



Université Abdelmalek Essadi  
Faculté des Sciences et Techniques - Tanger  
Département Génie Informatique



## **Rapport**

**Filière :**

**« Logiciels et systèmes intelligents »**

**LSI**

**SID & DM LSI2 2025**

---

**Travaux pratiques BI & DM**

---

**Réalisé par :**

Mohamed Amine BAHASSOU

Hodaifa ECHFFANI

**Encadré par :**

Prof. Abdelhadi FENNAN

# Table de matière

<b>1. Experiment 1 - Creation of Dataware house.....</b>	<b>3</b>
a. Creating the databases.....	3
<b>2. Experiment 2 - Weka.....</b>	<b>7</b>
<b>3. Experiment-3 : Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets.....</b>	<b>17</b>
<b>4. Experiment 4: Demonstrate performing classification on data sets.....</b>	<b>18</b>
<b>5. Experiment:5 Demonstrate performing clustering on data sets Clustering Tab.....</b>	<b>26</b>
<b>6. Experiment-6: Write a java program to prepare a simulated data set with unique instances.....</b>	<b>31</b>
<b>7. Experiment-7: Frequent Itemset and Association Rule Generation.....</b>	<b>34</b>
<b>8. Experiment 8 : Chi-Square Test Implementation.....</b>	<b>35</b>
<b>9. Experiment-9: Write a program of Naïve Bayesian classification using python programming language.....</b>	<b>36</b>
<b>10. Experiment-10: Implement a java program to perform Apriori algorithm.....</b>	<b>39</b>
<b>11. Experiment 11: Cluster Analysis using Simple K-Means Algorithm (Python).....</b>	<b>41</b>
<b>12. Experiment 12 :Similarity and Dissimilarity Ana-lysis.....</b>	<b>45</b>
<b>13. 13 Experiment - Data Visualization with Matplotlib.....</b>	<b>48</b>

# 1. Experiment 1 - Creation of Dataware house

## a. Creating the databases

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| HireBase |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)
```

## b. Creating Table

```
mysql> SHOW TABLES
-> ;
+-----+
| Tables_in_HireBase |
+-----+
| Customer |
| Hire |
| Van |
+-----+
3 rows in set (0.00 sec)

mysql>
```

### i. Verify Customer Data:

```
mysql> SELECT * FROM Customer LIMIT 10;
+-----+-----+-----+-----+-----+-----+-----+
| CustomerId | CustomerName | DateOfBirth | Town | TelephoneNo | DrivingLicenceNo | Occupation |
+-----+-----+-----+-----+-----+-----+-----+
| N01 | Customer01 | 2000-01-01 | Town01 | Phone01 | Licence01 | Occupation01 |
| N02 | Customer02 | 2000-01-02 | Town02 | Phone02 | Licence02 | Occupation02 |
| N03 | Customer03 | 2000-01-03 | Town03 | Phone03 | Licence03 | Occupation03 |
| N04 | Customer04 | 2000-01-04 | Town04 | Phone04 | Licence04 | Occupation04 |
| N05 | Customer05 | 2000-01-05 | Town05 | Phone05 | Licence05 | Occupation05 |
| N06 | Customer06 | 2000-01-06 | Town06 | Phone06 | Licence06 | Occupation06 |
| N07 | Customer07 | 2000-01-07 | Town07 | Phone07 | Licence07 | Occupation07 |
| N08 | Customer08 | 2000-01-08 | Town08 | Phone08 | Licence08 | Occupation08 |
| N09 | Customer09 | 2000-01-09 | Town09 | Phone09 | Licence09 | Occupation09 |
| N10 | Customer10 | 2000-01-10 | Town10 | Phone10 | Licence10 | Occupation10 |
+-----+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

### ii. Verify Van Data:

```
mysql> SELECT * FROM Van LIMIT 10;
```

RegNo	Make	Model	Year	Colour	CC	Class
Reg1	Make1	Model1	2009	White	2500	Medium
Reg10	Make10	Model10	2010	White	2500	Medium
Reg11	Make11	Model11	2011	White	3000	Large
Reg12	Make12	Model12	2008	White	2000	Small
Reg13	Make13	Model13	2009	Black	2500	Medium
Reg14	Make14	Model14	2010	Black	3000	Large
Reg15	Make15	Model15	2011	White	2000	Small
Reg16	Make16	Model16	2008	White	2500	Medium
Reg17	Make17	Model17	2009	White	3000	Large
Reg18	Make18	Model18	2010	Black	2000	Small

```
10 rows in set (0.00 sec)
```

### iii. Verify Hire Data:

```
mysql> SELECT * FROM Hire LIMIT 10;
```

HireId	HireDate	CustomerId	RegNo	NoOfDays	VanHire	SatNavHire	Insurance	DamageWaiver	TotalBill
H0001	2011-01-01	N01	Reg1	1	100.00	10.00	20.00	40.00	170.00
H0002	2011-01-02	N02	Reg2	2	200.00	20.00	40.00	80.00	340.00
H0003	2011-01-03	N03	Reg3	3	300.00	30.00	60.00	120.00	510.00
H0004	2011-01-04	N04	Reg4	1	100.00	10.00	20.00	40.00	170.00
H0005	2011-01-05	N05	Reg5	2	200.00	20.00	40.00	80.00	340.00
H0006	2011-01-06	N06	Reg6	3	300.00	30.00	60.00	120.00	510.00
H0007	2011-01-07	N07	Reg7	1	100.00	10.00	20.00	40.00	170.00
H0008	2011-01-08	N08	Reg8	2	200.00	20.00	40.00	80.00	340.00
H0009	2011-01-09	N09	Reg9	3	300.00	30.00	60.00	120.00	510.00
H0010	2011-01-10	N10	Reg10	1	100.00	10.00	20.00	40.00	170.00

```
10 rows in set (0.00 sec)
```

### c. Create the Data Warehouse

```
+-----+
| Database |
+-----+
| HireBase |
| TopHireDW |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
6 rows in set (0.00 sec)

mysql> 
```

### d. Creating Table

```
mysql> SHOW TABLES
-> ;
+-----+
| Tables_in_TopHireDW |
+-----+
| DimCustomer          |
| DimDate              |
| DimVan               |
| FactHire             |
+-----+
4 rows in set (0.00 sec)
```

i. Verify Date Dimension (DimDate):

```
mysql> SELECT * FROM DimDate LIMIT 15;
+-----+-----+-----+-----+-----+
| DateKey | Year   | Month  | Date       | DateString |
+-----+-----+-----+-----+-----+
| 0       | Unknown | Unknown | 0001-01-01 | Unknown    |
| 20060101 | 2006   | 2006-01 | 2006-01-01 | 2006-01-01 |
| 20060102 | 2006   | 2006-01 | 2006-01-02 | 2006-01-02 |
| 20060103 | 2006   | 2006-01 | 2006-01-03 | 2006-01-03 |
| 20060104 | 2006   | 2006-01 | 2006-01-04 | 2006-01-04 |
| 20060105 | 2006   | 2006-01 | 2006-01-05 | 2006-01-05 |
| 20060106 | 2006   | 2006-01 | 2006-01-06 | 2006-01-06 |
| 20060107 | 2006   | 2006-01 | 2006-01-07 | 2006-01-07 |
| 20060108 | 2006   | 2006-01 | 2006-01-08 | 2006-01-08 |
| 20060109 | 2006   | 2006-01 | 2006-01-09 | 2006-01-09 |
| 20060110 | 2006   | 2006-01 | 2006-01-10 | 2006-01-10 |
| 20060111 | 2006   | 2006-01 | 2006-01-11 | 2006-01-11 |
| 20060112 | 2006   | 2006-01 | 2006-01-12 | 2006-01-12 |
| 20060113 | 2006   | 2006-01 | 2006-01-13 | 2006-01-13 |
| 20060114 | 2006   | 2006-01 | 2006-01-14 | 2006-01-14 |
+-----+-----+-----+-----+-----+
15 rows in set (0.00 sec)
```

ii. Verify Customer Dimension (DimCustomer):

```
mysql> SELECT * FROM DimCustomer LIMIT 10;
```

CustomerKey	CustomerId	CustomerName	DateOfBirth	Town	TelephoneNo	DrivingLicenceNo	Occupation
1	N01	Customer01	2000-01-01	Town01	Phone01	Licence01	Occupation01
2	N02	Customer02	2000-01-02	Town02	Phone02	Licence02	Occupation02
3	N03	Customer03	2000-01-03	Town03	Phone03	Licence03	Occupation03
4	N04	Customer04	2000-01-04	Town04	Phone04	Licence04	Occupation04
5	N05	Customer05	2000-01-05	Town05	Phone05	Licence05	Occupation05
6	N06	Customer06	2000-01-06	Town06	Phone06	Licence06	Occupation06
7	N07	Customer07	2000-01-07	Town07	Phone07	Licence07	Occupation07
8	N08	Customer08	2000-01-08	Town08	Phone08	Licence08	Occupation08
9	N09	Customer09	2000-01-09	Town09	Phone09	Licence09	Occupation09
10	N10	Customer10	2000-01-10	Town10	Phone10	Licence10	Occupation10

```
10 rows in set (0.00 sec)
```

### iii. Verify Van Dimension (DimVan):

```
mysql> SELECT * FROM DimVan LIMIT 10;
```

VanKey	RegNo	Make	Model	Year	Colour	CC	Class
1	Reg1	Make1	Model1	2009	White	2500	Medium
2	Reg10	Make10	Model10	2010	White	2500	Medium
3	Reg11	Make11	Model11	2011	White	3000	Large
4	Reg12	Make12	Model12	2008	White	2000	Small
5	Reg13	Make13	Model13	2009	Black	2500	Medium
6	Reg14	Make14	Model14	2010	Black	3000	Large
7	Reg15	Make15	Model15	2011	White	2000	Small
8	Reg16	Make16	Model16	2008	White	2500	Medium
9	Reg17	Make17	Model17	2009	White	3000	Large
10	Reg18	Make18	Model18	2010	Black	2000	Small

```
10 rows in set (0.00 sec)
```

### iii ) ETL Implementation

```
mysql> INSERT INTO FactHire (
  -> SnapshotDateKey,
  -> HireDateKey,
  -> CustomerKey,
  -> VanKey,
  -> HireId,
  ->
  -> -- Measures from source
  -> NoOfDays,
  -> VanHire,
  -> SatNavHire,
  -> Insurance,
  -> DamageWaiver,
  -> TotalBill
  -> )...
```

#### 1. Check Row Count

```
mysql> SELECT COUNT(*) FROM FactHire;
+-----+
| COUNT(*) |
+-----+
|      1000 |
+-----+
1 row in set (0.00 sec)
```

## 2. Inspect Sample Data:

```
mysql> SELECT * FROM FactHire LIMIT 10;
```

SnapshotDateKey	HireDateKey	CustomerKey	VanKey	HireId	NoOfDays	VanHire	SatNavHire	Insurance	DamageWaiver	TotalBill
20110101	20110101	1	1	H0801	3	300.00	30.00	60.00	120.00	510.00
20110101	20110101	1	1	H0601	1	100.00	10.00	20.00	40.00	170.00
20110101	20110101	1	1	H0401	2	200.00	20.00	40.00	80.00	340.00
20110101	20110101	1	1	H0201	3	300.00	30.00	60.00	120.00	510.00
20110101	20110101	1	1	H0001	1	100.00	10.00	20.00	40.00	170.00
20110102	20110102	2	12	H0802	1	100.00	10.00	20.00	40.00	170.00
20110102	20110102	2	12	H0602	2	200.00	20.00	40.00	80.00	340.00
20110102	20110102	2	12	H0402	3	300.00	30.00	60.00	120.00	510.00
20110102	20110102	2	12	H0202	1	100.00	10.00	20.00	40.00	170.00
20110102	20110102	2	12	H0002	2	200.00	20.00	40.00	80.00	340.00

