The Sparks Foundation

Graduate Rotational Internship Program (GRIP) July2021 Batch¶

Data Science & Business Analytics

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

Prediction using Unsupervised ML

- By:Prathamesh Parab

Importing libraries

In [1]:

```
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
```

Load the iris dataset

In [2]:

```
df=pd.read_csv('Iris.csv')
df.drop(['Id'],axis=1,inplace=True)
```

In [3]:

```
df.head() #diplay the fisrt 5 dataset.
```

Out[3]:

	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 [4]:

```
df.shape
```

Out[4]:

```
(150, 5)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [6]:

```
df.isnull().sum()
```

Out[6]:

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

In [7]:

```
df.describe()
```

Out[7]:

	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 [8]:

```
df.drop_duplicates(inplace=True)
```

Label Encoding

In [9]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Species']=le.fit_transform(df['Species'])
df['Species'].value_counts()
```

Out[9]:

```
1    50
2    49
0    48
Name: Species, dtype: int64
```

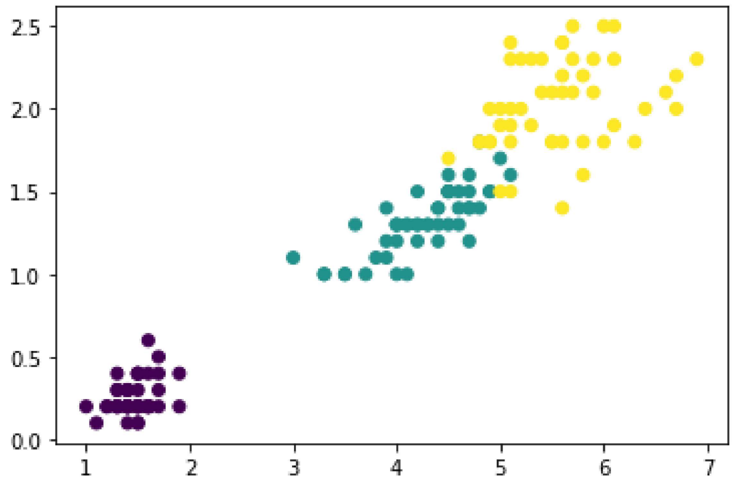
PetalLengthCm vs PetalWidthCm¶

In [10]:

```
plt.scatter(df['PetalLengthCm'],df['PetalWidthCm'],c=df.Species.values)
```

Out[10]:

```
<matplotlib.collections.PathCollection at 0x173b3051370>
```



In [11]:

```
#Finding the correlation between the features
df.corr()
```

Out[11]:

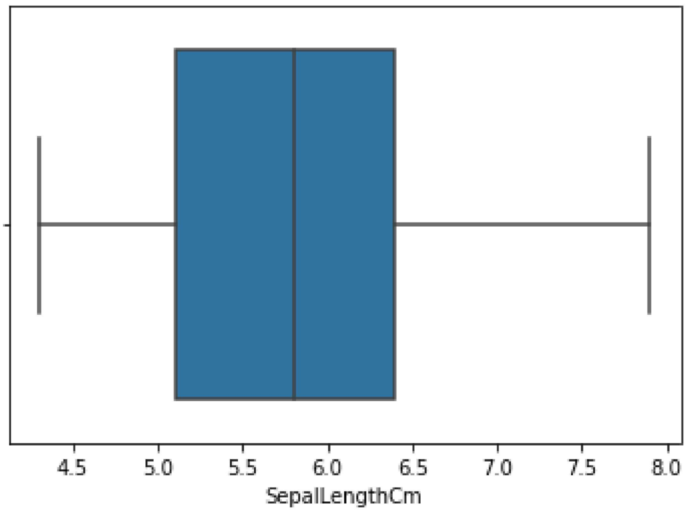
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
SepalLengthCm	1.000000	-0.109321	0.871305	0.817058	0.782904
SepalWidthCm	-0.109321	1.000000	-0.421057	-0.356376	-0.418348
PetalLengthCm	0.871305	-0.421057	1.000000	0.961883	0.948339
PetalWidthCm	0.817058	-0.356376	0.961883	1.000000	0.955693
Species	0.782904	-0.418348	0.948339	0.955693	1.000000

In [12]:

```
# Checking outliers
sns.boxplot(x=df['SepalLengthCm'])
```

Out[12]:

<AxesSubplot:xlabel='SepalLengthCm'>



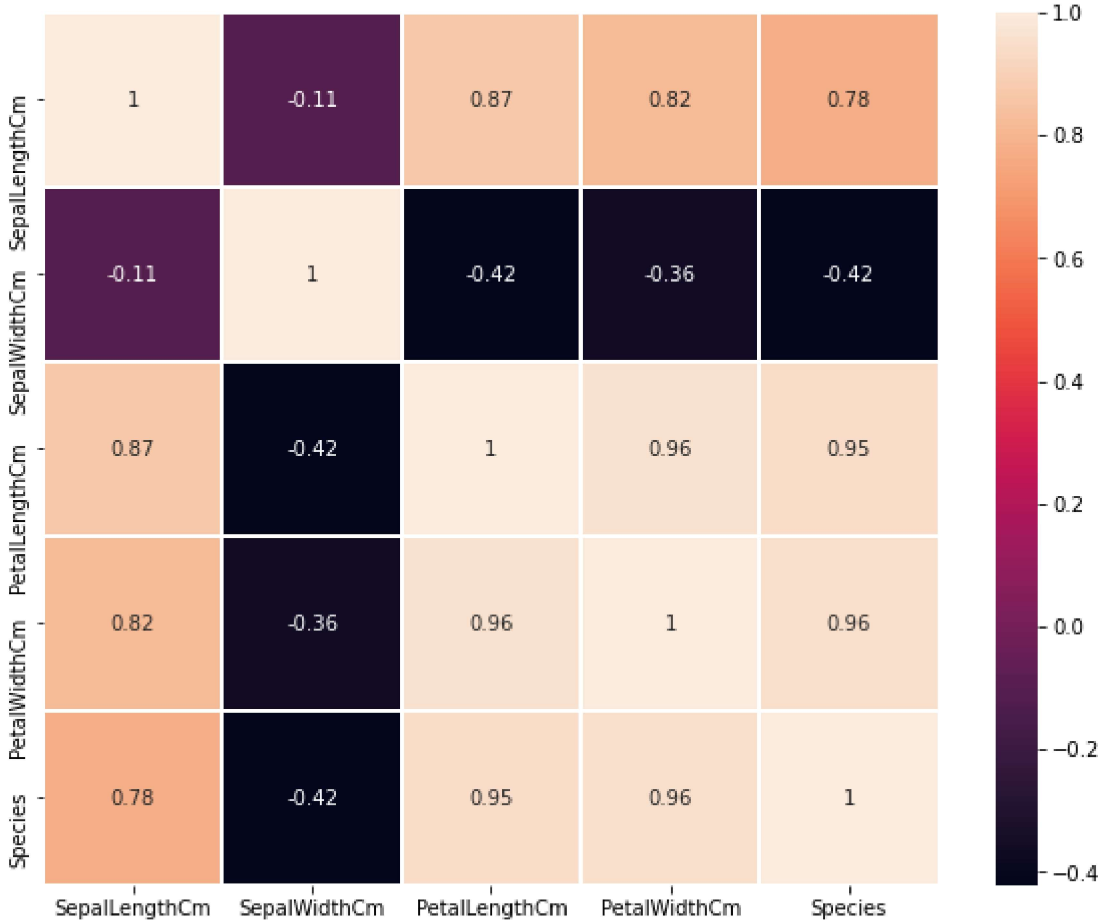
Data Visualization

In [13]:

```
fig=plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),linewidths=1,annot=True)
```

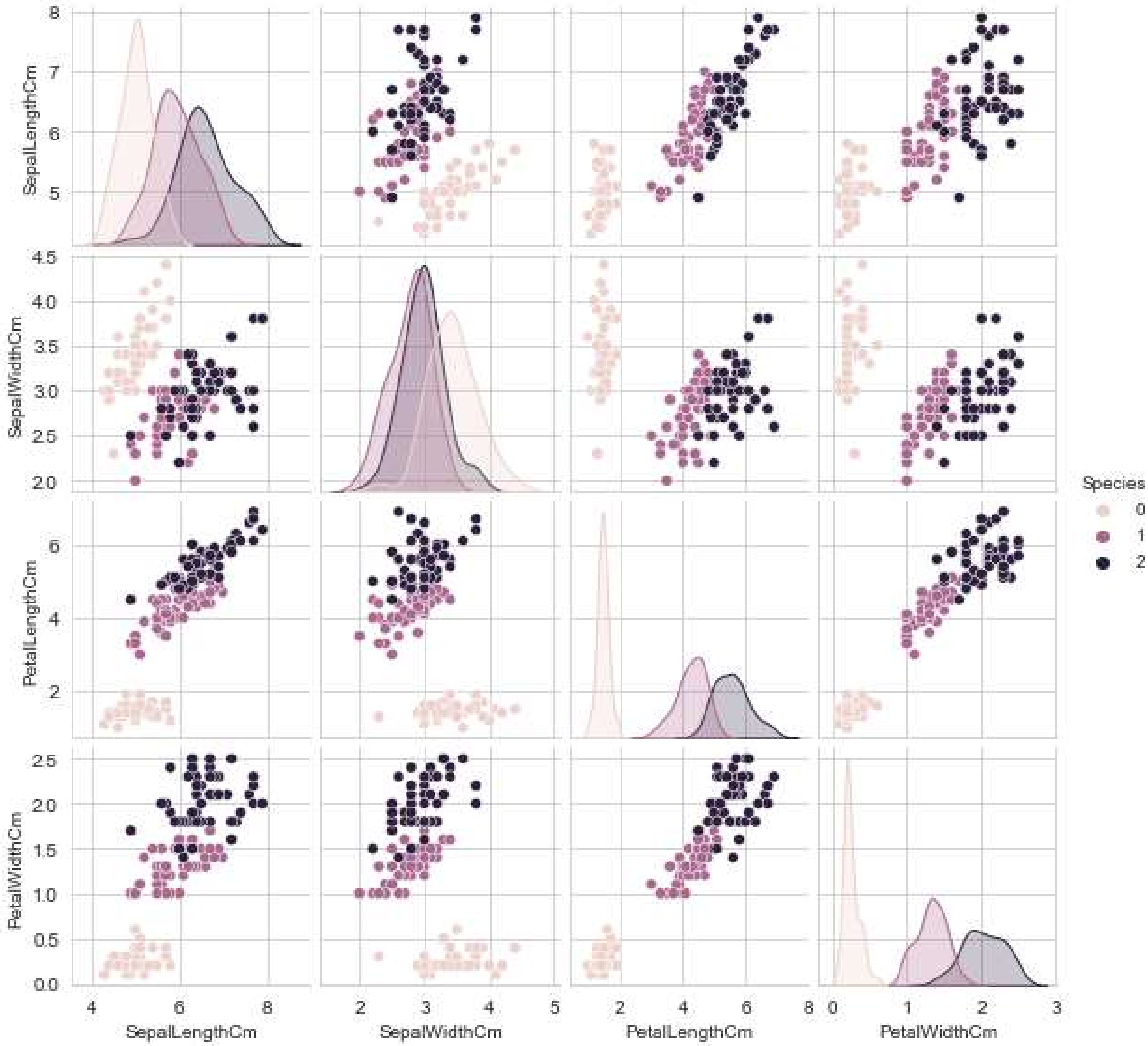
Out[13]:

<AxesSubplot:>



In [14]:

```
#Visualizing clearly about sample relation between features
sns.set_style("whitegrid");
sns.pairplot(df, hue="Species", height=2);
plt.show()
```



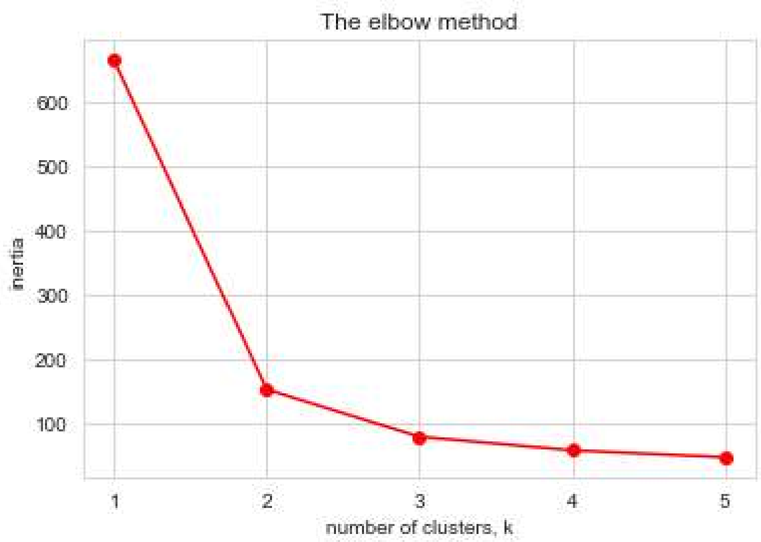
We can see that Species is mainly depend on Petal Length and Petal Width.

using petal_length and petal_width

Elbow Method using within-cluster-sum-of-squares(wcss)

In [17]:

```
from sklearn.cluster import KMeans
ks = range(1, 6)
inertias = []
x = df.iloc[:, [0, 1, 2, 3]].values
for k in ks:

    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(x)

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

# Plot ks vs inertias
plt.plot(ks, inertias, '-o',color='red')
plt.title('The elbow method')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



from the above graph, we choose the cluster where inertia doesn't decrease significantly with every iteration.

From this we choose the number of clusters as 3, we can also use Hierarchical clustering to get number of cluster to be used

Initialization using K-means++

In [18]:

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 5)
y_kmeans = kmeans.fit_predict(df)
y_kmeans
```

Out[18]:

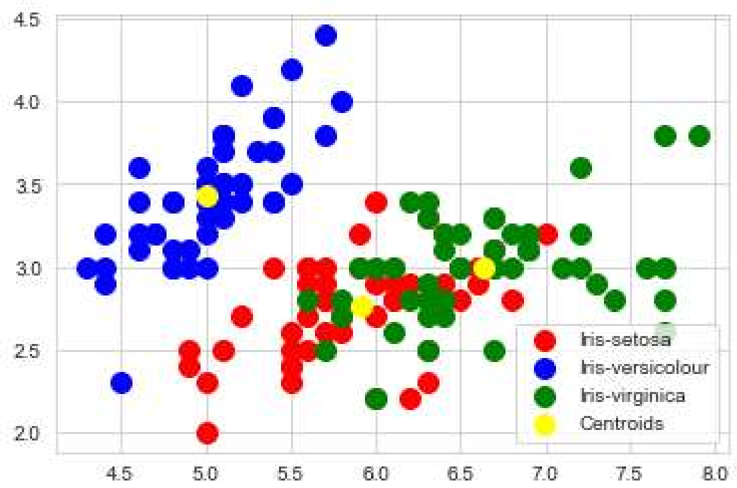
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [19]:

```
# Calculate the cluster labels: labels
labels = kmeans.predict(df)

# Visualising the clusters - On the first two columns
plt.scatter(x[labels == 0, 0], x[labels == 0, 1],
            s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[labels == 1, 0], x[labels == 1, 1],
            s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[labels == 2, 0], x[labels == 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[19]:

```
<matplotlib.legend.Legend at 0x173b54ffaf0>
```



We can see that our predicted graph is quite similar to the actual one.

Validating the model

In [20]:

```
# Create a DataFrame with labels and species as columns: df
df2 = pd.DataFrame({'labels':labels,'species':df['Species']})

# Create crosstab: ct
ct = pd.crosstab(df2['labels'],df2['species'])

print(ct)
```

```
species   0   1   2
labels
0         0  50   1
1        48   0   0
2         0   0  48
```

To cluster more accurately we normalize the data and apply clustering again.

In [21]:

```
# Perform the necessary imports
from sklearn.preprocessing import Normalizer
from sklearn.cluster import KMeans

# Create scaler: scaler
scaler = Normalizer()

x = scaler.fit_transform(x)

# Create KMeans instance: kmeans
kmeans = KMeans(n_clusters=3)

# Create pipeline: pipeline
kmeans.fit(x)
```

Out[21]:

```
KMeans(n_clusters=3)
```

In [22]:

```
# Create a DataFrame with labels and species as columns: df
df2 = pd.DataFrame({'labels':labels,'species':df['Species']})

# Create crosstab: ct
ct = pd.crosstab(df2['labels'],df2['species'])

# Display ct
print(ct)
```

```
species   0   1   2
labels
0         0  50   1
1        48   0   0
2         0   0  48
```

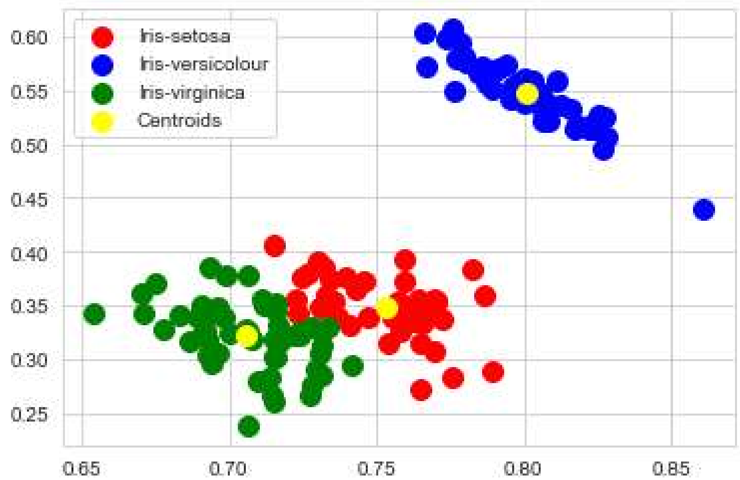
Therefore we can see the effect of doing normalize on data and the clustering has been improved compared to unnormalized data

In [23]:

```
# Calculate the cluster labels: labels
labels = kmeans.predict(x)

# Visualising the clusters - On the first two columns
plt.scatter(x[labels == 0, 0], x[labels == 0, 1],
            s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[labels == 1, 0], x[labels == 1, 1],
            s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[labels == 2, 0], x[labels == 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[23]:

```
<matplotlib.legend.Legend at 0x173b5590e50>
```



In [ ]:

```

```