An implementation of the k-Means algorithm using GreedyKCenters Algorithm

In [2]:

```
#Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random as rd

from sklearn.metrics.pairwise import euclidean_distances
```

In [3]:

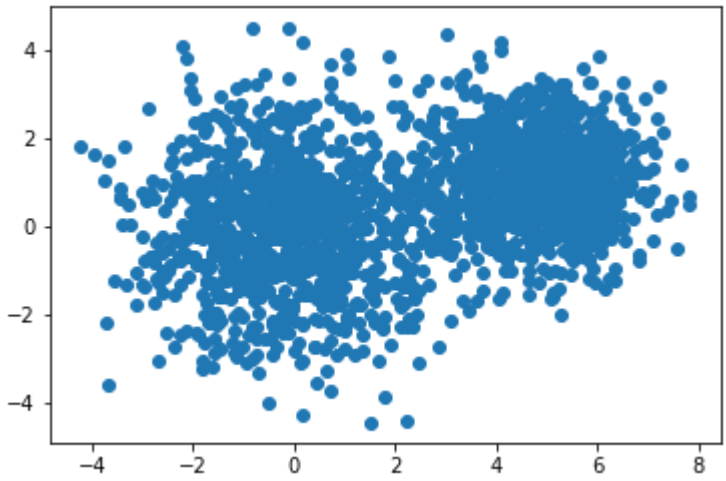
```
data1 = pd.read_csv("C:/Users/Karthik/Desktop/IE_529/clustering.csv", header = None)
data2 = pd.read_csv("C:/Users/Karthik/Desktop/IE_529/ShapedData.csv", header = None)
```

In [4]:

```
def greedy_kcenters( data1, k ):


    copy = data1.copy()
    Init_center = copy.sample( 1 )
    copy.drop(Init_center.index,inplace = True)
    copy.index = list(range( copy.shape[0] ) )

    while Init_center.shape[0] < k:

        #Get index of data point that has maximum minimum distance from any center
        ind = np.argmax(np.amin( euclidean_distances(copy, Init_center), axis=1 ))

        #Append data in temp_df at index ind into C
        Init_center = Init_center.append( copy.loc[ind] )

        #Remove that row from temp_df
        copy.drop(ind,inplace = True)
        #and change indices to 0,1,2,...,n-1
        copy.index = list( range( copy.shape[0] ) )
    del copy
    return( Init_center, np.amax(np.amin(euclidean_distances( data1, Init_center ), axis=1
```

In [5]:

```
plt.scatter(data1[0],data1[1])
plt.show()
```



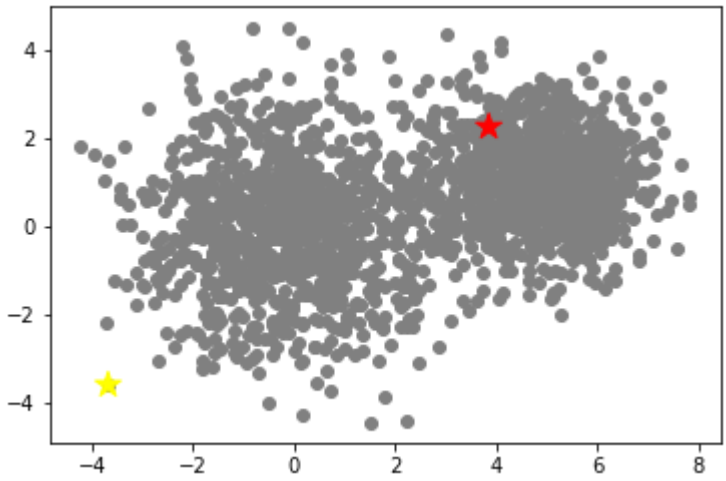
In [6]:

```
centers, dist_cost = greedy_kcenters( data1, 2 )
print("Centers:", pd.DataFrame(centers))
print("Cost:", dist_cost)
```

```
Centers:            0       1
1388  3.8464  2.2586
642  -3.6773 -3.5965
Cost: 6.33764293251
```

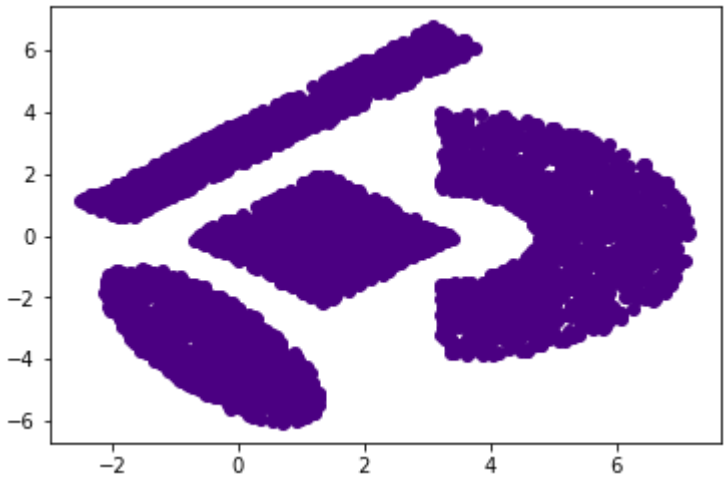
In [7]:

```
plt.scatter(data1[0], data1[1], color = 'grey')
plt.scatter(centers[0],centers[1], marker = "*", color = ['red', 'yellow'], s =180)
plt.show()
```



In [8]:

```
plt.scatter(data2[0],data2[1], color = 'indigo')
plt.show()
```



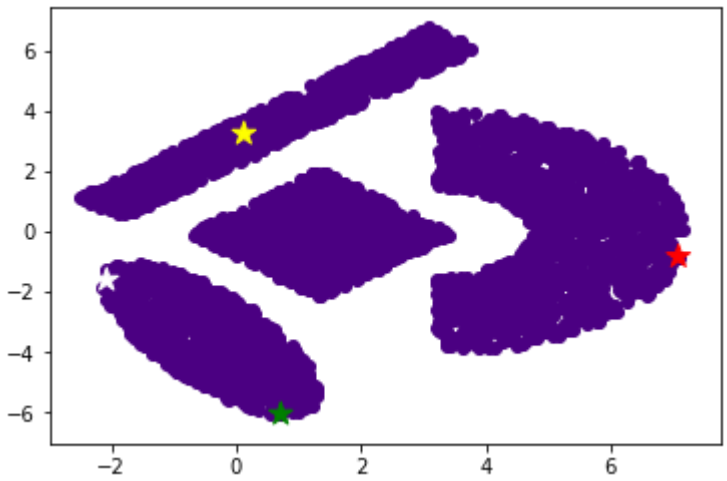
In [9]:

```
centers2, dist_cost2 = greedy_kcenters( data2, 4 )
print("Centers:", pd.DataFrame(centers2))
print("Cost:", dist_cost2)
```

```
Centers:             0        1
122   0.12568  3.25110
642   0.71578 -6.05150
1974  7.09040 -0.82791
2892 -2.07690 -1.60160
Cost: 4.79423423709
```

In [11]:

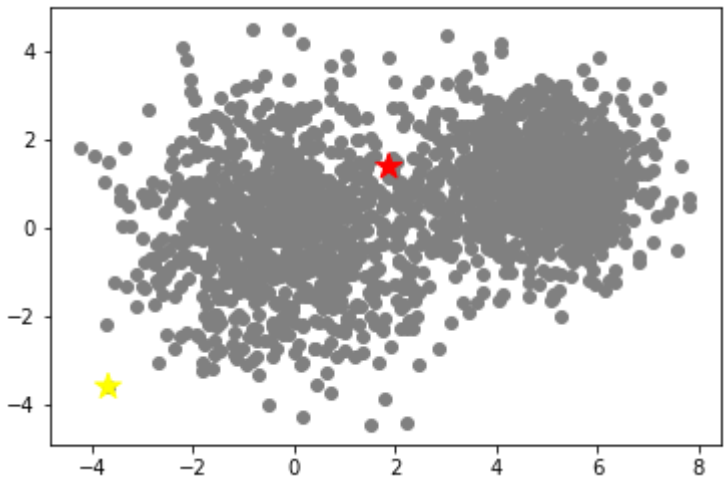
```
norm = plt.Normalize()
plt.scatter(data2[0], data2[1], color = 'indigo')
plt.scatter(centers2[0],centers2[1], marker = "*", color = ['yellow', 'green','red','white'
plt.show()
```



In [13]:

```
k = [2,4,5,7,8,9]
for i in k:
    centers, dist_cost = greedy_kcenters( data1, i )
    print("Centers:", pd.DataFrame(centers))
    print("Cost:", dist_cost)
    plt.scatter(data1[0], data1[1], color = 'grey')
    plt.scatter(centers[0],centers[1], marker = "*", color = ['red', 'yellow','green','blac
    plt.show()
```

```
Centers:           0       1
13   1.8697  1.3846
641 -3.6773 -3.5965
Cost: 6.00669092514
```



Heuristic of SINGLE SWAP

In [17]:

```
def sswap_centers( data, k ):

    centers,dist = greedy_kcenters( data, k )

    C0 = pd.DataFrame(euclidean_distances( data, centers )).min(axis=1).sum()
    print("Gk centers cost= ", C0)

    counter = centers.shape[0]

    beginning = False

    while counter > 0:

        for i in range( centers.shape[0] ):
            if beginning == True:
                beginning = False
                break

            #Create a copy of the dataframe df without the centers Q
            copy = pd.concat( [data,centers] )
            copy.drop_duplicates(keep=False)
            copy.index = list( range( copy.shape[0] ) )

            #Swap
            while copy.shape[0] != 0:
                centers_new = pd.concat( [ centers.drop(centers.index[i]), copy.loc[[0]] ]

                #Get cost of new centers
                Cost_new =  pd.DataFrame(euclidean_distances( data, centers_new )).min(axis

                #Check for reduced cost of more than gamma(which is taken to be 0.05)
                if Cost_new <= 0.95 * C0:
                    print('1')
                    centers = centers_new.copy()
                    print("SS_cost= ", Cost_new)
                    copy = pd.concat( [data,centers] )
                    copy.drop_duplicates(keep=False)
                    copy.index = list( range( copy.shape[0] ) )
                    beginning = True
                    counter = centers.shape[0]
                    break
                else:
                    copy.drop(0, inplace=True)
                    copy.index = list( range( copy.shape[0] ) )
            if beginning == False:
                print(counter)
                counter -= 1

    #Return the centers
    return( centers )
```