Question 1

In [1]:
```
from sklearn import datasets
Xb, yb = datasets.make_blobs(n_samples=40,
                             centers=3,
                             n_features=2,
                             cluster_std=1.0,
                             center_box=(-10.0, 10.0),
                             shuffle=True,
                            random_state=2)
```

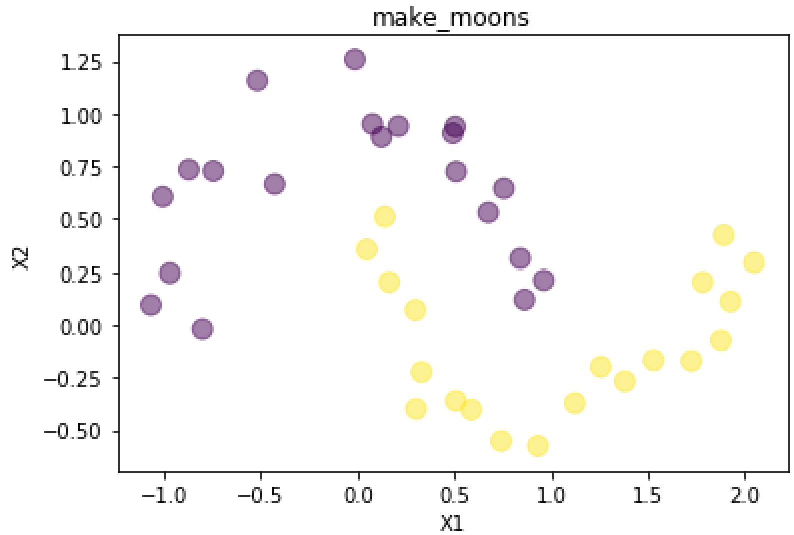
In [2]:
```
Xmn, ymn = datasets.make_moons(n_samples=40,
                               shuffle=True,
                               noise=0.1,
                               random_state=None)
```

In [3]:
```
Xc, yc = datasets.make_circles(n_samples=40,
                              shuffle=True,
                              noise=0.1,
                              random_state=None,
                              factor=0.01)
```

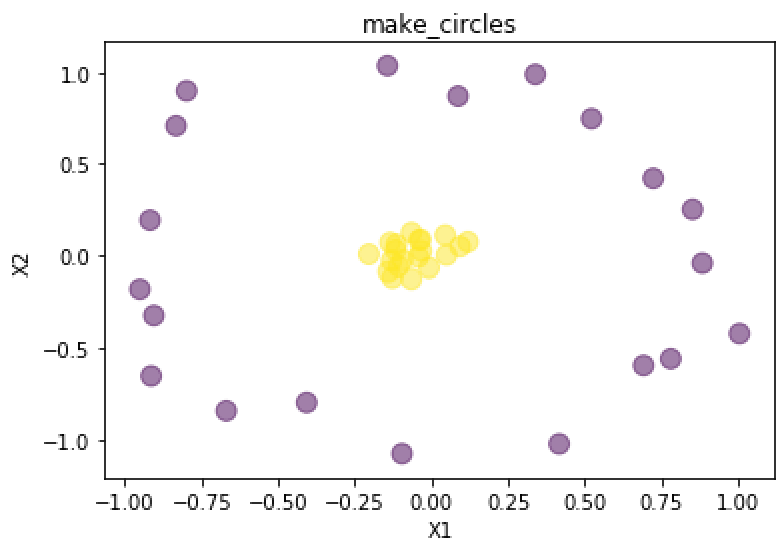
In [4]:
```
Xcl, rw,cl = datasets.make_biclusters((40,2), n_clusters=2, noise=20, minval=10, m
```

In [5]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(Xc[:,0], Xc[:,1], c=yc, s=100, alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Example of generate a data for clustering')
plt.show()
```

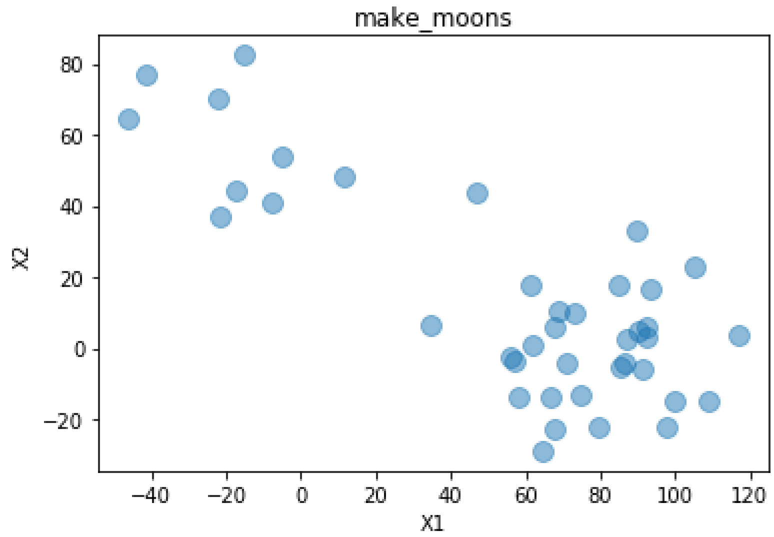
In [6]:
```
plt.scatter(Xmn[:,0], Xmn[:,1], c=ymn, s=100, alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('make_moons')
plt.show()
```



In [7]:
```
plt.scatter(Xc[:,0], Xc[:,1], c=yc, s=100, alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('make_circles')
plt.show()
```



In [8]:
```
plt.scatter(Xcl[:,0], Xcl[:,1],  s=100, alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('make_moons')
plt.show()
```



Question 2

- Compute similarity between two points
- Compute distance between two points

In [9]:
```
import sklearn.metrics.pairwise as pw
import numpy as np
```

In [10]:
```
p1 = np.array([1,2]); p2 = np.array([1.75,3]); p3 = np.array([1,3.5])
```

In [11]:
```
d1 = pw.euclidean_distances(p1.reshape(-1,2),  p2.reshape(-1,2) )[0]
d2 = pw.euclidean_distances(p1.reshape(-1,2),  p3.reshape(-1,2))[0]
if d1< d2:
    print('P2 is closer to P1: {}'.format(d1))
else:
    print('P3 is closer to P1: {}'.format(d2))
```

```
P2 is closer to P1: [1.25]
```

In [12]:
```
d1 = pw.manhattan_distances(p1.reshape(-1,2),  p2.reshape(-1,2) )[0]
d2 = pw.manhattan_distances(p1.reshape(-1,2),  p3.reshape(-1,2))[0]
if d1< d2:
    print('P2 is closer to P1: {}'.format(d1))
else:
    print('P3 is closer to P1: {}'.format(d2))
```

```
P3 is closer to P1: [1.5]
```

In [13]:
```
s1 = pw.cosine_similarity(p1.reshape(-1,2),  p2.reshape(-1,2) )[0,0]
s2 = pw.cosine_similarity(p1.reshape(-1,2),  p3.reshape(-1,2))[0,0]
if s1< s2:
    print('P2 is closer to P1: {}'.format(s1))
else:
    print('P3 is closer to P1: {}'.format(s2))
```

```
P3 is closer to P1: 0.982872186934
```

using equations

In [14]:
```
dot1_2 = np.dot(p1, p2)
dot1_3 = np.dot(p1, p3)

norm1 = np.linalg.norm(p1)
norm2 = np.linalg.norm(p2)
norm3 = np.linalg.norm(p3)

sim1_2 = dot1_2/(norm1 * norm2)
sim1_3 = dot1_3/(norm1 * norm3)

print("Similarity betwen P1 to P2 ={}, and P1 to P3={}".format(sim1_2, sim1_3))
```

```
Similarity betwen P1 to P2 =0.997925308968, and P1 to P3=0.982872186934
```

In [15]:
```
# compute Euclidean distance
euc1_2 = np.sqrt( (p1[0]-p2[0])**2 + (p1[1]-p2[1])**2 )
euc1_3 = np.sqrt( (p1[0]-p3[0])**2 + (p1[1]-p3[1])**2 )
print("distance from P1 to P2 ={}, and from P1 to P3={}".format(euc1_2, euc1_3))
```

```
distance from P1 to P2 =1.25, and from P1 to P3=1.5
```

In [16]:
```
# compute Cityblock distance
euc1_2 =( np.abs(p1[0]-p2[0]) + np.abs(p1[1]-p2[1]) )
euc1_3 =( np.abs(p1[0]-p3[0]) + np.abs(p1[1]-p3[1]) )
print("distance from P1 to P2 ={}, and from P1 to P3={}".format(euc1_2, euc1_3))
```

```
distance from P1 to P2 =1.75, and from P1 to P3=1.5
```

Question 3

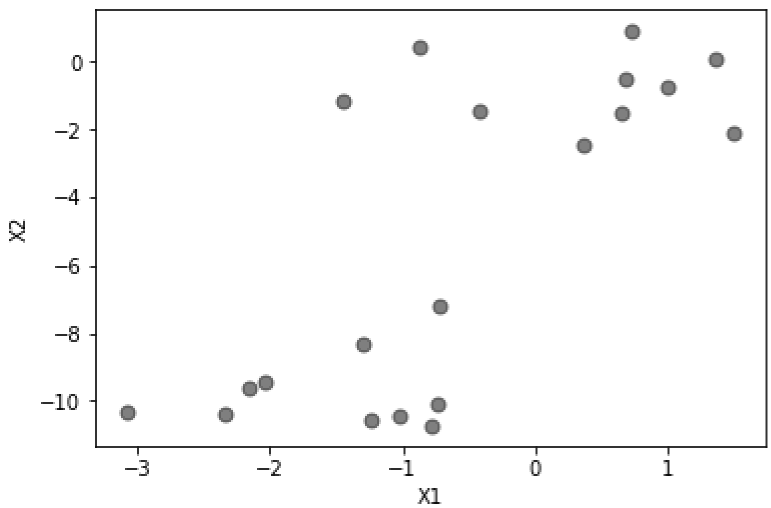
In [17]:
```
from sklearn.metrics import jaccard_similarity_score, adjusted_rand_score
```

In [18]:
```
# Using make_blobs
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=20,
                  centers=2,
                  n_features=2,
                  cluster_std=1,
                  random_state=2)
y
```

Out[18]:
```
array([0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0])
```

In [19]:
```
plt.scatter (X[:,0], X[:,1], c='k', facecolor ='none', s=50,  alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')

plt.show()
```



In [20]:
```
# random boolean mask for which values will be changed
mask = np.random.randint(0,2,size=y.shape).astype(np.bool)

# let us make ones more than zeros (maybe)
change= np.ones(y.shape)
yhat=np.copy(y)
yhat[mask] = change[mask]
print("y:", y)
print("yhat:",yhat)
```

```
('y:', array([0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0]))
('yhat:', array([1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]))
```

In [21]:
```
Agree    = y[y==yhat]
Disagree = np.ones(y[y!=yhat].shape).sum()
Pagree   = Agree[Agree == 1].sum()
Nagree   = len(Agree[Agree == 0])
Rand     = (Pagree + Nagree)/(Pagree + Nagree+Disagree)
Jaccard = Pagree/(Pagree + Disagree)
print('Jaccard:{}'.format(Jaccard))
print('Raw Rand:{}'.format(adjusted_rand_score(y,yhat)))
```

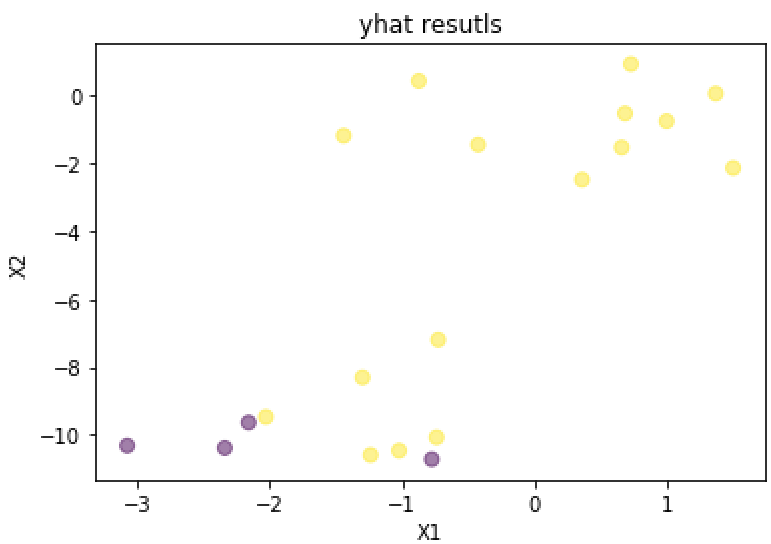
```
Jaccard:0.625
Raw Rand:0.130718954248
```

In [22]:
```
jaccard_similarity_score(y,yhat) # another version
```

Out[22]: 0.7

In [23]:
```
# you can generate zeros ones randomly
y_hat2 = np.random.randint(0,2,size=y.shape)
```

In [24]:
```
plt.scatter (X[:,0], X[:,1], c=yhat, facecolor ='none', s=50,  alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title("yhat resutls")
plt.show()
```



With the above results, Jaccard tells you we got 66%-75% correct. This means if we are in a two cluster situation, about 25% of datapoints in the other cluster are not labeled correctly. While, rand gives you the big picture, you just missed alot.

Question 4

In [25]:
```
# Compute Cohesion
cluster_1_datapoints = X[yhat==0, :]
cluster_1_mean = cluster_1_datapoints.mean(axis=0).reshape(1,2)
X2= np.tile(cluster_1_mean, (cluster_1_datapoints.shape[0],1) )
WS1 = ((cluster_1_datapoints - X2)**2).sum()


cluster_2_datapoints = X[yhat==1, :]
cluster_2_mean = cluster_2_datapoints.mean(axis=0).reshape(1,2)
X3= np.tile(cluster_2_mean, (cluster_2_datapoints.shape[0],1) )
WS2 = ((cluster_2_datapoints - X3)**2).sum()


WSS = WS1 + WS2
WSS
```

Out[25]:
```
312.2953884188511
```

In [26]:
```
# compute Separation
bs1 = len(cluster_1_datapoints) * ((X.mean(axis=0) - cluster_1_datapoints.mean(axi
bs2 = len(cluster_2_datapoints) * ((X.mean(axis=0) - cluster_2_datapoints.mean(axi
BSS = bs1 + bs2
BSS
```

Out[26]:
```
135.11368291120243
```

Conclusion our clustering is not good. We have high Cohesion and low Separation

In [27]:
```
from sklearn.metrics import silhouette_score
```

In [28]:
```
silhouette_score(X, yhat)
```

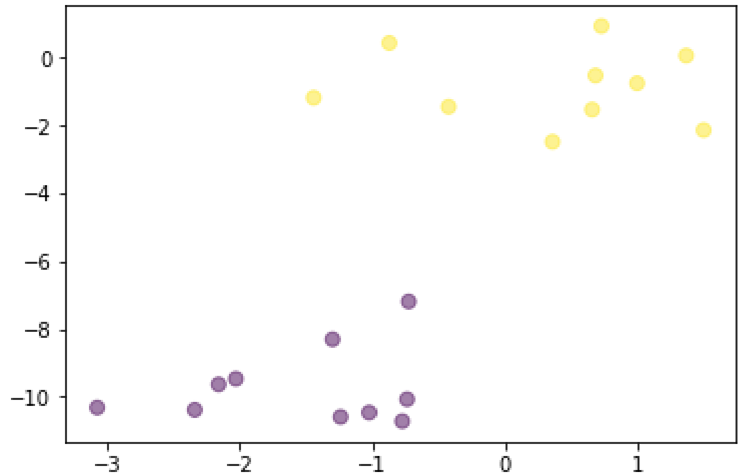
Out[28]:
```
0.21245027669618607
```

Question 5

In [29]:
```
from sklearn.cluster  import KMeans
```

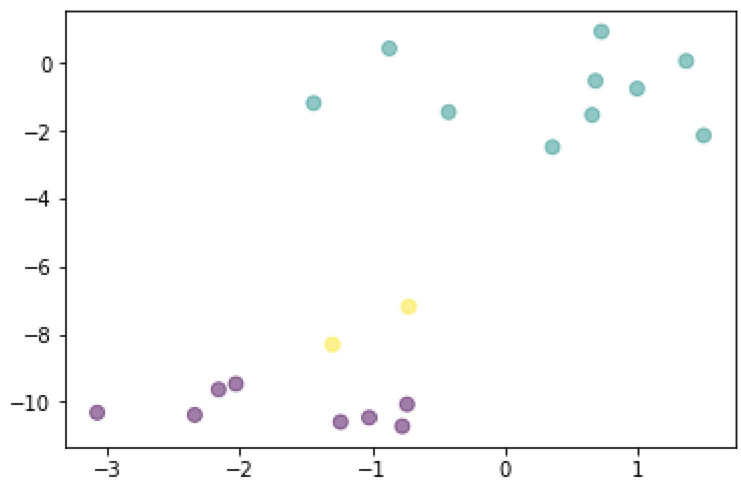
In [30]:
```
k_means = KMeans(n_clusters=2 )
k_means.fit(X)
yhat_new = k_means.predict(X)
```

In [31]:
```
plt.scatter (X[:,0], X[:,1], c=yhat_new, facecolor ='none', s=50,  alpha=0.5)
plt.show()
silhouette_score(X, yhat_new)
```



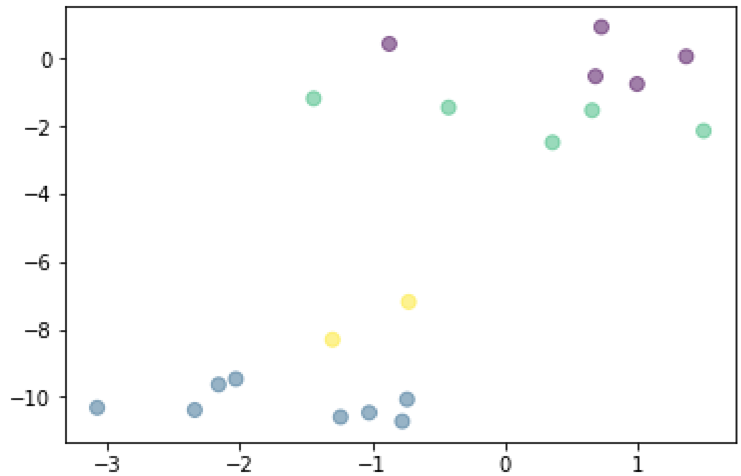
Out[31]: 0.7925105422189154

In [32]:
```
k_means = KMeans(n_clusters=3 )
k_means.fit(X)
yhat_new = k_means.predict(X)
plt.scatter (X[:,0], X[:,1], c=yhat_new, facecolor ='none', s=50,  alpha=0.5)
plt.show()
silhouette_score(X, yhat_new)
```



Out[32]: 0.6258747575320766

In [33]:
```
k_means = KMeans(n_clusters=4 )
k_means.fit(X)
yhat_new = k_means.predict(X)
plt.scatter (X[:,0], X[:,1], c=yhat_new, facecolor ='none', s=50,  alpha=0.5)
plt.show()
silhouette_score(X, yhat_new)
```



Out[33]: 0.4208133986107505

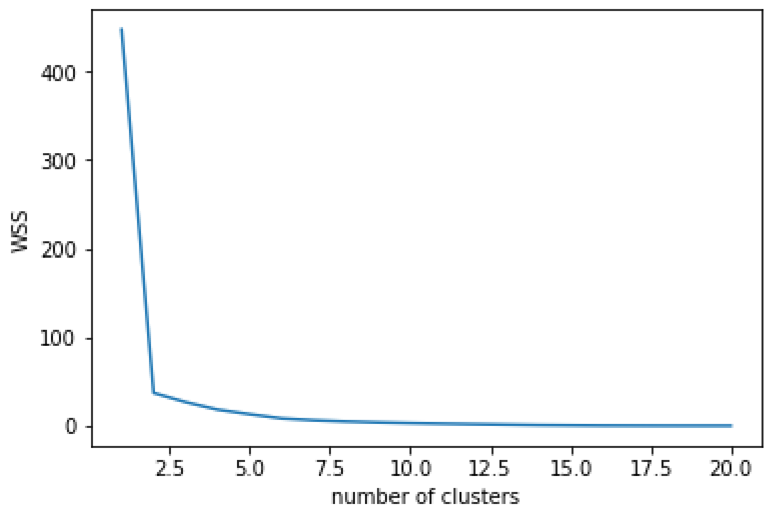
Question 6 (Elbow Method)

In [34]:
```
#help(KMeans)
def Find_K_Kmeans(X):
    # should go from 1 cluster to n number of clusters where WSS will be zero
    WSS = []
    WSS2 = []
    for i in range(1,len(X[:,0])+1):
        # initiat K-means
        kmeans = KMeans(n_clusters=i,  random_state=0)
        kmeans.fit(X)
        ys = kmeans.predict(X)
        d   = computeWSS(X, ys, kmeans.cluster_centers_)
       # print(d, kmeans.inertia_ )
        WSS.append(d)
        WSS2.append(kmeans.inertia_)
    return WSS, WSS2
```

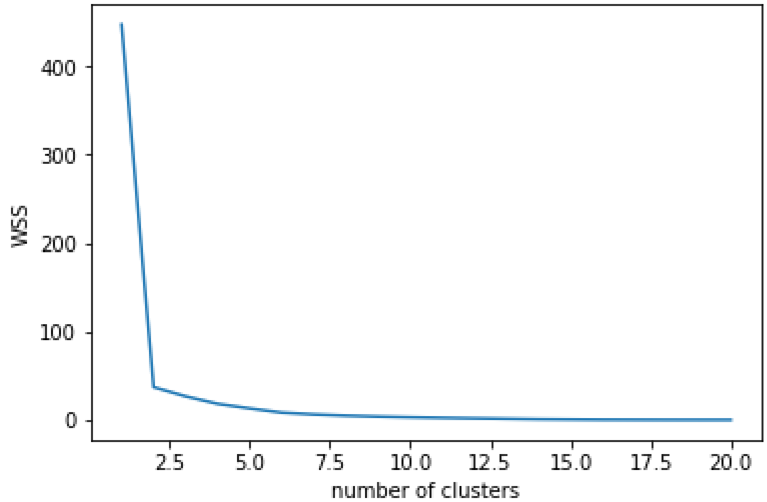
In [35]:
```
def computeWSS(X, Y, centroids):
    d = 0
    uniY = np.unique(Y)
    # compute the within distance of each clusters
    #1- select all points belong to cluster 1
    for i in range(len(uniY)):
        Xi = X[Y==uniY[i],:]
        cen = centroids[i]
        for j in range(len(Xi[:,0])):
            d += (Xi[j,0] - cen[0])**2 + (Xi[j,1] - cen[1])**2
    return d

wss,wss2 = Find_K_Kmeans(X)
```

In [36]:
```
# plot the wss and
num_clusters = range(1,len(wss)+1)
plt.plot(num_clusters, wss)
plt.xlabel('number of clusters')
plt.ylabel('WSS')
plt.show()
```



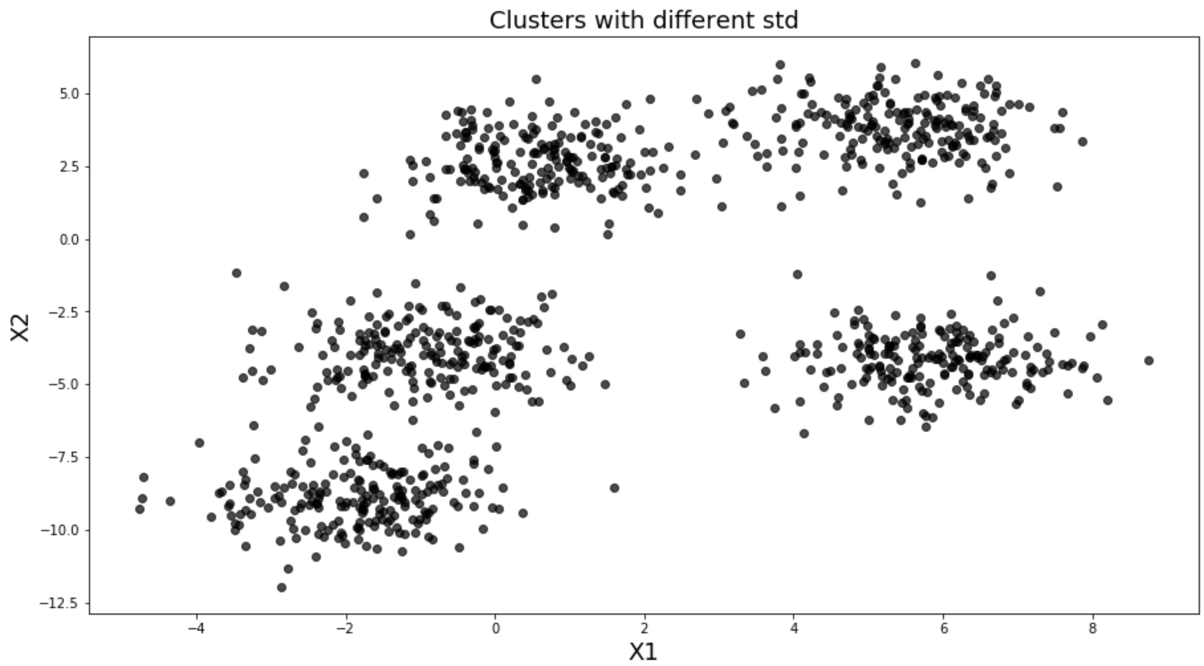
In [37]:
```
# plot the wss and
num_clusters = range(1,len(wss2)+1)
plt.plot(num_clusters, wss2)
plt.xlabel('number of clusters')
plt.ylabel('WSS')
plt.show()
```



Question 7

In [38]:
```
# using Silhouettee to detemine K
# make some data 1- with std = 1, and second with various std
Xeq, yeq   = make_blobs(n_samples = 1000, n_features=2, centers=5, random_state=40
```

In [39]:
```
plt.figure(figsize=[15,8])
plt.scatter (Xeq[:,0],Xeq[:,1], c= 'k', alpha = 0.7 )
plt.xlabel('X1',fontsize=18)
plt.ylabel('X2',fontsize=18)
plt.title('Clusters with different std',fontsize=18)
plt.show()
```

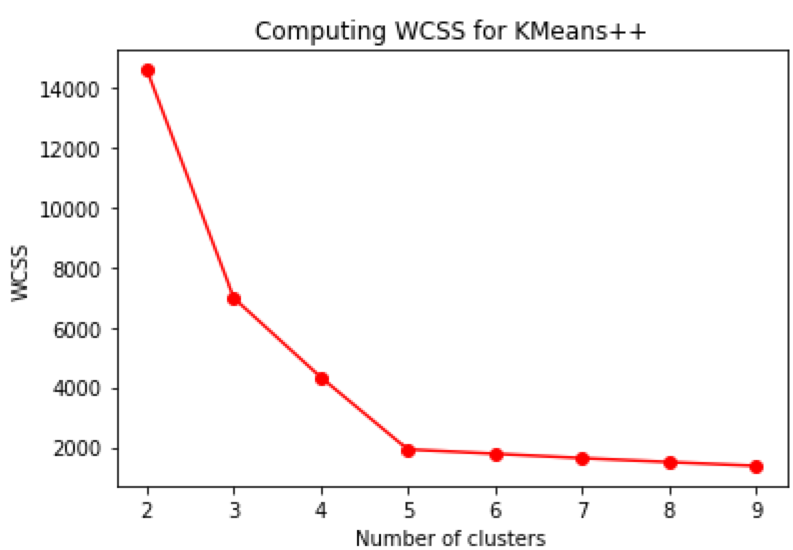


In [40]:
```
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9]
```

In [41]:
```
estimor =[]
SilAvg = []
wcss = []
for N in range_n_clusters:
    clusterer = KMeans(n_clusters=N, random_state=10)
    y_pred = clusterer.fit_predict(Xeq)
    silhouette_avg = silhouette_score(Xeq, y_pred)
    print("n_clusters =", N,"Avg. silhouette score:", silhouette_avg, 'Inetria:',
    estimor.append(clusterer)
    SilAvg.append(silhouette_avg)
    wcss.append(clusterer.inertia_)


plt.plot(range_n_clusters, wcss, 'ro-')
plt.title("Computing WCSS for KMeans++")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.show()
```

