Analysis of Spotify tracks

1 Project Setup and Data Acquisition

In [30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Why it matters:

These tools let us read the Spotify data, crunch numbers, and plot charts.

Next steps:

- Read the CSV into `spotify_df`
- Peek at `spotify_df.head()`
- Start cleaning or exploring features

---

2 Data Loading and Initial Exploration

a) Preview the first few rows to see what the data looks like:

2.a–b Data Loading & Initial Exploration

Goal:

Load the SpotifyTop50CSV, verify that rows and columns are intact, and get an initial look at data types and sample values.

In [31]:
```
data_path = "../data/spotifytoptracks.csv"
spotify_df = pd.read_csv(data_path)
spotify_df.head()
```

Out[31]:

	Unnamed: 0	artist	album	track_name	track_id	energ
0	0	The Weeknd	After Hours	Blinding Lights	0VjIjW4GlUZAMYd2vXMi3b	0.7
1	1	Tones And I	Dance Monkey	Dance Monkey	1rgnBhdG2JDFTbYkYRZAku	0.5
2	2	Roddy Ricch	Please Excuse Me For Being Antisocial	The Box	0nbXyq5TXYPCO7pr3N8S4I	0.5
3	3	SAINt JHN	Roses (Imanbek Remix)	Roses - Imanbek Remix	2Wo6QQD1KMDWeFkkjLqwx5	0.7
4	4	Dua Lipa	Future Nostalgia	Don't Start Now	3PfIrDoz19wz7qK7tYeu62	0.7

What I did

- Loaded the CSV into `spotify_df` with `pd.read_csv(data_path)`.
- Called `spotify_df.head()` to preview the first five rows.
- Visually scanned the output to confirm column names, their order, and example values.
- Noticed an extra index column (`Unnamed: 0`) was imported alongside the real features.

b) This shows the final rows to quickly verify the dataset's end state:

In [32]:
```
spotify_df = spotify_df.drop(columns=['Unnamed: 0'], errors='ignore')
spotify_df.tail()
```

Out[32]:

	artist	album	track_name	track_id	energy	danc
45	Juice WRLD	Goodbye & Good Riddance	Lucid Dreams	285pBltuF7vW8TeWk8hdRR	0.566	
46	Ariana Grande	Stuck with U	Stuck with U (with Justin Bieber)	4HBZA5flZLE435QTztThqH	0.450	
47	JP Saxe	If the World Was Ending (feat. Julia Michaels)	If the World Was Ending - feat. Julia Michaels	2kJwzbxV2ppxnQoYw4GLBZ	0.473	
48	Dua Lipa	Future Nostalgia	Physical	3AzjcOeAmA57TIOr9zF1ZW	0.844	
49	Travis Scott	ASTROWORLD	SICKO MODE	2xLMifQCjDGFmkHkpNLD9h	0.730	

2.a–b Data Loading & Initial Exploration

What I did

- Removed the extra index column with `spotify_df = spotify_df.drop(columns=['Unnamed: 0'], errors='ignore')`.
- Called `spotify_df.tail()` to preview the last five rows of the cleaned DataFrame.

Results

- The final five entries display only the real features (`artist`, `album`, `track_name`, `track_id`, `energy`, …, `genre`).
- No `Unnamed: 0` column remains, and all 50 tracks are still present.

What this means
The DataFrame now contains exactly the 16 intended Spotify features plus no extras.

Why it matters:

- Verifies there were no import errors, dropped rows, or misalignments before we move on.
- Provides a solid foundation for cleaning, validation, and deeper EDA.

Possible next steps:

- Check for duplicate or missing rows.

- Run `spotify_df.describe()` on numeric columns to get summary statistics.
- Begin feature-level cleaning or simple exploratory statistics.

---

c) Display summary statistics for numeric columns:

2.c Summary Statistics for Numeric Features

Goal:
Quickly review each numeric column’s basic stats.

In [33]:
```
spotify_df.describe()
```

Out[33]:

	energy	danceability	key	loudness	acousticness	speechine
count	50.000000	50.000000	50.000000	50.000000	50.000000	50.0000
mean	0.609300	0.716720	5.720000	-6.225900	0.256206	0.1241
std	0.154348	0.124975	3.709007	2.349744	0.265250	0.1168
min	0.225000	0.351000	0.000000	-14.454000	0.001460	0.0290
25%	0.494000	0.672500	2.000000	-7.552500	0.052800	0.0483
50%	0.597000	0.746000	6.500000	-5.991500	0.188500	0.0700
75%	0.729750	0.794500	8.750000	-4.285500	0.298750	0.1555
max	0.855000	0.935000	11.000000	-3.280000	0.934000	0.4870

What I did:

- Called `spotify_df.describe()` to get count, mean, std, min/max and quartiles.

2.c Summary Statistics for Numeric Features

Results:

- Every numeric column has 50 values (no missing data).
- Energy ranges from 0.23 to 0.85.
- Danceability spans 0.35 to 0.94.
- Loudness goes from -14.45 dB to -3.28 dB.
- Tempo varies between 76 BPM and 180 BPM.
- Duration runs from 140 526 ms to 312 820 ms.

What this means:
The features display moderate spread and no obvious “weird” ranges.

Why it matters:

- Lets you choose the right chart for each feature.
- Shows if any numbers are skewed and need fixing.

Possible next steps:

- Make simple histograms or boxplots for each feature.
- Move on to checking how features relate to each other.
- Run `spotify_df.info()` to confirm correct dtypes and non-null counts.

---

d) Inspect summary information about the Dataframe (data types and non-null counts):

2.d Inspect DataFrame Summary

Goal:
Quickly view each column’s data type and non-null count to verify structure and completeness.

In [34]:
```
spotify_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   artist            50 non-null     object 
 1   album             50 non-null     object 
 2   track_name        50 non-null     object 
 3   track_id          50 non-null     object 
 4   energy            50 non-null     float64
 5   danceability      50 non-null     float64
 6   key               50 non-null     int64  
 7   loudness          50 non-null     float64
 8   acousticness      50 non-null     float64
 9   speechiness       50 non-null     float64
 10  instrumentalness  50 non-null     float64
 11  liveness          50 non-null     float64
 12  valence           50 non-null     float64
 13  tempo             50 non-null     float64
 14  duration_ms       50 non-null     int64  
 15  genre             50 non-null     object 
dtypes: float64(9), int64(2), object(5)
memory usage: 6.4+ KB
```

What I did:

- Ran `spotify_df.info()` to display the schema, non-null counts, and memory usage of the DataFrame.

2.d Inspect DataFrame Summary

Results:

- **Entries:** 50 (index 0–49)
- **Columns:** 16 total
- **Non-null count:** 50 for every column (no missing values)
- **Dtypes:** 9× `float64`, 3× `int64`, 5× `object`
- **Memory usage:** ~6.8 KB

What this means:
All 16 features are fully populated and have the expected types (numeric metrics as floats/ints, metadata as objects).

Why it matters:

- Guarantees our analyses won't break on missing data.
- Confirms columns are correctly typed for calculations or visualizations.

Possible next steps:

- Begin deeper EDA now that the data schema is confirmed.

3 Data cleaning

a) Check for missing values:

3.a Check for Missing Values

Goal:
Determine whether any features in the dataset contain null (missing) entries.

In [35]:
```
spotify_df.isnull().sum()
```

Out[35]:
```
artist              0
album               0
track_name          0
track_id            0
energy              0
danceability        0
key                 0
loudness            0
acousticness        0
speechiness         0
instrumentalness    0
liveness            0
valence             0
tempo               0
duration_ms         0
genre               0
dtype: int64
```

What I did:

- Used `spotify_df.isnull().sum()` to count the number of nulls in each column.

3.a Check for Missing Values

Results:

- Every column reports 0 missing values.

What this means:
All 50 tracks have data for every feature, so our analyses use the complete dataset.

Why it matters:

- Guarantees our analysis won't be skewed by unexpected nulls.
- Eliminates delays from data-imputation, so finance and analytics teams can deliver reports on time.
- Builds trust with labels and artists by showing Spotify's systems capture every stream.

Possible next step:

- Proceed to duplicate-row checks or outlier detection.

---

b) Show number of duplicate rows:

3.b Duplicate Row Check

Goal:
Find out how many exact duplicate rows are in the dataset.

In [36]:
```
num_duplicates = spotify_df.duplicated().sum()
print("Number of duplicate rows:", num_duplicates)
```

```
Number of duplicate rows: 0
```

What I did:

- Used `spotify_df.duplicated().sum()` to count any fully identical rows.
- Printed the resulting number.

3.b Duplicate Row Check

Results:

- Number of duplicate rows: 0

What this means:
No track appears more than once in the dataset.

Why it matters:

- Ensures Top 50 rankings reflect unique listener events, not inflated counts.
- Protects Spotify from over-issuing royalties by avoiding double counting of streams.

Possible next steps:

- Verify uniqueness of key columns like `track_id` or `track_name`.
- Continue with analysis knowing each row represents a distinct song.

---

c) Treating Outliers:

3.c Treating Outliers

Goal:
Identify extreme values in each numeric feature using the IQR rule (1.5 × IQR beyond Q1/Q3).

In [37]:
```
"""
Cell 1: compute IQR bounds and print outlier counts
```

```
"""
num_cols = spotify_df.select_dtypes(include=np.number).columns.tolist()

bounds = {}

for col in num_cols:
    q1 = spotify_df[col].quantile(0.25)
    q3 = spotify_df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    higher_bound = q3 + 1.5 * iqr

    bounds[col] = (lower_bound, higher_bound)

    out = spotify_df[
        (spotify_df[col] < lower_bound)
        | (spotify_df[col] > higher_bound)
    ]
    print(col, "outliers:", len(out))
```

```
energy outliers: 0
danceability outliers: 3
key outliers: 0
loudness outliers: 1
acousticness outliers: 7
speechiness outliers: 6
instrumentalness outliers: 12
liveness outliers: 3
valence outliers: 0
tempo outliers: 0
duration_ms outliers: 2
```

What I did:
Cell 1:

- Used `spotify_df.select_dtypes(include=np.number)` to grab all numeric columns.
- For each column:
  - Calculated Q1 (25th percentile) and Q3 (75th percentile).
  - Computed IQR = Q3 − Q1.
  - Defined `lower_bound = Q1 − 1.5×IQR` and `higher_bound = Q3 + 1.5×IQR`, stored these in a `bounds` dict.
- Filtered the DataFrame to find any values below `lower_bound` or above `higher_bound`.
- Printed the count of outliers for each numeric feature.

In [38]:
```
"""
Cell 2: build a detailed outlier table
"""

outlier_records = []
for feature_name, (lower_bound, higher_bound) in bounds.items():
    for idx, val in spotify_df[feature_name].items():
```

```
        if val < lower_bound or val > higher_bound:
            outlier_records.append({
                'artist'    : spotify_df.at[idx, 'artist'],
                'track_name': spotify_df.at[idx, 'track_name'],
                'feature'   : feature_name
            })

outliers_df = (
    pd.DataFrame(outlier_records)
    .drop_duplicates(subset=['artist','track_name','feature'])
    .reset_index(drop=True)
)
outliers_df.index += 1
outliers_df
```

Out[38]:

	artist	track_name	feature
1	Lewis Capaldi	Before You Go	danceability
2	Billie Eilish	lovely (with Khalid)	danceability
3	JP Saxe	If the World Was Ending - feat. Julia Michaels	danceability
4	Billie Eilish	everything i wanted	loudness
5	Tones And I	Dance Monkey	acousticness
6	Powfu	death bed (coffee for your head)	acousticness
7	Lewis Capaldi	Someone You Loved	acousticness
8	Maroon 5	Memories	acousticness
9	Billie Eilish	everything i wanted	acousticness
10	Billie Eilish	lovely (with Khalid)	acousticness
11	JP Saxe	If the World Was Ending - feat. Julia Michaels	acousticness
12	Future	Life Is Good (feat. Drake)	speechiness
13	Billie Eilish	bad guy	speechiness
14	Cardi B	WAP (feat. Megan Thee Stallion)	speechiness
15	Eminem	Godzilla (feat. Juice WRLD)	speechiness
16	Maluma	Hawái	speechiness
17	Bad Bunny	Safaera	speechiness
18	The Weeknd	Blinding Lights	instrumentalness
19	Tones And I	Dance Monkey	instrumentalness
20	SAINt JHN	Roses - Imanbek Remix	instrumentalness
21	KAROL G	Tusa	instrumentalness
22	Post Malone	Circles	instrumentalness
23	Billie Eilish	everything i wanted	instrumentalness
24	Billie Eilish	bad guy	instrumentalness
25	BENEE	Supalonely (feat. Gus Dapperton)	instrumentalness
26	Surf Mesa	ily (i love you baby) (feat. Emilee)	instrumentalness
27	Regard	Ride It	instrumentalness
28	Black Eyed Peas	RITMO (Bad Boys For Life)	instrumentalness
29	Dua Lipa	Physical	instrumentalness
30	Roddy Ricch	The Box	liveness
31	Powfu	death bed (coffee for your head)	liveness

	artist	track_name	feature
32	Black Eyed Peas	RITMO (Bad Boys For Life)	liveness
33	Bad Bunny	Safaera	duration_ms
34	Travis Scott	SICKO MODE	duration_ms

What I did:

- **Cell 2:**
  - Initialized an empty `records` list.
  - Iterated over each `(col, (lower_bound, higher_bound))` in `bounds.items()`.
    - For each feature, found the row indices with values below `lo` (low outliers) or above `higher_bound` (high outliers) and appended a dict of `{artist, track_name, feature}`.
  - Converted `records` into a DataFrame, dropped any duplicate artist/track/feature rows, reset to a 1-based index, yielding `outliers_df`, which lists every track/feature pair flagged as an outlier.

> **Why two separate cells?**
> First calculate and check your outlier thresholds, then build the detailed table once you've confirmed those bounds.

3.c Treating Outliers

Results:

- Identified 34 outlier entries across six audio features (danceability, loudness, acousticness, speechiness, instrumentalness, liveness, and duration_ms).
- These outliers are genuine extremes that won't skew our basic summary statistics, so we'll leave them as is.

What this means:
A clear list of unusually extreme feature values identifies 34 track-feature outliers.

Why it matters:

- Outliers can mess up averages and correlations.
- Having a clear outlier table-built on our clean, gap-free dataset-lets us decide to remove, cap, or otherwise handle these extreme values before further analysis and reliably spot songs that deviate from the norm.

- Allows marketing to spotlight niche hits (e.g. ultra-acoustic or high-speechiness).

Possible next steps:

- Plot each feature's distribution with and without the outliers.
- Recalculate summary stats (means, correlations) to see how much the outliers were affecting them.

---

4 Exploratory Data Analysis

4.1. Basic Stats & Structure

a) Basic Stats & Structure (Observations & Features):

How many observations are there in this dataset?

How many features this dataset has?

4.1.a Basic Stats & Structure

Goal:
Determine how many observations (rows) and features (columns) are in our dataset.

In [39]:
```
num_tracks, num_attributes = spotify_df.shape

print(f"Total tracks (observations): {num_tracks}")
print(f"Total attributes (features):   {num_attributes}")
```

```
Total tracks (observations): 50
Total attributes (features):   16
```

What I did:

- Used `spotify_df.shape` to get the number of observations and features.
- Assigned the results to `num_tracks` and `num_attributes`.

4.1.a Basic Stats & Structure

Results:

- **Total observations:** 50

- **Total features:** 16

What this means:
We're analyzing exactly 50 songs with 16 pieces of information each, so everyone knows the project size.

Why it matters:

- Keeps us aware of any extra columns that could skew analyses if not removed or ignored.
- Clarifies project scope, so stakeholders know the analysis boundary.
- Enables quick turnaround by a small team, keeping costs low.
- Provides a solid basis for scaling to larger datasets with predictable effort.

Possible next steps:

- Inspect your feature lists.
- Visualize distributions with histograms or boxplots.

---

b) Basic Stats & Structure (Categorical & Numeric):

Which of the features are categorical?

Which of the features are numeric?

4.1.b Feature Types Overview

Goal:
Identify which columns in the raw dataset are categorical (text/object) vs. numeric.

In [40]:
```
cat_cols = spotify_df.select_dtypes(include=['object']).columns.tolist()

print("Categorical features:", cat_cols)
print("Numeric features:    ", num_cols)
```

```
Categorical features: ['artist', 'album', 'track_name', 'track_id', 'genre']
Numeric features:     ['energy', 'danceability', 'key', 'loudness', 'acousti
cness', 'speechiness', 'instrumentalness', 'liveness', 'valence', 'tempo',
'duration_ms']
```

What I did:

- Used `spotify_df.select_dtypes(include=['object']).columns` to list categorical columns.
- Re-used the `num_cols` list defined earlier (from 3.c) for all numeric columns.

4.1.b Feature Types Overview

Results:

- **Categorical features (5):** `artist`, `album`, `track_name`, `track_id`, `genre`
- **Numeric features (11):** `energy`, `danceability`, `key`, `loudness`, `acousticness`, `speechiness`, `instrumentalness`, `liveness`, `valence`, `tempo`, `duration_ms`
- **Total features:** 16

What this means:
We clearly distinguish which columns hold numbers versus text.

Why it matters:

- Helps choose the right summary (counts for categories, stats/histograms for numbers).
- Guides engineers on how to store and index each column for optimal query performance.
- Supports feature engineering in models by clearly signaling which variables can be normalized or scaled.

Possible next steps:

- Convert `duration_ms` into a more intuitive format (minutes:seconds).
- Verify there are no typos or inconsistent labels in the categorical fields.
- Generate summary statistics (mean, median) or histograms for each numeric feature.

4.2. Artist & Album Checks

a) Artist & Album Checks (Multi-Track Artists):

Are there any artists that have more than 1 popular track? If yes, which and how many?

4.2.a Multi-Track Artists

Goal:

Identify which artists have more than one song in the Top 50.

In [41]:
```
artist_counts = spotify_df['artist'].value_counts()
multi_artist = artist_counts[artist_counts > 1]
print(multi_artist)
```

```
artist
Dua Lipa         3
Billie Eilish    3
Travis Scott     3
Harry Styles     2
Lewis Capaldi    2
Justin Bieber    2
Post Malone      2
Name: count, dtype: int64
```

What I did:

- Used `spotify_df['artist'].value_counts()` to count tracks per artist.
- Filtered for counts greater than 1 to get `multi_artist`.

4.2.a Multi-Track Artists

Results:

- **Dua Lipa:** 3 tracks
- **Billie Eilish:** 3 tracks
- **Travis Scott:** 3 tracks
- **Harry Styles:** 2 tracks
- **Lewis Capaldi:** 2 tracks
- **Justin Bieber:** 2 tracks
- **Post Malone:** 2 tracks

What this means:

Seven artists each have more than one song in the Top 50, showing repeat success.

Why it matters:

- Highlights reliable partners (e.g., Dua Lipa, Billie Eilish) for exclusive deals and co-branded campaigns.
- Steers marketing spend toward acts that consistently deliver hits.
- Useful for understanding which artists dominate playlists and radio rotations.

Possible next steps:

- Compare audio features of multi-track artists' songs to spot common patterns.
- Check release dates to see if these songs came out around the same album.
- Drill into how marketing pushes or collaborations influenced their multiple entries.

---

b) Artist & Album Checks (Most Popular Artist):

Who was the most popular artist?

4.2.b Most Popular Artist

Goal:
Determine which artist appears most frequently among the Top 50 tracks.

In [42]:
```
max_count = artist_counts.max()
most_pop_artist = artist_counts[artist_counts == max_count]
print(most_pop_artist)
```

```
artist
Dua Lipa         3
Billie Eilish    3
Travis Scott     3
Name: count, dtype: int64
```

What I did:

- Re-used the `artist_counts` Series from section 4.2.a (`spotify_df['artist'].value_counts()`), which maps each artist to their number of Top 50 tracks.
- Called `.max()` on `artist_counts` to retrieve the highest frequency.
- Filtered `artist_counts` for entries equal to `max_count` to isolate the top artist(s).
- Displayed the filtered Series to keep the output.

4.2.b Most Popular Artist

Results:

- **Dua Lipa:** 3 tracks
- **Billie Eilish:** 3 tracks
- **Travis Scott:** 3 tracks

What this means:
Three artists tie at three hits apiece, so no one artist dominates.

Why it matters:

- Allows balanced promotional investments across top performers.
- Strengthens negotiation leverage by avoiding over-commitment to a single act.
- Reduces risk by diversifying spotlight among multiple high-value artists.

Possible next steps:

- Examine the release dates of each artist's tracks to spot timing patterns.
- Compare these artists' performance across different years.
- Analyze common audio features of their hits to uncover success factors.

---

c) Artist & Album Checks (Total Unique Artists):

How many artists in total have their songs in the top 50?

4.2.c Total Unique Artists

Goal:
Determine how many distinct artists have songs in the Top 50.

In [43]:
```
total_artists = spotify_df['artist'].nunique()
print(f"Total unique artists: {total_artists}")
```

```
Total unique artists: 40
```

What I did:

- Used `spotify_df['artist'].nunique()` to count unique artist names.
- Stored the result in `total_artists`.
- Printed out the total count.

4.2.c Total Unique Artists

Results:

- Total unique artists: **40**

What this means:
A total of 40 different artists appear in the Top 50.

Why it matters:

- Demonstrates platform diversity, appealing to both major stars and emerging talent.
- Underpins Spotify's "discovery" value proposition by showcasing wide appeal.
- Sets a benchmark for tracking artist variety year-over-year.

Possible next steps:

- Create a bar chart of track counts per artist to spot who appears most frequently.
- Compare artist diversity to previous years to see shifts in market concentration.
- Drill into artists with multiple hits to analyze factors behind their repeated success.

---

d) Artist & Album Checks (Multiple Top Tracks):

Are there any albums that have more than 1 popular track? If yes, which and how many?

4.2.d Albums with Multiple Top 50 Tracks

Goal:
Determine which albums have more than one track in the Top 50 and count them.

In [44]:
```
album_counts = spotify_df['album'].value_counts()
multi_album = album_counts[album_counts > 1]
print(multi_album)
```

```
album
Future Nostalgia        3
Hollywood's Bleeding    2
Fine Line               2
Changes                 2
Name: count, dtype: int64
```

What I did:

- Used `spotify_df['album'].value_counts()` to tally how many Top 50 tracks each album has.
- Filtered the counts with `> 1` to find albums contributing multiple hits.
- Printed the album names and their track counts.

4.2.d Albums with Multiple Top 50 Tracks

Results:

- **Future Nostalgia** (Dua Lipa): 3 tracks
- **Hollywood's Bleeding** (Post Malone): 2 tracks
- **Fine Line** (Harry Styles): 2 tracks
- **Changes** (Justin Bieber): 2 tracks

What this means:
Four albums each produced more than one Top 50 song.

Why it matters:

- Signals that full-album campaigns (e.g. Future Nostalgia) yield multiple hits.
- Justifies allocating promotional budgets at the album level, not just for singles.
- Demonstrates the album's overall appeal to listeners, unlocking sponsorship and co-branding opportunities around its release.

Possible next steps:

- Plot a bar chart of album vs. number of Top 50 tracks to visualize dominance.
- Compare multi-hit albums across different years for trends.
- Dive into the audio features of tracks on these albums to see what made them repeatedly chart.

---

e) Artist & Album Checks (Total Unique Albums):

How many albums in total have their songs in the top 50?

4.2.e Total Unique Albums

Goal:
Determine how many distinct albums have at least one track in the Top 50.

In [45]:
```
total_albums = spotify_df['album'].nunique()
print(f"Total unique albums: {total_albums}")
```

```
Total unique albums: 45
```

What I did:

- Used `spotify_df['album'].nunique()` to count unique album names.
- Stored the result in `total_albums`.
- Printed out the total count.

4.2.e Total Unique Albums

What this means:
There are 45 distinct albums represented in the Top 50.

Why it matters:

- Highlights broad listener engagement across many releases.
- Informs licensing budgets by estimating how many albums need to be cleared.
- Supports album-focused playlist strategies that cater to fans of full records.

Possible next steps:

- Create a bar chart of albums by number of Top 50 tracks to spot which albums contributed multiple hits.
- Compare the album count year-over-year to see if album diversity is increasing or decreasing.
- Drill into the top-contributing albums to analyze common features (genre, release date, artist).

4.3. Track-Level Queries

a) Track-Level Queries (High Danceability):

Which tracks have a danceability score above 0.7?

4.3.a High Danceability Tracks

Goal:
Identify which tracks have a danceability score above 0.7.

In [46]:
```
high_dance_sorted = (
    spotify_df[spotify_df['danceability'] > 0.7]
      [['artist','track_name','danceability']]
      .sort_values('danceability', ascending=False)
      .reset_index(drop=True)
)
high_dance_sorted.index = high_dance_sorted.index + 1
high_dance_sorted
```

Out[46]:

	artist	track_name	danceability
1	Cardi B	WAP (feat. Megan Thee Stallion)	0.935
2	Roddy Ricch	The Box	0.896
3	Regard	Ride It	0.880
4	Surfaces	Sunday Best	0.878
5	BENEE	Supalonely (feat. Gus Dapperton)	0.862
6	Travis Scott	goosebumps	0.841
7	Travis Scott	SICKO MODE	0.834
8	Drake	Toosie Slide	0.830
9	Tones And I	Dance Monkey	0.825
10	Eminem	Godzilla (feat. Juice WRLD)	0.808
11	Justin Bieber	Intentions (feat. Quavo)	0.806
12	KAROL G	Tusa	0.803
13	Future	Life Is Good (feat. Drake)	0.795
14	Dua Lipa	Don't Start Now	0.793
15	Topic	Breaking Me	0.789
16	Doja Cat	Say So	0.787
17	SAINt JHN	Roses - Imanbek Remix	0.785
18	Trevor Daniel	Falling	0.784
19	Maluma	Hawái	0.783
20	Lil Mosey	Blueberry Faygo	0.774
21	Jawsh 685	Savage Love (Laxed - Siren Beat)	0.767
22	Maroon 5	Memories	0.764
23	Shawn Mendes	Señorita	0.759
24	Post Malone	Sunflower - Spider-Man: Into the Spider-Verse	0.755
25	BTS	Dynamite	0.746
26	DaBaby	ROCKSTAR (feat. Roddy Ricch)	0.746
27	Dua Lipa	Break My Heart	0.730
28	Powfu	death bed (coffee for your head)	0.726
29	Black Eyed Peas	RITMO (Bad Boys For Life)	0.723
30	THE SCOTTS	THE SCOTTS	0.716
31	Billie Eilish	everything i wanted	0.704
32	Billie Eilish	bad guy	0.701

What I did:

- Filtered `spotify_df` for `spotify_df['danceability'] > 0.7`.
- Selected the columns `artist`, `track_name`, and `danceability`.
- Sorted by `danceability` in descending order.
- Reset the index to start from 1.

4.3.a High Danceability Tracks

Results:

- **Total tracks with danceability > 0.7:** 32

1. Cardi B - WAP (feat. Megan Thee Stallion) (0.935)
2. Roddy Ricch - The Box (0.896)
3. Regard - Ride It (0.880)
   …plus 29 more tracks

What this means:
32 tracks score above 0.7 on danceability, indicating strong rhythmic appeal.

Why it matters:

- Forms the core of “Workout” and “Party” playlists, boosting session time.
- Creates partnership potential with fitness and event brands.
- Justifies premium ad formats in high-energy listening contexts.

Possible next steps:

- Break down these 32 tracks by genre to see which style leads.
- Examine other features (energy, tempo) to understand what drives high danceability.
- Plot a histogram or boxplot of danceability scores to visualize the full range.

---

b) Track-Level Queries (Low Danceability):

Which tracks have a danceability score below 0.4?

4.3.b Low Danceability Tracks

Goal:
Identify which tracks have a danceability score below 0.4.

In [47]:
```
low_dance_sorted= (
    spotify_df[spotify_df['danceability'] < 0.4]
      [['artist','track_name','danceability']]
      .reset_index(drop=True)
)
low_dance_sorted.index = low_dance_sorted.index + 1
low_dance_sorted
```

Out[47]:

	artist	track_name	danceability
1	Billie Eilish	lovely (with Khalid)	0.351

What I did:

- Filtered the DataFrame for `spotify_df['danceability'] < 0.4`.
- Selected the columns `artist`, `track_name`, and `danceability`.
- Reset the index and shifted it to start from 1.

4.3.b Low Danceability Tracks

What this means:
Only one track falls below 0.4, marking it as uniquely mellow.

Why it matters:

- This outlier highlights a very mellow or introspective track on an otherwise upbeat list.
- Highlights “chill” or “mood” segment opportunities for focus playlists (e.g., “Late-Night Lounge”).
- Informs personalization engines to avoid low-danceability tracks in high-energy contexts.

Possible next steps:

- Check this track’s acousticness and valence to see what makes it less danceable.
- Compare its other feature values (e.g., tempo, energy) against the group average.
- Consider adjusting the danceability threshold (e.g., <0.5) to capture more “chill” tracks.

---

c) Track-Level Queries (Loudest Tracks):

Which tracks have their loudness above -5?

4.3.c Loudest Tracks

Goal:

Identify which tracks play louder than -5 dB in the dataset.

In [48]:
```
high_loud_sorted = (
    spotify_df[spotify_df['loudness'] > -5]
      [['artist','track_name','loudness']]
      .sort_values('loudness', ascending=False)
      .reset_index(drop=True)
)
high_loud_sorted.index = high_loud_sorted.index + 1
high_loud_sorted
```

Out[48]:

	artist	track_name	loudness
1	KAROL G	Tusa	-3.280
2	Travis Scott	goosebumps	-3.370
3	Dua Lipa	Break My Heart	-3.434
4	Maluma	Hawái	-3.454
5	Post Malone	Circles	-3.497
6	24kGoldn	Mood (feat. iann dior)	-3.558
7	Harry Styles	Adore You	-3.675
8	Travis Scott	SICKO MODE	-3.714
9	Dua Lipa	Physical	-3.756
10	Lady Gaga	Rain On Me (with Ariana Grande)	-3.764
11	Bad Bunny	Safaera	-4.074
12	Harry Styles	Watermelon Sugar	-4.209
13	Regard	Ride It	-4.258
14	Post Malone	Sunflower - Spider-Man: Into the Spider-Verse	-4.368
15	BTS	Dynamite	-4.410
16	Dua Lipa	Don't Start Now	-4.521
17	Doja Cat	Say So	-4.577
18	BENEE	Supalonely (feat. Gus Dapperton)	-4.746
19	Lewis Capaldi	Before You Go	-4.858

What I did:

- Filtered `spotify_df` for `spotify_df['loudness'] > -5`.
- Selected the columns `artist`, `track_name`, and `loudness`.
- Sorted descending by `loudness` (highest volume first).

- Reset the index to start from 1.

4.3.c Loudest Tracks

Results:

- **Total tracks with loudness > -5 dB:** 19

1. KAROL G - Tusa (-3.280 dB)
2. Travis Scott - goosebumps (-3.370 dB)
3. Dua Lipa - Break My Heart (-3.434 dB)
4. Maluma - Hawái (-3.454 dB)
5. Post Malone - Circles (-3.497 dB)
6. 24kGoldn - Mood (feat. iann dior) (-3.558 dB)
7. Harry Styles - Adore You (-3.675 dB)
8. Travis Scott - SICKO MODE (-3.714 dB)
9. Dua Lipa - Physical (-3.756 dB)
10. Lady Gaga - Rain On Me (with Ariana Grande) (-3.764 dB)
11. Bad Bunny - Safaera (-4.074 dB)
12. Harry Styles - Watermelon Sugar (-4.209 dB)
13. Regard - Ride It (-4.258 dB)
14. Post Malone - Sunflower (Spider-Man: Into the Spider-Verse) (-4.368 dB)
15. BTS - Dynamite (-4.410 dB)
16. Dua Lipa - Don’t Start Now (-4.521 dB)
17. Doja Cat - Say So (-4.577 dB)
18. BENEE - Supalonely (feat. Gus Dapperton) (-4.746 dB)
19. Lewis Capaldi - Before You Go (-4.858 dB)

What this means:
Nineteen tracks exceed –5 dB in loudness, driving high-intensity listening.

Why it matters:

- Loud tracks drive excitement and are often featured in workout or party playlists.
- Enables ad partners (e.g., fitness brands) to align messaging with high-intensity audio.
- Helps schedule ad breaks around peak sonic moments without jarring listeners.

Possible next steps:

- Compare these loudest tracks' energy and danceability to see if loudness correlates with other "intensity" metrics.
- Plot a loudness histogram to visualize the overall distribution.
- Examine genre breakdown of these high-volume tracks to identify style patterns.

---

d) Track-Level Queries (Softest Tracks):

Which tracks have their loudness below -8?

4.3.d Softest Tracks (Loudness < -8 dB)

Goal:
Identify which tracks fall below -8 dB in loudness.

In [49]:
```
low_loud_sorted = (
    spotify_df[spotify_df['loudness'] < -8]
      [['artist','track_name','loudness']]
      .sort_values('loudness', ascending=True)
      .reset_index(drop=True)
)
low_loud_sorted.index = low_loud_sorted.index + 1
low_loud_sorted
```

Out[49]:

	artist	track_name	loudness
1	Billie Eilish	everything i wanted	-14.454
2	Billie Eilish	bad guy	-10.965
3	Billie Eilish	lovely (with Khalid)	-10.109
4	JP Saxe	If the World Was Ending - feat. Julia Michaels	-10.086
5	Drake	Toosie Slide	-8.820
6	Powfu	death bed (coffee for your head)	-8.765
7	Travis Scott	HIGHEST IN THE ROOM	-8.764
8	Trevor Daniel	Falling	-8.756
9	Jawsh 685	Savage Love (Laxed - Siren Beat)	-8.520

What I did:

- Filtered the DataFrame for `spotify_df['loudness'] < -8`.
- Selected the columns `artist`, `track_name`, and `loudness`.
- Sorted the results ascending by `loudness` (most negative first).
- Reset the index and shifted it to start from 1.

4.3.d Softest Tracks

Results:

- **Total tracks:** 9

1. Billie Eilish - everything i wanted (-14.454 dB)
2. Billie Eilish - bad guy (-10.965 dB)
3. Billie Eilish - lovely (with Khalid) (-10.109 dB)
4. JP Saxe - If the World Was Ending (feat. Julia Michaels) (-10.086 dB)
5. Drake - Toosie Slide (-8.820 dB)
6. Powfu - death bed (coffee for your head) (-8.765 dB)
7. Travis Scott - HIGHEST IN THE ROOM (-8.764 dB)
8. Trevor Daniel - Falling (-8.756 dB)
9. Jawsh 685 - Savage Love (Laxed - Siren Beat) (-8.520 dB)

What this means:
Nine tracks dip below -8 dB, offering a softer sound profile.

Why it matters:

- Anchors ambient and wellness playlists for retail, spa, and relaxation contexts.
- Informs safe volume-limit recommendations to protect hearing and aid listeners using hearing-assist devices.
- Guides mastering teams on how quiet a “soft” track should be without losing clarity.

Possible next steps:

- Compare these tracks’ acousticness to see if they lean on live instruments.
- Plot a loudness distribution histogram to view the full range of volumes.
- Explore genre or mood correlations (e.g., valence) for these softer tracks.

---

e) Track-Level Queries (Longest Track):

Which track is the longest?

4.3.e Longest Track

Goal:
Find the longest track and display its duration in minutes : seconds instead of

milliseconds.

In [50]:
```
longest = spotify_df.loc[
    spotify_df['duration_ms'].idxmax(),
    ['artist', 'track_name', 'duration_ms']
]

total_seconds = longest['duration_ms'] // 1000
minutes, seconds = divmod(total_seconds, 60)
print(f"Longest track: {longest['artist']} - {longest['track_name']} "
      f"({minutes}:{seconds:02d})")
```

```
Longest track: Travis Scott - SICKO MODE (5:12)
```

What I did

- Used `idxmax()` on the `duration_ms` column to locate the row with the maximum duration.
- Retrieved the `artist`, `track_name`, and `duration_ms` for that row.
- Converted `duration_ms` into minutes and seconds.

4.3.e Longest Track

Results

- **Artist:** Travis Scott
- **Track:** SICKO MODE
- **Duration:** 5 min 12 s (312,820 ms)

What this means:
The longest track runs 5 minutes 12 seconds, extending average session length.

Why it matters:

- May influence skip rates-longer tracks can both boost or impede session duration.
- Informs recommendation engines on optimal track length for different user segments.
- Can be featured as a premium extended-play benefit for subscribers.

Possible next steps:

- Plot a histogram of all track lengths to visualize distribution.
- Examine whether track length correlates with features like acousticness or instrumentalness.
- Compare average durations across genres to see if certain styles favor longer compositions.

---

f) Track-Level Queries (Shortest Track):

Which track is the shortest?

4.3.f Shortest Track

Goal:
Identify which track in the Top 50 dataset has the shortest duration.

In [51]:
```
shortest = spotify_df.loc[
    spotify_df['duration_ms'].idxmin(),
    ['artist', 'track_name', 'duration_ms']
]

total_seconds = shortest['duration_ms'] // 1000
minutes, seconds = divmod(total_seconds, 60)
print(f"Shortest track: {shortest['artist']} - {shortest['track_name']} "
      f"({minutes}:{seconds:02d})")
```

```
Shortest track: 24kGoldn - Mood (feat. iann dior) (2:20)
```

What I did:

- Used `spotify_df['duration_ms'].idxmin()` to find the index of the shortest track.
- Extracted `artist`, `track_name`, and `duration_ms` into `shortest`.
- Converted `duration_ms` into minutes and seconds.

4.3.f Shortest Track

Results:

- **Artist:** 24kGoldn
- **Track Name:** Mood (feat. iann dior)
- **Duration:** 2 min 20s (140,526 ms)

What this means:
At just 2 minutes and 20 seconds, the shortest track invites repeat listens and mirrors the shift toward bite-sized content.

Why it matters:

- Aligns with micro-content strategies-short tracks can drive repeat plays and ad impressions.
- Acts as an effective opener or closer in dynamic playlists.

- Boosts social sharing potential thanks to its concise format.

Possible next steps:

- Compare with the longest track to see the full duration range.
- Plot a histogram of all `duration_ms` values to understand overall length distribution.
- Analyze whether shorter songs correlate with higher popularity metrics (e.g., danceability, loudness).

4.4. Genre Breakdown

a) Genre Breakdown (Most Popular):

Which genre is the most popular?

4.4.a Most Popular Genre

Goal:
Determine which genre appears most frequently among the Top 50 tracks.

In [52]:
```
genre_counts = spotify_df['genre'].value_counts()
most_pop_genre = genre_counts.idxmax()
count = genre_counts.max()
print(f"Most popular genre: {most_pop_genre} ({count} tracks)")
```

```
Most popular genre: Pop (14 tracks)
```

What I did:

- Used `spotify_df['genre'].value_counts()` to tally how many tracks each genre has.
- Found the genre with the highest count via `idxmax()`.
- Retrieved the maximum count using `.max()`.
- Printed a formatted message showing the top genre and its track count.

4.4.a Most Popular Genre

Results:

- Most popular genre: **Pop** (14 tracks)

What this means:
Pop makes up 28% of the Top 50, making it the leading genre in 2020.

Why it matters:

- Drive Pop-centric marketing (e.g., “Spotify Pop Week”) to leverage mass appeal.
- Reflects broad listener preferences - pop tracks are more likely to become hits.
- Use Pop insights as a bellwether for emerging cross-genre trends.

Possible next steps:

- Create a bar chart of all genre counts to visualize how Pop compares to others.
- Compare Pop’s share in previous years to identify rising or falling trends.
- Drill into Pop sub-genres (e.g., dance-pop, pop rap) for more insights.

---

b) Genre Breakdown (Single-Song):

Which genres have just one song on the top 50?

4.4.b Single-Song Genres

Goal:
Find all genres that appear exactly once in the Top 50 dataset.

In [53]:
```
genre_counts = spotify_df['genre'].value_counts()
single_song_genres = genre_counts[genre_counts == 1].index.tolist()
print(f"Total single song genres: {single_song_genres}")
```

```
Total single song genres: ['R&B/Hip-Hop alternative', 'Nu-disco', 'Pop/Soft
Rock', 'Pop rap', 'Hip-Hop/Trap', 'Dance-pop/Disco', 'Disco-pop', 'Dreampop/
Hip-Hop/R&B', 'Alternative/reggaeton/experimental', 'Chamber pop']
```

What I did:
I used `value_counts()` to see how many times each genre appears, then simply filtered for those with exactly one entry.

4.4.b Single-Song Genres

Results:

- R&B/Hip-Hop alternative
- Nu-disco
- Pop/Soft Rock
- Pop rap
- Hip-Hop/Trap

- Dance-pop/Disco
- Disco-pop
- Dreampop/Hip-Hop/R&B
- Alternative/reggaeton/experimental
- Chamber pop

What this means:
Ten niche genres each contribute exactly one track to the Top 50.

Why it matters:

- Highlights the breadth of tastes - listeners embraced both mainstream and very specific subgenres.
- Signals where to scout fresh talent and diversify beyond mainstream.
- Helps test marketing in small niche groups to attract related new audiences.

Possible next steps:

- Listen to the individual tracks in each genre to understand what set them apart.
- Compare with multi-track genres to see if any single-song genres grow in future years.
- Visualize the frequency of single-song versus multi-song genres in a pie chart to illustrate balance.

---

c) Genre Breakdown (Total Unique Genres):

How many genres in total are represented in the top 50?

4.4.c Total Unique Genres

Goal:
Determine how many distinct genres are represented among the Top 50 tracks.

In [54]:
```
total_genres = spotify_df['genre'].nunique()
print(f"Total unique genres: {total_genres}")
```

```
Total unique genres: 16
```

What I did:
I applied the `nunique()` method on the genre column to count all unique genre labels in the dataset.

4.4.c Total Unique Genres

Results:

- Total unique genres: 16

What this means:
Sixteen distinct genres appear in the Top 50.

Why it matters:

- A high genre count reflects diverse listener tastes - hits aren't confined to just pop or rap.
- Understanding this variety helps when tailoring playlists or marketing to different audiences.
- Shows Spotify can serve every kind of music taste.

Possible next steps:

- Create a bar chart of genre frequencies to see which genres dominate.
- Compare genre diversity with previous years to spot shifts in popular taste.
- Drill into the less-represented genres to highlight unique or emerging trends.

4.5. Correlation Overview

a) Correlation Overview (Positively Correlated):

Which features are strongly positively correlated?

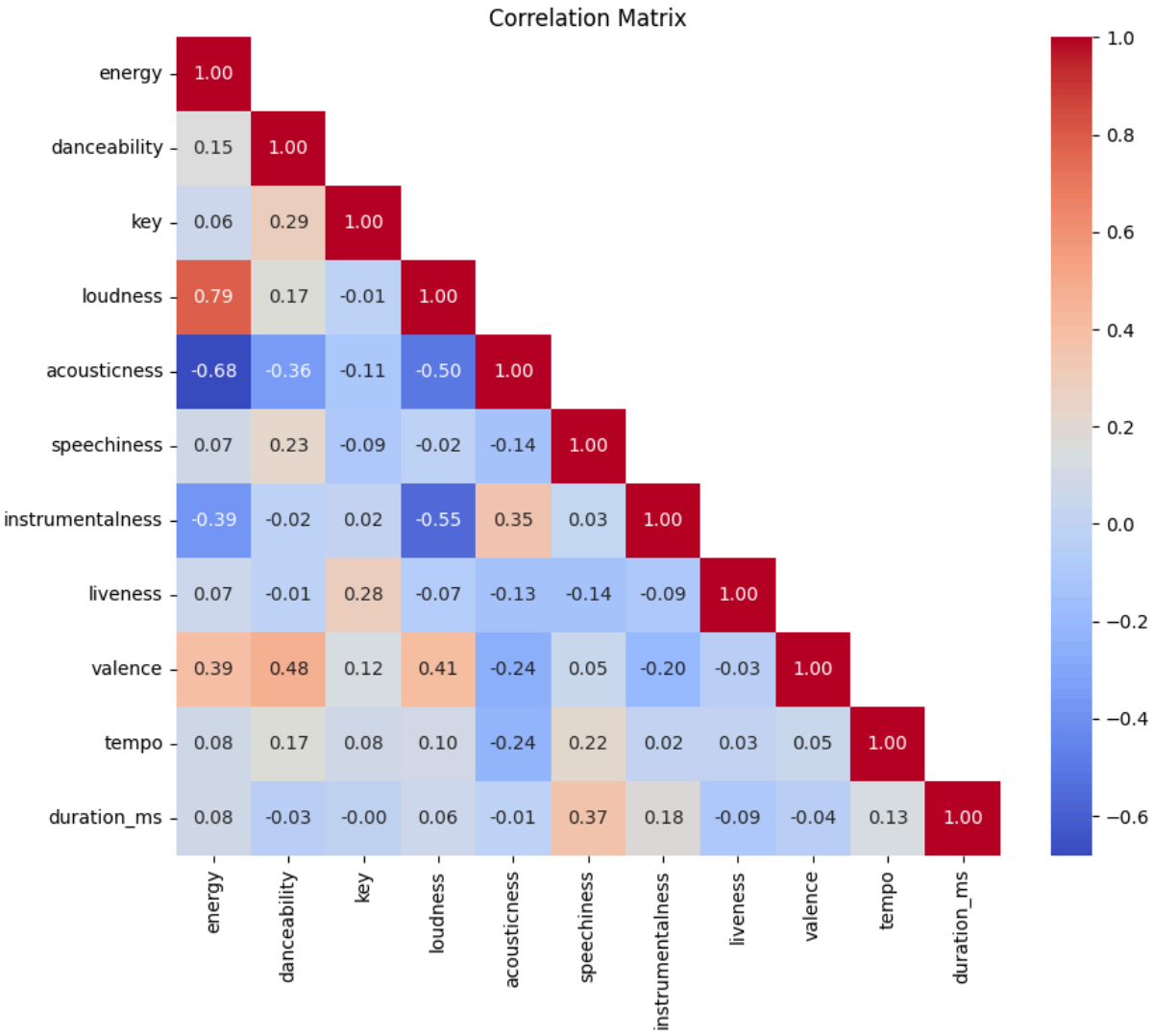
4.5.a Strongly Positive Correlations

Goal:
Find which pairs of numeric features move together (correlation r > 0.70).

In [55]:
```
# Select numeric columns and compute correlation
correlation_matrix = spotify_df[num_cols].corr()
# Mask the upper triangle
upper_triangle_mask = np.triu(
    np.ones_like(correlation_matrix, dtype=bool),
    k=1
)

# Plot
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(
    correlation_matrix,
    mask=upper_triangle_mask,
    annot=True,
    fmt=".2f",
    cmap="coolwarm"
)
plt.title("Correlation Matrix")
plt.show()
```



What I did:

- Reused `num_cols` from step 3c to filter out non-numeric columns in `spotify_df`.
- Calculated the pairwise Pearson correlation matrix using `correlation_matrix = spotify_df[num_cols].corr()`.
- Built a boolean mask with `upper_triangle_mask = np.triu(np.ones_like(correlation_matrix, dtype=bool), k=1)` to hide the upper triangle above the diagonal.

- Initialized a Matplotlib figure of size 10×8 inches via `plt.figure(figsize=(10,8))`.
- Plotted the masked correlation matrix with `sns.heatmap(correlation_matrix, mask=upper_triangle_mask, annot=True, fmt=".2f", cmap="coolwarm")`.
- Added the title `"Correlation Matrix"` using `plt.title("Correlation Matrix")`.
- Displayed the heatmap inline with `plt.show()`.

4.5.a Strongly Positive Correlations

Results:

- **energy & loudness:** r ≈ 0.79
- **Worth mentioning (moderate positives):**
  - valence & danceability (r ≈ 0.48): Happier songs tend to be more danceable.
  - valence & loudness (r ≈ 0.41): Feel-good tracks skew slightly louder.
  - instrumentalness & speechiness (r ≈ 0.35): Instrument-rich songs often include more spoken/vocal elements, though the link is weak.

What this means:
Energy and loudness correlate strongly (r≈0.79), showing that louder tracks feel more energetic.

Why it matters:

- We only need either energy or loudness to describe a song's "intensity".
- Knowing these relationships lets our audio teams and label partners hit precise loudness targets to deliver the right energy profile.
- Since loudness maps so closely to perceived energy, we can build "Workout," "Party," or "Gym" mixes by filtering on just one metric.

Possible next steps:

- Plot energy vs. loudness and valence vs. danceability with simple scatter charts.
- Color points by genre to see if certain styles diverge.
- Combine similar features into one group - for example, call valence and danceability "mood" features.

---

b) Correlation Overview (Negatively Correlated):

Which features are strongly negatively correlated?

4.5.b Strongly Negative Correlations

Goal:
Find pairs of numeric features that move strongly in opposite directions ($r < -0.70$).

What I did:
I used the same heatmap from 4.5.a to spot any deep blue cells indicating strong negative links.

4.5.b Strongly Negative Correlations

Results:

- **Strongly negative correlations ($r < -0.70$):** None

Worth mentioning (just below strong):

- acousticness & energy: $r \approx -0.68$
- instrumentalness & loudness: $r \approx -0.55$
- acousticness & loudness: $r \approx -0.50$

What this means:
No pair crosses $r < -0.70$, although acousticness vs. energy trends moderately negative ($r \approx -0.68$).

Why it matters:

- Informs recommendations to avoid mixing overly acoustic and high-energy tracks in single listening sessions.
- Helps us keep the vibe consistent, helping users stay engaged longer.
- Ensures smoother transitions, which improves overall listening satisfaction.

Possible next steps:

- Try a lower cutoff ($|r| > 0.50$) to include these pairs in our list.
- Make simple scatter plots for acousticness vs. energy and instrumentalness vs. loudness.

- See if these patterns hold true within each genre.

---

c) Correlation Overview (Not Correlated):

Which features are not correlated?

4.5.c Features That Are Not Correlated

Goal: Identify which feature pairs have almost no linear relationship (|r| < 0.10).

What I did:
I used the same heatmap from 4.5.a to spot the very pale cells that show these near-zero links.

4.5.c Features That Are Not Correlated

Results:
These pairs all have Pearson r between -0.10 and +0.10, indicating near-zero linear relationship.

- **energy & key:** r = 0.06
- **energy & speechiness:** r = 0.07
- **energy & liveness:** r = 0.07
- **energy & tempo:** r = 0.08
- **energy & duration_ms:** r = 0.08
- **danceability & instrumentalness:** r = 0.02
- **danceability & liveness:** r = 0.01
- **danceability & duration_ms:** r = 0.03
- **key & loudness:** r = 0.01
- **key & speechiness:** r = 0.09
- **key & instrumentalness:** r = 0.02
- **key & tempo:** r = 0.08
- **key & duration_ms:** r = 0.00
- **loudness & speechiness:** r = 0.02
- **loudness & liveness:** r = 0.07
- **loudness & duration_ms:** r = 0.06
- **acousticness & duration_ms:** r = 0.01
- **speechiness & instrumentalness:** r = 0.03
- **speechiness & valence:** r = 0.05
- **instrumentalness & liveness:** r = 0.09
- **instrumentalness & tempo:** r = 0.02
- **liveness & valence:** r = 0.03
- **liveness & tempo:** r = 0.03

- **liveness & duration_ms:** r = 0.09
- **valence & tempo:** r = 0.05
- **valence & duration_ms:** r = 0.04

Worth mentioning (just outside “no correlation”):
These pairs have r values just above ±0.10 (between -0.11 and 0.14), so they are just over our “no correlation” cutoff.

- **acousticness & key:** r = -0.11
- **speechiness & acousticness:** r = -0.14
- **liveness & acousticness:** r = -0.13
- **liveness & speechiness:** r = -0.14
- **valence & key:** r = 0.12
- **duration_ms & tempo:** r = 0.13

What this means:
Since these features don’t affect each other, we can tweak each one on its own to make our song recommendations more accurate.

Why it matters:

- Mixing independent traits lets us craft more varied and surprising recommendations, keeping listeners engaged.
- Speeds up tests by showing exactly which feature change impacts engagement.
- Sparks fresh playlist themes by combining independent audio traits.

Possible next steps:

- Draw scatter plots for a few examples (e.g., key vs. loudness; liveness vs. speechiness) to visualize their independence.
- Re-check these relationships within each genre to see if any style introduces a link.
- Use the heatmap as a quick reference for spotting independent features in future work.

4.6. Genre Comparisons

a) Genre Comparisons (Danceability):

How does the danceability score compare between Pop, Hip-Hop/Rap, Dance/Electronic, and Alternative/Indie genres?

4.6.a Danceability Comparison by Genre

Goal:
See how danceable songs are in Pop, Hip-Hop/Rap, Dance/Electronic and Alternative/Indie by looking at the middle, lowest, and highest danceability scores.

In [56]:
```
genres = ['Pop', 'Hip-Hop/Rap', 'Dance/Electronic', 'Alternative/Indie']

genre_subset = spotify_df[spotify_df['genre'].isin(genres)]

dance_stats = (
    genre_subset
      .groupby('genre')['danceability']
      .agg(
          median='median',
          minimum='min',
          maximum='max'
      )
      .loc[genres]
)
dance_stats
```

Out[56]:

genre	median	minimum	maximum
Pop	0.690	0.464	0.806
Hip-Hop/Rap	0.774	0.598	0.896
Dance/Electronic	0.785	0.647	0.880
Alternative/Indie	0.663	0.459	0.862

What I did:

- Defined a list `genres = ['Pop', 'Hip-Hop/Rap', 'Dance/Electronic', 'Alternative/Indie']`.
- Created a single filtered DataFrame containing only those genres `genre_subset = spotify_df[spotify_df['genre'].isin(genres)]`.
- Grouped `genre_subset` by `genre` and aggregated the `danceability` column to get the median, minimum, and maximum.
- Applied `.loc[genres]` to ensure the rows appear in the specified order.
- Stored the summary in `dance_stats` and displayed it.

4.6.a. Danceability Comparison by Genre

Results:

- **Pop:**
  - Median: 0.69
  - Range: 0.46 (lowest) to 0.81 (highest)
- **Hip-Hop/Rap:**
  - Median: 0.77
  - Range: 0.60 to 0.90
- **Dance/Electronic:**
  - Median: 0.79
  - Range: 0.65 to 0.88
- **Alternative/Indie:**
  - Median: 0.66
  - Range: 0.46 to 0.86

What this means:
Dance/Electronic leads in danceability (median 0.79), followed by Hip-Hop/Rap (median 0.77), then Pop (median 0.69), with Alternative/Indie trailing (median 0.66).

Why it matters:

- Dance/Electronic (median 0.79) and Hip-Hop/Rap (median 0.77) are your go-to for easy-to-dance-to playlists.
- Pop (median 0.69) gives a balance of dance-floor hits and more relaxed tunes.
- Alternative/Indie (median 0.66) covers a wide style range-perfect for mixing mellow and upbeat tracks.
- Use genre danceability profiles to tailor local marketing-e.g., focus on Dance/Electronic in markets with high gym and club usage.

Possible next steps:

- Draw boxplots or histograms to see how tightly each genre's scores cluster.
- Break Pop and Hip-Hop/Rap into sub-genres (ballads vs. bangers) for finer insights.

---

b) Genre Comparisons (Loudness):

How does the loudness score compare between Pop, Hip-Hop/Rap, Dance/Electronic, and Alternative/Indie

genres?

4.6.b Loudness Comparison by Genre

Goal:
See how loud songs are in Pop, Hip-Hop/Rap, Dance/Electronic and Alternative/Indie, by looking at the middle, quietest and loudest values.

In [57]:
```
loud_stats = (
    genre_subset
      .groupby('genre')['loudness']
      .agg(
          median='median',
          minimum='min',
          maximum='max'
      )
      .loc[genres]
)
loud_stats
```

Out[57]:

genre	median	minimum	maximum
Pop	-6.6445	-14.454	-3.280
Hip-Hop/Rap	-7.6480	-8.820	-3.370
Dance/Electronic	-5.4570	-7.567	-3.756
Alternative/Indie	-5.2685	-6.401	-4.746

What I did:

- Reused the same list `genres = ['Pop', 'Hip-Hop/Rap', 'Dance/Electronic', 'Alternative/Indie']`.
- Reused the filtered DataFrame `genre_subset` (already defined for danceability).
- Grouped `genre_subset` by `genre` and aggregated the `loudness` column to get the median, minimum, and maximum.
- Applied `.loc[genres]` to ensure the rows appear in the specified order.
- Stored the summary in `loud_stats` and displayed it.

4.6.b Loudness Comparison by Genre

Results:

- **Pop:**
  - Median: -6.64 dB
  - Range: -14.45 dB (quietest) to -3.28 dB (loudest)
- **Hip-Hop/Rap:**

    - Median: -7.65 dB
    - Range: -8.82 dB to -3.37 dB
- **Dance/Electronic:**
    - Median: -5.46 dB
    - Range: -7.57 dB to -3.76 dB
- **Alternative/Indie:**
    - Median: -5.27 dB
    - Range: -6.40 dB to -4.75 dB

What this means:
Dance/Electronic and Alternative/Indie tracks play the loudest (around –5 dB), while Hip-Hop/Rap is notably quieter (around –7.65 dB).

Why it matters:

- High-volume genres like Dance/Electronic and Indie are ideal for energizing venues (gyms, clubs) and can command sponsorships.
- Pop's full spectrum lets brands flex between subtle background music and attention-grabbing anthems in retail and broadcast contexts.
- Hip-Hop/Rap's consistent moderate volume fits lifestyle settings (cafés, lounges) and enables predictable ad-insertion without jarring level jumps.

Possible next steps:

- Draw boxplots to visualize how tight or spread out each genre's loudness is.
- Drill down into sub-genres (e.g., separating pop ballads from pop dance hits) to refine these insights.

---

c) Genre Comparisons (Acousticness):

How does the acousticness score compare between Pop, Hip-Hop/Rap, Dance/Electronic, and Alternative/Indie genres?

4.6.c Acousticness Comparison by Genre

Goal: See how much each genre uses live or "acoustic" sounds (guitars, pianos, etc.) compared to purely electronic production.

In [58]:
```
acoustic_stats = (
    genre_subset
      .groupby('genre')['acousticness']
```

```
      .agg(
          median='median',
          minimum='min',
          maximum='max'
      )
      .loc[genres]
)
acoustic_stats
```

Out[58]:

genre	median	minimum	maximum
Pop	0.2590	0.02100	0.902
Hip-Hop/Rap	0.1450	0.00513	0.731
Dance/Electronic	0.0686	0.01370	0.223
Alternative/Indie	0.6460	0.29100	0.751

What I did:

- Reused the same list `genres = ['Pop', 'Hip-Hop/Rap', 'Dance/Electronic', 'Alternative/Indie']`.
- Reused the filtered DataFrame `genre_subset` (already defined for danceability and loudness).
- Grouped `genre_subset` by `genre` and aggregated the `acousticness` column to get the median, minimum, and maximum.
- Applied `.loc[genres]` to ensure the rows appear in the specified order.
- Stored the summary in `acoustic_stats` and displayed it.

4.6.c. Acousticness Comparison by Genre

Results:

- **Pop:**
  - Median acousticness: 0.26
  - Range: 0.02 (lowest) to 0.90 (highest)
- **Hip-Hop/Rap:**
  - Median acousticness: 0.15
  - Range: 0.01 to 0.73
- **Dance/Electronic:**
  - Median acousticness: 0.07
  - Range: 0.01 to 0.22
- **Alternative/Indie:**
  - Median acousticness: 0.65
  - Range: 0.29 to 0.75

What this means:
Alternative/Indie is the most acoustic (median 0.65), Pop sits in the middle (median 0.26), Hip-Hop/Rap is lower (median 0.15), and Dance/Electronic is almost purely electronic (median 0.07).

Why it matters:

- Alternative/Indie's median 0.65 shows it's heavy on live instruments.
- Dance/Electronic's median 0.07 shows it's almost entirely electronic.
- Pop's 0.26 and Hip-Hop/Rap's 0.15 medians sit in between, mixing live and electronic.
- Clear genre acoustic profiles let us forecast listener engagement and tailor pricing or promotions by taste group.

Possible Next steps:

- Check if these differences hold in larger datasets or in sub-genres.

Spotify Tracks Analysis: Final Summary

Insights Gained from Analysis:

Dataset Structure & Quality:

- Dataset contains **50 unique tracks** with **16 attributes** each.
- Data is **clean, complete**, with **no missing values or duplicates**.
- **Outliers** identified in:
  - **Instrumentalness (12 outliers)**
  - **Acousticness (7 outliers)**
  - **Speechiness (6 outliers)**

Feature Distribution:

- **Danceability**:
  - Lowest: "lovely" by Billie Eilish (0.351)
  - Highest: "WAP" by Cardi B (0.935)
- **Loudness** varies significantly:
  - Quietest: "everything i wanted" by Billie Eilish (-14.454 dB)
  - Loudest: "Tusa" by KAROL G (-3.280 dB)
- **Duration**:
  - Longest track: "SICKO MODE" by Travis Scott (5:12)

    - Shortest track: "Mood" by 24kGoldn (2:20)

Artist and Album Patterns:

- **40 unique artists**; artists with multiple hits:
    - **Dua Lipa, Billie Eilish, Travis Scott**: 3 tracks each
    - **Harry Styles, Lewis Capaldi, Justin Bieber, Post Malone**: 2 tracks each
- Albums with multiple popular tracks:
    - "Future Nostalgia" by Dua Lipa: 3 tracks
    - Albums by Post Malone, Harry Styles, and Justin Bieber: 2 tracks each

Genre Insights:

- **Pop genre** dominates with **14 tracks**.
- **16 unique genres**; notable diversity with many single-track genres highlighting niche interests.

Feature Correlations:

- **Strong positive correlation**:
    - Energy & Loudness (**r ≈ 0.79**), indicating louder tracks are more energetic.
- **Moderate negative correlation**:
    - Acousticness & Energy (**r ≈ -0.68**), indicating acoustic tracks generally lower in energy.
- **No strong negative correlations**; many features independent (e.g., Energy & Key, r ≈ 0.06).

Genre-Specific Features:

- **Dance/Electronic, Hip-Hop/Rap**:
    - High danceability and energy, suitable for energetic contexts.
- **Alternative/Indie**:
    - Higher acousticness, ideal for mellow, reflective listening.

Most Promising Directions for Further Research:

Deeper Genre Analysis:

- Cluster songs by their audio features (tempo, energy, mood) to discover emerging micro-genres.

Artist Performance Trends:

- Model how an artist’s release schedule impacts their streaming growth.

Audience Engagement:

- Analyze skip and replay points within tracks to identify the “hook” that keeps listeners.

Cross-Feature Interaction:

- Study combinations like high danceability with sad lyrics to see which contrasts truly resonate.

This targeted analysis aims to enhance playlist curation, artist promotion strategies, and listener engagement.

My Further Improvements

1. When I read the CSV, only grab the real columns (so I never import that annoying “Unnamed: 0” in the first place).
2. Make every plot with Matplotlib and actually give each chart a title and axis labels-no more mystery graphs.
3. Right after loading, convert `duration_ms` to seconds and format a `duration_str = "M:SS"` column so I don’t have to redo it later.