Spotify popularity prediction Analysis & Modelling

November 22, 2023

[1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```python
df = pd.read_csv('/Users/Home/OneDrive/Desktop/Python/Spotify_dataset.csv')
```

[3]:
```python
df.head()
```

[3]:
```
   Unnamed: 0                track_id                 artists  \
0           0  5SuOikwiRyPMVoIQDJUgSV              Gen Hoshino
1           1  4qPNDBW1i3p13qLCt0Ki3A             Ben Woodward
2           2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3           3  6lfxq3CG4xtTiEg7opyCyx             Kina Grannis
4           4  5vjLSffimiIP26QG5WcN2K        Chord Overstreet

                                          album_name  \
0                                             Comedy
1                                   Ghost (Acoustic)
2                                     To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                            Hold On

                   track_name  popularity  duration_ms  explicit  \
0                      Comedy          73       230666     False
1            Ghost - Acoustic          55       149610     False
2              To Begin Again          57       210826     False
3  Can't Help Falling In Love          71       201933     False
4                     Hold On          82       198853     False

   danceability  energy  …  loudness  mode  speechiness  acousticness  \
0         0.676  0.4610  …    -6.746     0       0.1430        0.0322
1         0.420  0.1660  …   -17.235     1       0.0763        0.9240
```

```
2          0.438  0.3590  …    -9.734     1       0.0557         0.2100
3          0.266  0.0596  …   -18.515     1       0.0363         0.9050
4          0.618  0.4430  …    -9.681     1       0.0526         0.4690

   instrumentalness  liveness  valence    tempo  time_signature  track_genre
0          0.000001    0.3580    0.715   87.917               4     acoustic
1          0.000006    0.1010    0.267   77.489               4     acoustic
2          0.000000    0.1170    0.120   76.332               4     acoustic
3          0.000071    0.1320    0.143  181.740               3     acoustic
4          0.000000    0.0829    0.167  119.949               4     acoustic

[5 rows x 21 columns]
```

[4]:
```python
df.shape
```

[4]:
```
(114000, 21)
```

[5]:
```python
df.drop(df.columns[0], axis=1, inplace=True)
df.shape
```

[5]:
```
(114000, 20)
```

[6]:
```python
df.describe()
```

[6]:
```
          popularity   duration_ms   danceability         energy  \
count  114000.000000  1.140000e+05  114000.000000  114000.000000
mean       33.238535  2.280292e+05       0.566800       0.641383
std        22.305078  1.072977e+05       0.173542       0.251529
min         0.000000  0.000000e+00       0.000000       0.000000
25%        17.000000  1.740660e+05       0.456000       0.472000
50%        35.000000  2.129060e+05       0.580000       0.685000
75%        50.000000  2.615060e+05       0.695000       0.854000
max       100.000000  5.237295e+06       0.985000       1.000000

                 key       loudness           mode    speechiness  \
count  114000.000000  114000.000000  114000.000000  114000.000000
mean        5.309140      -8.258960       0.637553       0.084652
std         3.559987       5.029337       0.480709       0.105732
min         0.000000     -49.531000       0.000000       0.000000
25%         2.000000     -10.013000       0.000000       0.035900
50%         5.000000      -7.004000       1.000000       0.048900
75%         8.000000      -5.003000       1.000000       0.084500
max        11.000000       4.532000       1.000000       0.965000

        acousticness  instrumentalness       liveness        valence  \
count  114000.000000     114000.000000  114000.000000  114000.000000
mean        0.314910          0.156050       0.213553       0.474068
```

```
std          0.332523          0.309555        0.190378        0.259261
min          0.000000          0.000000        0.000000        0.000000
25%          0.016900          0.000000        0.098000        0.260000
50%          0.169000          0.000042        0.132000        0.464000
75%          0.598000          0.049000        0.273000        0.683000
max          0.996000          1.000000        1.000000        0.995000

               tempo  time_signature
count  114000.000000   114000.000000
mean      122.147837        3.904035
std        29.978197        0.432621
min         0.000000        0.000000
25%        99.218750        4.000000
50%       122.017000        4.000000
75%       140.071000        4.000000
max       243.372000        5.000000
```

[7]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 20 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   track_id          114000 non-null  object
 1   artists           113999 non-null  object
 2   album_name        113999 non-null  object
 3   track_name        113999 non-null  object
 4   popularity        114000 non-null  int64
 5   duration_ms       114000 non-null  int64
 6   explicit          114000 non-null  bool
 7   danceability      114000 non-null  float64
 8   energy            114000 non-null  float64
 9   key               114000 non-null  int64
 10  loudness          114000 non-null  float64
 11  mode              114000 non-null  int64
 12  speechiness       114000 non-null  float64
 13  acousticness      114000 non-null  float64
 14  instrumentalness  114000 non-null  float64
 15  liveness          114000 non-null  float64
 16  valence           114000 non-null  float64
 17  tempo             114000 non-null  float64
 18  time_signature    114000 non-null  int64
 19  track_genre       114000 non-null  object
dtypes: bool(1), float64(9), int64(5), object(5)
memory usage: 16.6+ MB
```

1 Data Analysis (EDA)

In EDA, we always compare with the dependent variable, here Popularity. So we see the relationships of different features with this variable.

- Missing Values
- Duplicate values
- All the numerical variables
- Distribution of the numerical variables: Check the skewness of the features.
- Categorical variables
- Cardinality of Categorical variables
- Outliers
- Relationship between independent and dependent features

[8]:
```python
## Missing values
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['total', 'Percent'])
missing_data.head(20)
```

[8]:

	total	Percent
album_name	1	0.000009
track_name	1	0.000009
artists	1	0.000009
track_id	0	0.000000
speechiness	0	0.000000
time_signature	0	0.000000
tempo	0	0.000000
valence	0	0.000000
liveness	0	0.000000
instrumentalness	0	0.000000
acousticness	0	0.000000
loudness	0	0.000000
mode	0	0.000000
key	0	0.000000
energy	0	0.000000
danceability	0	0.000000
explicit	0	0.000000
duration_ms	0	0.000000
popularity	0	0.000000
track_genre	0	0.000000

[9]:
```python
##Handling missing data
df = df.drop((missing_data[missing_data['total'] > 1]).index,1)
df.isnull().sum().max() #fast checking that there is no missing data missing..
```

[9]: 1

[10]:
```
df = df.dropna()
```

[11]:
```
df.isnull().sum().max()
```

[11]:
```
0
```

[12]:
```
## Dropping duplicates
df = df.drop_duplicates()
```

[13]:
```
df.shape
```

[13]:
```
(113549, 20)
```

[14]:
```
## Saving the file
df.to_csv("Spotify popularity prediction Analysis & Modelling", index=False)
```

[15]:
```
df['popularity'].describe()
```

[15]:
```
count    113549.000000
mean         33.324433
std          22.283855
min           0.000000
25%          17.000000
50%          35.000000
75%          50.000000
max         100.000000
Name: popularity, dtype: float64
```

2 Numerical Features

[16]:
```
feature_numerical=[feature for feature in df.columns if df[feature].dtype!='O']
print('Number of numerical columns=', len(feature_numerical))
df[feature_numerical].head()
```

```
Number of numerical columns= 15
```

[16]:
```
   popularity  duration_ms  explicit  danceability  energy  key  loudness  \
0          73       230666     False         0.676  0.4610    1    -6.746
1          55       149610     False         0.420  0.1660    1   -17.235
2          57       210826     False         0.438  0.3590    0    -9.734
3          71       201933     False         0.266  0.0596    0   -18.515
4          82       198853     False         0.618  0.4430    2    -9.681

   mode  speechiness  acousticness  instrumentalness  liveness  valence  \
0     0       0.1430        0.0322          0.000001    0.3580    0.715
1     1       0.0763        0.9240          0.000006    0.1010    0.267
2     1       0.0557        0.2100          0.000000    0.1170    0.120
3     1       0.0363        0.9050          0.000071    0.1320    0.143
```

```
4     1       0.0526        0.4690           0.000000     0.0829     0.167

      tempo  time_signature
0    87.917               4
1    77.489               4
2    76.332               4
3   181.740               3
4   119.949               4
```

[17]:
```
## Selecting out the discrete features among the numerical features and finding␣
↪their relationship with popularity
feature_discrete_numerical=[feature for feature in feature_numerical if␣
↪df[feature].nunique()<50]
feature_discrete_numerical
```

[17]:
```
['explicit', 'key', 'mode', 'time_signature']
```

[18]:
```
##Skewness and kurtosis
print('Skewness: %f' % df['popularity'].skew())
print('Kurtosis: %f' % df['popularity'].kurt())
```

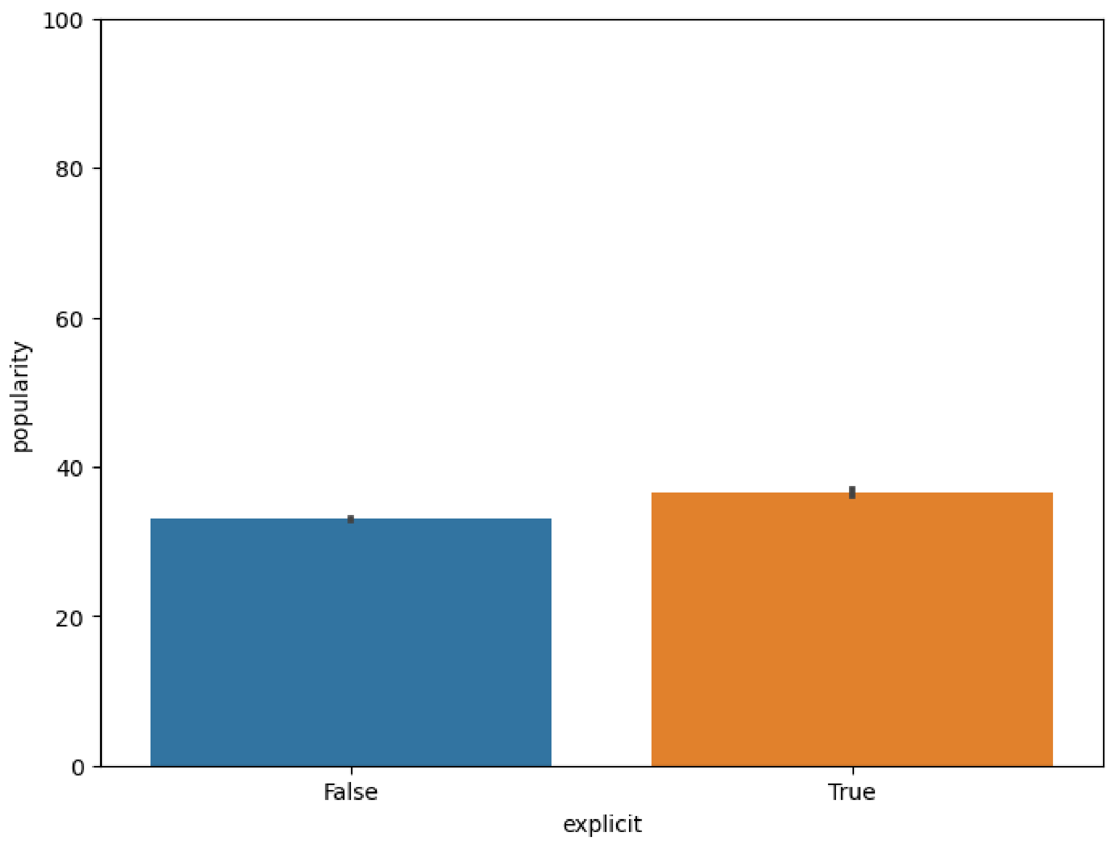
```
Skewness: 0.042229
Kurtosis: -0.924031
```

[19]:
```
##Standardizing data
popularity_scaled = StandardScaler().fit_transform(df['popularity'][:,np.
↪newaxis]);
low_range = popularity_scaled[popularity_scaled[:,0].argsort()][:10]
high_range= popularity_scaled[popularity_scaled[:,0].argsort()][-10:]
print('outer range (low) of the distribution:')
print(low_range)
print('\nouter range (high) of the distribution:')
print(high_range)
```

```
outer range (low) of the distribution:
[[-1.49545847]
 [-1.49545847]
 [-1.49545847]
 [-1.49545847]
 [-1.49545847]
 [-1.49545847]
 [-1.49545847]
 [-1.49545847]
 [-1.49545847]
 [-1.49545847]]

outer range (high) of the distribution:
[[2.90236374]
```

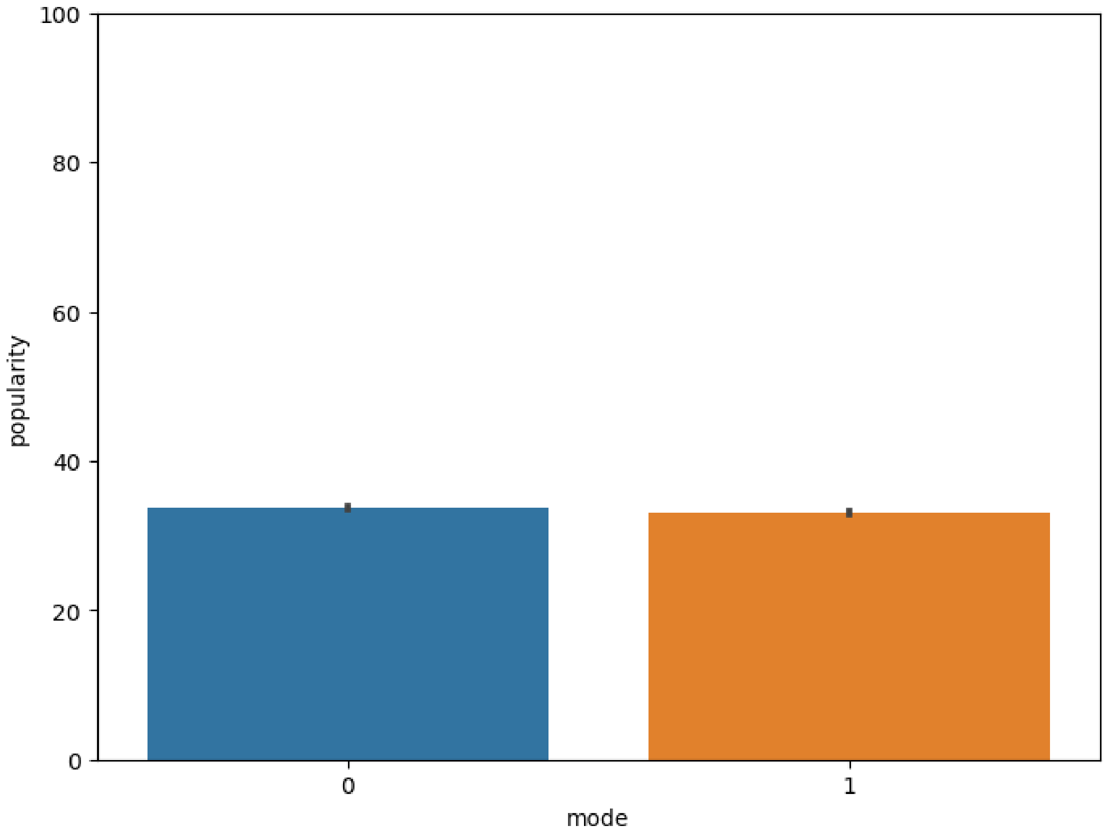
```
[2.90236374]
[2.90236374]
[2.90236374]
[2.90236374]
[2.90236374]
[2.90236374]
[2.94723948]
[2.99211522]
[2.99211522]]
```

[20]:
```
var = 'explicit'
data = pd.concat([df['popularity'], df[var]], axis=1)
fig, ax = plt.subplots(figsize=(8,6))
fig = sns.barplot(x=var, y='popularity', data=data)
fig.axis(ymin=0,ymax=100);
```

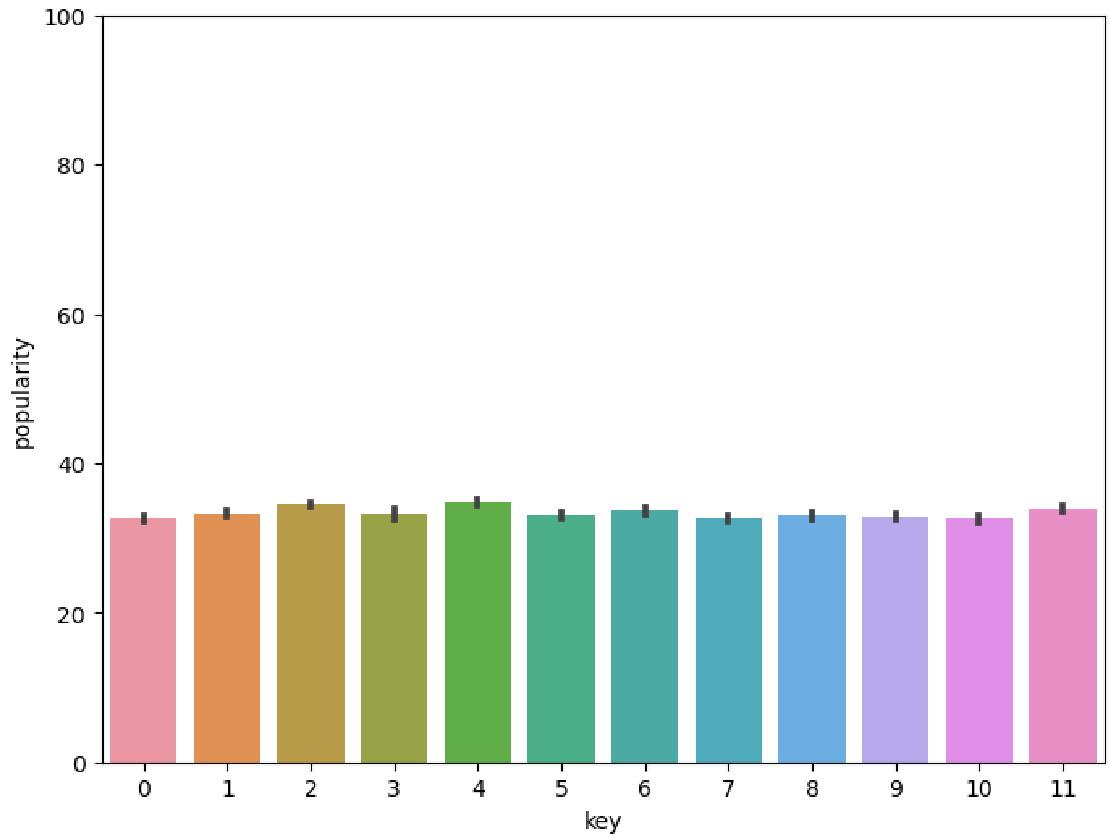


[21]:
```
var = 'mode'
data = pd.concat([df['popularity'], df[var]], axis=1)
fig, ax = plt.subplots(figsize=(8,6))
fig = sns.barplot(x=var, y='popularity', data=data)
```

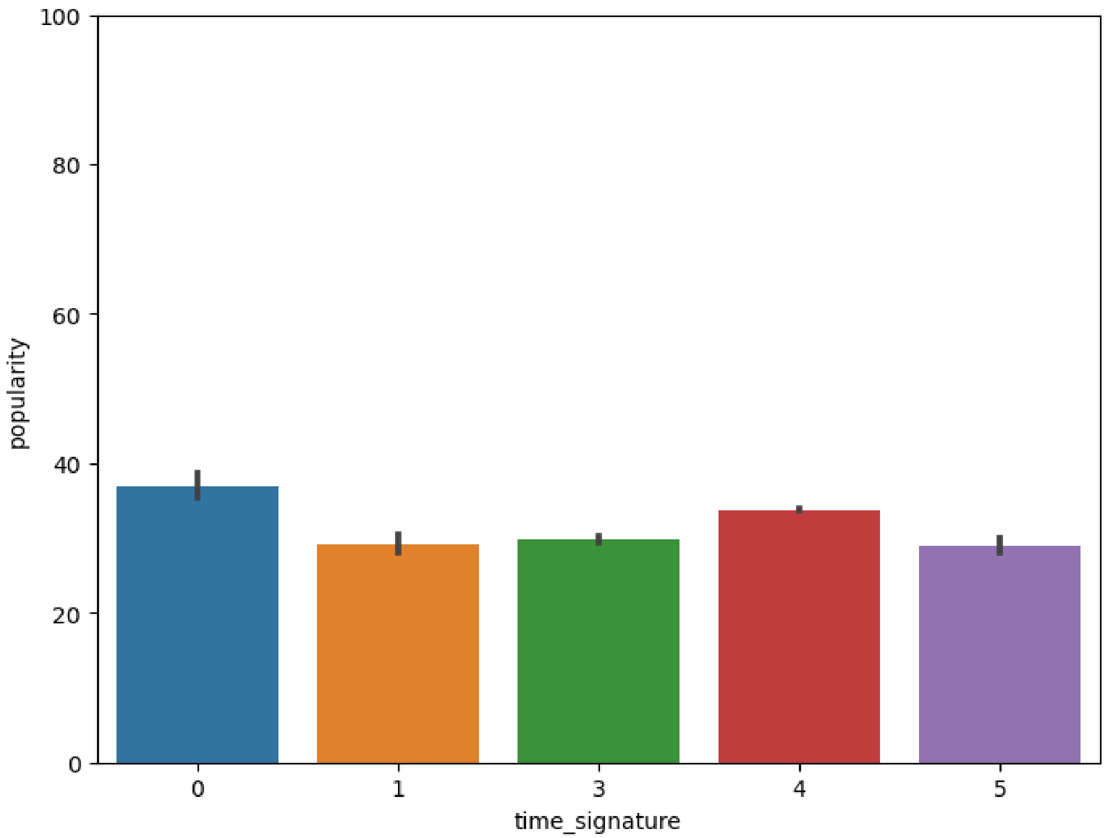
```
fig.axis(ymin=0,ymax=100);
```



[22]:
```
var = 'key'
data = pd.concat([df['popularity'], df[var]], axis=1)
fig, ax = plt.subplots(figsize=(8,6))
fig = sns.barplot(x=var, y='popularity', data=data)
fig.axis(ymin=0,ymax=100);
```



[23]:
```
var = 'time_signature'
data = pd.concat([df['popularity'], df[var]], axis=1)
fig, ax = plt.subplots(figsize=(8,6))
fig = sns.barplot(x=var, y='popularity', data=data)
fig.axis(ymin=0,ymax=100);
```



Observation:

1. We see that songs which contain explicit lyrics are more popular in comparision with sons that do not contain such lyrics.
2. Songs in all the keys are almost equally popular.
3. The tracks in both the modes are equally popular, the major as well as the minor.
4. The time signature (meter) is a notational convention to specify how many beats are in each bar.From the current looks, tracks having time_signature 0 and 4 are more popular than other.

Selecting the continuous features among the numerical features and finding their distribution

[24]:
```
features_continuous_numerical = [features for features in feature_numerical if␣
↪features not in feature_discrete_numerical]
```

[25]:
```
features_continuous_numerical
```

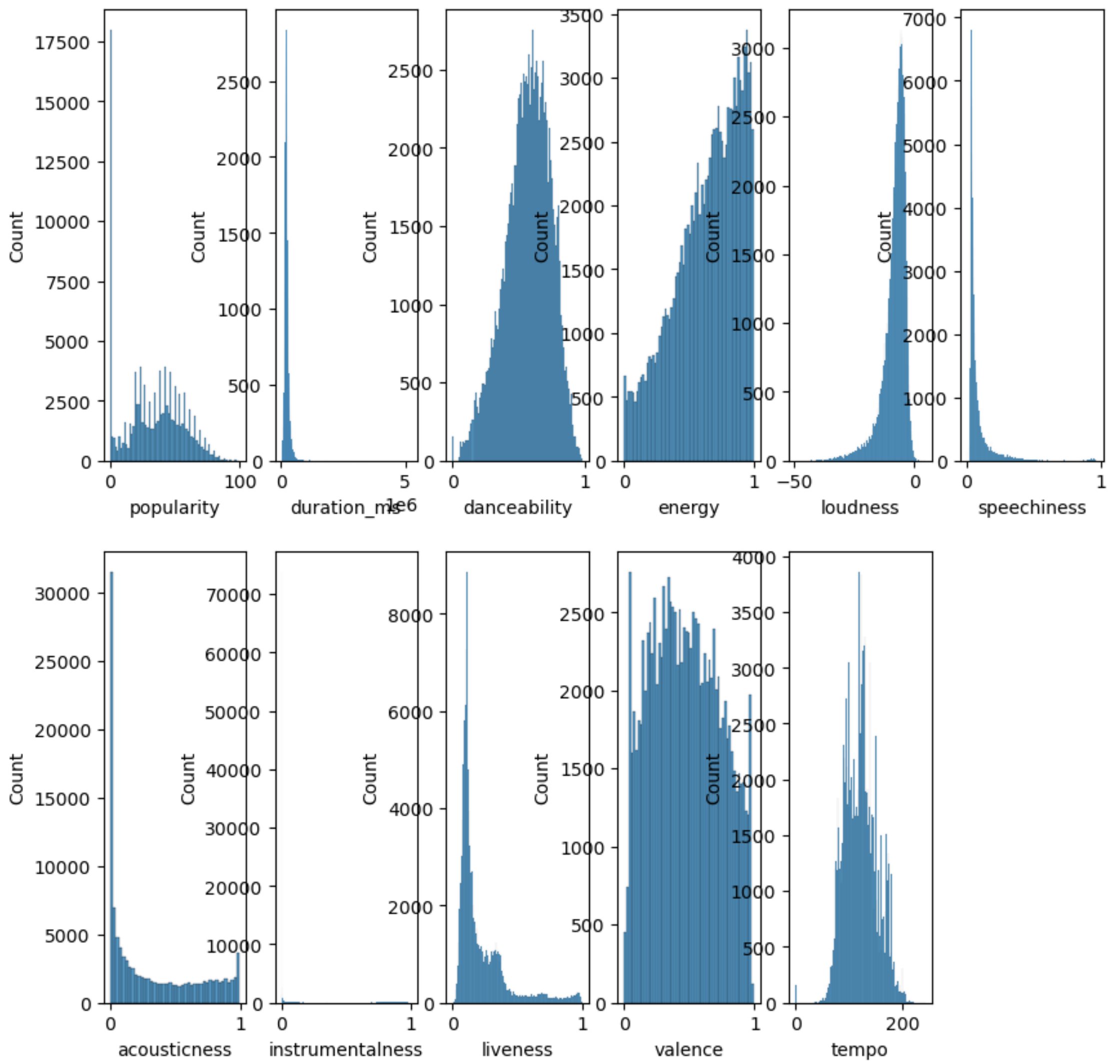
[25]:
```
['popularity',
 'duration_ms',
 'danceability',
 'energy',
 'loudness',
```

```
 'speechiness',
 'acousticness',
 'instrumentalness',
 'liveness',
 'valence',
 'tempo']
```

[26]:
```
list(enumerate(features_continuous_numerical))
```

[26]:
```
[(0, 'popularity'),
 (1, 'duration_ms'),
 (2, 'danceability'),
 (3, 'energy'),
 (4, 'loudness'),
 (5, 'speechiness'),
 (6, 'acousticness'),
 (7, 'instrumentalness'),
 (8, 'liveness'),
 (9, 'valence'),
 (10, 'tempo')]
```

[27]:
```
plt.figure(figsize = (10, 10))
for i,  col in enumerate(features_continuous_numerical):
    ax = plt.subplot(2, 6, i+1)
    sns.histplot(data=df, x = col, ax = ax)
```

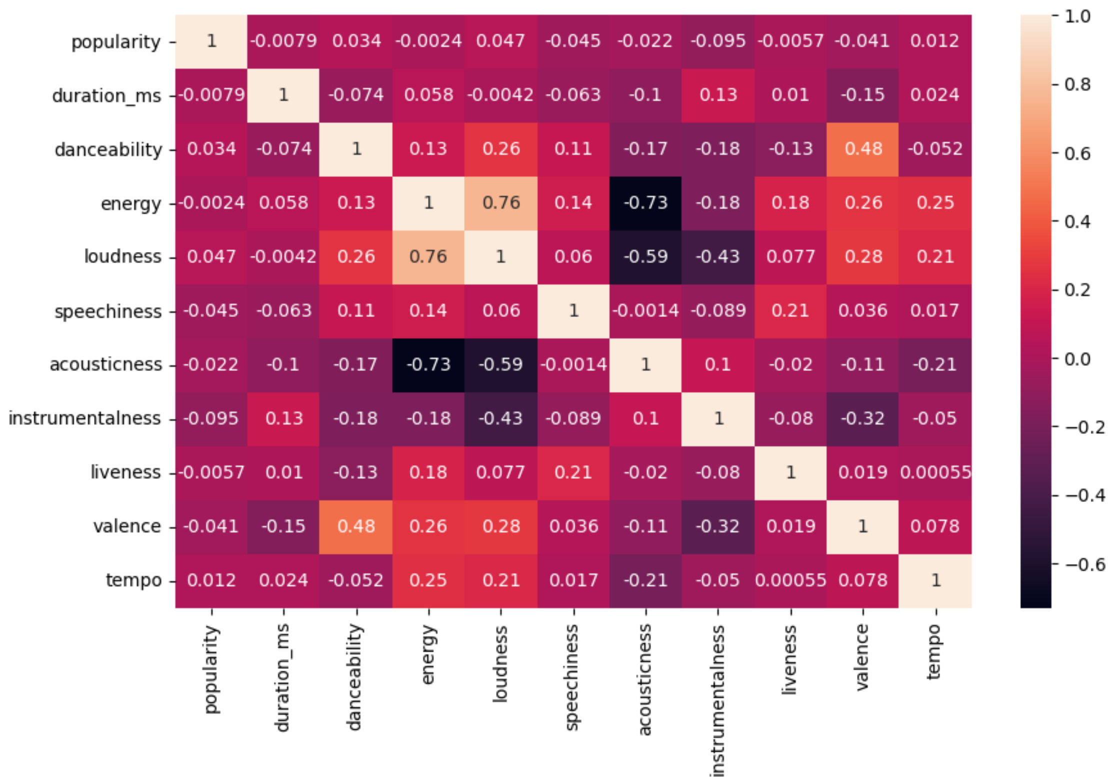


Observation:

- 1.We see that danceability, valence and tempo are almost normal distribution.
- 2.Loudness is left skewed.
- 3.Rest all are right skewed.

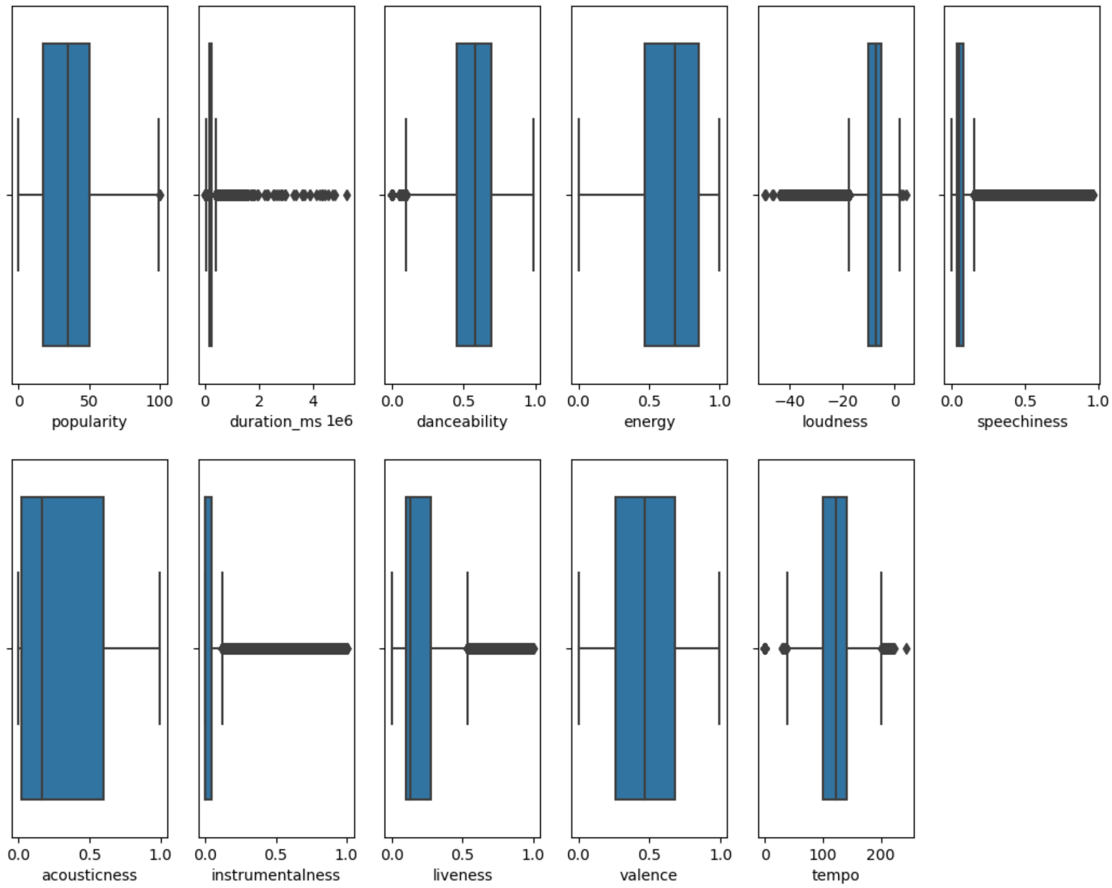
Lets check the correlation of the continuous features with the target.

[28]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df[features_continuous_numerical].corr(), annot=True)
plt.show()
```



We see that none of the continuous features has a great correlation with the target variable popularity. So we can perform the transformations if we opt for regresion model. The other models like SVM, Tree based methods, XG boost do not require such transformations. We would deal with the transformations in feature engineering.

[29]:
```
## Checking the outlier
plt.figure(figsize = (13, 10))
for i,  col in enumerate(features_continuous_numerical):
    ax = plt.subplot(2, 6, i+1)
    sns.boxplot(data=df, x = col, ax = ax)
```



We can see that apart from energy, acousticness and valence, there are a lot of outliers in most of the features.

[30]:
```
feature_categorical=[feature for feature in df.columns if df[feature].
  ↪dtypes=='O']
print('Number of categorical features:', len(feature_categorical))
df[feature_categorical].head()
```

```
Number of categorical features: 5
```

[30]:
```
                 track_id                   artists  \
0  5SuOikwiRyPMVoIQDJUgSV               Gen Hoshino
1  4qPNDBW1i3p13qLCtOKi3A              Ben Woodward
2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3  6lfxq3CG4xtTiEg7opyCyx               Kina Grannis
4  5vjLSffimiIP26QG5WcN2K          Chord Overstreet

                                          album_name  \
0                                             Comedy
1                                   Ghost (Acoustic)
```

```
2                                     To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                            Hold On

                   track_name track_genre
0                      Comedy    acoustic
1            Ghost - Acoustic    acoustic
2              To Begin Again    acoustic
3  Can't Help Falling In Love    acoustic
4                     Hold On    acoustic
```

[31]:
```python
list(enumerate(feature_categorical))
```

[31]:
```
[(0, 'track_id'),
 (1, 'artists'),
 (2, 'album_name'),
 (3, 'track_name'),
 (4, 'track_genre')]
```

[32]:
```python
for feature in feature_categorical:
    dataset=df.copy()
    print(feature, ': Number of unique entries:', dataset[feature].nunique())
```

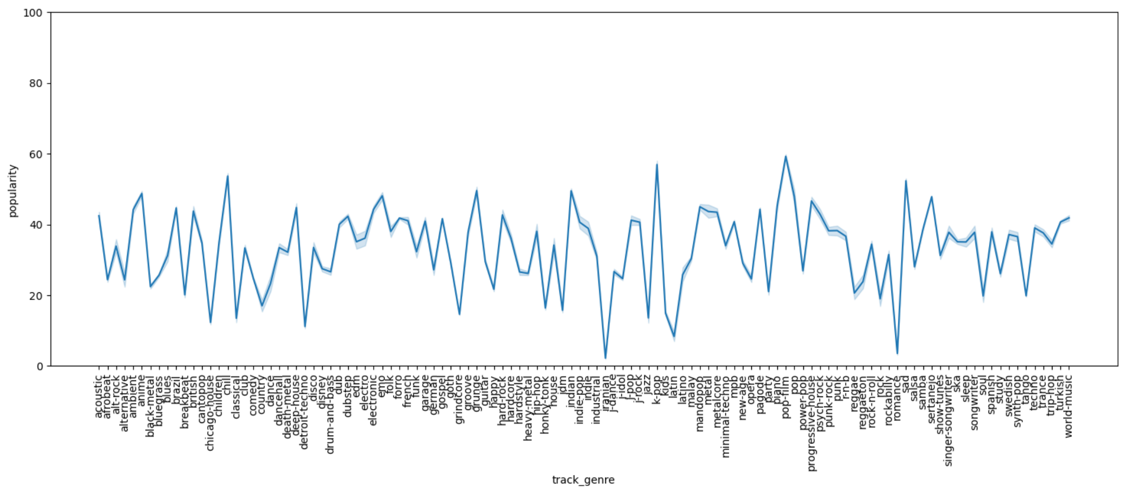
```
track_id : Number of unique entries: 89740
artists : Number of unique entries: 31437
album_name : Number of unique entries: 46589
track_name : Number of unique entries: 73608
track_genre : Number of unique entries: 114
```

Observation: There are a lot of unique entries in each of the categorical features.

Most of the categorical features are names like track name, album name, artist name, etc. track_id is unique for every song/track. We can later drop this. track_genre can have effect in popularity.

[33]:
```python
var = 'track_genre'
data = pd.concat([df['popularity'], df[var]], axis=1)
fig, ax = plt.subplots(figsize=(18,6))
fig = sns.lineplot(x=var, y='popularity', data=data)
plt.xticks(rotation=90)
fig.axis(ymin=0,ymax=100)
```

[33]:
```
(-5.65, 118.65, 0.0, 100.0)
```



3 Feature engineering

In feature engineering, we would perform the following steps:

- Convert the speechiness column to represent the presence of spoken words in a track.
- Remove the skewness of the data for continuous numerical features for prediction using regression algorithm.
- Encoding the categorical variables.
- Standardise the values of the variables to the same range.

4 Working on the speechiness column.

From the data description, the speechiness column detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.

So we can convert this column to discrete features. High, medium and low speechiness based on the range.

- 1.High for Values above 0.66
- 2.Medium for Values between 0.33 and 0.66
- 3.Low for values below 0.33

[34]:
```
df['speechiness'].sort_values()
```

[34]:
```
101681    0.000
98779     0.000
101663    0.000
```

```
101666    0.000
101667    0.000
          ...
18227     0.962
18432     0.962
18530     0.963
18504     0.963
18152     0.965
Name: speechiness, Length: 113549, dtype: float64
```

- Now, we would make a new column for the speechiness which wouuld depict whether the track has high,low or medium speechiness.

[35]:
```
speechiness_type=[]
for i in df.speechiness:
    if i<0.33:
        speechiness_type.append('Low')
    elif 0.33<=i<=0.66:
        speechiness_type.append('Medium')
    else:
        speechiness_type.append('High')
```

[36]:
```
df['speechiness_type']=speechiness_type
print(df.speechiness_type.value_counts())
df.head()
```

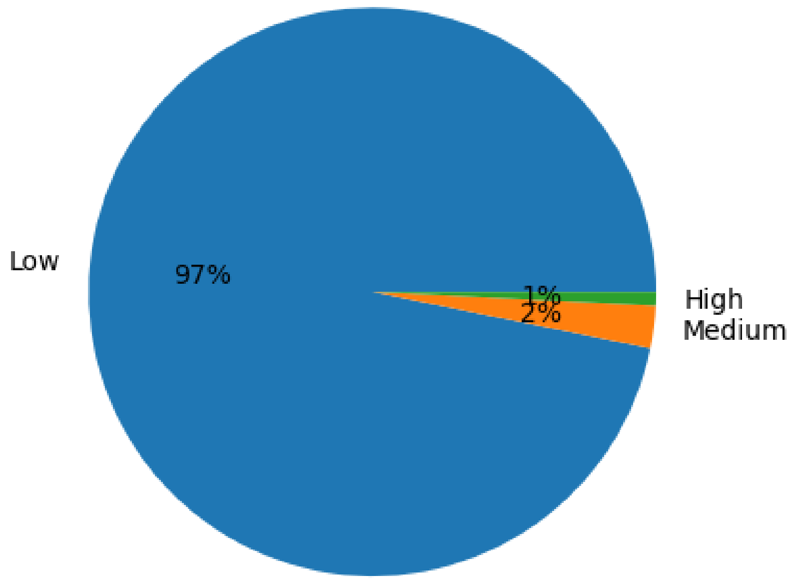
```
Low       109947
Medium      2726
High         876
Name: speechiness_type, dtype: int64
```

[36]:
```
                 track_id                 artists  \
0  5SuOikwiRyPMVoIQDJUgSV              Gen Hoshino
1  4qPNDBW1i3p13qLCt0Ki3A             Ben Woodward
2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3  6lfxq3CG4xtTiEg7opyCyx             Kina Grannis
4  5vjLSffimiIP26QG5WcN2K         Chord Overstreet

                                          album_name  \
0                                             Comedy
1                                   Ghost (Acoustic)
2                                     To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                            Hold On

                  track_name  popularity  duration_ms  explicit  \
0                     Comedy          73       230666     False
1           Ghost - Acoustic          55       149610     False
```

```
2              To Begin Again          57       210826     False
3  Can't Help Falling In Love          71       201933     False
4                     Hold On          82       198853     False

   danceability  energy  key  …  mode  speechiness  acousticness  \
0         0.676  0.4610    1  …     0       0.1430        0.0322
1         0.420  0.1660    1  …     1       0.0763        0.9240
2         0.438  0.3590    0  …     1       0.0557        0.2100
3         0.266  0.0596    0  …     1       0.0363        0.9050
4         0.618  0.4430    2  …     1       0.0526        0.4690

   instrumentalness  liveness  valence    tempo  time_signature  track_genre  \
0          0.000001    0.3580    0.715   87.917               4     acoustic
1          0.000006    0.1010    0.267   77.489               4     acoustic
2          0.000000    0.1170    0.120   76.332               4     acoustic
3          0.000071    0.1320    0.143  181.740               3     acoustic
4          0.000000    0.0829    0.167  119.949               4     acoustic

  speechiness_type
0              Low
1              Low
2              Low
3              Low
4              Low

[5 rows x 21 columns]
```

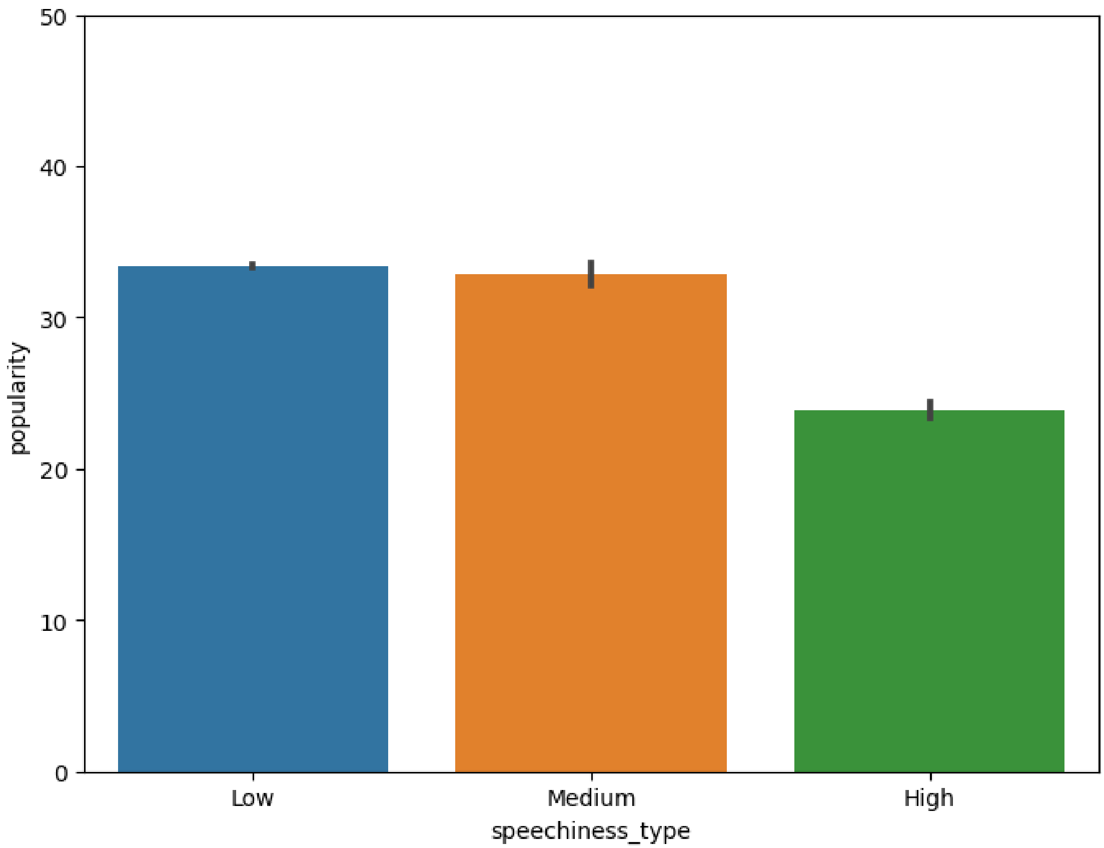
[37]:

```
plt.pie(x=df['speechiness_type'].value_counts(),labels=df['speechiness_type'].
 ↪unique(), autopct='%.0f%%')
plt.show()
```



So, 97% of tracks have low speechiness. Which means that most of the tracks are melodies rather than raps.

[38]:
```
## Lets check the relationship of the speechisess type with popularity.
var = 'speechiness_type'
data = pd.concat([df['popularity'], df[var]], axis=1)
fig, ax = plt.subplots(figsize=(8,6))
fig = sns.barplot(x=var, y='popularity', data=data)
fig.axis(ymin=0,ymax=50);
```



The median popularity of the tracks with low speechiness is high. It shows that people like more melodious songs as compared to rap songs.

5 Skewness

For regression models, we have to deal with the skewness of the continuous data. If the data is skewed, the regression models would not give good results for prediction.But for other models like decission tree, random forest etc. we do not need to modify the skewness. In this dataset, we saw that most of the continuous columns are skewed. We have to modify them for regression models. From EDA we found that the continuous features did not have a great correlation with the target variable popularity, So we can reduce their skewness and see the results.

Transforming the features to gaussian distribution for regression models.

[39]:
```
df.drop(df.columns[0], axis=1, inplace=True)
```

[40]:
```
#Selecting the numerical features:
feature_numerical=[feature for feature in df.columns if df[feature].dtypes!='O']
```

[41]:
```
#Selecting the discrete numerical features
feature_discrete_numerical=[feature for feature in feature_numerical if␣
↪df[feature].nunique()<50]
```

[42]:
```
#Selecting the continuous features
feature_continuous_numerical=[feature for feature in feature_numerical if␣
↪feature not in feature_discrete_numerical]
```

[43]:
```
df.shape
```

[43]: (113549, 20)

[44]:
```
dataset_log=df.copy()
dataset_reci=df.copy()
dataset_sqrt=df.copy()
dataset_expo=df.copy()
```

[45]:
```
from scipy import stats
```

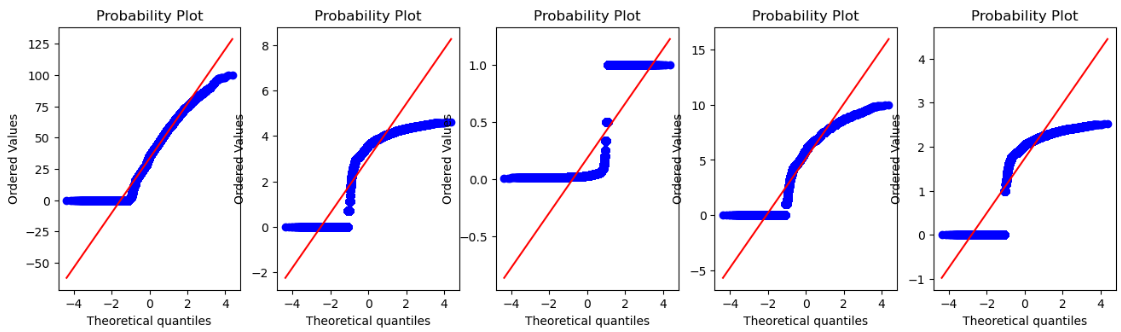
[46]:
```
for feature in feature_continuous_numerical:
    dataset_log[feature]=np.log(dataset_log[feature]+1)
    dataset_reci[feature]=1/(dataset_reci[feature]+1)
    dataset_sqrt[feature]=dataset_sqrt[feature]**(1/2)
    dataset_expo[feature]=dataset_expo[feature]**(1/5)
```

[47]:
```
from scipy.stats import skew
```

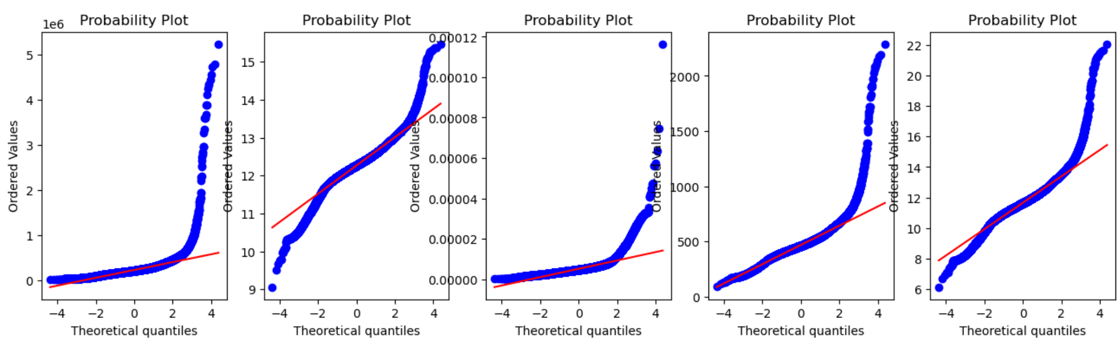
[48]:
```
for feature in feature_continuous_numerical:
    plt.figure(figsize=(16,4))
    plt.subplot(1,5,1)
    print(feature, 'original skewness:', skew(df[feature]))
    stats.probplot(df[feature], dist='norm', plot=plt)

    plt.subplot(1,5,2)
    print('logarithmic:', skew(dataset_log[feature]))
    stats.probplot(dataset_log[feature], dist='norm', plot=plt)

    plt.subplot(1,5,3)
    print('reciprocal: ', skew(dataset_reci[feature]))
    stats.probplot(dataset_reci[feature], dist='norm', plot=plt)

    plt.subplot(1,5,4)
    print('square-root:', skew(dataset_sqrt[feature]))
    stats.probplot(dataset_sqrt[feature], dist='norm', plot=plt)

    plt.subplot(1,5,5)
    print('exponential:', skew(dataset_expo[feature]))
```

```
    stats.probplot(dataset_expo[feature], dist='norm', plot=plt)

    plt.show()
```

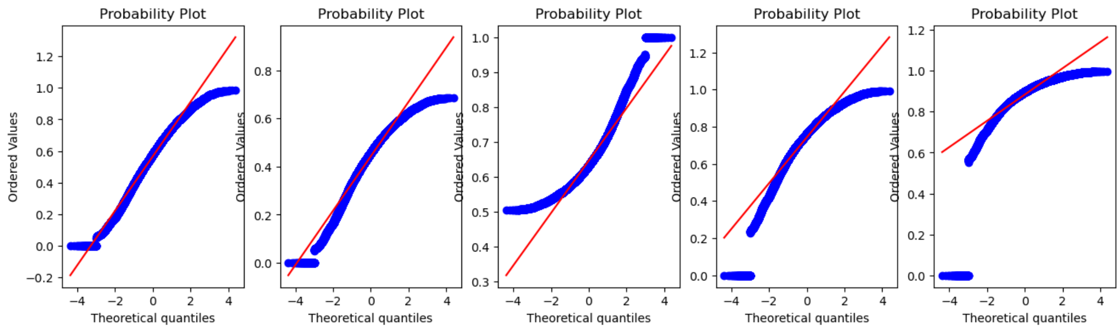
```
popularity original skewness: 0.042228099481099881
logarithmic: -1.3582344590230757
reciprocal:  1.9291529645017076
square-root: -0.8319211861334729
exponential: -1.637182842110766
```



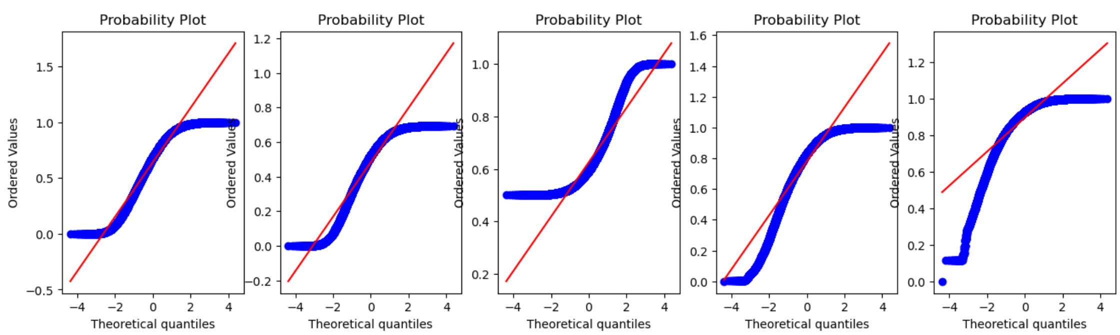

```
duration_ms original skewness: 10.814434004933338
logarithmic: -0.31958068443270016
reciprocal:  5.062710652230053
square-root: 1.7918392897162523
exponential: 0.3280005714079256
```



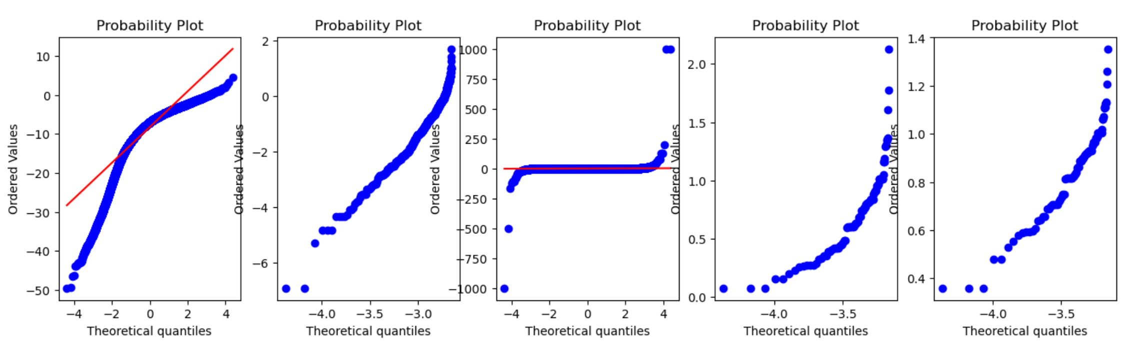

```
danceability original skewness: -0.4003991295600715
logarithmic: -0.7033043814280042
reciprocal:  1.0423257462450115
square-root: -1.078209424796711
exponential: -3.4689794311862756
```



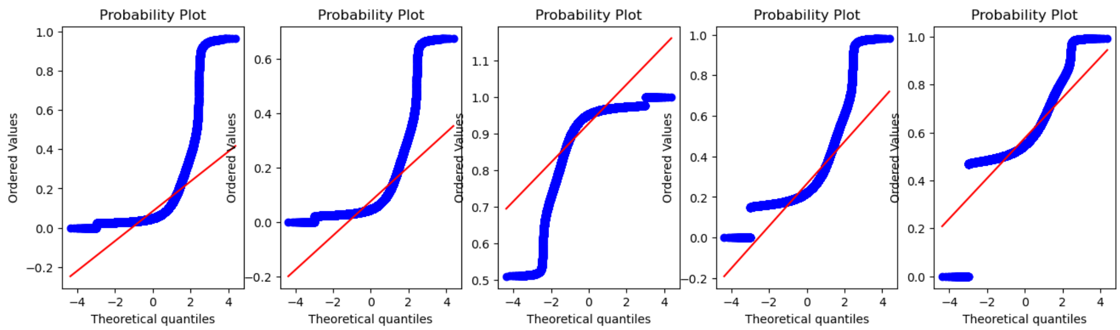
energy original skewness: -0.598542182428158
logarithmic: -0.8969101507400357
reciprocal: 1.2350332519840488
square-root: -1.2697363940462225
exponential: -2.2008891779513866



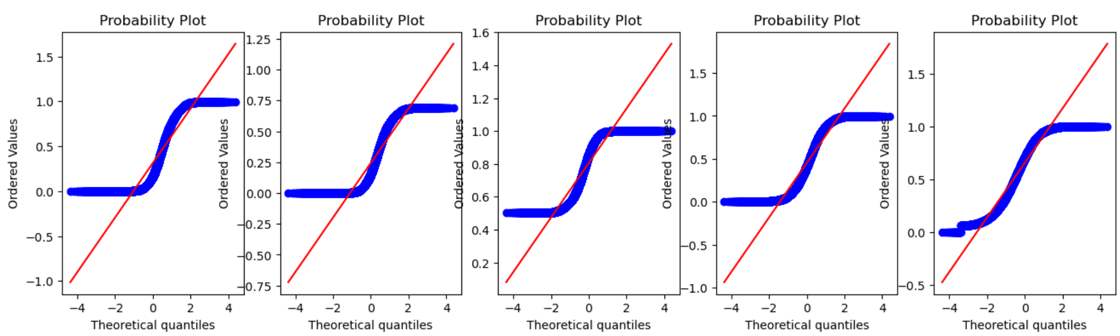
loudness original skewness: -2.0133133823721505
logarithmic: nan
reciprocal: 44.341700995068464
square-root: nan
exponential: nan



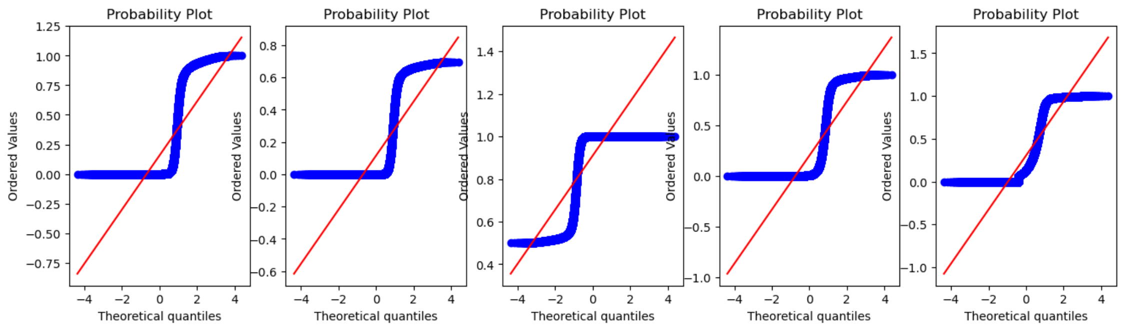
```
speechiness original skewness: 4.644508700286168
logarithmic: 3.7094268216147412
reciprocal:  -3.0189124109405174
square-root: 2.5106804244132315
exponential: 1.2585346167026825
```



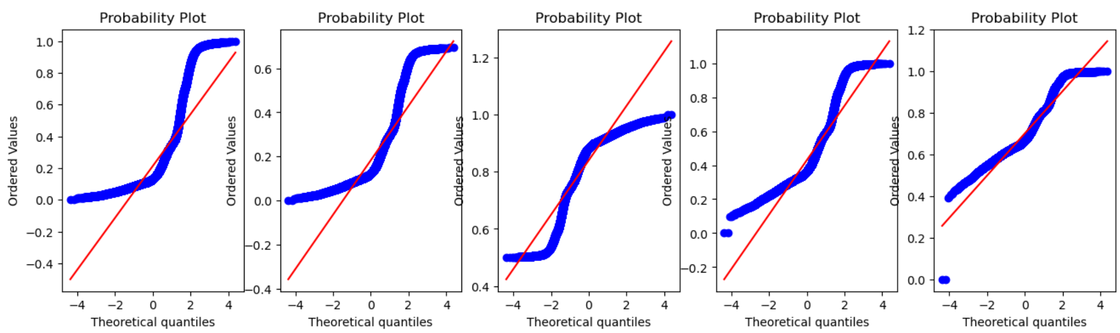

```
acousticness original skewness: 0.7302103030827026
logarithmic: 0.5532779927877304
reciprocal:  -0.3889970180312751
square-root: 0.19182418700584433
exponential: -0.45691650644198106
```



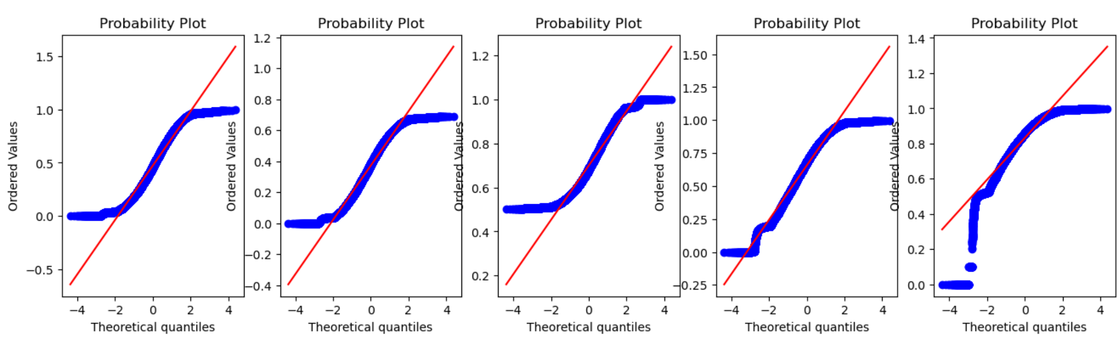

```
instrumentalness original skewness: 1.7377466866935405
logarithmic: 1.6547543527716166
reciprocal:  -1.5785769487599546
square-root: 1.4467885078056792
exponential: 0.929297080678929
```



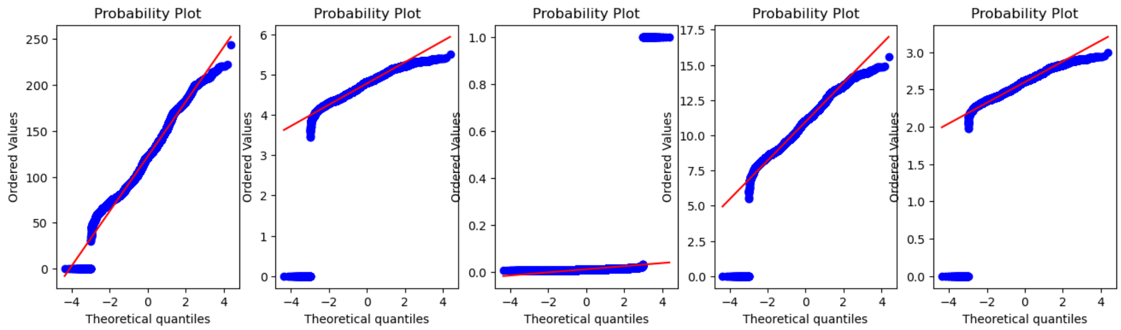

```
liveness original skewness: 2.1054497237799685
logarithmic: 1.7355487100916003
reciprocal:  -1.409425059871828
square-root: 1.3311332380750938
exponential: 0.8567637835174635
```




```
valence original skewness: 0.11477275798096229
logarithmic: -0.14268054744317982
reciprocal:  0.40566649459831094
square-root: -0.4685261380695686
exponential: -1.3963500280712502
```



```
tempo original skewness: 0.23160111991386964
logarithmic: -5.502750438724598
reciprocal:  26.694049451308892
square-root: -0.5935911783686282
exponential: -5.885488448988584
```



6 Observations:

1. None of the methods improve the skewness for popularity, danceability, energy, valence, tempo. It is better if the popularity, danceability, energy, valence, tempo is not transformed as it was already less skewed.
2. Exponential and logarithmic transformation gave good results for duration_ms.
3. For loudness, since there are negative values, exponential and logarithmic transformations don't work. Reciprocal do not give good results.
4. speechiness, instrumentalness, liveness is dealt well by exponential transformation.
5. acousticness is best dealt by square-root transformation. So, we would apply square root transformation to acousticness.Exponential transformation to speechiness, instrumentalness, liveness, duration_ms.

   Before doing any transformation, we can separate the data for regression and the other models.

[49]:
```
## Separating the data for regression and models
df['acousticness'] = df['acousticness']**(1/2)
df[['speechiness','instrumentalness','liveness','duration_ms']] =␣
↪df[['speechiness','instrumentalness','liveness','duration_ms']]**(1/5)
```

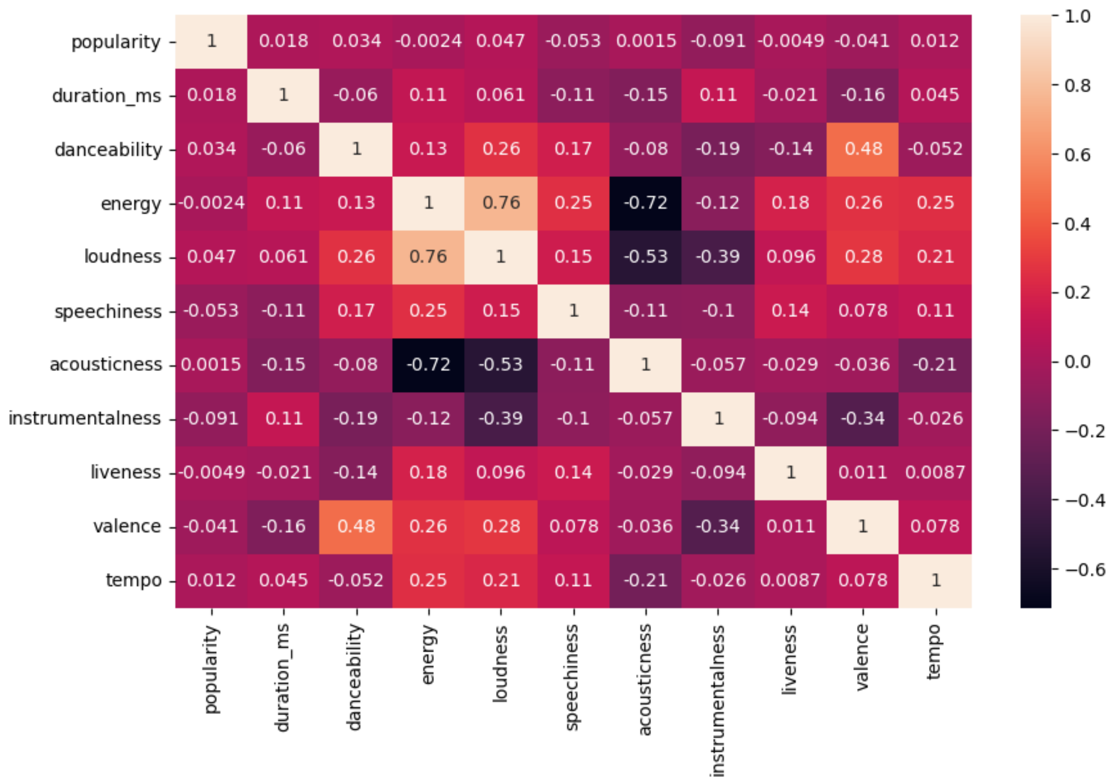
[50]:
```
df.head()
```

[50]:
```
                  artists                                         album_name  \
0             Gen Hoshino                                             Comedy
1            Ben Woodward                                   Ghost (Acoustic)
2  Ingrid Michaelson;ZAYN                                     To Begin Again
3            Kina Grannis  Crazy Rich Asians (Original Motion Picture Sou…
```

```
4       Chord Overstreet                                              Hold On

                 track_name  popularity  duration_ms  explicit  \
0                    Comedy          73    11.819435     False
1          Ghost - Acoustic          55    10.839073     False
2            To Begin Again          57    11.608733     False
3  Can't Help Falling In Love        71    11.509103     False
4                   Hold On          82    11.473778     False

   danceability  energy  key  loudness  mode  speechiness  acousticness  \
0         0.676  0.4610    1    -6.746     0     0.677746      0.179444
1         0.420  0.1660    1   -17.235     1     0.597730      0.961249
2         0.438  0.3590    0    -9.734     1     0.561269      0.458258
3         0.266  0.0596    0   -18.515     1     0.515206      0.951315
4         0.618  0.4430    2    -9.681     1     0.554878      0.684836

   instrumentalness  liveness  valence    tempo  time_signature track_genre  \
0          0.063221  0.814285    0.715   87.917               4    acoustic
1          0.088923  0.632214    0.267   77.489               4    acoustic
2          0.000000  0.651084    0.120   76.332               4    acoustic
3          0.147871  0.666983    0.143  181.740               3    acoustic
4          0.000000  0.607730    0.167  119.949               4    acoustic

  speechiness_type
0              Low
1              Low
2              Low
3              Low
4              Low
```

[51]:

```
plt.figure(figsize=(10,6))
sns.heatmap(df[features_continuous_numerical].corr(), annot=True)
plt.show()
```



7 Encoding the categorical columns

[52]:
```
feature_categorical=[feature for feature in df.columns if feature not in feature_numerical]
```

[53]:
```
dataset=df.copy()
for feature in feature_categorical:
    print(feature,': {}, missing values {}'.format(df[feature].nunique(), df[feature].isna().sum()))
```

```
artists : 31437, missing values 0
album_name : 46589, missing values 0
track_name : 73608, missing values 0
track_genre : 114, missing values 0
speechiness_type : 3, missing values 0
```

The track genre can definitely affect the popularity as it would depend on the individual. The artist name can also affect the song's popularity as a popular artist is likely to have more popular tracks. Track_ name and album_name can also affect the popularity. Since there are large number of unique entries in each of these columns, we would use BaseN encoder.

speechiness_type can be converted with one-hot encoding (precisely dummy encoding). As number of features in track_genre is high, so we can use BaseN encoding method.

[54]:
```
pip install category_encoders
```

```
Requirement already satisfied: category_encoders in
c:\users\home\anaconda3\lib\site-packages (2.6.0)
Requirement already satisfied: scipy>=1.0.0 in c:\users\home\anaconda3\lib\site-
packages (from category_encoders) (1.9.1)
Requirement already satisfied: patsy>=0.5.1 in c:\users\home\anaconda3\lib\site-
packages (from category_encoders) (0.5.2)
Requirement already satisfied: pandas>=1.0.5 in
c:\users\home\anaconda3\lib\site-packages (from category_encoders) (1.4.4)
Requirement already satisfied: scikit-learn>=0.20.0 in
c:\users\home\anaconda3\lib\site-packages (from category_encoders) (1.0.2)
Requirement already satisfied: statsmodels>=0.9.0 in
c:\users\home\anaconda3\lib\site-packages (from category_encoders) (0.13.2)
Requirement already satisfied: numpy>=1.14.0 in
c:\users\home\anaconda3\lib\site-packages (from category_encoders) (1.21.5)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\users\home\anaconda3\lib\site-packages (from
pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\home\anaconda3\lib\site-
packages (from pandas>=1.0.5->category_encoders) (2022.1)
Requirement already satisfied: six in c:\users\home\anaconda3\lib\site-packages
(from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\users\home\anaconda3\lib\site-
packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\home\anaconda3\lib\site-packages (from scikit-
learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in
c:\users\home\anaconda3\lib\site-packages (from
statsmodels>=0.9.0->category_encoders) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
c:\users\home\anaconda3\lib\site-packages (from
packaging>=21.3->statsmodels>=0.9.0->category_encoders) (3.0.9)
Note: you may need to restart the kernel to use updated packages.
```

[55]:
```
import category_encoders as ce
```

[56]:
```
encoder1=ce.BaseNEncoder(cols=['track_genre','album_name',␣
↪'track_name','artists'], base=10, return_df=True)
df=encoder1.fit_transform(df)
df.head()
```

[56]:
```
   artists_0  artists_1  artists_2  artists_3  artists_4  album_name_0  \
0          0          0          0          0          1             0
1          0          0          0          0          2             0
2          0          0          0          0          3             0
3          0          0          0          0          4             0
```

```
4          0          0          0          0          5             0

   album_name_1  album_name_2  album_name_3  album_name_4  …  acousticness  \
0             0             0             0             1  …      0.179444
1             0             0             0             2  …      0.961249
2             0             0             0             3  …      0.458258
3             0             0             0             4  …      0.951315
4             0             0             0             5  …      0.684836

   instrumentalness  liveness  valence    tempo  time_signature  \
0          0.063221  0.814285    0.715   87.917               4
1          0.088923  0.632214    0.267   77.489               4
2          0.000000  0.651084    0.120   76.332               4
3          0.147871  0.666983    0.143  181.740               3
4          0.000000  0.607730    0.167  119.949               4

   track_genre_0  track_genre_1  track_genre_2  speechiness_type
0              0              0              1               Low
1              0              0              1               Low
2              0              0              1               Low
3              0              0              1               Low
4              0              0              1               Low

[5 rows x 34 columns]
```

[57]:
```
df=pd.get_dummies(data=df, columns=['speechiness_type'], drop_first=True)
print(df.shape)
df.head()
```

```
(113549, 35)
```

[57]:
```
   artists_0  artists_1  artists_2  artists_3  artists_4  album_name_0  \
0          0          0          0          0          1             0
1          0          0          0          0          2             0
2          0          0          0          0          3             0
3          0          0          0          0          4             0
4          0          0          0          0          5             0

   album_name_1  album_name_2  album_name_3  album_name_4  …  \
0             0             0             0             1  …
1             0             0             0             2  …
2             0             0             0             3  …
3             0             0             0             4  …
4             0             0             0             5  …

   instrumentalness  liveness  valence    tempo  time_signature  \
0          0.063221  0.814285    0.715   87.917               4
1          0.088923  0.632214    0.267   77.489               4
```

```
2          0.000000  0.651084    0.120   76.332                4
3          0.147871  0.666983    0.143  181.740                3
4          0.000000  0.607730    0.167  119.949                4

   track_genre_0  track_genre_1  track_genre_2  speechiness_type_Low  \
0              0              0              1                     1
1              0              0              1                     1
2              0              0              1                     1
3              0              0              1                     1
4              0              0              1                     1

   speechiness_type_Medium
0                        0
1                        0
2                        0
3                        0
4                        0

[5 rows x 35 columns]
```

8 Feature Scaling

[58]:
```
df['explicit']=np.where(df['explicit']==False, 0,1)
```

[59]:
```
scaler=StandardScaler()
features_scaling=[feature for feature in feature_numerical if feature not in␣
↪['popularity','mode']]
scaler.fit(df[features_scaling])
```

[59]:
```
StandardScaler()
```

[60]:
```
scaler.transform(df[features_scaling])
```

[60]:
```
array([[ 0.1853255 , -0.30593202,  0.62839367, …,  0.92898358,
        -1.14299362,  0.22165951],
       [-0.92488754, -0.30593202, -0.84789057, …, -0.79939532,
        -1.4909088 ,  0.22165951],
       [-0.053284  , -0.30593202, -0.74408933, …, -1.36651965,
        -1.52951044,  0.22165951],
       …,
       [ 0.62849465, -0.30593202,  0.35735711, …,  1.03700726,
         0.34038354,  0.22165951],
       [ 0.75284102, -0.30593202,  0.11515423, …, -0.23612898,
         0.4598918 ,  0.22165951],
       [ 0.31240644, -0.30593202, -0.23661663, …,  0.90197766,
        -1.43389048,  0.22165951]])
```

[61]:
```
data_to_replace=pd.DataFrame(scaler.transform(df[features_scaling]),␣
↪columns=features_scaling)
```

[62]:
```
data_to_replace.head()
```

[62]:
```
   duration_ms  explicit  danceability    energy       key  loudness  \
0     0.185325 -0.305932      0.628394 -0.721328 -1.210476  0.298800
1    -0.924888 -0.305932     -0.847891 -1.896382 -1.210476 -1.794228
2    -0.053284 -0.305932     -0.744089 -1.127618 -1.491364 -0.297440
3    -0.166111 -0.305932     -1.735968 -2.320198 -1.491364 -2.049645
4    -0.206115 -0.305932      0.293923 -0.793026 -0.929587 -0.286864

   speechiness  acousticness  instrumentalness  liveness   valence     tempo  \
0     1.101557     -0.815926         -0.672445  1.078285  0.928984 -1.142994
1     0.234607      1.533645         -0.600612 -0.653979 -0.799395 -1.490909
2    -0.160432      0.021997         -0.849140 -0.474446 -1.366520 -1.529510
3    -0.659502      1.503789         -0.435859 -0.323181 -1.277786  1.987275
4    -0.229681      0.702936         -0.849140 -0.886924 -1.185194 -0.074292

   time_signature
0        0.221660
1        0.221660
2        0.221660
3       -2.092538
4        0.221660
```

[63]:
```
for feature in features_scaling:
    df[feature]=data_to_replace[feature].values
```

[64]:
```
df.isna().sum()
```

[64]:
```
artists_0                  0
artists_1                  0
artists_2                  0
artists_3                  0
artists_4                  0
album_name_0               0
album_name_1               0
album_name_2               0
album_name_3               0
album_name_4               0
track_name_0               0
track_name_1               0
track_name_2               0
track_name_3               0
track_name_4               0
popularity                 0
```

```
duration_ms                0
explicit                   0
danceability               0
energy                     0
key                        0
loudness                   0
mode                       0
speechiness                0
acousticness               0
instrumentalness           0
liveness                   0
valence                    0
tempo                      0
time_signature             0
track_genre_0              0
track_genre_1              0
track_genre_2              0
speechiness_type_Low       0
speechiness_type_Medium    0
dtype: int64
```

We would use correlation for feature selection

First separate the dependent and independent features.

[65]:
```python
X=df.drop(['popularity'], axis=1)
y=df['popularity']
```

[66]:
```python
X.head()
```

[66]:
```
   artists_0  artists_1  artists_2  artists_3  artists_4  album_name_0  \
0          0          0          0          0          1             0
1          0          0          0          0          2             0
2          0          0          0          0          3             0
3          0          0          0          0          4             0
4          0          0          0          0          5             0

   album_name_1  album_name_2  album_name_3  album_name_4  …  \
0             0             0             0             1  …
1             0             0             0             2  …
2             0             0             0             3  …
3             0             0             0             4  …
4             0             0             0             5  …

   instrumentalness  liveness   valence     tempo  time_signature  \
0         -0.672445  1.078285  0.928984 -1.142994        0.221660
1         -0.600612 -0.653979 -0.799395 -1.490909        0.221660
2         -0.849140 -0.474446 -1.366520 -1.529510        0.221660
```

```
3         -0.435859 -0.323181 -1.277786  1.987275       -2.092538
4         -0.849140 -0.886924 -1.185194 -0.074292        0.221660

   track_genre_0  track_genre_1  track_genre_2  speechiness_type_Low  \
0              0              0              1                     1
1              0              0              1                     1
2              0              0              1                     1
3              0              0              1                     1
4              0              0              1                     1

   speechiness_type_Medium
0                        0
1                        0
2                        0
3                        0
4                        0

[5 rows x 34 columns]
```

Now we would use train-test-split to prevent the overfitting.

[67]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.3,
 ↪random_state=7)
```

[68]:
```python
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(79484, 34) (34065, 34)
(79484,) (34065,)
```

[69]:
```python
def correlation(dataset,threshold):
    correlated_columns=set()
    correlation_matrix=dataset.corr()
    for i in range(len(correlation_matrix.columns)):
        for j in range(i):
            if abs(correlation_matrix.iloc[i,j])>threshold:
                colname=correlation_matrix.columns[i]
                correlated_columns.add(colname)
    return correlated_columns
```

[70]:
```python
corr_features=correlation(X_train,0.7)
print(len(set(corr_features)))
print(corr_features)
```

```
5
{'speechiness_type_Medium', 'album_name_0', 'loudness', 'acousticness',
'track_name_0'}
```

[71]:
```
X_train_corr=X_train.copy()
X_test_corr=X_test.copy()
```

[72]:
```
X_train_corr.drop(corr_features, axis=1, inplace=True)
X_test_corr.drop(corr_features, axis=1, inplace=True)
print(X_train_corr.shape, X_test_corr.shape)
```

```
(79484, 29) (34065, 29)
```

[73]:
```
X_train_corr.isna().sum()
```

[73]:
```
artists_0               0
artists_1               0
artists_2               0
artists_3               0
artists_4               0
album_name_1            0
album_name_2            0
album_name_3            0
album_name_4            0
track_name_1            0
track_name_2            0
track_name_3            0
track_name_4            0
duration_ms             0
explicit                0
danceability            0
energy                  0
key                     0
mode                    0
speechiness             0
instrumentalness        0
liveness                0
valence                 0
tempo                   0
time_signature          0
track_genre_0           0
track_genre_1           0
track_genre_2           0
speechiness_type_Low    0
dtype: int64
```

9 Model Training

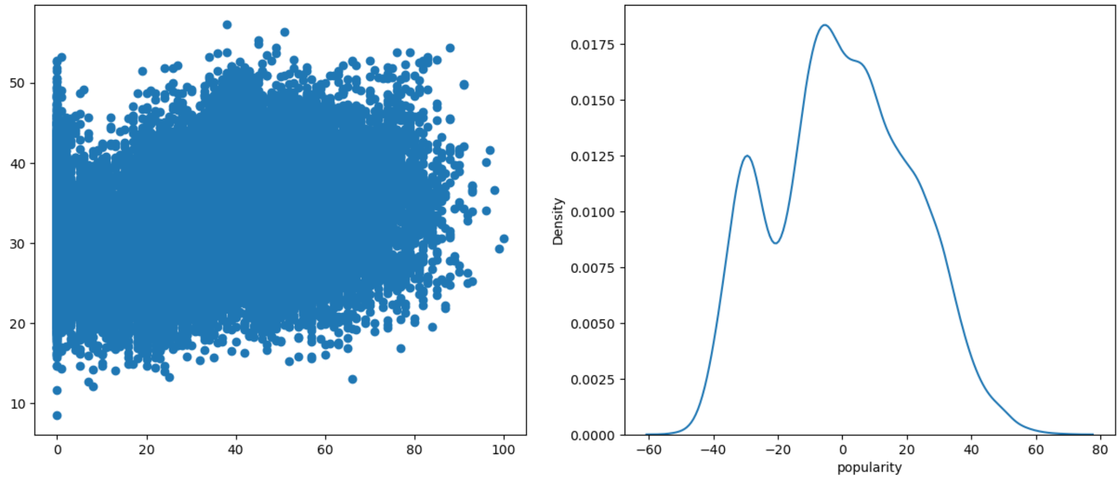
[74]:
```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from xgboost import XGBRegressor, XGBRFRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import BayesianRidge
```

[75]:
```
lr=LinearRegression()
lasso=Lasso()
ridge=Ridge()
xgbreg=XGBRegressor()
xgbrfreg=XGBRFRegressor()
dtree=DecisionTreeRegressor()
bayridge=BayesianRidge()
```

[76]:
```
def model(name):
    name.fit(X_train_corr,y_train)
    prediction=name.predict(X_test_corr)
    residual=y_test-prediction

    plt.figure(figsize=(15,6))

    plt.subplot(1,2,1)
    plt.scatter(y_test,prediction)

    plt.subplot(1,2,2)
    sns.distplot(residual, hist=False, kde=True)
    plt.show()
```

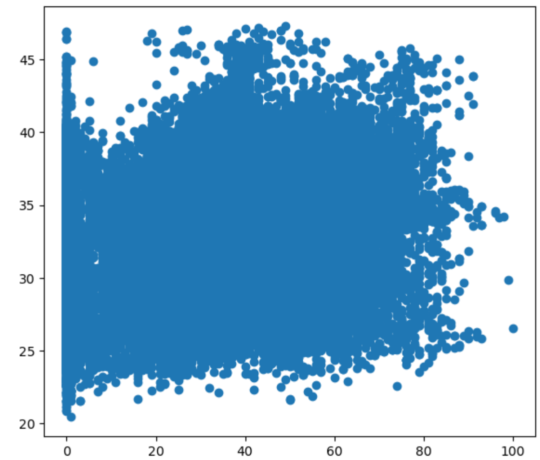
[77]:
```
import warnings
warnings.filterwarnings("ignore")
```

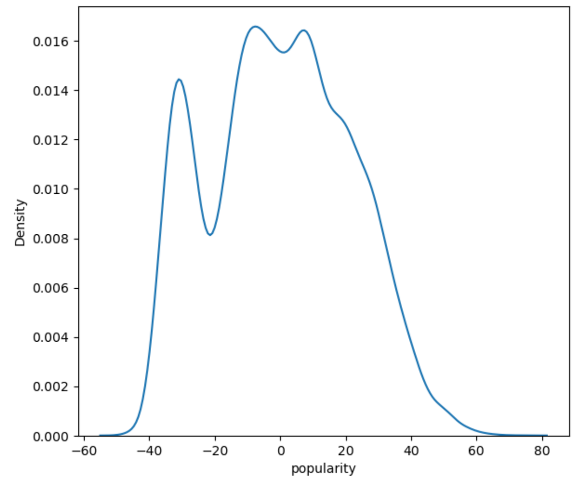
[78]:
```
model(lr)
```



[79]:

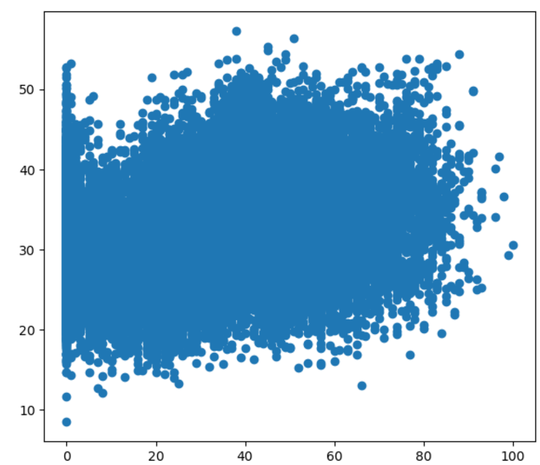
```
model(lasso)
```

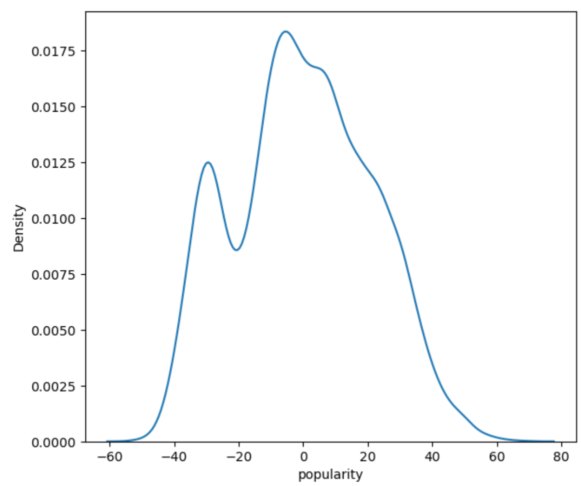




[80]:

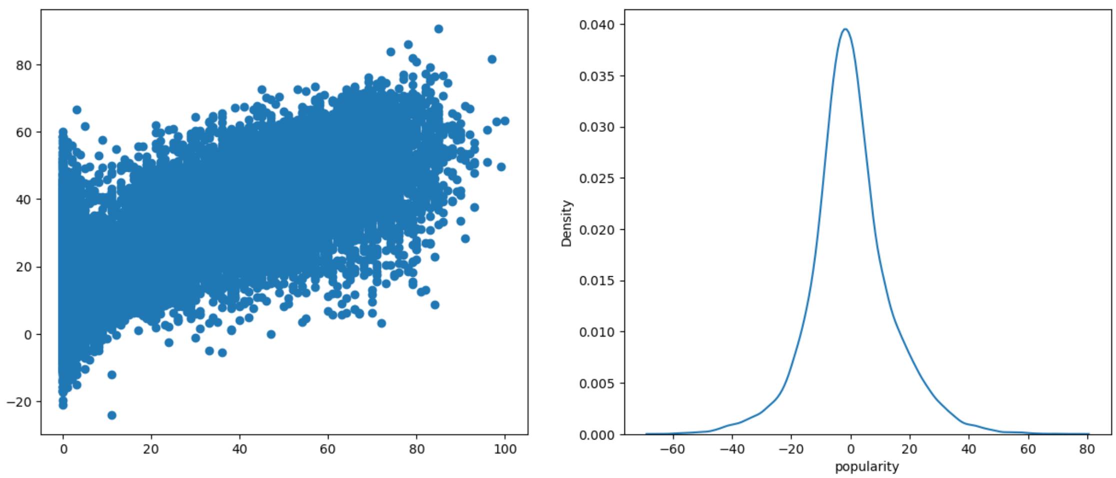
```
model(ridge)
```



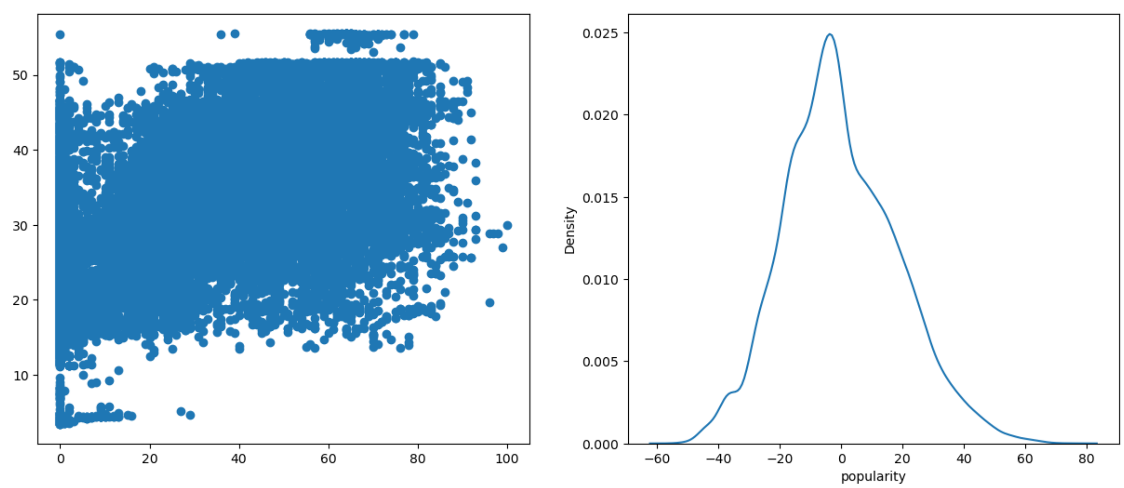


[81]:

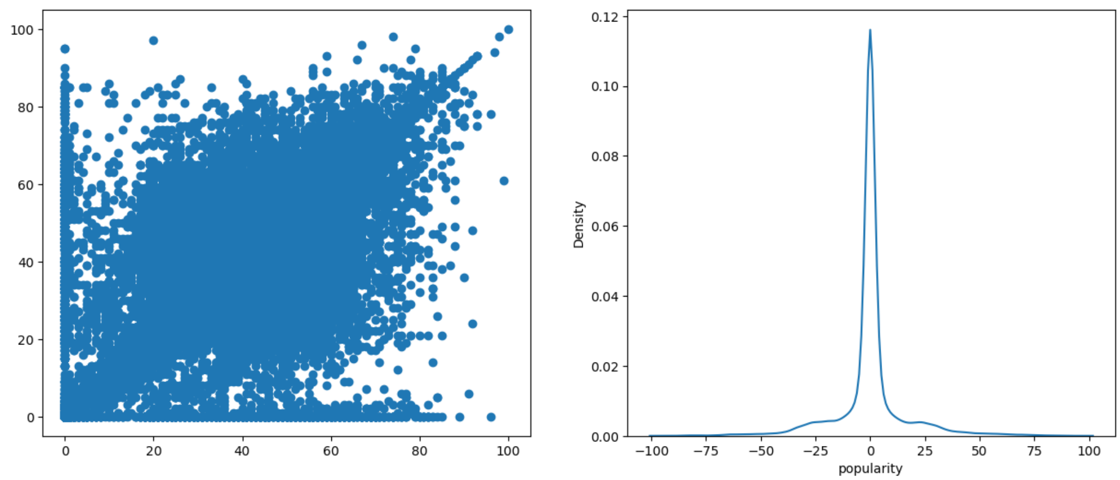
```
model(xgbreg)
```



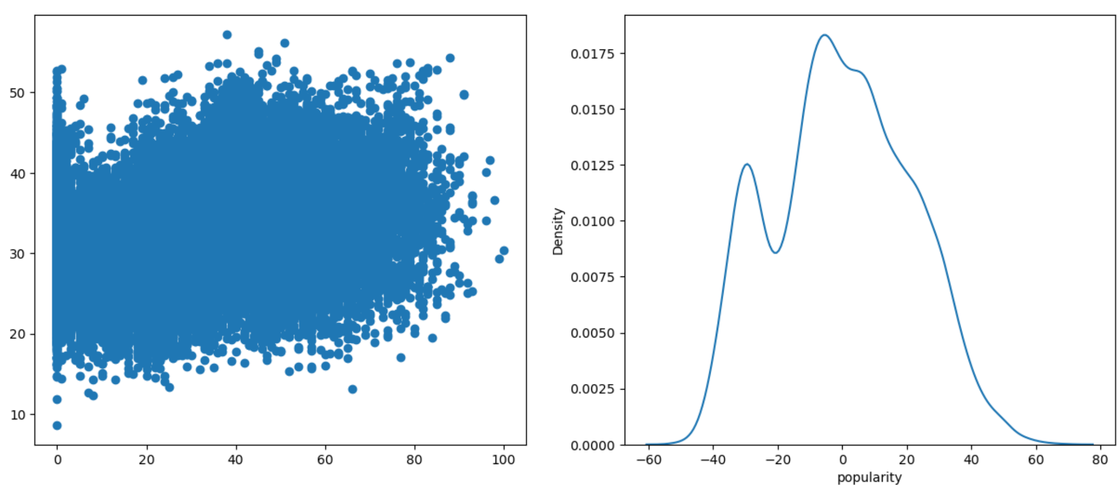

```
[82]: model(xgbrfreg)
```




```
[83]: model(dtree)
```




```
[84]: model(bayridge)
```



Comments:

1. Xgboost is giving the most promising results among the models
2. Linear models do not perform well
3. Decission tree is also performing well

10 Performance metrics

```
[85]: from sklearn.metrics import mean_squared_error, mean_absolute_error
      from sklearn.metrics import r2_score
```

[86]:
```
algos=[lr, lasso, ridge, xgbreg, xgbrfreg, dtree, bayridge]
MSE=[]
ABMSE=[]
R2_score=[]
for feature in algos:
    prediction=feature.predict(X_test_corr)
    mse=mean_squared_error(y_test, prediction)
    abmse=mean_absolute_error(y_test, prediction)
    score=r2_score(y_test, prediction)
    MSE.append(mse)
    ABMSE.append(abmse)
    R2_score.append(score)
```

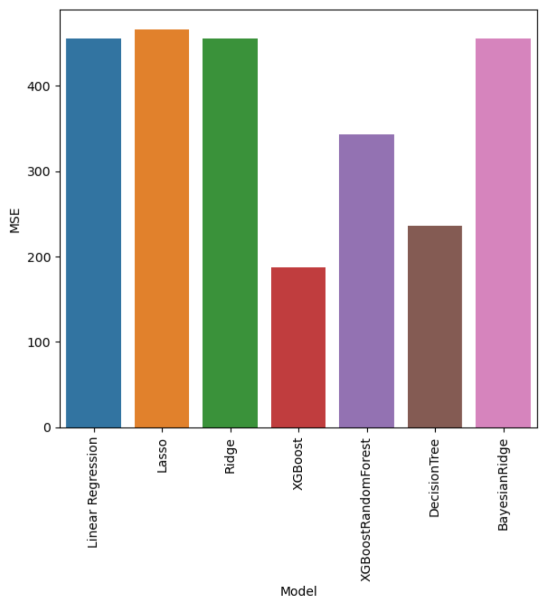
[87]:
```
algosname=['Linear Regression', 'Lasso', 'Ridge', 'XGBoost',␣
 ↪'XGBoostRandomForest','DecisionTree','BayesianRidge']
metrics=pd.DataFrame(list(zip(algosname,MSE,ABMSE,R2_score)),␣
 ↪columns=['Model','MSE', 'ABMSE', 'R2_score'])
```

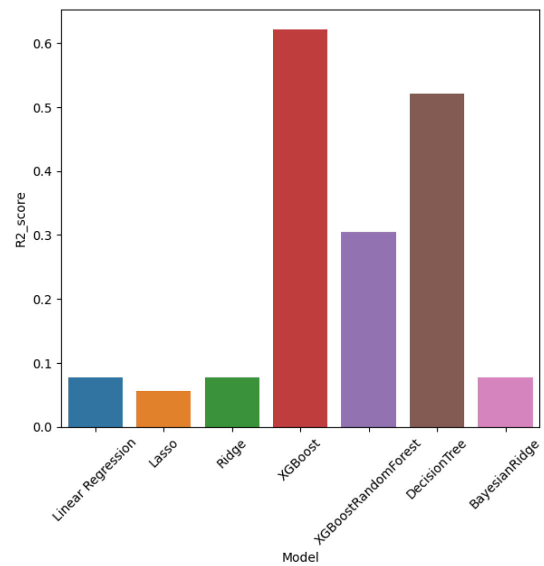
[88]:
```
metrics
```

[88]:

	Model	MSE	ABMSE	R2_score
0	Linear Regression	455.348812	17.556276	0.077834
1	Lasso	465.894997	17.976374	0.056476
2	Ridge	455.348820	17.556298	0.077834
3	XGBoost	186.876335	10.006477	0.621541
4	XGBoostRandomForest	343.394806	14.755475	0.304561
5	DecisionTree	236.190181	7.755409	0.521671
6	BayesianRidge	455.357794	17.561725	0.077815

[89]:
```
plt.figure(figsize=(15,6))

plt.subplot(1,2,1)
sns.barplot(x='Model', y='MSE', data=metrics)
plt.xticks(rotation=90)

plt.subplot(1,2,2)
sns.barplot(x='Model', y='R2_score', data=metrics)
plt.xticks(rotation=45)
plt.show()
```





Comments:

1. From the performance metrics, the XGBoost (Regressor) and Decission Tree performs better than the rest of the models.
2. XGBoost has the highest r2_score and the least mean squared error among the models, with 0.62 and 186.87 respectively.
3. Therefore, we will use this XGBoost model for future predictions.

The possible reason why the XGBoost and Decision Tree performs better could be because the data itself is not linear. Hence, the tree based models are performing well. Both decission tree and XgBoost use tree based models for predictions rather than fitting a line or a curve to the data points.