Clustering

Theoretical Questions

1. **What is unsupervised learning in the context of machine learning ?**

Ans: Unsupervised learning is a machine learning approach where the model learns patterns from unlabeled data. It identifies hidden structures such as groups, relationships, or patterns in the data without predefined outputs (e.g., clustering and dimensionality reduction).

2. **How does K-Means clustering algorithm work ?**

Ans: K-Means clustering works by grouping data into K clusters based on similarity:

1. Choose the number of clusters K.
2. Randomly initialize K centroids.
3. Assign each data point to the nearest centroid (usually using Euclidean distance).
4. Recalculate centroids as the mean of assigned points.
5. Repeat steps 3–4 until centroids no longer change or convergence is reached.

3. Explain the concept of a dendrogram in hierarchical clustering?

Ans: A dendrogram is a tree-like diagram used in hierarchical clustering to show how data points are grouped step by step.
Each leaf represents a data point, and branches show clusters being merged (or split).
The height of the branches indicates the distance or dissimilarity between clusters—lower height means more similarity.
By cutting the dendrogram at a certain height, we can decide the number of clusters.

4. **What is the main difference between K-Means and Hierarchical Clustering ?**

Ans:
Key difference in a nutshell (executive summary):

- K-Means is a *partition-based* clustering approach where you must predefine K (number of clusters) and it iteratively optimizes cluster centroids.

- Hierarchical Clustering is a *tree-based* approach that builds a dendrogram, does not require K upfront, and lets you choose clusters by cutting the tree.

At a glance:

- K-Means → faster, scalable, needs K beforehand
- Hierarchical → more interpretable, flexible, higher computational cost

5.What are the advantages of DBSCAN over K-Means ?

Ans: Advantages of DBSCAN over K-Means:

- No need to predefine number of clusters
- Can find clusters of arbitrary (non-spherical) shape
- Handles noise and outliers explicitly
- Works better when clusters have different densities
- Not sensitive to initialization like K-Means

6. When would you use Silhouette Score in clustering ?

Ans: Silhouette Score is used in clustering to evaluate how well data points are grouped. It helps measure cluster quality by checking how similar a point is to its own cluster compared to other clusters. It is commonly used to choose the optimal number of clusters (k) and to compare different clustering algorithms.

7. What are the limitations of Hierarchical Clustering ?
Ans: Limitations of Hierarchical Clustering:

- High computational complexity; not suitable for very large datasets.
- Once clusters are merged or split, they cannot be undone.
- Sensitive to noise and outliers.
- Results depend heavily on the choice of distance metric and linkage method.
- Difficult to decide the optimal number of clusters from the dendrogram.

8. Why is feature scaling important in clustering algorithms like K-Means ?

Ans: Feature scaling is important in clustering algorithms like K-Means because K-Means uses distance (usually Euclidean distance) to form clusters. If features are on different scales, features with larger values dominate the distance calculation, leading to biased and incorrect clusters. Scaling ensures all features contribute equally, improving accuracy, convergence, and cluster quality.

9. How does DBSCAN identify noise points?

Ans: DBSCAN identifies noise points as data points that do not have enough neighboring points within a given radius (ε – epsilon) to meet the minimum points (MinPts) requirement.

If a point is neither a core point nor reachable from any core point, DBSCAN labels it as noise (outlier).

10. Define inertia in the context of K-Means.

Ans: Inertia in K-Means refers to the sum of squared distances between each data point and the centroid of its assigned cluster.

It measures how compact the clusters are—lower inertia means tighter, better-formed clusters.

11.What is the elbow method in K-Means clustering ?

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

It works by:

- Running K-Means for different values of K
- Calculating WCSS (Within-Cluster Sum of Squares) for each K
- Plotting K vs WCSS

As K increases, WCSS decreases. The point where the decrease starts slowing down sharply (forming an “elbow”) is considered the optimal K, as adding more clusters beyond this gives diminishing returns.

12. Describe the concept of "density" in DBSCAN.

Ans: In DBSCAN, density refers to how closely data points are packed together in a region. A point is considered dense if it has at least a minimum number of points (MinPts) within a specified radius (ε / eps). Clusters are formed by connecting such dense regions, while points in low-density areas are treated as noise**.**

13. Can hierarchical clustering be used on categorical data ?

Ans: At a high level, hierarchical clustering is not directly suited for purely categorical data because it relies on distance measures (like Euclidean) meant for numerical values.

That said, it can be used indirectly by:

- Applying appropriate similarity measures (e.g., Hamming distance),
- Or transforming categorical data using encoding methods,
- Or using variants designed for mixed data (e.g., Gower distance).

Net-net: possible with adaptations, not out-of-the-box.

14. What does a negative Silhouette Score indicate?

Ans: A negative Silhouette Score indicates that data points are likely assigned to the wrong cluster—they are closer to points in other clusters than to their own. In short, it signals poor clustering quality and overlapping clusters.

15. Explain the term "linkage criteria" in hierarchical clustering?

Ans: In hierarchical clustering, linkage criteria refers to the rule used to measure the distance between two clusters when deciding which clusters should be merged (agglomerative) or split (divisive).

In simple terms, it defines how the distance between clusters is calculated.

Common linkage criteria include:

- Single linkage – distance between the closest points of clusters
- Complete linkage – distance between the farthest points
- Average linkage – average distance between all points
- Ward’s linkage – minimizes variance within clusters

It directly affects the shape and structure of the resulting clusters.

16. Why might K-Means clustering perform poorly on data with varying cluster sizes or densities?

Ans: K-Means can perform poorly with varying cluster sizes or densities because it assumes clusters are spherical, similar in size, and evenly dense. It uses distance to centroids, so larger or denser clusters dominate, smaller or sparse clusters get misclassified, and boundaries between clusters become inaccurate.

17. What are the core parameters in DBSCAN, and how do they influence clustering ?

Ans: In DBSCAN, clustering is driven by three core parameters that define density:

- ε (Epsilon): The maximum distance between two points to be considered neighbors. A larger ε creates bigger, fewer clusters; a smaller ε results in more, tighter clusters and more noise.

- MinPts: The minimum number of points required within ε to form a dense region. Higher MinPts makes clusters more robust and reduces noise; lower MinPts forms smaller, more sensitive clusters.

- Distance metric: Determines how distance is calculated (e.g., Euclidean, Manhattan). It directly affects neighborhood detection and cluster shape.

18. How does K-Means++ improve upon standard K-Means initialization ?

Ans: K-Means++ improves standard K-Means by choosing initial centroids more strategically rather than randomly. It spreads the starting centers far apart based on data distance, which reduces poor clustering, speeds up convergence, and improves final cluster quality.

19. What is agglomerative clustering?

Ans: Agglomerative clustering is a hierarchical clustering technique where each data point starts as its own cluster, and the closest clusters are merged step by step until a single cluster or a desired number of clusters is formed.

20. What makes Silhouette Score a better metric than just inertia for model evaluation?

Ans: From a metrics-alignment standpoint, Silhouette Score outperforms inertia because it delivers a normalized, interpretable measure of cluster quality by balancing intra-cluster cohesion and inter-cluster separation.

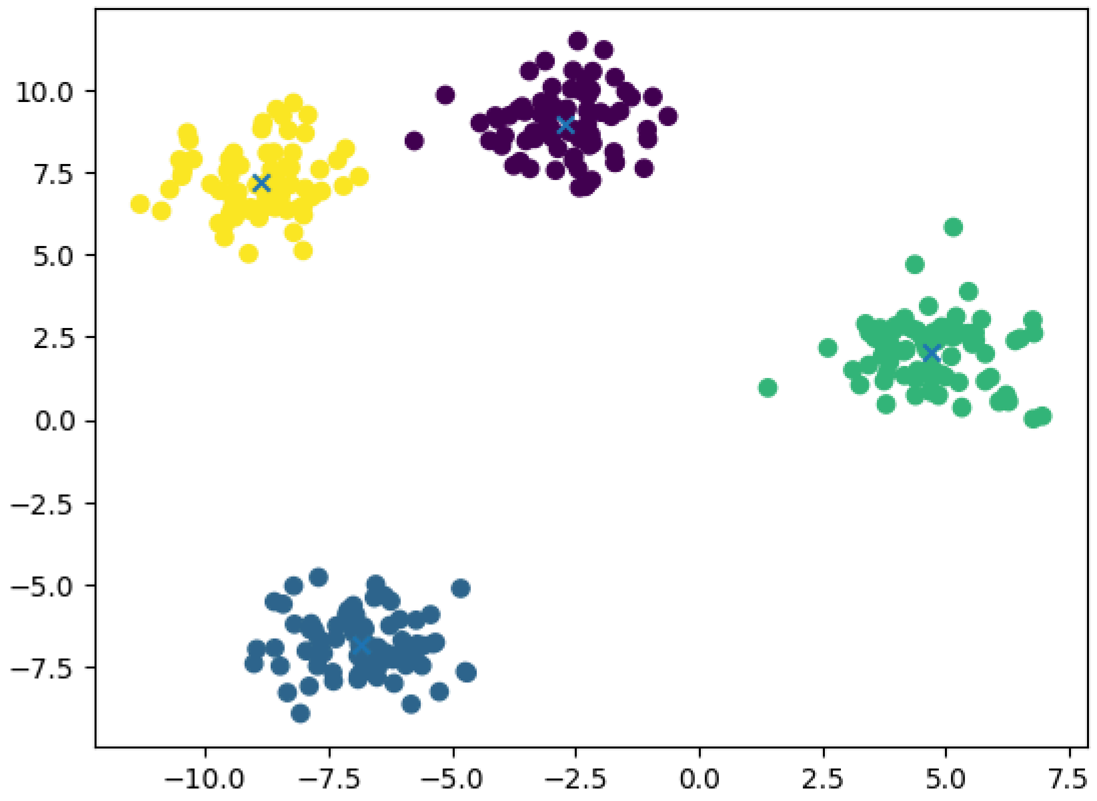
In contrast, inertia only measures compactness, always decreases as clusters increase, and lacks a natural benchmark—making it less decision-grade for model evaluation. Silhouette provides clearer signal for optimal *k* and comparative analysis.

Practical Questions

21.Generate synthetic data with 4 centers using make_blobs and apply K-Means clustering. Visualize using a scatter plot.
Ans:

```
from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt


# Generate data

X, _ = make_blobs(n_samples=300, centers=4, random_state=42)


# Apply K-Means

kmeans = KMeans(n_clusters=4, random_state=42)

labels = kmeans.fit_predict(X)


# Visualization

plt.scatter(X[:, 0], X[:, 1], c=labels)

plt.scatter(kmeans.cluster_centers_[:, 0],

            kmeans.cluster_centers_[:, 1], marker='x')

plt.show()
```

OutPut:



22. Load the Iris dataset and use Agglomerative Clustering to group the data into 3 clusters. Display the first 10 predicted labels.

