Cluter Analysis

In [21]:
```
## import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

## library versions
print('pandas version:', pd.__version__)
print('numpy version:', np.__version__)
print('seaborn version:', sns.__version__)
```

```
pandas version: 1.4.2
numpy version: 1.21.5
seaborn version: 0.11.2
```

In [42]:
```
## load dataset
df = pd.read_csv("als_data.csv")
df.head()
```

Out[42]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_s
0	1	65	57.0	40.5	38.0	0.066202	-0.965
1	2	48	45.0	41.0	39.0	0.010453	-0.921
2	3	38	50.0	47.0	45.0	0.008929	-0.914
3	4	63	47.0	44.0	41.0	0.012111	-0.598
4	5	63	47.0	45.5	42.0	0.008292	-0.444

5 rows × 101 columns

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

In [43]:
```
## drop ID columns
df.drop('ID', axis=1, inplace=True)
df.drop('SubjectID', axis=1, inplace=True)
```

In [44]:
```
df.head()
```

Out[44]:

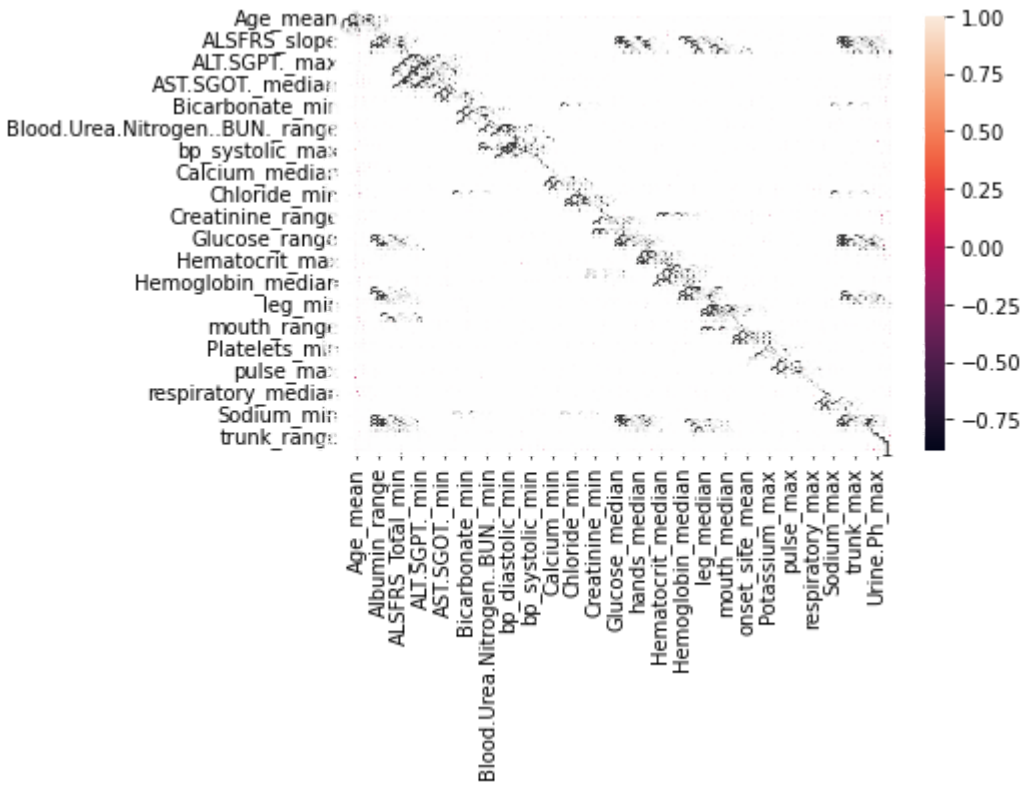
	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope
0	65	57.0	40.5	38.0	0.066202	-0.965608
1	48	45.0	41.0	39.0	0.010453	-0.921717
2	38	50.0	47.0	45.0	0.008929	-0.914787
3	63	47.0	44.0	41.0	0.012111	-0.598361
4	63	47.0	45.5	42.0	0.008292	-0.444039

5 rows × 99 columns

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2223 entries, 0 to 2222
Data columns (total 99 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Age_mean                        2223 non-null   int64
 1   Albumin_max                     2223 non-null   float64
 2   Albumin_median                  2223 non-null   float64
 3   Albumin_min                     2223 non-null   float64
 4   Albumin_range                   2223 non-null   float64
 5   ALSFRS_slope                    2223 non-null   float64
 6   ALSFRS_Total_max                2223 non-null   int64
 7   ALSFRS_Total_median             2223 non-null   float64
 8   ALSFRS_Total_min                2223 non-null   int64
 9   ALSFRS_Total_range              2223 non-null   float64
 10  ALT.SGPT._max                   2223 non-null   float64
 11  ALT.SGPT._median                2223 non-null   float64
 12  ALT.SGPT._min                   2223 non-null   float64
 13  ALT.SGPT._range                 2223 non-null   float64
```

In [9]:
```
## correlation matrix
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True)
plt.show()
```



In [22]:
```
df.shape
```

Out[22]: (2223, 99)

In [24]:
```
df.describe()
```

Out[24]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS
count	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.0
mean	54.550157	47.011134	43.952542	40.766347	0.013779	-0.7
std	11.396546	3.233980	2.654804	3.193087	0.009567	0.6
min	18.000000	37.000000	34.500000	24.000000	0.000000	-4.3
25%	47.000000	45.000000	42.000000	39.000000	0.009042	-1.0
50%	55.000000	47.000000	44.000000	41.000000	0.012111	-0.6
75%	63.000000	49.000000	46.000000	43.000000	0.015873	-0.2
max	81.000000	70.300000	51.100000	49.000000	0.243902	1.2

8 rows × 99 columns

In [25]:
```
len(df['Age_mean'].unique())
```

Out[25]: 61

In [27]:
```
len(df['Albumin_max'].unique())
```

Out[27]: 104

In [28]:
```
len(df['Albumin_median'].unique())
```

Out[28]: 107

In [29]:
```
len(df['Albumin_min'].unique())
```

Out[29]: 106

In [30]:
```
len(df['Albumin_range'].unique())
```

Out[30]: 1051

In [31]:
```
len(df['Sodium_range'].unique())
```

Out[31]: 996

In [33]:
```
len(df['Chloride_range'].unique())
```

Out[33]: 992

In [32]:
```
len(df['trunk_range'].unique())
```

Out[32]: 791

Observation

The range fields have significant more variance than the other fields. While they're not overwhelmingly unique to each participant, they have a more unique presence than the mean/median/min/max values.

In [45]:

```
df.drop('Albumin_range', axis=1, inplace=True)
df.drop('ALSFRS_Total_range', axis=1, inplace=True)
df.drop('ALT.SGPT._range', axis=1, inplace=True)
df.drop('AST.SGOT._range', axis=1, inplace=True)
df.drop('Bicarbonate_range', axis=1, inplace=True)
df.drop('Blood.Urea.Nitrogen..BUN._range', axis=1, inplace=True)
df.drop('bp_diastolic_range', axis=1, inplace=True)
df.drop('bp_systolic_range', axis=1, inplace=True)
df.drop('Calcium_range', axis=1, inplace=True)
df.drop('Chloride_range', axis=1, inplace=True)
df.drop('Creatinine_range', axis=1, inplace=True)
df.drop('Glucose_range', axis=1, inplace=True)
df.drop('hands_range', axis=1, inplace=True)
df.drop('Hematocrit_range', axis=1, inplace=True)
df.drop('leg_range', axis=1, inplace=True)
df.drop('mouth_range', axis=1, inplace=True)
df.drop('Potassium_range', axis=1, inplace=True)
df.drop('pulse_range', axis=1, inplace=True)
df.drop('respiratory_range', axis=1, inplace=True)
df.drop('Sodium_range', axis=1, inplace=True)
df.drop('trunk_range', axis=1, inplace=True)

