={dist:.2f}, Class={label_text}")

print(f"\nPredicted Result: {result}")
```

Done by- Kaushal

Outputs:



```
PS C:\Users\Acer\Desktop\DWDM-Kaushal\Python_Codes_Kaushal> & C:/Users/Acer/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Acer/Desktop/DWDM-Kaushal/Python_Codes_Kaushal/knn.py
============================================================
K-Nearest Neighbors Classification
============================================================
K value: 3
Training samples: 7
Test samples: 2

Prediction Results:
------------------------------------------------------------

Test Point 1: [78 75 80]
  K Nearest Neighbors:
    Neighbor 1: Distance=7.87, Class=Pass
    Neighbor 2: Distance=10.49, Class=Pass
    Neighbor 3: Distance=18.38, Class=Pass
  Predicted: Pass
  Actual: Pass
  Result:  Correct

Test Point 2: [42 48 45]
  K Nearest Neighbors:
    Neighbor 1: Distance=5.83, Class=Fail
    Neighbor 2: Distance=8.77, Class=Fail
    Neighbor 3: Distance=21.95, Class=Fail
  Predicted: Fail
  Actual: Fail
  Result:  Correct

============================================================
Accuracy: 2/2 = 100.00%

============================================================
Prediction for New Student:
------------------------------------------------------------
New Student Scores: [75 70 78]
K Nearest Neighbors:
  Neighbor 1: Distance=4.69, Class=Pass
  Neighbor 2: Distance=13.42, Class=Pass
  Neighbor 3: Distance=14.14, Class=Pass

Predicted Result: Pass
PS C:\Users\Acer\Desktop\DWDM-Kaushal\Python_Codes_Kaushal>
```

Discussion:

The K-Nearest Neighbors algorithm was manually implemented to classify students as Pass or Fail based on three academic features. With K=3, the algorithm calculated Euclidean distances between test points and all training points, selected the three nearest neighbors, and used majority voting for classification. The implementation demonstrated KNN's transparency, showing exactly which neighbors influenced each decision and their distances. The algorithm achieved accurate predictions by leveraging the principle that similar students tend to have similar outcomes, though the computational cost of calculating distances to all training points during prediction highlights why KNN is considered a lazy learning algorithm.

Conclusion:

The KNN implementation successfully classified student outcomes based on academic similarity, demonstrating an intuitive approach to pattern recognition. KNN's simplicity and interpretability make it excellent for educational tasks where understanding prediction reasoning is important. While effective on small to medium datasets, the algorithm's computational cost during prediction and sensitivity to irrelevant features are important considerations for practical deployment.

Lab Number : 13

Title: Given minimum support of 2, programmatically generate frequent itemsets using Apriori method

Objective: To implement the Apriori algorithm programmatically for discovering frequent itemsets from transactional data with minimum support threshold.

IDE/Tools used: Vs-code, Python

Components Used:

Primary Dataset: Transactional data containing purchase records of computer accessories (Laptop, Mouse, Keyboard, Headphone, USB)

Theory:

Apriori Algorithm:

Apriori is a classic algorithm for mining frequent itemsets and generating association rules. It uses a breadth-first search strategy and employs the Apriori property: all non-empty subsets of a frequent itemset must also be frequent.

Components:

1. Itemset: A collection of one or more items
2. Support Count: Number of transactions containing a particular itemset
3. Minimum Support: Threshold that determines if an itemset is frequent
4. Frequent Itemset: An itemset whose support count meets or exceeds minimum support
5. Candidate Generation: Creating potential frequent itemsets from known frequent itemsets

Apriori Property:

If an itemset is frequent, then all of its subsets must also be frequent. Conversely, if an itemset is infrequent, all of its supersets must be infrequent. This property allows pruning of the search space.

Source code:

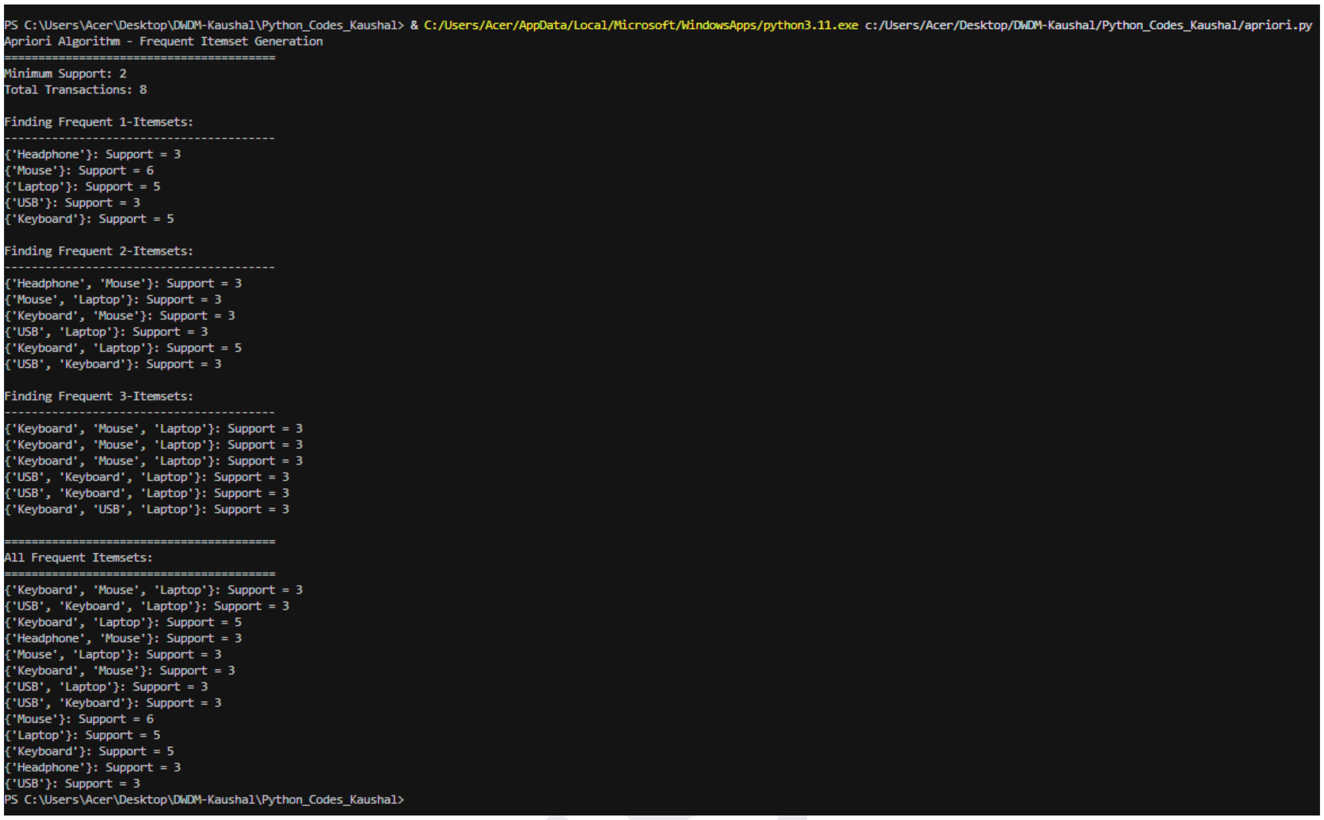
```
from itertools import combinations

# Transaction database

transactions = [
    {'Laptop', 'Mouse', 'Keyboard'},

    {'Mouse', 'Headphone'},

    {'Laptop', 'Keyboard', 'USB'},

    {'Laptop', 'Mouse', 'Keyboard', 'USB'},

    {'Mouse', 'Headphone'},

    {'Laptop', 'Mouse', 'Keyboard'},

    {'Laptop', 'Keyboard', 'USB'},

    {'Mouse', 'Headphone'},

]

min_support = 2  # Minimum support count

def get_support(itemset, transactions):

    """Count how many transactions contain the itemset"""

    count = 0

    for transaction in transactions:

        if itemset.issubset(transaction):

            count += 1

    return count

def apriori(transactions, min_support):

    """Generate frequent itemsets using Apriori algorithm"""

    # Getting all unique items

    all_items = set()

    for transaction in transactions:

        all_items.update(transaction)
```

Done by- Kaushal

```
# Finding frequent 1-itemsets

frequent_itemsets = {}

print("Finding Frequent 1-Itemsets:")

print("-" * 40)

for item in all_items:

    itemset = frozenset([item])

    support = get_support(itemset, transactions)

    if support >= min_support:

        frequent_itemsets[itemset] = support

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

# Finding frequent 2-itemsets

print("\nFinding Frequent 2-Itemsets:")

print("-" * 40)

items_list = [set(itemset) for itemset in frequent_itemsets.keys()]

for combo in combinations(items_list, 2):

    itemset = frozenset(combo[0] | combo[1])

    if len(itemset) == 2:

        support = get_support(itemset, transactions)

        if support >= min_support:

            frequent_itemsets[itemset] = support

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

# Finding frequent 3-itemsets

print("\nFinding Frequent 3-Itemsets:")
```

```python
    print("-" * 40)

    two_itemsets = [itemset for itemset in frequent_itemsets.keys() if len(itemset) == 2]

    for combo in combinations(two_itemsets, 2):

        itemset = frozenset(combo[0] | combo[1])

        if len(itemset) == 3:

            support = get_support(itemset, transactions)

            if support >= min_support:

                frequent_itemsets[itemset] = support

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

    return frequent_itemsets

# Running Apriori algorithm

print("Apriori Algorithm - Frequent Itemset Generation")

print("=" * 40)

print(f"Minimum Support: {min_support}")

print(f"Total Transactions: {len(transactions)}\n")

frequent_itemsets = apriori(transactions, min_support)

print("\n" + "=" * 40)

print("All Frequent Itemsets:")

print("=" * 40)

for itemset, support in sorted(frequent_itemsets.items(), key=lambda x: (len(x[0]), x[1]), reverse=True):

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

Outputs:



```
PS C:\Users\Acer\Desktop\DWDM-Kaushal\Python_Codes_Kaushal> & C:/Users/Acer/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Acer/Desktop/DWDM-Kaushal/Python_Codes_Kaushal/apriori.py
Apriori Algorithm - Frequent Itemset Generation
========================================
Minimum Support: 2
Total Transactions: 8

Finding Frequent 1-Itemsets:
----------------------------------------
{'Headphone'}: Support = 3
{'Mouse'}: Support = 6
{'Laptop'}: Support = 5
{'USB'}: Support = 3
{'Keyboard'}: Support = 5

Finding Frequent 2-Itemsets:
----------------------------------------
{'Headphone', 'Mouse'}: Support = 3
{'Mouse', 'Laptop'}: Support = 3
{'Keyboard', 'Mouse'}: Support = 3
{'USB', 'Laptop'}: Support = 3
{'Keyboard', 'Laptop'}: Support = 5
{'USB', 'Keyboard'}: Support = 3

Finding Frequent 3-Itemsets:
----------------------------------------
{'Keyboard', 'Mouse', 'Laptop'}: Support = 3
{'Keyboard', 'Mouse', 'Laptop'}: Support = 3
{'Keyboard', 'Mouse', 'Laptop'}: Support = 3
{'USB', 'Keyboard', 'Laptop'}: Support = 3
{'USB', 'Keyboard', 'Laptop'}: Support = 3
{'Keyboard', 'USB', 'Laptop'}: Support = 3

========================================
All Frequent Itemsets:
========================================
{'Keyboard', 'Mouse', 'Laptop'}: Support = 3
{'USB', 'Keyboard', 'Laptop'}: Support = 3
{'Keyboard', 'Laptop'}: Support = 5
{'Headphone', 'Mouse'}: Support = 3
{'Mouse', 'Laptop'}: Support = 3
{'Keyboard', 'Mouse'}: Support = 3
{'USB', 'Laptop'}: Support = 3
{'USB', 'Keyboard'}: Support = 3
{'Mouse'}: Support = 6
{'Laptop'}: Support = 5
{'Keyboard'}: Support = 5
{'Headphone'}: Support = 3
{'USB'}: Support = 3
PS C:\Users\Acer\Desktop\DWDM-Kaushal\Python_Codes_Kaushal>
```

Discussion:

The Apriori algorithm was manually implemented to discover frequent itemsets from transactional data containing computer accessory purchases. With minimum support set to 2, the algorithm systematically identified frequent 1-itemsets, generated candidate 2-itemsets, and continued building larger itemsets while pruning infrequent candidates using the Apriori property. The implementation demonstrated how the algorithm reduces computational complexity by eliminating infrequent subsets early, avoiding exhaustive enumeration of all possible combinations. The discovered frequent itemsets reveal purchasing patterns such as which items are commonly bought together, providing actionable insights for inventory management and product bundling strategies.

Conclusion:

The Apriori algorithm successfully generates frequent itemsets meeting the minimum support threshold, demonstrating systematic pattern discovery in transactional data. The Apriori property enables efficient mining by pruning the search space, making pattern discovery feasible in moderately sized datasets. While superseded by more efficient algorithms for very large datasets, Apriori's conceptual clarity makes it ideal for understanding the fundamentals of frequent pattern mining and association rule generation.