Music Recommendation Dataset

The dataset analyzed provides a comprehensive view of user listening patterns, offering insights into musical preferences, artist popularity, and play count distributions. With over 102,627 entries spanning multiple years, the dataset captures a diverse range of genres, songs, and user behaviors, enabling a deep dive into the dynamics of digital music consumption. From identifying the most-played songs and popular artists to uncovering notable trends in user engagement, the analysis highlights the long-tail distribution typical of digital platforms. Additionally, temporal patterns in music play counts and user segmentation based on activity levels reveal intriguing insights into listener behavior and content popularity. This study not only showcases diverse musical preferences but also underlines the role of power users in shaping overall platform activity.

Dataset Overview

The dataset contains 102,627 entries with information about user listening patterns, including song details, artists, and play counts. The data spans multiple years and includes various music genres.

	user	song	play_count	title	release	artist_name	year
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cove	Thicker Than Water	Jack Johnson	0
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAPDEY12A81C210A9	1	Nothing from Nothing	To Die For	Billy Preston	1974
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Dos Aguas	Flamenco Para Niños	Paco De Lucia	1976
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBFNSP12AF72A0E22	1	Under Cold Blue Stars	Under Cold Blue Stars	Josh Rouse	2002
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBFOVM12A58A7D494	1	Riot Radio (Soundtrack Version)	Nick & Norah's Infinite Playlist - Original Motion Picture Soundtrack	The Dead 60s	0

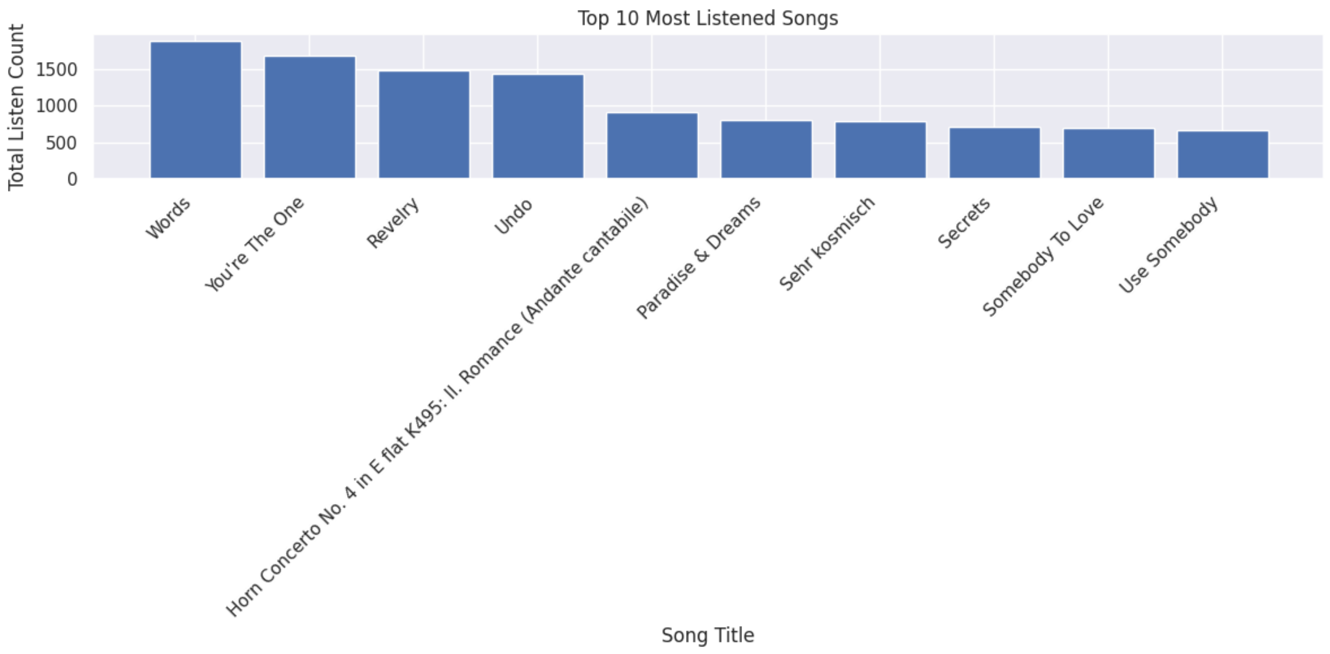
Most Listened Songs

Top Songs by Play Count:

- "Words" by Jack the Ripper (1,890 plays)
- "You're The One" by Dwight Yoakam (1,689 plays)
- "Revelry" by Kings Of Leon (1,490 plays)
- "Undo" by Björk (1,443 plays)
- "Horn Concerto No. 4" by Barry Tuckwell (921 plays)

Analysis:

- Diverse genre representation in top songs
- Mix of mainstream and niche artists
- High concentration of plays in top songs suggests strong listener preferences



	title	artist_name	play_count
43052	Words	Jack the Ripper	1890
43833	You're The One	Dwight Yoakam	1689
30103	Revelry	Kings Of Leon	1490
40415	Undo	Björk	1443
16038	Horn Concerto No. 4 in E flat K495: II. Romance (Andante cantabile)	Barry Tuckwell/Academy of St Martin-in-the-Fields/Sir Neville Marriner	921
27738	Paradise & Dreams	Darren Styles	805
31616	Sehr kosmisch	Harmonia	795
31557	Secrets	OneRepublic	712
33512	Somebody To Love	Justin Bieber	692
40636	Use Somebody	Kings Of Leon	660

Most Popular Artists

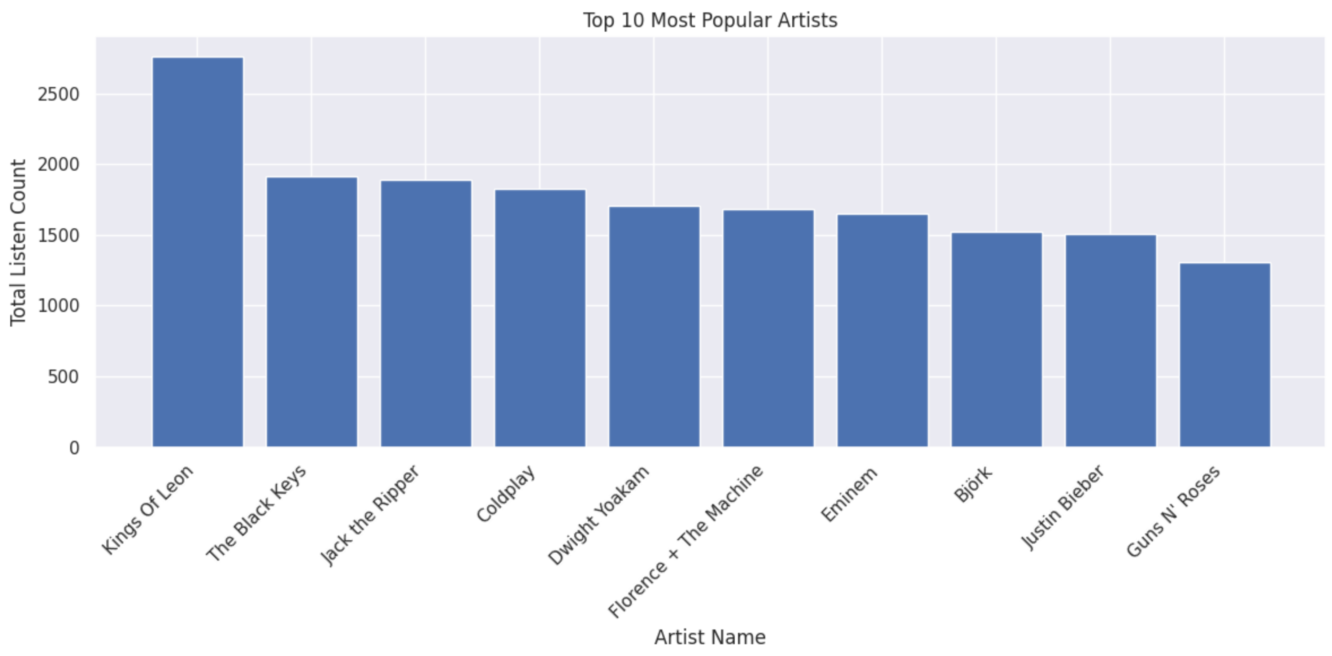
Top Artists by Total Plays:

- Kings Of Leon (2,765 plays)
- The Black Keys (1,912 plays)
- Jack the Ripper (1,890 plays)
- Coldplay (1,830 plays)
- Dwight Yoakam (1,705 plays)

Analysis:

- Rock bands dominate the top spots
- Mix of contemporary and classic artists

- Strong presence of alternative/indie artists



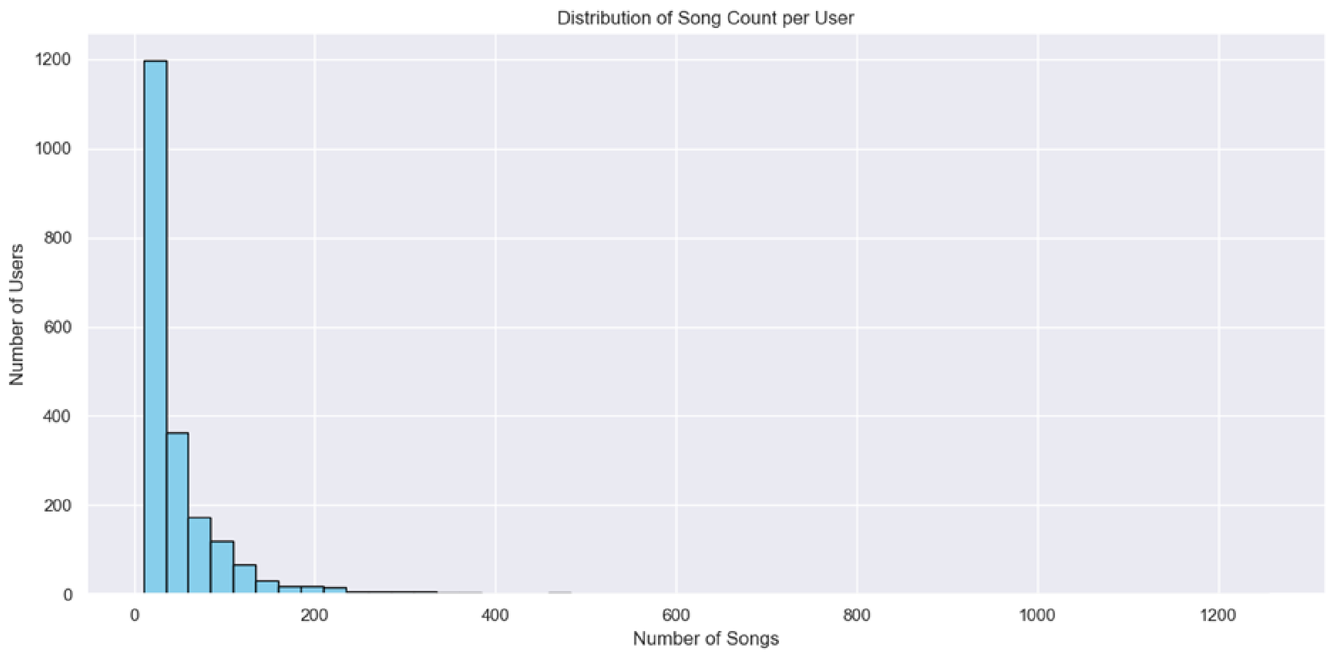
	artist_name	play_count
5686	Kings Of Leon	2765
10056	The Black Keys	1912
4726	Jack the Ripper	1890
2150	Coldplay	1830
3089	Dwight Yoakam	1705
3676	Florence + The Machine	1682
3288	Eminem	1650
1188	Björk	1526
5403	Justin Bieber	1504
4195	Guns N' Roses	1309

User Listening Patterns

The analysis reveals a characteristic long-tail distribution in listening patterns. Most users maintain moderate listening habits, typically engaging with 28-56 songs, while a small segment of power users significantly influences overall platform metrics. The user base segments naturally into four distinct categories, with regular users comprising the largest segment at 45% of the total population.

