Perspective HW7
Yuxin Wu(yuxin@uchicago.edu)
March. 14th, 2020

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
```

In [2]:

```
np.random.seed(996)
```

1.

In [3]:

```
input_1 = np.array([5,8,7,8,3,4,2,3,4,5])
input_2 = np.array([8,6,5,4,3,2,2,8,9,8])
```

In [4]:

```
label_ran = np.random.choice(3,10)
```

In [5]:

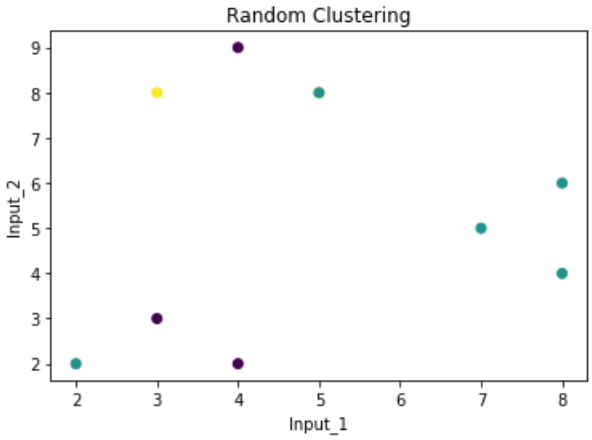
```
df = pd.DataFrame({"input_1": input_1, "input_2":input_2, "label":label_ran})
```

In [6]:

```
colors = list(df["label"])
plt.scatter(df['input_1'], df['input_2'], c = colors)
plt.xlabel('Input_1')
plt.ylabel('Input_2')
plt.title("Random Clustering")
```

Out[6]:

```
Text(0.5, 1.0, 'Random Clustering')
```



2.

In [7]:

```
centroids = {
    i: [df[df["label"] == i]["input_1"].mean(),\
        df[df["label"] == i]["input_2"].mean()]
    for i in range(3)
}
```

In [8]:

```
df3 = df.copy()
```

In [9]:

```
def clustering(df3, centroids):
    for i in centroids.keys():
        # sqrt((x1 - x2)^2 - (y1 - y2)^2)
        df3['distance_from_{}'.format(i)] = (
            np.sqrt(
                (df['input_1'] - centroids[i][0]) ** 2
                + (df['input_2'] - centroids[i][1]) ** 2
            )
        )
    centroid_distance_cols = ['distance_from_{}'.format(i) for i in centroids.keys()]
    df3['closest'] = df3.loc[:, centroid_distance_cols].idxmin(axis=1)
    df3['closest'] = df3['closest'].map(lambda x: int(x.lstrip('distance_from_')))
    return df3

dtt = clustering(df3, centroids)
dtt.drop(['label'], axis=1)
dtt.head()
```

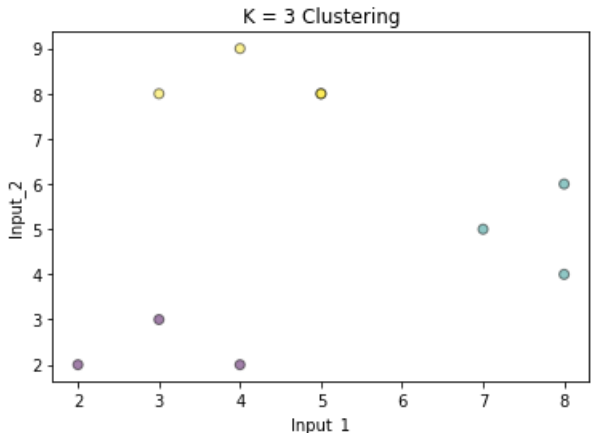
Out[9]:

	input_1	input_2	label	distance_from_0	distance_from_1	distance_from_2	closest
0	5	8	2	3.590110	3.162278	1.000000	2
1	8	6	1	4.533824	2.236068	4.472136	1
2	7	5	1	3.349959	1.000000	4.242641	1
3	8	4	1	4.384315	2.236068	5.656854	1
4	3	3	0	1.795055	3.605551	5.099020	0

3.

In [10]:

```
colors = list(dtt["closest"])
plt.scatter(dtt['input_1'], dtt['input_2'], c=colors, alpha=0.5, edgecolor='k')
plt.xlabel('Input_1')
plt.ylabel('Input_2')
plt.title("K = 3 Clustering")
plt.show()
```



4.

In [11]:

```
centroids2 = {
    i: [df[df["label"] == i]["input_1"].mean(),\
        df[df["label"] == i]["input_2"].mean()]
    for i in range(2)
}
```

In [12]:

```
df2 = df.copy()
```

In [13]:

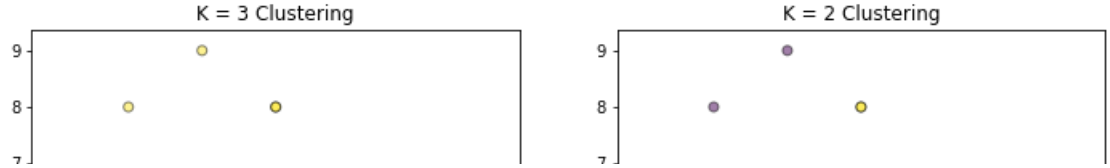
```
def clustering2(df2, centroids2):
    for i in centroids2.keys():
        # sqrt((x1 - x2)^2 - (y1 - y2)^2)
        df2['distance_from_{}'.format(i)] = (
            np.sqrt(
                (df2['input_1'] - centroids2[i][0]) ** 2
                + (df2['input_2'] - centroids2[i][1]) ** 2
            )
        )
    centroid_distance_cols = ['distance_from_{}'.format(i) for i in centroids2.keys()]
    df2['closest'] = df2.loc[:, centroid_distance_cols].idxmin(axis=1)
    df2['closest'] = df2['closest'].map(lambda x: int(x.lstrip('distance_from_')))
    return df2

dtt2 = clustering2(df2, centroids2)
dtt2.drop(['label'], axis=1)
dtt2.head()
```

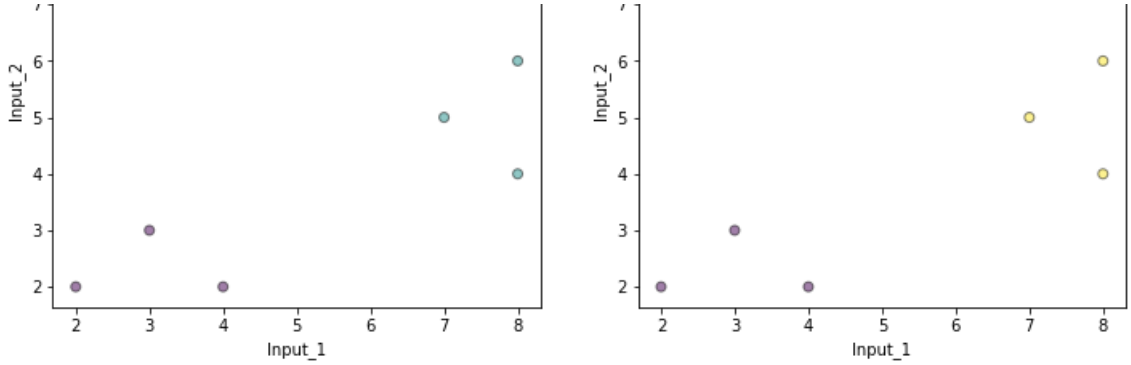
Out[13]:

	input_1	input_2	label	distance_from_0	distance_from_1	closest
0	5	8	2	3.590110	3.162278	1
1	8	6	1	4.533824	2.236068	1
2	7	5	1	3.349959	1.000000	1
3	8	4	1	4.384315	2.236068	1
4	3	3	0	1.795055	3.605551	0

In [14]:

```
fig, axs = plt.subplots(1,2, figsize =(12,5))
colors = list(dtt["closest"])
axs[0].scatter(dtt['input_1'], dtt['input_2'], c=colors, alpha=0.5, edgecolor='k')
axs[0].set_xlabel('Input_1')
axs[0].set_ylabel('Input_2')
axs[0].set_title("K = 3 Clustering")

colors = list(dtt2["closest"])
axs[1].scatter(dtt2['input_1'], dtt2['input_2'], c=colors, alpha=0.5, edgecolor='k')
axs[1].set_xlabel('Input_1')
axs[1].set_ylabel('Input_2')
axs[1].set_title("K = 2 Clustering")

plt.show()
```





5.

I would say when K = 3, it fits these data better. As we can see from the graphs above, when K = 2, the points in the same group are actually not that close. Especially the three points on at the top of the graph, they seem to belong to the same group, but they are assigned to different groups here. When K = 3, it makes more sense. When doing the clustering, chooing a right K number is also very important.

Application

6.

In [15]:

```
wiki = pd.read_csv("wiki.csv")
```

In [16]:

```
wiki_trans = StandardScaler().fit_transform(wiki)
```

In [17]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(wiki_trans)
principalDf = pd.DataFrame(pca.components_, columns = wiki.columns, index = ['principal component 1
', 'principal component 2']).T
principalDf.sort_values(by=['principal component 1'], ascending=False).head()
```

Out[17]:

	principal component 1	principal component 2
bi2	0.230924	0.083413
bi1	0.226193	0.056356
use3	0.218809	0.155136
use4	0.214558	0.160855
pu3	0.210863	0.028764

In [18]:

```
principalDf.sort_values(by=['principal component 2'], ascending=False).head()
```

Out[18]:

	principal component 1	principal component 2
exp4	0.099873	0.228482
use2	0.147852	0.218609

	use1	principal component 1	principal component 2
	vis3	0.175351	0.197642
domain_Engineering_Architecture		0.051309	0.171555

In [19]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(wiki_trans)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

In [20]:

```
principalDf
```

Out[20]:

	principal component 1	principal component 2
0	-0.150216	-1.982054
1	-3.314020	-0.791527
2	-4.682484	-0.312445
3	1.774200	1.985647
4	7.254695	2.012788
...	...	...
795	0.227143	1.473989
796	4.434784	-0.932248
797	1.449455	-0.170935
798	-2.888282	2.721061
799	-7.000656	2.805270

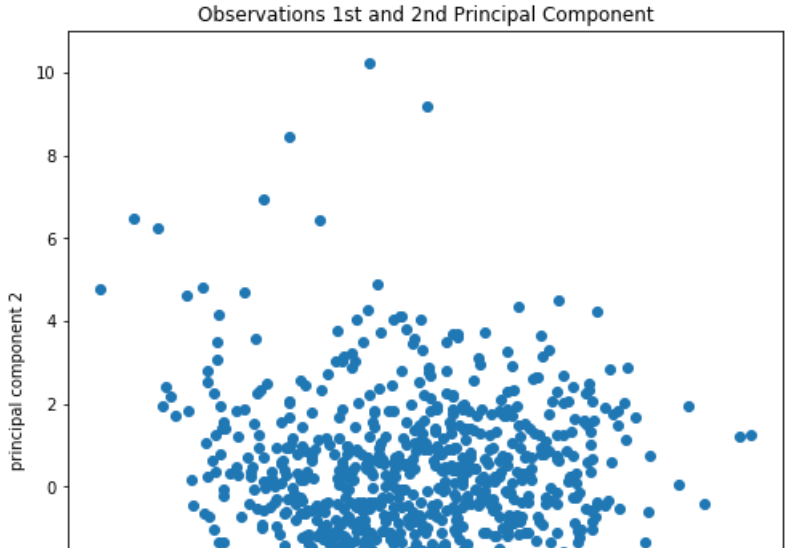
800 rows × 2 columns

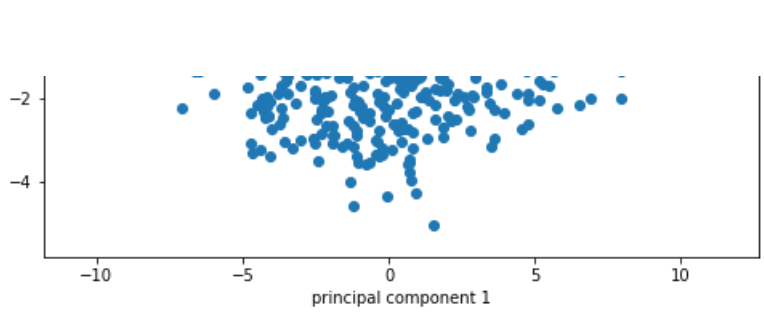
In [21]:

```
plt.figure(figsize = (8,8))
plt.title('Observations 1st and 2nd Principal Component')
plt.scatter(principalDf['principal component 1'], principalDf["principal component 2"])
plt.xlabel("principal component 1")
plt.ylabel("principal component 2")
```

Out[21]:

```
Text(0, 0.5, 'principal component 2')
```





As we can see from the results above, bi2, bi1, use3, use4, pu3 are the five variables that appear strongly correlated on the first principal component. And exp4, use2, use1, vis3, and domain_Engineering_Architecture are the five variables that appear strongly correlated on the second component.

7.

In [22]:

```
#PVE For PCn
pca = PCA()
pca.fit_transform(wiki_trans)
pve_all = pca.explained_variance_ratio_
print(pve_all)
```

```
[2.28106278e-01 6.37247454e-02 5.02370687e-02 4.07283521e-02
 3.76772356e-02 3.35209255e-02 3.03313773e-02 2.55217752e-02
 2.41742687e-02 2.39251475e-02 2.26565037e-02 2.07118345e-02
 2.02799632e-02 1.90332986e-02 1.79249263e-02 1.74765005e-02
 1.72633331e-02 1.61923173e-02 1.52846094e-02 1.45779108e-02
 1.43303520e-02 1.34971703e-02 1.29607608e-02 1.19257101e-02
 1.14687769e-02 1.12930650e-02 1.08554417e-02 9.88146747e-03
 9.51868716e-03 8.66253767e-03 8.63502268e-03 8.29878365e-03
 8.16074986e-03 7.89531673e-03 7.33346124e-03 7.27277692e-03
 6.91680403e-03 6.81634006e-03 6.60676170e-03 6.24976080e-03
 5.82409420e-03 5.81028140e-03 5.60030777e-03 5.42588559e-03
 5.38898417e-03 5.12077749e-03 5.05933842e-03 4.80033732e-03
 4.66136313e-03 4.53024524e-03 4.35630751e-03 3.84322030e-03
 3.76084687e-03 3.38273604e-03 2.35203635e-03 1.96716687e-03
 1.87953174e-04]
```

In [23]:

```
#PEV For PC1 and PC2
pca = PCA(n_components=2)
pca.fit_transform(wiki_trans)
pve = pca.explained_variance_ratio_
print(pve)
```

```
[0.22810628 0.06372474]
```

In [24]:

```
#cumulated PVE for PCn
pca = PCA()
pca.fit_transform(wiki_trans)
cum_pve_all = np.cumsum(pca.explained_variance_ratio_)
print(cum_pve_all)
```

```
[0.22810628 0.29183102 0.34206809 0.38279644 0.42047368 0.45399461
 0.48432598 0.50984776 0.53402203 0.55794717 0.58060368 0.60131551
 0.62159548 0.64062877 0.6585537  0.6760302  0.69329353 0.70948585
 0.72477046 0.73934837 0.75367872 0.76717589 0.78013665 0.79206236
 0.80353114 0.81482421 0.82567965 0.83556112 0.8450798  0.85374234
 0.86237736 0.87067615 0.8788369  0.88673221 0.89406567 0.90133845
 0.90825526 0.9150716  0.92167836 0.92792812 0.93375221 0.93956249
 0.9451628  0.95058869 0.95597767 0.96109845 0.96615779 0.97095812
 0.97561949 0.98014973 0.98450604 0.98834926 0.99211011 0.99549284
 0.99784488 0.99981205 1.        ]
```

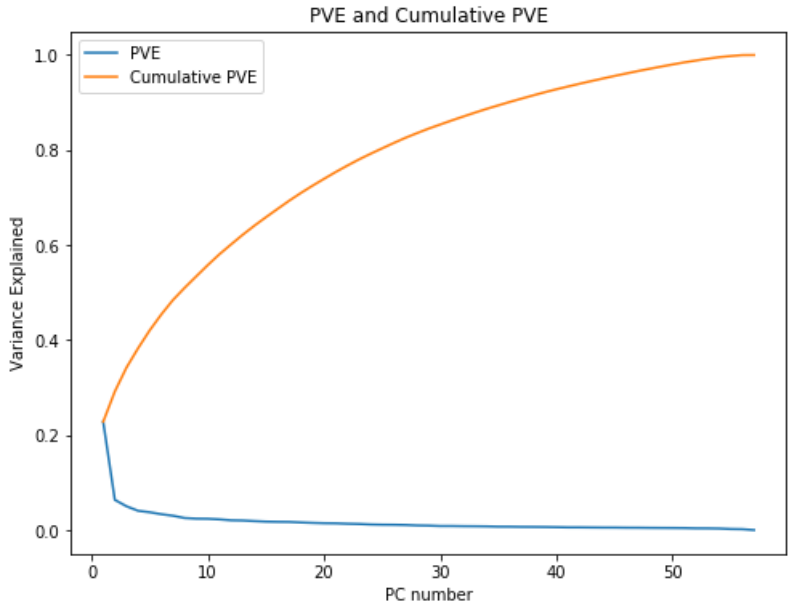
In [25]:

```
#cumulated PVE for PC1 and PC2
pca = PCA(n_components=2)
pca.fit_transform(wiki_trans)
cum_pve = np.cumsum(pca.explained_variance_ratio_)
print(cum_pve)
```

```
[0.22810628 0.29183101]
```

In [26]:

```
plt.figure(figsize=(8,6))
plt.plot(list(range(1,len(pve_all)+1)), pve_all, label = 'PVE')
plt.plot(list(range(1,len(pve_all)+1)), cum_pve_all, label = 'Cumulative PVE')
plt.xlabel("PC number")
plt.ylabel("Variance Explained")
plt.title("PVE and Cumulative PVE")
plt.legend()
plt.show()
```



According to the results above, the first component can explain 0.22810628 of the variance, and the second component can expalin 0.06372474 of the variance. All together, a cumulated 0.2918 of the variance can be explained by the first two component.

8.

In [27]:

```
tsne = TSNE(n_components=2, random_state = 996)
tsne_result = tsne.fit_transform(wiki_trans)
```

In [28]:

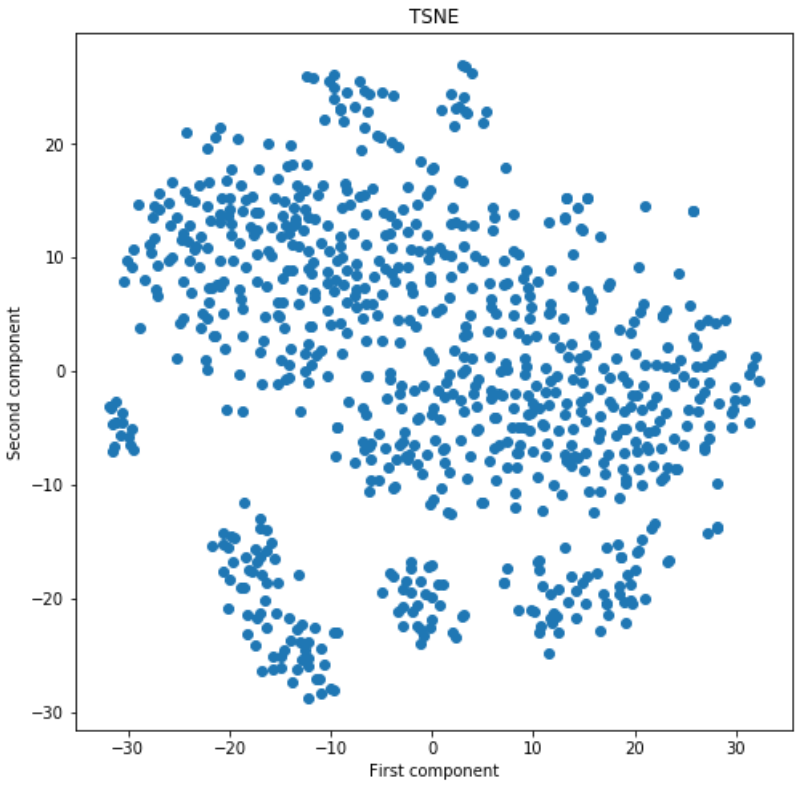
```
tsne_result = pd.DataFrame(tsne_result)
tsne_result = tsne_result.rename(columns = {0: 'tsne_1', 1:'tsne_2'})
```

In [29]:

```
plt.figure(figsize=(8,8))
plt.scatter(tsne_result['tsne_1'], tsne_result['tsne_2'])
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("TSNE")
```

Out[29]:

```
Text(0.5, 1.0, 'TSNE')
```



As we can see from the TSNE graph, there are serval small group and a large group of data. While in PCA graph, every dots are basically in a large group. In term of clusterring and capturing relationships between features, TSNE is definetely more accurate than PCA. But still the large group in TSNE graph may indicates that it can still not be so accurate.

9.

In [30]:

```
wiki_trans
```

Out[30]:

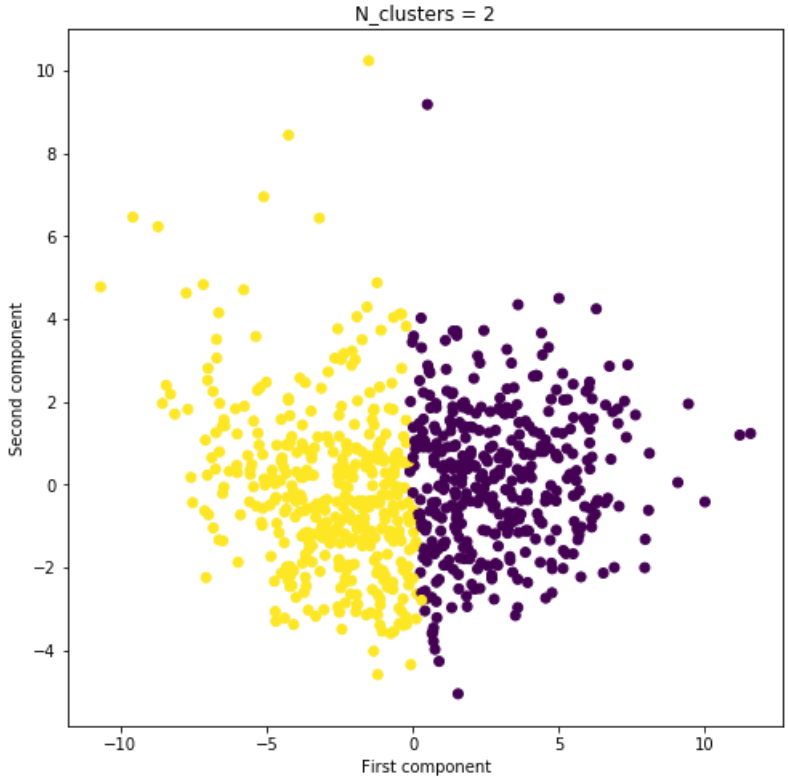
```
array([[-0.28718866, -0.86413245,  1.14257407, ..., -0.15171652,
        -0.05006262, -2.1618878 ],
       [-0.02204045, -0.86413245,  1.14257407, ..., -0.15171652,
        -0.05006262, -2.1618878 ],
       [-0.68491098, -0.86413245,  1.14257407, ..., -0.15171652,
        -0.05006262, -2.1618878 ],
       ...,
       [ 0.9059783 ,  1.15723001,  1.14257407, ..., -0.15171652,
        -0.05006262,  0.46255869],
       [-0.02204045,  1.15723001, -0.87521678, ..., -0.15171652,
        -0.05006262,  0.46255869],
       [ 0.37568187,  1.15723001,  1.14257407, ..., -0.15171652,
        -0.05006262,  0.46255869]])
```

In [31]:

```
km2 = KMeans(n_clusters = 2).fit(wiki_trans)
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(wiki_trans)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

plt.figure(figsize=(8,8))
plt.scatter(principalDf['principal component 1'],\
            principalDf['principal component 2'], c = km2.labels_)
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("N_clusters = 2")
```

