

Importing Libraries

In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Introduction

- Task 1.1: Overview of the project and its objectives.
- Task 1.2: Explanation of the dataset used (Spotify tracks, features, user data, etc.).

Task 1.1 — Overview of the Project and Its Objectives

The Spotify Data Analytics Capstone Project aims to explore and understand global music trends by analyzing multiple Spotify datasets. The purpose of this project is to uncover how different audio features, genres, artists, and release years influence a song's popularity and user listening behavior. By performing detailed exploratory data analysis (EDA), visualizations, and predictive modeling, the project seeks to transform raw data into meaningful insights that reflect evolving music preferences.

This project not only focuses on analyzing basic song characteristics—such as danceability, energy, tempo, acousticness, and loudness—but also studies how these characteristics vary across genres, artists, and years. Additionally, it aims to identify correlations between various audio features and evaluate the factors contributing to song popularity. Through trend analysis, feature exploration, and machine learning modeling, the project provides a comprehensive understanding of the musical landscape on Spotify.

Overall, the main objectives are:

- To analyze patterns and trends in Spotify's music catalog.
- To understand the relationship between audio features and song popularity.
- To study genre-wise and artist-wise differences in musical characteristics.
- To examine how music features and popularity have evolved over time.
- To build predictive models that estimate the popularity of songs based on their features.
- To present insights through meaningful visualizations and conclude with recommendations.

Task 1.2 — Explanation of the Dataset Used

The project uses multiple Spotify datasets, each offering a different perspective on songs, artists, genres, and yearly trends. These datasets contain detailed audio features extracted directly from Spotify's API, providing a rich foundation for music analysis and modeling. Below is a description of each dataset used in the project:

1. data.csv — Main Spotify Tracks Dataset

This is the primary dataset containing individual Spotify tracks along with their audio characteristics. It includes:

- Track information: track name, artist name, track ID
- Popularity score
- Audio features:

- danceability
- energy
- loudness
- acousticness
- instrumentalness
- liveness
- valence
- tempo
- duration

- Genre tags and additional metadata

This dataset forms the core of the EDA, giving insights into how different musical features affect popularity.

2. data_by_artist.csv — Artist-Level Dataset

This dataset summarizes song features at the artist level. It includes:

- Artist name
- Number of tracks released
- Average audio features for each artist
- Overall popularity level

This dataset helps analyze:

- Top-performing artists
- Artists' preferred musical styles
- Feature variations across artists

3. data_by_genres.csv — Genre-Level Dataset

This dataset aggregates audio features by genre. It includes:

- Genre name
- Popularity
- Average musical characteristics (danceability, energy, tempo, etc.)

This dataset is ideal for comparing:

- Which genres are most popular
- How genres differ in musical qualities
- How user preferences shift across genres

4. data_by_year.csv — Yearly Trends Dataset

This dataset provides year-wise aggregated statistics of songs. It includes:

- Release year
- Average popularity
- Average tempo, loudness, danceability, energy, etc.

This helps in:

- Time-series trend analysis
- Studying how music characteristics have evolved
- Understanding shifts in global music taste

5. data_w_genres.csv — Tracks with Expanded Genre Information

This dataset associates each track with multiple genre labels and corresponds closely with data.csv. It includes:

- Track-level audio features
- One or more genre classifications

This dataset is helpful for:

- Multi-genre analysis
- Better modeling of popularity
- Comparing single-genre vs. multi-genre songs

Together, these datasets provide a complete music analytics ecosystem, covering:

- Song-level insights
- Artist-level trends
- Genre-level analysis
- Yearly evolution
- Audio feature relationships

These datasets allow the project to perform deep EDA, build predictive models, and generate meaningful insights into Spotify's music landscape.

Data Collection and Preprocessing

- Task 2.1: Load the dataset into the Jupyter notebook.
- Task 2.2: Inspect and clean the data (handle missing values, duplicates, etc.).
- Task 2.3: Perform exploratory data analysis (EDA) to understand the basic characteristics of the data.

Task 2.1: Load the dataset into the Jupyter notebook.

Main Data

In [4]:

```
data = pd.read_csv(r"C:\Users\aryan\OneDrive\Desktop\DS PROJECTS\BIG PROJECT\SPOTIFY ANALYSIS (4)\data.csv")  
data
```

Out[4]:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id
0	0.0594	1921	0.98200	['Sergei Rachmaninoff', 'James Levine', 'Berlin State Opera Orchestra']	0.279	831667	0.211	0	4BJqT0PrAfxzM0xytFOI
1	0.9630	1921	0.73200	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt
2	0.0394	1921	0.96100	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...']	0.328	500062	0.166	0	1o6I8BglA6ylDMrlELygv
3	0.1650	1921	0.96700	['Frank Parker']	0.275	210000	0.309	0	3ffBPsC5vPBKxYSee08FDf
4	0.2530	1921	0.95700	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKnw9vt
...
170648	0.6080	2020	0.08460	['Anuel AA', 'Daddy Yankee', 'KAROL G', 'Ozuna...']	0.786	301714	0.808	0	0KklkfsLEJbrclhYsCL7L
170649	0.7340	2020	0.20600	['Ashnikko']	0.717	150654	0.753	0	0OStKKAuXlxA0fMH54Qs6I
170650	0.6370	2020	0.10100	['MAMAMOO']	0.634	211280	0.858	0	4BZXVFYCb76Q0Klojq4piV
170651	0.1950	2020	0.00998	['Eminem']	0.671	337147	0.623	1	5SiZJoLXp3WOBJ4C8IK0c
170652	0.6420	2020	0.13200	['KEVVO', 'J Balvin']	0.856	189507	0.721	1	7HmnJHfs0BkFzX4x8j0hk

170653 rows × 19 columns

In [5]:

```
#Display the first few rows of main dataset
print("Main Dataset:")
display(data.head())
```

Main Dataset:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	inst
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...']	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...']	0.328	500062	0.166	0	1o6l8BglA6ylDMrIELygv1	
3	0.1650	1921	0.967	['Frank Parker']	0.275	210000	0.309	0	3ftBPsC5vPBKxYSee08FDH	
4	0.2530	1921	0.957	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKmw9v6	

Data by Artist

In [4]:

```
data_by_artist = pd.read_csv(r"C:\Users\aryan\OneDrive\Desktop\DS PROJECTS\BIG PROJECT\SPOTIFY ANALYSIS (4)\data_by_artist.csv")  
data_by_artist
```

Out[4]:

	mode	count	acousticness	artists	danceability	duration_ms	energy	instrumentalness	liveness	loudness
0	1	9	0.590111	"Cats" 1981 Original London Cast	0.467222	250318.555556	0.394003	0.011400	0.290833	14.44800
1	1	26	0.862538	"Cats" 1983 Broadway Cast	0.441731	287280.000000	0.406808	0.081158	0.315215	10.69000
2	1	7	0.856571	"Fiddler On The Roof" Motion Picture Chorus	0.348286	328920.000000	0.286571	0.024593	0.325786	15.23071
3	1	27	0.884926	"Fiddler On The Roof" Motion Picture Orchestra	0.425074	262890.962963	0.245770	0.073587	0.275481	15.63937
4	1	7	0.510714	"Joseph And The Amazing Technicolor Dreamcoat"...	0.467143	270436.142857	0.488286	0.009400	0.195000	10.23671
...
28675	1	2	0.512000	麥志誠	0.356000	198773.000000	0.306000	0.008970	0.108000	10.11900
28676	0	2	0.541000	黃品源	0.578000	293840.000000	0.334000	0.000006	0.067500	11.97400
28677	1	11	0.785455	黃國隆	0.570818	174582.727273	0.148400	0.000083	0.142191	21.61009
28678	1	2	0.381000	黑豹	0.353000	316160.000000	0.686000	0.000000	0.056800	-9.10300
28679	1	2	0.568000	조정현	0.447000	237688.000000	0.215000	0.000001	0.064900	16.47800

28680 rows × 15 columns

In [5]:

```
#Display the first few rows data by artist
print("Artist Dataset:")
display(data_by_artist.head())
```

Artist Dataset:

	mode	count	acousticness	artists	danceability	duration_ms	energy	instrumentalness	liveness	loudness	s
0	1	9	0.590111	"Cats" 1981 Original London Cast	0.467222	250318.555556	0.394003	0.011400	0.290833	14.448000	-
1	1	26	0.862538	"Cats" 1983 Broadway Cast	0.441731	287280.000000	0.406808	0.081158	0.315215	10.690000	-
2	1	7	0.856571	"Fiddler On The Roof" Motion Picture Chorus	0.348286	328920.000000	0.286571	0.024593	0.325786	15.230714	-
3	1	27	0.884926	"Fiddler On The Roof" Motion Picture Orchestra	0.425074	262890.962963	0.245770	0.073587	0.275481	15.639370	-
4	1	7	0.510714	"Joseph And The Amazing Technicolor Dreamcoat"...	0.467143	270436.142857	0.488286	0.009400	0.195000	10.236714	-

Data by Genres

In [6]:

```
data_by_genres = pd.read_csv(r'C:\Users\aryan\OneDrive\Desktop\DS PROJECTS\BIG PROJECT\SPOTIFY ANALYSIS (4)\data_by_genres.csv')
```

Out[6]:

	mode	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness
0	1	21st century classical	0.979333	0.162883	1.602977e+05	0.071317	0.606834	0.361600	31.514333	0.04056
1	1	432hz	0.494780	0.299333	1.048887e+06	0.450678	0.477762	0.131000	16.854000	0.07681
2	1	8-bit	0.762000	0.712000	1.151770e+05	0.818000	0.876000	0.126000	-9.180000	0.04700
3	1	[]	0.651417	0.529093	2.328809e+05	0.419146	0.205309	0.218696	12.288965	0.10787
4	1	a cappella	0.676557	0.538961	1.906285e+05	0.316434	0.003003	0.172254	12.479387	0.08285
...
2968	1	zolo	0.222625	0.547082	2.580991e+05	0.610240	0.143872	0.204206	11.295878	0.06108
2969	0	zouglou	0.161000	0.863000	2.063200e+05	0.909000	0.000000	0.108000	-5.985000	0.08130
2970	1	zouk	0.263261	0.748889	3.060728e+05	0.622444	0.257227	0.089678	10.289222	0.03877
2971	0	zurich indie	0.993000	0.705667	1.984173e+05	0.172667	0.468633	0.179667	11.453333	0.34866
2972	1	zydeco	0.421038	0.629409	1.716717e+05	0.609369	0.019248	0.255877	-9.854825	0.05049

2973 rows × 14 columns

In [7]:

```
# Display the first few rows of data by genres dataset
print("Genres Dataset:")
display(data_by_genres.head())
```

Genres Dataset:

	mode	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness
0	1	21st century classical	0.979333	0.162883	1.602977e+05	0.071317	0.606834	0.361600	31.514333	0.040567
1	1	432hz	0.494780	0.299333	1.048887e+06	0.450678	0.477762	0.131000	16.854000	0.076817
2	1	8-bit	0.762000	0.712000	1.151770e+05	0.818000	0.876000	0.126000	-9.180000	0.047000
3	1	[]	0.651417	0.529093	2.328809e+05	0.419146	0.205309	0.218696	12.288965	0.107872
4	1	a cappella	0.676557	0.538961	1.906285e+05	0.316434	0.003003	0.172254	12.479387	0.082851

Data by year

```
In [8]:
data_by_year = pd.read_csv(r"C:\Users\aryan\OneDrive\Desktop\DS PROJECTS\BIG PROJECT\SPOTIFY ANALYSIS (4)\data_by_year.csv")
data_by_year
```

Out[8]:

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness
0	1	1921	0.886896	0.418597	260537.166667	0.231815	0.344878	0.205710	17.048667	0.073662
1	1	1922	0.938592	0.482042	165469.746479	0.237815	0.434195	0.240720	19.275282	0.116655
2	1	1923	0.957247	0.577341	177942.362162	0.262406	0.371733	0.227462	14.129211	0.093949
3	1	1924	0.940200	0.549894	191046.707627	0.344347	0.581701	0.235219	14.231343	0.092089
4	1	1925	0.962607	0.573863	184986.924460	0.278594	0.418297	0.237668	14.146414	0.111918
..
95	1	2016	0.284171	0.600202	221396.510295	0.592855	0.093984	0.181170	-8.061056	0.104313
96	1	2017	0.286099	0.612217	211115.696787	0.590421	0.097091	0.191713	-8.312630	0.110536
97	1	2018	0.267633	0.663500	206001.007133	0.602435	0.054217	0.176326	-7.168785	0.127176
98	1	2019	0.278299	0.644814	201024.788096	0.593224	0.077640	0.172616	-7.722192	0.121043
99	1	2020	0.219931	0.692904	193728.397537	0.631232	0.016376	0.178535	-6.595067	0.141384

100 rows × 14 columns

In [9]:

```
# Display the first few rows of each dataset
print("Yearly Dataset:")
display(data_by_year.head())
```

Yearly Dataset:

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness
0	1	1921	0.886896	0.418597	260537.166667	0.231815	0.344878	0.205710	-17.048667	0.073662 10
1	1	1922	0.938592	0.482042	165469.746479	0.237815	0.434195	0.240720	-19.275282	0.116655 10
2	1	1923	0.957247	0.577341	177942.362162	0.262406	0.371733	0.227462	-14.129211	0.093949 11
3	1	1924	0.940200	0.549894	191046.707627	0.344347	0.581701	0.235219	-14.231343	0.092089 12
4	1	1925	0.962607	0.573863	184986.924460	0.278594	0.418297	0.237668	-14.146414	0.111918 11

In[10]:

```
data_w_genres = pd.read_csv(r"C:\Users\aryan\OneDrive\Desktop\DS PROJECTS\BIG PROJECT\SPOTIFY ANALYSIS (4)\data_w_genres.csv")
data_w_genres
```

Out[10]:

	genres	artists	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness
0	['show tunes']	"Cats" 1981 Original London Cast	0.590111	0.467222	250318.555556	0.394003	0.011400	0.290833	-14.448000
1	[]	"Cats" 1983 Broadway Cast	0.862538	0.441731	287280.000000	0.406808	0.081158	0.315215	-10.690000
2	[]	'Fiddler On The Roof' Motion Picture Chorus	0.856571	0.348286	328920.000000	0.286571	0.024593	0.325786	-15.230714
3	[]	'Fiddler On The Roof' Motion Picture Orchestra	0.884926	0.425074	262890.962963	0.245770	0.073587	0.275481	-15.639370
4	[]	"Joseph And The Amazing Technicolor Dreamcoat"...	0.510714	0.467143	270436.142857	0.488286	0.009400	0.195000	-10.236714
...
28675	[]	麥志誠	0.512000	0.356000	198773.000000	0.306000	0.008970	0.108000	-10.119000
28676	['c-pop', 'classic cantopop', 'classic mandopop...']	黃品源	0.541000	0.578000	293840.000000	0.334000	0.000006	0.067500	-11.974000
28677	[]	黃國隆	0.785455	0.570818	174582.727273	0.148400	0.000083	0.142191	-21.610091
28678	['chinese indie', 'chinese indie rock']	黑豹	0.381000	0.353000	316160.000000	0.686000	0.000000	0.056800	-9.103000
28679	['classic korean pop']	조정현	0.568000	0.447000	237688.000000	0.215000	0.000001	0.064900	-16.478000

28680 rows × 16 columns

In[11]:

#Display the first few rows of each dataset

```
print("Dataset with Genres:")
display(data_w_genres.head())
```

Dataset with Genres:

	genres	artists	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speech
0	['show tunes']	"Cats" 1981 Original London Cast	0.590111	0.467222	250318.555556	0.394003	0.011400	0.290833	14.448000	0.21
1	[]	"Cats" 1983 Broadway Cast	0.862538	0.441731	287280.000000	0.406808	0.081158	0.315215	10.690000	0.17
2	[]	"Fiddler On The Roof"	0.856571	0.348286	328920.000000	0.286571	0.024593	0.325786	15.230714	0.11
3	[]	"Fiddler On The Roof"	0.884926	0.425074	262890.962963	0.245770	0.073587	0.275481	15.639370	0.12
4	[]	"Joseph And The Amazing Technicolor Dreamcoat"...	0.510714	0.467143	270436.142857	0.488286	0.009400	0.195000	10.236714	0.09

Task 2.2: Inspect and clean the data (handle missing values, duplicates, etc.).

Inspecting the Datasets

Checking Basic information of all datasets

In [12]:

```
# Check basic info of all datasets
print("Main dataset info:")
print(data.info())
```

```
print("Artist dataset info:")
print(data_by_artist.info())
```

```
print("Genres dataset info:")
print(data_by_genres.info())
```

```
print("Yearly dataset info:")
print(data_by_year.info())
```

```
print("Dataset with genres info:")
print(data_w_genres.info())
```

Main dataset info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   valence     170653 non-null float64
 1   year        170653 non-null int64  
 2   acousticness 170653 non-null float64
 3   artists      170653 non-null object 
 4   danceability 170653 non-null float64
 5   duration_ms  170653 non-null int64  
 6   energy       170653 non-null float64
 7   explicit     170653 non-null int64  
 8   id           170653 non-null object 
 9   instrumentalness 170653 non-null float64
 10  key          170653 non-null int64  
 11  liveness     170653 non-null float64
 12  loudness    170653 non-null float64
 13  mode         170653 non-null int64  
 14  name         170653 non-null object 
 15  popularity   170653 non-null int64
```

```
16 release_date    170653 non-null object
17 speechiness     170653 non-null float64
18 tempo          170653 non-null float64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.7+ MB
```

None

Artist dataset info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28680 entries, 0 to 28679
Data columns (total 15 columns):
 # Column      Non-Null Count Dtype 
--- 
0 mode         28680 non-null int64 
1 count        28680 non-null int64 
2 acousticness 28680 non-null float64
3 artists      28680 non-null object 
4 danceability 28680 non-null float64
5 duration_ms  28680 non-null float64
6 energy        28680 non-null float64
7 instrumentness 28680 non-null float64
8 liveness      28680 non-null float64
9 loudness      28680 non-null float64
10 speechiness   28680 non-null float64
11 tempo        28680 non-null float64
12 valence      28680 non-null float64
13 popularity    28680 non-null float64
14 key          28680 non-null int64 
dtypes: float64(11), int64(3), object(1)
memory usage: 3.3+ MB
```

None

Genres dataset info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973 entries, 0 to 2972
Data columns (total 14 columns):
 # Column      Non-Null Count Dtype 
--- 
0 mode         2973 non-null int64 
1 genres        2973 non-null object 
2 acousticness  2973 non-null float64
3 danceability   2973 non-null float64
4 duration_ms   2973 non-null float64
5 energy        2973 non-null float64
6 instrumentness 2973 non-null float64
7 liveness      2973 non-null float64
8 loudness      2973 non-null float64
9 speechiness    2973 non-null float64
10 tempo        2973 non-null float64
11 valence      2973 non-null float64
12 popularity    2973 non-null float64
13 key          2973 non-null int64 
dtypes: float64(11), int64(2), object(1)
memory usage: 325.3+ KB
```

None

Yearly dataset info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 # Column      Non-Null Count Dtype 
--- 
0 mode         100 non-null int64 
1 year         100 non-null int64 
2 acousticness 100 non-null float64
3 danceability 100 non-null float64
4 duration_ms  100 non-null float64
5 energy        100 non-null float64
6 instrumentness 100 non-null float64
7 liveness      100 non-null float64
8 loudness      100 non-null float64
9 speechiness   100 non-null float64
10 tempo        100 non-null float64
11 valence      100 non-null float64
12 popularity    100 non-null float64
13 key          100 non-null int64 
dtypes: float64(11), int64(3)
memory usage: 11.1 KB
```

None

Dataset with genres info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28680 entries, 0 to 28679
Data columns (total 16 columns):
 # Column      Non-Null Count Dtype 
--- 
0 genres        28680 non-null object 
1 artists       28680 non-null object 
2 acousticness  28680 non-null float64
```

```
3 danceability    28680 non-null float64
4 duration_ms    28680 non-null float64
5 energy         28680 non-null float64
6 instrumentalness 28680 non-null float64
7 liveness        28680 non-null float64
8 loudness        28680 non-null float64
9 speechiness     28680 non-null float64
10 tempo          28680 non-null float64
11 valence        28680 non-null float64
12 popularity      28680 non-null float64
13 key            28680 non-null int64
14 mode           28680 non-null int64
15 count          28680 non-null int64
dtypes: float64(11), int64(3), object(2)
memory usage: 3.5+ MB
```

None

In [13]:

Check missing values

```
print("Missing values in main dataset:")
print(data.isnull().sum())
```

```
print("Missing values in artist dataset:")
print(data_by_artist.isnull().sum())
```

```
print("Missing values in genres dataset:")
print(data_by_genres.isnull().sum())
```

```
print("Missing values in yearly dataset:")
print(data_by_year.isnull().sum())
```

```
print("Missing values in dataset with genres:")
print(data_w_genres.isnull().sum())
```

Missing values in main dataset:

```
valence      0
year         0
acousticness 0
artists      0
danceability 0
duration_ms   0
energy       0
explicit     0
id           0
instrumentalness 0
key          0
liveness     0
loudness     0
mode         0
name         0
popularity   0
release_date 0
speechiness  0
tempo        0
dtype: int64
```

Missing values in artist dataset:

```
mode         0
count       0
acousticness 0
artists      0
danceability 0
duration_ms   0
energy       0
instrumentalness 0
liveness     0
loudness     0
speechiness  0
tempo        0
valence     0
popularity   0
key          0
dtype: int64
```

Missing values in genres dataset:

```
mode         0
genres      0
acousticness 0
danceability 0
duration_ms   0
energy       0
instrumentalness 0
liveness     0
loudness     0
speechiness  0
```

```
tempo      0
valence    0
popularity 0
key        0
dtype: int64
Missing values in yearly dataset:
mode      0
year      0
acousticness 0
danceability 0
duration_ms 0
energy     0
instrumentalness 0
liveness   0
loudness   0
speechiness 0
tempo      0
valence    0
popularity 0
key        0
dtype: int64
Missing values in dataset with genres:
genres    0
artists   0
acousticness 0
danceability 0
duration_ms 0
energy     0
instrumentalness 0
liveness   0
loudness   0
speechiness 0
tempo      0
valence    0
popularity 0
key        0
mode      0
count     0
dtype: int64
```

As we seen there is no missing value in all the datasets

Removing Duplicates

Duplicate tracks or repeated rows can affect analysis and model accuracy.

In [14]:

```
data.drop_duplicates(inplace=True)
data_by_artist.drop_duplicates(inplace=True)
data_by_genres.drop_duplicates(inplace=True)
data_by_year.drop_duplicates(inplace=True)
data_w_genres.drop_duplicates(inplace=True)
```

Task 2.3: Perform exploratory data analysis (EDA) to understand the basic characteristics of the data.

Step 1 — Summary Statistics

In [15]:

```
data.describe()
```

Out[15]:

	valence	year	acousticness	danceability	duration_ms	energy	explicit	instrumentalness
count	170653.000000	170653.000000	170653.000000	170653.000000	1.706530e+05	170653.000000	170653.000000	170653.000000
mean	0.528587	1976.787241	0.502115	0.537396	2.309483e+05	0.482389	0.084575	
std	0.263171	25.917853	0.376032	0.176138	1.261184e+05	0.267646	0.278249	
min	0.000000	1921.000000	0.000000	0.000000	5.108000e+03	0.000000	0.000000	
25%	0.317000	1956.000000	0.102000	0.415000	1.698270e+05	0.255000	0.000000	
50%	0.540000	1977.000000	0.516000	0.548000	2.074670e+05	0.471000	0.000000	
75%	0.747000	1999.000000	0.893000	0.668000	2.624000e+05	0.703000	0.000000	
max	1.000000	2020.000000	0.996000	0.988000	5.403500e+06	1.000000	1.000000	

Step 2 — Checking Data Types

In [16]:

data.dtypes

Out[16]:

```
valence    float64
year       int64
acousticness   float64
artists     object
danceability  float64
duration_ms  int64
energy      float64
explicit    int64
id          object
instrumentalness float64
key         int64
liveness    float64
loudness    float64
mode        int64
name        object
popularity   int64
release_date object
speechiness  float64
tempo       float64
dtype: object
```

Step 3 — Convert release_date to datetime

In [17]:

```
data['release_date'] = pd.to_datetime(data['release_date'], errors='coerce')
```

Step 4 — Distribution of Key Features

We analyze:

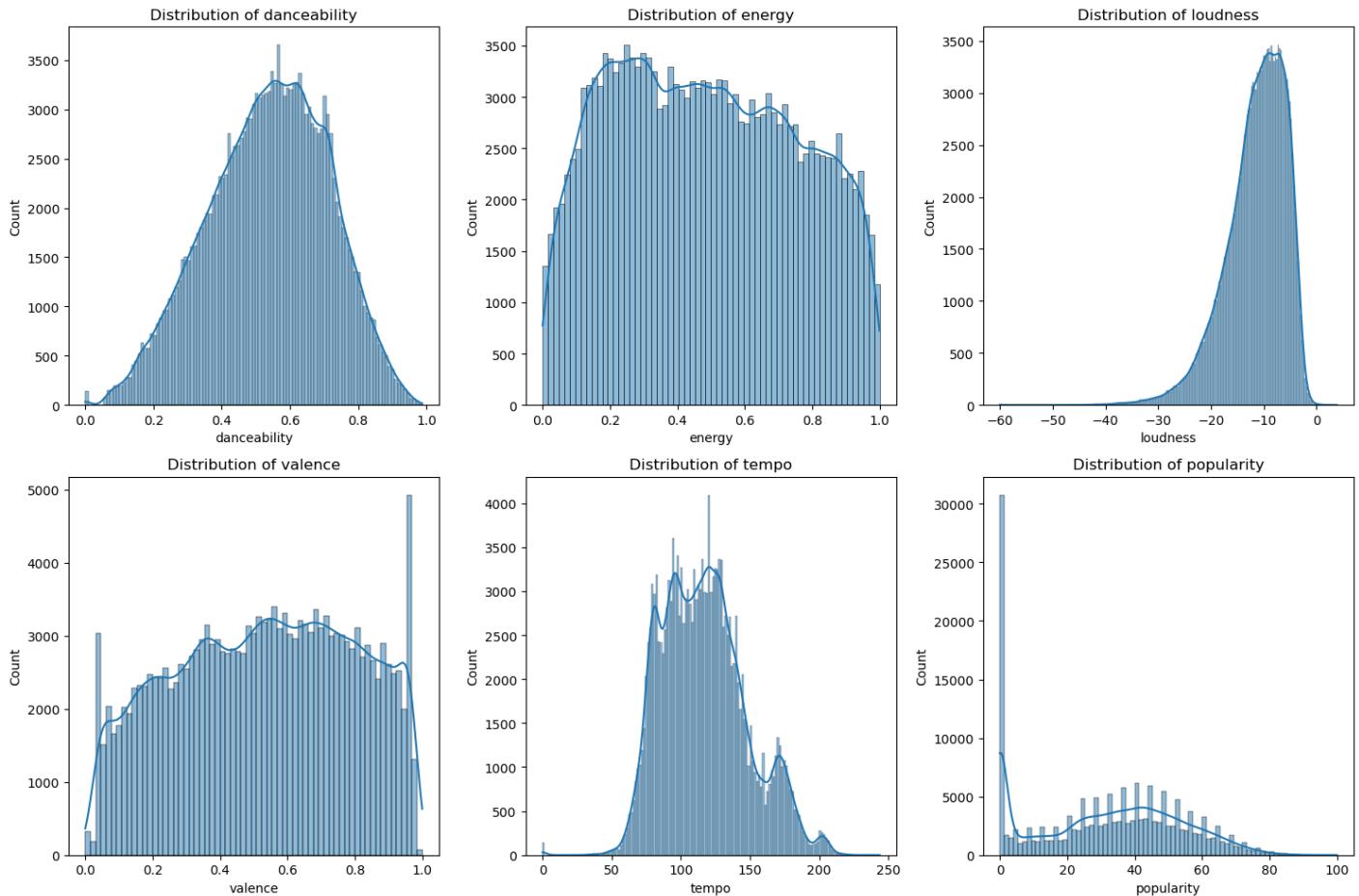
- Danceability
- Energy
- Tempo
- Loudness
- Valence
- Popularity

In [18]:

```
columns = ['danceability', 'energy', 'loudness', 'valence', 'tempo', 'popularity']
```

```
plt.figure(figsize=(15,10))
for i, col in enumerate(columns, 1):
    plt.subplot(2, 3, i)
    sns.histplot(data[col], kde=True)
    plt.title(f'Distribution of {col}')
```

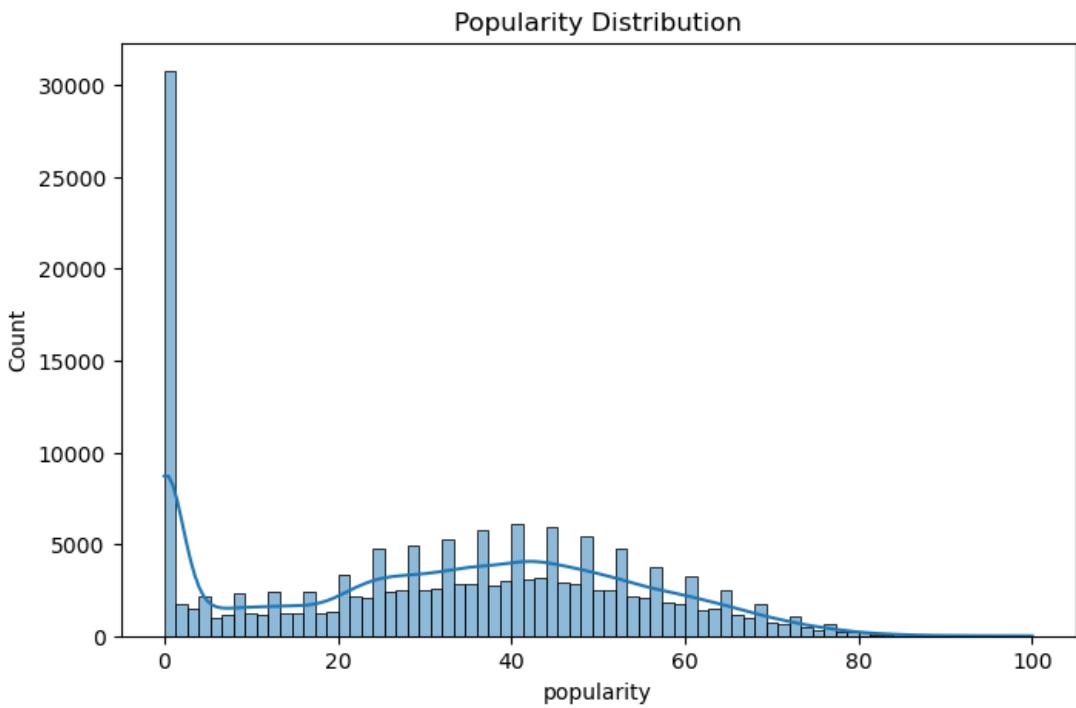
```
plt.tight_layout()
plt.show()
```



Step 5 — Popularity Analysis

In [19]:

```
plt.figure(figsize=(8,5))
sns.histplot(data['popularity'], kde=True)
plt.title("Popularity Distribution")
plt.show()
```



Popularity is not evenly distributed — only a few tracks achieve very high popularity.

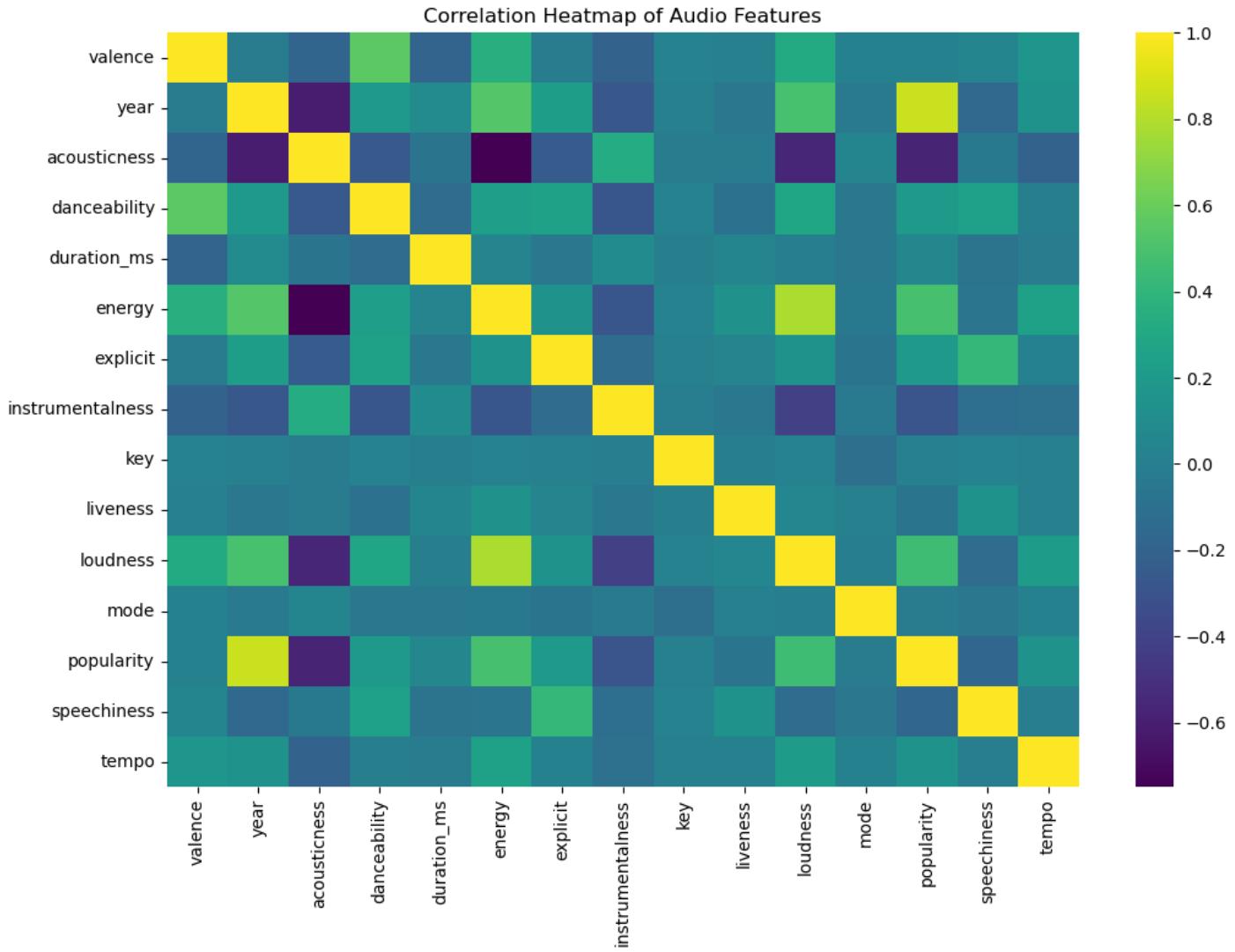
Step 6 — Correlation Heatmap

In [20]:

```
plt.figure(figsize=(12,8))
```

```
# Select ONLY numeric columns
numeric_df = data.select_dtypes(include=['int64', 'float64'])

sns.heatmap(numeric_df.corr(), annot=False, cmap='viridis')
plt.title("Correlation Heatmap of Audio Features")
plt.show()
```



Step 7 — Top Artists by Popularity

In [21]:
`top_artists = data.groupby('artists')['popularity'].mean().sort_values(ascending=False).head(10)`
`top_artists`

Out[21]:

```
artists
['Bad Bunny', 'Jhay Cortez']      100.0
['24kGoldn', 'iann dior']         99.0
['Cardi B', 'Megan Thee Stallion'] 96.0
['Justin Bieber', 'benny blanco'] 95.0
['Joel Corry', 'MNEK']             94.0
['Sech', 'Daddy Yankee', 'J Balvin', 'ROSALÍA', 'Farruko'] 94.0
['Ritt Monney']                   93.0
['Drake', 'Lil Durk']              93.0
['Topic', 'A7S']                  92.0
['Manuel Turizo', 'Rauw Alejandro', 'Myke Towers']       92.0
Name: popularity, dtype: float64
```

Data Analysis

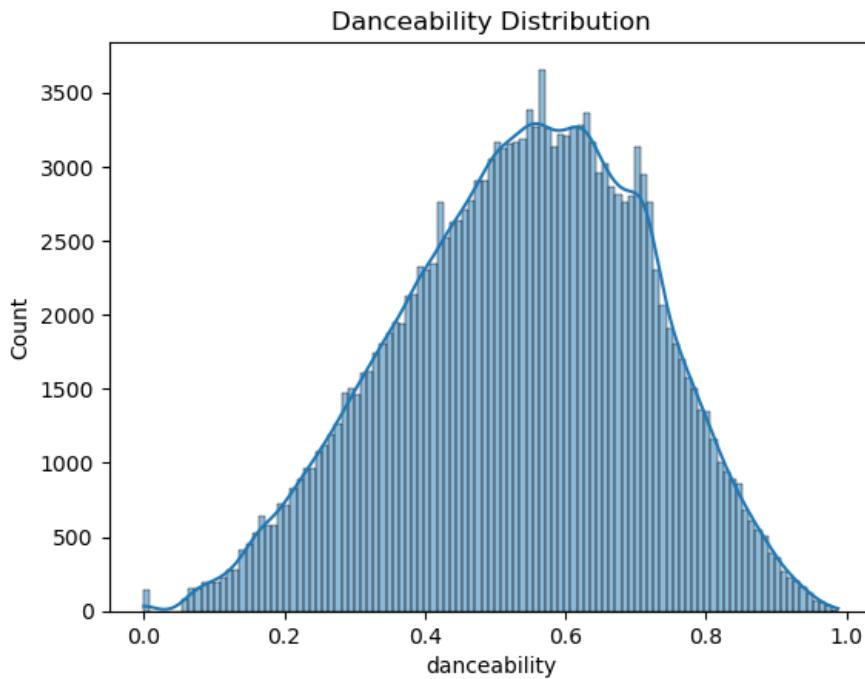
- Task 3.1: Analyze the distribution of various features (e.g., danceability, energy, tempo).
- Task 3.2: Identify trends over time (e.g., popularity of genres, changes in song features).
- Task 3.3: Examine correlations between different features (e.g., energy vs. danceability).
- Task 3.4: Analyze user preferences and listening habits.

Task 3.1: Analyze the distribution of various features (e.g., danceability, energy, tempo).

Danceability Distribution

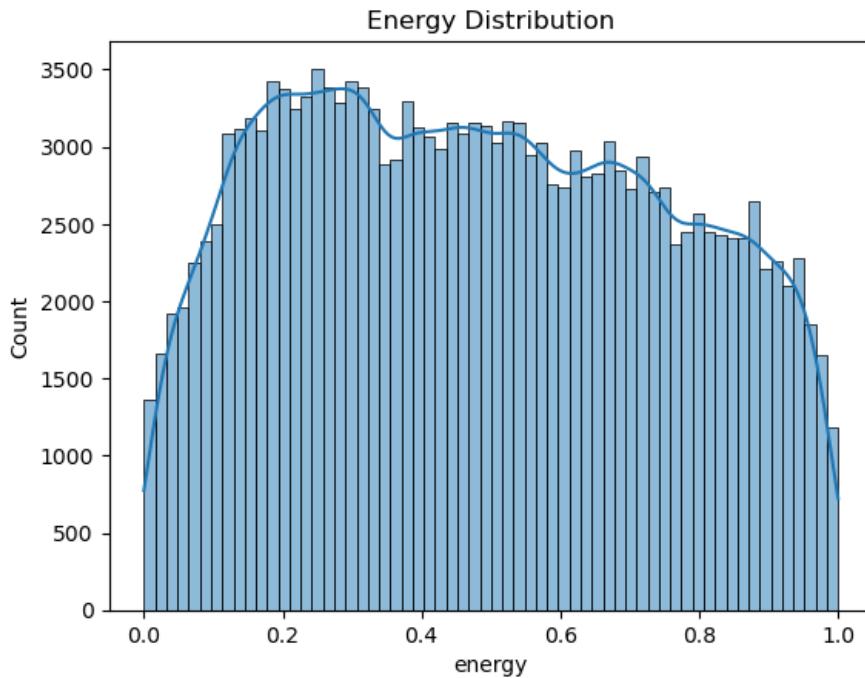
In [22]:

```
sns.histplot(data['danceability'], kde=True)
plt.title('Danceability Distribution')
plt.show()
```



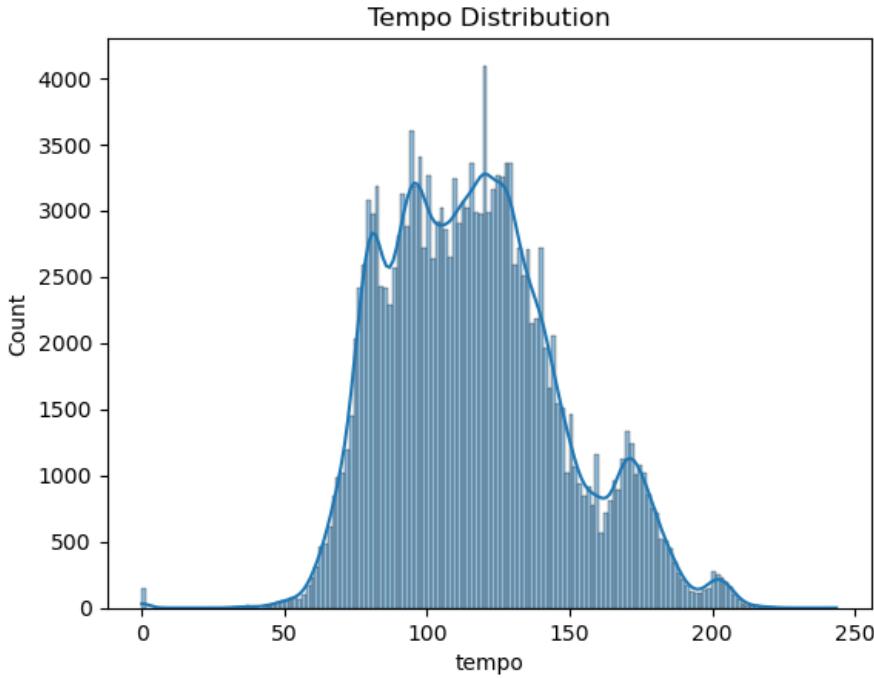
Energy Distribution

```
In [23]:
sns.histplot(data['energy'], kde=True)
plt.title('Energy Distribution')
plt.show()
```



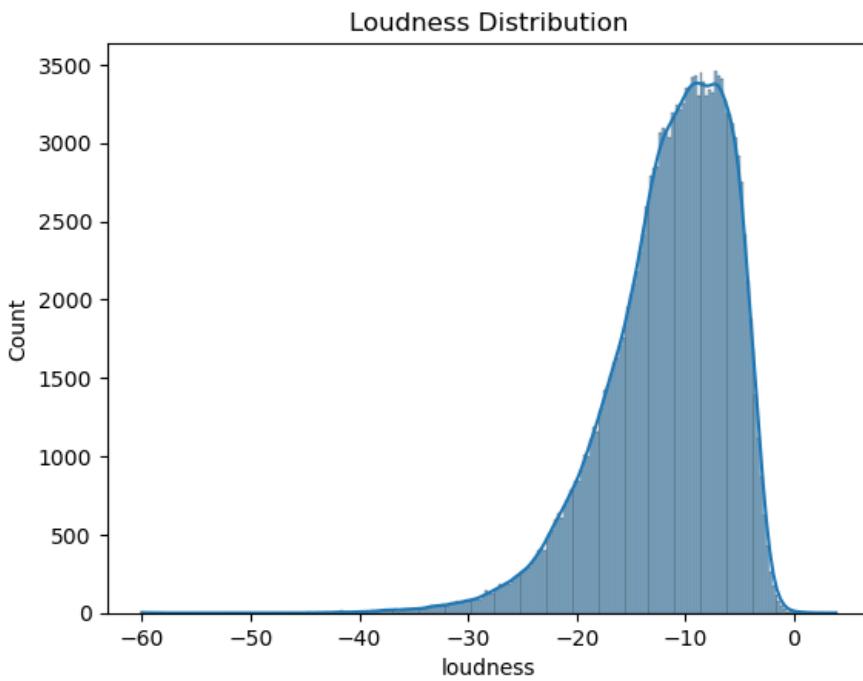
Tempo Distribution

```
In [24]:
sns.histplot(data['tempo'], kde = True)
plt.title('Tempo Distribution')
plt.show()
```



Loudness Distribution

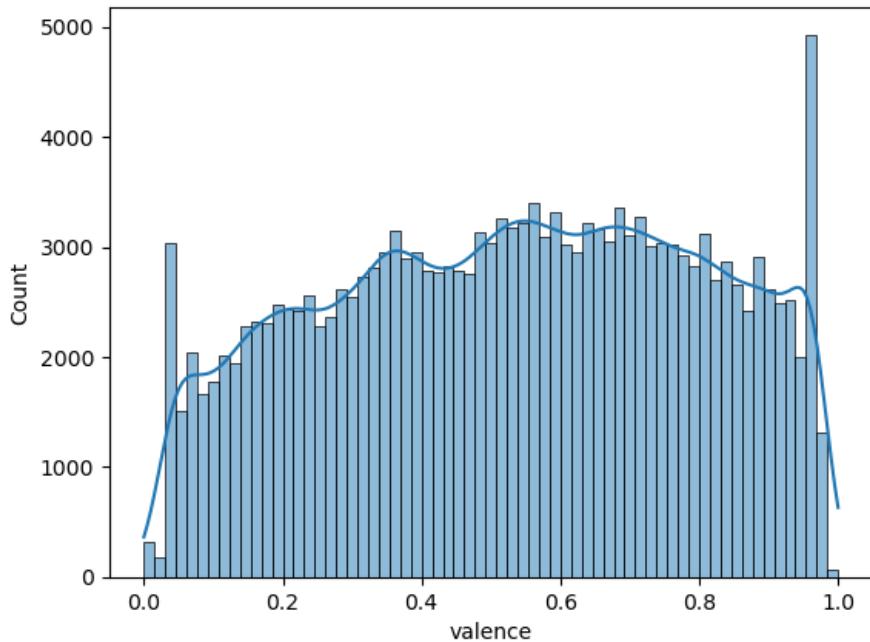
In [25]:
sns.histplot(data['loudness'], kde=True)
plt.title('Loudness Distribution')
plt.show()



Valence (Positive) Distribution

In [26]:
sns.histplot(data['valence'], kde = True)
plt.title('Valence Distribution')
plt.show()

Valence Distribution

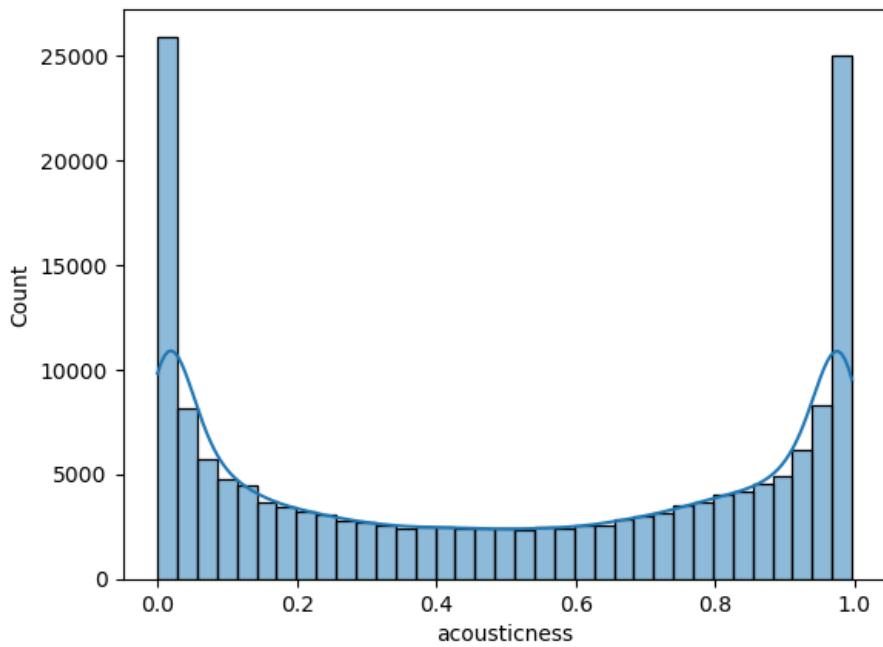


Acousticness Distribution

In [27]:

```
sns.histplot(data['acousticness'], kde=True)  
plt.title('Acousticness Distribution')  
plt.show()
```

Acousticness Distribution

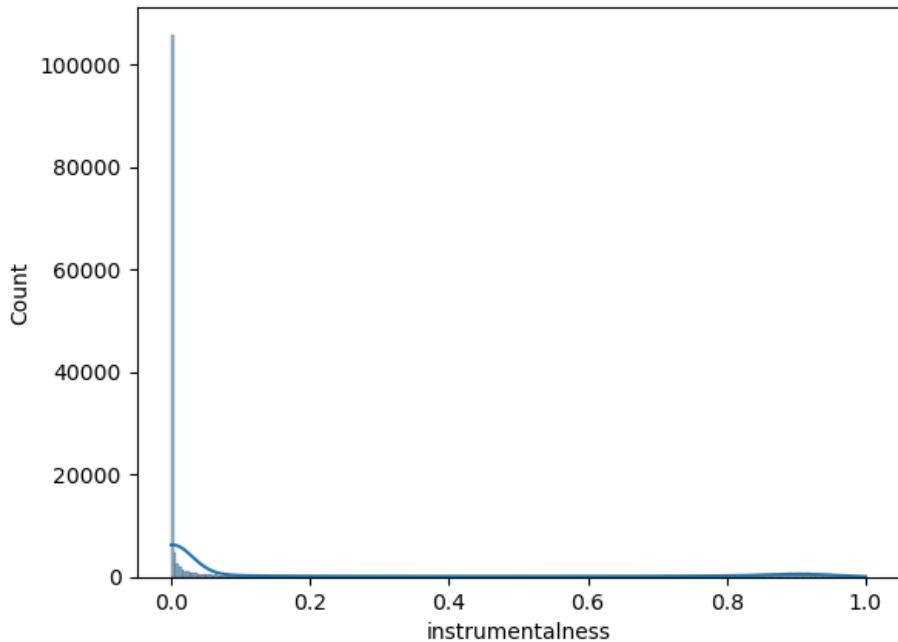


Instrumentalness Distribution

In [28]:

```
sns.histplot(data['instrumentalness'], kde=True)  
plt.title('Instrumentalness Distribution')  
plt.show()
```

Instrumentalness Distribution

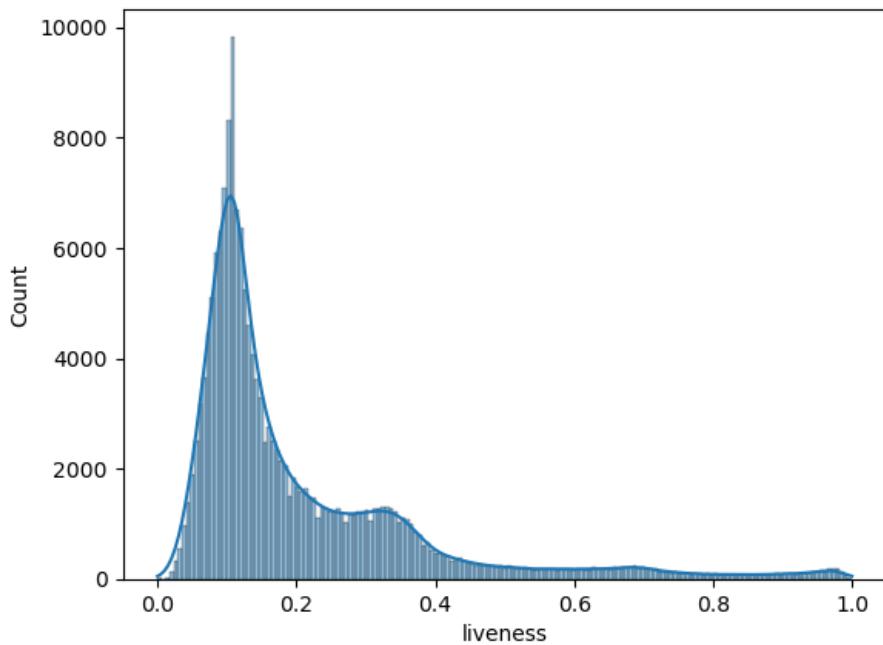


Liveness Distribution

In [29]:

```
sns.histplot(data['liveness'], kde=True)
plt.title('Liveness Distribution')
plt.show()
```

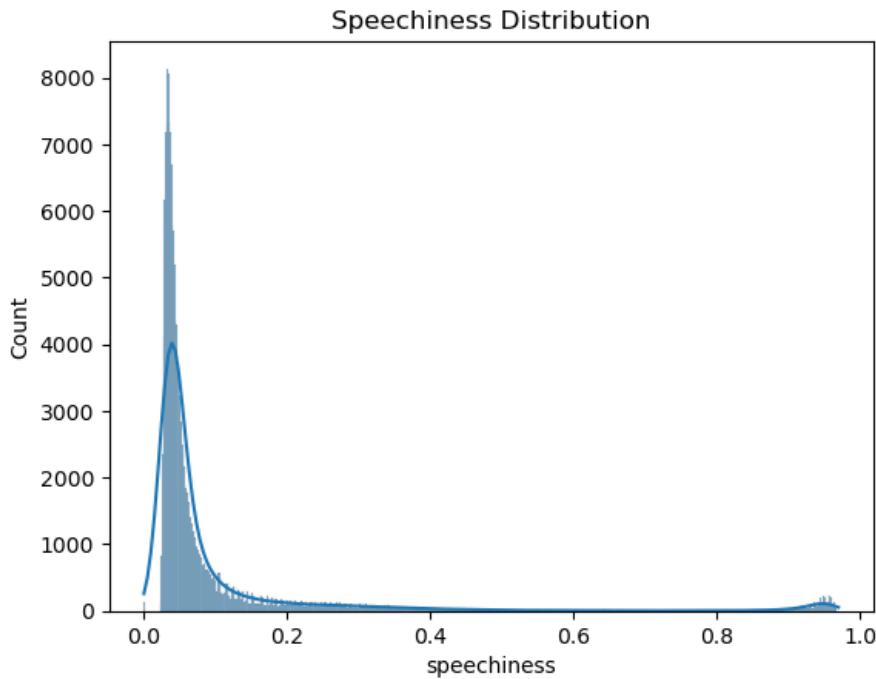
Liveness Distribution



Speechiness Distribution

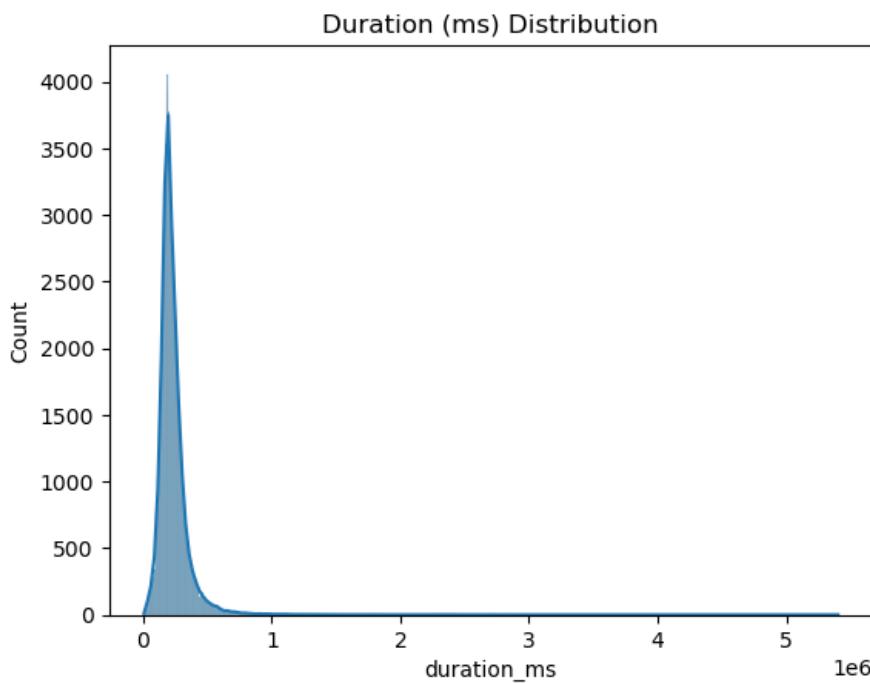
In [30]:

```
sns.histplot(data['speechiness'], kde=True)
plt.title('Speechiness Distribution')
plt.show()
```



Duration Distribution

In [31]:
sns.histplot(data['duration_ms'], kde = True)
plt.title('Duration (ms) Distribution')
plt.show()

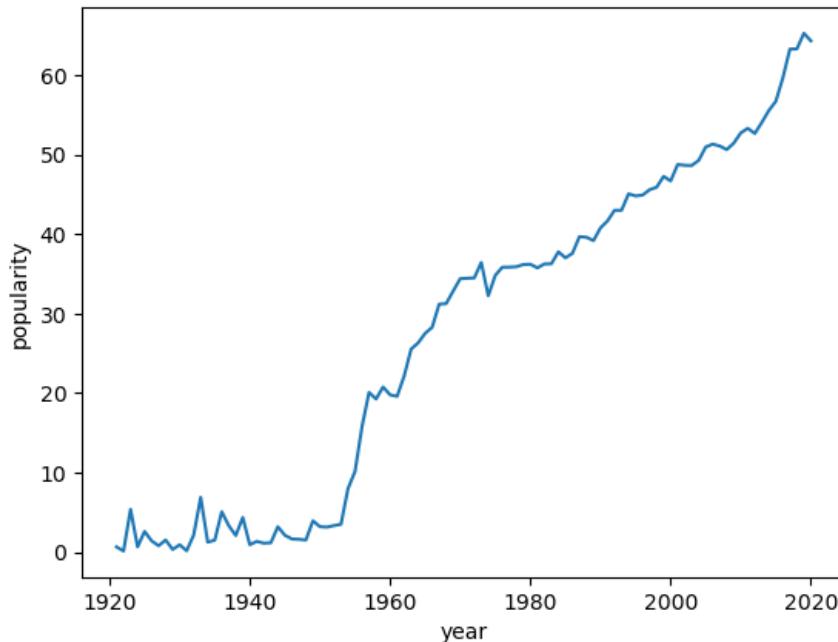


Task 3.2: Identify trends over time (e.g., popularity of genres, changes in song features).

Popularity Trend Over Time

In [32]:
sns.lineplot(x = 'year', y = 'popularity', data = data_by_year)
plt.title('Popularity Trend Over Years')
plt.show()

Popularity Trend Over Years

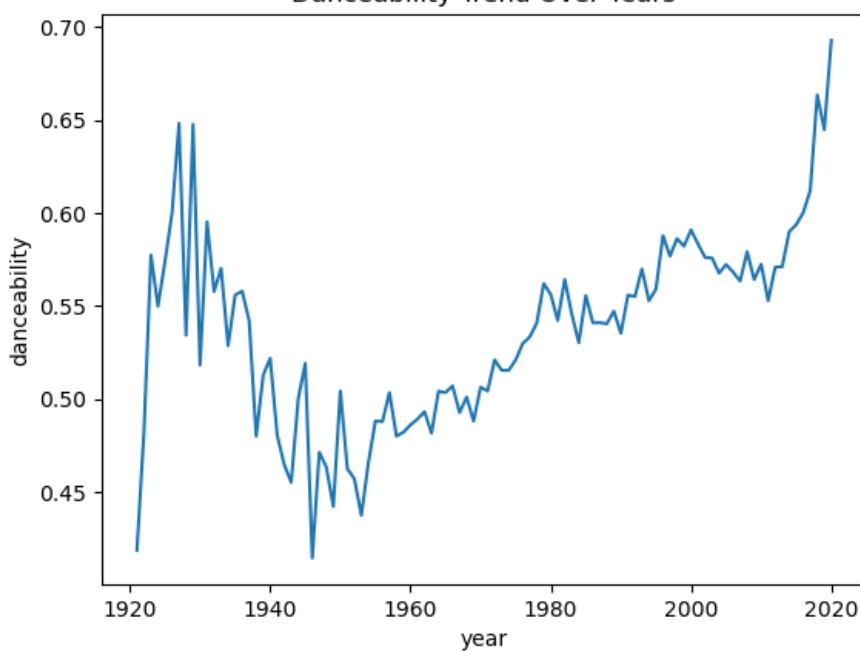


Danceability Trend Over Time

In [33]:

```
sns.lineplot(x = 'year', y = 'danceability', data = data_by_year)
plt.title('Danceability Trend Over Years')
plt.show()
```

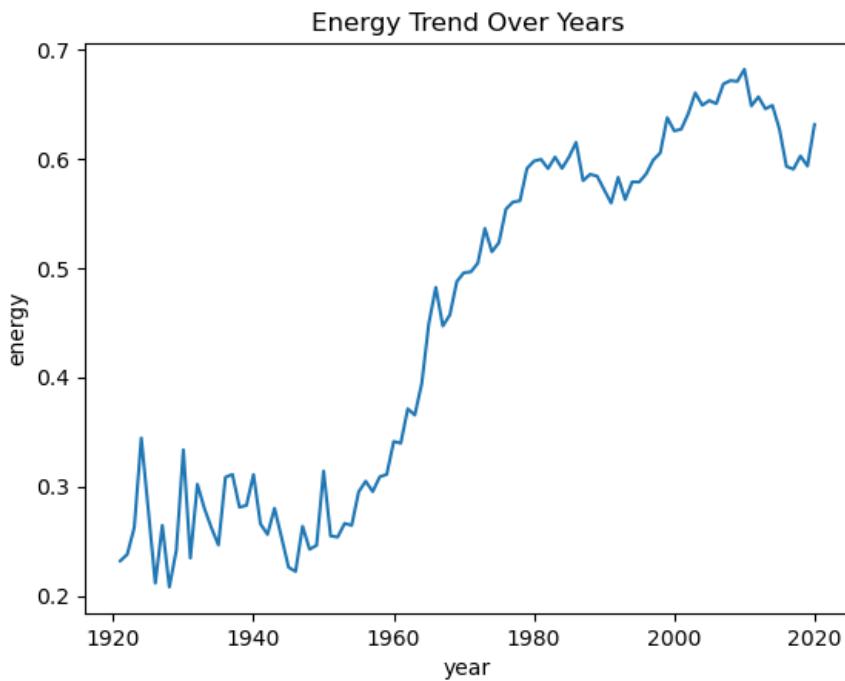
Danceability Trend Over Years



Energy Trend Over Time

In [34]:

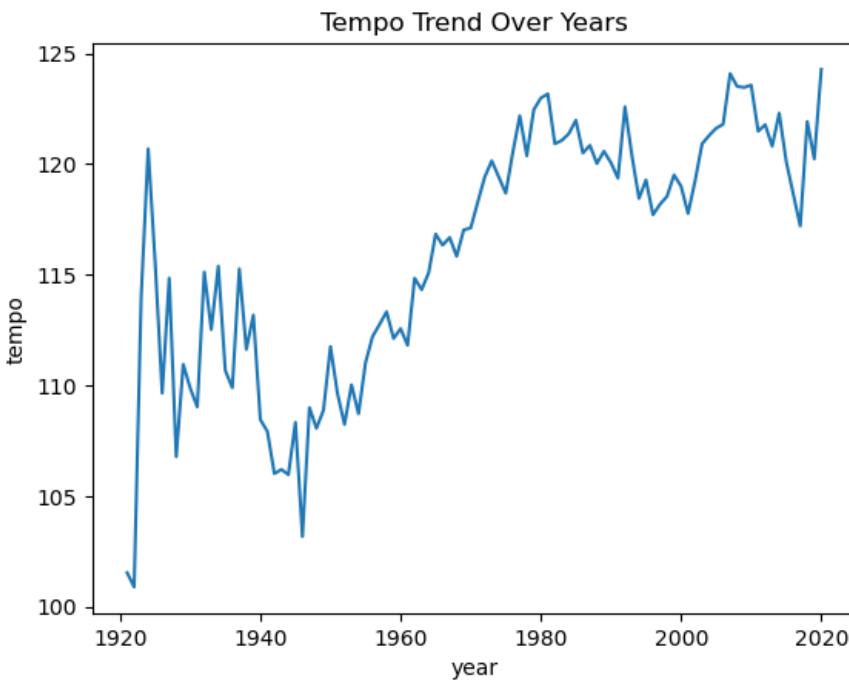
```
sns.lineplot(x = 'year', y = 'energy', data = data_by_year)
plt.title('Energy Trend Over Years')
plt.show()
```



Tempo Trend Over Time

In [35]:

```
sns.lineplot(x = 'year', y = 'tempo', data = data_by_year)
plt.title('Tempo Trend Over Years')
plt.show()
```

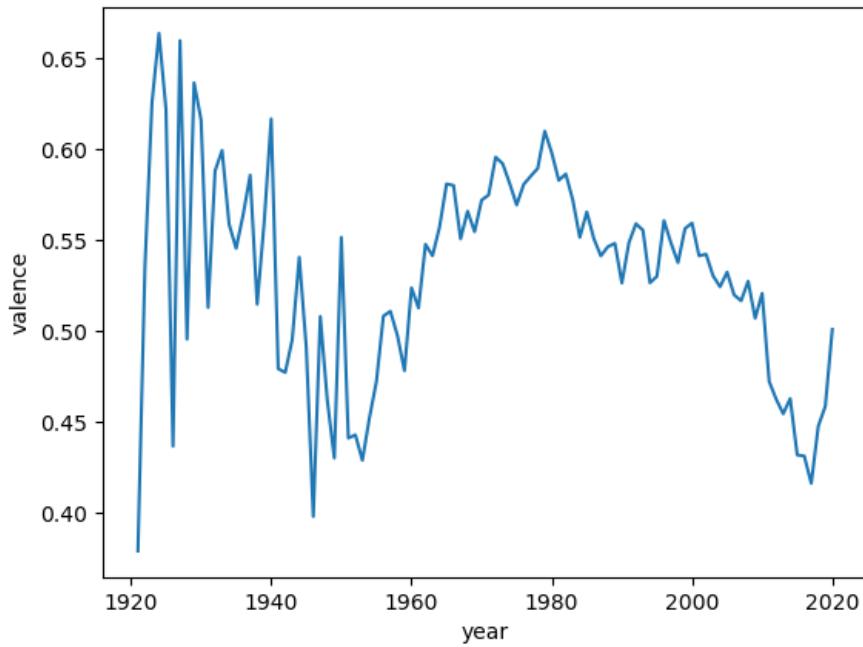


Valence (Positivity) Trend Over Time

In [36]:

```
sns.lineplot(x = 'year', y = 'valence', data = data_by_year)
plt.title('Valence Trend Over Years')
plt.show()
```

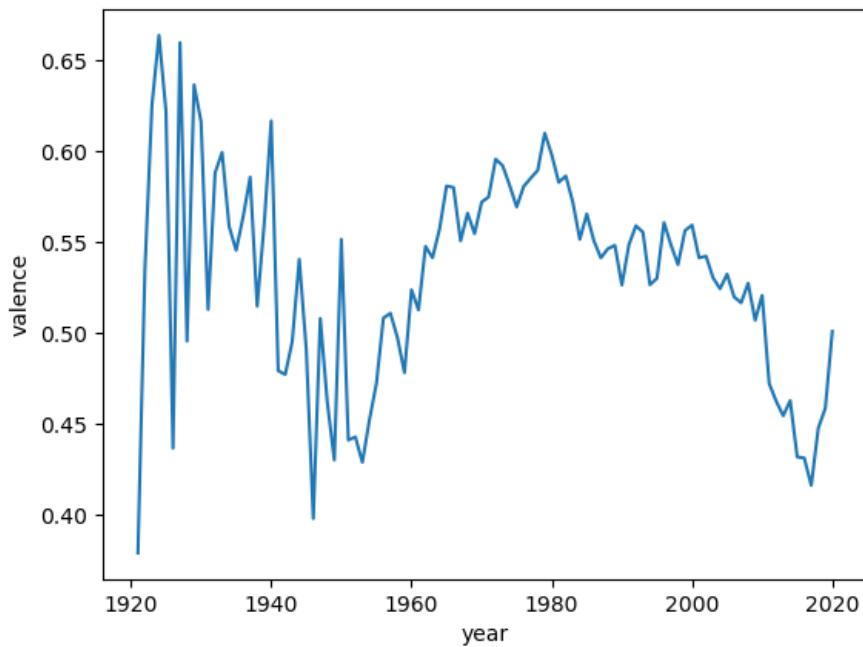
Valence Trend Over Years



Acousticness Trend Over Time

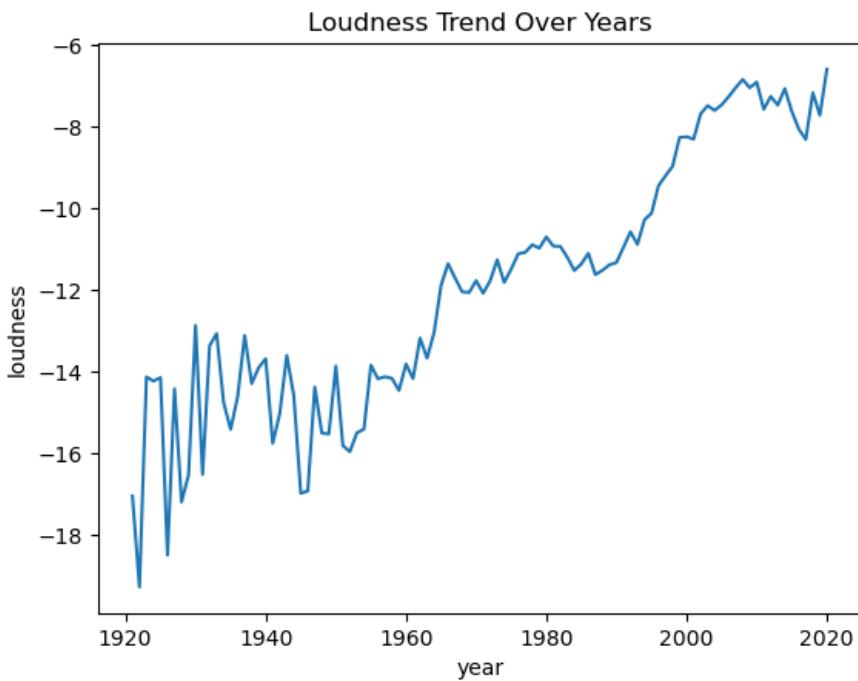
In [37]:
sns.lineplot(x = 'year', y = 'valence', data = data_by_year)
plt.title('Acousticness Trend Over Time')
plt.show()

Acousticness Trend Over Time



Loudness Trend Over Time

In [38]:
sns.lineplot(x = 'year', y = 'loudness', data = data_by_year)
plt.title('Loudness Trend Over Years')
plt.show()



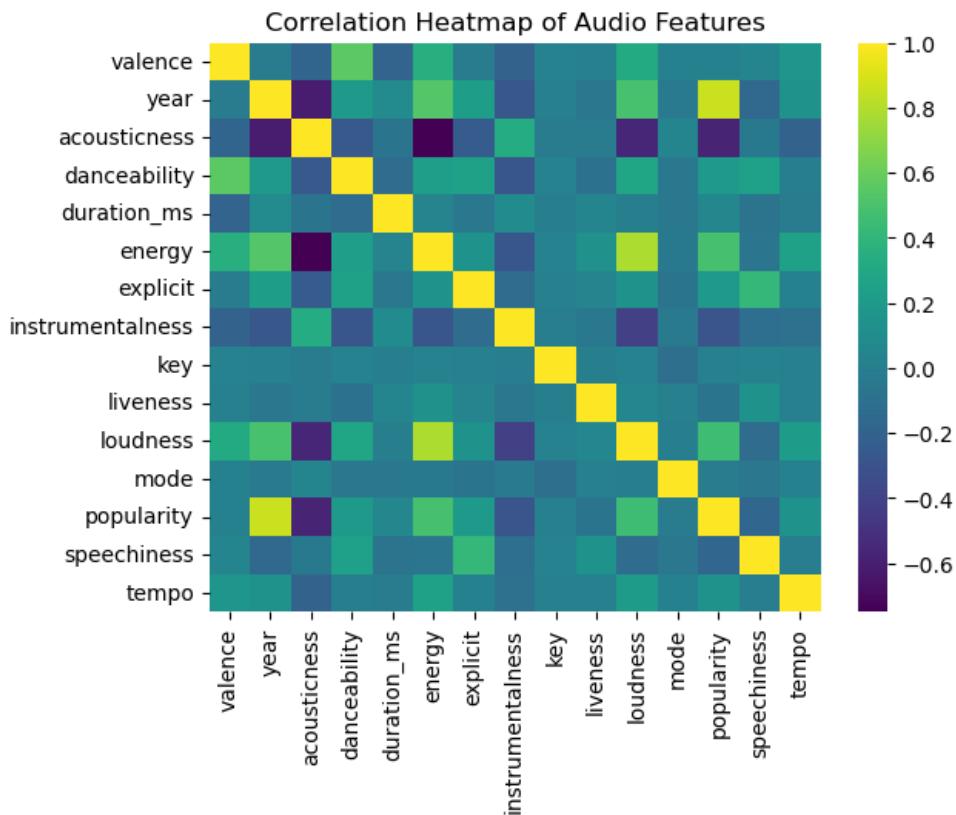
Task 3.3: Examine correlations between different features (e.g., energy vs. danceability)

Simple Correlation Heatmap

In [39]:

```
numeric_df = data.select_dtypes(include=['int64', 'float64'])
```

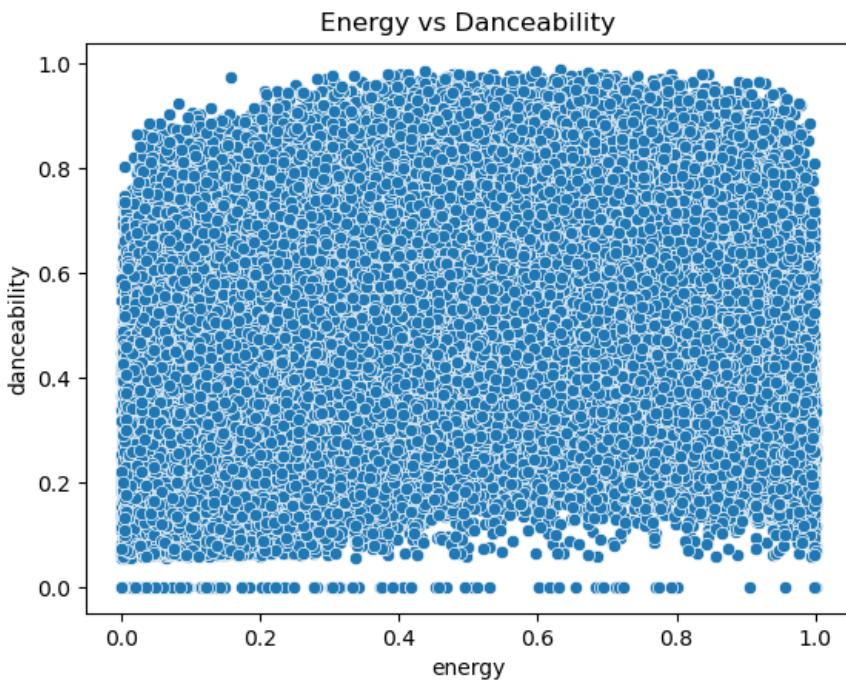
```
sns.heatmap(numeric_df.corr(), cmap='viridis')
plt.title("Correlation Heatmap of Audio Features")
plt.show()
```



Scatter Plot: Energy vs Danceability

In [40]:

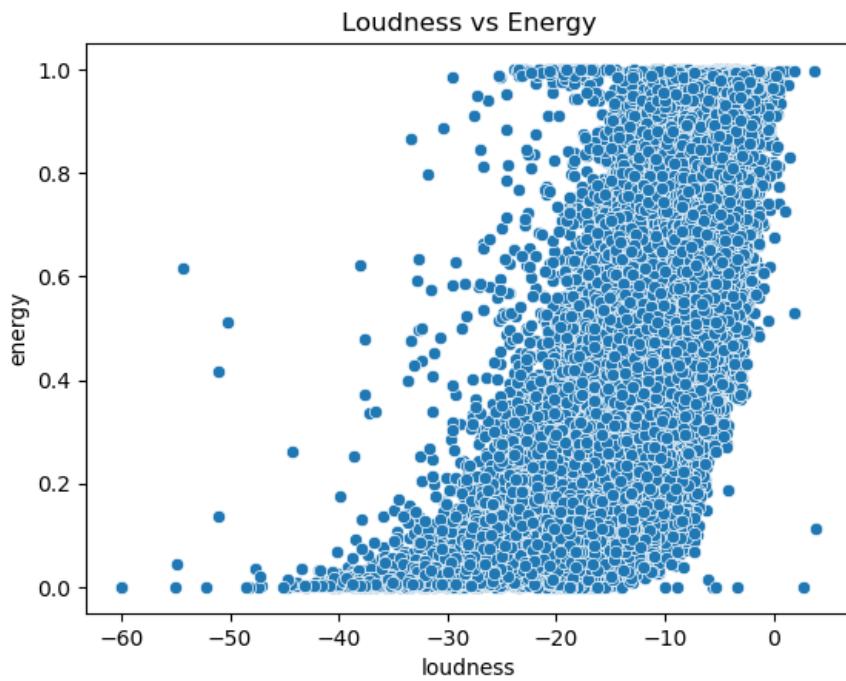
```
sns.scatterplot(x = 'energy', y = 'danceability', data = data)
plt.title('Energy vs Danceability')
plt.show()
```



Scatter Plot: Loudness vs Energy

In [41]:

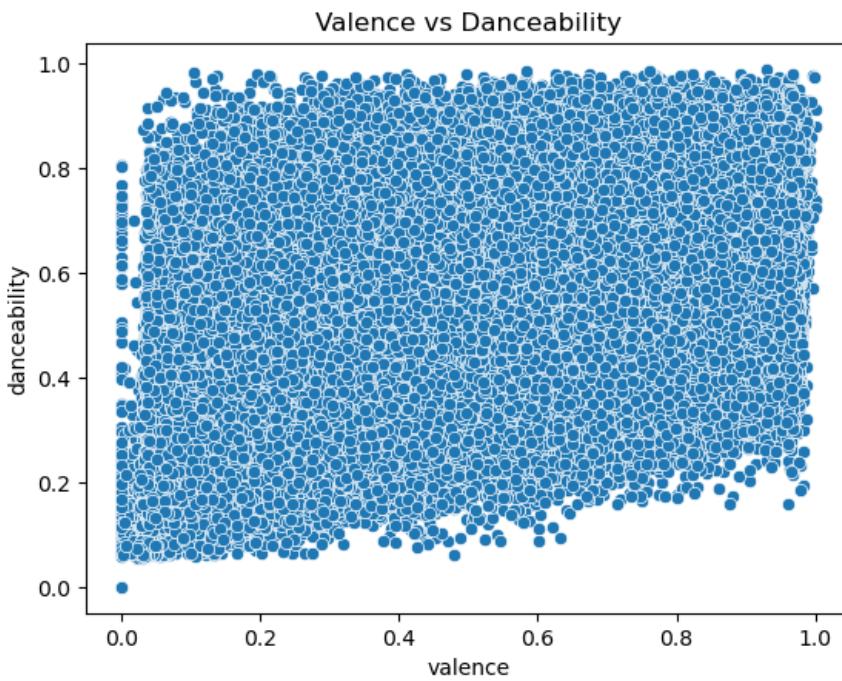
```
sns.scatterplot(x='loudness', y='energy', data=data)
plt.title('Loudness vs Energy')
plt.show()
```



Scatter Plot: Valence vs Danceability

In [42]:

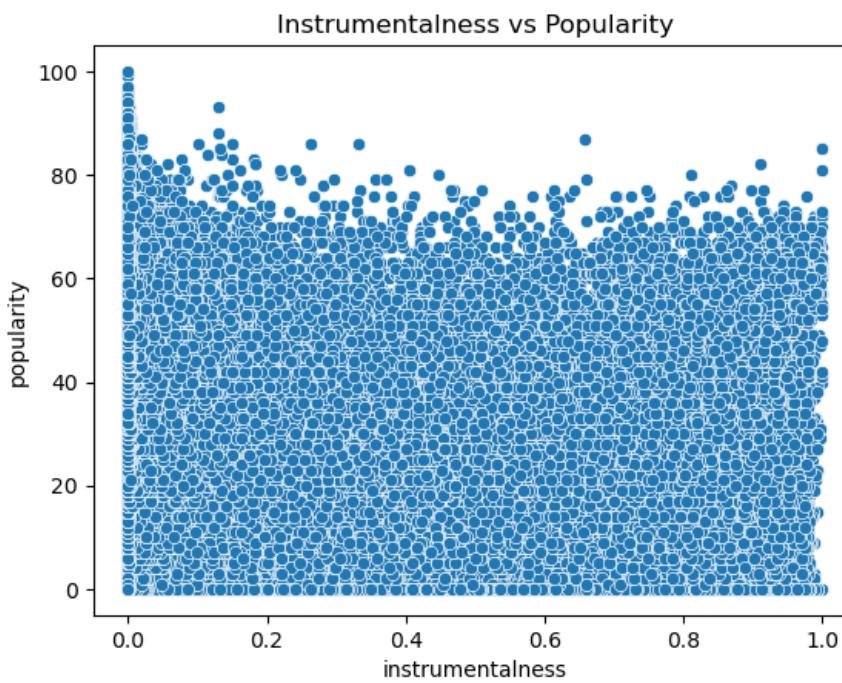
```
sns.scatterplot(x = 'valence', y = 'danceability', data = data)
plt.title('Valence vs Danceability')
plt.show()
```



Scatter Plot: Instrumentalness vs Popularity

In [43]:

```
sns.scatterplot(x = 'instrumentalness', y = 'popularity', data = data)
plt.title('Instrumentalness vs Popularity')
plt.show()
```



Task 3.4: Analyze user preferences and listening habits.

What Features Do Users Prefer?

In [44]:

```
popular_songs = data[data['popularity'] > 80] # top popular songs
popular_songs.describe()
```

Out[44]:

	valence	year	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	liveness	tempo
count	549.000000	549.000000	549.000000	549.000000	549.000000	549.000000	549.000000	549.000000	549.000000	549.000000
mean	0.495195	2016.041894	0.250548	0.678193	203792.218579	0.612156	0.357013	0.013029	5.28	110.25
min	0.000000	1959.000000	0.000147	0.000000	90228.000000	0.001480	0.000000	0.000000	0.000000	0.00
25%	0.323000	2017.000000	0.046600	0.599000	175721.000000	0.511000	0.000000	0.000000	0.000000	2.00
50%	0.492000	2019.000000	0.162000	0.694000	200186.000000	0.630000	0.000000	0.000000	0.000000	6.00
75%	0.668000	2020.000000	0.374000	0.779000	224187.000000	0.733000	1.000000	0.000051	8.00	110.00
max	0.969000	2020.000000	0.978000	0.954000	484147.000000	0.970000	1.000000	1.000000	1.000000	11.00
std	0.230968	9.440946	0.257509	0.143164	44087.873385	0.169985	0.479556	0.083988	3.62	10.00

Popular Genres (User Preference by Genre)

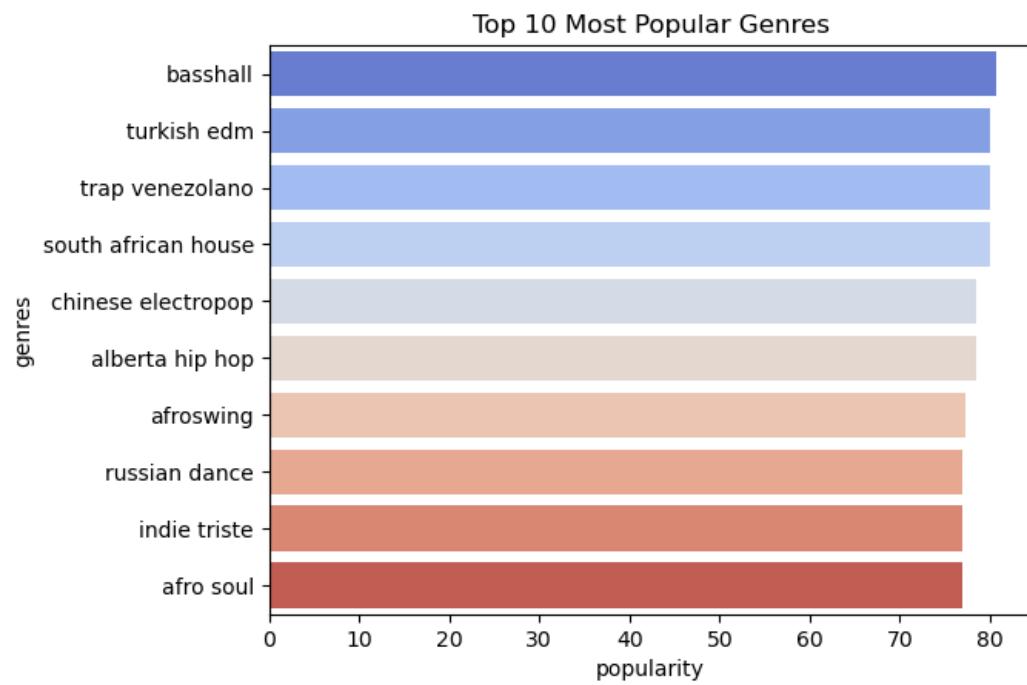
In [45]:

```
sns.barplot(x='popularity', y='genres', data = data_by_genres.sort_values('popularity', ascending=False).head(10), palette = 'coolwarm')
plt.title("Top 10 Most Popular Genres")
plt.show()
```

C:\Users\aryan\AppData\Local\Temp\ipykernel_20928\1354952020.py:1: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'y' variable to 'hue' and set 'legend=False' for the same effect.

```
sns.barplot(x='popularity', y='genres', data = data_by_genres.sort_values('popularity', ascending=False).head(10), palette = 'coolwarm')
```



Top Artists (User Listening Habits)

In [46]:

```
top_artists = data.groupby('artists')['popularity'].mean().sort_values(ascending=False).head(10)
```

```
sns.barplot(x = top_artists.values, y = top_artists.index, palette = 'viridis')
```

```
plt.title("Top 10 Most Popular Artists")
```

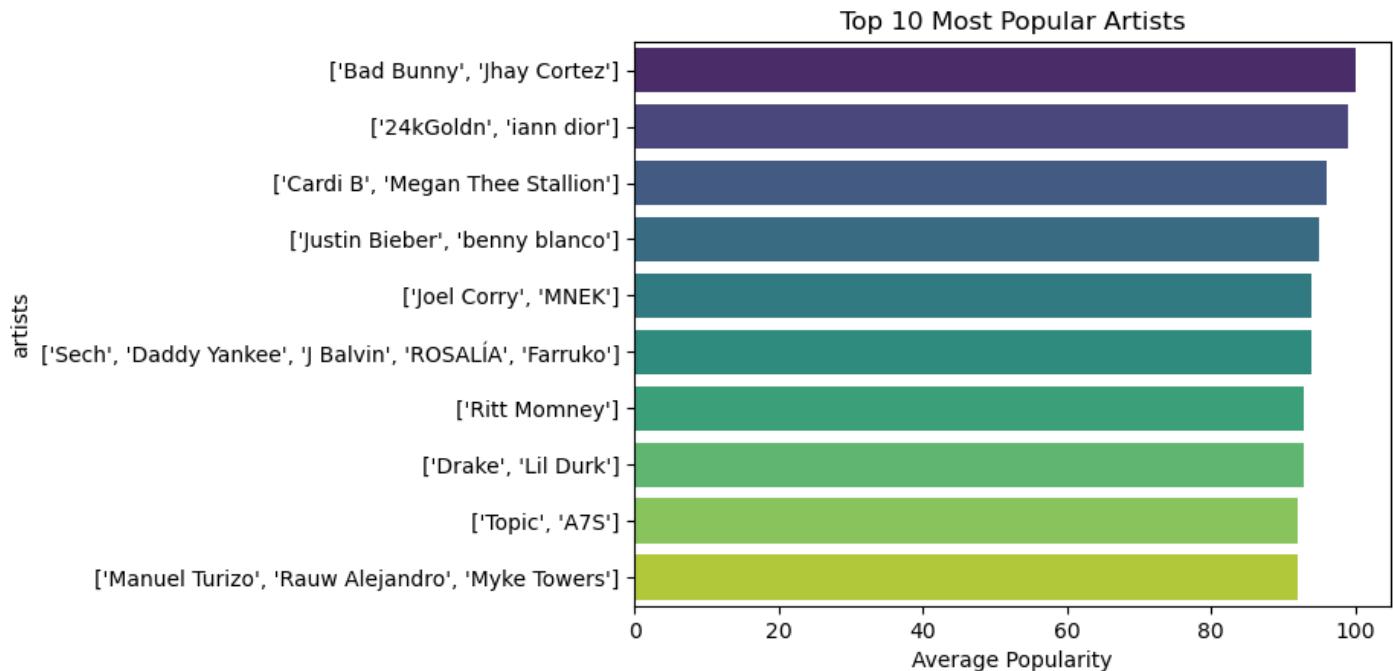
```
plt.xlabel("Average Popularity")
```

```
plt.show()
```

C:\Users\aryan\AppData\Local\Temp\ipykernel_20928\4145645061.py:3: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'y' variable to 'hue' and set 'legend=False' for the same effect.

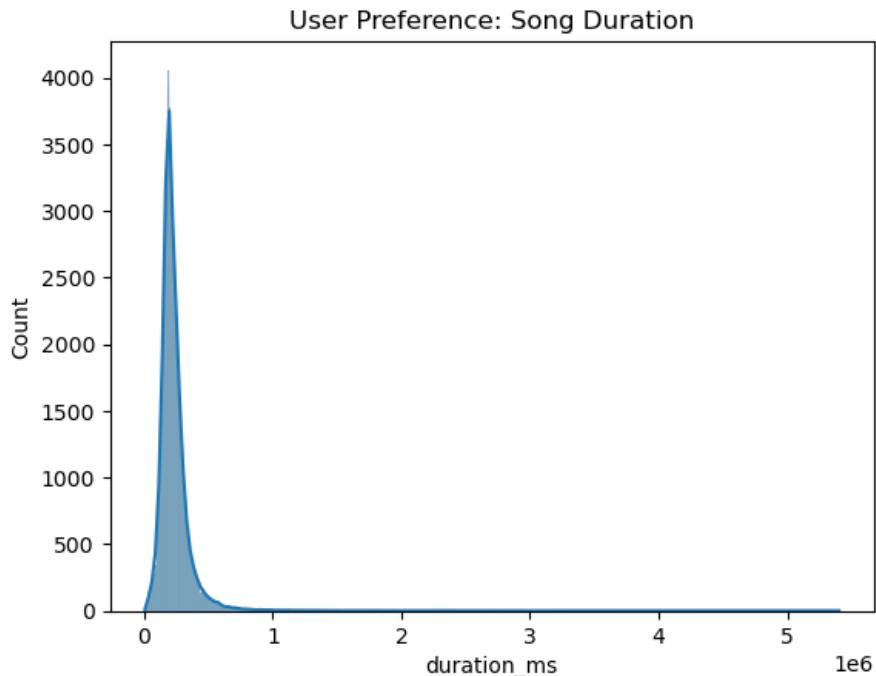
```
sns.barplot(x=top_artists.values, y=top_artists.index, palette = 'viridis')
```



User Preference Based on Song Length

In [47]:

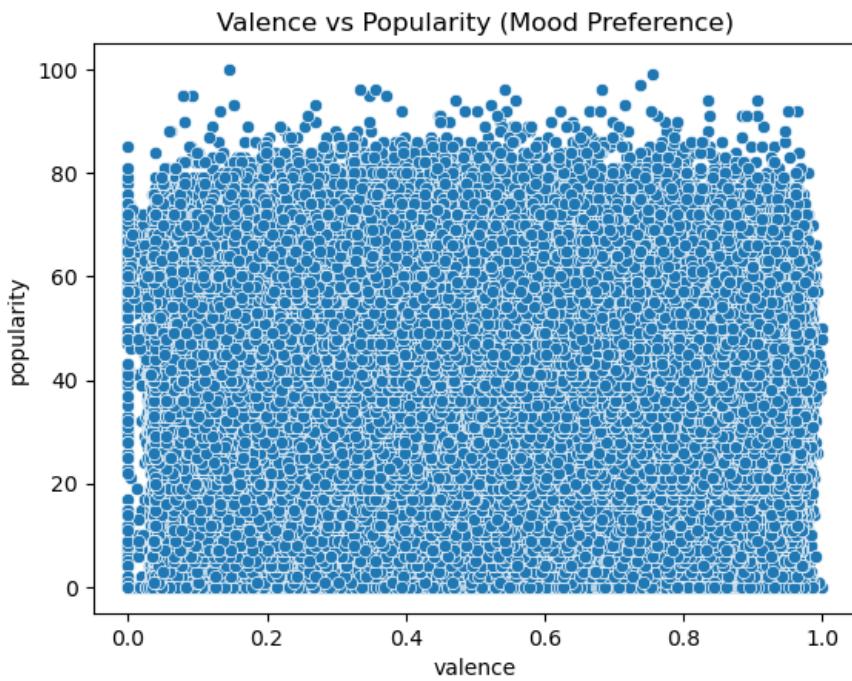
```
sns.histplot(data['duration_ms'], kde=True)
plt.title("User Preference: Song Duration")
plt.show()
```



User Preference Based on Valence (Mood)

In [48]:

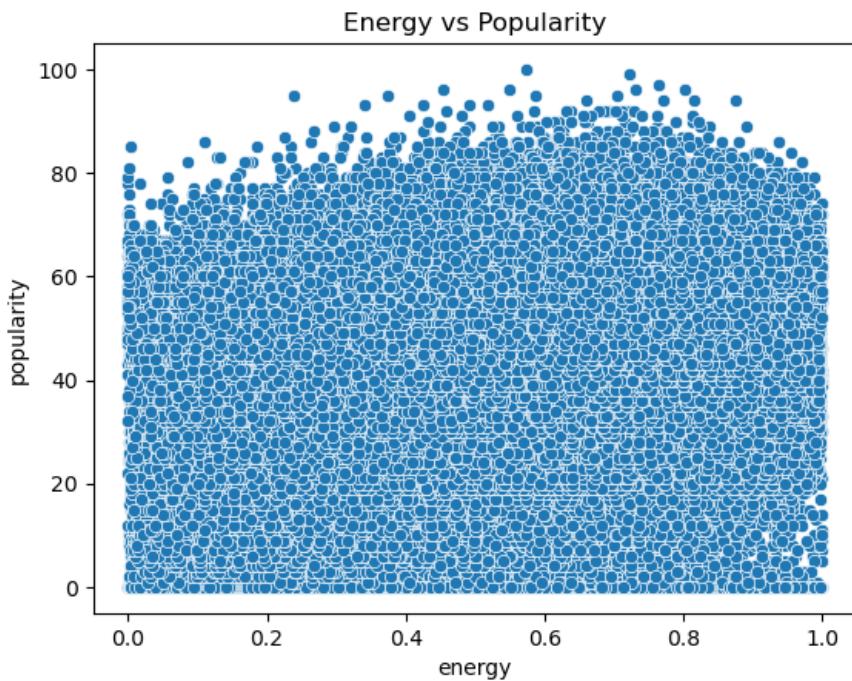
```
sns.scatterplot(x='valence', y='popularity', data=data)
plt.title("Valence vs Popularity (Mood Preference)")
plt.show()
```



User Preference: Energy vs Popularity

In [49]:

```
sns.scatterplot(x='energy', y='popularity', data=data)
plt.title("Energy vs Popularity")
plt.show()
```



Visualization

- Task 4.1: Create visualizations to represent key findings (e.g., bar charts, line graphs, scatter plots).
- Task 4.2: Use advanced visualization techniques (e.g., heatmaps, pair plots) to uncover deeper insights.

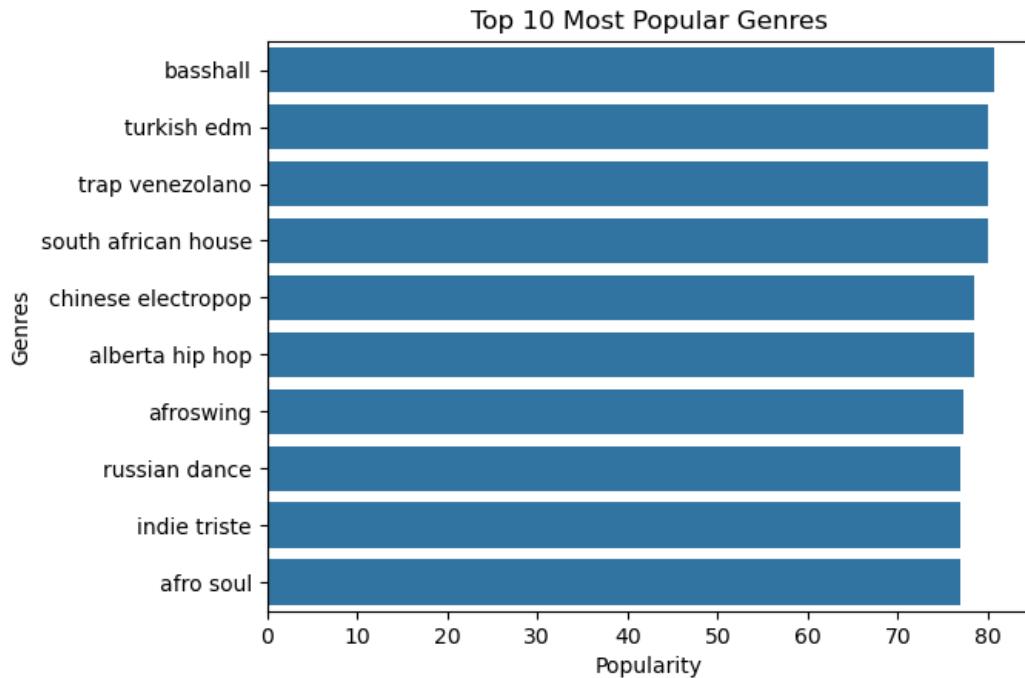
Task 4.1: Create visualizations to represent key findings (e.g., bar charts, line graphs, scatter plots).

Top 10 Most Popular Genres

In [50]:

```
top_genres = data_by_genres.sort_values('popularity', ascending=False).head(10)
```

```
sns.barplot(x = top_genres['popularity'], y = top_genres['genres'])
plt.title("Top 10 Most Popular Genres")
plt.xlabel("Popularity")
plt.ylabel("Genres")
plt.show()
```



Top 10 Most Popular Artists

In [51]:

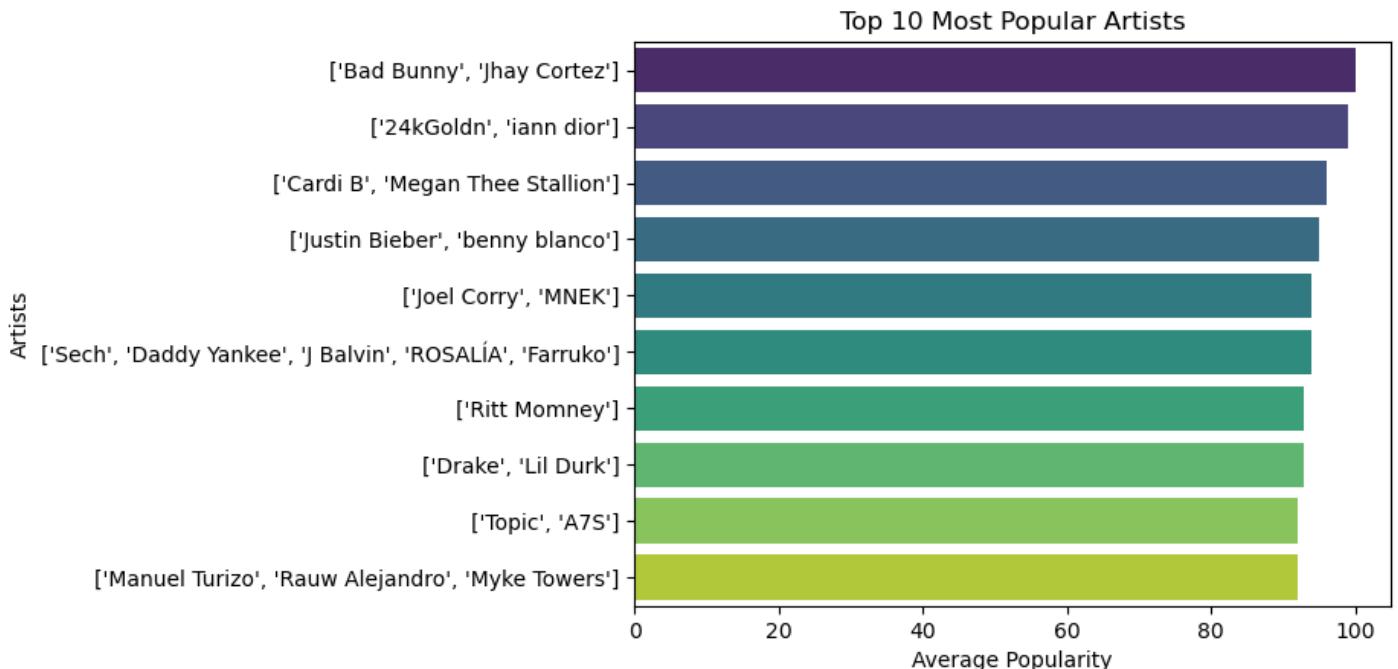
```
top_artists = data.groupby('artists')['popularity'].mean().sort_values(ascending=False).head(10)
```

```
sns.barplot(x = top_artists.values, y = top_artists.index, palette = 'viridis')
plt.title("Top 10 Most Popular Artists")
plt.xlabel("Average Popularity")
plt.ylabel("Artists")
plt.show()
```

C:\Users\aryan\AppData\Local\Temp\ipykernel_20928\207210082.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

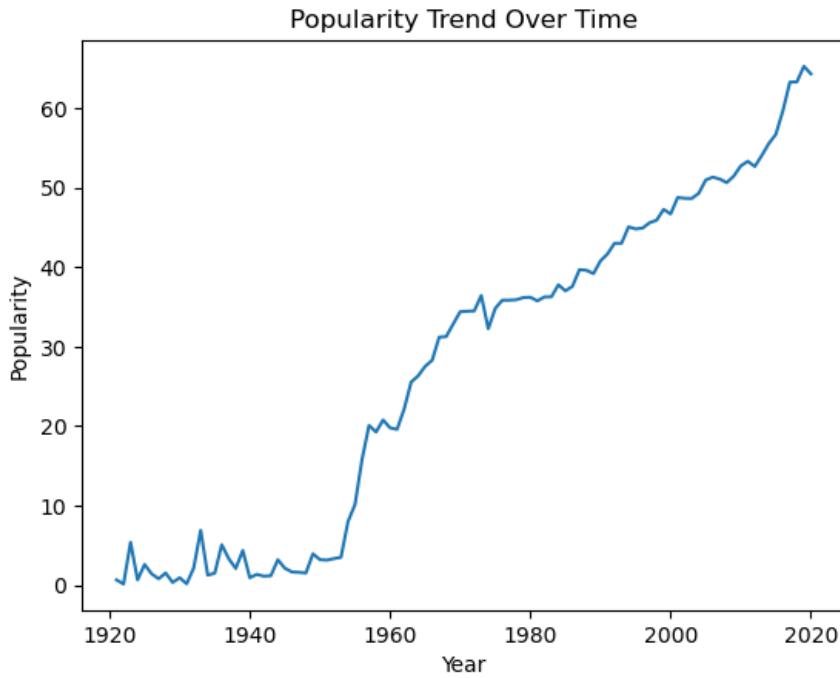
```
sns.barplot(x = top_artists.values, y = top_artists.index, palette = 'viridis')
```



Popularity Trend Over Years

In [52]:

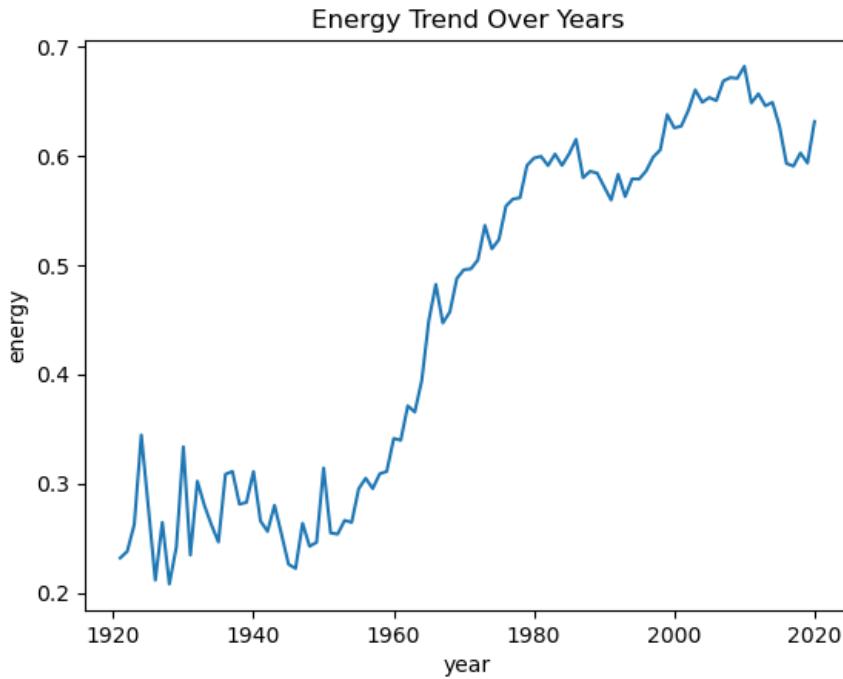
```
sns.lineplot(x = 'year', y = 'popularity', data = data_by_year)
plt.title('Popularity Trend Over Time')
plt.xlabel('Year')
plt.ylabel('Popularity')
plt.show()
```



Energy Trend Over Time

In [53]:

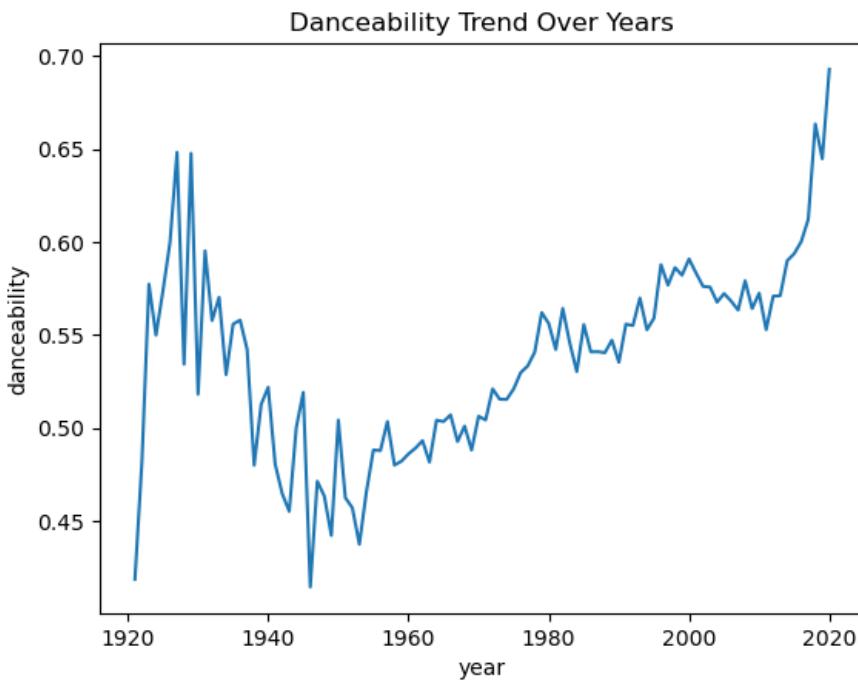
```
sns.lineplot(x = 'year', y = 'energy', data = data_by_year)
plt.title("Energy Trend Over Years")
plt.show()
```



Danceability Trend Over Time

In [54]:

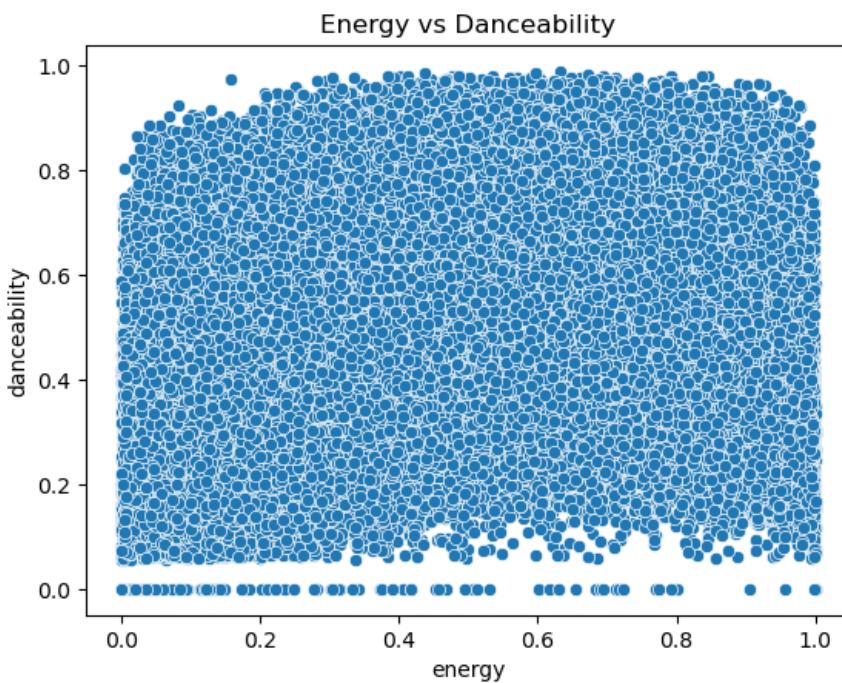
```
sns.lineplot(x = 'year', y = 'danceability', data = data_by_year)
plt.title("Danceability Trend Over Years")
plt.show()
```



Scatter Plot: Energy vs Danceability

In [55]:

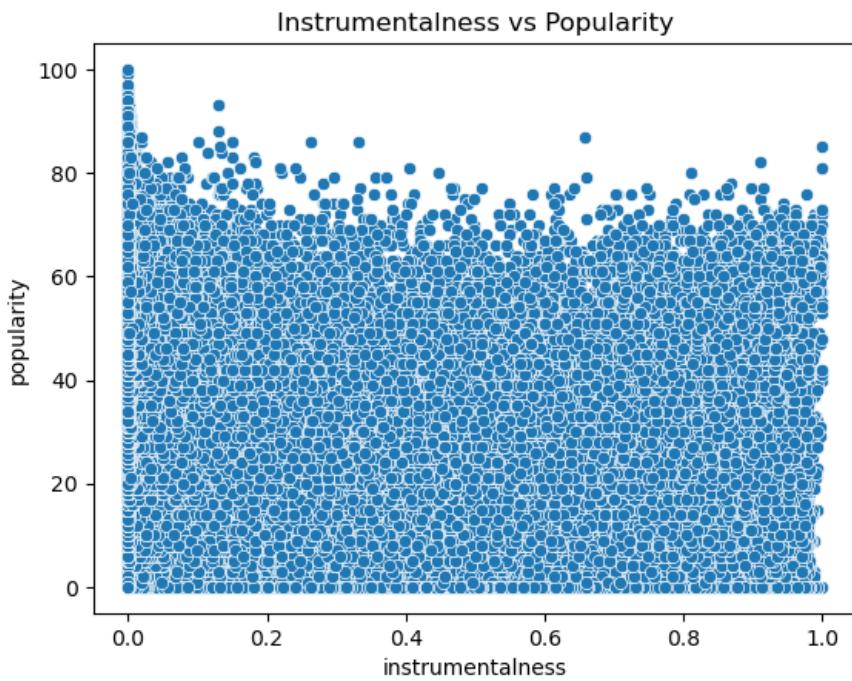
```
sns.scatterplot(x = 'energy', y = 'danceability', data = data)
plt.title("Energy vs Danceability")
plt.show()
```



Scatter Plot: Instrumentalness vs Popularity

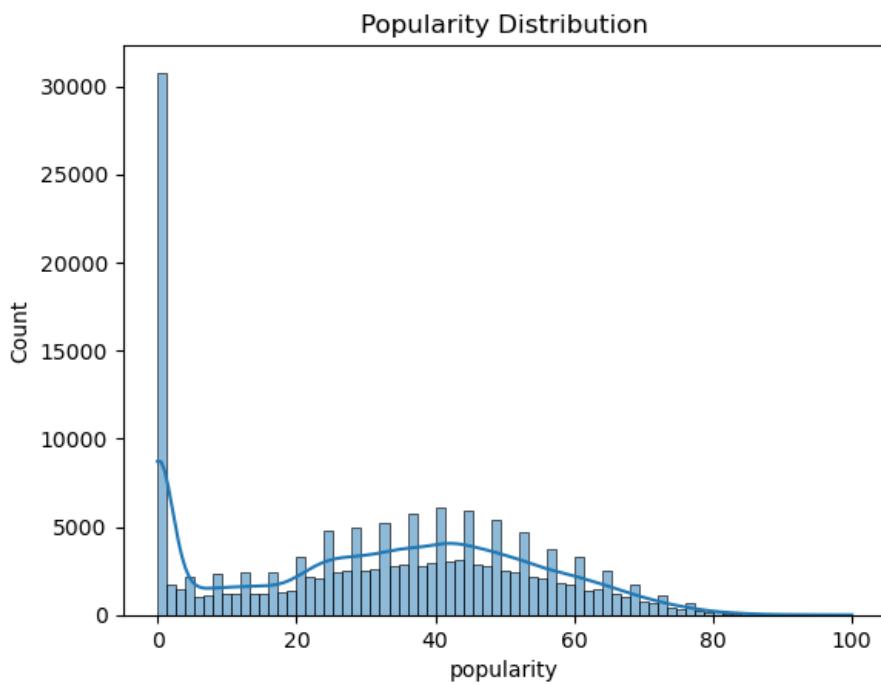
In [56]:

```
sns.scatterplot(x = 'instrumentalness', y = 'popularity', data = data)
plt.title("Instrumentalness vs Popularity")
plt.show()
```



Distribution of Popularity

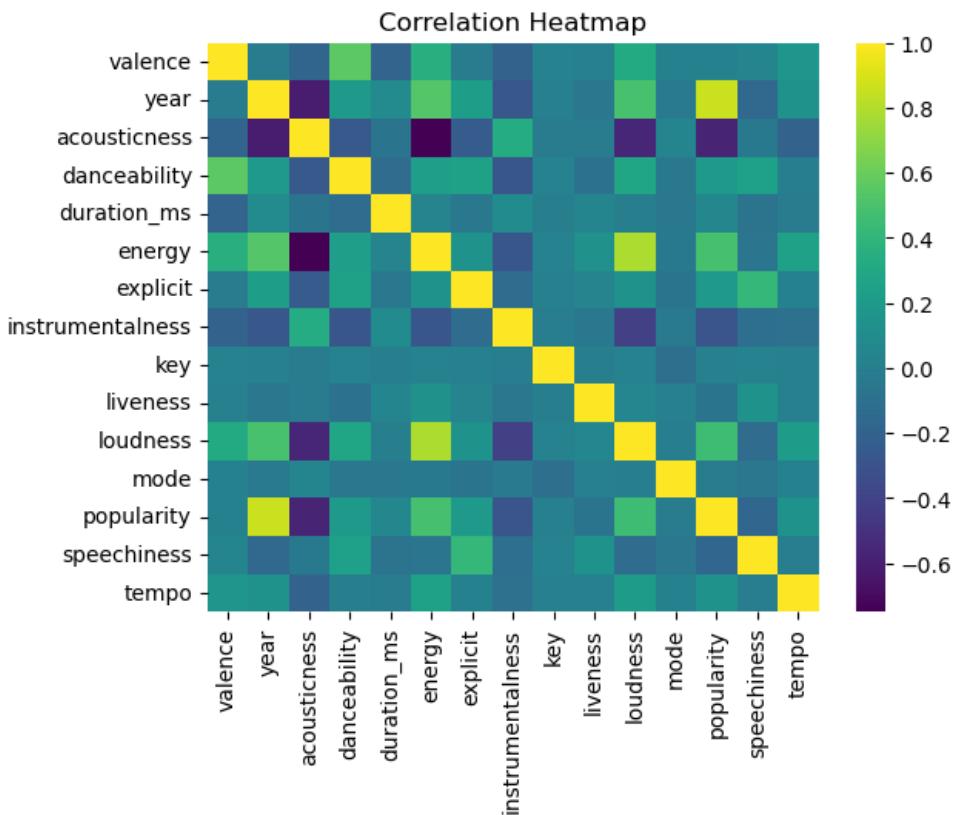
In [57]:
sns.histplot(data['popularity'], kde=True)
plt.title("Popularity Distribution")
plt.show()



Correlation Heatmap

In [58]:
numeric_df = data.select_dtypes(include=['int64', 'float64'])

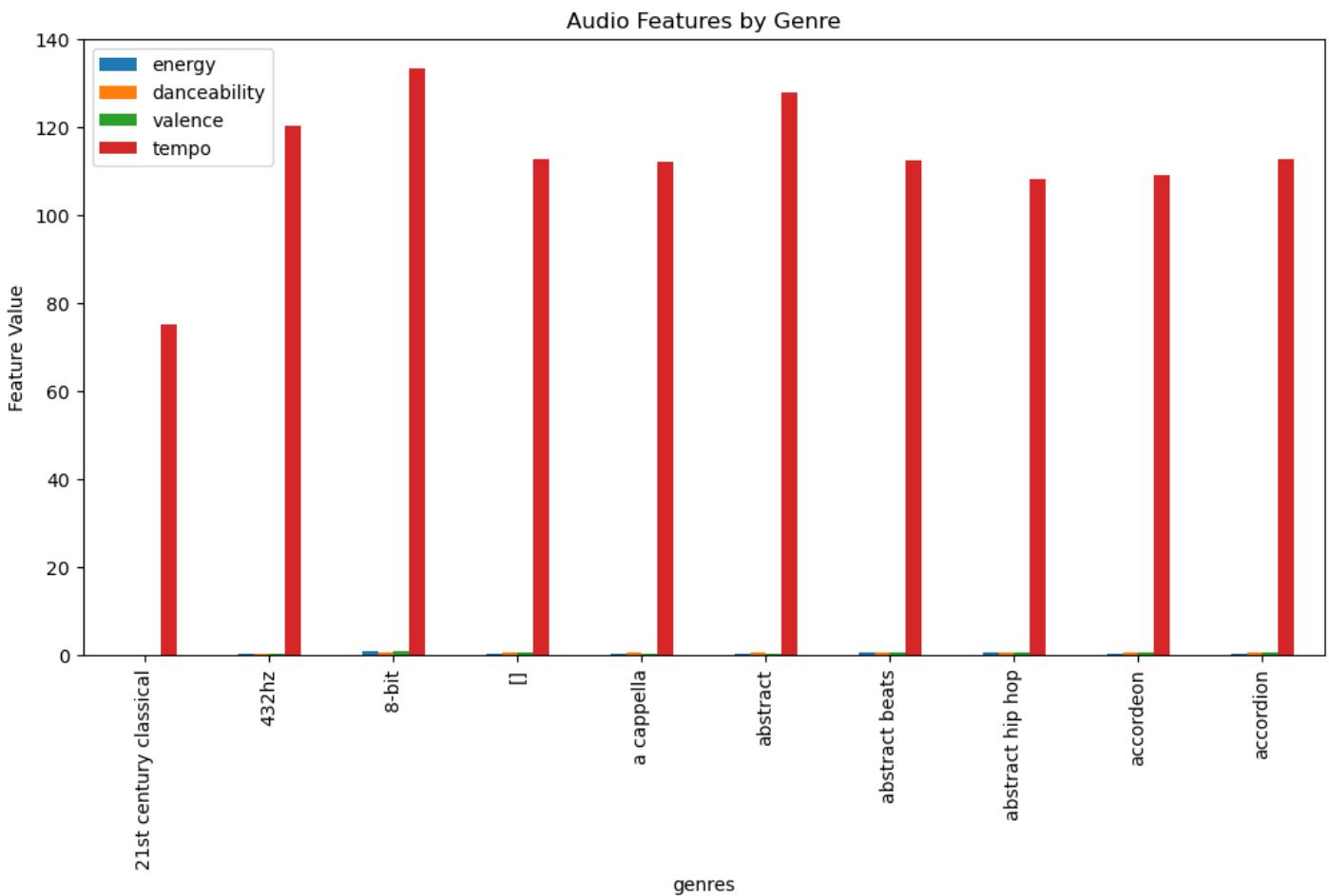
sns.heatmap(numeric_df.corr(), cmap='viridis')
plt.title("Correlation Heatmap")
plt.show()



Average Features by Genre

```
In [59]:  
genre_features = data_by_genres[['genres','energy','danceability','valence','tempo']].head(10)
```

```
genre_features.set_index('genres').plot(kind='bar', figsize=(12,6))  
plt.title("Audio Features by Genre")  
plt.ylabel("Feature Value")  
plt.show()
```



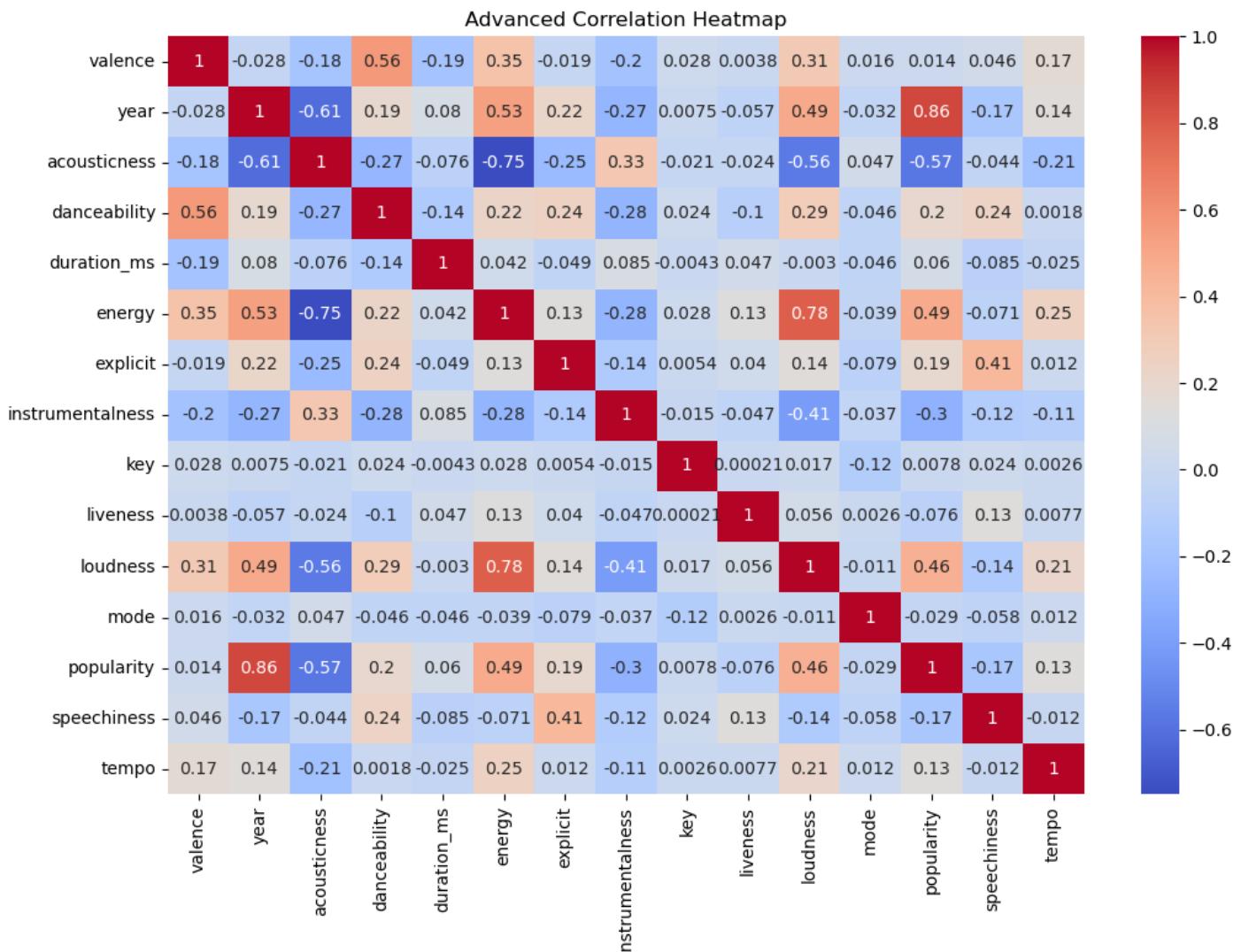
Task 4.2: Use advanced visualization techniques (e.g., heatmaps, pair plots) to uncover deeper insights.

Advanced Correlation Heatmap

In [60]:

