MUSIC GENRE CLASSIFICATION

Aayush Sagar CB.EN.U4ELC20002
Hari Varsha CB.EN.U4ELC20021
Naveen US CB.EN.U4ELC20043

References

- https://www.geeksforgeeks.org/ml-determine-the-optimal-value-of-k-in-k-means-clustering/
- https://www.geeksforgeeks.org/silhouette-algorithm-to-determine-the-optimal-value-of-k/?ref=rp
- https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/?ref=rp
- https://www.w3schools.com/python/matplotlib_scatter.asp
- Colab link: https://colab.research.google.com/drive/1XXyqGkhqIadtbkB65wlMlxsqxKgqi2b9?usp=sharing

PCA application (10 marks)

- Normalisation is done before PCA as it projects the original data onto directions which maximize the variance and brings all features at the same scale.
- Eigenvalues represent the total amount of variance that can be explained by a given principal component.
- Suitable eigen values are selected and insignificant features are dropped.
- Covariance matrix is calculated before and after PCA.

$$C = \begin{bmatrix} \mathrm{cov}(X,X) & \mathrm{cov}(X,Y) & \mathrm{cov}(X,Z) \\ \mathrm{cov}(Y,X) & \mathrm{cov}(Y,Y) & \mathrm{cov}(Y,Z) \\ \mathrm{cov}(Z,X) & \mathrm{cov}(Z,Y) & \mathrm{cov}(Z,Z) \end{bmatrix}$$

Finding the covariance matrix

```
[3] import pandas as pd
    from sklearn import datasets
    import matplotlib.pyplot as plt
    import numpy as np

    dataset = pd.read_csv('music_samples.csv')
    dataset.head(5)
```

1 to 5 of 5 entries Filter

index	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var	rolloff_mean	rolloff_\
0	66149	0.335406363	0.091048293	0.130405024	0.003521004	1773.065032	167541.6309	1972.744388	117335.7716	3714.560359	1080789.
1	66149	0.343065351	0.086146526	0.112699248	0.001449685	1816.693777	90525.69087	2010.051501	65671.87567	3869.682242	672244.7
2	66149	0.346814752	0.092242889	0.132003382	0.004620399	1788.539719	111407.4376	2084.565132	75124.92172	3997.63916	790712.6
3	66149	0.363638788	0.086856157	0.132564723	0.002447563	1655.289045	111952.2845	1960.039988	82913.63927	3568.300218	921652.4
4	66149	0.335579425	0.088128544	0.143288806	0.001700886	1630.656199	79667.26765	1948.503884	60204.02027	3469.992864	610211.0

```
[4] cols = dataset.columns.tolist()
    cols.insert(0, cols.pop(cols.index('label')))
    dataset = dataset.reindex(columns= cols)
    X = dataset.iloc[:,1:59].values
    y = dataset.iloc[:,0].values
```

```
[5] from sklearn.preprocessing import StandardScaler
    X_std = StandardScaler().fit_transform(X)
```

