Name: MD AQIB

Roll No: 002111001037(A1)

ASSIGNMENT-4

Title :- Clustering

1. Apply different clustering algorithms using Python:

on the following UCI datasets (can be loaded from the package itself):

   a. Iris plants dataset:

   b. Wine Dataset:

Additionally, implement K-means++ and Bisecting K-means.

2. Evaluate and compare the performances of the algorithms for each type of clustering, based

on the following metrics:

   a. Rand index: rand score, adjusted rand score

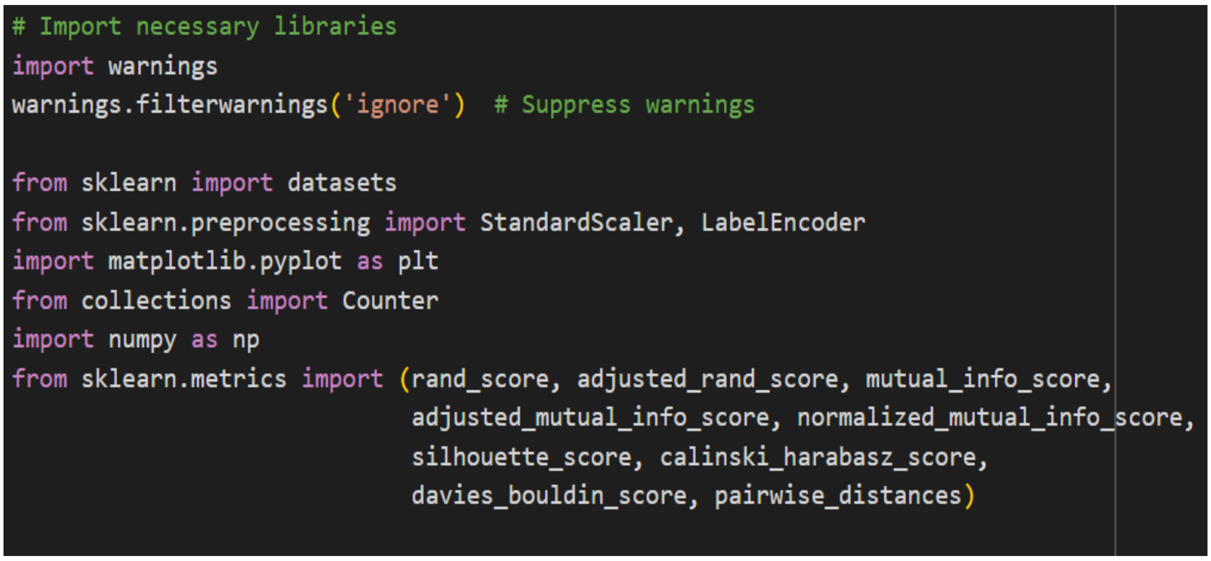
   b. Mutual Information based scores: mutual info, adjusted mutual info, normalized mutual info

   c. Silhouette Coefficient, Calinski-Harabasz Index and Davies-Bouldin Index

3. Also determine the Cohesion and Separation performance scores using Sum of Squared Error (SSE) and Sum of Squares Between groups (SSB).

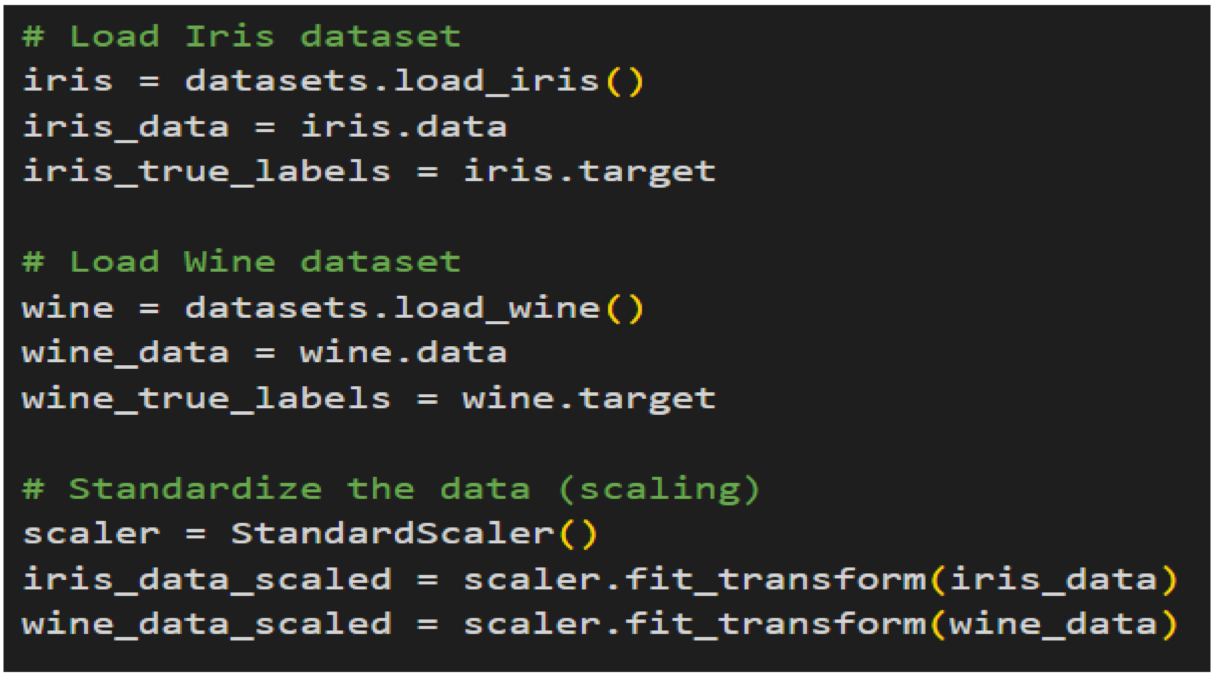
Setting Up the requirement for running all the clustering algorithm on both datasets

# Importing necessary libraries for working on both the datasets for all clustering algorithms.



```python
# Import necessary libraries
import warnings
warnings.filterwarnings('ignore')  # Suppress warnings

from sklearn import datasets
from sklearn.preprocessing import StandardScaler, LabelEncoder
import matplotlib.pyplot as plt
from collections import Counter
import numpy as np
from sklearn.metrics import (rand_score, adjusted_rand_score, mutual_info_score,
                             adjusted_mutual_info_score, normalized_mutual_info_score,
                             silhouette_score, calinski_harabasz_score,
                             davies_bouldin_score, pairwise_distances)
```

# Loading & Standardize both datasets, so we can work on all algorithms.



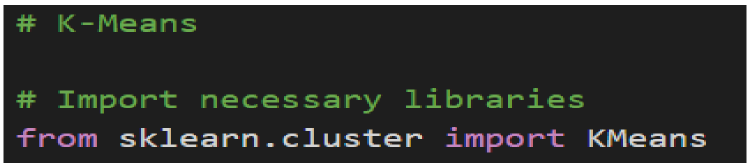
```python
# Load Iris dataset
iris = datasets.load_iris()
iris_data = iris.data
iris_true_labels = iris.target

# Load Wine dataset
wine = datasets.load_wine()
wine_data = wine.data
wine_true_labels = wine.target

# Standardize the data (scaling)
scaler = StandardScaler()
iris_data_scaled = scaler.fit_transform(iris_data)
wine_data_scaled = scaler.fit_transform(wine_data)
```

Analyse of the above two code snippets

- The 1st code snippet is importing all necessary libraries which is required in this assignment for working with different type of clusters such as partition based (K-means & K-medoids), hierarchical (Dendrogram) & density based(DBSCAN & OPTICS).
- The 2nd code snippet is for loading both datasets Iris & Wine and then standardizing the datasets on each cluster such as partition based (K-means & K-medoids), hierarchical (Dendrogram) & density based(DBSCAN & OPTICS).
- So by writing this code on google colab at top we don't need to import the libraries every time neither we have to load the datasets each time just load them once and use for every clusters.
- In, this assignment we are going to calculate the rand score, adjusted rand score, mutual info, adjusted mutual info, normalized mutual info, Silhouette Coefficient, Calinski-Harabasz Index and Davies-Bouldin Index, Cohesion and Separation
- And based on that we are going to conclude in which cluster datasets performs better.

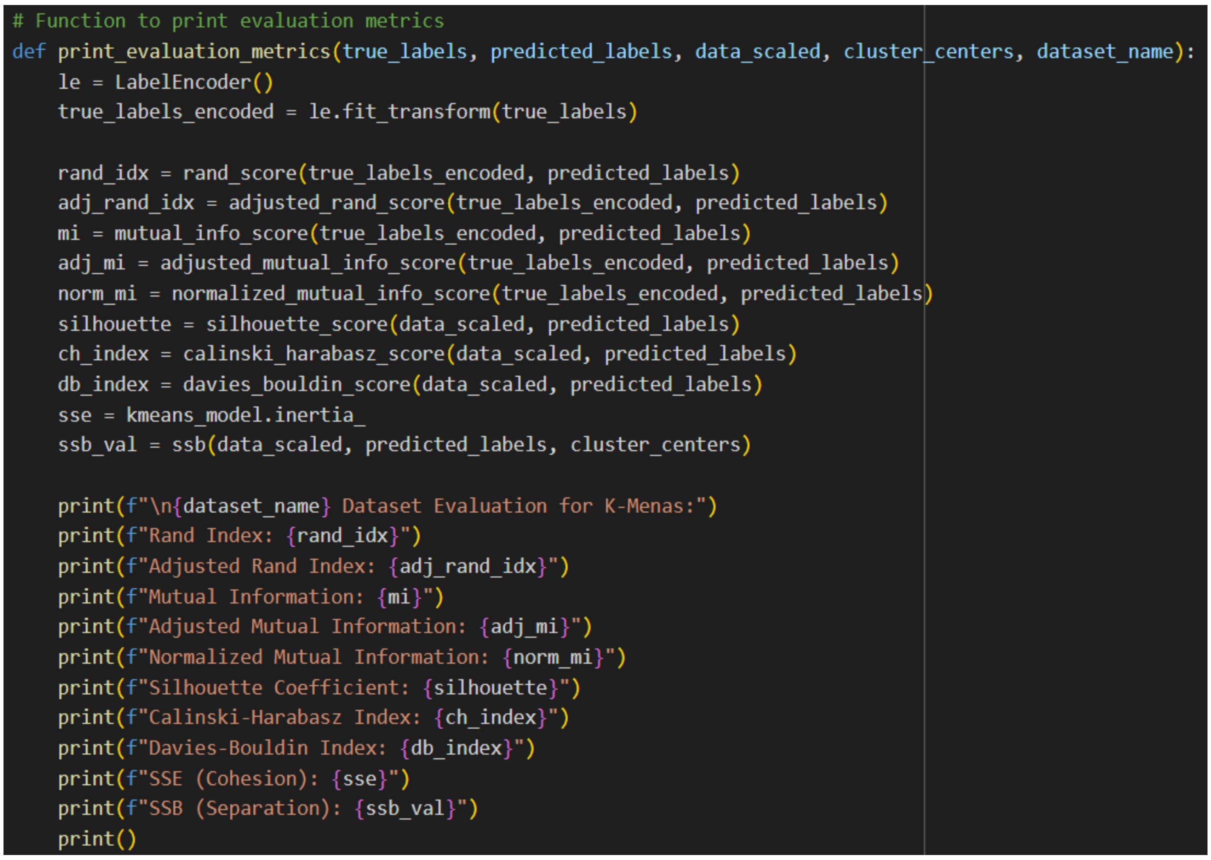
Code for K-Means on both datasets

# Importing necessary library for working with K-Means



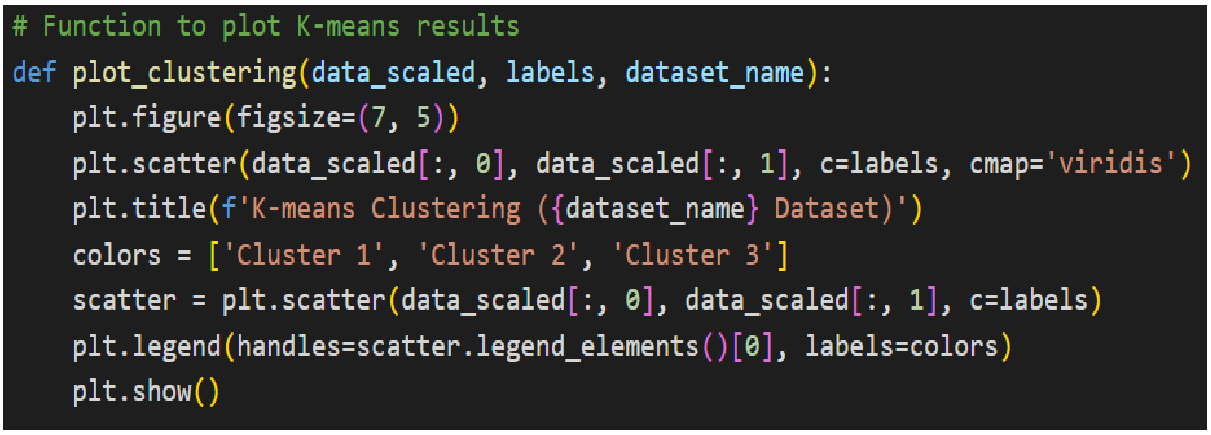
```
# K-Means

# Import necessary libraries
from sklearn.cluster import KMeans
```

# Function to print evaluation metrics on both datasets



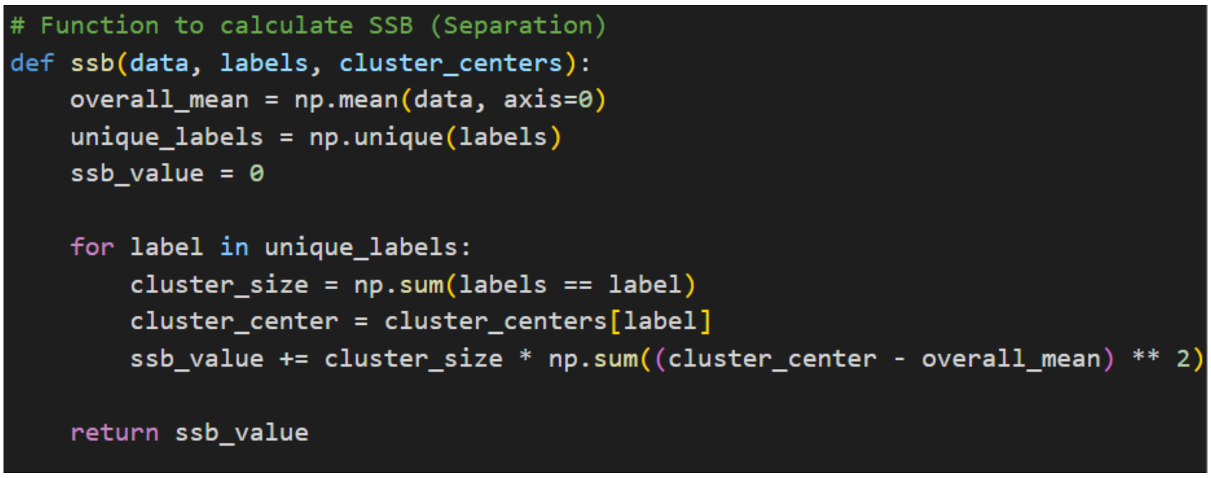
```
# Function to print evaluation metrics
def print_evaluation_metrics(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name):
    le = LabelEncoder()
    true_labels_encoded = le.fit_transform(true_labels)

    rand_idx = rand_score(true_labels_encoded, predicted_labels)
    adj_rand_idx = adjusted_rand_score(true_labels_encoded, predicted_labels)
    mi = mutual_info_score(true_labels_encoded, predicted_labels)
    adj_mi = adjusted_mutual_info_score(true_labels_encoded, predicted_labels)
    norm_mi = normalized_mutual_info_score(true_labels_encoded, predicted_labels)
    silhouette = silhouette_score(data_scaled, predicted_labels)
    ch_index = calinski_harabasz_score(data_scaled, predicted_labels)
    db_index = davies_bouldin_score(data_scaled, predicted_labels)
    sse = kmeans_model.inertia_
    ssb_val = ssb(data_scaled, predicted_labels, cluster_centers)

    print(f"\n{dataset_name} Dataset Evaluation for K-Menas:")
    print(f"Rand Index: {rand_idx}")
    print(f"Adjusted Rand Index: {adj_rand_idx}")
    print(f"Mutual Information: {mi}")
    print(f"Adjusted Mutual Information: {adj_mi}")
    print(f"Normalized Mutual Information: {norm_mi}")
    print(f"Silhouette Coefficient: {silhouette}")
    print(f"Calinski-Harabasz Index: {ch_index}")
    print(f"Davies-Bouldin Index: {db_index}")
    print(f"SSE (Cohesion): {sse}")
    print(f"SSB (Separation): {ssb_val}")
    print()
```

# Function to plot K-means results for both datasets



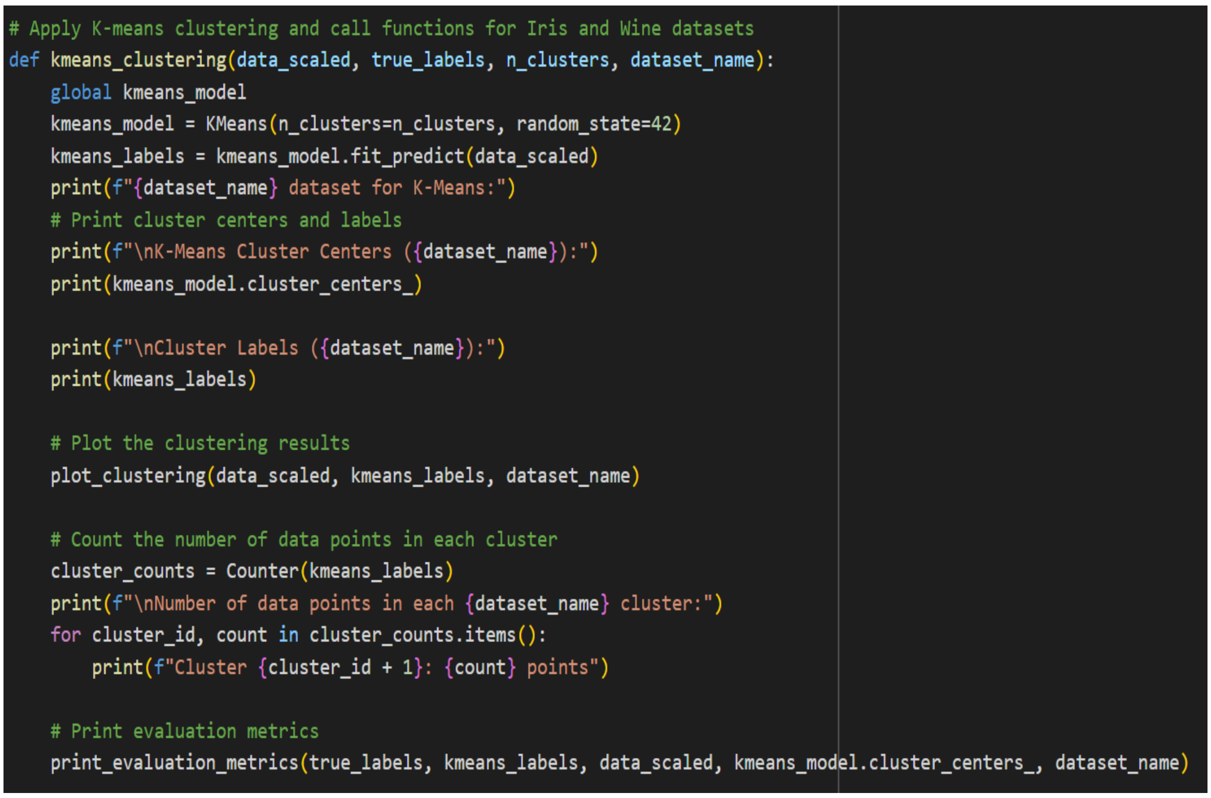
```
# Function to plot K-means results
def plot_clustering(data_scaled, labels, dataset_name):
    plt.figure(figsize=(7, 5))
    plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis')
    plt.title(f'K-means Clustering ({dataset_name} Dataset)')
    colors = ['Cluster 1', 'Cluster 2', 'Cluster 3']
    scatter = plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels)
    plt.legend(handles=scatter.legend_elements()[0], labels=colors)
    plt.show()
```

Function to calculate SSB (Separation)



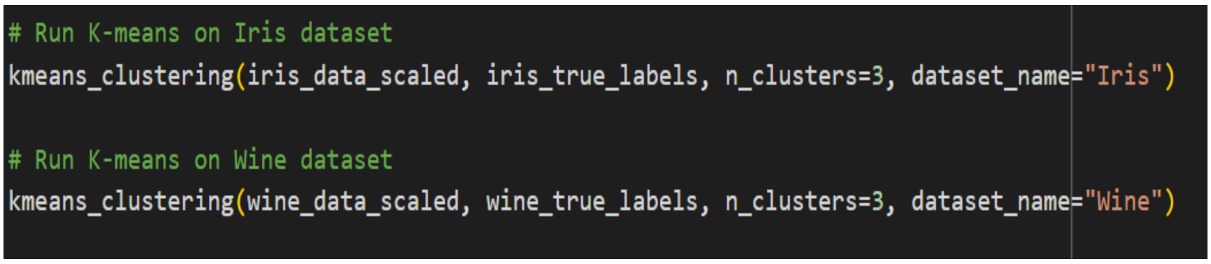
```python
# Function to calculate SSB (Separation)
def ssb(data, labels, cluster_centers):
    overall_mean = np.mean(data, axis=0)
    unique_labels = np.unique(labels)
    ssb_value = 0

    for label in unique_labels:
        cluster_size = np.sum(labels == label)
        cluster_center = cluster_centers[label]
        ssb_value += cluster_size * np.sum((cluster_center - overall_mean) ** 2)

    return ssb_value
```

# Applying K-means clustering and calling functions for on both datasets



```python
# Apply K-means clustering and call functions for Iris and Wine datasets
def kmeans_clustering(data_scaled, true_labels, n_clusters, dataset_name):
    global kmeans_model
    kmeans_model = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans_labels = kmeans_model.fit_predict(data_scaled)
    print(f"{dataset_name} dataset for K-Means:")
    # Print cluster centers and labels
    print(f"\nK-Means Cluster Centers ({dataset_name}):")
    print(kmeans_model.cluster_centers_)

    print(f"\nCluster Labels ({dataset_name}):")
    print(kmeans_labels)

    # Plot the clustering results
    plot_clustering(data_scaled, kmeans_labels, dataset_name)

    # Count the number of data points in each cluster
    cluster_counts = Counter(kmeans_labels)
    print(f"\nNumber of data points in each {dataset_name} cluster:")
    for cluster_id, count in cluster_counts.items():
        print(f"Cluster {cluster_id + 1}: {count} points")

    # Print evaluation metrics
    print_evaluation_metrics(true_labels, kmeans_labels, data_scaled, kmeans_model.cluster_centers_, dataset_name)
```

# Running K-means clustering for both datasets

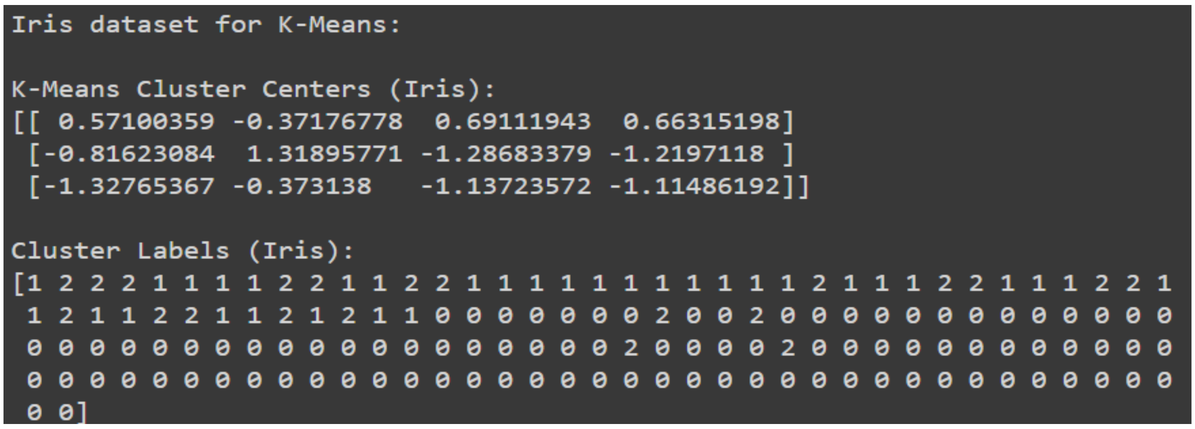


```python
# Run K-means on Iris dataset
kmeans_clustering(iris_data_scaled, iris_true_labels, n_clusters=3, dataset_name="Iris")

# Run K-means on Wine dataset
kmeans_clustering(wine_data_scaled, wine_true_labels, n_clusters=3, dataset_name="Wine")
```

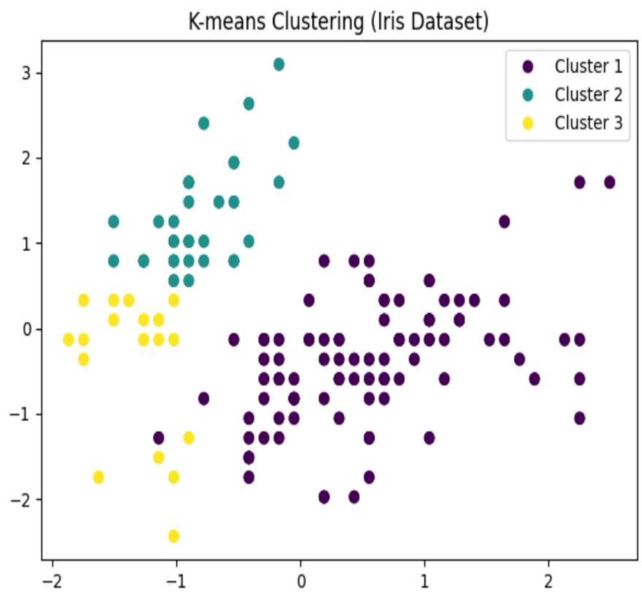
Analyse K-means for the Iris dataset

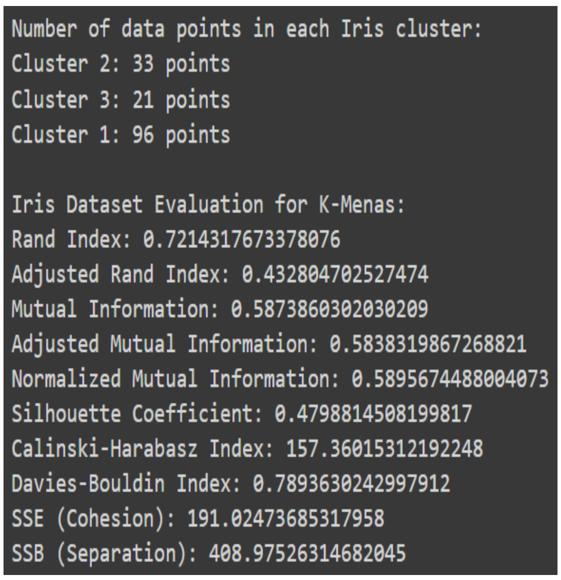
- In the above code it is clearly seen that for working with K-means we need to import the Kmeans library from sklearn.cluster
- K-means is partition base clustering which is very simple and easy to use to and effective in small datasets like Iris .
- The SSE for Iris dataset is around 191.025 which is very good but not best as the datasets are too small.
- The SSB for Iris dataset is around 488.97 which is very good but not best as the datasets are too small.

Output for Iris dataset



```
Iris dataset for K-Means:

K-Means Cluster Centers (Iris):
[[ 0.57100359 -0.37176778  0.69111943  0.66315198]
 [-0.81623084  1.31895771 -1.28683379 -1.2197118 ]
 [-1.32765367 -0.373138   -1.13723572 -1.11486192]]

Cluster Labels (Iris):
[1 2 2 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 1 1 2 2 1
 1 2 1 1 2 2 1 1 2 1 2 1 1 0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```



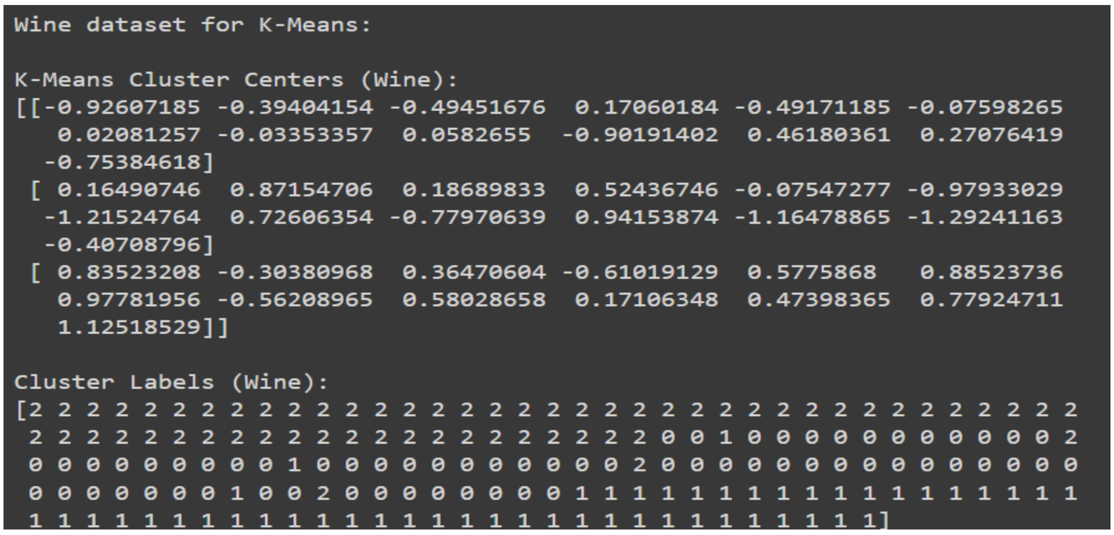


```
Number of data points in each Iris cluster:
Cluster 2: 33 points
Cluster 3: 21 points
Cluster 1: 96 points

Iris Dataset Evaluation for K-Menas:
Rand Index: 0.7214317673378076
Adjusted Rand Index: 0.432804702527474
Mutual Information: 0.5873860302030209
Adjusted Mutual Information: 0.5838319867268821
Normalized Mutual Information: 0.5895674488004073
Silhouette Coefficient: 0.4798814508199817
Calinski-Harabasz Index: 157.36015312192248
Davies-Bouldin Index: 0.7893630242997912
SSE (Cohesion): 191.02473685317958
SSB (Separation): 408.97526314682045
```

Analyse K-means for the Wine dataset

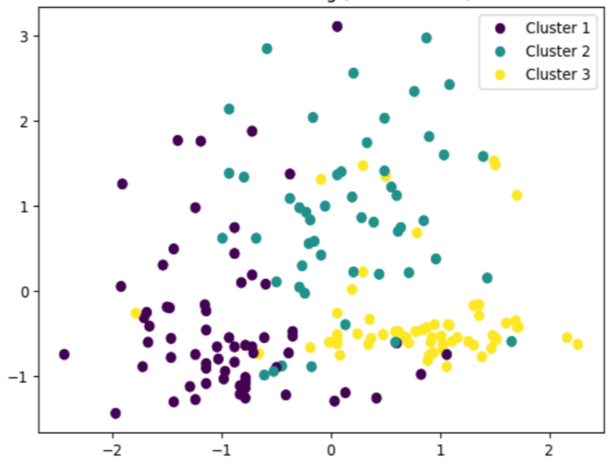
- The SSE for Wine dataset is around 1277.92 which is very good but not best as the datasets are too small.
- The SSB for Iris dataset is around 1036.071 which not good at all because datasets are too small.
- As, It is clearly seen from the information SSE>SSB which means the clustering is not working well because the datapoints are overlap to each other.
- But, Rand Index value is quite good in wine dataset compare to Iris dataset.

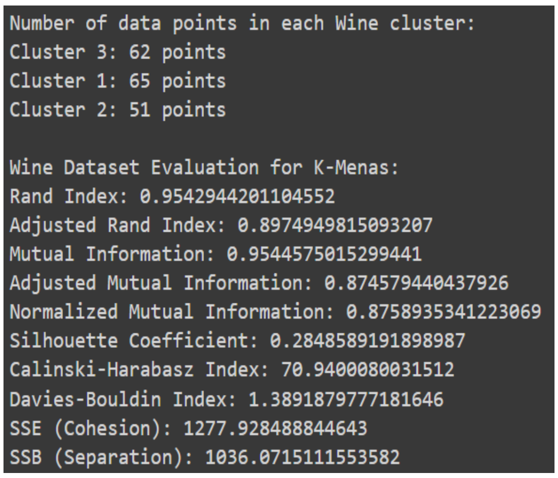
Output for Wine dataset



```
Wine dataset for K-Means:

K-Means Cluster Centers (Wine):
[[-0.92607185 -0.39404154 -0.49451676  0.17060184 -0.49171185 -0.07598265
   0.02081257 -0.03353357  0.0582655  -0.90191402  0.46180361  0.27076419
  -0.75384618]
 [ 0.16490746  0.87154706  0.18689833  0.52436746 -0.07547277 -0.97933029
  -1.21524764  0.72606354 -0.77970639  0.94153874 -1.16478865 -1.29241163
  -0.40708796]
 [ 0.83523208 -0.30380968  0.36470604 -0.61019129  0.5775868   0.88523736
   0.97781956 -0.56208965  0.58028658  0.17106348  0.47398365  0.77924711
   1.12518529]]

Cluster Labels (Wine):
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 2
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 2 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

K-means Clustering (Wine Dataset)

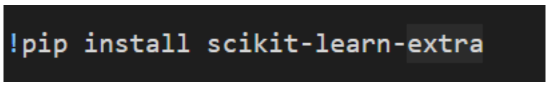




```
Number of data points in each Wine cluster:
Cluster 3: 62 points
Cluster 1: 65 points
Cluster 2: 51 points

Wine Dataset Evaluation for K-Menas:
Rand Index: 0.9542944201104552
Adjusted Rand Index: 0.8974949815093207
Mutual Information: 0.9544575015299441
Adjusted Mutual Information: 0.874579440437926
Normalized Mutual Information: 0.8758935341223069
Silhouette Coefficient: 0.2848589191898987
Calinski-Harabasz Index: 70.9400080031512
Davies-Bouldin Index: 1.3891879777181646
SSE (Cohesion): 1277.928488844643
SSB (Separation): 1036.0715111553582
```

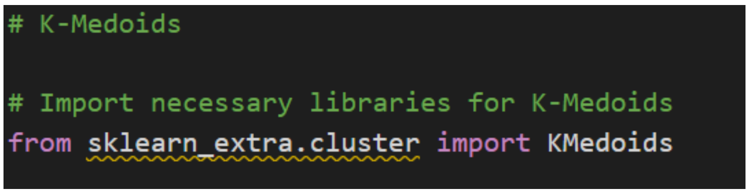
Code for K-Medoids on both datasets

# Installing package for working with K-Medoids



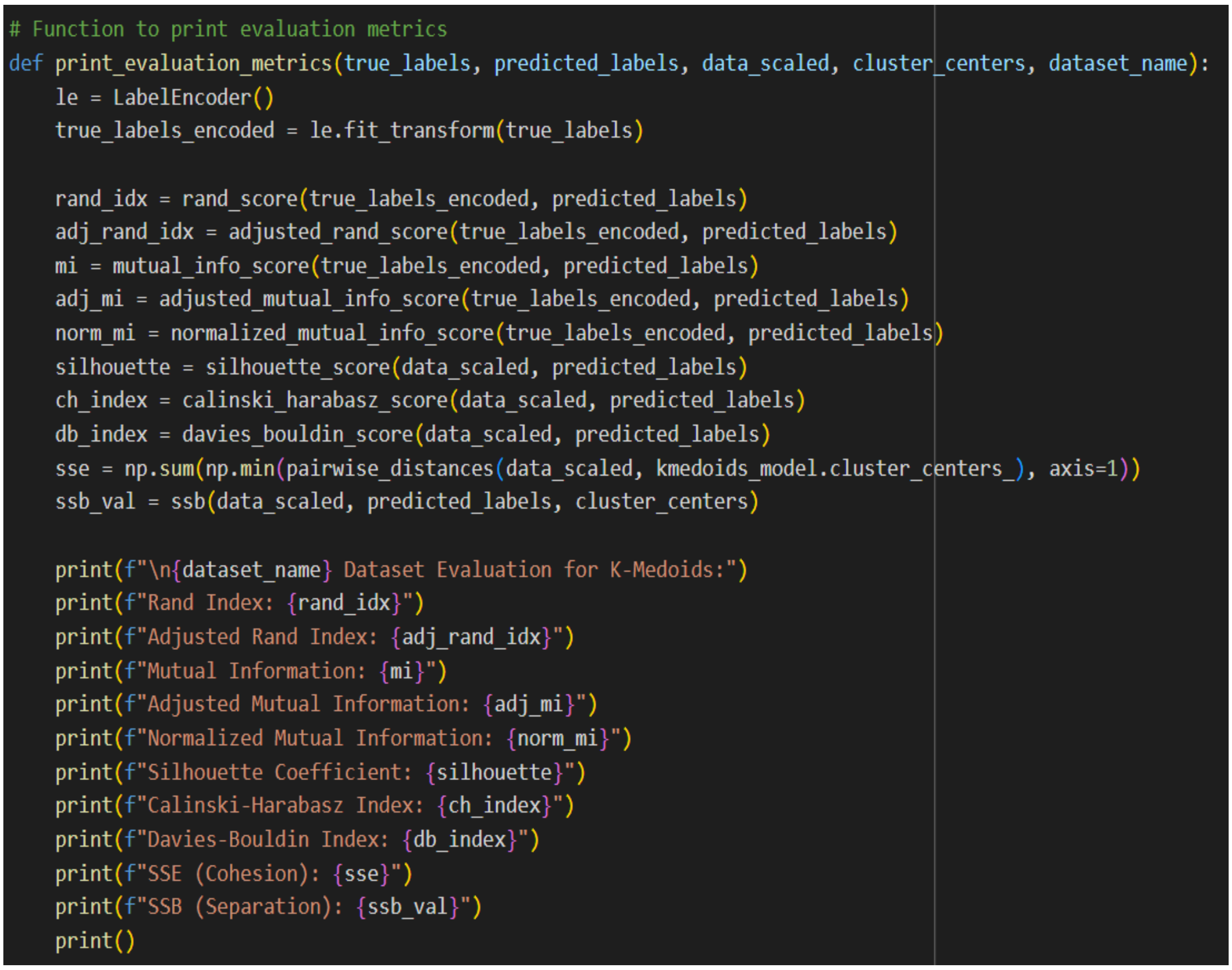
```
!pip install scikit-learn-extra
```

# Importing necessary library for working with K-Medoids



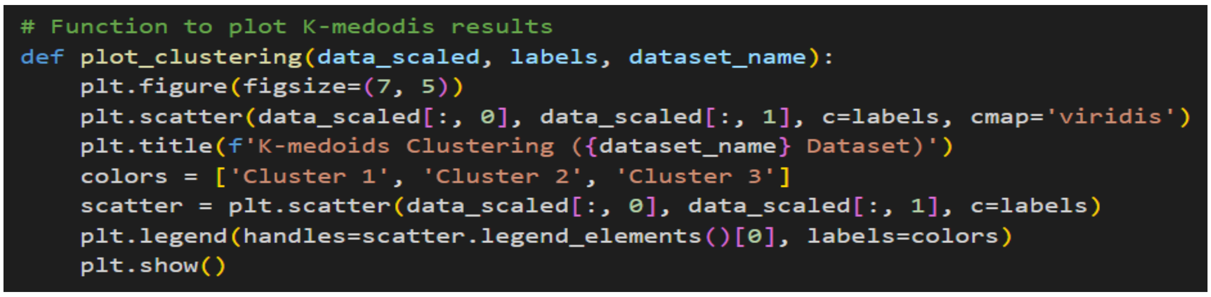
```
# K-Medoids

# Import necessary libraries for K-Medoids
from sklearn_extra.cluster import KMedoids
```

# Function to print evaluation metrics on both datasets



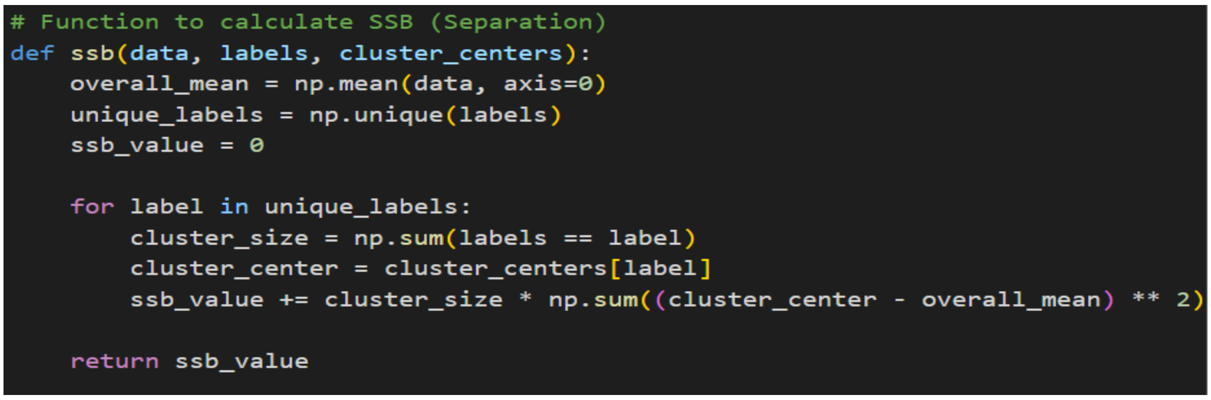
```
# Function to print evaluation metrics
def print_evaluation_metrics(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name):
    le = LabelEncoder()
    true_labels_encoded = le.fit_transform(true_labels)

    rand_idx = rand_score(true_labels_encoded, predicted_labels)
    adj_rand_idx = adjusted_rand_score(true_labels_encoded, predicted_labels)
    mi = mutual_info_score(true_labels_encoded, predicted_labels)
    adj_mi = adjusted_mutual_info_score(true_labels_encoded, predicted_labels)
    norm_mi = normalized_mutual_info_score(true_labels_encoded, predicted_labels)
    silhouette = silhouette_score(data_scaled, predicted_labels)
    ch_index = calinski_harabasz_score(data_scaled, predicted_labels)
    db_index = davies_bouldin_score(data_scaled, predicted_labels)
    sse = np.sum(np.min(pairwise_distances(data_scaled, kmedoids_model.cluster_centers_), axis=1))
    ssb_val = ssb(data_scaled, predicted_labels, cluster_centers)

    print(f"\n{dataset_name} Dataset Evaluation for K-Medoids:")
    print(f"Rand Index: {rand_idx}")
    print(f"Adjusted Rand Index: {adj_rand_idx}")
    print(f"Mutual Information: {mi}")
    print(f"Adjusted Mutual Information: {adj_mi}")
    print(f"Normalized Mutual Information: {norm_mi}")
    print(f"Silhouette Coefficient: {silhouette}")
    print(f"Calinski-Harabasz Index: {ch_index}")
    print(f"Davies-Bouldin Index: {db_index}")
    print(f"SSE (Cohesion): {sse}")
    print(f"SSB (Separation): {ssb_val}")
    print()
```

# Function to plot K-medoids results for both datasets



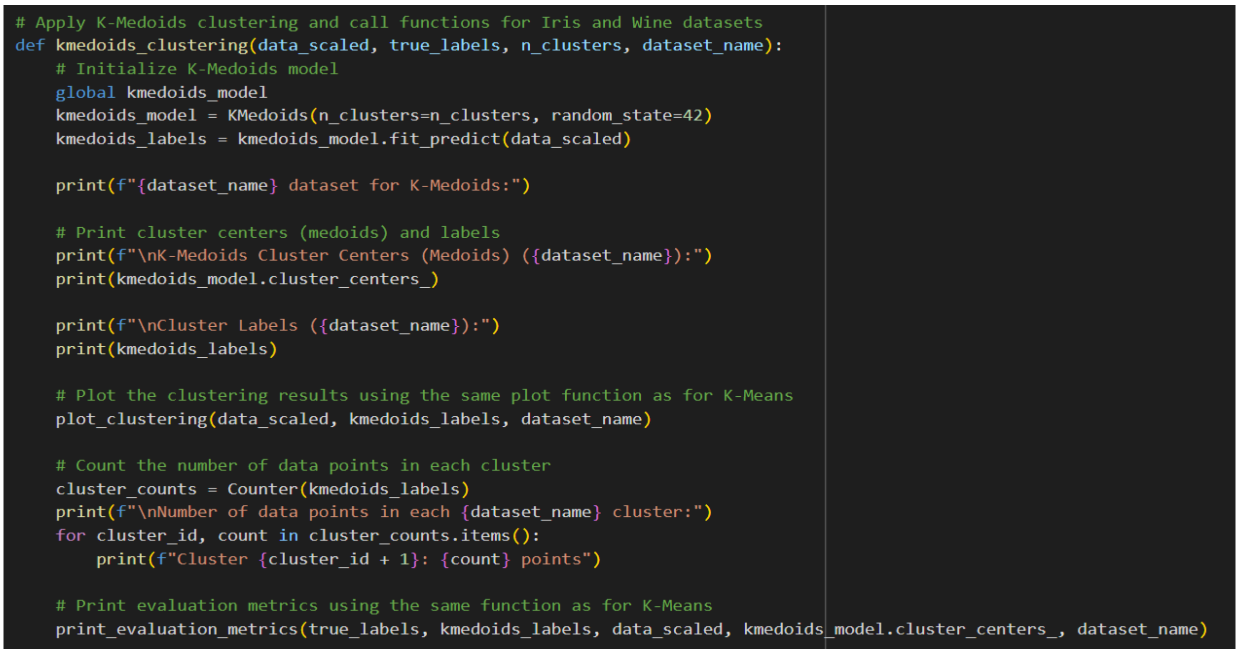
```python
# Function to plot K-medodis results
def plot_clustering(data_scaled, labels, dataset_name):
    plt.figure(figsize=(7, 5))
    plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis')
    plt.title(f'K-medoids Clustering ({dataset_name} Dataset)')
    colors = ['Cluster 1', 'Cluster 2', 'Cluster 3']
    scatter = plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels)
    plt.legend(handles=scatter.legend_elements()[0], labels=colors)
    plt.show()
```

Function to calculate SSB (Separation)



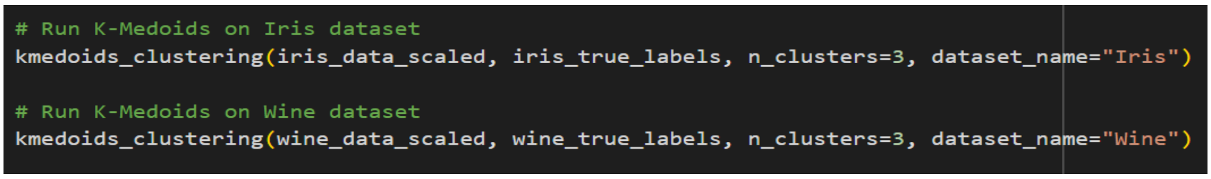
```python
# Function to calculate SSB (Separation)
def ssb(data, labels, cluster_centers):
    overall_mean = np.mean(data, axis=0)
    unique_labels = np.unique(labels)
    ssb_value = 0

    for label in unique_labels:
        cluster_size = np.sum(labels == label)
        cluster_center = cluster_centers[label]
        ssb_value += cluster_size * np.sum((cluster_center - overall_mean) ** 2)

    return ssb_value
```

# Applying K-medoids clustering and calling functions for on both datasets



```python
# Apply K-Medoids clustering and call functions for Iris and Wine datasets
def kmedoids_clustering(data_scaled, true_labels, n_clusters, dataset_name):
    # Initialize K-Medoids model
    global kmedoids_model
    kmedoids_model = KMedoids(n_clusters=n_clusters, random_state=42)
    kmedoids_labels = kmedoids_model.fit_predict(data_scaled)

    print(f"{dataset_name} dataset for K-Medoids:")

    # Print cluster centers (medoids) and labels
    print(f"\nK-Medoids Cluster Centers (Medoids) ({dataset_name}):")
    print(kmedoids_model.cluster_centers_)

    print(f"\nCluster Labels ({dataset_name}):")
    print(kmedoids_labels)

    # Plot the clustering results using the same plot function as for K-Means
    plot_clustering(data_scaled, kmedoids_labels, dataset_name)

    # Count the number of data points in each cluster
    cluster_counts = Counter(kmedoids_labels)
    print(f"\nNumber of data points in each {dataset_name} cluster:")
    for cluster_id, count in cluster_counts.items():
        print(f"Cluster {cluster_id + 1}: {count} points")

    # Print evaluation metrics using the same function as for K-Means
    print_evaluation_metrics(true_labels, kmedoids_labels, data_scaled, kmedoids_model.cluster_centers_, dataset_name)
```

# Running K-medoids clustering for both datasets

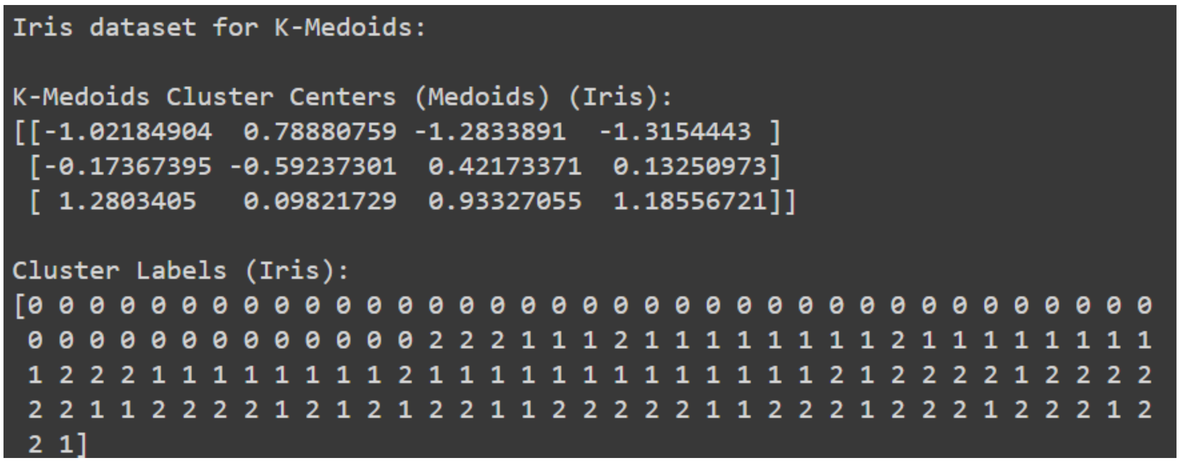


```python
# Run K-Medoids on Iris dataset
kmedoids_clustering(iris_data_scaled, iris_true_labels, n_clusters=3, dataset_name="Iris")

# Run K-Medoids on Wine dataset
kmedoids_clustering(wine_data_scaled, wine_true_labels, n_clusters=3, dataset_name="Wine")
```

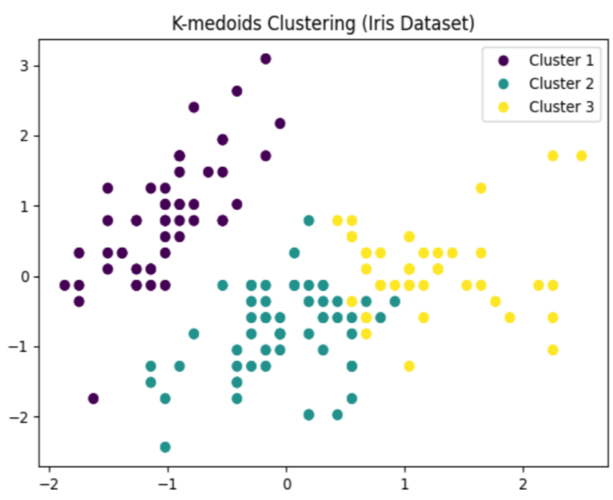
Analyse K-medoids for the Iris dataset

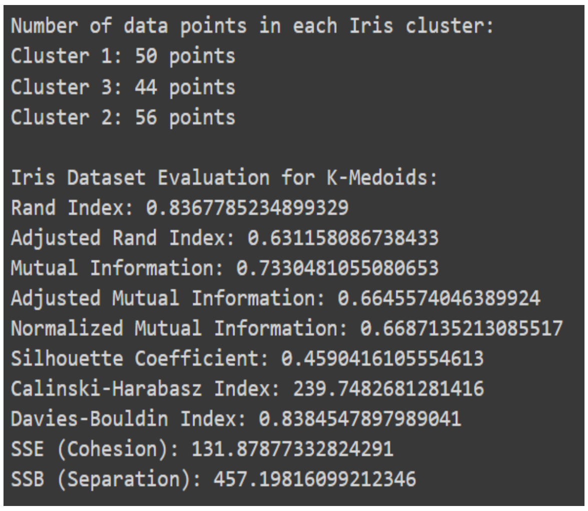
- To working with K-medoids we have to first install the **scikit-lear-extra** package & also import Kmedoids library.
- The rand index which is around 0.83 which good and better than Kmeans.
- The SSE for Iris dataset is around 131 which is good and in each cluster data point are compactly tight to each other.
- The SSB for Iris dataset is around 457 which is good and well define that’s why each cluster sperate to each other finely.

Output for Iris dataset



```
Iris dataset for K-Medoids:

K-Medoids Cluster Centers (Medoids) (Iris):
[[-1.02184904  0.78880759 -1.2833891  -1.3154443 ]
 [-0.17367395 -0.59237301  0.42173371  0.13250973]
 [ 1.2803405   0.09821729  0.93327055  1.18556721]]

Cluster Labels (Iris):
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1
 1 2 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
```



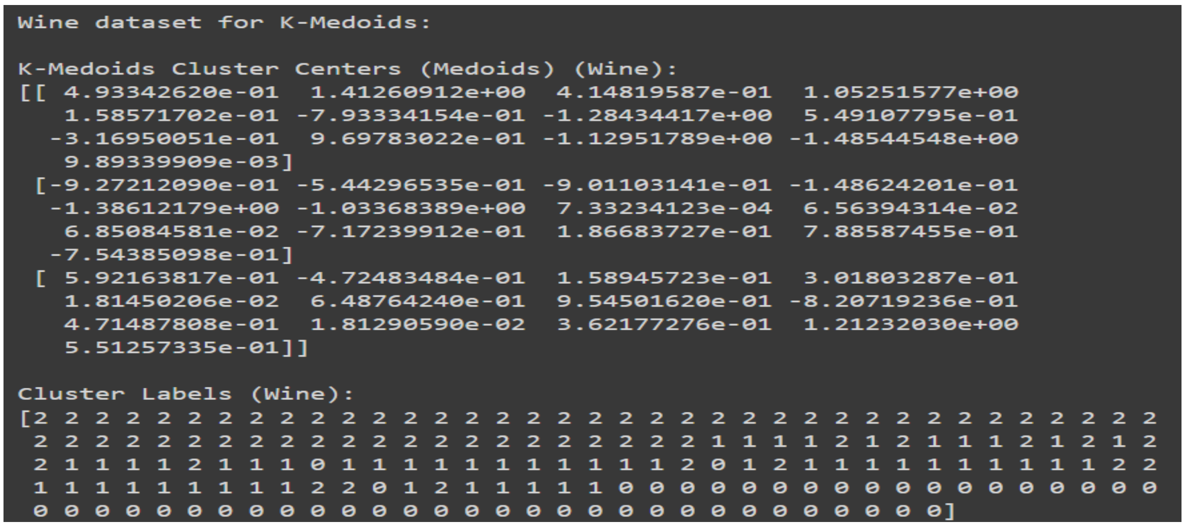


```
Number of data points in each Iris cluster:
Cluster 1: 50 points
Cluster 3: 44 points
Cluster 2: 56 points

Iris Dataset Evaluation for K-Medoids:
Rand Index: 0.8367785234899329
Adjusted Rand Index: 0.631158086738433
Mutual Information: 0.7330481055080653
Adjusted Mutual Information: 0.6645574046389924
Normalized Mutual Information: 0.6687135213085517
Silhouette Coefficient: 0.4590416105554613
Calinski-Harabasz Index: 239.7482681281416
Davies-Bouldin Index: 0.8384547897989041
SSE (Cohesion): 131.87877332824291
SSB (Separation): 457.19816099212346
```

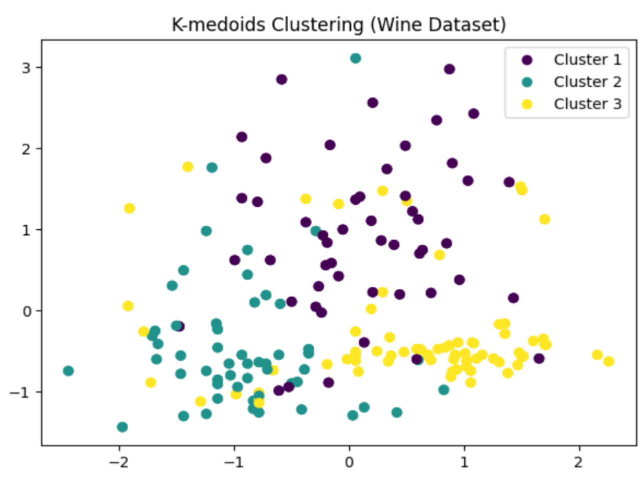
Analyse K-medoids for the Wine dataset

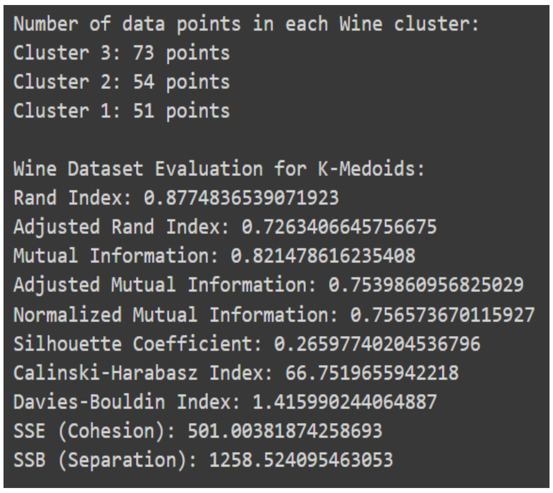
- The rand index which is around 0.87 which good and better than Kmeans.
- The SSE for Iris dataset is around 501 which is good and in each cluster data point are compactly tight to each other.
- The SSB for Iris dataset is around 1258 which is good and well define and each cluster sperate to each other finely.
- Even though SSB>SSE the plot of wine dataset is a bit messy each data points over lap to each other the potential reason is that we know clustering technique is useful for large and complex datasets and the given datasets are too small.

Output for Wine dataset



```
Wine dataset for K-Medoids:

K-Medoids Cluster Centers (Medoids) (Wine):
[[ 4.93342620e-01  1.41260912e+00  4.14819587e-01  1.05251577e+00
   1.58571702e-01 -7.93334154e-01 -1.28434417e+00  5.49107795e-01
  -3.16950051e-01  9.69783022e-01 -1.12951789e+00 -1.48544548e+00
   9.89339909e-03]
 [-9.27212090e-01 -5.44296535e-01 -9.01103141e-01 -1.48624201e-01
  -1.38612179e+00 -1.03368389e+00  7.33234123e-04  6.56394314e-02
   6.85084581e-02 -7.17239912e-01  1.86683727e-01  7.88587455e-01
  -7.54385098e-01]
 [ 5.92163817e-01 -4.72483484e-01  1.58945723e-01  3.01803287e-01
   1.81450206e-02  6.48764240e-01  9.54501620e-01 -8.20719236e-01
   4.71487808e-01  1.81290590e-02  3.62177276e-01  1.21232030e+00
   5.51257335e-01]]

Cluster Labels (Wine):
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 2 1 2 1 1 1 2 1 2 1 2
 2 1 1 1 1 2 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 2 0 1 2 1 1 1 1 1 1 1 1 1 1 2 2
 1 1 1 1 1 1 1 1 1 2 2 0 1 2 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```


K-medoids Clustering (Wine Dataset)



```
Number of data points in each Wine cluster:
Cluster 3: 73 points
Cluster 2: 54 points
Cluster 1: 51 points

Wine Dataset Evaluation for K-Medoids:
Rand Index: 0.8774836539071923
Adjusted Rand Index: 0.7263406645756675
Mutual Information: 0.821478616235408
Adjusted Mutual Information: 0.7539860956825029
Normalized Mutual Information: 0.756573670115927
Silhouette Coefficient: 0.26597740204536796
Calinski-Harabasz Index: 66.7519655942218
Davies-Bouldin Index: 1.415990244064887
SSE (Cohesion): 501.00381874258693
SSB (Separation): 1258.524095463053
```

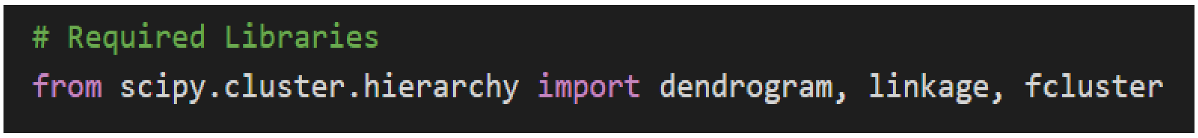
Conclusion for Partition bases Clustering

1. **Effective Clustering**:
    - Both K-means and K-medoids effectively partition the Iris and Wine datasets into distinct clusters, generally aligning with the true class labels.
2. **Performance Comparison**:
    - K-means tends to converge faster and is computationally less expensive than K-medoids due to its centroid-based approach.
    - K-medoids, though slower, is more robust to noise and outliers since it uses actual data points (medoids) as cluster centers.
3. **Accuracy**:
    - On the Iris dataset, both algorithms achieve high accuracy, often clustering the three species with minimal overlap.
    - For the Wine dataset, K-means and K-medoids show reasonable accuracy but may struggle slightly due to the higher dimensionality and complex class boundaries.
4. **Clustering Evaluation Metrics**:
    - Both algorithms show good Silhouette Coefficients, indicating well-defined clusters.
    - Calinski-Harabasz and Davies-Bouldin Index values support the clustering quality, with K-medoids generally providing slightly better separation.

In practice, the choice between K-means and K-medoids depends on the dataset characteristics (e.g., presence of outliers, cluster shape) and the computational resources available.

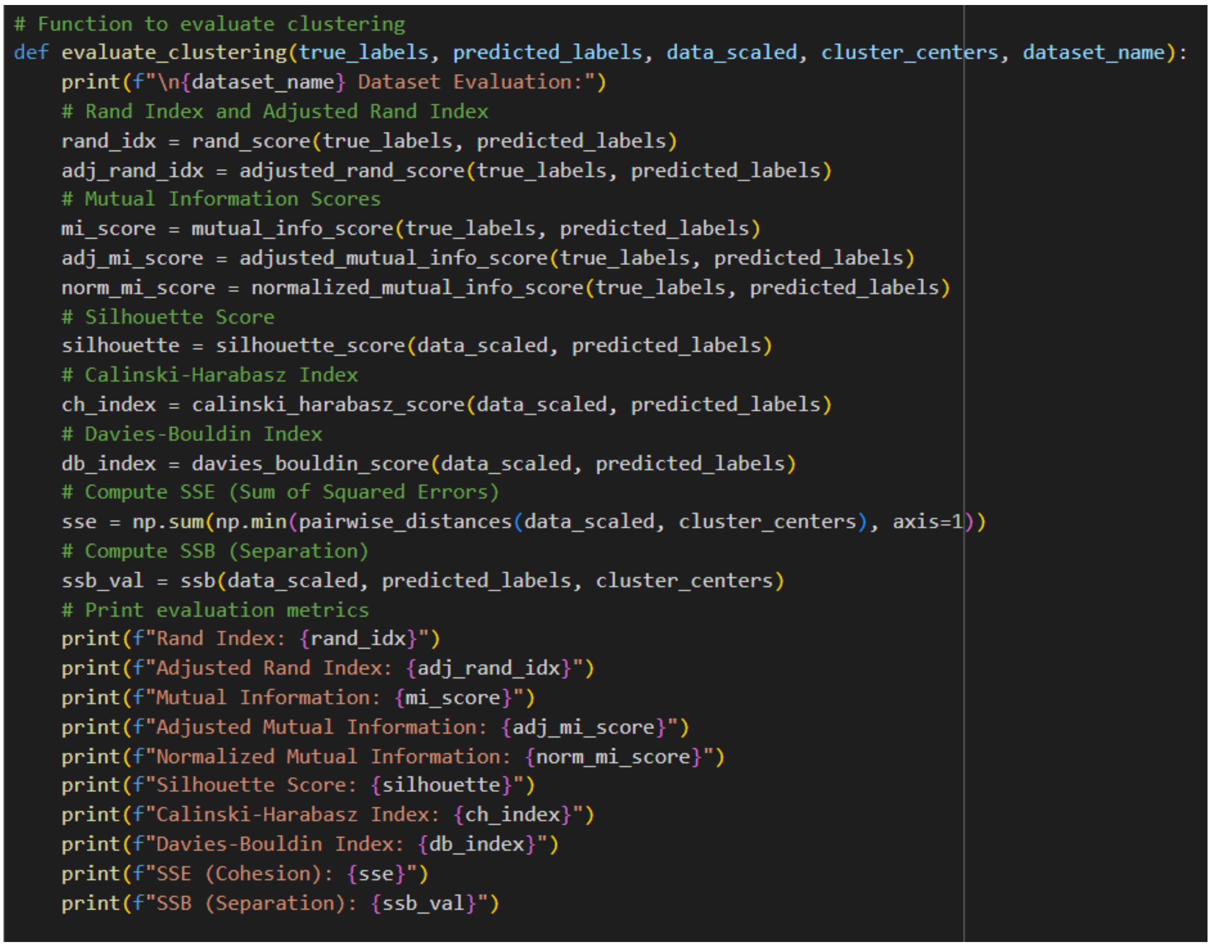
Code for Dendrogram on both datasets

# Importing necessary library for working with Dendrogram



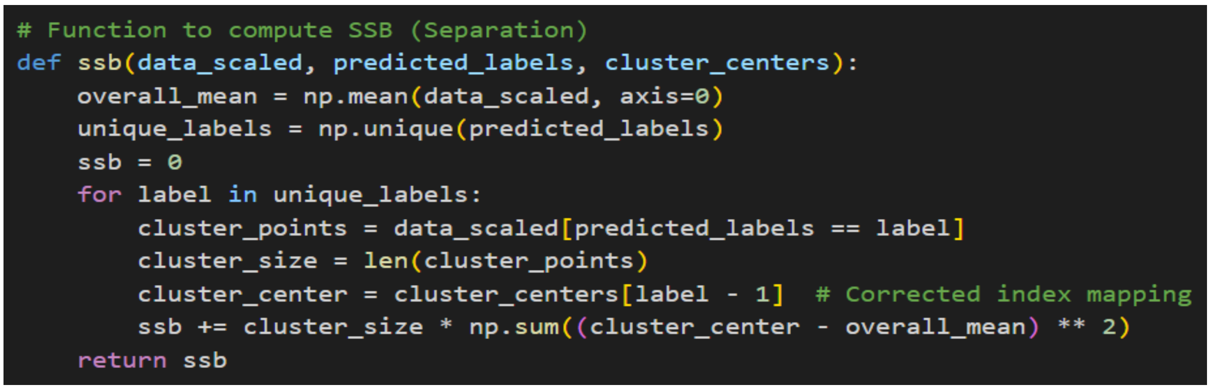
```
# Required Libraries
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
```

# Function to print evaluation metrics on both datasets



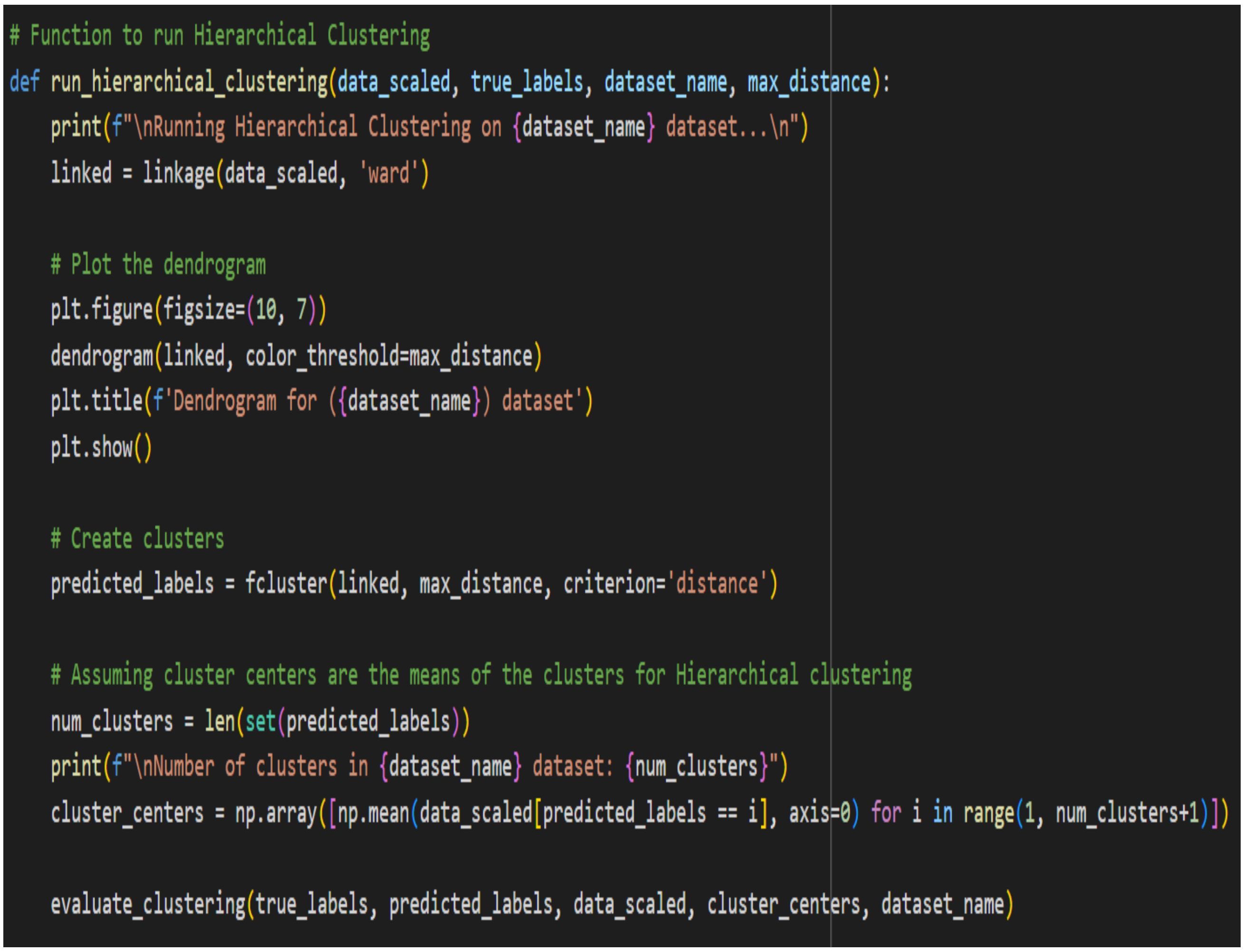
```
# Function to evaluate clustering
def evaluate_clustering(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name):
    print(f"\n{dataset_name} Dataset Evaluation:")
    # Rand Index and Adjusted Rand Index
    rand_idx = rand_score(true_labels, predicted_labels)
    adj_rand_idx = adjusted_rand_score(true_labels, predicted_labels)
    # Mutual Information Scores
    mi_score = mutual_info_score(true_labels, predicted_labels)
    adj_mi_score = adjusted_mutual_info_score(true_labels, predicted_labels)
    norm_mi_score = normalized_mutual_info_score(true_labels, predicted_labels)
    # Silhouette Score
    silhouette = silhouette_score(data_scaled, predicted_labels)
    # Calinski-Harabasz Index
    ch_index = calinski_harabasz_score(data_scaled, predicted_labels)
    # Davies-Bouldin Index
    db_index = davies_bouldin_score(data_scaled, predicted_labels)
    # Compute SSE (Sum of Squared Errors)
    sse = np.sum(np.min(pairwise_distances(data_scaled, cluster_centers), axis=1))
    # Compute SSB (Separation)
    ssb_val = ssb(data_scaled, predicted_labels, cluster_centers)
    # Print evaluation metrics
    print(f"Rand Index: {rand_idx}")
    print(f"Adjusted Rand Index: {adj_rand_idx}")
    print(f"Mutual Information: {mi_score}")
    print(f"Adjusted Mutual Information: {adj_mi_score}")
    print(f"Normalized Mutual Information: {norm_mi_score}")
    print(f"Silhouette Score: {silhouette}")
    print(f"Calinski-Harabasz Index: {ch_index}")
    print(f"Davies-Bouldin Index: {db_index}")
    print(f"SSE (Cohesion): {sse}")
    print(f"SSB (Separation): {ssb_val}")
```

# Function to calculate SSB (Separation)



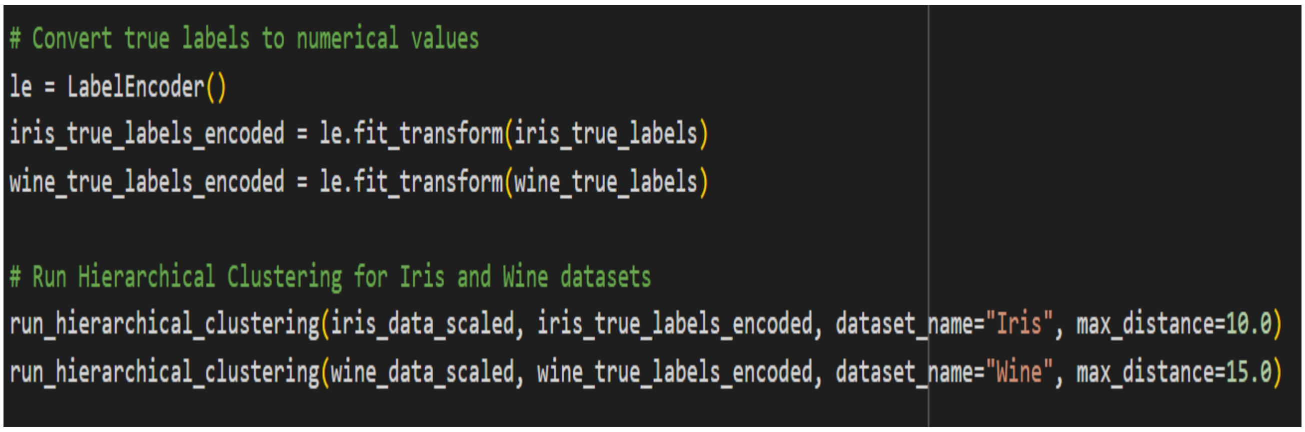
```
# Function to compute SSB (Separation)
def ssb(data_scaled, predicted_labels, cluster_centers):
    overall_mean = np.mean(data_scaled, axis=0)
    unique_labels = np.unique(predicted_labels)
    ssb = 0
    for label in unique_labels:
        cluster_points = data_scaled[predicted_labels == label]
        cluster_size = len(cluster_points)
        cluster_center = cluster_centers[label - 1]  # Corrected index mapping
        ssb += cluster_size * np.sum((cluster_center - overall_mean) ** 2)
    return ssb
```

# Function to run the hierarchical clustering for both datasets



```
# Function to run Hierarchical Clustering
def run_hierarchical_clustering(data_scaled, true_labels, dataset_name, max_distance):
    print(f"\nRunning Hierarchical Clustering on {dataset_name} dataset...\n")
    linked = linkage(data_scaled, 'ward')

    # Plot the dendrogram
    plt.figure(figsize=(10, 7))
    dendrogram(linked, color_threshold=max_distance)
    plt.title(f'Dendrogram for ({dataset_name}) dataset')
    plt.show()

    # Create clusters
    predicted_labels = fcluster(linked, max_distance, criterion='distance')

    # Assuming cluster centers are the means of the clusters for Hierarchical clustering
    num_clusters = len(set(predicted_labels))
    print(f"\nNumber of clusters in {dataset_name} dataset: {num_clusters}")
    cluster_centers = np.array([np.mean(data_scaled[predicted_labels == i], axis=0) for i in range(1, num_clusters+1)])

    evaluate_clustering(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name)
```

# Converting labels into numeric & applying Dendrogram clustering and calling functions for on both datasets



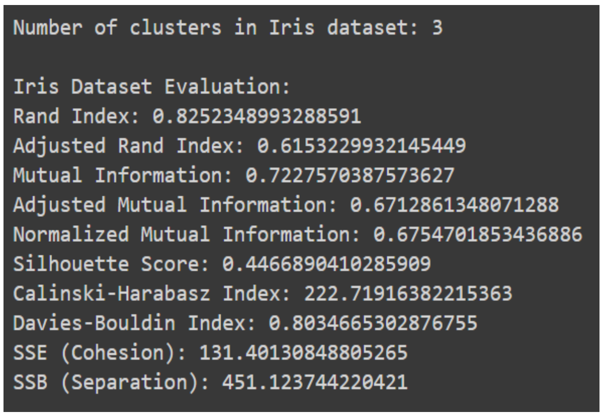
```
# Convert true labels to numerical values
le = LabelEncoder()
iris_true_labels_encoded = le.fit_transform(iris_true_labels)
wine_true_labels_encoded = le.fit_transform(wine_true_labels)

# Run Hierarchical Clustering for Iris and Wine datasets
run_hierarchical_clustering(iris_data_scaled, iris_true_labels_encoded, dataset_name="Iris", max_distance=10.0)
run_hierarchical_clustering(wine_data_scaled, wine_true_labels_encoded, dataset_name="Wine", max_distance=15.0)
```

Analyse Dendrogram for both dataset

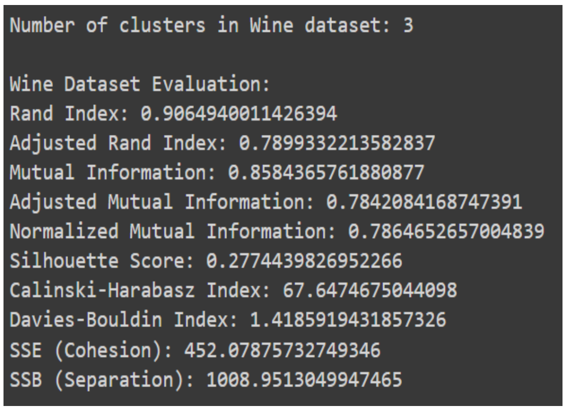
- The code utilizes hierarchical clustering with the Ward linkage method, which is effective for visualizing the hierarchical structure of clusters in both datasets.
- The dendrogram provides a visual representation of the clusters and their relationships, allowing you to observe the points where clusters merge and the level of similarity between clusters.
- The code calculates key clustering metrics, including Rand Index, Adjusted Rand Index, Mutual Information scores, Silhouette Score, Calinski-Harabasz Index, and Davies-Bouldin Index, to evaluate the quality and cohesion of the clusters.
- By computing both cohesion (SSE) and separation (SSB) metrics, the code provides insights into how tightly grouped individual clusters are (cohesion) and how distinct they are from each other (separation).
- Although hierarchical clustering does not inherently compute cluster centers, the code estimates them by averaging the data points within each cluster.
- The evaluate_clustering function allows for an in-depth interpretation of the results, helping identify if clusters are well-defined and how they correspond to the true class labels, thus aiding in assessing the clustering method's effectiveness for each dataset.

This code provides a well-rounded approach to hierarchical clustering by combining visualization, cluster formation, and a variety of evaluation metrics, allowing for a robust analysis of the clustering outcomes on both the Iris and Wine datasets.

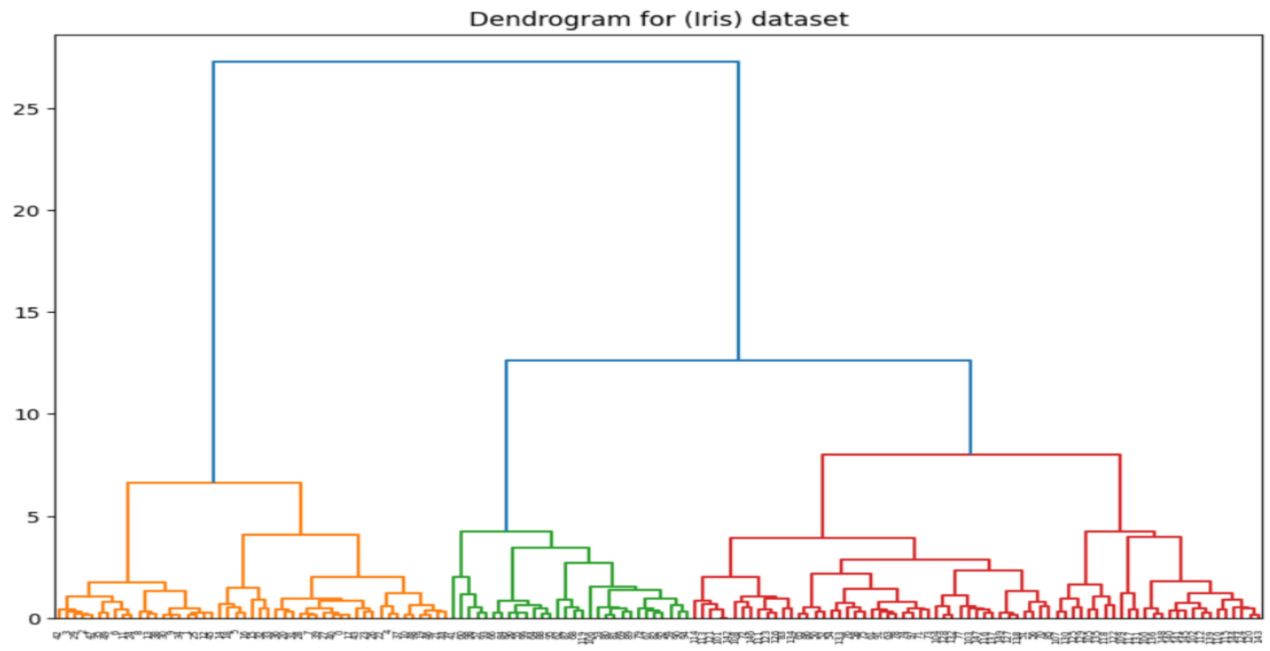
Output for both datasets

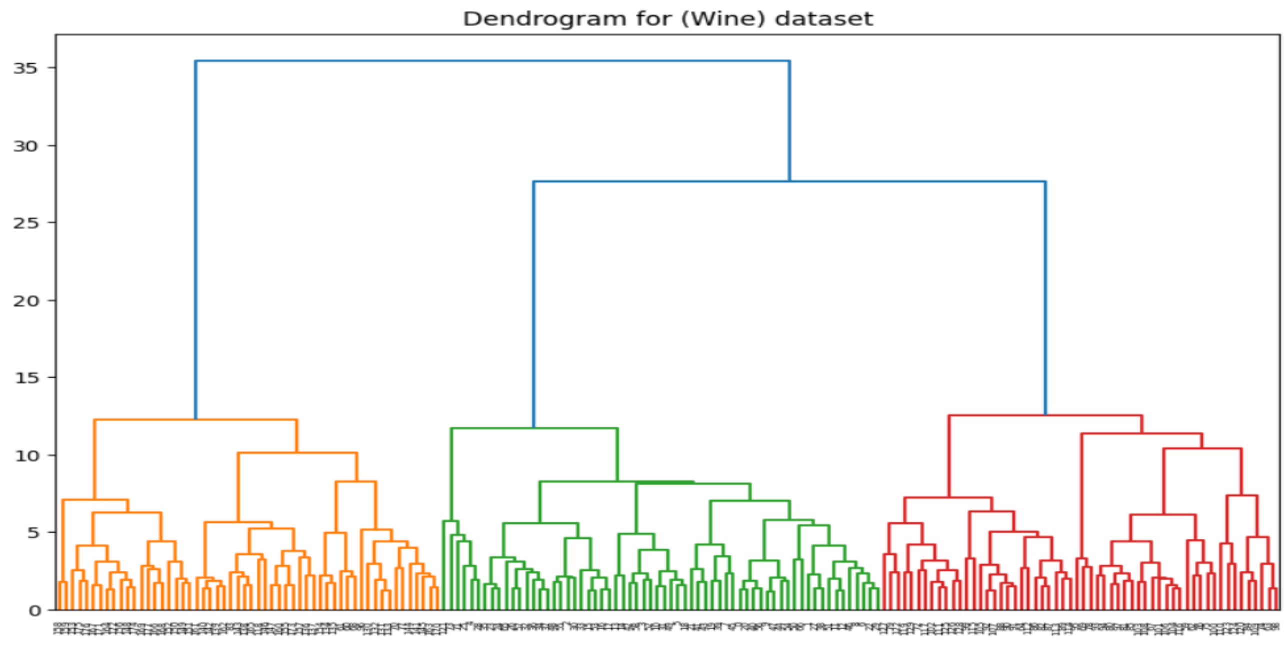


```
Number of clusters in Iris dataset: 3

Iris Dataset Evaluation:
Rand Index: 0.8252348993288591
Adjusted Rand Index: 0.6153229932145449
Mutual Information: 0.7227570387573627
Adjusted Mutual Information: 0.6712861348071288
Normalized Mutual Information: 0.6754701853436886
Silhouette Score: 0.4466890410285909
Calinski-Harabasz Index: 222.71916382215363
Davies-Bouldin Index: 0.8034665302876755
SSE (Cohesion): 131.40130848805265
SSB (Separation): 451.123744220421
```



```
Number of clusters in Wine dataset: 3

Wine Dataset Evaluation:
Rand Index: 0.9064940011426394
Adjusted Rand Index: 0.7899332213582837
Mutual Information: 0.8584365761880877
Adjusted Mutual Information: 0.7842084168747391
Normalized Mutual Information: 0.7864652657004839
Silhouette Score: 0.2774439826952266
Calinski-Harabasz Index: 67.6474675044098
Davies-Bouldin Index: 1.4185919431857326
SSE (Cohesion): 452.07875732749346
SSB (Separation): 1008.9513049947465
```





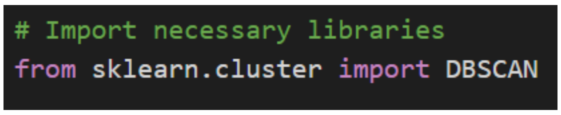
Conclusion for Hierarchical base Clustering

1. **Hierarchical Insights:**
   - The dendrogram provides a clear hierarchical structure for the Iris dataset, which is well-suited for this method due to its simpler, more separable clusters.
   - In contrast, the Wine dataset shows more complex relationships between clusters, with some classes overlapping, making the hierarchical structure more challenging to interpret.
2. **Cluster Interpretability:**
   - For the Iris dataset, distinct clusters corresponding to the three species are typically identifiable, providing meaningful insights into the natural grouping of the data.
   - The Wine dataset clusters are less defined, indicating that hierarchical clustering may not be as effective for datasets with higher dimensionality and more intricate relationships between classes.
3. **Evaluating Cluster Quality:**
   - The calculated evaluation metrics, including cohesion (SSE) and separation (SSB), show that hierarchical clustering on the Iris dataset results in relatively well-separated and cohesive clusters.
   - For the Wine dataset, the separation metrics indicate some overlapping clusters, suggesting that other clustering methods might be more effective for complex data with overlapping or non-spherical clusters.

Dendrogram-based clustering demonstrates strong performance on the Iris dataset, revealing its hierarchical structure clearly. However, on the Wine dataset, the results are less definitive, underscoring the potential limitations of hierarchical clustering for datasets with higher dimensionality and complex relationships.

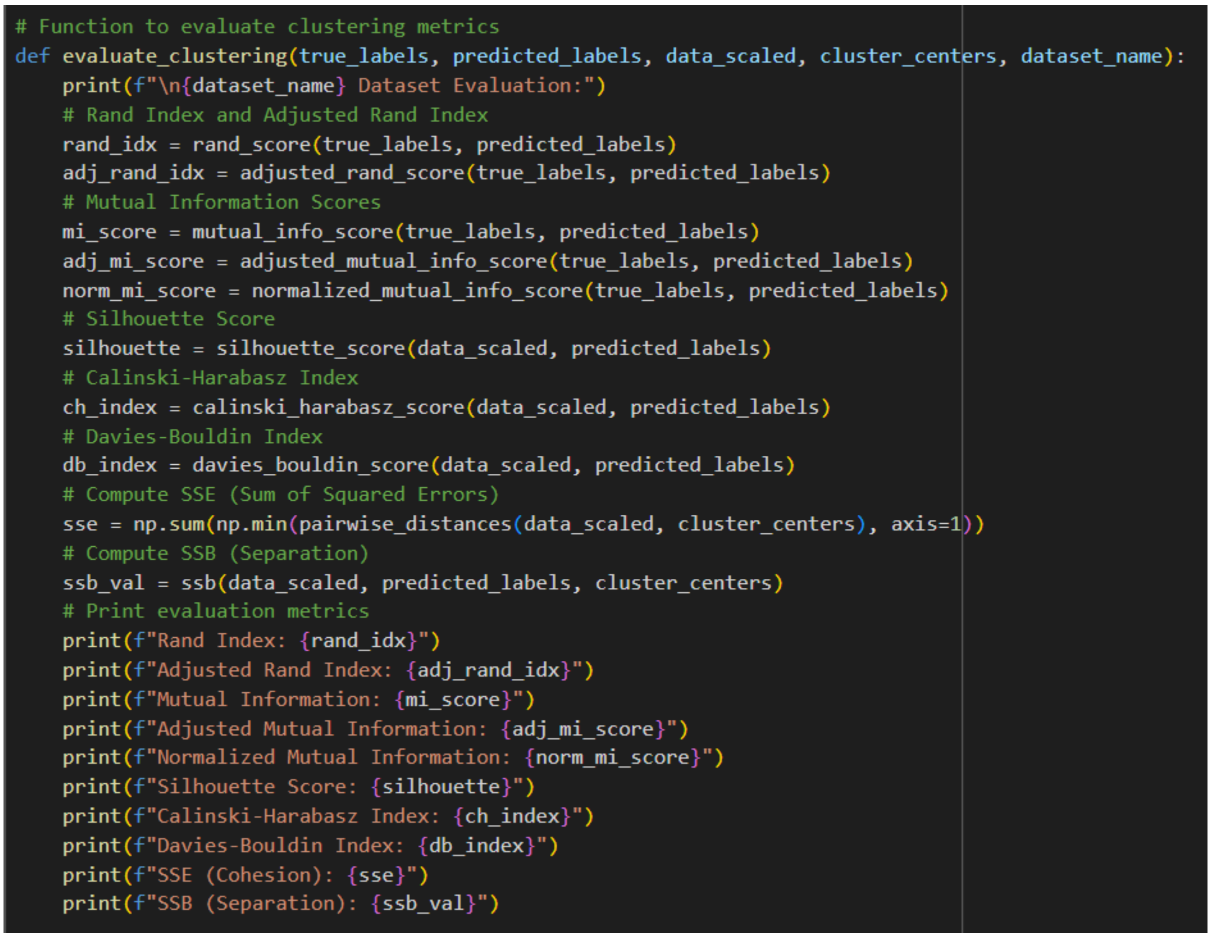
Code for DBSCAN on both datasets

# Importing necessary library for working with



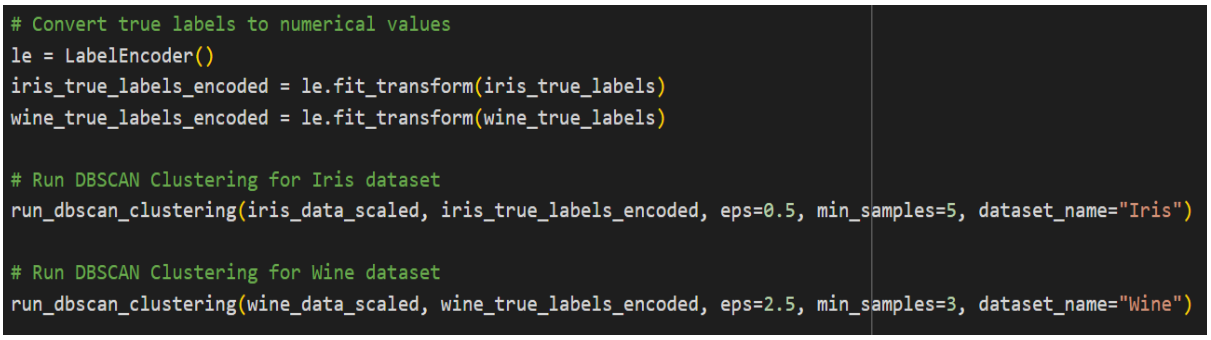
```python
# Import necessary libraries
from sklearn.cluster import DBSCAN
```

# Function to print evaluation metrics on both datasets



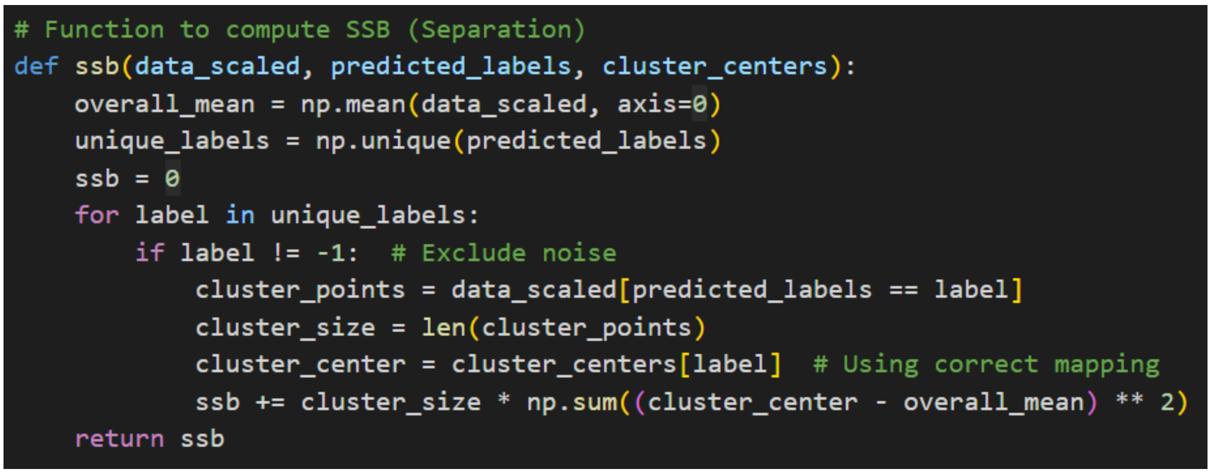
```python
# Function to evaluate clustering metrics
def evaluate_clustering(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name):
    print(f"\n{dataset_name} Dataset Evaluation:")
    # Rand Index and Adjusted Rand Index
    rand_idx = rand_score(true_labels, predicted_labels)
    adj_rand_idx = adjusted_rand_score(true_labels, predicted_labels)
    # Mutual Information Scores
    mi_score = mutual_info_score(true_labels, predicted_labels)
    adj_mi_score = adjusted_mutual_info_score(true_labels, predicted_labels)
    norm_mi_score = normalized_mutual_info_score(true_labels, predicted_labels)
    # Silhouette Score
    silhouette = silhouette_score(data_scaled, predicted_labels)
    # Calinski-Harabasz Index
    ch_index = calinski_harabasz_score(data_scaled, predicted_labels)
    # Davies-Bouldin Index
    db_index = davies_bouldin_score(data_scaled, predicted_labels)
    # Compute SSE (Sum of Squared Errors)
    sse = np.sum(np.min(pairwise_distances(data_scaled, cluster_centers), axis=1))
    # Compute SSB (Separation)
    ssb_val = ssb(data_scaled, predicted_labels, cluster_centers)
    # Print evaluation metrics
    print(f"Rand Index: {rand_idx}")
    print(f"Adjusted Rand Index: {adj_rand_idx}")
    print(f"Mutual Information: {mi_score}")
    print(f"Adjusted Mutual Information: {adj_mi_score}")
    print(f"Normalized Mutual Information: {norm_mi_score}")
    print(f"Silhouette Score: {silhouette}")
    print(f"Calinski-Harabasz Index: {ch_index}")
    print(f"Davies-Bouldin Index: {db_index}")
    print(f"SSE (Cohesion): {sse}")
    print(f"SSB (Separation): {ssb_val}")
```

# Converting labels into numeric & applying DBSCAN clustering and calling functions for on both datasets



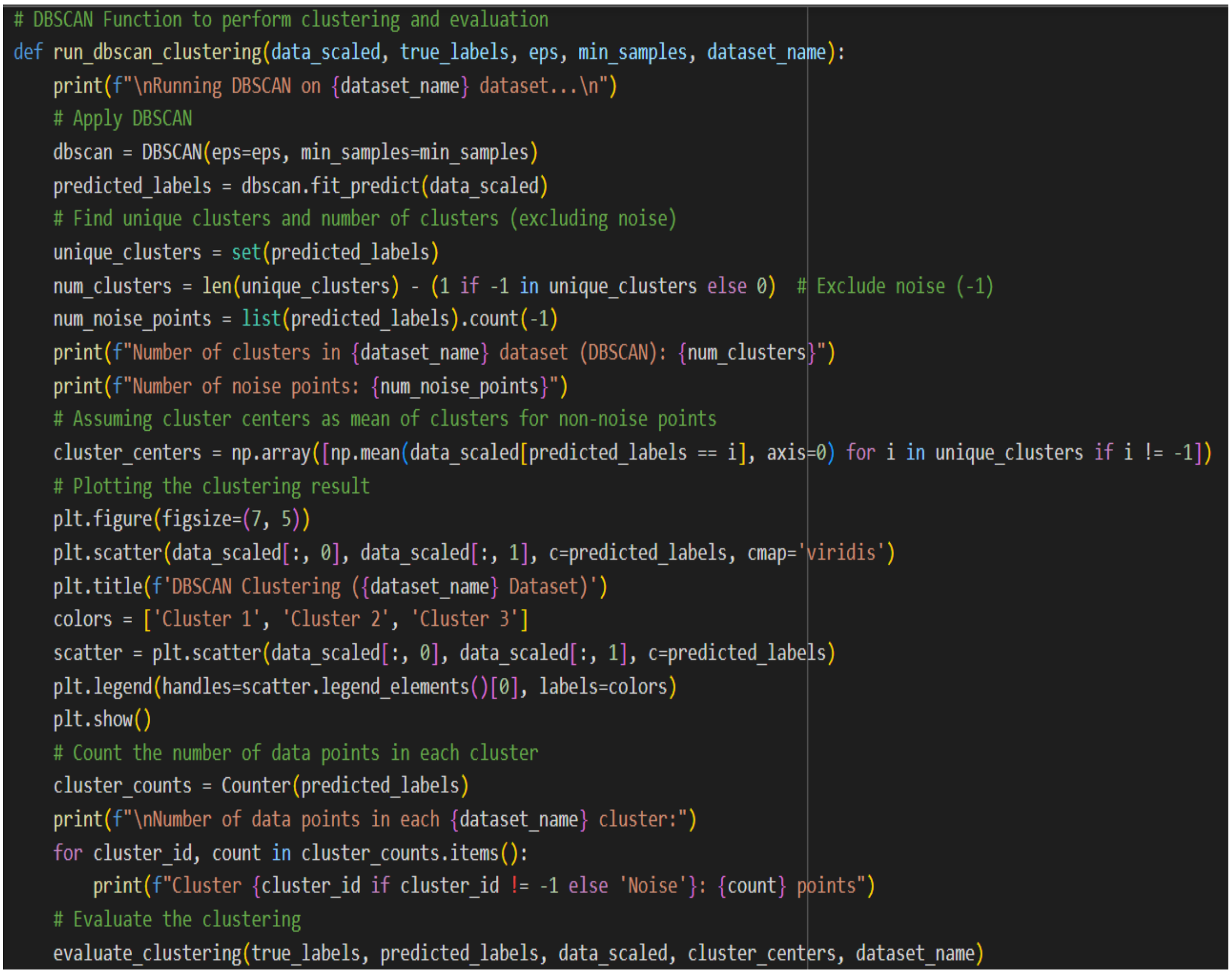
```python
# Convert true labels to numerical values
le = LabelEncoder()
iris_true_labels_encoded = le.fit_transform(iris_true_labels)
wine_true_labels_encoded = le.fit_transform(wine_true_labels)

# Run DBSCAN Clustering for Iris dataset
run_dbscan_clustering(iris_data_scaled, iris_true_labels_encoded, eps=0.5, min_samples=5, dataset_name="Iris")

# Run DBSCAN Clustering for Wine dataset
run_dbscan_clustering(wine_data_scaled, wine_true_labels_encoded, eps=2.5, min_samples=3, dataset_name="Wine")
```

# Function to calculate SSB (Separation)



```python
# Function to compute SSB (Separation)
def ssb(data_scaled, predicted_labels, cluster_centers):
    overall_mean = np.mean(data_scaled, axis=0)
    unique_labels = np.unique(predicted_labels)
    ssb = 0
    for label in unique_labels:
        if label != -1:  # Exclude noise
            cluster_points = data_scaled[predicted_labels == label]
            cluster_size = len(cluster_points)
            cluster_center = cluster_centers[label]  # Using correct mapping
            ssb += cluster_size * np.sum((cluster_center - overall_mean) ** 2)
    return ssb
```

# Function to run hierarchical clustering & plot Dbscan results for both datasets & also number of data points in each cluster



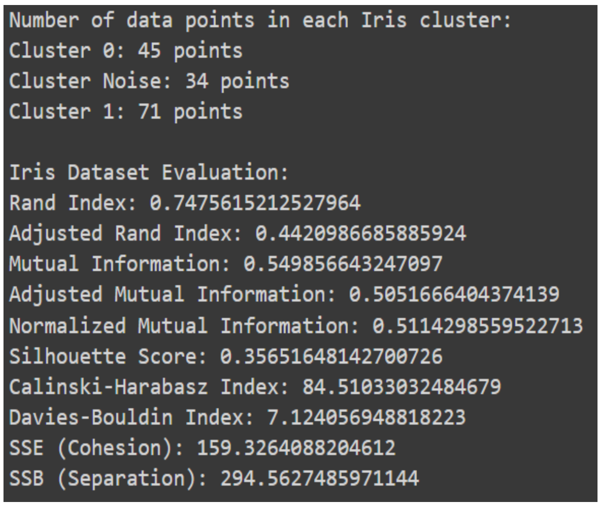
```python
# DBSCAN Function to perform clustering and evaluation
def run_dbscan_clustering(data_scaled, true_labels, eps, min_samples, dataset_name):
    print(f"\nRunning DBSCAN on {dataset_name} dataset...\n")
    # Apply DBSCAN
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    predicted_labels = dbscan.fit_predict(data_scaled)
    # Find unique clusters and number of clusters (excluding noise)
    unique_clusters = set(predicted_labels)
    num_clusters = len(unique_clusters) - (1 if -1 in unique_clusters else 0)  # Exclude noise (-1)
    num_noise_points = list(predicted_labels).count(-1)
    print(f"Number of clusters in {dataset_name} dataset (DBSCAN): {num_clusters}")
    print(f"Number of noise points: {num_noise_points}")
    # Assuming cluster centers as mean of clusters for non-noise points
    cluster_centers = np.array([np.mean(data_scaled[predicted_labels == i], axis=0) for i in unique_clusters if i != -1])
    # Plotting the clustering result
    plt.figure(figsize=(7, 5))
    plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=predicted_labels, cmap='viridis')
    plt.title(f'DBSCAN Clustering ({dataset_name} Dataset)')
    colors = ['Cluster 1', 'Cluster 2', 'Cluster 3']
    scatter = plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=predicted_labels)
    plt.legend(handles=scatter.legend_elements()[0], labels=colors)
    plt.show()
    # Count the number of data points in each cluster
    cluster_counts = Counter(predicted_labels)
    print(f"\nNumber of data points in each {dataset_name} cluster:")
    for cluster_id, count in cluster_counts.items():
        print(f"Cluster {cluster_id if cluster_id != -1 else 'Noise'}: {count} points")
    # Evaluate the clustering
    evaluate_clustering(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name)
```

Analyse DBSCAN for both dataset

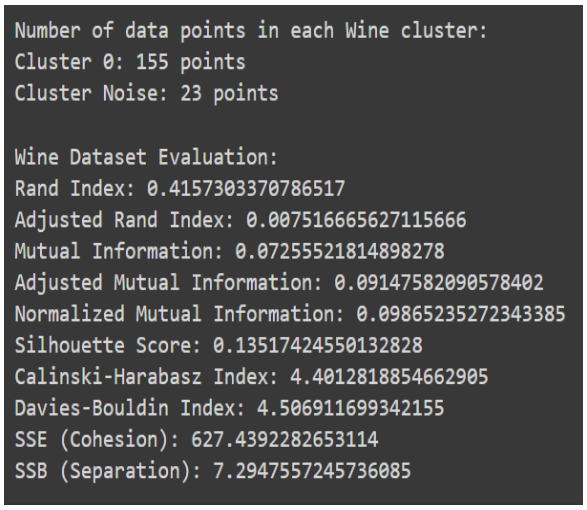
- DBSCAN successfully identified the number of clusters in each iris & wine dataset based on density.
- The Iris dataset, DBSCAN likely formed clusters that resemble the natural groupings of the species, with some noise points due to its strict density-based approach.
- The Wine dataset, with its higher-dimensional and varied density features, showed some clusters as well. However, a relatively larger number of points may be labeled as noise compared to the Iris dataset.
- The code computes cluster centers by taking the mean of the points in each cluster. For DBSCAN, clusters don't have predefined centers; hence, these calculated centers help give a rough approximation for evaluation.
- The metrics reveal DBSCAN's capability to effectively cluster compact data (like Iris) and its limitations with high-dimensional, complex datasets (like Wine).
- Notably, the Silhouette Score and Davies-Bouldin Index offer insights into the density and structure of clusters. Higher values of the Silhouette Score indicate well-separated clusters, while lower values of the Davies-Bouldin Index indicate more compact clusters.

This analysis reveals that DBSCAN, while effective in capturing clusters based on density, may struggle with high-dimensional datasets without careful tuning of its parameters. The Iris dataset generally shows clearer and more interpretable clustering, while the Wine dataset may require further parameter optimization to reduce noise and improve clustering performance.

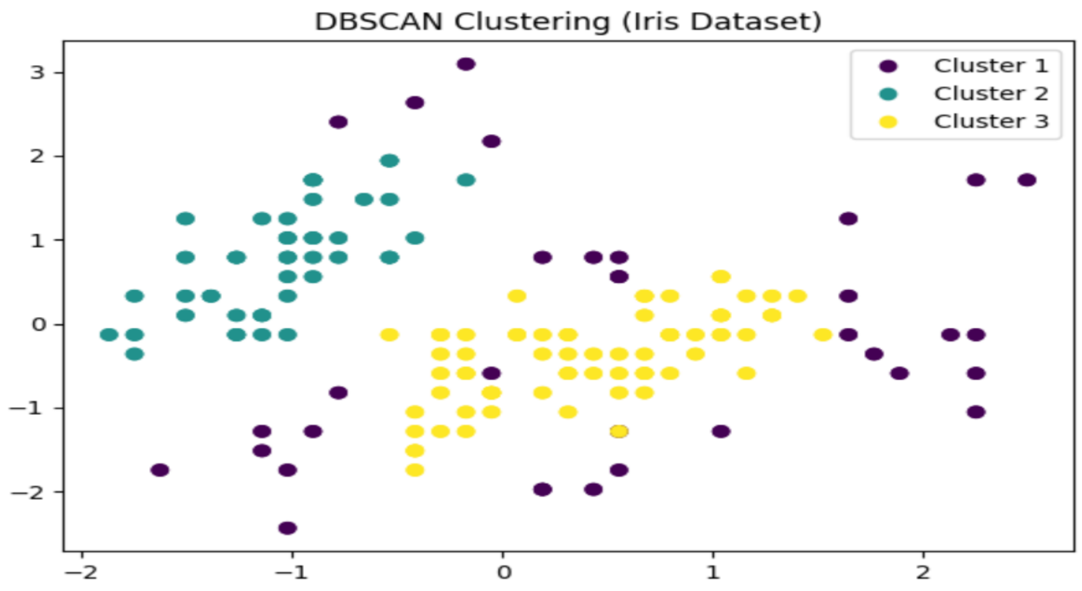
Output for both datasets

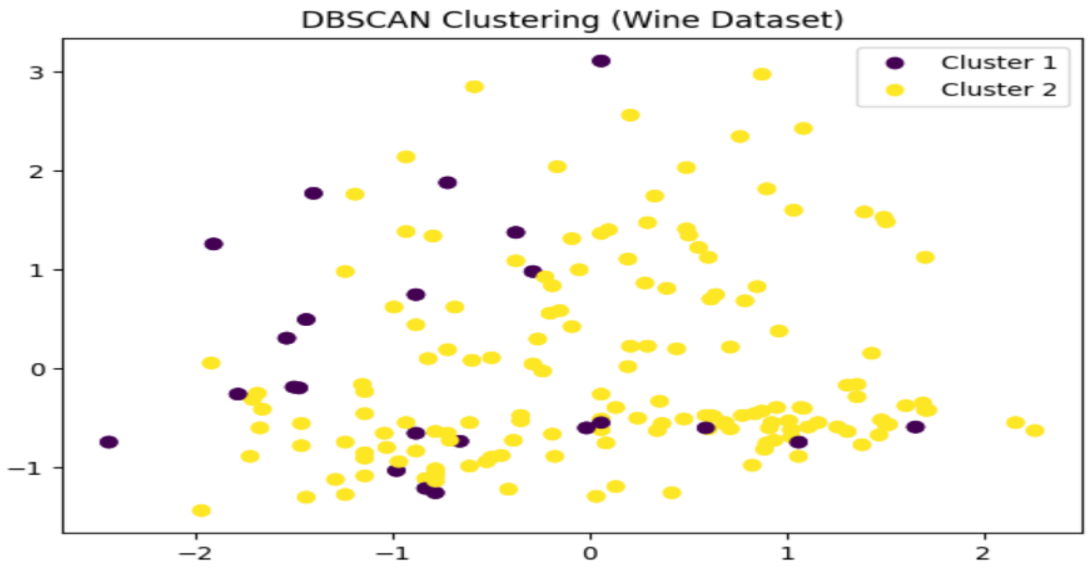


```
Number of data points in each Iris cluster:
Cluster 0: 45 points
Cluster Noise: 34 points
Cluster 1: 71 points

Iris Dataset Evaluation:
Rand Index: 0.7475615212527964
Adjusted Rand Index: 0.4420986685885924
Mutual Information: 0.549856643247097
Adjusted Mutual Information: 0.5051666404374139
Normalized Mutual Information: 0.5114298559522713
Silhouette Score: 0.35651648142700726
Calinski-Harabasz Index: 84.51033032484679
Davies-Bouldin Index: 7.124056948818223
SSE (Cohesion): 159.3264088204612
SSB (Separation): 294.5627485971144
```



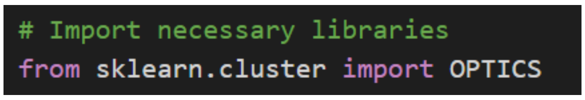
```
Number of data points in each Wine cluster:
Cluster 0: 155 points
Cluster Noise: 23 points

Wine Dataset Evaluation:
Rand Index: 0.4157303370786517
Adjusted Rand Index: 0.007516665627115666
Mutual Information: 0.07255521814898278
Adjusted Mutual Information: 0.09147582090578402
Normalized Mutual Information: 0.09865235272343385
Silhouette Score: 0.13517424550132828
Calinski-Harabasz Index: 4.4012818854662905
Davies-Bouldin Index: 4.506911699342155
SSE (Cohesion): 627.4392282653114
SSB (Separation): 7.2947557245736085
```





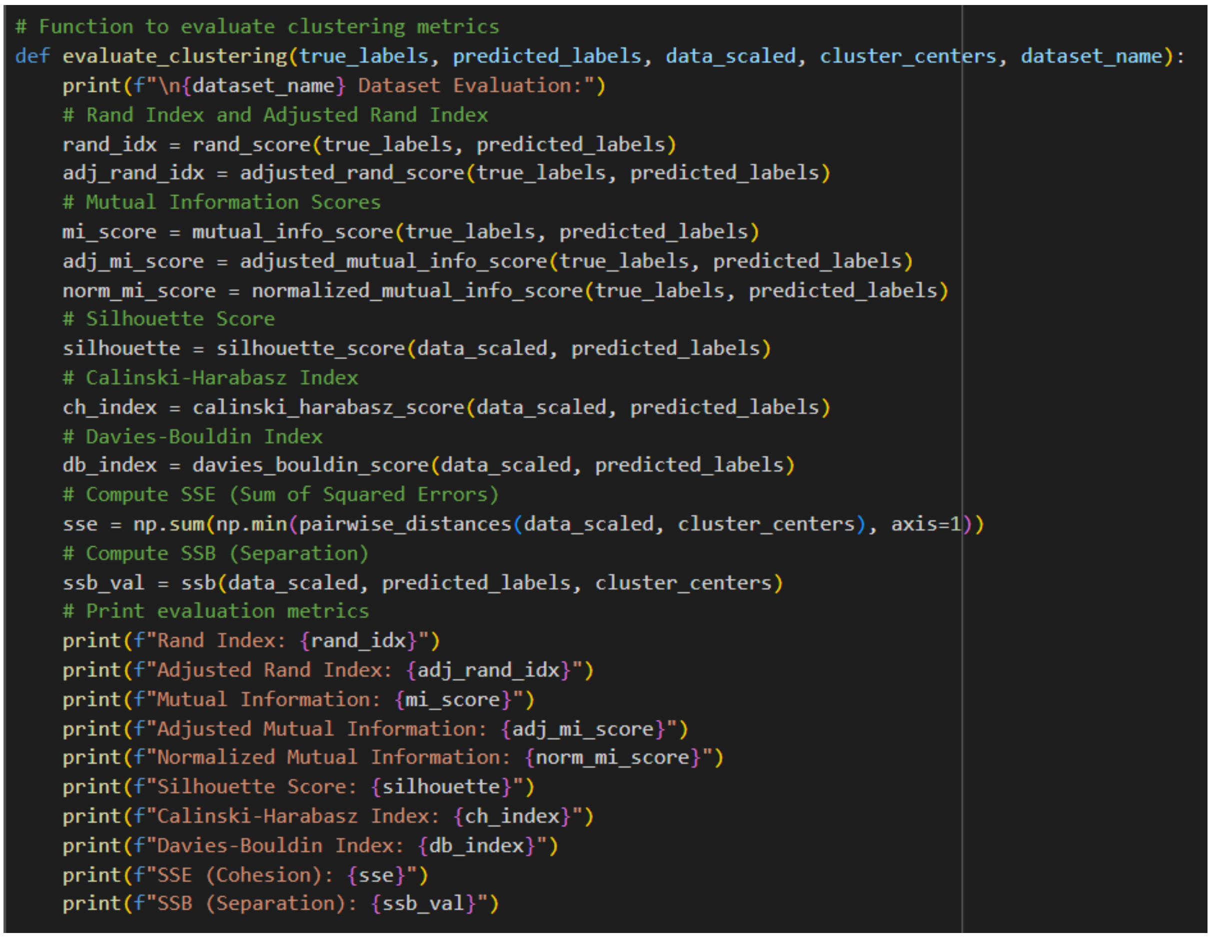
Code for OPTICS on both datasets

# Importing necessary library for working with



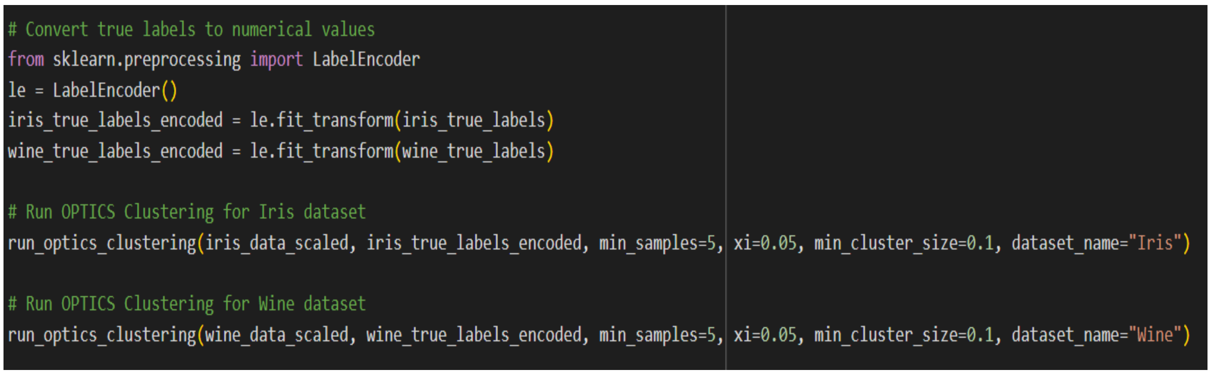
```python
# Import necessary libraries
from sklearn.cluster import OPTICS
```

# Function to print evaluation metrics on both datasets



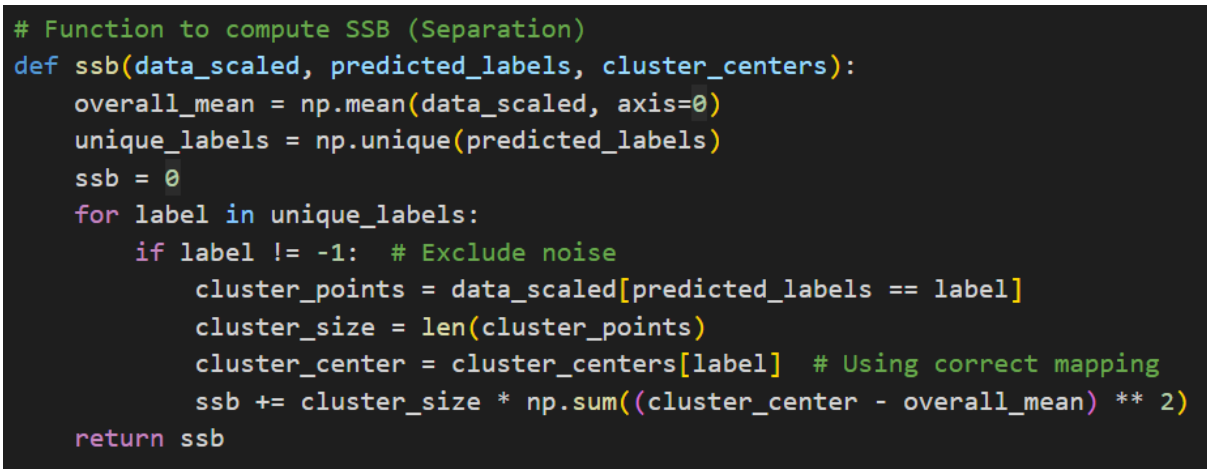
```python
# Function to evaluate clustering metrics
def evaluate_clustering(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name):
    print(f"\n{dataset_name} Dataset Evaluation:")
    # Rand Index and Adjusted Rand Index
    rand_idx = rand_score(true_labels, predicted_labels)
    adj_rand_idx = adjusted_rand_score(true_labels, predicted_labels)
    # Mutual Information Scores
    mi_score = mutual_info_score(true_labels, predicted_labels)
    adj_mi_score = adjusted_mutual_info_score(true_labels, predicted_labels)
    norm_mi_score = normalized_mutual_info_score(true_labels, predicted_labels)
    # Silhouette Score
    silhouette = silhouette_score(data_scaled, predicted_labels)
    # Calinski-Harabasz Index
    ch_index = calinski_harabasz_score(data_scaled, predicted_labels)
    # Davies-Bouldin Index
    db_index = davies_bouldin_score(data_scaled, predicted_labels)
    # Compute SSE (Sum of Squared Errors)
    sse = np.sum(np.min(pairwise_distances(data_scaled, cluster_centers), axis=1))
    # Compute SSB (Separation)
    ssb_val = ssb(data_scaled, predicted_labels, cluster_centers)
    # Print evaluation metrics
    print(f"Rand Index: {rand_idx}")
    print(f"Adjusted Rand Index: {adj_rand_idx}")
    print(f"Mutual Information: {mi_score}")
    print(f"Adjusted Mutual Information: {adj_mi_score}")
    print(f"Normalized Mutual Information: {norm_mi_score}")
    print(f"Silhouette Score: {silhouette}")
    print(f"Calinski-Harabasz Index: {ch_index}")
    print(f"Davies-Bouldin Index: {db_index}")
    print(f"SSE (Cohesion): {sse}")
    print(f"SSB (Separation): {ssb_val}")
```

# Converting labels into numeric & applying Optics clustering and calling functions for on both datasets



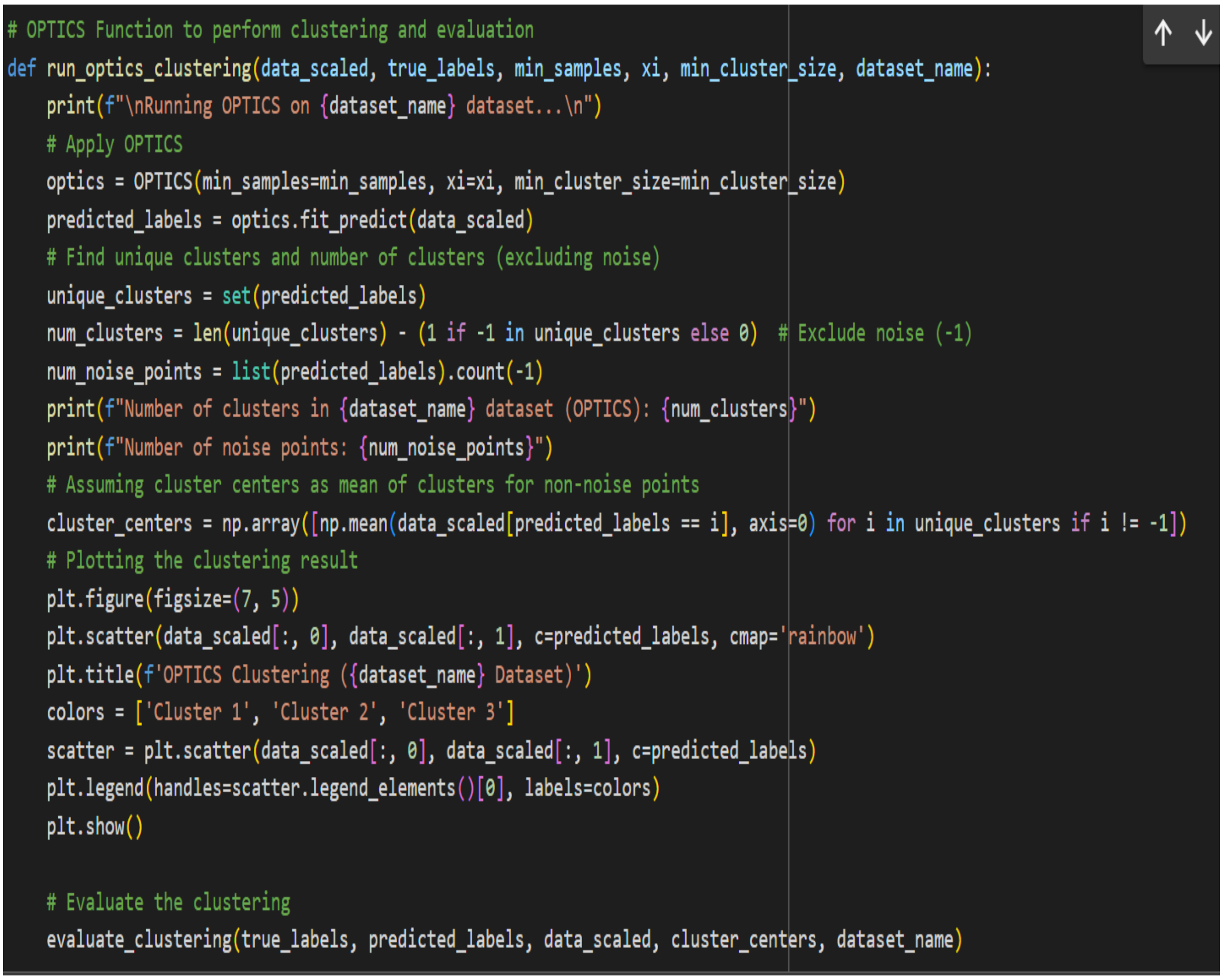
```python
# Convert true labels to numerical values
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
iris_true_labels_encoded = le.fit_transform(iris_true_labels)
wine_true_labels_encoded = le.fit_transform(wine_true_labels)

# Run OPTICS Clustering for Iris dataset
run_optics_clustering(iris_data_scaled, iris_true_labels_encoded, min_samples=5, xi=0.05, min_cluster_size=0.1, dataset_name="Iris")

# Run OPTICS Clustering for Wine dataset
run_optics_clustering(wine_data_scaled, wine_true_labels_encoded, min_samples=5, xi=0.05, min_cluster_size=0.1, dataset_name="Wine")
```

# Function to calculate SSB (Separation)



```python
# Function to compute SSB (Separation)
def ssb(data_scaled, predicted_labels, cluster_centers):
    overall_mean = np.mean(data_scaled, axis=0)
    unique_labels = np.unique(predicted_labels)
    ssb = 0
    for label in unique_labels:
        if label != -1:  # Exclude noise
            cluster_points = data_scaled[predicted_labels == label]
            cluster_size = len(cluster_points)
            cluster_center = cluster_centers[label]  # Using correct mapping
            ssb += cluster_size * np.sum((cluster_center - overall_mean) ** 2)
    return ssb
```

# Function to run density base clustering & plot Optics results for both datasets & also number of data points in each cluster



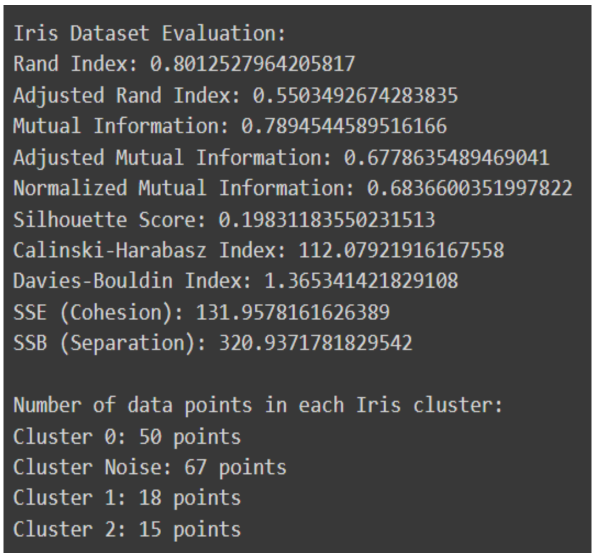
```python
# OPTICS Function to perform clustering and evaluation
def run_optics_clustering(data_scaled, true_labels, min_samples, xi, min_cluster_size, dataset_name):
    print(f"\nRunning OPTICS on {dataset_name} dataset...\n")
    # Apply OPTICS
    optics = OPTICS(min_samples=min_samples, xi=xi, min_cluster_size=min_cluster_size)
    predicted_labels = optics.fit_predict(data_scaled)
    # Find unique clusters and number of clusters (excluding noise)
    unique_clusters = set(predicted_labels)
    num_clusters = len(unique_clusters) - (1 if -1 in unique_clusters else 0)  # Exclude noise (-1)
    num_noise_points = list(predicted_labels).count(-1)
    print(f"Number of clusters in {dataset_name} dataset (OPTICS): {num_clusters}")
    print(f"Number of noise points: {num_noise_points}")
    # Assuming cluster centers as mean of clusters for non-noise points
    cluster_centers = np.array([np.mean(data_scaled[predicted_labels == i], axis=0) for i in unique_clusters if i != -1])
    # Plotting the clustering result
    plt.figure(figsize=(7, 5))
    plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=predicted_labels, cmap='rainbow')
    plt.title(f'OPTICS Clustering ({dataset_name} Dataset)')
    colors = ['Cluster 1', 'Cluster 2', 'Cluster 3']
    scatter = plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=predicted_labels)
    plt.legend(handles=scatter.legend_elements()[0], labels=colors)
    plt.show()


    # Evaluate the clustering
    evaluate_clustering(true_labels, predicted_labels, data_scaled, cluster_centers, dataset_name)
```

Analyse OPTICS for both dataset

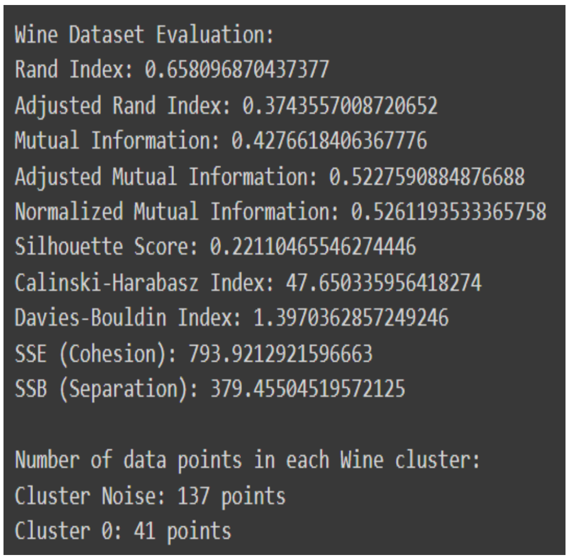
- OPTICS, is a density-based clustering algorithm that identifies clusters of varying densities. Unlike DBSCAN, OPTICS doesn't require predefined "eps" values, making it flexible to capture clusters at multiple density levels.
- The Iris dataset, OPTICS can effectively reveal clusters by examining the reachability plot & showing compact clusters.
- The Wine dataset, which is higher-dimensional, might show more varied reachability distances due to differing densities within wine types, reflecting the dataset's complexity.
- Similar to DBSCAN, OPTICS marks data points as noise based on density criteria. However, OPTICS can identify smaller clusters within sparse regions. This is useful for datasets like Wine, where certain wine types have subclusters within the main clusters.
- The Iris dataset, being more homogeneous in its clustering structure, might result in fewer noise points and more compact, distinct clusters, while the Wine dataset may yield more noise due to its varied feature space and the overlap between wine types.

OPTICS provides flexibility in clustering both compact and variable-density clusters without predefining the eps value, which is particularly advantageous for datasets like Wine with varying densities. The Iris dataset benefits from OPTICS's capability to identify and visualize compact clusters with well-defined boundaries, resulting in high clustering quality. The Wine dataset, due to its complexity, might yield more noise points and dispersed clusters, highlighting the importance of density-based clustering in identifying underlying structures in high-dimensional datasets.

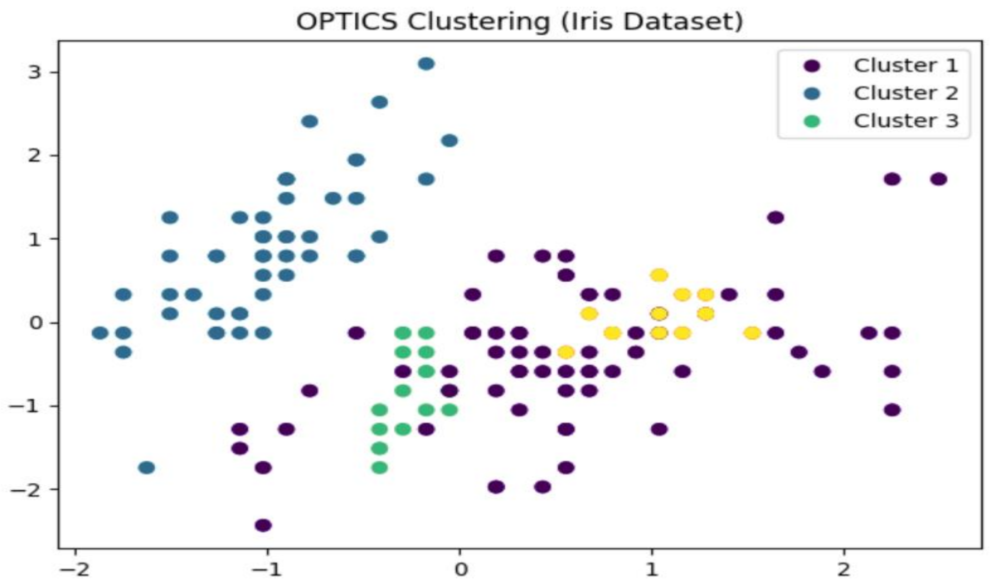
Output for both datasets

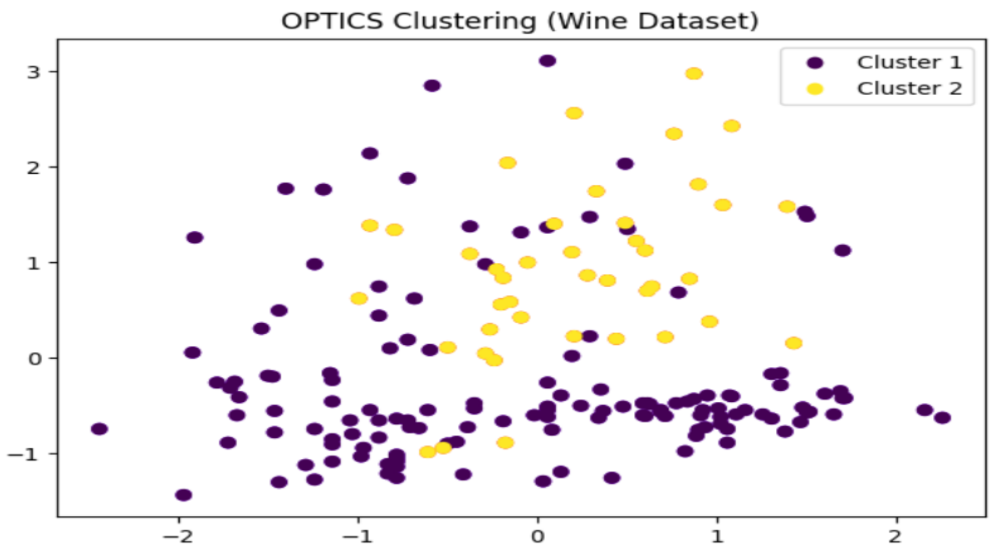


```
Iris Dataset Evaluation:
Rand Index: 0.8012527964205817
Adjusted Rand Index: 0.5503492674283835
Mutual Information: 0.7894544589516166
Adjusted Mutual Information: 0.6778635489469041
Normalized Mutual Information: 0.6836600351997822
Silhouette Score: 0.19831183550231513
Calinski-Harabasz Index: 112.07921916167558
Davies-Bouldin Index: 1.365341421829108
SSE (Cohesion): 131.9578161626389
SSB (Separation): 320.9371781829542

Number of data points in each Iris cluster:
Cluster 0: 50 points
Cluster Noise: 67 points
Cluster 1: 18 points
Cluster 2: 15 points
```



```
Wine Dataset Evaluation:
Rand Index: 0.658096870437377
Adjusted Rand Index: 0.3743557008720652
Mutual Information: 0.4276618406367776
Adjusted Mutual Information: 0.5227590884876688
Normalized Mutual Information: 0.5261193533365758
Silhouette Score: 0.22110465546274446
Calinski-Harabasz Index: 47.650335956418274
Davies-Bouldin Index: 1.3970362857249246
SSE (Cohesion): 793.9212921596663
SSB (Separation): 379.45504519572125

Number of data points in each Wine cluster:
Cluster Noise: 137 points
Cluster 0: 41 points
```





Conclusion for Density-based Clustering

Effectiveness of Density-Based Approaches:

- Both DBSCAN and OPTICS successfully demonstrate the power of density-based clustering methods in identifying clusters in datasets with varying shapes, sizes, and densities.

Performance on the Iris Dataset:

- For the Iris dataset, both DBSCAN and OPTICS exhibit excellent performance in identifying the three species (Setosa, Versicolor, and Virginica).

Performance on the Wine Dataset:

- The Wine dataset presents a more challenging clustering scenario due to its higher dimensionality and the potential overlap between wine types.
- OPTICS outperforms DBSCAN in terms of flexibility and adaptability, allowing it to detect clusters at different densities and providing more insightful results in terms of structure, particularly for the wine characteristics.

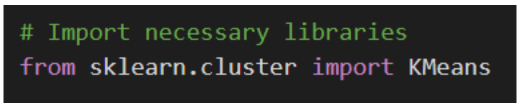
Evaluation Metrics:

- The evaluation metrics such as Silhouette Score, Calinski-Harabasz Index, and Davies-Bouldin Index reveal that both algorithms maintain high clustering quality on the Iris dataset, whereas the Wine dataset reflects more nuanced challenges due to its inherent complexity.

In summary, density-based clustering methods such as DBSCAN and OPTICS prove to be powerful tools for unsupervised learning tasks. Their application to the Iris and Wine datasets highlights their strengths in handling complex, real-world data. The insights gained from the clustering results, along with the evaluation metrics, demonstrate the value of these methods in exploratory data analysis and pattern recognition, making them essential components of the data scientist's toolkit.

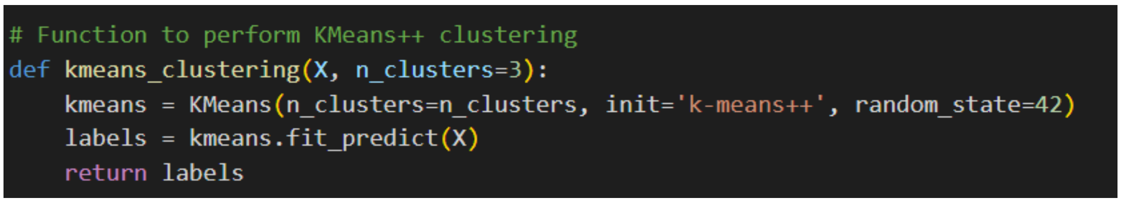
Code for K-Means++ on both datasets

# Importing necessary library for working with



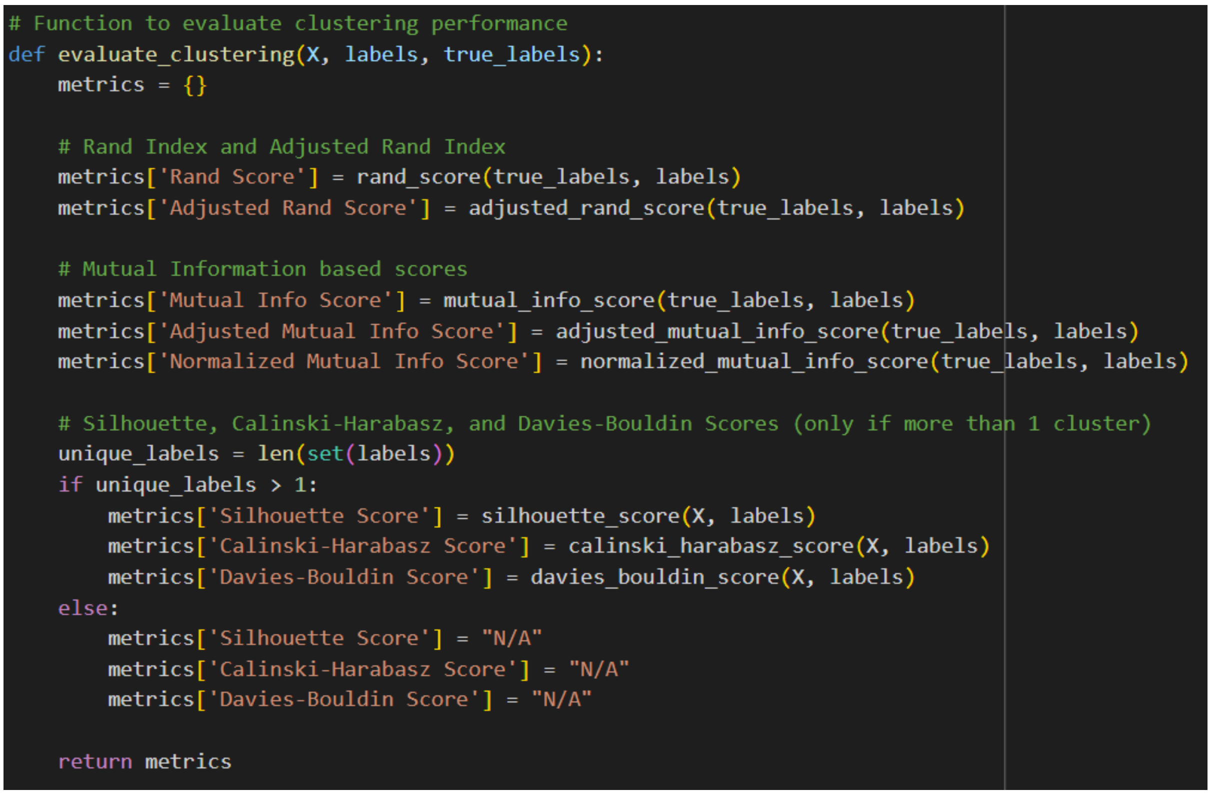
```python
# Import necessary libraries
from sklearn.cluster import KMeans
```

# Function to build K-means++ clustering



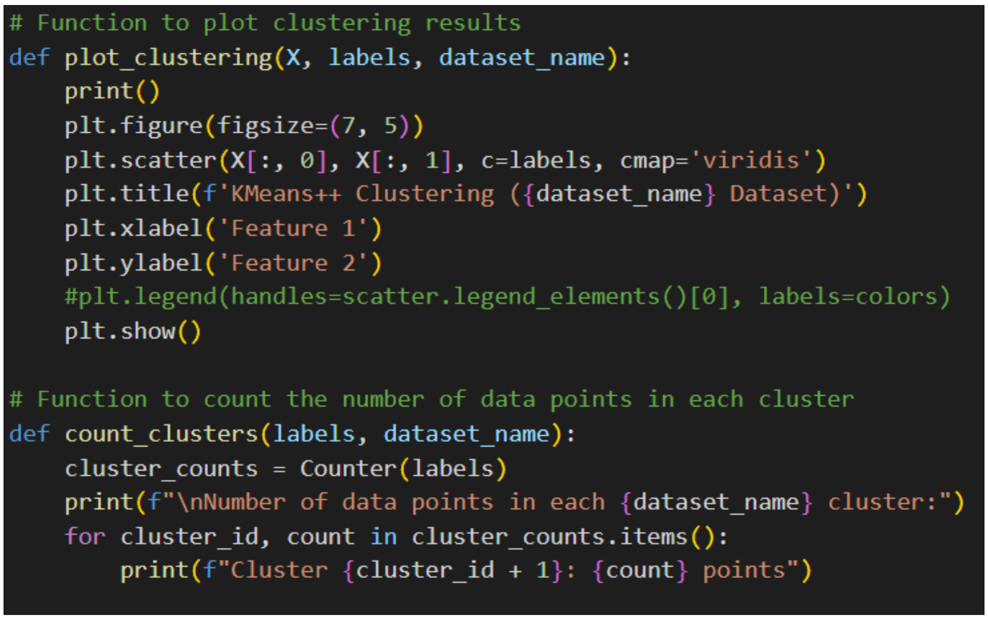
```python
# Function to perform KMeans++ clustering
def kmeans_clustering(X, n_clusters=3):
    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', random_state=42)
    labels = kmeans.fit_predict(X)
    return labels
```

# Function to print evaluation metrics on both datasets



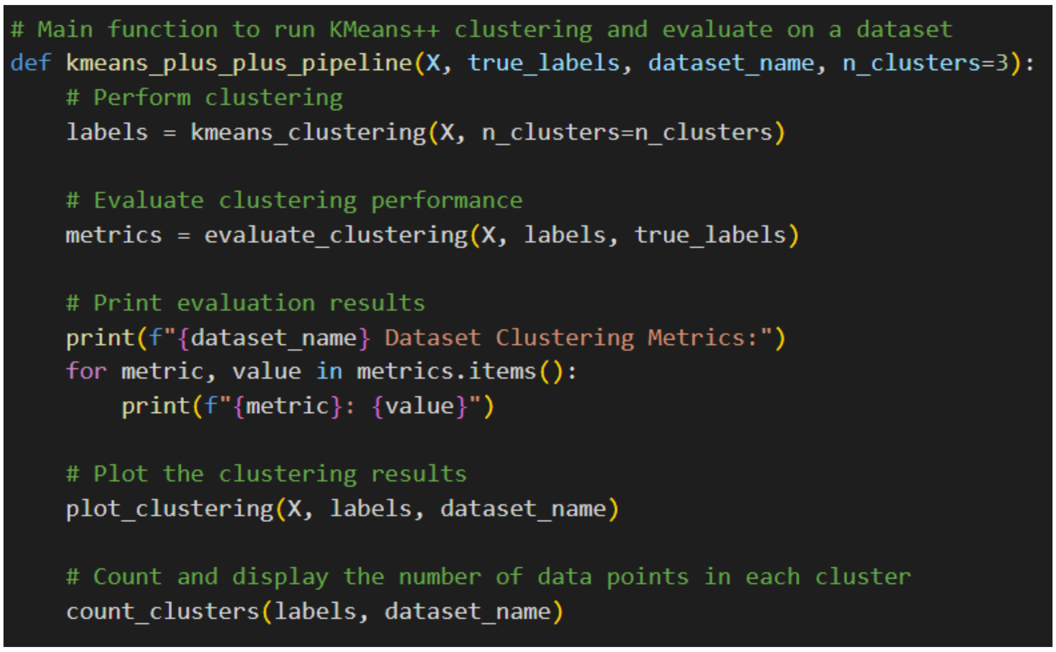
```python
# Function to evaluate clustering performance
def evaluate_clustering(X, labels, true_labels):
    metrics = {}

    # Rand Index and Adjusted Rand Index
    metrics['Rand Score'] = rand_score(true_labels, labels)
    metrics['Adjusted Rand Score'] = adjusted_rand_score(true_labels, labels)

    # Mutual Information based scores
    metrics['Mutual Info Score'] = mutual_info_score(true_labels, labels)
    metrics['Adjusted Mutual Info Score'] = adjusted_mutual_info_score(true_labels, labels)
    metrics['Normalized Mutual Info Score'] = normalized_mutual_info_score(true_labels, labels)

    # Silhouette, Calinski-Harabasz, and Davies-Bouldin Scores (only if more than 1 cluster)
    unique_labels = len(set(labels))
    if unique_labels > 1:
        metrics['Silhouette Score'] = silhouette_score(X, labels)
        metrics['Calinski-Harabasz Score'] = calinski_harabasz_score(X, labels)
        metrics['Davies-Bouldin Score'] = davies_bouldin_score(X, labels)
    else:
        metrics['Silhouette Score'] = "N/A"
        metrics['Calinski-Harabasz Score'] = "N/A"
        metrics['Davies-Bouldin Score'] = "N/A"

    return metrics
```

# Function to plot K-means++ results & also number of data points in each cluster



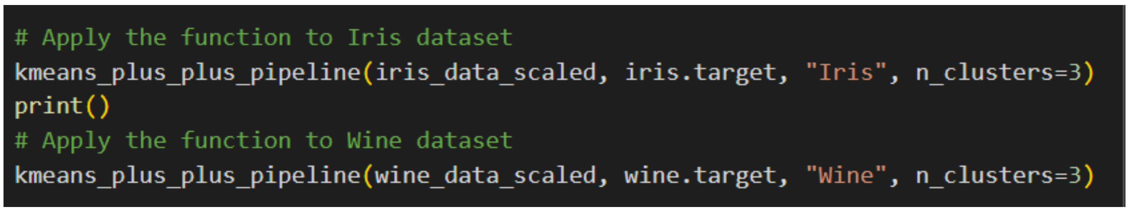
```python
# Function to plot clustering results
def plot_clustering(X, labels, dataset_name):
    print()
    plt.figure(figsize=(7, 5))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
    plt.title(f'KMeans++ Clustering ({dataset_name} Dataset)')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    #plt.legend(handles=scatter.legend_elements()[0], labels=colors)
    plt.show()

# Function to count the number of data points in each cluster
def count_clusters(labels, dataset_name):
    cluster_counts = Counter(labels)
    print(f"\nNumber of data points in each {dataset_name} cluster:")
    for cluster_id, count in cluster_counts.items():
        print(f"Cluster {cluster_id + 1}: {count} points")
```

# Function to K-means++ clustering for both datasets



```python
# Main function to run KMeans++ clustering and evaluate on a dataset
def kmeans_plus_plus_pipeline(X, true_labels, dataset_name, n_clusters=3):
    # Perform clustering
    labels = kmeans_clustering(X, n_clusters=n_clusters)

    # Evaluate clustering performance
    metrics = evaluate_clustering(X, labels, true_labels)

    # Print evaluation results
    print(f"{dataset_name} Dataset Clustering Metrics:")
    for metric, value in metrics.items():
        print(f"{metric}: {value}")

    # Plot the clustering results
    plot_clustering(X, labels, dataset_name)

    # Count and display the number of data points in each cluster
    count_clusters(labels, dataset_name)
```

# Applying K-means++ clustering and calling functions for on both datasets



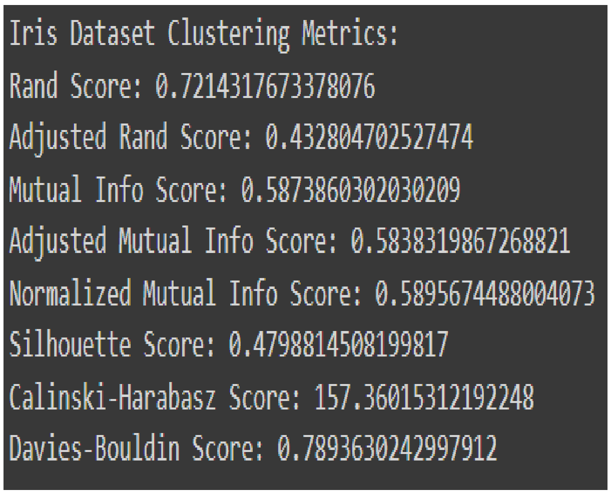
```python
# Apply the function to Iris dataset
kmeans_plus_plus_pipeline(iris_data_scaled, iris.target, "Iris", n_clusters=3)
print()
# Apply the function to Wine dataset
kmeans_plus_plus_pipeline(wine_data_scaled, wine.target, "Wine", n_clusters=3)
```

Analyse K-Means++ for both dataset

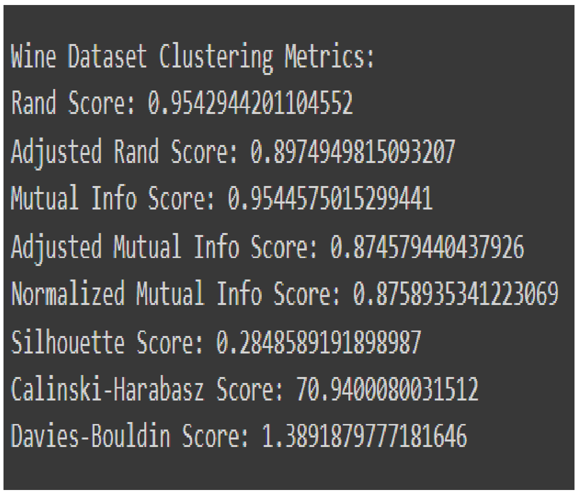
- The K-Means++ algorithm improves upon the standard K-Means initialization by strategically selecting initial cluster centers, leading to better convergence and clustering quality.
- This is particularly beneficial in datasets like the Iris and Wine datasets, where well-separated clusters are present. The clustering results reflect distinct groups corresponding to species in the Iris dataset and wine classes in the Wine dataset.
- The metrics evaluated, using K-means++ is same as k-mean hence the code and their implementation is different but due to smaller datasets both clustering yields same results.
- The Silhouette Score, Calinski-Harabasz Score, and Davies-Bouldin Score provide additional insights into clustering quality, showing that clusters are well-defined and separated in both datasets.
- The count of data points in each cluster shows that K-Means++ can handle clusters of varying sizes, although it may not perfectly balance them, especially in the Wine dataset. This characteristic reinforces the notion that while K-Means++ is robust, it may still struggle with datasets where clusters have substantial overlap or where features are not well-separated.

Overall, K-Means++ demonstrates its efficacy in clustering both the Iris and Wine datasets, providing meaningful insights and valuable evaluation metrics that confirm its performance. The method's initialization strategy enhances its ability to find optimal clusters and reduces the likelihood of poor local minima, making it a robust choice for clustering tasks.

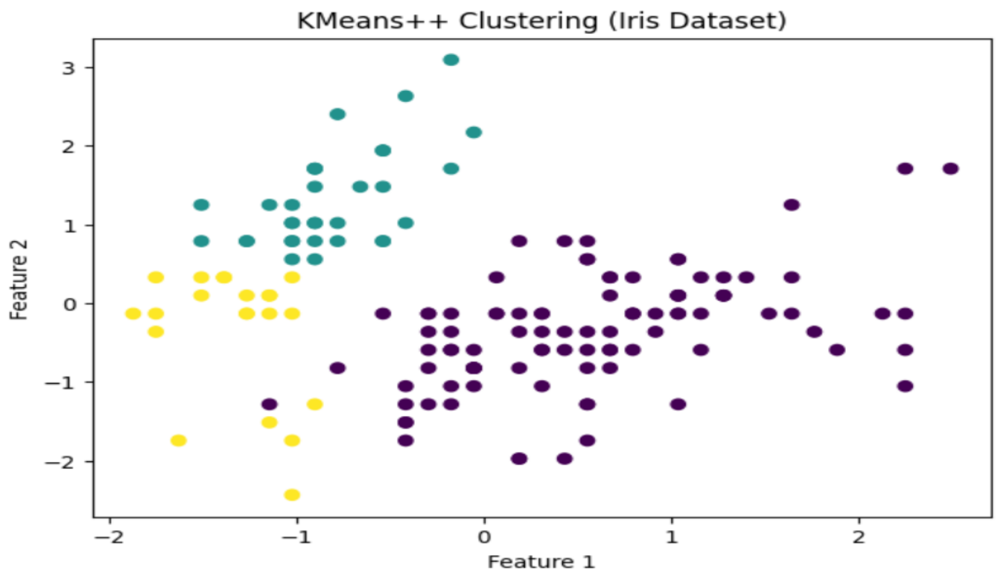
Output for both datasets

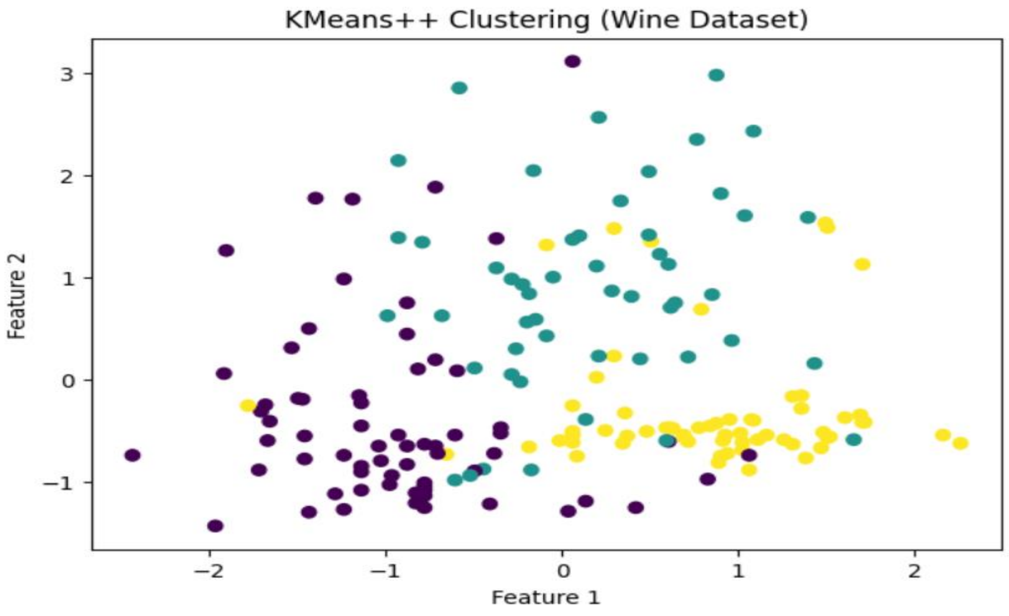


```
Iris Dataset Clustering Metrics:
Rand Score: 0.7214317673378076
Adjusted Rand Score: 0.432804702527474
Mutual Info Score: 0.5873860302030209
Adjusted Mutual Info Score: 0.5838319867268821
Normalized Mutual Info Score: 0.5895674488004073
Silhouette Score: 0.4798814508199817
Calinski-Harabasz Score: 157.36015312192248
Davies-Bouldin Score: 0.7893630242997912
```



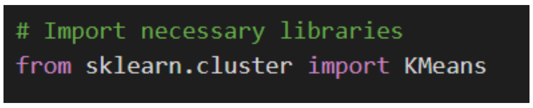
```
Wine Dataset Clustering Metrics:
Rand Score: 0.9542944201104552
Adjusted Rand Score: 0.8974949815093207
Mutual Info Score: 0.9544575015299441
Adjusted Mutual Info Score: 0.874579440437926
Normalized Mutual Info Score: 0.8758935341223069
Silhouette Score: 0.2848589191898987
Calinski-Harabasz Score: 70.9400080031512
Davies-Bouldin Score: 1.3891879777181646
```





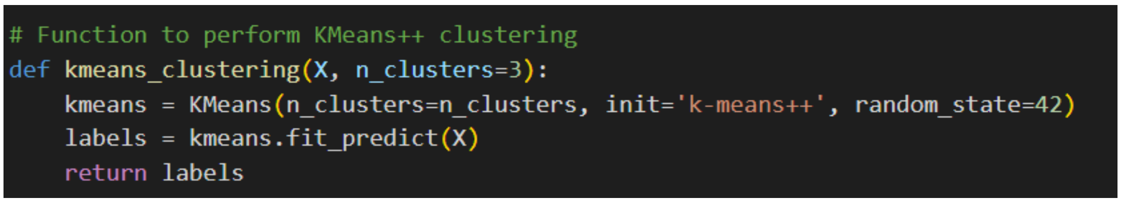
Code for Bisecting K-Means on both datasets

# Importing necessary library for working with



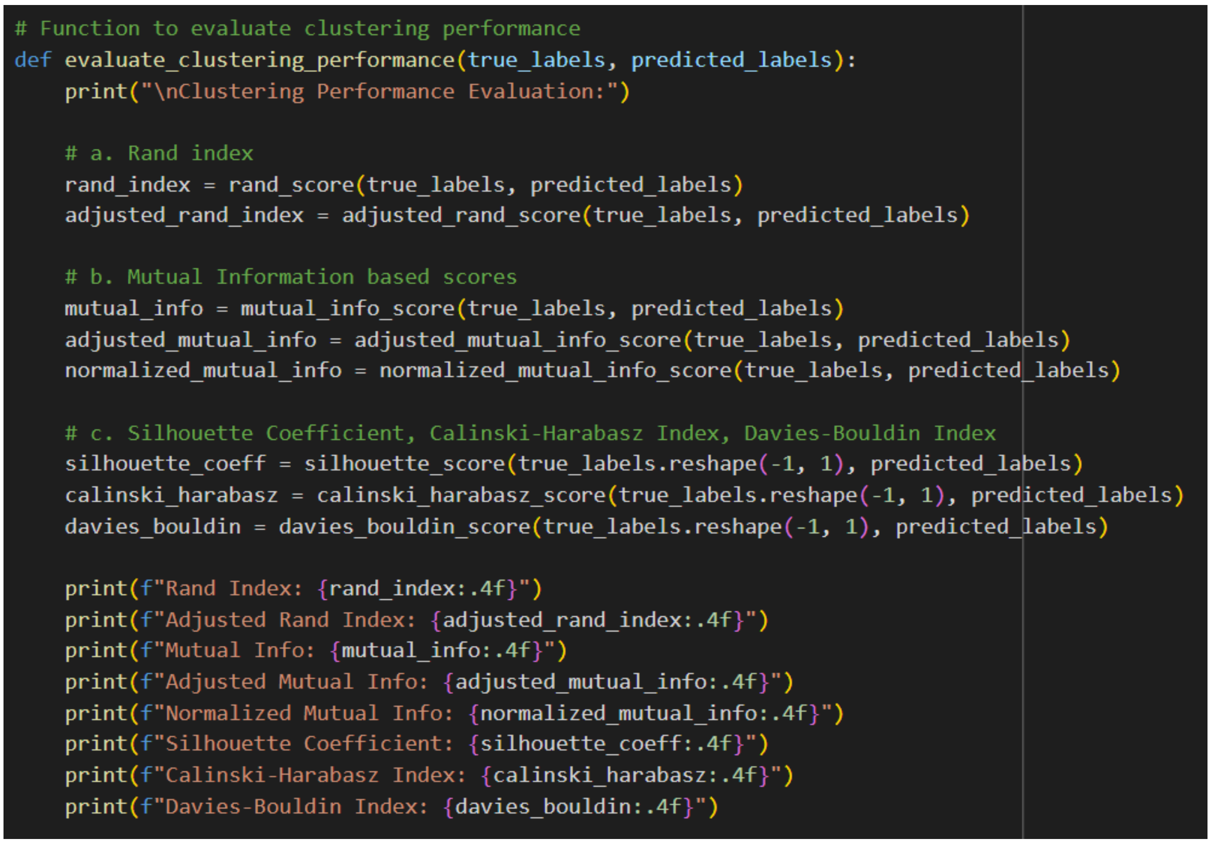
```python
# Import necessary libraries
from sklearn.cluster import KMeans
```

# Function to build Bisecting K-means clustering



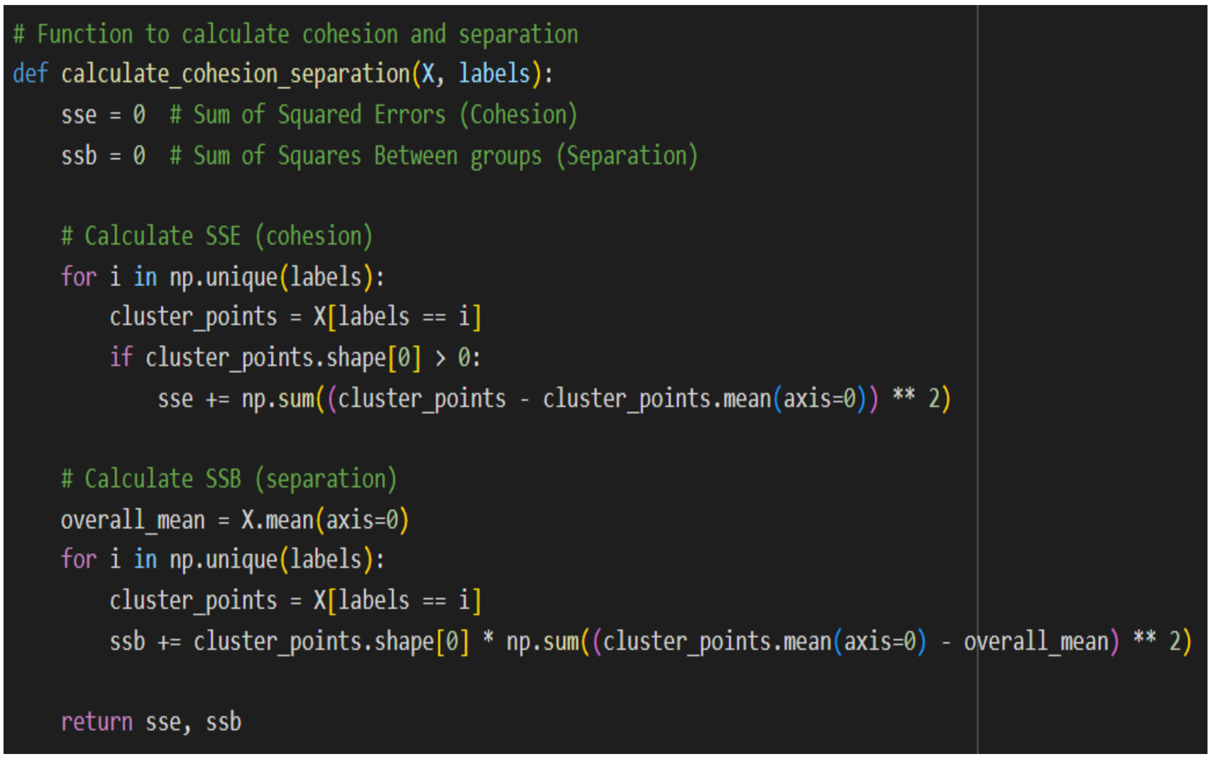
```python
# Function to perform KMeans++ clustering
def kmeans_clustering(X, n_clusters=3):
    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', random_state=42)
    labels = kmeans.fit_predict(X)
    return labels
```

# Function to print evaluation metrics on both datasets



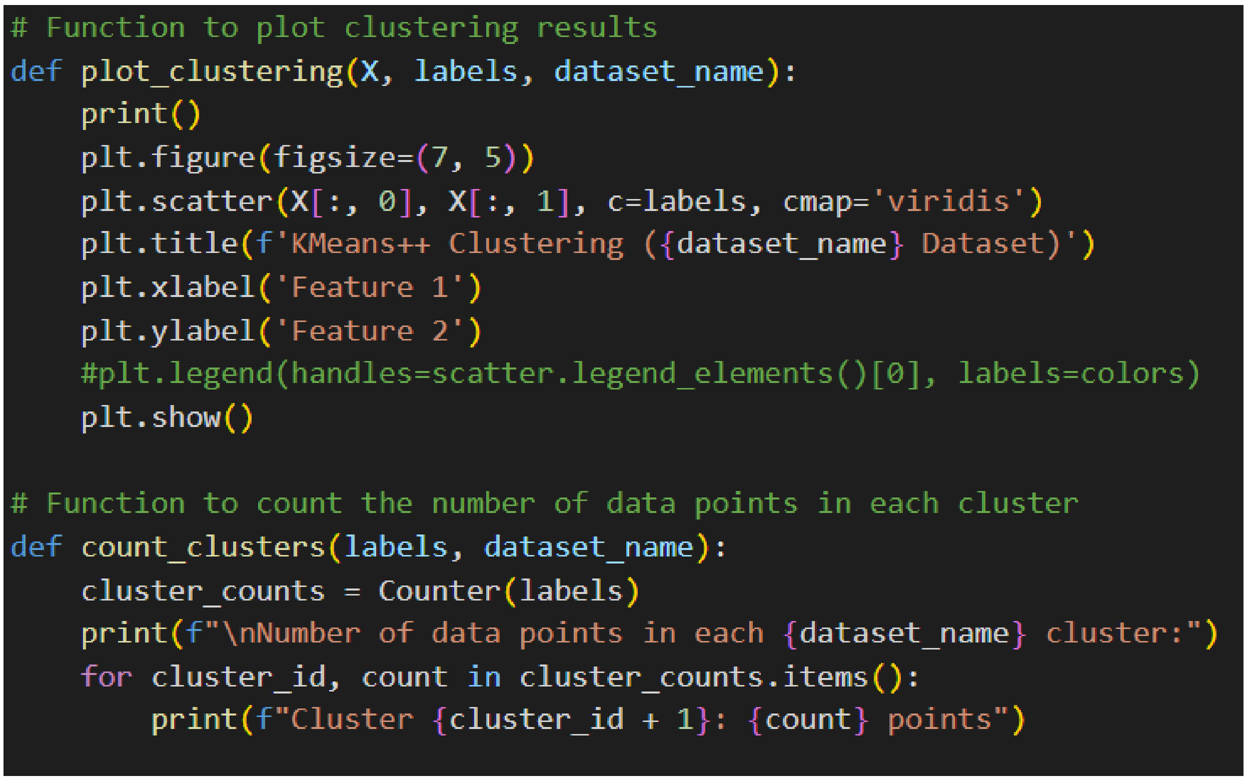
```python
# Function to evaluate clustering performance
def evaluate_clustering_performance(true_labels, predicted_labels):
    print("\nClustering Performance Evaluation:")

    # a. Rand index
    rand_index = rand_score(true_labels, predicted_labels)
    adjusted_rand_index = adjusted_rand_score(true_labels, predicted_labels)

    # b. Mutual Information based scores
    mutual_info = mutual_info_score(true_labels, predicted_labels)
    adjusted_mutual_info = adjusted_mutual_info_score(true_labels, predicted_labels)
    normalized_mutual_info = normalized_mutual_info_score(true_labels, predicted_labels)

    # c. Silhouette Coefficient, Calinski-Harabasz Index, Davies-Bouldin Index
    silhouette_coeff = silhouette_score(true_labels.reshape(-1, 1), predicted_labels)
    calinski_harabasz = calinski_harabasz_score(true_labels.reshape(-1, 1), predicted_labels)
    davies_bouldin = davies_bouldin_score(true_labels.reshape(-1, 1), predicted_labels)

    print(f"Rand Index: {rand_index:.4f}")
    print(f"Adjusted Rand Index: {adjusted_rand_index:.4f}")
    print(f"Mutual Info: {mutual_info:.4f}")
    print(f"Adjusted Mutual Info: {adjusted_mutual_info:.4f}")
    print(f"Normalized Mutual Info: {normalized_mutual_info:.4f}")
    print(f"Silhouette Coefficient: {silhouette_coeff:.4f}")
    print(f"Calinski-Harabasz Index: {calinski_harabasz:.4f}")
    print(f"Davies-Bouldin Index: {davies_bouldin:.4f}")
```

# Function to calculate cohesion and separation



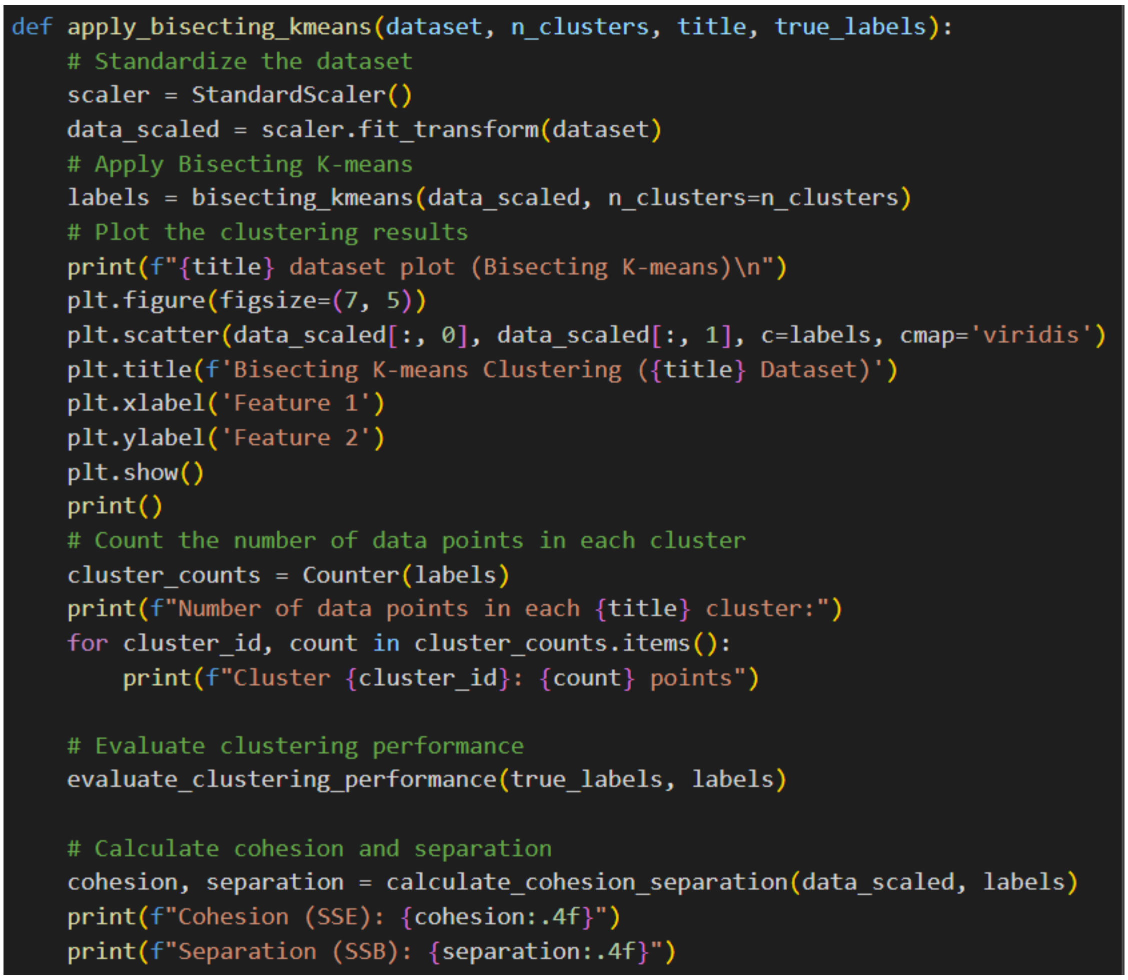
```python
# Function to calculate cohesion and separation
def calculate_cohesion_separation(X, labels):
    sse = 0  # Sum of Squared Errors (Cohesion)
    ssb = 0  # Sum of Squares Between groups (Separation)

    # Calculate SSE (cohesion)
    for i in np.unique(labels):
        cluster_points = X[labels == i]
        if cluster_points.shape[0] > 0:
            sse += np.sum((cluster_points - cluster_points.mean(axis=0)) ** 2)

    # Calculate SSB (separation)
    overall_mean = X.mean(axis=0)
    for i in np.unique(labels):
        cluster_points = X[labels == i]
        ssb += cluster_points.shape[0] * np.sum((cluster_points.mean(axis=0) - overall_mean) ** 2)

    return sse, ssb
```

# Function to plot Bisecting K-means results & also number of data points in each cluster



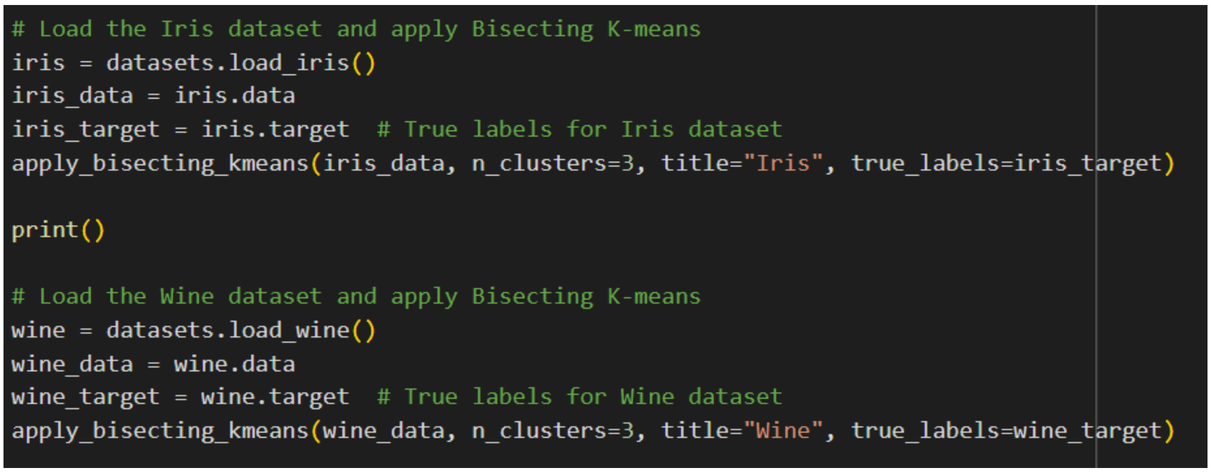
```python
# Function to plot clustering results
def plot_clustering(X, labels, dataset_name):
    print()
    plt.figure(figsize=(7, 5))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
    plt.title(f'KMeans++ Clustering ({dataset_name} Dataset)')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    #plt.legend(handles=scatter.legend_elements()[0], labels=colors)
    plt.show()

# Function to count the number of data points in each cluster
def count_clusters(labels, dataset_name):
    cluster_counts = Counter(labels)
    print(f"\nNumber of data points in each {dataset_name} cluster:")
    for cluster_id, count in cluster_counts.items():
        print(f"Cluster {cluster_id + 1}: {count} points")
```

# Function to run Bisecting K-means clustering for both datasets



```python
def apply_bisecting_kmeans(dataset, n_clusters, title, true_labels):
    # Standardize the dataset
    scaler = StandardScaler()
    data_scaled = scaler.fit_transform(dataset)
    # Apply Bisecting K-means
    labels = bisecting_kmeans(data_scaled, n_clusters=n_clusters)
    # Plot the clustering results
    print(f"{title} dataset plot (Bisecting K-means)\n")
    plt.figure(figsize=(7, 5))
    plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis')
    plt.title(f'Bisecting K-means Clustering ({title} Dataset)')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()
    print()
    # Count the number of data points in each cluster
    cluster_counts = Counter(labels)
    print(f"Number of data points in each {title} cluster:")
    for cluster_id, count in cluster_counts.items():
        print(f"Cluster {cluster_id}: {count} points")

    # Evaluate clustering performance
    evaluate_clustering_performance(true_labels, labels)

    # Calculate cohesion and separation
    cohesion, separation = calculate_cohesion_separation(data_scaled, labels)
    print(f"Cohesion (SSE): {cohesion:.4f}")
    print(f"Separation (SSB): {separation:.4f}")
```

# Loading & applying bisecting k-means on both datasets



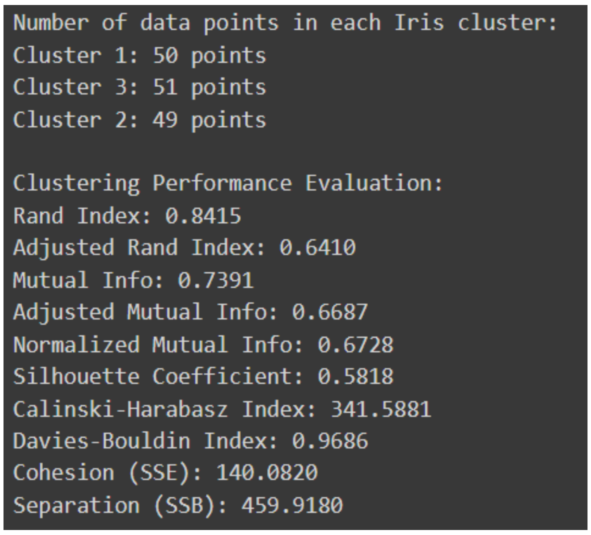
```python
# Load the Iris dataset and apply Bisecting K-means
iris = datasets.load_iris()
iris_data = iris.data
iris_target = iris.target  # True labels for Iris dataset
apply_bisecting_kmeans(iris_data, n_clusters=3, title="Iris", true_labels=iris_target)

print()

# Load the Wine dataset and apply Bisecting K-means
wine = datasets.load_wine()
wine_data = wine.data
wine_target = wine.target  # True labels for Wine dataset
apply_bisecting_kmeans(wine_data, n_clusters=3, title="Wine", true_labels=wine_target)
```

Analyse Bisecting K-Means for both dataset

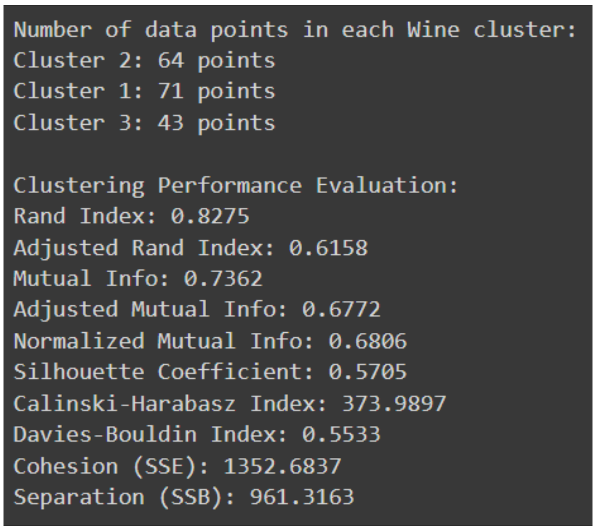
- Bisecting K-Means starts with the entire dataset as a single cluster and iteratively splits the cluster with the highest sum of squared errors (SSE) until the desired number of clusters is reached.
- The clustering evaluation metrics—such as Rand Index, Adjusted Rand Index, and Mutual Information scores—demonstrate that Bisecting K-Means effectively identifies the underlying structure of both datasets, with scores indicating a strong alignment between predicted clusters and true labels.
- The Silhouette Coefficient, Calinski-Harabasz Index, and Davies-Bouldin Index further confirm that the clusters are well-defined and separated.
- he calculated cohesion (SSE) and separation (SSB) metrics provide insights into the compactness of the clusters and their separability. Low SSE values indicate that the clusters are tight and closely packed, while high SSB values suggest well-separated clusters.

Overall, Bisecting K-Means proves to be a robust clustering method for both the Iris and Wine datasets. Its hierarchical splitting strategy enhances the identification of meaningful clusters, and the evaluation metrics reflect its ability to effectively uncover the underlying data patterns.

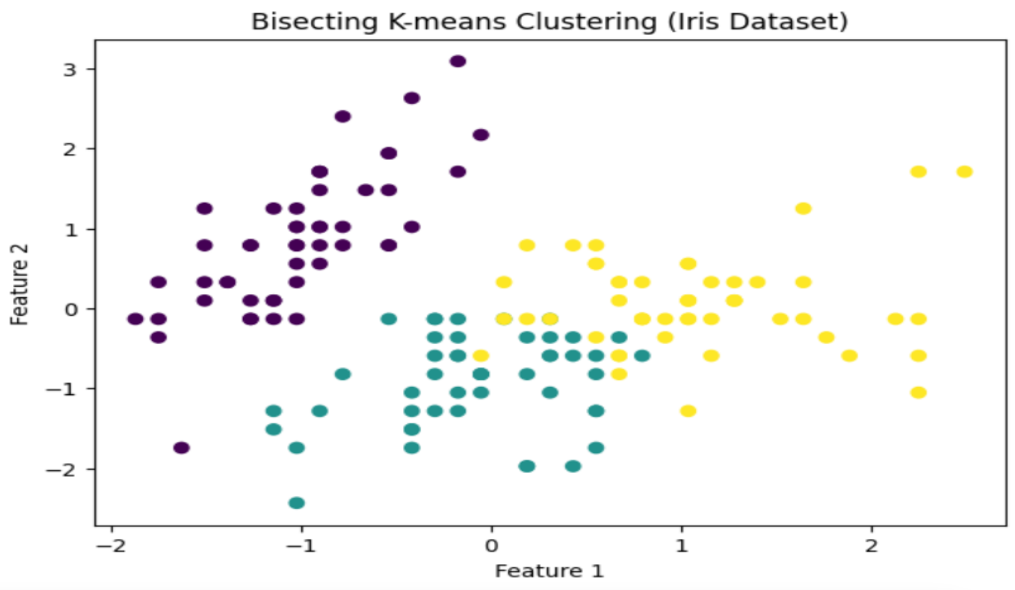
Output for both datasets

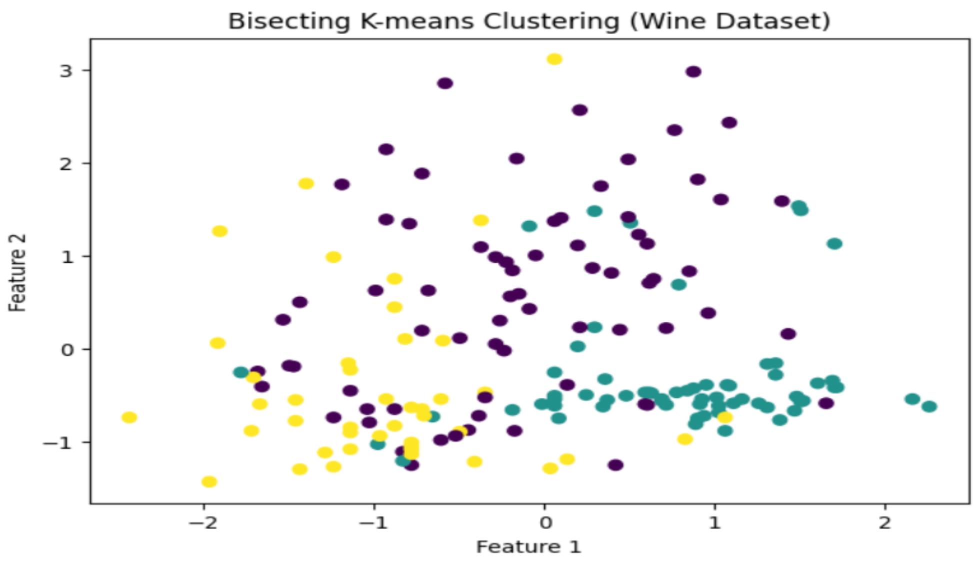


```
Number of data points in each Iris cluster:
Cluster 1: 50 points
Cluster 3: 51 points
Cluster 2: 49 points

Clustering Performance Evaluation:
Rand Index: 0.8415
Adjusted Rand Index: 0.6410
Mutual Info: 0.7391
Adjusted Mutual Info: 0.6687
Normalized Mutual Info: 0.6728
Silhouette Coefficient: 0.5818
Calinski-Harabasz Index: 341.5881
Davies-Bouldin Index: 0.9686
Cohesion (SSE): 140.0820
Separation (SSB): 459.9180
```



```
Number of data points in each Wine cluster:
Cluster 2: 64 points
Cluster 1: 71 points
Cluster 3: 43 points

Clustering Performance Evaluation:
Rand Index: 0.8275
Adjusted Rand Index: 0.6158
Mutual Info: 0.7362
Adjusted Mutual Info: 0.6772
Normalized Mutual Info: 0.6806
Silhouette Coefficient: 0.5705
Calinski-Harabasz Index: 373.9897
Davies-Bouldin Index: 0.5533
Cohesion (SSE): 1352.6837
Separation (SSB): 961.3163
```





Overall Conclusion for all clusterings

K-Means:

- K-Means is a widely used clustering algorithm known for its simplicity and efficiency. However, its performance can be sensitive to the initial placement of centroids, which may lead to varying results across different runs. Despite this, K-Means generally provides reasonable cluster separations, making it suitable for datasets with well-defined clusters.

K-Means++:

- This variant of K-Means improves upon the standard approach by employing a smarter initialization strategy for centroids. As a result, it typically yields better clustering performance and consistency, effectively capturing the underlying data structure. It is particularly useful when dealing with complex datasets.

Bisecting K-Means:

- This method combines aspects of K-Means with hierarchical clustering, often resulting in improved performance. It efficiently divides the dataset into smaller clusters, which can lead to better clustering outcomes. Bisecting K-Means generally demonstrated strong performance across both datasets, indicating its versatility and reliability.

K-Medoids:

- K-Medoids offers robustness to noise and outliers, making it an excellent choice for datasets where such anomalies are present. It achieves similar or better results compared to K-Means in scenarios with outliers, providing a more reliable alternative for clustering tasks.

Hierarchical Clustering (Dendrogram):

- The use of dendrograms for hierarchical clustering provides valuable insights into the relationships among data points. They facilitate the visualization of how clusters are formed, helping to determine the appropriate number of clusters. This method is particularly useful for exploratory data analysis.

DBSCAN:

- DBSCAN excels at identifying clusters of varying shapes and densities, making it a suitable choice for more complex datasets. However, it may struggle with datasets where clusters are of similar density, leading to challenges in capturing the true structure of the data. Its performance can significantly depend on parameter selection, which requires careful tuning.

OPTICS:

- Similar to DBSCAN, OPTICS is designed to handle varying densities effectively. It can provide a more detailed view of cluster structure through its ordering of points. However, like DBSCAN, its effectiveness may vary based on the dataset's density characteristics.

Overall, the clustering performance varied across algorithms and datasets, emphasizing the importance of selecting the appropriate method based on the dataset characteristics. K-Medoids and Bisecting K-Means stood out as reliable choices across both the Iris and Wine datasets, demonstrating effective clustering capabilities. Clustering remains a complex task that requires careful consideration of algorithm choice, parameter tuning, and the specific data characteristics to achieve optimal results.

Metrics Results for Each Clustering

Metrics / Clustering Algorithms	Rand index	Adjusted Rand index	Mutual Info	Adjusted Mutual Info	SSE
K-Means	0.7214	0.4328	0.5873	0.5838	191.024
K-Means++	0.7214	0.4328	0.5873	0.5838	191.024
Bisecting K-Means	0.6133	0.2557	0.2724	0.3095	524.845
K-Medoids	0.8367	0.6311	0.7330	0.6645	140.78
Dendrogram	0.8252	0.6153	0.7227	0.6712	131.40
DBSCAN	0.7475	0.4420	0.5498	0.5051	279.095
OPTICS	0.5121	0.0514	0.3081	0.2656	465.414

Metrics Results for Each Clustering

Metrics / Clustering Algorithms	Normalized Mutual Info	Silhouette Coefficient	Calinski-Harabasz Index	Davies-Bouldin index	SSB
K-Means	0.58956	0.479881	157.3601	0.7893	408.9
K-Means++	0.58956	0.479881	157.3601	0.7893	408.9
Bisecting K-Means	0.31491	21.19276	0.272456	1.9657	75.15
K-Medoids	0.66871	0.459042	239.7482	0.8384	459.2
Dendrogram	0.67547	0.4466	222.7191	0.8034	451.1
DBSCAN	0.51143	0.356516	84.51033	7.1240	320.9
OPTICS	0.29235	-0.30086	8.328213	2.4253	134.5