Assignment 0.1: K-Means Clustring

Submitted to Sir. Ahmad

Submitted by Obaid Ullah

Reg#: 2021-ag-2315

Degree: MS(SE) 2nd

Import Libraries

In [ ]:

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

Load iris dataset by using seaborn library which we've already imported

In [ ]:

```
df = sns.load_dataset('iris')
df.head()
```

Out[ ]:

	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

I've droped petal length,petal width and species columns and name new dataframe as 'df_clean'.

In [ ]:

```
df_clean = df.drop(['petal_length','petal_width','species'],axis=1)
df_clean
```

Out[ ]:

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...	...	...
145	6.7	3.0
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4
149	5.9	3.0

150 rows × 2 columns

In [ ]:

```
df_clean.head()
```

Out[ ]:

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

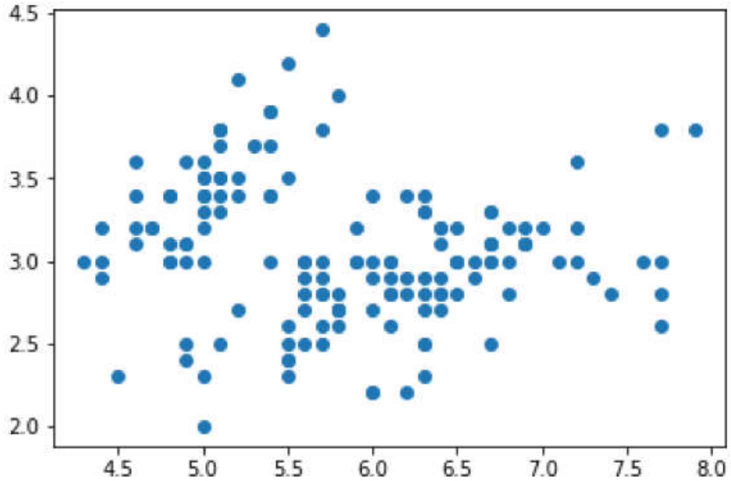
Scatter plot is ploted between sepal length and sepal width, to visualise our dataset.

In [ ]:

```
plt.scatter(df_clean['sepal_length'],df_clean['sepal_width'])
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x26bf2207a00>
```



for k-mean clustring

1. Start with k centroids by putting them at random place.

here k = 3

In [ ]:
```
km = KMeans(n_clusters=3)
km
```

Out[ ]:
```
▾ KMeans
KMeans(n_clusters=3)
```

Compute distance of every point from centroid and cluster them accordingly.

Here i've used fit_predict to predict our clusters which is 0,1,2 (k=3)

In [ ]:
```
y_predict = km.fit_predict(df_clean[['sepal_length','sepal_width']])
y_predict
```

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

also append cluster into df_clean

In [ ]:
```
df_clean['cluster']= y_predict
df_clean.head()
```

Out[ ]:

	sepal_length	sepal_width	cluster
0	5.1	3.5	1
1	4.9	3.0	1
2	4.7	3.2	1
3	4.6	3.1	1
4	5.0	3.6	1

Used cluster*centers* to findout our centriods

In [ ]:
```
km.cluster_centers_
```

Out[ ]:
```
array([[6.81276596, 3.07446809],
       [5.006     , 3.428     ],
       [5.77358491, 2.69245283]])
```

Visualization

1. created new dataframes as per clusters

2. Scatter plot of each cluster

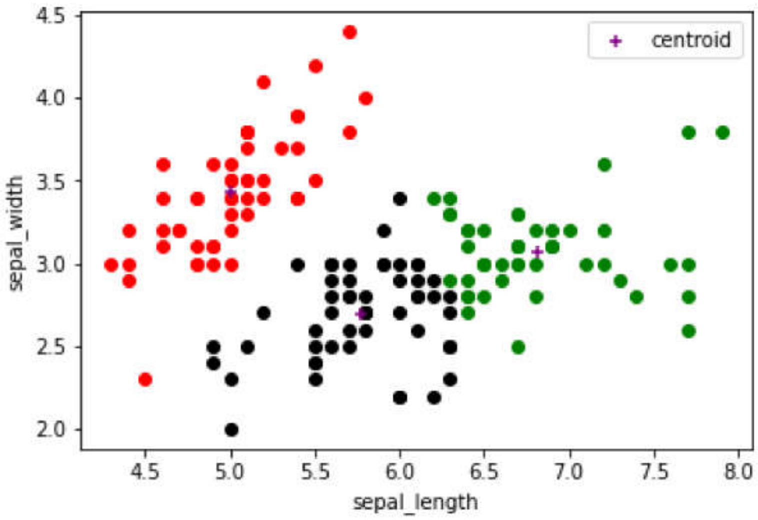
3. Set centroid label and markers

4. X & Y labels Description is given using plt.xlabel and plt.ylabel

In [ ]:
```
# New dataframes based on cluster
df1= df_clean[df_clean.cluster==0]
df2= df_clean[df_clean.cluster==1]
df3= df_clean[df_clean.cluster==2]
# Scatter plot of each cluster
plt.scatter(df1.sepal_length,df1['sepal_width'],color ='green')
plt.scatter(df2.sepal_length,df2['sepal_width'],color ='red')
plt.scatter(df3.sepal_length,df3['sepal_width'],color ='black')
# Centroid formatting
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],
color = 'purple',marker='+',label='centroid')
# X & Y label descriptions
plt.xlabel('sepal_length')
plt.ylabel('sepal_width')

plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x26bf2545600>
```



Use Elbow Technique to find out cluster number or simply k value.

So to apply Elbow technique

First, i've suggested range of k which is between 1-10, SSE(Sum of Square error) is defined as an array. After that for loop will run upto k_rng, in KMeans clusters will be equal to k and SSE will append inertia ok km into SSE array.

In [ ]:
```
#Elbow method to find out K value
k_rng = range(1,10)
sse = []
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df_clean[['sepal_length','sepal_width']])
    sse.append(km.inertia_)
```

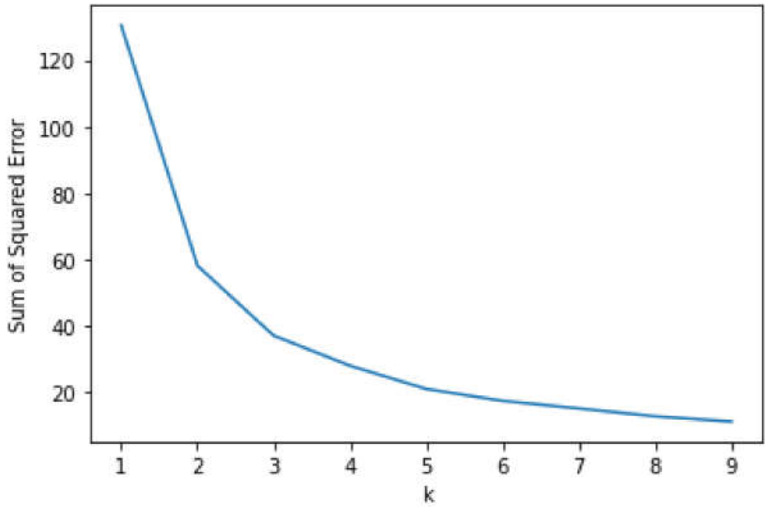
In [ ]:
```
sse
```

Out[ ]:
```
[130.4752666666667,
 58.20409278906672,
 37.05070212765958,
 27.990212038303696,
 21.002125982249435,
 17.41600702075702,
 15.121953809191663,
 12.766940447483554,
 11.185702824952827]
```

Now, Plot graph between SSE and k_rng It has a bend like Elbow, which appears at 2 & 3 but most appropriate place is 3 so that is why "k = 3" is most appropriate number for clustering. Although k=2 will also work but 3 will give us bit more clear clusters.

In [ ]:
```
plt.xlabel('k')
plt.ylabel('Sum of Squared Error')
plt.plot(k_rng,sse)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x26bf23d5cc0>]
```



Mean

Mean is the mathematical average of a set of two or more numbers

Formula

```
m = sum of the terms/number of terms
```

In [ ]:

```
from statistics import mean


mean([1,2,3,4,5,6])
```

Out[ ]:

```
3.5
```

Median

The middle number; found by ordering all data points and picking out the one in the middle (or if there are two middle numbers, taking the mean of those two numbers)

In [ ]:

```
from statistics import median


median([1,2,3,4,5])
```

Out[ ]:

```
3
```

In [ ]:

```
# if we have 2number as a median it will calculate thier mean
# and then take it as a median.
median([1,2,3,4,5,6])
```

Out[ ]:

```
3.5
```

Mode

The most frequent number—that is, the number that occurs the highest number of times. Example: The mode of {4 , 2, 4, 3, 2, 2} is 2 because it occurs three times, which is more than any other number.

In [ ]:
```
from statistics import mode


a = mode([1,0,3,8,6,5,4,3,2,3,4,5,6,7,8])
b = mode([1,2,2,3,4,5,6])
c = mode([1,2,3,4,4,5,6,0,1,1])
print (a)
print (b)
print (c)
```

```
3
2
1
```

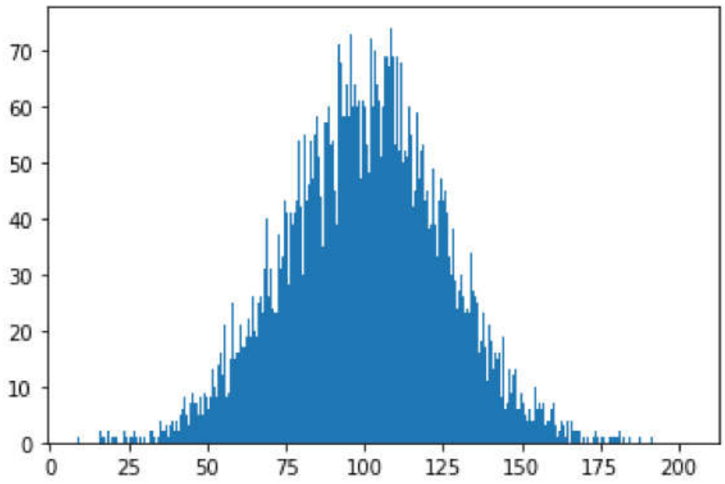
What Is Standard Deviation?

Standard deviation is a statistic that measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance.

What Is Variance?

The term variance refers to a statistical measurement of the spread between numbers in a data set. More specifically, variance measures how far each number in the set is from the mean (average), and thus from every other number in the set. Variance is often depicted by this symbol: σ2. It is used by both analysts and traders to determine volatility and market security.

In [ ]:
```
a = np.random.normal(100,25,10000)
plt.hist(a,500)
plt.show()
```



In [ ]:
```
a.mean()
```

Out[ ]:
```
100.1800928869029
```

In [ ]:
```
# Nothing but just the square of standard deviation.
a.var()
```

Out[ ]:
```
624.8884405300263
```

In [ ]:
```
a.std()
```

Out[ ]:
```
24.997768711027515
```

More than 3 from standard deviation is considered to be outliers(on both sides).

In [ ]:
```
R_outlier= 100+(3*24.99)
L_outlier= 100-(3*24.99)
print('Outliers from Right side of mean:',R_outlier)
print('Outliers from Left side of mean:',L_outlier)
```

```
Outliers from Right side of mean: 174.97
Outliers from Left side of mean: 25.03
```

CNN & RNN

Difference Between CNN and RNN

Convolutional Neural Networks	Recurrent Neural Networks
In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.	A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence.
It is suitable for spatial data like images.	RNN is used for temporal data, also called sequential data.
CNN is a type of feed-forward artificial neural network with variations of multilayer perceptron's designed to use minimal amounts of preprocessing.	RNN, unlike feed-forward neural networks- can use their internal memory to process arbitrary sequences of inputs.
CNN is considered to be more powerful than RNN.	RNN includes less feature compatibility when compared to CNN.
This CNN takes inputs of fixed sizes and generates fixed size outputs.	RNN can handle arbitrary input/output lengths.
CNN's are ideal for images and video processing.	RNNs are ideal for text and speech analysis.
Applications include Image Recognition, Image Classification, Medical Image Analysis, Face Detection and Computer Vision.	Applications include Text Translation, Natural Language Processing, Language Translation, Sentiment Analysis and Speech Analysis.

In [ ]:

In [ ]:

In [ ]: