Understanding Music Trends and Audience Preferences Using Spotify Data

Shashira Guntuka, Srilakshmi Narayana Murthy, Mahak Gupta, Akanksha Shukla, Mamta Jha
DATA 230 Data Visualization
San Jose State University

***Abstract*—This project dives into the Spotify Tracks Dataset mainly to reveal music trends, listener tastes, and the characteristics that make a song a hit. The dataset contains more than 114,000 tracks, with a great variety of musical features being provided, like danceability, energy, loudness, acousticness, tempo, valence, and popularity. First, we cleaned and processed the dataset: we removed duplicates, corrected data types, standardized text fields, and dealt with missing values. Next, feature engineering was performed to create new attributes that are more meaningful, e.g., mood, energy level, tempo category, duration buckets, expressive score, complexity score, and genre indicators. Through data visualization techniques, we were able to analyze the genre distribution, duration trends, artist activity, and the relationship between audio features and popularity. The project's main purpose is to turn raw musical features into clear insights that can identify what kinds of songs do well and what musical traits are more influential on listener behavior.**

***Index Terms*—music analytics, Spotify, data visualization, feature engineering, audio features, genre analysis**

I. Motivation

The impetus for this project came from the realization that music has become very much a matter of data today. For instance, Spotify is one of those platforms that provides thorough insight into every single song, and the corresponding metrics mirror actual listener activity. Music is so full of data that it can be analyzed with ease. Changes in music trends indicate shifts in cultures and personalities. Data analytics are the backbone of streaming services, which use them for recommendations, playlist-making, and hit prediction. The project presents an excellent chance to showcase one's skills in data cleaning, preprocessing, feature engineering, and visualization on a real dataset.

The dataset analysis will lead to formulating answers to questions of great significance such as:

- What makes certain songs get more famous than others?
- Is the preference of the audience for music that is energetic or music that is calm?
- What genres and artists are the most listened to among the streaming lot?
- What impact do moods and musical traits have on a song's popularity?

II. Objectives

The main objectives of this study are:

A. Identify song attributes that influence popularity

One of the central goals of this study is to understand *why certain songs become more popular than others*. The Spotify dataset provides a wide range of audio features such as **danceability, energy, valence, loudness, tempo, instrumentalness, and duration**. These features describe the mood, rhythm, and structure of a song.

By analyzing how these characteristics correlate with popularity scores, we aim to uncover patterns that explain what listeners tend to prefer. For example, we investigate whether high-energy songs perform better, whether danceable tracks attract more streams, or whether mood-related attributes such as valence influence listener engagement.

B. Explore genre-wise and duration-wise trends

The dataset includes songs across many genres—such as pop, rock, hip-hop, classical, electronic, and more. Each genre has its own style, audience, and performance characteristics. By studying genre distributions, we can see which genres dominate the dataset, how diverse the collection is, and which genres tend to produce longer or shorter songs.

In addition to genre, **song duration** is also an important factor. Modern music trends show a shift toward shorter tracks due to listener attention spans and streaming algorithms. By categorizing songs into duration buckets (Short, Medium, Long, Very Long), we analyze which durations are most common and whether certain duration ranges correspond to higher popularity.

This objective helps us identify:

- Which genres contribute the most content
- How duration varies across songs and genres
- Whether shorter or longer songs perform better

C. Analyze artist performance

Artists contribute differently to the overall dataset—some have thousands of tracks, while others have only a few. By analyzing artist-level statistics, we can identify:

- Which artists have the highest number of songs
- Which artists consistently produce popular tracks
- How artist activity influences overall dataset trends

This helps us understand how prolific artists shape the dataset and whether reputation or productivity correlates with higher popularity. Tracking the performance of top artists also provides insight into music industry patterns and artist engagement.

D. Understand listener mood and energy preferences

Spotify provides emotional and psychological indicators such as **valence** (happiness) and **energy**. These metrics allow us to derive new mood and energy categories like:

- **Happy / Neutral / Sad** (based on valence)
- **High Energy / Moderate / Chill** (based on energy)

By analyzing these categories, we can understand how emotional tone influences listener behavior. For example:

- Do listeners prefer upbeat and happy songs?
- Do chill, low-energy tracks perform better for certain genres?
- What combinations of energy and mood result in higher popularity?

This objective helps bridge musical structure with emotional interpretation, giving deeper insights into listener psychology and preferences.

E. Prepare a clean, enriched dataset for visualization

Before creating meaningful visualizations, the dataset must be thoroughly cleaned and enhanced. This objective focuses on transforming raw data into a structured, analysis-ready form.

This includes:

- Handling missing values
- Standardizing text formats
- Removing duplicates
- Fixing incorrect data types
- Engineering new features such as mood, tempo category, duration bucket, complexity score, expressive score, genre count, and cross-genre indicator

These enriched features allow us to create clearer and more informative graphs. For example, visualizing mood categories or tempo buckets is far more intuitive than interpreting raw numeric values like valence or BPM.