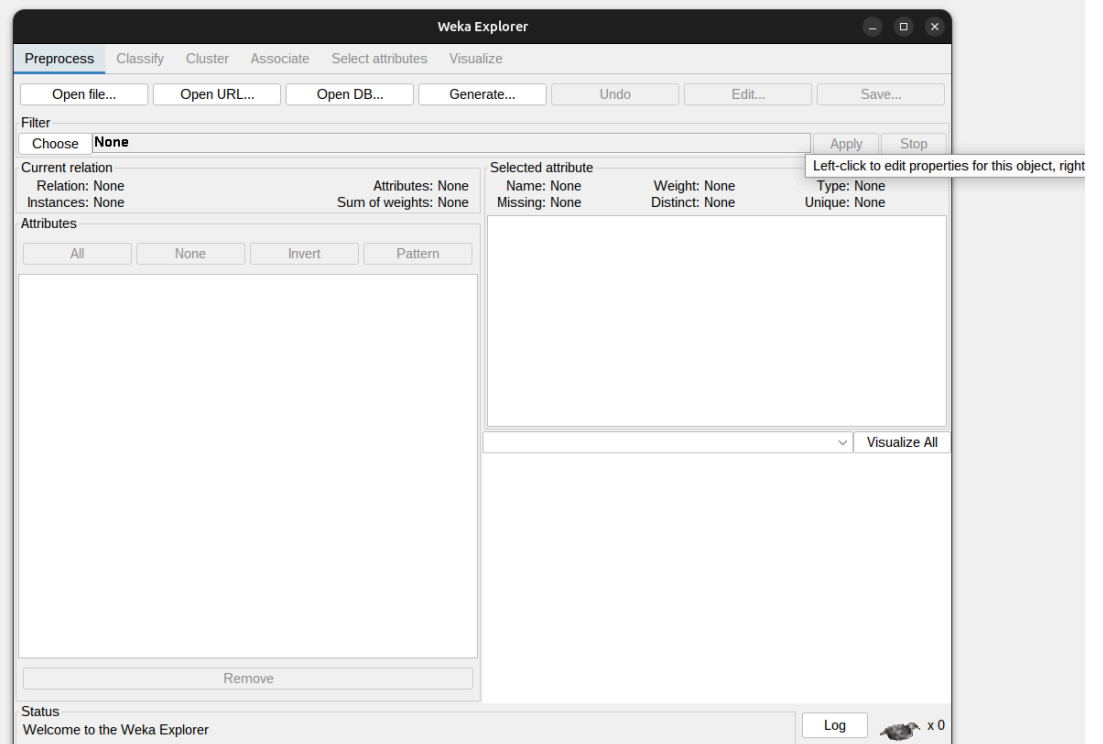
```
10 rows in set (0.00 sec)
```

## 2. Experiment 2 - Weka

### i. Installation

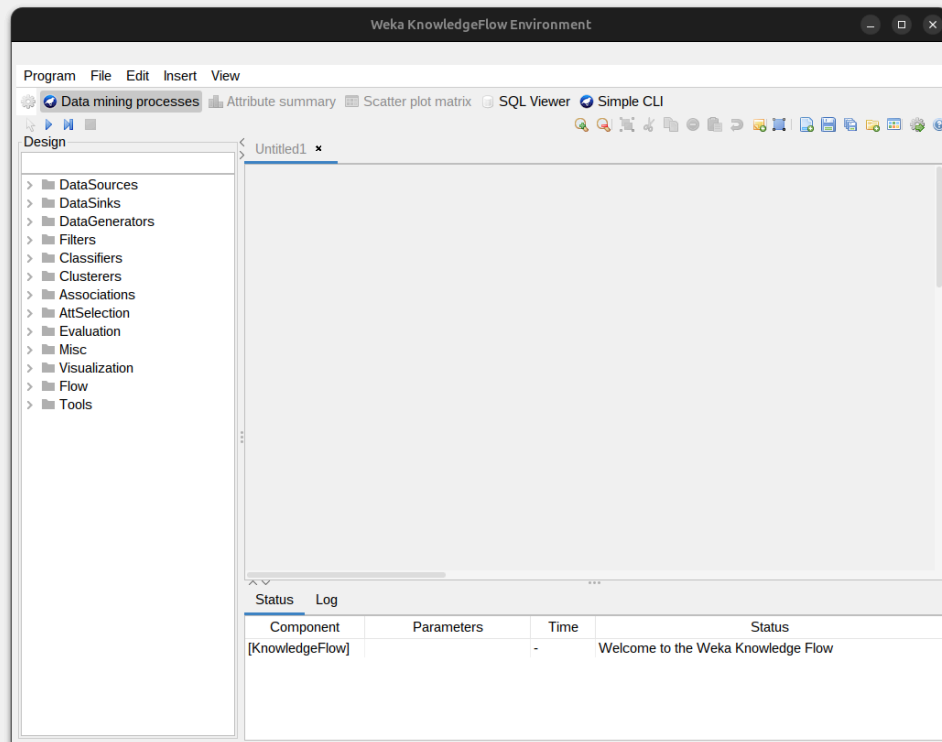
### ii. WEKA Toolkit Features

**Explorer** : Main interface for data preprocessing and analysis

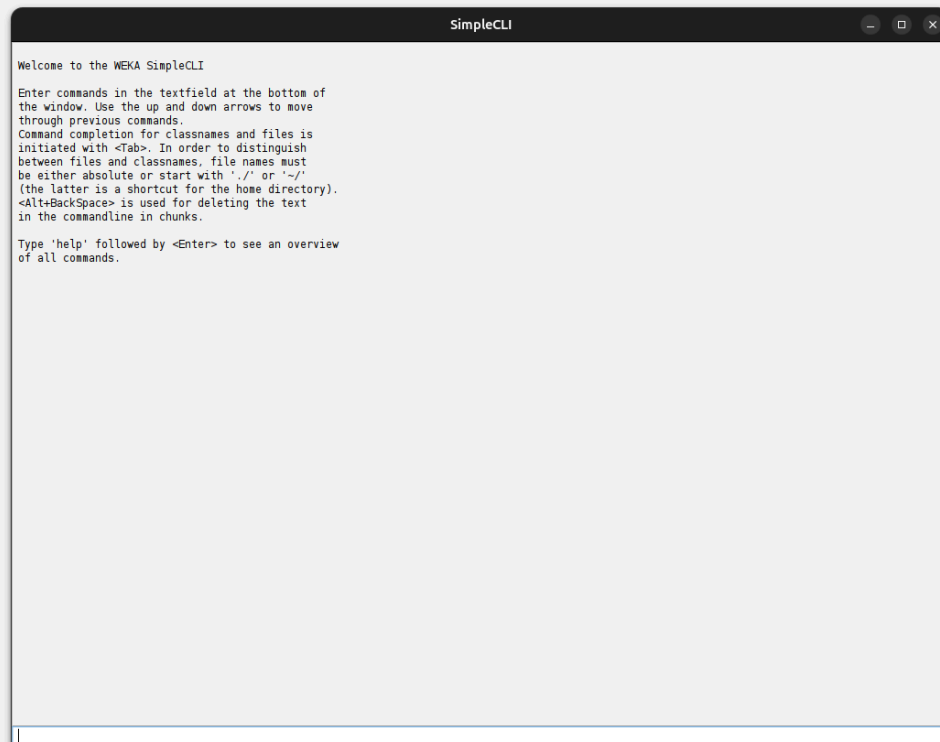


**KnowledgeFlow** : Drag-and-drop workflow design

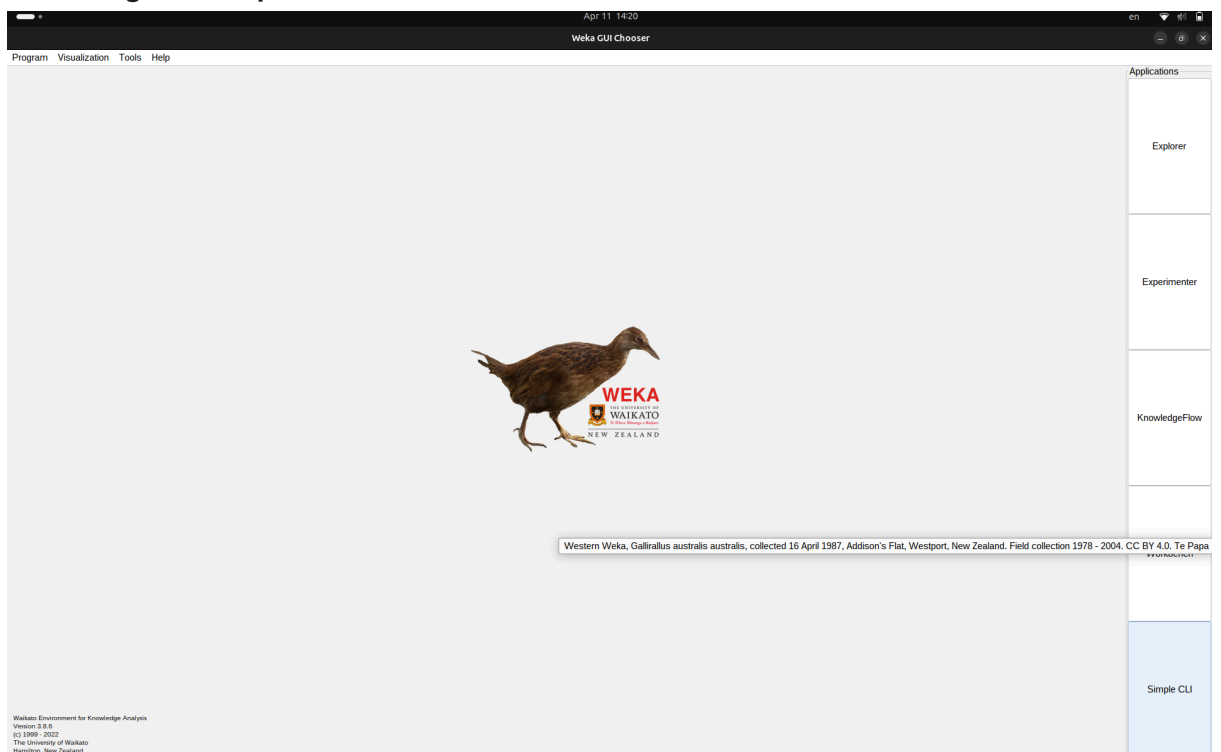


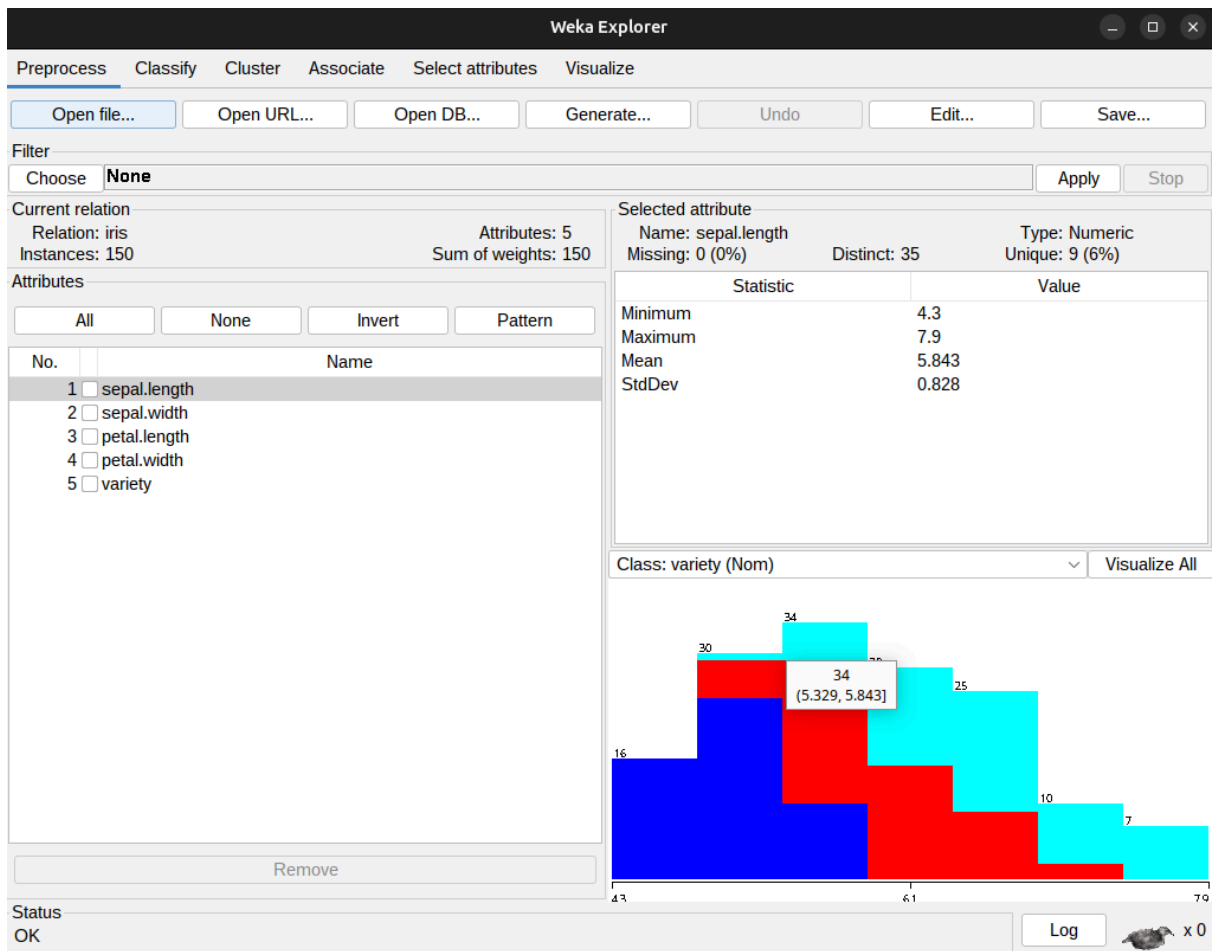


**SimpleCLI** : a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface.

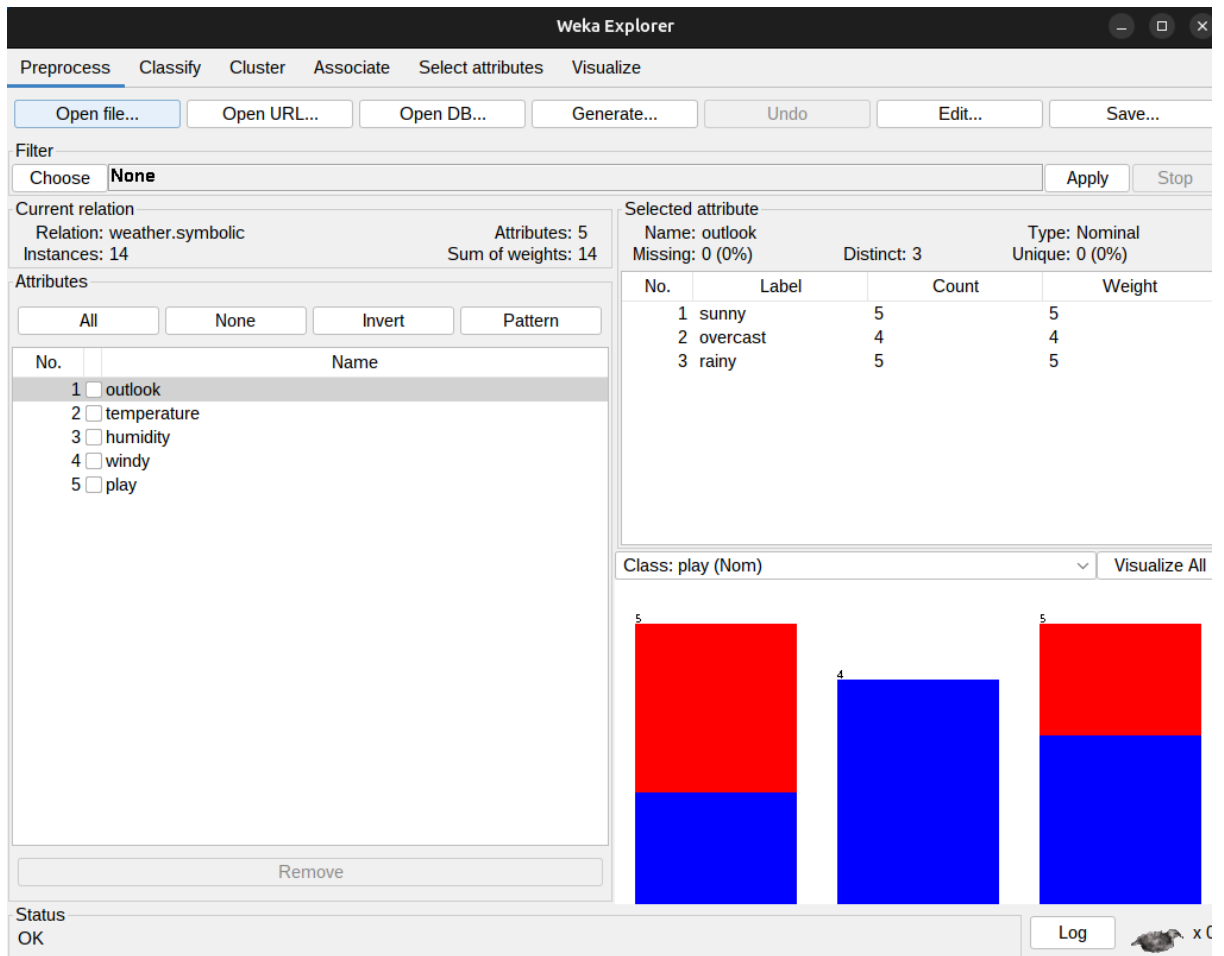


### iii. Navigate the options available in the WEKA:






















#### iv. Study the ARFF file format

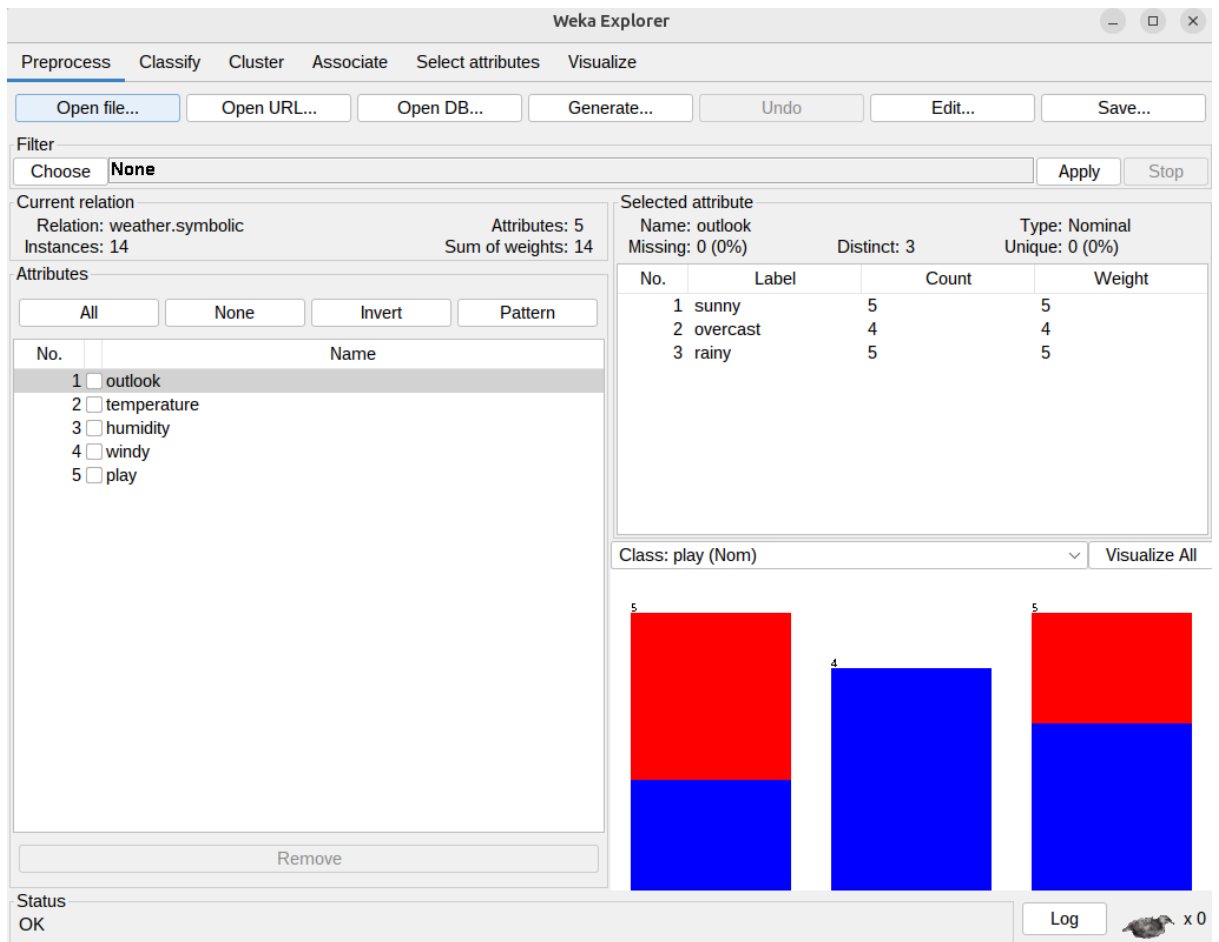


## vi : Dataset Exploration

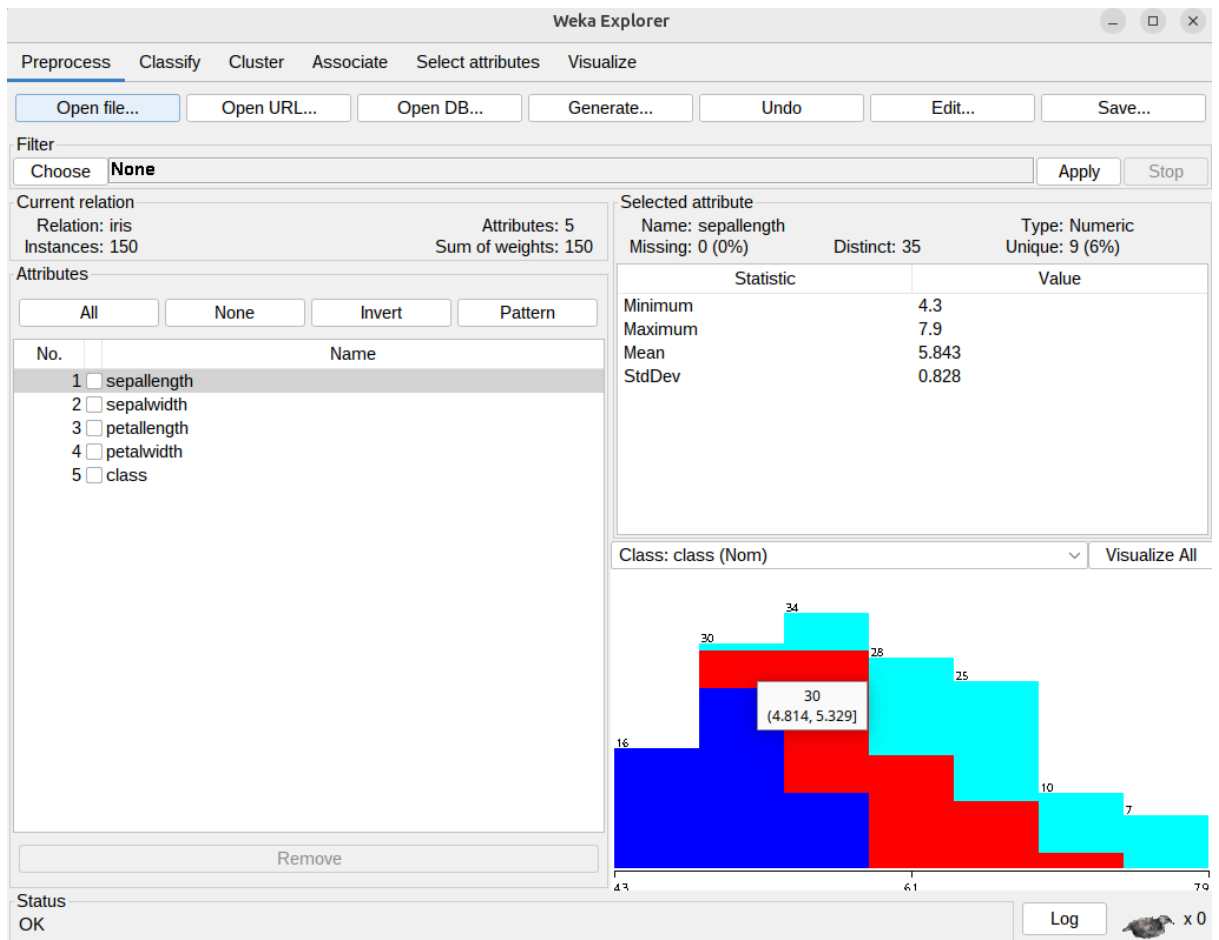
<div> <div> <div></div> <div></div> </div> <div> <div>Home</div> <div>/ weka-3-8-6-azul-zulu-linux</div> <div>/ weka-3-8-6</div> <div>/ data</div> </div> </div> <div> <div></div> <div></div> </div>				<div> <div>BB</div> <div></div> <div></div> <div></div> <div></div> </div>	
