```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
import warnings
warnings.simplefilter("ignore")
```

```python
image_data=pd.read_csv('/content/drive/MyDrive/image.txt', sep=",",header=None)
```

```python
image_data.head()
```

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 784 columns

```python
image_data.describe()
```

	0	1	2	3	4	5	6	7	8	9	...	774	775	776
count	10740.0	10740.0	10740.0	10740.0	10740.0	10740.0	10740.0	10740.0	10740.0	10740.0	...	10740.000000	10740.000000	10740.000000
mean	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.213222	0.089292	0.071508
std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.484832	3.260565	4.218220
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000
50%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000

```
# from sklearn.decomposition import TruncatedSVD

# svd = TruncatedSVD(n_components=11)
# image_data_svd = svd.fit_transform(image_data)
# print(svd.explained_variance_ratio_)
```

```
image_data.shape
```

```
(10740, 784)
```

```
image_data.tail()
```

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
10735	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10736	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10737	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10738	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10739	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 784 columns

```
# print(svd.explained_variance_ratio_.sum())
```

```
image_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10740 entries, 0 to 10739
Columns: 784 entries, 0 to 783
dtypes: int64(784)
memory usage: 64.2 MB
```

```
def euclidean_distance(point1, point2):
  dist = np.linalg.norm(point1 - point2)
  return dist
```

```
def manhattan_distance(point1, point2):
    return np.sum([abs(value1 - value2) for value1, value2 in zip(point1, point2)])
```

```
def cosine_similarity(point1,point2):
  return np.dot(point1, point2) / (np.linalg.norm(point1) * np.linalg.norm(point2))
```

```
def sse_k(data,cluster,centroids,K):
  sse = []
  count = [0]*K
  dist = [0]*K

  for i in range(data.shape[0]):
    value = cluster[i]
    dist[value-1]+= (euclidean_distance(data[i],centroids[value-1])**2)
    count[value-1]+=1
  # print(count[0])
  # print(count[1])
  # print(count[2])
  # print(count[3])
  # print(count[4])
  # print(count[5])
  # print(count[6])
  # print(count[7])
  # print(count[8])
```

```
  # print(count[9])

  # print(dist1)
  # print(dist2)
  # print(dist3)
  # print(dist4)
  # print(dist5)
  # print(dist6)
  # print(dist7)
  # print(dist8)
  # print(dist9)
  # print(dist10)
  #sse_1 = (dist1/count1)+(dist2/count2)+(dist3/count3)+(dist4/count4)+(dist5/count5)+(dist6/count6)+(dist7/count7)+(dist8/count8)+(dist9/co
  #sse.append(sse_1)
  sse_1 = sum(dist)
  #sse_total = np.array(sse)
  return sse_1,count
```

```
def ssb(data,count,centroids,K):
  cent_1 = [0]*len(centroids[0])
  for i in range(K):
    cent_1 += centroids[i]

  cent_1 /= K


  print(cent_1)
  value = 0
  for i in range(K):
    print((euclidean_distance(centroids[i],cent_1)**2))
    value+= count[i]*(euclidean_distance(centroids[i],cent_1)**2)
    print(value,i)
  return value
```

```
image_data_np = image_data.to_numpy()
image_np = image_data_np/255
```

```
!pip install umap-learn
import umap
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting umap-learn
  Downloading umap-learn-0.5.3.tar.gz (88 kB)
     |████████████████████████████████| 88 kB 3.4 MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from umap-learn) (1.21.6)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.8/dist-packages (from umap-learn) (1.0.2)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.8/dist-packages (from umap-learn) (1.7.3)
Requirement already satisfied: numba>=0.49 in /usr/local/lib/python3.8/dist-packages (from umap-learn) (0.56.4)
Collecting pynndescent>=0.5
  Downloading pynndescent-0.5.8.tar.gz (1.1 MB)
     |████████████████████████████████| 1.1 MB 36.4 MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from umap-learn) (4.64.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from numba>=0.49->umap-learn) (57.4.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.8/dist-packages (from numba>=0.49->umap-learn) (4.13.0)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.8/dist-packages (from numba>=0.49->umap-learn) (0.39.
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from pynndescent>=0.5->umap-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.22->umap-learn) (3.1.
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.8/dist-packages (from importlib-metadata->numba>=0.49->umap-learn) (3.
Building wheels for collected packages: umap-learn, pynndescent
  Building wheel for umap-learn (setup.py) ... done
  Created wheel for umap-learn: filename=umap_learn-0.5.3-py3-none-any.whl size=82829 sha256=0f7bf5e7a8b91d51cdc9908a168ac8d6df2a56f4b678
  Stored in directory: /root/.cache/pip/wheels/a9/3a/67/06a8950e053725912e6a8c42c4a3a241410f6487b8402542ea
  Building wheel for pynndescent (setup.py) ... done
  Created wheel for pynndescent: filename=pynndescent-0.5.8-py3-none-any.whl size=55513 sha256=7be405407d4d680bffbf72fe7cbd2bdaa19e8b061a
  Stored in directory: /root/.cache/pip/wheels/1c/63/3a/29954bca1a27ba100ed8c27973a78cb71b43dc67aed62e80c3
Successfully built umap-learn pynndescent
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.8 umap-learn-0.5.3
```

```
reducer = umap.UMAP(n_components=2)
image_data_umap = reducer.fit_transform(image_data_np)
```

```
image_data_umap
```

```
array([[ 0.9801331,  5.467487 ],
       [ 7.518575 ,  9.661499 ],
       [ 7.4222064, 10.963444 ],
```

```
       ...,
       [11.636683 ,  3.0930696],
       [ 9.7767935, -1.3638108],
       [ 0.9493922,  2.9968758]], dtype=float32)
```

```
from sklearn.manifold import TSNE

model = TSNE(n_components=2, random_state=0)
image_data_tsne = model.fit_transform(image_data_np)
```

```
image_data_tsne
```

```
array([[ 43.48313 ,  28.243101],
       [-39.58958 ,  36.908615],
       [-56.901978,  45.174625],
       ...,
       [ -9.441763, -68.224724],
       [ 63.392292, -31.949821],
       [ 53.610542,  59.242287]], dtype=float32)
```

```
import random
from collections import defaultdict
def K_Means_predict_man(data,K,max_iter,rand_seed):
  centroids = defaultdict(int)
  cluster = [0]*data.shape[0]
  random.seed(rand_seed)
  mylist = np.arange(data.shape[0])
  list1 = mylist.tolist()
  x = random.sample(list1,K)
  #print(x)
  for i in range(K):
  #initializing 1st cluster center
    num1 = x[i]
    #print(data[num1])
    centroids[i] = data[num1]

  iter=0
  #print(2)
```

```

for iteration in range(max_iter):
  iter+=1
  labels=defaultdict(list)
  #print(data.shape)
  #print(centroids)

  for keys in range(K):
    labels[keys]=[]

  for datapoint1 in range(len(data)):
    distance=[]
    for datapoint2 in range(K):
      dist=manhattan_distance(data[datapoint1],centroids[datapoint2])
      #print("Dp",data[i])
      #print("Cent",centroids[j])
      #print(dist)
      distance.append(dist)
    min_distance=min(distance)
    index=distance.index(min_distance)
    labels[index].append(data[datapoint1])
    cluster[datapoint1] = index+1
    centroid_old=dict(centroids)

  for i in range(K):
    label=labels[i]

    centroid_new=np.mean(label,axis=0)
    centroids[i]=centroid_new
    flag=1

  for i in range(K):
    a=centroids[i]
    b=centroid_old[i]
    temp = 0
    for i in range(len(a)):
      d = abs(a[i] - b[i])
      temp+=d
    if temp !=0:
```

```
        flag = 0


    if flag==1:
      break
  #print(iter)
  return labels,centroids,cluster,iter

```

```
import random
from collections import defaultdict
def K_Means_predict(data,K,max_iter,rand_seed):
  centroids = defaultdict(int)
  cluster = [0]*data.shape[0]
  random.seed(rand_seed)
  mylist = np.arange(data.shape[0])
  list1 = mylist.tolist()
  x = random.sample(list1,K)
  print("Initial Cluster Center Indices \n",x)
  print("Initial Cluster Centers\n")
  for i in range(K):
  #initializing 1st cluster center
    num1 = x[i]
    #print(data[num1])
    centroids[i] = data[num1]

  iter=0
  #print(2)

  for iteration in range(max_iter):
    iter+=1
    labels=defaultdict(list)
    #print(data.shape)
    #print(centroids)

    for keys in range(K):
      labels[keys]=[]

    for datapoint1 in range(len(data)):
```

```
      distance=[]
      for datapoint2 in range(K):
        dist=euclidean_distance(data[datapoint1],centroids[datapoint2])
        #print("Dp",data[i])
        #print("Cent",centroids[j])
        #print(dist)
        distance.append(dist)
      min_distance=min(distance)
      index=distance.index(min_distance)
      labels[index].append(data[datapoint1])
      cluster[datapoint1] = index+1
      centroid_old=dict(centroids)

    for i in range(K):
      label=labels[i]

      centroid_new=np.mean(label,axis=0)
      centroids[i]=centroid_new
      flag=1

    for i in range(K):
      a=centroids[i]
      b=centroid_old[i]
      temp = 0
      for i in range(len(a)):
        d = abs(a[i] - b[i])
        temp+=d
      if temp !=0:
        flag = 0


    if flag==1:
      break
  #print(iter)
  return labels,centroids,cluster,iter

```

```
import random
from collections import defaultdict
```

```
def K_Means_predict_man1(data,K,max_iter,rand_seed):
  centroids = defaultdict(int)
  cluster = [0]*data.shape[0]
  random.seed(rand_seed)
  mylist = np.arange(data.shape[0])
  list1 = mylist.tolist()
  x = random.sample(list1,K)
  #print(x)
  for i in range(K):
  #initializing 1st cluster center
    num1 = x[i]
    #print(data[num1])
    centroids[i] = data[num1]

  iter=0
  #print(2)

  for iteration in range(max_iter):
    iter+=1
    labels=defaultdict(list)
    #print(data.shape)
    #print(centroids)

    for keys in range(K):
      labels[keys]=[]

    for datapoint1 in range(len(data)):
      distance=[]
      for datapoint2 in range(K):
        dist=manhattan_distance(data[datapoint1],centroids[datapoint2])
        #print("Dp",data[i])
        #print("Cent",centroids[j])
        #print(dist)
        distance.append(dist)
      min_distance=min(distance)
      index=distance.index(min_distance)
      labels[index].append(data[datapoint1])
      cluster[datapoint1] = index+1
      centroid_old=dict(centroids)
```

```

    for i in range(K):
      label=labels[i]

      centroid_new=np.median(label,axis=0)
      centroids[i]=centroid_new
      flag=1

    for i in range(K):
      a=centroids[i]
      b=centroid_old[i]
      temp = 0
      for i in range(len(a)):
        d = abs(a[i] - b[i])
        temp+=d
      if temp !=0:
        flag = 0


    if flag==1:
      break
  #print(iter)
  return labels,centroids,cluster,iter

```

```
import random
from collections import defaultdict
def K_Means_predict1(data,K,max_iter,rand_seed):
  centroids = defaultdict(int)
  cluster = [0]*data.shape[0]
  random.seed(rand_seed)
  mylist = np.arange(data.shape[0])
  list1 = mylist.tolist()
  x = random.sample(list1,K)
  #print(x)
  for i in range(K):
  #initializing 1st cluster center
    num1 = x[i]
    centroids[i] = data[num1]
```

```
iter=0
#print(2)

for iteration in range(max_iter):
  iter+=1
  labels=defaultdict(list)
  #print(data.shape)
  #print(centroids)

  for keys in range(K):
    labels[keys]=[]

  for datapoint1 in range(len(data)):
    distance=[]
    for datapoint2 in range(K):
      dist=euclidean_distance(data[datapoint1],centroids[datapoint2])
      #print("Dp",data[i])
      #print("Cent",centroids[j])
      #print(dist)
      distance.append(dist)
    min_distance=min(distance)
    index=distance.index(min_distance)
    labels[index].append(data[datapoint1])
    cluster[datapoint1] = index+1
    centroid_old=dict(centroids)

  for i in range(K):
    label=labels[i]

    centroid_new=np.median(label,axis=0)
    centroids[i]=centroid_new
    flag=1

  for i in range(K):
    a=centroids[i]
    b=centroid_old[i]
    temp = 0
    for i in range(len(a)):
```

```
        d = abs(a[i] - b[i])
        temp+=d
      if temp !=0:
        flag = 0


    if flag==1:
      break
  #print(iter)
  return labels,centroids,cluster,iter

```

```
import random
from collections import defaultdict
def K_Means_predict_cos(data,K,max_iter,rand_seed):
  centroids = defaultdict(int)
  cluster = [0]*data.shape[0]
  random.seed(rand_seed)
  mylist = np.arange(data.shape[0])
  list1 = mylist.tolist()
  x = random.sample(list1,K)
  #print(x)
  for i in range(K):
  #initializing 1st cluster center
    num1 = x[i]
    centroids[i] = data[num1]

  iter=0
  #print(2)

  for iteration in range(max_iter):
    iter+=1
    labels=defaultdict(list)
    #print(data.shape)
    #print(centroids)

    for keys in range(K):
      labels[keys]=[]

```

```
  for datapoint1 in range(len(data)):
    distance=[]
    for datapoint2 in range(K):
      dist=cosine_similarity(data[datapoint1],centroids[datapoint2])
      #print("Dp",data[i])
      #print("Cent",centroids[j])
      #print(dist)
      distance.append(dist)
    min_distance=max(distance)
    index=distance.index(min_distance)
    labels[index].append(data[datapoint1])
    cluster[datapoint1] = index+1
    centroid_old=dict(centroids)

  for i in range(K):
    label=labels[i]

    centroid_new=np.mean(label,axis=0)
    #print(centroid_new)
    centroids[i]=centroid_new
    flag=1

  for i in range(K):
    a=centroids[i]
    b=centroid_old[i]
    temp = 0
    #print(a)
    #print(b)
    for i in range(len(a)):
      d = abs(a[i] - b[i])
      temp+=d
    if temp !=0:
      flag = 0


  if flag==1:
    break
#print(iter)
```

```
  return labels,centroids,cluster,iter
```

```
import random
from collections import defaultdict
def K_Means_predict_cos1(data,K,max_iter,rand_seed):
  centroids = defaultdict(int)
  cluster = [0]*data.shape[0]
  random.seed(rand_seed)
  mylist = np.arange(data.shape[0])
  list1 = mylist.tolist()
  x = random.sample(list1,K)
  #print(x)
  for i in range(K):
  #initializing 1st cluster center
    num1 = x[i]
    centroids[i] = data[num1]

  iter=0
  #print(2)

  for iteration in range(max_iter):
    iter+=1
    labels=defaultdict(list)
    #print(data.shape)
    #print(centroids)

    for keys in range(K):
      labels[keys]=[]

    for datapoint1 in range(len(data)):
      distance=[]
      for datapoint2 in range(K):
        dist=cosine_similarity(data[datapoint1],centroids[datapoint2])
        #print("Dp",data[i])
        #print("Cent",centroids[j])
        #print(dist)
        distance.append(dist)
      min_distance=max(distance)
      index=distance.index(min_distance)
```

```
      labels[index].append(data[datapoint1])
      cluster[datapoint1] = index+1
      centroid_old=dict(centroids)

    for i in range(K):
      label=labels[i]

      centroid_new=np.median(label,axis=0)
      centroids[i]=centroid_new
      flag=1

    for i in range(K):
      a=centroids[i]
      b=centroid_old[i]
      temp = 0
      for i in range(len(a)):
        d = abs(a[i] - b[i])
        temp+=d
      if temp !=0:
        flag = 0


    if flag==1:
      break
  #print(iter)
  return labels,centroids,cluster,iter

```

```
# from sklearn.decomposition import PCA

# pca = PCA(n_components=0.95)
# image_data_pca = pca.fit_transform(image_data)
```

```
sse_2 = []
value1 = 20
iterations = np.arange(value1)
iterations1 = np.arange(2,21,2)
from collections import defaultdict
```

```
for K in range(2,21,2):
  sse_1 = []
  for rand_seed in range(value1):
    classes,centroids,cluster,iter=K_Means_predict(image_data_tsne,K,10000,rand_seed)
  # classes1,centroids1,cluster1,iter1=K_Means_predict1(image_data_tsne,10,10000,rand_seed)
  # classes2,centroids2,cluster2,iter2=K_Means_predict_man(image_data_tsne,10,10000,rand_seed)
  # classes3,centroids3,cluster3,iter3=K_Means_predict_man1(image_data_tsne,10,10000,rand_seed)
  # classes4,centroids4,cluster4,iter4=K_Means_predict_cos(image_data_tsne,10,10000,rand_seed)
  # classes5,centroids5,cluster5,iter5=K_Means_predict_cos1(image_data_tsne,10,10000,rand_seed)
    for i in range(0,10):
      classes[i]=np.array(classes[i]).tolist()
    # for i in range(0,10):
    #   classes1[i]=np.array(classes1[i]).tolist()
    print("Iteration=%d \n"%rand_seed)
    print("Euclidean")
    print("Mean \n")
    print("Max Iteratins Run=%d \n"%iter)
    print("Final Centroids:",centroids)
    print("\n")
    SSE,count = sse_k(image_data_tsne,cluster,centroids,K)
    sse_1.append(SSE)
  print(sse_1)
  x = plt.subplot( )
  x.plot(iterations, sse_1, label='SSE')
  #x.plot(k_1, cv_auc, label='AUC CV')
  plt.title('Runs vs SSE')
  plt.xlabel('Random Center')
  plt.ylabel('SSE')
  x.legend()
  plt.show()
  sse_2.append(min(sse_1))

x = plt.subplot( )
x.plot(iterations1, sse_2, label='SSE')
#x.plot(k_1, cv_auc, label='AUC CV')
plt.title('K vs SSE')
plt.xlabel('K')
plt.ylabel('SSE')
x.legend()
```

```
plt.show()


  # for i in range(0,10):
  #  print("Cluster %d"%i,len(classes[i]))
  # print("Median \n")
  # print("Max Iteratins Run=%d \n"%iter1)
  # print("Final Centroids:",centroids1)
  # print("\n")
  # for i in range(0,10):
  #   print("Cluster %d"%i,len(classes1[i]))
  # print("\n")
  # print("Manhattan")
  # print("Mean \n")
  # print("Max Iteratins Run=%d \n"%iter2)
  # print("Final Centroids:",centroids2)
  # print("\n")
  # for i in range(0,10):
  #   print("Cluster %d"%i,len(classes2[i]))
  # print("Median \n")
  # print("Max Iteratins Run=%d \n"%iter3)
  # print("Final Centroids:",centroids3)
  # print("\n")
  # for i in range(0,10):
  #   print("Cluster %d"%i,len(classes3[i]))
  # print("\n")
  # print("Cosine")
  # print("Mean \n")
  # print("Max Iteratins Run=%d \n"%iter4)
  # print("Final Centroids:",centroids4)
  # print("\n")
  # for i in range(0,10):
  #   print("Cluster %d"%i,len(classes4[i]))
  # print("Median \n")
  # print("Max Iteratins Run=%d \n"%iter5)
  # #print("Final Centroids:",centroids5)
  # print("\n")
  # for i in range(0,10):
  #   print("Cluster %d"%i,len(classes5[i]))
```

```
#print("\n")
#print("**********XXXXX********")
#print(centroids)
```

```
Final Centroids: defaultdict(<class 'int'>, {0: array([-60.330273, -16.44391 ], dtype=float32), 1: a


Initial Cluster Center Indices
 [7775, 4407, 8669, 5730, 2336, 6252, 177, 6139, 7905, 4490]
Initial Cluster Centers

Iteration=12

Euclidean
Mean

Max Iteratins Run=53

Final Centroids: defaultdict(<class 'int'>, {0: array([ 43.39277 , -38.341625], dtype=float32), 1: a


Initial Cluster Center Indices
 [4243, 4763, 3042, 10682, 3777, 2411, 3689, 10501, 3069, 2133]
Initial Cluster Centers

Iteration=13

Euclidean
Mean

Max Iteratins Run=43

Final Centroids: defaultdict(<class 'int'>, {0: array([-53.351593,  14.918434], dtype=float32), 1: a


Initial Cluster Center Indices
 [1750, 10090, 10683, 8636, 4045, 4441, 4190, 4768, 1189, 7368]
Initial Cluster Centers

Iteration=14

Euclidean
Mean

Max Iteratins Run=39

Final Centroids: defaultdict(<class 'int'>, {0: array([ 43.400204, -38.321705], dtype=float32), 1: a
```

```
Initial Cluster Center Indices
 [3423, 190, 8541, 592, 2588, 3914, 276, 900, 2412, 6017]
Initial Cluster Centers

Iteration=15

Euclidean
Mean

Max Iteratins Run=34

Final Centroids: defaultdict(<class 'int'>, {0: array([-2.0399377, 69.93747  ], dtype=float32), 1: a


Initial Cluster Center Indices
 [5923, 7687, 7872, 4668, 6831, 3713, 7319, 95, 6709, 4241]
Initial Cluster Centers

Iteration=16

Euclidean
Mean

Max Iteratins Run=27

Final Centroids: defaultdict(<class 'int'>, {0: array([-21.026497 ,  -7.0507927], dtype=float32), 1:
```