Recommendation of Music System

```
# Import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import cosine_similarity
import seaborn as sns

import statsmodels.api as sm
from warnings import filterwarnings
import os

from scipy.spatial.distance import pdist, squareform
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
filterwarnings('ignore')

# Load the dataset
data_main = pd.read_csv('top 100 streamed
songs.csv').drop(columns=['id'])
data = data_main.drop(columns=['name'])
data_main.head()
```

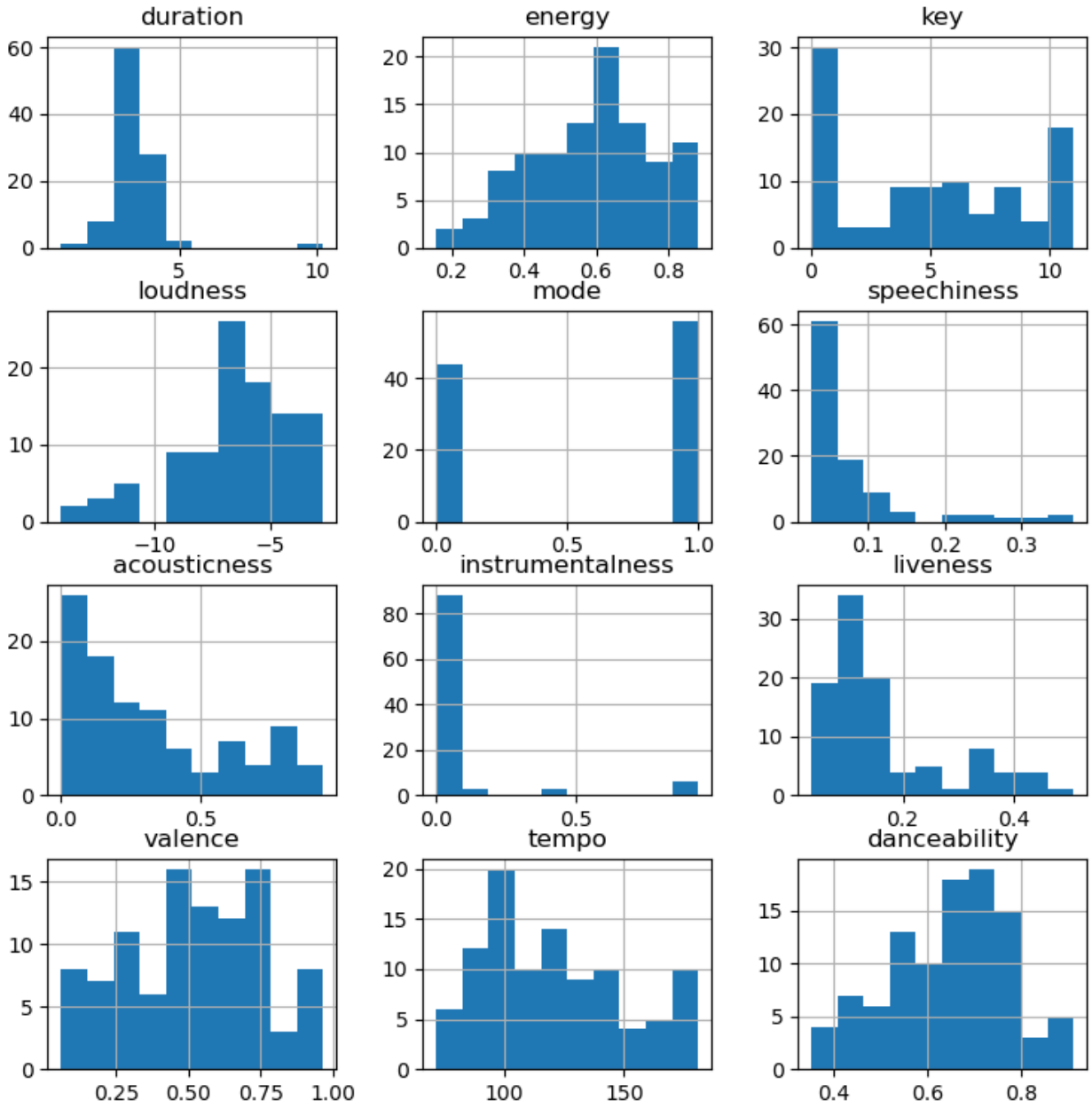
```
                                                name  duration  energy
key  \
0                              Good 4 U Olivia Rodrigo      2.97   0.664
9
1                   Stay The Kid LAROI & Justin Bieber      2.30   0.506
8
2                     Levitating Dua Lipa feat. DaBaby      3.38   0.825
6
3  Peaches Justin Bieber feat. Daniel Caesar & Gi...      3.30   0.696
0
4           Montero (Call Me By Your Name) Lil Nas X      2.30   0.503
8

   loudness  mode  speechiness  acousticness  instrumentalness
liveness  \
0    -5.044     1       0.1540       0.33500             0.000
0.0849
1   -11.275     1       0.0589       0.37900             0.868
0.1100
2    -3.787     0       0.0601       0.00883             0.000
0.0674
3    -6.181     1       0.1190       0.32100             0.000
0.4200
4    -6.725     0       0.2200       0.29300             0.000
0.4050
```

```
   valence    tempo  danceability
0    0.688  166.928         0.563
1    0.454  170.054         0.564
2    0.915  102.977         0.702
3    0.464   90.030         0.677
4    0.710  178.781         0.593
```

```
# viewing the first rows in our dataset
data.head()
```

```
   duration  energy  key  loudness  mode  speechiness  acousticness  \
0      2.97   0.664    9    -5.044     1       0.1540       0.33500
1      2.30   0.506    8   -11.275     1       0.0589       0.37900
2      3.38   0.825    6    -3.787     0       0.0601       0.00883
3      3.30   0.696    0    -6.181     1       0.1190       0.32100
4      2.30   0.503    8    -6.725     0       0.2200       0.29300

   instrumentalness  liveness  valence    tempo  danceability
0             0.000    0.0849    0.688  166.928         0.563
1             0.868    0.1100    0.454  170.054         0.564
2             0.000    0.0674    0.915  102.977         0.702
3             0.000    0.4200    0.464   90.030         0.677
4             0.000    0.4050    0.710  178.781         0.593
```

```
#plotting distribution plots of our data
data.hist(figsize = (9,9));
```




```
#having a general description of our data
data.describe().T
```

	count	mean	std	min	25%	50% \
duration	100.0	3.404900	0.927022	0.73000	2.910000	3.3700
energy	100.0	0.587650	0.168273	0.15700	0.477750	0.6090
key	100.0	5.050000	3.825420	0.00000	1.000000	5.0000
loudness	100.0	-6.577120	2.447338	-14.06700	-7.676250	

```
-6.2625
mode              100.0    0.560000   0.498888   0.00000   0.000000
1.0000
speechiness       100.0    0.075461   0.068065   0.02530   0.036100
0.0518
acousticness      100.0    0.314539   0.281076   0.00028   0.090750
0.2385
instrumentalness  100.0    0.070682   0.221947   0.00000   0.000000
0.0000
liveness          100.0    0.161737   0.112657   0.03410   0.088375
0.1200
valence           100.0    0.517354   0.237512   0.05920   0.329000
0.5420
tempo             100.0  121.548260  29.148613  71.88400  97.476250
117.0375
danceability      100.0    0.647900   0.126942   0.35200   0.566750
0.6635

                           75%      max
duration              3.630000   10.220
energy                0.707750    0.883
key                   8.000000   11.000
loudness             -4.786750   -2.724
mode                  1.000000    1.000
speechiness           0.080650    0.368
acousticness          0.519750    0.941
instrumentalness      0.000083    0.941
liveness              0.205250    0.509
valence               0.711500    0.967
tempo               141.733750  180.917
danceability          0.734000    0.910
```

By Method (1): Recommendation with Eucledian Distance

```
# feature selection
data_features = data.iloc[:,1:]

#scale the data
data_features_scaled =  StandardScaler().fit_transform(data_features)

def n_nearest_row(dataframe,input_row,n=5):
    print("Input song:\n",pd.DataFrame(data.iloc[input_row,:]).T)

    distances = pdist(dataframe.values, metric='euclidean')
    dist_matrix = squareform(distances)
    distances_from_input_row = pd.DataFrame(dist_matrix)
[input_row].sort_values()

    distances_from_input_row =
```

```
distances_from_input_row[1:n+1].sort_index()
    nearest_rows =
data_main[data.index.isin(distances_from_input_row.index)]

    output_df =
pd.concat((nearest_rows,distances_from_input_row),axis=1)

    columns = list(data_main.columns)
    columns.append("distance")
    output_df.columns=columns

    return output_df



nearest_5_row = n_nearest_row(data_features,96) # we will examine the
first index
print("\n\nNearest songs: ")
nearest_5_row
```

```
Input song:
     duration  energy   key  loudness  mode  speechiness  acousticness
\
96       3.37   0.748  11.0    -5.922   0.0       0.0589         0.305


    instrumentalness  liveness  valence    tempo  danceability
96               0.0    0.0811    0.964  163.984         0.672


Nearest songs:

                                            name  duration  energy
key  \
0                         Good 4 U Olivia Rodrigo      2.97   0.664
9
1              Stay The Kid LAROI & Justin Bieber      2.30   0.506
8
47  Cover Me In Sunshine P!nk & Willow Sage Hart      2.37   0.488
5
58                 Stressed Out twenty one pilots      3.37   0.637
4
80         rockstar (feat. 21 Savage) Post Malone      3.64   0.520
5

    loudness  mode  speechiness  acousticness  instrumentalness
liveness  \
0     -5.044     1       0.1540        0.3350          0.000000
0.0849
1    -11.275     1       0.0589        0.3790          0.868000
```

```
0.1100
47   -11.276     1       0.0568        0.0142          0.900000
0.1560
58    -5.677     0       0.1410        0.0462          0.000023
0.0602
80    -6.136     0       0.0712        0.1240          0.000070
0.1310
```

	valence	tempo	danceability	distance
0	0.688	166.928	0.563	3.813541
1	0.454	170.054	0.564	8.751547
47	0.107	160.013	0.543	9.118828
58	0.648	169.977	0.734	9.228556
80	0.129	159.801	0.585	7.371259

```
data_main.shape
```

```
(100, 13)
```

```
data.shape
```

```
(100, 12)
```

By Method(2): Recommendation with K-Nearest Algorithm

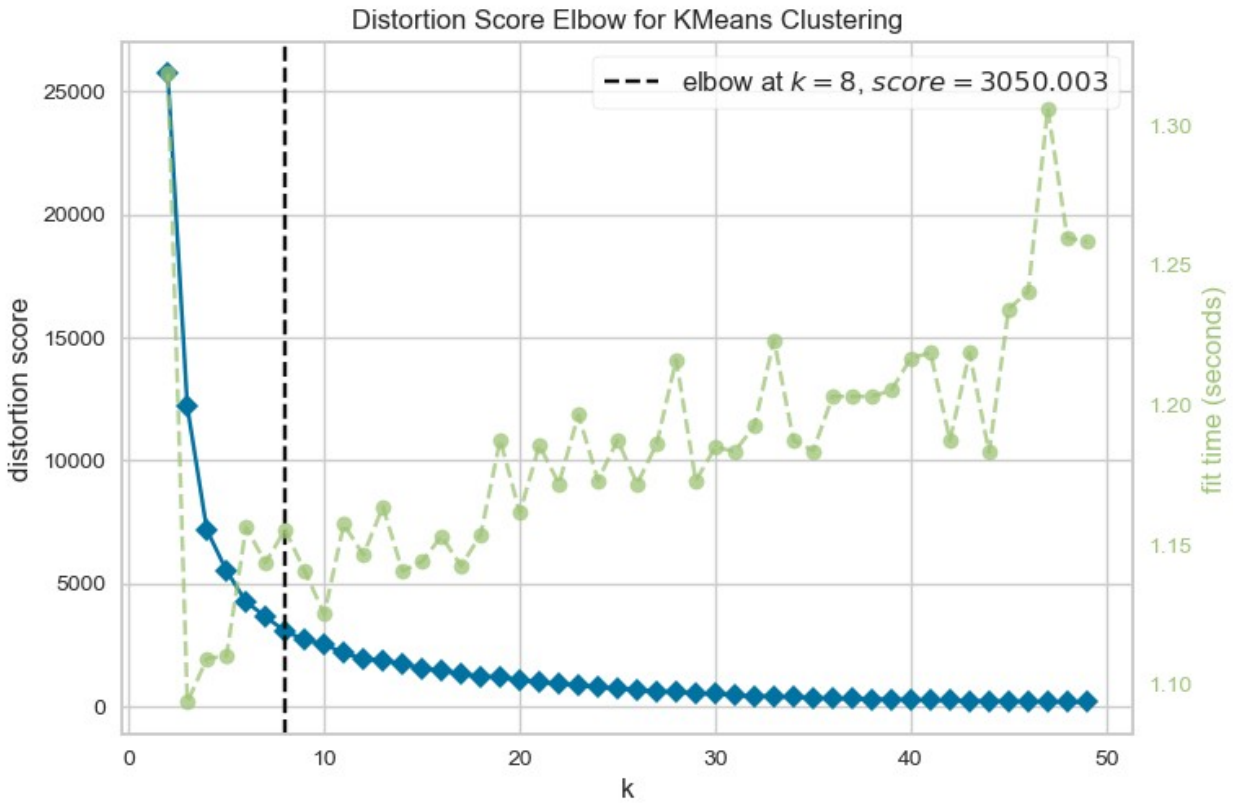
Inorder,to find the optimal values of K of my classification, Iused K-Means from the scikit-learn library and the elbow visualizer for the yellow brick library.

```
! pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in c:\users\lenovo\
anaconda3\lib\site-packages (1.5)
Requirement already satisfied: numpy>=1.16.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from yellowbrick) (1.23.5)
Requirement already satisfied: cycler>=0.10.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\users\
lenovo\anaconda3\lib\site-packages (from yellowbrick) (3.7.0)
Requirement already satisfied: scipy>=1.0.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from yellowbrick) (1.10.0)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from yellowbrick) (1.2.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (1.0.5)
Requirement already satisfied: pillow>=6.2.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (9.4.0)
Requirement already satisfied: packaging>=20.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
```

```
>yellowbrick) (22.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (1.4.4)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (4.25.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\
lenovo\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (2.8.2)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (3.0.9)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\
lenovo\anaconda3\lib\site-packages (from scikit-learn>=1.0.0-
>yellowbrick) (2.2.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick)
(1.2.0)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\anaconda3\
lib\site-packages (from python-dateutil>=2.7->matplotlib!
=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
```

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

kmeans = KMeans()
visualizer = KElbowVisualizer(kmeans, k=(2,50))

visualizer.fit(data_features)
visualizer.poof();
```



From the above plots we got to know determine the optimal value of k to 8.

```
kmeans = KMeans(n_clusters=8)
kmeans.fit(data_features)
data_groups = pd.concat((data_main,
pd.DataFrame(kmeans.labels_,columns=['Group'])), axis=1)
data_groups.head(5)
```

```
                                                 name  duration  energy
key  \
0                            Good 4 U Olivia Rodrigo      2.97   0.664
9
1                  Stay The Kid LAROI & Justin Bieber      2.30   0.506
8
2                    Levitating Dua Lipa feat. DaBaby      3.38   0.825
6
3  Peaches Justin Bieber feat. Daniel Caesar & Gi...      3.30   0.696
0
4           Montero (Call Me By Your Name) Lil Nas X      2.30   0.503
8

   loudness  mode  speechiness  acousticness  instrumentalness
liveness  \
0    -5.044     1       0.1540       0.33500             0.000
0.0849
```

```
1   -11.275     1       0.0589       0.37900              0.868
0.1100
2    -3.787     0       0.0601       0.00883              0.000
0.0674
3    -6.181     1       0.1190       0.32100              0.000
0.4200
4    -6.725     0       0.2200       0.29300              0.000
0.4050

   valence    tempo  danceability  Group
0    0.688  166.928         0.563      6
1    0.454  170.054         0.564      6
2    0.915  102.977         0.702      5
3    0.464   90.030         0.677      1
4    0.710  178.781         0.593      2
```

```
group5 = data_groups[data_groups['Group']==5]
group5
```

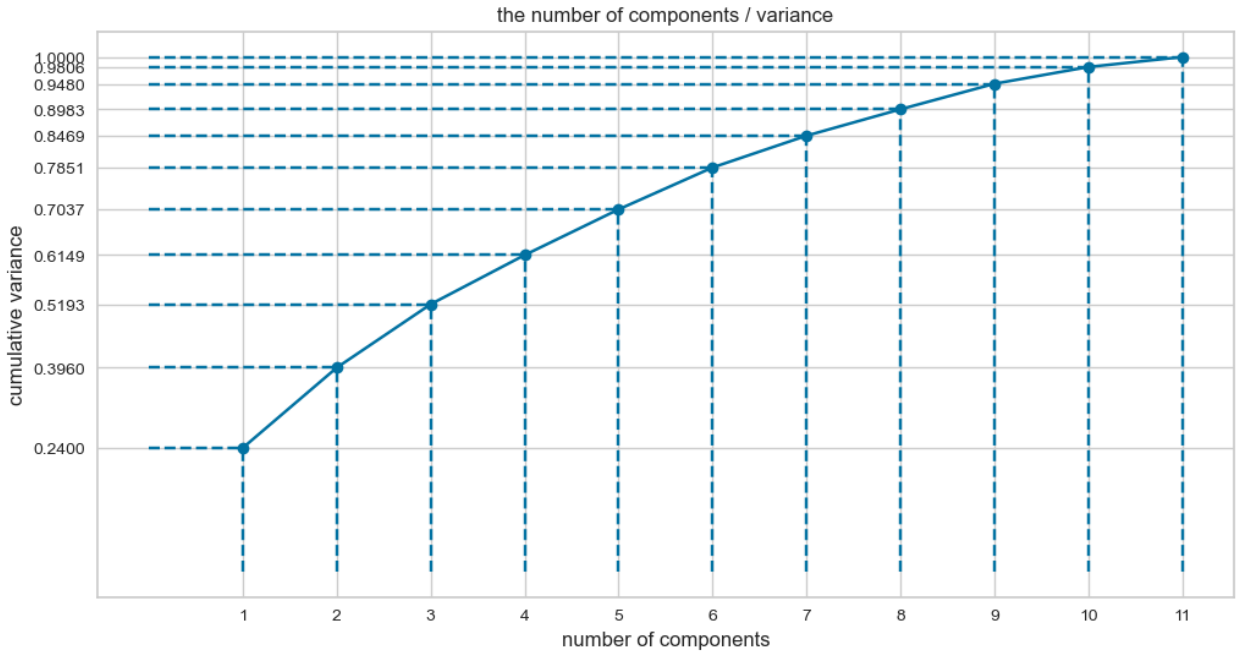
```
                                                  name  duration
energy  key  \
2                    Levitating Dua Lipa feat. DaBaby      3.38
0.825    6
13  Beautiful Mistakes (feat. Megan Thee Stallion)...      3.79
0.676   10
18          Fiel Los Legendarios, Wisin & Jhay Cortez      4.27
0.430    6
22                Levitating (feat. DaBaby) Dua Lipa      3.38
0.825    6
26             Famous Friends Chris Young + Kane Brown      2.74
0.513    6
32                                Levitating Dua Lipa      3.38
0.825    6
36                          Tusa KAROL G & Nicki Minaj      3.37
0.701    2
41                       Therefore I Am Billie Eilish      2.91
0.340   11
49    The Woo (feat. 50 Cent & Roddy Ricch) Pop Smoke      3.36
0.618    1
51                    You Right Doja Cat & The Weeknd      2.95
0.360    6
57                                     Pretend CNCO      3.34
0.883   11
59                       Therefore I Am Billie Eilish      2.91
0.340   11
63                                  Heather Conan Gray      3.30
0.425    5
67                            Dance Monkey Tones And I      3.49
0.588    6
68                  Shallow Lady Gaga & Bradley Cooper      3.60
```

```
0.385    7
69                                Memories Maroon 5      3.16
0.327   11
75            Talking to the Moon Acoustic Bruno Mars      3.63
0.333    1
97                       Therefore I Am Billie Eilish      2.91
0.340   11

    loudness  mode  speechiness  acousticness  instrumentalness
liveness  \
2     -3.787     0       0.0601       0.00883          0.000000
0.0674
13    -5.483     1       0.0270       0.03770          0.000000
0.1540
18    -7.477     0       0.0255       0.73500          0.941000
0.1240
22    -3.787     0       0.0601       0.00883          0.000000
0.0674
26    -8.619     1       0.0311       0.10100          0.000002
0.1310
32    -3.787     0       0.0601       0.00883          0.000000
0.0674
36    -3.275     1       0.2830       0.30800          0.000078
0.0516
41    -7.773     0       0.0697       0.21800          0.130000
0.0550
49    -5.655     1       0.1040       0.02210          0.000004
0.2590
51   -11.212     1       0.0279       0.91800          0.877000
0.1650
57    -3.862     0       0.1030       0.28400          0.000000
0.1400
59    -7.773     0       0.0697       0.21800          0.130000
0.0550
63    -7.301     1       0.0333       0.58400          0.000000
0.3220
67    -6.400     0       0.0924       0.69200          0.000104
0.1490
68    -6.362     1       0.0308       0.37100          0.000000
0.2310
69    -7.241     1       0.0557       0.84100          0.000000
0.0821
75    -6.423     0       0.0310       0.86700          0.000000
0.1070
97    -7.773     0       0.0697       0.21800          0.130000
0.0550

    valence    tempo  danceability  Group
2     0.915  102.977         0.702      5
```

```
13    0.721   99.048          0.713       5
18    0.300   97.976          0.685       5
22    0.915  102.977          0.702       5
26    0.685  102.040          0.717       5
32    0.915  102.977          0.702       5
36    0.619  100.954          0.796       5
41    0.716   94.009          0.889       5
49    0.286   99.700          0.490       5
51    0.756   99.956          0.699       5
57    0.834  101.983          0.680       5
59    0.716   94.009          0.889       5
63    0.270  102.078          0.357       5
67    0.513   98.027          0.824       5
68    0.323   95.799          0.572       5
69    0.595   91.050          0.775       5
75    0.208   99.511          0.352       5
97    0.716   94.009          0.889       5
```

Applying PCA to the dataset

```
plt.rcParams['figure.figsize']=(12,6)
pca = PCA().fit(data_features_scaled)

plt.ylabel('cumulative variance')
plt.title('the number of components / variance')
plt.xlabel('number of components')

x = np.arange(1, 12)
y = np.cumsum(pca.explained_variance_ratio_)

plt.xticks(x)
plt.yticks(y)

plt.plot(x, y, marker='o', linestyle='-', color='b')

plt.hlines(y, 0, x, linestyle='--')
plt.vlines(x, 0, y, linestyle='--')

plt.show()
```


the number of components / variance
cumulative variance
1.0000
0.9806
0.9480
0.8983
0.8469
0.7851
0.7037
0.6149
0.5193
0.3960
0.2400
1
2
3
4
5
6
7
8
9
10
11
number of components