In [50]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

k-Means Clustering 'by hand'

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

In [4]:

```
np.random.seed(0)
```

In [7]:

```
input1 = [5,8,7,8,3,4,2,3,4,5]
input2 = [8,6,5,4,3,2,2,8,9,8]
label = np.random.choice(3,10)
data = np.column_stack((input1,input2,label))
```

In [33]:

```
print(data)
```

```
[[5 8 1]
 [8 6 2]
 [7 5 2]
 [8 4 2]
 [3 3 0]
 [4 2 0]
 [2 2 0]
 [3 8 1]
 [4 9 1]
 [5 8 1]]
```

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

In [37]:

```
def k_means_clustering(k):
    k_means = (data.sample(k, replace=False))
    k_means2 = pd.DataFrame()
    clusters = pd.DataFrame()

    while not k_means2.equals(k_means):
        cluster_count = 0
        for idx, k_mean in k_means.iterrows():
            clusters[cluster_count] = (data[k_means.columns] - np.array(k_mean))
.pow(2).sum(1).pow(0.5)
            cluster_count += 1

        data['Cluster'] = clusters.idxmin(axis=1)
        k_means2 = k_means
        k_means = pd.DataFrame()
        k_means_frame = data.groupby('Cluster').agg(np.mean)

        k_means[k_means_frame.columns] = k_means_frame[k_means_frame.columns]
        return np.asarray(k_means),np.asarray(clusters)
```

In [26]:

```
k_means_clustering(3)
print(k_means)
print(clusters)
```

```
[1 2 2 2 0 0 0 1 1 1]
[[3.         2.33333333]
 [4.25       8.25      ]
 [7.66666667 5.        ]]
```

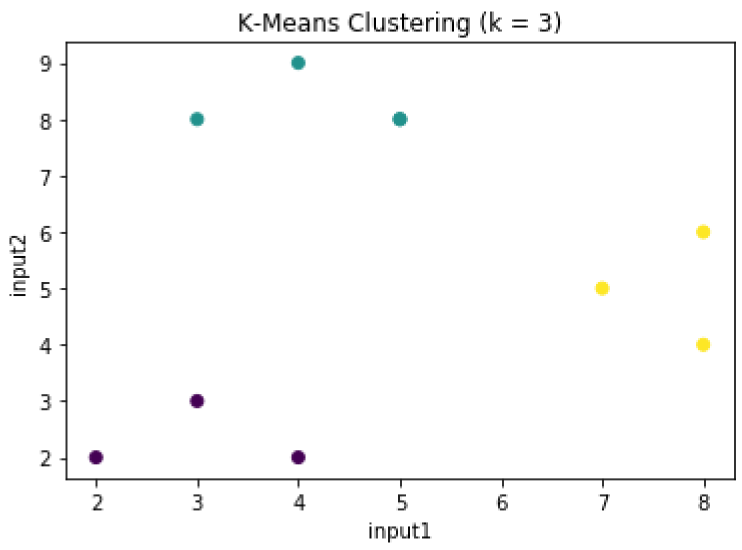
1. (5 points) Present a visual description of the final, converged (stopped) cluster assignments.

In [28]:

```
plt.scatter(input1,input2,c = clusters)
plt.title('K-Means Clustering (k = 3)')
plt.xlabel('input1')
plt.ylabel('input2')
```

Out[28]:

```
Text(0, 0.5, 'input2')
```



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 [27]:

```
k_means_clustering(2)
print(k_means)
print(clusters)
```

```
[1 1 1 1 0 0 0 1 1 1]
[[3.         2.33333333]
 [5.71428571 6.85714286]]
```

In [32]:

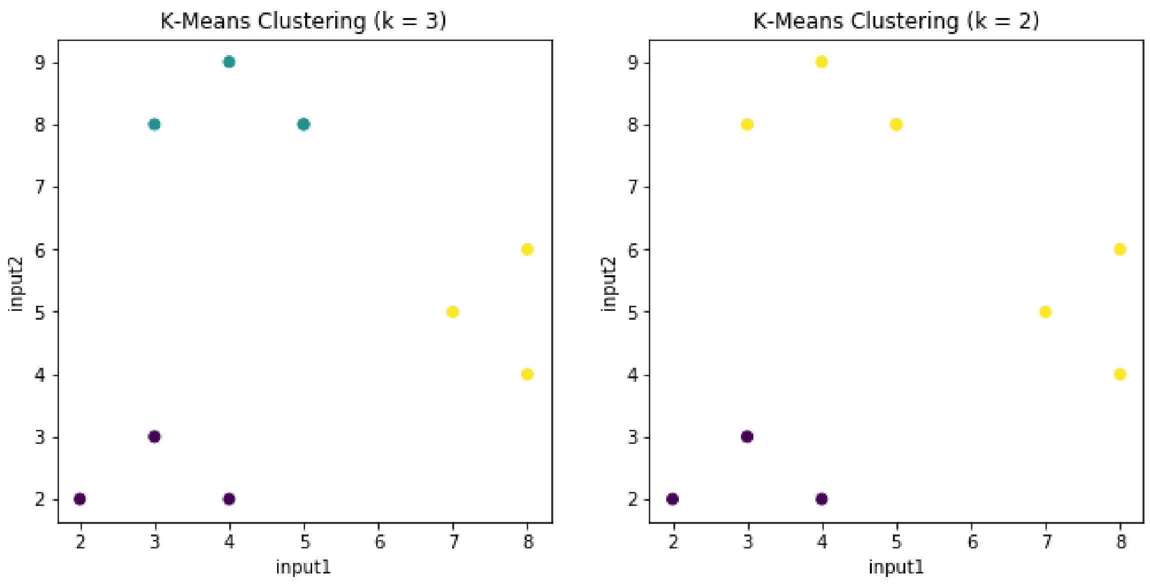
```
fig,ax = plt.subplots(1,2,figsize=(11,5))

ax[0].scatter(input1,input2,c = clusters)
ax[0].set_title('K-Means Clustering (k = 3)')
ax[0].set_xlabel('input1')
ax[0].set_ylabel('input2')

ax[1].scatter(input1,input2,c = clusters)
ax[1].set_title('K-Means Clustering (k = 2)')
ax[1].set_xlabel('input1')
ax[1].set_ylabel('input2')
```

Out[32]:

```
Text(0, 0.5, 'input2')
```



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?

According to the visualization, the k-means with k = 3 outperforms the kmeans with k=2. When k=2, it wrongly clustered two groups into the same 'yellow' group. In the 3-cluster plot, we can see that the clustering method clearly groups close points together, and between groups there are a large distance. In the 2-cluster plot, however, within group distance is too far and it is likely that this method cluster different groups into one.

Application

1. (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 [46]:

```
df = pd.read_csv('/Users/lijiaxuan/Downloads/problem-set-7-master/data/wiki.csv'
)
df = df.astype(int)
```

In [66]:

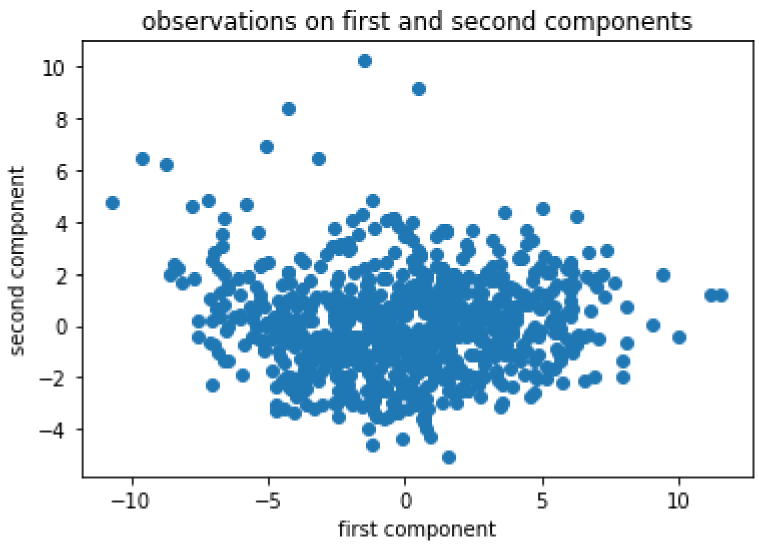
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(df)
pca = PCA(random_state = 0)
X_new = pca.fit_transform(X)
```

In [70]:

```
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[70]:

```
Text(0, 0.5, 'second component')
```



In [78]:

```
component = pd.DataFrame(pca.components_,columns = df.columns).T
component[0].sort_values(ascending = False)[:5]
```

Out[78]:

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

In [79]:

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

Out[79]:

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

The variables that are strongly correlated with the first components include: bi1,bi2(behavior intentions to use and recommend wiki), use3,use4(user behavior of recommending others to use wiki), and the usefulness for teaching (pu3). the variables that are strongly correlated with the second component include: exp4 (experience of contribute to wiki), use2(develop teaching with wiki), use1(develop educational activities), vis3(cite wiki for academics), domain_engineering_architecture(from the domain of engineer and architectures)

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 [84]:

```
pve = pca.explained_variance_ratio_
print('pve of all components:',pve)
pve_two = pve[0]+pve[1]
print('first two components pve:',pve_two)
```

```
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]
first two components pve: 0.291831023274837
```

The first two components explained 29.2% of the variance

In [86]:

```
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.        ]
```

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

In [163]:

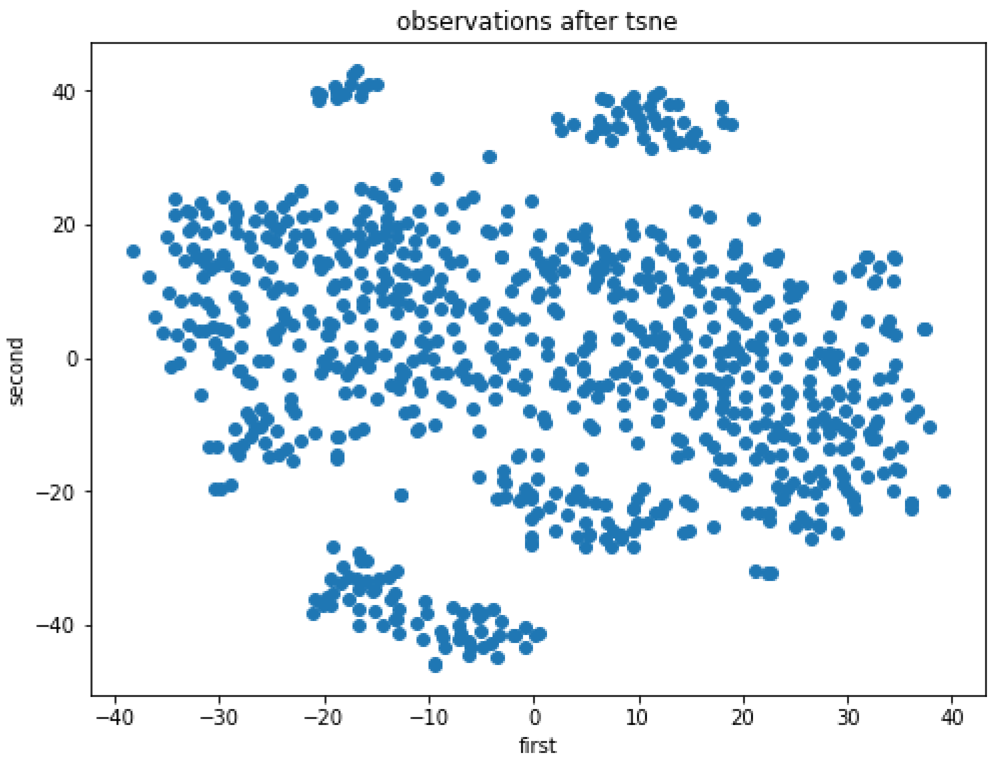
```
X_embedded = TSNE(n_components=2,random_state = 0,perplexity = 20).fit_transform
(X)
```

In [164]:

```
plt.figure(figsize=(8,6))
plt.scatter(X_embedded[:,0],X_embedded[:,1])
plt.title('observations after tsne')
plt.xlabel('first')
plt.ylabel('second')
```

Out[164]:

```
Text(0, 0.5, 'second')
```



there are some clustering on the upper right of the plot but generally speaking, it seems that tsne failed to cluster data

Clustering

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 [63]:

```
df = pd.read_csv('/Users/lijiaxuan/Downloads/problem-set-7-master/data/wiki.csv'
)

X = StandardScaler().fit_transform(df)
```

In [115]:

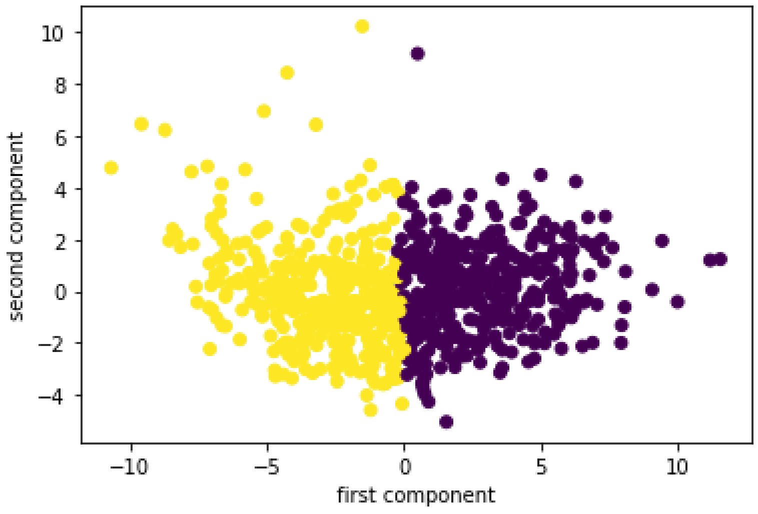
```
kmeans2 = KMeans(n_clusters = 2, random_state = 0).fit_predict(X)
```

In [118]:

```
plt.scatter(X_new[:,0],X_new[:,1],c = kmeans2)
plt.xlabel('first component')
plt.ylabel('second component')
```

Out[118]:

```
Text(0, 0.5, 'second component')
```

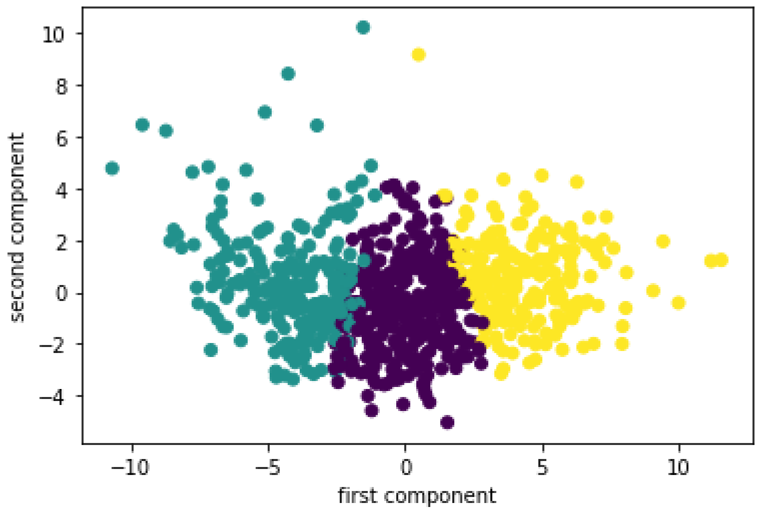


In [120]:

```
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')
```

Out[120]:

```
Text(0, 0.5, 'second component')
```

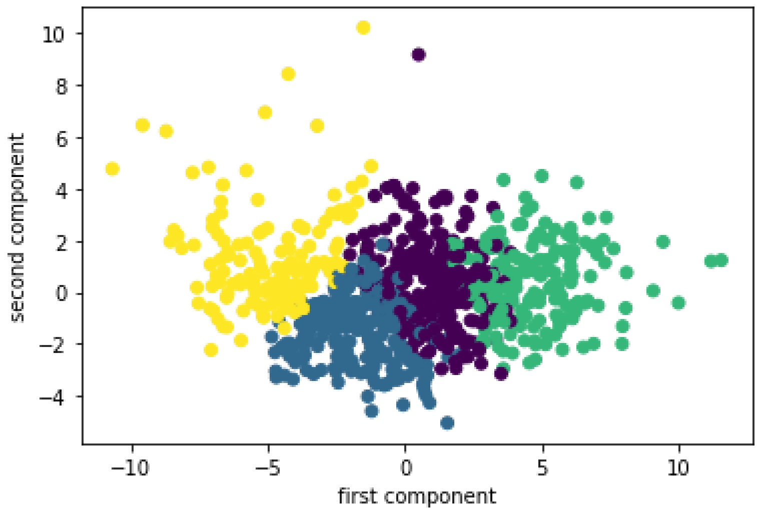


In [121]:

```
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')
```

Out[121]:

```
Text(0, 0.5, 'second component')
```



all of the four methods could seperate the data well, however, as k increases, the degree of overlapping increases. Moreover, the data are seperate on the first component, which means the first component explains the majority of the variance.

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 [128]:

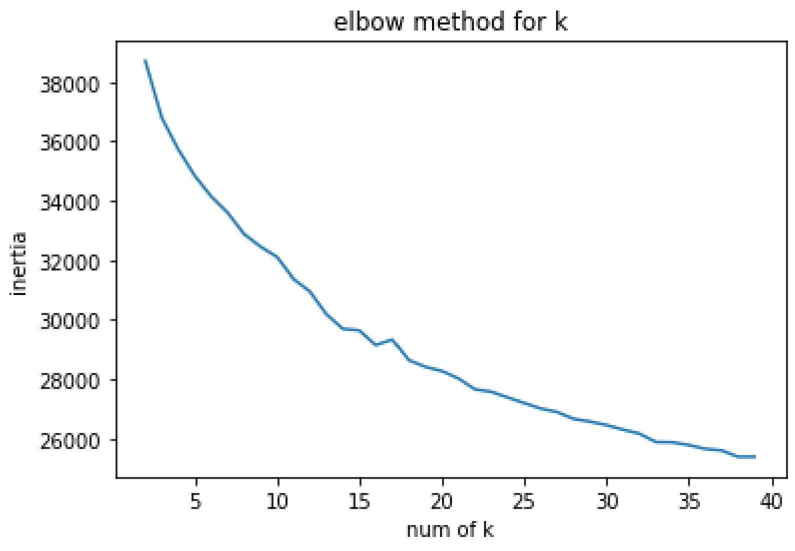
```
from sklearn.metrics import silhouette_score
elbow = []
sil = []
for i in range(2,40):
    kmeans = KMeans(n_clusters=i, random_state = 0).fit(X)
    elbow.append(kmeans.inertia_)
    sil.append(silhouette_score(X,kmeans.labels_))
```

In [129]:

```
plt.plot(range(2,40),elbow)
plt.title('elbow method for k')
plt.xlabel('num of k')
plt.ylabel('inertia')
```

Out[129]:

```
Text(0, 0.5, 'inertia')
```

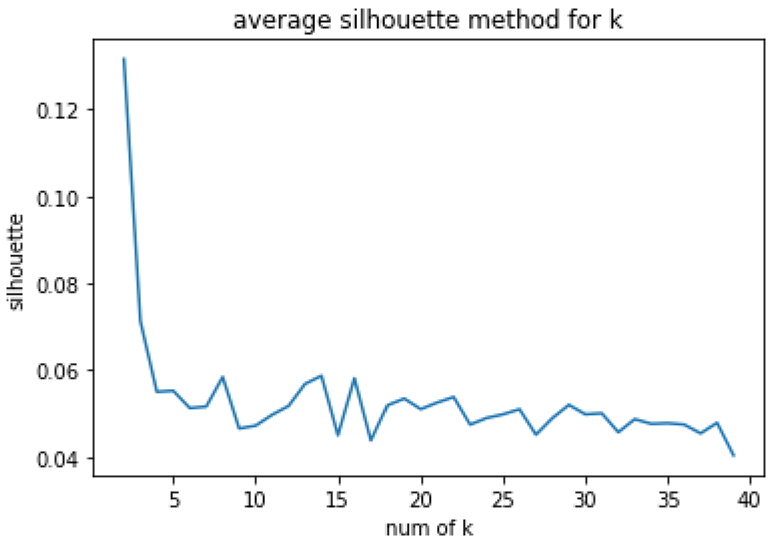


In [131]:

```
plt.plot(range(2,40),sil)
plt.title('average silhouette method for k')
plt.xlabel('num of k')
plt.ylabel('silhouette')
```

Out[131]:

```
Text(0, 0.5, 'silhouette')
```



In [138]:

```
def optimalK(data, nrefs=3, maxClusters=15):
    """
    Calculates KMeans optimal K using Gap Statistic from Tibshirani, Walther, Ha
stie
    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 kmeans getting re
sulting 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':gap}, ignore_index
=True)

    return (gaps.argmax() + 1, resultsdf)  # Plus 1 because index of 0 means 1 c
luster is optimal, index 2 = 3 clusters are optimal
```

In [143]:

```
num = optimalK(X,2,40)
```

In [144]:

```
print(num)
```

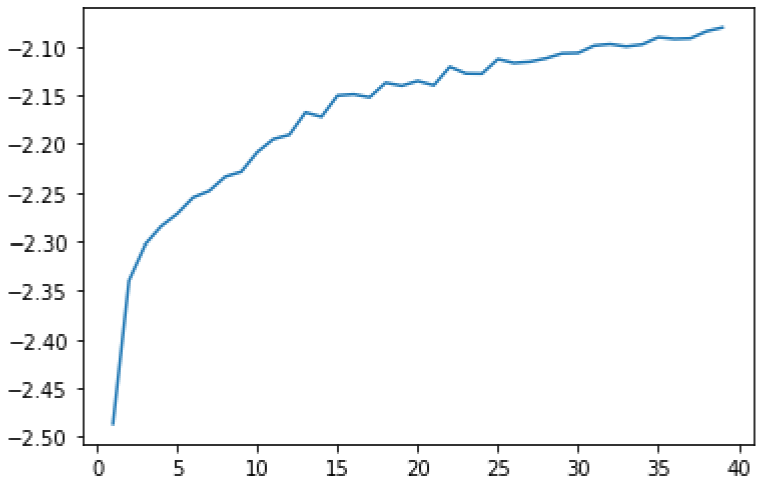
```
(39,     clusterCount       gap
0            1.0 -2.486732
1            2.0 -2.339466
2            3.0 -2.303172
3            4.0 -2.284576
4            5.0 -2.271813
5            6.0 -2.254971
6            7.0 -2.248263
7            8.0 -2.233619
8            9.0 -2.228531
9           10.0 -2.208291
10          11.0 -2.195011
11          12.0 -2.190514
12          13.0 -2.167549
13          14.0 -2.172041
14          15.0 -2.150177
15          16.0 -2.148915
16          17.0 -2.152012
17          18.0 -2.137041
18          19.0 -2.140184
19          20.0 -2.135237
20          21.0 -2.139701
21          22.0 -2.120653
22          23.0 -2.127541
23          24.0 -2.127774
24          25.0 -2.112740
25          26.0 -2.116762
26          27.0 -2.115448
27          28.0 -2.111975
28          29.0 -2.106868
29          30.0 -2.106422
30          31.0 -2.098879
31          32.0 -2.097310
32          33.0 -2.099799
33          34.0 -2.097664
34          35.0 -2.090210
35          36.0 -2.092018
36          37.0 -2.091447
37          38.0 -2.084153
38          39.0 -2.080250)
```

In [156]:

```
plt.plot(num[1]['clusterCount'],num[1]['gap'])
```

Out[156]:

```
[<matplotlib.lines.Line2D at 0x1a23dba438>]
```



In elbow method, the optimal number is at around 16, but it is not a clear elbow. In silbouette method, the optimal number is 2. In gap statistics, the optimal number is 39. The elbow method give unclear elbows, and the gap statistics increases gradually. As we can see the first component could cluster the data clearly, we will choose k=2 as our optimal k.

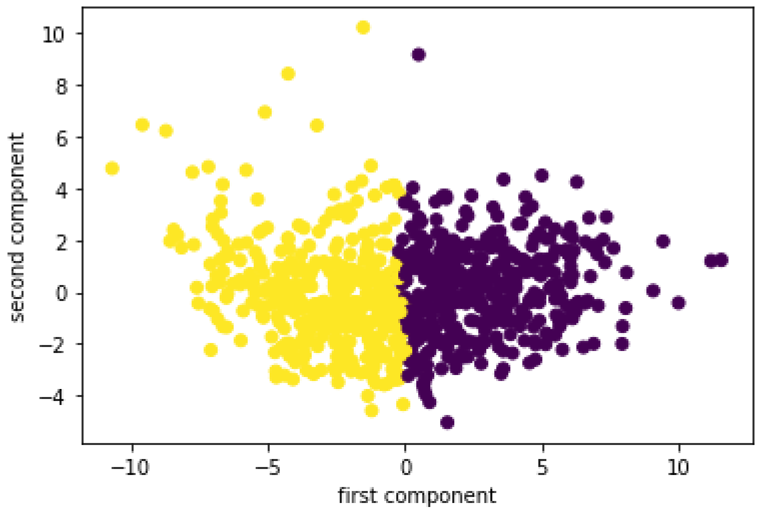
1. (15 points) Visualize the results of the optimal kˆ-means clustering model. First use the first and second principal components from PCA, and color-code each observation based on their cluster membership. Next use the first and second dimensions from t-SNE, and color-code each observation based on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

In [160]:

```
plt.scatter(X_new[:,0],X_new[:,1],c = kmeans2)
plt.xlabel('first component')
plt.ylabel('second component')
```

Out[160]:

```
Text(0, 0.5, 'second component')
```

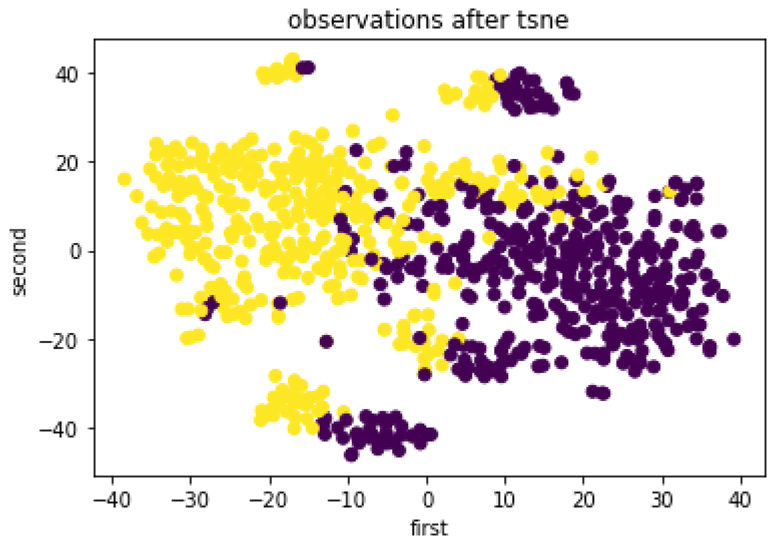


In [165]:

```
plt.scatter(X_embedded[:,0],X_embedded[:,1],c=kmeans2)
plt.title('observations after tsne')
plt.xlabel('first')
plt.ylabel('second')
```

Out[165]:

```
Text(0, 0.5, 'second')
```



from the two plots, 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.