```
In [19]: import pandas as pd
         import numpy as np
         import math
         import matplotlib.pyplot as plt
```

k-Means Clustering "By Hand"

```
In [20]: # data
         input_1 = [5,8,7,8,3,4,2,3,4,5]
         input_2 = [8,6,5,4,3,2,2,8,9,8]
```

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

```
In [21]: from numpy import random
         seed = random.seed(123)
```

```
In [22]: data = pd.DataFrame(input_1, input_2)
         data = data.reset_index()

         data = data.rename(columns={'index': 'input_1', 0: 'input_2'})
```

```
In [23]: cluster_labels = np.random.choice(3, 10, replace=True)
         data['label'] = cluster_labels
         data
```

Out[23]:

	input_1	input_2	label
0	8	5	2
1	6	8	1
2	5	7	2
3	4	8	2
4	3	3	0
5	2	4	2
6	2	2	2
7	8	3	1
8	9	4	2
9	8	5	1

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

```
In [24]: def calculate_kmeans(k, data, iterations):

             for i in range(max_iter):
                     # Calculate centroilds
                 centroids = {}
                 for k in range(num_k):
                     cent1 = data[data['label'] == k]['input_1'].mean()
                     cent2 = data[data['label'] == k]['input_2'].mean()
                     centroids[k] = (cent1, cent2)
                     # Update k labels for each observation
                 new_k_labels = []
                 for idx, row in data.iterrows():
                     distance = 50
                     for k, v in centroids.items():
                         euclidean = math.sqrt((row['input_1'] - v[0]) ** 2 + (ro
         w['input_2'] - v[1]) ** 2)
                         if euclidean < min_distance:
                             distance = euclidean
                             new_k = k
                     new_k_labels.append(new_k)
                 data['k_label'] = new_k_labels
             return data
```

```
In [25]: data = calculate_kmeans(3, data, 50)
```

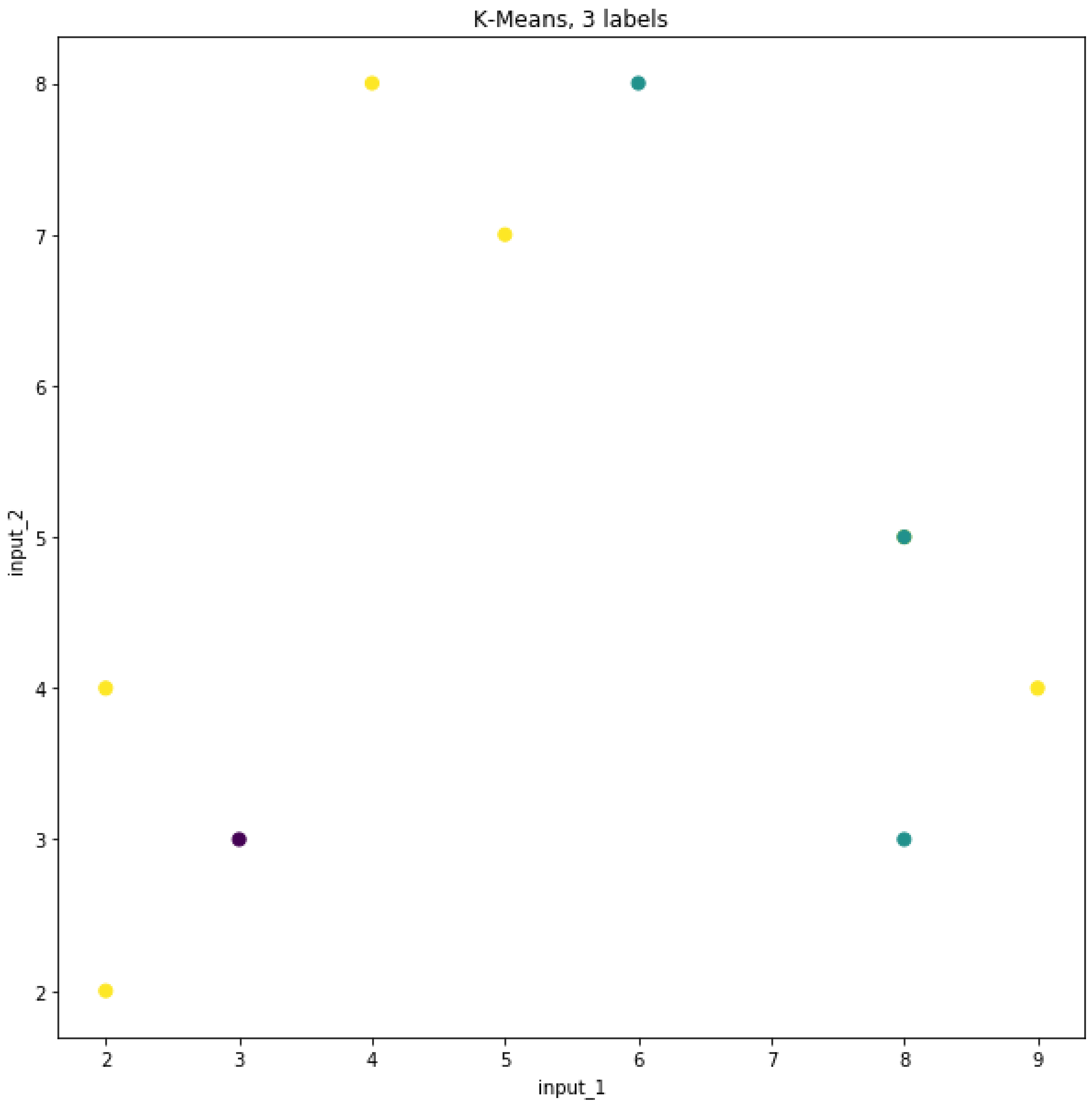
```
In [26]: data
```

Out[26]:

	input_1	input_2	label	k_label
0	8	5	2	1
1	6	8	1	1
2	5	7	2	2
3	4	8	2	2
4	3	3	0	0
5	2	4	2	0
6	2	2	2	0
7	8	3	1	1
8	9	4	2	1
9	8	5	1	1

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

In [51]:
```
fig = plt.figure(figsize=(10, 10))
colors = list(data['label'])
plt.scatter(data['input_1'], data['input_2'], c=colors, s=50)
plt.xlabel('input_1')
plt.ylabel('input_2')
plt.title('K-Means, 3 labels')
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 [28]:
```
np.random.seed(123)
data_2 = pd.DataFrame({'input_1': input_1, 'input_2': input_2})
new_labels = np.random.choice(2, 10, replace=True)
```

In [29]:
```
data_2['label'] = new_labels
data_2
```

Out[29]:

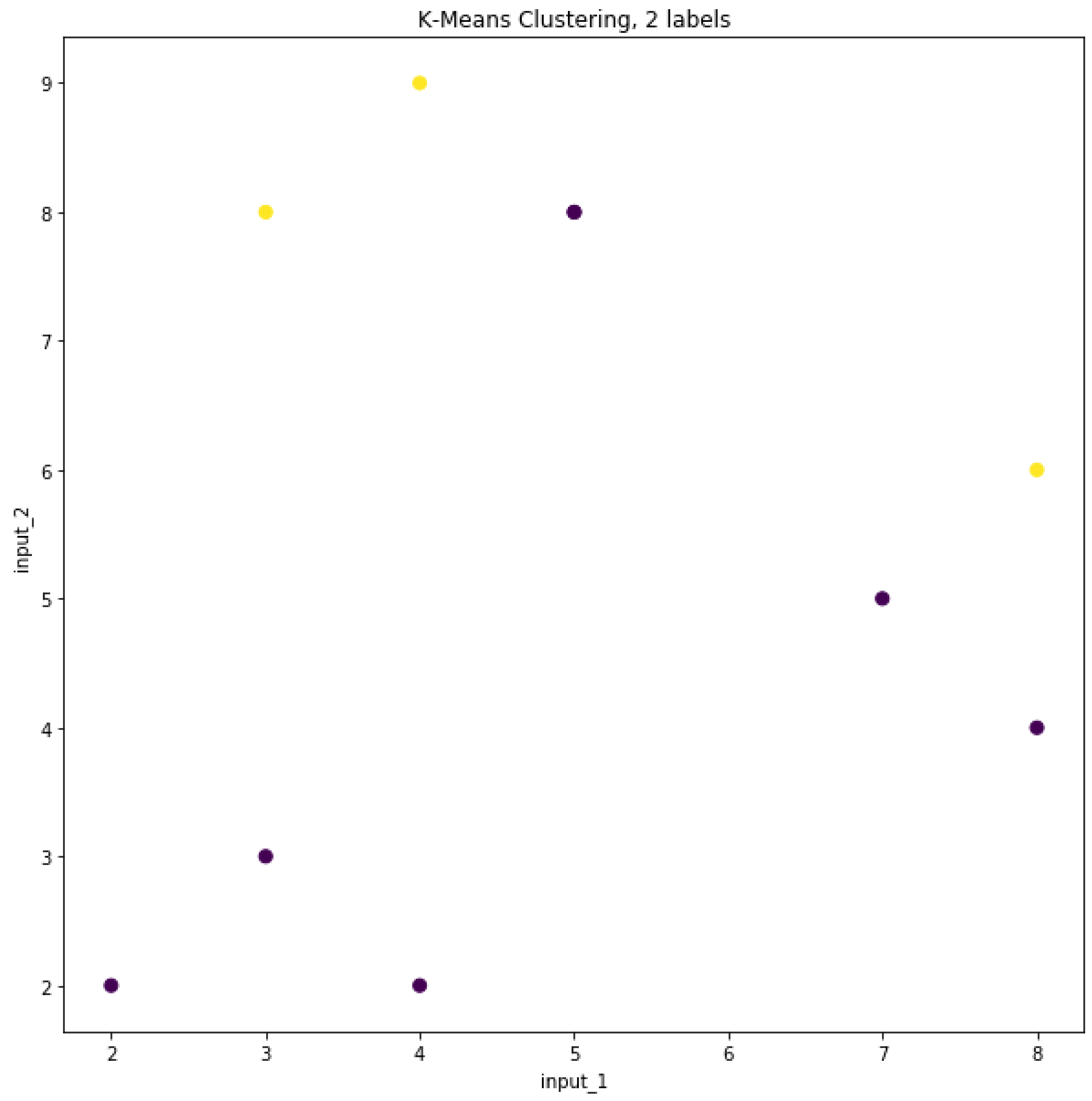
	input_1	input_2	label
0	5	8	0
1	8	6	1
2	7	5	0
3	8	4	0
4	3	3	0
5	4	2	0
6	2	2	0
7	3	8	1
8	4	9	1
9	5	8	0

In [30]:
```
k2_fitted = calculate_kmeans(2, data_2, 50)
k2_fitted
```

Out[30]:

	input_1	input_2	label	k_label
0	5	8	0	1
1	8	6	1	1
2	7	5	0	1
3	8	4	0	1
4	3	3	0	0
5	4	2	0	0
6	2	2	0	0
7	3	8	1	0
8	4	9	1	1
9	5	8	0	1

```
In [50]: fig = plt.figure(figsize=(10, 10))
         colors = list(data_2['label'])
         plt.scatter(data_2['input_1'], data_2['input_2'], c=colors, s=50)
         plt.xlabel('input_1')
         plt.ylabel('input_2')
         plt.title('K-Means Clustering, 2 labels')
         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?

For the dataset given with input_1 and input_2, it looks like three clusters is the best bet due to the clear distance between clusters. The top and bottom cluster spatial difference with two clusters is quite far. 3 clusters, even with randomly assigned label makes more sense.

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

In [32]:
```
import pandas as pd

wiki = pd.read_csv('./data/wiki.csv')
```

In [33]:
```
len(wiki.columns)
```

Out[33]:
```
57
```

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?

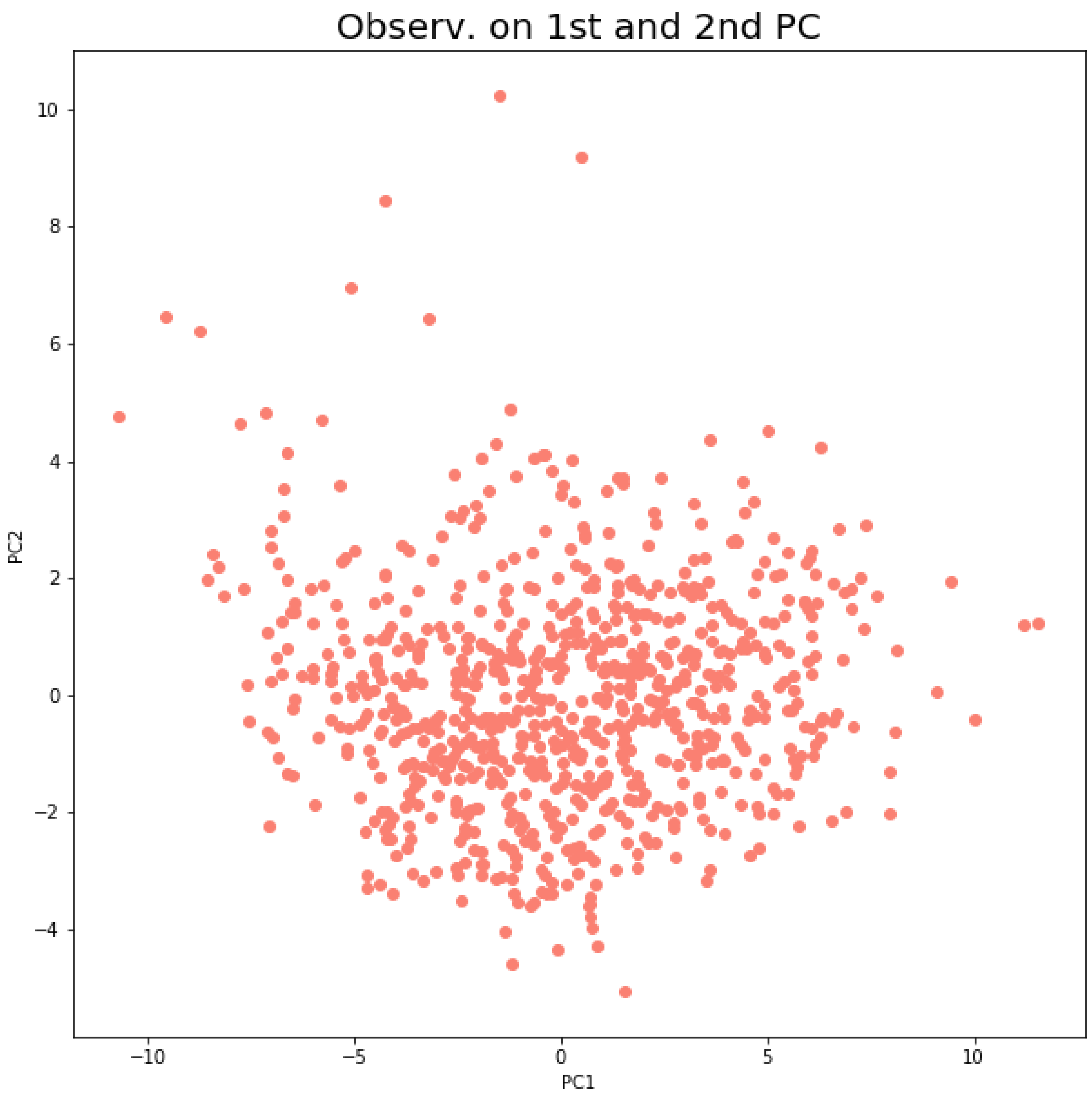
In [34]:
```
from sklearn.preprocessing import StandardScaler
wiki = StandardScaler().fit_transform(wiki)

wiki
```

Out[34]:
```
array([[-0.28718866, -0.86413245,  1.14257407, ..., -0.15171652,
        -0.05006262, -2.1618878 ],
       [-0.02204045, -0.86413245,  1.14257407, ..., -0.15171652,
        -0.05006262, -2.1618878 ],
       [-0.68491098, -0.86413245,  1.14257407, ..., -0.15171652,
        -0.05006262, -2.1618878 ],
       ...,
       [ 0.9059783 ,  1.15723001,  1.14257407, ..., -0.15171652,
        -0.05006262,  0.46255869],
       [-0.02204045,  1.15723001, -0.87521678, ..., -0.15171652,
        -0.05006262,  0.46255869],
       [ 0.37568187,  1.15723001,  1.14257407, ..., -0.15171652,
        -0.05006262,  0.46255869]])
```

In [35]:
```
# set model
from sklearn.decomposition import PCA
pca_mod = PCA(random_state = seed)
```

```
In [36]: # train model
         wiki_pca_reduced = pca_mod.fit_transform(wiki)
```

```
In [37]: # plot
         plt.figure(figsize=(10,10))
         plt.title('Observ. on 1st and 2nd PC', fontsize=20)
         plt.scatter(wiki_pca_reduced[:,0], wiki_pca_reduced[:,1], c='salmon')
         plt.xlabel('PC1')
         plt.ylabel('PC2');
```



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 [69]: wiki
```

```
Out[69]: array([[-0.28718866, -0.86413245,  1.14257407, ..., -0.15171652,
                 -0.05006262, -2.1618878 ],
                [-0.02204045, -0.86413245,  1.14257407, ..., -0.15171652,
                 -0.05006262, -2.1618878 ],
                [-0.68491098, -0.86413245,  1.14257407, ..., -0.15171652,
                 -0.05006262, -2.1618878 ],
                ...,
                [ 0.9059783 ,  1.15723001,  1.14257407, ..., -0.15171652,
                 -0.05006262,  0.46255869],
                [-0.02204045,  1.15723001, -0.87521678, ..., -0.15171652,
                 -0.05006262,  0.46255869],
                [ 0.37568187,  1.15723001,  1.14257407, ..., -0.15171652,
                 -0.05006262,  0.46255869]])
```

```
In [60]: # obtaining loading vectors for all PCs
         components = pd.DataFrame(pca_mod.components_[0])
```

```
In [61]: components[0].sort_values(ascending=False).head()
```

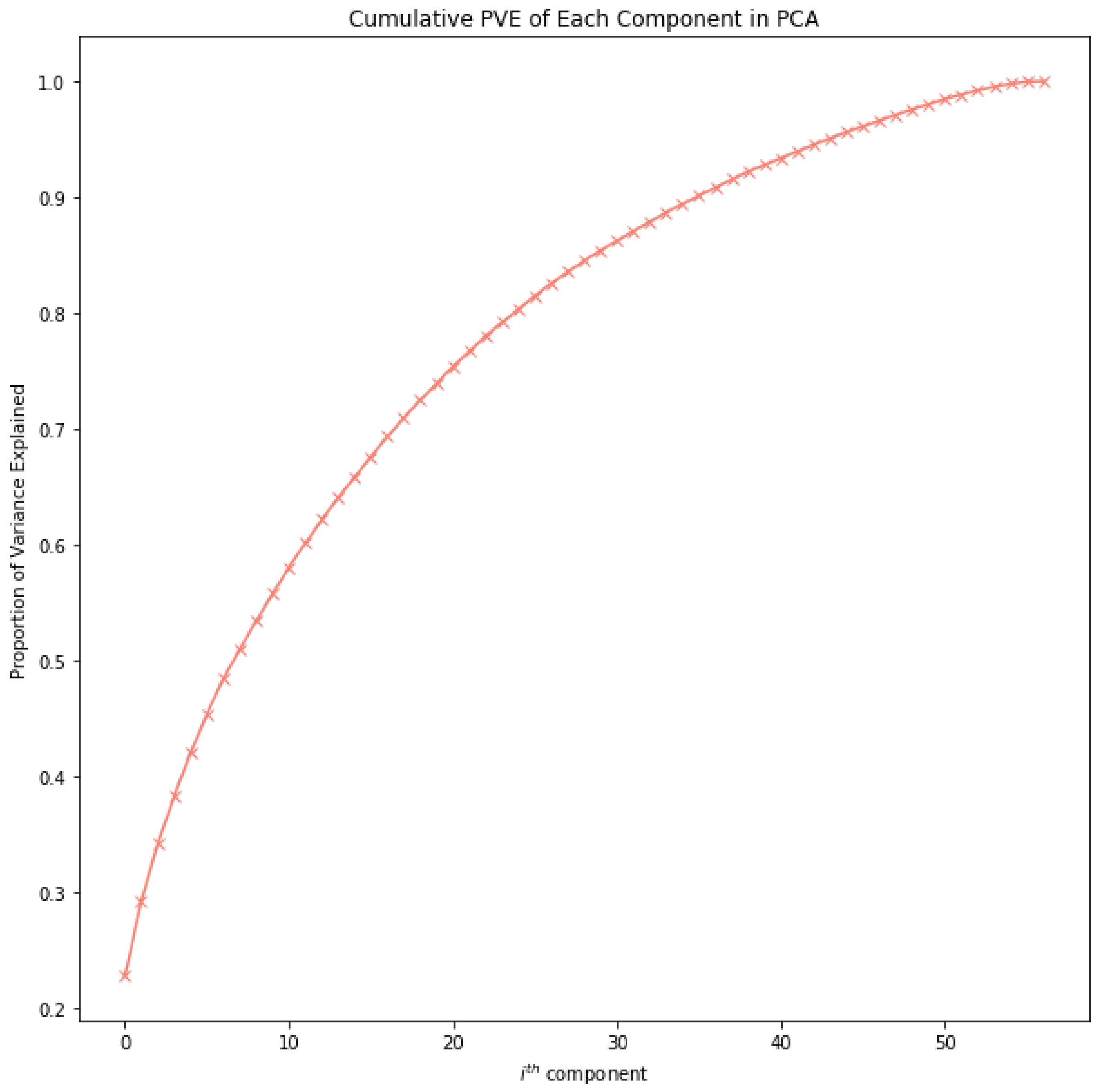
```
Out[61]: 38    0.230924
         37    0.226193
         29    0.218809
         30    0.214558
         7     0.210863
         Name: 0, dtype: float64
```

According to the wiki columns, b2 (38) and b1 (37) are the best predictors because they seem to be correlated with the first component the most. After that, user behvaior regarding students and colleagues Wiki usage is then calculated to be strongly correlated.

```
In [70]: components2 = pca_mod.components_[1]

         components2
```

```
Out[70]: array([ 0.08838498, -0.14946145,  0.0304355 ,  0.06236471,  0.13438736,
                 0.00827305,  0.01766879,  0.02877629, -0.27174129, -0.22236798,
                -0.06845852, -0.15101173, -0.22760242, -0.03812243, -0.06642188,
                -0.03347235, -0.10345841,  0.01091179, -0.02520763, -0.05621777,
                 0.19763474,  0.11110615, -0.05977452,  0.0440036 , -0.22992601,
                -0.22676039, -0.24232542,  0.1978275 ,  0.2186285 ,  0.15515157,
                 0.16086452,  0.02982325,  0.11437078,  0.01860471,  0.09417252,
                -0.13696754, -0.10629682,  0.05637427,  0.08343089, -0.24543982,
                -0.2020214 , -0.22098579, -0.20202201,  0.07054384, -0.02956048,
                -0.12641691,  0.22849427,  0.07609569, -0.01453674, -0.0154785 ,
                 0.1714838 , -0.01488715,  0.01313418,  0.00231128, -0.02359103,
                -0.00378453, -0.00530122])
```

In [77]:
```
#PVE

cumulative_variance_explained = np.cumsum(pca_mod.explained_variance_rat
io_)

plt.figure(figsize=(10,10))
plt.plot(np.arange(57), cumulative_variance_explained, marker='x',color=
'salmon')
plt.title('Cumulative PVE of Each Component in PCA')
plt.xlabel('$i^{th}$ component')
plt.ylabel('Proportion of Variance Explained');
```



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

In [39]:
```
#t-SNE analysis
from sklearn.manifold import TSNE
```

In [40]:
```
tsne = TSNE(n_components =2, perplexity = 2, random_state = seed)
```

In [41]:
```
wiki_tsne = tsne.fit_transform(wiki)
```

In [42]:
```
tsneDF = pd.DataFrame(wiki_tsne)
tsneDF
```

Out[42]:

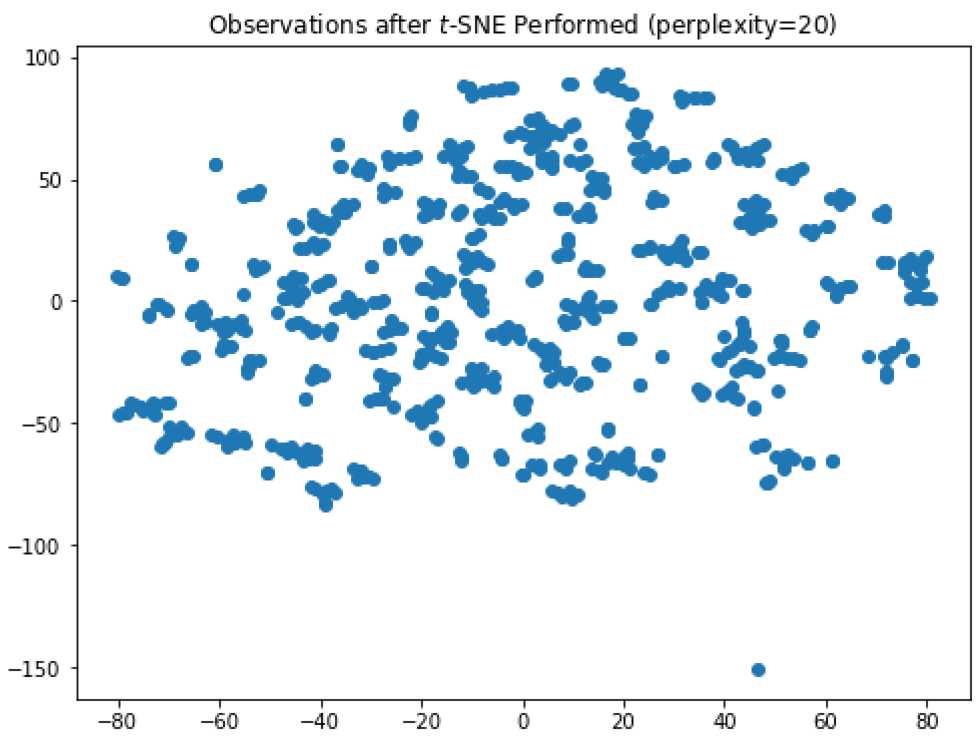
	0	1
0	-41.001713	-61.339176
1	-75.189201	-44.971779
2	43.843822	-11.569514
3	35.048393	-36.497978
4	-18.168839	-47.383102
...	...	...
795	12.292333	-1.650578
796	-70.310249	-3.670918
797	-68.567650	23.890991
798	-30.043873	14.467270
799	71.622383	34.581036

800 rows × 2 columns

In [43]:
```
tsneDF = tsneDF.rename(columns={0:"first", 1:"second"})
```

In [44]:
```
plt.figure(figsize=(8,6))
plt.title('Observations after $t$-SNE Performed (perplexity=20)')
plt.scatter(x='first', y='second', data=tsneDF)
```

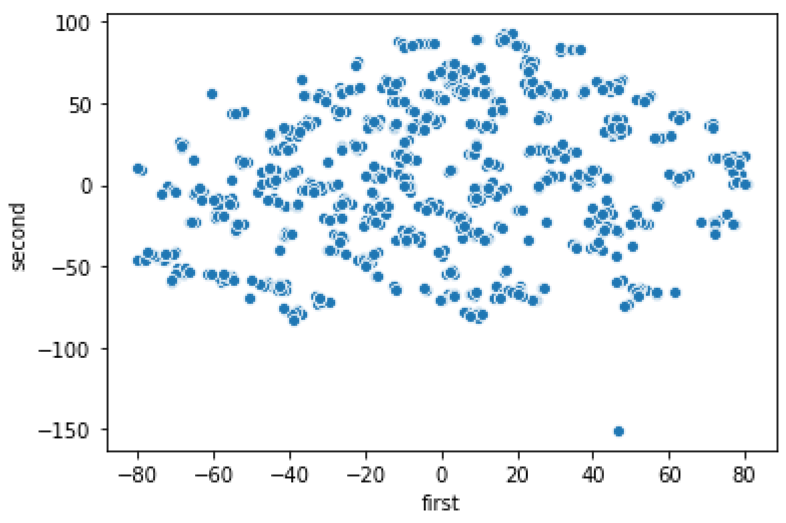
Out[44]:
```
<matplotlib.collections.PathCollection at 0x1a200db1d0>
```



It's hard to see, so let me try another package.

In [45]:
```
import seaborn as sns

sns.scatterplot(x='first', y='second', data=tsneDF)
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a202b3650>
```



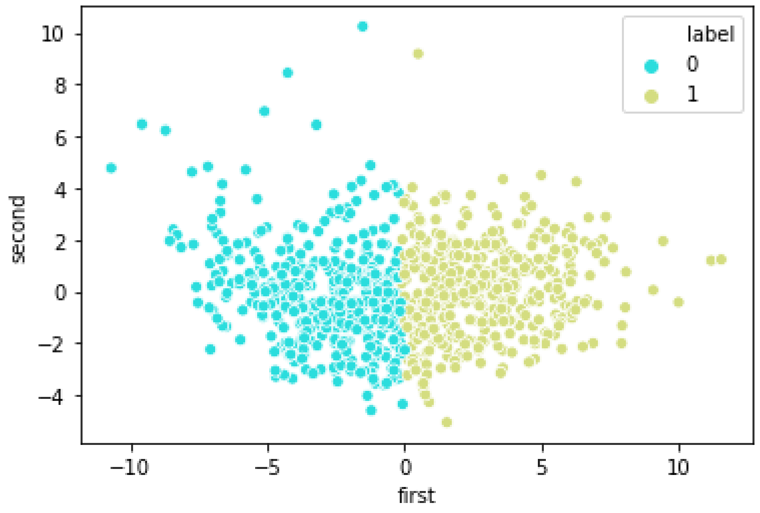
TSNE, as opposed to kmeans, is able to optimize for both how far clusters are and how groups seem to coalesce.

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.

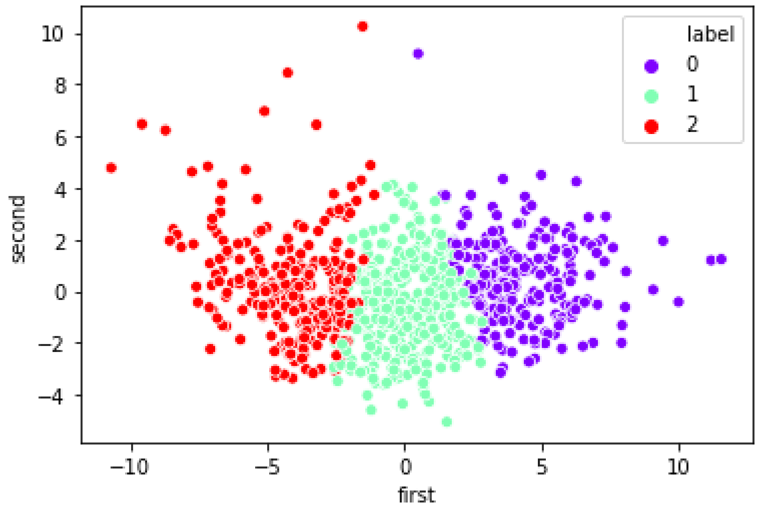
In [46]:
```
from sklearn.cluster import KMeans

#k_nums = [2,3,4]
#for k in k_nums:

wiki_kmeans = pd.DataFrame(wiki_pca_reduced[:, 0:2],
                           columns=["first", "second"])

kmeans = KMeans(n_clusters= 2, random_state = seed)


wiki_kmeans['label'] = kmeans.fit_predict(wiki)

sns.scatterplot(x='first', y='second', hue='label',
                data=wiki_kmeans, palette='rainbow')
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a20314f50>
```



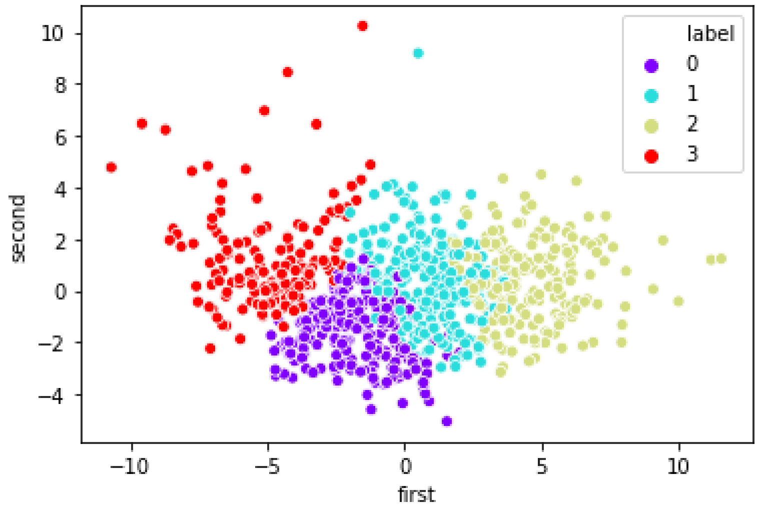
```
In [47]: kmeans = KMeans(n_clusters= 3, random_state = seed)


         wiki_kmeans['label'] = kmeans.fit_predict(wiki)

         sns.scatterplot(x='first', y='second', hue='label',
                         data=wiki_kmeans, palette='rainbow')
```

```
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20fb8750>
```



```
In [48]: kmeans = KMeans(n_clusters= 4, random_state = seed)


         wiki_kmeans['label'] = kmeans.fit_predict(wiki)

         sns.scatterplot(x='first', y='second', hue='label',
                         data=wiki_kmeans, palette='rainbow')
```

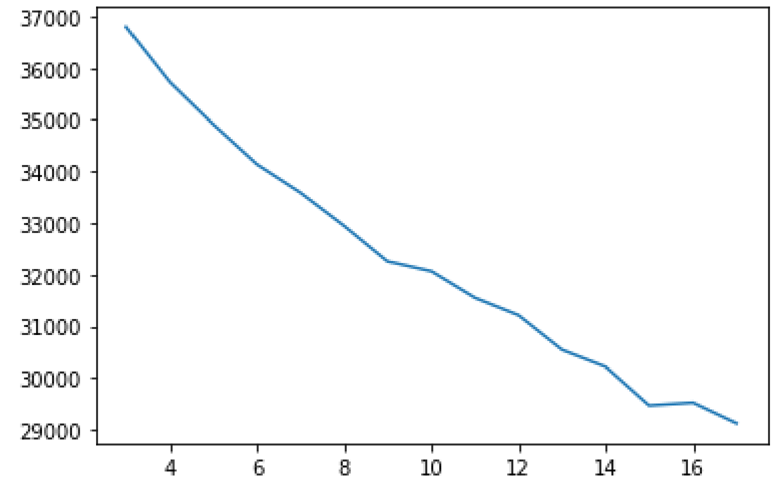
```
Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20fe5950>
```



A k=4 classification seems to be the best fit for the data. But to be honest it's not clear that the linear separation between clusters is optimized at different values of k.

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 [102]:
```
#elbow

df = wiki
imin, imax = 3, 18
distortions = []

for i in range(imin, imax):
    km = KMeans(n_clusters=i)
    km.fit(df)
    distortions.append(km.inertia_)

plt.plot(range(imin, imax), distortions)
plt.show()

best_n = np.argmax(np.gradient(distortions))
print('best n:', best_n)
kmodel = KMeans(n_clusters=best_n)
best_k_clusters = kmodel.fit_predict(df)
```

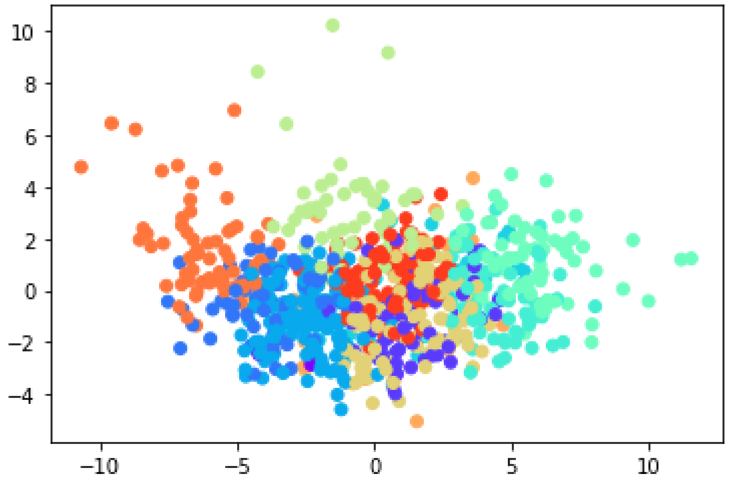


```
best n: 13
```

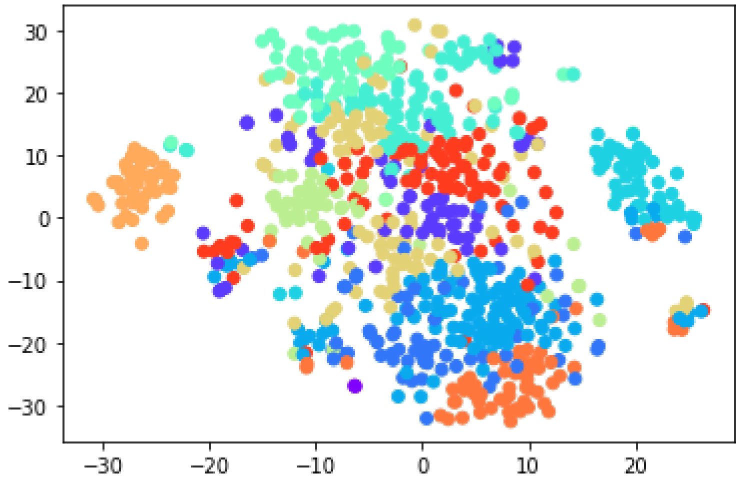
In [91]:
```
# I genuinely do not understand elbow method, Waggoner went through it s
o dast.
```

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 [106]:
```
pmodel2 = PCA(n_components=2).fit_transform(df)

colors = dict(zip(range(best_n), [plt.cm.rainbow(i/best_n) for i in rang
e(0, best_n+1)]))
cols = [colors[i] for i in best_k_clusters]

plt.scatter(pmodel2[:,0], pmodel2[:,1], c=cols)
plt.show()
```



In [107]:
```
tmodel2 = TSNE(n_components=2).fit_transform(df)

colors = dict(zip(range(best_n), [plt.cm.rainbow(i/best_n) for i in rang
e(0, best_n+1)]))
cols = [colors[i] for i in best_k_clusters]

plt.scatter(tmodel2[:,0], tmodel2[:,1], c=cols)
plt.show()
```



PCA looks at variance in the same space,whereas TSNE looks at the difference in probability distirbutions. PCA is linear and TSNE is not linear.

In [ ]: