K-Means is widely used by Data Scientists to solve several problems in different fields:

1. Anomaly Detection: It is used to identify the outliers or anomalies in the dataset like fraud detection etc.

2. Customer Segmentation: It is used to group customers into different clusters on the basis of their income, preferences, etc., which helps companies to decide their marketing strategy accordingly.

3. Image Segmentation: K-means can be used to segment an image into regions based on color or texture similarity

4. KMeans are also widely used for cluster analysis.

different methods to find the optimal value of K in the K-Means algorithm?

A. Some methods used to find the optimal value of K are:

1. Elbow Method: In this method, we plot the WCSS (Within-Cluster Sum of Square)against different values of the K, and we select the value of K at the elbow point in the graph, i.e., after which the value of WSCC remains constant (parallel to the x-axis).

2. Silhouette method: In this method, we calculate the silhouette coefficient of each data point. The silhouette coefficient measures how well the data point fits in the assigned cluster as compared to the other cluster. The average Silhouette coefficient for different K is calculated to find the optimal value of K with the highest coefficient value.

3. Gap statistic method: In this method, we compare the WCSS for different values of K with the expected sum of squares values randomly generated from a uniform distribution. The optimal value of K is the one with the largest gap between the observed and expected sum of squares.

Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster. A good model is one with low inertia AND a low number of clusters (K).

K-means Clustering is an iterative clustering method that segments data into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centroid).

Steps for Plotting K-Means Clusters

This article demonstrates how to visualize the clusters. We'll use the digits dataset for our cause.

1. Preparing Data for Plotting

First Let's get our data ready.

Double-click (or enter) to edit

Double-click (or enter) to edit

```
#Importing required modules

from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import numpy as np

#Load Data
data = load_digits().data
pca = PCA(2)

#Transform the data
df = pca.fit_transform(data)

df.shape
```

```
(1797, 2)
```

Digits dataset contains images of size 8×8 pixels, which is flattened to create a feature vector of length 64.

We used PCA to reduce the number of dimensions so that we can visualize the results using a 2D Scatter plot.

2. Apply K-Means to the Data

ow, let's apply K-mean to our data to create clusters.

Here in the digits dataset we already know that the labels range from 0 to 9, so we have 10 classes (or clusters).

But in real-life challenges when performing K-means the most challenging task is to determine the number of clusters.

There are various methods to determine the optimum number of clusters, i.e. Elbow method, Average Silhouette method. But determining the number of clusters will be the subject of another talk.

```
#Import required module
from sklearn.cluster import KMeans

#Initialize the class object
kmeans = KMeans(n_clusters= 10)

#predict the labels of clusters.
label = kmeans.fit_predict(df)

print(label)
```

```
[2 9 1 ... 1 5 8]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fr
  warnings.warn(
```

kmeans.fit_predict method returns the array of cluster labels each data point belongs to.

3. Plotting Label 0 K-Means Clusters

Now, it’s time to understand and see how can we plot individual clusters.

The array of labels preserves the index or sequence of the data points, so we can utilize this characteristic to filter data points using Boolean indexing with numpy.

```
import matplotlib.pyplot as plt

#filter rows of original data
filtered_label0 = df[label == 0]

#plotting the results
plt.scatter(filtered_label0[:,0] , filtered_label0[:,1])
plt.show()
```

The code above first filters and keeps the data points that belong to cluster label 0 and then creates a scatter plot.

See how we passed a Boolean series to filter [label == 0]. Indexed the filtered data and passed to plt.scatter as (x,y) to plot. x = filtered_label0[:, 0] , y = filtered_label0[:, 1].

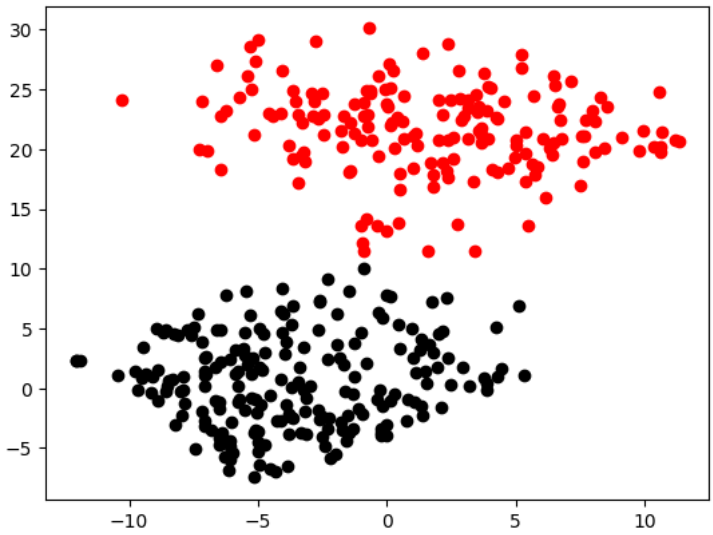
4. Plotting Additional K-Means Clusters

Now, that we have some idea, let's plot clusters with label 2 and 8.

```
#filter rows of original data
filtered_label2 = df[label == 2]

filtered_label8 = df[label == 8]

#Plotting the results
plt.scatter(filtered_label2[:,0] , filtered_label2[:,1] , color = 'r
plt.scatter(filtered_label8[:,0] , filtered_label8[:,1] , color = 'b
plt.show()
```



5. Plot All K-Means Clusters

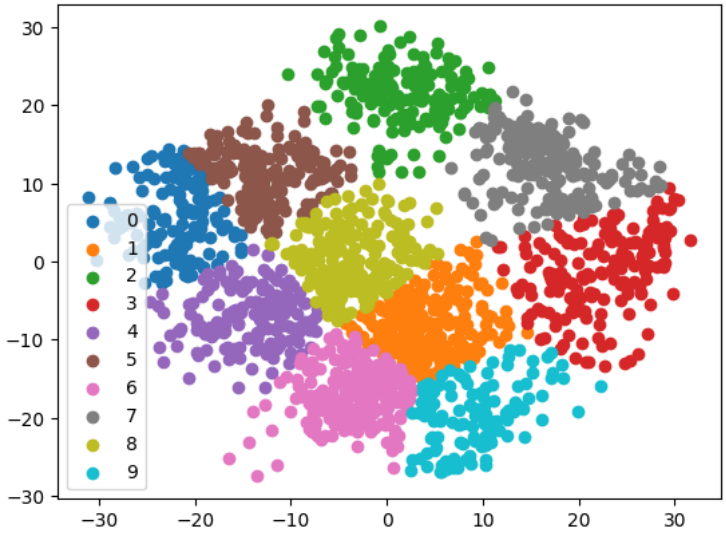
Now, that we got the working mechanism let's apply it to all the clusters.

```
#Getting unique labels

u_labels = np.unique(label)

#plotting the results:
```

```
for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.legend()
plt.show()
```



The above code iterates filtering the data according to each unique class one iteration at a time. The result we get is the final visualization of all the clusters.

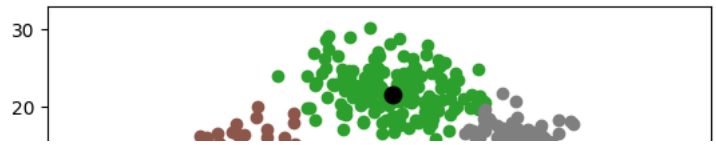
6. Plotting the Cluster Centroids

```
#Getting the Centroids
centroids = kmeans.cluster_centers_
u_labels = np.unique(label)
print(u_labels )
#plotting the results:

for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)

plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```

```
[0 1 2 3 4 5 6 7 8 9]
```



Loading the Data

We begin by loading the iris data set from the sklearn package. We simply load the data and store the resulting data frame in a variable.

```
from sklearn.datasets import load_iris
```

```
df = load_iris(as_frame=True).data
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Fitting the Kmeans Model

Our next step is to fit a kmeans model. We won't go into too much detail here about tuning the model.

For now, we will just select 3 clusters and then predict using fit_predict on our dataset. This results in an array containing predicted clusters for each row in our data frame.

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3)

label = kmeans.fit_predict(df)

print(label)
```

```
[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 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2
 2 0]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fr
  warnings.warn(
```

Plotting the KMeans Clusters

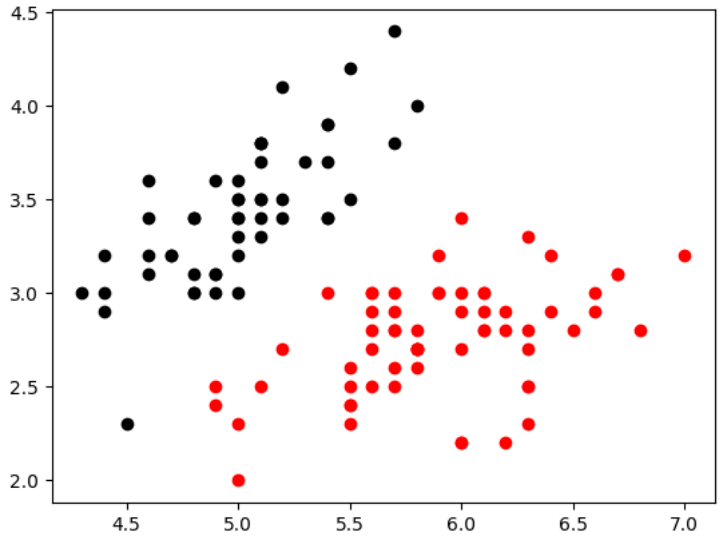
To plot the data, we can first filter our data set by the labels. This will give us three data sets with the rows filtered into their predicted clusters

```
label_0 = df[label == 0]
label_1 = df[label == 1]
label_2 = df[label == 2]
```

Now there are many ways to plot the data. The idea here is to plot our data sets and compare their respective features. Then, color the obeservations by their cluster.

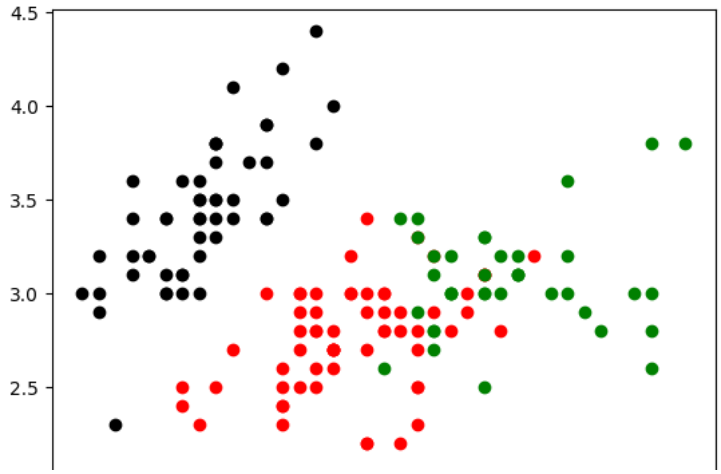
Let's start by plotting two of our clusters. We will compare sepal length (cm) which is the first column to sepal width (cm) the second column.

```
import matplotlib.pyplot as plt
cols = df.columns
plt.scatter(label_0[cols[0]], label_0[cols[1]], color = 'red')
plt.scatter(label_1[cols[0]], label_1[cols[1]], color = 'black')
plt.show()
```



In the graph above it is easy to see the speration of the first and second cluster when viewing length and width. Let's add our third cluster now.

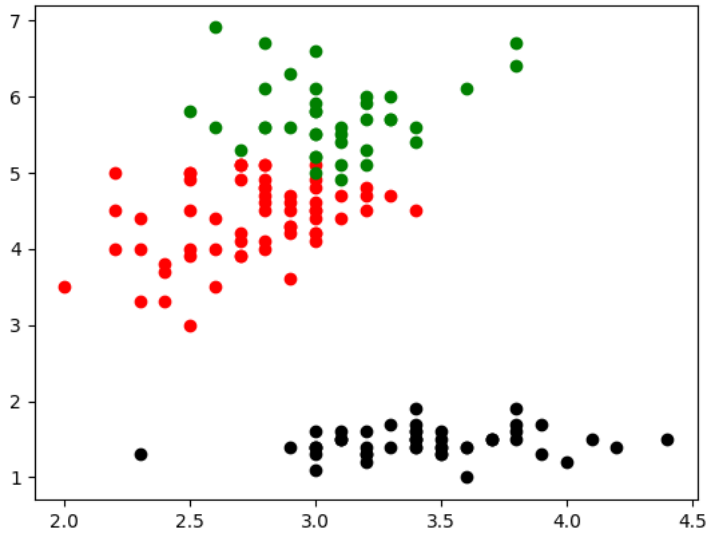
```
plt.scatter(label_0[cols[0]] , label_0[cols[1]], color = 'red')
plt.scatter(label_1[cols[0]] , label_1[cols[1]], color = 'black')
plt.scatter(label_2[cols[0]] , label_2[cols[1]], color = 'green')
plt.show()
```



Here we can see some slight overlap between the second and third cluster suggesting they could be combined, but there is still a nice separation.

Let's try to plot two different columns now. We will use the second and third column which represent sepal width (cm) and petal length (cm) respectively.

```
plt.scatter(label_0[cols[1]] , label_0[cols[2]], color = 'red')
plt.scatter(label_1[cols[1]] , label_1[cols[2]], color = 'black')
plt.scatter(label_2[cols[1]] , label_2[cols[2]], color = 'green')
plt.show()
```



Now, we could continue this way for each pair of features. For small data sets like this that is not a huge problem. For larger data sets, you may want to use feature reduction techniques such as PCA to only plot combinations of features that are important.

Problem

The dataset we are using here is the Mall Customers data (Download here).

https://github.com/makhan010385/DataSet

Dataset name: Mall_Customers.csv

It’s unlabeled data that contains the details of customers in a mall (features like genre, age, annual income(k$), and spending score).

Q.1 Find the Optimal Number of Clusters in K-Means Using Elbow methods

Q.2 Find the cluster of customers based on the relevant features of annual income and spending score.

Q.3 visualize the clusters using the scatter plot.

Q.4 find the Optimal Number of Clusters in K-Means Using Silhouette method