

Spotify Dataset EDA with Insights

```
# Import necessary libraries for data manipulation and visualization.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

/home/zero/.local/lib/python3.12/site-packages/pandas/core/arrays/masked.py:61: UserWarning: Pandas requires version '1.3.6'
from pandas.core import (
```

```
# Load the Spotify dataset and view its shape and first few rows.
df = pd.read_csv('spotify_dataset.csv')
print('Shape:', df.shape)
df.head()
```


Shape: (1556, 23)

	Index	Highest Charting Position	Number of Times Charted	Week of Highest Charting	Song Name	Streams	Artist	Artist Followers		Song ID	Genre	...	Danceabi
0	1	1	8	2021-07-23--2021-07-30	Beggin'	48,633,449	Måneskin	3377762	3Wrjm47oTz2sjlgck11l5e		['indie rock italiano', 'italian pop']	...	(
1	2	2	3	2021-07-23--2021-07-30	STAY (with Justin Bieber)	47,248,719	The Kid LAROI	2230022	5HCyWIXZPP0y6Gqq8TgA20		['australian hip hop']	...	(
2	3	1	11	2021-06-25--2021-07-02	good 4 u	40,162,559	Olivia Rodrigo	6266514	4ZtFanR9U6ndgddUvNcjcG		['pop']	...	(
3	4	3	5	2021-07-02--2021-07-09	Bad Habits	37,799,456	Ed Sheeran	83293380	6PQ88X9TkUIAUJZJHW2upE		['pop', 'uk pop']	...	(
4	5	5	1	2021-07-23--2021-07-30	INDUSTRY BABY (feat. Jack Harlow)	33,948,454	Lil Nas X	5473565	27NovPIUIRrOZoCHxABJwK		['lgbtq+ hip hop', 'pop rap']	...	(

5 rows × 23 columns

Drop columns not needed for EDA

```
# Drop columns that are IDs or not useful for exploratory analysis.
drop_cols = ['Index', 'Song ID', 'Chord', 'Week of Highest Charting']
df = df.drop(columns=[c for c in drop_cols if c in df.columns])
df.head()
```



	Highest Charting Position	Number of Times Charted	Song Name	Streams	Artist	Artist Followers	Genre	Release Date	Weeks Charted	Popularity	Danceability	Energy	Lo
0	1	8	Beggin'	48,633,449	Måneskin	3377762	['indie rock italiano', 'italian pop']	2017-12-08	2021-07-23--2021-07-30 2021-07-16-2021-07-23...	100	0.714	0.8	
1	2	3	STAY (with Justin Bieber)	47,248,719	The Kid LAROI	2230022	['australian hip hop']	2021-07-09	2021-07-23--2021-07-30 2021-07-16-2021-07-23...	99	0.591	0.764	
2	1	11	good 4 u	40,162,559	Olivia Rodrigo	6266514	['pop']	2021-05-21	2021-07-23--2021-07-30 2021-07-16-2021-07-23...	99	0.563	0.664	
3	3	5	Bad Habits	37,799,456	Ed Sheeran	83293380	['pop', 'uk pop']	2021-06-25	2021-07-23--2021-07-30 2021-07-16-2021-07-23...	98	0.808	0.897	
4	5	1	INDUSTRY BABY (feat. Jack Harlow)	33,948,454	Lil Nas X	5473565	['lgbtq+ hip hop', 'pop rap']	2021-07-23	2021-07-23--2021-07-30	96	0.736	0.704	

Convert numeric-like columns stored as text

```
# Convert columns stored as text (with commas, etc.) to numeric type for analysis.
numeric_like = [
    'Streams','Artist Followers','Popularity','Danceability','Energy',
    'Loudness','Speechiness','Acousticness','Liveness','Tempo','Duration (ms)','Valence',
    'Number of Times Charted'
]
for col in numeric_like:
    if col in df.columns:
        s = df[col].astype(str).str.replace(',',' ', case=False)
        s = s.str.strip()
        df[col] = pd.to_numeric(s, errors='coerce')

# Fix 'Weeks Charted' column: if it's a string with date ranges separated by \n, count the number of weeks for each song.
if 'Weeks Charted' in df.columns:
    df['Weeks Charted'] = df['Weeks Charted'].apply(
        lambda x: len(str(x).split('\n')) if pd.notnull(x) and isinstance(x, str) else pd.to_numeric(x, errors='coerce')
    )
```


Feature Engineering

```
# Feature engineering: extract year, create total chart presence, convert duration to minutes.
if 'Release Date' in df.columns:
    df['Release Date'] = pd.to_datetime(df['Release Date'], errors='coerce')
    df['Release Year'] = df['Release Date'].dt.year

if 'Weeks Charted' in df.columns and 'Number of Times Charted' in df.columns:
    df['Total Chart Presence'] = df['Weeks Charted'].fillna(0) + df['Number of Times Charted'].fillna(0)


if 'Duration (ms)' in df.columns:
    df['Duration (min)'] = df['Duration (ms)'] / (1000 * 60)

# Show summary statistics and check for missing values.
df.isnull().sum()
```



Highest Charting Position	0
Number of Times Charted	0
Song Name	0
Streams	0
Artist	0
Artist Followers	11
Genre	0
Release Date	28
Weeks Charted	0
Popularity	11
Danceability	11
Energy	11
Loudness	11
Speechiness	11
Acousticness	11
Liveness	11
Tempo	11
Duration (ms)	11
Valence	11
Release Year	28
Total Chart Presence	0
Duration (min)	11
dtype: int64	

```
df.describe(include='all')
```



	Highest Charting Position	Number of Times Charted	Song Name	Streams	Artist	Artist Followers	Genre	Release Date	Weeks Charted	Popularity	...	L
count	1556.000000	1556.000000	1556	1.556000e+03	1556	1.545000e+03	1556	1528	1556.000000	1545.000000	...	154
unique	NaN	NaN	1556	NaN	716	NaN	395	NaN	NaN	NaN	...	
top	NaN	NaN	Beggin'	NaN	Taylor Swift	NaN	[]	NaN	NaN	NaN	...	
freq	NaN	NaN	1	NaN	52	NaN	75	NaN	NaN	NaN	...	
mean	87.744216	10.668380	NaN	6.340219e+06	NaN	1.471690e+07	NaN	2019-03-08 10:36:07.539267072	10.668380	70.089320
min	1.000000	1.000000	NaN	4.176083e+06	NaN	4.883000e+03	NaN	1942-01-01 00:00:00	1.000000	0.000000	...	-2
25%	37.000000	1.000000	NaN	4.915322e+06	NaN	2.123734e+06	NaN	2020-01-17 00:00:00	1.000000	65.000000
50%	80.000000	4.000000	NaN	5.275748e+06	NaN	6.852509e+06	NaN	2020-06-19 00:00:00	4.000000	73.000000
75%	137.000000	12.000000	NaN	6.455044e+06	NaN	2.269875e+07	NaN	2021-01-14 06:00:00	12.000000	80.000000	...	
max	200.000000	142.000000	NaN	4.863345e+07	NaN	8.333778e+07	NaN	2021-08-13 00:00:00	142.000000	100.000000	...	
std	58.147225	16.360546	NaN	3.369479e+06	NaN	1.667579e+07	NaN	NaN	16.360546	15.824034	...	

11 rows × 22 columns

⌵ Handling Missing Values

```
before_na = df.isna().sum()
```

```

df.fillna(df.mean(numeric_only=True), inplace=True)
for col in df.select_dtypes(include=['object', 'category']).columns:
    if df[col].isna().any():
        df[col].fillna(df[col].mode()[0], inplace=True)

# Fill Release Date with median
if 'Release Date' in df.columns:
    date_non_na = df['Release Date'].dropna().sort_values().reset_index(drop=True)
    if len(date_non_na) > 0:
        median_idx = len(date_non_na) // 2
        median_date = date_non_na.iloc[median_idx]
        df['Release Date'].fillna(median_date, inplace=True)

after_na = df.isna().sum()
print('Missing values before:\n', before_na)
print('\nMissing values after:\n', after_na)

```

```

Missing values before:
Highest Charting Position      0
Number of Times Charted       0
Song Name                     0
Streams                       0
Artist                       0
Artist Followers              11
Genre                        0
Release Date                  28
Weeks Charted                 0
Popularity                   11
Danceability                 11
Energy                       11
Loudness                     11
Speechiness                  11
Acousticness                 11
Liveness                     11
Tempo                        11
Duration (ms)                 11
Valence                      11
Release Year                  28
Total Chart Presence          0
Duration (min)                11
dtype: int64

```

```

Missing values after:
Highest Charting Position      0
Number of Times Charted       0
Song Name                     0
Streams                       0
Artist                       0
Artist Followers              0
Genre                        0
Release Date                  0
Weeks Charted                 0
Popularity                   0
Danceability                 0
Energy                       0
Loudness                     0
Speechiness                  0
Acousticness                 0
Liveness                     0
Tempo                        0
Duration (ms)                 0
Valence                      0
Release Year                  0
Total Chart Presence          0
Duration (min)                0
dtype: int64

```

/tmp/ipykernel_6447/1203014306.py:14: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col]

```
df['Release Date'].fillna(median_date, inplace=True)
```

✓ Create main EDA DataFrame, including all audio features

```

base_cols = [
    'Song Name', 'Artist', 'Genre', 'Streams', 'Popularity',

```

```
'Release Year', 'Total Chart Presence', 'Duration (min)']
]
audio_features = [
    'Danceability', 'Energy', 'Loudness', 'Speechiness', 'Acousticness',
    'Liveness', 'Tempo', 'Valence'
]
all_cols = base_cols + [col for col in audio_features if col in df.columns]
df_clean = df[all_cols].copy()

# Optional: Mood Score
if all(col in df.columns for col in ['Valence', 'Acousticness']):
    df_clean['Mood Score'] = (df['Valence'] + (1 - df['Acousticness'])) / 2

df_clean.head()
```

↻

	Song Name	Artist	Genre	Streams	Popularity	Release Year	Total Chart Presence	Duration (min)	Danceability	Energy	Loudness	Speechiness
0	Beggin'	Måneskin	['indie rock italiano', 'italian pop']	48633449	100.0	2017.0	16	3.526000	0.714	0.800	-4.808	0.0504
1	STAY (with Justin Bieber)	The Kid LAROI	['australian hip hop']	47248719	99.0	2021.0	6	2.363433	0.591	0.764	-5.484	0.0483
2	good 4 u	Olivia Rodrigo	['pop']	40162559	99.0	2021.0	22	2.969117	0.563	0.664	-5.044	0.1540
3	Bad Habits	Ed Sheeran	['pop', 'uk pop']	37799456	98.0	2021.0	10	3.850683	0.808	0.897	-3.712	0.0348
4	INDUSTRY BABY	Lil Nas X	['lgbtq+ hip hop']	33948454	96.0	2021.0	2	3.533333	0.736	0.704	-7.409	0.0615

Replace zero values where zero is likely invalid

```
numeric_cols = df_clean.select_dtypes(include=['int64', 'float64']).columns
zero_counts = (df_clean[numeric_cols] == 0).sum()
print("Zero values per numeric column:\n", zero_counts)

# Replace zero with median for columns where zero is likely invalid
for col in numeric_cols:
    if col not in ['Streams', 'Popularity']:
        df_clean[col] = df_clean[col].replace(0, df_clean[col].median())
```

↻

Zero values per numeric column:	
Streams	0
Popularity	36
Release Year	0
Total Chart Presence	0
Duration (min)	0
Danceability	0
Energy	0
Loudness	0
Speechiness	0
Acousticness	0
Liveness	0
Tempo	0
Valence	0
Mood Score	0
dtype: int64	

Summary statistics

```
df_clean.info()
df_clean.shape
df_clean.isna().sum()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1556 entries, 0 to 1555
Data columns (total 17 columns):
#   Column                                Non-Null Count  Dtype
---  -
---
```

```

0 Song Name      1556 non-null object
1 Artist        1556 non-null object
2 Genre         1556 non-null object
3 Streams       1556 non-null int64
4 Popularity    1556 non-null float64
5 Release Year  1556 non-null float64
6 Total Chart Presence 1556 non-null int64
7 Duration (min) 1556 non-null float64
8 Danceability  1556 non-null float64
9 Energy        1556 non-null float64
10 Loudness     1556 non-null float64
11 Speechiness  1556 non-null float64
12 Acousticness 1556 non-null float64
13 Liveness     1556 non-null float64
14 Tempo        1556 non-null float64
15 Valence      1556 non-null float64
16 Mood Score   1556 non-null float64

```

```
dtypes: float64(12), int64(2), object(3)
```

```
memory usage: 206.8+ KB
```

```

Song Name      0
Artist         0
Genre          0
Streams        0
Popularity     0
Release Year   0
Total Chart Presence 0
Duration (min) 0
Danceability   0
Energy         0
Loudness       0
Speechiness    0
Acousticness   0
Liveness       0
Tempo          0
Valence        0
Mood Score     0
dtype: int64

```

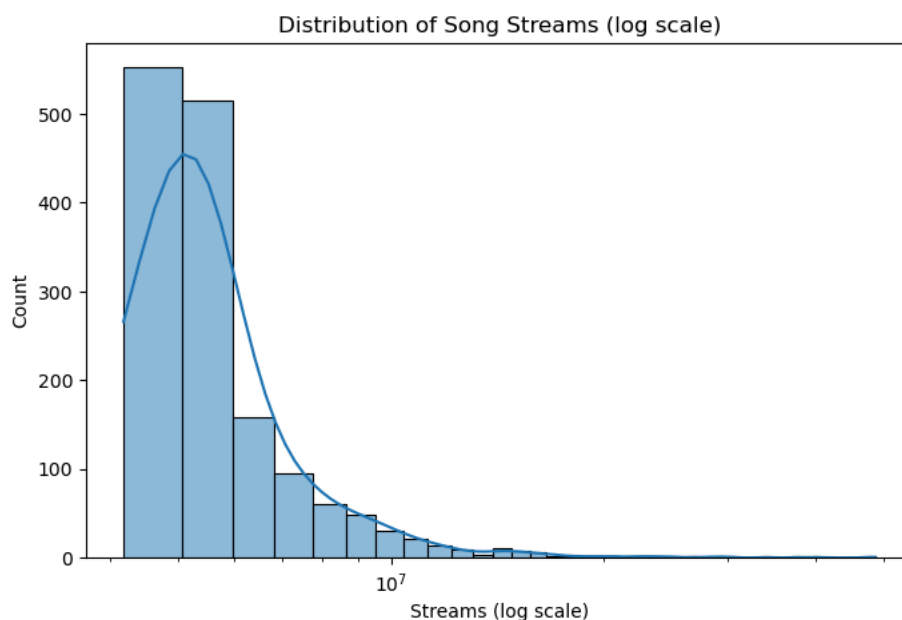
Visualizations

Use df_clean for all plots

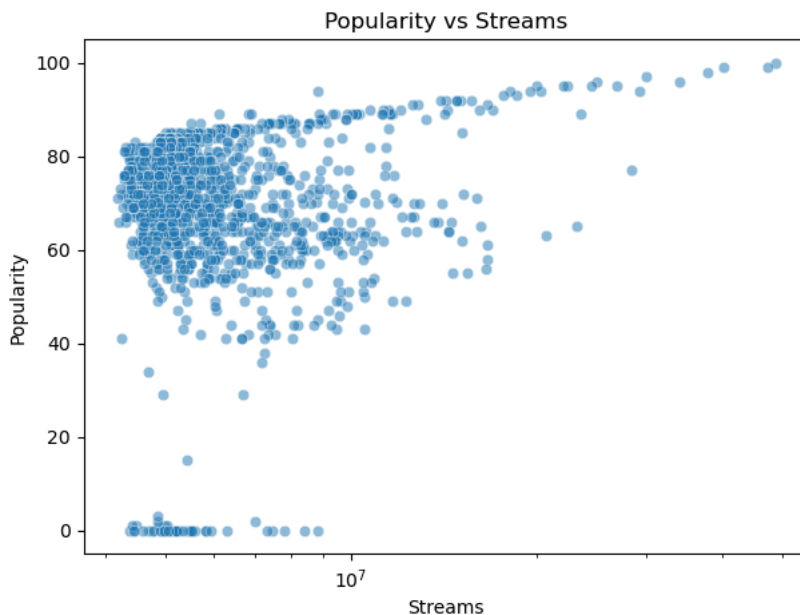
```

# 1. Distribution of Streams
plt.figure(figsize=(8,5))
sns.histplot(df_clean['Streams'], bins=50, kde=True)
plt.xscale("log")
plt.title("Distribution of Song Streams (log scale)")
plt.xlabel("Streams (log scale)")
plt.ylabel("Count")
plt.show()

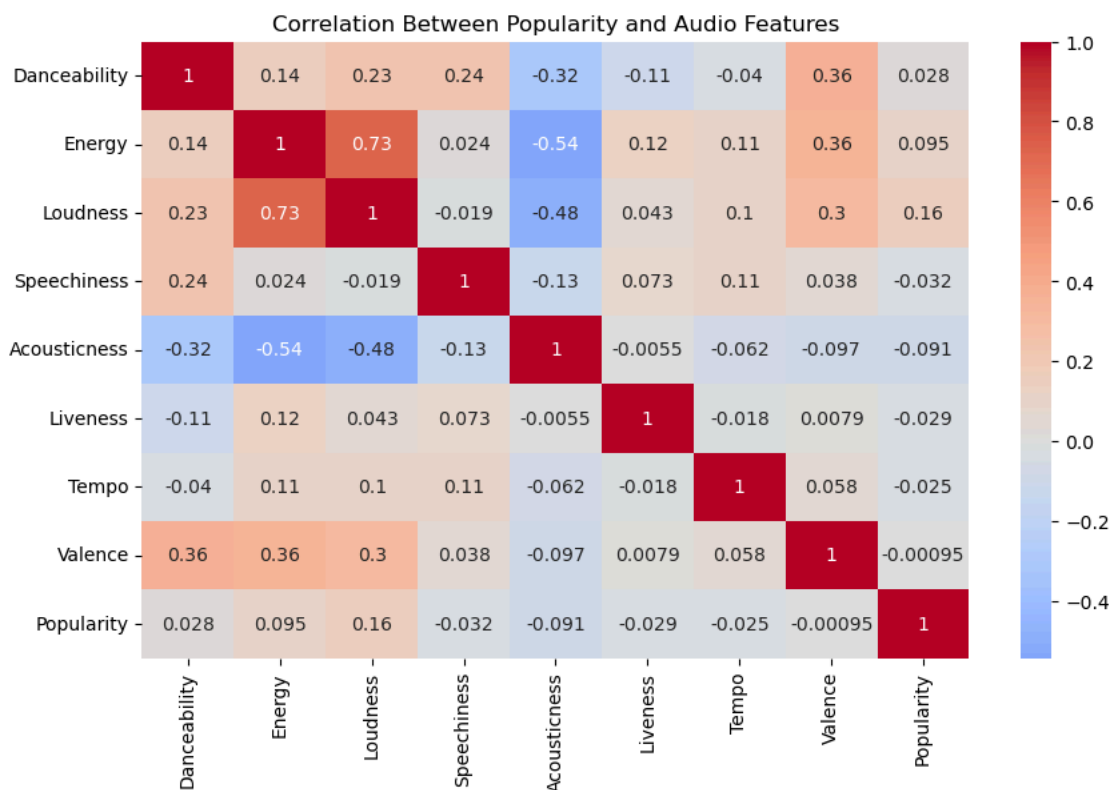
```



```
# 2. Popularity vs Streams
plt.figure(figsize=(7,5))
sns.scatterplot(data=df_clean, x='Streams', y='Popularity', alpha=0.5)
plt.xscale("log")
plt.title("Popularity vs Streams")
plt.show()
```



```
# 3. Correlation Heatmap of Audio Features (now using df_clean)
plt.figure(figsize=(10,6))
audio_for_corr = [col for col in audio_features + ['Popularity'] if col in df_clean.columns]
corr = df_clean[audio_for_corr].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', center=0)
plt.title("Correlation Between Popularity and Audio Features")
plt.show()
```



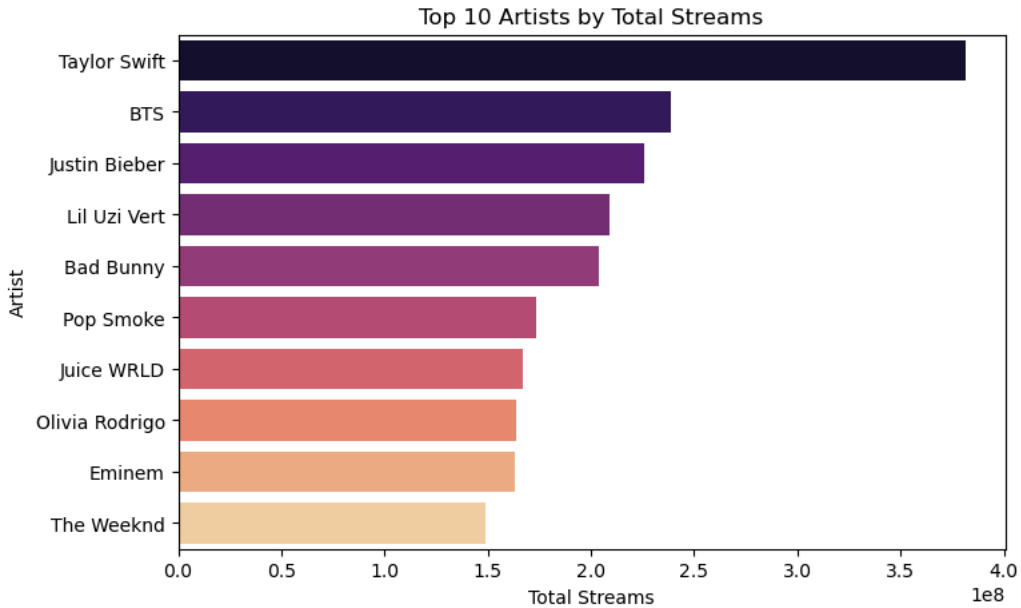
```
# 4. Top 10 Artists by Total Streams
if 'Artist' in df_clean.columns and 'Streams' in df_clean.columns:
    top_artists = df_clean.groupby('Artist')['Streams'].sum().nlargest(10)
```

```
top_artists = df_clean.groupby('Artist')['Streams'].sum().nlargest(10)
plt.figure(figsize=(8,5))
sns.barplot(x=top_artists.values, y=top_artists.index, palette="magma")
plt.title("Top 10 Artists by Total Streams")
plt.xlabel("Total Streams")
plt.show()
```

⚠ /tmp/ipykernel_6447/2627981847.py:5: FutureWarning:

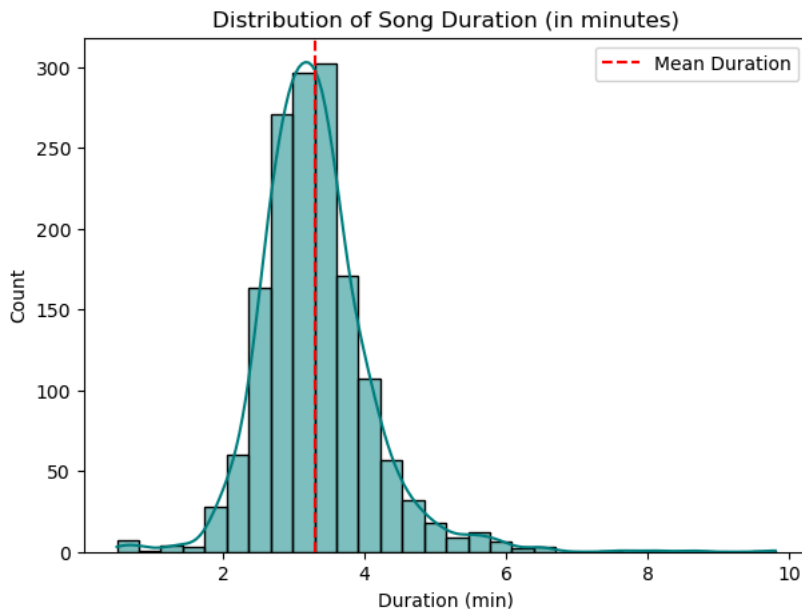
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

```
sns.barplot(x=top_artists.values, y=top_artists.index, palette="magma")
```



5. Average Duration of Songs

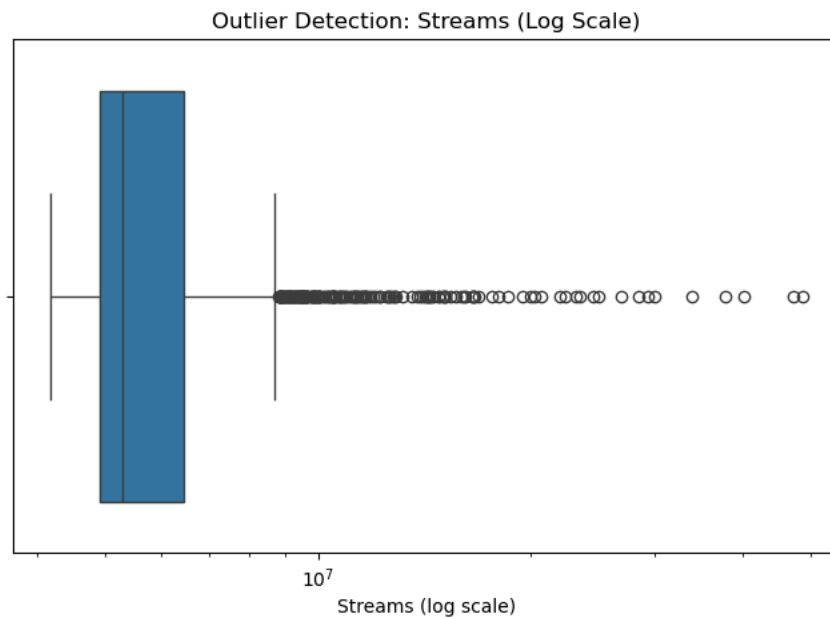
```
plt.figure(figsize=(7,5))
sns.histplot(df_clean['Duration (min)'], bins=30, kde=True, color="teal")
plt.axvline(df_clean['Duration (min)'].mean(), color='red', linestyle='--', label="Mean Duration")
plt.title("Distribution of Song Duration (in minutes)")
plt.xlabel("Duration (min)")
plt.legend()
plt.show()
```



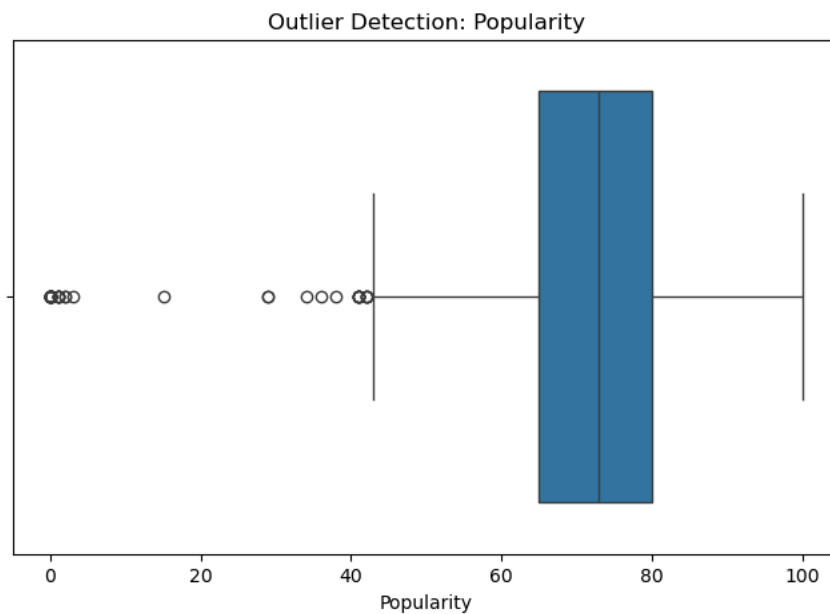
```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df_clean['Streams'])
plt.xscale("log")
plt.title("Outlier Detection: Streams (Log Scale)")
```



```
plt.xlabel("Streams (log scale)")
plt.show()
```



```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df_clean['Popularity'])
plt.title("Outlier Detection: Popularity")
plt.xlabel("Popularity")
plt.show()
```

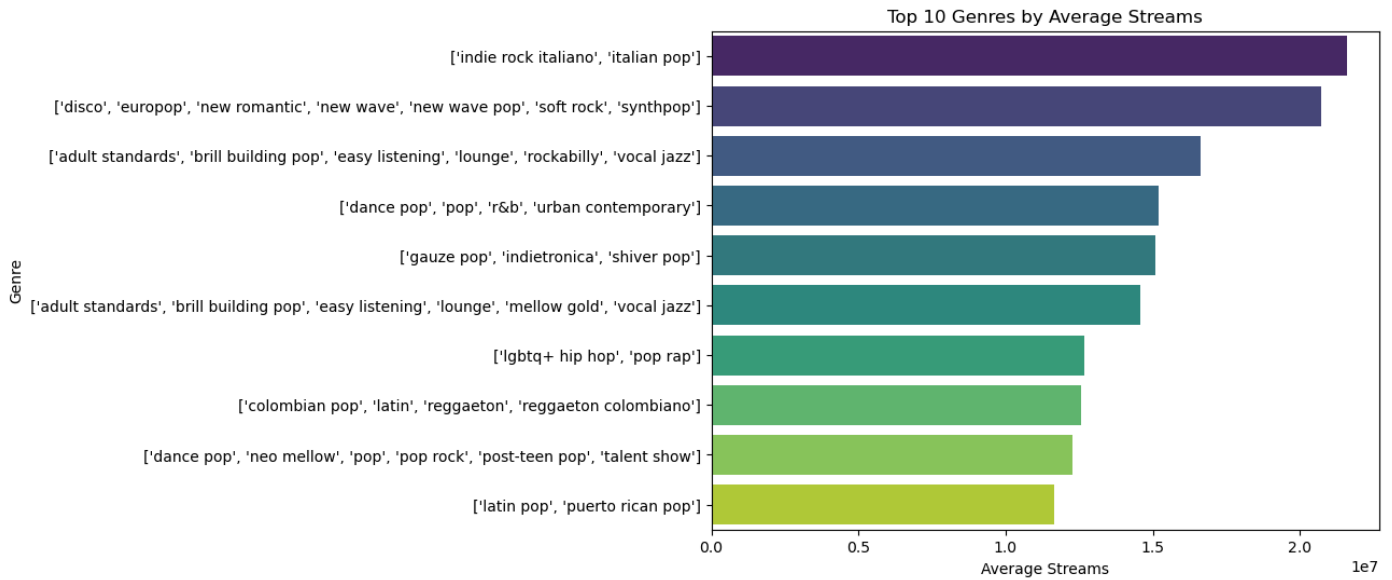


```
genre_stats = df_clean.groupby('Genre').agg({
    'Streams': 'mean',
    'Popularity': 'mean'
}).sort_values(by='Streams', ascending=False).head(10)
```

```
plt.figure(figsize=(8, 6))
sns.barplot(x=genre_stats['Streams'], y=genre_stats.index, palette='viridis')
plt.title("Top 10 Genres by Average Streams")
plt.xlabel("Average Streams")
plt.ylabel("Genre")
plt.show()
```

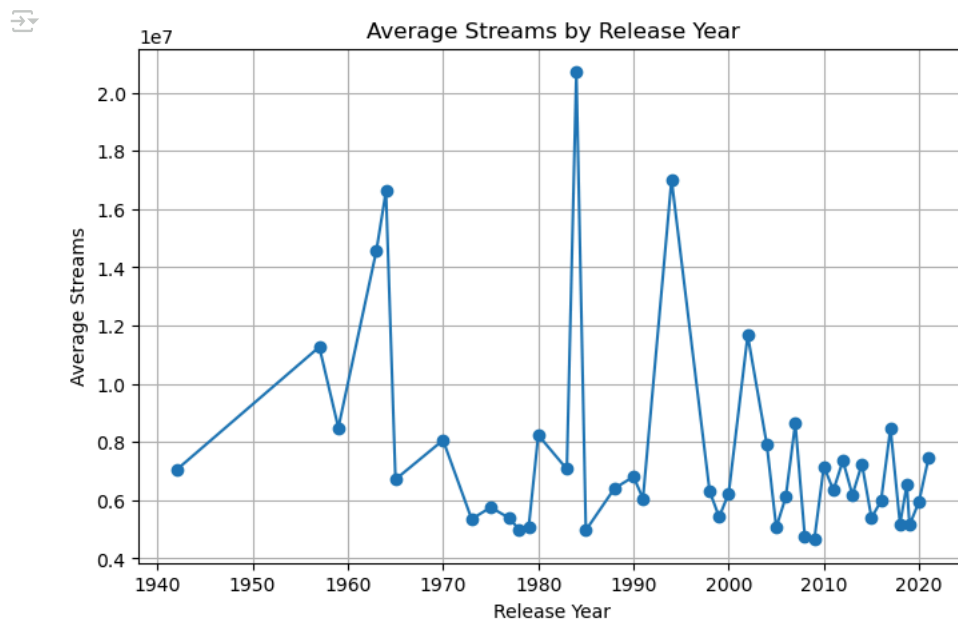
 /tmp/ipykernel_6447/2141890181.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and
 sns.barplot(x=genre_stats['Streams'], y=genre_stats.index, palette='viridis')



```
yearly_trends = df_clean.groupby('Release Year')['Streams'].mean()
```

```
plt.figure(figsize=(8, 5))
yearly_trends.plot(kind='line', marker='o')
plt.title("Average Streams by Release Year")
plt.xlabel("Release Year")
plt.ylabel("Average Streams")
plt.grid()
plt.show()
```



✓ Spotify Dataset – Final EDA Insights

1. Song Duration Distribution

- Most songs are between **2 to 5 minutes**, with a mean of ~3 minutes.
- Slight right skew due to a few longer tracks (7–8+ minutes).
- Confirms commercial music standard of ~3–4 minutes.

2. Top Artists by Total Streams

- **Taylor Swift** dominates, followed by **BTS** and **Justin Bieber**.
- Other high performers: **Lil Uzi Vert**, **Bad Bunny**, **Pop Smoke**, **Juice WRLD**, **Olivia Rodrigo**, **Eminem**, **The Weeknd**.
- Shows dominance of global pop and rap superstars.

3. Song Streams Distribution (Log Scale)

- Distribution is **heavily right-skewed**; most tracks have modest streams.
- Only a small fraction achieve massive popularity (millions to hundreds of millions).
- Indicates a **long-tail effect** – few songs drive most of the streaming volume.

4. Outlier Analysis

- **Streams**: Outliers are mega-hits significantly above median.
- **Popularity**: Most songs lie in the 60–90 range, but some low-popularity tracks still chart due to niche appeal or recent release.