Iris Species

Classify iris plants into three species in this classic dataset

The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species

In this study we try to cluster the Iris Dataset using Kmeans clustering (Unsupervised ML method)

Here I will try to run the K-Means on Iris dataset to classify our 3 classes of flowers, Iris setosa, Iris versicolor, Iris virginica (our classess) using the flowers sepal-length, sepal-width, petal-length and petal-width (our features)

In [66]:
```python
#import required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
```

In [4]:
```python
#read the iris dataset
df = pd.read_csv("C:/Users/pratrao/Downloads/archive/Iris.csv")
df.head()
```

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

In [5]:
```python
#go through the dataset
df.shape
```

Out[5]:
```
(150, 6)
```

In [7]:
```python
df.isnull().sum()
```

Out[7]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [8]:
```python
df.describe().T
```

Out[8]:

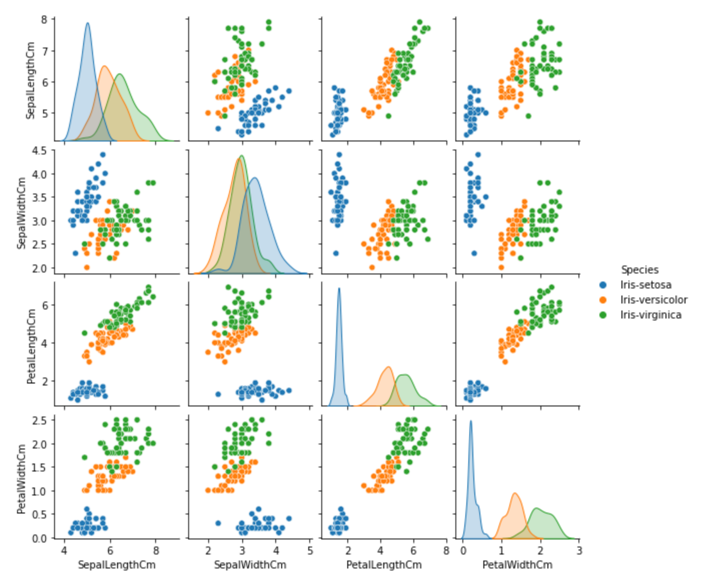
	count	mean	std	min	25%	50%	75%	max
Id	150.0	75.500000	43.445368	1.0	38.25	75.50	112.75	150.0
SepalLengthCm	150.0	5.843333	0.828066	4.3	5.10	5.80	6.40	7.9
SepalWidthCm	150.0	3.054000	0.433594	2.0	2.80	3.00	3.30	4.4
PetalLengthCm	150.0	3.758667	1.764420	1.0	1.60	4.35	5.10	6.9
PetalWidthCm	150.0	1.198667	0.763161	0.1	0.30	1.30	1.80	2.5

In [20]:
```python
df.value_counts("Species")
```

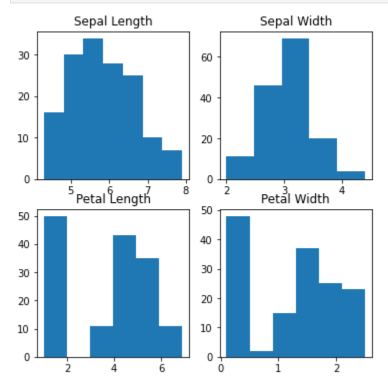
Out[20]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

In [22]:
```python
#data visualization of the dataset
sns.pairplot(df.drop(['Id'], axis = 1),
             hue='Species', height=2)
```

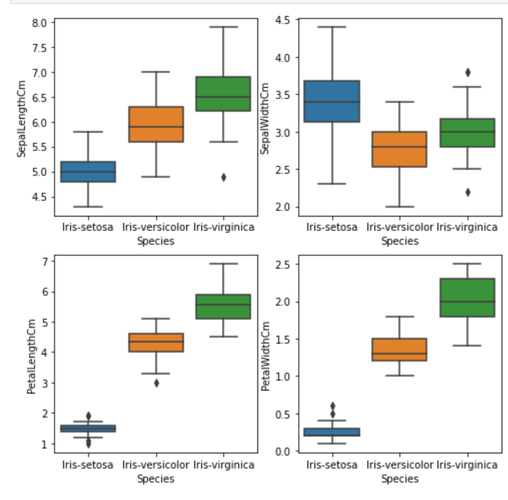
Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x27622a97400>
```



In [26]:
```python
fig, axes = plt.subplots(2, 2, figsize=(6,6))

axes[0,0].set_title("Sepal Length")
axes[0,0].hist(df['SepalLengthCm'], bins=7)

axes[0,1].set_title("Sepal Width")
axes[0,1].hist(df['SepalWidthCm'], bins=5);

axes[1,0].set_title("Petal Length")
axes[1,0].hist(df['PetalLengthCm'], bins=6);

axes[1,1].set_title("Petal Width")
axes[1,1].hist(df['PetalWidthCm'], bins=6);
```



In [30]:
```python
#boxplots to find outliers
def graph(y):
    sns.boxplot(x="Species", y=y, data=df)

plt.figure(figsize=(8,8))

# Adding the subplot at the specified
# grid position
plt.subplot(221)
graph('SepalLengthCm')

plt.subplot(222)
graph('SepalWidthCm')

plt.subplot(223)
graph('PetalLengthCm')

plt.subplot(224)
graph('PetalWidthCm')

plt.show()
```



In [32]:
```python
#encoding the target into numerical values
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Species'.
df['Species']= label_encoder.fit_transform(df['Species'])

df['Species'].unique()
```

Out[32]:
```
array([0, 1, 2])
```

In [39]:
```python
#normalizing the dataset
sc = StandardScaler()
df_norm = sc.fit_transform(df)
```

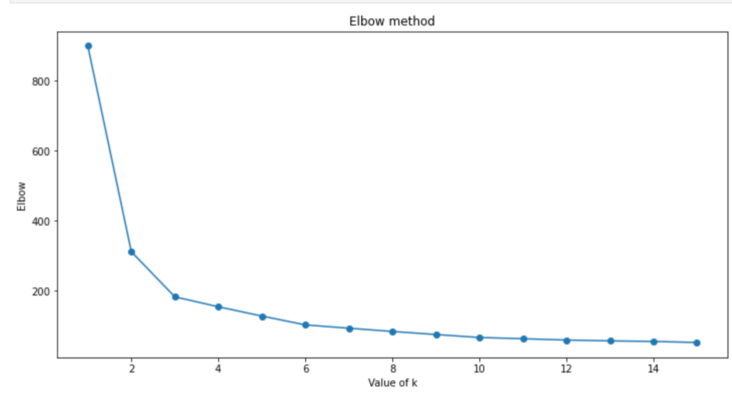
In [62]:
```python
#finding the best KMeans model by finding the right the number of clusters
cluster_errors = []

for num_cluster in range (1, 16):
    kmeans = KMeans(num_cluster, init = 'k-means++', max_iter = 10, n_init = 10, random_state = 5)
    kmeans.fit(df_norm)
    labels = kmeans.labels_
    centroids = kmeans.cluster_centers_
    cluster_errors.append(kmeans.inertia_)

clusters_df = pd.DataFrame({'cluster_errors':cluster_errors})
clusters_df.index = clusters_df.index + 1
clusters_df.index.name = 'num_clusters'
clusters_df.head()
```

```
C:\Users\pratrao\Anaconda3\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 chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[62]:

num_clusters	cluster_errors
1	900.000000
2	311.227561
3	182.245751
4	153.787256
5	127.248574

In [63]:
```python
#Elbow point/method
plt.figure(figsize = (12,6))
plt.plot(range(1,16), cluster_errors, marker = 'o')
plt.xlabel('Value of k')
plt.ylabel('Elbow')
plt.title('Elbow method')
plt.show()
```



From the above Elbow method, it is evident that the best/optimal number of clusters is 3.

In [87]:
```python
#creating the best KMeans model
x = df.iloc[:,0:3].values
```

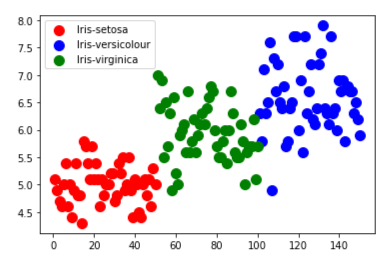
In [88]:
```python
kmeans_model = KMeans(n_clusters=3,init = 'k-means++', max_iter = 50, n_init = 10, random_state = 0)

y_kmeans = kmeans_model.fit_predict(x)
```

In [89]:
```python
kmeans.cluster_centers_
```

Out[89]:
```
array([[ 2.23596982e-01, -3.49918123e-01, -9.87459562e-01,
         1.63083946e-01,  6.15134139e-02,  1.11340443e-01],
       [-1.28174608e+00, -9.89537609e-01,  8.93215445e-01,
        -1.28440670e+00, -1.23409295e+00, -1.22474487e+00],
       [ 1.25736853e+00,  9.10105338e-01,  2.47858287e-01,
         9.96539870e-01,  1.32378673e+00,  1.22474487e+00],
       [ 7.26798255e-02,  2.58966625e-02, -2.33853114e-01,
         3.71388798e-01,  2.80166321e-01,  0.00000000e+00],
       [ 1.15776495e+00,  3.23752047e-01, -7.58270979e-01,
         8.04659688e-01,  8.66706723e-01,  1.22474487e+00],
       [-1.28174608e+00, -3.81390297e-01,  2.28824475e+00,
        -1.30065404e+00, -1.20028561e+00, -1.22474487e+00],
       [ 1.03283843e+00,  1.92656914e+00, -3.04937685e-01,
         1.42619186e+00,  1.03892863e+00,  1.22474487e+00],
       [ 1.02770632e+00,  2.12851559e+00,  1.57199748e+00,
         1.50201279e+00,  1.36030700e+00,  1.22474487e+00],
       [-1.56217959e+00, -1.43728174e+00, -2.57849018e-02,
        -1.35751975e+00, -1.36932229e+00, -1.22474487e+00],
       [ 8.46799213e-02, -8.19902590e-01, -1.66764404e+00,
        -1.85003090e-01, -2.17368644e-01,  1.54074396e-33],
       [-9.32762163e-01, -8.06439493e-01,  1.41772883e+00,
        -1.27176988e+00, -1.16689565e+00, -1.22474487e+00],
       [-2.30945240e-01,  9.57226177e-01, -4.78232795e-02,
         4.97385358e-01,  3.41391479e-01,  1.54074396e-33],
       [-1.40876596e-01,  2.62530386e-01, -1.74477836e+00,
         3.41952433e-01,  1.59520537e-01,  0.00000000e+00],
       [-8.57646641e-01, -1.29723056e+00,  1.69555264e-01,
        -1.32059397e+00, -1.32492882e+00, -1.22474487e+00],
       [-7.73666553e-01, -1.62768839e+00, -1.74477836e+00,
        -1.39813811e+00, -1.18150376e+00, -1.22474487e+00]])
```

In [91]:
```python
# Visualising the clusters - On the first two columns
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1],
            s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1],
            s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1],
            s = 100, c = 'green', label = 'Iris-virginica')

plt.legend()
```

Out[91]:
```
<matplotlib.legend.Legend at 0x27624b5a280>
```



In [93]:
```python
from sklearn.metrics import classification_report

target_names = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

print(classification_report(df['Species'], kmeans_model.labels_, target_names=target_names))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       0.00      0.00      0.00        50
 Iris-virginica       0.00      0.00      0.00        50

       accuracy                           0.33       150
      macro avg       0.33      0.33      0.33       150
   weighted avg       0.33      0.33      0.33       150
```

You can see in the classification report that, 91% of our data was predicted accurately. That's pretty good for an unsupervised algorithm.