Assignment 4.2: Clustering Exercise

You will be using the dataset als_data.csv to apply clustering methods for this assignment. This data gives anonymized data on ALS patients. With this data, complete the following steps:

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

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# importing warning to surpress the future warnings
import warnings
warnings.simplefilter(action='ignore')
```

In [2]:
```
# reading the data frame and saving to a variable
df = pd.read_csv('Datasets/als_data.csv')
```

In [3]:
```
# reviewing first 10 rows
df.head(10)
```

Out[3]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALS
0	1	65	57.0	40.5	38.0	0.066202	-0.965608	30	28.0	
1	2	48	45.0	41.0	39.0	0.010453	-0.921717	37	33.0	
2	3	38	50.0	47.0	45.0	0.008929	-0.914787	24	14.0	
3	4	63	47.0	44.0	41.0	0.012111	-0.598361	30	29.0	
4	5	63	47.0	45.5	42.0	0.008292	-0.444039	32	27.5	
5	6	36	51.0	47.0	46.0	0.009058	-0.118353	37	34.5	
6	7	55	46.0	44.0	40.0	0.010850	-1.225580	34	24.0	
7	8	55	45.0	42.0	38.0	0.018519	-0.760417	30	27.5	
8	9	37	48.0	46.0	41.0	0.012681	-1.010148	35	28.5	
9	11	72	44.0	42.0	38.0	0.010714	-0.107861	28	25.5	

10 rows × 101 columns

In [4]:
```
df.describe()
```

Out[4]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Tota
count	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	22
mean	1214.874944	54.550157	47.011134	43.952542	40.766347	0.013779	-0.728274	31.692308	
std	696.678300	11.396546	3.233980	2.654804	3.193087	0.009567	0.622329	5.314228	
min	1.000000	18.000000	37.000000	34.500000	24.000000	0.000000	-4.345238	11.000000	
25%	614.500000	47.000000	45.000000	42.000000	39.000000	0.009042	-1.086310	29.000000	
50%	1213.000000	55.000000	47.000000	44.000000	41.000000	0.012111	-0.620748	33.000000	
75%	1815.500000	63.000000	49.000000	46.000000	43.000000	0.015873	-0.283832	36.000000	
max	2424.000000	81.000000	70.300000	51.100000	49.000000	0.243902	1.207011	40.000000	

8 rows × 101 columns

In [5]:
```
# dropping columns that are not relevant
df.drop(['ID','SubjectID'], axis=1, inplace = True)
```

In [6]:
```
# reviewing the shape of the data frame to verify columns decreased
df.shape
```

Out[6]:
```
(2223, 99)
```

2. Apply a standard scalar to the data.

In [7]:
```
from sklearn.preprocessing import StandardScaler
```

In [8]:
```
scaler = StandardScaler()
```

In [9]:
```
scaled_df = scaler.fit_transform(df)
```

In [10]:
```
# reviewing the scaled_df
```

```
scaled_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]])
```

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

In [11]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import seaborn as sns
```

In [12]:
```
#instantiate the k-means clustering with 10 clusters
kmeans = KMeans(init="random", n_clusters=10, n_init=8, random_state=42)
```

In [13]:
```
#fit k-means algorithm to data
kmeans.fit(scaled_df)
```

Out[13]:
```
▾ KMeans
KMeans(init='random', n_clusters=10, n_init=8, random_state=42)
```

In [14]:
```
#view cluster assignments for each observation
kmeans.labels_
```

Out[14]:
```
array([4, 4, 5, ..., 8, 8, 0])
```

In [15]:
```
# generating the clusters / labels
labels = kmeans.fit_predict(scaled_df)
```

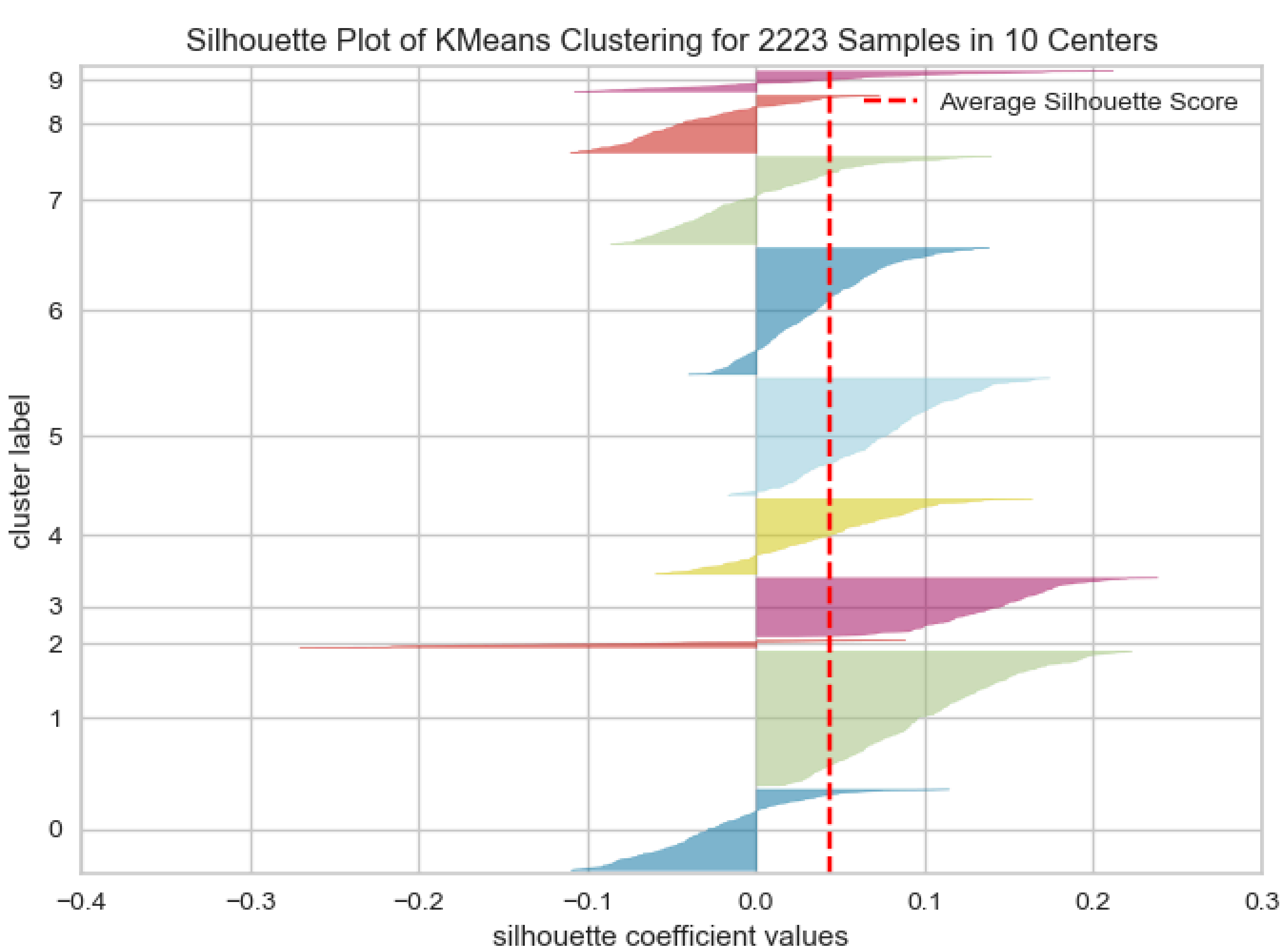
In [16]:
```
# generating the silouette score for the data frame
ss = silhouette_score(scaled_df, labels)
ss
```

Out[16]:
```
0.04369670037038112
```

In [17]:
```
# importing additional modules
from yellowbrick.cluster import SilhouetteVisualizer
```

In [18]:
```
# using the silhouette visualizer function from yellowbricks library to generate the plot
visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick')

visualizer.fit(scaled_df)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
plt.show()
```



The above figure shows the silhouette plot of the 10 clusters created during the kmeans generation. I used the module from the yellowbrick library to create the silhouette plot. It made the process of creating the plot easy and efficient.

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

The above figure shows the silhouette plot of the 10 clusters created during the kmeans generation. I used the module from the yellowbrick library to create the silhouette plot. It made the process of creating the plot easy and efficient. The ideal number of clusters appears to be 4 clusters. The reasoning for this is that out of the 10, one is below the average silhouette score and the majority of the others show a wide fluctuation in the silhouette plot.

I feel that the second standard may be a little subjective but ultimatley I tried to narrow down the clusters to the ones with the least fluctuation in the silhouette plot.

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

In [19]:
```
# initialize the kmeans clustering with the optimal number of clusters from part 4
ideal_kmeans = KMeans(init="random", n_clusters=4, n_init=10, random_state=42)
```

In [20]:
```
#fit k-means algorithm to data
ideal_kmeans.fit(scaled_df)
```

Out[20]:
```
KMeans
KMeans(init='random', n_clusters=4, n_init=10, random_state=42)
```

In [21]:
```
# generating the clusters / labels
labels = ideal_kmeans.fit_predict(scaled_df)
```

In [22]:
```
len(labels)
```

Out[22]:
```
2223
```

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

In [23]:
```
# importing additional tools
from sklearn.decomposition import PCA
```

In [24]:
```
# creating the pca variable
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(scaled_df) # fitting and tranforming the the pca to the scaled data
```

In [25]:
```
# saving the two pca tranformed components to a data frame
principal_df = pd.DataFrame(data = principalComponents , columns = ['component_1', 'component_2'])
```

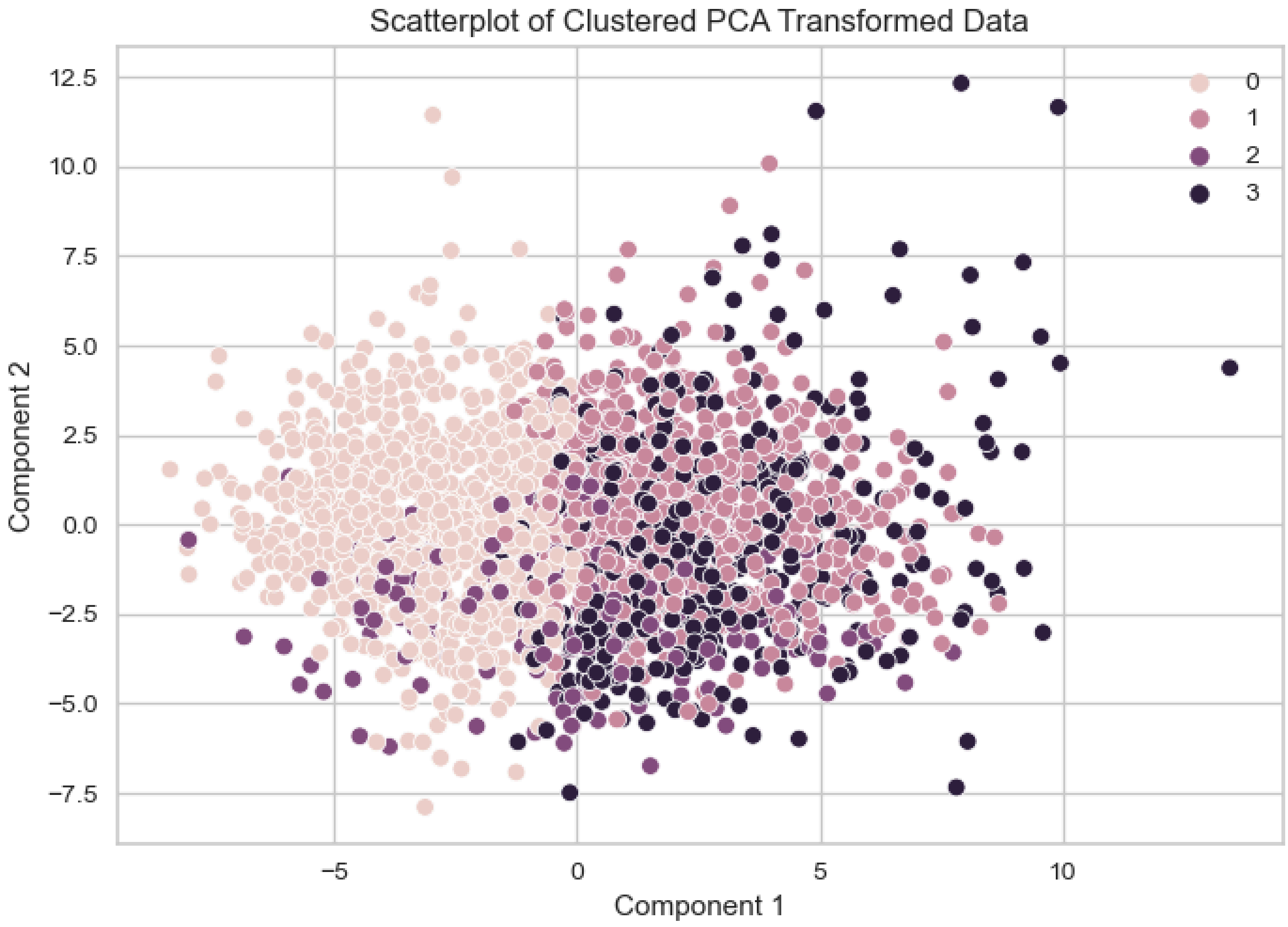
In [26]:
```
# reviewing the final data frame
principal_df.head(5)
```

Out[26]:

	component_1	component_2
0	-1.426765	-2.320810
1	-1.440227	-4.871206
2	1.617841	-0.428528
3	-1.920006	2.095116
4	0.297709	0.169031

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

In [28]:
```
# creating a scatterplot of the PCA fitted data frame
sns.scatterplot(x = principal_df['component_1'], y = principal_df['component_2'], hue = labels)
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.title('Scatterplot of Clustered PCA Transformed Data')
plt.show()
```



This makes apparent that I may have mis picked the ideal number of clusters. Even though it appears I made some mistakes it's interesting to see the mistakes.

8. Summarize your results and make a conclusion.

The final plot appears to make it clear that the ideal number of clusters should have been two. The number of clusters picked was four and by applying the colors by cluster to the PCA transformed data and plot it becomes evident that there may be some errors.

It's interesting to visually see the color-coded clusters. Since there are two distinct colors in two distinct clusters with a mix of the two remaining colors scattered without clusters, it makes it appear obvious that they should not be clustered.

I chose not to correct my cluster number pick on purpose, I wanted to leave this as an example of what I did so I could improve upon it in the future.