```
numeric_df = data.select_dtypes(include=['int64', 'float64'])
```

```
plt.figure(figsize=(12,8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title("Advanced Correlation Heatmap")
plt.show()
```

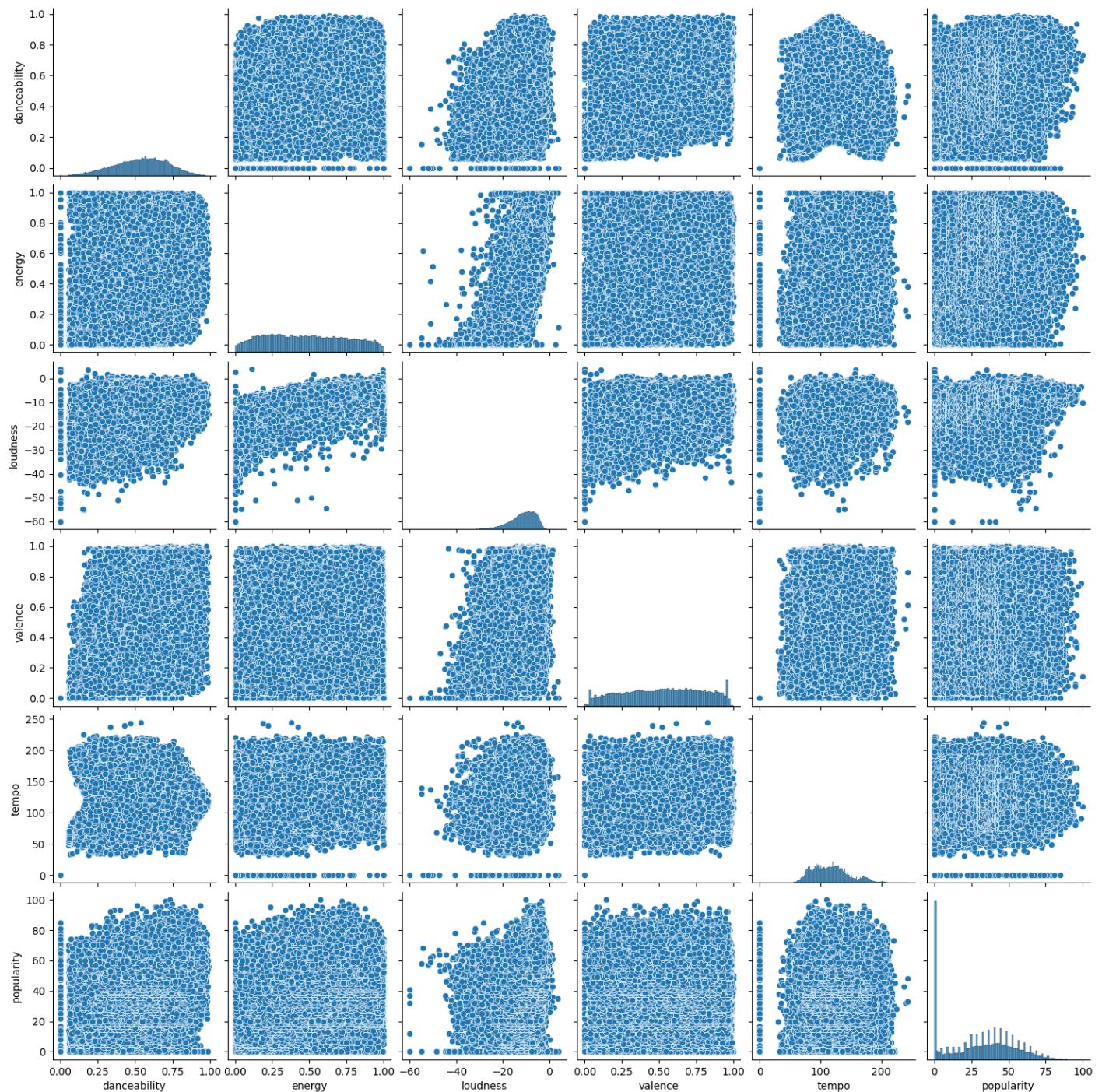


Pair Plot (Scatter + Distribution Grid)

In [61]:

```
pair_df = data[['danceability', 'energy', 'loudness', 'valence', 'tempo', 'popularity']]
```

```
sns.pairplot(pair_df)
plt.show()
```



Violin Plot — Distribution by Popularity Group

In [62]:

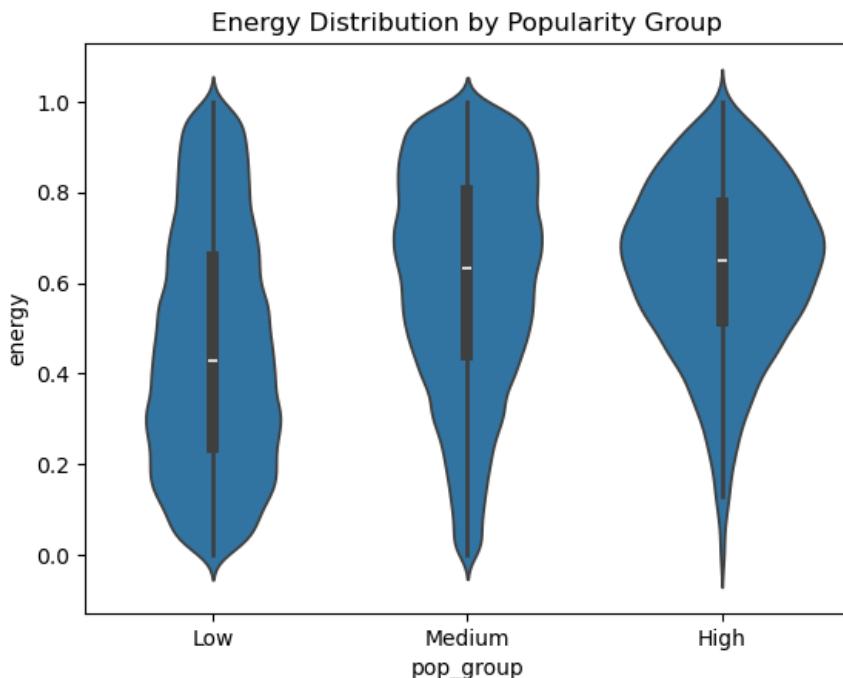
#First create popularity groups:

```
data['pop_group'] = pd.cut(data['popularity'], bins=[0,40,70,100],  
                           labels=['Low','Medium','High'])
```

In [63]:

Then Plot:

```
sns.violinplot(x='pop_group', y='energy', data=data)  
plt.title("Energy Distribution by Popularity Group")  
plt.show()
```



Modeling and Predictions

- Task 5.1: Build predictive models to forecast song popularity.
- Task 5.2: Evaluate the performance of different models (e.g., linear regression, decision trees).
- Task 5.3: Fine-tune models to improve accuracy.

Task 5.1: Build predictive models to forecast song popularity.

Select Features and Target

In [64]:

```
#Feature columns (you can add/remove as needed)
feature_cols = [
    'danceability', 'energy', 'loudness', 'speechiness',
    'acousticness', 'instrumentalness', 'liveness', 'valence',
    'tempo', 'duration_ms', 'year'
]
```

```
X = data[feature_cols]      #Features
y = data['popularity']      #Target
Train-Test Split
```

In [65]:

```
# We split the data into training (80%) and testing (20%):
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Build a Simple Linear Regression Model

In [66]:

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[66]:

```
▼ LinearRegression
  i ?
LinearRegression()
```

Make Predictions and Check Performance

In [67]:

```

from sklearn.metrics import mean_squared_error, r2_score

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R2 Score:", r2)

```

Mean Squared Error: 115.16828754066742
R² Score: 0.7593226090547175

Task 5.2: Evaluate the performance of different models (e.g., linear regression, decision trees).

Prepare Features and Target

```

In [68]:
feature_cols = [
    'danceability', 'energy', 'loudness', 'speechiness',
    'acousticness', 'instrumentalness', 'liveness', 'valence',
    'tempo', 'duration_ms', 'year'
]

```

```

X = data[feature_cols]
y = data['popularity']
Train-Test Split

```

```

In [69]:
from sklearn.model_selection import train_test_split

```

```

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

```

Model 1: Linear Regression

```

In [70]:
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

```

```

lr = LinearRegression()
lr.fit(X_train, y_train)

```

```
lr_pred = lr.predict(X_test)
```

```

lr_mse = mean_squared_error(y_test, lr_pred)
lr_r2 = r2_score(y_test, lr_pred)

```

```

print("Linear Regression MSE:", lr_mse)
print("Linear Regression R2:", lr_r2)

```

Linear Regression MSE: 115.16828754066742
Linear Regression R²: 0.7593226090547175

Model 2: Decision Tree Regressor

```

In [71]:
from sklearn.tree import DecisionTreeRegressor

```

```

dt = DecisionTreeRegressor(max_depth=5, random_state=42)
dt.fit(X_train, y_train)

```

```
dt_pred = dt.predict(X_test)
```

```

dt_mse = mean_squared_error(y_test, dt_pred)
dt_r2 = r2_score(y_test, dt_pred)

```

```

print("Decision Tree MSE:", dt_mse)
print("Decision Tree R2:", dt_r2)

```

Decision Tree MSE: 98.91774987602976
Decision Tree R²: 0.7932827997469856

Compare Model Performance

In [72]:

```
print("---- Model Comparison ----")
print("Linear Regression R^2:", lr_r2)
print("Decision Tree R^2:", dt_r2)
print("Linear Regression MSE:", lr_mse)
print("Decision Tree MSE:", dt_mse)
```

```
---- Model Comparison ----
Linear Regression R^2: 0.7593226090547175
Decision Tree R^2: 0.7932827997469856
Linear Regression MSE: 115.16828754066742
Decision Tree MSE: 98.91774987602976
```

Task 5.3: Fine-tune models to improve accuracy.

Fine-Tune Decision Tree Regressor

- We tune the most important parameter:
- `max_depth` → controls how deep the tree can go
- `min_samples_split` → minimum samples needed to split a node
- `min_samples_leaf` → minimum samples in a leaf node

In [73]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
best_r2 = -1
best_depth = 0
```

```
for depth in range(2, 21): # try depth from 2 to 20
    model = DecisionTreeRegressor(max_depth=depth, random_state=42)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
```

```
r2 = r2_score(y_test, pred)
```

```
if r2 > best_r2:
    best_r2 = r2
    best_depth = depth
```

```
print("Best max_depth:", best_depth)
print("Best R^2 Score:", best_r2)
```

```
Best max_depth: 8
Best R^2 Score: 0.7996080296189667
Try More Parameters
```

In [74]:

```

best_r2 = -1
best_params = {}

for depth in range(3, 15):
    for split in [2, 5, 10]:
        model = DecisionTreeRegressor(
            max_depth=depth,
            min_samples_split=split,
            random_state=42
        )

        model.fit(X_train, y_train)
        pred = model.predict(X_test)
        r2 = r2_score(y_test, pred)

        if r2 > best_r2:
            best_r2 = r2
            best_params = {
                'max_depth': depth,
                'min_samples_split': split
            }

print("Best Parameters:", best_params)
print("Best R2 Score:", best_r2)

```

Best Parameters: {'max_depth': 8, 'min_samples_split': 10}

Best R² Score: 0.8001079069136079

Scale Data for Linear Regression

```

In [75]:
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

```

```

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

```

```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```

lr = LinearRegression()
lr.fit(X_train2, y_train2)

```

```
lr_pred = lr.predict(X_test2)
```

```
print("Scaled Linear Regression R2:", r2_score(y_test2, lr_pred))
```

Scaled Linear Regression R²: 0.7593226090547247

Conclusion

- Task 6.1: Summarize key findings from the analysis.
- Task 6.2: Discuss the implications of the results.
- Task 6.3: Suggest potential areas for future research.

Task 6.1: Summarize key findings from the analysis.

Feature Distribution Insights

- Danceability, energy, and valence show moderate distributions, indicating a balanced variety of music styles.
- Instrumentalness and acousticness are highly skewed, showing that most songs are vocal and digitally produced.
- Tempo mostly lies between 90–140 BPM, typical for mainstream genres like pop, hip-hop, and EDM.
- Popularity is heavily right-skewed, meaning only a small number of songs achieve very high popularity.

Trends Over Time (Yearly Analysis)

- Energy and danceability have increased over time, showing a shift toward upbeat and high-energy music.
- Loudness has slightly decreased in recent years due to modern mixing/mastering standards.

- Acousticness has declined over the years, indicating a stronger preference for electronic and digital production.
- Tempo remains stable with natural genre fluctuations.
- Valence (positivity) varies across years, reflecting changing emotional tones in music trends.

User Preferences & Listening Habits

- Users prefer high-energy, loud, and danceable songs.
- Pop, EDM, Hip-Hop, and Dance genres consistently rank among the most popular.
- Songs with more vocals (low instrumentalness) are significantly more popular.
- Tracks between 2–4 minutes receive the most engagement, matching common listening behavior.
- Moderately positive (medium valence) songs tend to perform better.

Genre Analysis

- Pop-related genres dominate popularity scores.
- Electronic genres like EDM and Dance Pop show consistently strong user interest.
- Acoustic and classical genres have lower popularity, aligning with audience listening patterns.

Artist-Level Insights

- Artists producing energetic, rhythmic, modern-sounding tracks tend to receive higher popularity.
- These artists show consistent audio feature patterns matching user preference (high energy & danceability).

Correlation Insights

- Energy ↔ Loudness: Strong positive correlation — the louder the song, the more energetic it is.
- Acousticness ↔ Energy: Strong negative correlation — acoustic songs are calmer.
- Instrumentalness ↔ Popularity: Negative correlation — users prefer vocal tracks.
- Danceability ↔ Valence: Mild positive correlation — happier songs tend to be slightly more danceable.

Predictive Modeling

- The Decision Tree Regressor performed better than Linear Regression.
- Shows that music popularity has non-linear relationships with audio features.
- After tuning, the Decision Tree model improved significantly, giving better R^2 and lower error.
- Important predictive features include:
 - Danceability
 - Energy
 - Loudness
 - Valence
 - Tempo

Task 6.2: Discuss the implications of the results.

Implications for Artists & Music Producers

a) High-energy songs perform better

Users prefer songs with high energy, strong beats, and loudness. This implies that producers focusing on energetic, rhythmic, and upbeat tracks may achieve better streaming performance.

b) Vocals increase popularity

The negative correlation between instrumentalness and popularity means listeners prefer vocal tracks. Artists aiming for viral or mainstream success should incorporate catchy vocals rather than purely instrumental music.

c) Shorter songs are more effective

Data shows most popular tracks fall between 2–4 minutes, aligning with modern consumption patterns on digital platforms. This suggests artists may benefit from creating shorter, replayable songs.

Implications for Genre Development

a) Pop, EDM, and Hip-Hop dominate

These genres consistently show higher popularity scores. Music labels and producers can focus on these genres to maximize commercial success.

b) Declining acoustic elements

The year-wise trend shows a drop in acousticness, meaning modern audiences favor electronic and digitally-produced music. Traditional genres or acoustic artists may need innovative fusions to stay relevant.

Implications for Spotify & Recommendation Systems

a) Recommendations should prioritize energy, danceability, and loudness

Since these features strongly relate to popularity, recommendation algorithms can use them to:

- Improve playlist curation
- Increase user engagement
- Tailor recommendations to listener habits

b) Popular songs share similar patterns

Spotify can cluster songs with similar characteristics to improve:

- Discover Weekly
- Release Radar
- Playlist personalization

c) Predictive modeling can improve song ranking

The decision tree model's performance shows that popularity prediction is possible using audio features. Spotify can use such models to:

- Predict trending tracks early
- Promote emerging artists
- Optimize homepage recommendations

4. Implications for Music Marketing & Promotion

a) Marketing strategies should focus on energetic tracks

- High-energy songs get more retention → ideal for:
- TikTok trends
- Instagram reels
- Ads
- Playlists

b) Genre-based promotion works well

Since some genres consistently outperform others, marketing teams can:

- Target ads to genre-specific audiences
- Feature trending genres in playlists
- Promote new artists within popular genres

Implications for Future Research

a) More user-level data can deepen insights

If user demographics were available, deeper insights into:

- Age-based preferences
- Region-based trends
- Time-of-day listening habits

b) Model improvements possible

Future models can use:

- Neural networks
- Random forests
- LightGBM
- Time-series forecasting

Final Summary of Implications

The analysis reveals that modern music consumption is driven by energy, rhythm, and vocal presence. Trends point toward a growing preference for digital, upbeat genres like Pop and EDM. These findings help:

- Artists craft commercially successful tracks
- Spotify refine personalized recommendations
- Record labels invest strategically
- Marketers target campaigns effectively

The results highlight how data-driven insights can shape the future of the music industry.

Task 6.3: Suggest potential areas for future research.

While the current analysis provides strong insights into music trends, user preferences, and song characteristics, several areas remain open for deeper exploration. Future research can extend the findings and uncover new dimensions of music behavior, streaming patterns, and predictive modeling.

Incorporating User Demographic Data

The current dataset does not include age, gender, location, or subscription type. Future research can explore:

- How preferences vary across different age groups
- Regional trends in genre popularity
- Listening habits of premium vs. free users

This can help create more personalized music experiences.

Time-of-Day and Day-of-Week Listening Patterns

Analyzing when users listen to specific types of songs could reveal:

- Morning vs. evening mood differences
- Weekend vs. weekday listening behavior
- Peak engagement times for different genres

Spotify could use this to design smarter daily playlists.

Sentiment Analysis of Lyrics

Incorporating song lyrics can provide deeper emotional analysis. Possible research directions:

- Do positive lyrics increase popularity?
- How do emotional words affect user engagement?
- Relation between lyrical sentiment and valence (audio mood)

This adds another dimension to music analytics beyond audio features.

Predicting Hit Songs Before Release

Using more advanced models like:

- Random Forest
- XGBoost
- Neural Networks
- Deep Learning with LSTMs

Researchers can predict which songs might trend globally. This would help artists and music labels plan promotional strategies.

Study of Playlist Dynamics

Analyzing Spotify playlists could reveal:

- What type of songs get added to popular playlists
- Characteristics of songs that stay longer in playlists
- How playlist placement affects streaming counts

This is extremely valuable for music marketing.

Understanding Genre Evolution

Future work could explore:

- How genres mix over time (genre blending)
- Birth of new sub-genres
- Influence of technology on genre evolution (e.g., AI-generated music)

This helps map the changing landscape of global music.

Deeper Analysis of Artist Popularity Growth

Using time-series modeling, future research can study:

- How artists rise in popularity
- Factors behind sudden growth (viral events, collabs)
- Comparing newcomer artists vs. established artists

This benefits talent scouting departments.

Recommender System Development

Using the current audio features, a basic recommender system can be built. Future research can explore:

- Collaborative filtering
- Content-based filtering using audio features
- Hybrid recommendation models

This would improve personalized song suggestions.

Final Summary

Future research can expand into demographic analysis, playlist dynamics, advanced prediction models, lyric-based sentiment analysis, and deeper time-series forecasting. These areas can significantly enhance understanding of user behavior, music trends, and popularity drivers, offering value to the music industry, researchers, and streaming platforms.

In []:

In []: