DSC630 Week4 - Assignment

Guruprasad Velikadu Krishnamoorthy

Solution using Python

In [1]:
```
# Importing the Required libraries
import pandas as pd
import numpy as np
import os
import sys
import re
from datetime import datetime
# Importing the required packages for plotting graphs
import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import chart_studio.plotly as py
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
# Importing libraries for model building and standardization
from sklearn import preprocessing
from scipy.spatial.distance import cdist
from sklearn.cluster import KMeans
import sklearn.metrics as metrics
from sklearn.metrics import silhouette_score
# Importing libraries for plotting Silhouette plot
from yellowbrick.cluster import InterclusterDistance
from yellowbrick.cluster import SilhouetteVisualizer
```

In [2]:
```
# Setting global options for the notebook such as maxrows
pd.set_option('display.max_columns', 50)
pd.set_option('display.max_colwidth', None)
pd.set_option("display.max_rows", 100)
```

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Importing the Dataset
path=os.getcwd()
# Assigning a path for the file
als_file_path=path+"\\als_data.csv"
```

In [4]:
```
# Loading the source file into Pandas DataFrame
als_df_orig=pd.read_csv(als_file_path)
# Printing the shape of the dataframe
als_df_orig.shape
```

Out[4]:
```
(2223, 101)
```

In [5]:
```
# Making a copy of the original Dataframe
als_df=als_df_orig.copy()
# Printing top 5 rows of the Dataframe
als_df.head()
```

Out[5]:

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Tota
0	1	65	57.0	40.5	38.0	0.066202	-0.965608	30	
1	2	48	45.0	41.0	39.0	0.010453	-0.921717	37	
2	3	38	50.0	47.0	45.0	0.008929	-0.914787	24	
3	4	63	47.0	44.0	41.0	0.012111	-0.598361	30	
4	5	63	47.0	45.5	42.0	0.008292	-0.444039	32	

5 rows × 101 columns

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

Solution: Upon inspecting all the columns in the dataset, the unique Identifier columns does not have any information related to ALS, hence they can be removed.

In [6]:
```
# Dropping the Unique ID columns such as ID and SubjectId
als_df1=als_df.drop(["ID","SubjectID"],axis=1)
```

Building a correlation Matrix and identifying the redundant columns that are highly correlated.

In [7]:
```
# Creating correlation Matrix
corr_df=als_df1.corr().abs()
# Creating  a Mask to be applied on the correlation matrix
mask=np.triu(np.ones_like(corr_df,dtype=bool))
mask_corr_df=corr_df.mask(mask)
mask_corr_df.head()
```

Out[7]:

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max
Age_mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Albumin_max	0.276195	NaN	NaN	NaN	NaN	NaN	NaN
Albumin_median	0.349024	0.780141	NaN	NaN	NaN	NaN	NaN
Albumin_min	0.297121	0.596662	0.761269	NaN	NaN	NaN	NaN
Albumin_range	0.053197	0.223350	0.091822	0.369015	NaN	NaN	NaN

5 rows × 99 columns

In [8]:
```
# Redundant Columns with high correlation values are selected to be dropped
cols_to_drop=[cols for cols in mask_corr_df.columns if any(mask_corr_df[cols]>0.8)]
print(f"Additional redundant columns dropped are: {cols_to_drop}")
```

```
Additional redundant columns dropped are: ['ALSFRS_slope', 'ALSFRS_Total_max', 'ALSFRS_Total_median', 'ALSFRS_Total_m
in', 'ALSFRS_Total_range', 'ALT.SGPT._max', 'ALT.SGPT._median', 'ALT.SGPT._range', 'AST.SGOT._median', 'bp_systolic_m
ax', 'Creatinine_max', 'Creatinine_median', 'hands_max', 'hands_median', 'hands_min', 'Hematocrit_max', 'Hematocrit_m
edian', 'Hemoglobin_max', 'leg_max', 'leg_median', 'mouth_max', 'mouth_median', 'mouth_min', 'Platelets_max', 'Platel
ets_median', 'respiratory_min', 'trunk_max']
```

In [9]:
```
# Redundant columns with high correlation are dropped
als_df1=als_df.drop(cols_to_drop,axis=1)
```

```
In [10]: # Printing shape of the Dataframe after dropping the columns that are irrelevant or redundant
         als_df1.shape

Out[10]: (2223, 74)
```

2. Apply a standard scalar to the data.

```
In [11]: # Creating a Standard Scaler object
         scaler = preprocessing.StandardScaler()
```

```
In [12]: # Creating a list of all columns in the Dataframe
         all_cols=als_df1.columns
```

```
In [13]: # Fitting the Standard Scaler to the dataframe annd coverting it back to Dataframe
         als_df2 = scaler.fit_transform(als_df1.to_numpy())
         als_df3=pd.DataFrame(als_df2,columns=all_cols)
```

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

```
In [14]: # Creating a list of range of clusters
         sil_K = range(2,13)
         # Creating an empty list to store the results of the silhoutte scores
         sil_score = []
         # Looping through each cluster and calculating the silhoutte scores
         for var in sil_K:
             # Calculating the cluster number for each loop and storing in the variable labels
             labels=KMeans(n_clusters=var,init="k-means++",random_state=200).fit(als_df1).labels_
             # Computing the silhoutte score for each loop
             score = metrics.silhouette_score(als_df1,labels,metric="euclidean",sample_size=500,random_state=200)
             # Appending the results to the list
             sil_score.append(score)
             # Printing the results of the scores for each cluster
             print ("Silhouette score for k(clusters) = "+str(var)+" is "
                    +str(metrics.silhouette_score(als_df1,labels,metric="euclidean",sample_size=500,random_state=200)))
```

