In [128]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

from scipy.cluster import hierarchy
from sklearn import manifold
```

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.

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

In [63]:
```
f1 = [5,8,7,8,3,4,2,3,4,5]
f2 = [8,6,5,4,3,2,2,8,9,8]

data = pd.DataFrame(list(zip(f1, f2)), columns=['f1', 'f2'])
```

In [64]:
```
import random

def ini_clusters(k, df):
    index = [i for i in range(0,len(df))]
    df['cluster'] = '0'
    clusters = [[] for i in range(k)]
    random.seed(4)
    random.shuffle(index)
    for i in range(len(index)):
        df['cluster'].iloc[index[i]] = i % k
```

In [65]:
```
ini_clusters(3, data)
data
```

Out[65]:

	f1	f2	cluster
0	5	8	0
1	8	6	2
2	7	5	2
3	8	4	1
4	3	3	0
5	4	2	0
6	2	2	1
7	3	8	0
8	4	9	2
9	5	8	1

2.

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

In [68]:
```
from scipy.spatial import distance
def reassign(k, df):
    centroid = df.groupby(['cluster']).mean()
    change = False
    for index, row in df.iterrows():
        d = 999
        for i in range(k):
            temp = distance.euclidean(centroid.iloc[i],(row[0], row[1]))
            if temp < d:
                d = temp
                group = i
        if df['cluster'].iloc[index] != group:
            change = True
            df['cluster'].iloc[index] = group
    return change
```

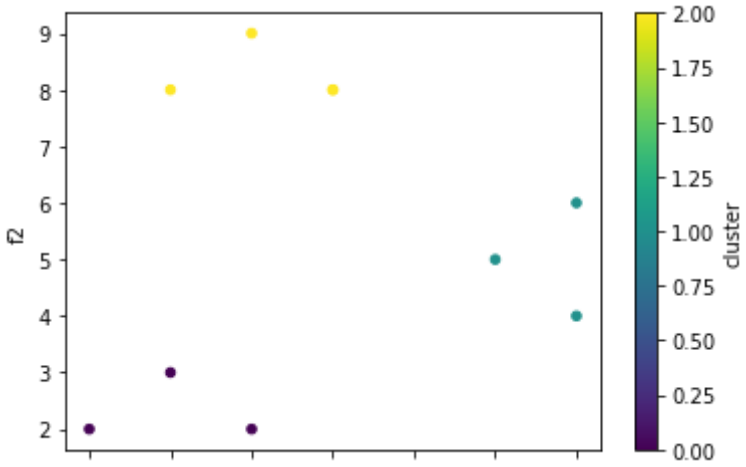
In [69]:
```
def kmc(k, df):
    while 1:
        state = reassign(k, df)
        if not state:
            break
    return df
```

3.

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

In [72]:
```
data = kmc(3, data)
data.plot.scatter(x='f1', y='f2', c='cluster',colormap='viridis')
```

Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20660050>

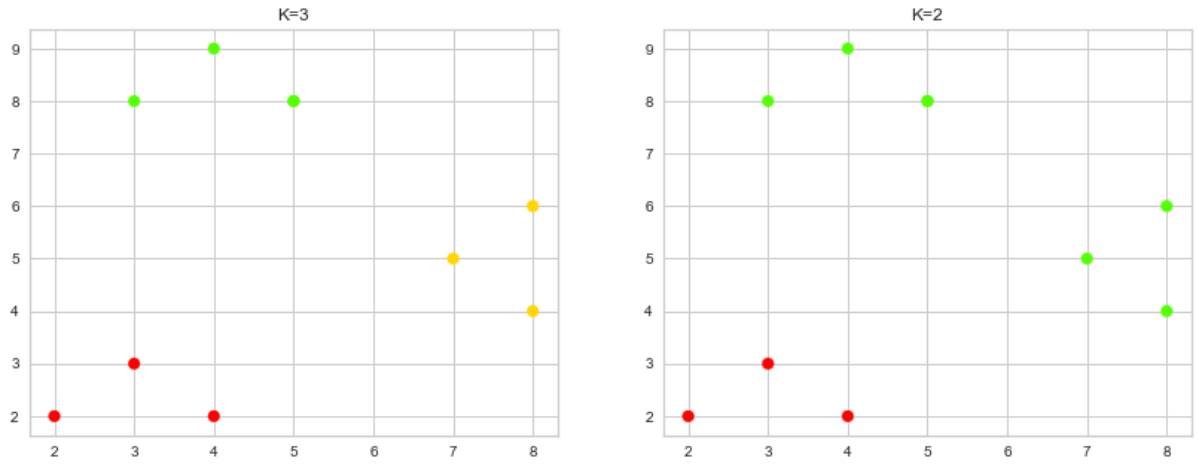


4.

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.

In [75]:
```
data2 = pd.DataFrame(list(zip(f1, f2)), columns=['f1', 'f2'])
ini_clusters(2, data2)
data2 = kmc(2, data2)
```

In [229]:
```
# ax1 = data.plot.scatter(x='f1', y='f2', c='cluster',colormap='viridis')
# ax2 = data2.plot.scatter(x='f1', y='f2', c='cluster',colormap='viridis')
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(14,5))
ax1.scatter('f1', 'f2', c='cluster', s=60, data=data, cmap=plt.cm.prism)
ax1.set_title('K=3')
ax2.scatter('f1', 'f2', c='cluster', s=60, data=data2, cmap=plt.cm.prism)
ax2.set_title('K=2');
```



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?

From the scatter plot, it is clear that the dataset are originally separated into 3 clusters with clear boundaries and far enough pairwise distance. Hence the initial hunch which catch the original configuration of the dataset pans out.

Application

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

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 compone
nt?
    What about the second principal component?
```

