Experiment 1

Aim: Study of ETL process and its tools.

Theory:

Dataset:

	Name	Department	Basic	Bonus
0	Employee_1	HR	41291.0	8181.0
1	Employee_2	Sales	43241.0	8635.0
2	Employee_3	IT	NaN	8896.0
3	Employee_4	HR	47441.0	8685.0
4	Employee_5	Marketing	49767.0	7768.0
...	...	...	...	...
195	Employee_196	Finance	40048.0	7673.0
196	Employee_197	Finance	49372.0	9106.0
197	Employee_198	Finance	33814.0	6700.0
198	Employee_199	HR	51541.0	9817.0
199	Employee_200	Finance	36934.0	3959.0

Code:

```
import pandas as pd

# 1. Extract: Read CSV
employee = pd.read_csv("employee_salary.csv") # upload first in Colab
display(employee.head()) #dipslay initial 5 rows from the table

# 2. Transform
# Handle missing values using mean
employee['Basic'] = employee['Basic'].fillna(employee['Basic'].mean())
employee['Bonus'] = employee['Bonus'].fillna(employee['Bonus'].mean())

# Calculate Total_Salary
employee['Total_Salary'] = employee['Basic'] + employee['Bonus']

# Add Tax column
employee['Tax'] = employee['Total_Salary'].apply(lambda x: 0.10 * x if x >= 40000 else 0.05 * x)

# 3. Load: Save as employee_salary_transformed.csv
employee.to_csv("employee_salary_transformed.csv", index=False)

print("Transformation complete! File saved as employee_salary_transformed.csv")
employee
```

Output:

File saved successfully

	Name	Department	Basic	Bonus	Total Salary	Total Tax
0	Employee_1	HR	41291.0	8181.0	49472.0	4947.2
1	Employee_2	Sales	43241.0	8635.0	51876.0	5187.6
2	Employee_3	IT	NaN	8896.0	NaN	NaN
3	Employee_4	HR	47441.0	8685.0	56126.0	5612.6
4	Employee_5	Marketing	49767.0	7768.0	57535.0	5753.5
...	...	...	...	...	...	...
195	Employee_196	Finance	40048.0	7673.0	47721.0	4772.1
196	Employee_197	Finance	49372.0	9106.0	58478.0	5847.8
197	Employee_198	Finance	33814.0	6700.0	40514.0	4051.4
198	Employee_199	HR	51541.0	9817.0	61358.0	6135.8
199	Employee_200	Finance	36934.0	3959.0	40893.0	4089.3

200 rows × 6 columns

Learning Outcomes:

Experiment 2

Aim: Program of Data warehouse cleansing to input names from users (inconsistent) and format them.

Theory:

Dataset:

Code:

```
import pandas as pd
df = pd.read_csv('employees_inconsistent_simple.csv')
print(df.head(5))
```

```
   ID             Name  Department
0   1    DAvid miLlEr     Finance
1   2     MARyann doE  Operations
2   3     lISa TaYlOr       Sales
3   4  suNiL  wILSON     Finance
4   5     SUNiL daVIS     Finance
```

```
def cleanse_name(name):
  cleaned = name.strip()
  cleaned = " ".join(cleaned.split())
  cleaned = cleaned.title()
  return cleaned
df['cleaned_name']= df["Name"].apply(cleanse_name) #apply
transformation
print(df)
```

Output:

```
      ID               Name  Department   cleaned_name
0      1      DAvid miLlEr     Finance   David Miller
1      2       MARyann doE  Operations    Maryann Doe
2      3       lISa TaYlOr       Sales    Lisa Taylor
3      4    suNiL  wILSON     Finance   Sunil Wilson
4      5       SUNiL daVIS     Finance    Sunil Davis
..   ...               ...         ...            ...
195  196     anANDA TayLOr   Marketing  Ananda Taylor
196  197    suniL   JONEs     Finance    Sunil Jones
197  198     rAhul  wilsON     Finance   Rahul Wilson
198  199    PeTer   kUmAr          HR    Peter Kumar
199  200    maRYanN  sMITh          IT  Maryann Smith

[200 rows x 4 columns]
```

Learning Outcomes:

Experiment 3

Aim: Program of Data warehouse cleansing to remove redundancy in data.

Theory:

Dataset:

Code:

```
from fuzzywuzzy import fuzz
import pandas as pd
import matplotlib.pyplot as plt
#Load csv
df = pd.read_csv("customer_data.csv")
print("Original Dataset:\n", df)
```

```
Original Dataset:
   Customer_ID        Name              Email       Phone          Address
0          101    John Doe   john@example.com  1234567890       123 Elm St
1          102  Jane Smith   jane@example.com  2345678901       456 Oak St
2          103    john doe   john@example.com  1234567890   123 Elm Street
3          104 Alice Brown  alice@example.com  3456789012      789 Pine St
4          105  Bob Martin    bob@example.com  4567890123    321 Maple Ave
5          106  Jane Smith   jane@example.com  2345678901   456 Oak Street
6          107     Jon Doe    jon@example.com  1234567890       123 Elm St
7          108  Ally Brown  alice@example.com  3456789012  789 Pine Street
```

```
#Standardize names and addresses
df['Name'] = df['Name'].str.strip().str.title()
df['Address'] = df['Address'].str.strip().str.title()
print("\nAfter Standardization\n", df)
#Remove exact deuplicates
df_no_duplicates = df.drop_duplicates()
print("After removing duplicates\n", df_no_duplicates)
```

```
After removing duplicates
   Customer_ID        Name              Email       Phone          Address
0          101    John Doe   john@example.com  1234567890       123 Elm St
1          102  Jane Smith   jane@example.com  2345678901       456 Oak St
2          103    John Doe   john@example.com  1234567890   123 Elm Street
3          104 Alice Brown  alice@example.com  3456789012      789 Pine St
4          105  Bob Martin    bob@example.com  4567890123    321 Maple Ave
5          106  Jane Smith   jane@example.com  2345678901   456 Oak Street
6          107     Jon Doe    jon@example.com  1234567890       123 Elm St
7          108  Ally Brown  alice@example.com  3456789012  789 Pine Street
```

```
#Detect near Duplicates using fuzzy matching
names = df_no_duplicates['Name'].tolist()
print("Potential Near Duplicates");
for i in range(len(names)):
    for j in range(i + 1, len(names)):
        similarity = fuzz.ratio(names[i],
        names[j]); if similarity > 80: # Threshold
        for near
duplicates
```

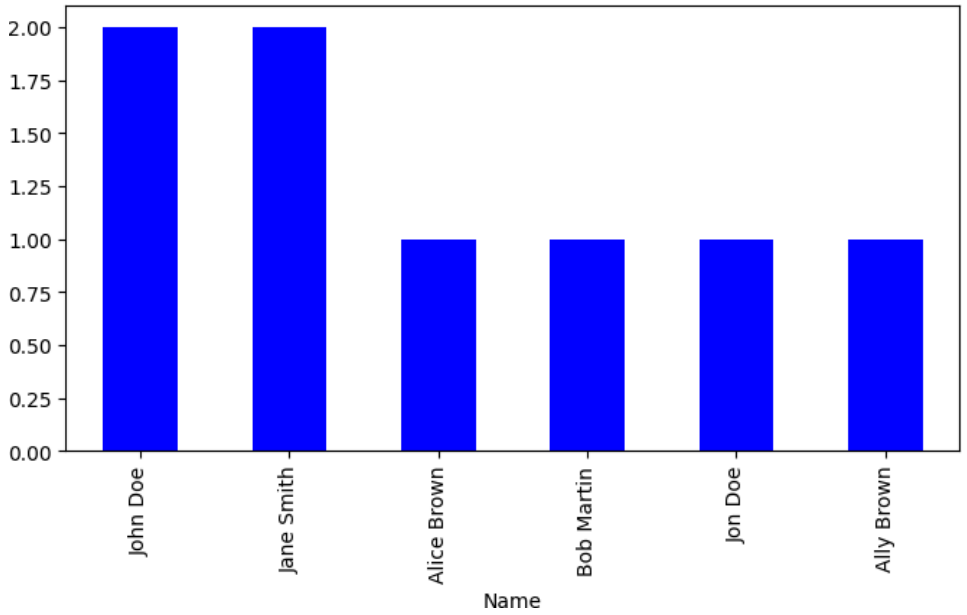
```
            print(f"'{names[i]}' and '{names[j]}'
(Similarity: {similarity}%)")
```

```
Potential Near Duplicates
'John Doe' and 'John Doe' (Similarity: 100%)
'John Doe' and 'Jon Doe' (Similarity: 93%)
'Jane Smith' and 'Jane Smith' (Similarity: 100%)
'John Doe' and 'Jon Doe' (Similarity: 93%)
```

