In this part, you are required to implement the K-means algorithm and apply your implementation on the given dataset (AllSamples.npy), which contains a set of 2-D points. You are required to implement the following strategy for choosing the initial cluster centers.

K-means-Strategy1-randomly pick the initial centers from the given samples.
Based on given code, choose the initial cluster centers at k=3 and 5.

```
3
[[ 7.52963009  8.79617112]
 [ 1.20162248  7.68639714]
 [ 7.1712312   5.16316266]]
5
[[ 2.58046907  6.53023549]
 [ 1.81229618  3.40781697]
 [ 2.0614632   8.22584366]
 [ 1.51180219  7.48293717]
 [ 7.25412082  2.77862318]]
```

without sklearn library, find final cluster centers, inertia(SSE) at k=3 and 5

```python
def difference(prev_centroid,new_centroid):
    d=0
    for i in range(len(prev_centroid)):
        #Euclidean distance
        d += np.linalg.norm(prev_centroid[i]-new_centroid[i])
    return d
```

```python
def assign_cluster(data,prev_centroid,k):
    cluster=[]
    for i in range(len(data)):
        distance=[] #distance between the centroid and data point
        for j in range(k):
            distance.append(np.linalg.norm(data[i] - prev_centroid[j]))
        index=np.argmin(distance) #return index when value is min
        cluster.append(index)
    return np.asarray(cluster)
```

```python
def new_centroid(data,cluster,k):
    centroid = []
    for i in range(k):
        array=[]
        for j in range(len(data)):
            if cluster[j]==i:
                array.append(data[j])
        centroid.append(np.mean(array,axis=0))
    return np.asarray(centroid)
```

```python
def sse(data,final_centroid,cluster,k):
    sse=0
    mean=[]
    for i in range(k):
        arr=[]
        for j in range(len(data)):
            if cluster[j]==i:
                arr.append(data[j])
        mean.append(np.mean(arr,axis=0))
    #print(mean)
    for i in range(k):
        for j in range(len(data)):
            if cluster[j]==i:
                sse += (np.linalg.norm(data[j]-mean[i]))**2
    #print(sse)
    return sse
```

```python
def k_means(data, k, i_point):

    diff = 10 #let's assume difference between the centroids is 10
    c_prev = i_point

    while diff>0.01:
        cluster = assign_cluster(data,c_prev,k) #assigns the data point to respective clusters
        #print(cluster)
        c_new = new_centroid(data,cluster,k) # to compute the new centroid point
        #print(c_new)
        diff = difference(c_prev,c_new) #to compute the difference between the centroids
        #print(diff)
        c_prev=c_new #new centroid -> centroid point

    print('Initial Cluster Centers')
    print(i_point)
    print('Final Cluster Centers')
    print(c_prev)
    s = sse(data,c_prev,cluster,k)
    print('SSE: ', s)

    return s
```

```
k_means(data, k1, i_point1)
```

```
  Initial Cluster Centers
  [[ 7.52963009  8.79617112]
   [ 1.20162248  7.68639714]
   [ 7.1712312   5.16316266]]
  Final Cluster Centers
  [[ 6.49724962  7.52297293]
   [ 2.56146449  6.08861338]
   [ 5.47740039  2.25498103]]
  SSE:  1293.77745239

1293.7774523911357
```

```
k_means(data, k2, i_point2)
```

```
  Initial Cluster Centers
  [[ 2.58046907  6.53023549]
   [ 1.81229618  3.40781697]
   [ 2.0614632   8.22584366]
   [ 1.51180219  7.48293717]
   [ 7.25412082  2.77862318]]
  Final Cluster Centers
  [[ 5.29629878  6.64908797]
   [ 3.21257461  2.49658087]
   [ 7.75648325  8.55668928]
   [ 2.51976116  7.02028909]
   [ 7.25262683  2.40015826]]
  SSE:  613.986628607

613.98662860666275
```

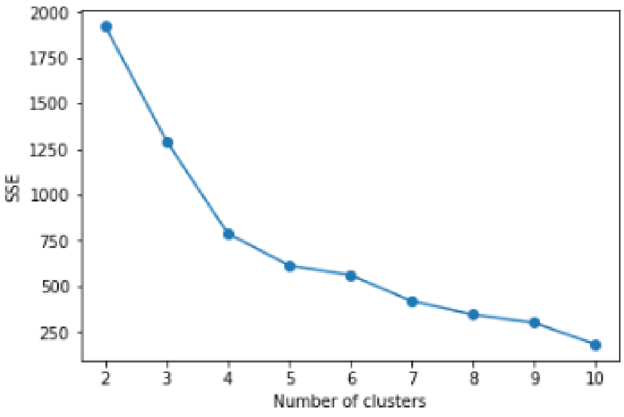
For the convenience iteration, modified Precode.py like below.

```
def initial(id, k):
    i = int(id)%150
    random.seed(i+500)
    init_idx = initial_point_idx(i,k,data.shape[0])
    init_s = init_point(data, init_idx)
    return init_s
```

Then, plotting the objective function values(SSE) vs number of clusters in range from 2 to 10

```
import matplotlib.pyplot as plt
```

```
sse_arr = []
k=range(2,11)
for i in k:
    s1 = k_means(data, i, initial('0471',i))
    #print(s1)
    sse_arr.append(s1)
plt.plot(k, sse_arr, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```



Therefore, we could select the optimum number of clusters for the k-mean clustering is 4 or 5, since the elbow is located at 4 or 5 (based above output).

K-means-Strategy2- pick the first center randomly; for the i-th center (i>1), choose a sample (among all possible samples) such that the average distance of this chosen one to all previous (i-1) centers is maximal.

Picking the first center randomly at k=4 and 6

```
4
[ 7.12751003  1.23747391]
6
[ 2.97661653  6.01021497]
```

Choose the rest of initial cluster centers measuring the average distance to maximize distance between clusters' point at k=4 and 6

```
import math
```

```
def max_centroid(kk, i_point):
    list = [i_point]

    for k in range(kk-1):
        max = 0
        new_centroid = []

        for d in data:
            dis = 0

            for l in list:
                #print(l)
                dis += math.sqrt( (d[0]-l[0])**2 + (d[1]-l[1])**2 )
                #print(l, dis)
                if np.array_equal(d,l):
                    dis = 0

            if max < dis:
                max = dis
                new_centroid = d

        list.append(new_centroid)
        #print(list)
        #print(max, add, list)

    arr=np.array(list)
   #print(arr)
    return arr
```

```
max_centroid(4, i_point1)
```

```
array([[ 7.12751003,  1.23747391],
       [ 2.95297924,  9.65073899],
       [ 9.26998864,  9.62492869],
       [ 3.85212146, -1.08715226]])
```

```
max_centroid(6, i_point2)
```

```
array([[ 2.97661653,  6.01021497],
       [ 9.26998864,  9.62492869],
       [ 3.85212146, -1.08715226],
       [ 2.95297924,  9.65073899],
       [ 6.5807212 , -0.0766824 ],
       [ 8.87578072,  8.96092361]])
```

without sklearn library, find final cluster centers, inertia(SSE) at k=4 and 6

```
def difference(prev_centroid,new_centroid):
    d=0
    for i in range(len(prev_centroid)):
        #Euclidean distance
        d += np.linalg.norm(prev_centroid[i]-new_centroid[i])
    return d

def assign_cluster(data,prev_centroid,k):
    cluster=[]
    for i in range(len(data)):
        distance=[] #distance between the centroid and data point
        for j in range(k):
            distance.append(np.linalg.norm(data[i] - prev_centroid[j]))
        index=np.argmin(distance) #return index when value is min
        cluster.append(index)
    return np.asarray(cluster)

def new_centroid(data,cluster,k):
    centroid = []
    for i in range(k):
        array=[]
        for j in range(len(data)):
            if cluster[j]==i:
                array.append(data[j])
        centroid.append(np.mean(array,axis=0))
    return np.asarray(centroid)

def sse(data,final_centroid,cluster,k):
    sse=0
    mean=[]
    for i in range(k):
        arr=[]
        for j in range(len(data)):
            if cluster[j]==i:
                arr.append(data[j])
        mean.append(np.mean(arr,axis=0))
    #print(mean)
    for i in range(k):
        for j in range(len(data)):
            if cluster[j]==i:
                sse += (np.linalg.norm(data[j]-mean[i]))**2
    #print(sse)
    return sse
```

