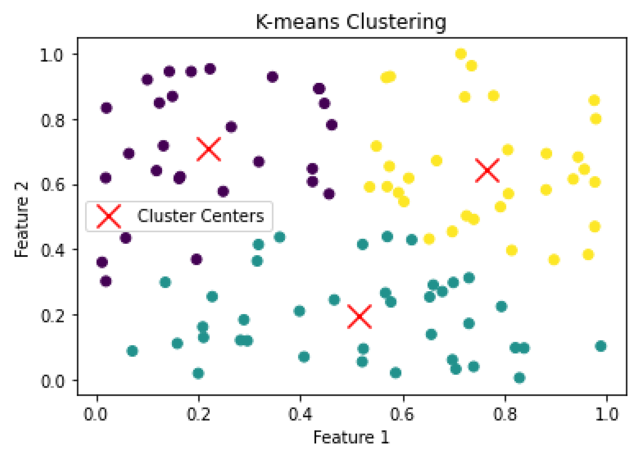
In [1]:

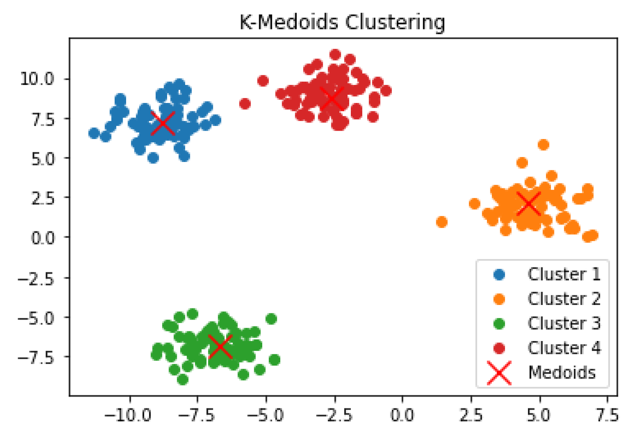
```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

# Sample data generation (replace with your data)
np.random.seed(0)
X = np.random.rand(100, 2)  # 100 points in 2 dimensions

# Initialize KMeans
kmeans = KMeans(n_clusters=3, random_state=0)

# Fit and predict clusters
clusters = kmeans.fit_predict(X)

# Visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], marker='x', c='red', s=200, label='Cluster Centers')
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



K-medoid Clustering

In [4]:

```python
import numpy as np
from sklearn.metrics import pairwise_distances
from random import sample

# Function to compute total cost (sum of distances) for a set of medoids
def compute_cost(X, medoids, clusters):
    cost = 0
    for medoid, cluster in zip(medoids, clusters):
        cost += np.sum(pairwise_distances(X[cluster], X[medoid].reshape(1, -1)))
    return cost

# K-medoids clustering using Partitioning Around Medoids (PAM)
def k_medoids(X, k, max_iter=300):
    m, n = X.shape
    # Randomly initialize medoids
    medoids = sample(range(m), k)

    for iteration in range(max_iter):
        clusters = [[] for _ in range(k)]
        # Assign each point to the nearest medoid
        for idx, point in enumerate(X):
            distances = [np.linalg.norm(point - X[medoid]) for medoid in medoids]
            closest_medoid = np.argmin(distances)
            clusters[closest_medoid].append(idx)

        new_medoids = []
        # Update medoids for each cluster
        for cluster in clusters:
            if len(cluster) == 0:
                continue
            distances_sum = np.sum(pairwise_distances(X[cluster], X[cluster]), axis=1)
            new_medoid = cluster[np.argmin(distances_sum)]
            new_medoids.append(new_medoid)

        # Check for convergence
        if set(medoids) == set(new_medoids):
            break

        medoids = new_medoids

    # Final cluster assignment
    final_clusters = [[] for _ in range(k)]
    for idx, point in enumerate(X):
        distances = [np.linalg.norm(point - X[medoid]) for medoid in medoids]
        closest_medoid = np.argmin(distances)
        final_clusters[closest_medoid].append(idx)

    # Compute final cost
    final_cost = compute_cost(X, medoids, final_clusters)

    return medoids, final_clusters, final_cost

# Example usage:
if __name__ == "__main__":
    from sklearn.datasets import make_blobs
    import matplotlib.pyplot as plt

    # Create sample data
    X, y = make_blobs(n_samples=300, centers=4, random_state=42)

    # Perform K-medoids clustering
    k = 4
    medoids, clusters, cost = k_medoids(X, k)

    # Plot the clusters and medoids
    for i, cluster in enumerate(clusters):
        plt.scatter(X[cluster, 0], X[cluster, 1], label=f'Cluster {i+1}')
    plt.scatter(X[medoids, 0], X[medoids, 1], s=200, c='red', label='Medoids', marker='x')
    plt.legend()
    plt.title('K-Medoids Clustering')
    plt.show()
```



K mean ++

In [4]:

```python
from sklearn.cluster import KMeans
import numpy as np

# Sample data: 2D points
X = np.array([
    [1, 2],
    [1, 4],
    [1, 0],
    [10, 2],
    [10, 4],
    [10, 0]
])

# Initialize KMeans with k-means++ initialization
k = 2
kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
kmeans.fit(X)

print("Cluster Centers:")
print(kmeans.cluster_centers_)

print("Labels:")
print(kmeans.labels_)
```

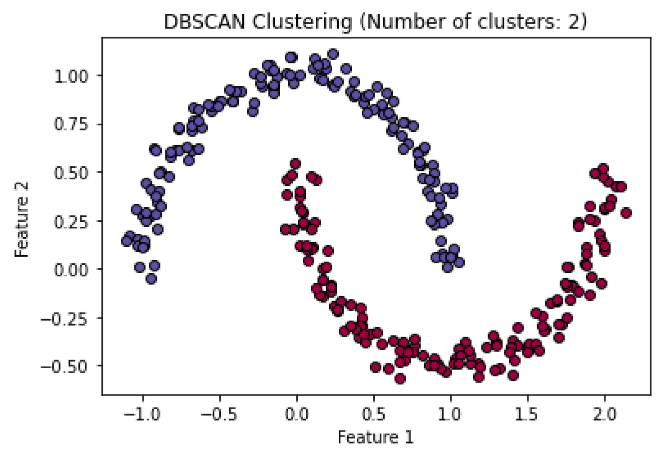
```
Cluster Centers:
[[10.  2.]
 [ 1.  2.]]
Labels:
[1 1 1 0 0 0]
```

DBSCAN using scikit-learn

In [6]:

```python
# Import necessary libraries
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
import numpy as np

# Generate sample data
X, y = make_moons(n_samples=300, noise=0.05, random_state=42)

# Initialize DBSCAN with parameters
dbscan = DBSCAN(eps=0.2, min_samples=5)

# Fit the model
dbscan.fit(X)

# Extract labels (-1 indicates noise)
labels = dbscan.labels_

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

# Plotting the results
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]

for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black color for noise
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title(f'DBSCAN Clustering (Number of clusters: {n_clusters})')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
Number of clusters found: 2
```



In [ ]: