Theory questions :

1) **What is unsupervised learning in the context of machine learning?** Unsupervised learning is a type of machine learning where the algorithm learns patterns from data without explicit labels. It is commonly used for clustering, anomaly detection, and dimensionality reduction.

2) **How does the K-Means clustering algorithm work?** K-Means clustering partitions data into **K clusters** by assigning each point to the nearest centroid, then iteratively updating centroids until convergence.

3) **Explain the concept of a dendrogram in hierarchical clustering.** A dendrogram is a tree-like diagram that visually represents how data points are grouped into clusters during hierarchical clustering.

4) **What is the main difference between K-Means and Hierarchical Clustering?** K-Means requires a predefined number of clusters and iteratively refines centroids, while hierarchical clustering builds a hierarchy without a predetermined number of clusters.

5) **What are the advantages of DBSCAN over K-Means?** DBSCAN doesn't require a fixed number of clusters, identifies noise points, and can handle clusters of arbitrary shapes.

6) **When would you use Silhouette Score in clustering?** Silhouette Score measures the quality of clustering by evaluating how well-separated the clusters are.

7) **What are the limitations of Hierarchical Clustering?**
Hierarchical clustering is computationally expensive, sensitive to noise, and struggles with large datasets.

8) **Why is feature scaling important in clustering algorithms like K-Means?**
Feature scaling ensures that all features contribute equally to distance calculations, preventing biased clustering.

9) **How does DBSCAN identify noise points?**
DBSCAN classifies points as noise if they don't belong to a dense cluster.

10) **Define inertia in the context of K-Means.**
Inertia measures how tightly grouped data points are around centroids; lower inertia suggests better clustering.

11) **What is the elbow method in K-Means clustering?**
The elbow method helps determine the optimal number of clusters by plotting inertia values and identifying the point of diminishing returns.

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

Density refers to the number of points within a neighborhood radius, which defines clusters.

13) Can hierarchical clustering be used on categorical data?

Yes, hierarchical clustering can be applied using appropriate distance metrics for categorical data.

14) What does a negative Silhouette Score indicate?

A negative Silhouette Score suggests poor clustering, indicating that a sample might be assigned to the wrong cluster.

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

Linkage criteria define how distances between clusters are calculated, influencing how clusters merge.

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

K-Means assumes uniform cluster distribution, so it struggles with identifying clusters of different sizes and densities.

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

The core parameters in DBSCAN are *Epsilon* (defines the radius for clustering) and *MinPts* (minimum points needed to form a dense cluster).

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

K-Means++ improves initialization by selecting better starting centroids, reducing convergence time and improving clustering accuracy.

19) What is agglomerative clustering?

Agglomerative clustering is a type of hierarchical clustering that starts with individual data points and iteratively merges them into larger clusters.

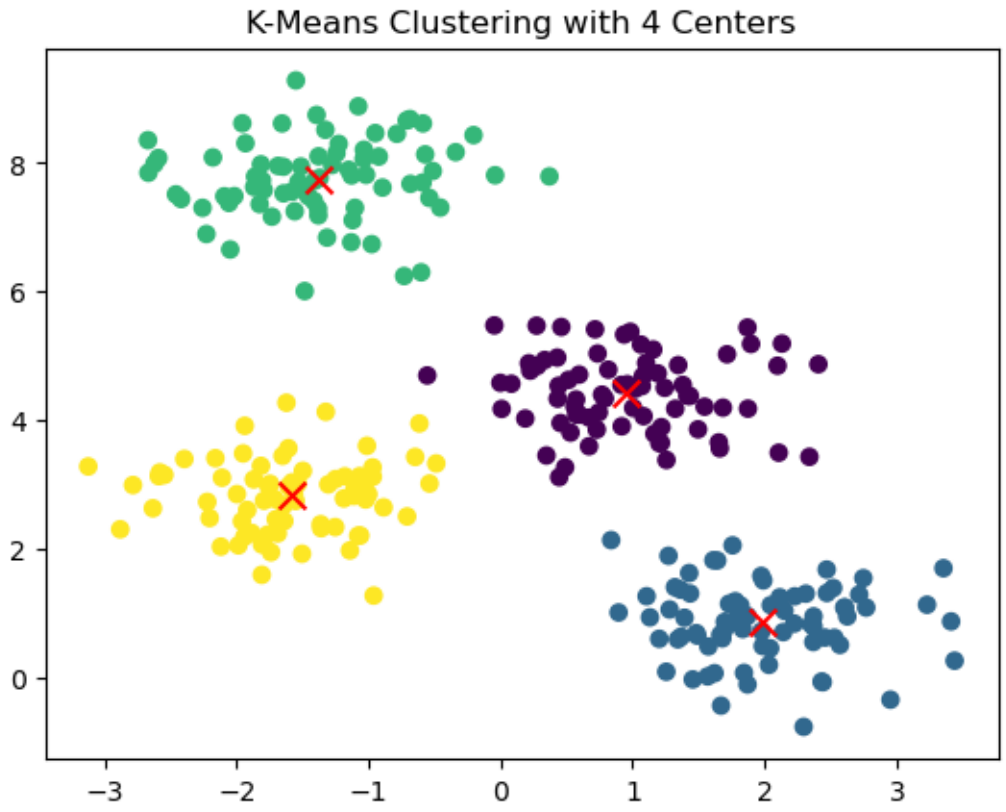
20) What makes Silhouette Score a better metric than just inertia for model evaluation?

Silhouette Score considers both intra-cluster distance and separation between clusters, making it a more comprehensive metric than inertia.

Assignment

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

[24]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

# Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=0)
pred_y = kmeans.fit_predict(X)

# Visualize
plt.scatter(X[:,0], X[:,1], c=pred_y, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            s=100, c='red', marker='x')
plt.title("K-Means Clustering with 4 Centers")
plt.show()
```



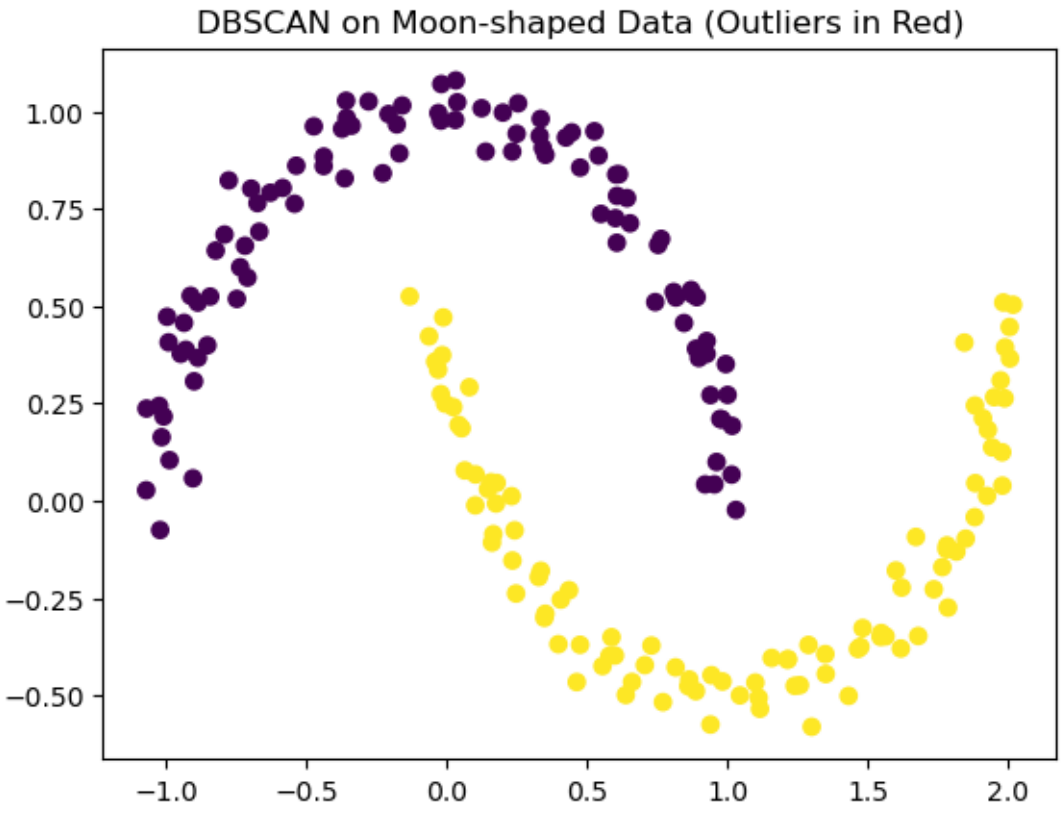
0.2 22) Load the Iris dataset and use Agglomerative Clustering to group the data into 3 clusters. Display the first 10 predicted labels

[26]:
```
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering

# Load Iris dataset
iris = load_iris()
X = iris.data

# Apply Agglomerative Clustering
agg = AgglomerativeClustering(n_clusters=3)
pred_y = agg.fit_predict(X)

# Display first 10 predicted labels
print("First 10 predicted labels:", pred_y[:10])
```

```
First 10 predicted labels: [1 1 1 1 1 1 1 1 1 1]
```

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

[28]:
```
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN

# Generate moon-shaped data
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

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

# Visualize with outliers highlighted
plt.scatter(X[:,0], X[:,1], c=pred_y, cmap='viridis')
outliers = X[pred_y == -1]
plt.scatter(outliers[:,0], outliers[:,1], c='red', marker='x', s=100)
plt.title("DBSCAN on Moon-shaped Data (Outliers in Red)")
plt.show()
```



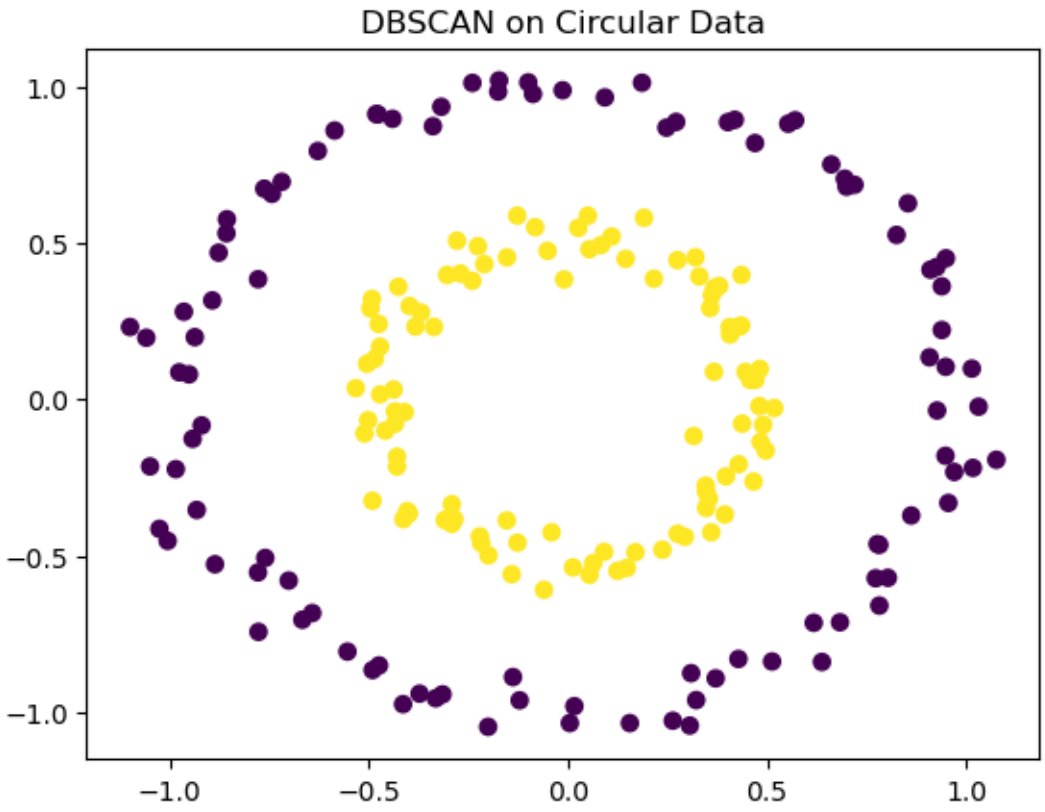
0.4 24) Load the Wine dataset and apply K-Means clustering after standardizing the features. Print the size of each cluster

[32]:
```
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
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
kmeans = KMeans(n_clusters=3, random_state=0)
pred_y = kmeans.fit_predict(X_scaled)

# Print cluster sizes
unique, counts = np.unique(pred_y, return_counts=True)
for cluster, count in zip(unique, counts):
    print(f"Cluster {cluster}: {count} samples")
```

```
Cluster 0: 65 samples
Cluster 1: 51 samples
Cluster 2: 62 samples
```

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

[34]:
```
from sklearn.datasets import make_circles

# Generate circular data
X, y = make_circles(n_samples=200, noise=0.05, factor=0.5, random_state=0)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.2, min_samples=5)
pred_y = dbscan.fit_predict(X)

# Visualize
plt.scatter(X[:,0], X[:,1], c=pred_y, cmap='viridis')
plt.title("DBSCAN on Circular Data")
plt.show()
```



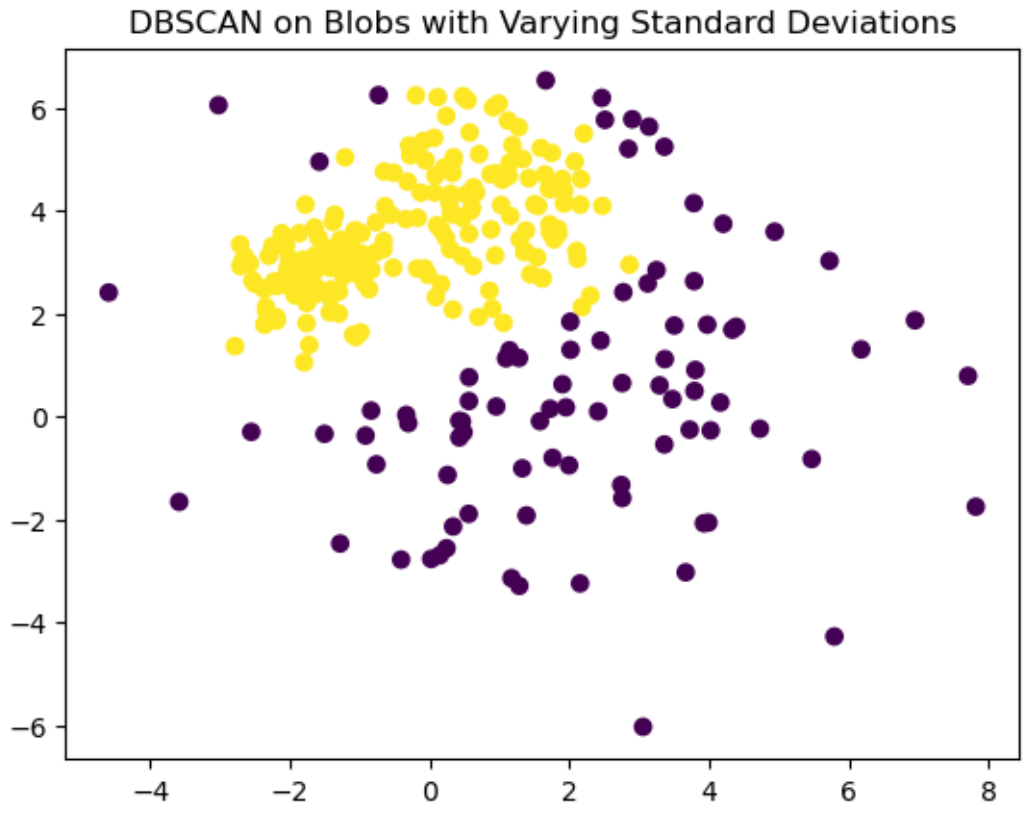
0.6 26) Load the Breast Cancer dataset, apply MinMaxScaler, and use K-Means with 2 clusters. Output the cluster centroids

[36]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
# Load dataset
data = load_breast_cancer()
X = data.data

# Scale data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Apply K-Means
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X_scaled)

# Output cluster centroids
print("Cluster centroids:")
print(kmeans.cluster_centers_)
```

```
Cluster centroids:
[[0.25535358 0.28833455 0.24696416 0.14388369 0.35743076 0.18019471
  0.10344776 0.1306603  0.34011829 0.25591606 0.06427485 0.18843043
  0.05975663 0.02870108 0.18158628 0.13242941 0.05821528 0.18069336
  0.17221057 0.08403996 0.2052406  0.32069002 0.19242138 0.09943446
  0.3571115  0.14873935 0.13142287 0.26231363 0.22639412 0.15437354]
 [0.50483563 0.39560329 0.50578661 0.36376576 0.46988732 0.42226302
  0.41838662 0.46928035 0.45899738 0.29945886 0.19093085 0.19112073
  0.17903433 0.13086432 0.18017962 0.25890126 0.12542475 0.30942779
  0.190072   0.13266975 0.48047448 0.45107371 0.4655302  0.31460597
  0.49868817 0.36391461 0.39027292 0.65827197 0.33752296 0.26041387]]
```

