·•·⊰·○☽ The Sparks Foundation ☾○·⊱·•·

Task 1 :Prediction using Unsupervised ML

Objectives:

From the given ‘Iris’ dataset, predict the optimum number of clusters and represent it visually.

Author: Neha Parameshwar Sonavane

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

In [2]:
```
#Load the iris dataset
data=pd.read_csv(r"C:\Users\DELL\Downloads\Iris (1).csv")
data.head()                                 #first 5 rows of the dataset
```

Out[2]:

	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

In [3]:
```
data.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 [4]:
```
data.isna().sum()                #To check null values
```

Out[4]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

There are no missing values present in our dataset.The data is cleaned.

In [5]:
```
data.shape            #To see the shape of our data
```

Out[5]:
```
(150, 6)
```

There are 150 rows and 6 columns in our dataset.

In [6]:
```
data.nunique()     #for unique values in our data
```

Out[6]:
```
Id               150
SepalLengthCm     35
SepalWidthCm      23
PetalLengthCm     43
PetalWidthCm      22
Species            3
dtype: int64
```

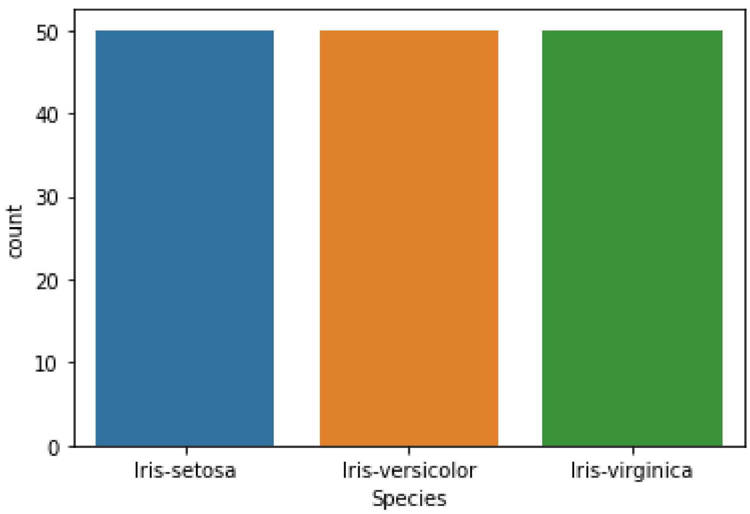
In [7]:
```
data['Species'].value_counts()
```

Out[7]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

In [8]:
```
#we clearly see that species has 3 different types as Iris_setosa, Iris_versic
```

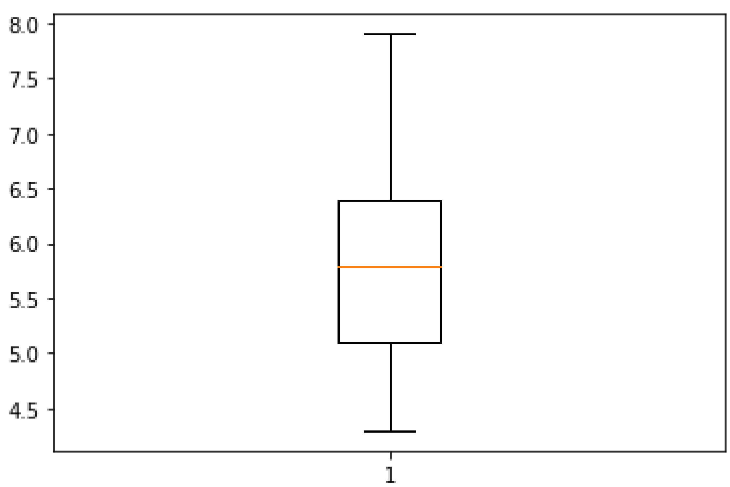
In [9]:
```
sns.countplot(data['Species'])
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



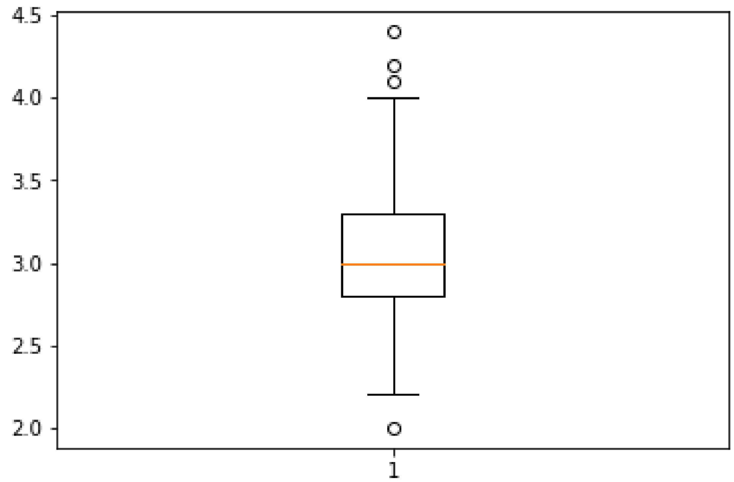
In [10]:
```
#To check whether outliers are present in feature variables we plot boxplot.
```

In [11]:
```
plt.boxplot(data['SepalLengthCm'])
plt.show()
```



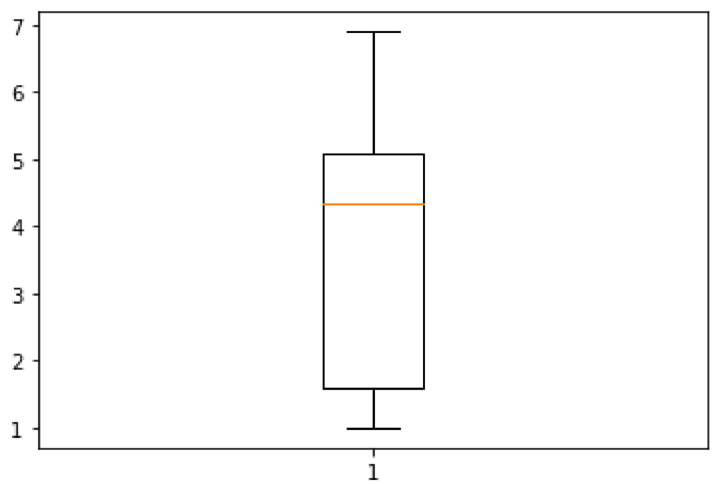
we clearly see that from above plot is that there are no outliers present in SepalLength variable.

In [12]:
```
plt.boxplot(data['SepalWidthCm'])
plt.show()
```



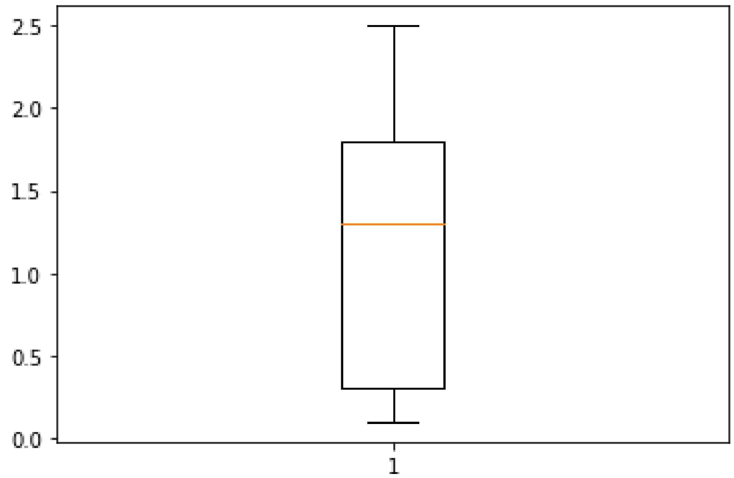
Here we observe that few outliers present in this variable.

In [13]:
```
plt.boxplot(data['PetalLengthCm'])
plt.show()
```



we clearly see that from above plot is that there are no outliers present in PetalLengthCm variable.

In [14]:
```
plt.boxplot(data['PetalWidthCm'])
plt.show()
```



we clearly see that from above plot is that there are no outliers present in PetalWidthCm variable.

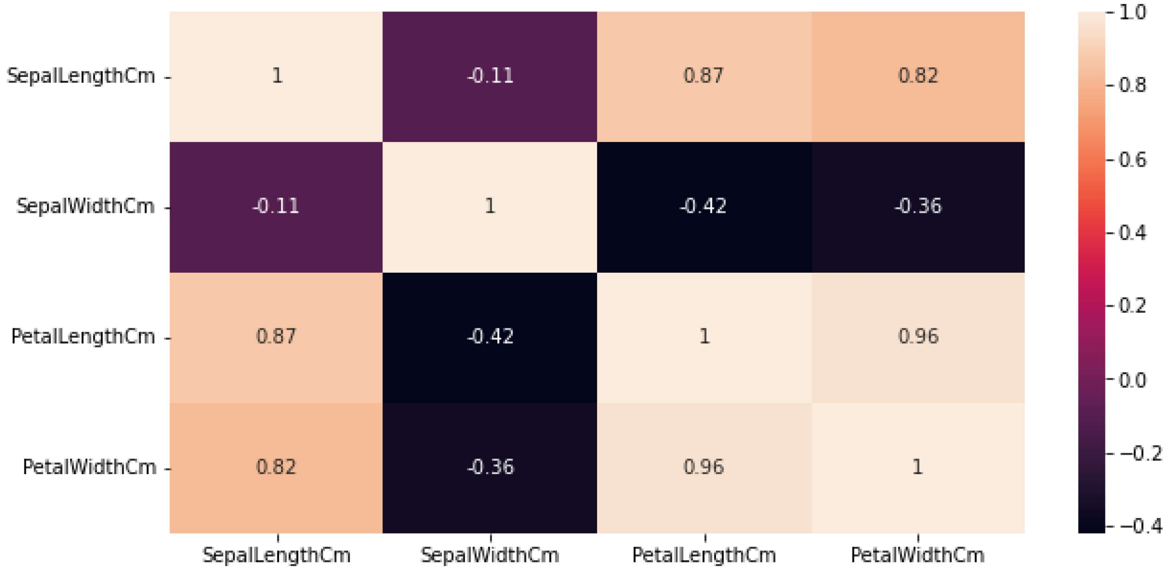
In [15]:
```
df=data.drop(['Id'],axis=1)
df.head()
```

Out[15]:

	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

In [16]:
```
plt.figure(figsize = (10,5))
sns.heatmap(df.corr(),annot=True)
```

Out[16]: <AxesSubplot:>



Q)How do you find the optimum number of clusters for K Means? How does one determine the value of K?

In [17]:
```
#Finding the optimum number clusters for k mean classification

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

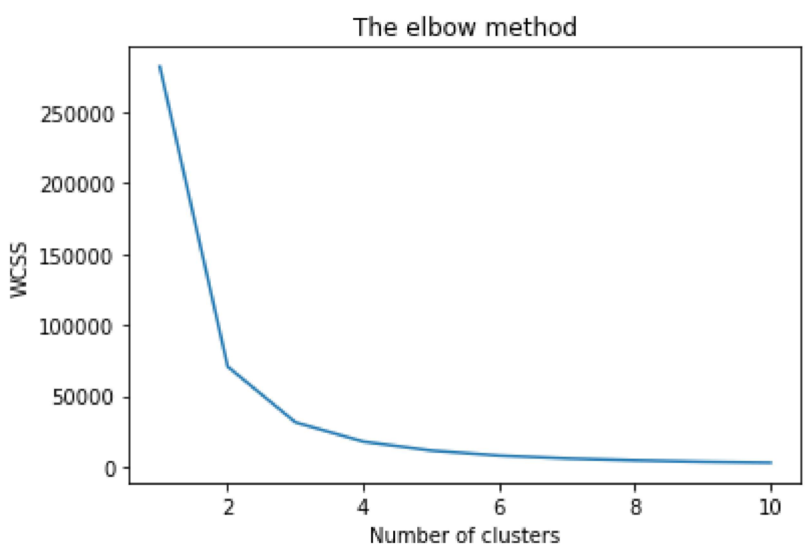
```
       [ 11. ,   5.4,   3.7,   1.5],
       [ 12. ,   4.8,   3.4,   1.6],
       [ 13. ,   4.8,   3. ,   1.4],
       [ 14. ,   4.3,   3. ,   1.1],
       [ 15. ,   5.8,   4. ,   1.2],
       [ 16. ,   5.7,   4.4,   1.5],
       [ 17. ,   5.4,   3.9,   1.3],
       [ 18. ,   5.1,   3.5,   1.4],
       [ 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],
```

In [18]:
```
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  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 sqaures
plt.show()
```

```
C:\ProgramData\Anaconda3\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.
  warnings.warn(
```



we 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.

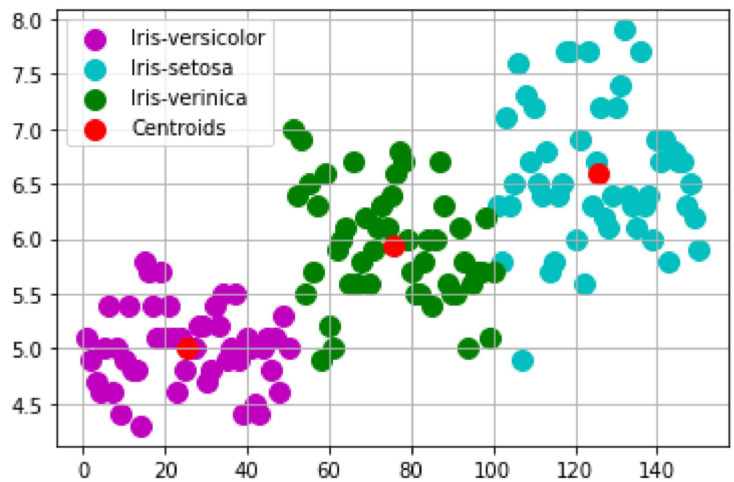
From this we choose the number of clusters as '3'.

In [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]:
```
# Visualising the clusters  -on the first two columns
plt.scatter(x[y_kmeans==0,0], x[y_kmeans == 0, 1],
            s = 100, c = 'm', label = 'Iris-versicolor')
plt.scatter(x[y_kmeans == 1,0], x[y_kmeans == 1,1],
            s = 100, c = 'c', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 2,0], x[y_kmeans == 2,1],
            s =100, c='g', label = 'Iris-verinica')

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s = 100, c ='r', label = 'Centroids')


plt.legend()
plt.grid()
plt.show()
```



In [21]:
```
#Here we clearly see three clusters with red dot as thier centroid.
```