Rim_Nak Won_HW07

March 15, 2020

Problem Set 7

MACS 30100 Winter 2020

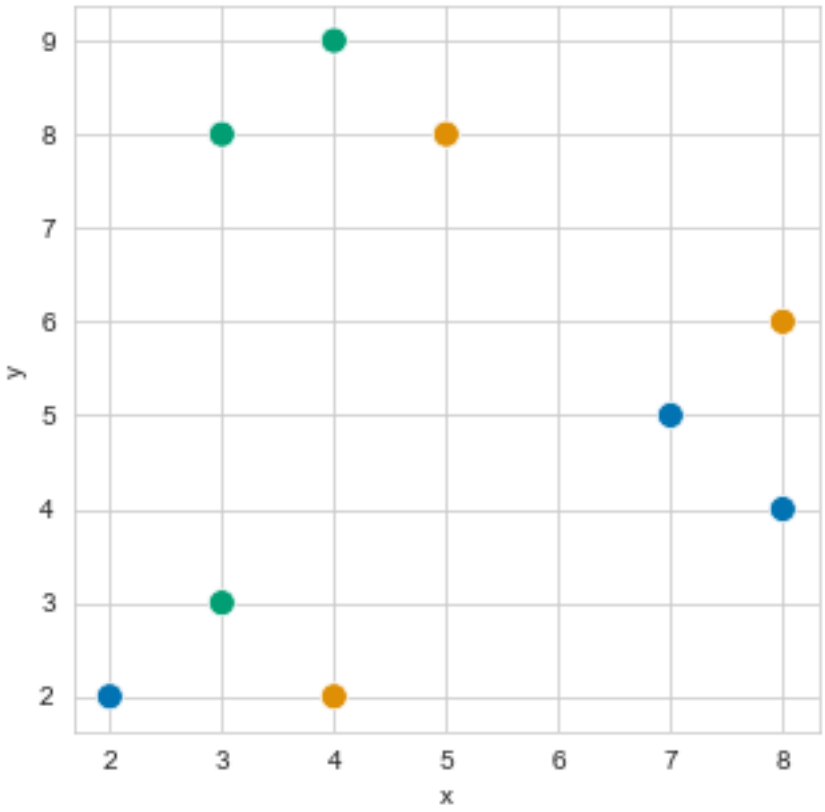
Nak Won Rim

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

sns.set_style("whitegrid")
```

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

[2]:
```
np.random.seed(40)
df = pd.DataFrame({'x': [5,8,7,8,3,4,2,3,4,5],
                   'y': [8,6,5,4,3,2,2,8,9,8],
                   'c': np.random.choice(3, 10)})
plt.figure(figsize = (5,5))
sns.scatterplot(x='x', y='y', hue='c', data=df, palette='colorblind',␣
↪legend=False, s=100);
```



From inspection, my intuition tells me that we have three clusters, one on the top lef, another in bottom left and the other in middle right.

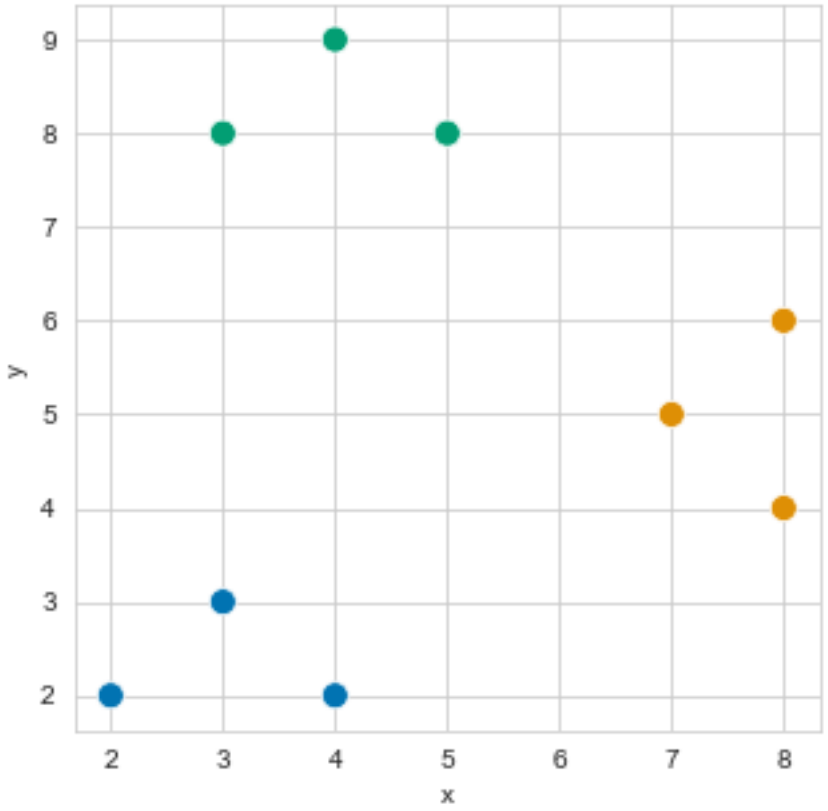
2. Compute the cluster centroid and update cluster assignments for each observation iteratively based on spatial similarity.

[3]:
```
def kmean(df, k):
    changed = True
    while changed:
        changed = False
        centroid = {i: (df[df['c'] == i]['x'].mean(),
                        df[df['c'] == i]['y'].mean()) for i in range(k)}
        for i in df.index:
            min_dist = 9999
            label = None
            for l, j in centroid.items():
                dist = (df.loc[i, 'x'] - j[0]) ** 2 + (df.loc[i, 'y'] - j[1])␣
↪** 2
                if dist < min_dist:
                    min_dist = dist
                    label = l
```

```
            if df.loc[i, 'c'] != label:
                changed = True
                df.loc[i, 'c'] = label
    return centroid
centroid = kmean(df, 3)
```

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

[4]:
```
plt.figure(figsize = (5,5))
sns.scatterplot(x='x', y='y', hue='c', data=df, palette='colorblind',␣
↪legend=False, s=100);
```

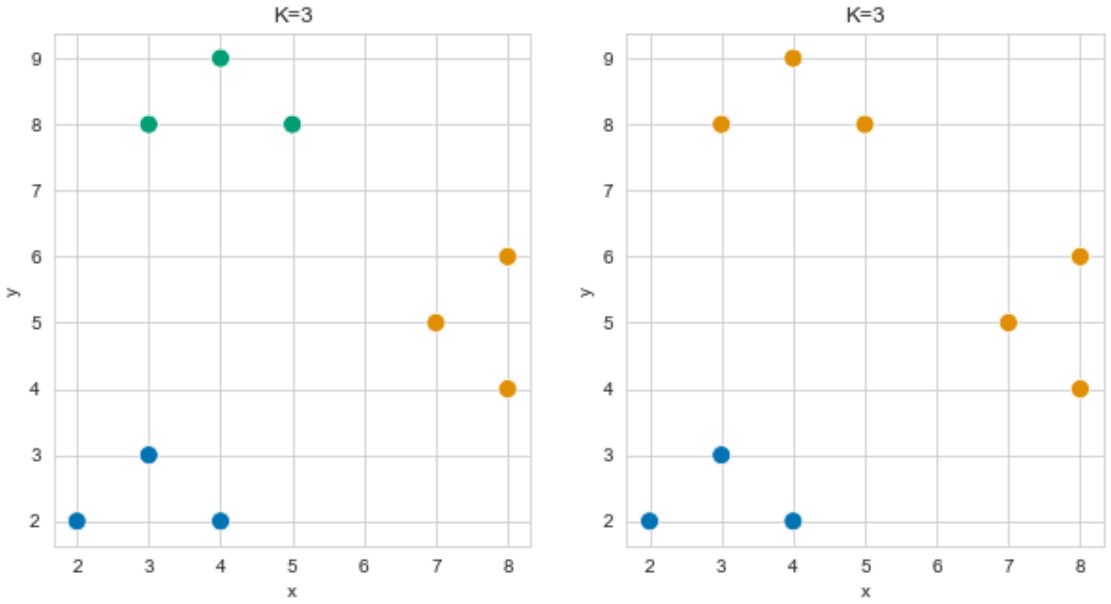


The k-means clustering gives the same conclusion as the inital intuition I had upon inspecting the plot in 1. We ended up with three clusters, one on the top lef, another in bottom left and the other in middle right.

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

[5]:
```
np.random.seed(40)
df2 = pd.DataFrame({'x': [5,8,7,8,3,4,2,3,4,5],
                    'y': [8,6,5,4,3,2,2,8,9,8],
```

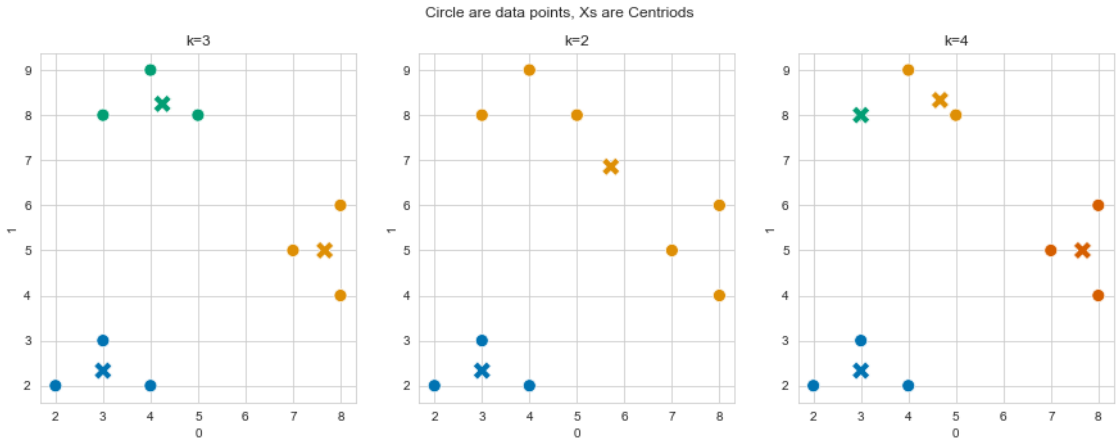
```
                    'c': np.random.choice(2, 10)})
centroid2 = kmean(df2, 2)
fig, ax = plt.subplots(ncols=2, figsize=(10, 5))
sns.scatterplot(x='x', y='y', hue='c', data=df, palette='colorblind',  s=100,␣
 ↪legend=False, ax=ax[0])
ax[0].set_title('K=3')
sns.scatterplot(x='x', y='y', hue='c', data=df2, palette='colorblind', s=100,␣
 ↪legend=False, ax=ax[1])
ax[1].set_title('K=3');
```



5. 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?

[6]:
```
np.random.seed(40)
df3 = pd.DataFrame({'x': [5, 8, 7, 8, 3, 4, 2, 3, 4, 5],
                    'y': [8, 6, 5, 4, 3, 2, 2, 8, 9, 8],
                    'c': np.random.choice(4, 10)})
centroid3 = kmean(df3, 4)
c1 = pd.DataFrame(centroid).T
c1['c'] = [0, 1, 2]
c2 = pd.DataFrame(centroid2).T
c2['c'] = [0, 1]
c3 = pd.DataFrame(centroid3).T
c3['c'] = [0, 1, 2, 3]
fig, ax = plt.subplots(ncols=3, figsize=(15, 5))
sns.scatterplot(x='x', y='y', hue='c', data=df, palette='colorblind',␣
 ↪legend=False, s=100, ax=ax[0])
```

```
sns.scatterplot(x=0, y=1, hue='c', data=c1, marker='X', palette='colorblind',␣
 ↪s=200, ax=ax[0], legend=False)
ax[0].set_title('k=3')
sns.scatterplot(x='x', y='y', hue='c', data=df2, palette='colorblind',␣
 ↪legend=False, s=100, ax=ax[1])
sns.scatterplot(x=0, y=1, hue='c', data=c2, marker='X', palette='colorblind',␣
 ↪s=200, ax=ax[1], legend=False)
ax[1].set_title('k=2')
sns.scatterplot(x='x', y='y', hue='c', data=df3, palette='colorblind',␣
 ↪legend=False, s=100, ax=ax[2])
sns.scatterplot(x=0, y=1, hue='c', data=c3, marker='X', palette='colorblind',␣
 ↪s=200, ax=ax[2], legend=False)
ax[2].set_title('k=4')
fig.suptitle('Circle are data points, Xs are Centriods');
```



[7]:
```
print('mean distance from centroid k=2:', df2.apply(lambda r: ((r['x'] -␣
 ↪centroid2[r['c']][0]) ** 2 + (r['x'] - centroid2[r['c']][0]) ** 2) ** 0.5,␣
 ↪axis=1).mean())
print('mean distance from centroid k=3:', df.apply(lambda r: ((r['x'] -␣
 ↪centroid[r['c']][0]) ** 2 + (r['x'] - centroid[r['c']][0]) ** 2) ** 0.5,␣
 ↪axis=1).mean())
print('mean distance from centroid k=4:', df3.apply(lambda r: ((r['x'] -␣
 ↪centroid3[r['c']][0]) ** 2 + (r['x'] - centroid3[r['c']][0]) ** 2) ** 0.5,␣
 ↪axis=1).mean())
```

```
mean distance from centroid k=2: 1.9394928855402447
mean distance from centroid k=3: 0.8956685895029601
mean distance from centroid k=4: 0.6599663291074442
```

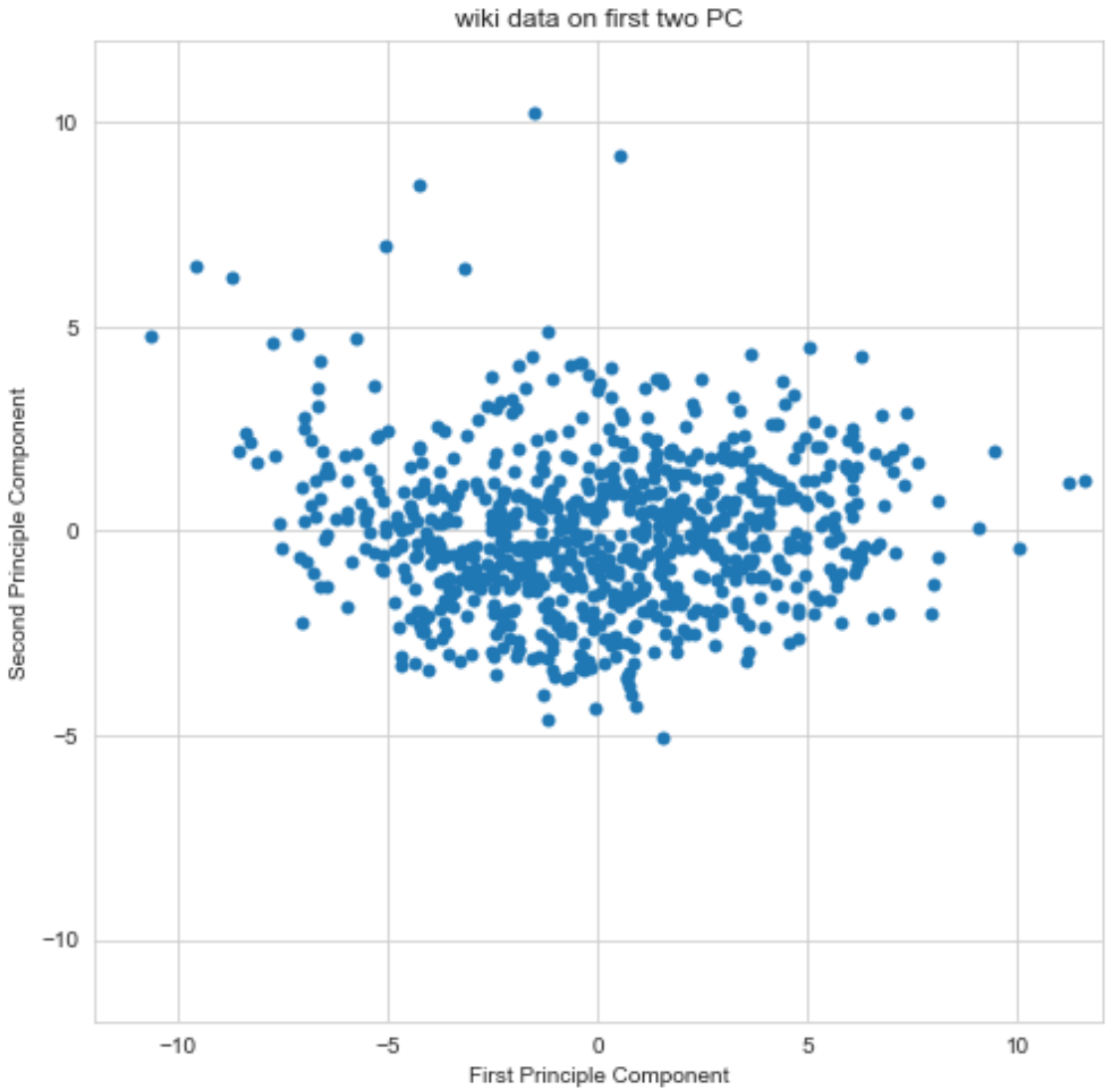
As shown in the above plot, $k = 3$ fits the data better compared to $k = 2$ or $k = 4$. When $k = 2$, two clusters in the $k = 3$ clusters gets grouped together, so the centroid gets placed in a weird position where it is just the middle point of the clusters. This increases the mean distance of data

points to the centroid a lot, implying that this is not a good number of k. On the other hand, when $k = 4$, one of the clusters just consist of one point, and a centroid is placed on top of that point. Although the mean distance decreases, this is a bad number of k since it is clearly overfitting the data.

6. 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?

[8]:
```
# loading data
wiki = pd.read_csv('data/wiki.csv')
# scaling the data
cols = wiki.columns
wiki = pd.DataFrame(StandardScaler().fit_transform(wiki), columns=cols)
# PCA
pca = PCA(random_state=40)
pca_ = pca.fit_transform(wiki)
pca_df = pd.DataFrame(data = pca_)
plt.figure(figsize = (8,8))
sns.scatterplot(x=0, y=1, data=pca_df, linewidth=0)
plt.xlabel('First Principle Component')
plt.ylabel('Second Principle Component')
plt.title('wiki data on first two PC')
plt.xlim(-12, 12)
plt.ylim(-12, 12);
```



[9]:
```
loadings = pd.DataFrame({'variable': cols,'PC1': pca.components_[0], 'PC2': pca.
 ↪components_[1]})
loadings['PC1_abs'] = loadings['PC1'].apply(abs)
loadings['PC2_abs'] = loadings['PC2'].apply(abs)
print('top 5 variables strongly correlated on the first principal component\n',␣
 ↪loadings.sort_values(by='PC1_abs', ascending=False).head().iloc[:,:3])
print('top 5 variables strongly correlated on the second principal␣
 ↪component\n', loadings.sort_values(by='PC2_abs', ascending=False).head().
 ↪iloc[:,:3])
```

```
top 5 variables strongly correlated on the first principal component
    variable       PC1       PC2
38      bi2  0.230924  0.083431
```

```
37      bi1  0.226193  0.056374
29     use3  0.218809  0.155152
30     use4  0.214558  0.160865
7       pu3  0.210863  0.028776
top 5 variables strongly correlated on the second principal component
    variable       PC1       PC2
8      peu1  0.061228 -0.271741
39     inc1  0.104667 -0.245440
26      sa3  0.120376 -0.242325
24      sa1  0.121658 -0.229926
46     exp4  0.099873  0.228494
```

One thing we can notice from the above plot is that the observations seems to vary more by the first principle component than the second principle component. Most of observations are varied within (-10, 10) of the first principle component, while most of observations are varied within (-5, 5) of the second principle component. This shows that the first principle component is capturing more variance than the second principle component.

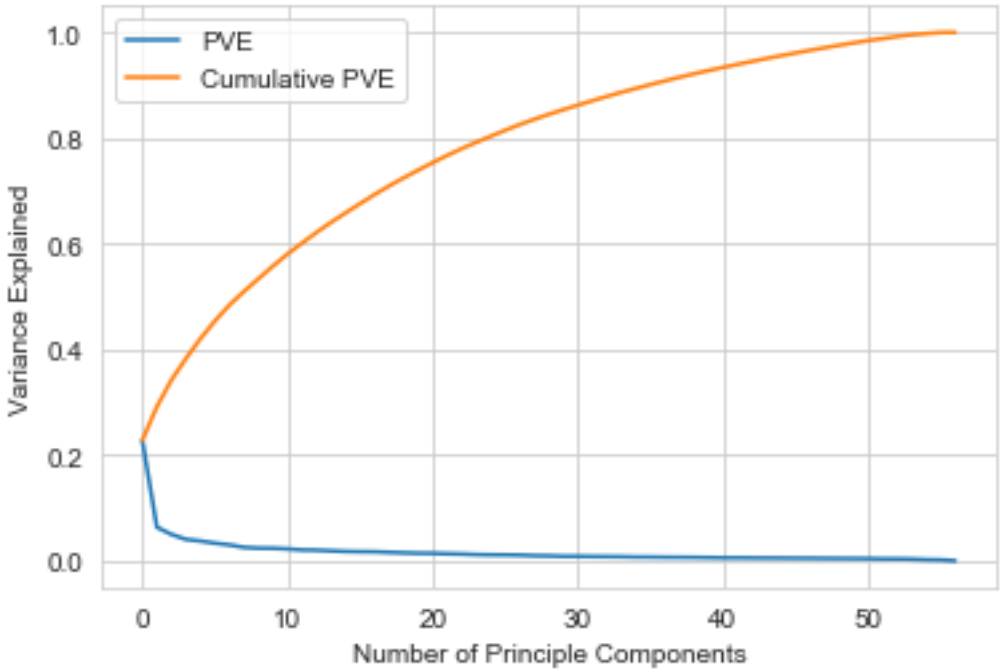
Top 5 variables strongly correlated on the first principal component were `bi2`, `bi1`, `use3`, `use4`, `pu3`. Top 5 variables strongly correlated on the second principal component were `peu1`, `inc1`, `sa3`, `sa1`, `exp4`. As we can see, the variables that are strongly related to each components are quite different. This is natural because the principle components are orthogonal (uncorrelated) to each other.

7. 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?

[10]:
```
print('variance explained by first principle component:', pca.
 ↪explained_variance_ratio_[0])
print('variance explained by second principle component:', pca.
 ↪explained_variance_ratio_[1])
print('variance explained by first two principle components:', pca.
 ↪explained_variance_ratio_[:2].sum())
```

```
variance explained by first principle component: 0.22810627785663384
variance explained by second principle component: 0.06372474541820296
variance explained by first two principle components: 0.2918310232748368
```

[11]:
```
cpve = [pca.explained_variance_ratio_[:i + 1].sum() for i in range(57)]
ex_var = pd.DataFrame({'PC':[i for i in range(57)], 'PVE':pca.
 ↪explained_variance_ratio_,
                       'CPVE': cpve})
sns.lineplot(x='PC', y='PVE', data=ex_var)
sns.lineplot(x='PC', y='CPVE', data=ex_var)
plt.xlabel('Number of Principle Components')
plt.ylabel('Variance Explained')
plt.legend(['PVE','Cumulative PVE']);
```



The proportion of variance explained by the first principle component was about 0.2281. The proportion of variance explained by the second principle component was about 0.0637. The proportion of variance explained by the first two principle components was about 0.2918. This suggest that there are quite a lot of variance unexplained by the first two principle components. We might want to use more principle components to capture more variance in the data.

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

Since it was unspecified, I used the default parameter for t-SNE in SKLearn (`n_component` $= 2$, `perflexity` $= 30$, etc).

[12]:
```
tsne = TSNE(random_state=40).fit_transform(wiki)
tsne_df = pd.DataFrame(tsne, columns=['Dimension 1', 'Dimension 2'])
plt.figure(figsize=(8,8))
sns.scatterplot(x='Dimension 1', y='Dimension 2', data=tsne_df, linewidth=0)
plt.title('wiki data on t-SNE');
```

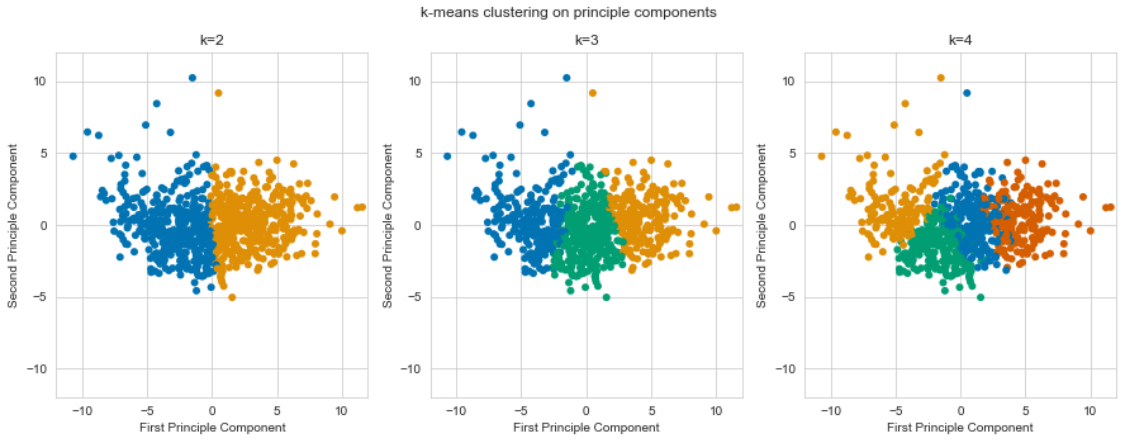
We can see that the there is a large cluster along the line of y (or dimension 2) = - x (or dimension 1), and three tiny clusters in the below that large cluster. However, since the size of the small clusters are so smaller than the large cluster, I do not think this is a significant clustering. I conclude that there does not seems to be a obvious clustering happening, at least to naked human eye. Even if clustering algorithm can find a clustering in this, I do not think it will be easily interpretable.

9. 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.

I note that I already did the scaling in question 7, so there is no additional scaling happening in this section (I am using the data from question 7, 8).

[13]:
```
pca_df['kmeans_2'] = KMeans(n_clusters=2, random_state=40).fit(wiki).labels_
pca_df['kmeans_3'] = KMeans(n_clusters=3, random_state=40).fit(wiki).labels_
pca_df['kmeans_4'] = KMeans(n_clusters=4, random_state=40).fit(wiki).labels_
```

[14]:
```
fig, ax = plt.subplots(ncols=3, figsize = (15, 5))
sns.scatterplot(x=0, y=1, data=pca_df, hue='kmeans_2', palette='colorblind',␣
 ↪linewidth=0, legend=False, ax=ax[0])
ax[0].set_xlabel('First Principle Component')
ax[0].set_ylabel('Second Principle Component')
ax[0].set_title('k=2')
ax[0].set_xlim(-12, 12)
ax[0].set_ylim(-12, 12)
sns.scatterplot(x=0, y=1, data=pca_df, hue='kmeans_3', palette='colorblind',␣
 ↪linewidth=0, legend=False, ax=ax[1])
ax[1].set_xlabel('First Principle Component')
ax[1].set_ylabel('Second Principle Component')
ax[1].set_title('k=3')
ax[1].set_xlim(-12, 12)
ax[1].set_ylim(-12, 12)
sns.scatterplot(x=0, y=1, data=pca_df, hue='kmeans_4', palette='colorblind',␣
 ↪linewidth=0, legend=False, ax=ax[2])
ax[2].set_xlabel('First Principle Component')
ax[2].set_ylabel('Second Principle Component')
ax[2].set_title('k=4')
ax[2].set_xlim(-12, 12)
ax[2].set_ylim(-12, 12)
fig.suptitle('k-means clustering on principle components');
```



One thing to note is that the first principle component seems to be important in the clustering, since the borderline of clusters seems to be quite related to the first principle component. This intuitively makes sense because there is relatively more variance alongside the first principle component than the second principle component. Also, $k = 2$ and $k = 3$ seems to be doing a little better in clustering

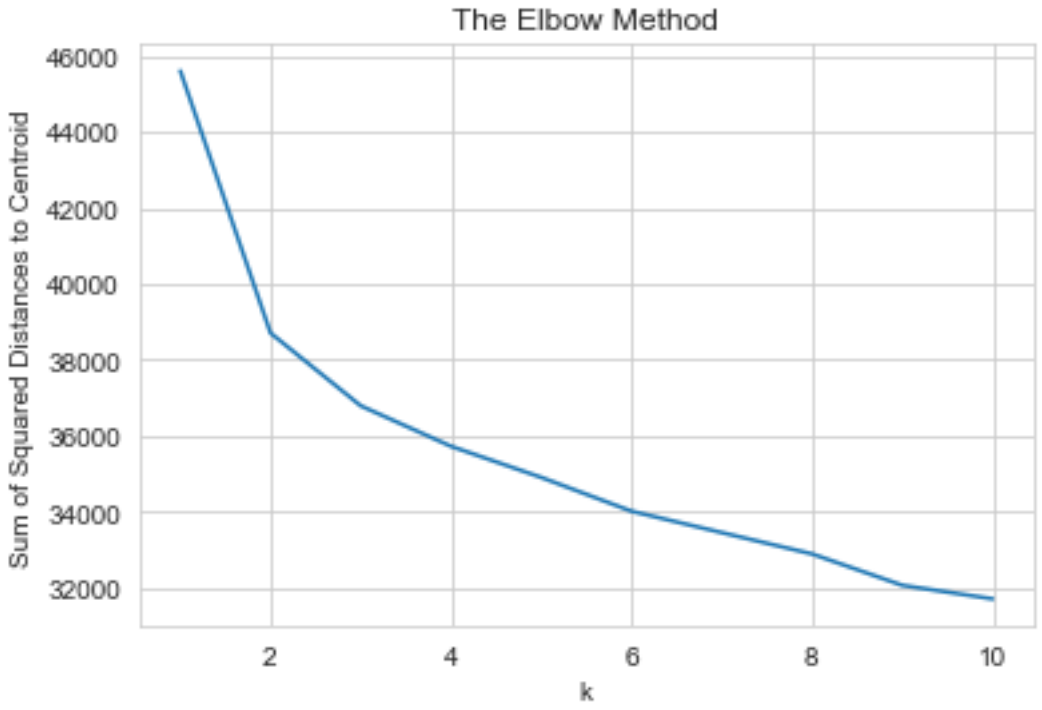
than $k = 4$, since the boundary is more clear in these two.

10. 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.

I note that I was not able to find a good implementation of gap statistics in python, so I am not using gap statistics.

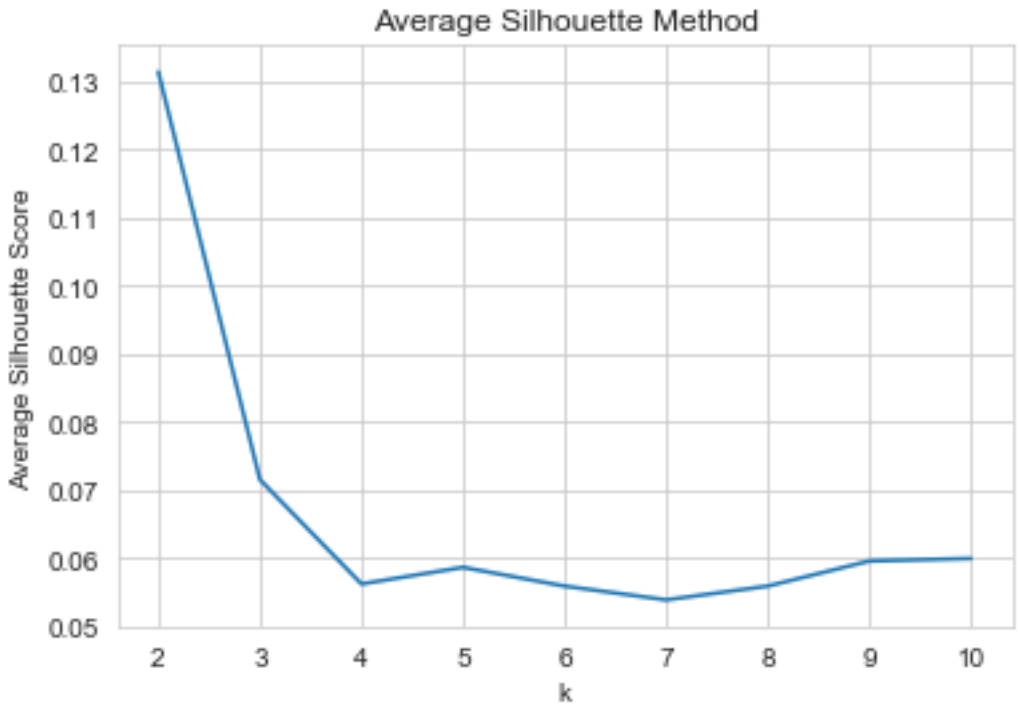
- The Elbow Method

[15]:
```
ssd = [KMeans(n_clusters=i, random_state=40).fit(wiki).inertia_ for i in␣
↪range(1,11)]
sns.lineplot(x=range(1,11), y=ssd)
plt.title('The Elbow Method')
plt.ylabel('Sum of Squared Distances to Centroid')
plt.xlabel('k');
```



- Average Silhouette Method

[16]:
```
shs = [silhouette_score(wiki, KMeans(n_clusters=i, random_state=40).fit(wiki).
↪predict(wiki)) for i in range(2,11)]
sns.lineplot(x=range(2,11), y=shs)
plt.title('Average Silhouette Method')
plt.xlabel('k')
plt.ylabel('Average Silhouette Score');
```



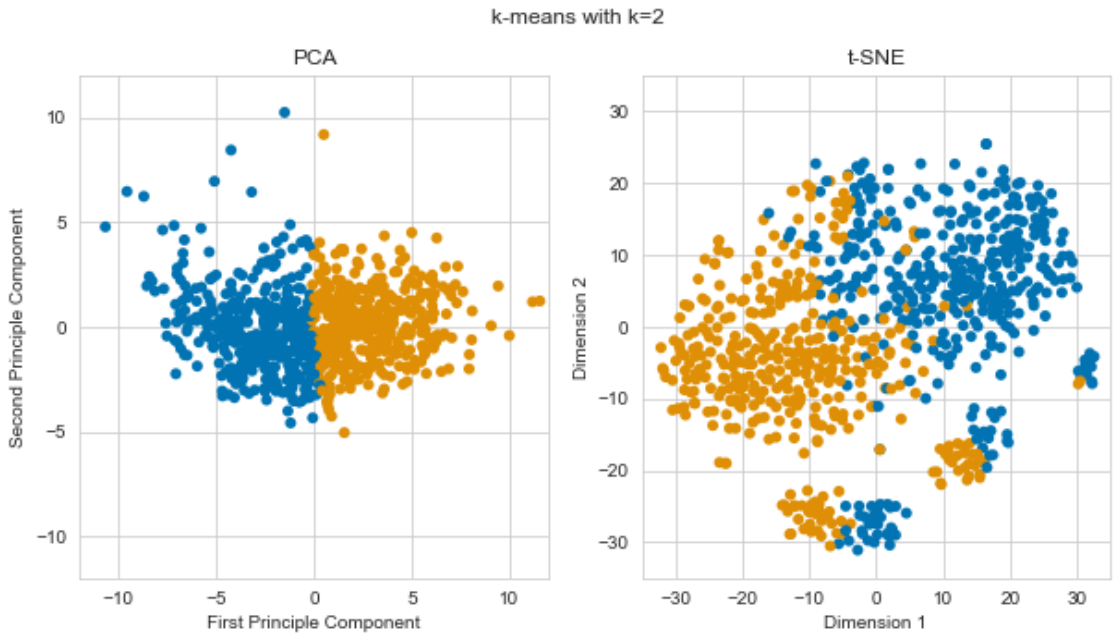
According to the elbow method, the optimal size of k seems to be 2 or 3, since it looks like the plot is giving 'the elbow' in that point (the decrease in sum of sqaured distance becomes more linear). According to average shilhouette method, the optimal size of k seems to be 2, because it shows the highest average silhouette score. Based on this I conclude the optimal number of clusters based on k-means clustering with scaled features is 2.

11. 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?

[17]:
```
tsne_df['kmeans_2'] = KMeans(n_clusters=2, random_state=40).fit(wiki).labels_

fig, ax = plt.subplots(ncols=2, figsize=(10,5))
sns.scatterplot(x=0, y=1, data=pca_df, hue='kmeans_2', palette='colorblind',␣
↪linewidth=0, legend=False, ax=ax[0])
ax[0].set_xlabel('First Principle Component')
ax[0].set_ylabel('Second Principle Component')
ax[0].set_title('PCA')
ax[0].set_xlim(-12, 12)
ax[0].set_ylim(-12, 12)
sns.scatterplot(x='Dimension 1', y='Dimension 2', hue='kmeans_2', legend=False,␣
↪data=tsne_df, linewidth=0, palette='colorblind', ax=ax[1])
ax[1].set_title('t-SNE')
```

```
ax[1].set_xlim(-35, 35)
ax[1].set_ylim(-35, 35)
fig.suptitle('k-means with k=2');
```



PCA seems to show much clear boundary than t-SNE. The boundary shown in PCA is almost linear, while boundary shown in t-SNE is much more complicated and fuzzy. Result from t-SNE corroborate that the small cluster shown in t-SNE (discussed in question 8) was not very significant, since the small clusters are not grouped together in k means.

One explanation why this result happened could be that t-SNE focuses more in the probability of one point being neighbor to one point individually, while PCA focuses on the overall variance of the data points. In other words, t-SNE reduced the dimension focusing more locally on data points (and its neighbors) while PCA reduced the dimension in more global scale. This difference in scope might be the difference between the difference in boundary, considering that k-means uses spatial distance for clustering.

Another explanation could be that PCA is a linear dimension reduction technique while t-SNE is a non-linear dimension reduction technique. The inherent latency inside non-linear dimension reduction technique might introduce some ambiguity to the dimensions, making it harder to interpret or use clustering algorithm on.