```
Silhouette score for k(clusters) = 2 is 0.6400626003583285
Silhouette score for k(clusters) = 3 is 0.5702678431677052
Silhouette score for k(clusters) = 4 is 0.5898027074416125
Silhouette score for k(clusters) = 5 is 0.5547243767479048
Silhouette score for k(clusters) = 6 is 0.519686805984683
Silhouette score for k(clusters) = 7 is 0.5500370841521615
Silhouette score for k(clusters) = 8 is 0.5440610729644003
Silhouette score for k(clusters) = 9 is 0.5463227944857818
Silhouette score for k(clusters) = 10 is 0.5592415612886305
Silhouette score for k(clusters) = 11 is 0.5546268388742226
Silhouette score for k(clusters) = 12 is 0.5500947737008739
```

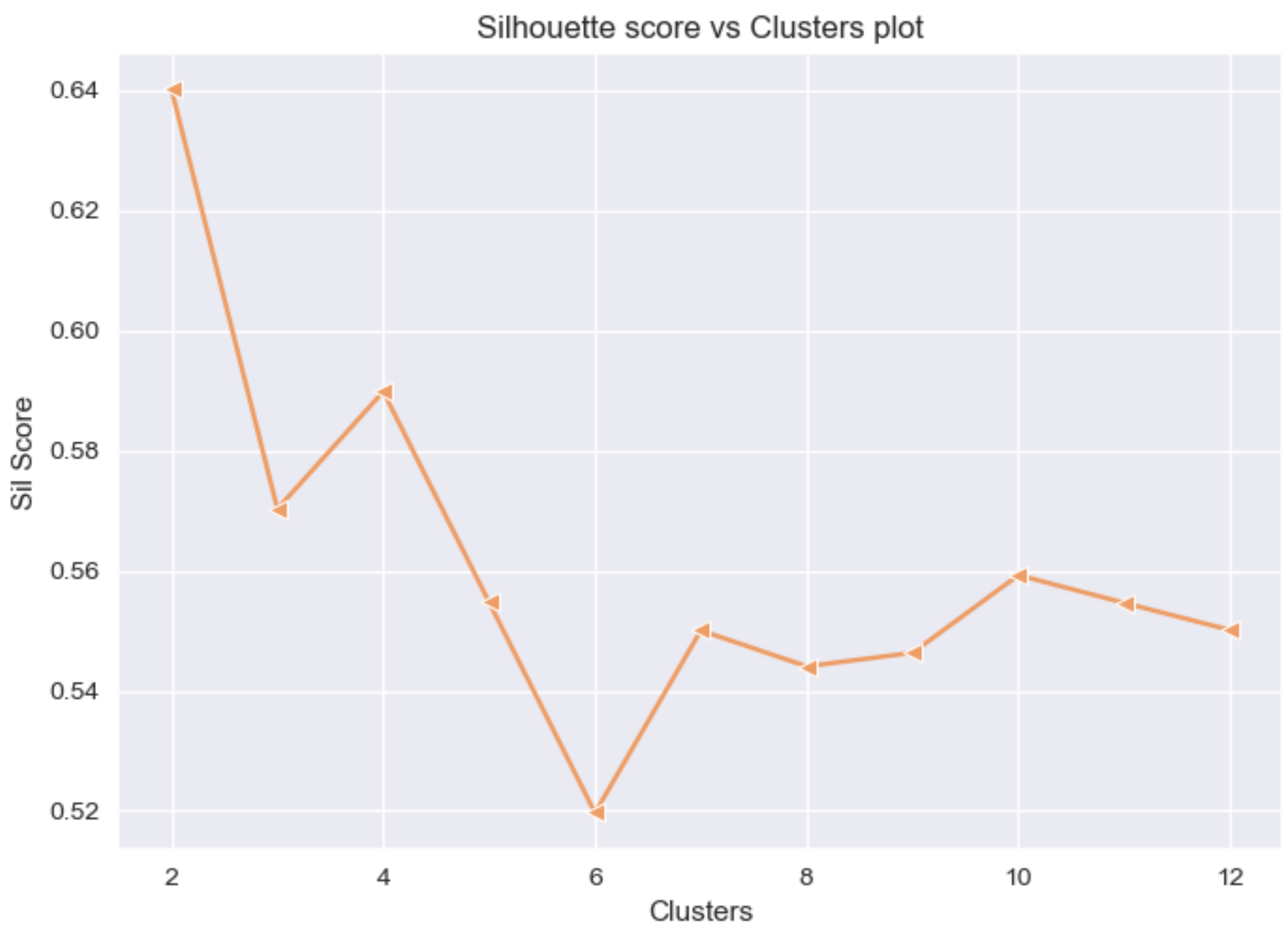
In [15]:
```
# Creating a dataframe with silhoutte score data
sil_df = pd.DataFrame({'Clusters' : sil_K, 'Sil Score' : sil_score})
sil_df
```

Out[15]:

	Clusters	Sil Score
0	2	0.640063
1	3	0.570268
2	4	0.589803
3	5	0.554724
4	6	0.519687
5	7	0.550037
6	8	0.544061
7	9	0.546323
8	10	0.559242
9	11	0.554627
10	12	0.550095

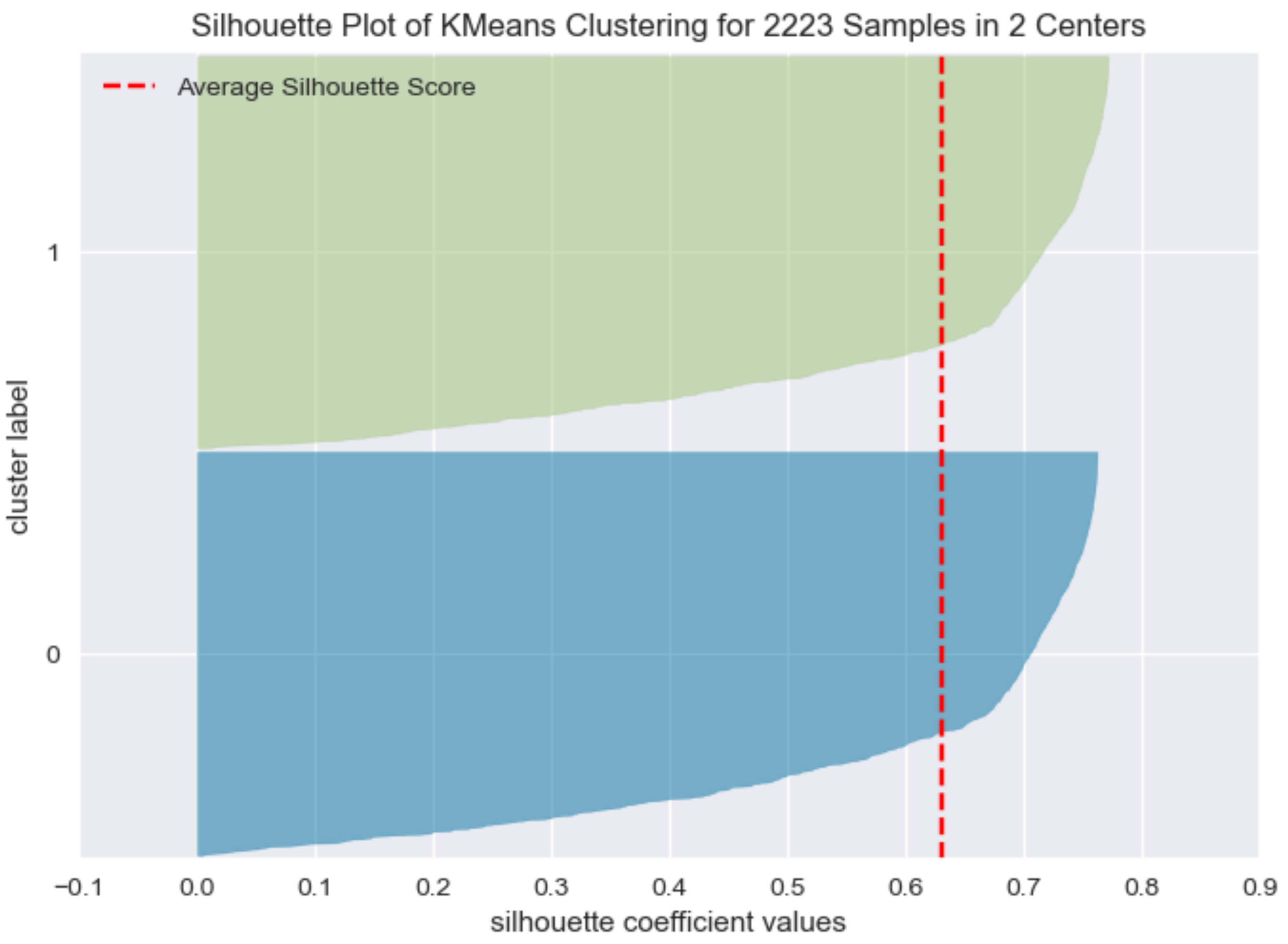
In [16]:
```
# Plotting Lineplot of the Silhouette score vs Clusters
sns.set_style("darkgrid")
sns.lineplot(x = 'Clusters', y = 'Sil Score', data = sil_df, marker="<",color="#EE9D65")
```

```
plt.title('Silhouette score vs Clusters plot')
plt.show()
```

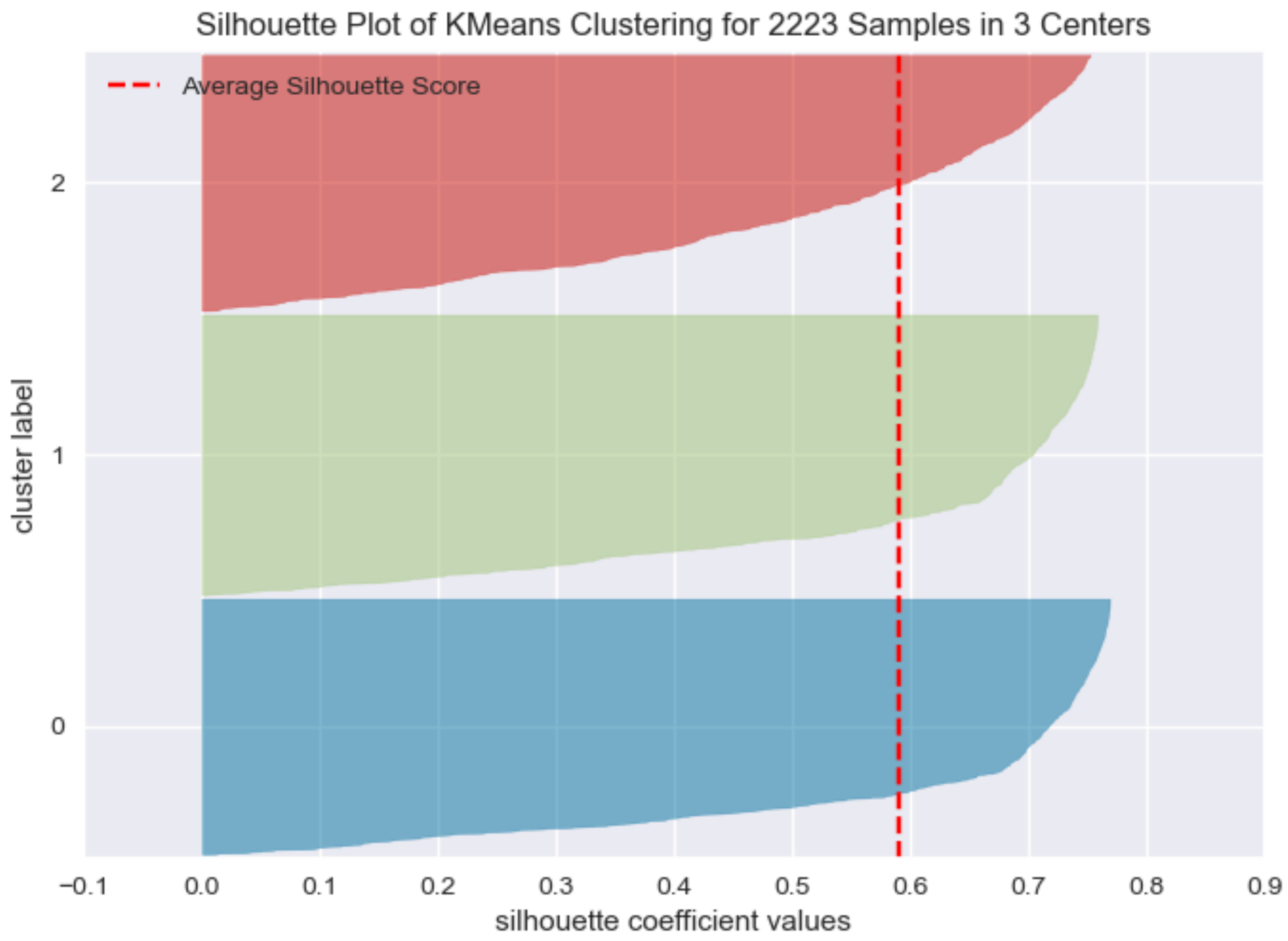


In [17]:
```
def create_silhouette_visuals(numb):
    """
    Function to create visualization of the silhoutte plots
    """
    # Creating a model with the number of cluster as input
    model_sil=KMeans(numb,random_state=42)
    # Creating visualization and plotting it
    visualizer = SilhouetteVisualizer(model_sil, colors='yellowbrick')
    visualizer.fit(als_df1)
    visualizer.show()
```

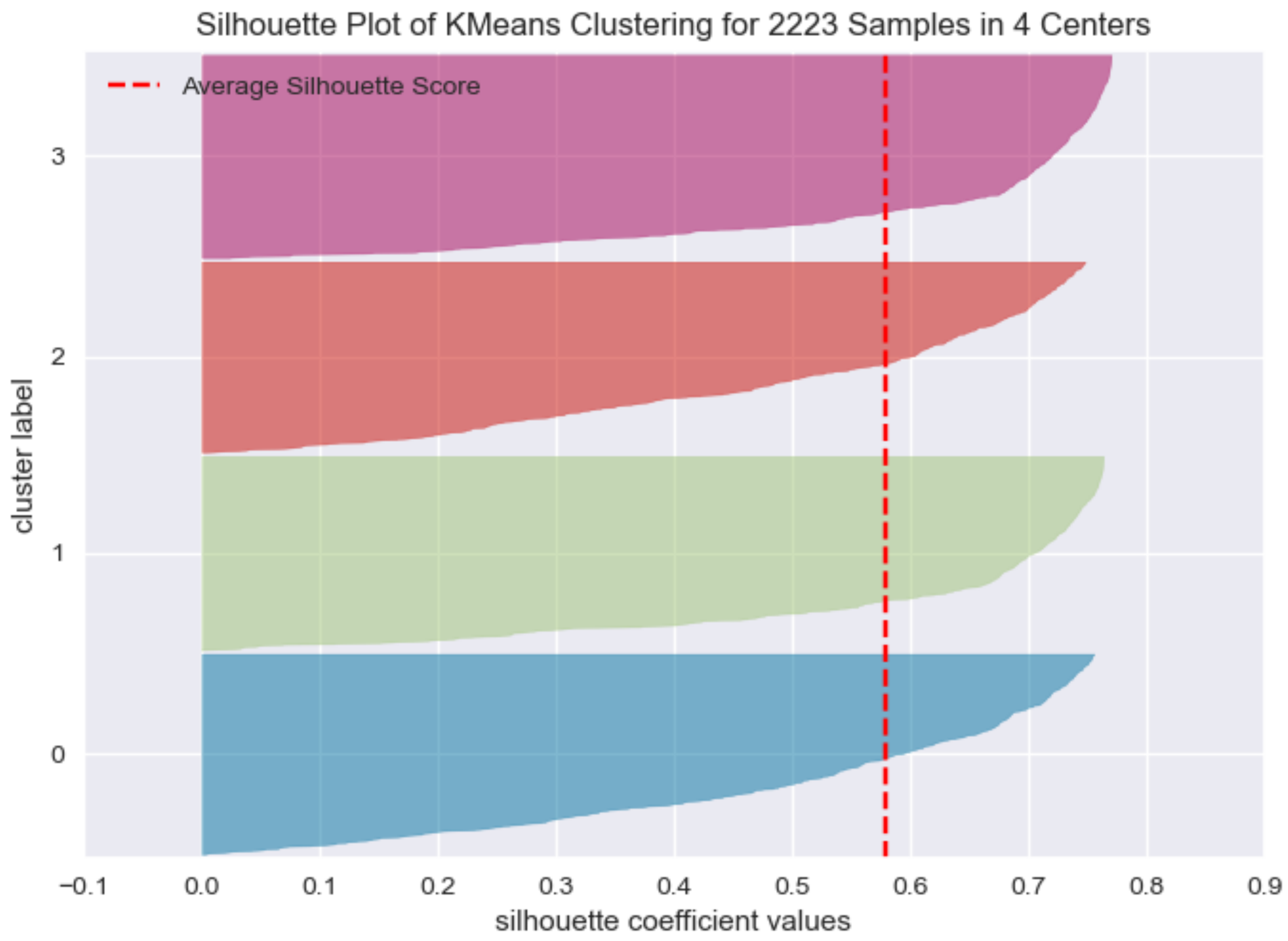
In [18]:
```
# Create silhoutte Plot for 2 clusters
create_silhouette_visuals(2)
```



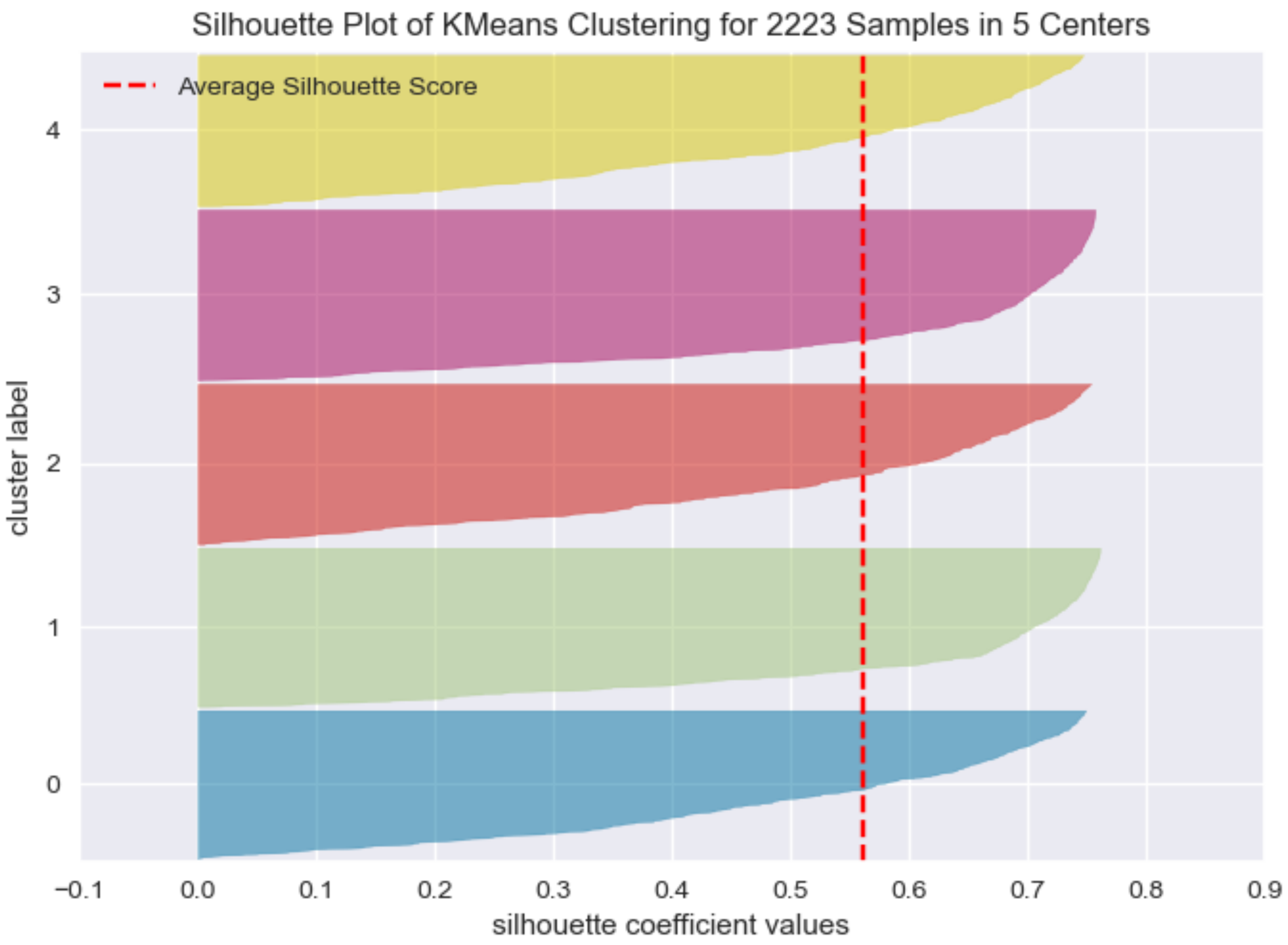
In [19]:
```
# Create silhoutte Plot for 3 clusters
create_silhouette_visuals(3)
```



```
In [20]: # Create silhoutte Plot for 4 clusters
         create_silhouette_visuals(4)
```



In [21]:
```
# Create silhoutte Plot for 5 clusters
create_silhouette_visuals(5)
```



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

The silhoutte plot helps us analyze how close are the points in one clusters are to the points in the neighboring clusters, thus enabling us to identify the optimal number of clusters. Silhoutte coefficient of 1 indicates that the clusters are well spaced and value of 0 indicates that they are very close to the decision boundry.

The silhoutte plot indicates that none of the clusters are below the average silhoutte score (indicated in red line) , which is a good indication that data is fairly distributed among the clusters. Another criteria to look at the Silhoutte plot is the thickness of the clusters. In the clusters 2 and 4, the thickness appears to be even across all clusters. Hence 2 and 4 clusters can be a good choice. Though 3 and 5 clusters have almost even sized clusters,2 and 4 appears to be a better choice.

Looking at the line plot of the Silhoutte score vs clusters, the results are pretty similar. Score is highest with 2 clusters and then drops a bit for 3 clusters and then increases for 4 clusters. ***Hence 2 or 4 clusters can be a better choice for this dataset***

Plotting Elbow Plot to validate the results

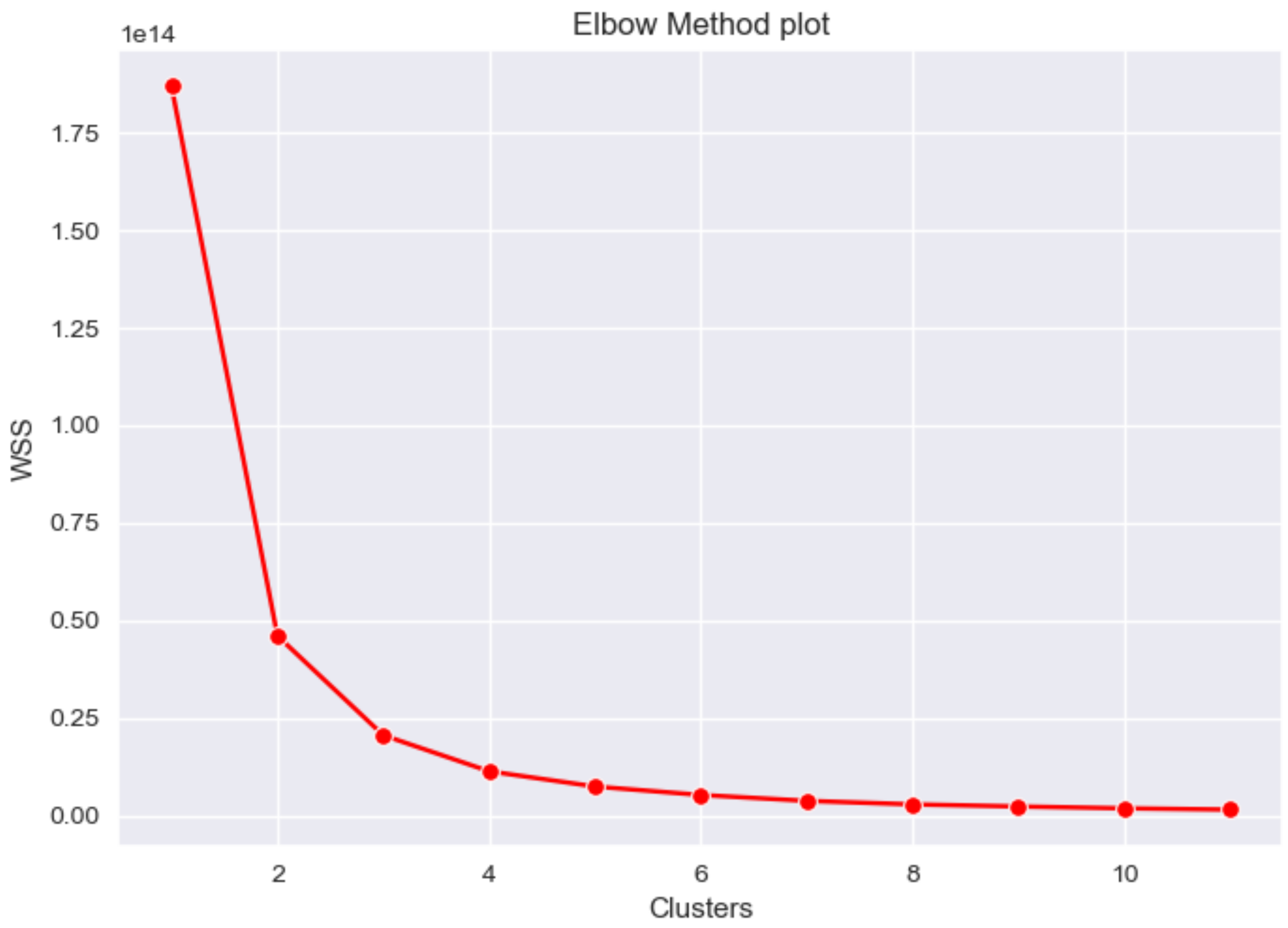
In [22]:
```python
# Creating Elbow plot to validate the results
num_clusters_elbow = range(1,12)
# Creating an empty list to store the WSS(Within Cluster Sum of Squared Errors)
wss_list = []
# Looping through each cluster and calculate the WSS values
for num_cluster in num_clusters_elbow:
    # Creating a model for each cluster and fitting model to als_df1
    kmeans=KMeans(n_clusters=num_cluster,init="k-means++")
    kmeans=kmeans.fit(als_df1)
    # Calculating the WSS values
    wss_inertia = kmeans.inertia_
    # Storing the results for each cluster to the list
    wss_list.append(wss_inertia)
```

In [23]:
```python
# Creating a Dataframe to store the results
elbow_df = pd.DataFrame({'Clusters' : num_clusters_elbow, 'WSS' : wss_list})
elbow_df
```

Out[23]:

	Clusters	WSS
0	1	1.867731e+14
1	2	4.600728e+13
2	3	2.052967e+13
3	4	1.129482e+13
4	5	7.391367e+12
5	6	5.229799e+12
6	7	3.717821e+12
7	8	2.817533e+12
8	9	2.280449e+12
9	10	1.812543e+12
10	11	1.483200e+12

In [24]:
```
# Plotting a lineplot of the dataframe to show the results
sns.lineplot(x = 'Clusters', y = 'WSS', data = elbow_df, marker="o",color="red")
plt.title('Elbow Method plot')
plt.show()
```



According to the elbow plot, 2 or 3 clusters can be an ideal choice for the number of clusters.

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

In [25]:
```
# Creating the model with 2 clusters
kmeans = KMeans(n_clusters=2,random_state=42)
model_2=kmeans.fit(als_df1)
model_2
```

Out[25]:
```
KMeans(n_clusters=2, random_state=42)
```

In [26]:
```
# Creating the model with 4 clusters
kmeans = KMeans(n_clusters=4,random_state=42)
model_4=kmeans.fit(als_df1)
model_4
```

Out[26]:
```
KMeans(n_clusters=4, random_state=42)
```

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

In [27]:
```
from sklearn.decomposition import PCA
# Set the n_components=2 to only retain 2 features
principal=PCA(n_components=2)
# Fitting and transforming the PCA object on the standardized data
principal.fit(als_df3)
# Creating a PCA transformed object with only 2 features
als_df4=principal.transform(als_df3)
```

In [28]:
```
# Convering als_df4 as dataframe
als_df4=pd.DataFrame(als_df4,columns=["col1","col2"])
# Check the dimensions of dataframe before PCA
print(f"Shape of the Dataframe Before PCA: {als_df3.shape}")
# Check the dimensions of dataframe after PCA
print(f"Shape of the Dataframe After PCA: {als_df4.shape}")
als_df4.head()
```

```
Shape of the Dataframe Before PCA: (2223, 74)
Shape of the Dataframe After PCA: (2223, 2)
```

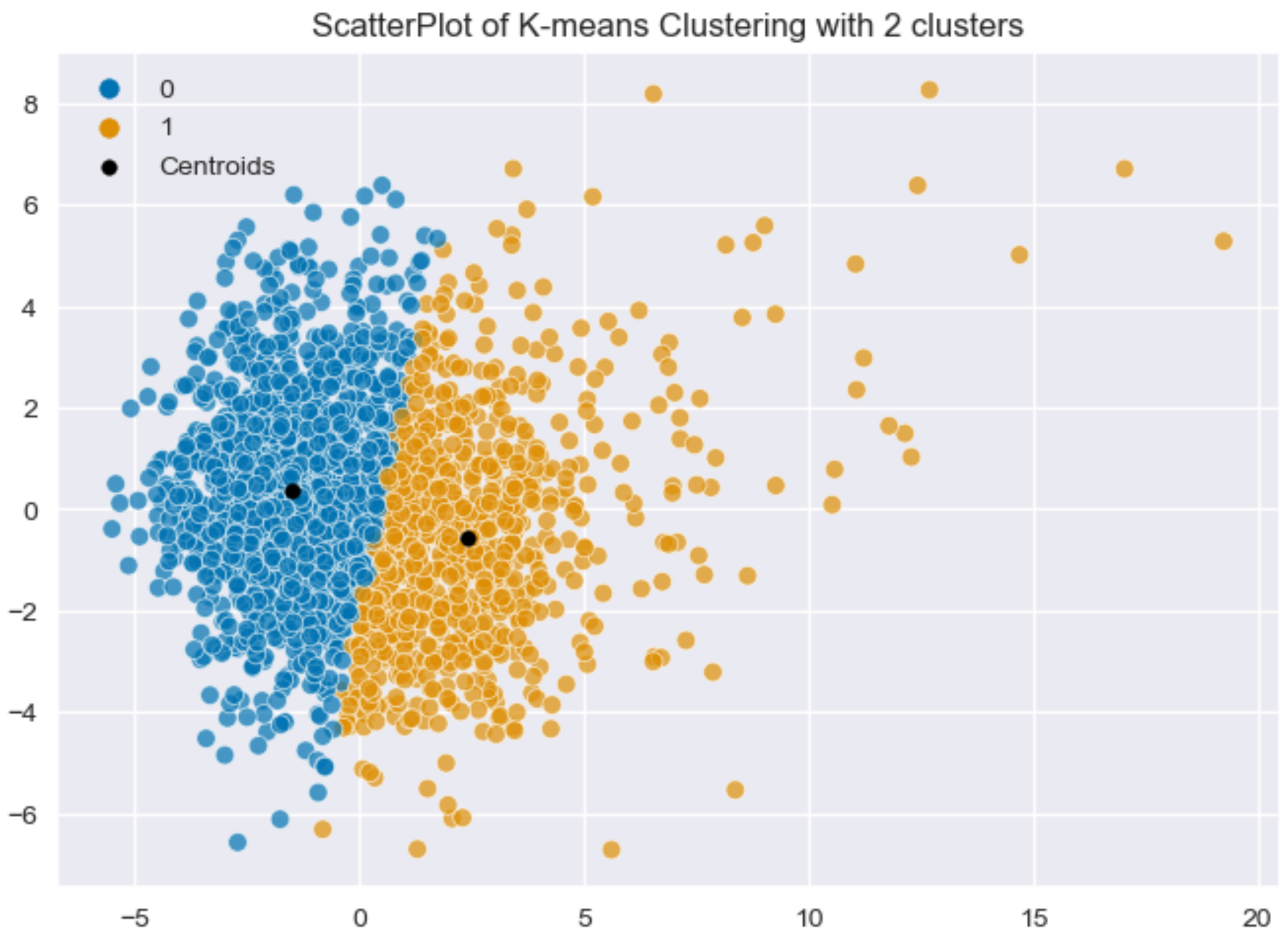
Out[28]:

	col1	col2
0	-0.685005	-0.086499
1	-0.815710	-3.410230
2	-2.297148	-1.335892
3	-2.493997	0.877613
4	0.131041	-0.019624

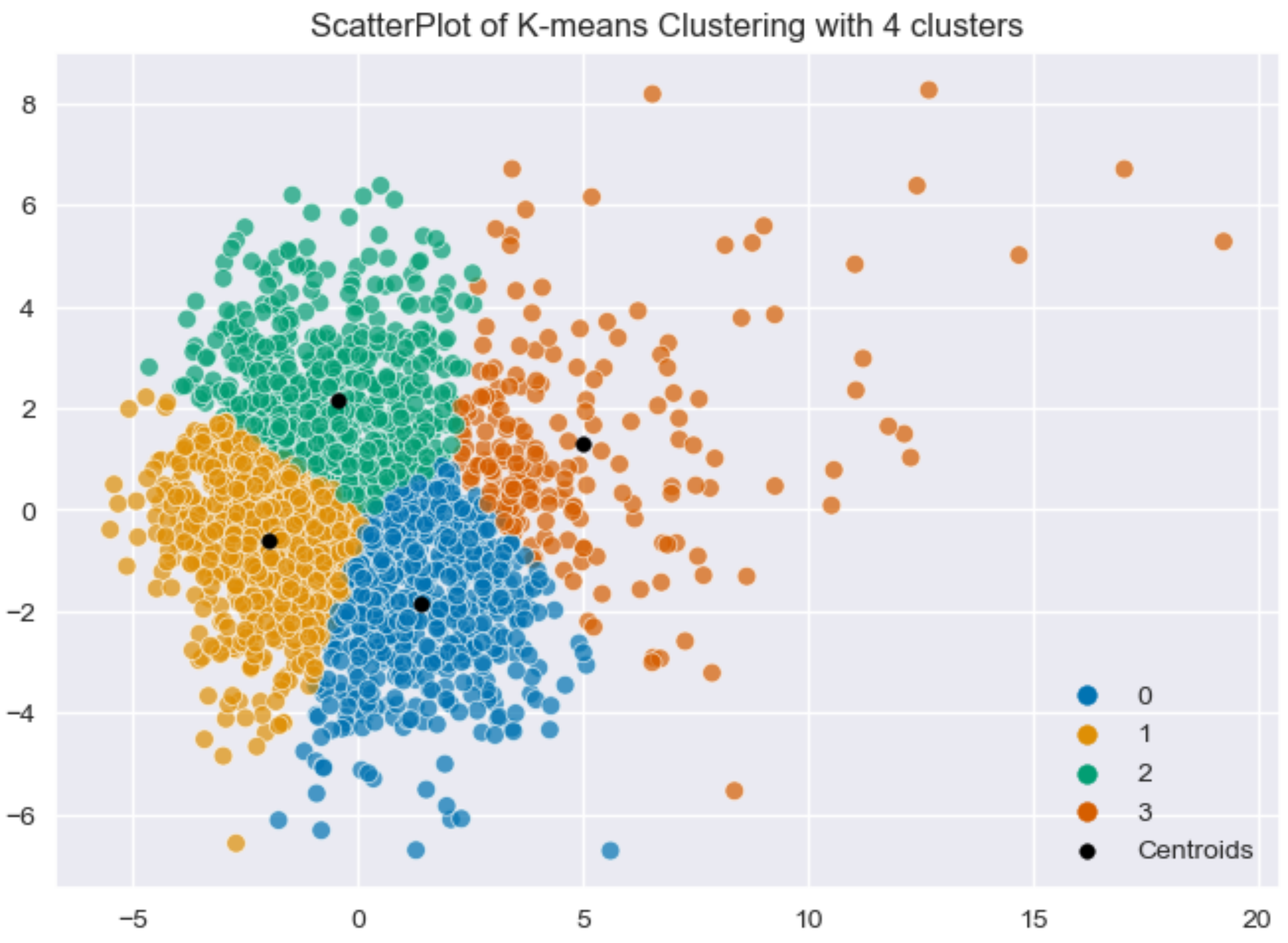
7. Make a scatterplot the PCA transformed data coloring each point by its cluster value.

In [29]:
```
def create_scatterplot(num):
    """
    This function takes the number of cluster as input and plots Scatterplot of the K means cluster
    """
    # Creating a kmeans model object with input as number of clusters
    kmeans = KMeans(n_clusters=num,random_state=42)
    # Creating cluster labels using the fit_predit function on the PCA converted data
    cluster_label = kmeans.fit_predict(als_df4)
    # Adding a new column to the Dataframe with the cluster details
    als_df4['cluster'] = cluster_label
    # Computing the centroids of each cluster
    centroids = kmeans.cluster_centers_
    # Creating Scatter plot of all datapoints in Dataframe seperated by clusters
    sns.scatterplot(x="col1",y="col2",data=als_df4,hue="cluster",palette =
"colorblind",alpha=0.7).set(title=f"ScatterPlot of K-means Clustering with {num} clusters",xlabel="",ylabel="")
    # Plotting Centroids on the scatterplot
    plt.scatter(centroids[:,0] , centroids[:,1] , s = 30, color = "black",label="Centroids")
    plt.legend()
```

In [30]:
```
# Creating Scatter Plot for 2 Clusters
create_scatterplot(2)
```



```
In [31]: # Creating Scatter Plot for 4 Clusters
         create_scatterplot(4)
```



8. Summarize your results and make a conclusion.

The results indicate that either 2 or 4 clusters can be a better choice for the dataset to identify the clusters in the ALS dataset. While using the PCA, the number of features were reduced to 2 that contains most of the information contained in the original features.

Week4_Assignment_Using_R

```r
# Calling the Libraries used
library(readxl)
library(dplyr)
library(lubridate)
library(readr)
library(ggplot2)
library(ggthemes)
library(tidyr)
library(DT)
library(scales)
library(stringr)
library(NbClust)
library(knitr)
library(NbClust)
library(cluster)
library(factoextra)
library(useful)
library(FactoMineR)
library(ggpubr)
library(kableExtra)
library(magrittr)
library(ggfortify)
```

```r
# Reading the data from CSV file and loading into a
# Dataframe
als_data_orig_df <- read.csv("als_data.csv")
# Printing the Dimensions of the Dataframe
dim(als_data_orig_df)
```

```
## [1] 2223  101
```

```r
# Validating the Dataframe to see if there are nulls in
# any of the columns
sum(is.na(als_data_orig_df))
```

```
## [1] 0
```

```r
# Checking for nulls by checking each column and getting
# the sum
sum(colSums(is.na(als_data_orig_df)) > 0)
```

```
## [1] 0
```

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

```
# Removing the Unique identifier columns that may not be
# required for clustering as they don't add much purpose
cols_to_drop1 <- c("ID", "SubjectID")
```

```
# Creating a correlation matrix and removing the columns
# that have correlation > 0.9
corr_matrix <- cor(als_data_orig_df)
# setting the threshold to 0.9
threshold <- 0.8
# Creating list of columns that have threshold over 0.9
# to be removed
cols_to_drop2 <- names(als_data_orig_df[apply(abs(corr_matrix) >=
    threshold & abs(corr_matrix) < 1, 1, any, na.rm = TRUE) ==
    1])
cols_to_drop <- unique(append(cols_to_drop1, cols_to_drop2))
cols_to_drop
```

```
##  [1] "ID"                  "SubjectID"           "ALSFRS_slope"
##  [4] "ALSFRS_Total_max"    "ALSFRS_Total_median" "ALSFRS_Total_min"
##  [7] "ALSFRS_Total_range"  "ALT.SGPT._max"       "ALT.SGPT._median"
## [10] "ALT.SGPT._min"       "ALT.SGPT._range"     "AST.SGOT._max"
## [13] "AST.SGOT._median"    "AST.SGOT._min"       "AST.SGOT._range"
## [16] "bp_systolic_max"     "bp_systolic_median"  "Creatinine_max"
## [19] "Creatinine_median"   "Creatinine_min"      "hands_max"
## [22] "hands_median"        "hands_min"           "Hematocrit_max"
## [25] "Hematocrit_median"   "Hematocrit_min"      "Hemoglobin_max"
## [28] "Hemoglobin_median"   "leg_max"             "leg_median"
## [31] "leg_min"             "mouth_max"           "mouth_median"
## [34] "mouth_min"           "mouth_range"         "Platelets_max"
## [37] "Platelets_median"    "Platelets_min"       "respiratory_min"
## [40] "respiratory_range"   "trunk_max"           "trunk_median"
## [43] "trunk_min"           "trunk_range"
```

```
# Creating a new dataframe after dropping the columns
als_df1 <- select(als_data_orig_df, -cols_to_drop)
# Printing the dimensions of the new dataframe after
# dropping the columns
dim(als_df1)
```

```
## [1] 2223   57
```

2. **Apply a standard scalar to the data.**

```
# Using the scale function to scale the data
als_df_std <- as.data.frame(scale(als_df1))
# Printing top few rows and 6 columns for the purpose of
# display
kbl(head(als_df_std[1:6, c(1:6)]), caption = "Dataframe with Standard scaler Data",
```

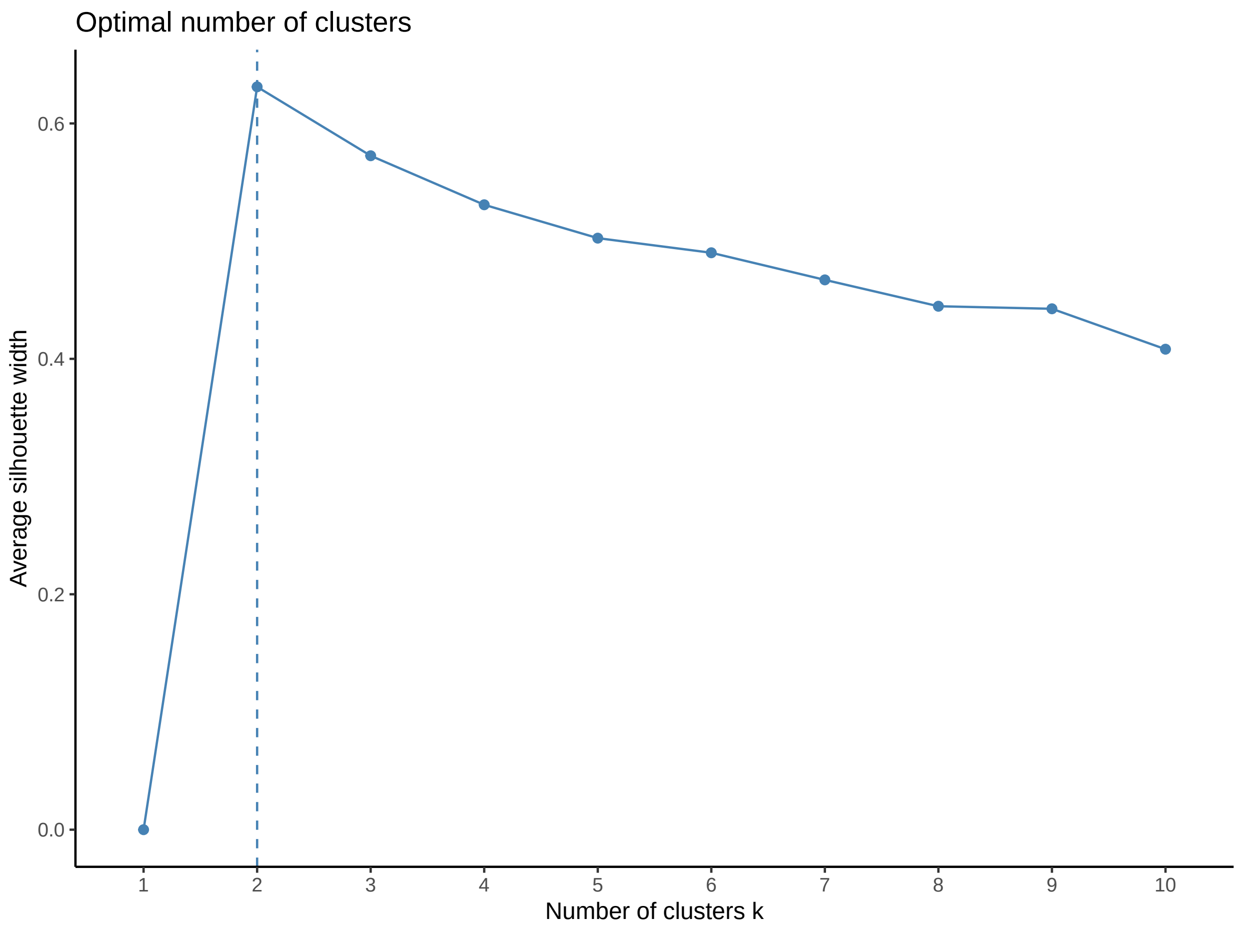
```r
    booktabs = T) %>%
    kable_styling(latex_options = c("striped", "hold_position"))
```

Table 1: Dataframe with Standard scaler Data

Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	Bicarbonate_max
0.9169307	3.0887223	-1.3004884	-0.8663552	5.4796966	-0.2835547
-0.5747494	-0.6218757	-1.1121506	-0.5531786	-0.3476471	0.3484623
-1.4522082	0.9242068	1.1479035	1.3258809	-0.5069886	1.2964878
0.7414389	-0.0034427	0.0178764	0.0731746	-0.1743216	-2.4956143
0.7414389	-0.0034427	0.5828899	0.3863511	-0.5735412	0.3484623
-1.6277000	1.2334233	1.1479035	1.6390574	-0.4934627	-0.5995633

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

```r
# Creating a silhoutte plot using NbClust package to
# identify the ideal number of clusters
fviz_nbclust(als_df1, kmeans, method = "silhouette") + theme_classic()
```

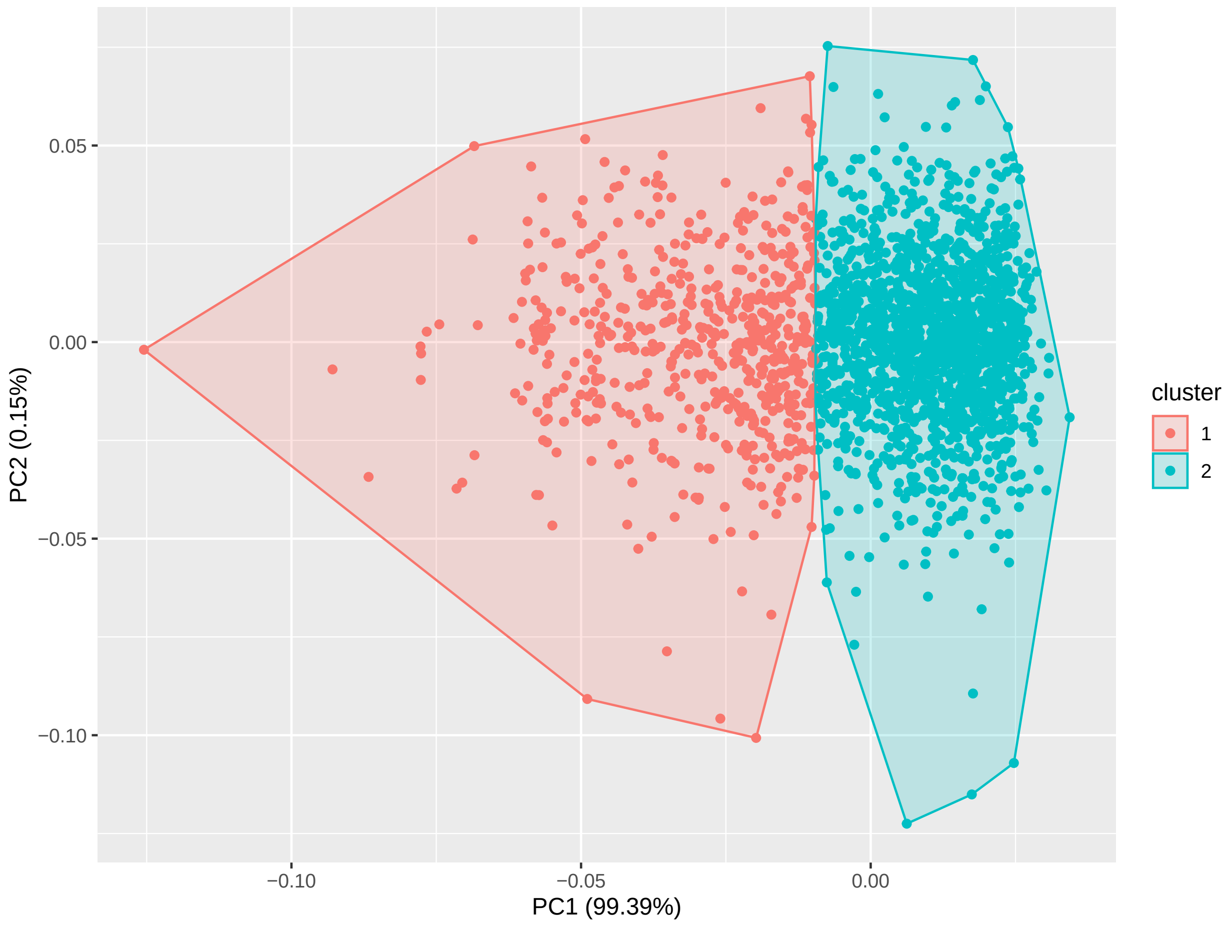


4. Use the plot created in (3) to choose on optimal number of clusters for K-means. Justify your choice.

The plot above indicates that *2 clusters* may be optimal for the dataset as this is also represented by the dotted line in the graph.

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

```r
# Using kmeans function to create a k-means model for the
# Dataframe with 2 clusters(obtained in the previous
# step)
KM = kmeans(als_df1, 2)
# Creating a plot of the clusters
autoplot(KM, als_df1, frame = TRUE)
```



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

```r
# Using PCA function to create Principal component
# Analysis Object from the standardized data with 2
# features
```

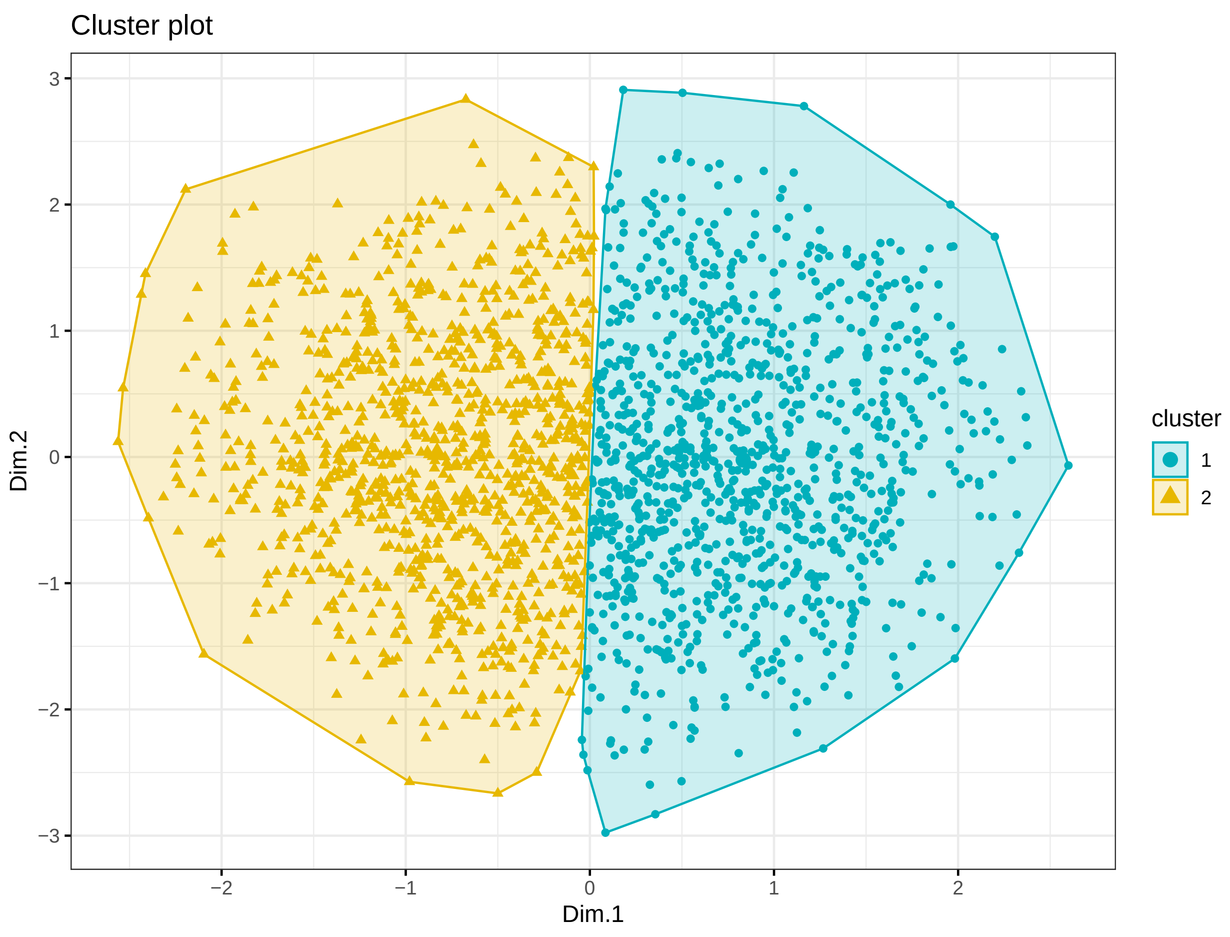
```r
pca_new <- PCA(t(als_df_std), ncp = 2, graph = FALSE)
# Creating a Dataframe of the coordinates of the PCA
# results
pca_df <- as.data.frame(pca_new$var$coord)
# Printing the top few rows from the PCA transformed
# Dataframe
kbl(head(pca_df[1:6, ]), caption = "Dataframe of PCA Transformed data",
    booktabs = T) %>%
    kable_styling(latex_options = c("striped", "hold_position"))
```

Table 2: Dataframe of PCA Transformed data

	Dim.1	Dim.2
V1	0.0629655	-0.4301202
V2	-0.2627420	-0.2481229
V3	0.5283589	-0.0788782
V4	0.3670588	-0.3261947
V5	-0.0051492	0.2411575
V6	0.5258376	-0.0654945

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

```r
# Creating a Kmeans model object from the standardized
# data with 2 clusters
KM_pca = kmeans(pca_df, 2)
# Creating a cluster plot using the fviz_cluster function
# on the PCA Dataframe
fviz_cluster(KM_pca, data = pca_df, palette = c("#00AFBB",
    "#E7B800"), geom = "point", ellipse.type = "convex", ggtheme = theme_bw())
```

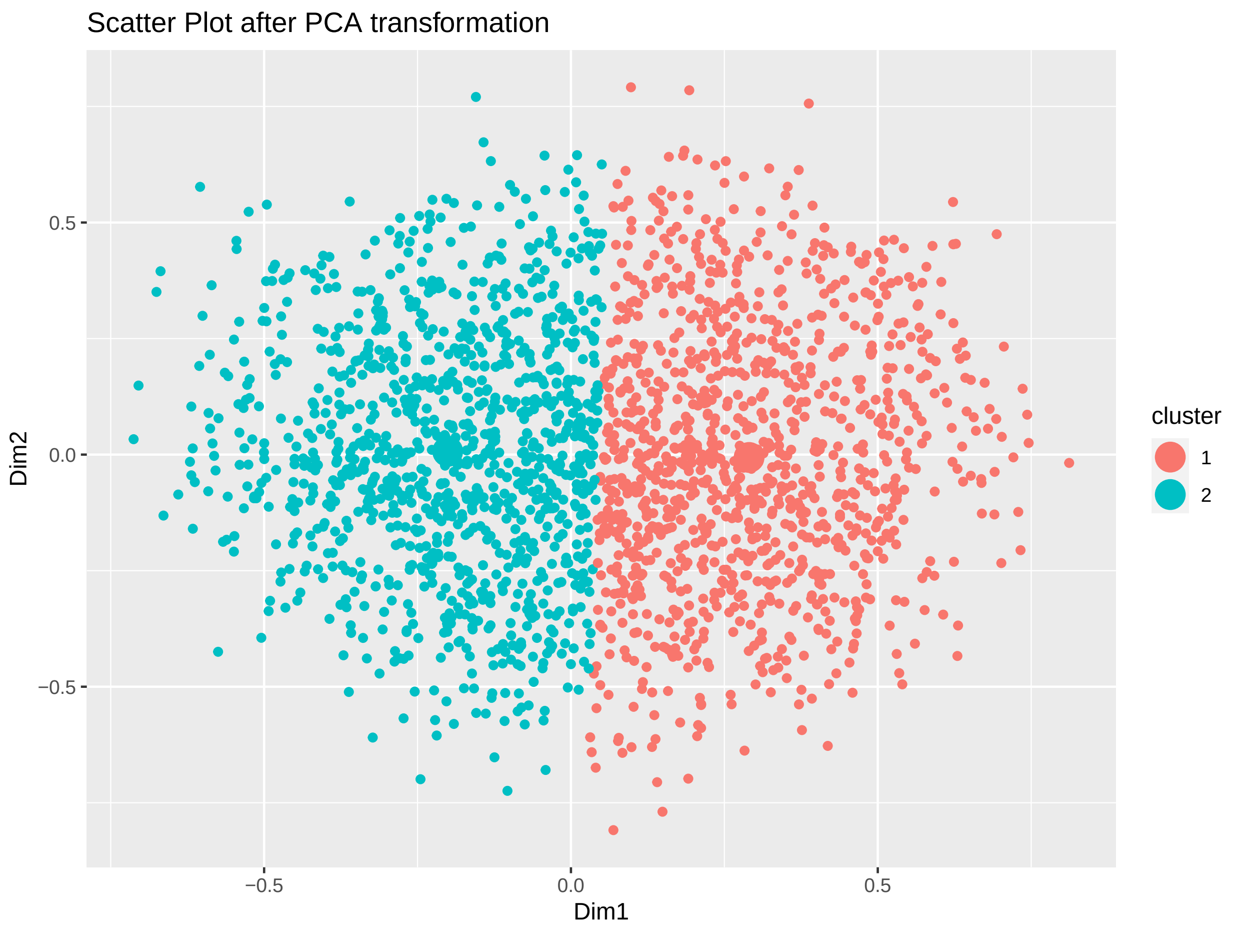


```r
# Another way of plotting Scatter Plot to the data
pca_df$cluster <- KM_pca$cluster
# Adding a new column to the PCA dataframe with cluster
# details
KM_pca$cluster <- as.factor(KM_pca$cluster)
# Printing the top few rows from the PCA transformed
# Dataframe
kbl(head(pca_df[1:6, ]), caption = "Dataframe of PCA Transformed data with Cluster details",
    booktabs = T) %>%
    kable_styling(latex_options = c("striped", "hold_position"))
```

Table 3: Dataframe of PCA Transformed data with Cluster details

	Dim.1	Dim.2	cluster
V1	0.0629655	-0.4301202	1
V2	-0.2627420	-0.2481229	2
V3	0.5283589	-0.0788782	1
V4	0.3670588	-0.3261947	1
V5	-0.0051492	0.2411575	2
V6	0.5258376	-0.0654945	1

```r
# Plotting a scatter plot of the PCA transformed data
# with Clusters and Centroids
ggplot(data = pca_df, aes(x = Dim.1, y = Dim.2)) + geom_point(aes(color = factor(cluster))) +
    scale_color_discrete(name = "cluster") + labs(title = "Scatter Plot after PCA transformation",
    x = "Dim1", y = "Dim2") + stat_mean(aes(color = factor(cluster)),
    size = 6)
```



8. Summarize your results and make a conclusion.

Similar to the results from Python analysis, 2 clusters appears to be a better choice for the dataset and the results are evident from the Scatter plot. The clusters are seperated and there is no overlap in the data between the clusters.