GRIP @THE SPARKS FOUNDATION

TASK 2 : Prediction using Unsupervised ML

AUTHOR : Yash Agarwal

OBJECTIVE : From the given 'IRIS' dataset, predict the Optimum number of Clusters and represent it visually.

STEP 1 : Importing the Dataset from the given link.

In [11]:
```python
# import the required liberaries

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import plotly.express as px
from sklearn import datasets
from sklearn.datasets import load_iris
```

In [18]:
```python
df=pd.read_csv('C:\\Users\\Yash Agarwal\\Desktop\\Iris.csv',index_col=0,header=0)
df
```

Out[18]:

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

150 rows × 5 columns

STEP 2 : Visualising our Data

In [19]:
```python
df.head()
```

Out[19]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

In [20]:
```python
# Analysis
print(df.shape)
print(df.info())
```

```
(150, 5)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
None
```

In [21]:
```python
print(df.isnull().sum())       # detects the missing values.
df.describe(include="all")     # generates the descriptive statistices which include the mean, median, mode
```

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

Out[21]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	Iris-versicolor
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.054000	3.758667	1.198667	NaN
std	0.828066	0.433594	1.764420	0.763161	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

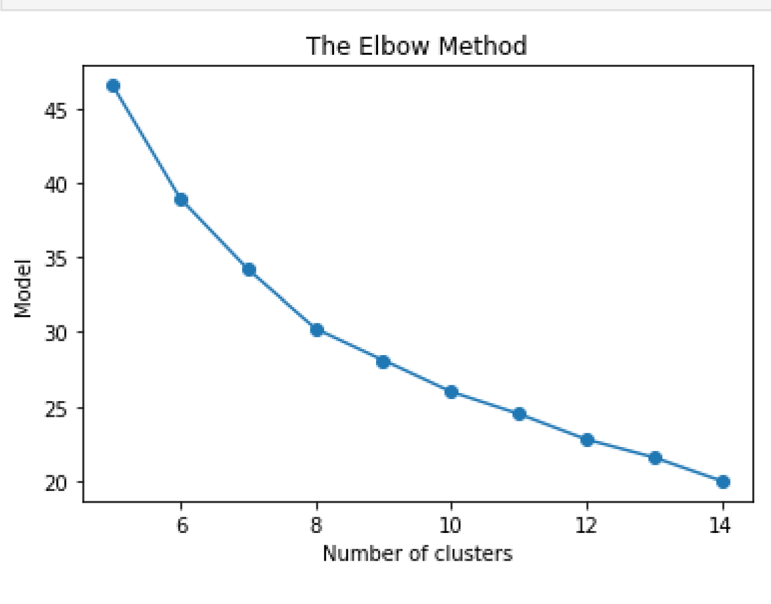
In [22]:
```python
y=df.values[:,:-1]
y
```

Out[22]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3.0, 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5.0, 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5.0, 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3.0, 1.4, 0.1],
       [4.3, 3.0, 1.1, 0.1],
       [5.8, 4.0, 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1.0, 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5.0, 3.0, 1.6, 0.2],
       [5.0, 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.0, 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [4.4, 3.0, 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5.0, 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5.0, 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3.0, 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5.0, 3.3, 1.4, 0.2],
       [7.0, 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4.0, 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1.0],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5.0, 2.0, 3.5, 1.0],
       [5.9, 3.0, 4.2, 1.5],
       [6.0, 2.2, 4.0, 1.0],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3.0, 4.5, 1.5],
       [5.8, 2.7, 4.1, 1.0],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4.0, 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3.0, 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3.0, 5.0, 1.7],
       [6.0, 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1.0],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1.0],
       [5.8, 2.7, 3.9, 1.2],
       [6.0, 2.7, 5.1, 1.6],
       [5.4, 3.0, 4.5, 1.5],
       [6.0, 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3.0, 4.1, 1.3],
       [5.5, 2.5, 4.0, 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3.0, 4.6, 1.4],
       [5.8, 2.6, 4.0, 1.2],
       [5.0, 2.3, 3.3, 1.0],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3.0, 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3.0, 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6.0, 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3.0, 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3.0, 5.8, 2.2],
       [7.6, 3.0, 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2.0],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3.0, 5.5, 2.1],
       [5.7, 2.5, 5.0, 2.0],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3.0, 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6.0, 2.2, 5.0, 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2.0],
       [7.7, 2.8, 6.7, 2.0],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6.0, 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3.0, 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3.0, 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2.0],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3.0, 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6.0, 3.0, 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3.0, 5.2, 2.3],
       [6.3, 2.5, 5.0, 1.9],
       [6.5, 3.0, 5.2, 2.0],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3.0, 5.1, 1.8]], dtype=object)
```

STEP 3: Find the Optimum number of Clusters.

In [24]:
```python
# We use the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans

model = []
for i in range(5, 15):
    kmeans = KMeans(n_clusters = i, random_state = 10)
    kmeans.fit(y)
    model.append(kmeans.inertia_)
plt.plot(range(5, 15), model)
plt.scatter(range(5, 15),model)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Model')
plt.show()
```



In [25]:
```python
print(model)
```

```
[46.535582051282034, 38.93873974358975, 34.19846461871464, 30.236524046129325, 28.11553453563981, 25.996554473304478, 24.514421536796547, 22.797199314574318, 21.591339247353954, 20.009916278166283]
```

Apply the K-means cluster on Data

In [26]:
```python
#fitting K-means to our dataset

kmeans = KMeans(n_clusters = 5, random_state = 10)
Y_pred = kmeans.fit_predict(y)

#kmeans.fit(x)
#Y_pred = kmeans.predict(x)
```

In [27]:
```python
Y_pred
```

Out[27]:
```
array([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, 0, 0, 0, 3, 0, 0, 0, 3, 0, 3, 3, 0, 3, 0, 3, 0,
       0, 3, 0, 3, 0, 3, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 0, 3, 0, 0, 0,
       3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 3, 3, 4, 0, 2, 4, 4, 2, 3, 2, 4, 2,
       4, 4, 4, 0, 4, 4, 4, 2, 2, 0, 4, 0, 2, 0, 4, 2, 0, 0, 4, 2, 2, 2,
       4, 0, 0, 2, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4, 0, 4, 4, 0])
```

In [28]:
```python
kmeans.n_iter_
```

Out[28]: 4

In [30]:
```python
df["predicted"]=Y_pred
df.head() #returns the first n rows of the dataset
```

Out[30]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	predicted
1	5.1	3.5	1.4	0.2	Iris-setosa	1
2	4.9	3.0	1.4	0.2	Iris-setosa	1
3	4.7	3.2	1.3	0.2	Iris-setosa	1
4	4.6	3.1	1.5	0.2	Iris-setosa	1
5	5.0	3.6	1.4	0.2	Iris-setosa	1

In [33]:
```python
df.predicted=df.predicted.map({0:'Iris-setosa',1:'Iris-versicolor',2:'Iris-virginica'})
df.head()
```

Out[33]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	predicted
1	5.1	3.5	1.4	0.2	Iris-setosa	Iris-versicolor
2	4.9	3.0	1.4	0.2	Iris-setosa	Iris-versicolor
3	4.7	3.2	1.3	0.2	Iris-setosa	Iris-versicolor
4	4.6	3.1	1.5	0.2	Iris-setosa	Iris-versicolor
5	5.0	3.6	1.4	0.2	Iris-setosa	Iris-versicolor

In [34]:
```python
sns.lmplot(data=df,x="SepalLengthCm", y="SepalWidthCm",
              fit_reg=False, # No regression line
hue="predicted",palette="Set1")
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s = 200, c = 'blue')
plt.title('sepal length vs sepal width',size=20)
plt.show()
```