Data Processing of song popularity project

In [ ]:
```
import pandas as pd
import seaborn as sns
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

Overview of project

Our project aims to use both audio features and social media marketing features to predict popularity of a song. We would hence need first a dataset that determines popularity of a song. In our case we chose a binary classification type (is popular or not). We decided to use the infamous Billboard. Our primary focus revolves around identifying whether a song has ever reached the esteemed Global Hot 100 chart, a vital indicator of its popularity. This choice was based on the fact that Billboard really takes everything into account for the popularity of a song(Online streaming, physical CD sales, Concerts, music video, social media , radio etc…). This seemed to be the best accessible indicator of a song's popularity. Now for the independent variables, we choose sets of data from various API sources(Spotify, Youtube and Billboard). For spotify, we are getting the audio features like (Tempo, danceability etc…), For Youtube, we are getting music video features like (Like count, view count etc…). The combination of all of these variables is justified by the versatility of the factors that makes a song popular. The song itself only will not determine its popularity.

In the following notebook, we are going to take you through the steps of data cleaning, data integretation and data analysis.

Data acquisition

Our data originates from multiple sources: Billboard API, Spotify API, and Youtube API.

With all of these tools, it took us long time to retrieve data due to many API access issues and quota restrictions. We retrieved data for Popular songs and added features from Spotify(fully) and Youtube(still haversting the data, since daily quota is limited to 10,000 units).

Since the data is hardly accessible, we retrieved it over days and kept storing it in a csv file for our convenience. Here is the final raw dataset with popular songs:

In [ ]:
```
popular=pd.read_csv('/content/gdrive/MyDrive/Capstone/popular_song.csv')
popular
```

Out[ ]:

	Title	Artist	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness	...	tempo	type
0	Bad Day	Daniel Powter	1	0.599	0.785	3.0	-4.013	1.0	0.0309	0.44800	...	140.046	audio_features
1	Temperature	Sean Paul	1	0.951	0.600	0.0	-4.675	0.0	0.0685	0.10600	...	125.040	audio_features
2	Promiscuous	Nelly Furtado Featuring Timbaland	1	0.808	0.970	10.0	-6.098	0.0	0.0506	0.05690	...	114.328	audio_features
3	You're Beautiful	James Blunt	1	0.675	0.479	0.0	-9.870	0.0	0.0278	0.63300	...	81.998	audio_features
4	Hips Don't Lie	Shakira Featuring Wyclef Jean	1	0.778	0.824	10.0	-5.892	0.0	0.0707	0.28400	...	100.024	audio_features
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1519	Flower Shops	ERNEST Featuring Morgan Wallen	1	0.527	0.461	7.0	-5.908	1.0	0.0269	0.11800	...	128.153	audio_features
1520	To The Moon!	JNR CHOI & Sam Tompkins	1	0.745	0.650	2.0	-11.814	1.0	0.3460	0.04510	...	144.047	audio_features
1521	Unholy	Sam Smith & Kim Petras	1	0.714	0.472	2.0	-7.375	1.0	0.0864	0.01300	...	131.121	audio_features
1522	One Mississippi	Kane Brown	1	0.471	0.846	0.0	-5.269	1.0	0.0389	0.00279	...	100.089	audio_features
1523	Circles Around This Town	Maren Morris	1	0.591	0.814	4.0	-4.986	1.0	0.0468	0.01500	...	149.900	audio_features

1524 rows × 23 columns

The table includes following columns:

Title, Artist, Track_id, Popular (0/1), Danceability, Energy, Key, Loudness, Mode, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Tempo, Duration_ms, Time_signature.

Data Cleaning

Our data was pretty clean, yet when requesting it from APIs we encountered a problem of missing data and duplicates. We had to deal with it in our former step of data acquisition, because collecting this data caused some errors with API and continuously stopped us from getting all the needed data.

Some steps that were done to deal with both problems:

1. Missing data occurred due to some inconsistencies of how the songs data was stored on Spotify (names or artist names were missing, some problem with ids). This was handled by considering an exception for the tracks that had track name or id missing, by skippping the row and continuing to retrieve other tracks.

2. Duplicates problem occurred on both Youtube and Billboard. For Billboard, it happened because some tracks made it to top 100,200 several years in a row. To avoid that we have to check if the song was already added to the popular set or not, and disregard it if it is there. For YouTube, the duplicates occurres because there were numerous videos posted with the same song. To keep it consistent, we limited each song to the first entry on YouTube (since it is the most popular one), which usually was an official video of the song.

Data Integration

To get the dataset above, we used different sources as mentioned earlier.

First, we collected data from the Billboard Top hot-100 from 2008 to 2022 (the biggest range for available data): We collected the track name and artist name from there:

In [ ]:
```
billboard=popular=pd.read_csv('/content/gdrive/MyDrive/Capstone/billboard.csv')
billboard
```

Out[ ]:

	Title	Artist
0	Bad Day	Daniel Powter
1	Temperature	Sean Paul
2	Promiscuous	Nelly Furtado Featuring Timbaland
3	You're Beautiful	James Blunt
4	Hips Don't Lie	Shakira Featuring Wyclef Jean
...	...	...
1519	Flower Shops	ERNEST Featuring Morgan Wallen
1520	To The Moon!	JNR CHOI & Sam Tompkins
1521	Unholy	Sam Smith & Kim Petras
1522	One Mississippi	Kane Brown
1523	Circles Around This Town	Maren Morris

1524 rows × 2 columns

The next step was to collect audio features for all the popular songs we got from the billboard API.

In here, we will use the audio features provided by Spotify. We use the search endpoint API by inputng the name of the artist and name of the song. Then we extract the id of the track.

From that point, we take that id and use the second end point that is the audio feature. we collected the following: danceability, energy, key, loudness, mode, speechiness, acousticness, tempo, type, id, uri, track_href, analysis_url, duration_ms, time_signature, track_id, index.

We added these features to the existing billboard dataFrame to get the final Popular dataframe below:

In [ ]:
```
popular=pd.read_csv('/content/gdrive/MyDrive/Capstone/popular_song.csv')
popular['Popular']=1
popular
```

Out[ ]:

	Title	Artist	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness	...	tempo	type
0	Bad Day	Daniel Powter	1	0.599	0.785	3.0	-4.013	1.0	0.0309	0.44800	...	140.046	audio_features
1	Temperature	Sean Paul	1	0.951	0.600	0.0	-4.675	0.0	0.0685	0.10600	...	125.040	audio_features
2	Promiscuous	Nelly Furtado Featuring Timbaland	1	0.808	0.970	10.0	-6.098	0.0	0.0506	0.05690	...	114.328	audio_features
3	You're Beautiful	James Blunt	1	0.675	0.479	0.0	-9.870	0.0	0.0278	0.63300	...	81.998	audio_features
4	Hips Don't Lie	Shakira Featuring Wyclef Jean	1	0.778	0.824	10.0	-5.892	0.0	0.0707	0.28400	...	100.024	audio_features
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1519	Flower Shops	ERNEST Featuring Morgan Wallen	1	0.527	0.461	7.0	-5.908	1.0	0.0269	0.11800	...	128.153	audio_features
1520	To The Moon!	JNR CHOI & Sam Tompkins	1	0.745	0.650	2.0	-11.814	1.0	0.3460	0.04510	...	144.047	audio_features
1521	Unholy	Sam Smith & Kim Petras	1	0.714	0.472	2.0	-7.375	1.0	0.0864	0.01300	...	131.121	audio_features
1522	One Mississippi	Kane Brown	1	0.471	0.846	0.0	-5.269	1.0	0.0389	0.00279	...	100.089	audio_features
1523	Circles Around This Town	Maren Morris	1	0.591	0.814	4.0	-4.986	1.0	0.0468	0.01500	...	149.900	audio_features

1524 rows × 23 columns

For the unpopular set, we generated random characters and used them through our Search spotify API endPoint, we then recorded the name of the song, the name of the artist and the track id.

Now that we have all the songs with their id, we can use the audio features endpoint with the id.

We can finally pull the audio features and store them in the unpopular dataset. The final dataframe looks like this below:

In [ ]:

```
unpopular=pd.read_csv('/content/gdrive/MyDrive/Capstone/unpopular_song.csv')
unpopular['Popular']=0
unpopular
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	...	liveness
0	This Is How We Do It	Montell Jordan	6uQKuonTU8VKBz5SHZuQXD	0	0.799	0.6230	0.0	-9.374	1.0	0.0812	...	0.4300
1	Homecoming	Kanye West	4iz9lGMjU1lXS51oPmUmTe	0	0.667	0.7470	1.0	-7.059	1.0	0.1890	...	0.1150
2	Believer	Imagine Dragons	0pqnGHJpmpxLKifKRmU6WP	0	0.776	0.7800	10.0	-4.374	0.0	0.1280	...	0.0810
3	Biking	Frank Ocean	2q0VexHJirnUPnEOhr2DxK	0	0.673	0.4630	2.0	-7.247	1.0	0.1910	...	0.0907
4	Nuyorican Soul	Carlos Henriquez	0ZDBZpIt3eUJ6LaApLJSiB	0	0.548	0.6050	5.0	-10.184	1.0	0.0428	...	0.0939
...	...	...	...	...	...	...	...	...	...	...	...	...
1807	Sundress	A$AP Rocky	2aPTvyE09vUCRwVvj0I8WK	0	0.721	0.7070	6.0	-6.364	1.0	0.0595	...	0.1430
1808	Agora Hills	Doja Cat	7dJYggqjKo71KI9sLzqCs8	0	0.750	0.6740	8.0	-6.128	0.0	0.0970	...	0.1220
1809	gfg	Miguel	7xK1qc3jSIlo0UHah9WHdn	0	0.673	0.8970	2.0	-4.421	1.0	0.1300	...	0.2160
1810	UNA NOCHE EN MEDELLÍN - REMIX	KAROL G	6ejks4eS7DOoYW8hrpRcDV	0	0.843	0.7640	10.0	-2.494	0.0	0.0741	...	0.0448
1811	Light White Noise	Evomin	3qBMFORCRUKzBPiftvwaJn	0	0.222	0.0507	1.0	-44.323	1.0	0.0681	...	0.1120

1812 rows × 22 columns

Outlier Detection and Management, Handling Missing Data, Cleaned Dataset

The data we got is pretty clean, but we still need to do some exploration to make sure we do not have outliers, missing data and other issues to fix.

First let's look at the datasets overview:

In [ ]:
```
popular.columns
```

Out[ ]:
```
Index(['Title', 'Artist', 'Popular', 'danceability', 'energy', 'key',
       'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
       'liveness', 'valence', 'tempo', 'type', 'id', 'uri', 'track_href',
       'analysis_url', 'duration_ms', 'time_signature', 'track_id', 'index'],
      dtype='object')
```

In [ ]:
```
unpopular.columns
```

Out[ ]:
```
Index(['Title', 'Artist', 'track_id', 'Popular', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'type', 'id', 'uri',
       'track_href', 'analysis_url', 'duration_ms', 'time_signature'],
      dtype='object')
```

Let us see the description of the features that our datasets contain:

In [ ]:
```
print('popular----------------------------------')
print(popular['acousticness'].describe())
print('unpopular----------------------------------')
print(unpopular['acousticness'].describe())
print('popular----------------------------------')
print(popular['energy'].describe())
print('unpopular----------------------------------')
print(unpopular['energy'].describe())
```

```
popular----------------------------------
count    1523.000000
mean        0.170391
std         0.212809
min         0.000053
25%         0.019100
50%         0.082000
75%         0.241500
max         0.981000
Name: acousticness, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        0.293147
std         0.308823
min         0.000000
25%         0.031700
50%         0.173000
75%         0.488000
max         0.996000
Name: acousticness, dtype: float64
popular----------------------------------
count    1523.000000
mean        0.664275
std         0.166151
min         0.040000
25%         0.557000
50%         0.678000
75%         0.793500
max         0.983000
Name: energy, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        0.600834
std         0.237044
min         0.000288
25%         0.457000
50%         0.631000
75%         0.779000
max         1.000000
Name: energy, dtype: float64
```

In [ ]:
```
print('popular----------------------------------')
print(popular['key'].describe())
print('unpopular----------------------------------')
print(unpopular['key'].describe())
print('popular----------------------------------')
print(popular['loudness'].describe())
print('unpopular----------------------------------')
```

```
print(unpopular['loudness'].describe())
```

```
popular----------------------------------
count    1523.000000
mean        5.247538
std         3.629134
min         0.000000
25%         1.500000
50%         5.000000
75%         8.000000
max        11.000000
Name: key, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        5.250138
std         3.640164
min         0.000000
25%         2.000000
50%         5.000000
75%         8.000000
max        11.000000
Name: key, dtype: float64
popular----------------------------------
count    1523.000000
mean       -5.977584
std         2.176387
min       -18.071000
25%        -7.040500
50%        -5.634000
75%        -4.521000
max        -1.190000
Name: loudness, dtype: float64
unpopular----------------------------------
count    1807.000000
mean       -9.049765
std         6.494373
min       -45.434000
25%       -10.246000
50%        -7.227000
75%        -5.309000
max         0.326000
Name: loudness, dtype: float64
```

In [ ]:

```
print('popular----------------------------------')
print(popular['mode'].describe())
print('unpopular----------------------------------')
print(unpopular['mode'].describe())
print('popular----------------------------------')
print(popular['instrumentalness'].describe())
print('unpopular----------------------------------')
print(unpopular['instrumentalness'].describe())
```

```
popular----------------------------------
count    1523.000000
mean        0.636901
std         0.481051
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: mode, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        0.589928
std         0.491983
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: mode, dtype: float64
popular----------------------------------
count    1523.000000
mean        0.014096
std         0.097801
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000009
max         0.945000
Name: instrumentalness, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        0.165479
std         0.321105
min         0.000000
25%         0.000000
50%         0.000021
75%         0.053700
max         0.999000
Name: instrumentalness, dtype: float64
```

In [ ]:

```
print('popular----------------------------------')
print(popular['liveness'].describe())
print('unpopular----------------------------------')
print(unpopular['liveness'].describe())
print('popular----------------------------------')
print(popular['speechiness'].describe())
print('unpopular----------------------------------')
print(unpopular['speechiness'].describe())
```

```
popular----------------------------------
count    1523.000000
mean        0.174657
std         0.128889
min         0.021000
25%         0.094050
50%         0.123000
75%         0.214500
max         0.851000
Name: liveness, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        0.180166
std         0.143101
min         0.026700
25%         0.096750
50%         0.121000
75%         0.215500
max         0.970000
Name: liveness, dtype: float64
popular----------------------------------
count    1523.000000
mean        0.103984
std         0.100577
min         0.023100
25%         0.039450
50%         0.058800
75%         0.126500
max         0.730000
Name: speechiness, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        0.118389
std         0.119306
min         0.000000
25%         0.040400
50%         0.063000
75%         0.151000
max         0.936000
Name: speechiness, dtype: float64
```

In [ ]:

```
print('popular----------------------------------')
print(popular['time_signature'].describe())
print('unpopular----------------------------------')
print(unpopular['time_signature'].describe())
print('popular----------------------------------')
print(popular['valence'].describe())
print('unpopular----------------------------------')
print(unpopular['valence'].describe())
```

```
popular----------------------------------
count    1523.000000
mean        3.978989
std         0.247627
min         1.000000
25%         4.000000
50%         4.000000
75%         4.000000
max         5.000000
Name: time_signature, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        3.919757
std         0.409849
min         0.000000
25%         4.000000
50%         4.000000
75%         4.000000
max         5.000000
Name: time_signature, dtype: float64
popular----------------------------------
count    1523.000000
mean        0.515465
std         0.220640
min         0.037200
25%         0.350000
50%         0.512000
75%         0.688000
max         0.969000
Name: valence, dtype: float64
unpopular----------------------------------
count    1807.000000
mean        0.479435
std         0.258979
min         0.000000
25%         0.266500
50%         0.487000
75%         0.686500
max         0.982000
Name: valence, dtype: float64
```

In [ ]:

```
print('popular----------------------------------')
print(popular['tempo'].describe())
print('unpopular----------------------------------')
print(unpopular['tempo'].describe())
print('popular----------------------------------')
print(popular['type'].describe())
print('unpopular----------------------------------')
print(unpopular['type'].describe())
print('popular----------------------------------')
print(popular['duration_ms'].describe())
print('unpopular----------------------------------')
print(unpopular['duration_ms'].describe())
```

```
popular----------------------------------
count    1523.000000
mean      121.651978
std        28.725218
min        51.316000
25%        98.029000
50%       120.462000
75%       140.039500
max       210.857000
Name: tempo, dtype: float64
unpopular----------------------------------
count    1807.000000
mean      121.495771
std        30.195706
min         0.000000
25%        97.994000
50%       120.125000
75%       140.905000
max       217.939000
Name: tempo, dtype: float64
popular----------------------------------
count               1523
unique                 1
top       audio_features
freq                1523
Name: type, dtype: object
unpopular----------------------------------
count               1807
unique                 1
top       audio_features
freq                1807
Name: type, dtype: object
popular----------------------------------
count      1523.000000
mean     218858.494419
std       43821.161731
min       36227.000000
25%      195033.500000
50%      216147.000000
75%      238802.500000
max      688453.000000
Name: duration_ms, dtype: float64
unpopular----------------------------------
count      1807.000000
mean     197192.415053
std       76146.069448
min       33103.000000
25%      150167.500000
50%      189978.000000
75%      231653.000000
max      785503.000000
Name: duration_ms, dtype: float64
```

One important step in cleaning data is removing the features that will not be needed for our analysis. Hence, we removed the following:

- uri(The Spotify URI for the track.)
- track_href(A link to the Web API endpoint providing full details of the track.)
- analysis_url(A URL to access the full audio analysis of this track. An access token is required to access this data.)
- type(The object type. Allowed values: "audio_features")

In [ ]:
```
popular=popular[["Title", "Artist", "track_id",'Popular', 'danceability', 'energy', 'key',
       'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
       'liveness', 'valence', 'tempo','duration_ms', 'time_signature' ]]
popular
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness
0	Bad Day	Daniel Powter	0mUyMawtxj1CJ76kn9gIZK	1	0.599	0.785	3.0	-4.013	1.0	0.0309	0.44800
1	Temperature	Sean Paul	0k2GOhqsrxDTAbFFSdNJjT	1	0.951	0.600	0.0	-4.675	0.0	0.0685	0.10600
2	Promiscuous	Nelly Furtado Featuring Timbaland	2gam98EZKrF9XuOkU13ApN	1	0.808	0.970	10.0	-6.098	0.0	0.0506	0.05690
3	You're Beautiful	James Blunt	0vg4WnUWvze6pBOJDTq99k	1	0.675	0.479	0.0	-9.870	0.0	0.0278	0.63300
4	Hips Don't Lie	Shakira Featuring Wyclef Jean	3ZFTkvIE7kyPt6Nu3PEa7V	1	0.778	0.824	10.0	-5.892	0.0	0.0707	0.28400
...	...	...	...	...	...	...	...	...	...	...	...
1519	Flower Shops	ERNEST Featuring Morgan Wallen	0De9jFjJ4eRLl7Yww2eBw1	1	0.527	0.461	7.0	-5.908	1.0	0.0269	0.11800
1520	To The Moon!	JNR CHOI & Sam Tompkins	5vUnjhBzRJJIAOJPde6zDx	1	0.745	0.650	2.0	-11.814	1.0	0.3460	0.04510
1521	Unholy	Sam Smith & Kim Petras	3nqQXoyQOWXiESFLlDF1hG	1	0.714	0.472	2.0	-7.375	1.0	0.0864	0.01300
1522	One Mississippi	Kane Brown	4FdPnT2cFrpWCmWZd7GXc3	1	0.471	0.846	0.0	-5.269	1.0	0.0389	0.00279
1523	Circles Around This Town	Maren Morris	13G5xv1wUKvJYbK0wYmioN	1	0.591	0.814	4.0	-4.986	1.0	0.0468	0.01500

1524 rows × 17 columns

In [ ]:

```
unpopular=unpopular[["Title", "Artist", "track_id",'Popular', 'danceability', 'energy', 'key',
       'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
       'liveness', 'valence', 'tempo','duration_ms', 'time_signature' ]]
unpopular
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness
0	This Is How We Do It	Montell Jordan	6uQKuonTU8VKBz5SHZuQXD	0	0.799	0.6230	0.0	-9.374	1.0	0.0812	0.0141
1	Homecoming	Kanye West	4iz9lGMjU1lXS51oPmUmTe	0	0.667	0.7470	1.0	-7.059	1.0	0.1890	0.3370
2	Believer	Imagine Dragons	0pqnGHJpmpxLKifKRmU6WP	0	0.776	0.7800	10.0	-4.374	0.0	0.1280	0.0622
3	Biking	Frank Ocean	2q0VexHJirnUPnEOhr2DxK	0	0.673	0.4630	2.0	-7.247	1.0	0.1910	0.6810
4	Nuyorican Soul	Carlos Henriquez	0ZDBZplt3eUJ6LaApLJSiB	0	0.548	0.6050	5.0	-10.184	1.0	0.0428	0.7080
...	...	...	...	...	...	...	...	...	...	...	...
1807	Sundress	A$AP Rocky	2aPTvyE09vUCRwVvj0I8WK	0	0.721	0.7070	6.0	-6.364	1.0	0.0595	0.1810
1808	Agora Hills	Doja Cat	7dJYggqjKo71KI9sLzqCs8	0	0.750	0.6740	8.0	-6.128	0.0	0.0970	0.2280
1809	gfg	Miguel	7xK1qc3jSIlo0UHah9WHdn	0	0.673	0.8970	2.0	-4.421	1.0	0.1300	0.1390
1810	UNA NOCHE EN MEDELLÍN - REMIX	KAROL G	6ejks4eS7DOoYW8hrpRcDV	0	0.843	0.7640	10.0	-2.494	0.0	0.0741	0.0356
1811	Light White Noise	Evomin	3qBMFORCRUKzBPiftvwaJn	0	0.222	0.0507	1.0	-44.323	1.0	0.0681	0.8980

1812 rows × 17 columns

Let's check for missing data:

In [ ]:

```
print(popular['acousticness'].isnull().sum())
print('----------------------------------')
print(popular['energy'].isnull().sum())
print('----------------------------------')
print(popular['key'].isnull().sum())
```

```python
print('----------------------------------')
print(popular['loudness'].isnull().sum())
print('----------------------------------')
print(popular['mode'].isnull().sum())
print('----------------------------------')
print(popular['instrumentalness'].isnull().sum())
print('----------------------------------')
print(popular['liveness'].isnull().sum())
print('----------------------------------')
print(popular['speechiness'].isnull().sum())
print('----------------------------------')
print(popular['time_signature'].isnull().sum())
print('----------------------------------')
print(popular['tempo'].isnull().sum())
print('----------------------------------')
print(popular['valence'].isnull().sum())
print('----------------------------------')
print(popular['duration_ms'].isnull().sum())
print('----------------------------------')

print('----------------unpopular----------------')

print(unpopular['acousticness'].isnull().sum())
print('----------------------------------')
print(unpopular['energy'].isnull().sum())
print('----------------------------------')
print(unpopular['key'].isnull().sum())
print('----------------------------------')
print(unpopular['loudness'].isnull().sum())
print('----------------------------------')
print(unpopular['mode'].isnull().sum())
print('----------------------------------')
print(unpopular['instrumentalness'].isnull().sum())
print('----------------------------------')
print(unpopular['liveness'].isnull().sum())
print('----------------------------------')
print(unpopular['speechiness'].isnull().sum())
print('----------------------------------')
print(unpopular['time_signature'].isnull().sum())
print('----------------------------------')
print(unpopular['tempo'].isnull().sum())
print('----------------------------------')
print(unpopular['valence'].isnull().sum())
print('----------------------------------')
print(unpopular['duration_ms'].isnull().sum())
print('----------------------------------')
```

```
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
1
----------------------------------
----------------unpopular----------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
5
----------------------------------
```

In here we see that we have the same number of missing rows in each column.

This could suggest that the missing data are from the same rows. Let's dive right in it

In [ ]:
```
popular[popular['acousticness'].isnull()==True]
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liv
718	#thatPOWER	will.i.am Featuring Justin Bieber	NaN	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

In [ ]:
```
unpopular[unpopular['acousticness'].isnull()==True]
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness	ir
941	White Noise in der Grotte	Xiskko	2Qh0vgdqdOr0g5C1ysGMzg	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
981	Ipnotizzato Dal Rumore Bianco	Passeggiate Al Chiaro	47MRiOYDPxibheDghy2Hum	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1068	LFRITH is the Future	Takashi Ohmama	7AbfD2ylaCua59rErkHQhD	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1204	DQQDM	Pranda	76FlHEyV5dbdjTCM2fTlxr	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1305	Gentle White Noise - Seamless	Natura Ferox	2COPUH8f4mlt1vzZG83zqU	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

Because the missing data are all on the same rows we cannot replace them with the average. We have to delete them

In [ ]:
```python
popular=popular.dropna()
popular
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness
0	Bad Day	Daniel Powter	0mUyMawtxj1CJ76kn9gIZK	1	0.599	0.785	3.0	-4.013	1.0	0.0309	0.44800
1	Temperature	Sean Paul	0k2GOhqsrxDTAbFFSdNJjT	1	0.951	0.600	0.0	-4.675	0.0	0.0685	0.10600
2	Promiscuous	Nelly Furtado Featuring Timbaland	2gam98EZKrF9XuOkU13ApN	1	0.808	0.970	10.0	-6.098	0.0	0.0506	0.05690
3	You're Beautiful	James Blunt	0vg4WnUWvze6pBOJDTq99k	1	0.675	0.479	0.0	-9.870	0.0	0.0278	0.63300
4	Hips Don't Lie	Shakira Featuring Wyclef Jean	3ZFTkvIE7kyPt6Nu3PEa7V	1	0.778	0.824	10.0	-5.892	0.0	0.0707	0.28400
...	...	...	...	...	...	...	...	...	...	...	...
1519	Flower Shops	ERNEST Featuring Morgan Wallen	0De9jFjJ4eRLI7Yww2eBw1	1	0.527	0.461	7.0	-5.908	1.0	0.0269	0.11800
1520	To The Moon!	JNR CHOI & Sam Tompkins	5vUnjhBzRJJIAOJPde6zDx	1	0.745	0.650	2.0	-11.814	1.0	0.3460	0.04510
1521	Unholy	Sam Smith & Kim Petras	3nqQXoyQOWXiESFLIDF1hG	1	0.714	0.472	2.0	-7.375	1.0	0.0864	0.01300
1522	One Mississippi	Kane Brown	4FdPnT2cFrpWCmWZd7GXc3	1	0.471	0.846	0.0	-5.269	1.0	0.0389	0.00279
1523	Circles Around This Town	Maren Morris	13G5xv1wUKvJYbK0wYmioN	1	0.591	0.814	4.0	-4.986	1.0	0.0468	0.01500

1523 rows × 17 columns

In [ ]:
```python
unpopular=unpopular.dropna()
unpopular
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness
0	This Is How We Do It	Montell Jordan	6uQKuonTU8VKBz5SHZuQXD	0	0.799	0.6230	0.0	-9.374	1.0	0.0812	0.0141
1	Homecoming	Kanye West	4iz9lGMjU1lXS51oPmUmTe	0	0.667	0.7470	1.0	-7.059	1.0	0.1890	0.3370
2	Believer	Imagine Dragons	0pqnGHJpmpxLKifKRmU6WP	0	0.776	0.7800	10.0	-4.374	0.0	0.1280	0.0622
3	Biking	Frank Ocean	2q0VexHJirnUPnEOhr2DxK	0	0.673	0.4630	2.0	-7.247	1.0	0.1910	0.6810
4	Nuyorican Soul	Carlos Henriquez	0ZDBZpIt3eUJ6LaApLJSiB	0	0.548	0.6050	5.0	-10.184	1.0	0.0428	0.7080
...	...	...	...	...	...	...	...	...	...	...	...
1807	Sundress	A$AP Rocky	2aPTvyE09vUCRwVvj0I8WK	0	0.721	0.7070	6.0	-6.364	1.0	0.0595	0.1810
1808	Agora Hills	Doja Cat	7dJYggqjKo71KI9sLzqCs8	0	0.750	0.6740	8.0	-6.128	0.0	0.0970	0.2280
1809	gfg	Miguel	7xK1qc3jSIlo0UHah9WHdn	0	0.673	0.8970	2.0	-4.421	1.0	0.1300	0.1390
1810	UNA NOCHE EN MEDELLÍN - REMIX	KAROL G	6ejks4eS7DOoYW8hrpRcDV	0	0.843	0.7640	10.0	-2.494	0.0	0.0741	0.0356
1811	Light White Noise	Evomin	3qBMFORCRUKzBPiftvwaJn	0	0.222	0.0507	1.0	-44.323	1.0	0.0681	0.8980

1807 rows × 17 columns

In [ ]:
```python
file_path = '/content/gdrive/MyDrive/Capstone/popular_cleaned_final.csv'
# Save the DataFrame to CSV
popular.to_csv(file_path, index=False)
```

In [ ]:
```python
file_path = '/content/gdrive/MyDrive/Capstone/unpopular_cleaned_final.csv'
```

```
# Save the DataFrame to CSV
unpopular.to_csv(file_path, index=False)
```

Let's calculate the means of our features to understand our data better:

In [ ]:

```
columns=['popular', 'unpopular']
features=pd.DataFrame({'Popular':[popular['danceability'].mean(),
popular['energy'].mean(),
popular['loudness'].mean(),
popular['speechiness'].mean(),
popular['acousticness'].mean(),
popular['instrumentalness'].mean(),
popular['liveness'].mean(),
popular['valence'].mean(),
popular['tempo'].mean(),
popular['key'].mean(),
popular['duration_ms'].mean()]
,'unpopular':[unpopular['danceability'].mean(),
unpopular['energy'].mean(),
unpopular['loudness'].mean(),
unpopular['speechiness'].mean(),
unpopular['acousticness'].mean(),
unpopular['instrumentalness'].mean(),
popular['key'].mean(),
unpopular['liveness'].mean(),
unpopular['valence'].mean(),
unpopular['tempo'].mean(),
unpopular['duration_ms'].mean()
]})
features.index=['danceability', 'energy',       'loudness',     'key',   'speechiness',   'acousticness', 'instru
features
```

Out[ ]:

	Popular	unpopular
danceability	0.663601	0.623411
energy	0.664275	0.600834
loudness	-5.977584	-9.049765
key	0.103984	0.118389
speechiness	0.170391	0.293147
acousticness	0.014096	0.165479
instrumentalness	0.174657	5.247538
liveness	0.515465	0.180166
valence	121.651978	0.479435
tempo	5.247538	121.495771
duration_ms	218858.494419	197192.415053

Let us analyse this a little bit feature-by-feature:

1. **Danceability**: This has a higher value for Popula compared to "Unpopular," suggesting that popular songs tend to be more danceable.
2. **Energy**: Popular songs have a higher energy value compared to "Unpopular" songs, indicating that popular songs are generally more energetic.
3. **Loudness**: Popular songs are typically louder (higher loudness value) than "Unpopular" songs.
4. **Speechiness**: This attribute measures the presence of spoken words in the music. "Popular" songs have slightly lower speechiness compared to Unpopular songs, meaning they might have more instrumental or less spoken content.
5. **Acousticness**: Unpopular songs have a higher acousticness value, indicating they are more likely to be acoustic or have acoustic elements.
6. **Instrumentalness**: Unpopular songs have a significantly higher instrumentalness value, suggesting that they are more likely to be instrumental without vocals.
7. **Liveness**: There is not much difference in liveness between Popular and Unpopular songs, as the values are relatively similar.
8. **Valence**: Popular songs have a slightly higher valence value, which implies they tend to be more positive or happier in mood.
9. **Tempo**: There is a very slight difference in tempo between the two categories, with Popular songs having a slightly higher tempo on average.
10. **Duration (in milliseconds)**: Popular songs tend to be longer in duration compared to "Unpopular" songs.

In [ ]:

```
popular.mean()
```

```
<ipython-input-53-1379d42f1e82>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprec
ated. In a future version, it will default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this warning.
  popular.mean()
```

Out[ ]:

```
Popular                   1.000000
danceability              0.663601
energy                    0.664275
key                       5.247538
loudness                 -5.977584
mode                      0.636901
speechiness               0.103984
acousticness              0.170391
instrumentalness          0.014096
liveness                  0.174657
valence                   0.515465
tempo                   121.651978
duration_ms          218858.494419
time_signature            3.978989
dtype: float64
```

Now, we will plot feature by feature to visualize the distribution of each feature:

In [ ]:

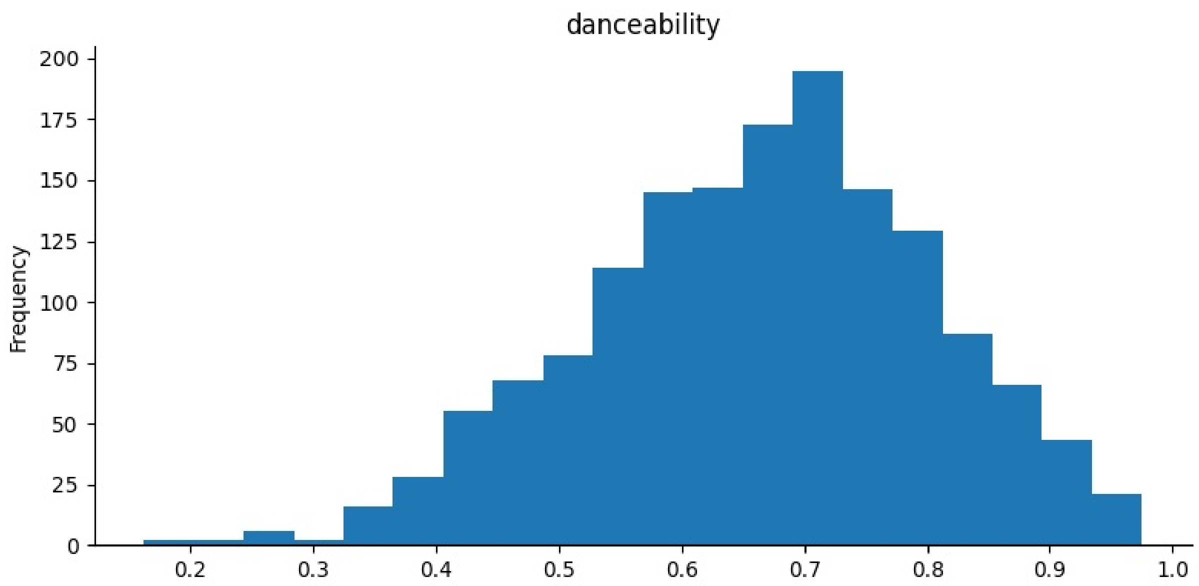
```
import numpy as np
from google.colab import autoviz
from matplotlib import pyplot as plt

def value_plot(df, y, figscale=1):
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()
def value_plot(df, y, figscale=1):
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

def value_plot(df, y, figscale=1):
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()
def histogram(df, colname, num_bins=20, figscale=1):
  df[colname].plot(kind='hist', bins=num_bins, title=colname, figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()
def violin_plot(df, value_colname, facet_colname, figscale=1, mpl_palette_name='Dark2', **kwargs):
  import seaborn as sns
  figsize = (12 * figscale, 1.2 * figscale * len(df[facet_colname].unique()))
  plt.figure(figsize=figsize)
  sns.violinplot(df, x=value_colname, y=facet_colname, palette=mpl_palette_name, **kwargs)
  sns.despine(top=True, right=True, bottom=True, left=True)
  return autoviz.MplChart.from_current_mpl_state()

def histogram(df, colname, num_bins=20, figscale=1):
  df[colname].plot(kind='hist', bins=num_bins, title=colname, figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()
```

First let us visualize the popular dataset:

In [ ]:

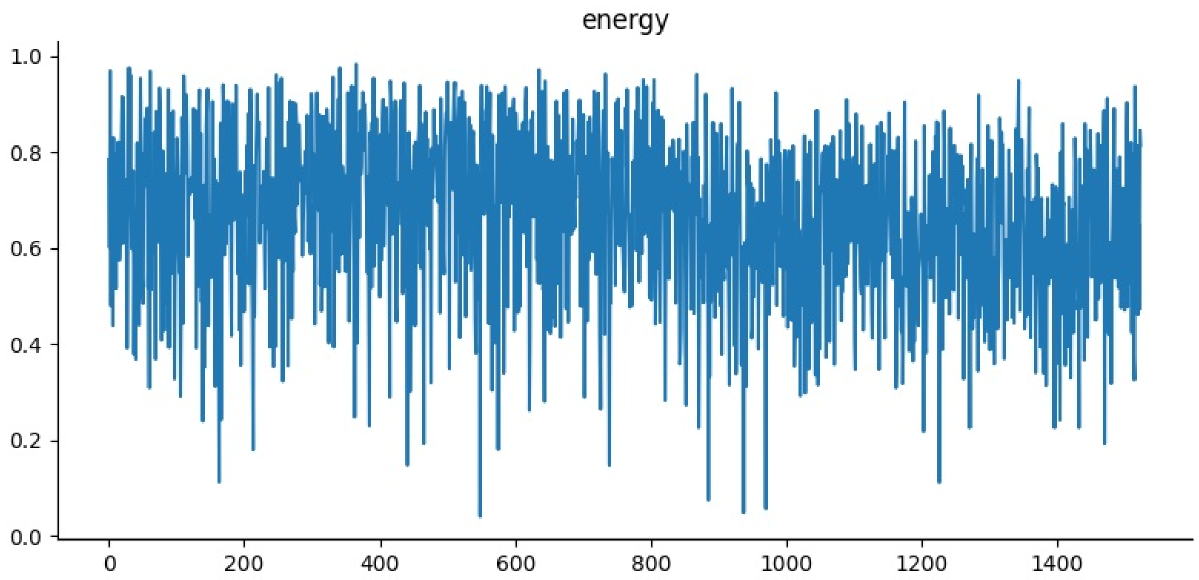
```
chart = histogram(popular, *['danceability'], **{})
chart
```

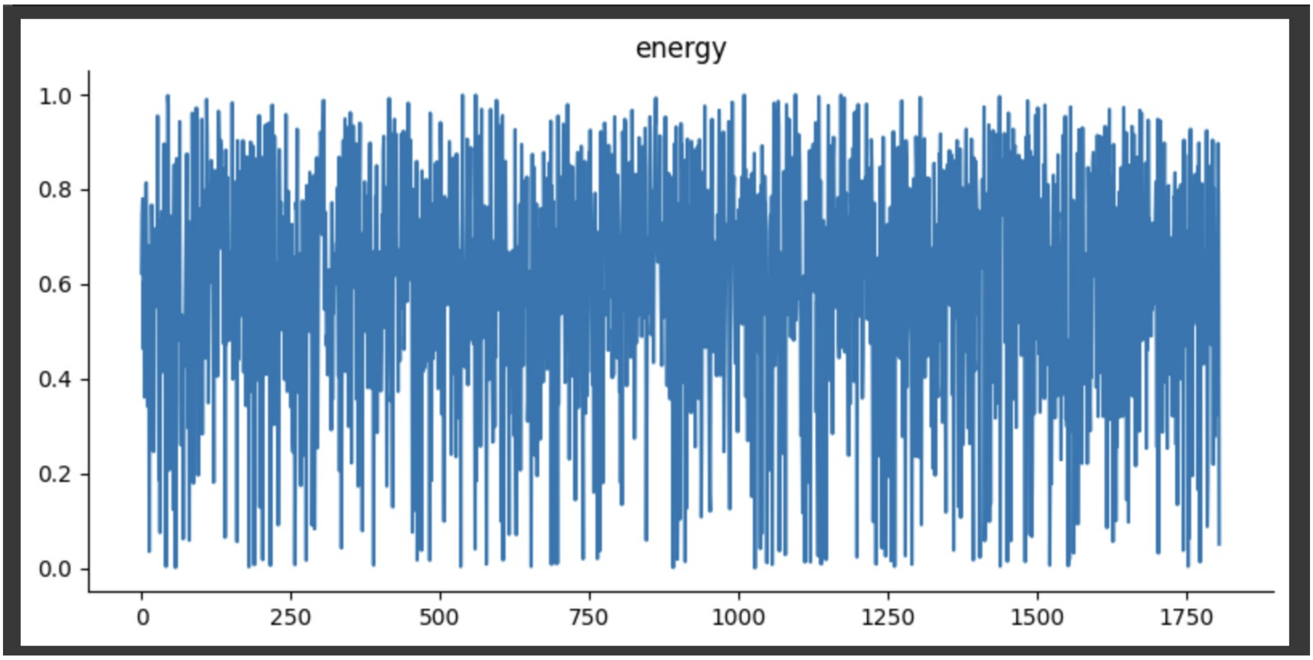
Out[ ]:



In [ ]:
```
chart = value_plot(popular, *['energy'], **{})
chart
```

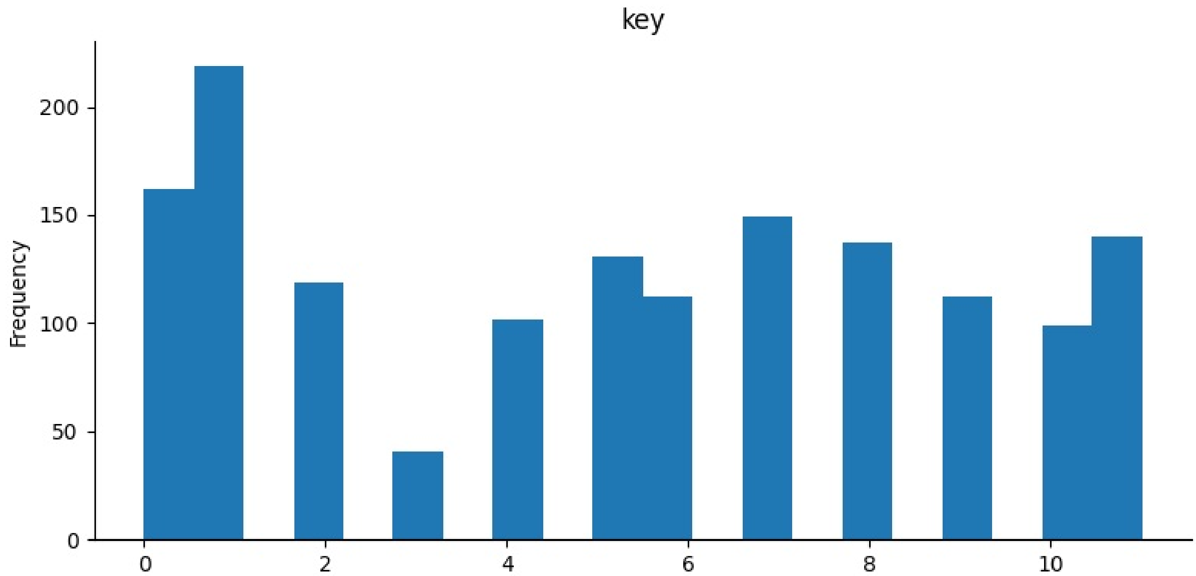
Out[ ]:

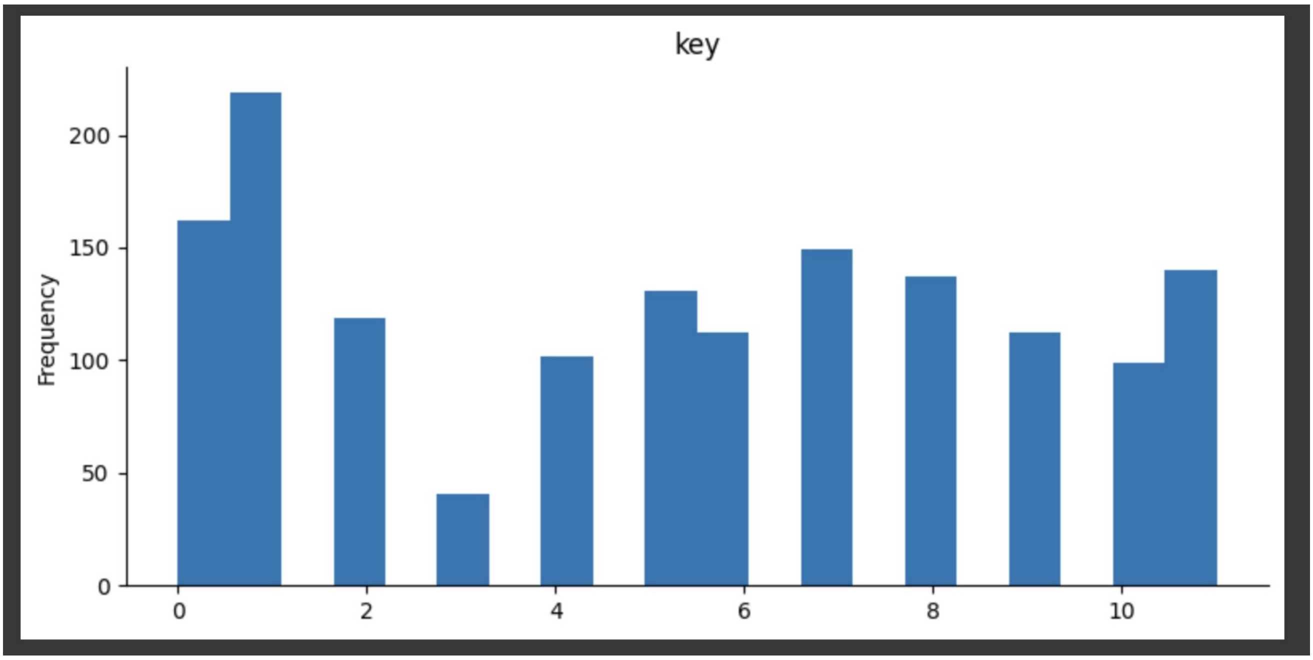




In [ ]:
```
chart = histogram(popular, *['key'], **{})
chart
```

Out[ ]:

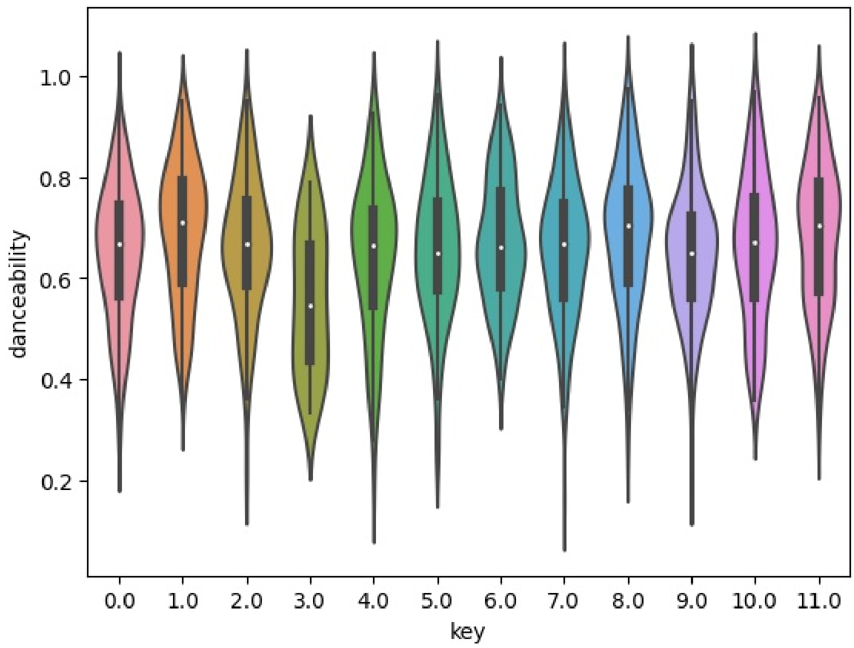


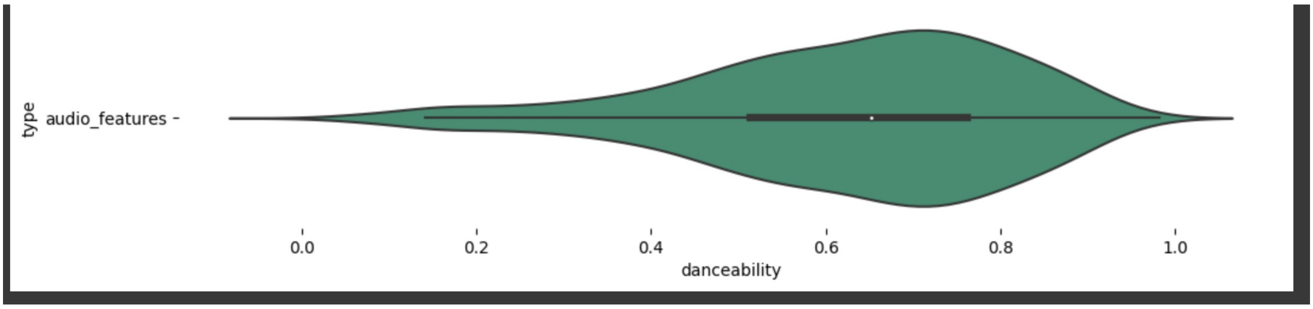


In [ ]:

```python
chart = sns.violinplot(x='key',
                       y='danceability',
                       data=popular)
chart
```

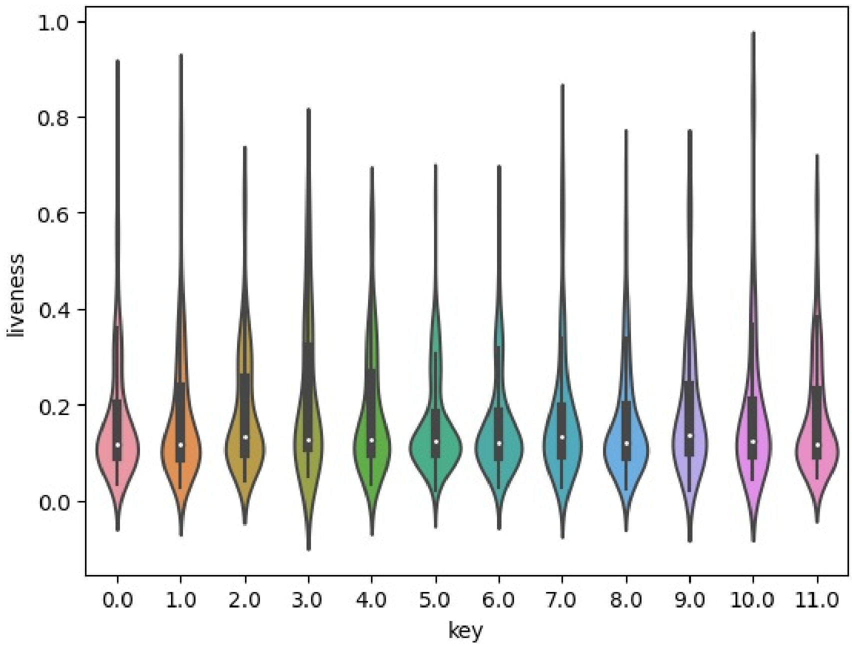
Out[ ]: <Axes: xlabel='key', ylabel='danceability'>





In [ ]:
```
chart = sns.violinplot(x='key',
                       y='liveness',
                       data=popular)
chart
```

Out[ ]: <Axes: xlabel='key', ylabel='liveness'>

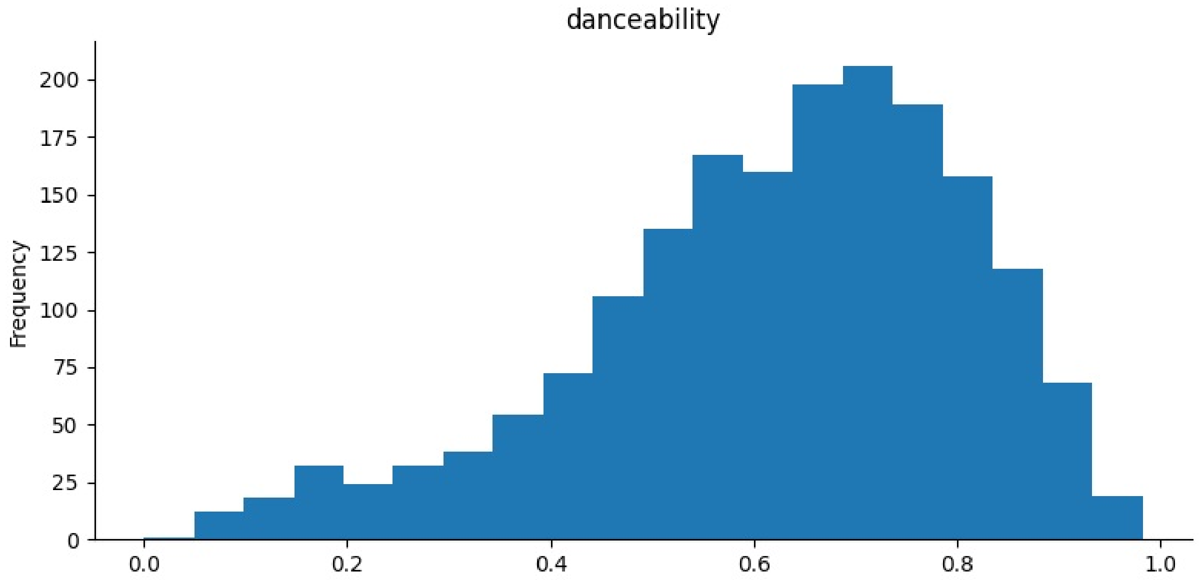


Now the unpopular data:

In [ ]:
```
def value_plot(df, y, figscale=1):
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

def value_plot(df, y, figscale=1):
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

def value_plot(df, y, figscale=1):
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

def histogram(df, colname, num_bins=20, figscale=1):
  df[colname].plot(kind='hist', bins=num_bins, title=colname, figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

def violin_plot(df, value_colname, facet_colname, figscale=1, mpl_palette_name='Dark2', **kwargs):
  import seaborn as sns
  figsize = (12 * figscale, 1.2 * figscale * len(df[facet_colname].unique()))
  plt.figure(figsize=figsize)
  sns.violinplot(df, x=value_colname, y=facet_colname, palette=mpl_palette_name, **kwargs)
  sns.despine(top=True, right=True, bottom=True, left=True)
  return autoviz.MplChart.from_current_mpl_state()

def histogram(df, colname, num_bins=20, figscale=1):
  df[colname].plot(kind='hist', bins=num_bins, title=colname, figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()
```

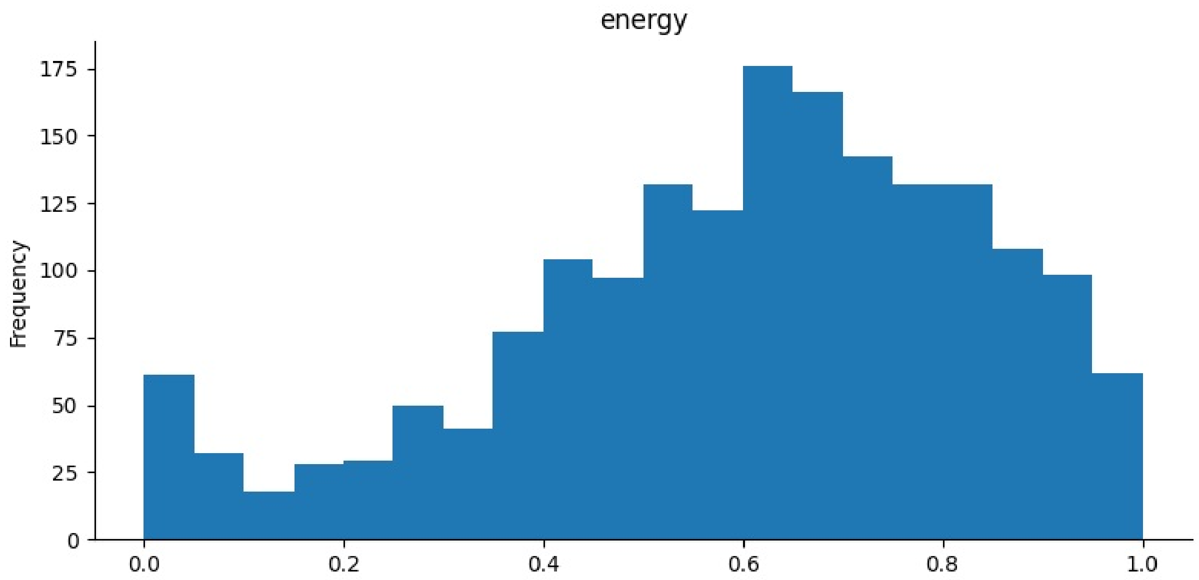
In [ ]:
```
chart = histogram(unpopular, *['danceability'], **{})
chart
```

Out[ ]:



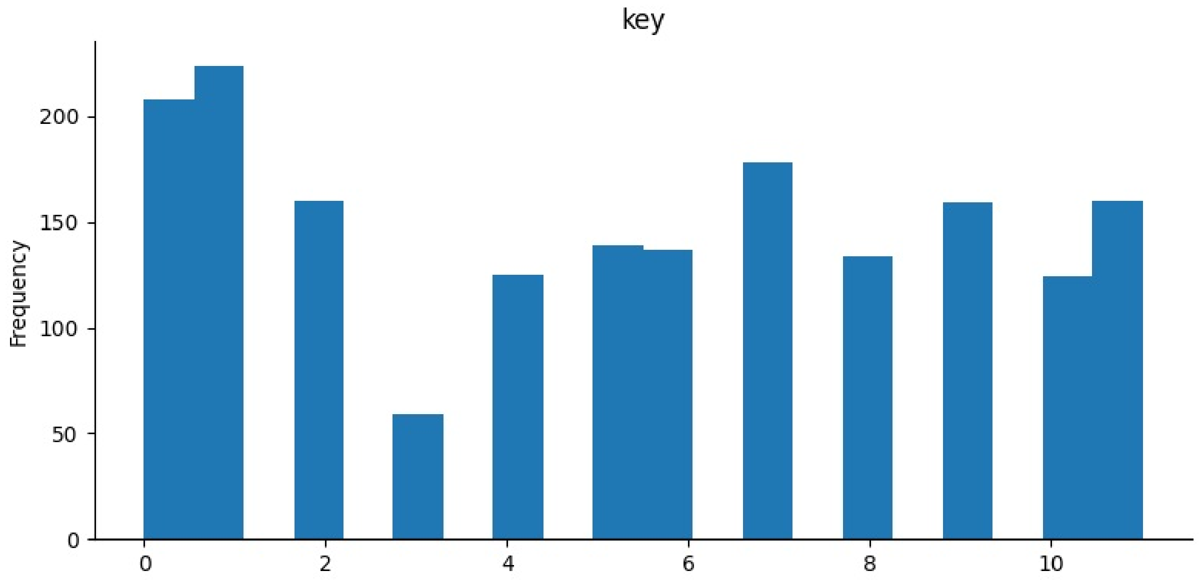
In [ ]:
```
chart = histogram(unpopular, *['energy'], **{})
chart
```

Out[ ]:



In [ ]:
```
chart = histogram(unpopular, *['key'], **{})
chart
```

Out[ ]:

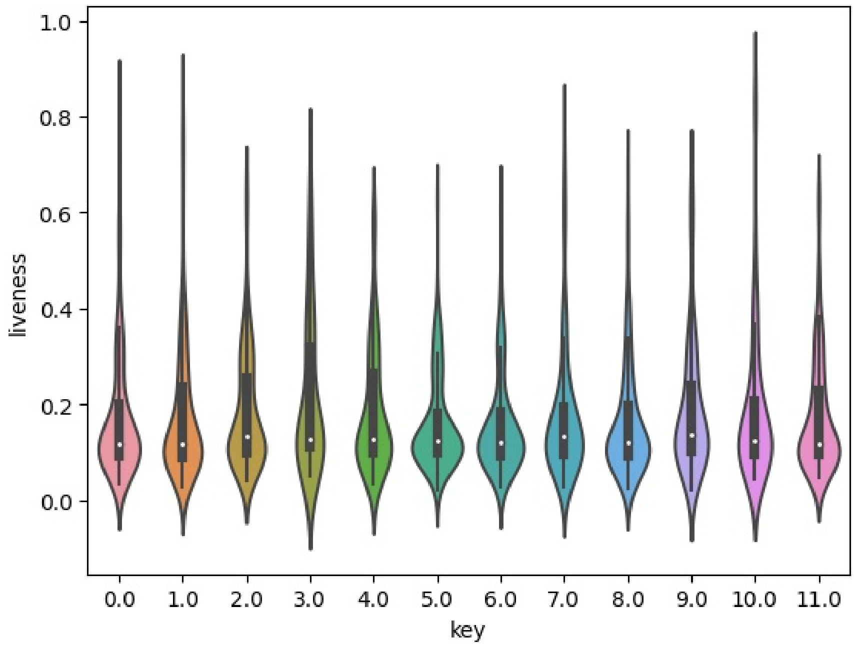


In [ ]:
```
chart = sns.violinplot(x='key',
```

```
                     y='liveness',
                     data=popular)
chart
```

Out[ ]:

<Axes: xlabel='key', ylabel='liveness'>

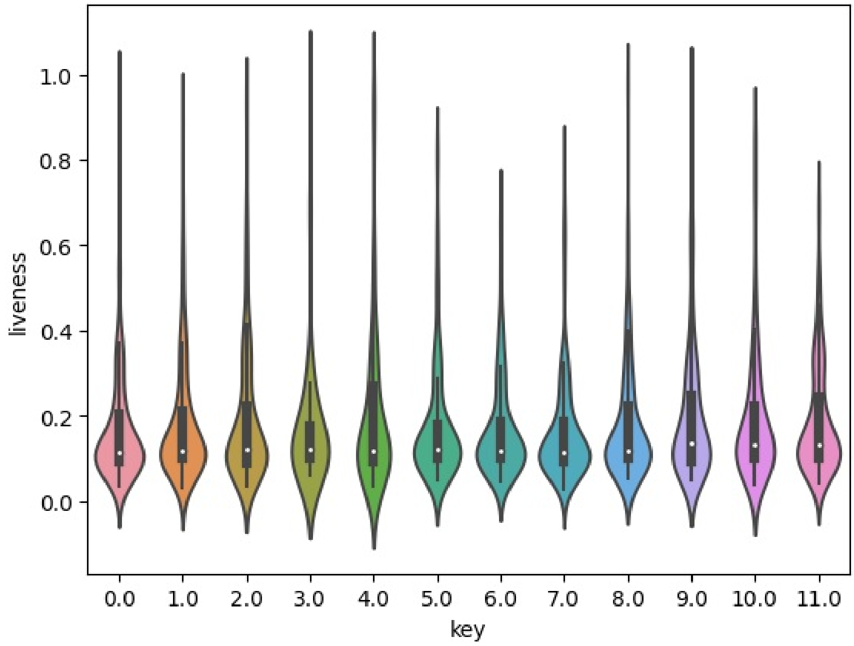


In [ ]:

```
chart = sns.violinplot(x='key',
                     y='liveness',
                     data=unpopular)
chart
```

Out[ ]:

<Axes: xlabel='key', ylabel='liveness'>



As we can wee, we do not have any significant outliers in our datasets based on the graphs.

Cleaned dataset

Once pre-processing and cleaning of data is done, we can take a look at our final datasets:

In [ ]:

```
popular
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness
0	Bad Day	Daniel Powter	0mUyMawtxj1CJ76kn9gIZK	1	0.599	0.785	3.0	-4.013	1.0	0.0309	0.44800
1	Temperature	Sean Paul	0k2GOhqsrxDTAbFFSdNJjT	1	0.951	0.600	0.0	-4.675	0.0	0.0685	0.10600
2	Promiscuous	Nelly Furtado Featuring Timbaland	2gam98EZKrF9XuOkU13ApN	1	0.808	0.970	10.0	-6.098	0.0	0.0506	0.05690
3	You're Beautiful	James Blunt	0vg4WnUWvze6pBOJDTq99k	1	0.675	0.479	0.0	-9.870	0.0	0.0278	0.63300
4	Hips Don't Lie	Shakira Featuring Wyclef Jean	3ZFTkvIE7kyPt6Nu3PEa7V	1	0.778	0.824	10.0	-5.892	0.0	0.0707	0.28400
...	...	...	...	...	...	...	...	...	...	...	...
1519	Flower Shops	ERNEST Featuring Morgan Wallen	0De9jFjJ4eRLI7Yww2eBw1	1	0.527	0.461	7.0	-5.908	1.0	0.0269	0.11800
1520	To The Moon!	JNR CHOI & Sam Tompkins	5vUnjhBzRJJIAOJPde6zDx	1	0.745	0.650	2.0	-11.814	1.0	0.3460	0.04510
1521	Unholy	Sam Smith & Kim Petras	3nqQXoyQOWXiESFLIDF1hG	1	0.714	0.472	2.0	-7.375	1.0	0.0864	0.01300
1522	One Mississippi	Kane Brown	4FdPnT2cFrpWCmWZd7GXc3	1	0.471	0.846	0.0	-5.269	1.0	0.0389	0.00279
1523	Circles Around This Town	Maren Morris	13G5xv1wUKvJYbK0wYmioN	1	0.591	0.814	4.0	-4.986	1.0	0.0468	0.01500

1523 rows × 17 columns

In [ ]:

```
unpopular
```

Out[ ]:

	Title	Artist	track_id	Popular	danceability	energy	key	loudness	mode	speechiness	acousticness
0	This Is How We Do It	Montell Jordan	6uQKuonTU8VKBz5SHZuQXD	0	0.799	0.6230	0.0	-9.374	1.0	0.0812	0.0141
1	Homecoming	Kanye West	4iz9lGMjU1lXS51oPmUmTe	0	0.667	0.7470	1.0	-7.059	1.0	0.1890	0.3370
2	Believer	Imagine Dragons	0pqnGHJpmpxLKifKRmU6WP	0	0.776	0.7800	10.0	-4.374	0.0	0.1280	0.0622
3	Biking	Frank Ocean	2q0VexHJirnUPnEOhr2DxK	0	0.673	0.4630	2.0	-7.247	1.0	0.1910	0.6810
4	Nuyorican Soul	Carlos Henriquez	0ZDBZplt3eUJ6LaApLJSiB	0	0.548	0.6050	5.0	-10.184	1.0	0.0428	0.7080
...	...	...	...	...	...	...	...	...	...	...	...
1807	Sundress	A$AP Rocky	2aPTvyE09vUCRwVvj0I8WK	0	0.721	0.7070	6.0	-6.364	1.0	0.0595	0.1810
1808	Agora Hills	Doja Cat	7dJYggqjKo71KI9sLzqCs8	0	0.750	0.6740	8.0	-6.128	0.0	0.0970	0.2280
1809	gfg	Miguel	7xK1qc3jSIlo0UHah9WHdn	0	0.673	0.8970	2.0	-4.421	1.0	0.1300	0.1390
1810	UNA NOCHE EN MEDELLÍN - REMIX	KAROL G	6ejks4eS7DOoYW8hrpRcDV	0	0.843	0.7640	10.0	-2.494	0.0	0.0741	0.0356
1811	Light White Noise	Evomin	3qBMFORCRUKzBPiftvwaJn	0	0.222	0.0507	1.0	-44.323	1.0	0.0681	0.8980

1807 rows × 17 columns

Workflow diagram

The workflow diagram for the data preprocessing stage is here: https://docs.google.com/spreadsheets/d/1LdNVKHg3EnHiZcuw9538fHyUf68EXCZ8zswLvNMp8f0/edit?usp=sharing.

To see the full project plan, please visit the following link: https://docs.google.com/spreadsheets/d/16t8u2G9vUrQpGFP8mHUrWB7OP01J-O2KPK3U6zfSoCQ/edit?usp=sharing

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js