In [950]:
```
# Importing the requires module
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import skew
from scipy.stats.mstats import winsorize
```

In [951]:
```
# Read the dataset and identify the shape and datatypes of the data
song_dat=pd.read_excel('../../Dataset/ML/Assignment/1673873388_rolling_stones_s
print(f'Shape: {song_dat.shape}\n------------------')
print(f'Data Types:\n-------------------\n {song_dat.dtypes}')
song_dat.head()
```

```
Shape: (1610, 17)
------------------
Data Types:
-------------------
 name                          object
album                         object
release_date          datetime64[ns]
track_number                   int64
id                            object
uri                           object
acousticness                 float64
danceability                 float64
energy                       float64
instrumentalness             float64
liveness                     float64
loudness                     float64
speechiness                  float64
tempo                        float64
valence                      float64
popularity                     int64
duration_ms                    int64
dtype: object
```

Out[951]:

	name	album	release_date	track_number	id	
0	Concert Intro Music - Live	Licked Live In NYC	2022-06-10	1	2IEkywLJ4ykbhi1yRQvmsT	spotify:track:2IEkyv
1	Street Fighting Man - Live	Licked Live In NYC	2022-06-10	2	6GVgVJBKkGJoRfarYRvGTU	spotify:track:6GVgVJI
2	Start Me Up - Live	Licked Live In NYC	2022-06-10	3	1Lu761pZ0dBTGpzxaQoZNW	spotify:track:1Lu761p
3	If You Can't Rock Me - Live	Licked Live In NYC	2022-06-10	4	1agTQzOTUnGNggyckEqiDH	spotify:track:1agTQz(
4	Donâ€™t Stop - Live	Licked Live In NYC	2022-06-10	5	7piGJR8YndQBQWVXv6KtQw	spotify:track:7piGJR8Y

Initial data inspection and data cleaning:

Check whether the data has duplicates, missing values, irrelevant (erroneous entries) values, or outliers.

In [952]:
```
def checking_data():
    null_cols=[]
    have_duplicates=False

    #Checking null value
    cols1=[x for x in song_dat if song_dat[x].isna().sum()!=0]

    #Checking duplicate value
    song_df=song_dat[song_dat.duplicated()]
    if(song_df.shape[0]>=1):
        have_duplicates=True

    if(len(cols1)==0):
        print('There was not any missing value')
    else:
        print(f'The columns with having null value are {cols1}')

    if(have_duplicates):
        print('There was a duplicate value')
    else:
        print('There was not any duplicate value ')


#Checking the dataset
checking_data()
```

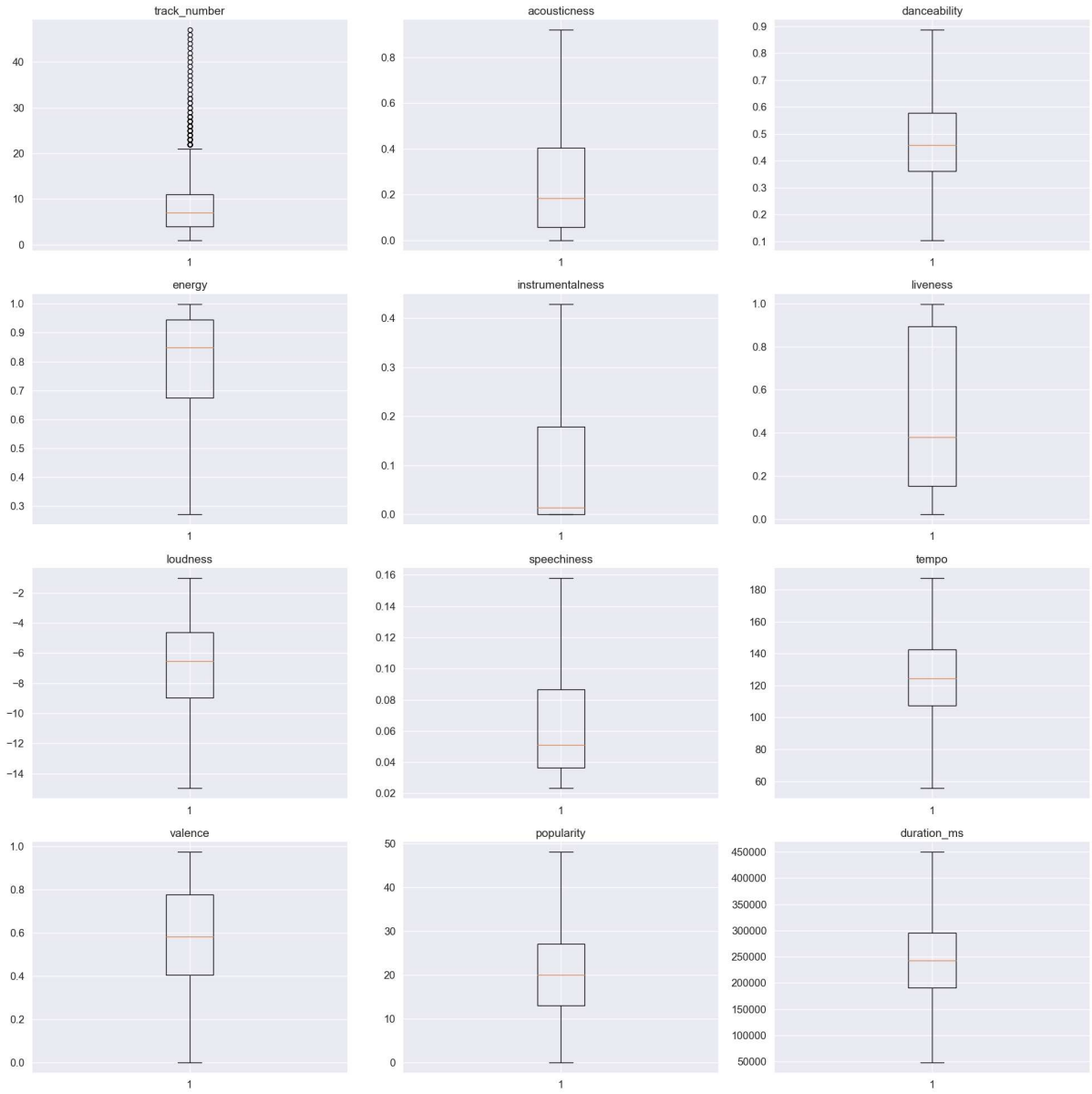
```
There was not any missing value
There was not any duplicate value
```

In [953]:
```
# Converting the date_time coloumn to object
song_dat['release_date']=song_dat['release_date'].astype(str)
# Identifying continuous and descrete data
continuous_data=[x for x in song_dat if song_dat[x].dtypes!='O']
descrete_data=[x for x in song_dat if x not in continuous_data]
print('Continuous data:\n',continuous_data)
print('-----------------------')
print('Descrete data:\n',descrete_data)
```

```
Continuous data:
 ['track_number', 'acousticness', 'danceability', 'energy', 'instrumentalnes
s', 'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'popularity',
'duration_ms']
-----------------------
Descrete data:
 ['name', 'album', 'release_date', 'id', 'uri']
```

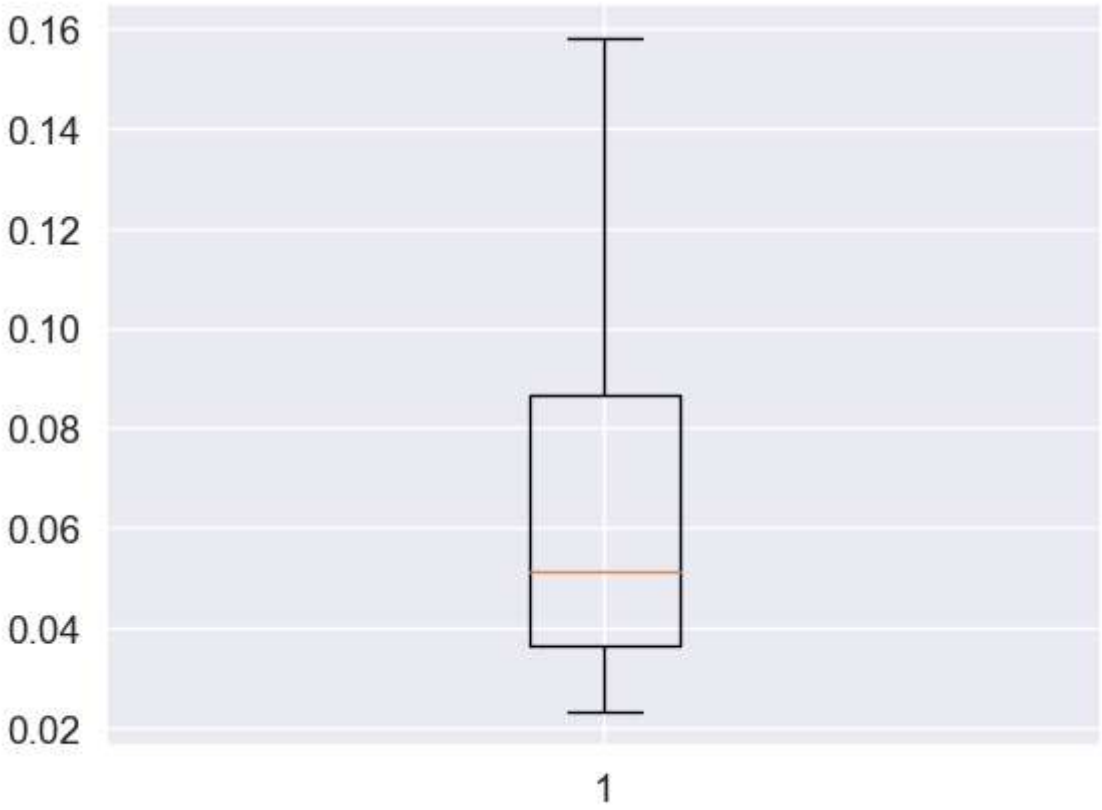
In [972]:
```
# Checking Outliers
def outlier_checker(coutinuous_df):
    print(f'Snewness Scores:\n {continuous_df.skew(axis=0)}')
    plt.figure(figsize=(20,20))
    for indx,col in enumerate(continuous_data):
        plt.subplot(4,3,indx+1)
        plt.title(col)
        plt.boxplot(continuous_df[col])
        plt.tight_layout()

continuous_df=song_dat.drop(descrete_data,axis=1)
outlier_checker(continuous_df)
```

```
Snewness Scores:
 track_number        1.777241
acousticness        0.860006
danceability        0.162052
energy             -0.916305
instrumentalness    1.187124
liveness            0.220033
loudness           -0.526771
speechiness         1.127801
tempo               0.214175
valence            -0.195988
popularity          0.378290
duration_ms         0.520714
dtype: float64
```

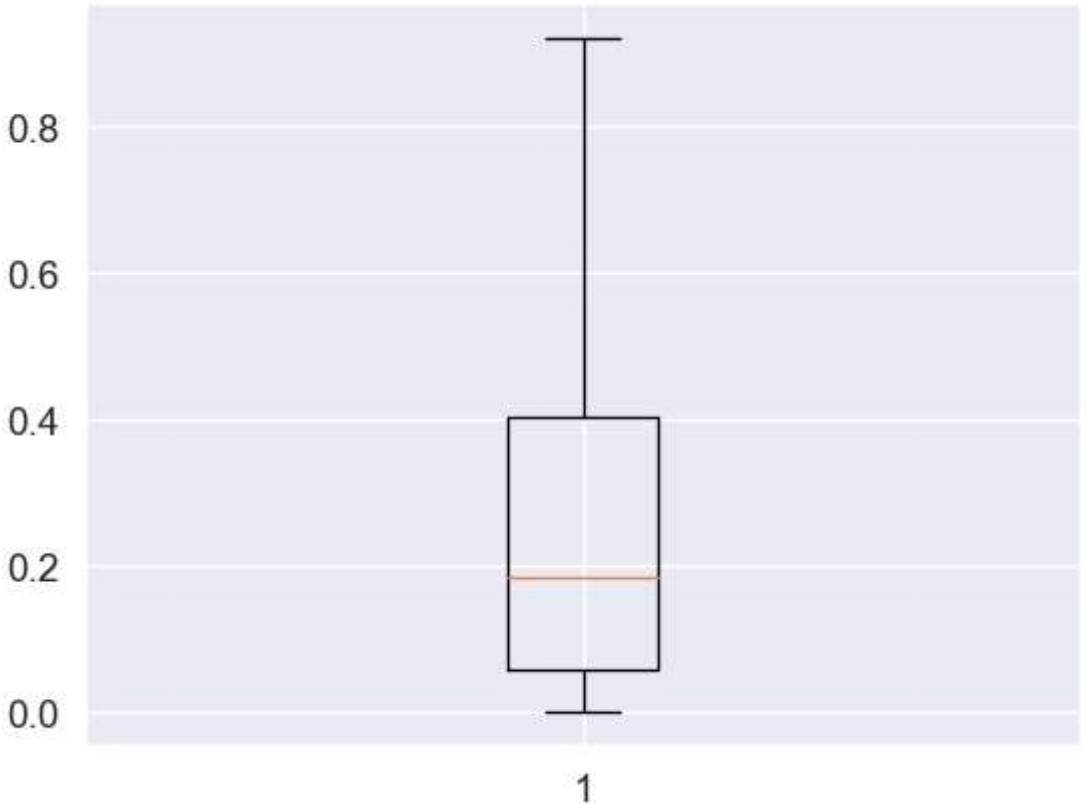


Depending on your findings, clean the data for further processing.

In [955]:
```
song_dat['speechiness']=winsorize(song_dat['speechiness'],limits=[0,.059])
plt.boxplot(song_dat['speechiness'])
plt.show()
```

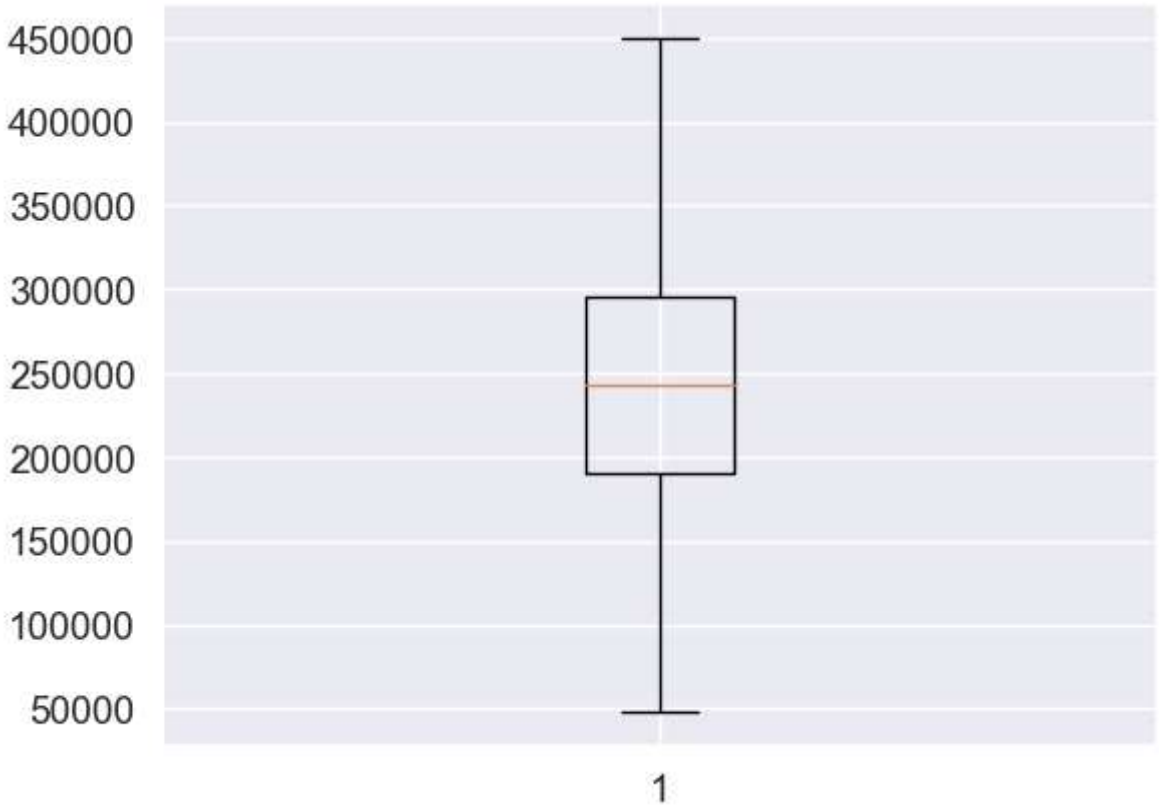


In [956]:
```
song_dat['acousticness']=winsorize(song_dat['acousticness'],limits=[0,0.003])
plt.boxplot(song_dat['acousticness'])
plt.show()
```

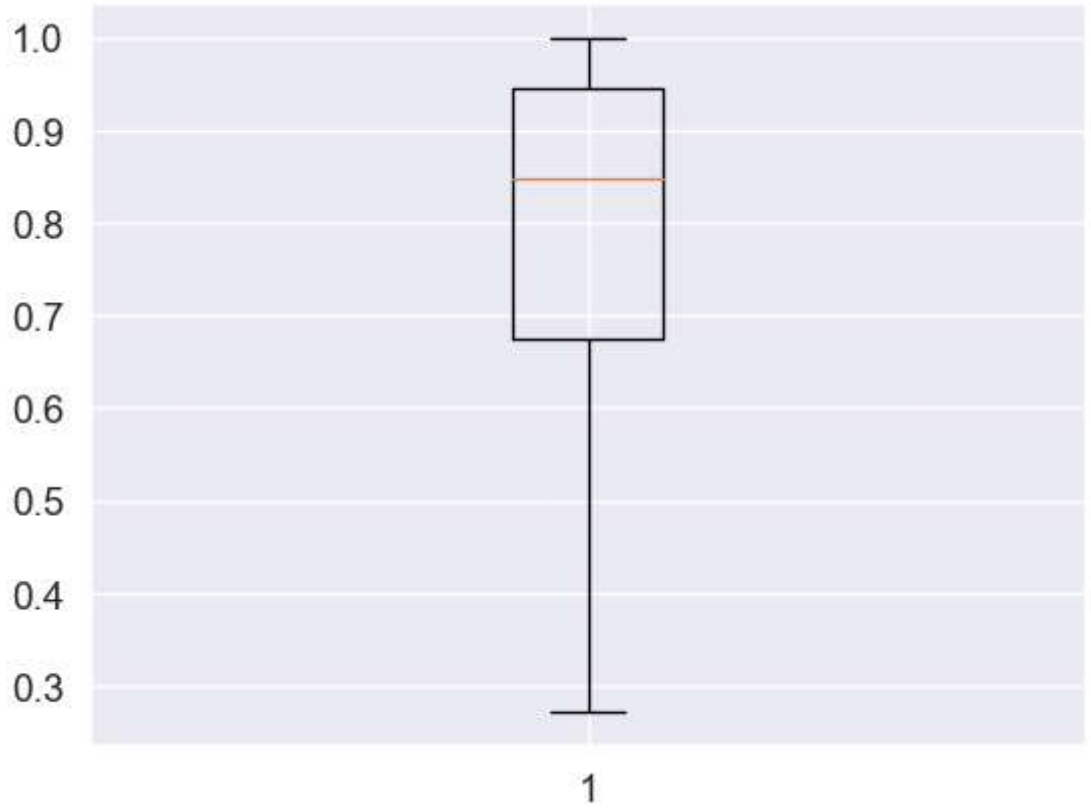


In [ ]:
```

```

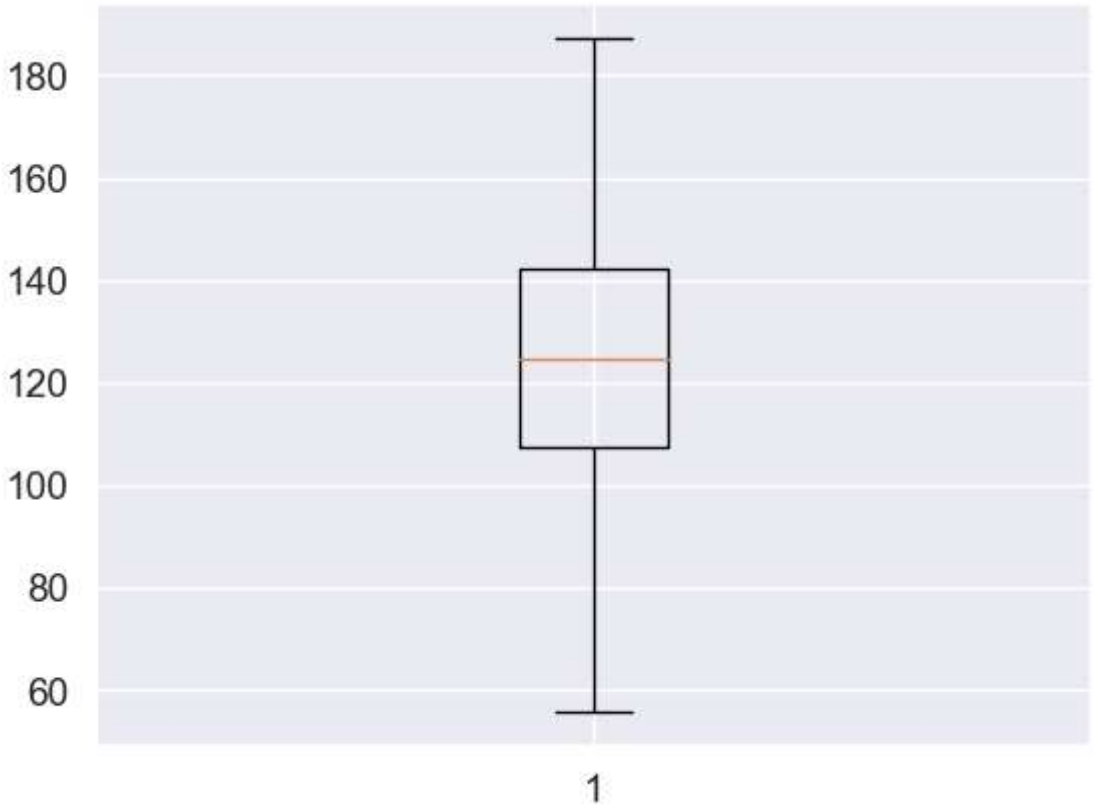
In [957]:
```
song_dat['duration_ms']=winsorize(song_dat['duration_ms'],limits=[0.005,0.045])
plt.boxplot(song_dat['duration_ms'])
plt.show()
```



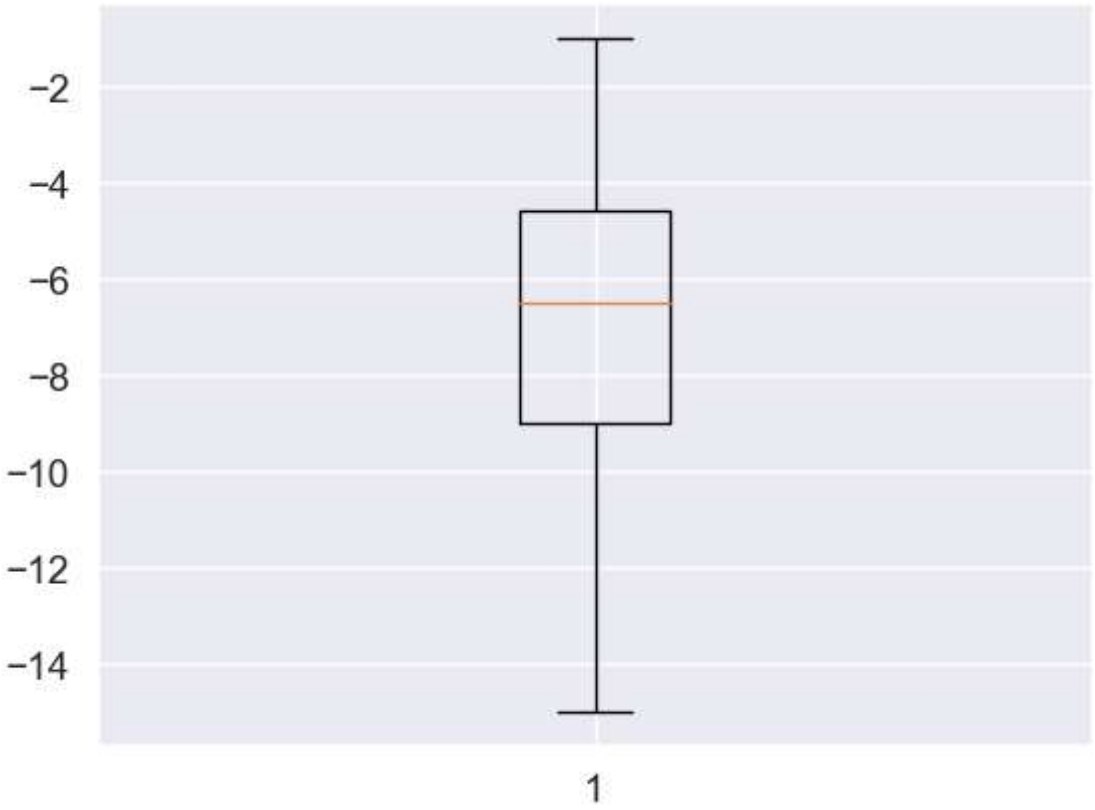
In [958]:
```
song_dat['energy']=winsorize(song_dat['energy'],limits=[0.005,0])
plt.boxplot(song_dat['energy'])
plt.show()
```



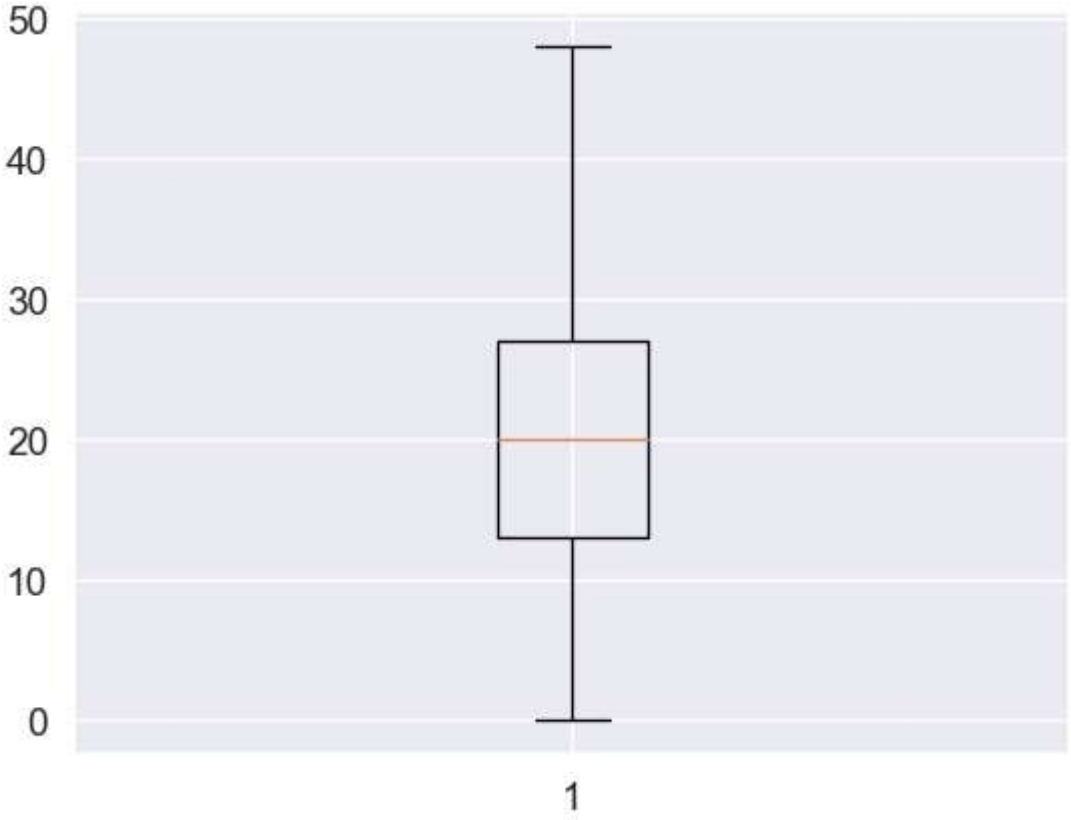
In [959]:
```
song_dat['tempo']=winsorize(song_dat['tempo'],limits=[0.001,0.03])
plt.boxplot(song_dat['tempo'])
plt.show()
```



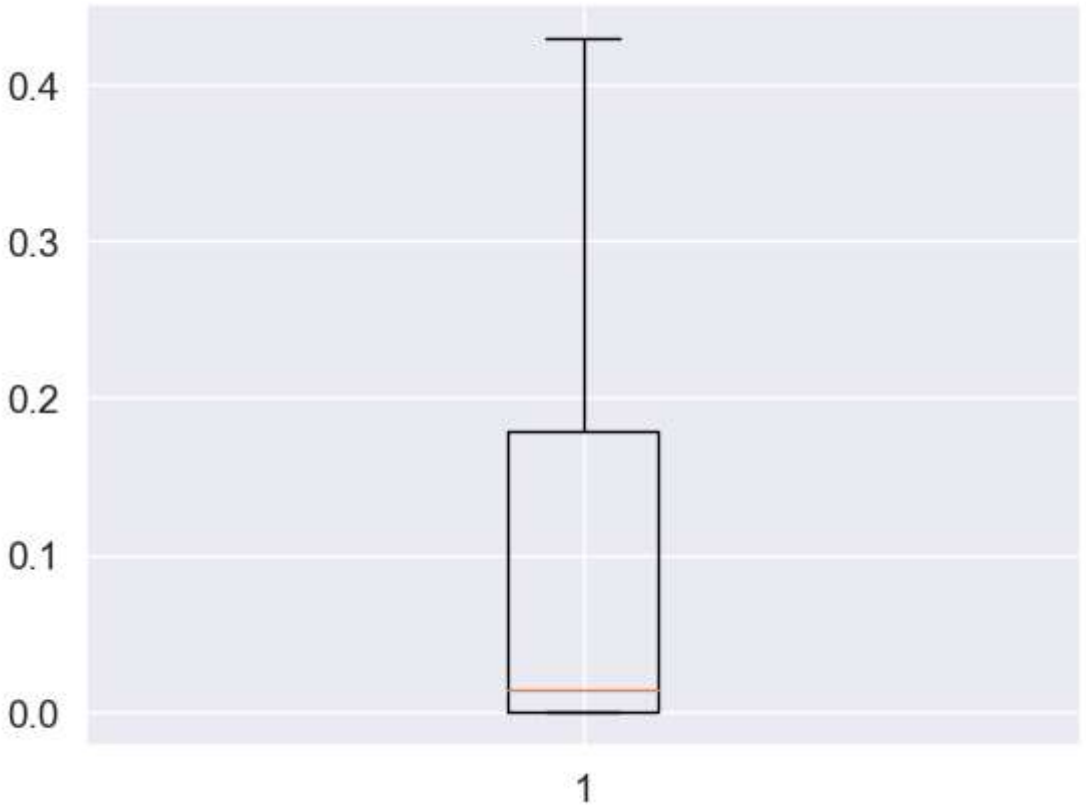
In [960]:
```
song_dat['loudness']=winsorize(song_dat['loudness'],limits=[0.009,0])
plt.boxplot(song_dat['loudness'])
plt.show()
```



In [961]:
```
song_dat['popularity']=winsorize(song_dat['popularity'],limits=[0,0.03])
plt.boxplot(song_dat['popularity'])
plt.show()
```

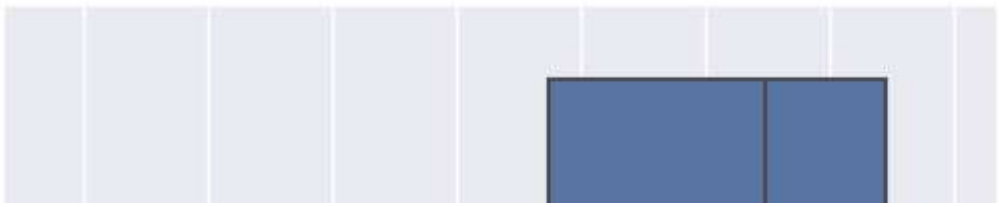


In [962]:
```
song_dat['instrumentalness']=winsorize(song_dat['instrumentalness'],limits=[0,0
plt.boxplot(song_dat['instrumentalness'])
plt.show()
```



In [963]:
```
for data in continuous_data:
    sns.boxplot(song_dat[data])
    plt.show()
```

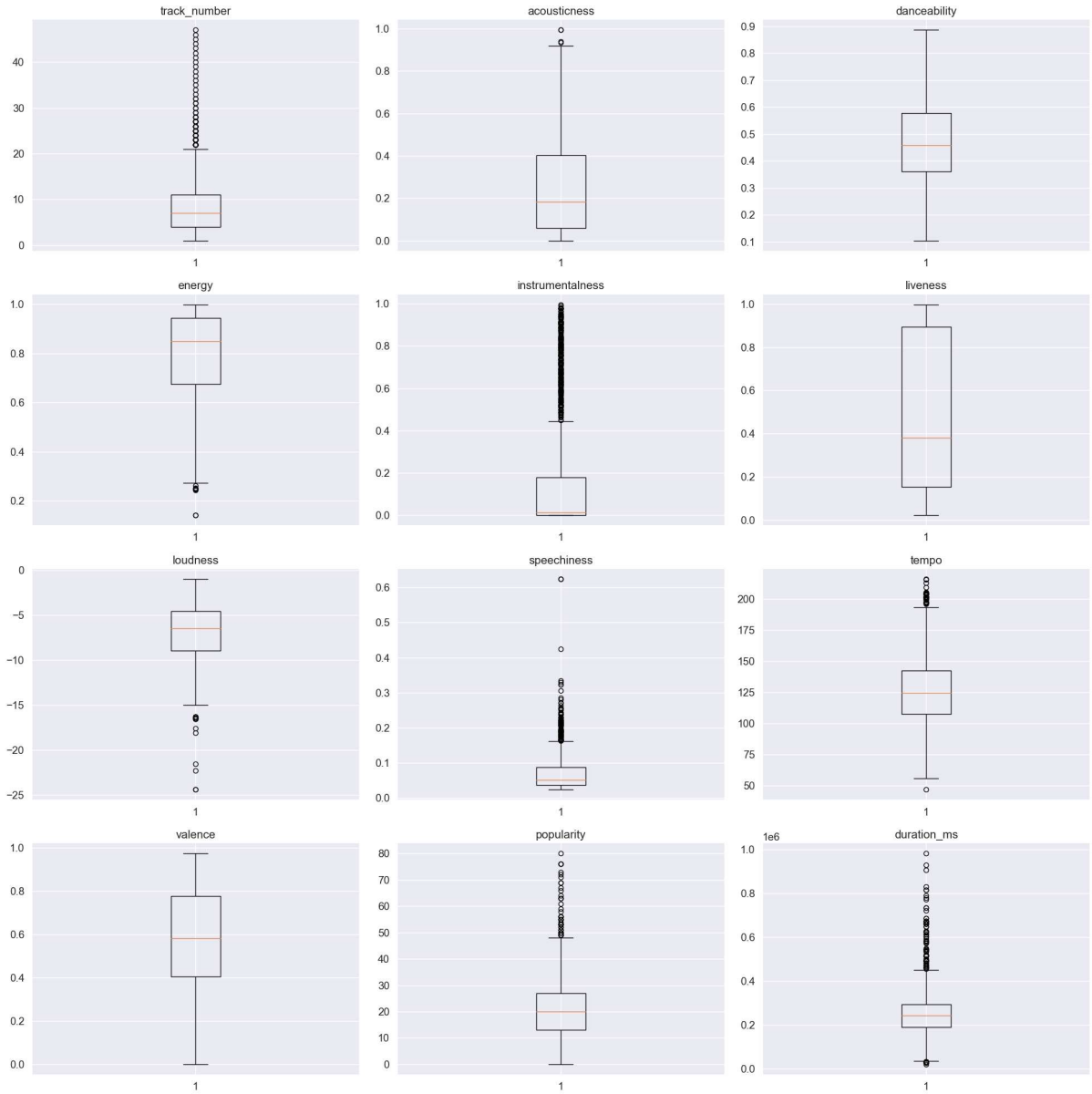

```
C:\Users\rajal\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futur
eWarning: Pass the following variable as a keyword arg: x. From version 0.1
2, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpreta
tion.
  warnings.warn(
```