df.head()
```

Out[45]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	ALSFRS_slope	ALSFRS_Total_
0	65	57.0	40.5	38.0	-0.965608	
1	48	45.0	41.0	39.0	-0.921717	
2	38	50.0	47.0	45.0	-0.914787	
3	63	47.0	44.0	41.0	-0.598361	
4	63	47.0	45.5	42.0	-0.444039	

5 rows × 78 columns

2. Apply a standard scaler to the data.

In [35]:
```
## import library
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

standardized = scaler.fit_transform(df)
print("Standardized Features:\n", standardized[:78])
```

```
Standardized Features:
 [[ 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]
 ...
 [ 1.18043386 -0.62201561 -1.11240084 ...  0.1926645   0.46305355
  -0.41915124]
 [ 0.39054322 -0.00344347 -0.35887998 ...  0.1926645  -1.13720768
  -0.41915124]
 [-0.39934742 -0.00344347  0.20626066 ... -0.88037551 -1.13720768
  -0.41915124]]
```

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

In [38]:
```
## import additional libraries
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

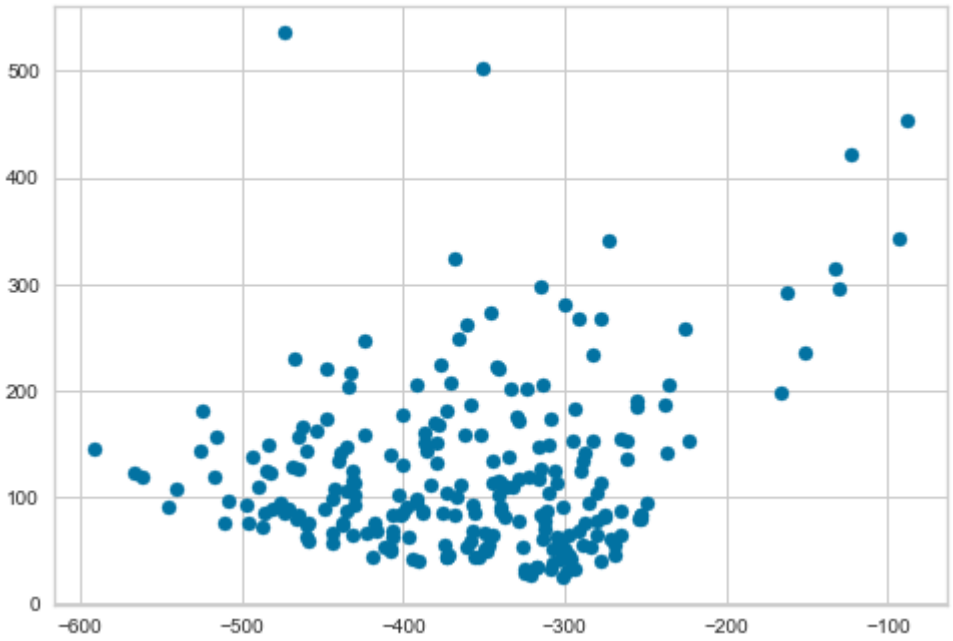
In [47]:
```
pca = PCA(2)

#Transform the data
df2 = pca.fit_transform(df)

df2.shape
```

Out[47]: (2223, 2)

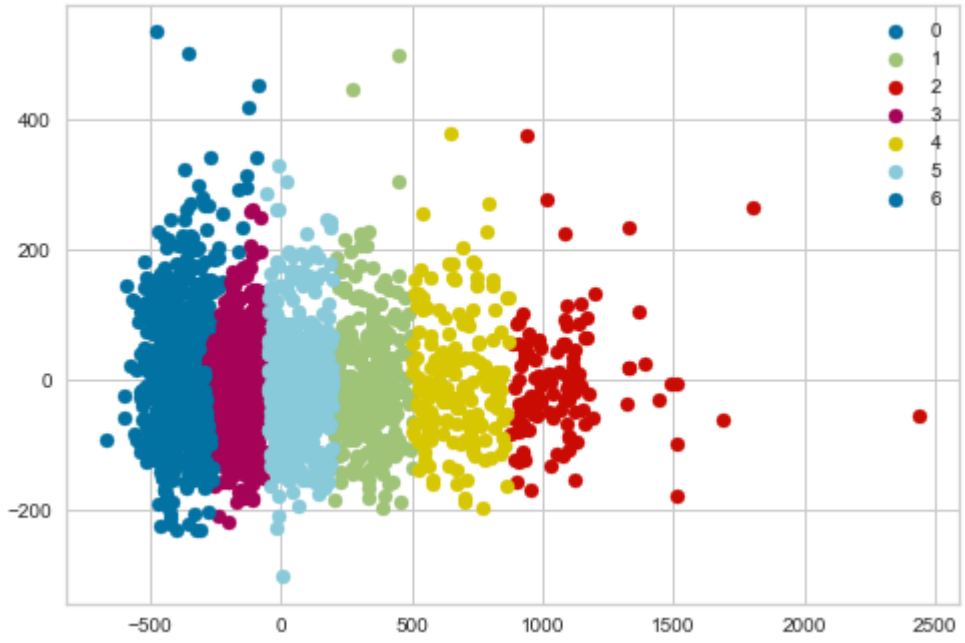
In [70]:
```
kmeans = KMeans(n_clusters= 7)
label = kmeans.fit_predict(df)

print(label)
```

```
[1 6 1 ... 2 3 0]
```

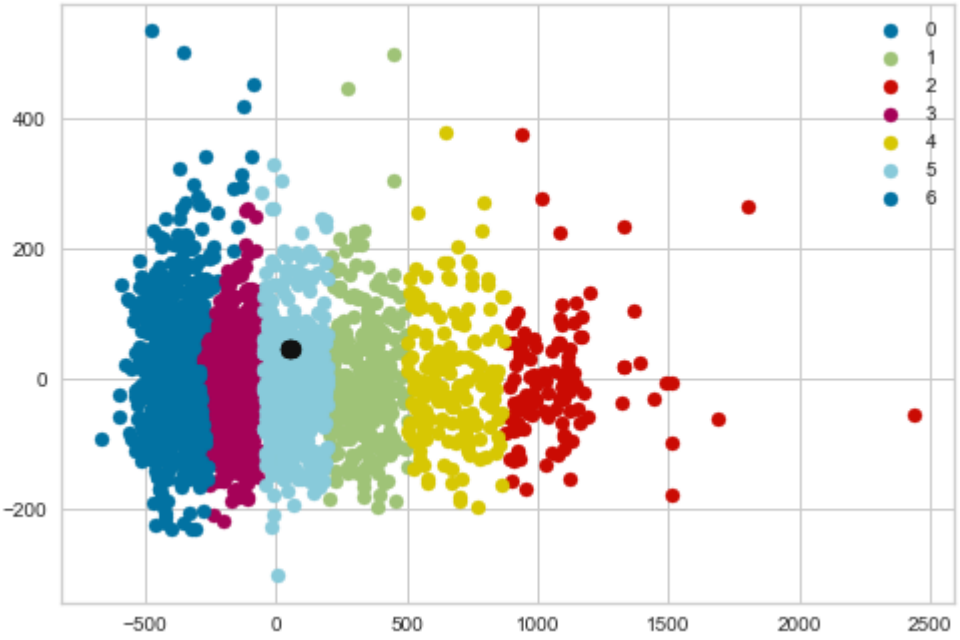
In [71]:
```
filtered_label0 = df2[label == 0]

## plot
plt.scatter(filtered_label0[:,0] , filtered_label0[:,1])
plt.show()
```



