

Task 2 - Prediction using Unsupervised ML (K-means Clustering method)

TSF-GRIP Internship

Data Science & Business Analytics Tasks¶

#GRIPAUG21

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

Submitted by: Mohd Tarique Khan

Date: 07/08/2021

```
In [1]: #Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from matplotlib.pyplot import figure

%matplotlib inline
```

```
In [2]: #Reading the Data
url = "https://drive.google.com/file/d/1l1q7YvbW2bt8VXjfm06brx66b10YiwK-/view?usp=sharing"
url2='https://drive.google.com/uc?id=' + url.split('/')[2]
df_iris = pd.read_csv(url2)
```

Exploratory Data Analysis (EDA)

```
In [3]: # Data Visualization
df_iris.head(10)
```

	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
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
In [4]: df_iris.shape
```

```
Out[4]: (150, 6)
```

```
In [5]: df_iris.describe()
```

	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

```
In [6]: df_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
```

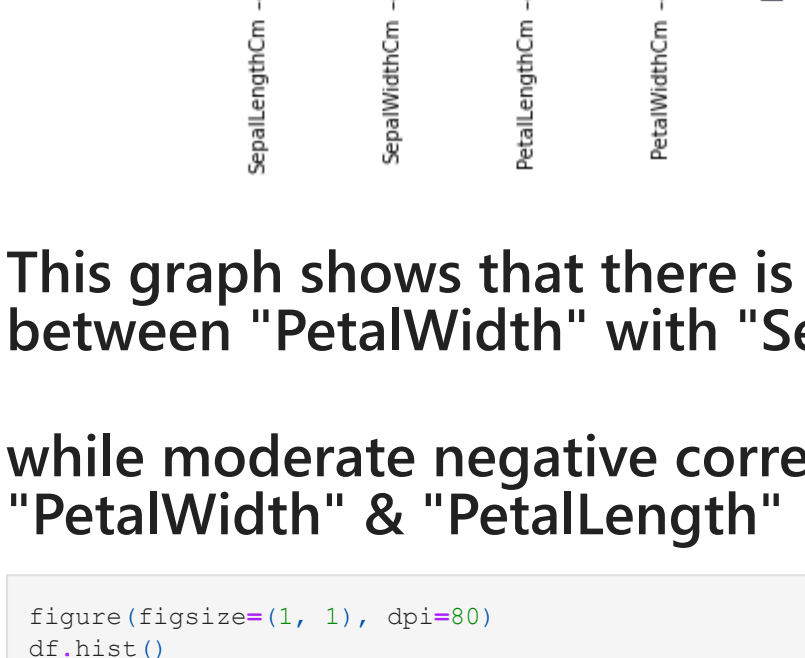
```
In [7]: # Dropping the Id Column from the dataframe as it is just a serial number and is of no use
df=df_iris.drop(["Id"], axis = 1)
df
```

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

150 rows × 5 columns

```
In [8]: # Creating a heatmap to understand the correlation between the variables
sns.heatmap(df.corr(), annot = True, linecolor='white', linewidths=.5)
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x245a12a2160>
```

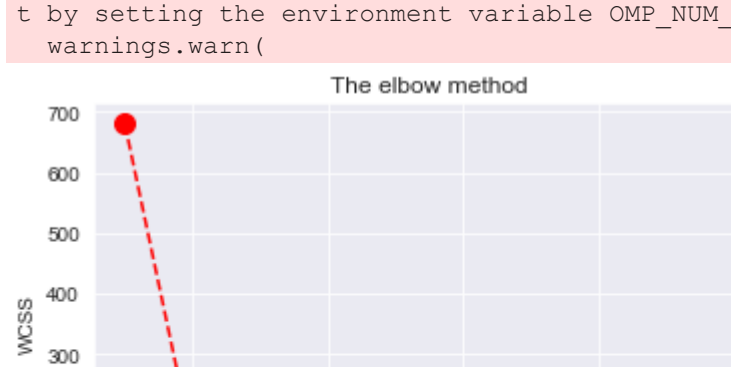


This graph shows that there is a strong positive correlation between "PetalWidth" with "SepalLength" & "PetalLength"

while moderate negative correlation between "SepalWidth" with "PetalWidth" & "PetalLength"

```
In [9]: figure(figsize=(1, 1), dpi=80)
df.hist()
```

```
Out[9]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000245A1426550>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x00000245A144EA00>],
  <matplotlib.axes._subplots.AxesSubplot object at 0x00000245A147CE50>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x00000245A14B52E0>]],
  dtype=object)
<Figure size 80x80 with 0 Axes>
```



Finding optimum numbers of Clusters for Kmeans classification

```
In [10]: # Data preparation for K-means clustering
x = df.iloc[:, [0, 1, 2, 3]].values
from sklearn.cluster import KMeans
wcss = []
```

```
In [11]: # Finding inertia on different k values
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'
sns.set_style('darkgrid')
plt.plot(range(1, 11), wcss, 'r--', marker='o', markersize=10, color='red')
```

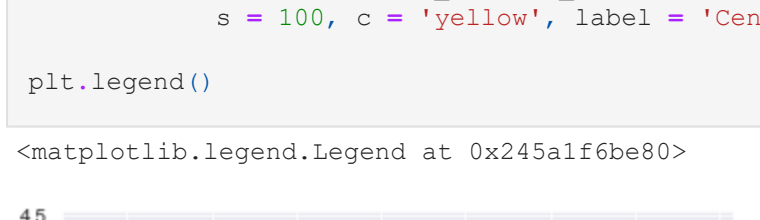
```
plt.title('The elbow method')
```

```
plt.xlabel('Number of clusters')
```

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

```
plt.show()
```

C:\Users\IT\Anaconda3\envs\tf_cpu\lib\site-packages\sklearn\cluster_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.



In Elbow method (from the above graph) 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 first we choose the number of clusters as * '2

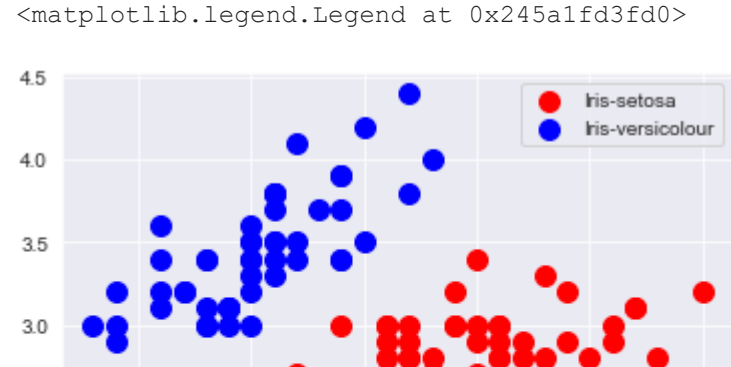
```
In [12]: # Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 2, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

```
In [13]: # 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, 0], kmeans.cluster_centers_[0, 1],
            s = 100, c = 'yellow', label = 'Centroids')
```

```
plt.legend()
```

```
Out[13]: <matplotlib.legend.Legend at 0x245a1f6be80>
```



According to this graph it is clear that model is unable to distinguish Iris-virginica while number of clusters is "2"

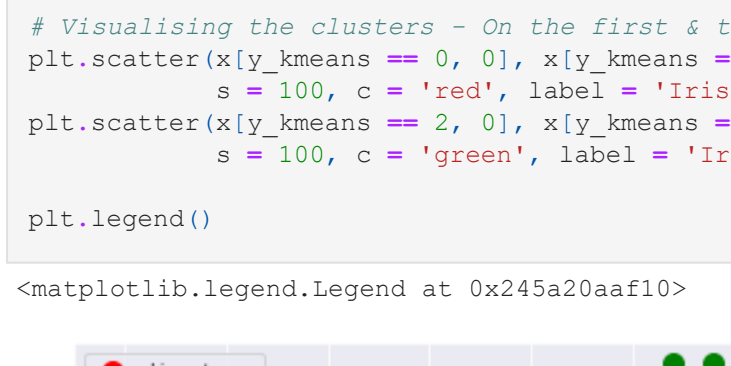
From this now we choose the number of clusters as * '3

```
In [14]: # 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 [15]: # 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.legend()
```

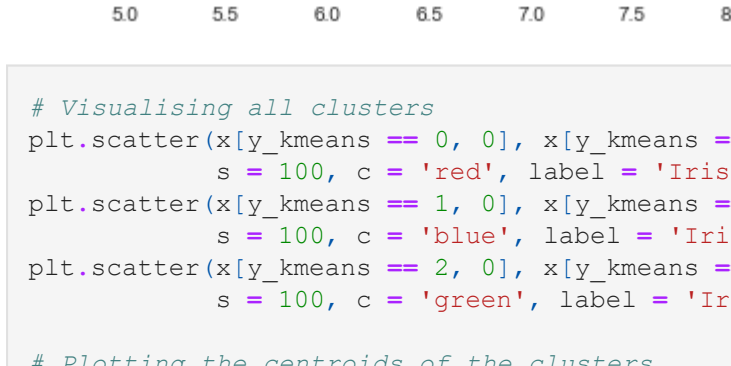
```
Out[15]: <matplotlib.legend.Legend at 0x245a1fd3fd0>
```



```
In [16]: # Visualising the clusters - On the second two columns
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.legend()
```

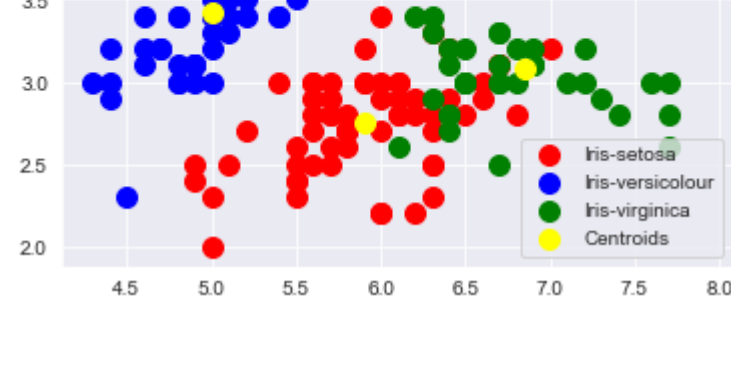
```
Out[16]: <matplotlib.legend.Legend at 0x245a202fe50>
```



```
In [17]: # Visualising the clusters - On the first & third 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 == 2, 0], x[y_kmeans == 2, 1],
            s = 100, c = 'green', label = 'Iris-virginica')
```

```
plt.legend()
```

```
Out[17]: <matplotlib.legend.Legend at 0x245a20aaf10>
```



```
In [18]: # Visualising all clusters
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, 0], kmeans.cluster_centers_[0, 1],
            s = 100, c = 'yellow', label = 'Centroids')
```

```
plt.legend()
```

```
Out[18]: <matplotlib.legend.Legend at 0x245a2113b50>
```


Conclusion: So Its clear from the above visualizations that best number of clusters for K-means classifier is "3"