```
('n_clusters =', 2, 'Avg. silhouette score:', 0.5244877785257671, 'Inetria:',
14586.207119779028)
('n_clusters =', 3, 'Avg. silhouette score:', 0.6041783018079782, 'Inetria:',
6983.874603628351)
('n_clusters =', 4, 'Avg. silhouette score:', 0.6197823412809729, 'Inetria:',
4338.634725309936)
('n_clusters =', 5, 'Avg. silhouette score:', 0.6639397841888107, 'Inetria:',
1925.6350660467747)
('n_clusters =', 6, 'Avg. silhouette score:', 0.5901503764129244, 'Inetria:',
1779.3930943232149)
('n_clusters =', 7, 'Avg. silhouette score:', 0.5150681826331834, 'Inetria:',
1634.8235413794873)
('n_clusters =', 8, 'Avg. silhouette score:', 0.45737529208086436, 'Inetria:',
1501.6850034729996)
('n_clusters =', 9, 'Avg. silhouette score:', 0.40483439720500825, 'Inetria:',
1376.504772346817)
```



In [ ]:

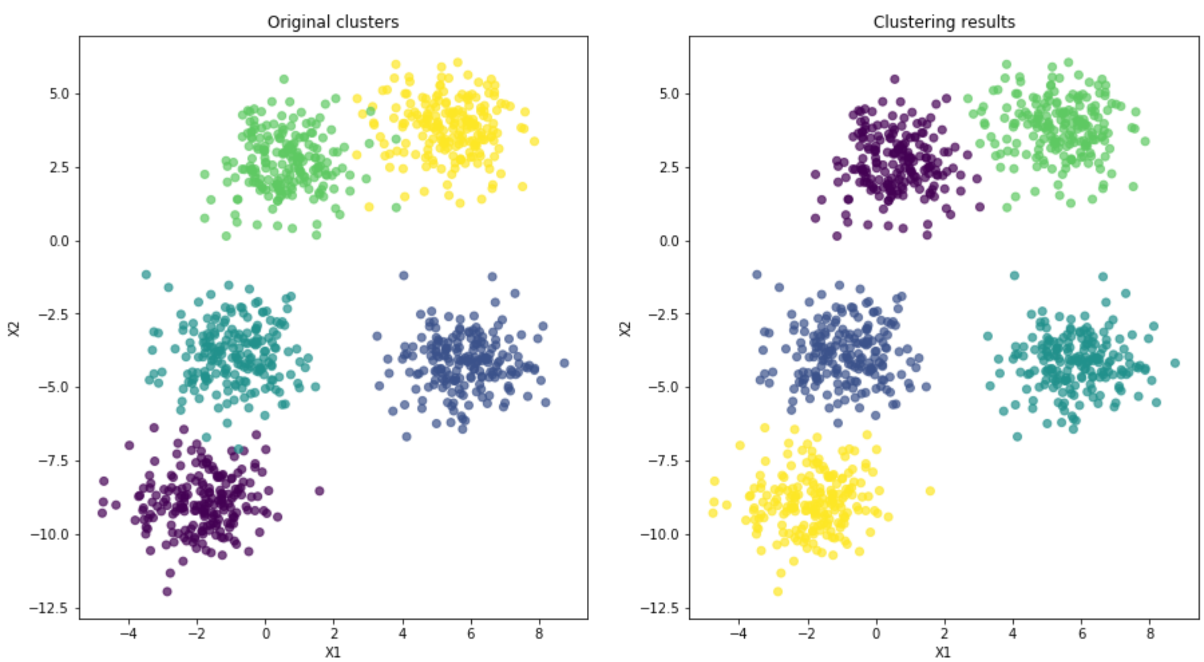
From the WSS plot, we can notice the elbow diagram happening at n=5. Thereofre, we should go with 5 clusters in this data. Another point is the silhouette of this number of clusters is the maximum 0.664. This confirms both methods agrees on the number of clusters in this dataset

In [42]:
```
# wining estimator with max silhouettee average
win_index = SilAvg.index(np.max(SilAvg))
```

In [43]:
```
# Get the wining model
KmeansWin = estimor [win_index]
```

In [44]:
```
# Predict the new labels
y_new = KmeansWin.predict(Xeq)
```

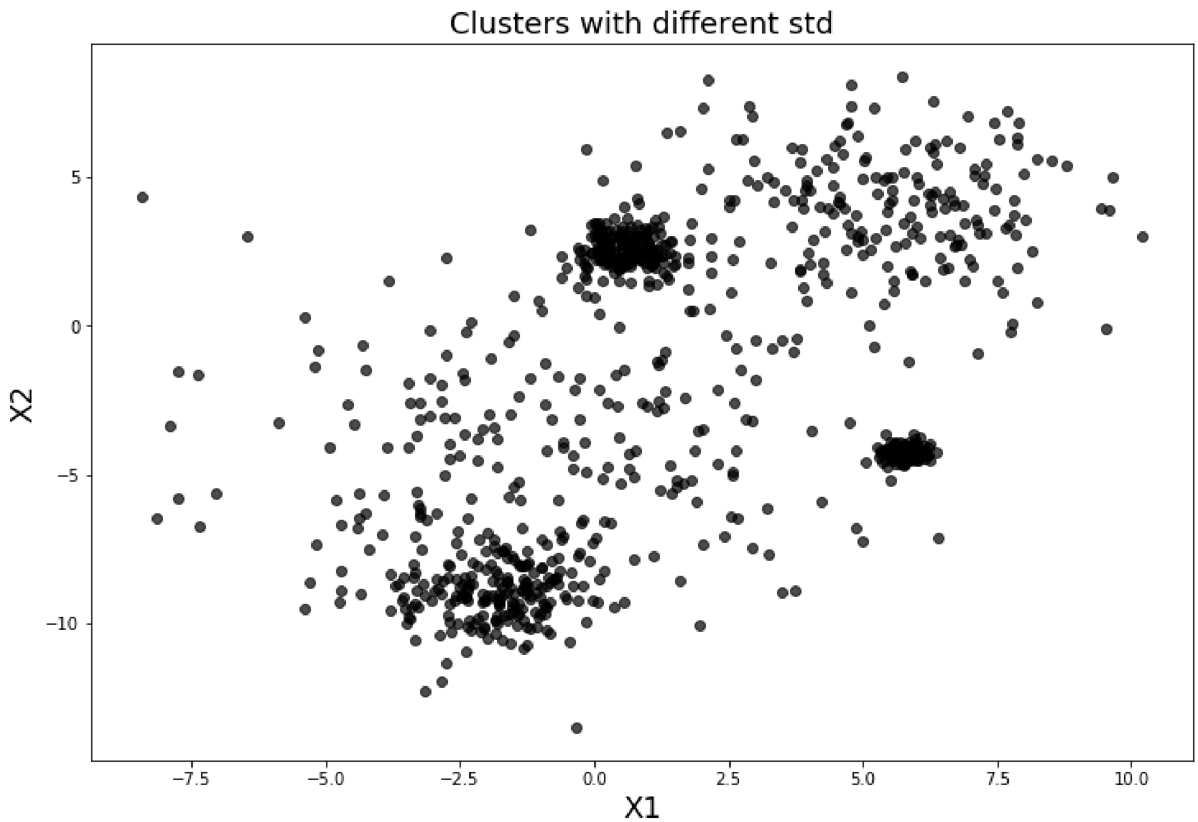
In [45]:
```
#plot results
plt.figure(figsize=[15,8])
plt.subplot(1,2,1)
plt.scatter (Xeq[:,0],Xeq[:,1], c= yeq, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Original clusters')
plt.subplot(1,2,2)
plt.scatter (Xeq[:,0],Xeq[:,1], c= y_new, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Clustering results')
plt.show()
```



Question 8

In [46]:
```
# different spread in each cluster
Xneq, yneq = make_blobs(n_samples = 1000, n_features=2, centers=5, cluster_std=[1,
```

In [47]:
```
plt.figure(figsize=[12,8])
plt.scatter (Xneq[:,0],Xneq[:,1], c= 'k', alpha = 0.7 )
plt.xlabel('X1',fontsize=18)
plt.ylabel('X2',fontsize=18)
plt.title('Clusters with different std',fontsize=18)
plt.show()
```



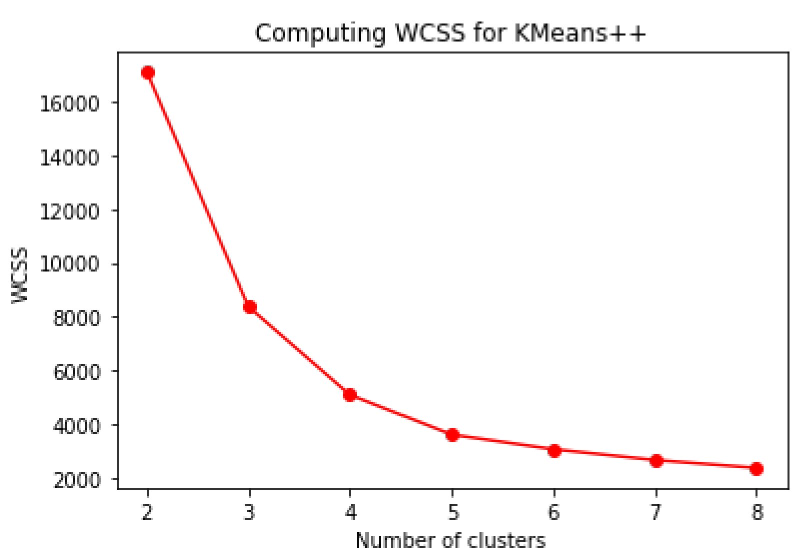
In [ ]:

In [48]:
```
# It is defined as a function because I want to re-use it

def estimate_Ks(X, max_clusters=9, plotFlag=True):
    range_n_clusters = np.arange(2,max_clusters)
    model =[]
    SilAvg = []
    wss = []
    for N in range_n_clusters:
        clusterer = KMeans(n_clusters=N, random_state=10)
        y_pred = clusterer.fit_predict(X)
        silhouette_avg = silhouette_score(X, y_pred)
        print("n_clusters =", N,"Avg. silhouette score:", silhouette_avg, 'Inetria
        model.append(clusterer)
        SilAvg.append(silhouette_avg)
        wss.append(clusterer.inertia_)
    if plotFlag:
        plt.plot(range_n_clusters, wss, 'ro-')
        plt.title("Computing WCSS for KMeans++")
        plt.xlabel("Number of clusters")
        plt.ylabel("WCSS")
        plt.show()
    return model, SilAvg, wss


model, SilAvg, wcss= estimate_Ks(Xneq)
```

```
('n_clusters =', 2, 'Avg. silhouette score:', 0.5121870784585945, 'Inetria:',
17104.388820691624)
('n_clusters =', 3, 'Avg. silhouette score:', 0.592245738152675, 'Inetria:', 8
386.284440842603)
('n_clusters =', 4, 'Avg. silhouette score:', 0.6201785130170808, 'Inetria:',
5098.094764802458)
('n_clusters =', 5, 'Avg. silhouette score:', 0.6225942667287063, 'Inetria:',
3612.4485275117167)
('n_clusters =', 6, 'Avg. silhouette score:', 0.6287068061837086, 'Inetria:',
3068.107017499279)
('n_clusters =', 7, 'Avg. silhouette score:', 0.609375372751648, 'Inetria:', 2
666.055453218961)
('n_clusters =', 8, 'Avg. silhouette score:', 0.549361906572172, 'Inetria:', 2
373.8166678134417)
```



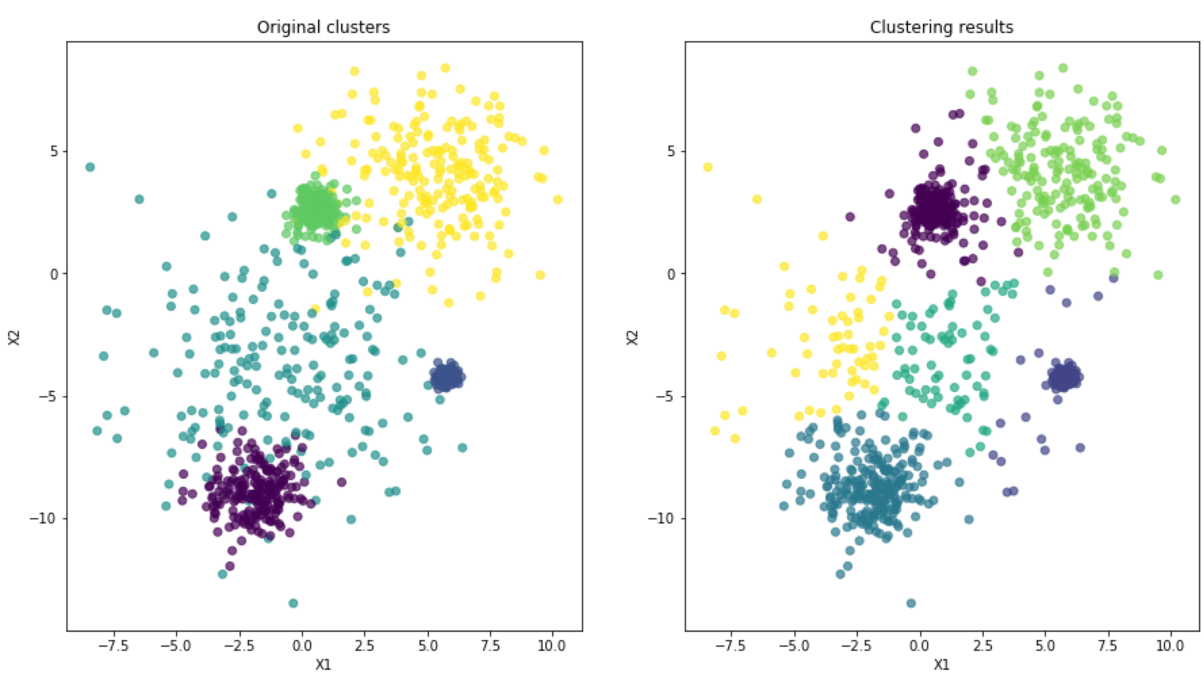
From the WSS plot, we can notice the elbow diagram happening at n=5. Thereofre, we should go with 5 clusters in this data. Another point is the silhouette of this number of clusters is the maximum 0.664. This confirms both methods agrees on the number of clusters in this dataset

In [49]:
```
# wining estimator with max silhouettee average
win_index = SilAvg.index(np.max(SilAvg))
```

In [50]:
```
# Get the wining model
KmeansWin = model [win_index]
```

In [51]:
```
# Predict the new labels
y_new = KmeansWin.predict(Xneq)
```

In [52]:
```
#plot results
plt.figure(figsize=[15,8])
plt.subplot(1,2,1)
plt.scatter (Xneq[:,0],Xneq[:,1], c= yneq, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Original clusters')
plt.subplot(1,2,2)
plt.scatter (Xneq[:,0],Xneq[:,1], c= y_new, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Clustering results')
plt.show()
```



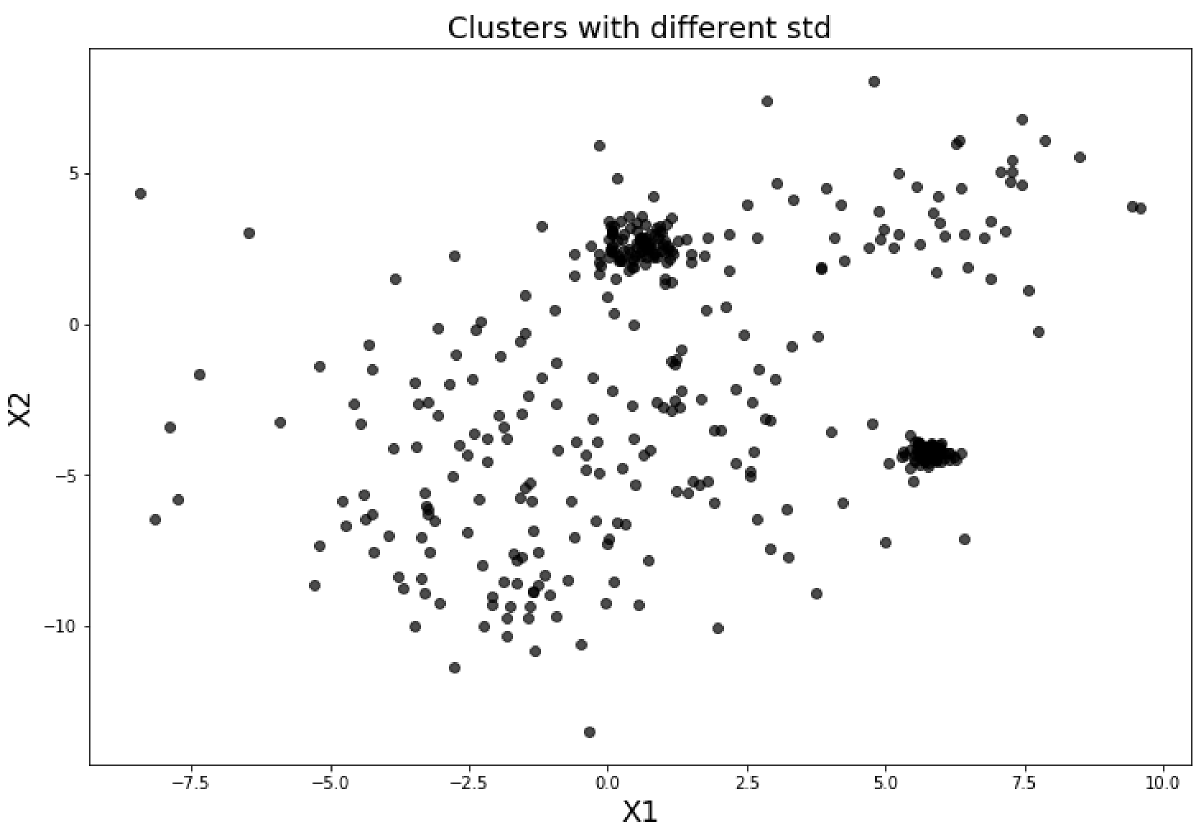
In [53]:
```
# different number of samples per cluster
Xneq_s = np.vstack((Xneq[yneq == 0][:30],
                    Xneq[yneq == 1][:100],
                    Xneq[yneq == 2][:150],
                    Xneq[yneq == 3][:99],
                    Xneq[yneq == 4][:50]))

yneq_s = np.hstack((yneq[yneq == 0][:30],
                    yneq[yneq == 1][:100],
                    yneq[yneq == 2][:150],
                    yneq[yneq == 3][:99],
                    yneq[yneq == 4][:50]))
```

In [54]:
```
Xneq_s.shape
```

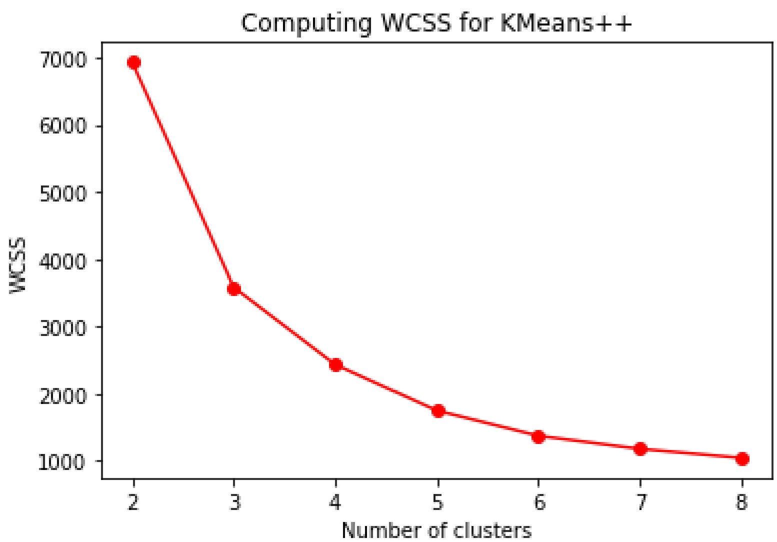
Out[54]: (429L, 2L)

In [55]:
```
plt.figure(figsize=[12,8])
plt.scatter (Xneq_s[:,0],Xneq_s[:,1], c= 'k', alpha = 0.7 )
plt.xlabel('X1',fontsize=18)
plt.ylabel('X2',fontsize=18)
plt.title('Clusters with different std',fontsize=18)
plt.show()
```



In [56]:
```
# Estimate K value
model, SilAvg, wcss= estimate_Ks(Xneq_s)
```

```
('n_clusters =', 2, 'Avg. silhouette score:', 0.47791885194743483, 'Inetria:',
6936.059064575423)
('n_clusters =', 3, 'Avg. silhouette score:', 0.5684518064743393, 'Inetria:',
3586.251105630054)
('n_clusters =', 4, 'Avg. silhouette score:', 0.5897414605537085, 'Inetria:',
2434.4993280194813)
('n_clusters =', 5, 'Avg. silhouette score:', 0.5918785149140441, 'Inetria:',
1746.6403676729235)
('n_clusters =', 6, 'Avg. silhouette score:', 0.6085776005404206, 'Inetria:',
1370.597238949345)
('n_clusters =', 7, 'Avg. silhouette score:', 0.5947440790239297, 'Inetria:',
1175.4329783227918)
('n_clusters =', 8, 'Avg. silhouette score:', 0.5952118442369955, 'Inetria:',
1041.5964168831833)
```

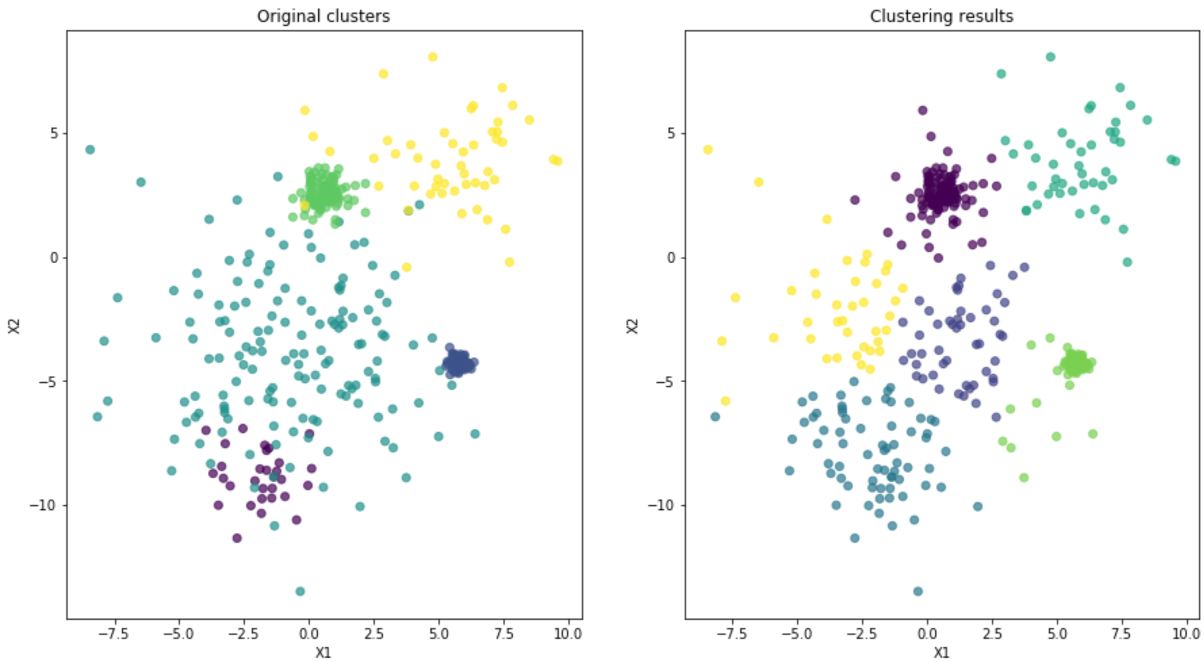


In [57]:
```
# wining estimator with max silhouettee average
win_index = SilAvg.index(np.max(SilAvg))
```

In [58]:
```
# Get the wining model
KmeansWin = model [win_index]
```

In [59]:
```
# Predict the new labels
y_new = KmeansWin.predict(Xneq_s)
```

In [60]:
```
#plot results
plt.figure(figsize=[15,8])
plt.subplot(1,2,1)
plt.scatter (Xneq_s[:,0],Xneq_s[:,1], c= yneq_s, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Original clusters')
plt.subplot(1,2,2)
plt.scatter (Xneq_s[:,0],Xneq_s[:,1], c= y_new, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Clustering results')
plt.show()
```

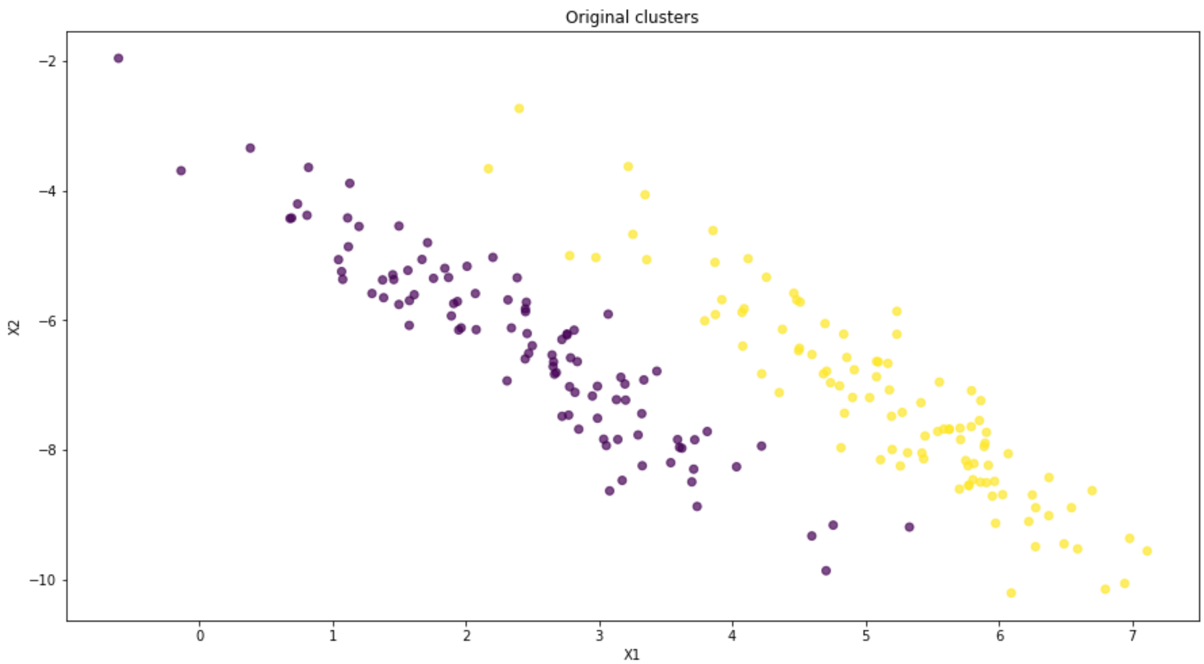


In [61]:
```
X, y = make_blobs (n_samples=200, centers =2, n_features=2, cluster_std=1.5, rando
```

In [62]:
```
# Anisotropicly distributed data Scikit (example)
transformation = [[0.60, -0.63], [-0.40, 0.85]]
X_aniso = np.dot(X, transformation)
```

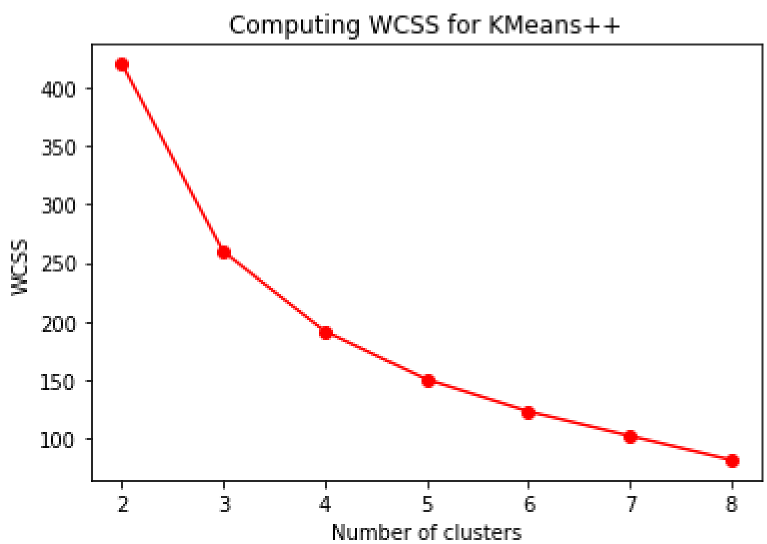
In [63]:
```
#plot results
plt.figure(figsize=[15,8])
plt.scatter (X_aniso[:,0],X_aniso[:,1], c= y, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Original clusters')
```

Out[63]:
```
Text(0.5,1,'Original clusters')
```



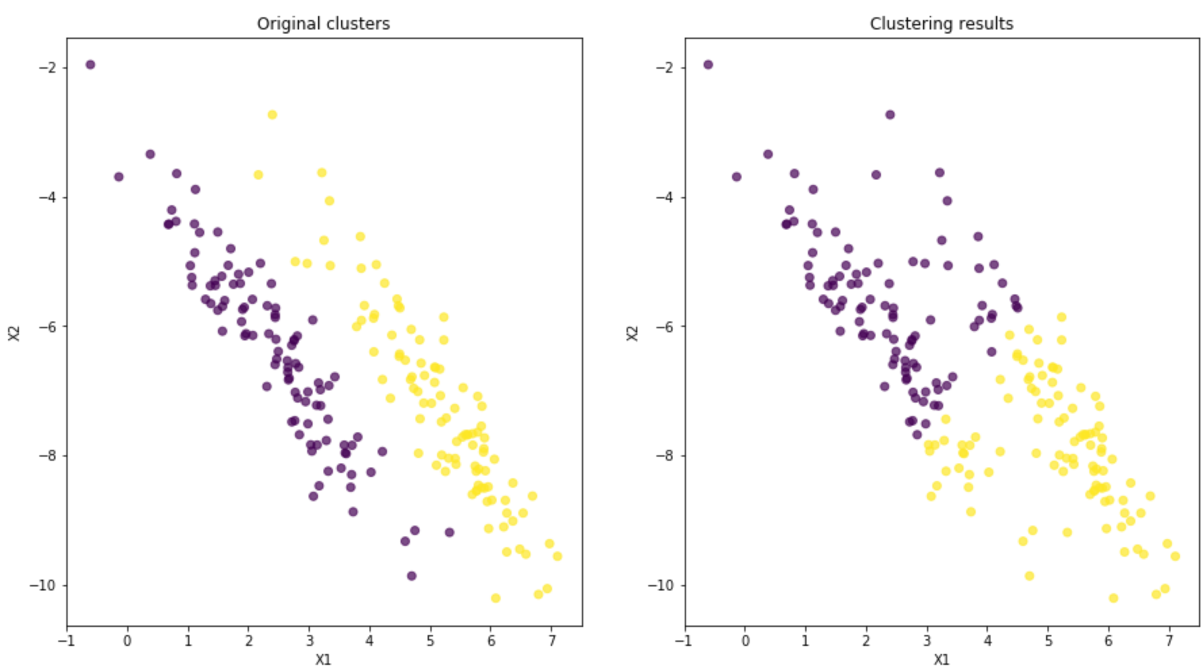
In [64]:
```
# Estimate K value
model, SilAvg, wcss= estimate_Ks(X_aniso)
```

```
('n_clusters =', 2, 'Avg. silhouette score:', 0.49661422560410196, 'Inetria:',
419.6778979853226)
('n_clusters =', 3, 'Avg. silhouette score:', 0.44034537571080995, 'Inetria:',
259.61414184474086)
('n_clusters =', 4, 'Avg. silhouette score:', 0.46822662956556743, 'Inetria:',
191.9976804672031)
('n_clusters =', 5, 'Avg. silhouette score:', 0.449059400064981, 'Inetria:', 1
50.8587509559075)
('n_clusters =', 6, 'Avg. silhouette score:', 0.42188412786503454, 'Inetria:',
123.49374686405702)
('n_clusters =', 7, 'Avg. silhouette score:', 0.42010661242924513, 'Inetria:',
102.61785868917205)
('n_clusters =', 8, 'Avg. silhouette score:', 0.43933764590853913, 'Inetria:',
82.07529583241254)
```



In [65]:

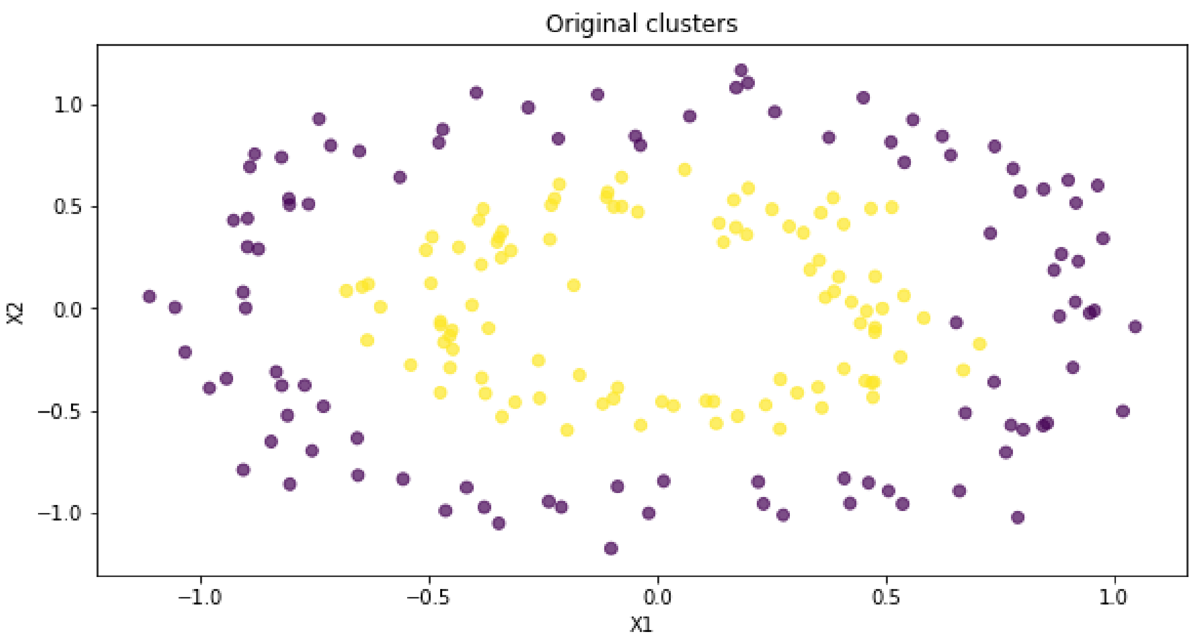
```
# wining estimator with max silhouettee average
win_index = SilAvg.index(np.max(SilAvg))
# Get the wining model
KmeansWin = model [win_index]
# Predict the new labels
y_new = KmeansWin.predict(X_aniso)
#plot results
plt.figure(figsize=[15,8])
plt.subplot(1,2,1)
plt.scatter (X_aniso[:,0],X_aniso[:,1], c= y, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Original clusters')
plt.subplot(1,2,2)
plt.scatter (X_aniso[:,0],X_aniso[:,1], c= y_new, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Clustering results')
plt.show()
```



Circles

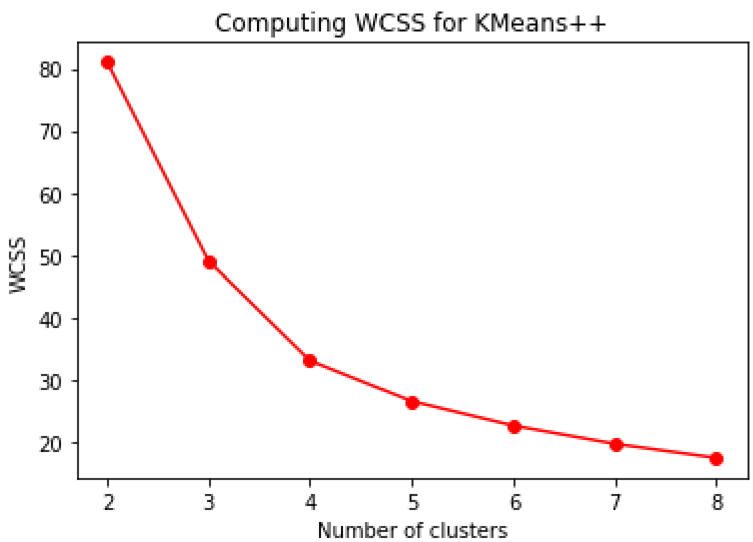
In [66]:
```
from sklearn.datasets import make_circles
X, y = make_circles (n_samples=200, noise=0.1, factor=0.5, random_state=40)
#plot results
plt.figure(figsize=[10,5])
plt.scatter (X[:,0],X[:,1], c= y, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Original clusters')
```

Out[66]: Text(0.5,1,'Original clusters')

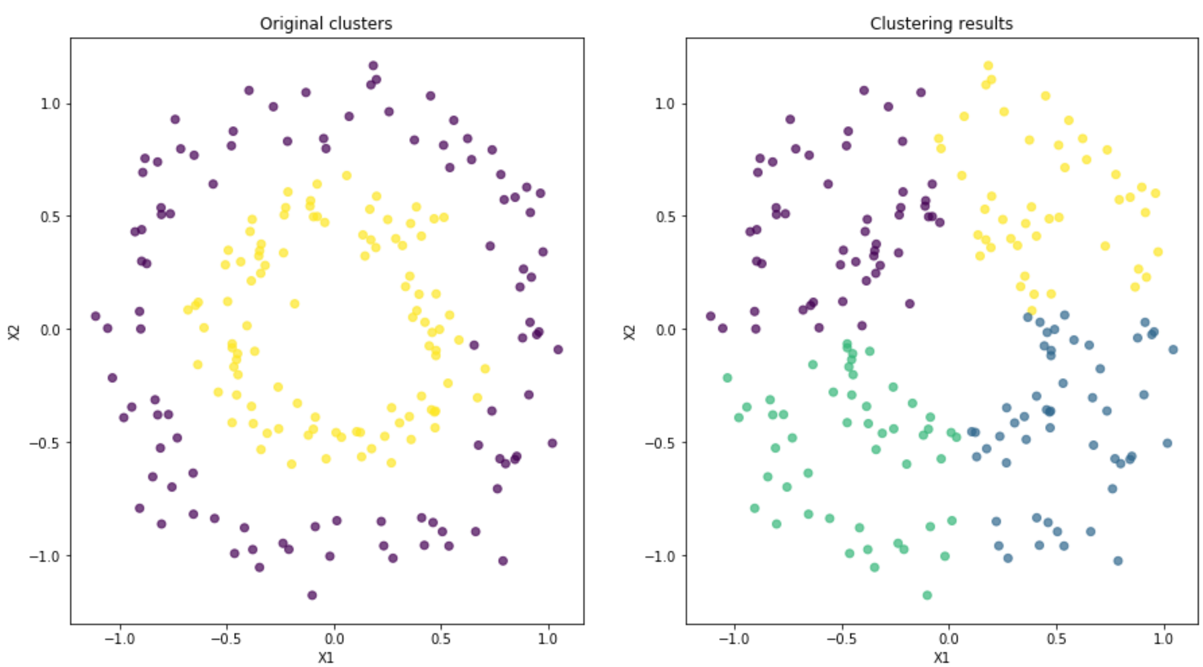


In [67]:
```
# Estimate K value
model, SilAvg, wcss= estimate_Ks(X)
```

```
('n_clusters =', 2, 'Avg. silhouette score:', 0.35045903254835364, 'Inetria:', 81.05677507201548)
('n_clusters =', 3, 'Avg. silhouette score:', 0.3854049225565831, 'Inetria:', 49.13016179820383)
('n_clusters =', 4, 'Avg. silhouette score:', 0.38882904446444655, 'Inetria:', 33.147339529206626)
('n_clusters =', 5, 'Avg. silhouette score:', 0.366079401448284, 'Inetria:', 26.64795986115853)
('n_clusters =', 6, 'Avg. silhouette score:', 0.33187027921204565, 'Inetria:', 22.731747444352422)
('n_clusters =', 7, 'Avg. silhouette score:', 0.34240506906594553, 'Inetria:', 19.80371551193892)
('n_clusters =', 8, 'Avg. silhouette score:', 0.3449455580616376, 'Inetria:', 17.549289111650122)
```



In [68]:
```
# wining estimator with max silhouettee average
win_index = SilAvg.index(np.max(SilAvg))
# Get the wining model
KmeansWin = model [win_index]
# Predict the new labels
y_new = KmeansWin.predict(X)
#plot results
plt.figure(figsize=[15,8])
plt.subplot(1,2,1)
plt.scatter (X[:,0],X[:,1], c= y, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Original clusters')
plt.subplot(1,2,2)
plt.scatter (X[:,0],X[:,1], c= y_new, alpha = 0.7 )
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Clustering results')
plt.show()
```



Optional Question

In [69]:
```
import pandas as pd
# load data from datafolder
dx = pd.read_csv('data\CC GENERAL.csv')
```

In [70]:
```
#feature names
print(dx.columns.values)
```

```
['CUST_ID' 'BALANCE' 'BALANCE_FREQUENCY' 'PURCHASES' 'ONEOFF_PURCHASES'
 'INSTALLMENTS_PURCHASES' 'CASH_ADVANCE' 'PURCHASES_FREQUENCY'
 'ONEOFF_PURCHASES_FREQUENCY' 'PURCHASES_INSTALLMENTS_FREQUENCY'
 'CASH_ADVANCE_FREQUENCY' 'CASH_ADVANCE_TRX' 'PURCHASES_TRX'
 'CREDIT_LIMIT' 'PAYMENTS' 'MINIMUM_PAYMENTS' 'PRC_FULL_PAYMENT' 'TENURE']
```

Data Preparation

- The raw data may contain undesided information
- May have missing data
- Not all ML algorithms supports missing values
- We need to check the data first

In [71]:
```
# For the train set
dx.describe()
```

Out[71]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

OK! there are some missing data in here

In [72]:
```
# another way to figureout that is
print(dx.isna().sum())
```

```
CUST_ID                               0
BALANCE                               0
BALANCE_FREQUENCY                     0
PURCHASES                             0
ONEOFF_PURCHASES                      0
INSTALLMENTS_PURCHASES                0
CASH_ADVANCE                          0
PURCHASES_FREQUENCY                   0
ONEOFF_PURCHASES_FREQUENCY            0
PURCHASES_INSTALLMENTS_FREQUENCY      0
CASH_ADVANCE_FREQUENCY                0
CASH_ADVANCE_TRX                      0
PURCHASES_TRX                         0
CREDIT_LIMIT                          1
PAYMENTS                              0
MINIMUM_PAYMENTS                    313
PRC_FULL_PAYMENT                      0
TENURE                                0
dtype: int64
```

In [73]:
```
# Just in case, Pandas provides the fillna() function to replace missing values wi
# Fill missing values with mean column values in the train set
dx.fillna(dx.median(), inplace=True)
```

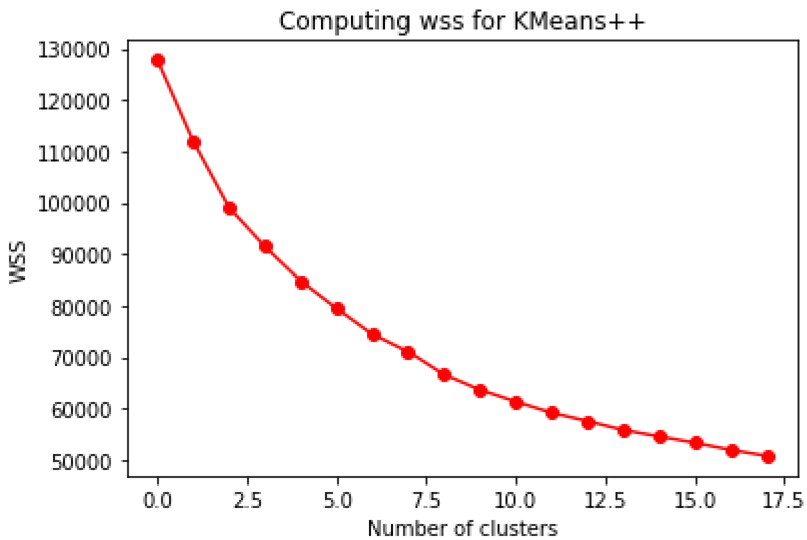
In [74]:
```
# We'll use all columns expect the CUST_ID
vals = dx.iloc[ :, 1:].values
```

In [75]:
```
from sklearn.preprocessing import StandardScaler
z_score= StandardScaler().fit(vals)
vals = z_score.transform(vals)
```

In [76]:
```
# Estimate K value
model, SilAvg, wcss= estimate_Ks(vals, max_clusters=20, plotFlag = False)
```

```
('n_clusters =', 2, 'Avg. silhouette score:', 0.2095641227607062, 'Inetria:',
127784.41768270916)
('n_clusters =', 3, 'Avg. silhouette score:', 0.2505645588142349, 'Inetria:',
111973.8440272938)
('n_clusters =', 4, 'Avg. silhouette score:', 0.1976791965228765, 'Inetria:',
99061.93984229019)
('n_clusters =', 5, 'Avg. silhouette score:', 0.19325195080511473, 'Inetria:',
91491.11253232486)
('n_clusters =', 6, 'Avg. silhouette score:', 0.20281912418016176, 'Inetria:',
84826.49993909677)
('n_clusters =', 7, 'Avg. silhouette score:', 0.21459002657465195, 'Inetria:',
79502.74783641608)
('n_clusters =', 8, 'Avg. silhouette score:', 0.22193282658122515, 'Inetria:',
74483.51387414828)
('n_clusters =', 9, 'Avg. silhouette score:', 0.21569986719782255, 'Inetria:',
71049.00401376278)
('n_clusters =', 10, 'Avg. silhouette score:', 0.22028760410426104, 'Inetri
a:', 66577.00127958001)
('n_clusters =', 11, 'Avg. silhouette score:', 0.216608745521806, 'Inetria:',
63635.82481325469)
('n_clusters =', 12, 'Avg. silhouette score:', 0.2164853901096295, 'Inetria:',
61335.510302283394)
('n_clusters =', 13, 'Avg. silhouette score:', 0.2188964323088831, 'Inetria:',
59159.13283811935)
('n_clusters =', 14, 'Avg. silhouette score:', 0.21961060821772874, 'Inetri
a:', 57540.37241259009)
('n_clusters =', 15, 'Avg. silhouette score:', 0.1994586522107506, 'Inetria:',
55814.27968398782)
('n_clusters =', 16, 'Avg. silhouette score:', 0.20585500333351175, 'Inetri
a:', 54560.1427517541)
('n_clusters =', 17, 'Avg. silhouette score:', 0.20757770876569642, 'Inetri
a:', 53324.5905679345)
('n_clusters =', 18, 'Avg. silhouette score:', 0.20568725878769945, 'Inetri
a:', 51938.68036161958)
('n_clusters =', 19, 'Avg. silhouette score:', 0.2071009408742376, 'Inetria:',
50779.38879196766)
```

In [77]:
```
# it is hard to estimate k becuase WSS will decrease by increasing number of Ks
plt.plot( wcss, 'ro-')
plt.title("Computing wss for KMeans++")
plt.xlabel("Number of clusters")
plt.ylabel("WSS")
plt.show()
```



maybe we want to go beyoud 7 clusters

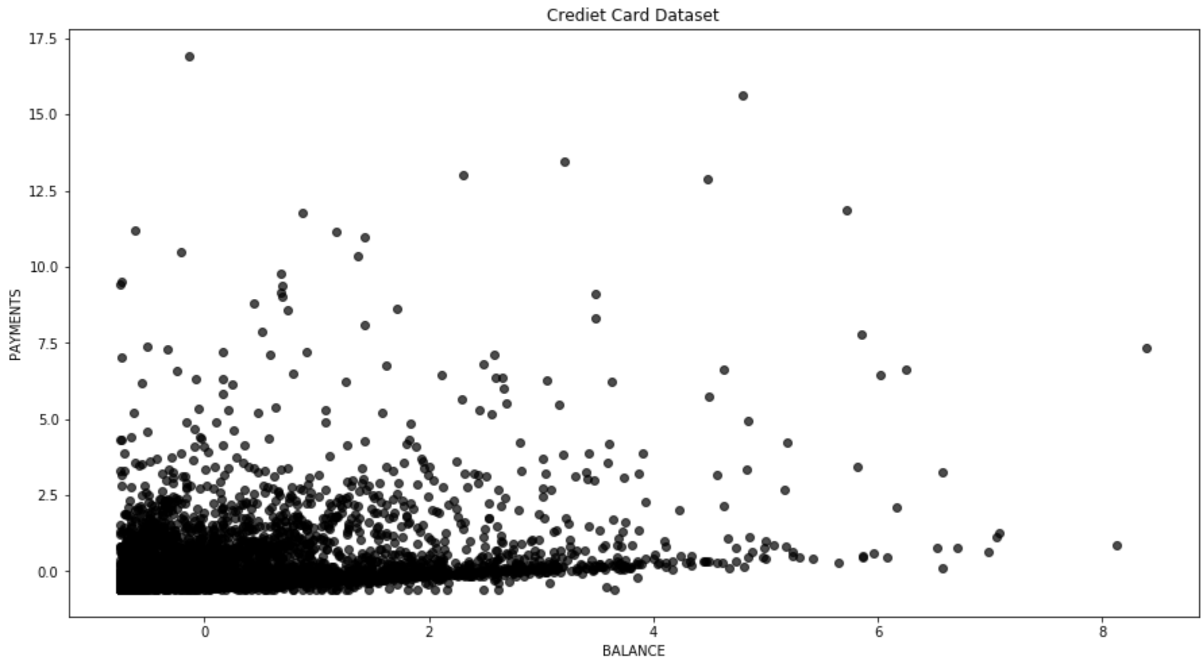
In [78]:
```
# Using Sillhouettee
# wining estimator with max silhouettee average
win_index = SilAvg.index(np.max(SilAvg))
# Get the wining model
print('Number of clusters must be:', win_index+2)
```

```
('Number of clusters must be:', 3)
```

In [79]:
```
# let us select two features
# We'll use all columns expect the CUST_ID
vals = dx[['BALANCE','PAYMENTS'  ]].values
z_score= StandardScaler().fit(vals)
vals = z_score.transform(vals)
```

In [80]:
```
#plot results
plt.figure(figsize=[15,8])
plt.scatter (vals[:,0],vals[:,1], c= 'k', alpha = 0.7 )
plt.xlabel('BALANCE')
plt.ylabel('PAYMENTS')
plt.title('Crediet Card Dataset')
```

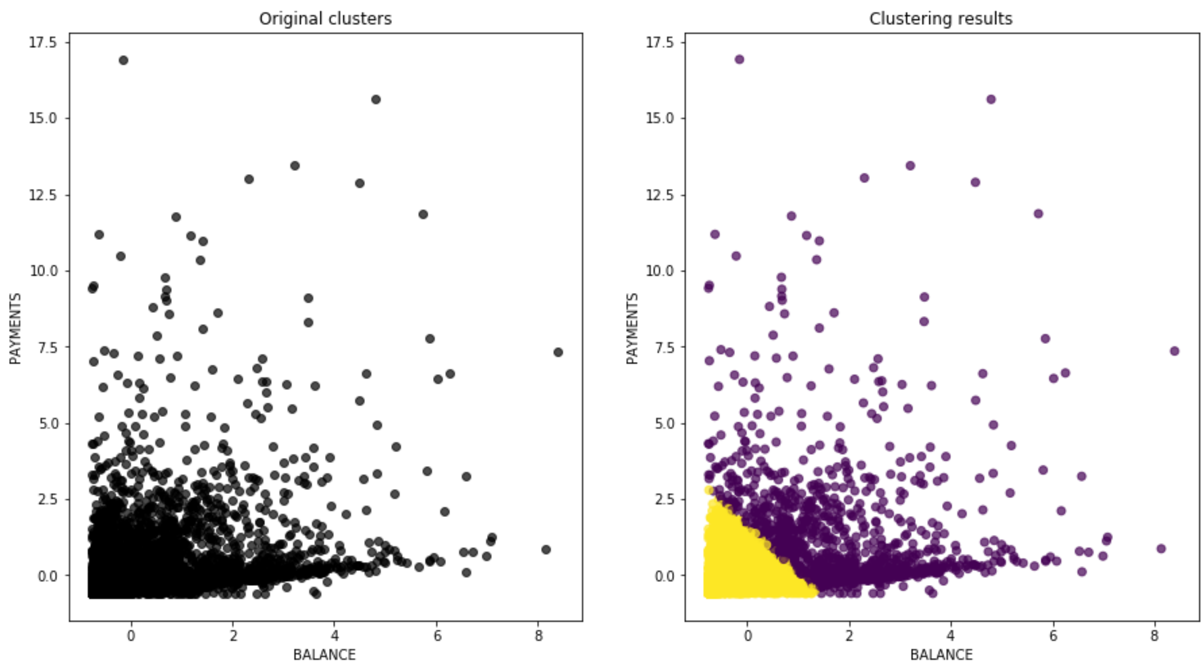
Out[80]: Text(0.5,1,'Crediet Card Dataset')



In [81]:
```
# Estimate K value
model, SilAvg, wcss= estimate_Ks(vals, max_clusters=20, plotFlag = False)
```

```
('n_clusters =', 2, 'Avg. silhouette score:', 0.6526482984856432, 'Inetria:',
10512.557751343698)
('n_clusters =', 3, 'Avg. silhouette score:', 0.6279869910306692, 'Inetria:',
7000.245211394776)
('n_clusters =', 4, 'Avg. silhouette score:', 0.6151432029747265, 'Inetria:',
5248.047372416747)
('n_clusters =', 5, 'Avg. silhouette score:', 0.5128444339485208, 'Inetria:',
4062.1216437635517)
('n_clusters =', 6, 'Avg. silhouette score:', 0.5168417905973602, 'Inetria:',
3370.5176008030858)
('n_clusters =', 7, 'Avg. silhouette score:', 0.4763986196169063, 'Inetria:',
2811.5317894687855)
('n_clusters =', 8, 'Avg. silhouette score:', 0.47840966243606536, 'Inetria:',
2472.684468818252)
('n_clusters =', 9, 'Avg. silhouette score:', 0.4603348430445739, 'Inetria:',
2206.353305597314)
('n_clusters =', 10, 'Avg. silhouette score:', 0.4626102137470566, 'Inetria:',
1975.7937903868365)
('n_clusters =', 11, 'Avg. silhouette score:', 0.47268634202422466, 'Inetri
a:', 1767.04191981705)
('n_clusters =', 12, 'Avg. silhouette score:', 0.47292418183681034, 'Inetri
a:', 1603.5975824621758)
('n_clusters =', 13, 'Avg. silhouette score:', 0.4734190972522372, 'Inetria:',
1451.9793274583142)
('n_clusters =', 14, 'Avg. silhouette score:', 0.4798015396852457, 'Inetria:',
1323.287094569807)
('n_clusters =', 15, 'Avg. silhouette score:', 0.4801392472443901, 'Inetria:',
1210.5643908343327)
('n_clusters =', 16, 'Avg. silhouette score:', 0.4687870783624091, 'Inetria:',
1130.744486474692)
('n_clusters =', 17, 'Avg. silhouette score:', 0.47364265519154225, 'Inetri
a:', 1042.41138720733)
('n_clusters =', 18, 'Avg. silhouette score:', 0.4701300741351766, 'Inetria:',
970.3438152609627)
('n_clusters =', 19, 'Avg. silhouette score:', 0.4755179430790857, 'Inetria:',
922.3149838128325)
```

In [82]:
```
# wining estimator with max silhouettee average
win_index = SilAvg.index(np.max(SilAvg))
# Get the wining model
KmeansWin = model [win_index]
# Predict the new labels
y_new = KmeansWin.predict(vals)
#plot results
plt.figure(figsize=[15,8])
plt.subplot(1,2,1)
plt.scatter (vals[:,0],vals[:,1], c= 'k', alpha = 0.7 )
plt.xlabel('BALANCE')
plt.ylabel('PAYMENTS')
plt.title('Original clusters')
plt.subplot(1,2,2)
plt.scatter (vals[:,0],vals[:,1], c= y_new, alpha = 0.7 )
plt.xlabel('BALANCE')
plt.ylabel('PAYMENTS')
plt.title('Clustering results')
plt.show()
```



In [ ]: