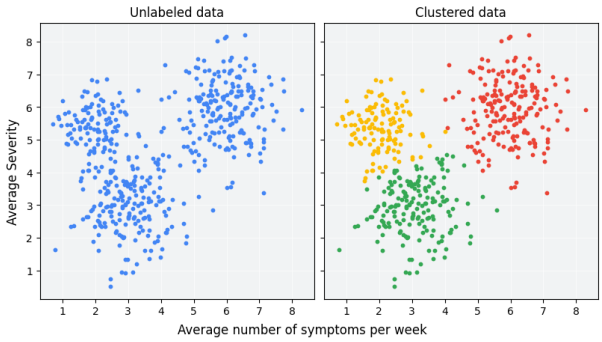
Université Abdelmalek Essaadi

Faculté des Sciences et Techniques de Tanger

Département Génie Informatique

Travaux pratiques BI & DM



Encadré par :
Abdelhadi FENNAN

Elaboré par :
ELJABLY Salaheddine

Cycle d'ingénieur Logiciels et Systèmes Intelligents
S3 - 2024/2025

Table des matières

Experiment 1 :

1 Introduction

This report presents the implementation of a data warehouse for a vehicle hire company. The data warehouse transforms operational data from the HireBase database into a dimensional model in the TopHireDW database to support analytical processing. The implementation uses PostgreSQL as the database management system.

2 Source Database Implementation

The first step was to create and populate the source database (HireBase) that contains the operational data of the vehicle hire company.

2.1 Database Creation

A new database called HireBase was created to store the operational data :

```
CREATE DATABASE hirebase;
```

Listing 1 – Create HireBase Database

2.2 Customer Table Implementation

The Customer table was created to store information about customers :

```
CREATE TABLE Customer (
    CustomerId VARCHAR(20) NOT NULL PRIMARY KEY,
    CustomerName VARCHAR(30),
    DateOfBirth DATE,
    Town VARCHAR(50),
    TelephoneNo VARCHAR(30),
    DrivingLicenceNo VARCHAR(30),
    Occupation VARCHAR(30)
);
```

Listing 2 – Create Customer Table

2.3 Van Table Implementation

The Van table stores information about the vehicles available for hire :

```sql
CREATE TABLE Van (
    RegNo VARCHAR(10) NOT NULL PRIMARY KEY,
    Make VARCHAR(30),
    Model VARCHAR(30),
    Year VARCHAR(4),
    Colour VARCHAR(20),
    CC INT,
    Class VARCHAR(10)
);
```

Listing 3 – Create Van Table

2.4 Hire Table Implementation

The Hire table records vehicle hire transactions :

```sql
CREATE TABLE Hire (
    HireId VARCHAR(10) NOT NULL PRIMARY KEY,
    HireDate DATE NOT NULL,
    CustomerId VARCHAR(20) NOT NULL,
    RegNo VARCHAR(10),
    NoOfDays INT,
    VanHire DECIMAL(10,2),
    SatNavHire DECIMAL(10,2),
    Insurance DECIMAL(10,2),
    DamageWaiver DECIMAL(10,2),
    TotalBill DECIMAL(10,2)
);
```

Listing 4 – Create Hire Table

3 Data Warehouse Implementation

After creating the source database, the next step was to design and implement the data warehouse using a star schema.

3.1 Database Creation

A new database called TopHireDW was created to store the data warehouse :

```sql
CREATE DATABASE tophiredw;
```

Listing 5 – Create TopHireDW Database

3.2 Dimension Tables Implementation

3.2.1 Date Dimension

The Date dimension table was created to support time-based analysis :

```
CREATE TABLE DimDate (
    DateKey INT NOT NULL PRIMARY KEY,
    Year VARCHAR(7),
    Month VARCHAR(7),
    Date DATE,
    DateString VARCHAR(10)
);
```

Listing 6 – Create Date Dimension

3.2.2 Customer Dimension

The Customer dimension table stores customer information :

```
CREATE TABLE DimCustomer (
    CustomerKey SERIAL PRIMARY KEY,
    CustomerId VARCHAR(20) NOT NULL,
    CustomerName VARCHAR(30),
    DateOfBirth DATE,
    Town VARCHAR(50),
    TelephoneNo VARCHAR(30),
    DrivingLicenceNo VARCHAR(30),
    Occupation VARCHAR(30)
);
```

Listing 7 – Create Customer Dimension

3.2.3 Van Dimension

The Van dimension table stores vehicle information :

```
CREATE TABLE DimVan (
    VanKey SERIAL PRIMARY KEY,
    RegNo VARCHAR(10) NOT NULL,
    Make VARCHAR(30),
    Model VARCHAR(30),
    Year VARCHAR(4),
    Colour VARCHAR(20),
    CC INT,
    Class VARCHAR(10)
);
```

Listing 8 – Create Van Dimension

3.3 Fact Table Implementation

The Fact table was created to store hire transaction details :

```
CREATE TABLE FactHire (
    HireId VARCHAR(10) NOT NULL PRIMARY KEY,
    HireDate INT NOT NULL,
    CustomerKey INT NOT NULL,
    VanKey INT NOT NULL,
    NoOfDays INT,
    VanHire DECIMAL(10,2),
    SatNavHire DECIMAL(10,2),
    Insurance DECIMAL(10,2),
    DamageWaiver DECIMAL(10,2),
    TotalBill DECIMAL(10,2),
    FOREIGN KEY (HireDate) REFERENCES DimDate(DateKey),
    FOREIGN KEY (CustomerKey) REFERENCES DimCustomer(CustomerKey),
    FOREIGN KEY (VanKey) REFERENCES DimVan(VanKey)
);
```

Listing 9 – Create Fact Hire Table

4 ETL Process

The Extract, Transform, and Load (ETL) process is crucial for populating the data warehouse with data from the source database.

4.1 Extract

Data was extracted from the source database (HireBase) using SQL queries to retrieve the required data.

4.2 Transform

The extracted data was transformed to fit the schema of the data warehouse. For instance, the HireDate in the source table was transformed into a DateKey in the DimDate dimension.

4.3 Load

The transformed data was loaded into the respective dimension and fact tables in the TopHireDW database.

5 OLAP Operations on the Data Warehouse

The following OLAP operations were performed on the data warehouse tables using PostgreSQL :

— **Fact Table :** `FactHire`

— **Dimension Tables :** `DimDate`, `DimCustomer`, `DimVan`

5.1 Roll-Up

The roll-up operation aggregates the total bill per year.

```sql
SELECT d."Year", SUM(f."TotalBill") AS total_revenue
FROM "FactHire" f
JOIN "DimDate" d ON f."HireDateKey" = d."DateKey"
GROUP BY d."Year"
ORDER BY d."Year";
```

```
tophiredw=# SELECT d."year" , SUM(f."totalbill") AS total_revenue FROM "facthire" f
tophiredw-# JOIN "dimdate" d ON f."hiredatekey" = d."datekey"
tophiredw-# GROUP BY d."year"
tophiredw-# ORDER BY d."year";
 year | total_revenue
------+---------------
 2011 |     339830.00
(1 row)

tophiredw=#
```

5.2 Drill-Down

The drill-down operation increases granularity by aggregating revenue per month.

```sql
SELECT d."Year", d."Month", SUM(f."TotalBill") AS monthly_revenue
FROM "FactHire" f
JOIN "DimDate" d ON f."HireDateKey" = d."DateKey"
GROUP BY d."Year", d."Month"
ORDER BY d."Year", d."Month";
```

```
 year |  month  | monthly_revenue
------+---------+-----------------
 2011 | 2011-01 |        52700.00
 2011 | 2011-02 |        47430.00
 2011 | 2011-03 |        52870.00
 2011 | 2011-04 |        51000.00
 2011 | 2011-05 |        52700.00
 2011 | 2011-06 |        51000.00
 2011 | 2011-07 |        32130.00
(7 rows)

tophiredw=#
```

5.3 Slice

The slice operation selects a sub-cube for a specific year (e.g., 2010).

```
SELECT f."HireId", f."TotalBill", c."CustomerName"
FROM "FactHire" f
JOIN "DimDate" d ON f."HireDateKey" = d."DateKey"
JOIN "DimCustomer" c ON f."CustomerKey" = c."CustomerKey"
WHERE d."Year" = '2010';
```

```
tophiredw=# SELECT f."hireid", f."totalbill", c."customername"
FROM "facthire" f
JOIN "dimdate" d ON f."hiredatekey" = d."datekey"
JOIN "dimcustomer" c ON f."customerkey" = c."customerkey"
WHERE d."year" = '2010';
 hireid | totalbill | customername
--------+-----------+--------------
(0 rows)

tophiredw=#
```

5.4 Dice

The dice operation selects a sub-cube with multiple conditions on dimensions.

```
SELECT f."HireId", f."TotalBill", d."Year", c."Town"
FROM "FactHire" f
JOIN "DimDate" d ON f."HireDateKey" = d."DateKey"
JOIN "DimCustomer" c ON f."CustomerKey" = c."CustomerKey"
WHERE d."Year" BETWEEN '2010' AND '2012'
  AND c."Town" IN ('Town01', 'Town02');
```

```
 AND c."Town" IN ('Town01', 'Town02');
 hireid | totalbill | year |  town
--------+-----------+------+--------
 H0001  |    170.00 | 2011 | Town01
 H0002  |    340.00 | 2011 | Town02
 H0101  |    340.00 | 2011 | Town01
 H0102  |    510.00 | 2011 | Town02
 H0201  |    510.00 | 2011 | Town01
 H0202  |    170.00 | 2011 | Town02
 H0301  |    170.00 | 2011 | Town01
 H0302  |    340.00 | 2011 | Town02
 H0401  |    340.00 | 2011 | Town01
 H0402  |    510.00 | 2011 | Town02
 H0501  |    510.00 | 2011 | Town01
 H0502  |    170.00 | 2011 | Town02
 H0601  |    170.00 | 2011 | Town01
 H0602  |    340.00 | 2011 | Town02
 H0701  |    340.00 | 2011 | Town01
 H0702  |    510.00 | 2011 | Town02
 H0801  |    510.00 | 2011 | Town01
 H0802  |    170.00 | 2011 | Town02
 H0901  |    170.00 | 2011 | Town01
 H0902  |    340.00 | 2011 | Town02
(20 rows)

tophiredw=#
```

5.5 Pivot (Rotate)

The pivot operation rotates data axes to present total revenue per year by customer occupation.

Step 1 : Enable tablefunc extension

```sql
CREATE EXTENSION IF NOT EXISTS tablefunc;
```

Step 2 : Perform Pivot Using `crosstab()`

```sql
SELECT *
FROM crosstab(
  $$
  SELECT d."Year", c."Occupation", SUM(f."TotalBill")
  FROM "FactHire" f
  JOIN "DimDate" d ON f."HireDateKey" = d."DateKey"
  JOIN "DimCustomer" c ON f."CustomerKey" = c."CustomerKey"
  GROUP BY d."Year", c."Occupation"
  ORDER BY 1,2
  $$,
  $$ SELECT DISTINCT "Occupation" FROM "DimCustomer" ORDER BY 1 $$
) AS ct (
  "Year" varchar,
```

```
  "Occupation1" numeric,
  "Occupation2" numeric,
  "Occupation3" numeric
);
```

These OLAP operations enable efficient and multidimensional analysis of hire transactions across time, customer, and vehicle dimensions.

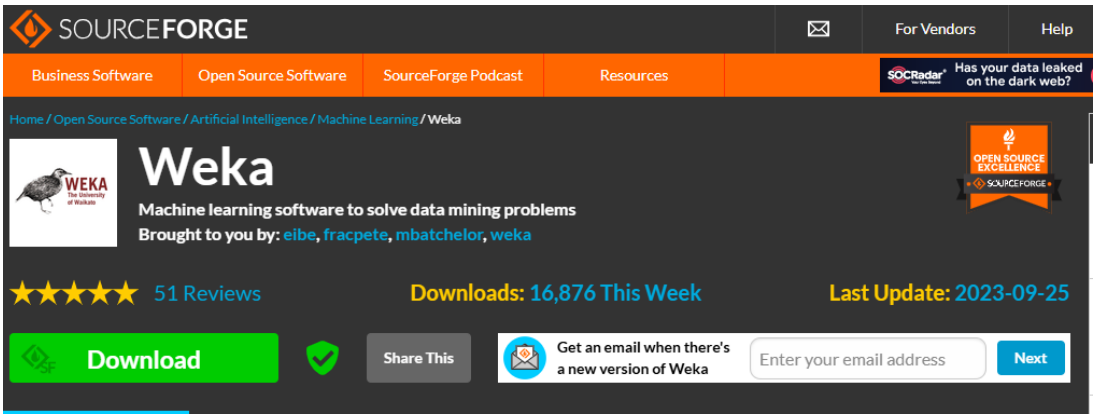
Experiment 2 : Data Processing

6 Integration WEKA for Data Processing

This section documents the successful integration of WEKA with a PostgreSQL database, including the setup process and data analysis workflow.

6.1 WEKA Installation and Configuration

The WEKA software package (version 3.8.6) was obtained from SourceForge (`https://sourceforge.net/projects/weka/`) and installed on an Ubuntu Linux system. The software was extracted to a dedicated directory, with the main executable file located at `weka.jar`.



