Hu_Chun_Week7

March 14, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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
from gap_statistic import optimalK
```

[2]:
```
SEED = 654321
```

0.1 k-Means Clustering "By Hand"

You fielded an experiment and collected observations for 10 respondents across two features. The data are:

```
input_1 = c(5,8,7,8,3,4,2,3,4,5)
input_2 = c(8,6,5,4,3,2,2,8,9,8)
```

After inspecting your data, you suspect 3 clusters likely characterize these data, but you'd like to check your intuition. Perform k-means clustering "by hand" on these data, initializing at k = 3. Be sure to set the seed for reproducibility. Specifically:

1.(5 points) Imitate the k-means random initialization part of the algorithm by assigning each observation to a cluster at random.

[3]:
```
np.random.seed(SEED)
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])
k_label = np.random.choice(3, len(input_1), replace=True)
kmeans3 = pd.DataFrame({'input_1': input_1, 'input_2': input_2, 'k_label':␣
↪k_label})

kmeans3
```

[3]:

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

4	3	3	2
5	4	2	2
6	2	2	1
7	3	8	2
8	4	9	0
9	5	8	1

2.(5 points) Compute the cluster centroid and update cluster assignments for each observation iteratively based on spatial similarity.

[4]:
```
def assign_clusters(k, df):

    for i in range(50):
        centroids = {}
        for c in range(k):
            cent1 = df[df['k_label'] == c]['input_1'].mean()
            cent2 = df[df['k_label'] == c]['input_2'].mean()
            centroids[c] = (cent1, cent2)

        new_labels = []
        for index, row in df.iterrows():
            min_distance = 10
            for key, value in centroids.items():
                distance = ((row['input_1'] - value[0]) ** 2 + (row['input_2']␣
↪- value[1])**2)**0.5
                if distance < min_distance:
                    min_distance = distance
                    new_k = key
            new_labels.append(new_k)
        df['k_label'] = new_labels

    return df
```

[5]:
```
kmean3 = assign_clusters(3, kmeans3)
kmean3
```

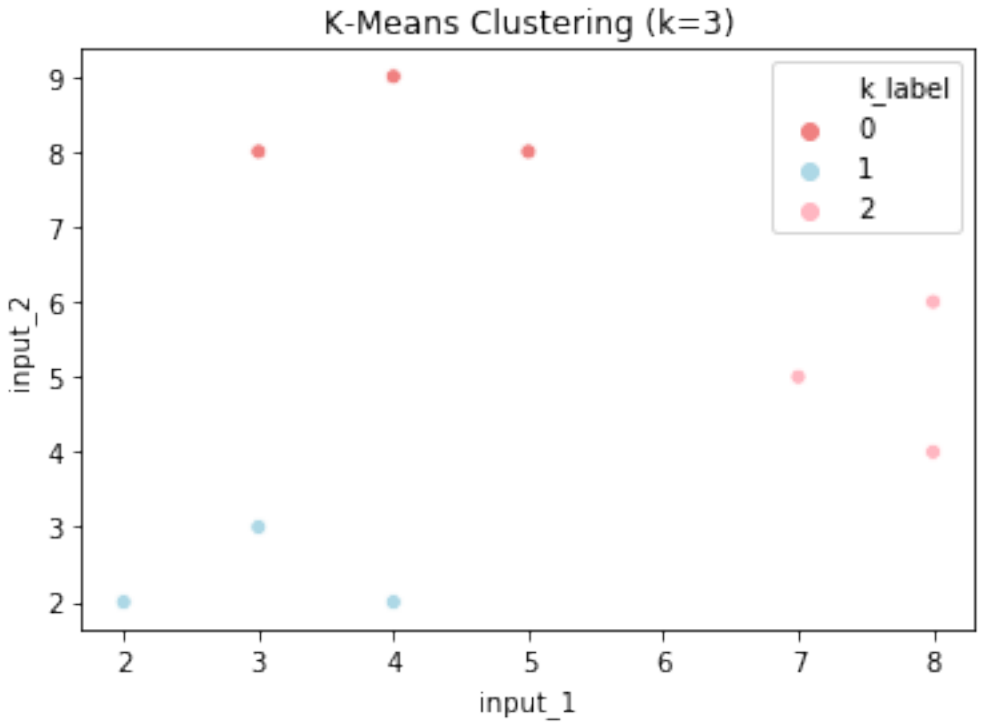
[5]:

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

3.(5 points) Present a visual description of the final, converged (stopped) cluster assignments.

[6]:
```
sns.scatterplot(x='input_1', y='input_2', hue='k_label', palette=['lightcoral',␣
↪'lightblue', 'lightpink'], data=kmeans3).set_title('K-Means Clustering␣
↪(k=3)')
```

[6]:
```
Text(0.5, 1.0, 'K-Means Clustering (k=3)')
```



4.(5 points) Now, repeat the process, but this time initialize at `k = 2` and present a final cluster assignment visually next to the previous search at `k = 3`.

[7]:
```
np.random.seed(SEED)
k_label = np.random.choice(2, len(input_1), replace=True)
kmeans2 = pd.DataFrame({'input_1': input_1, 'input_2': input_2, 'k_label':␣
↪k_label})

kmeans2
```

[7]:

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

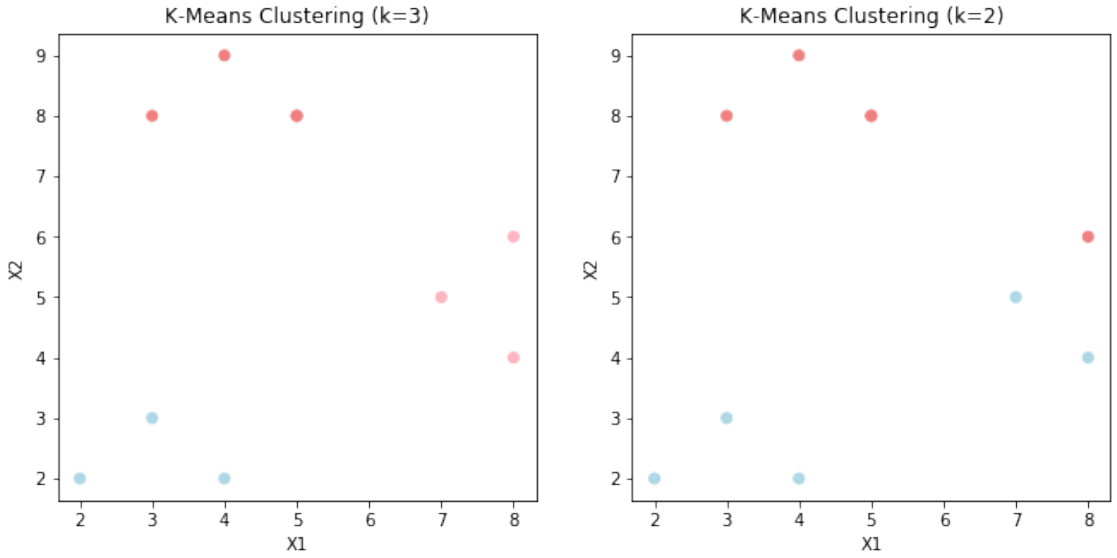
```
9        5        8        1
```

[8]:
```
kmeans2 = assign_clusters(2, kmeans2)
kmeans2
```

[8]:
```
   input_1  input_2  k_label
0        5        8        0
1        8        6        0
2        7        5        1
3        8        4        1
4        3        3        1
5        4        2        1
6        2        2        1
7        3        8        0
8        4        9        0
9        5        8        0
```

[9]:
```
fig, axs = plt.subplots(1, 2, figsize=(11,5))
colormap = np.array(['lightcoral', 'lightblue', 'lightpink'])

axs[0].scatter(kmeans3['input_1'], kmeans3['input_2'],␣
 ↪c=colormap[kmeans3['k_label']])
axs[0].set_xlabel('X1')
axs[0].set_ylabel('X2')
axs[0].set_title('K-Means Clustering (k=3)')

axs[1].scatter(kmeans2['input_1'], kmeans2['input_2'],␣
 ↪c=colormap[kmeans2['k_label']])
axs[1].set_xlabel('X1')
axs[1].set_ylabel('X2')
axs[1].set_title('K-Means Clustering (k=2)')
```

[9]:
```
Text(0.5, 1.0, 'K-Means Clustering (k=2)')
```



5.(10 points) Did your initial hunch of 3 clusters pan out, or would other values of k, like 2, fit these data better? Why or why not?

According to the plots, 3 clusters would fit the data better than 2 clusters. In k-means clustering with k=3, the within-cluster distance is minimized and the distance between clusters is maximized. While in 2 clusters, the two blue points on the right are apparently closer to the pink point, but they are are assigned to different clusters. This maximizes the distance within the cluster, which is against our purpose.

0.2 Application

0.2.1 dimension reduction

6.(15 points) Perform PCA on the dataset and plot the observations on the first and second principal components. Describe your results, e.g.,

What variables appear strongly correlated on the first principal component? What about the second principal component?

[10]:
```
wiki = pd.read_csv("wiki.csv")
features = wiki.columns
wiki = StandardScaler().fit_transform(wiki)
wiki
```

[10]:
```
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]])
```

[11]:
```
pca = PCA(n_components=2, random_state=SEED)
pcaDF = pd.DataFrame(data=pca.fit(wiki).components_.T, index=features,␣
 ↪columns=['PC1', 'PC2'])
pcaDF
```

[11]:

	PC1	PC2
age	-0.021805	0.088409
gender	-0.035086	-0.149489
phd	-0.030501	0.030414
yearsexp	-0.034190	0.062368
userwiki	0.081363	0.134372
pu1	0.192827	0.008277
pu2	0.190588	0.017673
pu3	0.210863	0.028776
peu1	0.061228	-0.271746
peu2	0.113719	-0.222371
peu3	0.100219	-0.068455
enj1	0.145666	-0.150999
enj2	0.131110	-0.227594
qu1	0.178057	-0.038127
qu2	0.163778	-0.066427
qu3	0.157956	-0.033473
qu4	-0.060797	-0.103477
qu5	0.183365	0.010903
vis1	0.171153	-0.025218
vis2	0.114559	-0.056225
vis3	0.175351	0.197639
im1	0.160432	0.111104
im2	0.077810	-0.059773
im3	0.160803	0.044000
sa1	0.121658	-0.229926
sa2	0.117590	-0.226761
sa3	0.120376	-0.242326
use1	0.181477	0.197818
use2	0.147852	0.218613
use3	0.218809	0.155153
use4	0.214558	0.160869
use5	0.206539	0.029819
pf1	0.102338	0.114378
pf2	0.103448	0.018605
pf3	0.109632	0.094177
jr1	0.080867	-0.136968
jr2	0.062216	-0.106296

```
bi1                                0.226193  0.056373
bi2                                0.230924  0.083429
inc1                               0.104667 -0.245432
inc2                               0.095802 -0.202002
inc3                               0.081402 -0.220978
inc4                               0.089707 -0.202014
exp1                               0.208592  0.070548
exp2                               0.195043 -0.029563
exp3                               0.144023 -0.126417
exp4                               0.099873  0.228482
exp5                               0.110628  0.076099
domain_Sciences                    0.021982 -0.014574
domain_Health.Sciences            -0.017158 -0.015531
domain_Engineering_Architecture    0.051309  0.171532
domain_Law_Politics               -0.094775 -0.014871
uoc_position_Associate             0.010922  0.013099
uoc_position_Assistant             0.007123  0.002360
uoc_position_Lecturer             -0.018041 -0.023594
uoc_position_Instructor            0.004251 -0.003783
uoc_position_Adjunct              -0.007849 -0.005306
```

[12]:
```
pcaDF.nlargest(5,'PC1')
```

[12]:
```
           PC1       PC2
bi2   0.230924  0.083429
bi1   0.226193  0.056373
use3  0.218809  0.155153
use4  0.214558  0.160869
pu3   0.210863  0.028776
```

The top 5 features that are strongly correlated to PC1: bi2, bi1, use3, use4, and pu3. The direction of correlation is positive.

[13]:
```
pcaDF.nsmallest(5,'PC2')
```

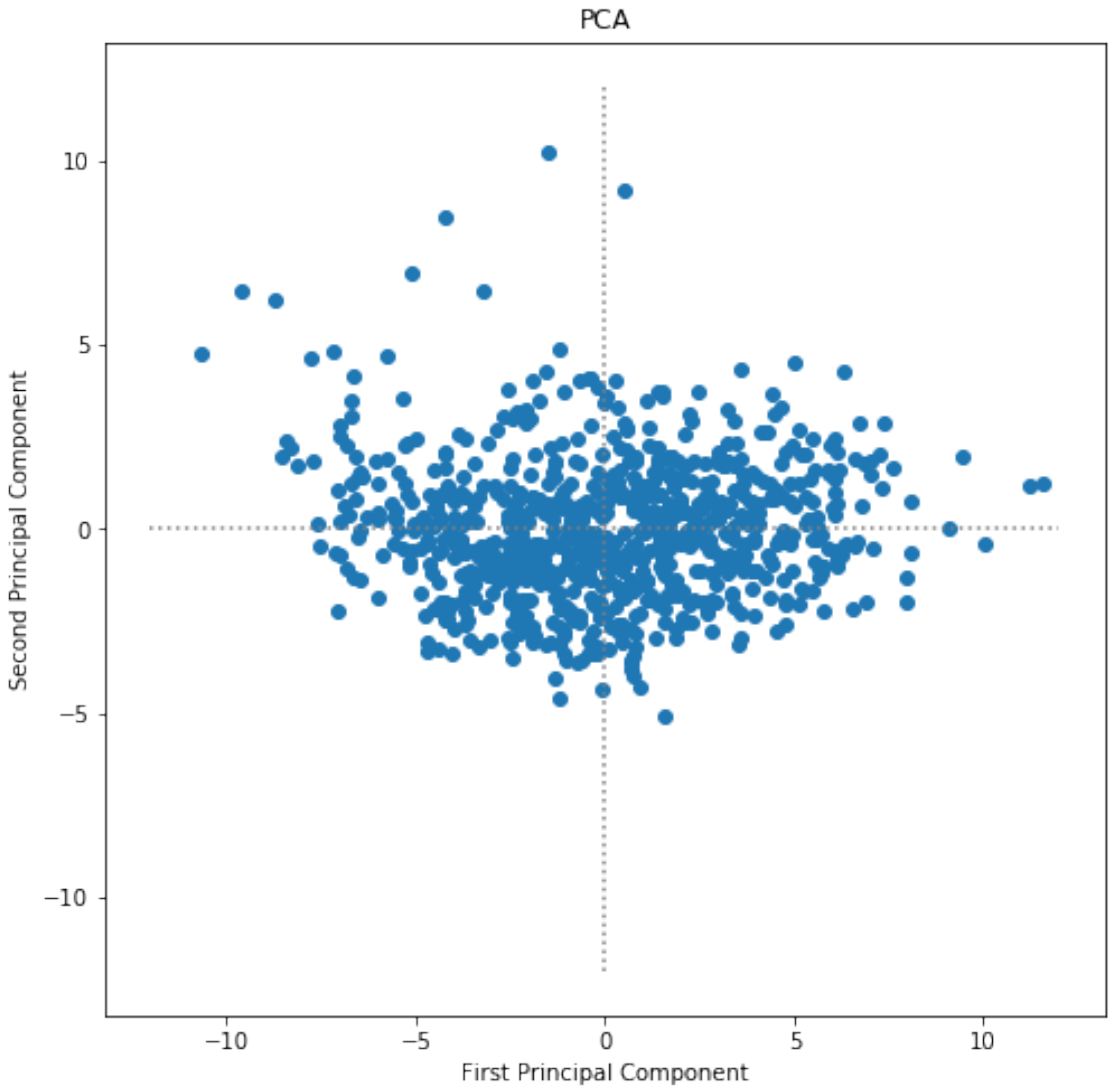
[13]:
```
           PC1       PC2
peu1  0.061228 -0.271746
inc1  0.104667 -0.245432
sa3   0.120376 -0.242326
sa1   0.121658 -0.229926
enj2  0.131110 -0.227594
```

The top 5 features that are strongly correlated to PC2: peu1, inc1, sa3, sa1, and enj2. The direction of correlation is negative.

[14]:
```
pca_DF = pd.DataFrame(data = pca.fit_transform(wiki), columns = ['PC1', 'PC2'])

plt.figure(figsize=(8, 8))
plt.scatter(pca_DF['PC1'], pca_DF['PC2'])
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')
```

[14]:
```
<matplotlib.collections.LineCollection at 0x12c5dd550>
```



7.(5 points) Calculate the proportion of variance explained (PVE) and the cumulative PVE for all the principal components. Approximately how much of the variance is explained by the first two principal components?

[15]:
```
pca.explained_variance_ratio_[0]
```

[15]:
```
0.2281062778566341
```

PVE: The first principal component explained 22.81% of the variance.

[16]:
```
pca.explained_variance_ratio_[1]
```

[16]:
```
0.06372474349228859
```

PVE: The second principal component explained 6.37% of the variance.

[17]:
```
sum(pca.explained_variance_ratio_)
```

[17]:
```
0.2918310213489227
```

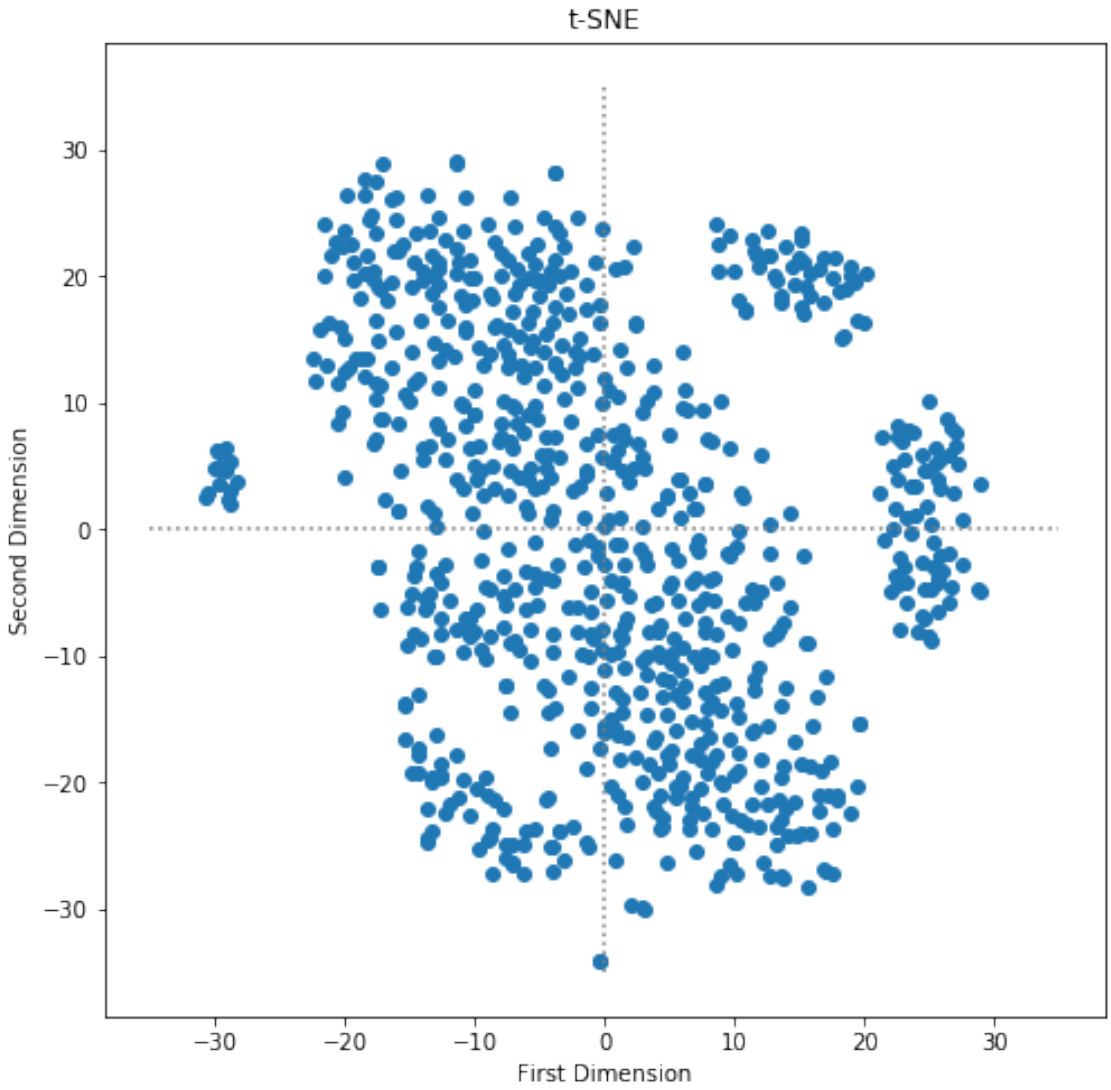
Cumulative PVE: The first two principal components explained 29.16% of the variance. The explained variance is not high, suggesting that there are a few other components that could explain our data well.

8.(10 points) Perform t-SNE on the dataset and plot the observations on the first and second dimensions. Describe your results.

[18]:
```
tsne = TSNE(n_components=2, random_state=SEED)
tsneDF = pd.DataFrame(data = tsne.fit_transform(wiki), columns = ['dim1',␣
↪'dim2'])
```

[19]:
```
fig = plt.figure(figsize=(8, 8))
plt.scatter(tsneDF['dim1'], tsneDF['dim2'])
plt.xlabel('First Dimension')
plt.ylabel('Second Dimension')
plt.title('t-SNE')

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

[19]:
```
<matplotlib.collections.LineCollection at 0x12c70d278>
```



Since t-SNE captures the structure of neighboring points in the input space and visualizes in the low-dimensional space, our plot reveals information about the relative distances between low dimensional points. From the plot above, there is a main cluster of points, suggesting that a majority of our data are similar to each other. There are also several small clustrs around – these are points that tend to be further away in the input space.

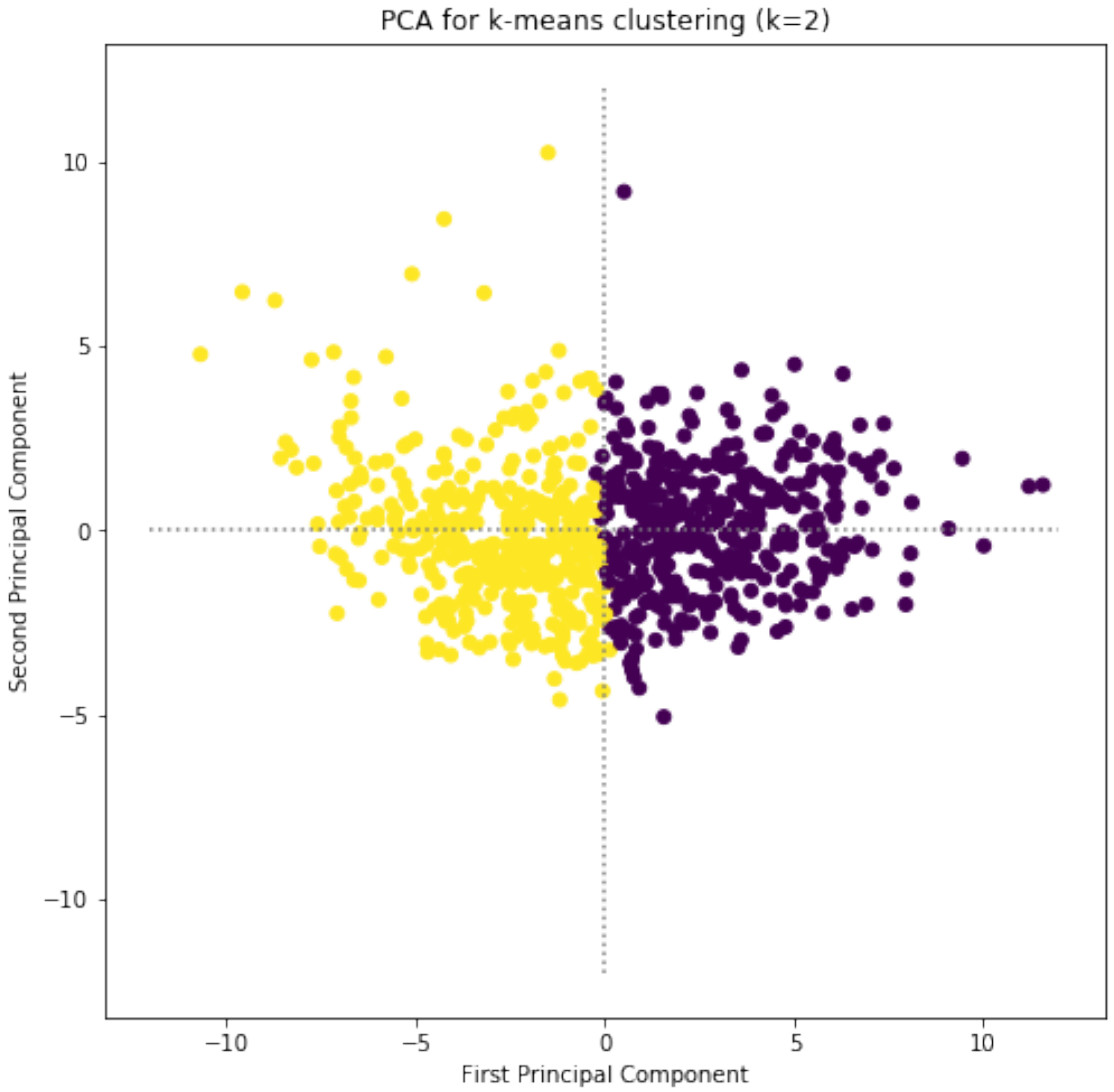
0.2.2 clustering

9.(15 points) Perform k-means clustering with $k = 2, 3, 4$. Be sure to scale each feature (i.e.,mean zero and standard deviation one). Plot the observations on the first and second principal components from PCA and color-code each observation based on their cluster membership. Discuss your results.

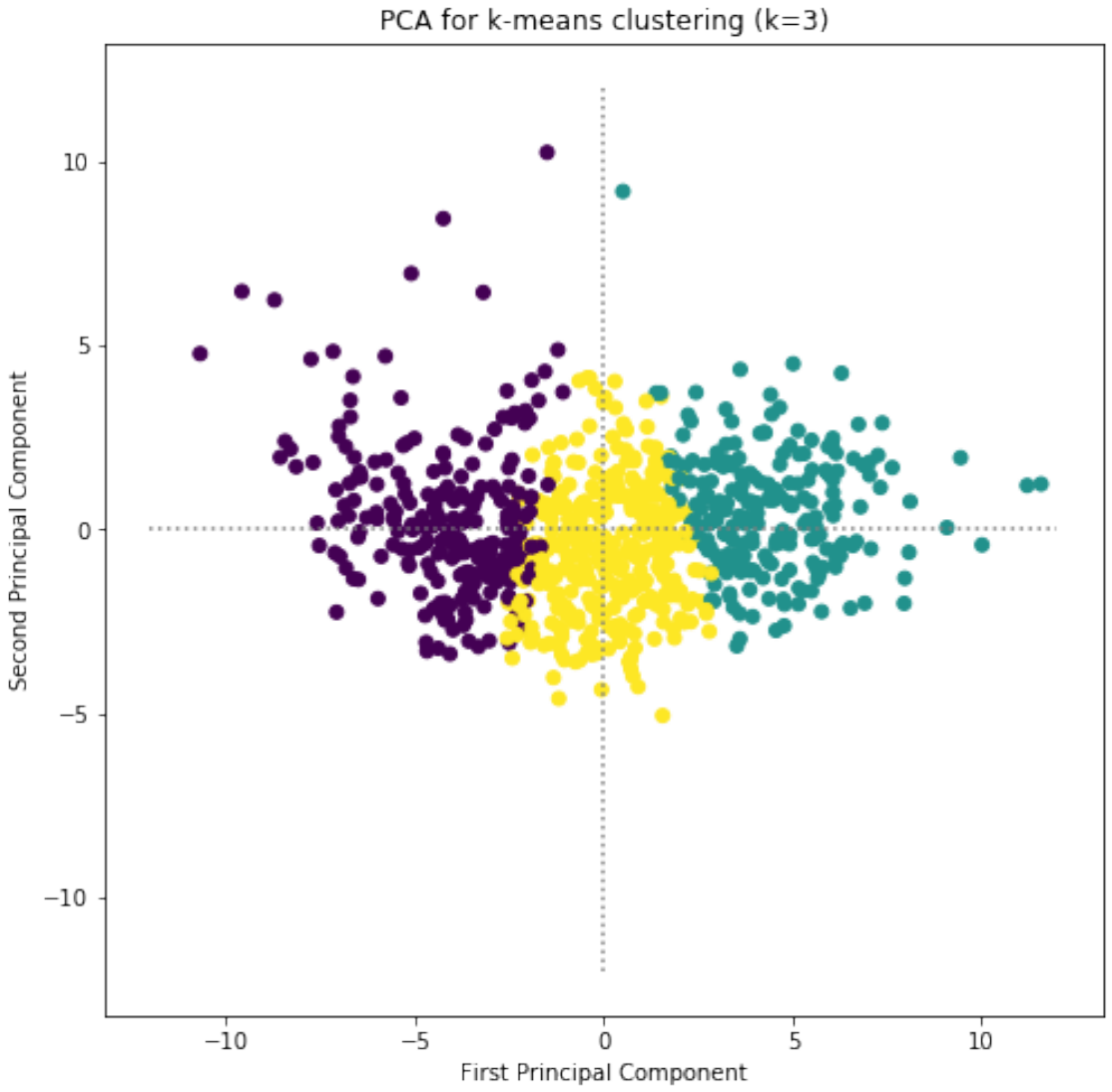
[20]:
```
def plot_kmeans(k):

```

```
    model = KMeans(n_clusters=k, random_state=SEED).fit(wiki)

    plt.figure(figsize=(8, 8))
    plt.scatter(pca_DF['PC1'], pca_DF['PC2'], c=model.labels_)

    plt.xlabel('First Principal Component')
    plt.ylabel('Second Principal Component')
    plt.title(f'PCA for k-means clustering (k={k})')

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

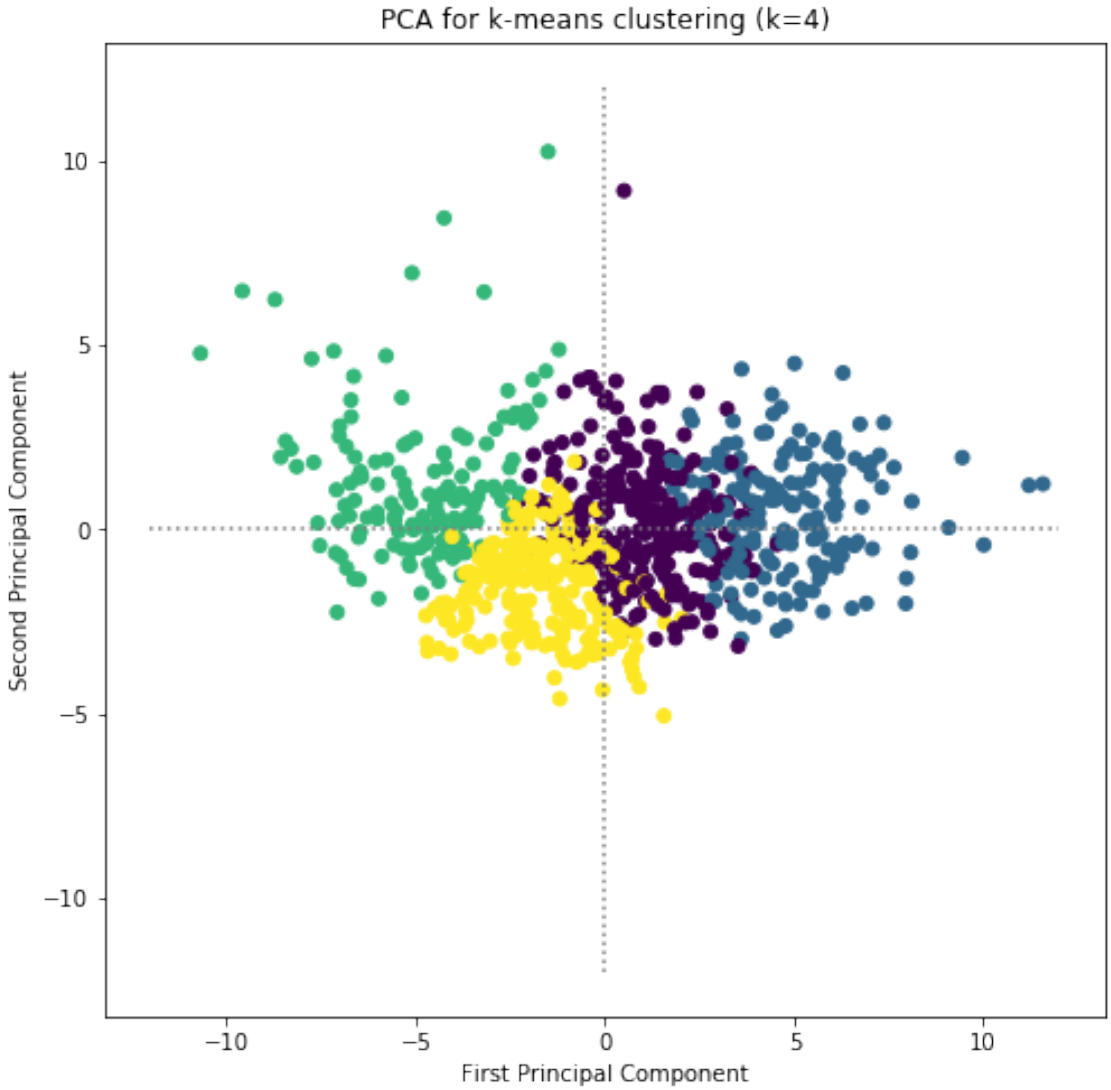
[21]:
```
plot_kmeans(2)
```



[22]:
```
plot_kmeans(3)
```



[23]:
```
plot_kmeans(4)
```



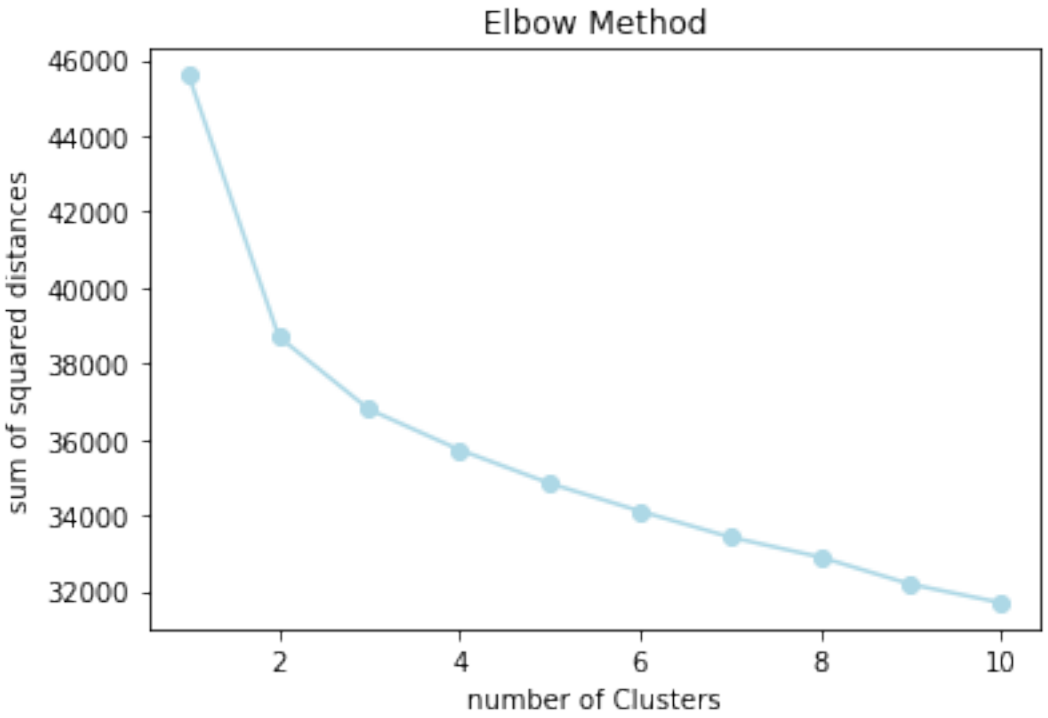
The k-means clusterings with k=2,3,4 can be primarily explained by the first principal component. At k=2 and k=3, the data produces a clear boundary between each cluster, and the influence of two principal components is equally divided in data. The boundary in k=4 tends to overlap with each other, suggesting that other factors besides PC1 and PC2 have an impact on our data.

10.(10 points) Use the elbow method, average silhouette, and/or gap statistic to identify the optimal number of clusters based on *k*-means clustering with scaled features.

[24]:
```
inertias = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=SEED).fit(wiki)
    inertias.append(kmeans.inertia_)
plt.plot(range(1, 11), inertias, '-o', color='lightblue')
plt.ylabel('sum of squared distances')
plt.xlabel('number of Clusters')
```

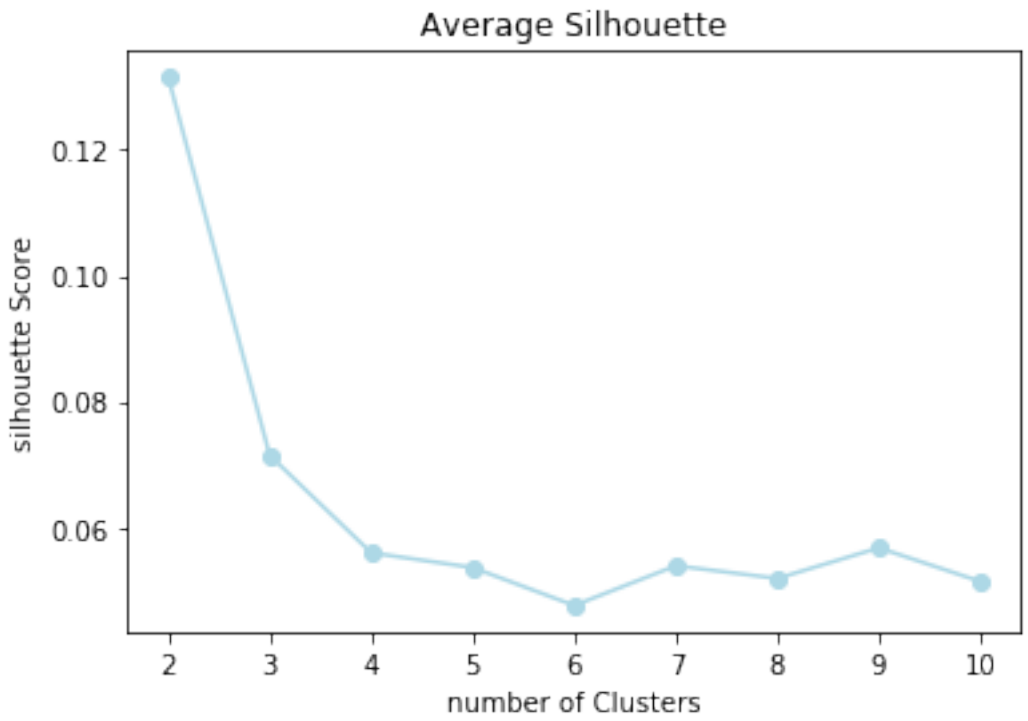
```
plt.title('Elbow Method')
```

[24]:
```
Text(0.5, 1.0, 'Elbow Method')
```



[25]:
```
silhouettes = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=SEED)
    silhouettes.append(silhouette_score(wiki, kmeans.fit_predict(wiki)))
plt.plot(range(2, 11), silhouettes, '-o', color='lightblue')
plt.ylabel('silhouette Score')
plt.xlabel('number of Clusters')
plt.title('Average Silhouette')
```

[25]:
```
Text(0.5, 1.0, 'Average Silhouette')
```



[26]:
```
opt = optimalK.OptimalK()
n_clusters = opt(wiki, cluster_array=np.arange(2, 11))
print('The optimal number of clusters is: ', n_clusters)
```

```
The optimal number of clusters is:  8
```

Both elbow method and silhouette score suggest that the optimal number of clusters is 2, and so we will choose k=2.

11.(15 points) Visualize the results of the optimal $\hat{k}$-means clustering model. First use the first and second principal components from PCA, and color-code each observation based on their cluster membership. Next use the first and second dimensions from t-SNE, and color-code each observation based on their cluster membership. Describe your results. How do your interpretations differ between PCA and t-SNE?

[27]:
```
fig, axs = plt.subplots(1, 2, figsize=(11,5))
k2 = KMeans(n_clusters=2, random_state=SEED).fit(wiki)
colormap = np.array(['lightcoral', 'lightblue'])

axs[0].scatter(pca_DF['PC1'], pca_DF['PC2'], c=colormap[k2.labels_])
axs[0].set_xlabel('X1')
axs[0].set_ylabel('X2')
axs[0].set_title('K-Means Clustering (k=2)')
axs[0].hlines(0, -12, 12, linestyles='dotted', colors='grey')
axs[0].vlines(0, -12, 12, linestyles='dotted', colors='grey')
```

```
axs[1].scatter(tsneDF['dim1'], tsneDF['dim2'], c=colormap[k2.labels_])
axs[1].set_xlabel('X1')
axs[1].set_ylabel('X2')
axs[1].set_title('T-SNE Plot (dim=2)')
axs[1].hlines(0,-35,35, linestyles='dotted', colors='grey')
axs[1].vlines(0,-35,35, linestyles='dotted', colors='grey')
```

[27]: `<matplotlib.collections.LineCollection at 0x131c14e48>`

From the plots above, we can see that PCA produces a better separation of the data points, while in t-SNE, the boundary of clusters tends to overlap with each other. t-SNE tries to map only local/neighbors as close to each other and ignores the global structure, which leads to fuzzy boundary and several small clusters around the main cluster. On the other hand, PCA preserves the larger distances between points, and is better at visualizing how our data relates with respect to the principal components.