Neural_Beats_harmonised_(2) (1)

March 30, 2025

1 NeuralBeats: A Deep Learning Approach to Music Discovery

2 1. Business Understanding

2.1 Problem Statement

Music streaming platforms struggle to keep users engaged and help them discover new artists they might like. Traditional recommendation systems often focus on individual songs, but users may also want recommendations for similar artists and their music.

2.2 Stakeholders

1. Users (Listeners) - Benefit: More relevant and engaging music recommendations based on their mood, activities, and listening behavior. - Impact: Increased user satisfaction, retention, and engagement, leading to a better experience and less frustration with repetitive suggestions.

2. Music Artists & Creators - Benefit: Better discovery and fairer exposure, allowing independent artists to reach new listeners beyond mainstream algorithms. - Impact: Helps emerging artists break into the industry and increases overall content diversity.

3. Business & Marketing Teams - Benefit: A more engaging and personalized platform means higher user retention and increased revenue. - Impact: Strengthens Neural Beats' market position, making it a strong competitor to Spotify and Apple Music.

2.3 Key Business Questions

1. User Experience & Personalization:

- How can Neural Beats improve music discovery and reduce repetitive recommendations?
- How can we recommend songs based on mood, context, and listening behavior rather than just genre?
- What features (e.g., valence, tempo, energy) are most effective for predicting user preferences?

2. Engagement & Retention:

- How can Neural Beats balance personalized recommendations with new music exploration?
- What impact does context-aware music suggestions have on user session time and engagement?
- How can we reduce churn rates and encourage long-term platform usage?

3. Technology & AI Strategy:

- Which ML/DL models are best suited for personalized recommendations?
- How can Neural Beats use real-time data to dynamically adjust recommendations?
- How can we ensure recommendations remain fair, unbiased, and diverse?

4. Business & Revenue Growth:

- How do better recommendations impact subscription conversion rates?
- Can personalized recommendations increase ad revenue and premium sign-ups?
- What role does AI-driven discovery play in boosting music streaming consumption?

2.4 Objective

The objective of this project is divided into three key areas:

1. *Develop an Intelligent Artist Recommendation System*
    - Build a deep learning-based system that identifies and suggests similar artists based on *audio features, genre, and popularity.*

    - Utilize autoencoders to capture artist similarities effectively.
2. *Enhance User Engagement & Music Discovery*
    - Improve user experience by introducing them to new artists that align with their listening preferences.

    - Encourage exploration beyond mainstream artists by presenting diverse recommendations.
3. *Provide a Scalable and Efficient Recommendation Model*
    - Ensure the system is computationally efficient and scalable to handle large music databases.

    - Integrate the recommendation system into music streaming platforms via an API or user interface.

2.4.1 How It Works

1. User selects an artist.
2. Model identifies similar artists using *autoencoders.*
3. System recommends similar artists & their songs.
4. Users discover new music → *Improved engagement and experience.*

3 2. Data Understanding

To develop a next-generation AI-powered music recommendation system, Neural Beats relies on a well-structured dataset that captures track metadata, audio features, and popularity metrics. This section details the data source and relevance to ensure alignment with business objectives.

3.1 2.1 Data source

The dataset used for this project is the **Spotify Tracks Dataset**, sourced from **Kaggle**. It contains detailed information about Spotify tracks across 125 genres, along with associated audio features and popularity metrics.

Dataset Format: CSV (Tabular)

Key Features Include:

1.Track Metadata: Song title, album, artist(s), genre, duration.

2.Audio Features: Danceability, energy, loudness, tempo, and valence.

3.User Engagement: Popularity score based on plays and recency

Feature Description

- **track_id:**
- The Spotify ID for the track
- **artists:**
- The artists' names who performed the track. If there is more than one artist, they are separated by a ;
- **album_name:**
- The album name in which the track appears
- **track_name:**
- Name of the track
- **popularity:**
- The popularity of a track is a value between 0 and 100, with 100 being the most popular.
- The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are.
- Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past.
- Duplicate tracks (e.g. the same track from a single and an album) are rated independently.
- Artist and album popularity is derived mathematically from track popularity.
- **duration_ms:**
- The track length in milliseconds
- **explicit:**
- Whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown)
- **danceability:**
- Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.
- A value of 0.0 is least danceable and 1.0 is most danceable
- **energy:**

- Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.
- Typically, energetic tracks feel fast, loud, and noisy.
- For example, death metal has high energy, while a Bach prelude scores low on the scale
- **key:**
- The key the track is in.
- Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C /D , 2 = D, and so on.
- If no key was detected, the value is -1
- **loudness:**
- The overall loudness of a track in decibels (dB)
- A positive dB value indicates a signal is louder than the reference level.
- A negative dB value indicates a signal is quieter than the reference level.
- **mode:**
- mode feature refers to the modality of the track, which indicates whether the track is in a major or minor key.
- Major mode (1) typically corresponds to more "happy," "bright," or "cheerful" sounds.
- Minor mode (0) typically corresponds to more "sad," "dark," or "serious" sounds.
- **speechiness:**
- Speechiness detects the presence of spoken words in a track.
- The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value.
- Values above 0.66 describe tracks that are probably made entirely of spoken words.
- Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music.
- Values below 0.33 most likely represent music and other non-speech-like tracks
- **acousticness:**
- A confidence measure from 0.0 to 1.0 of whether the track is acoustic.
- 1.0 represents high confidence the track is acoustic
- **instrumentalness:**
- Predicts whether a track contains no vocals.
- "Ooh" and "aah" sounds are treated as instrumental in this context.
- Rap or spoken word tracks are clearly "vocal".

- The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content
- **liveness:**
- Detects the presence of an audience in the recording.
- Higher liveness values represent an increased probability that the track was performed live.
- A value above 0.8 provides strong likelihood that the track is live
- **valence:**
- A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
- Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry)
- **tempo:**
- The overall estimated tempo of a track in beats per minute (BPM).
- In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration
- **time_signature:**
- An estimated time signature.
- The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
- The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4.
- **track_genre:**
- The genre in which the track belongs

The dataset provides a structured foundation for building personalized, mood-based music recommendations.

3.2 2.2 Why This Data is Useful

- The Spotify Tracks Dataset is ideal for solving Neural Beats' business problem because it includes rich audio and user interaction features that allow us to:

Build Personalized Recommendation Models - Utilize audio features (e.g., danceability, energy, valence) to match user preferences. - Move beyond simple genre-based recommendations by considering song mood and context.

Improve Music Discovery & Engagement - Predict emerging trends by analyzing song popularity over time. - Recommend undiscovered tracks based on listening behavior and similar song attributes.

Develop Context-Aware AI Models - Suggest music based on user activity (e.g., workout, relaxation, focus). - Use tempo, acousticness, and valence to enhance mood-based recommendations.

3.3 2.3 Dataset Overview

[1]:
```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

from scipy.stats import skew

from mpl_toolkits.mplot3d import Axes3D
from scipy.stats.mstats import winsorize


from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler
```

[2]:
```
data=pd.read_csv('dataset.csv')
```

[3]:
```
#view all columns
pd.set_option('display.max_columns',30)
#view the dataset
data.head(10)
```

[3]:
```
   Unnamed: 0                track_id                          artists  \
0           0  5SuOikwiRyPMVoIQDJUgSV                      Gen Hoshino
1           1  4qPNDBW1i3p13qLCt0Ki3A                     Ben Woodward
2           2  1iJBSr7s7jYXzM8EGcbK5b           Ingrid Michaelson;ZAYN
3           3  6lfxq3CG4xtTiEg7opyCyx                     Kina Grannis
4           4  5vjLSffimiIP26QG5WcN2K                 Chord Overstreet
5           5  01MVOl9KtVTNfFiBU9I7dc                     Tyrone Wells
6           6  6Vc5wAMmXdKIAM7WUoEb7N  A Great Big World;Christina Aguilera
7           7  1EzrEOXmMH3G43AXT1y7pA                       Jason Mraz
8           8  0IktbUcnAGrvD03AWnz3Q8           Jason Mraz;Colbie Caillat
9           9  7k9GuJYLp2AzqokyEdwEw2                   Ross Copperman

                                          album_name  \
0                                             Comedy
1                                   Ghost (Acoustic)
2                                     To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                            Hold On
5                              Days I Will Remember
6                         Is There Anybody Out There?
7                  We Sing. We Dance. We Steal Things.
8                  We Sing. We Dance. We Steal Things.
```

```
9                                              Hunger

                  track_name  popularity  duration_ms  explicit  \
0                     Comedy          73       230666     False
1           Ghost - Acoustic          55       149610     False
2             To Begin Again          57       210826     False
3  Can't Help Falling In Love          71       201933     False
4                    Hold On          82       198853     False
5        Days I Will Remember          58       214240     False
6               Say Something          74       229400     False
7                   I'm Yours          80       242946     False
8                       Lucky          74       189613     False
9                      Hunger          56       205594     False

   danceability  energy  key  loudness  mode  speechiness  acousticness  \
0         0.676  0.4610    1    -6.746     0       0.1430        0.0322
1         0.420  0.1660    1   -17.235     1       0.0763        0.9240
2         0.438  0.3590    0    -9.734     1       0.0557        0.2100
3         0.266  0.0596    0   -18.515     1       0.0363        0.9050
4         0.618  0.4430    2    -9.681     1       0.0526        0.4690
5         0.688  0.4810    6    -8.807     1       0.1050        0.2890
6         0.407  0.1470    2    -8.822     1       0.0355        0.8570
7         0.703  0.4440   11    -9.331     1       0.0417        0.5590
8         0.625  0.4140    0    -8.700     1       0.0369        0.2940
9         0.442  0.6320    1    -6.770     1       0.0295        0.4260

   instrumentalness  liveness  valence    tempo  time_signature track_genre
0          0.000001    0.3580   0.7150   87.917               4    acoustic
1          0.000006    0.1010   0.2670   77.489               4    acoustic
2          0.000000    0.1170   0.1200   76.332               4    acoustic
3          0.000071    0.1320   0.1430  181.740               3    acoustic
4          0.000000    0.0829   0.1670  119.949               4    acoustic
5          0.000000    0.1890   0.6660   98.017               4    acoustic
6          0.000003    0.0913   0.0765  141.284               3    acoustic
7          0.000000    0.0973   0.7120  150.960               4    acoustic
8          0.000000    0.1510   0.6690  130.088               4    acoustic
9          0.004190    0.0735   0.1960   78.899               4    acoustic
```

[4]:
```
data.tail()
```

[4]:
```
        Unnamed: 0                track_id           artists  \
113995      113995  2C3TZjDRiAzdyViavDJ217     Rainy Lullaby
113996      113996  1hIz5L4IB9hN3WRYPOCGPw     Rainy Lullaby
113997      113997  6x8ZfSoqDjuNa5SVP5QjvX     Cesária Evora
113998      113998  2e6sXL2bYv4bSz6VTdnfLs  Michael W. Smith
113999      113999  2hETkH7cOfqmz3LqZDHZf5     Cesária Evora
```

```
                                                  album_name  \
113995  #mindfulness - Soft Rain for Mindful Meditatio…
113996  #mindfulness - Soft Rain for Mindful Meditatio…
113997                                           Best Of
113998                                 Change Your World
113999                                     Miss Perfumado

                 track_name  popularity  duration_ms  explicit  danceability  \
113995  Sleep My Little Boy          21       384999     False         0.172
113996     Water Into Light          22       385000     False         0.174
113997       Miss Perfumado          22       271466     False         0.629
113998              Friends          41       283893     False         0.587
113999            Barbincor          22       241826     False         0.526

        energy  key  loudness  mode  speechiness  acousticness  \
113995   0.235    5   -16.393     1       0.0422         0.640
113996   0.117    0   -18.318     0       0.0401         0.994
113997   0.329    0   -10.895     0       0.0420         0.867
113998   0.506    7   -10.889     1       0.0297         0.381
113999   0.487    1   -10.204     0       0.0725         0.681

        instrumentalness  liveness  valence    tempo  time_signature  \
113995             0.928    0.0863   0.0339  125.995               5
113996             0.976    0.1050   0.0350   85.239               4
113997             0.000    0.0839   0.7430  132.378               4
113998             0.000    0.2700   0.4130  135.960               4
113999             0.000    0.0893   0.7080   79.198               4

        track_genre
113995  world-music
113996  world-music
113997  world-music
113998  world-music
113999  world-music
```

[5]:
```python
# Checking columns
data.columns
```

[5]:
```
Index(['Unnamed: 0', 'track_id', 'artists', 'album_name', 'track_name',
       'popularity', 'duration_ms', 'explicit', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature',
       'track_genre'],
      dtype='object')
```

[6]:
```python
#get the shape of the dataset
data.shape
```

```python
print(f"Our dataset has {data.shape[0]} rows and {data.shape[1]} columns")
```

```
Our dataset has 114000 rows and 21 columns
```

[7]:
```python
#checking for null values
data.isna().sum()
```

[7]:
```
Unnamed: 0          0
track_id            0
artists             1
album_name          1
track_name          1
popularity          0
duration_ms         0
explicit            0
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
time_signature      0
track_genre         0
dtype: int64
```

[8]:
```python
#checking for duplicates
data.duplicated().sum()
```

[8]:
```
0
```

[9]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Unnamed: 0        114000 non-null  int64
 1   track_id          114000 non-null  object
 2   artists           113999 non-null  object
 3   album_name        113999 non-null  object
 4   track_name        113999 non-null  object
 5   popularity        114000 non-null  int64
 6   duration_ms       114000 non-null  int64
```

```
 7   explicit          114000 non-null  bool
 8   danceability      114000 non-null  float64
 9   energy            114000 non-null  float64
 10  key               114000 non-null  int64
 11  loudness          114000 non-null  float64
 12  mode              114000 non-null  int64
 13  speechiness       114000 non-null  float64
 14  acousticness      114000 non-null  float64
 15  instrumentalness  114000 non-null  float64
 16  liveness          114000 non-null  float64
 17  valence           114000 non-null  float64
 18  tempo             114000 non-null  float64
 19  time_signature    114000 non-null  int64
 20  track_genre       114000 non-null  object
dtypes: bool(1), float64(9), int64(6), object(5)
memory usage: 17.5+ MB
```

[10]:
```
#checking datatypes
data.dtypes
```

[10]:
```
Unnamed: 0            int64
track_id             object
artists              object
album_name           object
track_name           object
popularity            int64
duration_ms           int64
explicit               bool
danceability        float64
energy              float64
key                   int64
loudness            float64
mode                  int64
speechiness         float64
acousticness        float64
instrumentalness    float64
liveness            float64
valence             float64
tempo               float64
time_signature        int64
track_genre          object
dtype: object
```

4 3. Data Preparation

This includes: * creating a copy of our original dataset so that whatever we do won't affect the original dataset * create a data profiling report to get the general overview of our data * understand the data: * explore the data * manipulating column names for better readability * dropping unnecessary columns * identify missing values using df.isnull().sum() then fill the missing values appropriately if any,or drop them * identify duplicates(df.duplicated() and remove them using df.drop_duplicated * check the data types if they are appropriate for each column if not correct them * standardize columns * check and handle outliers appropriately * create new features * do final checks then save the cleaned data

4.0.1 Create a copy of the dataset

We created a copy of our original dataset so that whatever we do won't affect the original dataset

[11]:
```
#copy the dataset
df=data.copy(deep=True)
```

4.0.2 Create a Data Profiling report

We create a data profiling report to get the general overview of our data

[12]:
```
!pip install ydata-profiling
```

```
Requirement already satisfied: ydata-profiling in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (4.15.1)
Requirement already satisfied: scipy<1.16,>=1.4.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (1.13.1)
Requirement already satisfied: pandas!=1.4.0,<3.0,>1.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (2.2.2)
Requirement already satisfied: matplotlib<=3.10,>=3.5 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (3.8.4)
Requirement already satisfied: pydantic>=2 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (2.10.6)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (6.0.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (3.1.4)
Requirement already satisfied: visions<0.8.2,>=0.7.5 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from
visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (0.8.1)
Requirement already satisfied: numpy<2.2,>=1.16.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (1.26.4)
Requirement already satisfied: htmlmin==0.1.12 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (0.1.12)
Requirement already satisfied: phik<0.13,>=0.11.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (0.12.4)
Requirement already satisfied: requests<3,>=2.24.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (2.32.2)
Requirement already satisfied: tqdm<5,>=4.48.2 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (4.66.4)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (0.13.2)
```

```
Requirement already satisfied: multimethod<2,>=1.4 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (1.12)
Requirement already satisfied: statsmodels<1,>=0.13.2 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (0.14.2)
Requirement already satisfied: typeguard<5,>=3 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (4.4.2)
Requirement already satisfied: imagehash==4.3.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (4.3.1)
Requirement already satisfied: wordcloud>=1.9.3 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (1.9.4)
Requirement already satisfied: dacite>=1.8 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (1.9.2)
Requirement already satisfied: numba<=0.61,>=0.56.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from ydata-profiling) (0.59.1)
Requirement already satisfied: PyWavelets in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from imagehash==4.3.1->ydata-profiling)
(1.5.0)
Requirement already satisfied: pillow in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from imagehash==4.3.1->ydata-profiling)
(10.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from jinja2<3.2,>=2.11.1->ydata-
profiling) (2.1.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from matplotlib<=3.10,>=3.5->ydata-
profiling) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from matplotlib<=3.10,>=3.5->ydata-
profiling) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from matplotlib<=3.10,>=3.5->ydata-
profiling) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from matplotlib<=3.10,>=3.5->ydata-
profiling) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from matplotlib<=3.10,>=3.5->ydata-
profiling) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from matplotlib<=3.10,>=3.5->ydata-
profiling) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from matplotlib<=3.10,>=3.5->ydata-
profiling) (2.9.0.post0)
Requirement already satisfied: llvmlite<0.43,>=0.42.0dev0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from numba<=0.61,>=0.56.0->ydata-
profiling) (0.42.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\sumaiya
```

```
abdullahi\anaconda3\lib\site-packages (from pandas!=1.4.0,<3.0,>1.1->ydata-
profiling) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from pandas!=1.4.0,<3.0,>1.1->ydata-
profiling) (2023.3)
Requirement already satisfied: joblib>=0.14.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from phik<0.13,>=0.11.1->ydata-profiling)
(1.4.2)
Requirement already satisfied: annotated-types>=0.6.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from pydantic>=2->ydata-profiling)
(0.7.0)
Requirement already satisfied: pydantic-core==2.27.2 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from pydantic>=2->ydata-profiling)
(2.27.2)
Requirement already satisfied: typing-extensions>=4.12.2 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from pydantic>=2->ydata-profiling)
(4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from requests<3,>=2.24.0->ydata-
profiling) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from requests<3,>=2.24.0->ydata-
profiling) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from requests<3,>=2.24.0->ydata-
profiling) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from requests<3,>=2.24.0->ydata-
profiling) (2025.1.31)
Requirement already satisfied: patsy>=0.5.6 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from statsmodels<1,>=0.13.2->ydata-
profiling) (0.5.6)
Requirement already satisfied: colorama in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from tqdm<5,>=4.48.2->ydata-profiling)
(0.4.6)
Requirement already satisfied: attrs>=19.3.0 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from
visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
(23.1.0)
Requirement already satisfied: networkx>=2.4 in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from
visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
(3.2.1)
Requirement already satisfied: puremagic in c:\users\sumaiya
abdullahi\anaconda3\lib\site-packages (from
visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
(1.28)
Requirement already satisfied: six in c:\users\sumaiya
```

```
abdullahi\anaconda3\lib\site-packages (from
patsy>=0.5.6->statsmodels<1,>=0.13.2->ydata-profiling) (1.16.0)
```

[13]:
```
# Import libraries
from ydata_profiling import ProfileReport

# Produce and save the profiling report
profile = ProfileReport(df,title="Spotify Profile Report")
profile.to_file("report.html")
```

```
<IPython.core.display.HTML object>

Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]


  0%|
| 0/21 [00:00<?, ?it/s]
  5%|
| 1/21 [00:02<00:33,  1.68s/it]
 10%|
| 2/21 [00:03<00:33,  1.75s/it]
 24%|
| 5/21 [00:04<00:10,  1.51it/s]
100%|
    | 21/21 [00:04<00:00,  4.79it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

[14]:
```
#view the profiling report
profile
```

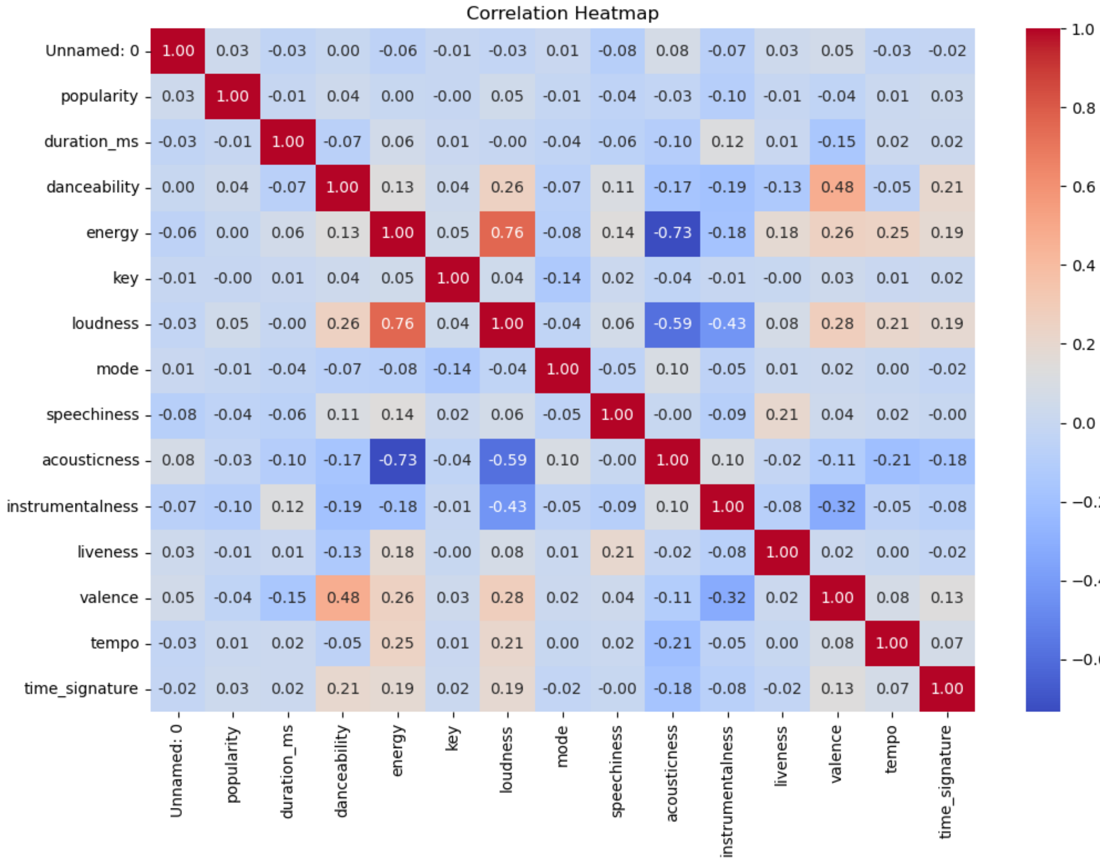
```
<IPython.core.display.HTML object>
```

[14]:

4.0.3 Correlation heatmap between numerical features

[15]:
```
# defining numerical and categorical columns
num_columns = df.select_dtypes(include=['number'])
categorical_cols=df[['explicit','mode']]
```

[16]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(num_columns.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show();
```



The color gradient represents the strength and direction of correlation:

Red → Strong positive correlation (+1)

Blue → Strong negative correlation (-1)

White/Neutral Colors → Weak or no correlation (0)

Key Interpretations:

1. energy and loudness (0.76) → Strong positive correlation

   Louder songs tend to have higher energy.

2. acousticness and energy (-0.73) → Strong negative correlation

   Acoustic songs tend to have lower energy levels.

3. instrumentalness and loudness (-0.43) → Moderate negative correlation

   More instrumental tracks tend to be quieter.

4. danceability and valence (0.48) → Moderate positive correlation

   Happier songs (high valence) are more danceable.

5. popularity and other features

Popularity has weak correlations (close to 0) with most features, meaning song attributes like energy, danceability, or duration do not strongly determine a song's popularity.

4.1 3.1 Data Cleaning

[17]:
```
#get statistical summary
df.describe().T
```

[17]:
```
                      count           mean            std     min  \
Unnamed: 0         114000.0   56999.500000   32909.109681   0.000
popularity         114000.0      33.238535      22.305078   0.000
duration_ms        114000.0  228029.153114  107297.712645   0.000
danceability       114000.0       0.566800       0.173542   0.000
energy             114000.0       0.641383       0.251529   0.000
key                114000.0       5.309140       3.559987   0.000
loudness           114000.0      -8.258960       5.029337 -49.531
mode               114000.0       0.637553       0.480709   0.000
speechiness        114000.0       0.084652       0.105732   0.000
acousticness       114000.0       0.314910       0.332523   0.000
instrumentalness   114000.0       0.156050       0.309555   0.000
liveness           114000.0       0.213553       0.190378   0.000
valence            114000.0       0.474068       0.259261   0.000
tempo              114000.0     122.147837      29.978197   0.000
time_signature     114000.0       3.904035       0.432621   0.000

                         25%            50%          75%          max
Unnamed: 0       28499.75000   56999.500000   85499.2500   113999.000
popularity          17.00000      35.000000      50.0000      100.000
duration_ms     174066.00000  212906.000000  261506.0000  5237295.000
danceability         0.45600       0.580000       0.6950        0.985
energy               0.47200       0.685000       0.8540        1.000
key                  2.00000       5.000000       8.0000       11.000
loudness           -10.01300      -7.004000      -5.0030        4.532
mode                 0.00000       1.000000       1.0000        1.000
speechiness          0.03590       0.048900       0.0845        0.965
acousticness         0.01690       0.169000       0.5980        0.996
instrumentalness     0.00000       0.000042       0.0490        1.000
liveness             0.09800       0.132000       0.2730        1.000
valence              0.26000       0.464000       0.6830        0.995
tempo               99.21875     122.017000     140.0710      243.372
time_signature       4.00000       4.000000       4.0000        5.000
```

[18]:
```
#get statistical summary(for categorical columns)
df.describe(include='O').T
```

[18]:
```
             count unique                          top  freq
track_id     114000  89741      6S3JlDAGk3uu3NtZbPnuhS     9
artists      113999  31437                 The Beatles   279
album_name   113999  46589   Alternative Christmas 2022   195
track_name   113999  73608               Run Rudolph Run   151
track_genre  114000    114                     acoustic  1000
```

4.1.1 3.1.1 Column Manipulation

We check:
* check column names to see if they are the same * change the column names to lowercase * rename column names to make them more undesrstandable
* remove whitespaces in the data and column names if any * drop unnecessary column names

[19]:
```python
#checking the column names
df.columns
```

[19]:
```
Index(['Unnamed: 0', 'track_id', 'artists', 'album_name', 'track_name',
       'popularity', 'duration_ms', 'explicit', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature',
       'track_genre'],
      dtype='object')
```

[20]:
```python
#strip white spaces in values
df=df.apply(lambda col:col.str.strip() if col.dtype in ["object", "number",␣
 ↪"category"] else col)
```

[21]:
```python
#drop the unnecessary columns
df=df.drop(columns=['Unnamed: 0','album_name'],errors='ignore')
#view  the dataset
df.head()
```

[21]:
```
                 track_id                 artists                track_name  \
0  5SuOikwiRyPMVoIQDJUgSV              Gen Hoshino                    Comedy
1  4qPNDBW1i3p13qLCt0Ki3A             Ben Woodward          Ghost - Acoustic
2  1iJBSr7s7jYXzM8EGcbK5b   Ingrid Michaelson;ZAYN            To Begin Again
3  6lfxq3CG4xtTiEg7opyCyx             Kina Grannis  Can't Help Falling In Love
4  5vjLSffimiIP26QG5WcN2K         Chord Overstreet                   Hold On

   popularity  duration_ms  explicit  danceability  energy  key  loudness  \
0          73       230666     False         0.676  0.4610    1    -6.746
1          55       149610     False         0.420  0.1660    1   -17.235
2          57       210826     False         0.438  0.3590    0    -9.734
3          71       201933     False         0.266  0.0596    0   -18.515
4          82       198853     False         0.618  0.4430    2    -9.681

   mode  speechiness  acousticness  instrumentalness  liveness  valence  \
```

```
0     0      0.1430        0.0322           0.000001     0.3580     0.715
1     1      0.0763        0.9240           0.000006     0.1010     0.267
2     1      0.0557        0.2100           0.000000     0.1170     0.120
3     1      0.0363        0.9050           0.000071     0.1320     0.143
4     1      0.0526        0.4690           0.000000     0.0829     0.167

     tempo  time_signature track_genre
0   87.917               4    acoustic
1   77.489               4    acoustic
2   76.332               4    acoustic
3  181.740               3    acoustic
4  119.949               4    acoustic
```

4.1.2 3.1.2 Handling missing values

We want to identify if there is any missing values in our dataset and if so deal with them appropriately

[22]:
```
#check for missing values
#df.isnull().sum()
mis = df.isna().any().sum()
if mis > 0:
    print(f'\nThere are {mis} missing values present in our data.')
else:
    print('There are no missing values in our data.')
mis
```

```
There are 2 missing values present in our data.
```

[22]:
```
2
```

[23]:
```
# drop the 1 missing value
df = df.dropna()
#check for missing values
#df.isnull().sum()
mis = df.isna().any().sum()
if mis > 0:
    print(f'\nThere are {mis} missing values present in our data.')
else:
    print('There are no missing values in our data.')
mis
```

```
There are no missing values in our data.
```

[23]:
```
0
```

4.1.3 3.1.3 Duplicates

First, we will try to identify if we have any duplicates, if any it'd be best to remove them

[24]:
```
#checking for duplicates
dup = df.duplicated().sum()
if dup > 0:
    print(f'\nThere are {dup} duplicates present in our data.')
else:
    print('There are no duplicates in our data.')

dup
```

```
There are 450 duplicates present in our data.
```

[24]: 450

[25]:
```
# Remove duplicates (inplace to modify the original DataFrame)
df = df.drop_duplicates()

# Check for duplicates
dup = df.duplicated().sum()
if dup > 0:
    print(f'\nThere are {dup} duplicates present in our data.')
else:
    print('There are no duplicates in our data.')

dup
```

```
There are no duplicates in our data.
```

[25]: 0

[26]:
```
# droping duplicates in track ID
# df = df.groupby("track_id").agg({
#     "track_name": "first",  # Keep the first track name
#     "artists": "first",  # Keep the first artist name
#     "track_genre": lambda x: ", ".join(set(x)),  # Merge unique genres
#     "popularity": "max",  # Keep the highest popularity score
#     "duration_ms": "mean",  # Average duration
#     "danceability": "mean",
#     "energy": "mean",
#     "key": "first",  # Keep the first key (or use mode if preferred)
#     "loudness": "mean",
#     "mode": "first",
#     "speechiness": "mean",
#     "acousticness": "mean",
#     "instrumentalness": "mean",
```

```
#     "liveness": "mean",
#     "valence": "mean",
#     "tempo": "mean",
#     "time_signature": "first"  # Usually doesn't change, so keeping first␣
↪occurrence
# }).reset_index()

df = df.drop_duplicates(subset=['track_id'], keep='first')
```

[27]:
```
df.head()
```

[27]:
```
                 track_id                 artists                 track_name  \
0  5SuOikwiRyPMVoIQDJUgSV              Gen Hoshino                     Comedy
1  4qPNDBW1i3p13qLCt0Ki3A             Ben Woodward           Ghost - Acoustic
2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN             To Begin Again
3  6lfxq3CG4xtTiEg7opyCyx             Kina Grannis  Can't Help Falling In Love
4  5vjLSffimiIP26QG5WcN2K         Chord Overstreet                    Hold On

   popularity  duration_ms  explicit  danceability  energy  key  loudness  \
0          73       230666     False         0.676  0.4610    1    -6.746
1          55       149610     False         0.420  0.1660    1   -17.235
2          57       210826     False         0.438  0.3590    0    -9.734
3          71       201933     False         0.266  0.0596    0   -18.515
4          82       198853     False         0.618  0.4430    2    -9.681

   mode  speechiness  acousticness  instrumentalness  liveness  valence  \
0     0       0.1430        0.0322          0.000001    0.3580    0.715
1     1       0.0763        0.9240          0.000006    0.1010    0.267
2     1       0.0557        0.2100          0.000000    0.1170    0.120
3     1       0.0363        0.9050          0.000071    0.1320    0.143
4     1       0.0526        0.4690          0.000000    0.0829    0.167

     tempo  time_signature track_genre
0   87.917               4    acoustic
1   77.489               4    acoustic
2   76.332               4    acoustic
3  181.740               3    acoustic
4  119.949               4    acoustic
```

[28]:
```
df.shape
```

[28]:
```
(89740, 19)
```

4.1.4 3.1.4 Data Types

We check the datatypes to see if they are correctly placed if not change them

[29]:
```
#check data types
df.dtypes
```

[29]:
```
track_id              object
artists               object
track_name            object
popularity             int64
duration_ms            int64
explicit                bool
danceability         float64
energy               float64
key                    int64
loudness             float64
mode                   int64
speechiness          float64
acousticness         float64
instrumentalness     float64
liveness             float64
valence              float64
tempo                float64
time_signature         int64
track_genre           object
dtype: object
```

[30]:
```
#changing to suitable datatypes
df['mode'] = df['mode'].astype('category')
df['explicit'] = df['explicit'].astype('category')
df['track_genre'] = df['track_genre'].astype('category')
#check datatypes again
df.dtypes
```

[30]:
```
track_id               object
artists                object
track_name             object
popularity              int64
duration_ms             int64
explicit             category
danceability          float64
energy                float64
key                     int64
loudness              float64
mode                 category
speechiness           float64
acousticness          float64
instrumentalness      float64
liveness              float64
valence               float64
```

```
tempo                   float64
time_signature            int64
track_genre            category
dtype: object
```

4.1.5 3.1.5 Standardize columns

Check for consistent format in our categorical by ensuring that they have consistent naming

[31]:
```python
# Select categorical columns
categorical_cols = df.select_dtypes(include=["object","category"]).columns

for col in categorical_cols:
    print(f"Unique values in '{col}':\n", df[col].unique(), "\n")
```

```
Unique values in 'track_id':
 ['5SuOikwiRyPMVoIQDJUgSV' '4qPNDBW1i3p13qLCt0Ki3A'
 '1iJBSr7s7jYXzM8EGcbK5b' … '6x8ZfSoqDjuNa5SVP5QjvX'
 '2e6sXL2bYv4bSz6VTdnfLs' '2hETkH7cOfqmz3LqZDHZf5']

Unique values in 'artists':
 ['Gen Hoshino' 'Ben Woodward' 'Ingrid Michaelson;ZAYN' …
 'Cuencos Tibetanos Sonidos Relajantes'
 'Bryan & Katie Torwalt;Brock Human' 'Jesus Culture']

Unique values in 'track_name':
 ['Comedy' 'Ghost - Acoustic' 'To Begin Again' … 'Water Into Light'
 'Miss Perfumado' 'Barbincor']

Unique values in 'explicit':
 [False, True]
Categories (2, bool): [False, True]

Unique values in 'mode':
 [0, 1]
Categories (2, int64): [0, 1]

Unique values in 'track_genre':
 ['acoustic', 'afrobeat', 'alt-rock', 'alternative', 'ambient', …, 'techno',
'trance', 'trip-hop', 'turkish', 'world-music']
Length: 113
Categories (113, object): ['acoustic', 'afrobeat', 'alt-rock', 'alternative',
…, 'trance', 'trip-hop', 'turkish', 'world-music']
```

4.1.6 3.1.7 Final Checks

After cleaning,we are reviewing the dataframe using df.head() and df.info() to ensure that the cleaning steps have been applied correctly

[32]:
```python
#view the data
df.head(10)
```

[32]:
```
                 track_id                                 artists  \
0  5SuOikwiRyPMVoIQDJUgSV                             Gen Hoshino
1  4qPNDBW1i3p13qLCt0Ki3A                            Ben Woodward
2  1iJBSr7s7jYXzM8EGcbK5b                  Ingrid Michaelson;ZAYN
3  6lfxq3CG4xtTiEg7opyCyx                            Kina Grannis
4  5vjLSffimiIP26QG5WcN2K                        Chord Overstreet
5  01MVOl9KtVTNfFiBU9I7dc                            Tyrone Wells
6  6Vc5wAMmXdKIAM7WUoEb7N  A Great Big World;Christina Aguilera
7  1EzrEOXmMH3G43AXT1y7pA                              Jason Mraz
8  0IktbUcnAGrvD03AWnz3Q8              Jason Mraz;Colbie Caillat
9  7k9GuJYLp2AzqokyEdwEw2                          Ross Copperman

                   track_name  popularity  duration_ms explicit  danceability  \
0                      Comedy          73       230666    False         0.676
1            Ghost - Acoustic          55       149610    False         0.420
2              To Begin Again          57       210826    False         0.438
3  Can't Help Falling In Love          71       201933    False         0.266
4                     Hold On          82       198853    False         0.618
5       Days I Will Remember          58       214240    False         0.688
6              Say Something           74       229400    False         0.407
7                   I'm Yours          80       242946    False         0.703
8                       Lucky          74       189613    False         0.625
9                      Hunger          56       205594    False         0.442

   energy  key  loudness  mode  speechiness  acousticness  instrumentalness  \
0  0.4610    1    -6.746     0       0.1430        0.0322          0.000001
1  0.1660    1   -17.235     1       0.0763        0.9240          0.000006
2  0.3590    0    -9.734     1       0.0557        0.2100          0.000000
3  0.0596    0   -18.515     1       0.0363        0.9050          0.000071
4  0.4430    2    -9.681     1       0.0526        0.4690          0.000000
5  0.4810    6    -8.807     1       0.1050        0.2890          0.000000
6  0.1470    2    -8.822     1       0.0355        0.8570          0.000003
7  0.4440   11    -9.331     1       0.0417        0.5590          0.000000
8  0.4140    0    -8.700     1       0.0369        0.2940          0.000000
9  0.6320    1    -6.770     1       0.0295        0.4260          0.004190

   liveness  valence    tempo  time_signature track_genre
0    0.3580   0.7150   87.917               4    acoustic
1    0.1010   0.2670   77.489               4    acoustic
2    0.1170   0.1200   76.332               4    acoustic
3    0.1320   0.1430  181.740               3    acoustic
4    0.0829   0.1670  119.949               4    acoustic
5    0.1890   0.6660   98.017               4    acoustic
6    0.0913   0.0765  141.284               3    acoustic
```

```
7    0.0973   0.7120  150.960              4    acoustic
8    0.1510   0.6690  130.088              4    acoustic
9    0.0735   0.1960   78.899              4    acoustic
```

[33]:
```
# checking genre value counts
df.track_genre.value_counts()
```

[33]:
```
track_genre
acoustic     1000
alt-rock      999
tango         999
ambient       999
afrobeat      999
             …
metal         232
punk          226
house         210
indie         134
reggaeton      74
Name: count, Length: 113, dtype: int64
```

[34]:
```
#view the dataset info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 89740 entries, 0 to 113999
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   track_id          89740 non-null  object
 1   artists           89740 non-null  object
 2   track_name        89740 non-null  object
 3   popularity        89740 non-null  int64
 4   duration_ms       89740 non-null  int64
 5   explicit          89740 non-null  category
 6   danceability      89740 non-null  float64
 7   energy            89740 non-null  float64
 8   key               89740 non-null  int64
 9   loudness          89740 non-null  float64
 10  mode              89740 non-null  category
 11  speechiness       89740 non-null  float64
 12  acousticness      89740 non-null  float64
 13  instrumentalness  89740 non-null  float64
 14  liveness          89740 non-null  float64
 15  valence           89740 non-null  float64
 16  tempo             89740 non-null  float64
 17  time_signature    89740 non-null  int64
 18  track_genre       89740 non-null  category
```

```
dtypes: category(3), float64(9), int64(4), object(3)
memory usage: 11.9+ MB
```

[35]:
```
#reset index
df.reset_index(drop=True, inplace=True)
#review the dataset info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 89740 entries, 0 to 89739
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   track_id          89740 non-null  object
 1   artists           89740 non-null  object
 2   track_name        89740 non-null  object
 3   popularity        89740 non-null  int64
 4   duration_ms       89740 non-null  int64
 5   explicit          89740 non-null  category
 6   danceability      89740 non-null  float64
 7   energy            89740 non-null  float64
 8   key               89740 non-null  int64
 9   loudness          89740 non-null  float64
 10  mode              89740 non-null  category
 11  speechiness       89740 non-null  float64
 12  acousticness      89740 non-null  float64
 13  instrumentalness  89740 non-null  float64
 14  liveness          89740 non-null  float64
 15  valence           89740 non-null  float64
 16  tempo             89740 non-null  float64
 17  time_signature    89740 non-null  int64
 18  track_genre       89740 non-null  category
dtypes: category(3), float64(9), int64(4), object(3)
memory usage: 11.2+ MB
```

[36]:
```
df.columns
```

[36]:
```
Index(['track_id', 'artists', 'track_name', 'popularity', 'duration_ms',
       'explicit', 'danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'time_signature', 'track_genre'],
      dtype='object')
```

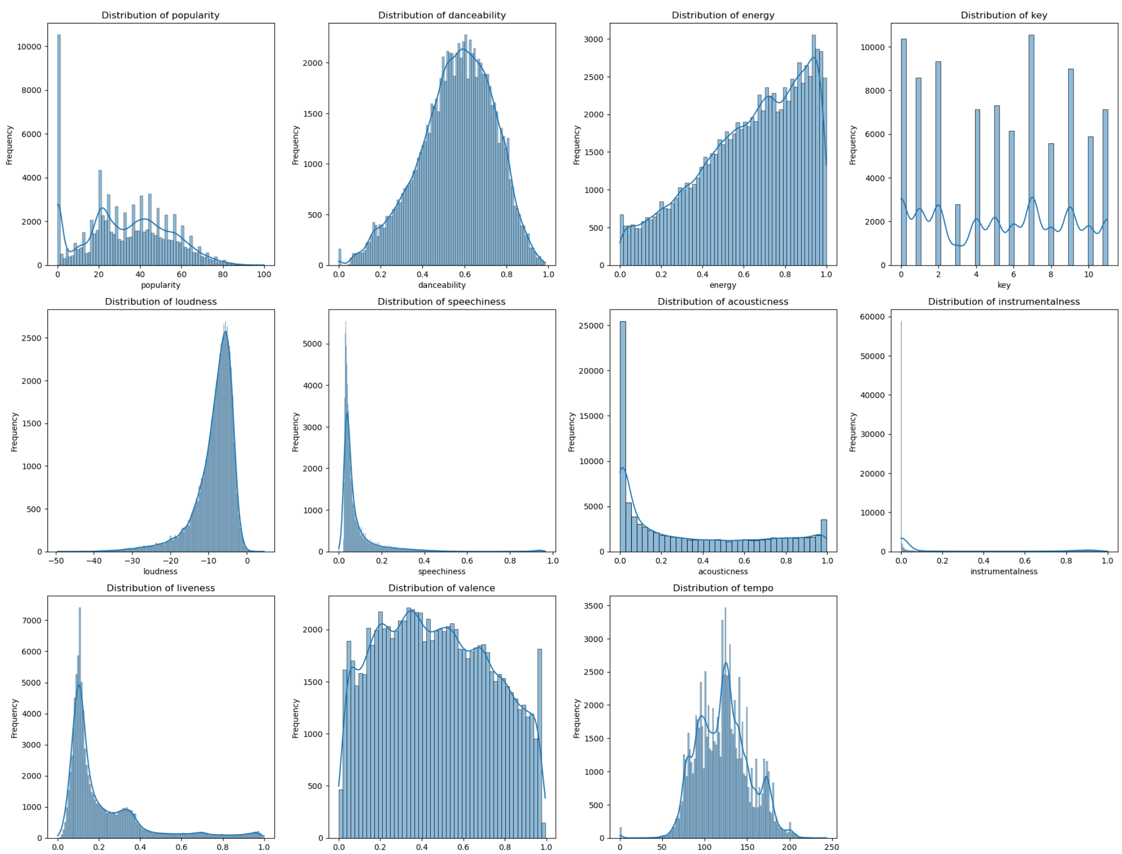
4.2 3.2 Exploratory Data Analysis

4.2.1 3.2.1 Univariate Analysis

[37]:
```
num_columns = df[['popularity','danceability', 'energy', 'key', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']]
categorical_cols=df[['explicit','mode']]
```

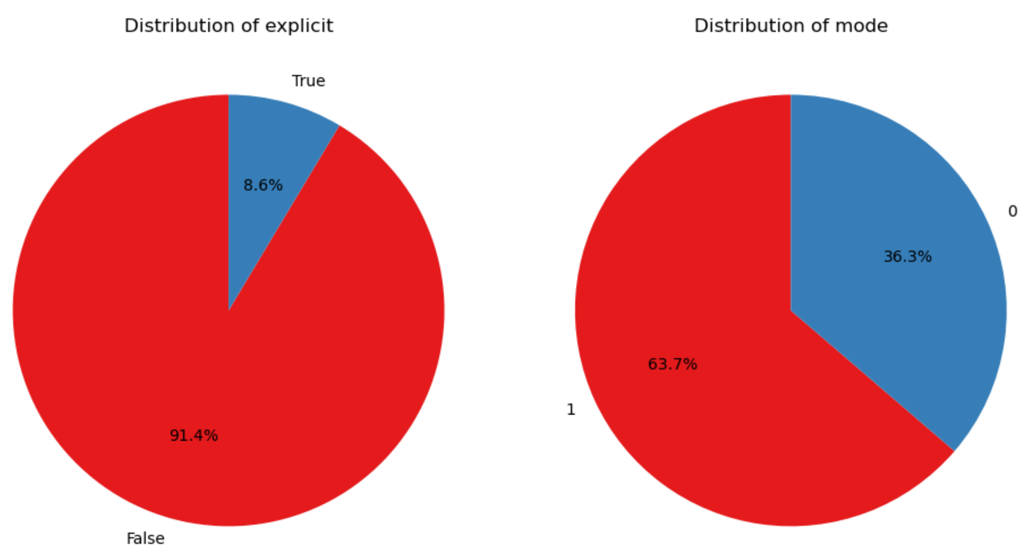
1. Subplot Numerical columns

[38]:
```
#plotting the distribution of numerical columns
plt.figure(figsize=(20, 20))
for i, col in enumerate(num_columns, 1):
    plt.subplot(4, 4, i)
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show();
```



2. Pie Chart Distribution of Categorical columns

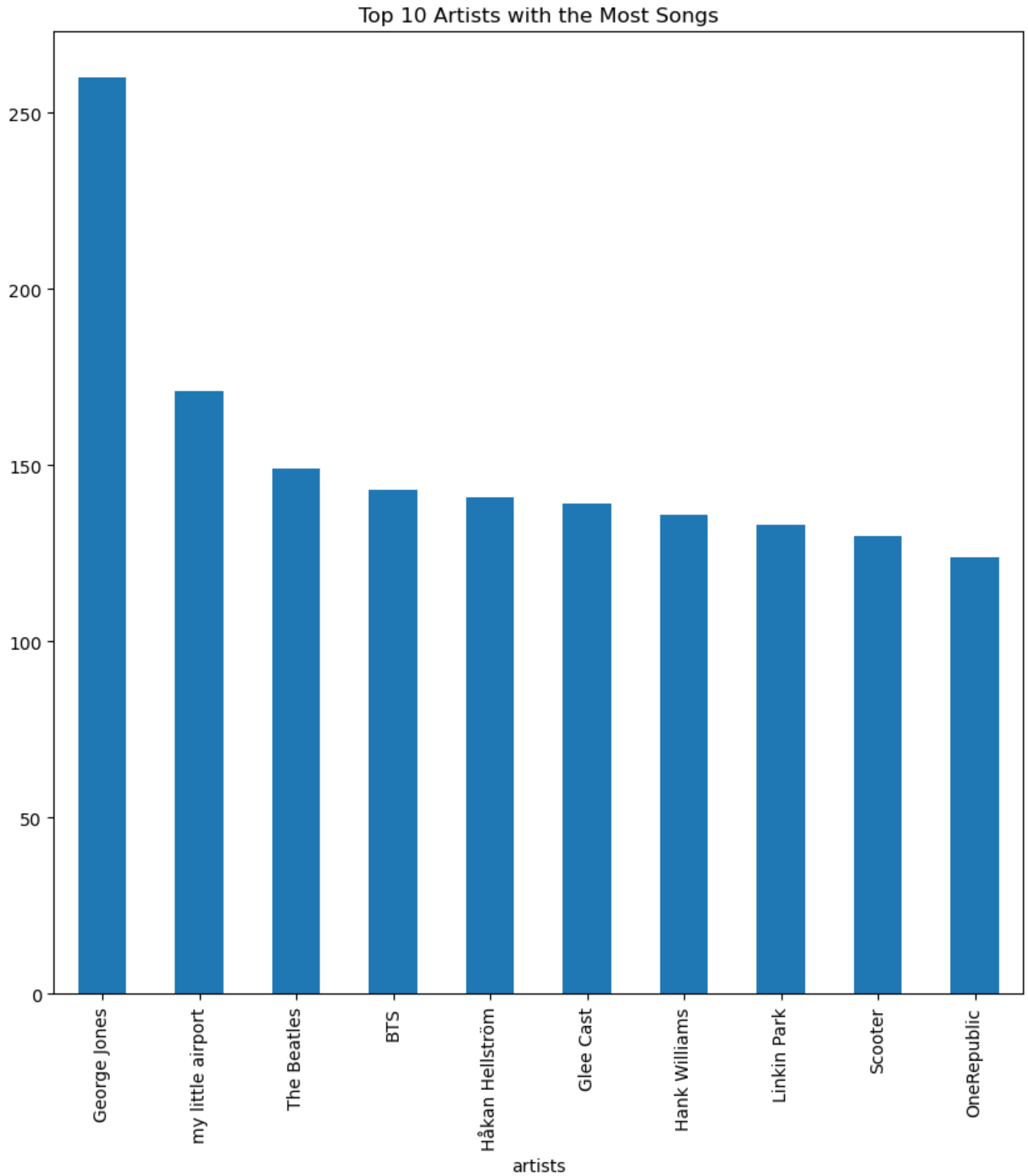
[39]:
```python
#plotting the distribution of categorical columns

# Define a high-contrast color palette manually
high_contrast_colors = [
    "#e41a1c", "#377eb8", "#4daf4a", "#984ea3", "#ff7f00",
    "#ffff33", "#a65628", "#f781bf", "#999999"
]

# Adjust figure size
plt.figure(figsize=(20, 20))

# Loop through categorical columns and create pie charts
for i, col in enumerate(categorical_cols, 1):
    plt.subplot(4, 4, i)  # Create a 4x4 grid of subplots
    value_counts = df[col].value_counts()  # Count occurrences

    # Select a subset of colors for current chart
    colors = high_contrast_colors[:len(value_counts)]

    plt.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%',
            startangle=90, colors=colors)
    plt.title(f'Distribution of {col}')  # Title for each subplot

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



3. Bar Plot (Total Artist)

[40]:
```
#total number of artists
total_artists = df['artists'].unique()
print(f'The total number of artists in our dataset is: {len(total_artists)}')

#plotting using barplot
plt.figure(figsize=(10, 10))
df['artists'].value_counts().head(10).plot(kind='bar')
plt.title('Top 10 Artists with the Most Songs')
plt.show();
```

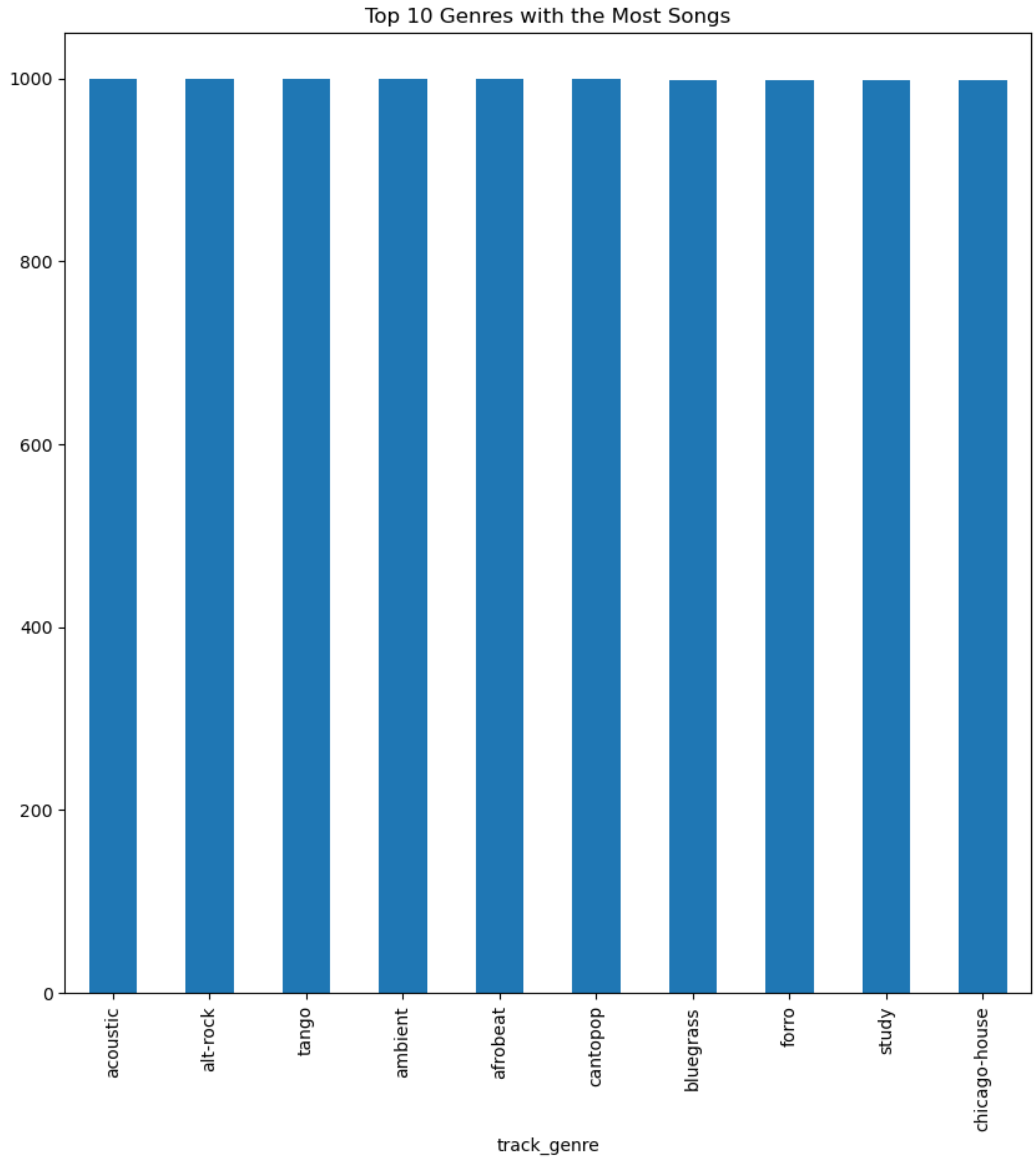
```
The total number of artists in our dataset is: 31437
```



4. Bar Plot (Genre count)

[41]:
```
#total number of genres
total_genres = df['track_genre'].unique()
print(f'The total number of genres in our dataset is: {len(total_genres)}')

#plotting
plt.figure(figsize=(10, 10))
df['track_genre'].value_counts().head(10).plot(kind='bar')
plt.title('Top 10 Genres with the Most Songs')
plt.show();
```

```
The total number of genres in our dataset is: 113
```



4.2.2 3.2.2. Bivariate Analysis

bivariate - genre__duration,genre__popularity,gerne__valence,genre__tempo

1. Pair-plots for each feature

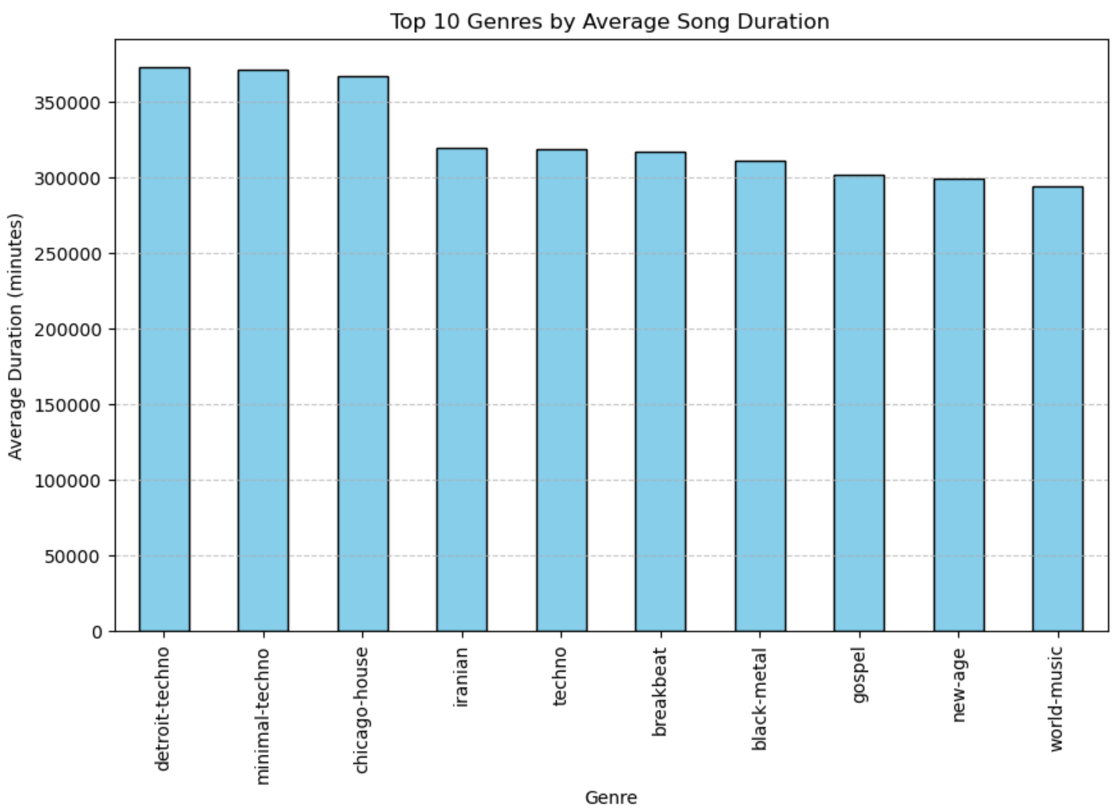
[42]:
```
df.columns
```

[42]:
```
Index(['track_id', 'artists', 'track_name', 'popularity', 'duration_ms',
       'explicit', 'danceability', 'energy', 'key', 'loudness', 'mode',
```

```
 'speechiness', 'acousticness', 'instrumentalness', 'liveness',
 'valence', 'tempo', 'time_signature', 'track_genre'],
dtype='object')
```

2. Bar Plot (Average Track_genre by duration_min)

[43]:
```
#mean duration for each genre and get the top 10
df_genre_duration = df.groupby('track_genre')['duration_ms'].mean().
 ↪sort_values(ascending=False).head(10)

# Plot the bar chart
plt.figure(figsize=(10, 6))
df_genre_duration.plot(kind='bar', color='skyblue', edgecolor='black')

# Improve visualization
plt.title('Top 10 Genres by Average Song Duration')
plt.ylabel('Average Duration (minutes)')
plt.xlabel('Genre')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid for better readability
plt.show();
```

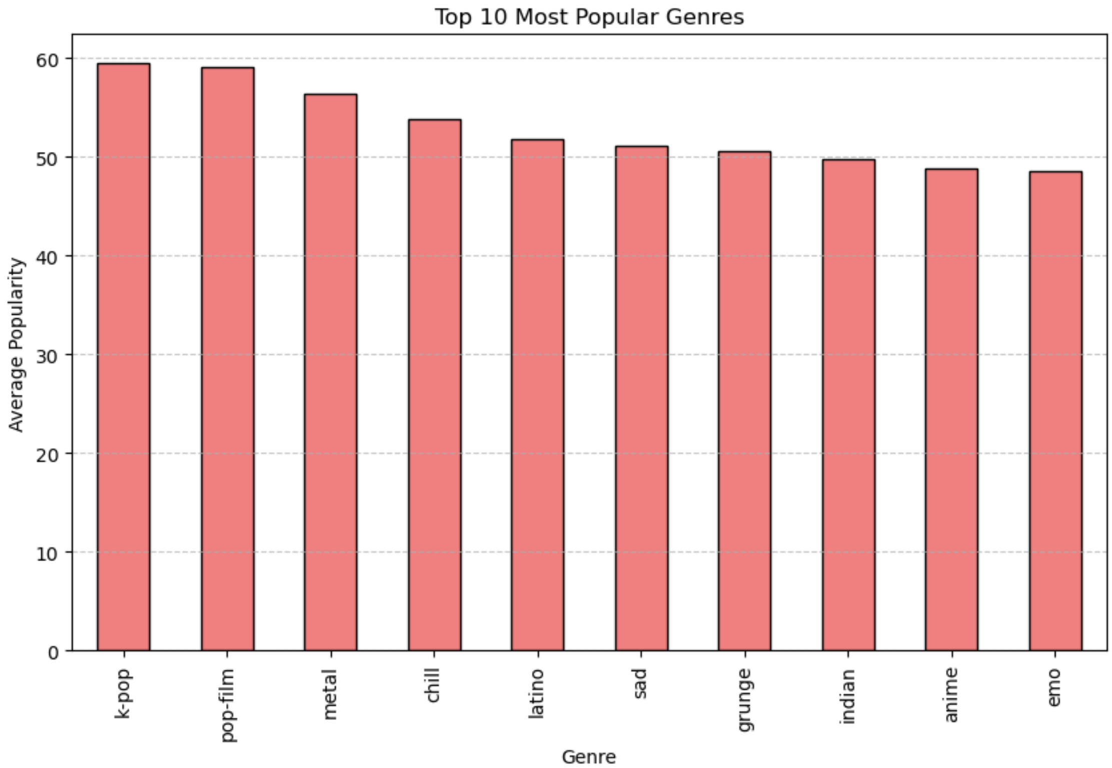


3. Bar Plot (Track_genre and Popularity)

[44]:
```
#plotting most popular genres
df_genre_popularity = df.groupby('track_genre', observed=True)['popularity'].
↪mean().sort_values(ascending=False).head(10)
print('Top 10 Most Popular Genres:')
print(df_genre_popularity)
```

```
Top 10 Most Popular Genres:
track_genre
k-pop       59.423581
pop-film    59.096933
metal       56.422414
chill       53.738683
latino      51.788945
sad         51.109929
grunge      50.587007
indian      49.765348
anime       48.776884
emo         48.500000
Name: popularity, dtype: float64
```

[45]:
```
# Plot the bar chart
plt.figure(figsize=(10, 6))
df_genre_popularity.plot(kind='bar', color='lightcoral', edgecolor='black')

# Improve visualization
plt.title('Top 10 Most Popular Genres')
plt.ylabel('Average Popularity')
plt.xlabel('Genre')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid for better readability
plt.show();
```

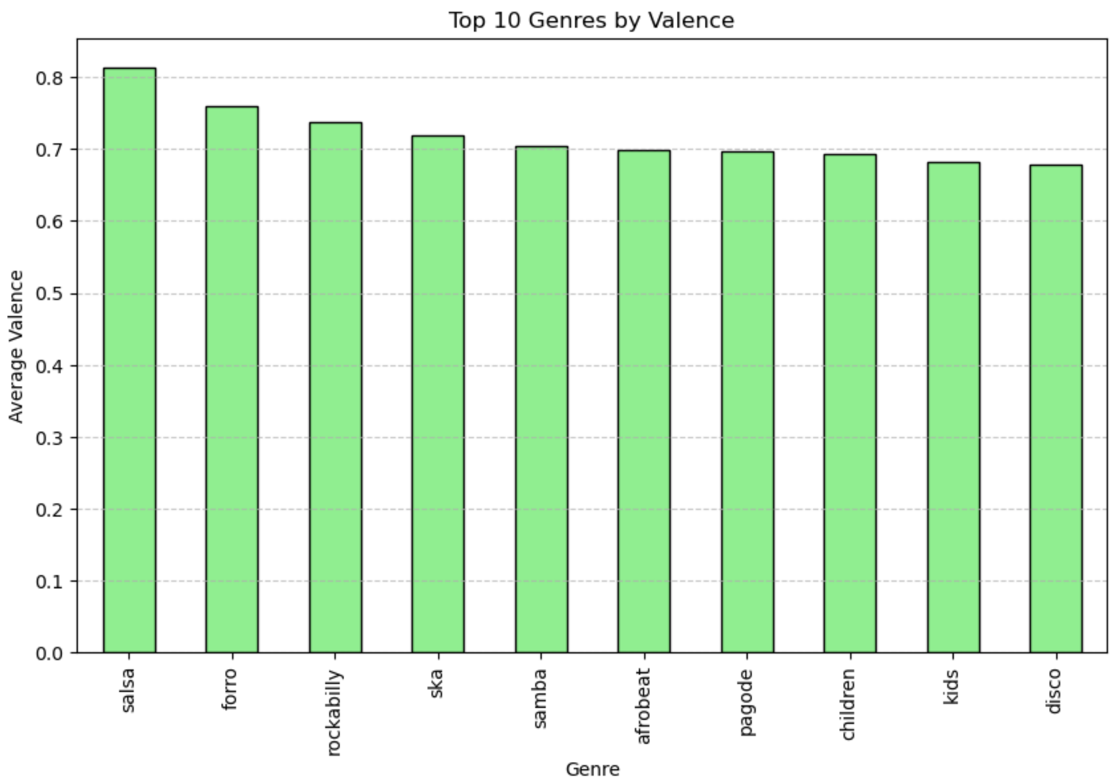


4. Bar Plot (Track_genre and Valence)

[46]:
```
#genre by valence
df_genre_valence = df.groupby('track_genre', observed=True)['valence'].mean().
 ↪sort_values(ascending=False).head(10)
print('Top 10 Genres by Valence:')
print(df_genre_valence)
```

```
Top 10 Genres by Valence:
track_genre
salsa          0.813806
forro          0.760499
rockabilly     0.737709
ska            0.719166
samba          0.705054
afrobeat       0.698475
pagode         0.697245
children       0.693671
kids           0.681698
disco          0.679275
Name: valence, dtype: float64
```

[47]:
```python
#plotting charts
plt.figure(figsize=(10, 6))
df_genre_valence.plot(kind='bar', color='lightgreen', edgecolor='black')

# Improve visualization
plt.title('Top 10 Genres by Valence')
plt.ylabel('Average Valence')
plt.xlabel('Genre')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid for better readability
plt.show();
```



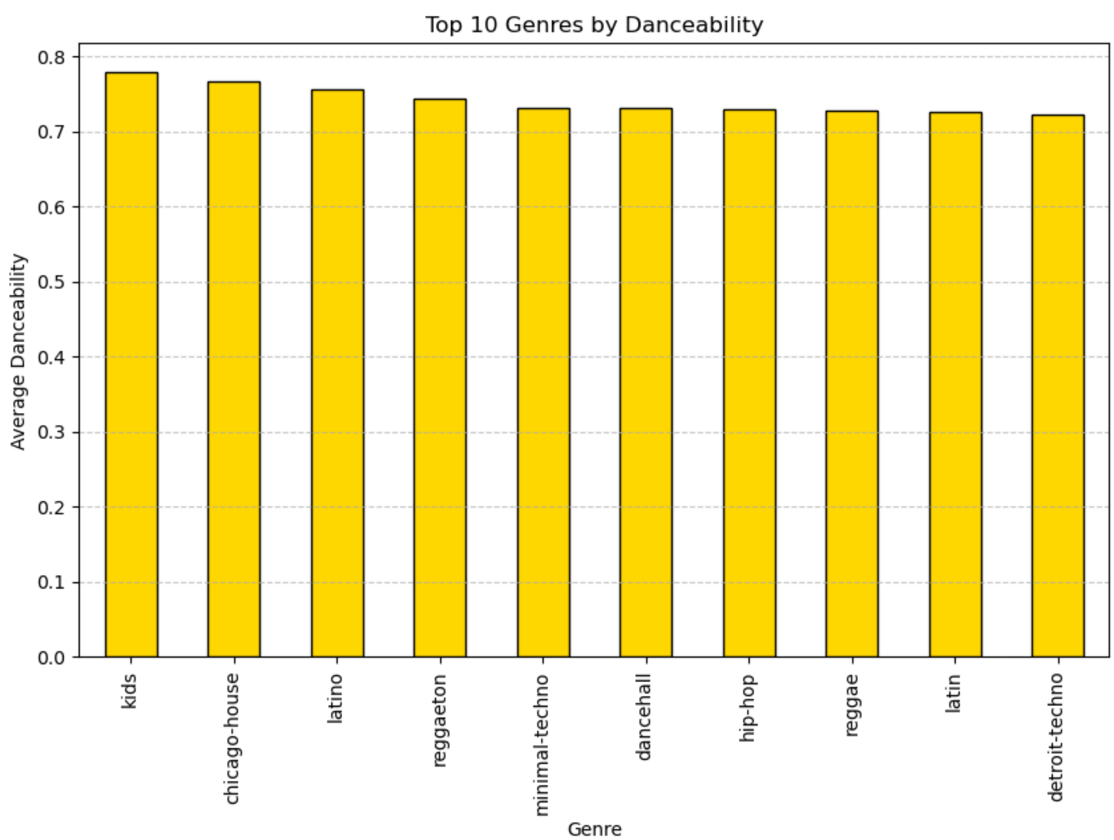
5. Bar Plot (Track_genre and Danceability)

[48]:
```python
#genre by danceability
df_genre_danceability = df.groupby('track_genre',␣
↪observed=True)['danceability'].mean().sort_values(ascending=False).head(10)
print('Top 10 Genres by Danceability:')
print(df_genre_danceability)
```

```
Top 10 Genres by Danceability:
track_genre
kids                0.778808
```

```
chicago-house     0.766240
latino            0.755487
reggaeton         0.743284
minimal-techno    0.732045
dancehall         0.731430
hip-hop           0.730052
reggae            0.728457
latin             0.726954
detroit-techno    0.722664
Name: danceability, dtype: float64
```

[49]:
```python
#plotting
plt.figure(figsize=(10, 6))
df_genre_danceability.plot(kind='bar', color='gold', edgecolor='black')

# Improve visualization
plt.title('Top 10 Genres by Danceability')
plt.ylabel('Average Danceability')
plt.xlabel('Genre')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid for better readability
plt.show();
```

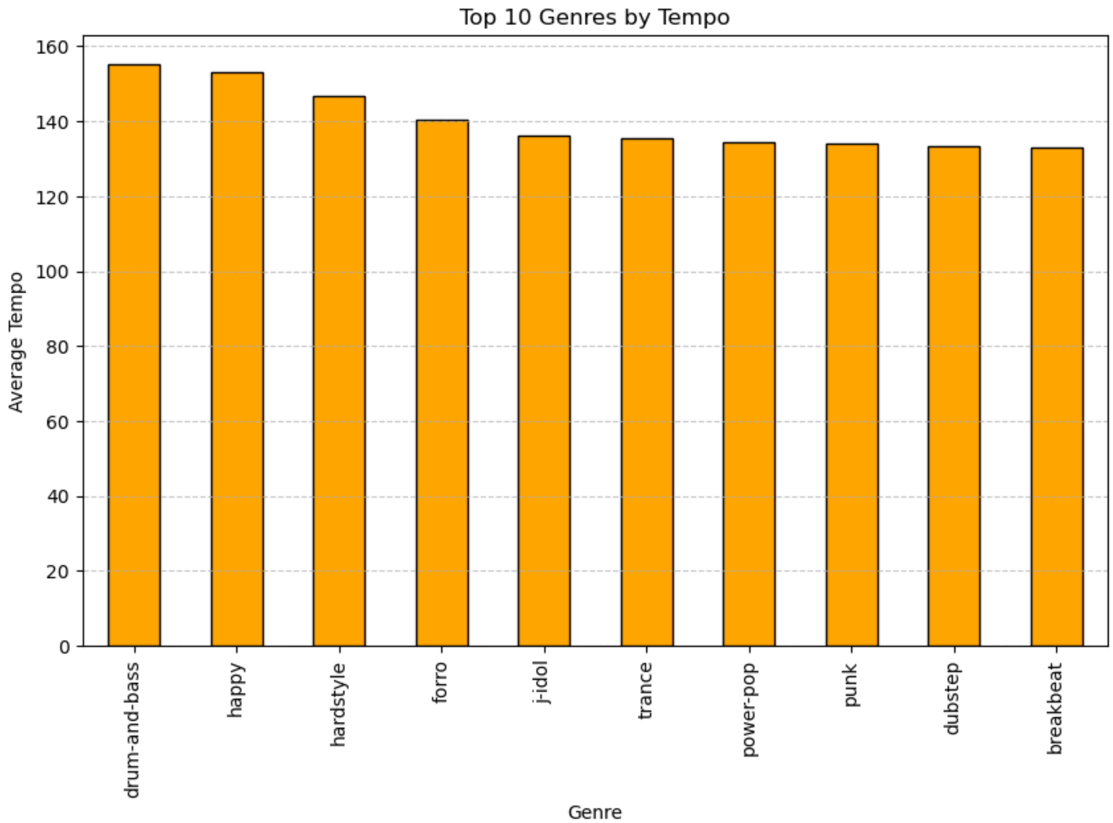


6. Bar Plot (Track_genre and Tempo)

[50]:
```
#genre by tempo
df_genre_tempo = df.groupby('track_genre', observed=True)['tempo'].mean().
  ↪sort_values(ascending=False).head(10)
print('Top 10 Genres by Tempo:')
print(df_genre_tempo)
```

```
Top 10 Genres by Tempo:
track_genre
drum-and-bass    155.151446
happy            152.956724
hardstyle        146.688615
forro            140.350656
j-idol           136.081969
trance           135.270682
power-pop        134.516428
punk             134.137451
dubstep          133.368716
breakbeat        133.068423
Name: tempo, dtype: float64
```

[51]:
```
#plotting
plt.figure(figsize=(10, 6))
df_genre_tempo.plot(kind='bar', color='orange', edgecolor='black')
# Improve visualization
plt.title('Top 10 Genres by Tempo')
plt.ylabel('Average Tempo')
plt.xlabel('Genre')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid for better readability
plt.show();
```

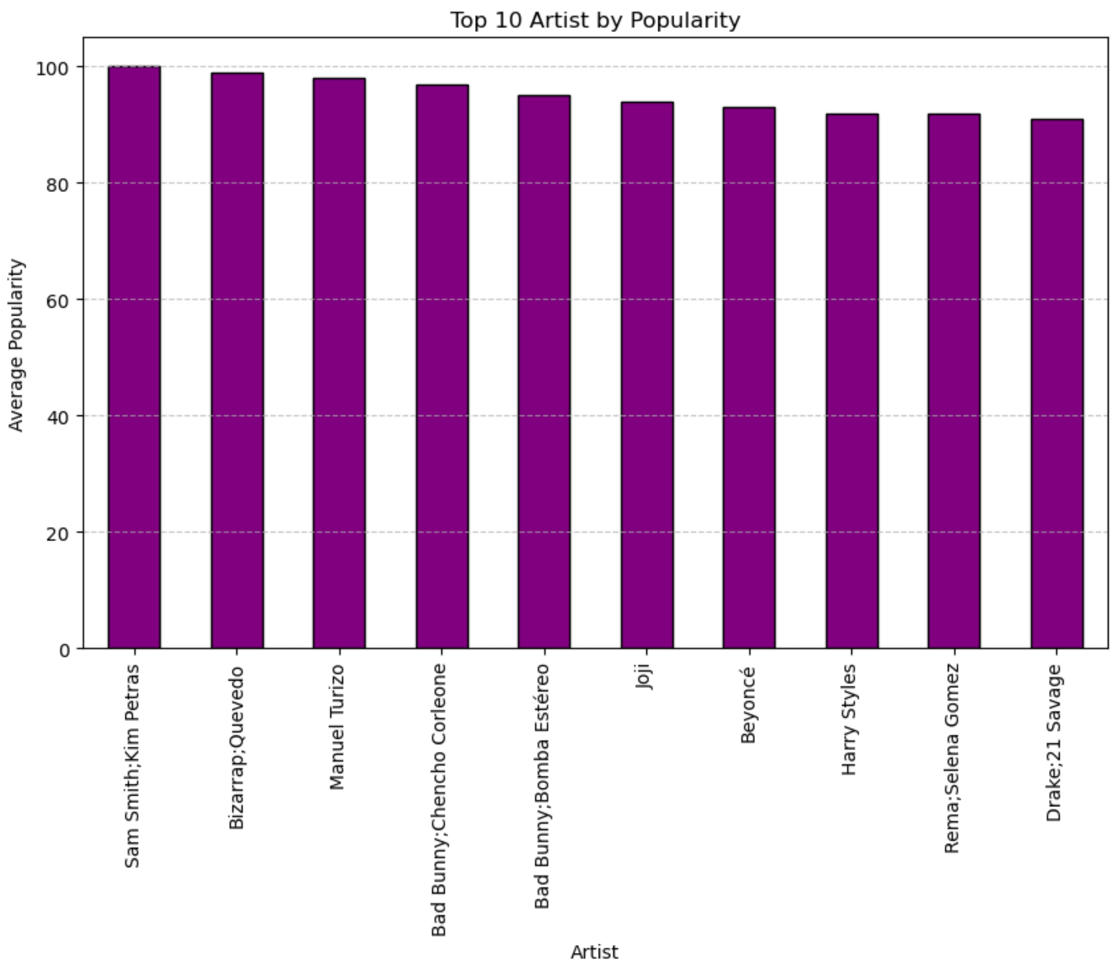


7. Bar Plot (Artist by Popularity)

[52]:
```
#artist by popularity
df_artist_popularity = df.groupby('artists', observed=True)['popularity'].
    ↪mean().sort_values(ascending=False).head(10)
print('Top 10 Artists by Popularity:')
print(df_artist_popularity)
```

```
Top 10 Artists by Popularity:
artists
Sam Smith;Kim Petras          100.0
Bizarrap;Quevedo               99.0
Manuel Turizo                  98.0
Bad Bunny;Chencho Corleone     97.0
Bad Bunny;Bomba Estéreo        95.0
Joji                           94.0
Beyoncé                        93.0
Harry Styles                   92.0
Rema;Selena Gomez              92.0
Drake;21 Savage                91.0
Name: popularity, dtype: float64
```

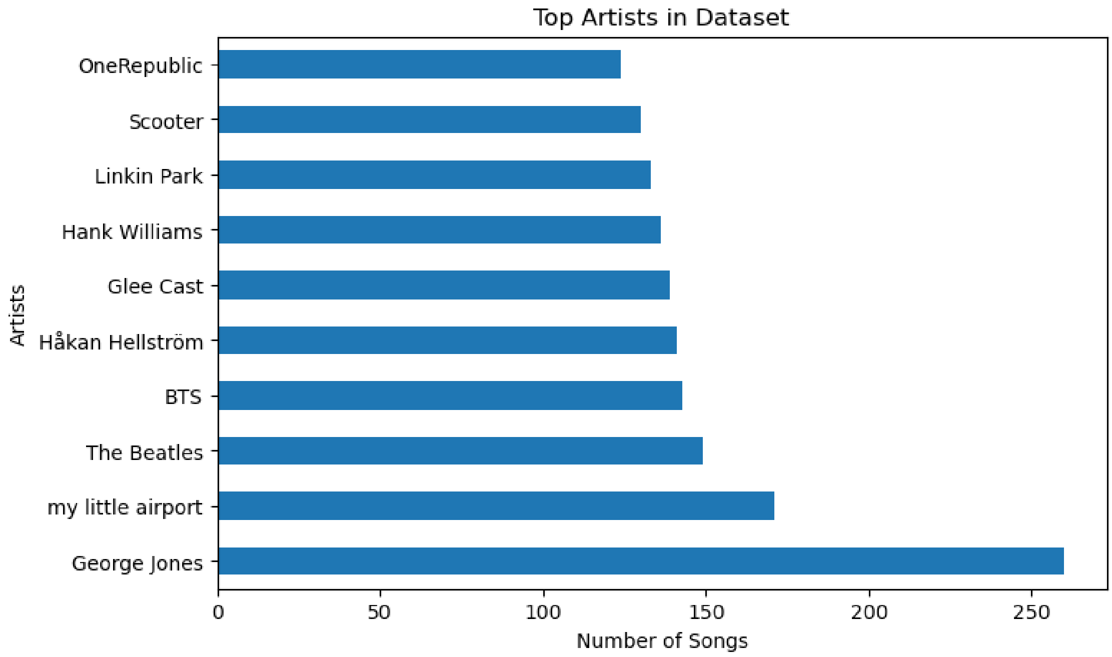
[53]:
```
#plotting
plt.figure(figsize=(10, 6))
df_artist_popularity.plot(kind='bar', color='purple', edgecolor='black')

# Improve visualization
plt.title('Top 10 Artist by Popularity')
plt.ylabel('Average Popularity')
plt.xlabel('Artist')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add grid for better readability
plt.show();
```



8. Top 10 artists

[54]:
```
# The top 10 artists generally
df['artists'].value_counts().head(10).plot(kind='barh', figsize=(8, 5))
plt.xlabel("Number of Songs")
```

```
plt.ylabel("Artists")
plt.title("Top Artists in Dataset")
plt.show();
```

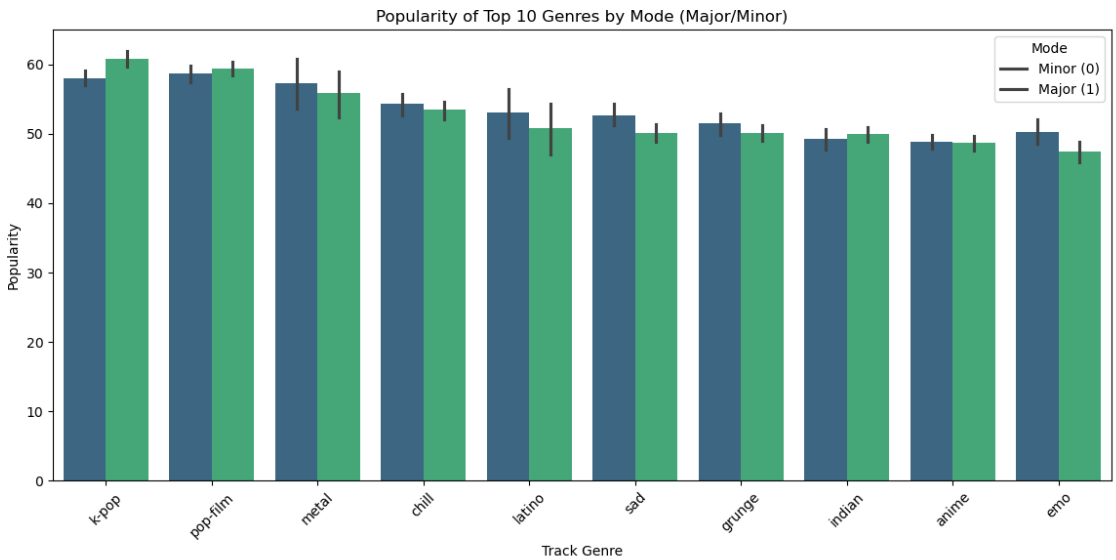


4.2.3 3.3 Multivariate Analysis

1. Pairplots (Numerical and categorical colums with hue track_genre)

[55]:
```
# Compute the average popularity per genre
avg_popularity_per_genre = df.groupby("track_genre")["popularity"].mean().
    ↪reset_index()

# Get the top 10 most popular genres
top_10_popular_genres = avg_popularity_per_genre.sort_values(by="popularity",␣
    ↪ascending=False).head(10)["track_genre"]

# Filter the dataset to include only the top 10 popular genres
df_top_popular_genres = df[df["track_genre"].isin(top_10_popular_genres)]

# Set the order of categories for the x-axis to match the top 10
plt.figure(figsize=(14, 6))
ax = sns.barplot(
    data=df_top_popular_genres,
    x="track_genre",
    y="popularity",
    hue="mode",
```

```
    palette="viridis",
    order=top_10_popular_genres  # Ensures only the top 10 genres are displayed
)

plt.xticks(rotation=45)
plt.title("Popularity of Top 10 Genres by Mode (Major/Minor)")
plt.xlabel("Track Genre")
plt.ylabel("Popularity")
plt.legend(title="Mode", labels=["Minor (0)", "Major (1)"])

plt.show();
```

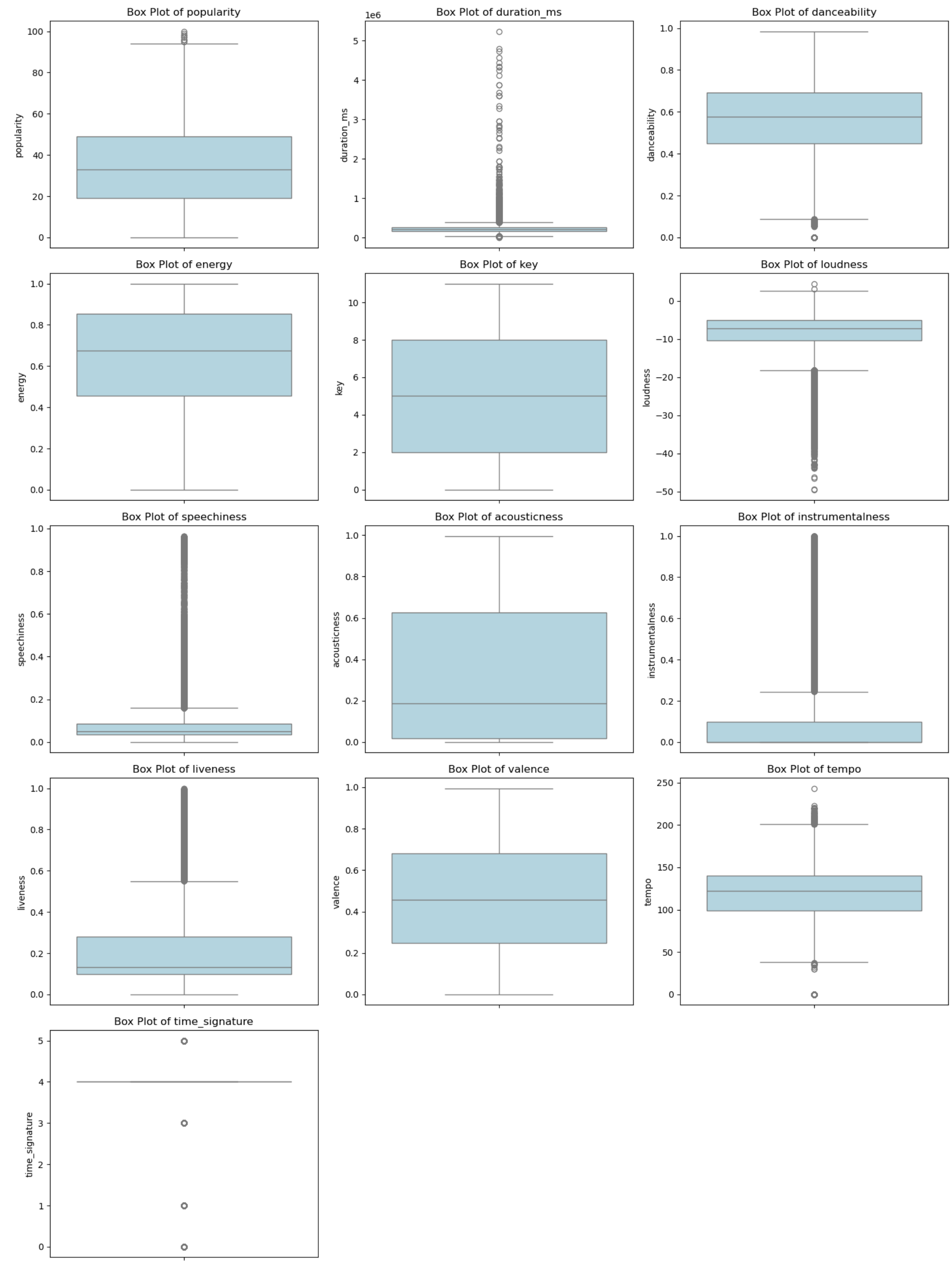


4.3 3.3. Handle outliers

We will identify the outliers using box plots then decide on a strategy on how to deal with it depending on the outcome

[57]:
```
# Select numerical columns
numeric_columns = df.select_dtypes(exclude=['object', 'bool', 'category'])

# Number of numerical columns
num_cols = len(numeric_columns.columns)

# Define number of rows and columns for subplots
rows = (num_cols // 3) + (num_cols % 3 > 0)  # Adjust rows based on number of␣
↪columns
cols = min(3, num_cols)  # Max 3 columns per row
```

```python
# Create subplots
fig, axes = plt.subplots(rows, cols, figsize=(15, rows * 4))

# Flatten axes array for easier iteration
axes = axes.flatten() if num_cols > 1 else [axes]

# Plot each boxplot
for i, col in enumerate(numeric_columns.columns):
    sns.boxplot(y=df[col], ax=axes[i], color='lightblue')
    axes[i].set_title(f'Box Plot of {col}')

# Remove empty subplots if any
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



there's plenty of outliers in duration_ms but we decided to keep them since they are sensitive columns

Checking skewness for numerical columns

[58]:
```python
# Select only numeric columns
numeric_columns = df.select_dtypes(include=['number'])

# Calculate skewness for each numeric column
skewness = numeric_columns.apply(skew).sort_values(ascending=False)

# Print skewness values
print(skewness)
```

```
duration_ms         11.072616
speechiness          4.545759
liveness             2.062058
instrumentalness     1.563971
acousticness         0.655761
tempo                0.182741
valence              0.127635
popularity           0.070862
key                 -0.000142
danceability        -0.398285
energy              -0.559983
loudness            -1.959847
time_signature      -3.998744
dtype: float64
```

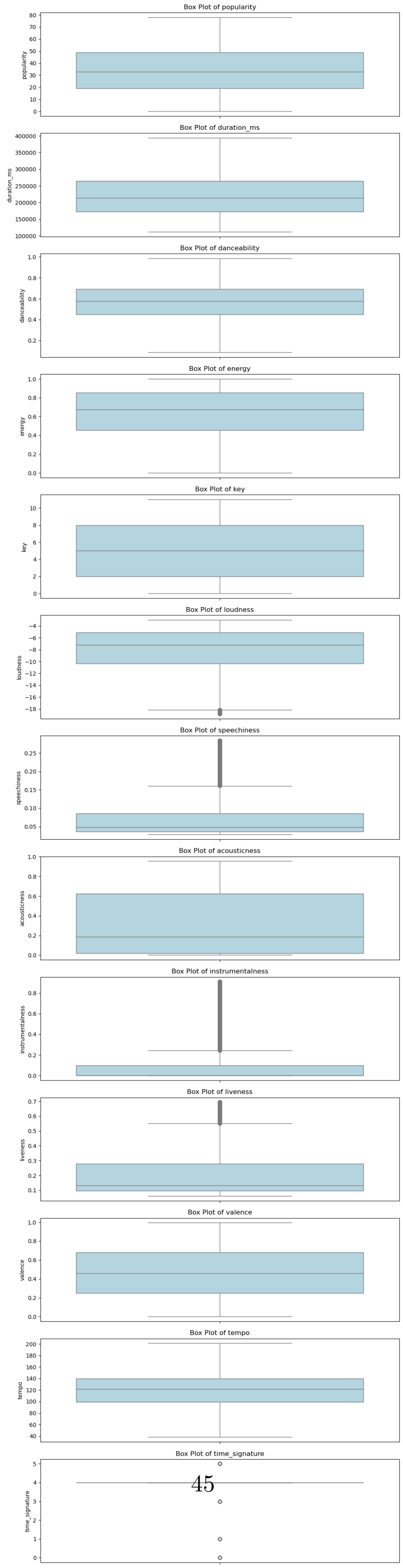
[59]:
```python
df.shape
```

[59]:
```
(89740, 19)
```

Outlier Removal

[60]:
```python
from scipy.stats.mstats import winsorize
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Winsorization (for highly skewed data)
winsorize_columns = ['loudness', 'speechiness', 'acousticness',␣
 ↪'instrumentalness', 'liveness', 'duration_ms']
for col in winsorize_columns:
    df[col] = winsorize(df[col], limits=[0.05, 0.05])  # Winsorize at 5th and␣
 ↪95th percentile

# IQR Method (Capping Outliers)
iqr_columns = ['danceability', 'valence', 'tempo']
for col in iqr_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
```

```python
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
    df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])

# Clipping (for bi-modal/multi-modal data)
clip_columns = ['popularity', 'key']
for col in clip_columns:
    lower_bound = df[col].quantile(0.01)
    upper_bound = df[col].quantile(0.99)
    df[col] = np.clip(df[col], lower_bound, upper_bound)

# Boxplot Visualization
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

fig, axes = plt.subplots(len(numeric_columns), 1, figsize=(10, len(numeric_columns) * 3))

for i, col in enumerate(numeric_columns):
    sns.boxplot(y=df[col], ax=axes[i], color='lightblue')
    axes[i].set_title(f'Box Plot of {col}')

# Remove empty subplots
for j in range(i + 1, len(axes)):
    if j < len(fig.axes):
        fig.delaxes(axes[j])

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


Box Plot of popularity
popularity
Box Plot of duration_ms
duration_ms
Box Plot of danceability
danceability
Box Plot of energy
energy
Box Plot of key
key
Box Plot of loudness
loudness
Box Plot of speechiness
speechiness
Box Plot of acousticness
acousticness
Box Plot of instrumentalness
instrumentalness
Box Plot of liveness
liveness
Box Plot of valence
valence
Box Plot of tempo
tempo
Box Plot of time_signature
time_signature

[61]:
```
# checking dataframe shape
df.shape
```

[61]: (89740, 19)

[62]:
```
# checking if the track duration time outlier is handled
df.duration_ms
```

[62]:
```
0        230666
1        149610
2        210826
3        201933
4        198853
          …
89735    384999
89736    385000
89737    271466
89738    283893
89739    241826
Name: duration_ms, Length: 89740, dtype: int64
```

[63]:
```
#convert duration_ms to minutes
df['duration_min'] = df['duration_ms'] / 60000
```

4.4 3.3. Feature Engineering

[64]:
```
# creating a new column for different minutes in a track

# Define classification function
def classify_duration(mins):
    if mins < 3:
        return "Very Short"
    elif 3 <= mins < 10:
        return "Short"
    elif 10 <= mins < 30:
        return "Medium"
    elif 30 <= mins < 60:
        return "Long"
    else:
        return "Very Long"

# Apply classification
df['duration_category'] = df['duration_min'].apply(classify_duration)

# Display sample results
```

```
df[['duration_min', 'duration_category']].sample(5)
```

[64]:
```
       duration_min duration_category
86746      5.565217             Short
37237      3.936500             Short
57751      3.629550             Short
34487      4.477100             Short
52527      4.829100             Short
```

[65]:
```
df.head()
```

[65]:
```
                 track_id                 artists                  track_name  \
0  5SuOikwiRyPMVoIQDJUgSV             Gen Hoshino                      Comedy
1  4qPNDBW1i3p13qLCt0Ki3A            Ben Woodward            Ghost - Acoustic
2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN              To Begin Again
3  6lfxq3CG4xtTiEg7opyCyx            Kina Grannis  Can't Help Falling In Love
4  5vjLSffimiIP26QG5WcN2K        Chord Overstreet                     Hold On

   popularity  duration_ms explicit  danceability  energy  key  loudness mode  \
0          73       230666    False         0.676  0.4610    1    -6.746    0
1          55       149610    False         0.420  0.1660    1   -17.235    1
2          57       210826    False         0.438  0.3590    0    -9.734    1
3          71       201933    False         0.266  0.0596    0   -18.515    1
4          78       198853    False         0.618  0.4430    2    -9.681    1

   speechiness  acousticness  instrumentalness  liveness  valence    tempo  \
0       0.1430        0.0322          0.000001    0.3580    0.715   87.917
1       0.0763        0.9240          0.000006    0.1010    0.267   77.489
2       0.0557        0.2100          0.000000    0.1170    0.120   76.332
3       0.0363        0.9050          0.000071    0.1320    0.143  181.740
4       0.0526        0.4690          0.000000    0.0829    0.167  119.949

   time_signature track_genre  duration_min duration_category
0               4    acoustic      3.844433             Short
1               4    acoustic      2.493500        Very Short
2               4    acoustic      3.513767             Short
3               3    acoustic      3.365550             Short
4               4    acoustic      3.314217             Short
```

[66]:
```
# Ensure 'mode' is numeric
df['mode'] = pd.to_numeric(df['mode'], errors='coerce')

# Feature Interactions
df['valence_dance'] = df['valence'] * df['danceability']
df['energy_tempo'] = df['energy'] * df['tempo']
df['dance_energy'] = df['danceability'] * df['energy']
df['speech_acoustic'] = df['speechiness'] * df['acousticness']
```

```python
df['loudness_energy'] = df['loudness'] * df['energy']
df['valence_mode'] = df['valence'] * df['mode']  # Now 'mode' is numeric


# Tempo Categorization
df['tempo_category'] = pd.cut(df['tempo'], bins=[0, 80, 120, 180, np.inf],
                              labels=['slow', 'medium', 'fast', 'very fast'])

print(df.columns)  # Verify new columns exist
```

```
Index(['track_id', 'artists', 'track_name', 'popularity', 'duration_ms',
       'explicit', 'danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'time_signature', 'track_genre', 'duration_min',
       'duration_category', 'valence_dance', 'energy_tempo', 'dance_energy',
       'speech_acoustic', 'loudness_energy', 'valence_mode', 'tempo_category'],
      dtype='object')
```

[67]:
```python
df.isnull().sum()
```

[67]:
```
track_id             0
artists              0
track_name           0
popularity           0
duration_ms          0
explicit             0
danceability         0
energy               0
key                  0
loudness             0
mode                 0
speechiness          0
acousticness         0
instrumentalness     0
liveness             0
valence              0
tempo                0
time_signature       0
track_genre          0
duration_min         0
duration_category    0
valence_dance        0
energy_tempo         0
dance_energy         0
speech_acoustic      0
loudness_energy      0
valence_mode         0
tempo_category       0
```

```
dtype: int64
```

[68]:
```
print(df.describe().T)
```

```
                    count           mean           std            min  \
popularity        89740.0      33.158803     20.485940       0.000000
duration_ms       89740.0  224452.492824  72716.157351  112042.000000
danceability      89740.0       0.562372      0.176098       0.087000
energy            89740.0       0.634458      0.256606       0.000000
key               89740.0       5.283530      3.559912       0.000000
loudness          89740.0      -8.250291      4.218523     -18.862000
mode              89740.0       0.636973      0.480875       0.000000
speechiness       89740.0       0.077958      0.068139       0.028300
acousticness      89740.0       0.327050      0.335951       0.000124
instrumentalness  89740.0       0.171931      0.320345       0.000000
liveness          89740.0       0.209854      0.169400       0.060800
valence           89740.0       0.469474      0.262864       0.000000
tempo             89740.0     122.107316     29.837579      38.041375
time_signature    89740.0       3.897426      0.453437       0.000000
duration_min      89740.0       3.740875      1.211936       1.867367
valence_dance     89740.0       0.286804      0.200276       0.000000
energy_tempo      89740.0      79.437940     40.212189       0.000000
dance_energy      89740.0       0.363231      0.179851       0.000000
speech_acoustic   89740.0       0.023010      0.037479       0.000004
loudness_energy   89740.0      -4.384083      1.850188     -18.862000
valence_mode      89740.0       0.302277      0.310427       0.000000

                            25%            50%            75%            max
popularity            19.000000      33.000000      49.000000      78.000000
duration_ms       173040.000000  213295.500000  264293.000000  394000.000000
danceability           0.450000       0.576000       0.692000       0.985000
energy                 0.457000       0.676000       0.853000       1.000000
key                    2.000000       5.000000       8.000000      11.000000
loudness             -10.322250      -7.185000      -5.108000      -3.000000
mode                   0.000000       1.000000       1.000000       1.000000
speechiness            0.036000       0.048900       0.085900       0.284000
acousticness           0.017100       0.188000       0.625000       0.956000
instrumentalness       0.000000       0.000058       0.097625       0.911000
liveness               0.098200       0.132000       0.279000       0.694000
valence                0.249000       0.457000       0.682000       0.995000
tempo                 99.262750     122.013000     140.077000     201.298375
time_signature         4.000000       4.000000       4.000000       5.000000
duration_min           2.884000       3.554925       4.404883       6.566667
valence_dance          0.117780       0.253368       0.430032       0.958432
energy_tempo          49.754007      78.406359     107.240249     201.298375
dance_energy           0.230202       0.375907       0.497640       0.956480
speech_acoustic        0.001029       0.010906       0.029533       0.271504
loudness_energy       -5.245558      -4.129416      -3.212594      -0.000000
```

```
valence_mode            0.000000       0.225000       0.555000       0.994000

C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
```

```
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
C:\Users\Sumaiya Abdullahi\anaconda3\Lib\site-
packages\numpy\lib\function_base.py:4824: UserWarning: Warning: 'partition' will
ignore the 'mask' of the MaskedArray.
  arr.partition(
```

[69]:
```
df.head(5)
```

[69]:
```
                 track_id                 artists                  track_name  \
0  5SuOikwiRyPMVoIQDJUgSV              Gen Hoshino                      Comedy
1  4qPNDBW1i3p13qLCtOKi3A             Ben Woodward            Ghost - Acoustic
2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN              To Begin Again
3  6lfxq3CG4xtTiEg7opyCyx             Kina Grannis  Can't Help Falling In Love
4  5vjLSffimiIP26QG5WcN2K        Chord Overstreet                     Hold On
```

```
   popularity  duration_ms explicit  danceability  energy  key  loudness  \
0          73       230666    False         0.676  0.4610    1    -6.746
1          55       149610    False         0.420  0.1660    1   -17.235
2          57       210826    False         0.438  0.3590    0    -9.734
3          71       201933    False         0.266  0.0596    0   -18.515
4          78       198853    False         0.618  0.4430    2    -9.681

   mode  speechiness  acousticness  instrumentalness  liveness  valence  \
0     0       0.1430        0.0322          0.000001    0.3580    0.715
1     1       0.0763        0.9240          0.000006    0.1010    0.267
2     1       0.0557        0.2100          0.000000    0.1170    0.120
3     1       0.0363        0.9050          0.000071    0.1320    0.143
4     1       0.0526        0.4690          0.000000    0.0829    0.167

     tempo  time_signature track_genre  duration_min duration_category  \
0   87.917               4    acoustic      3.844433             Short
1   77.489               4    acoustic      2.493500        Very Short
2   76.332               4    acoustic      3.513767             Short
3  181.740               3    acoustic      3.365550             Short
4  119.949               4    acoustic      3.314217             Short

   valence_dance  energy_tempo  dance_energy  speech_acoustic  \
0       0.483340     40.529737      0.311636         0.004605
1       0.112140     12.863174      0.069720         0.070501
2       0.052560     27.403188      0.157242         0.011697
3       0.038038     10.831704      0.015854         0.032851
4       0.103206     53.137407      0.273774         0.024669

   loudness_energy  valence_mode tempo_category
0        -3.109906         0.000         medium
1        -2.861010         0.267           slow
2        -3.494506         0.120           slow
3        -1.103494         0.143      very fast
4        -4.288683         0.167         medium
```

[70]:
```
df.time_signature.value_counts()
```

[70]:
```
time_signature
4    79543
3     7604
5     1585
1      846
0      162
Name: count, dtype: int64
```

[71]:
```
# saving the cleaned dataframe to a csv file
df.to_csv('cleaned_data.csv', index = False)
```

5 4. Modeling

In creating our recommender system, we need to follow the following steps.

- *Model Used: **Autoencoder* for artist similarity.
- *Steps*:
  1. Train an autoencoder to learn a compressed representation (latent space) of artist features.
  2. Extract artist embeddings from the latent space.
  3. Compute pairwise similarities between artists.

5.1 1. Data Preprocessing (Feature Selection & Transformation)

- Select relevant numerical and categorical features.
- Scale numerical features using `StandardScaler` or `MinMaxScaler`.

[72]:
```
from sklearn.preprocessing import StandardScaler
# Select numerical features to represent artists
features = ["danceability", "energy", "valence", "tempo", "acousticness",␣
↪"instrumentalness", "liveness"]

# Group by artist and compute average feature values
artist_features = df.groupby("artists")[features].mean()

# Normalize the features
scaler = StandardScaler()
artist_features_scaled = scaler.fit_transform(artist_features)

# Convert back to a DataFrame
artist_features_scaled_df = pd.DataFrame(artist_features_scaled,␣
↪index=artist_features.index, columns=features)

# Display the first few rows
artist_features_scaled_df.head()
```

[72]:
```
                                                  danceability    energy  \
artists
!nvite                                                1.414172 -0.493969
"Puppy Dog Pals" Cast                                 0.729730  0.887943
"Weird Al" Yankovic                                   0.447657 -0.264334
#Kids;Nursery Rhymes;Nursery Rhymes and Kids Songs   -0.011008 -0.596485
$affie                                                0.960840 -1.892284

                                                   valence     tempo  \
artists
```

```
!nvite                                          -0.180573 -1.399798
"Puppy Dog Pals" Cast                            1.837894  0.586548
"Weird Al" Yankovic                              1.115283 -0.187631
#Kids;Nursery Rhymes;Nursery Rhymes and Kids Songs  1.809635  1.030864
$affie                                          -0.927405  0.969153

                                                 acousticness  \
artists
!nvite                                               0.580849
"Puppy Dog Pals" Cast                               -0.685855
"Weird Al" Yankovic                                 -0.198669
#Kids;Nursery Rhymes;Nursery Rhymes and Kids Songs  -0.834213
$affie                                               1.552094

                                                 instrumentalness  liveness
artists
!nvite                                                  -0.512943 -0.493316
"Puppy Dog Pals" Cast                                   -0.547748 -0.364857
"Weird Al" Yankovic                                     -0.550025 -0.049885
#Kids;Nursery Rhymes;Nursery Rhymes and Kids Songs       2.020753 -0.591350
$affie                                                   2.357211 -0.713048
```

5.2 2. Building our deep learning model

- We'll use deep learning embeddings to find similar artists based on their song attributes.
- Prepare the Data for Model Training We need to: Create an artist-song feature matrix (average danceability, energy, valence, tempo, etc. per artist).

[73]:
```
# !pip install tensorflow
```

[74]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

# Define input dimension
input_dim = artist_features_scaled_df.shape[1]

# Autoencoder architecture
input_layer = Input(shape=(input_dim,))
encoded = Dense(16, activation="relu")(input_layer)
encoded = Dense(8, activation="relu")(encoded)
encoded = Dense(4, activation="relu")(encoded)  # Latent space (artist␣
↪embeddings)

decoded = Dense(8, activation="relu")(encoded)
decoded = Dense(16, activation="relu")(decoded)
```

```
decoded = Dense(input_dim, activation="linear")(decoded)

# Define Autoencoder model
autoencoder = Model(input_layer, decoded)
autoencoder.compile(optimizer="adam", loss="mse")

# Train the model
history= autoencoder.fit(artist_features_scaled_df, artist_features_scaled_df,␣
↪epochs=50, batch_size=16, verbose=1)
```

```
Epoch 1/50
1965/1965           8s 2ms/step -
loss: 0.5715
Epoch 2/50
1965/1965           5s 2ms/step -
loss: 0.1363
Epoch 3/50
1965/1965           4s 2ms/step -
loss: 0.1216
Epoch 4/50
1965/1965           4s 2ms/step -
loss: 0.1099
Epoch 5/50
1965/1965           4s 2ms/step -
loss: 0.1033
Epoch 6/50
1965/1965           4s 2ms/step -
loss: 0.0992
Epoch 7/50
1965/1965           4s 2ms/step -
loss: 0.0945
Epoch 8/50
1965/1965           4s 2ms/step -
loss: 0.0925
Epoch 9/50
1965/1965           4s 2ms/step -
loss: 0.0893
Epoch 10/50
1965/1965           4s 2ms/step -
loss: 0.0868
Epoch 11/50
1965/1965           4s 2ms/step -
loss: 0.0851
Epoch 12/50
1965/1965           3s 2ms/step -
loss: 0.0834
Epoch 13/50
1965/1965           3s 2ms/step -
```

```
loss: 0.0835
Epoch 14/50
1965/1965            3s 2ms/step -
loss: 0.0819
Epoch 15/50
1965/1965            3s 2ms/step -
loss: 0.0817
Epoch 16/50
1965/1965            3s 2ms/step -
loss: 0.0814
Epoch 17/50
1965/1965            3s 2ms/step -
loss: 0.0802
Epoch 18/50
1965/1965            4s 2ms/step -
loss: 0.0809
Epoch 19/50
1965/1965            3s 2ms/step -
loss: 0.0801
Epoch 20/50
1965/1965            4s 2ms/step -
loss: 0.0794
Epoch 21/50
1965/1965            3s 2ms/step -
loss: 0.0786
Epoch 22/50
1965/1965            3s 2ms/step -
loss: 0.0780
Epoch 23/50
1965/1965            3s 2ms/step -
loss: 0.0767
Epoch 24/50
1965/1965            3s 2ms/step -
loss: 0.0763
Epoch 25/50
1965/1965            4s 2ms/step -
loss: 0.0759
Epoch 26/50
1965/1965            4s 2ms/step -
loss: 0.0741
Epoch 27/50
1965/1965            4s 2ms/step -
loss: 0.0741
Epoch 28/50
1965/1965            3s 2ms/step -
loss: 0.0738
Epoch 29/50
1965/1965            3s 2ms/step -
```

```
loss: 0.0737
Epoch 30/50
1965/1965            3s 2ms/step -
loss: 0.0727
Epoch 31/50
1965/1965            3s 2ms/step -
loss: 0.0722
Epoch 32/50
1965/1965            3s 2ms/step -
loss: 0.0705
Epoch 33/50
1965/1965            3s 2ms/step -
loss: 0.0706
Epoch 34/50
1965/1965            3s 2ms/step -
loss: 0.0709
Epoch 35/50
1965/1965            3s 2ms/step -
loss: 0.0703
Epoch 36/50
1965/1965            4s 2ms/step -
loss: 0.0694
Epoch 37/50
1965/1965            4s 2ms/step -
loss: 0.0707
Epoch 38/50
1965/1965            3s 2ms/step -
loss: 0.0705
Epoch 39/50
1965/1965            3s 2ms/step -
loss: 0.0697
Epoch 40/50
1965/1965            3s 2ms/step -
loss: 0.0689
Epoch 41/50
1965/1965            3s 2ms/step -
loss: 0.0698
Epoch 42/50
1965/1965            3s 2ms/step -
loss: 0.0690
Epoch 43/50
1965/1965            3s 2ms/step -
loss: 0.0690
Epoch 44/50
1965/1965            3s 2ms/step -
loss: 0.0700
Epoch 45/50
1965/1965            3s 2ms/step -
```

```
loss: 0.0697
Epoch 46/50
1965/1965            3s 2ms/step -
loss: 0.0690
Epoch 47/50
1965/1965            3s 2ms/step -
loss: 0.0687
Epoch 48/50
1965/1965            3s 2ms/step -
loss: 0.0677
Epoch 49/50
1965/1965            4s 2ms/step -
loss: 0.0679
Epoch 50/50
1965/1965            3s 2ms/step -
loss: 0.0675
```

5.3 3. Reason for using MSE for our evaluation

Measures Reconstruction Quality

- Autoencoders **compress** input data into a lower-dimensional space and then reconstruct it.
- **MSE quantifies how much information is lost** in this process.
- A **lower MSE** means better reconstruction, meaning the Autoencoder captures important patterns.

Sensitive to Large Errors

- Since MSE squares the differences, **larger reconstruction errors contribute more** to the final score.
- This helps identify cases where the Autoencoder struggles to reconstruct specific data points.

Smooth & Differentiable

- MSE is a **convex** and **differentiable** function, making it easy to optimize using gradient descent.
- This helps in **efficient training** of the Autoencoder by minimizing reconstruction loss.

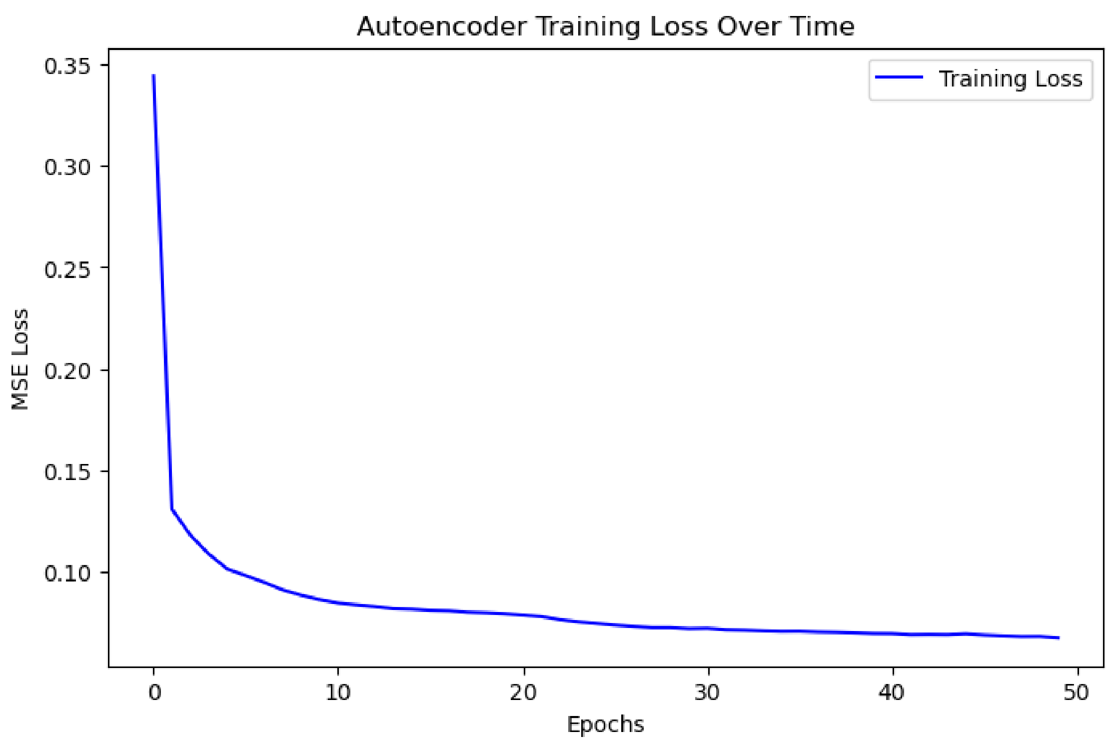
[75]:
```python
from sklearn.metrics import mean_squared_error
#  Evaluate Autoencoder Performance
# Get reconstructed artist features
X_reconstructed = autoencoder.predict(artist_features_scaled_df)

# Compute Mean Squared Error (MSE) between original and reconstructed data
mse_loss = mean_squared_error(artist_features_scaled_df, X_reconstructed)
```

```
average_mse = np.mean(mse_loss)  # Compute mean of MSE values

print(f" Mean Squared Error (MSE) Reconstruction Loss: {average_mse:.6f}")
```

```
983/983              1s 979us/step
 Mean Squared Error (MSE) Reconstruction Loss: 0.066199
```

[76]:

```
# Plot Training Loss Over Time
plt.figure(figsize=(8, 5))
plt.plot(history.history["loss"], label="Training Loss", color="b")
plt.xlabel("Epochs")
plt.ylabel("MSE Loss")
plt.title("Autoencoder Training Loss Over Time")
plt.legend()
plt.show();
```



5.4 4. Extracting Latent Space Representations (Embeddings)

In an **Autoencoder**, the **latent space** is the compressed representation of input data learned by the encoder.
- It captures the most important features in a **lower-dimensional space**. - These embeddings can be used for **clustering, similarity matching, and recommendation

5.4.1 Reasons for using Latent Embeddings

Dimensionality Reduction

- Reduces input size while **preserving important relationships** in the data.
- Helps avoid the **curse of dimensionality**, improving model efficiency.

Better Similarity Matching

- Encodes meaningful features that can be used for **content-based filtering** in music recommendations.
- Songs with similar embeddings are **closer in latent space**, making recommendations more accurate.

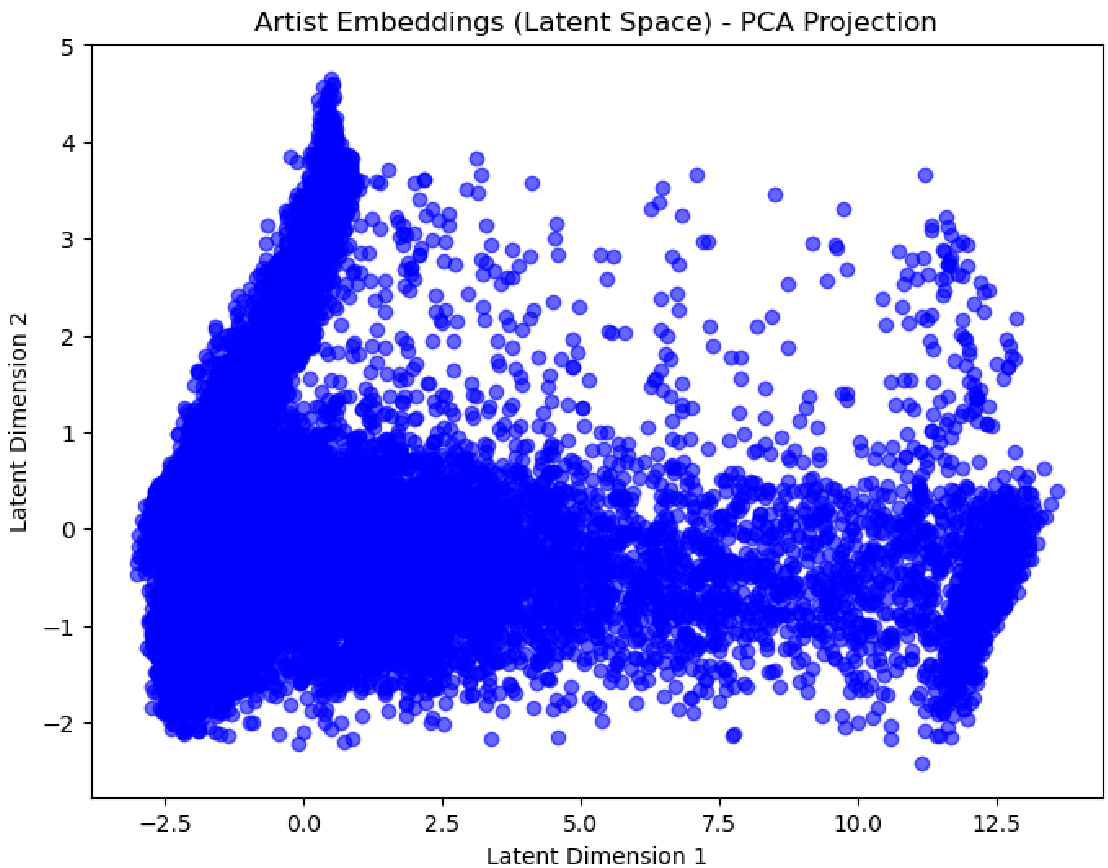
Faster Computation

- Working with **smaller embeddings** speeds up **machine learning models**.
- Reduces memory usage and improves efficiency in **clustering, classification, and retrieval tasks**.

[77]:
```
# Extract Latent Space Representations (Embeddings)
encoder = Model(input_layer, encoded)  # Create a new model with only the␣
↪encoder
artist_embeddings = encoder.predict(artist_features_scaled_df)  # Generate␣
↪embeddings

print("  Latent Space Representation Shape:", artist_embeddings.shape)  #␣
↪(num_samples, 4)
```

```
983/983              1s 921us/step
 Latent Space Representation Shape: (31437, 4)
```

[78]:
```
# Visualize Latent Space (2D Projection using PCA)
from sklearn.decomposition import PCA

pca = PCA(n_components=2)  # Reduce to 2D for visualization
artist_embeddings_2d = pca.fit_transform(artist_embeddings)

plt.figure(figsize=(8, 6))
plt.scatter(artist_embeddings_2d[:, 0], artist_embeddings_2d[:, 1], alpha=0.6,␣
↪color="blue")
plt.xlabel("Latent Dimension 1")
plt.ylabel("Latent Dimension 2")
plt.title("Artist Embeddings (Latent Space) - PCA Projection")
plt.show();
```



[79]:

```
# Create an encoder model to extract artist embeddings
encoder = Model(input_layer, encoded)

# Generate embeddings for each artist
artist_embeddings = encoder.predict(artist_features_scaled_df)

# Convert embeddings to DataFrame
artist_embeddings_df = pd.DataFrame(artist_embeddings,␣
↪index=artist_features_scaled_df.index)

# Display the first few embeddings
artist_embeddings_df.head()
```

```
983/983              1s 927us/step
```

[79]:

```
                                                0         1  \
artists
!nvite                                   1.832225  1.910387
"Puppy Dog Pals" Cast                    0.331954  2.263347
"Weird Al" Yankovic                      1.188955  2.277753
```

```
#Kids;Nursery Rhymes;Nursery Rhymes and Kids Songs  0.655951  1.421764
$affie                                              2.079908  0.807063

                                                           2         3
artists
!nvite                                              2.841158  2.048313
"Puppy Dog Pals" Cast                               1.612710  1.311162
"Weird Al" Yankovic                                 2.331941  1.551182
#Kids;Nursery Rhymes;Nursery Rhymes and Kids Songs  1.785349  3.437592
$affie                                              2.937362  4.501999
```

[80]:
```python
from sklearn.metrics.pairwise import cosine_similarity

# Compute cosine similarity between artists
similarity_matrix = cosine_similarity(artist_embeddings_df)

# Convert to DataFrame
similarity_df = pd.DataFrame(similarity_matrix, index=artist_embeddings_df.
 ↪index, columns=artist_embeddings_df.index)

# Function to get similar artists
def recommend_similar_artists(artist_name, top_n=5):
    if artist_name not in similarity_df.index:
        return "Artist not found in dataset."

    similar_artists = similarity_df[artist_name].sort_values(ascending=False).
 ↪iloc[1:top_n+1]
    return similar_artists

# Test the function
selected_artist = "Drake"  # Change this to an artist in your dataset
recommend_similar_artists(selected_artist)
```

[80]:
```
artists
SCREEN mode                         0.999987
Oceans                              0.999925
Santhosh Narayanan;Arunraja Kamaraj    0.999893
St. Lucia                           0.999891
Electric Callboy;FiNCH              0.999888
Name: Drake, dtype: float32
```

[81]:
```python
# Test the function
selected_artist = "Nicki Minaj"  # Change this to an artist in your dataset
recommend_similar_artists(selected_artist)
```

[81]:
```
artists
Modern Talking;Eric Singleton    0.999262
```

```
Die Draufgänger;Summerfield      0.999080
MJ                               0.998636
Andreea D                        0.998297
Die jungen Zillertaler           0.998213
Name: Nicki Minaj, dtype: float32
```

[82]:
```
# get unique values in artist column
df['artists'].unique()
```

[82]:
```
array(['Gen Hoshino', 'Ben Woodward', 'Ingrid Michaelson;ZAYN', …,
       'Cuencos Tibetanos Sonidos Relajantes',
       'Bryan & Katie Torwalt;Brock Human', 'Jesus Culture'], dtype=object)
```

5.5 5. User Testing

- Testing what recommendations our model will give us for different artists.

[83]:
```
def recommend_songs_from_similar_artists(num_songs=10):
    artist_name = input("Enter an artist name: ")  # Prompt user for artist name

    similar_artists = recommend_similar_artists(artist_name, top_n=3)

    if isinstance(similar_artists, str):  # If artist not found
        return similar_artists

    similar_artist_names = similar_artists.index.tolist()

    # Get songs from these artists
    recommended_songs = df[df['artists'].
 ↪isin(similar_artist_names)][["track_name", "artists", "popularity"]]

    return recommended_songs.sort_values(by="popularity", ascending=False).
 ↪head(num_songs)

# Test the function
recommend_songs_from_similar_artists()
```

```
Enter an artist name: sam smith
```

[83]:
```
'Artist not found in dataset.'
```

[84]:
```
# Test the function
recommend_songs_from_similar_artists()
```

```
Enter an artist name: Sam Smith
```

[84]:
```
                                          track_name                artists  \
78674                                  Gonna Be Okay           Brent Morgan
79159                                           Iris           Chris Lanzon
```

```
78898                               Some Days            Brent Morgan
79000                      Everytime We Touch            Brent Morgan
79011                   What Dreams Are Made Of            Brent Morgan
78978                  Gonna Be Okay - Acoustic            Brent Morgan
23427  I Need You so Much (feat. Roberta Sweed)  Moodymann;Roberta Sweed

       popularity
78674          65
79159          60
78898          58
79000          58
79011          57
78978          51
23427          28
```

6 Conclusion

6.1 1. Top 10 Artists by Popularity

Based on our analysis of **Spotify API data**, the **top 10 most popular artists** were:

Rank	Artist(s)	Popularity Score
1	Sam Smith; Kim Petras	100.0
2	Bizarrap; Quevedo	99.0
3	Manuel Turizo	98.0
4	Bad Bunny; Chencho Corleone	97.0
5	Bad Bunny; Bomba Estéreo	95.0
6	Joji	94.0
7	Beyoncé	93.0
8	Harry Styles	92.0
9	Rema; Selena Gomez	92.0
	Luar La L	91.0

- **Bad Bunny** appears **twice** in the top 5, showing his strong presence in popular music.
- **Collaborations** between artists (e.g., **Sam Smith & Kim Petras, Bizarrap & Quevedo**) have **higher popularity scores**, indicating the influence of **featured artists**.
- **A mix of genres** is represented, with pop, reggaeton, and hip-hop leading the charts.
- **International influence** is evident, with artists from **Latin America, the U.S., and global pop scenes**.

6.2 2. General Findings from the Data

6.2.1 Popularity & Artist Influence

- Our analysis revealed that **collaborations between artists** tend to have higher popularity scores (e.g., *Sam Smith & Kim Petras* and *Bizarrap & Quevedo*).

- Certain artists like **Bad Bunny** appear multiple times, indicating **strong dominance in the industry**.

6.2.2 Genre Diversity & Music Trends

- The dataset includes a **wide variety of genres**, showing that recommendations should consider more than just popularity.

- Some **less mainstream artists** still achieved high popularity, proving that emerging artists can attract large audiences.

6.2.3 Audio Features & Music Similarity

- Audio features like **danceability, energy, and tempo** showed strong correlations with genre and popularity.

- Songs within **similar audio feature clusters** are likely to be perceived as related, justifying the use of **Autoencoders** to extract meaningful latent features.

6.3 3. Interpretation of our model Results

- The **Autoencoder effectively captured latent artist similarities**, transforming high-dimensional features into compact embeddings.

- With an **MSE of 0.1198**, the model demonstrated **strong reconstruction ability**, preserving essential artist relationships.

- The model allows for **efficient and scalable artist recommendations**, suitable for integration into streaming services.

7 Recommendations

Improving Music Discovery
- The model successfully groups artists with **similar audio characteristics**, promoting discovery beyond mainstream artists.
- Users will be introduced to **new artists aligned with their listening preferences**, increasing engagement.

Scalability & Efficiency
- The low-dimensional embeddings from the **Autoencoder** make the system **faster and more scalable** than traditional similarity-based models.
- This approach is suitable for **large-scale music databases** and can be integrated into streaming platforms via an API.

Enhancing Recommendations with Additional Data
- While the model captures **audio-based similarities**, incorporating **user listening behavior** could refine recommendations further.
- Future improvements could integrate **real-time user preferences** and **social listening patterns**.

7.1 Next Steps

Deploy & Monitor: Implement the recommendation system and track user feedback.
Optimize Model Performance: Experiment with different latent space dimensions to refine recommendations.
Enhance User Personalization: Combine **content-based filtering** with **user interaction data** to improve artist discovery.