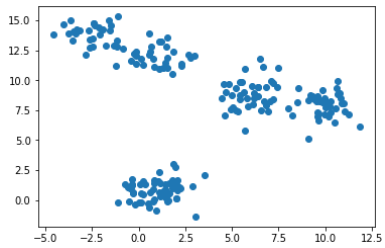
Problem A

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 23 10:36:42 2019
@author: yajunbai
"""
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import math

# problem a
# Parameters:
# X - is the input data
# ic - is the initial center
# iters - is the maximum iterations
# plot_progress - if True plots a figure of the data after each iteration
def Kmeans(X,centroids,iters=1,plot_progress=None):
    m,n=X.shape
    K=centroids.shape[0]
    previous_centroids=np.zeros((iters, centroids.shape[0], centroids.shape[1]))
    idx=np.zeros(m)
    for i in range (iters):
        previous_centroids[i,:]=centroids
        idx=find_closest_center(X,centroids)
        centroids = computecentroids(X, idx, K)
        if plot_progress:
            plotdatapts(X, idx, K, f'K={len(centroids)}', i=i+1)
    return centroids, idx
# Compute the closest center to the datapoints
def find_closest_center(X, center):
    m=X.shape[0]
    idx=np.zeros(m)
    for i in range (m):
        dist=np.square(np.sum(abs(X[i,:]-center)**2,axis=1))
        idx[i]=np.argmin(dist)
    return idx
# plots 2d datapoints
def plotdatapts(X,idx,K,t,centers=np.zeros((1,1)), i=None):
    if i == None:
        plt.title(t)
    else:
        plt.title(f'{t} interation={i}')
    color=cm.rainbow(np.linspace(0,1,K))
    plt.scatter(X[:,0],X[:,1],c=color[idx.astype(int),:])
    if centers.shape[1] > 1:
        plt.scatter(centers[:,0],centers[:,1],c='g', marker='+')
# plots 3d datapoints
def plotdatapts3d(X,idx,K,i=None):
    fig = plt.figure(figsize=(6,4))
    ax = Axes3D(fig)
    if i != None:
        plt.title(i)
    color=cm.rainbow(np.linspace(0,1,K))
    ax.scatter(X[:,0],X[:,1],X[:,2],c=color[idx.astype(int),:], marker='o')

# Compute the center by computing the mean of every cluster as a new center
def computecentroids(X,idx,K):
    m,n=X.shape
    centroids=np.zeros((K,n))
    for i in range (K):
        x=X[idx==i]
        if x.shape[0]>0:
            avg=np.mean(x,axis=0)
            centroids[i,:]=avg
    return centroids

# Compute the k-means cost for the clustering induced by given centers
def calculate_cost(X, idx, centroids):
    cost = 0
    for i in range(math.floor(idx.max())+1):
        datapoints = np.where(idx == i)
        center = centroids[i]
        for j in range(len(datapoints[0])):
            cost+=np.square(np.sum(abs(X[datapoints[0][j]] - center)**2))
    return cost
```

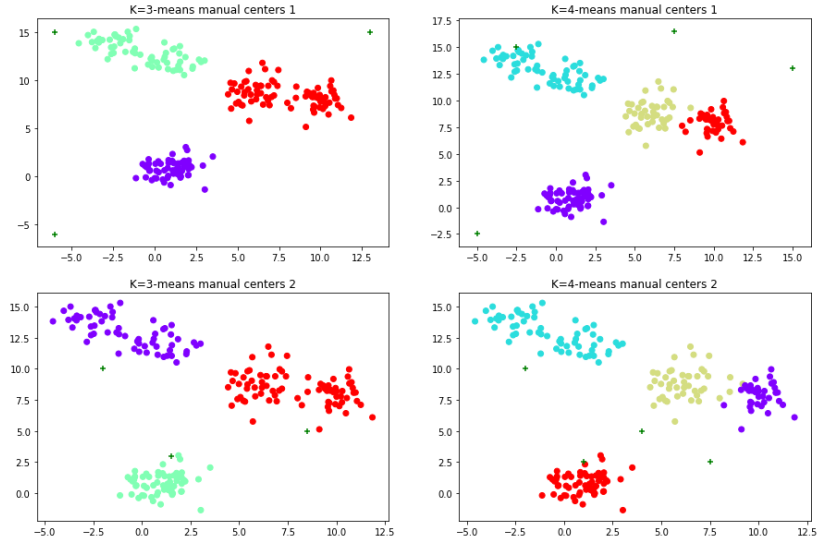
Problem B

```
# problem b, loading the data twodpoints.txt and plot the datapoints

X = np.loadtxt('twodpoints.txt', delimiter=',')
plt.scatter(X[:,0], X[:,1])
```



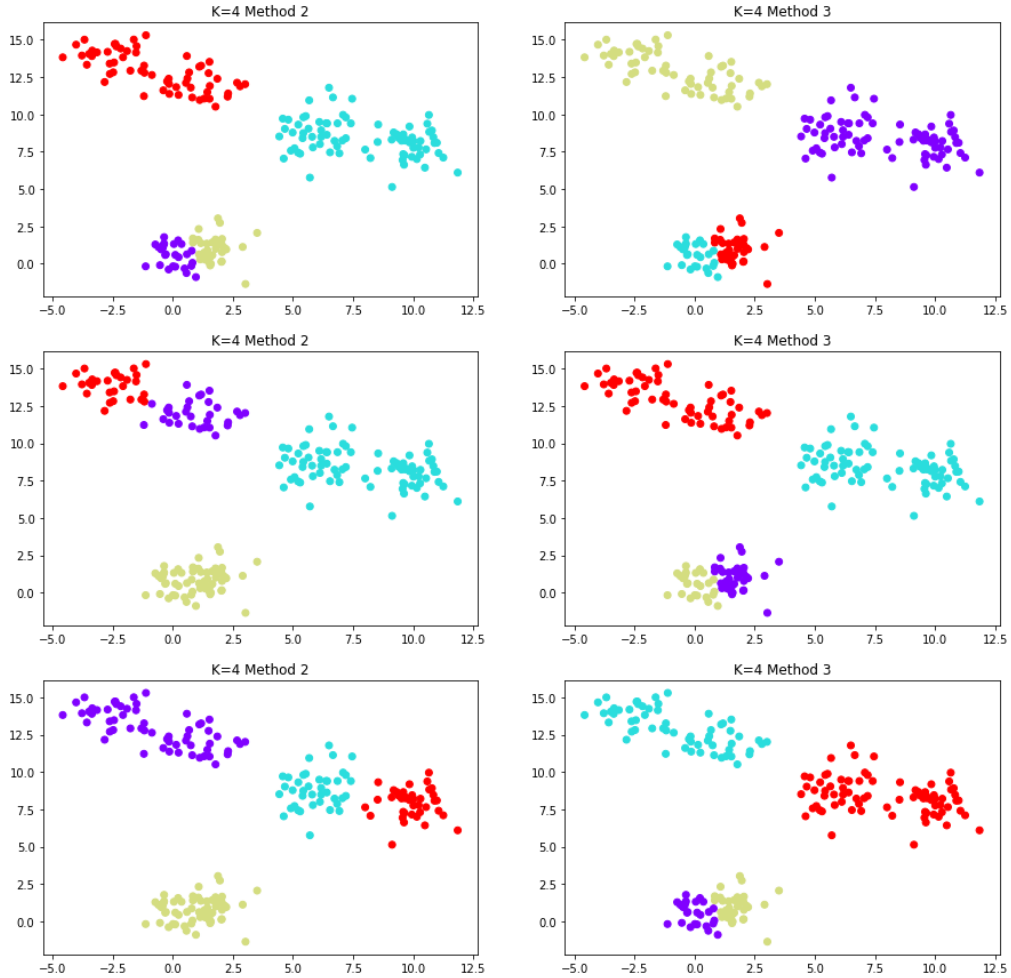
For the datapoints, 3 clusters would be suitable.
1 cluster located on the top, one in the middle and 1 at the bottom of the graph.
However, we can also get 4 clusters.
The cluster in the middle can be separated to two distinct clusters.
However, the datapoints in both clusters are not very far from each other through visual inspection.
Below are 4 plots for K-means algorithm using different manually selected centers for 3 and 4 clusters.

```
plt.figure(figsize=(15,10))
K=3
initial_centroids = np.array([[-6, -6], [-6, 15], [13, 15]])
centroids, idx = Kmeans(X, initial_centroids)
plt.subplot(2, 2, 1)
plotdatapts(X, idx, K, f"K=3-means manual centers 1",  initial_centroids)

K=4
initial_centroids = np.array([[-5, -2.5], [-2.5,15], [7.5, 16.5],[15, 13]])
centroids, idx = Kmeans(X, initial_centroids)
plt.subplot(2, 2, 2)
plotdatapts(X, idx, K, f"K=4-means manual centers 1",  initial_centroids)

K=3
initial_centroids = np.array([[-2, 10], [1.5, 3], [8.5, 5]])
centroids, idx = Kmeans(X, initial_centroids)
plt.subplot(2, 2, 3)
plotdatapts(X, idx, K, f"K=3-means manual centers 2",  initial_centroids)

K=4
initial_centroids = np.array([[7.5, 2.5], [-2,10], [4, 5],[1, 2.5]])
centroids, idx = Kmeans(X, initial_centroids)
plt.subplot(2, 2, 4)
plotdatapts(X, idx, K, f"K=4-means manual centers 2", initial_centroids)
```



Problem C

```
#Problem C
plt.figure(figsize=(15,15))
K=4
for i in range(1,7):
    if (i % 2) == 0:
        initial_centroids =  np.random.uniform(min(X[:,0]),max(X[:,1]),(K,2))
        centroids, idx = Kmeans(X, initial_centroids,20)
        plt.subplot(3, 2, i)
        plotdatapts(X, idx, K, f"K={K} Method 3")
    else:
        rand = np.random.randint(0,X.shape[0],K)
        initial_centroids = np.array([X[rand[0],:], X[rand[1],:],X[rand[2],:], X[rand[3],:]])
        centroids, idx = Kmeans(X, initial_centroids, 20)
        plt.subplot(3, 2, i)
        plotdatapts(X, idx, K, f"K={K} Method 2")
```



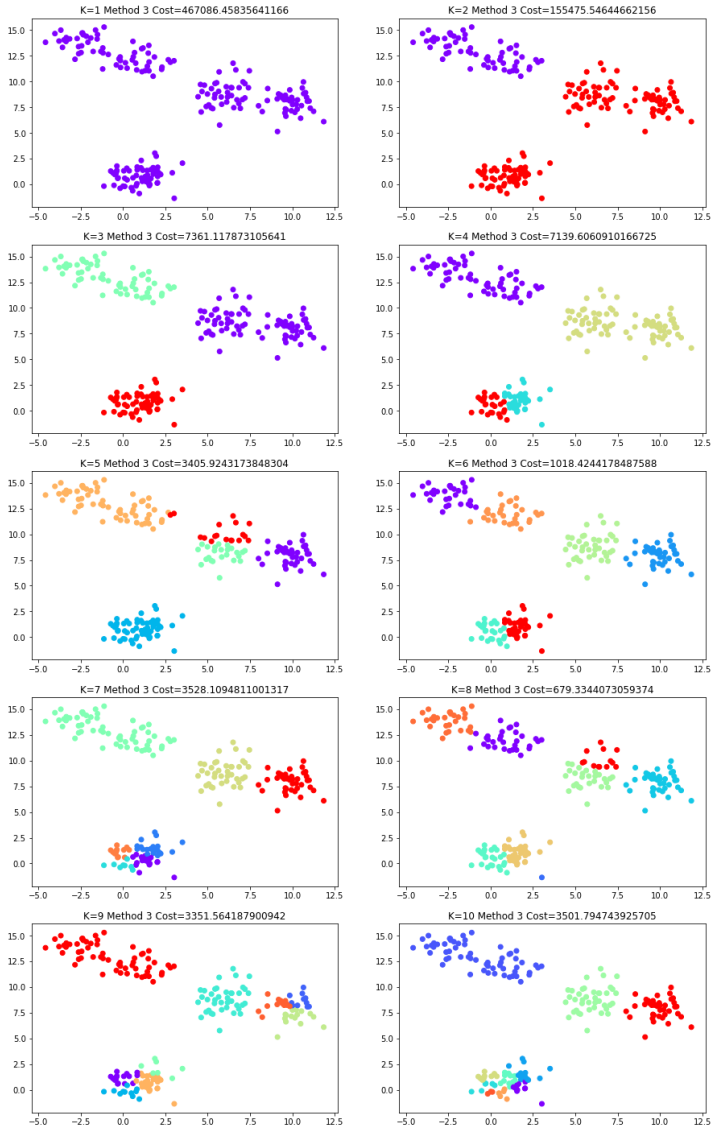
Method 2: k datapoints uniformly at random from the dataset as initial centers.

Method 3: The first center uniformly at random, and then chose each next center to be the datapoint that maximizes the sum of (euclidean) distances from the previous datapoints.

The above phenomenon from Problem B can be replicated using Method 2 and 3 of picking initial datapoints. However, in method 3, we can see the grouping is more similar after running several times.

Problem D

```
#problemD
plt.figure(figsize=(15,25))
for i in range(1,11):
    initial_centroids =  np.random.uniform(X.min(),X.max(),(i,2))
    centroids, idx = Kmeans(X, initial_centroids,20)
    plt.subplot(5, 2, i)
    plotdatapts(X, idx, i, f"K={i} Method 3 Cost={calculate_cost(X, idx, centroids)}")
```



we observed that when the number of the clusters (K means) increases, the cost decreases. Because the formula to calculate the cost of each K-means.

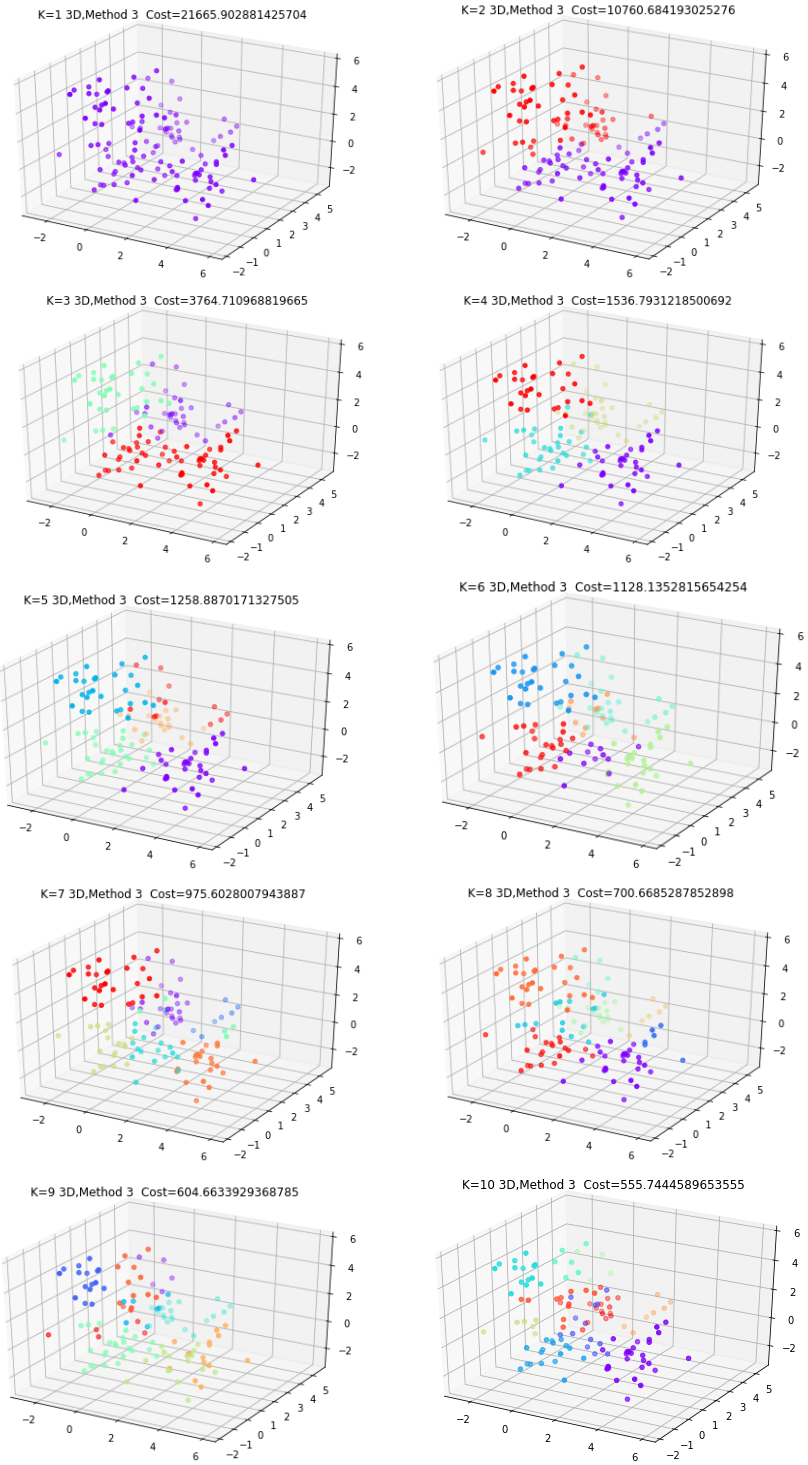
$$C(D, \mu_1, \mu_2, \ldots, \mu_k) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i(\mu_i)} |\mathbf{x}_j - \mu_j|^2$$

We calculate the cost by adding the distance difference between datapoints and the centers. When we get more clusters, we have more centers that produce less distance between the centers and the datapoints.

When we balance the number of clusters and cost to minimize the numbers and cost, a suitable number of clusters is 4 or 6.

Problem E

```
      #problemE with Method 3
X = np.loadtxt('threedpoints.txt', delimiter=',')
for i in range(1,11):
    initial_centroids =  np.random.uniform(X.min(),X.max(),(i,3))
    centroids, idx = Kmeans(X, initial_centroids,20)
    plotdatapts3d(X,idx, i, f"K={i} 3D,Method 3  Cost={calculate_cost(X, idx, centroids)}")
```



Problem E

From the pictures above, the distribution of the 3D datapoints in the dataset is quite messy. Obviously, it is not very easy to determine a suitable number of clusters from this plot since those datapoints of different color tend to mix together.

If we really need to pick one by the visualization, I will choose the number of 4. The reason is the value of cost at K=4 is smaller and all the datapoints in each cluster do not permeate into others' clusters, when k > 4, all the plots perform worth.

Finally, we can see that the phenomenon of consistency remains similar as the previous problem.

Problem F

```
#ProlemF
X = np.loadtxt('seeds_dataset.txt', delimiter='\t')
Y = X[:,X.shape[1]-1]
X = X[:,0:X.shape[1]-1]

for i in range(1,11):
    max_ = 0
    cost = 0
    for j in range(1,100):
        initial_centroids =  np.random.uniform(X.min(),X.max(),(i,7))
        centroids, idx = Kmeans(X, initial_centroids,5)
        answer = np.average(Y == (idx+1))
        if answer > max_:
            max_ = answer
            cost = calculate_cost(X, idx, centroids)
    print(f'K={i} : Accuracy={max_} Cost={cost}')
```

```
K=1 : Accuracy=0.3333333333333333 Cost=57814.26833809177
K=2 : Accuracy=0.6142857142857143 Cost=8145.230800332007
K=3 : Accuracy=0.8952380952380953 Cost=3974.0711913722535
K=4 : Accuracy=0.6666666666666666 Cost=8546.617628437558
K=5 : Accuracy=0.8047619047619048 Cost=3682.274387073998
K=6 : Accuracy=0.6523809523809524 Cost=8113.4026117707535
K=7 : Accuracy=0.6190476190476191 Cost=3224.0453328630215
K=8 : Accuracy=0.6666666666666666 Cost=9030.32153791597
K=9 : Accuracy=0.6666666666666666 Cost=8415.014842603143
K=10 : Accuracy=0.6666666666666666 Cost=8277.389969613578
```

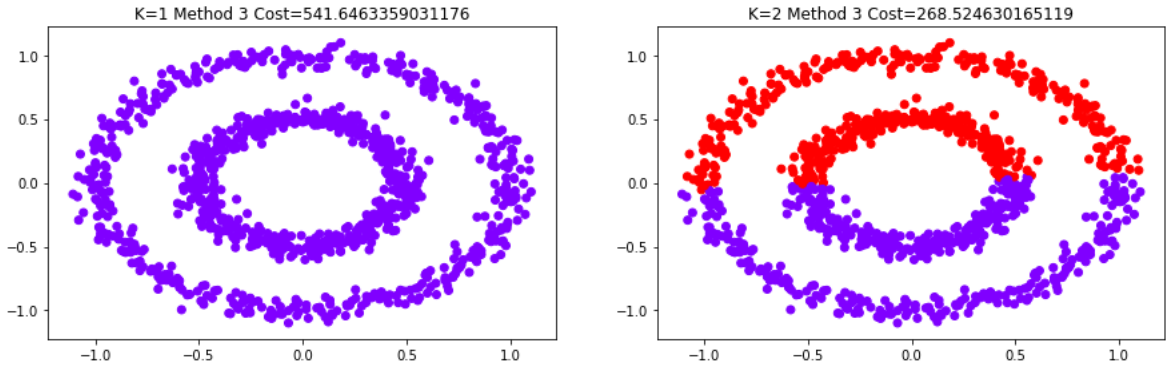
From the running result, a suitable number of clusters for this dataset will be 3.

Reason:

When K = 3, the calculated cost = 3974.0711913722535, which is almost the smallest. Also the empirical loss of this classifier = 0.1 = 1- 0.8952380952380953 which is the most accurate.
This method can be very useful in those cases when it is not easy to visualize (like 3 dimensional or more dadasets), therefore, we could classify our dataset to build a suitable 3-class classifier.

Problem G

```
#ProlemG
from sklearn.datasets import make_circles
# generate 2d classification dataset
X, y = make_circles(n_samples=1000, noise=0.05, factor=0.5)
plt.figure(figsize=(15,25))
for i in range(1,3):
    initial_centroids =  np.random.uniform(X.min(),X.max(),(i,2))
    centroids, idx = Kmeans(X, initial_centroids,20)
    plt.subplot(5, 2, i)
    plotdatapts(X, idx, i, f"K={i} Method 3 Cost={calculate_cost(X,idx, centroids)}")
```



The 2-means algorithm with the third initialization method fails to find the optimal 2-means clustering.

Reasons:
This method is trying to minimize the distance between the randomly picked centers and the datapoints. However, using this method without center can group such dataset.

Solution:
It could be fixed by using single linkage, which ensures that points that are close (or connected by a “path” of pairwise close points) will end up in the same cluster.