Jyothshna Epanagandla
11489606

INFO 5505 – Assignment 3

About the model:

K-means clustering is a type of unsupervised learning when dealing with unlabeled data (i.e., data without defined categories or groups). Iteratively, this method allocates each data point to one of K groups based on the qualities supplied. Data points are grouped together based on how similar their characteristics are. The goal of this approach is to find groups in data, where K is the number of groups.

Elbow Method

- The number of clusters K varies between one and 10. For each value of K, we compute the WCSS (Square Sum of Within-Clusters).
- WCSS is the sum of the squared distances between each point and the centroid of a cluster.
- **Graph:** When we look at the graph, we can see that it shifts quickly at one point, making an elbow shape. We plot the WCSS with the K value, the plot looks like an Elbow.
- As the number of clusters increases, the WCSS value will begin to decline.
- The WCSS value is greatest when K = 1. At this point, the graph begins to go almost parallel to the X-axis. This point corresponds to the optimum K value or the optimal number of clusters.

Dataset:

- There are 2 datasets:
  training (N1=2,223): **ALS_TrainingData_2223.csv**
  testing (N2=78): **ALS_TestingData_78.csv**

Given data is taken from the case study done on a devastating disease called amyotrophic lateral sclerosis (ALS) to find patterns. Here various categories are considered to analyze given data.

For the **train data**, there are 2223 rows and 101 columns
For the **test data,** we have 78 rows and 131 columns

```
import pandas as pd
dataframe_test = pd.read_csv('ALS_TestingData_78.csv')
dataframe_train = pd.read_csv('ALS_TrainingData_2223.csv')
```

Jyothshna Epanagandla
11489606

Here we are importing necessary packages and we are loading the given data sets so as to analyze the data.

Data preprocessing:

```
dataframe_train
```

	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	A
0	1	65	57.0	40.5	38.0	0.066202	-0.965608	30	28.0	22	0.021164	
1	2	48	45.0	41.0	39.0	0.010453	-0.921717	37	33.0	21	0.028725	
2	3	38	50.0	47.0	45.0	0.008929	-0.914787	24	14.0	10	0.025000	
3	4	63	47.0	44.0	41.0	0.012111	-0.598361	30	29.0	24	0.014963	
4	5	63	47.0	45.5	42.0	0.008292	-0.444039	32	27.5	20	0.020374	
...	...	...	...	...	...	...	...	...	...	...	...	
2218	2419	33	50.0	49.0	45.0	0.008772	-0.239501	35	32.5	30	0.009107	
2219	2420	61	47.0	45.0	42.0	0.009074	-0.388711	31	26.0	17	0.025408	
2220	2421	47	46.0	44.0	41.0	0.012111	-0.108631	26	23.0	20	0.010949	
2221	2422	37	49.0	44.0	39.0	0.017857	-0.855880	34	29.5	21	0.023214	
2222	2424	48	48.0	45.0	40.0	0.018476	-2.050562	37	34.0	11	0.059908	

2223 rows × 101 columns

```
dataframe_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2223 entries, 0 to 2222
Columns: 101 entries, ID to Urine.Ph_min
dtypes: float64(75), int64(26)
memory usage: 1.7 MB
```

```
dataframe_train.shape
```

```
(2223, 101)
```

Here we are checking given train data and getting the information such as the number of rows, columns, etc., in the data we are going to analyze.

Jyothshna Epanagandla
11489606

```
dataframe_test
```

	ID	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	AL
0	3	65.906849	46.0	44.0	43	0.024590	-1.767329	33	
1	4	54.000000	39.0	36.0	33	0.013100	-1.351852	32	
2	5	56.394521	46.0	43.0	39	0.009736	-0.412429	15	
3	6	72.619178	50.0	42.5	41	0.092784	-0.383403	34	
4	9	65.000000	45.0	42.0	36	0.021327	0.000000	37	
...	...	...	...	...	...	...	...	...	
73	96	45.000000	42.0	39.5	37	0.010823	-0.683521	33	
74	97	47.709589	47.0	44.0	41	0.005381	-0.324733	26	
75	98	59.928767	47.0	44.0	40	0.009333	-0.330616	33	
76	99	61.000000	42.0	39.0	38	0.007843	-0.439230	37	
77	100	47.210959	50.0	46.0	43	0.011129	-0.434524	35	

78 rows × 131 columns

```
dataframe_test.shape
```

```
(78, 131)
```

Here we are checking the test data and getting the information like the number of rows and columns in the data.

Jyothshna Epanagandla
11489606

```
dataframe_train.describe
```

```
<bound method NDFrame.describe of         ID  Age_mean  Albumin_max  ...  Urine.Ph_max  Urine.Ph_median  Urine.Ph_min
0        1        65         57.0  ...          6.00              6.0           6.0
1        2        48         45.0  ...          7.00              5.0           5.0
2        3        38         50.0  ...          6.00              5.0           5.0
3        4        63         47.0  ...          7.00              6.0           5.0
4        5        63         47.0  ...          6.00              5.0           5.0
...    ...       ...          ...  ...           ...              ...           ...
2218  2419        33         50.0  ...          7.00              6.0           5.0
2219  2420        61         47.0  ...          7.41              5.5           5.0
2220  2421        47         46.0  ...          9.00              6.0           5.0
2221  2422        37         49.0  ...          6.00              5.0           5.0
2222  2424        48         48.0  ...          5.00              5.0           5.0

[2223 rows x 101 columns]>
```

```
dataframe_test.describe
```

```
<bound method NDFrame.describe of      ID   Age_mean  ...  White.Blood.Cell..WBC._min  White.Blood.Cell..WBC._range
0     3  65.906849  ...                        6.60                      0.016148
1     4  54.000000  ...                        4.97                      0.006703
2     5  56.394521  ...                        5.01                      0.005410
3     6  72.619178  ...                        4.96                      0.076495
4     9  65.000000  ...                        7.94                      0.008507
..  ...        ...  ...                         ...                           ...
73   96  45.000000  ...                        5.56                      0.014913
74   97  47.709589  ...                        8.37                      0.004978
75   98  59.928767  ...                        4.48                      0.016280
76   99  61.000000  ...                        5.81                      0.009510
77  100  47.210959  ...                        4.88                      0.004547
```

Here we are getting statistics of the given train data and test data.

Jyothshna Epanagandla
11489606

```
dataframe_train.isnull().sum()
```

```
ID                0
Age_mean          0
Albumin_max       0
Albumin_median    0
Albumin_min       0
                 ..
trunk_min         0
trunk_range       0
Urine.Ph_max      0
Urine.Ph_median   0
Urine.Ph_min      0
Length: 101, dtype: int64
```

```
dataframe_test.isnull().sum()
```

```
ID                                0
Age_mean                          0
Albumin_max                       0
Albumin_median                    0
Albumin_min                       0
                                 ..
Urine.Ph_range                    0
White.Blood.Cell..WBC._max        0
White.Blood.Cell..WBC._median     0
White.Blood.Cell..WBC._min        0
White.Blood.Cell..WBC._range      0
Length: 131, dtype: int64
```

Here we are checking for the null values in the given data so as to avoid errors caused by the null values in the future

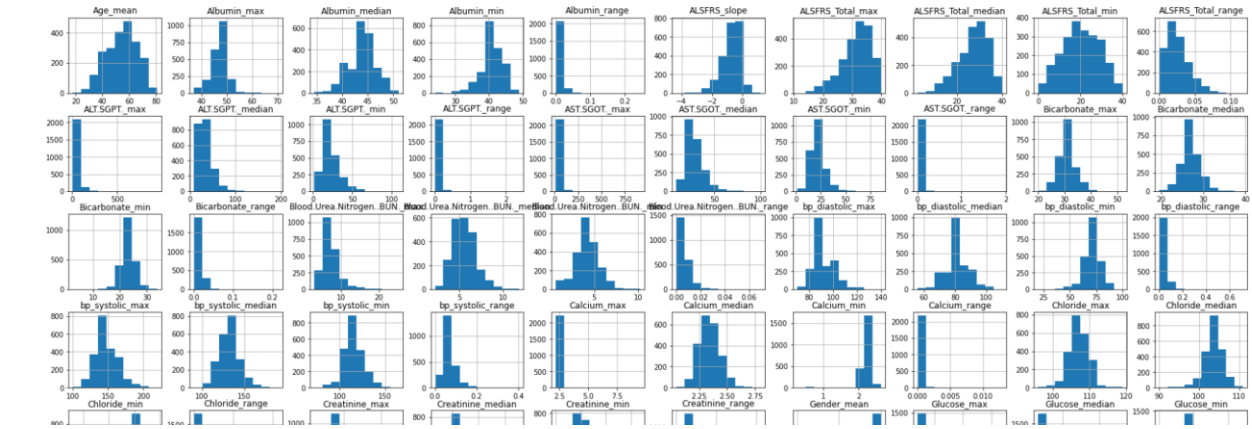
Data Visualization:

```
#From the train data, here we are dropping ID,SubjectID columns
dataframe_train = dataframe_train.drop(labels=['ID','SubjectID'], axis=1)
```

Here we are dropping ID and SubjectID columns as we no longer need them for our further analysis of data.

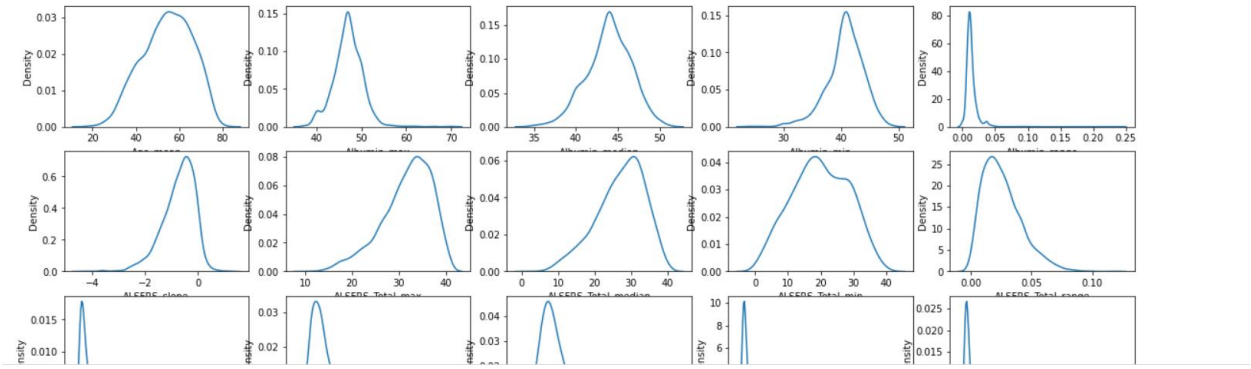
Jyothshna Epanagandla
11489606

```
[14] #Histograph creation
     #for all the columns in the train data(dataframe_train)
     hist_plot = dataframe_train.hist(figsize=(30,25))
```



Here we are creating histograms for each column present in the train data.

```
import matplotlib.pyplot as pl
import seaborn as sns
pl.figure(figsize=(20,55))
for j, column in enumerate(dataframe_train.columns):
    if dataframe_train[column].dtype != 'object':
        ax_is = pl.subplot(20, 5, j+1)
        sns.kdeplot(dataframe_train[column], ax=ax_is)
        pl.xlabel(column)
pl.show()
```



Getting log values for train data

```
columns = ['Albumin_max','Albumin_min','ALT.SGPT._max','ALT.SGPT._min','AST.SGOT._max','AST.SGOT._min','Bicarbonate_max',
        'Bicarbonate_min','bp_diastolic_max','bp_diastolic_min','Calcium_max','Calcium_min','Chloride_max','Chloride_min',
        'Glucose_max','Glucose_min','Hematocrit_max','Hematocrit_min','Hemoglobin_max','Hemoglobin_min','Potassium_max','Potassium_min',
        'Sodium_max','Sodium_min','Creatinine_max','Creatinine_min']
```

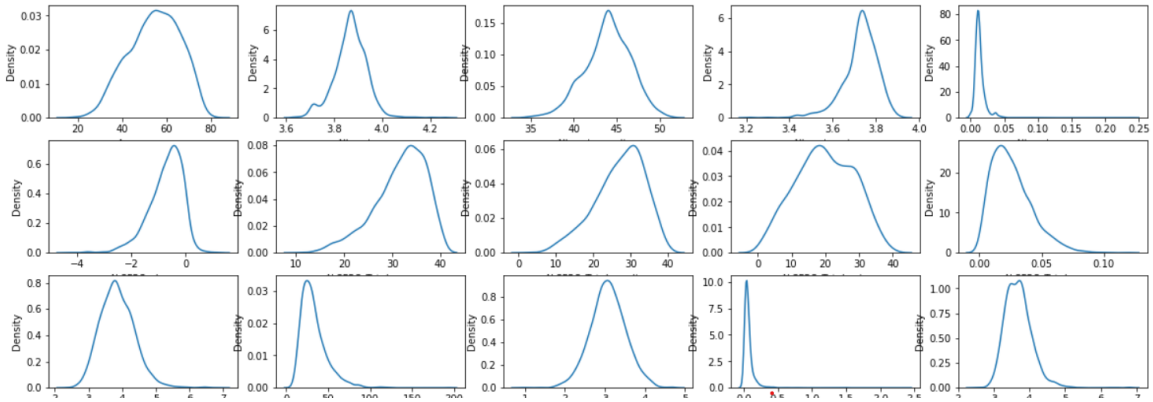
```
import numpy as np
for cols in columns:
    dataframe_train[cols] = np.log(1 + dataframe_train[cols])
```

Here we are considering random columns for data analysis.

Jyothshna Epanagandla
11489606

```
pl.figure(figsize=(20,50))
for e, cols in enumerate(dataframe_train.columns):
    if dataframe_train[cols].dtype != 'object':
        ax_is = pl.subplot(20, 5, e+1)
        sns.kdeplot(dataframe_train[cols], ax=ax_is)
        pl.xlabel(cols)

pl.show()
```

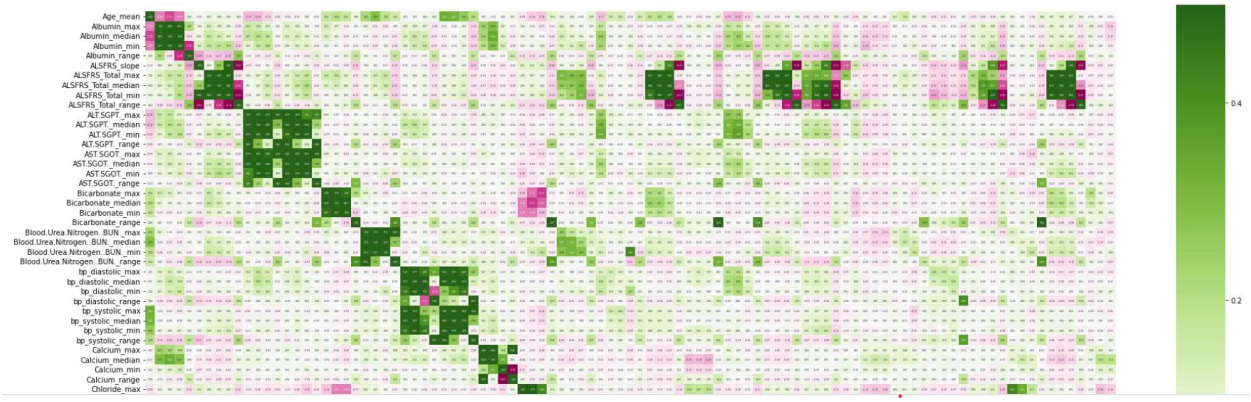


Here we are plotting density graphs

HeatMap:

```
dataframe_train_corr = dataframe_train.iloc[:, 1:].corr()
figure, ax_is = pl.subplots(figsize=(30, 25))

hm = sns.heatmap(dataframe_train.corr(), cbar=True, vmin=-0.5, vmax=0.5,linewidths=0,
                 fmt='.2f', annot_kws={'size': 3}, annot=True,
                 square=True, cmap="PiYG")
```



Here we are creating heatmap for the correlation of variables.

Principal Component Analysis:

Principal Component Analysis (PCA) method is used for increasing the speed of fitting the machine learning algorithm. If the algorithm is very slow because the input dimension is more, then we can use this technique to increase the efficiency of data analysis.

Jyothshna Epanagandla
11489606

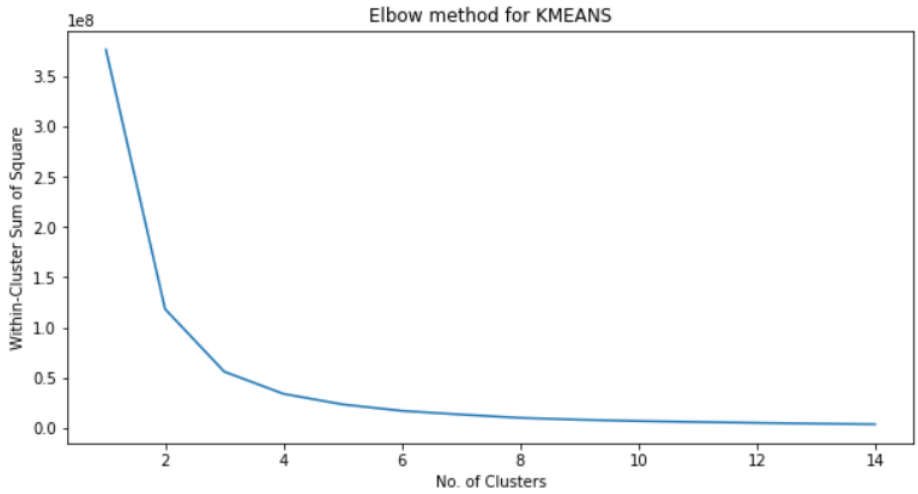
```
from sklearn.decomposition import PCA
km_PCA = PCA(n_components=1)
X_train = km_PCA.fit_transform(dataframe_train)
from sklearn.preprocessing import StandardScaler
stdrd_scaler = StandardScaler()
stdrd_scaler.fit_transform(X_train)
```

```
array([[ 0.83159079],
       [-0.83123685],
       [ 1.21172045],
       ...,
       [ 2.20192442],
       [-0.3022778 ],
       [-1.16768437]])
```

Here we are importing StandardScaler to standardize the data onto the unit scale i.e., mean=0 and variance=1. Through this, we can increase the performance efficiency of the algorithm.

```
from sklearn.cluster import KMeans
K_M_models = [KMeans(n_clusters=k, random_state=25).fit(X_train) for k in range (1, 15)]
K_M_inertia = [model.inertia_ for model in K_M_models]

figure, a_x = pl.subplots(figsize=(10,5))
pl.plot(range(1, 15), K_M_inertia)
pl.title('Elbow method for KMEANS')
pl.xlabel('No. of Clusters')
pl.ylabel('Within-Cluster Sum of Square')
pl.show()
```



The elbow method finds the optimal cluster number i.e., it is the point where the improvements decline suddenly creating an elbow shape.
In the above figure, we can see that graph is changing suddenly from k=2 to k=4.

Jyothshna Epanagandla
11489606

```
standard_scaler = StandardScaler()
standard_scaler.fit(X_test)
X_test = standard_scaler.transform(X_test)

def km_mod(X, n_neighbors):
    km_mod = KMeans(n_neighbors)
    km_mod.fit(X)
    km_clust_labels = km_mod.predict(X)
    km_center = km_mod.cluster_centers_
    return (km_clust_labels, km_center)
```

```
#for K=2
K=2
km_clust_label_1, km_center_1 = km_mod(X_train, K)
km_1 = pd.DataFrame(km_clust_label_1)
km_1
```

	0
0	1
1	0
2	1
3	0
4	1
...	...
2218	0

Jyothshna Epanagandla
11489606

```
#for k=3
K=3
km_clust_label_2, km_center_2 = km_mod(X_train, K)
km_2 = pd.DataFrame(km_clust_label_2)
km_2
```

	0
0	1
1	2
2	0
3	2
4	0
...	...
2218	1
2219	2
2220	0
2221	2
2222	2

2223 rows × 1 columns

Jyothshna Epanagandla
11489606

```
#for k=4
K=4
km_clust_label_3, km_center_3 = km_mod(X_train, K)
km_3 = pd.DataFrame(km_clust_label_3)
km_3
```

	0
0	3
1	2
2	3
3	2
4	1
...	...
2218	0
2219	0
2220	1
2221	0
2222	2

2223 rows × 1 columns

Here we created a function to return cluster labels and center values and then tested it for different k values i.e., k=2,3,4.

Jyothshna Epanagandla
11489606

```
#Cluster results by assigning cluster labels to cluster id in given dataset
dataframe_train['clusterid'] = km_clust_label_1
dataframe_train
```

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range	...	Sodium_min	Sodi
0	65	inf	40.5	inf	0.066202	-0.965608	30	28.0	22	0.021164	...	inf	
1	48	inf	41.0	inf	0.010453	-0.921717	37	33.0	21	0.028725	...	inf	
2	38	inf	47.0	inf	0.008929	-0.914787	24	14.0	10	0.025000	...	inf	
3	63	inf	44.0	inf	0.012111	-0.598361	30	29.0	24	0.014963	...	inf	
4	63	inf	45.5	inf	0.008292	-0.444039	32	27.5	20	0.020374	...	inf	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2218	33	inf	49.0	inf	0.008772	-0.239501	35	32.5	30	0.009107	...	inf	
2219	61	inf	45.0	inf	0.009074	-0.388711	31	26.0	17	0.025408	...	inf	
2220	47	inf	44.0	inf	0.012111	-0.108631	26	23.0	20	0.010949	...	inf	
2221	37	inf	44.0	inf	0.017857	-0.855880	34	29.5	21	0.023214	...	inf	
2222	48	inf	45.0	inf	0.018476	-2.050562	37	34.0	11	0.059908	...	inf	

2223 rows × 100 columns

Here we are adding cluster-id (where k=4) as a column to the train data so as to visualize the data.

```
# Inversing log transform of columns
for col in columns:
  dataframe_train[col] = np.exp(dataframe_train[col])
dataframe_train
```

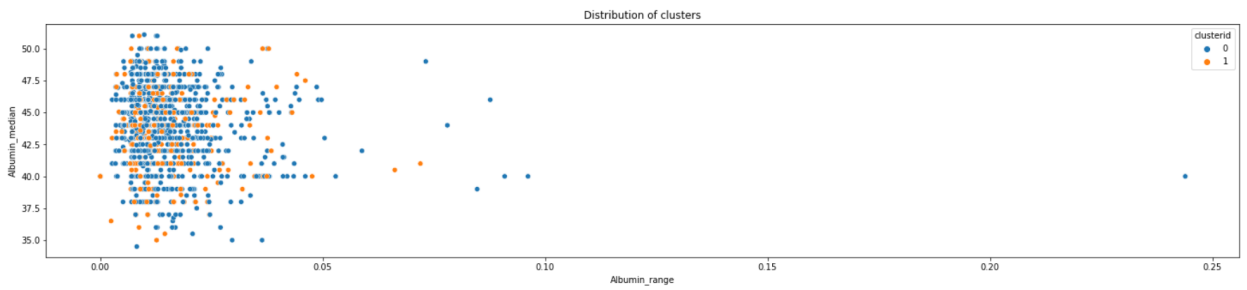
```
/usr/local/lib/python3.7/dist-packages/pandas/core/arraylike.py:364: RuntimeWarning: overflow encountered in exp
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

	Age_mean	Albumin_max	Albumin_median	Albumin_min	Albumin_range	ALSFRS_slope	ALSFRS_Total_max	ALSFRS_Total_median	ALSFRS_Total_min	ALSFRS_Total_range	...	
0	65	1.545539e+25	40.5	8.659340e+16	0.066202	-0.965608	30	28.0	22	0.021164	...	3.4
1	48	9.496119e+19	41.0	2.353853e+17	0.010453	-0.921717	37	33.0	21	0.028725	...	3.1
2	38	1.409349e+22	47.0	9.496119e+19	0.008929	-0.914787	24	14.0	10	0.025000	...	1.1
3	63	7.016736e+20	44.0	1.739275e+18	0.012111	-0.598361	30	29.0	24	0.014963	...	2.3
4	63	7.016736e+20	45.5	4.727839e+18	0.008292	-0.444039	32	27.5	20	0.020374	...	2.3
...	...	...	...	...	...	...	...	...	...	...	...	
2218	33	1.409349e+22	49.0	9.496119e+19	0.008772	-0.239501	35	32.5	30	0.009107	...	3.1
2219	61	7.016736e+20	45.0	4.727839e+18	0.009074	-0.388711	31	26.0	17	0.025408	...	4.6
2220	47	2.581313e+20	44.0	1.739275e+18	0.012111	-0.108631	26	23.0	20	0.010949	...	1.1
2221	37	5.184706e+21	44.0	2.353853e+17	0.017857	-0.855880	34	29.5	21	0.023214	...	3.1
2222	48	1.907347e+21	45.0	6.398435e+17	0.018476	-2.050562	37	34.0	11	0.059908	...	8.5

2223 rows × 100 columns

Here we are inversing the log transform of columns that we transformed earlier.

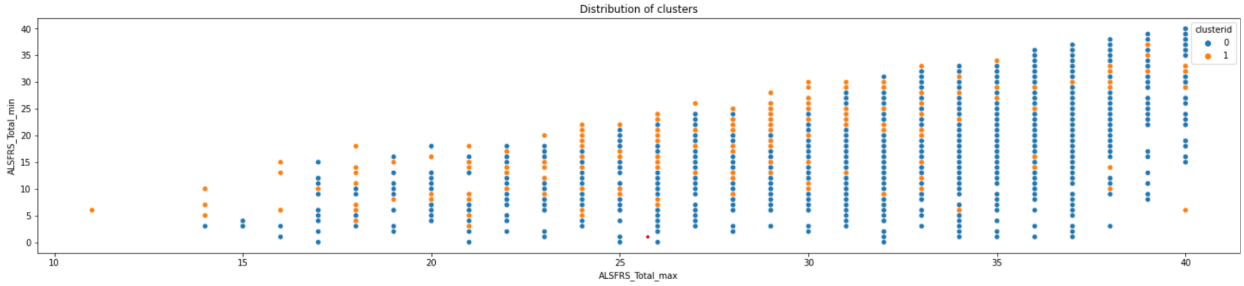
```
# Plotting clusters with two attributes at a time
pl.figure(figsize=(25,5))
sns.scatterplot(data=dataframe_train, x='Albumin_range', y='Albumin_median',hue='clusterid')
pl.title('Distribution of clusters')
pl.show()
```



Here we are plotting clusters for two attributes i.e., albumin_range and albumin_median taken from the train data.

Jyothshna Epanagandla
11489606

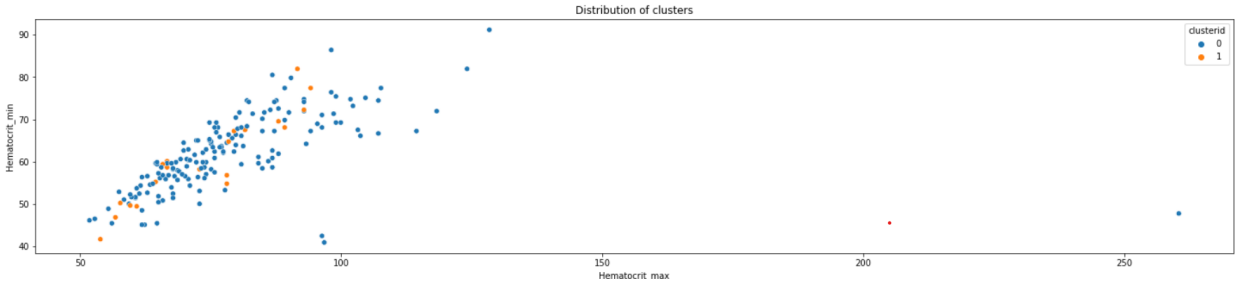
```
# Distribution of clusters
pl.figure(figsize=(25,5))
sns.scatterplot(data=dataframe_train, x='ALSFRS_Total_max', y='ALSFRS_Total_min',hue='clusterid')
pl.title('Distribution of clusters')
pl.show()
```



Here we are plotting clusters for two attributes i.e., ALSFRS_Total_max and ALSFRS_Total_min taken from the train data.

```
# Distribution of clusters

pl.figure(figsize=(25,5))
sns.scatterplot(data=dataframe_train, x='Hematocrit_max', y='Hematocrit_min',hue='clusterid')
pl.title('Distribution of clusters')
pl.show()
```



Here we are plotting clusters for two attributes i.e., Hematocrit_max and Hematocrit_min taken from the train data.

Jyothshna Epanagandla
11489606

```
from sklearn.metrics import silhouette_score
# For k=2

K_M_S_2= KMeans(n_clusters=2, random_state=25)
K_M_S_2.fit(X_train)

print('Silhoutte score of our model is ' + str(silhouette_score(X_train, K_M_S_2.labels_)))
```

```
Silhoutte score of our model is 0.6411814157596237
```

```
# For K=3

K_M_S_3 = KMeans(n_clusters=3, random_state=25)
K_M_S_3.fit(X_train)

print('Silhoutte score of our model is ' + str(silhouette_score(X_train, K_M_S_3.labels_)))
```

```
Silhoutte score of our model is 0.5937688686189928
```

```
# For k=4

K_M_S_4 = KMeans(n_clusters=4, random_state=25)
K_M_S_4.fit(X_train)

print('Silhoutte score of our model is ' + str(silhouette_score(X_train, K_M_S_4.labels_)))
```

```
Silhoutte score of our model is 0.5657511728509033
```

Here we are calculating the silhouette score for the three different values of k.

Silhouette score is computed using the mean intra-cluster distance and mean nearest cluster distance. This value ranges from 1 and -1.

Conclusion:

Here the values obtained for the k values are as follows :

Score = `0.6411814157596237` for K=2
Score = `0.5937688686189928` for K=3
Score = `0.5657511728509033` for K=4

By examining the silhouette co-efficient obtained for the k values we can conclude that the best fit for the model would be K=2.