Ans:

```
from sklearn.datasets import load_iris

from sklearn.cluster import AgglomerativeClustering



# Load Iris dataset

iris = load_iris()

X = iris.data



# Apply Agglomerative Clustering
```

```
model = AgglomerativeClustering(n_clusters=3)

labels = model.fit_predict(X)


# Display first 10 predicted labels

print(labels[:10])
```

Output: `[1 1 1 1 1 1 1 1 1 1]`

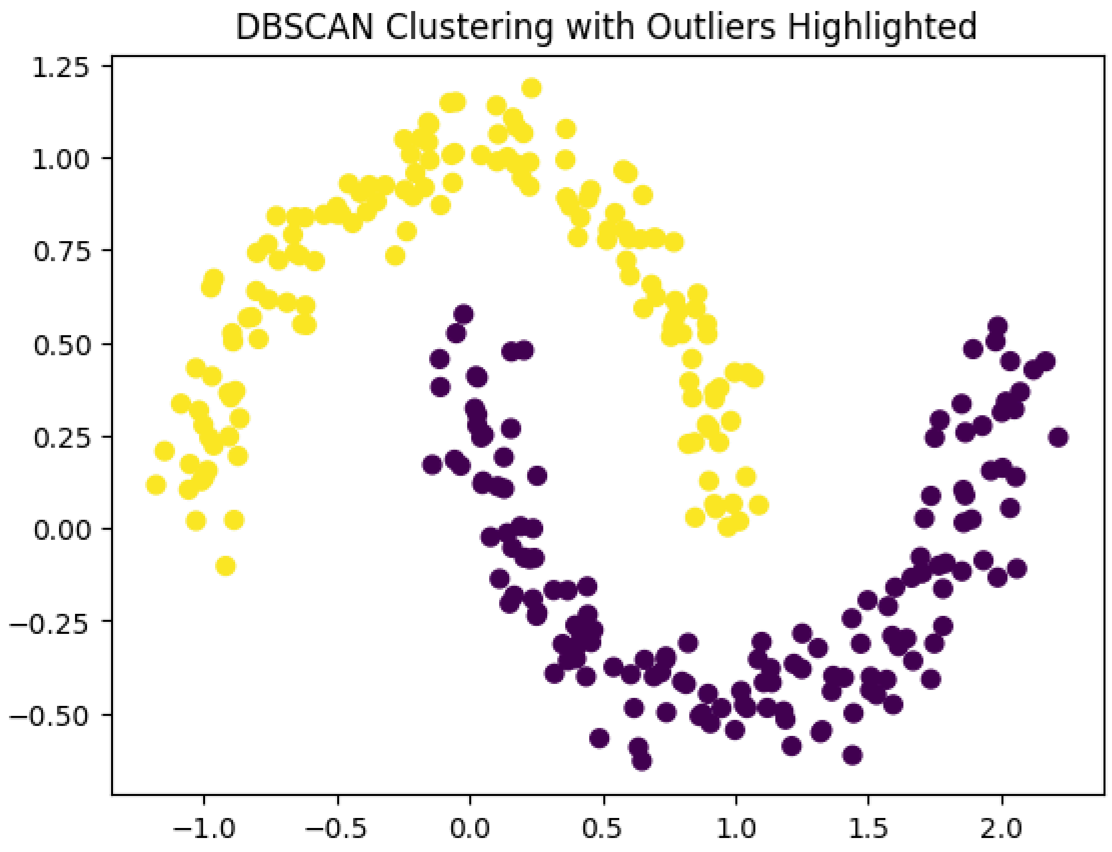
23.Generate synthetic data using make_moons and apply DBSCAN. Highlight outliers in the plot.

Ans:

```
from sklearn.datasets import make_moons

from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt

import numpy as np


# Generate synthetic data

X, _ = make_moons(n_samples=300, noise=0.08, random_state=42)


# Apply DBSCAN

dbscan = DBSCAN(eps=0.25, min_samples=5)

labels = dbscan.fit_predict(X)
```

```
# Identify outliers

outliers = labels == -1



# Plot

plt.scatter(X[~outliers, 0], X[~outliers, 1], c=labels[~outliers], s=40)

plt.scatter(X[outliers, 0], X[outliers, 1], c='red', s=60, marker='x')

plt.title("DBSCAN Clustering with Outliers Highlighted")

plt.show()
```

OutPut:



24. Load the Wine dataset and apply K-Means clustering after standardizing the features. Print the size of each cluster.
Ans:

```
from sklearn.datasets import load_wine

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

import numpy as np


# Load Wine dataset

wine = load_wine()

X = wine.data


# Standardize features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Apply K-Means clustering

kmeans = KMeans(n_clusters=3, random_state=42)

labels = kmeans.fit_predict(X_scaled)


# Print size of each cluster

unique, counts = np.unique(labels, return_counts=True)

cluster_sizes = dict(zip(unique, counts))
```

```
print("Cluster sizes:")

for cluster, size in cluster_sizes.items():

    print(f"Cluster {cluster}: {size}")
```

Output : `Cluster sizes:`

```
Cluster 0: 65

Cluster 1: 51

Cluster 2: 62
```

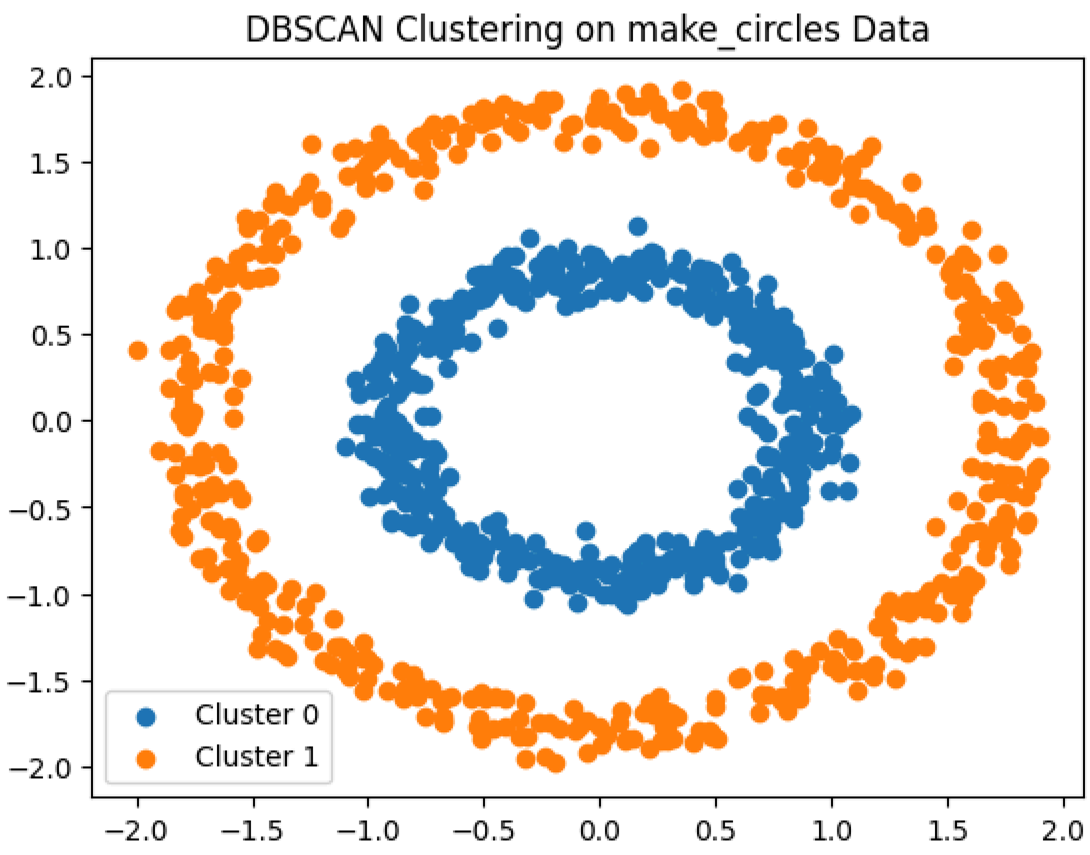
25. Use make_circles to generate synthetic data and cluster it using DBSCAN. Plot the result.

Ans: `import numpy as np`

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_circles

from sklearn.cluster import DBSCAN

from sklearn.preprocessing import StandardScaler



# Generate synthetic circular data

X, y = make_circles(n_samples=1000, factor=0.5, noise=0.05,
random_state=42)



# Scale the data

X_scaled = StandardScaler().fit_transform(X)
```

```python
# Apply DBSCAN

dbscan = DBSCAN(eps=0.3, min_samples=10)

labels = dbscan.fit_predict(X_scaled)


# Plot the clustering result

plt.figure()

unique_labels = set(labels)


for label in unique_labels:

    mask = labels == label

    plt.scatter(

        X_scaled[mask, 0],

        X_scaled[mask, 1],

        label=f"Cluster {label}" if label != -1 else "Noise"

    )


plt.title("DBSCAN Clustering on make_circles Data")

plt.legend()

plt.show()
```

Output:



26. Load the Breast Cancer dataset, apply MinMaxScaler, and use K-Means with 2 clusters. Output the cluster centroids.

Ans:

```
from sklearn.datasets import load_breast_cancer

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

import pandas as pd



# Load dataset

data = load_breast_cancer()
```

```
X = data.data


# Apply MinMaxScaler

scaler = MinMaxScaler()

X_scaled = scaler.fit_transform(X)


# Apply K-Means with 2 clusters

kmeans = KMeans(n_clusters=2, random_state=42)

kmeans.fit(X_scaled)


# Get cluster centroids

centroids = kmeans.cluster_centers_


# Convert centroids to DataFrame for readability

centroids_df = pd.DataFrame(centroids, columns=data.feature_names)


print("Cluster Centroids:")

print(centroids_df)
```

OutPut:

Cluster Centroids:

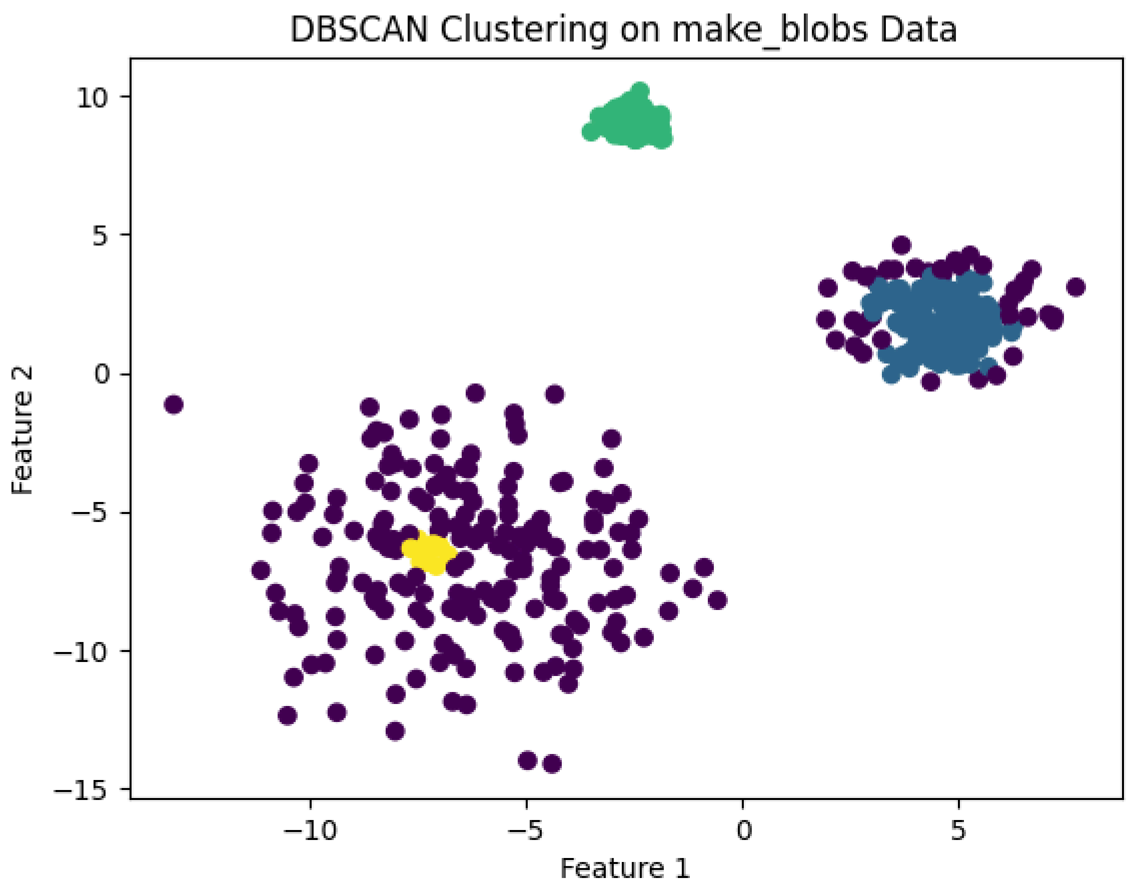
```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0     0.504836      0.395603        0.505787   0.363766         0.469887

1     0.255354      0.288335        0.246964   0.143884         0.357431


   mean compactness  mean concavity  mean concave points  mean symmetry  \

0          0.422263        0.418387             0.46928       0.458997

1          0.180195        0.103448             0.13066       0.340118


   mean fractal dimension  ...  worst radius  worst texture  worst
perimeter  \

0                0.299459  ...      0.480474       0.451074
0.465530

1                0.255916  ...      0.205241       0.320690
0.192421


   worst area  worst smoothness  worst compactness  worst concavity  \

0    0.314606          0.498688           0.363915         0.390273

1    0.099434          0.357112           0.148739         0.131423


   worst concave points  worst symmetry  worst fractal dimension

0              0.658272        0.337523                 0.260414

1              0.262314        0.226394                 0.154374


[2 rows x 30 columns]
```

27.Generate synthetic data using make_blobs with varying cluster standard deviations and cluster with DBSCAN

Ans: `from sklearn.datasets import make_blobs`

```
from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt



# Step 1: Generate synthetic data with varying cluster std

X, y = make_blobs(

    n_samples=600,

    centers=3,

    cluster_std=[0.3, 1.0, 2.5],  # different densities

    random_state=42

)



# Step 2: Apply DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=10)

labels = dbscan.fit_predict(X)



# Step 3: Plot results

plt.scatter(X[:, 0], X[:, 1], c=labels)

plt.title("DBSCAN Clustering on make_blobs Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()
```

Output:



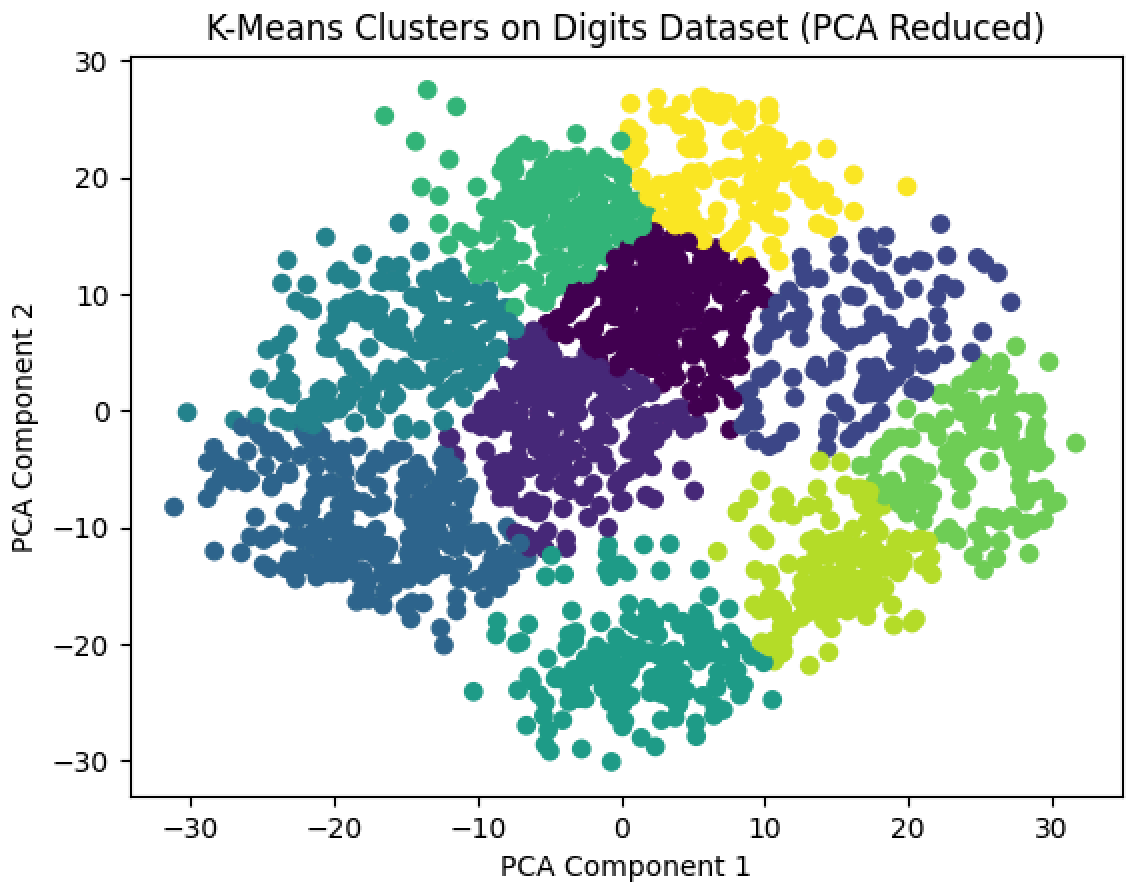
28. Load the Digits dataset, reduce it to 2D using PCA, and visualize clusters from K-Means.

Ans:

```
# Load required libraries

from sklearn.datasets import load_digits

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt



# Load Digits dataset
```

```
digits = load_digits()

X = digits.data

y = digits.target


# Reduce dimensions to 2D using PCA

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X)


# Apply K-Means clustering

kmeans = KMeans(n_clusters=10, random_state=42)

clusters = kmeans.fit_predict(X_pca)


# Visualize the clusters

plt.figure()

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters)

plt.xlabel("PCA Component 1")

plt.ylabel("PCA Component 2")

plt.title("K-Means Clusters on Digits Dataset (PCA Reduced)")

plt.show()
```

OutPut:



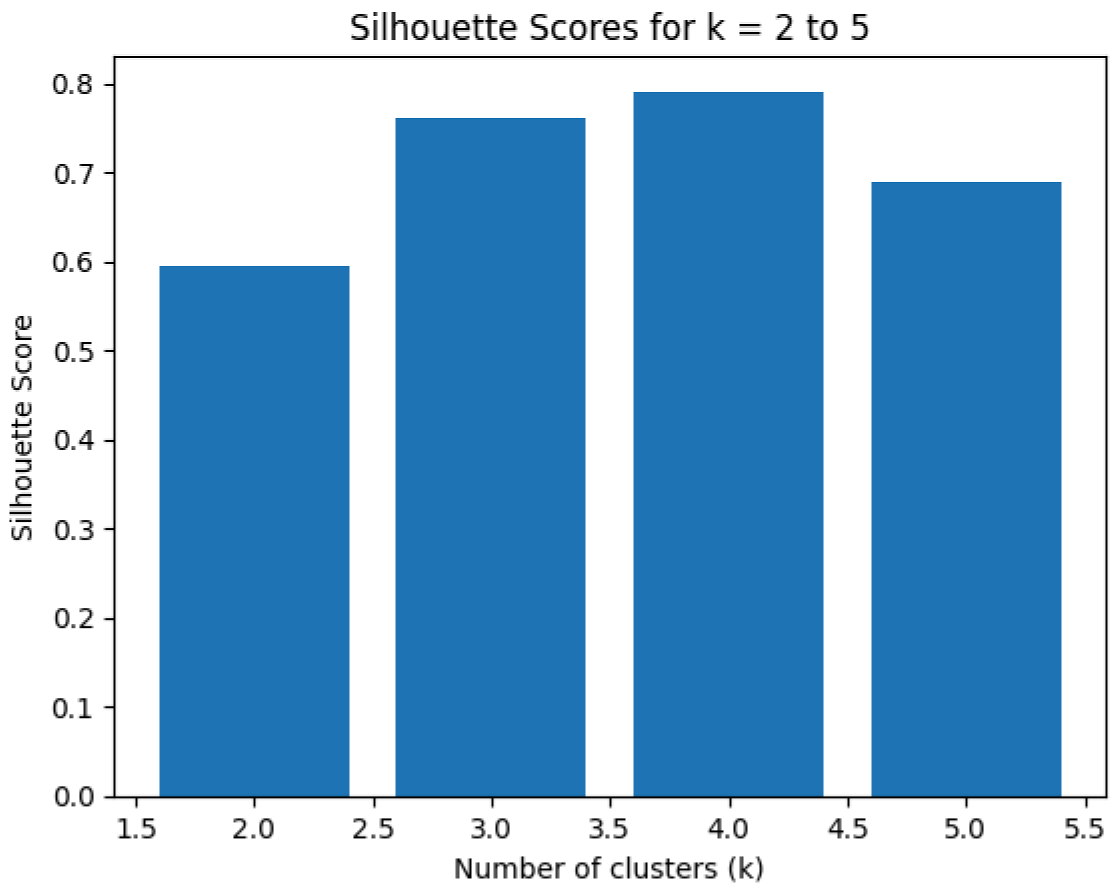
26.Create synthetic data using make_blobs and evaluate silhouette scores for k = 2 to 5. Display as a bar chart.

Ans:

```
from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

import matplotlib.pyplot as plt



# Create synthetic data

X, _ = make_blobs(n_samples=500, centers=4, cluster_std=1.0,
random_state=42)
```

```
# Evaluate silhouette scores for k = 2 to 5
k_values = range(2, 6)
sil_scores = []

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X)
    score = silhouette_score(X, labels)
    sil_scores.append(score)

# Display as bar chart
plt.figure()
plt.bar(k_values, sil_scores)
plt.xlabel("Number of clusters (k)")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Scores for k = 2 to 5")
plt.show()
```

Output:



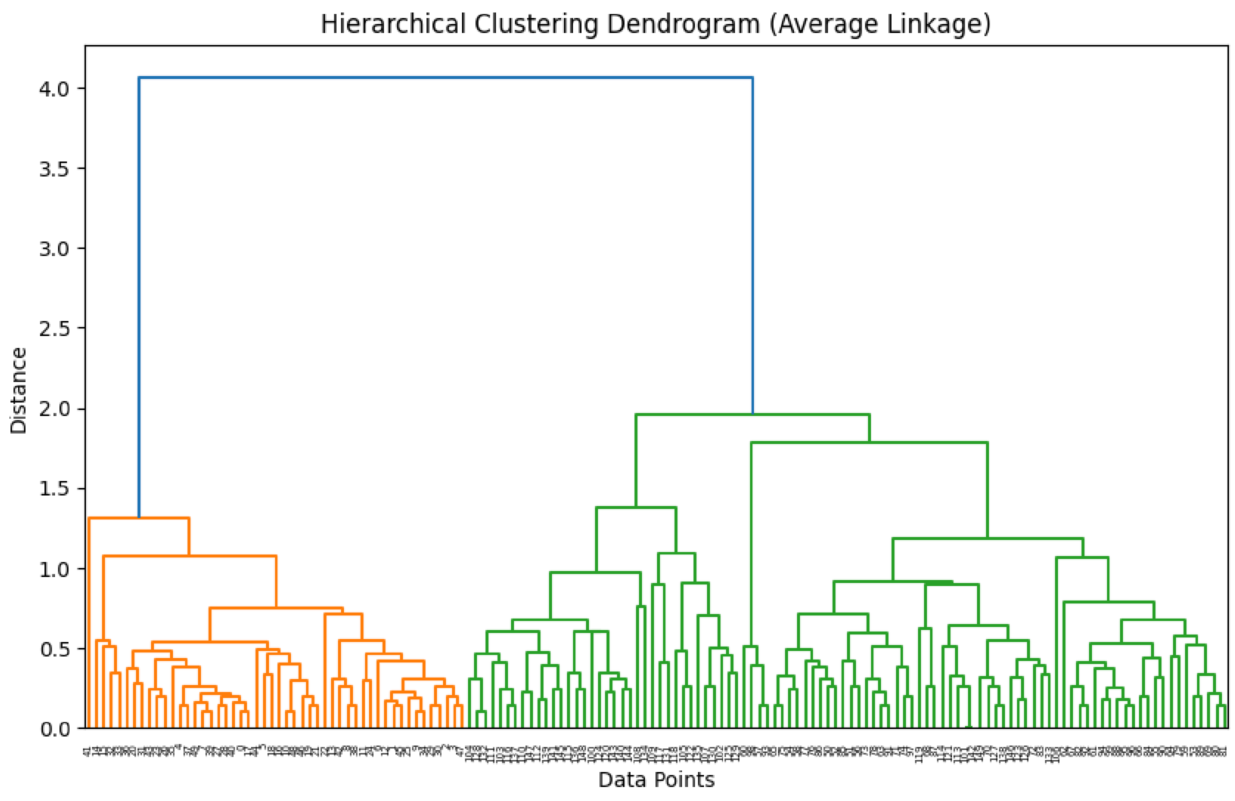
30. Load the Iris dataset and use hierarchical clustering to group data. Plot a dendrogram with average linkage.

Ans:

```
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from scipy.cluster.hierarchy import dendrogram, linkage



# Load Iris dataset

iris = load_iris()

X = iris.data  # features
```

```python
# Perform hierarchical clustering with average linkage

Z = linkage(X, method='average')


# Plot dendrogram

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

dendrogram(Z)

plt.title("Hierarchical Clustering Dendrogram (Average Linkage)")

plt.xlabel("Data Points")

plt.ylabel("Distance")

plt.show()
```

Output:



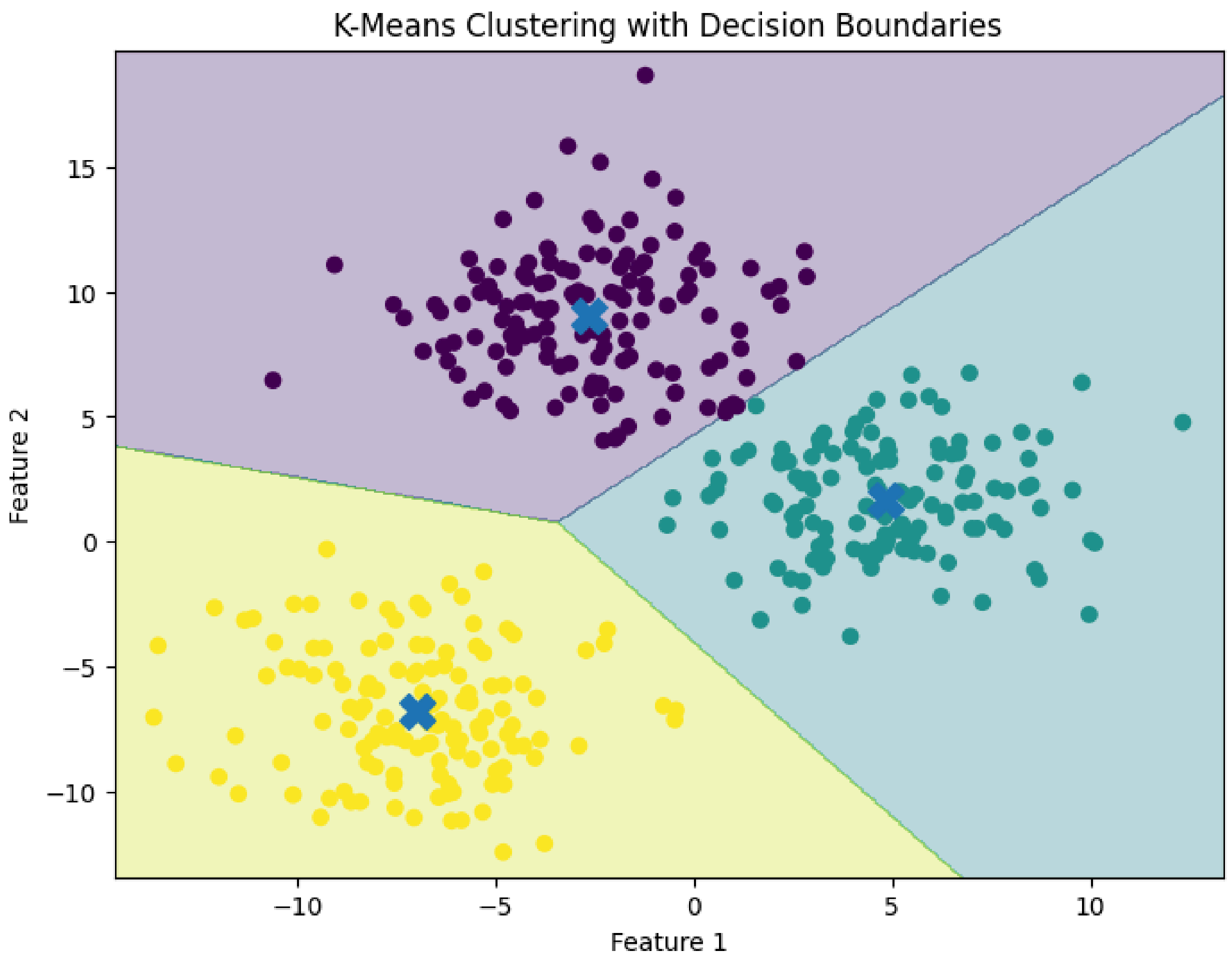
31. Generate synthetic data with overlapping clusters using make_blobs, then apply K-Means and visualize with decision boundaries.

Ans:

```python
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans


# 1. Generate synthetic data with overlapping clusters

X, y = make_blobs(

    n_samples=400,

    centers=3,

    cluster_std=2.5,   # higher std → more overlap

    random_state=42

)


# 2. Apply K-Means

kmeans = KMeans(n_clusters=3, random_state=42)

kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_


# 3. Create meshgrid for decision boundaries

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1


xx, yy = np.meshgrid(
    np.linspace(x_min, x_max, 500),
    np.linspace(y_min, y_max, 500)
)


Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)


# 4. Visualization
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], marker='X', s=200)
plt.title("K-Means Clustering with Decision Boundaries")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Output:



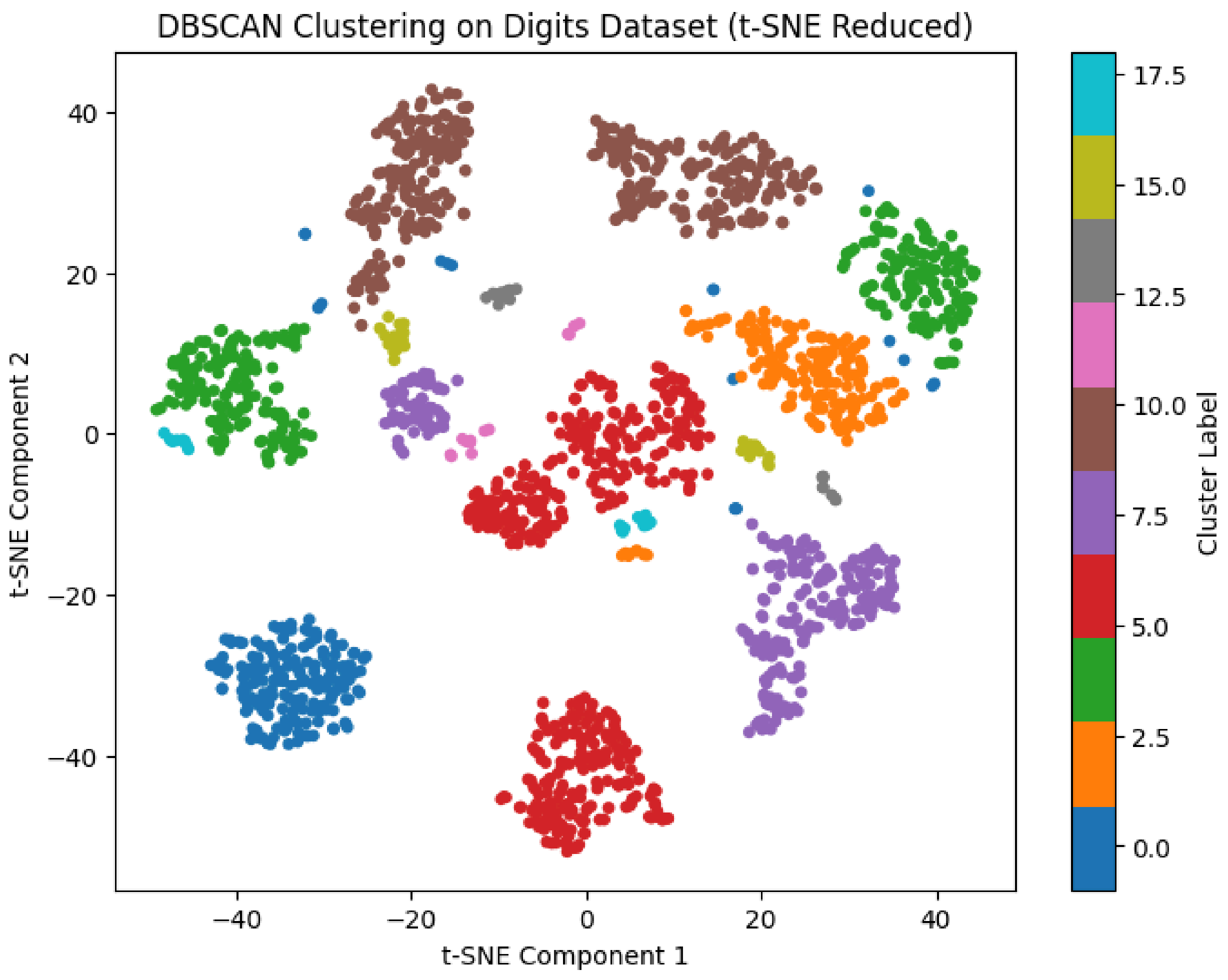
32. Load the Digits dataset and apply DBSCAN after reducing dimensions with t-SNE. Visualize the results.

Ans:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_digits

from sklearn.manifold import TSNE

from sklearn.cluster import DBSCAN

from sklearn.preprocessing import StandardScaler
```

```
# Load Digits dataset
digits = load_digits()
X = digits.data
y = digits.target

# Standardize features
X_scaled = StandardScaler().fit_transform(X)

# Reduce dimensions using t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
X_tsne = tsne.fit_transform(X_scaled)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=2.5, min_samples=5)
clusters = dbscan.fit_predict(X_tsne)

# Visualize results
plt.figure(figsize=(8, 6))
scatter = plt.scatter(
    X_tsne[:, 0],
    X_tsne[:, 1],
    c=clusters,
    cmap="tab10",

    s=15

)

plt.colorbar(scatter, label="Cluster Label")

plt.title("DBSCAN Clustering on Digits Dataset (t-SNE Reduced)")

plt.xlabel("t-SNE Component 1")

plt.ylabel("t-SNE Component 2")

plt.show()
```

Output:



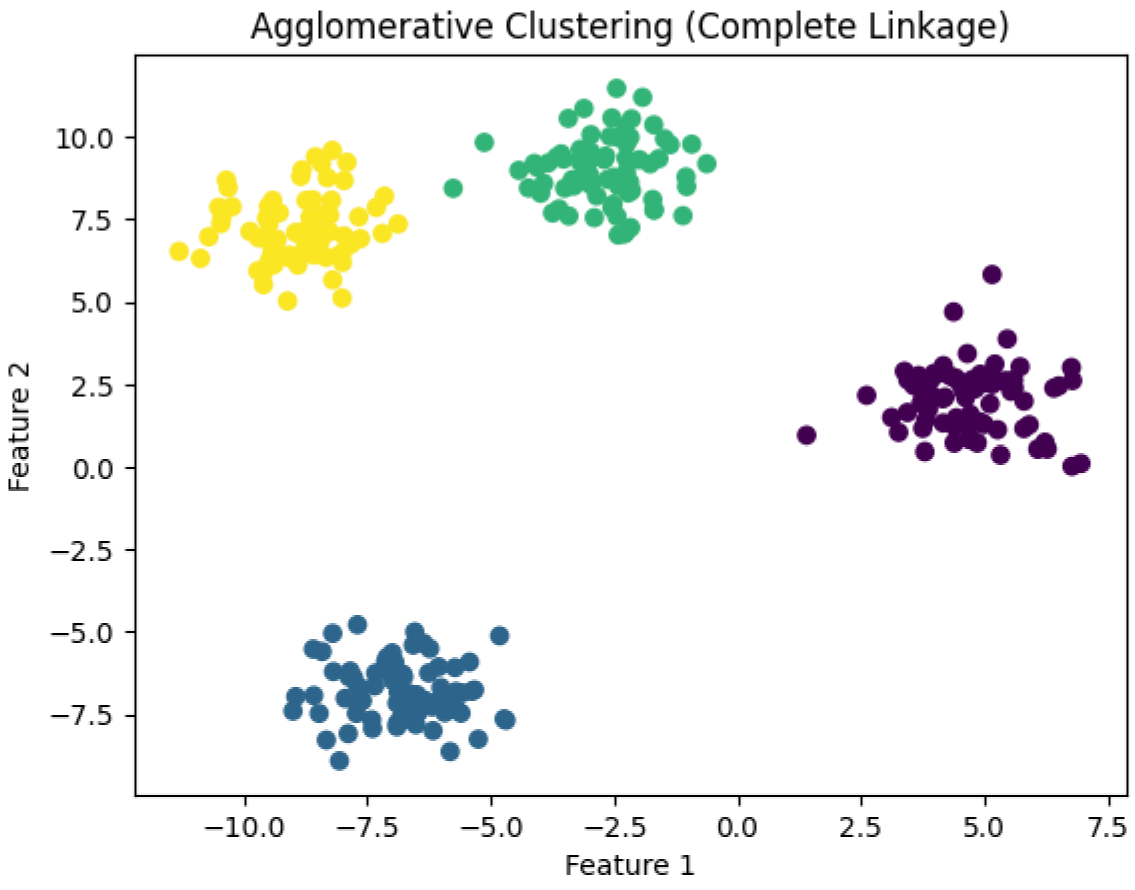
33. Generate synthetic data using make_blobs and apply Agglomerative Clustering with complete linkage. Plot the result.

Ans:

```python
# Generate synthetic data and apply Agglomerative Clustering (Complete Linkage)


import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.cluster import AgglomerativeClustering


# Step 1: Generate synthetic data

X, y = make_blobs(

    n_samples=300,

    centers=4,

    cluster_std=1.0,

    random_state=42

)


# Step 2: Apply Agglomerative Clustering with complete linkage

agglo = AgglomerativeClustering(

    n_clusters=4,

    linkage='complete'

)

labels = agglo.fit_predict(X)
```

```
# Step 3: Plot the clustering result

plt.figure()

plt.scatter(X[:, 0], X[:, 1], c=labels)

plt.title("Agglomerative Clustering (Complete Linkage)")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()
```

Output:

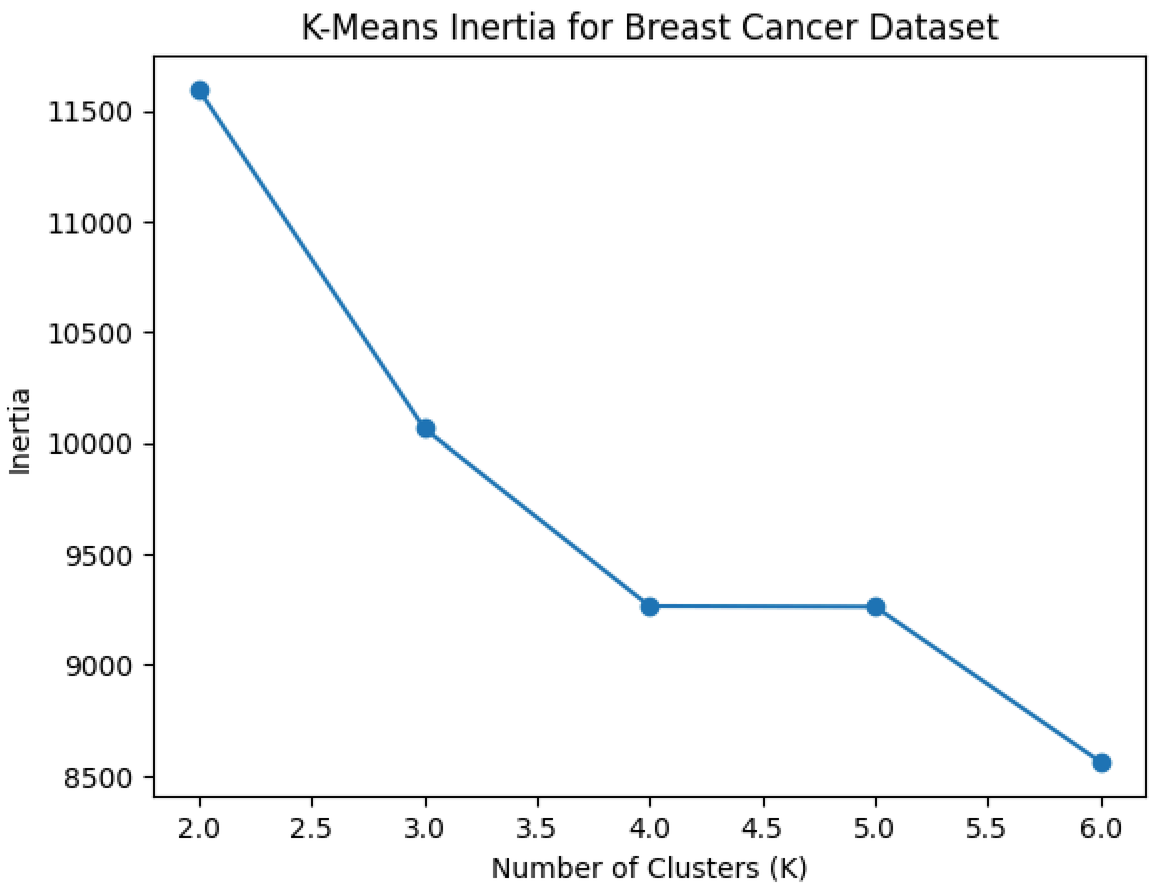


34. Load the Breast Cancer dataset and compare inertia values for K = 2 to 6 using K-Means. Show results in a line plot.

Ans:

```
# Import required libraries

import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler


# Load the Breast Cancer dataset

data = load_breast_cancer()

X = data.data


# Standardize the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Calculate inertia for K = 2 to 6

inertia_values = []

k_values = range(2, 7)


for k in k_values:

    kmeans = KMeans(n_clusters=k, random_state=42)

    kmeans.fit(X_scaled)
    inertia_values.append(kmeans.inertia_)


# Plot inertia vs K

plt.figure()

plt.plot(k_values, inertia_values, marker='o')

plt.xlabel('Number of Clusters (K)')

plt.ylabel('Inertia')

plt.title('K-Means Inertia for Breast Cancer Dataset')

plt.show()
```

Output:

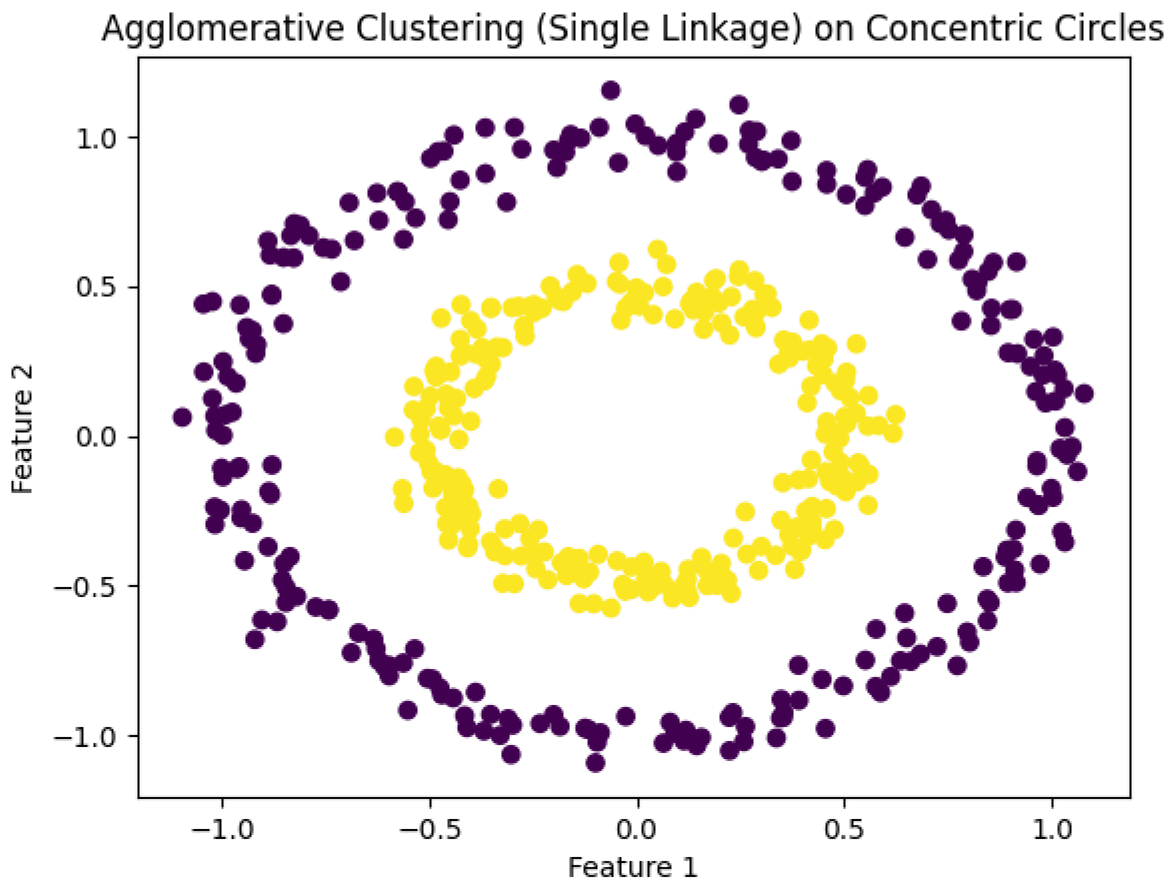


35. Generate synthetic concentric circles using make_circles and cluster using Agglomerative Clustering with single linkage.

Ans:

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_circles

from sklearn.cluster import AgglomerativeClustering



# Step 1: Generate synthetic concentric circles

X, y = make_circles(n_samples=500, factor=0.5, noise=0.05,
random_state=42)



# Step 2: Apply Agglomerative Clustering with single linkage

model = AgglomerativeClustering(

    n_clusters=2,

    linkage='single'

)



labels = model.fit_predict(X)



# Step 3: Visualize the clustering output

plt.scatter(X[:, 0], X[:, 1], c=labels)

plt.title("Agglomerative Clustering (Single Linkage) on Concentric
Circles")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")
plt.show()
```

Output:



36. Use the Wine dataset, apply DBSCAN after scaling the data, and count the number of clusters (excluding noise).

Ans: `from sklearn.datasets import load_wine`

```
from sklearn.preprocessing import StandardScaler

from sklearn.cluster import DBSCAN

import numpy as np



# Load Wine dataset
```

```
data = load_wine()

X = data.data



# Scale the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)



# Apply DBSCAN

dbscan = DBSCAN(eps=1.5, min_samples=5)

labels = dbscan.fit_predict(X_scaled)



# Count clusters (excluding noise label -1)

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



print("Number of clusters (excluding noise):", n_clusters)
```

Output:

```
Number of clusters (excluding noise): 0
```

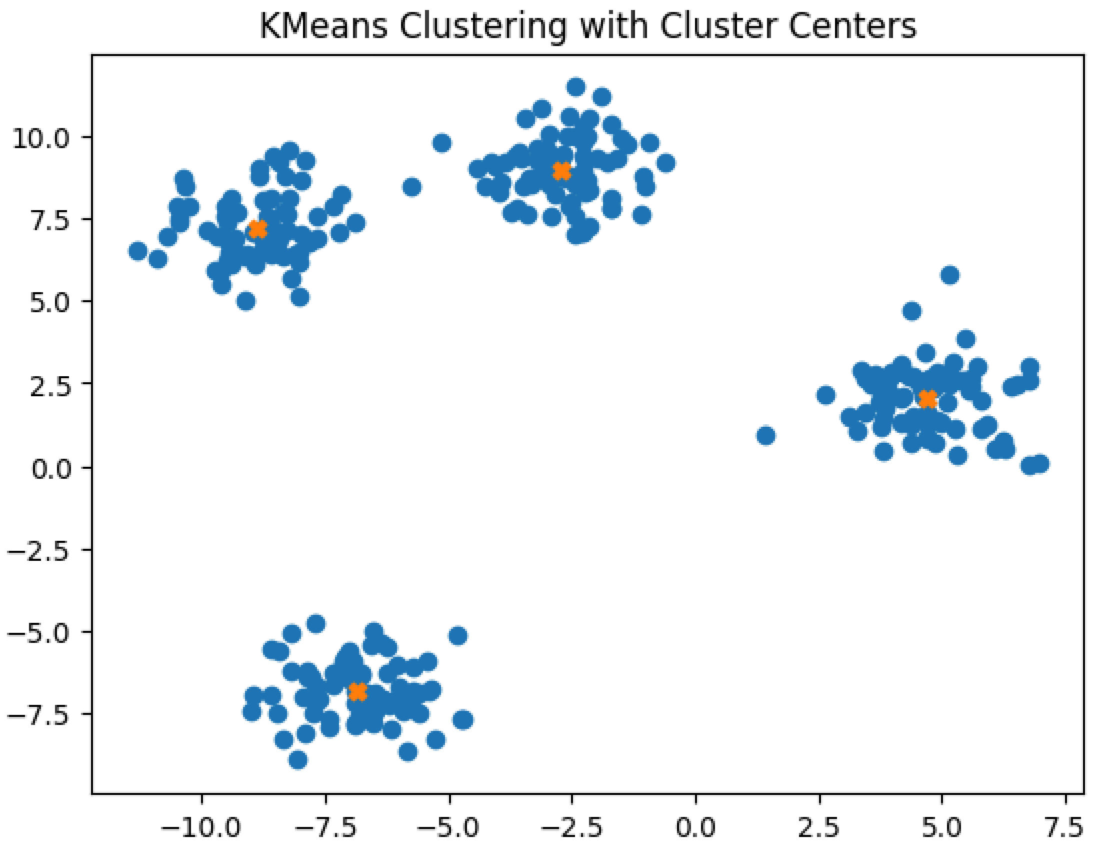
37. Generate synthetic data with make_blobs and apply KMeans. Then plot the cluster centers on top of the data points.

Ans:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
```

```python
from sklearn.cluster import KMeans


# Generate synthetic data

X, y = make_blobs(

    n_samples=300,

    centers=4,

    cluster_std=1.0,

    random_state=42

)


# Apply KMeans

kmeans = KMeans(n_clusters=4, random_state=42)

kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_


# Plot data points

plt.scatter(X[:, 0], X[:, 1])

# Plot cluster centers

plt.scatter(centers[:, 0], centers[:, 1], marker='X')


plt.title("KMeans Clustering with Cluster Centers")

plt.show()
```

OutPut:



38. Load the Iris dataset, cluster with DBSCAN, and print how many samples were identified as noise.

Ans:

```
from sklearn.datasets import load_iris

from sklearn.cluster import DBSCAN

from sklearn.preprocessing import StandardScaler

import numpy as np



# Load Iris dataset
```

```
iris = load_iris()

X = iris.data


# Scale features (important for DBSCAN)

X_scaled = StandardScaler().fit_transform(X)


# Apply DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=5)

labels = dbscan.fit_predict(X_scaled)


# Count noise points (label = -1)

noise_count = np.sum(labels == -1)


print("Number of samples identified as noise:", noise_count)
```

Output: `Number of samples identified as noise: 34`

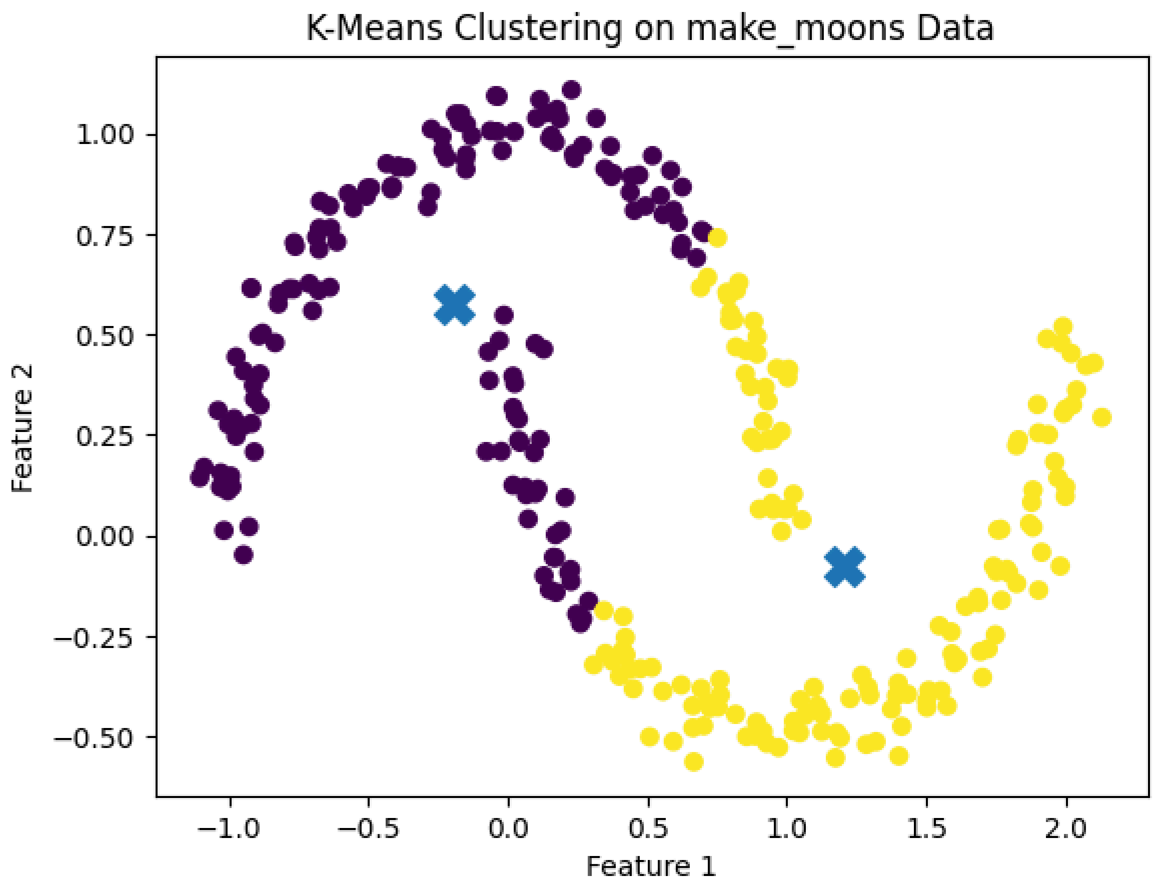
39. Generate synthetic non-linearly separable data using make_moons, apply K-Means, and visualize the clustering result.

Ans: `import numpy as np`

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_moons

from sklearn.cluster import KMeans
```

```python
# 1. Generate synthetic non-linearly separable data

X, y = make_moons(n_samples=300, noise=0.05, random_state=42)


# 2. Apply K-Means clustering

kmeans = KMeans(n_clusters=2, random_state=42)

labels = kmeans.fit_predict(X)


# 3. Visualize the clustering result

plt.figure()

plt.scatter(X[:, 0], X[:, 1], c=labels)

plt.scatter(kmeans.cluster_centers_[:, 0],

            kmeans.cluster_centers_[:, 1],

            marker='X', s=200)

plt.title("K-Means Clustering on make_moons Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()
```

Output:

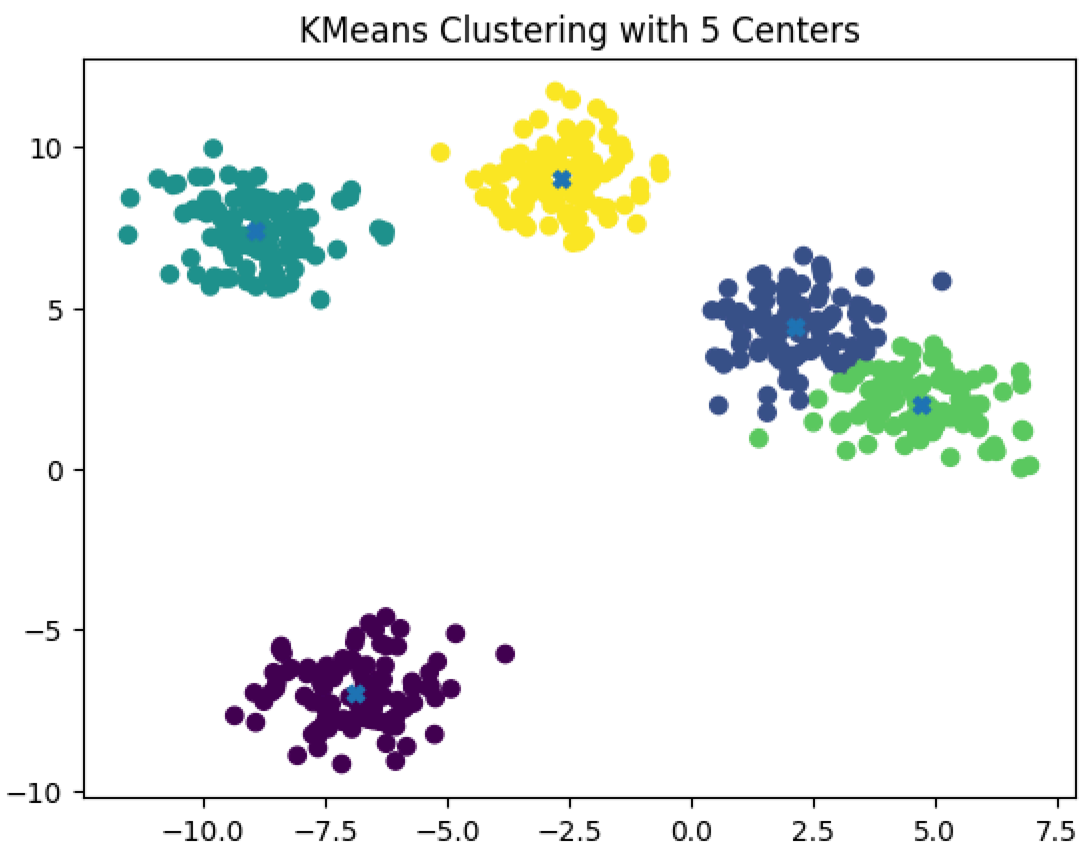


41. Generate synthetic blobs with 5 centers and apply KMeans. Then use silhouette_score to evaluate the clustering.

Ans:

```
from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

import matplotlib.pyplot as plt



# Generate synthetic data with 5 centers

X, y = make_blobs(
    n_samples=500,

    centers=5,

    cluster_std=1.0,

    random_state=42

)


# Apply KMeans

kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)

labels = kmeans.fit_predict(X)


# Evaluate clustering using Silhouette Score

score = silhouette_score(X, labels)

print("Silhouette Score:", score)


# Optional: visualize clusters

plt.scatter(X[:, 0], X[:, 1], c=labels)

plt.scatter(

    kmeans.cluster_centers_[:, 0],

    kmeans.cluster_centers_[:, 1],

    marker='X'

)

plt.title("KMeans Clustering with 5 Centers")

plt.show()
```

Output:



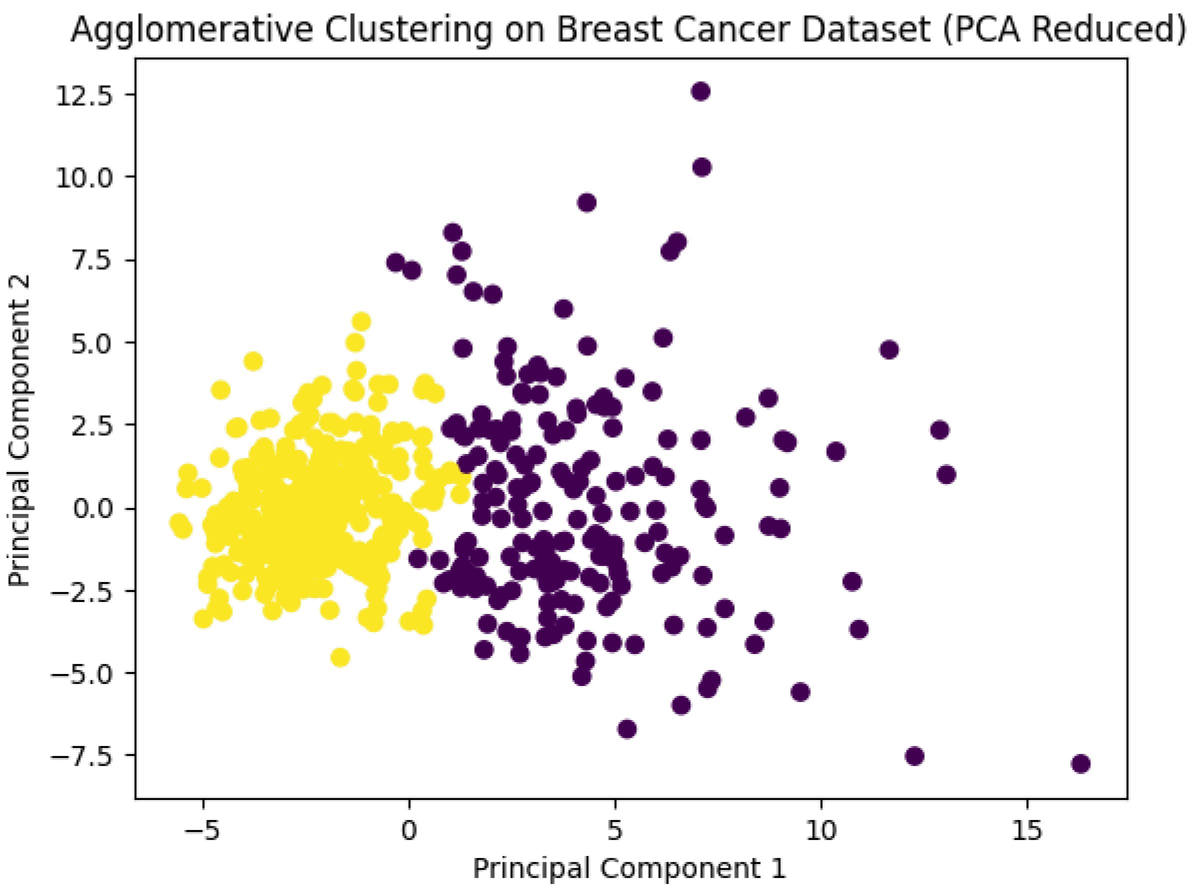
42. Load the Breast Cancer dataset, reduce dimensionality using PCA, and apply Agglomerative Clustering. Visualize in 2D.

Ans:

```
# Import required libraries

import numpy as np

import matplotlib.pyplot as plt



from sklearn.datasets import load_breast_cancer

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA
```

```python
from sklearn.cluster import AgglomerativeClustering


# 1. Load Breast Cancer dataset

data = load_breast_cancer()

X = data.data


# 2. Standardize the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# 3. Apply PCA (reduce to 2 dimensions)

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)


# 4. Apply Agglomerative Clustering

agglo = AgglomerativeClustering(n_clusters=2, linkage='ward')

clusters = agglo.fit_predict(X_pca)


# 5. Visualize the clusters in 2D

plt.figure()

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters)

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")
plt.title("Agglomerative Clustering on Breast Cancer Dataset (PCA
Reduced)")

plt.show()
```

Output:



43. Generate noisy circular data using make_circles and visualize clustering results from KMeans and DBSCAN side-by-side.

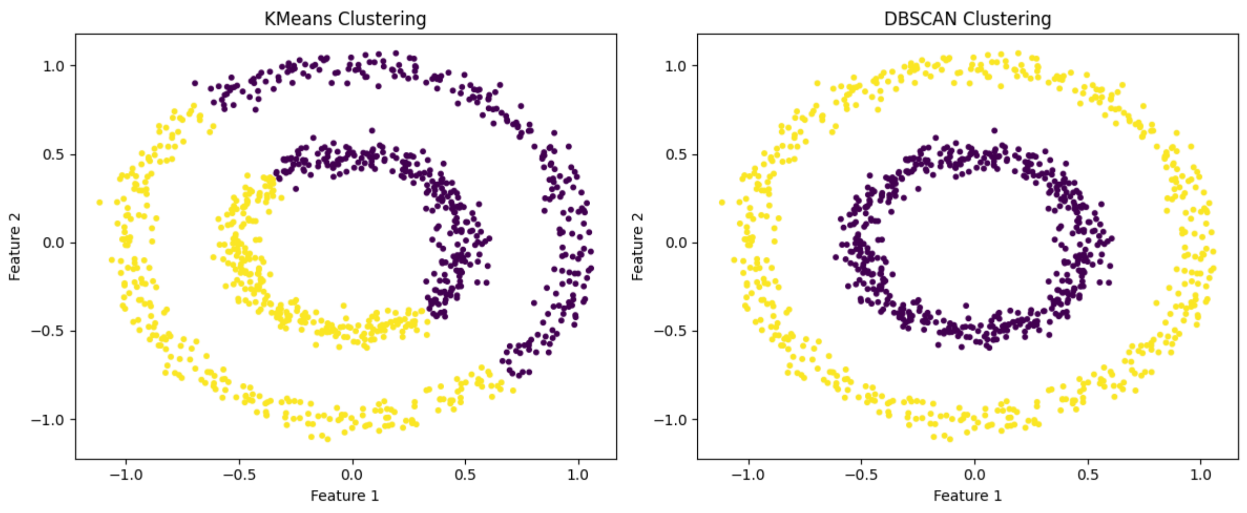
Ans:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_circles

from sklearn.cluster import KMeans, DBSCAN
```

```
# 1. Generate noisy circular data

X, y = make_circles(n_samples=1000, factor=0.5, noise=0.05,
random_state=42)


# 2. Apply KMeans clustering

kmeans = KMeans(n_clusters=2, random_state=42)

kmeans_labels = kmeans.fit_predict(X)


# 3. Apply DBSCAN clustering

dbscan = DBSCAN(eps=0.2, min_samples=5)

dbscan_labels = dbscan.fit_predict(X)


# 4. Visualization: side-by-side comparison

plt.figure(figsize=(12, 5))


# KMeans Plot

plt.subplot(1, 2, 1)

plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, s=10)

plt.title("KMeans Clustering")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")


# DBSCAN Plot
```

```
plt.subplot(1, 2, 2)

plt.scatter(X[:, 0], X[:, 1], c=dbscan_labels, s=10)

plt.title("DBSCAN Clustering")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")



plt.tight_layout()

plt.show()
```

Output:

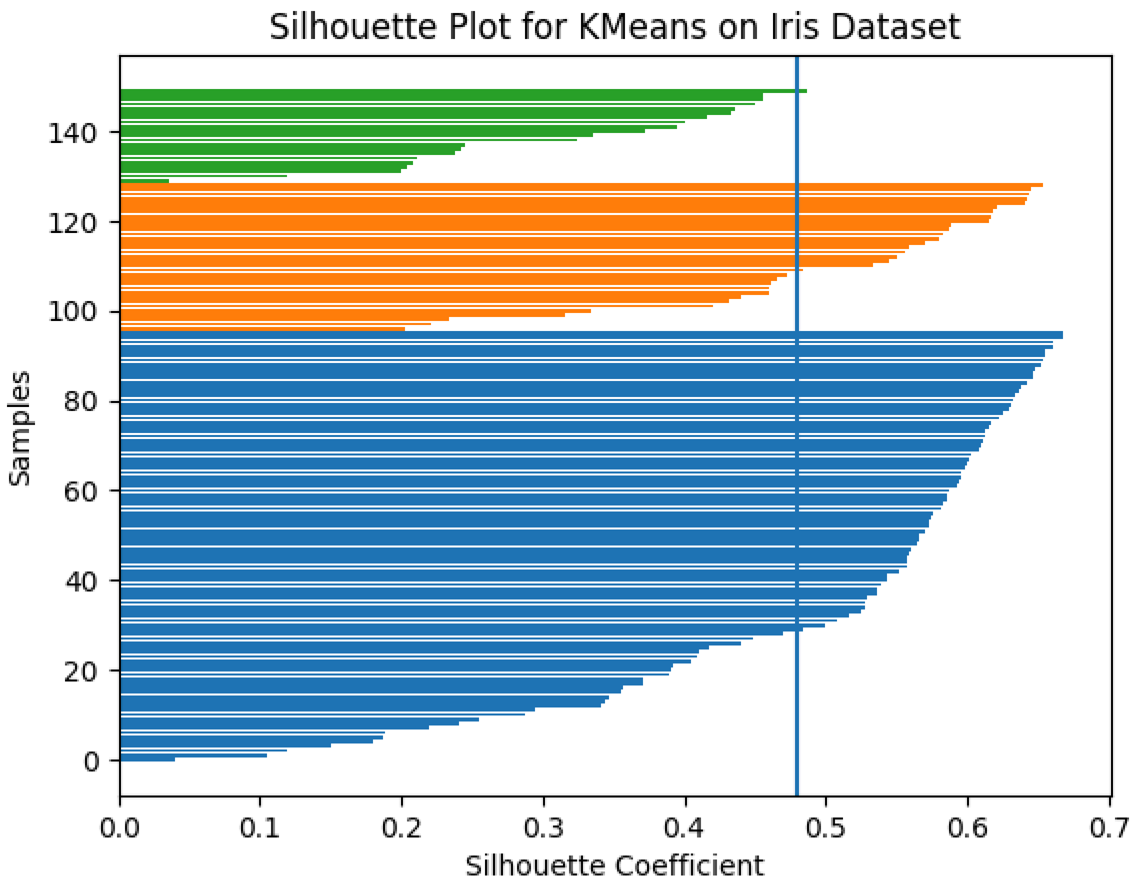


44. Load the Iris dataset and plot the Silhouette Coefficient for each sample after KMeans clustering.

Ans:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_samples, silhouette_score

from sklearn.preprocessing import StandardScaler


# Load Iris dataset

iris = load_iris()

X = iris.data

# Feature scaling (best practice for KMeans)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

# Apply KMeans

k = 3

kmeans = KMeans(n_clusters=k, random_state=42)

labels = kmeans.fit_predict(X_scaled)

# Compute silhouette values

silhouette_vals = silhouette_samples(X_scaled, labels)

avg_silzouette = silhouette_score(X_scaled, labels)

# Plot Silhouette Coefficient for each sample

y_lower = 0

plt.figure()


for i in range(k):

    cluster_silhouette_vals = silhouette_vals[labels == i]
    cluster_silhouette_vals.sort()

    y_upper = y_lower + len(cluster_silhouette_vals)

    plt.barh(range(y_lower, y_upper),

             cluster_silhouette_vals)

    y_lower = y_upper

plt.axvline(avg_silhouette)

plt.xlabel("Silhouette Coefficient")

plt.ylabel("Samples")

plt.title("Silhouette Plot for KMeans on Iris Dataset")

plt.show()
```

Output:



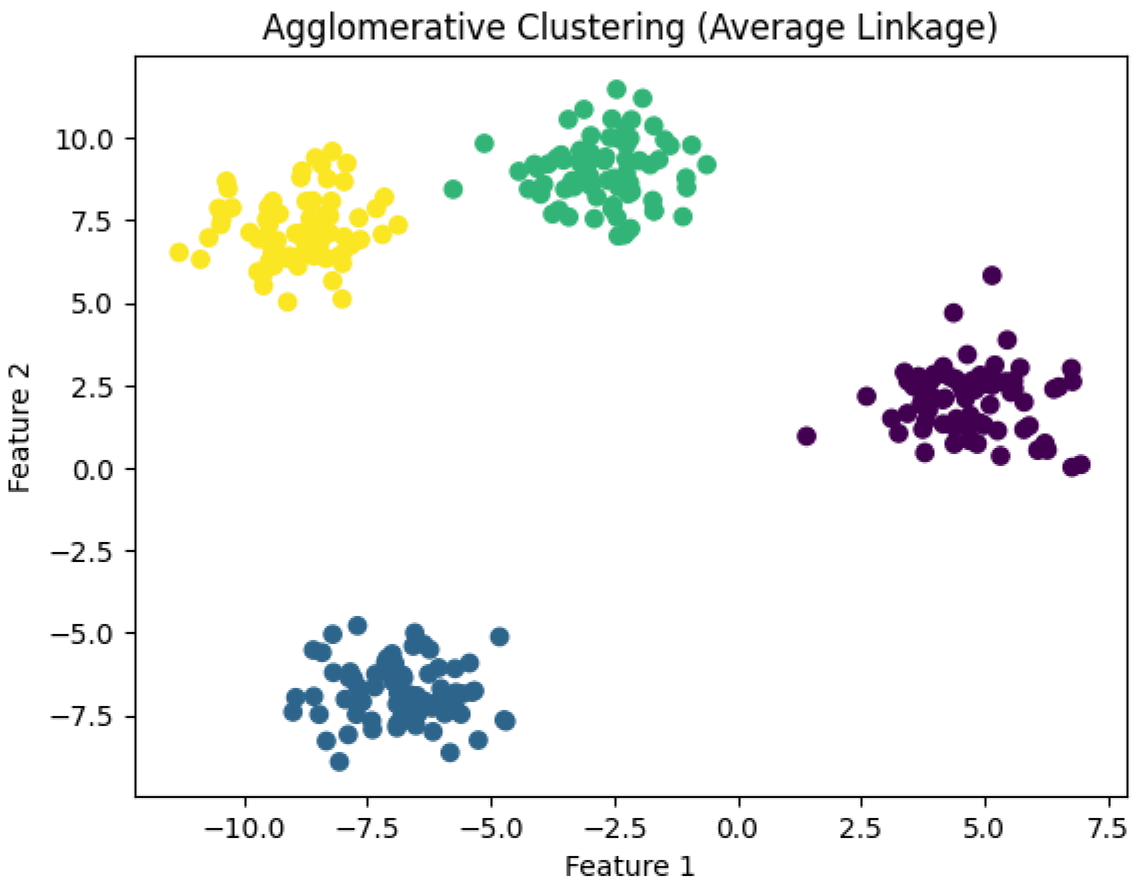
45. Generate synthetic data using make_blobs and apply Agglomerative Clustering with 'average' linkage. Visualize clusters.

Ans:

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.cluster import AgglomerativeClustering



# Step 1: Generate synthetic data

X, y_true = make_blobs(

    n_samples=300,

    centers=4,

    cluster_std=1.0,

    random_state=42

)



# Step 2: Apply Agglomerative Clustering with average linkage

agglo = AgglomerativeClustering(

    n_clusters=4,

    linkage='average'

)



y_pred = agglo.fit_predict(X)



# Step 3: Visualize clusters
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_pred)

plt.title("Agglomerative Clustering (Average Linkage)")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()
```

Output:



46. Load the Wine dataset, apply KMeans, and visualize the cluster assignments in a seaborn pairplot (first 4 features).

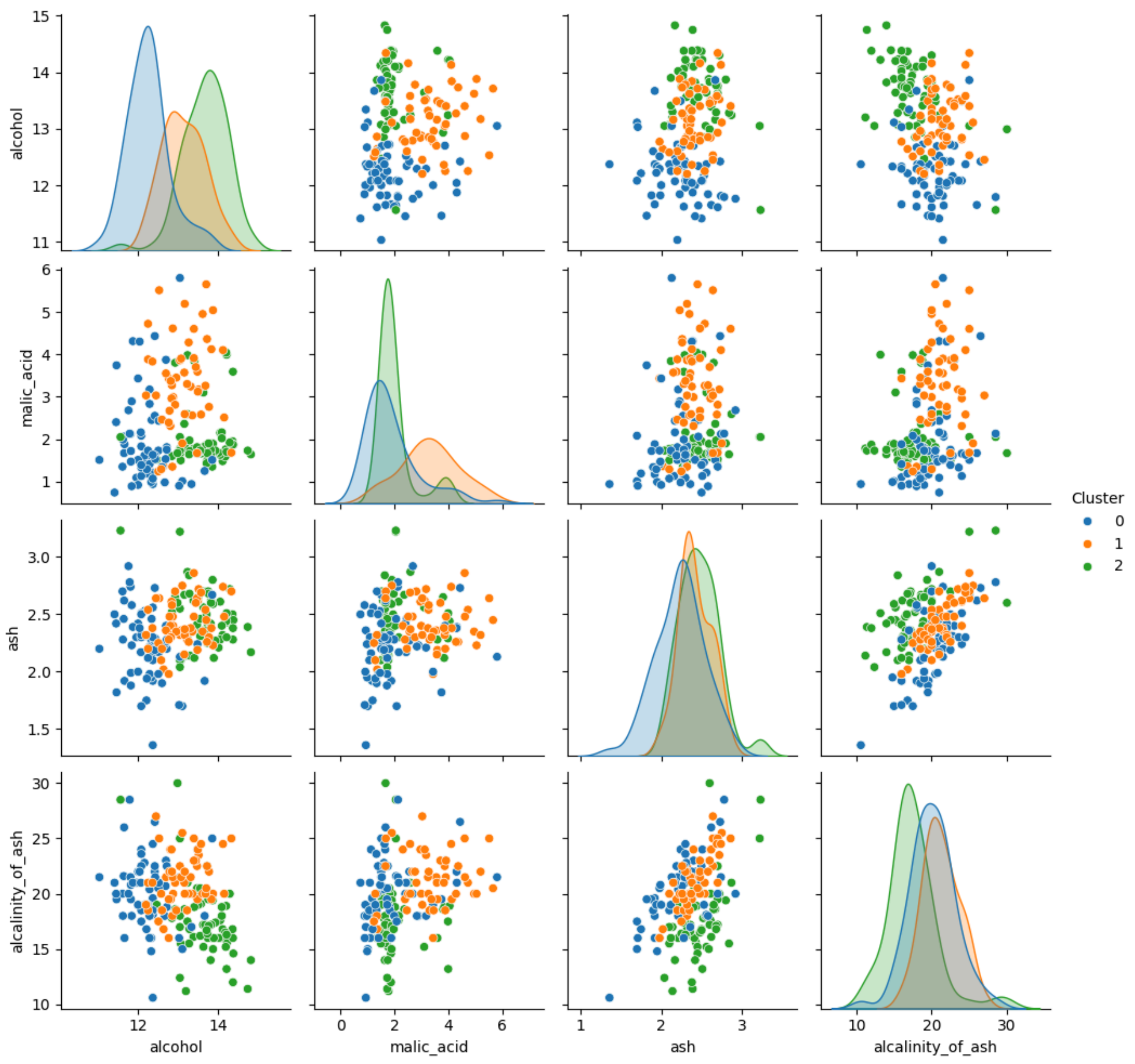
Ans:

```
# Load libraries

import pandas as pd
```

```python
import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load_wine

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler


# Load Wine dataset

wine = load_wine()

X = pd.DataFrame(wine.data, columns=wine.feature_names)


# Standardize features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Apply KMeans

kmeans = KMeans(n_clusters=3, random_state=42)

clusters = kmeans.fit_predict(X_scaled)


# Add cluster labels to dataframe

X_plot = X.iloc[:, :4].copy()   # first 4 features

X_plot["Cluster"] = clusters


# Seaborn pairplot
```

```
sns.pairplot(X_plot, hue="Cluster", palette="tab10")

plt.show()
```

Output :



47. Generate noisy blobs using make_blobs and use DBSCAN to identify both clusters and noise points. Print the count.

Ans:

```
# Import necessary libraries

from sklearn.datasets import make_blobs

from sklearn.cluster import DBSCAN

import numpy as np



# Step 1: Generate noisy blobs

X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0,
random_state=42)



# Add some random noise points

np.random.seed(42)

noise_points = np.random.uniform(low=-10, high=10, size=(20, 2))

X = np.vstack([X, noise_points])



# Step 2: Apply DBSCAN

dbscan = DBSCAN(eps=1.5, min_samples=5)  # eps and min_samples can be
tuned

labels = dbscan.fit_predict(X)



# Step 3: Count clusters and noise

n_clusters = len(set(labels)) - (1 if -1 in labels else 0)  # exclude
noise label (-1)
```

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


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

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

Output:

```
Number of clusters: 3

Number of noise points: 15
```

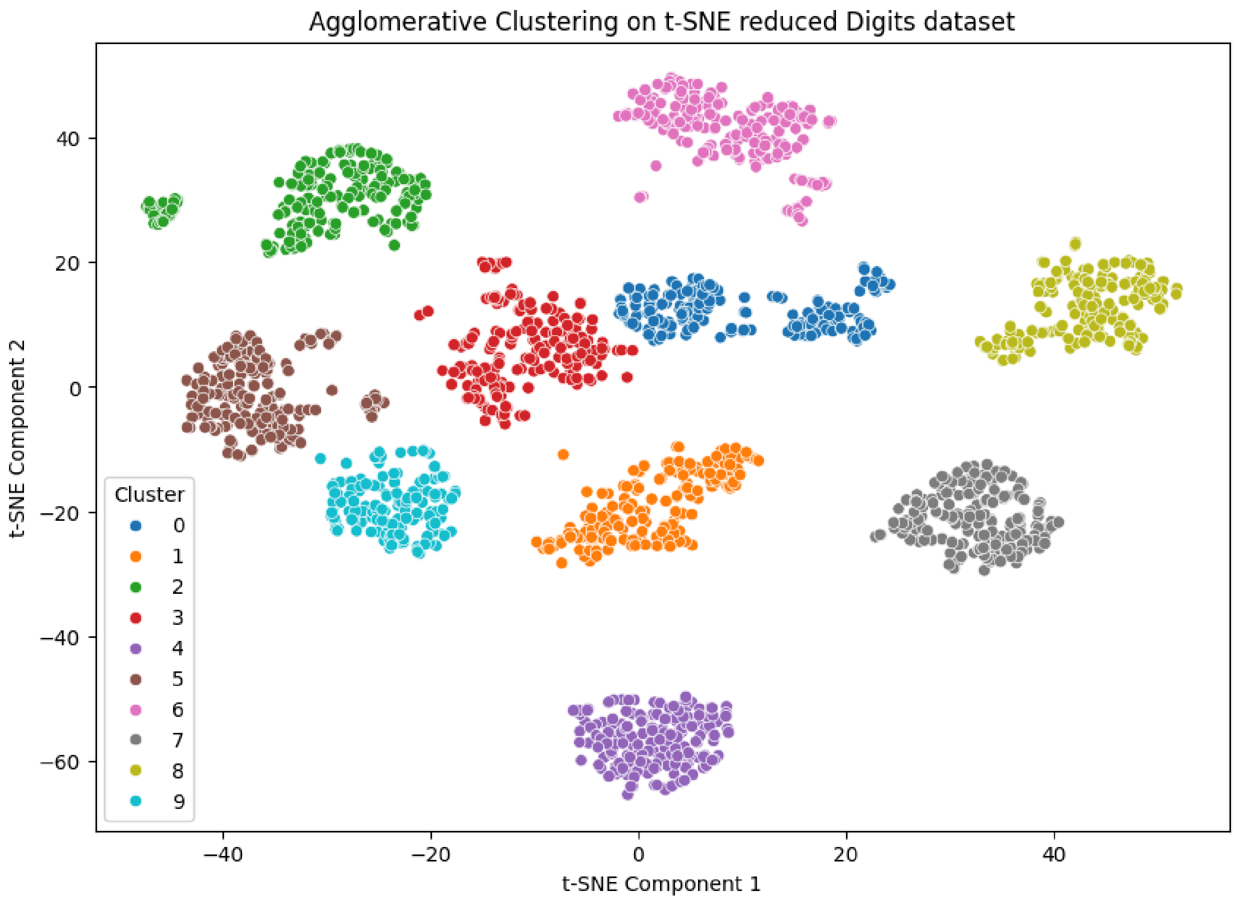
48. **Load the Digits dataset, reduce dimensions using t-SNE, then apply Agglomerative Clustering and plot the clusters.**

Ans:

```
# Import necessary libraries

import matplotlib.pyplot as plt

from sklearn.datasets import load_digits

from sklearn.manifold import TSNE

from sklearn.cluster import AgglomerativeClustering

import seaborn as sns


# Step 1: Load the Digits dataset

digits = load_digits()

X = digits.data

y = digits.target
```

```python
# Step 2: Reduce dimensions using t-SNE

tsne = TSNE(n_components=2, random_state=42, perplexity=30)

X_tsne = tsne.fit_transform(X)



# Step 3: Apply Agglomerative Clustering

agg_clust = AgglomerativeClustering(n_clusters=10)  # digits dataset has
10 classes (0-9)

clusters = agg_clust.fit_predict(X_tsne)



# Step 4: Plot the clusters

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

sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=clusters,
palette='tab10', legend='full')

plt.title("Agglomerative Clustering on t-SNE reduced Digits dataset")

plt.xlabel("t-SNE Component 1")

plt.ylabel("t-SNE Component 2")

plt.legend(title="Cluster")

plt.show()
```

Output:



Agglomerative Clustering on t-SNE reduced Digits dataset
t-SNE Component 1
t-SNE Component 2
Cluster
0
1
2
3
4
5
6
7
8
9
−40
−20
0
20
40
−60