GRIP : The Sparks Foundation

Data Science & Business Analytics

Author: Ume Salma Khan

Task 2 : From the given ‘Iris’ dataset, predict the optimum number of clusters and represent it visually.

In [ ]:
```
# Importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

1. Understand the data

In [25]:
```
# Importing the Data
iris_data = pd.read_csv('Iris.csv')
```

In [26]:
```
iris_data.head()
```

Out[26]:

	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 [27]:
```
iris_data.tail()
```

Out[27]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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 [29]:
```
iris_data.shape
```

Out[29]:
```
(150, 6)
```

In [31]:
```
iris_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 [32]:
```
iris_data.describe()
```

Out[32]:

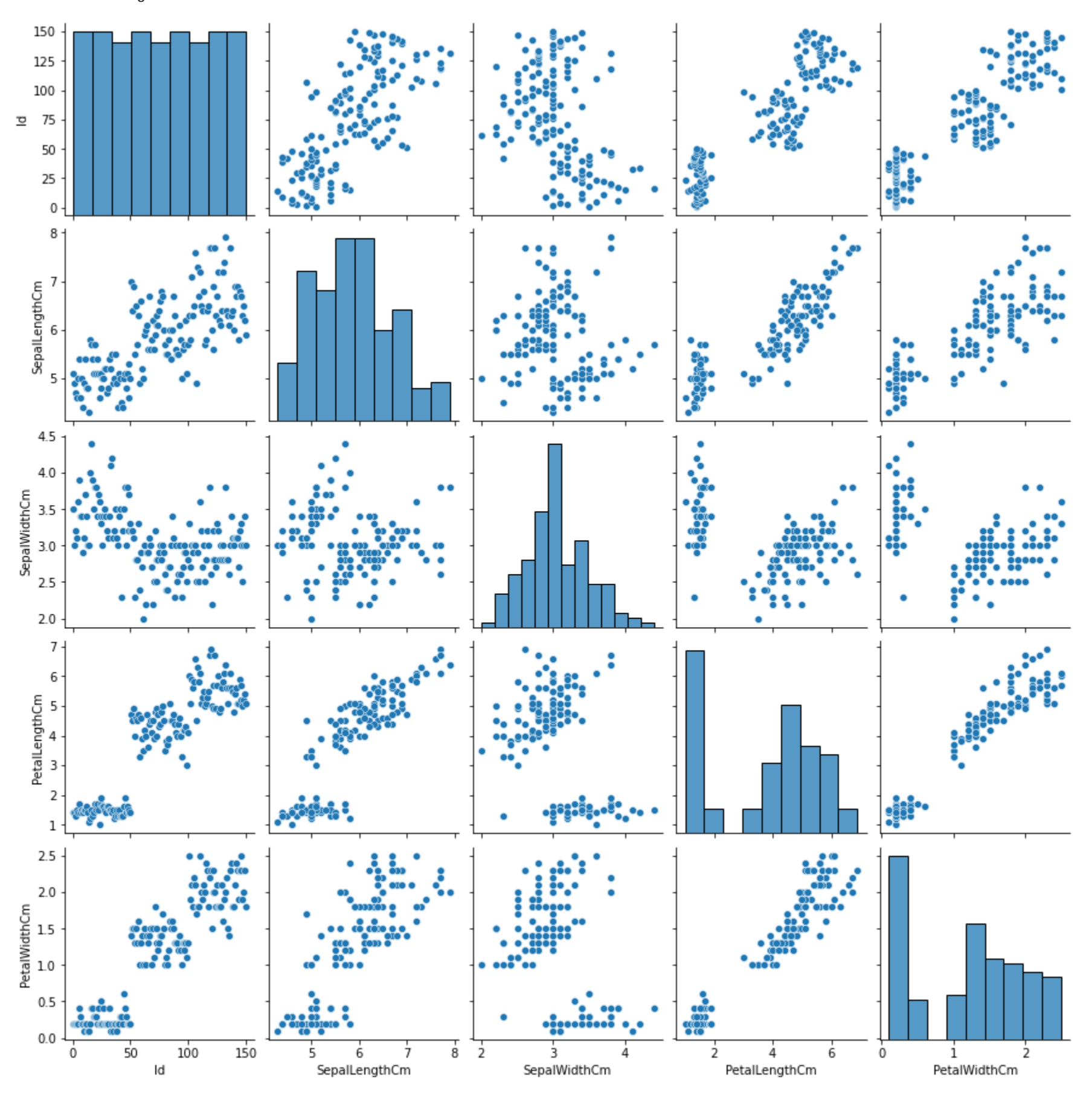
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	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 [34]:
```
iris_data['Species'].value_counts()
```

Out[34]:
```
Iris-setosa        50
Iris-virginica     50
Iris-versicolor    50
Name: Species, dtype: int64
```

In [39]:
```
sns.pairplot(iris_data)
```

Out[39]:
```
<seaborn.axisgrid.PairGrid at 0x4c7fa48>
```



2. Checking for NULL values or Cleaning the data

In [40]:
```
iris_data.isnull().sum()
```

Out[40]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

Their is no Null values are present. Thus, our data is clean.

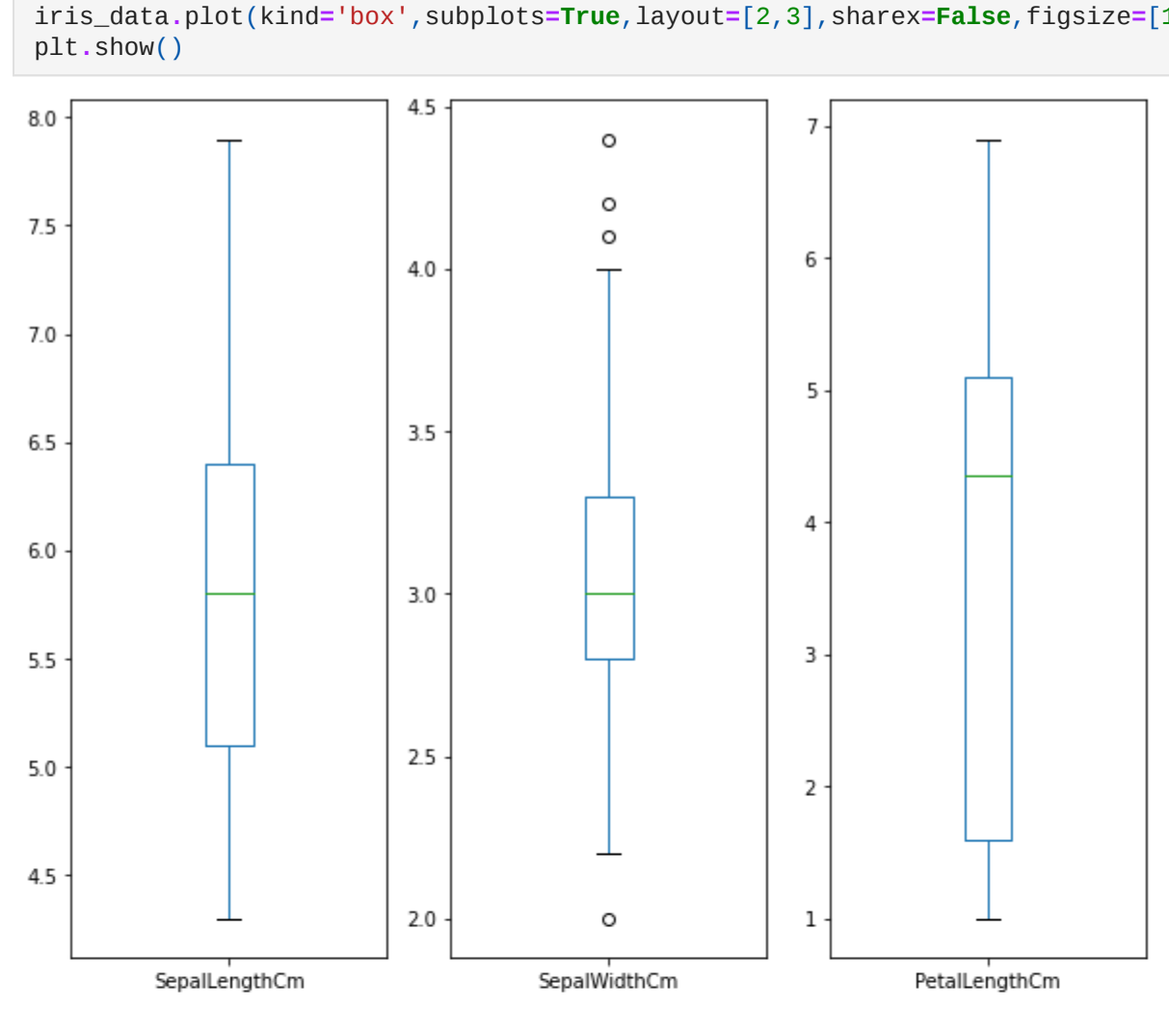
In [41]:
```
#Drop the and ID columns
iris_data = iris_data.drop(['Id'],axis=1)
```

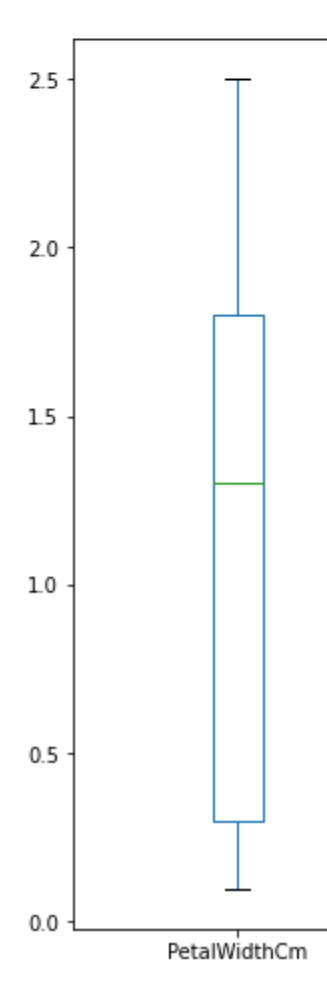
In [42]:
```
iris_data.head()
```

Out[42]:

	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 [43]:
```
#Checking for the outliers and removing it with IQR Method
iris_data.plot(kind='box',subplots=True,layout=[2,3],sharex=False,figsize=[10,18])
plt.show()
```



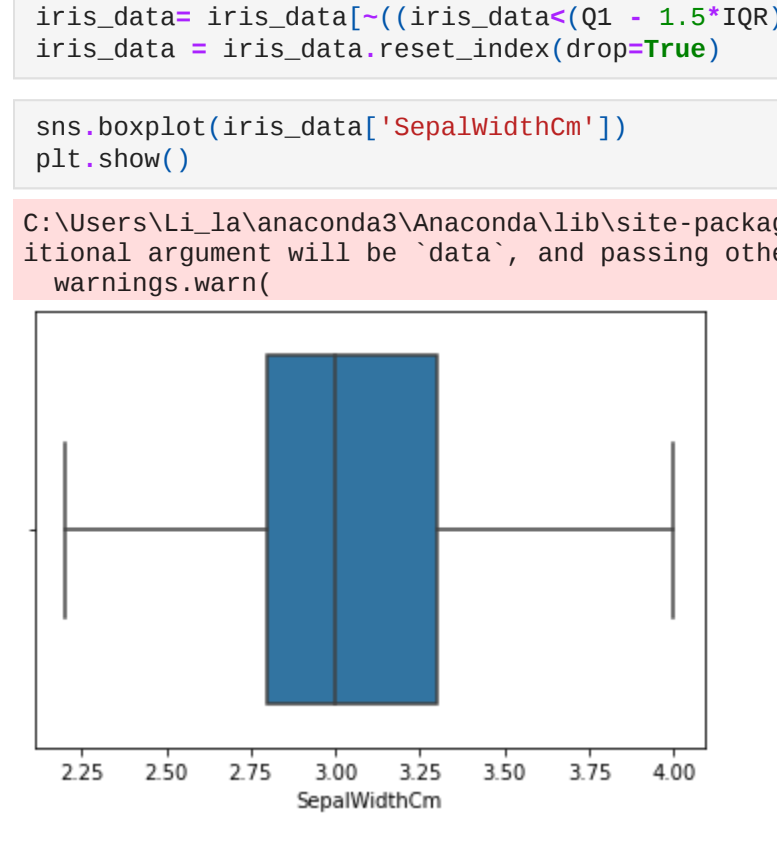


As you can see from the box plot only Sepalwidth contains the outliers. So, we have to remove those outliers.

In [45]:
```
#Calculate the first quartile
Q1 = iris_data.quantile(0.25)
#Calculate the third quartile
Q3 = iris_data.quantile(0.75)
#Calculate IQR
IQR = Q3-Q1
iris_data= iris_data[~((iris_data<(Q1 - 1.5*IQR)) | (iris_data> (Q3 + 1.5*IQR))).any(axis=1)]
iris_data = iris_data.reset_index(drop=True)
```

In [46]:
```
sns.boxplot(iris_data['SepalWidthCm'])
plt.show()
```

```
C:\Users\Li_la\anaconda3\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid pos
itional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Successfuly removed the outliers in the given dataset.

3. Scaling the feature before doing clustering

In [47]:
```
X = iris_data.iloc[:,0:4]
X
```

Out[47]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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
...	...	...	...	...
141	6.7	3.0	5.2	2.3
142	6.3	2.5	5.0	1.9
143	6.5	3.0	5.2	2.0
144	6.2	3.4	5.4	2.3
145	5.9	3.0	5.1	1.8

146 rows × 4 columns

In [48]:
```
#Scale the feature with StandardScaler
sc = StandardScaler()
x = sc.fit_transform(X)
x_scaled = pd.DataFrame(x,columns=X.columns)
x_scaled.head()
```

Out[48]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-0.910515	1.175789	-1.374878	-1.345899
1	-1.151122	-0.093924	-1.374878	-1.345899
2	-1.391729	0.413961	-1.431986	-1.345899
3	-1.512032	0.160019	-1.317771	-1.345899
4	-1.030819	1.429732	-1.374878	-1.345899

4. Perform K-Means clustering with K=3

In [49]:
```
kmeans = KMeans(n_clusters = 3, init ='k-means++',random_state=101)
```

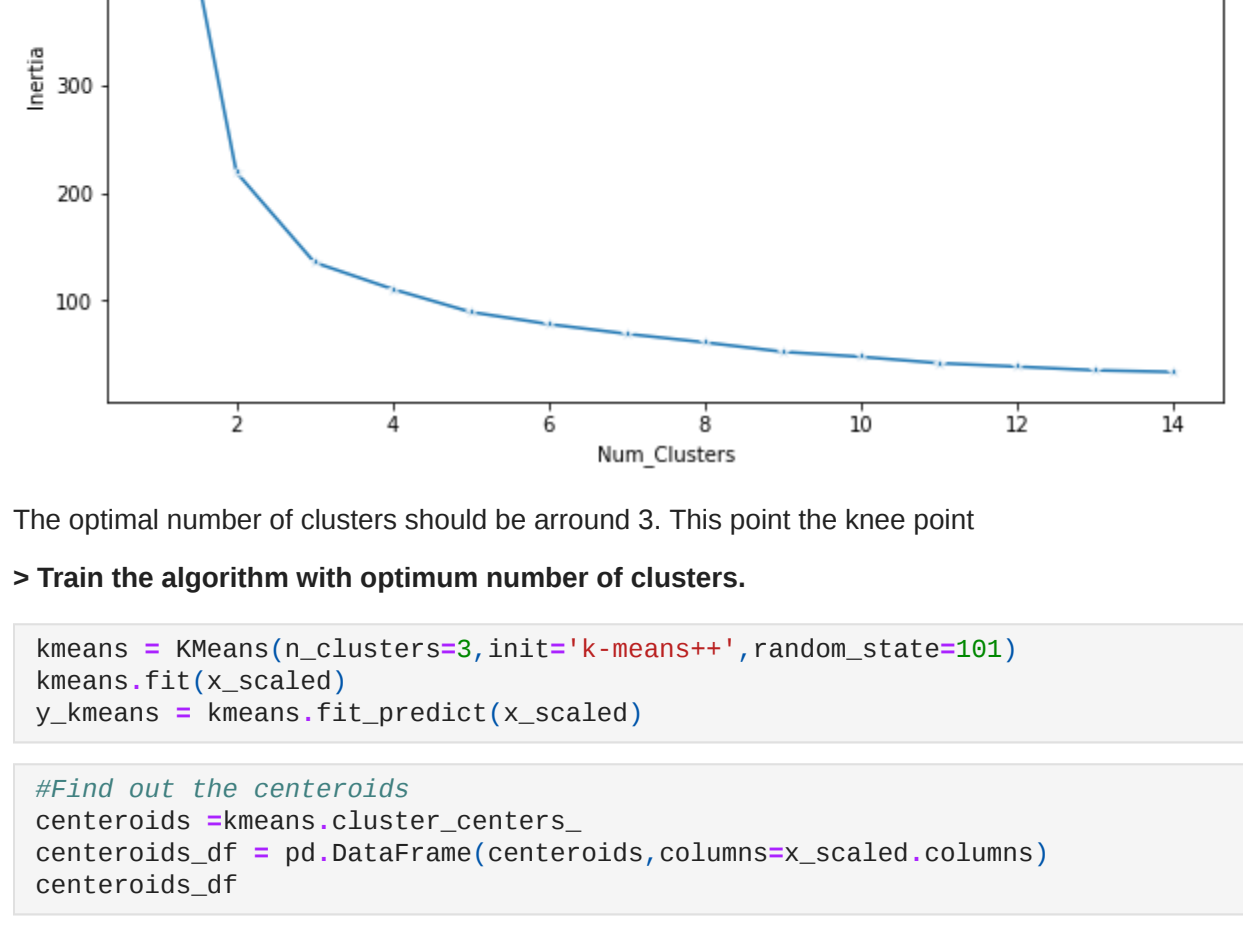
In [50]:
```
cluster_range = range(1,15)
cluster_errors = []
for num_clusters in cluster_range:
    kmeans = KMeans(n_clusters = num_clusters,init = 'k-means++',random_state = 1)
    kmeans.fit(x_scaled)
    cluster_errors.append(kmeans.inertia_)
cluster_df = pd.DataFrame({'Num_Clusters':cluster_range,'Inertia':cluster_errors})
cluster_df
```

```
C:\Users\Li_la\anaconda3\Anaconda\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 chun
ks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[50]:

	Num_Clusters	Inertia
0	1	584.000000
1	2	218.934500
2	3	135.398152
3	4	110.671976
4	5	89.442823
5	6	78.207029
6	7	69.202567
7	8	61.303155
8	9	52.595926
9	10	47.958254
10	11	42.050529
11	12	38.832888
12	13	35.464729
13	14	33.859195

In [51]:
```
#Visualize elbow to check value of K
plt.figure(figsize=(10,6))
sns.lineplot(x = cluster_df['Num_Clusters'],y = cluster_df['Inertia'],marker='*')
plt.show()
```



The optimal number of clusters should be arround 3. This point the knee point

> Train the algorithm with optimum number of clusters.

In [52]:
```
kmeans = KMeans(n_clusters=3,init='k-means++',random_state=101)
kmeans.fit(x_scaled)
y_kmeans = kmeans.fit_predict(x_scaled)
```

In [53]:
```
#Find out the centeroids
centeroids =kmeans.cluster_centers_
centeroids_df = pd.DataFrame(centeroids,columns=x_scaled.columns)
centeroids_df
```

Out[53]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	0.011095	-0.878425	0.363836	0.294282
1	-1.058975	0.835398	-1.338427	-1.286934
2	1.143036	0.230886	0.989099	1.023397

In [54]:
```
#Create dataframe for labels
df_labels = pd.DataFrame(kmeans.labels_,columns=list(['Label']))
df_labels['Label'] = df_labels['Label'].astype('object')
```

In [55]:
```
df_labels.head()
```

Out[55]:

	Label
0	1
1	1
2	1
3	1
4	1

In [58]:
```
#Join the label dataframe with iris dataset
iris_data_df = iris_data.join(df_labels)
```

In [68]:
```
iris_data_df.head()
```

Out[68]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	Label
0	5.1	3.5	1.4	0.2	Iris-setosa	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1

5. Compare the Observed and Actual values

In [69]:
```
print('Actual values:')
print(iris_data['Species'].value_counts())

print('Observed Values:')
print(iris_data_df['Label'].value_counts())
```

```
Actual values:
Iris-virginica     50
Iris-versicolor    49
Iris-setosa        47
Name: Species, dtype: int64
Observed Values:
0    56
1    47
2    43
Name: Label, dtype: int64
```

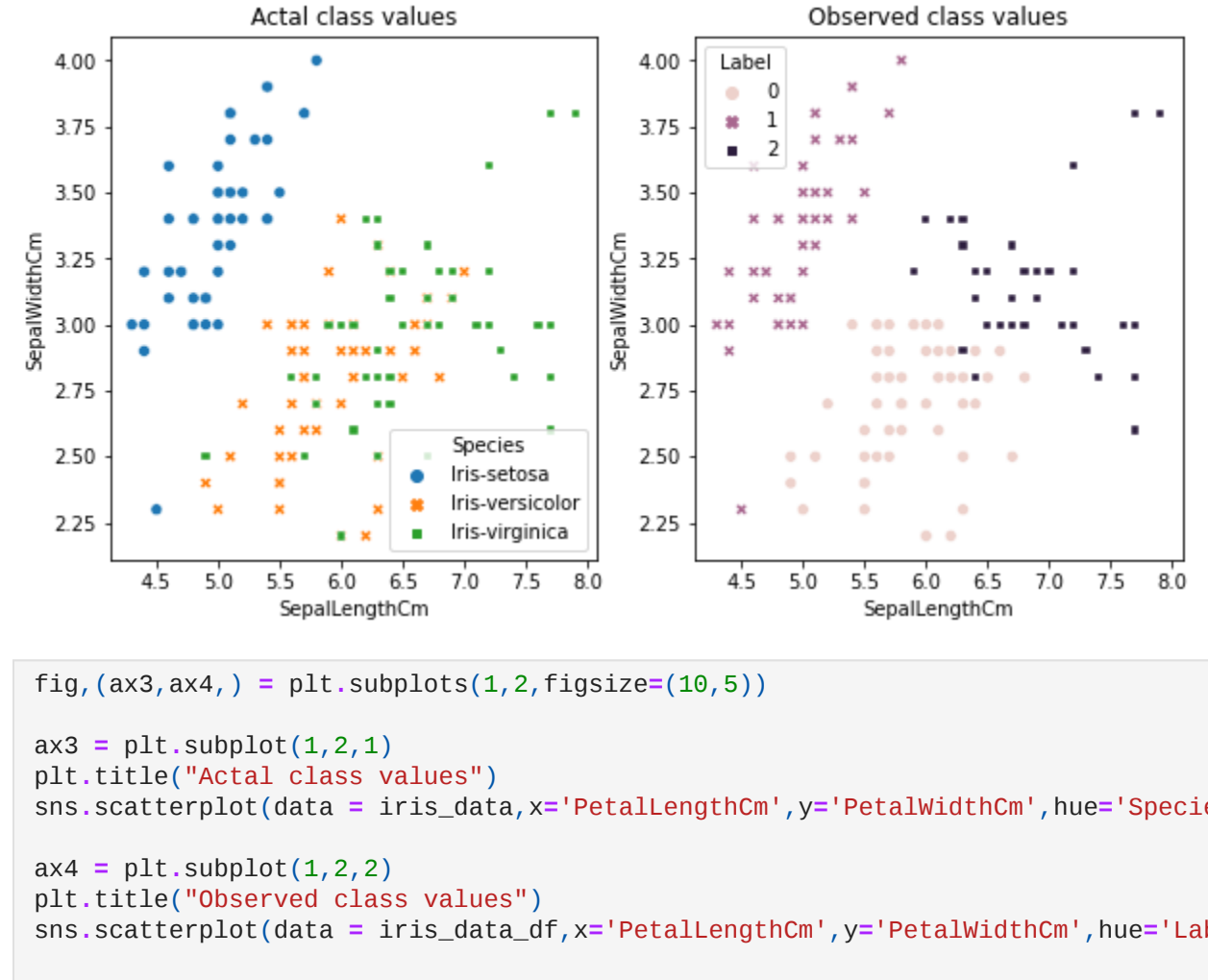
> Visualize the Actual and Observed values graphically

In [78]:
```
fig,(ax1,ax2,) = plt.subplots(1,2,figsize=(10,5))

ax1 = plt.subplot(1,2,1)
plt.title("Actal class values")
sns.scatterplot(data = iris_data,x='SepalLengthCm',y='SepalWidthCm',hue='Species',style='Species',ax=ax1)

ax2 = plt.subplot(1,2,2)
plt.title("Observed class values")
sns.scatterplot(data = iris_data_df,x='SepalLengthCm',y='SepalWidthCm',hue='Label',style='Label',ax=ax2)

plt.show()
```

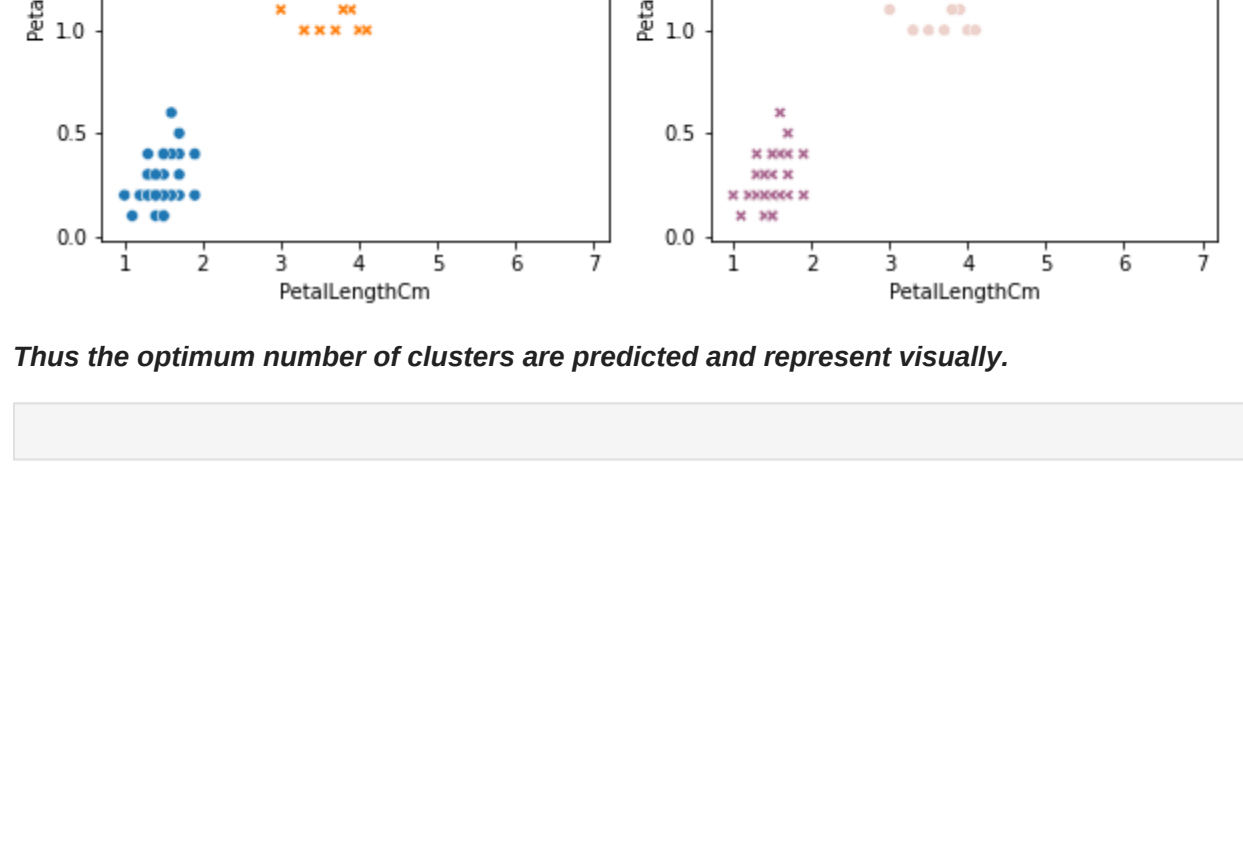


In [83]:
```
fig,(ax3,ax4,) = plt.subplots(1,2,figsize=(10,5))

ax3 = plt.subplot(1,2,1)
plt.title("Actal class values")
sns.scatterplot(data = iris_data,x='PetalLengthCm',y='PetalWidthCm',hue='Species',style='Species',ax=ax3)

ax4 = plt.subplot(1,2,2)
plt.title("Observed class values")
sns.scatterplot(data = iris_data_df,x='PetalLengthCm',y='PetalWidthCm',hue='Label',style='Label',ax=ax4)

plt.show()
```



Thus the optimum number of clusters are predicted and represent visually.

In [ ]: