The Sparks Foundation - GRIP - Data Science and Business Analytics - NOV'2020

TASK 2 : Prediction using unsupervised ML

Author : Gomini Gupta

Dataset used: Iris dataset, which is available in sklearn library.

- Alternatively, it can be downloaded through the following link - https://bit.ly/2TK5Xn5

Problem Statement :

- Predict the optimum number of clusters and represent it visually.

Import required libraries

In [1]:

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Read the data

In [2]:
```
data = pd.read_csv('Iris.csv')
data
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	S
0	1	5.1	3.5	1.4	0.2	
1	2	4.9	3.0	1.4	0.2	
2	3	4.7	3.2	1.3	0.2	
3	4	4.6	3.1	1.5	0.2	
4	5	5.0	3.6	1.4	0.2	
...	...	...	...	...	...	
145	146	6.7	3.0	5.2	2.3	\
146	147	6.3	2.5	5.0	1.9	\
147	148	6.5	3.0	5.2	2.0	\
148	149	6.2	3.4	5.4	2.3	\
149	150	5.9	3.0	5.1	1.8	\

150 rows × 6 columns

In [3]:
```
data.shape
```

Out[3]:
```
(150, 6)
```

In [4]:
```
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 [5]:

```
# dropping Id column

data.drop('Id', axis=1, inplace=True)
data.columns
```

Out[5]:

```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'Peta
lWidthCm',
       'Species'],
      dtype='object')
```

In [6]:

```
print(data.isnull().sum(), '\n\nNumber of duplicate rows: ' ,
```

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

Number of duplicate rows:  3
```

In [7]:

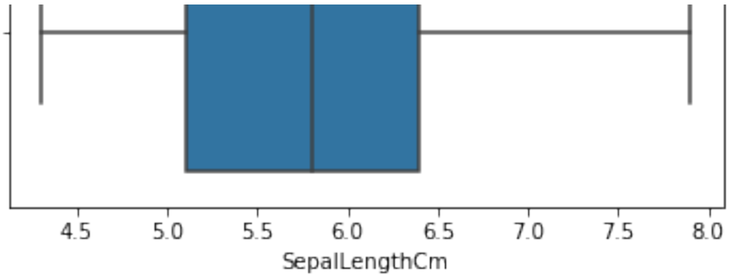
```
## drop duplicate rows

data.drop_duplicates(inplace=True)

data.shape[0]  # gives number of rows. Similarly, data.shape[

## now number of rows left 147, earlier there were 150 rows.
```

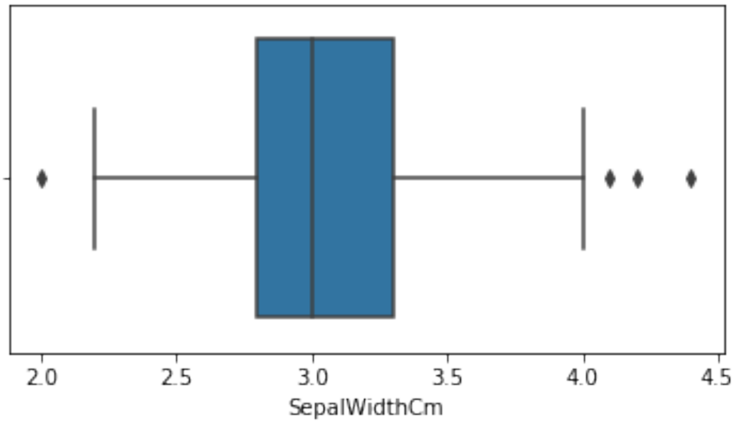
Out[7]:

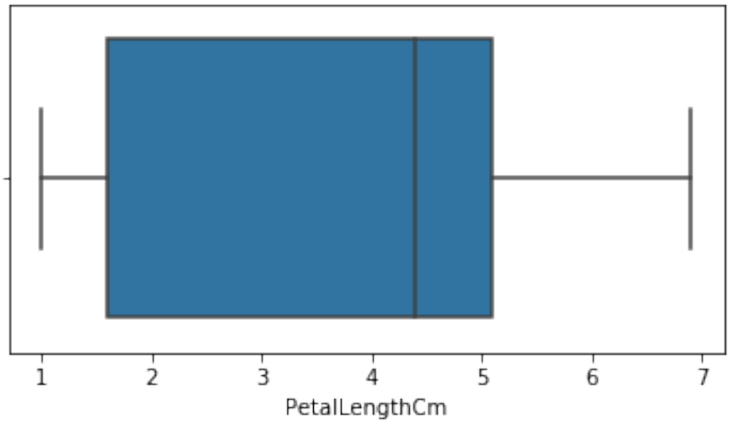
```
147
```

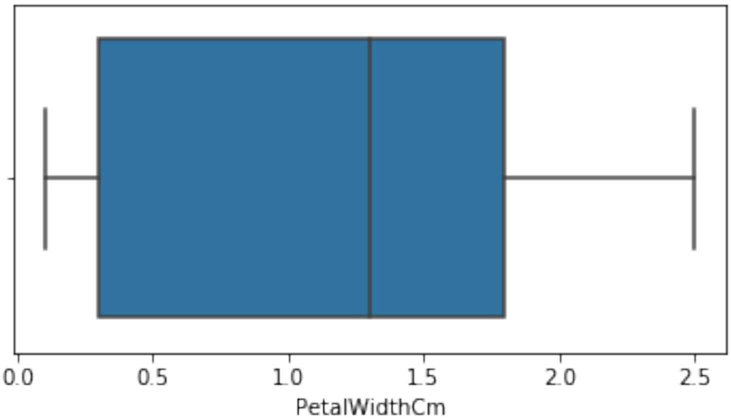
In [8]:

```
## Check for any outliers in the numeric data
for i in data.columns:
    if data[i].dtype=='float64':
        plt.figure(figsize=(6,3))
        sns.boxplot(data[i])
        plt.show()
```









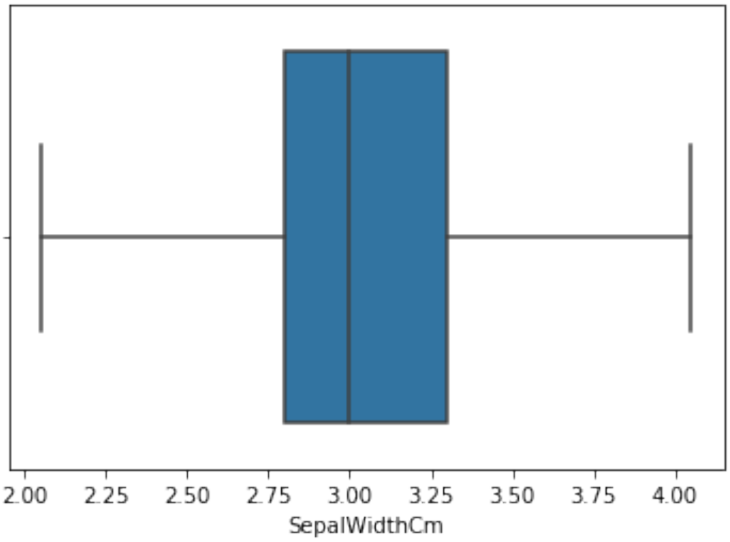
In [9]:

```
## Treating outliers present in the SepalWidthCm column

q1,q3 = np.percentile(data['SepalWidthCm'],[25,75])
iqr = q3-q1
lower_fence = q1 - (1.5*iqr)
upper_fence = q3 + (1.5*iqr)
data['SepalWidthCm'] = data['SepalWidthCm'].apply(lambda x: u
```

```
else lower_:
```

In [10]:

```
sns.boxplot(data['SepalWidthCm']);
```

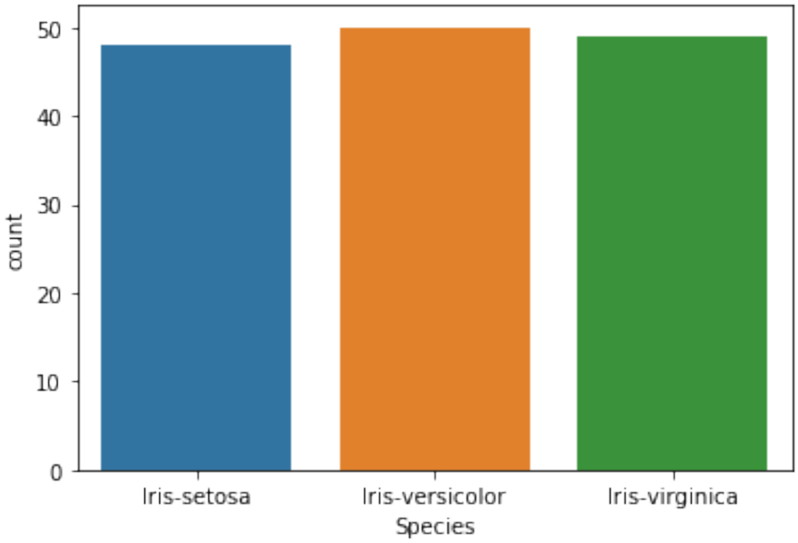


Understanding the data

In [11]:

```
## Target class
print(data.Species.value_counts())
sns.countplot(data.Species);
```

```
Iris-versicolor    50
Iris-virginica     49
Iris-setosa        48
Name: Species, dtype: int64
```



In [12]:

```
data.describe()
```

Out[12]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	147.000000	147.000000	147.000000	147.000000
mean	5.856463	3.052381	3.780272	1.208844
std	0.829100	0.426331	1.759111	0.757874
min	4.300000	2.050000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.050000	6.900000	2.500000

In [13]:

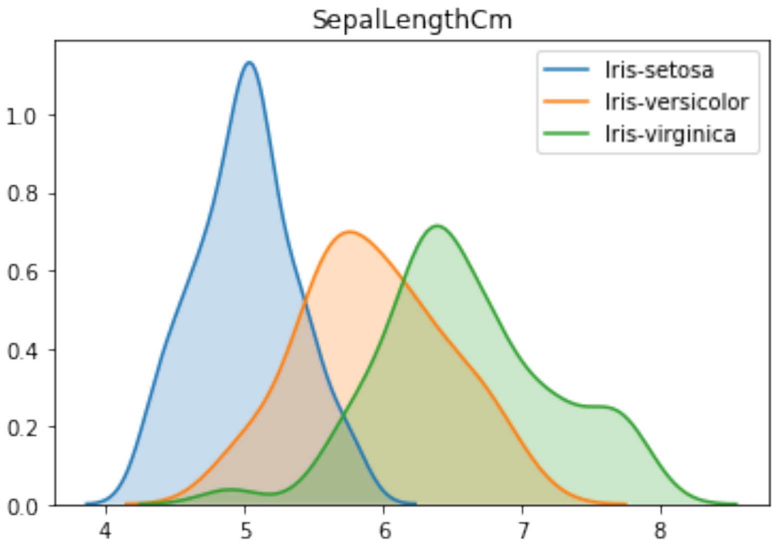
```
data.Species.unique()
```

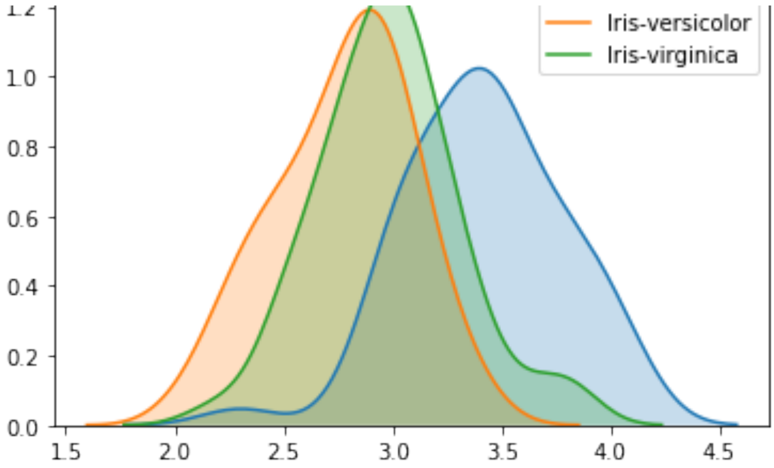
Out[13]:

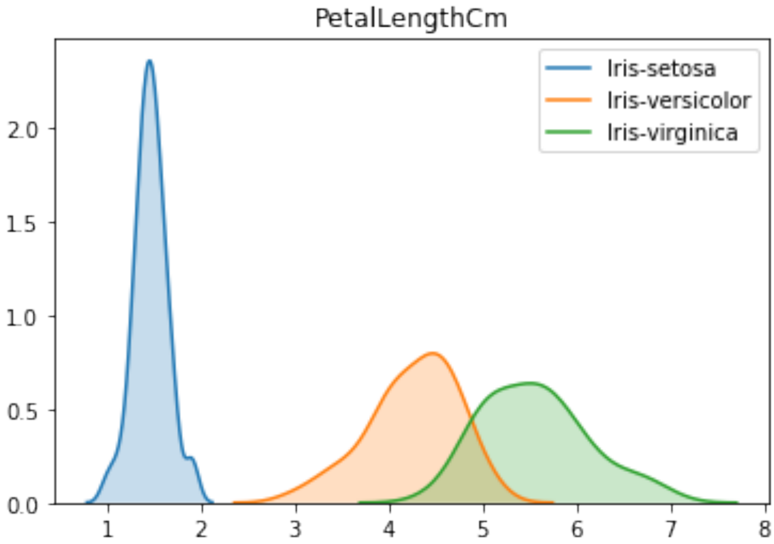
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dt
ype=object)
```

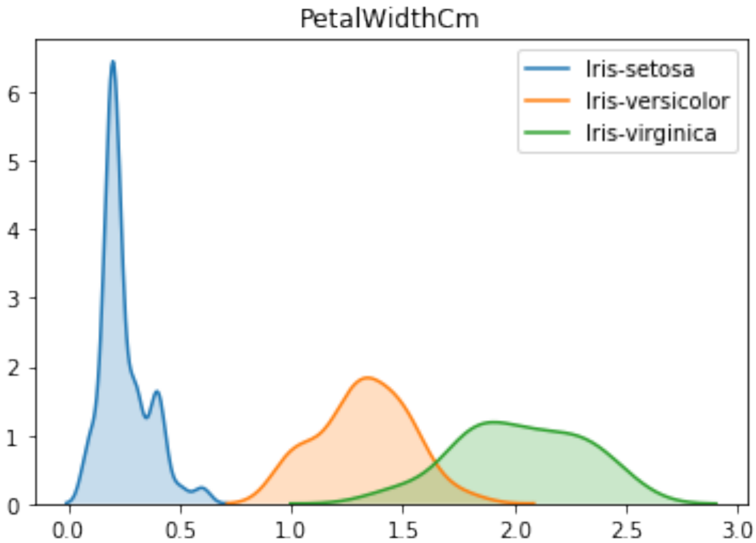
In [14]:

```
# Distributions of features by Species

for i in data.columns[:-1]:

    sns.kdeplot(data = data.loc[data.Species=='Iris-setosa'][:
    
    sns.kdeplot(data = data.loc[data.Species=='Iris-versicolo:

    sns.kdeplot(data = data.loc[data.Species=='Iris-virginica

    plt.title(i);

    plt.show()
```









In [15]:

```
## Inference: We can not distinguish between the species base
                # but we can clearly tell setosa apart from the
```

In [16]:

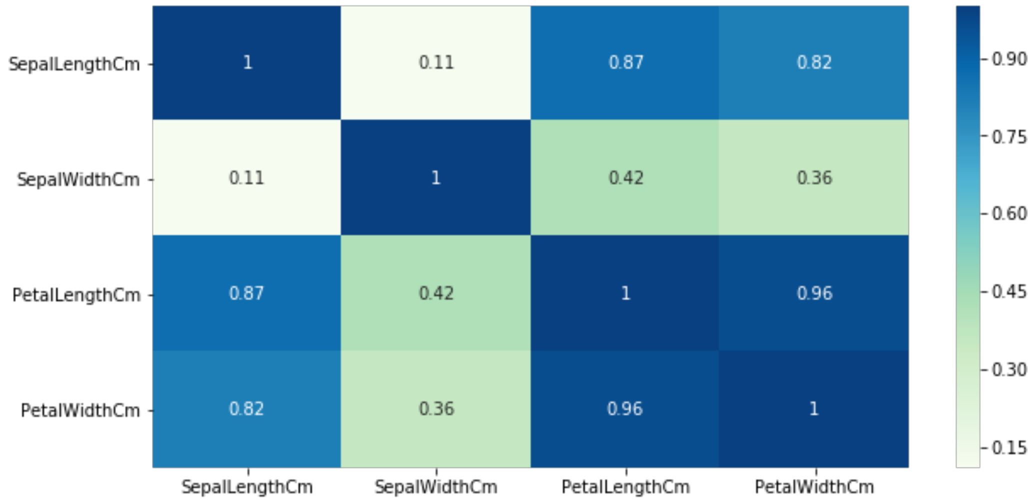
```
## Correlation Matrix

data.corr()
```

Out[16]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidt
SepalLengthCm	1.000000	-0.110155	0.871305	0.817
SepalWidthCm	-0.110155	1.000000	-0.420140	-0.355
PetalLengthCm	0.871305	-0.420140	1.000000	0.961
PetalWidthCm	0.817058	-0.355139	0.961883	1.000

In [17]:

```
plt.figure(figsize=(10,5))
sns.heatmap(abs(data.corr()), cmap='GnBu', annot=True);
```



K-means clustering

In [18]:

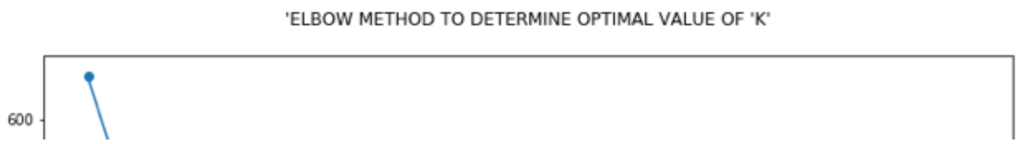
```
from sklearn.cluster import KMeans
```

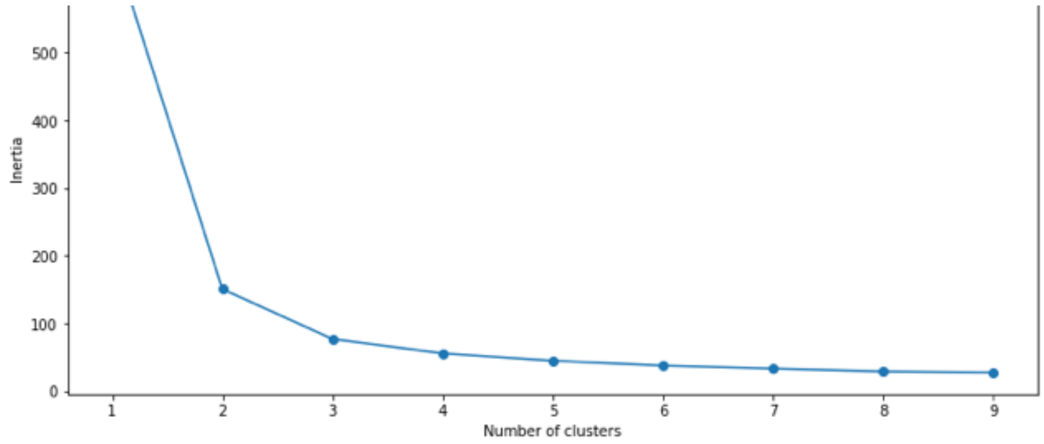
In [19]:

```
SSE = []
for i in range(1,10):
    kmeans = KMeans(n_jobs = -1, n_clusters = i, init='k-means
    kmeans.fit(data.iloc[:,[0,1,2,3]])
    SSE.append(kmeans.inertia_)
```

In [20]:

```
df = pd.DataFrame({'Cluster':range(1,10), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(df['Cluster'], df['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia');
plt.title("'ELBOW METHOD TO DETERMINE OPTIMAL VALUE OF 'K'\n"
```





In [21]:
```
kmeans = KMeans(n_jobs = -1, n_clusters = 3, init='k-means++'
kmeans.fit(data.iloc[:,[0,1,2,3]])
kmeans.cluster_centers_
```