In [100]:
```
wiki = pd.read_csv('wiki.csv')
wiki.shape
```

Out[100]:
```
(800, 57)
```

In [86]:
```
wiki.describe()
```

Out[86]:

	age	gender	phd	yearsexp	userwiki	pu1	pu2	
count	800.00000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.0
mean	42.16625	0.427500	0.433750	10.408750	0.136250	3.125000	3.136250	3.4
std	7.54767	0.495025	0.495902	6.756755	0.343268	1.004059	0.980504	1.0
min	23.00000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.0
25%	36.00000	0.000000	0.000000	5.000000	0.000000	2.000000	2.000000	3.0
50%	42.00000	0.000000	0.000000	10.000000	0.000000	3.000000	3.000000	3.0
75%	47.00000	1.000000	1.000000	15.000000	0.000000	4.000000	4.000000	4.0
max	69.00000	1.000000	1.000000	36.000000	1.000000	5.000000	5.000000	5.0

8 rows × 57 columns

In [101]:
```
X = pd.DataFrame(scale(wiki), index=wiki.index, columns=wiki.columns)
```

In [120]:
```
# The loading vectors
pca_loadings = pd.DataFrame(PCA(n_components=2).fit(X).components_.T,
                            index=wiki.columns, columns=['PC1', 'PC2'])
pca_loadings.sort_values(by=['PC1', 'PC2'], ascending=False)
```

Out[120]:

	PC1	PC2
bi2	0.230924	0.083428
bi1	0.226193	0.056369
use3	0.218809	0.155153
use4	0.214558	0.160866
pu3	0.210863	0.028775
exp1	0.208592	0.070548
use5	0.206539	0.029817
exp2	0.195043	-0.029563
pu1	0.192827	0.008266
pu2	0.190588	0.017663
qu5	0.183365	0.010918
use1	0.181477	0.197830
qu1	0.178057	-0.038114
vis3	0.175351	0.197643
vis1	0.171153	-0.025212
qu2	0.163778	-0.066405
im3	0.160803	0.043989
im1	0.160432	0.111096
qu3	0.157956	-0.033456
use2	0.147852	0.218628
enj1	0.145666	-0.151015
exp3	0.144023	-0.126432
enj2	0.131110	-0.227608
sa1	0.121658	-0.229926
sa3	0.120376	-0.242320
sa2	0.117590	-0.226756
vis2	0.114559	-0.056218
peu2	0.113719	-0.222369
exp5	0.110628	0.076108
pf3	0.109632	0.094177
inc1	0.104667	-0.245436
pf2	0.103448	0.018601
pf1	0.102338	0.114385
peu3	0.100219	-0.068452
exp4	0.099873	0.228483
inc2	0.095802	-0.202022

	PC1	PC2
inc4	0.089707	-0.202022
inc3	0.081402	-0.220986
userwiki	0.081363	0.134367
jr1	0.080867	-0.136968
im2	0.077810	-0.059790
jr2	0.062216	-0.106292
peu1	0.061228	-0.271746
domain_Engineering_Architecture	0.051309	0.171478
domain_Sciences	0.021982	-0.014532
uoc_position_Associate	0.010922	0.013101
uoc_position_Assistant	0.007123	0.002344
uoc_position_Instructor	0.004251	-0.003724
uoc_position_Adjunct	-0.007849	-0.005298
domain_Health.Sciences	-0.017158	-0.015489
uoc_position_Lecturer	-0.018041	-0.023610
age	-0.021805	0.088393
phd	-0.030501	0.030431
yearsexp	-0.034190	0.062368
gender	-0.035086	-0.149461
qu4	-0.060797	-0.103481
domain_Law_Politics	-0.094775	-0.014890

In [152]:
```
pca = PCA(n_components=2)
df_plot = pd.DataFrame(pca.fit_transform(X),
                       columns=['PC1', 'PC2'], index=X.index)
df_plot
```

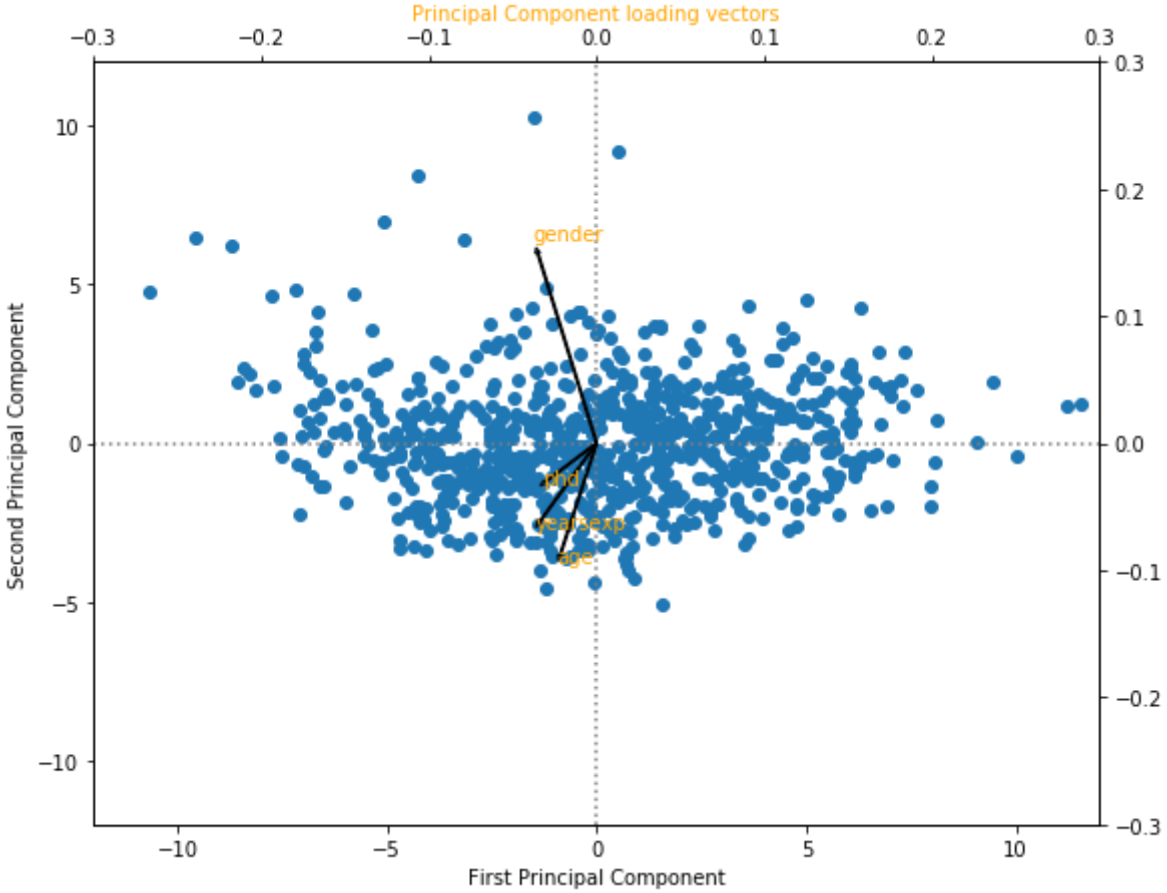
Out[152]:

	PC1	PC2
0	-0.150216	-1.981573
1	-3.314020	-0.791500
2	-4.682484	-0.311896
3	1.774200	1.985508
4	7.254695	2.012686
...	...	...
795	0.227143	1.473676
796	4.434784	-0.932148
797	1.449455	-0.170654
798	-2.888282	2.720822
799	-7.000656	2.805688

800 rows × 2 columns

In [122]:

```
fig , ax1 = plt.subplots(figsize=(9,7))

ax1.set_xlim(-12,12)
ax1.set_ylim(-12,12)

# Plot Principal Components 1 and 2
ax1.scatter(df_plot['PC1'], df_plot['PC2'])


# Plot reference lines
ax1.hlines(0,-12,12, linestyles='dotted', colors='grey')
ax1.vlines(0,-12,12, linestyles='dotted', colors='grey')

ax1.set_xlabel('First Principal Component')
ax1.set_ylabel('Second Principal Component')

# Plot Principal Component loading vectors, using a second y-axis.
ax2 = ax1.twinx().twiny()

ax2.set_ylim(-0.3,0.3)
ax2.set_xlim(-0.3,0.3)
ax2.tick_params(axis='y', colors='orange')
ax2.set_xlabel('Principal Component loading vectors', color='orange')

# Plot labels for vectors. Variable 'a' is a small offset parameter to s
eparate arrow tip and text.
a = 1.07
for i in range(4):
    ax2.annotate(pca_loadings[['PC1', 'PC2']].index[i],
                 (pca_loadings.PC1.iloc[i]*a,
                  -pca_loadings.PC2.iloc[i]*a),
                 color='orange')

# Plot vectors
ax2.arrow(0,0,pca_loadings.PC1[0], -pca_loadings.PC2[0])
ax2.arrow(0,0,pca_loadings.PC1[1], -pca_loadings.PC2[1])
ax2.arrow(0,0,pca_loadings.PC1[2], -pca_loadings.PC2[2])
ax2.arrow(0,0,pca_loadings.PC1[3], -pca_loadings.PC2[3]);
```


Principal Component loading vectors
gender
phd
yearsexp
age
Second Principal Component
First Principal Component

In [123]:
```
pca_loadings.sort_values(by='PC1', ascending=False)
```

Out[123]:

	PC1	PC2
bi2	0.230924	0.083428
bi1	0.226193	0.056369
use3	0.218809	0.155153
use4	0.214558	0.160866
pu3	0.210863	0.028775
exp1	0.208592	0.070548
use5	0.206539	0.029817
exp2	0.195043	-0.029563
pu1	0.192827	0.008266
pu2	0.190588	0.017663
qu5	0.183365	0.010918
use1	0.181477	0.197830
qu1	0.178057	-0.038114
vis3	0.175351	0.197643
vis1	0.171153	-0.025212
qu2	0.163778	-0.066405
im3	0.160803	0.043989
im1	0.160432	0.111096
qu3	0.157956	-0.033456
use2	0.147852	0.218628
enj1	0.145666	-0.151015
exp3	0.144023	-0.126432
enj2	0.131110	-0.227608
sa1	0.121658	-0.229926
sa3	0.120376	-0.242320
sa2	0.117590	-0.226756
vis2	0.114559	-0.056218
peu2	0.113719	-0.222369
exp5	0.110628	0.076108
pf3	0.109632	0.094177
inc1	0.104667	-0.245436
pf2	0.103448	0.018601
pf1	0.102338	0.114385
peu3	0.100219	-0.068452
exp4	0.099873	0.228483
inc2	0.095802	-0.202022

	PC1	PC2
inc4	0.089707	-0.202022
inc3	0.081402	-0.220986
userwiki	0.081363	0.134367
jr1	0.080867	-0.136968
im2	0.077810	-0.059790
jr2	0.062216	-0.106292
peu1	0.061228	-0.271746
domain_Engineering_Architecture	0.051309	0.171478
domain_Sciences	0.021982	-0.014532
uoc_position_Associate	0.010922	0.013101
uoc_position_Assistant	0.007123	0.002344
uoc_position_Instructor	0.004251	-0.003724
uoc_position_Adjunct	-0.007849	-0.005298
domain_Health.Sciences	-0.017158	-0.015489
uoc_position_Lecturer	-0.018041	-0.023610
age	-0.021805	0.088393
phd	-0.030501	0.030431
yearsexp	-0.034190	0.062368
gender	-0.035086	-0.149461
qu4	-0.060797	-0.103481
domain_Law_Politics	-0.094775	-0.014890

In [124]:
```
pca_loadings.sort_values(by='PC2', ascending=False)
```

Out[124]:

	PC1	PC2
exp4	0.099873	0.228483
use2	0.147852	0.218628
use1	0.181477	0.197830
vis3	0.175351	0.197643
domain_Engineering_Architecture	0.051309	0.171478
use4	0.214558	0.160866
use3	0.218809	0.155153
userwiki	0.081363	0.134367
pf1	0.102338	0.114385
im1	0.160432	0.111096
pf3	0.109632	0.094177
age	-0.021805	0.088393
bi2	0.230924	0.083428
exp5	0.110628	0.076108
exp1	0.208592	0.070548
yearsexp	-0.034190	0.062368
bi1	0.226193	0.056369
im3	0.160803	0.043989
phd	-0.030501	0.030431
use5	0.206539	0.029817
pu3	0.210863	0.028775
pf2	0.103448	0.018601
pu2	0.190588	0.017663
uoc_position_Associate	0.010922	0.013101
qu5	0.183365	0.010918
pu1	0.192827	0.008266
uoc_position_Assistant	0.007123	0.002344
uoc_position_Instructor	0.004251	-0.003724
uoc_position_Adjunct	-0.007849	-0.005298
domain_Sciences	0.021982	-0.014532
domain_Law_Politics	-0.094775	-0.014890
domain_Health.Sciences	-0.017158	-0.015489
uoc_position_Lecturer	-0.018041	-0.023610
vis1	0.171153	-0.025212
exp2	0.195043	-0.029563
qu3	0.157956	-0.033456

	PC1	PC2
qu1	0.178057	-0.038114
vis2	0.114559	-0.056218
im2	0.077810	-0.059790
qu2	0.163778	-0.066405
peu3	0.100219	-0.068452
qu4	-0.060797	-0.103481
jr2	0.062216	-0.106292
exp3	0.144023	-0.126432
jr1	0.080867	-0.136968
gender	-0.035086	-0.149461
enj1	0.145666	-0.151015
inc4	0.089707	-0.202022
inc2	0.095802	-0.202022
inc3	0.081402	-0.220986
peu2	0.113719	-0.222369
sa2	0.117590	-0.226756
enj2	0.131110	-0.227608
sa1	0.121658	-0.229926
sa3	0.120376	-0.242320
inc1	0.104667	-0.245436
peu1	0.061228	-0.271746

The result is plotted on the first and second principal components. And the loading of each variable are listed and sorted. `bi2` , `bi1` , `use3` , `use4` and `pu3` appear to be most strongly correlated on the first principal component (with loading > 0.20), while `exp4` and `use2` have the highest loading of principal component 2.

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?

```
    Approximately 29.18% of the variance is explained by the first two princ
ipal components, 22.81% by the first one and 6.37% by the second one
```

In [126]:
```
pca.explained_variance_ratio_
```

Out[126]:
```
array([0.22810628, 0.06372475])
```

In [127]:
```
sum(pca.explained_variance_ratio_)
```

Out[127]:
```
0.29183102291351565
```

8.

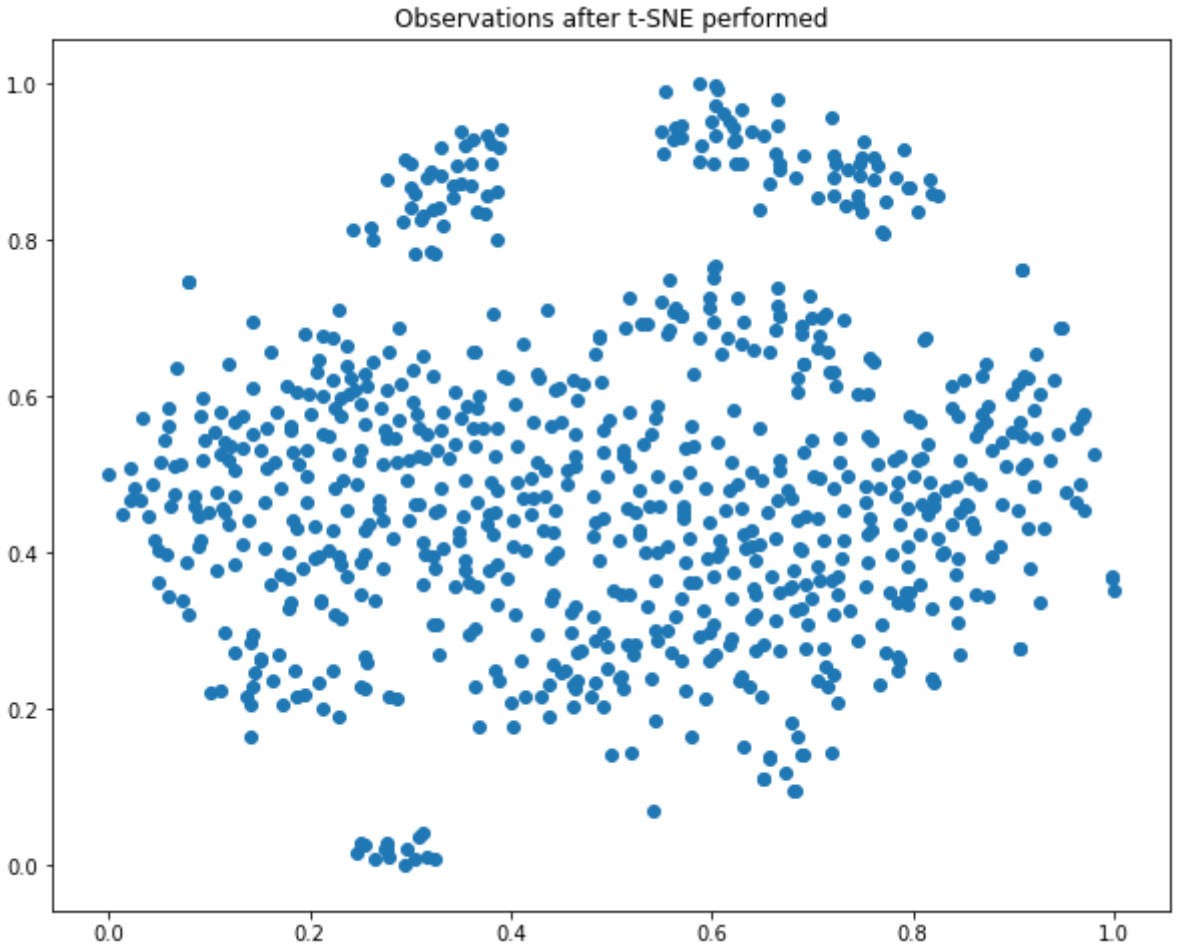
Perform t-SNE on the dataset and plot the observations on the first and second dimensions. Describe your results.

In [140]:
```
X = pd.DataFrame(scale(wiki), index=wiki.index, columns=wiki.columns)
```

In [141]:
```
tsne = manifold.TSNE(n_components=2, perplexity=5, init='pca', random_st
ate=501)
tsne.fit_transform(X)

tsne.embedding_
```

Out[141]:
```
array([[ 58.177002  ,  46.034405  ],
       [ 41.39709   ,  53.607243  ],
       [-42.78455   , -37.570396  ],
       ...,
       [ 14.319841  , -32.479748  ],
       [ -5.9014893 ,  49.498215  ],
       [-64.375084  ,  -0.76966196]], dtype=float32)
```

In [151]:
```
x_min, x_max = X_tsne.min(0), X_tsne.max(0)
X_norm = (X_tsne - x_min) / (x_max - x_min)  # 归一化
plt.figure(figsize=(10, 8))

plt.scatter(X_norm[:, 0], X_norm[:, 1])
plt.title('Observations after t-SNE performed')

plt.show()
```



The observations are plotted above after t-SNE (perplexity=5) performed. When compared with PCA, t-SNE has better performance on making clearly clustered visualization of data without any crowding problem that may exist using SNE. Each subgroup has clear margin.

Clustering

9.

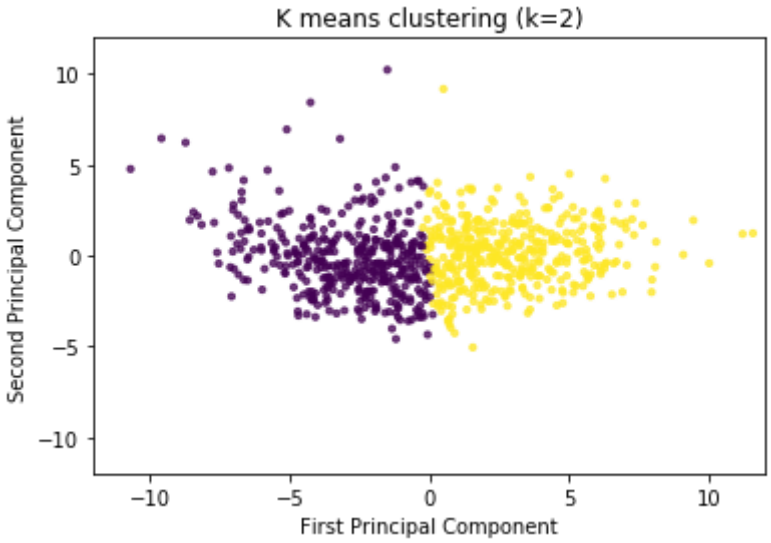
Perform k-means clustering with k = 2, 3, 4. Be sure to scale each feature (i.e.,mean zero and standard deviation one). Plot the observations on the first and second principal components from PCA and color-code each observation based on their cluster membership. Discuss your results.

In [210]:
```
# scaling
X = pd.DataFrame(scale(wiki), index=wiki.index, columns=wiki.columns)
```

In [216]:
```
def perform_kmc(k):
    kmc = KMeans(n_clusters=k, n_init=20).fit(X)
    label = kmc.labels_

    print('Sum of squared distances of samples to their closest cluster
 center is:' + str(kmc.inertia_))

    plt.xlim(-12,12)
    plt.ylim(-12,12)

    # Plot Principal Components 1 and 2
    plt.scatter(df_plot['PC1'], df_plot['PC2'], c=label,
                s=10, alpha=0.75)

    plt.xlabel('First Principal Component')
    plt.ylabel('Second Principal Component')
    plt.title(f'K means clustering (k={k})')
    plt.show()
```

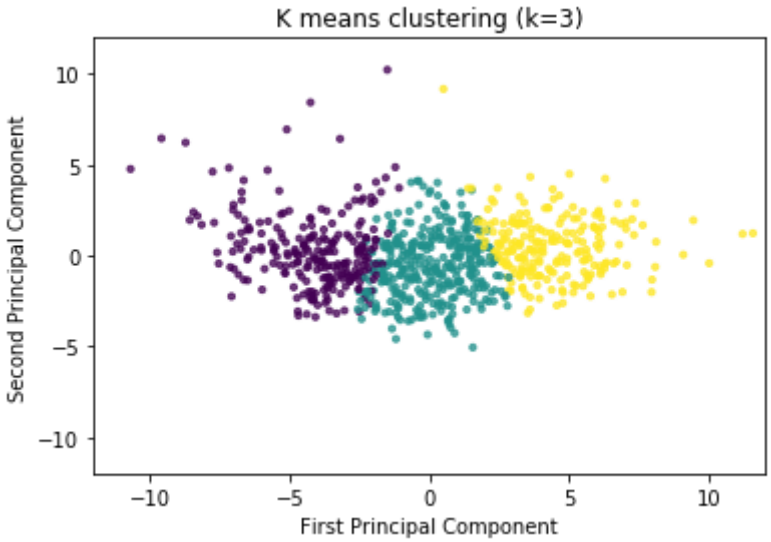
In [183]:
```
perform_kmc(2)
```

```
Sum of squared distances of samples to their closest cluster center is:
38704.38907656803
```



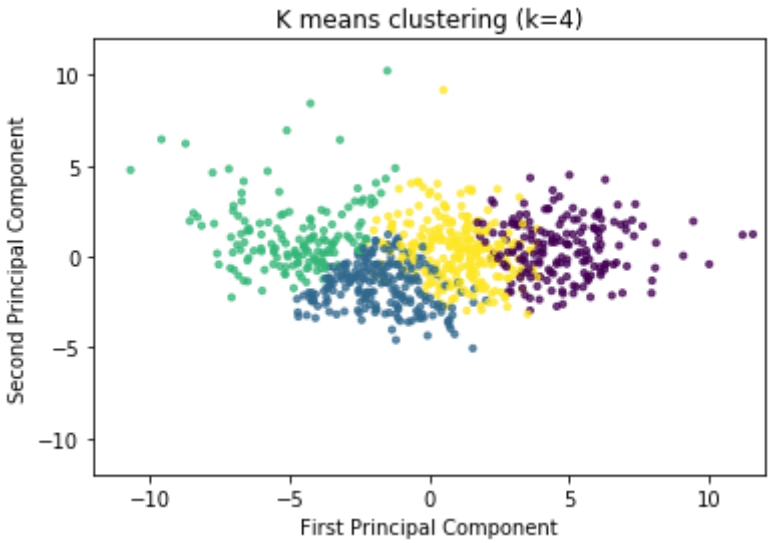
In [184]:
```
perform_kmc(3)
```

```
Sum of squared distances of samples to their closest cluster center is:
36791.631102728315
```



In [185]:
```
perform_kmc(4)
```

```
Sum of squared distances of samples to their closest cluster center is:
35725.80531433465
```



The results of performing k-means clustering with k=2, 3, 4 are depicted above. Compared with tSNE, PCA encounters crowding problem with its data visualiztaion. It is clear that, when k=2, the clusters are separated clearly with a vertical margin, suggesting that the clustering is highly dependent on the first principal component. As k increases, such vertical margin no longer exists, and the clustering become perplex. I also calculated the inertia (sum of squared distances of samples to their closest cluster center) which indicates that when k=4, the model yields the best performance.

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.

Reference

https://medium.com/@masarudheena/4-best-ways-to-find-optimal-number-of-clusters-for-clustering-with-python-code-706199fa957c (https://medium.com/@masarudheena/4-best-ways-to-find-optimal-number-of-clusters-for-clustering-with-python-code-706199fa957c)

for elbow

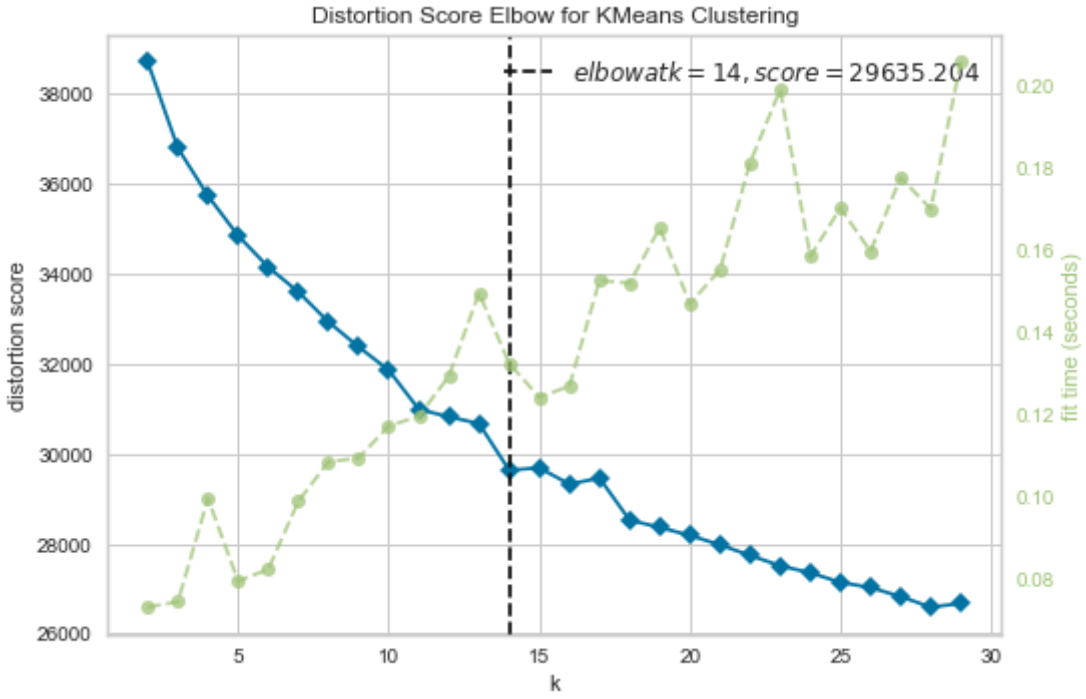
https://www.scikit-yb.org/en/latest/api/cluster/elbow.html (https://www.scikit-yb.org/en/latest/api/cluster/elbow.html)

for gap statistic

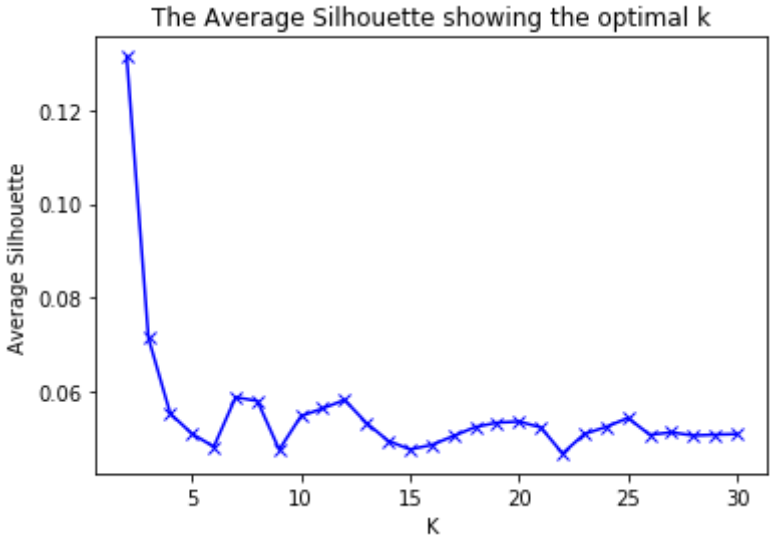
https://anaconda.org/milesgranger/gap-statistic/notebook (https://anaconda.org/milesgranger/gap-statistic/notebook)

In [190]:
```
K = range(2, 31)
```

In [207]:
```
# elbow method
from yellowbrick.cluster import KElbowVisualizer

# Instantiate the clustering model and visualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2,30))

visualizer.fit(X)        # Fit the data to the visualizer
visualizer.show();        # Finalize and render the figure
```



In [193]:
```
# average silhouette
from sklearn.metrics import silhouette_score
sa = []
for k in K:
    km = KMeans(n_clusters=k, random_state=10)
    km_labels = km.fit_predict(X)
    silhouette_avg = silhouette_score(X, km_labels)
    sa.append(silhouette_avg)

plt.plot(K, sa, 'bx-')
plt.xlabel('K')
plt.ylabel('Average Silhouette')
plt.title('The Average Silhouette showing the optimal k')
plt.show()
```

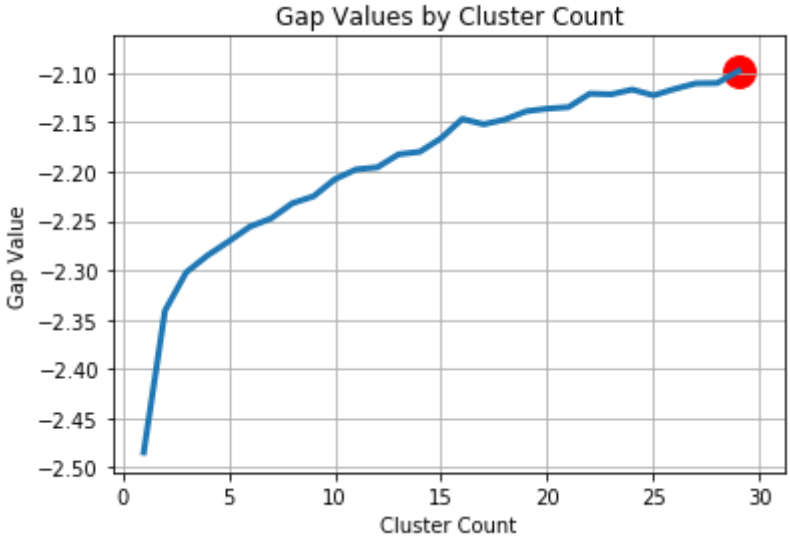


In [195]:
```
# gap statistics
def optimalK(data, nrefs=3, maxClusters=15):
    """
    Calculates KMeans optimal K using Gap Statistic from Tibshirani, Wal
ther, Hastie
    Params:
        data: ndarry of shape (n_samples, n_features)
        nrefs: number of sample reference datasets to create
        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 ge
tting resulting 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}, igno
re_index=True)

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

In [201]:
```
k, gapdf = optimalK(X, nrefs=10, maxClusters=30)
print('Optimal k is: ' + str(k))
```

```
Optimal k is: 29
```

In [202]:
```
plt.plot(gapdf.clusterCount, gapdf.gap, linewidth=3)
plt.scatter(gapdf[gapdf.clusterCount == k].clusterCount,
            gapdf[gapdf.clusterCount == k].gap, s=250, c='r')
plt.grid(True)
plt.xlabel('Cluster Count')
plt.ylabel('Gap Value')
plt.title('Gap Values by Cluster Count')
plt.show()
```



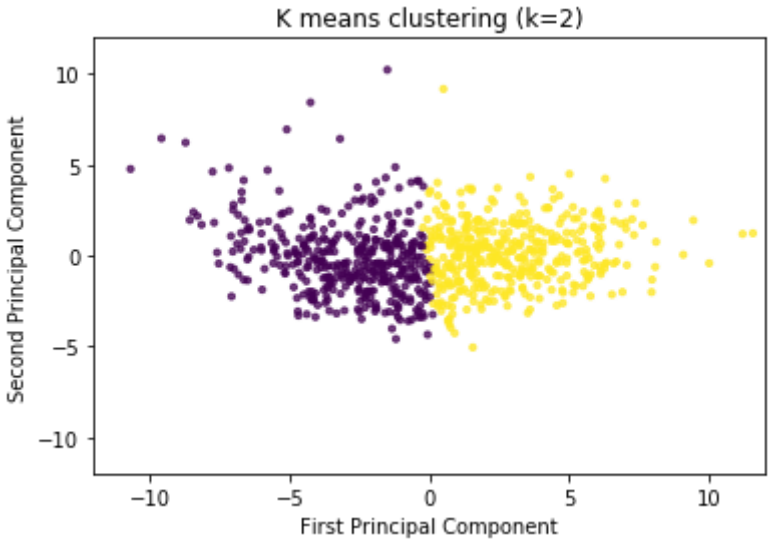
The optimal values of k yielded by the three methdos above are 14, 2 and 29 respectively.

11.

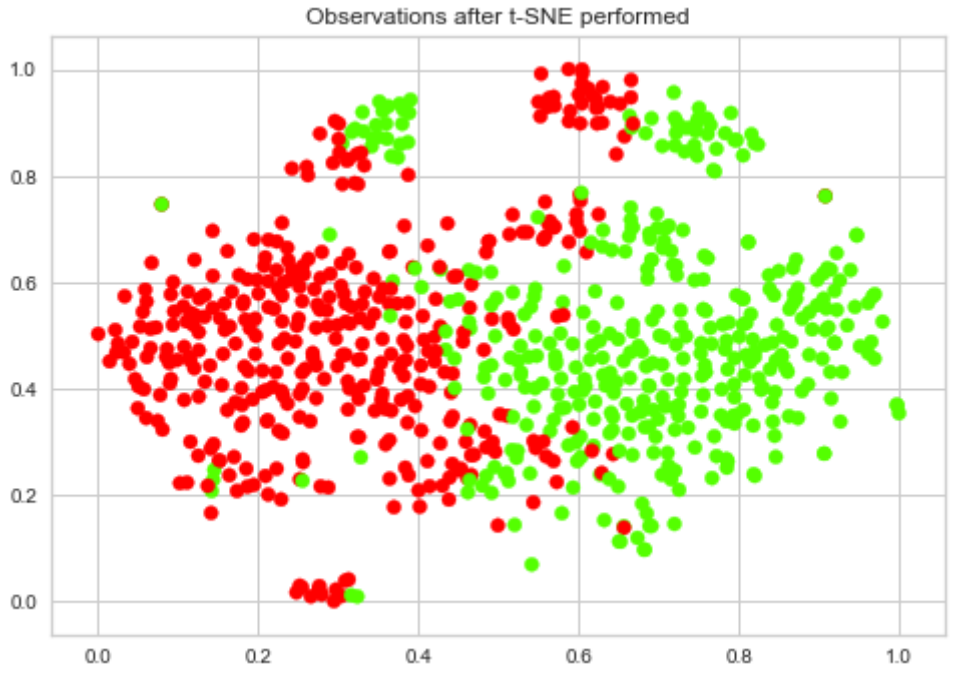
Visualize the results of the optimal ˆk-means clustering model. First use the first and second principal components from PCA, and color-code each observation based on their cluster membership. Next use the first and second dimensions from t-SNE, and color-code each observation based on their cluster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

In [183]:
```
perform_kmc(2)
```

```
Sum of squared distances of samples to their closest cluster center is:
38704.38907656803
```



In [224]:
```
label = KMeans(n_clusters=2, n_init=20).fit(X).labels_

plt.scatter(X_norm[:, 0], X_norm[:, 1], c=label, cmap=plt.cm.prism)
plt.title('Observations after t-SNE performed')

plt.show()
```



K-means clustering after different dimension-reduction methods yields quite different results.

For K-means after PCA, the plotted obsercations are clustered pretty clearly and the border is vertical line near PC1 = 0, indicating that the first principal component is important for telling the subgroups apart. The two clustered can basically be seperated by whether it has positive or negative value on the first principal component.

The k-means after t-SNE however, yields a quite perplex plot with no clear boundary. This is reasonable since t-SNE does not preserve distances nor density. It only to some extent preserves nearest-neighbors. This affects any density- or distance based algorithm such as k-means. Generally speaking, use t-SNE for visualization (and try different parameters) could be helpful, but rather do not run clustering afterwards, in particular do not use distance- or density based algorithms, as this information was intentionally lost.

Notes:

https://stats.stackexchange.com/questions/263539/clustering-on-the-output-of-t-sne/264647#264647 (https://stats.stackexchange.com/questions/263539/clustering-on-the-output-of-t-sne/264647#264647)

In [ ]: