In this ML model we would predict the popularity of music tracks based on their audio featuresFor this prediction we would be using regression techniques to forecast/predict the popularity score of song/track based on various music features and metadata Expected results would include accurate predictions of a song's future performance in terms of streams,downloads and chart positions Dataset should include songs with their musical features and historical data on song's popularity Problem Statement: develop a predictive model that can accurately estimate the popularity of music tracks based on their audio features

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
df=pd.read_csv('Spotify_data.csv')
```

In [3]:
```python
df
```

Out[3]:

	Unnamed: 0	Track Name	Artists	Album Name	Album ID	Track ID	Popularity	Relea Da
0	0	Not Like Us	Kendrick Lamar	Not Like Us	5JjnoGJyOxfSZUZtk2rRwZ	6AI3ezQ4o3HUoP6Dhudph3	96	202 05-
1	1	Houdini	Eminem	Houdini	6Xuu2z00jxRPZei4IJ9neK	2HYFX63wP3otVIvopRS99Z	94	202 05-
2	2	BAND4BAND (feat. Lil Baby)	Central Cee, Lil Baby	BAND4BAND (feat. Lil Baby)	4AzPr5SUpNF553eC1d3aRy	7iabz12vAuVQYyekFIWJxD	91	202 05-
3	3	I Don't Wanna Wait	David Guetta, OneRepublic	I Don't Wanna Wait	0wCLHkBRKcndhMQQpeo8Ji	331l3xABO0HMr1Kkyh2LZq	90	202 04-
4	4	Pedro	Jaxomy, Agatino Romero, Raffaella Carrà	Pedro	5y6RXjI5VPR0RyInghTbf1	48lxT5qJF0yYyf2z4wB4xW	89	202 03-
...	...	...	...	...	...	...	...	
222	222	Tu Chahiye	Pritam, Atif Aslam	Bajrangi Bhaijaan	4nZOPP0atfJbBlkedLYi7t	3aaiAWCet6sbfOfLSn3g7i	66	201 07-
223	223	Aabaad Barbaad (From "Ludo")	Pritam, Arijit Singh	Aabaad Barbaad (From "Ludo")	1PzsfgcbPbiW7uflc9Zi5Z	0hFUtSsV2itYEUTZGj6w5H	58	202 10-
224	224	Jag Ghoomeya	Vishal-Shekhar, Rahat Fateh Ali Khan, Irshad K...	Sultan	0tAi6H8acUKefYMIEuxcMA	4KCbZcshgibfJSCAkg87Lv	62	201 05-
225	225	Tumhe Kitna Pyaar Karte (From "Bawaal")	Mithoon, Arijit Singh, Manoj Muntashir	Tumhe Kitna Pyaar Karte (From "Bawaal")	20zQZcEhMLsDUn1LhPCEFY	03hJuEQpEQERrHpjcXKWzJ	65	202 07-
226	226	Bekhayali	Sachet Tandon	Kabir Singh	3uuu6u13U0KeVQsZ3CZKK4	4yMbbysldl7E3WgiaugnwM	61	201 06-

227 rows × 22 columns

In [4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0        227 non-null    int64  
 1   Track Name        227 non-null    object 
 2   Artists           227 non-null    object 
 3   Album Name        227 non-null    object 
 4   Album ID          227 non-null    object 
 5   Track ID          227 non-null    object 
 6   Popularity        227 non-null    int64  
 7   Release Date      227 non-null    object 
 8   Duration (ms)     227 non-null    int64  
 9   Explicit          227 non-null    bool   
 10  External URLs     227 non-null    object 
 11  Danceability      227 non-null    float64
 12  Energy            227 non-null    float64
 13  Key               227 non-null    int64  
 14  Loudness          227 non-null    float64
 15  Mode              227 non-null    int64  
 16  Speechiness       227 non-null    float64
 17  Acousticness      227 non-null    float64
 18  Instrumentalness  227 non-null    float64
 19  Liveness          227 non-null    float64
 20  Valence           227 non-null    float64
 21  Tempo             227 non-null    float64
dtypes: bool(1), float64(9), int64(5), object(7)
memory usage: 37.6+ KB
```

In [5]:
```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

In [6]:
```
df
```

Out[6]:

	Track Name	Artists	Album Name	Album ID	Track ID	Popularity	Release Date	Duratior (ms
0	Not Like Us	Kendrick Lamar	Not Like Us	5JjnoGJyOxfSZUZtk2rRwZ	6AI3ezQ4o3HUoP6Dhudph3	96	2024-05-04	27419
1	Houdini	Eminem	Houdini	6Xuu2z00jxRPZei4lJ9neK	2HYFX63wP3otVIvopRS99Z	94	2024-05-31	22723
2	BAND4BAND (feat. Lil Baby)	Central Cee, Lil Baby	BAND4BAND (feat. Lil Baby)	4AzPr5SUpNF553eC1d3aRy	7iabz12vAuVQYyekFIWJxD	91	2024-05-23	14073
3	I Don't Wanna Wait	David Guetta, OneRepublic	I Don't Wanna Wait	0wCLHkBRKcndhMQQpeo8Ji	331I3xABO0HMr1Kkyh2LZq	90	2024-04-05	14966
4	Pedro	Jaxomy, Agatino Romero, Raffaella Carrà	Pedro	5y6RXjI5VPR0RyInghTbf1	48lxT5qJF0yYyf2z4wB4xW	89	2024-03-29	14484
...	...	...	...	...	...	...	...	..
222	Tu Chahiye	Pritam, Atif Aslam	Bajrangi Bhaijaan	4nZOPP0atfJbBlkedLYi7t	3aaiAWCet6sbfOfLSn3g7i	66	2015-07-07	27268
223	Aabaad Barbaad (From "Ludo")	Pritam, Arijit Singh	Aabaad Barbaad (From "Ludo")	1PzsfgcbPbiW7uflc9Zi5Z	0hFUtSsV2itYEUTZGj6w5H	58	2020-10-21	30910
224	Jag Ghoomeya	Vishal-Shekhar, Rahat Fateh Ali Khan, Irshad K...	Sultan	0tAi6H8acUKefYMIEuxcMA	4KCbZcshgibfJSCAkg87Lv	62	2016-05-31	28199
225	Tumhe Kitna Pyaar Karte (From "Bawaal")	Mithoon, Arijit Singh, Manoj Muntashir	Tumhe Kitna Pyaar Karte (From "Bawaal")	20zQZcEhMLsDUn1LhPCEFY	03hJuEQpEQERrHpjcXKWzJ	65	2023-07-07	30523
226	Bekhayali	Sachet Tandon	Kabir Singh	3uuu6u13U0KeVQsZ3CZKK4	4yMbbysldl7E3WgiaugnwM	61	2019-06-14	37179

227 rows × 21 columns

In [7]:
```
df[df.duplicated()]
```

Out[7]:

	Track Name	Artists	Album Name	Album ID	Track ID	Popularity	Release Date	Dur
152	Tum Se (From "Teri Baaton Mein Aisa Uljha Jiya")	Sachin-Jigar, Raghav Chaitanya, Varun Jain, In...	Tum Se (From "Teri Baaton Mein Aisa Uljha Jiya")	3vVIhgkDoC0vRBba5drHPe	2ceeTJAzKy295Fm0VsaXtE	78	2024-02-02	26
154	Sajni (From "Laapataa Ladies")	Ram Sampath, Arijit Singh, Prashant Pandey	Sajni (From "Laapataa Ladies")	3I3kZyHUtEA9Y59rJkxtk6	5zCnGtCl5Ac5zlFHXaZmhy	83	2024-02-12	17
155	Dekhha Tenu (From "Mr. And Mrs. Mahi")	Mohammad Faiz, Jaani	Dekhha Tenu (From "Mr. And Mrs. Mahi")	1C3FmwSQAbjnZR6GRgnWQc	34Fh4HXZmnuBdtgejWUZg2	81	2024-05-14	28
158	Agar Ho Tum (From "Mr. And Mrs. Mahi")	Tanishk Bagchi, Kausar Munir	Agar Ho Tum (From "Mr. And Mrs. Mahi")	08PRzEfce7mwprUTvMmfh2	0a17mIL7XTvYqe9mxuPd3y	71	2024-05-20	25
163	Soni Soni (From "Ishq Vishk Rebound")	Darshan Raval, Jonita Gandhi, Rochak Kohli, Gu...	Soni Soni (From "Ishq Vishk Rebound")	3vBso6gFPmEwstdMXn3Ahi	36N5awamOX6XX5pQn3aFXZ	77	2024-05-24	17
164	Maiyya Mainu	Sachet Tandon	Jersey (Original Motion Picture Soundtrack)	1FrTddcjO9PzPaJX7SkQEC	3ygfdwvBJ2Y5XhJiiHFFZE	70	2022-04-26	23
166	Pehle Bhi Main	Vishal Mishra, Raj Shekhar	ANIMAL	0a183xiCHiC1GQd8ou7WXO	7yDHHVKLbvDmVw1XXhDDIO	80	2023-11-24	25
167	Apna Bana Le	Sachin-Jigar, Arijit Singh, Amitabh Bhattacharya	Bhediya (Original Motion Picture Soundtrack)	12sC6UjMWz6EaxnzyfCNMe	5bQ6oDLqvw8tywmnSmwEyL	74	2022-11-22	26
170	Satranga (From "ANIMAL")	Arijit Singh, Shreyas Puranik, Siddharth - Garima	Satranga (From "ANIMAL")	5mZX4EMwEyohNmVfLTDtXn	3yHyiUDJdz02FZ6jfUbsmY	80	2023-10-27	27
172	Kesariya (From "Brahmastra")	Pritam, Arijit Singh, Amitabh Bhattacharya	Kesariya (From "Brahmastra")	1HeX4SmCFW4EPHQDvHgrVS	6VBhH7CyP56BXjp8VsDFPZ	71	2022-07-17	26
173	Tera Ban Jaunga	Akhil Sachdeva, Tulsi Kumar	Kabir Singh	3uuu6u13U0KeVQsZ3CZKK4	3oWxFNsXstcancCR1wODR4	67	2019-06-14	23
175	Tum Kya Mile (From "Rocky Aur Rani Kii Prem Ka...	Pritam, Arijit Singh, Shreya Ghoshal, Amitabh ...	Tum Kya Mile (From "Rocky Aur Rani Kii Prem Ka...	5FtQVEQsWzRcpqh820ZoII	06LCamFUOtImIKi9mFRKiD	73	2023-06-28	27
188	Phir Aur Kya Chahiye (From "Zara Hatke Zara Ba...	Sachin-Jigar, Arijit Singh, Amitabh Bhattacharya	Phir Aur Kya Chahiye (From "Zara Hatke Zara Ba...	6j4QpObdnZpxNU52o2egBZ	5QW9K4A1gMnIi94YUxtsfM	70	2023-05-16	26
190	Raabta	Pritam, Arijit Singh	Agent Vinod	2DqQ34i4uuuZWTScsGIgHr	6FjbAnaPRPwiP3sciEYctO	70	2012-02-24	24
208	Agar Tum Saath Ho	Alka Yagnik, Arijit Singh	Tamasha	2CUXo26JAWIbQx0EVMnjpA	3hkC9EHFZNQPXrtl8WPHnX	71	2015-10-16	34

15 rows × 21 columns

In [8]:

```
df.isnull().sum()
```

Out[8]:
```
Track Name          0
Artists             0
Album Name          0
Album ID            0
Track ID            0
Popularity          0
Release Date        0
Duration (ms)       0
Explicit            0
External URLs       0
Danceability        0
Energy              0
Key                 0
Loudness            0
Mode                0
Speechiness         0
Acousticness        0
Instrumentalness    0
Liveness            0
Valence             0
Tempo               0
dtype: int64
```

In [10]:
```
df['Release Date']
```

Out[10]:
```
0      2024-05-04
1      2024-05-31
2      2024-05-23
3      2024-04-05
4      2024-03-29
          ...
222    2015-07-07
223    2020-10-21
224    2016-05-31
225    2023-07-07
226    2019-06-14
Name: Release Date, Length: 227, dtype: object
```

In [11]:
```
df['release_yr']=df['Release Date'].str.split('-').str[0]
```

In [12]:
```
df['release_month']=df['Release Date'].str.split('-').str[1]
df['release_date']=df['Release Date'].str.split('-').str[2]
```

In [13]:
```
df.columns.tolist()
```

Out[13]:
```
['Track Name',
 'Artists',
 'Album Name',
 'Album ID',
 'Track ID',
 'Popularity',
 'Release Date',
 'Duration (ms)',
 'Explicit',
 'External URLs',
 'Danceability',
 'Energy',
 'Key',
 'Loudness',
 'Mode',
 'Speechiness',
 'Acousticness',
 'Instrumentalness',
 'Liveness',
 'Valence',
 'Tempo',
 'release_yr',
 'release_month',
 'release_date']
```

In [14]:
```
df[['release_date','release_yr','release_month']]
```

Out[14]:

	release_date	release_yr	release_month
0	04	2024	05
1	31	2024	05
2	23	2024	05
3	05	2024	04
4	29	2024	03
...	...	...	...
222	07	2015	07
223	21	2020	10
224	31	2016	05
225	07	2023	07
226	14	2019	06

227 rows × 3 columns

In [15]:
```
df=df.drop(columns=['Release Date'],axis=1)
```

In [17]:
```
for i in ['release_date','release_yr','release_month']:
    df[i]=df[i].astype(int)
```

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Track Name        227 non-null    object
 1   Artists           227 non-null    object
 2   Album Name        227 non-null    object
 3   Album ID          227 non-null    object
 4   Track ID          227 non-null    object
 5   Popularity        227 non-null    int64
 6   Duration (ms)     227 non-null    int64
 7   Explicit          227 non-null    bool
 8   External URLs     227 non-null    object
 9   Danceability      227 non-null    float64
 10  Energy            227 non-null    float64
 11  Key               227 non-null    int64
 12  Loudness          227 non-null    float64
 13  Mode              227 non-null    int64
 14  Speechiness       227 non-null    float64
 15  Acousticness      227 non-null    float64
 16  Instrumentalness  227 non-null    float64
 17  Liveness          227 non-null    float64
 18  Valence           227 non-null    float64
 19  Tempo             227 non-null    float64
 20  release_yr        227 non-null    int32
 21  release_month     227 non-null    int32
 22  release_date      227 non-null    int32
dtypes: bool(1), float64(9), int32(3), int64(4), object(6)
memory usage: 36.7+ KB
```

In [19]:
```
[i for i in df.columns if df[i].dtype =='object' ]
```

Out[19]:
```
['Track Name',
 'Artists',
 'Album Name',
 'Album ID',
 'Track ID',
 'External URLs']
```

In [20]:
```
[i for i in df.columns if df[i].dtype != 'object']
```

Out[20]:
```
['Popularity',
 'Duration (ms)',
 'Explicit',
 'Danceability',
 'Energy',
 'Key',
 'Loudness',
 'Mode',
 'Speechiness',
 'Acousticness',
 'Instrumentalness',
 'Liveness',
 'Valence',
 'Tempo',
 'release_yr',
 'release_month',
 'release_date']
```

In [22]:
```
df.head()
```

Out[22]:

	Track Name	Artists	Album Name	Album ID	Track ID	Popularity	Duration (ms)	Explicit
0	Not Like Us	Kendrick Lamar	Not Like Us	5JjnoGJyOxfSZUZtk2rRwZ	6AI3ezQ4o3HUoP6Dhudph3	96	274192	True
1	Houdini	Eminem	Houdini	6Xuu2z00jxRPZei4IJ9neK	2HYFX63wP3otVIvopRS99Z	94	227239	True
2	BAND4BAND (feat. Lil Baby)	Central Cee, Lil Baby	BAND4BAND (feat. Lil Baby)	4AzPr5SUpNF553eC1d3aRy	7iabz12vAuVQYyekFIWJxD	91	140733	True
3	I Don't Wanna Wait	David Guetta, OneRepublic	I Don't Wanna Wait	0wCLHkBRKcndhMQQpeo8Ji	331I3xABO0HMr1Kkyh2LZq	90	149668	False
4	Pedro	Jaxomy, Agatino Romero, Raffaella Carrà	Pedro	5y6RXjI5VPR0RyInghTbf1	48lxT5qJF0yYyf2z4wB4xW	89	144846	False

5 rows × 23 columns

In [23]:
```
df[['Explicit','Mode']]
```

Out[23]:

	Explicit	Mode
0	True	1
1	True	0
2	True	1
3	False	0
4	False	1
...	...	...
222	False	1
223	False	1
224	False	1
225	False	0
226	False	0

227 rows × 2 columns

In [24]:
```
df['Explicit'].value_counts()
```

Out[24]:
```
Explicit
False    171
True      56
Name: count, dtype: int64
```

In [25]:
```
df['Explicit'].replace({False:0,True:1})
#was getting NaN after using map, so used replace
```

Out[25]:
```
0      1
1      1
2      1
3      0
4      0
      ..
222    0
223    0
224    0
225    0
226    0
Name: Explicit, Length: 227, dtype: int64
```

EDA

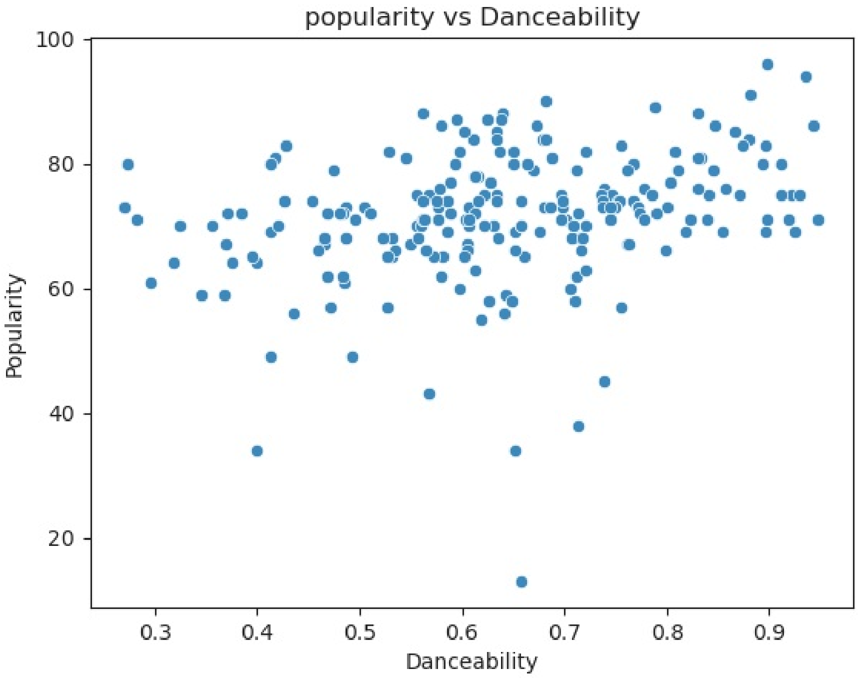
Target Var vs Music Features

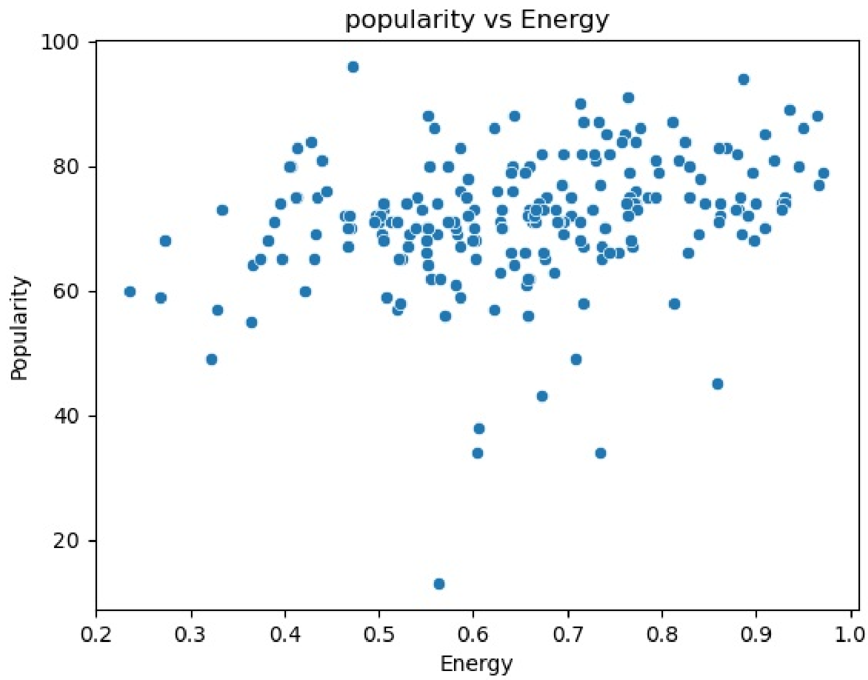
As popularity score is the col of prediction(target var.) See the relationship b/w all the music featues with taregt var

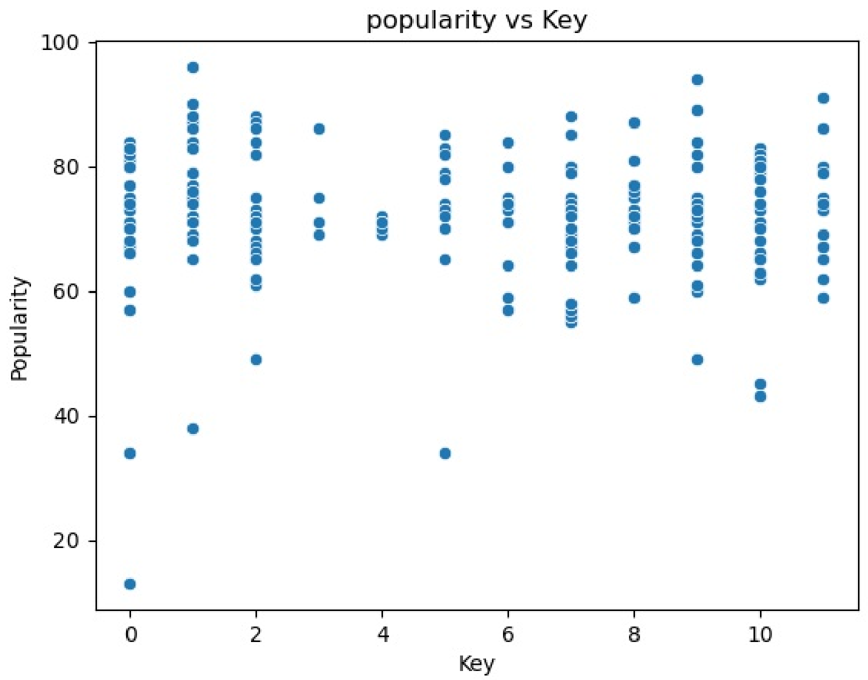
In [26]:
```
df.columns.tolist()
```

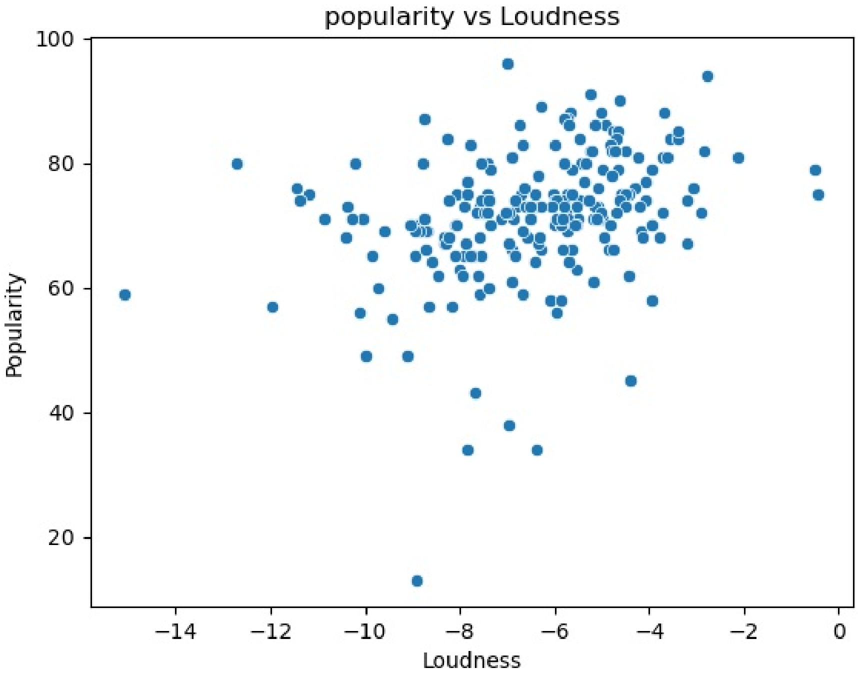
Out[26]:
```
['Track Name',
 'Artists',
 'Album Name',
 'Album ID',
 'Track ID',
 'Popularity',
 'Duration (ms)',
 'Explicit',
 'External URLs',
 'Danceability',
 'Energy',
 'Key',
 'Loudness',
 'Mode',
 'Speechiness',
 'Acousticness',
 'Instrumentalness',
 'Liveness',
 'Valence',
 'Tempo',
 'release_yr',
 'release_month',
 'release_date']
```

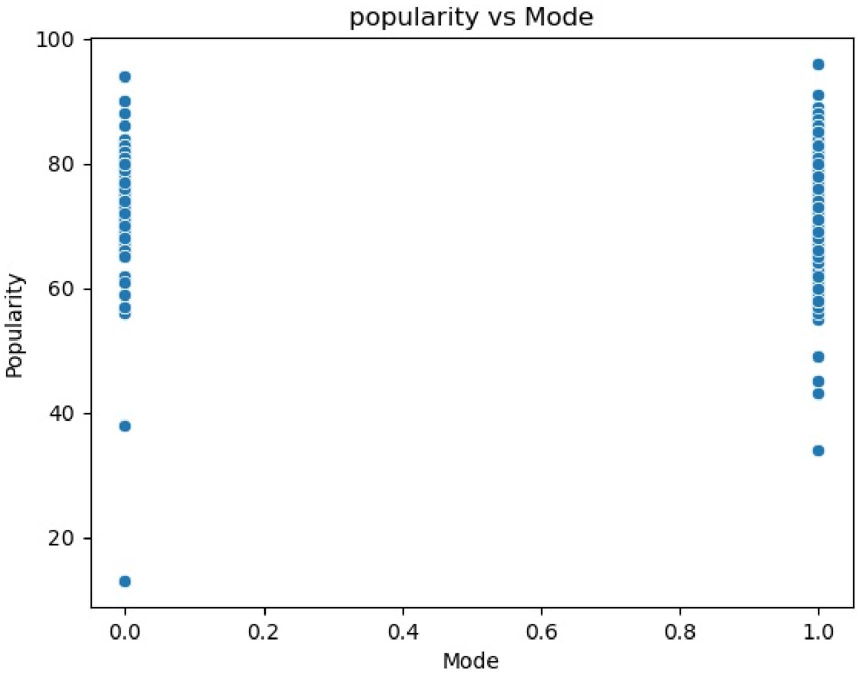
In [27]:
```
music_f=['Danceability','Energy','Key','Loudness','Mode','Speechiness',
 'Acousticness','Instrumentalness','Liveness','Valence','Tempo']
for i in music_f:
    sns.scatterplot(x=i,y='Popularity', data=df)
    plt.title(f'popularity vs {i}')
    plt.show()
```

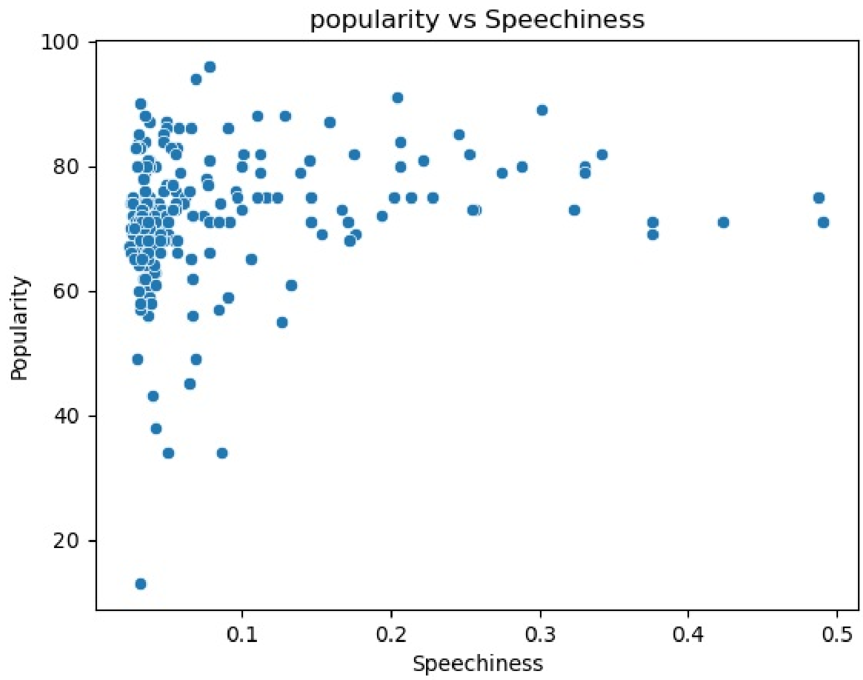


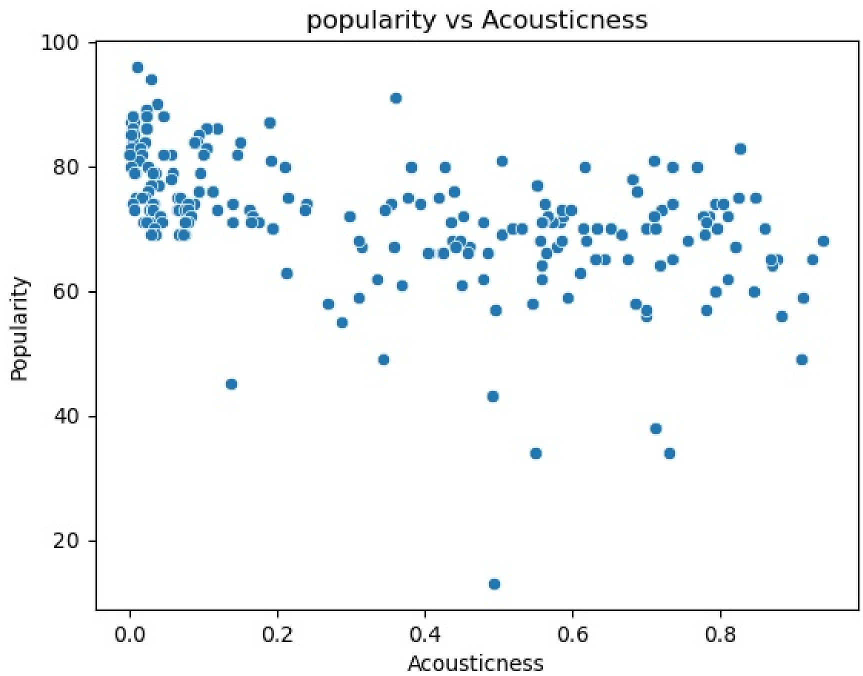

popularity vs Energy
Popularity
Energy
100
80
60
40
20
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

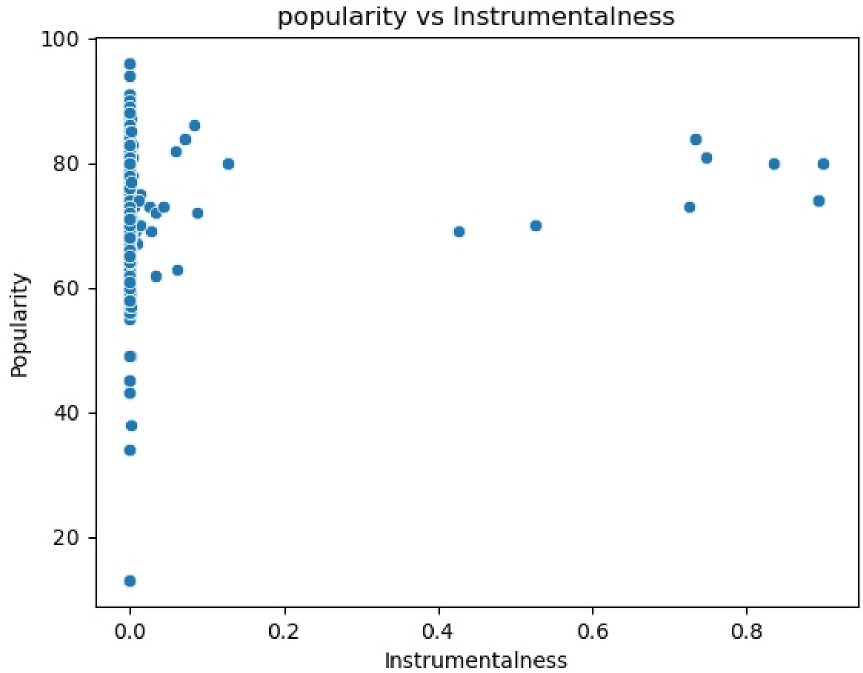

popularity vs Key
Popularity
Key
100
80
60
40
20
0
2
4
6
8
10

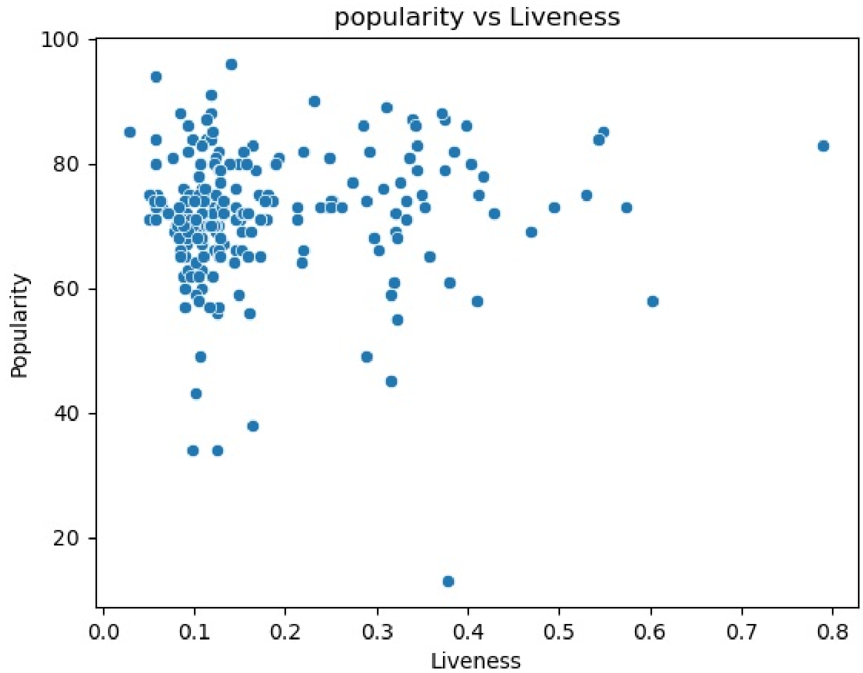

popularity vs Loudness
Popularity
Loudness
100
80
60
40
20
−14
−12
−10
−8
−6
−4
−2
0

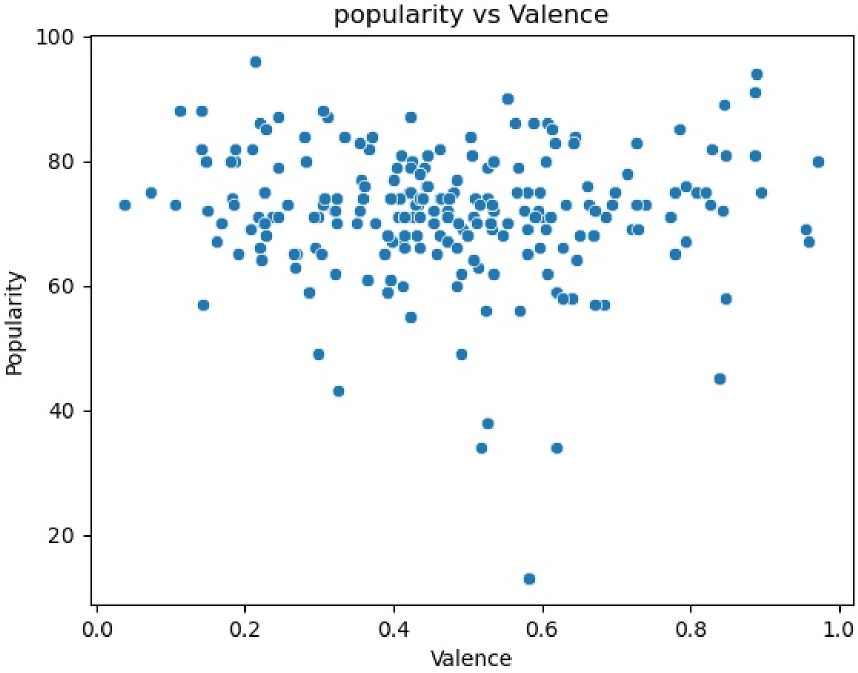

popularity vs Mode
Popularity
Mode
100
80
60
40
20
0.0
0.2
0.4
0.6
0.8
1.0

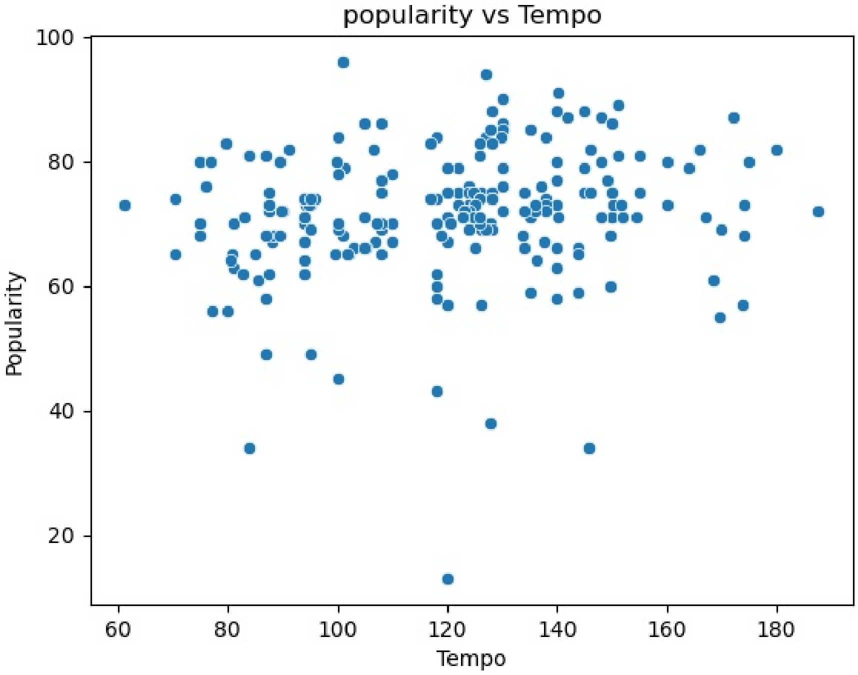

popularity vs Speechiness
Popularity
Speechiness
100
80
60
40
20
0.1
0.2
0.3
0.4
0.5


popularity vs Acousticness
Popularity
Acousticness
100
80
60
40
20
0.0
0.2
0.4
0.6
0.8


popularity vs Instrumentalness
Popularity
Instrumentalness
100
80
60
40
20
0.0
0.2
0.4
0.6
0.8


popularity vs Liveness
Popularity
Liveness
100
80
60
40
20
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8





Observation: 1. higher danceability, energy corelate with higher popularity score 2. higher acousticness has lower popularity score 3. lower loudness has lower popularity score 4. valence show weeker, unclear relationship(?) 5. tempo,liveness,instrumentalness,speechiness,key(?)

correlation b/w all the features

In [28]:
```
num_var=[i for i in df.columns if df[i].dtype != 'object' and 'bool']
```

In [29]:
```
num_data=df[num_var]
num_data
```

Out[29]:

	Popularity	Duration (ms)	Explicit	Danceability	Energy	Key	Loudness	Mode	Speechiness	Acousticness	Instrumentalness	Liv
0	96	274192	True	0.898	0.472	1	-7.001	1	0.0776	0.0107	0.000000	
1	94	227239	True	0.936	0.887	9	-2.760	0	0.0683	0.0292	0.000002	
2	91	140733	True	0.882	0.764	11	-5.241	1	0.2040	0.3590	0.000000	
3	90	149668	False	0.681	0.714	1	-4.617	0	0.0309	0.0375	0.000000	
4	89	144846	False	0.788	0.936	9	-6.294	1	0.3010	0.0229	0.000001	
...	...	...	...	...	...	...	...	...	...	...	...	
222	66	272680	False	0.565	0.744	7	-5.817	1	0.0446	0.4030	0.000000	
223	58	309103	False	0.626	0.522	7	-5.857	1	0.0317	0.6860	0.000000	
224	62	281992	False	0.484	0.565	11	-7.954	1	0.0347	0.4790	0.000002	
225	65	305232	False	0.602	0.374	10	-9.849	0	0.0328	0.9240	0.000008	
226	61	371791	False	0.296	0.582	9	-5.180	0	0.0413	0.4490	0.000000	

227 rows × 17 columns

In [30]:

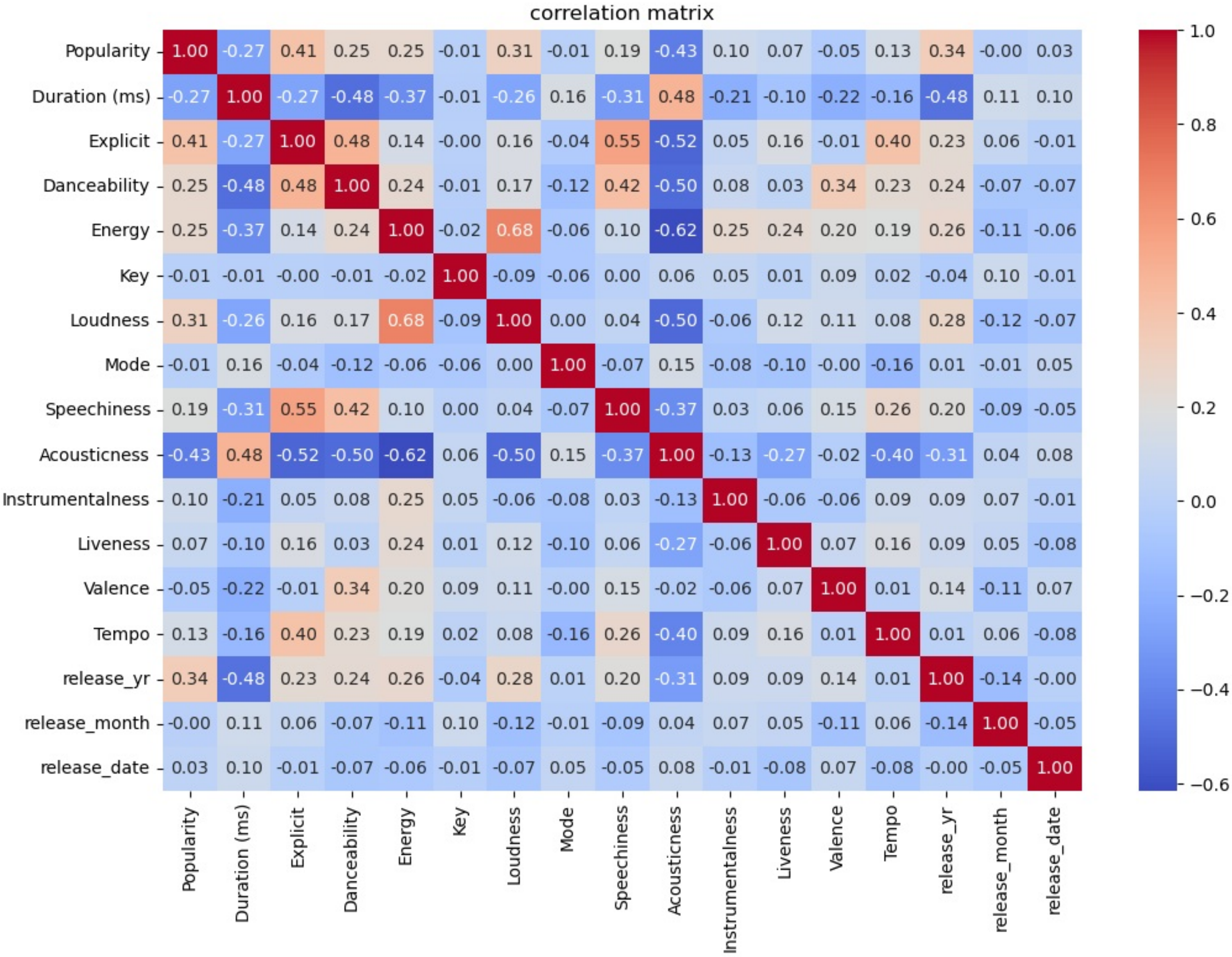
```
corr_matrix=num_data.corr()
corr_matrix
```

Out[30]:

	Popularity	Duration (ms)	Explicit	Danceability	Energy	Key	Loudness	Mode	Speechiness	Acousticn
Popularity	1.000000	-0.269510	0.405406	0.251928	0.250068	-0.008550	0.308110	-0.008246	0.190621	-0.43
Duration (ms)	-0.269510	1.000000	-0.265563	-0.484826	-0.365698	-0.012312	-0.256522	0.158170	-0.312642	0.47
Explicit	0.405406	-0.265563	1.000000	0.482242	0.137504	-0.004511	0.163645	-0.043564	0.551908	-0.51
Danceability	0.251928	-0.484826	0.482242	1.000000	0.242587	-0.013330	0.166232	-0.118235	0.419217	-0.49
Energy	0.250068	-0.365698	0.137504	0.242587	1.000000	-0.017352	0.678558	-0.063101	0.103059	-0.61
Key	-0.008550	-0.012312	-0.004511	-0.013330	-0.017352	1.000000	-0.093016	-0.061717	0.004854	0.05
Loudness	0.308110	-0.256522	0.163645	0.166232	0.678558	-0.093016	1.000000	0.001383	0.037858	-0.50
Mode	-0.008246	0.158170	-0.043564	-0.118235	-0.063101	-0.061717	0.001383	1.000000	-0.069425	0.14
Speechiness	0.190621	-0.312642	0.551908	0.419217	0.103059	0.004854	0.037858	-0.069425	1.000000	-0.36
Acousticness	-0.431117	0.476488	-0.517975	-0.498951	-0.616124	0.055651	-0.503469	0.147149	-0.367282	1.00
Instrumentalness	0.104846	-0.212550	0.050648	0.077016	0.250163	0.054473	-0.057236	-0.079684	0.031381	-0.12
Liveness	0.066110	-0.104685	0.160904	0.030781	0.239486	0.013291	0.121415	-0.096436	0.055212	-0.26
Valence	-0.045580	-0.217561	-0.006032	0.338242	0.201095	0.093400	0.107138	-0.004017	0.148823	-0.02
Tempo	0.131820	-0.160446	0.404604	0.232993	0.186659	0.015901	0.079587	-0.159426	0.264213	-0.40
release_yr	0.338920	-0.476429	0.230559	0.237685	0.259509	-0.038246	0.277937	0.009872	0.201099	-0.30
release_month	-0.002076	0.114417	0.057039	-0.070265	-0.106993	0.099660	-0.121783	-0.007514	-0.086650	0.03
release_date	0.025284	0.096617	-0.008382	-0.071339	-0.064620	-0.013759	-0.073216	0.051247	-0.047201	0.08

In [31]:

```
plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix,annot=True,cmap='coolwarm',fmt='.2f')
plt.title('correlation matrix')
plt.show()
```



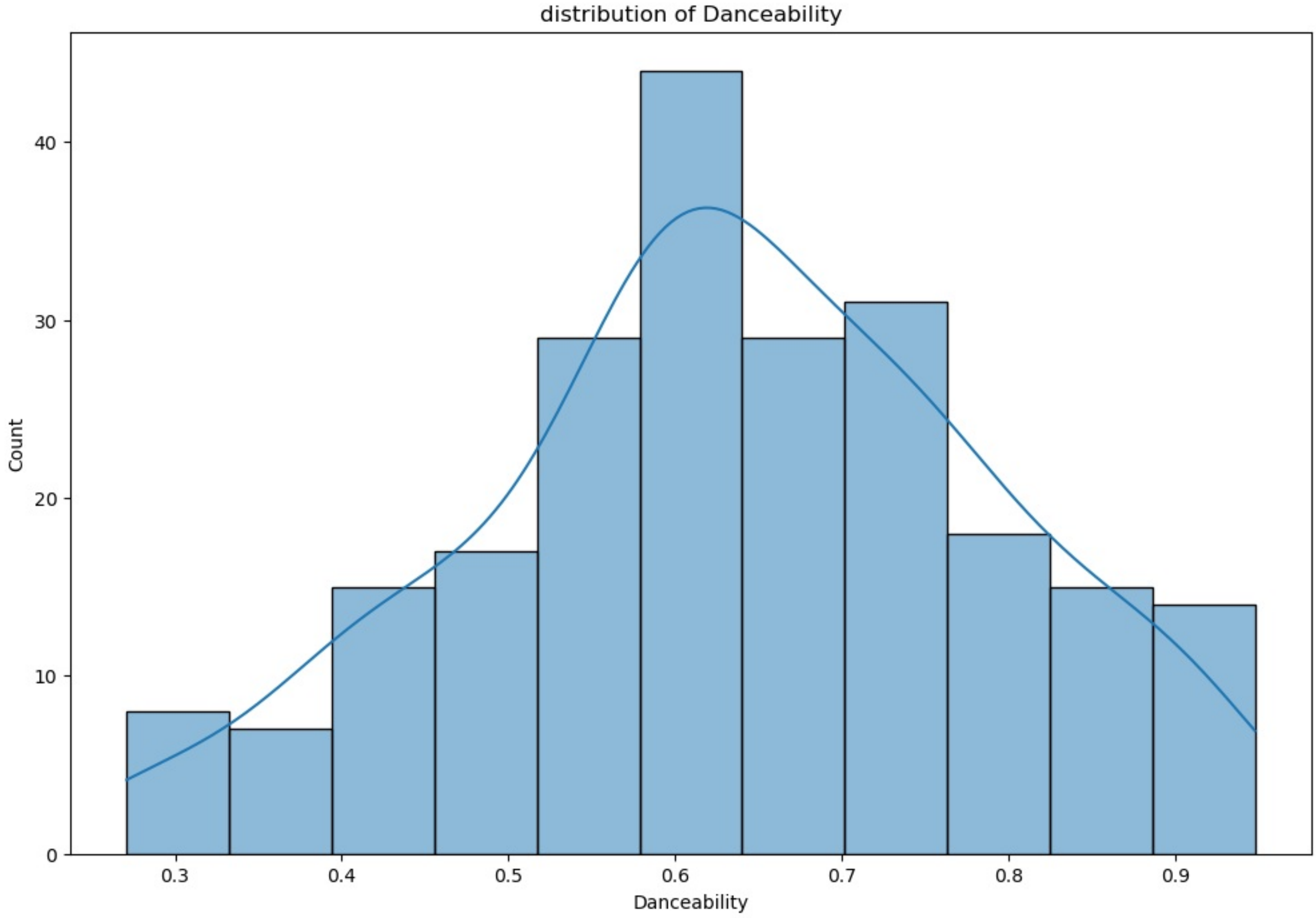
observation: -ve correlation-> 1.duration has moderate -ve correlation with popularity score 2.key,mode,valence,release month has low -ve correaltion with popularity score 3.acousticness has high -ve correlation with popularity +ve correlation-> 4.explicit,loudness,release yr has high +ve correlation with popularity score 5.danceability,energy,speechiness has moderate +ve correlation with popularity 6. instrumentalness,liveness,tempo, release date has low +ve correlation with popularity These above correlation's show that: 1. loud,dance able and energitic,speech songs has higher & moderate popularity 2. acoustic songs has less popularity

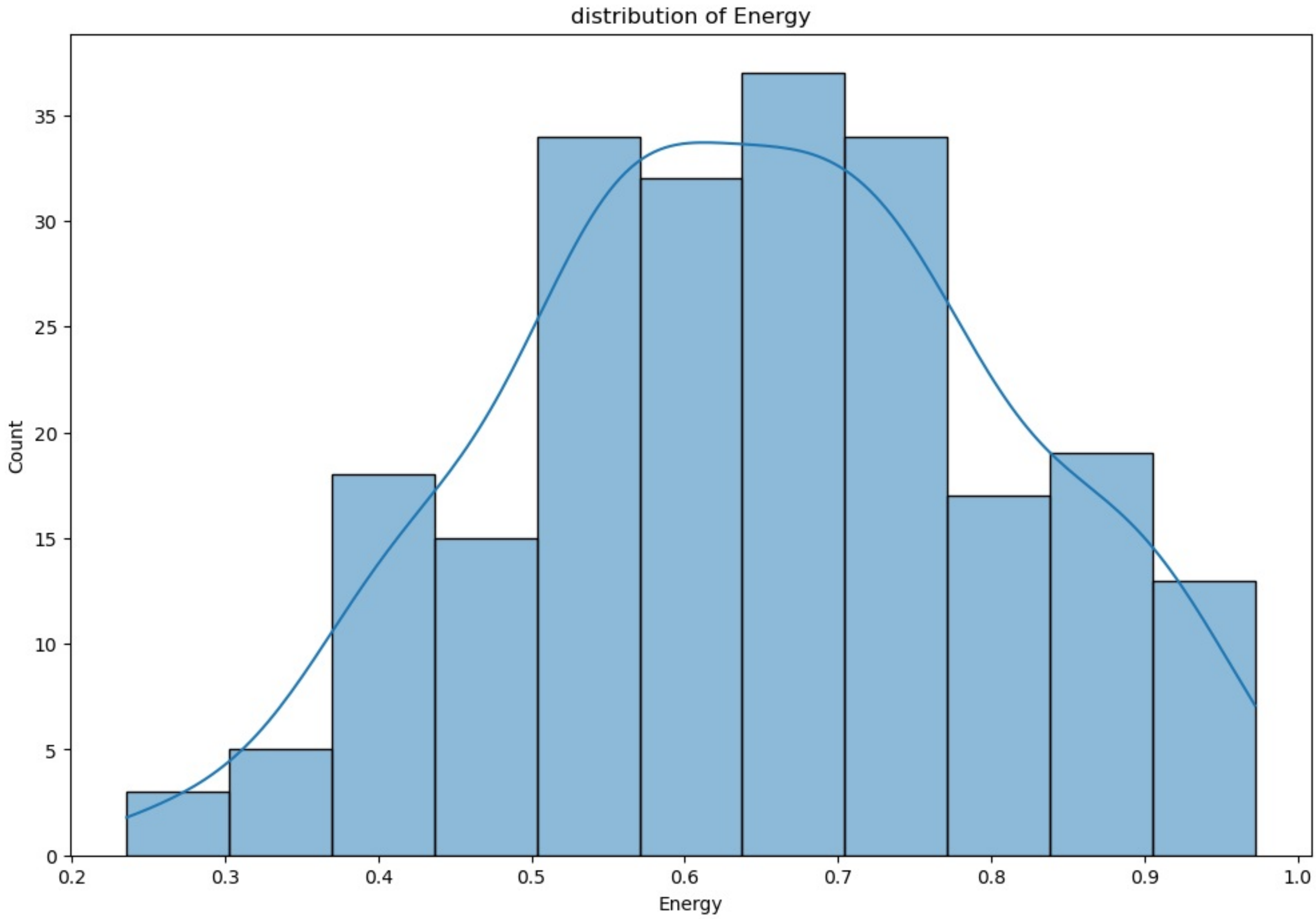
distribution of music features

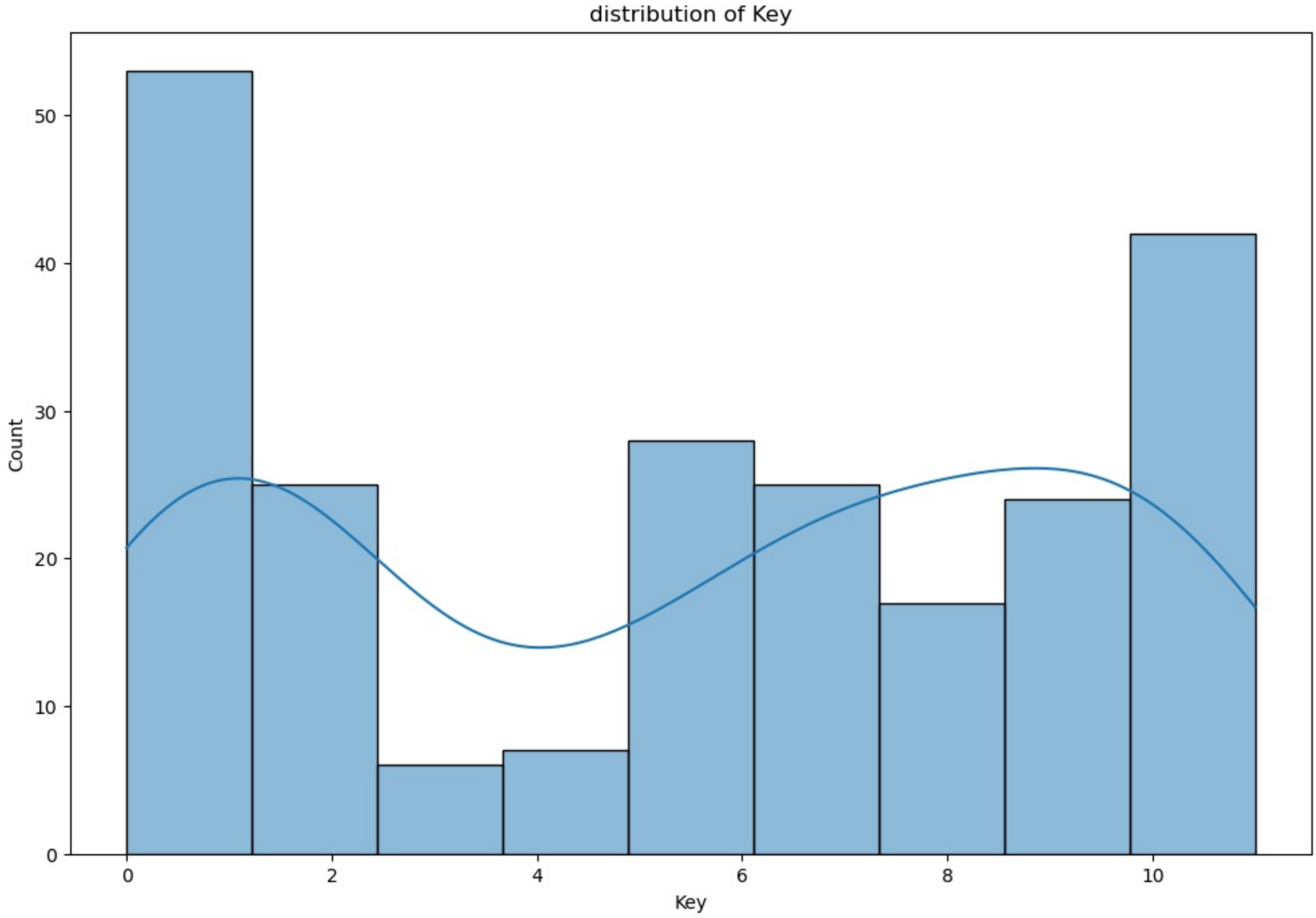
In [32]:
```
music_f
```

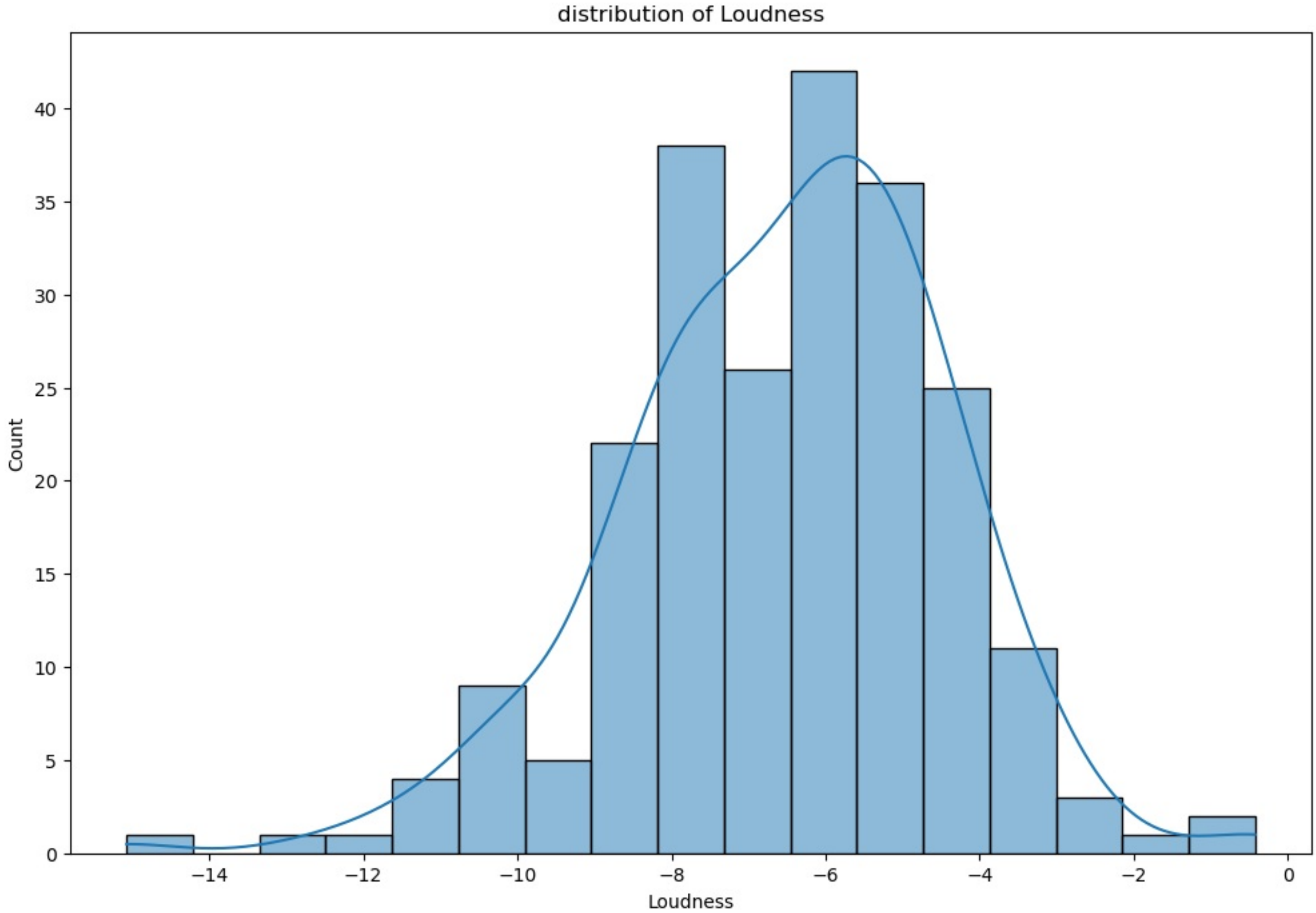
Out[32]:
```
['Danceability',
 'Energy',
 'Key',
 'Loudness',
 'Mode',
 'Speechiness',
 'Acousticness',
 'Instrumentalness',
 'Liveness',
 'Valence',
 'Tempo']
```

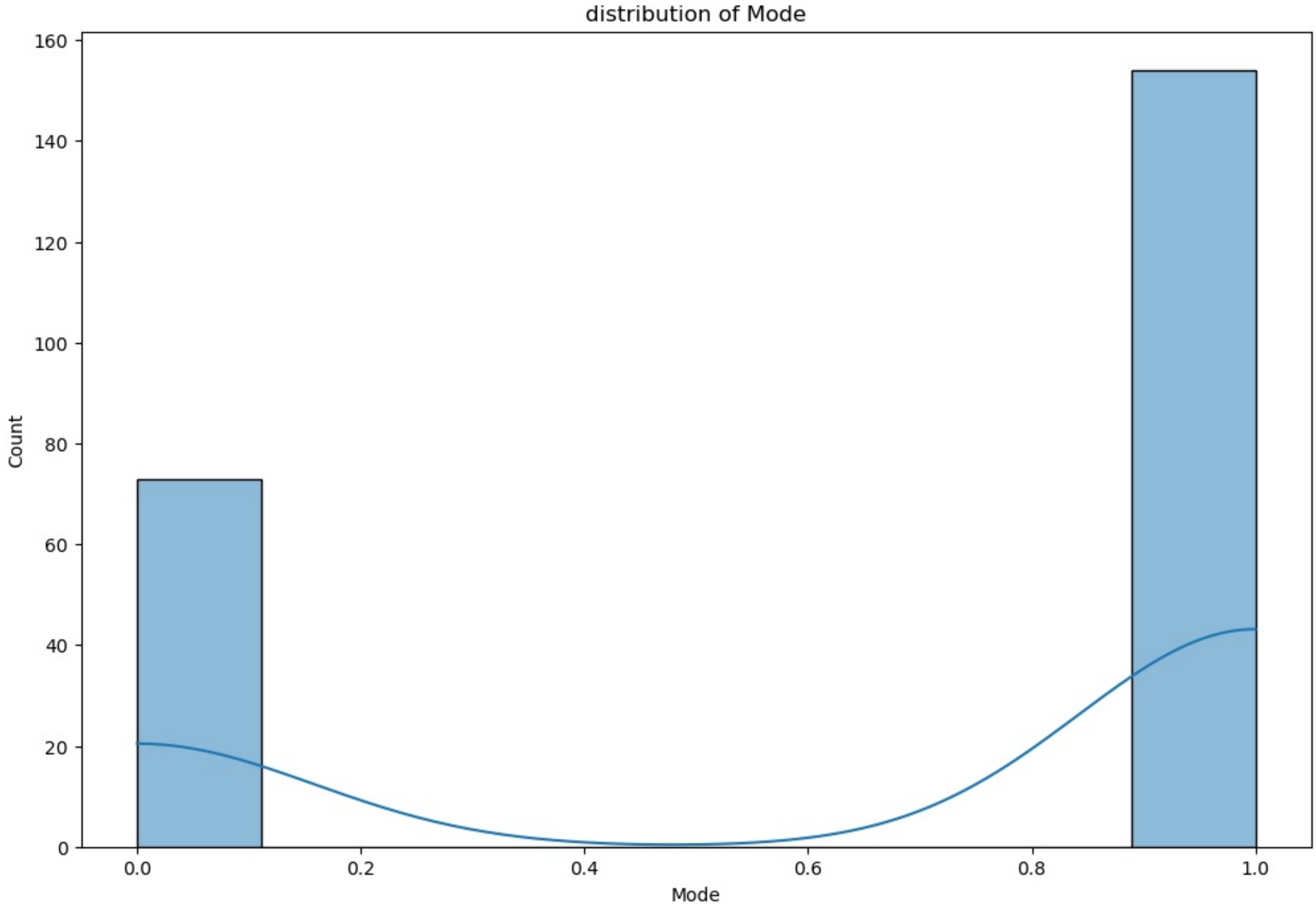
In [33]:
```
for i in music_f:
    plt.figure(figsize=(12,8))
    sns.histplot(df[i], kde=True)
    plt.title('distribution of {}'.format(i))
    plt.show()
```

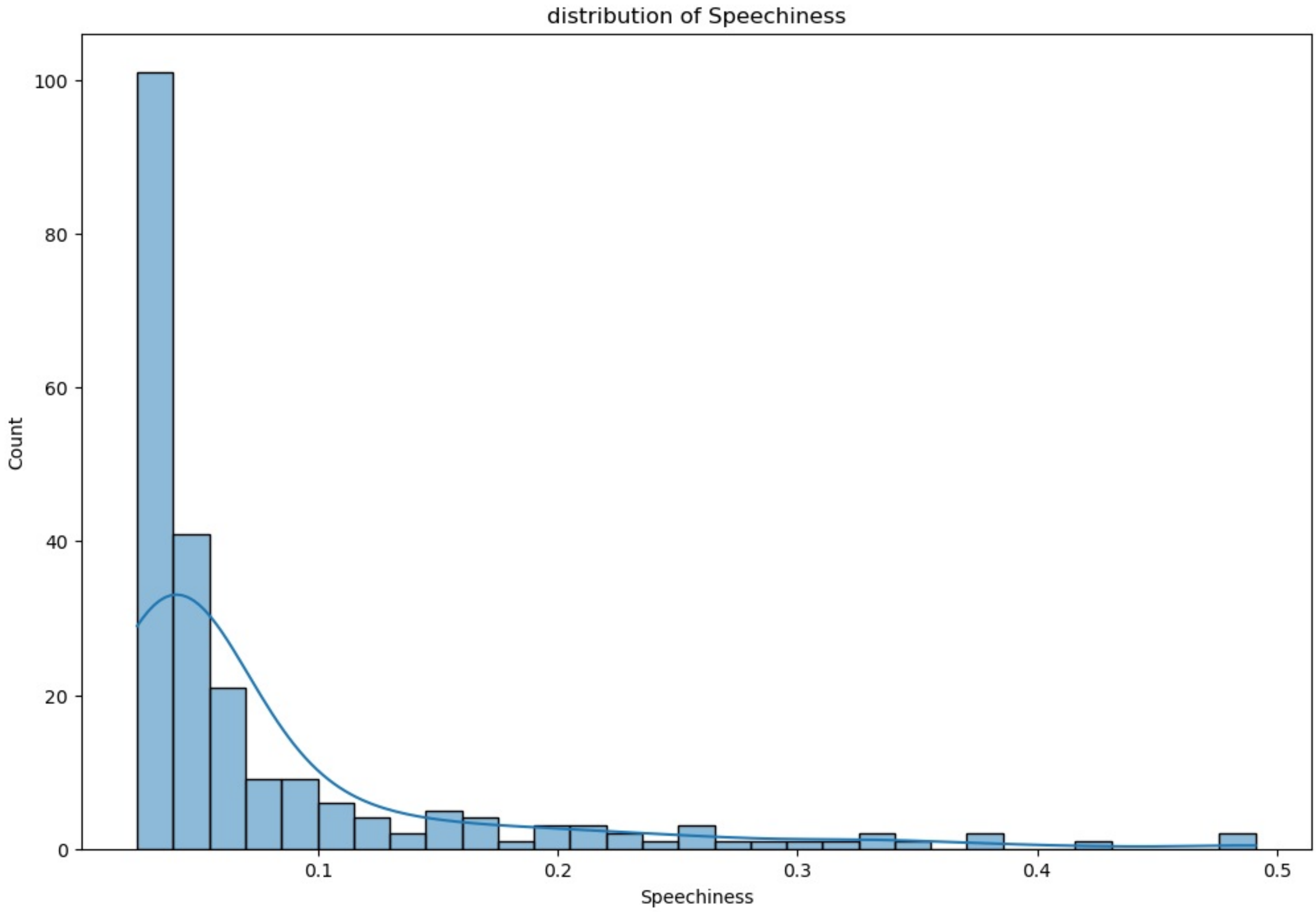

distribution of Danceability
Count
Danceability
0
10
20
30
40
0.3
0.4
0.5
0.6
0.7
0.8
0.9

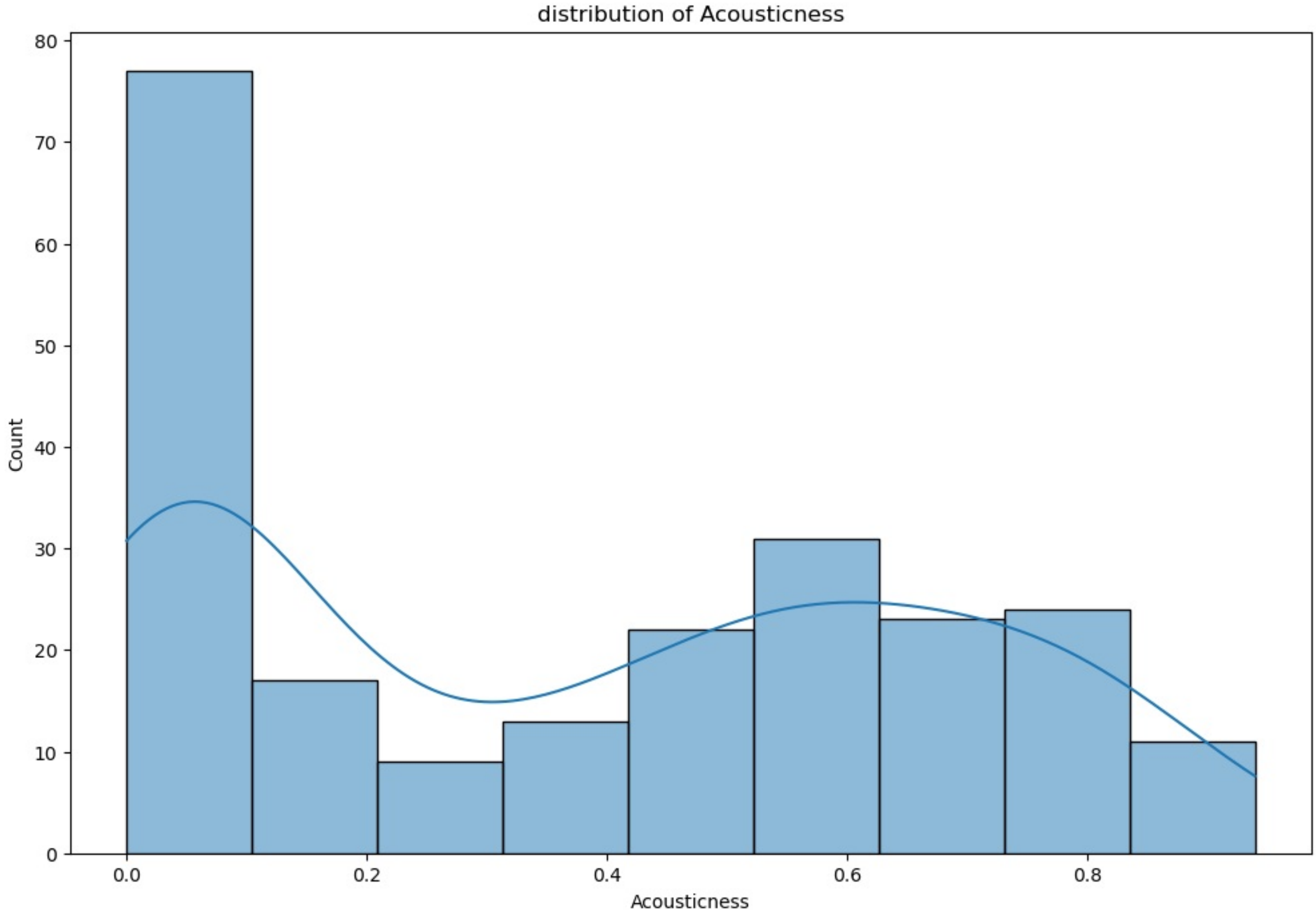

distribution of Energy
Count
Energy
0
5
10
15
20
25
30
35
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

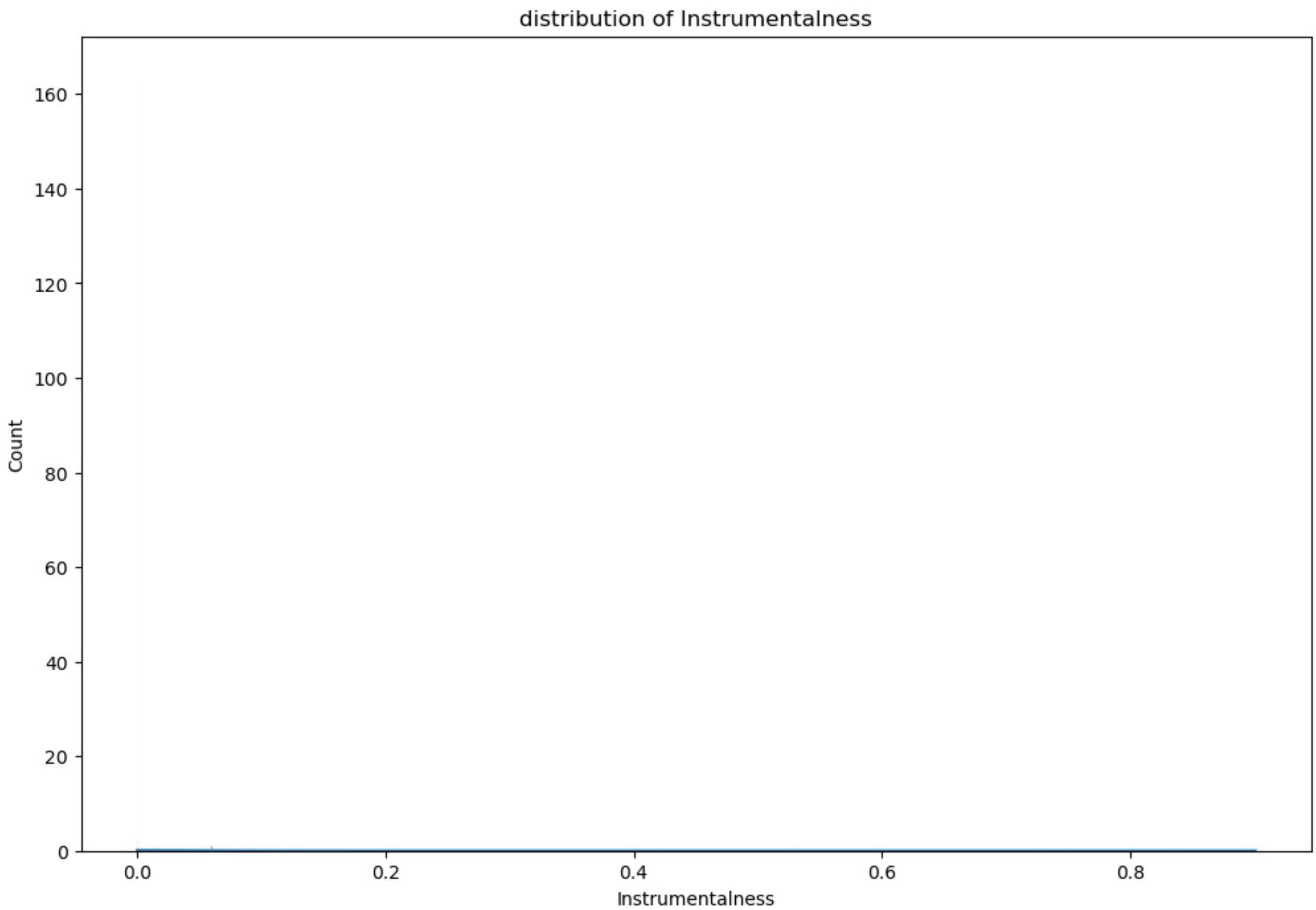

distribution of Key
Count
Key
0
10
20
30
40
50
0
2
4
6
8
10

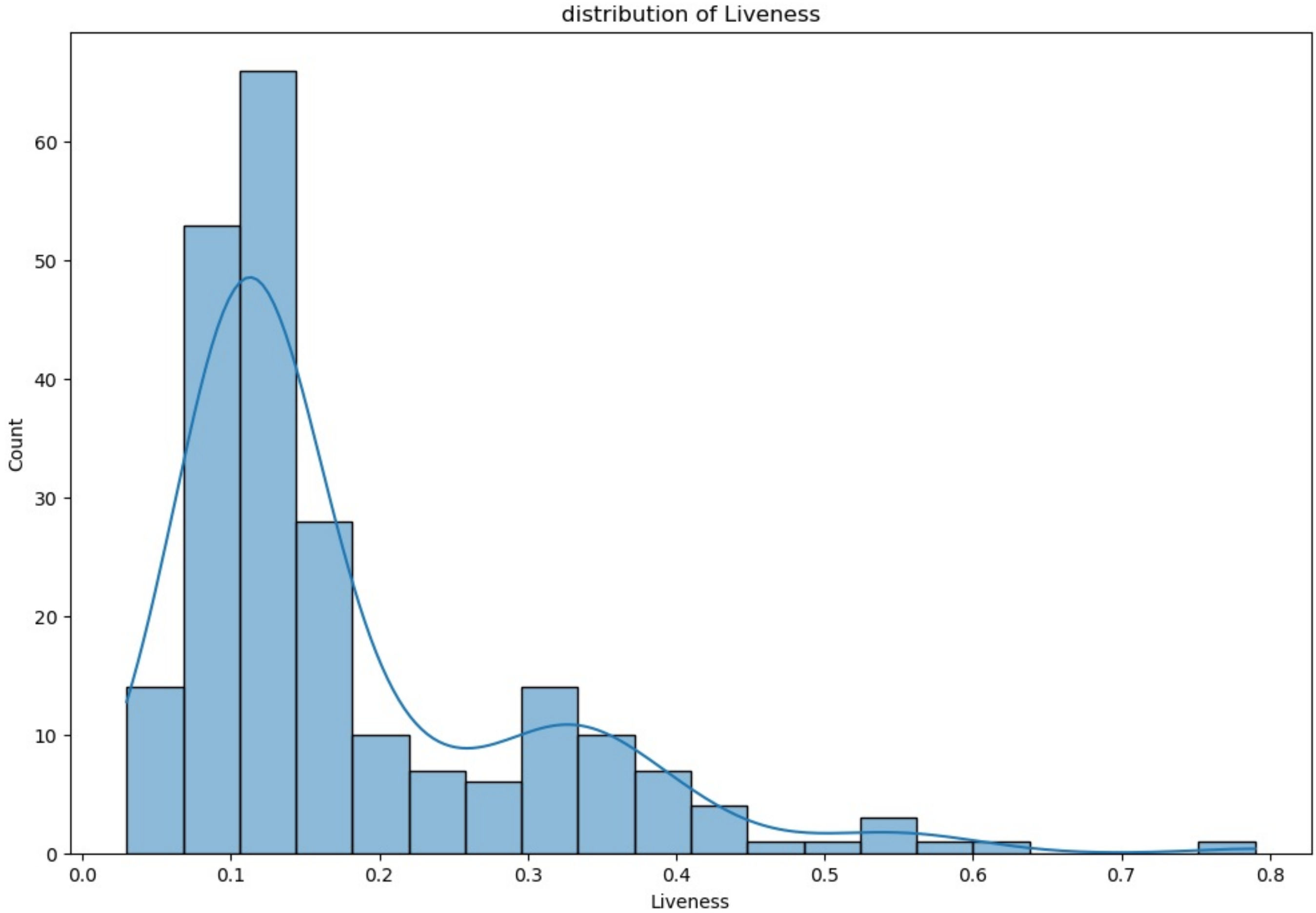

distribution of Loudness
Count
Loudness
0
5
10
15
20
25
30
35
40
−14
−12
−10
−8
−6
−4
−2
0

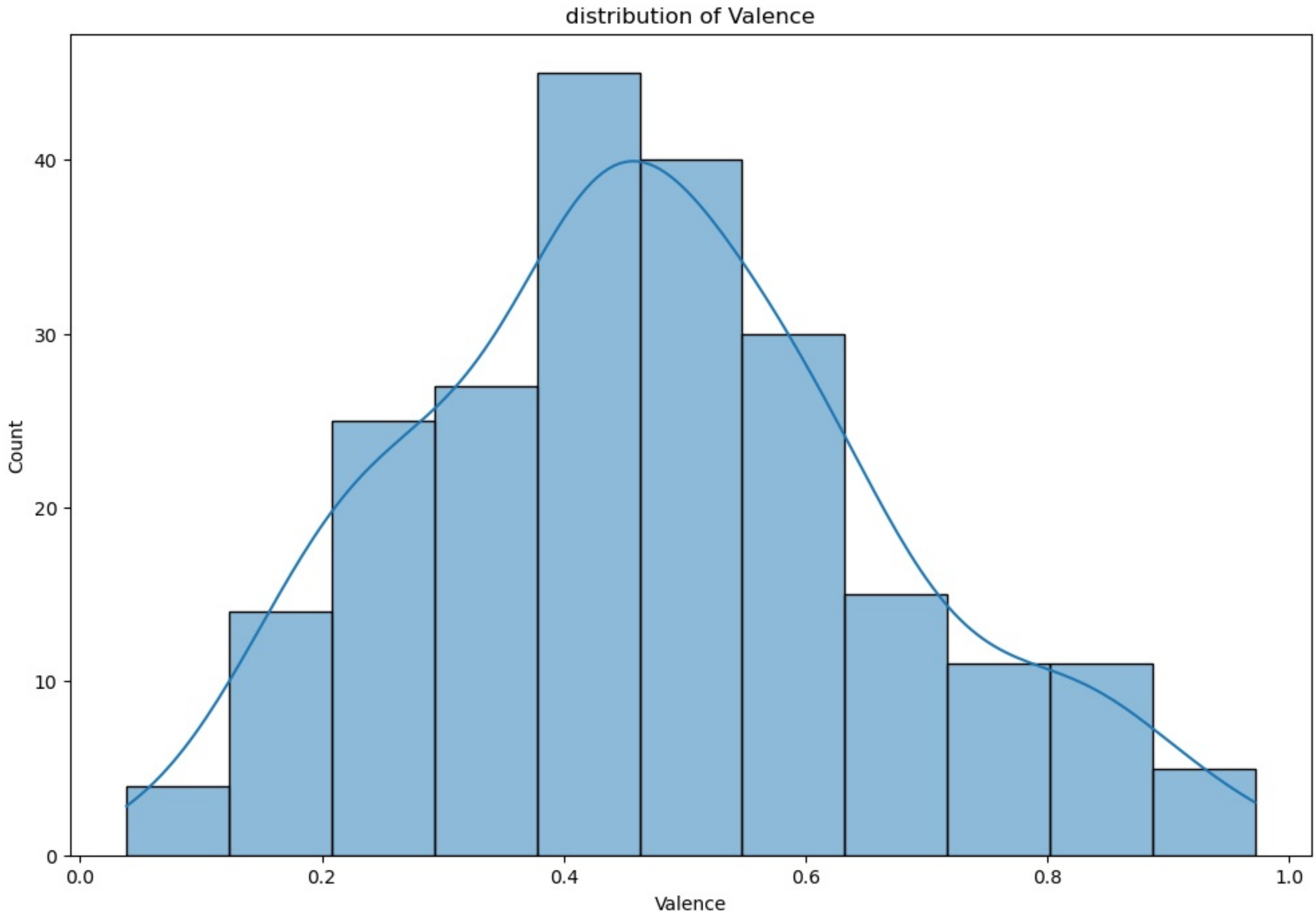

distribution of Mode
Count
Mode
0
20
40
60
80
100
120
140
160
0.0
0.2
0.4
0.6
0.8
1.0

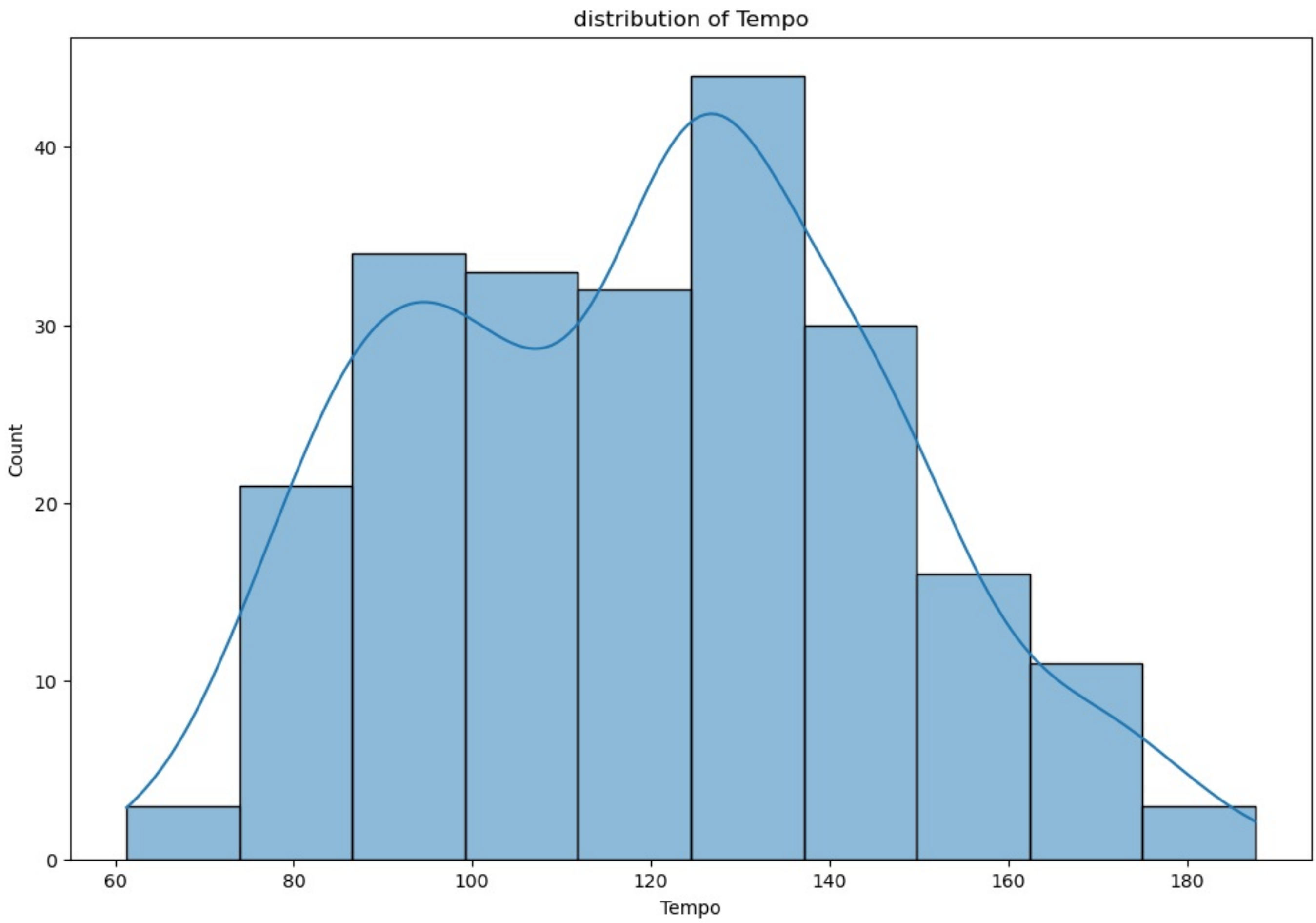

distribution of Speechiness
Count
Speechiness
0
20
40
60
80
100
0.1
0.2
0.3
0.4
0.5


distribution of Acousticness
80
70
60
50
40
30
20
10
0
Count
0.0
0.2
0.4
0.6
0.8
Acousticness


distribution of Instrumentalness
160
140
120
100
80
60
40
20
0
Count
0.0
0.2
0.4
0.6
0.8
Instrumentalness


distribution of Liveness
Count
Liveness
0
10
20
30
40
50
60
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


distribution of Valence
Count
Valence
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0



observation: 1.danceability, energy and valence has roughly bell shaped curve 2.loudness has near normal distribution centered around -6 3.acousticness is skewed towards lower values, same for liveness,speechiness this tells that: 1.bell shaped shows the balanced range of energy levels and good mix of danceability and positive emotions(valence) in tracks 2.skewness shows that most tracks are not acoustic, has liveness and speechiness

Feature Selection

With Correlation: loudness, danceability, energy,speechiness,livness

With Visualization: valence, tempo,acousticness

These features show significant realtionship with popularity, so can be used to train a music popularity model

Model Training

using these selected features train the model

In [34]:
```
#train test split
from sklearn.model_selection import train_test_split
```

In [35]:
```
feature_var=['Energy','Valence','Loudness','Danceability','Speechiness','Liveness','Tempo','Acousticness']
x=df[feature_var]
x
```

Out[35]:

	Energy	Valence	Loudness	Danceability	Speechiness	Liveness	Tempo	Acousticness
0	0.472	0.214	-7.001	0.898	0.0776	0.1410	101.061	0.0107
1	0.887	0.889	-2.760	0.936	0.0683	0.0582	127.003	0.0292
2	0.764	0.886	-5.241	0.882	0.2040	0.1190	140.113	0.3590
3	0.714	0.554	-4.617	0.681	0.0309	0.2320	129.976	0.0375
4	0.936	0.844	-6.294	0.788	0.3010	0.3110	151.019	0.0229
...	...	...	...	...	...	...	...	...
222	0.744	0.415	-5.817	0.565	0.0446	0.0853	134.068	0.4030
223	0.522	0.628	-5.857	0.626	0.0317	0.4100	118.001	0.6860
224	0.565	0.607	-7.954	0.484	0.0347	0.1050	82.653	0.4790
225	0.374	0.388	-9.849	0.602	0.0328	0.0840	101.855	0.9240
226	0.582	0.365	-5.180	0.296	0.0413	0.3190	168.400	0.4490

227 rows × 8 columns

In [36]:
```
y=df['Popularity']
y
```

Out[36]:
```
0      96
1      94
2      91
3      90
4      89
       ..
222    66
223    58
224    62
225    65
226    61
Name: Popularity, Length: 227, dtype: int64
```

In [37]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [38]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_train.shape)
```

```
(181, 8)
(181,)
(46, 8)
(181,)
```

transforming and fitting of train data & only transforming of test data on feature/independent variables(only)

For Scaling -> Standard Scaling

In [39]:
```
from sklearn.preprocessing import StandardScaler
```

In [40]:
```
scaler=StandardScaler()
```

In [41]:
```
x_train_scaled=scaler.fit_transform(x_train)
```

In [42]:
```
x_test_scaled=scaler.transform(x_test)
```

Trying this scaled data on different regression models and seeing model evaluation

linear regression

In [43]:
```
from sklearn.linear_model import LinearRegression
```

In [44]:
```
reg=LinearRegression()
```

In [45]:
```
reg.fit(x_train_scaled,y_train)
```

Out[45]:
```
▾ LinearRegression
LinearRegression()
```

In [46]:
```
y_predict=reg.predict(x_test_scaled)
```

In [47]:
```
print(y_predict[:5])
```

```
[73.80512835 67.31537775 71.69191033 62.79801137 73.13496811]
```

In [48]:
```
print(y_test[:5].tolist())
```

```
[87, 71, 85, 56, 69]
```

for model evaluation use evaluaiton metrics

In [49]:
```
from sklearn.metrics import mean_squared_error,r2_score
```

In [50]:
```
linear_reg_mean_sq=round(mean_squared_error(y_predict,y_test)*100,3)
```

In [51]:
```
linear_reg_mean_sq
```

Out[51]:
```
6933.888
```

In [52]:
```
linear_reg_r2=round(r2_score(y_predict,y_test)*100,3)
linear_reg_r2
```

Out[52]:
```
-165.636
```

decision tree

In [53]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [54]:
```
dec_tree=DecisionTreeRegressor(random_state=42)
```

In [55]:
```
dec_tree.fit(x_train_scaled,y_train)
```

Out[55]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

In [56]:
```
y_predict=dec_tree.predict(x_test_scaled)
```

In [57]:
```
print(y_predict[:5])
```

```
[75. 71. 75. 72. 84.]
```

In [58]:
```
print(y_test[:5].tolist())
```

```
[87, 71, 85, 56, 69]
```

In [59]:
```
#from sklearn.tree import plot_tree
#plt.figure(figsize=(12, 8))
#plot_tree(dec_tree, filled=True, feature_names=['x_train'])
#plt.show()
```

evaluating decison tree regressor

In [60]:
```
dec_tree_mean_sq=round(mean_squared_error(y_predict,y_test)*100,3)
dec_tree_mean_sq
```

Out[60]:
```
16282.609
```

In [61]:
```
dec_tree_r2=round(r2_score(y_predict,y_test)*100,3)
dec_tree_r2
```

Out[61]:
```
8.732
```

support vector machine(regressor)

In [62]:
```
from sklearn.svm import SVR
```

In [63]:
```
machine=SVR(kernel='poly')
```

In [64]:
```
machine.fit(x_train_scaled,y_train)
```

Out[64]:
```
▾ SVR
SVR(kernel='poly')
```

In [65]:
```
y_predict=machine.predict(x_test_scaled)
```

In [66]:
```
print(y_predict[:5])
```
```
[75.05426085 71.33622599 77.41365282 67.44008799 73.49151849]
```

In [67]:
```
print(y_test[:5].tolist())
```
```
[87, 71, 85, 56, 69]
```

model evaluation

In [68]:
```
svr_mean_sq=round(mean_squared_error(y_predict,y_test)*100,3)
svr_mean_sq
```
Out[68]:
```
6158.647
```

In [69]:
```
svr_r2=round(r2_score(y_predict,y_test)*100,3)
svr_r2
```
Out[69]:
```
-248.281
```

random forest

In [70]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [71]:
```
random_forest=RandomForestRegressor(n_estimators=100)
```

In [72]:
```
random_forest.fit(x_train_scaled,y_train)
```
Out[72]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [73]:
```
y_predict=random_forest.predict(x_test_scaled)
```

In [74]:
```
print(y_predict[:5])
```
```
[79.76   69.51   81.67   66.195 79.83 ]
```

In [75]:
```
print(y_test[:5].tolist())
```
```
[87, 71, 85, 56, 69]
```

model evaluation

In [76]:
```
rf_mean_sq=round(mean_squared_error(y_predict,y_test)*100,3)
rf_mean_sq
```
Out[76]:
```
5495.734
```

In [77]:
```
rf_r2=round(r2_score(y_predict,y_test)*100,3)
rf_r2
```
Out[77]:
```
7.172
```

In [78]:
```
eval_model_ss=pd.DataFrame({'linear reg':[linear_reg_mean_sq,linear_reg_r2],
                         'decision tree':[dec_tree_mean_sq,dec_tree_r2],
                        'svr':[svr_mean_sq,svr_r2],
                        'random forest':[rf_mean_sq,rf_r2]},
                       index=['mean square','r2_score'])
```

In [79]:
```
eval_model_ss
```
Out[79]:

	linear reg	decision tree	svr	random forest
mean square	6933.888	16282.609	6158.647	5495.734
r2_score	-165.636	8.732	-248.281	7.172

scaling method -> MinMax Scaler

In [82]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```
```
(181, 8)
(181,)
(46, 8)
(46,)
```

In [83]:
```
#from sklearn.preprocessing import MinMaxScaler
```

In [84]:
```
#scaler=MinMaxScaler(feature_range=(0,1))
```

In [85]:
```
#x_train_scaled=scaler.fit_transform(x_train)
```

In [86]:
```
#x_test_scaled=scaler.transform(x_test)
```

In [87]:
```
#x_train_scaled
```

using the same regression models and then evaluating them

Linear Regression

In [88]:
```
#reg.fit(x_train_scaled,y_train)
```

In [89]:
```
#y_predict=reg.predict(x_test_scaled)
```

In [90]:
```
#y_predict[:5]
```

In [91]:
```
#linear_reg_mean=round(mean_squared_error(y_predict,y_test)*100,3)
#linear_reg_mean
```

as u can see the computed values are same even alfter doing minmaxscaler, their is no need to continue and see the evaluation of the remaining 3 algorithms

In [92]:
```
eval_model_ss
```

Out[92]:

	linear reg	decision tree	svr	random forest
mean square	6933.888	16282.609	6158.647	5495.734
r2_score	-165.636	8.732	-248.281	7.172

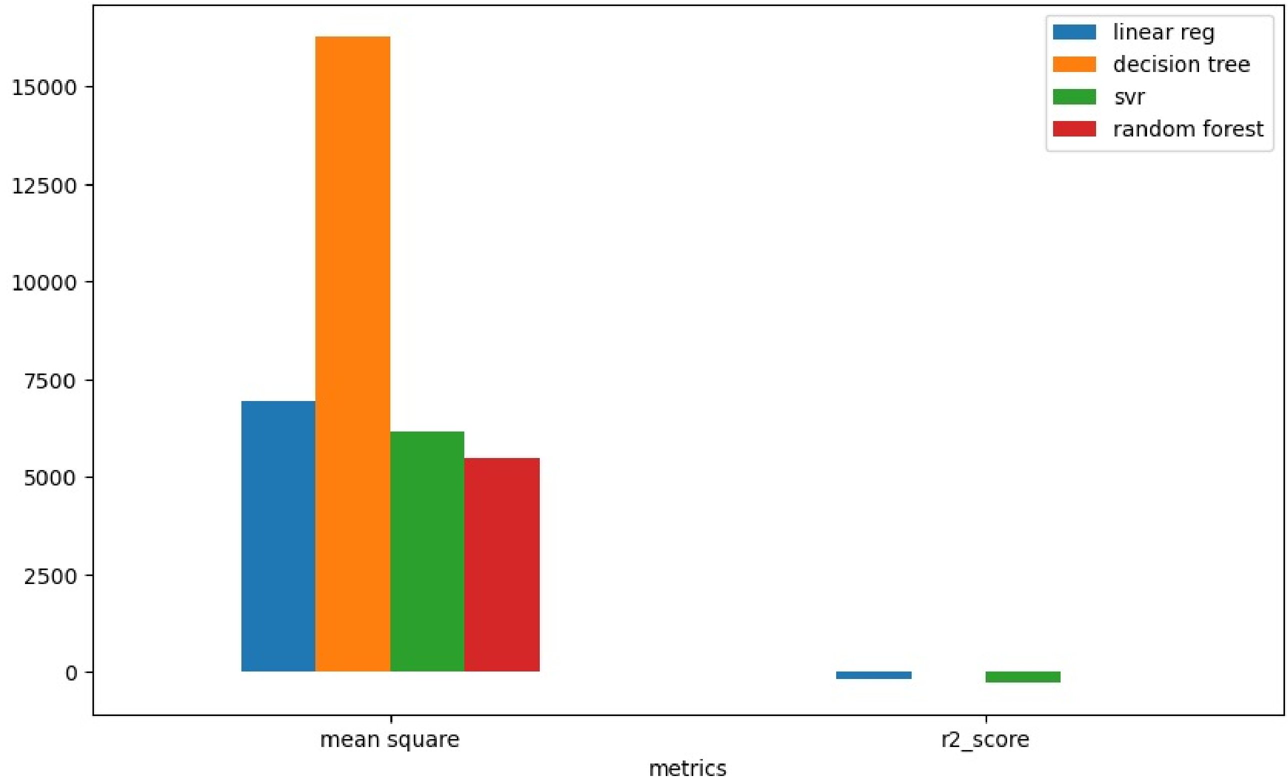
In [93]:
```
eval_model_ss.keys()
```

Out[93]:
```
Index(['linear reg', 'decision tree', 'svr', 'random forest'], dtype='object')
```

In [94]:
```
eval_model_ss.index
```

Out[94]:
```
Index(['mean square', 'r2_score'], dtype='object')
```

In [95]:
```
ax=eval_model_ss.plot(kind='bar',figsize=(10,6), legend=True)
ax.set_xlabel('metrics')
#ax.set_xticklabels(eval_model_ss.index,rotation=360)
plt.xticks(rotation=360)
plt.show()
```



From the above graph it is clear that Random Forest has smallest mean squared error, So we can use this model for the popularity prediction of an audio

To enhance the working of Random Forest we can use hyperparameter tuning and can again fit the model for prediction

Hyperparameter tuning

Grid Search

In [96]:
```
from sklearn.model_selection import GridSearchCV
```

In [97]:
```
param_grid={
    'n_estimators':[100,200,300],
    'max_depth':[10,20,30,None],
    'max_features':['sqrt','log2'],
    'min_samples_split':[2,5,10],
    'min_samples_leaf':[1,2,4]
}
```

In [98]:
```
grid_search=GridSearchCV(RandomForestRegressor(random_state=42),param_grid,refit=True,verbose=2,cv=5)
#cv=5->The dataset will be split into 5 folds, and the model will be trained and evaluated 5 times.
```

In [99]:
```
grid_search.fit(x_train_scaled,y_train)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time=
0.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=300; total time=
```

```
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=300; total time=
0.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=300; total time=
0.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time
=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time
=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time
=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time
=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=
0.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=
0.6s
```

```
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=
0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time
=   0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time
=   0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time
=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time
=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=
```

```
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=
0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=
0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=
0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=
0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=
0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=
0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total time=
0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total time=
0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total time=
0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total time=
0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time
=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total time
=   0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total time
=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time
=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time
=   0.7s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=
0.2s
```

```
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=2, n_estimators=300; total tim
e=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total tim
e=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total tim
e=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total tim
e=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total tim
e=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total tim
e=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total tim
e=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total tim
e=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total tim
e=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total tim
e=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total tim
e=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total tim
e=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total tim
e=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total tim
e=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total tim
e=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=5, n_estimators=300; total tim
e=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total ti
me=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total ti
me=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total ti
me=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total ti
me=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total ti
me=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total ti
me=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total ti
me=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total ti
me=   0.4s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total ti
me=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=200; total ti
me=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total ti
me=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total ti
me=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total ti
me=   0.8s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total ti
me=   0.7s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=4, min_samples_split=10, n_estimators=300; total ti
me=   0.8s
```

Out[99]:

▸ **GridSearchCV**

▸ **estimator: RandomForestRegressor**

▸ RandomForestRegressor

In [100…

```
best_para=grid_search.best_params_
best_para
```

Out[100…

```
{'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 100}
```

In [101…

```
best_model=grid_search.best_estimator_
best_model
```

Out[101…

```
RandomForestRegressor
RandomForestRegressor(max_depth=10, max_features='sqrt', random_state=42)
```

In [102…

```
y_predict=best_model.predict(x_test_scaled)
```

In [103…

```
mean_squared_g=round(mean_squared_error(y_predict,y_test)*100,3)
mean_squared_g
```

Out[103… 5390.301

In [104…

```
r2_score_gs=round(r2_score(y_predict,y_test)*100,3)
r2_score_gs
```

Out[104… -30.664

Randomized Search

In [105…

```
from sklearn.model_selection import RandomizedSearchCV
```

In [106…

```
param_grid
```

Out[106…

```
{'n_estimators': [100, 200, 300],
 'max_depth': [10, 20, 30, None],
 'max_features': ['sqrt', 'log2'],
 'min_samples_split': [2, 5, 10],
 'min_samples_leaf': [1, 2, 4]}
```

In [107…

```
random_search=RandomizedSearchCV(RandomForestRegressor(n_estimators=42),
                                 param_distributions=param_grid,n_iter=10,cv=5)
```

In [108…

```
random_search.fit(x_train_scaled,y_train)
```

Out[108…

```
RandomizedSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

In [109…

```
y_predict=random_search.predict(x_test_scaled)
```

In [110…

```
best_para=random_search.best_params_
best_para
```

Out[110…

```
{'n_estimators': 300,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 20}
```

In [111…

```
best_model=random_search.best_estimator_
best_model
```

Out[111…

```
RandomForestRegressor
RandomForestRegressor(max_depth=20, max_features='sqrt', min_samples_leaf=2,
                      n_estimators=300)
```

In [112…

```
mean_squared_rs=round(mean_squared_error(y_predict,y_test)*100,3)
mean_squared_rs
```

Out[112… 4472.217

In [113…

```
r2_score_rs=round(r2_score(y_predict,y_test)*100,3)
r2_score_rs
```

Out[113… -46.119

In [114…

```
hyperparam=pd.DataFrame({'gridSearch':[mean_squared_g,r2_score_gs],
                        'randomSearch':[mean_squared_rs,r2_score_rs]},
                       index=['mean square error','r2 score'])
```

In [115…

```
hyperparam
```

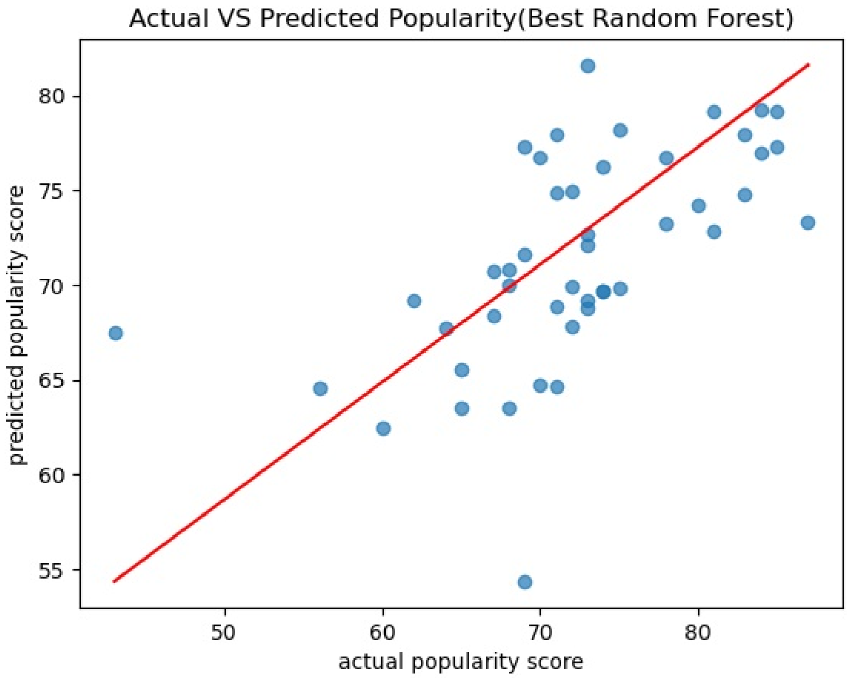
Out[115...

	gridSearch	randomSearch
mean square error	5390.301	4472.217
r2 score	-30.664	-46.119

from above it is clear that after using Randomized Search method for hyperparameter tuning the results aree much better than Grid Search. So we would be fitting the model on the basis of Randomized Search of Hyperparameter Tuning

In [116...

```
plt.scatter(y_test,y_predict,alpha=0.7)
plt.plot([min(y_test),max(y_test)],[min(y_predict),max(y_predict)],color='red')
plt.xlabel('actual popularity score')
plt.ylabel('predicted popularity score')
plt.title('Actual VS Predicted Popularity(Best Random Forest)')
plt.show()
```



The red line represents perfect predictions, where the predicted popularity would exactly match the actual popularity. Most of the points are clustered around this line, which indicates that the model is making reasonably accurate predictions. However, there are some deviations, particularly at lower popularity values, which suggest areas where the model's predictions are less precise.

In [ ]: