4.2 Assignment: Clustering Exercise

Keiuntae Smith

DSC 630 Predictive Analysis

29 June 2022

In [1]:
```
#Load the necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.style as style
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.decomposition import PCA
import warnings
from pylab import rcParams
```

In [2]:
```
# Load dataset
df = pd.read_csv('als_data.csv')

# View the dataframe
df.head(5)
```

Out[2]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	...	Sodium_min	Sodium_range	SubjectID	tru
0	1	65	57.0	40.5	38.0	0.066202	-0.965608	30	28.0	22	...	143.0	0.017422	533	
1	2	48	45.0	41.0	39.0	0.010453	-0.921717	37	33.0	21	...	136.0	0.010453	649	
2	3	38	50.0	47.0	45.0	0.008929	-0.914787	24	14.0	10	...	140.0	0.008929	1234	
3	4	63	47.0	44.0	41.0	0.012111	-0.598361	30	29.0	24	...	138.0	0.012469	2492	
4	5	63	47.0	45.5	42.0	0.008292	-0.444039	32	27.5	20	...	138.0	0.008292	2956	

5 rows × 101 columns

In [3]:
```
#check the number of columns and rows
df.shape
```

Out[3]:
```
(2223, 101)
```

In [4]:
```
#check the data types
df.dtypes
```

Out[4]:
```
ID                  int64
Age_mean            int64
Albumin_max       float64
Albumin_median    float64
Albumin_min       float64
                   ...
trunk_min           int64
trunk_range       float64
Urine.Ph_max      float64
Urine.Ph_median   float64
Urine.Ph_min      float64
Length: 101, dtype: object
```

In [5]:
```
df.describe()
```

Out[5]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	...	Sodium_min	Sodium_range
count	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	...	2223.000000	2223.00000
mean	1214.874944	54.550157	47.011134	43.952542	40.766347	0.013779	-0.728274	31.692308	27.104926	19.877193	...	136.755061	0.01500
std	696.678300	11.396546	3.233980	2.654804	3.193087	0.009567	0.622329	5.314228	6.633643	8.583509	...	2.715247	0.00928
min	1.000000	18.000000	37.000000	34.500000	24.000000	0.000000	-4.345238	11.000000	2.500000	0.000000	...	112.000000	0.00000
25%	614.500000	47.000000	45.000000	42.000000	39.000000	0.009042	-1.086310	29.000000	23.000000	14.000000	...	135.000000	0.01058
50%	1213.000000	55.000000	47.000000	44.000000	41.000000	0.012111	-0.620748	33.000000	28.000000	20.000000	...	137.000000	0.01312
75%	1815.500000	63.000000	49.000000	46.000000	43.000000	0.015873	-0.283832	36.000000	32.000000	27.000000	...	138.000000	0.01727
max	2424.000000	81.000000	70.300000	51.100000	49.000000	0.243902	1.207011	40.000000	40.000000	40.000000	...	145.000000	0.14285

8 rows × 101 columns

Remove any data that is not relevant to the patient's ALS condition.

In [6]:
```
#delete columns "ID" and "SubjectID" due to irrelevance
df = df.drop(['ID', 'SubjectID'], axis=1)
```

In [7]:
```
#print new dataframe to ensure deleted columns are removed
df.head()
```

Out[7]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range	...	Sodium_median	Sodium_
0	65	57.0	40.5	38.0	0.066202	-0.965608	30	28.0	22	0.021164	...	145.5	1
1	48	45.0	41.0	39.0	0.010453	-0.921717	37	33.0	21	0.028725	...	138.0	1
2	38	50.0	47.0	45.0	0.008929	-0.914787	24	14.0	10	0.025000	...	143.0	1
3	63	47.0	44.0	41.0	0.012111	-0.598361	30	29.0	24	0.014963	...	139.0	1
4	63	47.0	45.5	42.0	0.008292	-0.444039	32	27.5	20	0.020374	...	140.0	1

5 rows × 99 columns

Apply a standard scaler to the data.

In [8]:
```
#create the standard scaler
scaler = StandardScaler().fit(df)

print(scaler)
```

```
StandardScaler()
```

In [9]:
```
scaler.scale_
```

Out[9]:
```
array([1.13939823e+01, 3.23325265e+00, 2.65420656e+00, 3.19236858e+00,
       9.56466478e-03, 6.22189364e-01, 5.31303232e+00, 6.63215048e+00,
       8.58157841e+00, 1.61522631e-02, 4.48194928e+01, 1.55984156e+01,
       1.12286946e+01, 1.10863795e-01, 3.52807627e+01, 9.59175324e+00,
       7.39363902e+00, 8.40814304e-02, 3.16375986e+00, 2.19832971e+00,
       2.40797718e+00, 1.12772038e-02, 2.31901582e+00, 1.33493594e+00,
       1.35395130e+00, 5.06934886e-03, 8.75625655e+00, 7.24394773e+00,
       8.44176253e+00, 2.72759052e-02, 1.57505042e+01, 1.21582027e+01,
       1.11925191e+01, 3.31029799e-02, 1.80204995e-01, 8.83980021e-02,
       1.72748526e-01, 4.81015277e-04, 2.61637864e+00, 2.33501304e+00,
       3.44507061e+00, 1.16863386e-02, 1.95880211e+01, 1.73564461e+01,
       1.84441609e+01, 3.76571742e-02, 4.80743016e-01, 2.53329616e+00,
       1.24351855e+00, 1.24078475e+00, 6.20373369e-03, 1.97691826e+00,
       2.45954009e+00, 2.60713237e+00, 4.74659426e-03, 1.29302670e+01,
       1.21180144e+01, 1.15561060e+01, 1.12516472e-02, 1.27182613e+01,
       1.16002360e+01, 1.48872626e+01, 3.51263151e-02, 2.24414571e+00,
       2.28026773e+00, 2.12409176e+00, 4.37089222e-03, 1.89707411e+00,
       2.77122694e+00, 3.73923203e+00, 6.12646222e-03, 4.11707772e+02,
       4.19239747e-01, 7.01595267e+01, 5.26146659e+01, 4.93101429e+01,
       1.32511407e+00, 2.43020557e-01, 2.62991009e-01, 2.89749113e-03,
       1.11023677e+01, 8.43585139e+00, 7.76169923e+00, 2.73963423e-02,
       2.98448754e-01, 6.12247720e-01, 1.05519103e+00, 2.47654296e-03,
       2.33263204e+00, 1.78948325e+00, 2.71463607e+00, 9.28061415e-03,
       1.74726678e+00, 2.14559281e+00, 2.35756453e+00, 4.50183410e-03,
       9.31931700e-01, 6.24897974e-01, 4.37123536e-01])
```

In [10]:
```
#apply the scaler to the data
scaler.transform(df)
```

Out[10]:
```
array([[ 0.91713698,  3.08941722, -1.30078105, ..., -0.88037551,
         0.46305355,  1.86853157],
       [-0.57487867, -0.62201561, -1.11240084, ...,  0.1926645 ,
        -1.13720768, -0.41915124],
       [-1.45253494,  0.92441474,  1.14816173, ..., -0.88037551,
        -1.13720768, -0.41915124],
       ...,
       [-0.6626443 , -0.31272954,  0.01788044, ...,  2.33874452,
         0.46305355, -0.41915124],
       [-1.54030057,  0.61512867,  0.01788044, ..., -0.88037551,
        -1.13720768, -0.41915124],
       [-0.57487867,  0.3058426 ,  0.39464087, ..., -1.95341552,
        -1.13720768, -0.41915124]])
```

In [11]:
```
df_scaled = scaler.transform(df)
```

In [12]:
```
print(df_scaled)
```

```
[[ 0.91713698  3.08941722 -1.30078105 ... -0.88037551  0.46305355
   1.86853157]
 [-0.57487867 -0.62201561 -1.11240084 ...  0.1926645  -1.13720768
  -0.41915124]
 [-1.45253494  0.92441474  1.14816173 ... -0.88037551 -1.13720768
  -0.41915124]
 ...
 [-0.6626443  -0.31272954  0.01788044 ...  2.33874452  0.46305355
  -0.41915124]
 [-1.54030057  0.61512867  0.01788044 ... -0.88037551 -1.13720768
  -0.41915124]
 [-0.57487867  0.3058426   0.39464087 ... -1.95341552 -1.13720768
  -0.41915124]]
```

In [13]:
```
# check the mean
print(df_scaled.mean(axis=0))
```

```
[-1.74998717e-16 -4.29905524e-16 -5.36982364e-16 -2.66093940e-16
  7.67117663e-17 -4.15522068e-17 -2.07761034e-16 -1.22259378e-16
  0.00000000e+00  1.43834562e-17 -8.86979798e-17  8.23053326e-17
  2.06961953e-16  4.15522068e-17 -7.43145236e-17  3.03650742e-17
  4.54677032e-16 -5.59356630e-18  3.89951479e-16 -6.76821522e-16
  1.24656620e-16 -1.67806989e-16 -3.93147803e-16  4.18718391e-16
  7.43145236e-17 -3.51595596e-17  5.36982364e-16  1.66208827e-16
  1.03880517e-16  1.51825371e-16  6.07301484e-16 -1.09474083e-15
 -9.26933843e-17 -1.74199636e-16 -4.19917013e-16 -1.52464636e-15
  1.28731933e-15 -7.75108472e-17  4.54836848e-15  1.46311713e-15
  1.44793459e-15 -5.43375012e-17 -1.18263973e-16 -4.25111038e-16
  1.19862135e-16  6.39264720e-18  2.20945869e-16  6.71227956e-17
  1.51825371e-16 -1.40638238e-16  6.07301484e-17  7.35154427e-17
  1.55021694e-16 -7.67117663e-17 -7.03191191e-17  3.43604787e-17
 -3.52794217e-16  4.18718391e-16  3.99540450e-17  4.95430158e-16
 -4.63466922e-17  7.28761780e-16  8.94970607e-17  5.35384203e-17
  1.61014801e-16  6.71227956e-17  5.83329057e-17 -1.94975739e-16
 -2.28537137e-16  9.42915461e-17  7.99080899e-18 -1.08675002e-16
 -2.39724270e-17  2.20546328e-16  9.82869506e-17  1.99770225e-17
 -1.91779416e-17  2.02646916e-15  2.46596366e-15  6.39264720e-18
 -2.71687506e-16 -7.04789353e-16 -3.73969861e-16 -1.91779416e-16
 -1.24656620e-16 -3.45202949e-16  1.40638238e-16 -2.59701292e-16
 -5.01383310e-15 -1.00060910e-14 -4.25350763e-15 -2.21744950e-16
 -1.15067650e-16  1.24257080e-16  6.87209573e-17  1.24656620e-16
  1.01643090e-15  1.15067650e-16 -1.93377578e-16]
```

Create a plot of the cluster silhouette score versus the number of clusters in a K-means cluster.

In [14]:
```
# Creating an empty dataframe for number of clusters vs silhouette score.
silhouette_df_df = pd.DataFrame(columns=['Clusters', 'Silhouette Score'])
```

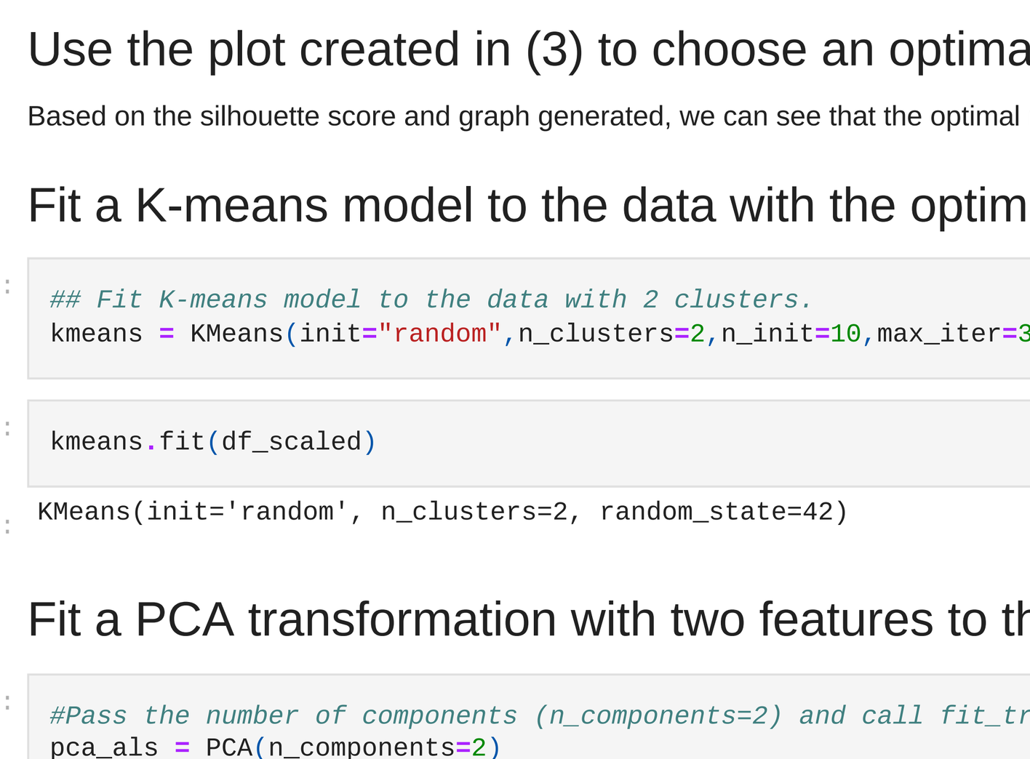
In [15]:
```
# Calculating silhouette score for n clusters and adding to dataframe.
for k in range(2, 20):
    kmeans_model = KMeans(n_clusters=k, random_state=10).fit(df_scaled)
    labels = kmeans_model.labels_
    silhouette_df = silhouette_score(df_scaled, labels, metric='euclidean')
    silhouette_df_df = silhouette_df_df.append({'Clusters' : k, 'Silhouette Score' : silhouette_df}, ignore_index=True)
```

In [16]:
```
#display the silhouette score dataframe
silhouette_df_df
```

Out[16]:

	Clusters	Silhouette Score
0	2.0	0.078877
1	3.0	0.068771
2	4.0	0.070831
3	5.0	0.075632
4	6.0	0.061511
5	7.0	0.049698
6	8.0	0.048834
7	9.0	0.047834
8	10.0	0.048486
9	11.0	0.047593
10	12.0	0.043922
11	13.0	0.039380
12	14.0	0.041426
13	15.0	0.039955
14	16.0	0.042995
15	17.0	0.037562
16	18.0	0.038471
17	19.0	0.035934

In [17]:
```
# Create a scatter plot of Clusters vs Silhouette Score
fig=plt.figure()
ax=fig.add_axes([0, 0, 1, 1])
ax.scatter(silhouette_df_df['Clusters'], silhouette_df_df['Silhouette Score'], color='green')
ax.set_xlabel('Clusters')
ax.set_ylabel('Silhouette Score')
ax.set_title('K-means Clusters vs Silhouette Score')
plt.show()
```



Use the plot created in (3) to choose an optimal number of clusters for K-means. Justify your choice.

Based on the silhouette score and graph generated, we can see that the optimal number of clusters for the data is 2.

Fit a K-means model to the data with the optimal number of clusters chosen in part (4).

In [18]:
```
## Fit K-means model to the data with 2 clusters.
kmeans = KMeans(init="random",n_clusters=2,n_init=10,max_iter=300,random_state=42)
```

In [19]:
```
kmeans.fit(df_scaled)
```

Out[19]:
```
KMeans(init='random', n_clusters=2, random_state=42)
```

Fit a PCA transformation with two features to the scaled data.

In [20]:
```
#Pass the number of components (n_components=2) and call fit_transform
pca_als = PCA(n_components=2)
pca_als.fit(df_scaled)
```

Out[20]:
```
PCA(n_components=2)
```

In [21]:
```
# transform the PCA
pca_als.transform(df_scaled)
```

Out[21]:
```
array([[-1.4267426 , -2.31901808],
       [-1.44024117, -4.87149062],
       [ 1.61784358, -0.42861302],
       ...,
       [-0.43290899,  4.24568346],
       [-0.33079461,  3.31754158],
       [ 1.46798598,  0.58206882]])
```

In [22]:
```
# Calculate resulting components scores for the elements in the dataset (X)
scores_pca_als=pca_als.transform(df_scaled)
```

In [23]:
```
# Create a new dataframe with the original features and add the PCA scores
pca_df = pd.concat([df.reset_index(drop=True), pd.DataFrame(scores_pca_als)], axis=1)
pca_df.columns.values[-2: ] = ['PC_1', 'PC_2']


pca_df['Segment K-means PCA'] = kmeans.labels_
```

In [24]:
```
# Add the names of the segments to the labels
pca_df['Segment'] = pca_df['Segment K-means PCA'].map({0:'first',1:'second', 2:'third', 3:'fourth'})
```

In [25]:
```
# display the head of the dataframe
pca_df.head()
```

Out[25]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range	...	trunk_median	trunk_min
0	65	57.0	40.5	38.0	0.066202	-0.965608	30	28.0	22	0.021164	...	7.0	7
1	48	45.0	41.0	39.0	0.010453	-0.921717	37	33.0	21	0.028725	...	7.0	5
2	38	50.0	47.0	45.0	0.008929	-0.914787	24	14.0	10	0.025000	...	0.0	0
3	63	47.0	44.0	41.0	0.012111	-0.598361	30	29.0	24	0.014963	...	5.0	3
4	63	47.0	45.5	42.0	0.008292	-0.444039	32	27.5	20	0.020374	...	4.0	1

5 rows × 103 columns

In [26]:
```
# display the tail of the dataframe
pca_df.tail()
```

Out[26]:

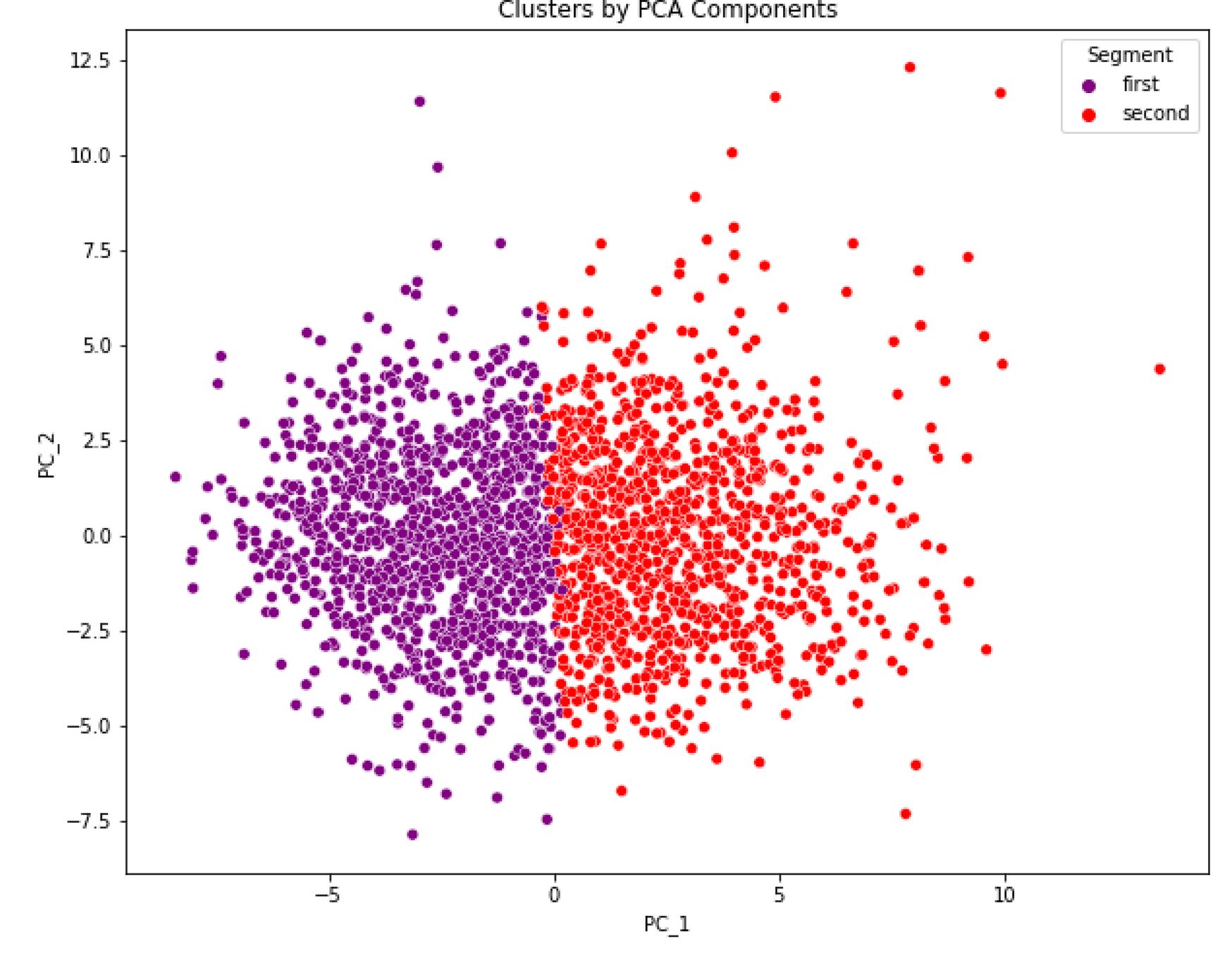
	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range	...	trunk_median	trunk_r
2218	33	50.0	49.0	45.0	0.008772	-0.239501	35	32.5	30	0.009107	...	5.0	
2219	61	47.0	45.0	42.0	0.009074	-0.388711	31	26.0	17	0.025408	...	4.0	
2220	47	46.0	44.0	41.0	0.012111	-0.108631	26	23.0	20	0.010949	...	4.0	
2221	37	49.0	44.0	39.0	0.017857	-0.855880	34	29.5	21	0.023214	...	4.5	
2222	48	48.0	45.0	40.0	0.018476	-2.050562	37	34.0	11	0.059908	...	8.0	

5 rows × 103 columns

Make a scatterplot of the PCA transformed data coloring each point by its cluster value.

In [27]:
```
# Plot scatterplot of PCA transformed data identifying each point by its cluster value
x_axis = pca_df['PC_2']
y_axis = pca_df['PC_1']
plt.figure(figsize =(10,8))
sns.scatterplot(y_axis,x_axis, hue=pca_df['Segment'], palette = ['purple', 'red'])
plt.title('Clusters by PCA Components')
plt.show()
```

```
/Users/keiuntaesmith/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From ve
rsion 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Summarize your results and make a conclusion.

The Silhouette Coefficient of 0.078877 signifies that the optimal number of clusters for the ALS data set is 2. There are two clear separated featured groups that are ascertained as a consequence of fitting the K-means model and PCA transformation. The properties of the clusters are condensed and clearly distinctive as visulaized in the scatter plot above. The left cluster appears to be more denser than the right cluster, which has more outliers.