Xu_Weijie_HW7

March 13, 2020

[1]:
```python
import pandas as pd
import numpy as np
import math
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_score
```

[2]:
```python
SEED = 970608
```

1 k-Means Clustering "By Hand"

1.1 Task 1

[3]:
```python
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])
```

[4]:
```python
# Assign initial label
np.random.seed(SEED)
df_k3 = pd.DataFrame({'input_1': input_1, 'input_2': input_2})
init_labels = np.random.choice(3, 10, replace=True)
df_k3['k_label'] = init_labels
df_k3
```

[4]:

	input_1	input_2	k_label
0	5	8	1
1	8	6	0
2	7	5	0
3	8	4	0
4	3	3	2
5	4	2	2
6	2	2	2
7	3	8	2
8	4	9	1
9	5	8	0

2 Task 2

[5]:
```python
def fit_kmeans(num_k, df, max_iter):

    fit_df = df.copy()
    for i in range(max_iter):
        # Calculate centroilds
        centroids = {}
        for k in range(num_k):
            cent1 = fit_df[fit_df['k_label'] == k]['input_1'].mean()
            cent2 = fit_df[fit_df['k_label'] == k]['input_2'].mean()
            centroids[k] = (cent1, cent2)
        # Update k labels for each observation
        new_k_labels = []
        for idx, row in fit_df.iterrows():
            min_distance = 50
            for k, v in centroids.items():
                eu_distance = math.sqrt((row['input_1'] - v[0]) ** 2 +␣
↪(row['input_2'] - v[1]) ** 2)
                if eu_distance < min_distance:
                    min_distance = eu_distance
                    new_k = k
            new_k_labels.append(new_k)
        fit_df['k_label'] = new_k_labels

    return fit_df
```

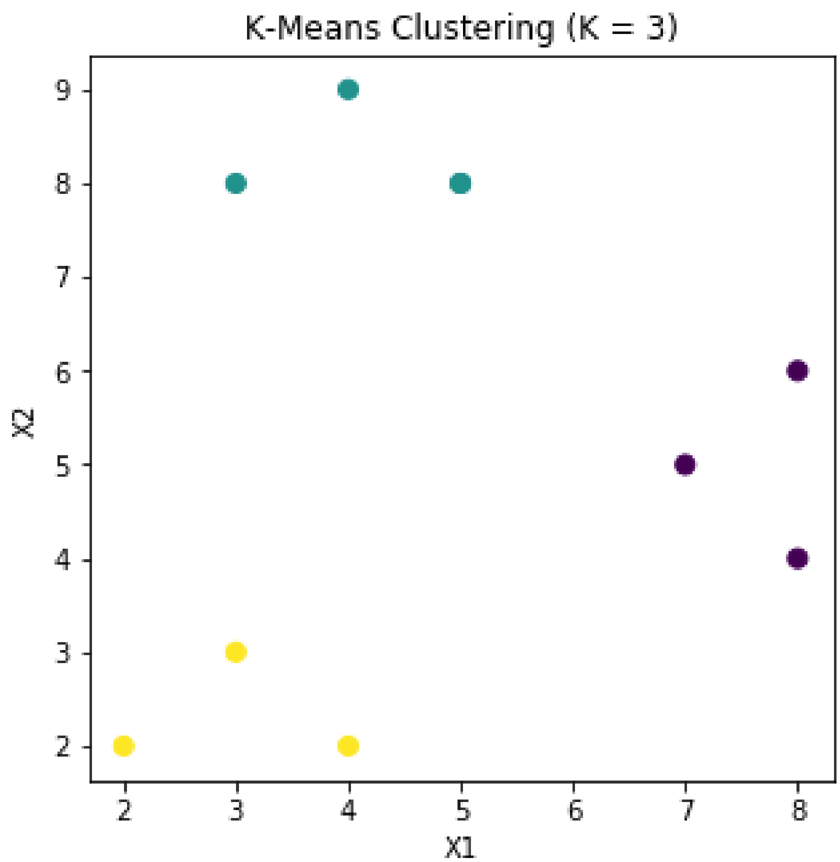
[6]:
```python
fit_k3 = fit_kmeans(3, df_k3, 50)
fit_k3
```

[6]:

	input_1	input_2	k_label
0	5	8	1
1	8	6	0
2	7	5	0
3	8	4	0
4	3	3	2
5	4	2	2
6	2	2	2
7	3	8	1
8	4	9	1
9	5	8	1

2.1 Task 3

[7]:
```
fig = plt.figure(figsize=(5, 5))
colors = list(fit_k3['k_label'])
plt.scatter(fit_k3['input_1'], fit_k3['input_2'], c=colors, s=50)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K-Means Clustering (K = 3)')
plt.show()
```



2.2 Task 4

[8]:
```
np.random.seed(SEED)
df_k2 = pd.DataFrame({'input_1': input_1, 'input_2': input_2})
init_labels = np.random.choice(2, 10, replace=True)
df_k2['k_label'] = init_labels
df_k2
```

[8]:

	input_1	input_2	k_label
0	5	8	1
1	8	6	0
2	7	5	0
3	8	4	0
4	3	3	0
5	4	2	0
6	2	2	0
7	3	8	1
8	4	9	0
9	5	8	1

[9]:

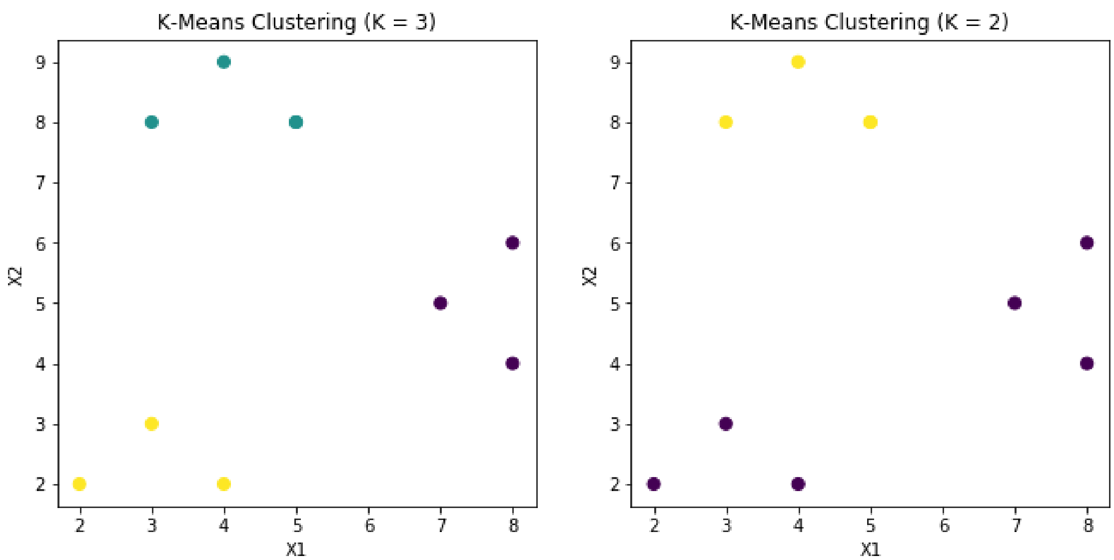
```
fit_k2 = fit_kmeans(2, df_k2, 50)
fit_k2
```

[9]:

	input_1	input_2	k_label
0	5	8	1
1	8	6	0
2	7	5	0
3	8	4	0
4	3	3	0
5	4	2	0
6	2	2	0
7	3	8	1
8	4	9	1
9	5	8	1

[10]:

```
# Plot k-means with k=3 and k=2
colors_k3, colors_k2 = list(fit_k3['k_label']), list(fit_k2['k_label'])

fig, axs = plt.subplots(1, 2, figsize=(11,5))

axs[0].scatter(fit_k3['input_1'], fit_k3['input_2'], c=colors_k3, s=50)
axs[0].set_xlabel('X1')
axs[0].set_ylabel('X2')
axs[0].set_title('K-Means Clustering (K = 3)')

axs[1].scatter(fit_k2['input_1'], fit_k2['input_2'], c=colors_k2, s=50)
axs[1].set_xlabel('X1')
axs[1].set_ylabel('X2')
axs[1].set_title('K-Means Clustering (K = 2)')

plt.show()
```



2.3 Task 5

According to the plots presented above, the first clustering with k=3 performs better than the clustering with k=2. When k=2, since it fails to tease apart the two clusters at bottom, the distance between different points within a single cluster is still very high. In fact, the distance between a point in the purple-colored cluster and a point in the yellow-colored cluster is very similar to the distance between a point on the left end and a point on the right end within the purple-colored cluster, which means that the within-cluster distance difference is similar to the cross-cluster difference. But this problem has been resolved to a great extent if k equals 3. Therefore, the first clustering performs better.

3 Dimension reduction

3.1 Task 6

According to the loading vectors, the top 5 variables that are highly correlated on the first component are "bi2", "bi1", "use3", "use4", and "pu3".

The top 5 variables highly correlated on the second component are "peu1", "inc1", "sa3", "sa1", and "exp4".

[11]:
```
X = pd.read_csv('data/wiki.csv')
features = X.columns
num_features = len(list(features))

# Standardize data set.
X = StandardScaler().fit_transform(X)
X
```

[11]: 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]])

[12]:
```
loading_labels = ['V{}'.format(i+1) for i in range(num_features)]
loading_df = pd.DataFrame(PCA(random_state=SEED).fit(X).components_.T,␣
↪index=features, columns=loading_labels)
loading_df[['V1', 'V2']]
```

[12]:

	V1	V2
age	-0.021805	0.088385
gender	-0.035086	-0.149461
phd	-0.030501	0.030435
yearsexp	-0.034190	0.062365
userwiki	0.081363	0.134387
pu1	0.192827	0.008273
pu2	0.190588	0.017669
pu3	0.210863	0.028776
peu1	0.061228	-0.271741
peu2	0.113719	-0.222368
peu3	0.100219	-0.068459
enj1	0.145666	-0.151012
enj2	0.131110	-0.227602
qu1	0.178057	-0.038122
qu2	0.163778	-0.066422
qu3	0.157956	-0.033472
qu4	-0.060797	-0.103458
qu5	0.183365	0.010912
vis1	0.171153	-0.025208
vis2	0.114559	-0.056218
vis3	0.175351	0.197635
im1	0.160432	0.111106
im2	0.077810	-0.059775
im3	0.160803	0.044004
sa1	0.121658	-0.229926
sa2	0.117590	-0.226760
sa3	0.120376	-0.242325

```
use1                              0.181477  0.197827
use2                              0.147852  0.218629
use3                              0.218809  0.155152
use4                              0.214558  0.160865
use5                              0.206539  0.029823
pf1                               0.102338  0.114371
pf2                               0.103448  0.018605
pf3                               0.109632  0.094173
jr1                               0.080867 -0.136968
jr2                               0.062216 -0.106297
bi1                               0.226193  0.056374
bi2                               0.230924  0.083431
inc1                              0.104667 -0.245440
inc2                              0.095802 -0.202021
inc3                              0.081402 -0.220986
inc4                              0.089707 -0.202022
exp1                              0.208592  0.070544
exp2                              0.195043 -0.029560
exp3                              0.144023 -0.126417
exp4                              0.099873  0.228494
exp5                              0.110628  0.076096
domain_Sciences                   0.021982 -0.014537
domain_Health.Sciences           -0.017158 -0.015478
domain_Engineering_Architecture   0.051309  0.171484
domain_Law_Politics              -0.094775 -0.014887
uoc_position_Associate            0.010922  0.013134
uoc_position_Assistant            0.007123  0.002311
uoc_position_Lecturer            -0.018041 -0.023591
uoc_position_Instructor           0.004251 -0.003785
uoc_position_Adjunct             -0.007849 -0.005301
```

[13]:
```
loading_df['V1'].iloc[(-np.abs(loading_df['V1'].values)).argsort()].head()
```

[13]:
```
bi2     0.230924
bi1     0.226193
use3    0.218809
use4    0.214558
pu3     0.210863
Name: V1, dtype: float64
```

[14]:
```
loading_df['V2'].iloc[(-np.abs(loading_df['V2'].values)).argsort()].head()
```

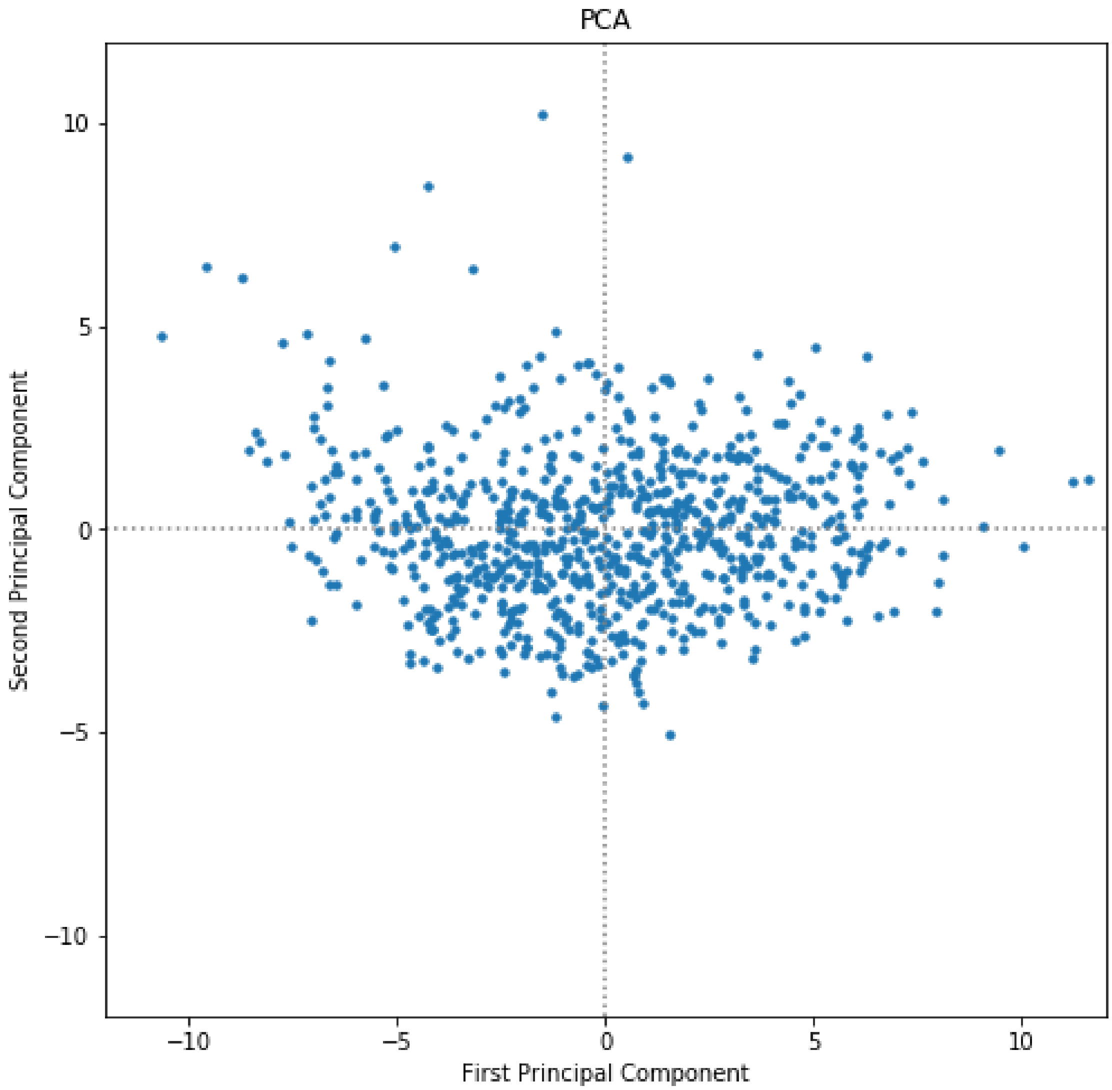
[14]:
```
peu1   -0.271741
inc1   -0.245440
sa3    -0.242325
sa1    -0.229926
exp4    0.228494
```

```
Name: V2, dtype: float64
```

[15]:
```
pca = PCA(random_state=SEED)
pc_labels = ['PC{}'.format(i+1) for i in range(num_features)]
pca_df = pd.DataFrame(pca.fit_transform(X), columns=pc_labels)
pca_df[['PC1', 'PC2']]
```

[15]:
```
          PC1       PC2
0   -0.150216 -1.982012
1   -3.314020 -0.791963
2   -4.682484 -0.312449
3    1.774200  1.985882
4    7.254695  2.013041
..        ...       ...
795  0.227143  1.474271
796  4.434784 -0.931830
797  1.449455 -0.170542
798 -2.888282  2.721003
799 -7.000656  2.805396

[800 rows x 2 columns]
```

[16]:
```
fig = plt.figure(figsize=(8, 8))

plt.xlim(-12,12)
plt.ylim(-12,12)

plt.scatter(pca_df['PC1'], pca_df['PC2'], s=11)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA')

plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
plt.vlines(0,-12,12, linestyles='dotted', colors='grey')

plt.show()
```



3.2 Task 7

According to the cumulative PVE, around 30% of the variance is explained by the first two components. More specifically, 22.81% of the variance is explained by the first component and 6.37% by the second component.

[17]:
```
# The proportion of variance explained (PVE)
pve = pca.explained_variance_ratio_
print(f'The proportion of variance explained (PVE):\n {pve}')
```

```
The proportion of variance explained (PVE):
 [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]
```

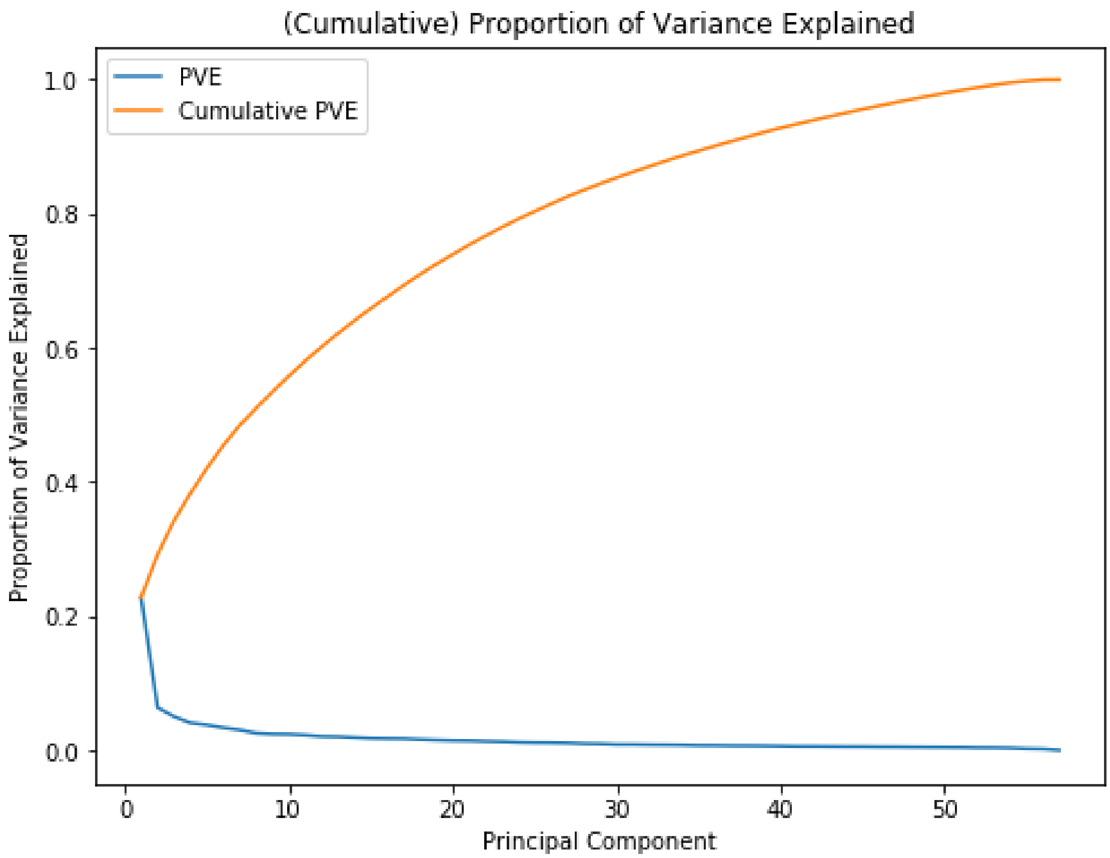
[18]:

```
# The cumulative PVE
cum_pve = np.cumsum(pca.explained_variance_ratio_)
print(f'The cumulative PVE:\n {cum_pve}')
```

```
The cumulative PVE:
 [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.        ]
```

[19]:

```
# Plot PVE and cumulative PVE

plt.figure(figsize=(8,6))
plt.plot(list(range(1, len(pve)+1)), pve, label='PVE')
plt.plot(list(range(1, len(pve)+1)), cum_pve, label='Cumulative PVE')
plt.ylabel('Proportion of Variance Explained')
plt.xlabel('Principal Component')
plt.title('(Cumulative) Proportion of Variance Explained')
plt.legend()

plt.show()
```



3.3 Task 8

As we can see from the plot below, the whole data set is separated into several clusters: there's a large block in the middle, surrounded by several small clusters with clear boundaries. This suggests that some observations in our original data set are correlated with each other, which sets them apart from others.

[20]:
```
X_embedded = TSNE(n_components=2, random_state=SEED)
tsne_df = pd.DataFrame(X_embedded.fit_transform(X), columns=['dim1', 'dim2'])
tsne_df
```

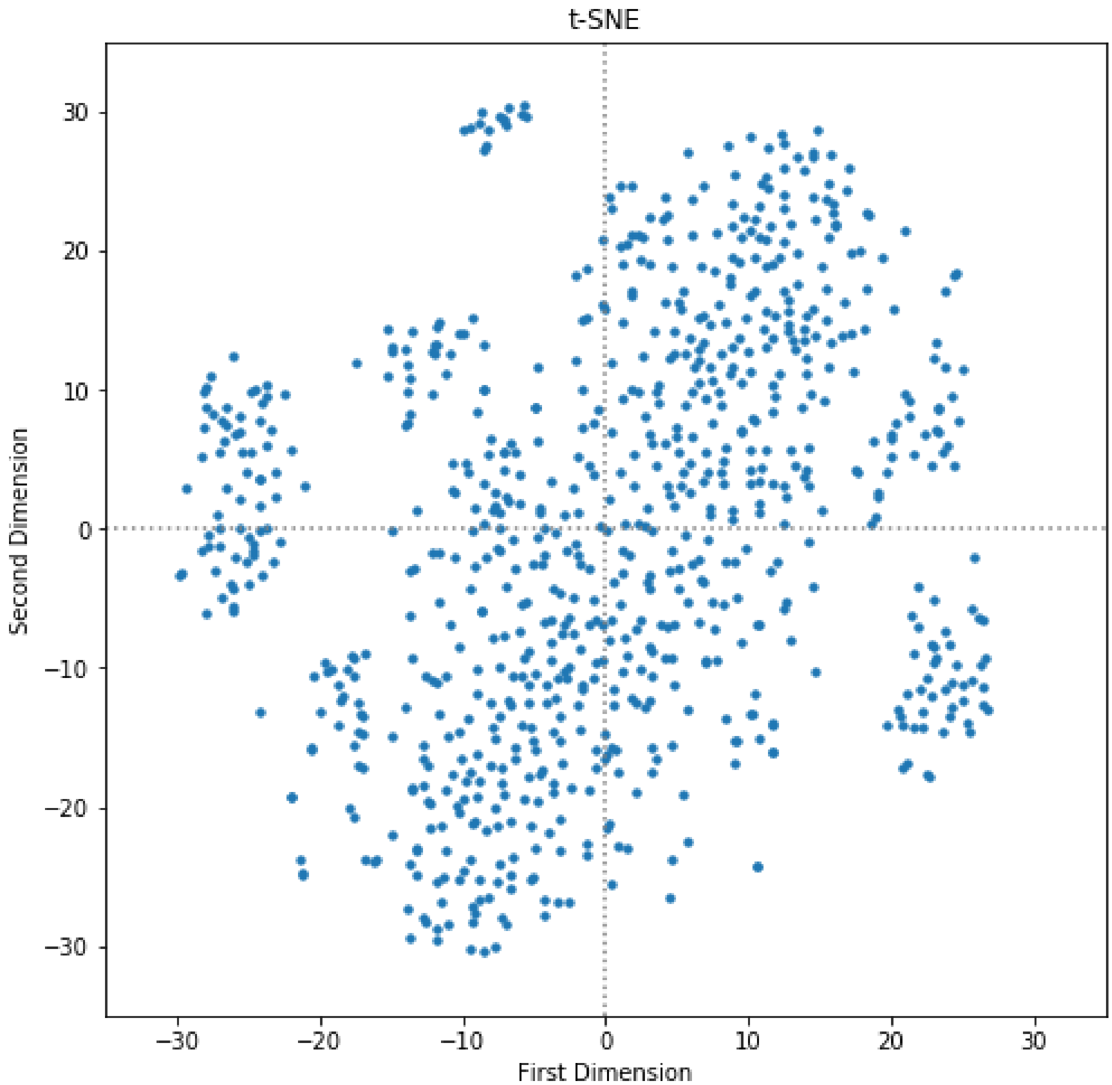
[20]:

	dim1	dim2
0	-21.001440	3.036607
1	-25.562788	8.068387
2	21.905931	-7.103955
3	20.714212	-13.510269
4	-13.735229	-29.467089
..	…	…
795	-4.813096	11.599584
796	-13.989577	-12.749982

```
797   3.051894  -3.302266
798  -4.906739   8.780779
799  23.157356  13.340831

[800 rows x 2 columns]
```

[21]:

```
fig = plt.figure(figsize=(8, 8))

plt.xlim(-35,35)
plt.ylim(-35,35)

plt.hlines(0,-35,35, linestyles='dotted', colors='grey')
plt.vlines(0,-35,35, linestyles='dotted', colors='grey')

plt.scatter(tsne_df['dim1'], tsne_df['dim2'], s=11)
plt.xlabel('First Dimension')
plt.ylabel('Second Dimension')
plt.title('t-SNE')

plt.show()
```



4 Clustering

4.1 Task 9

As we can see from the plots below, the k-means clustering with 2 and 3 clusters performs better to separate the data set since we can find a relatively clear boundary between each cluster. However, as for k-means clustering with k=4, the performance is not that good since clusters are overlapped and the boundaries between them become very fuzzy.

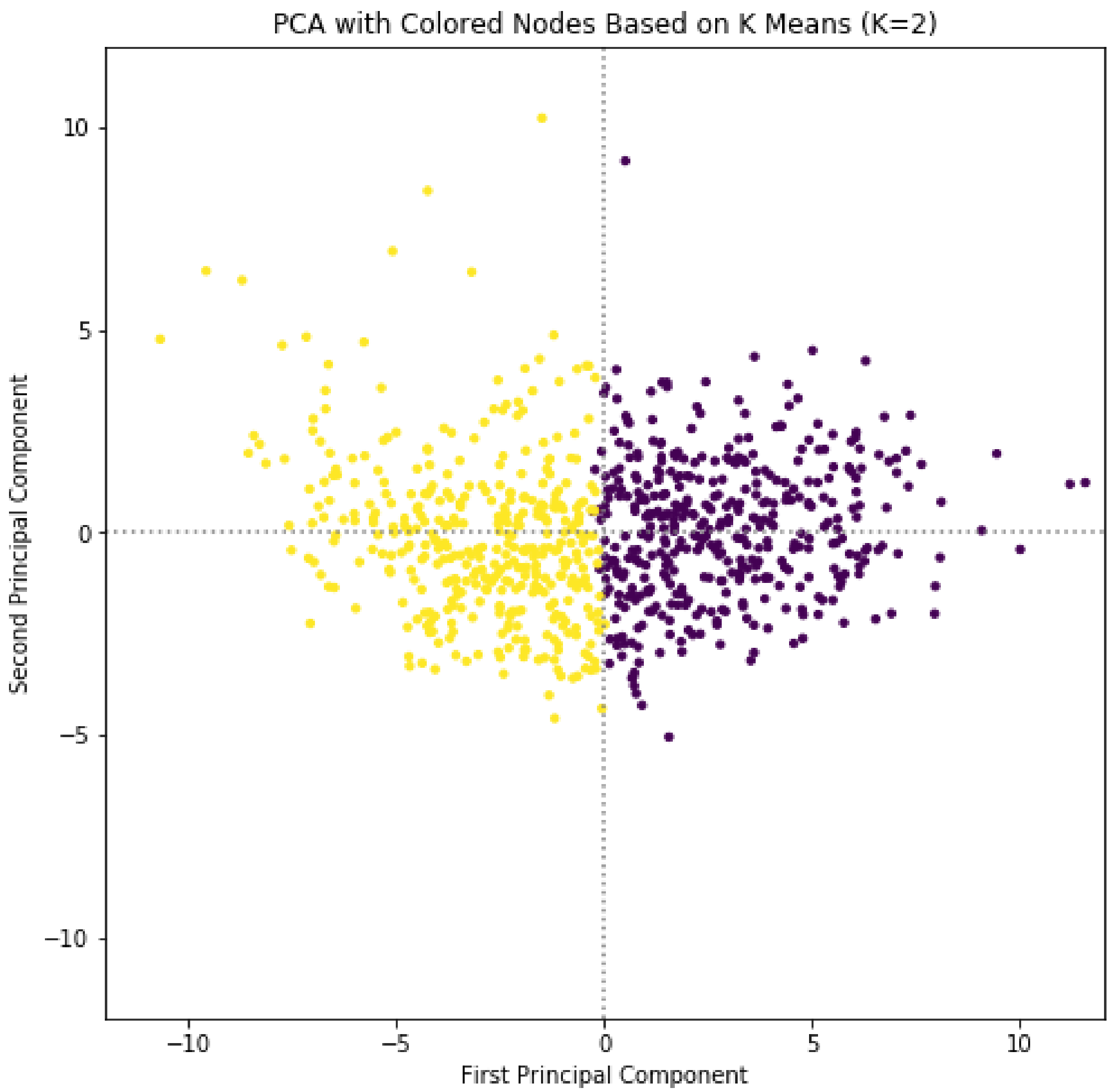
[22]:
```
X = pd.read_csv('data/wiki.csv')
features = X.columns
num_features = len(list(features))
```

```python
# Standardize data set.
X = StandardScaler().fit_transform(X)
X
```

[22]:
```
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]])
```

4.1.1 K = 2

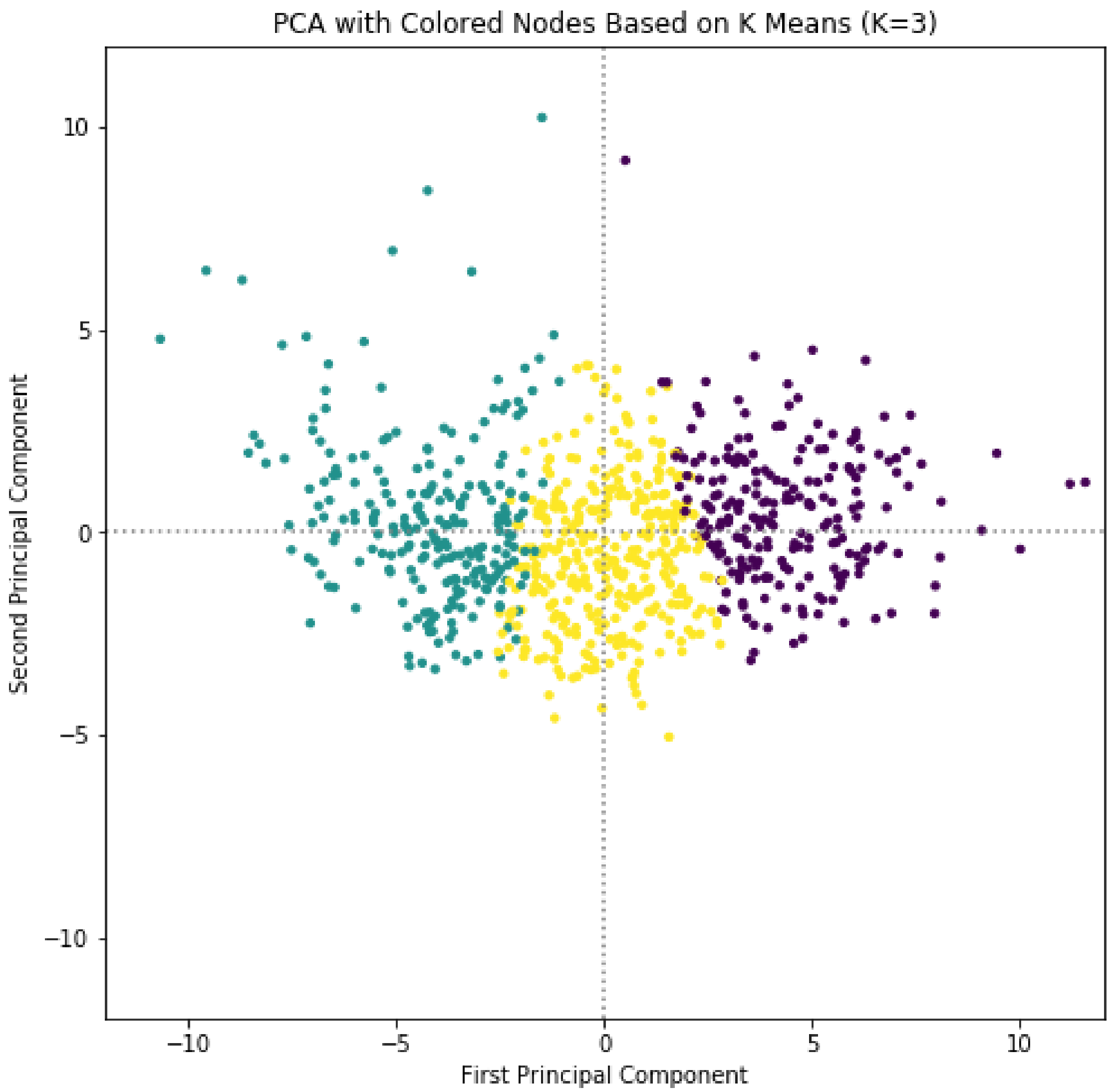
[23]:
```python
# Fit k means with k=2
kmeans2 = KMeans(n_clusters=2, random_state=SEED).fit(X)

# Plot obs on PC1 and PC2
fig = plt.figure(figsize=(8, 8))

plt.xlim(-12,12)
plt.ylim(-12,12)

plt.scatter(pca_df['PC1'], pca_df['PC2'], s=11, c=kmeans2.labels_)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA with Colored Nodes Based on K Means (K=2)')

plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
plt.vlines(0,-12,12, linestyles='dotted', colors='grey')

plt.show()
```



4.1.2 K = 3

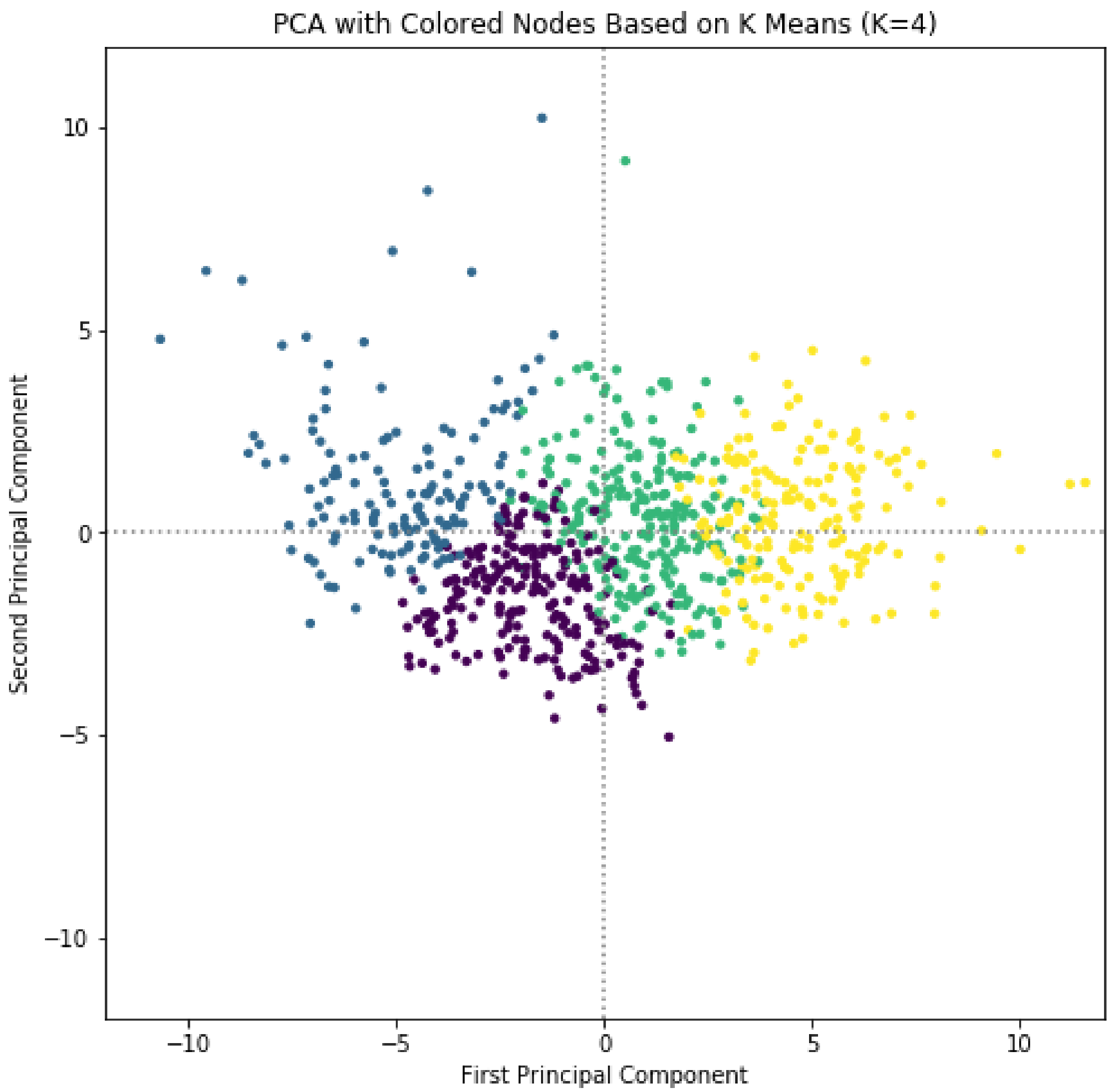
[24]:
```
# Fit k means with k=3
kmeans3 = KMeans(n_clusters=3, random_state=SEED).fit(X)

# Plot obs on PC1 and PC2
fig = plt.figure(figsize=(8, 8))

plt.xlim(-12,12)
plt.ylim(-12,12)

plt.scatter(pca_df['PC1'], pca_df['PC2'], s=11, c=kmeans3.labels_)
plt.xlabel('First Principal Component')
```

```python
plt.ylabel('Second Principal Component')
plt.title('PCA with Colored Nodes Based on K Means (K=3)')

plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
plt.vlines(0,-12,12, linestyles='dotted', colors='grey')

plt.show()
```



4.1.3 K = 4

[25]:
```
# Fit k means with k=4
kmeans4 = KMeans(n_clusters=4, random_state=SEED).fit(X)

# Plot obs on PC1 and PC2
fig = plt.figure(figsize=(8, 8))

plt.xlim(-12,12)
plt.ylim(-12,12)

plt.scatter(pca_df['PC1'], pca_df['PC2'], s=11, c=kmeans4.labels_)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA with Colored Nodes Based on K Means (K=4)')

plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
plt.vlines(0,-12,12, linestyles='dotted', colors='grey')

plt.show()
```



4.2 Task 10

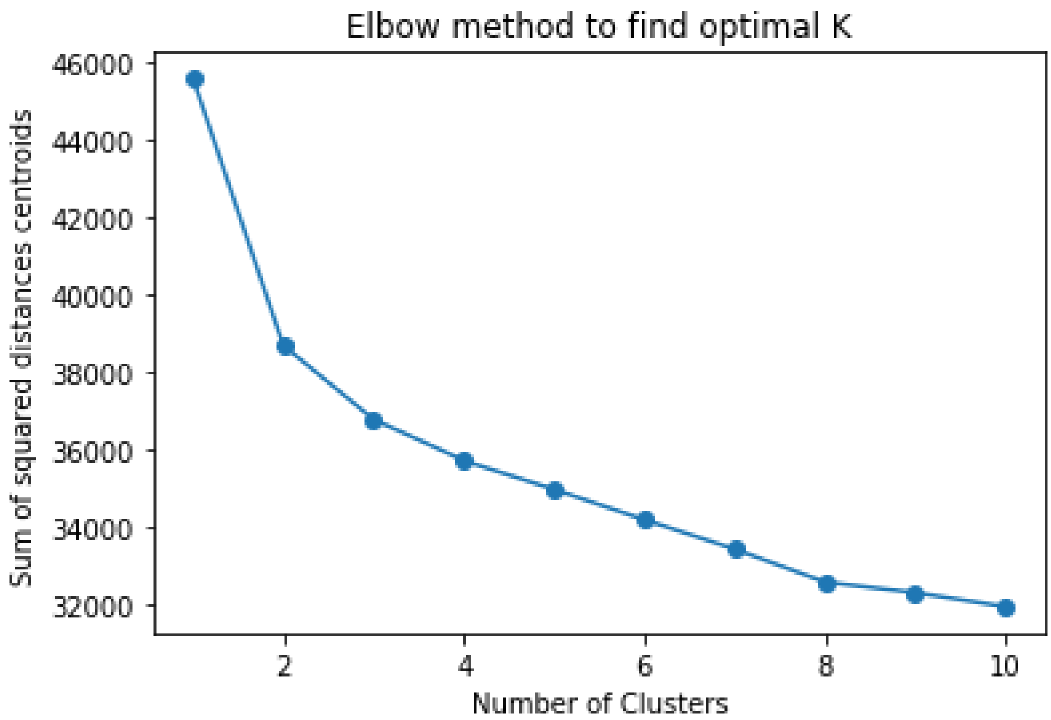
According to results of both the elbow method and the average silouette method, we would select k=2 as the optimal number of clusters for k-means clustering.

4.2.1 Elbow Method

The optimal value for k is approximately 2 since the sum of squared distance within culsters decreases the most when k increases from 1 to 2.

[26]:
```
inertias = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=SEED).fit(X)
    inertias.append(kmeans.inertia_)
```

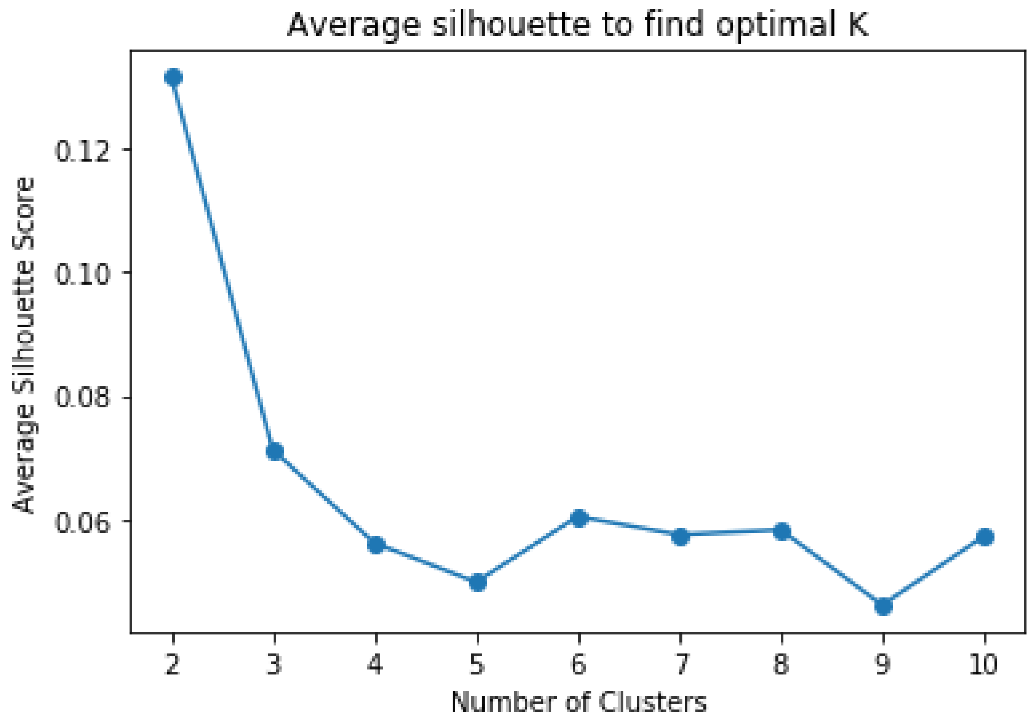
```
plt.plot(list(range(1, 11)), inertias, '-o')
plt.ylabel('Sum of squared distances centroids')
plt.xlabel('Number of Clusters')
plt.title('Elbow method to find optimal K')
plt.show()
```



4.2.2 Average silhouette

The optimal k is 2 since the average silhouette is the highest at this point.

[27]:
```
silhouettes = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=SEED)
    cluster_labels = kmeans.fit_predict(X)
    silhouettes.append(silhouette_score(X, cluster_labels))
plt.plot(list(range(2, 11)), silhouettes, '-o')
plt.ylabel('Average Silhouette Score')
plt.xlabel('Number of Clusters')
plt.title('Average silhouette to find optimal K')
plt.show()
```



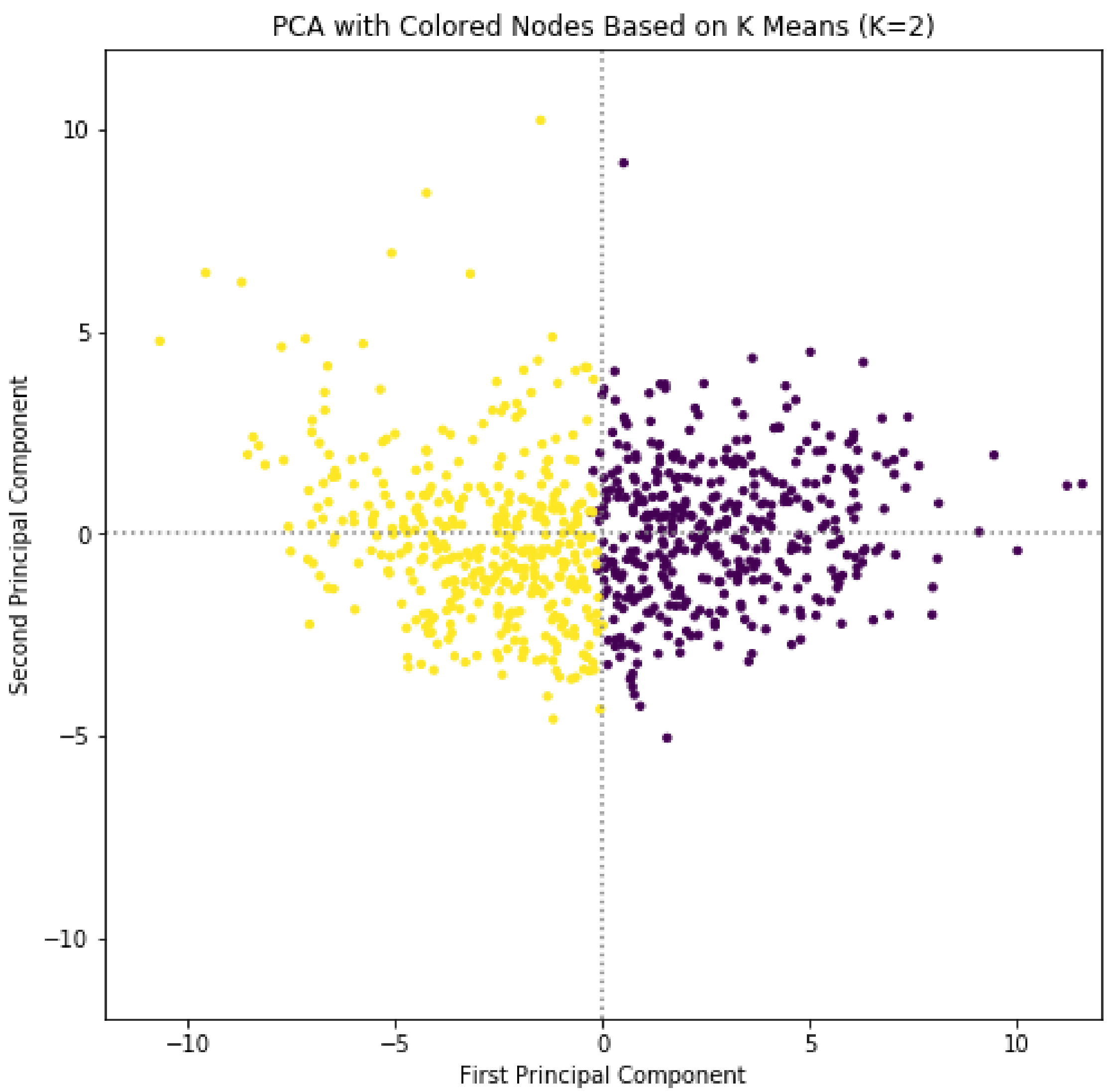
4.3 Task 11

According to the plots below, PCA with k-means separates our data set better since we can find a relatively clear boundary between clusters. However, for t-SNE, the boundary between clusters are very fuzzy and two clusters are overlapped. Since PCA holds a linear assumption for the data, one interpretation for this difference between the results of PCA and t-SNE may lie in the fact that there does exist a linear relationship pattern in the original data set, which helps PCA to perform better.

Furthermore, it seems that t-SNE and k-means are clustering the original data set in different ways: t-SNE separates the data set into a main block in the middle and several satellite clusters dispersed around it; however, k-means separates the data set into the upper and the lower parts, regardless of the satellite pattern in t-SNE's result. A possible reason that may lead to the conflict above is that t-SNE focuses too much on locality of data that may make it overlook the global pattern of the data set.

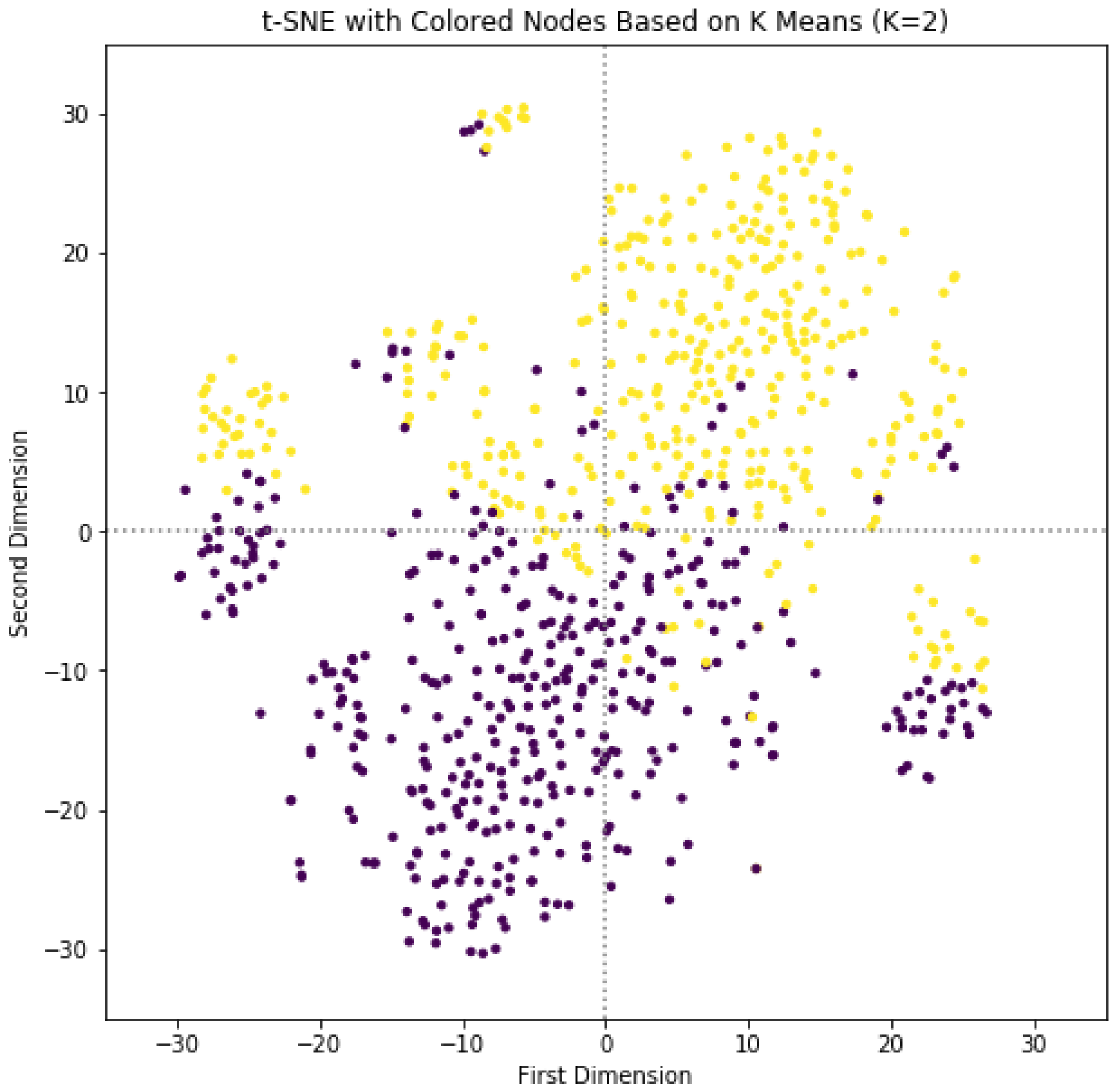
[28]:
```
# Fit k means with k=2
kmeans2 = KMeans(n_clusters=2, random_state=SEED).fit(X)

# Plot obs on PC1 and PC2
fig = plt.figure(figsize=(8, 8))

plt.xlim(-12,12)
plt.ylim(-12,12)
```

```
plt.scatter(pca_df['PC1'], pca_df['PC2'], s=11, c=kmeans2.labels_)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA with Colored Nodes Based on K Means (K=2)')

plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
plt.vlines(0,-12,12, linestyles='dotted', colors='grey')

plt.show()
```




```
[29]: fig = plt.figure(figsize=(8, 8))
```

```
plt.xlim(-35,35)
plt.ylim(-35,35)

plt.hlines(0,-35,35, linestyles='dotted', colors='grey')
plt.vlines(0,-35,35, linestyles='dotted', colors='grey')

plt.scatter(tsne_df['dim1'], tsne_df['dim2'], s=11, c=kmeans2.labels_)
plt.xlabel('First Dimension')
plt.ylabel('Second Dimension')
plt.title('t-SNE with Colored Nodes Based on K Means (K=2)')

plt.show()
```



```
[ ]:
```