LETS GROW MORE

Data Analytics Tasks

NAME : M.NIVETHITHAA

TASK 1 - Iris Flower Classification

DATA AUDIT

Importing Libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
import seaborn as sns
from sklearn.svm import SVC
from pandas.plotting import scatter_matrix as sm
from sklearn.cluster import KMeans
```

Displaying Raw Dataset

In [2]:
```
iris = datasets.load_iris()
iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_df
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

First five rows of the dataset

In [3]:
```
iris_df.head(5)
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Last five rows of the dataset

In [4]:
```
iris_df.tail(5)
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Shape of the dataset

In [5]:
```
iris_df.shape
```

Out[5]:
```
(150, 4)
```

Columns present in the dataset

In [6]:
```
iris_df.columns
```

Out[6]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

```
<h4>Summary of the dataset</h4>
```

In [7]:
```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

In [8]:
```
iris_df.describe()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Checking Datatypes

In [9]:
```
iris_df.dtypes
```

Out[9]:
```
sepal length (cm)    float64
sepal width (cm)     float64
petal length (cm)    float64
petal width (cm)     float64
dtype: object
```

Checking missing values

In [10]:
```
iris_df.isna().sum()
```

Out[10]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
dtype: int64
```

No Missing values

CORRELATION

In [11]:
```
corr_df= iris_df.corr()
corr_df
```

Out[11]:

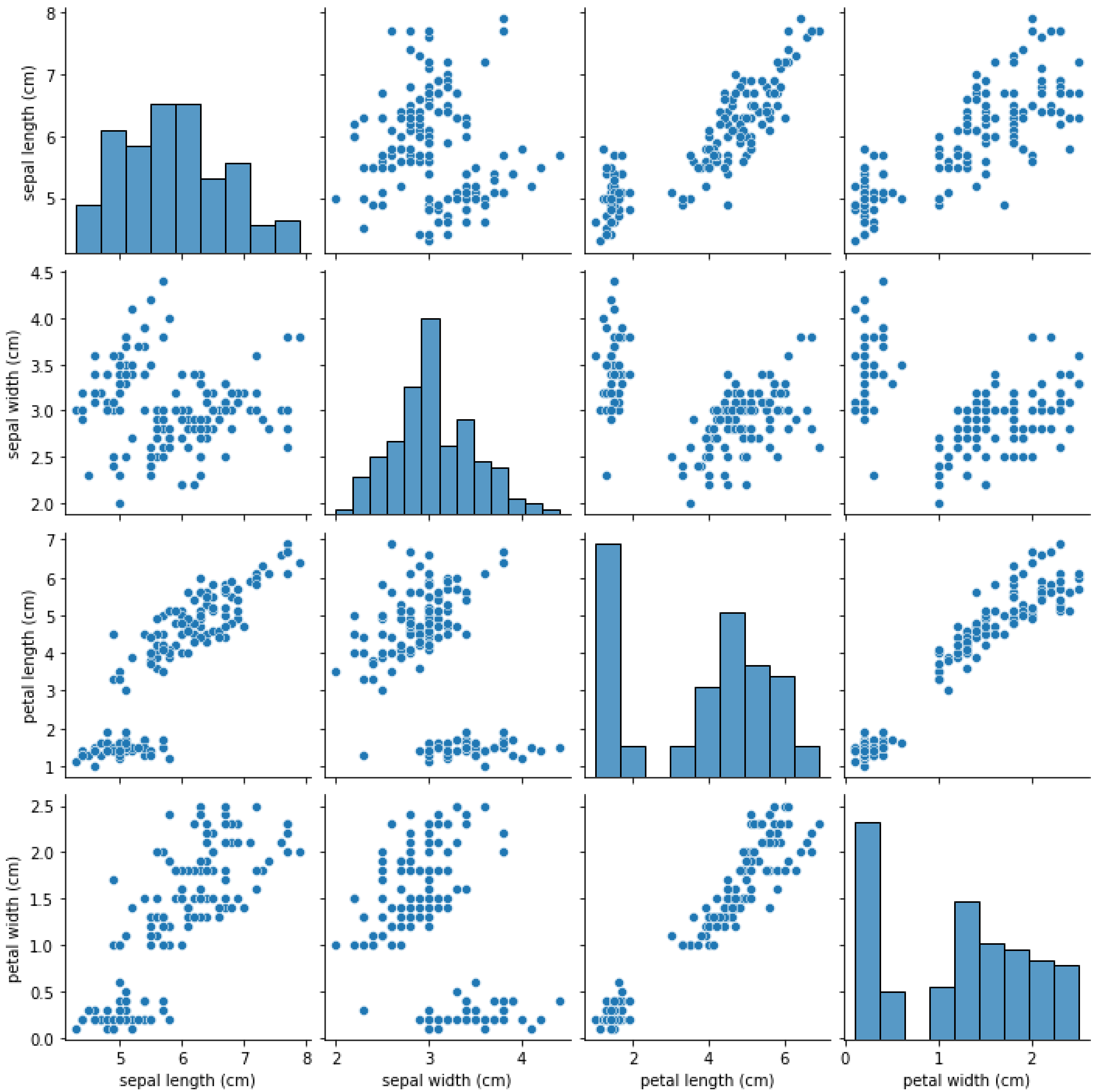
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

VISUALIZATION

```
<h3>Plotting</h3>
```

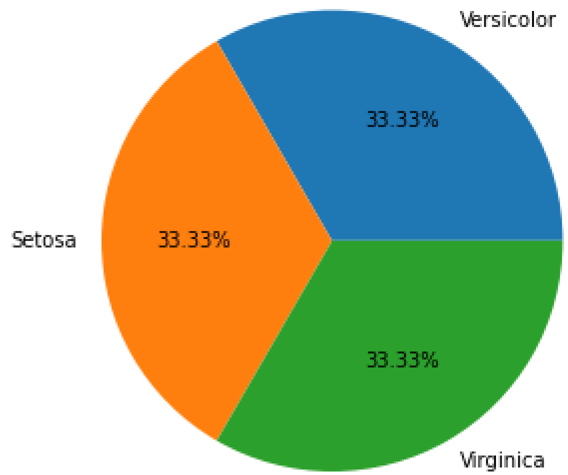
In [12]:
```
sns.pairplot(iris_df)
```

Out[12]: <seaborn.axisgrid.PairGrid at 0x228b981cdf0>


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

Pie Chart

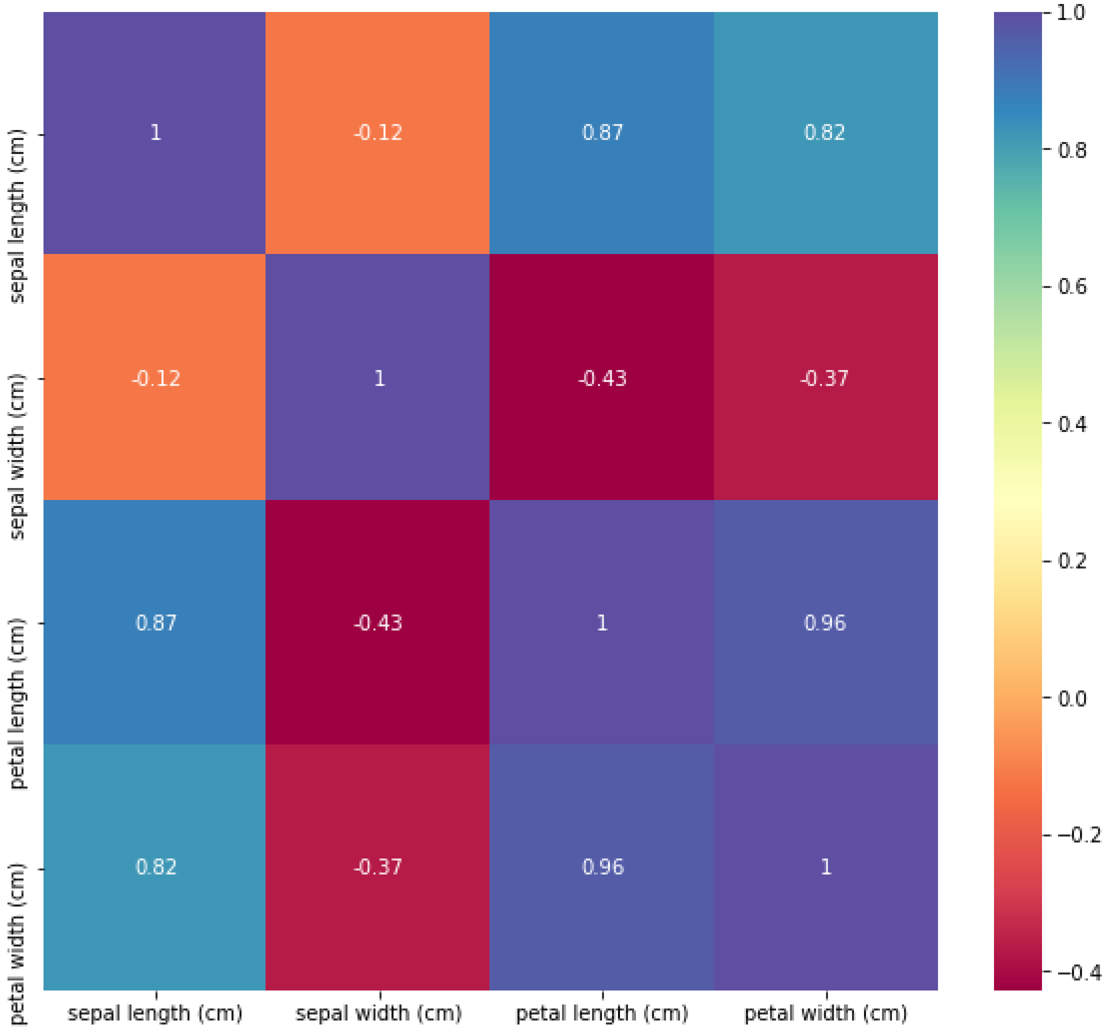
In [13]:
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
l = ['Versicolor', 'Setosa', 'Virginica']
s = [50,50,50]
ax.pie(s, labels = l,autopct='%1.2f%%')
plt.show()
```



Heat Map

In [14]:
```
plt.figure(figsize= [10,9])
sns.heatmap(corr_df, cmap='Spectral', annot=True)
```

Out[14]: <AxesSubplot:>



K-Means Clustering Algorithm

Predict the optimum number of clusters

Preparing the Data

Create X

In [15]:
```
x = iris_df.iloc[:, [0, 1, 2, 3,]].values
x
```

Out[15]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 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. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 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],
```

Create Y

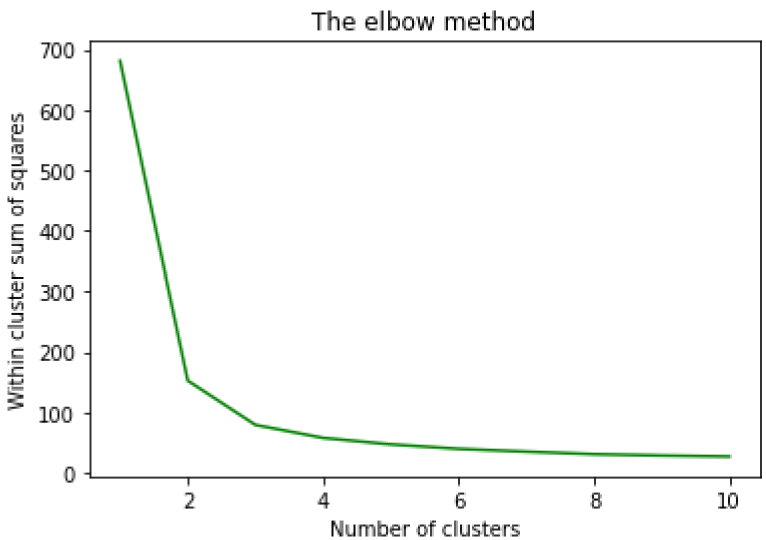
Elbow Method

In [16]:
```
#For creating y we need no of clusters. And we can find the optimum number of clusters using elbow method
```

-> In the Elbow method, we are actually varying the number of clusters (K) from 1 – 10.
-> For each value of K, we are calculating WCSS (Within-Cluster Sum of Square).
-> WCSS is the sum of squared distance between each point and the centroid in a cluster.
-> When we plot the WCSS with the K value, the plot looks like an Elbow.
-> When we analyze the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape.The K value corresponding to this point is the optimal K value or an optimal number of clusters

In [17]:
```
within_cluster_sum_of_squares = []
Range_of_cluster = range(1, 11)
for i in Range_of_cluster:
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(x)
    within_cluster_sum_of_squares.append(kmeans.inertia_)
```

In [18]:
```
plt.plot(Range_of_cluster, within_cluster_sum_of_squares,color="green")
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Within cluster sum of squares')
plt.show()
```



The point at which the elbow shape is created is 3.Therefore optimal number of clusters is 3

K-Means Clustering

In [19]:
```
y = KMeans(n_clusters=3)
```

Fit K means cluster model

In [20]:
```
y_kmean = y.fit_predict(x)
y_kmean
```

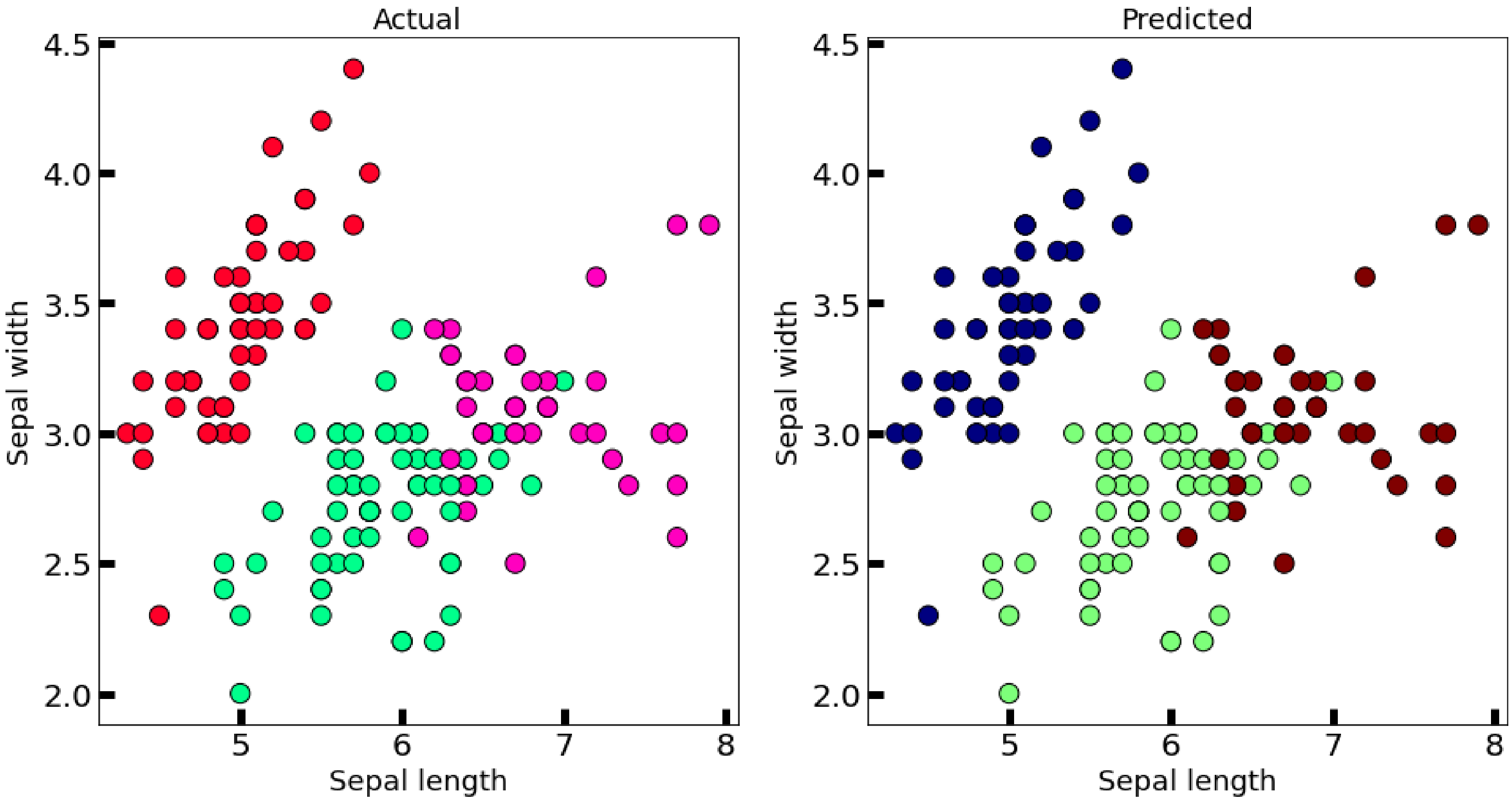
Out[20]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

Visualization of Results

Compare our original data vs clustered results

In [21]:
```
new_labels = y.labels_
fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(x[:, 0], x[:, 1], c=y_kmean, cmap='gist_rainbow',
edgecolor='k', s=150)
axes[1].scatter(x[:, 0], x[:, 1], c=new_labels, cmap='jet',
edgecolor='k', s=150)
axes[0].set_xlabel('Sepal length', fontsize=18)
axes[0].set_ylabel('Sepal width', fontsize=18)
axes[1].set_xlabel('Sepal length', fontsize=18)
axes[1].set_ylabel('Sepal width', fontsize=18)
axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[0].set_title('Actual', fontsize=18)
axes[1].set_title('Predicted', fontsize=18)
```

Out[21]:
```
Text(0.5, 1.0, 'Predicted')
```


Actual
Predicted
Sepal width
Sepal length
4.5
4.0
3.5
3.0
2.5
2.0
5
6
7
8