Spotify® Analysis

About Dataset

This dataset contains a comprehensive list of the most famous songs of 2023 as listed on Spotify. The dataset offers a wealth of features beyond what is typically available in similar datasets. It provides insights into each song's attributes, popularity, and presence on various music platforms. The dataset includes information such as track name, artist(s) name, release date, Spotify playlists and charts, streaming statistics, Apple Music presence, Deezer presence, Shazam charts, and various audio features.

Field	Description
track_name	Name of the song
artist(s)_name	Name of the artist(s) of the song
artist_count	Number of artists contributing to the song
released_year	Year when the song was released
released_month	Month when the song was released
released_day	Day of the month when the song was released
in_spotify_playlists	Number of Spotify playlists the song is included in
in_spotify_charts	Presence and rank of the song on Spotify charts
streams	Total number of streams on Spotify
in_apple_playlists	Number of Apple Music playlists the song is included in
in_apple_charts	Presence and rank of the song on Apple Music charts
in_deezer_playlists	Number of Deezer playlists the song is included in
in_deezer_charts	Presence and rank of the song on Deezer charts
in_shazam_charts	Presence and rank of the song on Shazam charts
bpm	Beats per minute, a measure of song tempo
key	Key of the song
mode	Mode of the song (major or minor)
danceability_%	Percentage indicating how suitable the song is for dancing
valence_%	Positivity of the song's musical content
energy_%	Perceived energy level of the song
acousticness_%	Amount of acoustic sound in the song

Field	Description
instrumentalness_%	Amount of instrumental content in the song
liveness_%	Presence of live performance elements
speechiness_%	Amount of spoken words in the song

Objective/Aim

- Content Analysis
- Playlist Popularity Analysis
- Chart Performance Analysis
- Genre and Mood Analysis
- User Engagement Trends
- Recommendation System Insights

Team Members

Noora Hussain Roll No: 3402	**Arun Krishna** Roll No: 3406	**Muhammed Efas** Roll No: 3411
Shyam Kiran Roll No: 3414	**Ashwin Venugopal** Roll No: 3424	**Aasim Mohammed** Roll No: 3488

Source of Data

kaggle

Importing the Necessary Library

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

Loading the Data

In [ ]:
```
df_org=pd.read_csv("spt.csv",encoding= 'unicode-escape')
```

In [ ]:
```
df_org.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   track_name            953 non-null    object
 1   artist(s)_name        953 non-null    object
 2   artist_count          953 non-null    int64 
 3   released_year         953 non-null    int64 
 4   released_month        953 non-null    int64 
 5   released_day          953 non-null    int64 
 6   in_spotify_playlists  953 non-null    int64 
 7   in_spotify_charts     953 non-null    int64 
 8   streams               953 non-null    object
 9   in_apple_playlists    953 non-null    int64 
 10  in_apple_charts       953 non-null    int64 
 11  in_deezer_playlists   953 non-null    object
 12  in_deezer_charts      953 non-null    int64 
 13  in_shazam_charts      903 non-null    object
 14  bpm                   953 non-null    int64 
 15  key                   858 non-null    object
 16  mode                  953 non-null    object
 17  danceability_%        953 non-null    int64 
 18  valence_%             953 non-null    int64 
 19  energy_%              953 non-null    int64 
 20  acousticness_%        953 non-null    int64 
 21  instrumentalness_%    953 non-null    int64 
 22  liveness_%            953 non-null    int64 
 23  speechiness_%         953 non-null    int64 
dtypes: int64(17), object(7)
memory usage: 178.8+ KB
```

In [ ]:
```
df_org.shape
```

Out[ ]: (953, 24)

Dimension of the Dataset is 953 x 24

Descriptive Statistics of the Dataframe

Quick overview of the distribution of the numerical data in the dataframe. This helps us in understanding key statistical measures for each colum

In [ ]:
```
df_org.describe()
```

Out[ ]:

	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_
count	953.000000	953.000000	953.000000	953.000000	953.000000	953.(
mean	1.556139	2018.238195	6.033578	13.930745	5200.124869	12.(
std	0.893044	11.116218	3.566435	9.201949	7897.608990	19.!
min	1.000000	1930.000000	1.000000	1.000000	31.000000	0.(
25%	1.000000	2020.000000	3.000000	6.000000	875.000000	0.(
50%	1.000000	2022.000000	6.000000	13.000000	2224.000000	3.(
75%	2.000000	2022.000000	9.000000	22.000000	5542.000000	16.(
max	8.000000	2023.000000	12.000000	31.000000	52898.000000	147.(

Cleaning Data

In [ ]:
```
#Renaming the column name of artist

df_org = df_org.rename(columns={'artist(s)_name': 'artists'})
```

In [ ]:
```
# Checking if there are any duplicate rows

print("Number of rows which are duplicated entirely: ", df_org.duplicated().sum())
```

```
Number of rows which are duplicated entirely:  0
```

Finding number of NULL values in each column and corresponding percentage.

In [ ]:
```
df_null = pd.DataFrame(columns=['Total Null Values', 'Null Percentage'])

def check_null(df):
    df['Total Null Values'] = df_org.isnull().sum()
    df['Null Percentage'] = (df_org.isnull().sum() / len(df_org)) * 100
    return df

check_null(df_null)
```

Out[ ]:

	Total Null Values	Null Percentage
track_name	0	0.00000
artists	0	0.00000
artist_count	0	0.00000
released_year	0	0.00000
released_month	0	0.00000
released_day	0	0.00000
in_spotify_playlists	0	0.00000
in_spotify_charts	0	0.00000
streams	0	0.00000
in_apple_playlists	0	0.00000
in_apple_charts	0	0.00000
in_deezer_playlists	0	0.00000
in_deezer_charts	0	0.00000
in_shazam_charts	50	5.24659
bpm	0	0.00000
key	95	9.96852
mode	0	0.00000
danceability_%	0	0.00000
valence_%	0	0.00000
energy_%	0	0.00000
acousticness_%	0	0.00000
instrumentalness_%	0	0.00000
liveness_%	0	0.00000
speechiness_%	0	0.00000

- The columns *in_shazam_charts* and *key* have NULL values.

Finding rows with null value in *in_shazam_charts* column

In [ ]:
```
df_org[df_org['in_shazam_charts'].isnull()][['track_name', 'in_shazam_charts']]
```

Out[ ]:

	track_name	in_shazam_charts
14	As It Was	NaN
54	Another Love	NaN
55	Blinding Lights	NaN
71	Heat Waves	NaN
73	Sweater Weather	NaN
86	Someone You Loved	NaN
127	Watermelon Sugar	NaN
158	Ghost	NaN
159	Under The Influence	NaN
180	Night Changes	NaN
243	Unstoppable	NaN
274	Shivers	NaN
320	Gangsta's Paradise	NaN
392	Calm Down	NaN
395	Space Song	NaN
403	One Kiss (with Dua Lipa)	NaN
410	INDUSTRY BABY (feat. Jack Harlow)	NaN
429	Bad Habits	NaN
434	Woman	NaN
440	Payphone	NaN
441	All I Want for Christmas Is You	NaN
442	Last Christmas	NaN
443	Rockin' Around The Christmas Tree	NaN
444	Jingle Bell Rock	NaN
446	Santa Tell Me	NaN
449	Snowman	NaN
500	ýýýabcdefu	NaN
501	Sacrifice	NaN
504	Out of Time	NaN
506	We Don't Talk About Bruno	NaN
507	Pepas	NaN
513	good 4 u	NaN
518	Need To Know	NaN
519	MONTERO (Call Me By Your Name)	NaN
520	love nwantiti (ah ah ah)	NaN
529	MONEY	NaN
531	Happier Than Ever	NaN

	track_name	in_shazam_charts
532	Moth To A Flame (with The Weeknd)	NaN
533	traitor	NaN
534	Toxic	NaN
535	drivers license	NaN
549	Love Nwantiti - Remix	NaN
554	Peaches (feat. Daniel Caesar & Giveon)	NaN
560	Life Goes On	NaN
566	Dynamite	NaN
584	Mood (feat. Iann Dior)	NaN
620	Dance Monkey	NaN
625	Arcade	NaN
727	Somebody That I Used To Know	NaN
927	I Really Want to Stay at Your House	NaN

Filling NULL values of *in_shazam_charts* with mean of *in_spotify_charts*, *in_deezer_charts*, *in_apple_charts*

In [ ]:

```
# Function to calculate the mean of non-null values in a row
def row_mean(row):
    non_null_values = row.dropna()
    if non_null_values.empty:
        return np.nan
    return int(non_null_values.mean())

# Applying the function to each row and filling null values in 'in_shazam_charts'
df_org['in_shazam_charts'] = df_org.apply(lambda row: row_mean(row[['in_spotify_charts
print('Number of NULL values in \'in_shazam_charts\':', df_org['in_shazam_charts'].isn
```

```
Number of NULL values in 'in_shazam_charts': 0
```

Finding rows with NULL values in *key* column

In [ ]:

```
df_org[df_org['key'].isnull()][['track_name', 'key']]
```

Out[ ]:

	track_name	key
12	Flowers	NaN
17	What Was I Made For? [From The Motion Picture ...	NaN
22	I Wanna Be Yours	NaN
35	Los del Espacio	NaN
44	Barbie World (with Aqua) [From Barbie The Album]	NaN
...	...	...
899	Hold Me Closer	NaN
901	After LIKE	NaN
903	B.O.T.A. (Baddest Of Them All) - Edit	NaN
938	Labyrinth	NaN
940	Sweet Nothing	NaN

95 rows × 2 columns

For the null values in the *key* column, the key of the song can be converted into integers using the standard Pitch Class Notation. The null values for the *key* column will have a value of -1.

Tonal Counterparts	Pitch Class
-1	NULL
0	C
1	C#
2	D
3	D#
4	E
5	F
6	F#
7	G
8	G#
9	A
10	A#
11	B

In [ ]:

```
# Mapping table
pitch_class_mapping = {
    np.nan:-1,
    'C': 0, 'C#': 1, 'D': 2, 'D#': 3,
    'E': 4, 'F': 5, 'F#': 6, 'G': 7,
    'G#': 8, 'A': 9, 'A#': 10, 'B': 11
}

df_org['key'] = df_org['key'].map(pitch_class_mapping)
```

Checking if there are still any NULL values present in any column

In [ ]:
```
check_null(df_null)
df_null
```

Out[ ]:

	Total Null Values	Null Percentage
track_name	0	0.0
artists	0	0.0
artist_count	0	0.0
released_year	0	0.0
released_month	0	0.0
released_day	0	0.0
in_spotify_playlists	0	0.0
in_spotify_charts	0	0.0
streams	0	0.0
in_apple_playlists	0	0.0
in_apple_charts	0	0.0
in_deezer_playlists	0	0.0
in_deezer_charts	0	0.0
in_shazam_charts	0	0.0
bpm	0	0.0
key	0	0.0
mode	0	0.0
danceability_%	0	0.0
valence_%	0	0.0
energy_%	0	0.0
acousticness_%	0	0.0
instrumentalness_%	0	0.0
liveness_%	0	0.0
speechiness_%	0	0.0

In *in_deezer_playlist* (object type), there are comma values for integers greater than 999. Convert the entire column to integer

In [ ]:
```
df_org['in_deezer_playlists'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 953 entries, 0 to 952
Series name: in_deezer_playlists
Non-Null Count  Dtype
--------------  -----
953 non-null    object
dtypes: object(1)
memory usage: 7.6+ KB
```

Replacing commas with blank space using regex and converting to integer

```
In [ ]: df_org['in_deezer_playlists'] = df_org['in_deezer_playlists'].str.replace(',', '', reg
        df_org['in_deezer_playlists'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 953 entries, 0 to 952
Series name: in_deezer_playlists
Non-Null Count  Dtype
--------------  -----
953 non-null    int64
dtypes: int64(1)
memory usage: 7.6 KB
```

While analysing the data, we found discrepancy in *streams* column. It should be in int64 data type, while it is currently in object data type.

Checking the particular track with discrepancy

```
In [ ]: df_org[df_org['streams'].apply(pd.to_numeric, errors='coerce').isna()]
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spot
574	Love Grows (Where My Rosemary Goes)	Edison Lighthouse	1	1970	1	1	

1 rows × 24 columns

Change the stream value of the particular song which is Invalid

```
In [ ]: df_org.loc[df_org['track_name'] == "Love Grows (Where My Rosemary Goes)", 'streams'] =
        df_org.loc[df_org['track_name'] == "Love Grows (Where My Rosemary Goes)"]
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spot
574	Love Grows (Where My Rosemary Goes)	Edison Lighthouse	1	1970	1	1	

1 rows × 24 columns

Change the datatype of *streams* from object to int

```
In [ ]: df_org['streams']=df_org['streams'].astype('int64')
        df_org['streams'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 953 entries, 0 to 952
Series name: streams
Non-Null Count  Dtype
--------------  -----
953 non-null    int64
dtypes: int64(1)
memory usage: 7.6 KB
```

Checking track names to check for discrepancy

In [ ]:
```python
unique_track = df_org['track_name'].unique()
unique_track.sort()
print(unique_track[:20])
print(unique_track[-20:])
```

```
["'Till I Collapse" '(It Goes Like) Nanana - Edit'
 '10 Things I Hate About You' '10:35' '2 Be Loved (Am I Ready)' '2055'
 '212' '25k jacket (feat. Lil Baby)' '295' '505' '69'
 'A Holly Jolly Christmas - Single Version' 'A Tale By Quincy'
 'A Tu Merced' 'A Veces (feat. Feid)' 'ALIEN SUPERSTAR' 'AM Remix'
 'AMARGURA' 'AMERICA HAS A PROBLEM (feat. Kendrick Lamar)' 'AMG']
['jealousy, jealousy' 'love nwantiti (ah ah ah)' 'lovely - Bonus Track'
 'on the street (with J. Cole)' 'positions' 'psychofreak (feat. WILLOW)'
 'pushin P (feat. Young Thug)' 'sentaDONA (Remix) s2'
 "she's all i wanna be" 'this is what falling in love feels like'
 'thought i was playing' 'traitor' 'un x100to' 'vampire'
 'we fell in love in october' 'you broke me first' 'ýýý98 Braves'
 'ýýýabcdefu' 'ýýýýýýýýýýýý' 'ýýýýýýýýýýýýýýýýýýýýý']
```

While analysing the dataset, we found that certain track names contains special characters. So we decided to replace it.

In [ ]:
```python
#Checking rows with special characters
characters_to_replace = ['ý', 'ï', '¿', 'Â', '½', 'Ã', '¯']

def contains_special_character(cell):
    return any(char in cell for char in characters_to_replace)

rows_with_special_characters = df_org[df_org.astype(str).apply(lambda row: any(contain


print("Number of rows with special characters: ", rows_with_special_characters.shape[0
print("Rows with special characters:")
rows_with_special_characters
```

```
Number of rows with special characters:  109
Rows with special characters:
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_
21	I Can See You (Taylorï¿½ï¿½ï¿½s Version) (From...	Taylor Swift	1	2023	7	7	
26	Calm Down (with Selena Gomez)	Rï¿½ï¿½ma, Selena G	2	2022	3	25	
36	Frï¿½ï¿½gil (feat. Grupo Front	Yahritza Y Su Esencia, Grupo Frontera	2	2023	4	7	
60	Tï¿½ï¿	dennis, MC Kevin o Chris	2	2023	5	4	
63	BESO	Rauw Alejandro, ROSALï¿½	2	2023	3	24	
...	...	...	...	...	...	...	
887	ALIEN SUPERSTAR	Beyoncï¿	1	2022	7	29	
913	XQ Te Pones Asï¿	Yandel, Feid	2	2022	9	13	
915	Sin Seï¿½ï	Ovy On The Drums, Quevedo	2	2022	7	22	
918	THE LONELIEST	Mï¿½ï¿½ne	1	2022	10	7	
929	Bamba (feat. Aitch & BIA)	Luciano, Aitch, Bï¿½	3	2022	9	22	

109 rows × 24 columns

- There are 109 rows with such special characters.

Changing the track name where there are special characters

In [ ]:

```
for char in characters_to_replace:
    df_org['track_name'] = df_org['track_name'].str.replace(char, '')

df_org.head(5)
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spotify_pl
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	
1	LALA	Myke Towers	1	2023	3	23	
2	vampire	Olivia Rodrigo	1	2023	6	30	
3	Cruel Summer	Taylor Swift	1	2019	8	23	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	

5 rows × 24 columns

While changing the track name, we came across songs whose track name is entirely special characters

In [ ]:

```
#Checking rows where track_name is empty or NULL

null_track_name_rows = df_org[df_org['track_name'].isnull() | (df_org['track_name'] ==
print("Rows with NULL or empty track name:")
null_track_name_rows
```

```
Rows with NULL or empty track name:
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spotify
174		YOASOBI	1	2023	4	12	
374		Fujii Kaze	1	2020	5	20	

2 rows × 24 columns

We change the track name by cross-referencing the artist and released date on the Internet

In [ ]:

```
#Replacing the track name with original

df_org.loc[374, 'track_name'] = 'Shinunoga E-Wa'
df_org.loc[174, 'track_name'] = 'Run Into The Night'
df_org.loc[[374, 174]]
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spotify
374	Shinunoga E-Wa	Fujii Kaze	1	2020	5	20	
174	Run Into The Night	YOASOBI	1	2023	4	12	

2 rows × 24 columns

Checking artists with special characters

In [ ]:
```
total_artists_with_special_characters = 0

for char in characters_to_replace:
    char_present = df_org['artists'].str.contains(char)
    values_with_char = df_org['artists'][char_present]
    if not values_with_char.empty:
        total_artists_with_special_characters += len(values_with_char)

print('Total number of artists with special characters: ', total_artists_with_special_
```

```
Total number of artists with special characters:  136
```

Replacing special characters in artist name with

In [ ]:
```
for char in characters_to_replace:
    df_org['artists'] = df_org['artists'].str.replace(char, '#')

df_org.head(5)
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spotify_pl
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	
1	LALA	Myke Towers	1	2023	3	23	
2	vampire	Olivia Rodrigo	1	2023	6	30	
3	Cruel Summer	Taylor Swift	1	2019	8	23	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	

5 rows × 24 columns

While trying to split the artists from *artists* column, we encountered an error. On further analysing it, we found that the artists attribute of the song 'Nobody Like U - From "Turning Red"' contains unwanted characters

In [ ]:
```
with pd.option_context('display.max_colwidth', None):
    print(df_org.loc[df_org['track_name'] == "Nobody Like U - From \"Turning Red\""][[
```

```
artists
759  Jordan Fisher, Josh Levi, Finneas O'Connell, 4*TOWN (From Disney and Pixar########
#s Turning Red), Topher Ngo, Grayson Vill
```

Fixing the artists of that track

In [ ]:
```
df_org.loc[df_org['track_name'] == "Nobody Like U - From \"Turning Red\"", 'artists']=
df_org.loc[df_org['track_name'] == "Nobody Like U - From \"Turning Red\""]
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spoti
759	Nobody Like U - From "Turning Red"	Jordan Fisher, Josh Levi, Finneas O'Connell, 4...	6	2022	2	25	

1 rows × 24 columns

Dealing with Duplicate Elements

Finding duplicate tracks by checking *track name* & *artist*. For duplicate tracks, we decided to keep the row with higher number of *streams*

In [ ]:

```
duplicate = df_org.sort_values(by='streams', ascending=False)[df_org.sort_values(by='s
duplicate
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spoti
764	About Damn Time	Lizzo	1	2022	4	14	
873	SNAP	Rosa Linn	1	2022	3	19	
482	SPIT IN MY FACE!	ThxSoMch	1	2022	10	31	
512	Take My Breath	The Weeknd	1	2021	8	6	

4 rows × 24 columns

Drop duplicate rows from the data

In [ ]:

```
df_org.drop(duplicate.index, axis=0, inplace=True)
df_org.shape
```

Out[ ]: (949, 24)

Resetting the index

In [ ]:

```
df_org.reset_index(drop=True, inplace=True)
df_org
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spotify_
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	
1	LALA	Myke Towers	1	2023	3	23	
2	vampire	Olivia Rodrigo	1	2023	6	30	
3	Cruel Summer	Taylor Swift	1	2019	8	23	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	
...	...	...	...	...	...	...	
944	My Mind & Me	Selena Gomez	1	2022	11	3	
945	Bigger Than The Whole Sky	Taylor Swift	1	2022	10	21	
946	A Veces (feat. Feid)	Feid, Paulo Londra	2	2022	11	3	
947	En La De Ella	Feid, Sech, Jhayco	3	2022	10	20	
948	Alone	Burna Boy	1	2022	11	4	

949 rows × 24 columns

Exporting the cleaned data to CSV and Making a copy of DataFrame to perform further analysis

In [ ]:

```
df_org.to_csv('cleaned_data.csv')
df_copy = df_org.copy()
```

At this stage, the pre-processing is complete. Now we move on to identifying outliers and other analysis.

Identifying Outliers

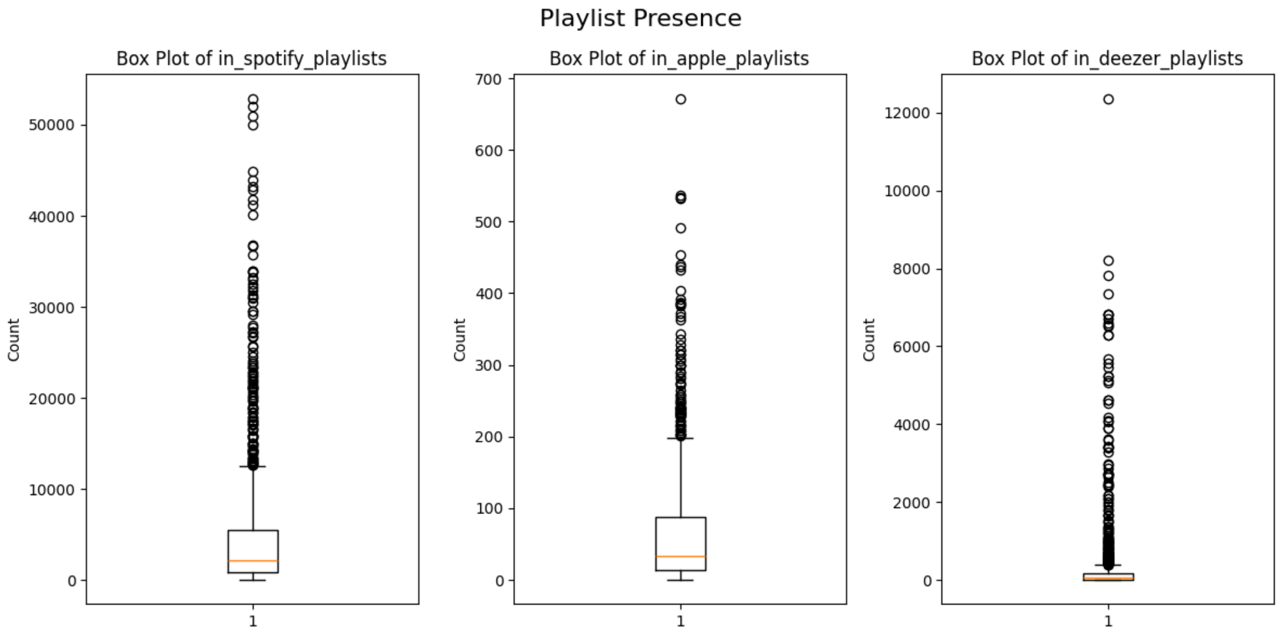
Removing columns with object type for correlation matrix

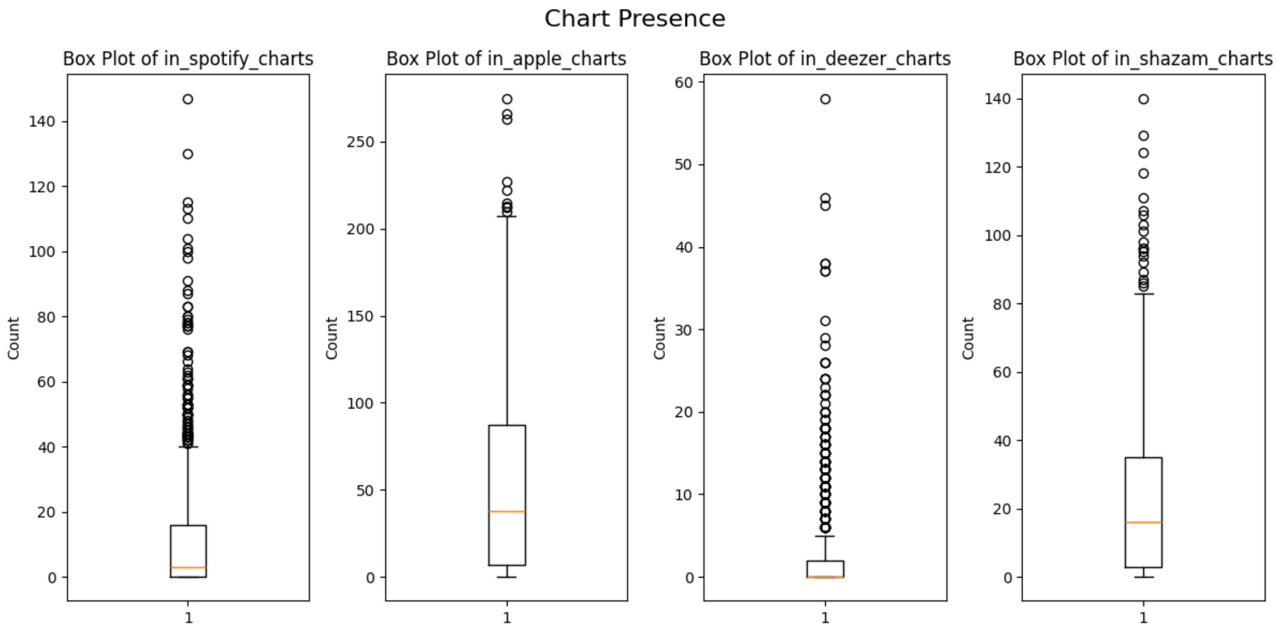
In [ ]:

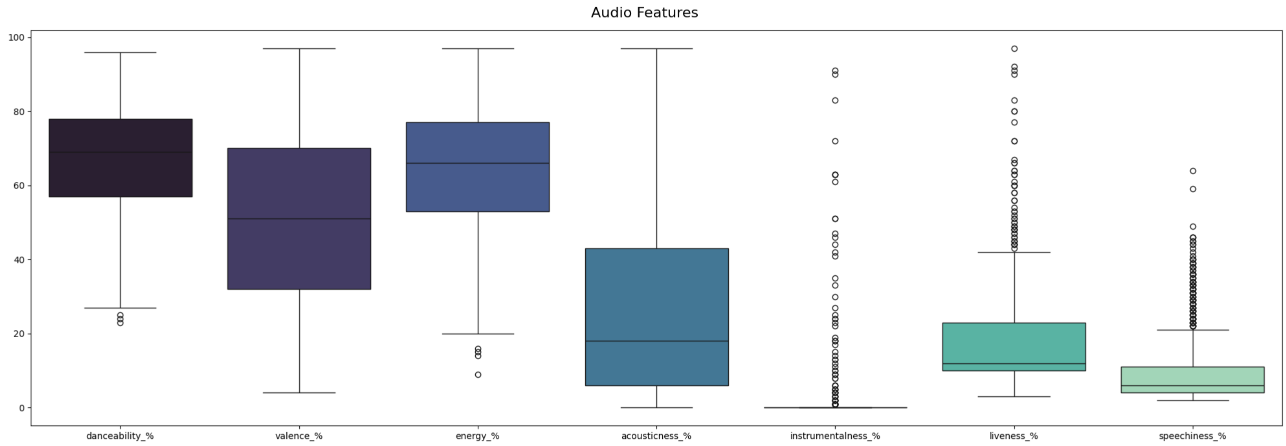
```
columns_to_drop = ['track_name', 'artists', 'mode']
out_check = df_copy.drop(columns=columns_to_drop)

columns = out_check.columns
```

Box plots of Playlist Presence, Chart Presence and Audio Features

```
In [ ]: # Plot 1: Playlist presence

plt.figure(figsize=(12, 6))
plt.suptitle('Playlist Presence', fontsize=16)

for i, column in enumerate(['in_spotify_playlists', 'in_apple_playlists', 'in_deezer_p
    plt.subplot(1, 3, i)
    plt.boxplot(out_check[column])
    plt.title(f'Box Plot of {column}')
    plt.ylabel('Count')

plt.tight_layout()
plt.show()

# Plot 2: Chart presence
plt.figure(figsize=(12, 6))
plt.suptitle('Chart Presence', fontsize=16)

for i, column in enumerate(['in_spotify_charts', 'in_apple_charts', 'in_deezer_charts'
    plt.subplot(1, 4, i)
    plt.boxplot(out_check[column])
    plt.title(f'Box Plot of {column}')
    plt.ylabel('Count')

plt.tight_layout()
plt.show()

# Plot 3: Audio features

plt.figure(figsize=(20, 7))
plt.suptitle('Audio Features', fontsize=16)

audio_features = ['danceability_%', 'valence_%', 'energy_%', 'acousticness_%', 'instru

# Use Seaborn's boxplot function to display multiple box plots
sns.boxplot(data=out_check[audio_features], palette='mako')
plt.tight_layout()
plt.show()
```







Correlation Analysis to find High Correlation

In [ ]:

```
columns_to_keep = df_copy.columns.difference(['track_name','artists','mode'])
df_selected = df_copy[columns_to_keep]

df_selected.corr()
```

Out[ ]:

	acousticness_%	artist_count	bpm	danceability_%	energy_%	in_app
acousticness_%	1.000000	-0.103223	-0.015914	-0.236012	-0.577502	-
artist_count	-0.103223	1.000000	-0.036736	0.207960	0.137882	-
bpm	-0.015914	-0.036736	1.000000	-0.146076	0.026973	
danceability_%	-0.236012	0.207960	-0.146076	1.000000	0.197616	-
energy_%	-0.577502	0.137882	0.026973	0.197616	1.000000	
in_apple_charts	-0.077860	-0.089804	0.034603	-0.025897	0.104120	
in_apple_playlists	-0.062320	-0.051329	0.027462	-0.028197	0.051658	
in_deezer_charts	-0.028517	-0.002622	0.031301	0.067460	0.093434	
in_deezer_playlists	-0.064188	-0.072244	-0.034531	-0.071539	0.065069	
in_shazam_charts	-0.078044	-0.074165	0.039972	-0.003990	0.111432	
in_spotify_charts	-0.057024	-0.020151	0.036597	0.030664	0.082617	
in_spotify_playlists	-0.065251	-0.102662	-0.017714	-0.107425	0.033533	
instrumentalness_%	0.044117	-0.049324	-0.004560	-0.089819	-0.037390	-
key	-0.018074	-0.000318	0.024242	0.028541	-0.005228	-
liveness_%	-0.048060	0.044970	-0.002786	-0.077736	0.116342	
released_day	-0.004992	-0.016583	-0.034405	0.049327	0.052256	
released_month	0.055019	0.038237	-0.039978	-0.046850	-0.083468	
released_year	-0.123366	0.088508	-0.006323	0.187294	0.095054	-
speechiness_%	-0.023848	0.119011	0.040275	0.185581	-0.004306	
streams	-0.004561	-0.136456	-0.002054	-0.104962	-0.026083	
valence_%	-0.082006	0.128441	0.041316	0.408211	0.358206	

21 rows × 21 columns

In [ ]:

```
correlation_matrix = df_selected.corr()

high_correlation_matrix = correlation_matrix[(correlation_matrix.abs() > 0.7) & (corre
high_correlations = (correlation_matrix.abs() >= 0.7) & (correlation_matrix.abs() < 1)

indices = [(i, j) for i in range(correlation_matrix.shape[0]) for j in range(correlati

print("Indices of correlations greater than or equal to 0.7 or less than or equal to -
print(indices)
```

```
Indices of correlations greater than or equal to 0.7 or less than or equal to -0.7:
[(5, 9), (6, 11), (6, 19), (8, 11), (9, 5), (9, 10), (10, 9), (11, 6), (11, 8), (11, 1
9), (19, 6), (19, 11)]
```

In [ ]:

```
non_nan_columns = high_correlation_matrix.dropna(axis=1, how='all').dropna(axis=0, how
non_nan_columns = non_nan_columns.fillna('')

non_nan_columns
```

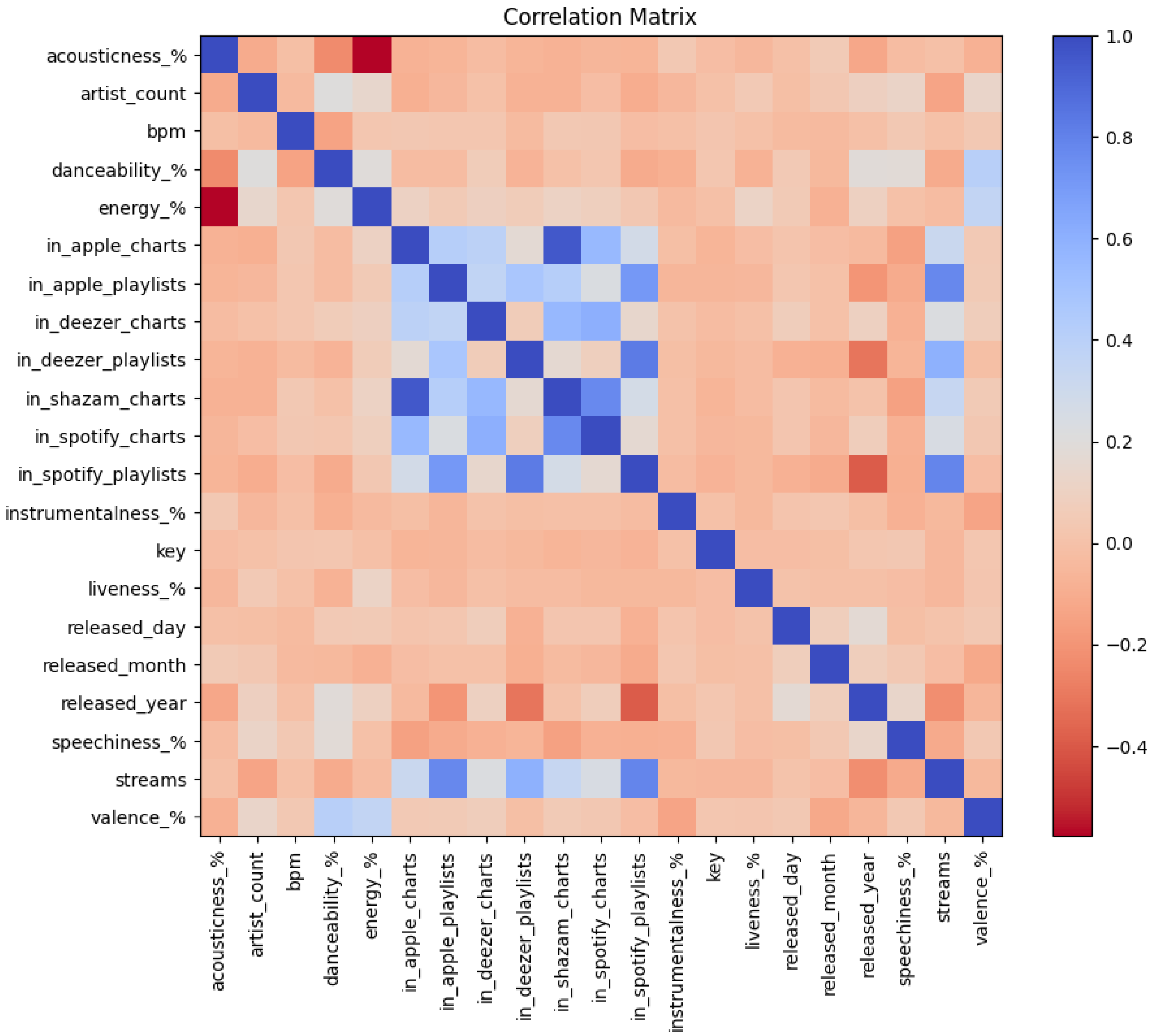
Out[ ]:

	in_apple_charts	in_apple_playlists	in_deezer_playlists	in_shazam_charts	in_
in_apple_charts				0.955245	
in_apple_playlists					
in_deezer_playlists					
in_shazam_charts	0.955245				
in_spotify_charts				0.766718	
in_spotify_playlists		0.708634	0.826524		
streams		0.773518			

Heatmap of Correlation Matrix

In [ ]:

```
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm_r', interpolation='nearest')
plt.colorbar()
plt.title('Correlation Matrix')
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotatio
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
plt.grid(False)
plt.show()
```



Queries

Query 1: Density Distribution of Release Date of Songs

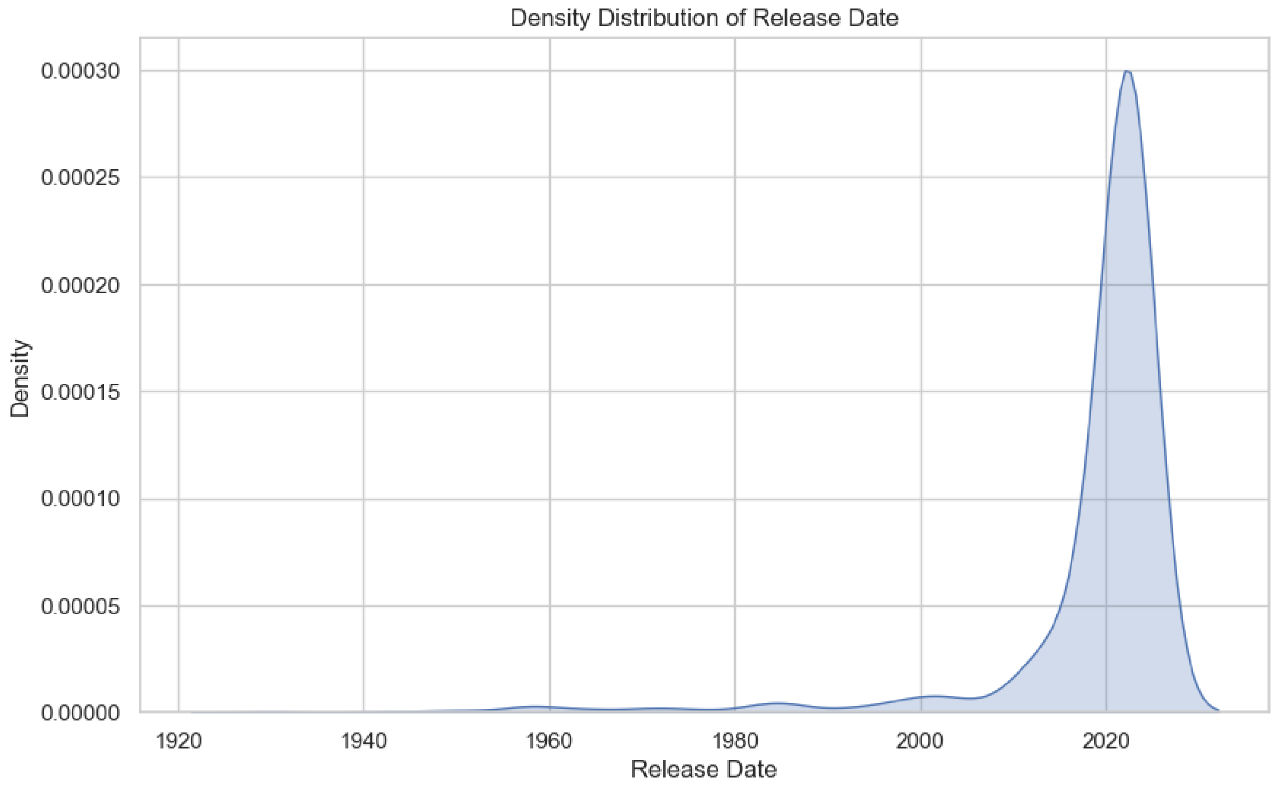
In [ ]:
```
df_with_datetime = df_copy.copy()
df_with_datetime['release_date'] = pd.to_datetime(df_with_datetime[['released_year', '
df_with_datetime[['track_name','release_date']]
```

Out[ ]:

	track_name	release_date
0	Seven (feat. Latto) (Explicit Ver.)	2023-07-14
1	LALA	2023-03-23
2	vampire	2023-06-30
3	Cruel Summer	2019-08-23
4	WHERE SHE GOES	2023-05-18
...	...	...
944	My Mind & Me	2022-11-03
945	Bigger Than The Whole Sky	2022-10-21
946	A Veces (feat. Feid)	2022-11-03
947	En La De Ella	2022-10-20
948	Alone	2022-11-04

949 rows × 2 columns

In [ ]:
```
#Kernel Density Estimate

sns.set(style="whitegrid")

plt.figure(figsize=(10, 6))
sns.kdeplot(df_with_datetime['release_date'], fill=True)

plt.xlabel('Release Date')
plt.ylabel('Density')
plt.title('Density Distribution of Release Date')

plt.show()
```



The above graph gives the distribution of Release Date of Songs which became viral in 2023.

- Newer songs tend to be streamed more, as shown in the graph, but older songs that dates back up to 1930 received a lot of streaming too.
- The older songs that received a lot of streaming could be classic songs, or songs that were brought back to popularity due to some usage in social media.

Splitting the track for multiple artists

In [ ]:
```
df_split_artists = df_copy.assign(artists=df_copy['artists'].str.split(',')).explode('
df_split_artists = df_split_artists.drop_duplicates(subset=['artists', 'track_name'])
df_split_artists = df_split_artists.apply(lambda x: x.str.strip() if x.dtype == "O" el
df_split_artists.reset_index(drop=True, inplace=True)
df_split_artists.to_csv('artist_split.csv')
df_split_artists
```

Out[ ]:

	track_name	artists	artist_count	released_year	released_month	released_day	in_spotify
0	Seven (feat. Latto) (Explicit Ver.)	Latto	2	2023	7	14	
1	Seven (feat. Latto) (Explicit Ver.)	Jung Kook	2	2023	7	14	
2	LALA	Myke Towers	1	2023	3	23	
3	vampire	Olivia Rodrigo	1	2023	6	30	
4	Cruel Summer	Taylor Swift	1	2019	8	23	
...	...	...	...	...	...	...	
1472	A Veces (feat. Feid)	Paulo Londra	2	2022	11	3	
1473	En La De Ella	Feid	3	2022	10	20	
1474	En La De Ella	Sech	3	2022	10	20	
1475	En La De Ella	Jhayco	3	2022	10	20	
1476	Alone	Burna Boy	1	2022	11	4	

1477 rows × 24 columns

To get the involvement of each artist in each track separately for the following analysis purposes, we split a track where multiple artists are present into new rows with single artist present in each

Query 2: Most Streamed Artists of 2023

In [ ]:

```
artist_streams = df_split_artists.groupby('artists')['streams'].sum()
artist_streams = artist_streams.sort_values(ascending=False)
df_artist_streams = pd.DataFrame(artist_streams)
df_artist_streams
```

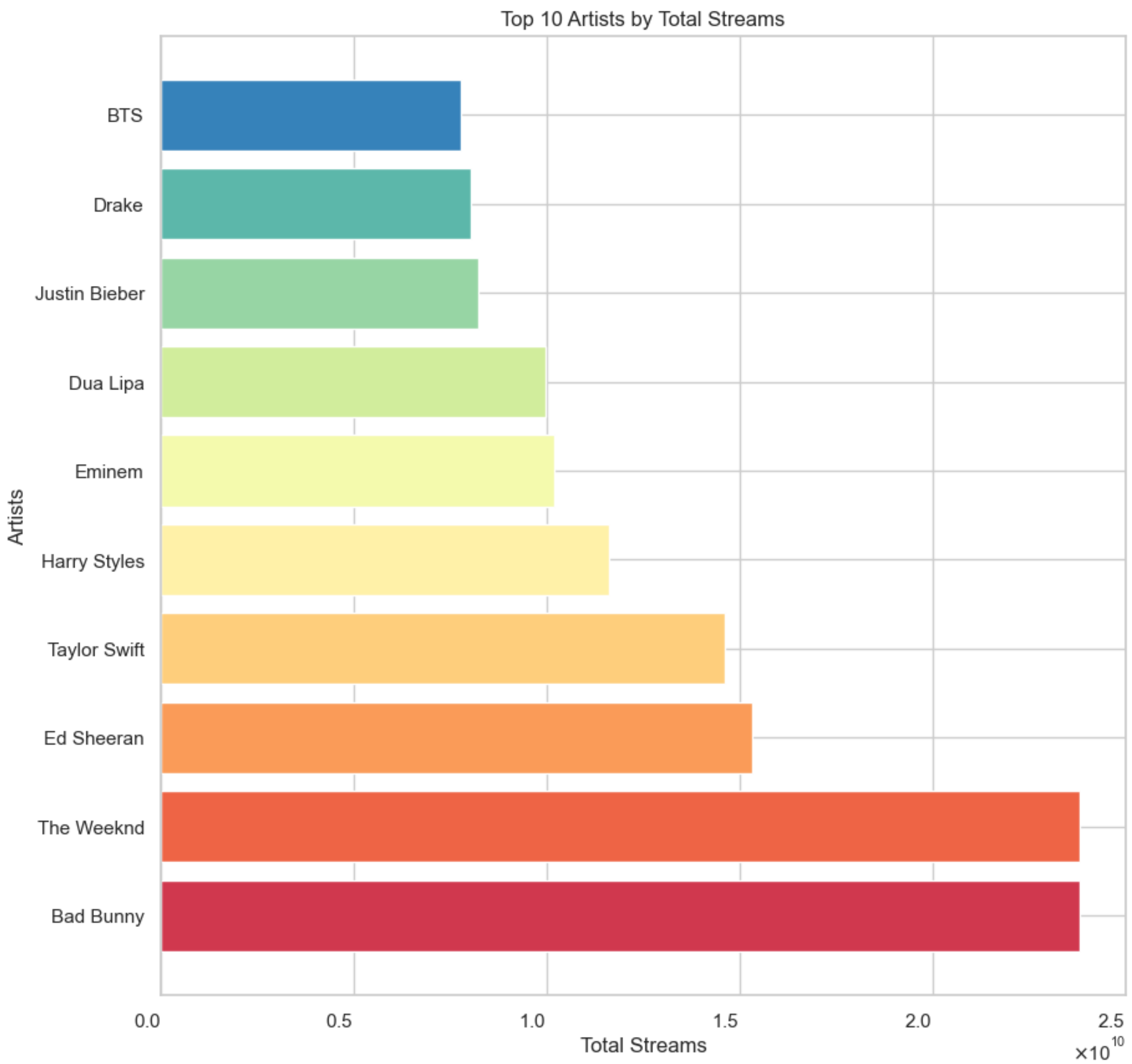
Out[ ]:

	streams
artists	
Bad Bunny	23813527270
The Weeknd	23799104954
Ed Sheeran	15316587718
Taylor Swift	14630378183
Harry Styles	11608645649
...	...
Toian	32761689
Beam	32761689
DJ 900	11956641
Sog	11599388
Sukriti Kakar	1365184

699 rows × 1 columns

In [ ]:

```
plt.figure(figsize=(10,10))
data = df_artist_streams.head(10)

colors = sns.color_palette("Spectral", n_colors=10)

plt.barh(data.index, data['streams'], color = colors)

plt.xticks(ha='right')

plt.gca().xaxis.set_major_formatter(ticker.ScalarFormatter(useMathText=True))


plt.ylabel('Artists')
plt.xlabel('Total Streams')
plt.title('Top 10 Artists by Total Streams')

plt.show()
```



- Bad Bunny is the most streamed artist of 2023, with 23813527270 (23.8 Billion) streams, followed closely by The Weeknd with 23799104954 (23.7 Billion) streams

Query 3: Artists Present in Most Playlists

In [ ]:
```
artist_playlists = df_split_artists.groupby('artists')[['in_spotify_playlists', 'in_ap

artist_playlists = artist_playlists.sum(axis=1).sort_values(ascending=False)

df_artist_playlists = pd.DataFrame({'Total_Playlists': artist_playlists})
df_artist_playlists
```

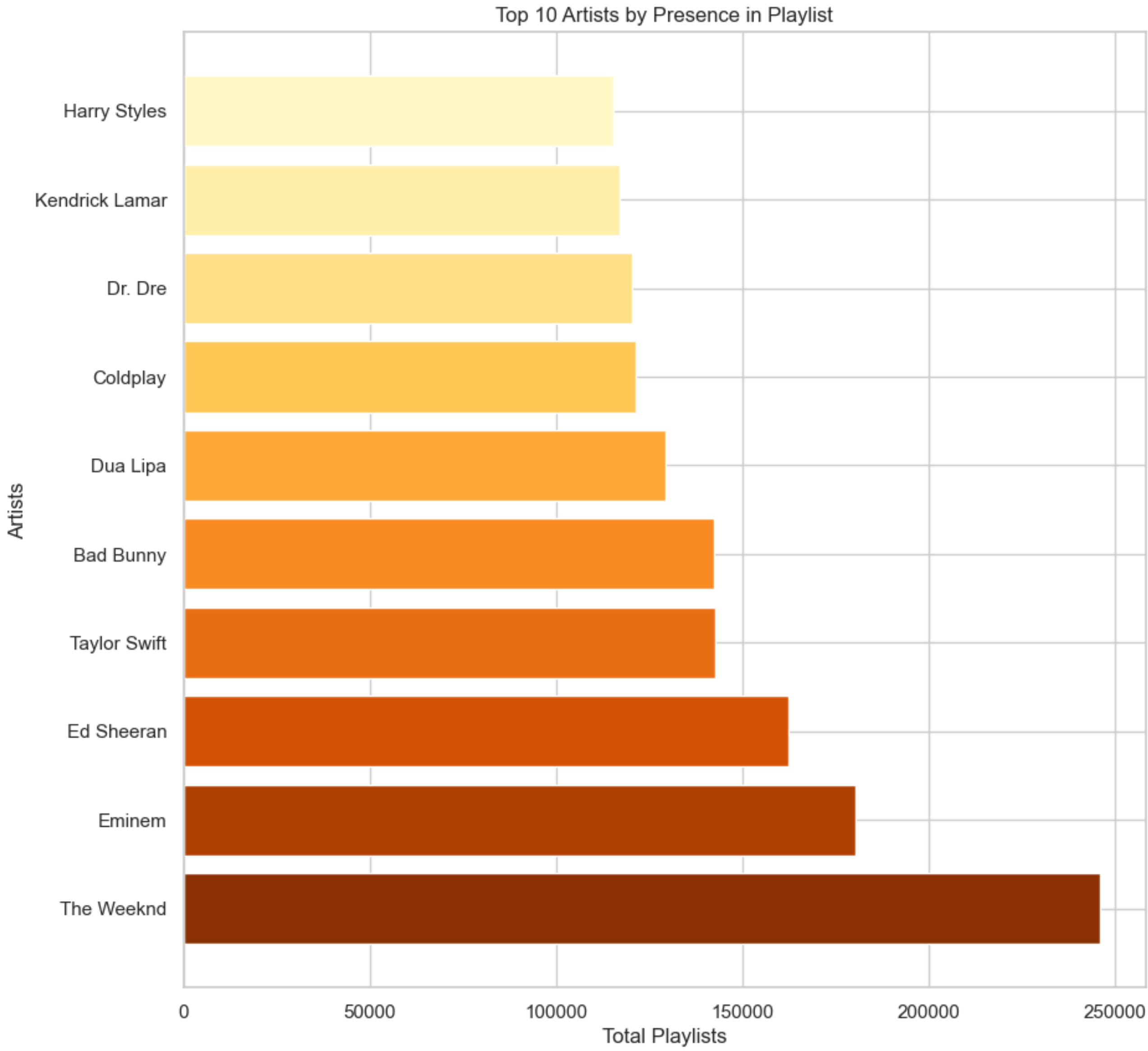
Out[ ]:

artists	Total_Playlists
The Weeknd	245924
Eminem	180355
Ed Sheeran	162567
Taylor Swift	142855
Bad Bunny	142461
...	...
Sukriti Kakar	153
Mahalini	138
Colde	115
Shubh	74
Jack Black	34

699 rows × 1 columns

In [ ]:

```
plt.figure(figsize=(10, 10))

colors = sns.color_palette("YlOrBr_r", n_colors=10)

data = df_artist_playlists.head(10)

plt.barh(data.index, data['Total_Playlists'], color = colors)
plt.xlabel('Total Playlists')
plt.ylabel('Artists')
plt.title('Top 10 Artists by Presence in Playlist')
plt.show()
```



- Even though Bad Bunny is the most streamed artist, The Weeknd is the artist present in most playlists. This indicates that The Weeknd has songs with higher repeat value and cover a wider genre

Query 4: Average Attributes of Songs of Top 15 Artists

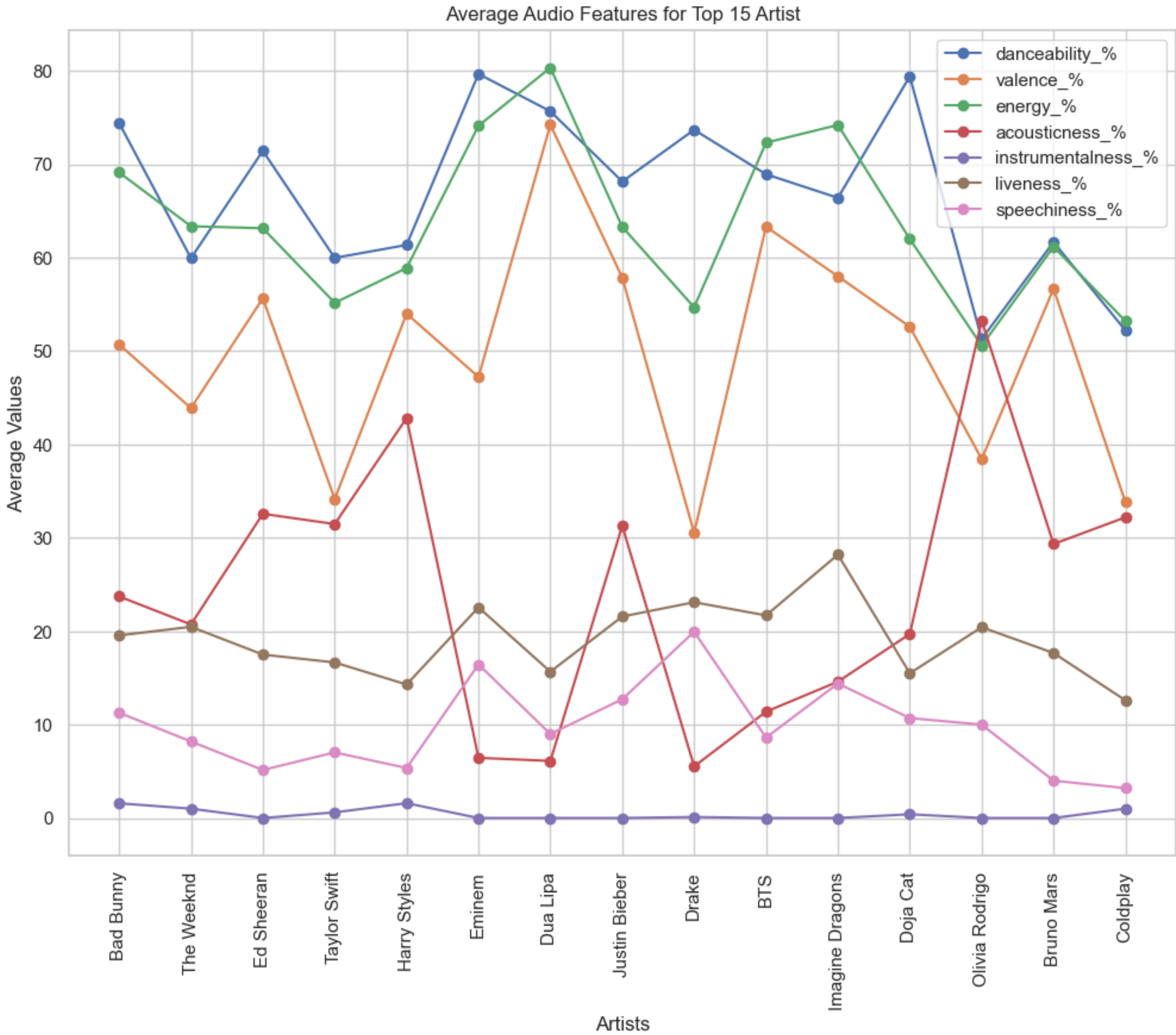
In [ ]:
```
top_artists = list(df_artist_streams.reset_index().nlargest(15, 'streams')['artists'])

artists_data = df_split_artists[df_split_artists['artists'].isin(top_artists)].copy()
artists_data['artists'] = pd.Categorical(artists_data['artists'], categories=top_artis
average_attributes = artists_data.groupby('artists', observed=False)[['danceability_%'
average_attributes.sort_index()
```

Out[ ]:

artists	danceability_%	valence_%	energy_%	acousticness_%	instrumentalness_%	liveness_
Bad Bunny	74.425000	50.700000	69.125000	23.725000	1.575000	19.5500
The Weeknd	59.944444	43.888889	63.361111	20.722222	1.000000	20.4722
Ed Sheeran	71.428571	55.642857	63.142857	32.571429	0.000000	17.5000
Taylor Swift	59.973684	34.157895	55.157895	31.473684	0.605263	16.6578
Harry Styles	61.352941	54.000000	58.882353	42.823529	1.588235	14.2941
Eminem	79.666667	47.222222	74.111111	6.444444	0.000000	22.5555
Dua Lipa	75.666667	74.222222	80.333333	6.111111	0.000000	15.6666
Justin Bieber	68.142857	57.857143	63.285714	31.285714	0.000000	21.5714
Drake	73.684211	30.526316	54.684211	5.526316	0.105263	23.1052
BTS	68.923077	63.307692	72.307692	11.384615	0.000000	21.6923
Imagine Dragons	66.400000	58.000000	74.200000	14.600000	0.000000	28.2000
Doja Cat	79.400000	52.600000	62.000000	19.700000	0.400000	15.5000
Olivia Rodrigo	51.285714	38.428571	50.571429	53.285714	0.000000	20.4285
Bruno Mars	61.666667	56.666667	61.166667	29.333333	0.000000	17.6666
Coldplay	52.200000	33.800000	53.200000	32.200000	1.000000	12.6000

In [ ]:

```
average_attributes.plot(kind='line', marker='o', figsize=(12, 9))

plt.xlabel('Artists')
plt.ylabel('Average Values')
plt.title('Average Audio Features for Top 15 Artist')
plt.xticks(rotation = 90)
plt.xticks(range(len(average_attributes.index)), average_attributes.index)

plt.show()
```



- The Top 15 Artists follow similar patterns for *instrumentalness*.

Query 5: Most and Least Streamed Songs of 2023

In [ ]:
```
top_songs_by_streams = df_copy.nlargest(15, 'streams')
top_songs_by_streams[['track_name', 'artists', 'streams']]
```

Out[ ]:

	track_name	artists	streams
55	Blinding Lights	The Weeknd	3703895074
179	Shape of You	Ed Sheeran	3562543890
86	Someone You Loved	Lewis Capaldi	2887241814
618	Dance Monkey	Tones and I	2864791672
41	Sunflower - Spider-Man: Into the Spider-Verse	Post Malone, Swae Lee	2808096550
162	One Dance	Drake, WizKid, Kyla	2713922350
84	STAY (with Justin Bieber)	Justin Bieber, The Kid Laroi	2665343922
140	Believer	Imagine Dragons	2594040133
723	Closer	The Chainsmokers, Halsey	2591224264
48	Starboy	The Weeknd, Daft Punk	2565529693
138	Perfect	Ed Sheeran	2559529074
71	Heat Waves	Glass Animals	2557975762
14	As It Was	Harry Styles	2513188493
691	Seo	Shawn Mendes, Camila Cabello	2484812918
324	Say You Won't Let Go	James Arthur	2420461338

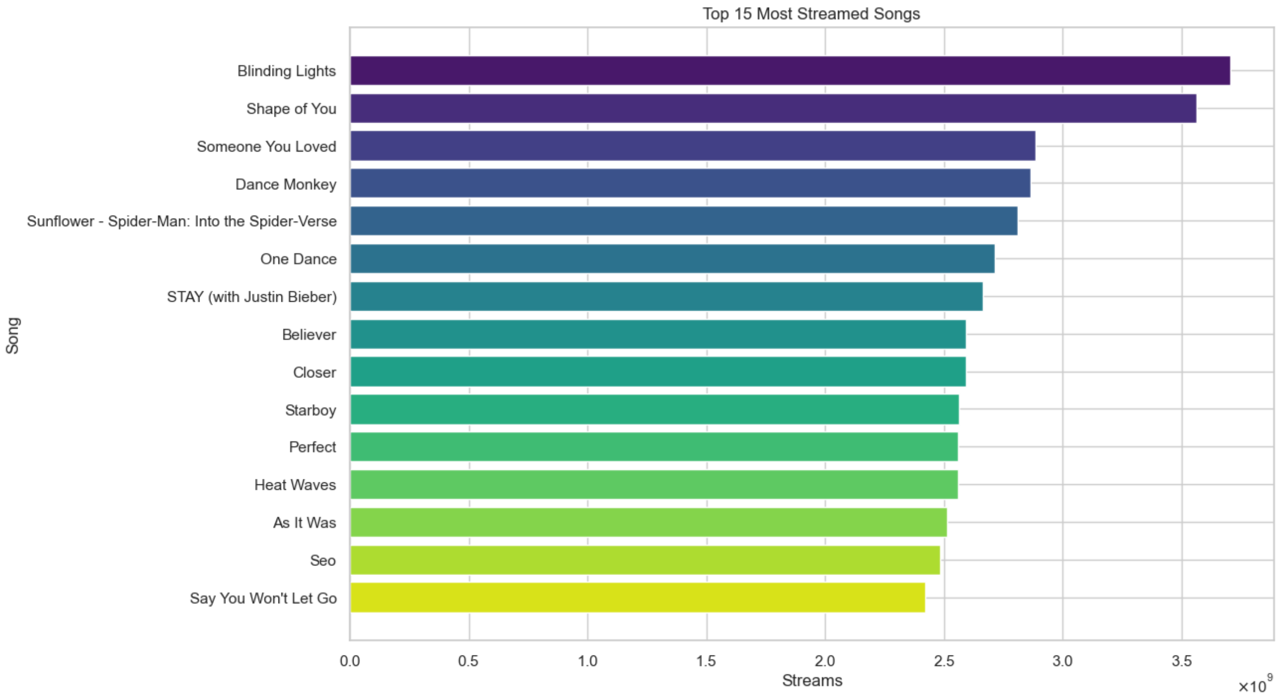
In [ ]:

```
bottom_songs_by_streams = df_copy.nsmallest(15, 'streams')
bottom_songs_by_streams[['track_name', 'artists', 'streams']]
```

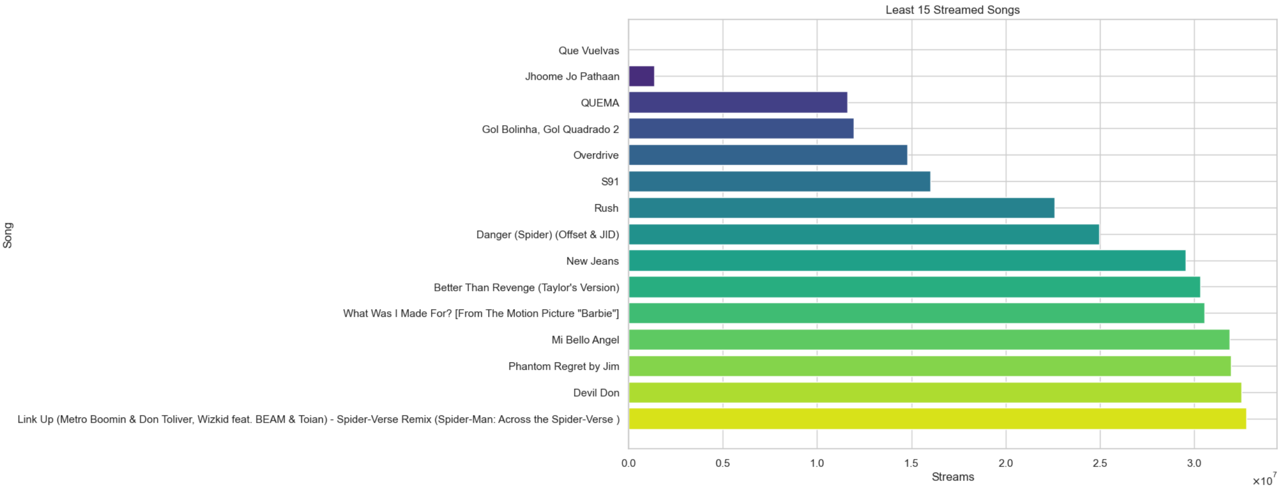
Out[ ]:

	track_name	artists	streams
123	Que Vuelvas	Carin Leon, Grupo Frontera	2762
393	Jhoome Jo Pathaan	Arijit Singh, Vishal Dadlani, Sukriti Kakar, V...	1365184
144	QUEMA	Sog, Ryan Castro, Peso Pluma	11599388
142	Gol Bolinha, Gol Quadrado 2	Mc Pedrinho, DJ 900	11956641
68	Overdrive	Post Malone	14780425
58	S91	Karol G	16011326
30	Rush	Troye Sivan	22581161
248	Danger (Spider) (Offset & JID)	Offset, JID	24975653
104	New Jeans	NewJeans	29562220
193	Better Than Revenge (Taylor's Version)	Taylor Swift	30343206
17	What Was I Made For? [From The Motion Picture ...	Billie Eilish	30546883
150	Mi Bello Angel	Natanael Cano	31873544
575	Phantom Regret by Jim	The Weeknd	31959571
379	Devil Don	Morgan Wallen	32526947
238	Link Up (Metro Boomin & Don Toliver, Wizkid fe...	WizKid, Toian, Metro Boomin, Don Toliver, Beam	32761689

In [ ]:
```
plt.figure(figsize=(12, 8))

colors = sns.color_palette("viridis", n_colors=15)

plt.barh(top_songs_by_streams['track_name'], top_songs_by_streams['streams'], color =

plt.gca().xaxis.set_major_formatter(ticker.ScalarFormatter(useMathText=True))

plt.xlabel('Streams')
plt.ylabel('Song')
plt.title('Top 15 Most Streamed Songs')
plt.gca().invert_yaxis()
plt.show()
```



In [ ]:
```
plt.figure(figsize=(12, 8))

colors = sns.color_palette("viridis", n_colors=15)

plt.barh(bottom_songs_by_streams['track_name'], bottom_songs_by_streams['streams'], co

plt.gca().xaxis.set_major_formatter(ticker.ScalarFormatter(useMathText=True))

plt.xlabel('Streams')
plt.ylabel('Song')
plt.title('Least 15 Streamed Songs')
plt.gca().invert_yaxis()
plt.show()
```



- Blinding Lights by The Weeknd is the most streamed song of 2023 with 3703895074 (3.7 Billion) streams, followed by Shape of You by Ed Sheeran with 3562543890 (3.5 Billion) streams.
- Que Vuelvas by Carin Leon and Grupo Frontera is the least streamed song of 2023 with 2762 streams.

Query 6: Number of Songs Released by Each Artist

In [ ]:
```
most_songs_artists = df_split_artists['artists'].value_counts()
most_songs_artists = pd.DataFrame(most_songs_artists)
most_songs_artists
```

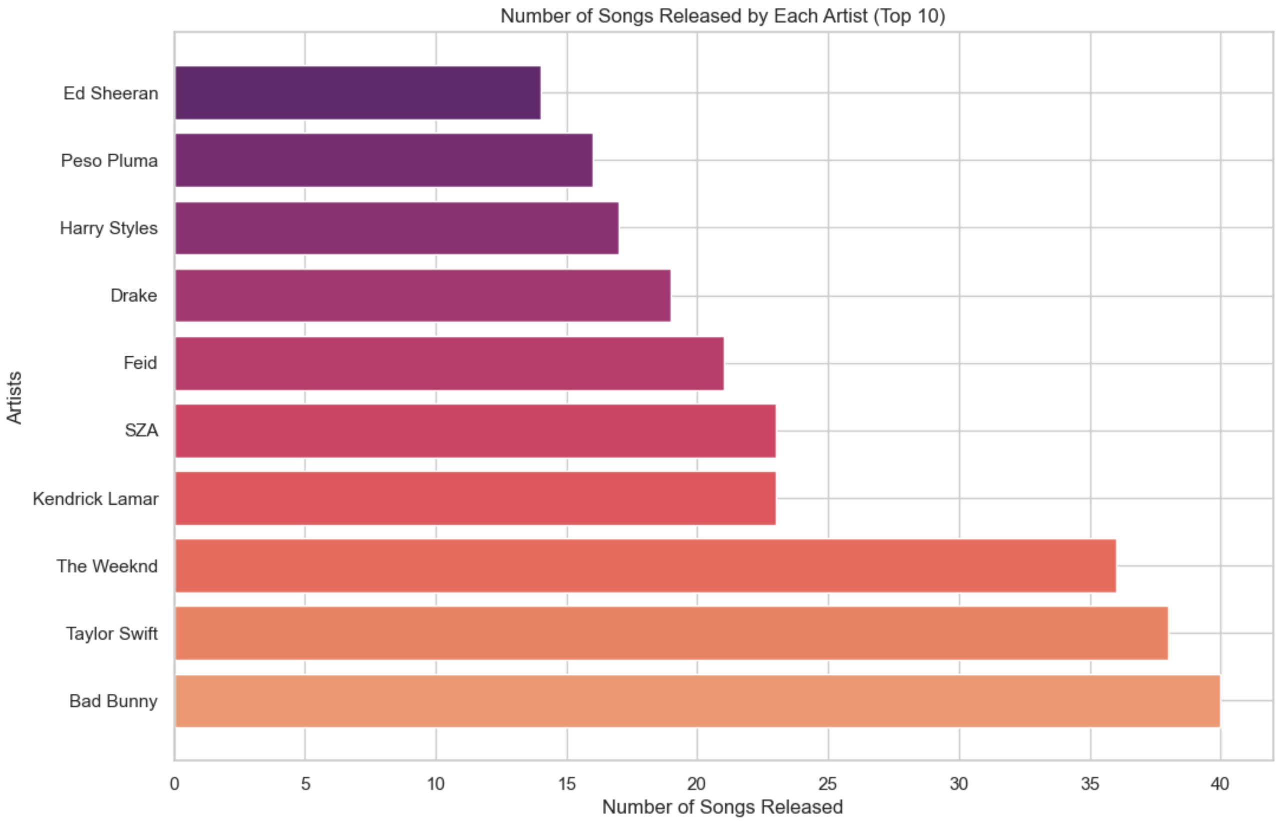
Out[ ]:

artists	count
Bad Bunny	40
Taylor Swift	38
The Weeknd	36
Kendrick Lamar	23
SZA	23
...	...
La Joaqui	1
Steve Aoki	1
FIFA Sound	1
Beach House	1
Selena Gomez	1

699 rows × 1 columns

In [ ]:
```
plt.figure(figsize=(12, 8))

colors = sns.color_palette("flare", n_colors=10)

data = most_songs_artists.head(10)

plt.barh(data.index, data['count'], color=colors)

plt.xlabel('Number of Songs Released')
```

```
plt.ylabel('Artists')
plt.title('Number of Songs Released by Each Artist (Top 10)')
plt.show()
```



- Bad Bunny is the artist whose songs became most popular in 2023 with 40 songs, followed by Taylor Swift with 38

Query 7: Top Songs by Playlist

In [ ]:
```
N = 10

top_spotify_songs = df_copy.nlargest(N, 'in_spotify_playlists')
top_apple_songs = df_copy.nlargest(N, 'in_apple_playlists')
top_deezer_songs = df_copy.nlargest(N, 'in_deezer_playlists')

print("Top Spotify Songs:")
print(top_spotify_songs[['track_name', 'in_spotify_playlists']].to_string(index=False)

print("\nTop Apple Music Songs:")
print(top_apple_songs[['track_name', 'in_apple_playlists']].to_string(index=False))

print("\nTop Deezer Songs:")
print(top_deezer_songs[['track_name', 'in_deezer_playlists']].to_string(index=False))
```

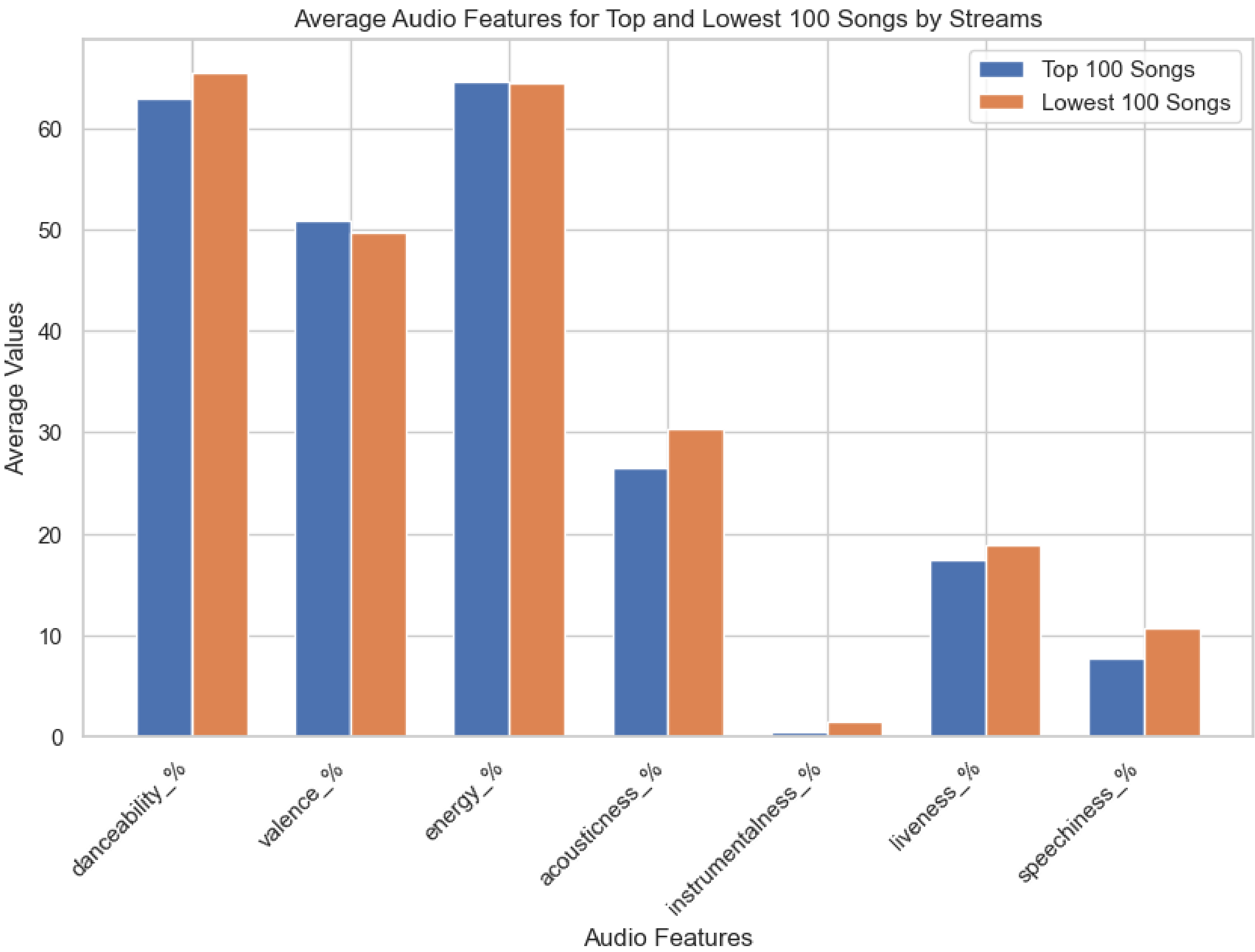
```
Top Spotify Songs:
                                track_name  in_spotify_playlists
                   Get Lucky - Radio Edit                 52898
                           Mr. Brightside                 51979
                  Wake Me Up - Radio Edit                 50887
Smells Like Teen Spirit - Remastered 2021                 49991
                               Take On Me                 44927
                          Blinding Lights                 43899
                                One Dance                 43257
             Somebody That I Used To Know                 42798
        Everybody Wants To Rule The World                 41751
                        Sweet Child O' Mine               41231

Top Apple Music Songs:
                track_name  in_apple_playlists
           Blinding Lights                 672
 One Kiss (with Dua Lipa)                  537
              Dance Monkey                 533
           Don't Start Now                 532
STAY (with Justin Bieber)                  492
                       Seo                 453
        Someone You Loved                  440
         Watermelon Sugar                  437
                 One Dance                 433
                 As It Was                 403

Top Deezer Songs:
                                track_name  in_deezer_playlists
Smells Like Teen Spirit - Remastered 2021                12367
                   Get Lucky - Radio Edit                 8215
                            The Scientist                 7827
                                     Numb                 7341
                             Shape of You                 6808
                               In The End                 6808
                                    Creep                 6807
                      Sweet Child O' Mine                 6720
                             Still D.R.E.                 6591
        Can't Hold Us (feat. Ray Dalton)                  6551
```

- Blinding lights, One Dance, Get Lucky - Radio Edit are the tracks that are in more than 1 platforms playlist

Query 8: Average Audio Features for Top and Lowest 10 Songs by Streams

In [ ]:
```
top_100_songs = df_copy.nlargest(100, 'streams')
lowest_100_songs = df_copy.nsmallest(100, 'streams')

columns_to_average = ['danceability_%', 'valence_%', 'energy_%', 'acousticness_%', 'in

top_100_average = top_100_songs[columns_to_average].mean()
lowest_100_average = lowest_100_songs[columns_to_average].mean()

fig, ax = plt.subplots(figsize=(10, 6))

bar_width = 0.35

bar_positions_top = range(len(top_100_average))
bar_positions_lowest = [pos + bar_width for pos in bar_positions_top]

ax.bar(bar_positions_top, top_100_average, width=bar_width, label='Top 100 Songs')
ax.bar(bar_positions_lowest, lowest_100_average, width=bar_width, label='Lowest 100 So
```

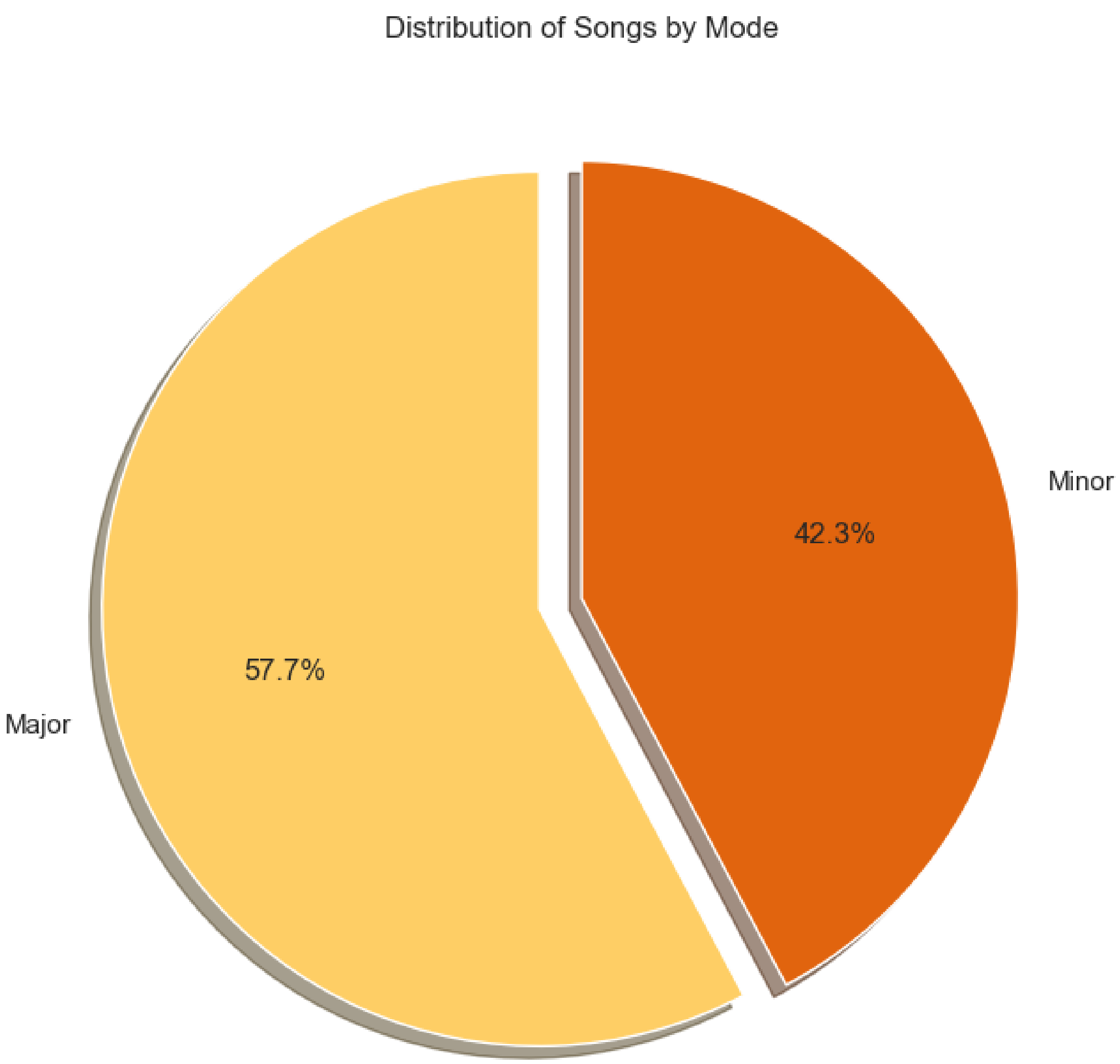
```
ax.set_xlabel('Audio Features')
ax.set_ylabel('Average Values')
ax.set_title('Average Audio Features for Top and Lowest 100 Songs by Streams')
ax.set_xticks([pos + bar_width / 2 for pos in bar_positions_top])
ax.set_xticklabels(columns_to_average, rotation=45, ha='right')
ax.legend()

plt.show()
```



- Listeners prefer to stream tracks that consists of more singing than acoustincess, speech and liveness.

Query 9: Distribution of Songs by Mode

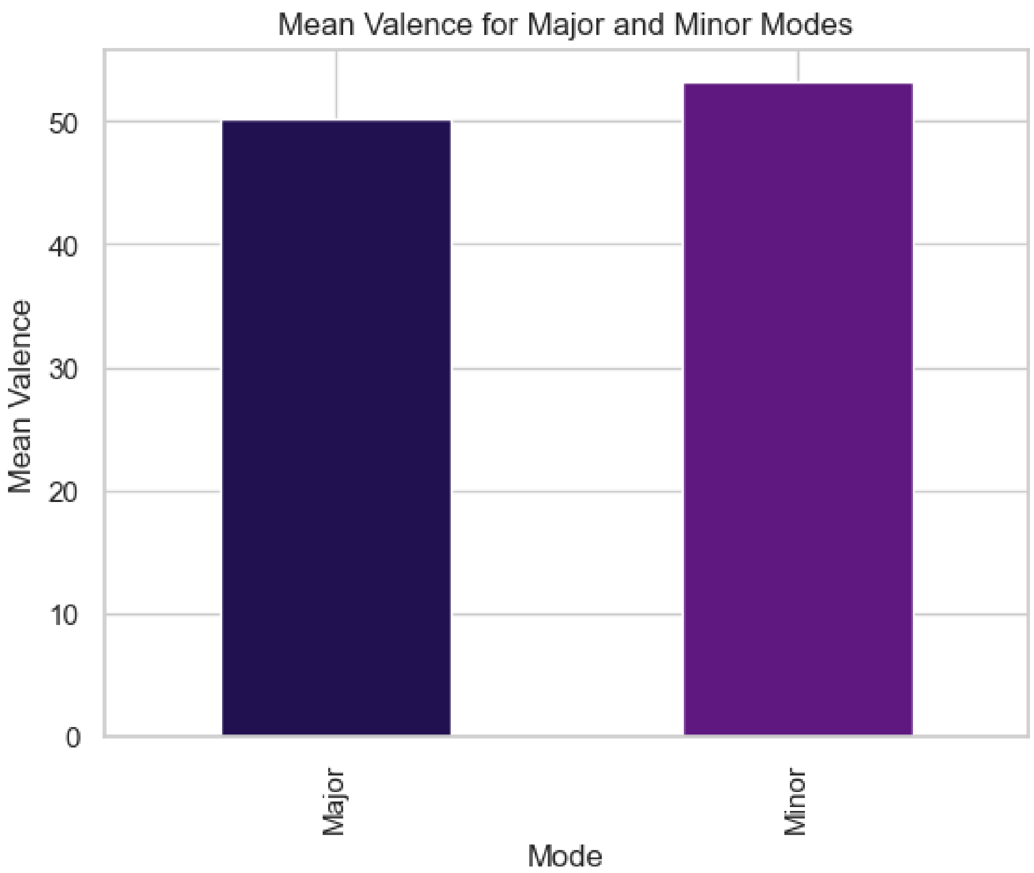
In [ ]:
```
# Assuming df is your DataFrame
mode_counts = df_copy['mode'].value_counts()

# Define a bright color palette
colors = sns.color_palette('YlOrBr', n_colors=len(mode_counts))

# Explode a slice to highlight
explode = [0.1] + [0] * (len(mode_counts) - 1)  # explode the first slice

# Plot a pie chart with styling
plt.figure(figsize=(8, 8))
plt.pie(mode_counts, labels=mode_counts.index, autopct='%1.1f%%', startangle=90,
        colors=colors, explode=explode, shadow=True)
plt.title('Distribution of Songs by Mode')
plt.show()
```



- Most of the songs are compossed on Major mode

Query 10: Mean Valence for Major and Minor Modes

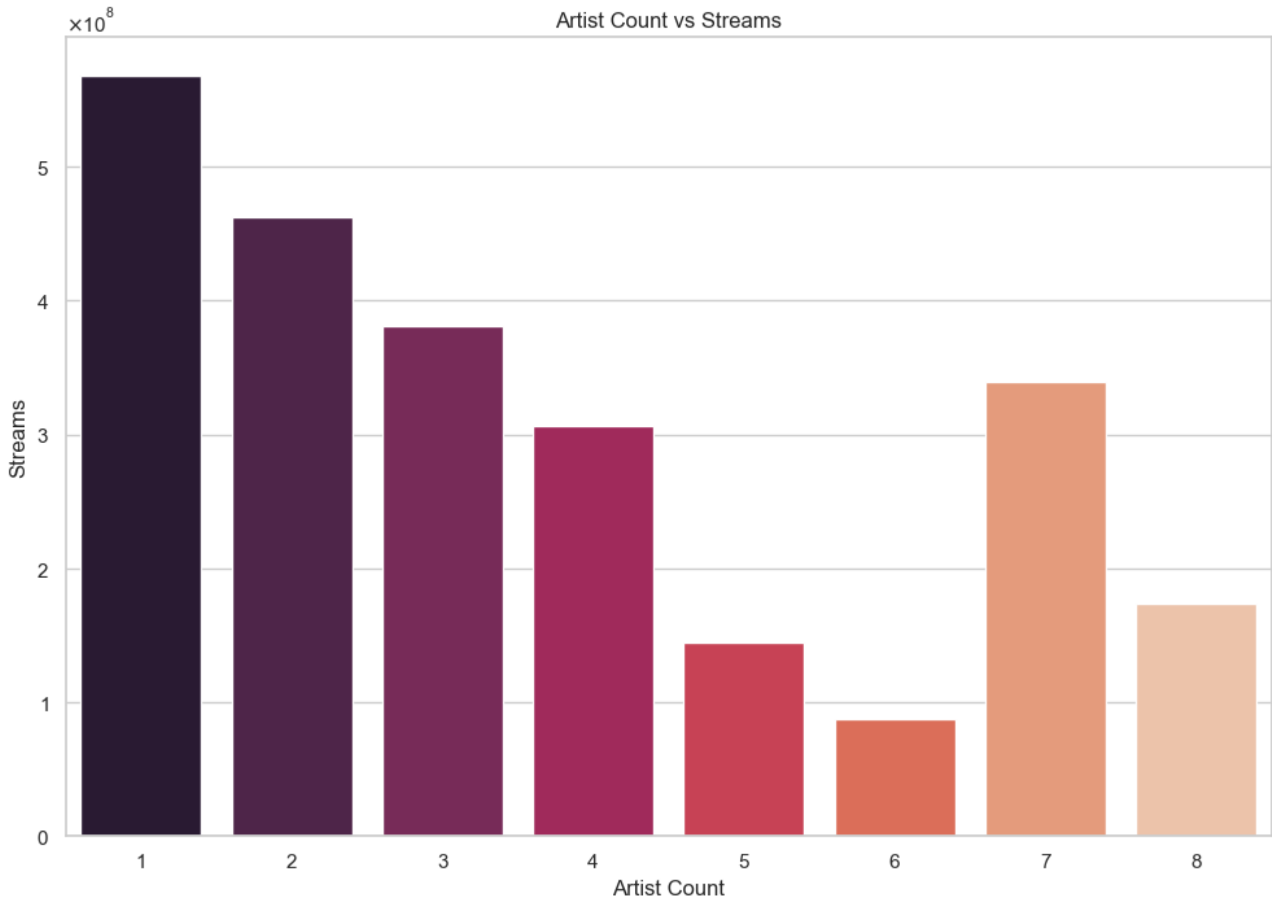
In [ ]:
```
mode_valence_means = df_copy.groupby('mode')['valence_%'].mean()

colors = sns.color_palette("magma")

mode_valence_means.plot(kind='bar', color = colors)

plt.xlabel('Mode')
plt.ylabel('Mean Valence')
plt.title('Mean Valence for Major and Minor Modes')

plt.show()
```



- Assumption: It is commonly believed that songs in major mode are generally more positive.
- Observation: Upon analyzing the graph, we found that songs in the minor mode exhibit a slightly higher level of valence. This contradicts the initial assumption.

Query 11: Single Artist v/s Multiple Artists

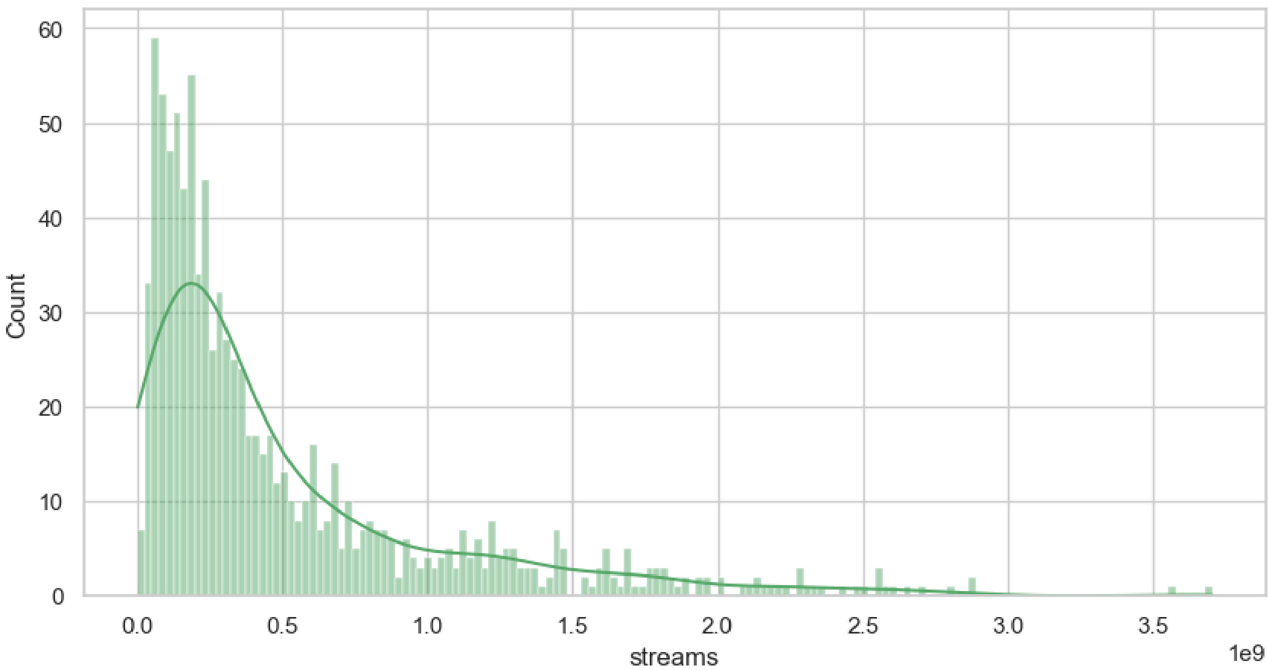
In [ ]:
```
df_artist_count = df_copy.groupby('artist_count')['streams'].mean().reset_index()

colors = sns.color_palette("rocket", n_colors=len(df_artist_count))


plt.figure(figsize=(12, 8))
sns.barplot(x='artist_count', y='streams', data=df_artist_count, palette=colors, hue='


plt.gca().yaxis.set_major_formatter(ticker.ScalarFormatter(useMathText=True))

plt.xlabel('Artist Count')
plt.ylabel('Streams')
plt.title('Artist Count vs Streams')

plt.show()
```



- Tracks with only 1 artist seem to be more popular and streamed more

Query 12: Visualisation of Streams and Count

In [ ]:
```
plt.figure(figsize=(10, 5))

sns.histplot(data=df_copy['streams'], color='g',bins=150, kde=True)

plt.show()
```



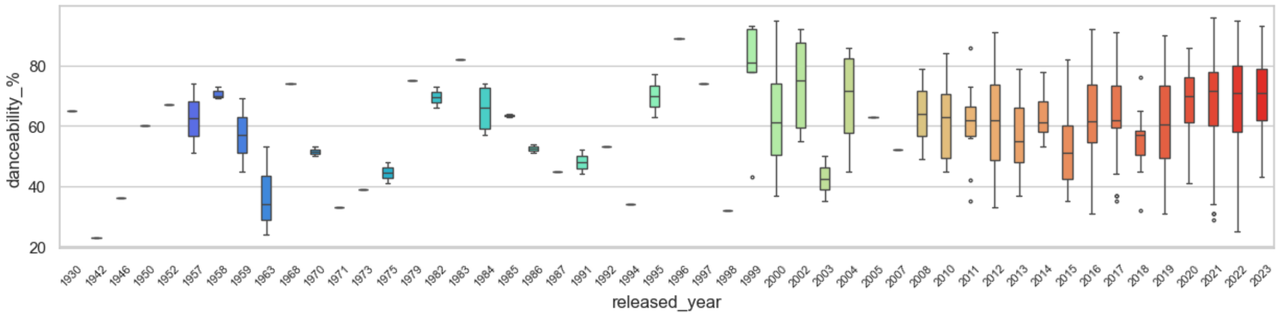
- Most of the songs have stream less than 0.5 Billion
- Songs with streams above 2.5 Billion are very rare

Query 13: Song Properties Throughout The Years

Danceability %

In [ ]:
```
plt.subplots(1,1, figsize=(15,3))
colors = sns.color_palette("rainbow",n_colors=50)
sns.boxplot(data=df_copy, x='released_year',y='danceability_%', width=0.4,fliersize=2,
plt.xticks(rotation=45,fontsize=8)
plt.show
```

Out[ ]: <function matplotlib.pyplot.show(close=None, block=None)>

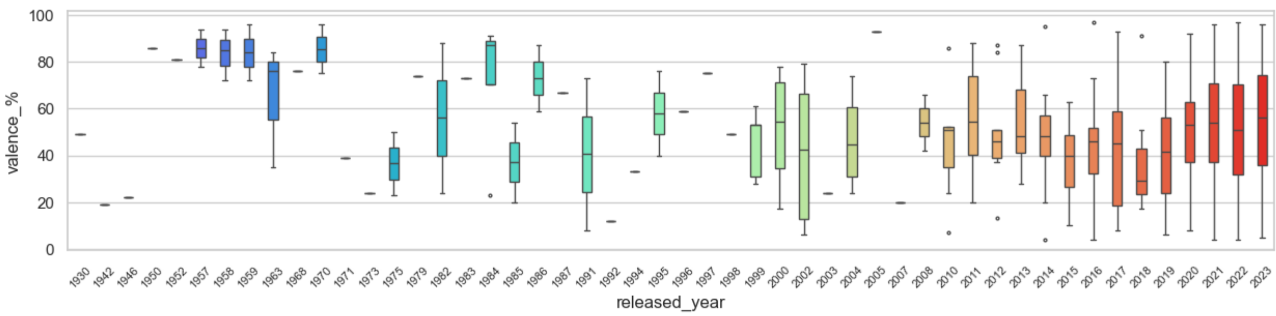


- Tracks that are released from the year 2020 to 2023 have almost similar median danceability, and almost similar interquartile range.
- Tracks that are released from 2008 to 2023 have wide range of danceability. It could be due to the majority of the top tracks were released in these years.
- The most danceable track in the top streamed songs was released in 2021.
- The least danceable track in the top streamed songs was released in 1942.

Valence %

In [ ]:
```
plt.subplots(1,1, figsize=(15,3))
colors = sns.color_palette("rainbow",n_colors=50)
sns.boxplot(data=df_copy, x='released_year',y='valence_%', width=0.4,fliersize=2,palet
plt.xticks(rotation=45,fontsize=8)
plt.show
```

Out[ ]: <function matplotlib.pyplot.show(close=None, block=None)>



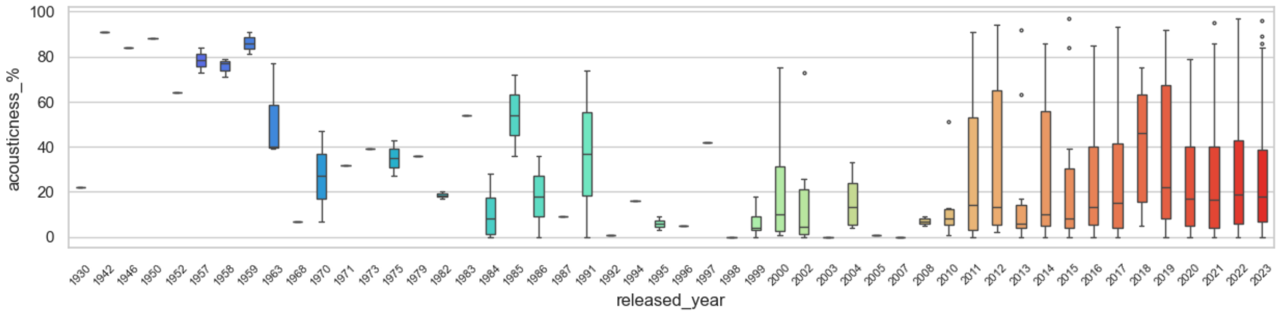
- Tracks from 2020 to 2023 shows wide variety of moods in the top streamed songs, long whiskers extending from low valence to high valence, and median values at

approximately 50%.
- Tracks from 2011 to 2023 has median valence at approximately 40 to 50%, with the exception of 2018. The range is also at the middle of the chart, ranging 20 to 70%, which shows the neutrality of the mood in the top streamed songs.

Acousticness %

In [ ]:
```
plt.subplots(1,1, figsize=(15,3))
colors = sns.color_palette("rainbow",n_colors=50)
sns.boxplot(data=df_copy, x='released_year',y='acousticness_%', width=0.4,fliersize=2,
plt.xticks(rotation=45,fontsize=8)
plt.show
```

Out[ ]: <function matplotlib.pyplot.show(close=None, block=None)>

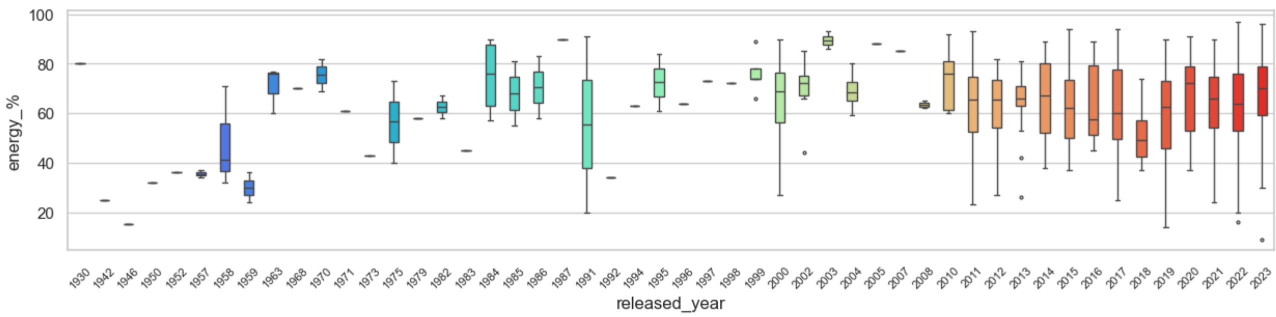


- Tracks from 2011 to 2023 contains high variety of songs with different acousticness values, as shown in the long whiskers and long interquartile range.
- Older tracks seem to fall under a small range of acousticness levels.

Energy %

In [ ]:
```
plt.subplots(1,1, figsize=(15,3))
colors = sns.color_palette("rainbow",n_colors=50)
sns.boxplot(data=df_copy, x='released_year',y='energy_%', width=0.4,fliersize=2,palett
plt.xticks(rotation=45,fontsize=8)
plt.show
```

Out[ ]: <function matplotlib.pyplot.show(close=None, block=None)>

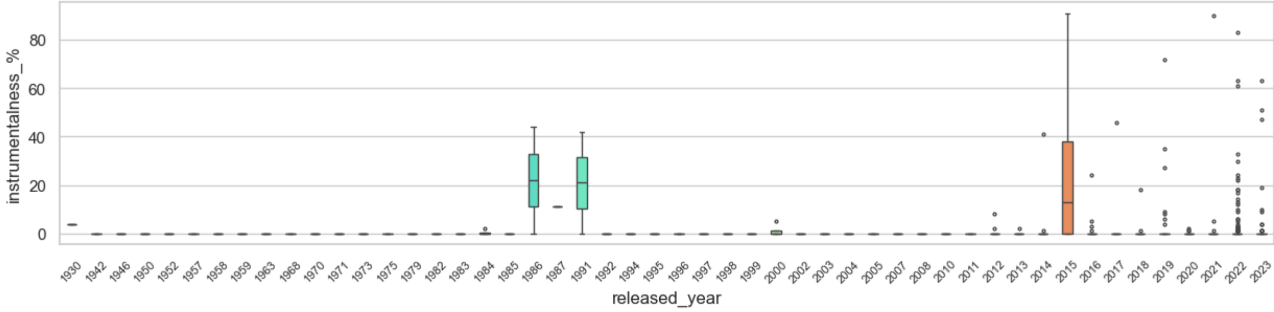


- Top tracks from 2011 to 2023 contains a wide range of tracks from energetic to less energetic, but with median values at the 50 to 60% energy percentage.
- The median values of the most streamed tracks are 50% or more, with the exception of 1958, 1959, and years with single tracks that made it to the most streamed. This shows that listeners prefer to listen to energetic tracks.

Instrumentalness %

In [ ]:
```
plt.subplots(1,1, figsize=(15,3))
colors = sns.color_palette("rainbow",n_colors=50)
sns.boxplot(data=df_copy, x='released_year',y='instrumentalness_%', width=0.4,fliersiz
plt.xticks(rotation=45,fontsize=8)
plt.show
```

Out[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

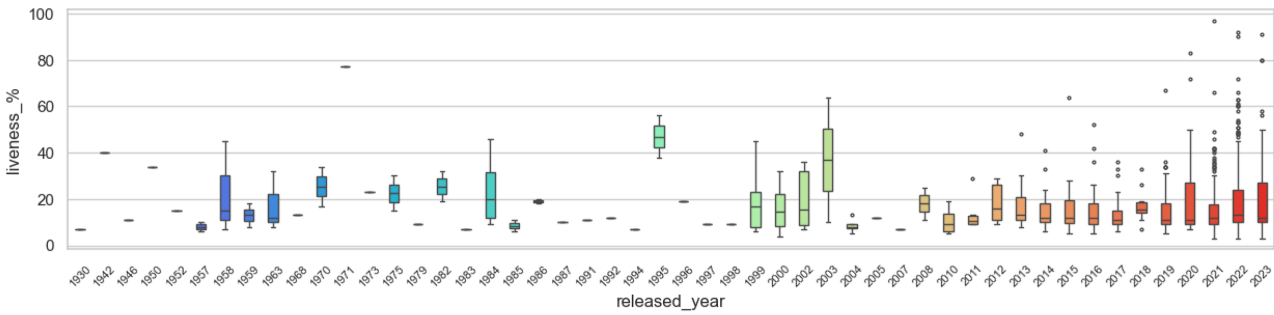


- The boxplot shows that majority of the most streamed tracks contains less instrumentalness levels, with some tracks that falls under the instrumental category identifies as outliers.
- Tracks released from 1986, 1991, and 2015, however, contains tracks that have considerably high instrumentalness levels, especially in 2015 where the boxplot whiskers reached the highest instrumentalness level.

Liveness %

In [ ]:
```
plt.subplots(1,1, figsize=(15,3))
colors = sns.color_palette("rainbow",n_colors=50)
sns.boxplot(data=df_copy, x='released_year',y='liveness_%', width=0.4,fliersize=2,pale
plt.xticks(rotation=45,fontsize=8)
plt.show
```

Out[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

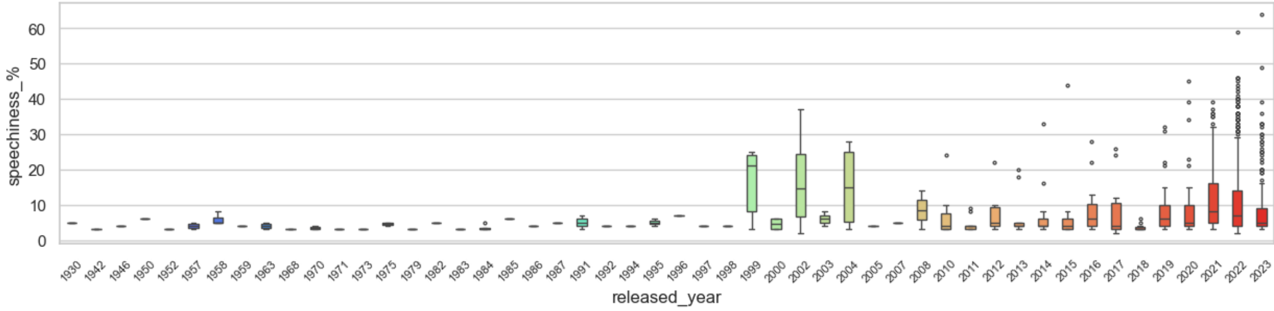


- A huge number of top streamed tracks have values of less than 50% liveness, with whiskers and interquartile range falling below 50%.
- Tracks which are performed live fall to the outliers, which means than listeners prefer to listen to recorded tracks.

Speechiness %

In [ ]:
```
plt.subplots(1,1, figsize=(15,3))
colors = sns.color_palette("rainbow",n_colors=50)
sns.boxplot(data=df_copy, x='released_year',y='speechiness_%', width=0.4,fliersize=2,p
plt.xticks(rotation=45,fontsize=8)
plt.show
```

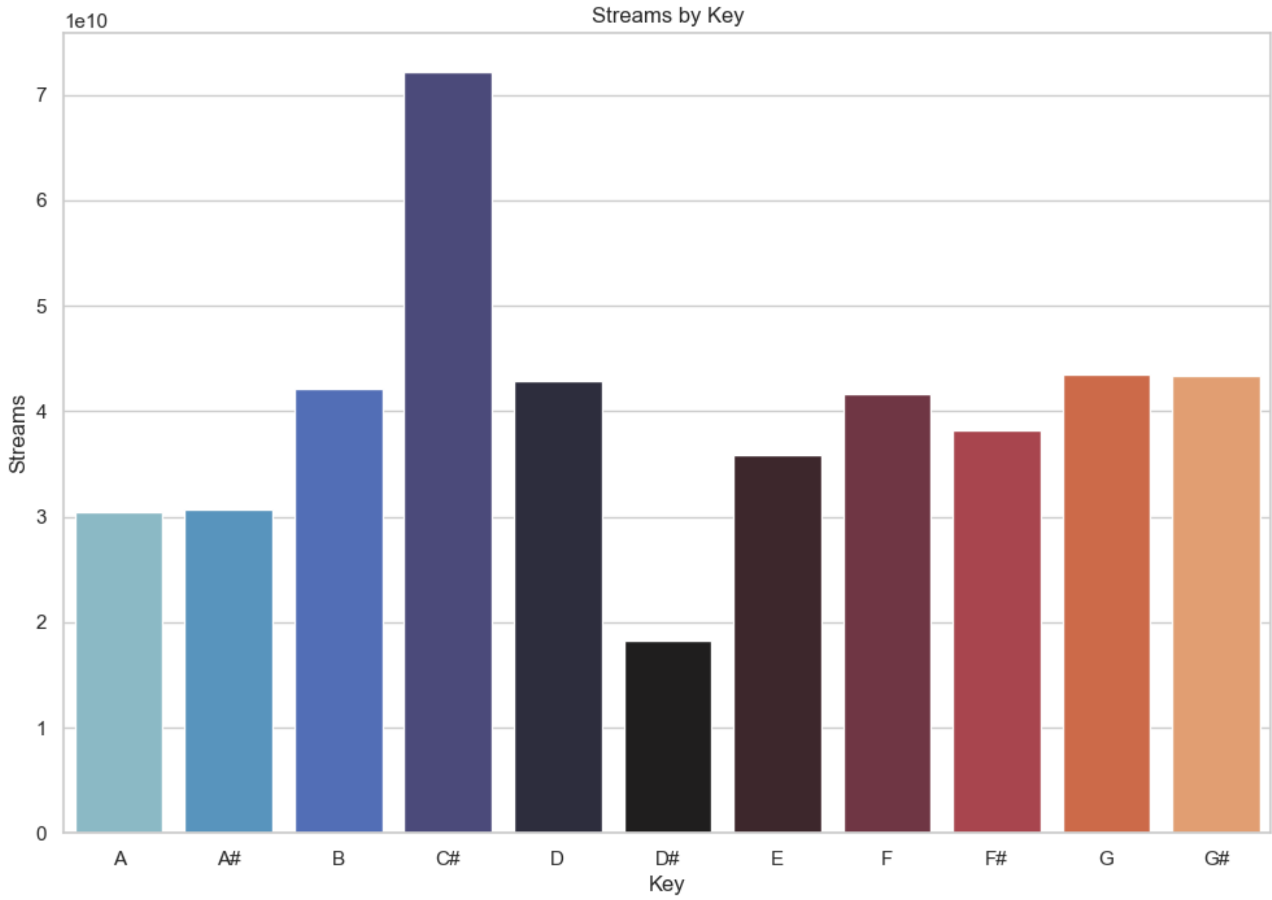
Out[ ]: <function matplotlib.pyplot.show(close=None, block=None)>



- Listeners prefer to listen to tracks with less speechiness or spoken words, as shown in the boxplot, where ticks and interquartile range fall below 30 to 40%, and outliers are rarely be seen above 50%.

Query 14: Most Streamed Key

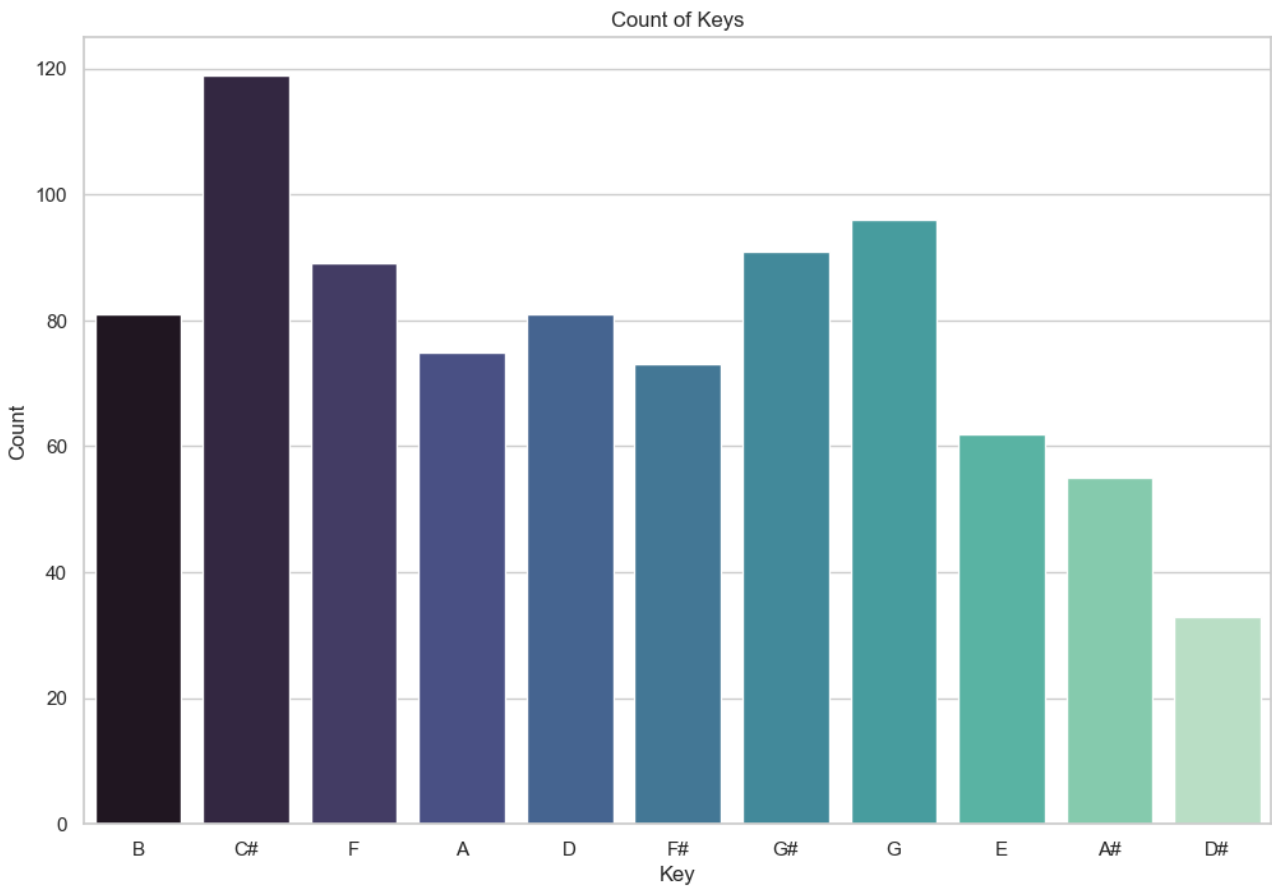
In [ ]:
```
plt.figure(figsize=(12, 8))
colorsKey = sns.color_palette("icefire", n_colors=11)

df_plot = df_copy.copy()
inverse_pitch_mapping = {v: k for k, v in pitch_class_mapping.items()}

df_plot['key'] = df_plot['key'].map(inverse_pitch_mapping)
df_key_sum = df_plot.groupby('key')['streams'].sum().reset_index()

sns.barplot(x='key', y='streams', data=df_key_sum, errorbar=None, palette=colorsKey, h

plt.xlabel('Key')
plt.ylabel('Streams')
plt.title('Streams by Key')
plt.show()
```



- C# is the most streamed key
- There are no streamed music with the key C
- The amount of streams for other keys have small variations

Query 15: Count of the occurrences of each key in the *key* column of the DataFrame

In [ ]:
```
plt.figure(figsize=(12, 8))
sns.countplot(x='key', data=df_plot, palette='mako', hue='key', legend=False)

plt.xlabel('Key')
plt.ylabel('Count')
plt.title('Count of Keys')

plt.show()
```



- C# is the most used key through out the dataset

Conclusion

Our analysis of music streaming data in 2023 reveals some interesting patterns and trends. Here are the key takeaways:

Popularity:

- Newer songs tend to be streamed more, but older classics and songs revived through social media also gain significant traction.
- Bad Bunny reigns supreme as the most streamed artist, while The Weeknd boasts the most playlist appearances, suggesting higher repeatability and genre diversity.
- Blinding Lights by The Weeknd takes the crown as the most streamed song, followed by Shape of You by Ed Sheeran.

Streaming trends:

- Listeners prefer songs with more singing and less speech, acousticness, and liveness.
- Major mode songs dominate, though minor mode exhibits surprisingly high valence, challenging the assumption of their negativity.
- Tracks with only 1 artist are more popular and streamed more.
- Songs with less than 0.5 Billion streams are most common, with those exceeding 2.5 Billion being rare.

Release date and characteristics:

- Songs released between 2020 and 2023 exhibit similar median and interquartile range for danceability.
- Tracks from 2008 to 2023 show a wider range of danceability, possibly due to the concentration of top tracks in this period.
- Tracks from 2020 to 2023 showcase a wider variety of moods, while 2011 to 2023 lean towards neutrality.
- Tracks from both periods exhibit a wide range of acousticness and energy levels, suggesting diversity in preferences.
- Older tracks appear to fall under a smaller range of acousticness levels.
- Listeners generally prefer energetic tracks, with most streamed tracks exceeding 50% energy percentage.
- Top streamed tracks generally have lower instrumentalness levels, with outliers identified as instrumental pieces.
- Tracks from specific years (1986, 1991, 2015) contain tracks with considerably higher instrumentalness levels.
- The majority of top streamed tracks have less than 50% liveness, indicating a preference for recorded music over live performances.
- Listeners favor songs with less speechiness, with most falling below 30-40% and outliers rarely exceeding 50%.

Key and Mode:

- C# emerges as the most streamed key, while C has no streamed tracks.
- Other keys show minimal variations in streaming numbers.
- C# is also the most used key across the entire dataset.

Overall Analysis

Our analysis unveils valuable insights into music streaming preferences in 2023. Listeners seem to favor newer releases but also appreciate classics and social media-driven trends. Singing, major mode, and moderate energy levels resonate well, while instrumentalness, liveness, and speechiness play a smaller role. Specific years present unique trends in danceability, mood, and instrumentalness, suggesting evolving preferences over time. C# reigns supreme as the most streamed key, highlighting its popularity among artists and audiences.

These findings offer valuable information for artists, music platforms, and anyone interested in understanding the current landscape of music streaming. They can be used to inform content creation, recommendation algorithms, and marketing strategies to better cater to listener preferences and drive engagement.