Out[31]:

```
Text(0.5, 1.0, 'N_clusters = 2')
```

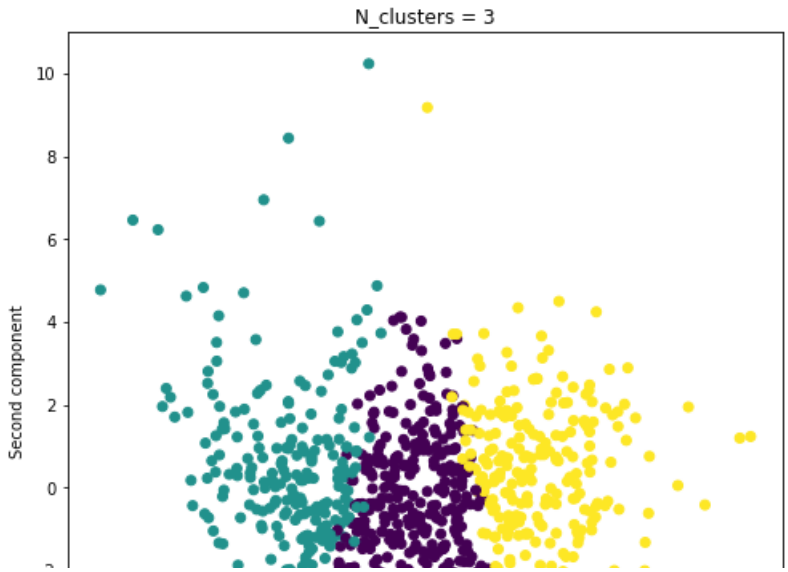


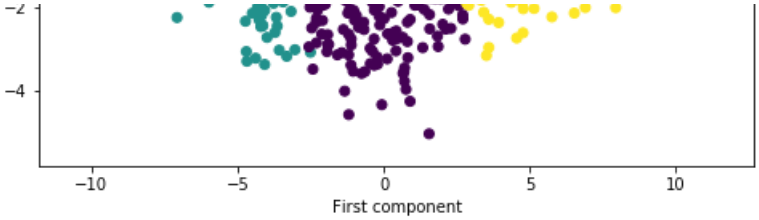
In [32]:

```
km2 = KMeans(n_clusters = 3).fit(wiki_trans)
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(wiki_trans)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

plt.figure(figsize=(8,8))
plt.scatter(principalDf['principal component 1'],\
            principalDf['principal component 2'], c = km2.labels_)
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("N_clusters = 3")
```

Out[32]:

```
Text(0.5, 1.0, 'N_clusters = 3')
```



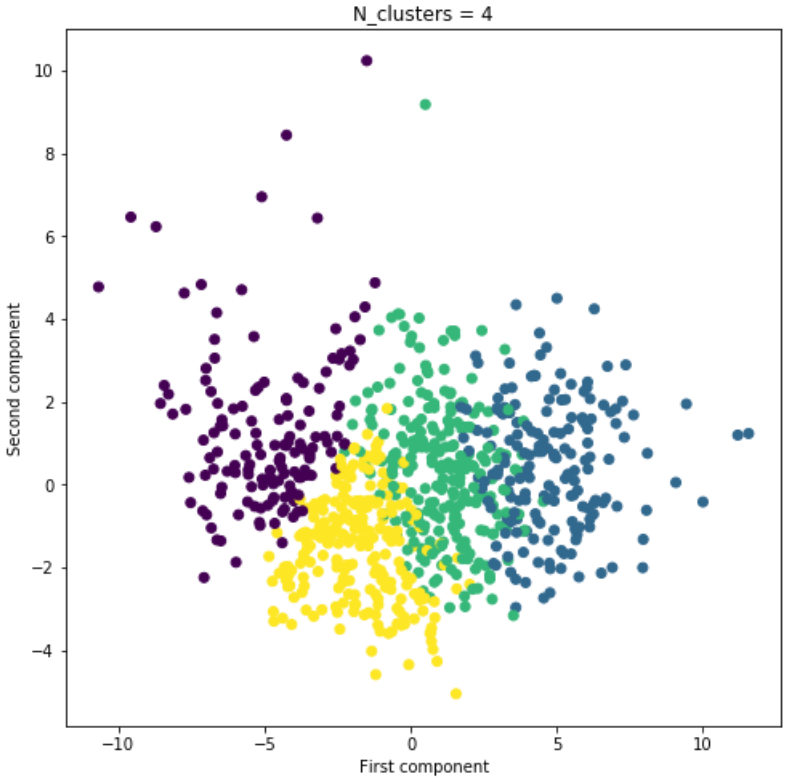


In [33]:

```
km2 = KMeans(n_clusters = 4).fit(wiki_trans)
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(wiki_trans)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

plt.figure(figsize=(8,8))
plt.scatter(principalDf['principal component 1'],\
            principalDf['principal component 2'], c = km2.labels_)
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("N_clusters = 4")
```

Out[33]:

```
Text(0.5, 1.0, 'N_clusters = 4')
```



As we can see from the graphs above, the clustering is mainly based on the value of first component. Especially when n_clusters =2 and 3. The value of second component barely contribute to the clustering. Only when n_clusters = 4, can we start to see that second component has a small contribute to the clustering.

10.

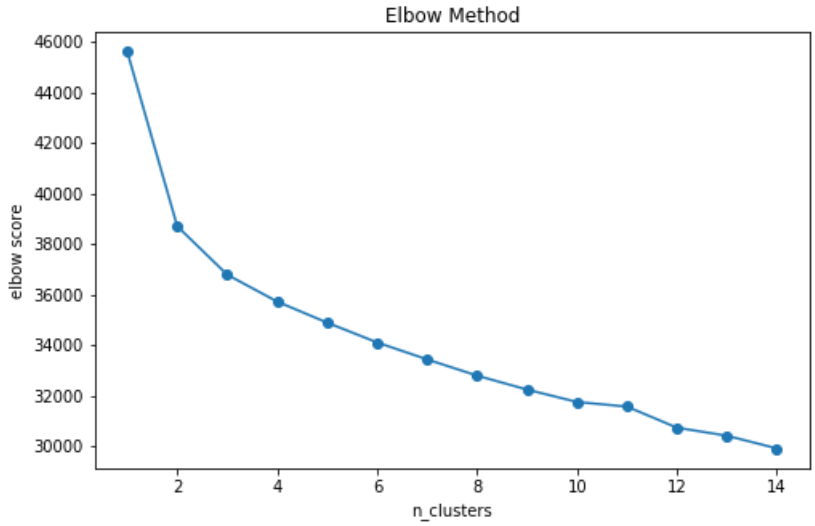
In [36]:

```
el_score = []

K = range(1,15)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(wiki_trans)
    el = el_score.append(kmeanModel.inertia_)
```

```
plt.figure(figsize=(8,5))
plt.scatter(list(range(1,15)),el_score)
plt.plot(list(range(1,15)),el_score)
plt.xlabel('n_clusters')
plt.ylabel('elbow score')
plt.title("Elbow Method")
```

Out[36]:

```
Text(0.5, 1.0, 'Elbow Method')
```

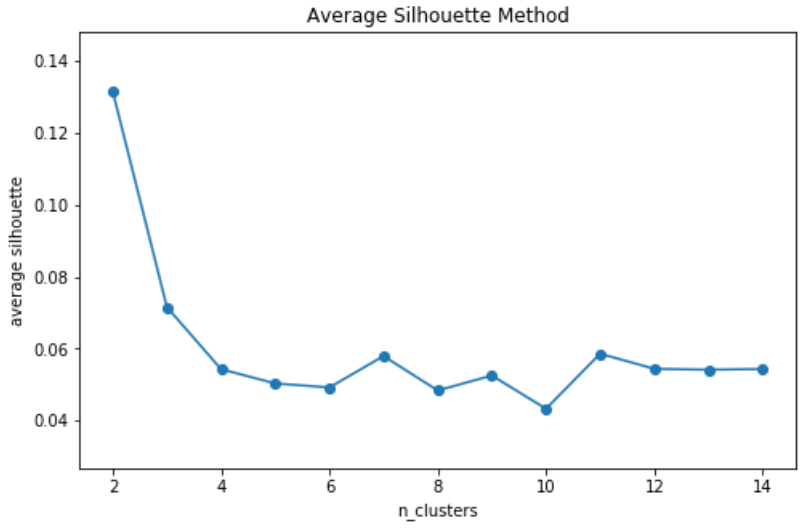


In [37]:

```
avgs_score = []

K = range(2,15)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    cluster_labels = kmeanModel.fit_predict(wiki_trans)
    silhouette_avg = silhouette_score(wiki_trans, cluster_labels)
    avgs_score.append(silhouette_avg)

plt.figure(figsize=(8,5))
plt.scatter(list(range(2,15)),avgs_score)
plt.plot(list(range(2,15)),avgs_score)
plt.xlabel('n_clusters')
plt.ylabel('average silhouette')
plt.title("Average Silhouette Method")
```

Out[37]:

```
Text(0.5, 1.0, 'Average Silhouette Method')
```



Based on the elbow method, we can see a small elbow at the value of 11, therfore, we set first optimal number of clusters as 11. Based on the average silhouette graph, when number = 2, the avg silhouette is the largest, thus we set the optimal number of clusters as 2.

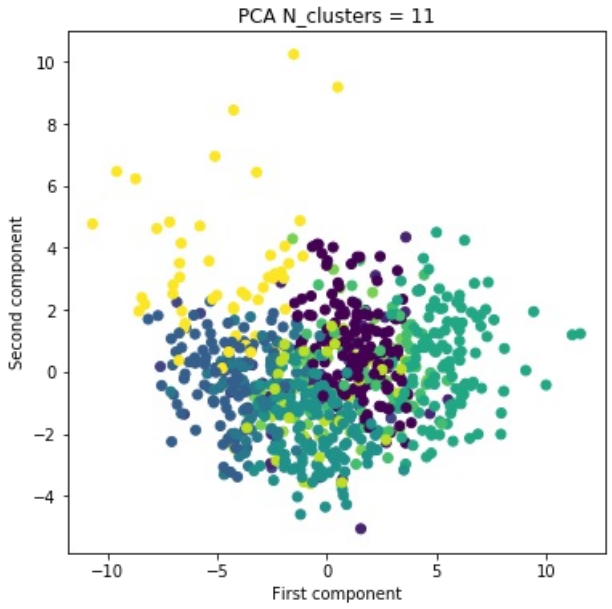
11.

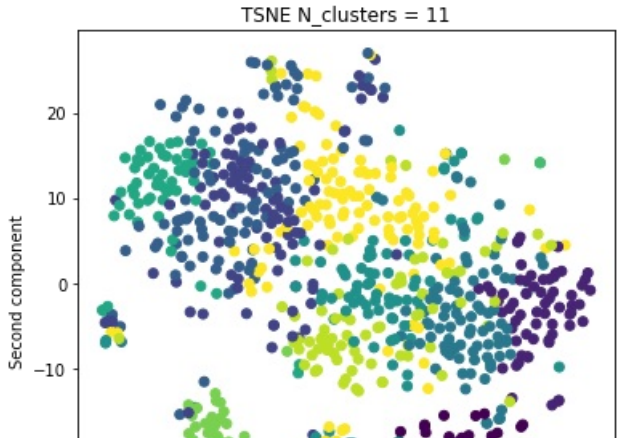
In [38]:

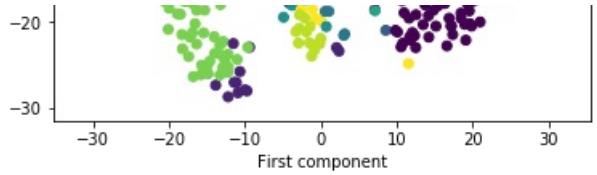
```
km2 = KMeans(n_clusters = 11).fit(wiki_trans)
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(wiki_trans)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

plt.figure(figsize=(6,6))
plt.scatter(principalDf['principal component 1'],\
            principalDf['principal component 2'], c = km2.labels_)
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("PCA N_clusters = 11")

km2 = KMeans(n_clusters = 11).fit(wiki_trans)
plt.figure(figsize=(6,6))
plt.scatter(tsne_result['tsne_1'],\
            tsne_result['tsne_2'], c = km2.labels_)
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("TSNE N_clusters = 11")
```

Out[38]:

```
Text(0.5, 1.0, 'TSNE N_clusters = 11')
```





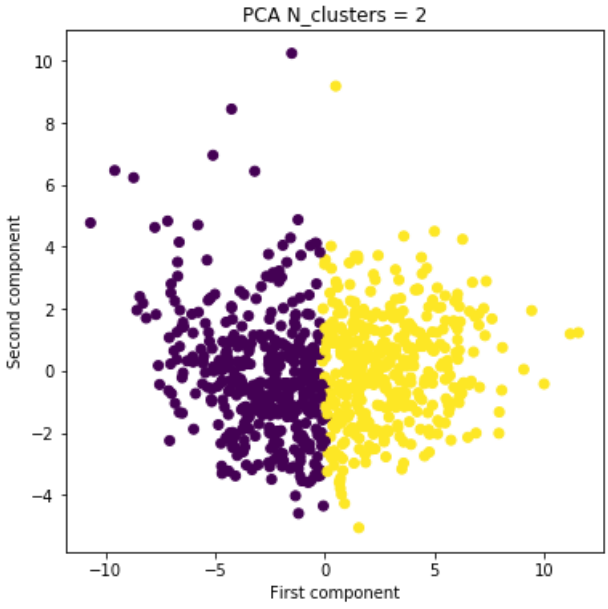


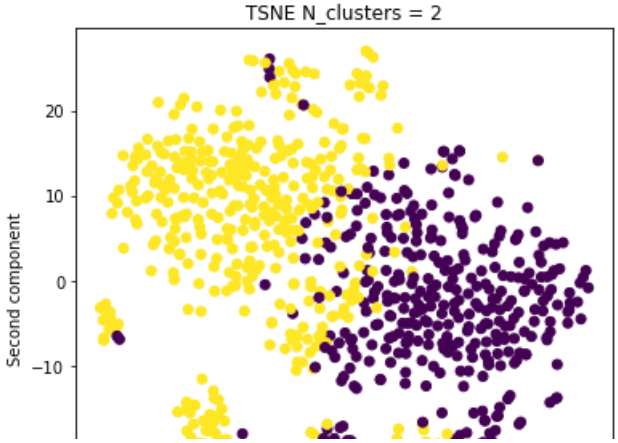
In [39]:

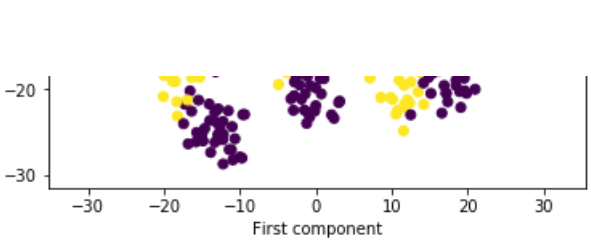
```
km2 = KMeans(n_clusters = 2).fit(wiki_trans)
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(wiki_trans)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

plt.figure(figsize=(6,6))
plt.scatter(principalDf['principal component 1'],\
            principalDf['principal component 2'], c = km2.labels_)
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("PCA N_clusters = 2")

km2 = KMeans(n_clusters = 2).fit(wiki_trans)
plt.figure(figsize=(6,6))
plt.scatter(tsne_result['tsne_1'],\
            tsne_result['tsne_2'], c = km2.labels_)
plt.xlabel('First component')
plt.ylabel('Second component')
plt.title("TSNE N_clusters = 2")
```

Out[39]:

```
Text(0.5, 1.0, 'TSNE N_clusters = 2')
```







Firstly, comparing n_clusters = 2 and n_cluster = 11. Visually speaking, n_clusters = 2 returns a much better result than n_clusters = 11. As we can see from the graph above,when n_cluster, the boundaries are fuzzy, and there are a lot of overlap. It is messy. But n_cluster = 2 can give us a much clearer result.

Secondly, comparing PCA and TSNE (only using n_cluster = 2). We can see that PCA has a much clearer boundary, but the boundary of TSNE is fuzzy and we can see some obvious overlap in TSNE graph.One possible reason for this is that PCA assumes that the relationship between the variables are linearly related. And as we can see from the graph above, the boundary in PCA graph is almost linear. This may indicates that there is a linear relationship among the features, thus, PCA can give us a better results.