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

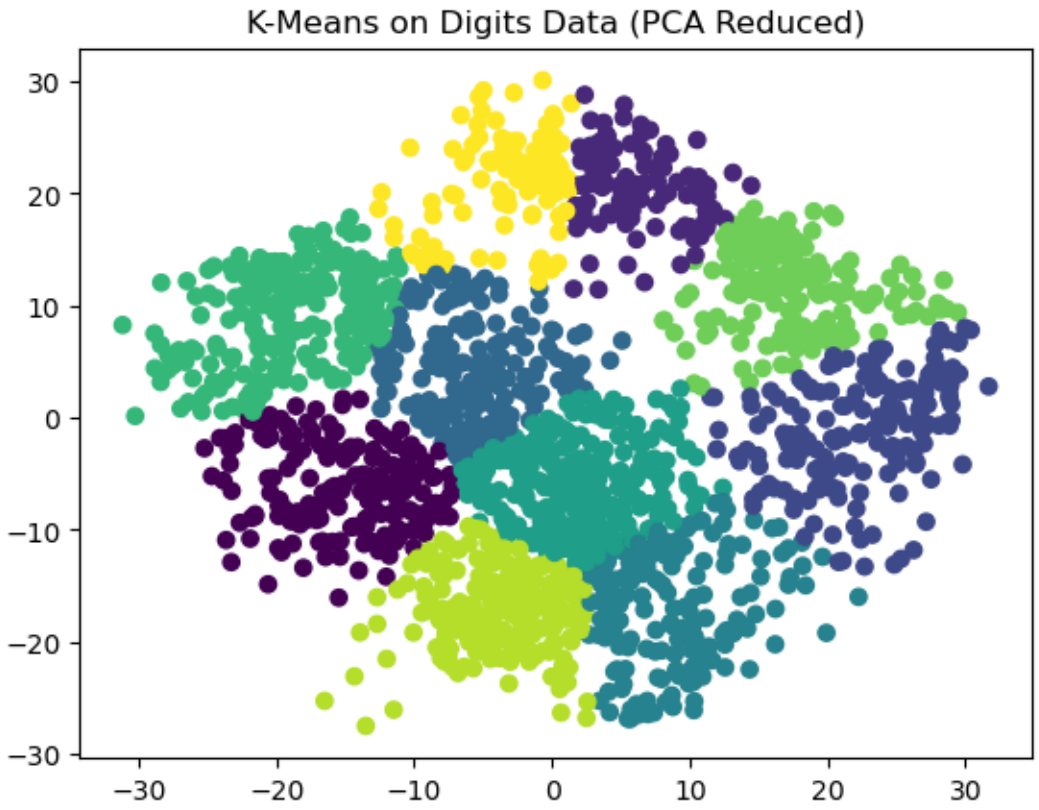
[38]:
```python
# Generate data with varying std
X, y = make_blobs(n_samples=300, centers=3, cluster_std=[1.0, 2.5, 0.5],␣
 ↪random_state=0)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.8, min_samples=10)
pred_y = dbscan.fit_predict(X)

# Visualize
plt.scatter(X[:,0], X[:,1], c=pred_y, cmap='viridis')
plt.title("DBSCAN on Blobs with Varying Standard Deviations")
plt.show()
```



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

[40]:
```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA

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

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

# Apply K-Means
kmeans = KMeans(n_clusters=10, random_state=0)
pred_y = kmeans.fit_predict(X_pca)

# Visualize
plt.scatter(X_pca[:,0], X_pca[:,1], c=pred_y, cmap='viridis')
plt.title("K-Means on Digits Data (PCA Reduced)")
```

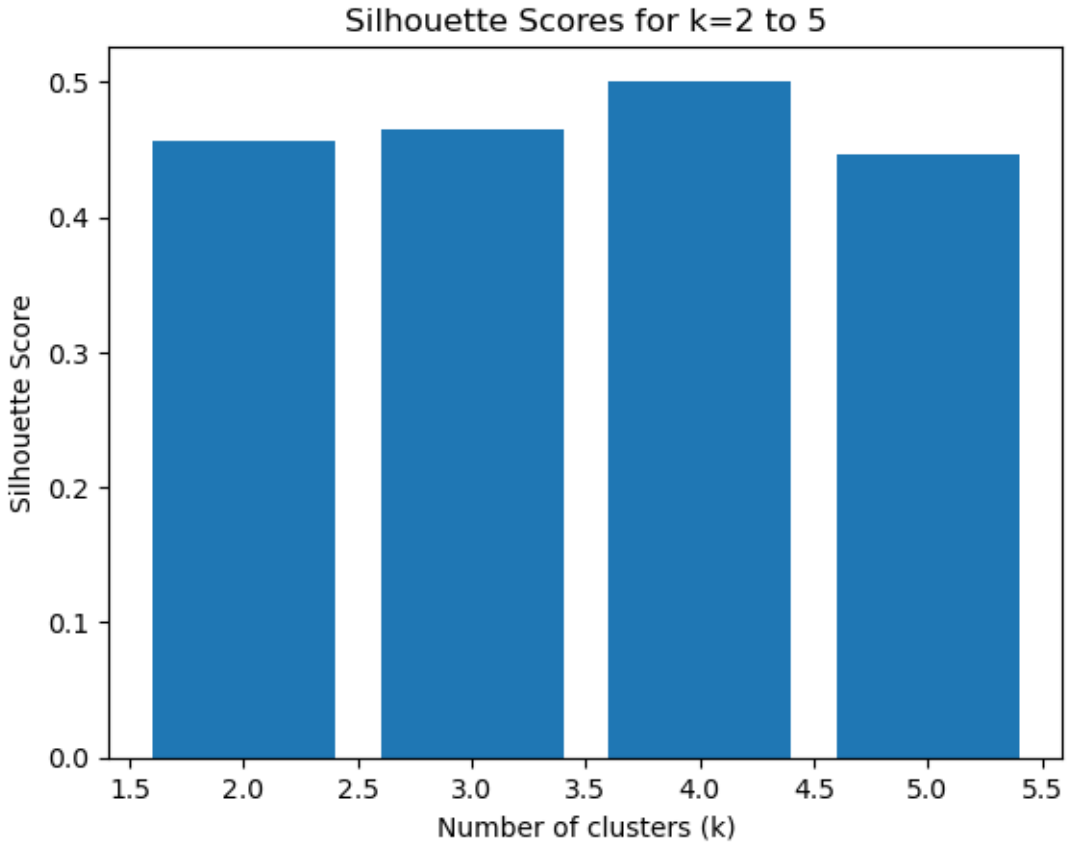
```
plt.show()
```



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

[42]:
```
from sklearn.metrics import silhouette_score
import numpy as np

# Generate synthetic data
X, y = make_blobs(n_samples=500, centers=4, random_state=0)

# Evaluate silhouette scores
silhouette_scores = []
for k in range(2, 6):
    kmeans = KMeans(n_clusters=k, random_state=0)
    pred_y = kmeans.fit_predict(X)
    score = silhouette_score(X, pred_y)
    silhouette_scores.append(score)

# Plot as bar chart
```

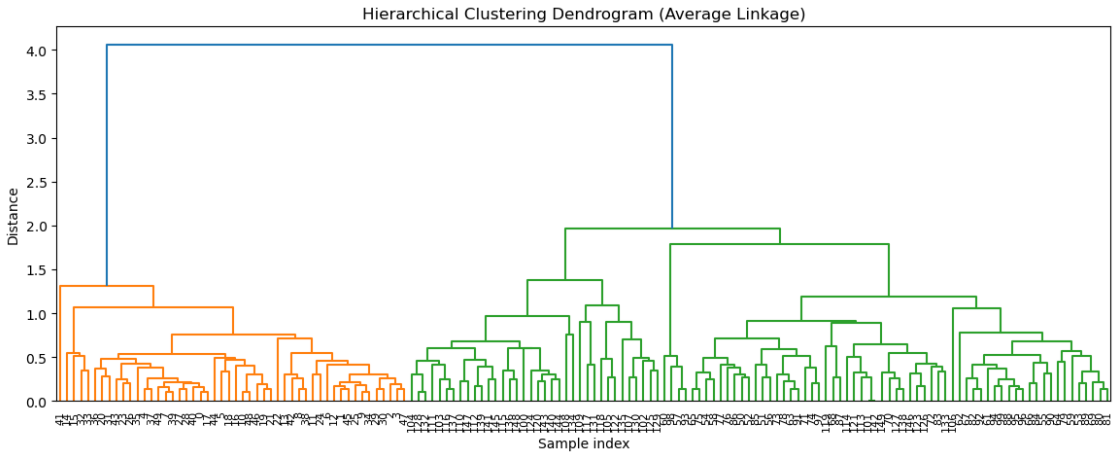
```
plt.bar(range(2,6), silhouette_scores)
plt.xlabel('Number of clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Scores for k=2 to 5')
plt.show()
```



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

[52]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
X = iris.data

# Perform hierarchical clustering
Z = linkage(X, 'average')
```

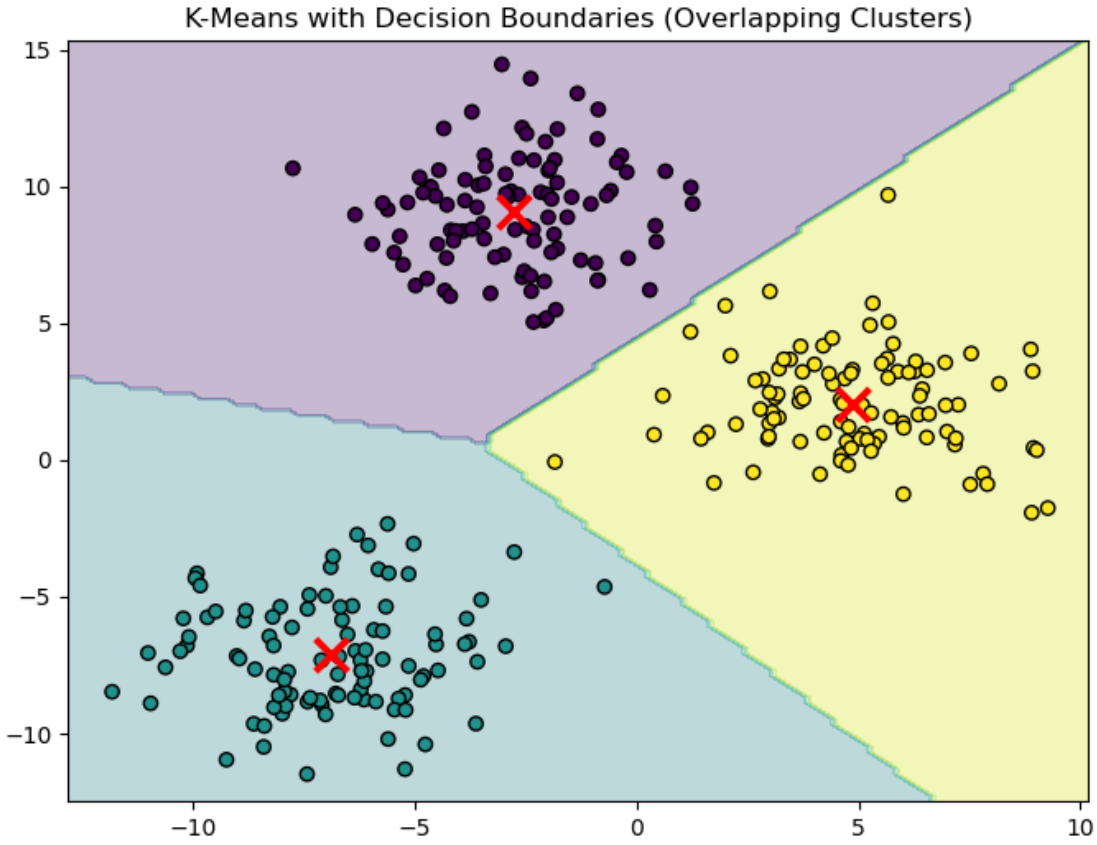
```
# Plot dendrogram
plt.figure(figsize=(14, 5))
plt.title('Hierarchical Clustering Dendrogram (Average Linkage)')
plt.xlabel('Sample index')
plt.ylabel('Distance')
dendrogram(Z, leaf_rotation=90., leaf_font_size=8.)
plt.show()
```



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

[74]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
# Generate overlapping clusters
X, y = make_blobs(n_samples=300, centers=3, cluster_std=2.0, random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
y_pred = kmeans.predict(X)

# Create mesh grid for decision boundaries
h = 0.2
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.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot
plt.figure(figsize=(8, 6))
```

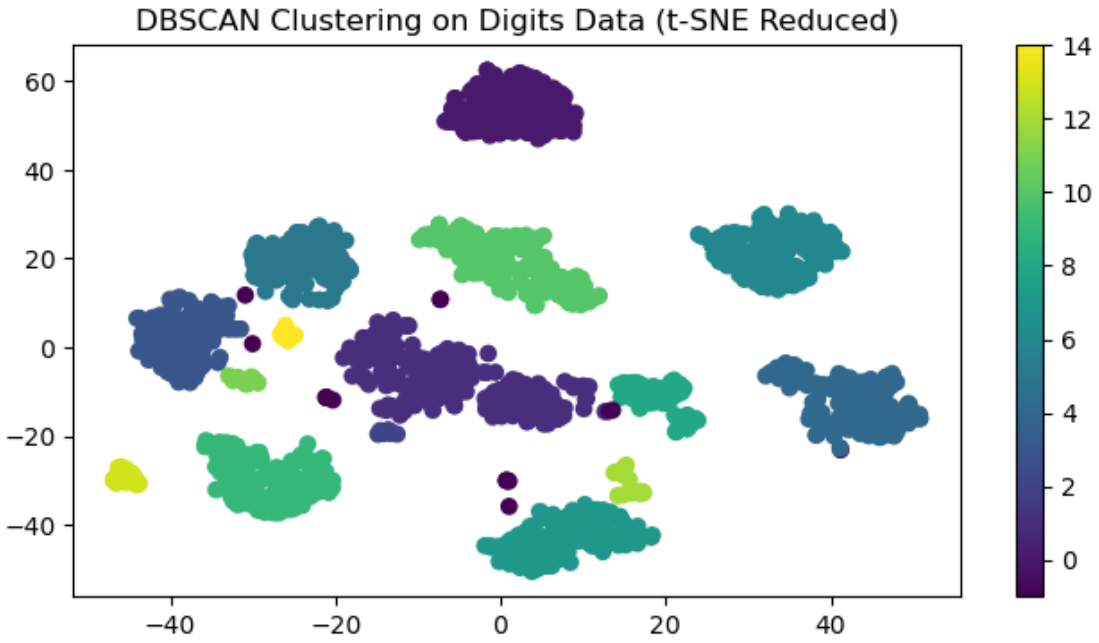
```
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', edgecolor='k')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            c='red', marker='x', s=200, linewidth=3)
plt.title("K-Means with Decision Boundaries (Overlapping Clusters)")
plt.show()
```



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

[82]:
```
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN
# Load digits
digits = load_digits()
X = digits.data

# Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)
```

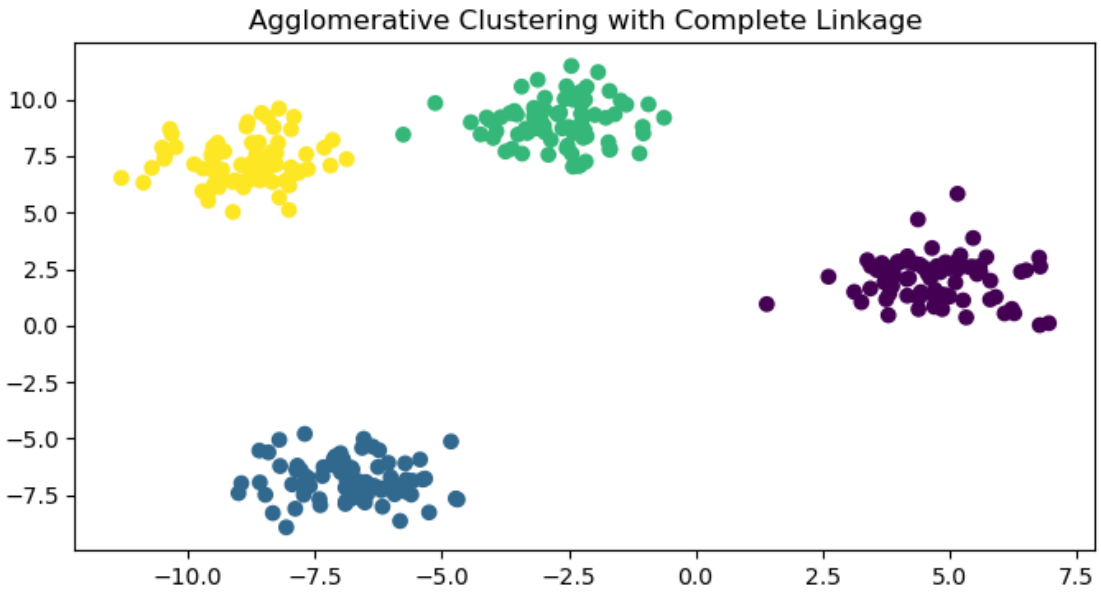
```
# Apply DBSCAN
dbscan = DBSCAN(eps=3, min_samples=5)
y_pred = dbscan.fit_predict(X_tsne)

# Visualize
plt.figure(figsize=(8, 4))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_pred, cmap='viridis')
plt.title("DBSCAN Clustering on Digits Data (t-SNE Reduced)")
plt.colorbar()
plt.show()
```



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

[83]:
```
from sklearn.cluster import AgglomerativeClustering
# Generate data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Apply Agglomerative Clustering
agg = AgglomerativeClustering(n_clusters=4, linkage='complete')
y_pred = agg.fit_predict(X)

# Plot
plt.figure(figsize=(8, 4))
```

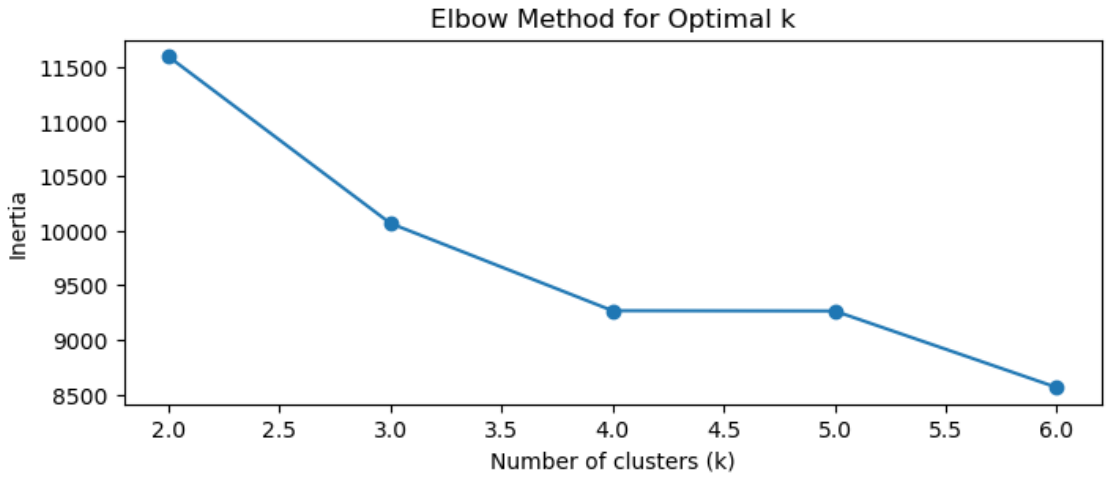
```
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
plt.title("Agglomerative Clustering with Complete Linkage")
plt.show()
```



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

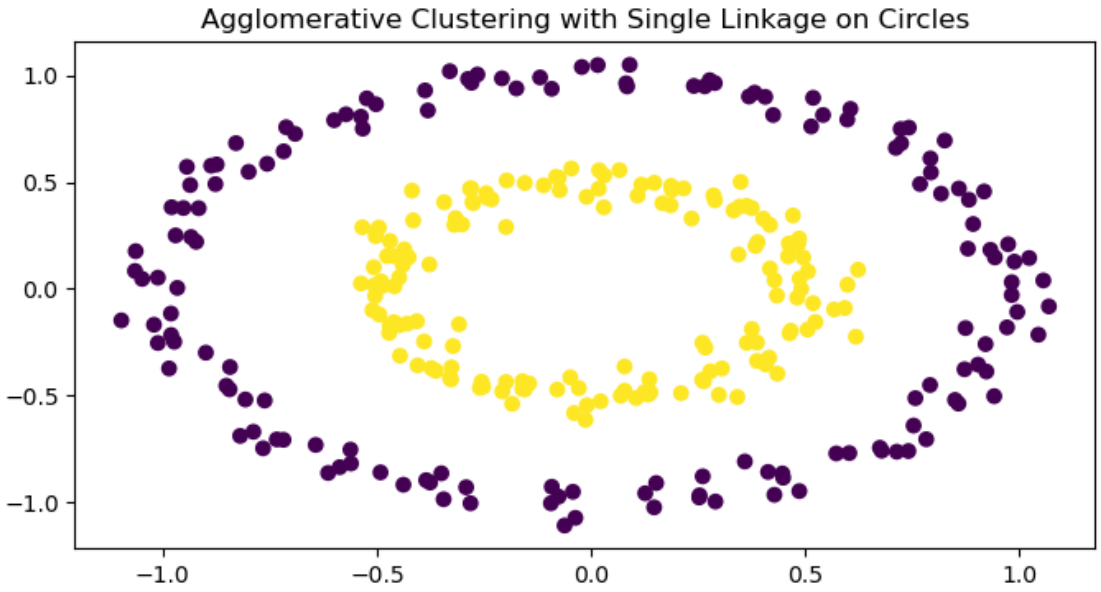
[88]:
```
from sklearn.datasets import load_breast_cancer
# Load data
data = load_breast_cancer()
X = data.data

# Scale data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Calculate inertia for k=2 to 6
inertias = []
for k in range(2, 7):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Plot
plt.figure(figsize=(8, 3))
plt.plot(range(2, 7), inertias, marker='o')
plt.xlabel('Number of clusters (k)')
```

```
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



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

[92]:
```
from sklearn.datasets import make_circles

# Generate concentric circles
X, y = make_circles(n_samples=300, noise=0.05, factor=0.5, random_state=42)

# Apply Agglomerative Clustering with single linkage
agg = AgglomerativeClustering(n_clusters=2, linkage='single')
y_pred = agg.fit_predict(X)

# Plot
plt.figure(figsize=(8, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
plt.title("Agglomerative Clustering with Single Linkage on Circles")
plt.show()
```



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

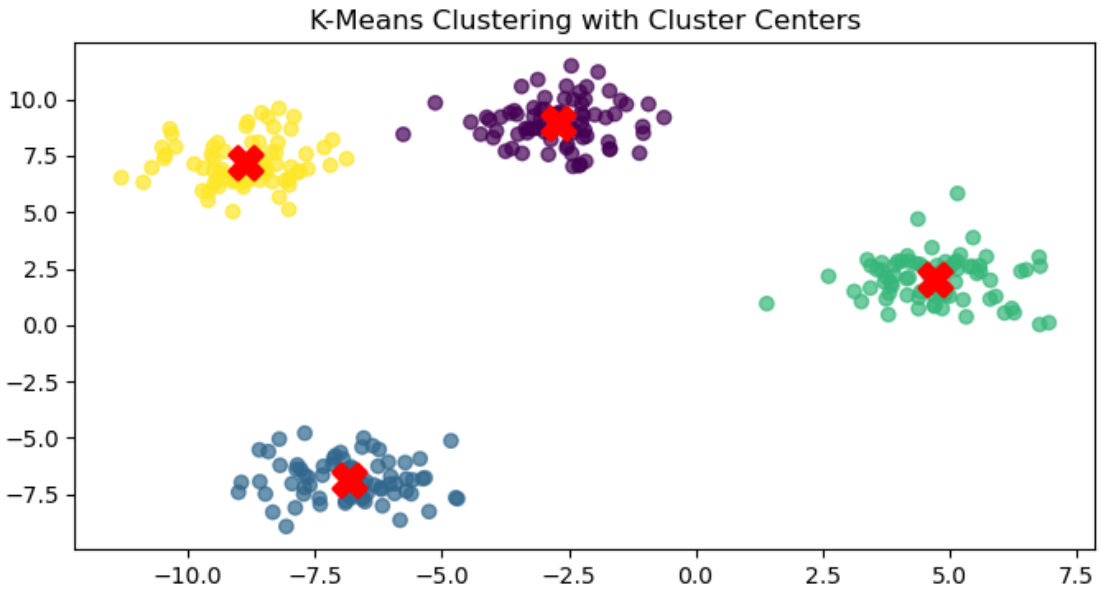
[94]:
```
from sklearn.datasets import load_wine

# Load and scale data
wine = load_wine()
X = wine.data
X_scaled = StandardScaler().fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=1.5, min_samples=5)
y_pred = dbscan.fit_predict(X_scaled)

# Count clusters (excluding noise)
n_clusters = len(set(y_pred)) - (1 if -1 in y_pred else 0)
print(f"Number of clusters found (excluding noise): {n_clusters}")
```

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

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

[98]:
```
# Generate data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Apply K-Means
```

```
kmeans = KMeans(n_clusters=4, random_state=42)
y_pred = kmeans.fit_predict(X)

# Plot
plt.figure(figsize=(8, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            c='red', marker='X', s=200, linewidth=2)
plt.title("K-Means Clustering with Cluster Centers")
plt.show()
```



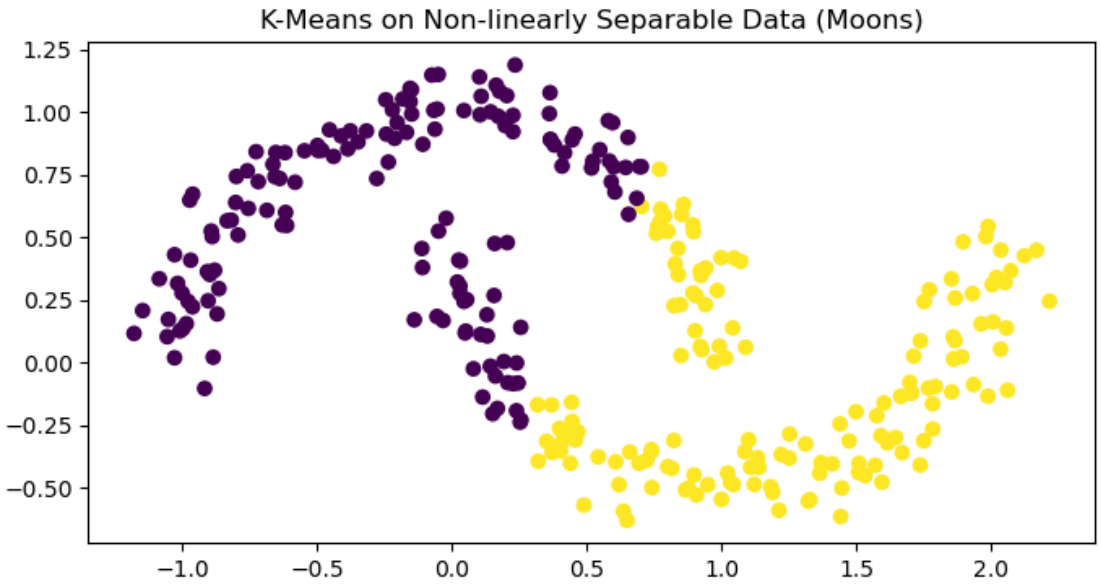
0.18 38) Load the Iris dataset, cluster with DBSCAN, and print how many samples were identified as noise

[100]:
```
from sklearn.datasets import load_iris
# Load iris data
iris = load_iris()
X = iris.data

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
y_pred = dbscan.fit_predict(X)

# Count noise points
n_noise = list(y_pred).count(-1)
print(f"Number of samples identified as noise: {n_noise}")
```

```
Number of samples identified as noise: 17
```

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

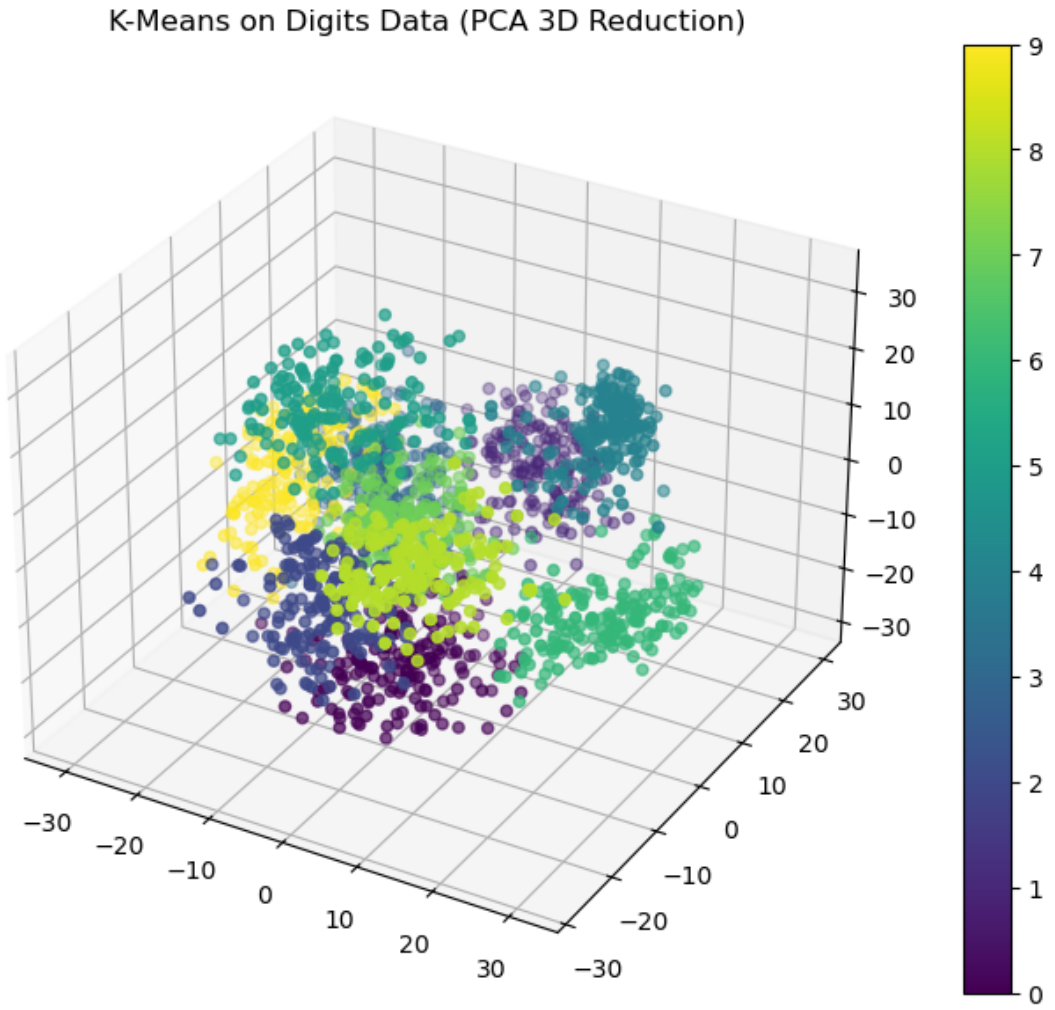
[104]:
```
# Generate moons data
X, y = make_moons(n_samples=300, noise=0.08, random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=2, random_state=42)
y_pred = kmeans.fit_predict(X)

# Plot
plt.figure(figsize=(8, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
plt.title("K-Means on Non-linearly Separable Data (Moons)")
plt.show()
```



0.20 40) Load the Digits dataset, apply PCA to reduce to 3 components, then use KMeans and visualize with a 3D scatter plot.

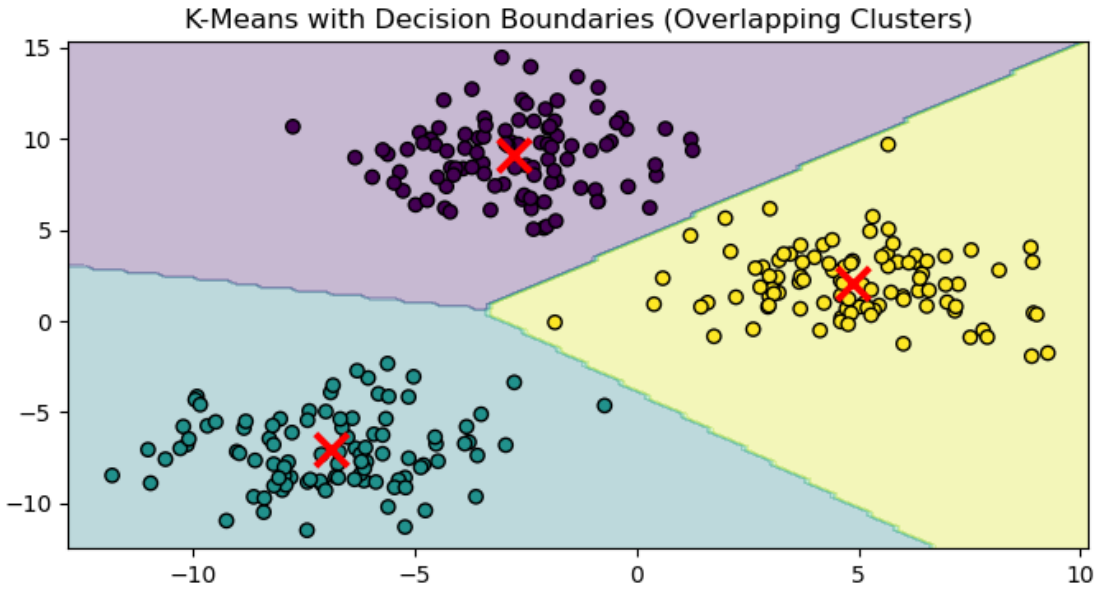
[120]:
```
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
# Load digits
digits = load_digits()
X = digits.data
```

```python
# Reduce to 3D with PCA
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)

# Apply K-Means
kmeans = KMeans(n_clusters=10, random_state=42)
y_pred = kmeans.fit_predict(X_pca)

# 3D plot
fig = plt.figure(figsize=(9, 7))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=y_pred,␣
 ↪cmap='viridis')
plt.title("K-Means on Digits Data (PCA 3D Reduction)")
plt.colorbar(scatter)
plt.show()
```



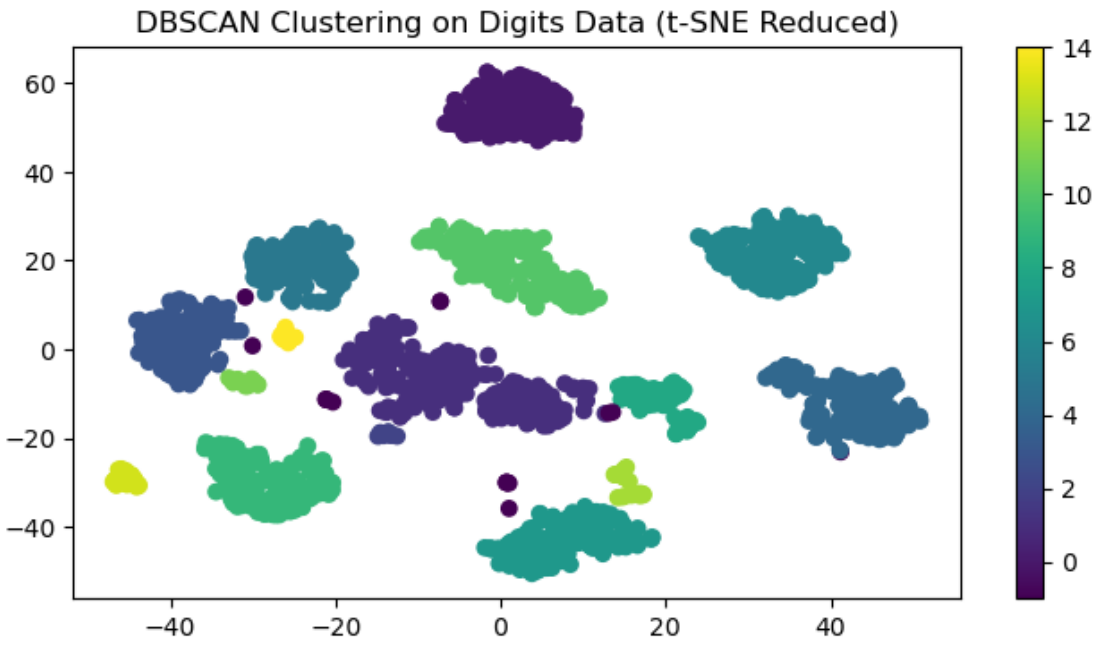
0.21 41)Generate synthetic blobs with 5 centers and apply KMeans. Then use silhouette_score to evaluate the clustering

[139]:
```
from sklearn.datasets import make_blobs
# Generate overlapping clusters
X, y = make_blobs(n_samples=300, centers=3, cluster_std=2.0, random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
y_pred = kmeans.predict(X)

# Create mesh grid for decision boundaries
h = 0.2
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.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot
plt.figure(figsize=(8, 4))
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', edgecolor='k')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            c='red', marker='x', s=200, linewidth=3)
plt.title("K-Means with Decision Boundaries (Overlapping Clusters)")
plt.show()
```



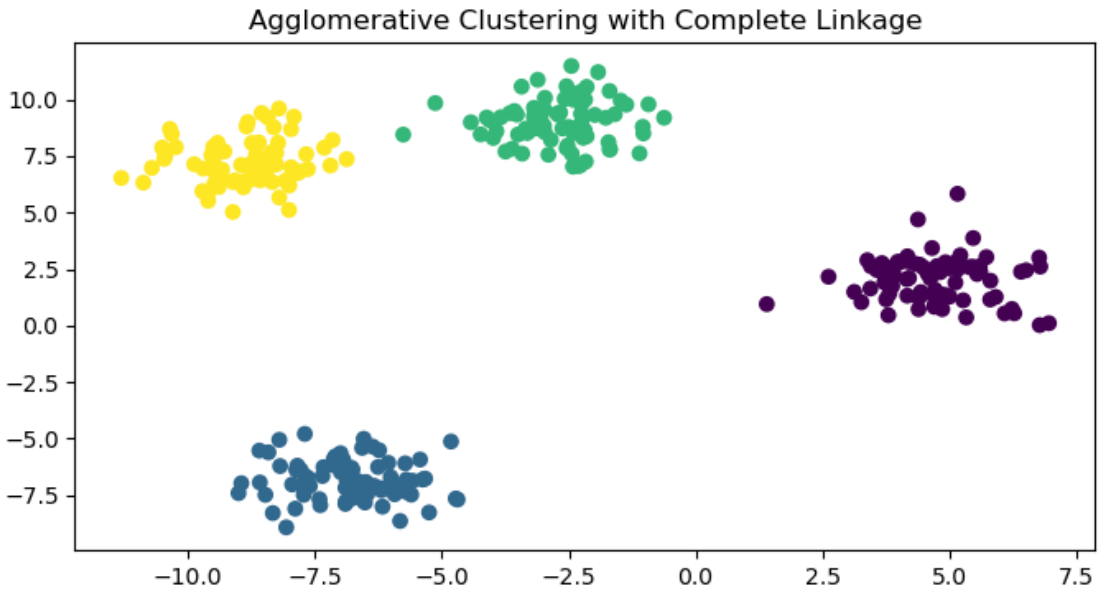
0.22 42) Load the Breast Cancer dataset, reduce dimensionality using PCA, and apply Agglomerative Clustering. Visualize in 2D

[142]:
```
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN
# Load digits
digits = load_digits()
X = digits.data

# Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=3, min_samples=5)
y_pred = dbscan.fit_predict(X_tsne)

# Visualize
plt.figure(figsize=(8, 4))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_pred, cmap='viridis')
plt.title("DBSCAN Clustering on Digits Data (t-SNE Reduced)")
plt.colorbar()
plt.show()
```



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

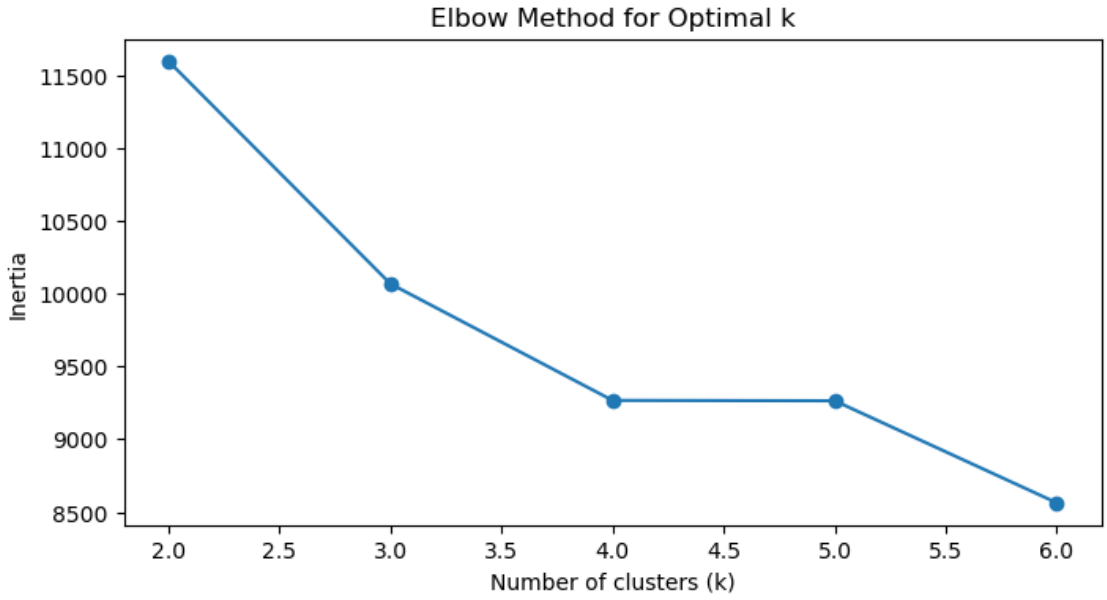
[146]:
```
from sklearn.cluster import AgglomerativeClustering
# Generate data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Apply Agglomerative Clustering
agg = AgglomerativeClustering(n_clusters=4, linkage='complete')
y_pred = agg.fit_predict(X)

# Plot
plt.figure(figsize=(8, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
plt.title("Agglomerative Clustering with Complete Linkage")
plt.show()
```



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

[150]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
# Load data
data = load_breast_cancer()
X = data.data
```

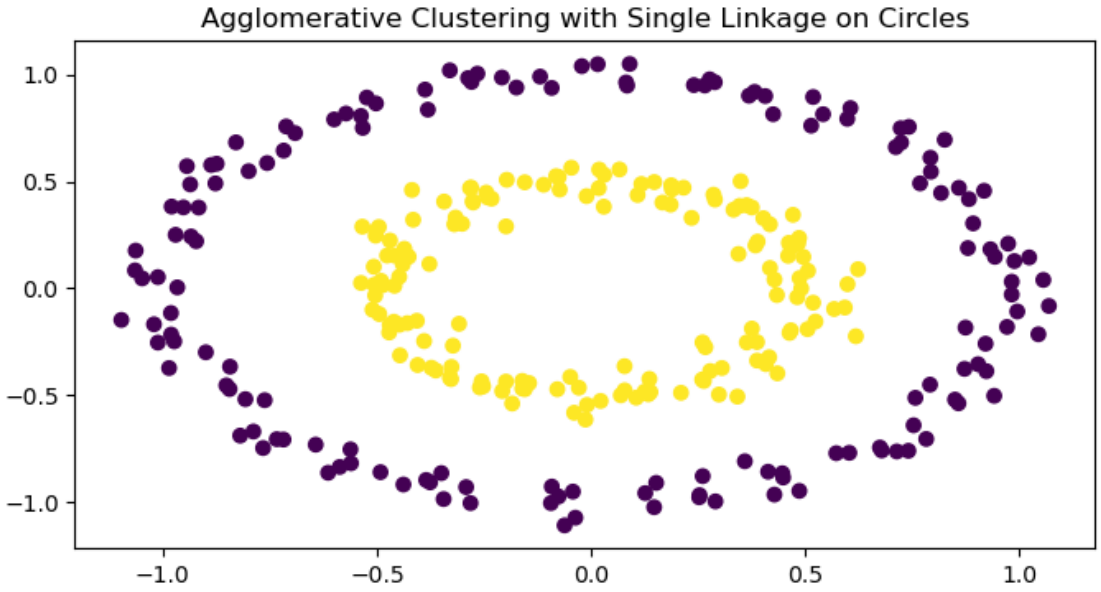
```
# Scale data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Calculate inertia for k=2 to 6
inertias = []
for k in range(2, 7):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Plot
plt.figure(figsize=(8, 4))
plt.plot(range(2, 7), inertias, marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



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

[152]:
```
from sklearn.datasets import make_circles

# Generate concentric circles
```

```
X, y = make_circles(n_samples=300, noise=0.05, factor=0.5, random_state=42)

# Apply Agglomerative Clustering with single linkage
agg = AgglomerativeClustering(n_clusters=2, linkage='single')
y_pred = agg.fit_predict(X)

# Plot
plt.figure(figsize=(8, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
plt.title("Agglomerative Clustering with Single Linkage on Circles")
plt.show()
```



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

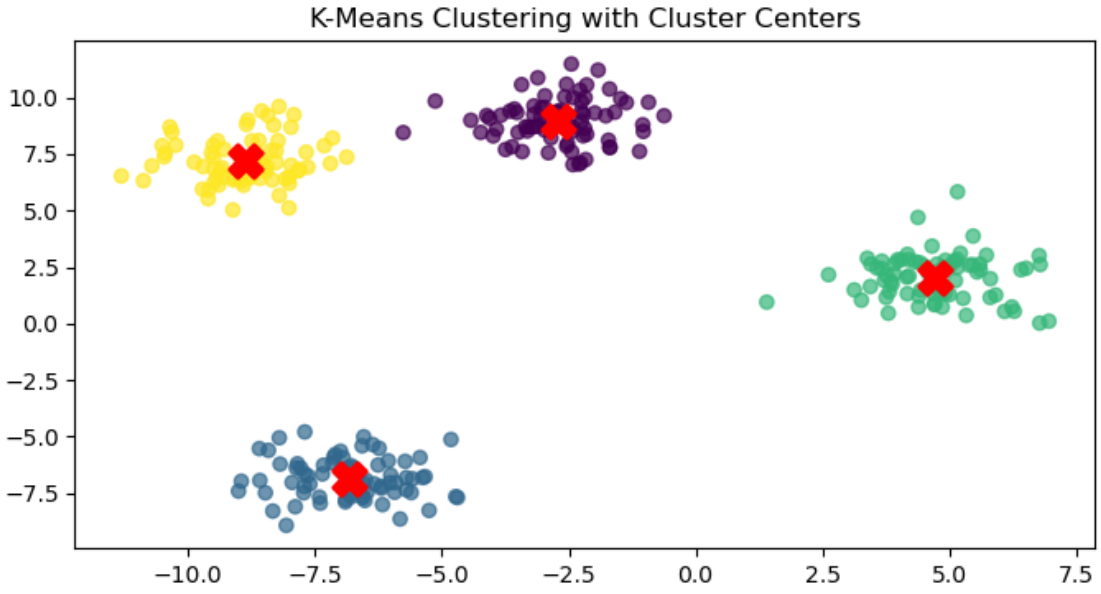
[154]:
```
from sklearn.datasets import load_wine
# Load and scale data
wine = load_wine()
X = wine.data
X_scaled = StandardScaler().fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=1.5, min_samples=5)
y_pred = dbscan.fit_predict(X_scaled)

# Count clusters (excluding noise)
n_clusters = len(set(y_pred)) - (1 if -1 in y_pred else 0)
```

```
print(f"Number of clusters found (excluding noise): {n_clusters}")
```

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

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

[156]:
```
# Generate data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=4, random_state=42)
y_pred = kmeans.fit_predict(X)

# Plot
plt.figure(figsize=(8, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            c='red', marker='X', s=200, linewidth=2)
plt.title("K-Means Clustering with Cluster Centers")
plt.show()
```



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

[158]:
```
from sklearn.datasets import load_iris

# Load iris data
iris = load_iris()
X = iris.data

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
y_pred = dbscan.fit_predict(X)

# Count noise points
n_noise = list(y_pred).count(-1)
print(f"Number of samples identified as noise: {n_noise}")
```

```
Number of samples identified as noise: 17
```