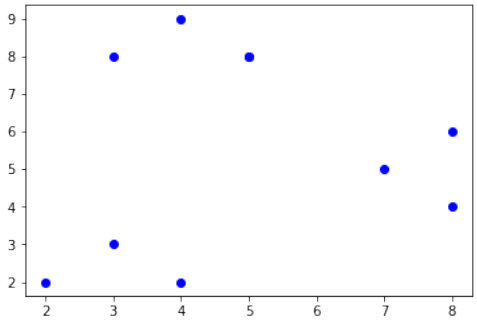
```
import pandas as pd
import numpy as np
import random
import math
import matplotlib.pyplot as plt
```

Question 1

```
input_1 = np.array([5,8,7,8,3,4,2,3,4,5]).reshape(-1,1)
input_2 = np.array([8,6,5,4,3,2,2,8,9,8]).reshape(-1,1)
X = np.concatenate([input_1,input_2],axis = 1)
```

```
plt.scatter(X[:,0],X[:,1],c = 'b')
plt.show()
```



random assigning clusters:

```
random.seed(2020)
clusters = []
for node in X:
    clusters.append(random.sample([0,1,2],1)[0])
```

```
clusters
```

```
[2, 2, 0, 2, 1, 1, 1, 1, 1, 2]
```

calculate the centorids and update the clusters:

```
centorids = []
for cluster in range(3):
    centorids.append(np.mean(X[np.array(clusters) == cluster],axis = 0))
```

```
centorids
```

```
[array([ 7.,  5.]), array([ 3.2,  4.8]), array([ 6.5,  6.5])]
```

```
clusters = []
for node in X:
    distance = np.sum((node - np.array(centorids))**2, axis = 1)
    clusters.append(np.argsort(distance)[0])
```

```
clusters
```

```
[2, 0, 0, 0, 1, 1, 1, 1, 2, 2]
```

calculate the clusters and represent through graph

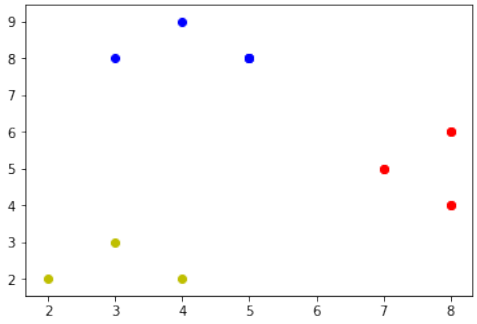
```
def iterate_the_cluster(old_clusters, X, K):
    centorids = []
    for cluster in range(K):
        centorids.append(np.mean(X[np.array(old_clusters) == cluster],axis = 0))
    clusters = []
    for node in X:
        distance = np.sum((node - np.array(centorids))**2, axis = 1)
        clusters.append(np.argsort(distance)[0])
    if clusters == old_clusters:
        return clusters
    clusters = iterate_the_cluster(clusters, X, K)
    return clusters
```

```
final_clusters = iterate_the_cluster(clusters, X, 3)
```

```
final_clusters
```

```
[2, 0, 0, 0, 1, 1, 1, 2, 2, 2]
```

```
colors = ['r','y','b']
count = 0
for node in X:
    plt.scatter(node[0],node[1],c = colors[final_clusters[count]])
    count += 1
plt.show()
```



when k =2:

```
random.seed(2020)
clusters = []
for node in X:
    clusters.append(random.sample([0,1],1)[0])

### calculate the centorids and update the clusters:

centorids = []
for cluster in range(2):
    centorids.append(np.mean(X[np.array(clusters) == cluster],axis = 0))

clusters = []
for node in X:
    distance = np.sum((node - np.array(centorids))**2, axis = 1)
    clusters.append(np.argsort(distance)[0])
```

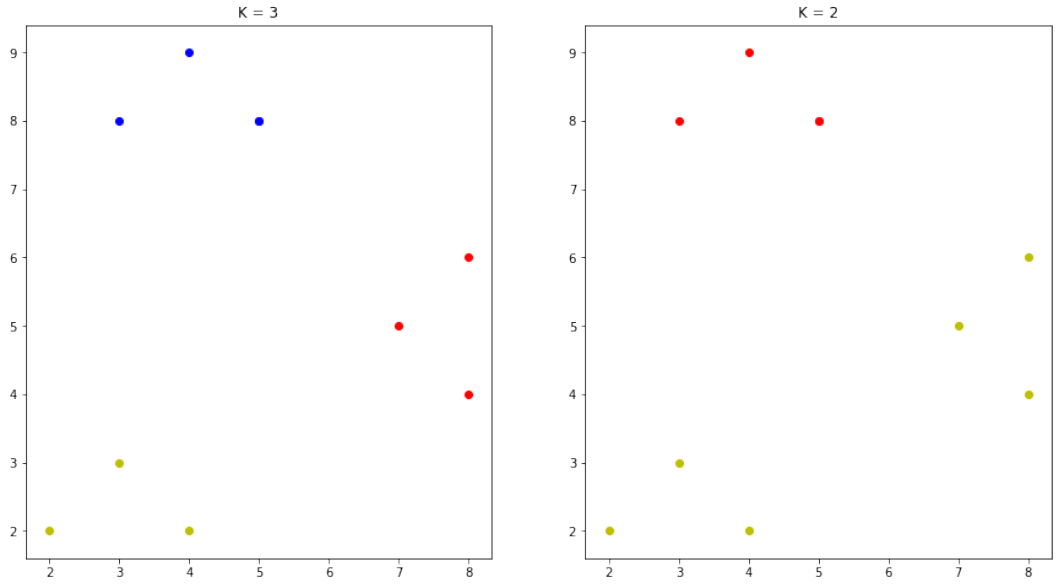
```
final_clusters_2 = iterate_the_cluster(clusters, X, 2)
```

```
final_clusters_2
```

```
[0, 1, 1, 1, 1, 1, 1, 0, 0, 0]
```

```
colors = ['r','y','b']
count = 0
plt.figure(figsize=(15,8))
for node in X:
    plt.subplot(1,2,1)
    plt.scatter(node[0],node[1],c = colors[final_clusters[count]])
    count += 1
    plt.title('K = 3')

count = 0
for node in X:
    plt.subplot(1,2,2)
    plt.scatter(node[0],node[1],c = colors[final_clusters_2[count]])
    count += 1
    plt.title('K = 2')

plt.show()
```

```
/Users/apple/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:106:
MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses
the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile,
this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes
instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



comments on number of centroids:

when k =3, the result pans out, and we can see k means finds 3 clusters at bottom, right and on the up of the space. When k = 2, the result is not good, cause the k is too small the split the obvious difference within clusters.

Question 2

```
from sklearn.decomposition import PCA
from sklearn import preprocessing
```

```
df_wiki = pd.read_csv('wiki.csv')
```

```
df_wiki.describe()
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

	age	gender	phd	yearsexp	userwiki	pu1	pu2	pu3	peu1	
count	800.00000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.00000	800.0
mean	42.16625	0.427500	0.433750	10.408750	0.136250	3.125000	3.136250	3.427500	4.31750	4.022
std	7.54767	0.495025	0.495902	6.756755	0.343268	1.004059	0.980504	1.052915	0.79843	0.835
min	23.00000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.00000	1.000
25%	36.00000	0.000000	0.000000	5.000000	0.000000	2.000000	2.000000	3.000000	4.00000	4.000
50%	42.00000	0.000000	0.000000	10.000000	0.000000	3.000000	3.000000	3.000000	4.00000	4.000
75%	47.00000	1.000000	1.000000	15.000000	0.000000	4.000000	4.000000	4.000000	5.00000	5.000
max	69.00000	1.000000	1.000000	36.000000	1.000000	5.000000	5.000000	5.000000	5.00000	5.000

8 rows × 57 columns

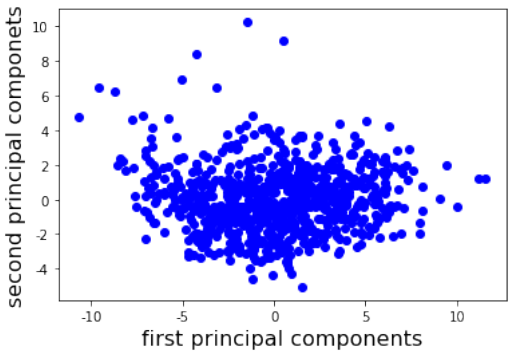
```
np.array(df_wiki).shape
```

```
(800, 57)
```

```
X_wiki = preprocessing.scale(np.array(df_wiki))
pca = PCA(n_components=X_wiki.shape[1])
pca.fit(X_wiki)
```

```
PCA(copy=True, iterated_power='auto', n_components=57, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
first_principle = np.dot(pca.components_[0].reshape(-1,1).T,X_wiki.T)
second_principle = np.dot(pca.components_[1].reshape(-1,1).T,X_wiki.T)

plt.scatter(first_principle, second_principle ,c = 'b')
plt.xlabel('first principal components',size = 16)
plt.ylabel('second principal componets', size = 16)
plt.show()
```




```
first_principle.shape
```

```
(1, 800)
```

```
features = df_wiki.columns
print(f'the most correlated variables on the first principal component: \n',\
      np.array(features)[np.argsort(pca.components_[0])[:-6:-1]])
print('\n')
print('--------------------')
print('\n')
print(f'the most correlated variables on the second principal component: \n',\
      np.array(features)[np.argsort(pca.components_[1])[:-6:-1]])
```

```
the most correlated variables on the first principal component:
 ['bi2' 'bi1' 'use3' 'use4' 'pu3']
```

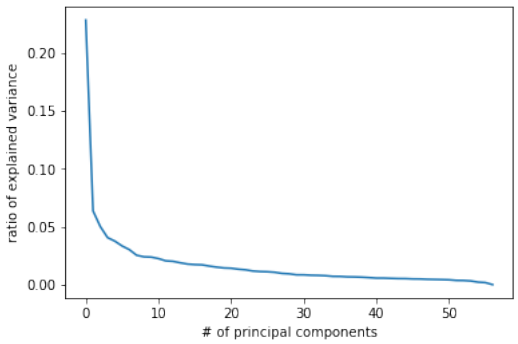
```
--------------------
```

```
the most correlated variables on the second principal component:
 ['exp4' 'use2' 'use1' 'vis3' 'domain_Engineering_Architecture']
```

```
print(pca.explained_variance_)
```

```
[  1.30183308e+01   3.63685656e+00   2.86709679e+00   2.32442160e+00
   2.15029029e+00   1.91308411e+00   1.73105232e+00   1.45656189e+00
   1.37965789e+00   1.36544021e+00   1.29303701e+00   1.18205213e+00
   1.15740466e+00   1.08625584e+00   1.02299955e+00   9.97407286e-01
   9.85241541e-01   9.24117235e-01   8.72313125e-01   8.31980895e-01
   8.17852381e-01   7.70301586e-01   7.39687973e-01   6.80616246e-01
   6.54538459e-01   6.44510340e-01   6.19534595e-01   5.63948581e-01
   5.43244223e-01   4.94382625e-01   4.92812308e-01   4.73622696e-01
   4.65744923e-01   4.50596299e-01   4.18530454e-01   4.15067118e-01
   3.94751269e-01   3.89017656e-01   3.77056738e-01   3.56682218e-01
   3.32388856e-01   3.31600540e-01   3.19617064e-01   3.09662557e-01
   3.07556544e-01   2.92249629e-01   2.88743219e-01   2.73961679e-01
   2.66030236e-01   2.58547163e-01   2.48620304e-01   2.19337729e-01
   2.14636568e-01   1.93057276e-01   1.34233864e-01   1.12268848e-01
   1.07267393e-02]
```

```
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('# of principal components')
plt.ylabel('ratio of explained variance')
plt.show()
```



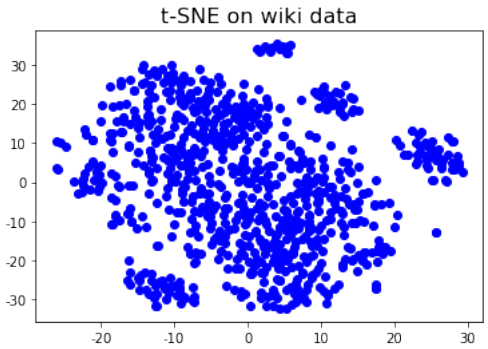
approximately, 16.65 variance are explained by the first and second principal components

```
print(pca.explained_variance_[0]+pca.explained_variance_[1])
```

```
16.6551873108
```

```
from sklearn.manifold import TSNE
X_embedded = TSNE(n_components=2).fit_transform(X_wiki)
```

```
plt.scatter(X_embedded[:,0],X_embedded[:,1],c = 'b')
plt.title('t-SNE on wiki data',size = 16)
plt.show()
```



t-SNE comments:

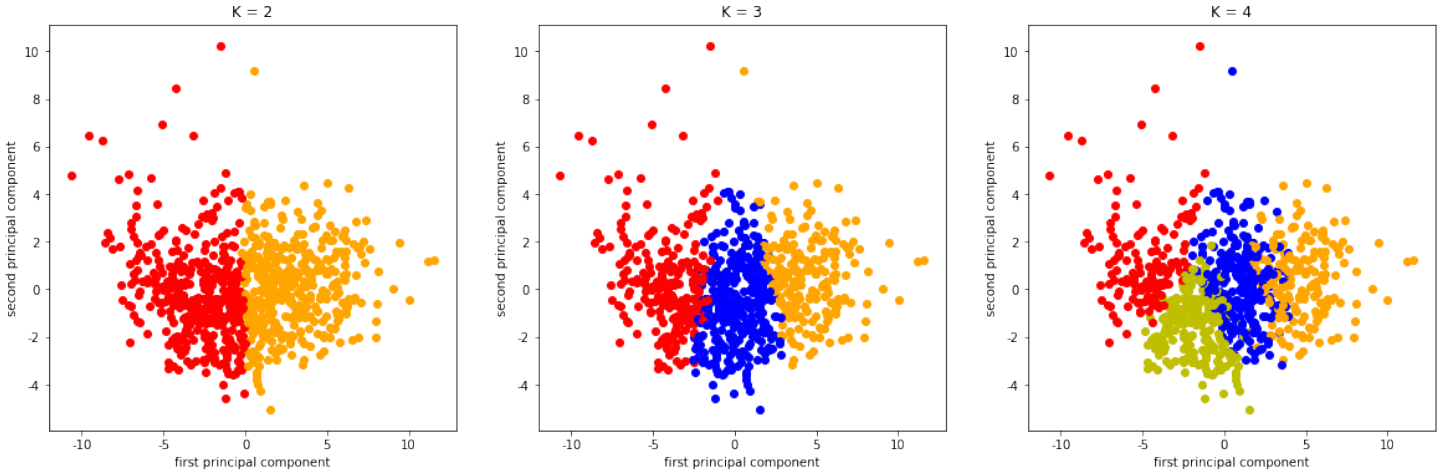
t-SNE seems to form clusters within the dataset,and the there is no difference on interpretation between the first dimension and the second dimension.

Question 3

```
first_principle = first_principle.reshape(800,1)
second_principle = second_principle.reshape(800,1)
```

```
colors = ['r','orange','b','y']
from sklearn.cluster import KMeans
kmeans_2 = KMeans(n_clusters=2, random_state=0).fit(X_wiki)
kmeans_3 = KMeans(n_clusters=3, random_state=0).fit(X_wiki)
kmeans_4 = KMeans(n_clusters=4, random_state=0).fit(X_wiki)
plt.figure(figsize=(20,6))
for i in range(len(X_wiki)):
    plt.subplot(1,3,1)
    plt.scatter(first_principle[i],second_principle[i],c = colors[kmeans_2.labels_[i]])
    plt.title('K = 2')
    plt.xlabel('first principal component')
    plt.ylabel('second principal component')

for i in range(len(X_wiki)):
    plt.subplot(1,3,2)
    plt.scatter(first_principle[i],second_principle[i],c = colors[kmeans_3.labels_[i]])
    plt.title('K = 3')
    plt.xlabel('first principal component')
    plt.ylabel('second principal component')

for i in range(len(X_wiki)):
    plt.subplot(1,3,3)
    plt.scatter(first_principle[i],second_principle[i],c = colors[kmeans_4.labels_[i]])
    plt.title('K = 4')
    plt.xlabel('first principal component')
    plt.ylabel('second principal component')
plt.show()
```

```
/Users/apple/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:106: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



comments:

We conducted kmeans on high dimension and we can see that first and second principal components can still represent the high dimension clusters, although there high dimension clustering is not exactly the same as that in low dimension(we can see overlapping when K>=3).

```
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.metrics import pairwise_distances
```

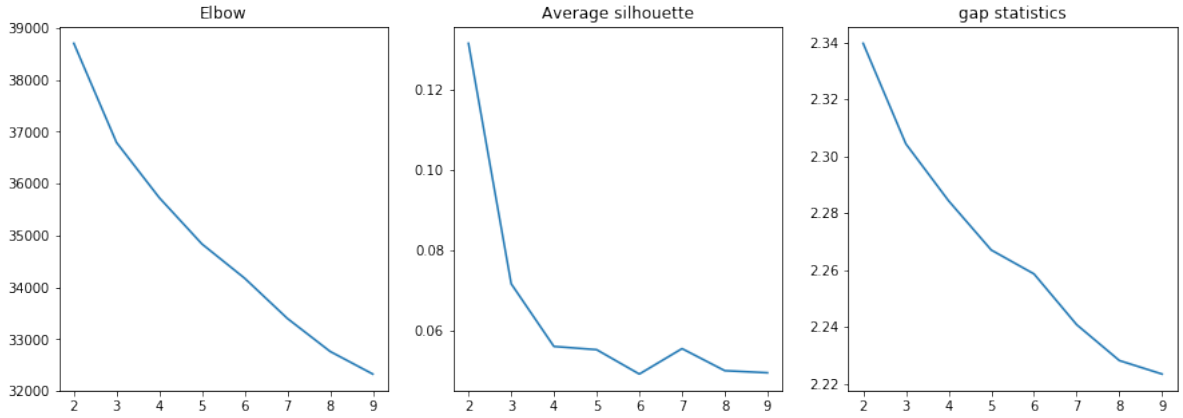
```
def compute_gap(clustering, data, k):
    if len(data.shape) == 1:
        data = data.reshape(-1, 1)
    reference = np.random.rand(*data.shape)
    clustering.n_clusters = k
    assignments = clustering.fit_predict(reference)
    reference_inertia = clustering.inertia_

    clustering.n_clusters = k
    assignments = clustering.fit_predict(data)
    ondata_inertia = clustering.inertia_

    gap = np.log(ondata_inertia) - np.log(reference_inertia)
    return gap
```

```
list(range(2,100,10))
```

```
[2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
```

```
## ELBOW, average sihouette, gap statistics:
distortion = []
silhouette_score_avg = []
gap_statistics = []
for k in range(2,10):
    kmean = KMeans(n_clusters=k, random_state=0).fit(X_wiki)
    distortion.append(kmean.inertia_)
    silhouette_score_avg.append(silhouette_score(X_wiki, kmean.labels_))
    gap_statistics.append(compute_gap(KMeans(), X_wiki, k))
```

```
x = range(2,10)
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
plt.plot(x,distortion)
plt.title('Elbow')
plt.subplot(1,3,2)
plt.plot(x,silhouette_score_avg)
plt.title('Average silhouette')
plt.subplot(1,3,3)
plt.plot(x,gap_statistics)
plt.title('gap statistics')
plt.show()
```

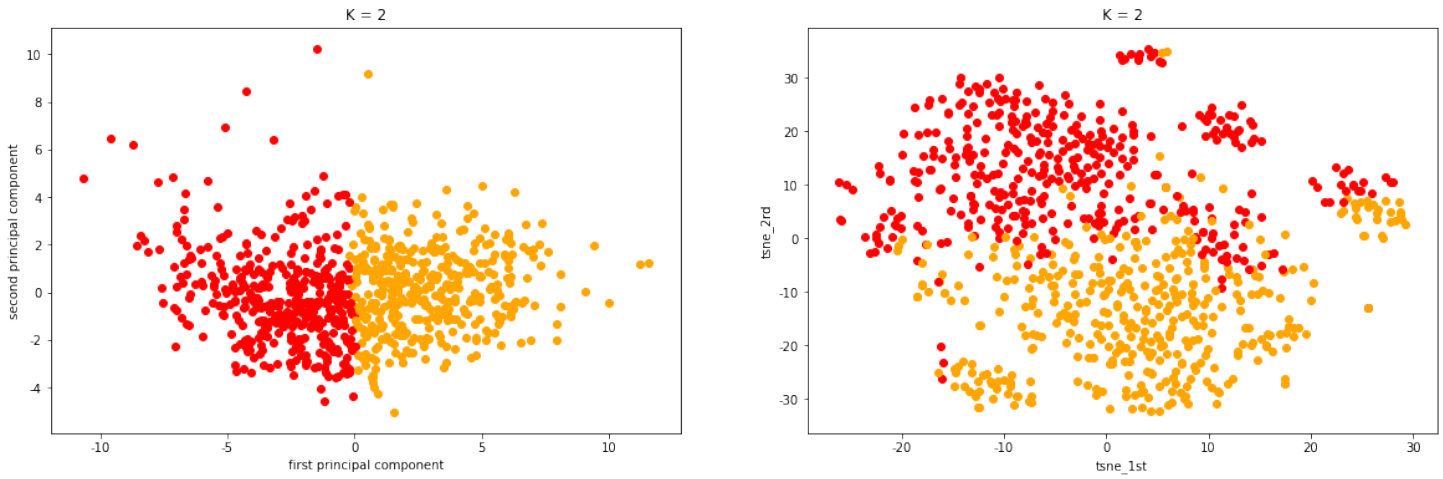


comments:

it seems that when k = 2 is the optimal clustering number of centroids compared with k >= 2 and k <= 10, generating the largest extra-cluster distance, the biggest difference between random data and the wiki data.

```
colors = ['r','orange']
kmeans_2 = KMeans(n_clusters=2, random_state=0).fit(X_wiki)

plt.figure(figsize=(20,6))
for i in range(len(X_wiki)):
    plt.subplot(1,2,1)
    plt.scatter(first_principle[i],second_principle[i],c = colors[kmeans_2.labels_[i]])
    plt.title('K = 2')
    plt.xlabel('first principal component')
    plt.ylabel('second principal component')

for i in range(len(X_wiki)):
    plt.subplot(1,2,2)
    plt.scatter(X_embedded[i,0],X_embedded[i,1],c = colors[kmeans_2.labels_[i]])
    plt.title('K = 2')
    plt.xlabel('tsne_1st')
    plt.ylabel('tsne_2rd')

plt.show()
```

```
/Users/apple/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:106: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



When k = 2, it seems that kmeans split the data based on the first principal components, while for tsne the split is not mainly based on either dimension, cause we can see the boundary of clustering on tsne is not linear and there is overlapping area. It is reasonable as tsne is not a linear dimension reduction method. And it seems that pca goes smoothly with kmeans, and pca can be more capable of interpreting the result of kmeans than tsne.