k-means Algorithm

1.Intialise the number of clusters (K).

2.randomly choose 'k' object from data set and assign its cluster centre

3.Assign / Reassign the data points to the cluster using similarity distance measure (Eucledian measure)

4.Find new cluster centre (updating cluster centre by taking means of all datapoints) 5.Repeat 3 and 4 till the stop criteria

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets


iris = datasets.load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Next steps: Generate code with iris_df | View recommended plots | New interactive sheet

```
iris_df.shape
```

```
(150, 4)
```

```
iris_df.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Optimum number of clusters of k-means algorithm

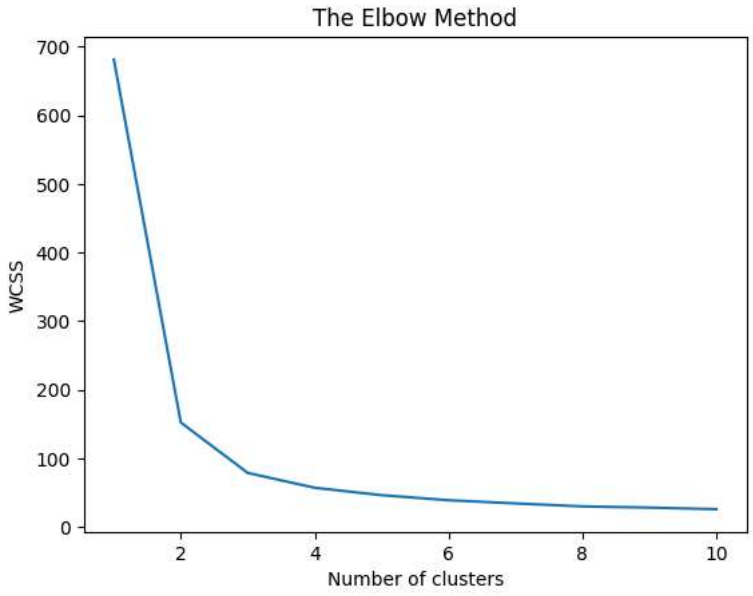
```
x = iris_df.iloc[:,[0,1,2,3]].values
from sklearn.cluster import KMeans
wcss = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
```

Applying k-means to the dataset

```
plt.plot(range(1,11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
```

```
plt.ylabel('WCSS')
plt.show()
```



Double-click (or enter) to edit

+ Code + Text

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(x)
```

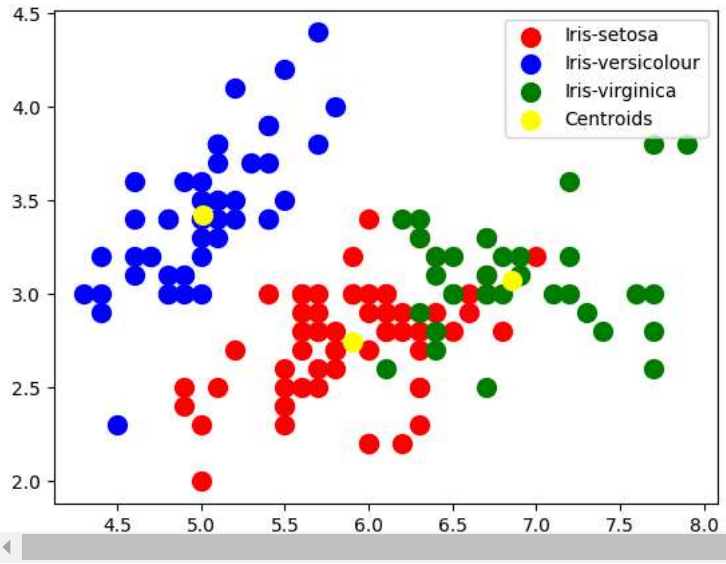
Visualising the cluster

```
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1],s = 100, c = 'green', label = 'Iris-virginica')

# plotting the centroids of the clusters

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ed2c4b207d0>
```




```
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster # Import linkage, dendrogram, and fcluster


# Load the Iris dataset
```

```
iris = load_iris()
X = iris.data
y = iris.target

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

# Perform hierarchical/agglomerative clustering
Z = linkage(X_scaled, method='ward')  # 'ward' minimizes the variance of clusters

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z, labels=iris.target_names[y])
plt.title('Hierarchical Clustering Dendrogram (Iris dataset)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```