```
def k_means(data, k, i_point):

    diff = 10 #let's assume difference between the centroids is 10
    c_prev = max_centroid(k, i_point)

    while diff>0.01:
        cluster = assign_cluster(data,c_prev,k) #assigns the data point to respective clusters
        #print(cluster)
        c_new = new_centroid(data,cluster,k) # to compute the new centroid point
        #print(c_new)
        diff = difference(c_prev,c_new) #to compute the difference between the centroids
        #print(diff)
        c_prev=c_new #new centroid -> centroid point

    print('Initial Cluster Centers')
    print(max_centroid(k, i_point))
    print('Final Cluster Centers')
    print(c_prev)
    s = sse(data,c_prev,cluster,k)
    print('SSE: ', s)

    return s
```

```
k_means(data, 4, i_point1)
```

```
Initial Cluster Centers
[[ 7.12751003  1.23747391]
 [ 2.95297924  9.65073899]
 [ 9.26998864  9.62492869]
 [ 3.85212146 -1.08715226]]
Final Cluster Centers
[[ 6.78374609  2.85019999]
 [ 3.34264769  6.92602803]
 [ 7.17928621  8.0520791 ]
 [ 2.85235149  2.28186483]]
SSE:  805.116645747

805.1166457472608
```

```
k_means(data, k2, i_point2)
```

```
Initial Cluster Centers
[[ 2.97661653  6.01021497]
 [ 9.26998864  9.62492869]
 [ 3.85212146 -1.08715226]
 [ 2.95297924  9.65073899]
 [ 6.5807212  -0.0766824 ]
 [ 8.87578072  8.96092361]]
Final Cluster Centers
[[ 3.502455    3.62870476]
 [ 7.75648325  8.55668928]
 [ 3.14506148  0.90770655]
 [ 2.52382885  7.02897469]
 [ 7.41419243  2.32169114]
 [ 5.46427736  6.83771354]]
SSE:  476.296570527

476.29657052696638
```

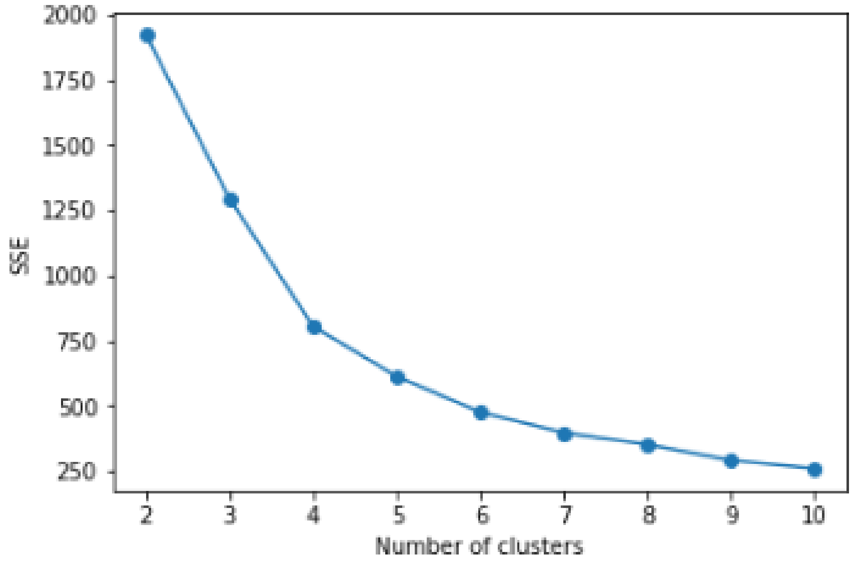
For the convenience iteration, modified Precode2.py like below.

```
def initial(id, k):
    i = int(id)%150
    random.seed(i+800)
    init_idx = initial_point_idx2(i,k,data.shape[0])
    init_s = data[init_idx,:]
    return init_s
```

Then, plotting the objective function values(SSE) vs number of clusters in range from 2 to 10

```
import matplotlib.pyplot as plt
```

```
sse_arr = []
k=range(2,11)
for i in k:
    s1 = k_means(data, i, initial('0471',i))
    #print(s1)
    sse_arr.append(s1)
plt.plot(k, sse_arr, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```



Therefore, we could select the optimum number of clusters for the k-mean clustering is 4 or 5, since the elbow is located at 4 or 5 (based above output).