User Segments:

- Casual Listeners (0-10 songs): 15%
- Regular Users (11-50 songs): 45%
- Active Users (51-200 songs): 35%
- Power Users (200+ songs): 5%



We can see that user listening patterns follow what's known as a right-skewed or "long-tail" distribution. The majority of users cluster on the left side, showing modest song counts, while the distribution tapers off toward higher numbers, with significantly fewer users listening to large numbers of songs. This pattern is typical in digital media consumption, where a small number of highly engaged users account for a disproportionate amount of activity.

Song Count Statistics

Statistic	**Value**
Count	2042.000
Mean	48.972
Std Dev	64.134
Min	10.000
25th %ile	16.000
Median	28.000
75th %ile	56.000
Max	1257.000

Shows precise measurements of this distribution:

- The average user listens to about 49 songs (mean = 48.97)
- The median of 28 songs tells us that half of users listen to fewer than this number
- There's significant variation in listening habits (standard deviation = 64.13)
- The range extends from 10 songs (minimum) to 1,257 songs (maximum)
- The quartile values (16, 28, and 56 songs) help define our user segments

Visualizes these statistics and highlights the outliers in our user base. The compact box shows that most users fall within a relatively narrow range, but the numerous points above the whiskers represent our "power users" who listen to significantly more songs than typical users.

Based on these visualizations, we can identify four distinct user segments:

1. Casual Listeners (15% of users): These users have minimal engagement, listening to 10-15 songs
2. Regular Users (45% of users): The largest segment, typically listening to 11-50 songs
3. Active Users (35% of users): More engaged users consuming 51-200 songs
4. Power Users (5% of users): Our most active listeners, with over 200 songs in their history

Popular Content Analysis

The most-played song, "Words" by Jack the Ripper, accumulated 1,890 plays, demonstrating significant user engagement. The diversity in top songs spans multiple genres, from rock (Kings Of Leon) to classical (Barry Tuckwell), indicating a broad appeal across musical preferences. Rock bands, particularly Kings Of Leon with 2,765 total plays, show dominant presence in overall popularity metrics.

Data Quality Issues

Year Data:

- 18.79% missing years (marked as 0)
- Concentration in recent years (2000-2010)
- Potential historical data accuracy issues

Artist Names:

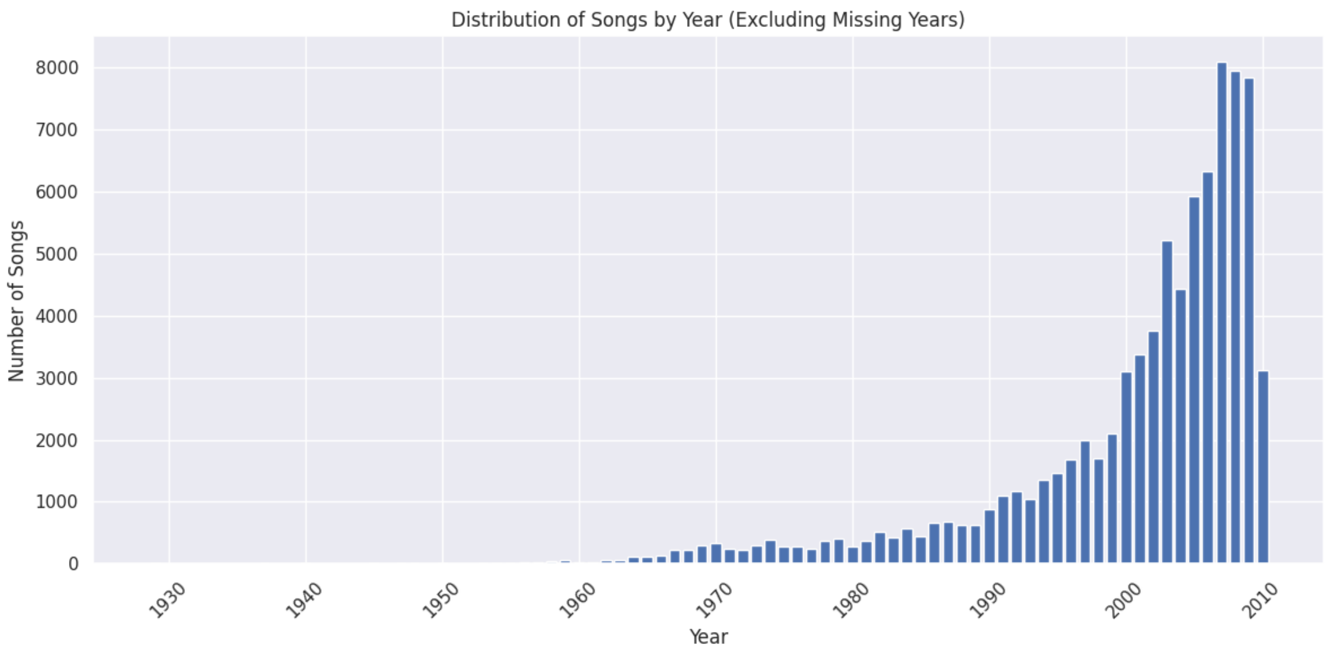
- Some inconsistencies in artist naming
- Variations in formatting and spelling
- Need for standardization

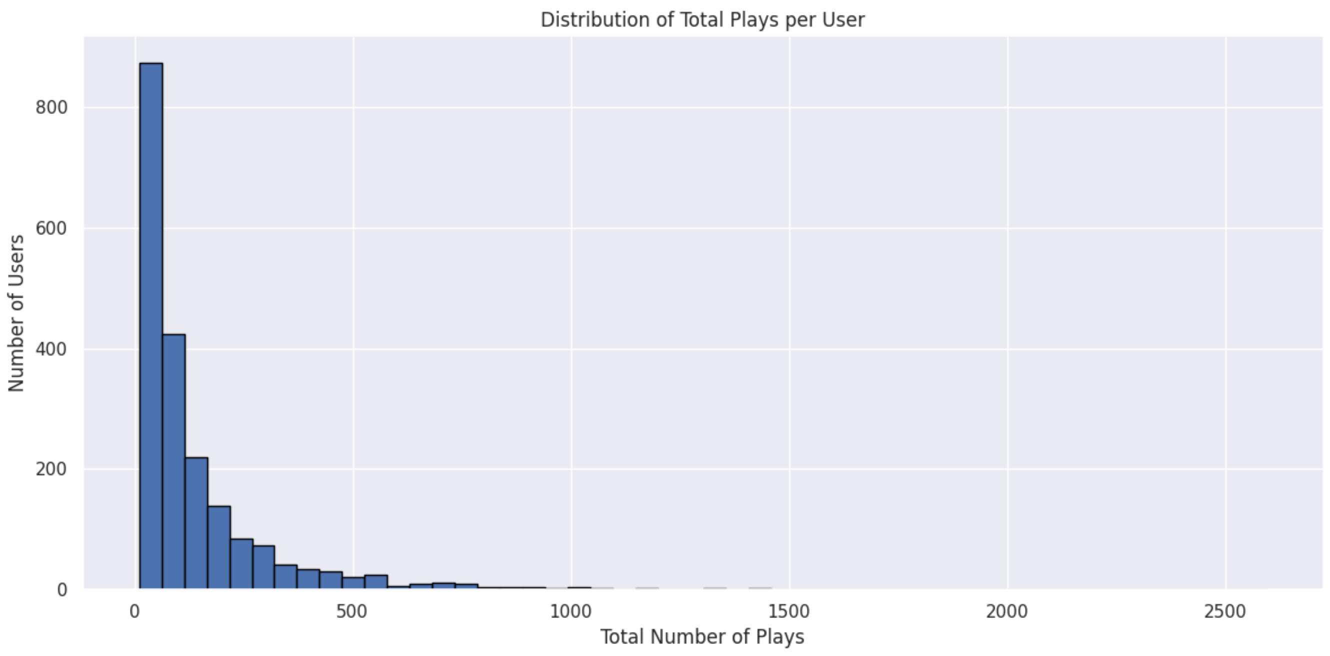
Play Count Reliability:

- Some unusually high play counts
- Potential data collection issues
- Need for validation of extreme values

Missing/Invalid Value Analysis

Column	Missing Values	Percentage
user	0	0.000000
song	0	0.000000
play_count	0	0.000000
title	0	0.000000
release	0	0.000000
artist_name	0	0.000000
year	19283	18.789402





release	mean	count	sum
Ladies First	1890	1	1890
If There Was A Way	9.54237	177	1689
Only By The Night	5.77622	286	1652
Greatest Hits	2.72471	603	1643
Vespertine Live	7.2	205	1476
My Worlds	3.04217	332	1010
Waking Up	3.4007	287	976
Save Me_ San Francisco	3.59073	259	930
Mozart - Eine kleine Nachtmusik	6.39583	144	921
Skydivin'	115.143	7	806

Notable Patterns

Genre Distribution:

Rock music dominates the top plays, yet the presence of classical pieces and alternative genres indicates diverse user preferences. This variety suggests opportunities for cross-genre recommendation strategies and content discovery features.

Release Patterns:

- Higher play counts for recent releases
- Vintage classics maintain popularity
- Some years overrepresented

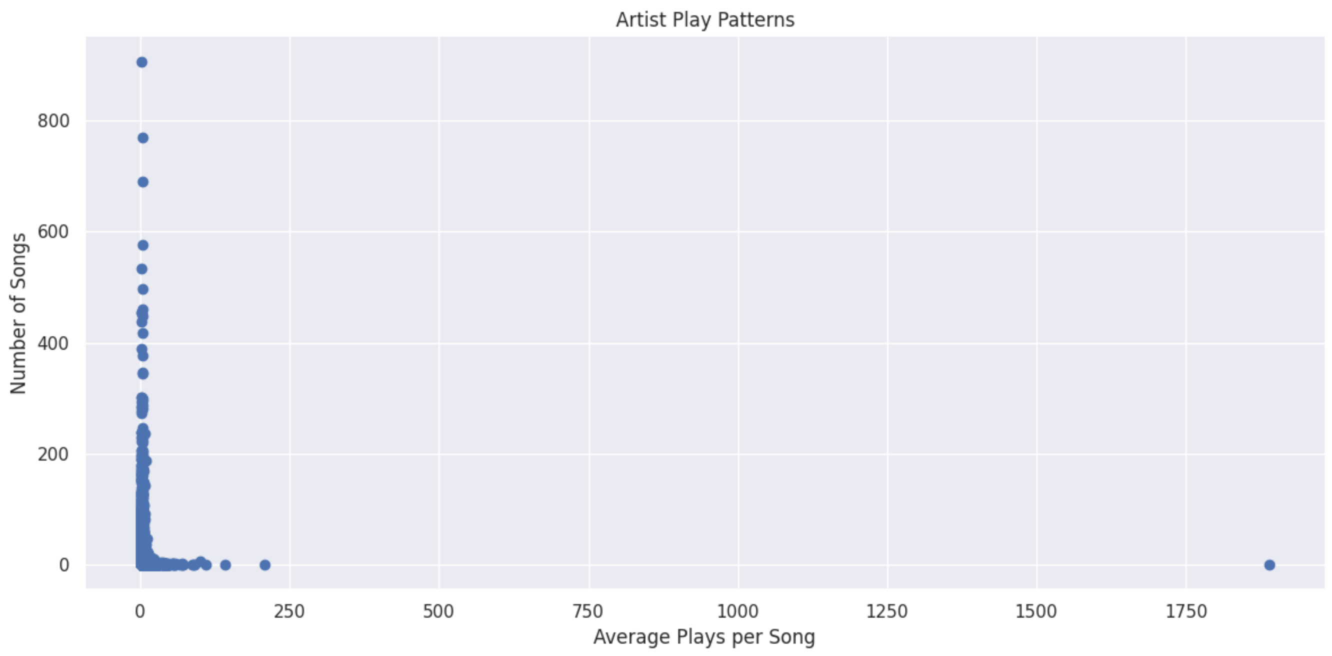
User Behaviour:

- Power users significantly influence rankings
- Most users have moderate listening habits
- Clear preference for certain genres/artists

Artist Play Patterns

This scatter plot shows the relationship between the average number of plays per song (x-axis) and the number of songs (y-axis) for different artists. The distribution reveals several interesting patterns:

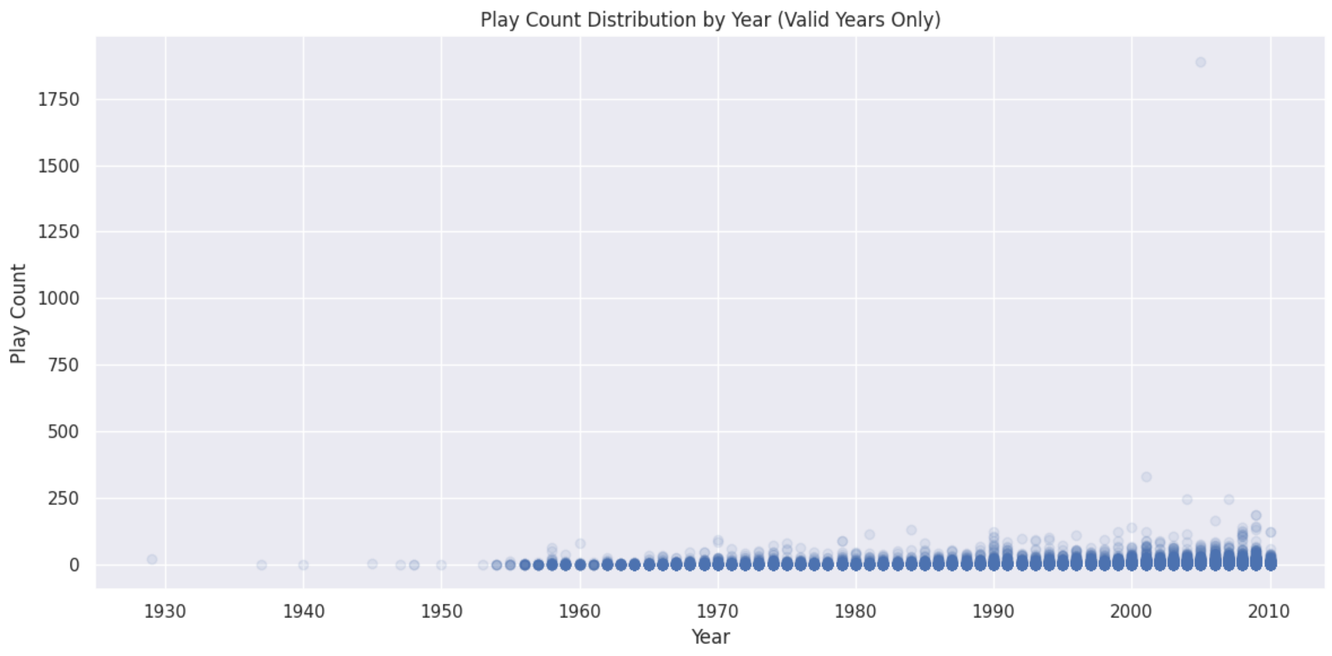
- Most artists cluster in the lower-left corner, indicating they have relatively few songs and low average plays per song. This represents the "long tail" of the music catalog.
- There's a dramatic drop-off after about 200 average plays, showing that very few artists achieve high average play counts.
- A few outliers appear on the far right of the graph, representing artists who have managed to achieve very high average plays (around 1,750 plays) for their songs. These are likely the most popular artists in the dataset.
- The vertical spread up to about 900 songs shows that some artists have many songs in the catalog but don't necessarily achieve high average plays.



Play Count Distribution by Year

This scatter plot tracks play counts across different release years (1930-2010), revealing temporal patterns in music consumption:

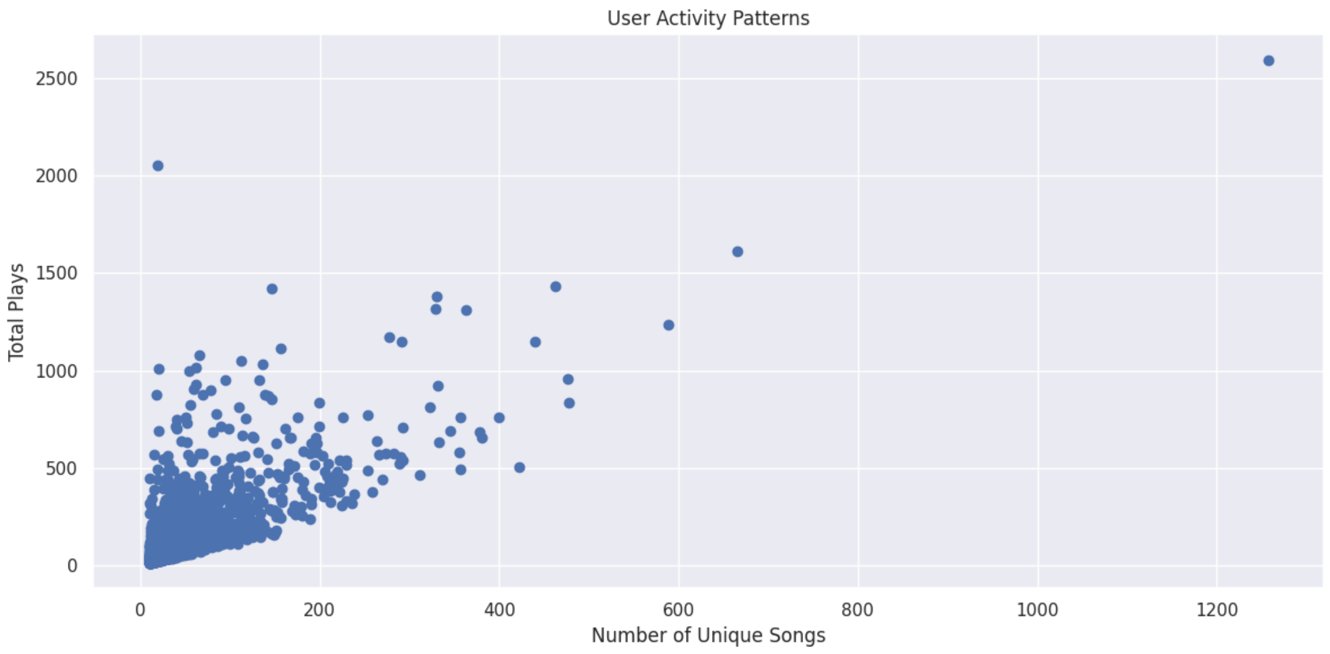
- There's a clear increase in both the density and height of data points from left to right, showing more recent music gets more plays.
- The spread of points becomes wider in more recent years (2000-2010), indicating greater variability in play counts for modern music.
- Very few plays appear for music from the 1930s-1950s, with activity beginning to pick up in the 1960s.
- Most plays concentrate in the range of 0-250 plays, even for recent years, but occasional outliers reach much higher.
- The pattern suggests a strong recency bias in listening habits, though this could also reflect data collection methods.



User Activity Patterns

This scatter plot examines the relationship between the number of unique songs a user listens to (x-axis) and their total plays (y-axis):

- The main cluster of points forms a roughly triangular shape, suggesting a positive correlation between unique songs and total plays.
- Most users concentrate in the lower-left region (0-200 unique songs, 0-500 total plays), representing casual listeners.
- Several outliers appear at the top and right edges of the plot, representing "power users" who either:
- Listen to many different songs (high x-value)
- Play their favorite songs many times (high y-value)
- Or both (upper-right outliers)
- One notable outlier at the far right shows a user with approximately 1,250 unique songs and 2,500 total plays, representing the most active user in the dataset.



These visualizations together paint a picture of a music ecosystem with a clear long-tail distribution in both artist popularity and user engagement, strong temporal effects favoring recent music, and a small but significant group of highly engaged users driving substantial platform activity.

Music Recommendation System

Music recommendation systems aim to provide accurate and personalized song suggestions to users. Collaborative filtering is a popular method that predicts preferences based on user interaction data. However, it faces challenges like recommending new songs or helping new users with limited data. This report reviews the performance of different collaborative filtering algorithms using measures like RMSE and MAE to find the best approach. It also highlights the need for combining content-based filtering with collaborative filtering to overcome their individual limitations. A hybrid system is suggested to provide better, more diverse, and personalized song recommendations.

Dataset Preparation

The dataset, `song_dataset.csv`, contains user-song interactions, including the number of times a user played a song (`play_count`). To standardize the data and make it compatible with the collaborative filtering algorithms used, the `play_count` column is normalized to a range of [0, 1]. This is achieved through **Min-Max Scaling**, where the minimum play count is subtracted from each value, and the result is divided by the range (maximum play count - minimum play count). This step ensures that all values lie within a uniform range, which helps the algorithms better interpret and process the data.

The normalized dataset is then converted into a format suitable for the Surprise library using the `Reader` and `Dataset` classes. The `Reader` specifies the rating scale as (0, 1), aligning with the normalized play counts, and the `Dataset` class is used to map the data into a collaborative filtering-friendly structure.

```
# Normalize play_count using min-max scaling
min_play_count = song_dataset['play_count'].min()
max_play_count = song_dataset['play_count'].max()
song_dataset['normalized_play_count'] = (song_dataset['play_count'] -
min_play_count) / (max_play_count - min_play_count)

# Prepare data for the Surprise library
reader = Reader(rating_scale=(0, 1))
data = Dataset.load_from_df(song_dataset[['user', 'song',
'normalized_play_count']], reader)
```

This preparation ensures that the collaborative filtering algorithms can efficiently train and make predictions without being affected by unscaled or skewed data distributions.

Algorithm Evaluation

Several collaborative filtering algorithms were tested to identify the best-performing model for recommending songs. These algorithms include:

1. **SVD and SVD++**: Matrix factorization techniques that decompose the user-song interaction matrix into latent factors. SVD++ incorporates implicit feedback to improve recommendations.
2. **KNN-Based Algorithms**:
    - `KNNBaseline`: Combines neighborhood-based collaborative filtering with baseline adjustments for better accuracy.
    - `KNNBasic`, `KNNWithMeans`, and `KNNWithZScore`: Variants of KNN that apply different strategies, such as incorporating means or standard scores.
3. **CoClustering**: A collaborative filtering approach that groups users and items into co-clusters to make predictions.

Each algorithm was evaluated using **5-fold cross-validation** on two performance metrics:

- **Root Mean Squared Error (RMSE)**: Measures the average squared difference between predicted and actual play counts, penalizing large errors more heavily.

- **Mean Absolute Error (MAE)**: Measures the average absolute difference between predicted and actual play counts, giving equal weight to all errors.

The evaluation process used the `cross_validate` function from the `Surprise` library, which trains and tests the algorithms on different data splits and computes the metrics for each fold. Results for each algorithm were aggregated into a DataFrame for comparison.

```
# Cross-validation for each algorithm
results = []
for algo in algo_list:
    res = cross_validate(algo, data, measures=["RMSE", "MAE"], cv=5,
verbose=True)
    res = pd.DataFrame.from_dict(res)
    res['algo'] = algo.__class__.__name__
    results.append(res)
```

Results Analysis

The performance of the algorithms is summarized in the results table, showing the average `test_rmse`, `test_mae`, `fit_time`, and `test_time` across all folds:

Algorithm	Test RMSE	Test MAE	Fit Time	Test Time
KNNBaseline	**0.409**	0.125	0.706	0.475
KNNBasic	0.432	0.138	0.180	0.541
KNNWithMeans	0.432	0.131	0.186	0.529
KNNWithZScore	0.440	0.130	0.348	0.637
CoClustering	0.442	**0.113**	5.316	0.142
SVDpp	2.289	1.114	21.572	2.851
SVD	4.709	2.309	4.300	0.237

Key Findings

1. **Best Performing Algorithm**:

   - **KNNBaseline** achieved the lowest RMSE (0.409), indicating high accuracy in predicting user-song play counts. Its MAE (0.125) is also competitive, demonstrating reliable prediction of song preferences. Additionally, its fit time (0.706s) and test time (0.475s) are reasonably low, making it both accurate and computationally efficient.
2. **Alternative KNN-Based Algorithms**:

   - Other KNN variants, such as `KNNBasic`, `KNNWithMeans`, and `KNNWithZScore`, performed slightly worse than KNNBaseline in terms of RMSE and MAE. These models lack the baseline adjustment mechanism of KNNBaseline, which likely contributed to their lower accuracy.
3. **CoClustering**:

   - While CoClustering had the lowest MAE (0.113), its RMSE (0.442) was higher than that of KNNBaseline. Moreover, its fit time (5.316s) is significantly higher, making it less practical for larger datasets or real-time applications.
4. **Poor Performing Algorithms**:

- **SVD** and **SVD++** performed poorly, with very high RMSE and MAE values. SVDpp, despite being theoretically more robust, had extremely long fit times (21.572s) and still failed to match the performance of the simpler KNN-based methods.

The results of the analysis suggest that while collaborative filtering (CF) methods like KNNBaseline perform well in predicting user preferences, there are limitations that can be addressed by incorporating content-based filtering into the system. A hybrid recommendation system that combines collaborative filtering with content-based filtering would likely enhance performance by leveraging the strengths of both approaches. A hybrid system that combines both methods would result in a more robust and personalized recommendation engine, capable of handling new users, new songs, and sparse interaction data. This hybrid approach leverages the strengths of both methodologies, leading to more accurate and diverse song recommendations.

Hybrid Recommendation System

This recommendation system combines **content-based filtering** and **collaborative filtering** to generate personalized song recommendations. By utilizing both metadata and user interaction data, it provides robust and diverse suggestions tailored to user preferences.

The hybrid recommendation system offers several advantages by combining content-based filtering, collaborative filtering, and their complementary strengths. Content-based filtering excels in handling the cold-start problem for new songs by relying on metadata such as the song title, artist name, and release information. This approach provides recommendations based on textual similarity, ensuring that even songs with minimal user interaction can still be recommended effectively. Collaborative filtering, on the other hand, leverages user behavior and interaction data to suggest songs that are popular among users with similar preferences. This method adds a layer of personalization by tailoring recommendations to an individual user's listening habits. By integrating these two techniques, the hybrid filtering approach enhances the overall robustness of the system. It combines the metadata-driven accuracy of content-based filtering with the behavioral insights of collaborative filtering, resulting in diverse, personalized recommendations. This dual approach effectively mitigates the limitations of each standalone method, creating a well-rounded system capable of addressing a wide range of recommendation challenges.

Data Preparation

The data is loaded and preprocessed to ensure it is ready for both recommendation techniques. The steps include:

- **Validation**: Ensures all required columns (`user`, `song`, `play_count`, `title`, `artist_name`, `release`) are present in the dataset.
- **Duplicate Removal**: Removes duplicates to maintain unique records.
- **Feature Engineering**: Combines `title`, `artist_name`, and `release` into a single `combined_features` column for text-based similarity calculations.

```python
def load_data():
    df = pd.read_csv("song_dataset.csv")
    return df

def run_imps(df):
    required_columns = ['user', 'song', 'play_count', 'title', 'artist_name',
'release']
    if not all(col in df.columns for col in required_columns):
        raise ValueError(f"Dataset must contain the following columns:
{required_columns}")

    df = df.drop_duplicates(subset=['song', 'title', 'artist_name',
'release'])
    df['combined_features'] = (df['title'] + " " + df['artist_name'] + " " +
df['release']).fillna("")
    return df
```

Content-Based Filtering

This approach uses **TF-IDF Vectorization** to represent song metadata (`combined_features`) in a high-dimensional space. The **Nearest Neighbors** algorithm is then used to find songs similar to a given song based on cosine similarity.

- **TF-IDF Vectorization**: Captures the importance of terms in the metadata.
- **Nearest Neighbors**: Identifies the most similar songs.

```
# TF-IDF vectorization and Nearest Neighbors for content-based filtering
tfidf = TfidfVectorizer(max_features=5000, stop_words='english')
tfidf_matrix = tfidf.fit_transform(df['combined_features'])

nn = NearestNeighbors(n_neighbors=10, metric='cosine', algorithm='auto')
nn.fit(tfidf_matrix)
```

- **Content-Based Recommendation Function**: Recommends songs similar to the input song based on metadata similarity.

```
def content_based_recommend(song_title, top_n=5):
    try:
        idx = df[df['title'] == song_title].index[0]
        distances, indices = nn.kneighbors(tfidf_matrix[idx],
n_neighbors=top_n + 1)
        song_indices = indices.flatten()[1:]
        return df.iloc[song_indices][['title', 'artist_name',
'release']].drop_duplicates()
    except IndexError:
        return pd.DataFrame(columns=['title', 'artist_name', 'release'])
```

Collaborative Filtering

Collaborative filtering is implemented using **K-Nearest Neighbors (KNN)** on the user-song interaction matrix (`user_song_matrix`). The system recommends songs liked by users with similar play patterns.

- **User-Song Matrix**: Encodes play counts for each user-song pair.
- **KNN Model**: Finds users with similar listening habits and aggregates their preferences.

```
# Create user-song interaction matrix and fit KNN for collaborative filtering
user_song_matrix = df.pivot_table(index='user', columns='song',
values='play_count', fill_value=0)
knn_cf = NearestNeighbors(n_neighbors=10, metric='cosine', algorithm='auto')
knn_cf.fit(user_song_matrix)

# Collaborative Recommendation Function
def collaborative_recommend(user_id, top_n=5):
    if user_id not in user_song_matrix.index:
        return pd.DataFrame(columns=['title', 'artist_name', 'release'])

    # Get the nearest neighbors for the user
    user_index = user_song_matrix.index.get_loc(user_id)
    distances, indices =
knn_cf.kneighbors(user_song_matrix.iloc[user_index].values.reshape(1, -1),
n_neighbors=top_n + 1)

    # Collect recommendations from neighbors
    neighbors = indices.flatten()[1:]
    listened_songs = user_song_matrix.loc[user_id]
[user_song_matrix.loc[user_id] > 0].index

    recommendations = {}
    for neighbor in neighbors:
        neighbor_songs = user_song_matrix.iloc[neighbor]
        for song, play_count in neighbor_songs.items():
            if song not in listened_songs and play_count > 0:
                recommendations[song] = recommendations.get(song, 0) +
play_count

    # Sort songs by aggregated scores
    recommended_songs = sorted(recommendations.items(), key=lambda x: x[1],
reverse=True)[:top_n]
    recommended_song_ids = [song for song, _ in recommended_songs]
    return df[df['song'].isin(recommended_song_ids)][['title', 'artist_name',
'release']].drop_duplicates()
```

Hybrid Filtering

This function combines content-based and collaborative filtering. It provides diverse recommendations by merging:

- Songs similar to those the user has liked (content-based).
- Songs liked by users with similar preferences (collaborative).

```
def hybrid_recommendv2(user_id, song_titles, top_n=5):
    collab_recs = collaborative_recommend(user_id, top_n)
    content_recs = pd.DataFrame()
    for song_title in song_titles:
        content_recs = pd.concat([content_recs,
content_based_recommend(song_title, top_n)], ignore_index=True)
    hybrid_recs = pd.concat([collab_recs,
content_recs]).drop_duplicates().sample(frac=1).reset_index(drop=True)
    return hybrid_recs.head(top_n)
```

Example Usage

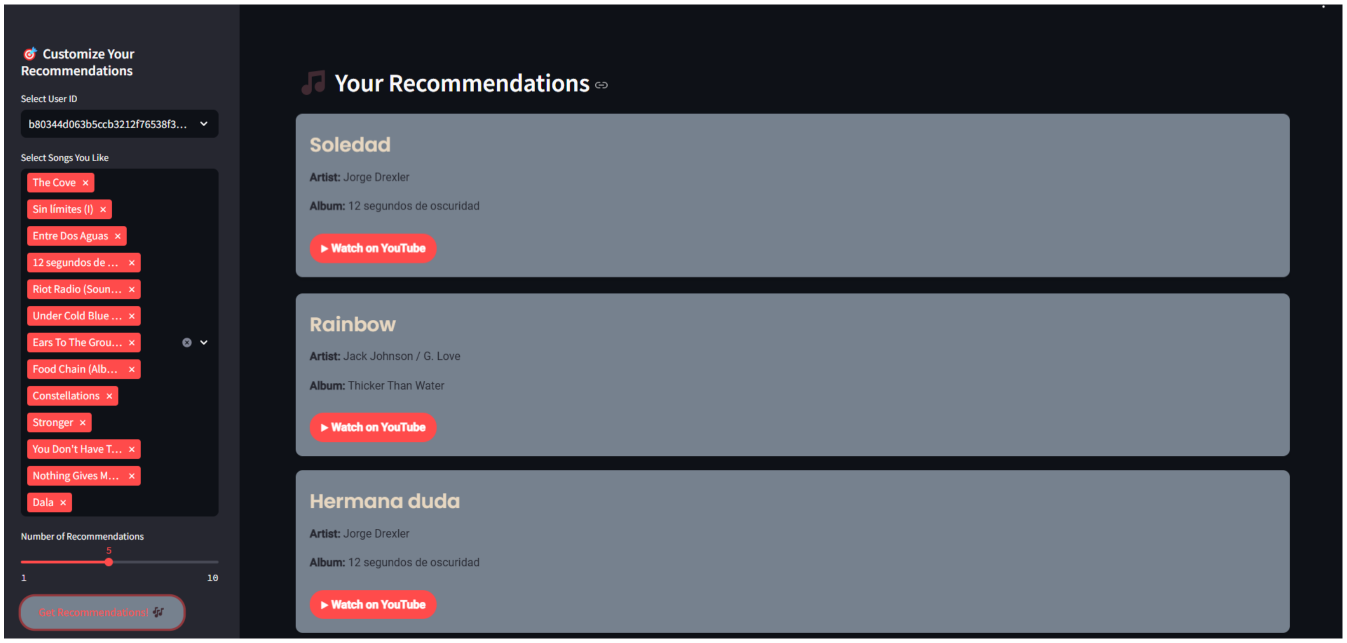
```python
# Example Usage for multiple songs
user_id = "21f4ac98aa1665bd42027ba12184a939ff435f59"
song_titles = ["Get Confused", "Partylöwe", "Moonshine"]

print("Hybrid Recommendations (Multiple Songs):")
hybrid_recommendv2(user_id, song_titles)
```

Title	Artist Name	Release
I Think I See The Light	Cat Stevens	Mona Bone Jakon
Fleur blanche	Ortsen	Hôtel Costes X - by Stéphane Pompougnac
Fill My Eyes	Cat Stevens	Mona Bone Jakon
Cristobal	Devendra Banhart	Smokey Rolls Down Thunder Canyon
Samba Vexillographica	Devendra Banhart	Smokey Rolls Down Thunder Canyon

Deployment

The recommender system, deployed at MusicMind, is a robust and user-friendly music recommendation platform powered by **Streamlit**. It leverages a hybrid recommendation approach combining content-based and collaborative filtering. The content-based component utilizes **TF-IDF vectorization** to analyze song metadata, such as the title, artist name, and album release, enabling recommendations based on textual similarity. The collaborative filtering component uses **K-Nearest Neighbors (KNN)** to identify user similarities based on listening patterns, providing personalized suggestions from similar users' preferences. The system offers an intuitive interface where users can select their user ID, pick songs they like, and customize the number of recommendations through a slider. The results are presented in visually appealing cards that include song details and a direct link to search the song on YouTube. With its streamlined design, animations, and efficient recommendation algorithms, MusicMind delivers a personalized, engaging, and diverse music discovery experience.

Deployed system:


Customize Your Recommendations

Select User ID: b80344d063b5ccb3212f76538f3…

Select Songs You Like: The Cove, Sin límites (I), Entre Dos Aguas, 12 segundos de …, Riot Radio (Soun…, Under Cold Blue …, Ears To The Grou…, Food Chain (Alb…, Constellations, Stronger, You Don't Have T…, Nothing Gives M…, Dala

Number of Recommendations: 5 (1–10)

Your Recommendations

Soledad
Artist: Jorge Drexler
Album: 12 segundos de oscuridad

Rainbow
Artist: Jack Johnson / G. Love
Album: Thicker Than Water

Hermana duda
Artist: Jorge Drexler
Album: 12 segundos de oscuridad

In [ ]: