Question 1: What is the difference between K-Means and Hierarchical Clustering? Provide a use case for each.

# K-Means Clustering

## What it is

K-Means is a **partition-based clustering algorithm** where:

- You **choose the number of clusters (K)** in advance
- Data points are grouped based on **distance to the nearest centroid**
- The algorithm **iteratively updates centroids** until convergence

## Key Characteristics

- You must **decide K beforehand**
- Works best with **large datasets**
- Fast and **computationally efficient**
- Assumes clusters are **spherical and similar in size**
- Sensitive to **outliers**

# Hierarchical Clustering

## What it is

Hierarchical clustering builds a **tree-like structure (dendrogram)**:

- Does **not require K initially**
- Can be **Agglomerative** (bottom-up) or **Divisive** (top-down)
- Shows how clusters are **nested within each other**

## Key Characteristics

- No need to choose K in advance
- More **interpretable** (via dendrogram)

- Computationally **expensive**
- Works well with **small to medium datasets**
- Less sensitive to initialization

## Key Differences (Quick Comparison)

| Feature | K-Means | Hierarchical |
|---|---|---|
| Need to define K | Yes | No |
| Speed | Fast | Slow |
| Dataset size | Large | Small–Medium |
| Interpretability | Low | High |
| Output | Flat clusters | Tree structure |

Question 2: Explain the purpose of the Silhouette Score in evaluating clustering algorithms.

## What is the Silhouette Score?

The **Silhouette Score** is used to **evaluate the quality of clustering** by measuring:

- **How well a data point fits within its own cluster**
- **How different it is from other clusters**

## How does it work (Intuition)?

For each data point:

- **a** = average distance to all other points **in the same cluster**
  (how close it is to its own cluster)
- **b** = average distance to points **in the nearest neighboring cluster**
  (how far it is from other clusters)

### Formula:

Silhouette Score=b−amax(a,b)\text{Silhouette Score} = \frac{b - a}{\max(a, b)}Silhouette Score=max(a,b)b−a

---

## 🔹 Score Range & Interpretation

| Score Value | Meaning |
|---|---|
| **+1** | Excellent clustering |
| **~0** | Overlapping clusters |
| **−1** | Wrong cluster assignment |

👉 **Higher score = better clustering**

Question 3: What are the core parameters of DBSCAN, and how do they influence the clustering process?

# What is DBSCAN?

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** groups data points based on **density**, not distance to centroids.

👉 It can:

- Find **arbitrarily shaped clusters**
- Identify **noise/outliers**
- Work **without specifying K**

# `eps` (Epsilon)

## What it means

- The **maximum distance** between two points for them to be considered neighbors

## Influence on Clustering

- **Small eps** → Many points become noise, clusters may split
- **Large eps** → Clusters may merge, loss of structure

# `min_samples` (or MinPts)

## What it means

- The **minimum number of points** required within an `eps` neighborhood to form a **dense region**

## Influence on Clustering

- **Low min_samples** → More clusters, sensitive to noise
- **High min_samples** → Fewer clusters, stricter density requirement

Question 4: Why is feature scaling important when applying clustering algorithms like K-Means and DBSCAN?

# Why Feature Scaling is Important in Clustering

Feature scaling means **bringing all features to a similar range** (e.g., using Standardization or Min-Max scaling).

Clustering algorithms like **K-Means and DBSCAN rely heavily on distance calculations** (usually Euclidean distance).

# Impact on K-Means

## Why scaling matters

- K-Means assigns points to the **nearest centroid**
- Centroids are updated based on **mean values**

## Without scaling

Example:

- Age: 18–60
- Income: 20,000–2,00,000

Income will **dominate the distance calculation**, and age will be almost ignored.

# Impact on DBSCAN

## Why scaling matters

- DBSCAN uses **distance (`eps`)** to define neighborhoods

Question 5: What is the Elbow Method in K-Means clustering and how does it help determine the optimal number of clusters?

## What is the Elbow Method?

The **Elbow Method** is a technique used in **K-Means clustering** to determine the **optimal number of clusters (K)**.

It works by analyzing how the **within-cluster sum of squares (WCSS)** changes as K increases.

## How the Elbow Method Works

1. Run K-Means for different values of **K** (e.g., 1 to 10)
2. Calculate **WCSS** for each K
3. Plot **K vs WCSS**
4. Look for a point where the decrease in WCSS **slows down sharply**

## Why is it called the “Elbow”?

- Initially, increasing K significantly reduces WCSS
- After a certain point, adding more clusters gives **diminishing returns**
- The bend (elbow) represents a **balance between complexity and performance**

Question 6: Generate synthetic data using make_blobs(n_samples=300, centers=4), apply KMeans clustering, and visualize the results with cluster centers. (Include your Python code and output in the code box below.)

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Generate synthetic data using make_blobs
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

# Get cluster centers
centers = kmeans.cluster_centers_

# Visualize the clusters and cluster centers
plt.figure()
plt.scatter(X[:, 0], X[:, 1])
plt.scatter(centers[:, 0], centers[:, 1], marker='x')
plt.title("K-Means Clustering on make_blobs Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Question 7: Load the Wine dataset, apply StandardScaler , and then train a DBSCAN model. Print the number of clusters found (excluding noise). (Include your Python code and output in the code box below.)

```
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import numpy as np

# Load the Wine dataset
wine = load_wine()
X = wine.data

# Apply StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train DBSCAN model
dbscan = DBSCAN(eps=1.5, min_samples=5)
labels = dbscan.fit_predict(X_scaled)

# Number of clusters (excluding noise)
unique_labels = set(labels)
num_clusters = len(unique_labels) - (1 if -1 in unique_labels else
0)

print("Number of clusters found (excluding noise):", num_clusters)

Output

Number of clusters found (excluding noise): 3
```

Question 8: Generate moon-shaped synthetic data using make_moons(n_samples=200, noise=0.1), apply DBSCAN, and highlight the outliers in the plot. (Include your Python code and output in the code box below.)

```
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import numpy as np

# Generate moon-shaped synthetic data
X, _ = make_moons(n_samples=200, noise=0.1, random_state=42)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(X)

# Identify outliers (noise points)
outliers = labels == -1

# Plot clusters and highlight outliers
plt.figure()
plt.scatter(X[~outliers, 0], X[~outliers, 1], c=labels[~outliers])
plt.scatter(X[outliers, 0], X[outliers, 1], marker='x')
plt.title("DBSCAN on make_moons with Outliers Highlighted")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Question 9: Load the Wine dataset, reduce it to 2D using PCA, then apply Agglomerative Clustering and visualize the result in 2D with a scatter plot. (Include your Python code and output in the code box below.)

```
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt

# Load the Wine dataset
wine = load_wine()
X = wine.data

# Reduce data to 2D using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Apply Agglomerative Clustering
agg = AgglomerativeClustering(n_clusters=3)
labels = agg.fit_predict(X_pca)

# Visualize the clusters
plt.figure()
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels)
plt.title("Agglomerative Clustering on Wine Dataset (PCA Reduced)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

Question 10: You are working as a data analyst at an e-commerce company. The marketing team wants to segment customers based on their purchasing behavior to run targeted promotions. The dataset contains customer demographics and their product purchase history across categories. Describe your real-world data science workflow using clustering: ● Which clustering algorithm(s) would you use and why? ● How would you preprocess the data (missing

values, scaling)? ● How would you determine the number of clusters? ● How would the marketing team benefit from your clustering analysis? (Include your Python code and output in the code box below.)

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Load dataset (example)
# df = pd.read_csv("customer_data.csv")

# Example dataset
data = {
    "Age": [25, 45, 30, 50, 23, 40],
    "Gender": ["M", "F", "F", "M", "M", "F"],
    "Annual_Income": [40000, 80000, 50000, 90000, 35000, 70000],
    "Spending_Score": [60, 40, 50, 30, 70, 45]
}
df = pd.DataFrame(data)

# Preprocessing
# Encode categorical columns
df_encoded = pd.get_dummies(df, columns=["Gender"],
drop_first=True)

# Handle missing values (if any)
df_encoded.fillna(df_encoded.mean(), inplace=True)

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_encoded)

# Determine optimal number of clusters (Elbow Method)
wcss = []
for k in range(2, 6):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(2, 6), wcss, marker='o')
plt.title("Elbow Method")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.show()

# Train K-Means with chosen clusters (example: 3)
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_scaled)
```

```
# Add cluster labels to dataset
df["Cluster"] = labels
print(df)

# Evaluate clustering
score = silhouette_score(X_scaled, labels)
print("Silhouette Score:", score)
```