Name		Size	Modified		
 airline.arff		2.4 kB	25 Jan 2022	☆	
 breast-cancer.arff		29.4 kB	25 Jan 2022	☆	
 contact-lenses.arff		2.9 kB	25 Jan 2022	☆	
 cpu.arff		5.6 kB	25 Jan 2022	☆	
 cpu.with.vendor.arff		7.0 kB	25 Jan 2022	☆	
 credit-g.arff		162.3 kB	25 Jan 2022	☆	
 diabetes.arff		37.4 kB	25 Jan 2022	☆	
 glass.arff		17.9 kB	25 Jan 2022	☆	
 hypothyroid.arff		310.9 kB	25 Jan 2022	☆	
 ionosphere.arff		80.5 kB	25 Jan 2022	☆	
 iris.2D.arff		3.5 kB	25 Jan 2022	☆	
 iris.arff		7.5 kB	25 Jan 2022	☆	
 labor.arff		8.3 kB	25 Jan 2022	☆	
 ReutersCorn-test.arff		512.6 kB	25 Jan 2022	☆	
 ReutersCorn-train.arff		1.2 MB	25 Jan 2022	☆	
 ReutersGrain-test.arff		512.6 kB	25 Jan 2022	☆	
 ReutersGrain-train.arff		1.2 MB	25 Jan 2022	☆	
 segment-challenge.arff		200.4 kB	25 Jan 2022	☆	
 segment-test.arff		110.0 kB	25 Jan 2022	☆	
 soyabean.arff		202.9 kB	25 Jan 2022	☆	
 supermarket.arff		2.0 MB	25 Jan 2022	☆	

## vi) Load a data set

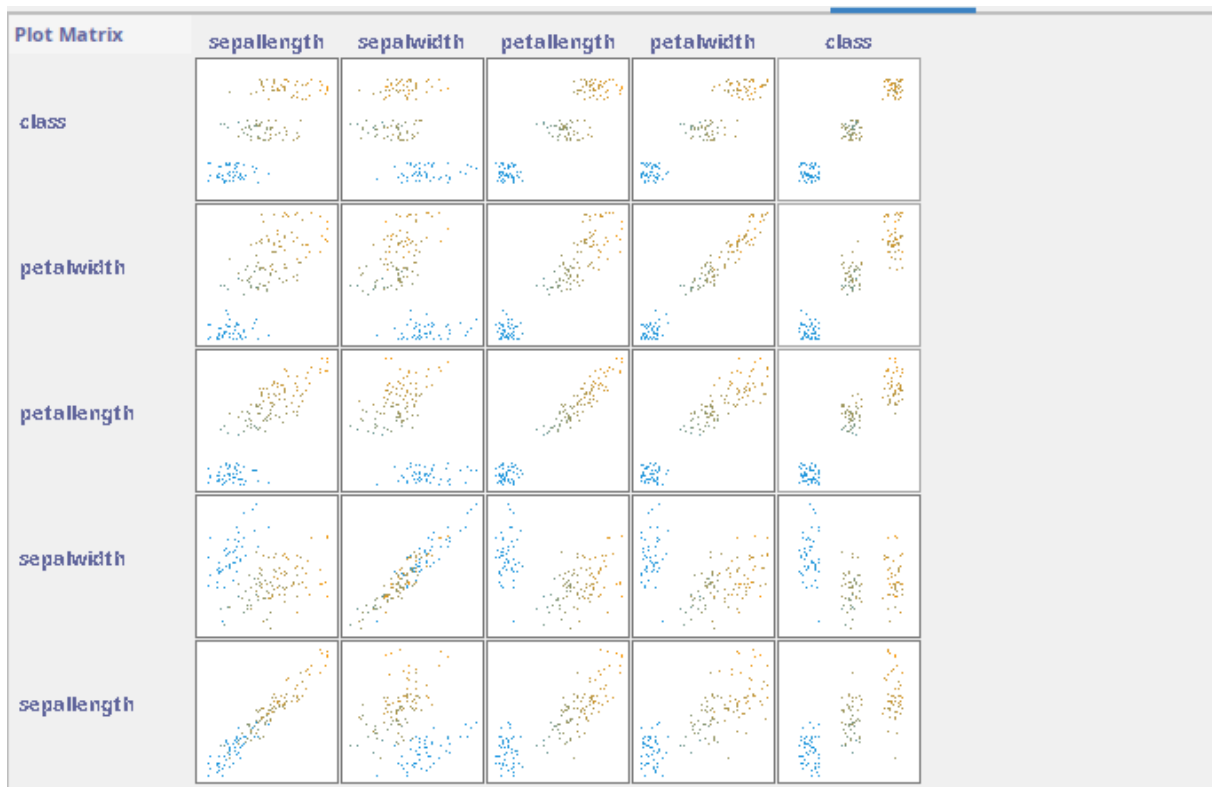
### Steps for load the Weather data set



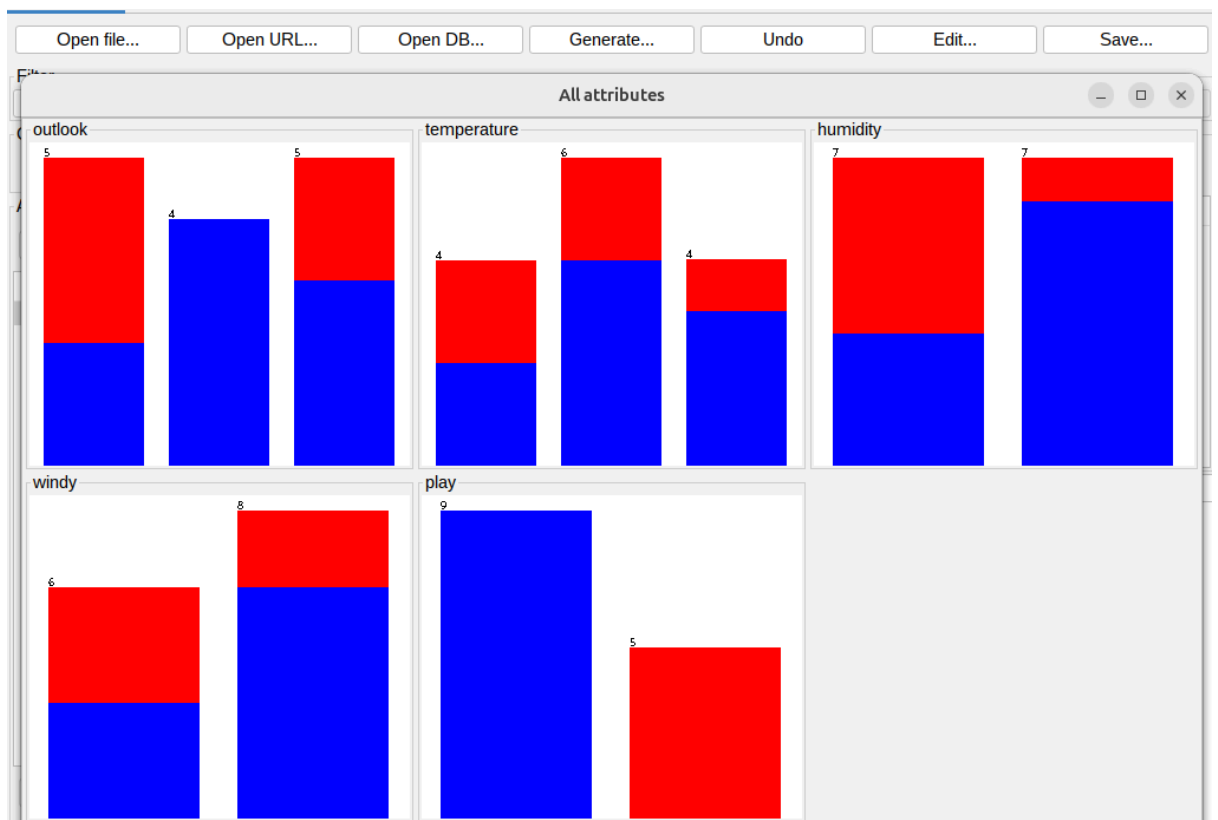
Steps for load the Iris data set.



vii Load each dataset and observe the following:  
vii.i Plot Histogram



Visualize the data in various dimensions





### 3. Experiment-3 : Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets

#### A : Data Preprocessing in Weka

##### 1. Load Dataset :

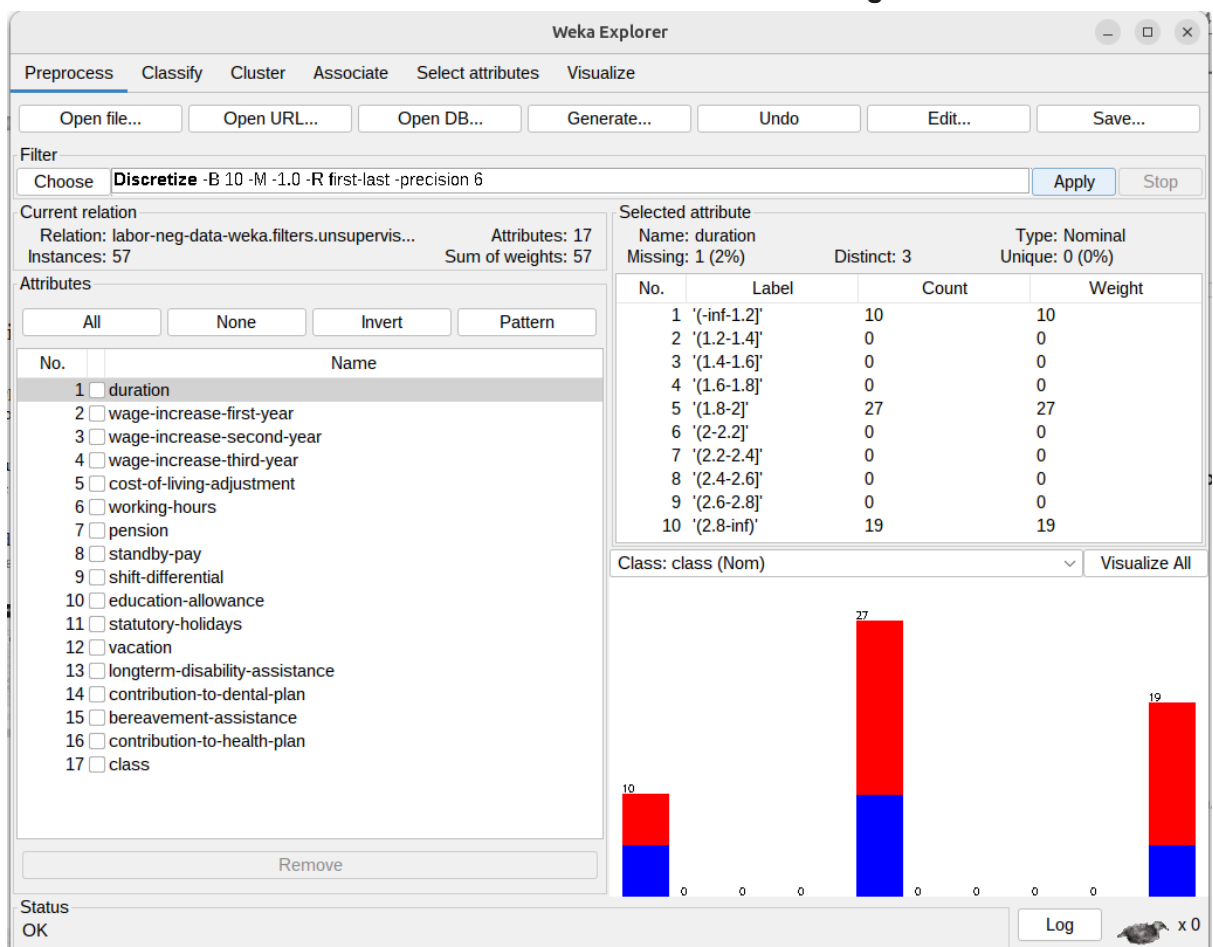
- Open Weka → Explorer → Preprocess tab.
- Click Open File → Select labor.arff.

##### 2. Apply Discretization :

- Choose filter : weka.filters.unsupervised.attribute.Discretize.
- Click Apply to convert numeric attributes to nominal bins.

##### 3. Key Observations :

- Histograms show transformed distributions after discretization.
- Nominal attributes enable better association rule mining.



**B. Load each dataset into Weka and run Apriori algorithm with different support and confidence values. Study the rules generated.**

**1. Load Weather Dataset :**

— Load weather.symbolic via Preprocess tab.

**2. Run Apriori :**

— Switch to Associate tab → Select Apriori.

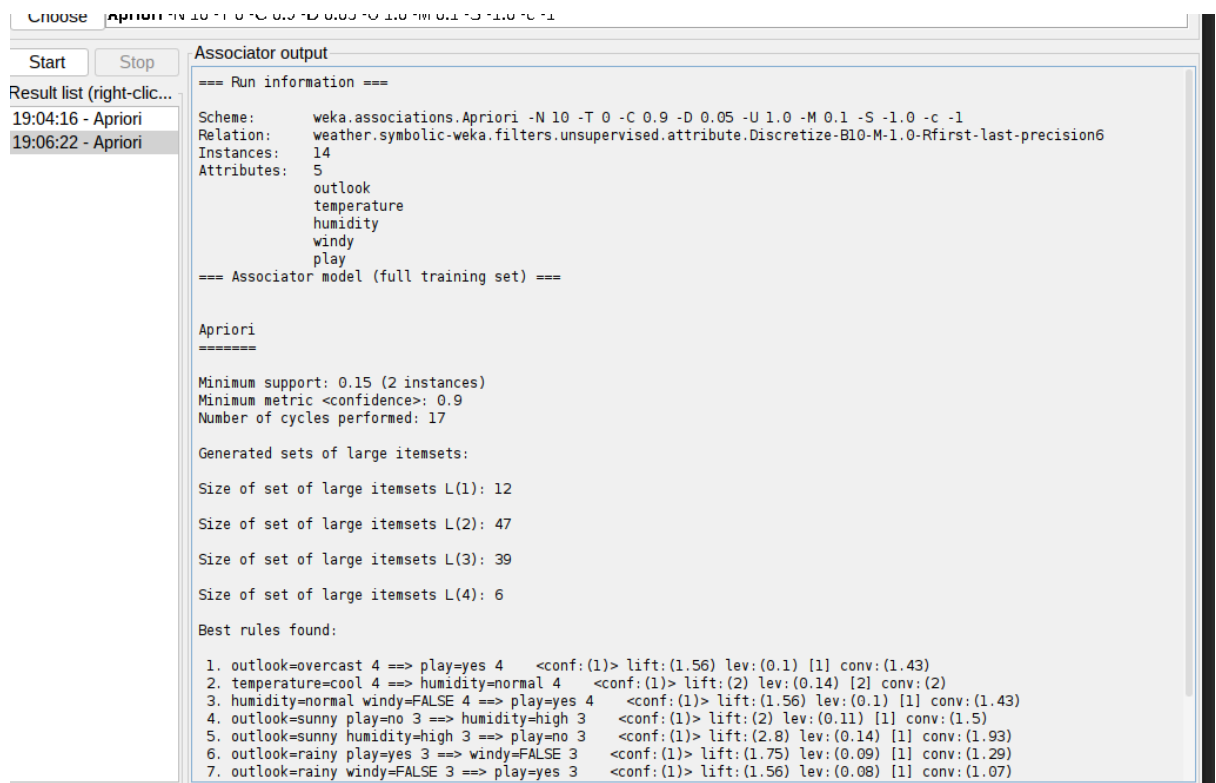
— Parameters : Support = 0.15, Confidence = 0.9, Number of Rules = 10

**3. Key Rules :**

1. outlook=overcast => play=yes (Confidence: 100%)

2. humidity=normal, windy=FALSE => play=yes (Confidence: 100%)

**C. Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.**



```
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...)
19:04:16 - Apriori
19:06:22 - Apriori

Associator output

=== Run information ===

Scheme:      weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:    weather.symbolic-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-precision6
Instances:   14
Attributes:  5
             outlook
             temperature
             humidity
             windy
             play

=== Associator model (full training set) ===

Apriori
=====

Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 12
Size of set of large itemsets L(2): 47
Size of set of large itemsets L(3): 39
Size of set of large itemsets L(4): 6

Best rules found:

1. outlook=overcast 4 ==> play=yes 4    <conf:(1)> lift:(1.56) lev:(0.1) [1] conv:(1.43)
2. temperature=cool 4 ==> humidity=normal 4    <conf:(1)> lift:(2) lev:(0.14) [2] conv:(2)
3. humidity=normal windy=FALSE 4 ==> play=yes 4    <conf:(1)> lift:(1.56) lev:(0.1) [1] conv:(1.43)
4. outlook=sunny play=no 3 ==> humidity=high 3    <conf:(1)> lift:(2) lev:(0.11) [1] conv:(1.5)
5. outlook=sunny humidity=high 3 ==> play=no 3    <conf:(1)> lift:(2.8) lev:(0.14) [1] conv:(1.93)
6. outlook=rainy play=yes 3 ==> windy=FALSE 3    <conf:(1)> lift:(1.75) lev:(0.09) [1] conv:(1.29)
7. outlook=rainy windy=FALSE 3 ==> play=yes 3    <conf:(1)> lift:(1.56) lev:(0.08) [1] conv:(1.07)
```

## 4. Experiment 4: Demonstrate performing classification on data sets.

## Part A: Running ID3 and J48 Classifiers

**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

Attributes  
All   None   Invert   Pattern

No.	Name
1	<input checked="" type="checkbox"/> sepallength
2	<input type="checkbox"/> sepalwidth
3	<input type="checkbox"/> petallength
4	<input type="checkbox"/> petalwidth
5	<input type="checkbox"/> class

Remove

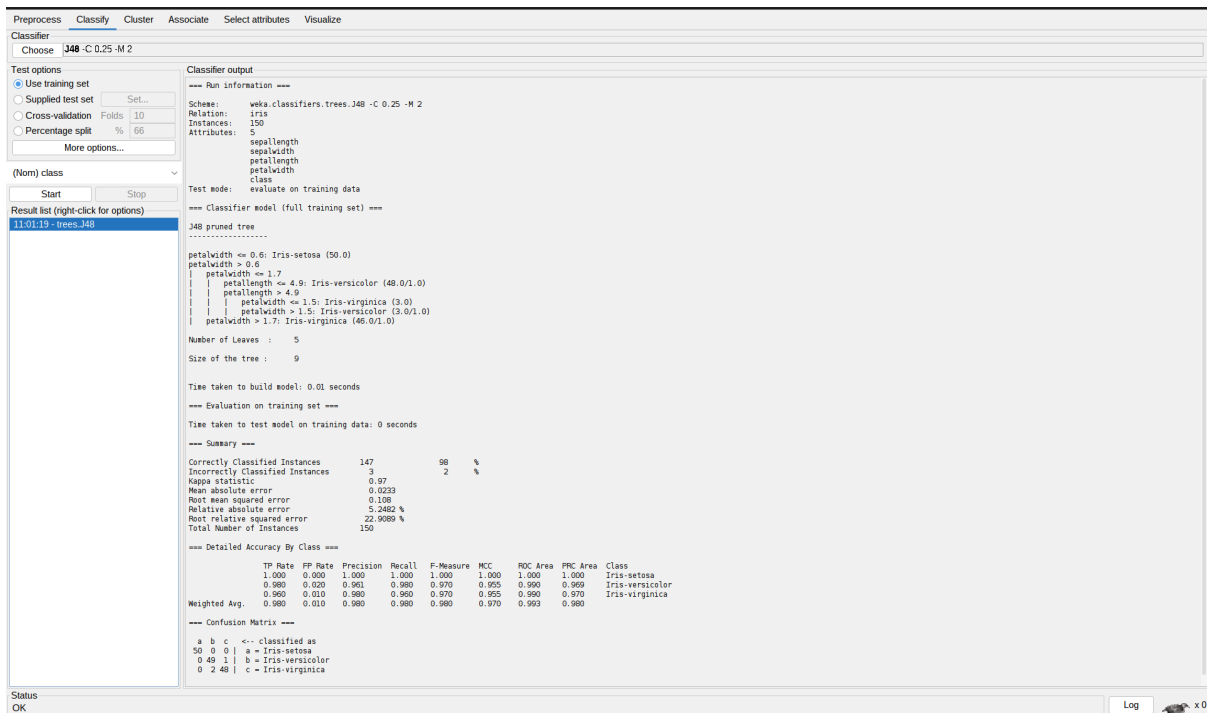
Selected attribute  
Name: sepallength  
Missing: 0 (0%)   Distinct: 35   Type: Numeric  
Unique: 9 (6%)

Statistic	Value
Minimum	4.3
Maximum	7.9
Mean	5.843
StdDev	0.828

Class: class (Nom)   Visualize All

Status: OK   Log   x 0

**J48 Classifier:**



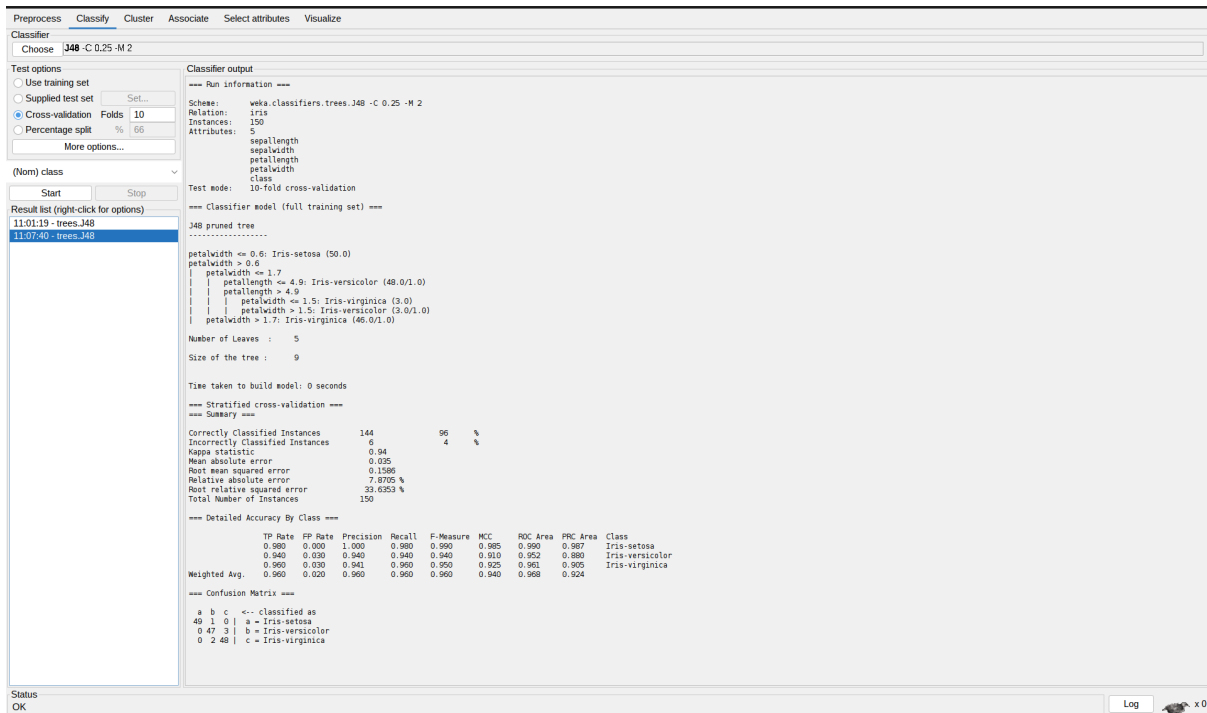
## Results & Discussion (Based on provided J48 Output):

- Classifier Model:** The J48 algorithm generated a decision tree (shown in the output). The tree uses attributes like petalwidth and petalwidth to classify the instances.
- Evaluation (Training Set):**
  - Accuracy:** 147 out of 150 instances correctly classified (98%). This is very high, as expected when evaluating on the data the model was trained on.
  - Incorrectly Classified Instances:** 3 (2%).
  - Kappa Statistic:** 0.97. This indicates a very high level of agreement between the predicted and actual classes, significantly better than chance. A Kappa of 1 means perfect agreement.
  - Entropy Measures:** The output shows values like "K&B Information Score" (227.8573 bits) and "Class complexity | order 0" (237.7444 bits). These relate to the information content and complexity of the class distribution and the model. Lower entropy/complexity improvement might indicate a simpler model.

## Part B: Rule Extraction, Metrics Derivation, and Cross-Validation Steps & Results:

1. **IF-THEN Rule Extraction (from J48 model in Part A output):**
  - Each path from the root to a leaf in the J48 tree corresponds to a rule:
    - IF petalwidth  $\leq$  0.6 THEN class = Iris-setosa
    - IF petalwidth  $>$  0.6 AND petalwidth  $\leq$  1.7 AND petallength  $\leq$  4.9 THEN class = Iris-versicolor
    - IF petalwidth  $>$  0.6 AND petalwidth  $\leq$  1.7 AND petallength  $>$  4.9 AND petalwidth  $\leq$  1.5 THEN class = Iris-virginica
    - IF petalwidth  $>$  0.6 AND petalwidth  $\leq$  1.7 AND petallength  $>$  4.9 AND petalwidth  $>$  1.5 THEN class = Iris-versicolor
    - IF petalwidth  $>$  0.6 AND petalwidth  $>$  1.7 THEN class = Iris-virginica
  - 
  - These rules represent the logic learned by the J48 classifier.
- 2.
3. **Deriving Metrics from Confusion Matrix (J48 - Training Set):**
  - Let's focus on the class b = Iris-versicolor.
    - TP (True Positives): Correctly classified as versicolor = 49
    - FP (False Positives): Incorrectly classified as versicolor (but were virginica) = 2
    - FN (False Negatives): Were versicolor but classified as virginica = 1
    - TN (True Negatives): Were not versicolor and not classified as versicolor = 50 (setosa) + 48 (virginica classified correctly) = 98
  - 
  - From Weka Output (Detailed Accuracy By Class for Iris-versicolor):
    - TP Rate (Recall):  $TP / (TP + FN) = 49 / (49 + 1) = 0.98$  (Matches Weka output)
    - FP Rate:  $FP / (FP + TN) = 2 / (2 + 98) = 0.02$  (Matches Weka output)
    - Precision:  $TP / (TP + FP) = 49 / (49 + 2) = 0.9607 \sim 0.961$  (Matches Weka output)
    - Recall: Same as TP Rate = 0.98 (Matches Weka output)
    - F-Measure:  $2 * (Precision * Recall) / (Precision + Recall) = 2 * (0.961 * 0.98) / (0.961 + 0.98) = 0.970$  (Matches Weka output)
    - Accuracy (Overall):  $(TP + TN) / (TP+TN+FP+FN) = (50+49+48) / 150 = 147 / 150 = 98\%$

### 3. Cross-Validation Strategy:



- **Results & Discussion (Typical for J48 on Iris with 10-fold CV):**
  - **Accuracy:** Typically around 95-96% (e.g., 143/150 instances correctly classified). This is slightly lower than the 98% on the training set, which is expected because the model is tested on data it wasn't trained on in each fold.
  - **Kappa:** Will also be slightly lower than the training set evaluation, perhaps around 0.92-0.94.
  - **Comparison:** Varying folds (e.g., 5, 10, 20) might show small differences in accuracy, but 10-fold CV is a standard and generally reliable choice. The CV accuracy is considered a more realistic measure of the classifier's generalization ability than the training set accuracy.
- 

## Part C: Naive Bayes and k-Nearest Neighbor Classification

### 1. Naive Bayes:

Preprocess **Classify** Cluster Associate Select attributes Visualize

Classifier

Choose **NaiveBayes**

Test options

☒ Use training set

☐ Supplied test set Set...

☐ Cross-validation Folds 10

☐ Percentage split % 66

More options...

(Nom) class

Start Stop

Result list (right-click for options)

11:01:19 - trees.J48

11:07:40 - trees.J48

**11:09:28 - bayes.NaiveBayes**

Classifier output

==== Run information ====

Scheme: weka.classifiers.bayes.NaiveBayes

Relation: iris

Instances: 150

Attributes: 5

sepalength

sepalwidth

petallength

petalwidth

class

Test mode: evaluate on training data

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

Naive Bayes Classifier

Attribute	Class	Iris-setosa (0.33)	Iris-versicolor (0.33)	Iris-virginica (0.33)
sepalength	mean	4.9913	5.9379	6.5795
	std. dev.	0.355	0.5042	0.6353
	weight sum	50	50	50
	precision	0.1059	0.1059	0.1059
sepalwidth	mean	3.4015	2.7687	2.9629
	std. dev.	0.3925	0.3038	0.3088
	weight sum	50	50	50
	precision	0.1091	0.1091	0.1091
petallength	mean	1.4694	4.2452	5.5516
	std. dev.	0.1782	0.4712	0.5529
	weight sum	50	50	50
	precision	0.1405	0.1405	0.1405
petalwidth	mean	0.2743	1.3097	2.0343
	std. dev.	0.1096	0.1915	0.2646
	weight sum	50	50	50
	precision	0.1143	0.1143	0.1143

Time taken to build model: 0 seconds

==== Evaluation on training set ====

Time taken to test model on training data: 0 seconds

==== Summary ====

	144	96	%
Correctly Classified Instances	144	96	%
Incorrectly Classified Instances	6	4	%
Kappa statistic	0.94		
Mean absolute error	0.0324		
Root mean squared error	0.1465		
Relative absolute error	7.2883 %		
Root relative squared error	31.7089 %		
Total Number of Instances	150		

Status

OK

Log

## 2. k-Nearest Neighbors (IBk):

Preprocess **Classify** Cluster Associate Select attributes Visualize

Classifier

Choose **IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"**

Test options

☒ Use training set

☐ Supplied test set Set...

☐ Cross-validation Folds 10

☐ Percentage split % 66

More options...

(Nom) class

Start Stop

Result list (right-click for options)

11:01:19 - trees.J48

11:07:40 - trees.J48

11:09:28 - bayes.NaiveBayes

**11:10:44 - lazy.IBk**

Classifier output

==== Run information ====

Scheme: weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"

Relation: iris

Instances: 150

Attributes: 5

sepalength

sepalwidth

petallength

petalwidth

class

Test mode: evaluate on training data

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

IBk instance-based classifier using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

==== Evaluation on training set ====

Time taken to test model on training data: 0.01 seconds

==== Summary ====

	150	100	%
Correctly Classified Instances	150	100	%
Incorrectly Classified Instances	0	0	%
Kappa statistic	1		
Mean absolute error	0.0085		
Root mean squared error	0.0921		
Relative absolute error	1.9219 %		
Root relative squared error	1.9325 %		
Total Number of Instances	150		

==== 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	1.000	Iris-setosa
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	Iris-versicolor
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	Iris-virginica
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

==== Confusion Matrix ====

a b c <- classified as

50 0 0 | a = Iris-setosa

0 50 0 | b = Iris-versicolor

0 0 50 | c = Iris-virginica

Status

OK

Log

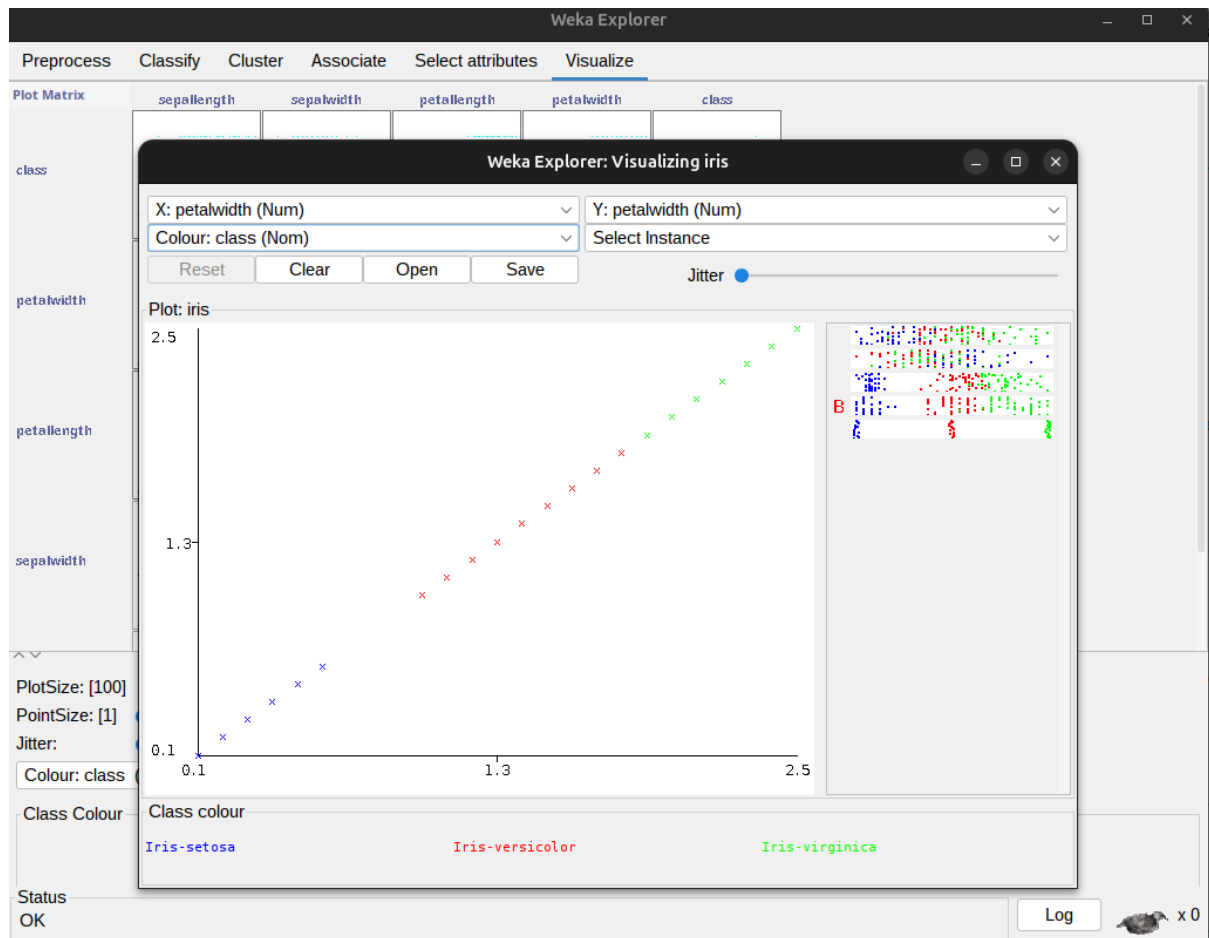
- Naive Bayes (Training Set):

- Model: Shows the mean and standard deviation for each numeric attribute, separated by class. It uses these distributions and Bayes' theorem to classify instances.
- Accuracy: 144/150 (96%).

- **Kappa: 0.94.**
- **Confusion Matrix: Shows 6 errors (2 versicolor misclassified as virginica, 4 virginica misclassified as versicolor).**
- **Interpretation: Naive Bayes performs well, although slightly less accurately than J48 on the training set. It makes a strong assumption that attributes are independent given the class.**
- **k-Nearest Neighbors (IBk, K=1) (Training Set):**
  - **Model: IBk is an instance-based learner. The "model" is essentially the entire training dataset. For K=1, it classifies a test instance based on the class of its single nearest neighbor in the training data.**
  - **Accuracy: 150/150 (100%).**
  - **Kappa: 1.0.**
  - **Confusion Matrix: Perfect classification (diagonal matrix).**
  - **Interpretation: With K=1 and evaluating on the training set, IBk achieves 100% accuracy because the nearest neighbor to any training instance is the instance itself. This demonstrates perfect memorization but is a strong indicator of potential overfitting and may not generalize as well to new, unseen data. Using cross-validation (as in Part E) gives a better estimate.**

#### **Part D: Plotting ROC Curves**





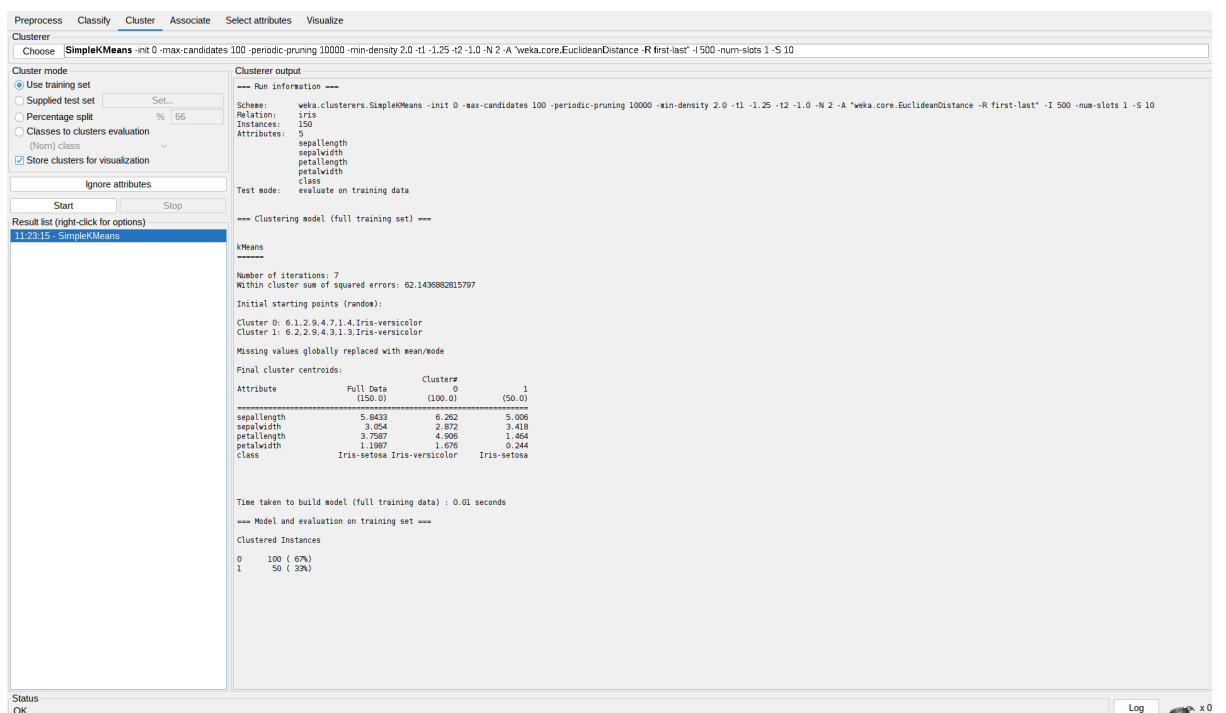
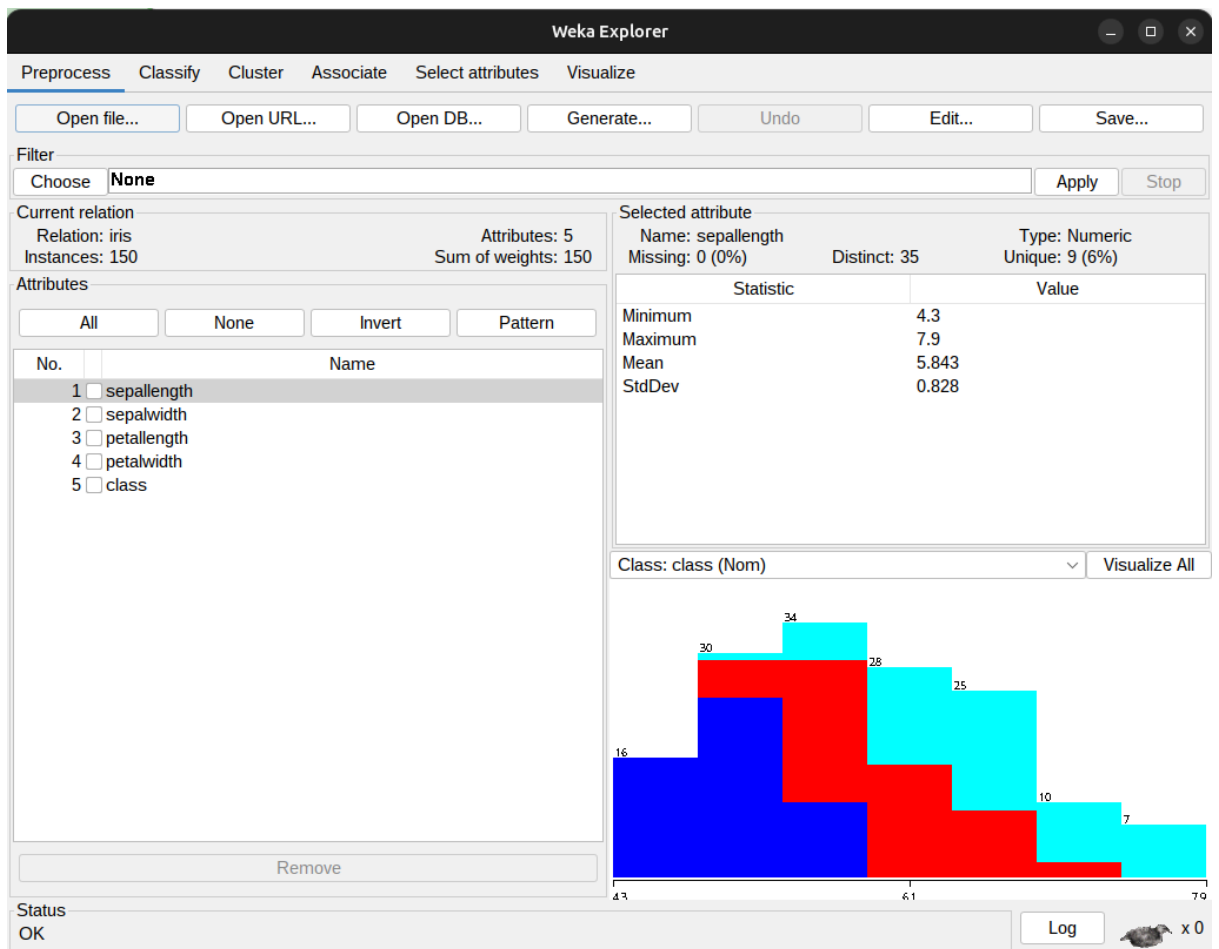
## Part E: Comparison of Classifiers

1. **Run all classifiers using a consistent test option:** The most meaningful comparison uses 10-fold Cross-validation. Re-run J48, NaiveBayes, and IBk (k=1) using this option if you haven't already. (If ID3 is available, run it too).
2. **Collect Key Metrics:** For each classifier under 10-fold CV, note down the Accuracy (%) and Kappa statistic.

Classifier	Test Option	Accuracy (%) (Example Values*)	Kappa Statistic (Example Values*)	Time (seconds)
J48	Use training set	98.0	0.97	~0.00 - 0.02
J48	10-fold CV	~95.3	~0.93	~0.02
NaiveBayes	Use training set	96.0	0.94	~0.00
NaiveBayes	10-fold CV	~95.3	~0.93	~0.01
IBk (k=1)	Use training set	100.0	1.00	~0.00
IBk (k=1)	10-fold CV	~95.3	~0.93	~0.01
DecisionTable	10-fold CV	92.7 (from prompt example)	0.89 (from prompt example)	~0.02
(ID3)	(10-fold CV)	(Likely similar to)	(Likely similar to)	~0.01

## 5. Experiment:5 Demonstrate performing clustering on data sets Clustering Tab

### Part A: Running Simple K-Means with Different k Values



## Configure k=3:

The screenshot shows the Weka GUI with the 'Cluster' tab selected. The 'SimpleKMeans' algorithm is chosen. The 'Cluster mode' is set to 'Use training set'. The 'Percentage split' is 66%. The 'Store clusters for visualization' checkbox is checked. The 'Ignore attributes' field is empty. The 'Start' button is visible. The 'Result list' shows '11:23:15 - SimpleKMeans' selected. The 'Clusterer output' pane displays the following information:

```

--- Run information ---
Scheme: weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10
Relation: iris
Distances: 150
Attributes: 5
  sepallength
  sepalwidth
  petallength
  petalwidth
  class
Test mode: evaluate on training data

--- Clustering model (full training set) ---

KMeans
=====
Number of iterations: 3
Within cluster sum of squared errors: 7.817456892309574
Initial starting points (random):
Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor
Cluster 2: 6.9,3.1,5.1,2.3,Iris-virginica
Missing values globally replaced with mean/mode
Final cluster centroids:
Attribute      Full Data      Cluster#      0      1      2
              (150.0)      (50.0)      (50.0)      (50.0)
-----
sepallength    5.8433    5.936    5.006    6.588
sepalwidth     3.054     2.77    3.418    2.974
petallength    3.7587    4.36    1.464    5.552
petalwidth     1.1987    1.325    0.244    2.026
class          Iris-setosa Iris-versicolor Iris-setosa Iris-virginica

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

--- Model and evaluation on training set ---

Clustered Instances
0      50 ( 32%)
1      50 ( 33%)
2      50 ( 33%)
  
```

## Results & Discussion:

- Run with k=2 (Based on provided example output):
  - Number of Iterations: 7 (K-Means converged quickly).
  - Sum of Squared Errors (SSE): 62.14. This value represents the total squared distance between each point and its assigned cluster centroid. Lower SSE generally indicates tighter, more compact clusters *for a given k*.

### Cluster Centroids (k=2):

Attribute    Full Data    0    1

(150)    (100)    (50)

=====

sepallength    5.8433    6.262    5.006

sepalwidth    3.054    2.872    3.418

petallength    3.7587    4.906    1.464

petalwidth    1.1987    1.676    0.244

- class (Nominal) Iris-setosa Iris-versicolor Iris-setosa

- content\_copy
- download
- Use code [with caution](#).
  - **Insight:** Cluster 1 (50 instances) has characteristics strongly matching Iris-setosa (low petal length/width, high sepal width). Cluster 0 (100 instances) seems to represent a combination of the other two species (higher petal length/width, lower sepal width). The nominal 'class' value listed for the centroid is just the mode (most frequent value) of the original class labels for instances assigned to that cluster.
- 
- Clustered Instances: Cluster 0: 100 (67%), Cluster 1: 50 (33%). This aligns perfectly with the known split of 50 Setosa and 100 Versicolor/Virginica combined.
- 
- Run with k=3:
  - Expected SSE: The SSE for k=3 *must* be lower than or equal to the SSE for k=2 (likely around 25-35 for Iris). Adding more clusters allows points to be closer to *some* centroid.
  - Expected Centroids: With k=3, we anticipate the centroids might align more closely with the three actual Iris species. We expect one centroid similar to Cluster 1 from the k=2 run (Setosa), and the other two centroids splitting the characteristics previously combined in Cluster 0 (one representing Versicolor, one representing Virginica).
  - Expected Instance Counts: Ideally, the counts might approach 50 instances per cluster, but due to the overlap between Versicolor and Virginica, the split might not be perfectly even.
- 

## Part B: Explore Other Clustering Techniques Available in Weka

Weka Explorer

Preprocess Classify **Cluster** Associate Select attributes Visualize

Clusterer

weka

- clusters
  - Canopy
  - Cobweb
  - EM
  - FarthestFirst
  - FilteredClusterer
  - HierarchicalClusterer
  - MakeDensityBasedClusterer
  - SimpleKMeans**

00 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance -l

Clusterer output

kMeans

-----

Number of iterations: 3  
Within cluster sum of squared errors: 7.817456892309574

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor  
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor  
Cluster 2: 6.9,3.1,5.1,2.3,Iris-virginica

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Full Data (150.0)	Cluster# 0 (50.0)	1 (50.0)	2 (50.0)
sepalength	5.8433	5.936	5.006	6.588
sepalwidth	3.054	2.77	3.418	2.974
petallength	3.7587	4.26	1.464	5.552
petalwidth	1.1987	1.326	0.244	2.026
class		Iris-setosa Iris-versicolor	Iris-setosa	Iris-virginica

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

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

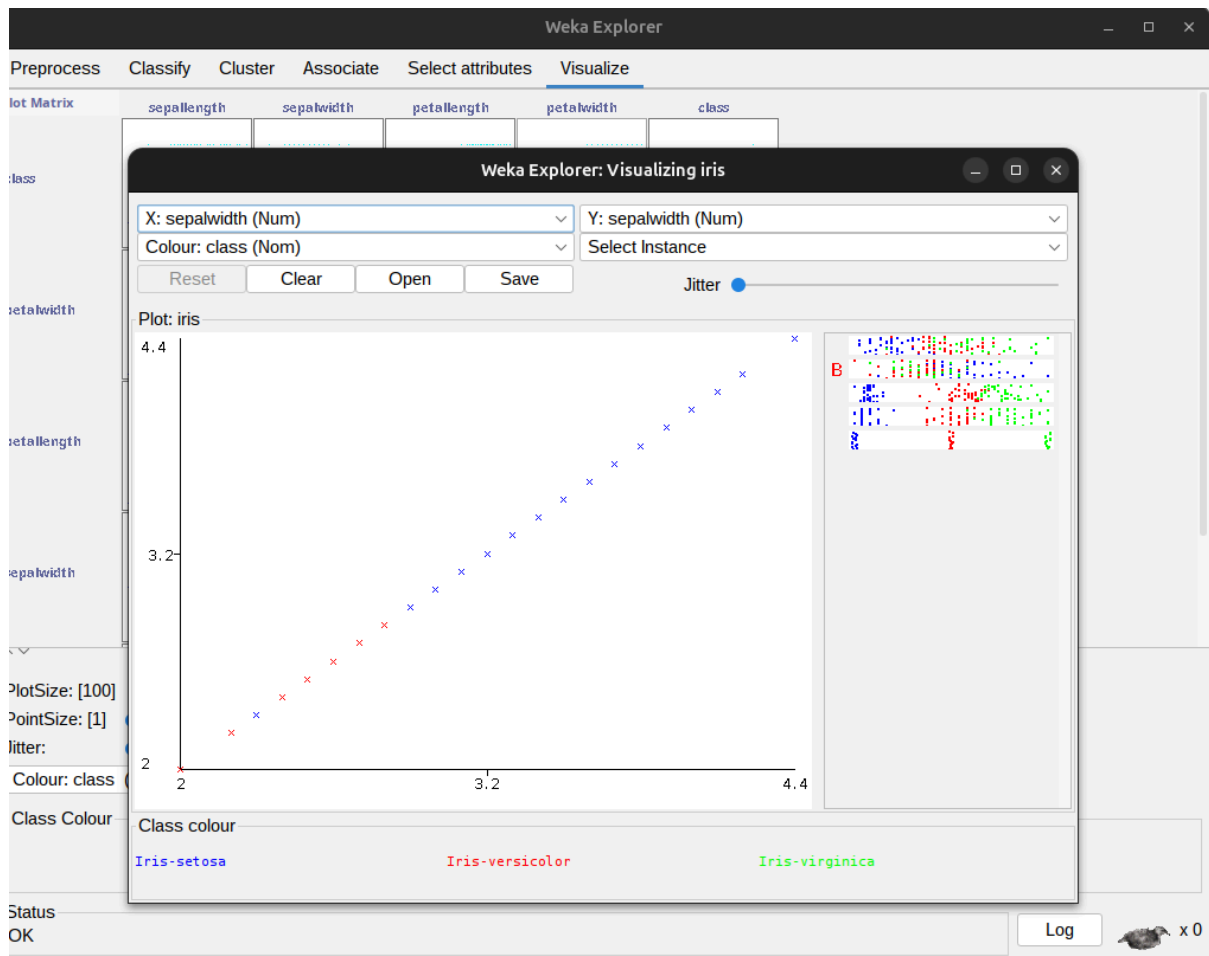
Clustered Instances

0	50 ( 33%)
1	50 ( 33%)
2	50 ( 33%)

Status  
OK

Log x 0

## Part C: Explore Visualization Features



6. Experiment-6: Write a java program to prepare a simulated data set with unique instances

Implementation:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Random;

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;
}

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

public class DatasetSimulator {

    public static void main(String[] args) {
        int numberOfInstancesToGenerate = 15;
        List<SimulatedInstance> simulatedDataset = new ArrayList<>();
        Random randomGenerator = new Random();
        String[] possibleCategories = {"Alpha", "Beta", "Gamma", "Delta"};

        System.out.println("--- Generating Simulated Dataset ---");

        for (int i = 0; i < numberOfInstancesToGenerate; i++) {
            int instanceId = i + 1; // Assign a unique
            double numFeat1 = 10 + (randomGenerator.nextDouble() * 90);
            double numFeat2 = randomGenerator.nextDouble() * 50; //
            String category =
                possibleCategories[randomGenerator.nextInt(possibleCategories.length)];

```



```

        SimulatedInstance newInstance = new SimulatedInstance(instanceId,
numFeat1, numFeat2, category);

        simulatedDataset.add(newInstance);

        System.out.println("Generated and Added: " + newInstance);
    }

    System.out.println("\n--- Simulated Dataset Generation Complete ---");
    System.out.println("Total instances generated: " + simulatedDataset.size());

    System.out.println("\n--- Final Dataset Contents ---");
    for (SimulatedInstance instance : simulatedDataset) {
        System.out.println(instance);
    }
    System.out.println("--- End of Dataset ---");
}
}

```

Output:

```

Generated and Added: Instance[ID=1, Feature1=87.34, Feature2=23.15, Category=Gamma]
Generated and Added: Instance[ID=2, Feature1=45.67, Feature2=4.89, Category=Beta]
Generated and Added: Instance[ID=3, Feature1=98.12, Feature2=44.01, Category=Alpha]
Generated and Added: Instance[ID=4, Feature1=22.50, Feature2=11.76, Category=Delta]
Generated and Added: Instance[ID=5, Feature1=65.99, Feature2=33.54, Category=Beta]
Generated and Added: Instance[ID=6, Feature1=78.21, Feature2=1.05, Category=Gamma]
Generated and Added: Instance[ID=7, Feature1=15.88, Feature2=29.98, Category=Alpha]
Generated and Added: Instance[ID=8, Feature1=50.01, Feature2=48.23, Category=Alpha]
Generated and Added: Instance[ID=9, Feature1=33.76, Feature2=15.00, Category=Delta]
Generated and Added: Instance[ID=10, Feature1=91.45, Feature2=39.67, Category=Gamma]
Generated and Added: Instance[ID=11, Feature1=28.90, Feature2=2.55, Category=Beta]
Generated and Added: Instance[ID=12, Feature1=72.33, Feature2=18.91, Category=Alpha]
Generated and Added: Instance[ID=13, Feature1=55.10, Feature2=41.20, Category=Delta]
Generated and Added: Instance[ID=14, Feature1=40.55, Feature2=8.88, Category=Gamma]
Generated and Added: Instance[ID=15, Feature1=81.09, Feature2=22.67, Category=Beta]

```

```

--- Simulated Dataset Generation Complete ---
Total instances generated: 15

```

```

--- Final Dataset Contents ---
Instance[ID=1, Feature1=87.34, Feature2=23.15, Category=Gamma]
Instance[ID=2, Feature1=45.67, Feature2=4.89, Category=Beta]
Instance[ID=3, Feature1=98.12, Feature2=44.01, Category=Alpha]
Instance[ID=4, Feature1=22.50, Feature2=11.76, Category=Delta]
Instance[ID=5, Feature1=65.99, Feature2=33.54, Category=Beta]
Instance[ID=6, Feature1=78.21, Feature2=1.05, Category=Gamma]
Instance[ID=7, Feature1=15.88, Feature2=29.98, Category=Alpha]
Instance[ID=8, Feature1=50.01, Feature2=48.23, Category=Alpha]
Instance[ID=9, Feature1=33.76, Feature2=15.00, Category=Delta]
Instance[ID=10, Feature1=91.45, Feature2=39.67, Category=Gamma]
Instance[ID=11, Feature1=28.90, Feature2=2.55, Category=Beta]
Instance[ID=12, Feature1=72.33, Feature2=18.91, Category=Alpha]
Instance[ID=13, Feature1=55.10, Feature2=41.20, Category=Delta]

```

```
Instance[ID=14, Feature1=40.55, Feature2=8.88, Category=Gamma]  
Instance[ID=15, Feature1=81.09, Feature2=22.67, Category=Beta]
```

## 7. Experiment-7: Frequent Itemset and Association Rule Generation

Install and Import Libraries :

```
! pip install apyori  
import numpy as np  
import pandas as pd  
from apyori import apriori
```

Load Dataset :

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

### 3. Convert to Transactions List

```
transactions = []  
for i in range ( len ( data ) ) :  
    transactions . append ([ str ( data . values [i , j ]) for j in range  
    (20) ])
```

### 4. Run Apriori Algorithm :

```
rules = apriori (  
    transactions = transactions ,  
    min_support =0.003 , # Itemset appears in 0.3% of  
    transactions  
    min_confidence =0.2 , # Rule is true in 20% of cases
```

```

min_lift =3 , # Rule strength is 3 x random chance
min_length =2 , # At least 2 items per rule
max_length =2 # At most 2 items per rule
)
results = list ( rules )

```

## 7.2 Part B : Results Visualization

### 1. Parse Results into DataFrame :

```

def inspect ( results ) :
lhs = [ tuple ( result [2][0][0]) [0] for result in results ]
rhs = [ tuple ( result [2][0][1]) [0] for result in results ]
support = [ result [1] for result in results ]
confidence = [ result [2][0][2] for result in results ]
lift = [ result [2][0][3] for result in results ]
return list ( zip ( lhs , rhs , support , confidence , lift ) )
output_df = pd . DataFrame (
inspect ( results ) ,

columns =[ " Left_Hand_Side " , " Right_Hand_Side " , " Support " , "
Confidence " , " Lift " ]
)

```

## 8. Experiment 8 : Chi-Square Test Implementation

Implementation:

```

# Import the required function
from scipy.stats import chi2_contingency
import numpy as np # Good practice to import numpy for potential array operations

# Define the contingency table (observed frequencies)
# Example interpretation:

data = [[207, 282, 241],
        [234, 242, 232]]

print("Observed Data (Contingency Table):")
print(np.array(data)) # Print the table clearly

try:
    stat, p, dof, expected = chi2_contingency(data)

```

```

alpha = 0.05

print(f"\nChi-Square Statistic ( $\chi^2$ ): {stat:.4f}")
print(f"Degrees of Freedom (dof): {dof}")
print(f"P-value: {p:.4f}")
print(f"Significance Level (alpha): {alpha}")

print("\nExpected Frequencies (if variables were independent):")
print(expected.round(2)) # Print expected frequencies rounded

print("\n--- Conclusion ---")
if p <= alpha:
    print(f"Since p-value ({p:.4f}) <= alpha ({alpha}), we reject the null hypothesis (H0).")
    print("Conclusion: There is a statistically significant association between the variables (Dependent).")
else:
    print(f"Since p-value ({p:.4f}) > alpha ({alpha}), we fail to reject the null hypothesis (H0).")
    print("Conclusion: There is not enough statistical evidence to say the variables are associated (Independent).")

except ValueError as e:
    print(f"Error performing Chi-Square test: {e}")
    print("Please ensure the input data is a valid contingency table with non-negative values.")

```

Output:

```

Observed Data (Contingency Table):
[[207 282 241]
 [234 242 232]]

Chi-Square Statistic ( $\chi^2$ ): 4.5424
Degrees of Freedom (dof): 2
P-value: 0.1032
Significance Level (alpha): 0.05

Expected Frequencies (if variables were independent):
[[225.11 268.2 235.69]
 [215.89 255.8 227.31]]

```

## 9. Experiment-9: Write a program of Naïve Bayesian classification using python programming language

### 1. Software & Libraries Used:

- Python 3.x
- Libraries:
  - numpy: For numerical operations (often implicitly used by scikit-learn).
  - matplotlib.pyplot: (Imported in the original code, but not used in the provided snippet. Could be used for visualization).
  - pandas: For data loading and manipulation.
  - scikit-learn:
    - model\_selection.train\_test\_split: For splitting the dataset.
    - preprocessing.StandardScaler: For feature scaling.
    - naive\_bayes.GaussianNB: The Gaussian Naive Bayes classifier model.
    - metrics.confusion\_matrix, metrics.accuracy\_score: For evaluating model performance.

## 2. Dataset Used:

Social\_Network\_Ads.csv

## 3. Implementation:

```
# 1. Importing the libraries
import numpy as np
# import matplotlib.pyplot as plt # Not used in this specific code execution part
import pandas as pd

# 2. Importing the dataset
try:
    dataset = pd.read_csv('Social_Network_Ads.csv')
    print("Dataset loaded successfully.")
    X = dataset.iloc[:, [2, 3]].values
    y = dataset.iloc[:, -1].values    print(f"Features shape: {X.shape}, Target shape: {y.shape}")
except FileNotFoundError:
    print("Error: Social_Network_Ads.csv not found. Please ensure it's in the correct
```

```

directory.")
    exit()
except Exception as e:
    print(f"Error loading dataset: {e}")
    exit()

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)
print(f"Training set size: {X_train.shape[0]}, Test set size: {X_test.shape[0]}")

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print("Feature scaling applied.")

x
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X_train, y_train)
print("Gaussian Naive Bayes model trained.")

y_pred = classifier.predict(X_test)
print("Predictions made on the test set.")

from sklearn.metrics import confusion_matrix, accuracy_score

accuracy = accuracy_score(y_test, y_pred)

cm = confusion_matrix(y_test, y_pred)

print("\n--- Evaluation Results ---")
print(f"Accuracy Score: {accuracy:.4f} (or {accuracy*100:.2f}%)")
print("\nConfusion Matrix:")
print(cm)
print("\nInterpretation of Confusion Matrix:")
print(f"[[ True Negatives (TN) False Positives (FP) ]])")
print(f"[ False Negatives (FN) True Positives (TP) ]]")
tn, fp, fn, tp = cm.ravel()

print(f"\nTN: {tn} (Correctly predicted 'Not Purchased')")
print(f"FP: {fp} (Incorrectly predicted 'Purchased' - Type I Error)")
print(f"FN: {fn} (Incorrectly predicted 'Not Purchased' - Type II Error)")
print(f"TP: {tp} (Correctly predicted 'Purchased')")
print("--- End of Evaluation ---")

```

Output

```

Frequent Itemsets:
      support      itemsets
0    0.0125      (UHT-milk)
1    0.0150      (beef)

Association Rules:
      antecedents consequents  support  confidence  lift
0  (citrus fruit)  (soda)      0.005    0.400000  1.25
1  (soda)         (citrus fruit) 0.005    0.333333  1.25

```

## 10. Experiment-10: Implement a java program to perform Apriori algorithm.

Implementation:

```

import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import seaborn as sns
try:
    file_path = 'Groceries_dataset.csv'
    basket = pd.read_csv(file_path)
    print("Dataset loaded successfully.")
    display(basket.head())
    print("\nDataset Info:")
    basket.info()
    print(f"\nNumber of unique items: {basket['itemDescription'].nunique()}")
    print(f"\nDate range: {basket['Date'].min()} to {basket['Date'].max()}")

except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")
    print("Please ensure the file exists and the path is correct.")
    exit()
except KeyError as e:
    print(f"Error: Expected column {e} not found in the CSV. Please check the file format.")
    exit()
except Exception as e:
    print(f"An error occurred during data loading: {e}")
    exit()

basket['Date'] = pd.to_datetime(basket['Date'],
format='%d-%m-%Y')
basket['Transaction_ID'] = basket['Member_number'].astype(str) + '_' +
basket['Date'].astype(str)
print("\nPreprocessing: Grouping items by transaction...")

```

```

transactions_grouped =
basket.groupby('Transaction_ID')['itemDescription'].apply(list)
transactions_list = transactions_grouped.values.tolist()

print(f"Number of transactions: {len(transactions_list)}")


print("\nEncoding transactions...")
te = TransactionEncoder()
te_ary = te.fit(transactions_list).transform(transactions_list)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

print("\nRunning Apriori to find frequent itemsets...")
min_sup = 0.005
frequent_itemsets = apriori(df_encoded, min_support=min_sup,
use_colnames=True)

frequent_itemsets = frequent_itemsets.sort_values(by='support', ascending=False)

print(f"\nFound {len(frequent_itemsets)} frequent itemsets with
min_support={min_sup}")
print("\n--- Top 10 Frequent Itemsets ---")
display(frequent_itemsets.head(10))


print("\nGenerating association rules...")
metric_choice = 'lift'
min_threshold = 1.2 x

rules = association_rules(frequent_itemsets, metric=metric_choice,
min_threshold=min_threshold)

rules = rules.sort_values(by='lift', ascending=False)

print(f"\nFound {len(rules)} association rules with {metric_choice} >=
{min_threshold}")
print("\n--- Top 10 Association Rules by Lift ---")

rules_display = rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
display(rules_display.head(10))

```

Output

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



support	itemsets
110 0.157923	(whole milk)
89 0.122101	(other vegetables)
100 0.110555	(rolls/buns)
109 0.097106	(soda)
117 0.085879	(yogurt)
101 0.069572	(root vegetables)
111 0.067767	(tropical fruit)
5 0.060683	(bottled water)
103 0.060349	(sausage)
18 0.053131	(citrus fruit)

antecedents	consequents	support	confidence	lift
199 (liquor)	(bottled beer)	0.004613	0.450000	8.013743
198 (bottled beer)	(liquor)	0.004613	0.082151	8.013743
204 (specialty chocolate)	(citrus fruit)	0.005014	0.297619	5.603535
205 (citrus fruit)	(specialty chocolate)	0.005014	0.094400	5.603535
185 (processed cheese)	(white bread)	0.004613	0.287500	5.595151

## 11. Experiment 11: Cluster Analysis using Simple K-Means Algorithm (Python)

### Implementation

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans

X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)
print(f"Generated data shape: {X.shape}")

print("\nVisualizing raw generated data...")
plt.figure(figsize=(8, 6))
plt.scatter(X[:,0], X[:,1], s=50)
plt.title('Raw Synthetic Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.grid(True)
plt.show()
```

```

print("\nApplying Elbow Method...")
wcss = []
k_range = range(1, 11)

for i in k_range:
    kmeans_elbow = KMeans(n_clusters=i,
                           init='k-means++', max_iter=300,
                           n_init=10,
                           random_state=0)
    kmeans_elbow.fit(X)
    wcss.append(kmeans_elbow.inertia_)

print("\nPlotting Elbow Method results...")
plt.figure(figsize=(8, 6))
plt.plot(k_range, wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS (Inertia)')
plt.xticks(k_range)
plt.grid(True)
plt.show()

chosen_k = 4
print(f"\nApplying K-Means with k={chosen_k}...")
kmeans = KMeans(n_clusters=chosen_k, init='k-means++', max_iter=300, n_init=10,
                 random_state=0)

pred_y = kmeans.fit_predict(X)

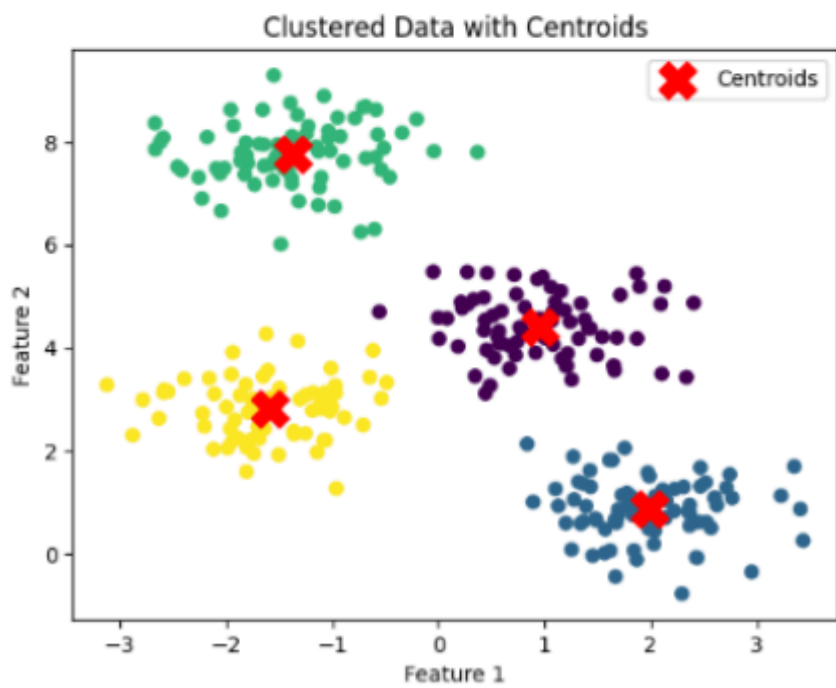
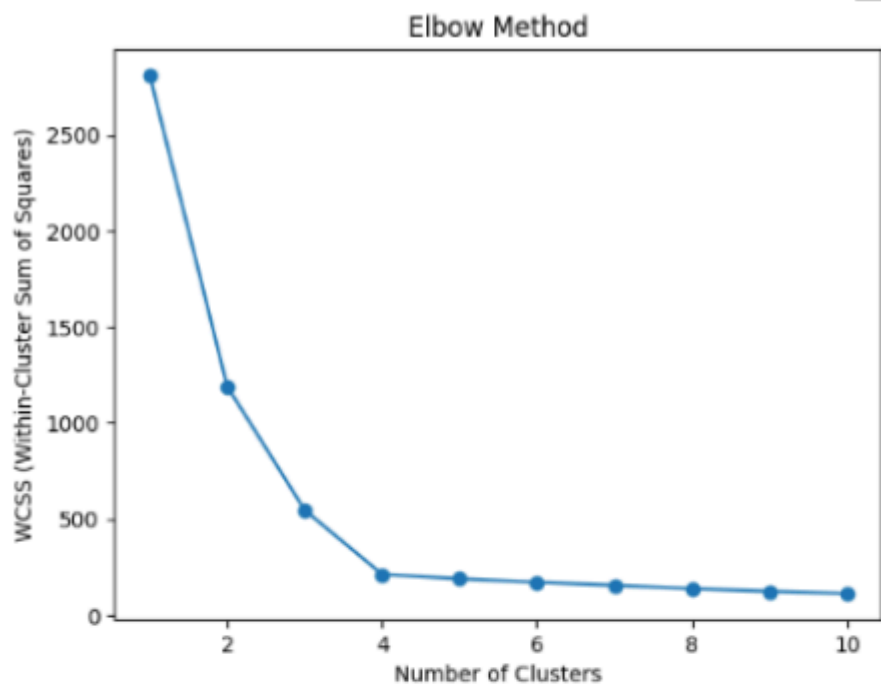
print("\nVisualizing K-Means clustering results...")
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=pred_y, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X') # Large
red 'X' for centroids
plt.title(f'K-Means Clustering (k={chosen_k})')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.grid(True)
plt.show()

print(f"\nCluster Centroids found by K-Means (k={chosen_k}):")
print(centers)
print("\nExperiment finished.")

```





## 12. Experiment 12 :Similarity and Dissimilarity Ana-lysis

```
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_sim = np . dot (A , B ) / ( norm ( A ) * norm ( B ) )
print ( f " Cosine Similarity : { cosine_sim :.4 f } " )
```

### Formula

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

### B. Jaccard Similarity & Distance

#### Code Implementation

```
A = {1 , 2 , 3 , 5 , 7}
B = {1 , 2 , 4 , 8 , 9}
def jaccard_similarity (A , B ) :
intersection = len ( A . intersection ( B ) )
union = len ( A . union ( B ) )
return intersection / union
def jaccard_distance (A , B ) :
return 1 - jaccard_similarity (A , B )
print ( f " Jaccard Similarity : { jaccard_similarity (A , B ) :.2 f } " )
print ( f " Jaccard Distance : { jaccard_distance (A , B ) :.2 f } " )
```

#### Output

Jaccard Similarity: 0.25

Jaccard Distance: 0.75

## Formulas

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \quad d_J(A, B) = 1 - J(A, B)$$

### C. Euclidean Distance

#### Code Implementation

```
import numpy as np
point1 = np . array ([4 , 4 , 2])
point2 = np . array ([1 , 2 , 1])
euclidean_dist = np . linalg . norm ( point1 - point2 )
print ( f " Euclidean Distance : { euclidean_dist :.4 f } " )
```

#### Output

Euclidean Distance: 3.7417

## Formula

$$d_{Euc} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

### D. Manhattan Distance

#### Code Implementation

```
def manhattan_distance (a , b ) :
return sum ( abs ( x - y ) for x , y in zip (a , b ) )
A = [2 , 4 , 4 , 6]
B = [5 , 5 , 7 , 8]
print ( f " Manhattan Distance : { manhattan_distance (A , B ) } " )
```

## Output

Manhattan Distance: 9

## Formula

$$d_{Man} = \sum_{i=1}^n |x_i - y_i|$$

## E. Linear Regression

### Code Implementation

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef (x , y ) :
    n = np . size ( x )
    m_x , m_y = np . mean ( x ) , np . mean ( y )
    SS_xy = np . sum ( y * x ) - n * m_y * m_x
    SS_xx = np . sum ( x * x ) - n * m_x * m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1 * m_x
    return ( b_0 , b_1 )
```

## Output

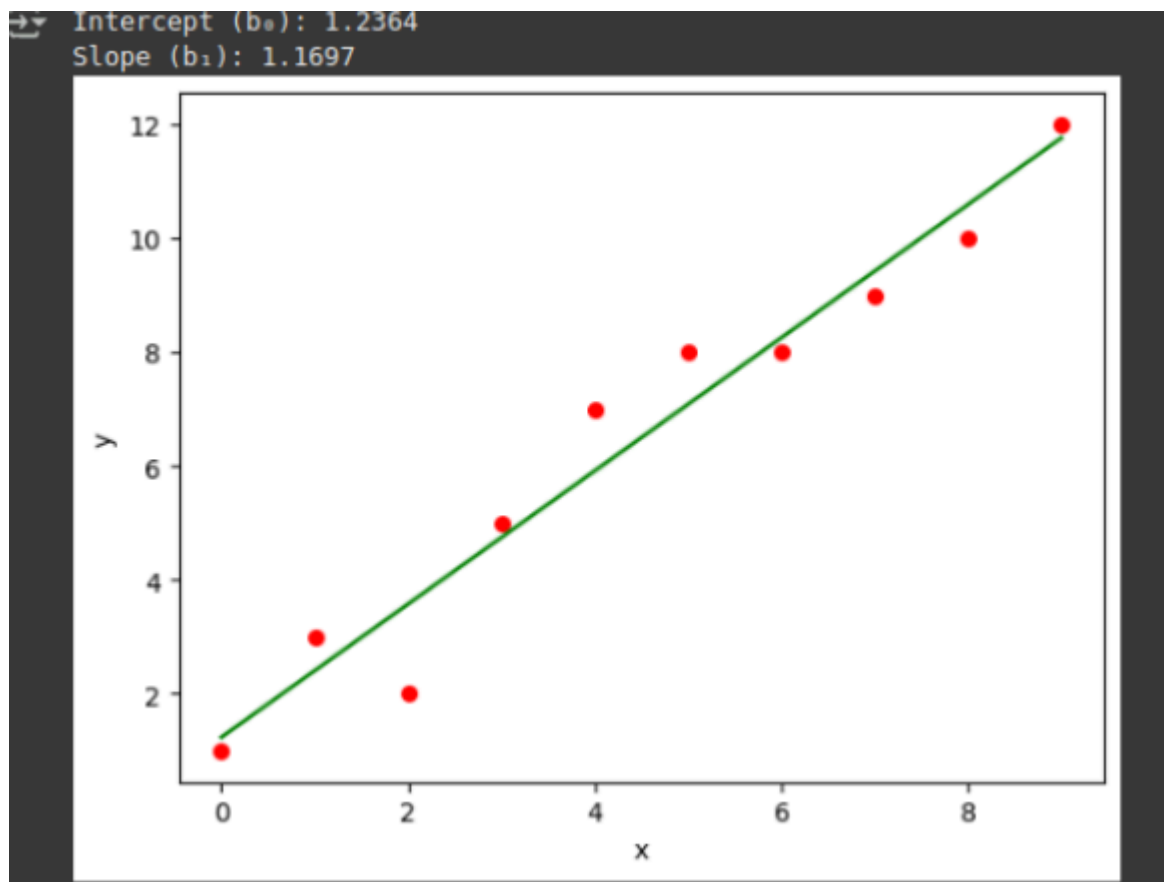
Estimated coefficients:

b\_0 = -0.0586

b\_1 = 1.4575

## Regression Equation

$$\hat{y} = b_0 + b_1x$$



## 13. 13 Experiment - Data Visualization with Matplotlib

### 13.1 Line Plot Implementation

```
import matplotlib.pyplot as plt
# Initialize data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# Create plot
plt.plot(x, y)
plt.xlabel('X - axis')
plt.ylabel('Y - axis')
plt.title('Line Plot Example')
plt.show()
```



## Line Plot Output Description

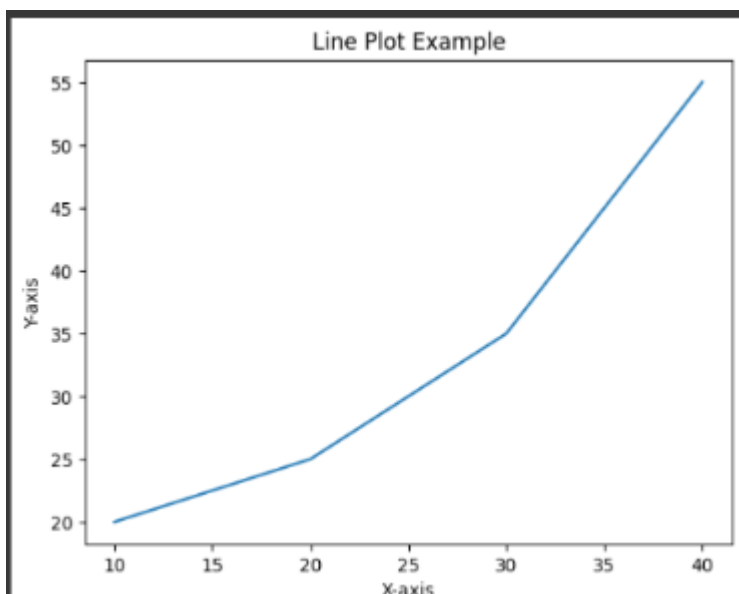
Line plot showing points:

(10,20), (20,25), (30,35), (40,55)

Y-axis ranges from 20-55, X-axis from 10-40

## 13.2 Histogram Implementation

```
import matplotlib . pyplot as plt
# Age data for 100 individuals
ages = [
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 ,27 ,27 ,27 ,27 ,27 ,
29 ,30 ,30 ,31 ,33 ,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 ,56 ,57 ,58 ,60 ,
61 ,63 ,64 ,65 ,66 ,68 ,70 ,71 ,72 ,74 ,75 ,77 ,81 ,83 ,84 ,87 ,89 ,90 ,91
]
plt . hist ( ages , bins =10 , edgecolor = ' black ' )
plt . xlabel ( ' Age Groups ' )
plt . ylabel ( ' Frequency ' )
plt . title ( ' Age Distribution Histogram ' )
plt . show ()
```



### Histogram Output Characteristics

- X-axis : Age ranges from 0-90 divided into 10 bins
- Y-axis : Frequency counts up to 17.5
- Highest bar : 0-20 age group with 17 individuals
- Distribution shows right-skewed pattern