In [964]:
```
df=song_dat.drop(descrete_data,axis=1)
```

In [965]:
```
outlier_checker(df)
```

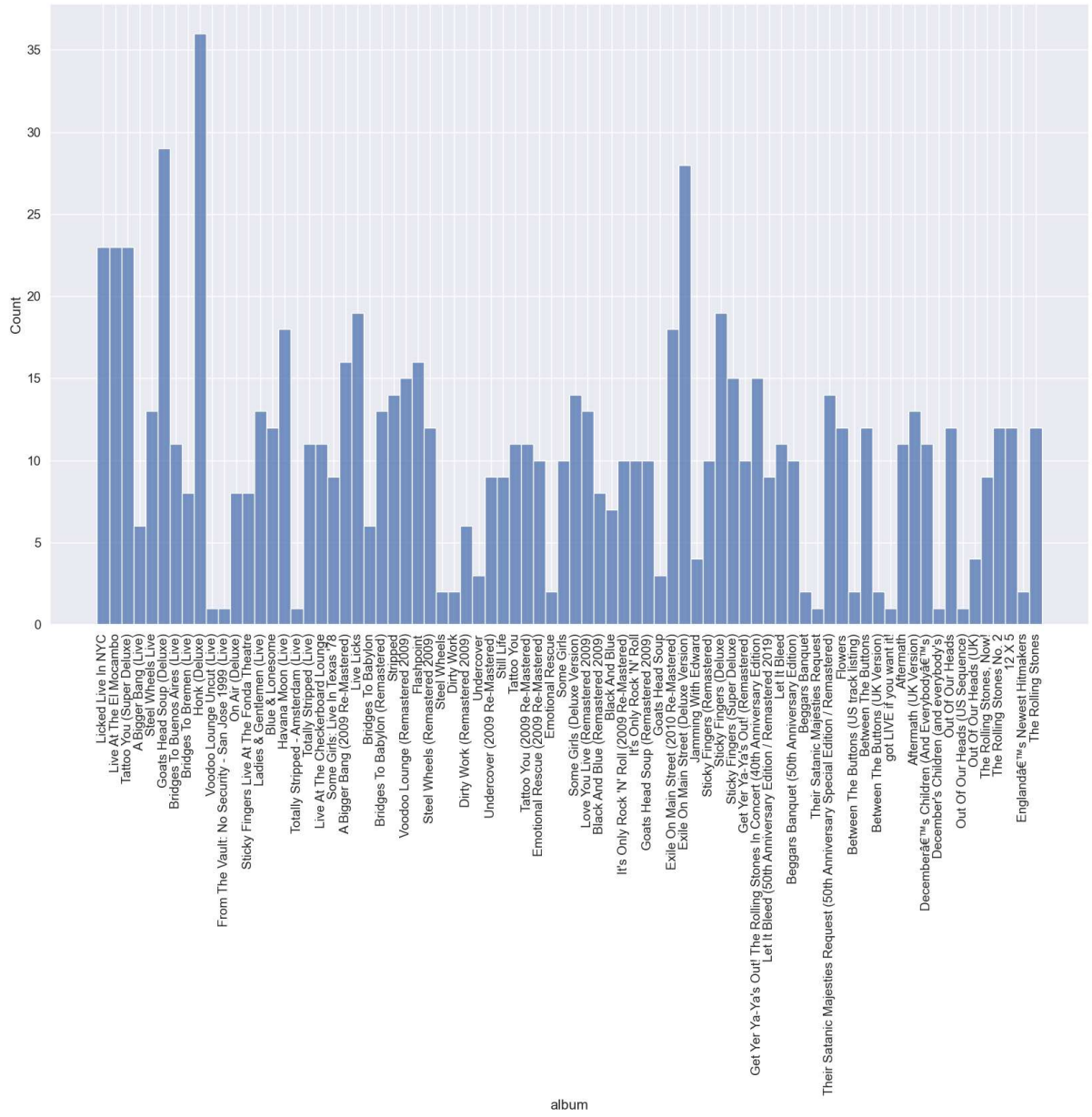
```
Snewness Scores:
 track_number        1.777241
acousticness         0.868785
danceability         0.162052
energy              -0.941379
instrumentalness     1.667856
liveness             0.220033
loudness            -0.888034
speechiness          3.230335
tempo                0.361867
valence             -0.195988
popularity           0.883930
duration_ms          1.924778
dtype: float64
```



Perform Exploratory Data Analysis and Feature Engineering

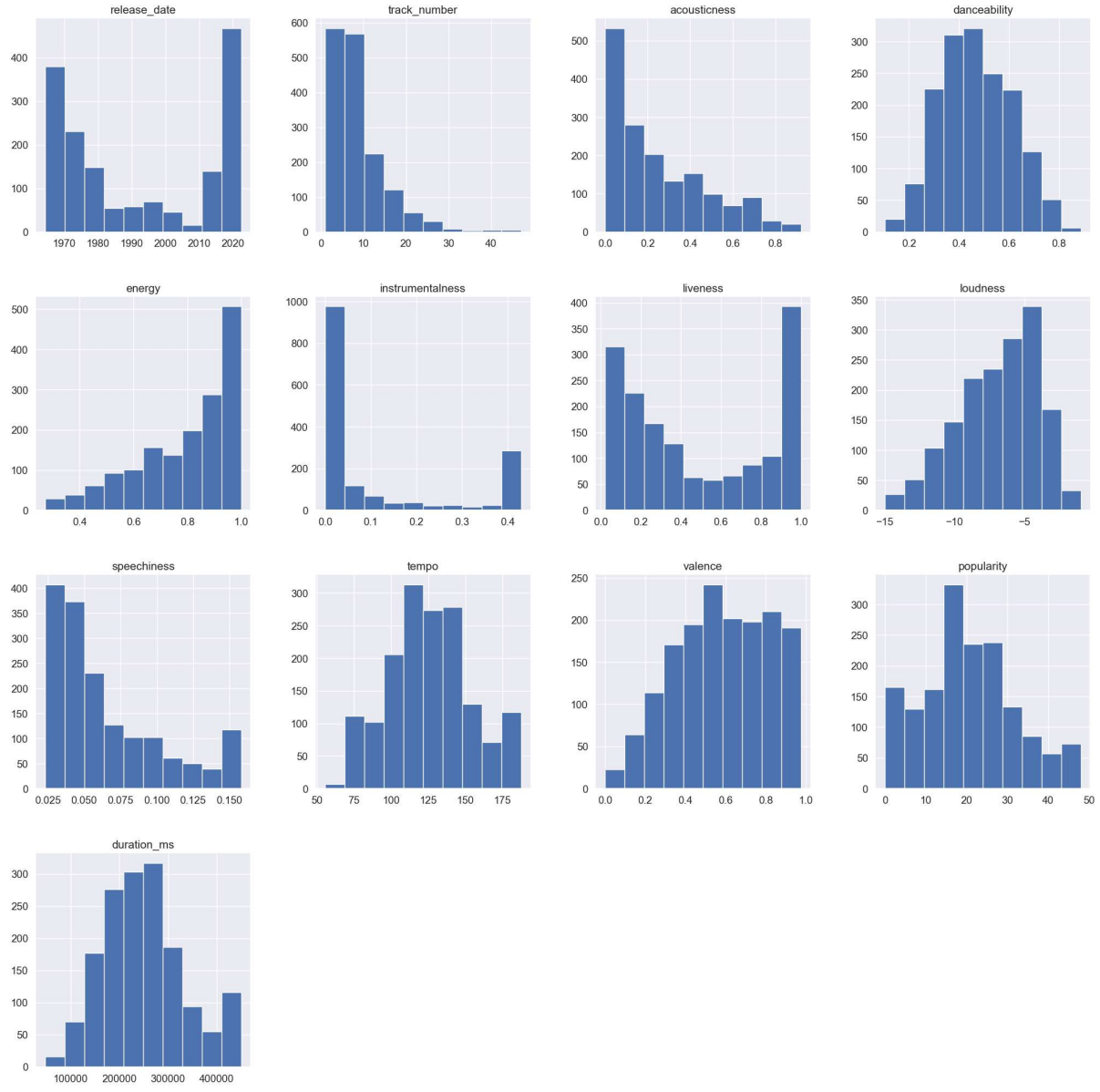
Use appropriate visualizations to find out which two albums should be recommended to anyone based on the number of popular songs in an album.

In [983]:
```
# Removing the features that are not much affect the popularity of the song
song_new=song_dat.copy()
song_new=song_new[['name','album','popularity']]
# Since the popularity rating maximum was 80 So lets take all the song that's p
song_new=song_new.loc[song_new['popularity']>=20]
plt.figure(figsize=(20,12))
plt.xticks(rotation='vertical')
sns.histplot(song_new['album'])
plt.show()
```



Perform exploratory data analysis to dive deeper into different features of songs and identify the pattern.

In [1041]:
```
df=song_dat.drop(['id','uri'],axis=1)
df['release_date']=pd.to_datetime(df['release_date'])

# Check the data distribution for each numerical columns
df.hist(figsize=(25,25))
plt.show()
```



In [1042]:
```
# Checking the numbers of unique value for the columns
print(f'Unique values for numerical columns: \n{df.nunique().sort_values()}')
print(f'\n Unnique values for descrete columns:\n {df.describe(include="O")}')
```

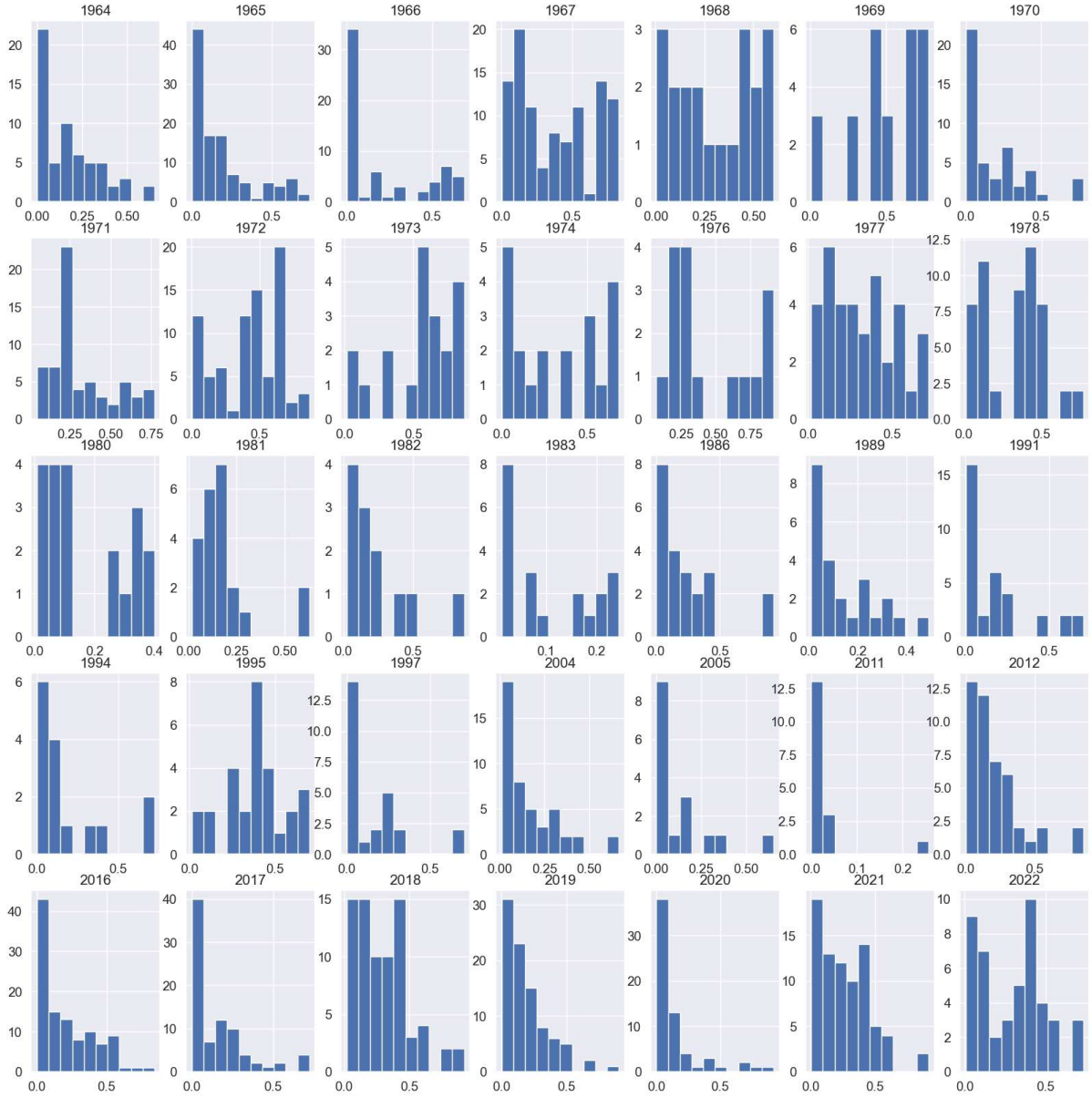
```
Unique values for numerical columns:
track_number          47
popularity            48
release_date          57
album                 90
energy               503
danceability         518
speechiness          588
valence              701
liveness             757
acousticness         930
instrumentalness     944
name                 954
duration_ms         1246
loudness            1316
tempo               1378
dtype: int64

 Unnique values for descrete columns:
                       name                        album
count                  1610                         1610
unique                  954                           90
top     Brown Sugar - Live  Voodoo Lounge Uncut (Live)
freq                     16                           56
```

In [1043]:
```
# Find the years
years=df['release_date'].dt.year.unique()
# Flip to arrange in increasing order
years=np.flip(years)

# Creating an array of dataframe based on the year as all the value that have t
years_df=[]
for year in years:
    years_df.append(df[df['release_date'].dt.year==year])
```

In [1044]:
```
# Show the data distribution for all the columns for the each years
# for indx,df in enumerate(years):
#     print(years[indx],"Data")
#     years_df[indx].hist(figsize=(25,25))
#     plt.savefig(str(years[indx]))
#     plt.show()
```

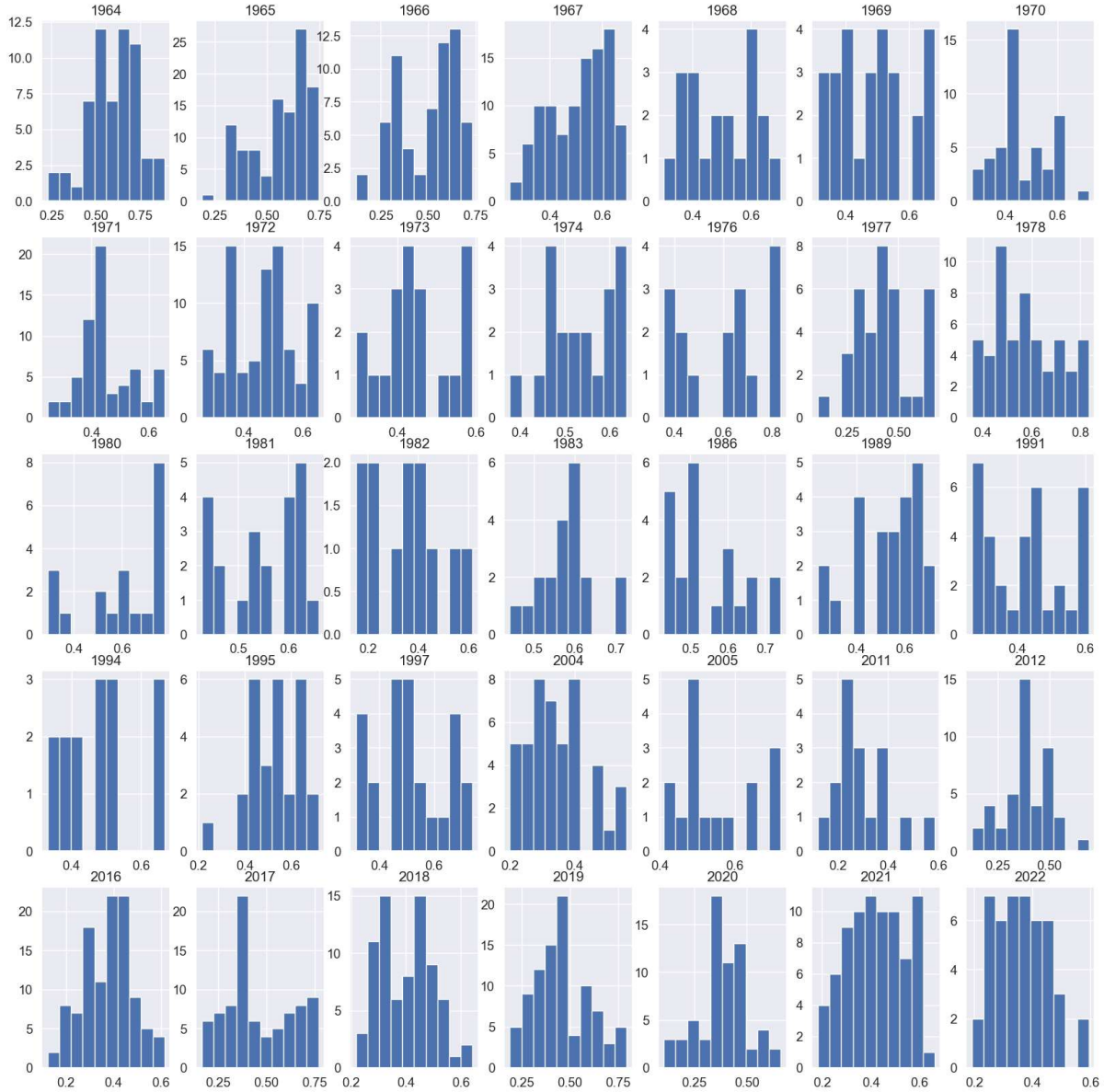
Acousticness Data for all the year

In [1045]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df_=song_dat.loc[song_dat['release_date'].dt.year==i[1]]['acousticness']
    df_.hist()
```



Danceability Data for all the year

In [1047]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df_=df.loc[df['release_date'].dt.year==i[1]]['danceability']
    df_.hist()
```



Energy Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df_=df.loc[new_song_dat['release_date'].dt.year==i[1]]['energy']
    df_.hist()
```

Instrumentalness Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df=new_song_dat.loc[new_song_dat['release_date'].dt.year==i[1]]['instrument
    df.hist()
```

Liveness Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df=new_song_dat.loc[new_song_dat['release_date'].dt.year==i[1]]['liveness']
    df.hist()
```

Speechiness Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df=new_song_dat.loc[new_song_dat['release_date'].dt.year==i[1]]['speechines
    df.hist()
```

Tempo Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df=new_song_dat.loc[new_song_dat['release_date'].dt.year==i[1]]['tempo']
    df.hist()
```

Valence Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df=new_song_dat.loc[new_song_dat['release_date'].dt.year==i[1]]['valence']
    df.hist()
```

Popularity Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df=new_song_dat.loc[new_song_dat['release_date'].dt.year==i[1]]['popularity
    df.hist()
```

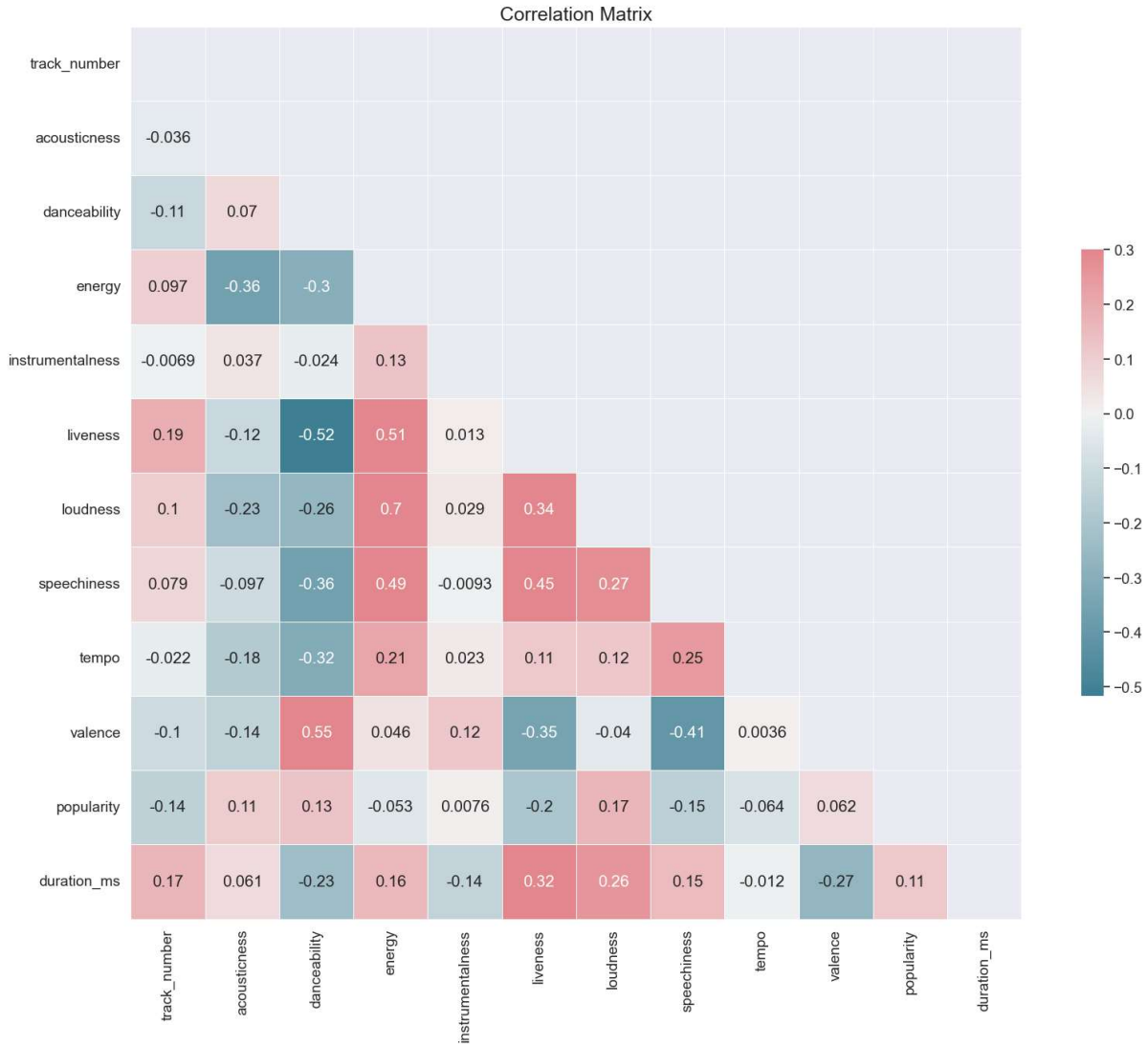
Duration Data for all the year

In [ ]:
```
plt.figure(figsize=(20,20))
for i in enumerate(years):
    plt.subplot(5,7,i[0]+1)
    plt.title(i[1])
    df=new_song_dat.loc[new_song_dat['release_date'].dt.year==i[1]]['duration_m
    df.hist()
```

Discover how a song's popularity relates to various factors and how this has changed over time.

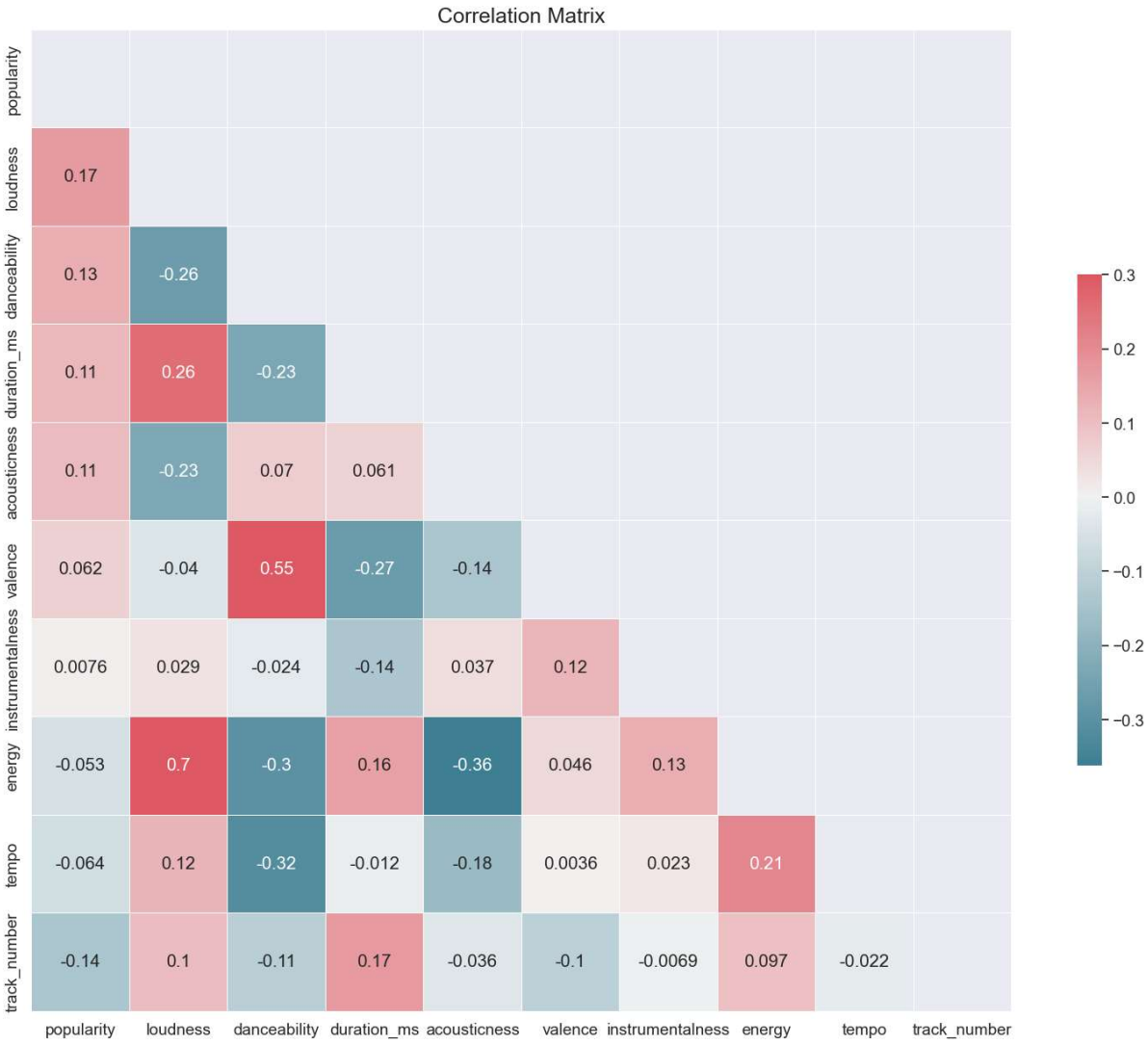
In [1048]:
```
# Ploting histogram to check how the other features are correlated with the pop
corr = song_dat.corr()

mask = np.triu(np.ones_like(corr, dtype=np.bool))

f, ax = plt.subplots(figsize=(20, 15))

cmap = sns.diverging_palette(220, 10, as_cmap=True)

plt.title('Correlation Matrix', fontsize=18)

sns.heatmap(corr,
            mask=mask,
            cmap=cmap,
            vmax=.3,
            center=0,
            square=True,
            linewidths=.5,
            cbar_kws={"shrink": .5},
            annot=True)

plt.show()
```

```
C:\Users\rajal\AppData\Local\Temp\ipykernel_12260\3676987240.py:4: Deprecatio
nWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence
this warning, use `bool` by itself. Doing this will not modify any behavior a
nd is safe. If you specifically wanted the numpy scalar type, use `np.bool_`
here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations)
  mask = np.triu(np.ones_like(corr, dtype=np.bool))
```


Correlation Matrix
track_number
acousticness
danceability
energy
instrumentalness
liveness
loudness
speechiness
tempo
valence
popularity
duration_ms
-0.036
-0.11 0.07
0.097 -0.36 -0.3
-0.0069 0.037 -0.024 0.13
0.19 -0.12 -0.52 0.51 0.013
0.1 -0.23 -0.26 0.7 0.029 0.34
0.079 -0.097 -0.36 0.49 -0.0093 0.45 0.27
-0.022 -0.18 -0.32 0.21 0.023 0.11 0.12 0.25
-0.1 -0.14 0.55 0.046 0.12 -0.35 -0.04 -0.41 0.0036
-0.14 0.11 0.13 -0.053 0.0076 -0.2 0.17 -0.15 -0.064 0.062
0.17 0.061 -0.23 0.16 -0.14 0.32 0.26 0.15 -0.012 -0.27 0.11
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5

In [1049]:
```
# Reduced the features to select the most 10 coorelated features with the popul
# number of variables to be selected
k = 10

# finding the most correlated variables
cols = song_dat.corr().nlargest(k, 'popularity')['popularity'].index
corr=song_dat[cols].corr()
mask = np.triu(np.ones_like(corr, dtype=np.bool))

f, ax = plt.subplots(figsize=(20, 15))

cmap = sns.diverging_palette(220, 10, as_cmap=True)

plt.title('Correlation Matrix', fontsize=18)

sns.heatmap(corr,
            mask=mask,
            cmap=cmap,
            vmax=.3,
            center=0,
            square=True,
            linewidths=.5,
            cbar_kws={"shrink": .5},
            annot=True)

plt.show()
```

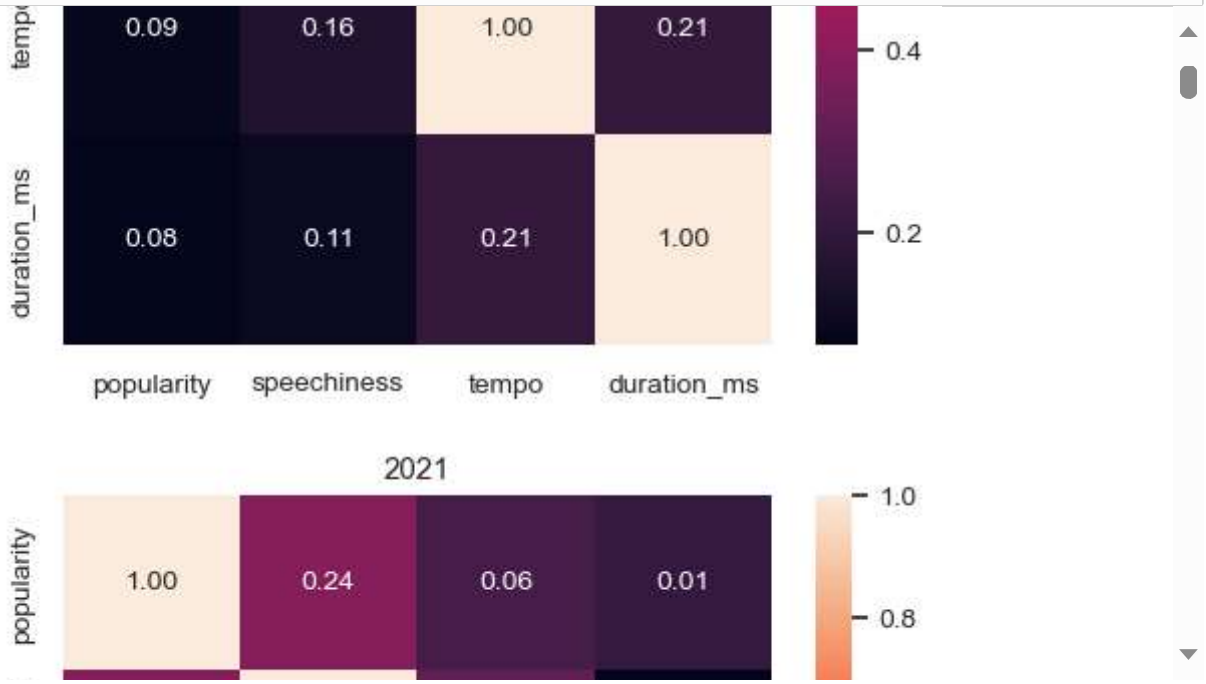
```
C:\Users\rajal\AppData\Local\Temp\ipykernel_12260\922673208.py:8: Deprecation
Warning: `np.bool` is a deprecated alias for the builtin `bool`. To silence t
his warning, use `bool` by itself. Doing this will not modify any behavior an
d is safe. If you specifically wanted the numpy scalar type, use `np.bool_` h
ere.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations)
  mask = np.triu(np.ones_like(corr, dtype=np.bool))
```



- Negative coorelation between popularity and speechiness
- Negative coorelation between tempo and popularity
- Negative coorelation between energy and popularity
- Positive coorelatoin between danceability and popularity
- Positive coorelaton between loudness and popularity
- Positive coorelaton between acousticness and popularity

Yearly data to check what features in which are more coolreated to the popularity

In [1050]:
```
sns.set(font_scale=0.9)
for indx,year in enumerate(np.flip(years)):
    cols = years_df[indx].corr().nlargest(4, 'popularity')['popularity'].index
    cm = np.corrcoef(years_df[indx][cols].values.T)

    #plotting the heatmap
    f, ax = plt.subplots(figsize=(5,5))
    plt.title(year)
    hm = sns.heatmap(cm, cbar=True, annot=True, fmt='.2f', annot_kws={'size': 1
    plt.tight_layout()
```



In [1051]:
```
print("Features That are coorelated with the popularity for different years")
print("-----------------------------------------------------------------------")
corelation_arr=[]
for indx,year in enumerate(np.flip(years)):
    cols = years_df[indx].corr().nlargest(4, 'popularity')['popularity'].index
    cm = np.corrcoef(years_df[indx][cols].values.T)
    for col in cols[1:]:
        corelation_arr.append(col)
    print(year,cols.values[1:])
```

```
Features That are coorelated with the popularity for different years
-----------------------------------------------------------------------
2022 ['speechiness' 'tempo' 'duration_ms']
2021 ['duration_ms' 'acousticness' 'track_number']
2020 ['danceability' 'valence' 'acousticness']
2019 ['loudness' 'liveness' 'acousticness']
2018 ['speechiness' 'loudness' 'tempo']
2017 ['loudness' 'duration_ms' 'acousticness']
2016 ['loudness' 'instrumentalness' 'danceability']
2012 ['danceability' 'acousticness' 'valence']
2011 ['danceability' 'acousticness' 'liveness']
2005 ['loudness' 'tempo' 'speechiness']
2004 ['loudness' 'energy' 'danceability']
1997 ['loudness' 'acousticness' 'duration_ms']
1995 ['loudness' 'energy' 'duration_ms']
1994 ['duration_ms' 'danceability' 'valence']
1991 ['loudness' 'duration_ms' 'danceability']
1989 ['loudness' 'danceability' 'energy']
1986 ['loudness' 'energy' 'duration_ms']
1983 ['speechiness' 'energy' 'loudness']
1982 ['danceability' 'valence' 'energy']
1981 ['loudness' 'energy' 'duration_ms']
1980 ['loudness' 'speechiness' 'energy']
1978 ['acousticness' 'duration_ms' 'loudness']
1977 ['speechiness' 'loudness' 'duration_ms']
1976 ['loudness' 'danceability' 'liveness']
1974 ['loudness' 'energy' 'tempo']
1973 ['duration_ms' 'instrumentalness' 'liveness']
1972 ['danceability' 'valence' 'duration_ms']
1971 ['track_number' 'liveness' 'loudness']
1970 ['instrumentalness' 'loudness' 'acousticness']
1969 ['duration_ms' 'loudness' 'liveness']
1968 ['loudness' 'liveness' 'energy']
1967 ['danceability' 'valence' 'instrumentalness']
1966 ['danceability' 'instrumentalness' 'valence']
1965 ['danceability' 'loudness' 'valence']
1964 ['energy' 'speechiness' 'acousticness']
```

In [1052]:
```
df=pd.DataFrame(corelation_arr)
```

In [1053]:
```
df.value_counts()
```

Out[1053]:
```
loudness            23
danceability        14
duration_ms         14
energy              11
acousticness        10
valence              8
liveness             7
speechiness          7
instrumentalness     5
tempo                4
track_number         2
dtype: int64
```

In [1054]:
```
years_df[0]
```

Out[1054]:

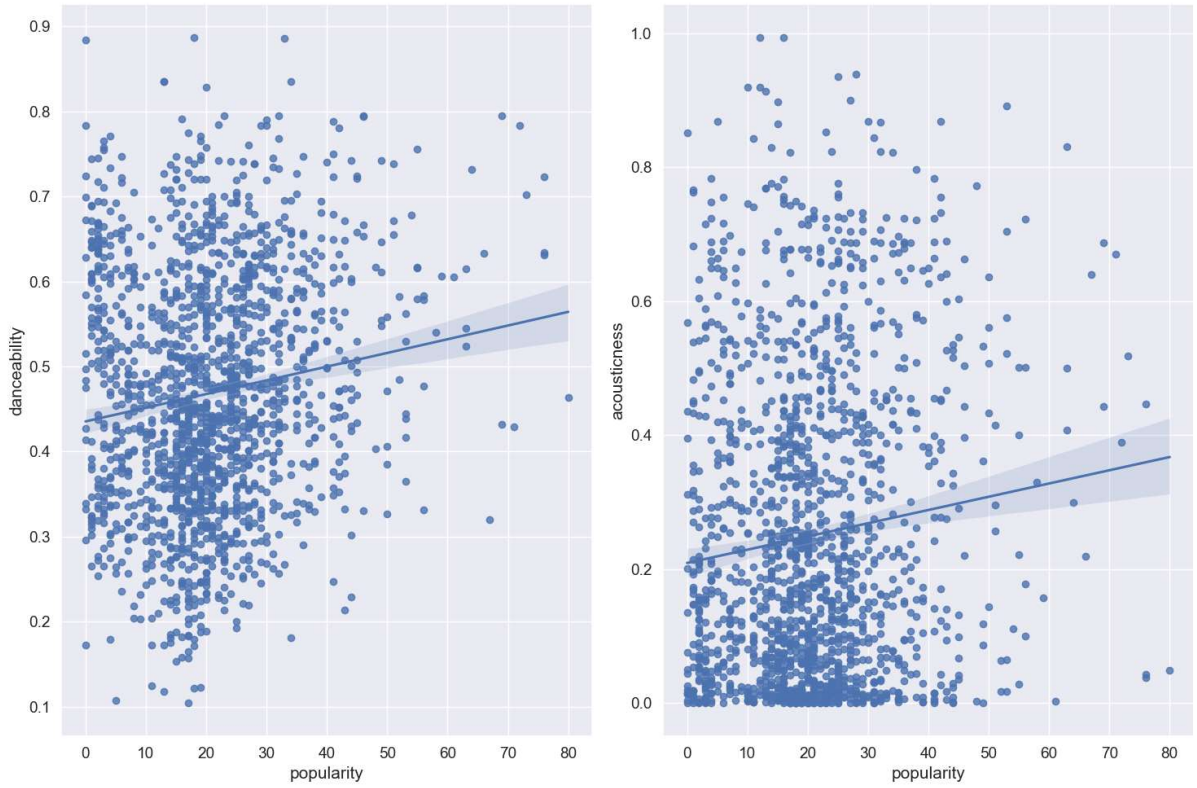
	name	album	release_date	track_number	acousticness	danceability	ene
1550	Around And Around	12 x 5	1964-10-17	1	0.47700	0.317	0
1551	Confessin' The Blues	12 x 5	1964-10-17	2	0.16700	0.479	0
1552	Empty Heart	12 x 5	1964-10-17	3	0.00690	0.498	0
1553	Time Is On My Side	12 x 5	1964-10-17	4	0.28800	0.232	0
1554	Good Times, Bad Times - Mono Version	12 x 5	1964-10-17	5	0.05460	0.438	0
1555	It's All Over Now	12 x 5	1964-10-17	6	0.02320	0.606	0
1556	2120 South Michigan Avenue - Mono	12 x 5	1964-10-17	7	0.00135	0.612	0

Some of the most important features to make the song popular

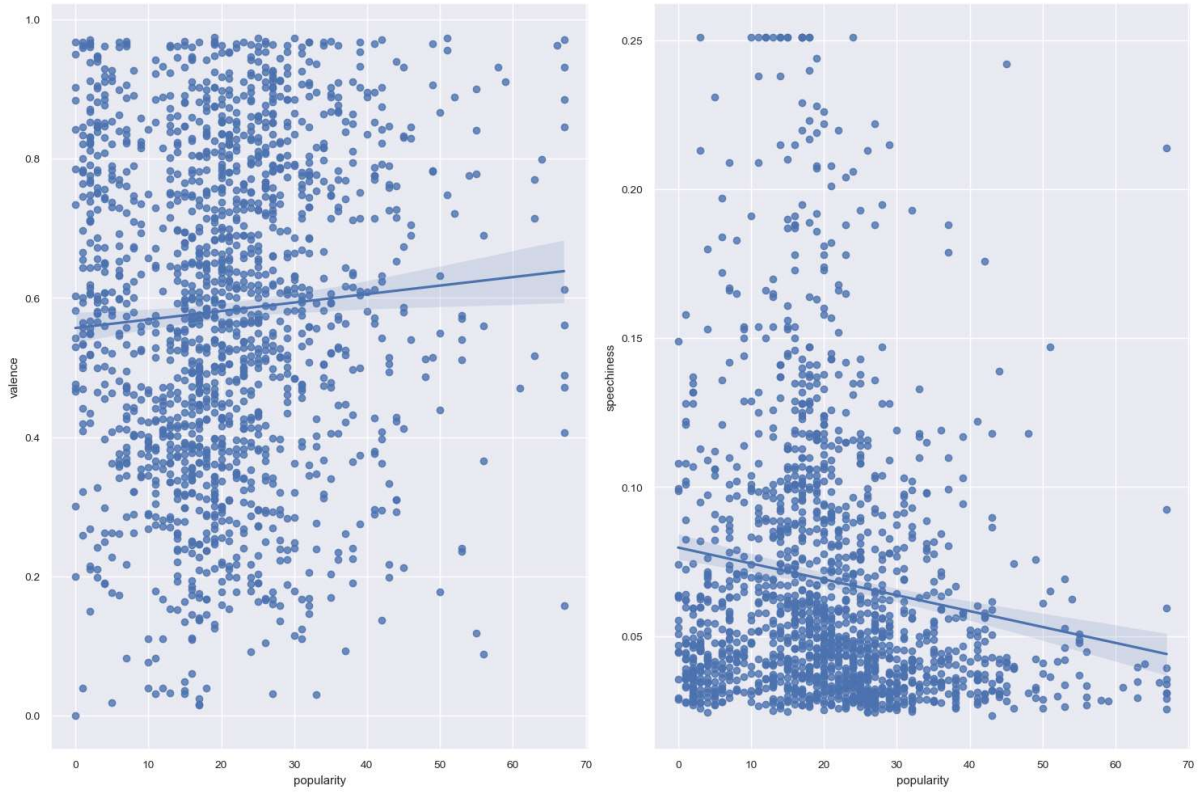
- Duration of the song is the most important feature to become popular as if as song was only 10 second long no one's gonna like it
- Energy is also important as a song which give energy is mostly want by everyone despite of age
- Danceability is aslo an important as if the song is good at dancing most of the people like the song though they cannot dance as they are admire by the beautiful dance

In [ ]:
```
# scatter plots with linear regression line
fig, (ax1, ax2) = plt.subplots(ncols=2, sharex=True, figsize=(15,10));
sns.regplot(x=song_dat['popularity'], y=song_dat['loudness'],ax=ax1);
sns.regplot(x=song_dat['popularity'], y=song_dat['energy'],ax=ax2);
plt.tight_layout()
```

In [553]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, sharex=True, figsize=(15,10));
sns.regplot(x=song_dat['popularity'], y=song_dat['danceability'],ax=ax1);
sns.regplot(x=song_dat['popularity'], y=song_dat['acousticness'],ax=ax2);
plt.tight_layout()
```



In [325]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, sharex=True, figsize=(15,10));
sns.regplot(x=song_dat['popularity'], y=song_dat['valence'],ax=ax1);
sns.regplot(x=song_dat['popularity'], y=song_dat['speechiness'],ax=ax2);
plt.tight_layout()
```



Comment on the importance of dimensionality reduction techniques, share your ideas and explain your observations.

In [334]:
```
X=new_song_dat.drop(['name','album','release_date','id','uri'],axis=1)
X.head()
```

Out[334]:

	track_number	acousticness	danceability	energy	instrumentalness	liveness	loudness	speec
0	1	0.0824	0.463	0.993	0.996000	0.932	-12.913	
1	2	0.4370	0.326	0.965	0.233000	0.961	-4.803	
2	3	0.4160	0.386	0.969	0.400000	0.956	-4.936	
3	4	0.5670	0.369	0.985	0.000107	0.895	-5.535	
4	5	0.4000	0.303	0.969	0.055900	0.966	-5.098	

In [336]:
```
X.shape
```

Out[336]: (1610, 12)

In [338]:
```
scaler=MinMaxScaler()
```

In [379]:
```
scale_song_df=pd.DataFrame(scaler.fit_transform(X),columns=X.columns)
scale_song_df.head()
```

Out[379]:

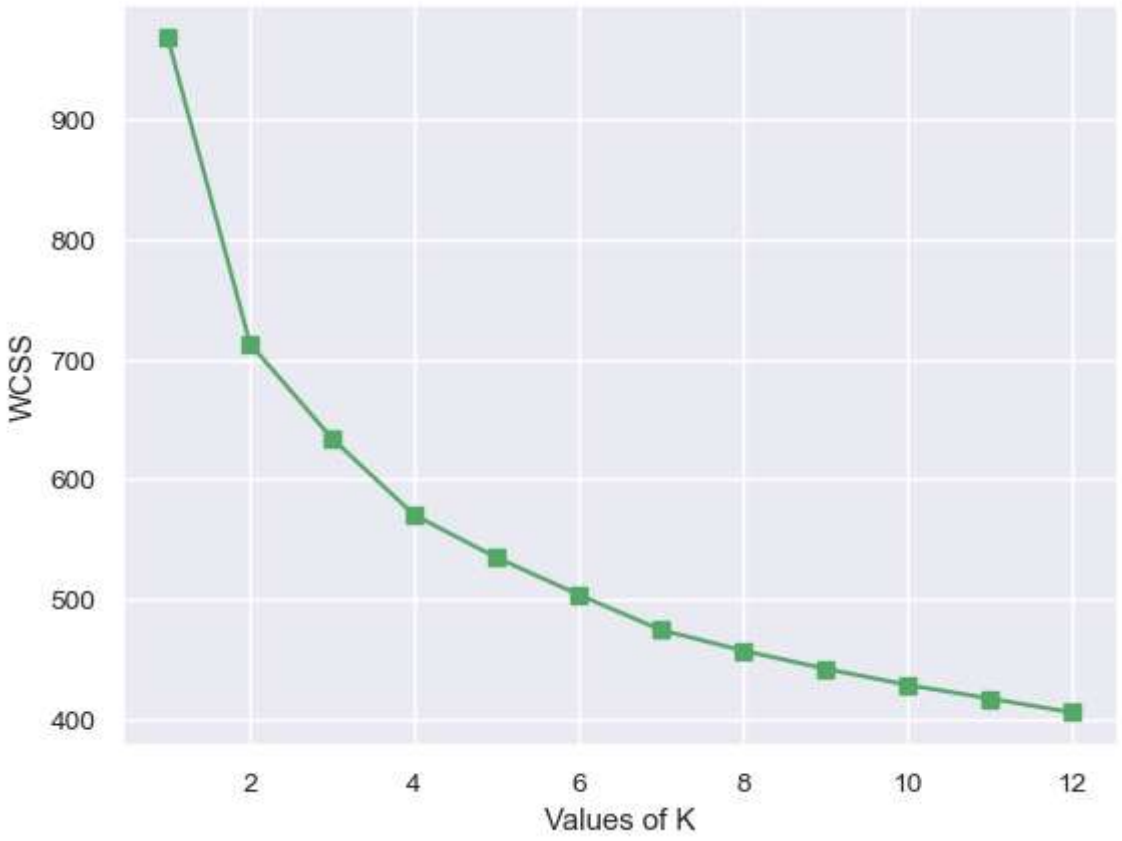
	track_number	acousticness	danceability	energy	instrumentalness	liveness	loudness	spe
0	0.000000	0.090144	0.458493	0.992032	1.000000	0.932384	0.235184	
1	0.021739	0.478113	0.283525	0.954847	0.233936	0.962094	0.756460	
2	0.043478	0.455137	0.360153	0.960159	0.401606	0.956972	0.747911	
3	0.065217	0.620346	0.338442	0.981408	0.000107	0.894478	0.709410	
4	0.086957	0.437631	0.254151	0.960159	0.056124	0.967216	0.737498	

In [371]:
```
scale_song=scaler.fit_transform(X)
```

Perform Cluster Analysis

Identify the right number of clusters

In [372]:
```
wcss={}

for i in range(1,13):
    kmeans=KMeans(n_clusters=i,init='k-means++',random_state=1)
    kmeans.fit(scale_song)
    wcss[i]=kmeans.inertia_

plt.plot(wcss.keys(),wcss.values(),'gs-')
plt.xlabel('Values of K')
plt.ylabel('WCSS')
plt.show()
```



In [536]:
```
for indx,year in enumerate(np.flip(years)):
    plt.title(year)
    sns.heatmap(years_df[indx].corr(),annot=True,annot_kws={'size': 6})
    plt.show()
```

In [373]:
```
wcss
```

Out[373]:
```
{1: 968.4874549469472,
 2: 713.5626145121196,
 3: 634.2570862999588,
 4: 570.533984132216,
 5: 534.9874942148475,
 6: 504.03299316989177,
 7: 474.6086886295142,
 8: 457.3522065348385,
 9: 442.25359276929873,
 10: 428.7429218031383,
 11: 417.4095203192527,
 12: 405.8281504175772}
```

1964

- track_number,danceability,speechiness are corelated with the popularity

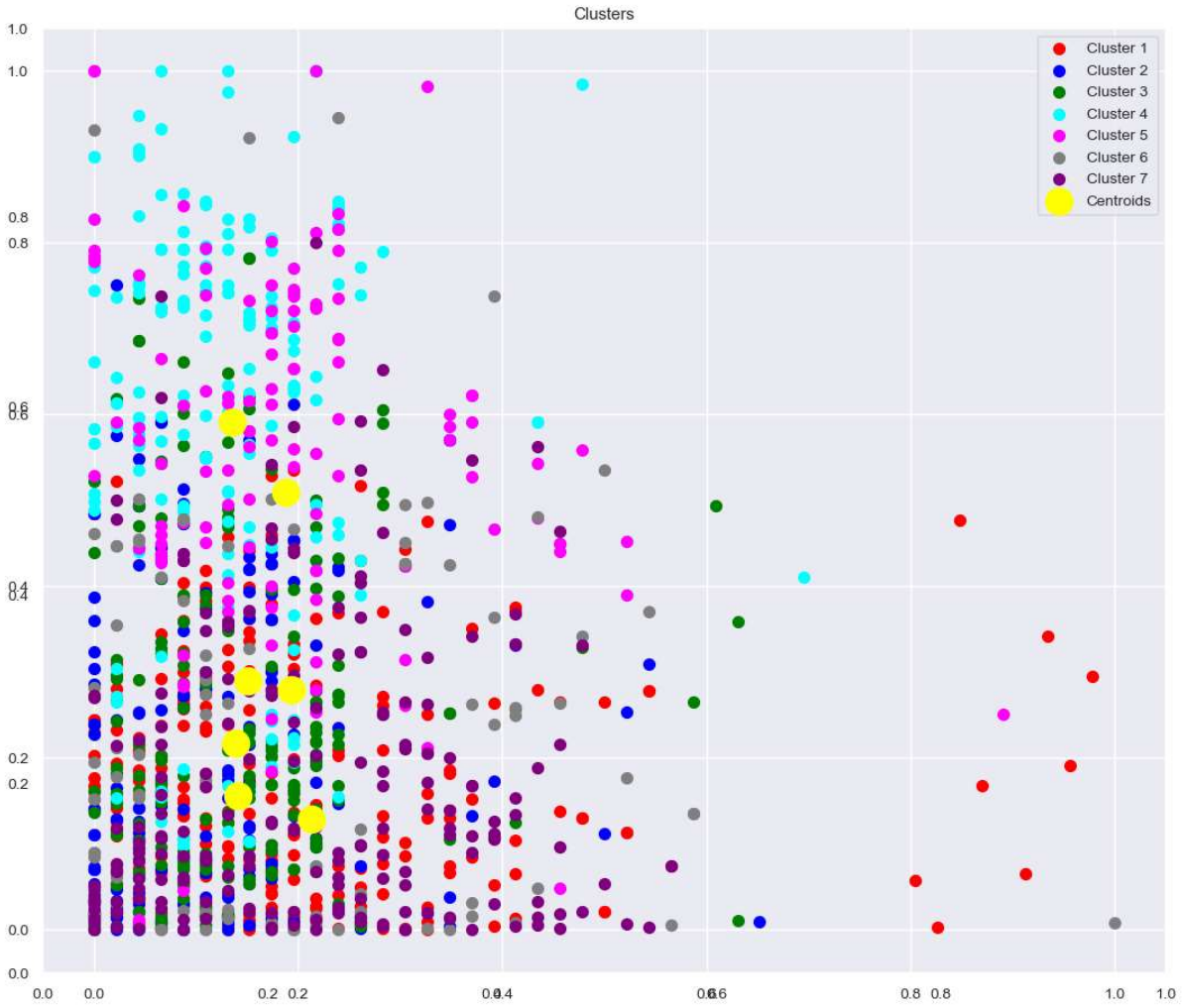
Use appropriate clustering algorithm

In [374]:
```
kmeans=KMeans(n_clusters=7)
kmeans.fit(scale_song)
```

Out[374]:
```
KMeans(n_clusters=7)
```

In [376]:
```
# visualizing clusters
fig, ax = plt.subplots(figsize=(13,11))
ax = fig.add_subplot(111)
plt.scatter(scale_song[y_kmeans == 0,0],scale_song[y_kmeans == 0,1], s= 50, c=
plt.scatter(scale_song[y_kmeans == 1,0], scale_song[y_kmeans == 1,1], s= 50, c=
plt.scatter(scale_song[y_kmeans == 2,0], scale_song[y_kmeans == 2,1], s= 50, c=
plt.scatter(scale_song[y_kmeans == 3,0], scale_song[y_kmeans == 3,1], s= 50, c=
plt.scatter(scale_song[y_kmeans == 4,0], scale_song[y_kmeans == 4,1], s= 50, c=
plt.scatter(scale_song[y_kmeans == 5,0], scale_song[y_kmeans == 5,1], s= 50, c=
plt.scatter(scale_song[y_kmeans == 6,0], scale_song[y_kmeans == 6,1], s= 50, c=

# centroids
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s= 300
plt.title('Clusters')
plt.legend()
```

Out[376]: <matplotlib.legend.Legend at 0x1745eef03a0>



In [408]:
```
scale_song_df['cluster']=kmeans.labels_
scale_song_df.head()
```

Out[408]:

	track_number	acousticness	danceability	energy	instrumentalness	liveness	loudness	spe
0	0.000000	0.090144	0.458493	0.992032	1.000000	0.932384	0.235184	
1	0.021739	0.478113	0.283525	0.954847	0.233936	0.962094	0.756460	
2	0.043478	0.455137	0.360153	0.960159	0.401606	0.956972	0.747911	
3	0.065217	0.620346	0.338442	0.981408	0.000107	0.894478	0.709410	
4	0.086957	0.437631	0.254151	0.960159	0.056124	0.967216	0.737498	

Define each cluster based on the features

In [394]:
```
scale_song_df.groupby('cluster')['popularity'].mean().sort_values(ascending=Fal
```

Out[394]:
```
cluster
6    0.404493
4    0.396977
3    0.323332
2    0.264832
5    0.263649
0    0.258334
1    0.248812
Name: popularity, dtype: float64
```

In [395]:
```
# Number of song for each cluster
scale_song_df['cluster'].value_counts()
```

Out[395]:
```
0    334
4    313
1    267
3    193
6    188
2    160
5    155
Name: cluster, dtype: int64
```

In [413]:
```
new_song_dat['cluster']=kmeans.labels_
```

In [416]:
```
# Checking the song for cluster 6 as it has the highly most popular song
new_song_dat.loc[new_song_dat['cluster']==6].sort_values('popularity',ascending
```

Out[416]:

uri	acousticness	danceability	energy	instrumentalness	liveness	loudness	spee
/a4jFySlun4jLSuMhiq	0.6700	0.429	0.554	0.000152	0.1050	-6.128	
)op3rbfAkc1LGXgipW	0.6870	0.432	0.389	0.010700	0.0788	-6.517	
oTC0KyXMq1Oxfien0	0.5180	0.702	0.668	0.000000	0.0588	-9.237	
CGoWYBabAeVWGiD	0.4470	0.634	0.630	0.039000	0.1700	-8.277	
:7pn6Lme1NP7qQqQ	0.6400	0.320	0.620	0.000064	0.2530	-9.686	
...	...	...	...	...	...	...	
G5fK61iLghV8DeIbfX	0.5890	0.628	0.370	0.000034	0.0849	-14.689	
y6NunNMBuWPIWSz	0.0977	0.342	0.300	0.004190	0.1220	-12.071	
)WI6kJJKkhKZAK9Ep	0.0910	0.337	0.300	0.004690	0.1200	-12.182	
4ki6nMV2qgyaPYOfl1	0.7660	0.648	0.532	0.000000	0.4180	-9.344	
)95bg4K9fJAjWX0NY	0.4520	0.265	0.667	0.166000	0.3540	-8.868	

In [ ]:
```
```