Task 2: Prediction using Unsupervised ML

**Predict the optimum number of clusters and represent it visually Name: Praveen Kumar G**

# Importing necessay libraries

In [72]:

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

**%matplotlib** inline

# Loading data in DataFrame

In [73]:

df **=** pd**.**read\_csv("Iris.csv", index\_col **=** 0) df**.**head()

Out[73]: **SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Id** |  | | | | |
| **1** | 5.1 | 3.5 | 1.4 | 0.2 Iris-setosa |  |
| **2** | 4.9 | 3.0 | 1.4 | 0.2 Iris-setosa |  |
| **3** | 4.7 | 3.2 | 1.3 | 0.2 Iris-setosa |  |
| **4** | 4.6 | 3.1 | 1.5 | 0.2 Iris-setosa |  |
| **5** | 5.0 | 3.6 | 1.4 | 0.2 Iris-setosa |  |
| In [74]: | df**.**shape |  |  |  |  |  |
| Out[74]: | (150, 5) |  |  |  |  |  |
| In [75]: | df**.**info() |  |  |  |  |  |

<class 'pandas.core.frame.DataFrame'> Int64Index: 150 entries, 1 to 150 Data columns (total 5 columns):

# Column Non-Null Count Dtype

1. SepalLengthCm 150 non-null float64
2. SepalWidthCm 150 non-null float64
3. PetalLengthCm 150 non-null float64
4. PetalWidthCm 150 non-null float64
5. Species 150 non-null object dtypes: float64(4), object(1)

memory usage: 7.0+ KB

In [76]:

df**.**describe()

Out[76]:

In [77]:

**SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **count** | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| **mean** | 5.843333 | 3.054000 | 3.758667 | 1.198667 |
| **std** | 0.828066 | 0.433594 | 1.764420 | 0.763161 |
| **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 |

# First we need to find the optimum number of clusters for K-Means. Here we will use The Elbow Method to determine the value of k in K-Means.

**The Elbow Method**

**In Elbow method we calculate the**  Within-Cluster-Sum of Squared Errors (WCSS) **for different values of k, and choose the k for which WCSS becomes first starts to diminish. In the plot of**  WCSS-versus-k **, this is visible as an elbow.**

x **=** df**.**iloc[:, :4]**.**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\_)

pd**.**DataFrame({"Number of Clusters":range(1,11),"WCSS":wcss})

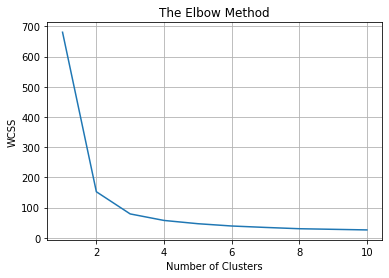
|  |  |  |  |
| --- | --- | --- | --- |
| Out[77]: |  | **Number of Clusters** | **WCSS** |
|  | **0** | 1 | 680.824400 |
|  | **1** | 2 | 152.368706 |
|  | **2** | 3 | 78.940841 |
|  | **3** | 4 | 57.345409 |
|  | **4** | 5 | 46.535582 |
|  | **5** | 6 | 38.938740 |
|  | **6** | 7 | 34.190688 |
|  | **7** | 8 | 29.905374 |
|  | **8** | 9 | 27.927882 |
|  | **9** | 10 | 25.955497 |

## Plotting Number of Clusters vs. WCSS

In [78]:

plt**.**plot(range(1,11), wcss) plt**.**title("The Elbow Method") plt**.**xlabel("Number of Clusters") plt**.**ylabel("WCSS")

plt**.**grid() plt**.**show()



## As expected, the plot looks like an arm with a clear elbow at k = 3. Applying k-means to the dataset with Number of Clusters as k = 3

In [79]:

kmeans **=** KMeans(n\_clusters **=** 3, init **=** 'k-means++',

max\_iter **=** 300, n\_init **=** 10, random\_state **=** 0) y\_kmeans **=** kmeans**.**fit\_predict(x)

# Visualizing the clusters on the first two columns

In [82]:

plt**.**figure(figsize**=**[10,8])

plt**.**scatter(x[y\_kmeans **==** 0,0], x[y\_kmeans **==** 0,1],

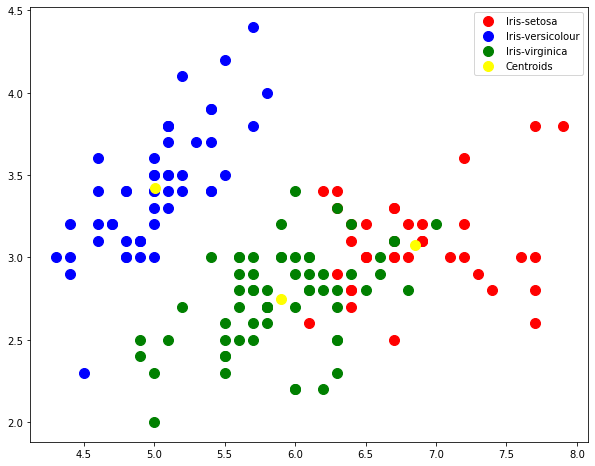
s **=** 100, c **=** "red", label **=** 'Iris-setosa') plt**.**scatter(x[y\_kmeans **==** 1, 0], x[y\_kmeans **==** 1, 1],

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')

plt**.**scatter(kmeans**.**cluster\_centers\_[:, 0], kmeans**.**cluster\_centers\_[:,1], s **=** 100, c **=** 'yellow', label **=** 'Centroids')

plt**.**legend() plt**.**show()



# Visualizing the clusters on the first three columns

In [84]:

plt**.**figure(figsize**=**[10,10])

ax **=** plt**.**axes(projection **=**"3d")

ax**.**scatter3D(x[y\_kmeans **==** 0, 0], x[y\_kmeans **==** 0, 1], x[y\_kmeans **==** 0, 2], s **=** 50, c **=** "red", label **=** 'Iris-setosa')

ax**.**scatter3D(x[y\_kmeans **==** 1, 0], x[y\_kmeans **==** 1, 1], x[y\_kmeans **==** 1, 2], s **=** 50, c **=** 'blue', label **=** 'Iris-versicolour')

ax**.**scatter3D(x[y\_kmeans **==** 2, 0], x[y\_kmeans **==** 2, 1], x[y\_kmeans **==** 2, 2], s **=** 50, c **=** 'green', label **=** 'Iris-virginica')

ax**.**scatter(kmeans**.**cluster\_centers\_[:, 0], kmeans**.**cluster\_centers\_[:,1], kmeans**.**cluster\_centers\_[:,2], s **=** 50, c **=** 'yellow', label **=** 'Centroids', alpha **=** 0.8)

plt**.**legend() plt**.**show()