III. Dataset Overview

We used the **Spotify Tracks Dataset from Kaggle**, containing approximately **114,000 tracks** and **20+ features**. It includes:

A. Metadata

- track_id
- track_name
- artists
- album_name
- track_genre

B. Popularity & Duration

- popularity (0–100 scale indicating listener engagement)
- duration_ms
- explicit (whether the track contains explicit content)

C. Audio Features (from Spotify API)

- danceability
- energy
- speechiness
- acousticness
- instrumentalness
- liveness
- valence
- tempo
- loudness
- key, mode, time_signature

D. Dataset Characteristics

- Very large: 114k tracks → useful for visualization and trend analysis
- Covers multiple genres: pop, rock, hip-hop, classical, electronic, etc.
- Contains rich quantitative features that describe the **mood**, **style**, and **structure** of each song
- Offers popularity scores that help study **listener preferences**

This diversity makes the dataset perfect for studying what makes music appealing.

IV. Methodology

The methodology follows a systematic data-science workflow.

A. Data Preprocessing

- **Loaded the CSV file and inspected its shape and columns.**

The dataset was first loaded into Python, and a quick inspection was performed to understand its shape, number of rows and columns, and overall structure. We checked the data types of each column, the presence of missing values, and reviewed sample rows to get a sense of the formatting and content. This initial check helped identify issues such as unnecessary columns, inconsistent types, and areas that required cleaning.

```
PATH = r"C:\Users\tarun\Downloads\dataset.csv\dataset.csv"

df = pd.read_csv(PATH)

print("Shape (rows, cols):", df.shape)
print("\nColumns:", list(df.columns))
print("\nDtypes:")
print(df.dtypes)

print("\nSample rows:")
display(df.head(5))

print("\nNull counts (top 10):")
print(df.isna().sum().sort_values(ascending=False).head(10))
```

Fig. 1. Dataset inspection output showing shape and structure.

- **Removed unused columns such as Unnamed: 0.**

An extra column, Unnamed: 0, was found in the dataset. This column was likely created during an earlier export

process and did not provide any meaningful information. It was removed to reduce noise and avoid confusion during later steps.

```
cols_before = set(df.columns)
df.drop(columns=["Unnamed: 0"], inplace=True, errors="ignore")
dropped = cols_before - set(df.columns)

print("Dropped columns:", dropped if dropped else "None")
print("New shape:", df.shape)
```

Fig. 2. Removing unused columns from the dataset.

- **Fixed numerical and categorical data types.**

Several columns required type adjustments. All audio features such as danceability, energy, loudness, tempo, and valence were converted to numeric types to ensure correct calculations. Similarly, categorical fields like track_genre were assigned categorical types for easier grouping. The explicit column was standardized to a consistent boolean or integer format (0 or 1). Ensuring that each column had the correct type is important for avoiding errors in analysis.

```
numeric_should_be = [
    "popularity","duration_ms","danceability","energy","key","loudness",
    "speechiness","acousticness","instrumentalness","liveness","valence",
    "tempo","time_signature","mode"
]
numeric_present = [c for c in numeric_should_be if c in df.columns]


for c in numeric_present:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Categorical conversions (only if present)
if "track_genre" in df.columns:
    df["track_genre"] = df["track_genre"].astype("category")

if "explicit" in df.columns:

    if df["explicit"].dtype != bool:

        mapping = {"true": True, "false": False, "1": True, "0": False}
        df["explicit"] = (
            df["explicit"]
            .astype(str).str.lower().map(mapping)
            .fillna(df["explicit"].astype(bool))
        )
    df["explicit"] = df["explicit"].astype(bool)

print("Dtype summary after coercion:")
print(df.dtypes)
```

Fig. 3. Data type corrections applied to the dataset.

- **Handled missing values by replacing unknown text fields with ”Unknown”.**

Missing values were mainly found in text fields such as track_name, artists, and album_name. To maintain consistency, these missing entries were filled with the placeholder ”Unknown.” This prevents issues when grouping or aggregating data later on. Since the missing values were minimal, this approach had no negative impact on the dataset.

```
fill_map = {}
for col in ["artists", "album_name", "track_name"]:
    if col in df.columns:
        fill_map[col] = "Unknown"

missing_before = df.isna().sum().sum()
df.fillna(fill_map, inplace=True)
missing_after_fill_text = df.isna().sum().sum()

print(f"Total missing before: {missing_before}")
print(f"Total missing after text fill: {missing_after_fill_text}")

rem = df.isna().sum()
print(rem[rem > 0].sort_values(ascending=False))
```

Fig. 4. Handling missing values by filling with ”Unknown”.

- **Removed duplicates based on track_id and semantic combinations (track name + artist + duration).**

Given the large size of the dataset, it was important to check for duplicates. Duplicate tracks can appear due to repeated entries or slightly different formatting. We first removed exact duplicates across all columns. Then, to catch more subtle duplicates, we compared combinations of track name, artist name, and duration. Tracks with identical values in these fields were considered repeats and removed. This ensured that each song contributed to the analysis only once.

```
rows_before = len(df)
df = df.drop_duplicates().reset_index(drop=True)
print("Removed exact duplicate rows:", rows_before - len(df))

dup_id_count = df.duplicated(subset=["track_id"]).sum() if "track_id" in df.columns else 0
print("Potential duplicates by track_id:", dup_id_count)

sem_parts = []
if "track_name" in df.columns:
    sem_parts.append(df["track_name"].astype(str).str.lower().str.strip())
if "artists" in df.columns:
    sem_parts.append(df["artists"].astype(str).str.lower().str.strip())
if "duration_ms" in df.columns:
    sem_parts.append((df["duration_ms"] // 1000).astype(str))

if sem_parts:
    df["__sem_key__"] = " | ".join(sem_parts) if isinstance(sem_parts, str) else sem_parts[0]
    for part in sem_parts[1:]:
        df["__sem_key__"] = df["__sem_key__"] + " | " + part
else:
    df["__sem_key__"] = pd.NA
```

Fig. 5. Duplicate removal process.

```
sort_cols = ["popularity"] if "popularity" in df.columns else None
if sort_cols:
    df = df.sort_values(sort_cols, ascending=False)

before = len(df)
if "track_id" in df.columns:
    df = df.drop_duplicates(subset=["track_id"], keep="first")
after_track_id = before - len(df)

before2 = len(df)
if df["__sem_key__"].notna().any():
    df = df.drop_duplicates(subset=["__sem_key__"], keep="first")
after_sem = before2 - len(df)

df = df.drop(columns="__sem_key__", errors="ignore").reset_index(drop=True)

print(f"Removed by track_id: {after_track_id}")
print(f"Removed by semantic key: {after_sem}")
print("Final shape:", df.shape)
```

```
Removed exact duplicate rows: 450
Potential duplicates by track_id: 23809
Removed by track_id: 23809
Removed by semantic key: 7262
Final shape: (82479, 20)
```

Fig. 6. Duplicate removal results.

This ensured the data was clean, consistent, and ready for analysis.

B. Data Cleaning

Data Cleaning

Cell 1 — Text cleaning (trim + standardized capitalization)

```
In [11]: # Phase 2 — Step 1: Text cleaning
         # - Trim whitespace
         # - Title-case or sentence-case (choose what fits best; here: title-case)

         text_cols = [c for c in ["track_name", "album_name", "artists"] if c in df.col

         def clean_text(s: pd.Series, mode="title"):
             s = s.astype(str).str.strip()
             if mode == "title":
                 s = s.str.title()
             elif mode == "lower":
                 s = s.str.lower()
             elif mode == "upper":
                 s = s.str.upper()
             return s

         for c in text_cols:
             df[c] = clean_text(df[c], mode="title")

         print("Cleaned text columns:", text_cols)
         display(df[text_cols].head(5))
```

```
Cleaned text columns: ['track_name', 'album_name', 'artists']
```

	track_name	album_name	artis
0	Unholy (Feat. Kim Petras)	Unholy (Feat. Kim Petras)	Sam Smith;Kim Petr
1	Quevedo: Bzrp Music Sessions, Vol. 52	Quevedo: Bzrp Music Sessions, Vol. 52	Bizarrap;Queve
2	La Bachata	La Bachata	Manuel Turi

Fig. 7. Data cleaning workflow.

Standardization of fields - In the first step of the cleaning process, we standardized all text-based columns such as *track_name*, *album_name,* and *artists*. This included trimming leading and trailing whitespace and applying consistent capitalization (title-case), ensuring that similar values were not treated as separate categories due to formatting inconsistencies. By applying a reusable text-cleaning function to each selected column, we eliminated issues like inconsistent casing, extra spaces, and mixed formatting. This step was crucial because text columns often contain semantic information—such as song titles and artist names—that need to be uniform for accurate grouping, filtering, and visualization.

Range Validation of Audio Attributes - Next, we performed detailed range verification on Spotify's normalized audio features, including *danceability*, *energy*, *speechiness*, *acousticness*, and *instrumentalness*, which are all expected to fall between 0 and 1. The code examined minimum and maximum values for each of these features to confirm whether any values fell outside the expected range. This check helped us identify potential data-entry errors or anomalies and ensured the dataset adhered to Spotify's feature definitions. For additional numeric features, such as *duration_ms* and *tempo*, we conducted positivity checks to confirm that values were logically valid. These validations increased the reliability of the data before performing statistical analysis and visualizations.

Cell 2 — Outlier detection (summaries for key numeric columns)

```
In [12]: # Phase 2 — Step 2: Outlier detection (non-destructive)

         num_cols = [c for c in [
             "duration_ms", "tempo", "loudness", "danceability", "energy",
             "speechiness", "acousticness", "instrumentalness", "liveness", "valence"
         ] if c in df.columns]

         summary = []
         for c in num_cols:
             q1, q3 = df[c].quantile([0.25, 0.75])
             iqr = q3 - q1
             low, high = q1 - 1.5*iqr, q3 + 1.5*iqr
             summary.append({
                 "column": c,
                 "min": float(df[c].min()),
                 "q1": float(q1),
                 "median": float(df[c].median()),
                 "q3": float(q3),
                 "max": float(df[c].max()),
                 "IQR": float(iqr),
                 "lower_fence": float(low),
                 "upper_fence": float(high),
                 "pct_below": float((df[c] < low).mean()*100),
                 "pct_above": float((df[c] > high).mean()*100),
             })

         outlier_df = pd.DataFrame(summary).sort_values("pct_above", ascending=False)
         display(outlier_df)
```

	column	min	q1	median	q3	max	IQR
7	instrumentalness	0.000	0.0000	0.000086	0.1450	1.000	0.1450
5	speechiness	0.000	0.0360	0.049100	0.0866	0.965	0.0506

Fig. 8. Range validation results.

Outlier Detection Using Statistical Thresholds - The next step involved detecting outliers in key numerical columns like *tempo*, *duration_ms*, *loudness*, *danceability*, *energy*, and *valence*. Using the interquartile range (IQR) method, we computed Q1, Q3, and IQR for each feature and calculated the lower and upper fences for potential outliers. We did not remove these values, but instead summarized them in a separate dataframe to assess how many points fell outside expected ranges. This non-destructive outlier analysis gave us insight into which features contained extreme values—information that is essential when interpreting distributions, spotting anomalies, and planning feature engineering.

```
# 4.1 Remaining NaNs?
nulls = df.isna().sum()
print("Columns with remaining NaNs:")
print(nulls[nulls > 0].sort_values(ascending=False) if (nulls > 0).any() else "None")

# 4.2 Expected ranges for 0-1 bounded features
bounded_01 = [c for c in ["danceability","energy","speechiness","acousticness","instrument
              if c in df.columns]
range_issues = {}
for c in bounded_01:
    min_c, max_c = df[c].min(), df[c].max()
    range_issues[c] = (float(min_c), float(max_c), bool((min_c < 0) or (max_c > 1)))

print("\n0-1 bounded feature ranges (min, max, out_of_bounds?):")
for k, (mn, mx, bad) in range_issues.items():
    print(f"{k:18s} -> ({mn:.4f}, {mx:.4f})  OOB={bad}")

# 4.3 Positivity checks for duration & tempo
if "duration_ms" in df.columns:
    print("\nDuration_ms >= 0 ?", bool((df["duration_ms"] >= 0).all()))
if "tempo" in df.columns:
    print("Tempo > 0 ?", bool((df["tempo"] > 0).all()))

# 4.4 Loudness sanity (typical range ~ [-60, 0])
if "loudness" in df.columns:
    mn, mx = df["loudness"].min(), df["loudness"].max()
    print(f"Loudness range: ({mn:.2f}, {mx:.2f})  [Expected ~ -60..0]")
```

```
Columns with remaining NaNs:
None

0-1 bounded feature ranges (min, max, out_of_bounds?):
danceability       -> (0.0000, 0.9850)  OOB=False
energy             -> (0.0000, 1.0000)  OOB=False
speechiness        -> (0.0000, 0.9650)  OOB=False
```

Fig. 9. Outlier detection methodology.

Cell 4 — Encoding categorical variables (explicit → 0/1; genre categorical)

```
# Phase 2 — Step 3: Encode categorical variables

# explicit -> binary 0/1
if "explicit" in df.columns:
    # If already boolean this will convert to 0/1; if not, try to coerce
    if df["explicit"].dtype != bool:
        mapping = {"true": True, "false": False, "1": True, "0": False, "y
        df["explicit"] = (
            df["explicit"].astype(str).str.lower().map(mapping)
            .fillna(df["explicit"].astype(bool))
        )
    df["explicit"] = df["explicit"].astype(int)

# track_genre -> category
if "track_genre" in df.columns:
    df["track_genre"] = df["track_genre"].astype("category")

print("Encoded 'explicit' to 0/1 (if present) and set 'track_genre' as cat
print(df[["explicit"]].head() if "explicit" in df.columns else "No 'explic
print(df["track_genre"].cat.categories[:] if "track_genre" in df.columns e
```

```
Encoded 'explicit' to 0/1 (if present) and set 'track_genre' as categorica
   explicit
0         0
1         0
2         0
3         1
4         0
Index(['acoustic', 'afrobeat', 'alt-rock', 'alternative', 'ambient', 'anim
       'black-metal', 'bluegrass', 'blues', 'brazil',
       ...
       'spanish', 'study', 'swedish', 'synth-pop', 'tango', 'techno', 'tra
       'trip-hop', 'turkish', 'world-music'],
```

Fig. 10. Outlier detection results.

Transforming Text Categories into Numerical Formats - For categorical cleaning, we transformed the **explicit** field into a clean binary format by mapping text values such as "True", "False", "1", and "0" into consistent integers (1 and 0). This ensured that the explicitness of a track could be used efficiently in modeling and visualizations. Similarly, the **track_genre** column was converted into a *pandas Category* type, making it memory-efficient and easier to work with for grouped analysis and plotting. Encoding these categorical variables made the dataset more structured, machine-friendly, and ready for downstream feature engineering, aggregation, and visualization tasks.

These cleaning steps reduced noise and improved reliability.

C. Feature Engineering

Feature engineering was a major part of our project. We created the following new features: Feature engineering played a crucial role in enhancing our analysis of the Spotify dataset. While the raw dataset provided numerous numerical audio attributes, many of these metrics were difficult to interpret directly. To transform these raw attributes into meaningful, human-interpretable insights, we engineered several new features. These derived attributes allowed us to categorize songs based on mood, energy, tempo, duration, genre, and expressive characteristics, which made visual patterns more apparent and increased the overall analytical depth of the project.

1) Duration-Based Features: **1. duration_min**

We converted the song duration from milliseconds to minutes since minute-scale values are more intuitive for interpretation. This enabled easier comparison of song lengths and allowed us to analyze trends in song duration across genres and popularity levels.

2. log_duration_min

Duration often exhibits a right-skewed distribution (many short songs, fewer long songs). Applying a log transformation helped normalize the distribution, making duration-based visualizations clearer and reducing the influence of extreme outliers.

3. duration_bucket (Short / Medium / Long / Very Long)

To simplify duration analysis, we categorized tracks into bins: Short ($<$ 2 min), Medium (2–4 min), Long (4–6 min), Very Long ($>$ 6 min).

This categorical representation allowed us to compare duration preferences across genres and assess whether shorter or longer tracks tend to receive higher listener engagement.

```
# --- Duration features ---
df_feat["duration_min"] = df_feat["duration_ms"] / 60000
df_feat["log_duration_min"] = np.log1p(df_feat["duration_min"])
```

Fig. 11. Duration bucket creation.

```
# Average features for artist and genre insights
df_feat["duration_bucket"] = pd.cut(
    df_feat["duration_min"], bins=[0, 2, 3.5, 5, np.inf],
    labels=["Short", "Medium", "Long", "Very Long"]
)
```

Fig. 12. Duration feature engineering results.

2) Tempo Features: **tempo_category (Slow / Medium / Fast)**

Tempo indicates the speed of a song and strongly influences how listeners perceive energy and mood.

We categorized tempo into:

- Slow – typically softer or emotional tracks
- Medium – balanced rhythm, suitable for most genres
- Fast – high-energy or dance tracks

These categories made it easier to visualize which tempo ranges are most common in popular songs or dominant within specific genres.

```
# --- Tempo categories ---
df_feat["tempo_category"] = pd.cut(
    df_feat["tempo"], bins=[-np.inf, 90, 120, np.inf],
    labels=["Slow", "Medium", "Fast"]
)
```

Fig. 13. Tempo categorization.

3) Mood & Expressive Features: **1. mood (Happy / Neutral / Sad)**

Using valence, which measures musical positivity, we derived an emotional classification:

- Happy – high valence (bright, cheerful songs)
- Neutral – moderate valence
- Sad – low valence (melancholic or emotional songs)

This classification helped us analyze how emotional tone correlates with popularity and genre trends.

```
# --- Mood from valence ---
df_feat["mood"] = np.select(
    [df_feat["valence"] > 0.7, df_feat["valence"] < 0.3],
    ["Happy", "Sad"], default="Neutral"
)
```

Fig. 14. Mood classification.

2. energy_level (High Energy / Moderate / Chill)

Based on the energy feature, we grouped tracks into levels reflecting intensity:

- Chill – low energy
- Moderate – mid-range
- High Energy – intense or upbeat tracks

These categories supported visual comparisons between listener energy preferences and genre-specific tendencies.

```
# --- Energy level ---
df_feat["energy_level"] = np.select(
    [df_feat["energy"] > 0.7, df_feat["energy"] < 0.3],
    ["High Energy", "Chill"], default="Moderate"
)
```

Fig. 15. Energy level categorization.

3. danceability_cat (Low / Medium / High)

Danceability is crucial for genres like pop, EDM, and hip-hop. Categorizing it enabled us to explore: whether high-danceability songs tend to be more popular which genres produce the most danceable tracks. This feature proved essential in trend discovery around modern mainstream music.

```
# --- Danceability category ---
df_feat["danceability_cat"] = pd.cut(
    df_feat["danceability"], bins=[0, 0.33, 0.66, 1],
    labels=["Low", "Medium", "High"], include_lowest=True
)
```

Fig. 16. Danceability categorization.

4) Genre-Based Features: **1. Genre_list**

Some tracks contain multiple genres. We split combined genre strings into a list format, allowing analysis of cross-genre influences and better segmentation.

2. primary_genre

To simplify analysis, we chose the first or most dominant genre as the track's primary classification. This helped cleanly visualize distributions and focus on major categories.

3. genre_count

We counted the number of genres associated with each track. A higher count often indicates genre fusion or experimental compositions.

4. cross_genre indicator

This binary feature identifies whether a track spans more than one genre. It allowed us to analyze: whether cross-genre tracks tend to be more popular how common fusion genres are across the dataset.

5) Interaction & Composite Features: **1. energy_x_danceability**

We created this interaction term to capture the combined effect of a song being both energetic and danceable. High values typically align with upbeat hits commonly found in pop and EDM.

2. Acoustic_x_instrumental

This feature reflects how strongly a piece leans toward acoustic or instrumental qualities. It was especially useful in distinguishing organic genres (classical, unplugged) from heavily produced electronic ones.

3. complexity_score

Calculated as the mean of normalized acousticness, instrumentalness, and speechiness, this metric represents structural and acoustic complexity.

It helped us evaluate: how complex songs differ from mainstream tracks genre-specific complexity patterns.

```
Click to add a breakpoint | (mean of acousticness, instrumentalness, speechiness) ---
for c in ["acousticness", "instrumentalness", "speechiness"]:
    rng = df_feat[c].max() - df_feat[c].min()
    df_feat[c + "_norm"] = (df_feat[c] - df_feat[c].min()) / rng if rng != 0 else 0
df_feat["complexity_score"] = (
    df_feat["acousticness_norm"] + df_feat["instrumentalness_norm"] + df_feat["speechiness_norm"]
) / 3
```

Fig. 17. Complexity score calculation.

4. expressive_score

This composite score combines valence, energy, and danceability to reflect the overall expressive emotion of a track. Higher scores indicate lively, cheerful music, while lower scores correspond to calm or melancholic tracks.

```
# Combine expressive indicators
df_feat["expressive_score"] = (
    (1 - df_feat["instrumentalness"]) * 0.4 +
    df_feat["speechiness"] * 0.3 +
    df_feat["valence"] * 0.3
)
```

Fig. 18. Expressive score calculation.

5. acoustic_vs_electronic

We compared acousticness against electronic-leaning features to classify whether a song has a more natural or synthesized sound profile. This classification enriched our genre analysis and trend visualizations.

```
# Acoustic–Electronic axis
df_feat["acoustic_vs_electronic"] = df_feat["acousticness"] - df_feat["energy"]
```

Fig. 19. Acoustic vs electronic classification.

6. Popularity_norm

To account for skewed popularity distribution, we normalized the popularity scores. The normalized values supported fair comparison across genres and allowed efficient statistical modeling.

```
# Popularity normalized (0–1 scale)
pop_rng = df_feat["popularity"].max() - df_feat["popularity"].min()
df_feat["popularity_norm"] = (df_feat["popularity"] - df_feat["popularity"].min()) / pop_rng
```

Fig. 20. Popularity normalization.

Summary of Feature Engineering Impact

These engineered features transformed raw numerical audio metrics into meaningful categories and composite indicators that aligned with how listeners perceive music. They significantly improved the interpretability of visualizations and allowed us to discover trends such as: emotional tones favored by listeners how duration and tempo relate to popularity which genres dominate specific energy or mood categories acoustic vs. electronic trends in modern music. Overall, feature engineering strengthened the analytical foundation of our project and enabled deeper insights into Spotify's musical landscape.

D. Visualizations

For the below visualizations, we have categorized genres as these groups and created a calculated field on tableau and will be used throughout the dashboard.

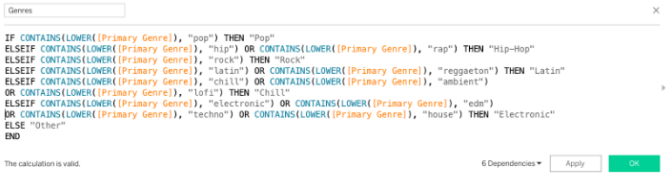

Fig. 21. Genre categorization for visualizations.

1) Genre Emotion Map: This Visualization helps us to understand how genres differ emotionally and how these differences might help in shaping listener preferences and popularity. The Emotion Map is created to visually represent how the different types of Spotify genres vary in their emotional profile. We are primarily showing 6 types of Genres that we have grouped based on their similarity and types. On this chart we are showing the genre's average energy and valence. These two features are useful for spotify audio analysis. Energy reflects intensity and activity, we get to know how loud or fast a song feels. Valence on the other hand is about emotional essence if a song is happy sounding music or is it sad or dark sound music. From this plot we are able to create a clear emotional mapping of Spotify's musical space across broader genres. This grouping helps to understand why certain genres consistently perform well in popularity while others serve more nice or mood specific purposes. The trend line highlights a positive relationship between Energy and Valence meaning genres that are more energetic also tend to be emotionally brighter. This reinforces why upbeat genres like Latin and Pop dominate the top-right quadrant, while calmer genres like Chill fall lower on both scales.

The axes add meaningful quadrant interpretation. The Upper-right represents Happy + Energetic emotion while the Lower-right quadrant represents Energetic + Dark emotion. The Upper-left represents Calm + Positive while Lower-left represents Sad + Calm.

We have also added filters to make the dashboard interactive and let users experience how emotional positioning changes with different filters like the Popularity filter lets users see if the hit songs group is different from non-hit tracks. Similarly the Mood filter helps to see the difference with Happy, Neutral, or Sad tracks to observe mood-driven shifts. The Is-Hit filter differentiates between the mainstream songs from general catalog tracks.

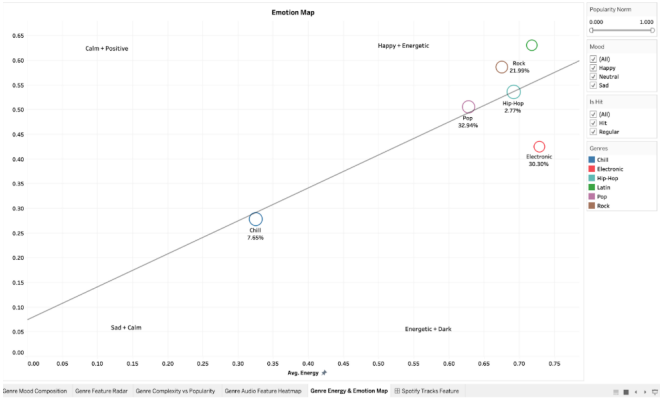

Fig. 22. Genre Emotion Map.

2) Audio features heatmap: This visualization compares different genres across audio features like acousticness, danceability, energy, instrumentalness, speechiness and valence. Different shades/ intensity of the same color is used to show the value of these features. Darker or deeper shade shows higher values while lighter intensity shows lower values.

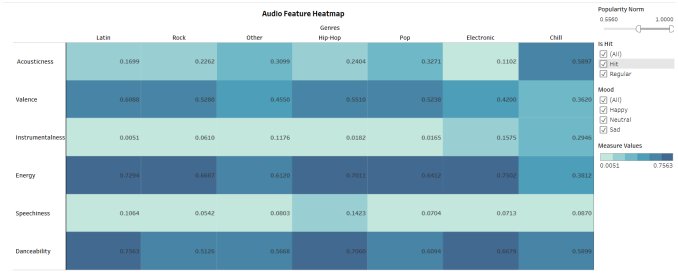

Fig. 23. Audio Features Heatmap.

Key findings:

- **Electronic**: Low acousticness, and high danceability and energy.
- **Chill**: High acousticness but low energy and valence.
- **Hip-Hop**: High speechiness and danceability.
- **Latin**: High valence and danceability.
- **Rock**: Lower danceability and medium level energy
- **Pop**: Low acousticness but high energy.
- **Other**: Mixed

3) Mood Distribution: The graph visualizes how different mood categories - Happy, Neutral, and Sad are distributed across selected music genres such as Latin, Hip-Hop, Rock, Pop, Electronic, and Chill. Each bar is stacked to show the percentage share of each mood within that genre, giving a clear comparison of emotional tone across musical styles.

The mood categories were generated from the valence score (a Spotify audio feature that describes how positive or uplifting a track sounds).

- High valence - Happy
- Medium valence - Neutral
- Low valence - Sad

The visualization also includes interactive filters (e.g., Hit vs Regular songs, Popularity range), but the primary focus remains on mood distribution across genres.

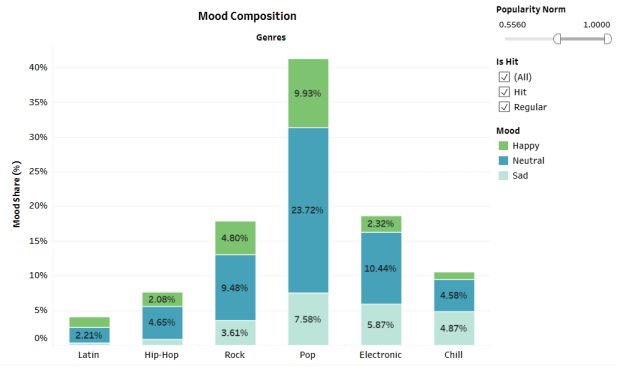

Fig. 24. Mood Distribution across Genres.

Key Findings:

1) **Pop music is the most emotionally diverse genre**
   Pop stands out with the highest percentages across all mood categories. For example, Neutral mood alone accounts for 23.72% of pop tracks, followed by 9.93% Happy and 7.58% Sad. This emotional variety is likely a reason pop reaches the widest audience.
2) **Neutral mood dominates across most genres**
   Almost every genre has Neutral as its leading mood category. For instance, Electronic shows 10.44% Neutral, while Rock has 9.48% Neutral. This suggests listeners tend to prefer tracks that are neither extremely sad nor overly happy, but rather emotionally moderate.
3) **Happy tracks are generally less common than Neutral ones**
   Happy mood percentages are noticeably lower across many genres. For example, Hip-Hop has only around 2% Happy tracks, and Electronic is similar. Emotional positivity appears less frequent than neutrality or mild sadness in mainstream music trends.
4) **Electronic and Hip-Hop lean toward emotionally neutral-to-serious tones**
   Both genres show comparatively low Happy percentages. Their mood distribution reflects the stylistic nature of these genres:
   - Electronic → more rhythmic and atmospheric
   - Hip-Hop → expressive, narrative-driven themes
5) **Sad mood tracks appear consistently but not dominantly**
   Sad tracks do not dominate any genre but remain present in most of them. Pop, for example, has 7.58% Sad tracks, while Electronic has 5.87% Sad, showing that emotional depth plays a constant—though secondary—role in modern music.
6) **Mood patterns vary noticeably by genre, reflecting listener expectations**
   - Pop → wide emotional range

- Rock → expressive and varied
- Electronic → stable, mostly neutral mood
- Hip-Hop → reflective or serious tones
- Chill → balanced emotional profile

4) Genres Complexity vs Popularity: This chart tells the story of how ”musical complexity” and ”popularity” get across different genres, they don't always get along very well. The complexity score is built from acousticness, instrumentalness, and speechiness, which together tell us how textured a song feels. Popularity reflects how much people actually play these songs on Spotify. When you look across all the genres, a clear pattern pops out: the simpler a song is, the more the crowd seems to love it. Take Latin, Pop, and Rock, for example. These genres are the lightweights in terms of complexity; their scores sit around 0.08 to 0.12. But popularity-wise? They're the superstars, landing between 0.75 and 0.78. It fits perfectly with what we hear on the radio every day. People love catchy, easy-to-digest songs that get stuck in your head. Then you move into Hip-Hop and Electronic, which live in the ”moderately complex but still very likable” zone. They bring in more layers, samples, beats, creative production but they still keep listeners hooked. A little complexity doesn't scare anyone away here. And then there's Chill the outlier. Chill music scores the highest on complexity, around 0.259 it drops to one of the lowest popularity levels at about 0.73. Chill tracks are moodier, more atmospheric, sometimes instrumental. They're beautiful but not always mainstream material.

On the right side of the dashboard, the Popularity Norm slider gives us control over which songs we're focusing on. Since popularity has been scaled between 0 and 1, the slider lets us zoom in on just the top hits or widen our view to include lesser-known tracks. Slide it right to look at only the biggest bangers, slide it left to see everything. It's a simple way to test whether the complexity-popularity trend holds true for both the viral songs and the quiet underdogs.

The visualization shows mainstream music stays simple, while complex music tends to find a smaller, more niche audience and the slider helps us explore that relationship from every angle.

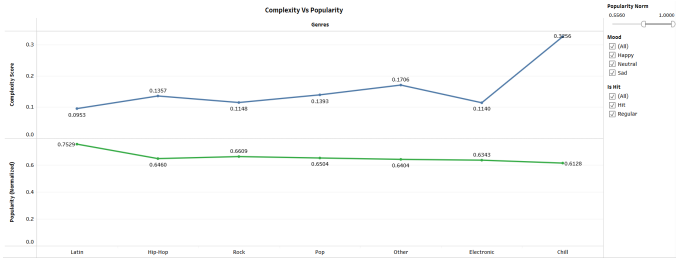

Fig. 25. Genres Complexity vs Popularity.

5) Genre feature Radar chart: **Radar chart** explains how musical characters differs across genre, popularity, mood. In the above chart, Radar shows the combined audio features of Pop and Rock genre. It reflects high energy and strong danceability as Pop and Rock shows rhythmic structure. Instrumentalness is low, indicating that most of the tracks are vocal driven rather than purely instrumental.

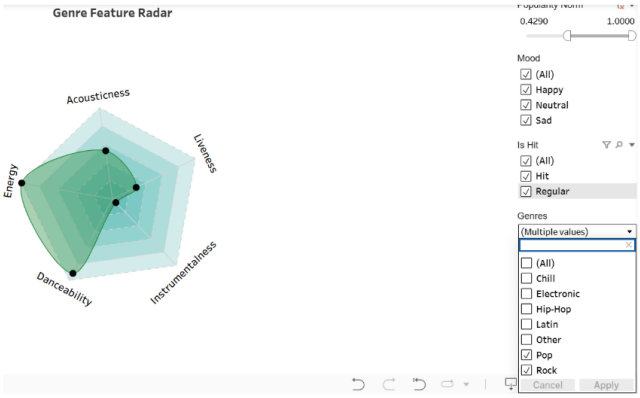

Fig. 26. Genre Feature Radar Chart - Pop and Rock.

When the Sad mood category is applied, there is an immediate drop in energy and danceability as it tends to be more calmer. Liveness remains moderate, reflecting the mix of emotions. Overall sad songs shows low energy and low danceability.

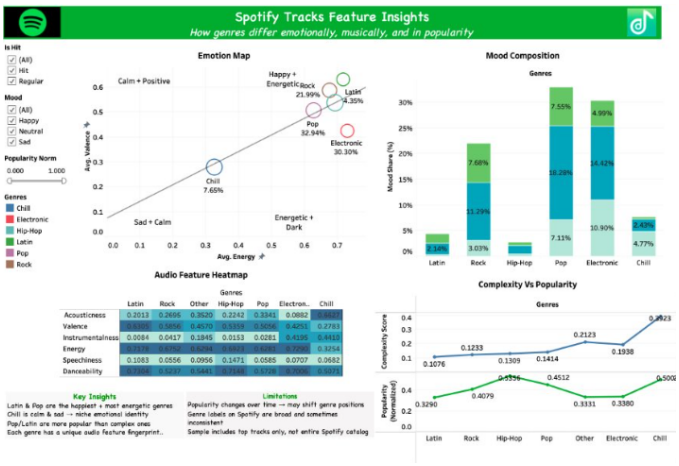

Fig. 27. Final Tableau Dashboard

REFERENCES

[1] Spotify Tracks Dataset, Kaggle, Available: https://www.kaggle.com/
[2] Spotify Web API Documentation, Available: https://developer.spotify.com/documentation/web-api/
[3] Tableau Dashboard: https://public.tableau.com/app/profile/mamta.jha/viz/Final230/Spoti