TASK 2 : PREDICTION USING UNSUPERVISED ML

From the given Iris dataset, predict the optimal number of clusters and represent it visually. AUTHOR : TEJASWAREDDY

GRIP : THE SPARKS FOUNDATION

DATA SCIENCE AND BUSINESS ANALYTICS INTERN

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
```

In [20]: df = pd.read_csv("C:\Users\Tejaswi\Downloads\Iris.csv")

In [21]: df.head()

Out[21]:

	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

EDA

In [22]: df.shape

Out[22]: (150, 6)

In [23]: df.describe()

Out[23]:

	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 [24]: df.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: 7.0 KB
```

In [25]: df.corr()

Out[25]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

Looks like our data doesn't need preprocessing. Let's check for outliers.

In [26]:
```
def outlier_detection(df):
    # set variable to hold outliers
    # detecting the Q1, Q3 values and runs using IQRs first
    # loop through the numerical columns and do the outlined correctly:
    for col in df:
        for i in df.index:
            q1 = np.percentile(df[col],25)
            q3 = np.percentile(df[col],75)
            iqr = q3 - q1
            lower_bound = q1 -(1.5*iqr)
            upper_bound = q3 +(1.5*iqr)
            if df[col][i] > upper_bound:
                df[col] = df[col].replace(df[col][i],
                                          upper_bound)
                print('Outlier replaced')
            elif df[col][i] < lower_bound:
                df[col] = df[col].replace(df[col][i],
                                          lower_bound)
                print('Outlier replaced')
    # Removing the rows having values which can be called outliers
    # set the basis of their 2 scores of +3 or -3
    for col in df:
        for i in df.index:
            z = (df.loc[i,col]-
                 df.mean(axis=0,numeric_only=True)[col])
            if z > 3:
                df.drop(i,inplace=True)
            elif z < -3:
                df.drop(i,inplace=True)
    df = df.reset_index(drop=True)
    print('Outliers removed')
    return df
```

In [27]: df = outlier_detection(df)

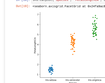
Out[27]: 0

No Outliers!

Data Visualizations

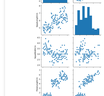
In [28]: sns.countplot(df['Species'])

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x1bf2d5a8c40>



In [29]: sns.catplot("Species", "PetalLengthCm", data = df)

Out[29]: <seaborn.axisgrid.FacetGrid at 0x1bf2d6e3b20>



In [30]: sns.pairplot(data=df)

Out[30]: <seaborn.axisgrid.PairGrid at 0x1bf2e1a7e40>

Let's look at the Elbow graph to find k.

In [31]:
```
x = df.iloc[:, [1, 2, 3]].values
inertia = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(x)
    inertia.append(kmeans.inertia_)

plt.plot(range(1, 11), inertia)
plt.title('Elbow Graph')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

In [32]: x

Out[32]:
```
array([[3.5, 1.4, 0.2],
       [3.0, 1.4, 0.2],
       [3.2, 1.3, 0.2],
       [3.1, 1.5, 0.2],
       [3.6, 1.4, 0.2],
       [3.9, 1.7, 0.4],
       [3.4, 1.4, 0.3],
       [3.4, 1.5, 0.2],
       [2.9, 1.4, 0.2],
       [3.1, 1.5, 0.1],
       [3.7, 1.5, 0.2],
       [3.4, 1.6, 0.2],
       [3.0, 1.4, 0.1],
       [3.0, 1.1, 0.1],
       [4.0, 1.2, 0.2],
       [4.4, 1.5, 0.4],
       [3.9, 1.3, 0.4],
       [3.5, 1.4, 0.3],
       [3.8, 1.7, 0.3],
       [3.8, 1.5, 0.3],
       [3.4, 1.7, 0.2],
       [3.7, 1.5, 0.4],
       [3.6, 1.0, 0.2],
       [3.3, 1.7, 0.5],
       [3.4, 1.9, 0.2],
       [3.0, 1.6, 0.2],
       [3.4, 1.6, 0.4],
       [3.5, 1.5, 0.2],
       [3.4, 1.4, 0.2],
       [3.2, 1.6, 0.2],
       [3.1, 1.6, 0.2],
       [3.4, 1.5, 0.4],
       [4.1, 1.5, 0.1],
       [4.2, 1.4, 0.2],
       [3.1, 1.5, 0.1],
       [3.2, 1.2, 0.2],
       [3.5, 1.3, 0.2],
       [3.6, 1.4, 0.1],
       [3.0, 1.3, 0.2],
       [3.4, 1.5, 0.2],
       [3.5, 1.3, 0.3],
       [2.3, 1.3, 0.3],
       [3.2, 1.3, 0.2],
       [3.5, 1.6, 0.6],
       [3.8, 1.9, 0.4],
       [3.0, 1.4, 0.3],
       [3.8, 1.6, 0.2],
       [3.2, 1.4, 0.2],
       [3.7, 1.5, 0.2],
       [3.3, 1.4, 0.2],
       [3.2, 4.7, 1.4],
       [3.2, 4.5, 1.5],
       [3.1, 4.9, 1.5],
       [2.3, 4.0, 1.3],
       [2.8, 4.6, 1.5],
       [2.8, 4.5, 1.3],
       [3.3, 4.7, 1.6],
       [2.4, 3.3, 1.0],
       [2.9, 4.6, 1.3],
       [2.7, 3.9, 1.4],
       [2.0, 3.5, 1.0],
       [3.0, 4.2, 1.5],
       [2.2, 4.0, 1.0],
       [2.9, 4.7, 1.4],
       [2.9, 3.6, 1.3],
       [3.1, 4.4, 1.4],
       [3.0, 4.5, 1.5],
       [2.7, 4.1, 1.0],
       [2.2, 4.5, 1.5],
       [2.5, 3.9, 1.1],
       [3.2, 4.8, 1.8],
       [2.8, 4.0, 1.3],
       [2.5, 4.9, 1.5],
       [2.8, 4.7, 1.2],
       [2.9, 4.3, 1.3],
       [3.1, 4.8, 1.8],
       [2.7, 5.1, 1.6],
       [3.0, 5.9, 2.1],
       [3.0, 5.6, 1.8],
       [2.9, 5.8, 2.4],
       [3.0, 6.6, 2.1],
       [2.5, 4.5, 1.7],
       [2.9, 6.3, 1.8],
       [2.5, 5.8, 1.8],
       [3.6, 6.1, 2.5],
       [3.2, 5.1, 2.0],
       [2.7, 5.3, 1.9],
       [3.0, 5.5, 2.1],
       [2.5, 5.0, 2.0],
       [2.8, 5.1, 2.4],
       [3.2, 5.3, 2.3],
       [3.0, 5.5, 1.8],
       [3.8, 6.7, 2.2],
       [2.6, 6.9, 2.3],
       [2.2, 5.0, 1.5],
       [3.2, 5.7, 2.3],
       [2.8, 4.9, 2.0],
       [2.8, 6.7, 2.0],
       [2.7, 4.9, 1.8],
       [3.3, 5.7, 2.1],
       [3.2, 6.0, 1.8],
       [2.8, 4.8, 1.8],
       [3.0, 4.9, 1.8],
       [2.8, 5.6, 2.1],
       [3.0, 5.8, 2.2],
       [3.8, 6.6, 2.1],
       [2.6, 5.0, 2.0],
       [3.3, 5.7, 2.5],
       [3.0, 5.2, 2.3],
       [2.5, 5.0, 1.9],
       [3.0, 5.2, 2.0],
       [3.4, 5.4, 2.3],
       [3.0, 5.1, 1.8]])
```

In [33]: kmeans = KMeans(n_clusters = 3, init = 'k-means++',
                 max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)

In [34]: # Predicts the clusters labels: labels
labels = kmeans.predict(x)

In [35]: labels

Out[35]:
```
array([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, 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
```