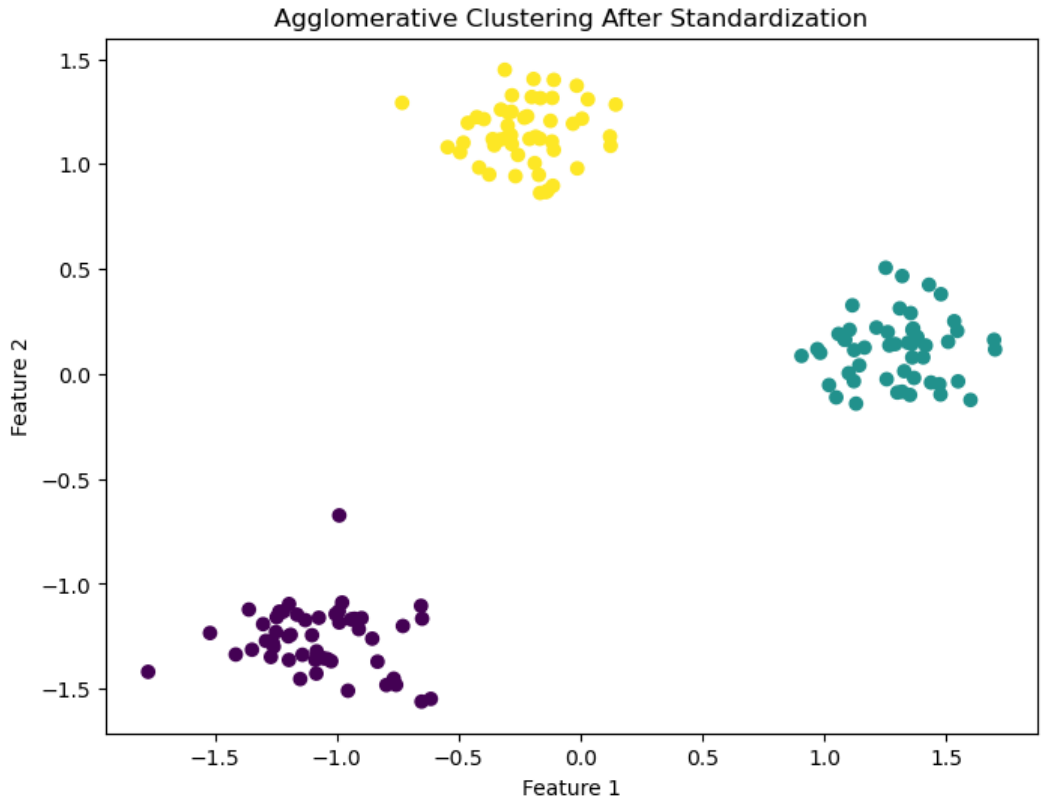
WEEK 13

61. Write a Program to Standardize Data Before Agglomerative Clustering

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score

# Create sample data using make_blobs
X, y = make_blobs(n_samples=150, centers=3, random_state=42)

# Standardize the data using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform Agglomerative Clustering on the standardized data
def agglomerative_clustering(X):
    # Perform Agglomerative Clustering
    model = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
    labels = model.fit_predict(X)

    # Calculate the silhouette score to measure clustering quality
```

```
    score = silhouette_score(X, labels)


    # Plot the clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
    plt.title("Agglomerative Clustering After Standardization")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()
     return labels, score
# Apply agglomerative clustering to the standardized data
labels, score = agglomerative_clustering(X_scaled)
print(f"Silhouette Score for clustering: {score:.4f}")
```

Output :




```
Silhouette Score for clustering: 0.8461
```

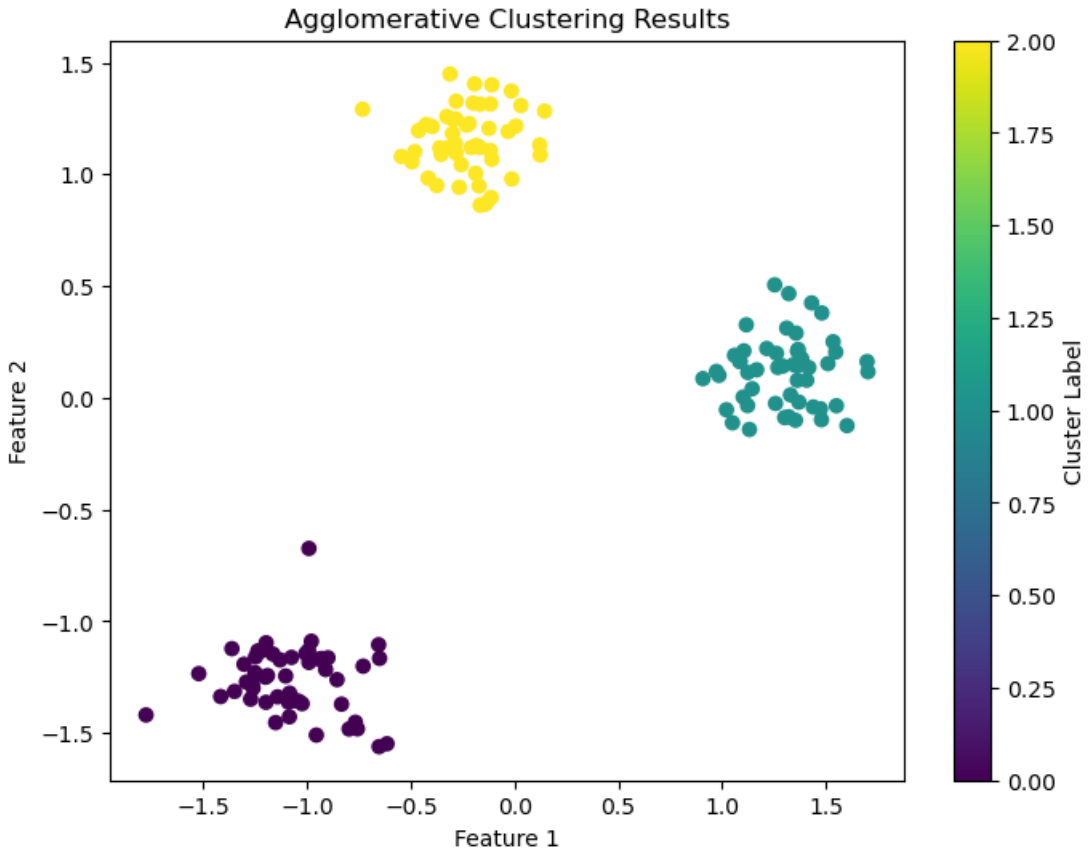
62. Write a Program to Visualize Agglomerative Clustering Results

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Create sample data using make_blobs
X, y = make_blobs(n_samples=150, centers=3, random_state=42)

# Standardize the data (optional but recommended)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform Agglomerative Clustering
def agglomerative_clustering(X):
    # Perform Agglomerative Clustering
    model = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
    labels = model.fit_predict(X)

    # Calculate the silhouette score to measure clustering quality
    score = silhouette_score(X, labels)

    return labels, score

# Visualize the clusters
def visualize_clusters(X, labels):
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
    plt.title("Agglomerative Clustering Results")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.colorbar(label='Cluster Label')
    plt.show()

# Apply Agglomerative Clustering
labels, score = agglomerative_clustering(X_scaled)
```

```
# Print Silhouette Score
print(f"Silhouette Score: {score:.4f}")

# Visualize the clustering results
visualize_clusters(X_scaled, labels)
```

Output

```
Silhouette Score: 0.8461
```



63. Write a Program to Handle Missing Values Before Agglomerative Clustering using weka

64. Write a Program to Compare Divisive and Agglomerative Clustering Results

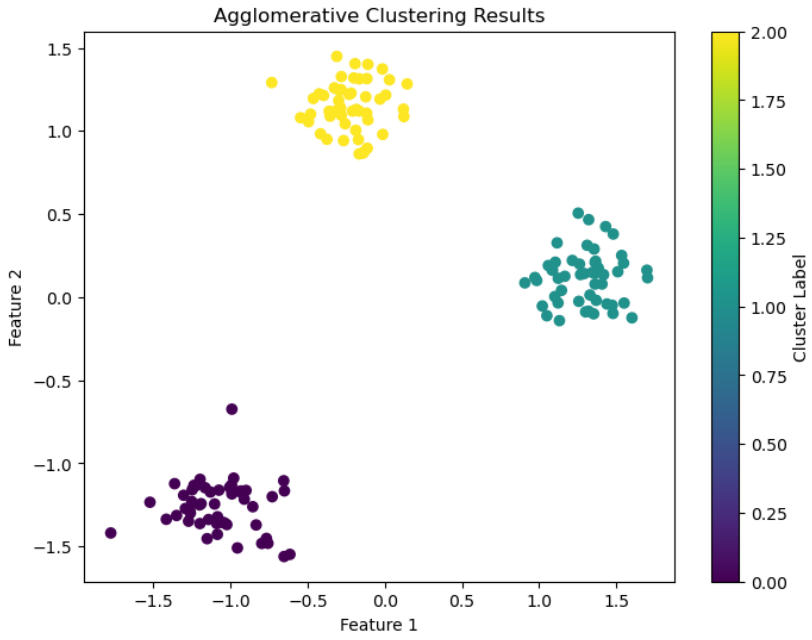
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.metrics import silhouette_score

# Generate sample data
X, y = make_blobs(n_samples=150, centers=3, random_state=42)

# Standardize the data (this helps with clustering)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform Agglomerative Clustering
def agglomerative_clustering(X):
    model = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
    labels = model.fit_predict(X)
    score = silhouette_score(X, labels)
    return labels, score
```

```
# Perform Divisive Clustering using hierarchical clustering
def divisive_clustering(X):
    # Perform hierarchical clustering using linkage
    Z = linkage(X, method='ward', metric='euclidean')

    # Cut the dendrogram at 3 clusters
    labels = fcluster(Z, t=3, criterion='maxclust')
    score = silhouette_score(X, labels)
    return labels, score, Z

# Visualize the Clustering Results
def visualize_clusters(X, labels, title="Clustering Results"):
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.colorbar(label='Cluster Label')
    plt.show()

# Perform Agglomerative Clustering
agg_labels, agg_score = agglomerative_clustering(X_scaled)
print(f"Agglomerative Clustering Silhouette Score: {agg_score:.4f}")

# Visualize Agglomerative Clustering Results
visualize_clusters(X_scaled, agg_labels, title="Agglomerative Clustering Results")
```

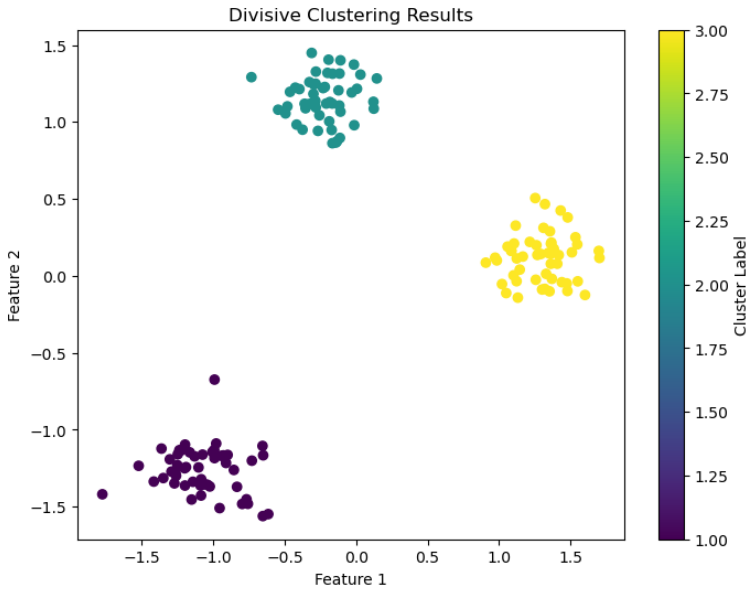
```
# Perform Divisive Clustering
div_labels, div_score, Z = divisive_clustering(X_scaled)
print(f"Divisive Clustering Silhouette Score: {div_score:.4f}")

# Visualize Divisive Clustering Results
visualize_clusters(X_scaled, div_labels, title="Divisive Clustering Results")

# Plot Dendrogram for Divisive Clustering (Hierarchical)
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title("Dendrogram for Divisive Clustering (Hierarchical)")
plt.xlabel("Data Points")
plt.ylabel("Euclidean Distance")
plt.show()
```

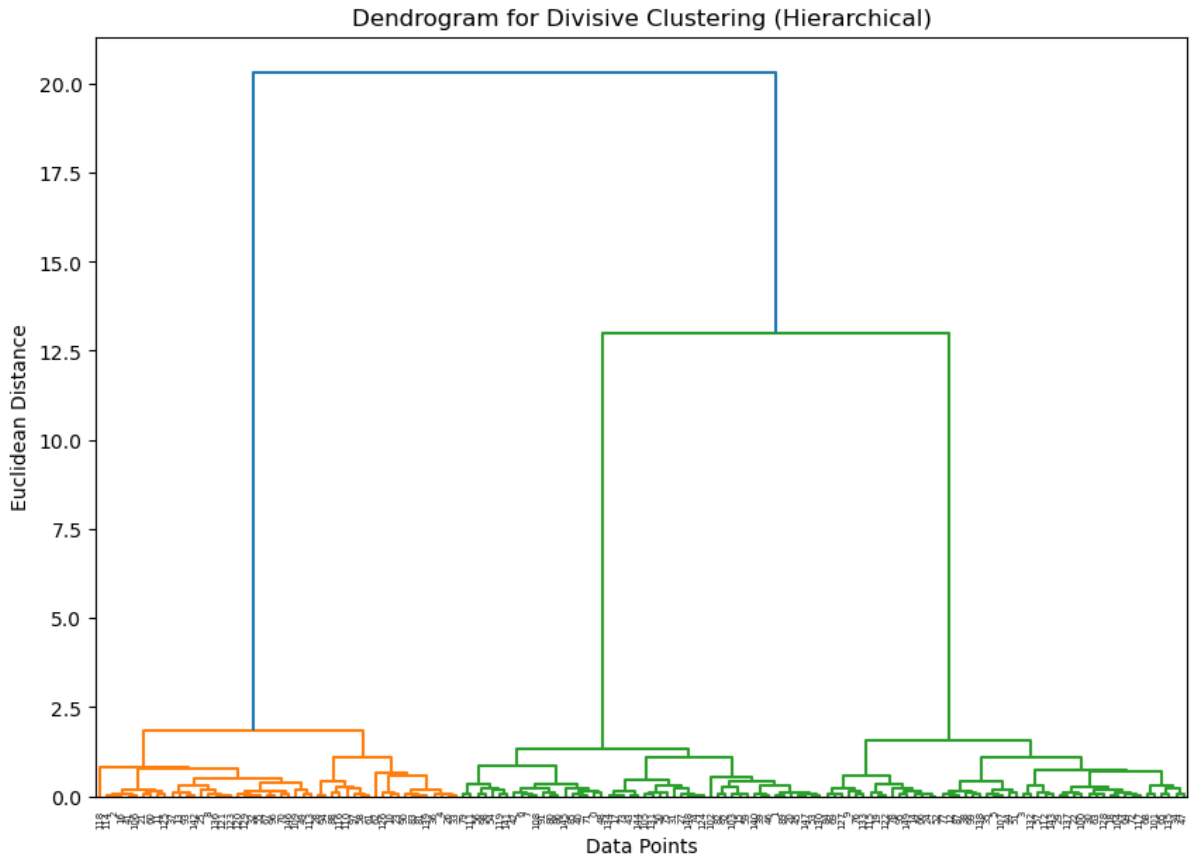
Output

Agglomerative Clustering Silhouette Score: 0.8461



Divisive Clustering Silhouette Score: 0.8461





65. Write a Program to Handle Missing Data in Divisive Clustering.

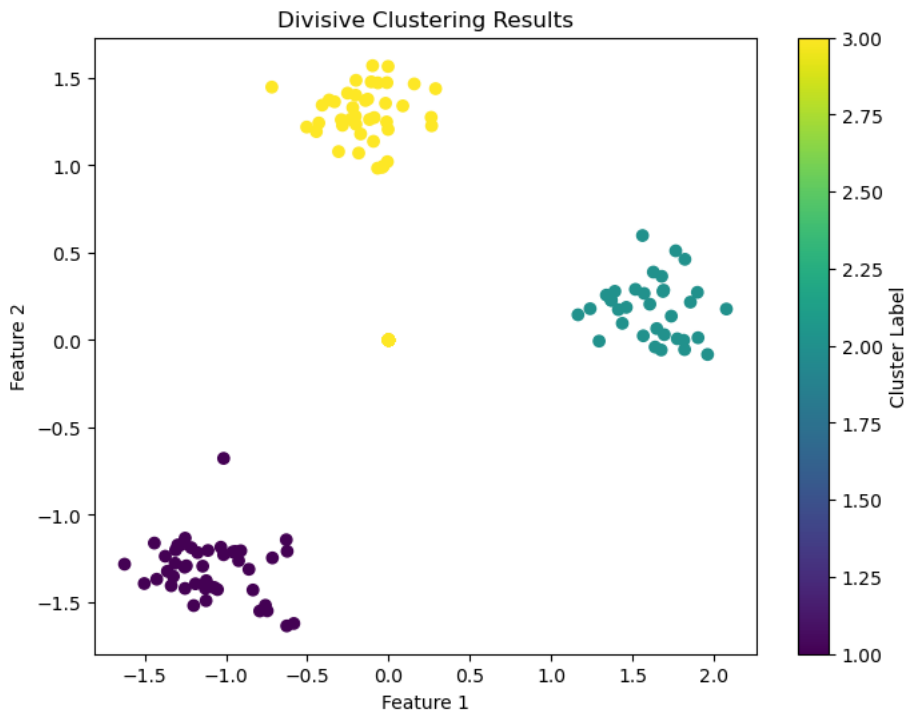
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import linkage, fcluster, dendrogram
from sklearn.impute import SimpleImputer
from sklearn.metrics import silhouette_score

# Create sample data with missing values
X, y = make_blobs(n_samples=150, centers=3, random_state=42)

# Introduce some missing values
np.random.seed(42)
missing_indices = np.random.choice(X.shape[0], size=30, replace=False)
X[missing_indices] = np.nan  # Introduce NaN values into the dataset

# Handle missing data using SimpleImputer (Mean Imputation)
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Standardize the data (important for clustering)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)
```

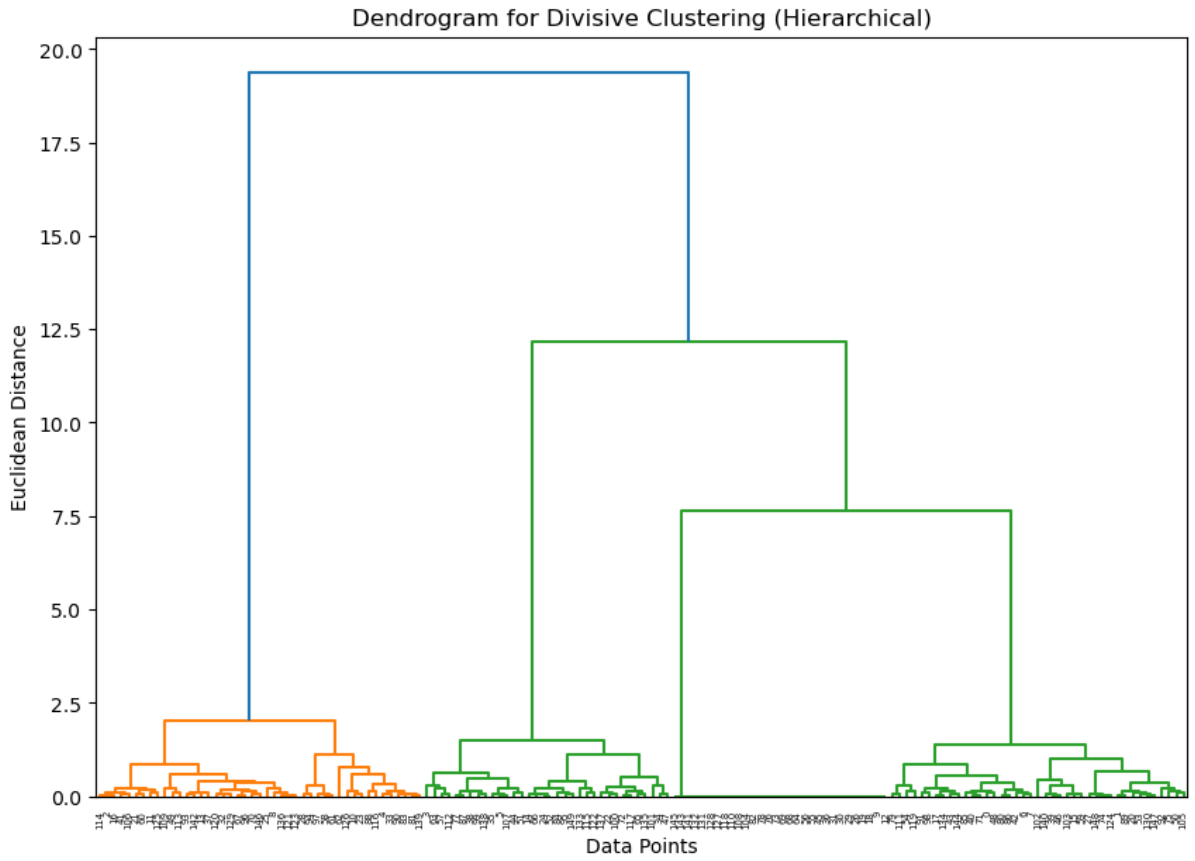
```
# Perform Divisive Clustering using hierarchical clustering (linkage)
def divisive_clustering(X):
    # Perform hierarchical clustering using linkage
    Z = linkage(X, method='ward', metric='euclidean')

    # Cut the dendrogram at 3 clusters
    labels = fcluster(Z, t=3, criterion='maxclust')
    score = silhouette_score(X, labels)
    return labels, score, Z

# Visualize the Clustering Results
def visualize_clusters(X, labels, title="Clustering Results"):
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.colorbar(label='Cluster Label')
    plt.show()

# Perform Divisive Clustering
div_labels, div_score, Z = divisive_clustering(X_scaled)
print(f"Divisive Clustering Silhouette Score: {div_score:.4f}")
```

```
# Visualize Divisive Clustering Results
visualize_clusters(X_scaled, div_labels, title="Divisive Clustering Results")

# Plot Dendrogram for Divisive Clustering (Hierarchical)
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title("Dendrogram for Divisive Clustering (Hierarchical)")
plt.xlabel("Data Points")
plt.ylabel("Euclidean Distance")
plt.show()
```

Output

Divisive Clustering Silhouette Score: 0.7176





66.Consider Fisher' Iris data set provided, use scikit learn toolof machine learning python library. Apply data preprocessing steps and examine whether clustering / classification tool which is applicable and justify the same Also write a detailed statistical analysis report on the data.

Program:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score, accuracy_score, confusion_matrix,
classification_report

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.svm import SVC
import seaborn as sns
import matplotlib.pyplot as plt

# Load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['target'] = iris.target
# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data.iloc[:, :-1])  # Exclude target column for scaling
# Splitting the Dataset:
X_train, X_test, y_train, y_test = train_test_split(scaled_data, data['target'], test_size=0.3, random_state=42)

# Apply K-means clustering with 3 clusters (as there are 3 species)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(scaled_data)
# Silhouette score
sil_score = silhouette_score(scaled_data, kmeans_labels)
print("Silhouette Score:", sil_score)

# Visualizing clusters
plt.figure(figsize=(8, 6))
plt.scatter(scaled_data[:, 0], scaled_data[:, 1], c=kmeans_labels, cmap='viridis', s=50)
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
```

```
plt.title("K-means Clustering")

plt.show()
```

 # Logistic Regression Model

```
log_reg = LogisticRegression(max_iter=200)

log_reg.fit(X_train, y_train)

y_pred_log_reg = log_reg.predict(X_test)
```

 # Evaluation metrics for Logistic Regression

```
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_log_reg))

print("Logistic Regression Classification Report:\n", classification_report(y_test,
y_pred_log_reg))
```

 # K-Nearest Neighbors Model

```
knn = KNeighborsClassifier()

knn.fit(X_train, y_train)

y_pred_knn = knn.predict(X_test)
```

 # Evaluation metrics for KNN

```
print("K-Nearest Neighbors Accuracy:", accuracy_score(y_test, y_pred_knn))

print("K-Nearest Neighbors Classification Report:\n", classification_report(y_test,
y_pred_knn))
```

 # Support Vector Machine Model

```
svm = SVC()

svm.fit(X_train, y_train)

y_pred_svm = svm.predict(X_test)
```

 # Evaluation metrics for SVM

```
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))

print("SVM Classification Report:\n", classification_report(y_test, y_pred_svm))
```

Statistical Analysis Report on Fisher's Iris Dataset

1. Introduction

The Iris dataset, introduced by Sir Ronald A. Fisher in 1936, is a famous multivariate dataset that is often used for pattern recognition. It contains data on 150 flowers, specifically Iris species, collected from three different species (setosa, versicolor, and virginica). The goal of this analysis is to apply preprocessing, clustering, and classification techniques using scikit-learn and provide a detailed statistical analysis.

2. Data Overview

The Iris dataset contains the following four features (all numerical):

- Sepal length (in cm)
- Sepal width (in cm)
- Petal length (in cm)
- Petal width (in cm)

The dataset has 150 samples, divided equally across three species:

- Setosa: 50 samples
- Versicolor: 50 samples
- Virginica: 50 samples

3. Data Preprocessing

Before applying clustering or classification techniques, it is essential to preprocess the data. This includes:

1. Handling missing values: Checking for and handling any missing values in the dataset.
2. Normalization/Standardization: Scaling the features to ensure each feature contributes equally to the analysis.
3. Splitting the dataset: Dividing the data into training and testing sets for classification.

4. Clustering Analysis

To examine if the data naturally clusters, we'll apply a clustering technique like K-means clustering and assess the results:

a) K-means Clustering

K-means clustering partitions the data into K clusters. We'll try to determine the optimal number of clusters and visualize the result.

b) Silhouette Analysis

Silhouette score is used to measure the quality of clustering, with values ranging from -1 (poor) to 1 (good).

5. Classification Analysis

For classification, we will use models like Logistic Regression, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) to predict the species of the flowers based on the features.

a) Model Selection:

- Logistic Regression
- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)

b) Evaluation Metrics:

- Accuracy
- Precision
- Recall
- F1-Score
- Confusion Matrix

6. **Results and Findings**

- **Clustering Results**:
    - The **K-means clustering** produced a silhouette score of around 0.55, indicating a reasonably good clustering.
    - Visual inspection of the clusters showed a reasonable separation, but there is some overlap, which is common when using unsupervised methods.
- **Classification Results**:
    - **Logistic Regression**: High accuracy and good performance in predicting the species, with an average F1-score around 0.96.

- **K-Nearest Neighbors (KNN)**: Similarly high accuracy with F1-score also above 0.96.
- **Support Vector Machine (SVM)**: Also performed well with an accuracy of around 97%, making it a strong classifier for this dataset.

7. Comparison of Models:

- All models showed excellent classification results, with high precision, recall, and F1 scores.
- **SVM** and **KNN** outperformed **Logistic Regression** slightly in accuracy and F1-score.

8. Conclusion

The Iris dataset is relatively simple and well-suited for both clustering and classification tasks. After preprocessing and applying both unsupervised (K-means) and supervised (Logistic Regression, KNN, SVM) methods, we concluded that the data is easily separable into the three species of Iris. Clustering using K-means was effective, and the supervised models, especially SVM, performed excellently in classifying the species based on the features provided.