In [72]:
```
## all plot
u_labels = np.unique(label)
for i in u_labels:
    plt.scatter(df2[label == i , 0] , df2[label == i , 1] , label = i)
plt.legend()
plt.show()
```



In [73]:
```
## plot centroids
centroids = kmeans.cluster_centers_
u_labels = np.unique(label)
for i in u_labels:
    plt.scatter(df2[label == i , 0] , df2[label == i , 1] , label = i)
plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```



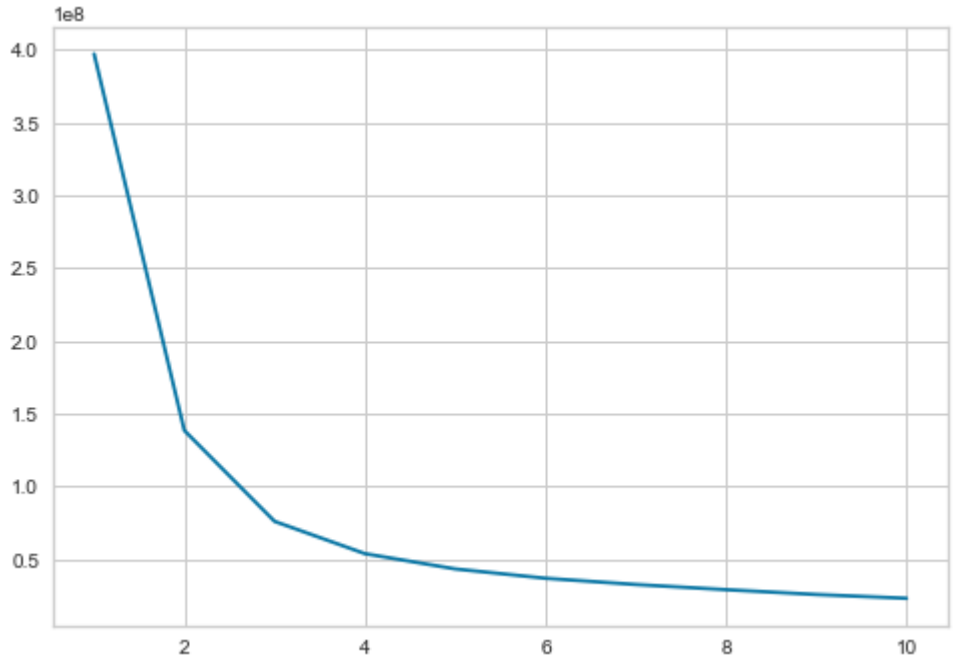
In [83]:
```
print(f'Silhouette Score: {silhouette_score(df2, label)}')
```

```
Silhouette Score: 0.3627250977809936
```

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

In [89]:

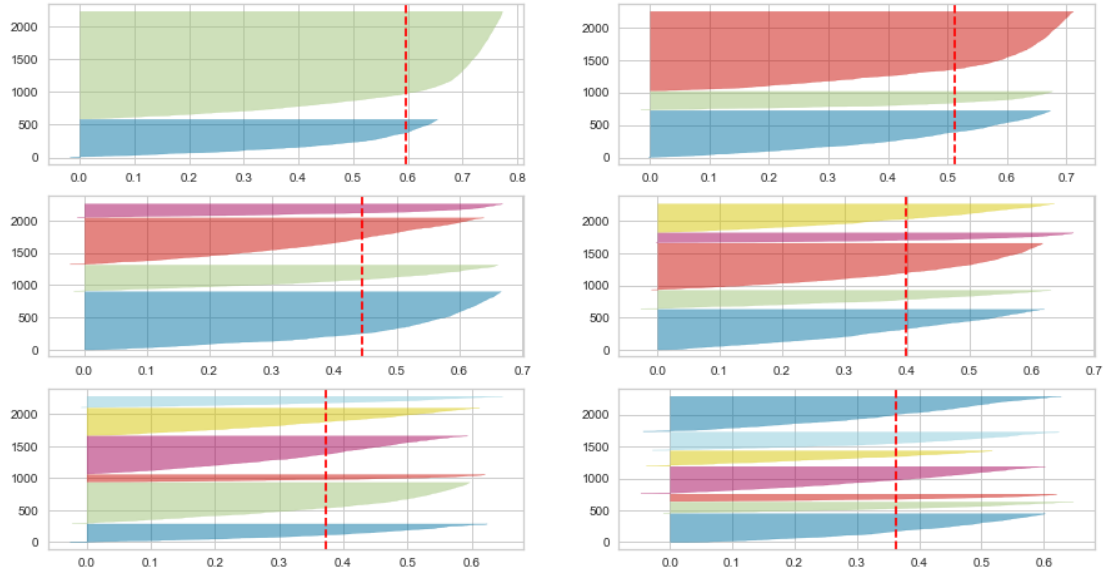
```
wcss = []

for i in range(1, 11):
    clustering = KMeans(n_clusters=i, init='k-means++', random_state=4
    clustering.fit(df2)
    wcss.append(clustering.inertia_)

ks = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sns.lineplot(x = ks, y = wcss);
```



Based on the above elbow plot, our data appears to cluster at 4.

In [92]:

```
## import additional libraries

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from yellowbrick.cluster import SilhouetteVisualizer

fig, ax = plt.subplots(3, 2, figsize=(15,8))
for i in [2, 3, 4, 5, 6, 7]:
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=10
    q, mod = divmod(i, 2)
    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[
    visualizer.fit(df2)
```

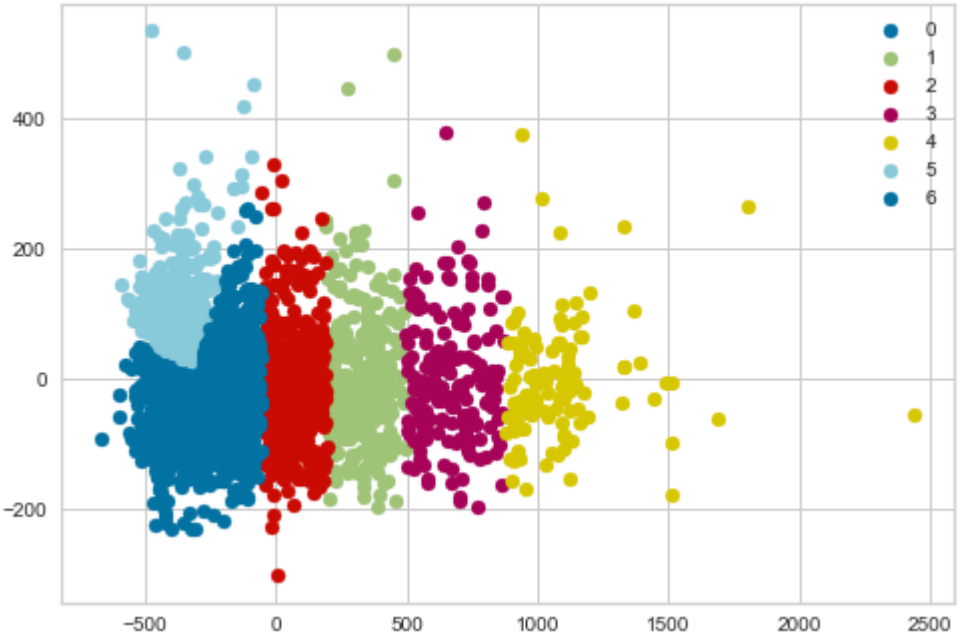


Visually, the sillhouette scores do not appear consistent. While the elbow score indicates 3 would be optimal, the sillhouette visualization of 3/4 clusters has a high variability. On the contrary, the size distortion decreases as the clusters increases. While 7 is not wholly uniform, it is the closest of the options. I feel as thought the sillhouette offers a more complete guidance on the number of clusters.

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

** completed in earlier step **

In [94]:
```
## all plot
u_labels = np.unique(label)
for i in u_labels:
    plt.scatter(df2[label == i , 0] , df2[label == i , 1] , label = i)
plt.legend()
plt.show()
```



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

In [101]:
```
## import additional library
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df2)
