SPOTIFY
DATABASE
WRAPPED.

```
# IT REALLY ISN’T WRAPPED.
```

TABLE OF CONTENTS.

01

Introduction.

02

Describing the dataset.

03

Discussing the code.

04

Vizualising the main findings.

05

Summary & Challenges.

06

Q&A

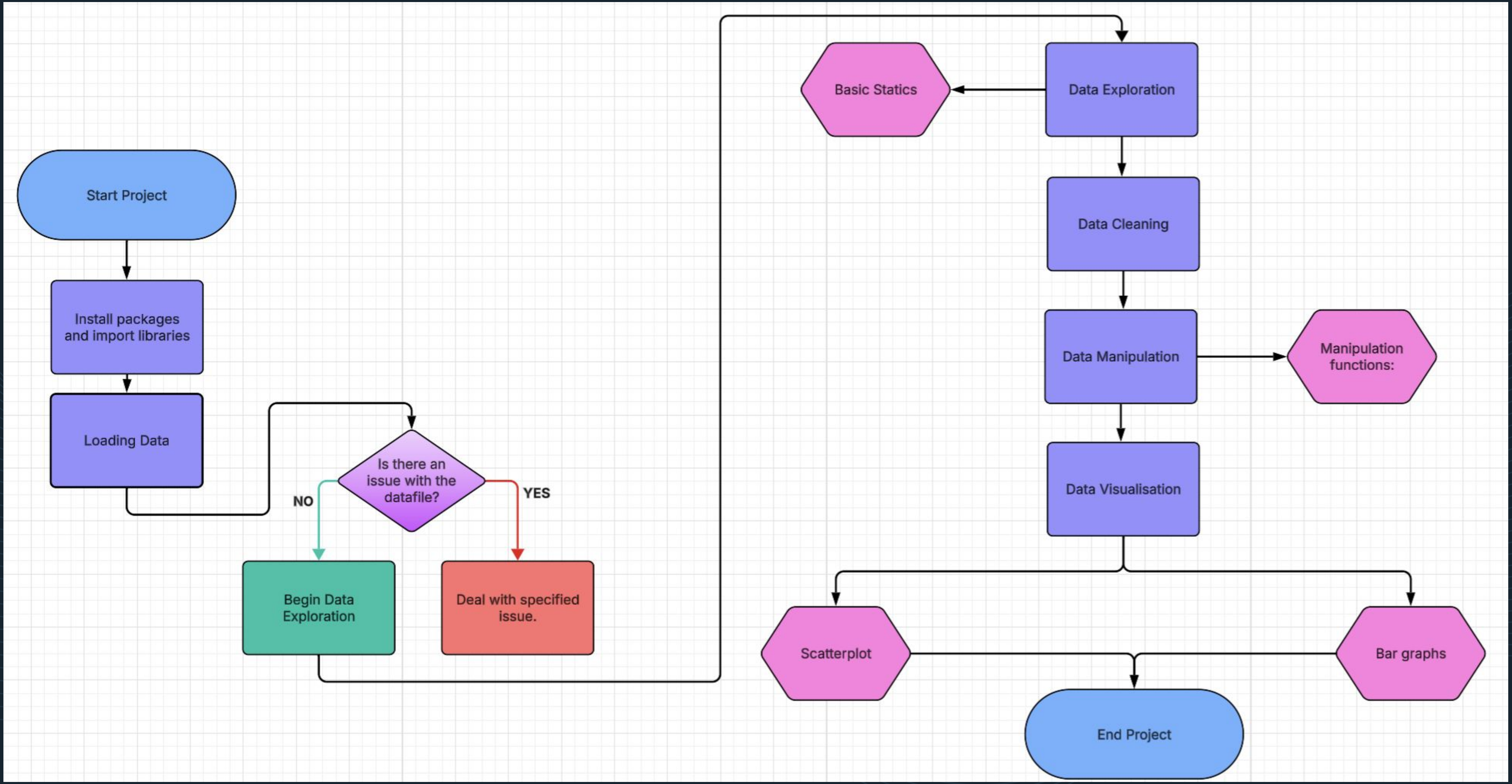
DESCRIBING THE DATASET

A Spotify music dataset: 114k tracks with metadata (artist, album, genre) and audio features (danceability, energy, valence, tempo, etc.), plus popularity, duration, and explicit flag,

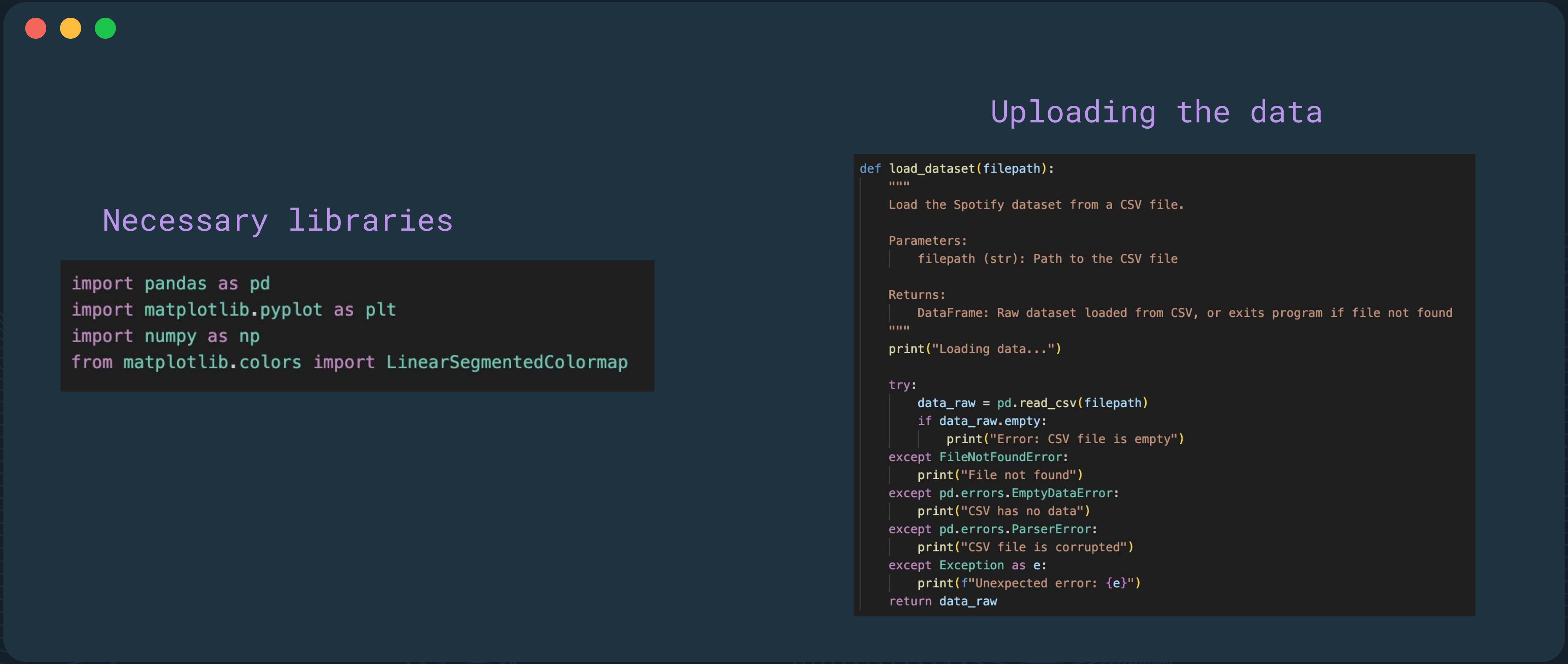
It’s perfect to explore what drives song popularity, compare genres, and build clear visuals/models about musical traits and trends,

Kaggle & Trial & Error.

DATA PROCESSING FLOWCHART



LOADING



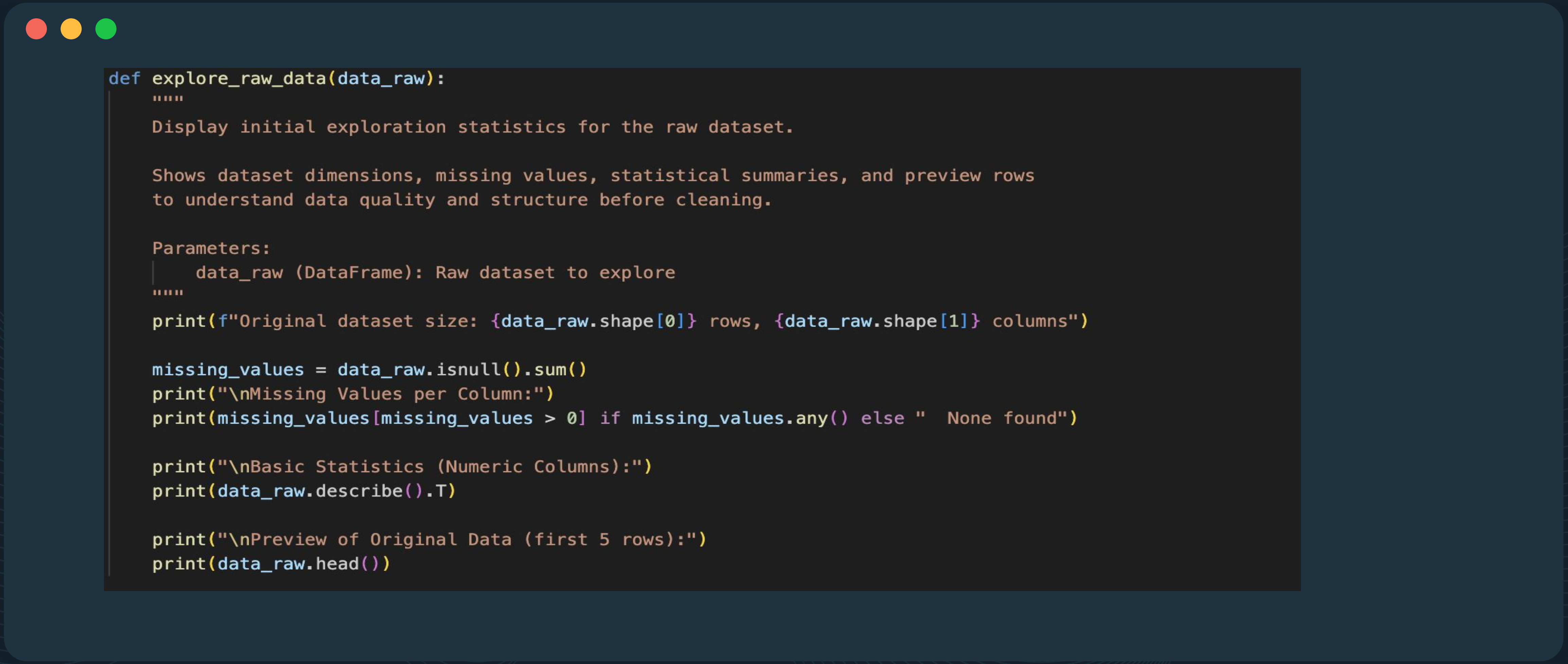
Uploading the data

Necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import LinearSegmentedColormap
```

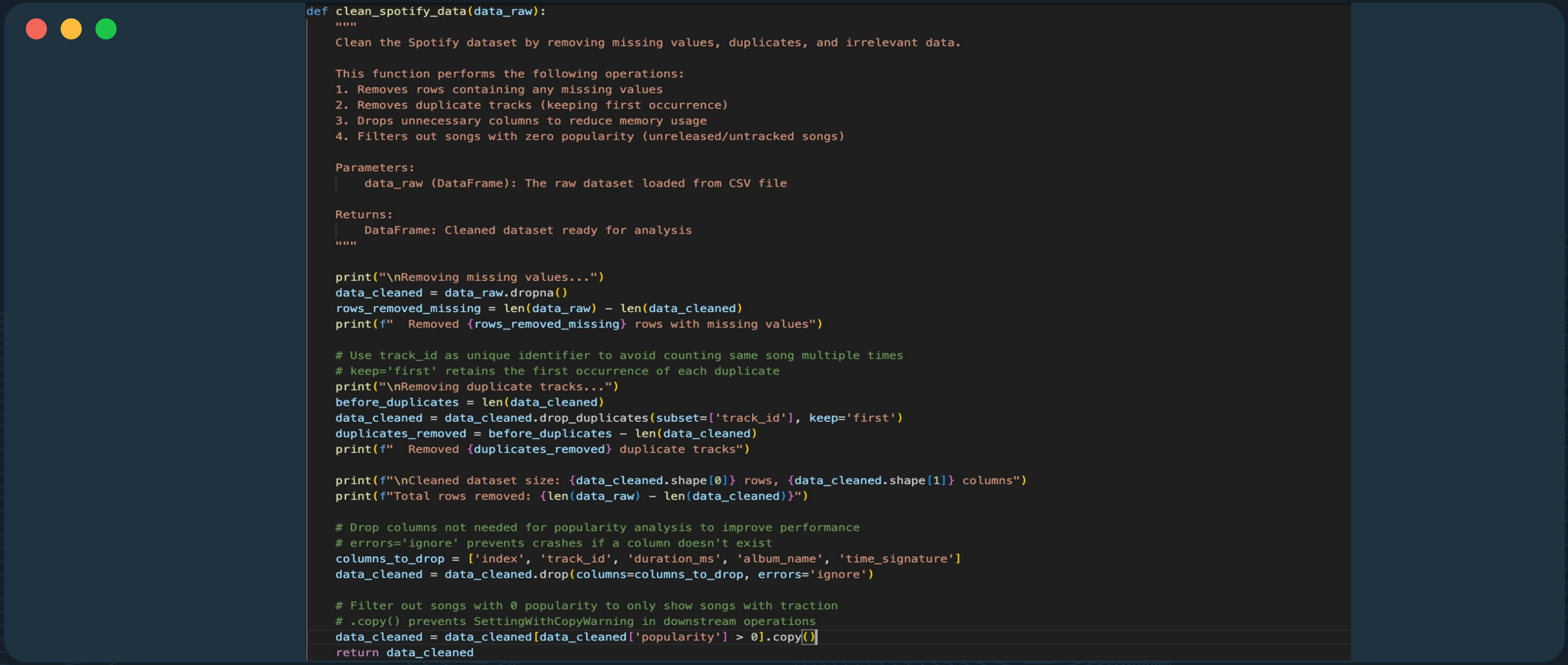
```
def load_dataset(filepath):
    """
    Load the Spotify dataset from a CSV file.

    Parameters:
        filepath (str): Path to the CSV file

    Returns:
        DataFrame: Raw dataset loaded from CSV, or exits program if file not found
    """
    print("Loading data...")

    try:
        data_raw = pd.read_csv(filepath)
        if data_raw.empty:
            print("Error: CSV file is empty")
    except FileNotFoundError:
        print("File not found")
    except pd.errors.EmptyDataError:
        print("CSV has no data")
    except pd.errors.ParserError:
        print("CSV file is corrupted")
    except Exception as e:
        print(f"Unexpected error: {e}")
    return data_raw
```

EXPLORING



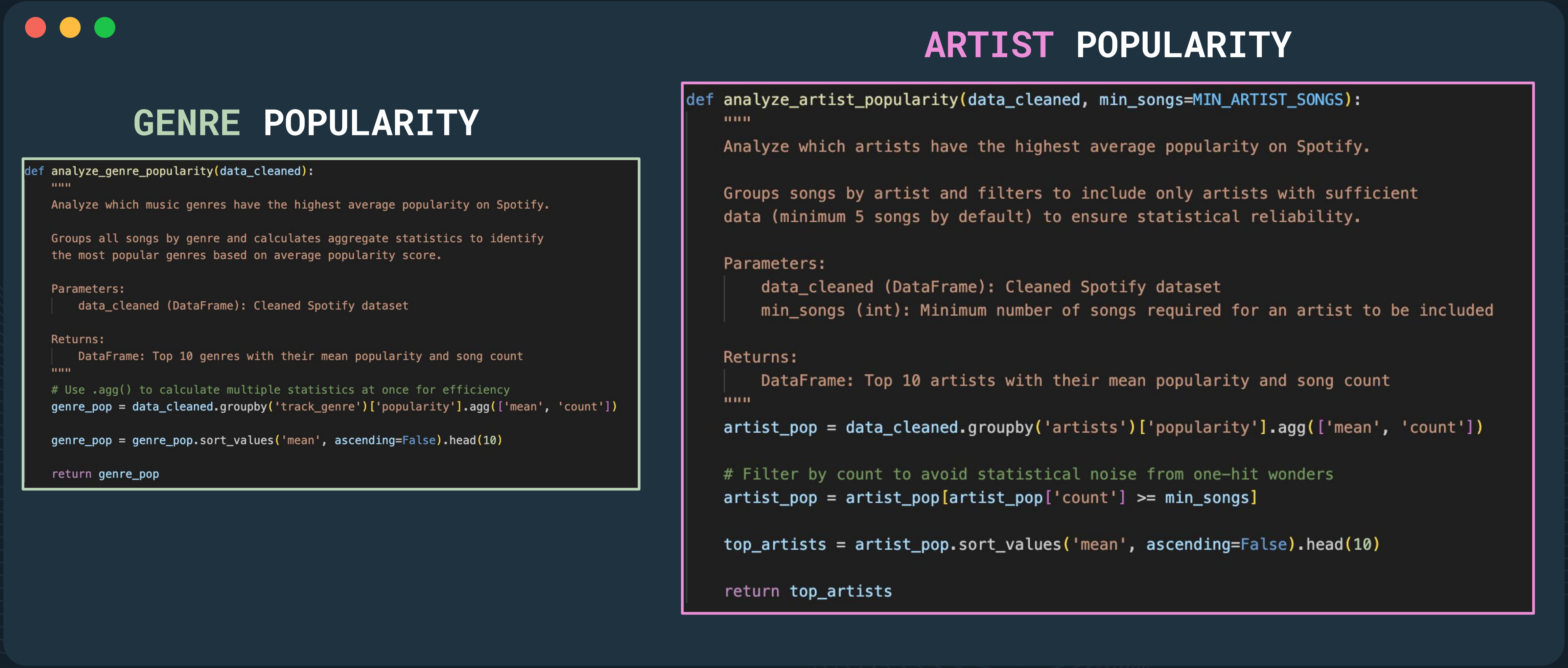
```python
def explore_raw_data(data_raw):
    """
    Display initial exploration statistics for the raw dataset.

    Shows dataset dimensions, missing values, statistical summaries, and preview rows
    to understand data quality and structure before cleaning.

    Parameters:
        data_raw (DataFrame): Raw dataset to explore
    """
    print(f"Original dataset size: {data_raw.shape[0]} rows, {data_raw.shape[1]} columns")

    missing_values = data_raw.isnull().sum()
    print("\nMissing Values per Column:")
    print(missing_values[missing_values > 0] if missing_values.any() else "  None found")

    print("\nBasic Statistics (Numeric Columns):")
    print(data_raw.describe().T)

    print("\nPreview of Original Data (first 5 rows):")
    print(data_raw.head())
```

CLEANING



```python
def clean_spotify_data(data_raw):
    """
    Clean the Spotify dataset by removing missing values, duplicates, and irrelevant data.

    This function performs the following operations:
    1. Removes rows containing any missing values
    2. Removes duplicate tracks (keeping first occurrence)
    3. Drops unnecessary columns to reduce memory usage
    4. Filters out songs with zero popularity (unreleased/untracked songs)

    Parameters:
        data_raw (DataFrame): The raw dataset loaded from CSV file

    Returns:
        DataFrame: Cleaned dataset ready for analysis
    """

    print("\nRemoving missing values...")
    data_cleaned = data_raw.dropna()
    rows_removed_missing = len(data_raw) - len(data_cleaned)
    print(f"  Removed {rows_removed_missing} rows with missing values")

    # Use track_id as unique identifier to avoid counting same song multiple times
    # keep='first' retains the first occurrence of each duplicate
    print("\nRemoving duplicate tracks...")
    before_duplicates = len(data_cleaned)
    data_cleaned = data_cleaned.drop_duplicates(subset=['track_id'], keep='first')
    duplicates_removed = before_duplicates - len(data_cleaned)
    print(f"  Removed {duplicates_removed} duplicate tracks")

    print(f"\nCleaned dataset size: {data_cleaned.shape[0]} rows, {data_cleaned.shape[1]} columns")
    print(f"Total rows removed: {len(data_raw) - len(data_cleaned)}")

    # Drop columns not needed for popularity analysis to improve performance
    # errors='ignore' prevents crashes if a column doesn't exist
    columns_to_drop = ['index', 'track_id', 'duration_ms', 'album_name', 'time_signature']
    data_cleaned = data_cleaned.drop(columns=columns_to_drop, errors='ignore')

    # Filter out songs with 0 popularity to only show songs with traction
    # .copy() prevents SettingWithCopyWarning in downstream operations
    data_cleaned = data_cleaned[data_cleaned['popularity'] > 0].copy()
    return data_cleaned
```

LET'S MANIPULATE BASIC DATA.



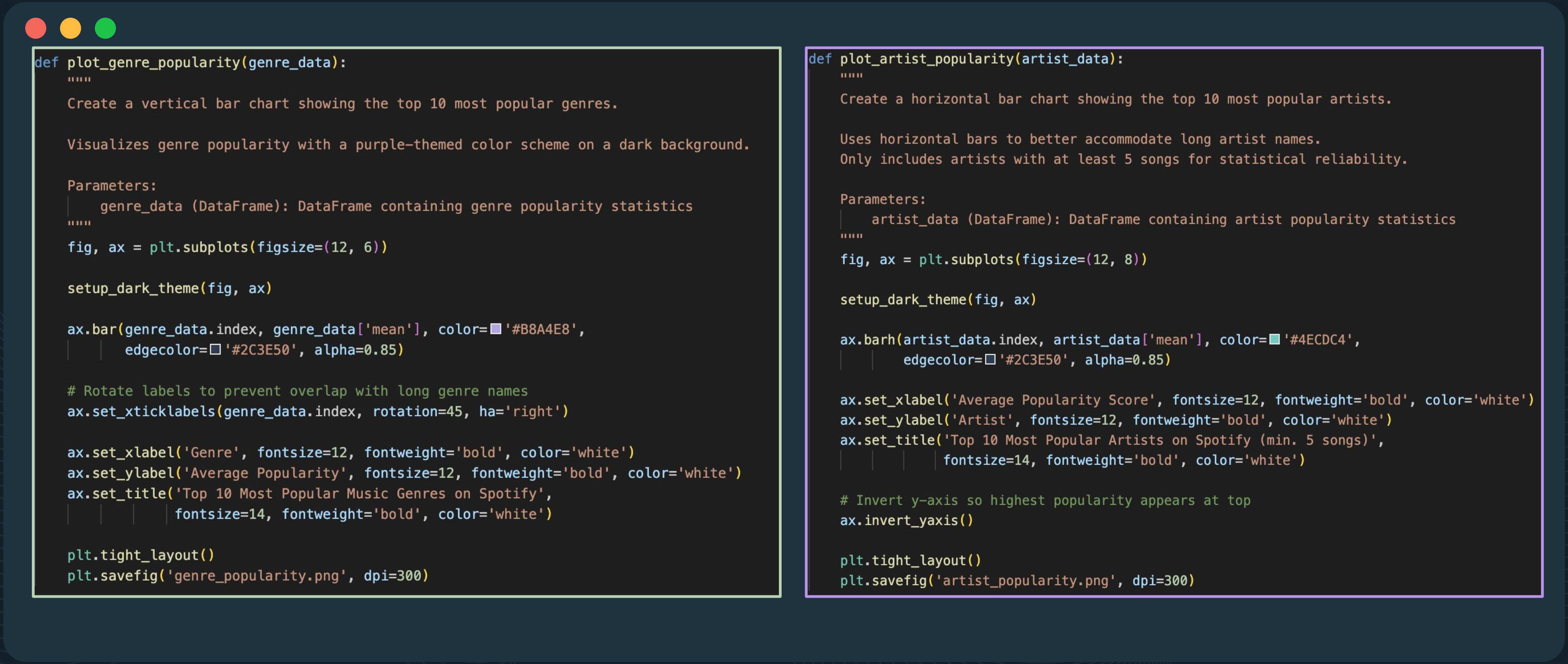
GENRE POPULARITY

```python
def analyze_genre_popularity(data_cleaned):
    """
    Analyze which music genres have the highest average popularity on Spotify.

    Groups all songs by genre and calculates aggregate statistics to identify
    the most popular genres based on average popularity score.

    Parameters:
        data_cleaned (DataFrame): Cleaned Spotify dataset

    Returns:
        DataFrame: Top 10 genres with their mean popularity and song count
    """
    # Use .agg() to calculate multiple statistics at once for efficiency
    genre_pop = data_cleaned.groupby('track_genre')['popularity'].agg(['mean', 'count'])

    genre_pop = genre_pop.sort_values('mean', ascending=False).head(10)

    return genre_pop
```

ARTIST POPULARITY

```python
def analyze_artist_popularity(data_cleaned, min_songs=MIN_ARTIST_SONGS):
    """
    Analyze which artists have the highest average popularity on Spotify.

    Groups songs by artist and filters to include only artists with sufficient
    data (minimum 5 songs by default) to ensure statistical reliability.

    Parameters:
        data_cleaned (DataFrame): Cleaned Spotify dataset
        min_songs (int): Minimum number of songs required for an artist to be included

    Returns:
        DataFrame: Top 10 artists with their mean popularity and song count
    """
    artist_pop = data_cleaned.groupby('artists')['popularity'].agg(['mean', 'count'])

    # Filter by count to avoid statistical noise from one-hit wonders
    artist_pop = artist_pop[artist_pop['count'] >= min_songs]

    top_artists = artist_pop.sort_values('mean', ascending=False).head(10)

    return top_artists
```

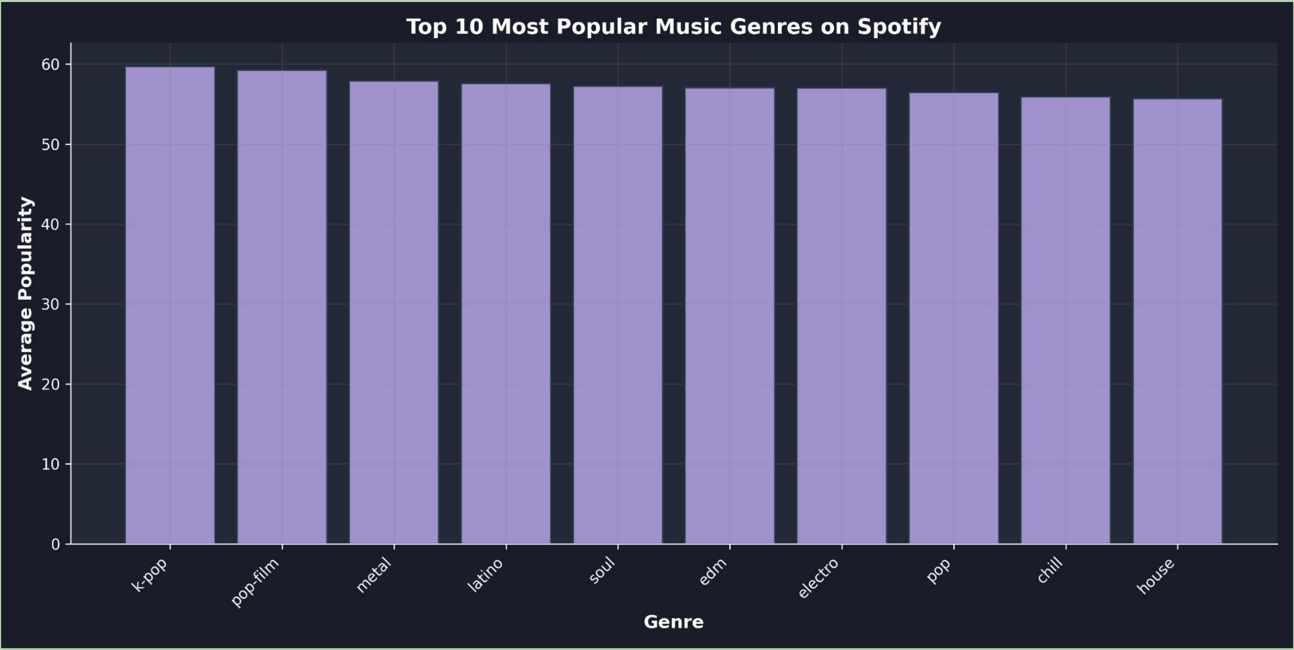
VISUALIZING



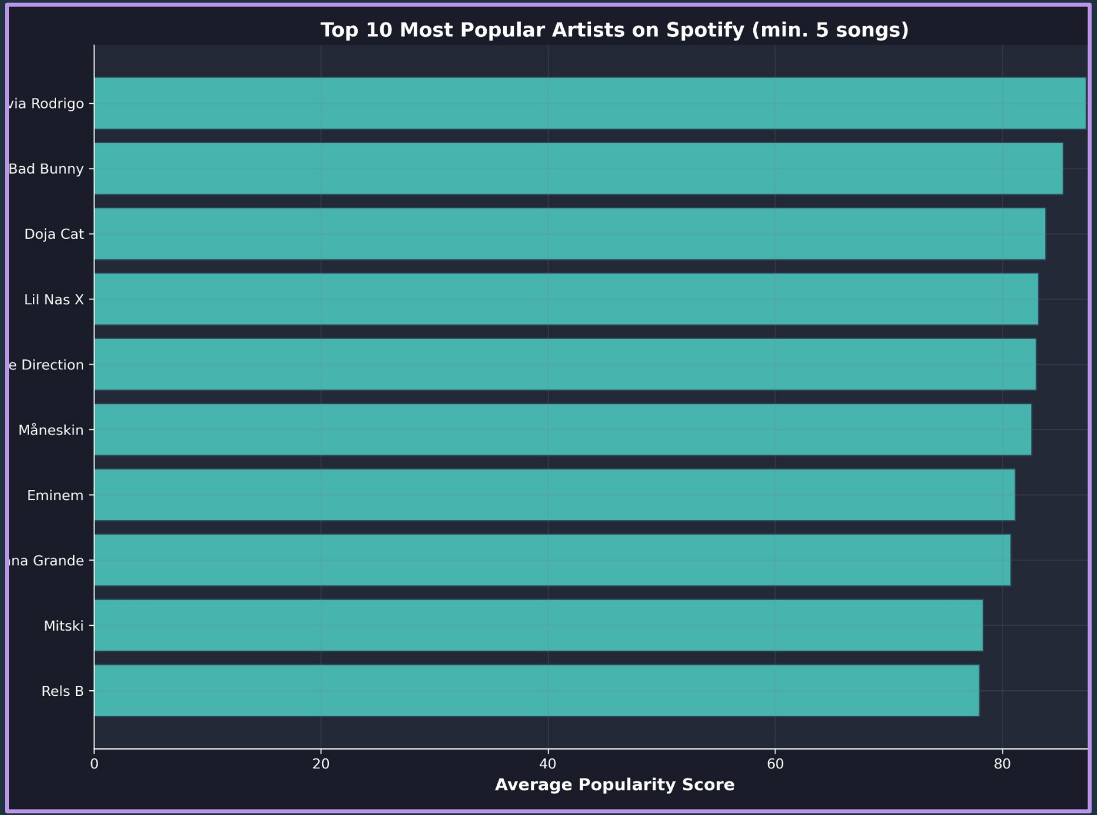
```python
def plot_genre_popularity(genre_data):
    """
    Create a vertical bar chart showing the top 10 most popular genres.

    Visualizes genre popularity with a purple-themed color scheme on a dark background.

    Parameters:
        genre_data (DataFrame): DataFrame containing genre popularity statistics
    """
    fig, ax = plt.subplots(figsize=(12, 6))

    setup_dark_theme(fig, ax)

    ax.bar(genre_data.index, genre_data['mean'], color=■'#B8A4E8',
           edgecolor=■'#2C3E50', alpha=0.85)

    # Rotate labels to prevent overlap with long genre names
    ax.set_xticklabels(genre_data.index, rotation=45, ha='right')

    ax.set_xlabel('Genre', fontsize=12, fontweight='bold', color='white')
    ax.set_ylabel('Average Popularity', fontsize=12, fontweight='bold', color='white')
    ax.set_title('Top 10 Most Popular Music Genres on Spotify',
                 fontsize=14, fontweight='bold', color='white')

    plt.tight_layout()
    plt.savefig('genre_popularity.png', dpi=300)
```

```python
def plot_artist_popularity(artist_data):
    """
    Create a horizontal bar chart showing the top 10 most popular artists.

    Uses horizontal bars to better accommodate long artist names.
    Only includes artists with at least 5 songs for statistical reliability.

    Parameters:
        artist_data (DataFrame): DataFrame containing artist popularity statistics
    """
    fig, ax = plt.subplots(figsize=(12, 8))

    setup_dark_theme(fig, ax)

    ax.barh(artist_data.index, artist_data['mean'], color=■'#4ECDC4',
            edgecolor=■'#2C3E50', alpha=0.85)

    ax.set_xlabel('Average Popularity Score', fontsize=12, fontweight='bold', color='white')
    ax.set_ylabel('Artist', fontsize=12, fontweight='bold', color='white')
    ax.set_title('Top 10 Most Popular Artists on Spotify (min. 5 songs)',
                 fontsize=14, fontweight='bold', color='white')

    # Invert y-axis so highest popularity appears at top
    ax.invert_yaxis()

    plt.tight_layout()
    plt.savefig('artist_popularity.png', dpi=300)
```

VISUALIZING

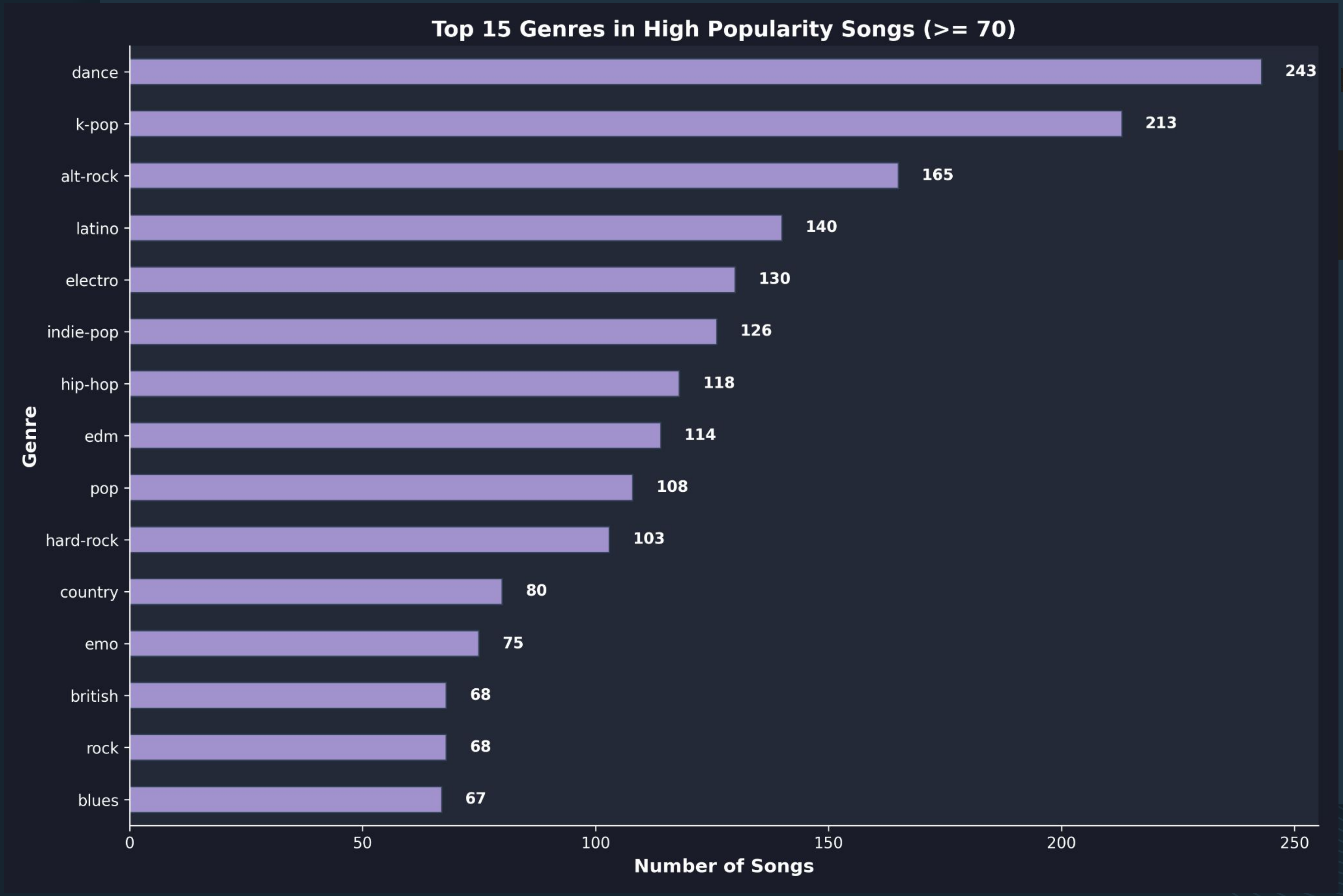


Top 10 Most Popular Music Genres on Spotify



Top 10 Most Popular Artists on Spotify (min. 5 songs)

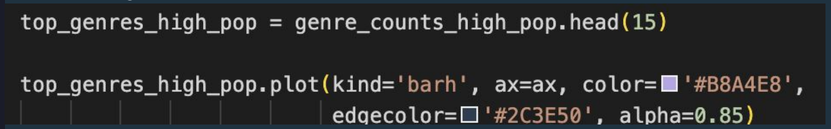
ANALYZING



Filter for “hit songs” (popularity >= 70):

```
high_pop_songs = data_cleaned[data_cleaned['popularity'] >= threshold]
```

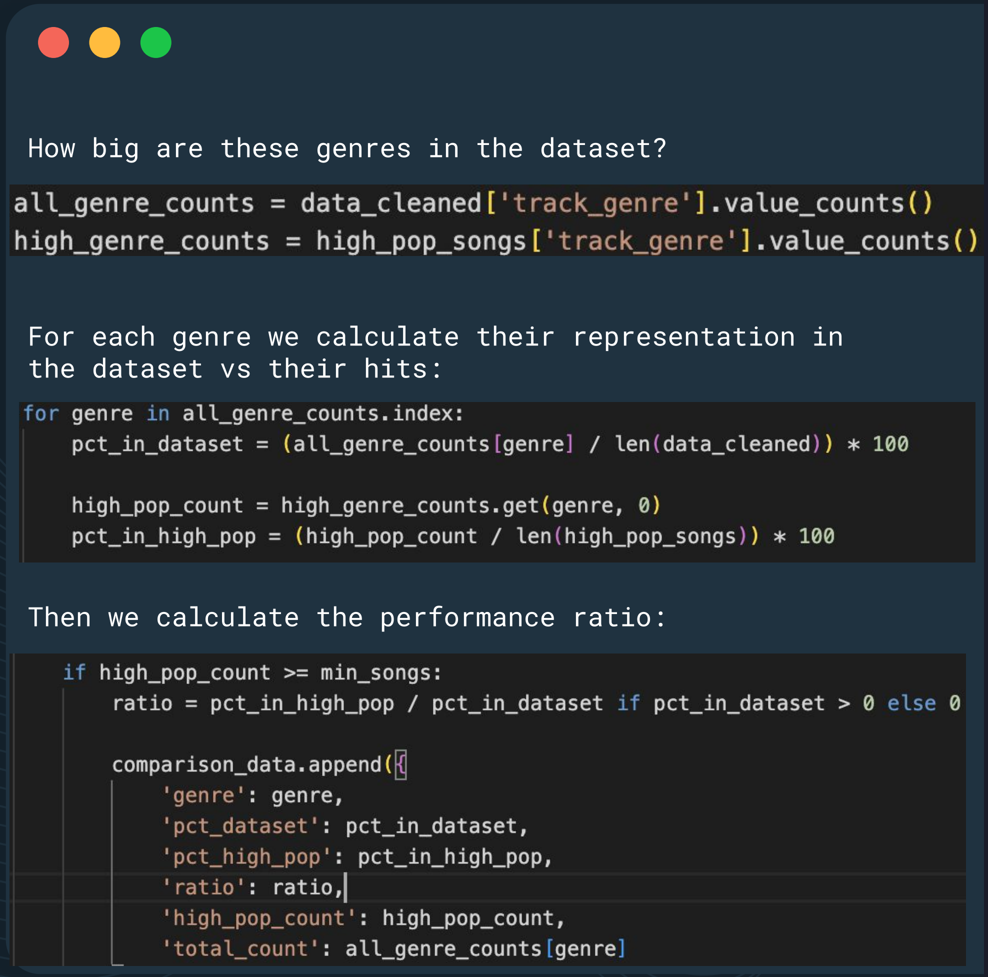
Which genres dominate the hits?



```
top_genres_high_pop = genre_counts_high_pop.head(15)

top_genres_high_pop.plot(kind='barh', ax=ax, color=■'#B8A4E8',
                         edgecolor=□'#2C3E50', alpha=0.85)
```

Dance and k-pop seem considerably large in the hit category. But are they actually **GOOD** at producing hits?

THE MISMATCH



How big are these genres in the dataset?

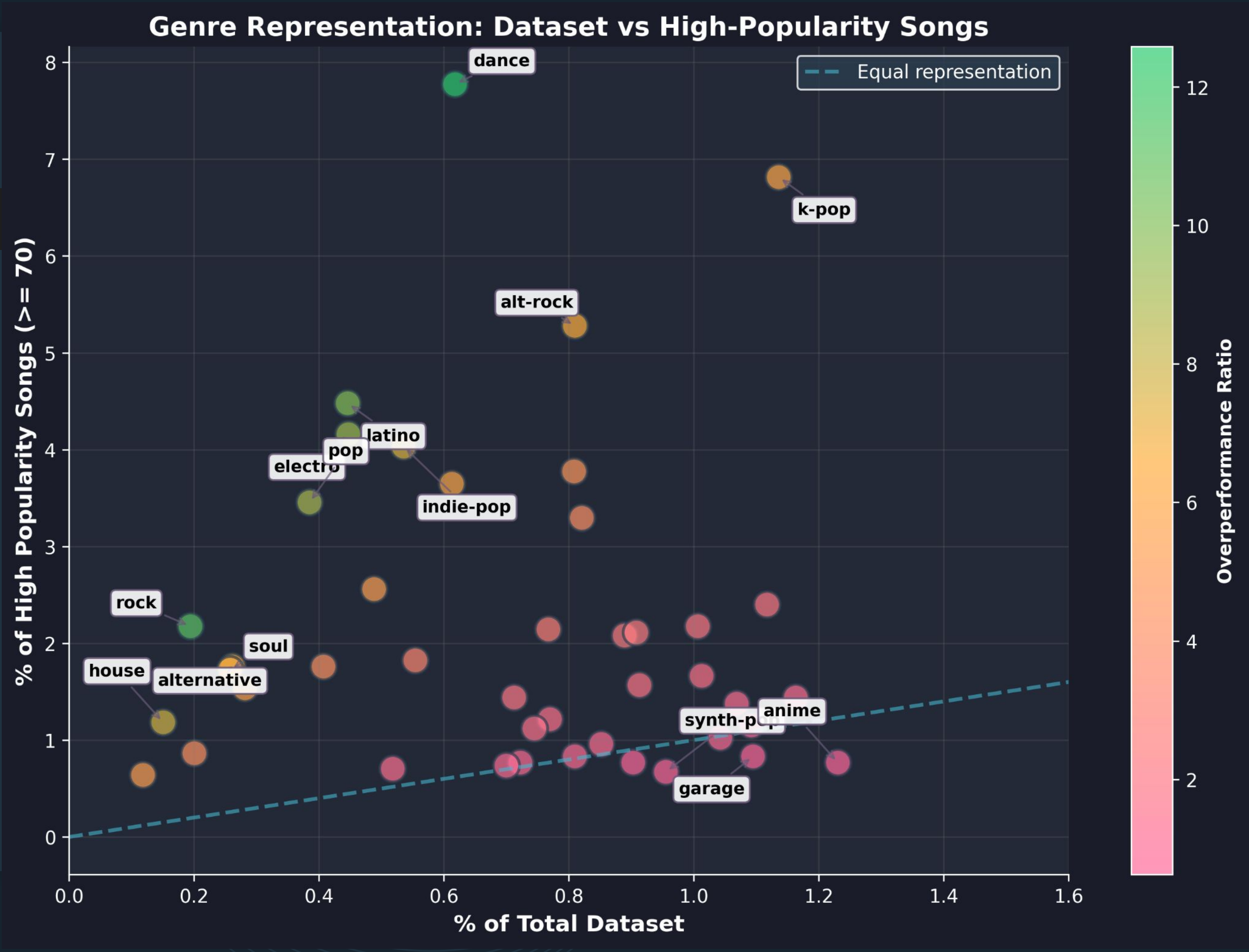
```
all_genre_counts = data_cleaned['track_genre'].value_counts()
high_genre_counts = high_pop_songs['track_genre'].value_counts()
```

For each genre we calculate their representation in the dataset vs their hits:

```
for genre in all_genre_counts.index:
    pct_in_dataset = (all_genre_counts[genre] / len(data_cleaned)) * 100

    high_pop_count = high_genre_counts.get(genre, 0)
    pct_in_high_pop = (high_pop_count / len(high_pop_songs)) * 100
```

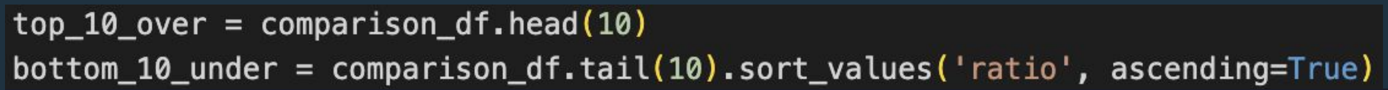
Then we calculate the performance ratio:

```
    if high_pop_count >= min_songs:
        ratio = pct_in_high_pop / pct_in_dataset if pct_in_dataset > 0 else 0

        comparison_data.append({
            'genre': genre,
            'pct_dataset': pct_in_dataset,
            'pct_high_pop': pct_in_high_pop,
            'ratio': ratio,
            'high_pop_count': high_pop_count,
            'total_count': all_genre_counts[genre]
```

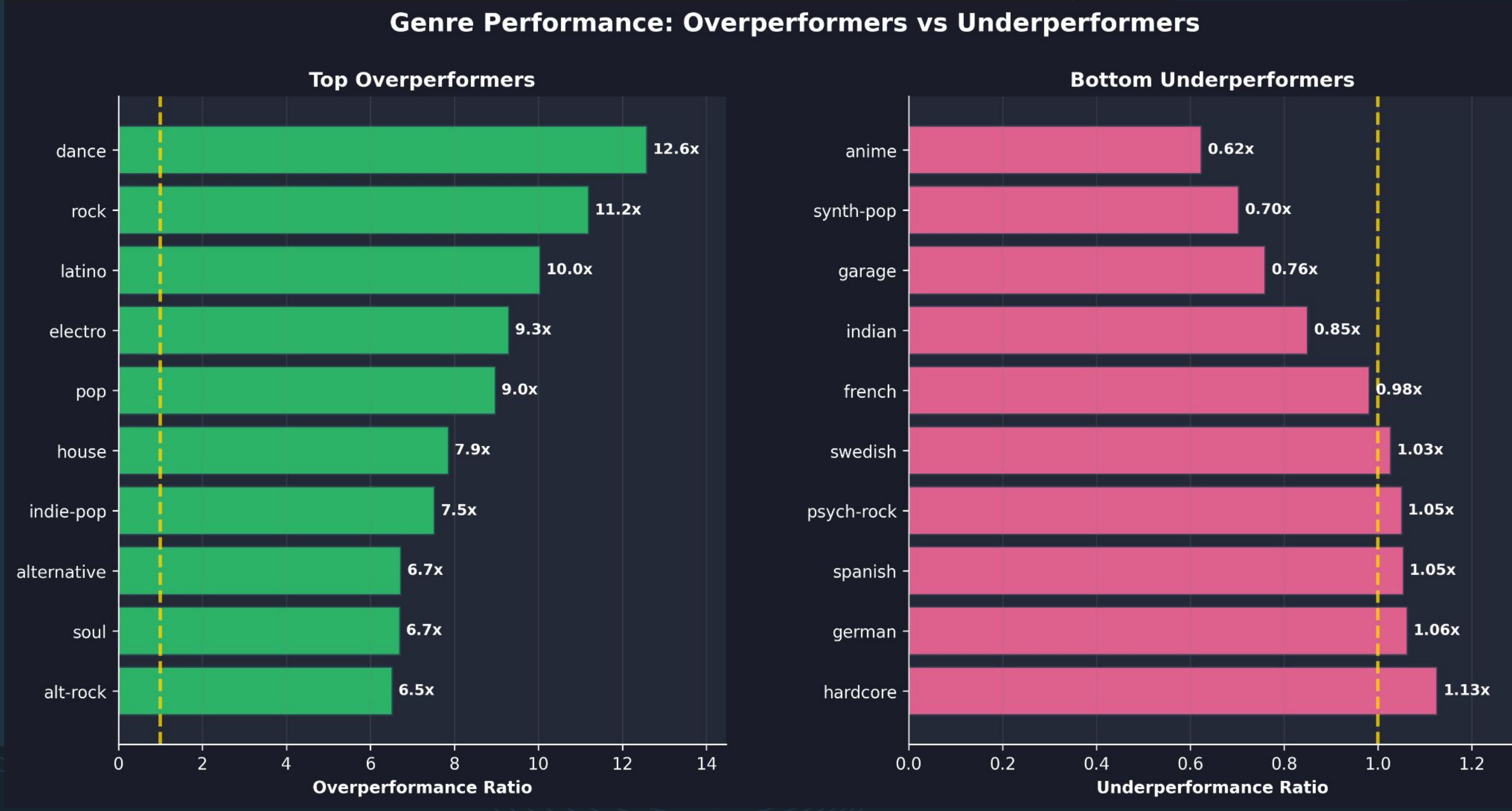


GENRE PERFORMANCE

- **Dance: 12.6x ratio** → Only 0.6% of dataset, but 7.9% of hits
- **K-pop: 6.0x ratio**, despite being 6.8% of hits
- **Anime: 0.62x** ratio, Struggles to break into mainstream



```
top_10_over = comparison_df.head(10)
bottom_10_under = comparison_df.tail(10).sort_values('ratio', ascending=True)
```



Genre Performance: Overperformers vs Underperformers

SUMMARY

CONCLUSION

What makes a **Hit** song?

- Specific genres and artists
- Dance music is the most likely genre to be a hit

Presence != Success

- Larger genres often have lower ratios

CHALLENGES

1. Defining **‘High Popularity’**
   - >= 60 is 12.2% of the songs - 9,806 songs
   - >= 70 is 3.9% of the songs - 3,126 songs
2. Finding strong correlations for **Popularity**
   - Audio features have little impact on popularity

   = Shifted focus to genre efficiency instead
3. Making the code **Readable** and **Generalized**
   - Used modular functions
   - Robust error handling for possible other datasets
   - Configurable constants at top of file

THANK YOU!

Do you have any questions?

Patrick Verschoor
Camdon Watson
Ionuț Marian