# Clustering

**Clustering** is an unsupervised machine learning technique used to group a set of objects into clusters or groups based on their similarities. The main idea is that objects within the same cluster are more similar to each other than to those in other clusters. In other words, it is a way of discovering inherent structures or patterns in a dataset without using labeled data.

Clustering is often used in tasks where we want to identify groups or patterns in data but do not have predefined labels. It is widely applied in areas like customer segmentation, image recognition, and anomaly detection.

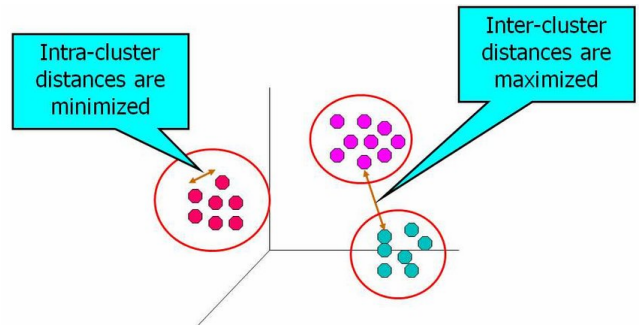
**Key Concepts in Clustering:**

1. **Similarity or Dissimilarity:** Clustering relies on a measure of similarity (or distance) between objects. Common measures of similarity/distance include:
    - **Euclidean Distance:** The straight-line distance between two points in a Euclidean space.
    - **Cosine Similarity:** Measures the angle between two vectors, often used in text analysis.
    - **Manhattan Distance:** The sum of the absolute differences of coordinates.
2. **Centroids:** In some clustering algorithms, such as K-means, a "centroid" represents the center of a cluster. It is the mean of all points in that cluster.
3. **Clusters:** A cluster is a group of objects that share common characteristics.

**What is good clustering?**

Good clustering refers to forming groups (clusters) such that:

1. **Intra-cluster similarity is high**: Data points within the same cluster are as similar as possible (minimizing the distance between points and their cluster centroid). This is called **compactness**.
2. **Inter-cluster dissimilarity is high**: Different clusters are well-separated from each other, meaning the distance between cluster centroids is maximized. This ensures **distinctiveness**.




**Types of Clustering Algorithms:**

1. **K-means Clustering:**
    - Partitions the data into **K** clusters based on a predefined number of clusters (K).
    - Assigns each point to the nearest cluster centroid and updates centroids iteratively.
2. **Hierarchical Clustering:**
    - Builds a hierarchy of clusters either in a bottom-up (agglomerative) or top-down (divisive) manner.
    - No need to specify the number of clusters beforehand.
3. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):**
    - Groups together points that are closely packed together while marking points in low-density regions as outliers.
    - Does not require the number of clusters to be specified.
4. **Gaussian Mixture Models (GMM):**
    - Assumes that the data is generated from a mixture of several Gaussian distributions (normal distributions) and clusters points based on probability.
5. **Mean Shift:**
    - A sliding window technique that shifts points towards the densest region of data points and forms clusters based on the density of the data.

**Applications of Clustering:**

- **Customer Segmentation:** Grouping customers based on purchasing behavior, demographics, etc.
- **Image Segmentation:** Dividing an image into distinct regions for further analysis.
- **Anomaly Detection:** Identifying unusual or outlier behavior in datasets (e.g., fraud detection).
- **Recommendation Systems:** Grouping similar items together to recommend products based on past preferences.

Explanation of Distance Metrics

## 1. Euclidean Distance

Euclidean distance is the most common distance metric, often referred to as the "straight-line distance" or "as-the-crow-flies" distance between two points in a multi-dimensional space.

**Formula:**

For two points $P_1 = (x_1, y_1, \ldots, x_n)$ and $P_2 = (y_1, y_2, \ldots, y_n)$, the Euclidean distance $d(P_1, P_2)$ in an $n$-dimensional space is:

$$d(P_1, P_2) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

**Example:**

For points $P_1 = (1, 2)$ and $P_2 = (4, 6)$ in 2D:

$$d(P_1, P_2) = \sqrt{(4-1)^2 + (6-2)^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

The Euclidean distance is **5**.

**Usage:** It's most suitable when the data is continuous, and you're measuring the "direct" distance between points, like geographic distances.

## 2. Cosine Similarity

Cosine similarity measures the **angle** between two vectors, focusing on the **direction** rather than the magnitude. It is used to measure how similar two vectors are, especially useful for high-dimensional spaces like text analysis (e.g., in document comparison).

**Formula:**

For two vectors $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$, the cosine similarity is:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\| \|B\|}$$

Where:

- $A \cdot B = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$ is the dot product.
- $\|A\| = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$ is the magnitude (norm) of vector $A$.
- $\|B\| = \sqrt{b_1^2 + b_2^2 + \ldots + b_n^2}$ is the magnitude (norm) of vector $B$.

The result ranges from **-1** (completely opposite) to **1** (completely similar), with **0** indicating orthogonal (no similarity).

**Example:**

For vectors $A = (1, 2)$ and $B = (2, 3)$:

- Dot product: $A \cdot B = 1 \times 2 + 2 \times 3 = 2 + 6 = 8$
- Magnitudes: $\|A\| = \sqrt{1^2 + 2^2} = \sqrt{5}$, $\|B\| = \sqrt{2^2 + 3^2} = \sqrt{13}$

Cosine similarity:

$$\text{Cosine Similarity} = \frac{8}{\sqrt{5} \times \sqrt{13}} = \frac{8}{\sqrt{65}} \approx \frac{8}{8.06} \approx 0.993$$

This indicates that the vectors are very similar.

**Usage:** Cosine similarity is often used in text mining (e.g., comparing documents) because it focuses on the **orientation** of vectors, making it invariant to the length (magnitude) of the vectors.

## 3. Manhattan Distance (L1 Distance)

Manhattan distance, also called **taxicab** or **city block distance**, is the sum of the absolute differences between the coordinates of two points. It is called "Manhattan" because it resembles the grid-like street pattern of the Manhattan borough in New York City (where you can only move along the grid, not diagonally).

**Formula:**

For two points $P_1 = (x_1, y_1, \ldots, x_n)$ and $P_2 = (y_1, y_2, \ldots, y_n)$, the Manhattan distance is:

$$d(P_1, P_2) = \sum_{i=1}^{n} |x_i - y_i|$$

**Example:**

For points $P_1 = (1, 2)$ and $P_2 = (4, 6)$ in 2D:

$$d(P_1, P_2) = |4 - 1| + |6 - 2| = 3 + 4 = 7$$

The Manhattan distance is **7**.

**Usage:** Manhattan distance is suitable for cases where movement is restricted to horizontal and vertical directions (such as grid-based or lattice data).

**K-Means Clustering**

**K-means** is a popular unsupervised machine learning algorithm used to partition a dataset into **K** distinct clusters, where each cluster contains data points that are similar to each other. The goal is to minimize the **within-cluster variance** (or the sum of squared distances between points and their respective cluster centroids).

**Steps in K-Means Clustering:**

1. **Initialize K centroids**:
    - Choose **K** random points from the dataset as initial centroids (the centers of clusters).
2. **Assign points to the nearest centroid**:

    - For each data point, calculate its distance to each of the K centroids and assign the point to the nearest centroid.
3. **Recalculate centroids**:
    - Once all points are assigned to a cluster, calculate the new centroid by taking the mean of all the points in that cluster.
4. **Repeat steps 2 and 3**:
    - Reassign points to the new centroids, then recalculate the centroids. Continue this process until the centroids do not change significantly, meaning the algorithm has converged.
5. **End**:
    - The algorithm stops when the centroids no longer change, or the number of iterations exceeds a specified limit

Q. Apply K(=2)-Means algorithm over the data (185, 72), (170, 56), (168, 60), (179,68), (182,72), (188,77) up to two iterations and show the clusters. Initially choose first two objects as initial centroids.

*Solution:*

Given, number of clusters to be created (K) = 2 say c1 and c2,

number of iterations = 2 and

The given data points can be represented in tabular form as:

| Instance | X | Y |
|---|---|---|
| **1** | 185 | 72 |
| **2** | 170 | 56 |
| **3** | 168 | 60 |
| **4** | 179 | 68 |
| **5** | 182 | 72 |
| **6** | 188 | 77 |

also, first two objects as initial centroids:
Centroid for first cluster c1 = (185, 72)
Centroid for second cluster c2 = (170, 56)

***Iteration 1:*** Now calculating similarity by using *Euclidean distance* measure as:

$d(c1, 3) = \sqrt{(185-168)^2 + (72-60)^2} = \sqrt{(17)^2 + (12)^2} = \sqrt{289 + 144} = \sqrt{433}$

$d(c2, 3) = \sqrt{(170-168)^2 + (56-60)^2} = \sqrt{(2)^2 + (-4)^2} = \sqrt{4 + 16} = \sqrt{20}$

Here, d(c2, 3) < d(c1, 3)

So, data point 3 belongs to c2.

$d(c1, 4) = \sqrt{(185-179)^2 + (72-68)^2} = \sqrt{(6)^2 + (4)^2} = \sqrt{36 + 16} = \sqrt{52}$

$d(c2, 4) = \sqrt{(170-179)^2 + (56-68)^2} = \sqrt{(-9)^2 + (-12)^2} = \sqrt{81 + 144} = \sqrt{225}$

Here, d(c1, 4) < d(c2, 4)

So, data point 4 belongs to c1.

$d(c1, 5) = \sqrt{(185-182)^2 + (72-72)^2} = \sqrt{(3)^2 + (0)^2} = \sqrt{9}$

$d(c2, 5) = \sqrt{(170-182)^2 + (56-72)^2} = \sqrt{(-12)^2 + (-16)^2} = \sqrt{144 + 256} = \sqrt{400}$

Here, d(c1, 5) < d(c2, 5)

So, data point 5 belongs to c1.

$d(c1, 6) = \sqrt{(185-188)^2 + (72-77)^2} = \sqrt{(-3)^2 + (-5)^2} = \sqrt{9 + 25} = \sqrt{34}$

$d(c2, 6) = \sqrt{(170-188)^2 + (56-77)^2} = \sqrt{(-18)^2 + (-21)^2} = \sqrt{324 + 441} = \sqrt{765}$

Here, d(c1, 6) < d(c2, 6)

So, data point 6 belongs to c1.

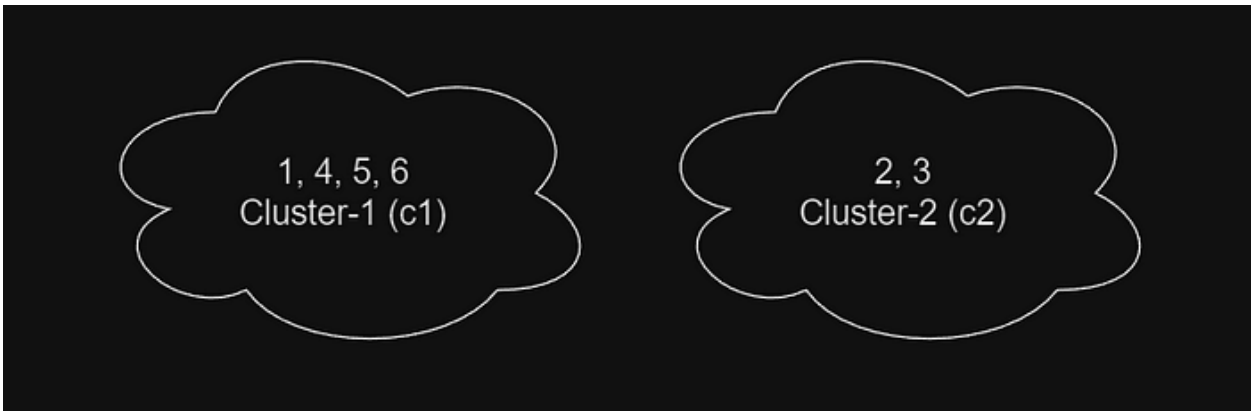
Euclidean distance calculation

Representing above information in tabular form:

| Instance | X | Y | Distance(C1) | Distance(C2) | Cluster |
|---|---|---|---|---|---|
| 1 | 185 | 72 | | | c1 |
| 2 | 170 | 56 | | | c2 |
| 3 | 168 | 60 | √433 | √20 | c2 |
| 4 | 179 | 68 | √52 | √225 | c1 |
| 5 | 182 | 72 | √9 | √400 | c1 |
| 6 | 188 | 77 | √34 | √765 | c1 |

Distance of each data points from cluster centroids

The resulting cluster after first iteration is:




Data points cluster

***Iteration 2:*** Now calculating centroid for each cluster:

$$Centroid\ for\ first\ cluster\ c1 = \left(\frac{185+179+182+188}{4}, \frac{72+68+72+77}{4}\right) = (183.5, 72.25)$$

$$Centroid\ for\ second\ cluster\ c2 = \left(\frac{170+168}{2}, \frac{56+60}{2}\right) = (169, 58)$$

Calculating centroid as mean of data points

Now, again calculating similarity:

$$d(c1,1) = \sqrt{(183.5-185)^2 + (72.25-72)^2} = 1.5207$$
$$d(c2,1) = \sqrt{(169-185)^2 + (58-72)^2} = 21.2603$$

Here, d (c1, 1) < d (c2, 1)
So, data point 1 belongs to c1.

$$d(c1,2) = \sqrt{(183.5-170)^2 + (72.25-56)^2} = 21.1261$$
$$d(c2,2) = \sqrt{(169-170)^2 + (58-56)^2} = 2.2361$$

Here, d (c2, 2) < d (c1, 2)
So, data point 2 belongs to c2.

$$d(c1,3) = \sqrt{(183.5-168)^2 + (72.25-60)^2} = 19.7563$$
$$d(c2,3) = \sqrt{(169-168)^2 + (58-60)^2} = 2.2361$$

Here, d (c2, 3) < d (c1, 3)
So, data point 3 belongs to c2.

$$d(c1,4) = \sqrt{(183.5-179)^2 + (72.25-68)^2} = 6.1897$$
$$d(c2,4) = \sqrt{(169-179)^2 + (58-68)^2} = 14.1421$$

Here, d (c1, 4) < d (c2, 4)
So, data point 4 belongs to c1.

$$d(c1,5) = \sqrt{(183.5-182)^2 + (72.25-72)^2} = 1.5207$$
$$d(c2,5) = \sqrt{(169-182)^2 + (58-72)^2} = 19.1050$$

Here, d (c1, 5) < d (c2, 5)
So, data point 5 belongs to c1.

$$d(c1,6) = \sqrt{(183.5-188)^2 + (72.25-77)^2} = 6.5431$$
$$d(c2,6) = \sqrt{(169-188)^2 + (58-77)^2} = 26.8701$$

Here, d (c1, 6) < d (c2, 6)
So, data point 6 belongs to c1.

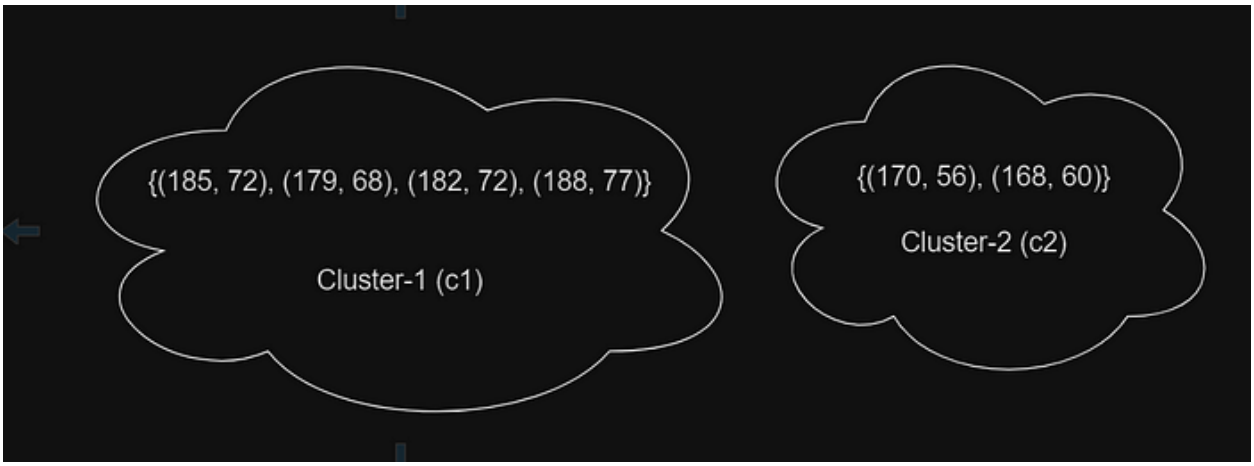
Distance calculation between data points and centroids

Representing above information in tabular form.

| Instance | X | Y | Distance(C1) | Distance(C2) | Cluster |
|---|---|---|---|---|---|
| **1** | 185 | 72 | 1.5207 | 21.2603 | c1 |
| **2** | 170 | 56 | 21.1261 | 2.2361 | c2 |
| **3** | 168 | 60 | 19.7563 | 2.2361 | c2 |
| **4** | 179 | 68 | 6.1897 | 14.1421 | c1 |
| **5** | 182 | 72 | 1.5207 | 19.105 | c1 |
| **6** | 188 | 77 | 6.5431 | 26.8701 | c1 |

Distance of each data points from cluster centroids

The resulting cluster after second iteration is:




Data points cluster

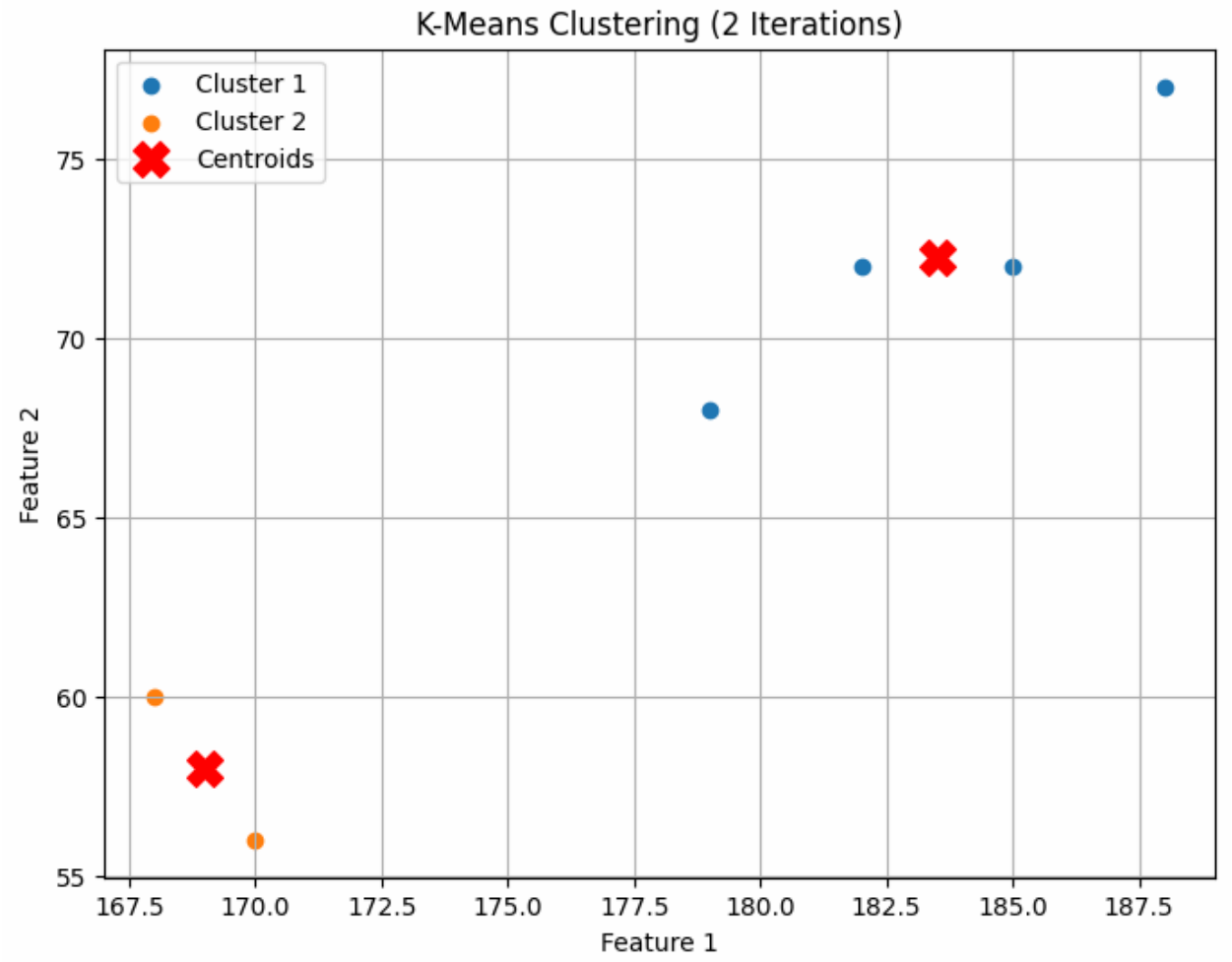
As we have already completed two iteration as asked by our question, the numerical ends here.

Since, the clustering doesn’t change after second iteration, so terminate the iteration

even if question doesn’t say so.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Data points
data = np.array([[185, 72], [170, 56], [168, 60], [179, 68], [182, 72], [188, 77]])

# Define initial centroids
initial_centroids = np.array([[185, 72], [170, 56]])

# Create a KMeans instance with n_clusters=2 and max_iter=2
kmeans = KMeans(n_clusters=2, init=initial_centroids, n_init=1, max_iter=2,
random_state=42)

# Fit the model to the data
kmeans.fit(data)

# Retrieve cluster labels and centroids after two iterations
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Visualize the clusters and centroids
plt.figure(figsize=(8, 6))
for cluster_id in range(2):
    cluster_points = data[labels == cluster_id]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {cluster_id + 1}')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label='Centroids')
plt.title('K-Means Clustering (2 Iterations)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
```

```
plt.grid(True)
plt.show()
```



## 3.7. **Sum of Squared Error (SSE), Silhouette Coefficient, and Dunn Index**

In clustering, we need to evaluate how well our algorithm has performed. Various metrics can be used to assess the quality of clusters, and three of the most common ones are **Sum of Squared Error (SSE)**, **Silhouette Coefficient**, and **Dunn Index**.

### 1. **Sum of Squared Error (SSE)**

- **Definition**: SSE (also called **Within-Cluster Sum of Squares** or **WCSS**) measures the total squared distance between each data point and the centroid of its assigned cluster. It quantifies how compact the clusters are.
- **Formula**:

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} \left(\text{distance}(x_i, c_j)\right)^2$$

  Where:

  - $n$ is the number of data points.
  - $k$ is the number of clusters.
  - $x_i$ is the $i$-th data point.
  - $c_j$ is the centroid of the $j$-th cluster.

- **Interpretation**:
  - A **lower SSE** indicates that the clusters are more compact and better defined, meaning that the data points are closer to their centroids.
  - A **higher SSE** suggests that the clusters are more spread out and less well-defined.
- **Use in K-Means**: SSE is often used in the **Elbow Method** to determine the optimal number of clusters. As $k$ increases, SSE typically decreases, but the rate of decrease slows down after a certain point (the "elbow").

2. **Silhouette Coefficient**

- **Definition**: The **Silhouette Coefficient** is a measure of how similar a data point is to its own cluster compared to other clusters. It combines both cohesion (how close points are within a cluster) and separation (how far apart clusters are).
- **Formula**:

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Where:

- $a(i)$ is the average distance between point $i$ and all other points in the same cluster.
- $b(i)$ is the average distance between point $i$ and all points in the nearest cluster.
- $S(i)$ ranges from $-1$ to $+1$:
  - **+1**: The point is well clustered (far from other clusters).
  - **0**: The point is on or near the decision boundary between two clusters.
  - **-1**: The point is misclassified.

- **Interpretation**:
  - **Higher Silhouette Scores**: Indicate better-defined clusters with points that are well-separated from other clusters.
  - **Lower Silhouette Scores**: Indicate poorly defined clusters, where points might be close to points from other clusters.
- **Use**: Silhouette Coefficient is a good overall measure for cluster quality. It can be used to compare different clustering solutions, and a higher average Silhouette Score suggests better clustering.

3. **Dunn Index**

- **Definition**: The **Dunn Index** is another clustering evaluation metric that aims to identify clusters that are well-separated and compact. It is defined as the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance.
- **Formula**:

$$\text{Dunn Index} = \frac{\min_{i \neq j} \left( \text{distance}(C_i, C_j) \right)}{\max_i \left( \text{diameter}(C_i) \right)}$$

  Where:

  - $C_i$ and $C_j$ are two different clusters.
  - $\text{distance}(C_i, C_j)$ is the distance between the centroids of clusters $C_i$ and $C_j$.
  - $\text{diameter}(C_i)$ is the maximum distance between any two points within cluster $C_i$.
- **Interpretation**:
  - A **higher Dunn Index** indicates that the clusters are well-separated (larger inter-cluster distance) and compact (smaller intra-cluster diameter).
  - A **lower Dunn Index** suggests that the clusters overlap or are poorly separated, indicating weak clustering.
- **Use**: The Dunn Index is useful for comparing different clustering algorithms or parameter settings. However, it is sensitive to the scale of the data and the number of clusters.

## Summary of Metrics:

| Metric | What it Measures | Range | Ideal Value |
|---|---|---|---|
| **SSE (Sum of Squared Error)** | Measures compactness of clusters. Lower is better. | [0, ∞) | Lower is better |
| **Silhouette Coefficient** | Measures both cohesion and separation of clusters. | [-1, 1] | Higher is better |
| **Dunn Index** | Measures cluster separation and compactness. Higher is better. | [0, ∞) | Higher is better |

## Use Cases:

- **SSE**: Primarily used in the Elbow Method to determine the optimal number of clusters.
- **Silhouette Coefficient**: Used to evaluate and compare the quality of clustering, especially when the number of clusters is already known.
- **Dunn Index**: Used for comparing clustering algorithms, especially when the goal is to have well-separated and compact clusters.

Let's go through a mathematical example of calculating **Sum of Squared Error (SSE)**, **Silhouette Coefficient**, and **Dunn Index** for two clusters. We'll start with some simple data points.

## Given Data:

We have 6 data points that we want to cluster into 2 clusters.

| Data Point | Feature 1 (x) | Feature 2 (y) |
|---|---|---|
| A | 2 | 3 |
| B | 3 | 3 |
| C | 6 | 7 |
| D | 7 | 8 |
| E | 5 | 6 |
| F | 8 | 8 |

## Step 1: K-Means Clustering (k=2)

Let's assume we run **K-Means** and get the following clusters:

- **Cluster 1**: {A, B, E}
- **Cluster 2**: {C, D, F}

## Step 2: Calculate Sum of Squared Error (SSE)

SSE measures the sum of squared distances between each point and its cluster centroid. First, we need to calculate the centroids for each cluster.

**Centroid of Cluster 1 (C1):**

- Points: A(2, 3), B(3, 3), E(5, 6)
- Centroid $C_1 = \left(\frac{2+3+5}{3}, \frac{3+3+6}{3}\right) = \left(\frac{10}{3}, \frac{12}{3}\right) = (3.33, 4)$

**Centroid of Cluster 2 (C2):**

- Points: C(6, 7), D(7, 8), F(8, 8)
- Centroid $C_2 = \left(\frac{6+7+8}{3}, \frac{7+8+8}{3}\right) = \left(\frac{21}{3}, \frac{23}{3}\right) = (7, 7.67)$

Now, we calculate the **SSE** for each cluster:

**SSE for Cluster 1:**

$$SSE_1 = \sum_{i \in \text{Cluster 1}} (\text{distance}(x_i, C_1))^2$$

- For point A(2, 3):

$$\text{distance}(A, C_1) = \sqrt{(2-3.33)^2 + (3-4)^2} = \sqrt{(-1.33)^2 + (-1)^2} = \sqrt{1.7689 + 1} = \sqrt{2.7689} \approx 1.664$$

$$\text{SSE for A} = 1.664^2 \approx 2.77$$

- For point B(3, 3):

$$\text{distance}(B, C_1) = \sqrt{(3-3.33)^2 + (3-4)^2} = \sqrt{(-0.33)^2 + (-1)^2} = \sqrt{0.1089 + 1} = \sqrt{1.1089} \approx 1.054$$

$$\text{SSE for B} = 1.054^2 \approx 1.11$$

- For point E(5, 6):

$$\text{distance}(E, C_1) = \sqrt{(5-3.33)^2 + (6-4)^2} = \sqrt{(1.67)^2 + (2)^2} = \sqrt{2.7889 + 4} = \sqrt{6.7889} \approx 2.607$$

$$\text{SSE for E} = 2.607^2 \approx 6.80$$

**Total SSE for Cluster 1:**

$$SSE_1 = 2.77 + 1.11 + 6.80 = 10.68$$

**SSE for Cluster 2:**

$$SSE_2 = \sum_{i \in \text{Cluster 2}} (\text{distance}(x_i, C_2))^2$$

- For point C(6, 7):

$$\text{distance}(C, C_2) = \sqrt{(6-7)^2 + (7-7.67)^2} = \sqrt{(-1)^2 + (-0.67)^2} = \sqrt{1 + 0.4489} = \sqrt{1.4489} \approx 1.204$$

$$\text{SSE for C} = 1.204^2 \approx 1.45$$

- For point D(7, 8):

$$\text{distance}(D, C_2) = \sqrt{(7-7)^2 + (8-7.67)^2} = \sqrt{0^2 + (0.33)^2} = \sqrt{0.1089} \approx 0.33$$

$$\text{SSE for D} = 0.33^2 \approx 0.11$$

- For point F(8, 8):

$$\text{distance}(F, C_2) = \sqrt{(8-7)^2 + (8-7.67)^2} = \sqrt{(1)^2 + (0.33)^2} = \sqrt{1 + 0.1089} = \sqrt{1.1089} \approx 1.054$$

$$\text{SSE for F} = 1.054^2 \approx 1.11$$

**Total SSE for Cluster 2:**

$$SSE_2 = 1.45 + 0.11 + 1.11 = 2.67$$

**Total SSE (for both clusters):**

$$SSE_{\text{total}} = SSE_1 + SSE_2 = 10.68 + 2.67 = 13.35$$

### Step 3: Calculate Silhouette Coefficient

To calculate the **Silhouette Coefficient** for each point, we need to compute:

- $a(i)$: The average distance between point $i$ and all other points in the same cluster.
- $b(i)$: The average distance between point $i$ and all points in the nearest cluster.

Let's calculate the silhouette coefficient for point A (from Cluster 1) as an example.

**For point A (Cluster 1):**

- **Cohesion** $a(A)$: The average distance between A and all other points in Cluster 1 (B and E):

$$a(A) = \frac{\text{distance}(A, B) + \text{distance}(A, E)}{2}$$

  - $\text{distance}(A, B) = \sqrt{(2-3)^2 + (3-3)^2} = \sqrt{1} = 1$
  - $\text{distance}(A, E) = \sqrt{(2-5)^2 + (3-6)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$

$$a(A) = \frac{1 + 4.24}{2} = 2.62$$

- **Separation** $b(A)$: The average distance between A and all points in Cluster 2 (C, D, F):

$$b(A) = \frac{\text{distance}(A, C) + \text{distance}(A, D) + \text{distance}(A, F)}{3}$$

  - $\text{distance}(A, C) = \sqrt{(2-6)^2 + (3-7)^2} = \sqrt{16+16} = \sqrt{32} \approx 5.66$
  - $\text{distance}(A, D) = \sqrt{(2-7)^2 + (3-8)^2} = \sqrt{25+25} = \sqrt{50} \approx 7.07$
  - $\text{distance}(A, F) = \sqrt{(2-8)^2 + (3-8)^2} = \sqrt{36+25} = \sqrt{61} \approx 7.81$

$$b(A) = \frac{5.66 + 7.07 + 7.81}{3} = 6.51$$

Now, we can compute the **Silhouette Score** for point A:

$$S(A) = \frac{b(A) - a(A)}{\max(a(A), b(A))} = \frac{6.51 - 2.62}{\max(2.62, 6.51)} = \frac{3.89}{6.51} \approx 0.60$$

**Step 4: Dunn Index**

The **Dunn Index** is calculated as the ratio of the minimum distance between any two clusters (inter-cluster distance) to the maximum intra-cluster distance (diameter).

- **Inter-cluster distance:** The distance between the centroids of Cluster 1 and Cluster 2:

$$\text{distance}(C_1, C_2) = \sqrt{(3.33 - 7)^2 + (4 - 7.67)^2} = \sqrt{(-3.67)^2 + (-3.67)^2} = \sqrt{13.4689 + 13.4689} = \sqrt{26.9378} \approx 5.19$$

- **Intra-cluster distance:** The maximum distance within each cluster.
  - For Cluster 1, the maximum distance between any two points is between A and E:

$$\text{distance}(A, E) = 4.24$$

  - For Cluster 2, the maximum distance between any two points is between C and F:

$$\text{distance}(C, F) = 7.81$$

- **Dunn Index:**

$$\text{Dunn Index} = \frac{\text{Inter-cluster distance}}{\max(\text{Intra-cluster distance})} = \frac{5.19}{\max(4.24, 7.81)} = \frac{5.19}{7.81} \approx 0.66$$

**Conclusion:**

- **SSE** for the clustering: 13.35
- **Silhouette Score** for point A: 0.60
- **Dunn Index:** 0.66

These metrics provide insights into the clustering quality. Lower SSE indicates compact clusters, higher Silhouette Score indicates well-separated and cohesive clusters, and a higher Dunn Index suggests well-separated clusters with minimal overlap.

# Density-Based Clustering Algorithms

**Density**-Based **Clustering** refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density

## DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

The DBSCAN algorithm uses two parameters:

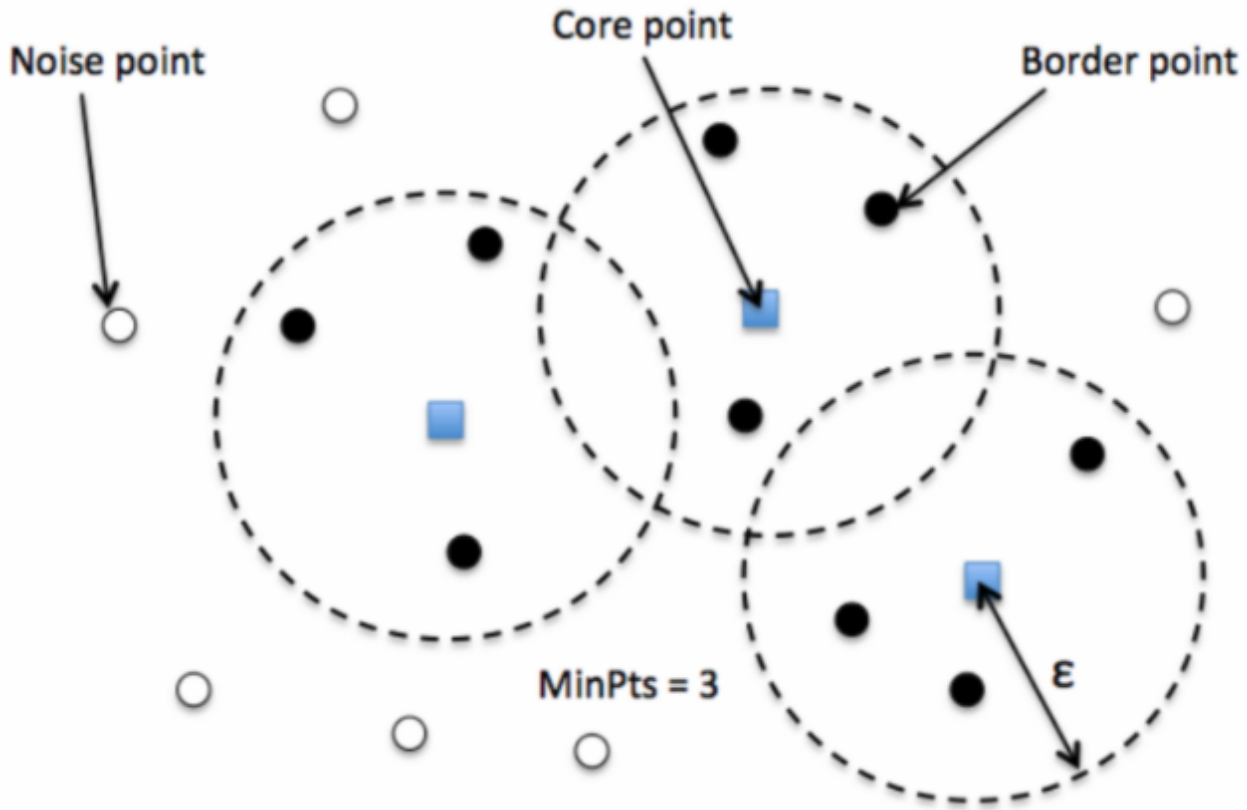
- **minPts:** The minimum number of points (a threshold) clustered together for a region to be considered dense.
- **eps (ε):** A distance measure that will be used to locate the points in the neighborhood of any point.

These parameters can be understood if we explore two concepts called Density Reachability and Density Connectivity.

**Reachability** in terms of density establishes a point to be reachable from another if it lies within a particular distance (eps) from it.

**Connectivity**, on the other hand, involves a transitivity based chaining-approach to determine whether points are located in a particular cluster. For example, p and q points could be connected if p->r->s->t->q, where a->b means b is in the neighborhood of a.

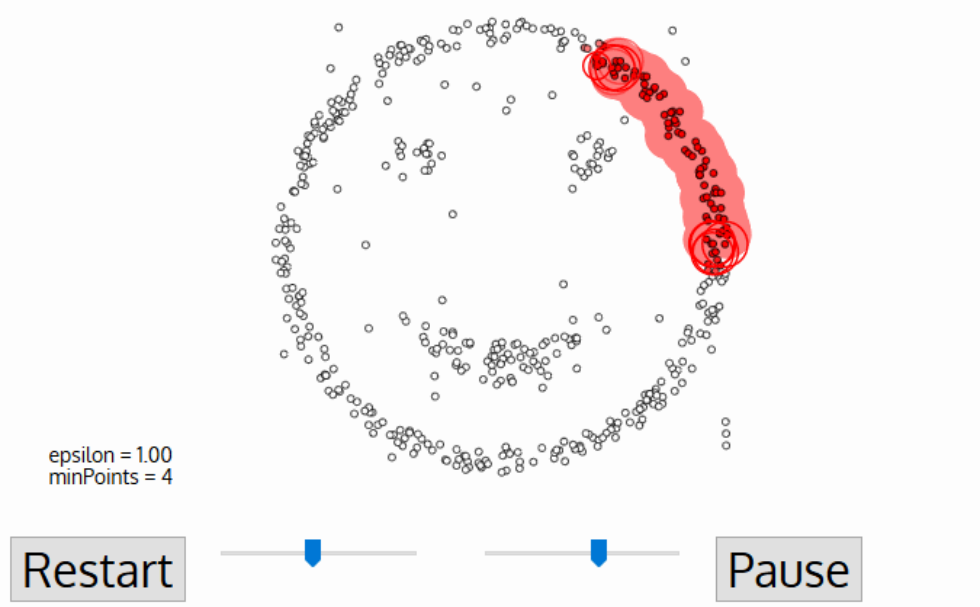
There are three types of points after the DBSCAN clustering is complete:




- **Core** — This is a point that has at least *m* points within distance *n* from itself.
- **Border** — This is a point that has at least one Core point at a distance *n*.
- **Noise** — This is a point that is neither a Core nor a Border. And it has less than *m* points within distance *n* from itself.

**Algorithmic steps for DBSCAN clustering**

1. The algorithm proceeds by arbitrarily picking up a point in the dataset (until all points have been visited).
2. If there are at least ‘minPoint’ points within a radius of ‘ε’ to the point then we consider all these points to be part of the same cluster.
3. The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point




It starts with a random unvisited starting data point. All points within a distance ‘Epsilon – Ɛ classify as neighborhood points.

1. You need a minimum number of points within the neighborhood to start the clustering process. Under such circumstances, the current data point becomes the first point in the cluster. Otherwise, the point gets labeled as ‘Noise.’ In either case, the current point becomes a visited point.
2. All points within the distance Ɛ become part of the same cluster. Repeat the procedure for all the new points added to the cluster group.

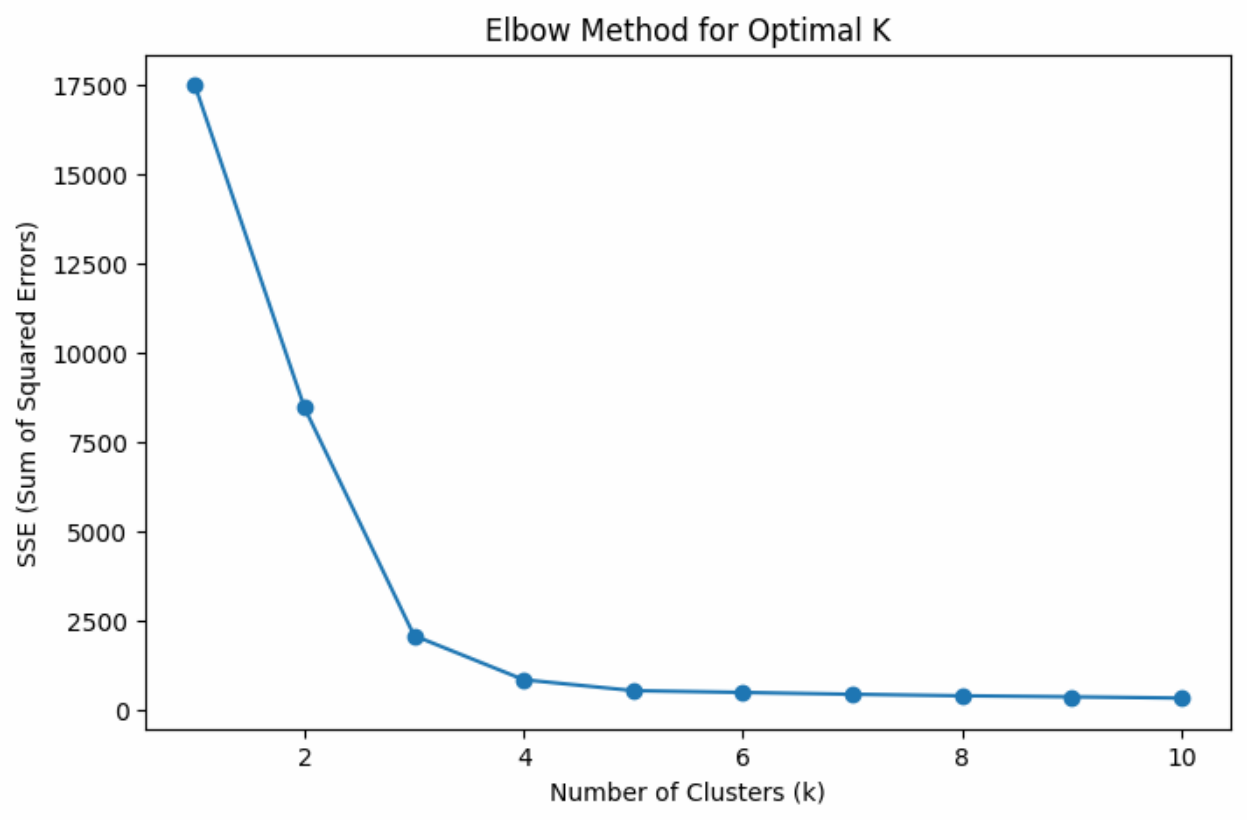
3. Continue with the process until you visit and label each point within the Ɛ neighborhood of the cluster.
4. On completion of the process, start again with a new unvisited point thereby leading to the discovery of more clusters or noise. At the end of the process, you ensure that you mark each point as either cluster or noise.

### Comparison: K-Means Clustering vs. DBSCAN Clustering

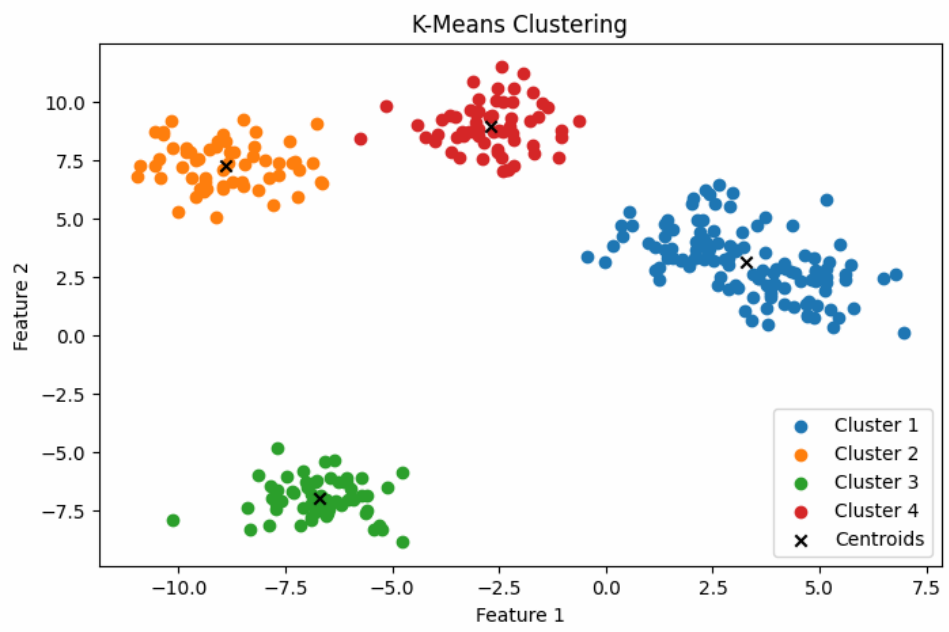
| S.No. | K-Means Clustering | DBSCAN Clustering |
|---|---|---|
| **1** | Clusters formed are more or less spherical or convex in shape and must have the same feature size. | Clusters formed are arbitrary in shape and may not have the same feature size. |
| **2** | Sensitive to the number of clusters specified. | Does not require the number of clusters to be specified. |
| **3** | More efficient for large datasets. | Struggles with efficiently handling high-dimensional datasets. |
