Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant

Unsupervised Learning

Clustering, Dimensionality Reduction

AGENDA

1. Intro to Unsupervised Learning
2. Clustering Techniques
3. Dimensionality Reduction
4. Code Exercise

What's the best advice you've ever received, and who gave it to you?

1. Intro to Unsupervised Learning

A type of machine learning where the model learns patterns from unlabeled data, finding hidden structures or groupings without supervision.

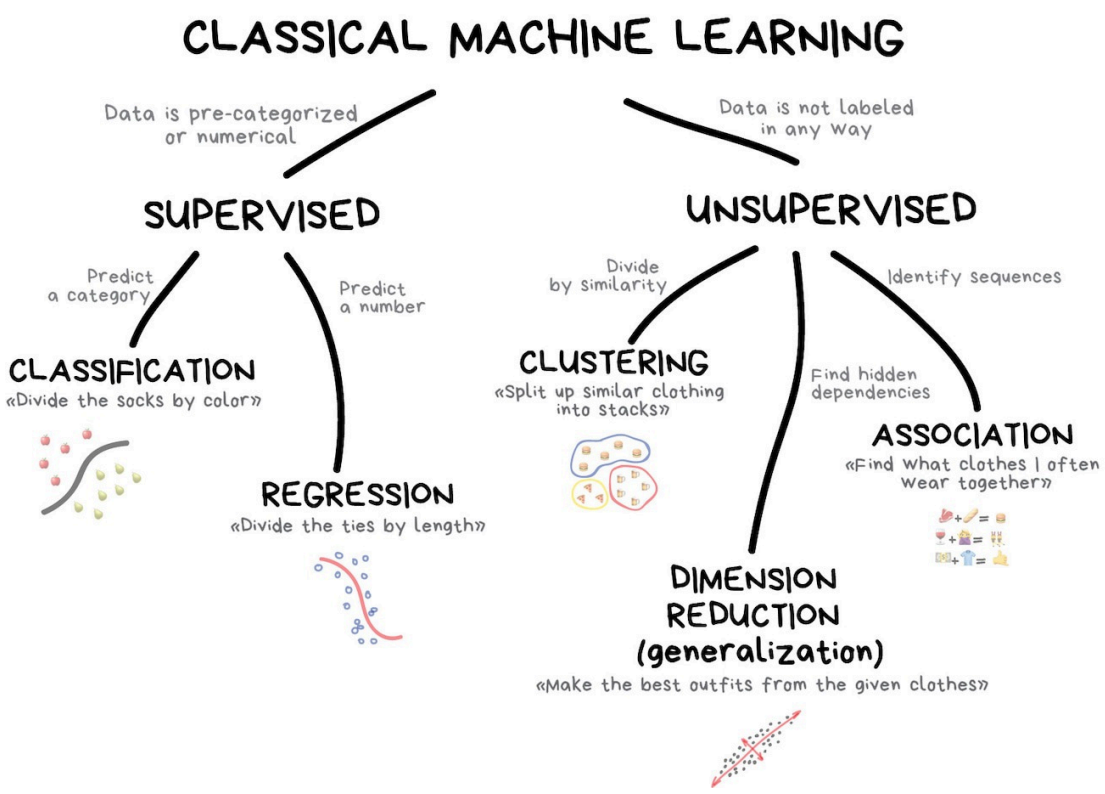
- **Key Differences from Supervised Learning**:
  - No labeled data (no target or outcome variable).
  - Focuses on finding patterns and structures within data.

Real-World Applications of Unsupervised Learning

- **Customer Segmentation**: Grouping customers based on purchasing behavior for targeted marketing.
- **Anomaly Detection**: Identifying unusual patterns or outliers in network security or credit card transactions.
- **Document Clustering**: Organizing documents into thematic groups for topic analysis.

Types of Unsupervised Learning

1. **Clustering**: Grouping similar data points together.
2. **Dimensionality Reduction**: Reducing the number of features while retaining essential information.



Clustering

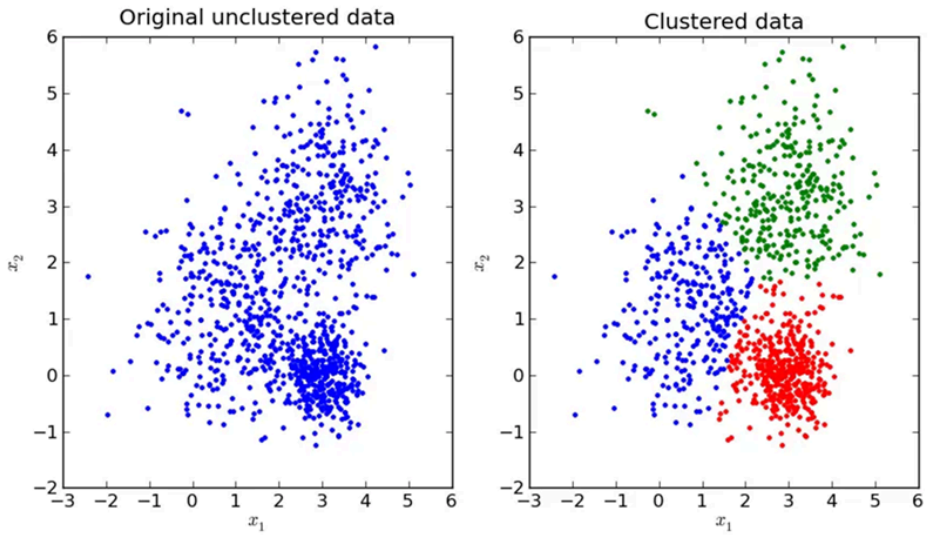
- **Definition**: Clustering is a technique to organize data into groups, or clusters, based on similarity, where data points within each cluster are more similar to each other than to those in other clusters.
- **Goals**:
  - **Identify Patterns**: Find natural groupings within data.
  - **Simplify Data**: Make complex datasets more manageable and interpretable.

Example Imagine you have a dataset of various animals with features like size, weight, and habitat. Using clustering, you might group these animals into clusters such as:

1. Mammals: Animals like elephants, lions, and whales that share similar traits.

2. Birds: Animals like eagles, parrots, and sparrows with common features.

3. Reptiles: Animals like snakes, lizards, and turtles that are grouped based on their shared characteristics. Each cluster represents a group of animals with similar attributes, helping you understand patterns and relationships in the data.



Hard Clustering vs Soft Clustering

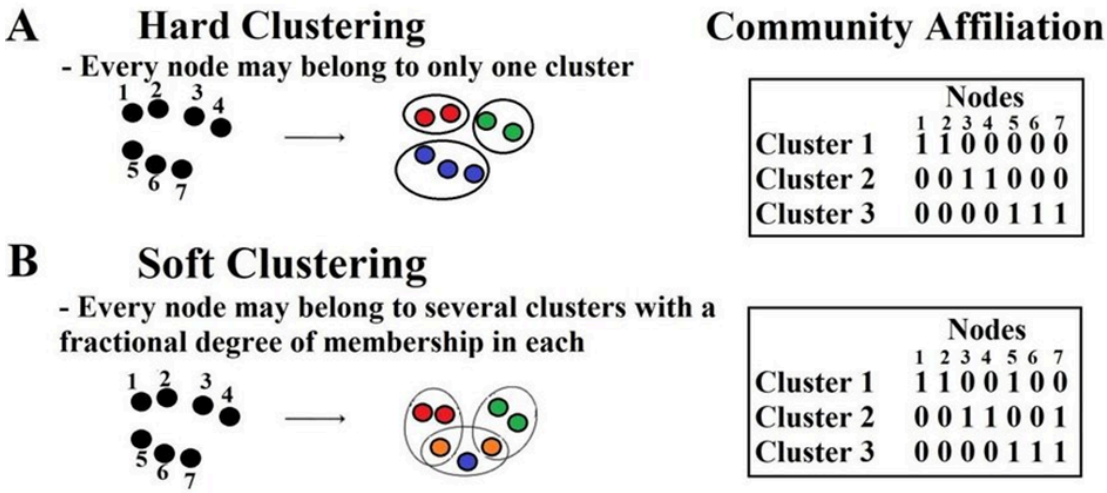
Hard Clustering assigns each data point to exactly one cluster. Once a data point is assigned to a cluster, it is considered part of that cluster with a definitive membership.

Characteristics:

● **Exclusive Membership**: Each data point belongs to a single cluster.

● **Clear Boundaries**: Clusters have clear boundaries, and data points are distinctly categorized.

● **Example Algorithm**: *K-Means Clustering* is a classic example of hard clustering.



Soft Clustering (also known as **Fuzzy Clustering**) allows data points to belong to multiple clusters with varying degrees of membership. Each data point has a probability or degree of membership for each cluster.

Characteristics:

● **Probabilistic Membership**: Each data point can belong to multiple clusters, with a certain degree of membership (e.g., 70% to cluster A, 30% to cluster B).

● **Overlapping Clusters:** Clusters can overlap, and there are no strict boundaries.

● **Example Algorithm**: Fuzzy C-Means Clustering is a common example of soft clustering.

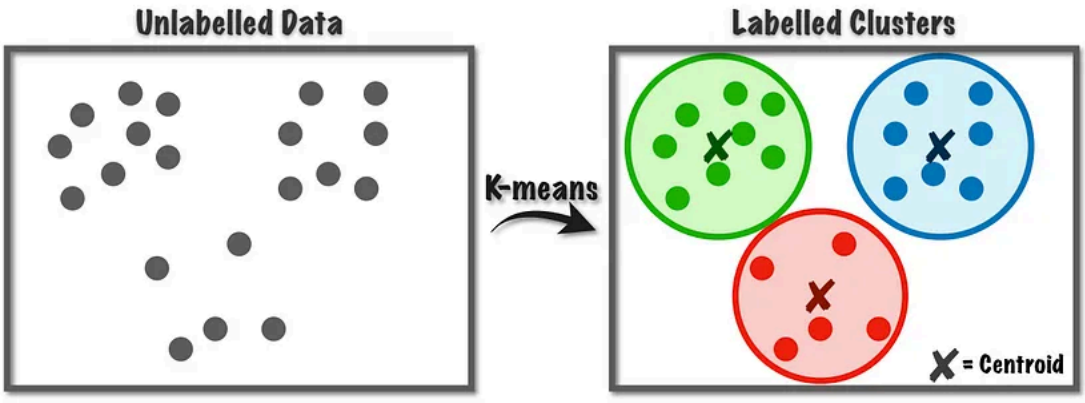
Popular Clustering Algorithms

1. **K-Means Clustering**: Partitions data into K clusters by minimizing the variance within each cluster.
2. **Hierarchical Clustering**: Builds a tree of clusters based on similarity, either merging smaller clusters or dividing a large cluster.
3. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**: Groups data points based on density, capable of handling noise and clusters of varying shapes.

K-Means Clustering

- Resource 1 - https://neptune.ai/blog/k-means-clustering
- Resource 2 - https://www.pinecone.io/learn/k-means-clustering

It is a popular algorithm used in unsupervised machine learning to partition data into k distinct clusters. Each cluster is defined by a centroid, which is the average of all points in the cluster.



1. **Initialization**: Randomly select K points as initial cluster centroids.
2. **Assignment**: Assign each data point to the nearest centroid.
3. **Update**: Recalculate the centroids by averaging the points in each cluster.
4. **Convergence**: Repeat assignment and update steps until centroids stabilize.

Centroid initialization methods

- **Random Data Points (Naive Initialization)**: K random data points are selected and defined as centroids.

```
kmeans = KMeans(n_clusters=3, init='random')
```

- **K-Means++ Initialization**: Centroids chosen in a way that spreads them out more evenly. The goal is to spread out the initial centroid by assigning the first centroid randomly then selecting the rest of the centroids based on the maximum squared distance. The idea is to push the centroids as far as possible from one another.

```
kmeans = KMeans(n_clusters=3, init='k-means++')
```

Choosing K (Number of Clusters)

- **Elbow Method**: The idea behind the elbow method is to plot the inertia (sum of squared distances from each point to its assigned centroid) for different values of K and look for an "elbow" in the plot. The point where the inertia decreases at a slower rate (forming an elbow) is typically considered the optimal K.

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

# Run KMeans for different K values and store inertia
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plot Elbow graph
plt.plot(range(1, 11), inertia)
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```

- **Silhouette Score**: The silhouette score measures how similar a point is to its own cluster compared to other clusters. It combines ideas of cohesion (how close points in a cluster are) and separation (how far apart the clusters are). A higher silhouette score indicates better-defined clusters.

```
from sklearn.metrics import silhouette_score

silhouette_scores = []
for k in range(2, 11):  # Silhouette score is undefined for K=1
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    score = silhouette_score(X, kmeans.labels_)
    silhouette_scores.append(score)

# Plot Silhouette Scores
plt.plot(range(2, 11), silhouette_scores)
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
```

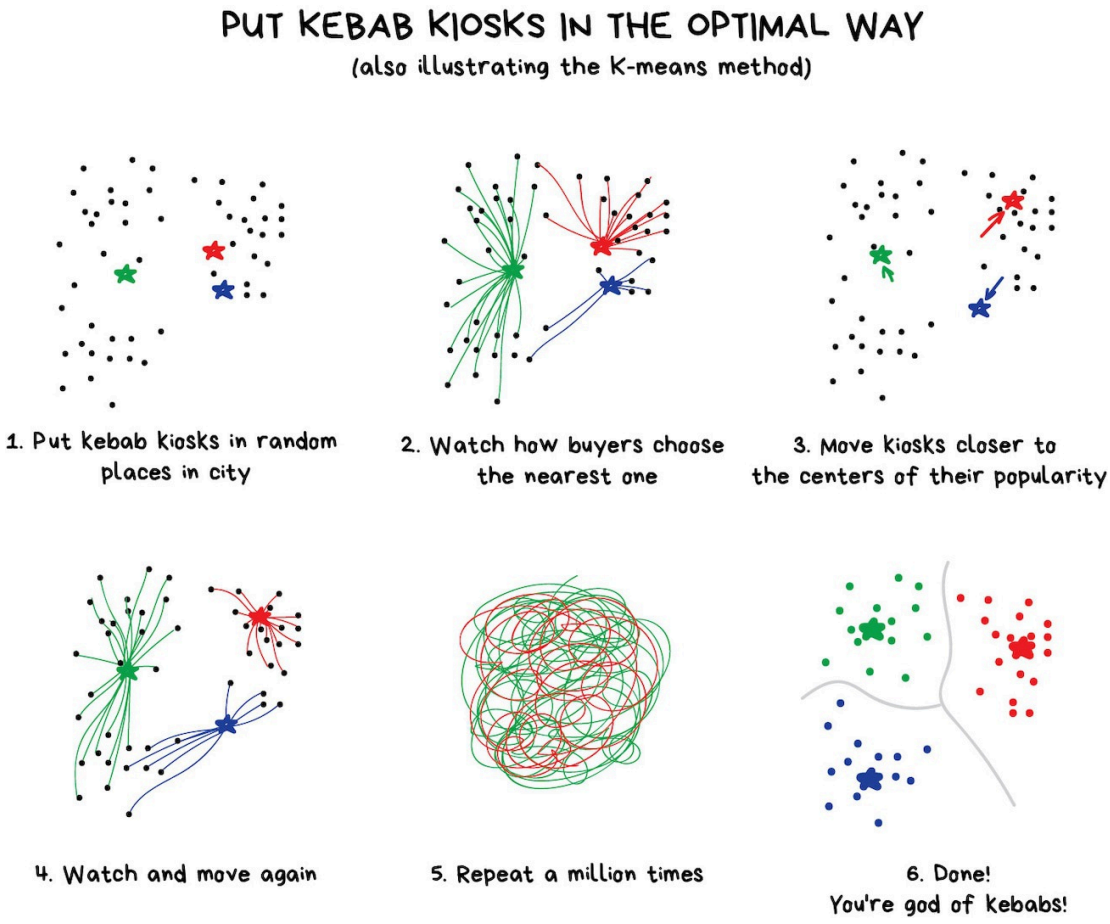
```
plt.title('Silhouette Method')
plt.show()
```

Practical Considerations

- **Initialization Methods**: Using methods like `k-means++` to choose better initial centroids and improve convergence.
- **Handling Outliers**: Outliers can skew centroids; consider data preprocessing to manage outliers.

Demo - https://www.philippe-fournier-viger.com/tools/kmeans_demo.php

The below illustration is from this blog - https://vas3k.com/blog/machine_learning/index.html



Demo & Small Activity (15-20 mins)

1. **Dataset**: Load a small dataset (e.g., Iris dataset or a synthetic dataset).
2. **Implementation**: Apply K-Means using Python's `scikit-learn` library.
3. **Visualization**:
   - Plot the clusters to visualize the groups.
   - Discuss the interpretation of each cluster and what it reveals about the data.

Example Code:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Load dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Apply K-Means
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
df['Cluster'] = kmeans.fit_predict(df)

# Visualize clusters
plt.scatter(df.iloc[:, 0], df.iloc[:, 1], c=df['Cluster'],
cmap='viridis')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
plt.title("K-Means Clustering on Iris Dataset")
plt.show()
```

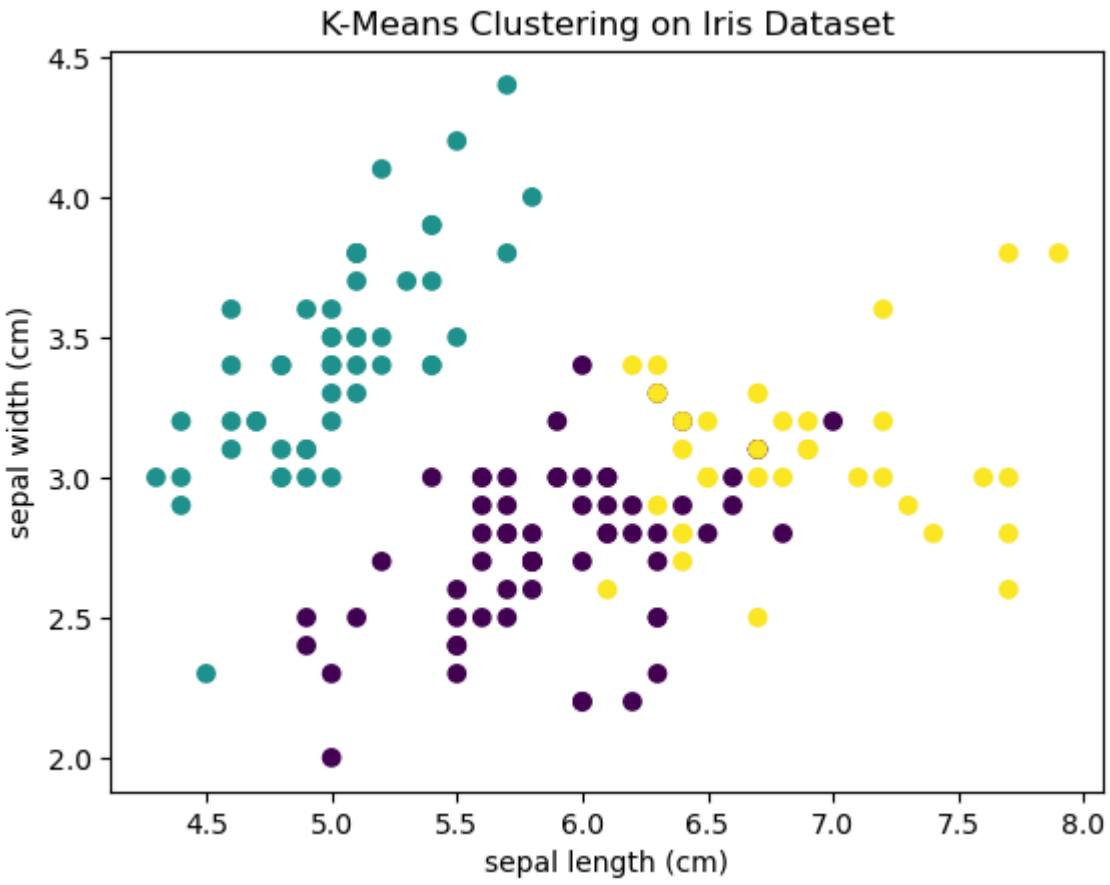
In [31]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Load dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Apply K-Means
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
df['Cluster'] = kmeans.fit_predict(df)

# Visualize clusters
plt.scatter(df.iloc[:, 0], df.iloc[:,1], c=df['Cluster'], cmap='viridis')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
plt.title("K-Means Clustering on Iris Dataset")
plt.show()
```

```
C:\Users\devid\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\devid\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [32]:
```
df.iloc[:, 0]
```

Out[32]:
```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: sepal length (cm), Length: 150, dtype: float64
```

2. Clustering Techniques and Evaluation

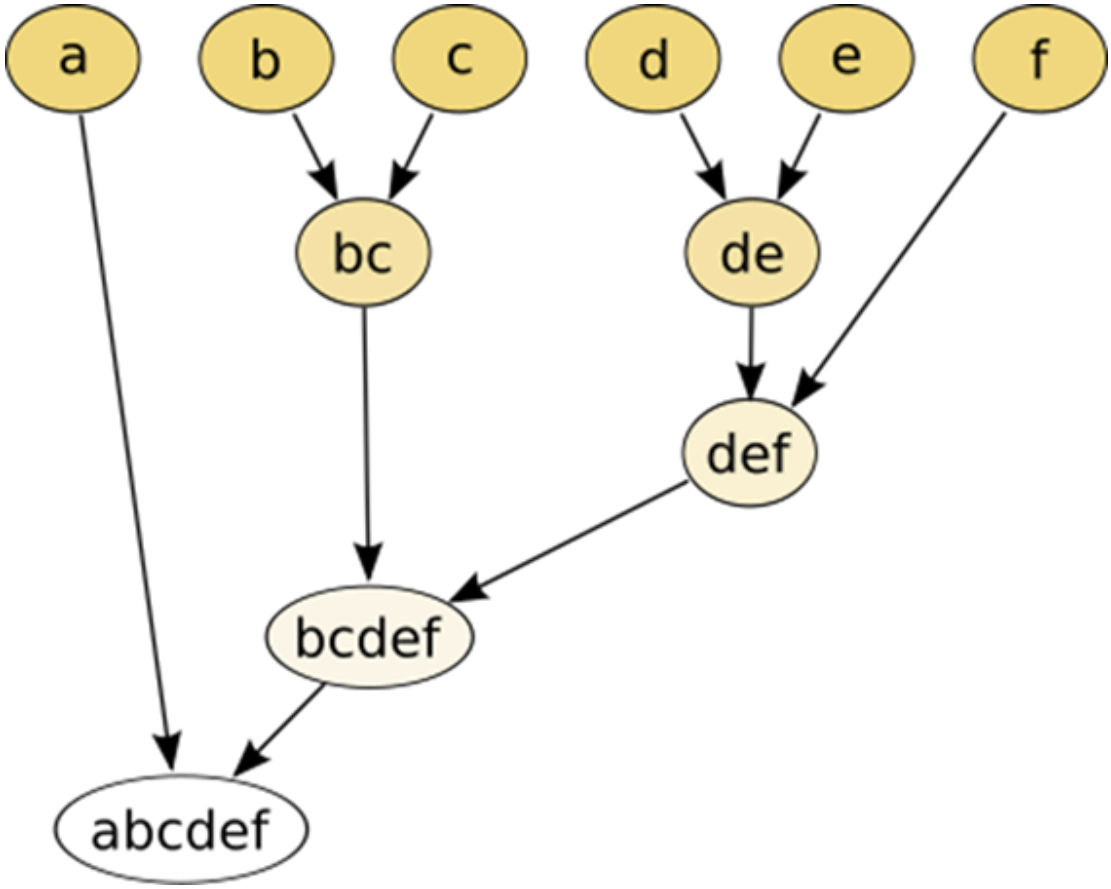
Hierarchical Clustering

https://d33wubrfki0l68.cloudfront.net/1e7845d2aa41a8e3654a37a877bc9f0fd391cd00/ceb7f/clustering-dendrogram.mp4

Hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters. Unlike partitioning methods like K-Means, which divide data into a fixed number of clusters, hierarchical clustering creates a tree-like structure (dendrogram) that shows the arrangement of clusters at various levels of granularity.

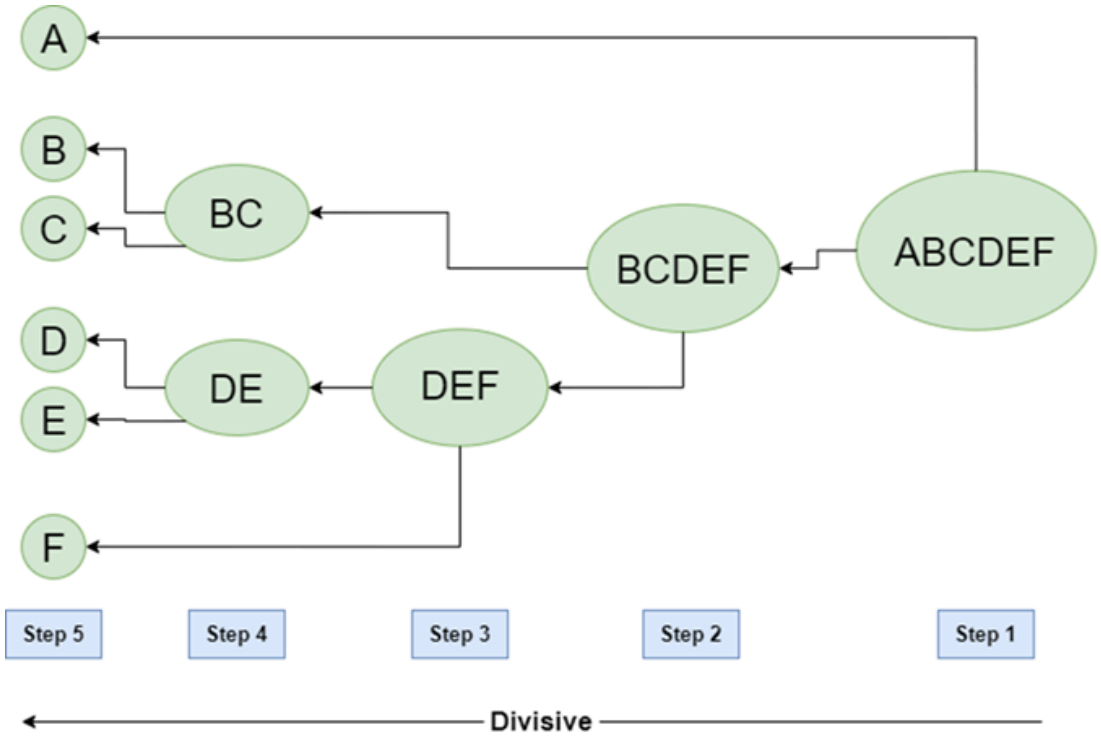
Key Features of Hierarchical Clustering

1. **Types of Hierarchical Clustering**:
   - **Agglomerative Clustering**: This is a bottom-up approach where each data point starts as its own cluster. The algorithm iteratively merges the closest pairs of clusters until only one cluster remains or a desired number of clusters is reached.
   - **Divisive Clustering**: This is a top-down approach, starting with one all-encompassing cluster and recursively splitting it into smaller clusters.



Agglomerative Hierarchical Clustering (Bottom-Up Approach):

- Start: Begin with each data point as its own cluster.
- Merge: Repeatedly merge the two closest clusters based on a distance metric until all points are in a single cluster or a stopping criterion is met.
- Result: A dendrogram is created showing how clusters are merged at each step



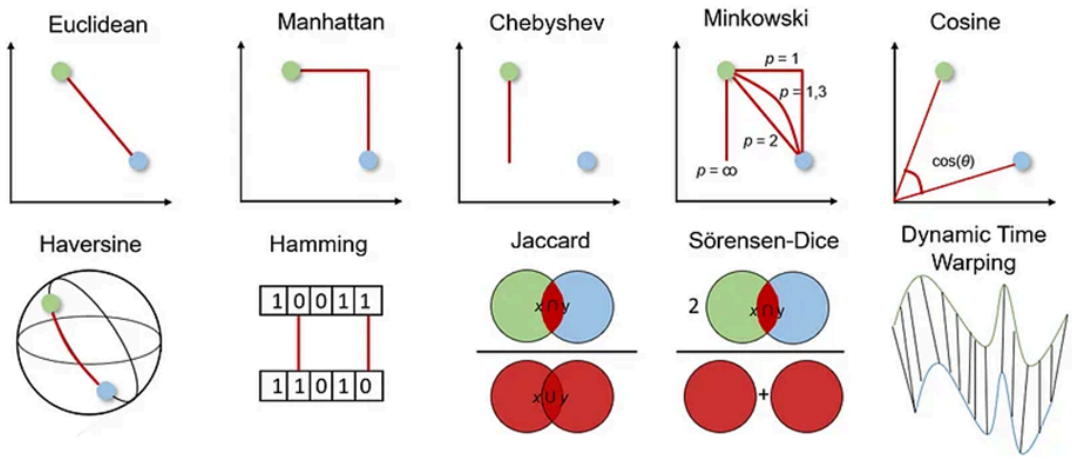
Divisive Hierarchical Clustering (Top-Down Approach):

- Start: Begin with all data points in a single cluster.
- Split: Repeatedly split the cluster into smaller clusters until each data point is in its own cluster or a stopping criterion is met.
- Result: A dendrogram is created showing how clusters are split at each step.

2. **Dendrogram**: A visual representation of the clustering process. The dendrogram shows the arrangement of clusters and allows you to choose a level at which to cut the tree to obtain a desired number of clusters.

3. **Linkage Criteria**: The method used to calculate the distance between clusters during the merging or splitting process. Common linkage criteria include:
   - **Single Linkage**: The distance between the closest points of two clusters.
   - **Complete Linkage**: The distance between the farthest points of two clusters.
   - **Average Linkage**: The average distance between all pairs of points in the two clusters.
   - **Ward's Method**: Aims to minimize the total within-cluster variance.

4. **Distance Metrics**: The choice of distance metric (e.g., Euclidean distance, Manhattan distance) can significantly affect the resulting clusters. The metric defines how the distance between points is calculated.



The **Euclidean distance** is the most widely used distance measure in clustering. It calculates the straight-line distance between two points in n-dimensional space. The formula for Euclidean distance is:

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

The **Manhattan distance** is also called the Taxicab or City-Block distance as the distance between two real-valued vectors is calculated as if one could only move at right angles. This distance measure is often used for discrete and binary attributes to get a realistic path.

$$d(p, q) = \Sigma_{i=1}^{n} |p_i - q_i|$$

5. **Applications**: Hierarchical clustering is widely used in various fields, including:
    - **Biology**: For clustering gene expression data or species classification.
    - **Market Research**: For customer segmentation based on purchasing behavior.
    - **Image Analysis**: For grouping similar images or features.

Advantages and Disadvantages

- **Advantages**:
    - No need to pre-specify the number of clusters.
    - The dendrogram provides a clear visual representation of data structure.

    - Can handle different shapes and sizes of clusters.
- **Disadvantages**:
    - Computationally expensive for large datasets (time complexity can be O(n^3)).
    - Sensitive to noise and outliers.
    - Once a decision is made to merge clusters, it cannot be undone.

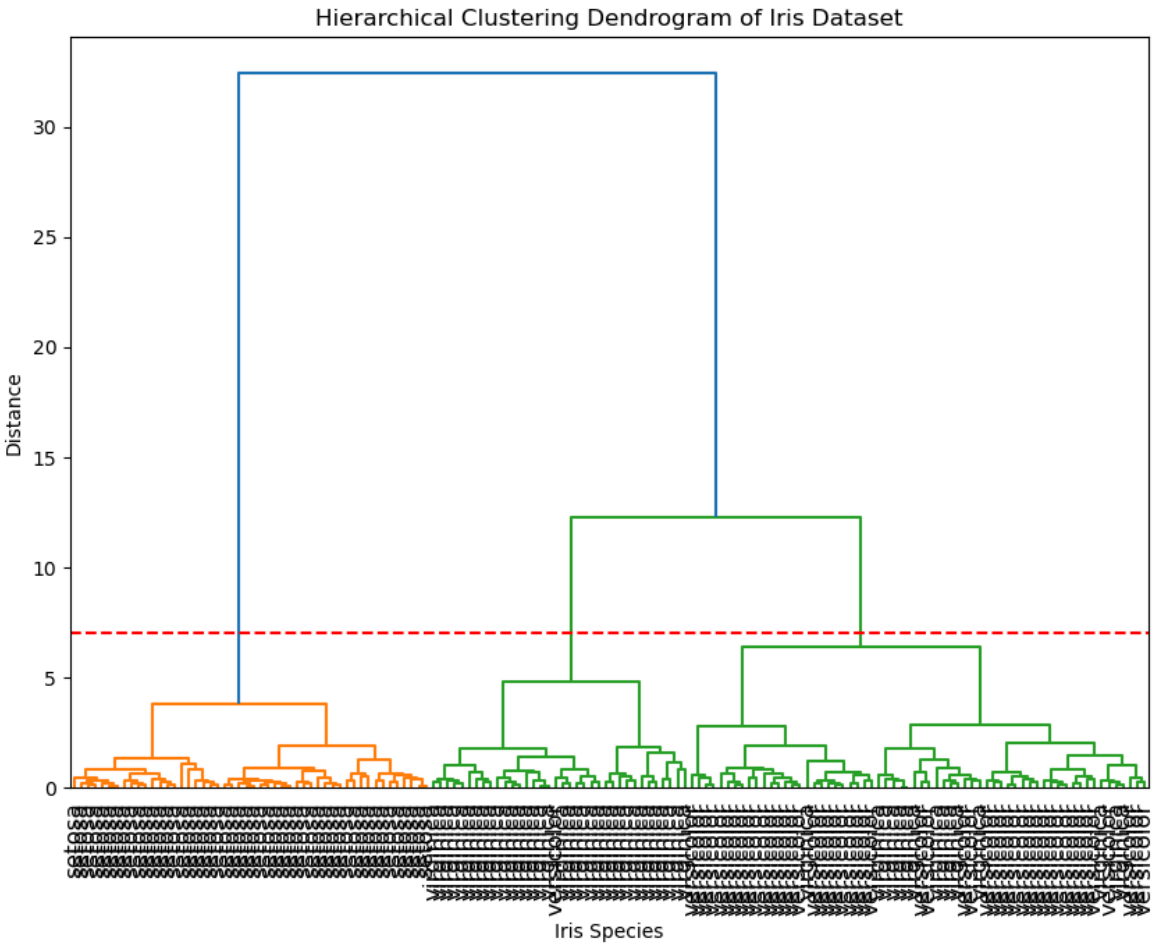
Exercise: Hierarchical Clustering with Dendrogram Visualization

In [52]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.datasets import load_iris
```

In [53]:
```
# Load the Iris dataset
iris = load_iris()
X = iris.data  # features
y = iris.target  # target labels
```

In [54]:
```
# Perform hierarchical clustering using Ward's method
Z = linkage(X, method='ward')
```

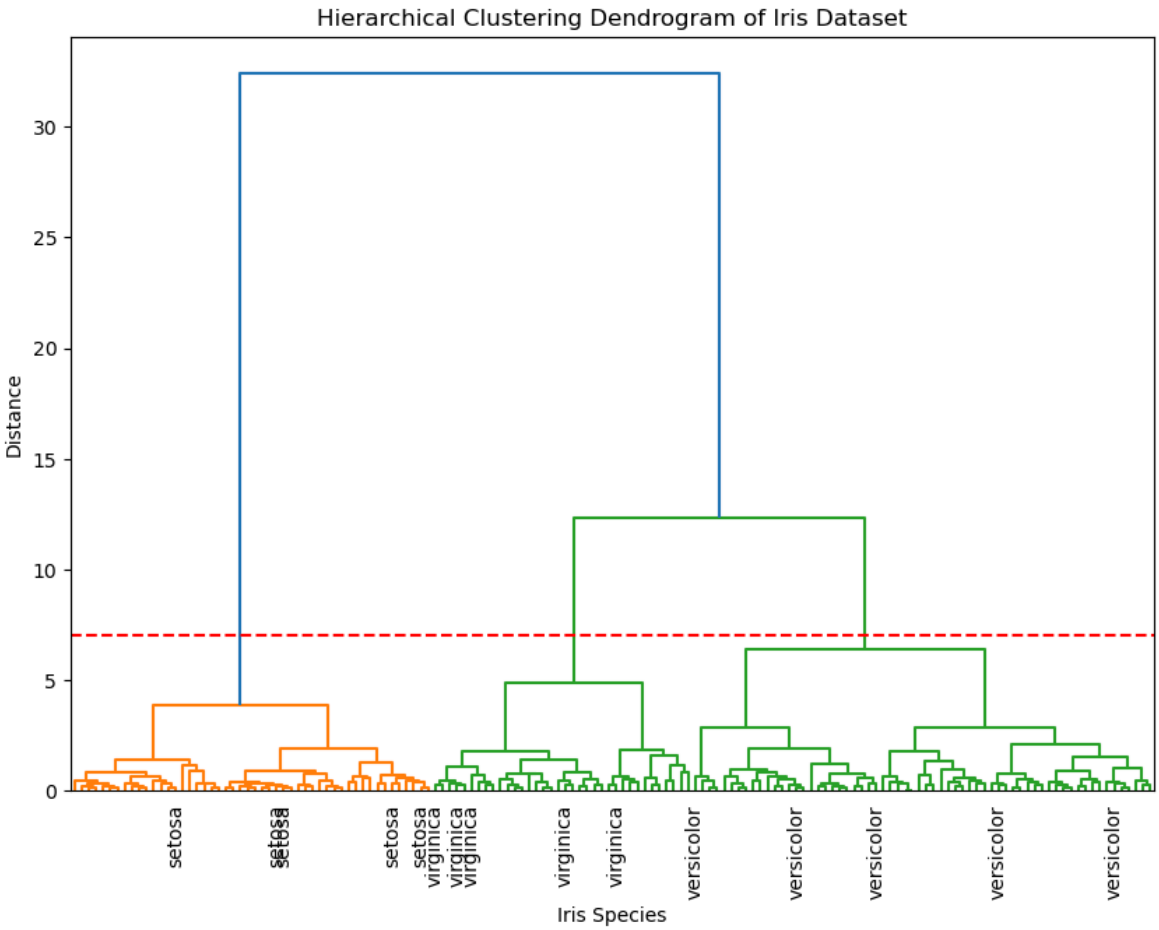
In [55]:
```
# Create a dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z, labels=iris.target_names[y], leaf_rotation=90, leaf_font_size=12)
plt.title('Hierarchical Clustering Dendrogram of Iris Dataset')
plt.xlabel('Iris Species')
plt.ylabel('Distance')
plt.axhline(y=7, color='r', linestyle='--')  # Optional line to indicate cut-off
plt.show()
```



In [56]:
```
# Generate labels for the dendrogram
# Show label for every 20th entry, otherwise leave it empty
reduced_labels = [
    iris.target_names[label] if index % 10 == 0 else ''
    for index, label in enumerate(y)
]
```

In [57]:
```
# Create a dendrogram
plt.figure(figsize=(10, 7))


# Create the dendrogram with the reduced labels
dendrogram(Z, labels=reduced_labels, leaf_rotation=90, leaf_font_size=10)
plt.title('Hierarchical Clustering Dendrogram of Iris Dataset')
plt.xlabel('Iris Species')
plt.ylabel('Distance')
plt.axhline(y=7, color='r', linestyle='--')  # Optional line to indicate cut-off
plt.show()
```



- https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html#sphx-glr-auto-examples-cluster-plot-agglomerative-dendrogram-py

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) (Advanced/Optional)

Cluster assignments are determined based on the density of data points in a region and assigned where there are high densities of data points separated by low-density regions.

How DBSCAN Works Learn More...

- **Core Points**: Points with at least `min_samples` neighbors within a specified radius `eps`.
- **Border Points**: Points within `eps` of a core point but with fewer than `min_samples` neighbors.
- **Noise Points**: Points that don't fit as core or border points and are thus considered outliers.

Parameters of DBSCAN

- **eps (Radius)**: Defines the neighborhood distance.

- **min_samples**: The minimum number of points needed to form a core point.

Strengths and Limitations

- **Strengths**:
  - Suitable for clusters of varying shapes and sizes.
  - Handles noise well, automatically identifying outliers.
- **Limitations**:
  - Struggles with clusters of varying density.
  - Sensitive to `eps` and `min_samples` values, which require tuning.

Evaluating Clustering Results (Advanced/Optional)

Metrics for Clustering Quality

1. **Silhouette Score**:
   - Measures the cohesion within clusters and separation between clusters.
   - Values range from -1 to 1; higher values indicate better-defined clusters.
2. **Davies-Bouldin Index**:
   - Measures the average similarity ratio between clusters, considering intra-cluster and inter-cluster distances.
   - Lower values indicate better clustering.
3. **Adjusted Rand Index (ARI)**:
   - Compares clustering performance with a known "ground truth," accounting for the possibility of random assignments.
   - Commonly used for evaluating clustering on synthetic or labeled datasets.

Practical Demo on Metrics (10-15 mins)

1. **Goal**: Apply these metrics to evaluate the clustering quality for both K-Means and DBSCAN.
2. **Implementation**:
   - Calculate and compare Silhouette Score, Davies-Bouldin Index, and (if a ground truth exists) Adjusted Rand Index.
   - Discuss how each metric reflects clustering quality and provides insights into algorithm performance.

Example Code:

```
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score, davies_bouldin_score,
adjusted_rand_score
from sklearn.datasets import make_blobs

# Generate synthetic dataset
```

```
data, labels_true = make_blobs(n_samples=300, centers=4,
random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=4)
kmeans_labels = kmeans.fit_predict(data)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(data)

# Evaluate K-Means
print("K-Means Silhouette Score:", silhouette_score(data,
kmeans_labels))
print("K-Means Davies-Bouldin Index:", davies_bouldin_score(data,
kmeans_labels))
print("K-Means Adjusted Rand Index (if ground truth is available):",
```

3. Dimensionality Reduction and Practical Application

Dimensionality reduction is a process in machine learning and data analysis that involves reducing the number of input variables or features in a dataset while preserving as much information as possible.

By reducing dimensions, we simplify the dataset, make it easier to visualize, and often improve model performance.

Challenges of High-Dimensional Data

- **Curse of Dimensionality**: As the number of features increases, the volume of the space increases, making data sparse and challenging to analyze. After a certain threshold more features would reduce the model performance.
- **Computational Cost**: High-dimensional data increases the time and resources needed for processing and modeling.
- **Overfitting**: Models can become overly complex and fit noise rather than the underlying data distribution.
- **Noise**: More features can introduce irrelevant data, making it harder to detect meaningful patterns.

To address the curse of dimensionality, Feature engineering techniques are used which include feature selection and feature extraction.

Benefits of Dimensionality Reduction

- **Simplifies Models**: Reducing the number of features can lead to simpler and more interpretable models.
- **Improves Interpretability**: Fewer dimensions allow for clearer insights and easier understanding of the data.

- **Visualizes High-Dimensional Data**: Enables the visualization of complex datasets in 2D or 3D spaces.

Examples of Dimensionality Reduction Techniques

- **Principal Component Analysis (PCA)**: Finds new "principal components" or directions in the data that maximize variance.
- **Linear Discriminant Analysis (LDA)**: Similar to PCA, but maximizes the separation between different classes rather than variance.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding)**: A non-linear technique primarily used for visualizing high-dimensional data in two or three dimensions.

Principal Component Analysis (PCA)

A method used to simplify complex datasets by reducing the number of dimensions (features) while still keeping most of the important information. It works by finding new "directions" in the data where there is the most variation (spread) and then projecting the data onto these directions.

PCA Scikit Learn

Learn PCA - GeeksForGeeks

- **Purpose**: To simplify datasets by projecting them onto the directions (principal components) where the variance is maximized.

```
# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)  # Project data onto the two principal components
```

How PCA Works

1. **Standardization**: Scale the data to have a mean of zero and a variance of one.
2. **Covariance Matrix**: Calculate the covariance matrix to understand how variables relate to each other.
3. **Eigenvalues and Eigenvectors**: Compute the eigenvalues and eigenvectors of the covariance matrix, which indicate the direction and magnitude of variance.
4. **Selection of Top Principal Components**: Choose the top components based on eigenvalues to form a new feature space.

Limitations of PCA

- **Captures Only Linear Relationships**: PCA may fail to capture complex, non-linear relationships in the data.
- **Loss of Interpretability**: The principal components may not always be easily interpretable in terms of original features.

t-SNE and UMAP (Advanced/Optional)

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Purpose**: Primarily used for visualizing high-dimensional data in 2D or 3D space.
- **Steps**:
  - Convert the high-dimensional data into probability distributions.
  - Minimize the divergence between the original and the embedded distribution.
- **Tuning Parameters**:
  - **Perplexity**: Balances attention between local and global aspects of the data.
- **Pros and Cons**:
  - **Pros**: Effective for non-linear data visualizations.
  - **Cons**: Computationally expensive and can be sensitive to parameter settings.

UMAP (Uniform Manifold Approximation and Projection)

- **Definition**: A non-linear dimensionality reduction technique that is often faster and preserves more of the global structure of data than t-SNE.
- **Tuning Parameters**:
  - **n_neighbors**: Affects local versus global structure.
  - **min_dist**: Controls the minimum distance between points in the low-dimensional space.
- **Pros**: Often provides more interpretable results and is scalable to larger datasets.

Resources:

- https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html#sphx-glr-auto-examples-cluster-plot-agglomerative-dendrogram-py
- https://www.philippe-fournier-viger.com/tools/kmeans_demo.php
- https://neptune.ai/blog/k-means-clustering
- https://vas3k.com/blog/machine_learning
- https://www.pinecone.io/learn/k-means-clustering/

Live Exercise

Now it's your turn!

Task 1: description of task

- instructions

Live Exercise Solutions

In [84]:
```
#solutions
```

Programming Interveiw Questions

What is the primary difference between supervised and unsupervised learning, particularly in terms of data labeling?

Answer: The primary difference is that in supervised learning, the model is trained on labeled data, meaning each training example is paired with an output label. In contrast, unsupervised learning works with unlabeled data, where the algorithm attempts to find hidden patterns or groupings in the data without prior knowledge of the outcomes.

Describe the K-Means clustering algorithm. What are the main steps involved in the K-Means algorithm?

Answer: K-Means clustering is an algorithm that partitions data into K distinct clusters based on similarity. The main steps involved are:

1. **Initialization**: Choose K initial centroids randomly from the data points.
2. **Assignment**: Assign each data point to the nearest centroid, forming K clusters.
3. **Update**: Calculate new centroids by taking the mean of all data points assigned to each cluster.
4. **Convergence**: Repeat the assignment and update steps until the centroids do not change significantly (convergence).

What is the purpose of the elbow method in K-Means clustering, and how does it help in selecting the optimal number of clusters?

Answer: The elbow method helps determine the optimal number of clusters (K) in K-Means clustering by plotting the within-cluster sum of squares (WCSS) against different values of K. As K increases, WCSS decreases. The "elbow" point in the plot, where the rate of decrease sharply changes, indicates the ideal number of clusters to choose, balancing model complexity and performance.

Explain the concept of hierarchical clustering. What are the two main approaches to hierarchical clustering, and how do they differ?

Answer: Hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters. The two main approaches are:

1. **Agglomerative Clustering (Bottom-Up)**: Starts with each data point as its own cluster and merges them iteratively based on proximity until a single cluster is formed.
2. **Divisive Clustering (Top-Down)**: Starts with all data points in one cluster and recursively splits them into smaller clusters.

The key difference is in their approach: agglomerative builds up clusters, while divisive breaks down clusters.

What is Principal Component Analysis (PCA), and how does it facilitate dimensionality reduction? What are the main steps involved in applying PCA to a dataset?

Answer: Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction while preserving as much variance as possible in the dataset. The main steps involved in applying PCA are:

1. **Standardization**: Normalize the dataset to have a mean of zero and a standard deviation of one.
2. **Covariance Matrix Computation**: Calculate the covariance matrix to identify how features vary with respect to one another.
3. **Eigenvalue and Eigenvector Calculation**: Compute the eigenvalues and eigenvectors of the covariance matrix to identify principal components.
4. **Feature Vector Formation**: Select the top k eigenvectors corresponding to the largest eigenvalues to form a new feature space.
5. **Recasting Data**: Transform the original dataset into the new feature space by projecting it onto the principal components.

PCA effectively reduces the number of dimensions while maintaining the essence of the data.

In [ ]:

Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant