Instructor: Nasir Mahmood Date: 23 November/2020

Student: Fahad Ahmad

Assignment 4

1.Import both the training and testing CSV files.

```
#Import data into a pandas DataFrame as .csv file
ALStrain = pd.read_csv(url,sep=',',index_col=0)
ALStest = pd.read_csv(url1,sep=',',index_col=0)

frame = [ALStrain, ALStest]
ALSdata = pd.concat(frame,ignore_index=True)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API
  import pandas.util.testing as tm
```

- **Peak into the data by random sampling and displaying 8 observations.**

```
ALStrain.sample(8)
```

ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range
1048	40	49.0	46.0	45.0	0.007143	-0.620748	40	35.5	31	0.016071
1058	57	48.0	46.0	43.0	0.009042	0.000000	38	37.0	34	0.007233
1592	62	47.0	44.0	41.0	0.012111	-0.362103	34	31.0	28	0.016575
840	57	47.0	44.0	41.0	0.012111	-1.600877	33	28.0	15	0.048387
1414	54	48.4	46.3	43.1	0.013350	-0.217262	35	31.0	29	0.015385
1617	62	47.0	44.0	41.0	0.012111	-0.109020	36	36.0	35	0.002653
403	52	50.0	43.0	42.0	0.021390	-1.303571	30	27.0	17	0.034759
572	62	49.0	48.0	45.0	0.009281	-0.788580	34	24.0	21	0.023636

8 rows × 100 columns

```
[94] ALStest.sample(8)
```

ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range
12	54.550685	47.0	45.00	42	0.009225	-0.439230	34	19.5	11	0.044402
94	24.605479	45.5	42.25	39	0.013761	-0.458543	25	21.0	19	0.018293
98	59.928767	47.0	44.00	40	0.009333	-0.330616	33	24.0	10	0.030667
52	67.000000	41.0	38.00	35	0.012448	-1.230086	31	21.5	8	0.047521
11	53.934247	46.0	45.00	41	0.010965	-0.434524	19	16.5	13	0.011881
9	65.000000	45.0	42.00	36	0.021327	0.000000	37	37.0	37	0.000000
23	61.000000	45.0	40.00	38	0.017588	-0.598753	39	36.5	33	0.015075
30	51.000000	46.0	43.00	40	0.012959	-1.494152	29	16.0	11	0.038877

8 rows × 130 columns

- **Combine Training and Testing CSV files into a single DataFrame.**

```
#Import data into a pandas DataFrame as .csv file
ALStrain = pd.read_csv(url,sep=',',index_col=0)
ALStest = pd.read_csv(url1,sep=',',index_col=0)

frame = [ALStrain, ALStest]
ALSdata = pd.concat(frame,ignore_index=True)
```

```
[95] ALSdata.sample(8)
```

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range
419	65.0	48.6	45.0	39.0	0.017235	-0.099401	27	27.0	26	0.001825
857	51.0	44.0	40.0	38.0	0.010870	-0.383807	37	31.5	29	0.014493
1589	64.0	46.0	43.0	40.0	0.015707	-0.547062	27	23.5	19	0.020942
664	37.0	50.0	46.5	45.0	0.008913	-0.451286	39	36.0	33	0.010850
308	57.0	49.0	46.0	43.0	0.014118	-1.110097	28	10.0	2	0.051485
1653	56.0	50.8	44.0	40.0	0.029428	-2.124694	35	31.0	13	0.059946
523	32.0	51.0	48.0	45.0	0.010152	-0.215721	29	27.0	17	0.021314
1091	62.0	44.0	42.0	40.0	0.007181	-0.111416	35	34.0	33	0.003591

8 rows × 100 columns

2.Perform basic data exploration, summarization and preliminary visualization on dataset.

- **Verify integrity of the data (missing, null, invalid values)**

  With the merging of the training and test data set, there were NaN values produced as the test dataset had extra variables which were not in the training dataset.

```
+ Code  + Text                                                    RAM Disk   Editing
     memory usage: 2.3 MB

[10] #Can export from Google colab
     #ALSdata.to_csv(r'ALSdata.csv')

[11] #There are NaN values , because the ALS test dataset had new columns and must be removed

     print(ALSdata.isnull().sum())

     Age_mean                          0
     Albumin_max                       0
     Albumin_median                    0
     Albumin_min                       0
     Albumin_range                     0
                                      ...
     Urine.Ph_range                 2223
     White.Blood.Cell..WBC._max     2223
     White.Blood.Cell..WBC._median  2223
     White.Blood.Cell..WBC._min     2223
     White.Blood.Cell..WBC._range   2223
     Length: 130, dtype: int64

[12] #Does seem that there are missing values
     ALSdata.info()
     ALSdata.applymap(np.isreal).all()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 2301 entries, 0 to 2300
```

```
[14] #Drop columns with NaN values #Is this dropping columns with NaN values?
     ALSdata = ALSdata.dropna(axis=1)
```

```
ALSdata.head(5)
```

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range	A
0	65.0	57.0	40.5	38.0	0.066202	-0.965608	30	28.0	22	0.021164	
1	48.0	45.0	41.0	39.0	0.010453	-0.921717	37	33.0	21	0.028725	
2	38.0	50.0	47.0	45.0	0.008929	-0.914787	24	14.0	10	0.025000	
3	63.0	47.0	44.0	41.0	0.012111	-0.598361	30	29.0	24	0.014963	
4	63.0	47.0	45.5	42.0	0.008292	-0.444039	32	27.5	20	0.020374	

5 rows × 100 columns

- **Show structure and datatype of the columns**

```
ALSdata.info()
```

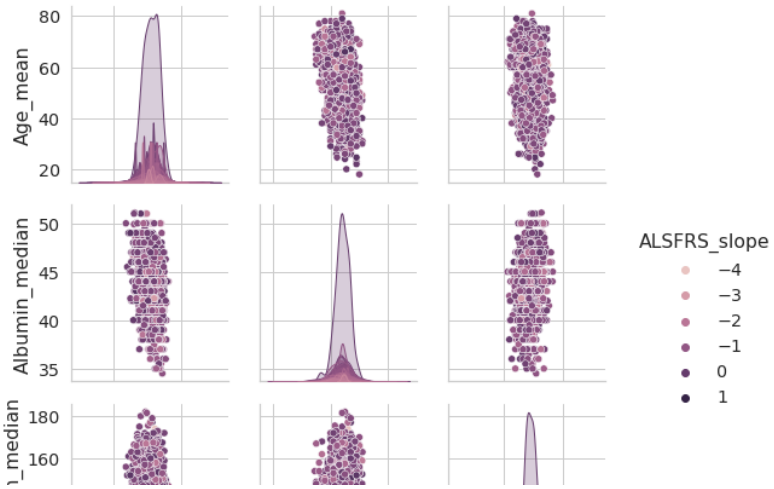
```
 43  Creatinine_median              2301 non-null   float64
 44  Creatinine_min                 2301 non-null   float64
 45  Creatinine_range               2301 non-null   float64
 46  Gender_mean                    2301 non-null   int64
 47  Glucose_max                    2301 non-null   float64
 48  Glucose_median                 2301 non-null   float64
 49  Glucose_min                    2301 non-null   float64
 50  Glucose_range                  2301 non-null   float64
 51  hands_max                      2301 non-null   int64
 52  hands_median                   2301 non-null   float64
 53  hands_min                      2301 non-null   int64
 54  hands_range                    2301 non-null   float64
 55  Hematocrit_max                 2301 non-null   float64
 56  Hematocrit_median              2301 non-null   float64
 57  Hematocrit_min                 2301 non-null   float64
 58  Hematocrit_range               2301 non-null   float64
 59  Hemoglobin_max                 2301 non-null   float64
 60  Hemoglobin_median              2301 non-null   float64
 61  Hemoglobin_min                 2301 non-null   float64
 62  Hemoglobin_range               2301 non-null   float64
 63  leg_max                        2301 non-null   int64
 64  leg_median                     2301 non-null   float64
 65  leg_min                        2301 non-null   int64
 66  leg_range                      2301 non-null   float64
 67  mouth_max                      2301 non-null   int64
 68  mouth_median                   2301 non-null   float64
 69  mouth_min                      2301 non-null   int64
 70  mouth_range                    2301 non-null   float64
```

- **Use 2 plots type to get a sense of the structure of the data.**

```
[16] Features_of_interest = ['Age_mean','Albumin_median', 'Hemoglobin_median', 'ALSFRS_slope']
     Record1=ALSdata[Features_of_interest]
```

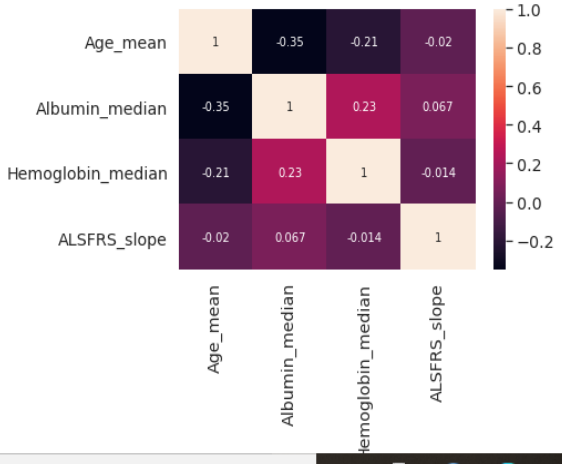
```
[17] sns.pairplot(Record1, hue = 'ALSFRS_slope')
```

```
<seaborn.axisgrid.PairGrid at 0x7fc28866a7f0>
```




```
[18] #Do correlations with select variables
     print(Record1.corr(method='pearson'))
     sns.heatmap(Record1.corr(method = 'pearson'), annot = True)
```

```
                    Age_mean  Albumin_median  Hemoglobin_median  ALSFRS_slope
Age_mean            1.000000       -0.352520          -0.205231     -0.019990
Albumin_median     -0.352520        1.000000           0.230923      0.066573
Hemoglobin_median  -0.205231        0.230923           1.000000     -0.013947
ALSFRS_slope       -0.019990        0.066573          -0.013947      1.000000
<matplotlib.axes._subplots.AxesSubplot at 0x7fc283d47dd8>
```



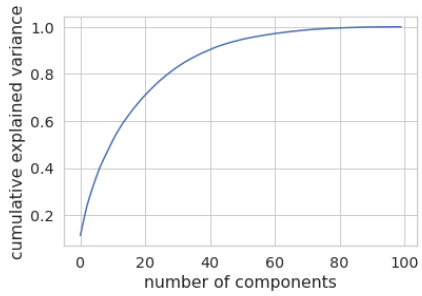
3.Using SKLearn perform PCA on the dataset to reduce the dimensionality of such a high dimensional dataset.

```
[24] #Using SKLearn perform PCA on the dataset to reduce the dimensionality of such a high
     #dimensional dataset. (5 marks)
     #How many components are enough to explain almost all of the data variance?
     #Show the percentage of variance explained by each of the selected components

     from sklearn.decomposition import PCA

     #assuming components are 8
     pca = PCA(n_components=8)
     principalComponents = pca.fit_transform(std_X)
     principalDf = pd.DataFrame(data = principalComponents,columns = ['principal component 1', 'principal component 2', 'principal component 3','principal component 4
     principalDf
     #'principal component 5', 'principal component 6'
     # 'principal component 4'
```

	principal component 1	principal component 2	principal component 3	principal component 4	principal component 5	principal component 6	principal component 7	principal component 8
0	-1.564831	-1.649734	3.824562	-2.048055	-4.010174	0.447730	2.046524	3.129604
1	-1.564778	-4.556931	2.171606	0.192965	-0.101932	1.224472	-2.124308	-1.375524
2	1.391688	-0.922851	-3.154862	-5.409162	-0.660358	1.386944	-1.940094	1.556096
3	-1.980780	1.794367	-2.496204	-1.399636	-0.244667	0.189275	2.552880	-0.899781
4	0.184430	-0.323466	-2.887351	2.823624	0.461742	-0.754318	-1.403298	-1.913720

- **How many components are enough to explain almost all of the data variance?**
  80 components are enough to explain almost all the variance.

```
#looks like 80 components explain the variance
pca = PCA().fit(std_X)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



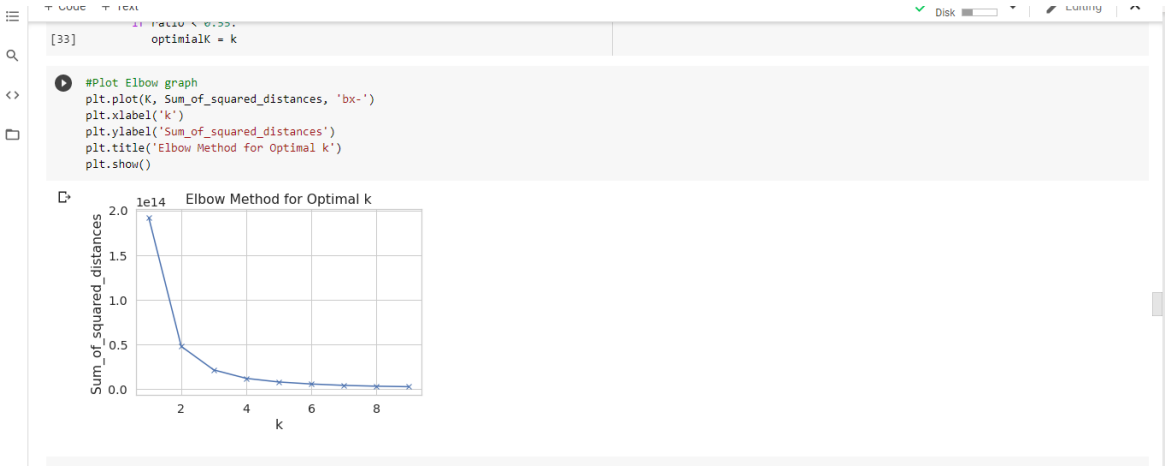
- **Show the percentage of variance explained by each of the selected components.**

```
pca = PCA(n_components=80)
principalComponents = pca.fit_transform(std_X)
pca.explained_variance_ratio_
```

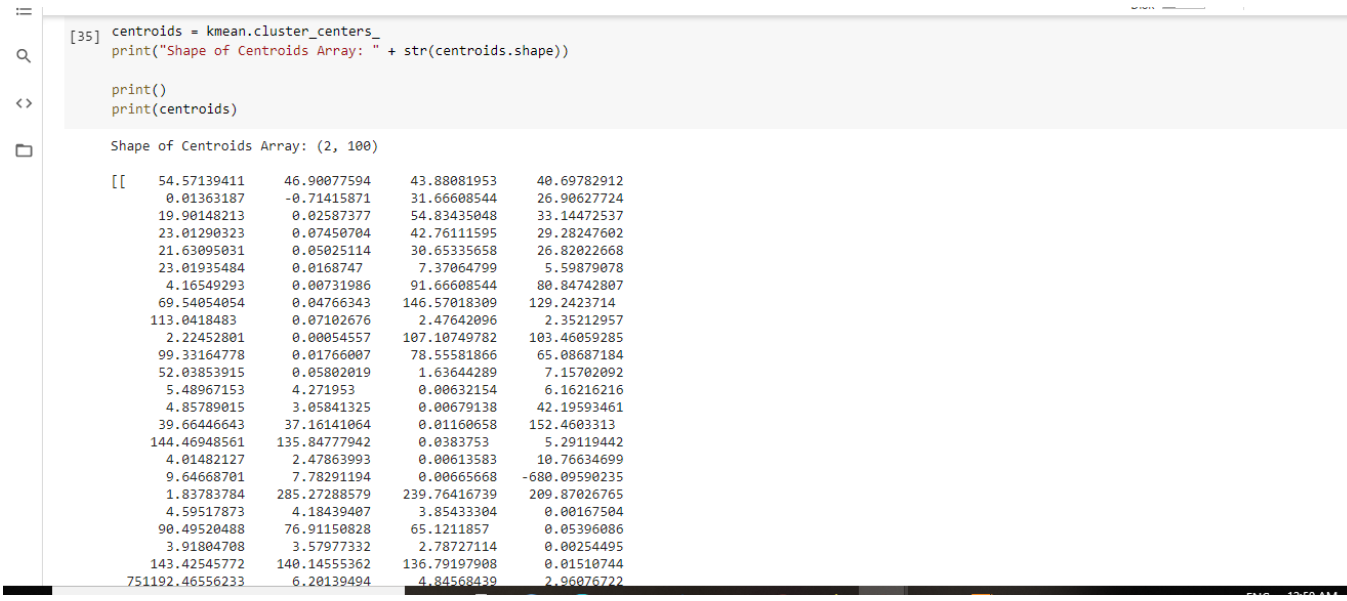
```
array([0.11412025, 0.06343331, 0.06065604, 0.04585207, 0.0425687 ,
       0.0390617 , 0.03676432, 0.03030579, 0.0286206 , 0.02841939,
       0.02775277, 0.0259821 , 0.0230879 , 0.02216568, 0.01934767,
       0.01885815, 0.01857634, 0.01694277, 0.01668287, 0.0162304 ,
       0.0148033 , 0.01431511, 0.01400788, 0.01346172, 0.01302761,
       0.01203336, 0.01149555, 0.01111184, 0.0106483 , 0.01005252,
       0.00990252, 0.00906912, 0.00881027, 0.00823282, 0.0075153 ,
       0.00732002, 0.00709569, 0.00685291, 0.00618436, 0.00599239,
       0.00590326, 0.00578525, 0.00555023, 0.0048705 , 0.00467846,
       0.00425226, 0.00415026, 0.00377709, 0.00374589, 0.00352726,
       0.00331795, 0.00297671, 0.00292313, 0.00266778, 0.00257965,
       0.00241242, 0.00233704, 0.00227316, 0.00217495, 0.00202   ,
       0.00195441, 0.00185004, 0.00181788, 0.00177258, 0.00167852,
       0.00164209, 0.00150853, 0.00144348, 0.00140357, 0.00133248,
       0.00127275, 0.00122266, 0.00110613, 0.00090169, 0.00086985,
       0.00080973, 0.00076441, 0.00073853, 0.00068251, 0.00061621])
```

4.Initialize and train a K-Means clustering model on our raw dataset(not on our principal components).

From implementation of elbow method, we can see that there are 2 clusters in the ALS dataset.



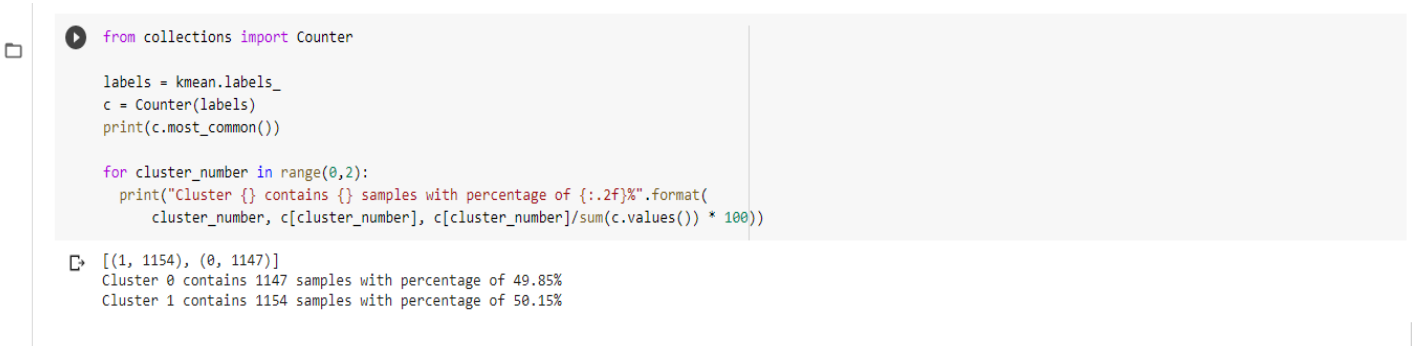
- **Show shape(# of rows/columns) of the centroids produced.**



```
[35] centroids = kmean.cluster_centers_
     print("Shape of Centroids Array: " + str(centroids.shape))

     print()
     print(centroids)

     Shape of Centroids Array: (2, 100)

     [[    54.57139411     46.90077594     43.88081953     40.69782912
            0.01363187     -0.71415871     31.66608544     26.90627724
           19.90148213      0.02587377     54.83435048     33.14472537
           23.01290323      0.07450704     42.76111595     29.28247602
           21.63095031      0.05025114     30.65335658     26.82022668
           23.01935484      0.0168747       7.37064799      5.59879078
            4.16549293      0.00731986     91.66608544     80.84742807
           69.54054054      0.04766343    146.57018309    129.2423714
          113.0418483       0.07102676      2.47642096      2.35212957
            2.22452801      0.00054557    107.10749782    103.46059285
           99.33164778      0.01766007     78.55581866     65.08687184
           52.03853915      0.05802019      1.63644289      7.15702092
            5.48967153      4.271953        0.00632154      6.16216216
            4.85789015      3.05841325      0.00679138     42.19593461
           39.66446643     37.16141064      0.01160658    152.4603313
          144.46948561    135.84777942      0.0383753       5.29119442
            4.01482127      2.47863993      0.00613583     10.76634699
            9.64668701      7.78291194      0.00665668   -680.09590235
            1.83783784    285.27288579    239.76416739    209.87026765
            4.59517873      4.18439407      3.85433304      0.00167504
           90.49520488     76.91150828     65.1211857       0.05396086
            3.91804708      3.57977332      2.78727114      0.00254495
          143.42545772    140.14555362    136.79197908      0.01510744
       751192.46556233      6.20139494      4.84568439      2.96076722
```

- **Show percentage of the number samples to each cluster as such.**

```
[(1, 1154), (0, 1147)]
Cluster 0 contains 1147 samples with percentage of 49.85%
Cluster 1 contains 1154 samples with percentage of 50.15%
```



```
from collections import Counter

labels = kmean.labels_
c = Counter(labels)
print(c.most_common())

for cluster_number in range(0,2):
  print("Cluster {} contains {} samples with percentage of {:.2f}%".format(
      cluster_number, c[cluster_number], c[cluster_number]/sum(c.values()) * 100))

[(1, 1154), (0, 1147)]
Cluster 0 contains 1147 samples with percentage of 49.85%
Cluster 1 contains 1154 samples with percentage of 50.15%
```

```
Cluster 0 contains 321 samples with percentage of 13.95%
Cluster 1 contains 316 samples with percentage of 13.73%
Cluster 2 contains 331 samples with percentage of 14.39%
Cluster 3 contains 319 samples with percentage of 13.86%
Cluster 4 contains 349 samples with percentage of 15.17%
Cluster 5 contains 337 samples with percentage of 14.65%
Cluster 6 contains 328 samples with percentage of 14.25%
```

5. Using the silhouette_score metric from sklearn, compute the mean Silhouette Coefficient of all samples and show it.

- **What number of clustering gives you a higher Silhouette score (try values of 2, 7, and 10)**
  With the number of clusters set at two, there is the highest silhouette score. Which makes sense as 2 clusters was seen as the ideal number of clusters for the ALSdata as shown from implementation of the Elbow method.

With 7 clusters, mean silhouette score is 0.55.

```
mean_sihouette_score = ss(Record2_array, labels)
print(mean_sihouette_score)
```

```
0.5517132137994109
```

With 10 clusters, mean silhouette score is 0.546.

```
[64] mean_sihouette_score = ss(Record3_array, labels)
     print(mean_sihouette_score)
```

```
0.5461974833927342
```

- **Produce a plot that resembles below. The actual values may differ which is fine. Refer to matplotlib.pyplot documentation for plot and figure functions.**

```
[91] #Line graph

     #https://www.kaggle.com/vipulgandhi/kmeans-detailed-explanation
     import matplotlib.pyplot as plt

     #Create an empty array of mean silhouette scores
     silhouette_scores = []

     #Implement for loop, running kmeans on clusters ranging from 2 to 50 in the ALSdata
     for n_cluster in range(2, 50):
         silhouette_scores.append(ss(ALSdata, KMeans(n_clusters = n_cluster).fit_predict(ALSdata)))

     # Plotting a line graph, plotting mean silhouette score with number of clusters
     k = [2, 3, 4, 5, 6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49]

     plt.plot(k, silhouette_scores, color='green', linestyle='dashed', linewidth = 2, marker='o', markeredgecolor = 'green', markerfacecolor='red')
     plt.title('K-means')
     plt.xlabel('Number of Clusters')
     plt.ylabel('Mean Silhouette Score')
     plt.show()
```

Below is the Plot of Silhouette scores by Cluster as generated from K-means clustering algorithm.