GRIP : The Spark Foundation

Data Science and Business Analytics Intern

Author : Nilima Nemade

TASK I : Prediction Using Unsupervised ML

Dataset Used:Iris Dataset,Which is available in sklearn library.

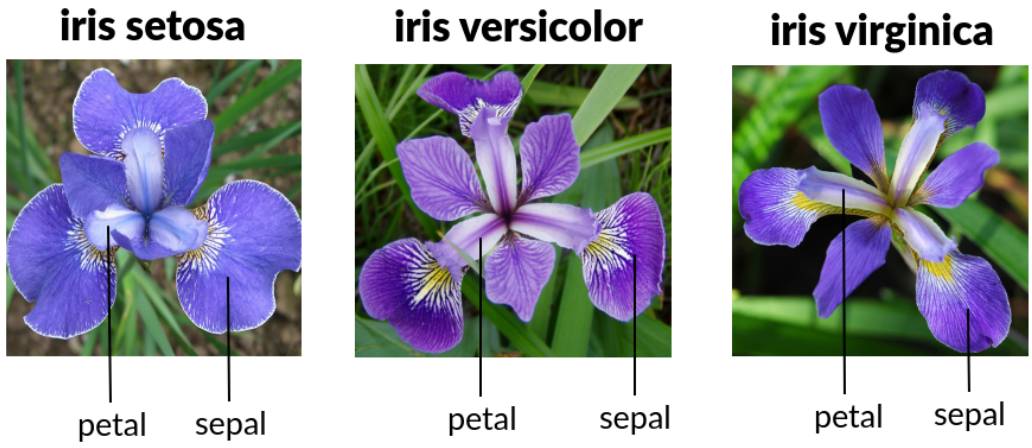
Alternatively,it can be downloaded through the following link:

problem Statement: Predict the optimum number of clusters and represent it visually.

In [4]:

```
from PIL import Image
imag=Image.open('irispic.png')
imag
```

Out[4]:



In [5]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Import Libraries and Load Dataset First, we need to import libraries: pandas (loading dataset), numpy (matrix manipulation), matplotlib and seaborn (visualization), and sklearn (building classifiers).

In [6]:

```
iris=pd.read_csv("Iris_dataset.csv")
iris
```

Out[6]:

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

150 rows × 6 columns

To load the dataset, we can use the read_csv function from pandas

In [7]:

```
iris.head()
```

Out[7]:

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

These are the first 5 rows of data.

In [8]:

```
iris.tail()
```

Out[8]:

	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

These are the last 5 rows of data.

In [10]:

```
iris.isnull().sum()
```

Out[10]:

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

Dataset is completely balanced data. No null value present in data set.

In [11]:

```
iris.shape
```

Out[11]:

```
(150, 6)
```

In [15]:

```
iris['Species'].value_counts()
```

Out[15]:

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

The iris dataset consist of 150 data instances. There are 3 classes Iris Setosa, Iris Versicolor, Iris Virginica. each have 50 instances.

In [16]:

```
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
```

The data is equally balanced we got equal no of counts.

In [18]:

```
iris.describe()
```

Out[18]:

	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

We see all the columns of our data set are free from null values and four of them are numeric while one is a categorical column which also is seen by the dtypes function. Numerical Summary First, we look at a numerical summary of each attribute through describe:

1: K- Means Clustering

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. It is popular for cluster analysis in data mining. k-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, better Euclidean solutions can be found using k-medians and k-medoids.

Finding the optimum number of clusters for K-Means and determining the value of K.

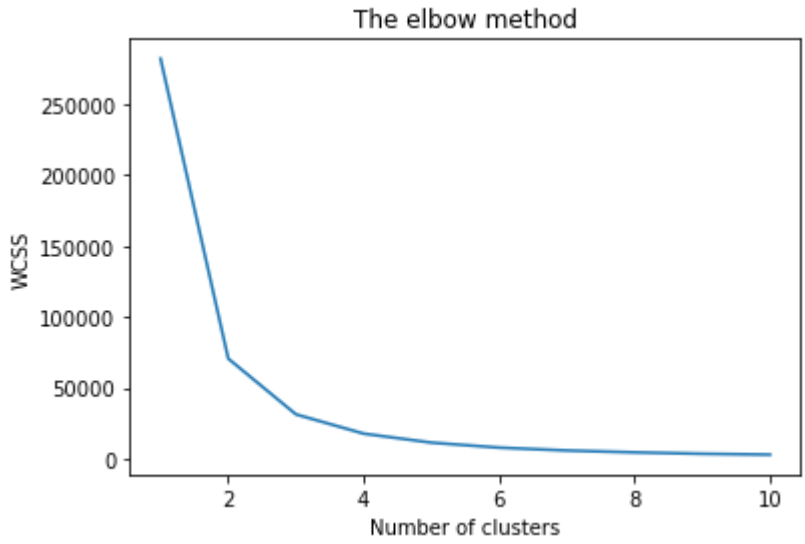
In [23]:

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

x = iris.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_)
# Plotting the results onto a line graph,
# `allowing us to observe 'The elbow'
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()
```



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.

In [24]:

```
# 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 [25]:

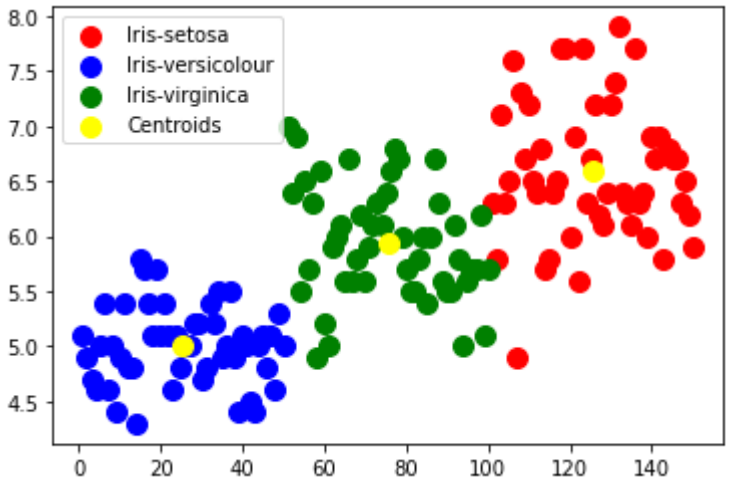
```
# Visualising the clusters - On the first two columns
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1],
            s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1],
            s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 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()
```

Out[25]:

```
<matplotlib.legend.Legend at 0x1b643fc48b0>
```



In [27]:

```
iris["Category"]=y_kmeans
iris
```

Out[27]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	Category
0	1	5.1	3.5	1.4	0.2	Iris-setosa	1
1	2	4.9	3.0	1.4	0.2	Iris-setosa	1
2	3	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5	5.0	3.6	1.4	0.2	Iris-setosa	1
...	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica	0
146	147	6.3	2.5	5.0	1.9	Iris-virginica	0
147	148	6.5	3.0	5.2	2.0	Iris-virginica	0
148	149	6.2	3.4	5.4	2.3	Iris-virginica	0
149	150	5.9	3.0	5.1	1.8	Iris-virginica	0

150 rows × 7 columns

In [ ]:

```

```