Out[21]:
```
array([[5.90327869, 2.75      , 4.38196721, 1.42622951],
       [5.01041667, 3.41979167, 1.4625    , 0.25      ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

In [22]:
```
kmeans.labels_
```

Out[22]:
```
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, 0, 0, 2, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2
, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2
, 2, 2, 0,
       2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0])
```

In [23]:
```
data['cluster'] = kmeans.labels_

data
```

Out[23]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Specie
0	5.1	3.5	1.4	0.2	Iri setos
1	4.9	3.0	1.4	0.2	Iri setos
2	4.7	3.2	1.3	0.2	Iri setos
3	4.6	3.1	1.5	0.2	Iri setos

4	5.0	3.6	1.4	0.2	Iri setos
...	...	...	...	...	
145	6.7	3.0	5.2	2.3	Iri virgini
146	6.3	2.5	5.0	1.9	Iri virgini
147	6.5	3.0	5.2	2.0	Iri virgini
148	6.2	3.4	5.4	2.3	Iri virgini
149	5.9	3.0	5.1	1.8	Iri virgini

147 rows × 6 columns

In [24]:

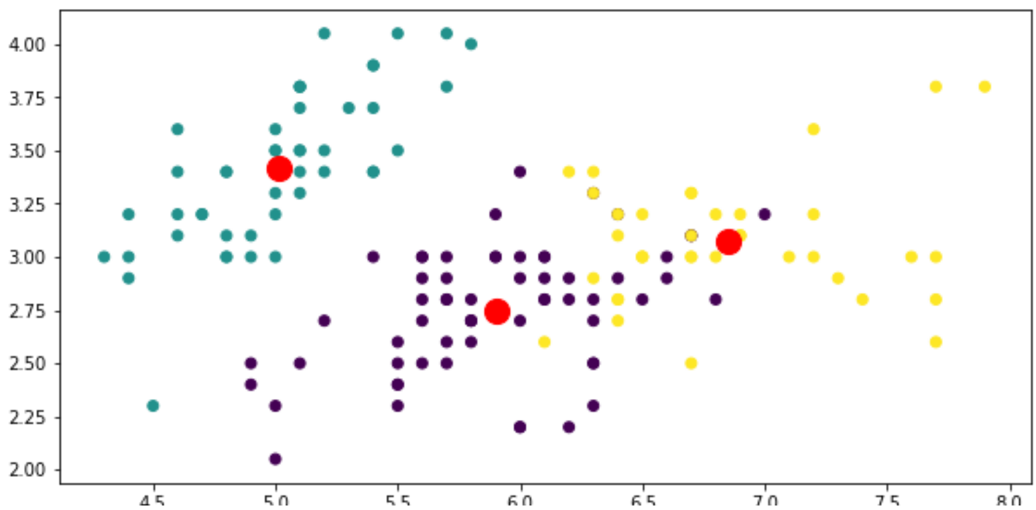
```
display(data['cluster'].value_counts(), data['Species'].value_
```

```
0    61
1    48
2    38
Name: cluster, dtype: int64
Iris-versicolor    50
Iris-virginica     49
Iris-setosa        48
Name: Species, dtype: int64
```

In [25]:

```
plt.figure(figsize=(10,5))
plt.scatter(data['SepalLengthCm'], data['SepalWidthCm'], c=da
plt.title('Predicted Clusters\n')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_cen
plt.show()
```

Predicted Clusters



In [26]:

```
data.loc[data['Species']=='Iris-setosa']['cluster'].value_cou
```

Out[26]:

```
1    48
Name: cluster, dtype: int64
```

In [27]:

```
data.loc[data['Species']=='Iris-versicolor']['cluster'].value_
```

Out[27]:

```
0    48
2     2
Name: cluster, dtype: int64
```

In [28]:

```
data.loc[data['Species']=='Iris-virginica']['cluster'].value_
```

Out[28]:

```
2    36
0    13
Name: cluster, dtype: int64
```

In [32]:

```
data['Species_encoded'] = data['Species'].apply(lambda x: 1 i
data
```

Out[32]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Specie
0	5.1	3.5	1.4	0.2	Iri setos
1	4.9	3.0	1.4	0.2	Iri setos
2	4.7	3.2	1.3	0.2	Iri setos
3	4.6	3.1	1.5	0.2	Iri setos
4	5.0	3.6	1.4	0.2	Iri setos
...	...	...	...	...	
145	6.7	3.0	5.2	2.3	Iri virginic
146	6.3	2.5	5.0	1.9	Iri virginic
147	6.5	3.0	5.2	2.0	Iri virginic
148	6.2	3.4	5.4	2.3	Iri virginic
149	5.9	3.0	5.1	1.8	Iri