K-Means vs K-Medians on a Simple Dataset

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `numpy` for numerical operations, `matplotlib` for data visualisation, and `sklearn` for perform K-Means and K-Medians clustering.

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

2. Defining the dataset

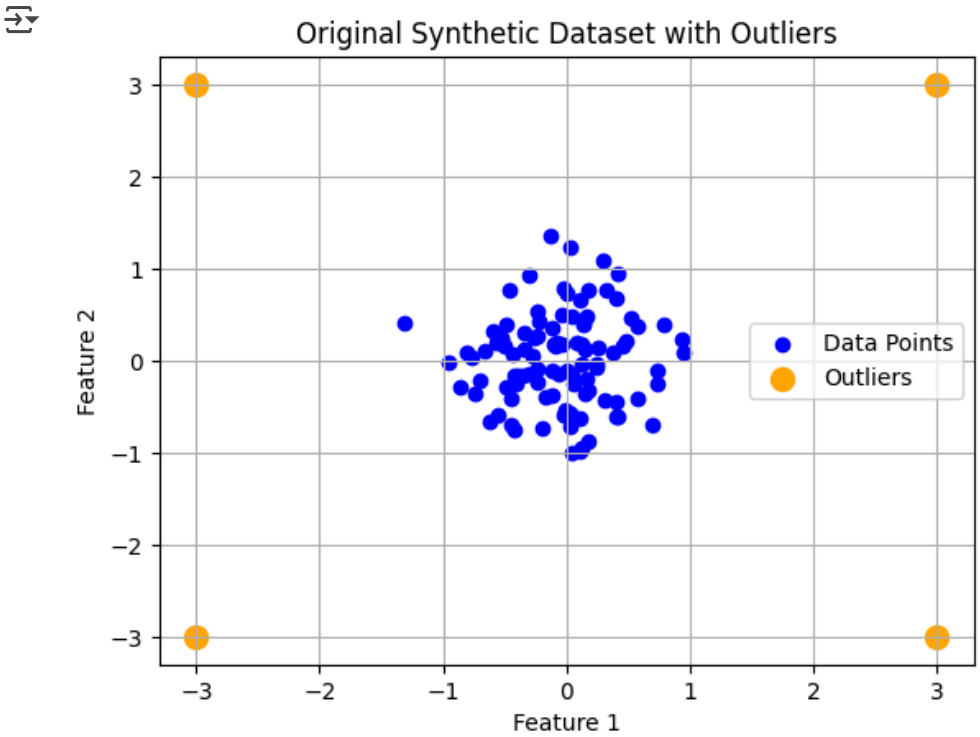
Here, we generate synthetic data consisting of 100 normal data points clustered around the origin. We also introduce four outliers located far from the main cluster. Combining both gives us the dataset `X`.

```
np.random.seed(42)
data = np.random.randn(100, 2) * 0.5
outliers = np.array([[3, 3], [3, -3], [-3, 3], [-3, -3]])
X = np.vstack((data, outliers))
```

We also create a scatterplot to visualise these data points, displaying their initial distribution.

```
def plot_original_data(X, outliers):
    plt.scatter(X[:, 0], X[:, 1], color='blue', label='Data Points')
    plt.scatter(outliers[:, 0], outliers[:, 1], color='orange', label='Outliers', s=100)
    plt.title('Original Synthetic Dataset with Outliers')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_original_data(X, outliers)
```



3. Defining the plotting function

In this section, we define a function to take the dataset, cluster labels, and cluster centres to create a scatterplot. It visually distinguishes between different clusters and highlights the centres.

```
def plot_clusters(X, labels, centers, title):
    plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
    plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.grid(True)
    plt.show()
```

4. Applying K-means clustering

Now, we instantiate the K-means algorithm with two clusters and fit it to our data `X`. We then retrieve the predicted labels and the cluster centres.

```
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X)
kmeans_centres = kmeans.cluster_centers_
```

5. Applying K-medoids clustering

Next, similar to K-means, we apply K-medians clustering (using K-medoids) to the dataset. The labels and centres are retrieved for comparison.

```
kmedoids = KMedoids(n_clusters=2, random_state=42)
kmedoids_labels = kmedoids.fit_predict(X)
kmedoids_centres = kmedoids.cluster_centers_
```

6. Visualising the results

Finally, we plot the clustering results side by side for K-means and K-medians. This visual comparison allows us to observe how K-medians is more robust to outliers, as it will likely label the outliers more accurately compared to K-means.

```
def plot_clustering_results(X, kmeans_labels, kmeans_centers, kmedoids_labels, kmedoids_centers):
    plt.figure(figsize=(12,6))

    # K-means clustering plot
    plt.subplot(1, 2, 1)
    plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, s=50, cmap='viridis', alpha=0.6)
    plt.scatter(kmeans_centers[:, 0], kmeans_centers[:, 1], c='red', s=200, alpha=0.75, marker='X', label='Centroids')
    plt.title('K-means Clustering', fontsize=16)
    plt.xlabel('Feature 1', fontsize=14)
    plt.ylabel('Feature 2', fontsize=14)
    plt.grid(True)
    plt.legend()
    plt.axis('equal')

    # K-medians clustering plot
    plt.subplot(1, 2, 2)
    plt.scatter(X[:, 0], X[:, 1], c=kmedoids_labels, s=50, cmap='viridis', alpha=0.6)
    plt.scatter(kmedoids_centers[:, 0], kmedoids_centers[:, 1], c='red', s=200, alpha=0.75, marker='X', label='Centroids'
    plt.title('K-medians Clustering (K-medoids)', fontsize=16)
    plt.xlabel('Feature 1', fontsize=14)
    plt.ylabel('Feature 2', fontsize=14)
    plt.grid(True)
    plt.legend()
    plt.axis('equal')
```

```
    plt.tight_layout()
    plt.show()

plot_clustering_results(X, kmeans_labels, kmeans_centres, kmedoids_labels, kmedoids_centres)
```