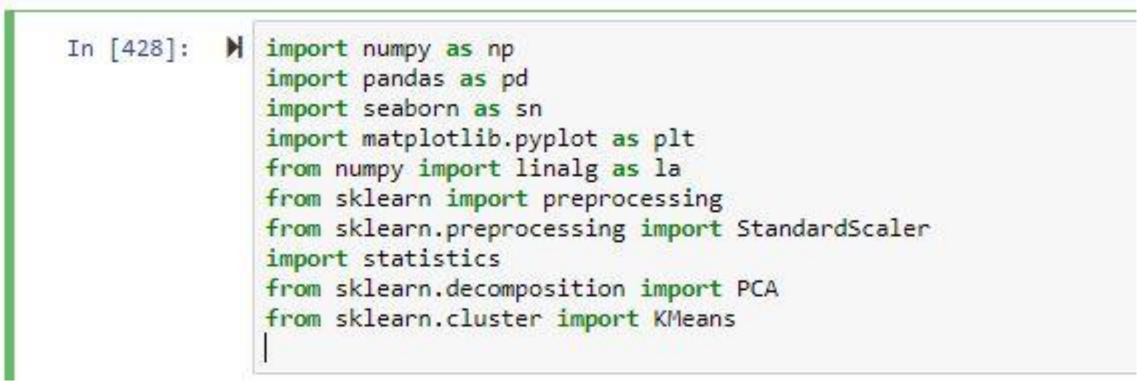
PROJECT 2

GEOSTATISTICS

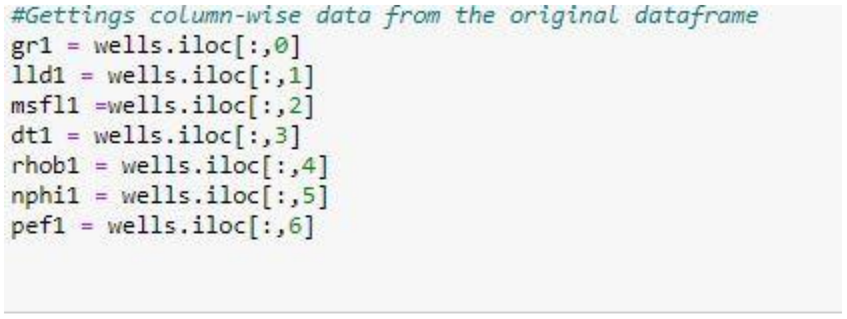
AASHRIT BAGAREDDY



```
In [428]:  import numpy as np
           import pandas as pd
           import seaborn as sn
           import matplotlib.pyplot as plt
           from numpy import linalg as la
           from sklearn import preprocessing
           from sklearn.preprocessing import StandardScaler
           import statistics
           from sklearn.decomposition import PCA
           from sklearn.cluster import KMeans
```

Figure 1: Imported Libraries and functions

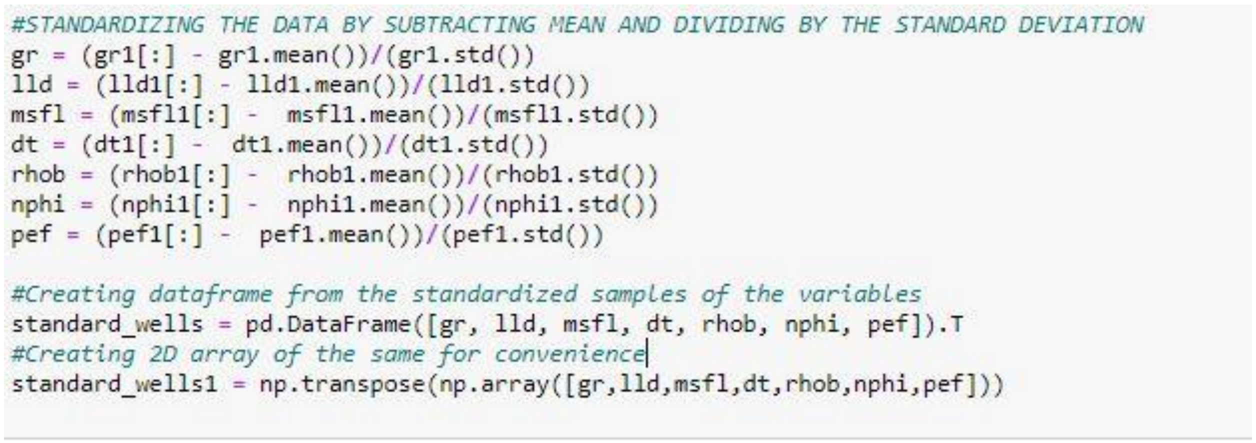
The libraries shown in Fig.1 were used, while some may not have been used in the final code.



```
#Gettings column-wise data from the original dataframe
gr1 = wells.iloc[:,0]
lld1 = wells.iloc[:,1]
msfl1 =wells.iloc[:,2]
dt1 = wells.iloc[:,3]
rhob1 = wells.iloc[:,4]
nphi1 = wells.iloc[:,5]
pef1 = wells.iloc[:,6]
```

Figure 2 : Column-wise variable sample data

The column wise data was separated individually for individual variable data GR, LLD, MSFL, DT, RHOB, NPHI and PEF into individual columns from an original data frame.

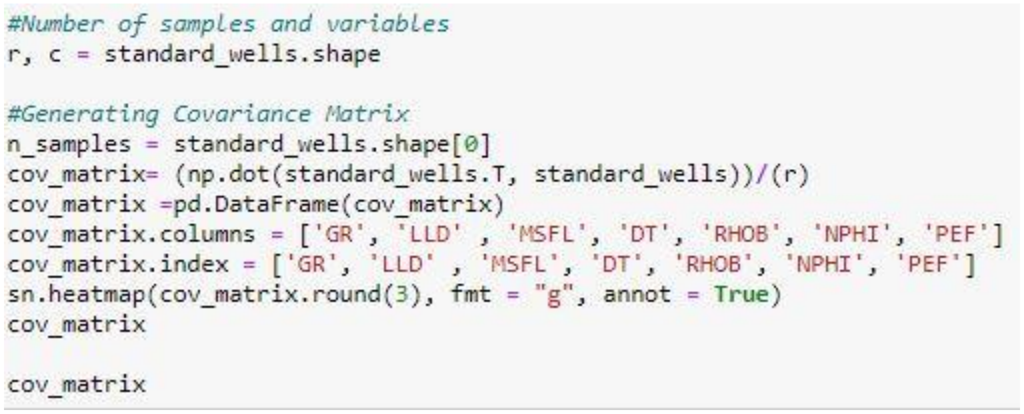


```
#STANDARDIZING THE DATA BY SUBTRACTING MEAN AND DIVIDING BY THE STANDARD DEVIATION
gr = (gr1[:] - gr1.mean())/(gr1.std())
lld = (lld1[:] - lld1.mean())/(lld1.std())
msfl = (msfl1[:] -  msfl1.mean())/(msfl1.std())
dt = (dt1[:] -  dt1.mean())/(dt1.std())
rhob = (rhob1[:] -  rhob1.mean())/(rhob1.std())
nphi = (nphi1[:] -  nphi1.mean())/(nphi1.std())
pef = (pef1[:] -  pef1.mean())/(pef1.std())

#Creating dataframe from the standardized samples of the variables
standard_wells = pd.DataFrame([gr, lld, msfl, dt, rhob, nphi, pef]).T
#Creating 2D array of the same for convenience
standard_wells1 = np.transpose(np.array([gr,lld,msfl,dt,rhob,nphi,pef]))
```

Figure 3 : Normalization of the data

The data columns separated out from the previous step are standardized individually to a mean of 0 and a variance of 1 in Fig 3. The individual data is then used to construct a dataframe of the normalized data.



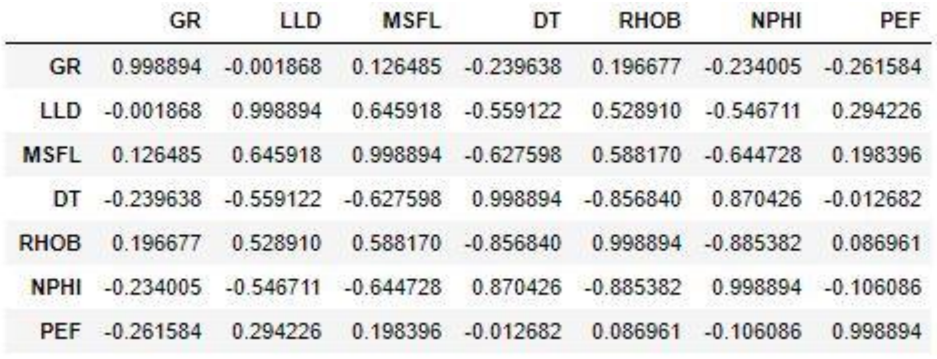
```
#Number of samples and variables
r, c = standard_wells.shape

#Generating Covariance Matrix
n_samples = standard_wells.shape[0]
cov_matrix= (np.dot(standard_wells.T, standard_wells))/(r)
cov_matrix =pd.DataFrame(cov_matrix)
cov_matrix.columns = ['GR', 'LLD' , 'MSFL', 'DT', 'RHOB', 'NPHI', 'PEF']
cov_matrix.index = ['GR', 'LLD' , 'MSFL', 'DT', 'RHOB', 'NPHI', 'PEF']
sn.heatmap(cov_matrix.round(3), fmt = "g", annot = True)
cov_matrix

cov_matrix
```

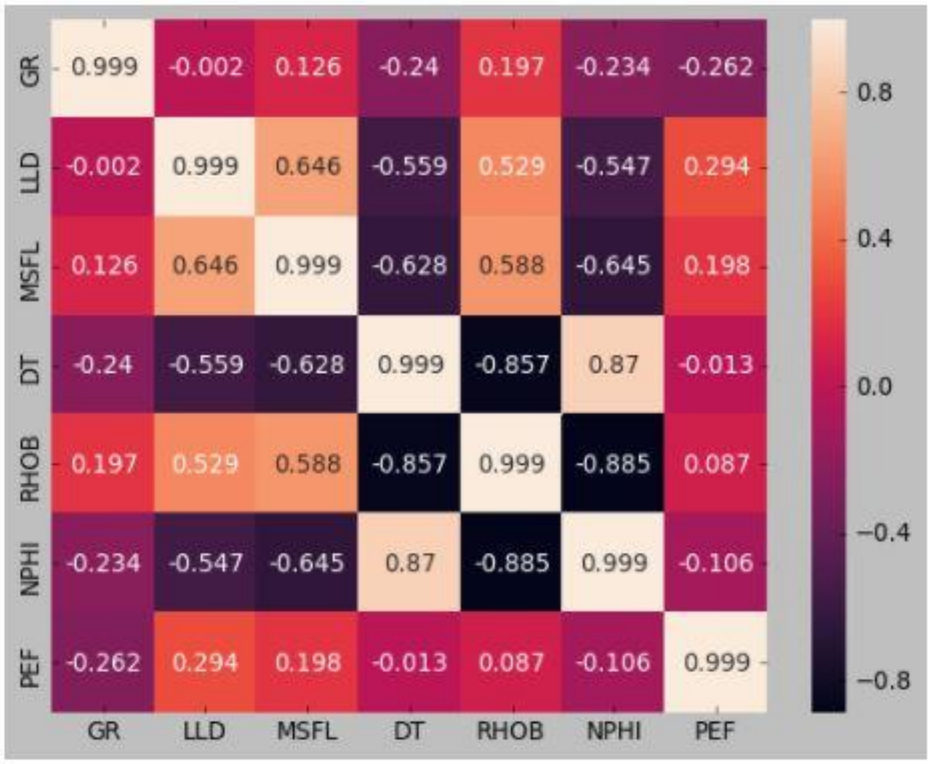
Figure 4 : Generation of Covariance Matrix

The covariance matrix for the given normalized dataset is generated by multiplying the transpose of the dataset by itself, divided by the total number of samples(Fig 4).



	GR	LLD	MSFL	DT	RHOB	NPHI	PEF
GR	0.998894	-0.001868	0.126485	-0.239638	0.196677	-0.234005	-0.261584
LLD	-0.001868	0.998894	0.645918	-0.559122	0.528910	-0.546711	0.294226
MSFL	0.126485	0.645918	0.998894	-0.627598	0.588170	-0.644728	0.198396
DT	-0.239638	-0.559122	-0.627598	0.998894	-0.856840	0.870426	-0.012682
RHOB	0.196677	0.528910	0.588170	-0.856840	0.998894	-0.885382	0.086961
NPHI	-0.234005	-0.546711	-0.644728	0.870426	-0.885382	0.998894	-0.106086
PEF	-0.261584	0.294226	0.198396	-0.012682	0.086961	-0.106086	0.998894

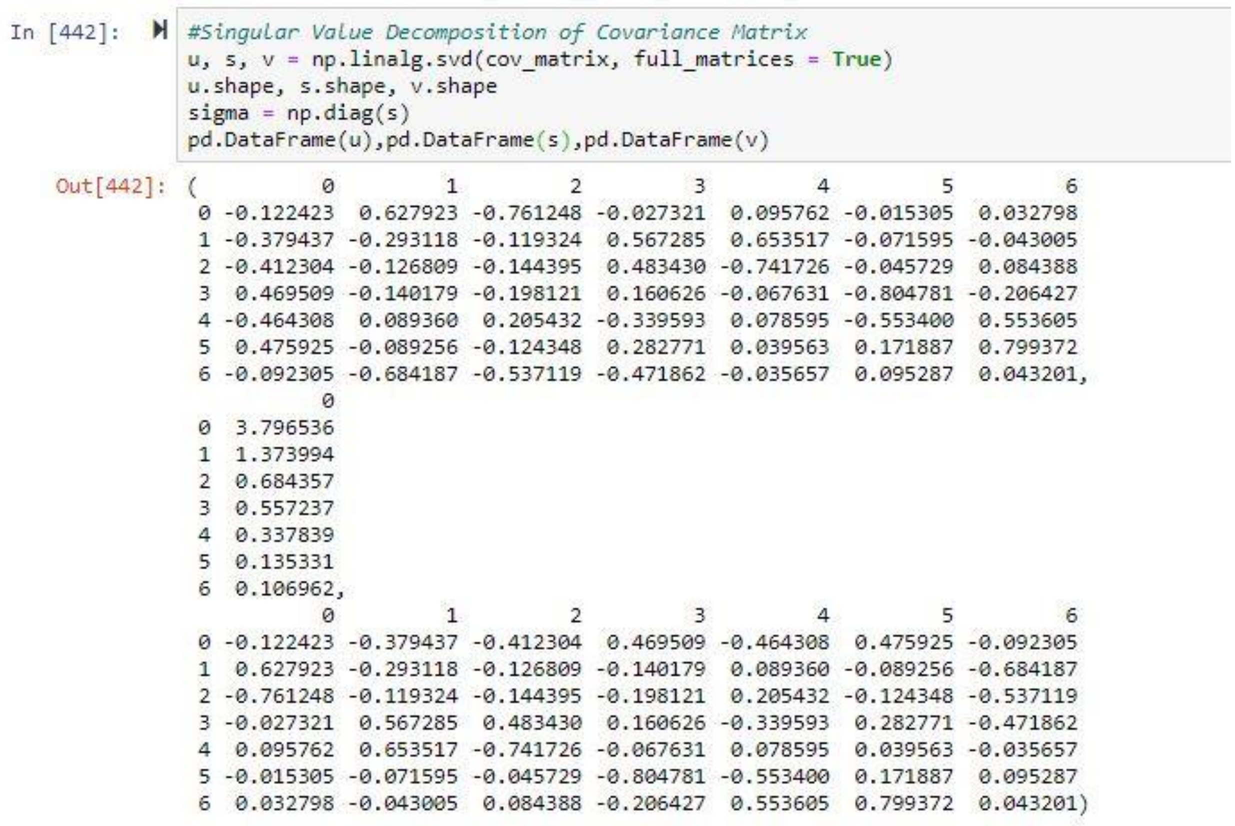
Figure 5 : Covariance Matrix



	GR	LLD	MSFL	DT	RHOB	NPHI	PEF
GR	0.999	-0.002	0.126	-0.24	0.197	-0.234	-0.262
LLD	-0.002	0.999	0.646	-0.559	0.529	-0.547	0.294
MSFL	0.126	0.646	0.999	-0.628	0.588	-0.645	0.198
DT	-0.24	-0.559	-0.628	0.999	-0.857	0.87	-0.013
RHOB	0.197	0.529	0.588	-0.857	0.999	-0.885	0.087
NPHI	-0.234	-0.547	-0.645	0.87	-0.885	0.999	-0.106
PEF	-0.262	0.294	0.198	-0.013	0.087	-0.106	0.999

Figure 6 : Covariance Matrix Heatmap

Figures 5 and 6 show the generated covariance matrix. The diagonal values are equal to the variance of individual variables as covariance of an array with itself is 1(as data is standardized).



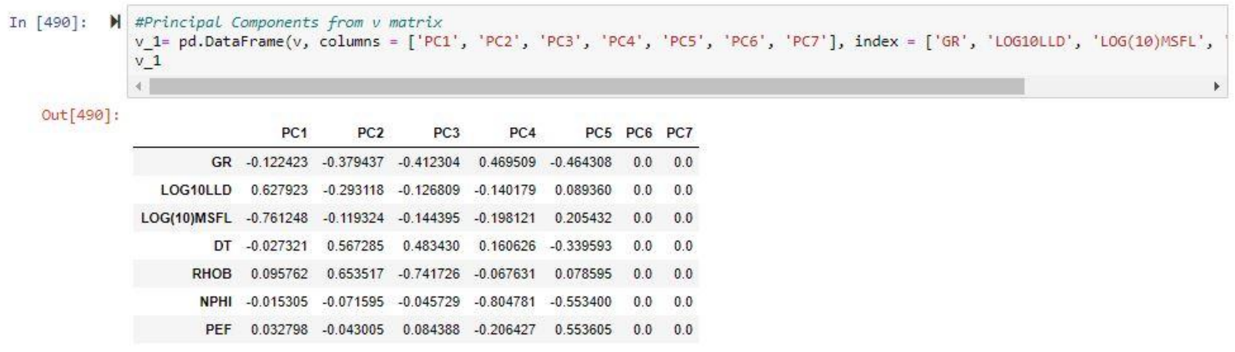
```
In [442]:  #Singular Value Decomposition of Covariance Matrix
           u, s, v = np.linalg.svd(cov_matrix, full_matrices = True)
           u.shape, s.shape, v.shape
           sigma = np.diag(s)
           pd.DataFrame(u),pd.DataFrame(s),pd.DataFrame(v)
```

```
Out[442]: (          0         1         2         3         4         5         6
           0 -0.122423  0.627923 -0.761248 -0.027321  0.095762 -0.015305  0.032798
           1 -0.379437 -0.293118 -0.119324  0.567285  0.653517 -0.071595 -0.043005
           2 -0.412304 -0.126809 -0.144395  0.483430 -0.741726 -0.045729  0.084388
           3  0.469509 -0.140179 -0.198121  0.160626 -0.067631 -0.804781 -0.206427
           4 -0.464308  0.089360  0.205432 -0.339593  0.078595 -0.553400  0.553605
           5  0.475925 -0.089256 -0.124348  0.282771  0.039563  0.171887  0.799372
           6 -0.092305 -0.684187 -0.537119 -0.471862 -0.035657  0.095287  0.043201,
                     0
           0  3.796536
           1  1.373994
           2  0.684357
           3  0.557237
           4  0.337839
           5  0.135331
           6  0.106962,
                     0         1         2         3         4         5         6
           0 -0.122423 -0.379437 -0.412304  0.469509 -0.464308  0.475925 -0.092305
           1  0.627923 -0.293118 -0.126809 -0.140179  0.089360 -0.089256 -0.684187
           2 -0.761248 -0.119324 -0.144395 -0.198121  0.205432 -0.124348 -0.537119
           3 -0.027321  0.567285  0.483430  0.160626 -0.339593  0.282771 -0.471862
           4  0.095762  0.653517 -0.741726 -0.067631  0.078595  0.039563 -0.035657
           5 -0.015305 -0.071595 -0.045729 -0.804781 -0.553400  0.171887  0.095287
           6  0.032798 -0.043005  0.084388 -0.206427  0.553605  0.799372  0.043201)
```

Figure 7 : Singular Value Decomposition

The singular value decomposition is performed on the generated covariance matrix. This singular value decomposition results in u,s and v matrices, where u and v are the the left hand singular vectors matrix and the right hand singular vectors matrix respectively, and s is the singular values matrix. The 3 matrices are showcased in Fig 7.

In this case, 'u' matrix would be similar to transpose of 'v' matrix.



```
In [490]:  #Principal Components from v matrix
           v_1= pd.DataFrame(v, columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'], index = ['GR', 'LOG10LLD', 'LOG(10)MSFL',
           v_1
```

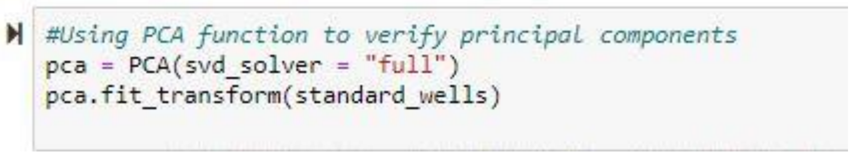
Out[490]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
GR	-0.122423	-0.379437	-0.412304	0.469509	-0.464308	0.0	0.0
LOG10LLD	0.627923	-0.293118	-0.126809	-0.140179	0.089360	0.0	0.0
LOG(10)MSFL	-0.761248	-0.119324	-0.144395	-0.198121	0.205432	0.0	0.0
DT	-0.027321	0.567285	0.483430	0.160626	-0.339593	0.0	0.0
RHOB	0.095762	0.653517	-0.741726	-0.067631	0.078595	0.0	0.0
NPHI	-0.015305	-0.071595	-0.045729	-0.804781	-0.553400	0.0	0.0
PEF	0.032798	-0.043005	0.084388	-0.206427	0.553605	0.0	0.0

Figure 8 : Principal component matrix

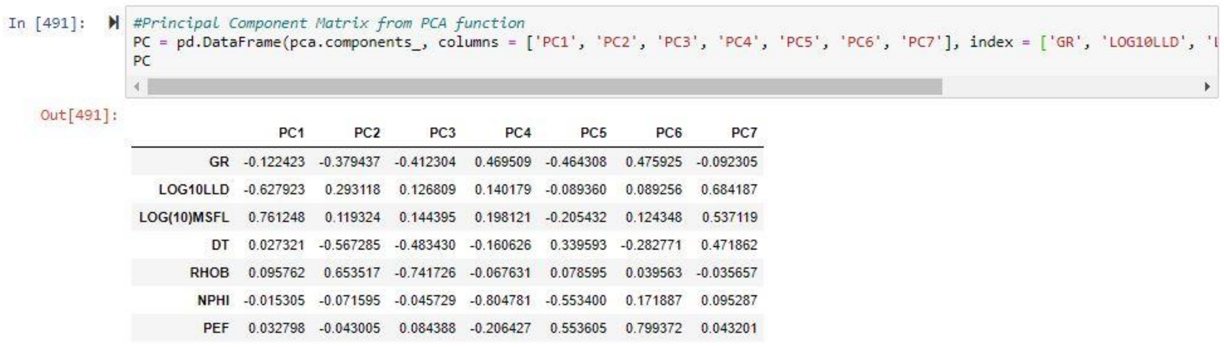
The matrix 'v' generated from the singular value decomposition is the principal component matrix(Fig 8)

The principal components were verified by performing Principal Component Analysis using an inbuilt PCA function(Fig 9).



```
#Using PCA function to verify principal components
pca = PCA(svd_solver = "full")
pca.fit_transform(standard_wells)
```

Figure 9 : PCA - Using PCA function



```
In [491]:  #Principal Component Matrix from PCA function
           PC = pd.DataFrame(pca.components_, columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'], index = ['GR', 'LOG10LLD', 'L
           PC
```

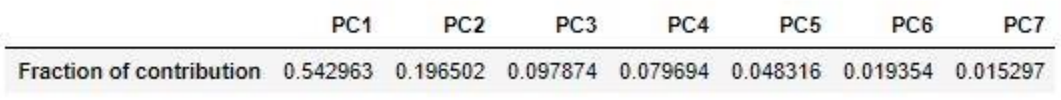
Out[491]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
GR	-0.122423	-0.379437	-0.412304	0.469509	-0.464308	0.475925	-0.092305
LOG10LLD	-0.627923	0.293118	0.126809	0.140179	-0.089360	0.089256	0.684187
LOG(10)MSFL	0.761248	0.119324	0.144395	0.198121	-0.205432	0.124348	0.537119
DT	0.027321	-0.567285	-0.483430	-0.160626	0.339593	-0.282771	0.471862
RHOB	0.095762	0.653517	-0.741726	-0.067631	0.078595	0.039563	-0.035657
NPHI	-0.015305	-0.071595	-0.045729	-0.804781	-0.553400	0.171887	0.095287
PEF	0.032798	-0.043005	0.084388	-0.206427	0.553605	0.799372	0.043201

Figure 10 : Principal components through PCA function

The components generated were equal to the values from our calculations, as we can observe from Figures 8 and 10.

We proceed with plotting the total contribution of the different principal components towards the variance of the data, in order to analyze the different components.



	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Fraction of contribution	0.542963	0.196502	0.097874	0.079694	0.048316	0.019354	0.015297

Figure 11 : Contribution fractions for each of the principal components

The total contribution by the first 5 components equals to a fraction of 0.9653, which is about 96.5%. Hence the first 5 components are considered as they explain 96.5% of the variance.

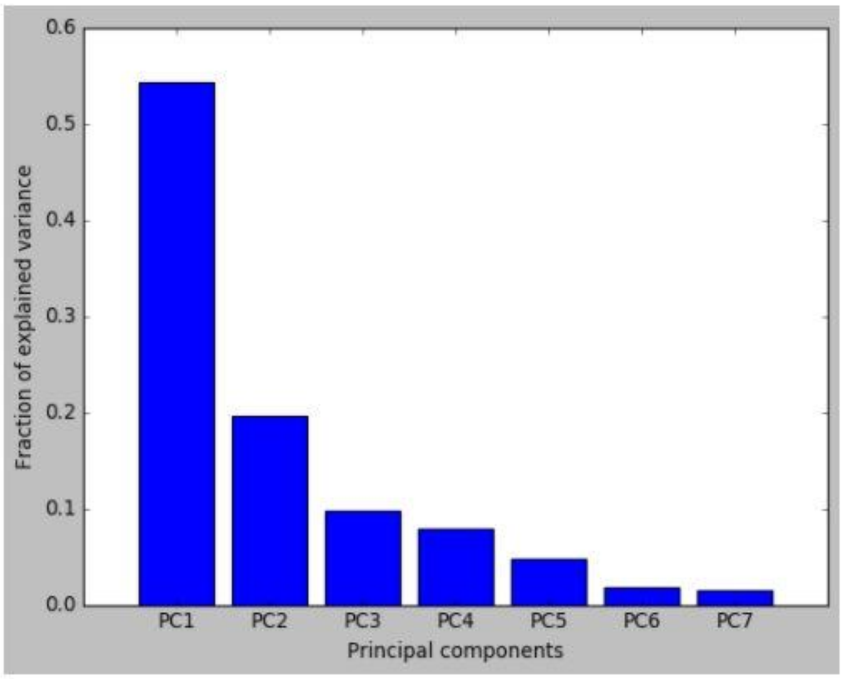


Figure 12 : Fraction of explained variance for each principal component

The fraction of variance explained by each principal component variable is graphed on a bar plot in Fig 12. As we can observe, the last 2 components PC6 and PC7 contribute very less relative to the first 5 principal components.

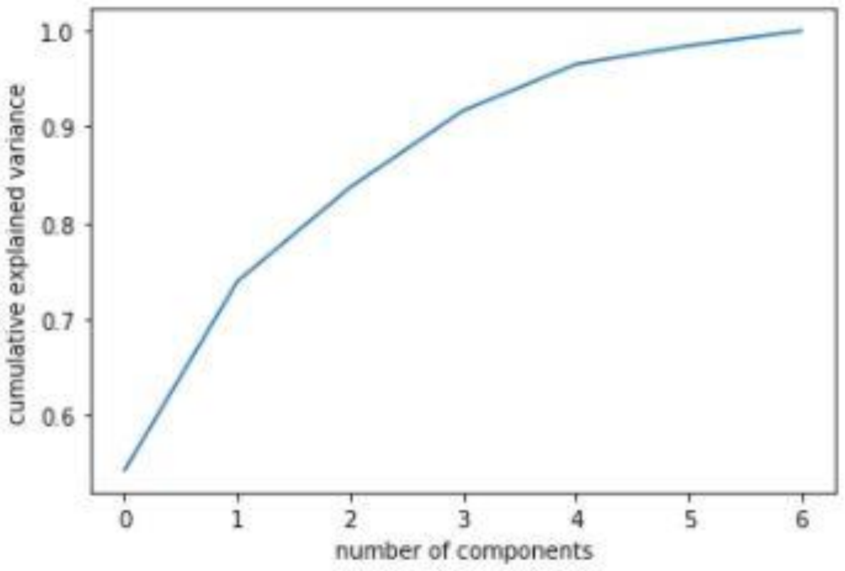
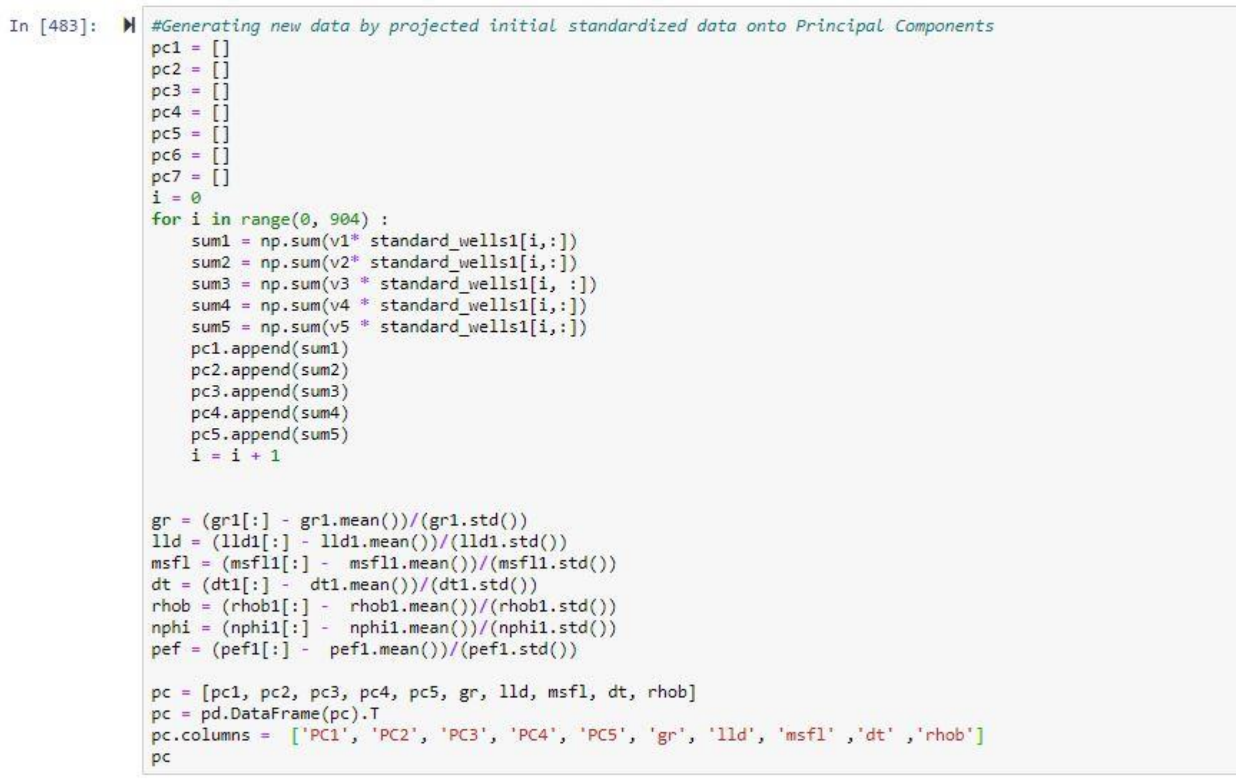


Figure 13 : Cumulative Explained Variance for number of components

The cumulative explained variance is graphed in the Fig 13. The appropriate number of components is taken as 5 from the figures 13 and 12. The components PC1, PC2, PC3 , PC4 and PC5 are considered for further analysis.



```
In [483]:  #Generating new data by projected initial standardized data onto Principal Components
           pc1 = []
           pc2 = []
           pc3 = []
           pc4 = []
           pc5 = []
           pc6 = []
           pc7 = []
           i = 0
           for i in range(0, 904) :
               sum1 = np.sum(v1* standard_wells1[i,:])
               sum2 = np.sum(v2* standard_wells1[i,:])
               sum3 = np.sum(v3 * standard_wells1[i, :])
               sum4 = np.sum(v4 * standard_wells1[i,:])
               sum5 = np.sum(v5 * standard_wells1[i,:])
               pc1.append(sum1)
               pc2.append(sum2)
               pc3.append(sum3)
               pc4.append(sum4)
               pc5.append(sum5)
               i = i + 1


           gr = (gr1[:] - gr1.mean())/(gr1.std())
           lld = (lld1[:] - lld1.mean())/(lld1.std())
           msfl = (msfl1[:] -  msfl1.mean())/(msfl1.std())
           dt = (dt1[:] -  dt1.mean())/(dt1.std())
           rhob = (rhob1[:] -  rhob1.mean())/(rhob1.std())
           nphi = (nphi1[:] -  nphi1.mean())/(nphi1.std())
           pef = (pef1[:] -  pef1.mean())/(pef1.std())

           pc = [pc1, pc2, pc3, pc4, pc5, gr, lld, msfl, dt, rhob]
           pc = pd.DataFrame(pc).T
           pc.columns =  ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'gr', 'lld', 'msfl' ,'dt' ,'rhob']
           pc
```

Figure 14 : Projecting well log data onto principal component axes

The well data taken is then projected onto the principal component axes to generate transformed data. This is achieved by manually slicing the principal component matrix.

The scatterplots are then generated for 5 principal components, and their correlation with the normalized well log data, in Fig 15

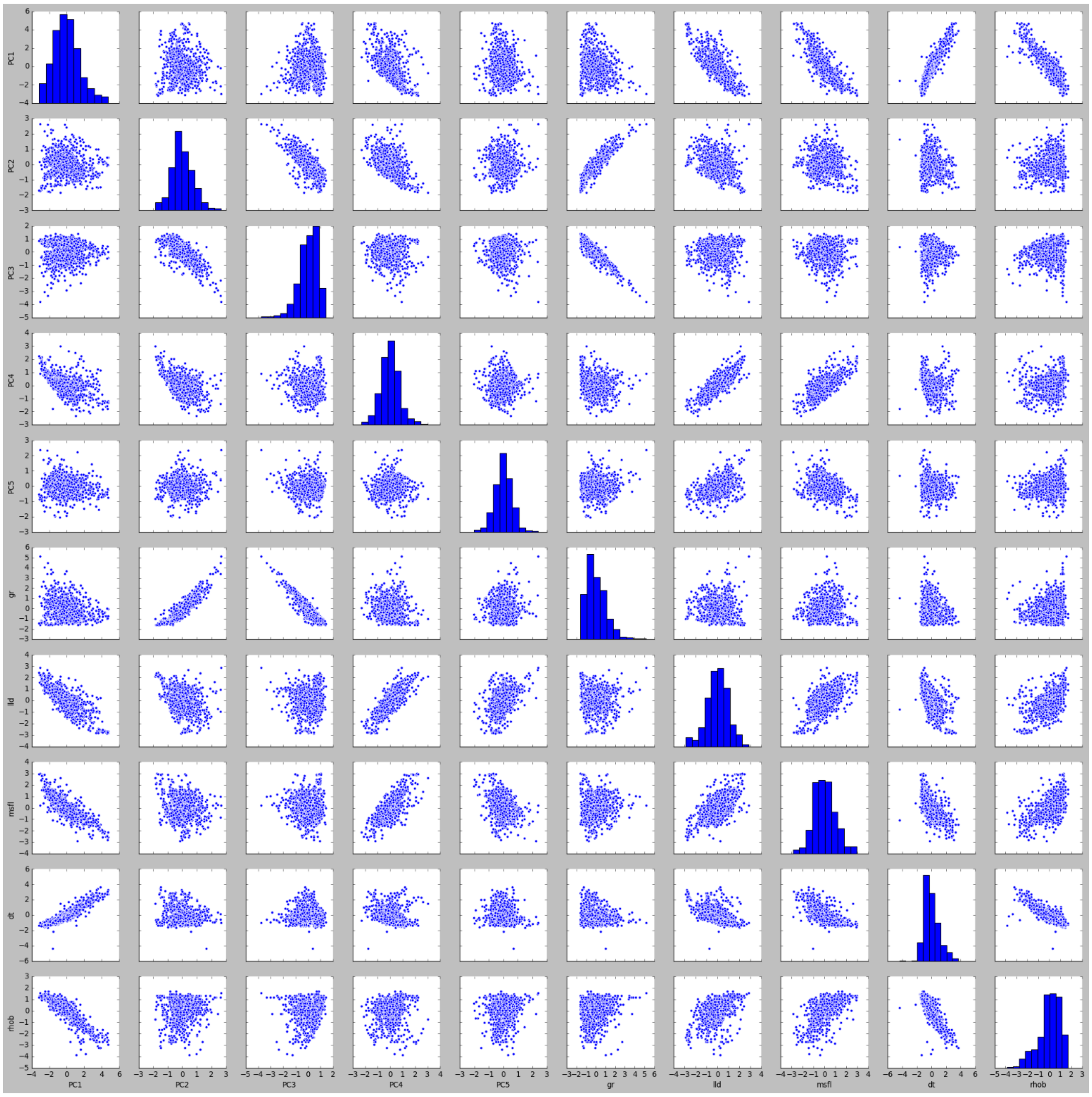


Figure 15 : PCA scatterplots

The scatterplots for each pair of variables are generated , and the correlation of data can be observed from the scatterplots for each variable pair. For example, Positive correlations can be observed in the pairs of GR-PC2 and DT-PC1 while negative correlations can be observed in GR-PC3.

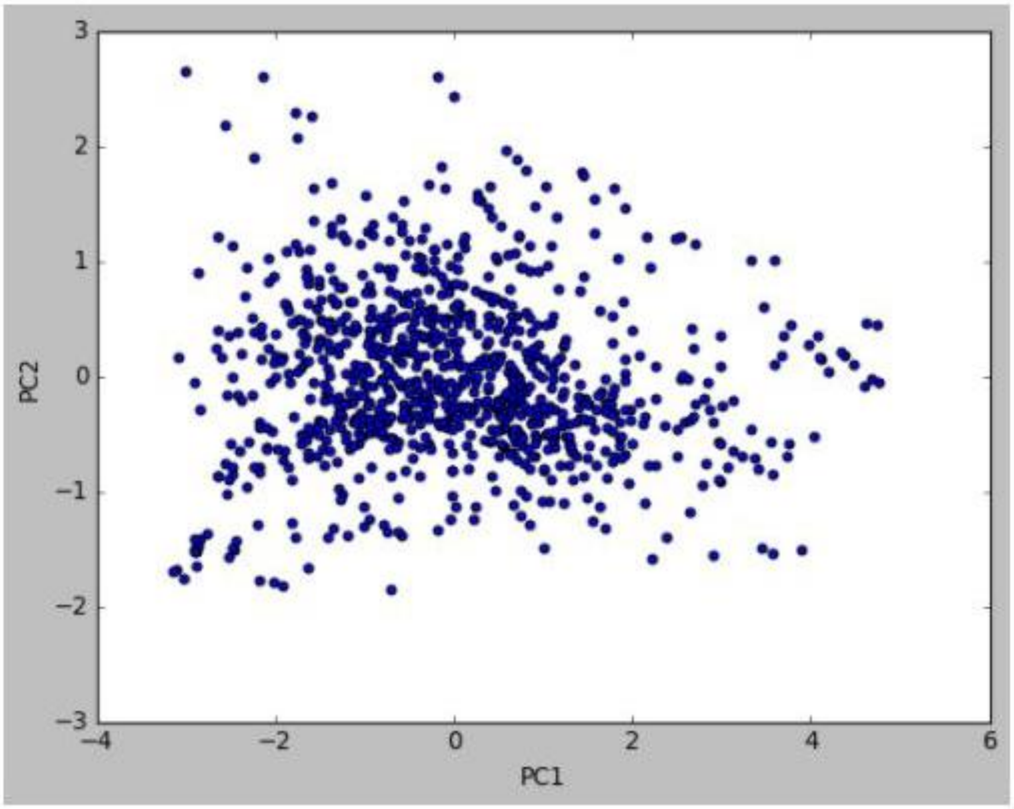


Figure 16 : Scatter plot between PC1 and PC2

The sum of squared error(SSE) curve is generated for the number of clusters considered during k_means clustering. (Fig 17)

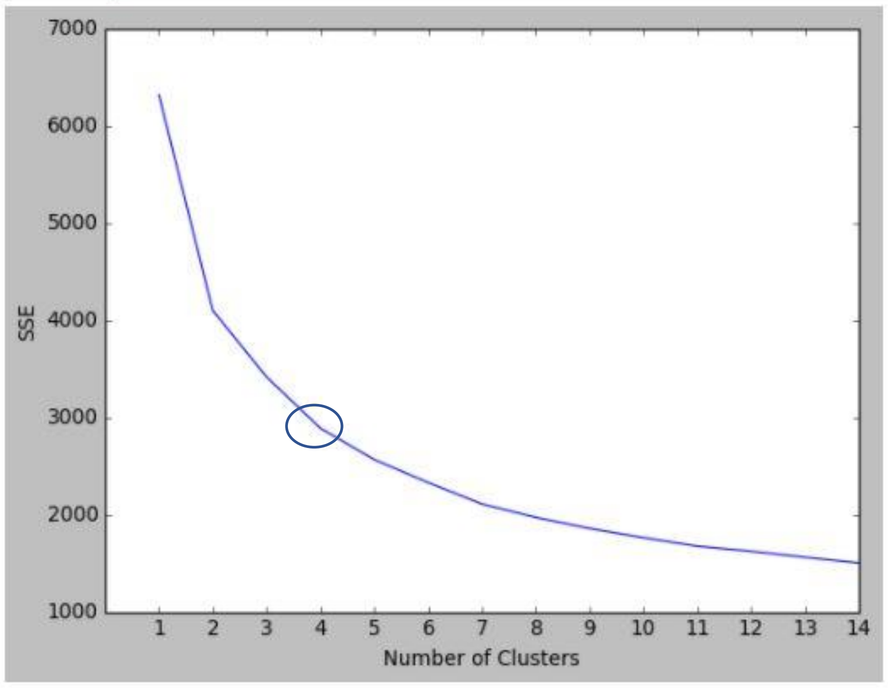
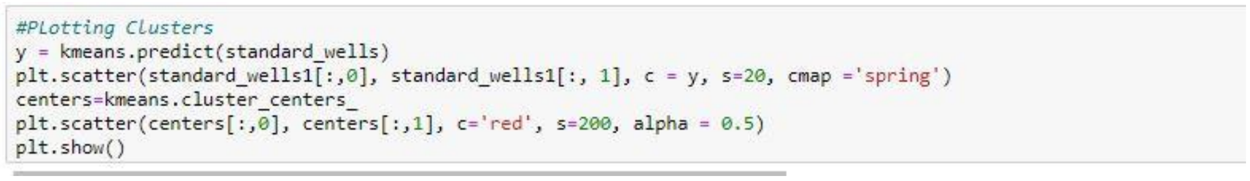


Figure 17 : Sum of Squared error plot vs number of clusters

From Fig 17, we observe the elbow of the plot at about 4 clusters. Hence, we consider 4 clusters for the K-means clustering.



```
#PLotting Clusters
y = kmeans.predict(standard_wells)
plt.scatter(standard_wells1[:,0], standard_wells1[:, 1], c = y, s=20, cmap ='spring')
centers=kmeans.cluster_centers_
plt.scatter(centers[:,0], centers[:,1], c='red', s=200, alpha = 0.5)
plt.show()
```

Figure 18 : Plotting clusters with cluster centers – Code

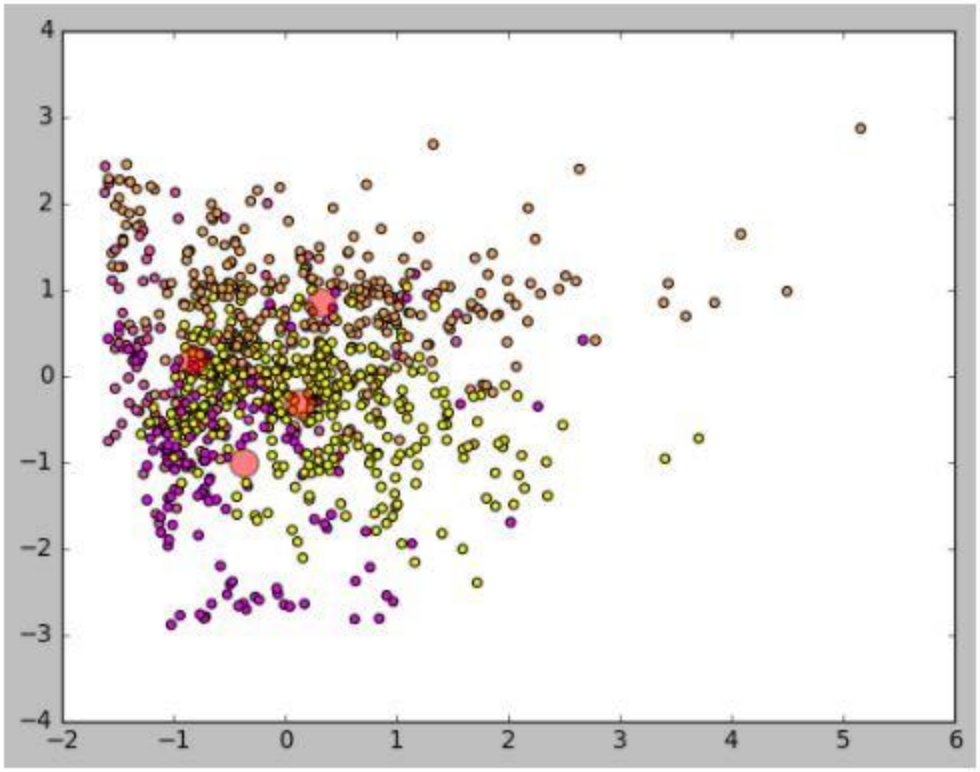


Figure 19 : Cluster map - 4 cluster Centers

The K-means clustering algorithm is run on the standardized dataset, and the appropriate number of components is found to be 4, from Fig 17. The cluster map is generated for first 2 well logs with the 4 cluster centroids indicated with red circles in Fig 19.

EXECUTIVE SUMMARY

The original dataset provided to us in refined to include the GR, log10(LLD), log10(MSFL), DT, RHOB, NPHI and PEF well logs. This dataset is then normalized such that the mean is 0 and the variance is 1. The normalized dataset is then used to generate a covariance matrix, and singular value decomposition (SVD) is performed on the covariance matrix in order to decompose it into U,S, and V matrices, which represent the left and right singular vectors and the singular values.

The matrix ‘V’ contains the principal axes component values. An in-built PCA function is used to verify the principal component matrix. After verification, the principal components are characterized based on the fractions of their total contribution and their cumulative total contributions. The number of components was chosen as 5 after analyzing the principal component matrix and noting the total contribution. These values are used to transform the original log data into the principal component axes.

The scatterplots for pairs of features are generated in Fig 15, also plotting the principal components with the normalized data, to observe the correlations between the variables.

The K-means clustering is performed on the normalized dataset. A SSE plot from Fig 17 shows the knee of the elbow at 4 clusters, hence 4 clusters are chosen for the cluster analysis. Fig 19 is the cluster map generated for an example pair of variables.