Theoretical Questions

In [ ]:
```
#What is unsupervised learning in the context of machine learning
'''
Unsupervised learning is a type of machine learning where the algorithm learns from
predefined categories or outcomes. Unlike supervised learning, where the model is t
prices or classifying emails), unsupervised learning finds **patterns, structures,

Key Goals of Unsupervised Learning:
1.Clustering – Grouping similar data points together.
   Example: Customer segmentation (grouping customers by purchasing behavior).

2.Dimensionality Reduction – Simplifying data by reducing the number of variables w
   Example: Principal Component Analysis (PCA) to visualize high-dimensional data i

3.Anomaly Detection – Identifying unusual data points.
   Example: Fraud detection in transactions.

4.Association Rule Learning – Discovering relationships between variables.
   Example: Market Basket Analysis (people who buy X often buy Y).

 Common Algorithms in Unsupervised Learning:
1.K-Means Clustering
2.Hierarchical Clustering
3.DBSCAN
4.Principal Component Analysis (PCA)
5.t-SNE (for visualization)
'''
# How does K-Means clustering algorithm work
'''
How It Works – Step by Step:
1.Choose the number of clusters (K)
You decide how many clusters (groups) you want the algorithm to find.

2.Initialize Centroids Randomly
Randomly pick K points in the data space as the initial "centroids" (centers of clu

3.Assign Each Data Point to the Nearest Centroid
Each data point is assigned to the cluster with the nearest centroid (based on dist

4.Recalculate Centroids
For each cluster, calculate the mean of all the points in that cluster. This mean b

5.Repeat Steps 3 and 4
Continue reassigning data points and updating centroids until:
The centroids no longer change significantly (convergence), or
A maximum number of iterations is reached
'''
#Explain the concept of a dendrogram in hierarchical clustering
'''
A dendrogram is a tree-like diagram that shows how data points or clusters are merg
helps you visualize the structure of clusters and decide how many clusters to choos

Hierarchical Clustering
```

```
There are two types:
1.Agglomerative (bottom-up):
Start with each point as its own cluster and merge them step-by-step.
2.Divisive (top-down):
Start with one big cluster and split it step-by-step.

In both cases, a dendrogram is used to track these merges or splits.

How to Read a Dendrogram
Leaves (bottom): Each individual data point.
Branches (lines connecting points): Clusters being merged.
Height (vertical axis): Distance (or dissimilarity) between clusters when they're m

Example Analogy:
Imagine grouping animals:
First, all animals are individual.
Then you group similar ones: cats with tigers, dogs with wolves.
Then bigger clusters: cats+dogs = mammals.
The dendrogram would show these merges as branches from leaves to the trunk.
'''
# What is the main difference between K-Means and Hierarchical Clustering?
'''
 K-Means Clustering vs Hierarchical Clustering
1.Number of Clusters
K-Means: You must specify K (number of clusters) in advance.
Hierarchical: No need to specify K at the beginning; you can choose later by cuttin

2.Approach
K-Means: Partitioning method – divides data into non-overlapping groups.
Hierarchical: Builds a tree (dendrogram) that shows how clusters are merged or spli

3.Structure
K-Means: Creates flat clusters.
Hierarchical: Creates nested clusters (one inside another).

4.Speed and Scalability
K-Means: Faster, good for large datasets.
Hierarchical: Slower, not ideal for large datasets.

5.Cluster Shape Assumption
K-Means: Works best with spherical or equally sized clusters.
Hierarchical: Can handle complex shapes and structures.

6.Reproducibility
K-Means: Depends on random initialization, may give different results each time.
Hierarchical: Deterministic, gives the same result every time.

7.Visualization
K-Means: No built-in visualization structure.
Hierarchical: Uses a dendrogram to visualize the clustering process.

8.Flexibility
K-Means: Less flexible if the number of clusters is unknown.
Hierarchical: More flexible for exploring data and deciding optimal clusters.
'''
#What are the advantages of DBSCAN over K-Means
```

```
'''
Advantages of DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

1.No Need to Specify Number of Clusters (K)
DBSCAN: You don't need to specify the number of clusters in advance.
K-Means: Requires you to set K manually.

2.Can Find Arbitrary-Shaped Clusters
DBSCAN can detect clusters of any shape, not just spherical ones.
K-Means works best with round/spherical clusters.

3.Handles Noise and Outliers
DBSCAN treats outliers as noise, which improves cluster quality.
K-Means includes all points in clusters, even noisy ones.

4.More Robust to Outliers
DBSCAN doesn't force every point into a cluster.
K-Means can be easily influenced by outliers, shifting centroids.

5.No Need for Cluster Center Initialization
K-Means is sensitive to initial centroid positions and can give different results.
DBSCAN does not require centroid initialization.

6.Can Work with Non-Linearly Separable Data
DBSCAN can cluster data where clusters are not linearly separable.
K-Means fails in such cases unless clusters are well-separated.
'''
#When would you use Silhouette Score in clustering?
'''
The Silhouette Score is used in clustering to evaluate how well your data has been
similar an object is to its own cluster compared to other clusters.

1.Choosing the optimal number of clusters (K)
It's especially helpful in K-means, K-medoids, or other clustering methods where yo
You compute the Silhouette Score for different values of k and choose the one that

2. Measuring cluster quality
After clustering, it tells you how compact and well-separated the clusters are.
The score ranges from -1 to 1:
+1: Sample is well matched to its own cluster and poorly matched to neighboring clu
0: Sample is on or very close to the decision boundary between two clusters.
-1: Sample may have been assigned to the wrong cluster.

3. Comparing different clustering algorithms
You can compare clustering performance across methods like K-Means, Agglomerative C
'''
#What are the limitations of Hierarchical Clustering?
'''
1. Not scalable for large datasets
Time complexity is O(n²) or worse, depending on the implementation.
Doesn't scale well when the number of data points is very high (e.g., thousands or

2. No clear objective function
Unlike K-means (which minimizes intra-cluster variance), hierarchical clustering do
This makes it harder to evaluate the "best" result objectively.
```

```
3. Sensitive to noise and outliers
A single noisy data point or outlier can drastically affect the tree structure (den

4. Once merged or split, can’t be undone
It’s a greedy algorithm: once two clusters are merged (agglomerative) or a cluster
permanent, even if it's suboptimal.

5. Choosing the right number of clusters is tricky
There's no built-in mechanism like the "elbow method" or Silhouette Score baked int
You usually have to manually cut the dendrogram, which can be subjective.

6. Distance metric sensitivity
The results can vary a lot depending on the distance metric (Euclidean, Manhattan,
complete, average, ward).

7. Hard to interpret for high-dimensional data
In high-dimensional spaces, distance metrics become less meaningful (curse of dimen
to interpret.
'''
# Why is feature scaling important in clustering algorithms like K-Means?
'''
1. K-Means uses distance to form clusters
K-Means assigns points to the nearest cluster based on distance to the cluster cent
Features with larger numerical ranges can dominate the distance metric, making othe

2. Without scaling, features with bigger units skew results
Imagine a dataset with:
Age in range 0–100
Income in range 0–100,000
Without scaling:
The clustering will mostly be influenced by Income, even though Age might be just a

3. Scaling ensures all features contribute equally
After scaling (e.g., using StandardScaler or MinMaxScaler), all features are on a s
StandardScaler → mean = 0, std = 1
MinMaxScaler → range = [0, 1]
This lets K-Means treat each feature fairly in the distance calculation.

4. Better convergence and performance
K-Means typically converges faster and forms more meaningful clusters when features
'''
#How does DBSCAN identify noise points?
'''
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) identifies noi
density ,specifically, the idea that clusters are dense regions of points separated

Key Concepts in DBSCAN:
ε (epsilon): Radius of neighborhood around a point.
minPts: Minimum number of points required to form a dense region

DBSCAN Classifies Points Into 3 Types:
1.Core Point:
Has at least minPts points (including itself) within its ε-radius.
It’s in a dense region and starts forming a cluster.

2.Border Point:
```

```
Has fewer than minPts points within ε, but is within ε of a core point.
Belongs to a cluster, but doesn’t start one.

3.Noise Point (Outlier):
Not a core point AND not within ε of any core point.
Doesn’t belong to any cluster

So, how DBSCAN identifies noise:
It scans each point in the dataset.
If a point cannot be reached from any core point (directly or indirectly), it's lab
'''
#Define inertia in the context of K-Means?
'''
Definition of Inertia:
Inertia is the sum of squared distances between each data point and the centroid of
Formula:  k
          ∑ ∑||x - μi ||2
         i=1 xeci
Where:
k is number of clusters
Ci is set of points in cluster i
μi is centroid of cluster i
||x - μi||2 is squared Euclidean distance between point x and its cluster centroid

Use of Inertia:
Lower inertia means tighter, more compact clusters.
You can use it to:
Evaluate clustering performance.
Apply the elbow method to help choose the optimal number of clusters

Limitations:
Inertia always decreases as K increases, so it doesn't indicate the "best" K on its
It’s sensitive to feature scale, so always scale features before using K-Means
'''
#What is the elbow method in K-Means clustering?
'''
The Elbow Method is a popular technique used in K-Means clustering to help you choo

Why do we need it?
K-Means requires you to specify the number of clusters K in advance.
But how do you know what K is best? That’s where the elbow method helps

How it works:
1.Run K-Means clustering for a range of values of K (e.g., from 1 to 10).
2.For each K, calculate the inertia (within-cluster sum of squares).
3.Plot K vs Inertia.
4.Look for a point on the curve where the rate of decrease sharply slows down — thi

Why it's called the "elbow":
1.On the plot, inertia decreases rapidly at first (more clusters = better fit),
2.But after a certain point, the improvement becomes marginal.
3.The “elbow” is where the curve starts to flatten — that’s your ideal K.
'''
#Describe the concept of "density" in DBSCAN?
'''
"density" in DBSCAN
```

```
In DBSCAN, density refers to how closely packed data points are in a region. Instea
(like K-Means), DBSCAN groups together areas of high point density and separates th
boundaries).

Key Components of Density:
1.ε (epsilon):
The radius around a data point.
Think of it as a neighborhood zone.
2.minPts:
The minimum number of points required within an ε-radius to consider that area "den

 How DBSCAN uses density:
Starts with a random point.
If it’s a core point, it forms a cluster by pulling in all directly density-reachab
Expands the cluster by checking neighbors of neighbors.
Continues until no more points can be added.
Points not reachable from any core point are labeled as noise (outliers)

Benefits of using density:
Can find clusters of arbitrary shapes (not just spherical).
Automatically identifies noise points.
No need to specify the number of clusters beforehand
'''
#Can hierarchical clustering be used on categorical data?
'''
Yes, Hierarchical Clustering can be used on categorical data, but with some importa
By default, many implementations (like Scikit-learn's AgglomerativeClustering) assu
which doesn't work well with categorical variables. But with the right distance mea
data.

How to use Hierarchical Clustering on categorical data:
1. Use an appropriate distance metric
Categorical data needs special distance measures, like:
Hamming distance (for binary/categorical data)
Jaccard distance (for binary attributes or sets)
Gower distance (for mixed data types: categorical + numerical)
2. Convert data if needed
If your data is in string labels, you may need to encode it first (e.g., one-hot en
One-hot preserves more information but increases dimensionality.
Label encoding may introduce false numerical relationships.
3. Use libraries that support custom distances
Scikit-learn doesn’t directly support non-Euclidean distances in AgglomerativeClust
Use scipy.cluster.hierarchy with a precomputed distance matrix.
Use the linkage function in SciPy with your custom distance matrix.
Try specialized libraries like kmodes (which includes k-prototypes for mixed data a
'''
#What does a negative Silhouette Score indicate?
'''
A negative Silhouette Score is a red flag in clustering — it indicates that a data

When the score is negative:
It means a > b → the point is closer to another cluster than to its own.
This suggests a bad clustering assignment for that point.

Implications of a Negative Score:
Poor separation between clusters.
```

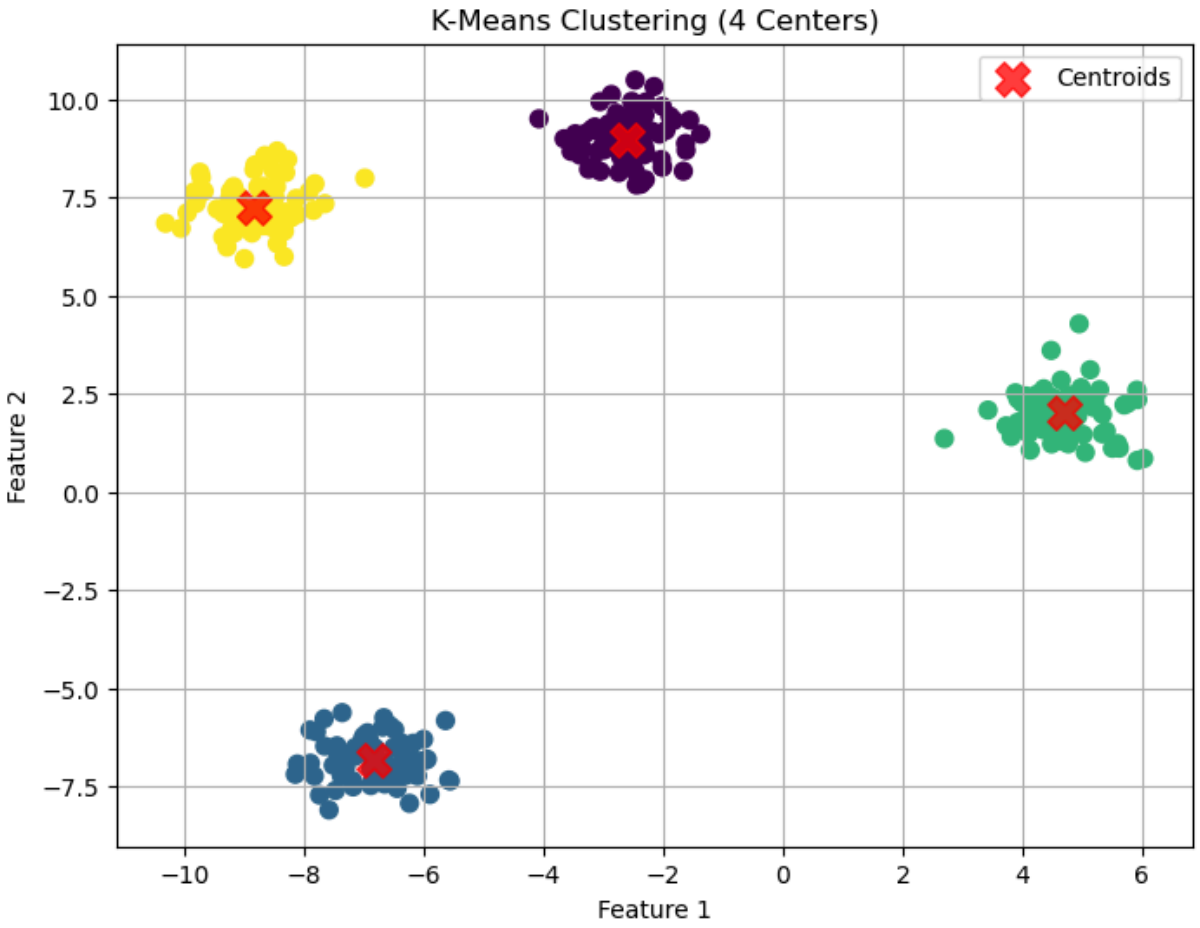
```
Overlapping clusters.
Possibly the wrong number of clusters (K) was chosen.
Could also indicate no clear structure in the data for clustering.

What to do if you get negative scores:
Try a different number of clusters (K).
Use a different clustering algorithm (e.g., DBSCAN or Agglomerative).
Perform feature scaling or dimensionality reduction (e.g., PCA).
Inspect the data for noise or overlapping classes
'''
#Explain the term "linkage criteria" in hierarchical clustering?
'''
The term "linkage criteria" in Hierarchical Clustering refers to the method used to
when merging them.
Since hierarchical clustering is built on repeatedly merging (or splitting) cluster
clusters to merge next — that rule is defined by the linkage criterion.

 What they influence:
The shape and size of clusters.
The structure of the dendrogram.
The sensitivity to noise or outliers.

Intuition behind each:
Single Linkage: Tends to form long, chain-like clusters (can cause chaining effect)
Complete Linkage: Produces compact, tightly bound clusters.
Average Linkage: A balance between single and complete — not too tight or too loose
Ward's: Best when you want clusters of similar size and variance (often used with E
'''
#Why might K-Means clustering perform poorly on data with varying cluster sizes or
'''
Why K-Means struggles with varying sizes or densities:
1. Assumes equal-sized clusters
K-Means tries to minimize the sum of squared distances to the nearest centroid, whi
Spherical
Equal in size
Equal in density
If one cluster is larger or denser, K-Means might:
Split it into multiple clusters
Or merge smaller clusters together incorrectly

2. Sensitive to density variations
K-Means doesn't consider density, only distance to the centroid.
So:
A dense cluster with many points close together might be underrepresented.
A sparse cluster might incorrectly "pull in" nearby points that don't really belong

3. Centroid-based approach is limiting
K-Means calculates the mean (centroid) of all points in a cluster.
If the actual data shape is non-spherical or uneven, the centroid may not represent

4. Fixed number of clusters
You have to specify K manually, which is hard if clusters are irregular or nested w
'''
#What are the core parameters in DBSCAN, and how do they influence clustering?
'''
1. eps (epsilon)
```

```
It defines the radius around a data point — the neighborhood size.
Think of it as a bubble drawn around a point.
If other points fall within this bubble, they are considered neighbors.
Influence:
Too small eps → many small clusters or all points labeled as noise.
Too large eps → clusters may merge or everything becomes one cluster.
Choosing the right eps is critical and can be done using a k-distance plot.
2. min_samples (or minPts)
Minimum number of points required within eps radius for a point to be considered a
Determines density threshold for forming a cluster.
Influence:
Small min_samples → more clusters, possibly noisy ones.
Large min_samples → fewer, denser clusters; may label more points as noise.

A common default for min_samples is 4 or more.
Use a k-distance graph (plot the distance to the k-th nearest neighbor for each poi
'''
#How does K-Means++ improve upon standard K-Means initialization?
'''
What's the problem with standard K-Means?
In standard K-Means, the initial cluster centroids are chosen randomly. This can le
Poor clustering results
Slow convergence
Centroids getting stuck in local minima
Highly different results on different runs

How K-Means++ fixes this:
Steps in K-Means++ Initialization:
1.Randomly choose 1st centroid from the data points.
2.For each data point x,compute the distance  D(x) to the nearest chosen centroid.
3.Select the next centroid from the remaining points with probability proportional
chance).
4.Repeat until K centroids are chosen.
'''
# What is agglomerative clustering?
'''
Definition:
Agglomerative Clustering starts with each data point as its own cluster and repeate
until all points belong to one big cluster (or until a stopping criterion is met).

How it works (step-by-step):
Start: Each point is its own cluster.
Compute distances between all clusters (based on a linkage criterion).
Merge the two closest clusters.
Update the distance matrix.
Repeat steps 2–4 until:
You reach a desired number of clusters, or
All points are merged into a single cluster

Pros:
No need to specify number of clusters up front.
Works with different distance metrics and linkage methods.
Good for nested or hierarchical structure in data.

Cons:
Computationally expensive (especially for large datasets).
```

```
Not suitable for very large datasets without optimization.
Results can vary depending on linkage and distance choice.
'''
#What makes Silhouette Score a better metric than just inertia for model evaluation
'''
While inertia is commonly used in K-Means to evaluate clustering performance, the S
better, more informative metric,especially when you're trying to judge how well the

Inertia (used in K-Means):
Measures the sum of squared distances of points to their closest centroid.
Lower inertia = tighter clusters.

Limitations of Inertia:
Always decreases as you increase the number of clusters — makes it hard to decide t
Doesn't tell you how well-separated the clusters are.
Works only with K-Means, not generalizable to other algorithms.

In short:
Inertia is good for finding the "elbow" point.
Silhouette Score gives a richer, more balanced view of cluster quality.
Use both together when possible for robust evaluation.
```

Practical Questions

In [40]:
```
#Generate synthetic data with 4 centers using make_blobs and apply K-Means clusteri
import warnings
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Step 1: Generate synthetic data
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=42)

# Step 2: Apply K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=42)
y_kmeans = kmeans.fit_predict(X)

# Step 3: Plot the clustered data
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

# Plot the cluster centers
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X', l

plt.title("K-Means Clustering (4 Centers)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```

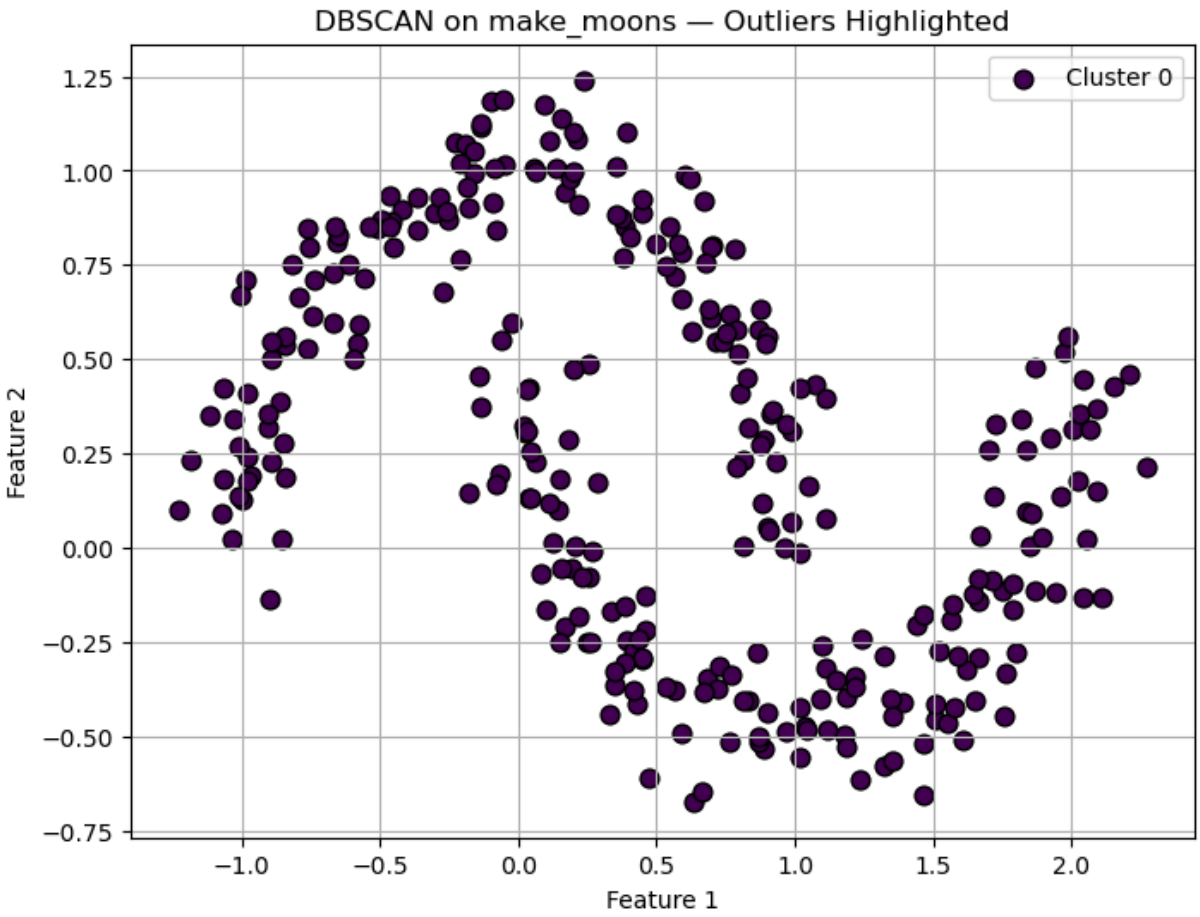


In [2]:
```
#Load the Iris dataset and use Agglomerative Clustering to group the data into 3 cl
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Use all features

# Step 2: Apply Agglomerative Clustering
agg_clustering = AgglomerativeClustering(n_clusters=3)
labels = agg_clustering.fit_predict(X)

# Step 3: Display the first 10 predicted labels
print("First 10 predicted cluster labels:")
print(labels[:10])
```

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

In [3]:
```
#Generate synthetic data using make_moons and apply DBSCAN. Highlight outliers in t
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN

# Step 1: Generate synthetic moon-shaped data
X, _ = make_moons(n_samples=300, noise=0.1, random_state=42)
```

```
# Step 2: Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(X)

# Step 3: Plot the result
plt.figure(figsize=(8, 6))

# Plot core clusters
unique_labels = set(labels)
for label in unique_labels:
    if label == -1:
        # Noise (outliers)
        color = 'red'
        marker = 'x'
        label_name = 'Outliers'
    else:
        color = plt.cm.viridis(label / len(unique_labels))
        marker = 'o'
        label_name = f'Cluster {label}'

    plt.scatter(X[labels == label, 0], X[labels == label, 1],
                c=[color], label=label_name, marker=marker, s=60, edgecolor='k')

plt.title("DBSCAN on make_moons – Outliers Highlighted")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```

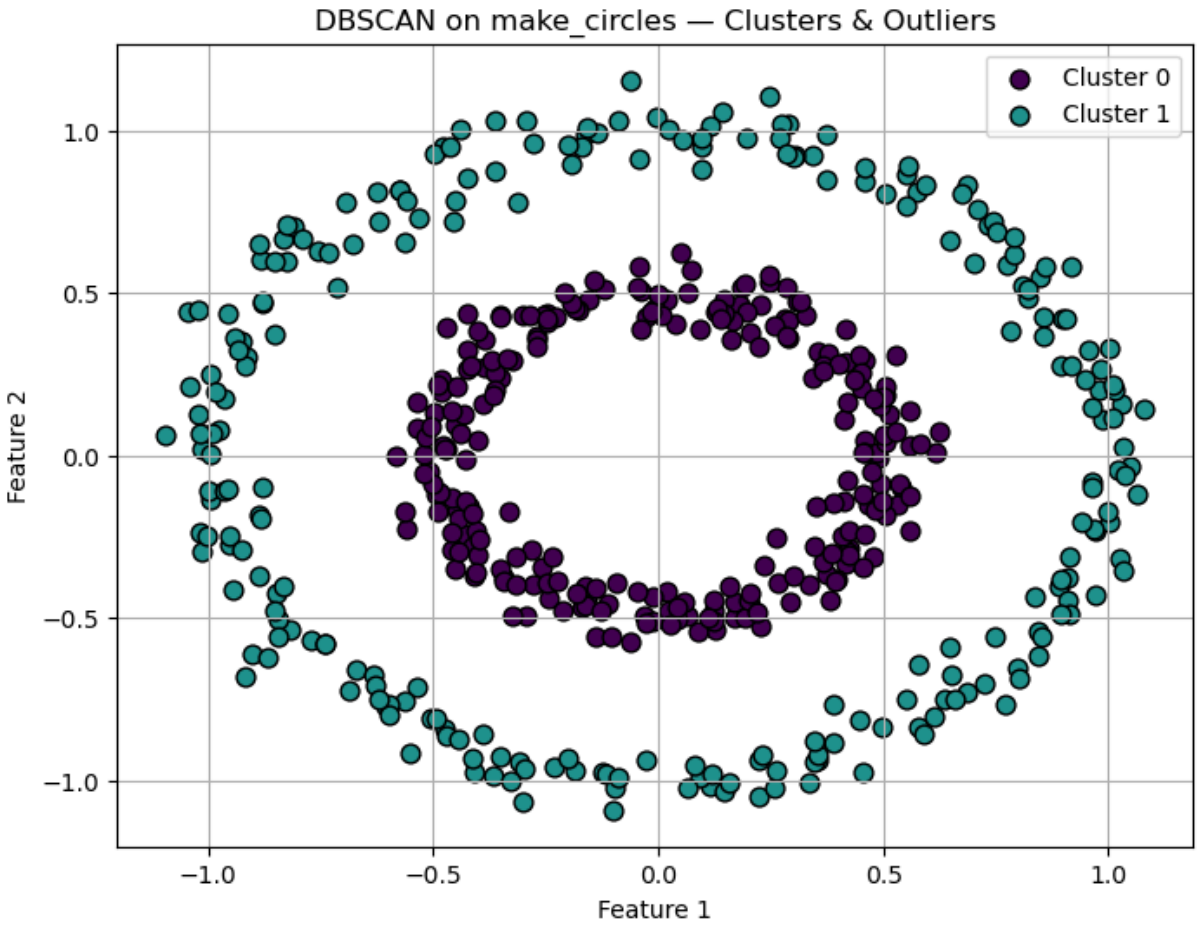


In [4]:
```python
#Load the Wine dataset and apply K-Means clustering after standardizing the feature
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import numpy as np

# Step 1: Load the Wine dataset
wine = load_wine()
X = wine.data

# Step 2: Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_scaled)

# Step 4: Print the size of each cluster
unique, counts = np.unique(labels, return_counts=True)
print("Cluster sizes:")
for cluster_id, size in zip(unique, counts):
    print(f"Cluster {cluster_id}: {size} samples")
```

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

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/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)
```

In [5]:
```python
# Use make_circles to generate synthetic data and cluster it using DBSCAN. Plot the
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.cluster import DBSCAN

# Step 1: Generate synthetic circular data
X, _ = make_circles(n_samples=500, factor=0.5, noise=0.05, random_state=42)

# Step 2: Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.15, min_samples=5)
labels = dbscan.fit_predict(X)

# Step 3: Plot the result
plt.figure(figsize=(8, 6))

# Plot clusters and noise
unique_labels = set(labels)
for label in unique_labels:
    is_noise = label == -1
    label_name = 'Outliers' if is_noise else f'Cluster {label}'
    marker = 'x' if is_noise else 'o'
    color = 'red' if is_noise else plt.cm.viridis(label / len(unique_labels))

    plt.scatter(X[labels == label, 0], X[labels == label, 1],
                c=[color], label=label_name, marker=marker, s=60, edgecolor='k')

plt.title("DBSCAN on make_circles – Clusters & Outliers")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```

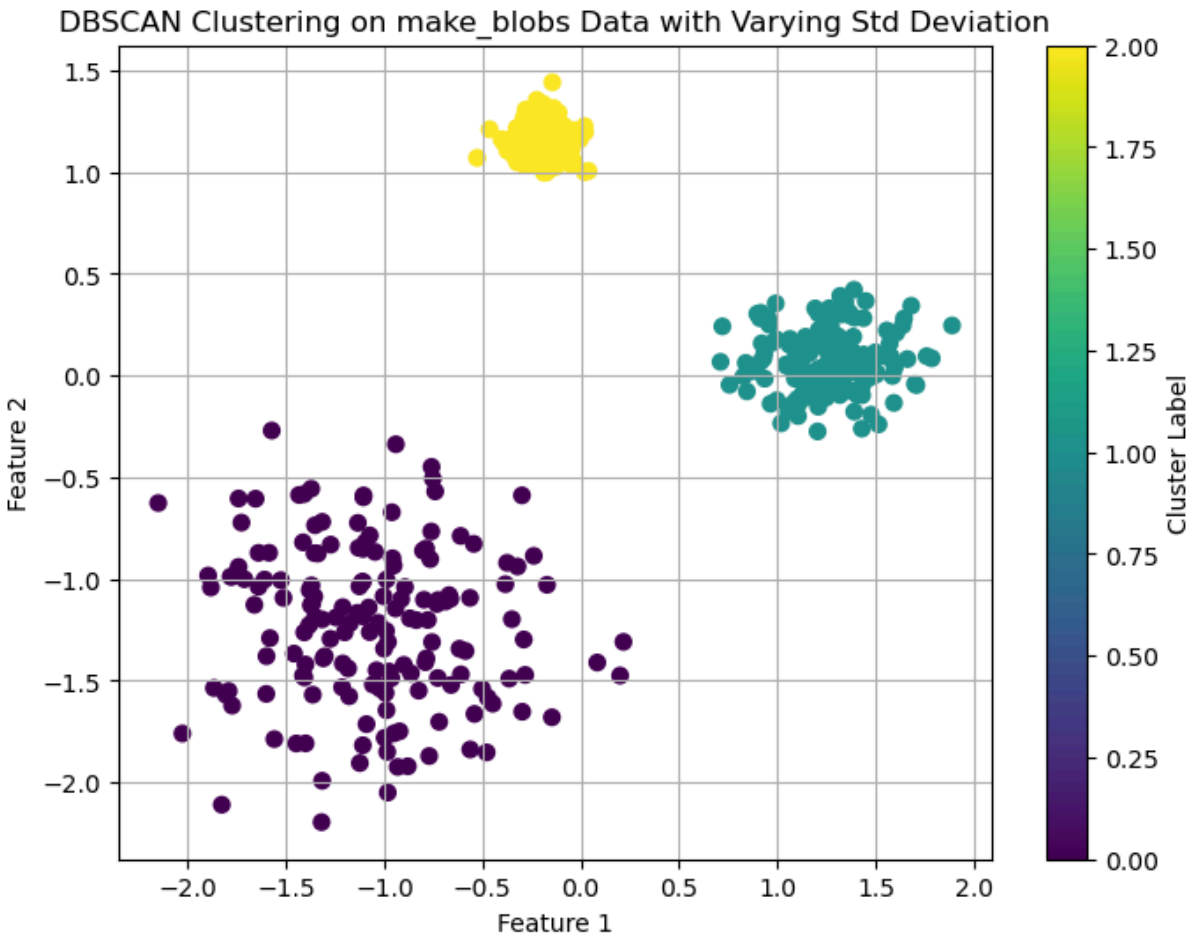


In [2]:
```
#Load the Breast Cancer dataset, apply MinMaxScaler, and use K-Means with 2 cluster
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import pandas as pd

# Load dataset
data = load_breast_cancer()
X = data.data

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

# Apply KMeans clustering with 2 clusters and explicit n_init to avoid warning
kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)
kmeans.fit(X_scaled)

# Print cluster centroids
centroids_df = pd.DataFrame(kmeans.cluster_centers_, columns=data.feature_names)
print("Cluster Centroids:")
print(centroids_df)
```

```
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]
```

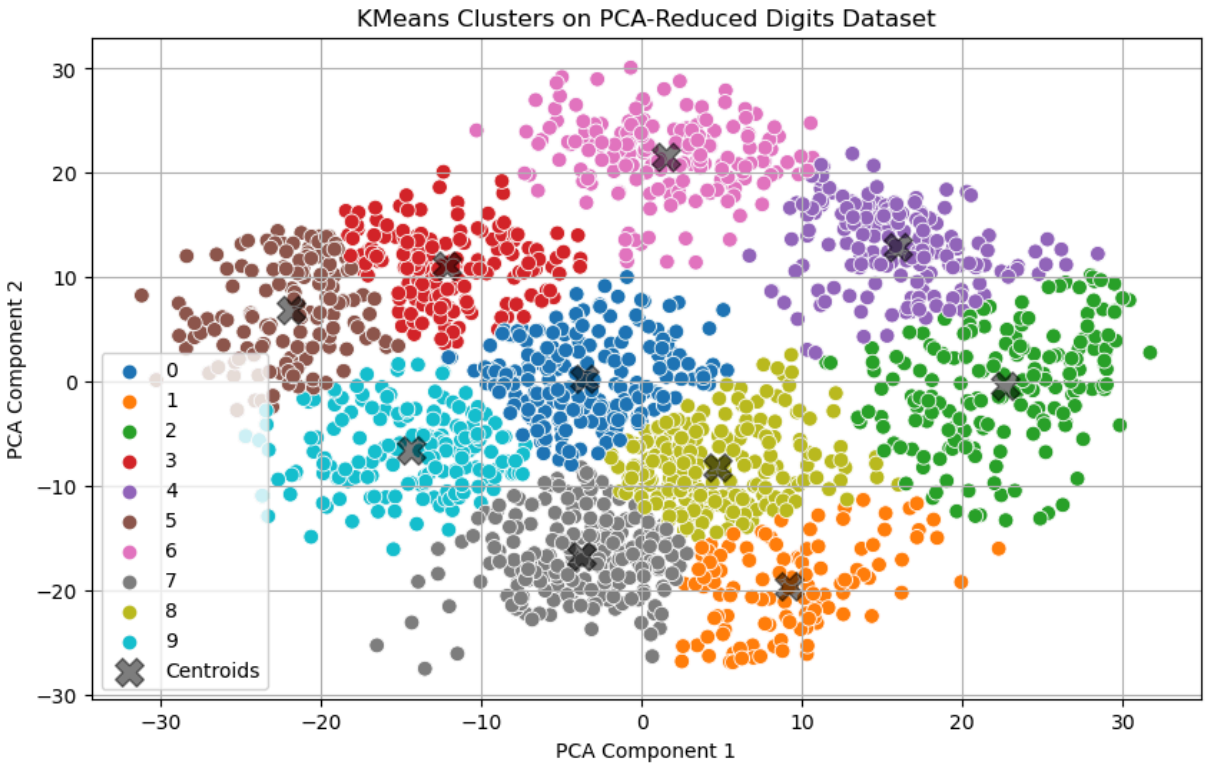
In [3]:

```python
#Generate synthetic data using make_blobs with varying cluster standard deviations

import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Step 1: Generate synthetic data with varying cluster std deviations
X, y = make_blobs(n_samples=500,
                  centers=3,
                  cluster_std=[0.5, 1.0, 2.5],
                  random_state=42)

#  Standardize the data for DBSCAN
X_scaled = StandardScaler().fit_transform(X)

# Step 2: Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X_scaled)

# Step 3: Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', s=40)
plt.title("DBSCAN Clustering on make_blobs Data with Varying Std Deviation")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.colorbar(label='Cluster Label')
plt.grid(True)
plt.show()
```



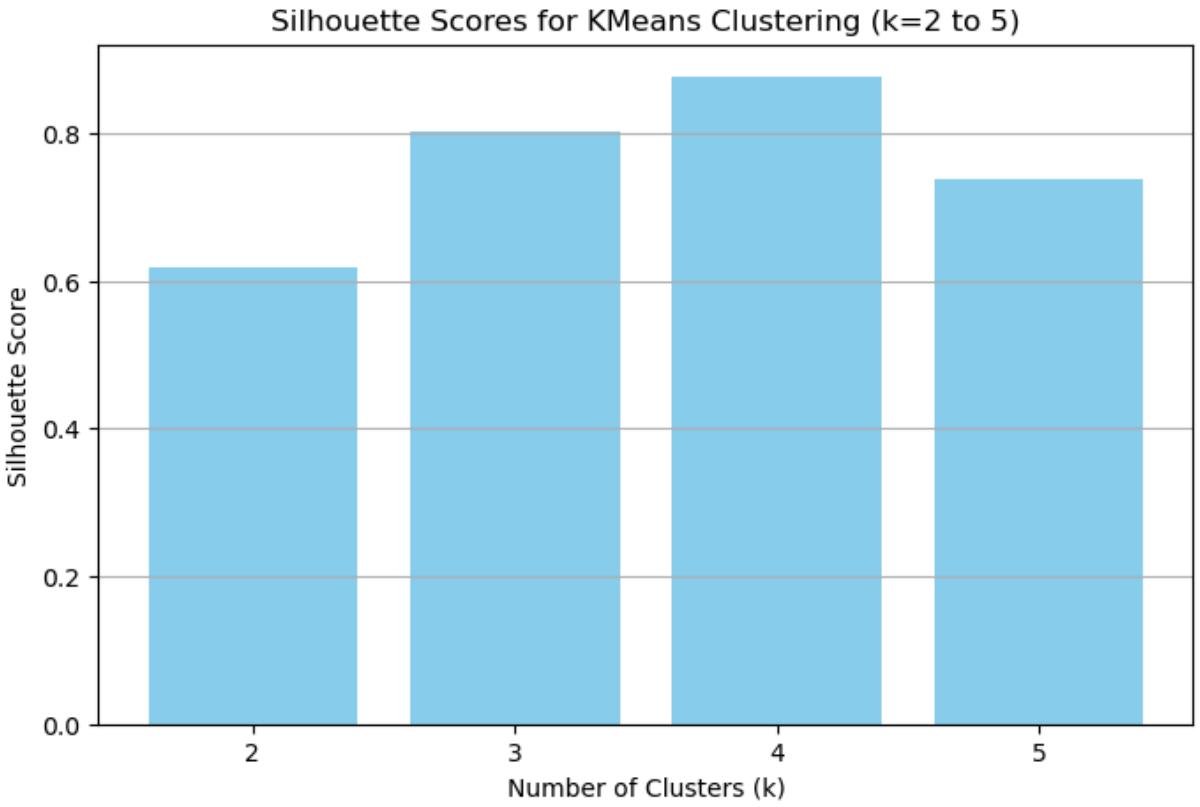
In [4]:
```
#Load the Digits dataset, reduce it to 2D using PCA, and visualize clusters from K-
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import seaborn as sns

# Step 1: Load the Digits dataset
digits = load_digits()
X = digits.data
y = digits.target

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

# Step 3: Apply KMeans clustering
kmeans = KMeans(n_clusters=10, n_init=10, random_state=42)
clusters = kmeans.fit_predict(X_pca)

# Step 4: Visualize the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=clusters, palette="tab10", s=60,
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            c='black', s=200, alpha=0.5, marker='X', label='Centroids')
plt.title("KMeans Clusters on PCA-Reduced Digits Dataset")
plt.xlabel("PCA Component 1")
```

```
plt.ylabel("PCA Component 2")
plt.legend()
plt.grid(True)
plt.show()
```

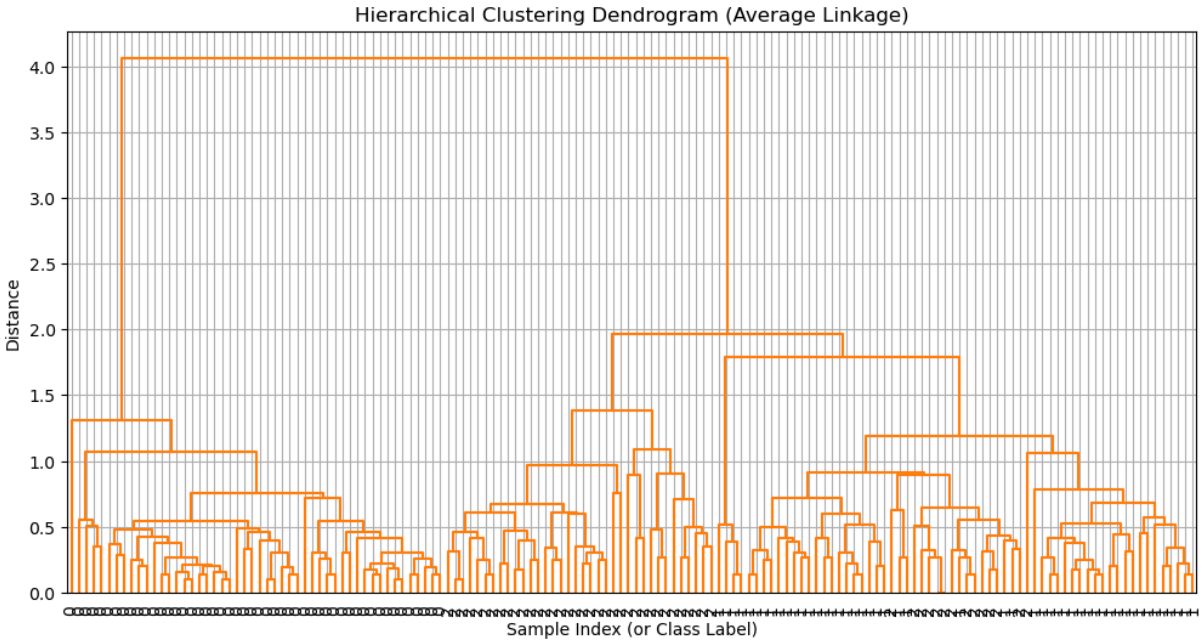


In [5]:
```
#Create synthetic data using make_blobs and evaluate silhouette scores for k = 2 to
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Step 1: Generate synthetic data
X, y = make_blobs(n_samples=500, centers=4, cluster_std=0.6, random_state=42)

# Step 2: Evaluate silhouette scores for k = 2 to 5
silhouette_scores = []
k_values = range(2, 6)

for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    labels = kmeans.fit_predict(X)
    score = silhouette_score(X, labels)
    silhouette_scores.append(score)

# Step 3: Plot as a bar chart
plt.figure(figsize=(8, 5))
plt.bar(k_values, silhouette_scores, color='skyblue')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Scores for KMeans Clustering (k=2 to 5)")
plt.xticks(k_values)
```

```
plt.grid(axis='y')
plt.show()
```

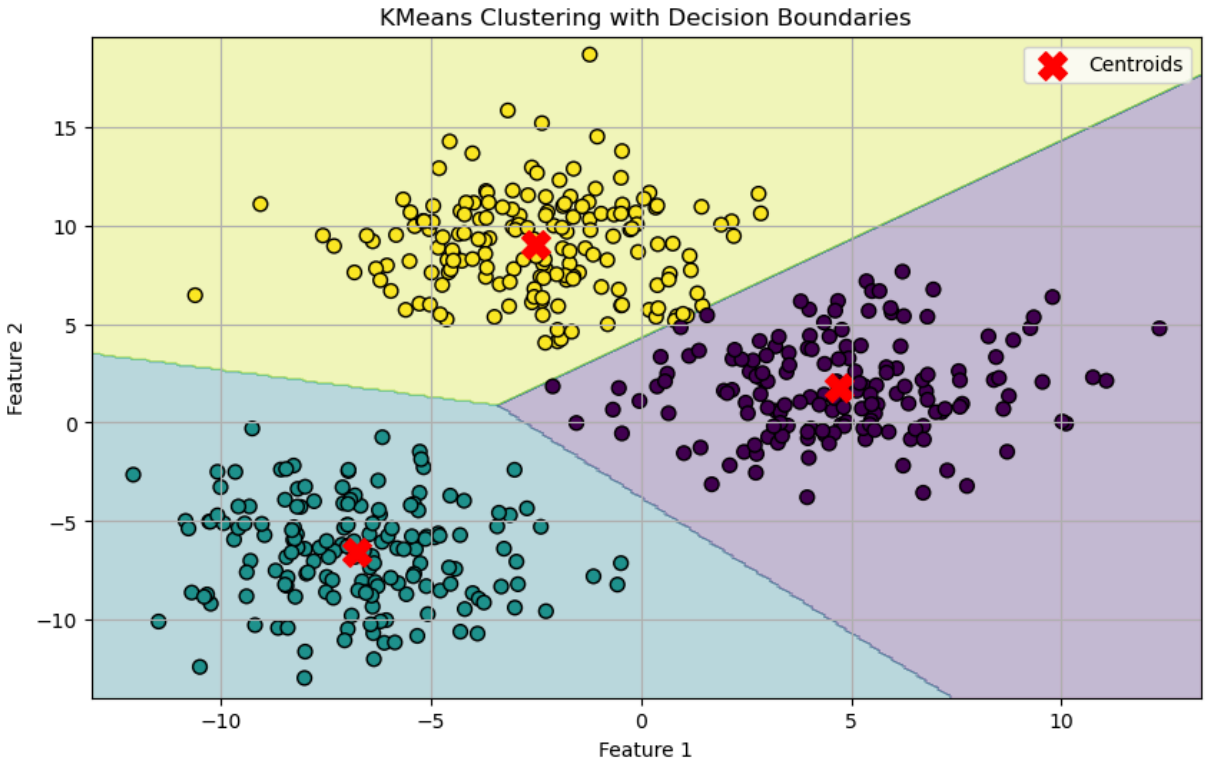


In [6]:
```
# Load the Iris dataset and use hierarchical clustering to group data. Plot a dendr
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import linkage, dendrogram

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data

# Step 2: Perform hierarchical clustering with 'average' linkage
linked = linkage(X, method='average')

# Step 3: Plot the dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linked,
           labels=iris.target,
           leaf_rotation=90,
           leaf_font_size=10,
           color_threshold=7)
plt.title("Hierarchical Clustering Dendrogram (Average Linkage)")
plt.xlabel("Sample Index (or Class Label)")
plt.ylabel("Distance")
plt.grid(True)
plt.show()
```



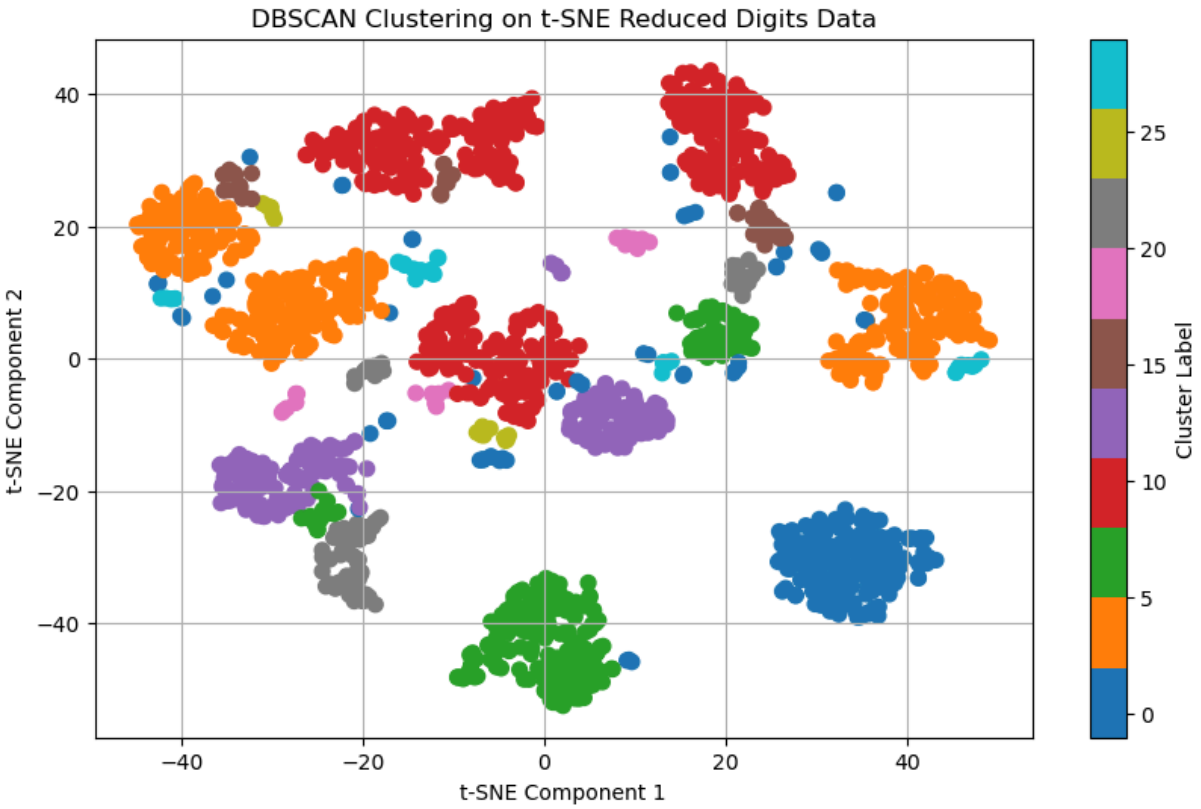
In [7]:
```
#Generate synthetic data with overlapping clusters using make_blobs, then apply K-M
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Step 1: Generate synthetic data with overlapping clusters
X, y = make_blobs(n_samples=500, centers=3, cluster_std=2.5, random_state=42)

# Step 2: Apply KMeans clustering
kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
kmeans.fit(X)
labels = kmeans.predict(X)
centroids = kmeans.cluster_centers_

# Step 3: Visualize with decision boundaries

h = 0.1
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))

# Predict cluster for each point in mesh
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundaries and data points
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', edgecolor='k', s=50)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='X', label='Ce
plt.title("KMeans Clustering with Decision Boundaries")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
```

```
plt.grid(True)
plt.show()
```

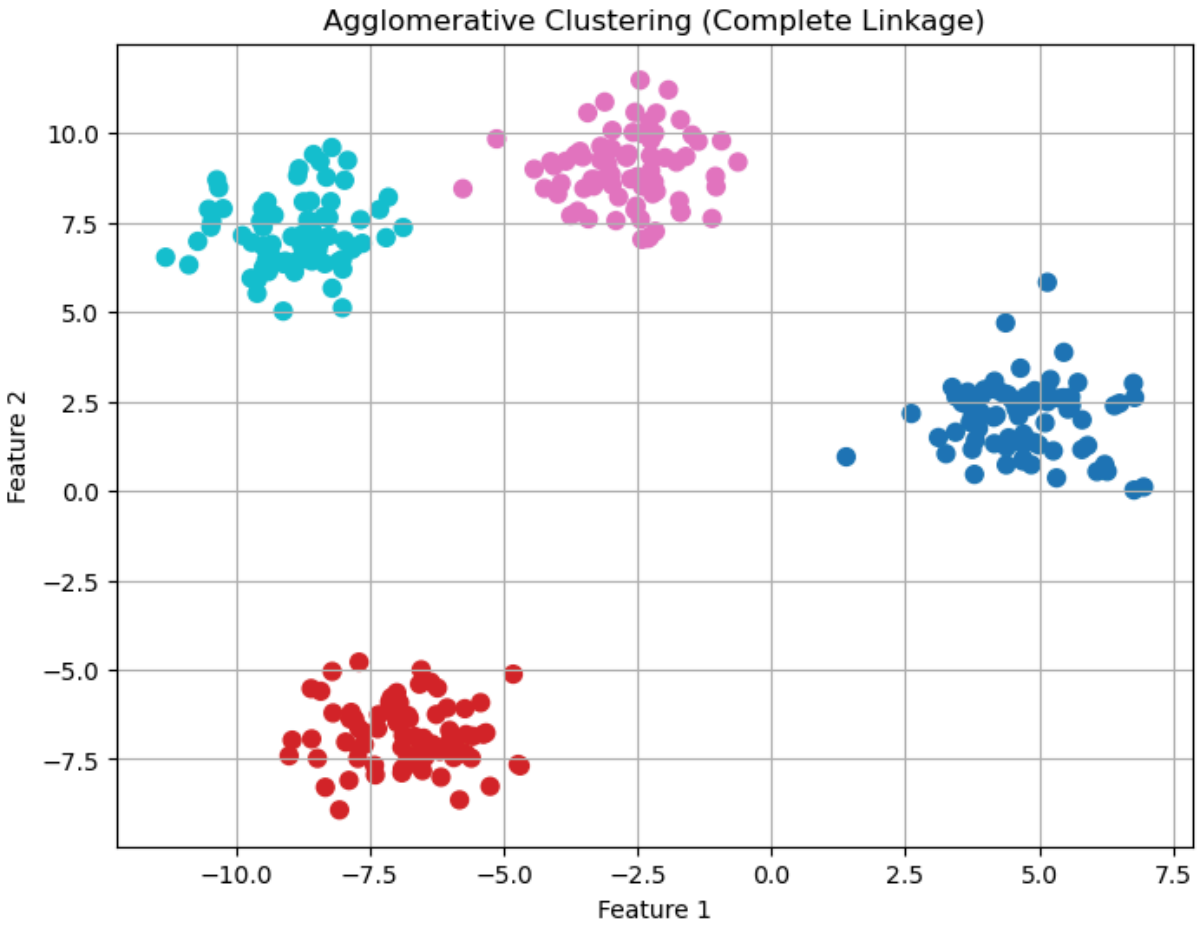


In [8]:
```
#Load the Digits dataset and apply DBSCAN after reducing dimensions with t-SNE. Vis
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Step 1: Load the Digits dataset
digits = load_digits()
X = digits.data

# Step 2: Standardize the data
X_scaled = StandardScaler().fit_transform(X)

# Step 3: Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30, init='pca', learning_ra
X_tsne = tsne.fit_transform(X_scaled)

# Step 4: Apply DBSCAN
dbscan = DBSCAN(eps=2, min_samples=5)
labels = dbscan.fit_predict(X_tsne)

# Step 5: Visualize the results
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=labels, cmap='tab10', s=50)
plt.title("DBSCAN Clustering on t-SNE Reduced Digits Data")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.colorbar(label="Cluster Label")
```

```
plt.grid(True)
plt.show()
```

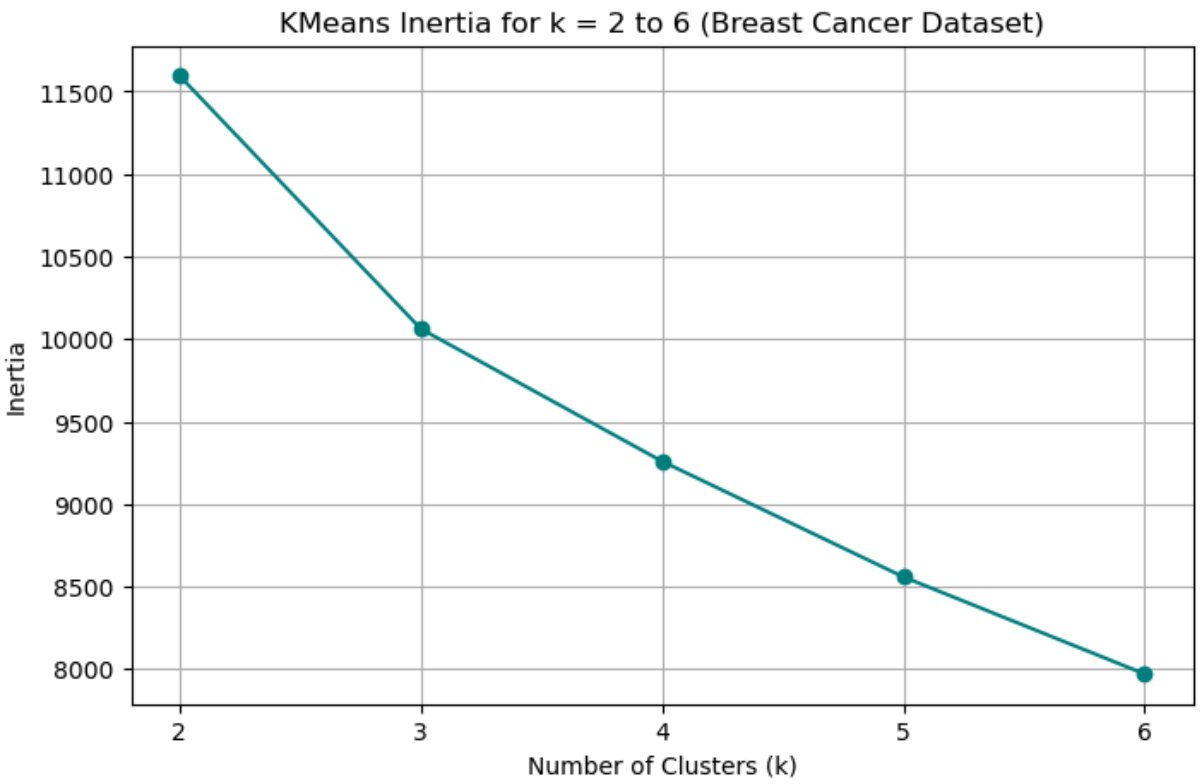


In [9]:
```
#Generate synthetic data using make_blobs and apply Agglomerative Clustering with c
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

# 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 complete linkage
agglo = AgglomerativeClustering(n_clusters=4, linkage='complete')
labels = agglo.fit_predict(X)

# Step 3: Plot the clustered data
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='tab10', s=50)
plt.title("Agglomerative Clustering (Complete Linkage)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.show()
```

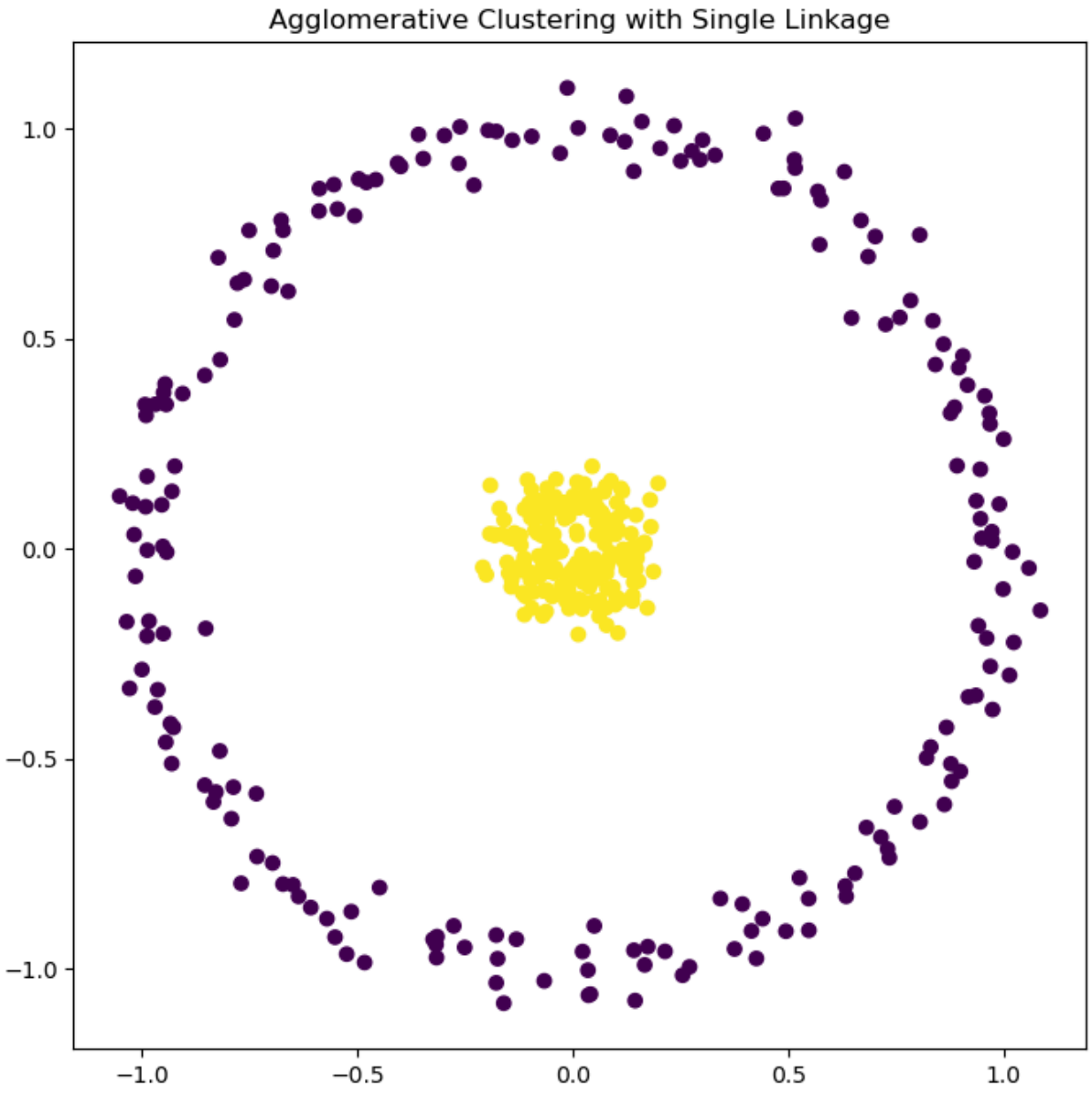


In [10]:
```
#Load the Breast Cancer dataset and compare inertia values for K = 2 to 6 using K-M
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Step 1: Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data

# Step 2: Standardize the data
X_scaled = StandardScaler().fit_transform(X)

# Step 3: Apply KMeans for k = 2 to 6 and store inertia
inertia_values = []
k_values = range(2, 7)

for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X_scaled)
    inertia_values.append(kmeans.inertia_)

# Step 4: Plot the inertia values (Elbow plot)
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia_values, marker='o', linestyle='-', color='teal')
plt.title("KMeans Inertia for k = 2 to 6 (Breast Cancer Dataset)")
plt.xlabel("Number of Clusters (k)")
```

```
plt.ylabel("Inertia")
plt.xticks(k_values)
plt.grid(True)
plt.show()
```



In [11]:
```
#Generate synthetic concentric circles using make_circles and cluster using Agglome
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.cluster import AgglomerativeClustering
import numpy as np

X, y = make_circles(n_samples=400, factor=0.1, noise=0.05)
clustering = AgglomerativeClustering(n_clusters=2, linkage='single').fit(X)

plt.figure(figsize=(8, 8))
plt.scatter(X[:, 0], X[:, 1], c=clustering.labels_, cmap='viridis')
plt.title("Agglomerative Clustering with Single Linkage")
plt.show()
```

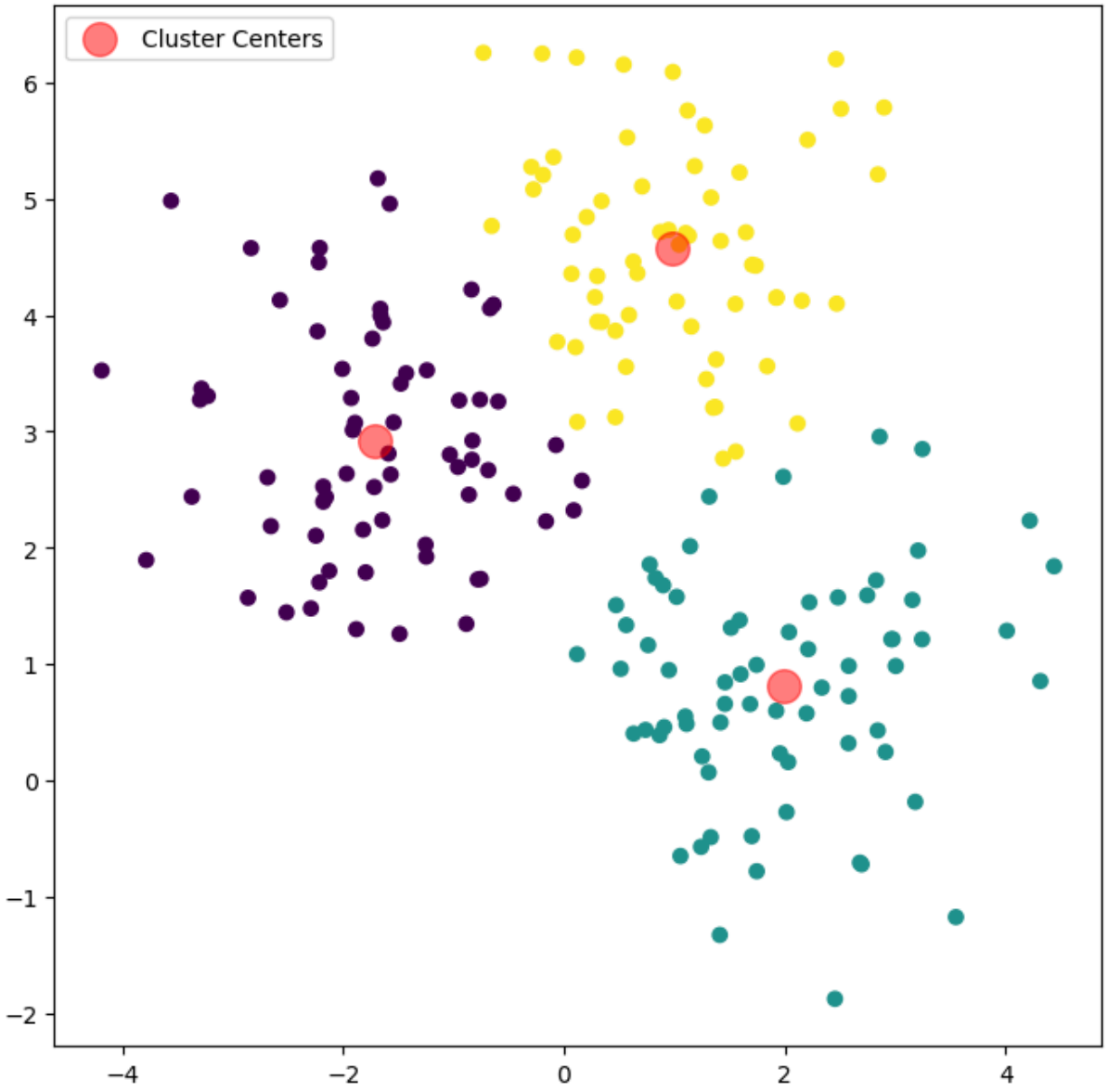


In [15]:
```
#Use the Wine dataset, apply DBSCAN after scaling the data, and count the number of
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import numpy as np

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

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

#Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X_scaled)

#Count the number of clusters (excluding noise)
```

```
n_clusters = len(np.unique(dbscan.labels_)) - (1 if -1 in dbscan.labels_ else 0)
print("Number of clusters:", n_clusters)
```

```
Number of clusters: 0
```

In [38]:
```
#Generate synthetic data with make_blobs and apply KMeans. Then plot the cluster ce


import warnings
X, y = make_blobs(n_samples=200, centers=3, n_features=2, random_state=0)

#Apply K-Means clustering
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

#Plot the data points and cluster centers
plt.figure(figsize=(8, 8))
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red',
plt.legend()
plt.show()
```

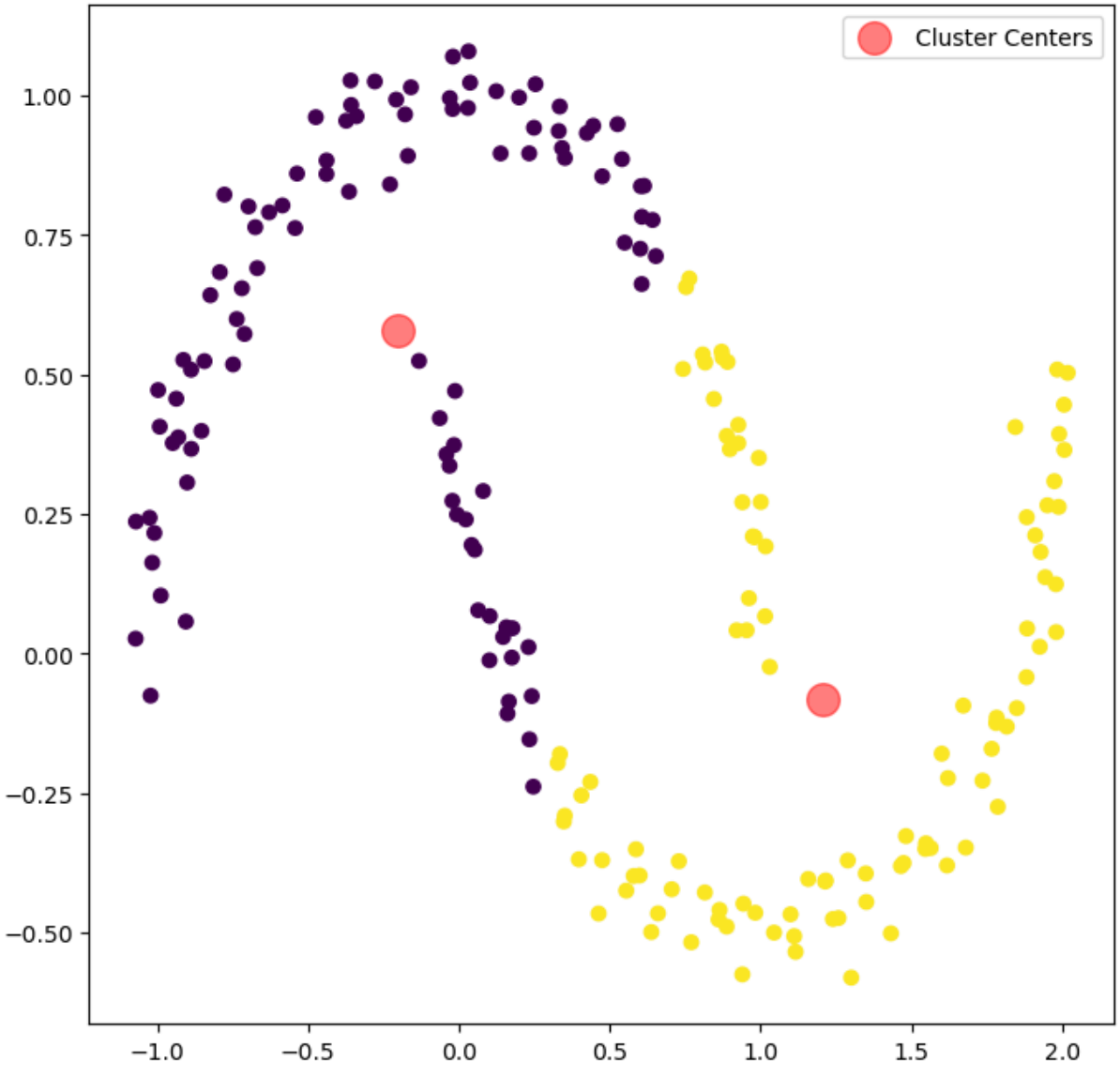


In [18]:
```
#Load the Iris dataset, cluster with DBSCAN, and print how many samples were identi

X, y = make_blobs(n_samples=200, centers=3, n_features=2, random_state=0)


dbscan = DBSCAN(eps=0.5, min_samples=10)
dbscan.fit(X)


noise_points = np.sum(dbscan.labels_ == -1)
print("Number of samples identified as noise:", noise_points)
```

```
Number of samples identified as noise: 158
```

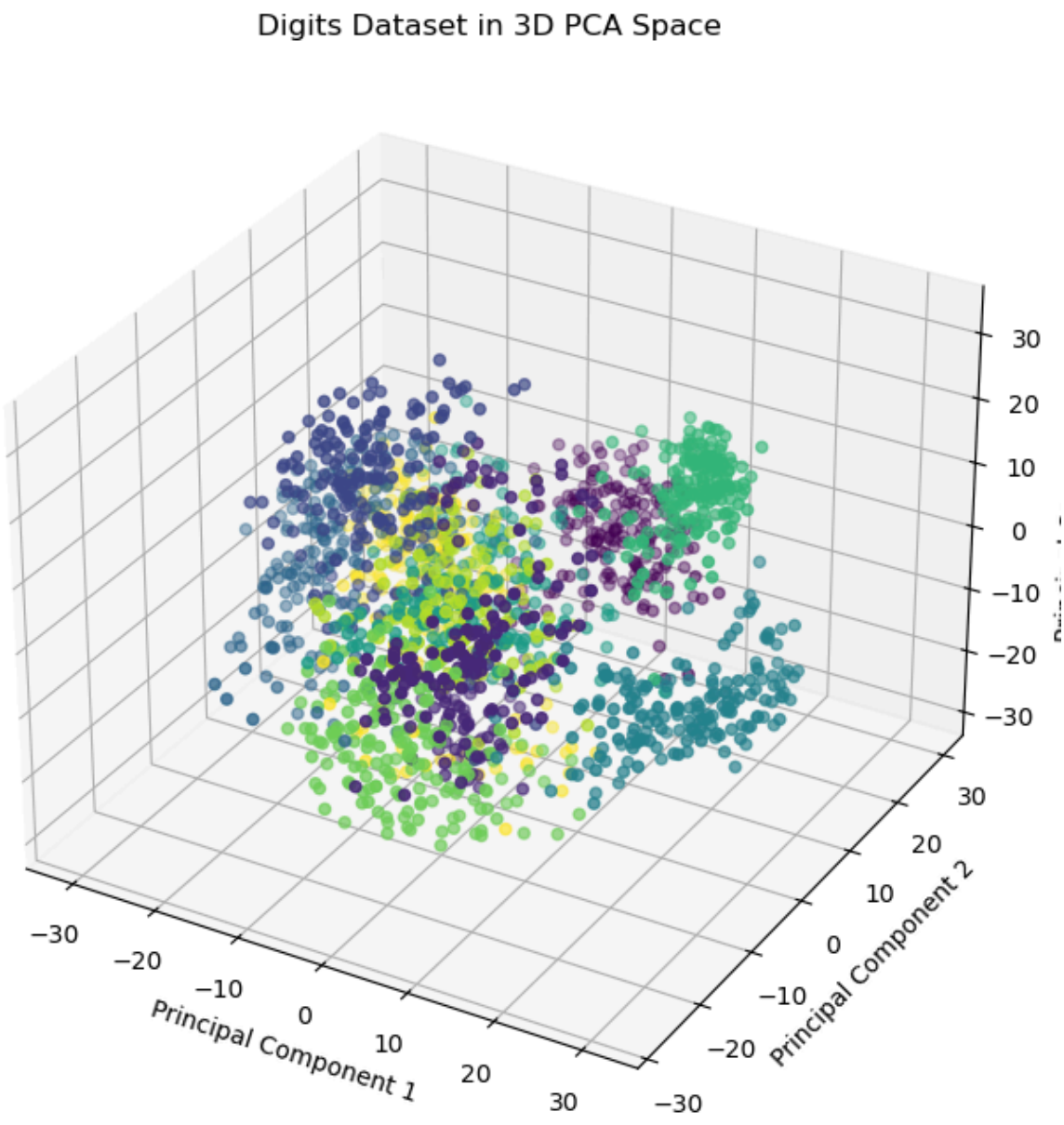
In [37]:
```
#Generate synthetic non-linearly separable data using make_moons, apply K-Means, an
import warnings
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
```

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)


plt.figure(figsize=(8, 8))
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red',
plt.legend()
plt.show()
```



In [20]:
```
#Load the Digits dataset, apply PCA to reduce to 3 components, then use KMeans and

from mpl_toolkits.mplot3d import Axes3D



digits = load_digits()
X = digits.data
y = digits.target
```

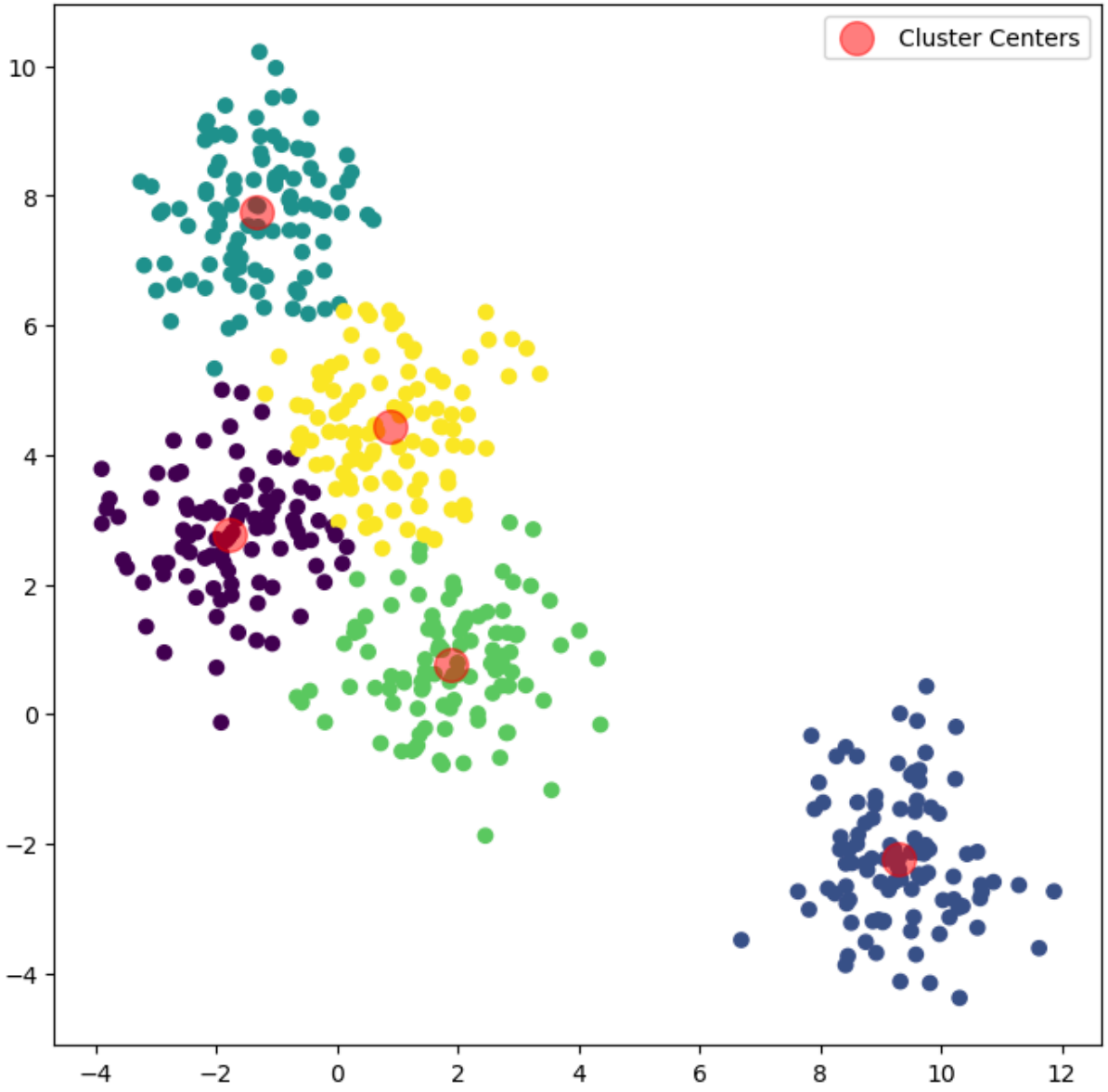
```
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)


fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=y, cmap='viridis')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
plt.title('Digits Dataset in 3D PCA Space')
plt.show()
```



In [36]:
```
# Generate synthetic blobs with 5 centers and apply KMeans. Then use silhouette_sco
import warnings
X, y = make_blobs(n_samples=500, centers=5, n_features=2, random_state=0)
```

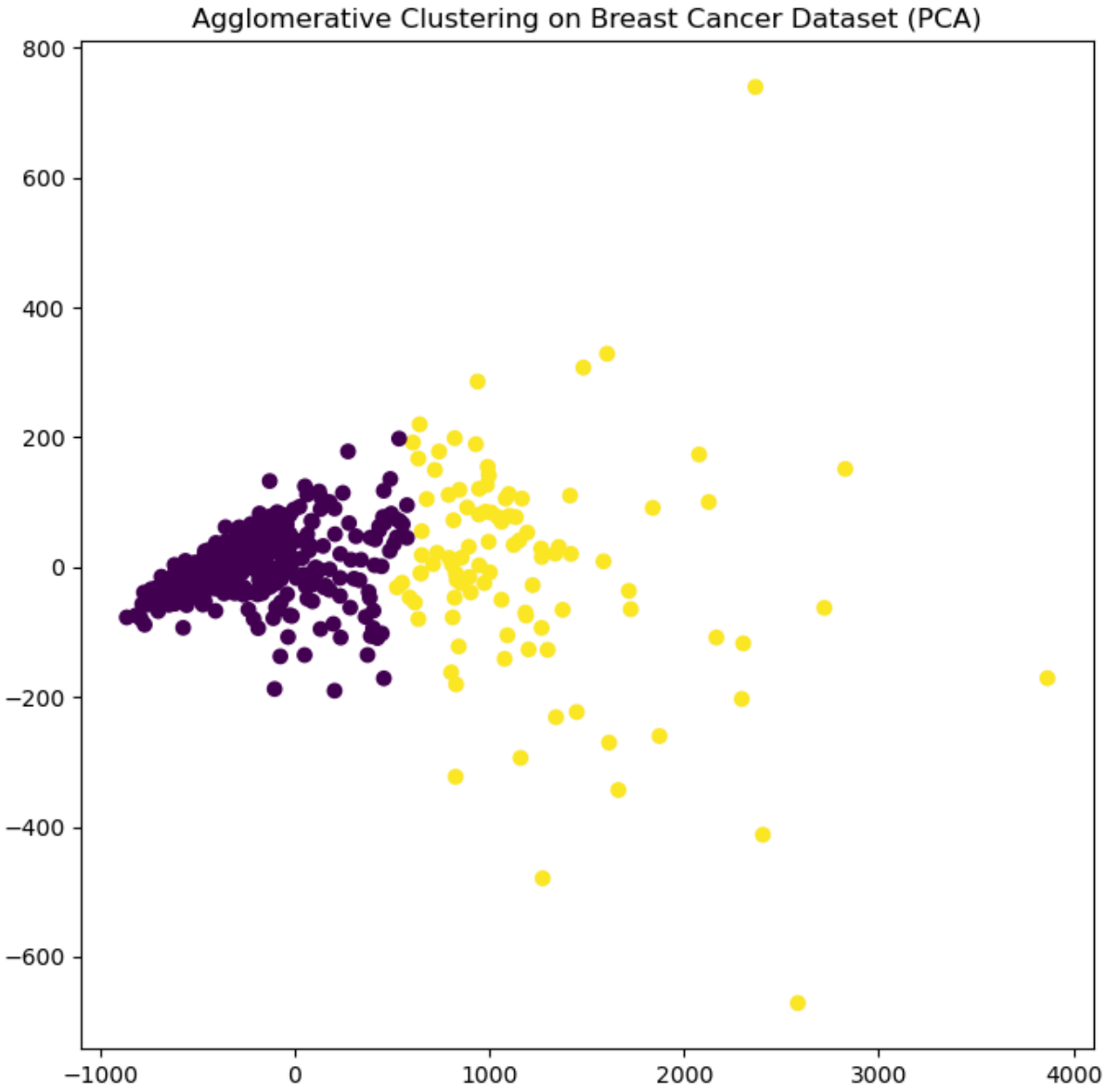
```
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)


silhouette = silhouette_score(X, kmeans.labels_)
print("Silhouette Score:", silhouette)


plt.figure(figsize=(8, 8))
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red',
plt.legend()
plt.show()
```

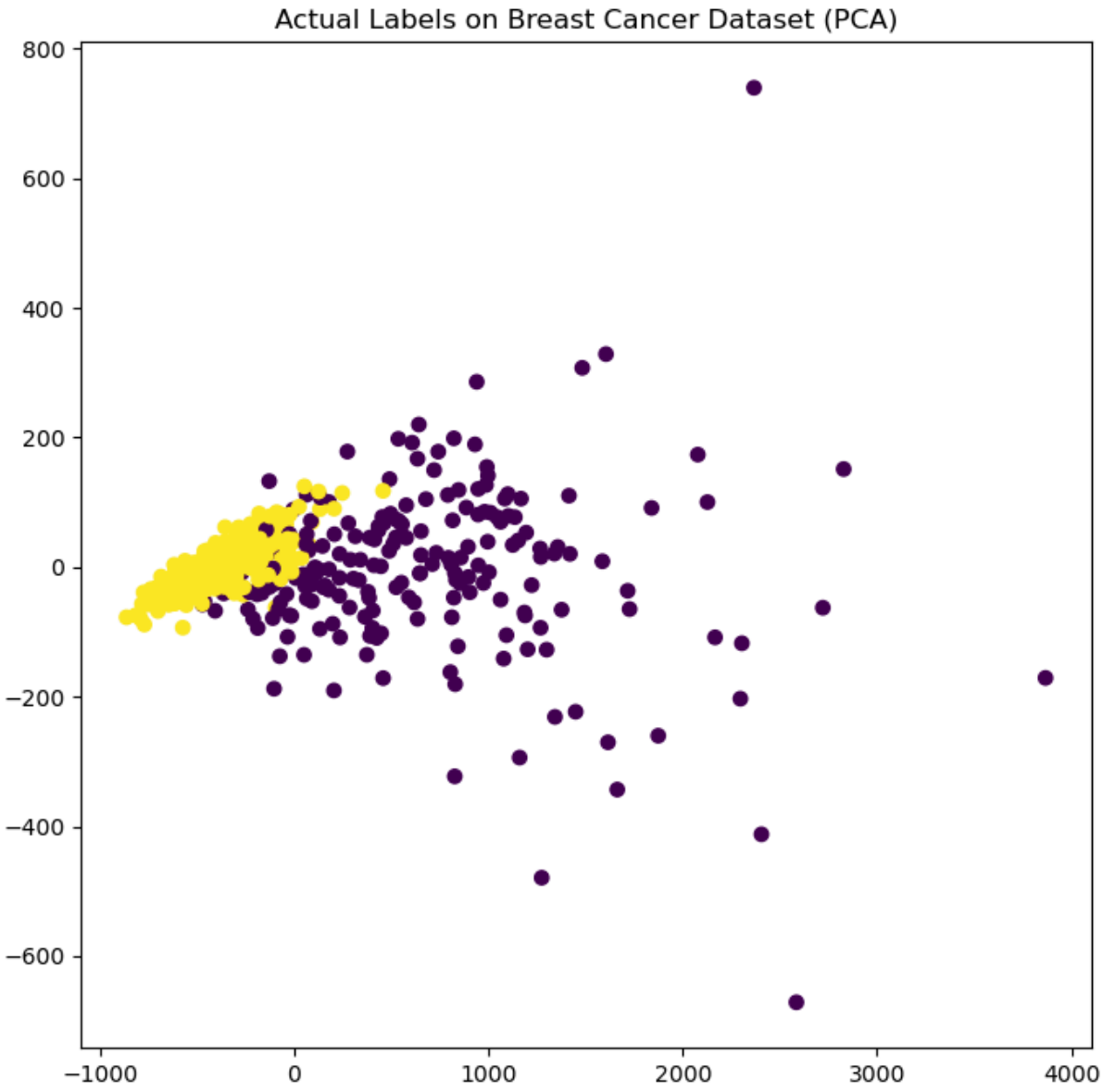
```
Silhouette Score: 0.5588132258276344
```



In [22]:
```
#Load the Breast Cancer dataset, reduce dimensionality using PCA, and apply Agglome
cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
```

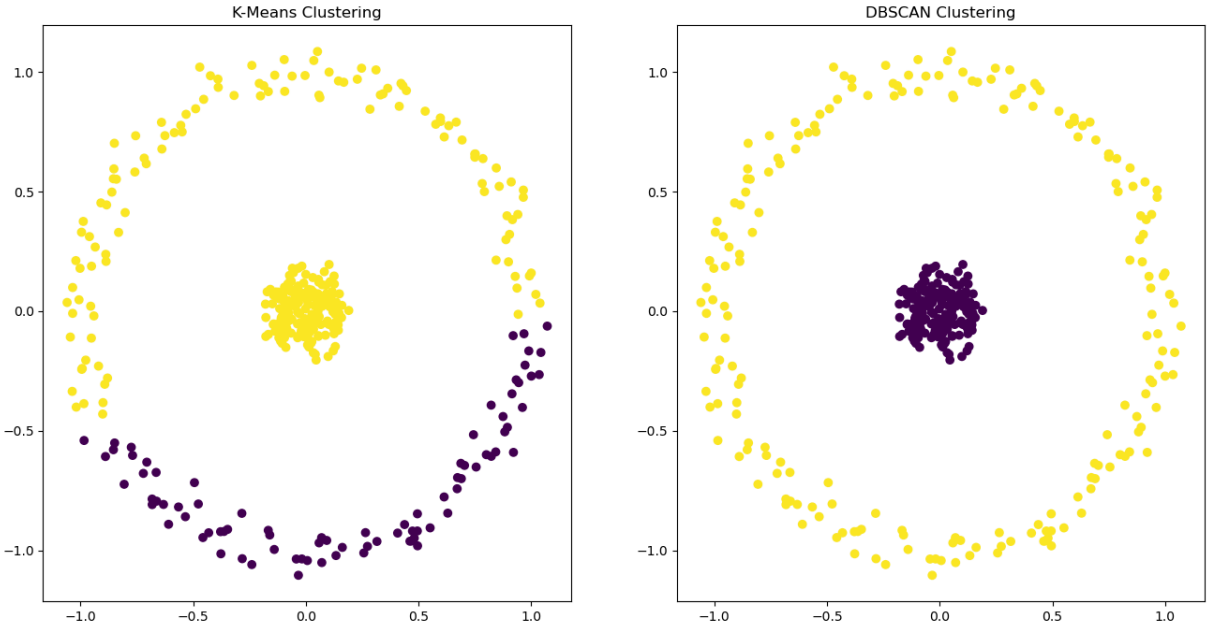
```
#Apply PCA to reduce dimensionality to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

#Apply Agglomerative clustering
agglo = AgglomerativeClustering(n_clusters=2)
agglo.fit(X_pca)
labels = agglo.labels_

#Visualize the clustering result
plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis')
plt.title('Agglomerative Clustering on Breast Cancer Dataset (PCA)')
plt.show()

plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title('Actual Labels on Breast Cancer Dataset (PCA)')
plt.show()
```





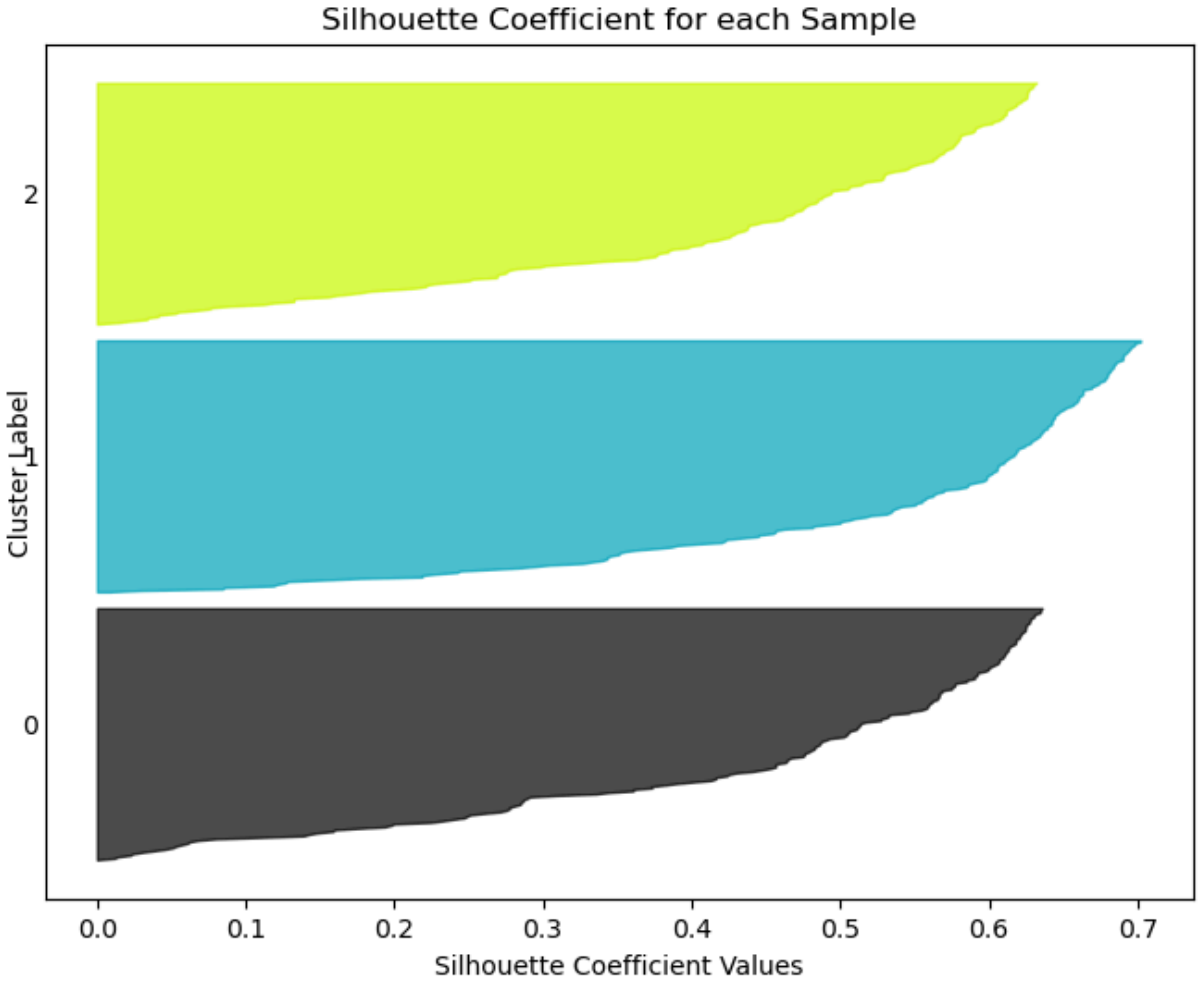
In [34]:
```
#Generate noisy circular data using make_circles and visualize clustering results f
import warnings
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=400, factor=0.1, noise=0.05)


kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
kmeans_labels = kmeans.labels_


dbscan = DBSCAN(eps=0.2, min_samples=10)
dbscan.fit(X)
dbscan_labels = dbscan.labels_


fig, axs = plt.subplots(1, 2, figsize=(16, 8))

axs[0].scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap='viridis')
```

```
axs[0].set_title('K-Means Clustering')

axs[1].scatter(X[:, 0], X[:, 1], c=dbscan_labels, cmap='viridis')
axs[1].set_title('DBSCAN Clustering')

plt.show()
```



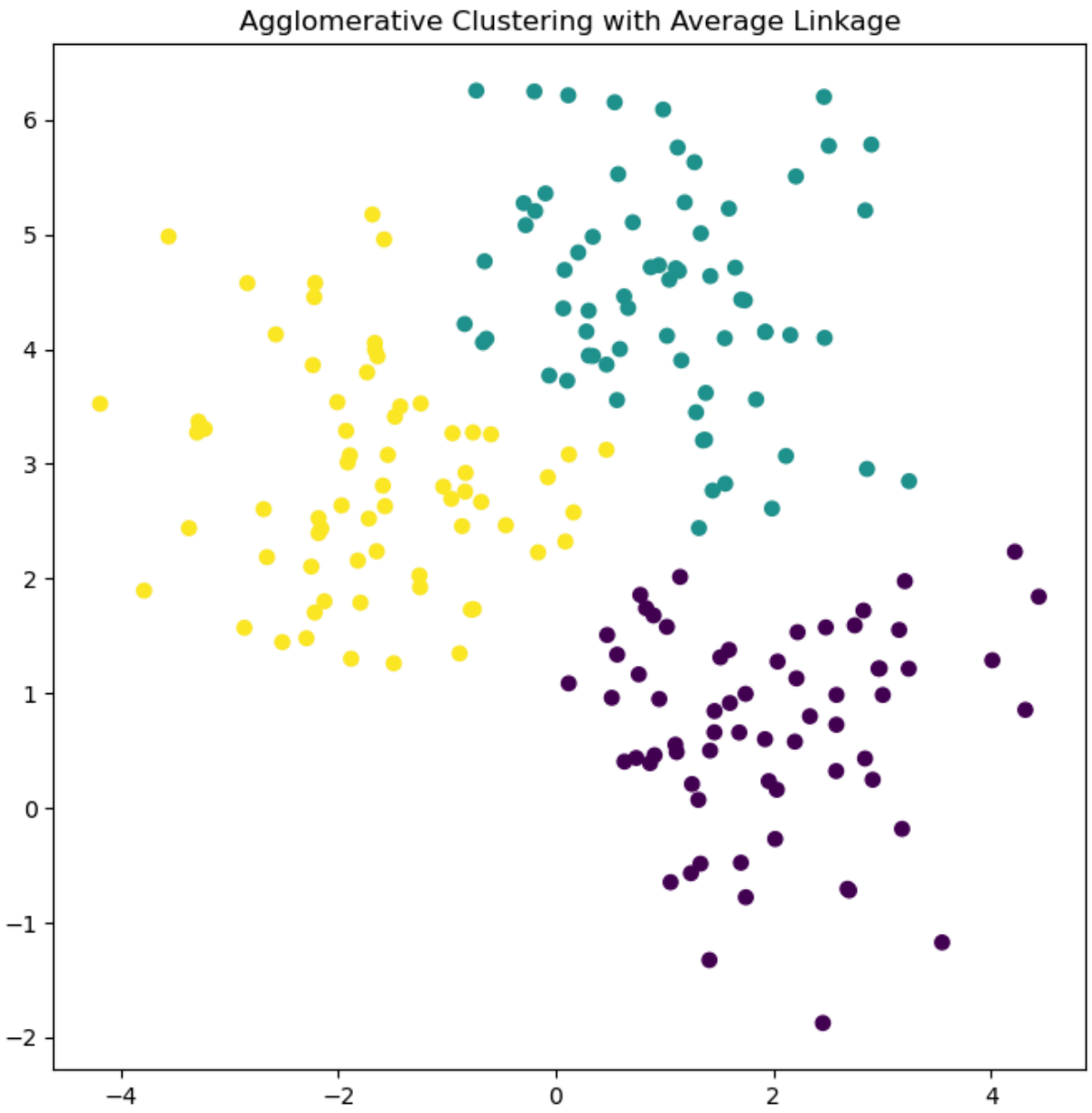
In [33]:
```
#Load the Iris dataset and plot the Silhouette Coefficient for each sample after KM
import warnings
from sklearn.metrics import silhouette_score, silhouette_samples

X, y = make_blobs(n_samples=500, centers=3, n_features=2, random_state=0)


kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
labels = kmeans.labels_


silhouette_values = silhouette_samples(X, labels)


plt.figure(figsize=(8, 6))
y_lower = 10
for i in range(3):
    ith_cluster_silhouette_values = silhouette_values[labels == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = plt.cm.nipy_spectral(float(i) / 3)
    plt.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)
    plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10
```

```
plt.title("Silhouette Coefficient for each Sample")
plt.xlabel("Silhouette Coefficient Values")
plt.ylabel("Cluster Label")
plt.yticks([])
plt.show()


avg_silhouette = silhouette_score(X, labels)
print("Average Silhouette Score:", avg_silhouette)
```



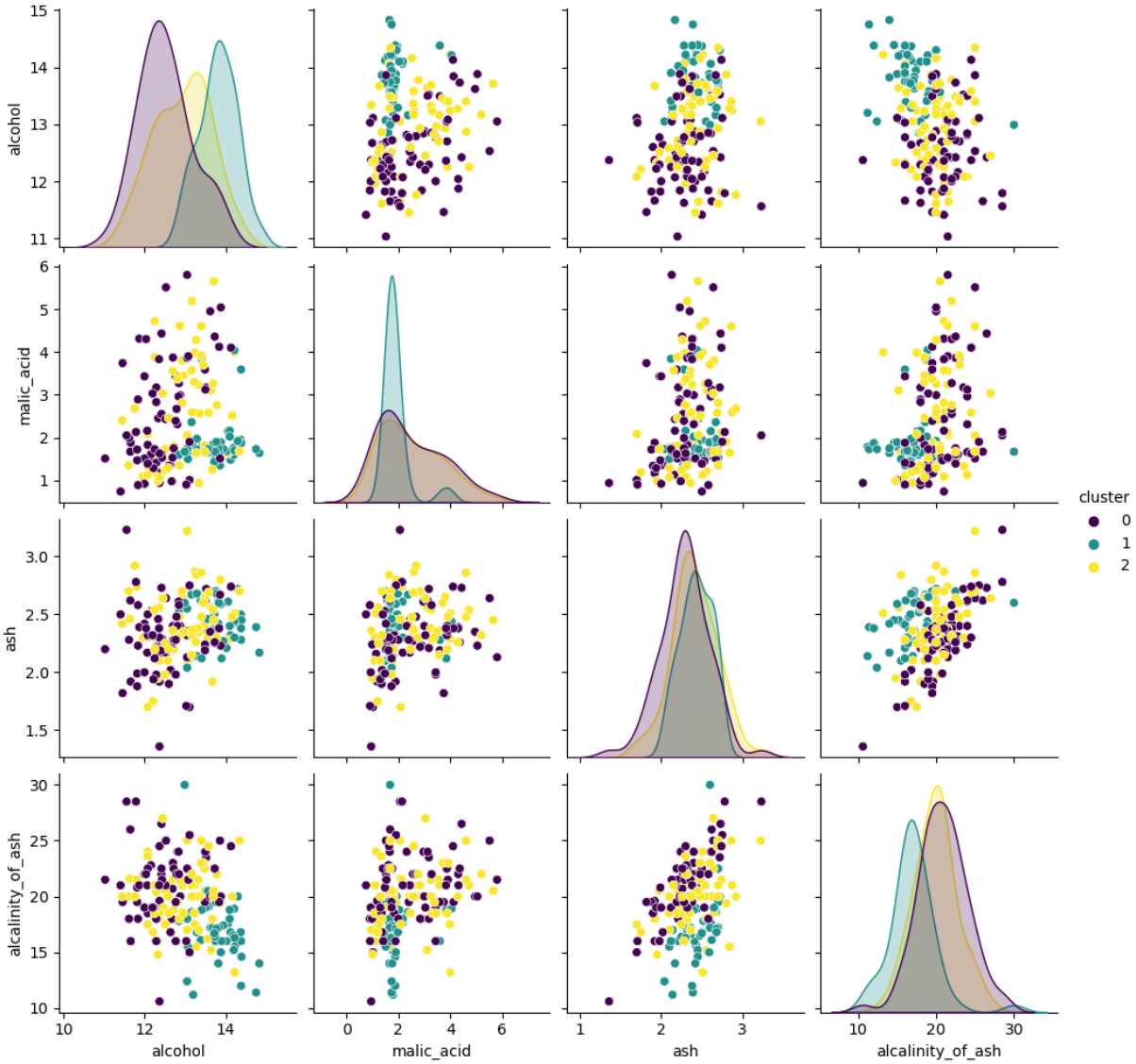

```
Average Silhouette Score: 0.47229538453502573
```

In [28]:
```
#Generate synthetic data using make_blobs and apply Agglomerative Clustering with '


X, y = make_blobs(n_samples=200, centers=3, n_features=2, random_state=0)
agglo = AgglomerativeClustering(n_clusters=3, linkage='average')
agglo.fit(X)
labels = agglo.labels_

plt.figure(figsize=(8, 8))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('Agglomerative Clustering with Average Linkage')
plt.show()
```



In [30]:
```
#Load the Wine dataset, apply KMeans, and visualize the cluster assignments in a se
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.cluster import KMeans
import pandas as pd

warnings.simplefilter(action='ignore', category=FutureWarning)

wine = load_wine()
X = wine.data
y = wine.target
feature_names = wine.feature_names

kmeans = KMeans(n_clusters=3, n_init=10)
kmeans.fit(X)
labels = kmeans.labels_
```

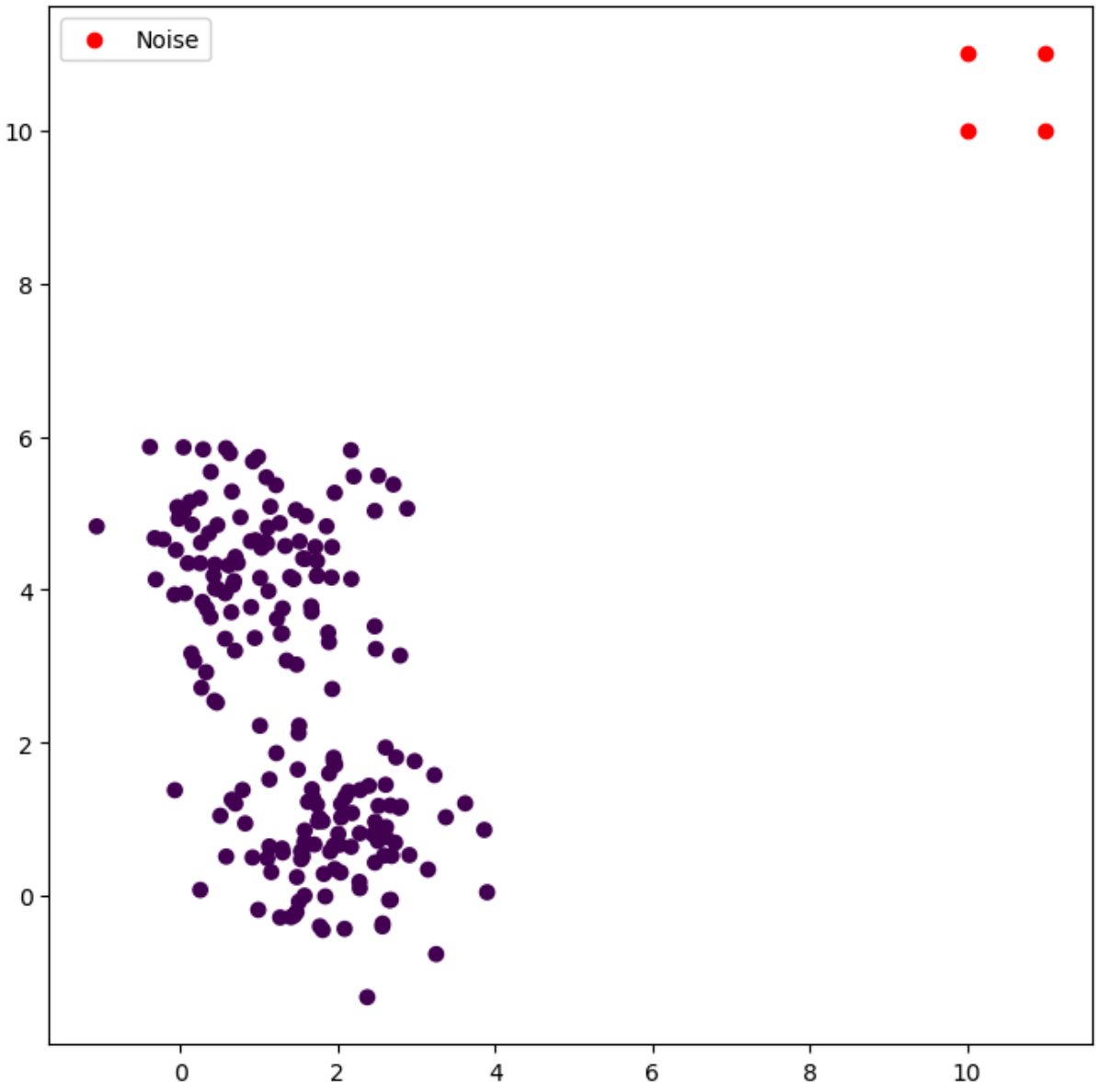
```
df = pd.DataFrame(X, columns=feature_names)
df['cluster'] = labels

sns.pairplot(df, hue='cluster', vars=feature_names[:4], palette='viridis')
plt.show()
```



In [31]:
```
#Generate noisy blobs using make_blobs and use DBSCAN to identify both clusters and
X, y = make_blobs(n_samples=200, centers=2, n_features=2, random_state=0, cluster_s
X = np.vstack((X, [[10, 10], [10, 11], [11, 10], [11, 11]])) # Adding some noise po

dbscan = DBSCAN(eps=1, min_samples=10)
dbscan.fit(X)
labels = dbscan.labels_

noise_points = X[labels == -1]
cluster_points = X[labels != -1]

print(f"Number of clusters: {len(np.unique(labels)) - (1 if -1 in labels else 0)}")
print(f"Number of noise points: {len(noise_points)}")
print(f"Number of cluster points: {len(cluster_points)}")
```

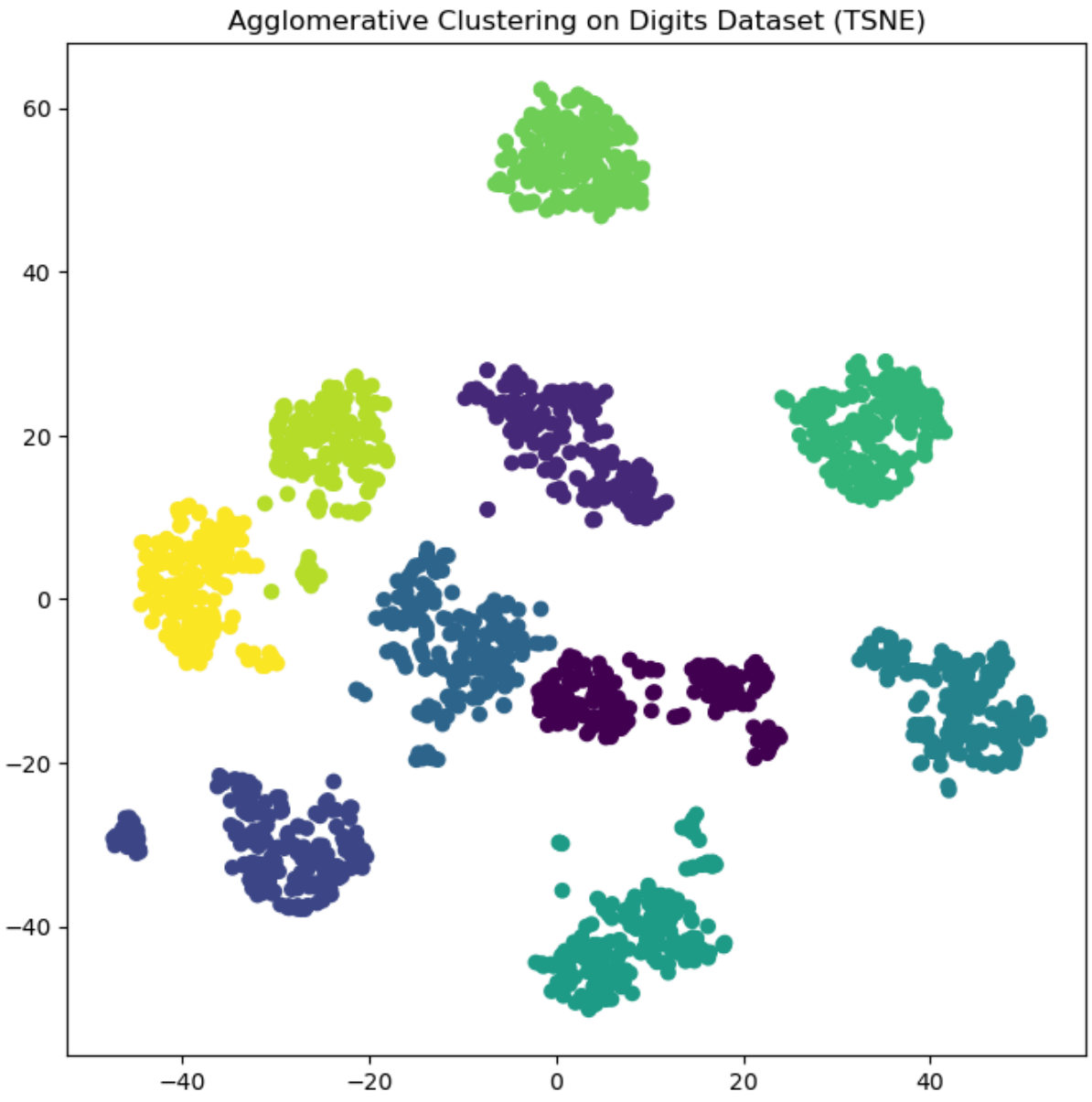
```
plt.figure(figsize=(8, 8))
plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=labels[labels != -1], cma
plt.scatter(noise_points[:, 0], noise_points[:, 1], c='red', label='Noise')
plt.legend()
plt.show()
```

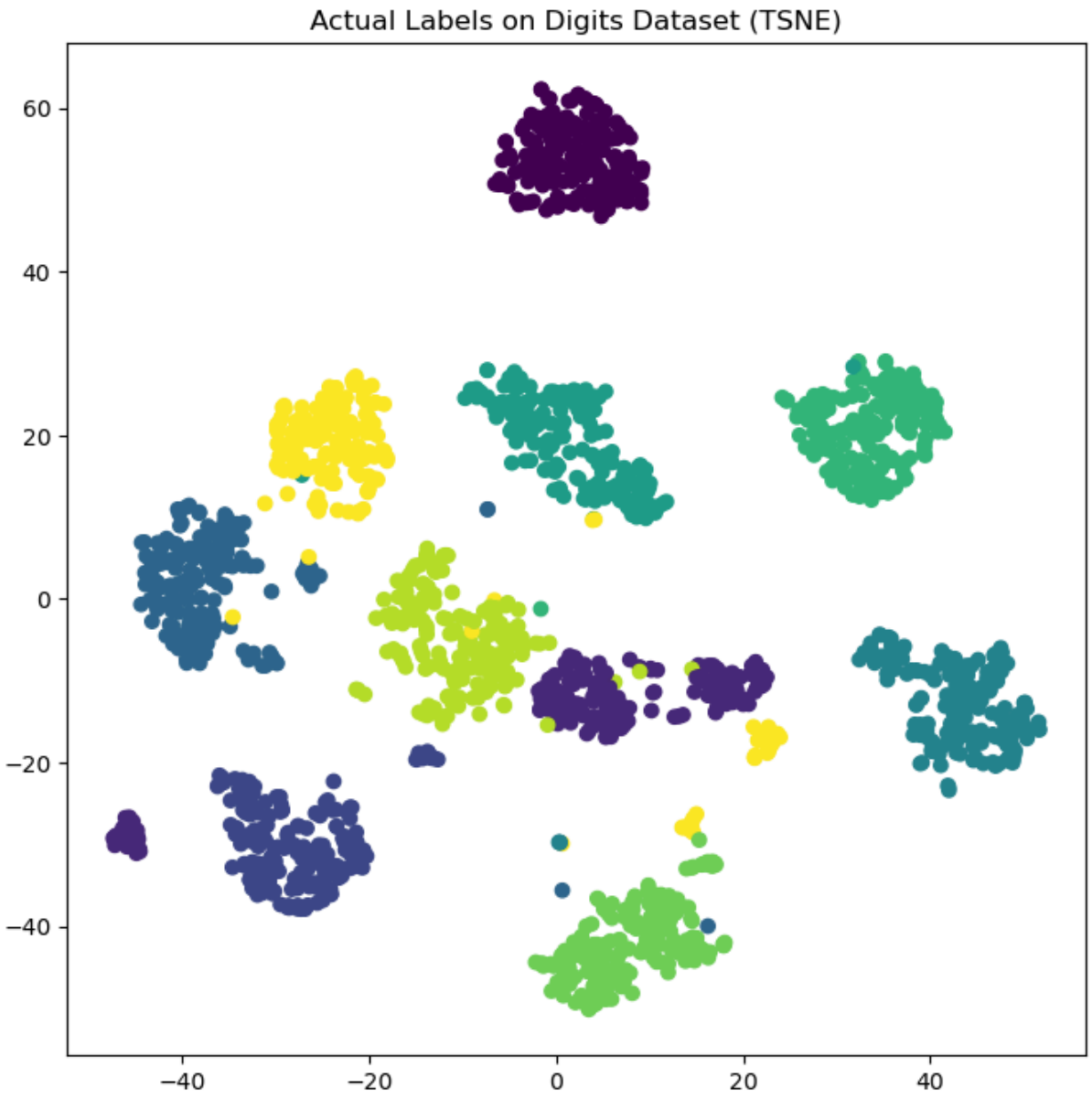
```
Number of clusters: 1
Number of noise points: 4
Number of cluster points: 200
```



In [39]:
```
#Load the Digits dataset, reduce dimensions using t-SNE, then apply Agglomerative C

from sklearn.manifold import TSNE


digits = load_digits()
X = digits.data
y = digits.target

tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X)
```

```
agglo = AgglomerativeClustering(n_clusters=10)
agglo.fit(X_tsne)
labels = agglo.labels_

plt.figure(figsize=(8, 8))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=labels, cmap='viridis')
plt.title('Agglomerative Clustering on Digits Dataset (TSNE)')
plt.show()

plt.figure(figsize=(8, 8))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis')
plt.title('Actual Labels on Digits Dataset (TSNE)')
plt.show()
```




Actual Labels on Digits Dataset (TSNE)
60
40
20
0
−20
−40
−40
−20
0
20
40

In [ ]:

In [ ]: