1. K-means by hand

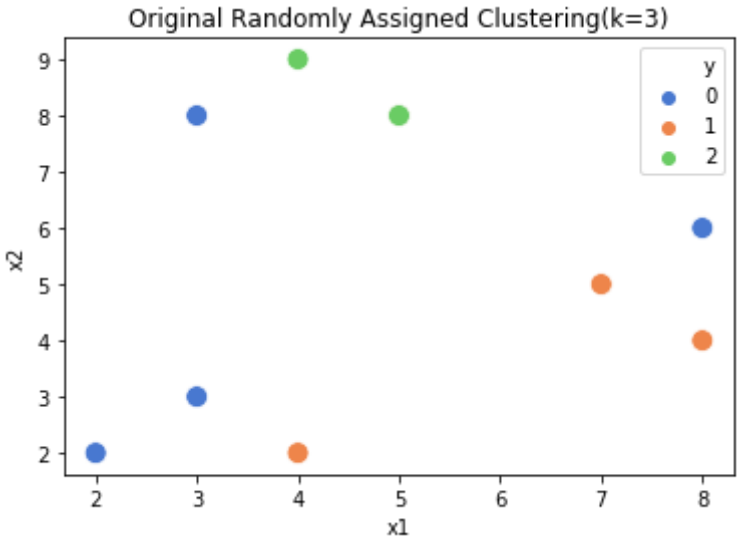
In [160]:
```
import pandas as pd
import numpy as np
import math
import seaborn as sns
import matplotlib.pyplot as plt
```

In [145]:
```
np.random.seed(5566)
df = pd.DataFrame({'x1':[5,8,7,8,3,4,2,3,4,5], 'x2':[8,6,5,4,3,2,2,8,9,8]})
```

In [153]:
```
def fit_kmeans(df, iter_n, k):

    for i in range(iter_n):
        df['y']= np.random.choice(k,df.shape[0])
        df_old = df.copy()
        centroids = get_centroids(df,k)
        df = fit(df,centroids)

    return df, df_old

def get_centroids(df,k):
    cen = {}
    for label in range(k):
        cen_x1 = df[df['y'] == label]['x1'].mean()
        cen_x2 = df[df['y'] == label]['x2'].mean()
        cen[label] = (cen_x1, cen_x2)

    return cen

def fit(df,centroids):

    for idx, row in df.iterrows():
        min_dist = math.inf
        dist_ = {}

        for label, cen in centroids.items():
            dist = math.sqrt((row['x1'] - cen[0]) ** 2 + (row['x2'] - cen[1
            dist_[label] = dist

        row['y'] = min(dist_, key=dist_.get)

    return df
```

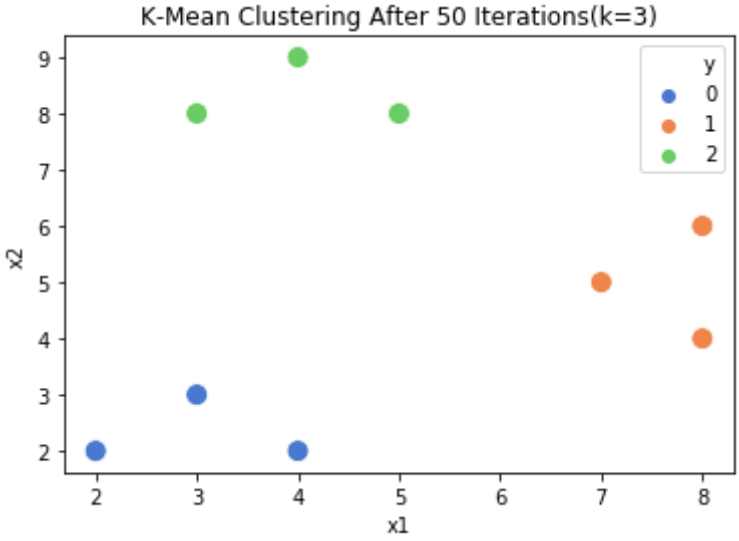
In [170]:
```
# with k = 3 iterate 50 times
m = fit_kmeans(df, 50, 3)
sns.scatterplot(m[1]['x1'], m[1]['x2'], hue = m[1]['y'], s =120, palette =
plt.title('Original Randomly Assigned Clustering(k=3)')
```

Out[170]:
```
Text(0.5, 1.0, 'Original Randomly Assigned Clustering(k=3)')
```



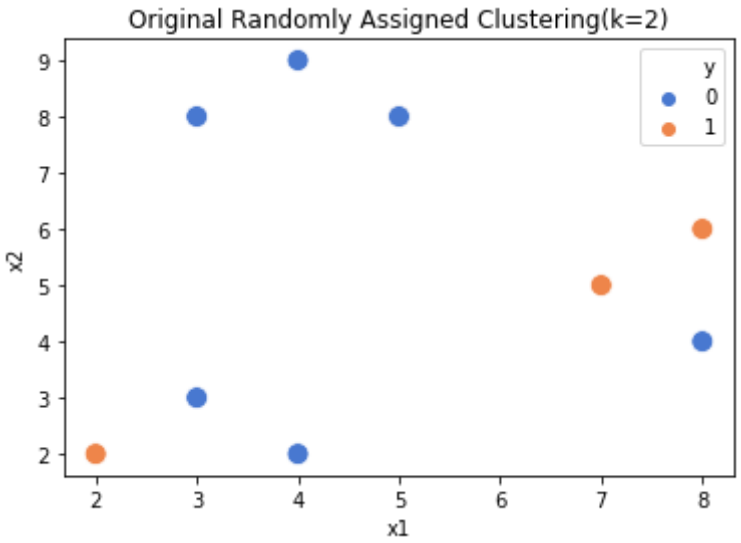
In [171]:
```
sns.scatterplot(m[0]['x1'], m[0]['x2'], hue = m[0]['y'], s =120, palette =
plt.title('K-Mean Clustering After 50 Iterations(k=3)')
```

Out[171]:
```
Text(0.5, 1.0, 'K-Mean Clustering After 50 Iterations(k=3)')
```



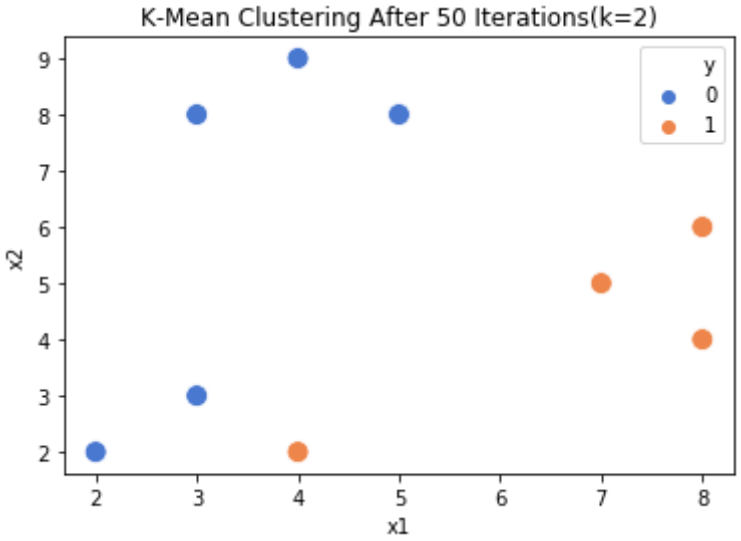
In [173]:
```
# with k = 2 iterate 50 times
m2 = fit_kmeans(df, 50, 2)
sns.scatterplot(m2[1]['x1'], m[1]['x2'], hue = m2[1]['y'], s =120, palette
plt.title('Original Randomly Assigned Clustering(k=2)')
```

Out[173]:
```
Text(0.5, 1.0, 'Original Randomly Assigned Clustering(k=2)')
```



In [174]:
```
sns.scatterplot(m2[0]['x1'], m2[0]['x2'], hue = m2[0]['y'], s =120, palette
plt.title('K-Mean Clustering After 50 Iterations(k=2)')
```

Out[174]:
```
Text(0.5, 1.0, 'K-Mean Clustering After 50 Iterations(k=2)')
```



As we can see, the original distribution of the data is obviously spatially 3 clusters. The k=3 model performed really well, sorting the data's into 3 clusters after the iterations. The k=2 model, however, with the same number of iterations,seperated the datas with a clear-cut hyper-plane(2-d line) into lower right and upper left, resulting in a heterogenius lower-left cluster, obviously not an ideal clustering. This is becuase the k=2 clustering could not capture the true relation.

2. Application

In [283]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

Dimension Reduction

In [284]:
```
x = pd.read_csv('wiki.csv')
columns = x.columns
num_feat = x.shape[1]
x = StandardScaler().fit_transform(x)
pca =PCA(n_components = 2, random_state = 5566).fit(x)
df = pd.DataFrame(data = pca.components_, columns = columns, index = ['vect
df.head()
```

Out[284]:

	age	gender	phd	yearsexp	userwiki	pu1	pu2	pu3	pe
vector1	-0.021805	-0.035086	-0.030501	-0.034190	0.081363	0.192827	0.190588	0.210863	0.0612
vector2	0.088380	-0.149480	0.030468	0.062369	0.134421	0.008262	0.017657	0.028764	-0.2717

2 rows × 57 columns

6. Top 5 loading

In [237]:
```
print('Top 5 loading features in vector 1')
print('==================================')
print(abs(df.loc['vector1']).sort_values(ascending = False).head())
```

```
Top 5 loading features in vector 1
==================================
bi2     0.230924
bi1     0.226193
use3    0.218809
use4    0.214558
pu3     0.210863
Name: vector1, dtype: float64
```

In [238]:
```
print('Top 5 loading features in vector 2')
print('==================================')
print(abs(df.loc['vector2']).sort_values(ascending = False).head())
```

```
Top 5 loading features in vector 2
==================================
peu1    0.271731
inc1    0.245429
sa3     0.242334
sa1     0.229933
exp4    0.228526
Name: vector2, dtype: float64
```

7. Variance explained

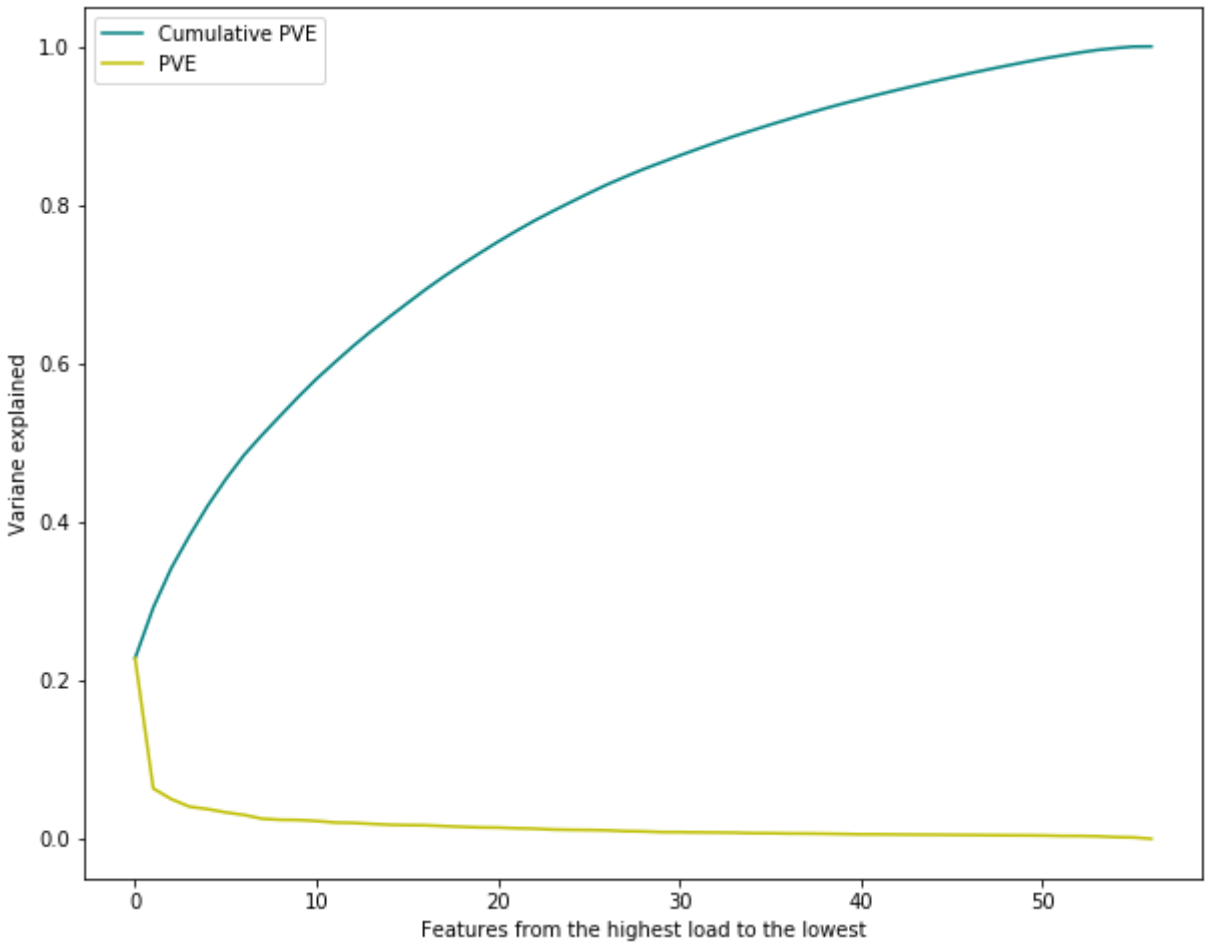
In [291]:
```
pca_all =PCA( random_state = 5566).fit(x)
dfall = pca_all.explained_variance_ratio_
cumu = np.cumsum(dfall)
pd.DataFrame({'Variance explained':dfall,'Cumulative Variance Explained':cu
```

Out[291]:

	Variance explained	Cumulative Variance Explained
0	0.228106	0.228106
1	0.063725	0.291831
2	0.050237	0.342068
3	0.040728	0.382796
4	0.037677	0.420474
5	0.033521	0.453995
6	0.030331	0.484326
7	0.025522	0.509848
8	0.024174	0.534022
9	0.023925	0.557947
10	0.022657	0.580604
11	0.020712	0.601316
12	0.020280	0.621595
13	0.019033	0.640629
14	0.017925	0.658554
15	0.017477	0.676030
16	0.017263	0.693294
17	0.016192	0.709486
18	0.015285	0.724770
19	0.014578	0.739348
20	0.014330	0.753679
21	0.013497	0.767176
22	0.012961	0.780137
23	0.011926	0.792062
24	0.011469	0.803531
25	0.011293	0.814824
26	0.010855	0.825680
27	0.009881	0.835561
28	0.009519	0.845080
29	0.008663	0.853742
30	0.008635	0.862377

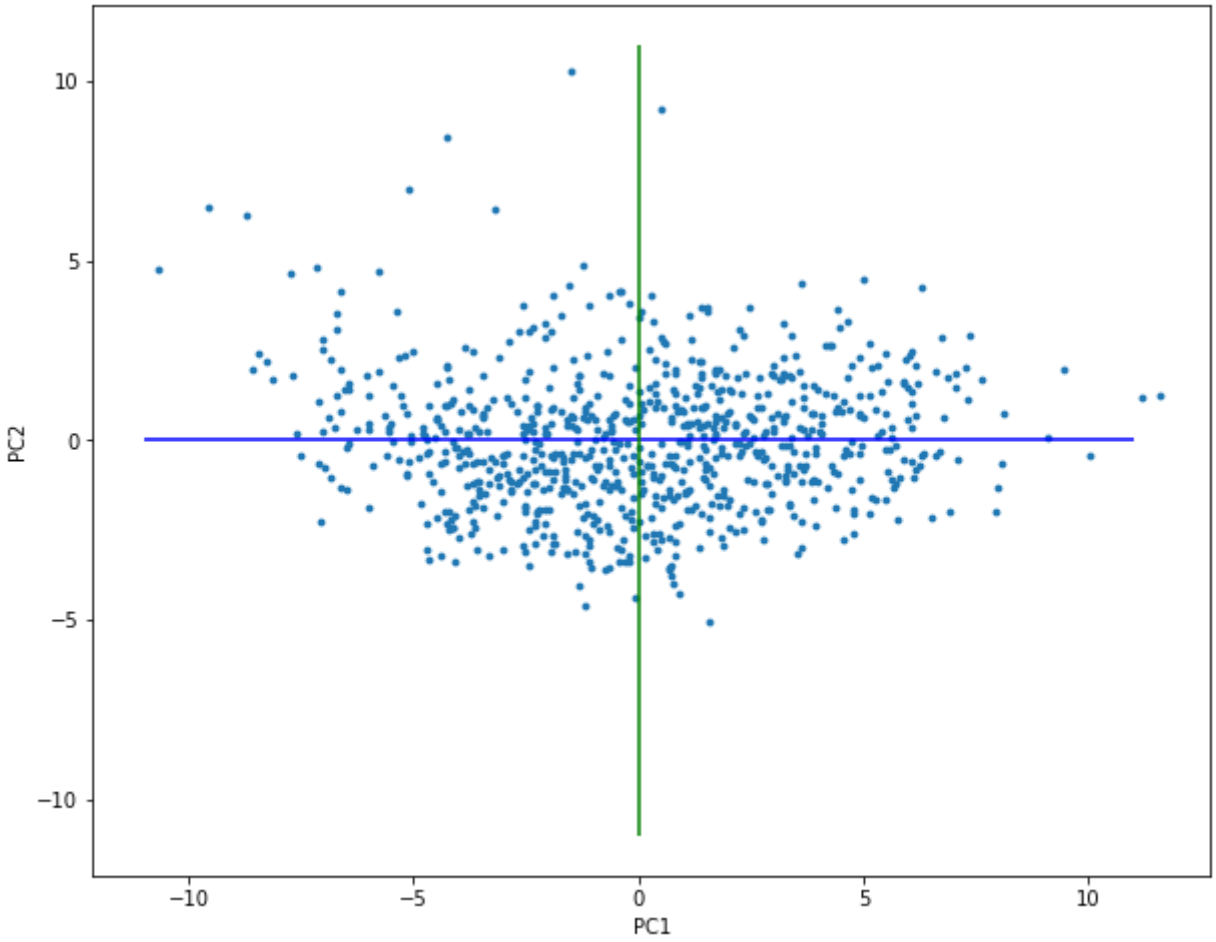
	Variance explained	Cumulative Variance Explained
31	0.008299	0.870676
32	0.008161	0.878837
33	0.007895	0.886732
34	0.007333	0.894066
35	0.007273	0.901338
36	0.006917	0.908255
37	0.006816	0.915072
38	0.006607	0.921678
39	0.006250	0.927928
40	0.005824	0.933752
41	0.005810	0.939562
42	0.005600	0.945163
43	0.005426	0.950589
44	0.005389	0.955978
45	0.005121	0.961098
46	0.005059	0.966158
47	0.004800	0.970958
48	0.004661	0.975619
49	0.004530	0.980150
50	0.004356	0.984506
51	0.003843	0.988349
52	0.003761	0.992110
53	0.003383	0.995493
54	0.002352	0.997845
55	0.001967	0.999812
56	0.000188	1.000000

In [300]:
```
plt.figure(figsize=(10, 8))
plt.plot(cumu, label = 'Cumulative PVE',C = 'teal')
plt.plot(dfall, label = 'PVE', C = 'y')
plt.xlabel('Features from the highest load to the lowest')
plt.ylabel('Variane explained')
plt.legend()
plt.show()
```



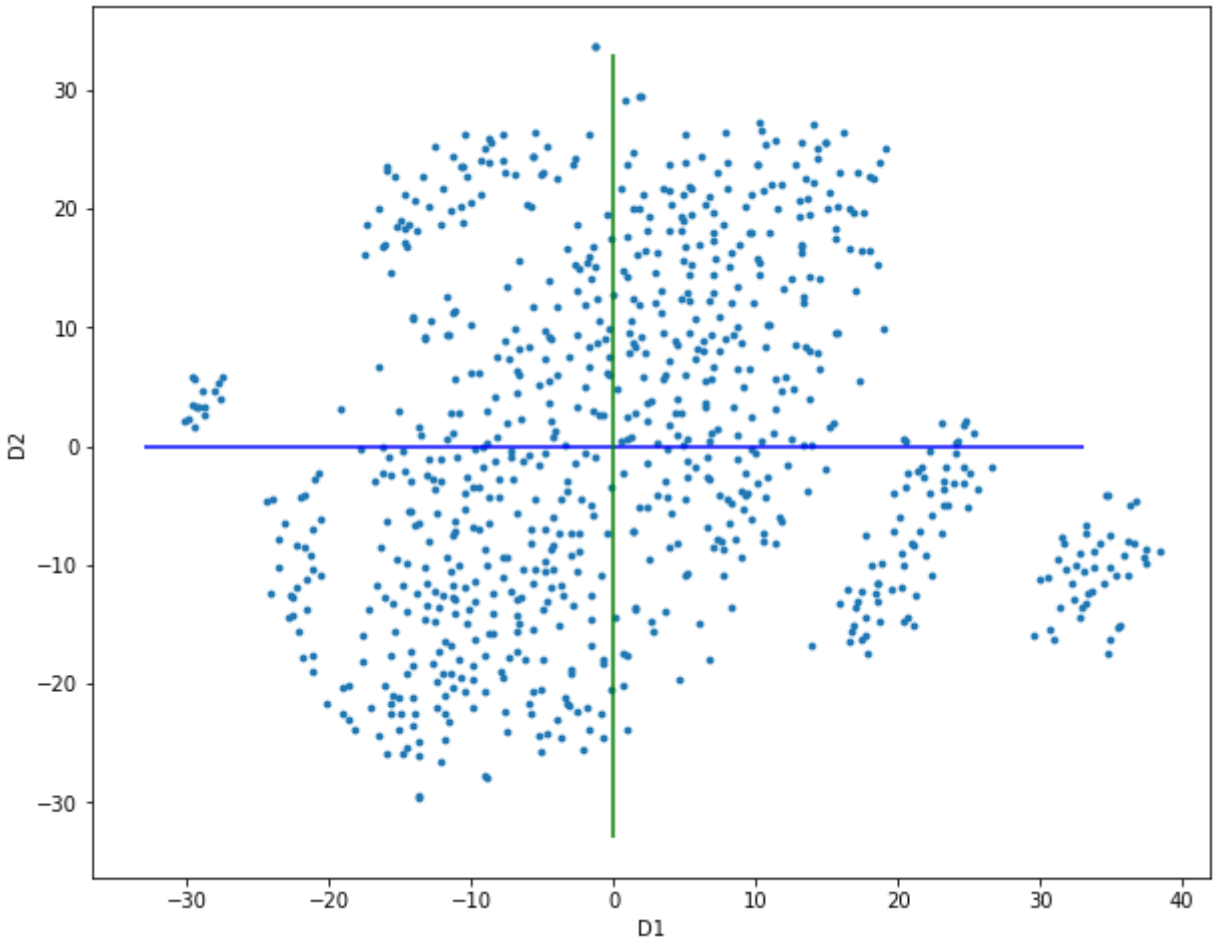
In [286]:
```
plt.figure(figsize=(10, 8))
pca_tran = pca.fit_transform(x)
pve = pca.explained_variance_ratio_

print('Variance explained by PC1(blue):',pve[0])
print('Variance explained by PC2(green):',pve[1])
pca_df = pd.DataFrame(pca_tran, columns=['PC1','PC2'])
sns.scatterplot(x = 'PC1', y = 'PC2', data = pca_df, linewidth = 0, s = 15)
plt.hlines(0,-11,11, colors = 'b')
plt.vlines(0,-11,11, colors = 'g')
plt.show();
```

```
Variance explained by PC1(blue): 0.22810627785663376
Variance explained by PC2(green): 0.06372474155878183
```



8. t-SNE VS PCA

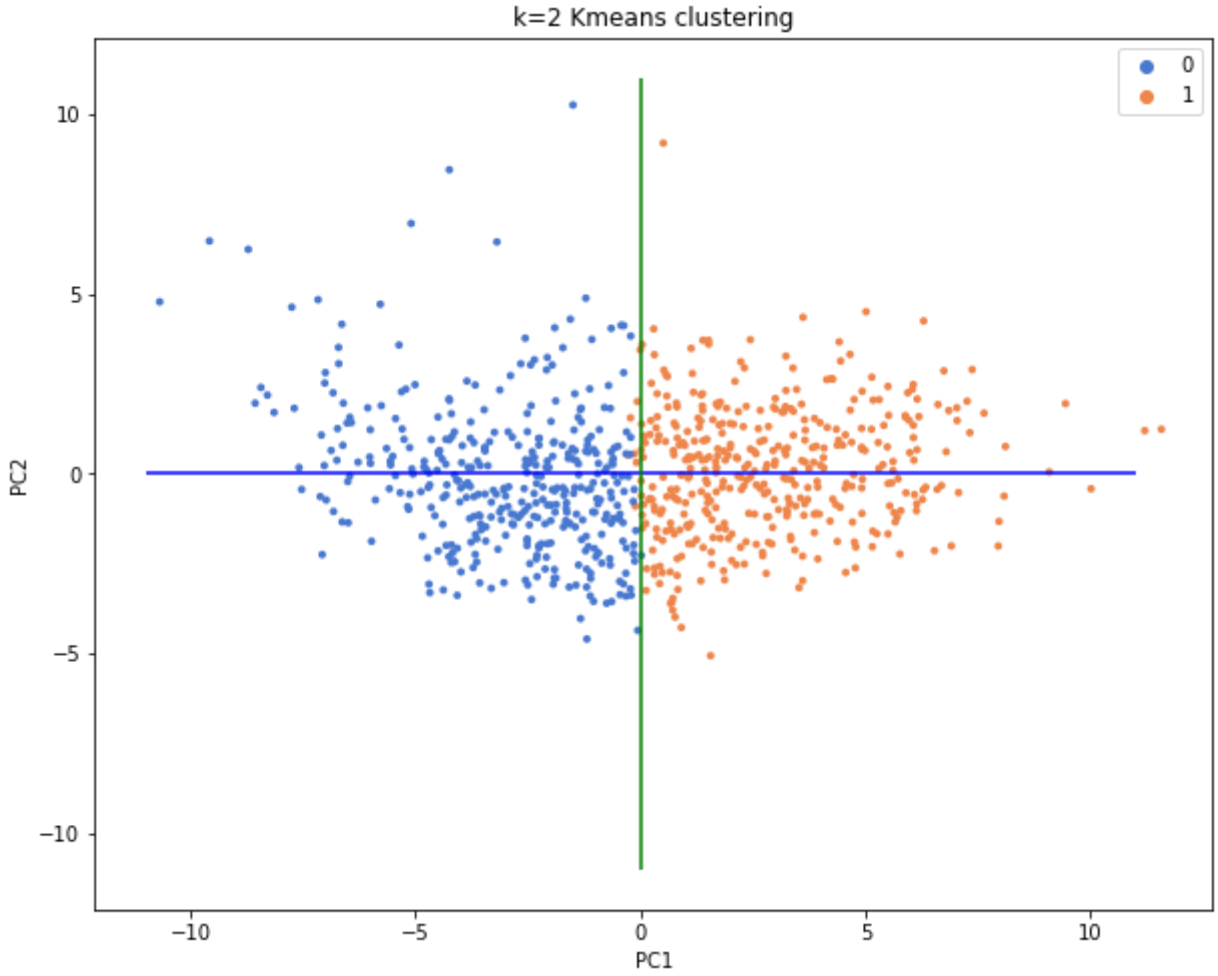
In [305]:
```
xt = TSNE(n_components=2, random_state=5566)
tsne = pd.DataFrame(xt.fit_transform(x), columns=['D1', 'D2'])

plt.figure(figsize=(10, 8))
sns.scatterplot(x = 'D1', y = 'D2', data = tsne, linewidth = 0, s = 15)
plt.hlines(0,-33,33, colors = 'b')
plt.vlines(0,-33,33, colors = 'g')
plt.show();
```



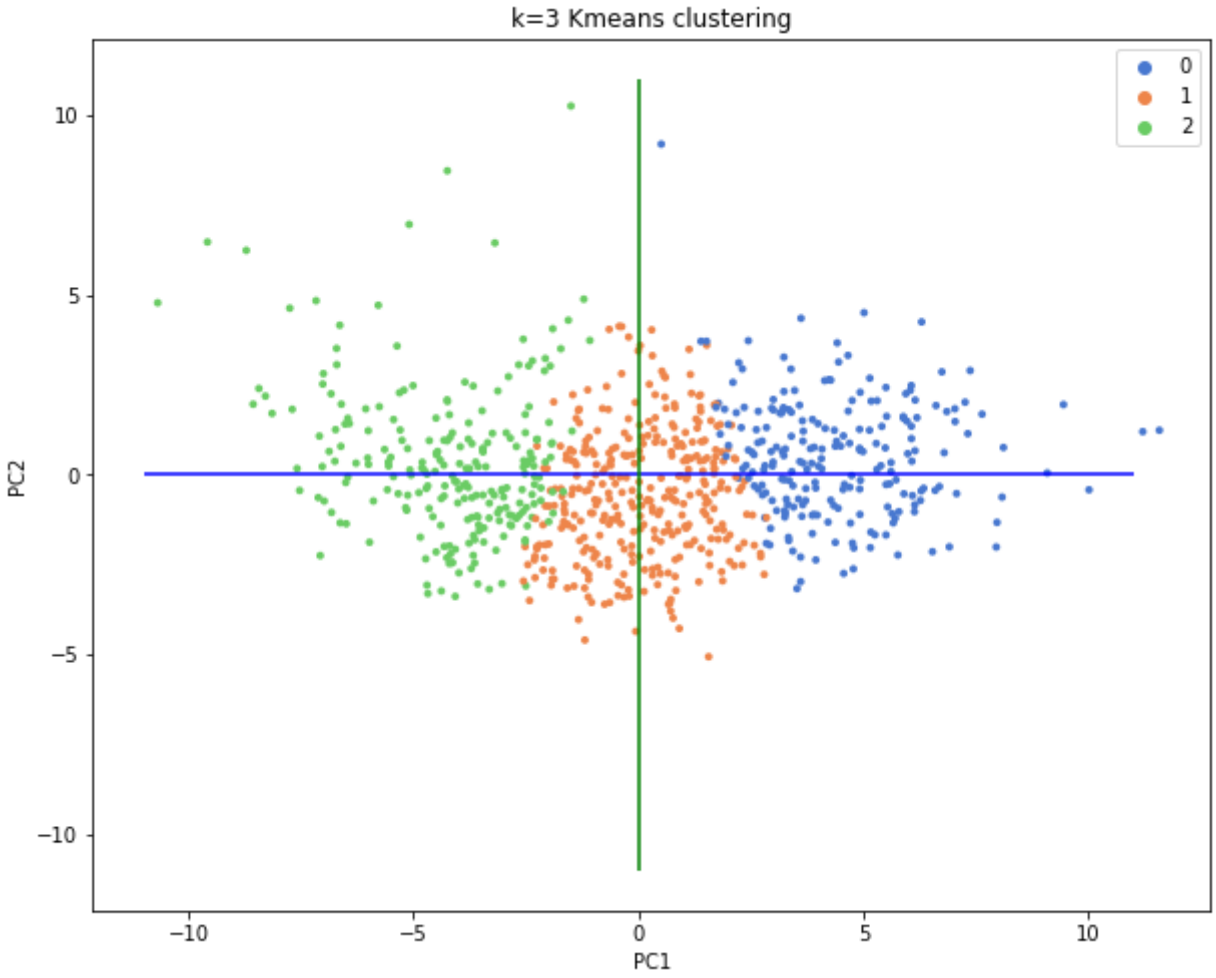
For this dataset, PCA was not able to produce clear-cut clustering (i.e. no visible boundaries), while t-SNE gives observable clustering with the 2 dimensions. This may be due to the fact that we have a very high dimensional dataset. PCA is a linear dimension reduction technique. t-SNE is much more suitable for the job, since it is non-linear and deals with high-dimensional data better, but it also takes longer than PCA

Clustering

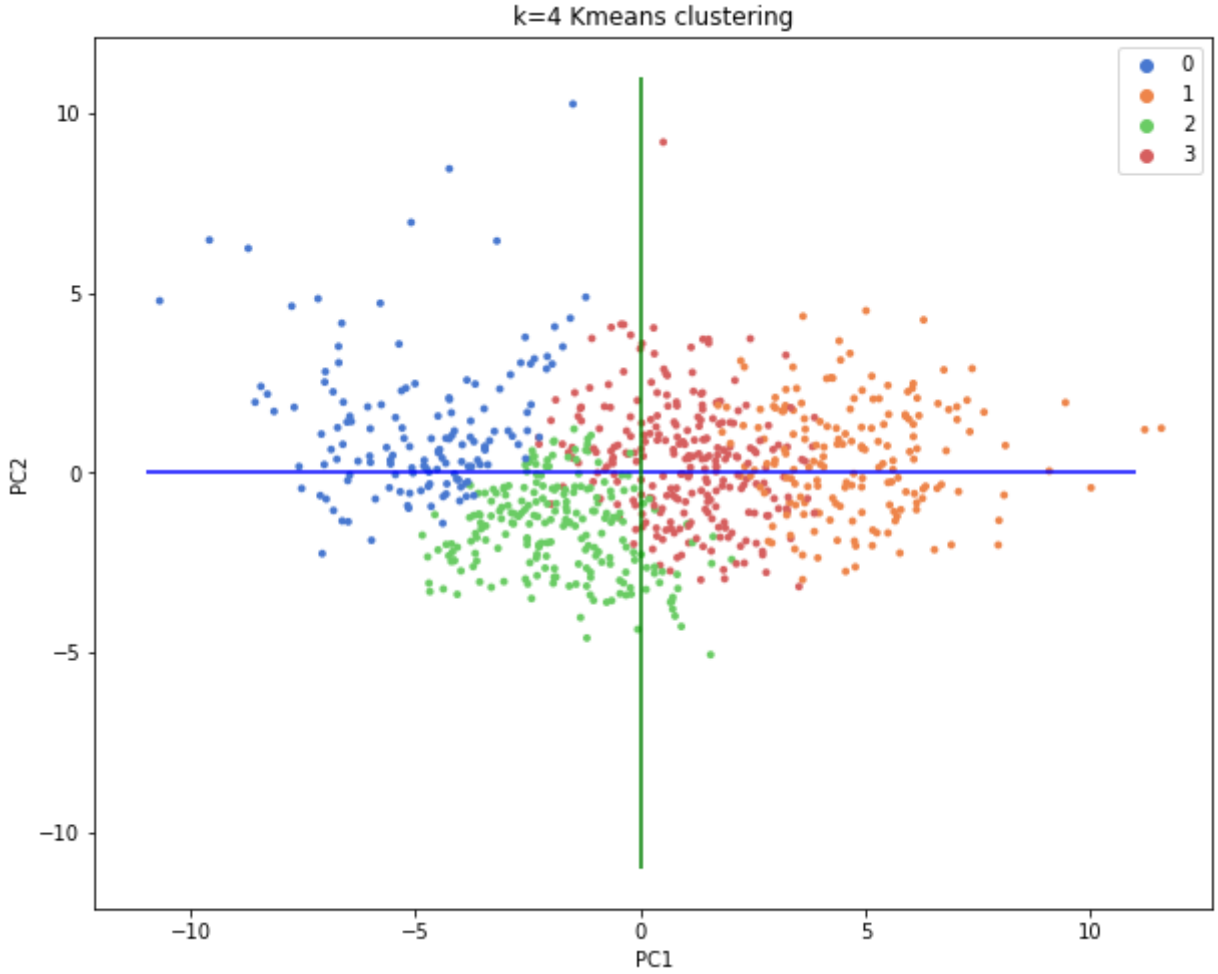
In [316]:
```
# k=2 on the pca-transformed data
kmeans2 = KMeans(n_clusters=2, random_state=5566).fit(x)
plt.figure(figsize=(10, 8))
sns.scatterplot(x = 'PC1', y = 'PC2', data = pca_df, linewidth = 0, s = 15,
plt.title('k=2 Kmeans clustering')
plt.hlines(0,-11,11, colors = 'b')
plt.vlines(0,-11,11, colors = 'g')
plt.show();
```



In [314]:
```
# k=3 on the pca-transformed data
kmeans2 = KMeans(n_clusters=3, random_state=5566).fit(x)
plt.figure(figsize=(10, 8))
sns.scatterplot(x = 'PC1', y = 'PC2', data = pca_df, linewidth = 0, s = 15,
plt.title('k=3 Kmeans clustering')
plt.hlines(0,-11,11, colors = 'b')
plt.vlines(0,-11,11, colors = 'g')
plt.show();
```



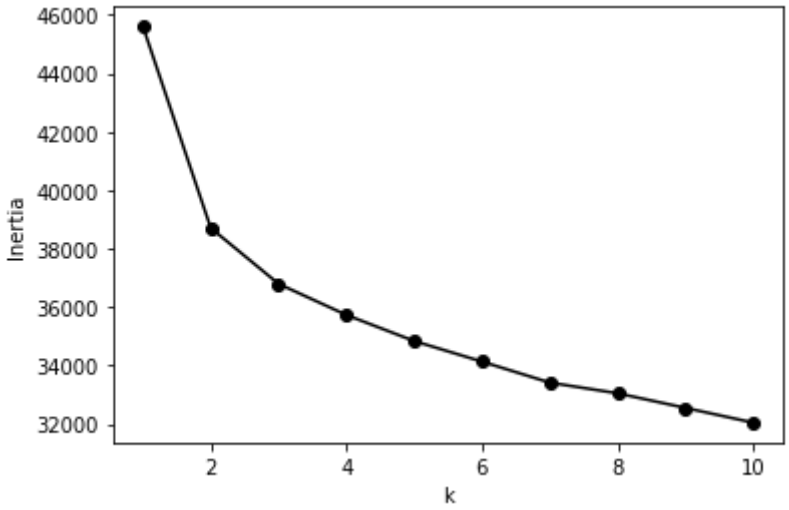
In [317]:
```
# k=4 on the pca-transformed data
kmeans2 = KMeans(n_clusters=4, random_state=5566).fit(x)
plt.figure(figsize=(10, 8))
sns.scatterplot(x = 'PC1', y = 'PC2', data = pca_df, linewidth = 0, s = 15,
plt.title('k=4 Kmeans clustering')
plt.hlines(0,-11,11, colors = 'b')
plt.vlines(0,-11,11, colors = 'g')
plt.show();
```



With k = 2, we find a clearer seperation, with less overlapping than k = 3, k = 4. However, this does not mean k=2 is the best seperation. t-SNE provided us with 4 or 5 clear clusterings. As stated above, this dataset is with very high dimension. It could very well be the case that any 2-D projection we can perform on these clusters may not show clean-cut boundaries. That is to say, if we can visualize in say 3-D, the more overlapping clusterings may actually have clear-cut planes speperating the clusters.

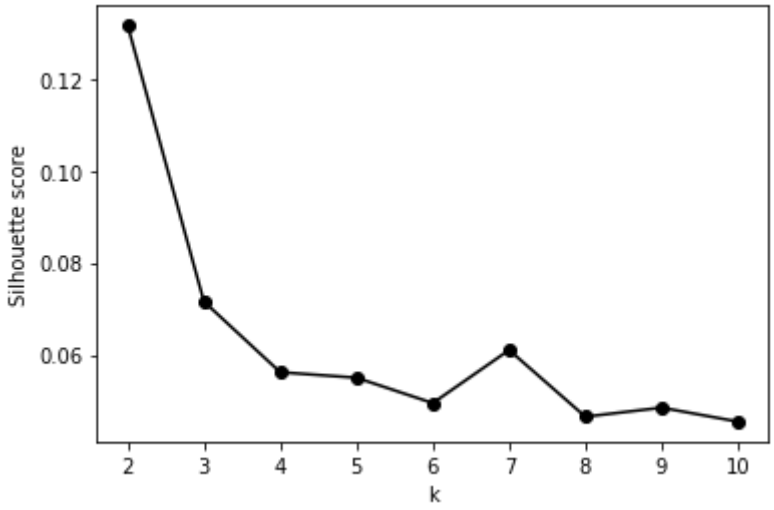
Elbow Method

In [327]:
```
l = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=5566).fit(x)
    l.append(kmeans.inertia_)

plt.plot(list(range(1,11)),l, 'ko')
plt.plot(list(range(1,11)),l, 'k')
plt.ylabel('Inertia')
plt.xlabel('k')
plt.show()
```



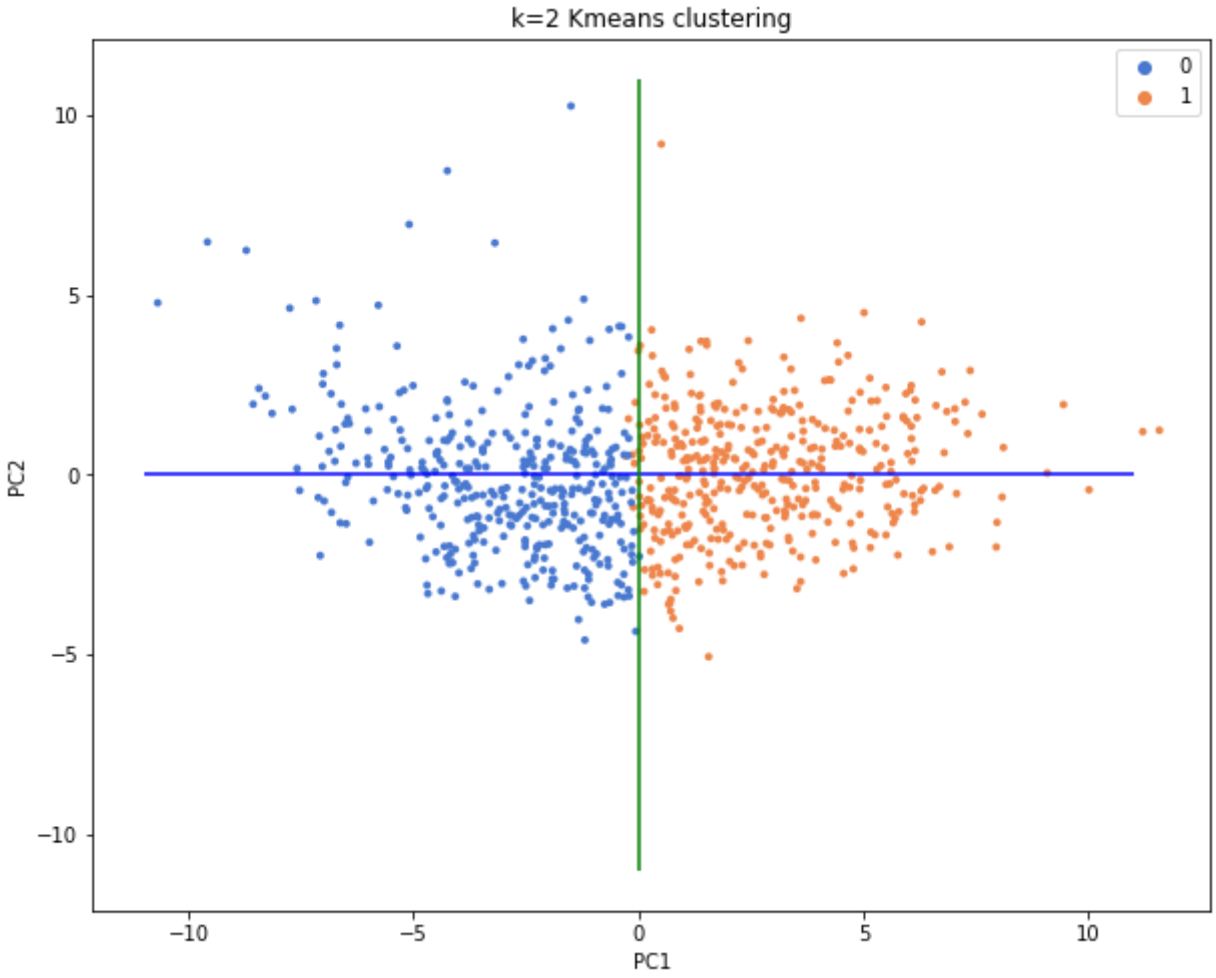
We can see from above, the steepest decrease happened when k = 1 to k=2, meaning that k=2 is the optimal number of clusters, because sum of squared distance within culsters decreased the most between k = 1 and k = 2.

In [331]:
```
l = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=5566)
    c = kmeans.fit_predict(x)
    l.append(silhouette_score(x, c))

plt.plot(list(range(2,11)),l, 'ko')
plt.plot(list(range(2,11)),l, 'k')
plt.ylabel('Silhouette score')
plt.xlabel('k')
plt.show()
```

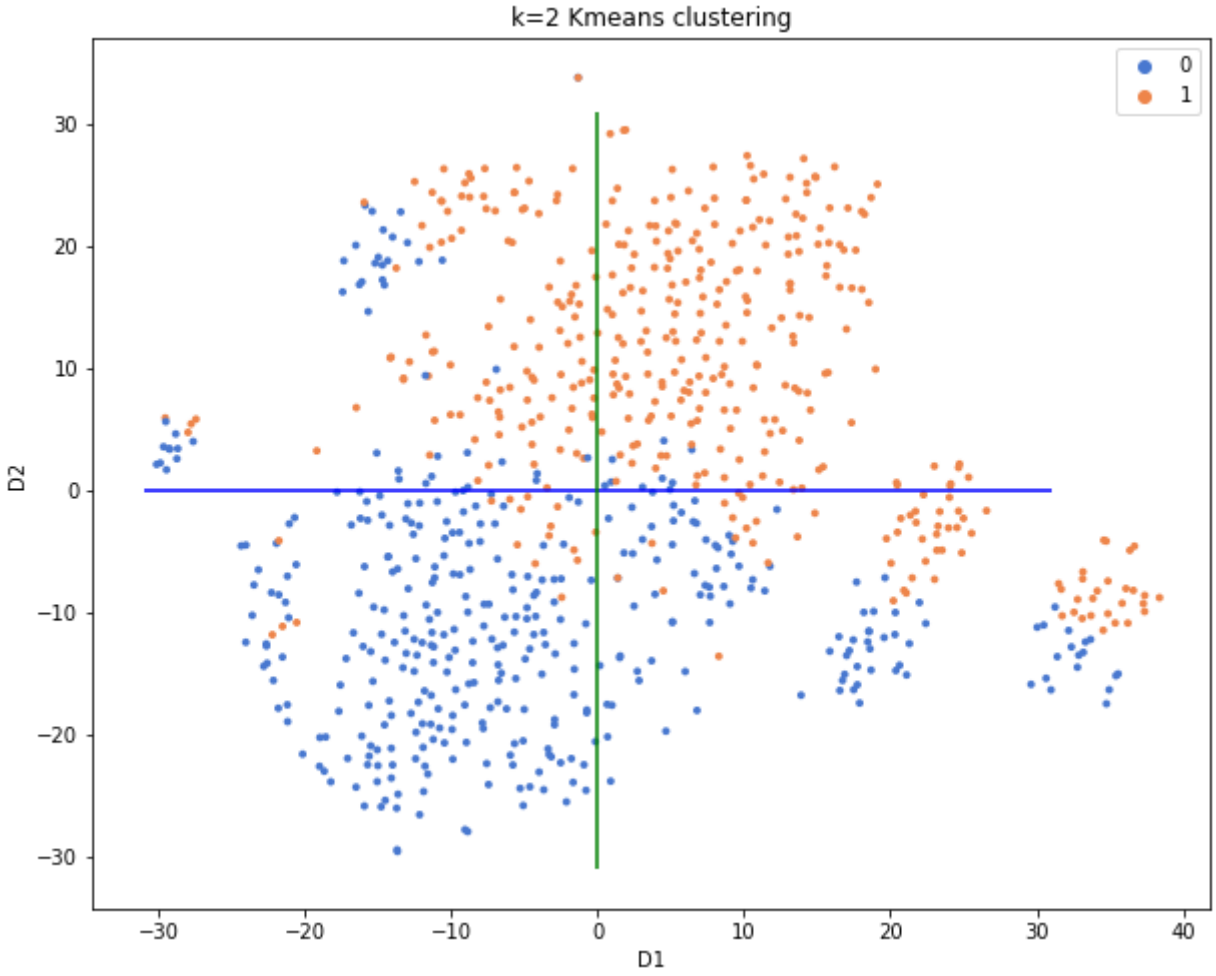


This confirms our findings above: k = 2 is the optimal number of clusters. As shown above k = 2 has the highest score.

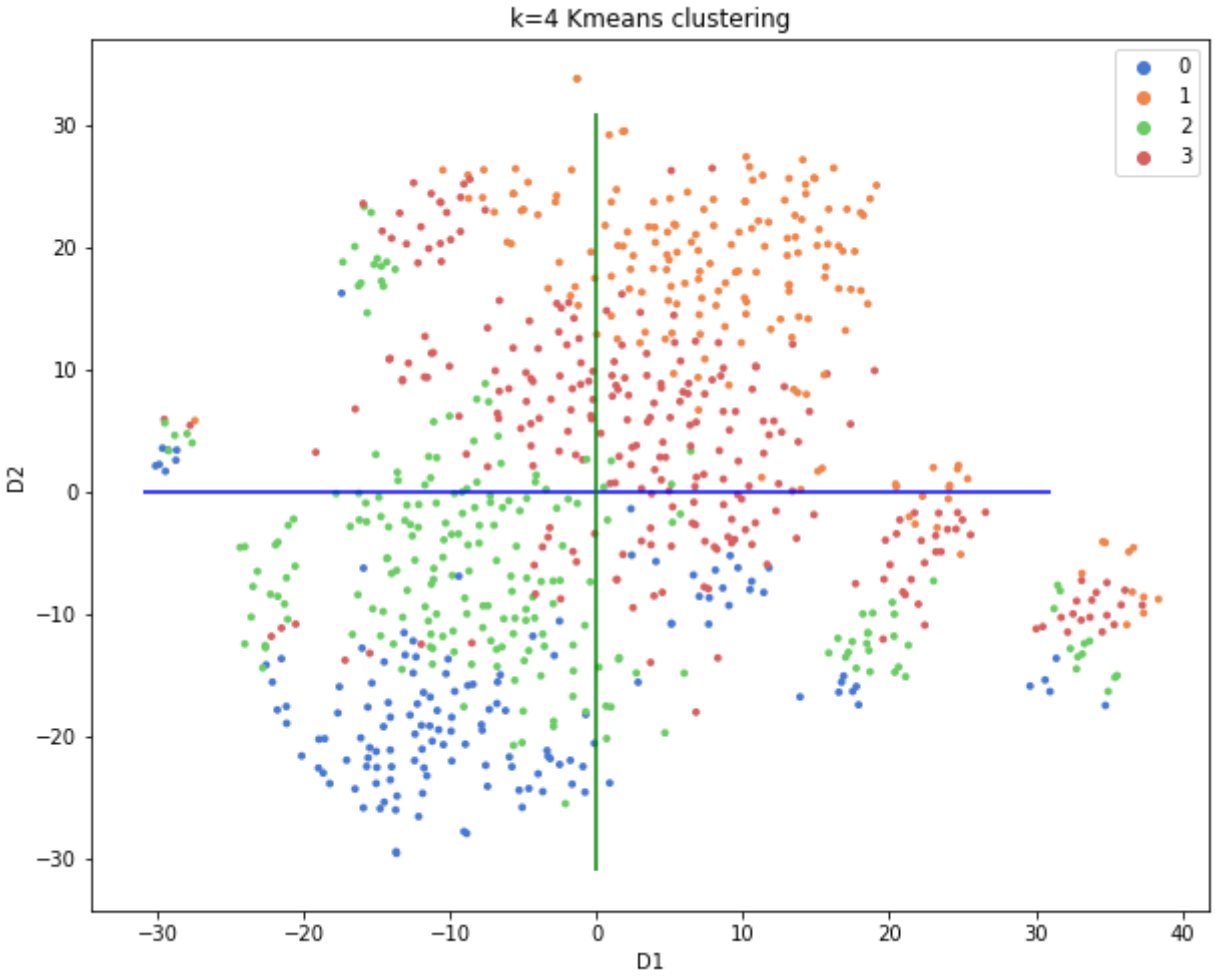
In [332]:
```
# k=2 on the pca-transformed data
kmeans2 = KMeans(n_clusters=2, random_state=5566).fit(x)
plt.figure(figsize=(10, 8))
sns.scatterplot(x = 'PC1', y = 'PC2', data = pca_df, linewidth = 0, s = 15,
plt.title('k=2 Kmeans clustering')
plt.hlines(0,-11,11, colors = 'b')
plt.vlines(0,-11,11, colors = 'g')
plt.show();
```



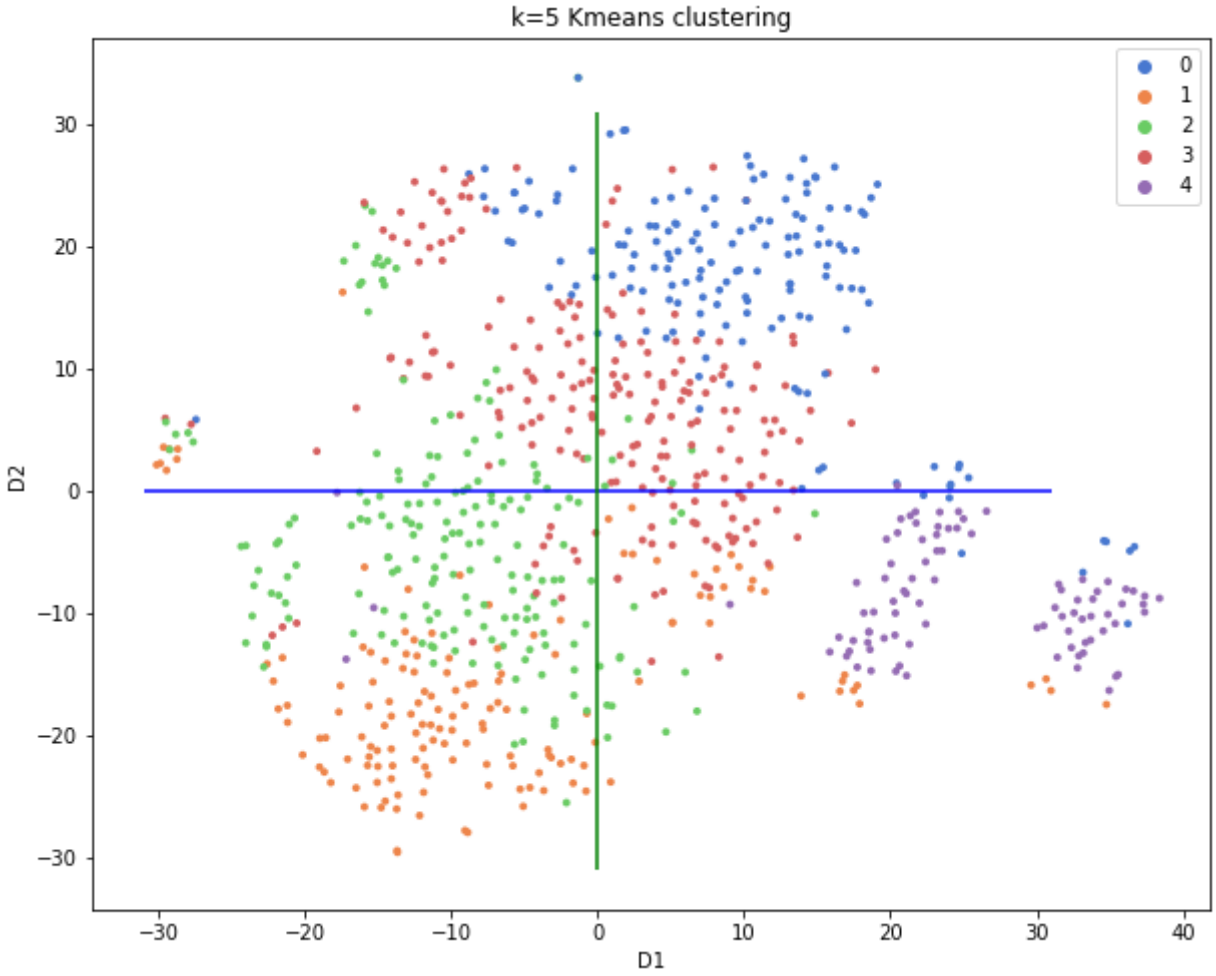
In [335]:
```
plt.figure(figsize=(10, 8))
sns.scatterplot(x = 'D1', y = 'D2', data = tsne, linewidth = 0, s = 15, hue
plt.title('k=2 Kmeans clustering')
plt.hlines(0,-31,31, colors = 'b')
plt.vlines(0,-31,31, colors = 'g')
plt.show();
```



In [338]:
```
plt.figure(figsize=(10, 8))
kmeans2 = KMeans(n_clusters=4, random_state=5566).fit(x)
sns.scatterplot(x = 'D1', y = 'D2', data = tsne, linewidth = 0, s = 15, hue
plt.title('k=4 Kmeans clustering')
plt.hlines(0,-31,31, colors = 'b')
plt.vlines(0,-31,31, colors = 'g')
plt.show();
```



In [339]:
```
plt.figure(figsize=(10, 8))
kmeans2 = KMeans(n_clusters=5, random_state=5566).fit(x)
sns.scatterplot(x = 'D1', y = 'D2', data = tsne, linewidth = 0, s = 15, hue
plt.title('k=5 Kmeans clustering')
plt.hlines(0,-31,31, colors = 'b')
plt.vlines(0,-31,31, colors = 'g')
plt.show();
```



PCA with Kmeans clusters gave us a clearer boudnary between clusters, while tSNE gave us overlapping boundaries. As shown in the last two graphs, even with the visible 4-5 clusters we can observe on a 2-D plane, tSNE did not give a clear seperation. t-SNE differs from PCA by looking at only small, local pairwise distances. PCA looks at large/global scale pairwise distances that maximize distance with origin. Kmeans probably correlates with the latter since the spatial clustering process is similiar. However, this is not to say that t-SNE performed badly becuase this is a really high dimensial dataset.