# Experiment 1

**Aim: Study of ETL process and its tools. Theory:**

**Dataset:**



**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:**

****

**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))



def cleanse\_name(name): cleaned = name.strip()

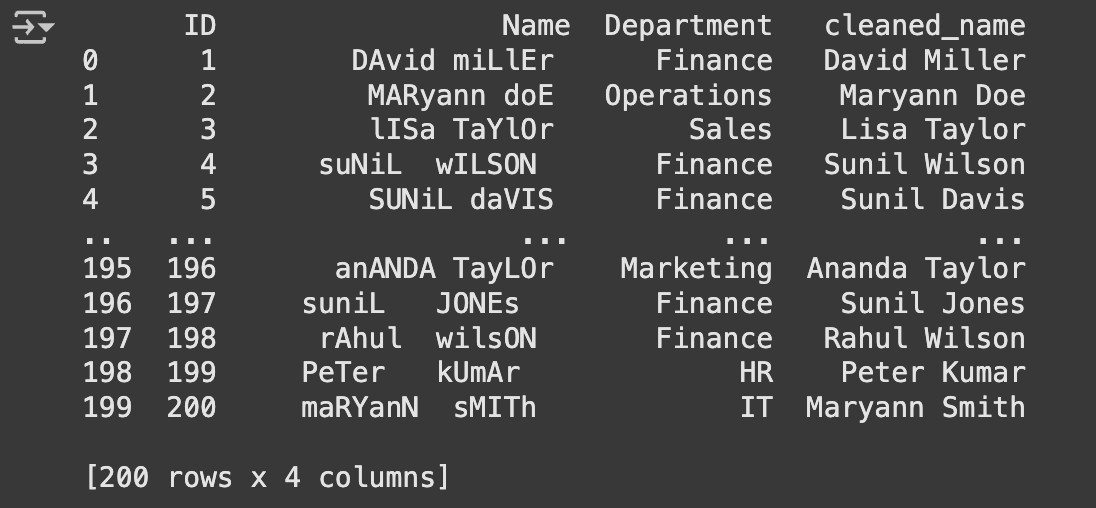
cleaned = " ".join(cleaned.split()) cleaned = cleaned.title()

return cleaned

df['cleaned\_name']= df["Name"].apply(cleanse\_name) #apply transformation

print(df)

**Output:**

****

**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)



#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)



#Detect near Duplicates using fuzzy matching names = df\_no\_duplicates['Name'].tolist() print("Potential Near Duplicates");

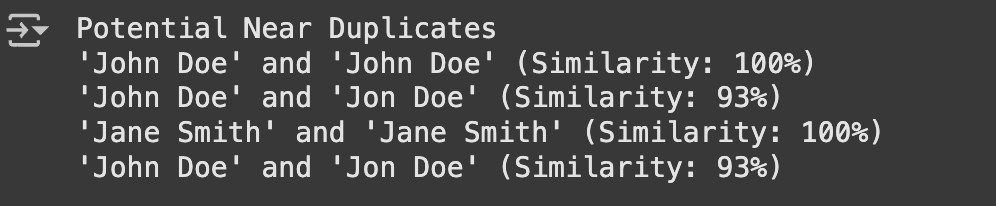
for i in range(len(names)):

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}%)")

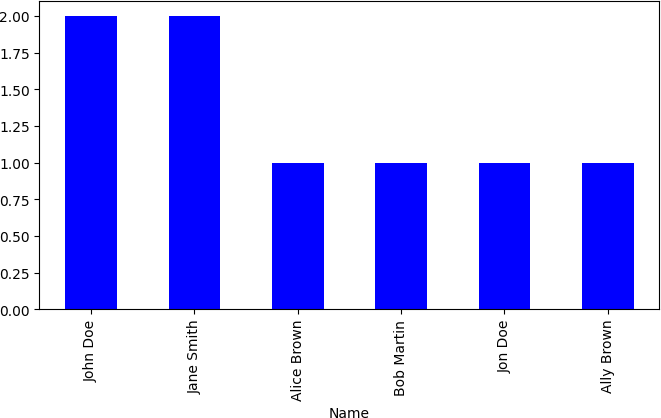


df\_no\_duplicates=df\_no\_duplicates.drop\_duplicates(subset= ['Phone'], keep='first')

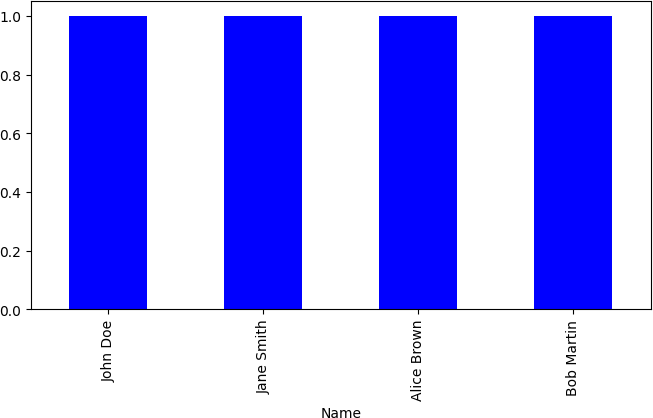
print("After Removing Duplicates : \n", df\_no\_duplicates)



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



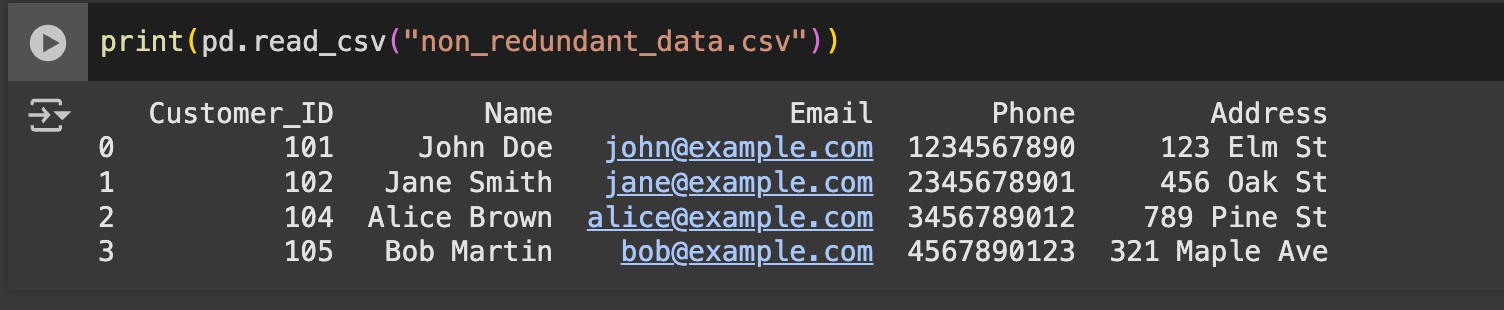
plt.figure(figsize=(8,4)) df\_no\_duplicates['Name'].value\_counts().plot(kind='bar', color='blue')



df\_no\_duplicates.to\_csv("non\_redundant\_data.csv", index=False)

print("File Saved”)

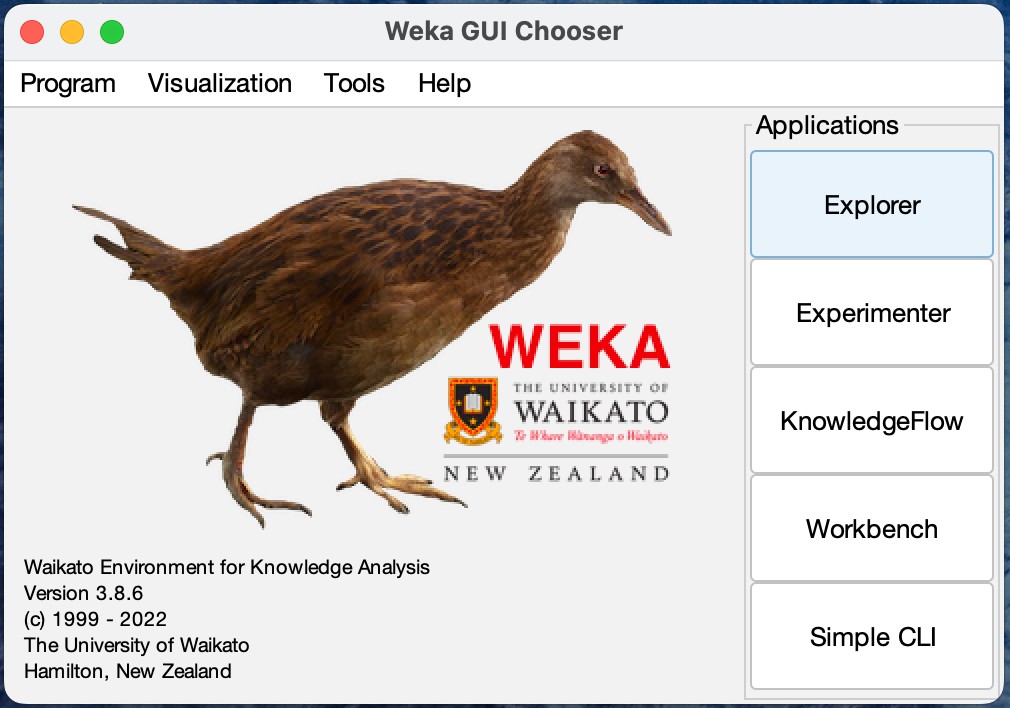
**Output:**

****

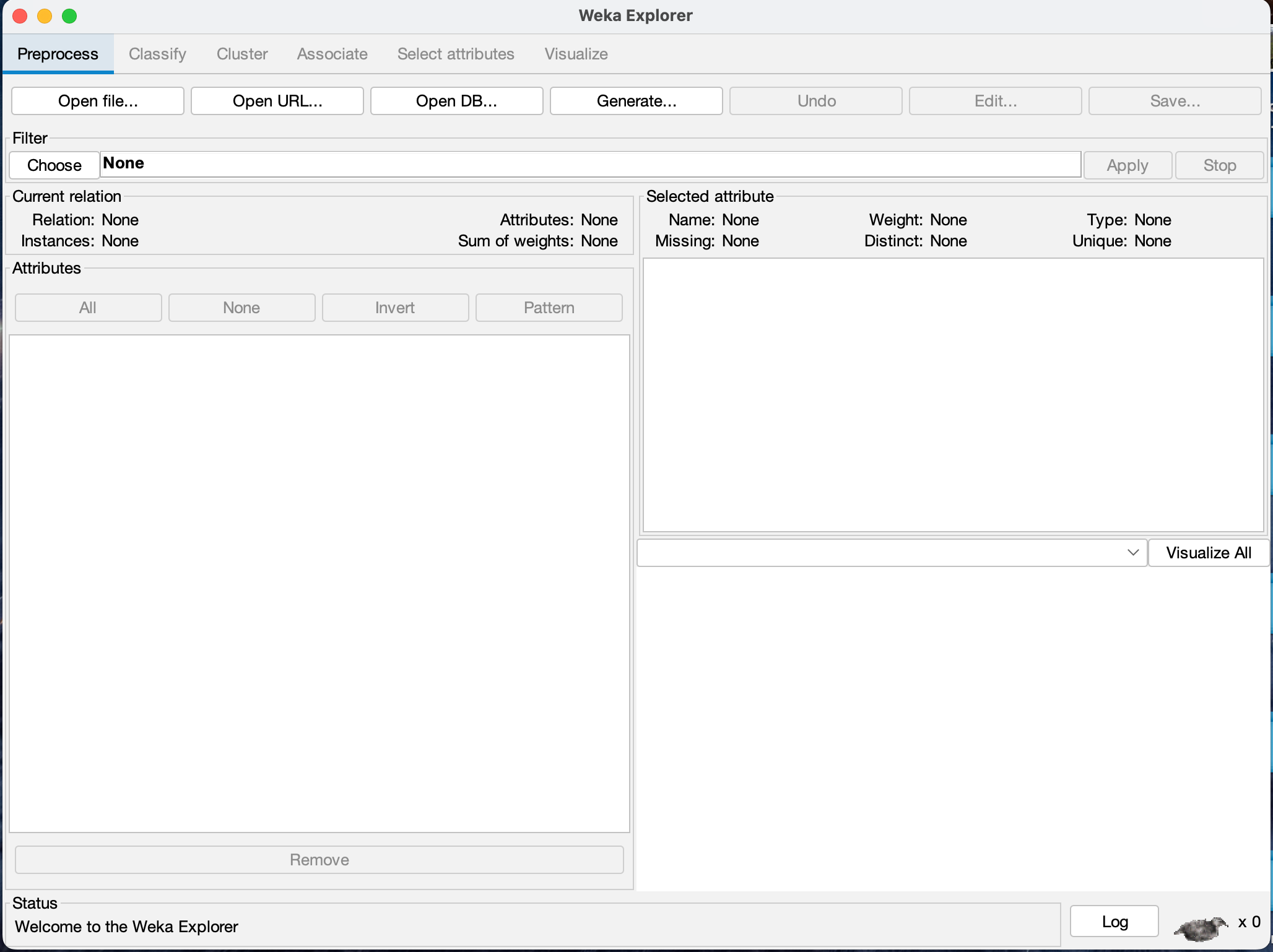
**Learning Outcomes:**

# Experiment 4

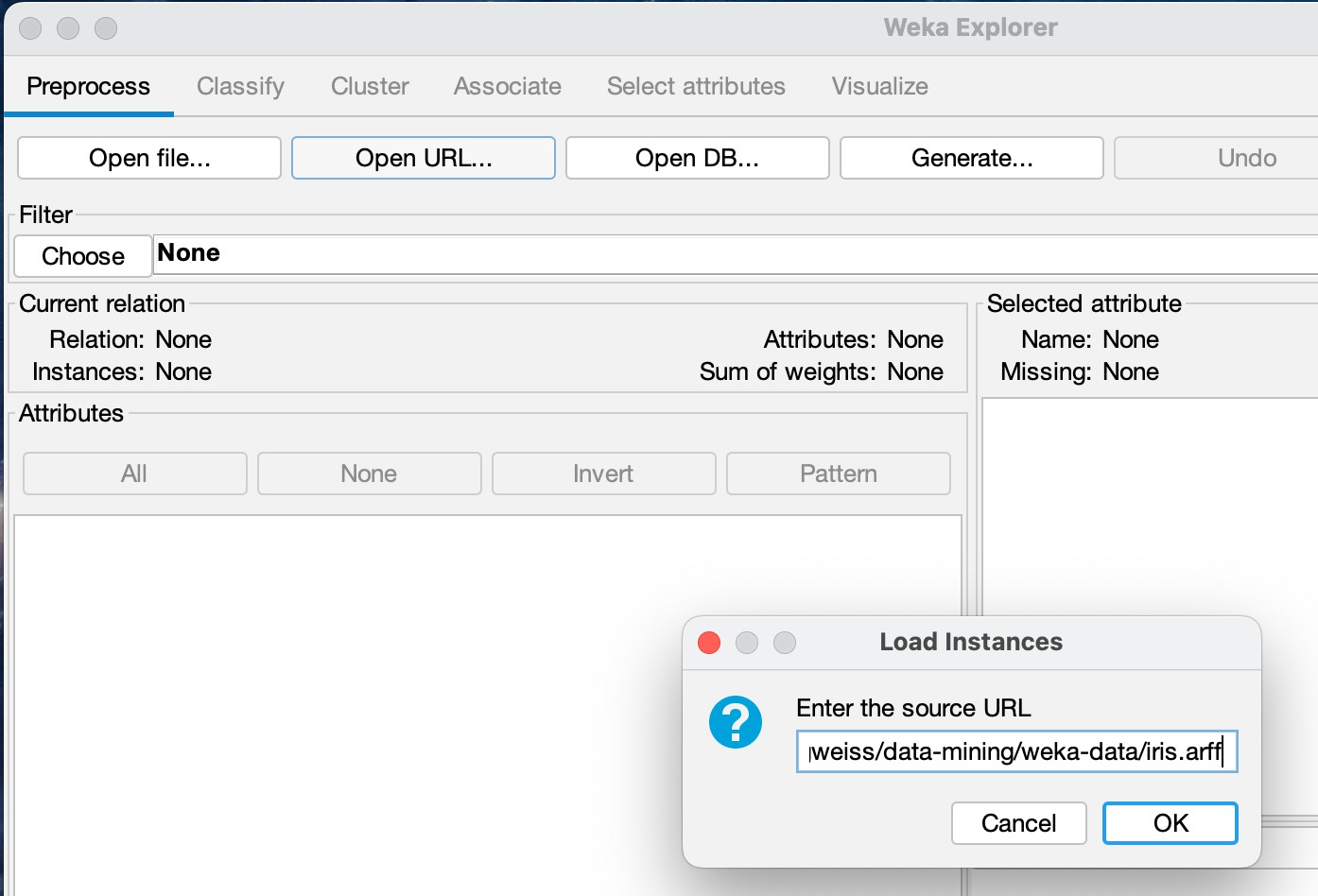
**Aim: Introduction to WEKA tool. Theory:**



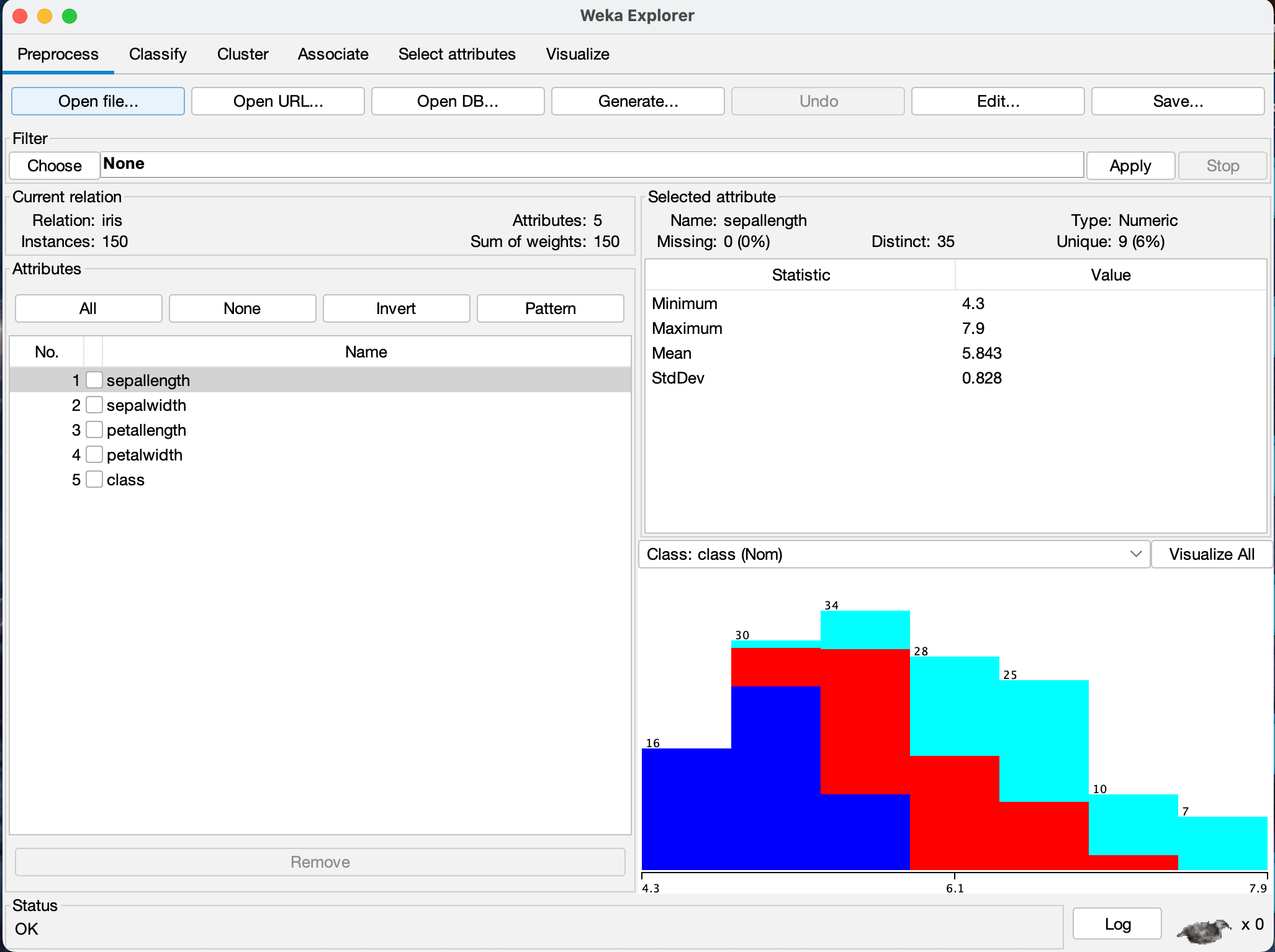
Homescreen of Weka



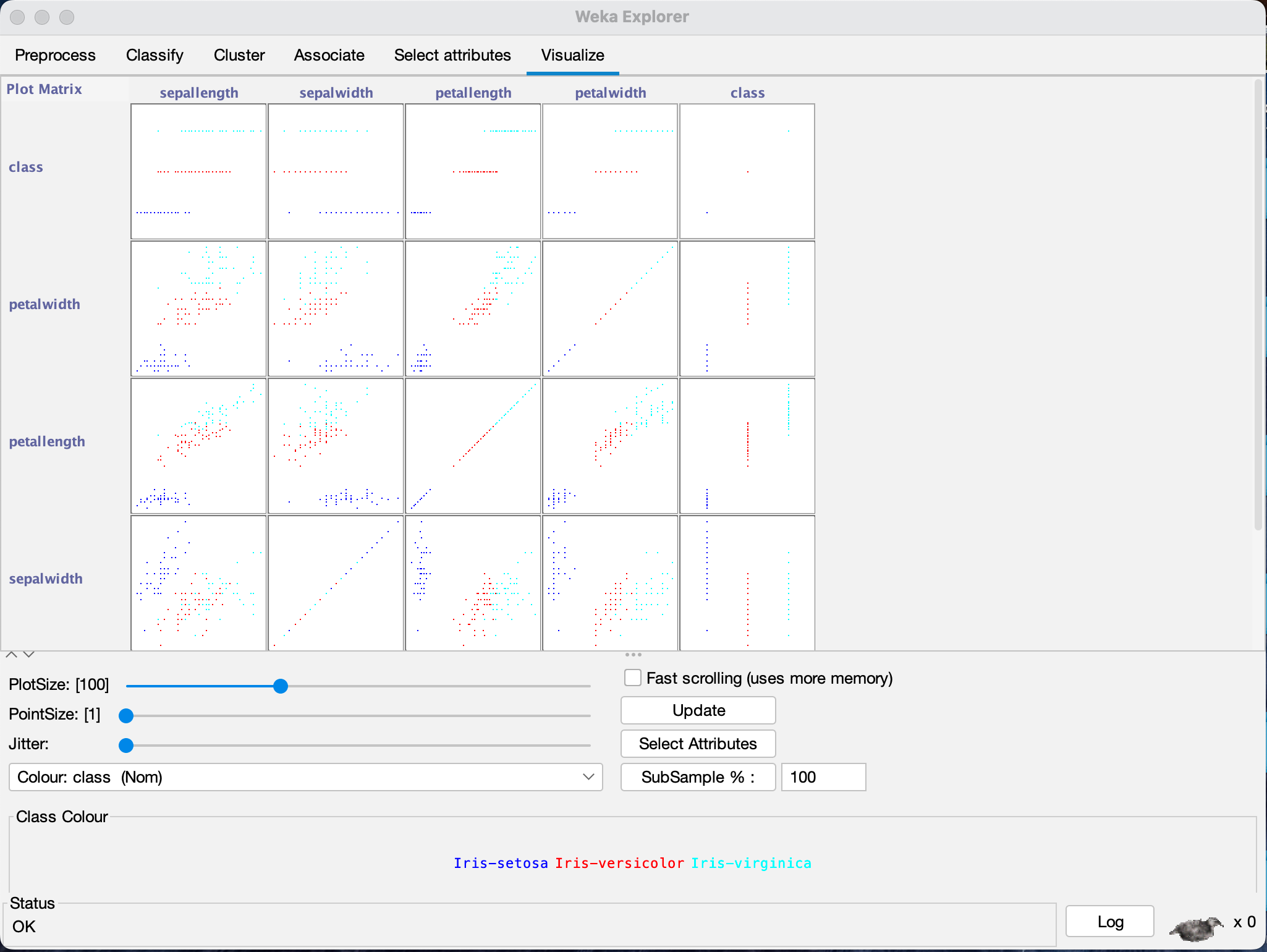
Weka Explorer



Opening Iris Dataset using URL



Preprocess Tab in Weka



Visualisation of Dataset

**Learning Outcomes:**

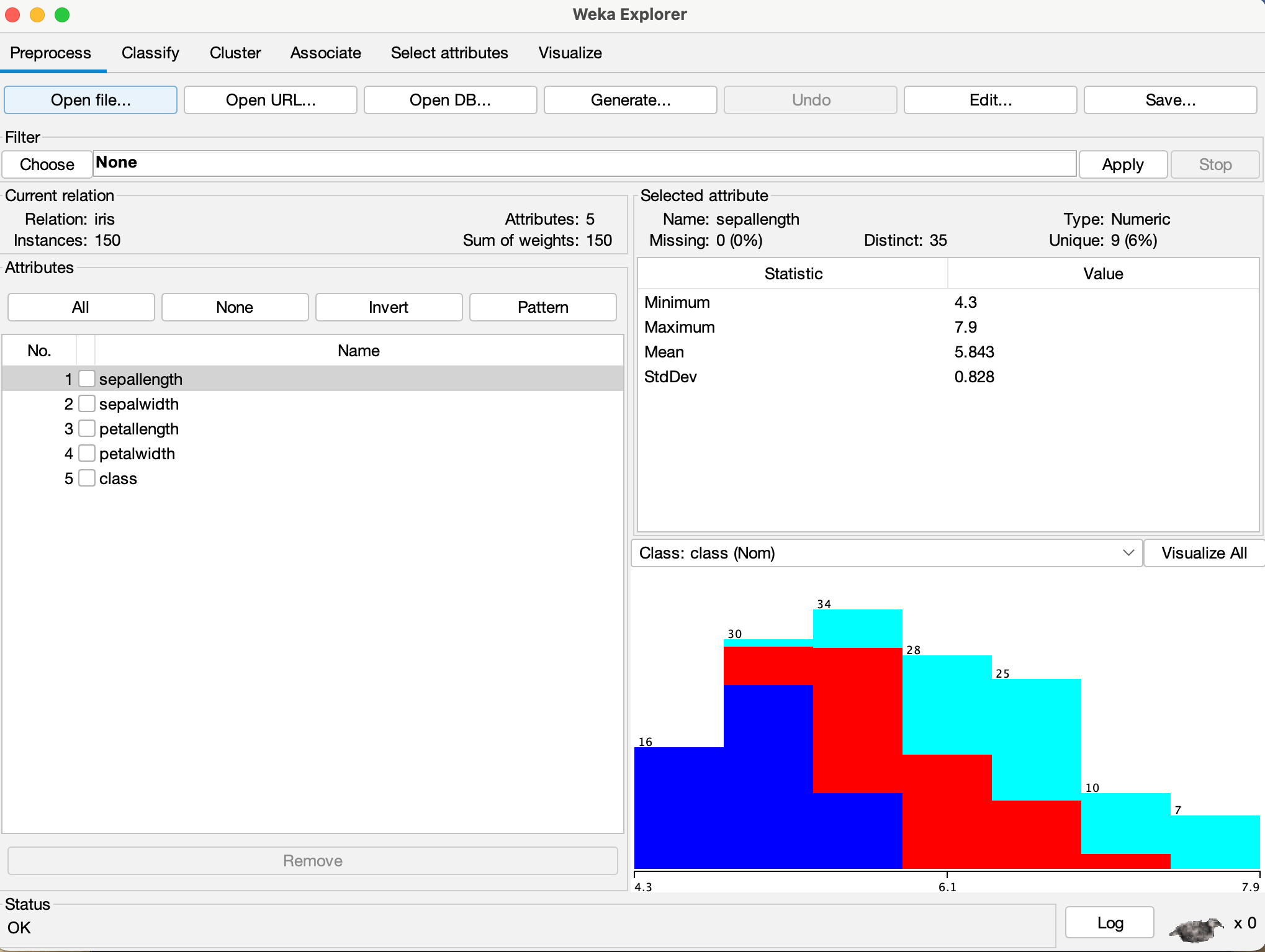
# Experiment 5

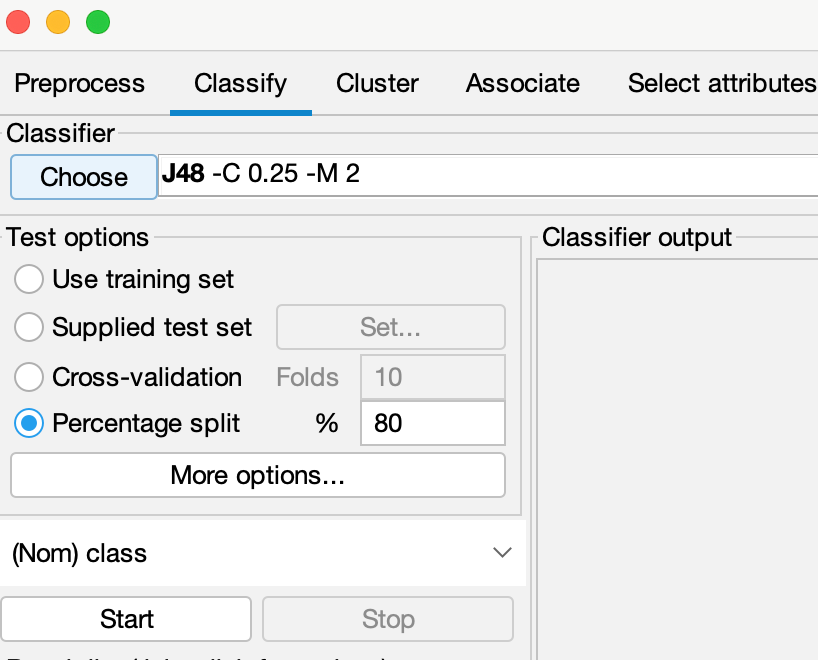
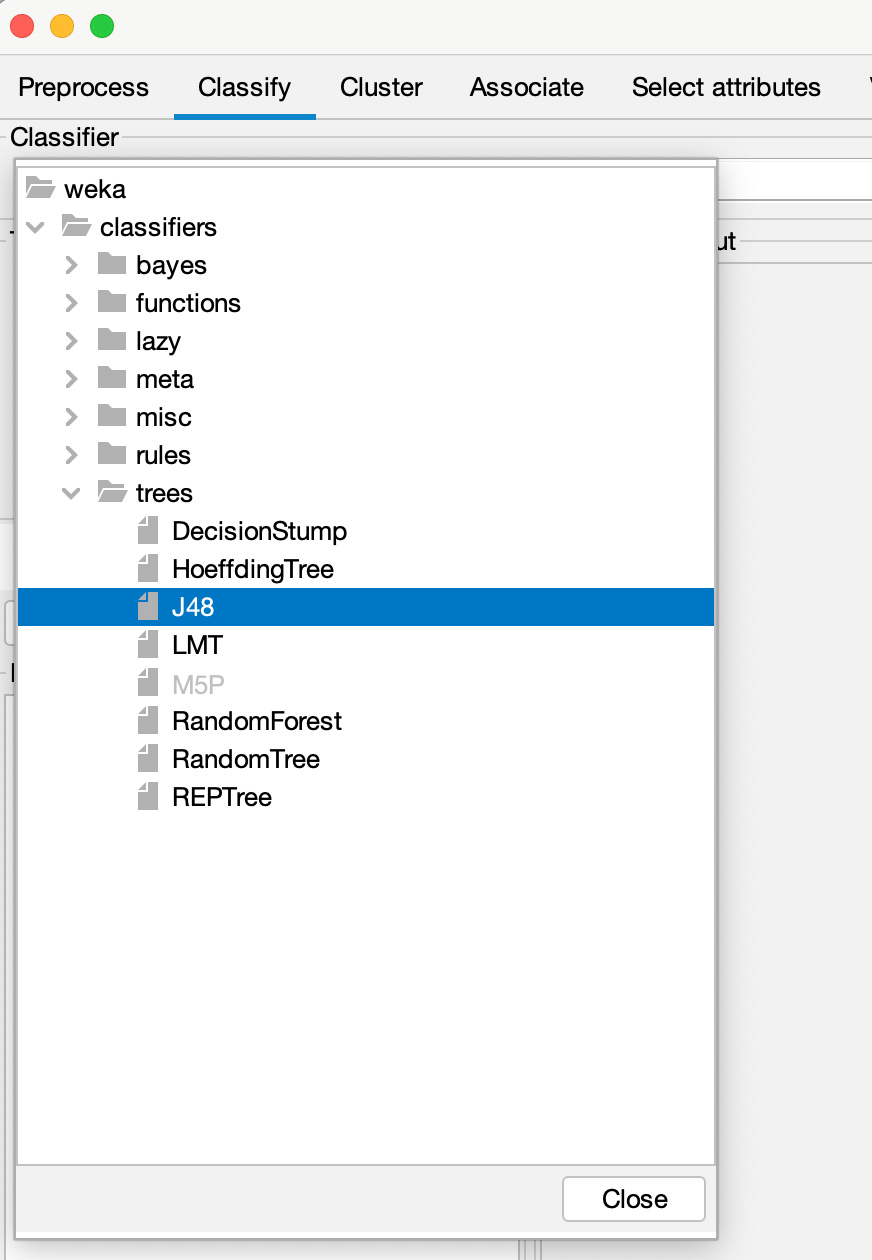
**Aim: Implementation of Classification technique on ARFF files using WEKA.**

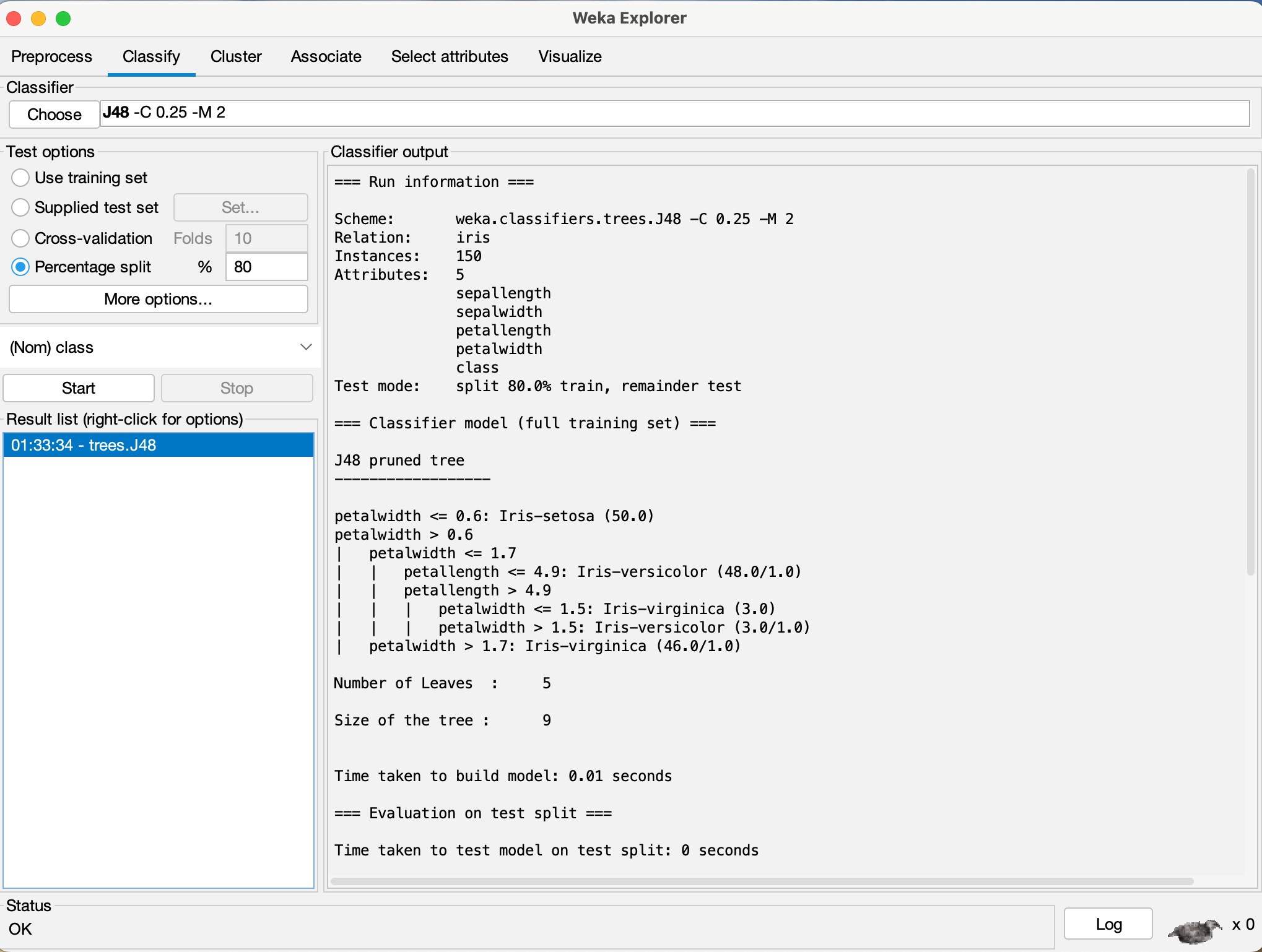
**Theory:**

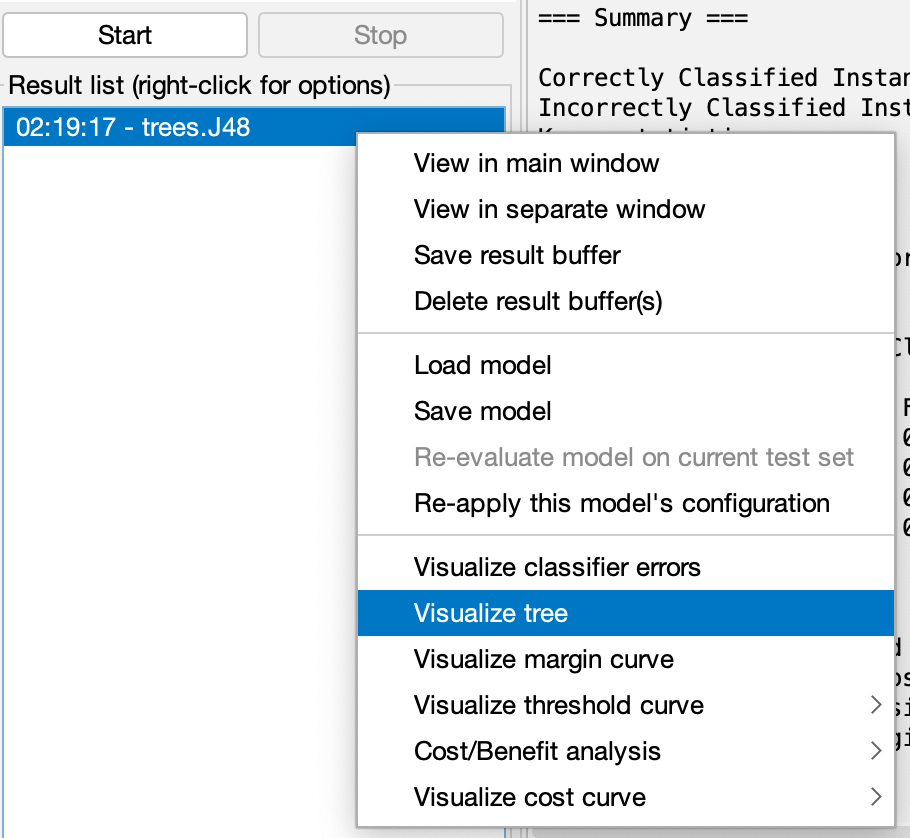
**Dataset:**

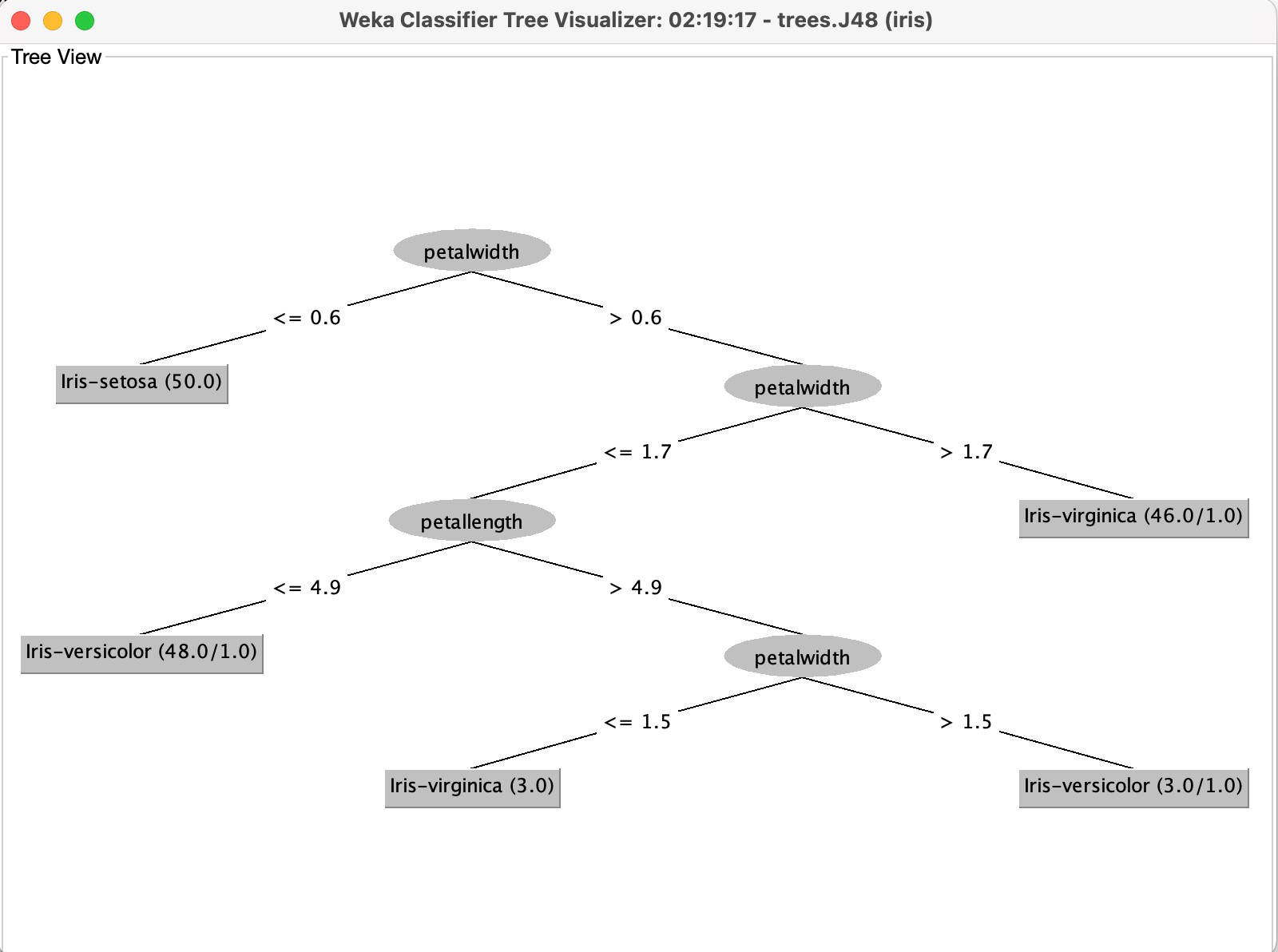
**Snapshots:**

****

****



****



**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  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 | | | | | | 0 |  | % |
| === 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

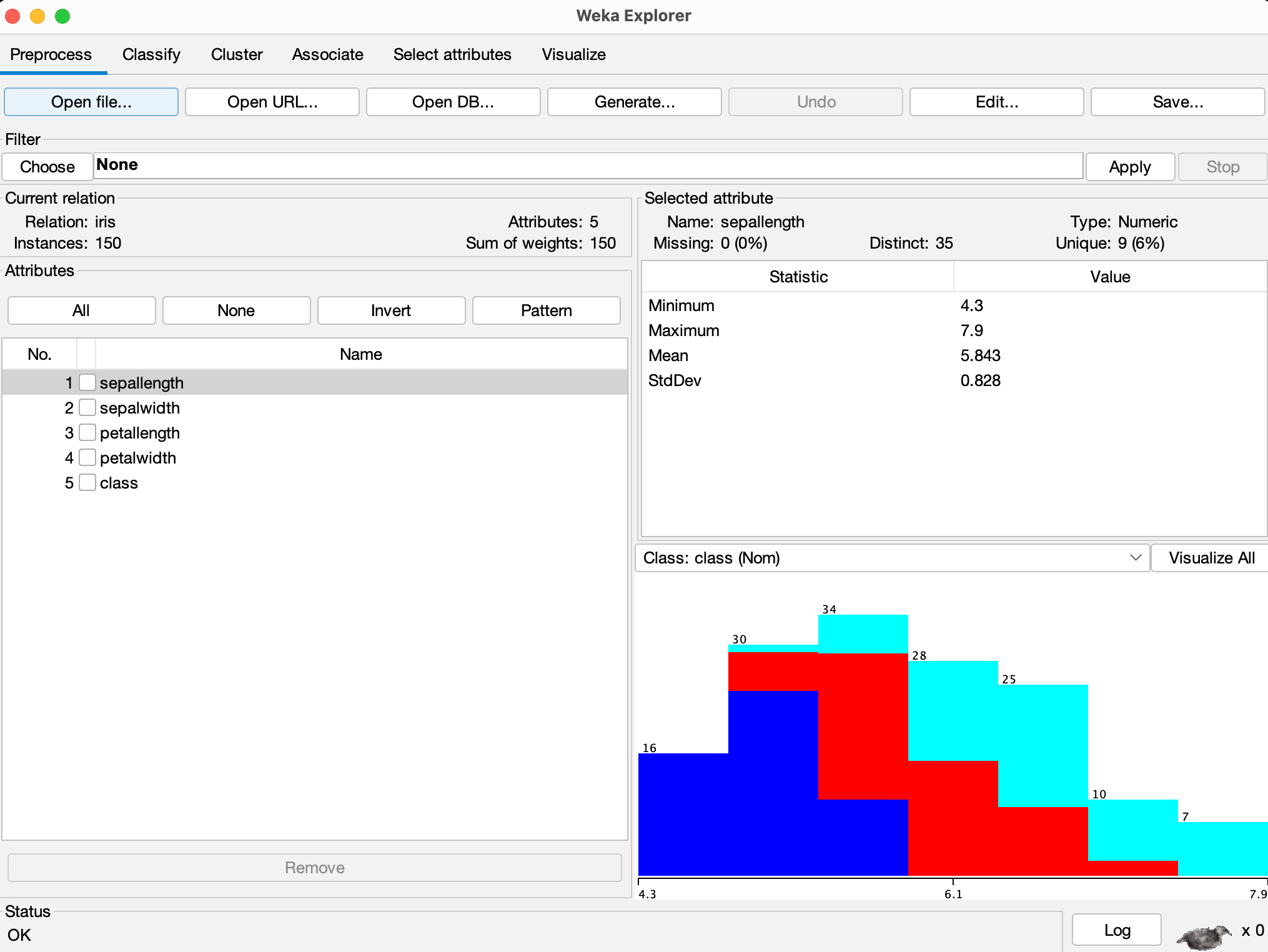
**Aim: Implementation of Clustering technique on ARFF files using WEKA.**

**Theory:**

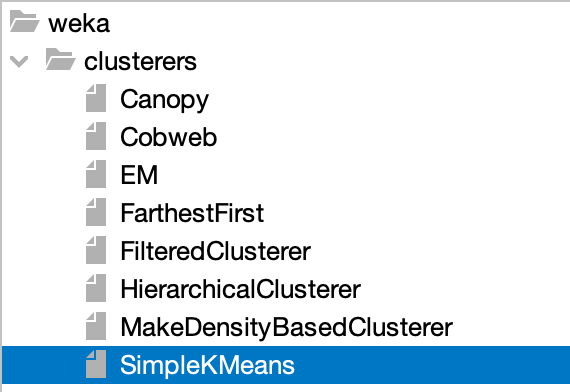
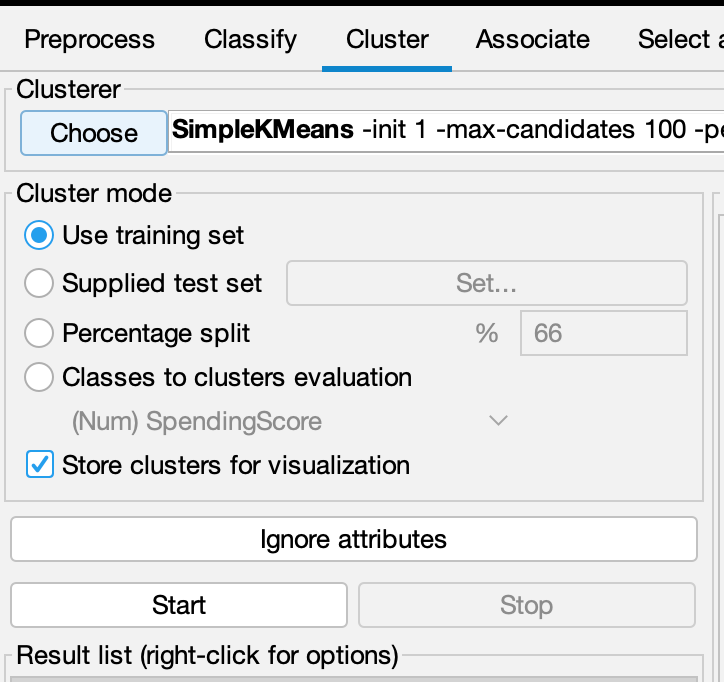
**Dataset:**

**Procedure:**

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

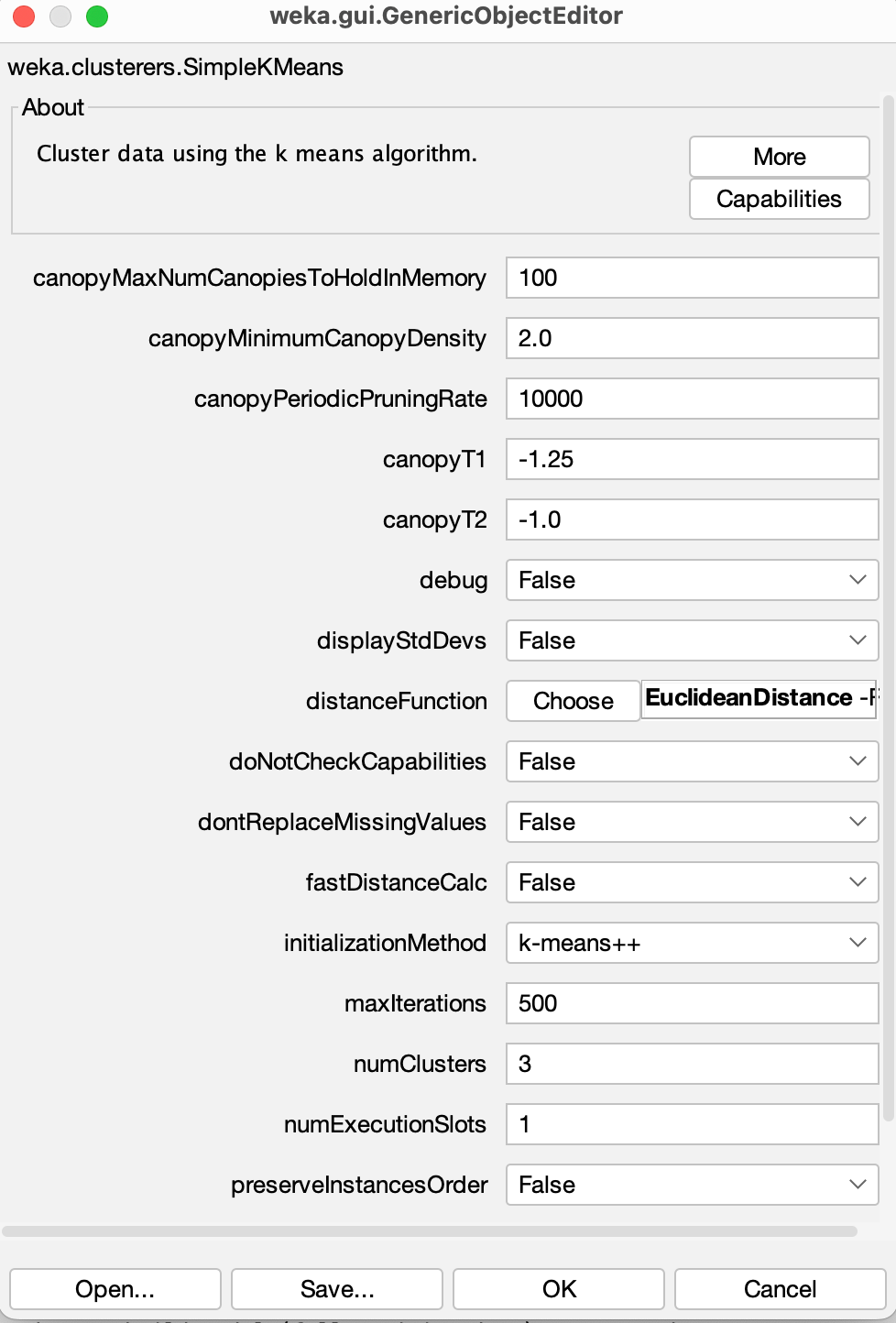


1. Go to Cluster tab -> Under Cluster Mode, keep: Use training set -> Click on Choose Pick: SimpleKMeans



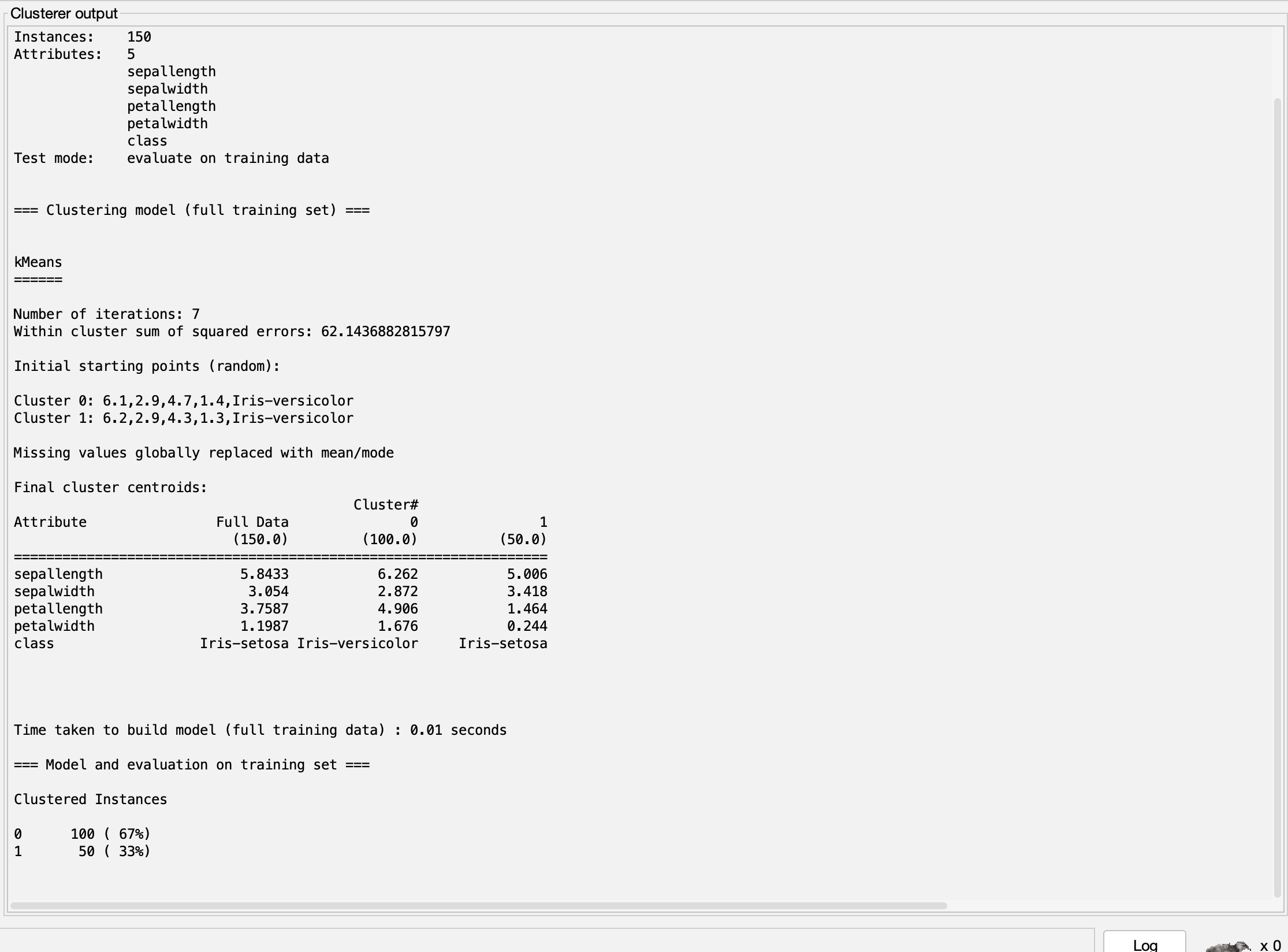
1. Set K-Means Parameters: [numClusters = 3, Distance function: EuclideanDistance

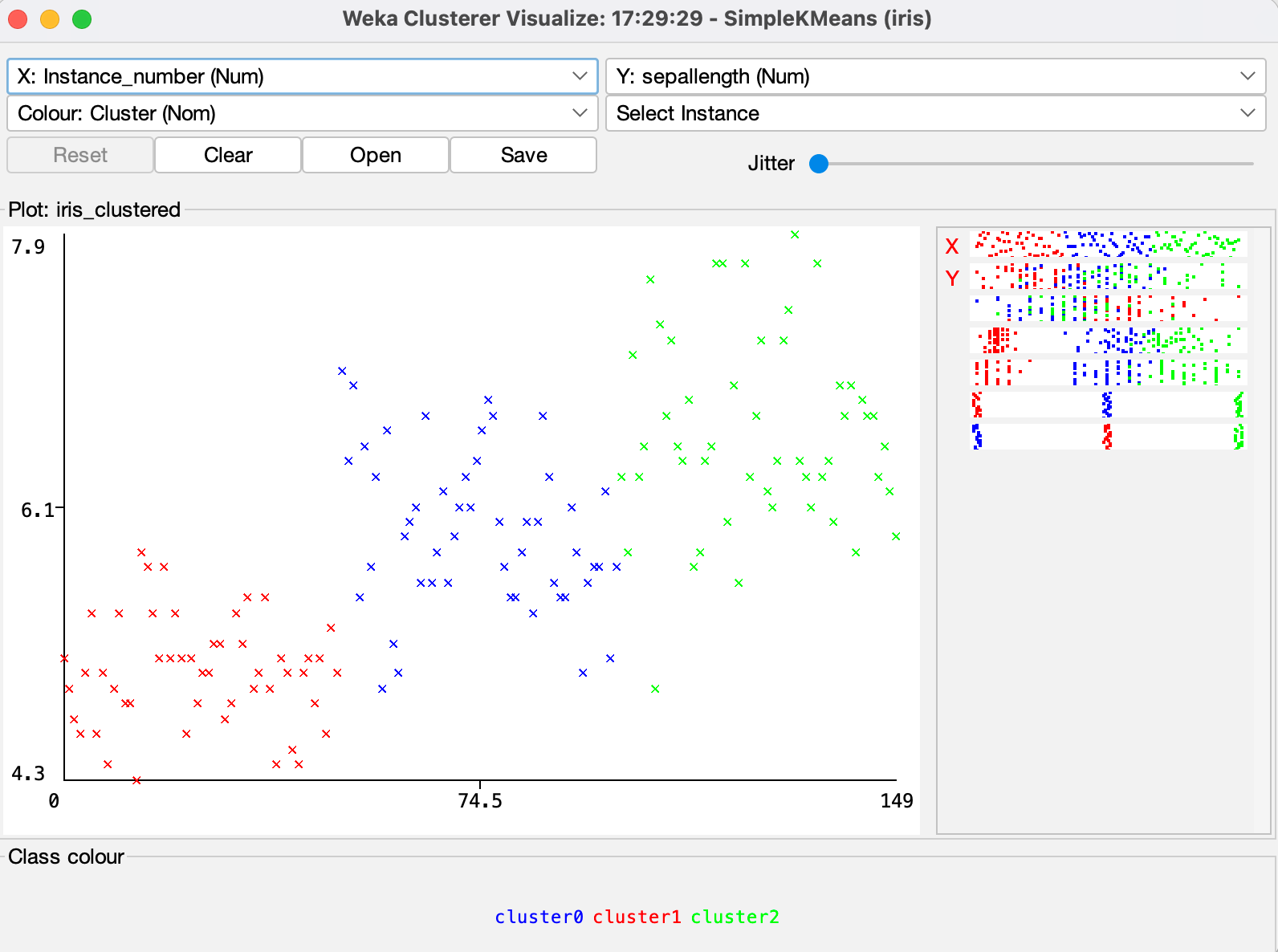
,Initialization method: k-means++] -> Click OK



5. Run the clustering

**Output:**

****



**Learning Outcomes:**

# Experiment 7

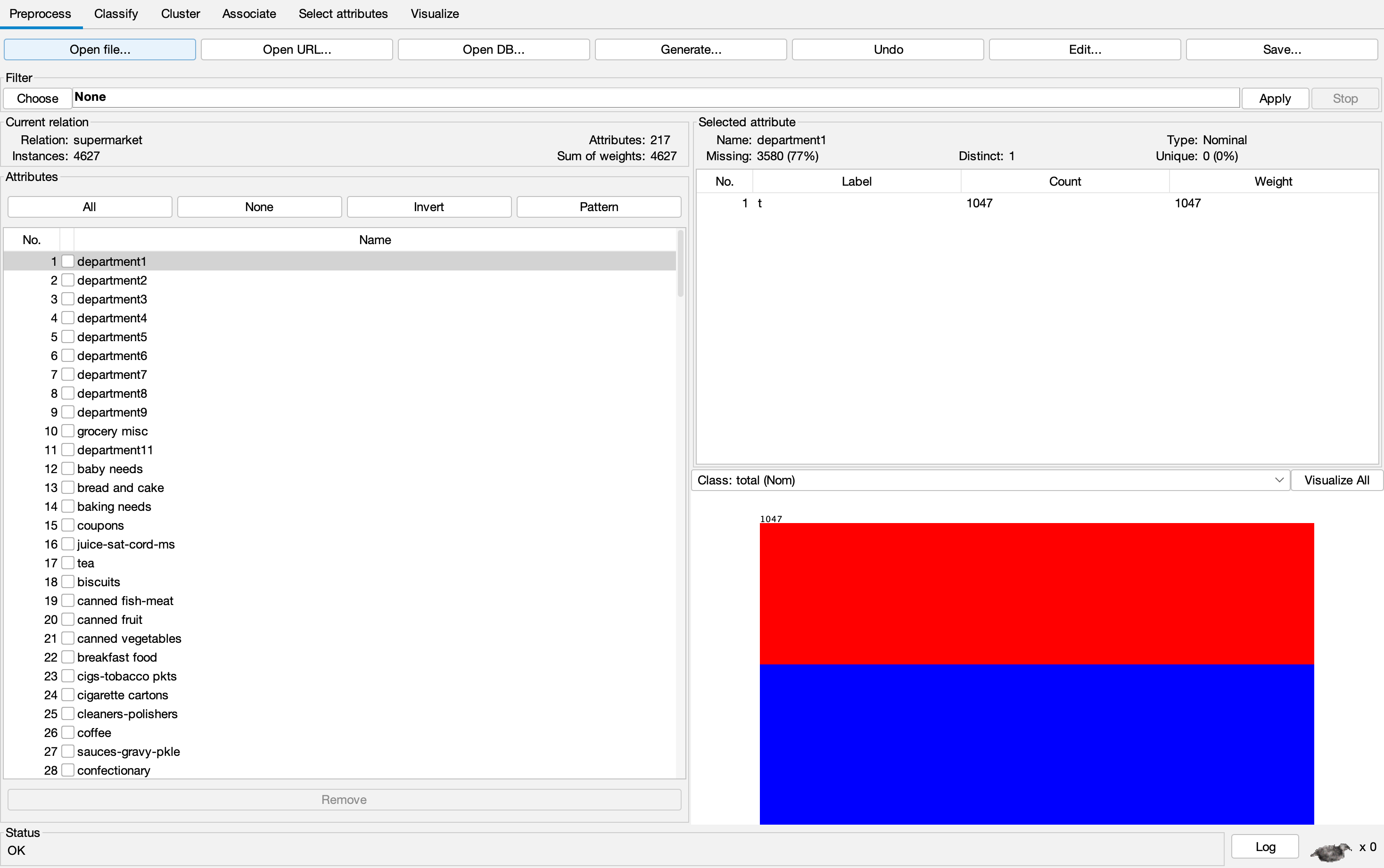
**Aim: Implementation of Association Rule technique on ARFF files using WEKA.**

**Theory:**

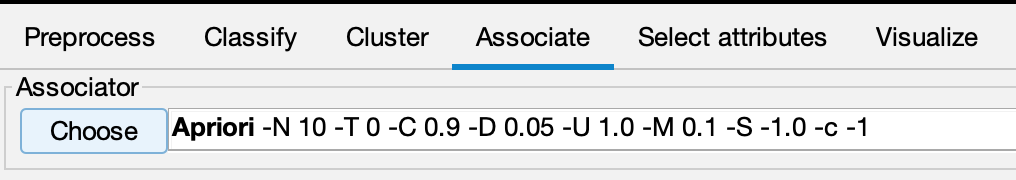
**Dataset:**

**Procedure:**

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

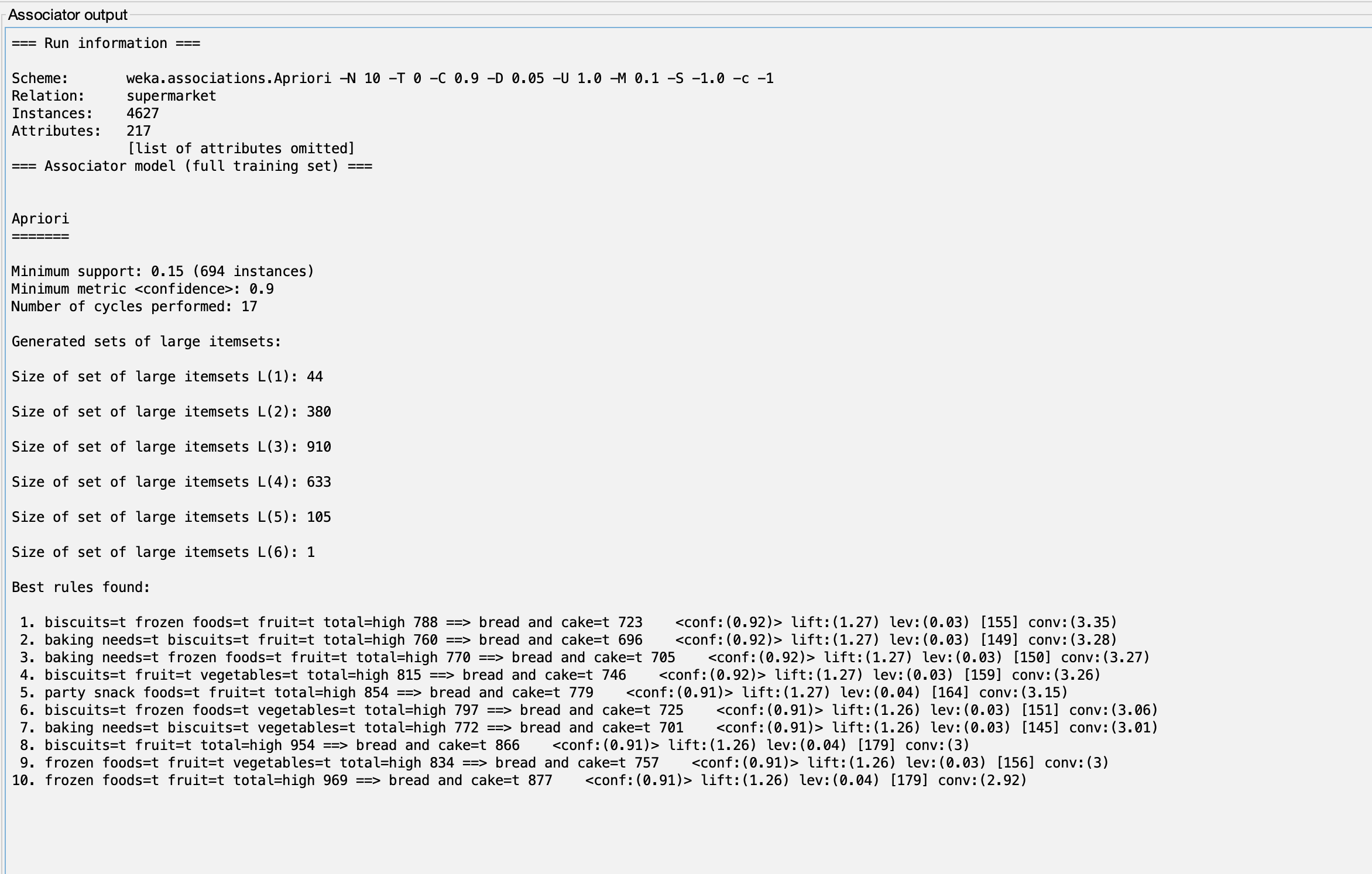


1. Go to Associate tab -> Choose Algorithm (Apriori)



1. Run by clicking on Start

**Output:**

****

**Learning Outcomes:**

# Experiment 8

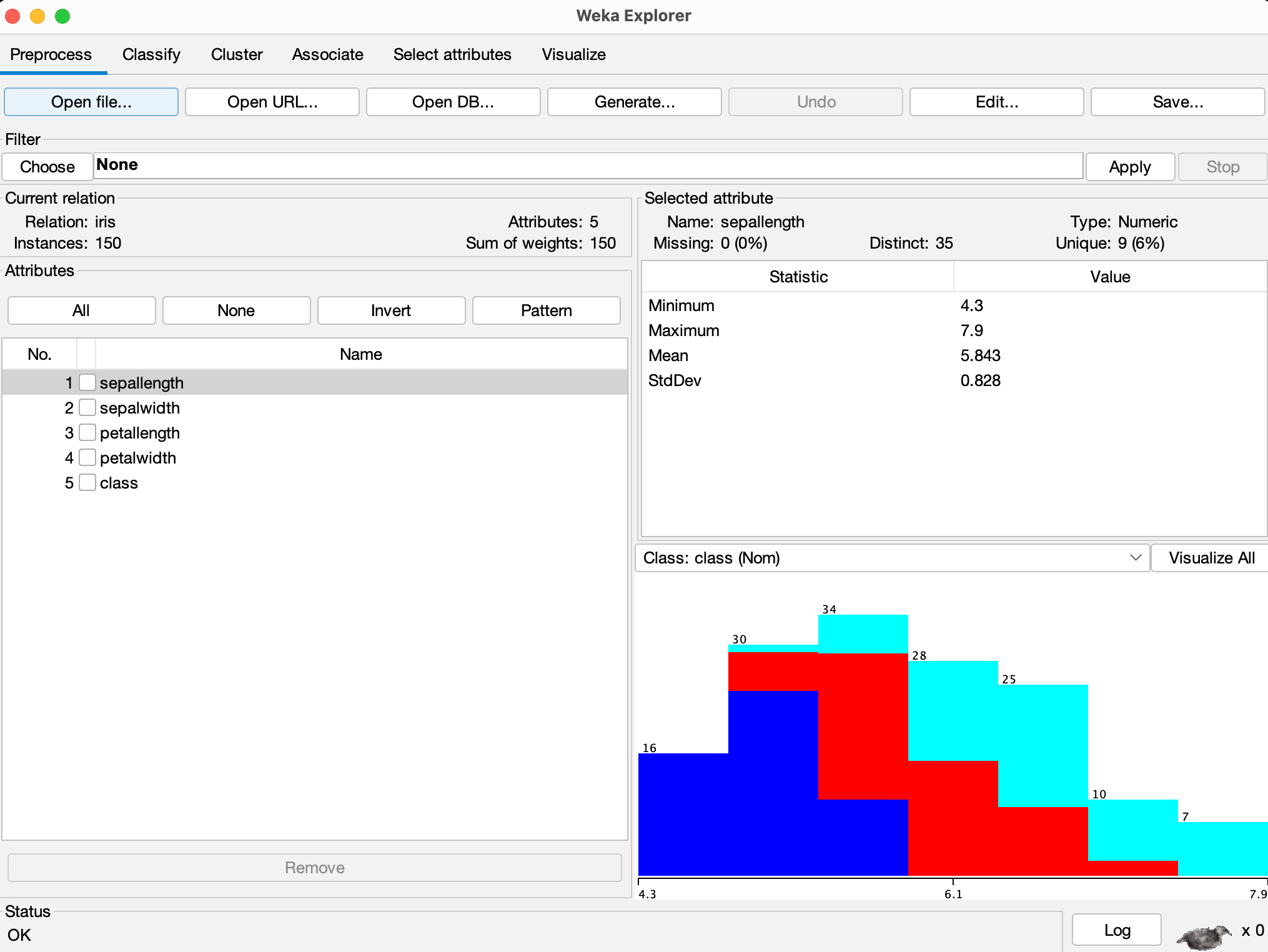
**Aim: Implementation of Visualization technique on ARFF files using WEKA.**

**Theory:**

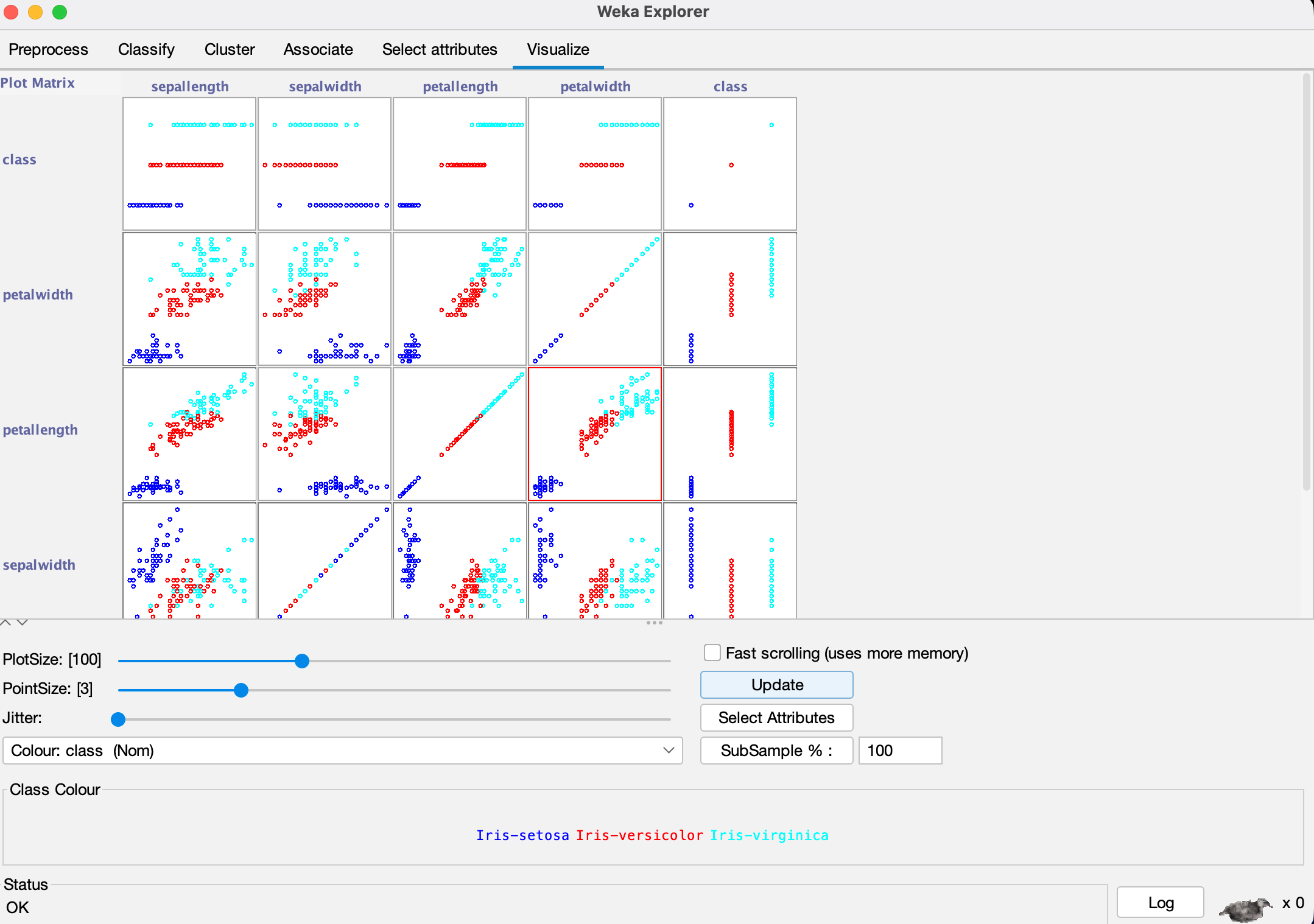
**Dataset:**

**Procedure:**

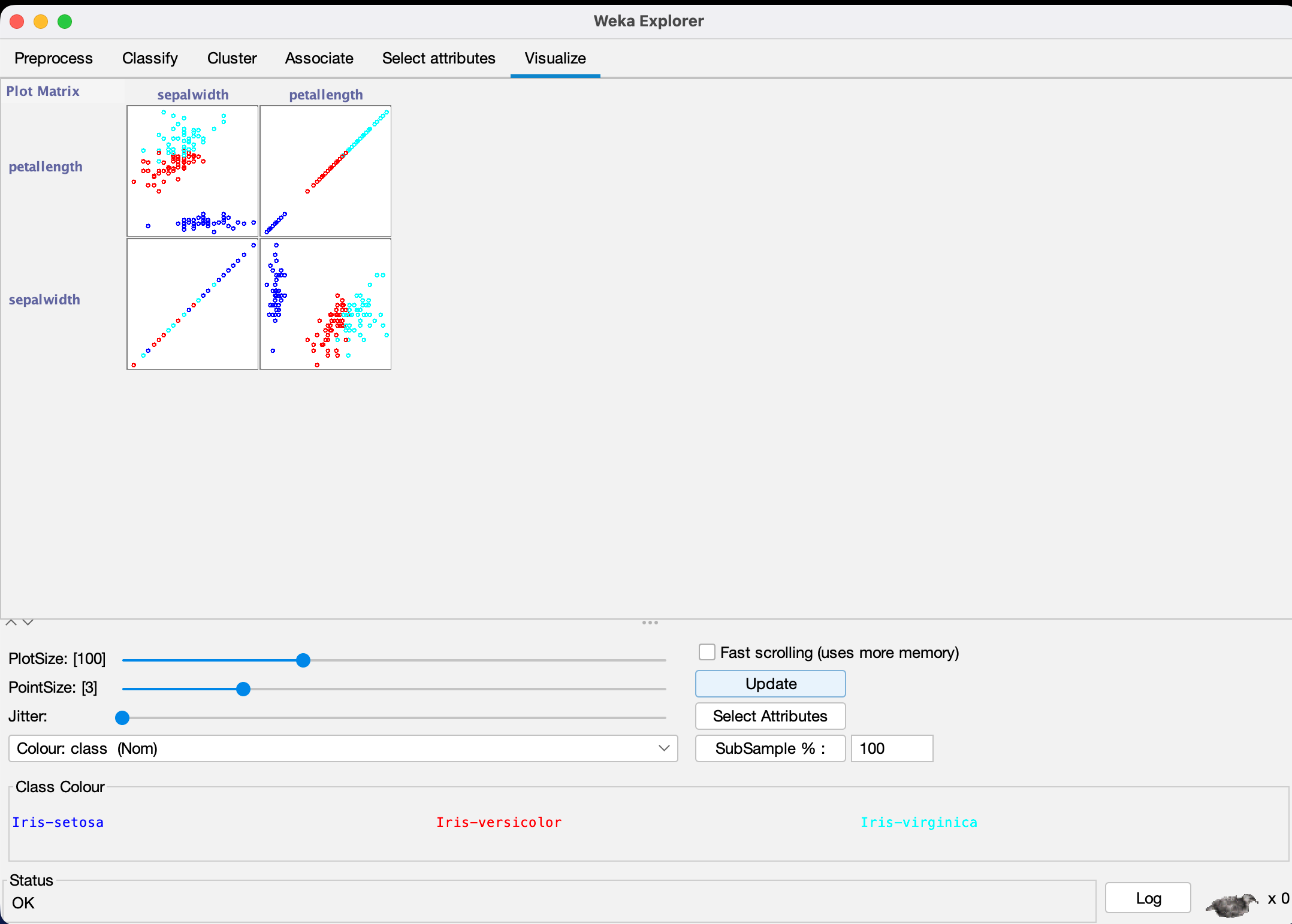
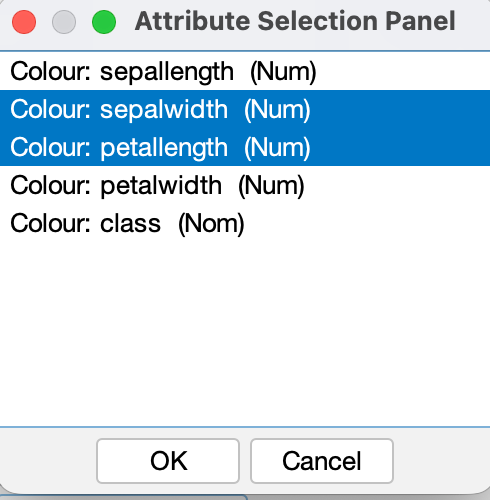
1. WEKA GUI Chooser -> Explorer -> Open .arff file



1. Go to Visualize tab.
   1. Click Visualize All



* 1. 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)

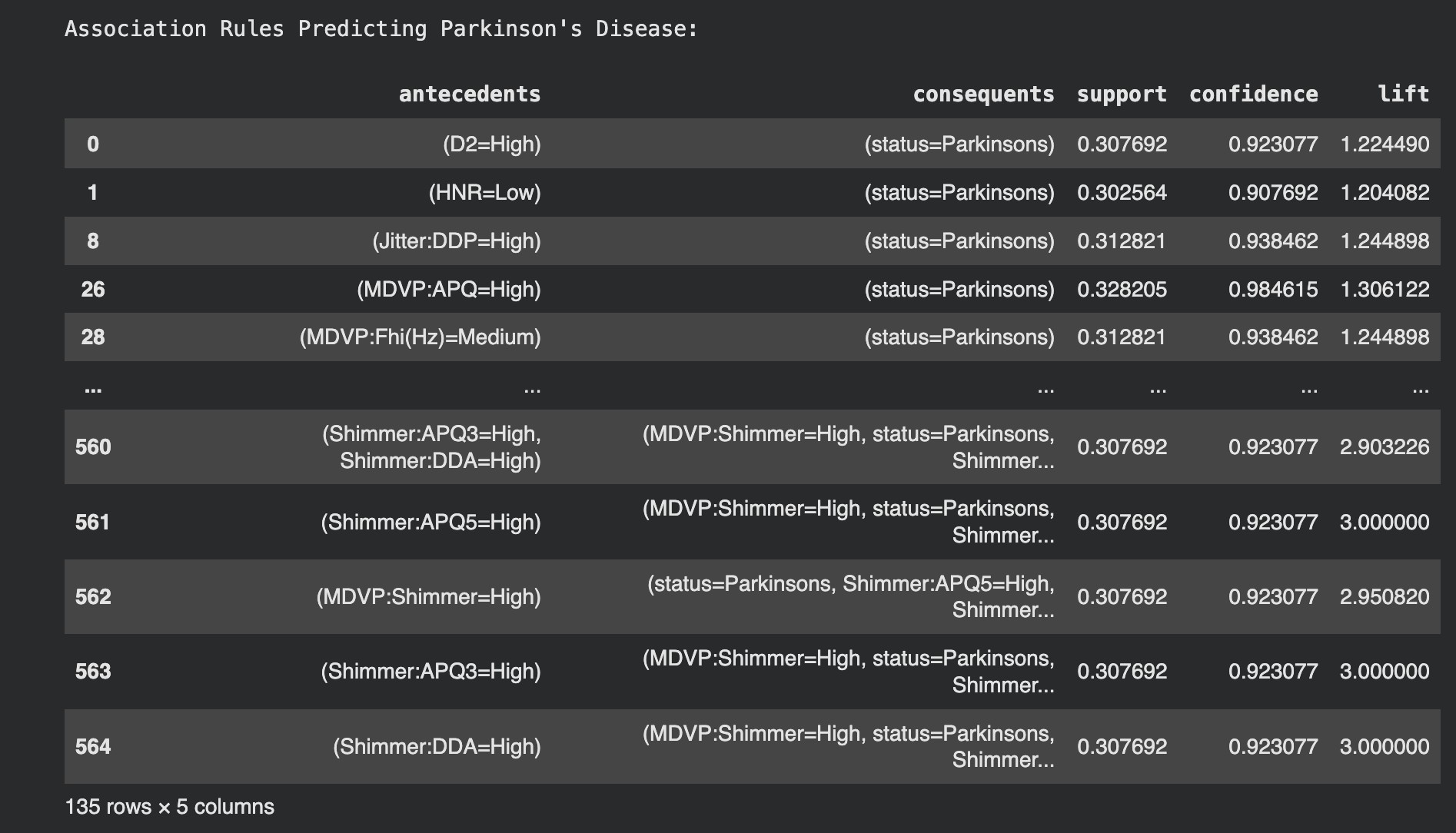
# Filter rules predicting Parkinson's disease rules\_pd =

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:**

****

**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, confusion\_matrix

silhouette\_score, confusion\_matrix) # 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]

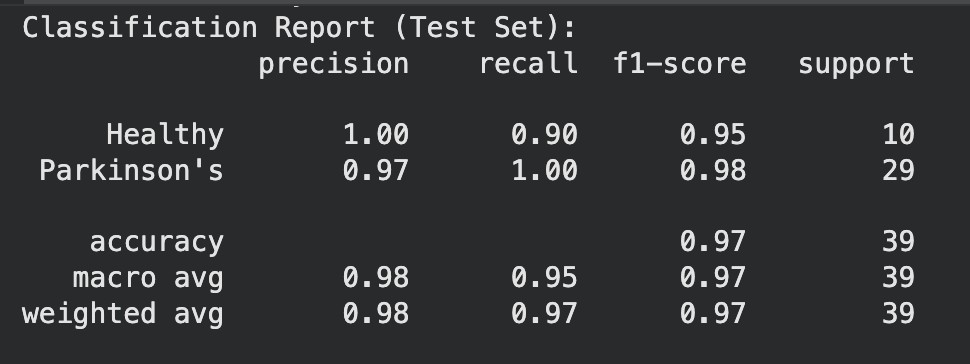
colors = ['#4CAF50', '#2196F3']

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)

s

**Output:**

****

**Learning Outcomes:**