MACHINE LEARNING LABORATORY

ASSIGNMENT 4
Name: Atmik Goswami
Roll No: 002211001104
Section: A2

Github: github.com/atmikgoswami/ML-Lab

DATASETS:

1. **Iris Plants Dataset**
    - Features: Sepal Length, Sepal Width, Petal Length, Petal Width
    - Classes: Setosa, Versicolor, Virginica
    - Total Samples: 150

Sample Data:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

2. **Wine Dataset**
    - Features: 13 numeric features (e.g., alcohol, malic_acid, ash, etc.)
    - Classes: class_0, class_1, class_2
    - Total Samples: 178

Sample Data:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	class_0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	class_0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	class_0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	class_0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	class_0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740.0	class_2
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750.0	class_2
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835.0	class_2
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840.0	class_2
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560.0	class_2

178 rows × 14 columns

Apply the below clustering algorithms using Python:
a. Partition based: K-means, K-medoids/PAM
b. Hierarchical: Dendrogram
c. Density based: DBSCAN, OPTICS
on the following UCI datasets (can be loaded from the package itself):
a. Iris plants dataset: https://archive.ics.uci.edu/ml/datasets/Iris/
b. Wine Dataset: https://archive.ics.uci.edu/ml/datasets/wine
Additionally, implement K-means++ and Bisecting K-means.

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

Code:

```
!pip install scikit-learn-extra -q
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.datasets import load_wine
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN, OPTICS,
BisectingKMeans
from sklearn_extra.cluster import KMedoids
from scipy.cluster.hierarchy import dendrogram, linkage

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)
results = {}
def get_centroids(X, labels):
    centroids = []
    for i in np.unique(labels):
      if i != -1: # Ignore noise points labeled as -1
        centroids.append(np.mean(X[labels == i], axis=0))
    return np.array(centroids)

def calculate_sse(X, labels, centroids):
    sse = 0
    unique_labels = np.unique(labels)
```

```
    cluster_map = {label: idx for idx, label in enumerate(unique_labels) if label != -1} # Map cluster labels (excluding -1) to centroid indices

    if not centroids.size: # Handle case with no valid centroids found
        return np.inf # Or some indicator of inability to calculate

    for i in unique_labels:
        if i == -1: # Skip noise points
            continue
        if i not in cluster_map: # Should not happen if get_centroids works correctly, but for safety
             continue
        centroid_index = cluster_map[i]
        if centroid_index >= len(centroids): # Safety check for index out of bounds
            continue
        center = centroids[centroid_index]
        cluster_points = X[labels == i]
        if cluster_points.size > 0: # Ensure cluster is not empty
            sse += np.sum((cluster_points - center) ** 2)
    return sse


def calculate_ssb(X, labels, centroids):
    data_centroid = np.mean(X, axis=0)
    ssb = 0
    unique_labels = np.unique(labels)
    cluster_map = {label: idx for idx, label in enumerate(unique_labels) if label != -1}

    if not centroids.size:
        return 0 # Or handle appropriately

    for i in unique_labels:
        if i == -1: # Skip noise points
            continue
        if i not in cluster_map:
            continue
        centroid_index = cluster_map[i]
        if centroid_index >= len(centroids):
             continue

        center = centroids[centroid_index]
        n_points = np.sum(labels == i)
        if n_points > 0:
            ssb += n_points * np.sum((center - data_centroid) ** 2)
    return ssb

# **IRIS DATASET**

iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
```

```
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})

X = df.drop(['species'], axis=1)
y = df['species']
X_scaled = StandardScaler().fit_transform(X)
n_clusters = len(np.unique(y))

# K-Means

kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
kmeans.fit(X_scaled)
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['KMeans'] = df_metrics

print("--- KMeans Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")

# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'Iris-setosa')
```

```
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'Iris-versicolour')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'Iris-virginica')


# Centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c
= 'red', label = 'Centroids')

plt.legend()

# K-Medoids

kmedoid = KMedoids(n_clusters=n_clusters, random_state=42)
kmedoid.fit(X_scaled)
centroids = kmedoid.cluster_centers_
labels = kmedoid.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['KMedoids'] = df_metrics

print("--- KMedoids Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")
```

```
# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'Iris-setosa')
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'Iris-versicolour')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'Iris-virginica')


# Centroids of the clusters
plt.scatter(centroids[:, 0], centroids[:,1], s = 100, c = 'red', label =
'Centroids')

plt.legend()

# Dendogram

agg_cluster = AgglomerativeClustering(n_clusters=n_clusters)
agg_labels = agg_cluster.fit_predict(X_scaled)
centroids = get_centroids(X_scaled, agg_labels)
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, agg_labels, centroids),
        calculate_ssb(X_scaled, agg_labels, centroids),
        adjusted_rand_score(y, agg_labels),
        adjusted_mutual_info_score(y, agg_labels),
        normalized_mutual_info_score(y, agg_labels),
        silhouette_score(X_scaled, agg_labels),
        calinski_harabasz_score(X_scaled, agg_labels),
        davies_bouldin_score(X_scaled, agg_labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['Dendogram'] = df_metrics

print("--- Dendogram Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")
```

```
plt.figure(figsize=(12, 6))
linked = linkage(X_scaled, method='ward')
dendrogram(linked, orientation='top', distance_sort='descending',
show_leaf_counts=True)
plt.title(f'Hierarchical Clustering Dendrogram (Iris Dataset)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

# Density-based algorithms

density_params = {
        "DBSCAN": {'eps': 0.8 , 'min_samples': 5},
        "OPTICS": {'min_samples': 5}
    }

density_algs = {
    "DBSCAN": DBSCAN(**density_params["DBSCAN"]),
    "OPTICS": OPTICS(**density_params["OPTICS"])
}

# Define the standard list of metrics used elsewhere
metrics_list = [
    'Sum of Squared Errors (SSE)',
    'Sum of Squares Between (SSB)',
    'Adjusted Rand Index (ARI)',
    'Adjusted Mutual Information (AMI)',
    'Normalized Mutual Information (NMI)',
    'Silhouette Score',
    'Calinski-Harabasz Score',
    'Davies-Bouldin Score'
]


for name, alg in density_algs.items():
    print(f"--- Running {name} ---")
    labels = alg.fit_predict(X_scaled)
    # Check unique labels excluding noise label -1
    unique_labels = set(labels)
    n_clusters_found = len(unique_labels) - (1 if -1 in unique_labels else 0)
    print(f"Found {n_clusters_found} clusters (excluding noise).")

    if n_clusters_found < 2:
        print(f"Skipping metrics calculation for {name} as less than 2 clusters
were found.")
        # Create a placeholder DataFrame with N/A values
        na_values = ['N/A'] * len(metrics_list)
        df_metrics = pd.DataFrame({'Metric': metrics_list, 'Value': na_values})
        results[f'{name}'] = df_metrics # Assign placeholder DataFrame to the
results dictionary
```

```
        print(df_metrics.to_string(index=False))
        print("\n")
        continue

    # Calculate centroids and metrics only if 2 or more clusters are found
    centroids = get_centroids(X_scaled, labels) # Ensure get_centroids handles the
noise label -1
    metrics_data = {
        'Metric': metrics_list,
        'Value': [
            calculate_sse(X_scaled, labels, centroids),
            calculate_ssb(X_scaled, labels, centroids),
            adjusted_rand_score(y, labels),
            adjusted_mutual_info_score(y, labels),
            normalized_mutual_info_score(y, labels),
            silhouette_score(X_scaled, labels),
            calinski_harabasz_score(X_scaled, labels),
            davies_bouldin_score(X_scaled, labels)
        ]
    }

    df_metrics = pd.DataFrame(metrics_data)

    results[f'{name}'] = df_metrics # Assign metrics DataFrame to the results
dictionary

    print(f"--- {name} Evaluation Metrics ---")
    print(df_metrics.to_string(index=False))
    print("\n")

# KMeans++

kmeansplus = KMeans(n_clusters=n_clusters, random_state=42, n_init=10,
init='k-means++')
kmeansplus.fit(X_scaled)
centroids = kmeansplus.cluster_centers_
labels = kmeansplus.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
```

```
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['KMeans++'] = df_metrics

print("--- KMeans++ Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")

# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'Iris-setosa')
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'Iris-versicolour')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'Iris-virginica')


# Centroids of the clusters
plt.scatter(centroids[:, 0], centroids[:,1], s = 100, c = 'red', label =
'Centroids')

plt.legend()

# Bisecting K-means

bkmeans = BisectingKMeans(n_clusters=n_clusters, n_init=10, random_state=42)
bkmeans.fit(X_scaled)
centroids = bkmeans.cluster_centers_
labels = bkmeans.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
```

```
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['Bisecting KMeans'] = df_metrics

print("--- Bisecting KMeans Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")

# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'Iris-setosa')
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'Iris-versicolour')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'Iris-virginica')


# Centroids of the clusters
plt.scatter(centroids[:, 0], centroids[:,1], s = 100, c = 'red', label =
'Centroids')

plt.legend()
comparison_list = []

# Iterate through the results dictionary
for name, df_metric in results.items():
    processed_series = df_metric.set_index('Metric')['Value'].rename(name)
    comparison_list.append(processed_series)

comparison_df = pd.concat(comparison_list, axis=1)

print("--- Performance Comparison of Clustering Algorithms on Iris Dataset ---")
display(comparison_df.apply(pd.to_numeric, errors='coerce').round(4))

# **WINE DATASET**
```

```
wine = load_wine()

df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
df['target'] = wine.target
df['target'] = df['target'].apply(lambda x: wine.target_names[x])

X = df.drop('target', axis=1)
y = df['target']
X_scaled = StandardScaler().fit_transform(X)
n_clusters = len(np.unique(y))

# KMeans

kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
kmeans.fit(X_scaled)
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['KMeans'] = df_metrics

print("--- KMeans Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")
```

```
# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'class_0')
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'class_1')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'class_2')


# Centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c
= 'red', label = 'Centroids')

plt.legend()

# KMedoids

kmedoid = KMedoids(n_clusters=n_clusters, random_state=42)
kmedoid.fit(X_scaled)
centroids = kmedoid.cluster_centers_
labels = kmedoid.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['KMedoids'] = df_metrics

print("--- KMedoids Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
```

```
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")

# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'class_0')
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'class_1')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'class_2')


# Centroids of the clusters
plt.scatter(centroids[:, 0], centroids[:,1], s = 100, c = 'red', label =
'Centroids')

plt.legend()

# Dendogram

agg_cluster = AgglomerativeClustering(n_clusters=n_clusters)
agg_labels = agg_cluster.fit_predict(X_scaled)
centroids = get_centroids(X_scaled, agg_labels)
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, agg_labels, centroids),
        calculate_ssb(X_scaled, agg_labels, centroids),
        adjusted_rand_score(y, agg_labels),
        adjusted_mutual_info_score(y, agg_labels),
        normalized_mutual_info_score(y, agg_labels),
        silhouette_score(X_scaled, agg_labels),
        calinski_harabasz_score(X_scaled, agg_labels),
        davies_bouldin_score(X_scaled, agg_labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['Dendogram'] = df_metrics
```

```
print("--- Dendogram Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(12, 6))
linked = linkage(X_scaled, method='ward')
dendrogram(linked, orientation='top', distance_sort='descending',
show_leaf_counts=True)
plt.title(f'Hierarchical Clustering Dendrogram (Iris Dataset)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
# Density-based algorithms

density_params = {
        "DBSCAN": {'eps': 0.8 , 'min_samples': 5},
        "OPTICS": {'min_samples': 5}
    }

density_algs = {
    "DBSCAN": DBSCAN(**density_params["DBSCAN"]),
    "OPTICS": OPTICS(**density_params["OPTICS"])
}

metrics_list = [
    'Sum of Squared Errors (SSE)',
    'Sum of Squares Between (SSB)',
    'Adjusted Rand Index (ARI)',
    'Adjusted Mutual Information (AMI)',
    'Normalized Mutual Information (NMI)',
    'Silhouette Score',
    'Calinski-Harabasz Score',
    'Davies-Bouldin Score'
]


for name, alg in density_algs.items():
    print(f"--- Running {name} ---")
    labels = alg.fit_predict(X_scaled)
    unique_labels = set(labels)
    n_clusters_found = len(unique_labels) - (1 if -1 in unique_labels else 0)
    print(f"Found {n_clusters_found} clusters (excluding noise).")

    if n_clusters_found < 2:
        print(f"Skipping metrics calculation for {name} as less than 2 clusters
were found.")
        na_values = ['N/A'] * len(metrics_list)
        df_metrics = pd.DataFrame({'Metric': metrics_list, 'Value': na_values})
        results[f'{name}'] = df_metrics
        print(df_metrics.to_string(index=False))
```

```
        print("\n")
        continue

    centroids = get_centroids(X_scaled, labels)
    metrics_data = {
        'Metric': metrics_list,
        'Value': [
            calculate_sse(X_scaled, labels, centroids),
            calculate_ssb(X_scaled, labels, centroids),
            adjusted_rand_score(y, labels),
            adjusted_mutual_info_score(y, labels),
            normalized_mutual_info_score(y, labels),
            silhouette_score(X_scaled, labels),
            calinski_harabasz_score(X_scaled, labels),
            davies_bouldin_score(X_scaled, labels)
        ]
    }

    df_metrics = pd.DataFrame(metrics_data)

    results[f'{name}'] = df_metrics

    print(f"--- {name} Evaluation Metrics ---")
    print(df_metrics.to_string(index=False))
    print("\n")

# KMeans++

kmeansplus = KMeans(n_clusters=n_clusters, random_state=42, n_init=10,
init='k-means++')
kmeansplus.fit(X_scaled)
centroids = kmeansplus.cluster_centers_
labels = kmeansplus.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
```

```
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['KMeans++'] = df_metrics

print("--- KMeans++ Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")

# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'class_0')
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'class_1')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'class_2')


# Centroids of the clusters
plt.scatter(centroids[:, 0], centroids[:,1], s = 100, c = 'red', label =
'Centroids')

plt.legend()

# Bisecting K-means

bkmeans = BisectingKMeans(n_clusters=n_clusters, n_init=10, random_state=42)
bkmeans.fit(X_scaled)
centroids = bkmeans.cluster_centers_
labels = bkmeans.labels_
metrics_data = {
    'Metric': [
        'Sum of Squared Errors (SSE)',
        'Sum of Squares Between (SSB)',
        'Adjusted Rand Index (ARI)',
        'Adjusted Mutual Information (AMI)',
        'Normalized Mutual Information (NMI)',
        'Silhouette Score',
        'Calinski-Harabasz Score',
        'Davies-Bouldin Score'
    ],
    'Value': [
        calculate_sse(X_scaled, labels, centroids),
        calculate_ssb(X_scaled, labels, centroids),
```

```
        adjusted_rand_score(y, labels),
        adjusted_mutual_info_score(y, labels),
        normalized_mutual_info_score(y, labels),
        silhouette_score(X_scaled, labels),
        calinski_harabasz_score(X_scaled, labels),
        davies_bouldin_score(X_scaled, labels)
    ]
}

df_metrics = pd.DataFrame(metrics_data)

results['Bisecting KMeans'] = df_metrics

print("--- Bisecting KMeans Evaluation Metrics ---")
print(df_metrics.to_string(index=False))
print("\n")

plt.figure(figsize=(10, 8))
sns.set_theme(style="whitegrid")

# Cluster points
plt.scatter(X_scaled[labels == 0, 0], X_scaled[labels == 0, 1], s = 100, c =
'purple', label = 'class_0')
plt.scatter(X_scaled[labels == 1, 0], X_scaled[labels == 1, 1], s = 100, c =
'orange', label = 'class_1')
plt.scatter(X_scaled[labels == 2, 0], X_scaled[labels == 2, 1], s = 100, c =
'green', label = 'class_2')


# Centroids of the clusters
plt.scatter(centroids[:, 0], centroids[:,1], s = 100, c = 'red', label =
'Centroids')

plt.legend()

comparison_list = []

# Iterate through the results dictionary
for name, df_metric in results.items():
    processed_series = df_metric.set_index('Metric')['Value'].rename(name)
    comparison_list.append(processed_series)

comparison_df = pd.concat(comparison_list, axis=1)

print("--- Performance Comparison of Clustering Algorithms on Wine Dataset ---")
display(comparison_df.apply(pd.to_numeric, errors='coerce').round(4))
```

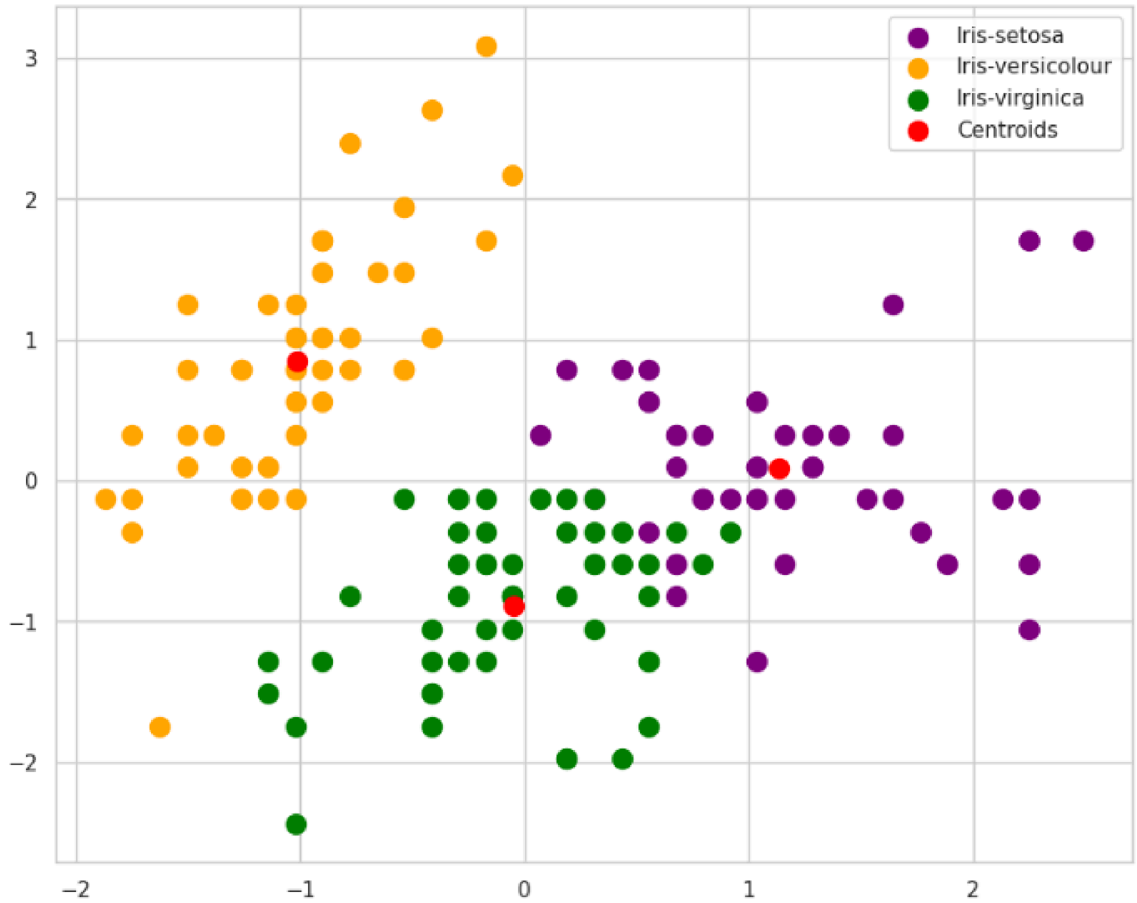
Results and Discussion:

Iris Dataset:

KMeans:

```
--- KMeans Evaluation Metrics ---
                              Metric       Value
          Sum of Squared Errors (SSE)  139.820496
         Sum of Squares Between (SSB)  460.179504
            Adjusted Rand Index (ARI)    0.620135
   Adjusted Mutual Information (AMI)    0.655223
 Normalized Mutual Information (NMI)    0.659487
                     Silhouette Score    0.459948
              Calinski-Harabasz Score  241.904402
                 Davies-Bouldin Score    0.833595
```

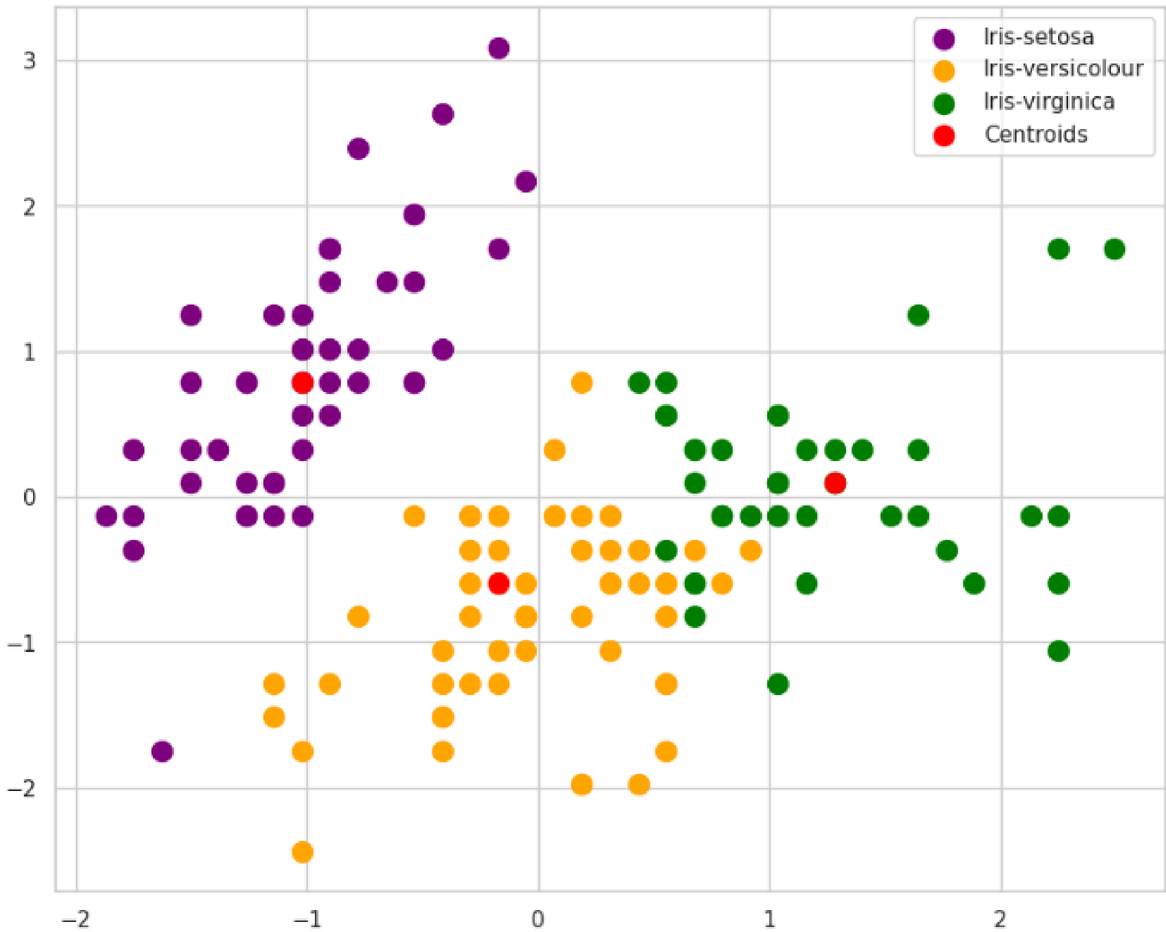
```
<matplotlib.legend.Legend at 0x793014a6b690>
```



KMedoids:

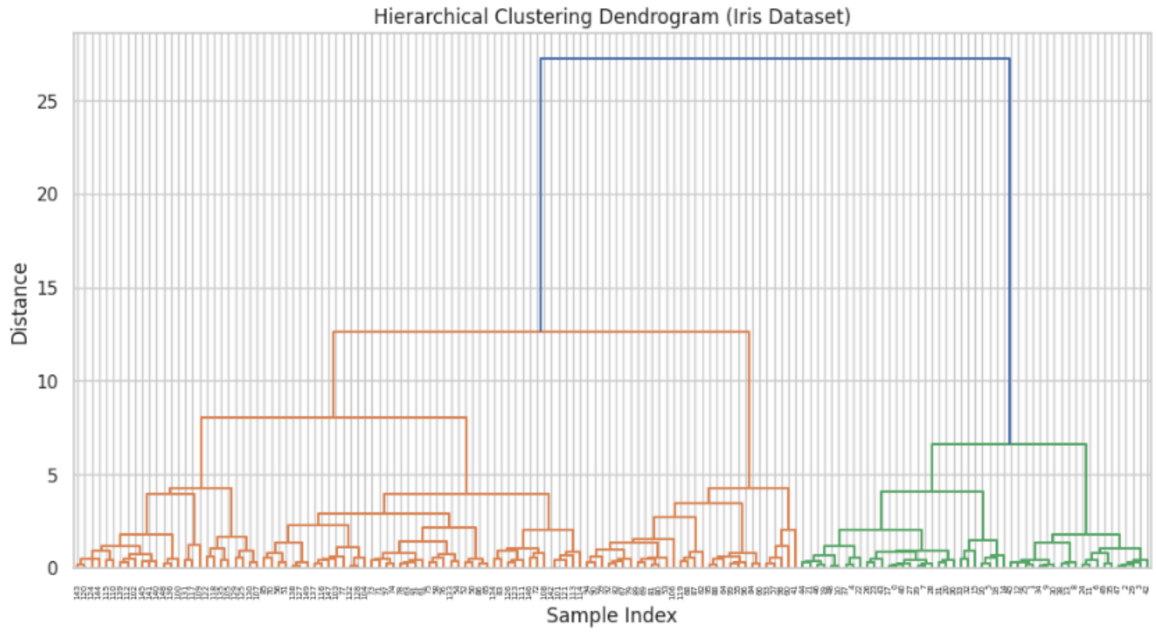
```
--- KMedoids Evaluation Metrics ---
                              Metric       Value
          Sum of Squared Errors (SSE) 148.610356
         Sum of Squares Between (SSB) 457.198161
            Adjusted Rand Index (ARI)   0.631158
      Adjusted Mutual Information (AMI)   0.664557
    Normalized Mutual Information (NMI)   0.668714
                     Silhouette Score   0.459042
              Calinski-Harabasz Score 239.748268
                 Davies-Bouldin Score   0.838455
```

```
<matplotlib.legend.Legend at 0x79301094bfd0>
```



Dendogram:

```
--- Dendogram Evaluation Metrics ---
                              Metric       Value
         Sum of Squared Errors (SSE) 1697.304675
        Sum of Squares Between (SSB)  504.127365
            Adjusted Rand Index (ARI)    0.615323
  Adjusted Mutual Information (AMI)    0.671286
Normalized Mutual Information (NMI)    0.675470
                    Silhouette Score    0.446689
             Calinski-Harabasz Score  222.719164
                Davies-Bouldin Score    0.803467
```



DBSCAN and OPTICS:

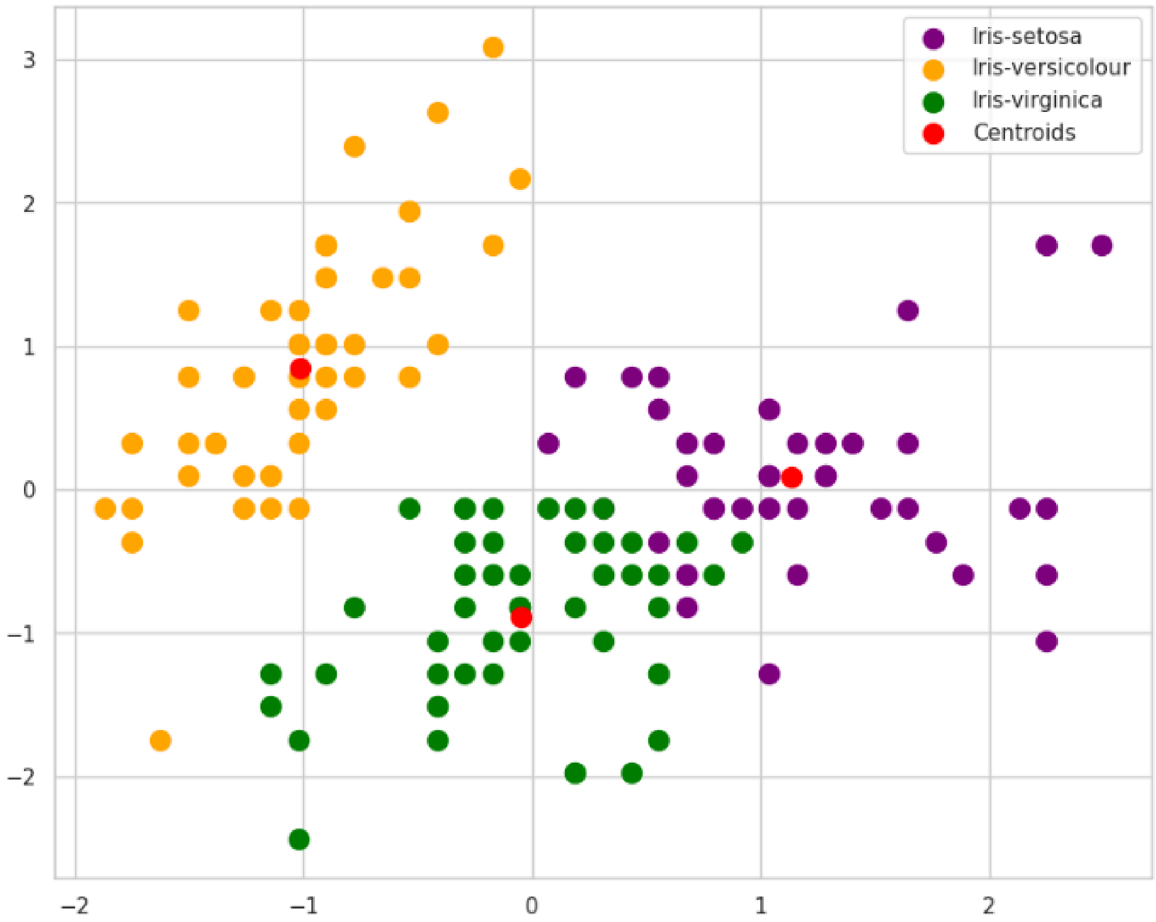
```
--- Running DBSCAN ---
Found 2 clusters (excluding noise).
--- DBSCAN Evaluation Metrics ---
                              Metric       Value
         Sum of Squared Errors (SSE)  190.101197
        Sum of Squares Between (SSB)  366.877525
            Adjusted Rand Index (ARI)    0.551755
  Adjusted Mutual Information (AMI)    0.684771
Normalized Mutual Information (NMI)    0.689979
                    Silhouette Score    0.521697
             Calinski-Harabasz Score  126.221166
                Davies-Bouldin Score    1.943201
```

```
--- Running OPTICS ---
Found 5 clusters (excluding noise).
--- OPTICS Evaluation Metrics ---
                              Metric       Value
         Sum of Squared Errors (SSE)    3.788517
        Sum of Squares Between (SSB)  129.762736
            Adjusted Rand Index (ARI)    0.051416
  Adjusted Mutual Information (AMI)    0.265690
Normalized Mutual Information (NMI)    0.292357
                    Silhouette Score   -0.300865
             Calinski-Harabasz Score    8.328211
                Davies-Bouldin Score    2.425310
```

KMeans++:

```
--- KMeans++ Evaluation Metrics ---
                              Metric       Value
            Sum of Squared Errors (SSE) 139.820496
          Sum of Squares Between (SSB) 460.179504
               Adjusted Rand Index (ARI)   0.620135
    Adjusted Mutual Information (AMI)   0.655223
  Normalized Mutual Information (NMI)   0.659487
                        Silhouette Score   0.459948
                  Calinski-Harabasz Score 241.904402
                     Davies-Bouldin Score   0.833595
```

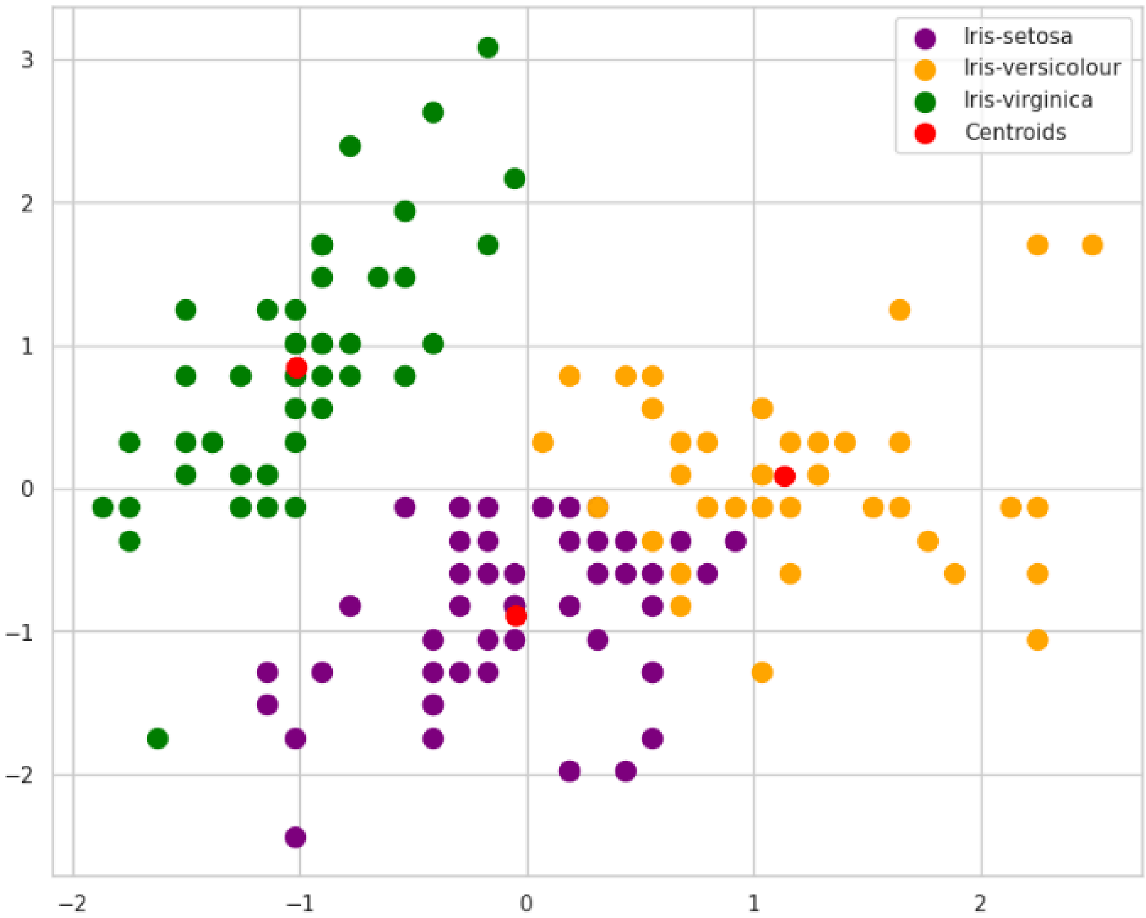
```
<matplotlib.legend.Legend at 0x79300bed5090>
```



Bisecting KMeans:

```
--- Bisecting KMeans Evaluation Metrics ---
                               Metric       Value
          Sum of Squared Errors (SSE) 139.820496
         Sum of Squares Between (SSB) 460.179504
            Adjusted Rand Index (ARI)   0.620135
     Adjusted Mutual Information (AMI)   0.655223
   Normalized Mutual Information (NMI)   0.659487
                     Silhouette Score   0.459948
              Calinski-Harabasz Score 241.904402
                 Davies-Bouldin Score   0.833595
```

```
<matplotlib.legend.Legend at 0x79300bde5090>
```



Performance Comparison:

```
--- Performance Comparison of Clustering Algorithms on Iris Dataset ---
```

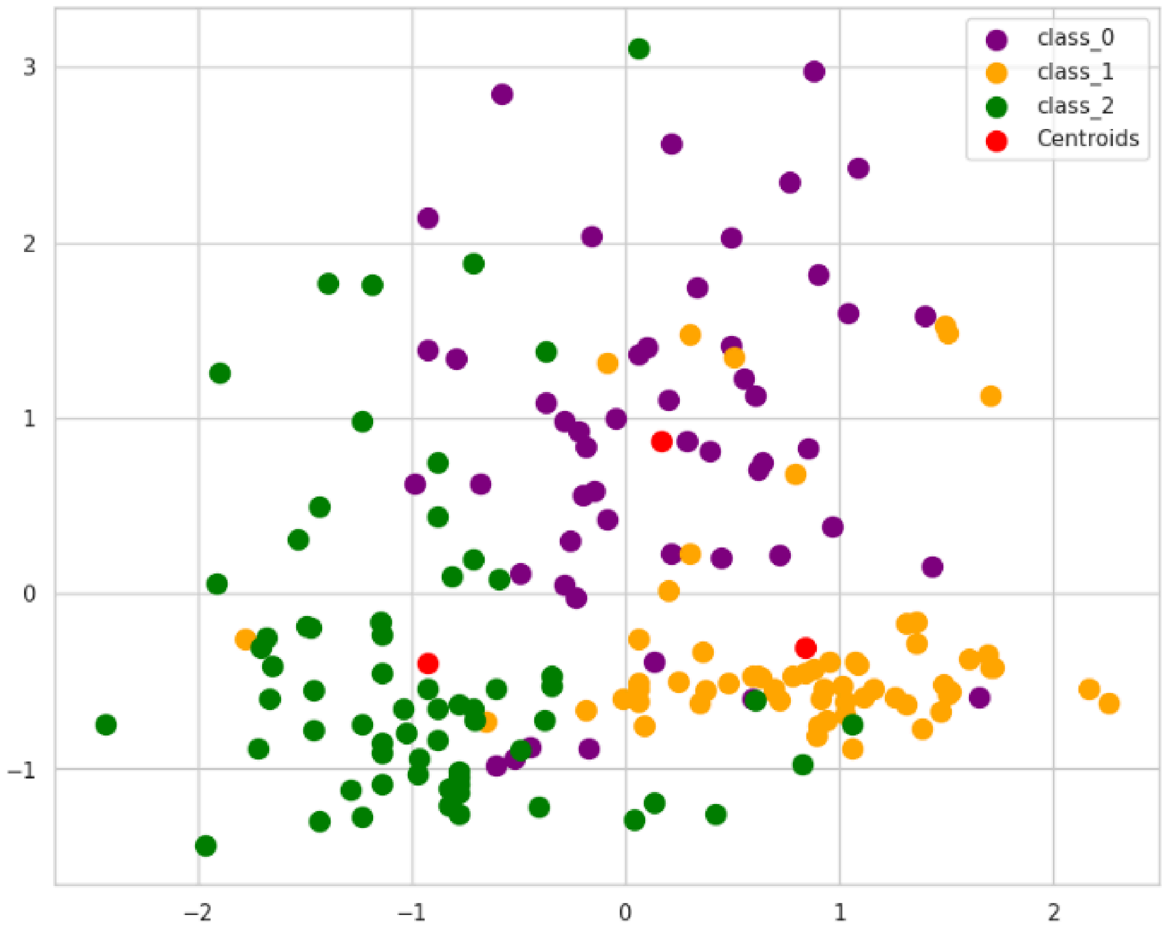
Metric	KMeans	KMedoids	Dendogram	DBSCAN	OPTICS	KMeans++	Bisecting KMeans
Sum of Squared Errors (SSE)	139.8205	148.6104	1697.3047	190.1012	3.7885	139.8205	139.8205
Sum of Squares Between (SSB)	460.1795	457.1982	504.1274	366.8775	129.7627	460.1795	460.1795
Adjusted Rand Index (ARI)	0.6201	0.6312	0.6153	0.5518	0.0514	0.6201	0.6201
Adjusted Mutual Information (AMI)	0.6552	0.6646	0.6713	0.6848	0.2657	0.6552	0.6552
Normalized Mutual Information (NMI)	0.6595	0.6687	0.6755	0.6900	0.2924	0.6595	0.6595
Silhouette Score	0.4599	0.4590	0.4467	0.5217	-0.3009	0.4599	0.4599
Calinski-Harabasz Score	241.9044	239.7483	222.7192	126.2212	8.3282	241.9044	241.9044
Davies-Bouldin Score	0.8336	0.8385	0.8035	1.9432	2.4253	0.8336	0.8336

Wine Dataset:

KMeans:

```
--- KMeans Evaluation Metrics ---
                              Metric        Value
          Sum of Squared Errors (SSE) 1277.928489
         Sum of Squares Between (SSB) 1036.071511
            Adjusted Rand Index (ARI)    0.897495
     Adjusted Mutual Information (AMI)    0.874579
   Normalized Mutual Information (NMI)    0.875894
                      Silhouette Score    0.284859
               Calinski-Harabasz Score   70.940008
                  Davies-Bouldin Score    1.389188
```

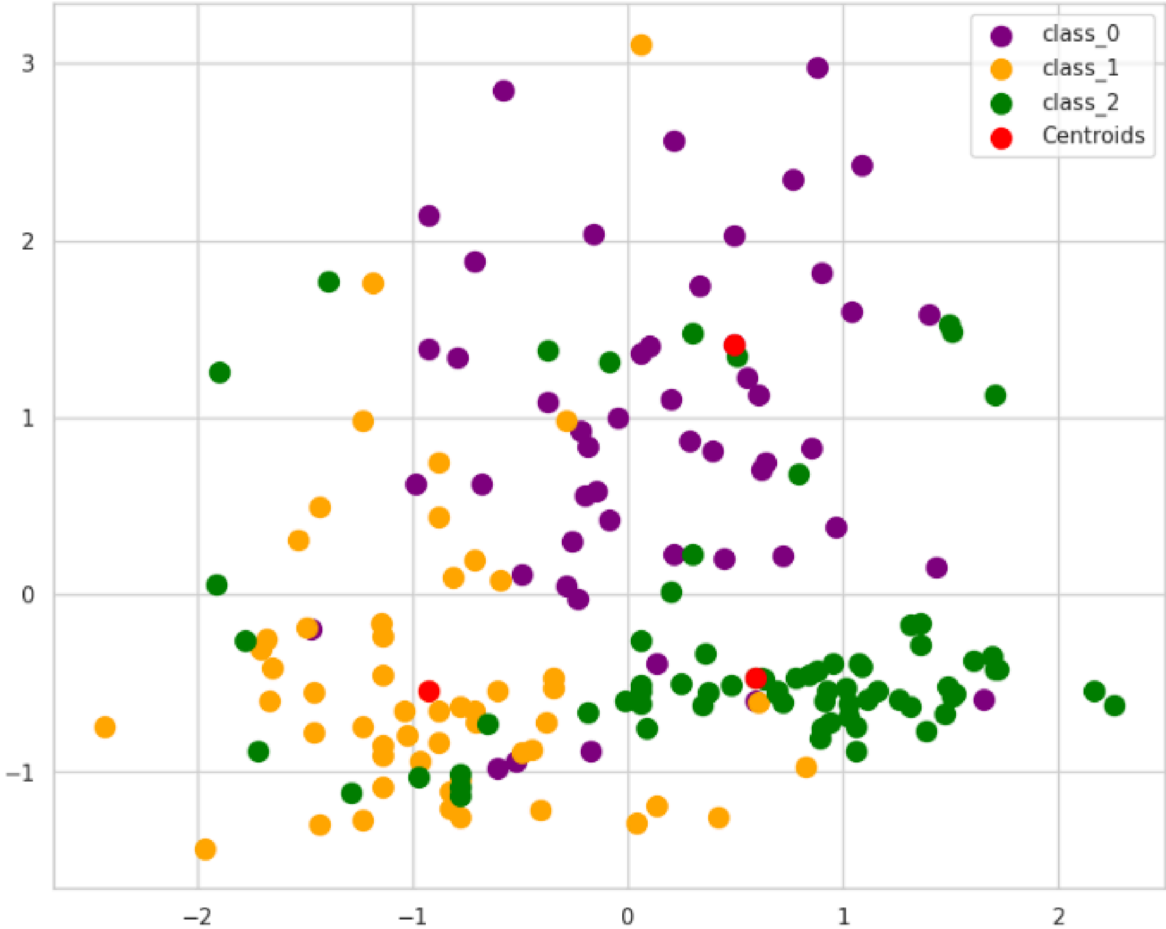
```
<matplotlib.legend.Legend at 0x79300ba6b690>
```



KMedoids:

```
--- KMedoids Evaluation Metrics ---
                              Metric        Value
          Sum of Squared Errors (SSE) 1564.606349
         Sum of Squares Between (SSB) 1258.524095
            Adjusted Rand Index (ARI)    0.726341
   Adjusted Mutual Information (AMI)    0.753986
 Normalized Mutual Information (NMI)    0.756574
                    Silhouette Score    0.265977
             Calinski-Harabasz Score   66.751966
                Davies-Bouldin Score    1.415990
```

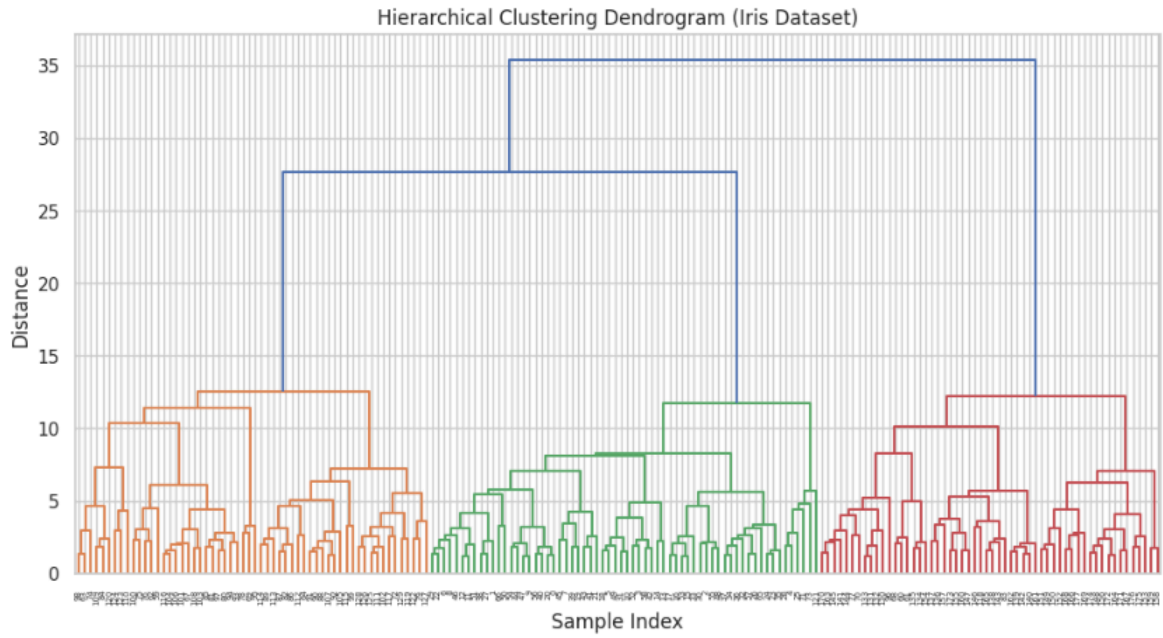
```
<matplotlib.legend.Legend at 0x79300ba5fb90>
```



Dendogram:

--- Dendogram Evaluation Metrics ---

Metric	Value
Sum of Squared Errors (SSE)	3538.722556
Sum of Squares Between (SSB)	1303.104448
Adjusted Rand Index (ARI)	0.789933
Adjusted Mutual Information (AMI)	0.784208
Normalized Mutual Information (NMI)	0.786465
Silhouette Score	0.277444
Calinski-Harabasz Score	67.647468
Davies-Bouldin Score	1.418592



DBSCAN and OPTICS:

--- Running DBSCAN ---

Found 0 clusters (excluding noise).

Skipping metrics calculation for DBSCAN as less than 2 clusters were found.

Metric	Value
Sum of Squared Errors (SSE)	N/A
Sum of Squares Between (SSB)	N/A
Adjusted Rand Index (ARI)	N/A
Adjusted Mutual Information (AMI)	N/A
Normalized Mutual Information (NMI)	N/A
Silhouette Score	N/A
Calinski-Harabasz Score	N/A
Davies-Bouldin Score	N/A

--- Running OPTICS ---

Found 4 clusters (excluding noise).

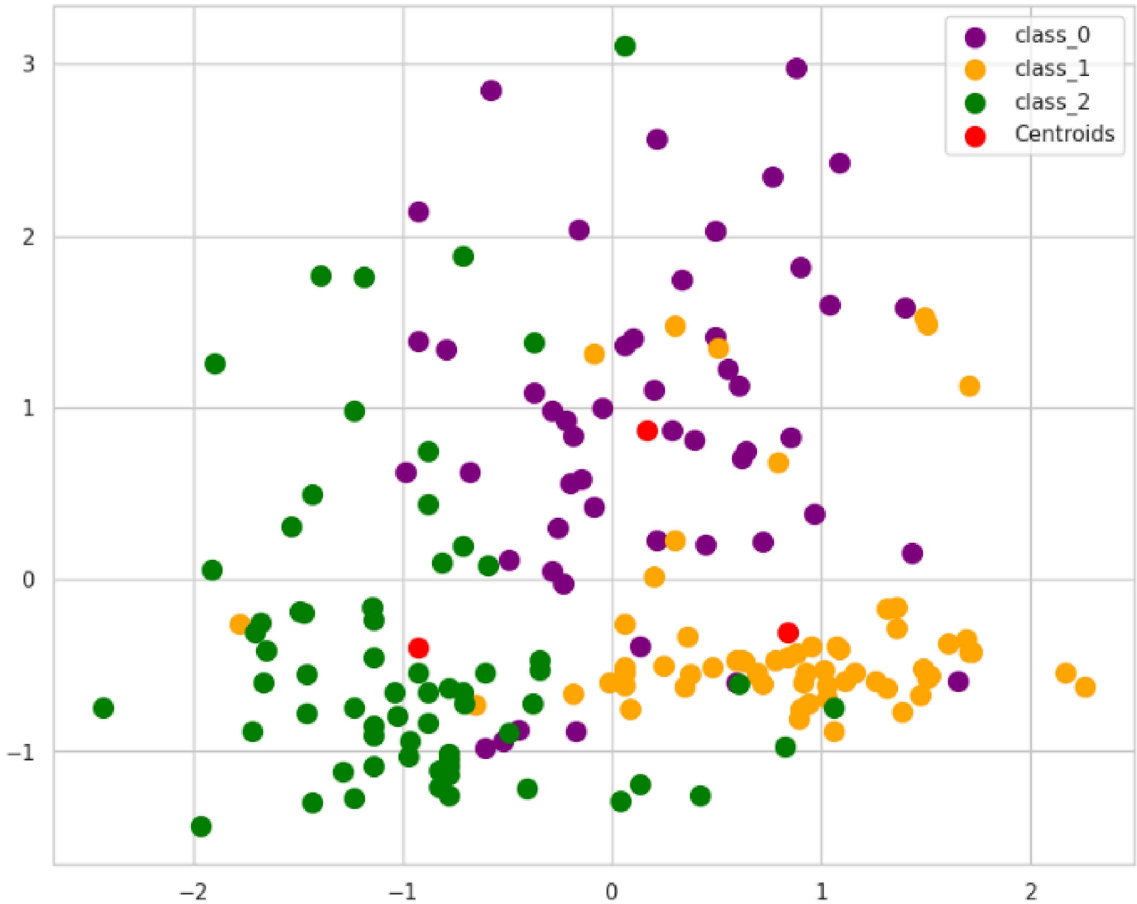
--- OPTICS Evaluation Metrics ---

Metric	Value
Sum of Squared Errors (SSE)	38.868103
Sum of Squares Between (SSB)	225.896473
Adjusted Rand Index (ARI)	0.035817
Adjusted Mutual Information (AMI)	0.167991
Normalized Mutual Information (NMI)	0.194946
Silhouette Score	-0.133640
Calinski-Harabasz Score	5.057149
Davies-Bouldin Score	1.619371

KMeans++:

```
--- KMeans++ Evaluation Metrics ---
                             Metric        Value
           Sum of Squared Errors (SSE) 1277.928489
          Sum of Squares Between (SSB) 1036.071511
             Adjusted Rand Index (ARI)    0.897495
    Adjusted Mutual Information (AMI)    0.874579
  Normalized Mutual Information (NMI)    0.875894
                      Silhouette Score    0.284859
               Calinski-Harabasz Score   70.940008
                  Davies-Bouldin Score    1.389188
```

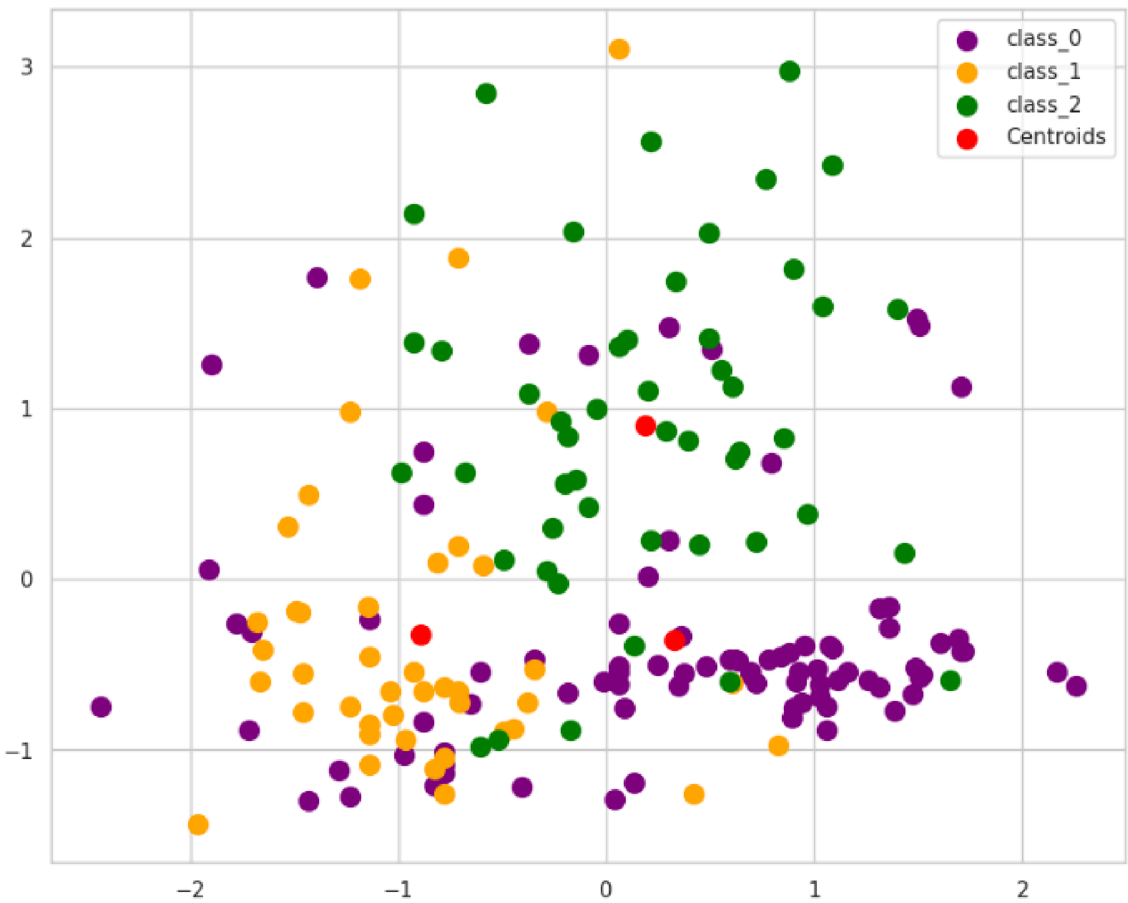
```
<matplotlib.legend.Legend at 0x7930109823d0>
```



Bisecting KMeans:

```
--- Bisecting KMeans Evaluation Metrics ---
                              Metric        Value
          Sum of Squared Errors (SSE) 1375.112890
         Sum of Squares Between (SSB)  938.887110
            Adjusted Rand Index (ARI)    0.590631
    Adjusted Mutual Information (AMI)    0.701892
  Normalized Mutual Information (NMI)    0.705090
                    Silhouette Score     0.234076
             Calinski-Harabasz Score    59.742457
                Davies-Bouldin Score     1.520319
```

```
<matplotlib.legend.Legend at 0x79300bea8890>
```



Performance Comparison:

Metric	DBSCAN	OPTICS	KMeans	KMedoids	Dendogram	KMeans++	Bisecting KMeans
Sum of Squared Errors (SSE)	NaN	38.8681	1277.9285	1564.6063	3538.7226	1277.9285	1375.1129
Sum of Squares Between (SSB)	NaN	225.8965	1036.0715	1258.5241	1303.1044	1036.0715	938.8871
Adjusted Rand Index (ARI)	NaN	0.0358	0.8975	0.7263	0.7899	0.8975	0.5906
Adjusted Mutual Information (AMI)	NaN	0.1680	0.8746	0.7540	0.7842	0.8746	0.7019
Normalized Mutual Information (NMI)	NaN	0.1949	0.8759	0.7566	0.7865	0.8759	0.7051
Silhouette Score	NaN	-0.1336	0.2849	0.2660	0.2774	0.2849	0.2341
Calinski-Harabasz Score	NaN	5.0571	70.9400	66.7520	67.6475	70.9400	59.7425
Davies-Bouldin Score	NaN	1.6194	1.3892	1.4160	1.4186	1.3892	1.5203

Discussion:

Iris Dataset Analysis:

- **Top Performers:** Partition-based and hierarchical methods performed strongly. KMedoids, Agglomerative Clustering (Dendrogram), and the KMeans variants (standard, ++, and Bisecting) all achieved high external validation scores (Adjusted Rand Index ~0.62-0.63, Normalized Mutual Information ~0.66-0.68), indicating a good match with the true three flower species.
- **Density-Based Performance:** Density-based methods struggled. With the chosen parameters, DBSCAN incorrectly identified only two clusters, while OPTICS performed very poorly, finding five clusters and yielding a negative Silhouette Score, suggesting incorrect cluster assignments.

Wine Dataset Analysis:

- **Clear Winner:** KMeans and KMeans++ were the standout performers, achieving an excellent Adjusted Rand Index (ARI) of ~0.90. This shows a very high correspondence to the true wine classes. Agglomerative (Dendrogram) clustering also performed well (ARI ~0.79).
- **Density-Based Failure:** The density-based methods were unsuccessful on this dataset. DBSCAN failed to find any clusters, and OPTICS performed poorly on all metrics. This highlights that the effectiveness of DBSCAN and OPTICS is highly dependent on parameter tuning (e.g., `eps` and `min_samples`) and the density structure of the data.

Conclusion:

For both the Iris and Wine datasets, the partition-based methods (especially KMeans and KMeans++) and hierarchical clustering proved most effective at identifying the underlying group structures, closely matching the ground-truth labels. The density-based algorithms, with the selected parameters, were not well-suited for these datasets.