Task-2: Prediction using Unsupervised ML

Author: Rudra Pratap Singh

In [1]:
```
#In this task I am going to predict the optimum number of clusters and represent it visually.
#From the given iris dataset.
#I am going to use python language for this task.
#I am procedind step by step.
```

Importing Required Libraries

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import datasets
import seaborn as sns
from sklearn.cluster import KMeans
```

Reading Data

In [4]:
```
#We use pandas to read the data.
data=pd.read_csv('Iris.csv')
print("Your data is: ")
data
```

Your data is:

Out[4]:

	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

In [5]:
```
#It will print last ten rows.
data.tail(10)
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
140	141	6.7	3.1	5.6	2.4	Iris-virginica
141	142	6.9	3.1	5.1	2.3	Iris-virginica
142	143	5.8	2.7	5.1	1.9	Iris-virginica
143	144	6.8	3.2	5.9	2.3	Iris-virginica
144	145	6.7	3.3	5.7	2.5	Iris-virginica
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]:
```
#If we want to know number of rows and number of columns then we use shape function.
data.shape
```

Out[6]: (150, 6)

In [7]:
```
#Statisticals details of given data is printed using describe() function.
data.describe()
```

Out[7]:

	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 [8]:
```
data['PetalLengthCm']#This will print the specific column from the data.
```

Out[8]:
```
0      1.4
1      1.4
2      1.3
3      1.5
4      1.4
      ...
145    5.2
146    5.0
147    5.2
148    5.4
149    5.1
Name: PetalLengthCm, Length: 150, dtype: float64
```

In [9]:
```
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: 6.5+ KB
```

In [10]:
```
data.isna().sum()#Checking the NaN values.
```

Out[10]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [11]:
```
data.PetalLengthCm.unique()#Printing and visualising the unique details using unique() function.
```

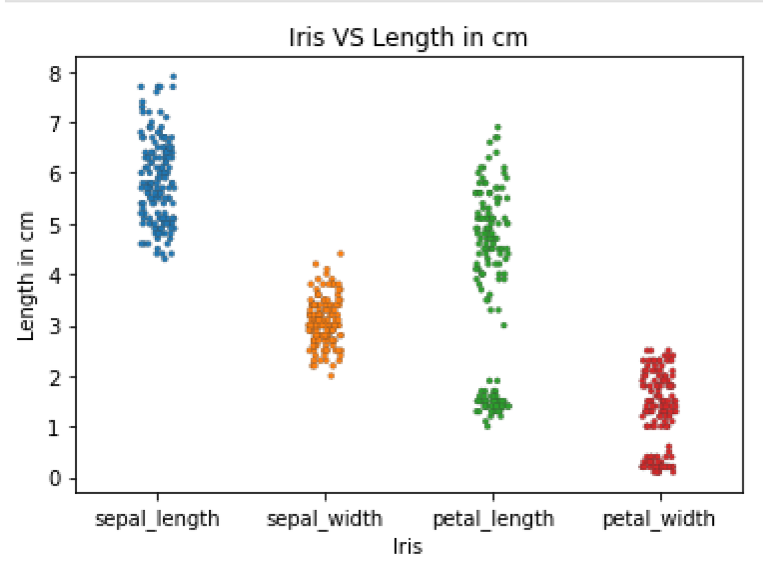
Out[11]:
```
array([1.4, 1.3, 1.5, 1.7, 1.6, 1.1, 1.2, 1. , 1.9, 4.7, 4.5, 4.9, 4. ,
       4.6, 3.3, 3.9, 3.5, 4.2, 3.6, 4.4, 4.1, 4.8, 4.3, 5. , 3.8, 3.7,
       5.1, 3. , 6. , 5.9, 5.6, 5.8, 6.6, 6.3, 6.1, 5.3, 5.5, 6.7, 6.9,
       5.7, 6.4, 5.4, 5.2])
```

In [12]:
```
data.Species.unique()
```

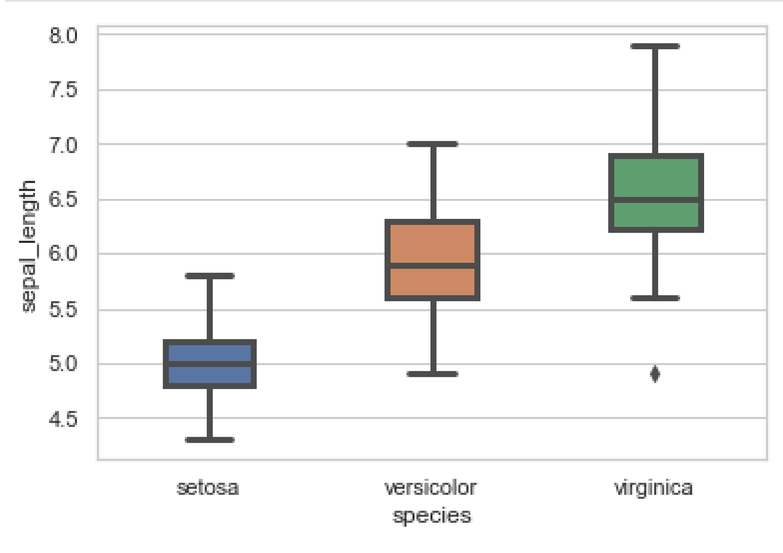
Out[12]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

Visualization of the given data

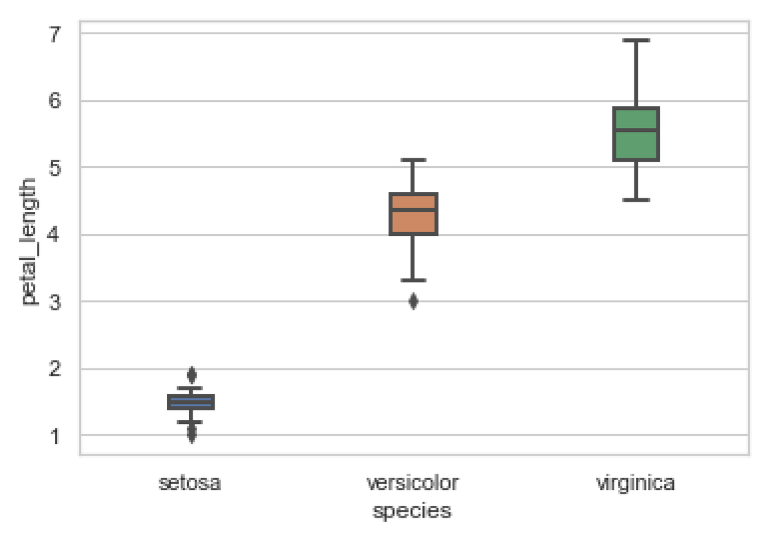
In [13]:
```
#Most code in the docs will use the load_dataset() function to get quick access to an example dataset.
iris=sns.load_dataset('iris');
#It is used to draw a scatter plot based on the category.
p=sns.stripplot(data=iris,size=3,linewidth=0.2);
plt.xlabel('Iris')
plt.ylabel('Length in cm')
plt.title("Iris VS Length in cm")
plt.show()
```



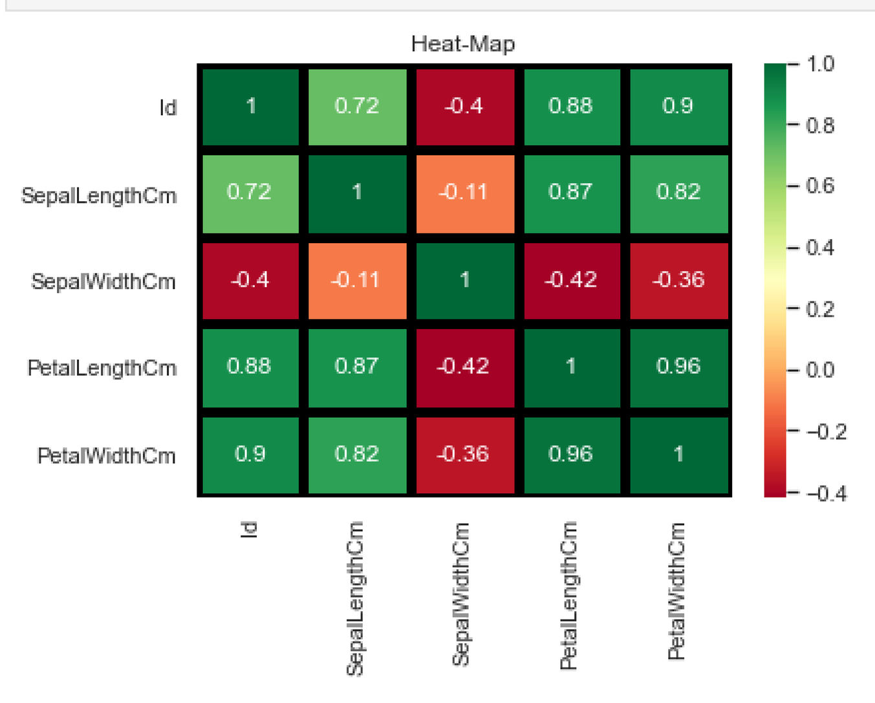
In [14]:
```
#A box plot helps to maintain the distribution of quantitative data in such a way that it facilitates the comparisons
#between variables or across levels of a categorical variable.
sns.set(style='whitegrid')
sns.boxplot(x='species',y='sepal_length',data=iris,linewidth=3,width=0.4)
plt.show()
```



In [15]:
```
sns.boxplot(x='species',y='petal_length',data=iris,linewidth=2,width=0.2)
plt.show()
```



In [16]:
```
#Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn',linewidth=5,linecolor='black')
plt.title("Heat-Map")
plt.show()
```



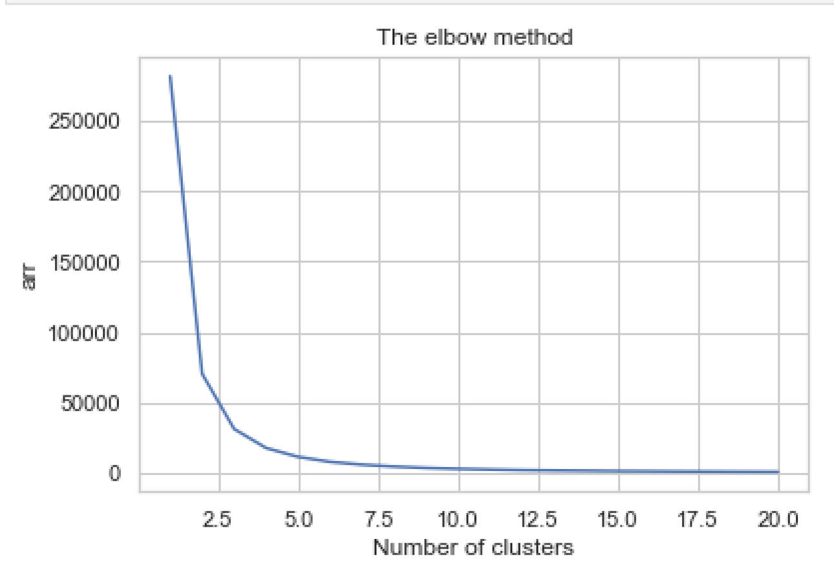
Using k-means clustering finding the optimum number of clusters.

In [17]:
```
#The k-means clustering method is an unsupervised machine learning technique used to identify clusters of data objects.
x=data.iloc[:,[0,1,2,3]].values
arr=[]
for i in range(1,21):
    kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(x)
    arr.append(kmeans.inertia_)
    print(i,kmeans.inertia_)
```

```
1 281831.54466666654
2 70581.3808
3 31320.711199999994
4 17758.792503556186
5 11468.968747023802
6 7921.863473076923
7 5911.632365518538
8 4541.979023391812
9 3571.911095588236
10 2943.9331008403374
11 2464.0812948717935
12 2039.845238927738
13 1785.6793065268062
14 1571.0434581529594
15 1353.0395353535357
16 1225.361512626262
17 1080.3908308080813
18 980.2366587301586
19 875.790099206349
20 808.788115079365
```

The elbow method

In [18]:
```
#The technique to determine K, the number of clusters, is called the elbow method.
#When K increases, the centroids are closer to the clusters centroids.
plt.plot(range(1,21),arr)
plt.xlabel("Number of clusters")
plt.ylabel("arr")
plt.title("The elbow method")
plt.show()
```



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

Out[19]:
```
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, 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, 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])
```

Visualizing the clusters.

In [20]:
```
plt.figure(figsize=(9,9))
plt.scatter(x[Kmeans==0,0],x[Kmeans==0,1],s=100,color='yellow',label='Iris-setosa')
plt.scatter(x[Kmeans==1,0],x[Kmeans==1,1],s=100,color='black',label='Iris-versicolor')
plt.scatter(x[Kmeans==2,0],x[Kmeans==2,1],s=100,color='blue',label='Iris-virginica')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],color='red',label='Centroids')
plt.xlabel("Sepal length in cm")
plt.ylabel("Petal length in cm")
plt.title("Iris Flower Cluster")
plt.legend()
plt.show()
```