| **4** | Does not work well with outliers and noisy datasets. | Efficiently handles outliers and noisy datasets. |
| **5** | In anomaly detection, assigns anomalous points to the same cluster as “normal” data points. | Locates regions of high density separated by regions of low density, effectively identifying anomalies. |
| **6** | Requires one parameter: **Number of clusters (K)**. | Requires two parameters: **Radius (R)** and **Minimum Points (M)**: |
| | | - **R**: Defines a radius to identify dense regions. |
| | | - **M**: Minimum number of points required in a neighborhood to form a cluster. |
| **7** | Handles data with varying densities effectively. | Performs poorly on sparse datasets or datasets with varying densities. |

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.spatial.distance import cdist
from sklearn.datasets import make_blobs

# Generate synthetic dataset
X, _ = make_blobs(n_samples=300, n_features=2, centers=5, random_state=42,
cluster_std=1.0)

# Elbow Method to determine optimal K
sse = []
range_k = range(1, 11)

for k in range_k:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)

# Plot the Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(range_k, sse, marker='o')
plt.title("Elbow Method for Optimal K")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("SSE (Sum of Squared Errors)")
plt.show()

# Perform K-Means with the optimal K
optimal_k = 4  # Replace this value based on the elbow method observation
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X)

# Cluster Validation Metrics
# 1. Sum of Squared Errors (SSE)
sse_final = kmeans.inertia_
print(f"Sum of Squared Errors (SSE): {sse_final}")

# 2. Silhouette Coefficient
```

```
silhouette_avg = silhouette_score(X, clusters)
print(f"Silhouette Coefficient: {silhouette_avg:.4f}")

# 3. Dunn Index
def dunn_index(X, labels, centroids):
    intra_cluster_distances = []
    for cluster_id in np.unique(labels):
        cluster_points = X[labels == cluster_id]
        intra_dist = np.max(cdist(cluster_points, cluster_points,
metric='euclidean'))
        intra_cluster_distances.append(intra_dist)

    inter_cluster_distances = cdist(centroids, centroids,
metric='euclidean')
    np.fill_diagonal(inter_cluster_distances, np.inf)
    min_inter_cluster_distance = np.min(inter_cluster_distances)

    dunn = min_inter_cluster_distance / max(intra_cluster_distances)
    return dunn

dunn_idx = dunn_index(X, clusters, kmeans.cluster_centers_)
print(f"Dunn Index: {dunn_idx:.4f}")

# Visualize the Clusters
plt.figure(figsize=(8, 5))
for i in range(optimal_k):
    plt.scatter(X[clusters == i, 0], X[clusters == i, 1], label=f"Cluster
{i+1}")
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
color='black', marker='x', label='Centroids')
plt.title("K-Means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```




Sum of Squared Errors (SSE): 861.1580192160816
Silhouette Coefficient: 0.7323
Dunn Index: 0.7775




DBSCAN Implementation:

1s

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN

from sklearn.datasets import make_circles

from sklearn.metrics import silhouette_score

from scipy.spatial.distance import cdist

# Generate spherical data

X, _ = make_circles(n_samples=500, factor=0.2, noise=0.1,
random_state=42)

# Perform DBSCAN clustering

eps = 0.15  # Maximum distance between two samples for them to be
considered neighbors

min_samples = 3  # Minimum number of points to form a dense region

dbscan = DBSCAN(eps=eps, min_samples=min_samples)

labels = dbscan.fit_predict(X)

# Number of clusters (excluding noise points labeled as -1)

n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

n_noise = list(labels).count(-1)

print(f"Number of clusters: {n_clusters}")

print(f"Number of noise points: {n_noise}")

# Visualize the clusters

plt.figure(figsize=(8, 5))

unique_labels = set(labels)

colors = [plt.cm.tab10(i / float(len(unique_labels) - 1)) for i in
range(len(unique_labels))]
```

```
for k, col in zip(unique_labels, colors):
    if k == -1:
        col = [0, 0, 0, 1]  # Noise points as black

    class_member_mask = (labels == k)
    xy = X[class_member_mask]
    plt.scatter(xy[:, 0], xy[:, 1], c=[col], label=f"Cluster {k}" if k != -1 else "Noise", edgecolor='k', s=50)

plt.title(f"DBSCAN Clustering on Spherical Data (eps={eps}, min_samples={min_samples})")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend(loc="best")
plt.show()
```

Number of clusters: 2
Number of noise points: 3