Task 3: Unsupervised ML Algorithm K- Means Clustering

Method Used

K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k means clustering, we have the specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps: Reassign data points to the cluster whose centroid is closest. Calculate new centroid of each cluster. These two steps are repeated till the within cluster variation cannot be reduced any further. The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline

#Importing Sci-kit learn libraries

from sklearn.cluster import KMeans
```

In [2]:
```
# Load the iris dataset
df=pd.read_csv('Iris.csv',index_col=0)
```

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

Out[3]:
```
(150, 5)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

In [5]:
```
df.head()
```

Out[5]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
Id					
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 [6]:
```
df.describe()
```

Out[6]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
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

In [7]:
```
df.groupby('Species').sum()
```

Out[7]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Species				
Iris-setosa	250.3	170.9	73.2	12.2
Iris-versicolor	296.8	138.5	213.0	66.3
Iris-virginica	329.4	148.7	277.6	101.3

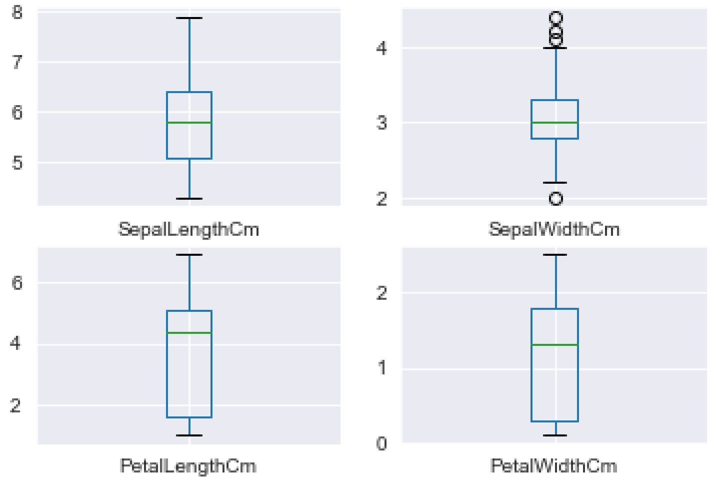
In [8]:
```
df.groupby('Species').size()
```

Out[8]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Data Visualization

In [9]:
```
plt.figure(figsize=(20,20))
sns.set_style('darkgrid')
df.plot(kind='box',layout=(2,2),sharex=False,sharey=False,subplots=True)

plt.show()
```

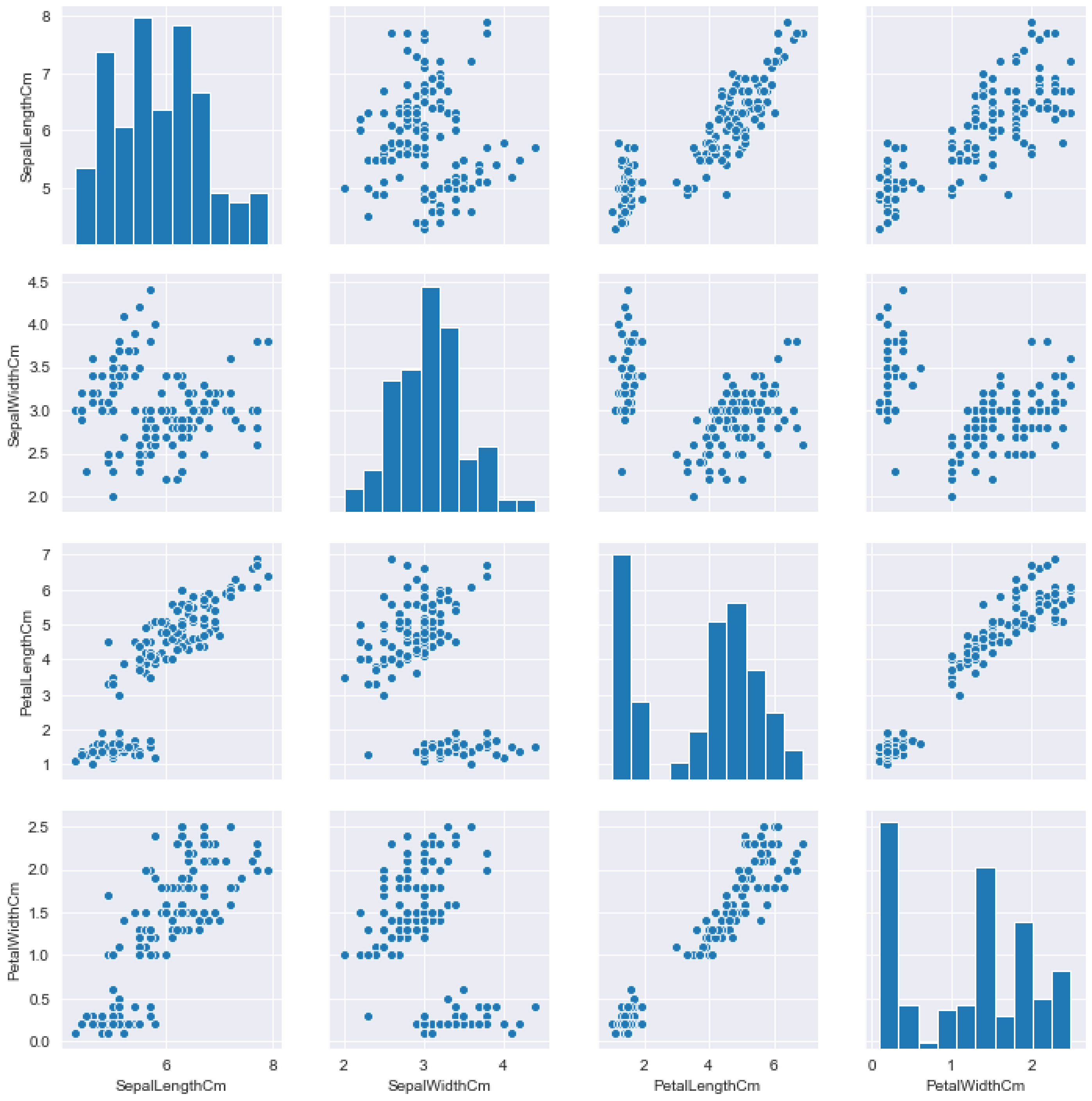
```
<Figure size 1440x1440 with 0 Axes>
```



Pair-plot

In [10]:
```
sns.pairplot(df ,palette='viridis')
```

```
Out[10]: <seaborn.axisgrid.PairGrid at 0x1fe4c8fda48>
```


SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm

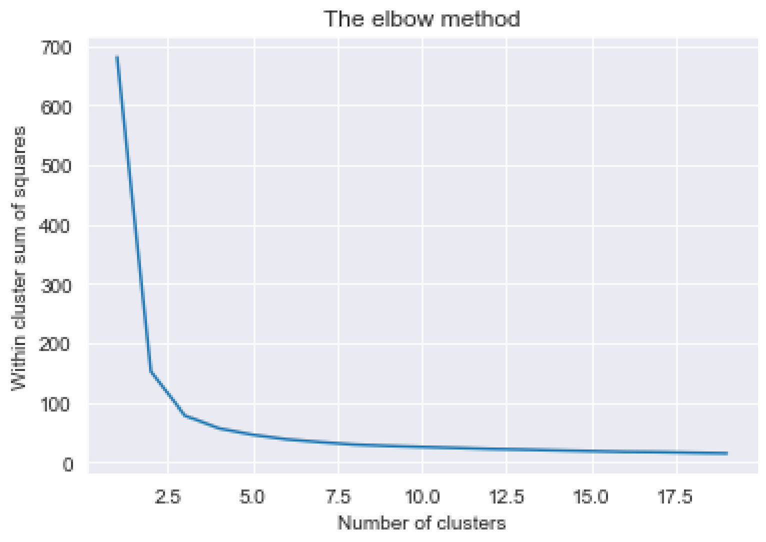
How do you find the optimum number of clusters for K Means? How does one determine the value of K?

We will use inertia here. The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum-of-squares criterion:

\sum*{i=0}^{n}\min*{\mu_j \in C}(||x_i - \mu_j||^2)

Inertia can be recognized as a measure of how internally coherent clusters are.

In [11]:
```
# Finding the optimum number of clusters for k-means classification

x = df.iloc[:, [0, 1, 2, 3]].values


clusters = []

for i in range(1, 20):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
                    max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(x)
    clusters.append(kmeans.inertia_)

# Plotting the results onto a line graph,
# `allowing us to observe 'The elbow'
plt.plot(range(1, 20), clusters)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Within cluster sum of squares')
plt.show()
```



This is called 'The elbow method' from the above graph, the optimum clusters is where the elbow occurs. This is when the within cluster sum of squares (WCSS) doesn't decrease significantly with every iteration.

From this we choose the number of clusters as **3**.

K Means Cluster Creation

Applying kmeans to the dataset / Creating the kmeans classifier

```
In [12]: kmeans = KMeans(n_clusters = 3, init = 'k-means++',max_iter = 300, n_init = 10, random_state = 0)
```

Fit the model to all the data except for the Species label.

```
In [13]: kmeans.fit(df.drop('Species',axis=1))

Out[13]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
                n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
                random_state=0, tol=0.0001, verbose=0)
```

What are the cluster center vectors?

```
In [14]: kmeans.cluster_centers_

Out[14]: array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
                [5.006     , 3.418     , 1.464     , 0.244     ],
                [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

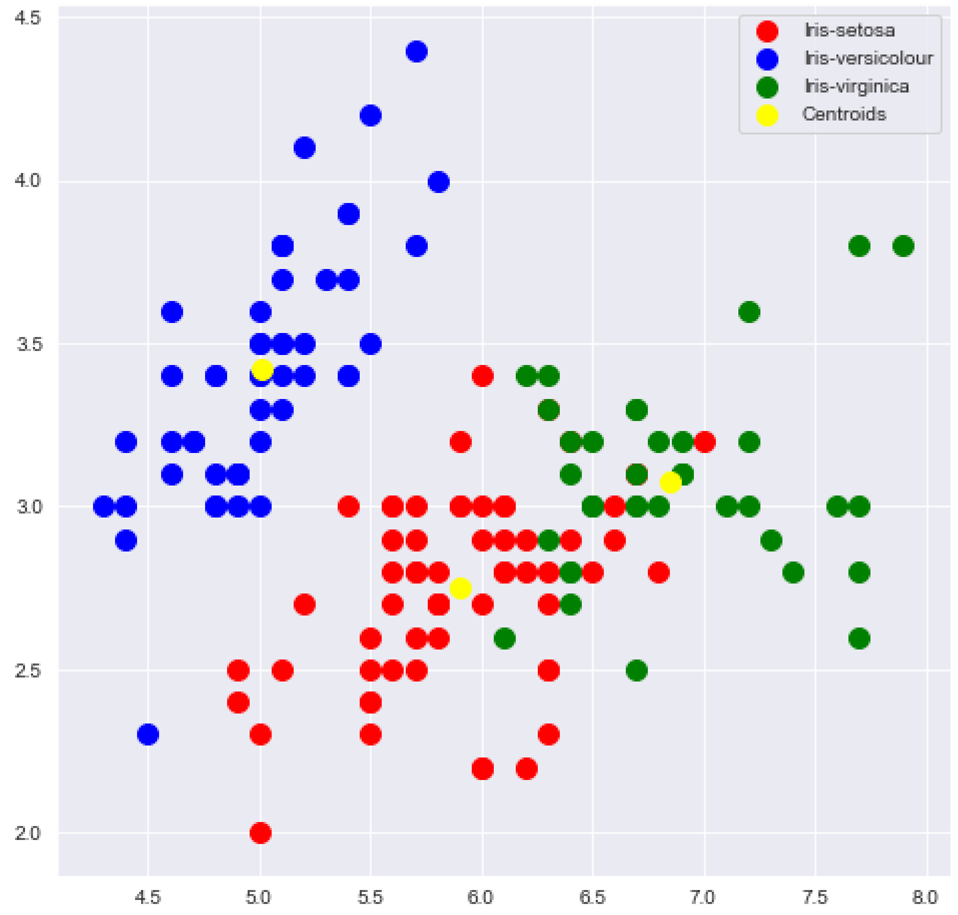
In [15]:
```
kmeans.labels_
```

Out[15]:
```
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, 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, 0, 2, 2, 2, 0, 2, 2, 0])
```

In [16]:
```
y_kmeans = kmeans.fit_predict(x)
```

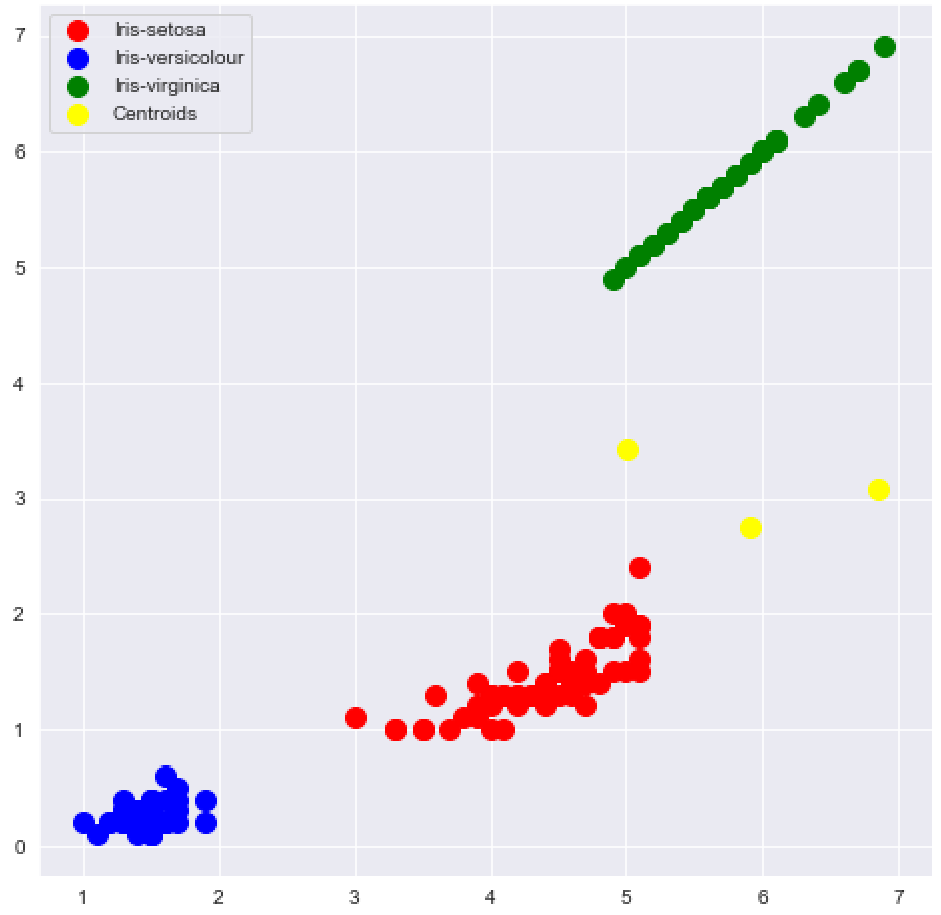
In [17]:
```
# Visualising the clusters - On the first two columns

plt.figure(figsize=(8,8))
sns.set_style('darkgrid')

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')

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1],
            s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
```

```
Out[17]: <matplotlib.legend.Legend at 0x1fe4ca27d08>
```


Iris-setosa
Iris-versicolour
Iris-virginica
Centroids
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

In [18]:
```
#Visualising the clusters - On the last two columns

plt.figure(figsize=(8,8))
sns.set_style('darkgrid')

plt.scatter(x[y_kmeans == 0, 2], x[y_kmeans == 0, 3],
            s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 2], x[y_kmeans == 1, 3],
            s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 2], x[y_kmeans == 2, 2],
            s = 100, c = 'green', label = 'Iris-virginica')

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1],
            s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
```

```
Out[18]: <matplotlib.legend.Legend at 0x1fe4e73ab88>
```



So, the optimum number of clusters are 3.