GRIPJUNE21 @ The Spark Foundation

Author : Cheekireddy Dhamini

Prediction of the optimum number of clusters using Unsupervised ML

Importing the packages

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

Loading the Dataset

In [2]:
```
iris = sns.load_dataset("iris")
iris.head()
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Exploratry Data Analysis

In [3]:
```
iris.info()
```

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

In [4]:
```
iris['species'].unique()
```

Out[4]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

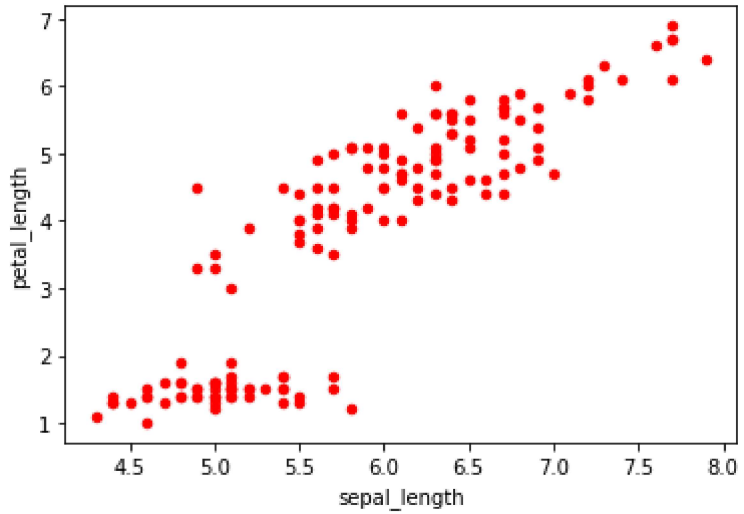
In [5]:
```
iris.describe()
```

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
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

Data Visualization¶

In [6]:
```
iris.plot(kind='scatter',x='sepal_length',y='petal_length',c='red')
plt.show()
```



Data Pre-processing

In [7]:
```
# Data Preprocessing
iris.drop(columns=["species"],axis=1,inplace=True)
```

In [8]:
```
iris.head()
```

Out[8]:

	sepal_length	sepal_width	petal_length	petal_width
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

In [9]:
```
x = iris.iloc[:,[0,1,2,3]].values
```

In [10]:
```
from sklearn.cluster import KMeans
```

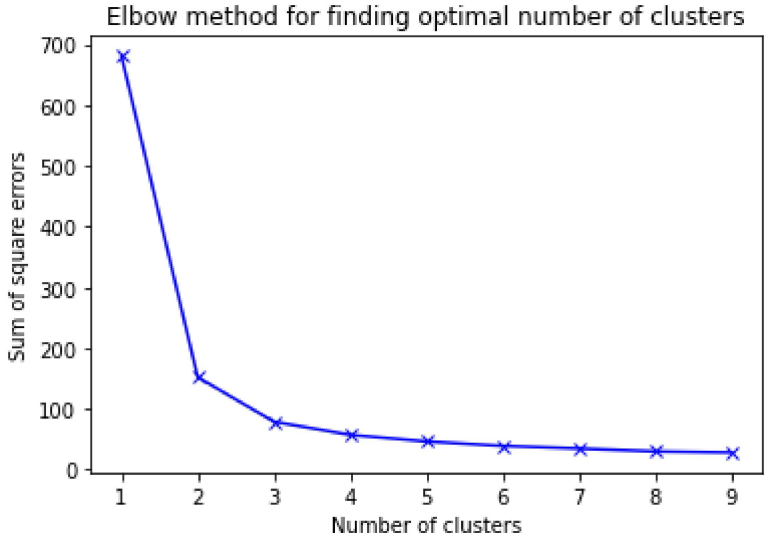
In [11]:
```
sse = []
k_range = range(1,10)
for i in k_range:
    km = KMeans(n_clusters=i)
    km.fit(x)
    sse.append(km.inertia_)
```

```
C:\Users\mr\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to hav
e 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(
```

In [12]:
```
sse
```

Out[12]:
```
[681.3705999999996,
 152.34795176035797,
 78.851441426146,
 57.25552380952379,
 46.44618205128204,
 39.03998724608725,
 34.83091630591632,
 30.064593073593088,
 28.271132317974427]
```

In [13]:
```
#Plotting the sse result graph
k_range = range(1,10)
plt.plot(k_range,sse,'bx-')
plt.title("Elbow method for finding optimal number of clusters")
plt.xlabel("Number of clusters")
plt.ylabel('Sum of square errors')
plt.show()
```



From the above graph we can say that optimal value of cluster for KMeans is 3, since it is the place where elbow occurs

In [14]:
```
km = KMeans(n_clusters=3)
y_pred1 = km.fit_predict(x)
```

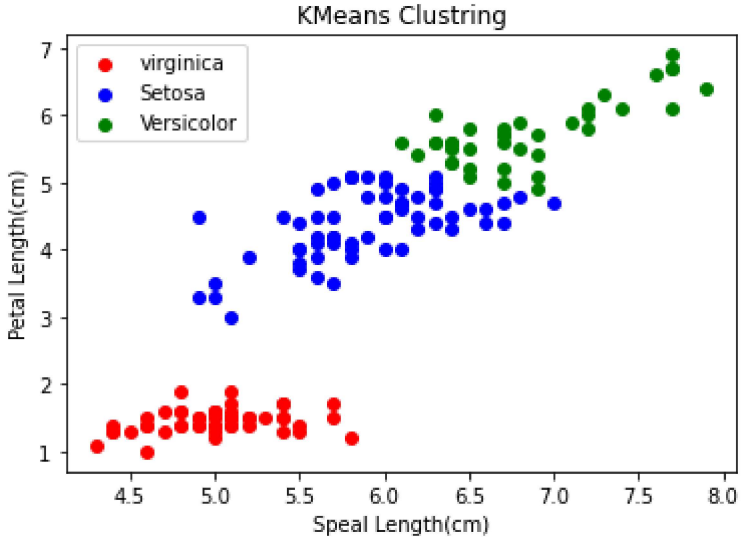
In [15]:
```
y_pred = pd.DataFrame(y_pred1)
```

In [16]:
```
y_pred.columns=['Predict']
iris['clusters'] = y_pred
iris.head()
```

Out[16]:

	sepal_length	sepal_width	petal_length	petal_width	clusters
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [17]:
```
# Visualizing the clusters
plt.scatter(iris.iloc[y_pred1==0,0],iris.iloc[y_pred1==0,2],c='red',label='virginica')
plt.scatter(iris.iloc[y_pred1==1,0],iris.iloc[y_pred1==1,2],c='blue',label='Setosa')
plt.scatter(iris.iloc[y_pred1==2,0],iris.iloc[y_pred1==2,2],c='green',label='Versicolor')
plt.xlabel("Speal Length(cm)")
plt.ylabel("Petal Length(cm)")
plt.title("KMeans Clustring")
plt.legend()
plt.show()
```



In [18]:
```
iris.head()
```

Out[18]:

	sepal_length	sepal_width	petal_length	petal_width	clusters
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [19]:
```
x = iris[['sepal_length','sepal_width','petal_width','petal_length']]
y = iris['clusters']
```

In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=101)
```

In [22]:
```
from sklearn.linear_model import LogisticRegression
```

In [23]:
```
log=LogisticRegression()
```

In [24]:
```
log.fit(x_train,y_train)
```

```
C:\Users\mr\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs fa
iled to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/pre
processing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.or
g/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[24]:
```
LogisticRegression()
```

In [25]:
```
p=log.predict(x_test)
```

Accuracy of the model

In [26]:
```
from sklearn import metrics
metrics.confusion_matrix(p,y_test)
```

Out[26]:
```
array([[10,  0,  0],
       [ 0, 20,  1],
       [ 0,  1,  6]], dtype=int64)
```

In [27]:
```
metrics.accuracy_score(p,y_test)
```

Out[27]:
```
0.9473684210526315
```