In [14]:
```
import pandas as pd
import numpy as np
import math
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
import warnings
warnings.filterwarnings('ignore')
```

k-Means Clustering "By Hand"

You fielded an experiment and collected observations for 10 respondents across two features. The data are:

input_1 = c(5,8,7,8,3,4,2,3,4,5)

input_2 = c(8,6,5,4,3,2,2,8,9,8)

After inspecting your data, you suspect 3 clusters likely characterize these data, but you'd like to check your intuition. Perform k-means clustering "by hand" on these data, initializing at k = 3. Be sure to set the seed for reproducibility. Specifically:

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

In [19]:
```
np.random.seed(666)
```

In [20]:
```
x1_x2_3 = pd.DataFrame()
```

```
x1_x2_3['x1'] = [5,8,7,8,3,4,2,3,4,5]
x1_x2_3['x2'] = [8,6,5,4,3,2,2,8,9,8]
```

In [21]:
```
k_num = np.random.choice(3,10) #max 3
x1_x2_3['k_num'] = k_num
```

In [22]:
```
x1_x2_3
```

Out[22]:

	x1	x2	k_num
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 [29]:
```
def clust_fit(k, kdf, stop):
    k_fit = df.copy()
    for i in range(stop):
        c = {}
        for i in range(k):
            ctrd_1 = k_fit[k_fit['k_num'] == k]['x1'].mean()
            ctrd_2 = k_fit[k_fit['k_num'] == k]['x2'].mean()
```

```
            c[i] = (ctrd_1, ctrd_2)
#calculated the centroids

        labels_new = []
        for index, row in k_fit.iterrows():
            dist_min = 50
            for k, v in c.items():
                euc_d = math.sqrt((row['x1'] - v[0]) ** 2 + (row['x2']
- v[1]) ** 2)
                if euc_d < dist_min:
                    dist_min = euc_d
                    k_new = i
            labels_new.append(k_new)
        k_fit['k_num'] = labels_new

    return k_fit
```

In [30]:
```
clust_3 = clust_fit(3, x1_x2_3, 50)
clust_3
```

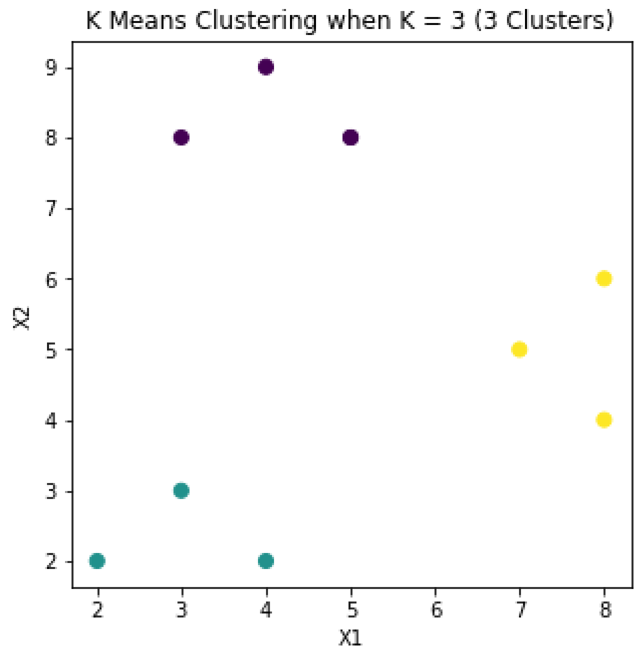
Out[30]:

	x1	x2	k_num
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 [ ]:
```
fig = plt.figure(figsize=(4, 4))
colors = list(clust_3['k_num'])
plt.scatter(clust_3['x1'], clust_3['x2'], c=colors, s=50)
```

In [18]:
```
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K Means Clustering when K = 3 (3 Clusters)')
plt.show()
```



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 [26]:
```
np.random.seed(666)
x1_x2_2 = pd.DataFrame()
```

```
x1_x2_2['x1'] = [5,8,7,8,3,4,2,3,4,5]
x1_x2_2['x2'] = [8,6,5,4,3,2,2,8,9,8]

k_num = np.random.choice(2,10) #max 3
x1_x2_2['k_num'] = k_num

x1_x2_2
```

Out[26]:

	x1	x2	k_num
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]:

```
clust_2 = clust_fit(2, x1_x2_2, 50)
clust_2
```

Out[31]:

	x1	x2	k_num
0	5	8	0
1	8	6	0
2	7	5	0
3	8	4	0
4	3	3	1

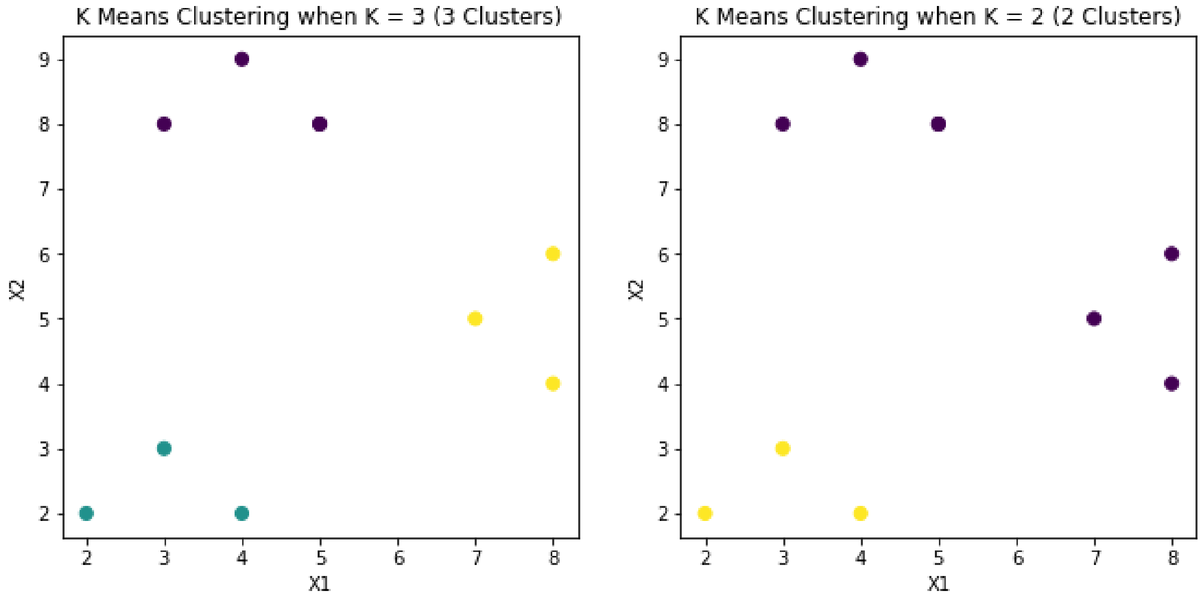
	x1	x2	k_num
5	4	2	1
6	2	2	1
7	3	8	0
8	4	9	0
9	5	8	0

In [ ]:

```
colors3, colors2 = list(clust_3['k_num']), list(clust_2['k_num'])
fig, axes = plt.subplots(1, 2, figsize=(10,5))
```

In [32]:

```
axes[0].scatter(clust_3['x1'], clust_3['x2'], c=colors3, s=50)
axes[0].set_xlabel('X1')
axes[0].set_ylabel('X2')
axes[0].set_title('K Means Clustering when K = 3 (3 Clusters)')
axes[1].scatter(clust_2['x1'], clust_2['x2'], c=colors2, s=50)
axes[1].set_xlabel('X1')
axes[1].set_ylabel('X2')
axes[1].set_title('K Means Clustering when K = 2 (2 Clusters)')
plt.show()
```



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?

The plots above indicate that 3 clusters do better than 2 clusters for this data. The distance between points within the two same-colored clusters in the k=2 plot is very close to the between-cluster distance, but the k=3 plot resolves this by simply considering the second purple cluster a third, distinct cluster.

In [ ]:

```

```

wiki.csv contains a data set of survey responses from university faculty members related to their perceptions and practices of using Wikipedia as a teaching resource. Documentation for this dataset can be found at the UCI machine learning repository. The dataset has been pre-processed for you as follows:

Include only employees of UOC and remove OTHER*, UNIVERSITY variables Impute missing values Convert domain and uoc_position to dummy variables

Dimension reduction

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 [34]:
```
wiki_df = pd.read_csv('wiki.csv')
```

In [35]:
```
wiki_df
```

Out[35]:

	age	gender	phd	yearsexp	userwiki	pu1	pu2	pu3	peu1	peu2	...	exp5	domain_Scie
0	40	0	1	14	0	4	4	3	5	5	...	2	
1	42	0	1	18	0	2	3	3	4	4	...	4	
2	37	0	1	13	0	2	2	2	4	4	...	3	
3	40	0	0	13	0	3	3	4	3	3	...	4	
4	51	0	0	8	1	4	3	5	5	4	...	4	
...	...	...	...	...	...	...	...	...	...	...	...	...	
795	62	1	0	15	0	2	5	4	5	5	...	2	
796	46	1	1	21	0	3	4	5	4	5	...	1	
797	49	1	1	23	0	3	3	3	5	5	...	2	
798	42	1	0	19	1	2	2	3	4	3	...	2	
799	45	1	1	20	0	2	2	2	4	2	...	1	

800 rows × 57 columns

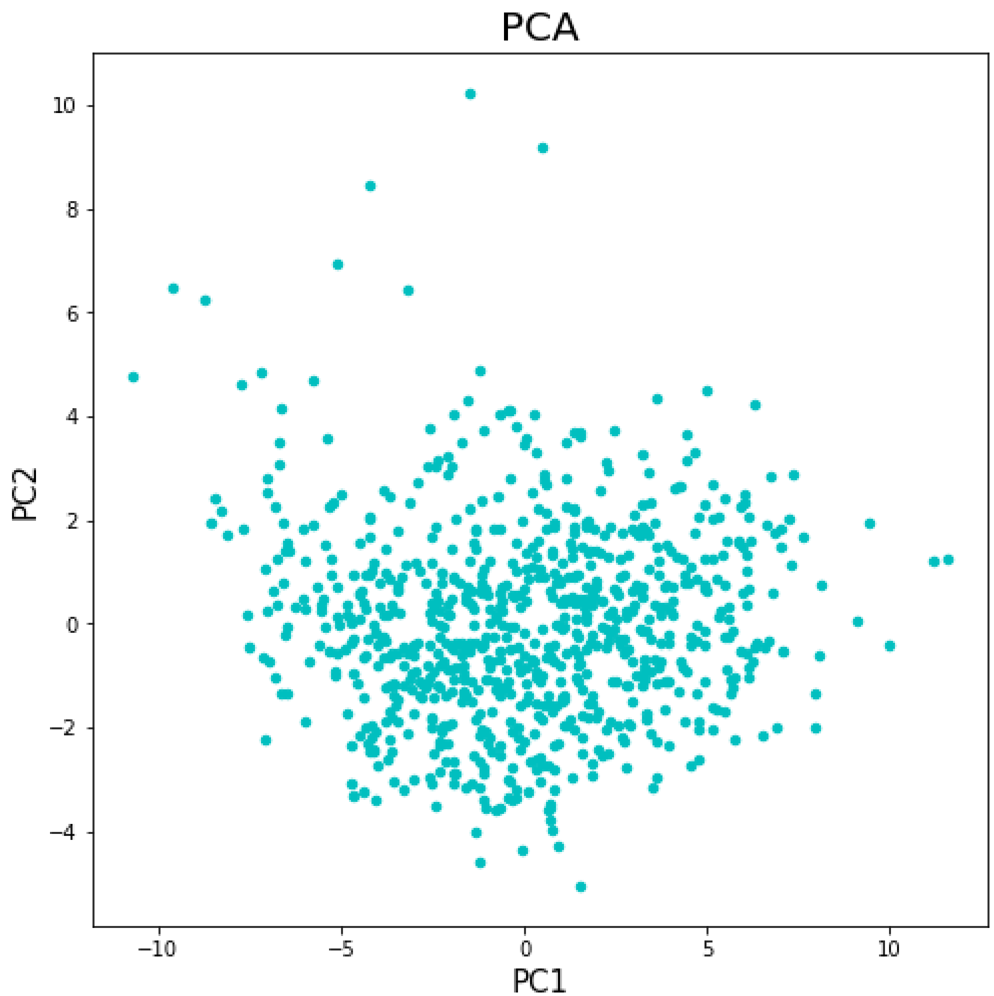
In [37]:
```
scale = StandardScaler().fit_transform(wiki_df)
```

In [ ]:
```
comp = PCA(random_state=np.random.seed(666))
```

In [ ]:
```
scale_PCA = pca.fit_transform(scale)
```

In [ ]:
```
plt.figure(figsize=(8,8))
plt.title('PCA', fontsize=20)
```

In [39]:
```
plt.scatter(scale_PCA[:,0], scale_PCA[:,1], s=20, c='c')
plt.xlabel('PC1', fontsize=15)
plt.ylabel('PC2', fontsize=15);
```




```
In [ ]: load = pd.DataFrame(comp.components_, columns=wiki_df.columns).T
```

```
In [40]: load[0].sort_values(ascending=False).head()
```

```
Out[40]: bi2     0.230924
         bi1     0.226193
         use3    0.218809
         use4    0.214558
```

```
pu3     0.210863
Name: 0, dtype: float64
```

Above are the loading vectors for all PCs, sorted (top listed first)

In [41]:
```
load[1].sort_values(ascending=False).head()
```

Out[41]:
```
exp4                              0.228494
use2                              0.218629
use1                              0.197827
vis3                              0.197635
domain_Engineering_Architecture   0.171484
Name: 1, dtype: float64
```

Above are the loading vectors for the second component (top listed first)

- What variables appear strongly correlated on the first principal component?

The output above indicates that bi2 is the most strongly correlated variable on the first principal component. According to the documentation we were referred to: bi2= "In the future I will use Wikipedia in my teaching activity", bi1= "In the future I will recommend the use of Wikipedia to my colleagues and students", use3 = "I recommend my students to use Wikipedia", use4 = "I recommend my colleagues to use Wikipedia", and pu3 = "Wikipedia is useful for teaching". Bi1 and bi2 are labelled "behavioral intention", while use3 and 4 are labelled "use behavior", and pu3 is labelled "perceived usefulness". Together, this suggests that this component primarily relates to wiki's perceived usefulness, especially as it relates to wanting to recommend wiki.

- What about the second principal component?

According to the output above, exp4 (experience of contributing to wikipedia) is the most strongly correlated variable on the second principal component. According to the documentation we were referred to: exp4 = "I contribute to Wikipedia (editions, revisions, articles improvement...)", use2 = "I use Wikipedia as a platform to develop educational activities with students", use1 = " I use Wikipedia to develop my teaching materials", and vis3 = "I cite Wikipedia in my academic papers". Exp4 is labelled "experience", use1 and 2 are labelled "use behavior", and vis3 is labelled "visibility". Together, this suggests that this component strongly relates to education, or wiki use related to education and academia.

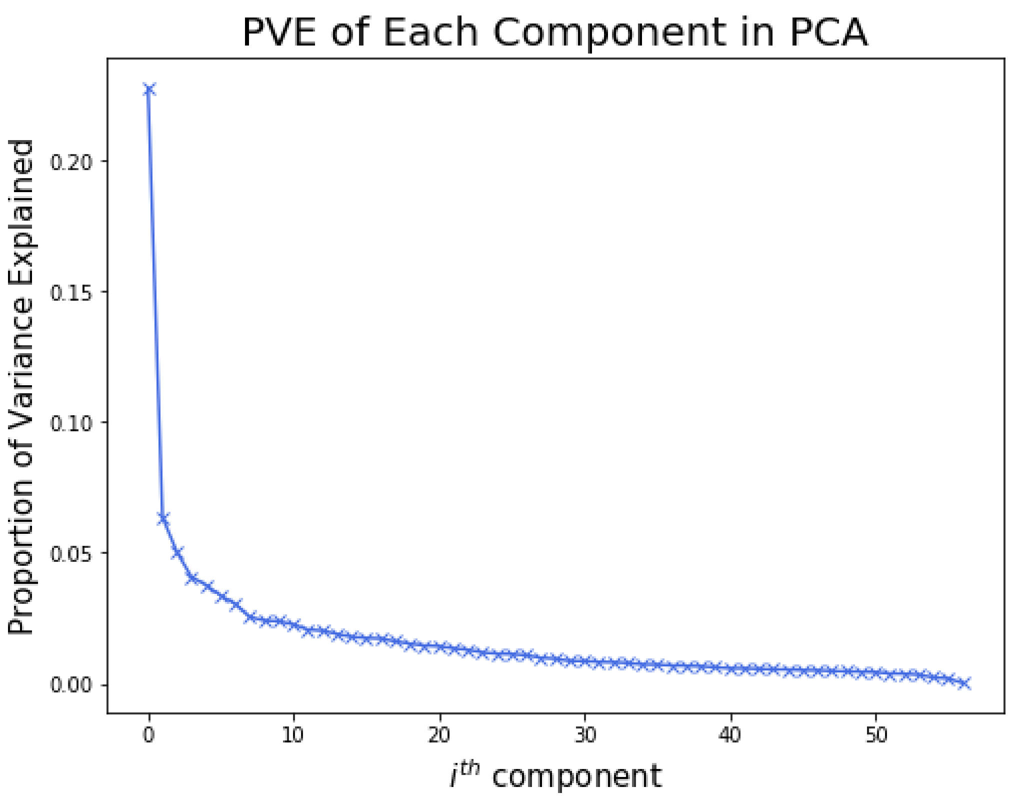
*in other iterations of this PCA, I found that peu1, inc1, sa3, sa1, and sa2 were the most strongly related to PCA2. These also relate to education/academia, which is in the same general topic as the ones I describe earlier, which makes sense.

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 [ ]:
```
plt.title('PVE of Each Component in PCA', fontsize=20)
plt.xlabel('$i^{th}$ component', fontsize=15)
plt.ylabel('Proportion of Variance Explained', fontsize=15);
```

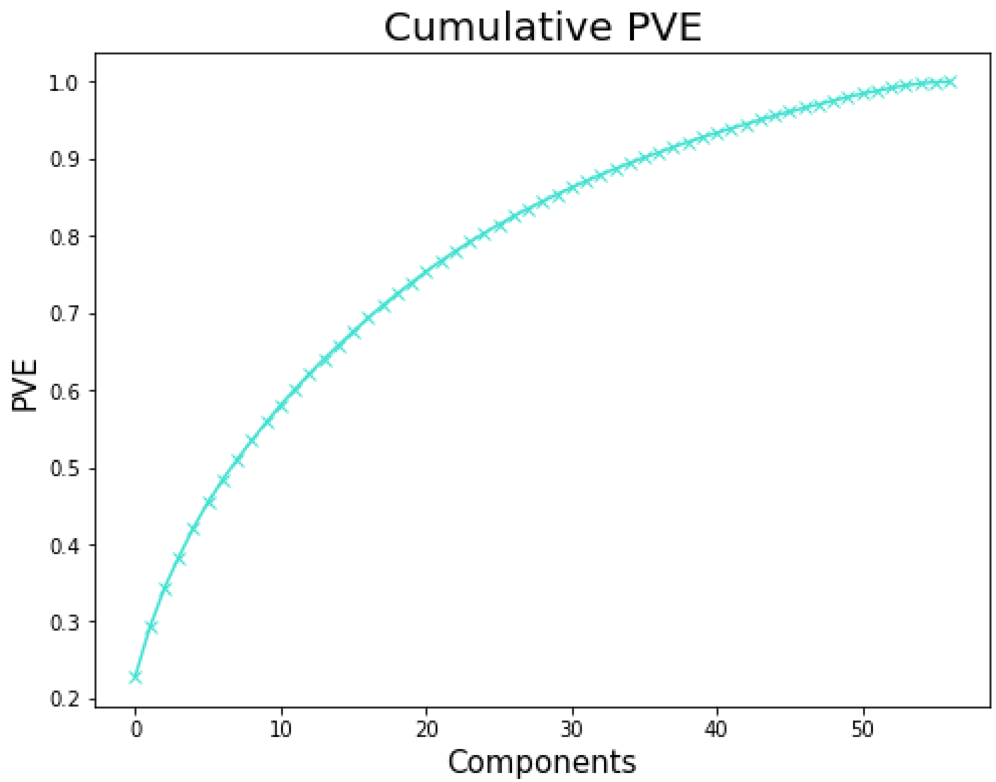
In [42]:
```
plt.figure(figsize=(8,6))
plt.plot(np.arange(57), comp.explained_variance_ratio_, marker='x', col
or='royalblue')
print(f'The first and second component explained about {round(comp.expl
ained_variance_ratio_[:2].sum(), 4)*100}% of variance')
```

```
The first and second component explained about 29.18% of variance
```




```
In [ ]: cve = np.cumsum(pca.explained_variance_ratio_)
        plt.figure(figsize=(9,6))
        plt.plot(np.arange(57), cve, marker='x', color='turquoise')
```

```
In [50]: plt.title('Cumulative PVE', fontsize=20)
         plt.xlabel('Components', fontsize=15)
         plt.ylabel('PVE', fontsize=15);
```



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

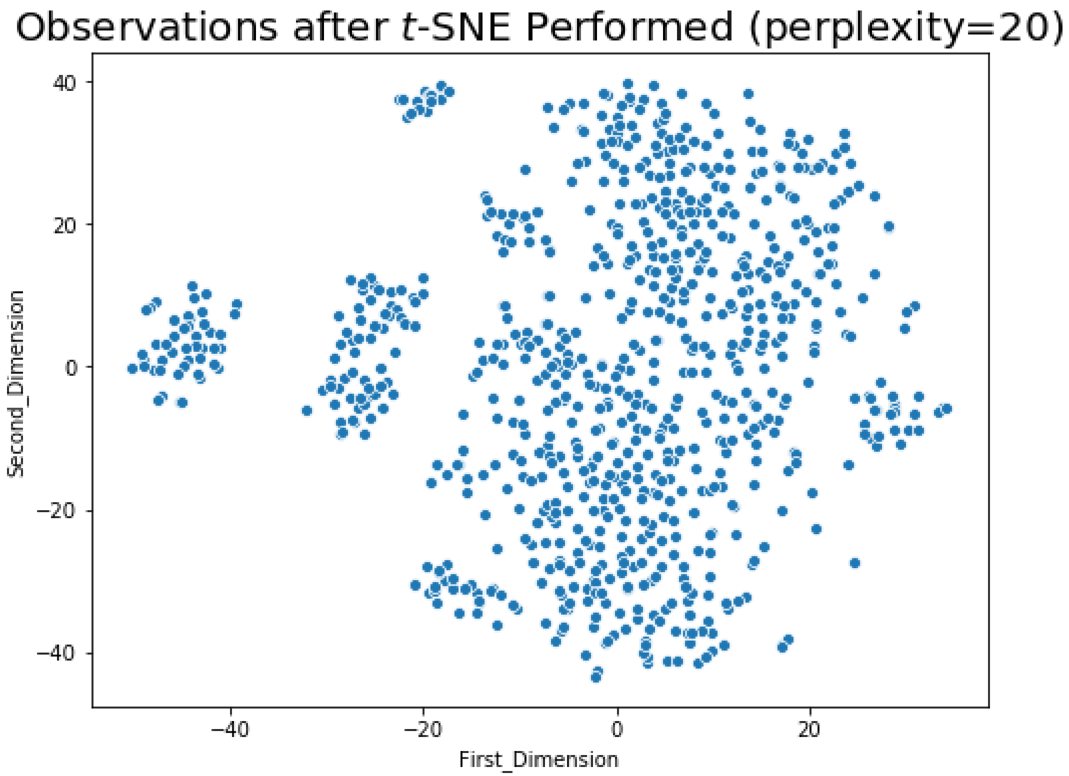
In [45]:
```
tsne = TSNE(n_components=2, perplexity=20, random_state=np.random.seed(
666))
X_e = tsne.fit_transform(scale)
```

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
tsnep = pd.DataFrame(X_e, columns=['First_Dimension', 'Second_Dimensio
n'])
plt.figure(figsize=(8,6))
```

```
plt.title('Observations after $t$-SNE Performed (perplexity=20)', fonts
ize=20)
```

In [48]:
```
sns.scatterplot(x='First_Dimension', y='Second_Dimension', data=tsnep);
```



The clusters in the plot above suggest that some observations in the data are more highly correlated than others. The chunks are quite distinct (we can make out at least four clearly). Based on the large chunk in the middle, many of the datapoints are actually quite similar to each other.

1. *Clustering*

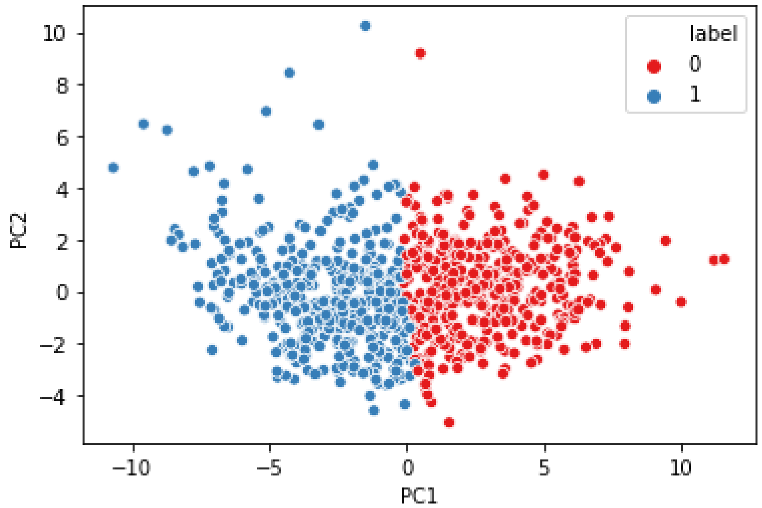
(15 points) Perform k-means clustering with $k = 2, 3, 4$. Be sure to scale each feature (i.e.,mean zero and standard deviation one). Plot the observations on the first and second principal components from PCA and color-code each observation based on their cluster membership. Discuss your results.

In [ ]:
```
#for k=2
k2 = KMeans(n_clusters=2, random_state=np.random.seed(666))
```

In [ ]:
```
k2plot = pd.DataFrame(scale_PCA[:,0:2], columns=['PC1', 'PC2'])
```

In [ ]:
```
k2plot['label'] = k2.fit_predict(scale)
```

In [53]:
```
sns.scatterplot(x='PC1', y='PC2', hue='label', data=k2_df, palette='Set
1');
```



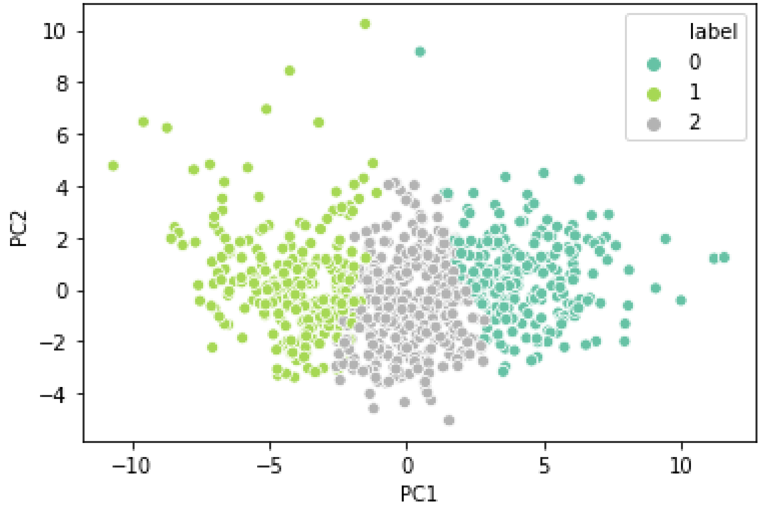
In [ ]:
```
#for k=3
```

In [ ]:
```
k3 = KMeans(n_clusters=3, random_state=np.random.seed(666))
```

In [ ]:
```
k3plot = pd.DataFrame(X_pc[:,0:2], columns=['PC1', 'PC2'])
```

In [ ]:
```
k3plot['label'] = k3.fit_predict(scale)
```

In [56]:
```
sns.scatterplot(x='PC1', y='PC2', hue='label', data=k3plot, palette='Se
t2');
```



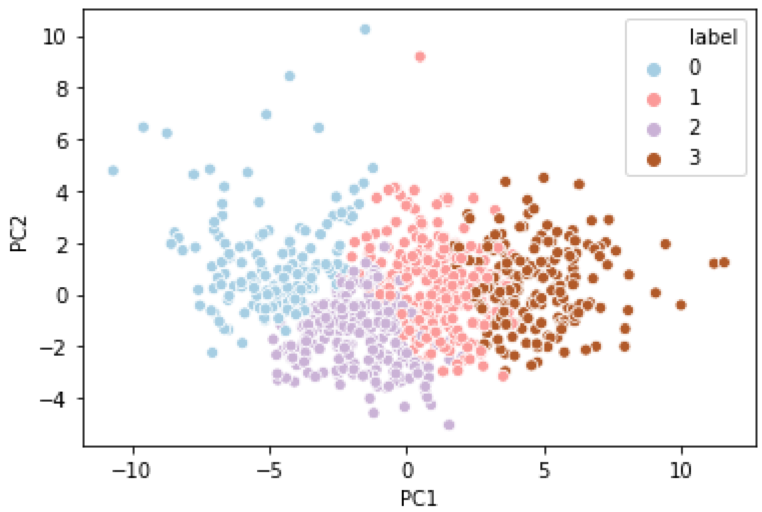
In [ ]:
```
# For k = 4
```

In [ ]:
```
k4 = KMeans(n_clusters=4, random_state=np.random.seed(666))
```

In [ ]:
```
k4plot = pd.DataFrame(scale_PCA[:,0:2], columns=['PC1', 'PC2'])
```

In [ ]:
```
k4plot['label'] = k4.fit_predict(scale)
```

In [60]:
```
sns.scatterplot(x='PC1', y='PC2', hue='label', data=k4plot, palette='Pa
ired');
```



Based on the plots above, the first two principal components do a pretty good job of explaining the variance in the data. We start to eee some overlap with k=3 and k=4 (the boundaries bleeding into each other in a sense), while the divide in clusters for k=2 is much cleaner.

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.

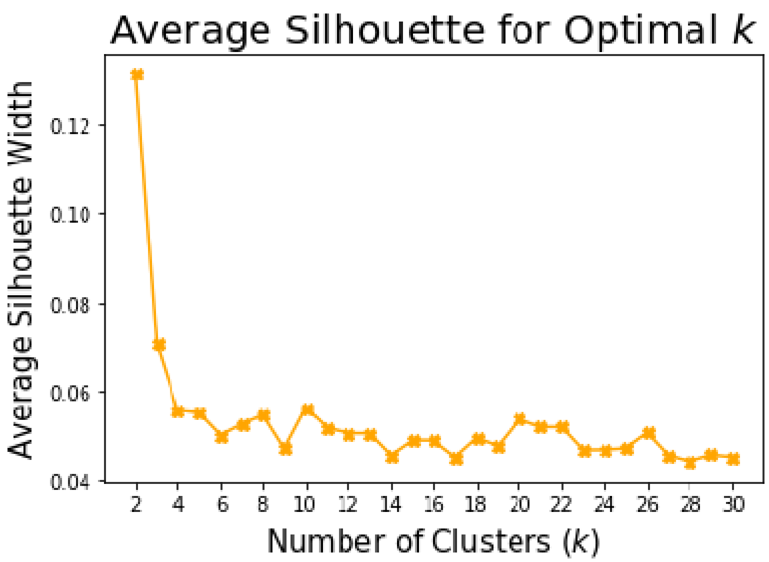
In [62]:
```
inert_s = []
sil_avg =[]
set_span = range(2,31)
```

In [ ]:
```
for i in set_span:
    model = KMeans(n_clusters=i, random_state=np.random.seed(666)).fit(
scale)
    inert_s.append(model.inertia_)
    sil.avg.append(silhouette_score(scale, model.labels_))
```

In [ ]:
```
plt.plot(test_range, avg_sil_score, marker='X', color='orange')
```

```
plt.title('Average Silhouette for Optimal $k$', fontsize=20)
```

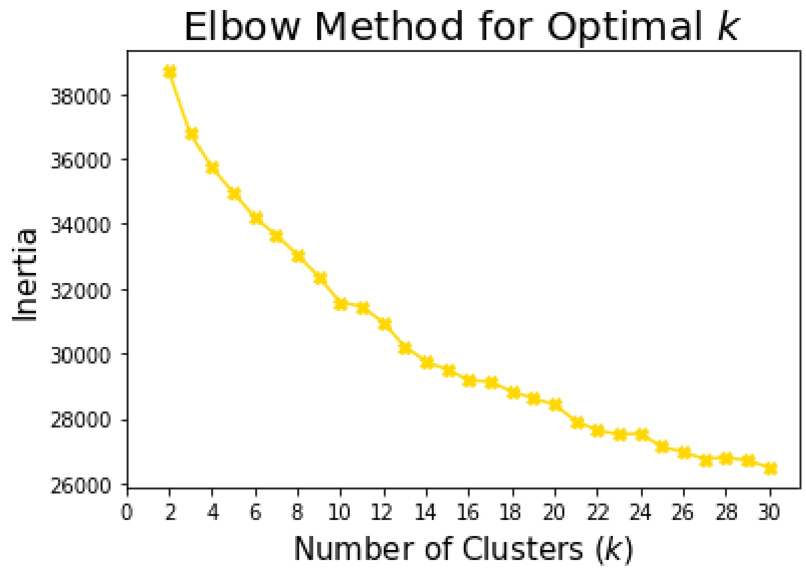
In [ ]:
```
plt.xlabel('Number of Clusters ($k$)', fontsize=15)
plt.xticks(np.arange(2,max(test_range)+2,2))
plt.ylabel('Average Silhouette Width', fontsize=15);
```



The highest average silhouette width corresponds to k=2.

In [ ]:
```
plt.plot(test_range, inert_s, marker='X', color='gold')
plt.title('Elbow Method for Optimal $k$', fontsize=20)
plt.xlabel('Number of Clusters ($k$)', fontsize=15)
```

In [70]:
```
plt.xticks(np.arange(0,max(test_range)+2,2))
plt.ylabel('Inertia', fontsize=15);
```

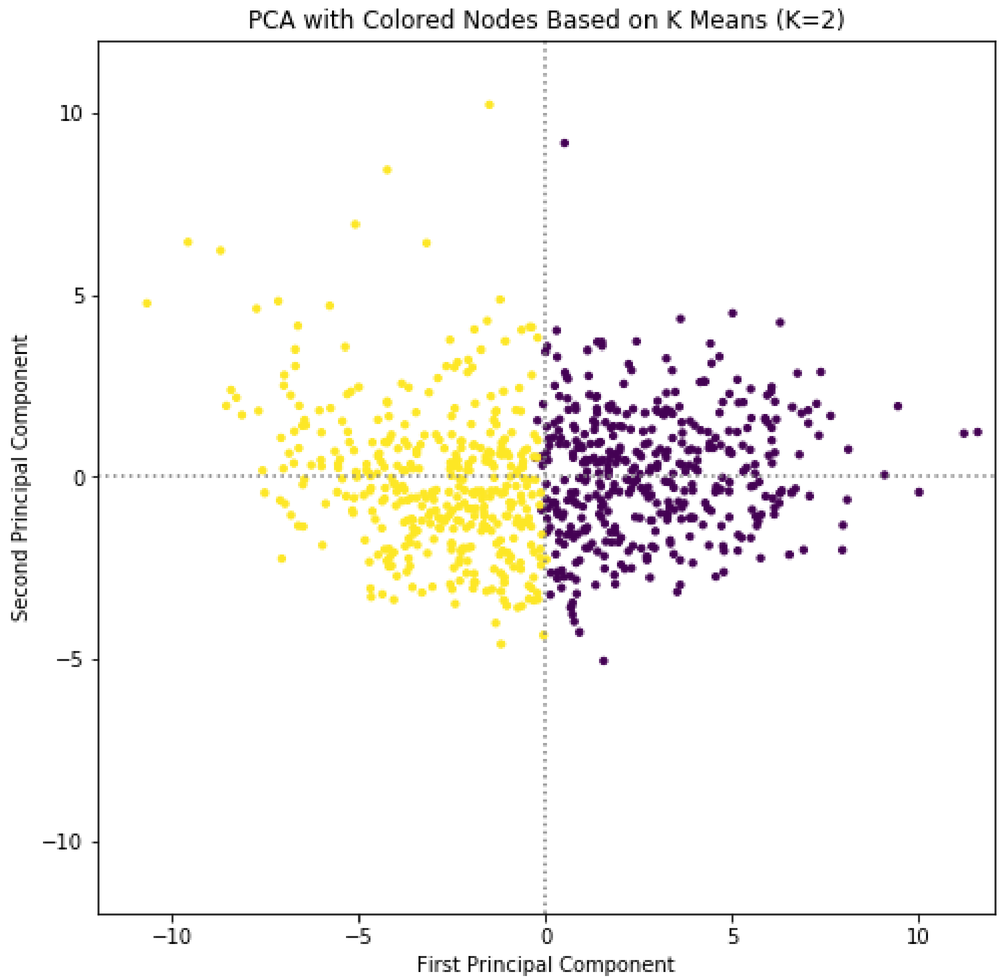


There is no clear elbow in the plot above. There is a slight bend at k=11, but beyond that, other statistical tools may need to be employed to determine the optimal 'k' (ie, average silhouette above).

1. (15 points) 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?

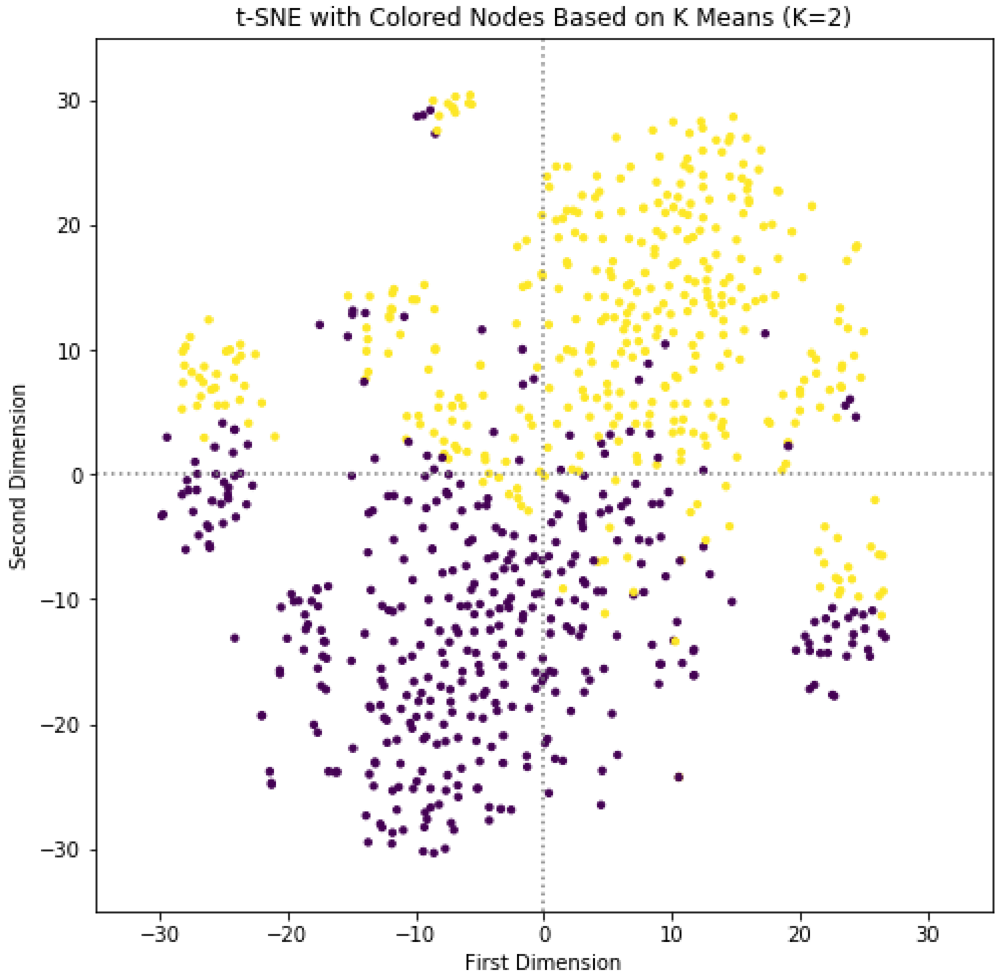
In [ ]:
```
# Fit k means with k=2
SEED = np.random.seed(666)
kmeans2 = KMeans(n_clusters=2, random_state=SEED).fit(scale)
# Plot obs on PC1 and PC2
fig = plt.figure(figsize=(8, 8))
plt.xlim(-12,12)
plt.ylim(-12,12)
```

In [ ]:
```
plt.scatter(load[0], load[1], s=11, c=kmeans2.labels_)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA with Colored Nodes Based on K Means (K=2)')
plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
plt.vlines(0,-12,12, linestyles='dotted', colors='grey')
plt.show()
```



In [ ]:
```
fig = plt.figure(figsize=(8, 8))
plt.xlim(-35,35)
plt.ylim(-35,35)
plt.hlines(0,-35,35, linestyles='dotted', colors='grey')
plt.vlines(0,-35,35, linestyles='dotted', colors='grey')
```

In [ ]:
```
plt.scatter(tsnep['First_Dimension'], tsnep['Second_Dimension'], s=11,
c=kmeans2.labels_)
plt.xlabel('First Dimension')
plt.ylabel('Second Dimension')
plt.title('t-SNE with Colored Nodes Based on K Means (K=2)')
plt.show()
```



The main difference in the output is that, with PCA, it's largely the first principal component that is projecting an impact on the data (the vertical boundary), in separating the clusters. However, in the tSNE plots, both dimensions are having an impact in differentiating the clusters. Additionally, in PCA, the distinction is much more clear (between the clusters), and less so in tSNE. A linear pattern in the data might explain this result.

```
In [ ]:
```