Q1: Kmeans algorithm in Python from scratch

(a)

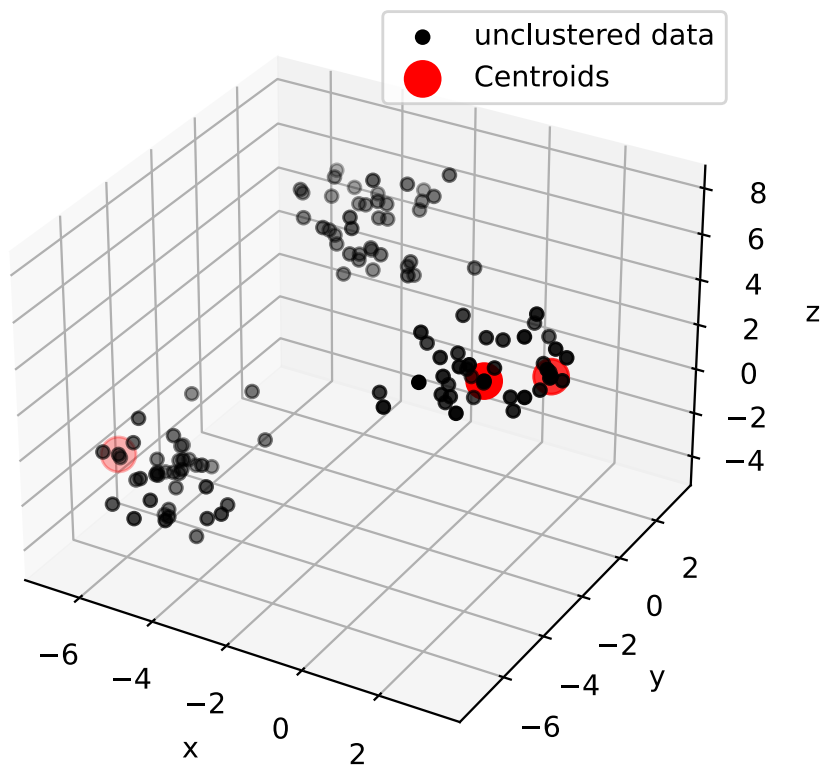
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random as rd
from collections import defaultdict
import matplotlib.cm as cm
```

In [17]:
```
# read dataset
data = pd.read_csv("alpha_shape.csv")
X = data.values
# centroids of our clusters
Centroids=np.array([]).reshape(3,0)
K = 3
rd.seed(10)
for i in range(K): #initialize our centroids randomly
    rand=rd.randint(0,len(X)-1)
    Centroids=np.c_[Centroids,X[rand]]
Centroids
```

Out[17]:
```
array([[ 3.01304436, -6.36875405,  2.14329881],
       [-1.78295949, -5.65020464, -3.44530181],
       [ 3.69825523, -1.16589813,  4.40986279]])
```

In [25]:
```
#Visualize the randomly initialized centroid values
fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(projection='3d')
ax.scatter(X[:,0],X[:,1],X[:,2],c = "black",label = "unclustered data")
ax.scatter(Centroids[0,:],Centroids[1,:],Centroids[2,:],s = 150,c = "red",label = "Centroids")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
ax.legend()
```

Out[25]:
```
<matplotlib.legend.Legend at 0x7fbc58c44280>
```



(b)

In [101…
```
def kmeans(x,k,num_iter):
    m = len(x)
    Centroids=np.array([]).reshape(3,0)
    rd.seed(10)
    for i in range(k): #initialize our centroids randomly
        rand=rd.randint(0,m-1)
        Centroids=np.c_[Centroids,x[rand]]

    for n in range(num_iter):
    # 1] Store distances of the data points from all the centroids
    # centroids of our clusters


        EuclideanDistance=np.array([]).reshape(m,0)
        #ManhattanDistance=np.array([]).reshape(m,0)
        for i in range(k):
            tempDist=np.sum((x-Centroids[:,i])**2,axis=1)
            EuclideanDistance=np.c_[EuclideanDistance,tempDist]
        # 2] Gets minimum of all distances found and assigns a number between 1 and 5 to each training example
        C=np.argmin(EuclideanDistance,axis=1)+1
        # 3] Initialize dictionary to store (x,y) coordinates for each point
        Y={}
        for i in range(k):
            Y[i+1]=np.array([]).reshape(3,0)
        # 4] For each training instance, we store it's coorinates in the category allocated to it
        # (print values of C to visualize)
        for i in range(m):
            Y[C[i]]=np.c_[Y[C[i]],x[i]] # C[i] : number between 1 and 5, the 'key' of Y

        # Change shape of dictionary values
        for i in range(k):
            Y[i+1]=Y[i+1].T

        # 5] Update our centroids
        for i in range(k):
            Centroids[:,i]=np.mean(Y[i+1],axis=0)

    return Y,Centroids,C
```

In [119…
```
Y,Centroids,C = kmeans(X,3,20)
```

In [104…
```
Centroids1 = Centroids[:,0]
Centroids2 = Centroids[:,1]
Centroids3 = Centroids[:,2]
print("Centroids1: ",Centroids1)
print("Centroids2: ",Centroids2)
print("Centroids3: ",Centroids3)
```

```
Centroids1:  [ 1.58614169 -2.16739026  3.62546335]
Centroids2:  [-5.32056873 -4.91565431 -2.07455069]
Centroids3:  [-3.18814754  0.94220378  5.52755187]
```

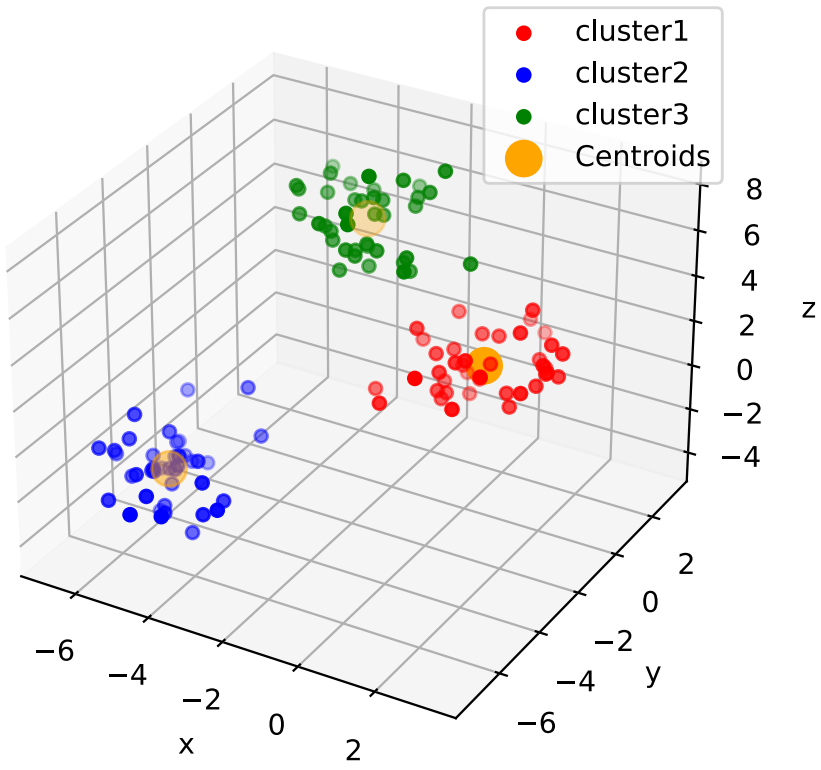
In [105…
```
C
```

Out[105…
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 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, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
```

(c)

In [106…
```
#Final Output:
color=['red','blue','green']
labels=['cluster1','cluster2','cluster3']

fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(projection='3d')

for i in range(3):
    ax.scatter(Y[i+1][:,0],Y[i+1][:,1],Y[i+1][:,2],c=color[i],label=labels[i])

ax.scatter(Centroids[0,:],Centroids[1,:],Centroids[2,:],s = 150,c = "orange",label = "Centroids")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
ax.legend()
```

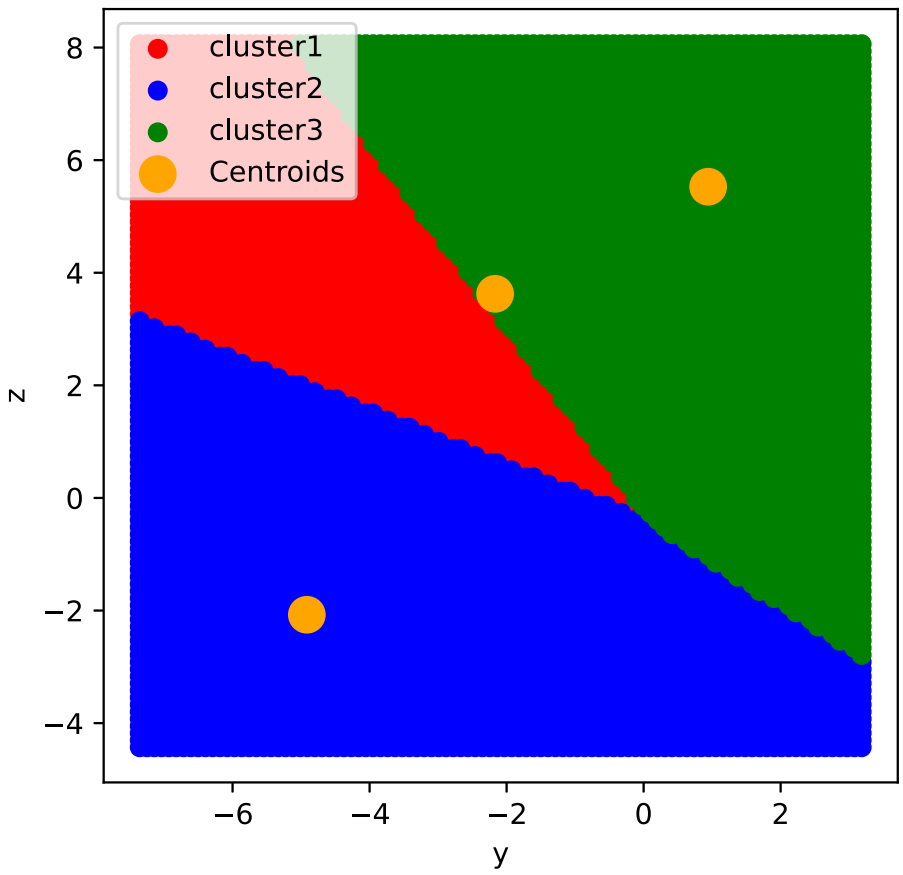
Out[106… <matplotlib.legend.Legend at 0x7fbc58e74b50>



In [122…
```
k = 3
x1 = np.mean(X[:,0])
x1 = x1*np.ones((10000,1))
y1 = np.linspace(np.min(X[:,1]),np.max(X[:,1]),100)
z1 = np.linspace(np.min(X[:,2]),np.max(X[:,2]),100)
y1_new, z1_new = np.meshgrid(y1,z1)
new_X_1 = np.hstack((x1,y1_new.ravel().reshape(-1,1),z1_new.ravel().reshape(-1,1)))
n = len(new_X_1)
EuclideanDistance=np.array([]).reshape(n,0)
for i in range(k):
    tempDist=np.sum((new_X_1-Centroids[:,i])**2,axis=1)
    EuclideanDistance=np.c_[EuclideanDistance,tempDist]
C=np.argmin(EuclideanDistance,axis=1)+1
Y={}
for i in range(k):
    Y[i+1]=np.array([]).reshape(3,0)
for i in range(n):
    Y[C[i]]=np.c_[Y[C[i]],new_X_1[i]] # C[i] : number between 1 and 5, the 'key' of Y
for i in range(k):
    Y[i+1]=Y[i+1].T
```

In [123…
```
color=['red','blue','green']
labels=['cluster1','cluster2','cluster3']

fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot()

for i in range(3):
    ax.scatter(Y[i+1][:,1],Y[i+1][:,2],c=color[i],label=labels[i])

ax.scatter(Centroids[1,:],Centroids[2,:],s = 150,c = "orange",label = "Centroids")
ax.set_xlabel("y")
ax.set_ylabel("z")
#ax.set_zlabel("z")
ax.legend()
```

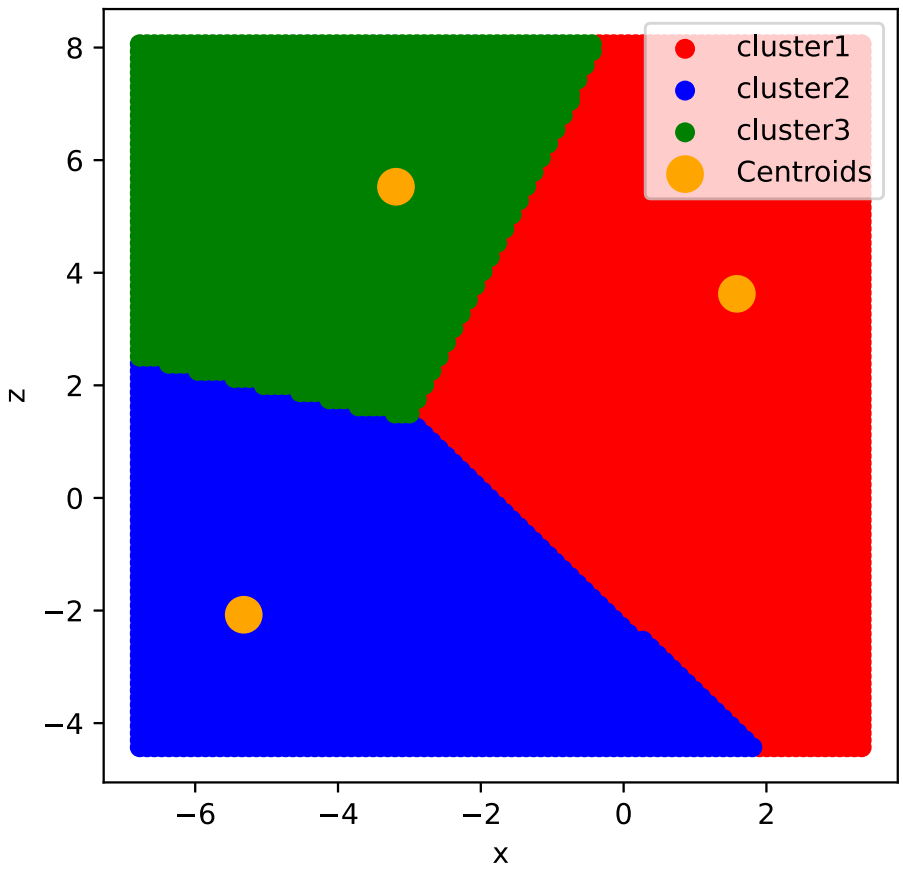
Out[123… <matplotlib.legend.Legend at 0x7fbc68c25580>



In [125…

```
k = 3
y2 = np.mean(X[:,1])
y2 = y2*np.ones((10000,1))
x2 = np.linspace(np.min(X[:,0]),np.max(X[:,0]),100)
z2 = np.linspace(np.min(X[:,2]),np.max(X[:,2]),100)
x2_new, z1_new = np.meshgrid(x2,z2)
new_X_2 = np.hstack((x2_new.ravel().reshape(-1,1),y2,z2_new.ravel().reshape(-1,1)))
n = len(new_X_2)
EuclideanDistance=np.array([]).reshape(n,0)
for i in range(k):
    tempDist=np.sum((new_X_2-Centroids[:,i])**2,axis=1)
    EuclideanDistance=np.c_[EuclideanDistance,tempDist]
C=np.argmin(EuclideanDistance,axis=1)+1
Y={}
for i in range(k):
    Y[i+1]=np.array([]).reshape(3,0)
for i in range(n):
    Y[C[i]]=np.c_[Y[C[i]],new_X_2[i]] # C[i] : number between 1 and 5, the 'key' of Y
for i in range(k):
    Y[i+1]=Y[i+1].T

color=['red','blue','green']
labels=['cluster1','cluster2','cluster3']

fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot()

for i in range(3):
    ax.scatter(Y[i+1][:,0],Y[i+1][:,2],c=color[i],label=labels[i])

ax.scatter(Centroids[0,:],Centroids[2,:],s = 150,c = "orange",label = "Centroids")
ax.set_xlabel("x")
ax.set_ylabel("z")
#ax.set_zlabel("z")
ax.legend()
```

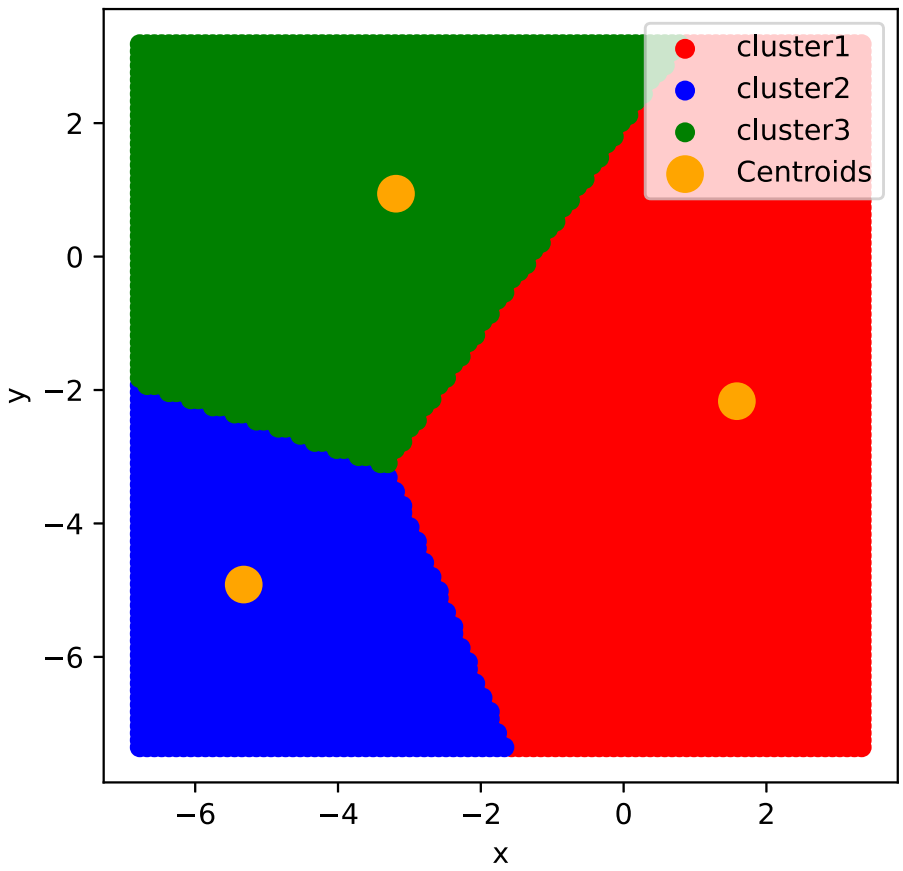
Out[125… <matplotlib.legend.Legend at 0x7fbc49aaba60>



In [126…

```
z3 = np.mean(X[:,2])
z3 = z3*np.ones((10000,1))
x3 = np.linspace(np.min(X[:,0]),np.max(X[:,0]),100)
y3 = np.linspace(np.min(X[:,1]),np.max(X[:,1]),100)
x3_new, y3_new = np.meshgrid(x3,y3)
new_X_3 = np.hstack((x3_new.ravel().reshape(-1,1),y3_new.ravel().reshape(-1,1),z3))
n = len(new_X_3)
EuclideanDistance=np.array([]).reshape(n,0)
for i in range(k):
    tempDist=np.sum((new_X_3-Centroids[:,i])**2,axis=1)
    EuclideanDistance=np.c_[EuclideanDistance,tempDist]
C=np.argmin(EuclideanDistance,axis=1)+1
Y={}
for i in range(k):
    Y[i+1]=np.array([]).reshape(3,0)
for i in range(n):
    Y[C[i]]=np.c_[Y[C[i]],new_X_3[i]] # C[i] : number between 1 and 5, the 'key' of Y
for i in range(k):
    Y[i+1]=Y[i+1].T

color=['red','blue','green']
labels=['cluster1','cluster2','cluster3']

fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot()

for i in range(3):
    ax.scatter(Y[i+1][:,0],Y[i+1][:,1],c=color[i],label=labels[i])

ax.scatter(Centroids[0,:],Centroids[1,:],s = 150,c = "orange",label = "Centroids")
ax.set_xlabel("x")
ax.set_ylabel("y")
#ax.set_zlabel("z")
ax.legend()
```

Out[126… <matplotlib.legend.Legend at 0x7fbc69335a90>



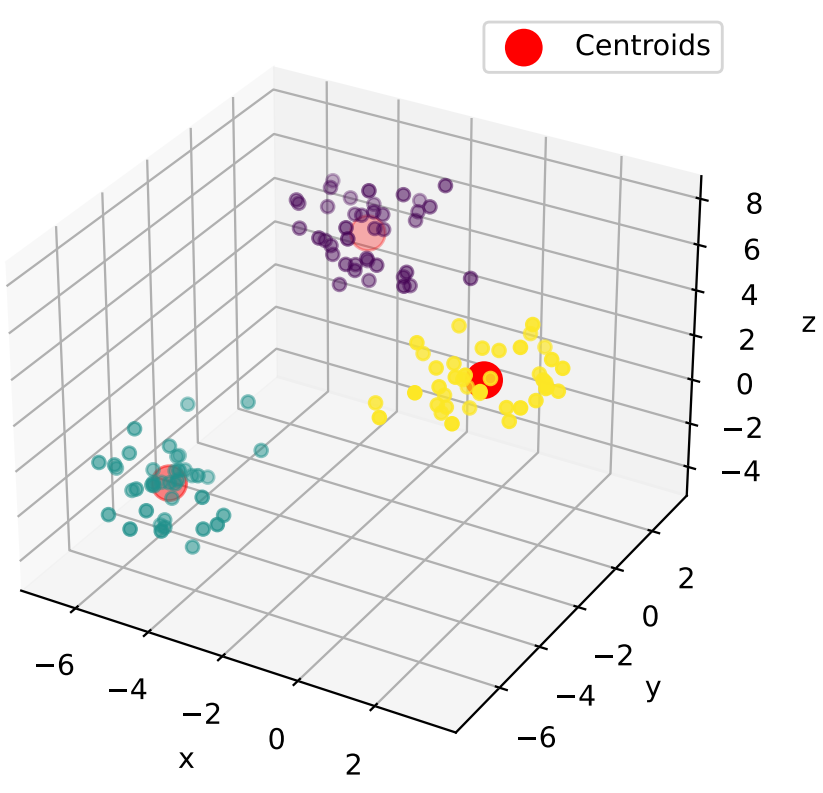
(d)

In [130…
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
print("labels: ",labels)
print("centroids: ",centroids)
```

```
labels:  [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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 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 0 0 0 0 0
 0 0 0 0 0 0 0 0 0]
centroids:  [[-3.18814754  0.94220378  5.52755187]
 [-5.32056873 -4.91565431 -2.07455069]
 [ 1.58614169 -2.16739026  3.62546335]]
```

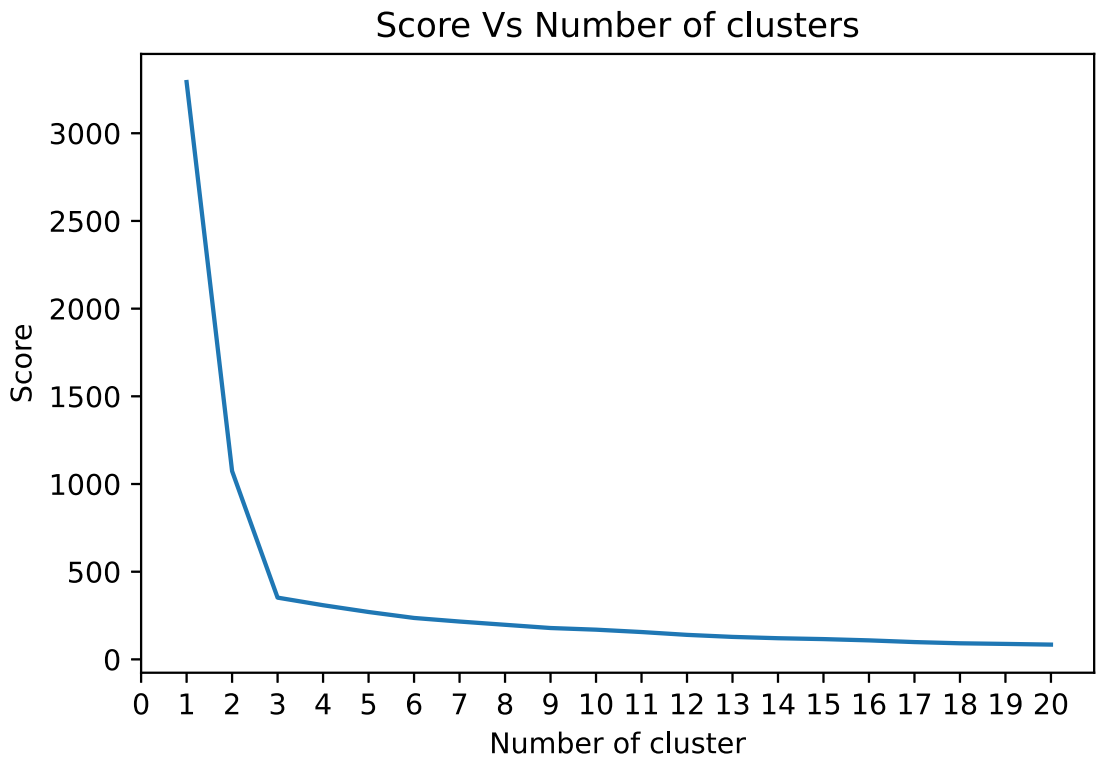
In [133…
```
#Visualize the randomly initialized centroid values
fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(projection='3d')
ax.scatter(X[:,0],X[:,1],X[:,2],c = labels)
ax.scatter(centroids[:,0],centroids[:,1],centroids[:,2],s = 150,c = "red",label = "Centroids")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
ax.legend()
```

Out[133… <matplotlib.legend.Legend at 0x7fbc69861d00>



(e)

In [142…
```
Score = []
clusters = []
for i in range(20):
    kmeans = KMeans(n_clusters=i+1, random_state=0).fit(X)
    Score.append(-kmeans.score(X))
    clusters.append(i+1)

plt.xlabel('Number of cluster')
plt.ylabel('Score')
plt.plot(clusters,Score)
plt.xticks(np.arange(0,21,step = 1))
plt.title('Score Vs Number of clusters')
plt.show()
```



I think K should be 3 because there's obviously a elbow point at K = 3.

Q2: Clustering Algorithms

(a) Spectral Clustering from Scratch

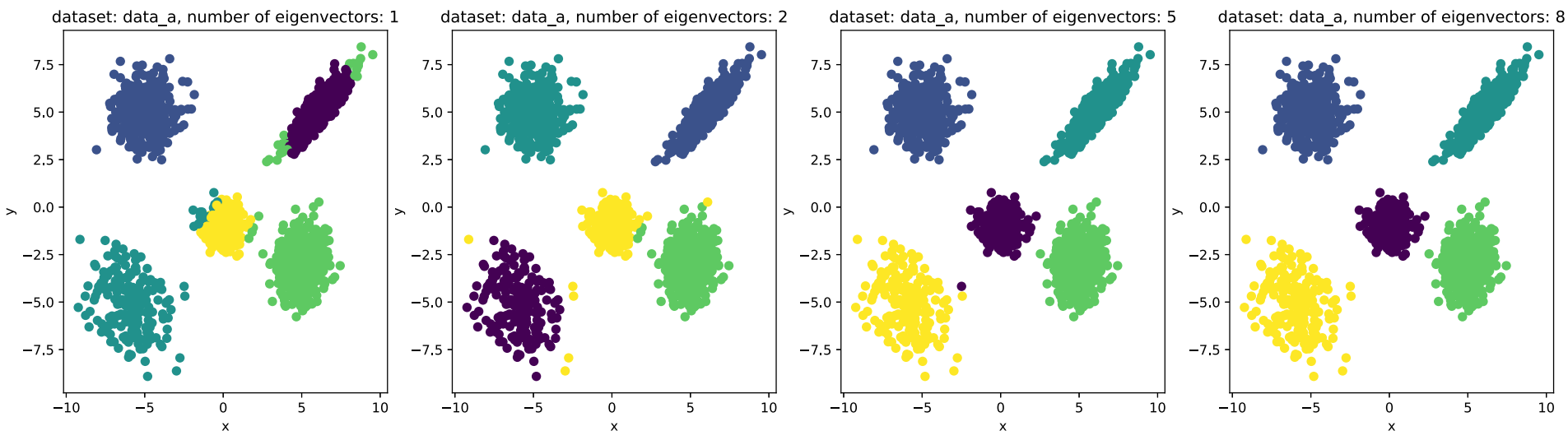
In [2]:
```
data_a = np.loadtxt("data_a.txt")
data_b = np.loadtxt("data_b.txt")
data_c = np.loadtxt("data_c.txt")
```

In [148…
```
def affinity_matrix(x,sigma):
    n = len(x)
    W = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            m = x[i]-x[j]
            W[i,j] = np.exp((-(m[0]**2+m[1]**2))/(sigma**2))
    return W
```

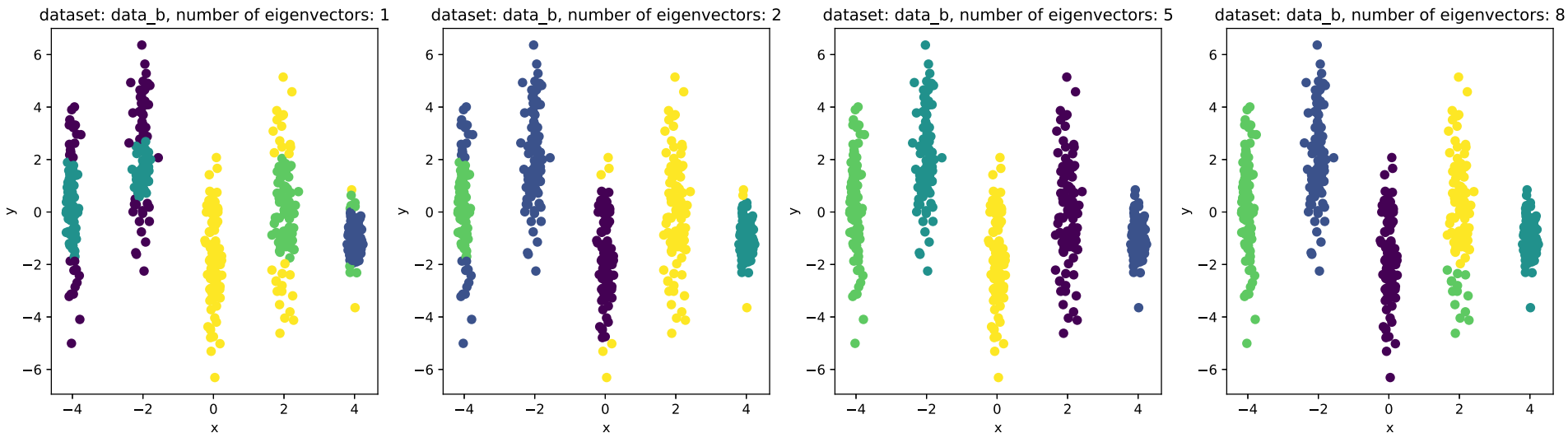
In [158…
```
def D_matrix(W):
    n = len(W)
    # D = np.eye((n,n))
    diag = np.sum(W,axis = 1)
    D = np.diag(diag**(-0.5))
    return D
```

In [260…
```
def draw(dataset,sigma,data_name):
    eigen_vector = [1,2,5,8]
    fig = plt.figure(figsize = (20,5))

    for i in eigen_vector:
        W = affinity_matrix(dataset,sigma)
        D = D_matrix(W)
        n = len(D)
        L = np.eye(n) - D@W@D
        w,v = np.linalg.eig(L)
        X = v[:,1:i+1]
        kmeans = KMeans(n_clusters=5, random_state=0).fit(np.real(X))
        labels = kmeans.labels_
        ax = fig.add_subplot(1,4,eigen_vector.index(i)+1)
        ax.scatter(dataset[:,0],dataset[:,1],c = labels)
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.set_title(f"dataset: {data_name}, number of eigenvectors: {i}")
```

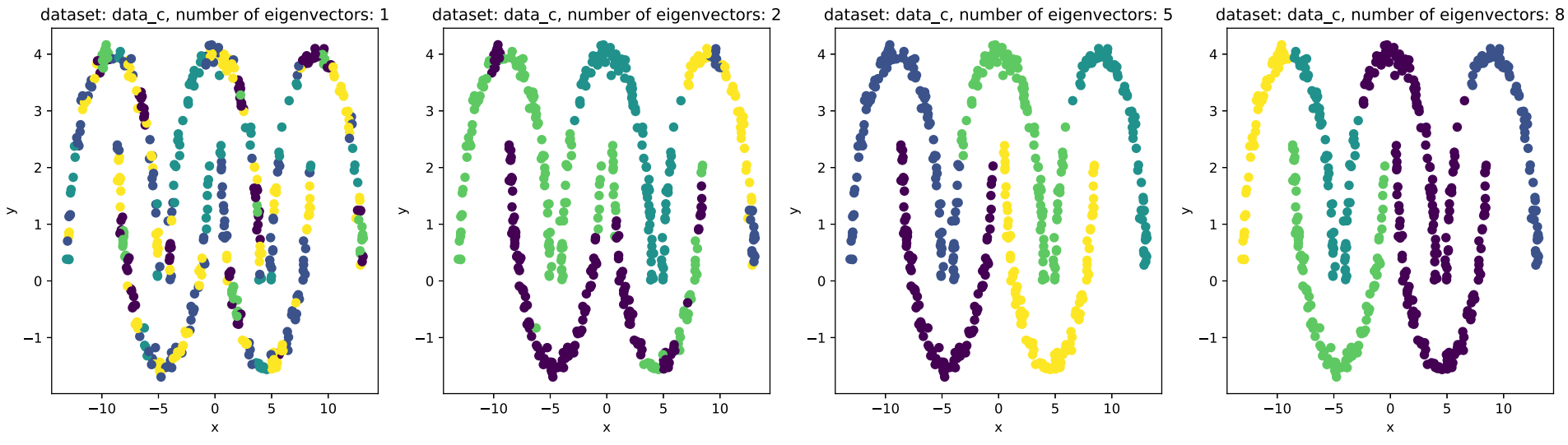
In [255…
```
# Dataset a
draw(data_a,4.0,"data_a")
```



In [261…
```
# Dataset b
draw(data_b,0.8,"data_b")
```



In [277…
```
# Dataset c
draw(data_c,0.6,"data_c")
```



Quality of Spectral Clustering as a function of the number of eigenvectors

Dataset (a): 8 > 5 > 2 > 1

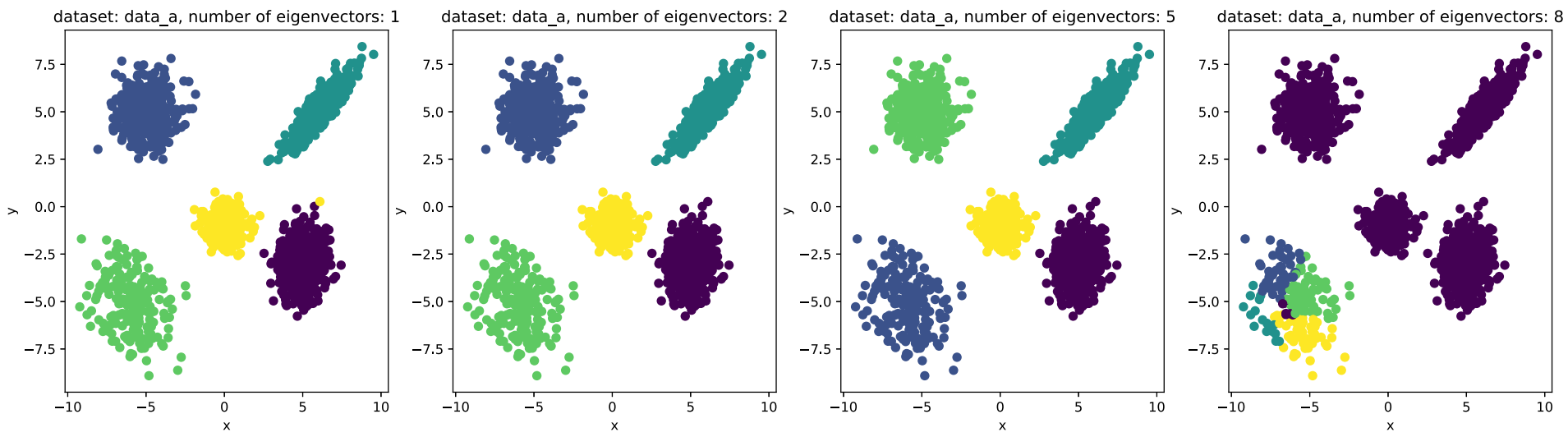
Dataset (b): 5 > 8 > 2 > 1

Dataset (c): 5 > 8 > 2 > 1

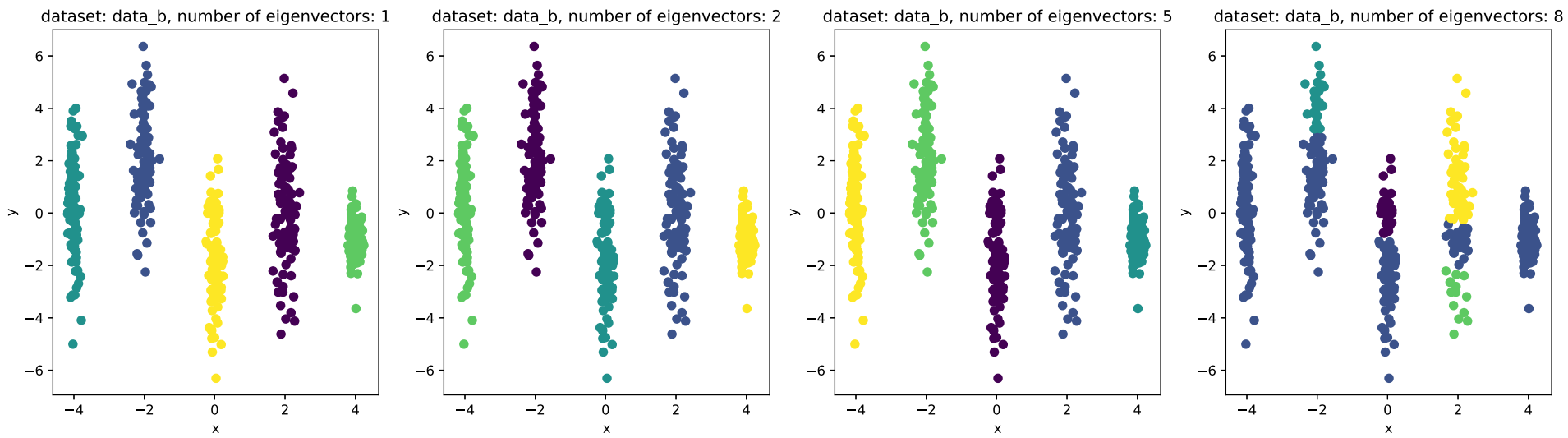
(b) Spectral Clustering using scikit

In [292…
```
from sklearn.cluster import SpectralClustering
def SC(dataset,sigma,data_name):
    eigen_vector = [1,2,5,8]
    fig = plt.figure(figsize = (20,5))

    for i in eigen_vector:
        clustering = SpectralClustering(n_clusters = 5,n_components = i,affinity = "rbf",random_state=5, gamma = s
        labels = clustering.labels_
        ax = fig.add_subplot(1,4,eigen_vector.index(i)+1)
        ax.scatter(dataset[:,0],dataset[:,1],c = labels)
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.set_title(f"dataset: {data_name}, number of eigenvectors: {i}")
```

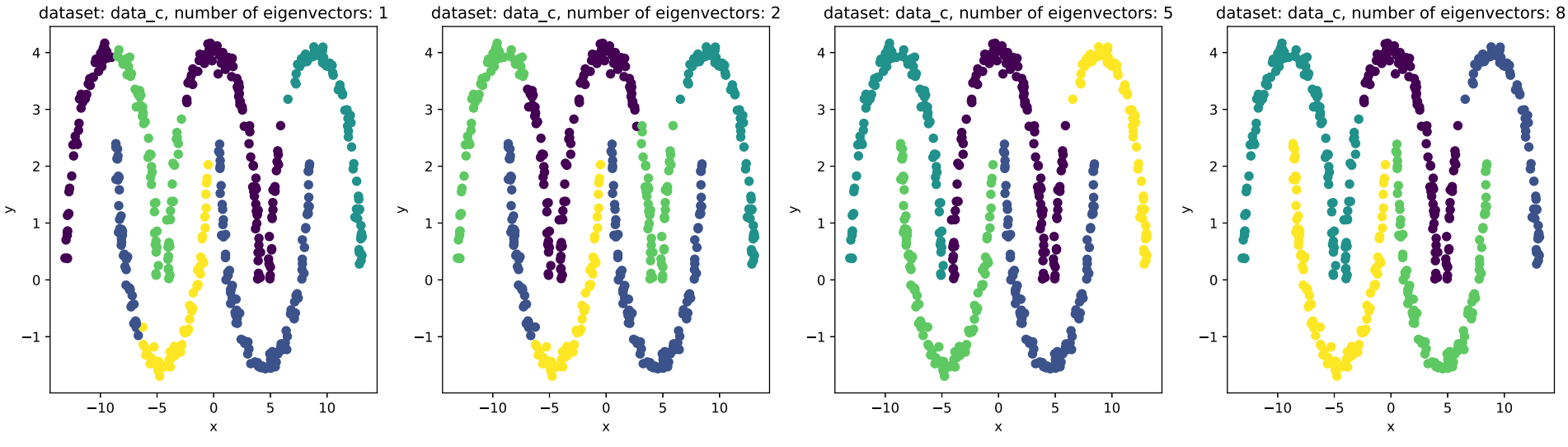
In [297…
```
# Dataset a
SC(data_a,1,"data_a")
```



In [293…
```
# Dataset b
SC(data_b,5,"data_b")
```



In [294…
```
# Dataset c
SC(data_c,5,"data_c")
```



Quality of Spectral Clustering as a function of the number of eigenvectors

Dataset (a): 5 = 2 > 1 > 8

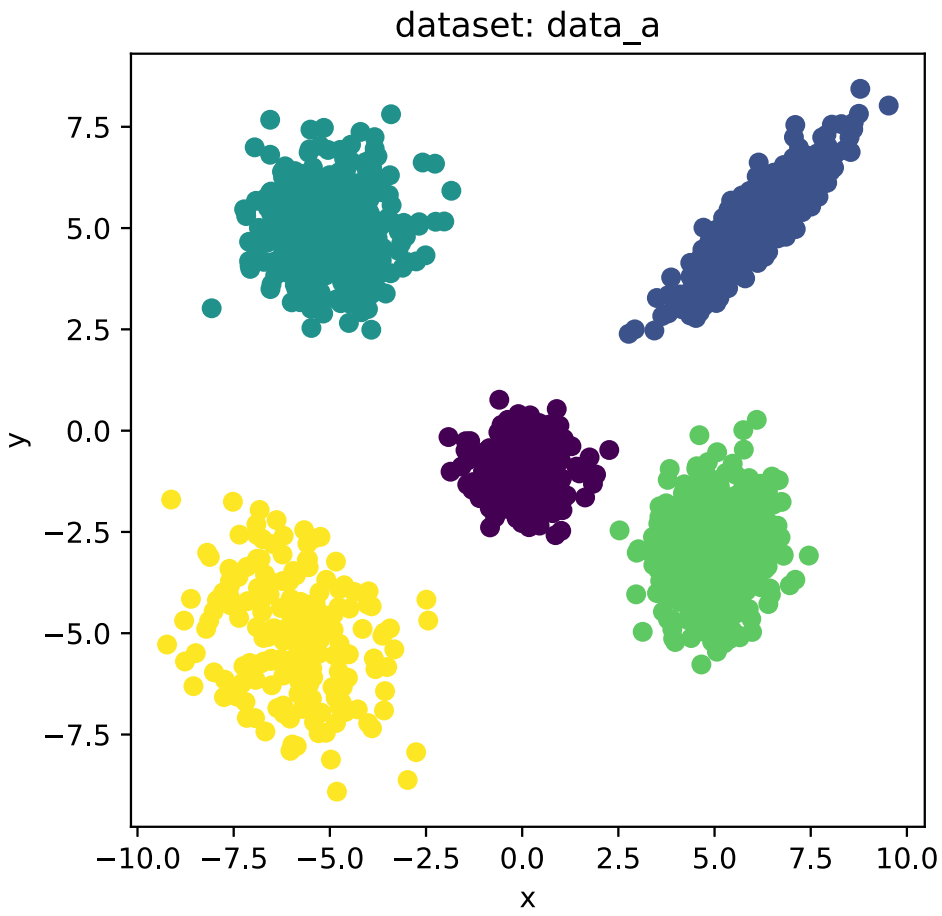
Dataset (b): 5 = 2 = 1 > 8

Dataset (c): 5 > 8 > 2 = 1

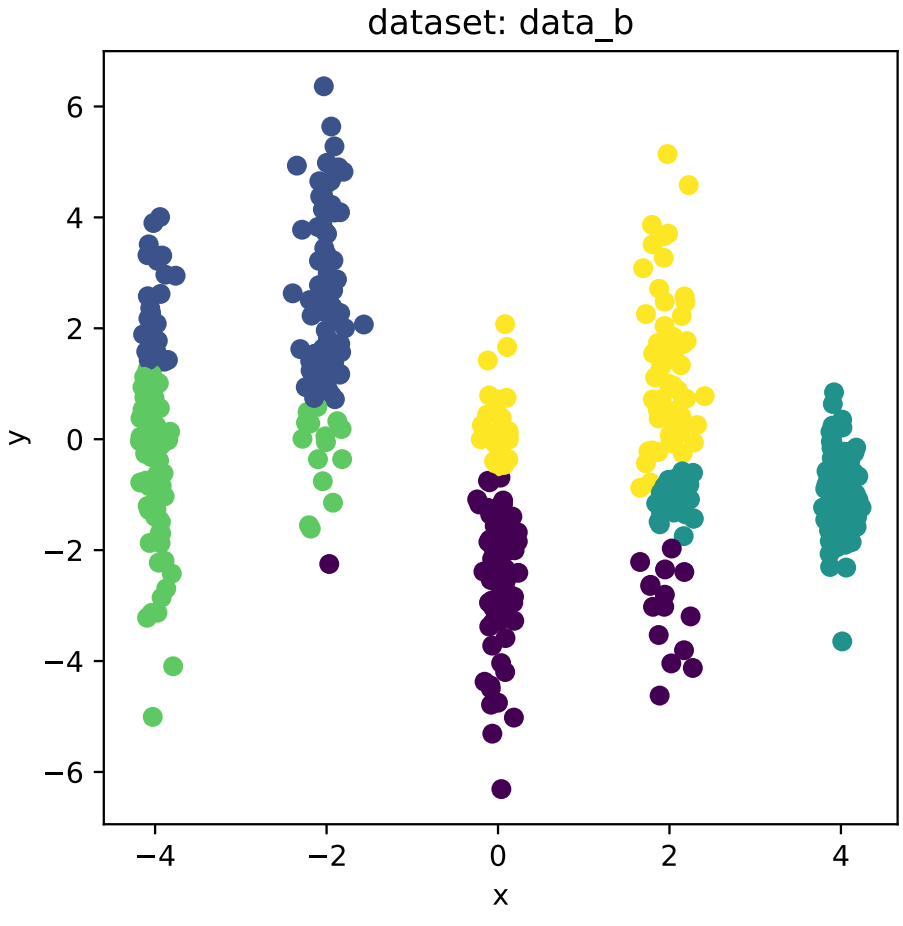
(c) Kmeans Clustering using scikit

In [299…
```
from sklearn.cluster import KMeans
def kmeans(dataset,dataname):
    kmeans = KMeans(n_clusters=5, random_state=0).fit(dataset)
    labels = kmeans.labels_
    fig = plt.figure(figsize = (5,5))
    plt.scatter(dataset[:,0],dataset[:,1],c = labels)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(f"dataset: {dataname}")
```

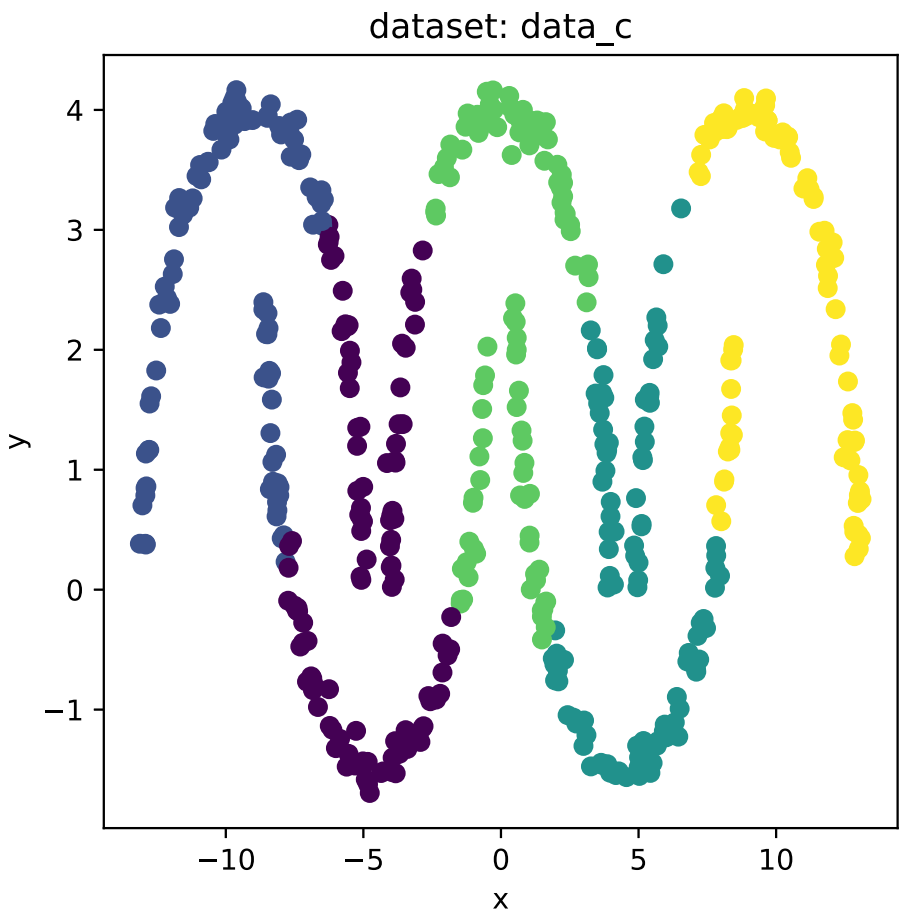
In [300…
```
# Dataset a
kmeans(data_a,"data_a")
```



In [301…
```
# Dataset b
kmeans(data_b,"data_b")
```



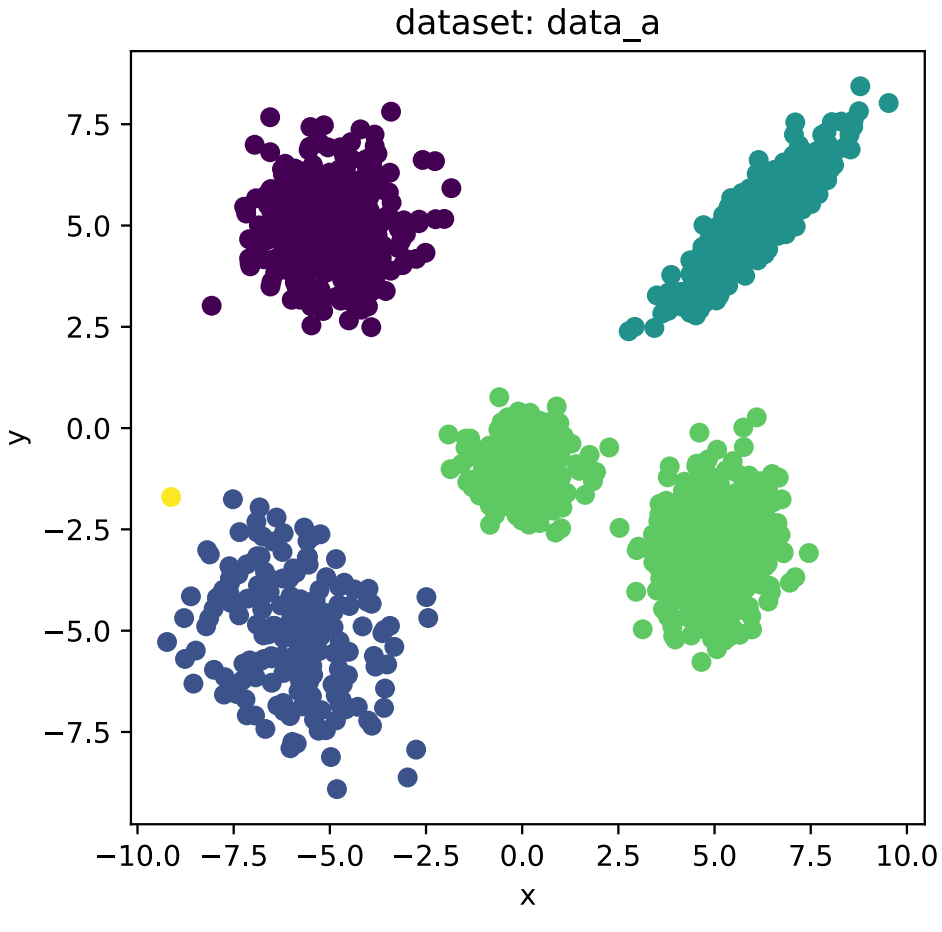
In [302…
```
# Dataset c
kmeans(data_c,"data_c")
```



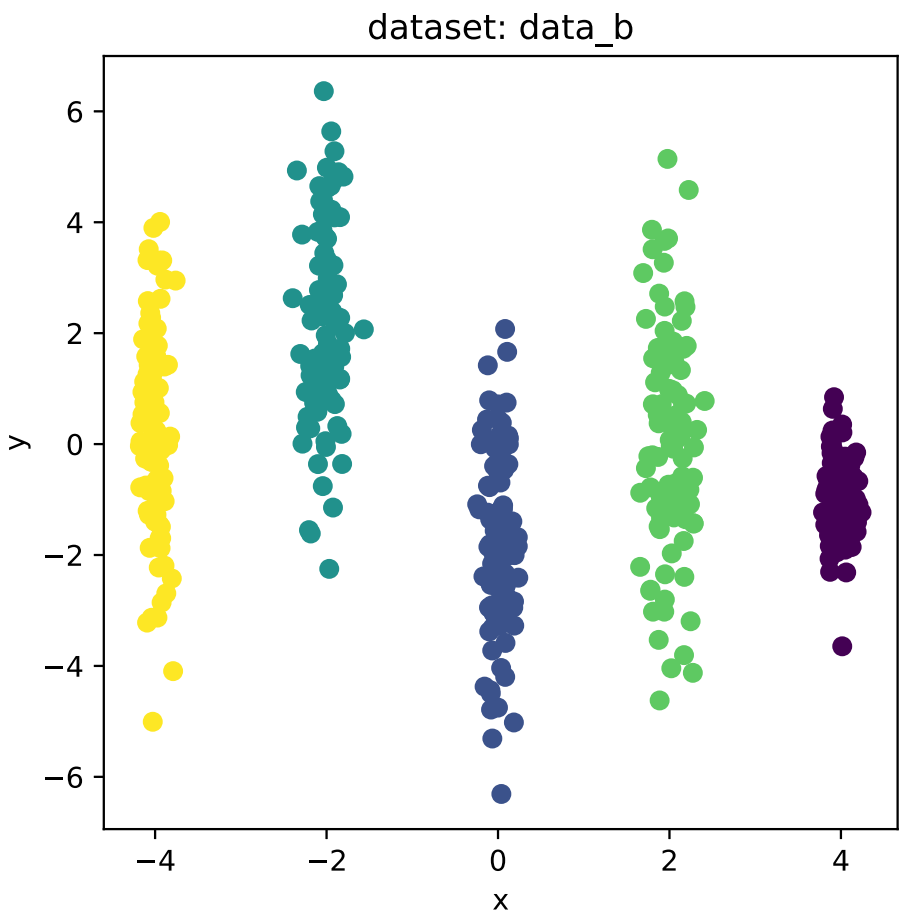
(d) Hierarchical Single Link using scikit

In [341…
```
from sklearn.cluster import AgglomerativeClustering
def Hierarchical(dataset,dataname):
    clustering = AgglomerativeClustering(n_clusters = 5, linkage = "single").fit(dataset)
    labels = clustering.labels_
    fig = plt.figure(figsize = (5,5))
    plt.scatter(dataset[:,0],dataset[:,1],c = labels)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(f"dataset: {dataname}")
```

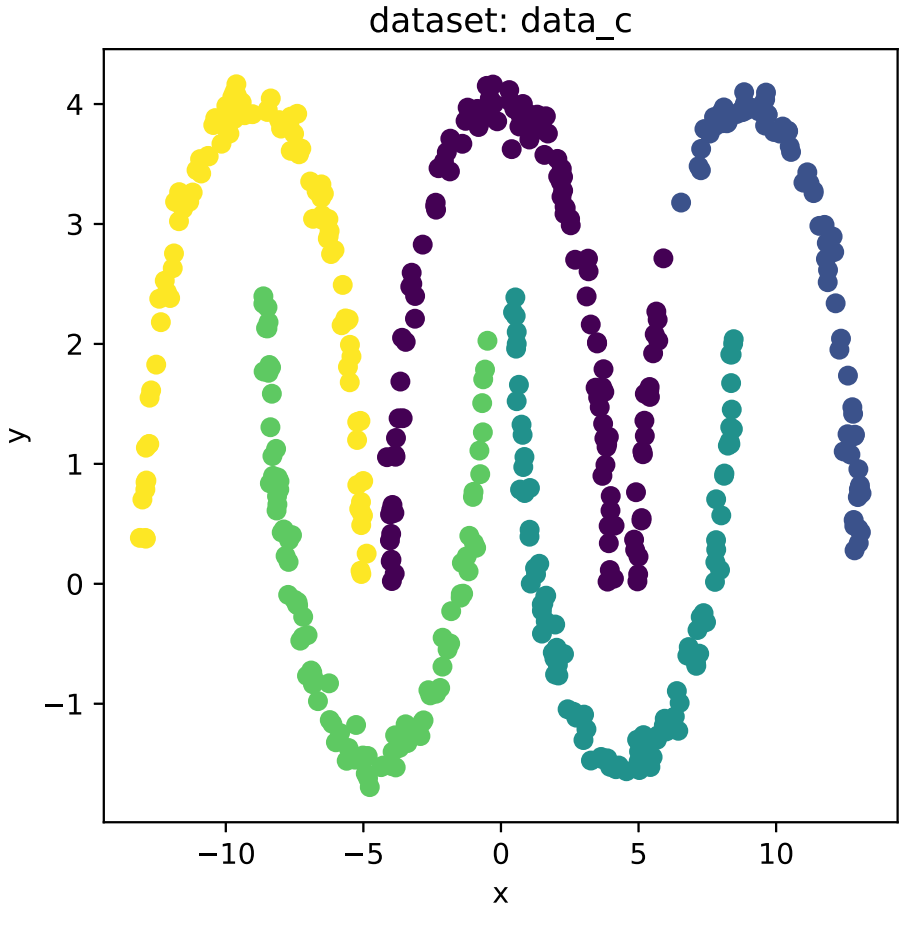
In [342…
```
# Dataset a
Hierarchical(data_a,"data_a")
```



In [305…
```
# Dataset b
Hierarchical(data_b,"data_b")
```

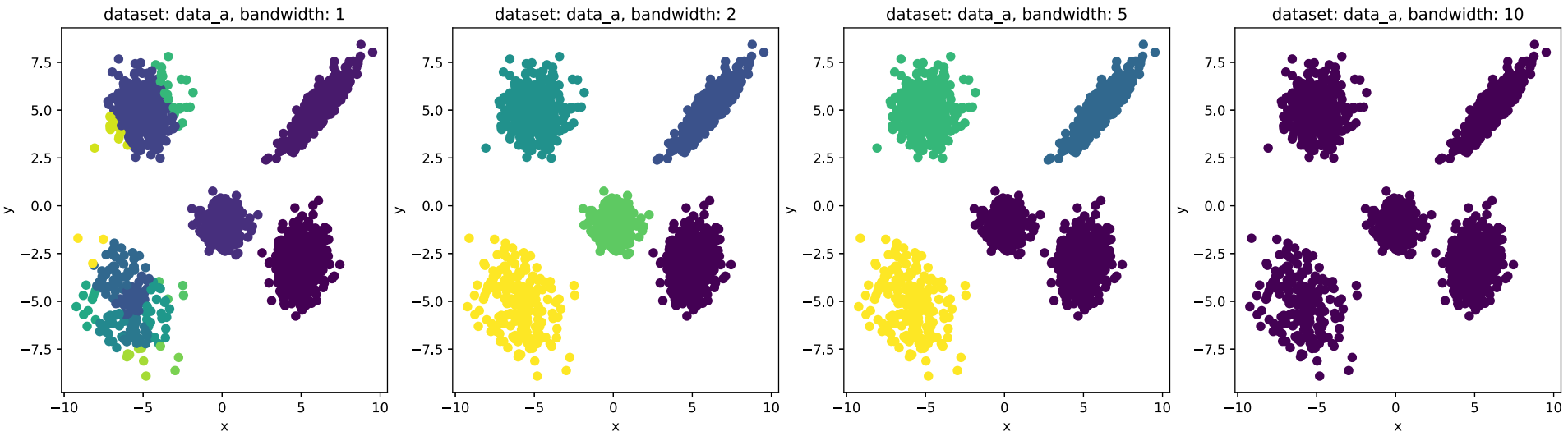


In [307…
```
# Dataset c
Hierarchical(data_c,"data_c")
```



(e) Mean-Shift Clustering using scikit

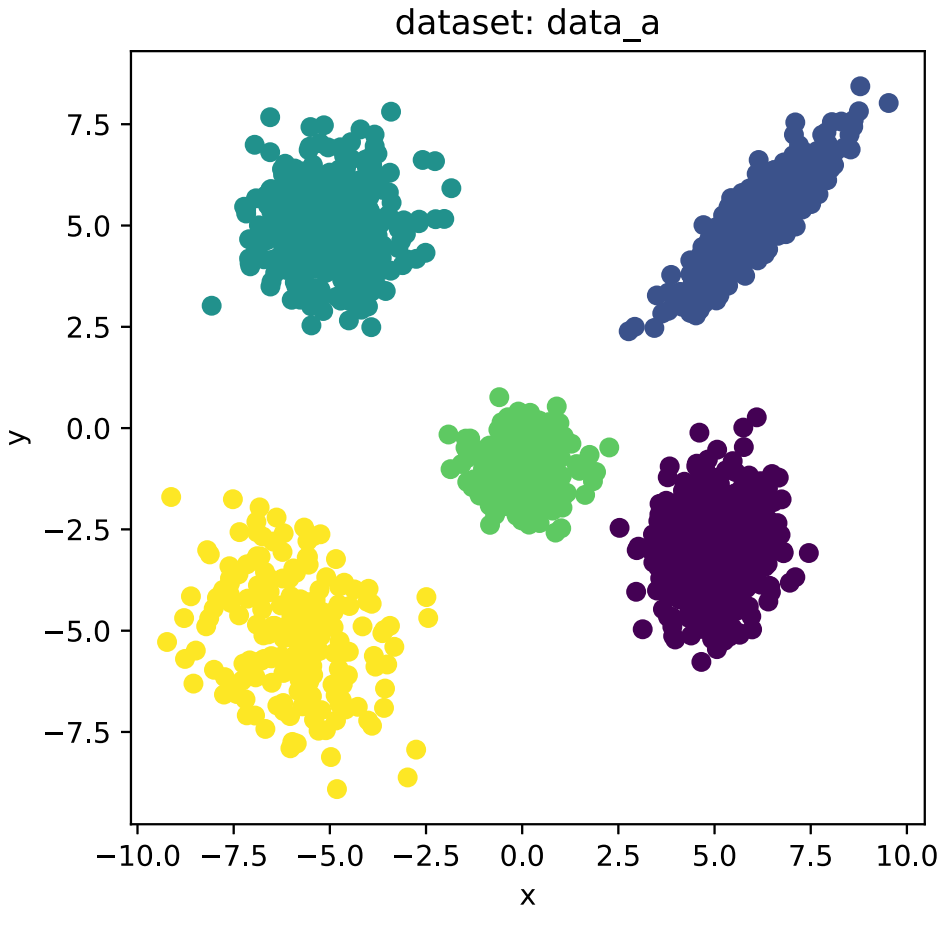
In [324…]
```
dataset = data_a
fig = plt.figure(figsize = (20,5))
bandwidth = [1,2,5,10]
for i in bandwidth:
    clustering = MeanShift(bandwidth = i).fit(dataset)
    labels = clustering.labels_
    ax = fig.add_subplot(1,4, bandwidth.index(i)+1)
    ax.scatter(dataset[:,0],dataset[:,1],c = labels)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_title(f"dataset: data_a, bandwidth: {i}")
```



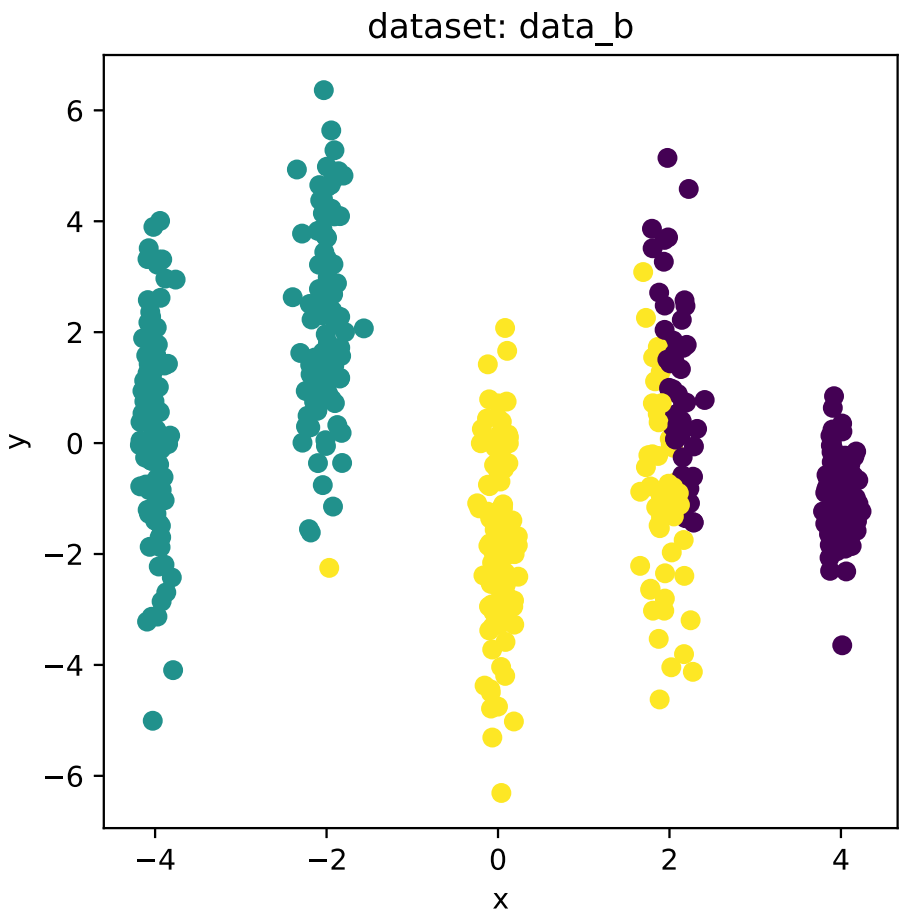
I choose bandwidth = {1,2,5,10}. I can figure out when bandwidth is small, it can not separate the dataset perfectly. Also, when bandwidth is large, some close clusters can be classify as the same cluster. Therefore, I choolse bandwidth to be 2.

In [12]:
```
from sklearn.cluster import MeanShift
def MS(dataset,dataname,bandwidth):
    clustering = MeanShift(bandwidth = bandwidth).fit(dataset)
    labels = clustering.labels_
    fig = plt.figure(figsize = (5,5))
    plt.scatter(dataset[:,0],dataset[:,1],c = labels)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(f"dataset: {dataname}")
```

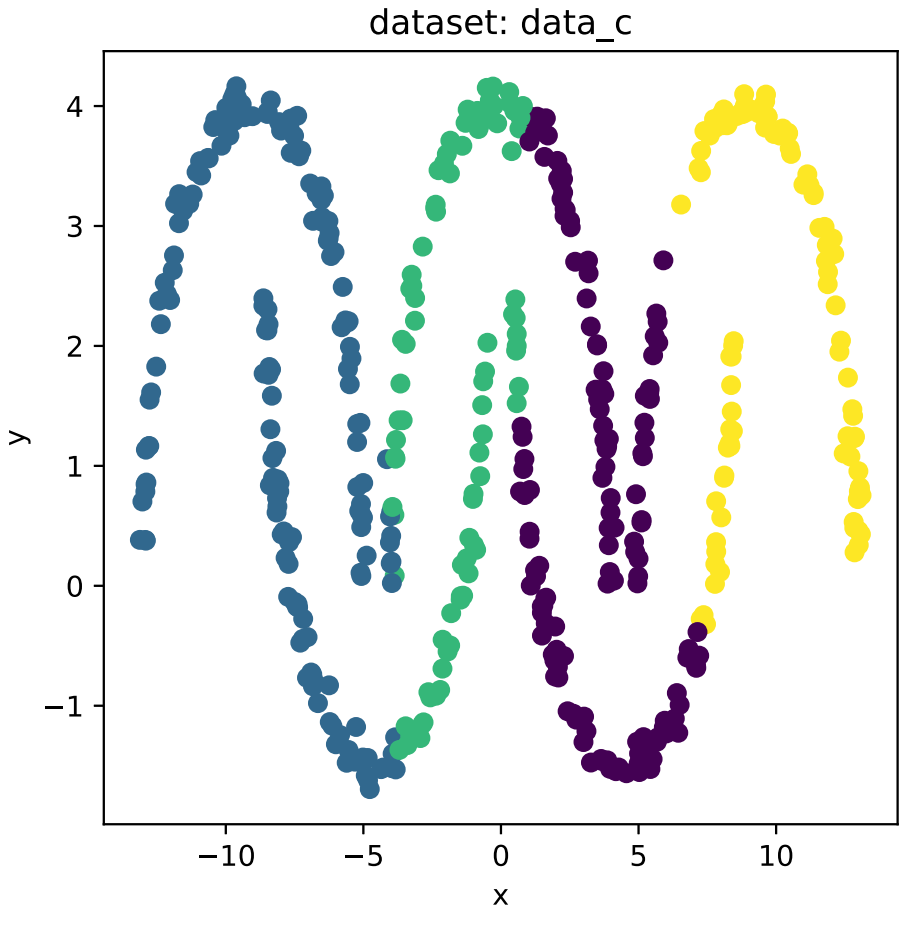
In [13]:
```
# Dataset a
MS(data_a,"data_a",2)
```



In [30]:
```
# Dataset b
MS(data_b,"data_b",2)
```



In [31]:
```
# Dataset c
MS(data_c,"data_c",3)
```



The effect of bandwidth: if bandwidth is too big, two clusters which are very close to each other will be identified as the one cluster. When bandwidth is too small, the number of clusters are big so that those points which should be one cluster are identified to be different clusters.

(f) Discussion

1)

Dataset (a): Spectral(2,5) = Kmeans = Mean Shift > Hierarchical Single Link

Dataset (b): Spectral(1,2,5) = Hierarchical Single Link > Mean Shift > Kmeans

Dataset (c): Spectral(5) = Hierarchical Single Link > Kmeans > Mean Shift

From those figures above, Spectral Clustering Methods can do a pretty good job in both three kinds of datasets. For dataset a, except Hierarchical Single link, the other three methods can separate data points perfectly. For dataset c, Spectral Clustering and Hierarchical Single link can perform well. For dataset c(Moon), Spectral Clustering and Hierarchical Single link can do pretty good although no one method can really separate data points perfectly.

2)

I think SVM can separate data_b accurately because data_b is linearly separable dataset. For the data_a and data_c, I think SVM can use 'rbf' kernel to solve these datasets, however, it cannot separate them absolute correctly and also face some overfitting problems.