EXPERIMENT – 04
Implement a K Means Clustering to classify the given dataset

1. Aim:
Implement a K Means Clustering to classify the given data set.

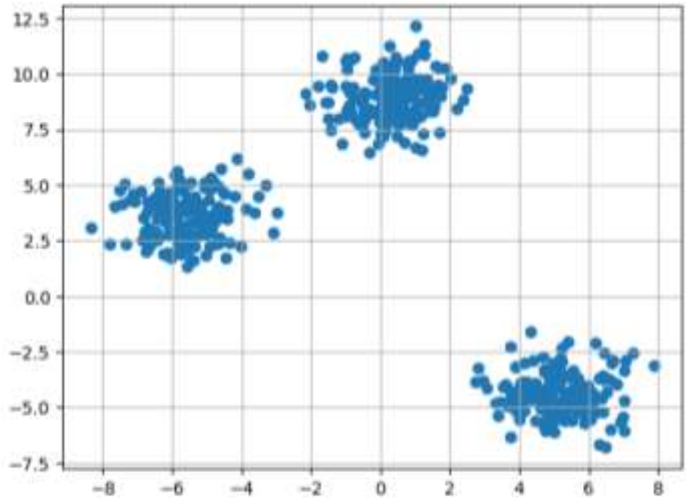
2. Objectives:
- To understand the working of the K-means clustering algorithm.
- To cluster data points based on similarity.
- To visualize the clusters and interpret the results.

3. Brief Theory:
K-Means clustering is the most popular unsupervised machine learning algorithm. K-Means clustering is used to find intrinsic groups within the unlabelled dataset and draw inferences from them. The K-means algorithm identifies a certain number of centroids within a data set, a centroid being the arithmetic mean of all the data points belonging to a particular cluster. The algorithm then allocates every data point to the nearest cluster as it attempts to keep the clusters as small as possible (the ‘means’ in K-means refers to the task of averaging the data or finding the centroid). At the same time, K-means attempts to keep the other clusters as different as possible.
K-means is an iterative procedure that
- Starts by guessing the initial centroids, and then
- Refines this guess by
    - Repeatedly assigning examples to their closest centroids, and then
    - Re computing the centroids based on the assignments.

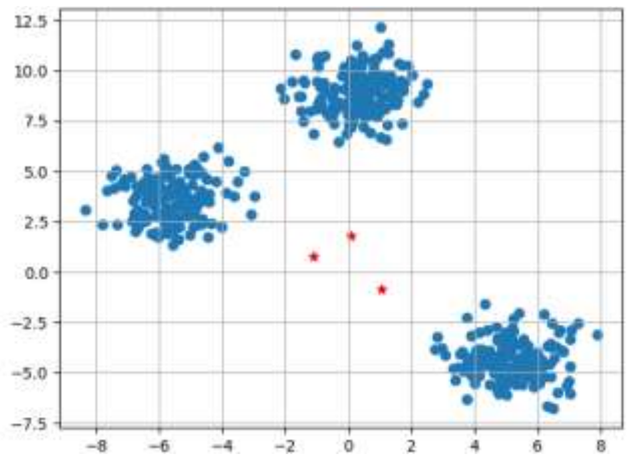
4. Hints
1. Create or Load the Dataset
- You can either create synthetic data using make blobs or use a real-world dataset like the Iris dataset.
- Choose the number of clusters (K) and apply the K-means algorithm.
- Visualize the Clusters and centroids for your data.
- Interpret the Results: Measure the within-cluster variation to evaluate the quality of clustering.
- Use Elbow Method: To find the optimal number of clusters.

5. Simple example

```
#random dataset
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X,y = make_blobs(n_samples = 500,n_features = 2,centers = 3,random_state = 23)
fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0],X[:,1])
plt.show()
```



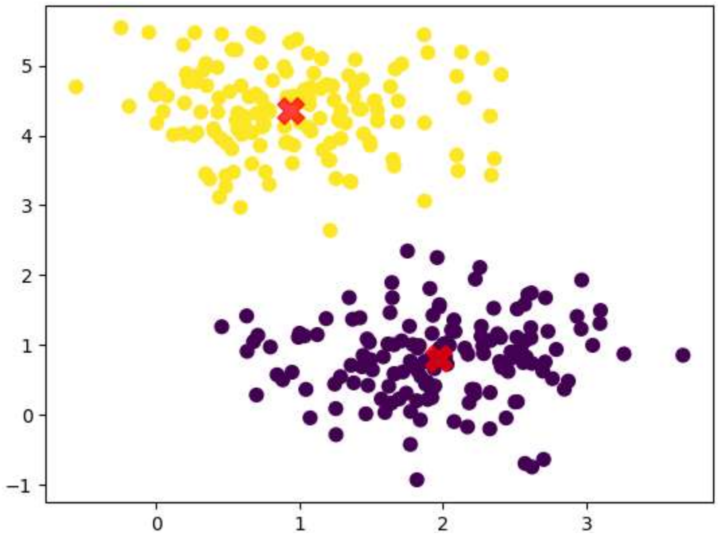
```
#Using centers
k = 3
clusters = {}
np.random.seed(23)
for idx in range(k):
    center = 2*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }
    clusters[idx] = cluster
clusters
plt.scatter(X[:,0],X[:,1])
plt.grid(True)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '*',c = 'red')
plt.show()
```



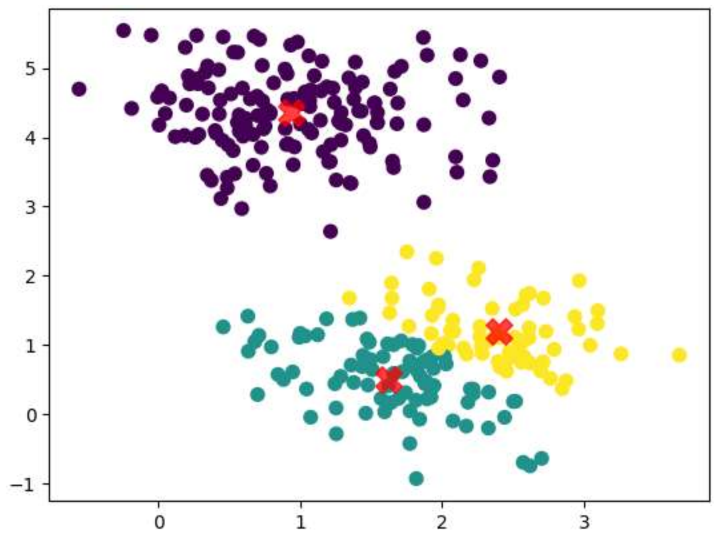
```
# Different values of k
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Generate sample data
X, y = make_blobs(n_samples=300, centers=2, cluster_std=0.60, random_state=0)
# Fit K-Means model
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
# Predict cluster labels
y_kmeans = kmeans.predict(X)
# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

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

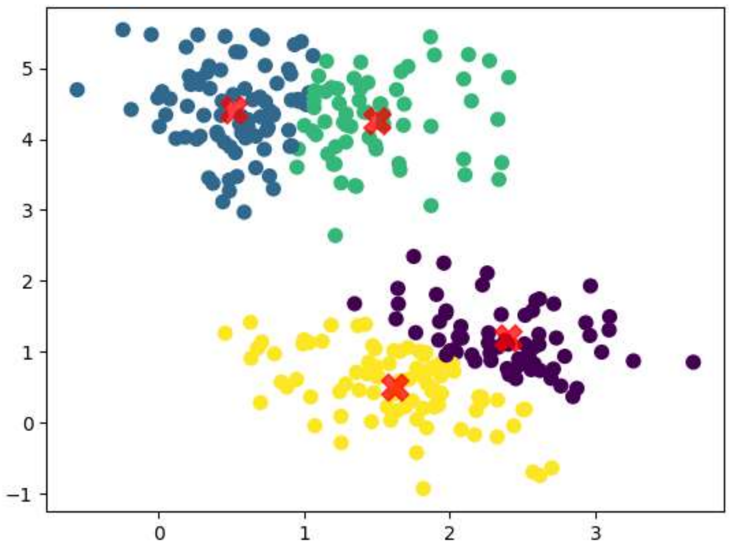
K = 2



K =3



K = 4



6. **Implementation of application (T-Shirt size prediction):**

Python code:

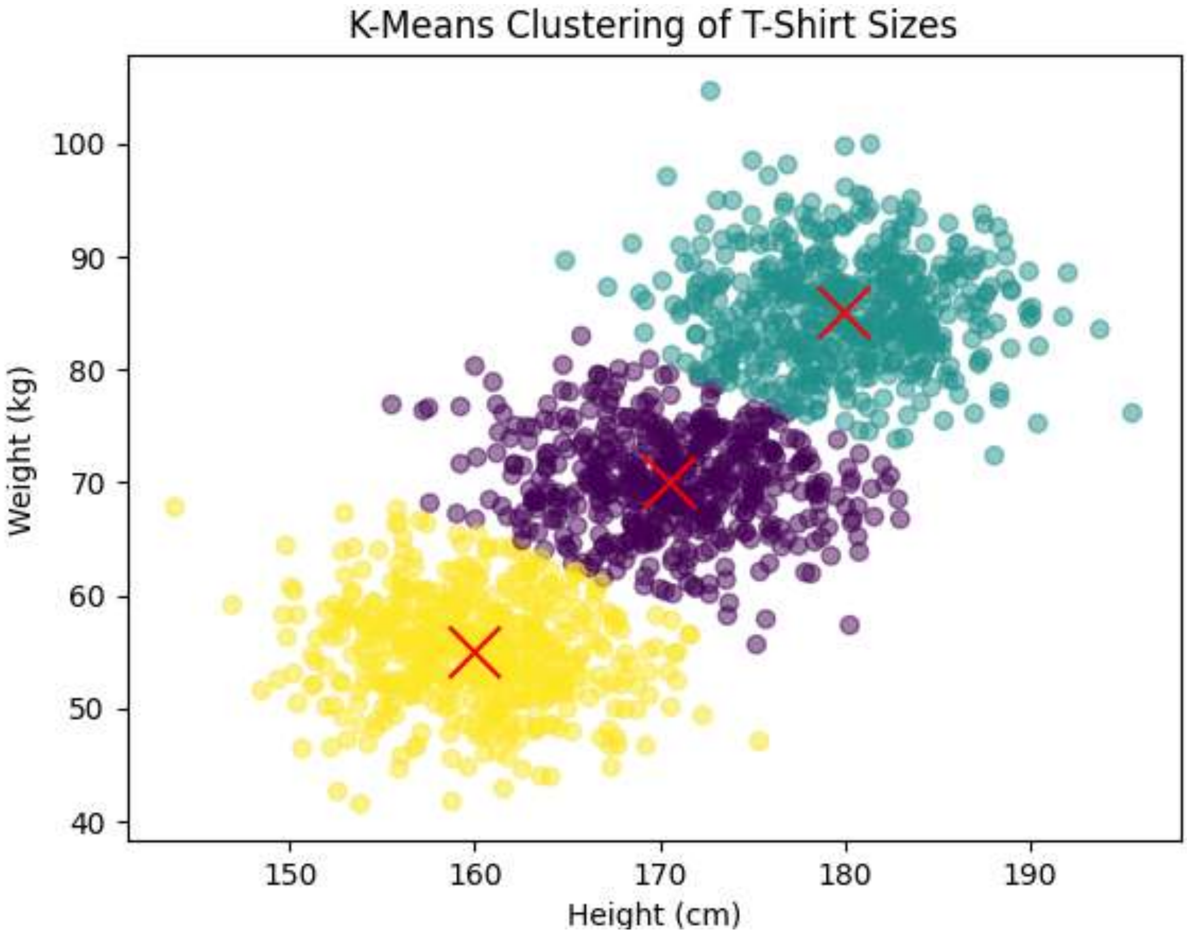
```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Generate synthetic data for Height and Weight for three clusters
# Cluster 1: Short height, light weight
height_cluster1 = np.random.normal(160, 5, 500)  # Mean 160 cm, std deviation 5
weight_cluster1 = np.random.normal(55, 5, 500)   # Mean 55 kg, std deviation 5

# Cluster 2: Average height, medium weight
height_cluster2 = np.random.normal(170, 5, 500)  # Mean 170 cm, std deviation 5
weight_cluster2 = np.random.normal(70, 5, 500)   # Mean 70 kg, std deviation 5

# Cluster 3: Tall height, heavy weight
height_cluster3 = np.random.normal(180, 5, 500)  # Mean 180 cm, std deviation 5
weight_cluster3 = np.random.normal(85, 5, 500)   # Mean 85 kg, std deviation 5

# Combine the data into a single dataset
height = np.concatenate([height_cluster1, height_cluster2, height_cluster3])
weight = np.concatenate([weight_cluster1, weight_cluster2, weight_cluster3])

# Create a DataFrame
data = pd.DataFrame({'Height': height, 'Weight': weight})

# Perform K-Means clustering
```

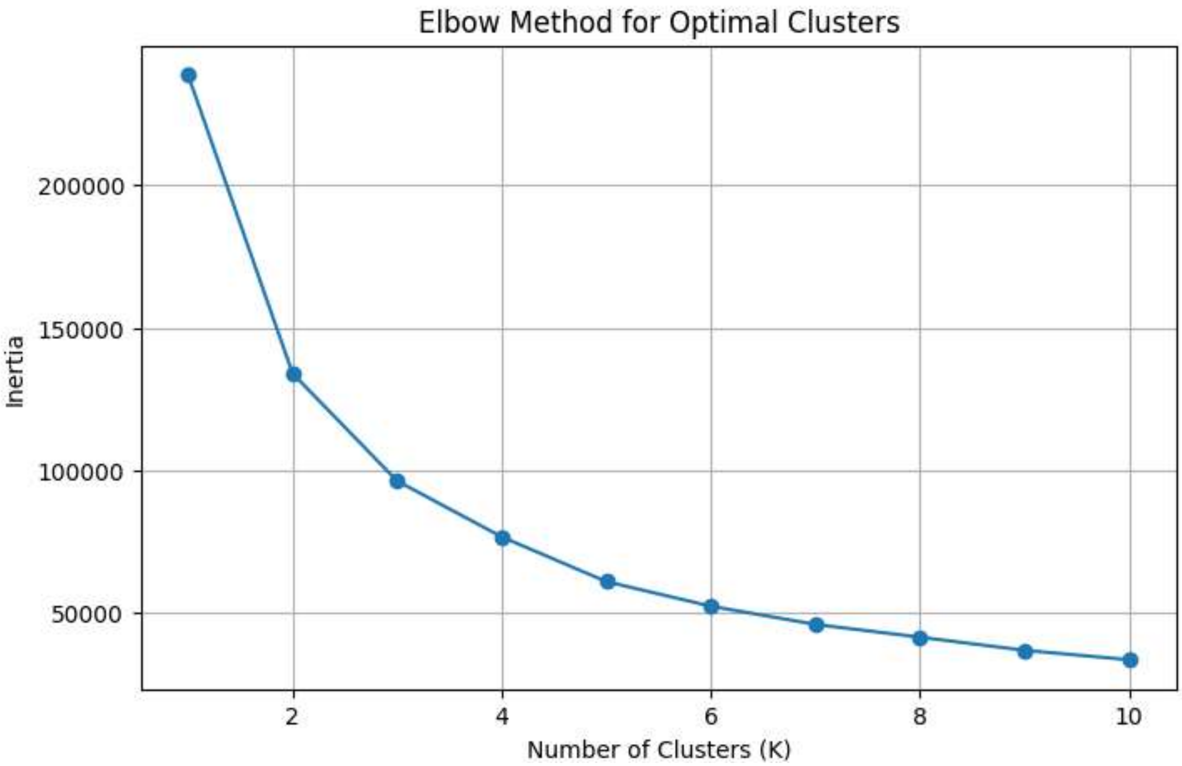
```
kmeans = KMeans(n_clusters=3, random_state=42)
data['Cluster'] = kmeans.fit_predict(data[['Height', 'Weight']])

# Get centroids of the clusters
centroids = kmeans.cluster_centers_

# Print first few rows of the dataset and cluster centroids
print(data.head())
print("\nCluster centroids:\n", centroids)

# Plotting the clusters
plt.scatter(data['Height'], data['Weight'], c=data['Cluster'], cmap='viridis', alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='red', marker='x')  # Plot centroids
plt.title('K-Means Clustering of T-Shirt Sizes')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

7. **Result:**

Cluster centroids:

```
 [[170.5392409   70.07077026]
 [179.96117178  85.16807605]
 [159.94558614  55.08115412]]
```





8. Conclusion:

In conclusion, this experiment successfully implemented the K-means clustering algorithm to classify a dataset into distinct clusters. By choosing different values of k (the number of clusters), we observed how data points were grouped based on proximity to the calculated centroids. The experiment demonstrated how K-means operates by iteratively updating cluster centroids and reassigning data points, ensuring the final clusters are well-defined. The practical application of K-means in predicting T-shirt sizes based on height and weight provided a real-world scenario for the algorithm's use. Clustering identified three distinct groups representing different body types, highlighting K-means' effectiveness in unsupervised learning tasks like customer segmentation or image compression.

In summary, K-means is a powerful tool for partitioning datasets without labelled outcomes. The experiment's results, visualizations, and code implementations validated its efficiency in pattern recognition and classification, offering insights into real-world applications.

9. Link to the code uploaded on: https://github.com/Niti0209/K-Means-clustering.git

10. List of Reference used for implementation:

- www.kaggle.com
- www.geeksforgeeks.com