In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import random
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
from sklearn.metrics import silhouette_score
from sklearn import preprocessing
from sklearn.metrics import silhouette_score
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

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

In [11]:
```
x1 = [5,8,7,8,3,4,2,3,4,5]
x2 = [8,6,5,4,3,2,2,8,9,8]
```

In [12]:
```
np.random.seed(123)
df = pd.DataFrame({'x1': x1, 'x2': x2})
labels = np.random.choice(3, 10, replace=True)
df['k_label'] = labels
df
```

Out[12]:

| | x1 | x2 | k_label |
|---|---|---|---|
| **0** | 5 | 8 | 2 |
| **1** | 8 | 6 | 1 |
| **2** | 7 | 5 | 2 |
| **3** | 8 | 4 | 2 |
| **4** | 3 | 3 | 0 |
| **5** | 4 | 2 | 2 |
| **6** | 2 | 2 | 2 |
| **7** | 3 | 8 | 1 |
| **8** | 4 | 9 | 2 |
| **9** | 5 | 8 | 1 |

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

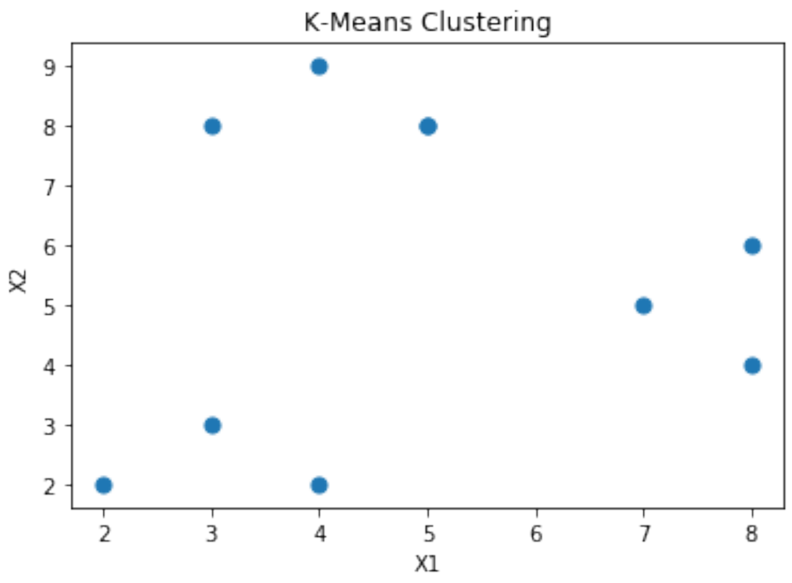
In [17]:
```
def k_means(num_k, df, max_iter):
    fit_df = df.copy()
    for i in range(max_iter):
        centroids = {}
        for k in range(num_k):
            cent1 = fit_df[fit_df['k_label'] == k]['x1'].mean()
            cent2 = fit_df[fit_df['k_label'] == k]['x2'].mean()
            centroids[k] = (cent1, cent2)
        new_k_labels = []
        for idx, row in fit_df.iterrows():
            min_distance = 50
            for k, v in centroids.items():
                eu_distance = math.sqrt((row['x1'] - v[0]) ** 2 + (row['
                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
k3 = k_means(3, df, 50)
k3
```

Out[17]:

| | x1 | x2 | k_label |
|---|---|---|---|
| **0** | 5 | 8 | 1 |
| **1** | 8 | 6 | 2 |
| **2** | 7 | 5 | 2 |
| **3** | 8 | 4 | 2 |
| **4** | 3 | 3 | 0 |
| **5** | 4 | 2 | 0 |
| **6** | 2 | 2 | 0 |
| **7** | 3 | 8 | 1 |
| **8** | 4 | 9 | 1 |
| **9** | 5 | 8 | 1 |

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

In [18]:
```python
fig = plt.figure()
plt.scatter(k3['x1'], k3['x2'], s=50)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K-Means Clustering')
plt.show()
```



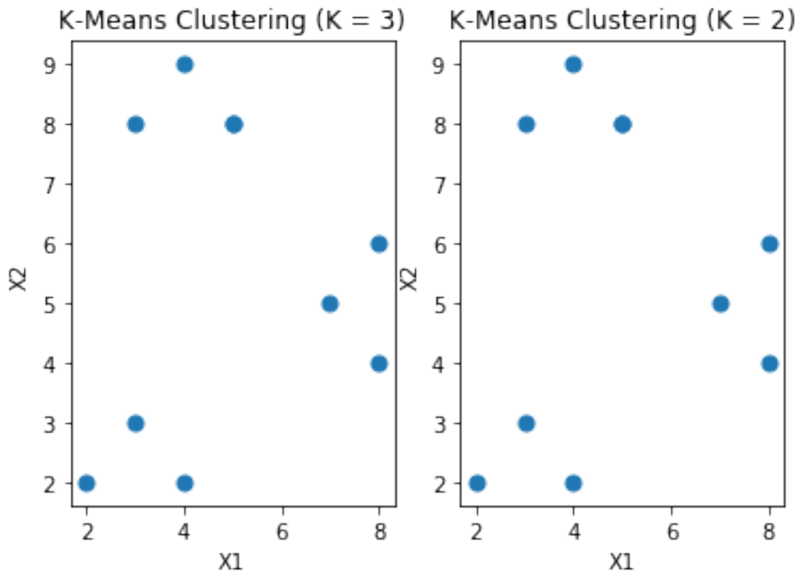
4.(5 points) Now, repeat the process, but this time initialize at k = 2 and present a final cluster assignment visually next to the previous search at k = 3.

In [19]:
```
k2 = k_means(2, df, 50)
k2
```

Out[19]:

|  | x1 | x2 | k_label |
|---|---|---|---|
| **0** | 5 | 8 | 1 |
| **1** | 8 | 6 | 1 |
| **2** | 7 | 5 | 1 |
| **3** | 8 | 4 | 1 |
| **4** | 3 | 3 | 0 |
| **5** | 4 | 2 | 0 |
| **6** | 2 | 2 | 0 |
| **7** | 3 | 8 | 1 |
| **8** | 4 | 9 | 1 |
| **9** | 5 | 8 | 1 |

In [20]:
```
fig, axs = plt.subplots(1, 2)
axs[0].scatter(k3['x1'], k3['x2'], s=50)
axs[0].set_xlabel('X1')
axs[0].set_ylabel('X2')
axs[0].set_title('K-Means Clustering (K = 3)')
axs[1].scatter(k2['x1'], k2['x2'], s=50)
axs[1].set_xlabel('X1')
axs[1].set_ylabel('X2')
axs[1].set_title('K-Means Clustering (K = 2)')
plt.show()
```



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 above graphs, k=3 fits better than k=2 because the data is naturally scattered into three clusters. K-means tries to cluster data points into 2 clusters when k=2, which means it basically seperated the middle cluster to reach two clusters. Also, when k=2, the clusters are different each time when we run the algorithm, depending on how the centers are assigned (randomly) as first.

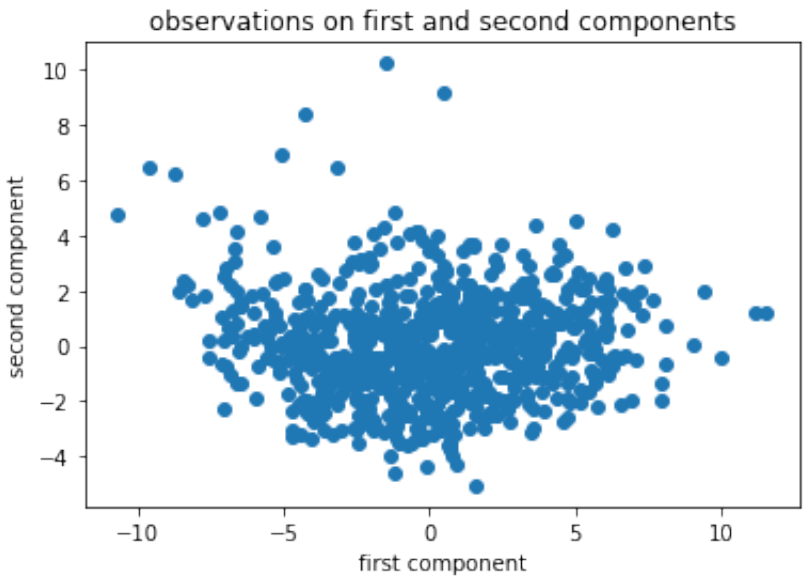
6.(15 points) Perform PCA on the dataset and plot the observations on the first and second principal components. Describe your results, e.g.,What variables appear strongly correlated on the first principal component? What about the second principal component?

In [24]:
```
df_wiki = pd.read_csv('/Users/hejielu/Desktop/data/wiki.csv')
```

In [25]:
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(df_wiki)
pca = PCA(random_state = 0)
X_new = pca.fit_transform(X)
```

In [26]:
```
plt.scatter(X_new[:,0],X_new[:,1])
plt.title('observations on first and second components')
plt.xlabel('first component')
plt.ylabel('second component')
```

Out[26]:
```
Text(0, 0.5, 'second component')
```



In [28]:
```
component = pd.DataFrame(pca.components_,columns = df_wiki.columns).T
```

In [30]:
```
component[0].sort_values(ascending = False)[:5]
```

Out[30]:
```
bi2      0.230924
bi1      0.226193
use3     0.218809
use4     0.214558
pu3      0.210863
Name: 0, dtype: float64
```

In [31]:
```
component[1].sort_values(ascending = False)[:5]
```

Out[31]:
```
exp4                                0.228494
use2                                0.218629
use1                                0.197827
vis3                                0.197635
domain_Engineering_Architecture    0.171484
Name: 1, dtype: float64
```

According to the above, the variables ‘bi2’, ‘bi1’, ‘use3’, ‘use4’, ‘pu3’ are strongly correlated with the first principal component, while ‘exp4’, ‘use2’, ‘use1’, ‘vis3’ are strongly correlated with the second one.

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?

In [32]:
```
pve = pca.explained_variance_ratio_
print('pve of all components:',pve)
```

```
pve of all components: [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 [33]:
```
pve_total = pve[0]+pve[1]
print('The first two components can explained:',pve_total)
```

```
The first two components can explained: 0.291831023274837
```

In [34]:
```
cve = np.cumsum(pve)
print(cve)
```

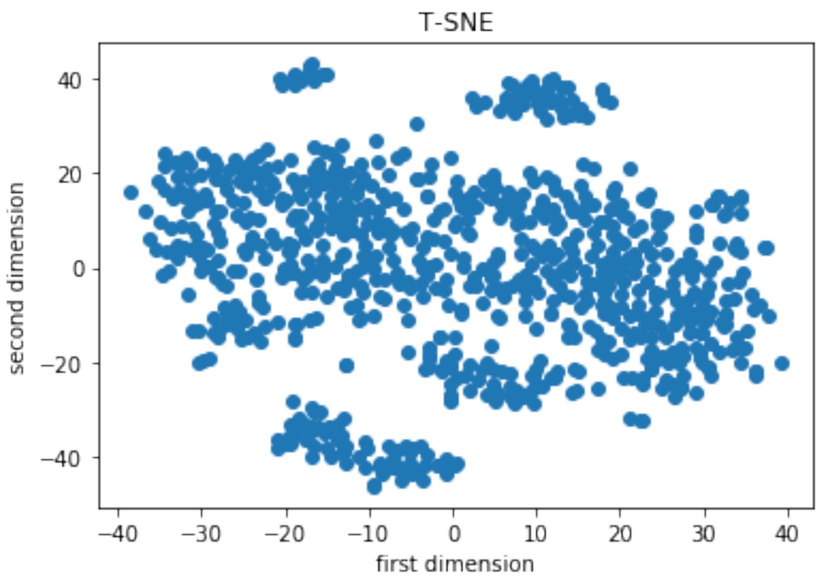
```
[0.22810628 0.29183102 0.34206809 0.38279644 0.42047368 0.45399461
 0.48432598 0.50984776 0.53402203 0.55794717 0.58060368 0.60131551
 0.62159548 0.64062877 0.6585537  0.6760302  0.69329353 0.70948585
 0.72477046 0.73934837 0.75367872 0.76717589 0.78013665 0.79206236
 0.80353114 0.81482421 0.82567965 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.        ]
```

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

In [35]:
```
X_tsne = TSNE(n_components=2,random_state = 0,perplexity = 20).fit_trans
```

In [37]:
```
plt.figure()
plt.scatter(X_tsne[:,0],X_tsne[:,1])
plt.title('T-SNE')
plt.xlabel('first dimension')
plt.ylabel('second dimension')
```

Out[37]:
```
Text(0, 0.5, 'second dimension')
```



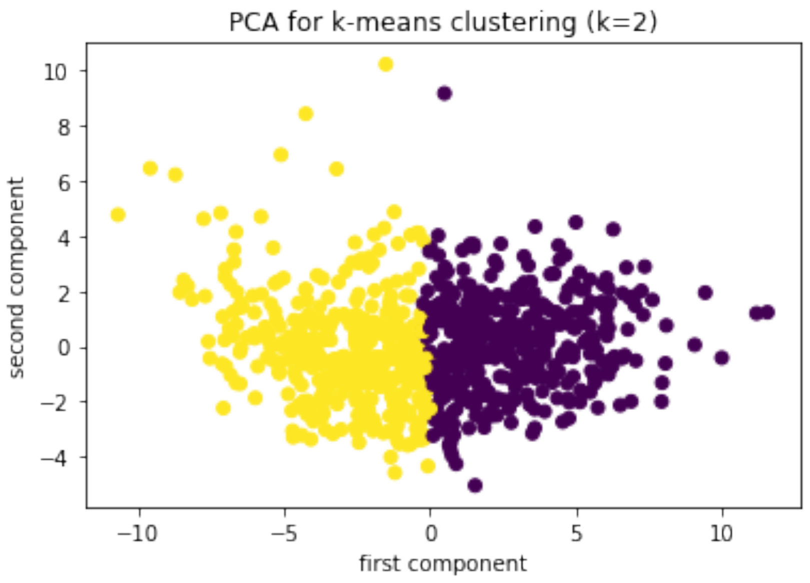
According to the above graph, there are some clustering on middle of the plot. Generally speaking, it seems that t-SNE is better at dealing with non-linear relationships, the plot suggests that the relationships between variables may be non-linear.

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 com- ponents from PCA and color-code each observation based on their cluster membership. Discuss your results.

In [ ]:
```
df_k = pd.read_csv('/Users/hejielu/Desktop/data/wiki.csv')
X = StandardScaler().fit_transform(df_k)
```

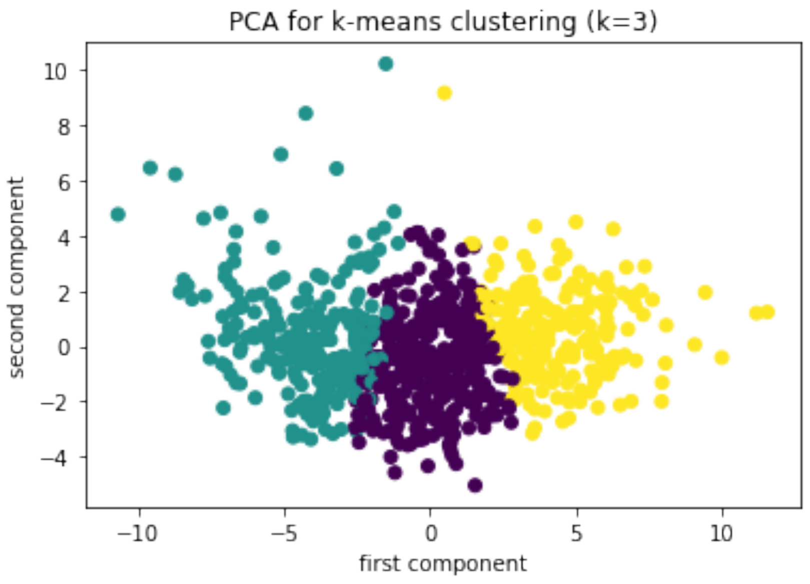
In [39]:
```
kmeans2 = KMeans(n_clusters = 2, random_state = 0).fit_predict(X)
plt.scatter(X_new[:,0],X_new[:,1],c = kmeans2)
plt.xlabel('first component')
plt.ylabel('second component')
plt.title('PCA for k-means clustering (k=2)')
```

Out[39]:
```
Text(0.5, 1.0, 'PCA for k-means clustering (k=2)')
```



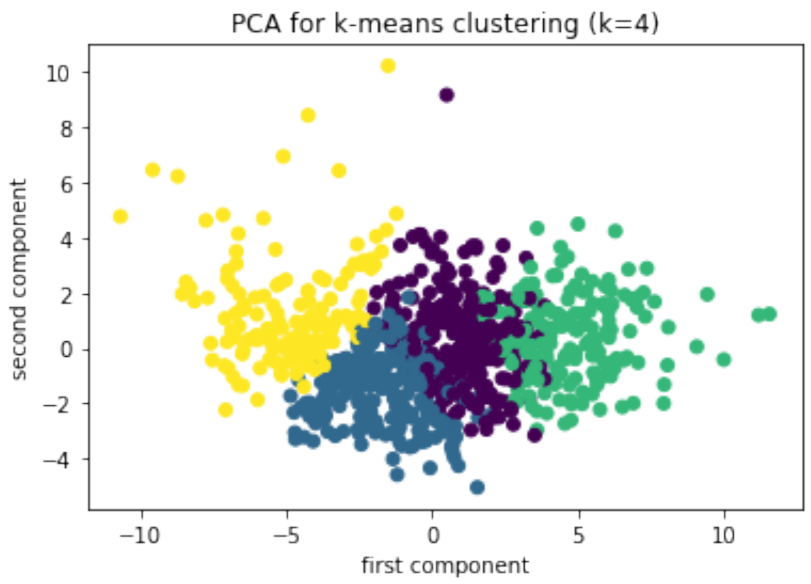
In [40]:
```
kmeans3 = KMeans(n_clusters = 3, random_state = 0).fit_predict(X)
plt.scatter(X_new[:,0],X_new[:,1],c = kmeans3)
plt.xlabel('first component')
plt.ylabel('second component')
plt.title('PCA for k-means clustering (k=3)')
```

Out[40]:
```
Text(0.5, 1.0, 'PCA for k-means clustering (k=3)')
```



In [41]:
```
kmeans4 = KMeans(n_clusters = 4, random_state = 0).fit_predict(X)
plt.scatter(X_new[:,0],X_new[:,1],c = kmeans4)
plt.xlabel('first component')
plt.ylabel('second component')
plt.title('PCA for k-means clustering (k=4)')
```

Out[41]:
```
Text(0.5, 1.0, 'PCA for k-means clustering (k=4)')
```

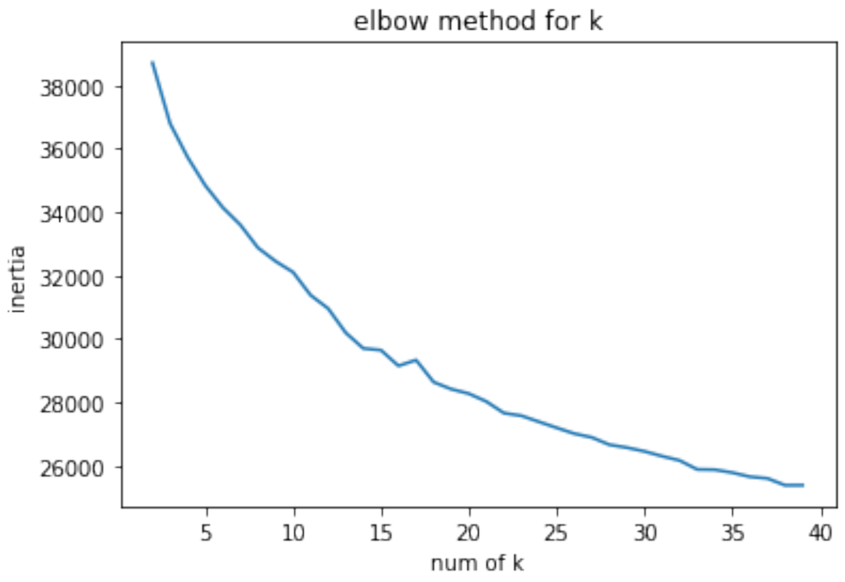


According to the above graphs, the k-means clustering with 2 and 3 clusters performs better to separate the data set since we can find a relatively clear boundary between each cluster. However, as for k-means clustering with k=4, the performance is not that good since clusters are overlapped and the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

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.

In [43]:
```
elbow = []
sil = []
for i in range(2,40):
    kmeans = KMeans(n_clusters=i, random_state = 0).fit(X)
    elbow.append(kmeans.inertia_)
    sil.append(silhouette_score(X,kmeans.labels_))
plt.plot(range(2,40),elbow)
plt.title('elbow method for k')
plt.xlabel('num of k')
plt.ylabel('inertia')
```

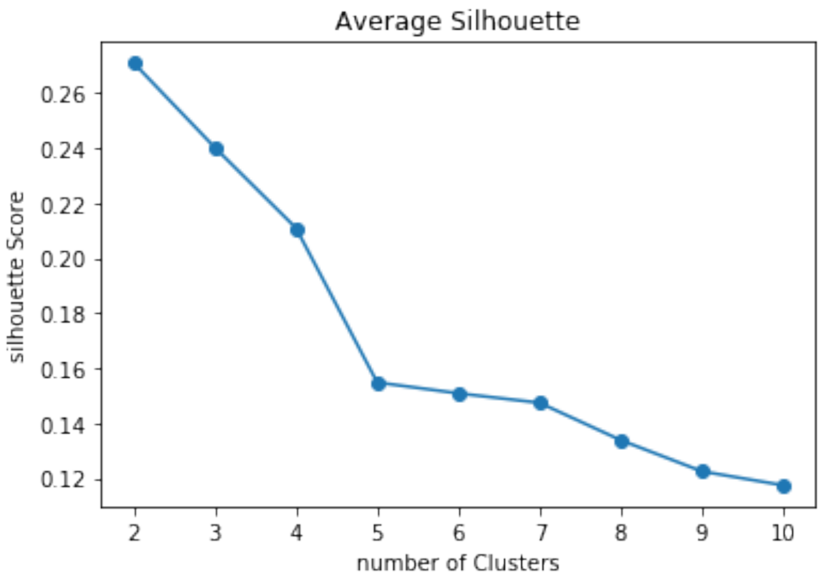
Out[43]:
```
Text(0, 0.5, 'inertia')
```



As k=2, it is clear that PCA better presents how k-means detects clusters--i.e. based on point distances. This is because PCA conducts a linear transformation on points and therefore does not alter the Euclidean relationship among points, which k-means relies on. On the contrary, t-SNE projects high-dimensional distributions to lower dimensions with a priority of keeping the distributional features. In another word, the transformation is non-linear and the input features are no longer identifiable. Therefore, it does not correspond to what k-means tries to capture.

In [47]:
```
silhouettes = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=1234)
    silhouettes.append(silhouette_score(df_wiki, kmeans.fit_predict(df_w
plt.plot(range(2, 11), silhouettes, '-o')
plt.ylabel('silhouette Score')
plt.xlabel('number of Clusters')
plt.title('Average Silhouette')
```

Out[47]:
```
Text(0.5, 1.0, 'Average Silhouette')
```

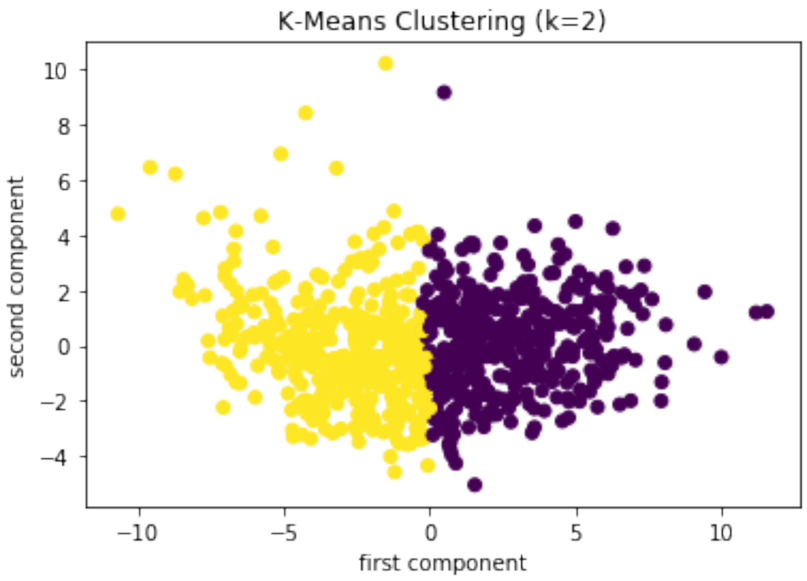


The optimal k is 2 since the average silhouette is the highest at this point.

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

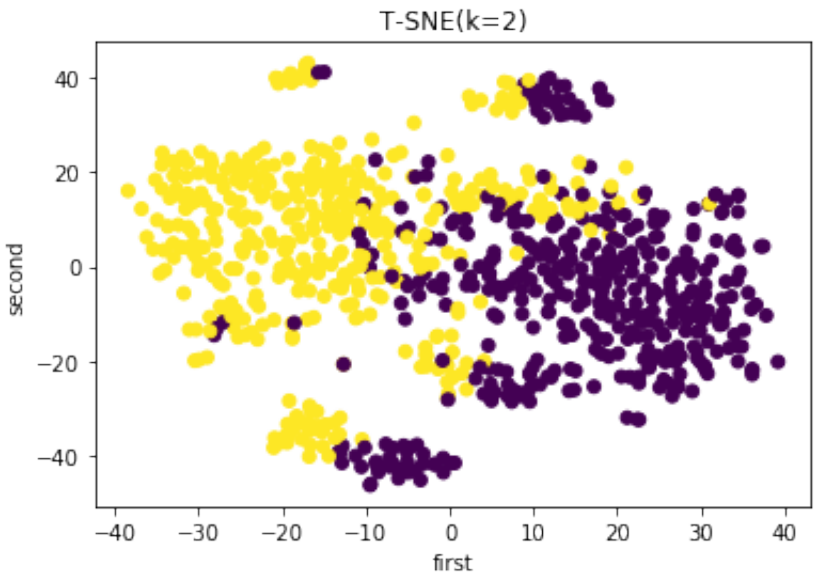
In [51]:
```
plt.scatter(X_new[:,0],X_new[:,1],c = kmeans2)
plt.xlabel('first component')
plt.ylabel('second component')
plt.title('K-Means Clustering (k=2)')
```

Out[51]:
```
Text(0.5, 1.0, 'K-Means Clustering (k=2)')
```



In [52]:
```
plt.scatter(X_tsne[:,0],X_tsne[:,1],c=kmeans2)
plt.title('T-SNE(k=2)')
plt.xlabel('first')
plt.ylabel('second')
```

Out[52]:
```
Text(0, 0.5, 'second')
```



According to the above graphs, we can see the pca method seperates data well with the first component, and there are merely overlapping between two clusters. For the tsne, the boundary is not as clear as in the pca method. It seems more than the first component participates in the clustering.Overall, PCA gives us a better visualization of the data, while for t-SNE, the clusters are overlapped with a fuzzy boundary.