|  |  |  |
| --- | --- | --- |
| Assignment No. | Description | Date |
| 1 | K-means Clustering on the Iris Dataset |  |
|  |  |  |
| 2 | K-medoid Clustering on Iris dataset. |  |
|  |  |  |
| 3 | Apply AGNES (single linkage, complete linkage, average linkage) on Iris dataset for clustering. |  |
|  |  |  |
| 4 | Apply DIANA (single linkage, complete linkage, average linkage) on Iris dataset for clustering. |  |
|  |  |  |
| 5 | Draw decision tree using ID3 algorithm on golf playing dataset. |  |
|  |  |  |
| 6 | Apply CART on buy computer dataset. |  |
|  |  |  |
| 7 | Apply naïve Bayesian algorithm on buy computer dataset to identify class label of unknown samples. |  |
|  |  |  |
| 8 | Apply back propagation algorithm on sample {1,0,1} with the class label {1,0}. (Where network topology: 3-2-2-2, all biases and weights are initialized at 0) |  |
|  |  |  |
| 9 | Apply fuzzy c means algorithm on Boston Housing Dataset. |  |
|  |  |  |
| 10 | Apply perceptron for realization of logic gates. (bias= 1) |  |
|  |  |  |
| 11 | Apply Madeline algorithm for Bipolar XOR gates. (Weights are randomly initialized, v0, v1, v2 = 0.5, bias values are set to 1, learning rate = 0.5, Network topology: 2-2-1) |  |
|  |  |  |
|  |  |  |
| 12 | Apply Madeline algorithm for variable network topology. |  |
|  |  |  |
|  |  |  |
| 13 | Propositional logic using prolog |  |
|  |  |  |
| 14 | Realization of TSP using Genetic algorithm |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. **K-means Clustering on the Iris Dataset.**

**Data Description:**

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

SepalLengthCm

SepalWidthCm

PetalLengthCm

PetalWidthCm

**Algorithm:**

Algorithm: K-Means

Input: Unlabeled Dataset, Number of clusters/group(K)

Output: K number of groups of given dataset

Steps:

1. Start.
2. Select randomly K numbers points/centroid from the given dataset.
3. For each point, calculate the distance from the centroids and assign then to the closest centroid cluster.
4. Calculate the variance and place a new centroid of each cluster.
5. Repeat step 2 for the new centroids.
6. If any changes occur on any cluster, then do step 3. else Stop.
7. End.

**Program:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def print\_group(clusters):

for cluster\_id, cluster\_points in clusters.items():

print(f"\nCluster {cluster\_id} ({len(cluster\_points)} points):")

for index, point in cluster\_points:

print(f" Index: {index}, Data Point: {point}")

def calculate\_wcss(dataset, centroids, clusters):

wcss = 0

for cluster\_id, cluster\_points in clusters.items():

centroid = centroids[cluster\_id]

for index, point in cluster\_points:

# Use the data point directly for WCSS calculation

wcss += np.sum((point - centroid) \*\* 2)

return wcss

def kmeans(dataset,no\_of\_centroid,test = 0):

# Ensure dataset is a NumPy array for efficient computation

if isinstance(dataset, pd.DataFrame):

dataset = dataset.to\_numpy()

# Choosing initial centroids

random\_indices = np.random.randint(0,len(dataset),no\_of\_centroid)

centroids = dataset[random\_indices,:]

if test ==0: # If this is not test case then print

print(f"Initial centroids are : {centroids}")

# K means algorithm

clusters = {i: [] for i in range(no\_of\_centroid)} # Creating clusters / groups

tolerance = 1e-2 # Convergence tolerance (distance change between centroids)

shifted = True # Variable to track either centroids are shifted

iter = 0

while shifted:

iter = iter + 1

shifted = False

distances = np.sqrt(np.sum((dataset[:, np.newaxis, :] - centroids[np.newaxis, :, :])\*\*2, axis=2))

if test ==0: # If this is not test case then print

print(f"distance from the centroids:")

print(distances)

labels = np.argmin(distances, axis=1) # Assign points to the nearest centroid

clusters = {i: [] for i in range(no\_of\_centroid)} # Creating clusters / groups

for i, label in enumerate(labels):

clusters[label].append((i, dataset[i])) # Append (index, data point) to the cluster

if test == 0:

print\_group(clusters)

new\_centroids = np.array([np.mean([point[1] for point in clusters[i]], axis=0) if len(clusters[i]) > 0 else centroids[i] for i in range(no\_of\_centroid)])

if test ==0:

print(f"New centroids are : {new\_centroids}")

centroid\_shift = np.linalg.norm(new\_centroids - centroids) # Calculate the Euclidean distance between centroids

# If the centroid shift is smaller than the tolerance, stop

if centroid\_shift >= tolerance:

centroids = new\_centroids

shifted = True

else:

if test ==0:

print(f"Convergence reached after {iter} iterations.")

wcss = calculate\_wcss(dataset, centroids, clusters)

return centroids, clusters, wcss

if \_\_name\_\_ == '\_\_main\_\_':

np.random.seed(42)

dataset\_name = 'Iris.csv'

# Read Input data

csv\_file = pd.read\_csv(f'Dataset/{dataset\_name}') # Read Dataset

dataset = csv\_file.iloc[:,1:-1] # Exclusion of "ID" and "Species" column

# csv\_file = pd.read\_csv('Dataset/kmeans\_test.csv')

# dataset = csv\_file.iloc[:, 1:]

# # K means Clustering

# no\_of\_centroid = 2

# k\_means\_centroids,k\_means\_clusters,wcss = kmeans(dataset,no\_of\_centroid,test = 0)

# K-means Clustering

max\_clusters = 10

wcss\_values = []

for no\_of\_centroid in range(1, max\_clusters + 1):

centroids, clusters, wcss = kmeans(dataset, no\_of\_centroid,test = 1)

wcss\_values.append(wcss)

# Plot WCSS to find the "elbow point"

plt.plot(range(1, max\_clusters + 1), wcss\_values, marker='o')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS')

plt.title('Elbow Method for Optimal K')

plt.savefig(f"Elbow Method for Optimal K {dataset\_name}.png")

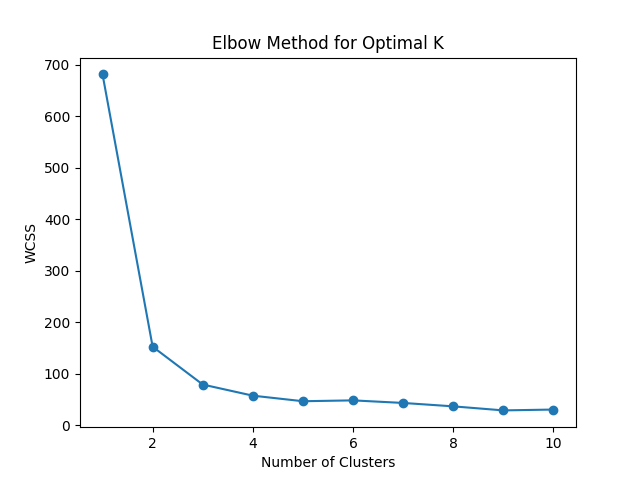
plt.show()

no\_of\_centroid = 2

print(f"After Elbow Method, the optimal number of centroid is {no\_of\_centroid}");

centroids, clusters, wcss = kmeans(dataset, no\_of\_centroid,test = 0)

Output:



Cluster 0 (97 points):

Index: 50, Data Point: [7. 3.2 4.7 1.4]

Index: 51, Data Point: [6.4 3.2 4.5 1.5]

Index: 52, Data Point: [6.9 3.1 4.9 1.5]

Index: 53, Data Point: [5.5 2.3 4. 1.3]

Index: 54, Data Point: [6.5 2.8 4.6 1.5]

Index: 55, Data Point: [5.7 2.8 4.5 1.3]

Index: 56, Data Point: [6.3 3.3 4.7 1.6]

Index: 58, Data Point: [6.6 2.9 4.6 1.3]

Index: 59, Data Point: [5.2 2.7 3.9 1.4]

Index: 60, Data Point: [5. 2. 3.5 1. ]

Index: 61, Data Point: [5.9 3. 4.2 1.5]

Index: 62, Data Point: [6. 2.2 4. 1. ]

Index: 63, Data Point: [6.1 2.9 4.7 1.4]

Index: 64, Data Point: [5.6 2.9 3.6 1.3]

Index: 65, Data Point: [6.7 3.1 4.4 1.4]

Index: 66, Data Point: [5.6 3. 4.5 1.5]

Index: 67, Data Point: [5.8 2.7 4.1 1. ]

Index: 68, Data Point: [6.2 2.2 4.5 1.5]

Index: 69, Data Point: [5.6 2.5 3.9 1.1]

Index: 70, Data Point: [5.9 3.2 4.8 1.8]

Index: 71, Data Point: [6.1 2.8 4. 1.3]

Index: 72, Data Point: [6.3 2.5 4.9 1.5]

Index: 73, Data Point: [6.1 2.8 4.7 1.2]

Index: 74, Data Point: [6.4 2.9 4.3 1.3]

Index: 75, Data Point: [6.6 3. 4.4 1.4]

Index: 76, Data Point: [6.8 2.8 4.8 1.4]

Index: 77, Data Point: [6.7 3. 5. 1.7]

Index: 78, Data Point: [6. 2.9 4.5 1.5]

Index: 79, Data Point: [5.7 2.6 3.5 1. ]

Index: 80, Data Point: [5.5 2.4 3.8 1.1]

Index: 81, Data Point: [5.5 2.4 3.7 1. ]

Index: 82, Data Point: [5.8 2.7 3.9 1.2]

Index: 83, Data Point: [6. 2.7 5.1 1.6]

Index: 84, Data Point: [5.4 3. 4.5 1.5]

Index: 85, Data Point: [6. 3.4 4.5 1.6]

Index: 86, Data Point: [6.7 3.1 4.7 1.5]

Index: 87, Data Point: [6.3 2.3 4.4 1.3]

Index: 88, Data Point: [5.6 3. 4.1 1.3]

Index: 89, Data Point: [5.5 2.5 4. 1.3]

Index: 90, Data Point: [5.5 2.6 4.4 1.2]

Index: 91, Data Point: [6.1 3. 4.6 1.4]

Index: 92, Data Point: [5.8 2.6 4. 1.2]

Index: 94, Data Point: [5.6 2.7 4.2 1.3]

Index: 95, Data Point: [5.7 3. 4.2 1.2]

Index: 96, Data Point: [5.7 2.9 4.2 1.3]

Index: 97, Data Point: [6.2 2.9 4.3 1.3]

Index: 99, Data Point: [5.7 2.8 4.1 1.3]

Index: 100, Data Point: [6.3 3.3 6. 2.5]

Index: 101, Data Point: [5.8 2.7 5.1 1.9]

Index: 102, Data Point: [7.1 3. 5.9 2.1]

Index: 103, Data Point: [6.3 2.9 5.6 1.8]

Index: 104, Data Point: [6.5 3. 5.8 2.2]

Index: 105, Data Point: [7.6 3. 6.6 2.1]

Index: 106, Data Point: [4.9 2.5 4.5 1.7]

Index: 107, Data Point: [7.3 2.9 6.3 1.8]

Index: 108, Data Point: [6.7 2.5 5.8 1.8]

Index: 109, Data Point: [7.2 3.6 6.1 2.5]

Index: 110, Data Point: [6.5 3.2 5.1 2. ]

Index: 111, Data Point: [6.4 2.7 5.3 1.9]

Index: 112, Data Point: [6.8 3. 5.5 2.1]

Index: 113, Data Point: [5.7 2.5 5. 2. ]

Index: 114, Data Point: [5.8 2.8 5.1 2.4]

Index: 115, Data Point: [6.4 3.2 5.3 2.3]

Index: 116, Data Point: [6.5 3. 5.5 1.8]

Index: 117, Data Point: [7.7 3.8 6.7 2.2]

Index: 118, Data Point: [7.7 2.6 6.9 2.3]

Index: 119, Data Point: [6. 2.2 5. 1.5]

Index: 120, Data Point: [6.9 3.2 5.7 2.3]

Index: 121, Data Point: [5.6 2.8 4.9 2. ]

Index: 122, Data Point: [7.7 2.8 6.7 2. ]

Index: 123, Data Point: [6.3 2.7 4.9 1.8]

Index: 124, Data Point: [6.7 3.3 5.7 2.1]

Index: 125, Data Point: [7.2 3.2 6. 1.8]

Index: 126, Data Point: [6.2 2.8 4.8 1.8]

Index: 127, Data Point: [6.1 3. 4.9 1.8]

Index: 128, Data Point: [6.4 2.8 5.6 2.1]

Index: 129, Data Point: [7.2 3. 5.8 1.6]

Index: 130, Data Point: [7.4 2.8 6.1 1.9]

Index: 131, Data Point: [7.9 3.8 6.4 2. ]

Index: 132, Data Point: [6.4 2.8 5.6 2.2]

Index: 133, Data Point: [6.3 2.8 5.1 1.5]

Index: 134, Data Point: [6.1 2.6 5.6 1.4]

Index: 135, Data Point: [7.7 3. 6.1 2.3]

Index: 136, Data Point: [6.3 3.4 5.6 2.4]

Index: 137, Data Point: [6.4 3.1 5.5 1.8]

Index: 138, Data Point: [6. 3. 4.8 1.8]

Index: 139, Data Point: [6.9 3.1 5.4 2.1]

Index: 140, Data Point: [6.7 3.1 5.6 2.4]

Index: 141, Data Point: [6.9 3.1 5.1 2.3]

Index: 142, Data Point: [5.8 2.7 5.1 1.9]

Index: 143, Data Point: [6.8 3.2 5.9 2.3]

Index: 144, Data Point: [6.7 3.3 5.7 2.5]

Index: 145, Data Point: [6.7 3. 5.2 2.3]

Index: 146, Data Point: [6.3 2.5 5. 1.9]

Index: 147, Data Point: [6.5 3. 5.2 2. ]

Index: 148, Data Point: [6.2 3.4 5.4 2.3]

Index: 149, Data Point: [5.9 3. 5.1 1.8]

Cluster 1 (53 points):

Index: 0, Data Point: [5.1 3.5 1.4 0.2]

Index: 1, Data Point: [4.9 3. 1.4 0.2]

Index: 2, Data Point: [4.7 3.2 1.3 0.2]

Index: 3, Data Point: [4.6 3.1 1.5 0.2]

Index: 4, Data Point: [5. 3.6 1.4 0.2]

Index: 5, Data Point: [5.4 3.9 1.7 0.4]

Index: 6, Data Point: [4.6 3.4 1.4 0.3]

Index: 7, Data Point: [5. 3.4 1.5 0.2]

Index: 8, Data Point: [4.4 2.9 1.4 0.2]

Index: 9, Data Point: [4.9 3.1 1.5 0.1]

Index: 10, Data Point: [5.4 3.7 1.5 0.2]

Index: 11, Data Point: [4.8 3.4 1.6 0.2]

Index: 12, Data Point: [4.8 3. 1.4 0.1]

Index: 13, Data Point: [4.3 3. 1.1 0.1]

Index: 14, Data Point: [5.8 4. 1.2 0.2]

Index: 15, Data Point: [5.7 4.4 1.5 0.4]

Index: 16, Data Point: [5.4 3.9 1.3 0.4]

Index: 17, Data Point: [5.1 3.5 1.4 0.3]

Index: 18, Data Point: [5.7 3.8 1.7 0.3]

Index: 19, Data Point: [5.1 3.8 1.5 0.3]

Index: 20, Data Point: [5.4 3.4 1.7 0.2]

Index: 21, Data Point: [5.1 3.7 1.5 0.4]

Index: 22, Data Point: [4.6 3.6 1. 0.2]

Index: 23, Data Point: [5.1 3.3 1.7 0.5]

Index: 24, Data Point: [4.8 3.4 1.9 0.2]

Index: 25, Data Point: [5. 3. 1.6 0.2]

Index: 26, Data Point: [5. 3.4 1.6 0.4]

Index: 27, Data Point: [5.2 3.5 1.5 0.2]

Index: 28, Data Point: [5.2 3.4 1.4 0.2]

Index: 29, Data Point: [4.7 3.2 1.6 0.2]

Index: 30, Data Point: [4.8 3.1 1.6 0.2]

Index: 31, Data Point: [5.4 3.4 1.5 0.4]

Index: 32, Data Point: [5.2 4.1 1.5 0.1]

Index: 33, Data Point: [5.5 4.2 1.4 0.2]

Index: 34, Data Point: [4.9 3.1 1.5 0.1]

Index: 35, Data Point: [5. 3.2 1.2 0.2]

Index: 36, Data Point: [5.5 3.5 1.3 0.2]

Index: 37, Data Point: [4.9 3.1 1.5 0.1]

Index: 38, Data Point: [4.4 3. 1.3 0.2]

Index: 39, Data Point: [5.1 3.4 1.5 0.2]

Index: 40, Data Point: [5. 3.5 1.3 0.3]

Index: 41, Data Point: [4.5 2.3 1.3 0.3]

Index: 42, Data Point: [4.4 3.2 1.3 0.2]

Index: 43, Data Point: [5. 3.5 1.6 0.6]

Index: 44, Data Point: [5.1 3.8 1.9 0.4]

Index: 45, Data Point: [4.8 3. 1.4 0.3]

Index: 46, Data Point: [5.1 3.8 1.6 0.2]

Index: 47, Data Point: [4.6 3.2 1.4 0.2]

Index: 48, Data Point: [5.3 3.7 1.5 0.2]

Index: 49, Data Point: [5. 3.3 1.4 0.2]

Index: 57, Data Point: [4.9 2.4 3.3 1. ]

Index: 93, Data Point: [5. 2.3 3.3 1. ]

Index: 98, Data Point: [5.1 2.5 3. 1.1]

New centroids are : [[6.30103093 2.88659794 4.95876289 1.69587629]

[5.00566038 3.36037736 1.56226415 0.28867925]]

Convergence reached after 5 iterations.

\*\*This is the final clusters only.

**Conclusion:**

K-Means Clustering is an Unsupervised Machine Learning algorithm which groups the unlabeled dataset into different clusters. Here Iris dataset is clustered in 2 groups as per the elbow finding method.

1. **K-medoids Clustering on the Iris Dataset.**

**Data Description:**

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

1. SepalLengthCm
2. SepalWidthCm
3. PetalLengthCm
4. PetalWidthCm

**Algorithm:**

1. Start with an initial set of medoids (selected randomly or using an initialization method).
2. Iteratively:
   * Replace a medoid with a non-medoid data point if it reduces the total sum of distances (sum of squared errors, SSE) within the resulting cluster.
   * Continue this process until no further improvement can be made.

The objective is to minimize the Sum of Squared Errors (SSE), which is defined as:

**Program:**

import numpy as np

from sklearn\_extra.cluster import KMedoids

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import pandas as pd

# Sample data with 4 columns

dataset = pd.read\_csv("/content/Iris.csv")

X\_data = dataset.iloc[:,1:-1]

data = X\_data.to\_numpy()

# Step 1: Normalize the data

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

clustering\_score = []

for i in range(1, 11):

kmedoids = KMedoids(n\_clusters = i, init = 'random', random\_state = 42)

kmedoids.fit(data\_scaled)

clustering\_score.append(kmedoids.inertia\_) # inertia\_ = Sum of squared distances of samples to their closest cluster center.

plt.figure(figsize=(10,6))

plt.plot(range(1, 11), clustering\_score)

plt.scatter(5,clustering\_score[4], s = 200, c = 'red', marker='\*')

plt.title('The Elbow Method')

plt.xlabel('No. of Clusters')

plt.ylabel('Clustering Score')

plt.show()

# Step 2: Apply K-medoids

k = 5 # Choose the number of clusters

kmedoids = KMedoids(n\_clusters=k, metric='euclidean', random\_state=42)

labels = kmedoids.fit\_predict(data\_scaled)

# Step 3: Evaluate the clustering

print("Cluster labels:", labels)

print("Medoids:", kmedoids.cluster\_centers\_)

print("Silhouette Score:", silhouette\_score(data\_scaled, labels))

# Step 4: Reduce dimensions for visualization

pca = PCA(n\_components=2)

data\_2d = pca.fit\_transform(data\_scaled)

# Step 5: Plot the clusters

plt.figure(figsize=(8, 6))

for cluster in np.unique(labels):

plt.scatter(data\_2d[labels == cluster, 0], data\_2d[labels == cluster, 1], label=f'Cluster {cluster}')

# Plot the medoids

medoid\_2d = pca.transform(kmedoids.cluster\_centers\_)

plt.scatter(medoid\_2d[:, 0], medoid\_2d[:, 1], c='red', marker='X', s=200, label='Medoids')

plt.title('K-Medoids Clustering Visualization')

plt.xlabel('PCA Component 1')

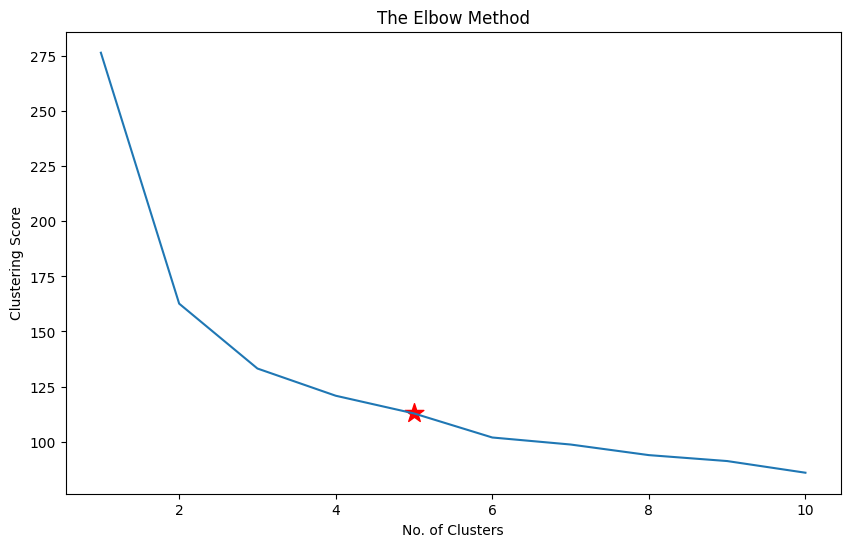
plt.ylabel('PCA Component 2')

plt.legend()

plt.grid()

plt.show()

**Output:**

****

Cluster labels: [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

4 4 4 4 3 4 4 4 4 4 4 4 4 2 0 2 3 1 0 0 3 0 3 3 0 3 0 0 0 0 3 3 3 0 0 1 0

0 0 1 2 0 3 3 3 3 1 0 0 0 3 0 3 3 0 3 3 3 0 0 0 3 0 2 1 2 1 2 2 3 2 1 2 2

1 2 1 1 2 2 2 2 3 2 1 2 1 2 2 1 1 1 2 2 2 1 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2

2 1]

Medoids: [[ 0.31099753 -0.1249576 0.47843012 0.26469891]

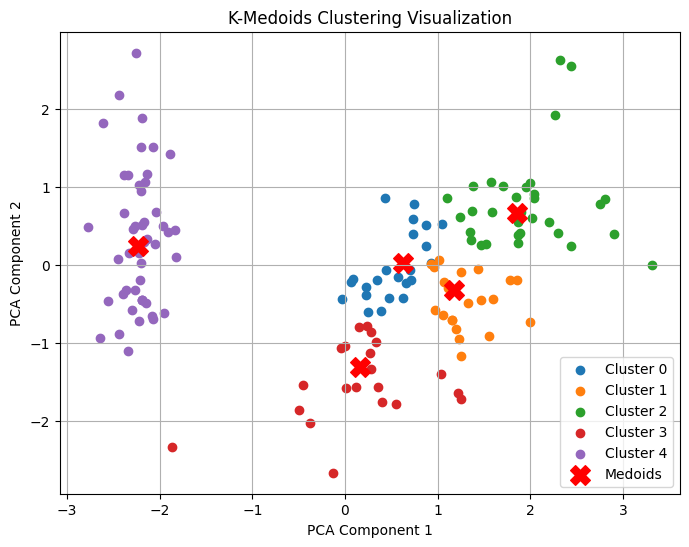
[ 0.4321654 -0.58776353 0.59216153 0.79059079]

[ 1.2803405 0.10644536 0.93335575 1.1850097 ]

[-0.29484182 -1.28197243 0.08037019 -0.12972 ]

[-1.02184904 0.80065426 -1.2844067 -1.31297673]]

Silhouette Score: 0.3759993156661742

****

**Conclusion:**

K-Medoids clustering mechanism in Partition Clustering. First, Clustering is the process of breaking down an abstract group of data points/ objects into classes of similar objects such that all the objects in one cluster have similar traits. A group of n objects is broken down into k number of clusters based on their similarities. Here I have use PCA(Principal Component Analysis) for dimension reduction, which makes it easier to represent in 2D space.

1. **Apply AGNES (single linkage, complete linkage, average linkage) on Iris dataset for clustering.**

**Data Description:**

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

1. SepalLengthCm
2. SepalWidthCm
3. PetalLengthCm
4. PetalWidthCm

**Algorithm:**

**Input:** Dataset Iris

**Output:** Dendrogram of the dataset

#### Steps:

1. **Initialization:**
   * Treat each data point as a separate cluster.
2. **Compute Distance Matrix:**
   * Calculate the distance between each pair of clusters. The distance can be defined using various metrics such as Euclidean distance, cosine similarity, etc.
3. **Merge Closest Clusters:**
   * Identify the two clusters that are closest (based on the distance metric) and merge them into a single cluster.
4. **Update Distance Matrix:**
   * After merging, update the distance matrix to reflect the distance between the newly formed cluster and the remaining clusters.
5. **Repeat:**
   * Repeat steps 3 and 4 until the stopping condition is reached.
6. **Result:**
   * The result is a **dendrogram**, which is a tree-like diagram that shows the arrangement of clusters at different levels of similarity.

**Source Code:**

import numpy as np

import pandas as pd

from sklearn import datasets

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import AgglomerativeClustering

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from scipy.cluster.hierarchy import linkage, dendrogram

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data  # Features (Sepal Length, Sepal Width, Petal Length, Petal Width)

y = iris.target  # True labels (not used in unsupervised learning)

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Define a function to apply AGNES with different linkage methods

def agnes\_clustering(linkage\_method):

    agnes = AgglomerativeClustering(n\_clusters=3, linkage=linkage\_method)

    cluster\_labels = agnes.fit\_predict(X\_scaled)

    # Compute linkage matrix for dendrogram

    Z = linkage(X\_scaled, method=linkage\_method)

    # Reduce dimensionality for visualization using PCA

    pca = PCA(n\_components=2)

    X\_pca = pca.fit\_transform(X\_scaled)

    # Plot the clusters

    plt.figure(figsize=(8, 6))

    for cluster in range(3):

        plt.scatter(

            X\_pca[np.where(cluster\_labels == cluster), 0],  # Fixed indexing issue

            X\_pca[np.where(cluster\_labels == cluster), 1],

            label=f"Cluster {cluster + 1}"

        )

    plt.title(f"AGNES Clustering ({linkage\_method.capitalize()} Linkage)")

    plt.xlabel("Principal Component 1")

    plt.ylabel("Principal Component 2")

    plt.legend()

    plt.show()

    # Plot the dendrogram

    plt.figure(figsize=(10, 6))

    dendrogram(Z)

    plt.title(f"Dendrogram ({linkage\_method.capitalize()} Linkage)")

    plt.xlabel("Data Points")

    plt.ylabel("Distance")

    plt.show()

# Apply AGNES with single linkage

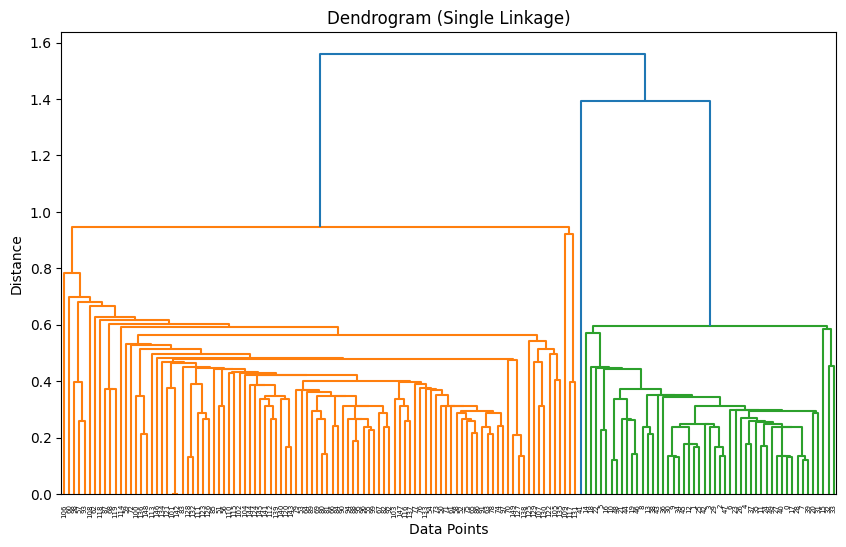
agnes\_clustering("single")

# Apply AGNES with complete linkage

agnes\_clustering("complete")

# Apply AGNES with average linkage

agnes\_clustering("average")

**Output:**

