Week 4 Assignment - Descriptive Modeling

Name: Kesav Adithya Venkidusamy
Course: DSC630 - Predictive Analytics
Instructor: Fadi Alsaleem

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:

Importing all the libraries required for this exercise

In [26]:
```
## Importing libraries required for this assignment
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
```

In [13]:
```
## Display all columns in pandas dataframe
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Load the Dataset into dataframe

In [14]:
```
## Load the ALS data into a dataframe
als_df = pd.read_csv('als_data.csv')
als_df.head(5)
```

Out[14]:

	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	

In [15]:
```
## Printing number of rows and columns of als dataframe
als_df.shape
```

Out[15]:
```
(2223, 101)
```

In [16]:
```
## Printing the dtype for each of the column
als_df.dtypes
```

Out[16]:
```
ID                                    int64
Age_mean                              int64
Albumin_max                         float64
Albumin_median                      float64
Albumin_min                         float64
Albumin_range                       float64
ALSFRS_slope                        float64
ALSFRS_Total_max                      int64
ALSFRS_Total_median                 float64
ALSFRS_Total_min                      int64
ALSFRS_Total_range                  float64
ALT.SGPT._max                       float64
ALT.SGPT._median                    float64
ALT.SGPT._min                       float64
ALT.SGPT._range                     float64
AST.SGOT._max                         int64
AST.SGOT._median                    float64
AST.SGOT._min                       float64
AST.SGOT._range                     float64
Bicarbonate_max                     float64
Bicarbonate_median                  float64
Bicarbonate_min                     float64
Bicarbonate_range                   float64
Blood.Urea.Nitrogen..BUN._max       float64
Blood.Urea.Nitrogen..BUN._median    float64
Blood.Urea.Nitrogen..BUN._min       float64
Blood.Urea.Nitrogen..BUN._range     float64
bp_diastolic_max                      int64
bp_diastolic_median                 float64
bp_diastolic_min                      int64
bp_diastolic_range                  float64
bp_systolic_max                       int64
bp_systolic_median                  float64
bp_systolic_min                       int64
bp_systolic_range                   float64
Calcium_max                         float64
Calcium_median                      float64
Calcium_min                         float64
Calcium_range                       float64
Chloride_max                        float64
Chloride_median                     float64
Chloride_min                        float64
Chloride_range                      float64
Creatinine_max                      float64
Creatinine_median                   float64
Creatinine_min                      float64
Creatinine_range                    float64
Gender_mean                           int64
Glucose_max                         float64
Glucose_median                      float64
Glucose_min                         float64
Glucose_range                       float64
hands_max                             int64
hands_median                        float64
hands_min                             int64
hands_range                         float64
Hematocrit_max                      float64
Hematocrit_median                   float64
Hematocrit_min                      float64
Hematocrit_range                    float64
Hemoglobin_max                      float64
Hemoglobin_median                   float64
Hemoglobin_min                      float64
```

```
Hemoglobin_range                   float64
leg_max                              int64
leg_median                         float64
leg_min                              int64
leg_range                          float64
mouth_max                            int64
mouth_median                       float64
mouth_min                            int64
mouth_range                        float64
onset_delta_mean                     int64
onset_site_mean                      int64
Platelets_max                        int64
Platelets_median                   float64
Platelets_min                      float64
Potassium_max                      float64
Potassium_median                   float64
Potassium_min                      float64
Potassium_range                    float64
pulse_max                            int64
pulse_median                       float64
pulse_min                            int64
pulse_range                        float64
respiratory_max                      int64
respiratory_median                 float64
respiratory_min                      int64
respiratory_range                  float64
Sodium_max                         float64
Sodium_median                      float64
Sodium_min                         float64
Sodium_range                       float64
SubjectID                            int64
trunk_max                            int64
trunk_median                       float64
trunk_min                            int64
trunk_range                        float64
Urine.Ph_max                       float64
Urine.Ph_median                    float64
Urine.Ph_min                       float64
dtype: object
```

In [17]:

```python
## Looking at summary information about your data (total, mean, min, max, freq, unique,
als_df.describe()
```

Out[17]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS
count	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.000000	2223.
mean	1214.874944	54.550157	47.011134	43.952542	40.766347	0.013779	-0.
std	696.678300	11.396546	3.233980	2.654804	3.193087	0.009567	0.
min	1.000000	18.000000	37.000000	34.500000	24.000000	0.000000	-4.
25%	614.500000	47.000000	45.000000	42.000000	39.000000	0.009042	-1.
50%	1213.000000	55.000000	47.000000	44.000000	41.000000	0.012111	-0.
75%	1815.500000	63.000000	49.000000	46.000000	43.000000	0.015873	-0.
max	2424.000000	81.000000	70.300000	51.100000	49.000000	0.243902	1.

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

On analyzing all the features present in the data set, I see SubjectID and ID features are irrelavent to patient's ALS condition. So, these variables can be removed from the dataset.

In [18]:
```
# Removing Patients ID and SubjectID fields from the dataset
als_df.drop(["SubjectID", "ID"], axis=1, inplace=True)
```

In [19]:
```
## Validating the dataset after removing the columns
als_df.shape
```

Out[19]:
```
(2223, 99)
```

In [20]:
```
## Displaying few records in the dataframe
als_df.head()
```

Out[20]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Tot
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	

2. Apply a standard scalar to the data.

In [22]:
```
## Define standard scaler
scaler = StandardScaler()

## Transform data
X = scaler.fit_transform(als_df)
print(X)
```

```
[[ 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 [23]:
```
## Calculating the shape of dataframe
X.shape
```

Out[23]:
```
(2223, 99)
```

In [25]:
```
## Calculating the mean and standard deviation of the dataset
np.mean(X),np.std(X)
```

Out[25]:
```
(-8.908541299845311e-17, 1.0)
```

Observation:

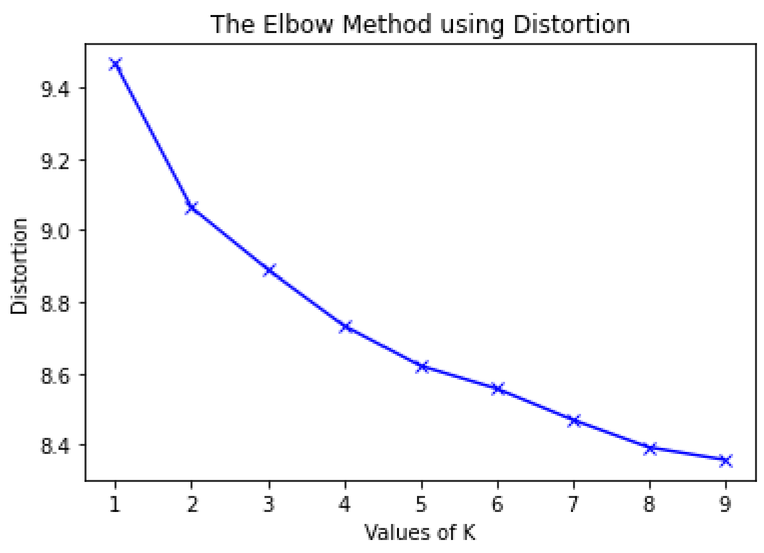
1. The mean value is close to zero as expected as the mean value should be close to 0 after applying StandardScalar
2. The Standard deviation is 1 which is also as expected

In [27]:
```

## Reference: https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmea
## Building the clustering model and calculating the values of the Distortion and Inert

distortions = []
inertias = []
mapping1 = {}
mapping2 = {}
K = range(1, 10)

for k in K:
    # Building and fitting the model
    kmeanModel = KMeans(n_clusters=k).fit(X)
    kmeanModel.fit(X)

    distortions.append(sum(np.min(cdist(X, kmeanModel.cluster_centers_,'euclidean'), ax
    inertias.append(kmeanModel.inertia_)

    mapping1[k] = sum(np.min(cdist(X, kmeanModel.cluster_centers_,'euclidean'), axis=1)
    mapping2[k] = kmeanModel.inertia_
```

In [28]:
```

## Tabulating and Visualizing the results
## Using the different values of Distortion
for key, val in mapping1.items():
    print(f'{key} : {val}')
```

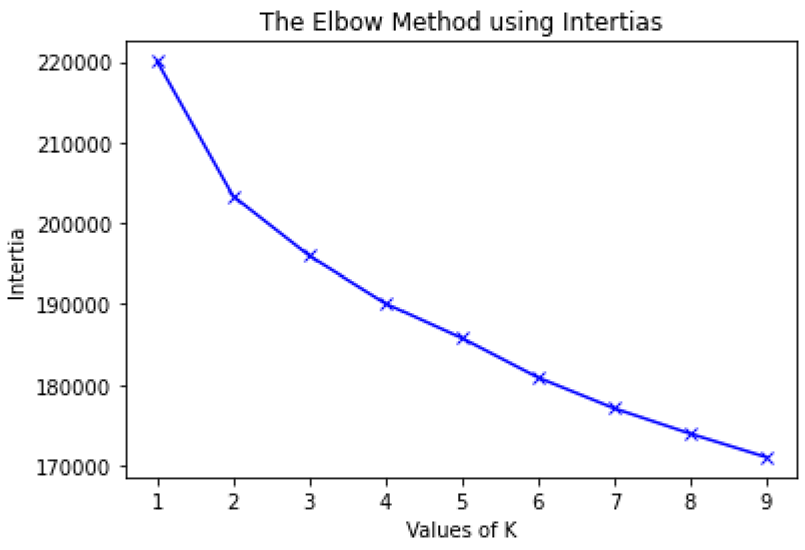
```
1 : 9.466971249370545
2 : 9.063781809801627
3 : 8.890733728366843
4 : 8.734143328933625
5 : 8.621578780871832
6 : 8.557595851395476
7 : 8.47019365921882
8 : 8.392994844892927
9 : 8.357677447737448
```

In [31]:
```

# Plot Elbow method to find K
plt.plot(K, distortions, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Distortion')
plt.title('The Elbow Method using Distortion')
plt.show()
```



In [32]:

```
# Plot Elbow method to find K
plt.plot(K, inertias, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Intertia')
plt.title('The Elbow Method using Intertias')
plt.show()
```



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

In [38]:

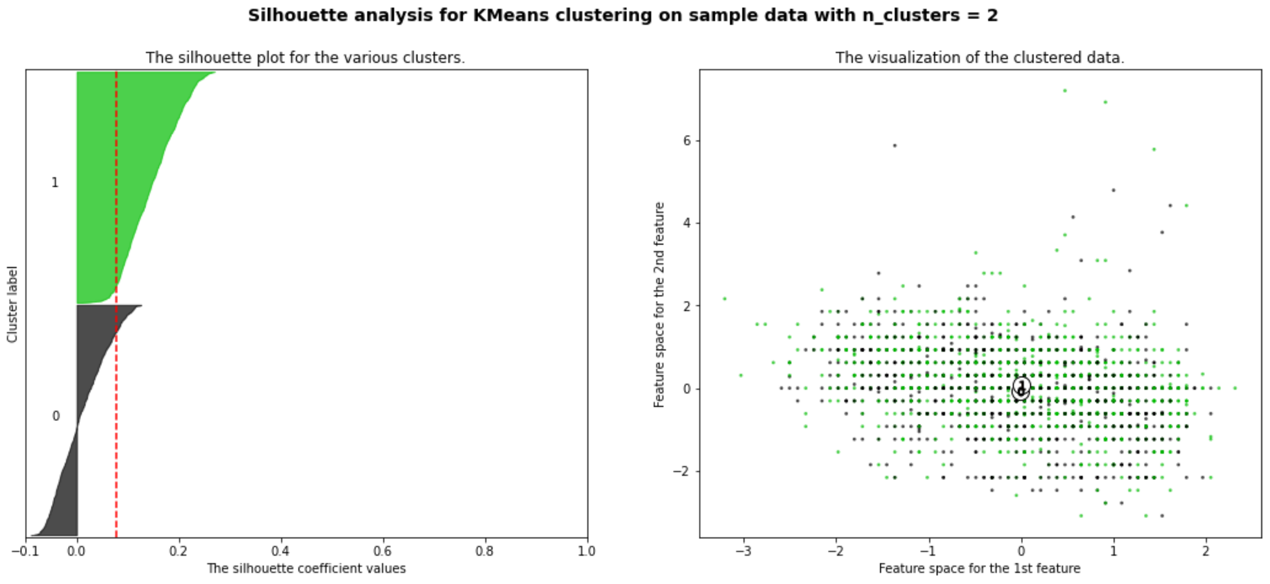
```
## Importing the libraries required for silhousette
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
import matplotlib.style as style
```

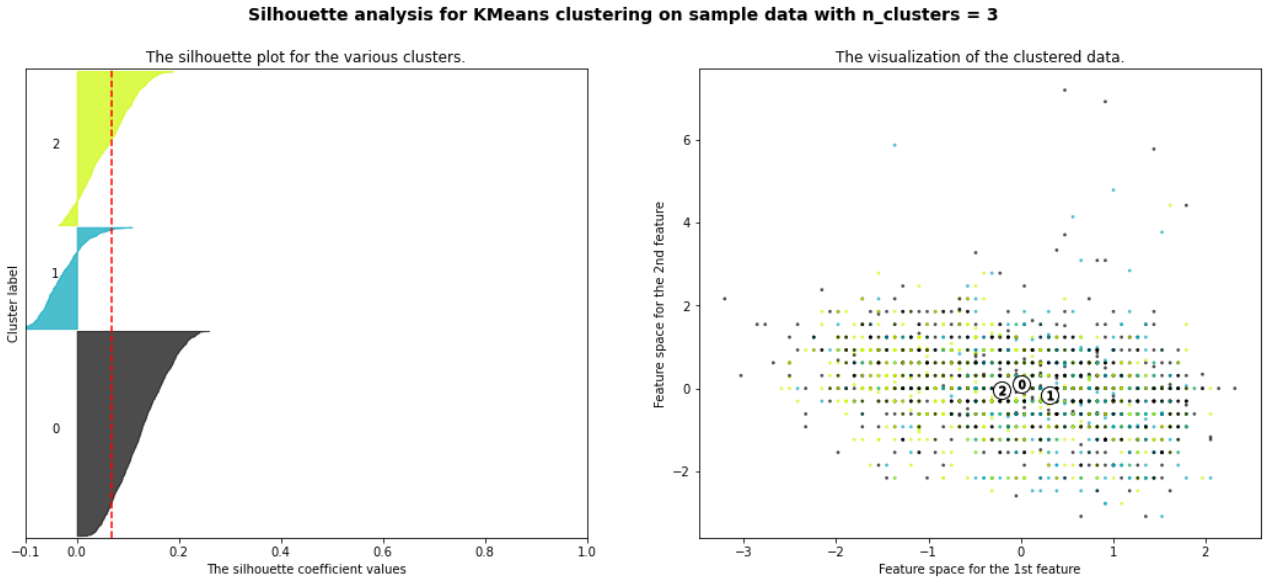
In [39]:

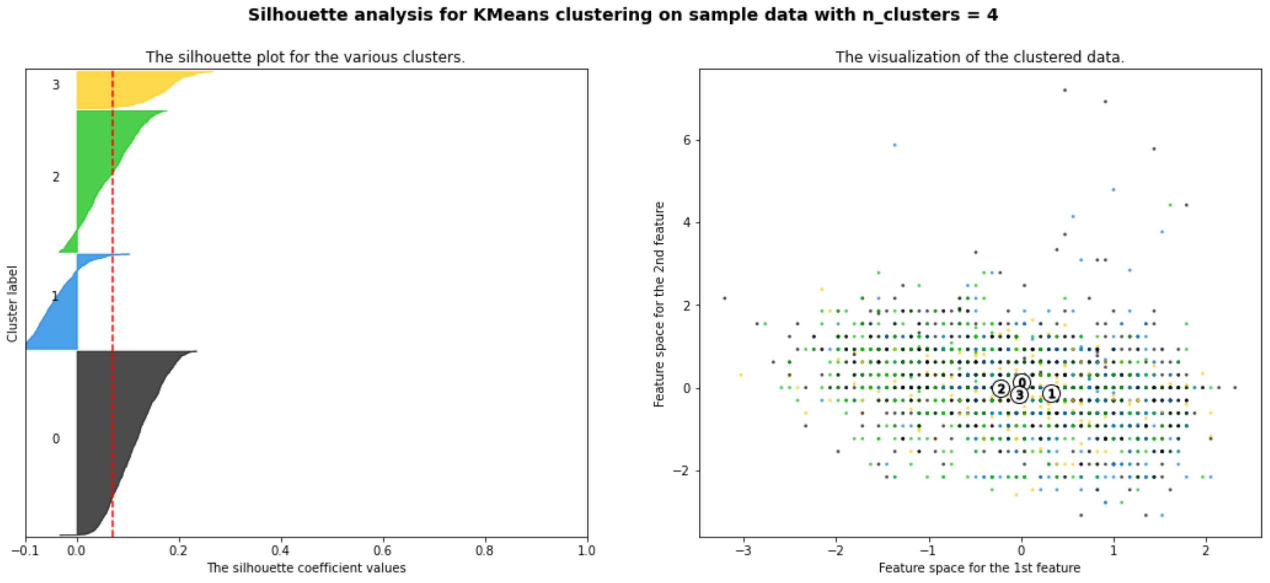
```
## Plotting the cluster silhouette score versus the number of clusters in a K-means clu

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

```
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=42)
    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)

    silhouette_avg_n_clusters.append(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()

style.use("fivethirtyeight")
plt.plot(range_n_clusters, silhouette_avg_n_clusters)
plt.xlabel("Number of Clusters (k)")
plt.ylabel("silhouette score")
plt.show()
```

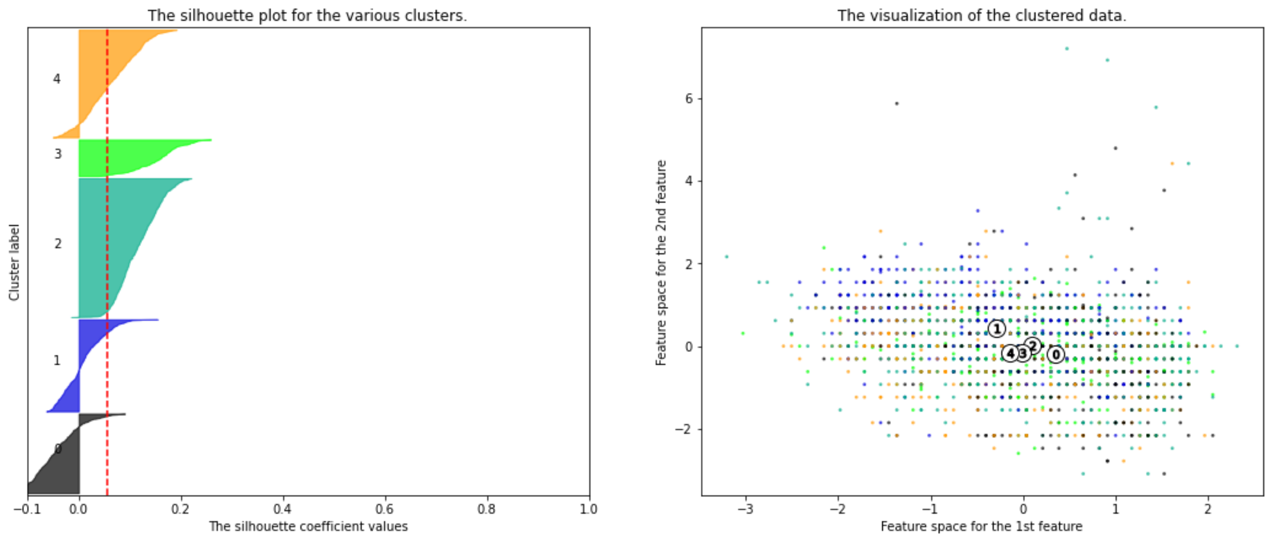
```
For n_clusters = 2 The average silhouette_score is : 0.07878005888570402
For n_clusters = 3 The average silhouette_score is : 0.0687707291658565
For n_clusters = 4 The average silhouette_score is : 0.06973816142698218
For n_clusters = 5 The average silhouette_score is : 0.05697679932842005
For n_clusters = 6 The average silhouette_score is : 0.06477886829610223
For n_clusters = 7 The average silhouette_score is : 0.05187647631845004
For n_clusters = 8 The average silhouette_score is : 0.04954004349267961
For n_clusters = 9 The average silhouette_score is : 0.04393719582297171
For n_clusters = 10 The average silhouette_score is : 0.046121611845315456
```


Silhouette analysis for KMeans clustering on sample data with n_clusters = 2
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

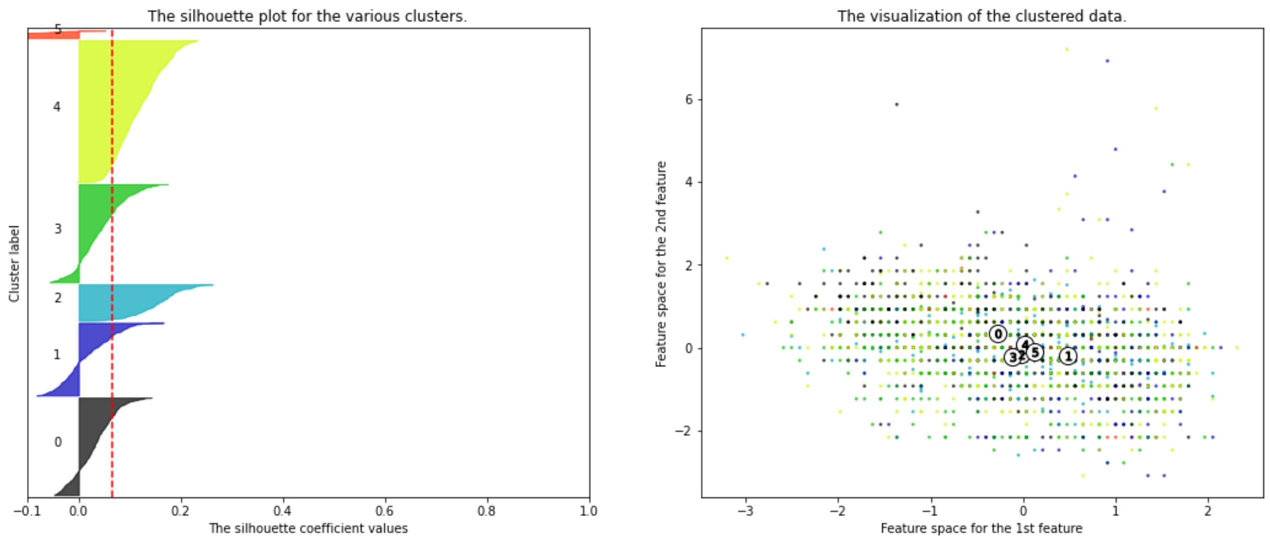

Silhouette analysis for KMeans clustering on sample data with n_clusters = 3
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

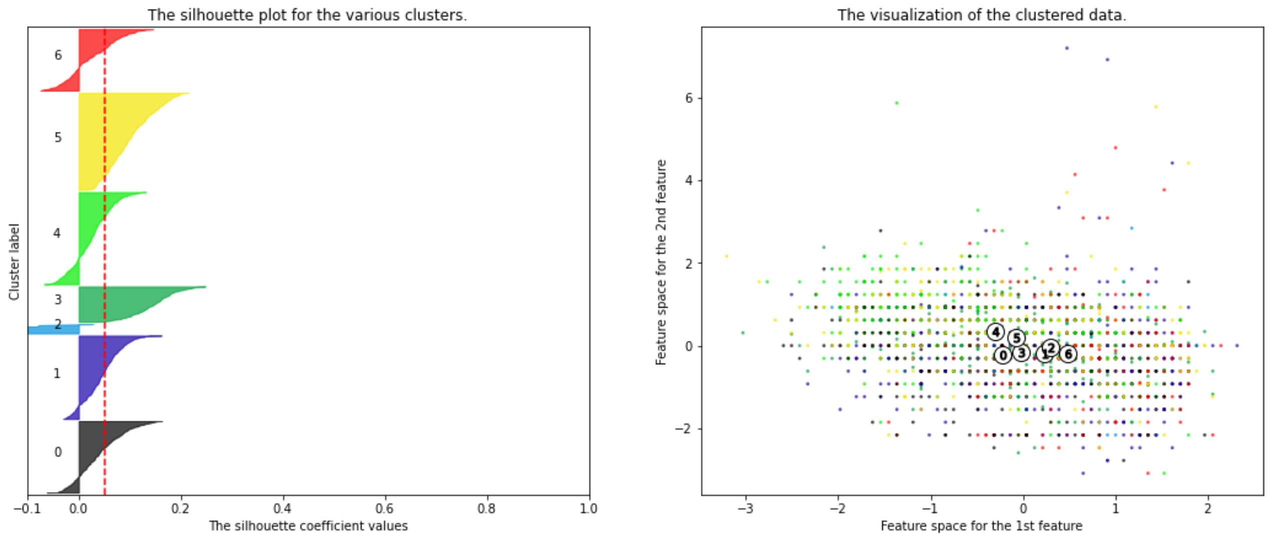
Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

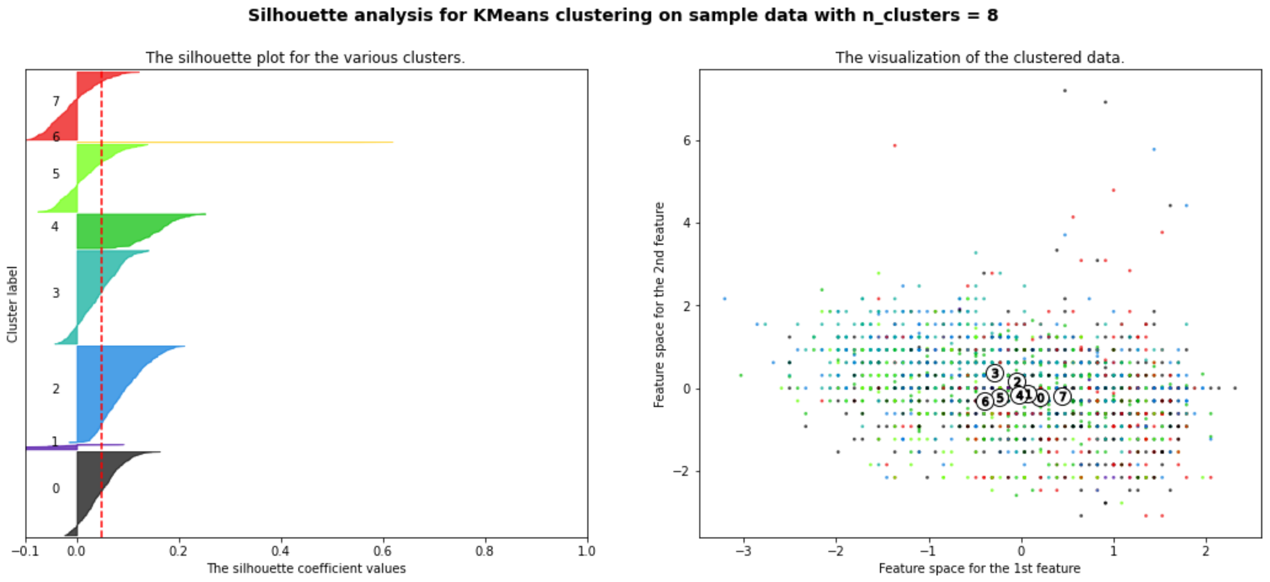


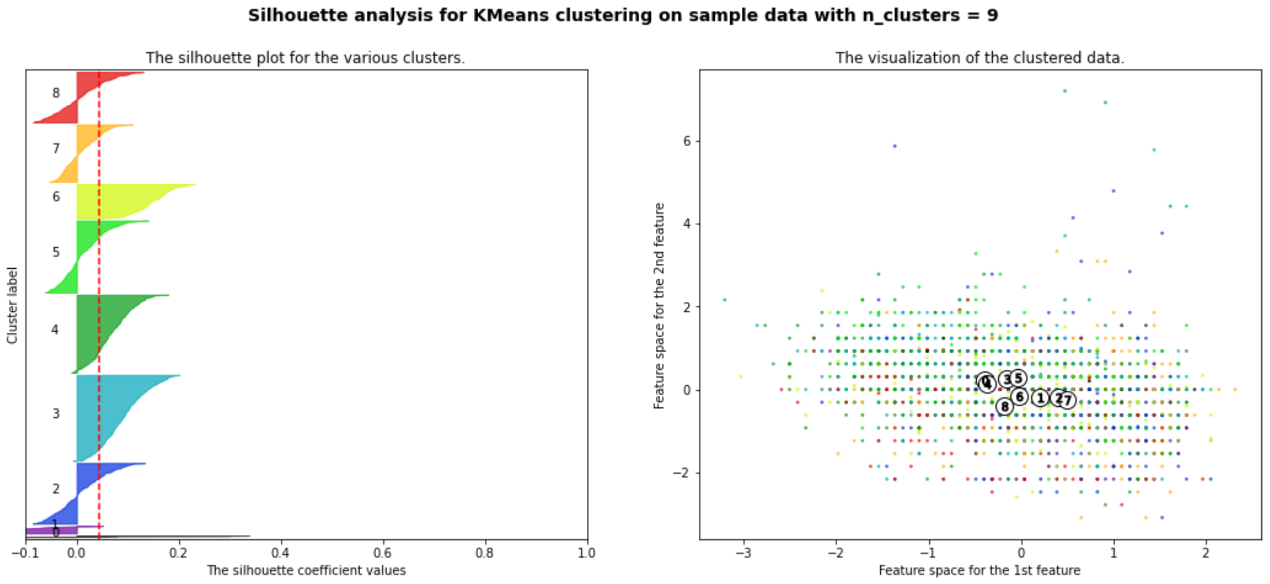
Silhouette analysis for KMeans clustering on sample data with n_clusters = 6

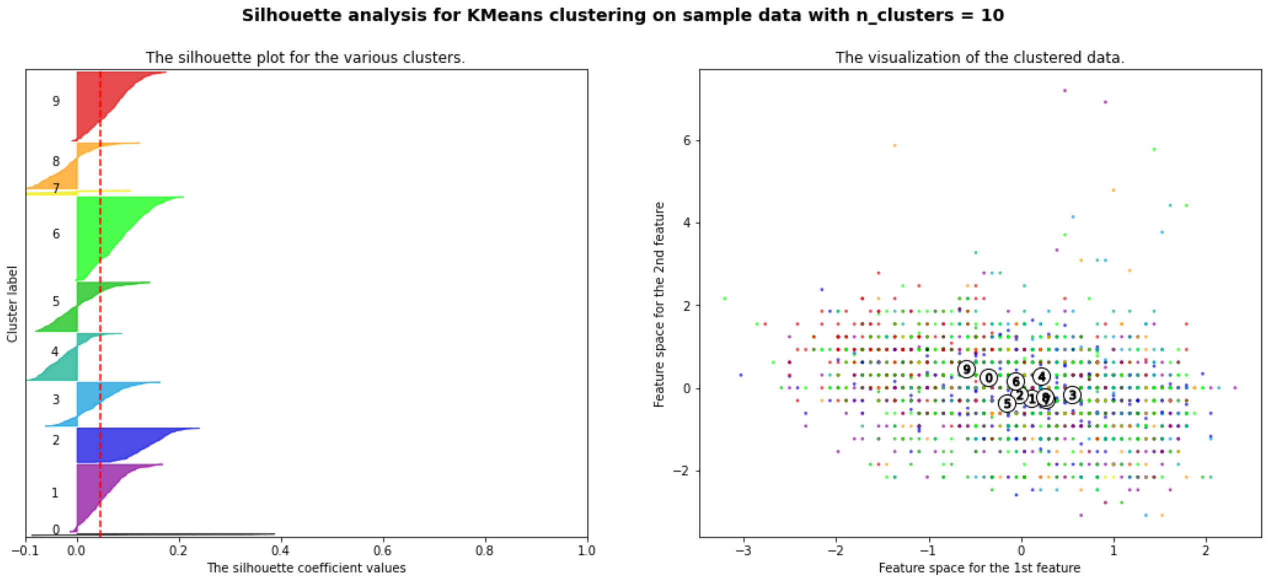


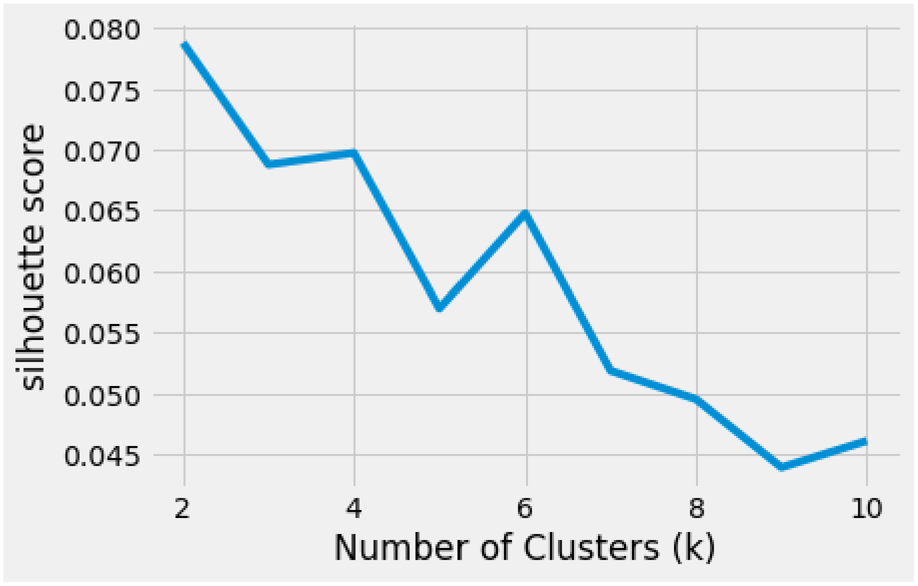
Silhouette analysis for KMeans clustering on sample data with n_clusters = 7




Silhouette analysis for KMeans clustering on sample data with n_clusters = 8
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 9
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 10
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature



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

Observation

1. None of the silhouettes have clusters that are below-average silhouette scores, so none of the cluster values can be considered bad picks for the given dataset.
2. However, plotting the average silhouette scores for each k shows that the best choice for k is 2 since it has the maximum score (0.07878005888570402). This can also be validated in the Elbow method plot above.

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

In [40]:
```
## Using the cluster value as 2
kmeans = KMeans(init="random",n_clusters=2, n_init=10, max_iter=300, random_state=42)
```

In [41]:
```
## Fit the model to the data with cluster value as 2
kmeans.fit(X)
```

Out[41]:
```
KMeans(init='random', n_clusters=2, random_state=42)
```

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

In [43]:
```
## Import PCA library
from sklearn.decomposition import PCA
```

In [44]:
```
## Reference 1: https://365datascience.com/tutorials/python-tutorials/pca-k-means/
## Reference 2: https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e
## Initialize PCA for 2 features
pca_als = PCA(n_components=2)
pca_als.fit(X)
```

Out[44]:
```
PCA(n_components=2)
```

In [45]:

```
# Transform PCA
pca_als.transform(X)
```

Out[45]:

```
array([[-1.4267157 , -2.31976744],
       [-1.44023996, -4.87137188],
       [ 1.6178652 , -0.42935149],
       ...,
       [-0.43290564,  4.2452173 ],
       [-0.330793  ,  3.31718127],
       [ 1.46800221,  0.58155086]])
```

In [49]:

```
# Creating a variable called score to store transformed PCA
score_pca_als = pca_als.transform(X)
```

In [51]:

```
# Create a new dataframe with the original features and add the PCA scores and assigned
df_segm_pca_kmeans = pd.concat([als_df.reset_index(drop=True), pd.DataFrame(score_pca_a
df_segm_pca_kmeans.columns.values[-2: ] = ['PCA_1', 'PCA_2']
```

In [52]:

```
# The last column we addd contains the pca k-means clustering labels
df_segm_pca_kmeans['Segment K-means PCA'] = kmeans.labels_
```

In [56]:

```
# Add the names of the segments to the labels
df_segm_pca_kmeans['Segment'] = df_segm_pca_kmeans['Segment K-means PCA'].map({0:'first
```

In [57]:

```
## Show few records from the dataframe
df_segm_pca_kmeans.head()
```

Out[57]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Tot
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	

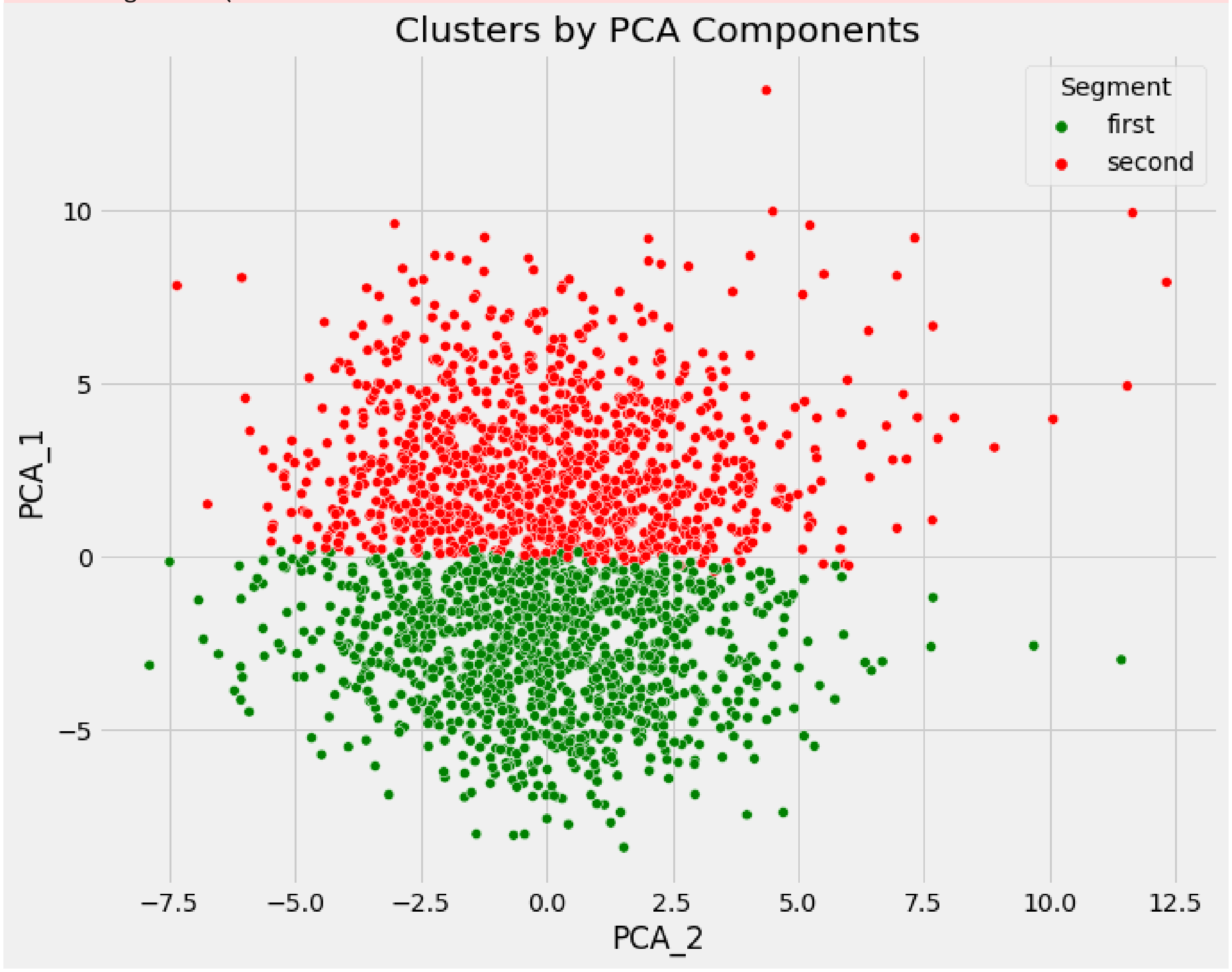
7. Make a scatterplot of the PCA transformed data coloring each point by its cluster value.

In [58]:

```
# Plot scatterplot of PCA transformed data coloring each point by its cluster value
x_axis = df_segm_pca_kmeans['PCA_2']
y_axis = df_segm_pca_kmeans['PCA_1']
plt.figure(figsize =(10,8))
sns.scatterplot(x_axis,y_axis, hue=df_segm_pca_kmeans['Segment'], palette = ['g', 'r'])
plt.title('Clusters by PCA Components')
plt.show()
```

```
C:\Users\KesavAdithya\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarni
```

```
ng: Pass the following variables as keyword args: x, y. From version 0.12, the only vali
d positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



In [61]:

```python
## Calculating Eigen values for each components
print('Eigenvalues: {}'.format(pca_als.explained_variance_))
print('Explained Variance Ratio: {}'.format(pca_als.explained_variance_ratio_))
```

```
Eigenvalues: [11.22229079  6.38502006]
Explained Variance Ratio: [0.11330548 0.06446614]
```

8. Summarize your results and make a conclusion.

1. By plotting the results of K-means algorithm with PCA, we see all the cluster clusters are jumbled all together. However, when we employ PCA prior to using K-means we can visually separate almost the entire data set. That was one of the biggest goals of PCA - to reduce the number of variables by combining them into bigger, more meaningful features.
2. There is some overlap between the red and green segments. But, as a whole, both the segments are clearly separated.
3. Here we can see, the first component explained 11.33% variance and the second component explained 6.4% variance
4. Looking at Eigenvalues, we see Pricipal Compenent 1 is ~64% (11.22/17.7) and PC2 is about ~36% (6.3/17.6). This means that the Y axis accounts for ~64% of the variation in the dataset, and X axis accounts for the remaining ~36%.

In [ ]: