K-means Clustering for Us arrest dataset

Algorithm

Step1. Read the csv file , USArrests.csv.

```
dataset = pd.read_csv('USArrests.csv')
```

Step2. Select 4 columns of the dataset.

```
 X = dataset.iloc[:, [1, 2, 3, 4]].values
```

Step3. Assign the number of clusters, tolerence and maximum number of iterations.

```
 #Initialization through constructor
def __init__(self, k =3, tolerance = 0.0001, max_iterations = 500):
        self.k = k
        self.tolerance = tolerance
        self.max_iterations = max_iterations
```

Step4. initialize the centroids

```
for i in range(self.k):
                self.centroids[i] = data[i]
```

Step5. D*istance* between the data points and the centroid is calculated(euclidian distance) and minimum is chosen and that datapoint is appended to the self.classes list.

```
for features in data:
                        distances = [np.linalg.norm(features - self.centroids[centroid]) for
                                centroid in self.centroids]
                        classification = distances.index(min(distances))
                        self.classes[classification].append(features)
```

Step6. Store the previous value of centroid in previous dictionary.

```
previous = dict(self.centroids)
```

Step7. Average the cluster data points to re-calculate the centroids.

```
 for classification in self.classes:
                        self.centroids[classification] = np.average(self.classes[classification],
                                                            axis = 0)
```

Step8. A flag isOptimal is set to check when the variation becomes constant, check whether the percentage difference between the previous and current centroid points is less than the tolerance value.(i.e.)

Step9. Now calculate the distance between each data point and the final value of the cluster.

```
def pred(self, data):
        distances = [np.linalg.norm(data - self.centroids[centroid]) for centroid in
                self.centroids]
        classification = distances.index(min(distances))
        return classification
```

Step9. Plot graph to visualize the clusters.

```
Plotting the centroids.
for centroid in km.centroids:
        plt.scatter(km.centroids[centroid][0], km.centroids[centroid][1], s = 130, marker = "x")

plotting the datapoints:
 for classification in km.classes:
        color = colors[classification]
        for features in km.classes[classification]:
                plt.scatter(features[0], features[1], color = color,s = 30)
```