Clustering Exercise Using ALS Patient Data

KJ MoChroi
DSC 630 Winter 2022
Bellevue University

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

First I will import the data, look at it, and remove any unnecesary columns. This method is based on (Applied Predictive Analytics) page 119.

In [1]:
```
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.decomposition import PCA
```

In [2]:
```
als_df = pd.read_csv("als_data.csv")
```

In [3]:
```
als_df.head()
```

Out[3]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_
0	1	65	57.0	40.5	38.0	0.066202	-0.965608	
1	2	48	45.0	41.0	39.0	0.010453	-0.921717	
2	3	38	50.0	47.0	45.0	0.008929	-0.914787	
3	4	63	47.0	44.0	41.0	0.012111	-0.598361	
4	5	63	47.0	45.5	42.0	0.008292	-0.444039	

5 rows × 101 columns

In [4]:
```
als_df.drop(['ID', 'SubjectID'], axis=1, inplace=True)
```

For this next section I will compare the correlations to see if there are redundant variables to remove

In [5]:
```
# create correlation matrix
corr_matrix = als_df.corr()
```

In [6]:

```
# this function highlights high correlation values
def high_corr(val):
    color = 'yellow' if (val > 0.9 or val < -0.9) else ''
    return 'background-color: {}'.format(color)

corr_matrix.style.applymap(high_corr)
```

Out[6]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_ra
Age_mean	1.000000	-0.276195	-0.349024	-0.297121	0.053
Albumin_max	-0.276195	1.000000	0.780141	0.596662	0.223
Albumin_median	-0.349024	0.780141	1.000000	0.761269	-0.091
Albumin_min	-0.297121	0.596662	0.761269	1.000000	-0.369
Albumin_range	0.053197	0.223350	-0.091822	-0.369015	1.000
ALSFRS_slope	-0.015301	0.037438	0.059234	0.112154	-0.226
ALSFRS_Total_max	0.049054	0.113349	0.153280	0.158924	-0.142
ALSFRS_Total_median	0.057733	0.090439	0.128122	0.172940	-0.166
ALSFRS_Total_min	0.041025	0.058077	0.099099	0.188007	-0.188
ALSFRS_Total_range	0.038163	-0.072609	-0.099966	-0.151358	0.264
ALT.SGPT._max	-0.130050	0.091963	0.101377	0.034264	0.015
ALT.SGPT._median	-0.189788	0.137417	0.187458	0.162226	-0.051
ALT.SGPT._min	-0.142516	0.085657	0.147607	0.173052	-0.072
ALT.SGPT._range	-0.058298	0.008734	0.000620	-0.033994	0.109
AST.SGOT._max	-0.030284	0.060325	0.048049	0.007431	0.005
AST.SGOT._median	-0.024973	0.096926	0.124172	0.109144	-0.047
AST.SGOT._min	-0.002730	0.032387	0.080313	0.114516	-0.071
AST.SGOT._range	-0.010642	0.004965	-0.014476	-0.036657	0.105
Bicarbonate_max	0.165844	0.125698	0.107236	0.025970	-0.010
Bicarbonate_median	0.191592	0.066279	0.046939	0.007117	0.015
Bicarbonate_min	0.169390	-0.028314	-0.040693	-0.029353	0.058
Bicarbonate_range	0.062786	0.008282	-0.007052	-0.016733	0.160
Blood.Urea.Nitrogen..BUN._max	0.218799	0.064980	-0.002630	-0.033055	0.008
Blood.Urea.Nitrogen..BUN._median	0.286131	-0.027990	-0.054270	-0.066050	0.006
Blood.Urea.Nitrogen..BUN._min	0.183931	-0.059244	-0.028821	0.026912	-0.026
Blood.Urea.Nitrogen..BUN._range	0.142506	0.010039	-0.073901	-0.083650	0.171
bp_diastolic_max	0.005481	0.084957	0.114683	0.075712	-0.011
bp_diastolic_median	0.013110	0.084578	0.156109	0.137294	-0.030
bp_diastolic_min	0.017555	0.071161	0.136431	0.161045	-0.015

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_ra
bp_diastolic_range	0.046004	-0.074635	-0.086270	-0.095153	0.16(
bp_systolic_max	0.327372	-0.009723	-0.029938	-0.052392	0.00
bp_systolic_median	0.317989	-0.005592	-0.001736	-0.005762	0.01
bp_systolic_min	0.258714	0.045570	0.049809	0.049491	0.01
bp_systolic_range	0.196450	-0.123900	-0.144217	-0.138054	0.16
Calcium_max	0.008545	0.155105	0.150450	0.131857	-0.01
Calcium_median	-0.010566	0.286746	0.336714	0.293908	0.00
Calcium_min	0.006702	0.094279	0.128773	0.169644	-0.00
Calcium_range	0.024142	-0.012157	-0.053379	-0.057257	0.13
Chloride_max	-0.091505	0.035649	0.037558	0.017007	-0.10
Chloride_median	-0.141916	-0.009322	0.029715	0.031809	-0.11(
Chloride_min	-0.160119	-0.072399	-0.006120	0.059682	-0.08
Chloride_range	0.101587	-0.025731	-0.082791	-0.068252	0.20
Creatinine_max	0.053231	0.098292	0.110474	0.099011	-0.02(
Creatinine_median	0.041418	0.050303	0.090986	0.117553	-0.01
Creatinine_min	0.010950	0.009317	0.040177	0.139861	-0.01
Creatinine_range	0.086564	0.026042	0.010782	-0.067427	0.15
Gender_mean	-0.168238	0.161496	0.241774	0.225021	-0.06(
Glucose_max	0.120487	0.010767	0.009631	-0.012418	-0.01
Glucose_median	0.133152	0.026261	0.026086	0.009595	-0.00(
Glucose_min	-0.007844	-0.018847	0.056898	0.090714	-0.03
Glucose_range	0.132036	-0.028908	-0.056251	-0.062965	0.11
hands_max	0.173512	-0.001439	-0.007101	0.015838	-0.07
hands_median	0.185773	-0.023482	-0.028485	0.015642	-0.09
hands_min	0.179134	-0.033976	-0.028484	0.036182	-0.10(
hands_range	-0.013099	-0.041942	-0.055227	-0.071624	0.18
Hematocrit_max	-0.034760	0.091769	0.036835	-0.019991	0.02
Hematocrit_median	-0.041498	0.090150	0.042328	-0.012497	0.02
Hematocrit_min	-0.057266	0.094050	0.054505	0.028419	0.00
Hematocrit_range	0.070984	-0.059026	-0.117044	-0.146325	0.23
Hemoglobin_max	-0.181186	0.159079	0.205137	0.189739	-0.02
Hemoglobin_median	-0.206245	0.152803	0.238150	0.239787	-0.05
Hemoglobin_min	-0.189124	0.102448	0.202851	0.298236	-0.10
Hemoglobin_range	0.056863	-0.044372	-0.107180	-0.146709	0.20

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_ra
leg_max	-0.047906	0.139818	0.179596	0.167852	-0.073
leg_median	-0.029621	0.134157	0.166601	0.175830	-0.083
leg_min	-0.041440	0.091300	0.130834	0.174243	-0.126
leg_range	0.039065	-0.008516	-0.015230	-0.037524	0.185
mouth_max	-0.055647	0.050458	0.099309	0.115960	-0.145
mouth_median	-0.054257	0.064704	0.109045	0.144995	-0.145
mouth_min	-0.034408	0.062803	0.108634	0.176298	-0.165
mouth_range	0.038895	-0.116434	-0.153355	-0.196108	0.223
onset_delta_mean	-0.039550	-0.003759	0.048405	0.025833	0.038
onset_site_mean	-0.090055	0.006648	0.027399	0.054884	-0.045
Platelets_max	0.037074	-0.104745	-0.141115	-0.189825	0.091
Platelets_median	0.002051	-0.115375	-0.137441	-0.104450	0.057
Platelets_min	0.006589	-0.109240	-0.122950	-0.076696	0.056
Potassium_max	0.040688	-0.004157	-0.009839	-0.002625	-0.003
Potassium_median	0.144203	-0.004223	0.012496	0.018269	-0.023
Potassium_min	0.034911	-0.028865	0.006738	0.071035	-0.026
Potassium_range	0.053625	-0.041060	-0.062979	-0.047411	0.163
pulse_max	-0.077080	0.014157	0.000910	-0.037239	0.048
pulse_median	-0.066583	-0.009456	-0.001371	-0.002951	0.031
pulse_min	-0.033929	-0.002595	-0.001306	0.007479	0.035
pulse_range	0.012406	-0.081420	-0.089154	-0.100420	0.229
respiratory_max	-0.071695	0.056793	0.058894	0.061554	-0.088
respiratory_median	-0.055059	0.090280	0.104120	0.107924	-0.102
respiratory_min	-0.055515	0.081607	0.108920	0.158138	-0.138
respiratory_range	0.064244	-0.127569	-0.156803	-0.161566	0.193
Sodium_max	0.028854	0.051798	0.049189	0.014051	-0.004
Sodium_median	0.005432	0.013677	0.007380	0.046761	0.030
Sodium_min	-0.039712	-0.040877	-0.017818	0.074240	0.020
Sodium_range	0.080449	-0.048029	-0.069380	-0.086457	0.177
trunk_max	0.071736	0.080261	0.097823	0.095974	-0.086
trunk_median	0.089325	0.057346	0.084183	0.113639	-0.110
trunk_min	0.065092	0.042903	0.072317	0.134409	-0.124
trunk_range	0.036559	-0.064183	-0.088370	-0.114685	0.211
Urine.Ph_max	-0.001532	0.075413	0.056894	0.034555	-0.011

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_ra
Urine.Ph_median	0.002561	-0.046885	-0.042497	-0.001877	0.038
Urine.Ph_min	-0.008615	-0.138430	-0.118506	-0.049783	0.022

According to the correlation matrix above, there are two redundant variables. I will remove 'Hematocrit_max' and 'Hematocrit_min' and I will leave 'Hematocrit_median' in the dataframe.

In [7]:
```
als_df.drop(['Hematocrit_max', 'Hematocrit_min'], axis=1, inplace=True)
```

In [8]:
```
# Any missing data?
als_df.isna().sum().sum()
```

Out[8]:
```
0
```

2. Apply a standard scalar to the data.

In [9]:
```
# create object
scalar = StandardScaler()
```

In [10]:
```
# standardization
als_scaled = scalar.fit_transform(als_df)
```

In [11]:
```
als_scaled
```

Out[11]:
```
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]])
```

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

The following code was sourced from a tutorial online that can be found at the following website: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py

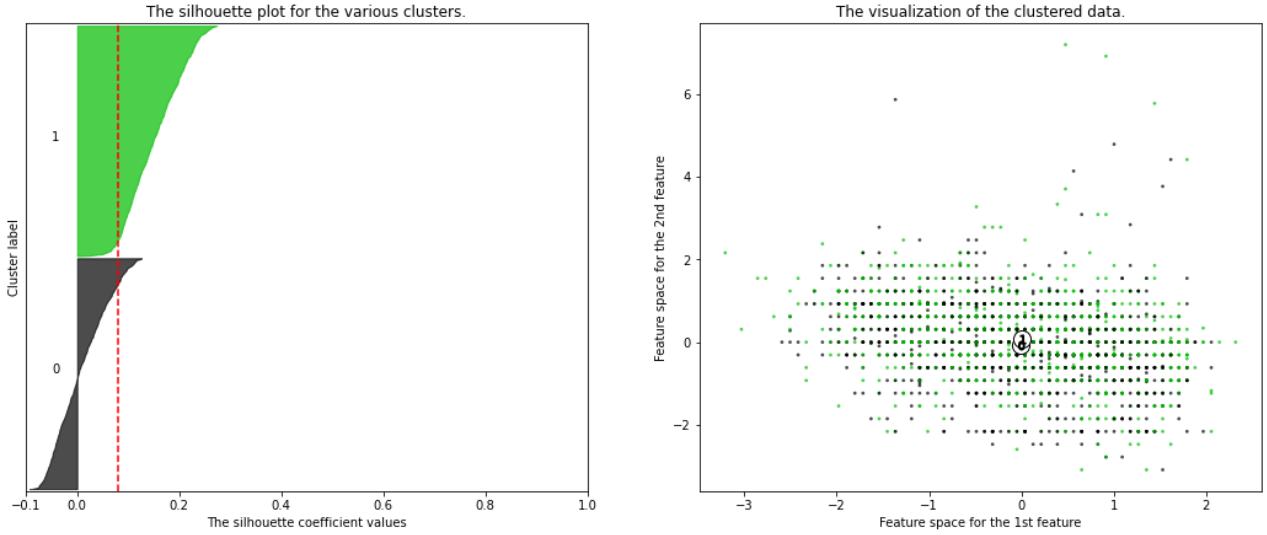
In [12]:
```
X = als_scaled

range_n_clusters = [2, 3, 4, 5, 6,7, 8, 9, 10]

for n_clusters in range_n_clusters:
```

```
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print(
        "For n_clusters =",
        n_clusters,
        "The average silhouette_score is :",
        silhouette_avg,
    )

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
```

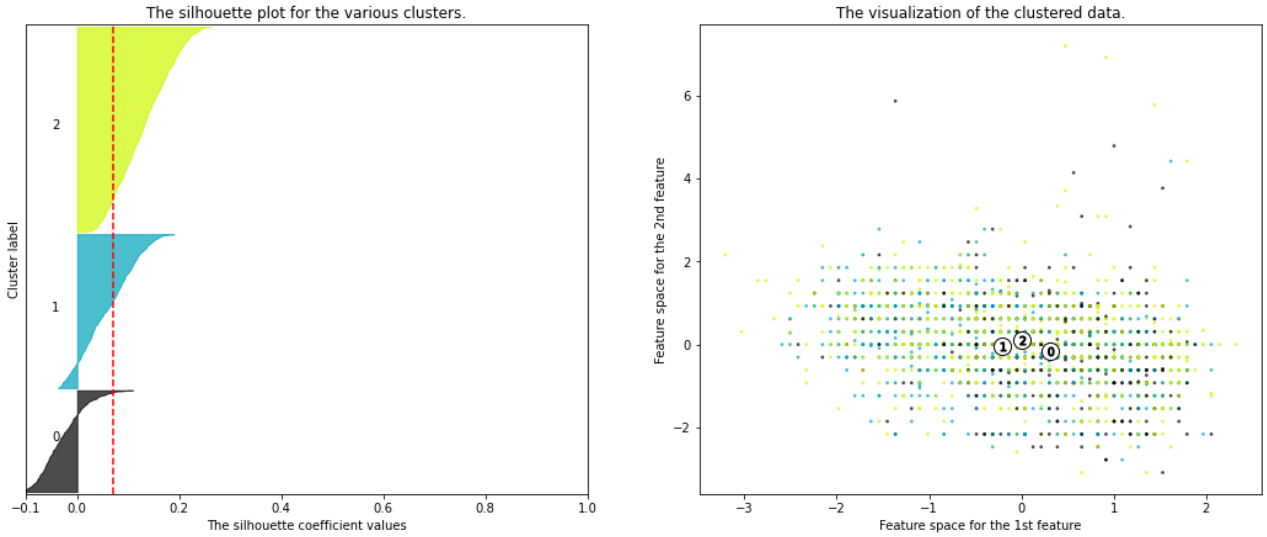
```
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(
        X[:, 0], X[:, 1], marker=".", s=30, lw=0, alpha=0.7, c=colors, edgecolor="k"
    )

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(
        centers[:, 0],
        centers[:, 1],
        marker="o",
        c="white",
        alpha=1,
        s=200,
        edgecolor="k",
    )

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor="k")

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(
        "Silhouette analysis for KMeans clustering on sample data with n_clusters = %d"
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )

plt.show()
```

```
For n_clusters = 2 The average silhouette_score is : 0.08038173811738773
For n_clusters = 3 The average silhouette_score is : 0.07040893083421151
For n_clusters = 4 The average silhouette_score is : 0.05210973554529861
For n_clusters = 5 The average silhouette_score is : 0.05712969619618897
For n_clusters = 6 The average silhouette_score is : 0.04047689865763422
For n_clusters = 7 The average silhouette_score is : 0.04361954954738954
For n_clusters = 8 The average silhouette_score is : 0.04426670836990529
For n_clusters = 9 The average silhouette_score is : 0.03746952706901966
For n_clusters = 10 The average silhouette_score is : 0.036686234733113124
```

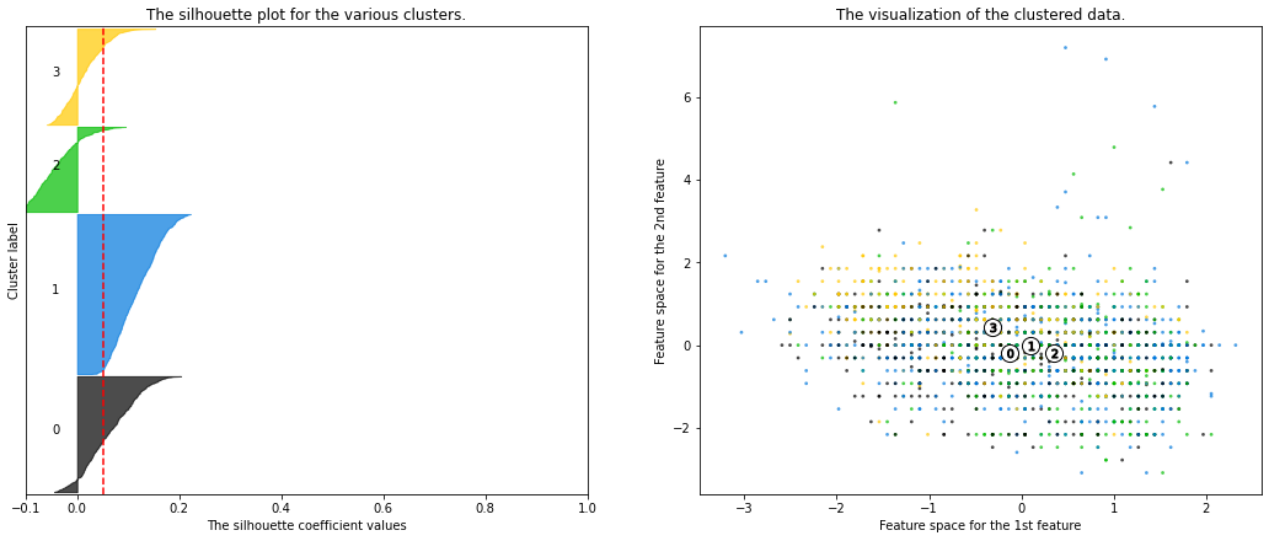
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2



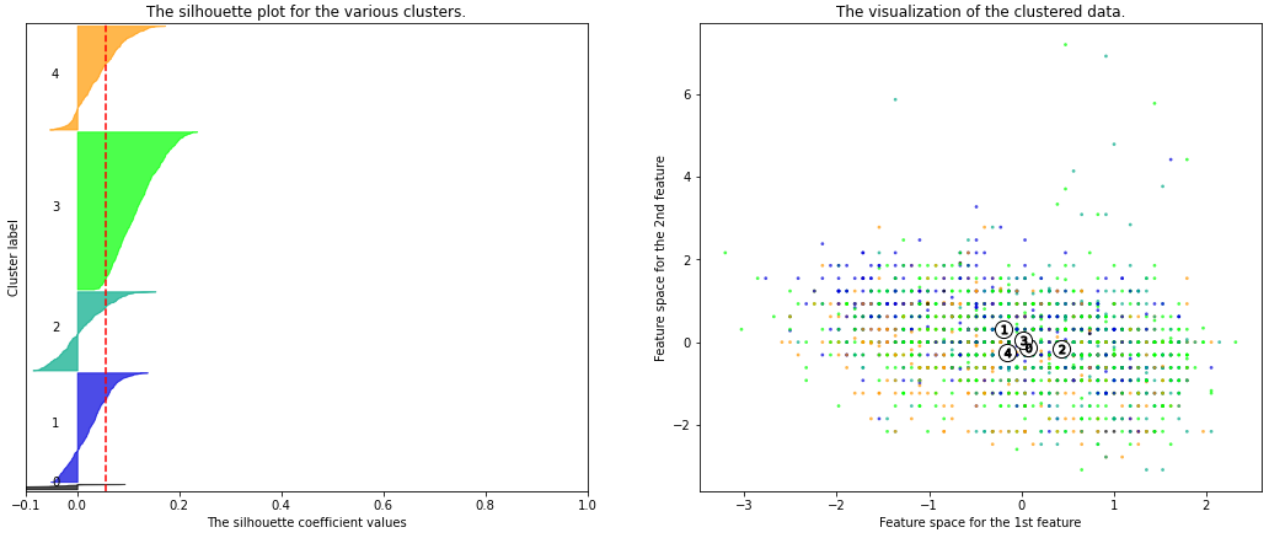
Silhouette analysis for KMeans clustering on sample data with n_clusters = 3



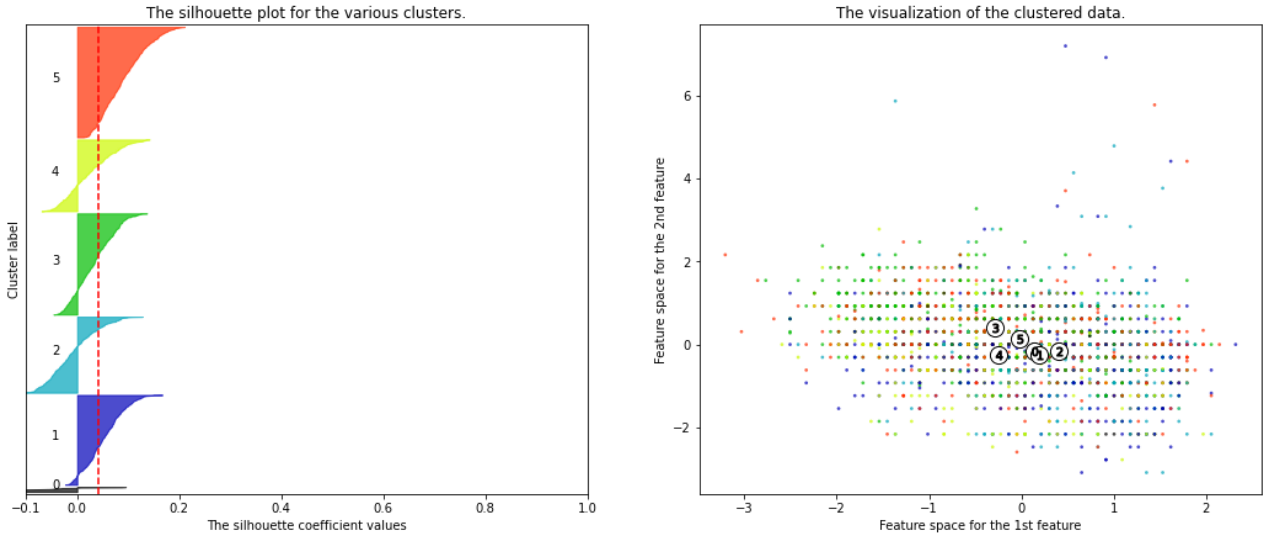
Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



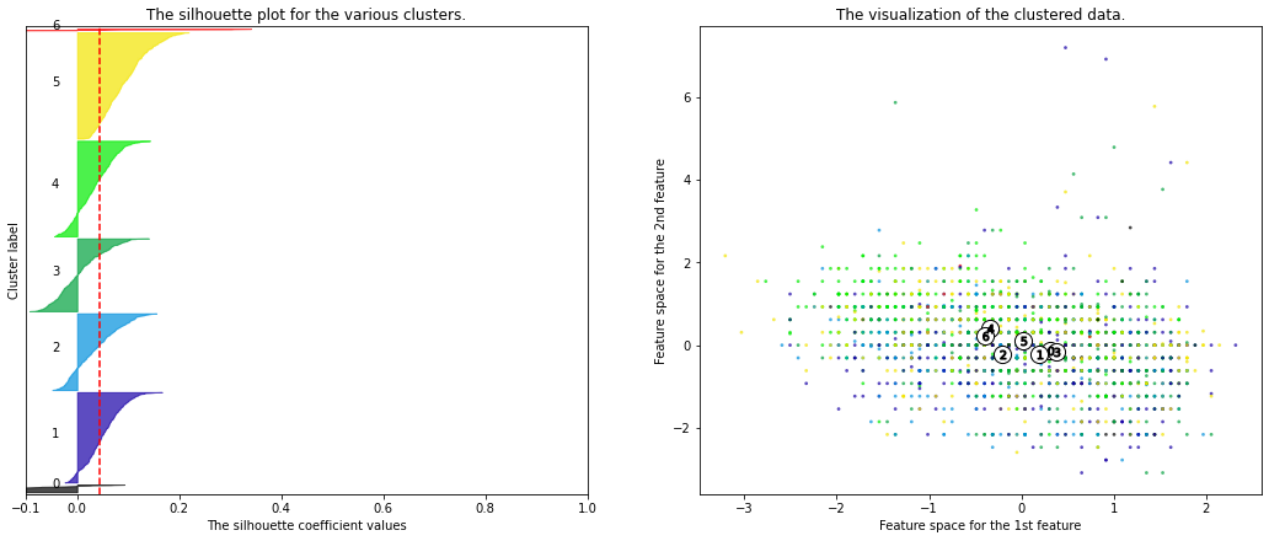
Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

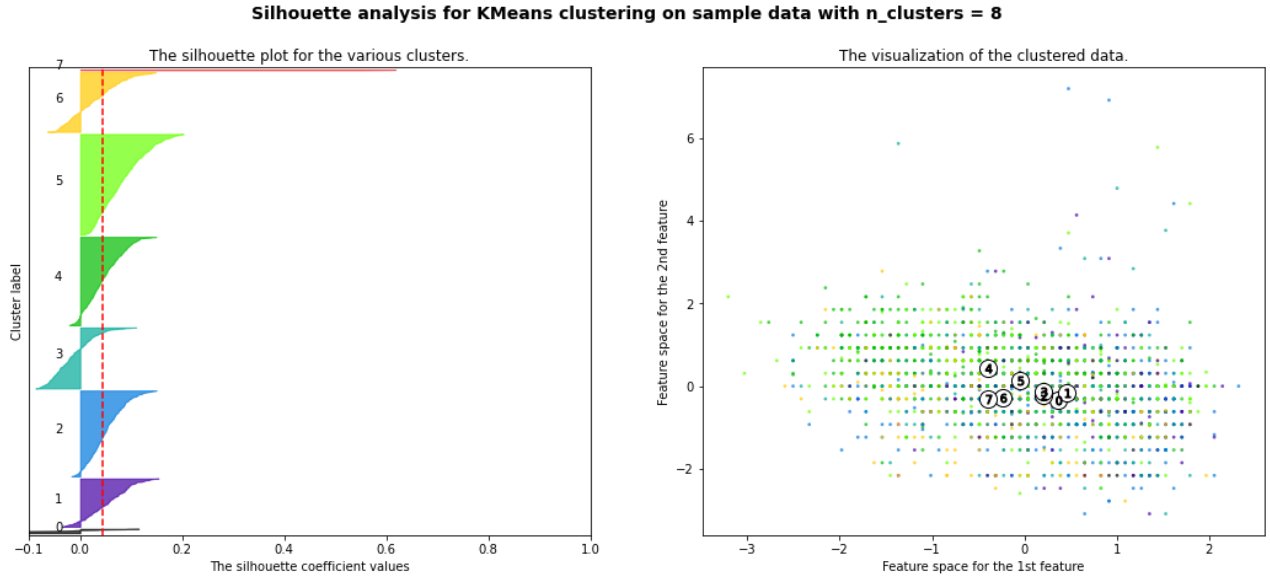


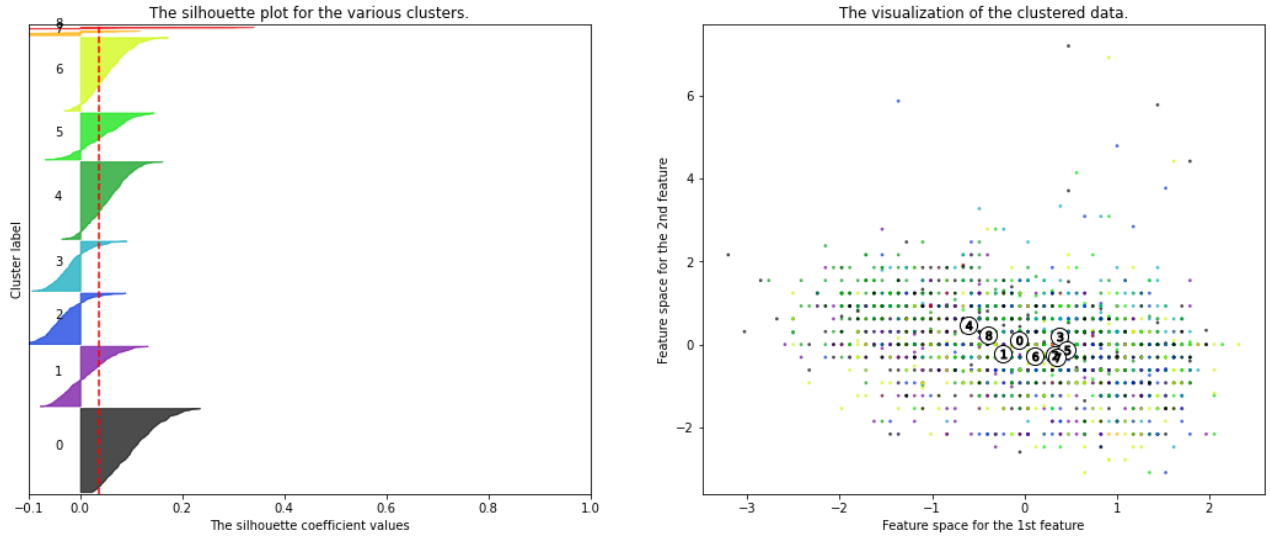
Silhouette analysis for KMeans clustering on sample data with n_clusters = 6

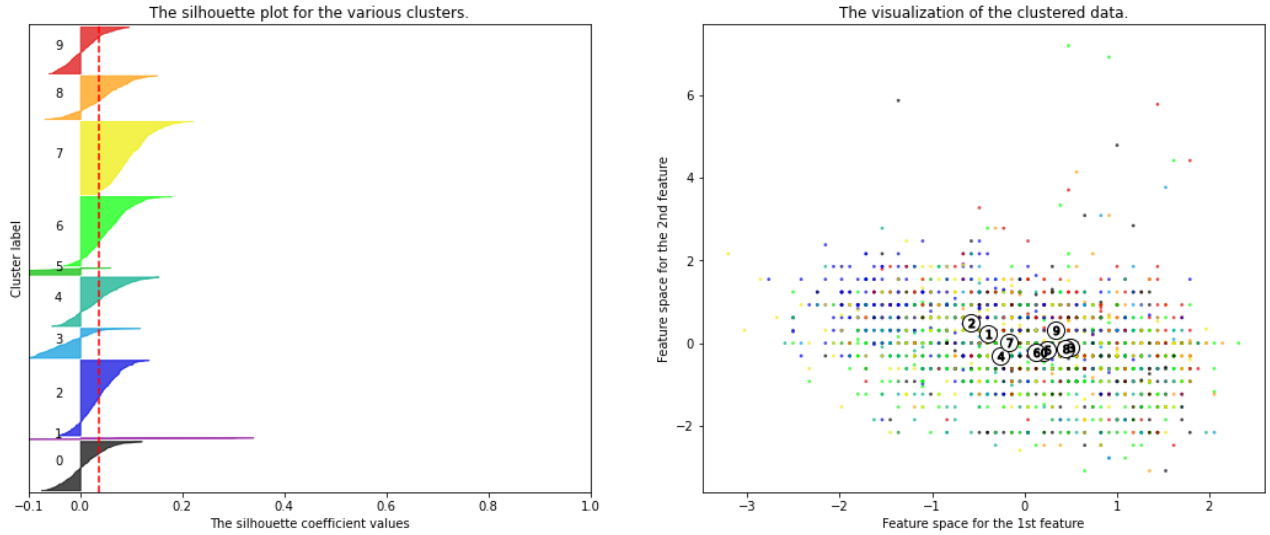


Silhouette analysis for KMeans clustering on sample data with n_clusters = 7









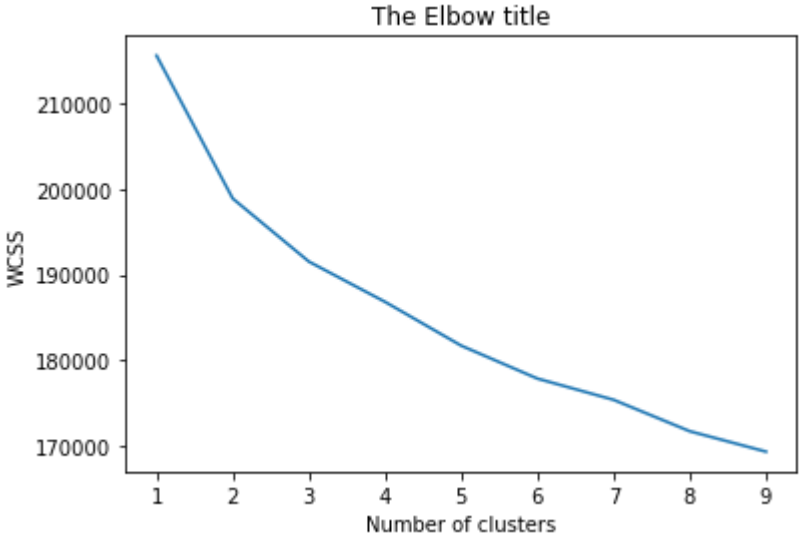
In [13]:

```
# An alternative approach: WCSS and Elbow Plot
wcss=[]
for i in range(1,10):
    kmeans = KMeans(i)
```

```
    kmeans.fit(X)
    wcss_iter = kmeans.inertia_
    wcss.append(wcss_iter)

number_clusters = range(1,10)
plt.plot(number_clusters,wcss)
plt.title('The Elbow title')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=9.
  warnings.warn(
```

Out[13]:
```
Text(0, 0.5, 'WCSS')
```



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

After reviewing both the silhouette scores and the elbow plot, both of these metrics indicate that the best choice for k is k=2. The silhouette score, though very low for all values of k, was the highest for k=2. Furthermore, the "elbow" of the elbow plot is also at k=2.

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

In [14]:
```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
label = kmeans.fit_predict(X)
```

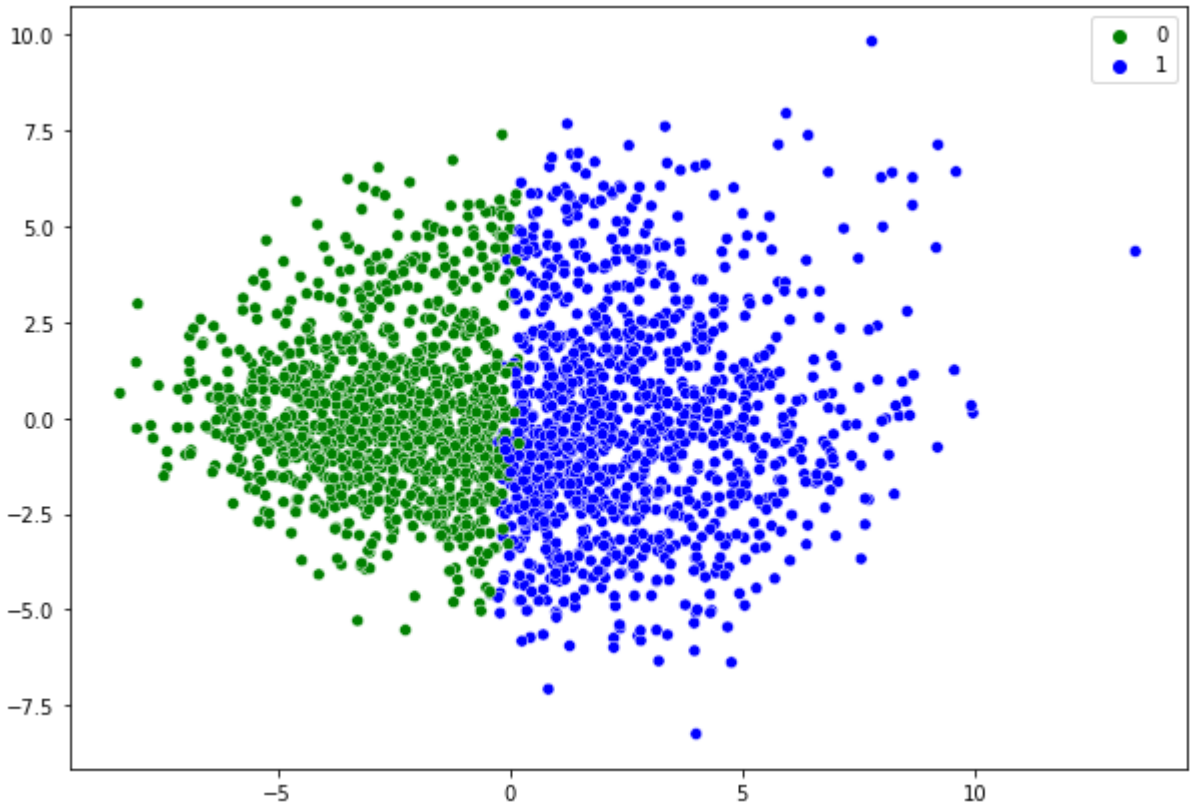
6. Fit a PCA transformation with two features to the scaled data.

In [15]:
```
pca_2 = PCA(n_components=2)
pca_2.fit(X)
X_pca_2 = pca_2.transform(X)
```

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

In [16]:
```
plt.figure(figsize=(10,7))
sns.scatterplot(x=X_pca_2[:,0], y=X_pca_2[:,1], hue=label, palette = ['green', 'blue'])
```

Out[16]: <AxesSubplot:>



8. Summarize your results and make a conclusion.

In conclusion, when doing a k-means clustering unsupervised learning algorithm on the ALS dataframe, you will find that k=2 is the optimal number of clusters. This can be seen from the silhouette scores for each value of k, with k=2 being the highest, and also from the elbow plot which has an elbow at k=2. After doing a PCA transform to reduce the number of variables, we are able to two-dimensionally plot the resultant clusters via scatterplot.