THE SPARKS FOUNDATION

DATA SCIENCE AND BUSINESS ANALYTICS INTERNSHIP

AUTHOR: MEGHA

TASK 2- Prediction using Unsupervised ML

In this task we will predict the optimum number of clusters from given 'Iris' dataset using K means clustering algorithm and represent it visually.

1. Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Reading Dataset

In [2]:
```
iris = pd.read_csv('Iris.csv')
print('data imported')
```

```
data imported
```

In [3]:
```
#determining the shape of dataset
iris.shape
```

Out[3]:
```
(150, 6)
```

In [4]:
```
#first 5 values in the datset
iris.head()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [5]:

```
#last 5 values in the dataset
iris.tail()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [6]:

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 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), int64(1), object(1)
memory usage: 7.2+ KB
```

In [7]:

```
iris.describe()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

3. Finding the optimum number of clusters for K Means

In [8]:
```
#consists of attribute(inputs)
iris.iloc[:, [0, 1, 2, 3]].values
```

Out[8]:
```
array([[  1. ,   5.1,   3.5,   1.4],
       [  2. ,   4.9,   3. ,   1.4],
       [  3. ,   4.7,   3.2,   1.3],
       [  4. ,   4.6,   3.1,   1.5],
       [  5. ,   5. ,   3.6,   1.4],
       [  6. ,   5.4,   3.9,   1.7],
       [  7. ,   4.6,   3.4,   1.4],
       [  8. ,   5. ,   3.4,   1.5],
       [  9. ,   4.4,   2.9,   1.4],
       [ 10. ,   4.9,   3.1,   1.5],
       [ 11. ,   5.4,   3.7,   1.5],
       [ 12. ,   4.8,   3.4,   1.6],
       [ 13. ,   4.8,   3. ,   1.4],
       [ 14. ,   4.3,   3. ,   1.1],
       [ 15. ,   5.8,   4. ,   1.2],
       [ 16. ,   5.7,   4.4,   1.5],
       [ 17. ,   5.4,   3.9,   1.3],
       [ 18. ,   5.1,   3.5,   1.4],
       [ 19. ,   5.7,   3.8,   1.7],
       [ 20. ,   5.1,   3.8,   1.5],
       [ 21. ,   5.4,   3.4,   1.7],
       [ 22. ,   5.1,   3.7,   1.5],
```

```
[ 23. ,   4.6,   3.6,   1. ],
[ 24. ,   5.1,   3.3,   1.7],
[ 25. ,   4.8,   3.4,   1.9],
[ 26. ,   5. ,   3. ,   1.6],
[ 27. ,   5. ,   3.4,   1.6],
[ 28. ,   5.2,   3.5,   1.5],
[ 29. ,   5.2,   3.4,   1.4],
[ 30. ,   4.7,   3.2,   1.6],
[ 31. ,   4.8,   3.1,   1.6],
[ 32. ,   5.4,   3.4,   1.5],
[ 33. ,   5.2,   4.1,   1.5],
[ 34. ,   5.5,   4.2,   1.4],
[ 35. ,   4.9,   3.1,   1.5],
[ 36. ,   5. ,   3.2,   1.2],
[ 37. ,   5.5,   3.5,   1.3],
[ 38. ,   4.9,   3.1,   1.5],
[ 39. ,   4.4,   3. ,   1.3],
[ 40. ,   5.1,   3.4,   1.5],
[ 41. ,   5. ,   3.5,   1.3],
[ 42. ,   4.5,   2.3,   1.3],
[ 43. ,   4.4,   3.2,   1.3],
[ 44. ,   5. ,   3.5,   1.6],
[ 45. ,   5.1,   3.8,   1.9],
[ 46. ,   4.8,   3. ,   1.4],
[ 47. ,   5.1,   3.8,   1.6],
[ 48. ,   4.6,   3.2,   1.4],
[ 49. ,   5.3,   3.7,   1.5],
[ 50. ,   5. ,   3.3,   1.4],
[ 51. ,   7. ,   3.2,   4.7],
[ 52. ,   6.4,   3.2,   4.5],
[ 53. ,   6.9,   3.1,   4.9],
[ 54. ,   5.5,   2.3,   4. ],
[ 55. ,   6.5,   2.8,   4.6],
[ 56. ,   5.7,   2.8,   4.5],
[ 57. ,   6.3,   3.3,   4.7],
[ 58. ,   4.9,   2.4,   3.3],
[ 59. ,   6.6,   2.9,   4.6],
[ 60. ,   5.2,   2.7,   3.9],
[ 61. ,   5. ,   2. ,   3.5],
[ 62. ,   5.9,   3. ,   4.2],
[ 63. ,   6. ,   2.2,   4. ],
[ 64. ,   6.1,   2.9,   4.7],
[ 65. ,   5.6,   2.9,   3.6],
[ 66. ,   6.7,   3.1,   4.4],
[ 67. ,   5.6,   3. ,   4.5],
```

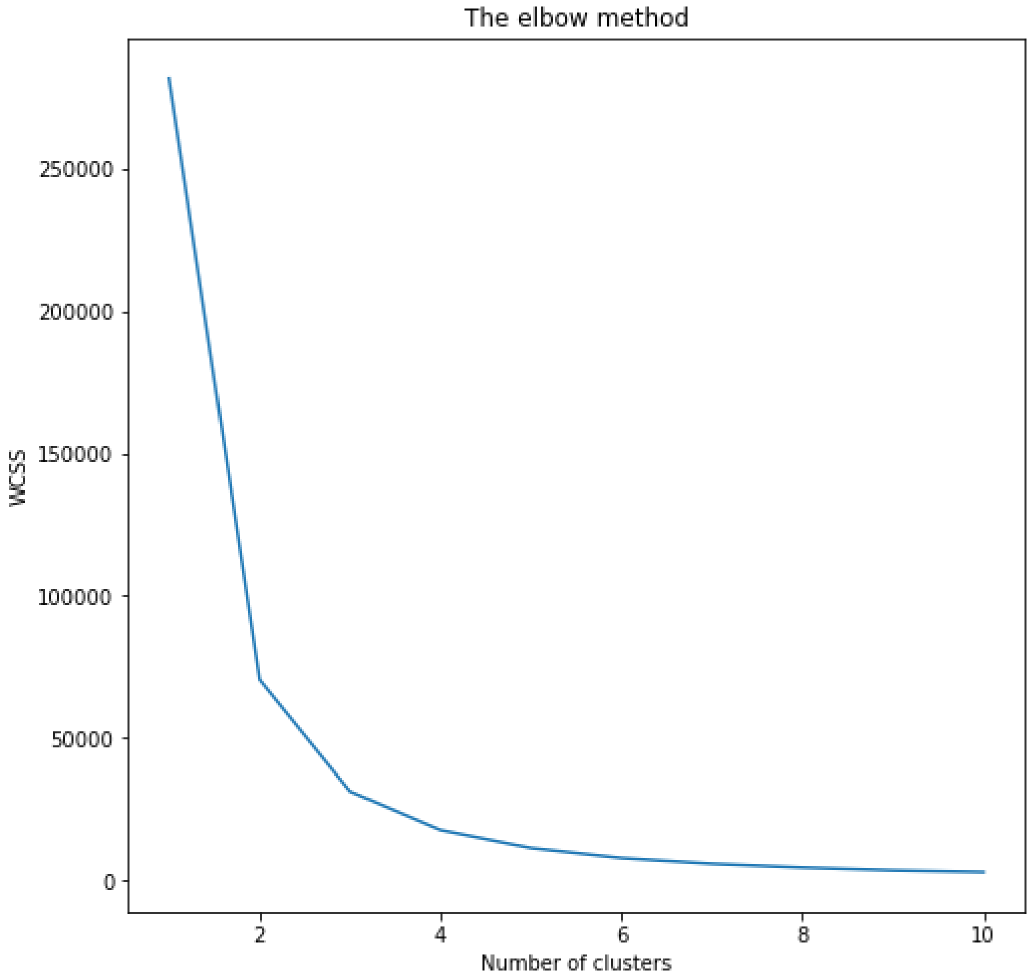
```
[ 68. ,   5.8,   2.7,   4.1],
[ 69. ,   6.2,   2.2,   4.5],
[ 70. ,   5.6,   2.5,   3.9],
[ 71. ,   5.9,   3.2,   4.8],
[ 72. ,   6.1,   2.8,   4. ],
[ 73. ,   6.3,   2.5,   4.9],
[ 74. ,   6.1,   2.8,   4.7],
[ 75. ,   6.4,   2.9,   4.3],
[ 76. ,   6.6,   3. ,   4.4],
[ 77. ,   6.8,   2.8,   4.8],
[ 78. ,   6.7,   3. ,   5. ],
[ 79. ,   6. ,   2.9,   4.5],
[ 80. ,   5.7,   2.6,   3.5],
[ 81. ,   5.5,   2.4,   3.8],
[ 82. ,   5.5,   2.4,   3.7],
[ 83. ,   5.8,   2.7,   3.9],
[ 84. ,   6. ,   2.7,   5.1],
[ 85. ,   5.4,   3. ,   4.5],
[ 86. ,   6. ,   3.4,   4.5],
[ 87. ,   6.7,   3.1,   4.7],
[ 88. ,   6.3,   2.3,   4.4],
[ 89. ,   5.6,   3. ,   4.1],
[ 90. ,   5.5,   2.5,   4. ],
[ 91. ,   5.5,   2.6,   4.4],
[ 92. ,   6.1,   3. ,   4.6],
[ 93. ,   5.8,   2.6,   4. ],
[ 94. ,   5. ,   2.3,   3.3],
[ 95. ,   5.6,   2.7,   4.2],
[ 96. ,   5.7,   3. ,   4.2],
[ 97. ,   5.7,   2.9,   4.2],
[ 98. ,   6.2,   2.9,   4.3],
[ 99. ,   5.1,   2.5,   3. ],
[100. ,   5.7,   2.8,   4.1],
[101. ,   6.3,   3.3,   6. ],
[102. ,   5.8,   2.7,   5.1],
[103. ,   7.1,   3. ,   5.9],
[104. ,   6.3,   2.9,   5.6],
[105. ,   6.5,   3. ,   5.8],
[106. ,   7.6,   3. ,   6.6],
[107. ,   4.9,   2.5,   4.5],
[108. ,   7.3,   2.9,   6.3],
[109. ,   6.7,   2.5,   5.8],
[110. ,   7.2,   3.6,   6.1],
[111. ,   6.5,   3.2,   5.1],
[112. ,   6.4,   2.7,   5.3],
```

```
       [113. ,   6.8,   3. ,   5.5],
       [114. ,   5.7,   2.5,   5. ],
       [115. ,   5.8,   2.8,   5.1],
       [116. ,   6.4,   3.2,   5.3],
       [117. ,   6.5,   3. ,   5.5],
       [118. ,   7.7,   3.8,   6.7],
       [119. ,   7.7,   2.6,   6.9],
       [120. ,   6. ,   2.2,   5. ],
       [121. ,   6.9,   3.2,   5.7],
       [122. ,   5.6,   2.8,   4.9],
       [123. ,   7.7,   2.8,   6.7],
       [124. ,   6.3,   2.7,   4.9],
       [125. ,   6.7,   3.3,   5.7],
       [126. ,   7.2,   3.2,   6. ],
       [127. ,   6.2,   2.8,   4.8],
       [128. ,   6.1,   3. ,   4.9],
       [129. ,   6.4,   2.8,   5.6],
       [130. ,   7.2,   3. ,   5.8],
       [131. ,   7.4,   2.8,   6.1],
       [132. ,   7.9,   3.8,   6.4],
       [133. ,   6.4,   2.8,   5.6],
       [134. ,   6.3,   2.8,   5.1],
       [135. ,   6.1,   2.6,   5.6],
       [136. ,   7.7,   3. ,   6.1],
       [137. ,   6.3,   3.4,   5.6],
       [138. ,   6.4,   3.1,   5.5],
       [139. ,   6. ,   3. ,   4.8],
       [140. ,   6.9,   3.1,   5.4],
       [141. ,   6.7,   3.1,   5.6],
       [142. ,   6.9,   3.1,   5.1],
       [143. ,   5.8,   2.7,   5.1],
       [144. ,   6.8,   3.2,   5.9],
       [145. ,   6.7,   3.3,   5.7],
       [146. ,   6.7,   3. ,   5.2],
       [147. ,   6.3,   2.5,   5. ],
       [148. ,   6.5,   3. ,   5.2],
       [149. ,   6.2,   3.4,   5.4],
       [150. ,   5.9,   3. ,   5.1]])
```

In [18]:

```
# Finding the optimum number of clusters for k-means classification

x = iris.iloc[:, [0, 1, 2, 3]].values

from sklearn.cluster import KMeans
```

```python
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_)

# Plotting the results onto a line graph,
# `allowing us to observe 'The elbow'
plt.figure(figsize=(8,8))
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') # Within cluster sum of squares
plt.show()
```



This is a graph which will help us to findout number of clusters

You can clearly see why it is called 'The elbow method' from the above graph, the optimum clusters is where the elbow occurs. This is when the within cluster sum of squares (WCSS) doesn't decrease significantly with every iteration.

From this we choose the number of clusters as '3'.

Model Testing

In [19]:
```
# Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++',
                max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

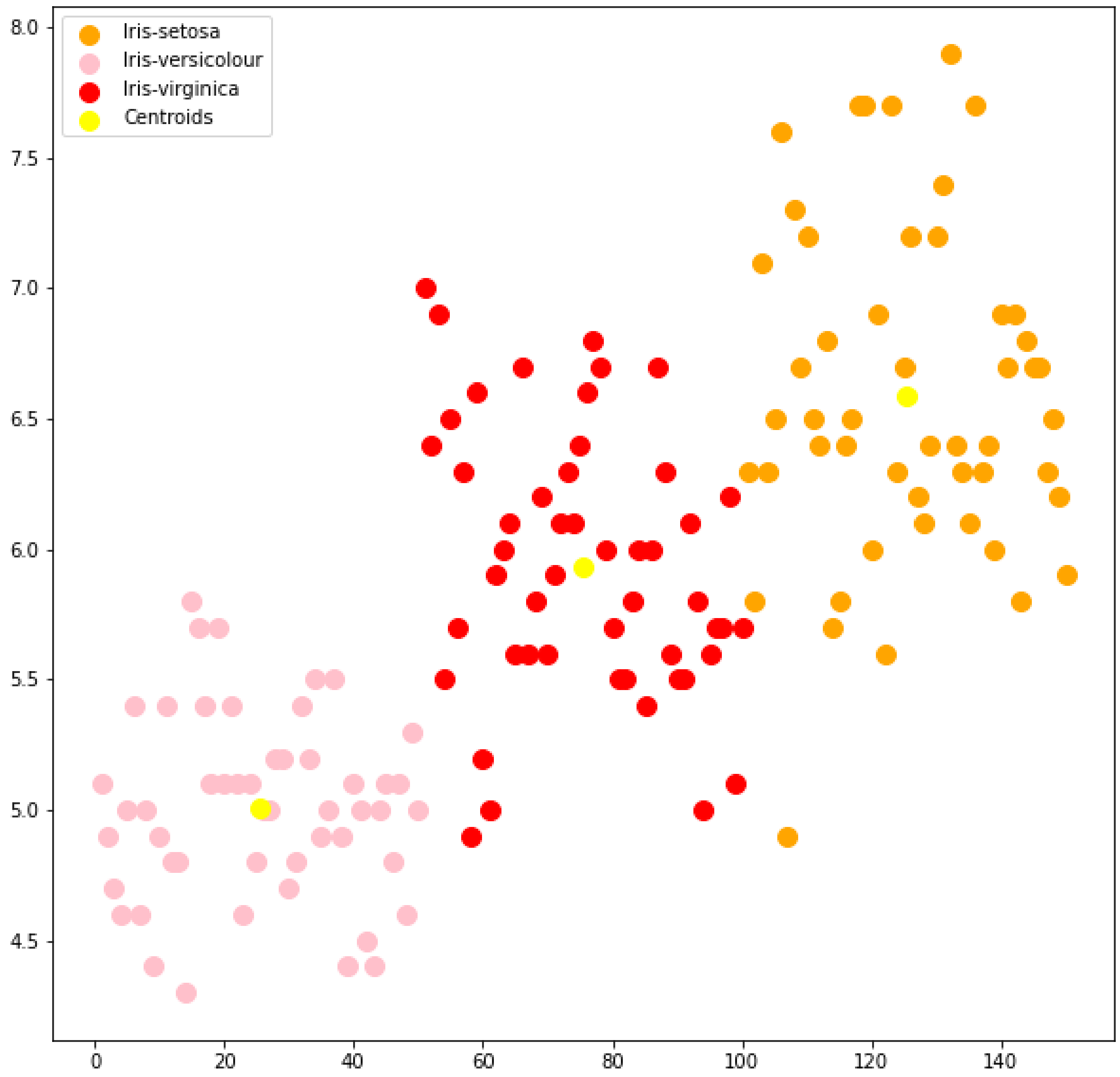
In [20]:
```
y_kmeans
```

Out[20]:
```
array([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, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 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, 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])
```

Visualisation of Clusters

In [21]:
```
# Visualising the clusters - On the first two columns
plt.figure(figsize=(10,10))
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1],
            s = 100, c = 'orange', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1],
            s = 100, c = 'pink', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1],
            s = 100, c = 'red', 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()
```

Out[21]:
```
<matplotlib.legend.Legend at 0x2a507542250>
```



This is a graphic way to represent our distribution