The WEKA environment was successfully launched using the command :

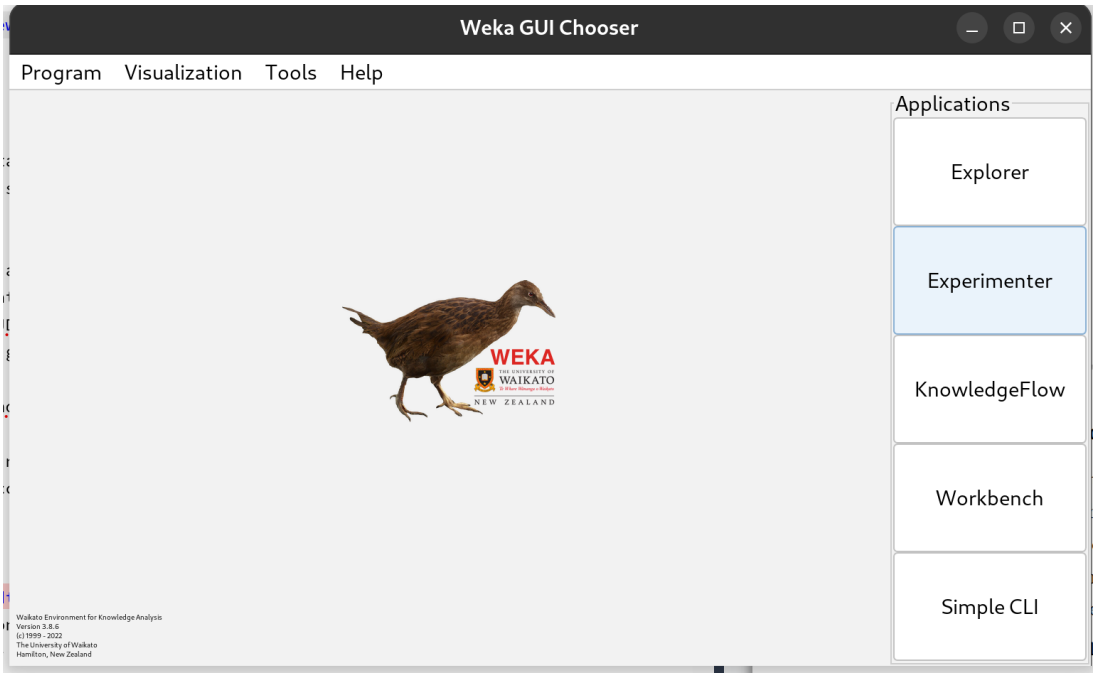
```
java -jar weka.jar
```

6.2 Running the WEKA GUI

The WEKA Graphical User Interface was successfully launched using the following method :

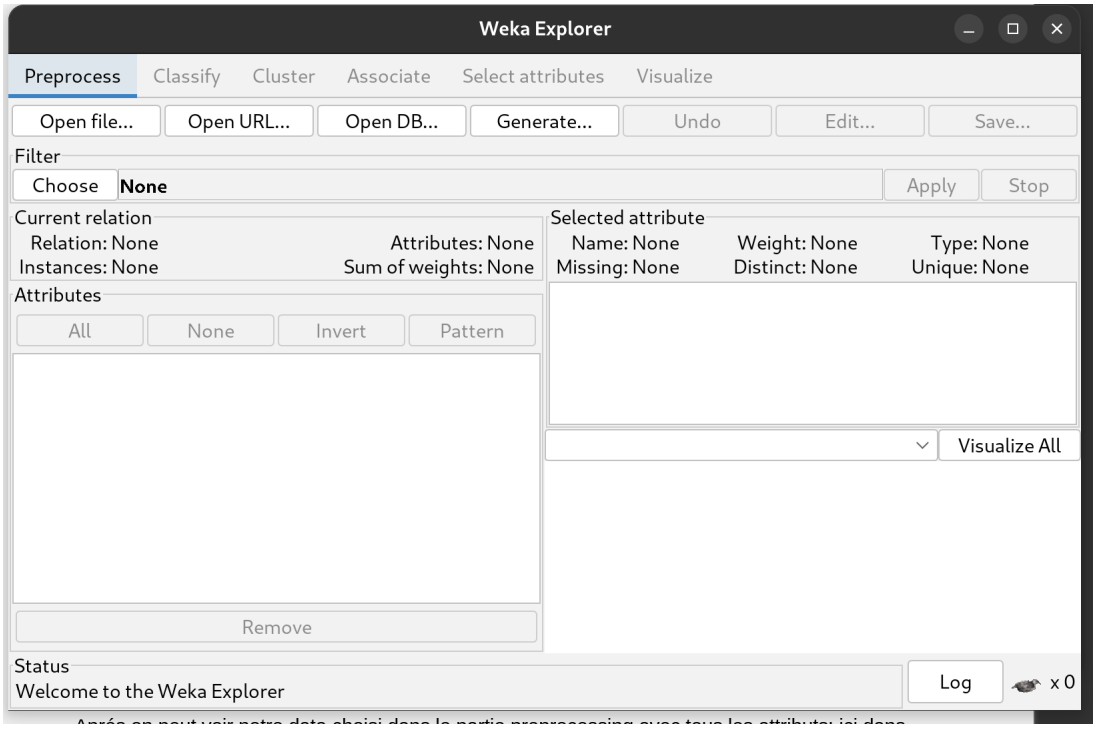
After completing the installation, the WEKA environment was initialized by executing the GUI launcher from the command line. The launch command included both the core WEKA library :

```
java -cp "weka.jar" weka.gui.GUIChooser
```

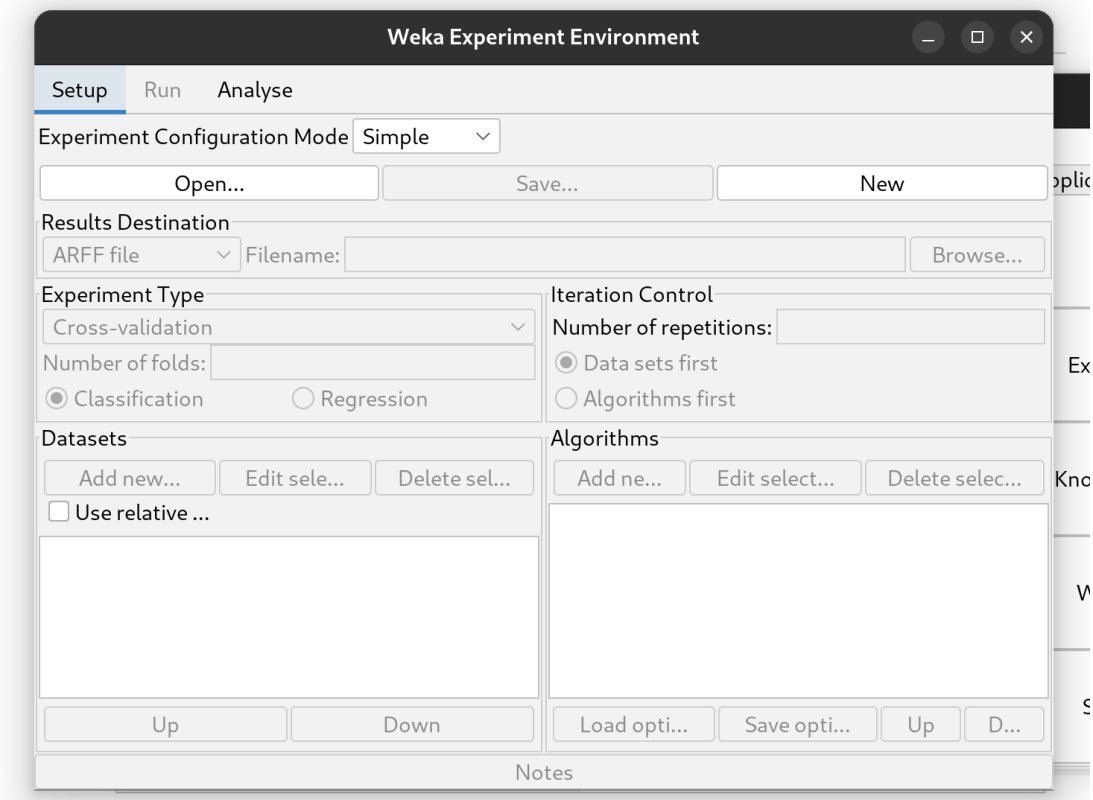


This command successfully initiated the WEKA GUI Chooser, which provides access to all of WEKA's main interfaces including :

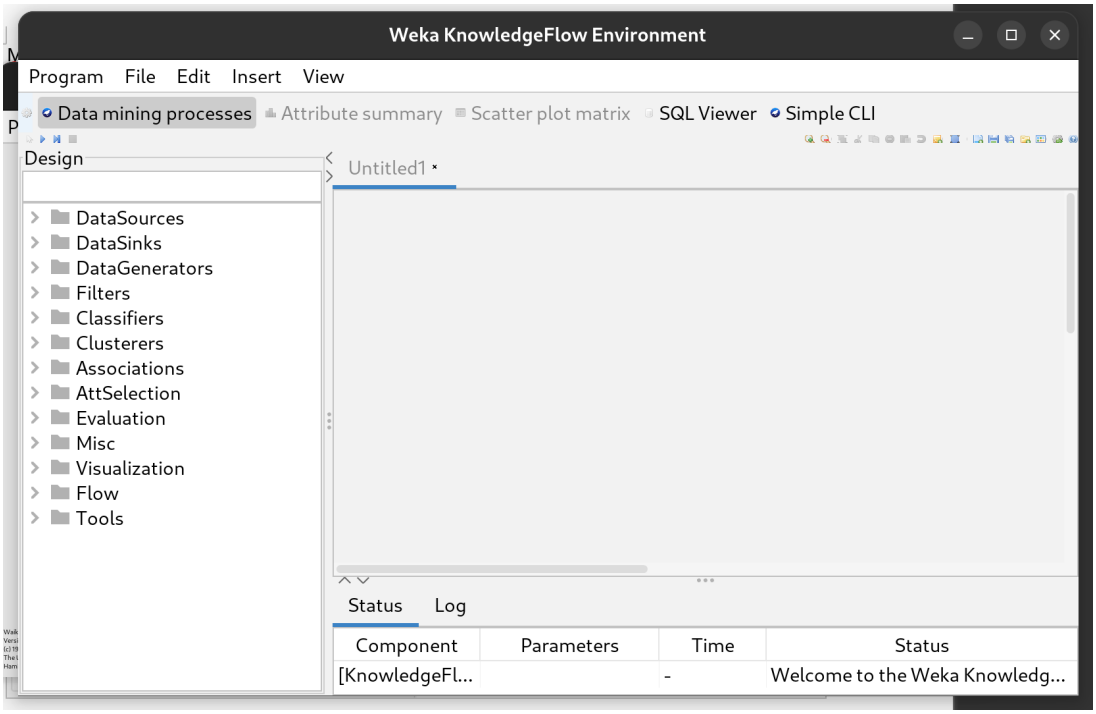
— The Explorer (for data preprocessing and analysis)



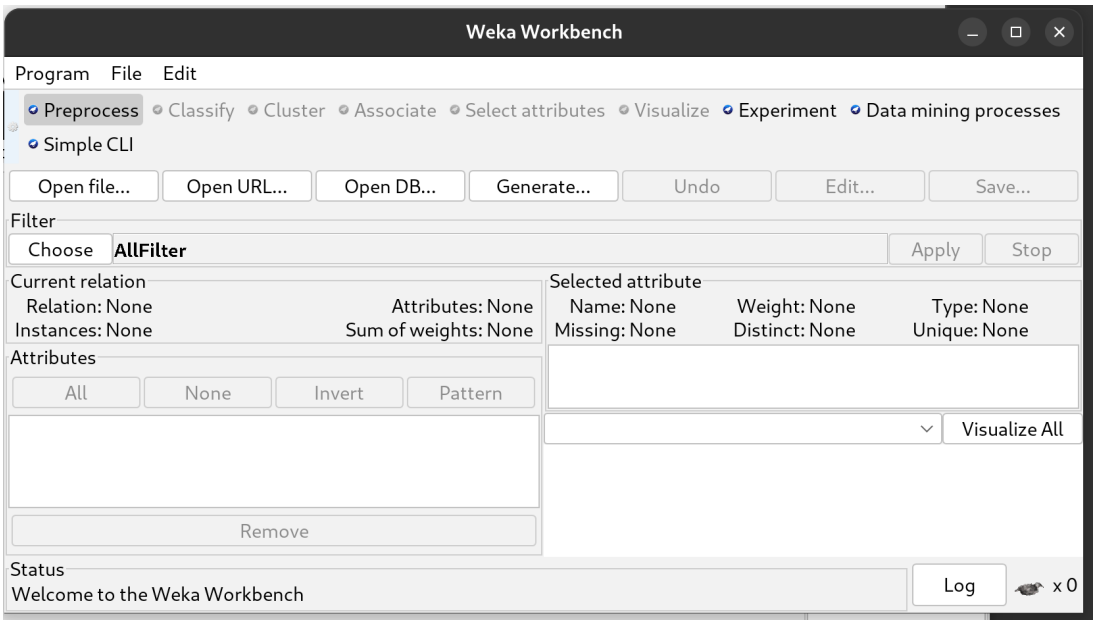
— The Experimenter (for experimental comparisons)



— The Knowledge Flow (for visual pipeline construction)



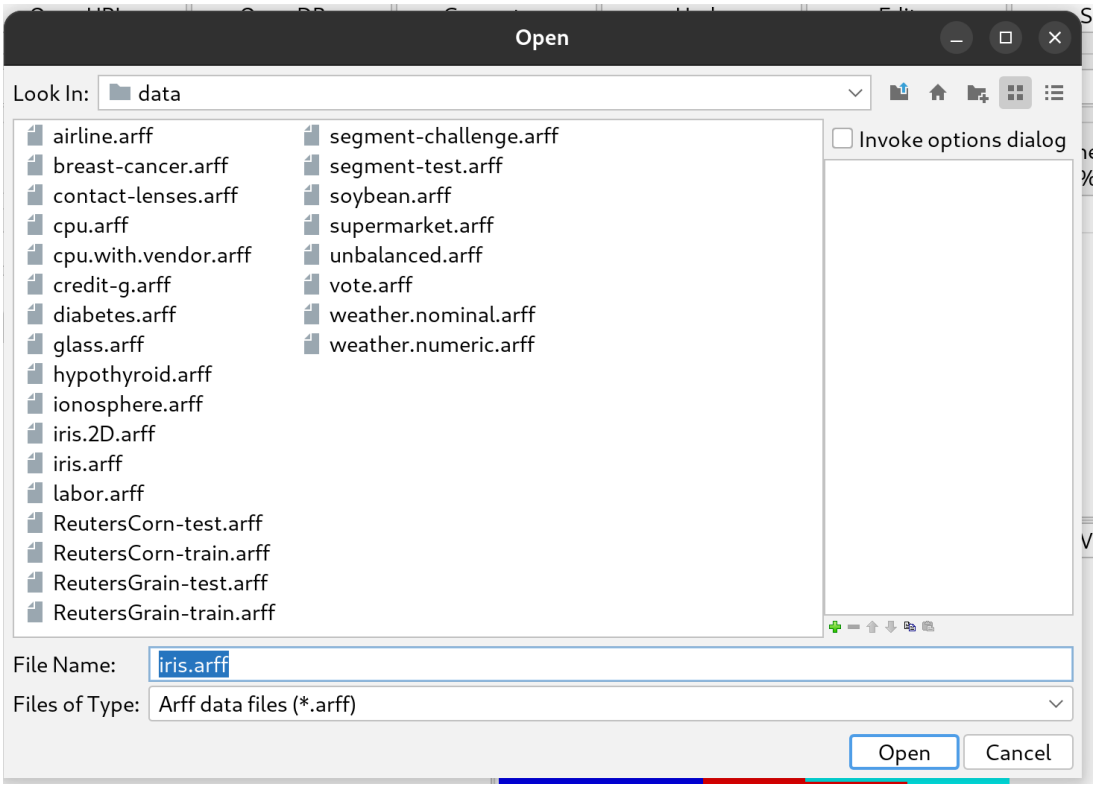
— The Workbench (all tools in one interface)



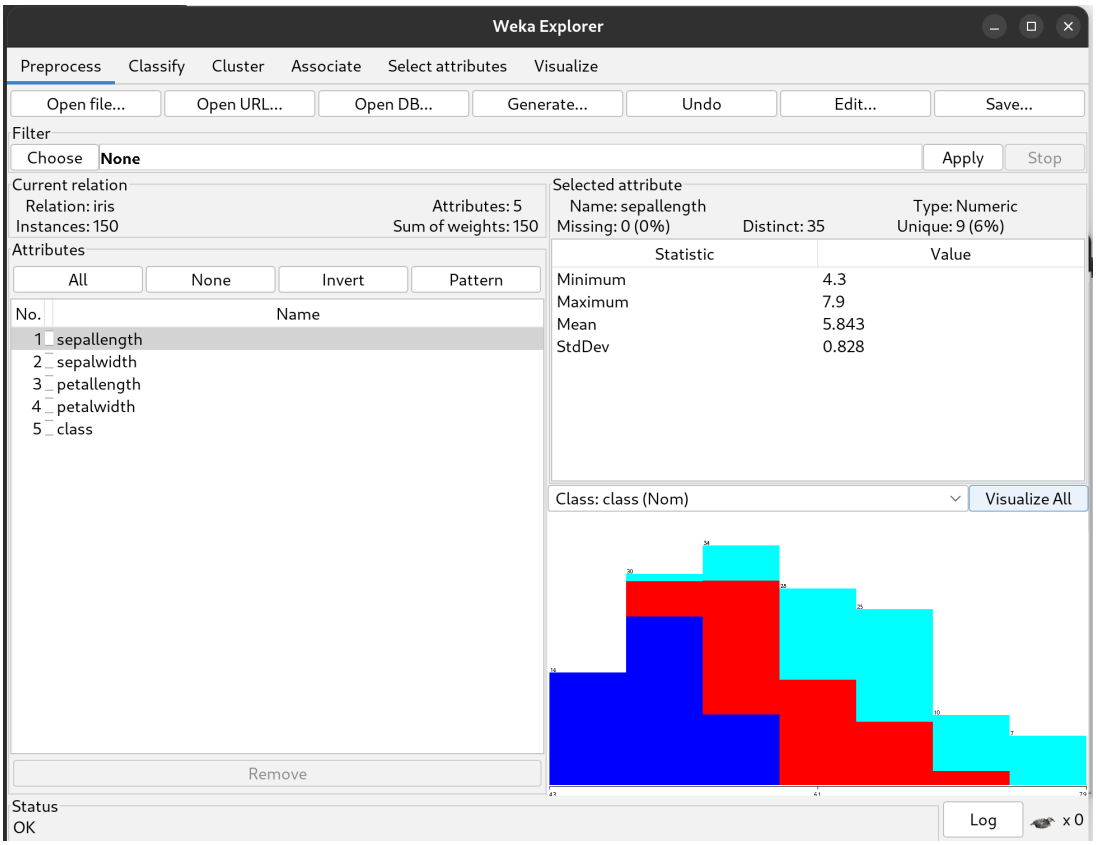
The GUI launched without errors, confirming proper installation and Java environment configuration. All core functionalities were verified to be operational through initial testing of the Explorer interface's data loading and visualization capabilities.

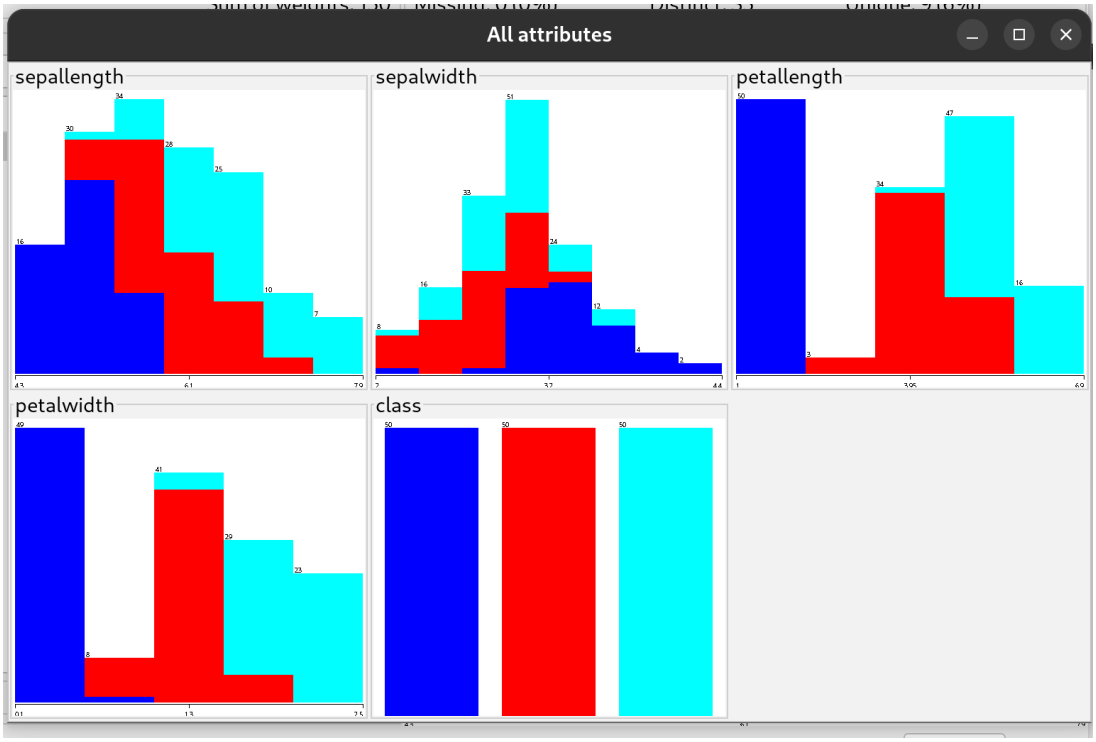
6.3 Visualisation :

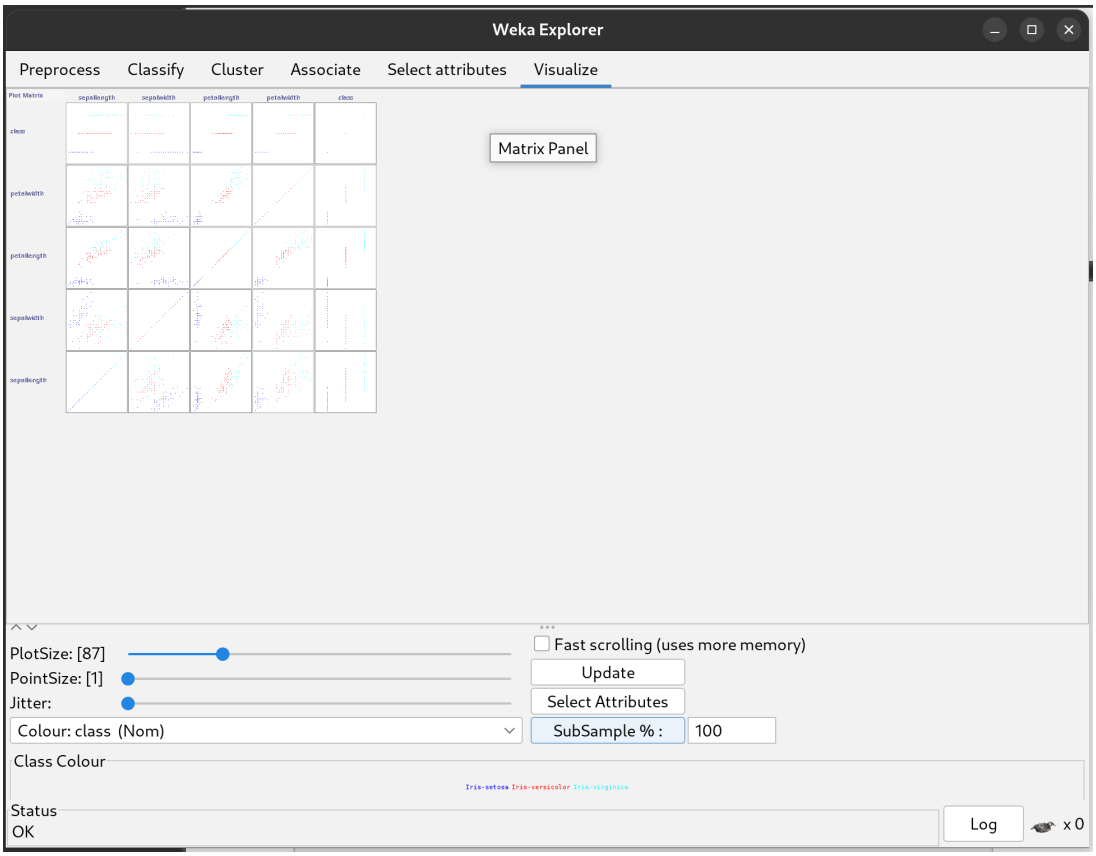
First we Chose the dataset to Process **iris.arff** :

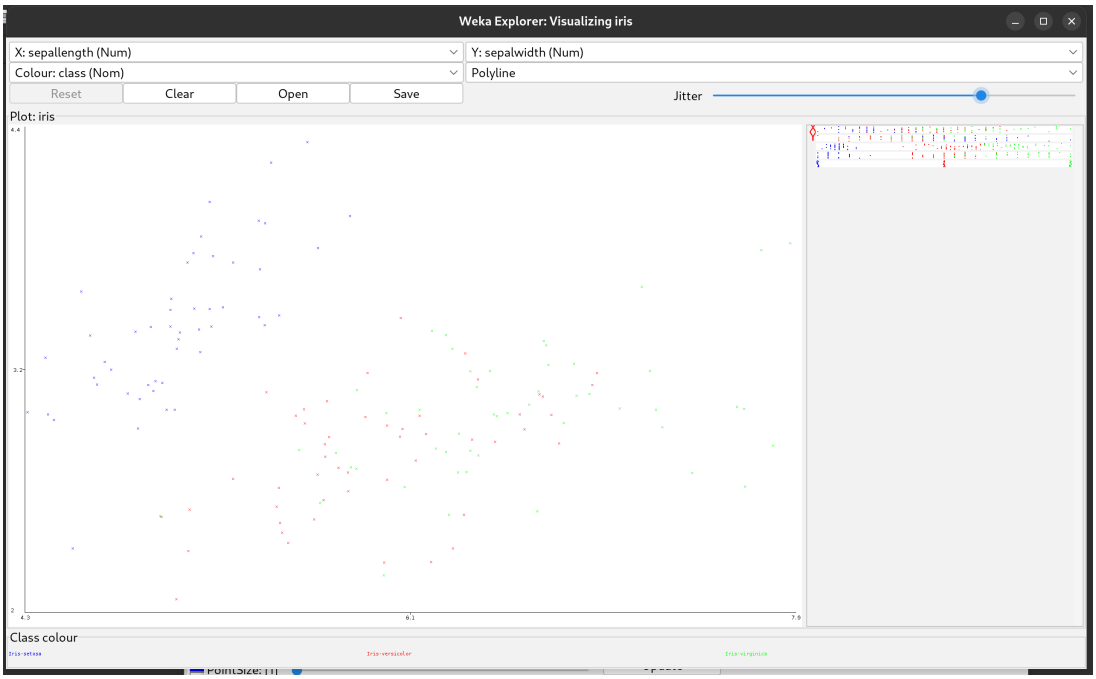


and then We visualise using the Explorer :


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150
Selected attribute
Name: sepallength
Missing: 0 (0%)
Distinct: 35
Type: Numeric
Unique: 9 (6%)
Attributes
All
None
Invert
Pattern
No.
Name
1 sepallength
2 sepalwidth
3 petallength
4 petalwidth
5 class
Statistic
Value
Minimum 4.3
Maximum 7.9
Mean 5.843
StdDev 0.828
Class: class (Nom)
Visualize All
Remove
Status
OK
Log
x 0


All attributes
sepallength
sepalwidth
petallength
petalwidth
class

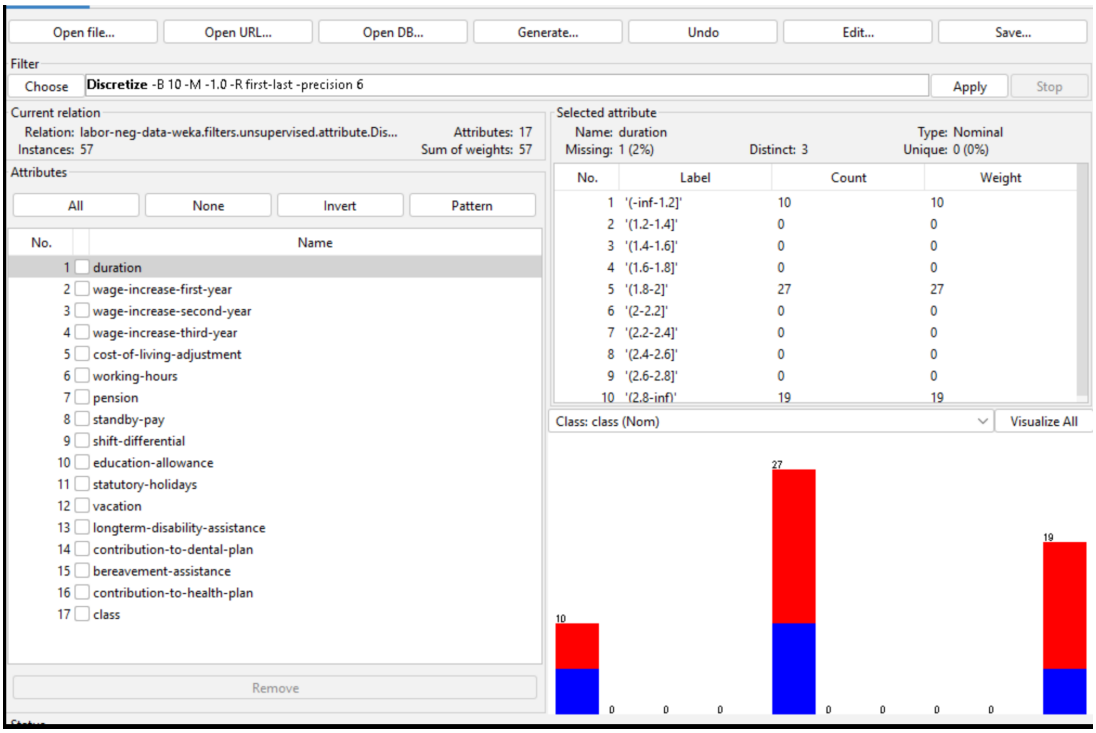

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Matrix Panel
PlotSize: [87]
PointSize: [1]
Jitter:
Fast scrolling (uses more memory)
Update
Select Attributes
Colour: class (Nom)
SubSample % :
100
Class Colour
Status
OK
Log
x 0


Weka Explorer: Visualizing iris
X: sepallength (Num)
Y: sepalwidth (Num)
Colour: class (Nom)
Polyline
Reset
Clear
Open
Save
Jitter
Plot: iris
Class colour

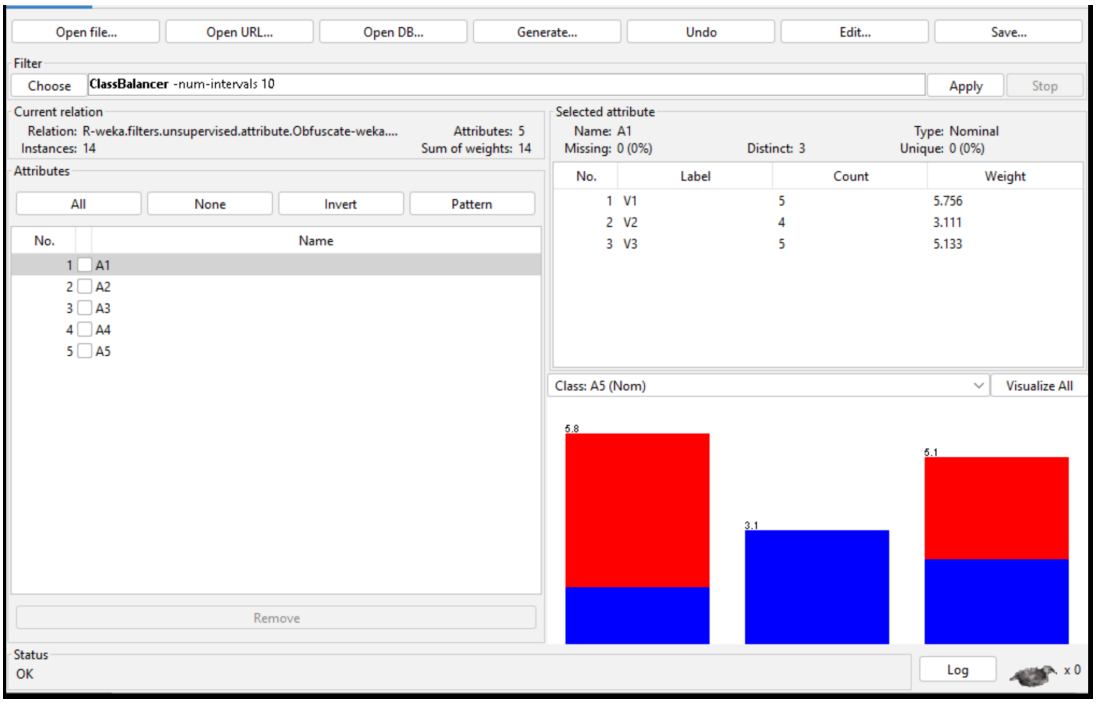
Experiment 3 : Preprocessing Tasks

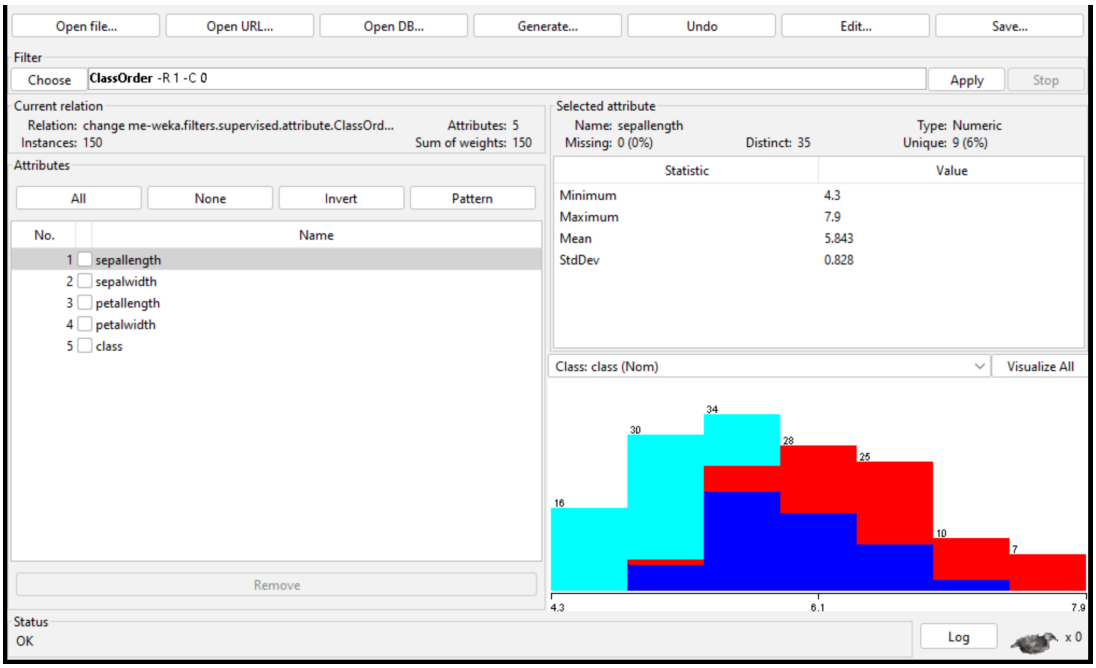
7 Preprocessing Tasks with WEKA :

After remaining in the preprocessing part, we will use the filter option on the labor dataset, and we will choose the unsupervised discrete option.

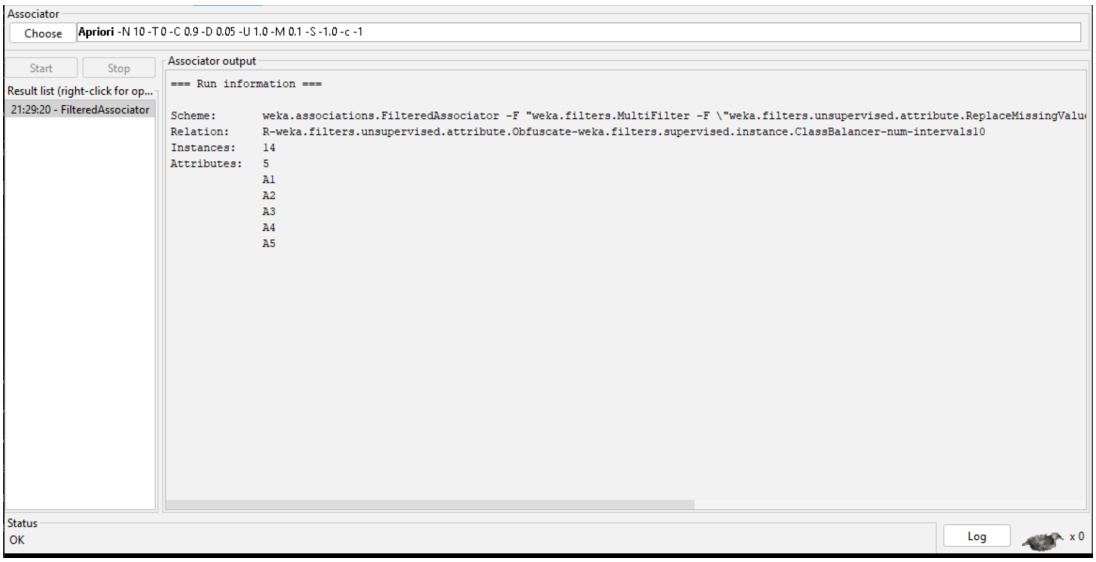


We can test these filters on other datasets such as weather and iris.

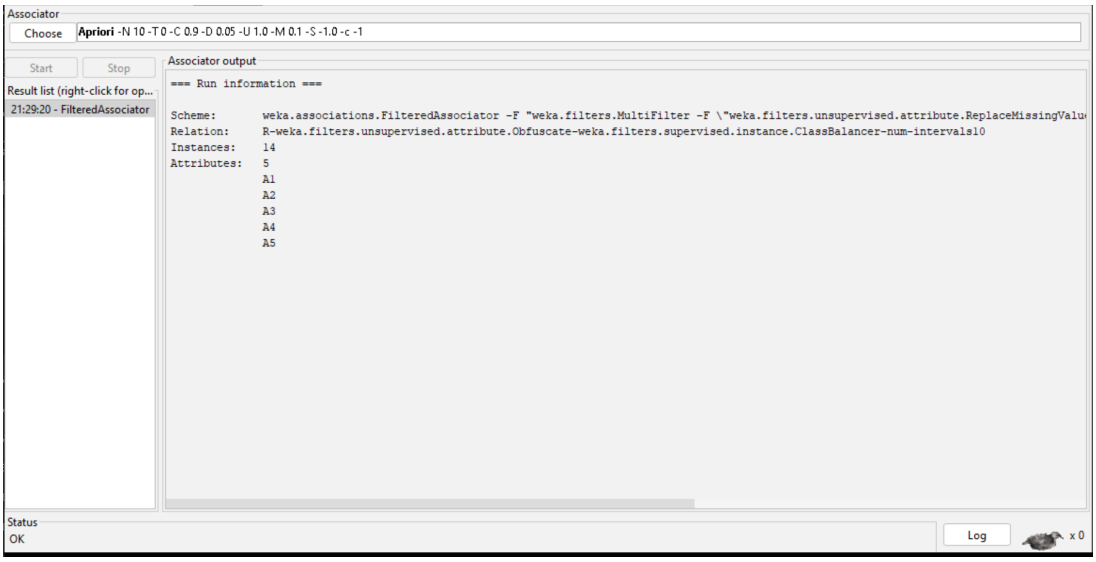




Now we're going to choose the 'Associate' section . we're only using the filtered associator.



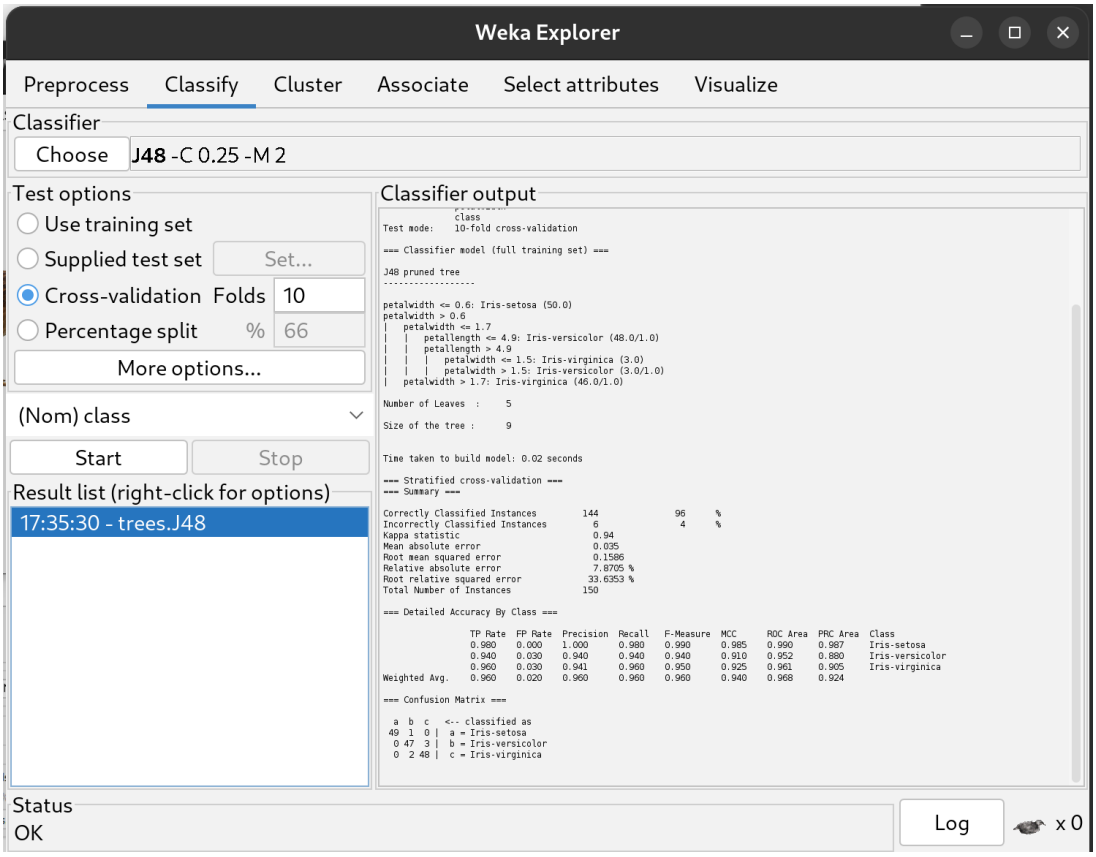
And we're going to do the same thing for the weather dataset and apply the same associators


Associator
Choose
Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Start
Stop
Result list (right-click for op...
21:29:20 - FilteredAssociator
Associator output
=== Run information ===
Scheme: weka.associations.FilteredAssociator -F "weka.filters.MultiFilter -F \"weka.filters.unsupervised.attribute.ReplaceMissingValu
Relation: R-weka.filters.unsupervised.attribute.Obfuscate-weka.filters.supervised.instance.ClassBalancer-num-intervals10
Instances: 14
Attributes: 5
A1
A2
A3
A4
A5
Status
OK
Log
x 0

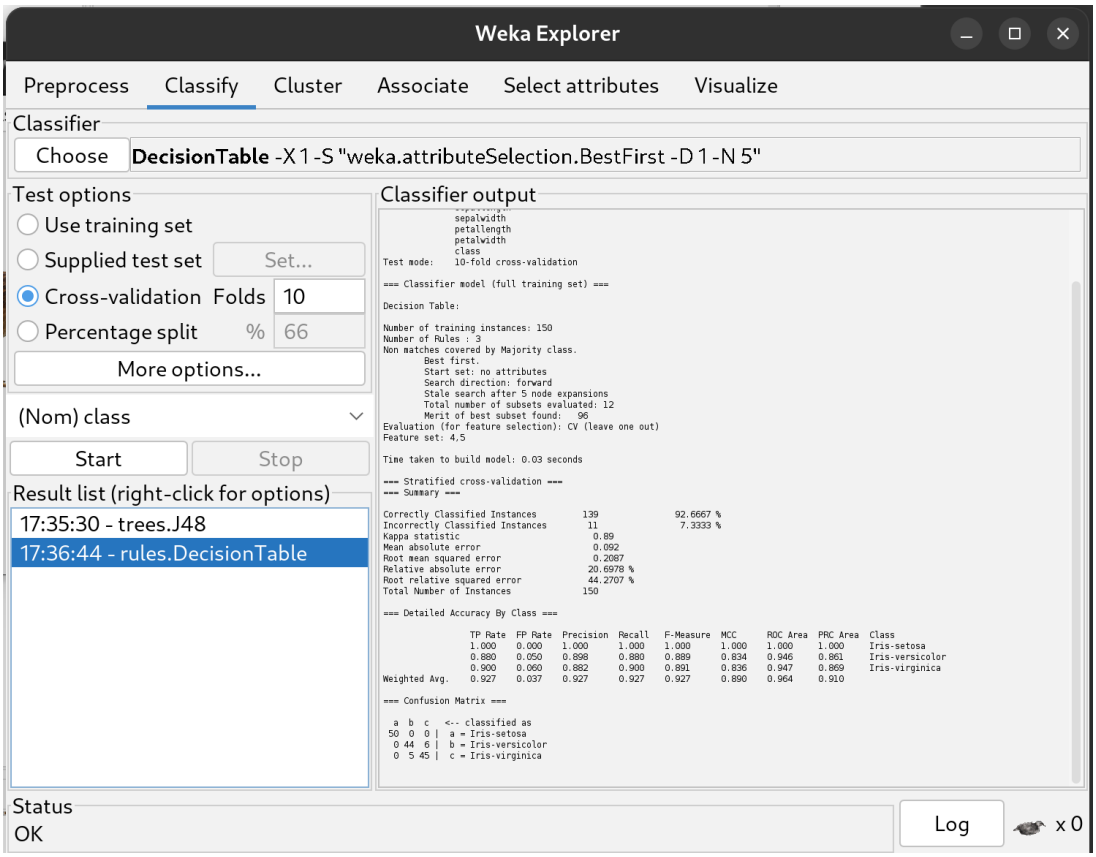
Experiment 4 : Classification

8 Demonstrate performing classification on data sets :

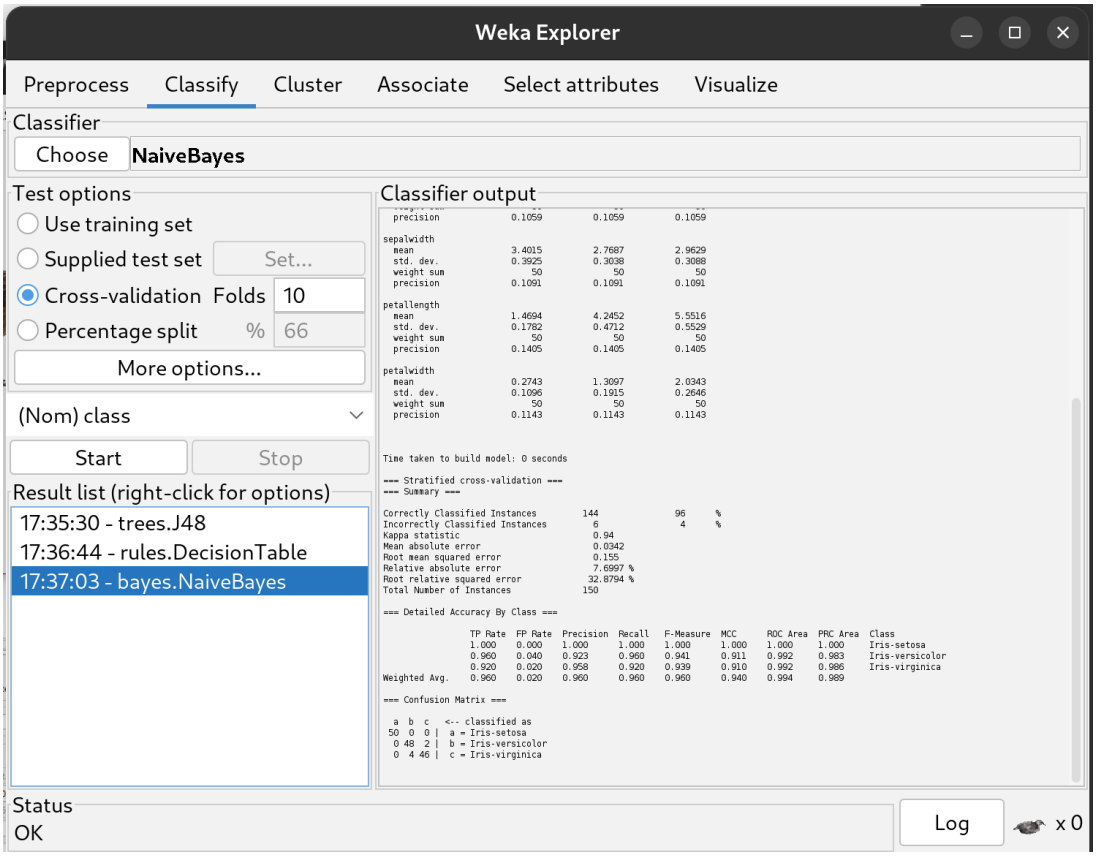
For the part of the classification , we are gonna test on **iris** dataset , we will start by **J48** algorithm :



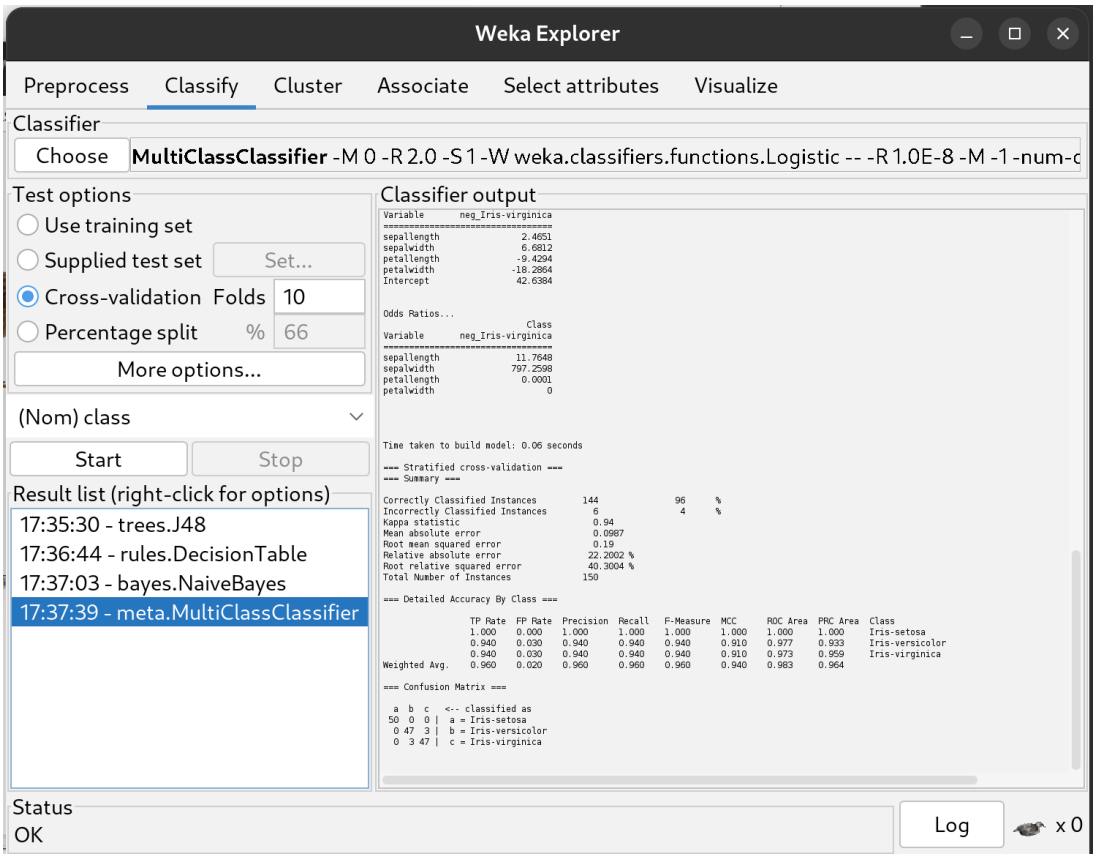
Now we gonna use **Decision Tree** on the same dataset (iris) :



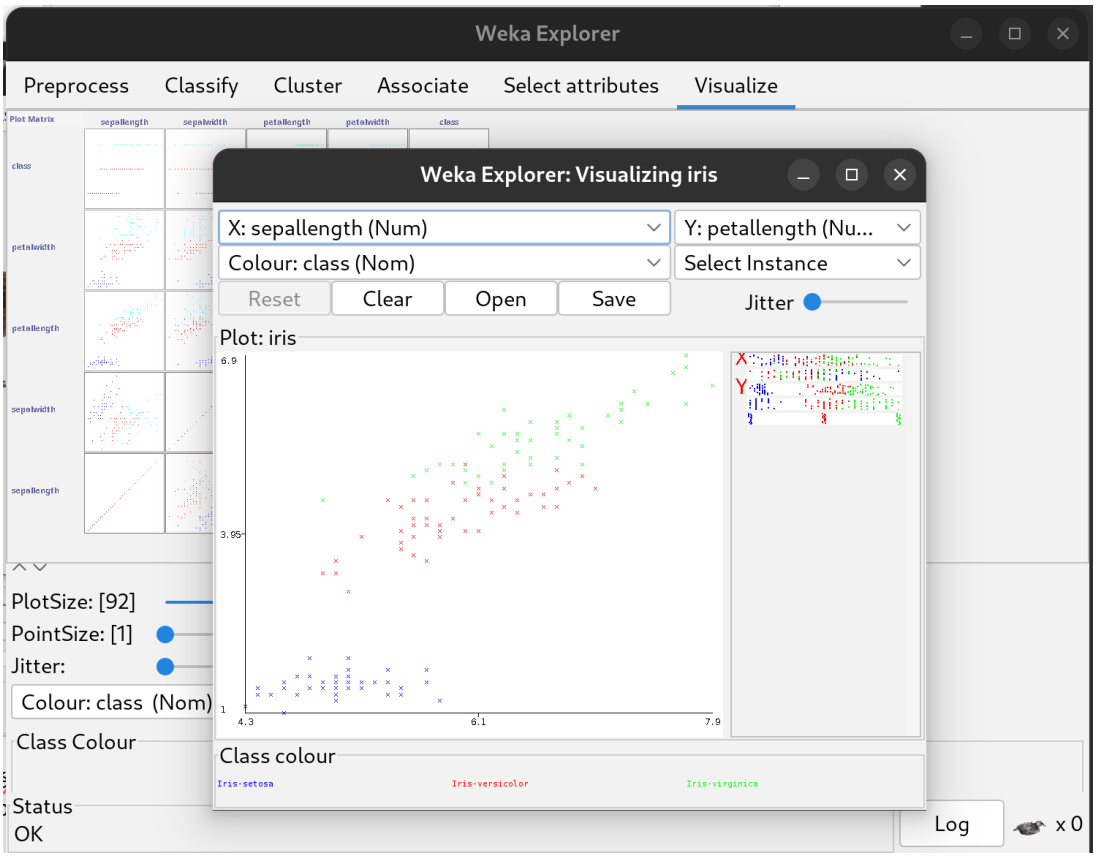
and For **Naive Bays** :



For **K-means** i did not find the algorithm in WEKA so i just used **Multi-Class Classifier**



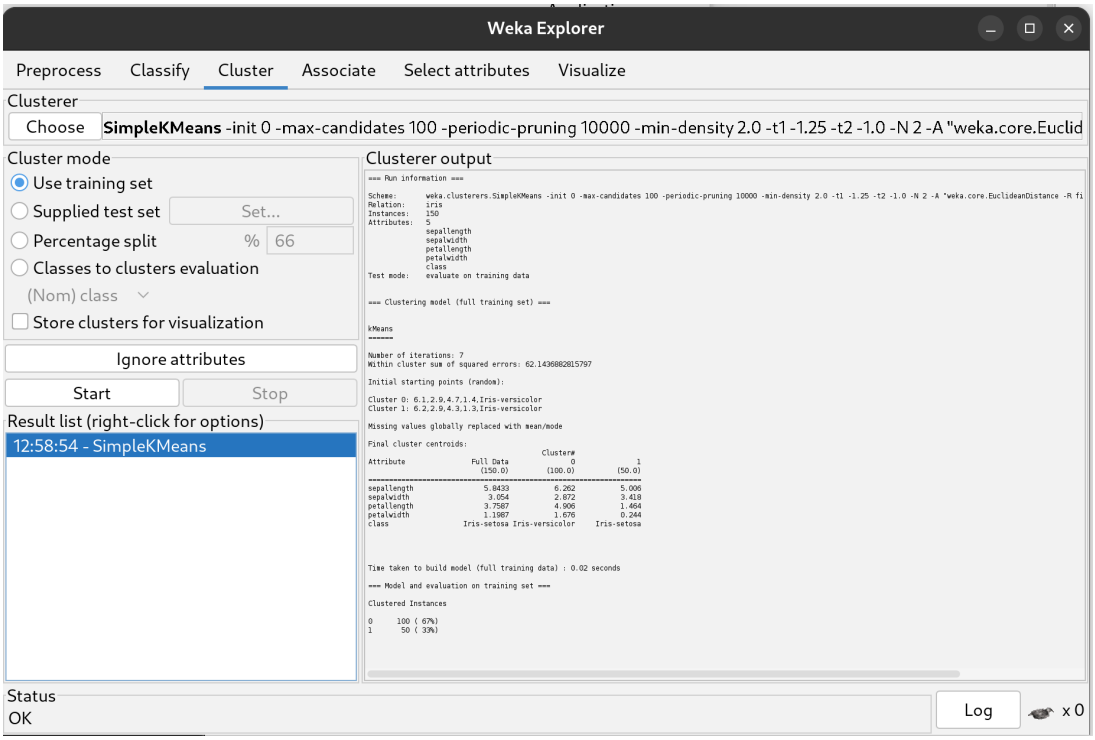
And Finally we Visualise :


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Plot Matrix
sepallength
sepalwidth
petallength
petalwidth
class
Weka Explorer: Visualizing iris
X: sepallength (Num)
Y: petallength (Nu...
Colour: class (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: iris
6.9
3.95
1
4.3
6.1
7.9
Class colour
Iris-setosa
Iris-versicolor
Iris-virginica
PlotSize: [92]
PointSize: [1]
Jitter:
Colour: class (Nom)
Class Colour
Status
OK
Log
x 0

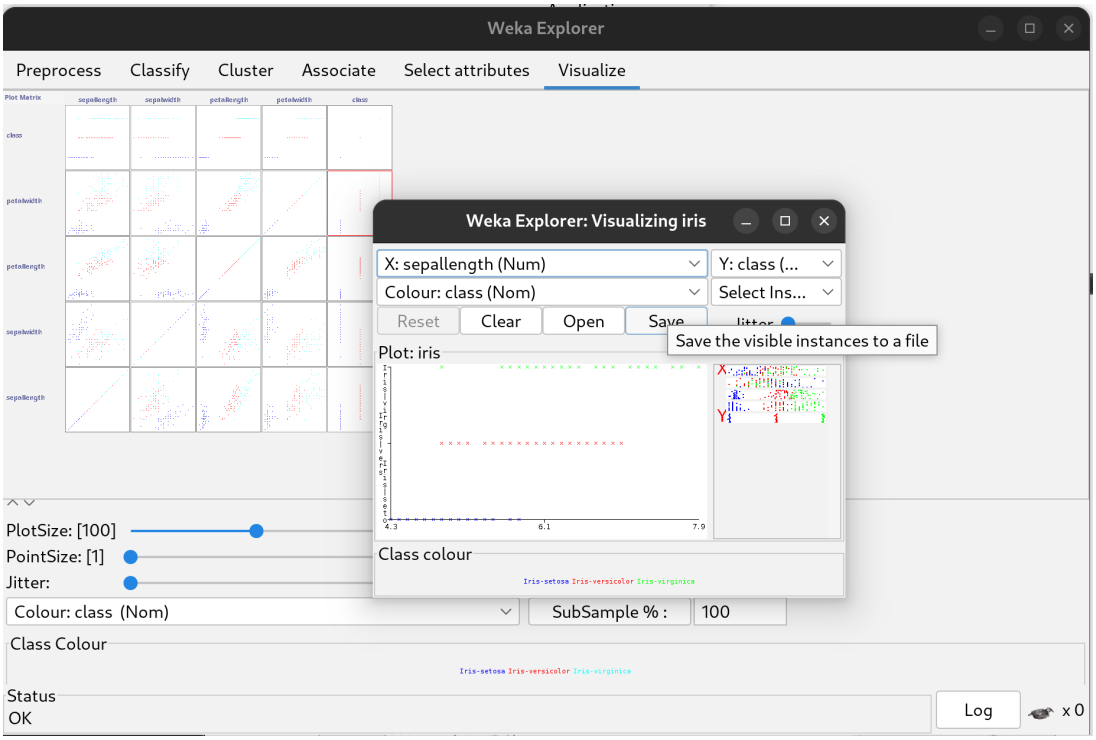
Experiment 5 : Clustering

9 Demonstrate performing clustering on data sets Clustering Tab :

I used the **SimpleKmeans** o perform Clustering on the **iris** dataset :



And for the Visualisation :

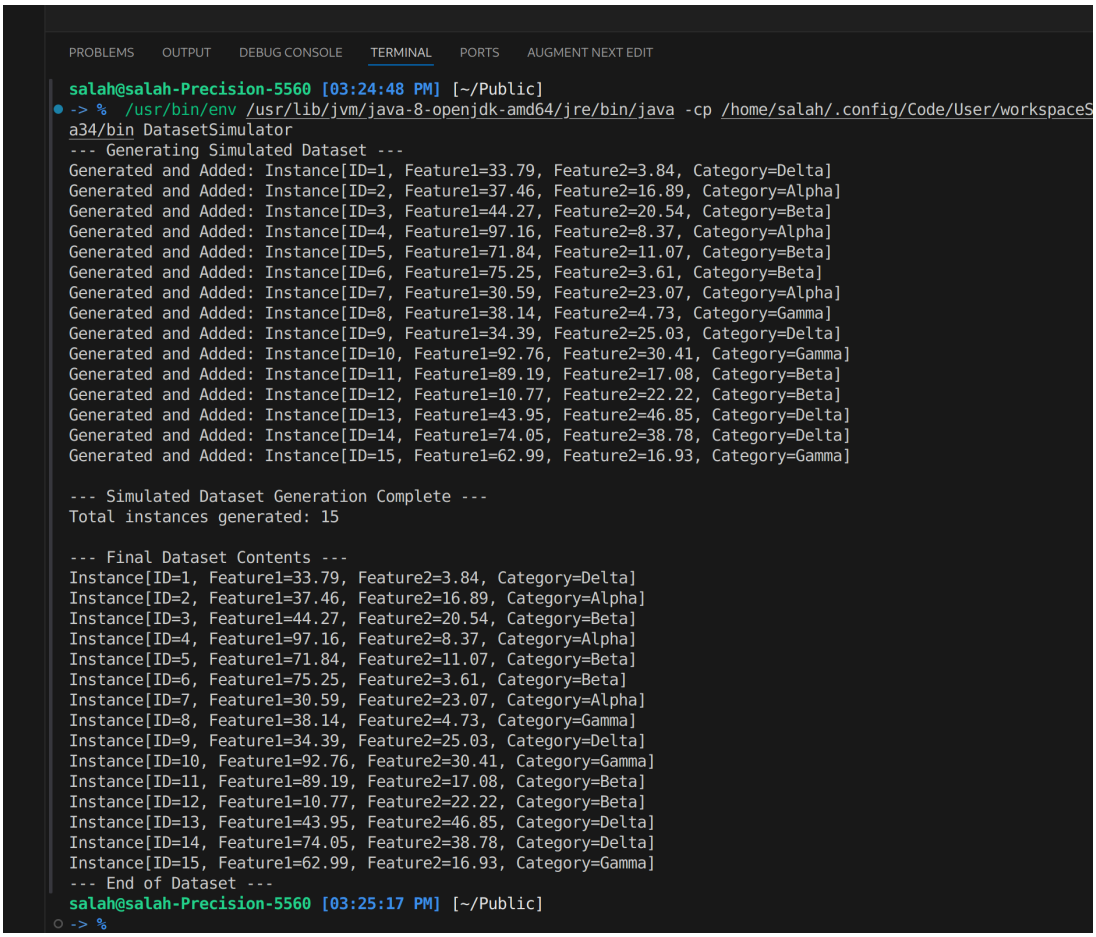

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Weka Explorer: Visualizing iris
X: sepallength (Num)
Y: class (...
Colour: class (Nom)
Select Ins...
Reset
Clear
Open
Save
Save the visible instances to a file
Plot: iris
Class colour
PlotSize: [100]
PointSize: [1]
Jitter:
Colour: class (Nom)
SubSample % :
100
Class Colour
Iris-setosa Iris-versicolor Iris-virginica
Status
OK
Log
x 0

Experiment 6 :

10 Prepare a simulated data set with unique instances :

```java
import java.util.*;


class SimulatedInstance {
    private int id;
    private double numericalFeature1;
    private double numericalFeature2;
    private String categoricalFeature;

    // Constructor
    public SimulatedInstance(int id, double numFeat1, double numFeat2, String catFeat) {
        this.id = id;
        this.numericalFeature1 = numFeat1;
        this.numericalFeature2 = numFeat2;
        this.categoricalFeature = catFeat;
    }

    public int getId() {
        return id;
    }

    public double getNumericalFeature1() {
        return numericalFeature1;
    }

    public double getNumericalFeature2() {
        return numericalFeature2;
    }

    public String getCategoricalFeature() {
        return categoricalFeature;
    }

    @Override
    public String toString() {
        return String.format("Instance[ID=%d, Feature1=%.2f, Feature2=%.2f, Category=%s]",
                             id, numericalFeature1, numericalFeature2, categoricalFeature);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        SimulatedInstance instance = (SimulatedInstance) o;
        return id == instance.id; // Uniqueness defined by ID
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
}
```

and the Output :


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
AUGMENT NEXT EDIT
salah@salah-Precision-5560 [03:24:48 PM] [~/Public]
-> % /usr/bin/env /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java -cp /home/salah/.config/Code/User/workspaceS
a34/bin DatasetSimulator
--- Generating Simulated Dataset ---
Generated and Added: Instance[ID=1, Feature1=33.79, Feature2=3.84, Category=Delta]
Generated and Added: Instance[ID=2, Feature1=37.46, Feature2=16.89, Category=Alpha]
Generated and Added: Instance[ID=3, Feature1=44.27, Feature2=20.54, Category=Beta]
Generated and Added: Instance[ID=4, Feature1=97.16, Feature2=8.37, Category=Alpha]
Generated and Added: Instance[ID=5, Feature1=71.84, Feature2=11.07, Category=Beta]
Generated and Added: Instance[ID=6, Feature1=75.25, Feature2=3.61, Category=Beta]
Generated and Added: Instance[ID=7, Feature1=30.59, Feature2=23.07, Category=Alpha]
Generated and Added: Instance[ID=8, Feature1=38.14, Feature2=4.73, Category=Gamma]
Generated and Added: Instance[ID=9, Feature1=34.39, Feature2=25.03, Category=Delta]
Generated and Added: Instance[ID=10, Feature1=92.76, Feature2=30.41, Category=Gamma]
Generated and Added: Instance[ID=11, Feature1=89.19, Feature2=17.08, Category=Beta]
Generated and Added: Instance[ID=12, Feature1=10.77, Feature2=22.22, Category=Beta]
Generated and Added: Instance[ID=13, Feature1=43.95, Feature2=46.85, Category=Delta]
Generated and Added: Instance[ID=14, Feature1=74.05, Feature2=38.78, Category=Delta]
Generated and Added: Instance[ID=15, Feature1=62.99, Feature2=16.93, Category=Gamma]
--- Simulated Dataset Generation Complete ---
Total instances generated: 15
--- Final Dataset Contents ---
Instance[ID=1, Feature1=33.79, Feature2=3.84, Category=Delta]
Instance[ID=2, Feature1=37.46, Feature2=16.89, Category=Alpha]
Instance[ID=3, Feature1=44.27, Feature2=20.54, Category=Beta]
Instance[ID=4, Feature1=97.16, Feature2=8.37, Category=Alpha]
Instance[ID=5, Feature1=71.84, Feature2=11.07, Category=Beta]
Instance[ID=6, Feature1=75.25, Feature2=3.61, Category=Beta]
Instance[ID=7, Feature1=30.59, Feature2=23.07, Category=Alpha]
Instance[ID=8, Feature1=38.14, Feature2=4.73, Category=Gamma]
Instance[ID=9, Feature1=34.39, Feature2=25.03, Category=Delta]
Instance[ID=10, Feature1=92.76, Feature2=30.41, Category=Gamma]
Instance[ID=11, Feature1=89.19, Feature2=17.08, Category=Beta]
Instance[ID=12, Feature1=10.77, Feature2=22.22, Category=Beta]
Instance[ID=13, Feature1=43.95, Feature2=46.85, Category=Delta]
Instance[ID=14, Feature1=74.05, Feature2=38.78, Category=Delta]
Instance[ID=15, Feature1=62.99, Feature2=16.93, Category=Gamma]
--- End of Dataset ---
salah@salah-Precision-5560 [03:25:17 PM] [~/Public]
-> %

Experiment 7

11 Generate frequent item sets/association rules using apriori :

Step 1 : Data Preprocessing

1.1 Installing Required Package

Use the following command to install the `apyori` library :

```
pip install apyori
```

1.2 Importing Required Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from apyori import apriori
```

1.3 Loading the Dataset

We use a dataset containing market basket data :

```
data = pd.read_csv('Market_Basket_Optimisation.csv', header=None)
```

1.4 Transforming Data into List Format

```
transactions = []

for i in range(0, 7501):
    transactions.append([str(data.values[i, j])
                         for j in range(0, 20)
                         if pd.notna(data.values[i, j])])
```

Step 2 : Training the Apriori Model

We train the Apriori algorithm using minimum support, confidence, and lift thresholds.

```python
rules = apriori(
    transactions=transactions,
    min_support=0.003,
    min_confidence=0.2,
    min_lift=3,
    min_length=2,
    max_length=2
)

results = list(rules)
```

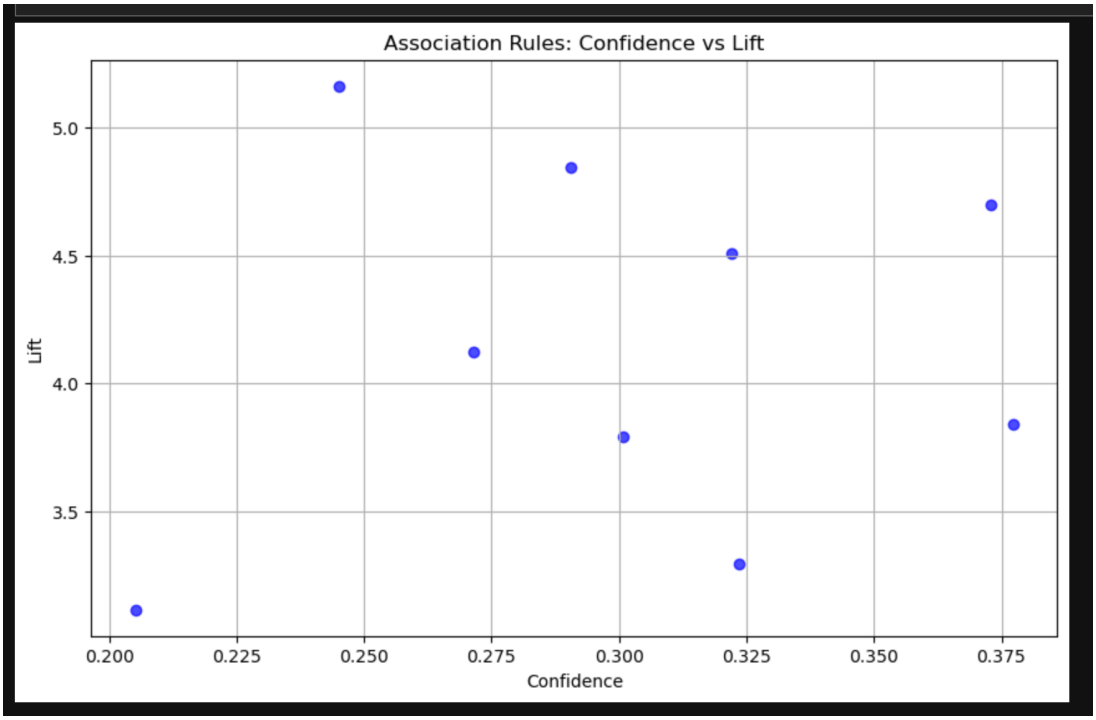
Step 3 : Visualizing the Results

3.1 Formatting the Rules into a DataFrame

We parse the output of the Apriori algorithm to extract useful metrics.

```
          Left_Hand_Side Right_Hand_Side   Support  Confidence      Lift
3         fromage blanc           honey  0.003333    0.245098  5.164271
0           light cream         chicken  0.004533    0.290598  4.843951
2                 pasta        escalope  0.005866    0.372881  4.700812
8                 pasta          shrimp  0.005066    0.322034  4.506672
7     whole wheat pasta       olive oil  0.007999    0.271493  4.122410
5          tomato sauce     ground beef  0.005333    0.377358  3.840659
1  mushroom cream sauce        escalope  0.005733    0.300699  3.790833
4         herb & pepper     ground beef  0.015998    0.323450  3.291994
6           light cream       olive oil  0.003200    0.205128  3.114710
```

3.2 Plotting Lift vs Confidence



Experiment 8 :

12 Calculate chi-square value using python :

1. Importing Required Library

```
from scipy.stats import chi2_contingency
```

2. Defining the Contingency Table

```
# Contingency table
data = [[207, 282, 241],
        [234, 242, 232]]
```

3. Performing the Chi-Square Test

```
stat, p, dof, expected = chi2_contingency(data)
```

4. Interpreting the Result

```
alpha = 0.05
print("p value is", p)

if p <= alpha:
    print("Dependent (reject H0)")
else:
    print("Independent (H0 holds true)")
```

Output

```
Chi-Square Statistic = 4.542228269825232
Degrees of Freedom = 2
Expected Frequencies:
[[223.87343533 266.00834492 240.11821975]
 [217.12656467 257.99165508 232.88178025]]
p-value = 0.1031971404730939
Result: Independent (H0 holds true)
```

Observation

The chi-square test was conducted to determine whether there is a significant association between the two categorical variables in the dataset. The calculated chi-square statistic is 4.54 with 2 degrees of freedom, and the corresponding p-value is 0.1032.

Since the p-value is greater than the chosen significance level $\alpha = 0.05$, we **fail to reject the null hypothesis**.

This indicates that there is **no significant relationship** between the two variables in the contingency table. Thus, the variables are considered **statistically independent**.

Experiment 9 :

13 Naïve Bayes Classification using Python :

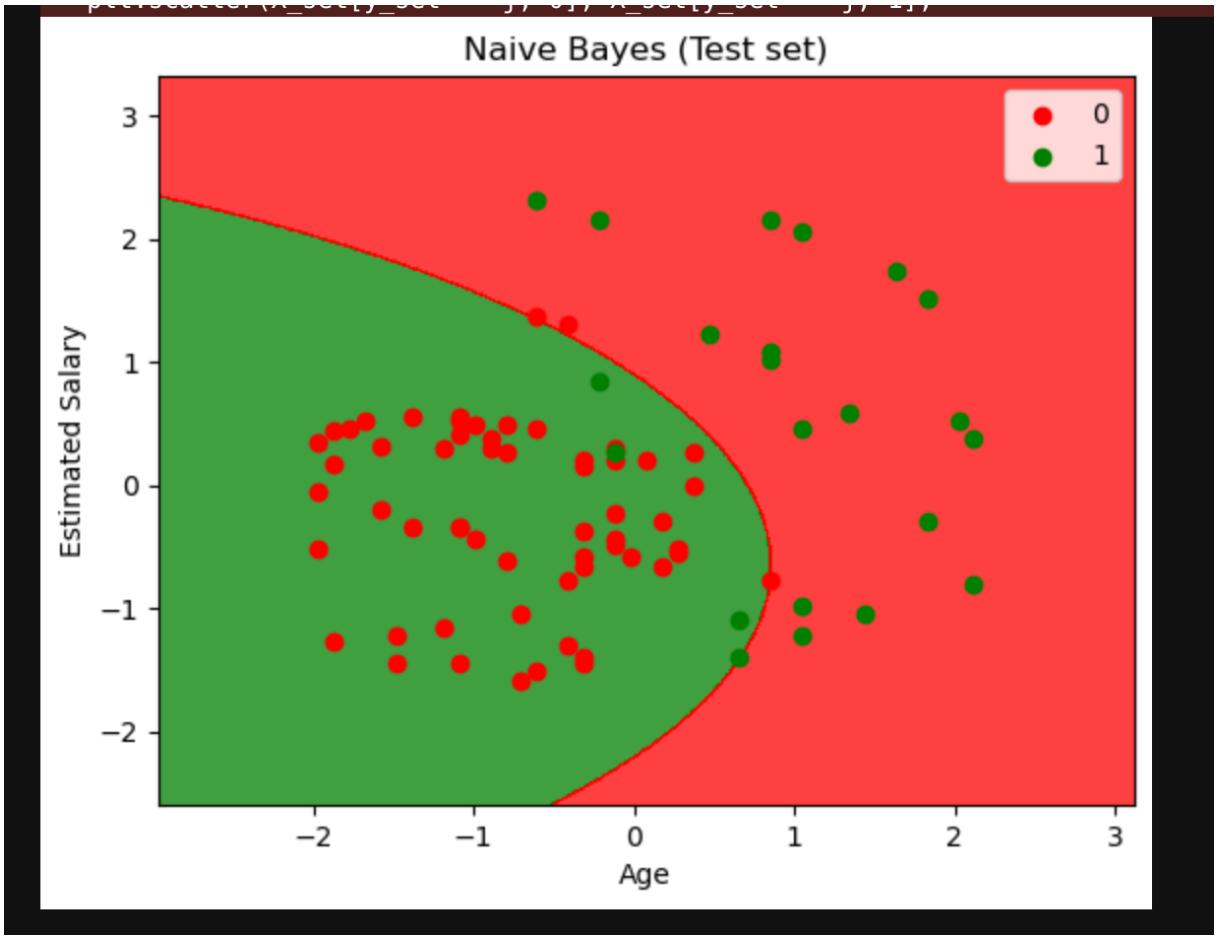
```
[12]: import numpy as np
      import matplotlib.pyplot as plt
      import pandas as pd

•[13]: # Importing the dataset
       dataset = pd.read_csv('Social_Network_Ads.csv')
       X = dataset.iloc[:, [2, 3]].values  # Features: Age and EstimatedSalary
       y = dataset.iloc[:, -1].values  # Target: Purchased (0 or 1)

[14]: # Splitting the dataset into Training and Test sets
      from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

[15]: # Feature Scaling (standardization)
      from sklearn.preprocessing import StandardScaler
      sc = StandardScaler()
      X_train = sc.fit_transform(X_train)
      X_test = sc.transform(X_test)

[16]: # Training the Naive Bayes model on the Training set
      from sklearn.naive_bayes import GaussianNB
      classifier = GaussianNB()
      classifier.fit(X_train, y_train)

[16]: ▾ GaussianNB
      GaussianNB()

[17]: # Predicting the Test set results
      y_pred = classifier.predict(X_test)

[18]: # Making the Confusion Matrix and calculating Accuracy
      from sklearn.metrics import confusion_matrix, accuracy_score
      cm = confusion_matrix(y_test, y_pred)
      ac = accuracy_score(y_test, y_pred)
```

Output

the confusion matrix and the accuracy score :

```
•[19]: print("Confusion Matrix:")
       print(cm)
       print("Accuracy Score:", ac)

       Confusion Matrix:
       [[55  3]
        [ 4 18]]
       Accuracy Score: 0.9125
```

13.1 Visualisation on the Test set :



Observation

The Naive Bayes model has performed well in classifying the purchase behavior of users, achieving a high accuracy of 91.25%. However, there are some misclassifications :

3 false positives : The model incorrectly predicted that users would make a purchase when they did not.

4 false negatives : The model incorrectly predicted that users would not make a purchase when they actually did.

Experiment 10 :

14 Naïve Bayes Classification using Python :

```
[4]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import re
     from mlxtend.frequent_patterns import apriori
     from mlxtend.frequent_patterns import association_rules
     from mlxtend.preprocessing import TransactionEncoder
     from mpl_toolkits.mplot3d import Axes3D
     import networkx as nx
     basket = pd.read_csv("Groceries_dataset.csv")
     display(basket.head())
```

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

```
[ ]:
```

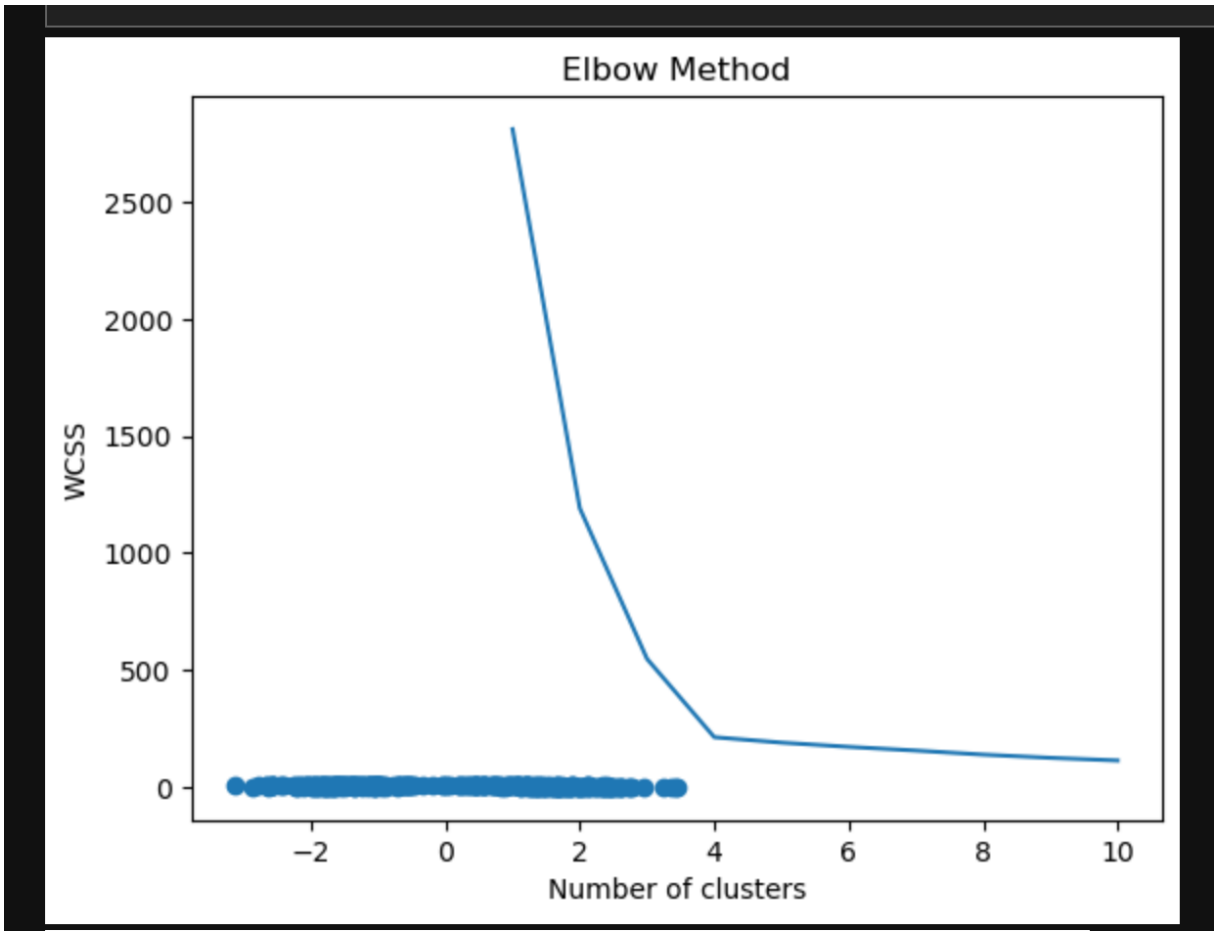
Experiment 11 :

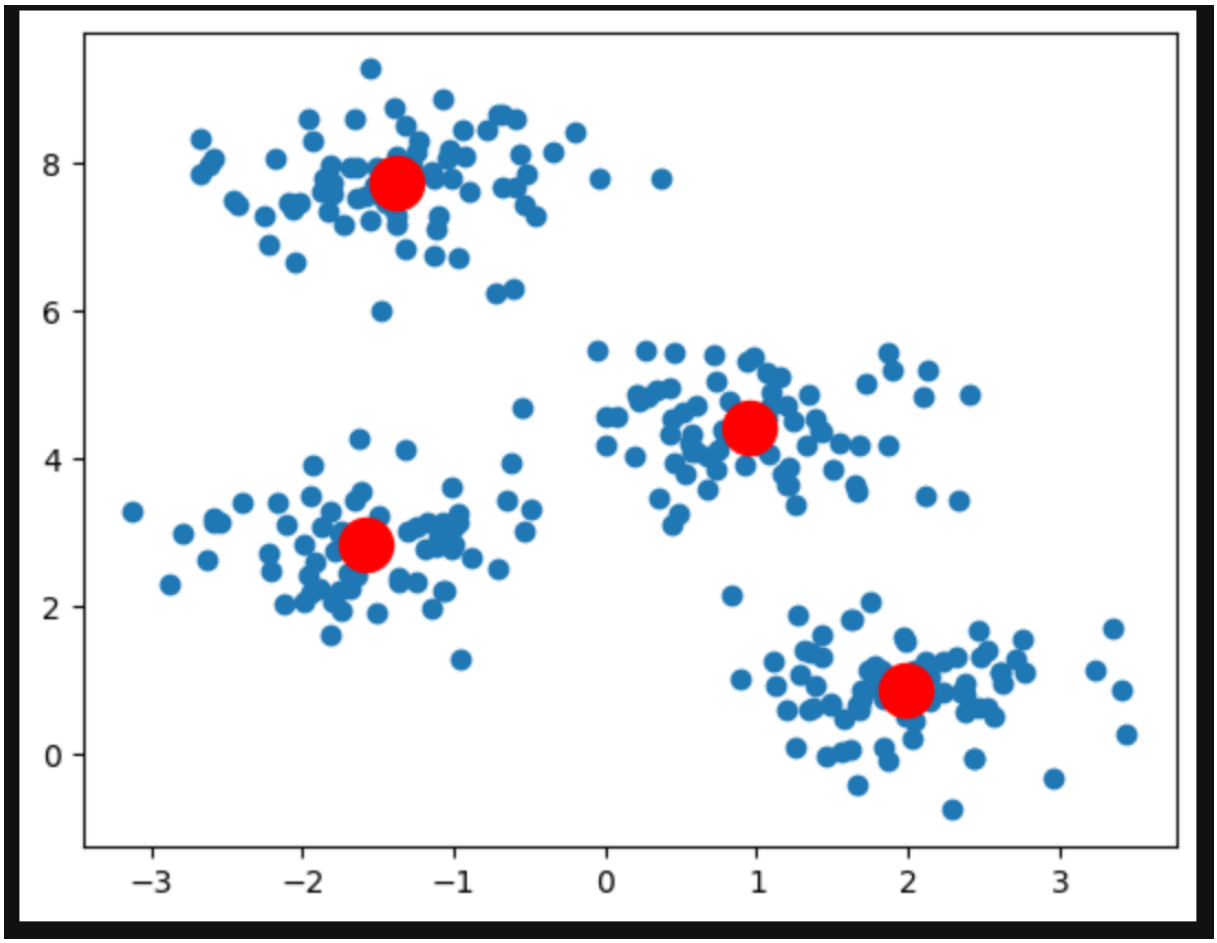
15 Cluster analysis using simple k-means algorithm Python :

Python Code :

[1]:
```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate sample data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Scatter plot of the generated data
plt.scatter(X[:, 0], X[:, 1])

# Elbow method to find optimal number of clusters
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the elbow graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # Within-cluster sum of squares
plt.show()

# Fit KMeans with the optimal number of clusters (4 clusters)
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0)
pred_y = kmeans.fit_predict(X)

# Scatter plot with cluster centers
plt.scatter(X[:, 0], X[:, 1])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.show()
```

Visualisation :


Elbow Method
WCSS
2500
2000
1500
1000
500
0
−2
0
2
4
6
8
10
Number of clusters



Experiment 12 :

16 Compute/display dissimilarity matrix using python :

— Cosine Similarity

— Jaccard Similarity and Jaccard Distance

16.1 Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors. It is computed using the following formula :

$$Similarity = \frac{A \cdot B}{\|A\|\|B\|}$$

Where :

— $A \cdot B$ is the dot product of vectors A and B, calculated as the sum of element-wise products of A and B.

— $\|A\|$ is the L2 norm of vector A, calculated as the square root of the sum of the squares of its elements.

— $\|B\|$ is the L2 norm of vector B, calculated similarly to A.

Example : Given two vectors :

$$A = [2, 1, 2, 3, 2, 9]$$

$$B = [3, 4, 2, 4, 5, 5]$$

The Python code to compute the Cosine Similarity is as follows :

```
import numpy as np
from numpy.linalg import norm

A = np.array([2, 1, 2, 3, 2, 9])
B = np.array([3, 4, 2, 4, 5, 5])

cosine = np.dot(A, B) / (norm(A) * norm(B))
print("Cosine Similarity:", cosine)
```

Output :

$$CosineSimilarity = 0.897$$

17 Jaccard Similarity

The Jaccard similarity is a statistic used to measure the similarity between two sets. It is calculated using the formula :

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Where :

— $A \cap B$ is the intersection of sets A and B.

— $A \cup B$ is the union of sets A and B.

Example : Let $A = \{1, 2, 3, 5, 7\}$ and $B = \{1, 2, 4, 8, 9\}$. To calculate the Jaccard similarity :

— Intersection : $A \cap B = \{1, 2\}$

— Union : $A \cup B = \{1, 2, 3, 5, 7, 4, 8, 9\}$

Thus, the Jaccard similarity is :

$$J(A, B) = \frac{2}{8} = 0.25$$

The Python code to compute the Jaccard Similarity is :

```
def jaccard_similarity(A, B):
    nominator = A.intersection(B)
    denominator = A.union(B)
    return len(nominator) / len(denominator)

A = {1, 2, 3, 5, 7}
B = {1, 2, 4, 8, 9}

similarity = jaccard_similarity(A, B)
print("Jaccard Similarity:", similarity)
```

Output :

$$JaccardSimilarity = 0.25$$

18 Jaccard Distance

Jaccard Distance is the dissimilarity measure and is calculated using the following formula :

$$d_J(A, B) = 1 - J(A, B)$$

Where $J(A, B)$ is the Jaccard similarity.

Example : Using the same sets A and B :

$$d_J(A, B) = 1 - 0.25 = 0.6$$

The Python code to compute the Jaccard Distance is :

```
def jaccard_distance(A, B):
    nominator = A.symmetric_difference(B)
    denominator = A.union(B)
    return len(nominator) / len(denominator)

distance = jaccard_distance(A, B)
print("Jaccard Distance:", distance)
```

Output :

$$JaccardDistance = 0.6$$

```
M11: 2, M01: 1, M10: 2, M00: 1
Jaccard Similarity: 0.4
Jaccard Distance: 0.6
```

```
[5]: # Define a list of sets or vectors (for example, A, B, C)
```

Experiment 13 :

19 Visualize the data sets using matplotlib in python.(Histogram Box Plot, Bar chart, PieChart) :

19.1 Line Plot

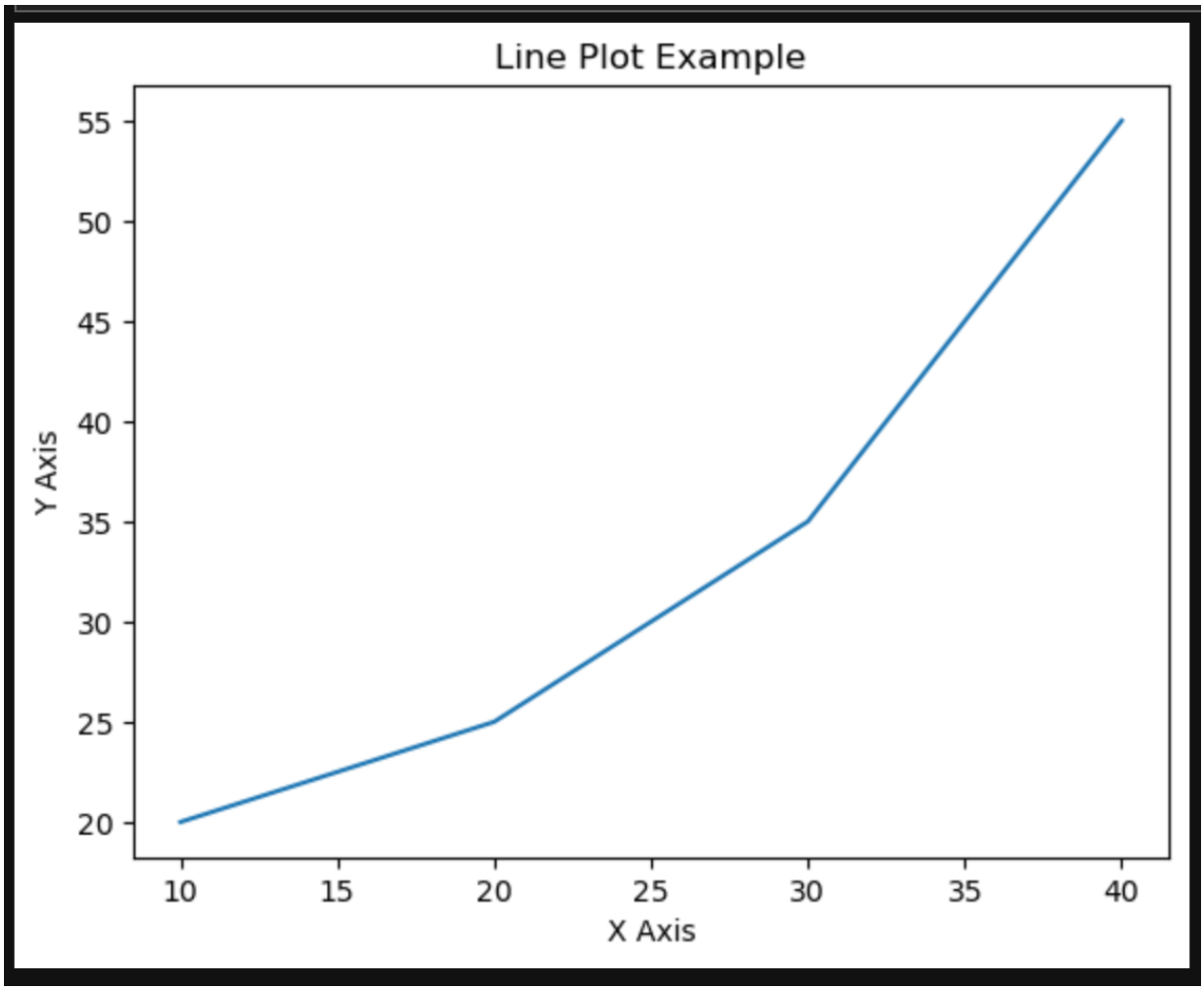
A line plot is used to display data points connected by straight lines. In this case, we will plot the data using two lists, x and y, and visualize the relationship between these data points.

19.1.1 Code for Line Plot

The following Python code generates a simple line plot :

```
import matplotlib.pyplot as plt

# initializing the data for the line plot
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]

# plotting the data (Line plot)
plt.plot(x, y)
plt.title("Line Plot Example")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.show()
```

Listing 10 – Python code for Line Plot



19.1.2 Explanation

In the code above :

— We define two lists, x and y, representing the data points.

— We use `plt.plot(x, y)` to plot the data.

— Titles and labels are added to the plot using `plt.title()`, `plt.xlabel()`, and `plt.ylabel()`.

— `plt.show()` displays the plot.

19.2 Histogram

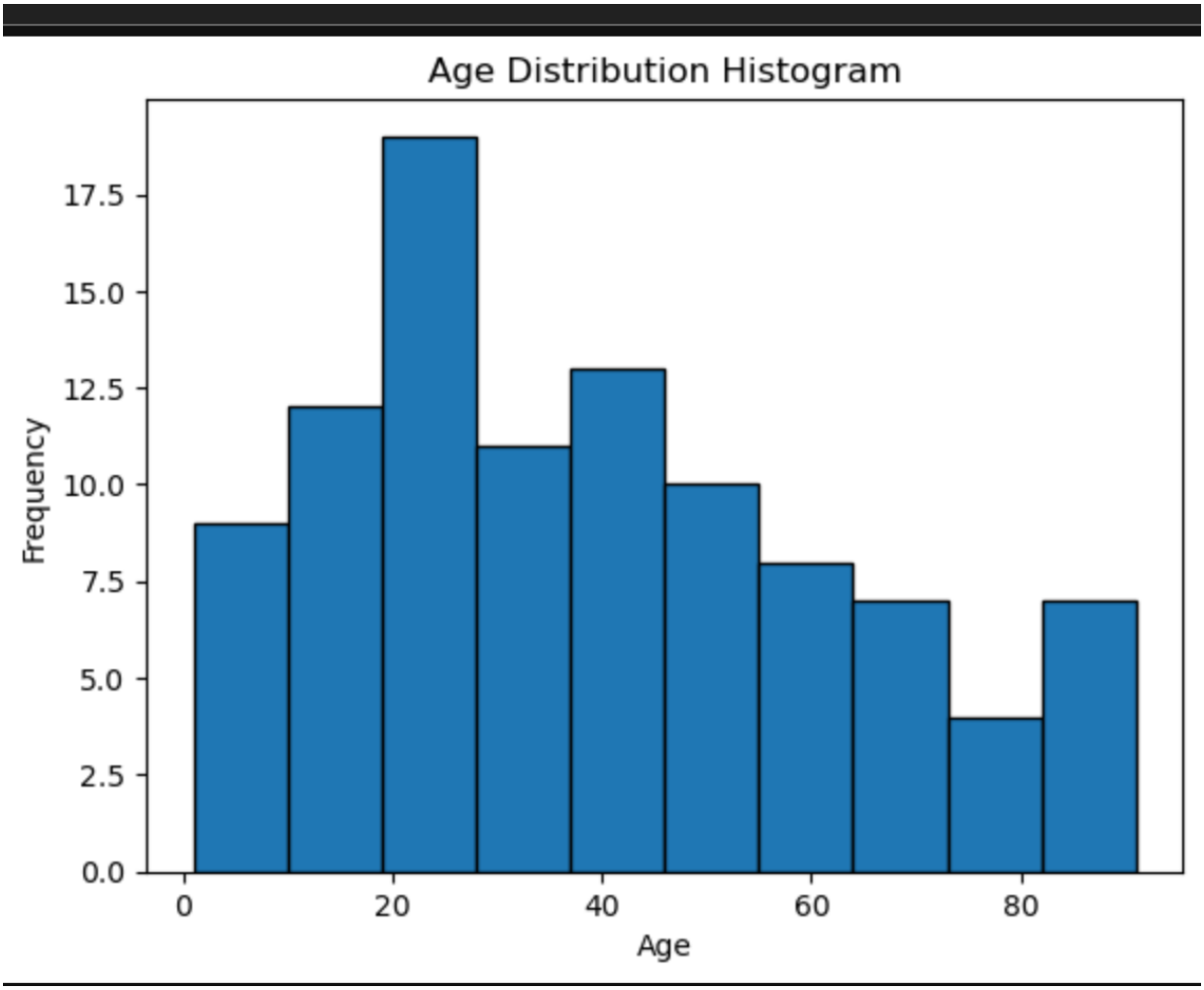
A histogram is a graphical representation of the distribution of numerical data. The data is divided into bins, and the frequency of data points within each bin is plotted as bars.

19.2.1 Code for Histogram

We will now create a histogram using a set of data that represents the ages of 100 individuals. The Python code for this is as follows :

```
import matplotlib.pyplot as plt

# Data representing the ages of 100 individuals
x = [1,1,2,3,3,5,7,8,9,10,
10,11,11,13,13,15,16,17,18,18,
18,19,20,21,21,23,24,24,25,25,
25,25,26,26,26,27,27,27,27,27,
29,30,30,31,33,34,34,34,35,36,
36,37,37,38,38,39,40,41,41,42,
43,44,45,45,46,47,48,48,49,50,
51,52,53,54,55,55,56,57,58,60,
61,63,64,65,66,68,70,71,72,74,
75,77,81,83,84,87,89,90,90,91]

# Plotting the histogram
plt.hist(x, bins=10, edgecolor='black')  # 'bins=10' defines 10 bins for
    the histogram
plt.title("Age Distribution Histogram")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

Listing 11 – Python code for Histogram



19.2.2 Explanation

In the histogram code :

— The list x contains the ages of 100 individuals.

— We use `plt.hist(x, bins=10)` to create the histogram with 10 bins.

— The `edgecolor='black'` argument makes the borders of the bars black for better visibility.

— Titles and labels are added using `plt.title()`, `plt.xlabel()`, and `plt.ylabel()`.

— `plt.show()` displays the histogram.