```

In [103]:
```
principal_Df = pd.DataFrame(data = principalComponents, columns = ['pr
```

In [104]:
```
principal_Df.head()
```

Out[104]:

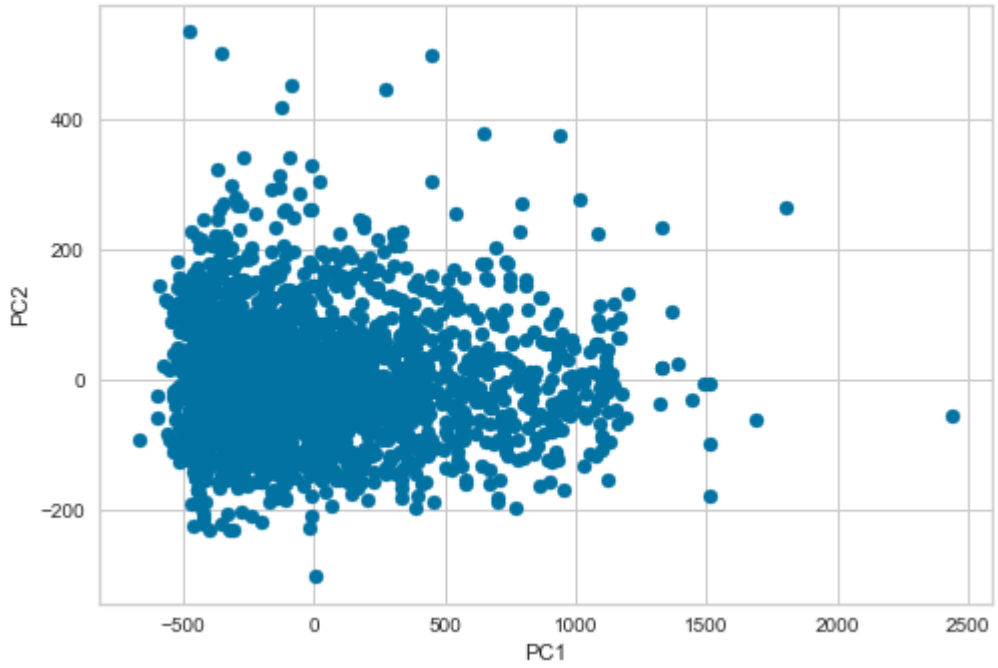
	principal component 1	principal component 2
0	342.448354	-137.816705
1	-341.982581	21.963924
2	498.926605	-62.593579
3	-317.547109	-19.159624
4	1083.984978	92.445683

In [105]:
```
print('Explained variation per principal component: {}'.format(pca.exp
```

```
Explained variation per principal component: [0.94899842 0.05100158]
```

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

In [117]:
```
plt.scatter(principalComponents[:,0], principalComponents[:,1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



8. Summarize your results and make a conclusion.

This was a very challenging assignment. Despite working in descriptive analytics regularly, doing this work without a target was difficult, especially when bringing in the PCA. What I found most interesting was the inconsistency between the elbow method and the sillhouette for suggested number of clusters. While I did drop several columns, it makes me wonder if I was meant to drop additional columns and how that would best be done without a target. Referencing the Applied Predictive Analytics text on variable selection, all of the comparison types referenced input versus a target, which didn't suit this type of analysis. Historically, I've understood that if the number of unique values aligns closely to the total number of rows, the field is less likely to indicate anything, as it's too unique to the row.