```
In [1]: 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()
```
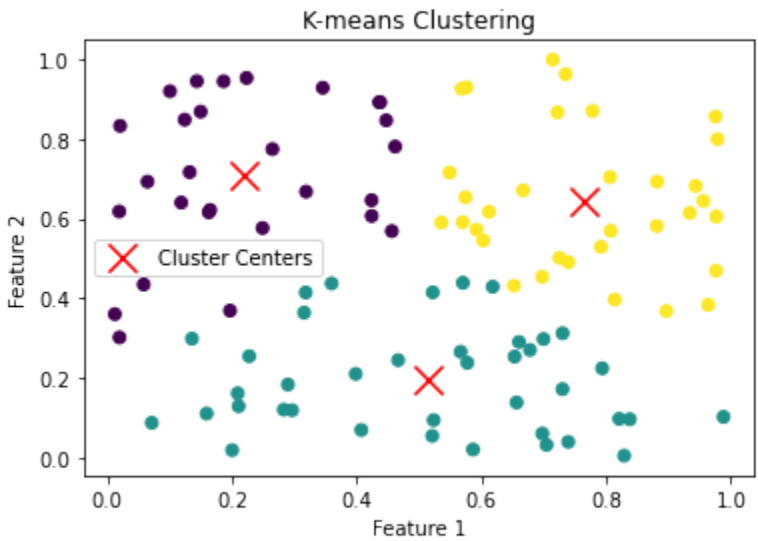


## K-medoid Clustering

```
In [4]:  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()
```



```
In [ ]:
```