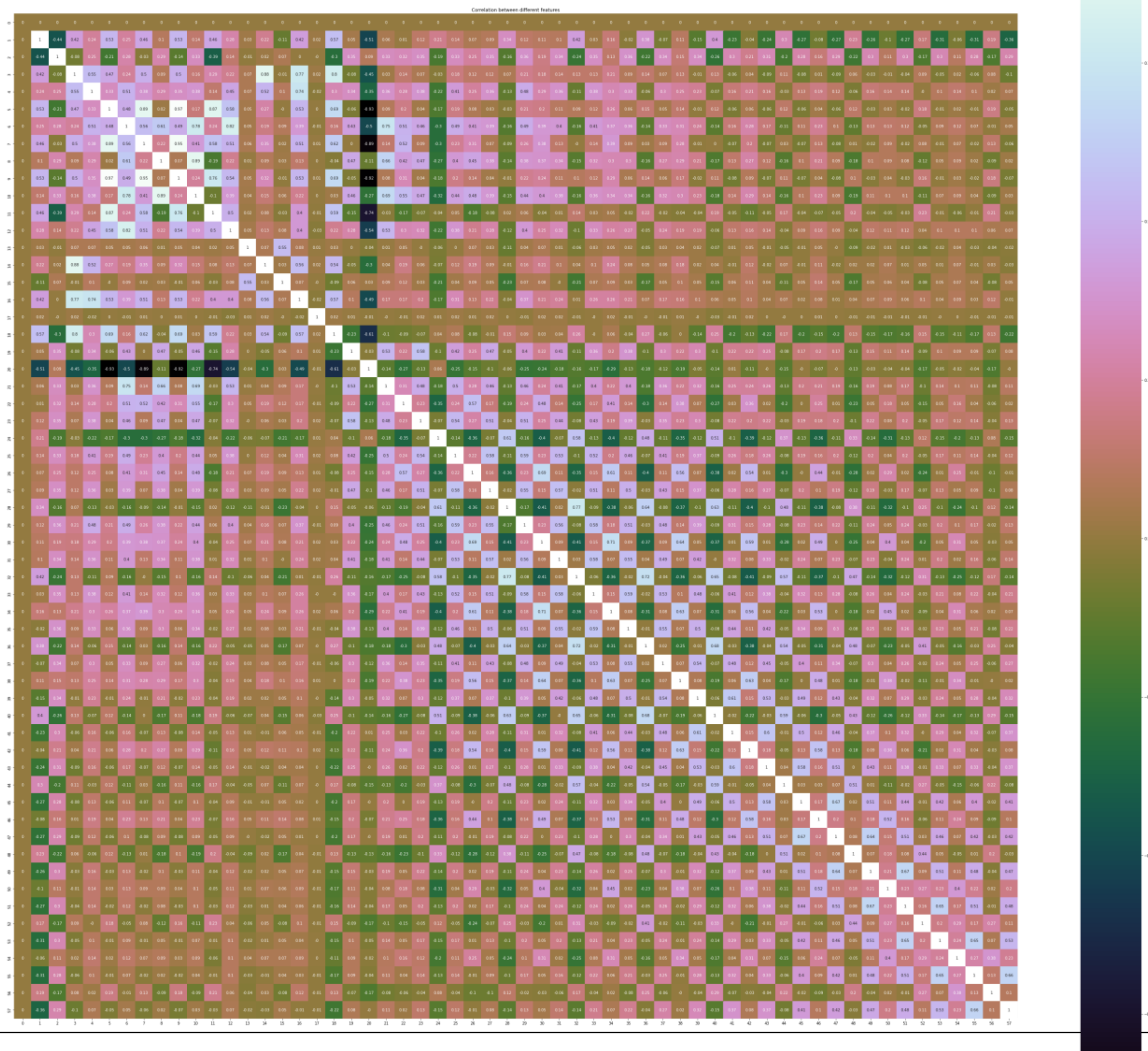
```
#Finding Co-variance matrix of actual dataset
data = pd.DataFrame(X_std)
covMatrix=pd.DataFrame.cov(data)
covMatrix=np.round(covMatrix, decimals=2)

print("\nCo-variance matrix of actual dataset after normalization\n",covMatrix)
```

```
2    0.30  0.11  0.30 -0.17  0.30  0.11  0.28 -0.17  0.29
3   -0.03 -0.01 -0.04  0.09 -0.05  0.02 -0.06  0.08 -0.10
4    0.16  0.14  0.14 -0.00  0.10  0.14  0.10  0.02  0.07
5   -0.03  0.03 -0.02  0.18 -0.01  0.02 -0.01  0.19 -0.05
6    0.13  0.13  0.12 -0.05  0.09  0.12  0.07 -0.01  0.05
7   -0.02  0.09 -0.02  0.08 -0.01  0.07 -0.02  0.13 -0.06
8    0.10  0.09  0.08 -0.12  0.05  0.09  0.02 -0.09  0.02
9   -0.03  0.04 -0.03  0.16 -0.01  0.03 -0.02  0.18 -0.07
10   0.11  0.10  0.10 -0.11  0.07  0.09  0.04 -0.09  0.03
11  -0.04 -0.05 -0.03  0.23 -0.01 -0.06 -0.01  0.21 -0.03
12   0.12  0.11  0.12  0.04  0.10  0.10  0.10  0.06  0.07
13  -0.02  0.01 -0.03 -0.06 -0.02  0.04 -0.03 -0.04 -0.02
14   0.02  0.07  0.01  0.05  0.01  0.07 -0.01  0.03 -0.03
15   0.05  0.06  0.04 -0.08  0.05  0.07  0.04 -0.08  0.05
16   0.07  0.09  0.06  0.10  0.04  0.09  0.03  0.12 -0.01
```

Heatmap of covariance matrix before PCA

```
[22] import seaborn as sns
     plt.figure(figsize=(58,58))
     sns.heatmap(covMatrix, vmax=1, square=True,annot=True,cmap='cubehelix')

     plt.title('Covariance between different features')
```

```
plt.title('Correlation between different features')
```



Finding the eigen values and eigen vector

```
eig_vals, eig_vecs = np.linalg.eig(covMatrix)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.          0.          0.         ...  0.          0.
   1.        ]
 [-0.01085409  0.27966744  0.0141295  ...  0.00726114 -0.07511056
   0.        ]
 [-0.14625596 -0.14898455  0.04462647 ...  0.15776689 -0.15613246
   0.        ]
 ...
 [-0.09155678 -0.10639622  0.14098028 ... -0.01168182  0.12998387
   0.        ]
 [ 0.0186369   0.09058497  0.05793939 ... -0.10315841  0.08035949
   0.        ]
 [-0.08819064 -0.12538107  0.12995034 ...  0.00838401 -0.0053148
   0.        ]]

Eigenvalues
[1.14229517e+01 7.72275973e+00 5.89736407e+00 3.75969252e+00
 2.48752010e+00 2.09365023e+00 1.68501391e+00 1.51363860e+00
 1.43054219e+00 1.22005260e+00 1.05794669e+00 9.93506315e-01
 9.70156630e-01 8.46588631e-01 7.74195395e-01 6.70581746e-01
 6.37889716e-01 6.14520040e-01 1.16266684e-03 4.50521943e-03
 8.89535206e-03 3.61425840e-02 5.14140681e-02 6.42899843e-02
 1.08382313e-01 5.56539279e-01 1.45750501e-01 5.21600738e-01
 1.64340114e-01 4.99132980e-01 1.80923039e-01 1.93651640e-01
 4.85493720e-01 4.75929800e-01 2.13634950e-01 4.59811661e-01
 2.31087100e-01 2.32731997e-01 2.41958992e-01 2.55589388e-01
 2.57717852e-01 2.71492942e-01 2.84736884e-01 2.97382323e-01
 3.08896780e-01 4.46151146e-01 4.35780145e-01 4.22884799e-01
 4.17479201e-01 3.28071897e-01 3.35579308e-01 3.47975884e-01
 3.69683948e-01 3.66554486e-01 4.00031639e-01 3.87077554e-01
 3.90964313e-01 0.00000000e+00]
```

Listing out the index of eigen values that are less than one

```
for i in range(len(eig_vals)):
  temp = eig_vals[i]
  if temp<1:
    print (i)
```

```
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Dropping the eigen vectors having eigen values less than one

```
[ ] d=pd.DataFrame(eig_vecs)
    d=d.drop([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,
    d.shape
```

```
(58, 11)
```

```
[ ] #PCA algorithm
    a = np.asarray(d)
    a=np.transpose(a)
    data_modd=np.transpose(X_std) #Finding the transpose of the actual dataset
    res=np.dot (a, data_modd) #Dot - product
    data=np.transpose(res) #Assigning the transpose of the result value to the new variable #updated dataset after removing insignificant features
    data=pd.DataFrame(data) #Converting into new dataset with the help of DataFrame
    print("The new dataset after removing insignificant features")
    data.head()
    data.shape
```

```
The new dataset after removing insignificant features
(9990, 11)
```

```
covMatrix1.shape
```

Co-variance matrix of actual dataset after normalization

	0	1	2	3	4	5	6	7	8	9	...	48 \
0	0.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00
1	0.0	1.00	-0.44	0.42	0.24	0.53	0.25	0.46	0.10	0.53	...	0.23
2	0.0	-0.44	1.00	-0.08	0.25	-0.21	0.28	-0.03	0.29	-0.14	...	-0.22
3	0.0	0.42	-0.08	1.00	0.55	0.47	0.24	0.50	0.09	0.50	...	0.06
4	0.0	0.24	0.25	0.55	1.00	0.33	0.51	0.38	0.29	0.35	...	-0.06
5	0.0	0.53	-0.21	0.47	0.33	1.00	0.48	0.89	0.02	0.97	...	0.12
6	0.0	0.25	0.28	0.24	0.51	0.48	1.00	0.56	0.61	0.49	...	-0.13
7	0.0	0.46	-0.03	0.50	0.38	0.89	0.56	1.00	0.22	0.95	...	0.01
8	0.0	0.10	0.29	0.09	0.29	0.02	0.61	0.22	1.00	0.07	...	-0.18
9	0.0	0.53	-0.14	0.50	0.35	0.97	0.49	0.95	0.07	1.00	...	0.10
10	0.0	0.14	0.33	0.16	0.38	0.17	0.78	0.41	0.89	0.24	...	-0.19
11	0.0	0.46	-0.39	0.29	0.14	0.87	0.24	0.58	-0.19	0.76	...	0.20
12	0.0	0.28	0.14	0.22	0.45	0.58	0.82	0.51	0.22	0.54	...	-0.04
13	0.0	0.03	-0.01	0.07	0.07	0.05	0.05	0.06	0.01	0.05	...	-0.09
14	0.0	0.22	0.02	0.88	0.52	0.27	0.19	0.35	0.09	0.32	...	0.02

Covariance Matrix after PCA

```
[ ] sum=0
    for i in range(len (covMatrix1)):
      for j in range(len (covMatrix1)):
        if i>j:
          sum += covMatrix1[i][j]
        sum-round (sum, 2)
    print("Sum of upper triangle covariance matrix before PCA :", sum)
```

```
Sum of upper triangle covariance matrix before PCA : 170.87999999999997
```

```
#Finding Co-variance matrix of actual dataset
data = pd.DataFrame(data)
covMatrix=pd.DataFrame.cov(data)
covMatrix=np.round(covMatrix, decimals=2)
print("\nCo-variance matrix of actual dataset after normalization\n",covMatrix)
covMatrix.shape
```

```
Co-variance matrix of actual dataset after normalization
        0     1     2     3     4     5     6     7     8     9     10
0   11.43  0.00 -0.00 -0.00  0.00 -0.00  0.00 -0.00 -0.00  0.00  0.01
1    0.00  7.72  0.00  0.00 -0.00  0.00 -0.00 -0.00  0.00 -0.01  0.00
2   -0.00  0.00  5.90  0.00  0.00 -0.01  0.00 -0.00  0.00 -0.00 -0.00
3   -0.00  0.00  0.00  3.76 -0.00  0.00 -0.00 -0.01  0.00 -0.01 -0.00
4    0.00 -0.00  0.00 -0.00  2.48 -0.01  0.00  0.00  0.00 -0.00  0.00
5   -0.00  0.00 -0.01  0.00 -0.01  2.10  0.00  0.00  0.00  0.00  0.00
6    0.00 -0.00  0.00 -0.00  0.00  0.00  1.68  0.00  0.00  0.00 -0.00
7   -0.00 -0.00 -0.00 -0.01  0.00  0.00  0.00  1.51  0.00  0.00 -0.00
8   -0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.42  0.00 -0.00
9    0.00 -0.01 -0.00 -0.01 -0.00  0.00  0.00  0.00  0.00  1.22 -0.00
10   0.01  0.00 -0.00 -0.00  0.00  0.00 -0.00 -0.00 -0.00 -0.00  1.06
(11, 11)
```

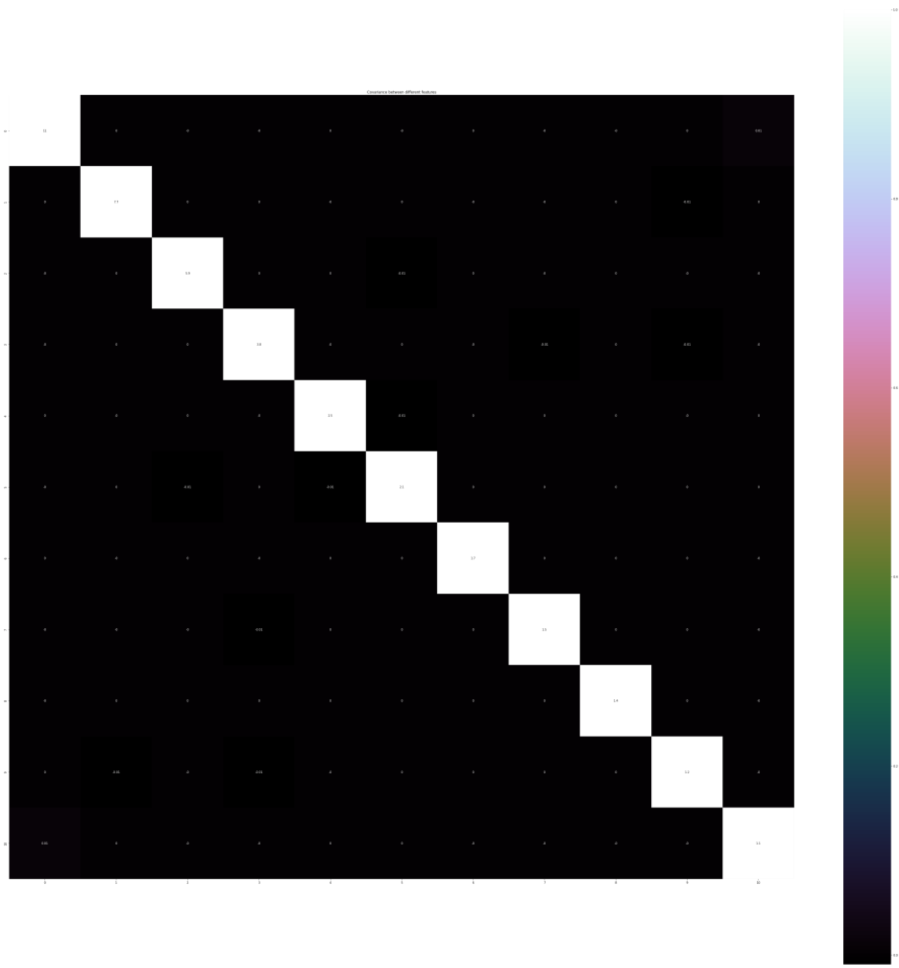
```
[ ] sum1=0
    for i in range(len (covMatrix)):
      for j in range(len (covMatrix)):
        if i>j:
          sum1 += covMatrix[i][j]
        sum1-round (sum1, 2)
    print("Sum of upper triangle covariance matrix after PCA :", sum1)
```

```
Sum of upper triangle covariance matrix before PCA : -0.04
```

PCA inference

- The idea behind Principal Component Analysis is "Dimensionally reduction" - to reduce the features which has low to zero impact in the dataset.
- The eleven highest eigen values from all the eigen values of the dataset were selected as they can explain most of the variance in the data.
- Before application of PCA, the dataset was a 9990x58 matrix.
- The sum of the upper triangular matrix was 170.88 before application of PCA.
- After application of PCA, the dataset was reduced to a 9990x11 matrix.
- The sum of the upper triangular matrix was –0.04 after application of PCA.
- It can be inferred from the sum of upper triangular matrix that the covariance among any two features have drastically reduced.

PCA inference



From the heat covariance
heat map after PCA
we can see that
That the covariance
between any two
Principle component
is very low

K Means Clustering (10 marks)

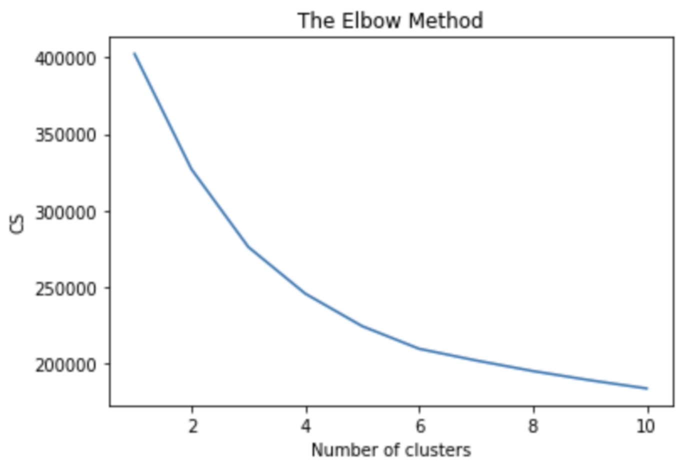
- K Means clustering is applied to the principal components of the Dataset after PCA.
- The effect of PCA on K Means clustering is K means accuracy increased after the dimension reduction.
- The effect of normalisation is it increases the clustering quality and helps to improve the speed, accuracy, and efficiency of the database.

K Means Clustering

```
[14] #K-Means model with two clusters
     from sklearn.cluster import KMeans
     kmeans = KMeans(n_clusters=2, random_state=0)
     kmeans.fit(data)
     kmeans.cluster_centers_
     kmeans.inertia_
```

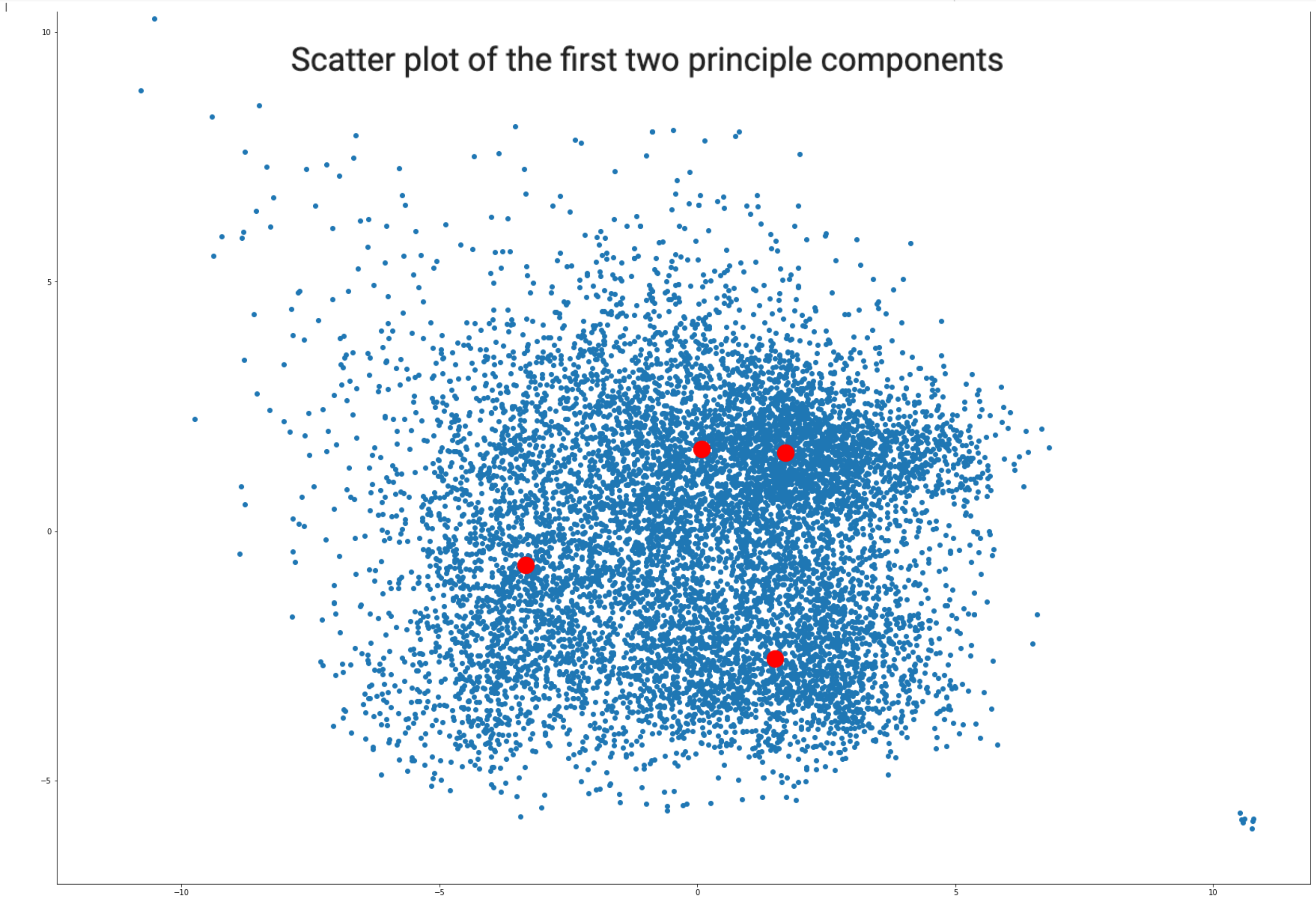
```
327094.1444743024
```

```
from sklearn.cluster import KMeans
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(data)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```

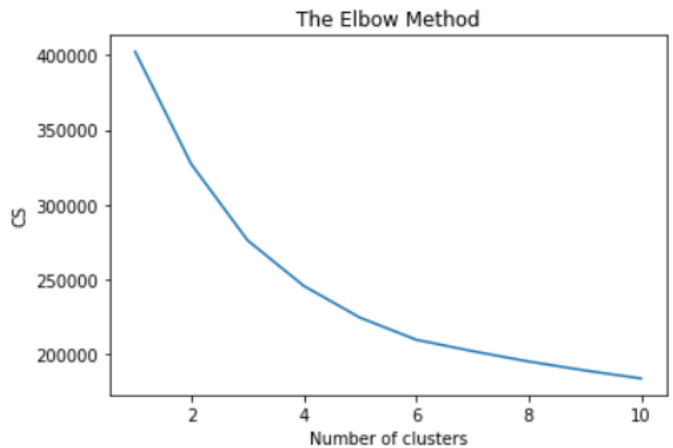


```
[ ] from sklearn.cluster import KMeans

    kmeans = KMeans(n_clusters=4,random_state=0)
    kmeans.fit(data)
    labels = kmeans.labels_
```

```
plt.figure(figsize=(30,30))
plt.scatter(data[1], data[2])
plt.scatter(kmeans.cluster_centers_[:, 1], kmeans.cluster_centers_[:, 2], s=500, c='red')
plt.show()
```



K Means Clustering inference



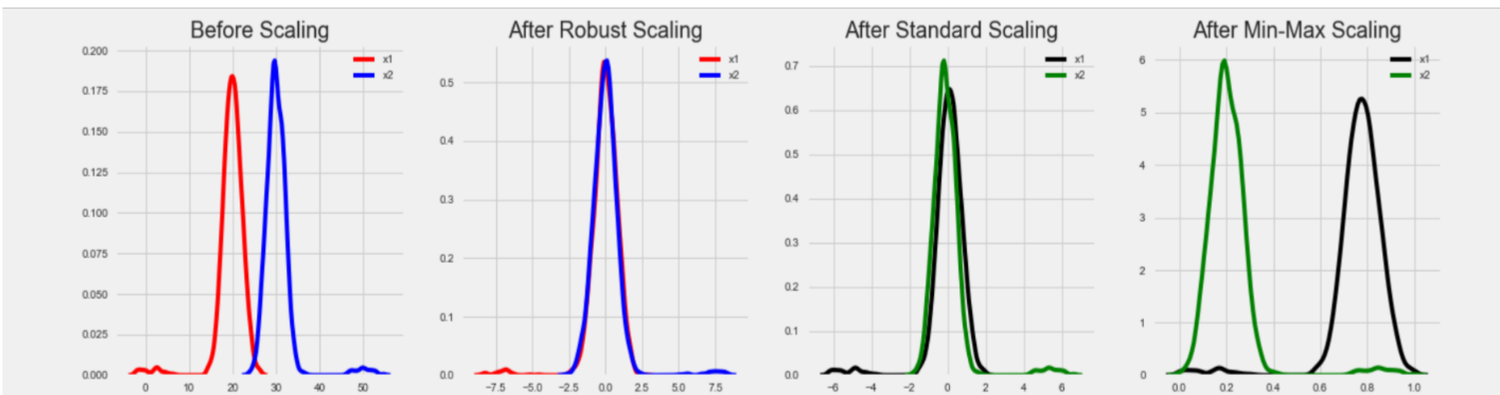
- The elbow method was used to find the optimal K value for applying the K-Means clustering algorithm.
- K=4 was chosen as the K value for clustering by seeing the curve.

Clustering inference

- The data was then plotted into a scatter plot and distributed into four clusters; decent clustering was observed for certain set of principal components.
- The original dataset has 10 predefined labels meant for supervised learning but when K-Means clustering (an unsupervised learning algorithm) is applied on the dataset, it reduces the labels to 4. Thus, it can be concluded that the K-Means clustering algorithm is not a suitable fit for this particular-dataset.

Miscellaneous

- The different types of scaling like Robust scaling, Standard scaling, Min-Max scaling were understood, and the Standard Scaler was implemented for normalizing the data. - Reference link.



$$\frac{\mathbf{x_i - Q_1(x)}}{\mathbf{Q_3(x) - Q_1(x)}}$$

$$x' = \frac{x - \bar{x}}{\sigma}$$

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Miscellaneous

- An Alternate method - the silhouette algorithm, to find the optimal K-value before applying K-Means clustering.- https://www.kaggle.com/code/funxexcel/p2-sklearn-k-means-elbow-and-silhouette-method/notebook

Optimal K value using Silhouette algorithm

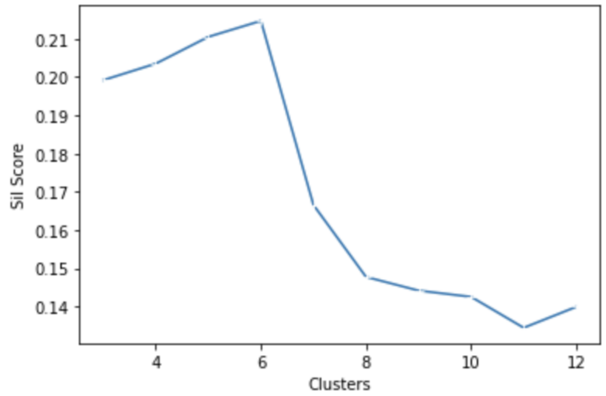
[17]

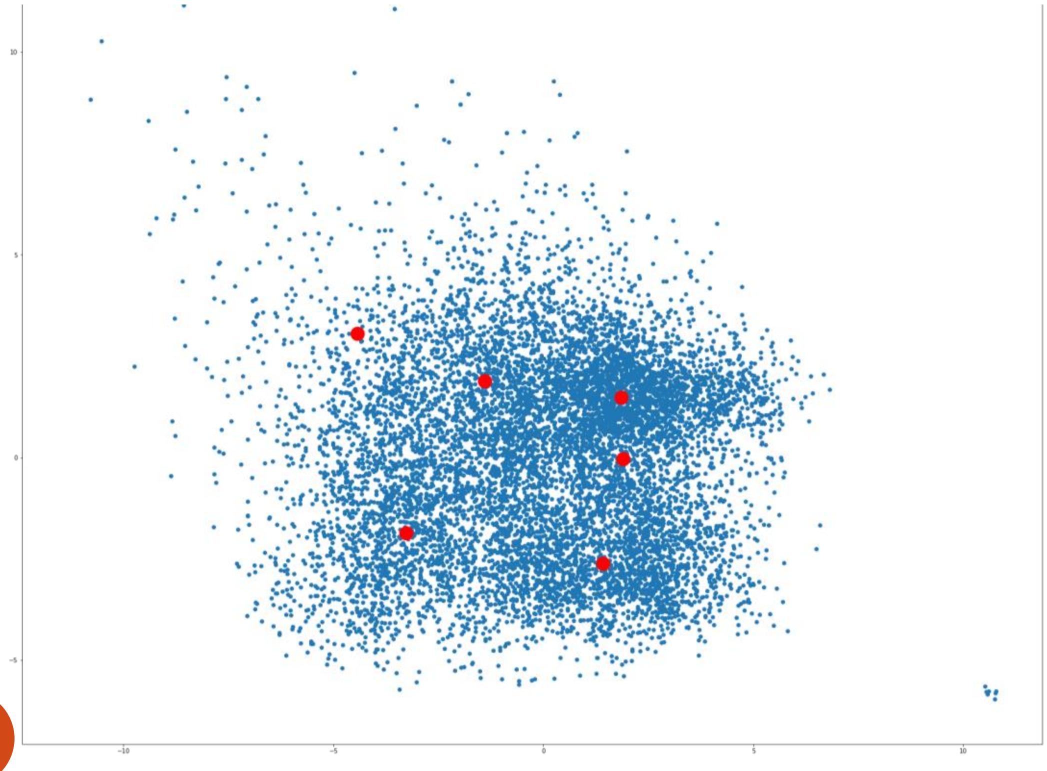
```
import sklearn.metrics as metrics
import sklearn.cluster as cluster

SK = range(3,13)
sil_score = []
for i in SK:
    labels=cluster.KMeans(n_clusters=i,init="k-means++",random_state=200).fit(data).labels_
    score = metrics.silhouette_score(data,labels,metric="euclidean",sample_size=1000,random_state=200)
    sil_score.append(score)
    print ("Silhouette score for k(clusters) = "+str(i)+" is "
           +str(metrics.silhouette_score(data,labels,metric="euclidean",sample_size=1000,random_state=200)))
```

```
Silhouette score for k(clusters) = 3 is 0.19917714634535774
Silhouette score for k(clusters) = 4 is 0.20350306830513312
Silhouette score for k(clusters) = 5 is 0.21050556456444933
Silhouette score for k(clusters) = 6 is 0.21468768647843395
Silhouette score for k(clusters) = 7 is 0.1665354059090207
Silhouette score for k(clusters) = 8 is 0.14772112366370566
Silhouette score for k(clusters) = 9 is 0.14414638157583776
Silhouette score for k(clusters) = 10 is 0.1425089138113267
Silhouette score for k(clusters) = 11 is 0.13441458072570267
Silhouette score for k(clusters) = 12 is 0.13989401140389154
```

```
sil_centers = pd.DataFrame({'Clusters' : SK, 'Sil Score' : sil_score})
sns.lineplot(x = 'Clusters', y = 'Sil Score', data = sil_centers, marker="+")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ae2a16a50>
```

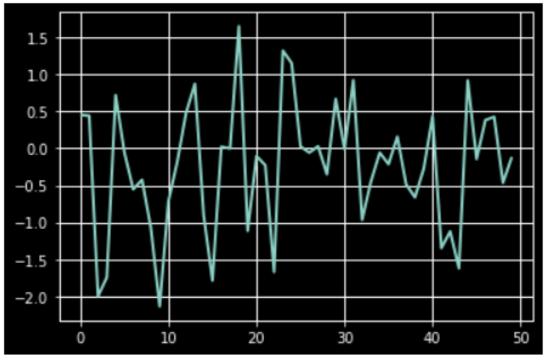




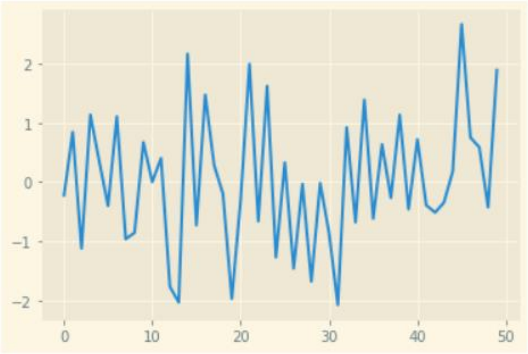
K = 6 is chosen as it has the highest silhouette score.

Miscellaneous

- The various built-in styles in style package of matplotlib was explored.- https://www.geeksforgeeks.org/style-plots-using-matplotlib/



dark_background



Solarize_Light2

Miscellaneous

- Plotting clusters using Scatter plot was learnt - https://stackoverflow.com/questions/12487060/matplotlib-color-according-to-class-labels
- Plotting of heat maps using seaborn was learnt - https://blog.quantinsti.com/creating-heatmap-using-python-seaborn/