Hierarchical Clustering is a popular method in unsupervised learning used to group similar objects into clusters. The main purpose of hierarchical clustering is to build a hierarchy of clusters that can be visually represented using a tree-like diagram called a dendrogram. This method is particularly useful when the relationships between data points are hierarchical or nested.

Purpose of Hierarchical Clustering

- **Exploratory Data Analysis (EDA)**: Helps in understanding the natural structure and patterns within the data without any prior labels.
- **Data Summarization**: Provides a high-level overview of the data through the dendrogram, making it easier to identify significant clusters.
- **Anomaly Detection**: Identifies outliers or unusual data points that do not fit well into any cluster.
- **Creating Taxonomies**: Useful in fields like biology for creating taxonomies of species based on genetic similarities.

Types of Data Suitable for Hierarchical Clustering

Hierarchical clustering can be applied to various types of data, including:

- **Numerical Data**: Continuous variables where distance metrics like Euclidean distance can be applied.
- **Categorical Data**: Using methods like Gower distance to measure dissimilarity.
- **Mixed Data**: A combination of numerical and categorical data.

Examples of Suitable Data

- **Genomic Data**: Clustering genes or organisms based on genetic information.
- **Customer Segmentation**: Grouping customers based on purchasing behavior, demographics, etc.
- **Document Clustering**: Organizing a set of documents into clusters based on content similarity.

Real-Life Applications

- **Healthcare**: Clustering patients based on medical records to identify patterns in diseases and treatments.
- **Marketing**: Grouping products based on customer reviews and purchase patterns to optimize marketing strategies.
- **Finance**: Clustering stocks or investment portfolios to identify similar performance patterns.

Hierarchical clustering is a versatile tool that can be applied across various domains to uncover hidden patterns and relationships in data, aiding in better decision-making and insights.

To implement hierarchical clustering on a large dataset, we'll use a publicly available dataset that is suitable for clustering. For this example, we'll use the `Mall Customers` dataset, which contains information about customers of a mall. This dataset includes features like annual income and spending score, which are useful for clustering.

We'll perform the following steps:

1. Load the dataset.
2. Preprocess and visualize the data before clustering.
3. Apply hierarchical clustering.
4. Visualize the clusters after clustering.

Step 1: Load the Dataset

First, let's load the dataset and inspect its structure.

```
import pandas as pd


# Load the Mall Customers dataset
url = 'https://raw.githubusercontent.com/sharmaroshan/Mall-Customer-Segmentation-Dataset/master/Mall_Customers.csv'
data = pd.read_csv(url)


# Display the first few rows of the dataset
data.head()
```

Step 2: Preprocess and Visualize the Data Before Clustering

We'll focus on the `Annual Income (k$)` and `Spending Score (1-100)` for clustering. Let's visualize these features.

```
import matplotlib.pyplot as plt


# Extract relevant features
X = data[['Annual Income (k$)', 'Spending Score (1-100)']]


# Visualize the data before clustering
plt.figure(figsize=(10, 6))
```

```
plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'], c='blue', marker='o', s=50)

plt.title('Customer Data - Before Clustering')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.show()
```

Step 3: Apply Hierarchical Clustering

We'll perform hierarchical clustering on the extracted features and plot the dendrogram to decide the number of clusters.

```
from scipy.cluster.hierarchy import dendrogram, linkage


# Perform hierarchical clustering

linked = linkage(X, method='ward')


# Plot the dendrogram

plt.figure(figsize=(10, 7))

dendrogram(linked,

        orientation='top',

        distance_sort='descending',

        show_leaf_counts=True)

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('Sample index')

plt.ylabel('Distance')

plt.show()
```

Step 4: Visualize the Clusters After Clustering

Based on the dendrogram, we decide the number of clusters. Let's say we choose 5 clusters. We then use `fcluster` to get the cluster labels and visualize the clustered data.

```
from scipy.cluster.hierarchy import fcluster
```

```
# Decide the number of clusters (e.g., 5)

max_d = 150  # threshold to apply (determined from dendrogram)

clusters = fcluster(linked, max_d, criterion='distance')


# Add the cluster labels to the dataset

X['Cluster'] = clusters


# Visualize the clustered data

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

plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'], c=X['Cluster'], cmap='rainbow', marker='o', s=50)

plt.title('Customer Data - After Clustering')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.show()
```

Explanation

- `import pandas as pd`: This line imports the `pandas` library, which is used for data manipulation and analysis.
- `url = 'https://raw.githubusercontent.com/sharmaroshan/Mall-Customer-Segmentation-Dataset/master/Mall_Customers.csv'`: This line sets the URL for the dataset, which is stored in a CSV file.
- `data = pd.read_csv(url)`: This line reads the CSV file from the given URL into a pandas DataFrame called `data`.
- `data.head()`: This method displays the first five rows of the DataFrame to give a quick overview of the data.

- `import matplotlib.pyplot as plt`: This line imports the `matplotlib.pyplot` module, which is used for plotting graphs and visualizations.
- `X = data[['Annual Income (k$)', 'Spending Score (1-100)']]`: This line extracts the columns `Annual Income (k$)` and `Spending Score (1-100)` from the DataFrame `data` into a new DataFrame `X`. These columns are the features we will use for clustering.
- `plt.figure(figsize=(10, 6))`: This line creates a new figure for plotting with a size of 10 inches by 6 inches.
- `plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'], c='blue', marker='o', s=50)`: This line creates a scatter plot of the data with `Annual Income (k$)` on

the x-axis and `Spending Score (1-100)` on the y-axis. The points are colored blue (`c='blue'`), shaped as circles (`marker='o'`), and have a size of 50 (`s=50`).

- `plt.title('Customer Data - Before Clustering')`: This line sets the title of the plot.
- `plt.xlabel('Annual Income (k$)')`: This line sets the label for the x-axis.
- `plt.ylabel('Spending Score (1-100)')`: This line sets the label for the y-axis.
- `plt.show()`: This line displays the plot.

- `from scipy.cluster.hierarchy import dendrogram, linkage`: This line imports the `dendrogram` and `linkage` functions from the `scipy.cluster.hierarchy` module. The `linkage` function is used to perform hierarchical clustering, and the `dendrogram` function is used to plot the dendrogram.
- `linked = linkage(X, method='ward')`: This line performs hierarchical clustering on the DataFrame `X` using the Ward's method (`method='ward'`). The result, `linked`, is an array containing the hierarchical clustering encoded as a linkage matrix.
- `plt.figure(figsize=(10, 7))`: This line creates a new figure for the dendrogram plot with a size of 10 inches by 7 inches.
- `dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)`: This line plots the dendrogram for the hierarchical clustering result. The `orientation='top'` parameter ensures the dendrogram is oriented with the tree's root at the top. The `distance_sort='descending'` parameter sorts the distances in descending order, and `show_leaf_counts=True` shows the number of data points in each leaf.
- `plt.title('Hierarchical Clustering Dendrogram')`: This line sets the title of the dendrogram plot.
- `plt.xlabel('Sample index')`: This line sets the label for the x-axis of the dendrogram plot.
- `plt.ylabel('Distance')`: This line sets the label for the y-axis of the dendrogram plot.
- `plt.show()`: This line displays the dendrogram plot.

- `from scipy.cluster.hierarchy import fcluster`: This line imports the `fcluster` function from the `scipy.cluster.hierarchy` module. The `fcluster` function is used to form flat clusters from the hierarchical clustering defined by the linkage matrix.
- `max_d = 150`: This line sets the maximum distance threshold (`max_d`) for forming clusters. This value is determined from the dendrogram.
- `clusters = fcluster(linked, max_d, criterion='distance')`: This line forms flat clusters from the hierarchical clustering encoded in the `linked` array using the `max_d` threshold. The resulting cluster labels are stored in the array `clusters`.
- `X['Cluster'] = clusters`: This line adds a new column `Cluster` to the DataFrame `X`, containing the cluster labels for each data point.
- `plt.figure(figsize=(10, 6))`: This line creates a new figure for the clustered data plot with a size of 10 inches by 6 inches.
- `plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'], c=X['Cluster'], cmap='rainbow', marker='o', s=50)`: This line creates a scatter plot of the clustered data. The points are colored based on their cluster labels (`c=X['Cluster']`), using the `rainbow` colormap (`cmap='rainbow'`), shaped as circles (`marker='o'`), and have a size of 50 (`s=50`).

- `plt.title('Customer Data - After Clustering')`: This line sets the title of the clustered data plot.
- `plt.xlabel('Annual Income (k$)')`: This line sets the label for the x-axis of the clustered data plot.
- `plt.ylabel('Spending Score (1-100)')`: This line sets the label for the y-axis of the clustered data plot.
- `plt.show()`: This line displays the clustered data plot.

Graph Explanation

Interpretation:

- **X-axis**: Represents the annual income of the customers in thousands of dollars.
- **Y-axis**: Represents the spending score of the customers, which is a measure of how much they spend at the mall, scaled from 1 to 100.
- **Blue Dots**: Each dot represents a customer in the dataset. The position of the dot on the graph corresponds to their annual income and spending score.

This scatter plot provides a visual overview of the dataset, showing how the customers are distributed based on their income and spending score. Before clustering, all points are of the same color, indicating no prior grouping.

Hierarchical Clustering Dendrogram

```
# Plot the dendrogram

plt.figure(figsize=(10, 7))

dendrogram(linked,

        orientation='top',

        distance_sort='descending',

        show_leaf_counts=True)

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('Sample index')

plt.ylabel('Distance')

plt.show()
```

Interpretation:

- **X-axis**: Represents the sample indices or the original ordering of the data points.
- **Y-axis**: Represents the distance or dissimilarity between clusters.
- **Dendrogram Structure**: The tree-like structure shows how clusters are formed at different levels of similarity. Each merge represents the union of two clusters at a certain dissimilarity distance.
- **Horizontal Lines**: Each horizontal line represents a merge. The height of the line indicates the distance at which the clusters were merged.
- **Leaf Counts**: The number of original samples in each cluster (leaf) is shown at the bottom.

The dendrogram helps us decide the number of clusters by looking at where the vertical distance between merges is largest. This is often called the "elbow" method.

Visualization After Clustering

```
# Visualize the clustered data

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

plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'], c=X['Cluster'],
cmap='rainbow', marker='o', s=50)

plt.title('Customer Data - After Clustering')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.show()
```

Interpretation:

- **X-axis**: Represents the annual income of the customers in thousands of dollars.
- **Y-axis**: Represents the spending score of the customers.
- **Colored Dots**: Each dot now has a color representing its assigned cluster. Different colors indicate different clusters, helping to visually distinguish between groups of customers with similar characteristics.
- **Clusters**: The clusters shown on the plot represent groups of customers that have been grouped together based on their annual income and spending score.

This plot shows the result of hierarchical clustering. Each color represents a different cluster, illustrating how the algorithm has grouped customers based on the chosen features. By comparing this plot to the original scatter plot, we can see how the clustering algorithm has identified natural groupings within the data.

Summary of Insights

- **Before Clustering**: The plot shows no inherent grouping, and all data points are represented as a single color.
- **Dendrogram**: This plot helps determine the optimal number of clusters by visualizing the hierarchical merging of clusters at various distances.
- **After Clustering**: The scatter plot with colored dots shows the identified clusters, making it easy to see which customers are grouped together based on their annual income and spending score.

These visualizations provide valuable insights into the customer segments, allowing for targeted marketing strategies and better customer relationship management.

Answer of Ahmad Butt Question:

Selecting the optimal threshold (`max_d`) for hierarchical clustering can be done through various approaches, including visual inspection of the dendrogram, using statistical methods, or applying metrics like the silhouette score to find the most meaningful clusters. Here are detailed steps and approaches to decide the threshold dynamically:

1. Visual Inspection of the Dendrogram

When you plot a dendrogram, look for the largest vertical distance between horizontal lines that do not intersect any other lines below them. This is often referred to as the "elbow" method in hierarchical clustering. The threshold can be set just below this largest vertical gap to determine the number of clusters.

```
plt.figure(figsize=(10, 7))

dendrogram(linked,

        orientation='top',

        distance_sort='descending',

        show_leaf_counts=True)

plt.axhline(y=150, color='r', linestyle='--')  # Example static threshold

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('Sample index')

plt.ylabel('Distance')

plt.show()
```

2. Statistical Methods (Elbow Method)

Instead of a static threshold, you can look for an "elbow" in the plot of distances at which clusters were merged. The elbow point is where the within-cluster variance decreases sharply.

3. Silhouette Analysis

Silhouette analysis provides a quantitative method to decide the number of clusters by measuring how similar a point is to its own cluster compared to other clusters.

```
from sklearn.metrics import silhouette_score

from scipy.cluster.hierarchy import fcluster



# Calculate silhouette scores for different cluster counts

def calculate_silhouette_scores(linked, X, max_clusters=10):

    silhouette_scores = []

    for n_clusters in range(2, max_clusters + 1):

        cluster_labels = fcluster(linked, n_clusters, criterion='maxclust')

        silhouette_avg = silhouette_score(X, cluster_labels)

        silhouette_scores.append(silhouette_avg)

    return silhouette_scores



# Calculate silhouette scores

silhouette_scores = calculate_silhouette_scores(linked, X)



# Plot silhouette scores

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

plt.plot(range(2, 11), silhouette_scores, marker='o')
```

```
plt.title('Silhouette Scores for Various Numbers of Clusters')

plt.xlabel('Number of Clusters')

plt.ylabel('Silhouette Score')

plt.show()


# Find the optimal number of clusters

optimal_clusters = np.argmax(silhouette_scores) + 2  # +2 because range starts from 2

print(f'Optimal number of clusters: {optimal_clusters}')
```

4. Dynamic Threshold Selection using Inconsistency Method

The inconsistency method calculates the inconsistency of each cluster in the hierarchical tree and allows for dynamic thresholding based on statistical measure

```
from scipy.cluster.hierarchy import inconsistent, fcluster


# Calculate inconsistency statistics

incons = inconsistent(linked)


# Determine threshold using inconsistency statistics

depth = 5  # Typically between 3 and 5

incons_threshold = np.mean(incons[-depth:, 3]) + 1.25 * np.std(incons[-depth:, 3])


# Apply dynamic threshold to form flat clusters

clusters = fcluster(linked, incons_threshold, criterion='inconsistent')
```

5. Gap Statistics

Gap statistics compare the total within intra-cluster variation for different numbers of clusters with their expected values under null reference distribution of the data. However, this method is more complex to implement.

Explanation

- **Dendrogram**: Visualizes the hierarchical clustering and helps to inspect large vertical gaps for an initial guess of `max_d`.
- **Silhouette Scores Plot**: Helps to determine the optimal number of clusters by selecting the highest score.
- **Dynamic Thresholding**: Uses silhouette scores to determine the number of clusters and applies this number to form flat clusters.

By combining these methods, you can more reliably determine the optimal number of clusters and the appropriate threshold for hierarchical clustering in your dataset.

Answer Ansa Question

K-means clustering and hierarchical clustering are two popular unsupervised learning techniques used for grouping similar data points into clusters. They have different approaches, advantages, and disadvantages. Here's a detailed comparison:

K-means Clustering

Key Characteristics:

1. **Partitioning Method**: K-means is a partitioning method that divides the data into K distinct, non-overlapping subsets (clusters).
2. **Centroid-Based**: It uses centroids (mean of the data points in a cluster) to define and update clusters.
3. **Iterative Process**: It starts with an initial set of K centroids and iteratively updates them until convergence.

Algorithm Steps:

1. **Initialization**: Choose K initial centroids randomly or using a specific method (e.g., K-means++ initialization).
2. **Assignment**: Assign each data point to the nearest centroid.
3. **Update**: Recalculate the centroids as the mean of the assigned data points.
4. **Repeat**: Repeat the assignment and update steps until the centroids no longer change significantly.

Advantages:

1. **Scalability**: K-means is computationally efficient and can handle large datasets.
2. **Simplicity**: The algorithm is simple to understand and implement.

3. **Speed**: Converges quickly, especially with efficient initialization methods like K-means++.

Disadvantages:

1. **Predefined K**: Requires the number of clusters (K) to be specified beforehand.
2. **Sensitivity to Initialization**: The final clusters can vary depending on the initial centroids.
3. **Spherical Clusters**: Assumes clusters are spherical and equally sized, which may not always be the case.
4. **Sensitive to Outliers**: Outliers can significantly affect the centroids and the resulting clusters.

Hierarchical Clustering

Key Characteristics:

1. **Agglomerative or Divisive**: Can be agglomerative (bottom-up, starting with individual points and merging them) or divisive (top-down, starting with all points in one cluster and splitting them).
2. **Dendrogram**: Creates a tree-like structure (dendrogram) that shows the merging or splitting process at various levels.
3. **Distance-Based**: Uses a distance matrix to measure the similarity between data points or clusters.

Algorithm Steps (Agglomerative):

1. **Initialization**: Treat each data point as a single cluster.
2. **Merging**: Find the pair of clusters that are closest together and merge them.
3. **Repeat**: Repeat the merging process until all data points are in a single cluster.

Advantages:

1. **No Need for K**: Does not require the number of clusters to be specified in advance.
2. **Dendrogram**: Provides a visual representation (dendrogram) to help decide the number of clusters and understand the data's hierarchical structure.
3. **Flexibility**: Can capture complex cluster structures and is not restricted to spherical clusters.

Disadvantages:

1. **Scalability**: Computationally expensive for large datasets (especially the agglomerative approach).
2. **Static Clusters**: Once a merge or split is done, it cannot be undone (no re-evaluation of clusters).
3. **Distance Matrix Requirement**: Requires computation and storage of a distance matrix, which can be memory-intensive for large datasets.

Choosing between hierarchical clustering and K-means clustering depends on several factors, including the nature of your data, the size of your dataset, your specific clustering goals, and computational considerations. Here's a guide to help you decide when to use each method:

When to Use Hierarchical Clustering

1. **Small to Medium-Sized Datasets**:
   - Hierarchical clustering is computationally intensive, especially the agglomerative approach, which has a time complexity of O(n3)O(n^3)O(n3). It's more suitable for smaller datasets (typically fewer than a few thousand samples).
2. **No Predefined Number of Clusters**:
   - Use hierarchical clustering if you do not have a clear idea of the number of clusters. The dendrogram provides a visual tool to decide the optimal number of clusters.
3. **Understanding Data Structure**:
   - If you want to understand the nested structure of your data and the relationships between clusters at different levels, hierarchical clustering is beneficial. The dendrogram helps in visualizing how clusters merge and split at various distance thresholds.
4. **Complex Cluster Shapes**:
   - Hierarchical clustering can capture complex and non-spherical cluster shapes, making it more flexible than K-means, which assumes spherical clusters.
5. **Interpretability**:
   - If interpretability and a visual representation of the clustering process are important, hierarchical clustering is a good choice. The dendrogram provides a clear and interpretable view of the clustering hierarchy.

When to Use K-means Clustering

1. **Large Datasets**:
   - K-means is much more scalable and efficient for large datasets, with a time complexity of O(n·k·t)O(n \cdot k \cdot t)O(n·k·t), where nnn is the number of samples, kkk is the number of clusters, and ttt is the number of iterations.
2. **Predefined Number of Clusters**:
   - If you know the number of clusters (or can reasonably estimate it), K-means is suitable. It requires the number of clusters kkk to be specified beforehand.
3. **Speed and Efficiency**:
   - When you need a quick and computationally efficient clustering method, K-means is preferred. It converges quickly, especially with optimized initialization methods like K-means++.
4. **Spherical Clusters**:
   - Use K-means if the clusters are expected to be roughly spherical and of similar size. K-means works best when clusters are well-separated and have similar variances.
5. **Dimensionality**:

- K-means can handle higher-dimensional data more efficiently than hierarchical clustering.