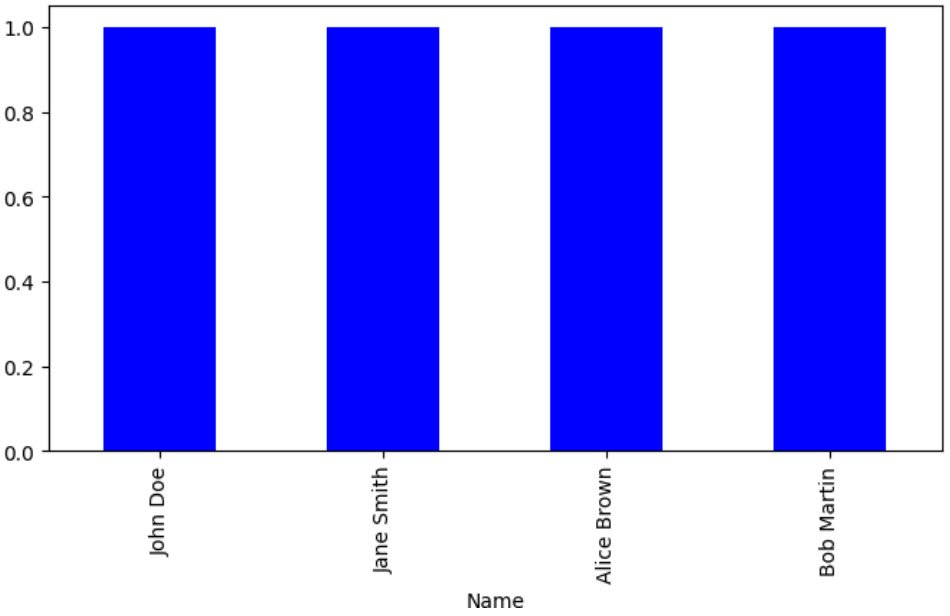
```
df_no_duplicates=df_no_duplicates.drop_duplicates(subset=
['Phone'], keep='first')
print("After Removing Duplicates : \n", df_no_duplicates)
```

```
After Removing Duplicates :
   Customer_ID         Name              Email       Phone        Address
0          101     John Doe   john@example.com  1234567890     123 Elm St
1          102   Jane Smith   jane@example.com  2345678901     456 Oak St
3          104  Alice Brown  alice@example.com  3456789012    789 Pine St
4          105    Bob Martin    bob@example.com  4567890123  321 Maple Ave
```

```
plt.figure(figsize=(8,4))
df['Name'].value_counts().plot(kind='bar', color='blue')
```



```
plt.figure(figsize=(8,4))
df_no_duplicates['Name'].value_counts().plot(kind='bar',
color='blue')
```




```
df_no_duplicates.to_csv("non_redundant_data.csv",
index=False)
print("File Saved”)
```

Output:

```
print(pd.read_csv("non_redundant_data.csv"))
```

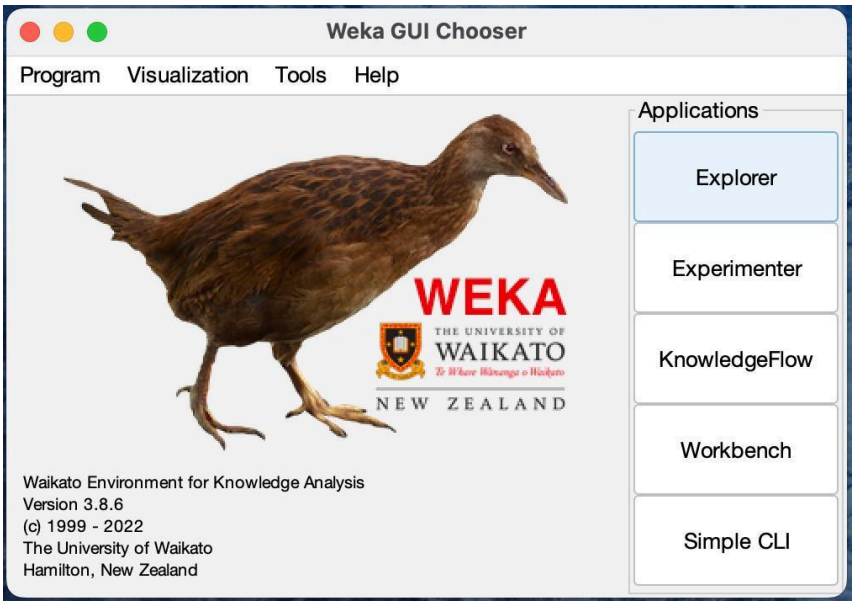
	Customer_ID	Name	Email	Phone	Address
0	101	John Doe	john@example.com	1234567890	123 Elm St
1	102	Jane Smith	jane@example.com	2345678901	456 Oak St
2	104	Alice Brown	alice@example.com	3456789012	789 Pine St
3	105	Bob Martin	bob@example.com	4567890123	321 Maple Ave

Learning Outcomes:

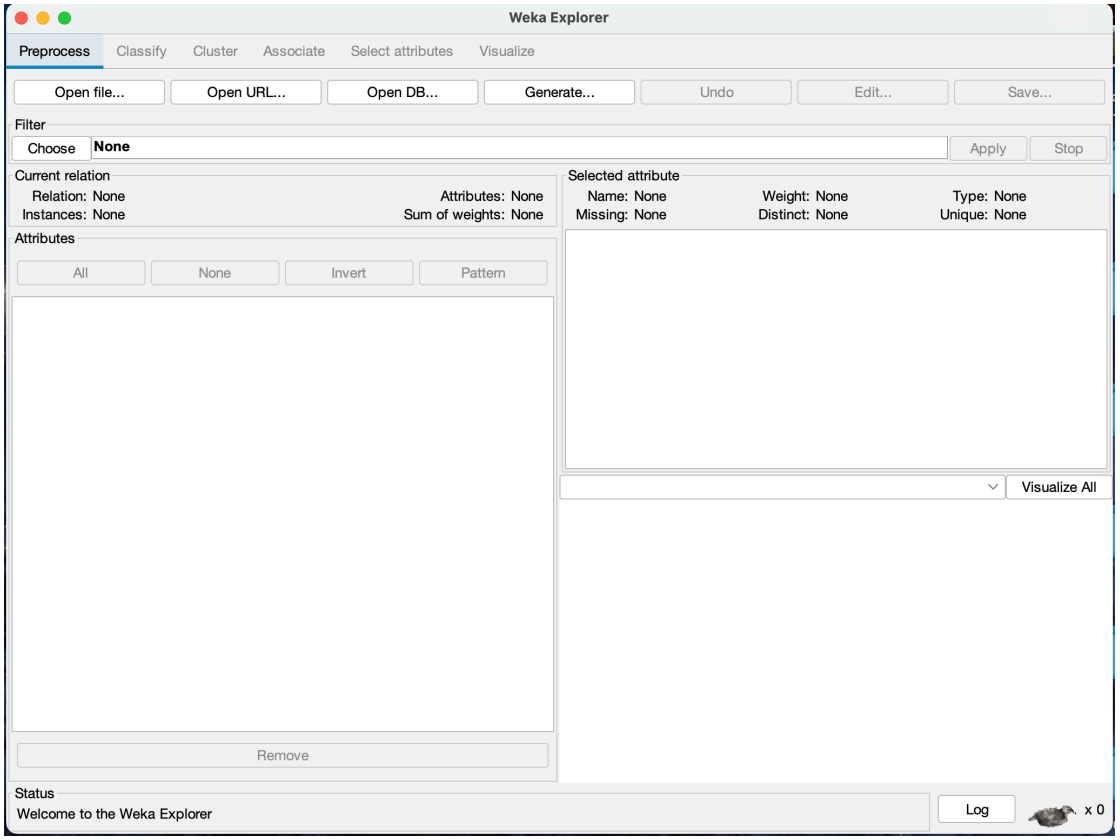
Experiment 4

Aim: Introduction to WEKA tool.

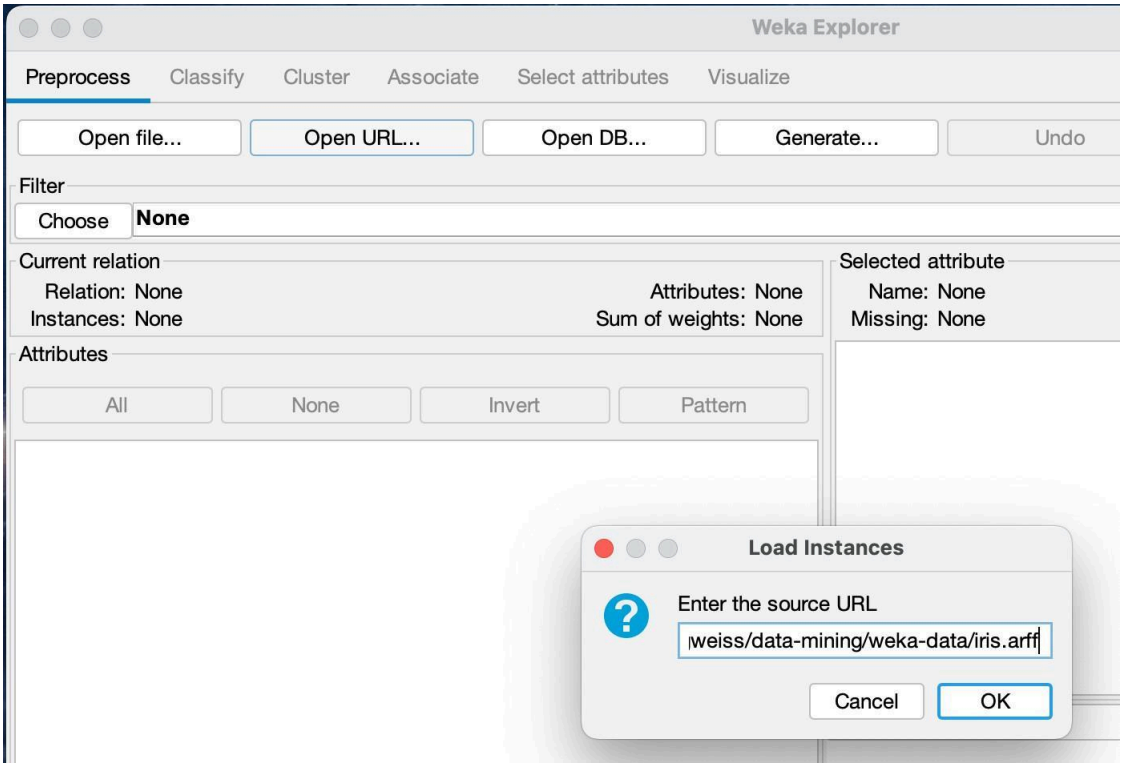
Theory:



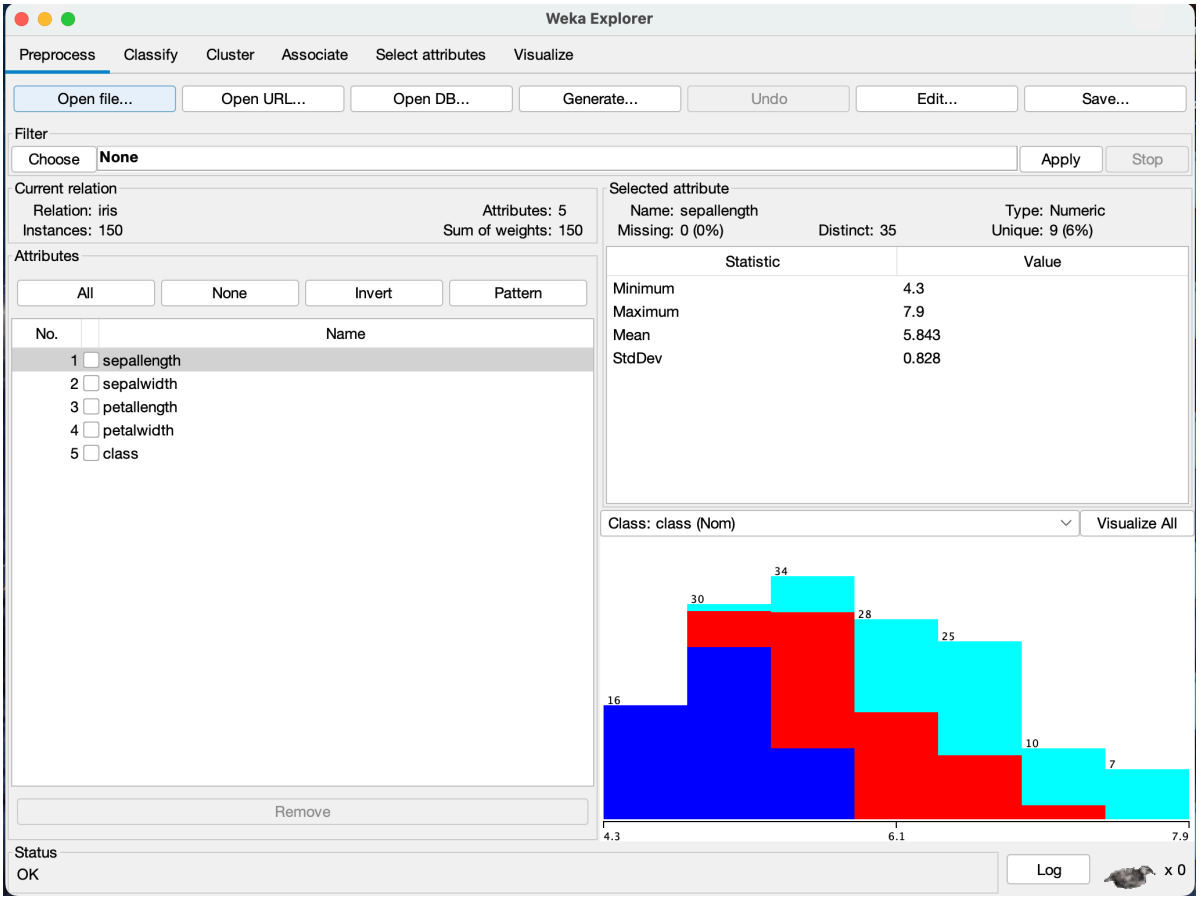
Homescreen of Weka



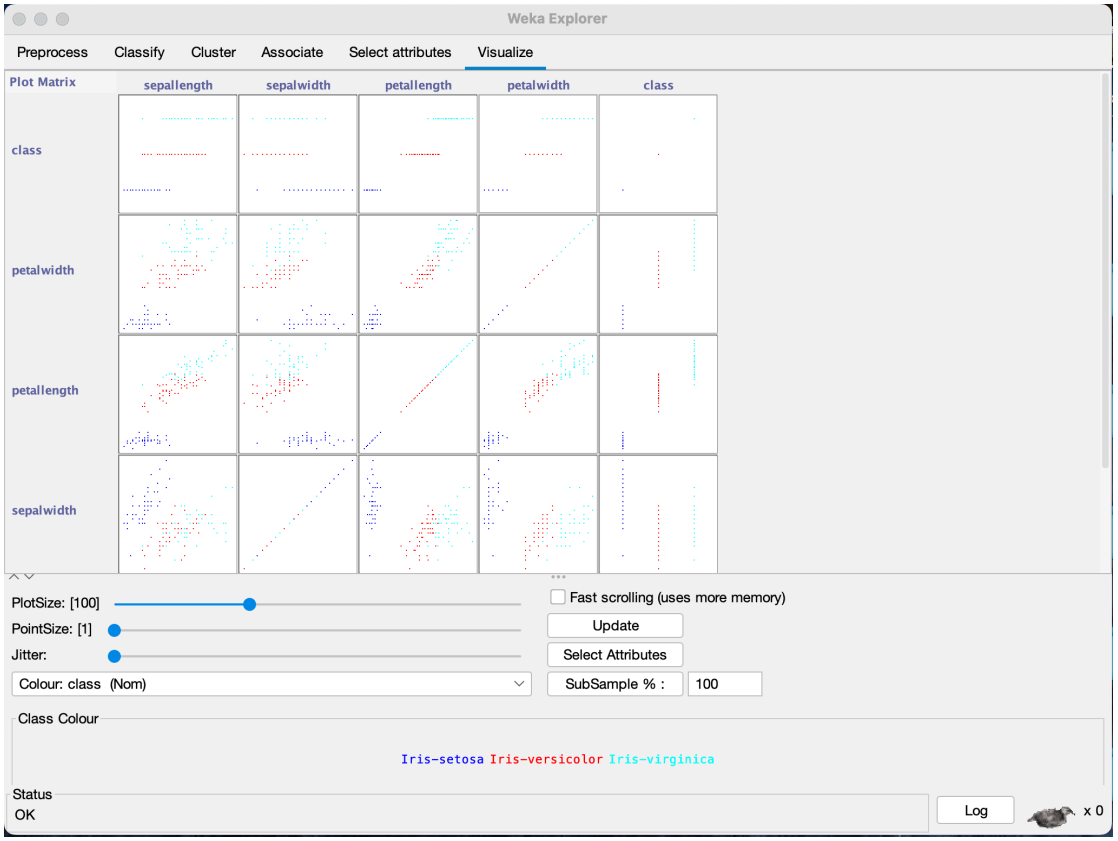
Weka Explorer



Opening Iris Dataset using URL



Preprocess Tab in Weka



Visualisation of Dataset

Learning Outcomes:

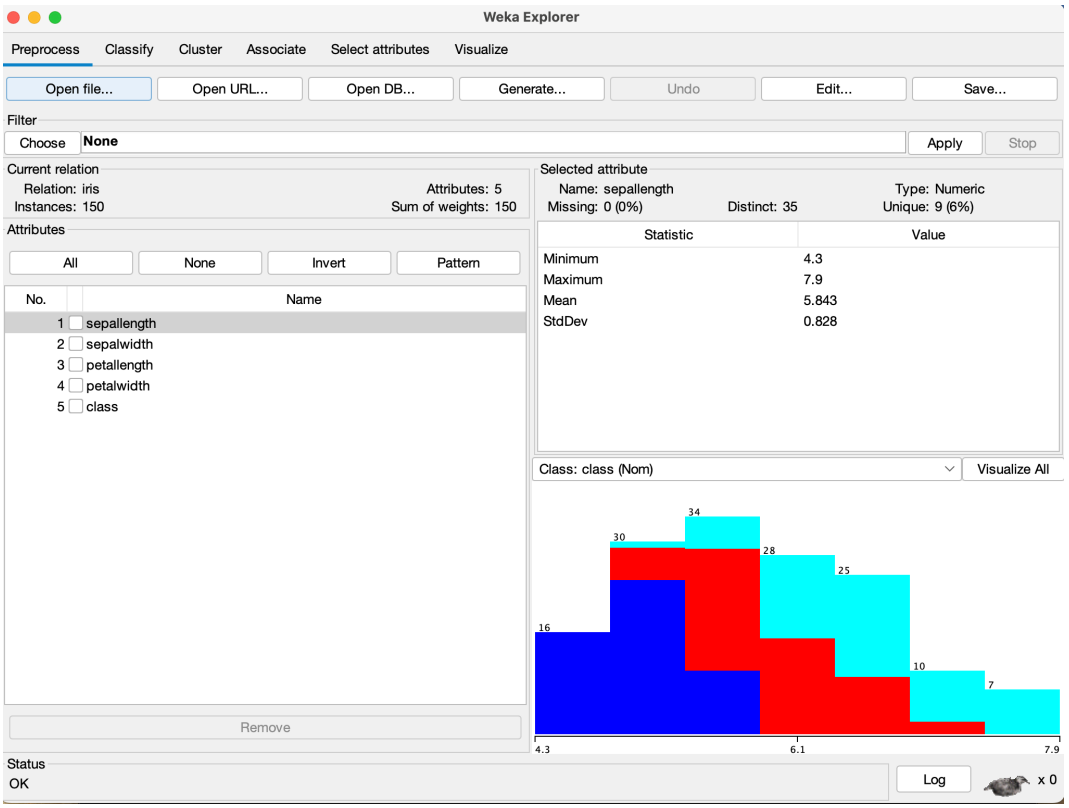
Experiment 5

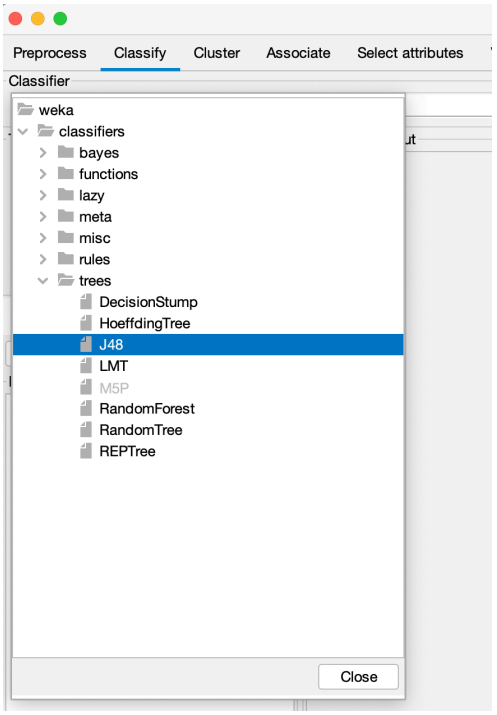
Aim: Implementation of Classification technique on ARFF files using WEKA.

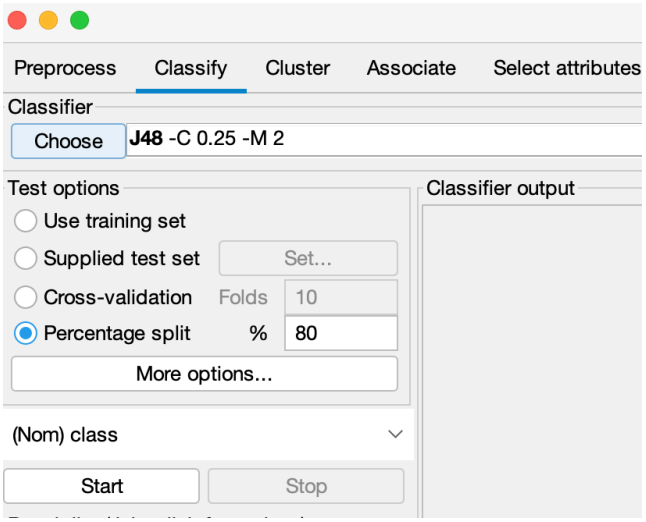
Theory:

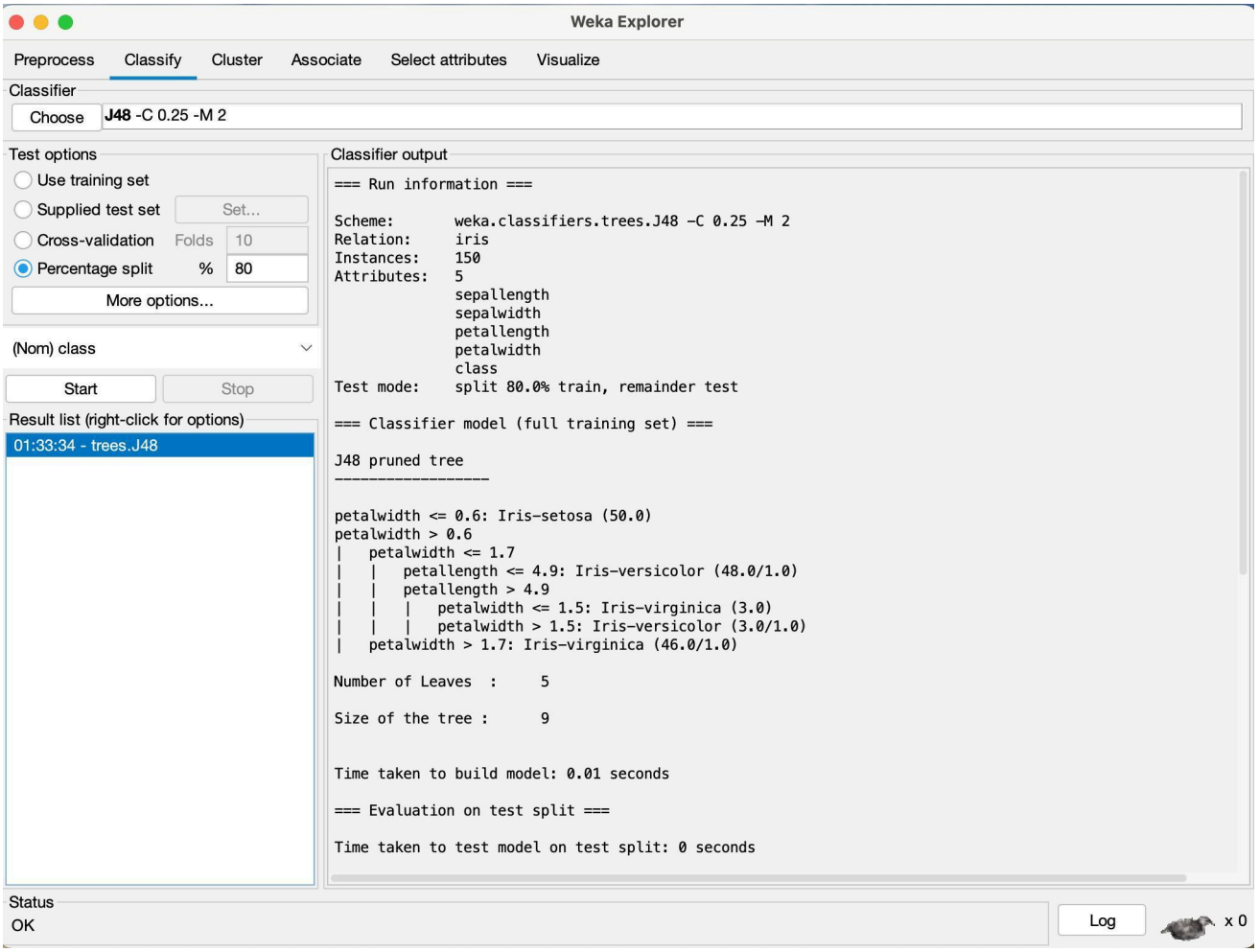
Dataset:

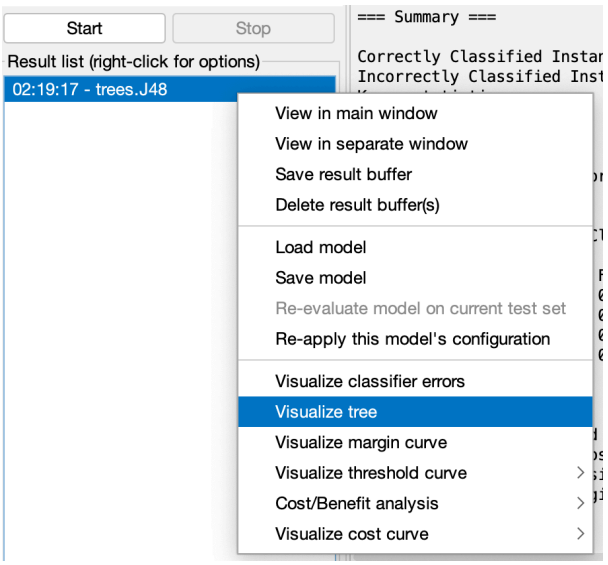
Snapshots:

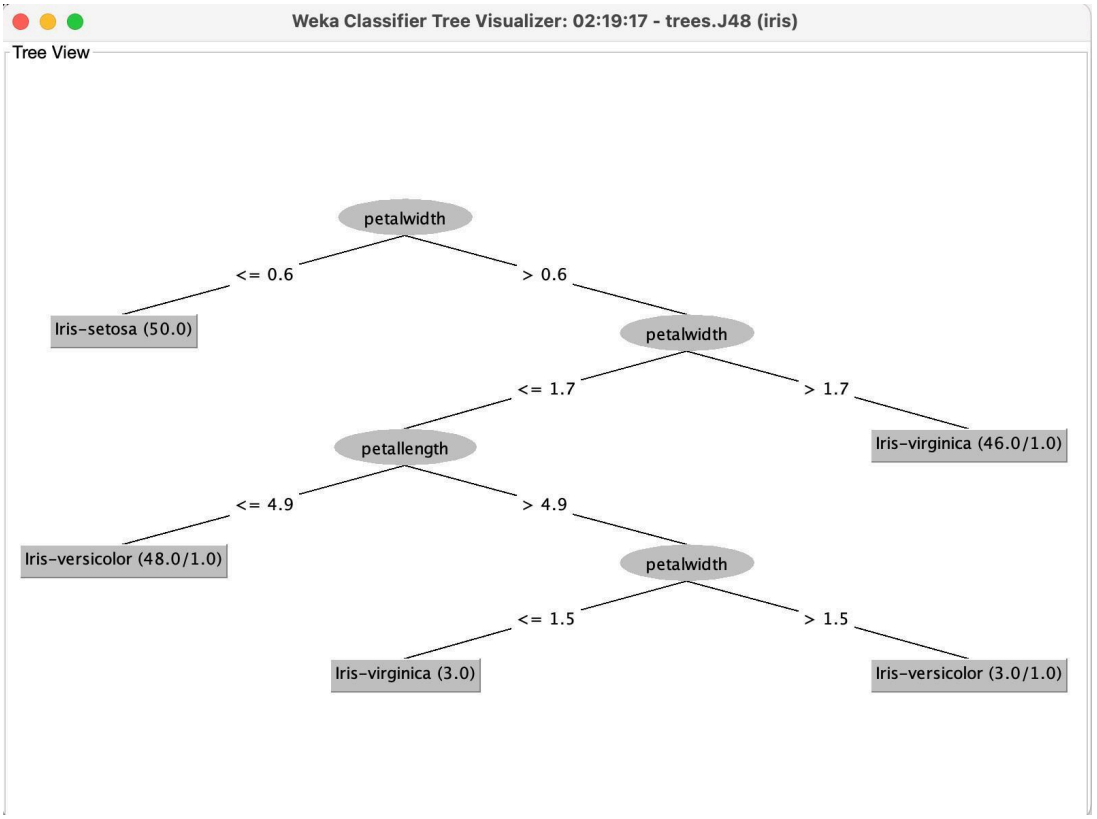

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150
Selected attribute
Name: sepallength
Missing: 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


Preprocess
Classify
Cluster
Associate
Select attributes
Classifier
weka
classifiers
bayes
functions
lazy
meta
misc
rules
trees
DecisionStump
HoeffdingTree
J48
LMT
M5P
RandomForest
RandomTree
REPTree
Close


Preprocess
Classify
Cluster
Associate
Select attributes
Classifier
Choose
J48 -C 0.25 -M 2
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
80
More options...
Classifier output
(Nom) class
Start
Stop


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
J48 -C 0.25 -M 2
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
80
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
01:33:34 - trees.J48
Classifier output
=== Run information ===
Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode: split 80.0% train, remainder test
=== Classifier model (full training set) ===
J48 pruned tree
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
| petalwidth <= 1.7
| | petallength <= 4.9: Iris-versicolor (48.0/1.0)
| | petallength > 4.9
| | | petalwidth <= 1.5: Iris-virginica (3.0)
| | | petalwidth > 1.5: Iris-versicolor (3.0/1.0)
| petalwidth > 1.7: Iris-virginica (46.0/1.0)
Number of Leaves : 5
Size of the tree : 9
Time taken to build model: 0.01 seconds
=== Evaluation on test split ===
Time taken to test model on test split: 0 seconds
Status
OK
Log
x 0


Start
Stop
Result list (right-click for options)
02:19:17 - trees.J48
=== Summary ===
Correctly Classified Instar
Incorrectly Classified Inst
View in main window
View in separate window
Save result buffer
Delete result buffer(s)
Load model
Save model
Re-evaluate model on current test set
Re-apply this model's configuration
Visualize classifier errors
Visualize tree
Visualize margin curve
Visualize threshold curve
Cost/Benefit analysis
Visualize cost curve



Output:

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M
2 Relation:   iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    split 80.0% train, remainder test

=== Classifier model (full training set)

=== J48 pruned tree
------------------

petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (48.0/1.0)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3.0)
|   |   |   petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|   petalwidth > 1.7: Iris-virginica

(46.0/1.0) Number of Leaves    :    5
```

```
Size of the tree :    9


Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances          30               100       %
Incorrectly Classified Instances         0                 0       %
Kappa statistic                          1
Mean absolute error                      0.0105
Root mean squared error                  0.0166
Relative absolute error                  2.3665 %
Root relative squared error              3.5274 %
Total Number of Instances               30

=== 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     Iris-setosa
                 1.000    0.000    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     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
 11  0  0 |  a = Iris-setosa
  0 10  0 |  b =
         Iris-versicolor
  0  0  9 |  c = Iris-virginica
```

Learning Outcomes:

Experiment 6

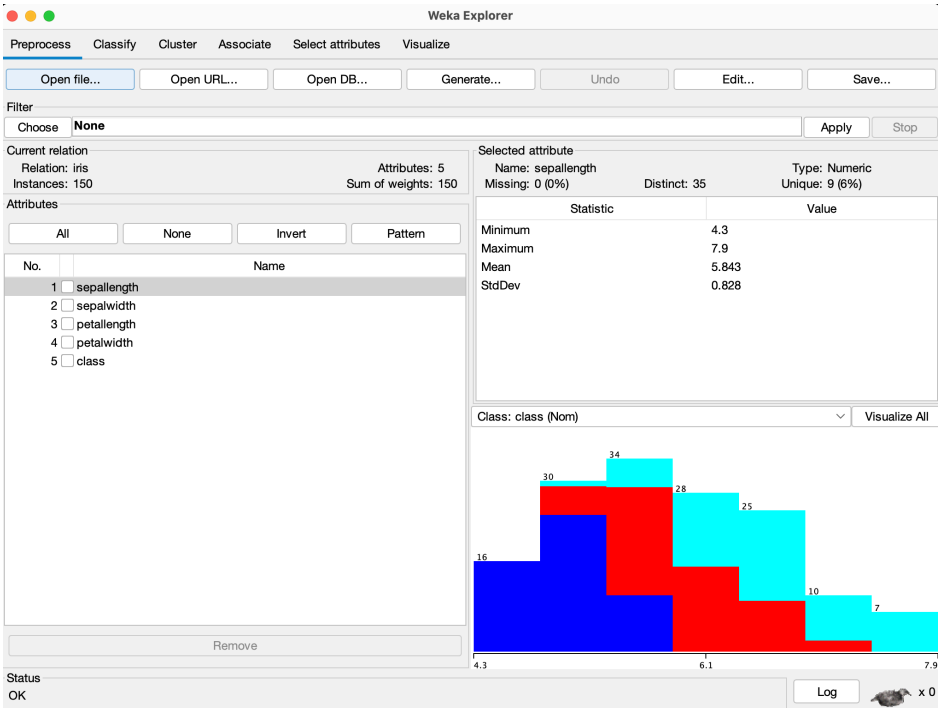
Aim: Implementation of Clustering technique on ARFF files using WEKA.

Theory:

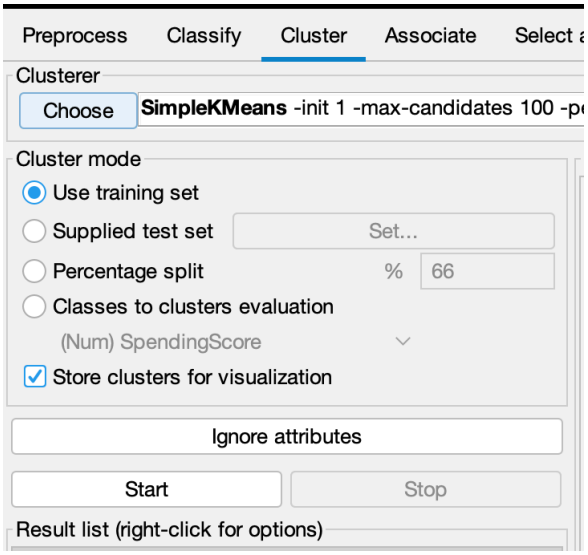
Dataset:

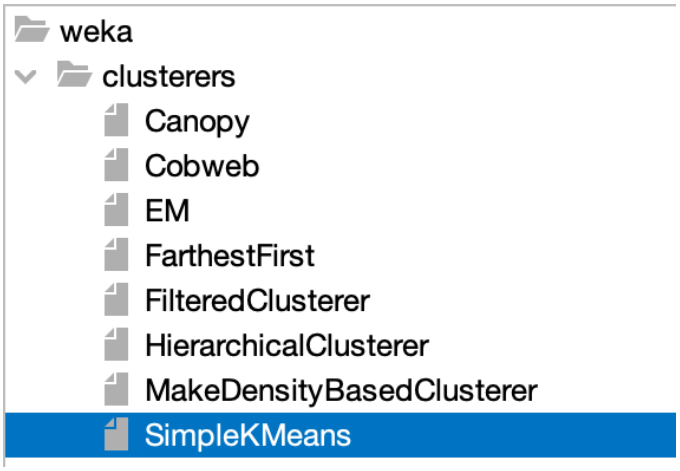
Procedure:

1. WEKA GUI Chooser -> Explorer -> Open .arff file

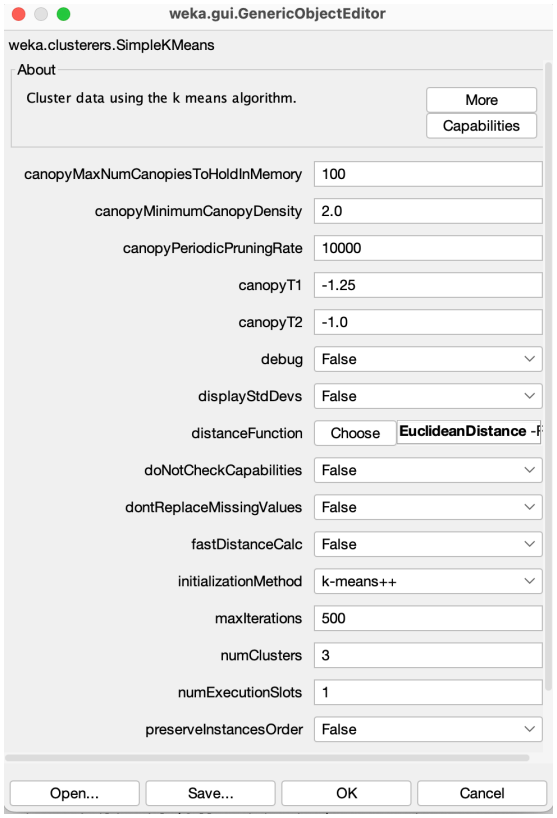


2. Go to Cluster tab -> Under Cluster Mode, keep: Use training set -> Click on Choose Pick: SimpleKMeans



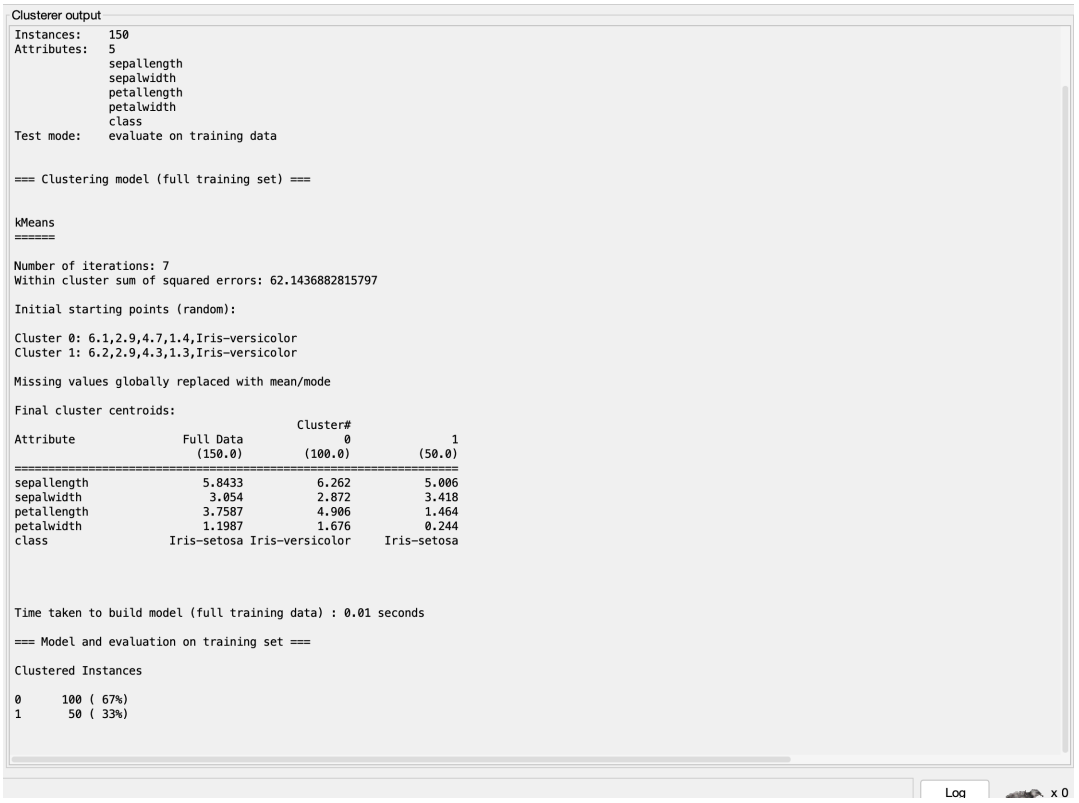


3. Set K-Means Parameters: [numClusters = 3, Distance function: EuclideanDistance ,Initialization method: k-means++] -> Click OK



5. Run the clustering

Output:



Learning Outcomes:

Experiment 7

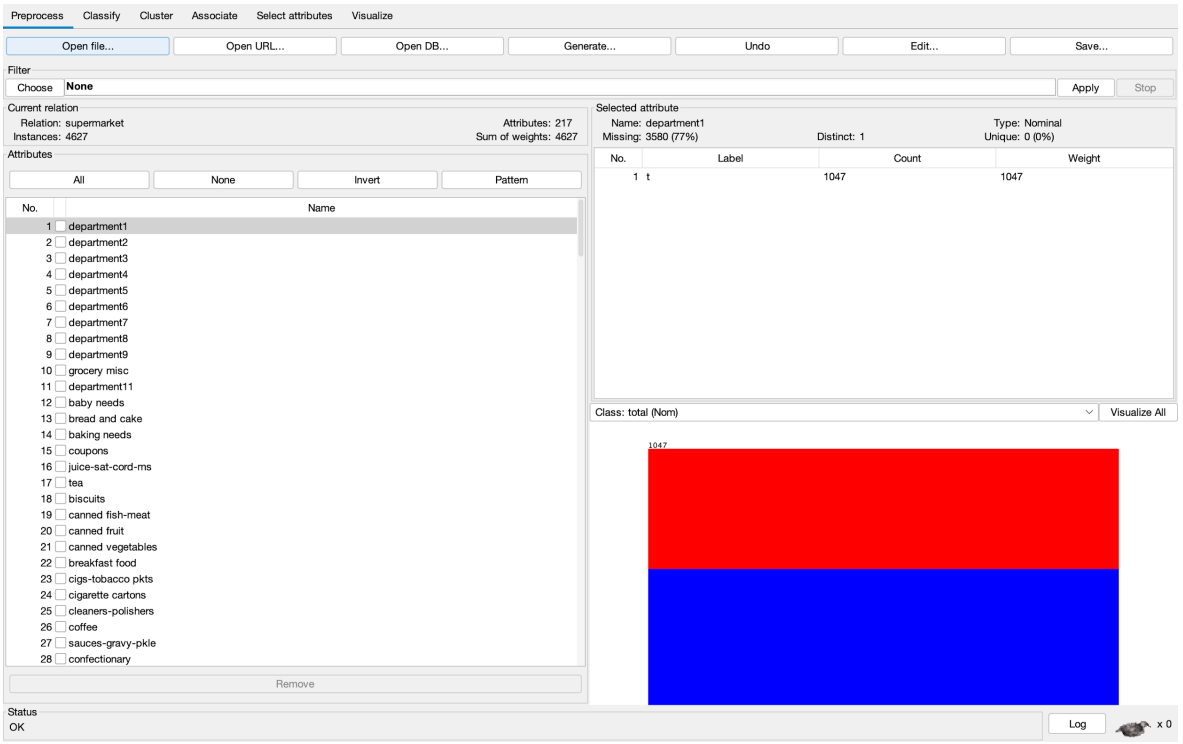
Aim: Implementation of Association Rule technique on ARFF files using WEKA.

Theory:

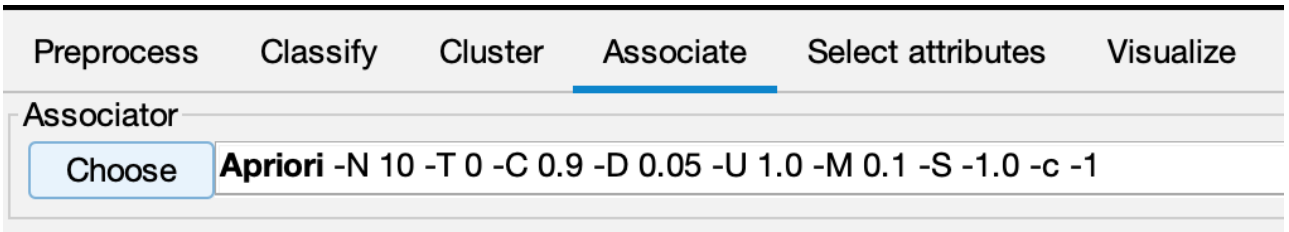
Dataset:

Procedure:

1. WEKA GUI Chooser -> Explorer -> Open .arff file



2. Go to Associate tab -> Choose Algorithm (Apriori)



3. Run by clicking on Start

Output:

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:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

Learning Outcomes:

Experiment 8

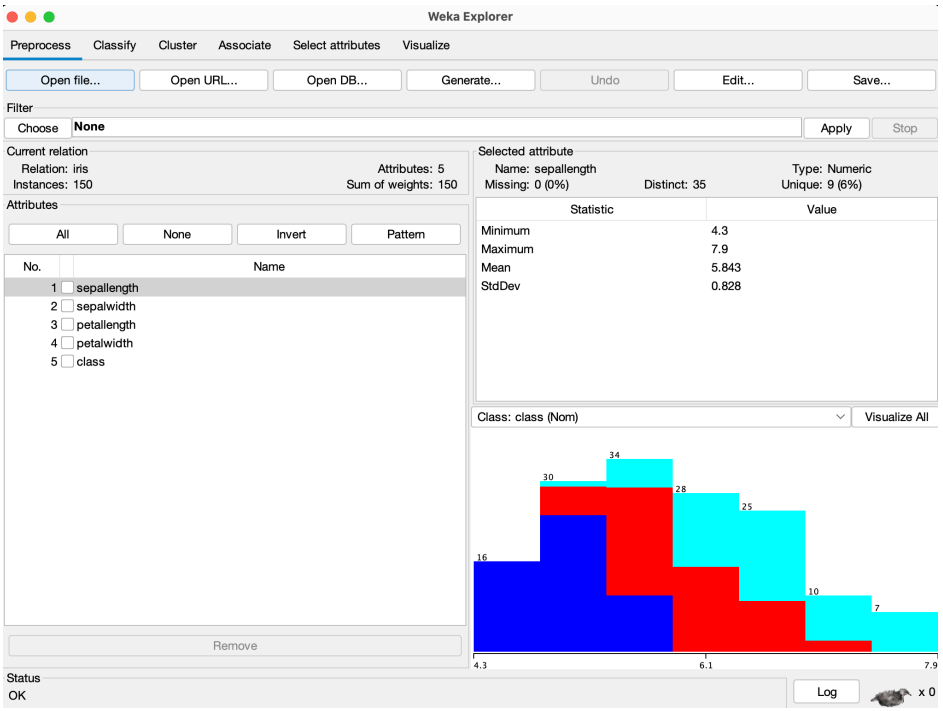
Aim: Implementation of Visualization technique on ARFF files using WEKA.

Theory:

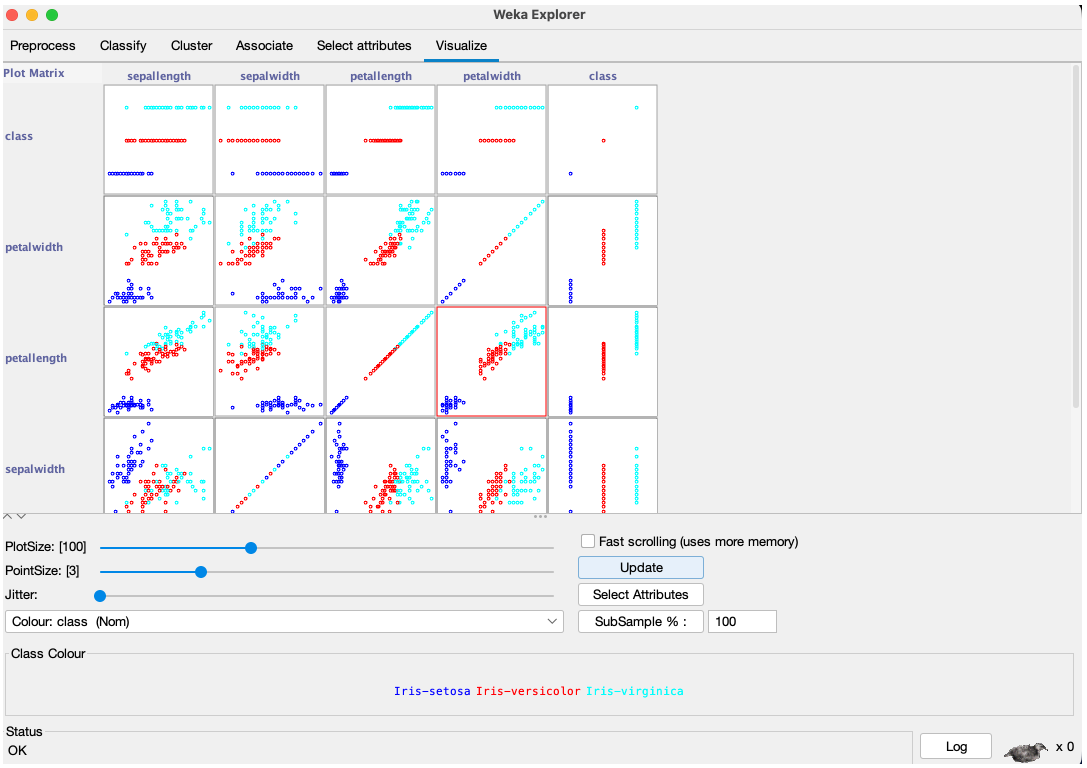
Dataset:

Procedure:

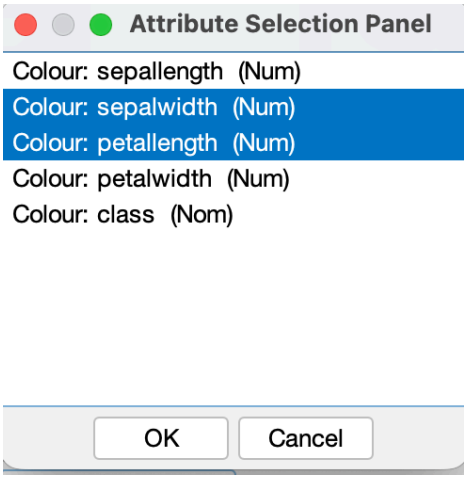
1. WEKA GUI Chooser -> Explorer -> Open .arff file

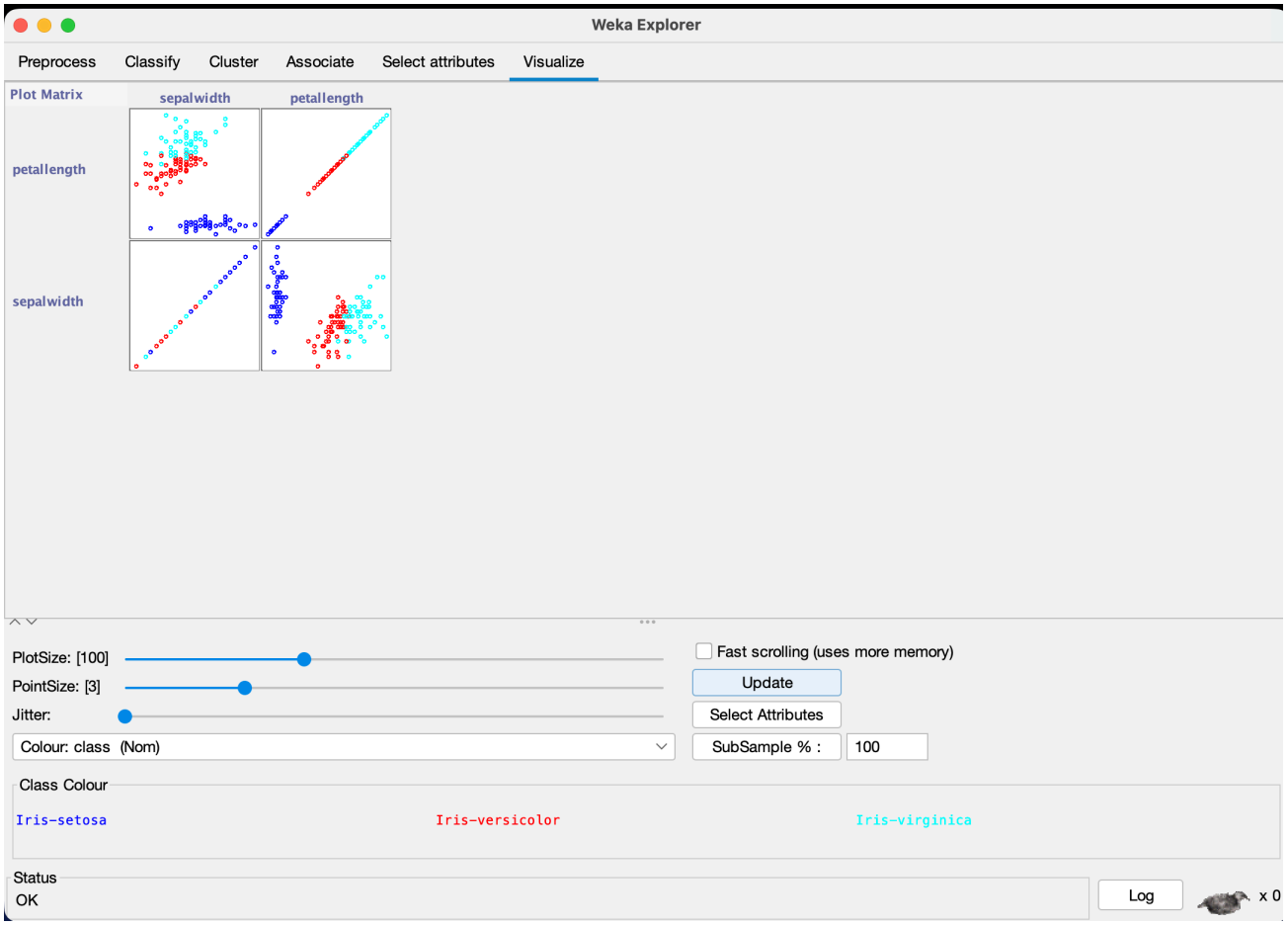


2. Go to Visualize tab.
   a. Click Visualize All



b. Visualizing Selected Attributes





Learning Outcomes:

Experiment 9

Aim: Apply the Apriori Algorithm focusing on discriminating between patients with Parkinson's disease and other neurological disorders using the Voice Recording Dataset.

Theory:

Dataset:

Code:

```
import warnings, gc, io
warnings.filterwarnings('ignore')


!pip install pandas mlxtend

import pandas as pd

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
parkinsons/parkinsons.data"
df = pd.read_csv(url)
df.head()

# Drop name column
df = df.drop("name", axis=1)

# Convert status column into labels
df['status'] = df['status'].map({0: "Healthy", 1: "Parkinsons"})

# Binning continuous features
for col in df.columns:
    if col != "status":
        df[col] = pd.qcut(df[col], q=3, labels=['Low', 'Medium',
'High'])


from mlxtend.preprocessing import TransactionEncoder

# Convert df to list of lists (transactions)
transactions = []
for i in range(len(df)):
    row_items = []
    for col in df.columns:
        row_items.append(f"{col}={df[col].iloc[i]}")
    transactions.append(row_items)

# Transaction Encoder
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)

df_encoded = pd.DataFrame(te_ary, columns=te.columns_)
df_encoded.head()

from mlxtend.frequent_patterns import apriori, association_rules
# find frequent itemsets
freq_items = apriori(df_encoded, min_support=0.3,
use_colnames=True)
```

```
freq_items.sort_values(by="support", ascending=False).head()


rules = association_rules(freq_items, metric="confidence",
min_threshold=0.7)

# Filter rules predicting Parkinson's disease
rules_pd =
rules[rules['consequents'].astype(str).str.contains("status=Parkin
sons")]

rules_pd.sort_values(by="confidence", ascending=False).head(10)


print("\nAssociation Rules Predicting Parkinson's Disease:\n")
display(rules_pd[['antecedents','consequents','support','confidenc
e','lift']])
```

Output:

Association Rules Predicting Parkinson's Disease:

	antecedents	consequents	support	confidence	lift
0	(D2=High)	(status=Parkinsons)	0.307692	0.923077	1.224490
1	(HNR=Low)	(status=Parkinsons)	0.302564	0.907692	1.204082
8	(Jitter:DDP=High)	(status=Parkinsons)	0.312821	0.938462	1.244898
26	(MDVP:APQ=High)	(status=Parkinsons)	0.328205	0.984615	1.306122
28	(MDVP:Fhi(Hz)=Medium)	(status=Parkinsons)	0.312821	0.938462	1.244898
...	...	...	...	...	...
560	(Shimmer:APQ3=High, Shimmer:DDA=High)	(MDVP:Shimmer=High, status=Parkinsons, Shimmer...	0.307692	0.923077	2.903226
561	(Shimmer:APQ5=High)	(MDVP:Shimmer=High, status=Parkinsons, Shimmer...	0.307692	0.923077	3.000000
562	(MDVP:Shimmer=High)	(status=Parkinsons, Shimmer:APQ5=High, Shimmer...	0.307692	0.923077	2.950820
563	(Shimmer:APQ3=High)	(MDVP:Shimmer=High, status=Parkinsons, Shimmer...	0.307692	0.923077	3.000000
564	(Shimmer:DDA=High)	(MDVP:Shimmer=High, status=Parkinsons, Shimmer...	0.307692	0.923077	3.000000

135 rows × 5 columns

Learning Outcomes:

Experiment 10

Aim: Implement the K-Means Clustering and K-Nearest Neighbors (KNN) algorithms on the Parkinson's Disease Voice Recording Dataset from the UCI Machine Learning Repository. Perform classification and clustering of patients based on their vocal features, analyze clustering results, evaluate classification accuracy, and compare the performance of both methods.

Theory:

Dataset:

Code:

```
# Install required libraries
!pip install scikit-learn pandas numpy matplotlib seaborn -q

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (classification_report,
accuracy_score,
adjusted_rand_score,                    silhouette_scor

confusion_matrix                        e,

                                        confusion_matri

                                        x) # Added

from sklearn.decomposition import PCA # Added PCA
import warnings
warnings.filterwarnings('ignore')
print("\n Parkinson's Analysis: K-Means & KNN")

# LOAD & PREPROCESS DATA
try:
    # Load from direct URL
    url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/parkinsons/parkinsons.data"
    df = pd.read_csv(url)

    y = df['status'].values  # 1 = Parkinson's, 0 = Healthy
    X = df.drop(['name', 'status'], axis=1)
    print(f"Dataset loaded: {X.shape[0]} samples, {X.shape[1]}
features")
except Exception as e:
    print(f"Error loading data: {e}")
    exit()

# Split data for supervised learning (KNN)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # For K-Means (all data)
X_train_scaled = scaler.fit_transform(X_train) # For KNN (train)
X_test_scaled = scaler.transform(X_test) # For KNN (test)
```

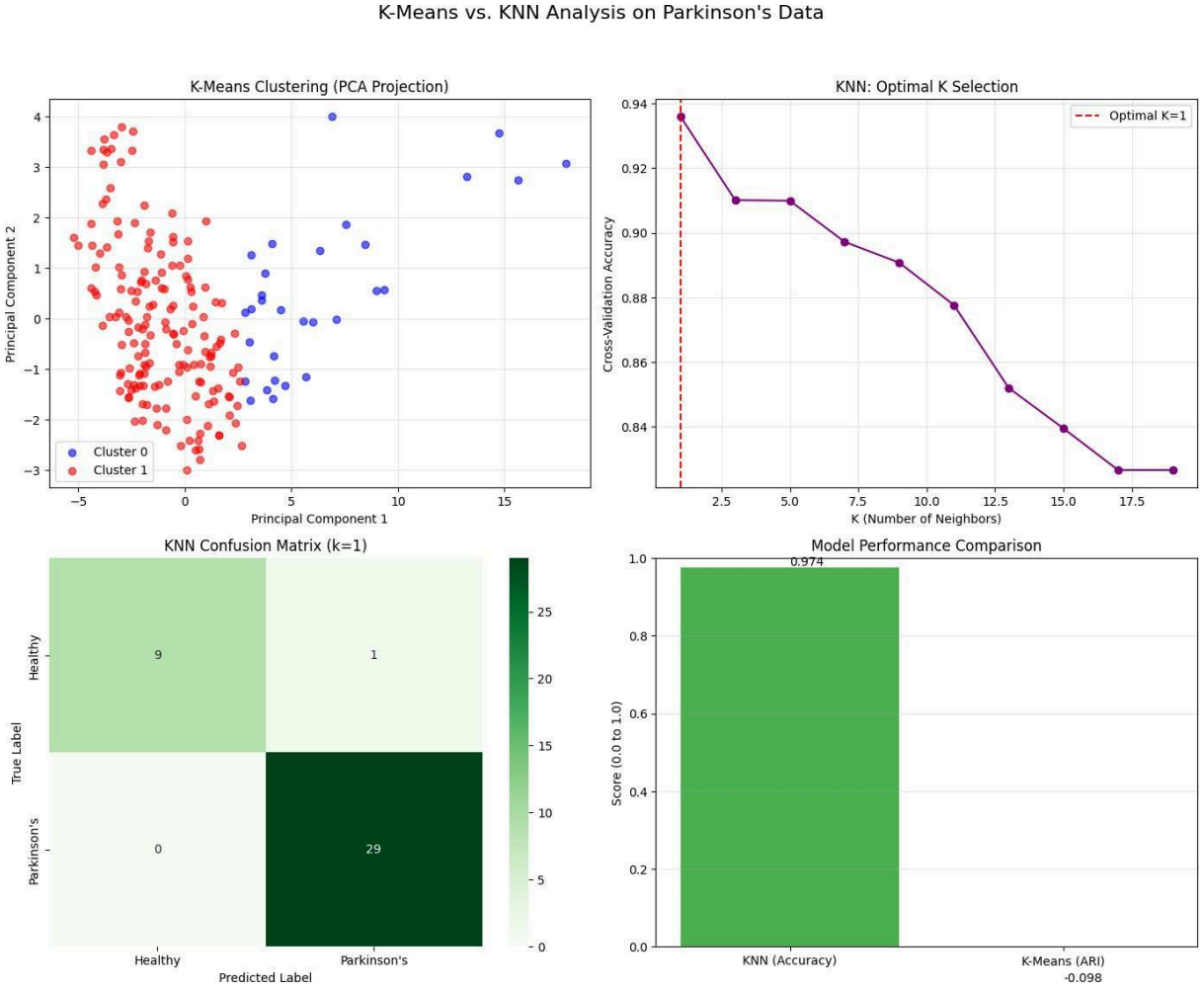
```
print("Data split and standardized")
# Apply PCA for visualization (fit on all scaled data)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
print("PCA applied for visualization")

# K-MEANS CLUSTERING (UNSUPERVISED)
print("\nK-Means Clustering (k=2)")

# Apply K-Means with K=2 (since we know there are 2 classes)
kmeans_2 = KMeans(n_clusters=2, random_state=42, n_init=10)
cluster_labels = kmeans_2.fit_predict(X_scaled)

# Clustering Evaluation Metrics
sil_score = silhouette_score(X_scaled, cluster_labels)
ari_score = adjusted_rand_score(y, cluster_labels) # Compare
clusters to true labels

print(f"Silhouette Score: {sil_score:.4f}")
print(f"Adjusted Rand Index (ARI): {ari_score:.4f} (vs true
labels)")

# K-NEAREST NEIGHBORS (SUPERVISED)
print("\nK-Nearest Neighbors (KNN)")

# Find optimal K for KNN using cross-validation
k_values = range(1, 20, 2)
cv_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_scaled, y_train, cv=5,
scoring='accuracy')
    cv_scores.append(scores.mean())

# Get the best K
optimal_k = k_values[np.argmax(cv_scores)]
print(f"Optimal K found: {optimal_k} (CV Accuracy:
{max(cv_scores):.4f})")

# Train KNN with optimal K
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)
knn_optimal.fit(X_train_scaled, y_train)

# Predictions
y_pred_test = knn_optimal.predict(X_test_scaled)

# Evaluation
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f"\nKNN Test Accuracy (k={optimal_k}): {test_accuracy:.4f}")
print(f"Classification Report (Test Set):")
print(classification_report(y_test, y_pred_test,
```

```
target_names=['Healthy',"Parkinson's"]))

 # VISUALIZATION
 print("\nVisualizations")

 fig, axes = plt.subplots(2, 2, figsize=(14, 12))
 fig.suptitle('K-Means vs. KNN Analysis on Parkinson\'s Data',
 fontsize=16)

 # --- Plot 1: K-Means Clustering (PCA) ---
 ax1 = axes[0, 0]
 ax1.scatter(X_pca[cluster_labels == 0, 0], X_pca[cluster_labels ==
 0, 1], c='blue', label='Cluster 0', alpha=0.6)
 ax1.scatter(X_pca[cluster_labels == 1, 0], X_pca[cluster_labels ==
 1, 1], c='red', label='Cluster 1', alpha=0.6)
 ax1.set_title('K-Means Clustering (PCA Projection)')
 ax1.set_xlabel('Principal Component 1')
 ax1.set_ylabel('Principal Component 2')
 ax1.legend()
 ax1.grid(True, alpha=0.3)

 # --- Plot 2: KNN 'K' Selection ---
 ax2 = axes[0, 1]
 ax2.plot(k_values, cv_scores, 'o-', c='purple')
 ax2.axvline(optimal_k, color='red', linestyle='--',
 label=f'Optimal K={optimal_k}')
 ax2.set_title('KNN: Optimal K Selection')
 ax2.set_xlabel('K (Number of Neighbors)')
 ax2.set_ylabel('Cross-Validation Accuracy')
 ax2.legend()
 ax2.grid(True, alpha=0.3)

 # --- Plot 3: KNN Confusion Matrix ---
 ax3 = axes[1, 0]
 cm = confusion_matrix(y_test, y_pred_test)
 sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', ax=ax3,
             xticklabels=['Healthy', "Parkinson's"],
             yticklabels=['Healthy', "Parkinson's"])
 ax3.set_title(f'KNN Confusion Matrix (k={optimal_k})')
 ax3.set_ylabel('True Label')
 ax3.set_xlabel('Predicted Label')

 # --- Plot 4: Performance Comparison ---
 ax4 = axes[1, 1]
 methods = ['KNN (Accuracy)', 'K-Means (ARI)']
 scores = [test_accuracy, ari_score]
 colors = ['#4CAF50', '#2196F3']
 bars = ax4.bar(methods, scores, color=colors)
 ax4.set_title('Model Performance Comparison')
 ax4.set_ylabel('Score (0.0 to 1.0)')
 ax4.set_ylim(0, 1.0)
```

```
# Add text labels
for i, bar in enumerate(bars):
    yval = bar.get_height()
    ax4.text(bar.get_x() + bar.get_width()/2.0, yval,
f'{scores[i]:.3f}', va='bottom')
ax4.grid(True, axis='y', alpha=0.3)


plt.tight_layout(rect=[0, 0.03, 1, 0.95]) # Adjust for suptitle
plt.savefig('kmeans_knn_minimal_analysis.png')
plt.show()
```

Output:

```
Classification Report (Test Set):
              precision    recall  f1-score   support

     Healthy       1.00      0.90      0.95        10
  Parkinson's      0.97      1.00      0.98        29

    accuracy                           0.97        39
   macro avg       0.98      0.95      0.97        39
weighted avg       0.98      0.97      0.97        39
```



Learning Outcomes: