k-Means Clustering “By Hand”

In [24]:

```
import pandas as pd
import numpy as np
import math
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
```

In [67]:

```
def optimalK(data, nrefs=3, maxClusters=15):
    """
    Calculates KMeans optimal K using Gap Statistic from Tibshir
ani, Walther, Hastie
    Params:
        data: ndarry of shape (n_samples, n_features)
        nrefs: number of sample reference datasets to create
        maxClusters: Maximum number of clusters to test for
    Returns: (gaps, optimalK)
    """
    gaps = np.zeros((len(range(1, maxClusters)),))
    resultsdf = pd.DataFrame({'clusterCount':[], 'gap':[]})
    for gap_index, k in enumerate(range(1, maxClusters)):

        # Holder for reference dispersion results
        refDisps = np.zeros(nrefs)

        # For n references, generate random sample and perform k
means getting resulting dispersion of each loop
        for i in range(nrefs):

            # Create new random reference set
            randomReference = np.random.random_sample(size=data.
shape)
```

```
            # Fit to it
            km = KMeans(k)
            km.fit(randomReference)

            refDisp = km.inertia_
            refDisps[i] = refDisp

        # Fit cluster to original data and create dispersion
        km = KMeans(k)
        km.fit(data)

        origDisp = km.inertia_

        # Calculate gap statistic
        gap = np.log(np.mean(refDisps)) - np.log(origDisp)

        # Assign this loop's gap statistic to gaps
        gaps[gap_index] = gap

        resultsdf = resultsdf.append({'clusterCount':k, 'gap':ga
p}, ignore_index=True)

    return (gaps.argmax() + 1, resultsdf)  # Plus 1 because inde
x of 0 means 1 cluster is optimal, index 2 = 3 clusters are opti
mal
```

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

In [26]:

```
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])
# Assign initial label
np.random.seed(666)
k3 = pd.DataFrame({'input_1': input_1, 'input_2': input_2})
init_labels = np.random.choice(3, 10, replace=True)
k3['k_label'] = init_labels
k3
```

Out[26]:

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

1. (5 points) Compute the cluster centroid and update cluster assignments for each observation iteratively based on spatial similarity.

In [27]:

```
def compute_clusters(num_k, df):
    for i in range(50):
        centroids = {}
        for c in range(num_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 k,v in centroids.items():
                distance = math.sqrt((row['input_1'] - v[0]) **
2 + (row['input_2'] - v[1])**2)
                if distance < min_distance:
                    min_distance = distance
                    new_k = k
            new_labels.append(new_k)
        df['k_label'] = new_labels
    return df
```

In [28]:

```
k3 = compute_clusters(3, k3)
k3
```

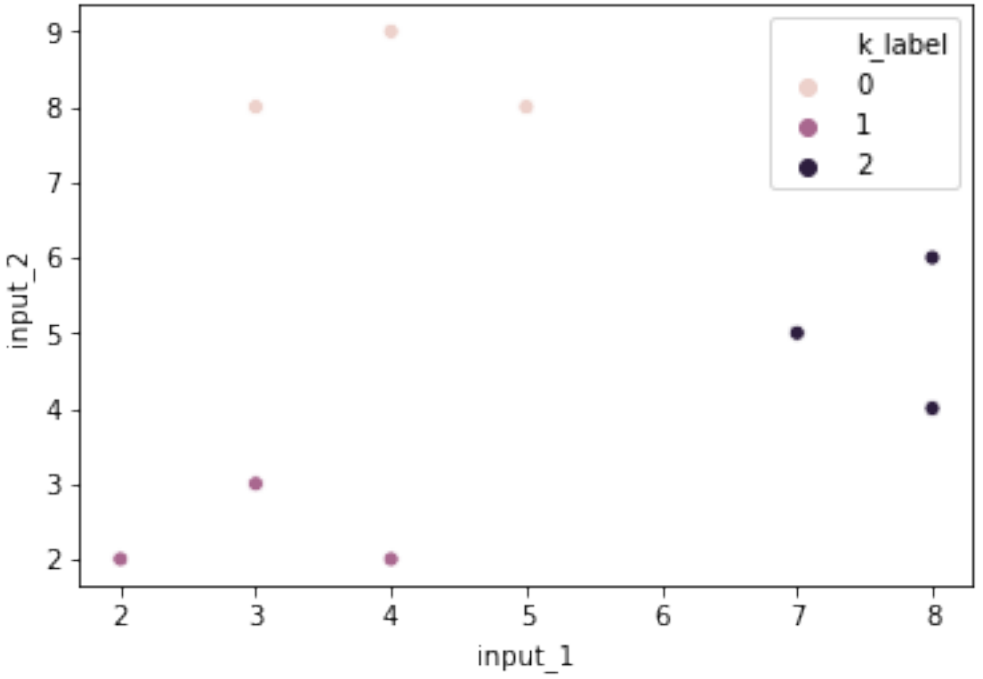
Out[28]:

	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

1. (5 points) Present a visual description of the final, converged (stopped) cluster assignments.

In [29]:

```
sns.scatterplot(x='input_1', y='input_2', hue='k_label', data=k3
);
```



1. (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.

In [30]:

```
np.random.seed(666)
k_label = np.random.choice(2, len(input_1), replace=True)
k2 = pd.DataFrame({'input_1': input_1, 'input_2': input_2, 'k_la
bel':k_label})
k2
```

Out[30]:

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

In [31]:

```
k2 = compute_clusters(2, k2)
k2
```

Out[31]:

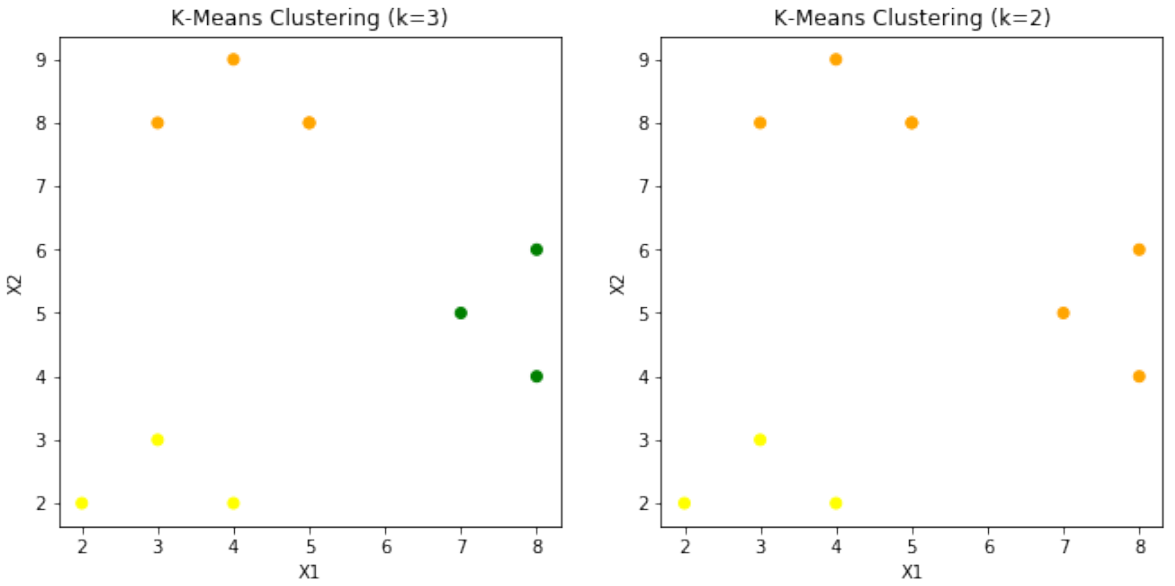
	input_1	input_2	k_label
0	5	8	0
1	8	6	0
2	7	5	0
3	8	4	0
4	3	3	1
5	4	2	1
6	2	2	1
7	3	8	0
8	4	9	0
9	5	8	0

In [38]:

```
fig, axs = plt.subplots(1, 2, figsize=(11,5))
colormap = np.array(['orange', 'yellow', 'green'])
axs[0].scatter(k3['input_1'], k3['input_2'],c=colormap[k3['k_lab
el']])
axs[0].set_xlabel('X1')
axs[0].set_ylabel('X2')
axs[0].set_title('K-Means Clustering (k=3)')
axs[1].scatter(k2['input_1'], k2['input_2'],c=colormap[k2['k_lab
el']])
axs[1].set_xlabel('X1')
axs[1].set_ylabel('X2')
axs[1].set_title('K-Means Clustering (k=2)')
```

Out[38]:

```
Text(0.5,1,'K-Means Clustering (k=2)')
```



1. (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?

When k=3, the K-means clustering performs better because the distance within cluster is minimized and the distance between clusters is maximized. When k=2, the orange dots on the right side and the orange dots on the top are clustered into the same group, this leads to larger distance within the cluster.

Application

1. (15 points) Perform PCA on the dataset and plot the observations on the first and second principal components. Describe your results,

In [10]:

```
X = pd.read_csv("data/wiki.csv")
features = X.columns
X = StandardScaler().fit_transform(X)
X
```

Out[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]])
```

In [14]:

```
pca = PCA(n_components=2, random_state=np.random.seed(666))
pca_df = pd.DataFrame(data=pca.fit(X).components_.T, index=featu
res,columns=['PC1', 'PC2'])
pca_df
```

Out[14]:

	PC1	PC2
age	-0.021805	0.088371
gender	-0.035086	-0.149453
phd	-0.030501	0.030429
yearsexp	-0.034190	0.062344
userwiki	0.081363	0.134408
pu1	0.192827	0.008277
pu2	0.190588	0.017675
pu3	0.210863	0.028786
peu1	0.061228	-0.271749
peu2	0.113719	-0.222383
peu3	0.100219	-0.068494
enj1	0.145666	-0.151016
enj2	0.131110	-0.227613
qu1	0.178057	-0.038125
qu2	0.163778	-0.066427
qu3	0.157956	-0.033491
qu4	-0.060797	-0.103425
qu5	0.183365	0.010920
vis1	0.171153	-0.025198
vis2	0.114559	-0.056223
vis3	0.175351	0.197637
im1	0.160432	0.111096
im2	0.077810	-0.059794
im3	0.160803	0.044001
sa1	0.121658	-0.229905
sa2	0.117590	-0.226743
sa3	0.120376	-0.242307
use1	0.181477	0.197838

use1	0.181477	0.197838
use2	0.147852	0.218635
use3	0.218809	0.155159
use4	0.214558	0.160866
use5	0.206539	0.029834
pf1	0.102338	0.114348
pf2	0.103448	0.018585
pf3	0.109632	0.094148
jr1	0.080867	-0.136968
jr2	0.062216	-0.106295
bi1	0.226193	0.056385
bi2	0.230924	0.083443
inc1	0.104667	-0.245448
inc2	0.095802	-0.202041
inc3	0.081402	-0.220999
inc4	0.089707	-0.202025
exp1	0.208592	0.070549
exp2	0.195043	-0.029548
exp3	0.144023	-0.126404
exp4	0.099873	0.228494
exp5	0.110628	0.076085
domain_Sciences	0.021982	-0.014575
domain_Health.Sciences	-0.017158	-0.015437
domain_Engineering_Architecture	0.051309	0.171463
domain_Law_Politics	-0.094775	-0.014877
uoc_position_Associate	0.010922	0.013151
uoc_position_Assistant	0.007123	0.002291
uoc_position_Lecturer	-0.018041	-0.023569
uoc_position_Instructor	0.004251	-0.003773
uoc_position_Adjunct	-0.007849	-0.005310

In [15]:

```
pca_df.nlargest(5,'PC1')
```

Out[15]:

	PC1	PC2
bi2	0.230924	0.083443
bi1	0.226193	0.056385
use3	0.218809	0.155159
use4	0.214558	0.160866
pu3	0.210863	0.028786

In [16]:

```
pca_df.nsmallest(5,'PC1')
```

Out[16]:

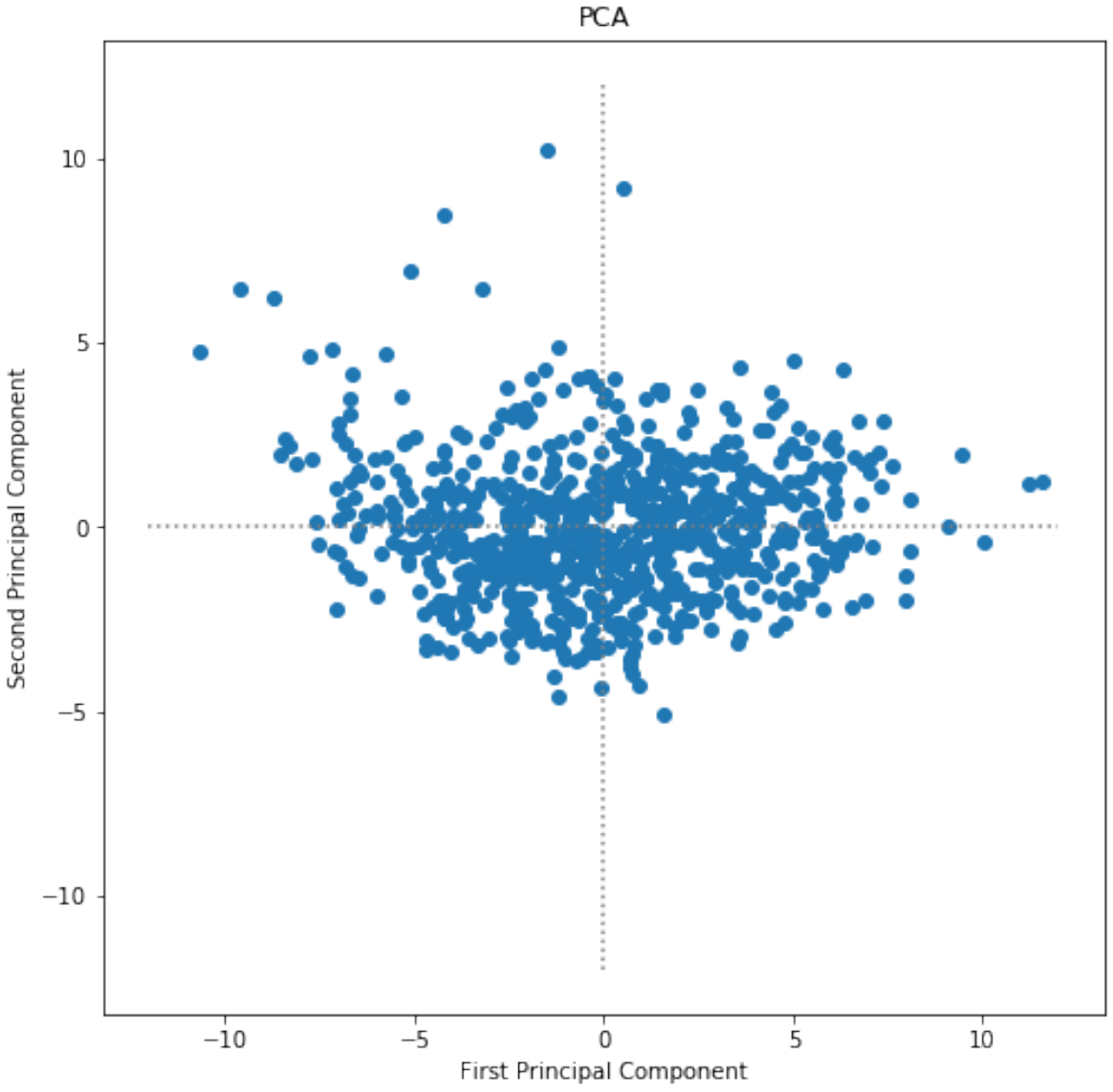
	PC1	PC2
domain_Law_Politics	-0.094775	-0.014877
qu4	-0.060797	-0.103425
gender	-0.035086	-0.149453
yearsexp	-0.034190	0.062344
phd	-0.030501	0.030429

In [17]:

```
pca_df = pd.DataFrame(data = pca.fit_transform(X), columns = ['P
C1', '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')
```

Out[17]:

```
<matplotlib.collections.LineCollection at 0x1249c3e9
0>
```



1. (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?

In [18]:

```
pca.explained_variance_ratio_[0]
```

Out[18]:

```
0.22810627785663407
```

In [19]:

```
pca.explained_variance_ratio_[1]
```

Out[19]:

```
0.06372474356547082
```

In [20]:

```
sum(pca.explained_variance_ratio_)
```

Out[20]:

```
0.2918310214221049
```

The cumulative PVE is calculated as above. This means that the first two principal components explained 29.18% of the variance, indicating that there are other components that would explain the data better.

1. (10 points) Perform t-SNE on the dataset and plot the observations on the first and second dimensions. Describe your results.

In [21]:

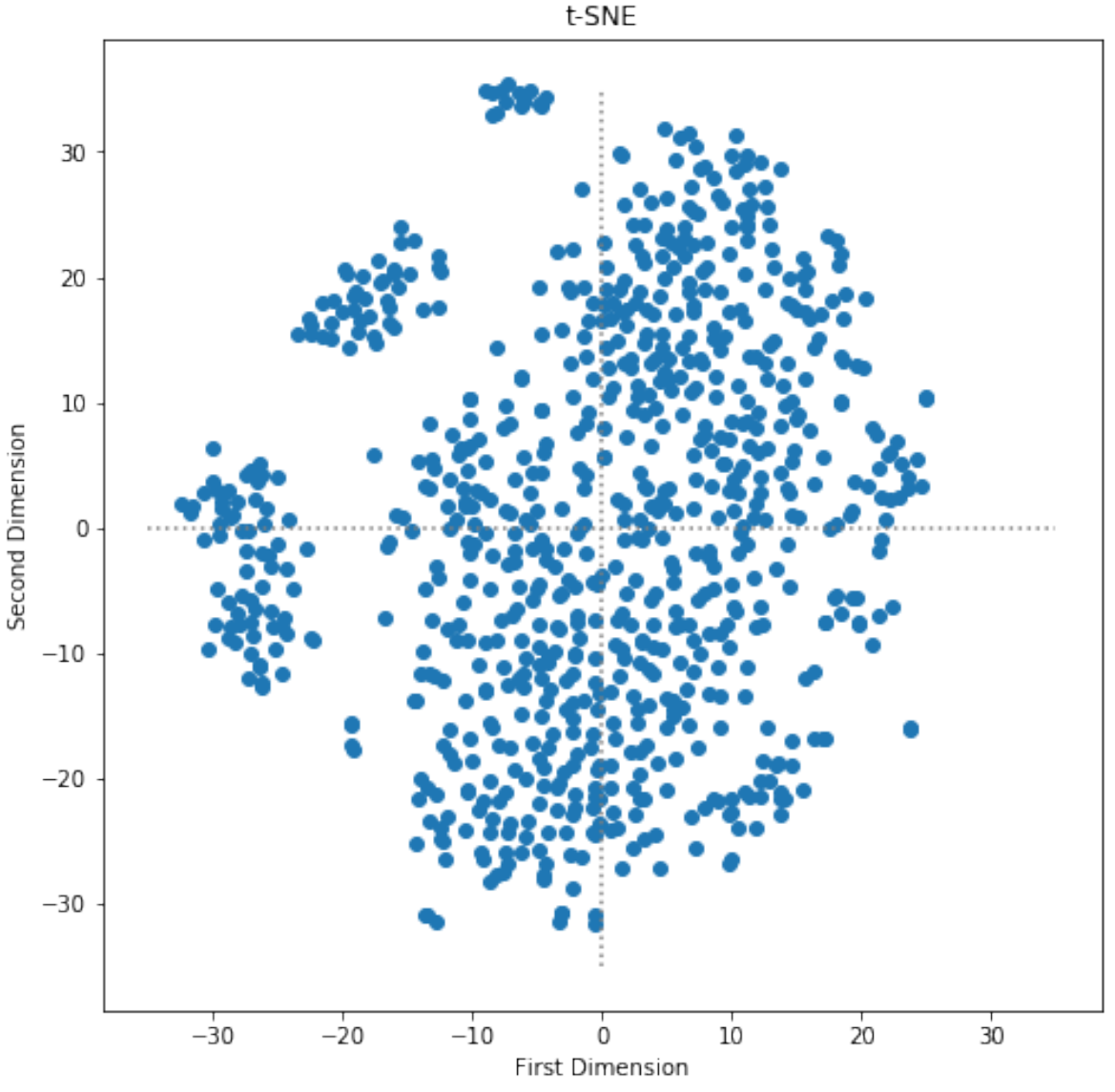
```
tsne = TSNE(n_components=2, random_state=np.random.seed(666))
tsne_df = pd.DataFrame(data = tsne.fit_transform(X),
                       columns = ['dim1','dim2'])
```

In [43]:

```
fig = plt.figure(figsize=(8, 8))
plt.scatter(tsne_df['dim1'], tsne_df['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')
```

Out[43]:

```
<matplotlib.collections.LineCollection at 0x128a70fd
0>
```



As the t-sne plot above shows, the small group of points are distributed around the majority group of points. This means that a large proportion of our data are similar.

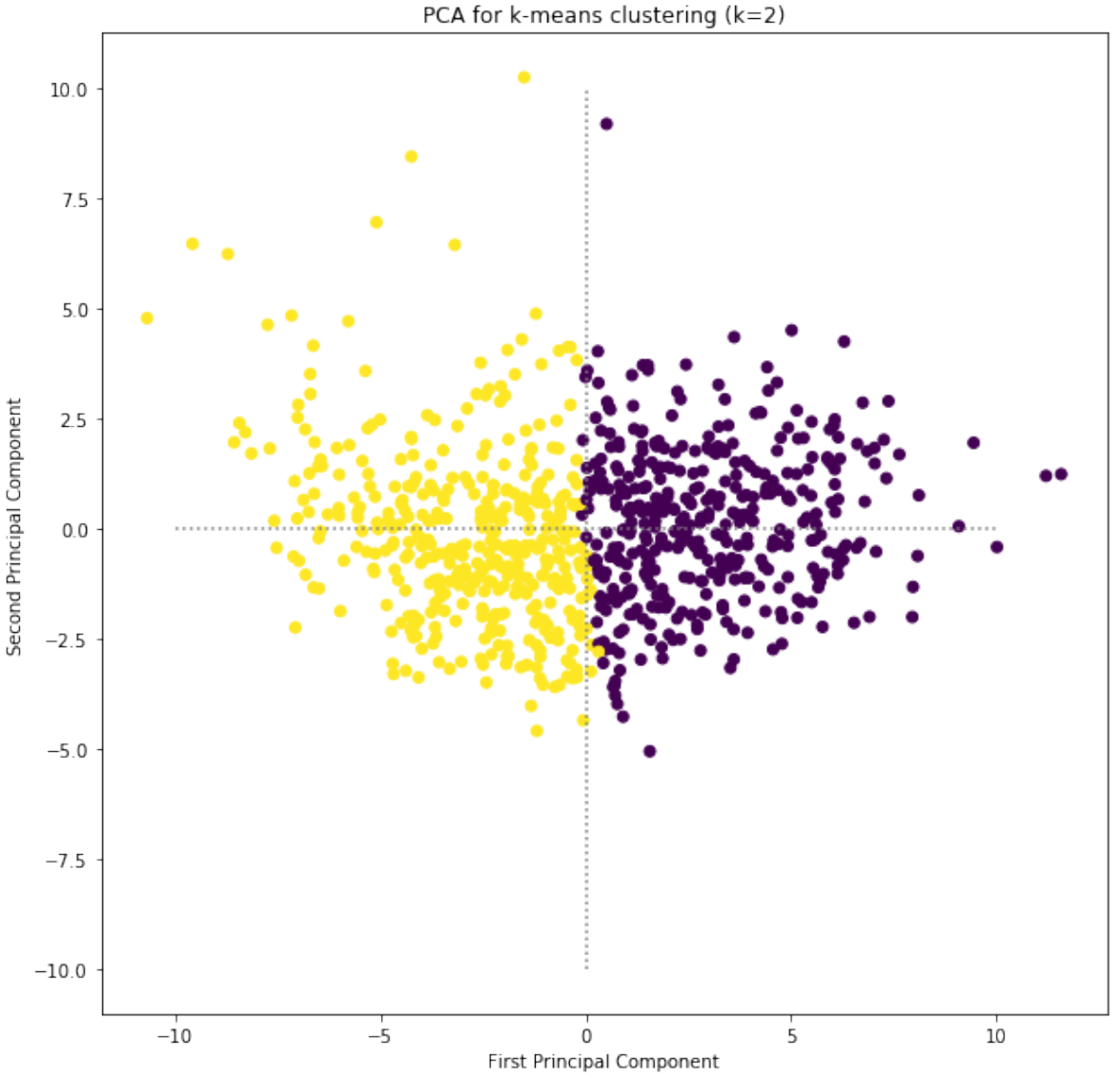
1. (15 points) Perform k-means clustering with k = 2, 3, 4. Be sure to scale each feature (i.e.,mean zero and standard deviation one). Plot the observations on the first and second principal components from PCA and color-code each observation based on their cluster membership. Discuss your results.

In [53]:

```
def plot_kmeans(k):
    model = KMeans(n_clusters=k, random_state=np.random.seed(666
)).fit(X)
    plt.figure(figsize=(10, 10))
    plt.scatter(pca_df['PC1'], pca_df['PC2'], c=model.labels_)
    plt.hlines(0, -10, 10, linestyles='dotted', colors='grey')
    plt.vlines(0, -10, 10, linestyles='dotted', colors='grey')
    plt.xlabel('First Principal Component')
    plt.ylabel('Second Principal Component')
    plt.title(f'PCA for k-means clustering (k={k})');
```

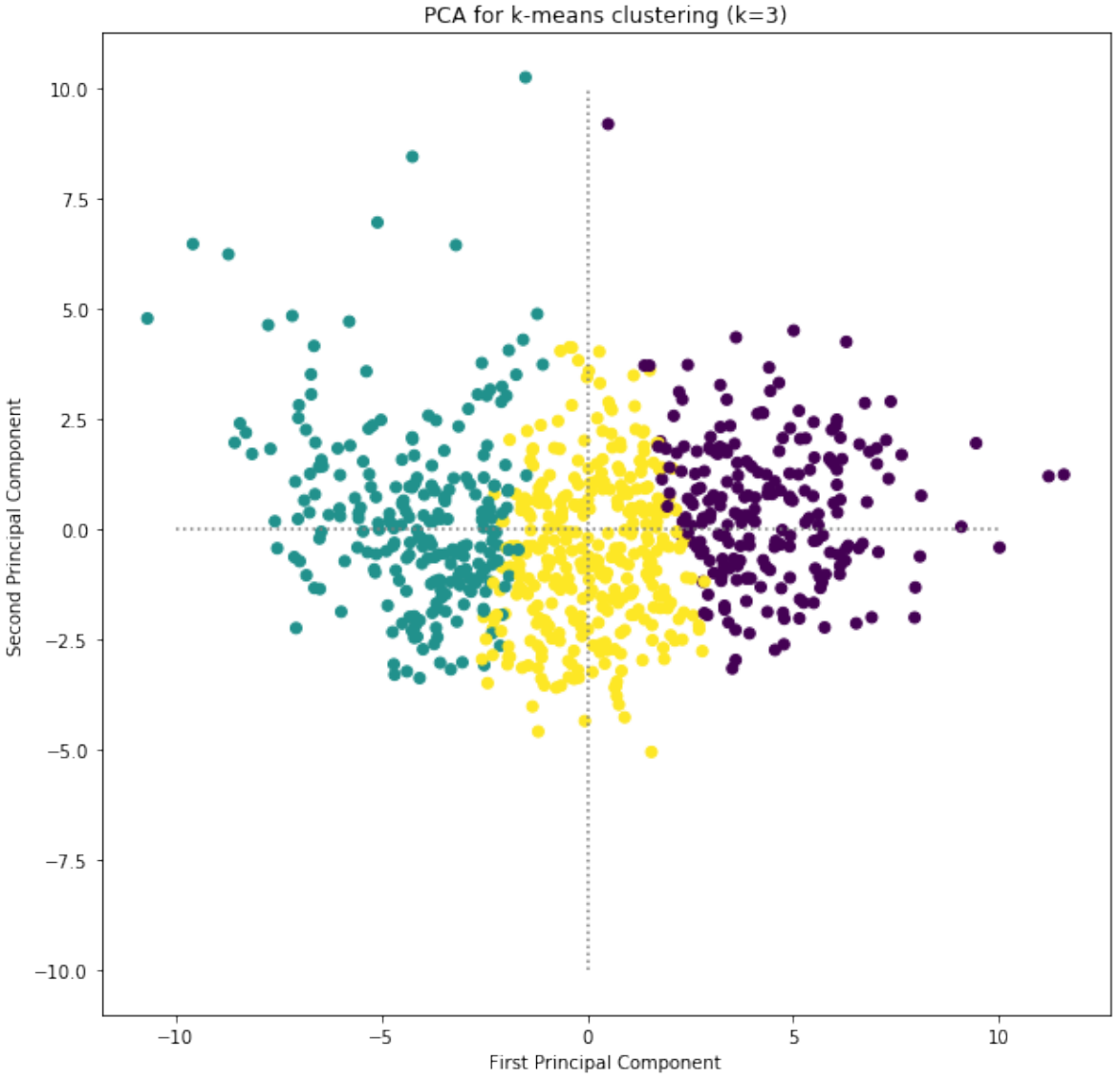
In [54]:

```
plot_kmeans(2)
```


PCA for k-means clustering (k=2)
Second Principal Component
First Principal Component
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10

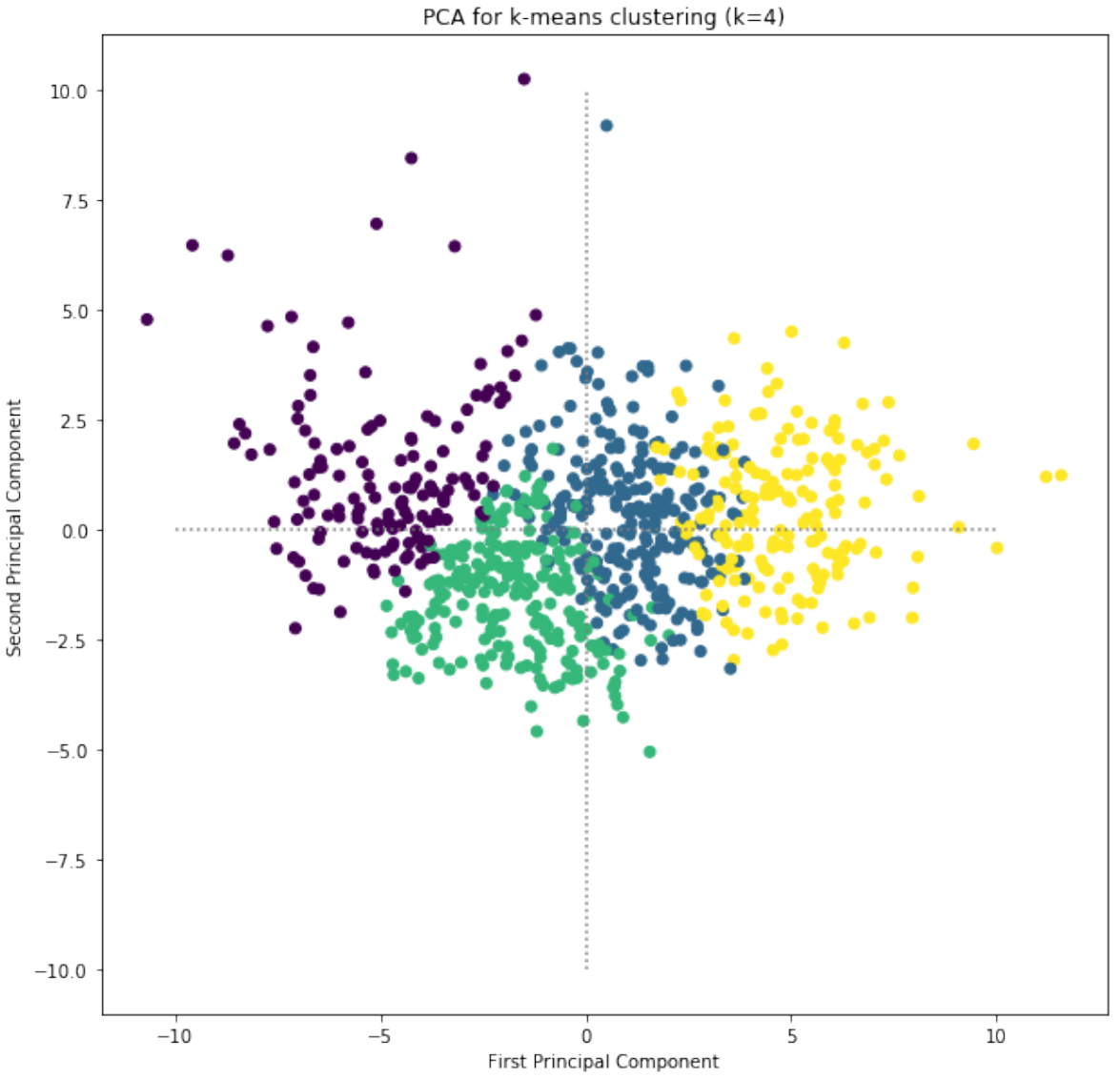
In [55]:

```
plot_kmeans(3)
```



In [56]:

```
plot_kmeans(4)
```



According to the plots above, k-means clustering with 2 clusters performs the best among these ks. When k=2, the points between clusters have a clear boundary and the data seems to be divided almost equally. Whilst in k=3 or k=4, there are obvious overlap between clusters.

1. (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.

In [ ]:

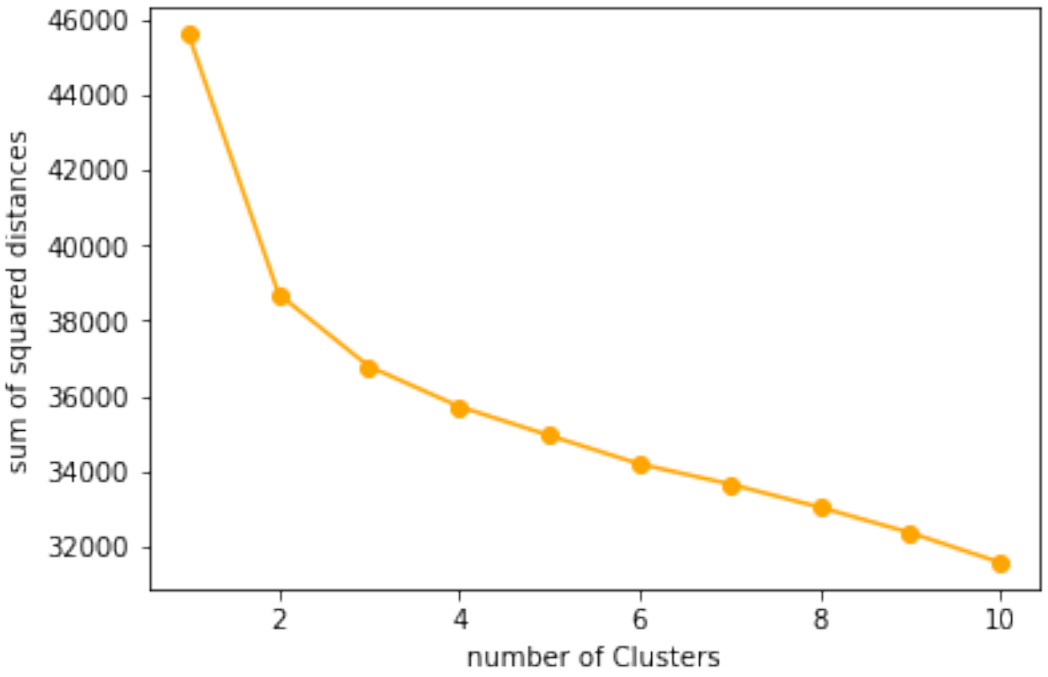
```
#Elbow
```

In [60]:

```
inertias = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=np.random.seed(666)).fit(X)
    inertias.append(kmeans.inertia_)
plt.plot(range(1, 11), inertias, '-o', color='orange')
plt.ylabel('sum of squared distances')
plt.xlabel('number of Clusters')
```

Out[60]:

```
Text(0.5,0,'number of Clusters')
```



In [ ]:

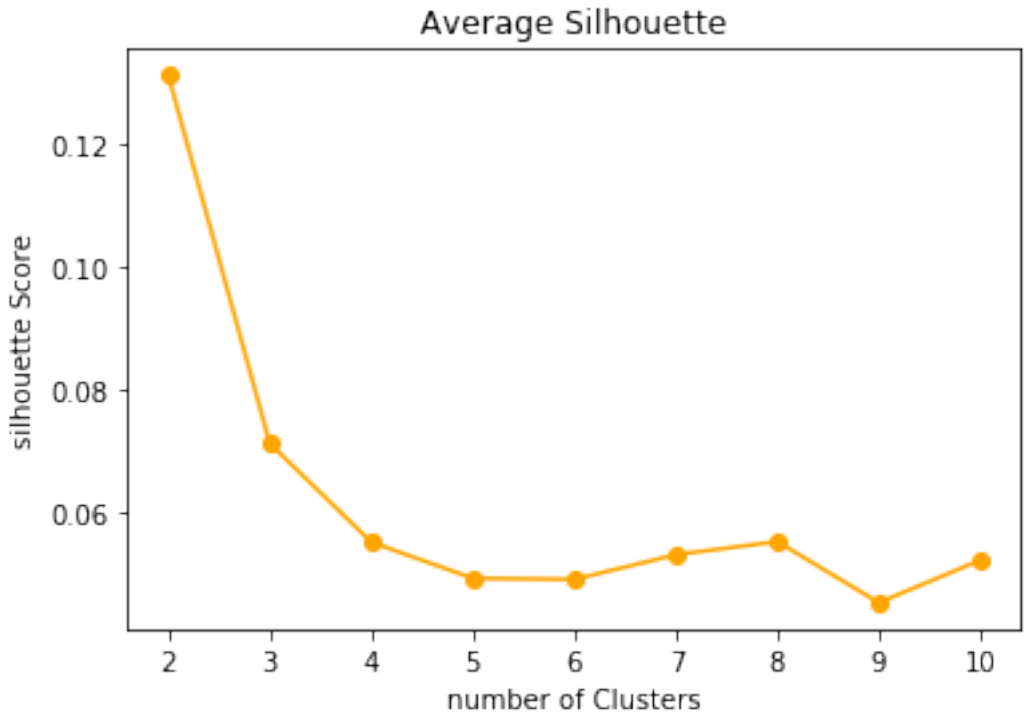
```
#Average silhouette
```

In [62]:

```
silhouettes = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=np.random.seed(66
6))
    silhouettes.append(silhouette_score(X, kmeans.fit_predict(X)
))
plt.plot(range(2, 11), silhouettes, '-o', color='orange')
plt.ylabel('silhouette Score')
plt.xlabel('number of Clusters')
plt.title('Average Silhouette')
```

Out[62]:

```
Text(0.5,1,'Average Silhouette')
```



In [ ]:

```
#Gap statistic
```

In [73]:

```
optimalK(X)
```

Out[73]:

```
(14,     clusterCount       gap
 0              1.0 -2.486717
 1              2.0 -2.341597
 2              3.0 -2.303923
 3              4.0 -2.287654
 4              5.0 -2.270340
 5              6.0 -2.255712
 6              7.0 -2.240608
 7              8.0 -2.231651
 8              9.0 -2.224691
 9             10.0 -2.220175
 10            11.0 -2.201755
 11            12.0 -2.188952
 12            13.0 -2.183336
 13            14.0 -2.180020)
```

Compare all three methods above, the optimal k is 2.

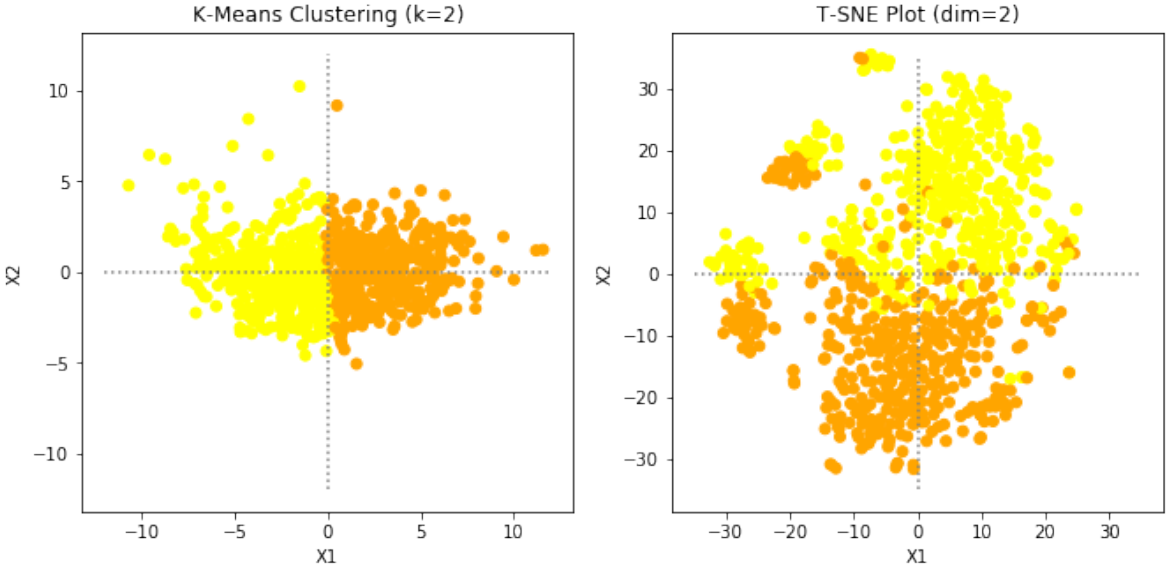
1. (15 points) Visualize the results of the optimal ˆk-means clustering model. First use the first and second principal components from PCA, and color-code each observation based on their cluster membership. Next use the first and second dimensions from t-SNE, and color-code each observation based on their cluster membership. Describe your results. How do your interpretations differ between PCA and t-SNE?

In [72]:

```
fig, axs = plt.subplots(1, 2, figsize=(11,5))
k2 = KMeans(n_clusters=2, random_state=np.random.seed(666)).fit(
X)
colormap = np.array(['orange', 'yellow'])
axs[0].scatter(pca_df['PC1'], pca_df['PC2'], c=colormap[k2.label
s_])
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(tsne_df['dim1'], tsne_df['dim2'], c=colormap[k2.l
abels_])
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')
```

Out[72]:

```
<matplotlib.collections.LineCollection at 0x124a84f9
0>
```



From the plots above, we can see that:

1) PCA performs a better separation of the data points and it preserves larger distances between points, and it is good at visualizing how the data relates to the principal components.

2) In t-SNE, the boundary of clusters seems to overlap a lot. This results from the characteristic of t-SNE that it only tries to map neighboring points while ignoring the larger picture.