Prediction using Unsupervised ML @Sandeep

From the given 'Iris' dataset, predict the optimum number of clusters and represent it visually.

Implementation using Python

Importing the libraries

In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn import datasets
#KMeans class from the sklearn library.
from sklearn.cluster import KMeans
```

In [2]:

```
# Importing the dataset
iris = datasets.load_iris()
iris_DF = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_DF
```

Out[2]:

	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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [3]:

```
print("Data Shape :",iris_DF.shape)
```

```
Data Shape : (150, 4)
```

In [4]:

```
print("Columns : ", iris_DF.columns)
```

```
Columns :  Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

In [5]:

```
iris_DF.describe() #Generate descriptive statistics
```

Out[5]:

	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

In [6]:

```
iris_DF.info() #information about a DataFrame including the index dtype and column dtype
s, non-null values and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
dtypes: float64(4)
memory usage: 4.8 KB
```

In [7]:

```
iris_DF.index
```

Out[7]:

```
RangeIndex(start=0, stop=150, step=1)
```

In [8]:

```
X = iris_DF.iloc[:,:].values # Using iloc() fuction , store required data frame values i
nto X variable
```

Implement KMeans with k=5

In [9]:

```
#implement k-means clustering using k=5 (arbitrarily)
#instantiate the KMeans class and assign it to the variable kmeans5
# We are going to use the fit predict method that returns for each #observation which clu
ster it belongs to.
#The cluster to which client belongs and it will return this cluster numbers into a
#single vector that is  called y K-means
kmeans5=KMeans(n_clusters=5)
y_kmeans5=kmeans5.fit_predict(X)
print(y_kmeans5)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 4 1 1 1 4 1 4 4 1 4 1 4 1 1 4 1 4 1 4 1 1
 1 1 1 1 1 4 4 4 4 1 4 1 1 1 4 4 4 1 4 4 4 4 4 1 4 4 2 1 3 2 2 3 4 3 2 3 2
 2 2 1 2 2 2 3 3 1 2 1 3 1 2 3 1 1 2 3 3 3 2 1 1 3 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
```

In [10]:

```
kmeans5.cluster_centers_ #display the means or the averages of the points
```

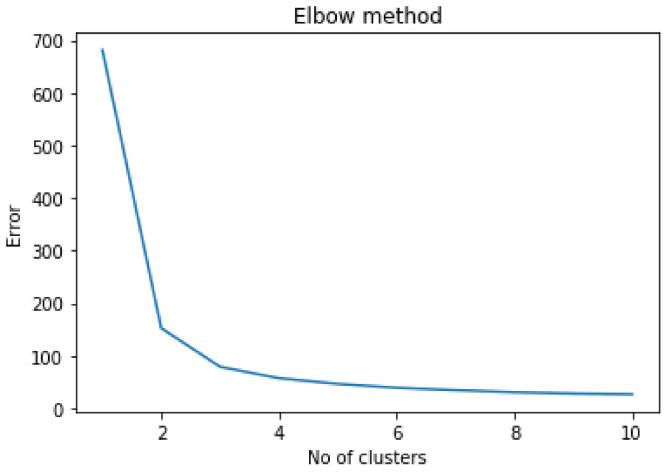
Out[10]:

```
array([[5.006     , 3.428     , 1.462     , 0.246     ],
       [6.20769231, 2.85384615, 4.74615385, 1.56410256],
       [6.52916667, 3.05833333, 5.50833333, 2.1625    ],
       [7.475     , 3.125     , 6.3       , 2.05      ],
       [5.508     , 2.6       , 3.908     , 1.204     ]])
```

Apply Elbow method to find optimal number of clustres in the datasets

In [11]:

```
Error =[]

for i in range(1, 11):

    kmeans = KMeans(n_clusters = i).fit(X)

    kmeans.fit(X)

    Error.append(kmeans.inertia_)

import matplotlib.pyplot as plt

plt.plot(range(1, 11), Error)

plt.title('Elbow method')

plt.xlabel('No of clusters')

plt.ylabel('Error')

plt.show()
```



As We can see in the grapth that the optimal value of k is between 2 and 4, as the elbow-like shape is formed at k=3 in the above graph. So Let's implement k-means again using k=3

Implement K-Means with k=3

In [12]:

```
#implement k-means clustering using k=3 (arbitrarily)
#instantiate the KMeans class and assign it to the variable kmeans3
kmeans3=KMeans(n_clusters=3)
y_kmeans3 = kmeans3.fit_predict(X)
```

```
print(y_kmeans3)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0
 2 0]
```

In [13]:

```
kmeans3.cluster_centers_ #display the means or the averages of the points
```

Out[13]:

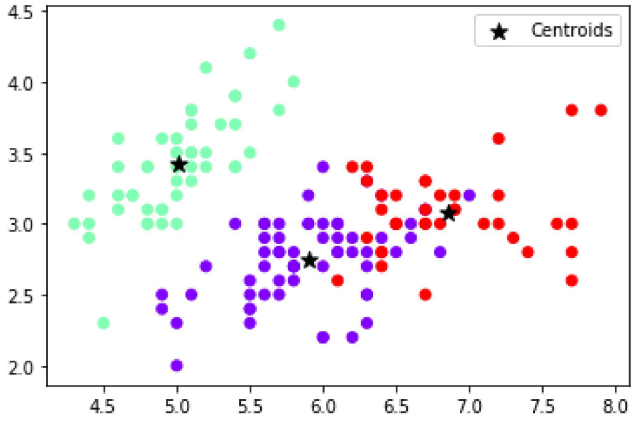
```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

visualize the three clusters

In [14]:

```
#Plot the centroid. This time we're going to use the cluster centres
#attribute that returns here the coordinates of the centroid.
plt.scatter(X[:,0],X[:,1], c=y_kmeans3, cmap='rainbow')
plt.scatter(kmeans3.cluster_centers_[:, 0], kmeans3.cluster_centers_[:,1], marker='*',
            s = 100, c = 'black', label = 'Centroids')
plt.legend()
print("Visulizing k-Menas clustering")
```

```
Visulizing k-Menas clustering
```



So we predicted the optimum number of clusters is = 3

Thanks

In [ ]: