

# Spotify\_Final\_Data\_Analysis-Current

October 3, 2023

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

```
[2]: sns.set_style("darkgrid")
```

## 0.1 Data Importing -

```
[3]: df = pd.read_csv(r"C:\Users\amitm\Desktop\data.csv")
df.drop("Unnamed: 0", axis = 1, inplace =True)
df.head()
```

```
[3]:
```

	acousticness	danceability	duration_ms	energy	instrumentalness	key	\
0	0.0102	0.833	204600	0.434	0.021900	2	
1	0.1990	0.743	326933	0.359	0.006110	1	
2	0.0344	0.838	185707	0.412	0.000234	2	
3	0.6040	0.494	199413	0.338	0.510000	5	
4	0.1800	0.678	392893	0.561	0.512000	5	

	liveness	loudness	mode	speechiness	tempo	time_signature	valence	\
0	0.1650	-8.795	1	0.4310	150.062	4.0	0.286	
1	0.1370	-10.401	1	0.0794	160.083	4.0	0.588	
2	0.1590	-7.148	1	0.2890	75.044	4.0	0.173	
3	0.0922	-15.236	1	0.0261	86.468	4.0	0.230	
4	0.4390	-11.648	0	0.0694	174.004	4.0	0.904	

	target	song_title	artist
0	1	Mask Off	Future
1	1	Redbone	Childish Gambino
2	1	Xanny Family	Future
3	1	Master Of None	Beach House
4	1	Parallel Lines	Junior Boys

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

```
[4]:
```

	acousticness	danceability	duration_ms	energy	instrumentalness	key	\
0	0.0102	0.833	204600	0.434	0.021900	2	

1	0.1990	0.743	326933	0.359	0.006110	1
2	0.0344	0.838	185707	0.412	0.000234	2
3	0.6040	0.494	199413	0.338	0.510000	5
4	0.1800	0.678	392893	0.561	0.512000	5

	liveness	loudness	mode	speechiness	tempo	time_signature	valence	\
0	0.1650	-8.795	1	0.4310	150.062	4.0	0.286	
1	0.1370	-10.401	1	0.0794	160.083	4.0	0.588	
2	0.1590	-7.148	1	0.2890	75.044	4.0	0.173	
3	0.0922	-15.236	1	0.0261	86.468	4.0	0.230	
4	0.4390	-11.648	0	0.0694	174.004	4.0	0.904	

	target	song_title	artist
0	1	Mask Off	Future
1	1	Redbone	Childish Gambino
2	1	Xanny Family	Future
3	1	Master Of None	Beach House
4	1	Parallel Lines	Junior Boys

## 0.2 data cleaning -

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

```
[5]: acousticness      0
danceability          0
duration_ms          0
energy                0
instrumentalness      0
key                  0
liveness              0
loudness              0
mode                  0
speechiness           0
tempo                 0
time_signature        0
valence               0
target                0
song_title            0
artist                0
dtype: int64
```

```
[6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2017 entries, 0 to 2016
Data columns (total 16 columns):
#   Column              Non-Null Count  Dtype
---
```

```

0    acousticness      2017 non-null    float64
1    danceability      2017 non-null    float64
2    duration_ms       2017 non-null    int64
3    energy            2017 non-null    float64
4    instrumentalness   2017 non-null    float64
5    key               2017 non-null    int64
6    liveness          2017 non-null    float64
7    loudness          2017 non-null    float64
8    mode              2017 non-null    int64
9    speechiness       2017 non-null    float64
10   tempo             2017 non-null    float64
11   time_signature     2017 non-null    float64
12   valence            2017 non-null    float64
13   target             2017 non-null    int64
14   song_title         2017 non-null    object
15   artist             2017 non-null    object

```

dtypes: float64(10), int64(4), object(2)

memory usage: 252.2+ KB

```
[7]: df.shape
```

```
[7]: (2017, 16)
```

```
[8]: df.columns
```

```
[8]: Index(['acousticness', 'danceability', 'duration_ms', 'energy',
          'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
          'speechiness', 'tempo', 'time_signature', 'valence', 'target',
          'song_title', 'artist'],
          dtype='object')
```

```
[9]: len(df.columns)
```

```
[9]: 16
```

```
[10]: df.describe()
```

```
[10]:
```

	acousticness	danceability	duration_ms	energy \					
count	2017.000000	2017.000000	2.017000e+03	2017.000000					
mean	0.187590	0.618422	2.463062e+05	0.681577					
std	0.259989	0.161029	8.198181e+04	0.210273					
min	0.000003	0.122000	1.604200e+04	0.014800					
25%	0.009630	0.514000	2.000150e+05	0.563000					
50%	0.063300	0.631000	2.292610e+05	0.715000					
75%	0.265000	0.738000	2.703330e+05	0.846000					
max	0.995000	0.984000	1.004627e+06	0.998000					

	instrumentalness	key	liveness	loudness	mode \				
--	------------------	-----	----------	----------	--------	--	--	--	--

count	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000
mean	0.133286	5.342588	0.190844	-7.085624	0.612295
std	0.273162	3.648240	0.155453	3.761684	0.487347
min	0.000000	0.000000	0.018800	-33.097000	0.000000
25%	0.000000	2.000000	0.092300	-8.394000	0.000000
50%	0.000076	6.000000	0.127000	-6.248000	1.000000
75%	0.054000	9.000000	0.247000	-4.746000	1.000000
max	0.976000	11.000000	0.969000	-0.307000	1.000000

	speechiness	tempo	time_signature	valence	target
count	2017.000000	2017.000000	2017.000000	2017.000000	2017.000000
mean	0.092664	121.603272	3.968270	0.496815	0.505702
std	0.089931	26.685604	0.255853	0.247195	0.500091
min	0.023100	47.859000	1.000000	0.034800	0.000000
25%	0.037500	100.189000	4.000000	0.295000	0.000000
50%	0.054900	121.427000	4.000000	0.492000	1.000000
75%	0.108000	137.849000	4.000000	0.691000	1.000000
max	0.816000	219.331000	5.000000	0.992000	1.000000

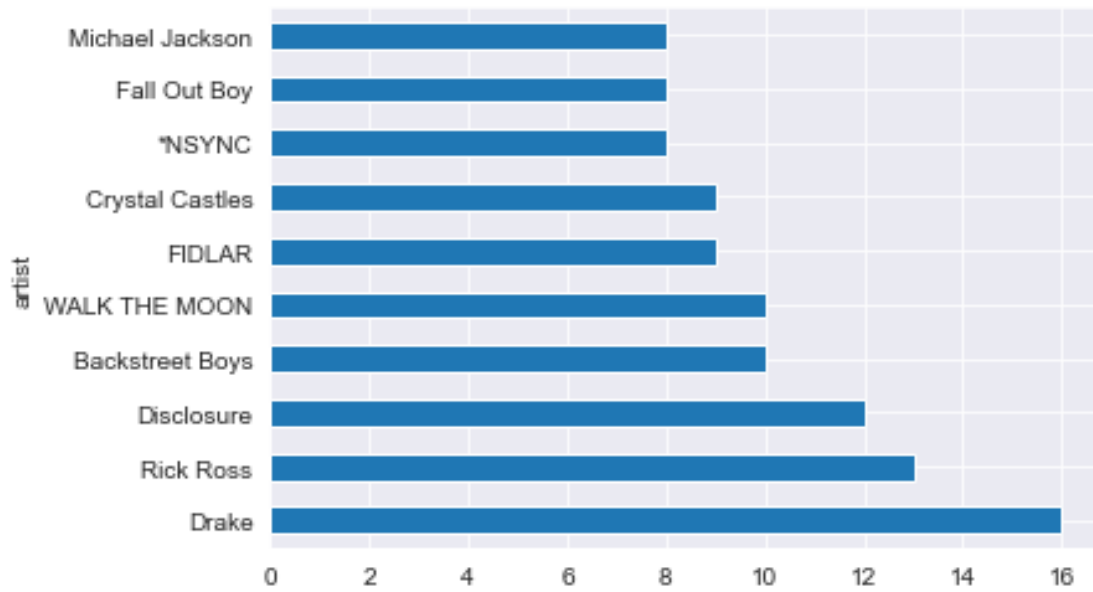
### 0.3 DATA ANALYSIS -

#### 1 Top 10 Popular Artists?

```
[23]: top_ten_artists = df.groupby("artist").count().sort_values(by="song_title",
    ↪ascending=False)["song_title"][:10]
top_ten_artists
```

```
[23]: artist
Drake          16
Rick Ross      13
Disclosure      12
Backstreet Boys 10
WALK THE MOON  10
FIDLAR         9
Crystal Castles 9
*NSYNC         8
Fall Out Boy   8
Michael Jackson 8
Name: song_title, dtype: int64
```

```
[12]: top_ten_artists.plot.barh()
plt.show()
```

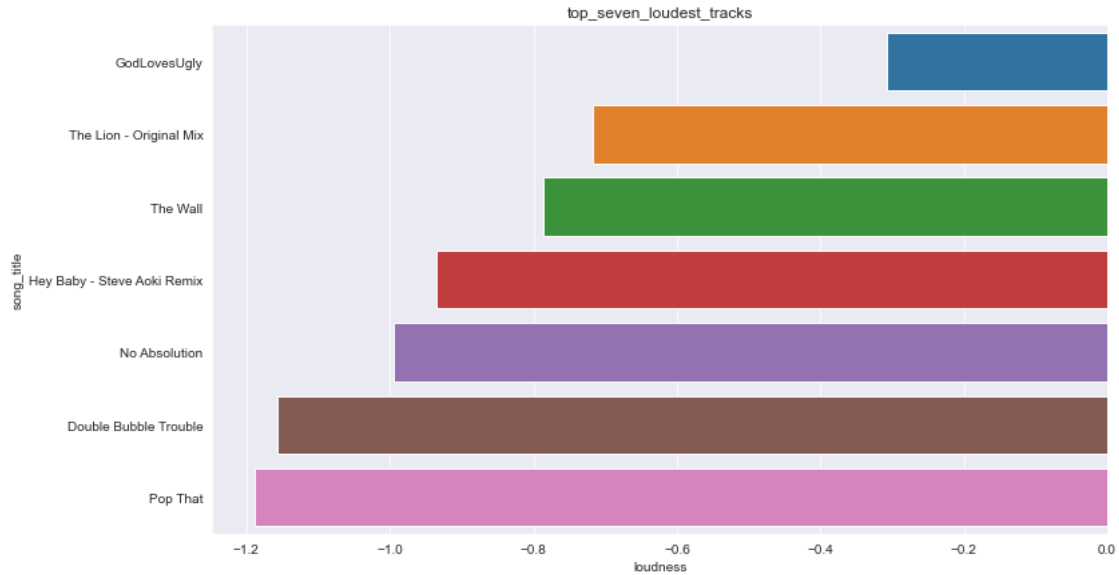


### 1.1 Top 7 Loudest tracks?

```
[13]: top_seven_loudest_tracks = df[["loudness", "song_title"]].
      ↪sort_values(by="loudness",ascending = False)[:7]
top_seven_loudest_tracks
```

```
[13]:      loudness      song_title
195    -0.307      GodLovesUgly
636    -0.718  The Lion - Original Mix
1443   -0.787      The Wall
2010   -0.935  Hey Baby - Steve Aoki Remix
1299   -0.994      No Absolution
205    -1.157  Double Bubble Trouble
629    -1.188      Pop That
```

```
[14]: plt.figure(figsize=(12,7))
sns.barplot(x="loudness" , y= "song_title",data =top_seven_loudest_tracks)
plt.title("top_seven_loudest_tracks")
plt.show()
```



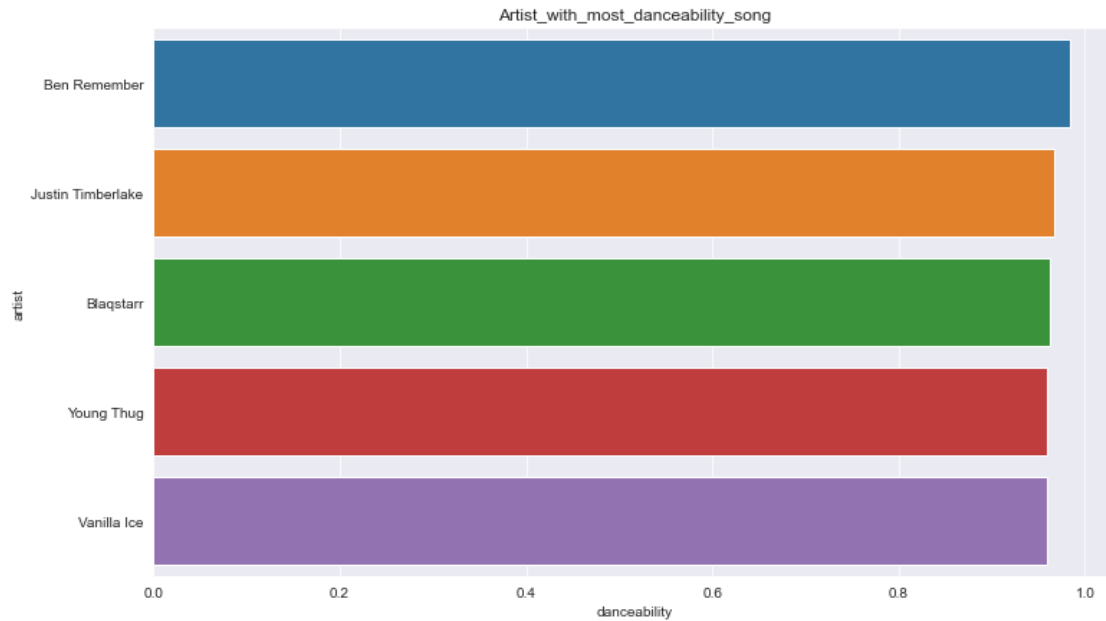
## 1.2 Artists with more danceability songs?

```
[15]: top_five_artists_danceable_songs=df[["artist", "song_title", "danceability"]].
      ↪sort_values(by="danceability",ascending=False)[:5]
top_five_artists_danceable_songs
```

```
[15]:
```

	artist	song_title	danceability
1433	Ben Remember	Flashwind - Radio Edit	0.984
1901	Justin Timberlake	SexyBack	0.967
604	Blaqstarr	Check Me Out Like	0.962
32	Young Thug	Best Friend	0.959
1957	Vanilla Ice	Ice Ice Baby	0.959

```
[16]: plt.figure(figsize=(12,7))
      sns.barplot(x="danceability" , y= "artist", data_
      ↪top_five_artists_danceable_songs)
      plt.title("Artist_with_most_danceability_song")
      plt.show()
```



### 1.3 Top 10 Instrumental songs?

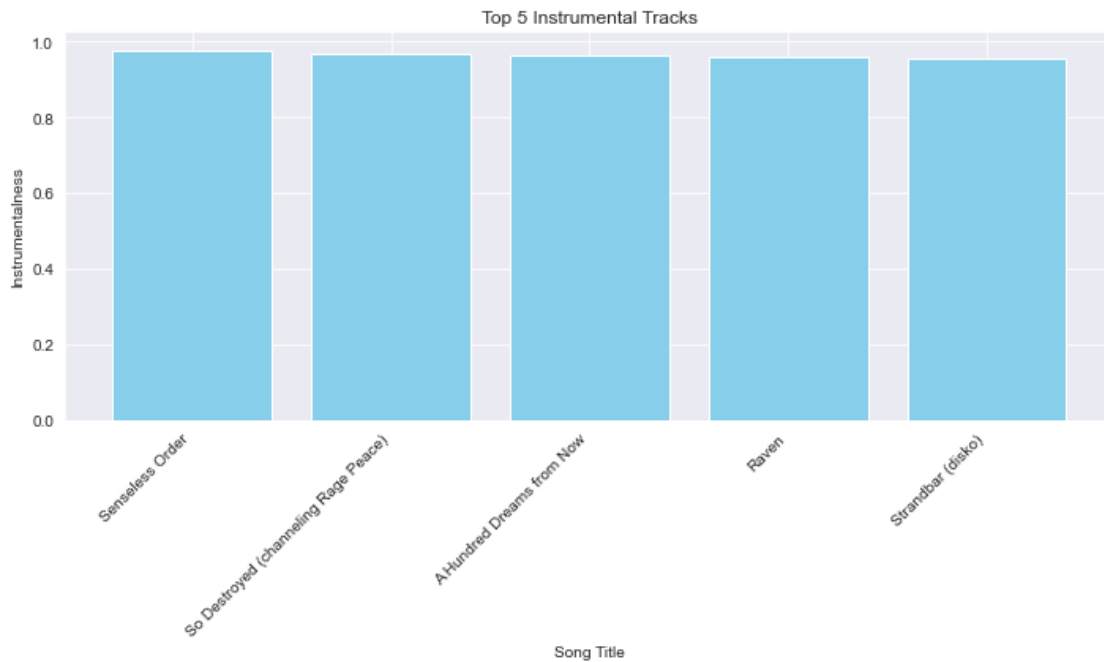
```
[17]: top_ten_instrumental_tracks = df[["song_title", "artist", "instrumentalness"]].
      ↪sort_values(by="instrumentalness", ascending = False) [:5]
top_ten_instrumental_tracks
```

```
[17]:
```

	song_title	artist	instrumentalness
1313	Senseless Order	Signs of the Swarm	0.976
271	So Destroyed (channeling Rage Peace)	Prince Rama	0.968
1575	A Hundred Dreams from Now	Ray Bryant	0.964
1619	Raven	John Dahlbäck	0.958
725	Strandbar (disko)	Todd Terje	0.957

```
[18]: song_titles = top_ten_instrumental_tracks["song_title"]
instrumentalness_values = top_ten_instrumental_tracks["instrumentalness"]
# Create a bar chart
plt.figure(figsize=(10, 6)) # Set the figure size (width, height)
plt.bar(song_titles, instrumentalness_values, color='skyblue')
plt.xlabel("Song Title")
```

```
plt.ylabel("Instrumentalness")
plt.title("Top 5 Instrumental Tracks")
plt.xticks(rotation=45, ha="right") # Rotate x-axis labels for readability
# Show the plot
plt.tight_layout() # Adjust layout to prevent clipping of labels
plt.show()
```

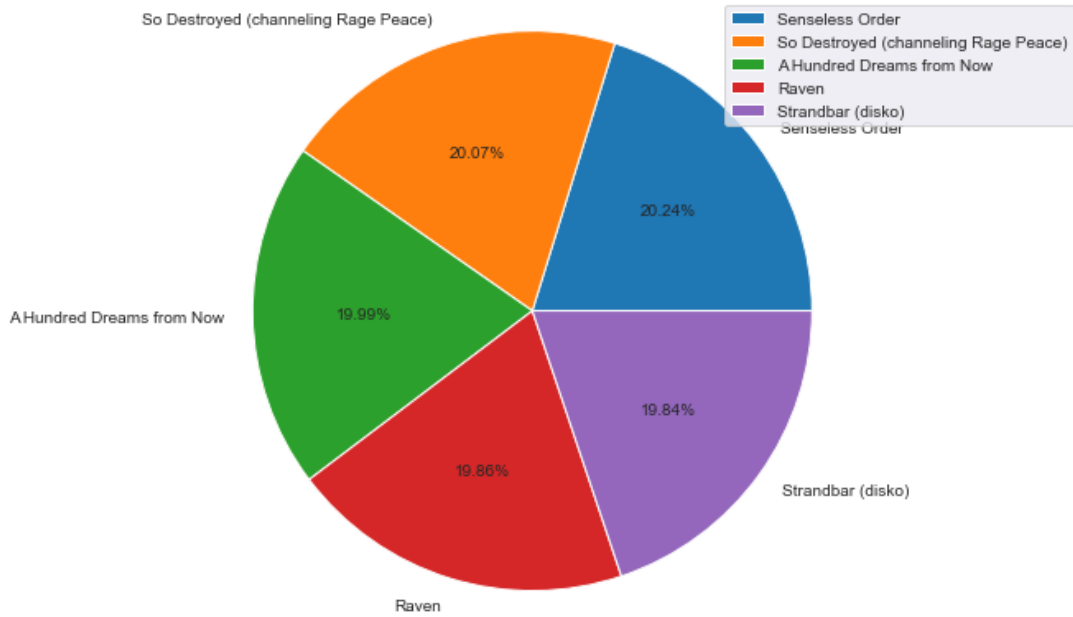


```
[19]: data = top_ten_instrumental_tracks["instrumentalness"]
```

#### 1.4 Top 5 instrumental songs - visualizing via pie chart-

```
[20]: plt.figure(figsize=(12, 7))
plt.pie(data, autopct='%1.2f%%', labels = top_ten_instrumental_tracks.song_title)
plt.axis('equal')
plt.legend(top_ten_instrumental_tracks.song_title, loc="best")
plt.show()
```





## 1.5 Multiple feature plot-

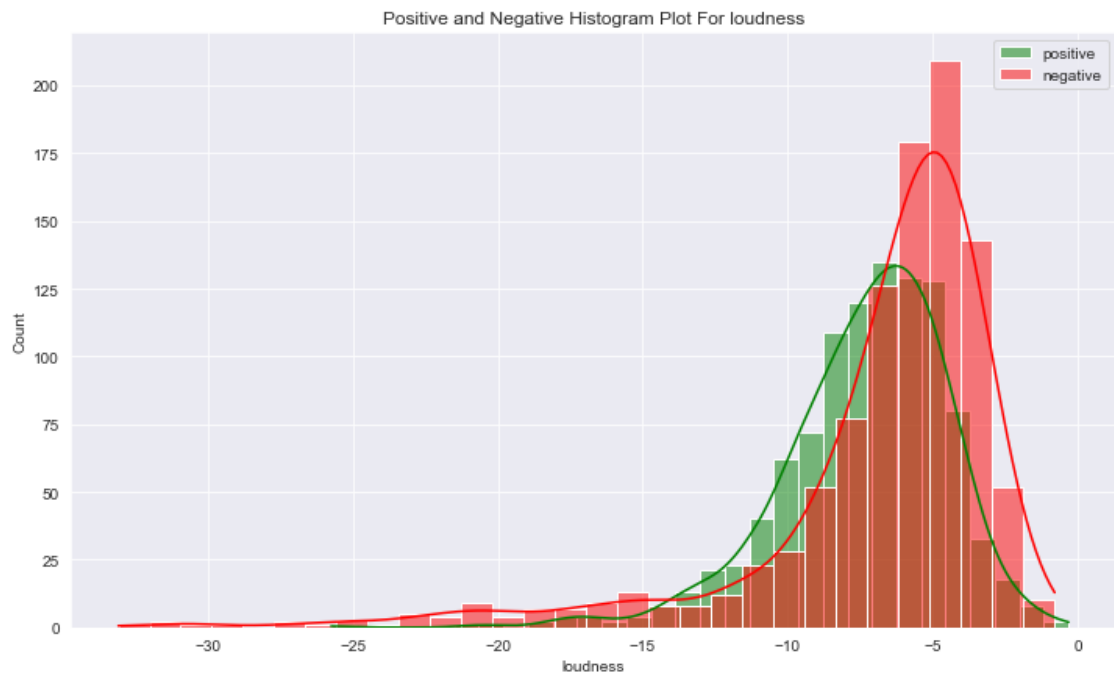
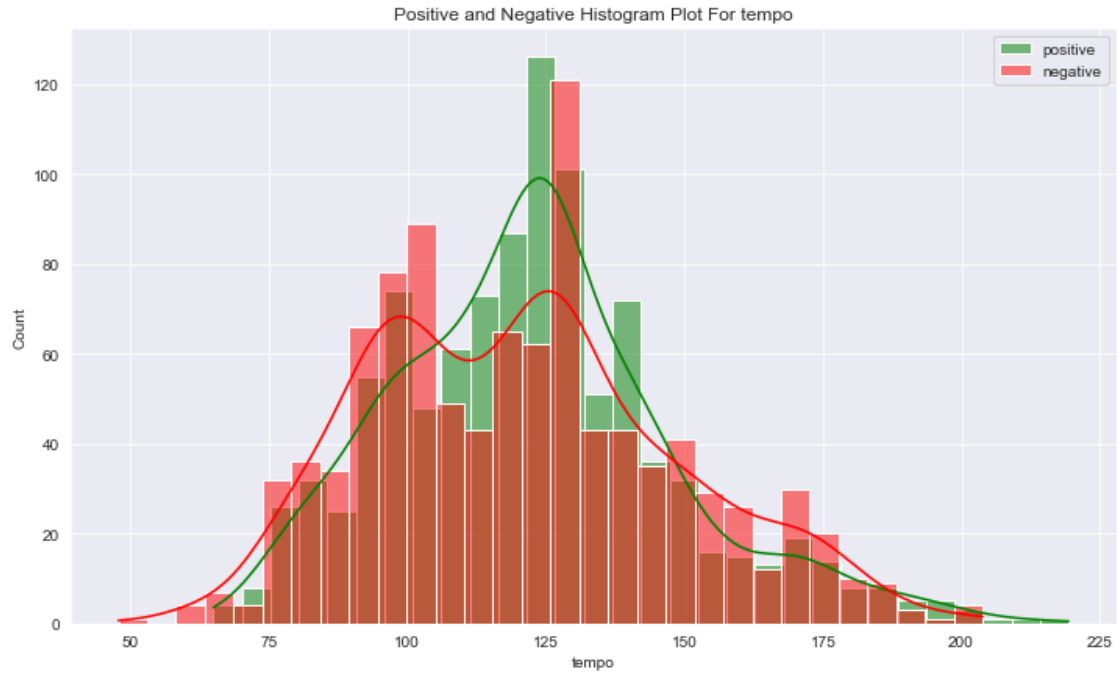
```
[21]: interest_feature_cols = [
    ↪ ["tempo", "loudness", "acousticness", "danceability", "duration_ms", "energy", "instrumentalness",
    "liveness", "speechiness", "new_valence"]
```

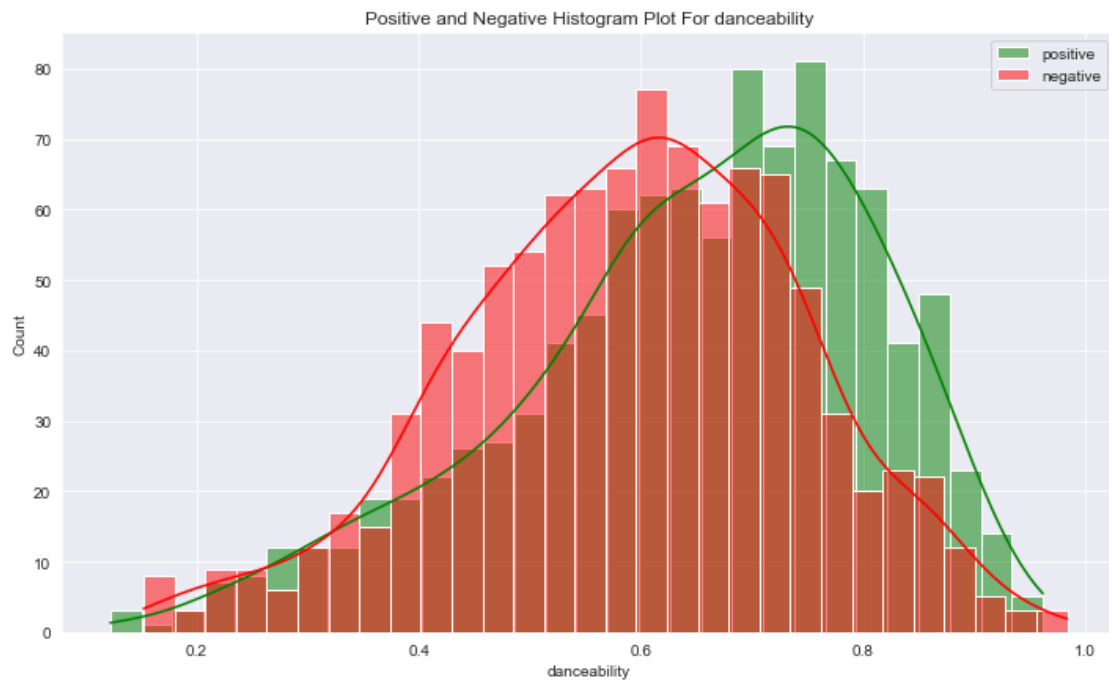
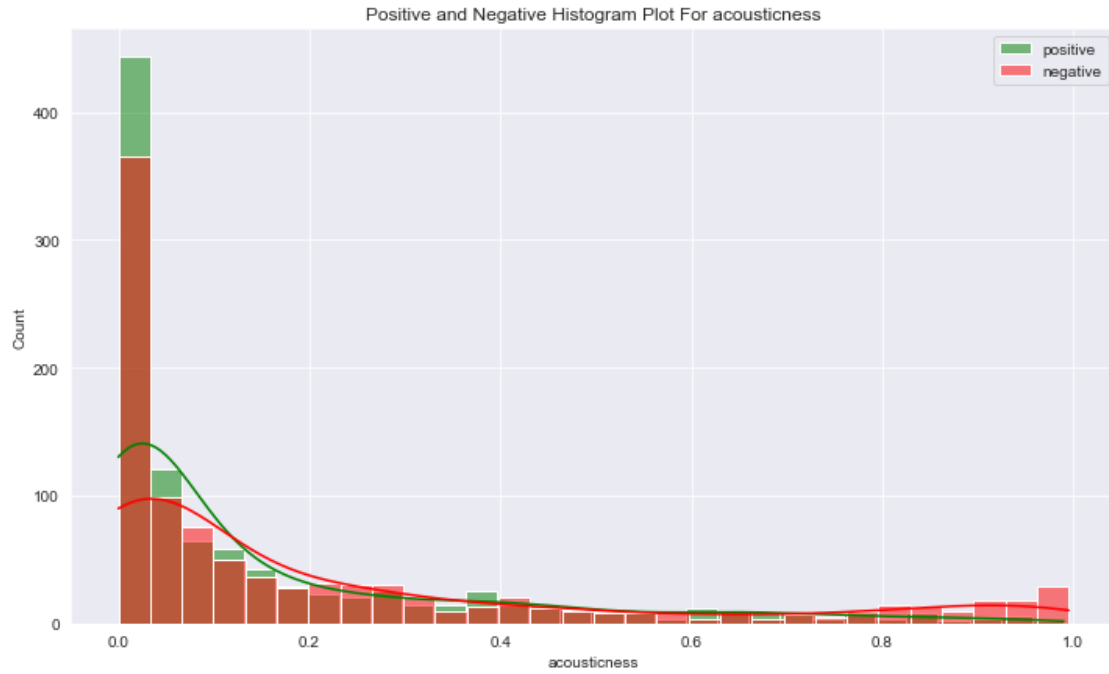
```
[22]: for feature_col in interest_feature_cols:
    pos_data = df[df["target"] == 1][feature_col]
    neg_data = df[df["target"] == 0][feature_col]

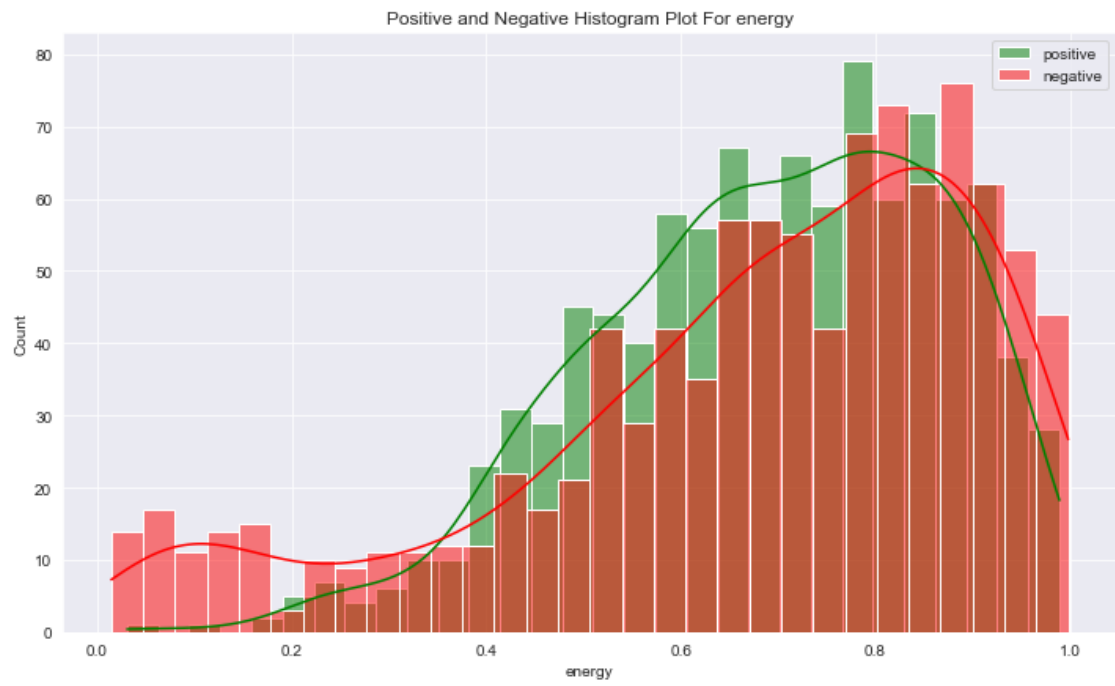
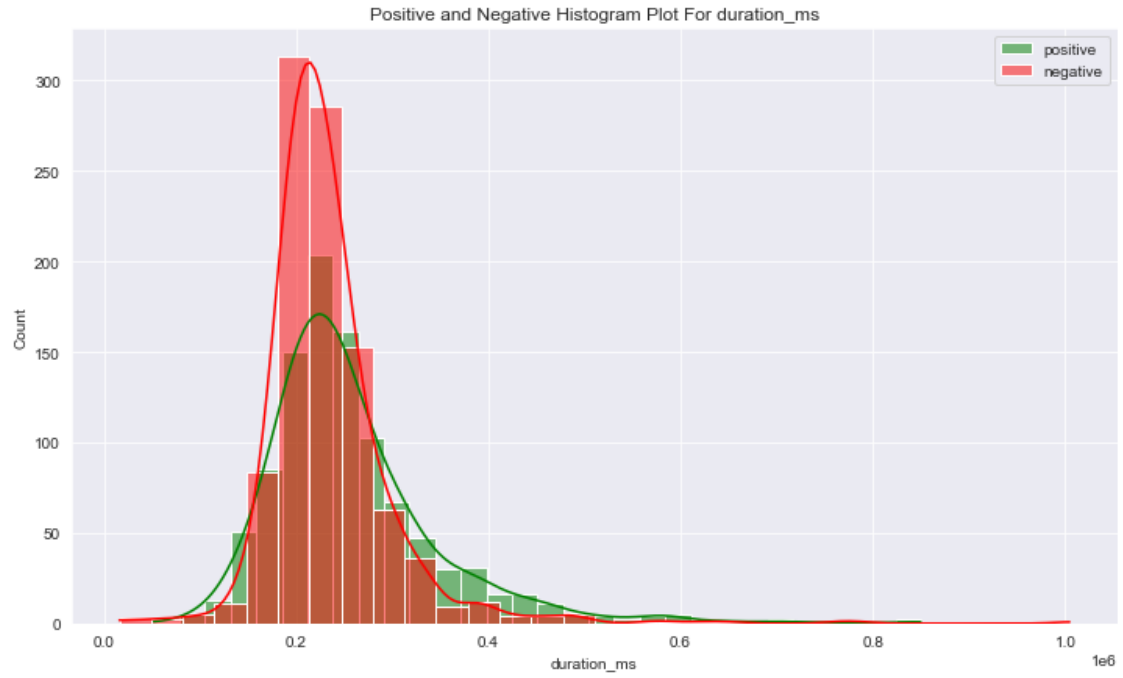
    plt.figure(figsize=(12, 7))

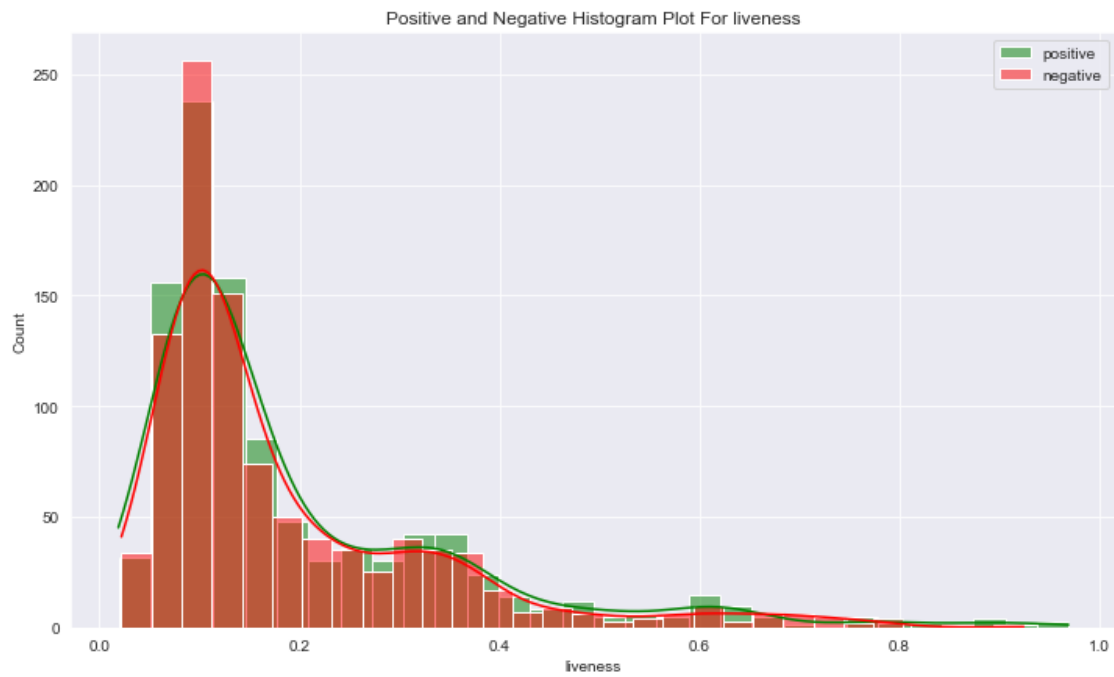
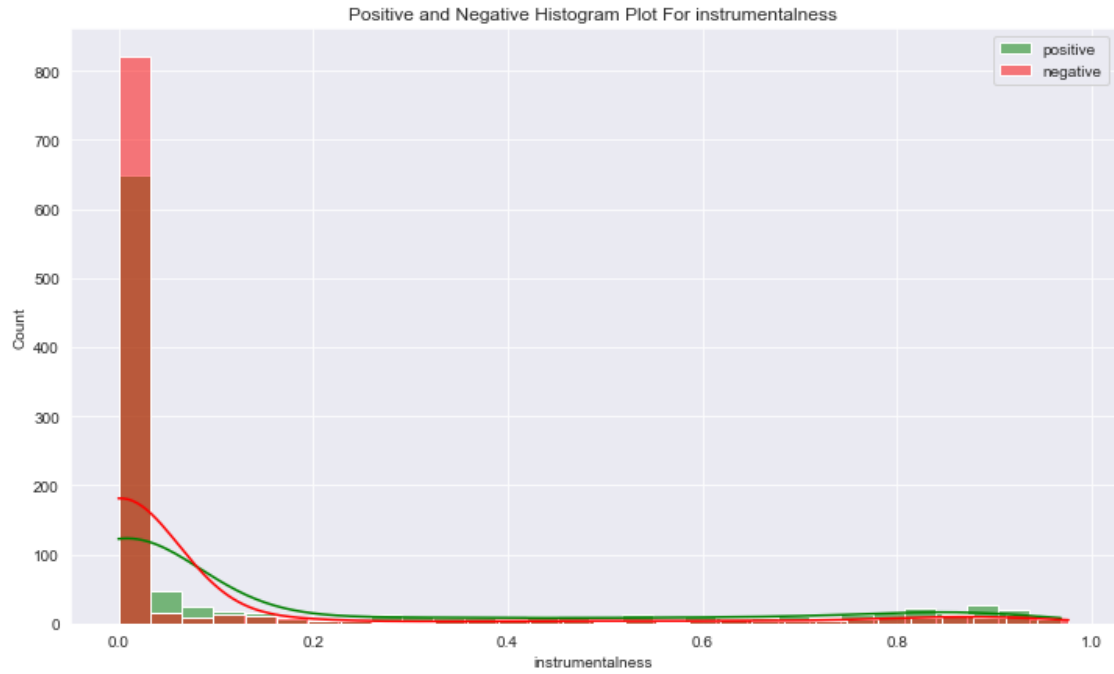
    sns.histplot(pos_data, bins=30, label="positive", color="green", kde=True)
    sns.histplot(neg_data, bins=30, label="negative", color="red", kde=True)

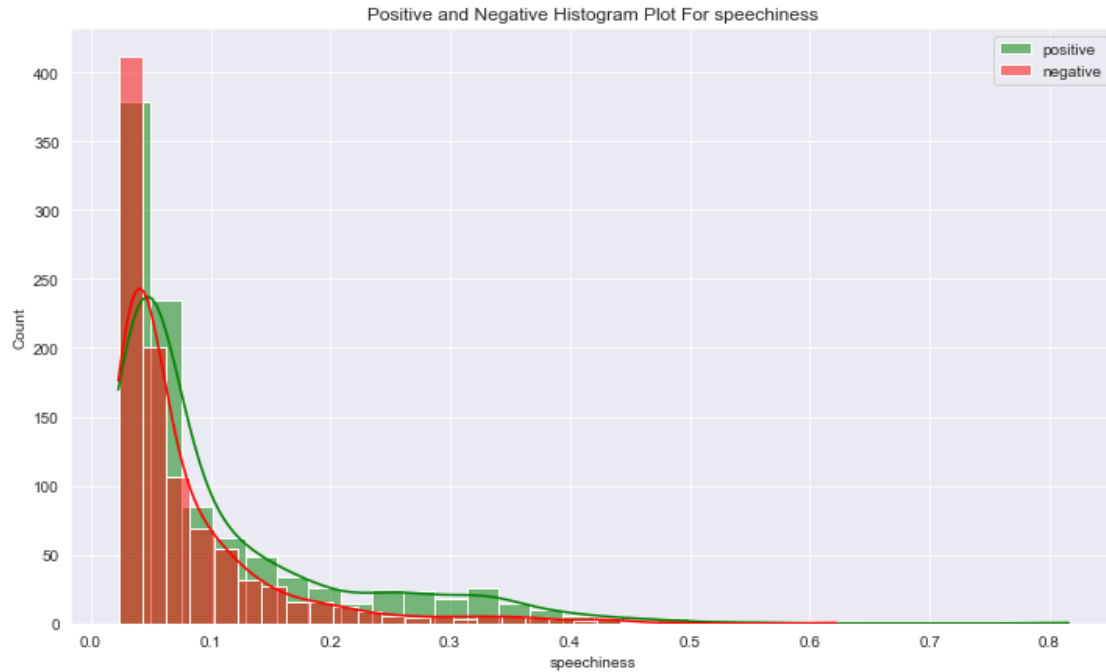
    plt.legend(loc="upper right")
    plt.title(f"Positive and Negative Histogram Plot For {feature_col}")
    plt.show()
```











```

-----
KeyError                                Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3629, in Index.
    ↪ get_loc(self, key, method, tolerance)
    3628 try:
-> 3629     return self._engine.get_loc(casted_key)
    3630 except KeyError as err:

File ~\anaconda3\lib\site-packages\pandas\_libs\index.pyx:136, in pandas._libs.
    ↪ index.IndexEngine.get_loc()

File ~\anaconda3\lib\site-packages\pandas\_libs\index.pyx:163, in pandas._libs.
    ↪ index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:5198, in pandas._libs.hashtable.
    ↪ PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:5206, in pandas._libs.hashtable.
    ↪ PyObjectHashTable.get_item()

KeyError: 'new_valence'

```

The above exception was the direct cause of the following exception:

```

KeyError                                Traceback (most recent call last)

```

```

Input In [22], in <cell line: 1>()
      1 for feature_col in interest_feature_cols:
----> 2     pos_data = df[df["target"] == 1][feature_col]
      3     neg_data = df[df["target"] == 0][feature_col]
      5     plt.figure(figsize=(12, 7))

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:3505, in DataFrame.
-> __getitem__(self, key)
      3503 if self.columns.nlevels > 1:
      3504     return self._getitem_multilevel(key)
-> 3505 indexer = self.columns.get_loc(key)
      3506 if is_integer(indexer):
      3507     indexer = [indexer]

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3631, in Index.
-> get_loc(self, key, method, tolerance)
      3629     return self._engine.get_loc(casted_key)
      3630 except KeyError as err:
-> 3631     raise KeyError(key) from err
      3632 except TypeError:
      3633     # If we have a listlike key, _check_indexing_error will raise
      3634     # InvalidIndexError. Otherwise we fall through and re-raise
      3635     # the TypeError.
      3636     self._check_indexing_error(key)

KeyError: 'new_valence'

```

## 1.6 Top 10 energetic tracks?

```

[24]: top_ten_energetic_tracks = df[["energy", "song_title", "artist"]].
      ↪sort_values(by="energy", ascending = True)[:10]
top_ten_energetic_tracks

```

```

[24]:      energy      song_title \
1594  0.0148      Lyric Pieces, Book I Op. 12: I. Arietta
1595  0.0156  String Quartet No. 4 in C Major, D. 46: II. An...
1537  0.0161      Blue in Green
1598  0.0230  Piano Quartet in E flat, Op.47: 3. Andante can...
1596  0.0288      8 Fantasiestücke, Op.12 : 1. Des Abends
1545  0.0291      Blue and Sentimental
1530  0.0295      I'm a Fool to Want You
1531  0.0302      I Was So Young, and You Were So Beautiful
817   0.0310  Mozart: Requiem in D Minor, K. 626: VIII. Lacr...
1876  0.0347      Nocturne No.1 In B Flat Minor, Op.9 No.1

      artist

```

```

1594      Edvard Grieg
1595      Franz Schubert
1537      Miles Davis
1598      Robert Schumann
1596      Robert Schumann
1545      Julian Dash
1530      Passport Quartet
1531      Bill Charlap
817   Nikolaus Harnoncourt
1876      Frédéric Chopin

```

## 1.7 most popular artist?

```

[25]: artist_counts = df["artist"].value_counts()
      most_popular_artist = artist_counts.idxmax()
      song_count = artist_counts.max()
      print(f"The most popular artist is {most_popular_artist} with {song_count}
      ↪songs.")

```

The most popular artist is Drake with 16 songs.

```

[26]: df.head()

```

```

[26]:   acoustictness  danceability  duration_ms  energy  instrumentalness  key  \
0         0.0102         0.833      204600    0.434           0.021900    2
1         0.1990         0.743      326933    0.359           0.006110    1
2         0.0344         0.838      185707    0.412           0.000234    2
3         0.6040         0.494      199413    0.338           0.510000    5
4         0.1800         0.678      392893    0.561           0.512000    5

      liveness  loudness  mode  speechiness  tempo  time_signature  valence  \
0     0.1650    -8.795     1      0.4310  150.062           4.0    0.286
1     0.1370   -10.401     1      0.0794  160.083           4.0    0.588
2     0.1590    -7.148     1      0.2890   75.044           4.0    0.173
3     0.0922   -15.236     1      0.0261   86.468           4.0    0.230
4     0.4390   -11.648     0      0.0694  174.004           4.0    0.904

      target  song_title  artist
0         1      Mask Off    Future
1         1      Redbone  Childish Gambino
2         1   Xanny Family    Future
3         1  Master Of None    Beach House
4         1  Parallel Lines    Junior Boys

```



## 1.8 Most Common Song or most trending song?

```
[27]: common_duration = df["duration_ms"].value_counts().idxmax()
      most_common_song = df[df["duration_ms"] == common_duration]["song_title"].
      ↪iloc[0]
      print("Most Common Duration:", common_duration, "seconds")
      print("Top 1 Song with this Duration:", most_common_song)
```

Most Common Duration: 192000 seconds

Top 1 Song with this Duration: Kerosene

## 1.9 top 10 tracks with most valence

```
[30]: top_10_tracks = df[["song_title", "artist"]].sort_values(by="new_valence",
      ↪ascending=False)[:10]
      top_10_tracks
```

```
-----
KeyError                                Traceback (most recent call last)
Input In [30], in <cell line: 1>()
----> 1 top_10_tracks =
      ↪df[["song_title", "artist"]].sort_values(by="new_valence", ascending=False)[:
      ↪10]
      2 top_10_tracks

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:311, in
      ↪deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args,
      ↪**kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:6322, in DataFrame.
      ↪sort_values(self, by, axis, ascending, inplace, kind, na_position,
      ↪ignore_index, key)
    6318 elif len(by):
    6319     # len(by) == 1
    6321     by = by[0]
-> 6322     k = self._get_label_or_level_values(by, axis=axis)
    6324     # need to rewrap column in Series to apply key function
    6325     if key is not None:
    6326         # error: Incompatible types in assignment (expression has type
    6327         # "Series", variable has type "ndarray")
```

```

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:1840, in NDFrame.
    ↪ _get_label_or_level_values(self, key, axis)
    1838     values = self.axes[axis].get_level_values(key)._values
    1839 else:
-> 1840     raise KeyError(key)
    1842 # Check for duplicates
    1843 if values.ndim > 1:

KeyError: 'new_valence'

```

## 2 handling ERRORS

```

[ ]: # Rename the 'valence' column to 'new_valence' in the DataFrame containing
    ↪ 'valence'
df.rename(columns={'valence': 'new_valence'}, inplace=True)

```

```
[32]: df.columns
```

```

[32]: Index(['acousticness', 'danceability', 'duration_ms', 'energy',
            'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
            'speechiness', 'tempo', 'time_signature', 'new_valence', 'target',
            'song_title', 'artist'],
           dtype='object')

```

```

[33]: # Sort the DataFrame by the "new_valence" column in descending order
top_ten_tracks = df.sort_values(by="new_valence",
    ↪ ascending=False)[["song_title", "new_valence"]][:5]

# Display the top ten tracks
print(top_ten_tracks)

```

	song_title	new_valence
460	Abataka - Original Mix	0.992
912	I'm Walkin' - 2002 Digital Remaster	0.975
1966	To Roz Bikini (Itsy, Bitsy, Teenie, Weenie)	0.974
207	Look at You	0.973
48	Azon de ma gnin kpevi	0.973

```
[34]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2017 entries, 0 to 2016
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   acousticness          2017 non-null   float64
1   danceability           2017 non-null   float64

```

```

2  duration_ms      2017 non-null   int64
3  energy           2017 non-null   float64
4  instrumentalness 2017 non-null   float64
5  key              2017 non-null   int64
6  liveness         2017 non-null   float64
7  loudness         2017 non-null   float64
8  mode             2017 non-null   int64
9  speechiness      2017 non-null   float64
10 tempo            2017 non-null   float64
11 time_signature   2017 non-null   float64
12 new_valence      2017 non-null   float64
13 target           2017 non-null   int64
14 song_title       2017 non-null   object
15 artist           2017 non-null   object
dtypes: float64(10), int64(4), object(2)
memory usage: 252.2+ KB

```

### 3 most trending genre?

```

[35]: import pandas as pd

# Assuming you have already loaded your data into a DataFrame named 'df'
# Define a function to apply the DAX formula to each row
def calculate_genre(row):
    if row['danceability'] >= 0.7 and row['energy'] >= 0.7:
        return "High Energy Dance"
    elif row['acousticness'] >= 0.7:
        return "Low Energy Acoustic"
    elif row['new_valence'] >= 0.7:
        return "Happy Pop"
    elif row['new_valence'] < 0.3:
        return "Sad Pop"
    elif row['instrumentalness'] >= 0.5:
        return "Instrumental"
    elif row['speechiness'] >= 0.5:
        return "Spoken Word"
    elif row['tempo'] > 120:
        return "Upbeat"
    else:
        return "Other"

# Apply the DAX formula to create the 'genre' column
df['genre'] = df.apply(calculate_genre, axis=1)

```

```

[36]: df.columns

```

```
[36]: Index(['acousticness', 'danceability', 'duration_ms', 'energy',
          'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
          'speechiness', 'tempo', 'time_signature', 'new_valence', 'target',
          'song_title', 'artist', 'genre'],
          dtype='object')
```

### 3.1 Merging the current dataset with the genre dataset which i created using DAX func on POWER BI

```
[37]: import pandas as pd

# Specify the file path to your Excel file
xlsx_file_path = r'C:\Users\amitm\Desktop\Spotify_EDA\COUNT OF TRACKS BY GENRE.
↳xlsx'

# Load the Excel data into a DataFrame
df_power_bi = pd.read_excel(xlsx_file_path)

# Check the first few rows of the DataFrame to ensure it loaded correctly
print(df_power_bi.head())
```

	genre	song_title
0	Upbeat	407
1	Sad Pop	391
2	Other	339
3	Happy Pop	324
4	High Energy Dance	310

```
[38]: # Merge the two DataFrames on the 'genre' column
merged_df = df.merge(df_power_bi, on='genre', how='left')
```

```
[39]: print(merged_df.shape)
```

(2017, 18)

```
[40]: print(merged_df.head())
```

	acousticness	danceability	duration_ms	energy	instrumentalness	key	\
0	0.0102	0.833	204600	0.434	0.021900	2	
1	0.1990	0.743	326933	0.359	0.006110	1	
2	0.0344	0.838	185707	0.412	0.000234	2	
3	0.6040	0.494	199413	0.338	0.510000	5	
4	0.1800	0.678	392893	0.561	0.512000	5	

	liveness	loudness	mode	speechiness	tempo	time_signature	\
0	0.1650	-8.795	1	0.4310	150.062	4.0	
1	0.1370	-10.401	1	0.0794	160.083	4.0	
2	0.1590	-7.148	1	0.2890	75.044	4.0	
3	0.0922	-15.236	1	0.0261	86.468	4.0	

```

4      0.4390   -11.648      0      0.0694  174.004      4.0

      new_valence  target      song_title_x      artist      genre \
0      0.286      1      Mask Off      Future      Sad Pop
1      0.588      1      Redbone      Childish Gambino      Upbeat
2      0.173      1      Xanny Family      Future      Sad Pop
3      0.230      1      Master Of None      Beach House      Sad Pop
4      0.904      1      Parallel Lines      Junior Boys      Happy Pop

      song_title_y
0      391
1      407
2      391
3      391
4      324

```

```

[41]: # Define the criteria and corresponding genre labels
criteria = [
    (merged_df['tempo'] > 120),
    (merged_df['danceability'] >= 0.7) & (merged_df['energy'] >= 0.7),
    (merged_df['acousticness'] >= 0.7),
    (merged_df['new_valence'] >= 0.7),
    (merged_df['new_valence'] < 0.3),
    (merged_df['instrumentalness'] >= 0.5),
    (merged_df['speechiness'] >= 0.5)
]

genre_labels = [
    "Upbeat",
    "High Energy Dance",
    "Low Energy Acoustic",
    "Happy Pop",
    "Sad Pop",
    "Instrumental",
    "Spoken Word"
]

# Create a new column 'popularity_score' based on the criteria and labels
merged_df['popularity_score'] = np.select(criteria, genre_labels,
    ↪default="Other")

# Count the occurrences of each genre in 'popularity_score' to find the most
    ↪trending genre
most_trending_genre = merged_df['popularity_score'].value_counts().idxmax()

print("Most Trending Genre:", most_trending_genre)

```

Most Trending Genre: Upbeat

```
[42]: # List all the column names in the DataFrame
column_names = df.columns.tolist()

# Print the list of column names
print("Column Names:")
for column_name in column_names:
    print(column_name)
```

```
Column Names:
acousticness
danceability
duration_ms
energy
instrumentalness
key
liveness
loudness
mode
speechiness
tempo
time_signature
new_valence
target
song_title
artist
genre
```

```
[43]: # Define the criteria and corresponding genre labels
criteria = [
    (merged_df['tempo'] > 120),
    (merged_df['danceability'] >= 0.7) & (merged_df['energy'] >= 0.7),
    (merged_df['acousticness'] >= 0.7),
    (merged_df['new_valence'] >= 0.7),
    (merged_df['new_valence'] < 0.3),
    (merged_df['instrumentalness'] >= 0.5),
    (merged_df['speechiness'] >= 0.5)
]

genre_labels = [
    "Upbeat",
    "High Energy Dance",
    "Low Energy Acoustic",
    "Happy Pop",
    "Sad Pop",
    "Instrumental",
    "Spoken Word"
]
```

```

# Create a new column 'popularity_score' based on the criteria and labels
merged_df['popularity_score'] = np.select(criteria, genre_labels,
↳ default="Other")

# Count the occurrences of each genre in 'popularity_score' to find the most
↳ trending genre
most_trending_genre = merged_df['popularity_score'].value_counts().idxmax()

print("Most Trending Genre:", most_trending_genre)

```

Most Trending Genre: Upbeat

[44]: `pip install nbconvert[webpdf]`

```

Requirement already satisfied: nbconvert[webpdf] in
c:\users\amitm\anaconda3\lib\site-packages (6.4.4)
Requirement already satisfied: traitlets>=5.0 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (5.1.1)
Requirement already satisfied: pygments>=2.4.1 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (2.11.2)
Requirement already satisfied: mistune<2,>=0.8.1 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (0.8.4)
Requirement already satisfied: bleach in c:\users\amitm\anaconda3\lib\site-
packages (from nbconvert[webpdf]) (4.1.0)
Requirement already satisfied: nbformat>=4.4 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (5.3.0)
Requirement already satisfied: defusedxml in c:\users\amitm\anaconda3\lib\site-
packages (from nbconvert[webpdf]) (0.7.1)
Requirement already satisfied: pandocfilters>=1.4.1 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (1.5.0)
Requirement already satisfied: testpath in c:\users\amitm\anaconda3\lib\site-
packages (from nbconvert[webpdf]) (0.5.0)
Requirement already satisfied: jupyter-core in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (4.9.2)
Requirement already satisfied: beautifulsoup4 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (4.11.1)
Requirement already satisfied: jupyterlab-pygments in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (0.1.2)
Requirement already satisfied: entrypoints>=0.2.2 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (0.4)
Requirement already satisfied: Jinja2>=2.4 in c:\users\amitm\anaconda3\lib\site-
packages (from nbconvert[webpdf]) (2.11.3)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (0.5.13)
Requirement already satisfied: pyppeteer<1.1,>=1 in
c:\users\amitm\anaconda3\lib\site-packages (from nbconvert[webpdf]) (1.0.2)
Requirement already satisfied: MarkupSafe>=0.23 in
c:\users\amitm\anaconda3\lib\site-packages (from Jinja2>=2.4->nbconvert[webpdf])

```

(2.0.1)

Requirement already satisfied: jupyter-client>=6.1.5 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (6.1.12)

Requirement already satisfied: nest-asyncio in  
c:\users\amitm\anaconda3\lib\site-packages (from  
nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (1.5.5)

Requirement already satisfied: tornado>=4.1 in  
c:\users\amitm\anaconda3\lib\site-packages (from jupyter-  
client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (6.1)

Requirement already satisfied: python-dateutil>=2.1 in  
c:\users\amitm\anaconda3\lib\site-packages (from jupyter-  
client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (2.8.2)

Requirement already satisfied: pyzmq>=13 in c:\users\amitm\anaconda3\lib\site-  
packages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[webpdf])  
(22.3.0)

Requirement already satisfied: pywin32>=1.0 in  
c:\users\amitm\anaconda3\lib\site-packages (from jupyter-  
core->nbconvert[webpdf]) (302)

Requirement already satisfied: jsonschema>=2.6 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
nbformat>=4.4->nbconvert[webpdf]) (4.4.0)

Requirement already satisfied: fastjsonschema in  
c:\users\amitm\anaconda3\lib\site-packages (from  
nbformat>=4.4->nbconvert[webpdf]) (2.15.1)

Requirement already satisfied: pypersistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
jsonschema>=2.6->nbformat>=4.4->nbconvert[webpdf]) (0.18.0)

Requirement already satisfied: attrs>=17.4.0 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
jsonschema>=2.6->nbformat>=4.4->nbconvert[webpdf]) (21.4.0)

Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
pypeteer<1.1,>=1->nbconvert[webpdf]) (1.4.4)

Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
pypeteer<1.1,>=1->nbconvert[webpdf]) (4.64.0)

Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
pypeteer<1.1,>=1->nbconvert[webpdf]) (1.26.9)

Requirement already satisfied: websockets<11.0,>=10.0 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
pypeteer<1.1,>=1->nbconvert[webpdf]) (10.4)

Requirement already satisfied: pyee<9.0.0,>=8.1.0 in  
c:\users\amitm\anaconda3\lib\site-packages (from  
pypeteer<1.1,>=1->nbconvert[webpdf]) (8.2.2)

Requirement already satisfied: importlib-metadata>=1.4 in  
c:\users\amitm\anaconda3\lib\site-packages (from



```

pyppeteer<1.1,>=1->nbconvert[webpdf]) (4.11.3)
Requirement already satisfied: certifi>=2021 in
c:\users\amitm\anaconda3\lib\site-packages (from
pyppeteer<1.1,>=1->nbconvert[webpdf]) (2021.10.8)
Requirement already satisfied: zipp>=0.5 in c:\users\amitm\anaconda3\lib\site-
packages (from importlib-metadata>=1.4->pyppeteer<1.1,>=1->nbconvert[webpdf])
(3.7.0)
Requirement already satisfied: six>=1.5 in c:\users\amitm\anaconda3\lib\site-
packages (from python-dateutil>=2.1->jupyter-
client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (1.16.0)
Requirement already satisfied: colorama in c:\users\amitm\anaconda3\lib\site-
packages (from tqdm<5.0.0,>=4.42.1->pyppeteer<1.1,>=1->nbconvert[webpdf])
(0.4.4)
Requirement already satisfied: soupsieve>1.2 in
c:\users\amitm\anaconda3\lib\site-packages (from
beautifulsoup4->nbconvert[webpdf]) (2.3.1)
Requirement already satisfied: packaging in c:\users\amitm\anaconda3\lib\site-
packages (from bleach->nbconvert[webpdf]) (21.3)
Requirement already satisfied: webencodings in
c:\users\amitm\anaconda3\lib\site-packages (from bleach->nbconvert[webpdf])
(0.5.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
c:\users\amitm\anaconda3\lib\site-packages (from
packaging->bleach->nbconvert[webpdf]) (3.0.4)
Note: you may need to restart the kernel to use updated packages.

```

[ ]: