Xu_Yilun_HW07

March 14, 2020

[1]:
```
import random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabaz_score
from sklearn import metrics
from scipy.spatial.distance import cdist
from sklearn.metrics import silhouette_score
from gap_statistic import optimalK
from yellowbrick.cluster.elbow import kelbow_visualizer
```

1 k-Means Clustering "By Hand"

1.1 1

[2]:
```
x_1 = [5,8,7,8,3,4,2,3,4,5]
x_2 = [8,6,5,4,3,2,2,8,9,8]
def get_data(x_1,x_2, k):
    random.seed(20)
    X = list(map(lambda x, y:(x,y), x_1, x_2))
    data = {'x_1': x_1, 'x_2': x_2, 'X': X}
    data = pd.DataFrame(data)
    centers = random.sample(X, k)
    data['label'] = 'undecided'
    for i in range(len(x_1)):
        if data['X'][i] in centers:
            data['label'][i] = data['X'][i]
        else:
            data['label'][i] = random.choice(centers)
    data['distance_dict'] = [0]*len(x_1)
```

```
    return data
labeled_data = get_data(x_1, x_2, k=3)
labeled_data
```

[2]:

	x_1	x_2	X	label	distance_dict
0	5	8	(5, 8)	(3, 3)	0
1	8	6	(8, 6)	(8, 6)	0
2	7	5	(7, 5)	(7, 5)	0
3	8	4	(8, 4)	(8, 6)	0
4	3	3	(3, 3)	(3, 3)	0
5	4	2	(4, 2)	(7, 5)	0
6	2	2	(2, 2)	(7, 5)	0
7	3	8	(3, 8)	(3, 3)	0
8	4	9	(4, 9)	(3, 3)	0
9	5	8	(5, 8)	(7, 5)	0

We can see the initialized labels of each obeservation in the column named label.

1.2 2

[3]:
```
def dis(x, y):
    return ((x[0]-y[0])**2 + (x[1]-y[1])**2)**0.5
def dis_dict(x, centers):
    dic = {}
    for center in centers:
        dic[(center)] = dis(x, center)
    return dic

def update_label(data):
    centers = set(data['label'].unique())
    data['distance_dict'] = data.apply(lambda x:
                                       dis_dict(x['X'], centers), axis = 1)
    data['label'] = data.apply(lambda x:min(x['distance_dict'],
                                            key=x['distance_dict'].get),
                                        axis = 1)
    print(set(data['label'].unique()))
    df = data.groupby('label')[['x_1','x_2']].mean()
    df['new_label'] = df.apply(lambda x:(x['x_1'], x['x_2']), axis = 1)
    df = df.reset_index()
    for i in range(len(data)):
        for j in range(len(df)):
            if data['label'][i] == df['label'][j]:
                data['label'][i] = df['new_label'][j]
    centers = set(data.label.unique())
    data['distance_dict'] = data.apply(lambda x:
                                       dis_dict(x['label'], centers), axis = 1)
    data['label'] = data.apply(lambda x:min(x['distance_dict'],
```

```
                                                    key=x['distance_dict'].get),
                                      axis = 1)
    return data
def finish_kmean(data):
    while True:
        labels = set(data['label'].unique())
        new_data = update_label(data)
        new_labels = set(data['label'].unique())
        if labels == new_labels:
            break
        else:
            data = new_data
    return data
k3_data = get_data(x_1,x_2,3)
k3_data = finish_kmean(get_data(x_1,x_2,3))
```

```
{(8, 6), (7, 5), (3, 3)}
{(3.0, 2.3333333333333335), (5.5, 7.75), (6.0, 5.666666666666667)}
{(3.0, 2.3333333333333335), (7.666666666666667, 5.0), (4.25, 8.25)}
```

Here we see the cluster centroids in different iteration rounds. The last line represents the final (converged) cluster centroids.

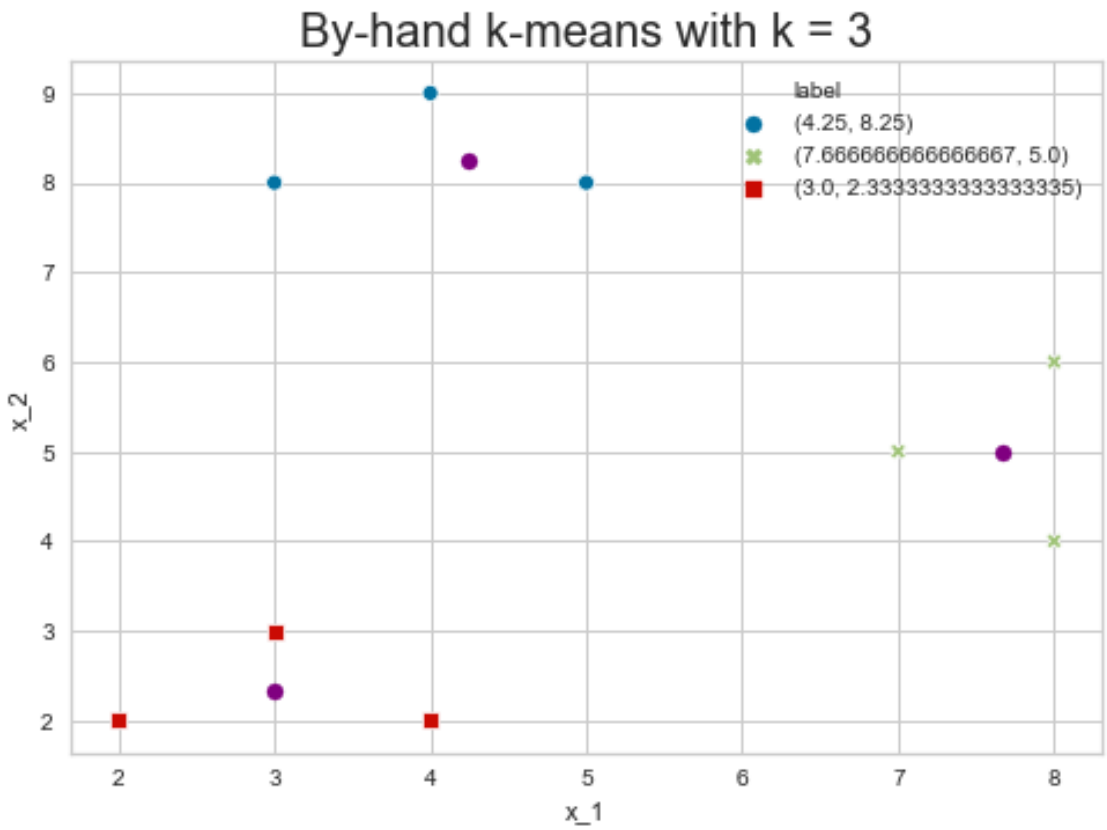
[4]:
```
X = list(map(lambda x, y:(x,y), x_1, x_2))
k3_kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
k3_kmeans.cluster_centers_
```

[4]:
```
array([[7.66666667, 5.        ],
       [4.25      , 8.25      ],
       [3.        , 2.33333333]])
```

Now we used the package in sklearn to check our answers. We can see that the answers from the sklearn package match our results, which justifies our calculation.

1.3 3

[5]:
```
k3=sns.scatterplot(data = k3_data, x='x_1',y='x_2', style = 'label',
                   hue = 'label')
labels = k3_data.label.unique()
plot_labels_x = [i[0] for i in labels]
plot_labels_y = [i[1] for i in labels]
plt.scatter(plot_labels_x, plot_labels_y,c = 'purple')
plt.title("By-hand k-means with k = 3", size = 20)
plt.show()
```



1.4 4

[6]:
```
k2_data = get_data(x_1,x_2,2)
k2_data = finish_kmean(k2_data)
```

```
{(7, 5), (3, 3)}
{(5.714285714285714, 6.857142857142857), (3.0, 2.3333333333333335)}
```

Here we see the cluster centroids in different iteration rounds. The last line represents the final (converged)cluster centroids.

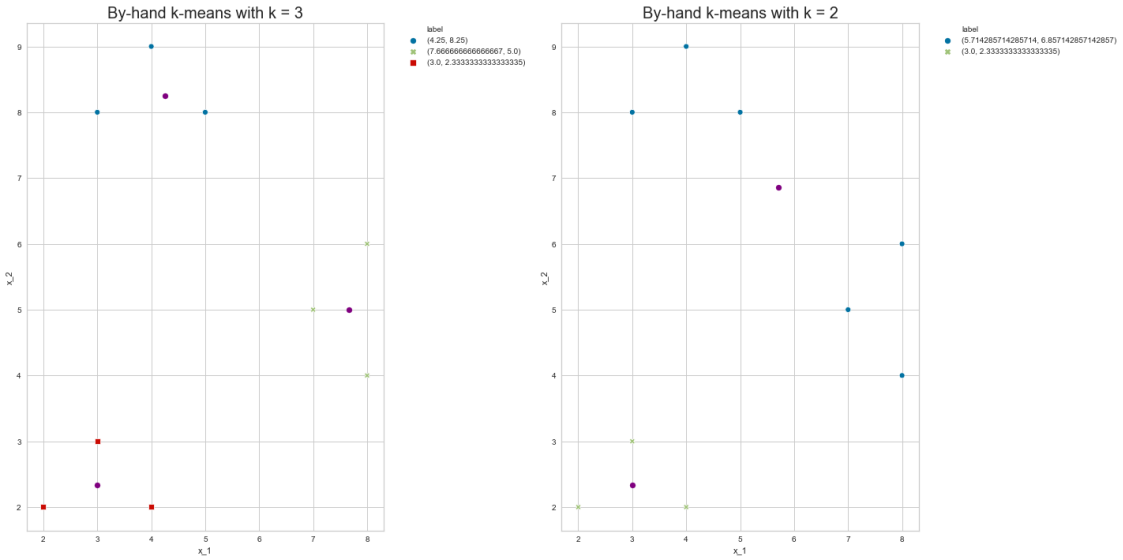
[7]:
```
X = list(map(lambda x, y:(x,y), x_1, x_2))
k2_kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
k2_kmeans.cluster_centers_
```

[7]:
```
array([[3.        , 2.33333333],
       [5.71428571, 6.85714286]])
```

Now we used the package in sklearn to check our answers. We can see that the answers from the sklearn package match our results, which justifies our calculation.

[8]:
```
#visualization
plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
k3=sns.scatterplot(data = k3_data, x='x_1',y='x_2', style = 'label',
                   hue = 'label')
labels = k3_data.label.unique()
plot_labels_x = [i[0] for i in labels]
plot_labels_y = [i[1] for i in labels]
plt.scatter(plot_labels_x, plot_labels_y,c = 'purple')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title("By-hand k-means with k = 3", size = 20)

plt.subplot(1, 2, 2)
k2=sns.scatterplot(data = k2_data, x='x_1',y='x_2', style = 'label',
                   hue = 'label')
labels = k2_data.label.unique()
plot_labels_x = [i[0] for i in labels]
plot_labels_y = [i[1] for i in labels]
plt.scatter(plot_labels_x, plot_labels_y,c = 'purple')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title("By-hand k-means with k = 2", size = 20)
plt.tight_layout()
```



1.5 5

[9]:
```
print("k = 2:",k2_kmeans.inertia_)
print("k = 3:",k3_kmeans.inertia_)
```

```
k = 2: 46.952380952380956
```

```
k = 3: 8.833333333333332
```

I think the best value for k is 3. At first, we can clearly observe from the above plots that the data are clustered into 3 groups. It is so clear that we can get the conclusion straightforward. Second, we can use a quantitative method to judge. Here we use the sum of squared distances of samples to their closest cluster center. According to the above results, we can see that when k equals to 3, the the sum of squared distances of samples to their closest cluster center is much smaller. We can try more different k values and the results will be the same.

2 Application

[10]:
```
wiki = pd.read_csv('C:/Users/mac/Desktop/temp/problem-set-7-master/data/wiki.
↪csv')
wiki_std = pd.DataFrame(StandardScaler().fit_transform(wiki))
random.seed(20)
```

2.1 Dimension reduction

2.1.1 6

[11]:
```
pca = PCA(n_components = 2)
pca.fit(wiki_std)
wiki_transformed = pca.fit_transform(wiki_std)
wiki_pca = pd.DataFrame(pca.components_, columns = wiki.columns,
                        index = ['PC_1','PC_2']).T
wiki_pca.iloc[wiki_pca['PC_1'].abs().argsort()[::-1]].head(10)
```

[11]:

	PC_1	PC_2
bi2	0.230924	0.083439
bi1	0.226193	0.056381
use3	0.218809	0.155158
use4	0.214558	0.160872
pu3	0.210863	0.028799
exp1	0.208592	0.070568
use5	0.206539	0.029827
exp2	0.195043	-0.029539
pu1	0.192827	0.008300
pu2	0.190588	0.017698

According to the above dataframe, we can see that bi2, bi1, uses 3 and other 7 variables are the top 10 variables which are highly correlated on the first component. According to the variable explanation, bi2 and bi1 both describes behavioral intention of recommending or using Wikipedia.In addition, variables including use3, use4 and use5 decribe how indiviual uses wikipedia, and the sum of their coeffcients are more than 0.6. According to these facts, we can deduce that people's behavioral intention and how they use wikipedia are strongly related to the first component.

[12]:
```
wiki_pca.iloc[wiki_pca['PC_2'].abs().argsort()[::-1]].head(10)
```

[12]:

	PC_1	PC_2
peu1	0.061228	-0.271735
inc1	0.104667	-0.245446
sa3	0.120376	-0.242324
sa1	0.121658	-0.229929
exp4	0.099873	0.228472
enj2	0.131110	-0.227598
sa2	0.117590	-0.226769
peu2	0.113719	-0.222372
inc3	0.081402	-0.220997
use2	0.147852	0.218628

According to the above dataframe, we can see that peu1, inc1, sa 3 and other 7 variables are the top 10 variables which are highly correlated on the second component. Variables including sa1, sa2 and sa3 all decribe the social image of wikipedia, and the absolute value of the sum of coefficients of these three variables are nearly 0.7. We can deduce that the social image of wikipedia is strongly related to the second component.

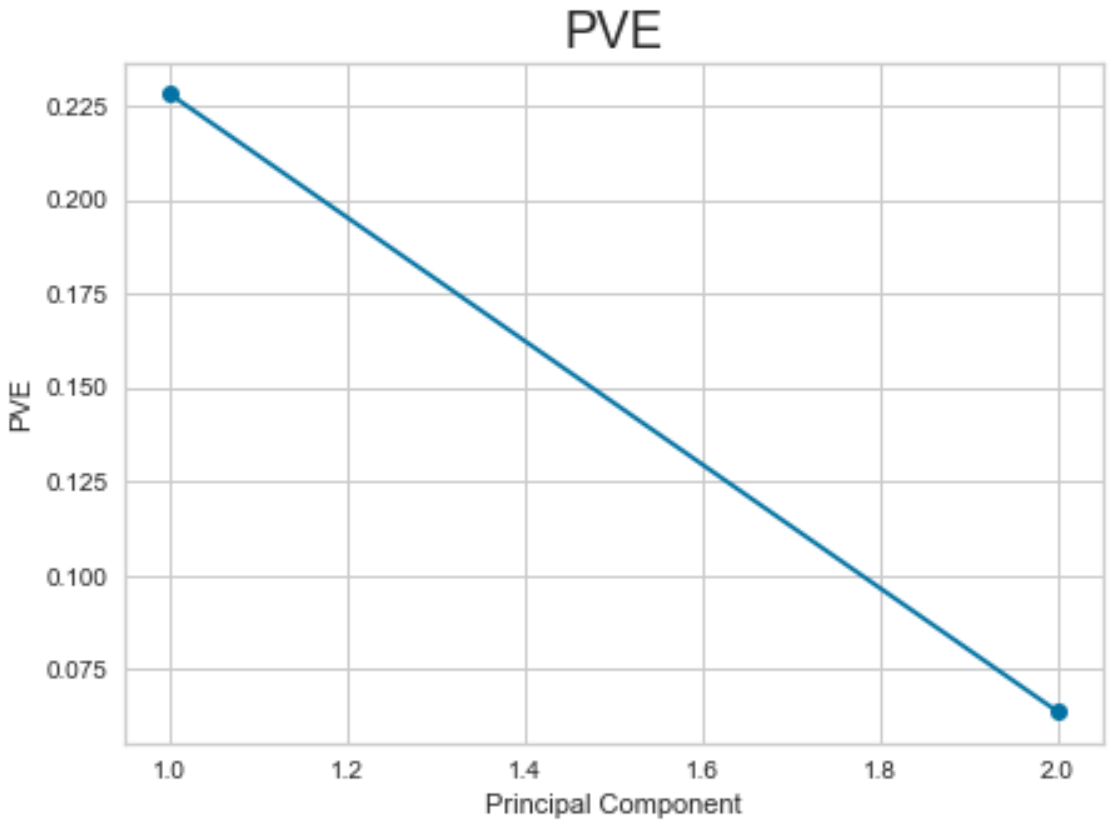
In addition, we can see that in the variables which are more correlated on the first component, the coefficients are all positive. in the variables which are more correlated on the second component, the coefficients are almost negative. Therefore, we may deduce that the two components describe the data in two opposite directions.

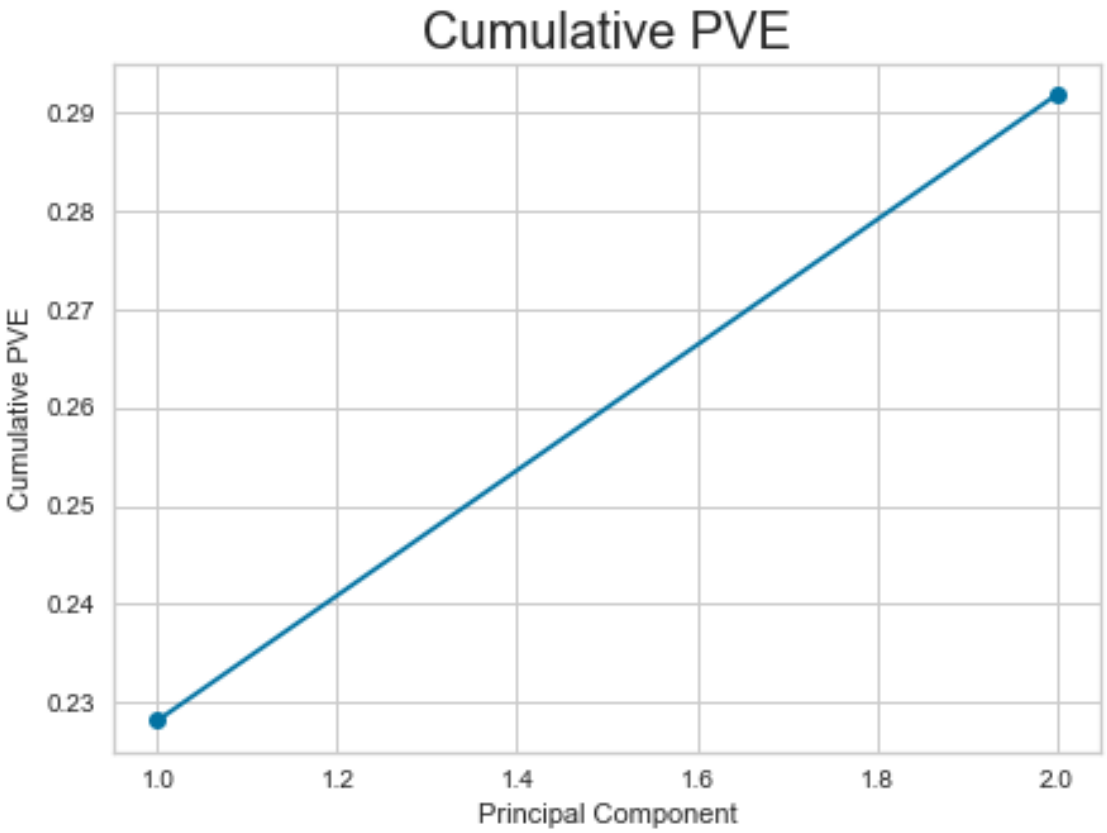
2.1.2 7

[13]:
```
plt.figure(figsize=[7,5])
plt.plot([1,2],pca.explained_variance_ratio_,'-o')
plt.ylabel('PVE')
plt.xlabel('Principal Component')
plt.title('PVE', size = 20)

plt.figure(figsize=[7,5])
plt.plot([1,2],np.cumsum(pca.explained_variance_ratio_),'-o')
plt.ylabel('Cumulative PVE')
plt.xlabel('Principal Component')
plt.title('Cumulative PVE', size = 20)
```

[13]:
```
Text(0.5, 1.0, 'Cumulative PVE')
```


PVE
0.225
0.200
0.175
0.150
0.125
0.100
0.075
PVE
1.0
1.2
1.4
1.6
1.8
2.0
Principal Component



[14]:
```
print('Total variance being explained: ', np.cumsum(pca.
↪explained_variance_ratio_)[-1])
```

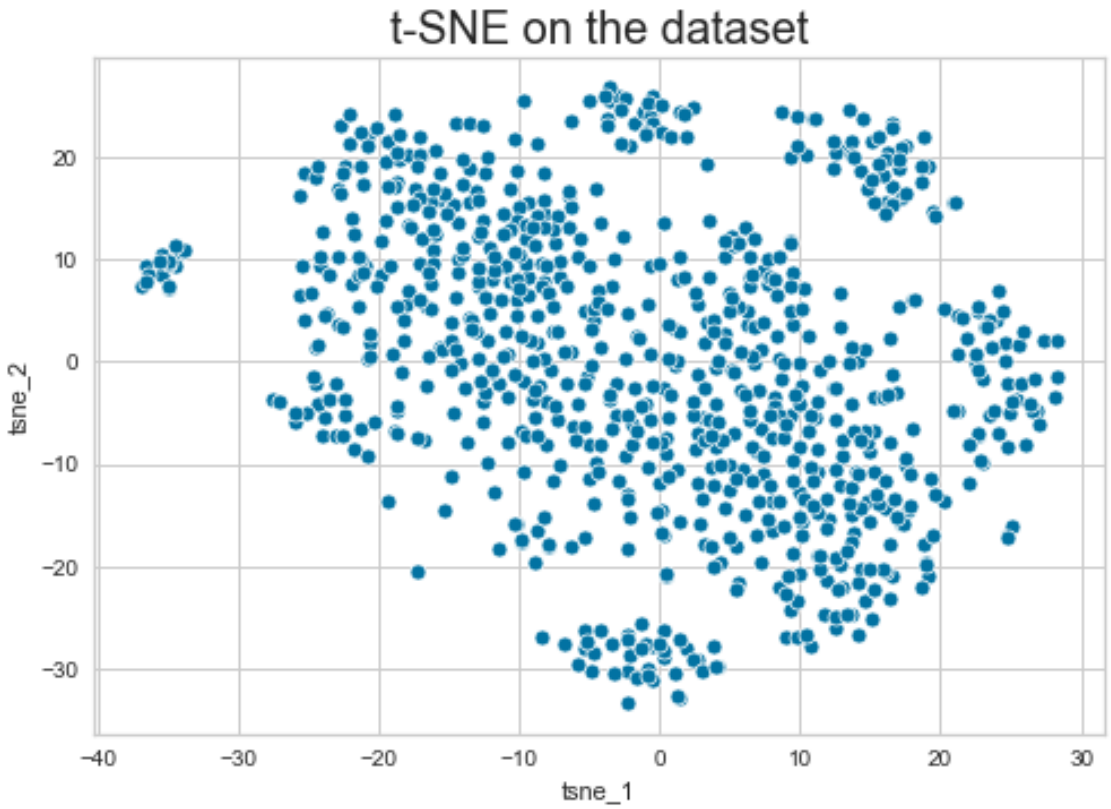
```
Total variance being explained:  0.29183101850920007
```

2.1.3 8

[15]:
```
tsne = TSNE(n_components=2, random_state = 20)
tsne_results = tsne.fit_transform(wiki_std)
tsne_results = pd.DataFrame(tsne_results)
tsne_results = tsne_results.rename(columns = {0:'tsne_1', 1:'tsne_2'})

sns.scatterplot(x='tsne_1', y='tsne_2', palette=sns.color_palette("hls",␣
↪10),data=tsne_results,legend="full")
plt.title('t-SNE on the dataset', size=20)
```

[15]:
```
Text(0.5, 1.0, 't-SNE on the dataset')
```



According to the above plot, we can see that the data are mainly composed by a huge group of observations and several scattered much smaller groups (about 5).According to this fact, we can estimate that clustering algorithm may work well after the data is processed with t-SNE. This phenomenon shows one advantage of t-SNE over PCA: t-SNE are better able to avoid the problem of overlapping. At the same time, since the different groups of data are clearly represented in the above plot, we can find that according to t-SNE as a non-linear algorithm, it can more accurately capture the complex polynomial relationship between features compared to PCA.

2.2 Clustering

2.2.1 9

[16]:
```
wiki_transformed = pd.DataFrame(wiki_transformed)
wiki_transformed = wiki_transformed.rename(columns = {0:'pca_1', 1:'pca_2'})
k2_wiki = KMeans(n_clusters=2, random_state=20).fit(wiki_std)
k2_wiki.cluster_centers_

k3_wiki = KMeans(n_clusters=3, random_state=20).fit(wiki_std)
k3_wiki.cluster_centers_

k4_wiki = KMeans(n_clusters=4, random_state=20).fit(wiki_std)
k4_wiki.cluster_centers_
```

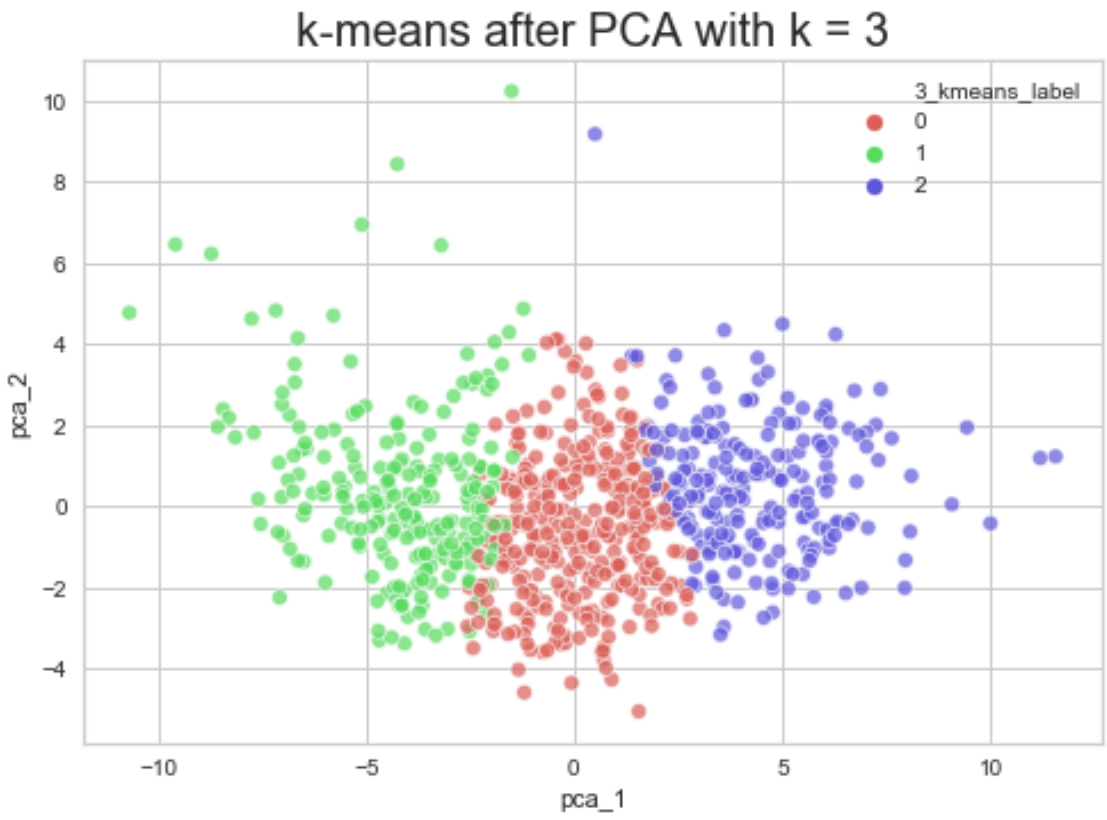
```
pca_analysis = wiki_transformed.copy()
pca_analysis['2_kmeans_label'] = k2_wiki.labels_
pca_analysis['3_kmeans_label'] = k3_wiki.labels_
pca_analysis['4_kmeans_label'] = k4_wiki.labels_
```

[17]:
```
sns.scatterplot(x='pca_1', y='pca_2', palette=sns.color_palette("hls", 2),data␣
↪= pca_analysis, hue = '2_kmeans_label',
                legend="full",alpha = 0.7)
plt.title('k-means after PCA with k = 2', size = 20)
```

[17]: Text(0.5, 1.0, 'k-means after PCA with k = 2')

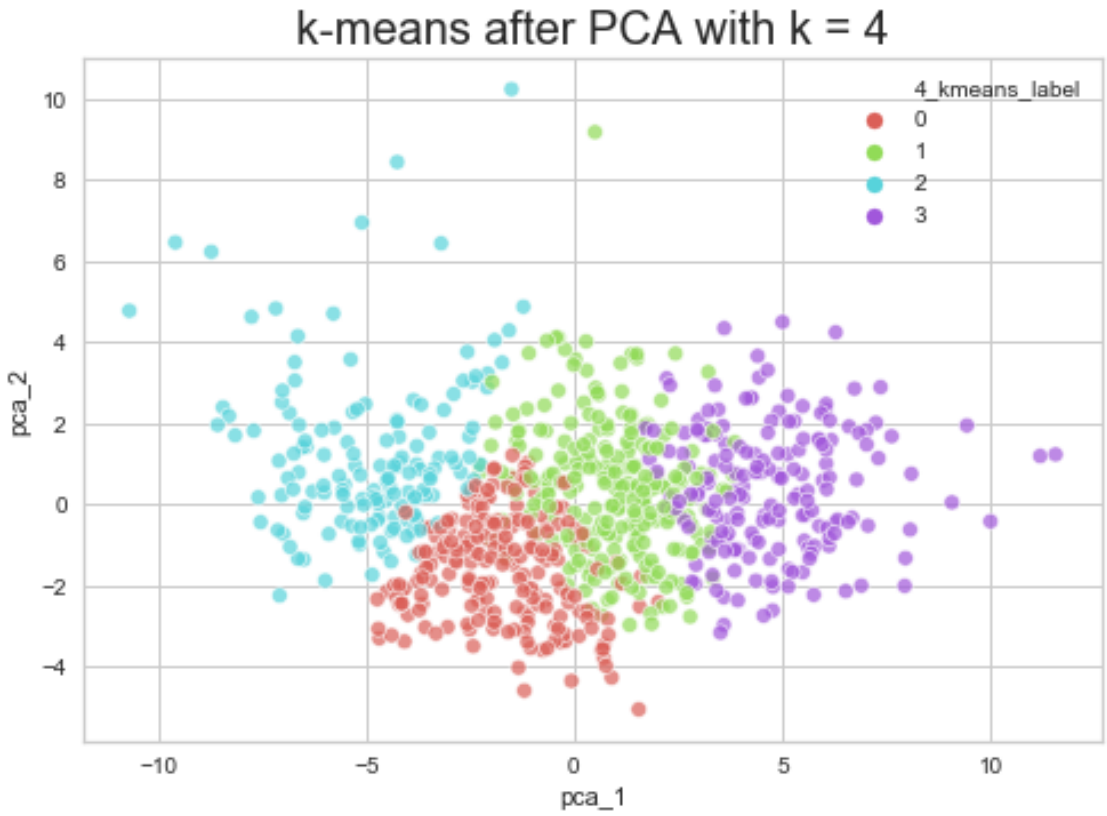
[18]:
```
sns.scatterplot(x='pca_1', y='pca_2', palette=sns.color_palette("hls", 3),data␣
↪= pca_analysis, hue = '3_kmeans_label',
                legend="full",alpha = 0.7)
plt.title('k-means after PCA with k = 3', size = 20)
```

[18]: Text(0.5, 1.0, 'k-means after PCA with k = 3')



[19]:
```
sns.scatterplot(x='pca_1', y='pca_2', palette=sns.color_palette("hls", 4),data␣
↪= pca_analysis, hue = '4_kmeans_label',
                legend="full",alpha = 0.7)
plt.title('k-means after PCA with k = 4', size = 20)
```

[19]:
```
Text(0.5, 1.0, 'k-means after PCA with k = 4')
```

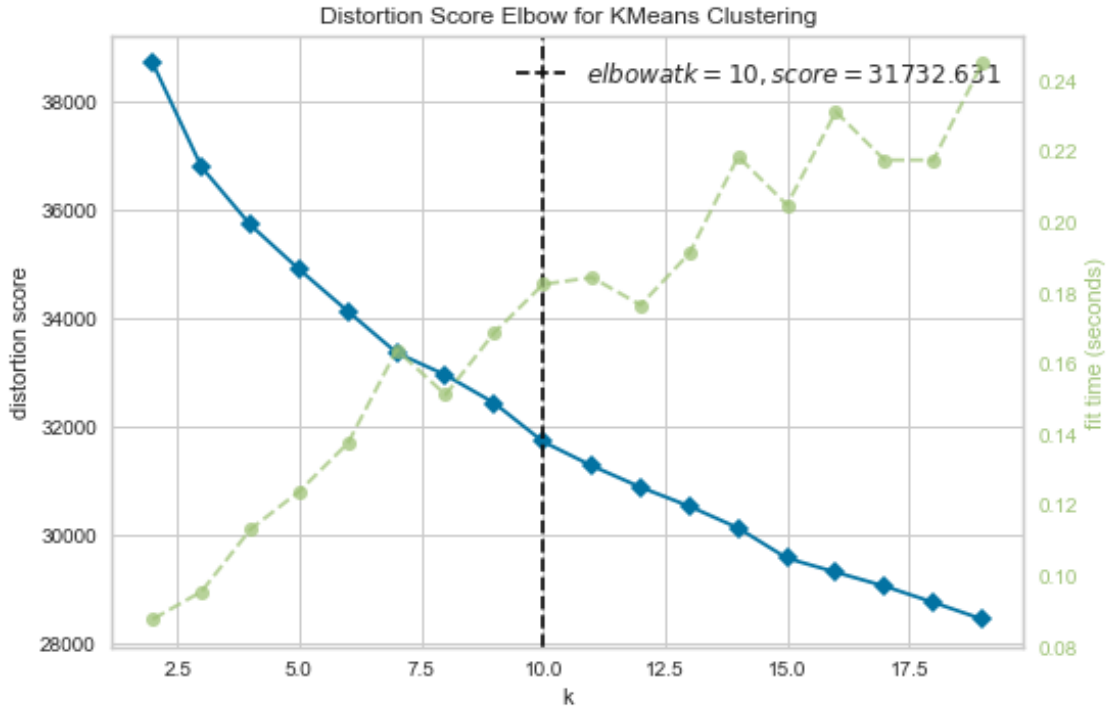


According to the above plots, we can see that using the dimension reduction method of PCA, the clustered groups are very closed to one another, and in some regions they overlap.In addition, we can see that many plots also overlap with other plots. This may give us some hints that we may try another non-linear dimension reduction method.

2.2.2 10

The elbow method

[20]:
```
kelbow_visualizer(KMeans(random_state=20), wiki_std, k=(2,20), locate_elbow =␣
↪True)
```



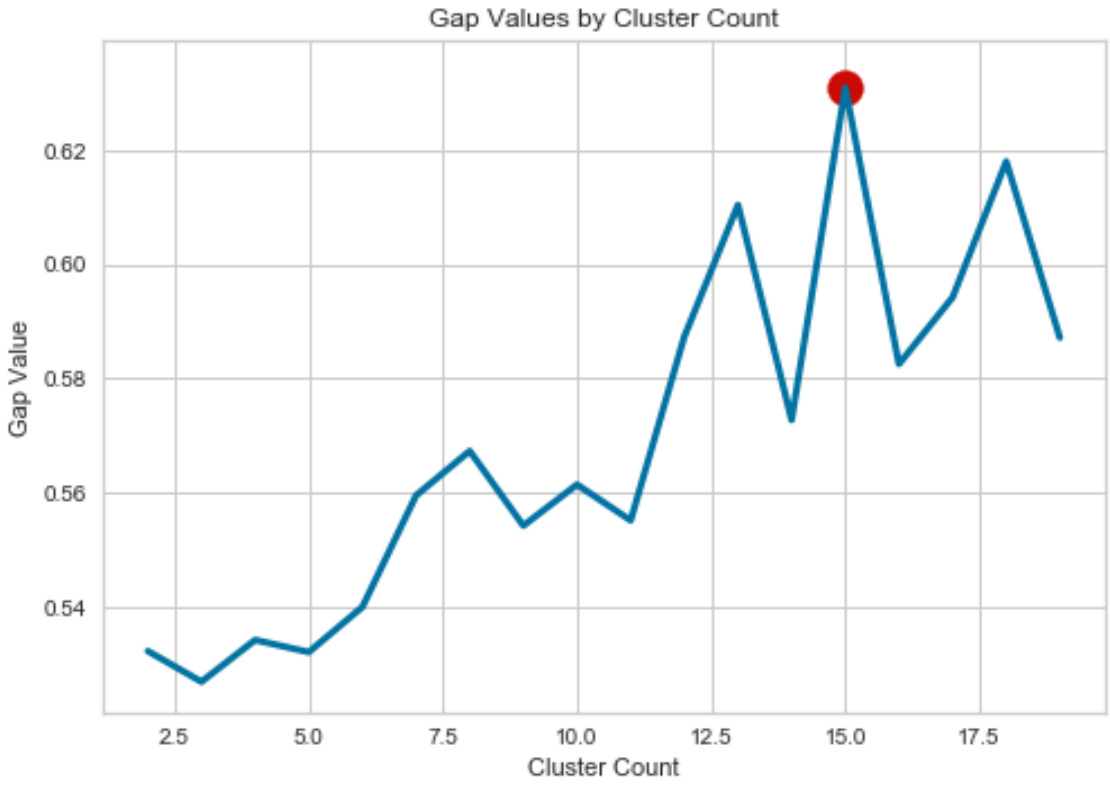
[20]:
```
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at
0x000002AD4B880348>,
                 k=None, locate_elbow=True, metric='distortion', model=None,
                 timings=True)
```

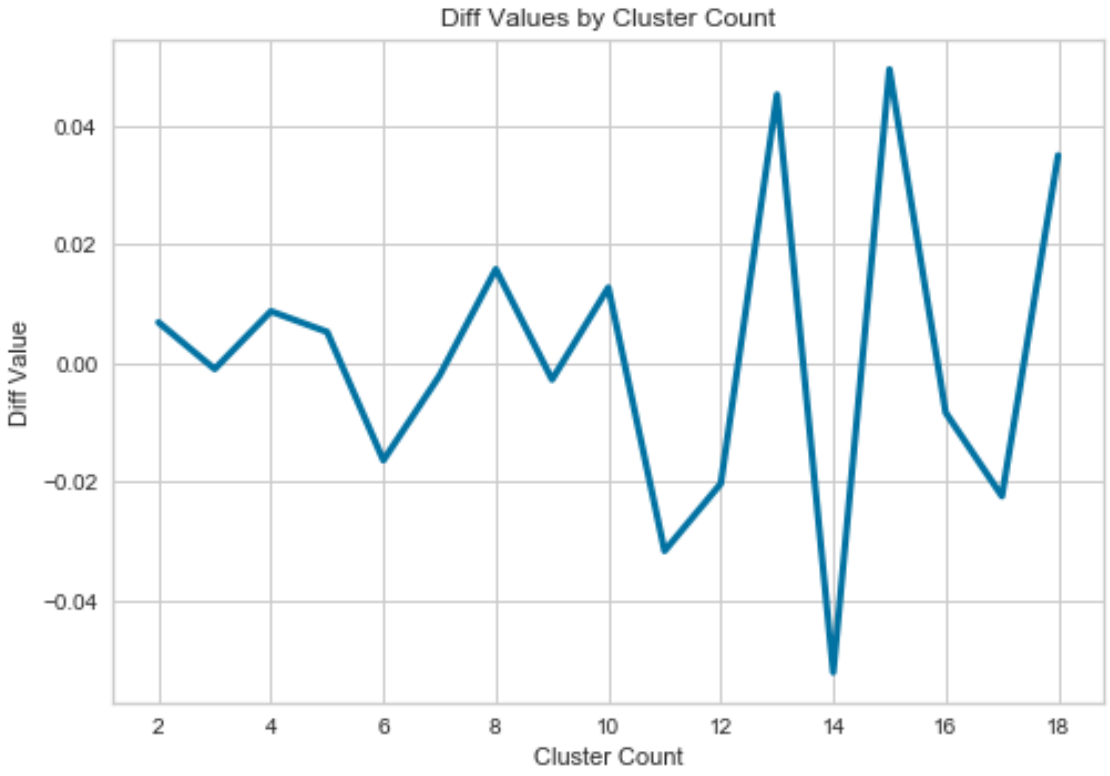
According to the above plot, the optimal k value is 10 if we use the elbow method as the selection standard.

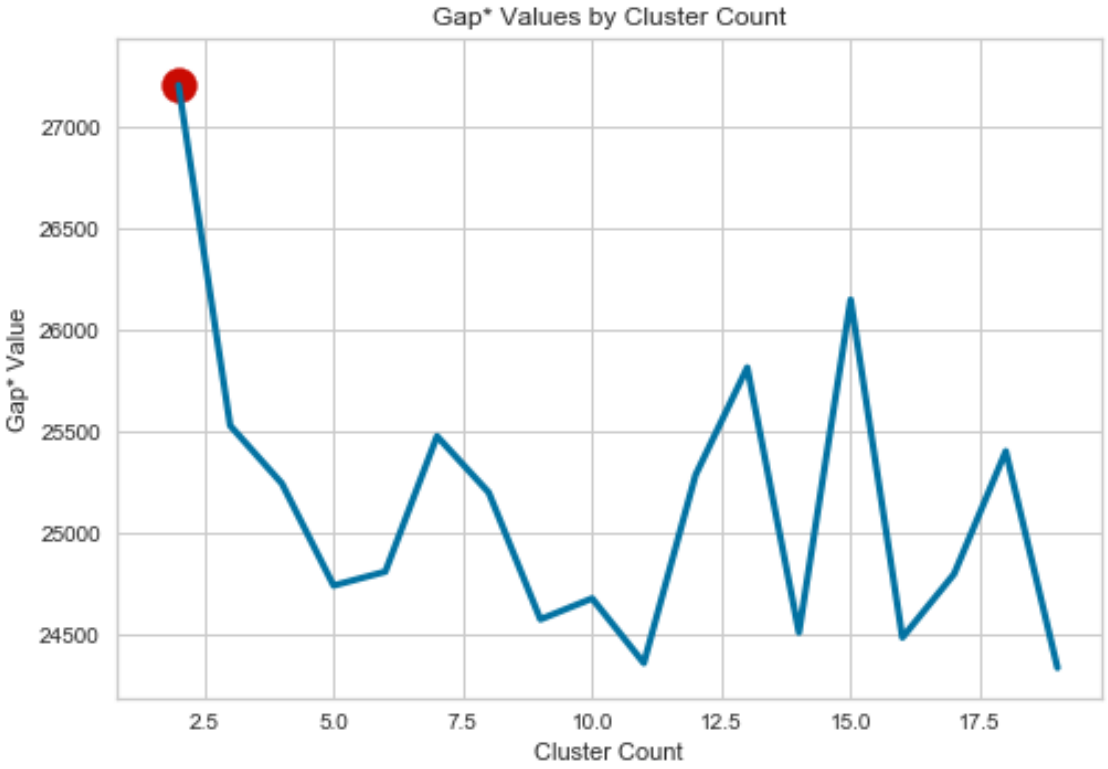
Gap statistic

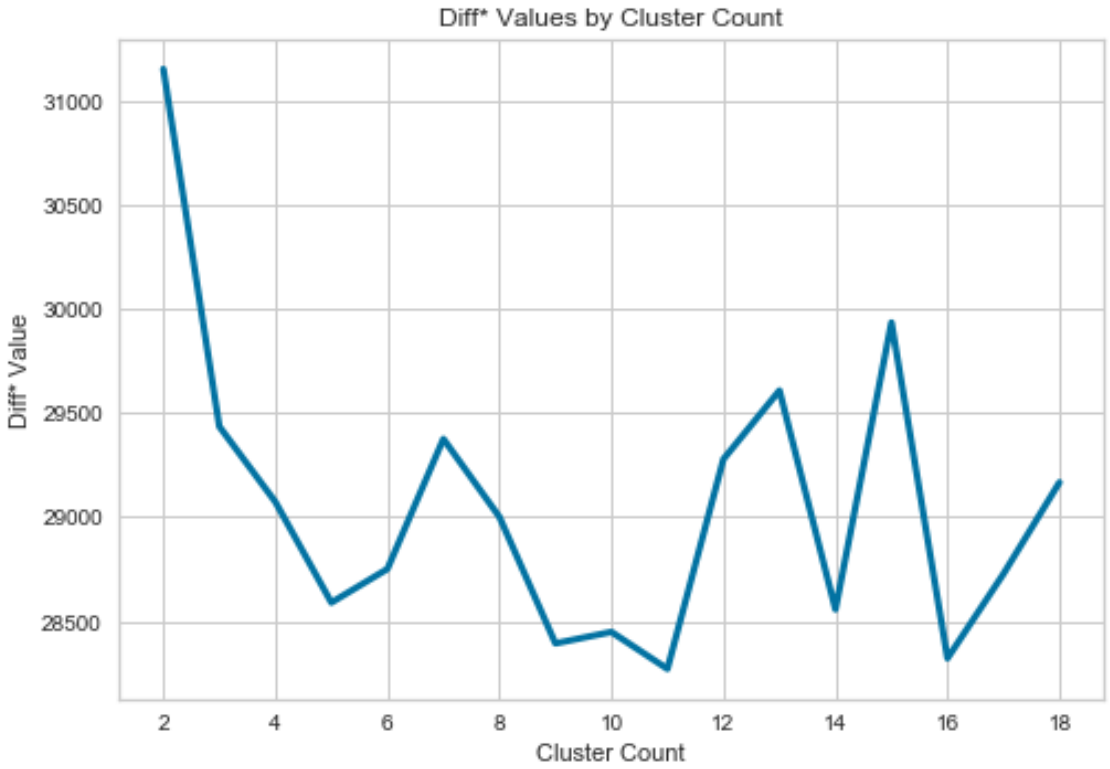
[21]:
```
np.random.seed(124)
opt = optimalK.OptimalK()
k_gap = opt(wiki_std, cluster_array=np.arange(2, 20))
print("According to the above plot, the optimal k value is ", k_gap, " if we␣
 ↪use gap statistic as the selection standard.")
opt.plot_results()
```

```
According to the above plot, the optimal k value is  15  if we use gap statistic
as the selection standard.
```


Gap Values by Cluster Count
Gap Value
0.62
0.60
0.58
0.56
0.54
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Cluster Count

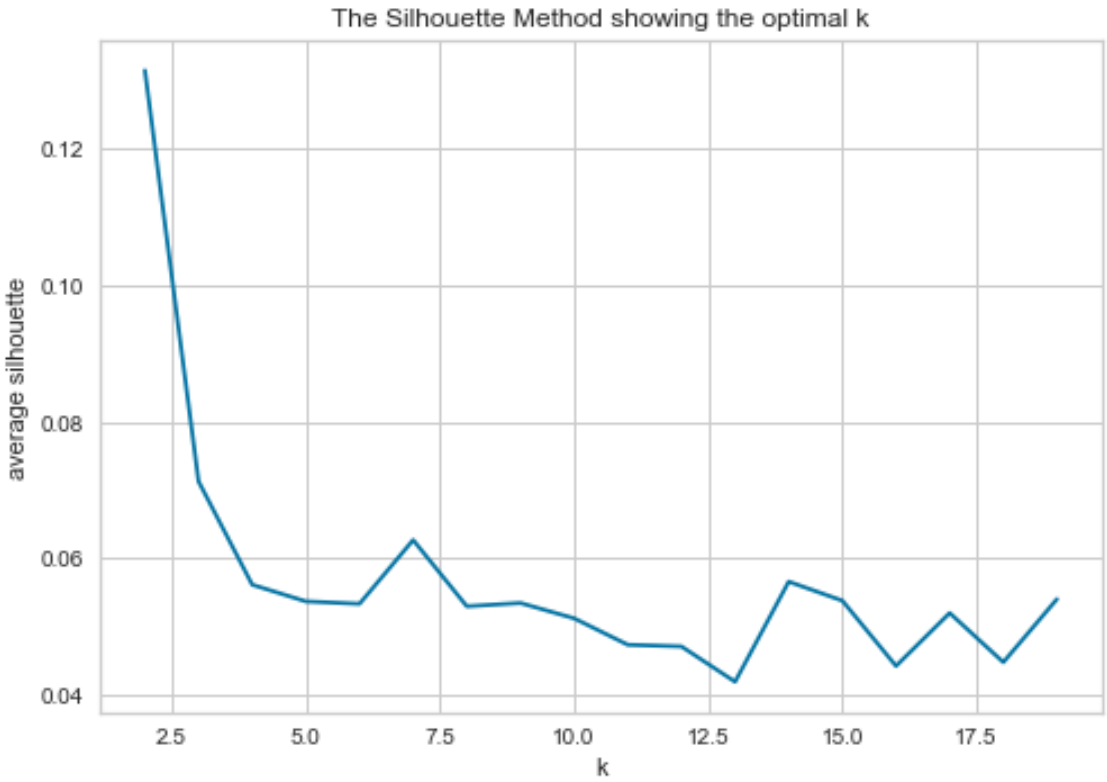

Diff Values by Cluster Count
Diff Value
0.04
0.02
0.00
−0.02
−0.04
2
4
6
8
10
12
14
16
18
Cluster Count


Gap* Values by Cluster Count
Gap* Value
27000
26500
26000
25500
25000
24500
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Cluster Count



Average silhouette

[22]:
```
avg_sil = {}
K = range(2, 20)
for k in K:
    model = KMeans(n_clusters=k, random_state=20)
    preds = model.fit_predict(wiki_std)
    score = silhouette_score(wiki_std, preds, metric = 'euclidean')
    avg_sil[k] = score
avg_sil_values = list(avg_sil.values())
plt.plot(K, avg_sil_values, 'bx-')
plt.xlabel('k')
plt.ylabel('average silhouette')
plt.title('The Silhouette Method showing the optimal k')
plt.show()
```



Since we get two different optimal k values from the two methods (the elbow method and gap statistic), we use average silhouette to pick one of them. Average silhouette measures the compactness of data after clustering. We want to pick the optimal value with higher average silhouette score. If they have the same scores, we prefer the smaller one.

[23]:
```
if avg_sil[10] >= avg_sil[k_gap]:
    chosen_k = 10
else:
    chosen_k = k_gap
print("According to the above standards, we set the value of optimal k as ",␣
 ↪chosen_k)
```

```
According to the above standards, we set the value of optimal k as  15
```

2.2.3 11

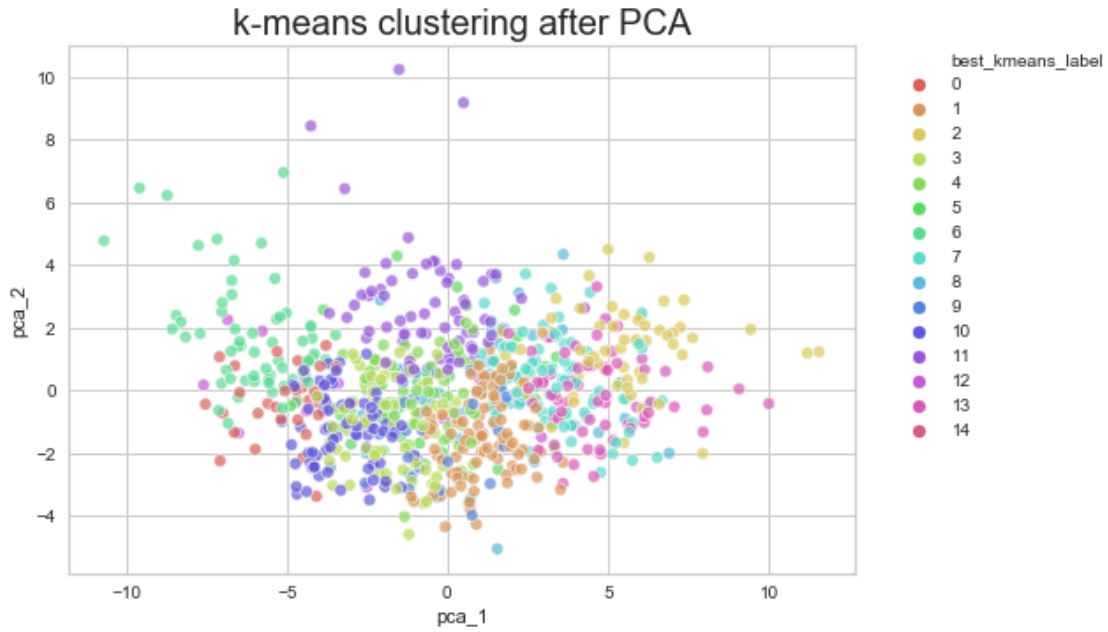
PCA and k-means

[24]:
```
best_wiki = KMeans(n_clusters=chosen_k, random_state=20).fit(wiki_std)
best_wiki.cluster_centers_

pca_analysis_best = wiki_transformed.copy()
pca_analysis_best['best_kmeans_label'] = best_wiki.labels_
```

```
sns.scatterplot(x='pca_1', y='pca_2', palette=sns.color_palette("hls",␣
↪chosen_k), data=pca_analysis_best,
                hue = 'best_kmeans_label', legend="full", alpha = 0.7)
plt.title("k-means clustering after PCA", size = 20)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

[24]: <matplotlib.legend.Legend at 0x2ad4d88e688>

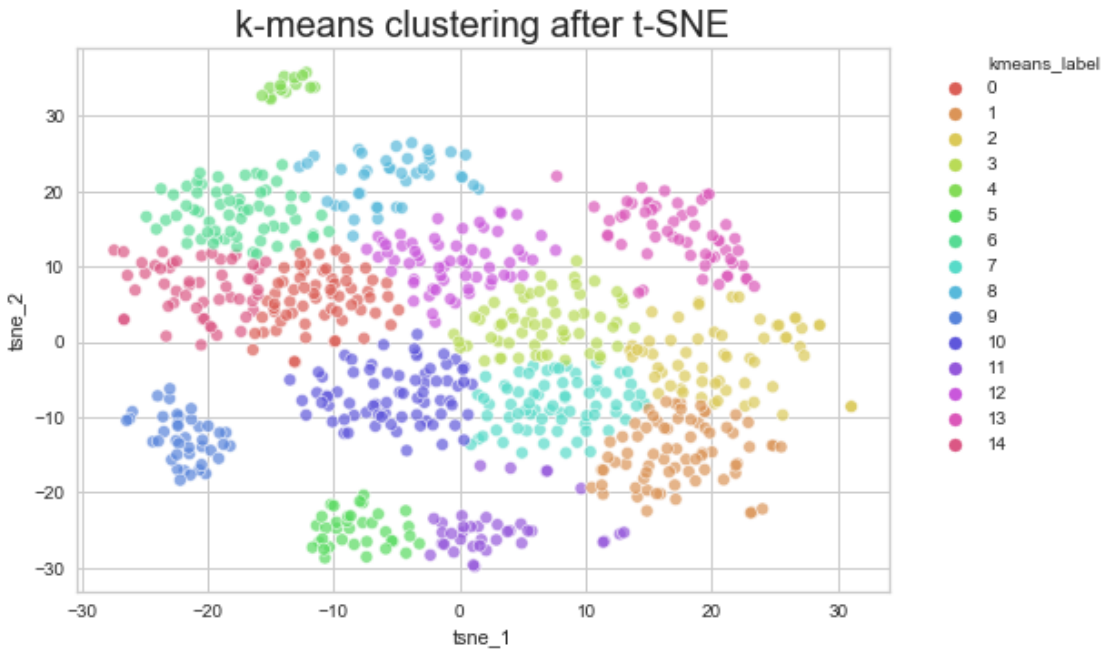


t-SNE and k-means

[25]:
```
best_tsne = TSNE(n_components=2)
tsned_wiki = best_tsne.fit_transform(wiki_std)
tsned_wiki = pd.DataFrame(tsned_wiki)
tsned_wiki = tsned_wiki.rename(columns = {0:'tsne_1', 1:'tsne_2'})
final_wiki = KMeans(n_clusters=chosen_k, random_state=20).fit(tsned_wiki)
final_tsne_ks = tsned_wiki.copy()
final_tsne_ks['kmeans_label'] = final_wiki.labels_

sns.scatterplot(x='tsne_1', y='tsne_2',palette=sns.color_palette("hls",␣
↪chosen_k), data = final_tsne_ks,
                hue = 'kmeans_label', legend="full", alpha = 0.7)
plt.title("k-means clustering after t-SNE", size = 20)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

[25]: <matplotlib.legend.Legend at 0x2ad4d817b88>



The good thing is that the clustering after t-SNE does not have much overlapped boundaries, but the clustering after PCA has a lot of overlapped points. According to the above plots, we can see that the clustering after t-SNE has clearer boundaries. The clustering effect is better when we apply t-SNE instead of PCA. In other words, the groups are more clearly presented in the clustering after t-SNE.It means that t-SNE can better deal with the issue of projecting points of high dimensions than PCA. In terms of data with high dimensions, PCA will place observations in very near locations, while t-SNE can tackle this problem. t-SNE is more capable of setting observations apart.However, we can find that t-SNE costs more time than PCA.