Music Recommendation System Potential Solutions

A modified MIT Applied Data Science project using the million-song dataset by Lin Zhan

(Notebook links: https://colab.research.google.com/drive/1Fjk56d4y70Pg93mh01jvPtzDzRuD2ZLW#scrollTo=O7lzqhfahAcf; https://colab.research.google.com/drive/1qQSxEqQ1amlunk1w194viXvS7O0B6zux)

Contents

Overview

The business objective is to increase subscription and advertising revenue. To achieve this, the product goal is to increase the number of daily active users by improving user engagement and satisfaction on the platform.

Problem:

User engagement and satisfaction are low on the music streaming platform, which leads to subscription losses.

People spend 1.5 ~ 2.5 hours per day listening to appx. 22 ~ 50 songs.

Currently, users are manually searching and browsing songs for their playlists. A small portion of songs were discovered and listened to frequently by a large number of users, but a large portion of songs were only discovered by a few users

This experience is not optimal. Users struggle to discover new songs from a catalog of millions of songs to their preference.

Solution:

A recommendation system to enhance user experience with personalization and content discovery and use different models based on segmentation of user behaviors:

- Use a rule-based ranking model for cold-start users
- Use the **Hybrid Clustering model** for predicted play counts and the **DistilBERT model** for a content-based recommendation for existing users
- Consider using **Neural Network with NLP** in the next phase

Raw Data Analysis

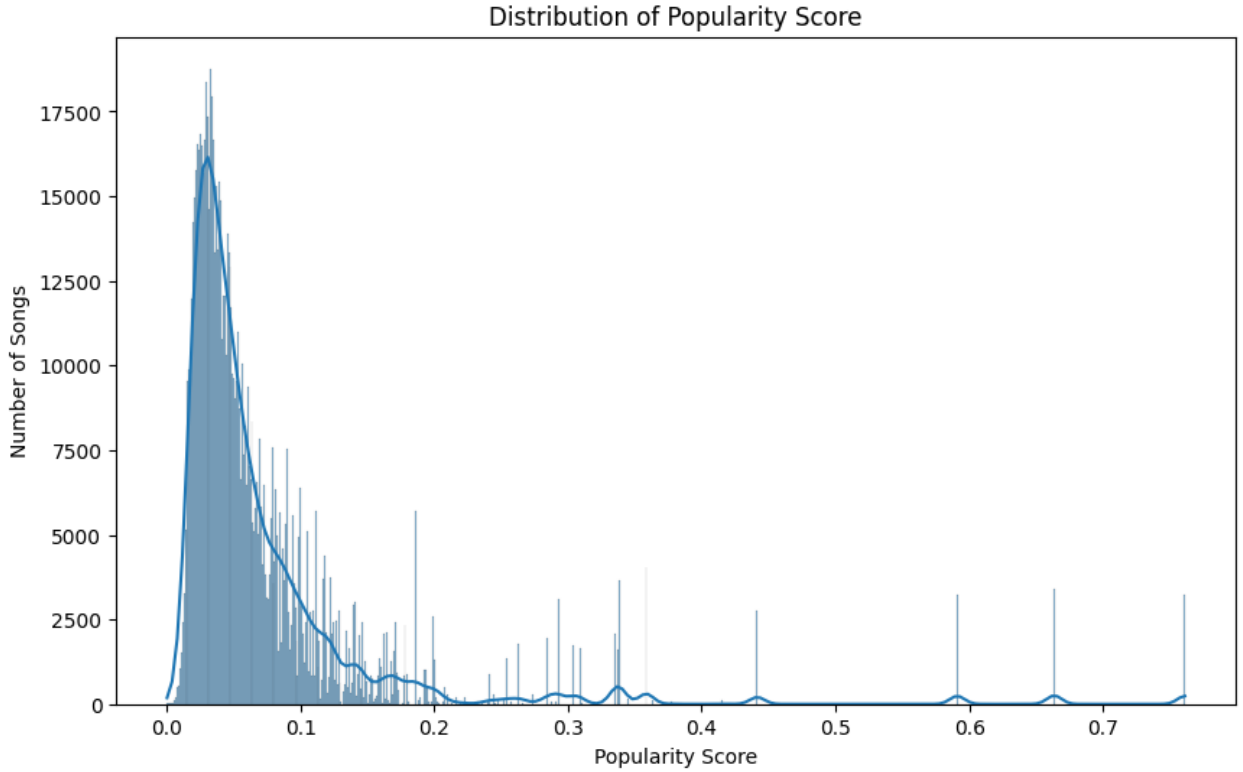
Data Overview

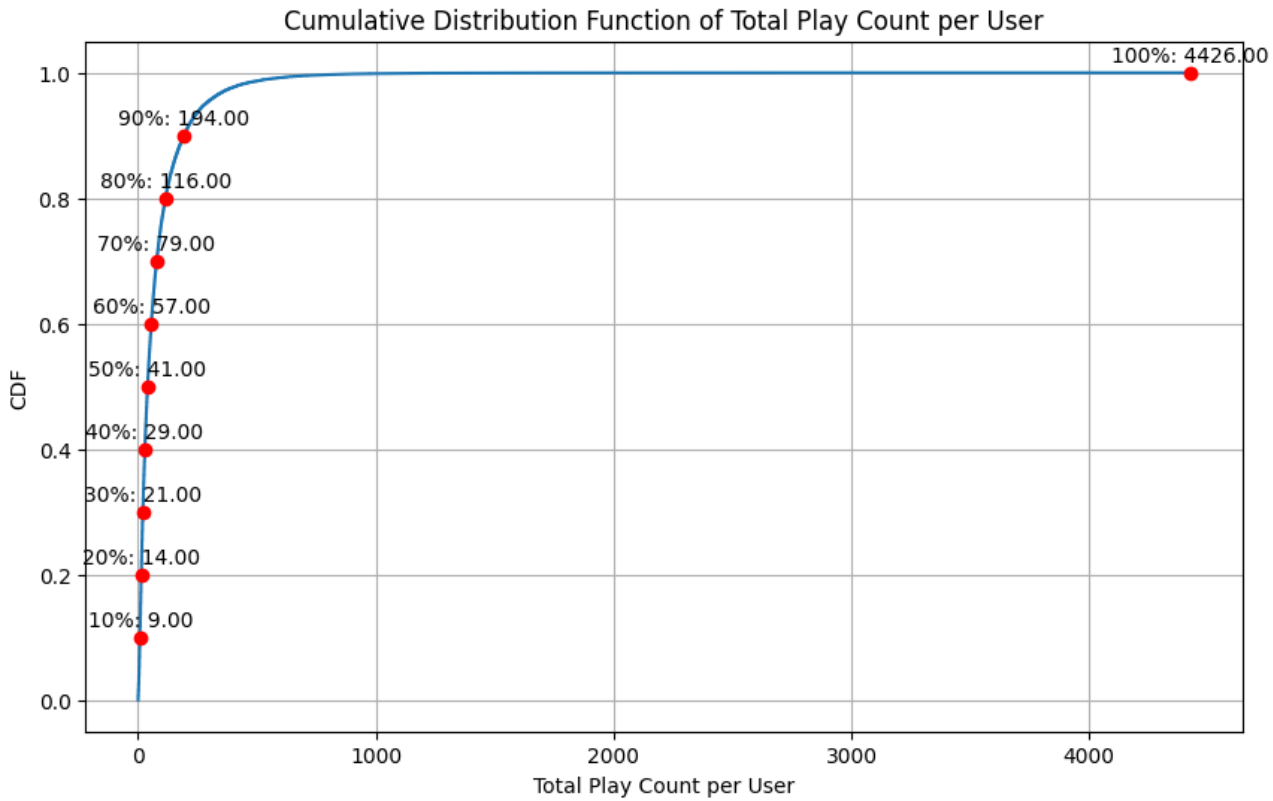
There are two main datasets: User Song Play Counts (2M rows) and Song Details (1M rows).

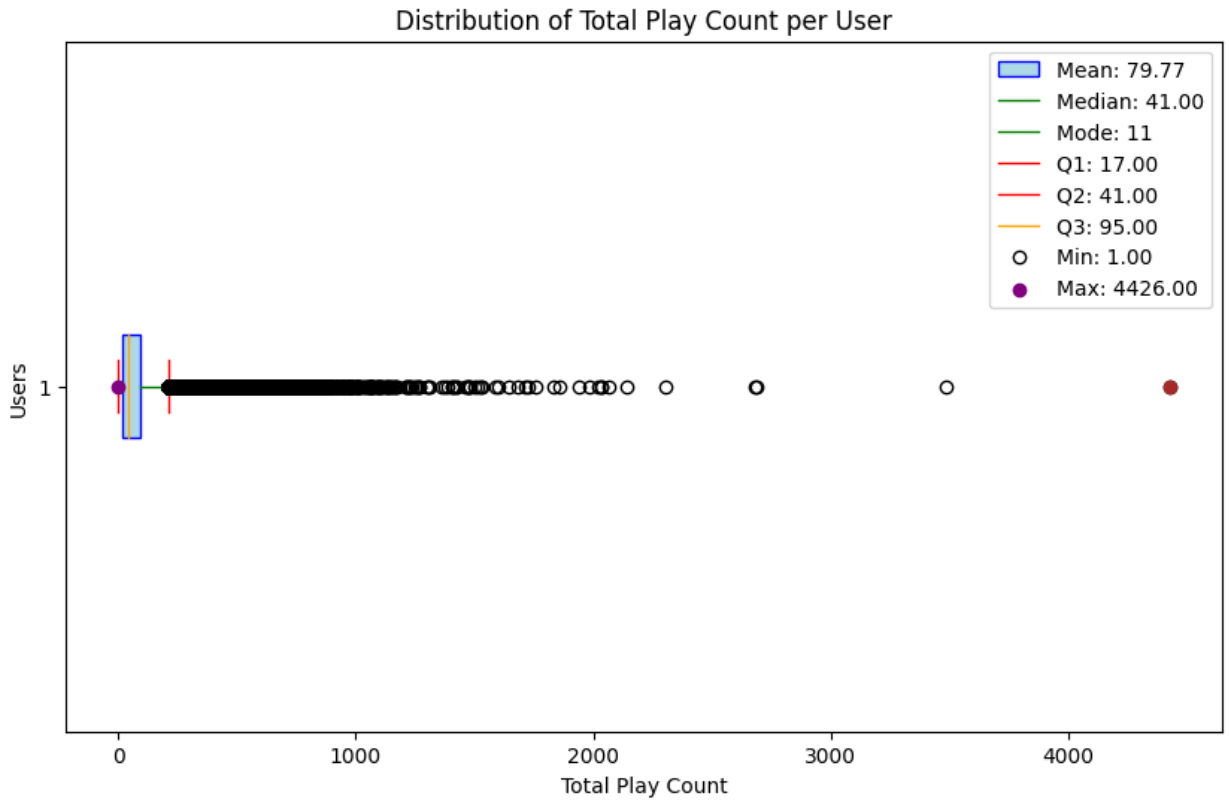
After joining Song Details to User Song Play Counts and feature engineering, the dataset includes user ID, song ID, play counts, and additional song metadata. The dataset contains 76,353 unique users and 10,000 unique songs.

Next, I will analyze user-song interactions by looking at user preferences, song play frequencies, song discovery, and song popularity.

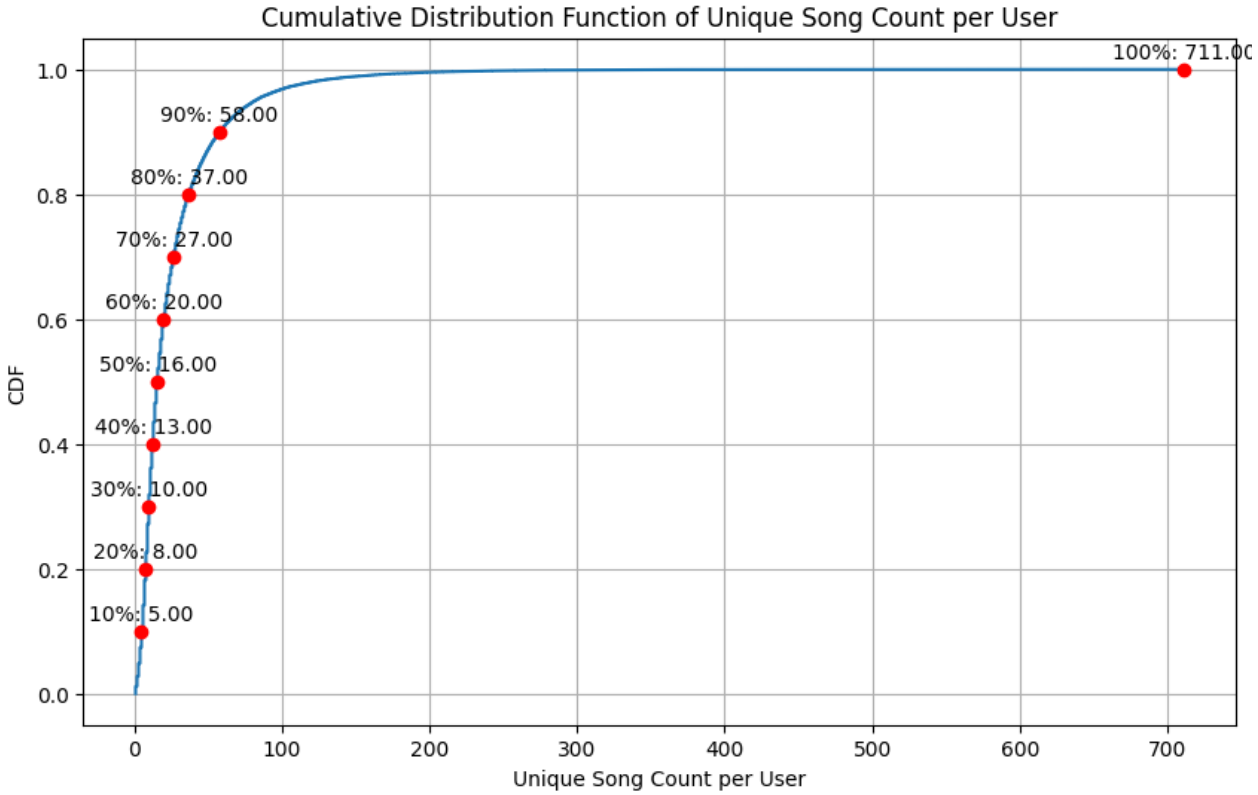
The data is highly skewed. I will apply filtering criteria based on the findings to finalize a data population to experiment with different models for the recommendation system.

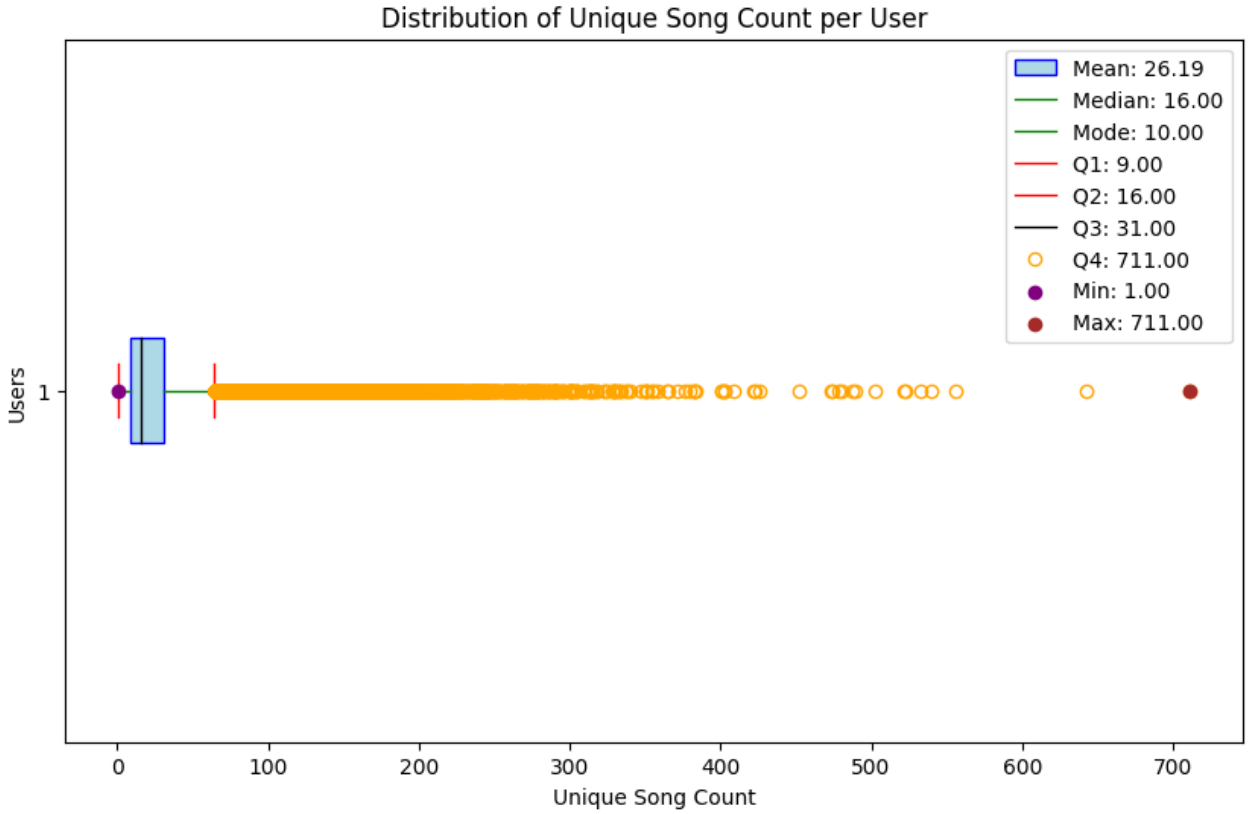






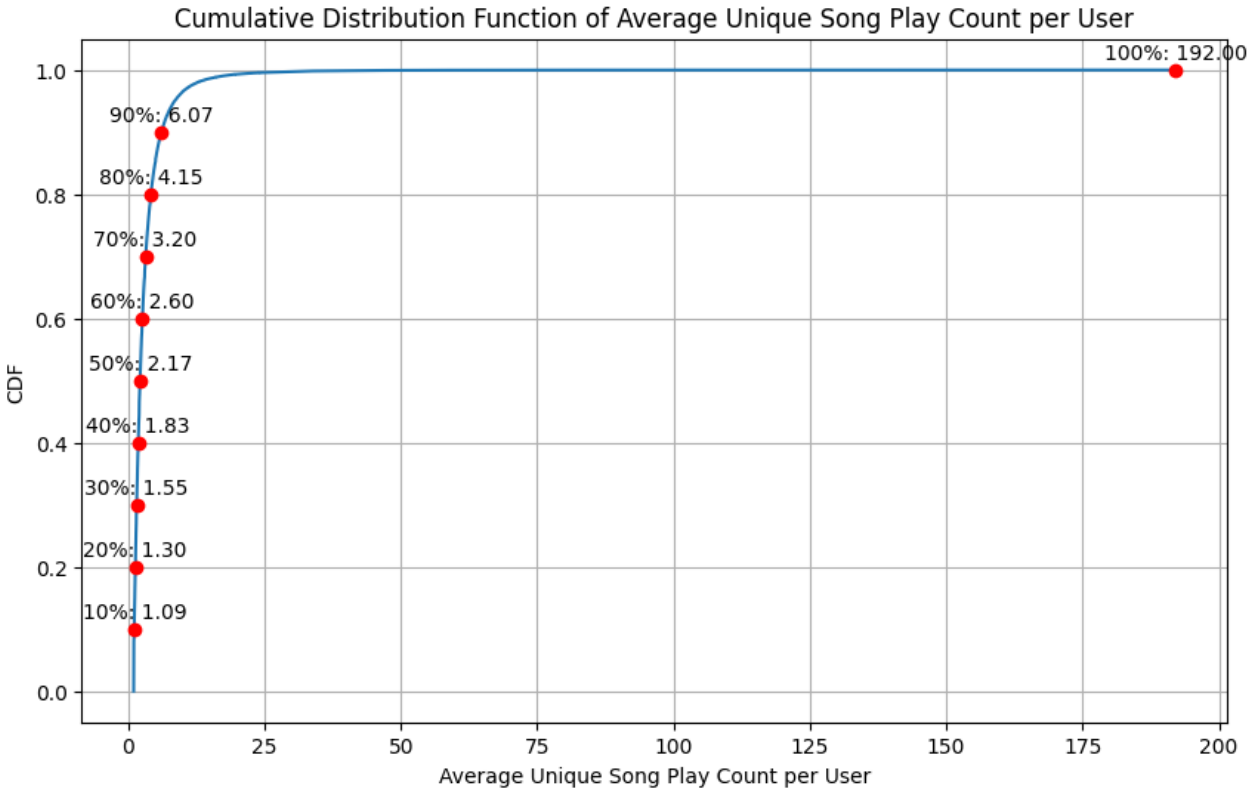
1) User Song Play Frequency (Total Play Count per User):
The dataset is filtered for a list of users who played at least 80 times.

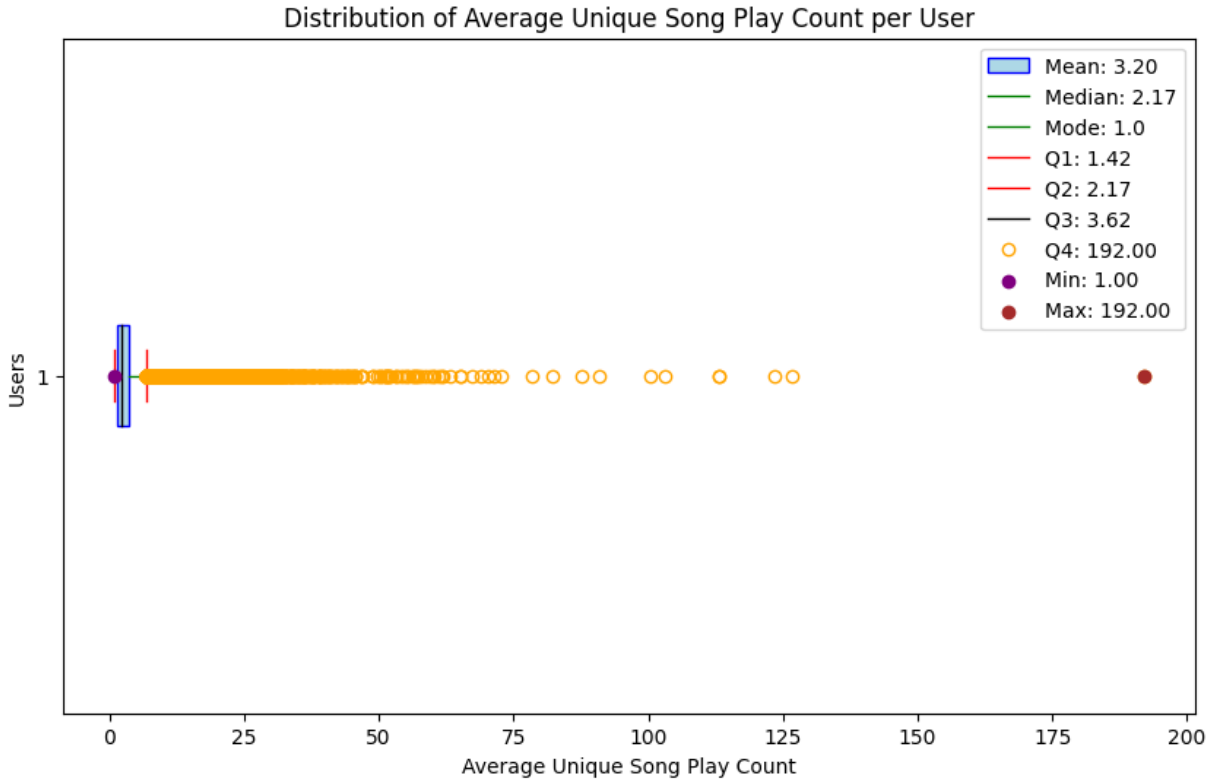




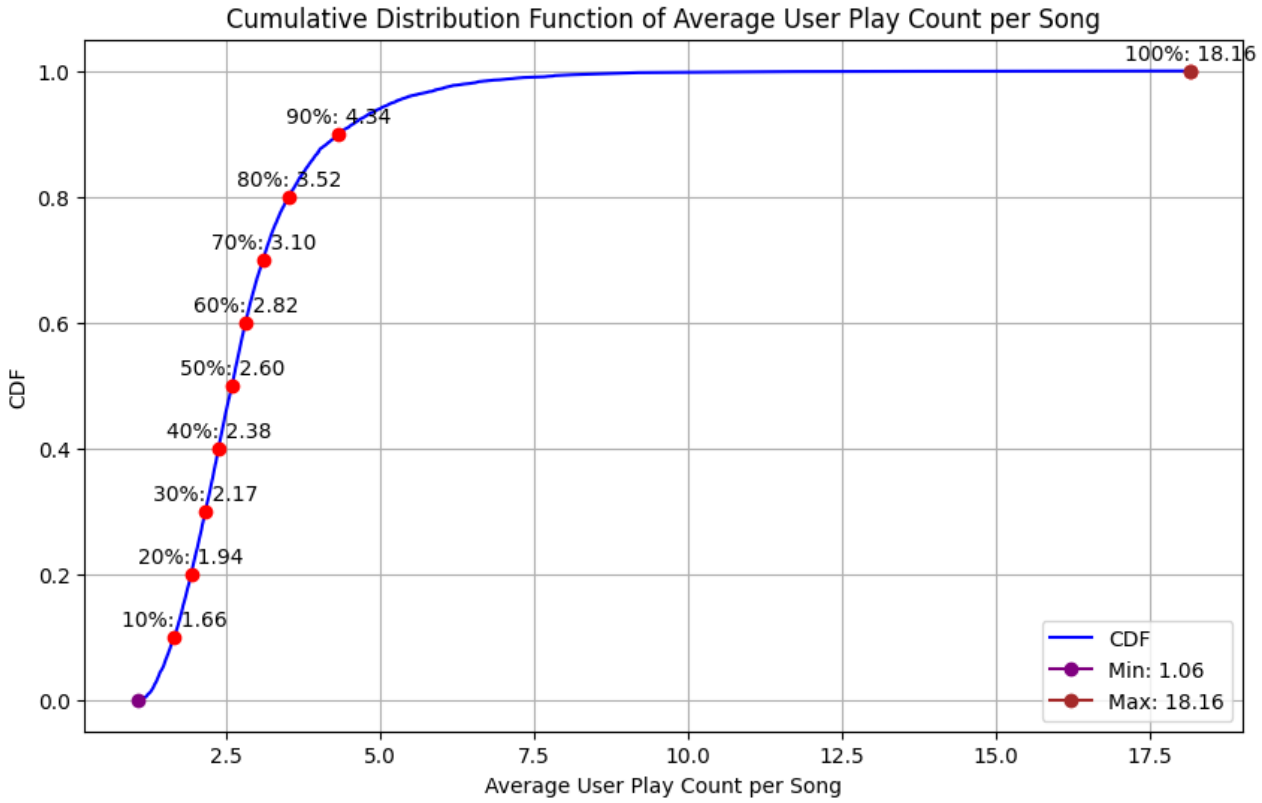
2) User Song Diversity (Unique Song Count per User):

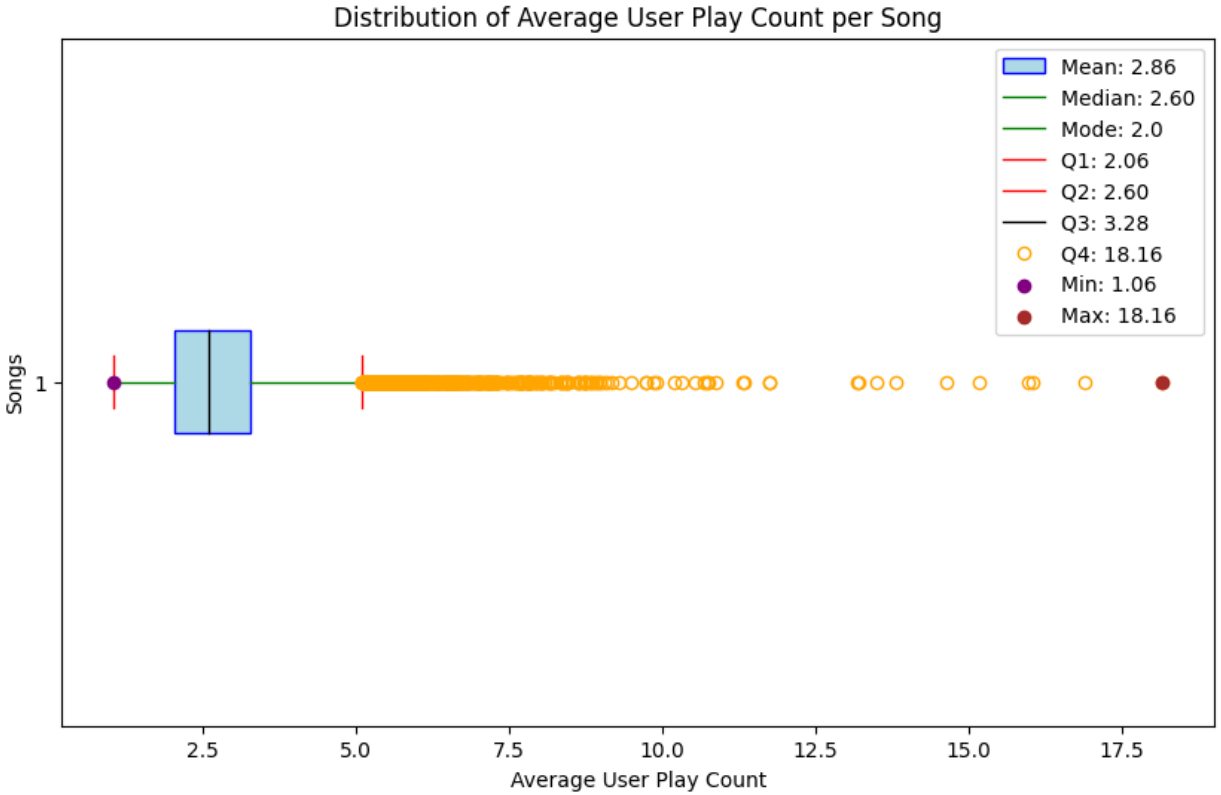
The dataset is filtered for a list of users who played at least 3 unique songs.





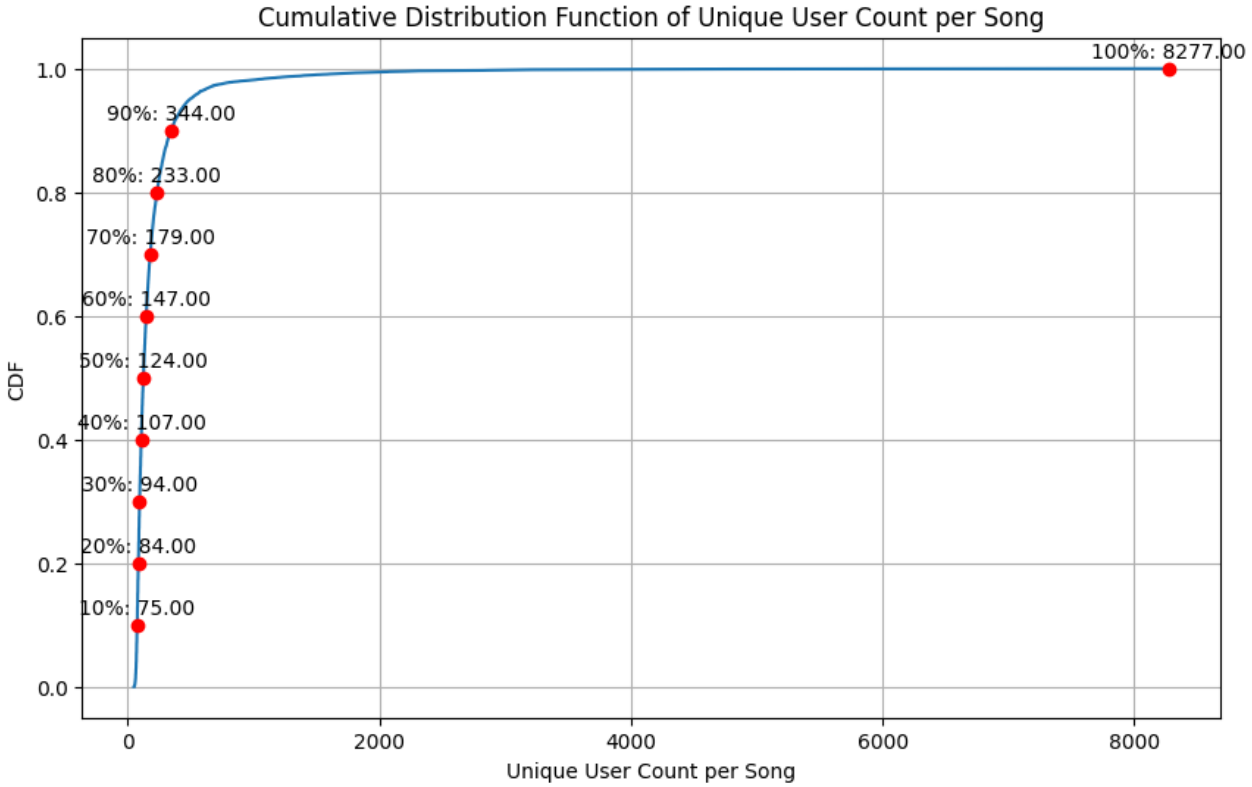
3) User Song Preference (Average Unique Song Play Count per User):
Since some users play the same song repeatedly, which can skew the data, I will not use this for the filter criteria.

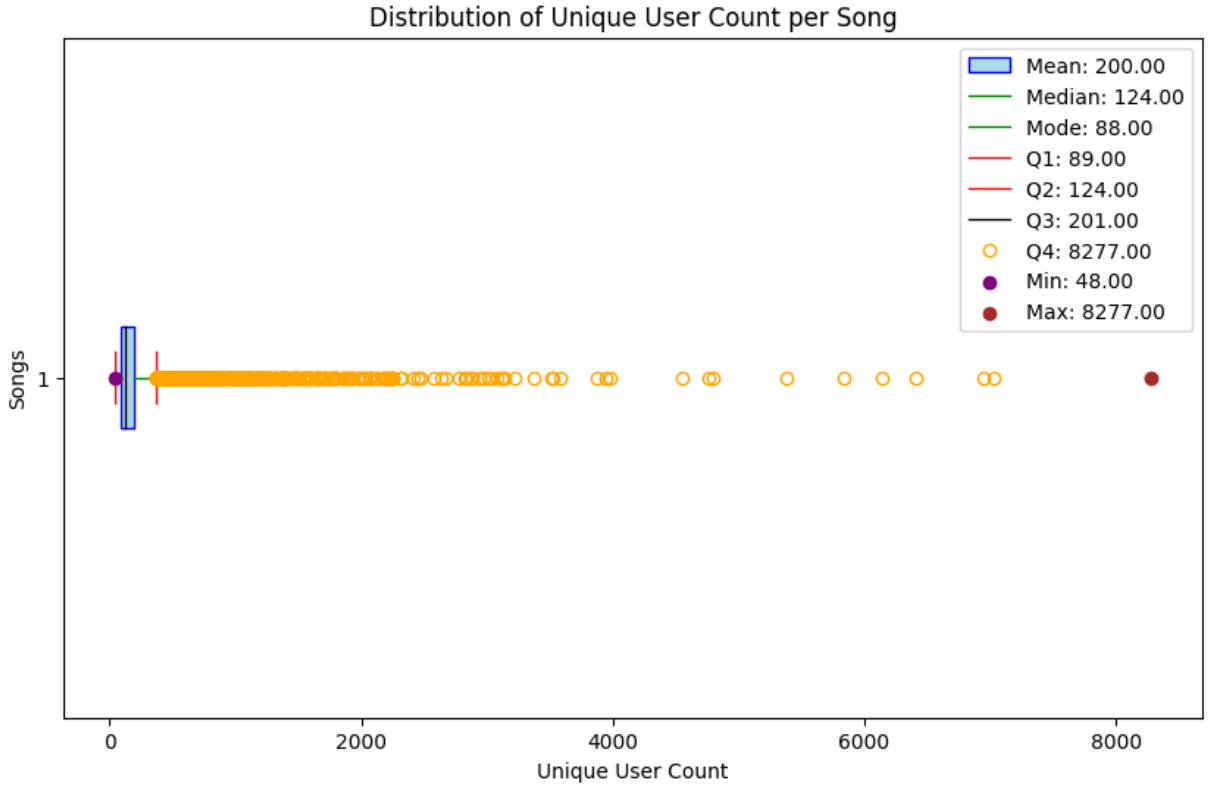




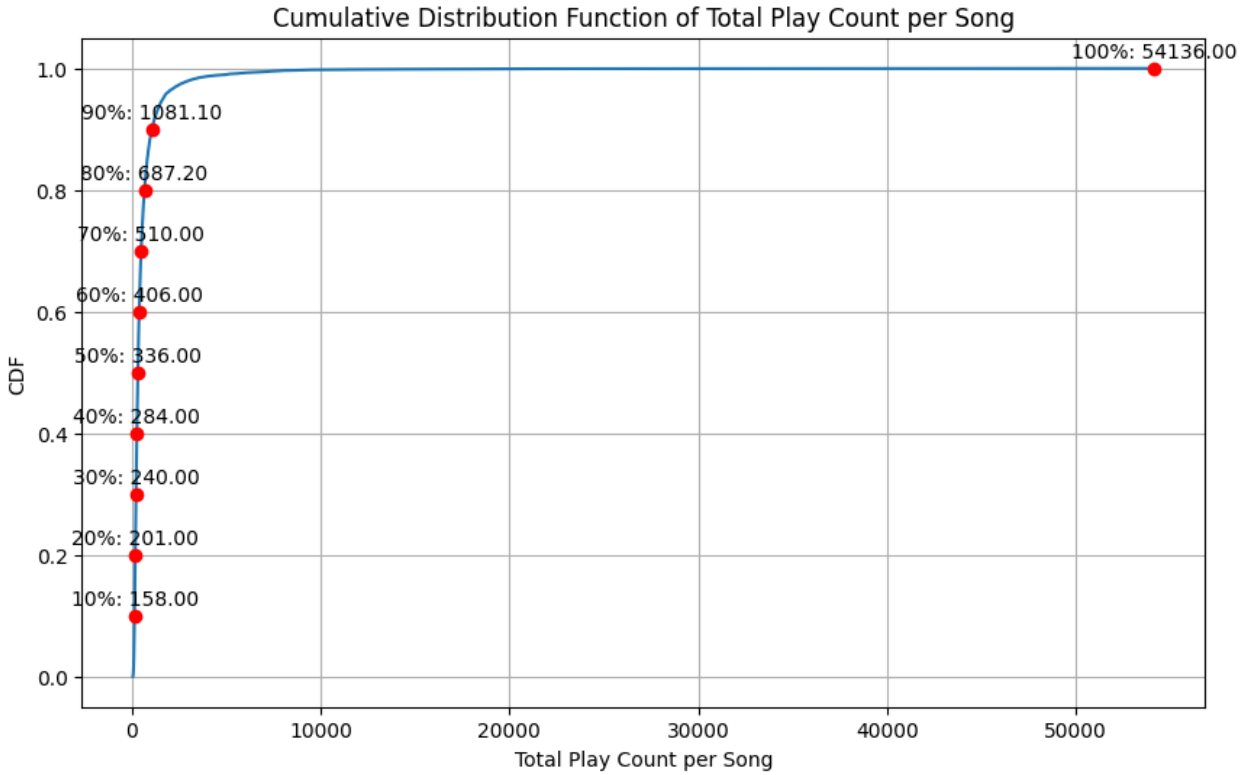
4) Song User Preference (Average User Play Count per Song):

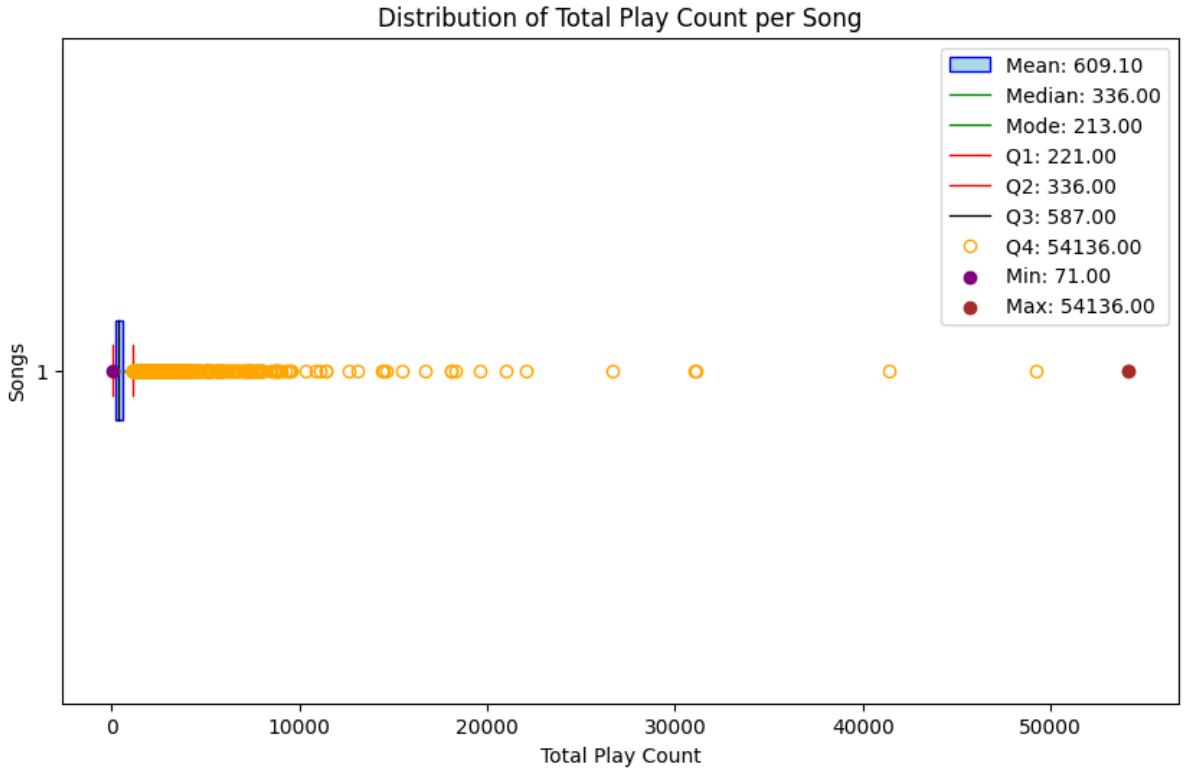
I will filter a list of songs that had an average user play count of over 2 times





5) Song Discovery (Unique Users Count per Song):
I want to include songs discovered by a small number of people to diversify the recommended song list.





6) Song Play Total Count:

This measure does not give a good picture of the popularity of songs.

Final Data Population Data Analysis

Final Data Population:

- **A list of users who played at least 80 times and 3 unique songs.**
- **A list of songs that had an average user play count of over 2 times.**
- **Added a new Song Popularity Score column**

Song Popularity Score is calculated using weighted average play per song (60%) and total play count (40%) and normalized to represent the population ranking.

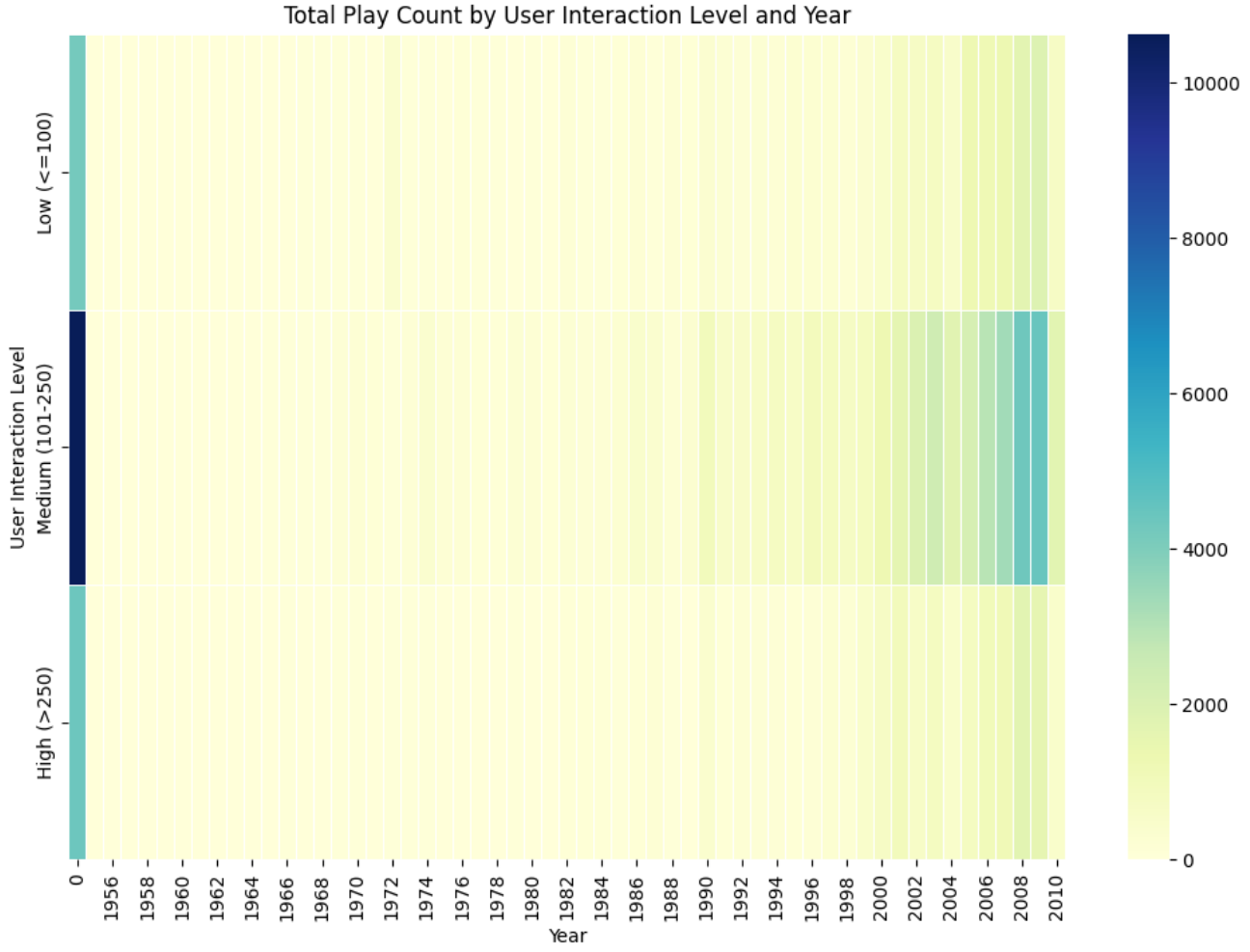
940,730 rows and 8 columns, 20,642 unique users, 7,711 unique songs, 3,109 unique artists, 4,772 unique release names, and release years ranging from 1954 to 2010.

Each user played a minimum of 3 unique songs, which were played at least 1.5 times and discovered by at least 14 users.

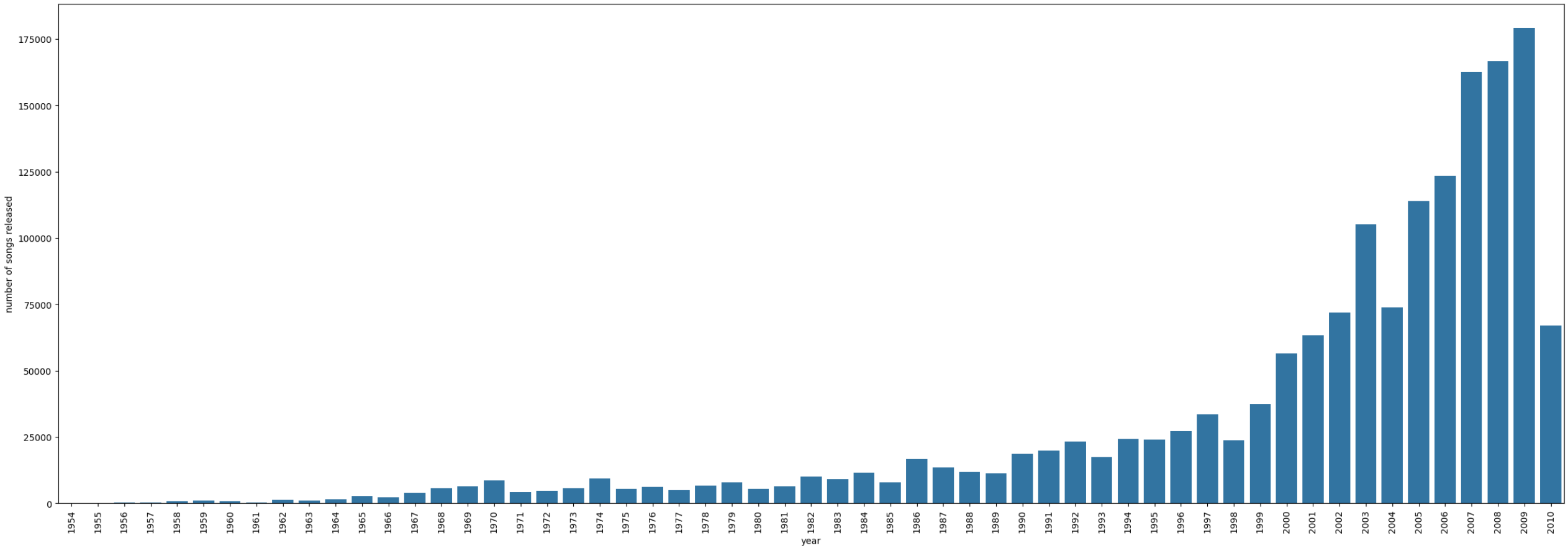
The dataset also contains a range of songs from infrequently played to the most popular ones. The average song gets played 4 times.

Most songs have low engagement in total plays and unique user plays. This indicates the importance of addressing data sparsity and popularity bias in the models.

The current dataset contains majority of users have a medium interaction with the platform, with an average 191 total play count, while a small portion of users have high interactions (max: 4,350 total play count).



There is significant user interest in both classical and contemporary music.



Model Experiments and Comparisons

Rank Based Recommendation System

A rule-based ranking system that is suitable for low engagement and cold-start users. Easy to implement, scale, and can customize logic conditionally.

Logic 1: Popular Songs

10/30 songs from top 20 most popular based on Population Score.

Logic 2: Discovery Songs

20/30 songs from top 2 average play count songs in each popularity percentile (5%-90%).

Application

Recommend a mix of songs to users until more user-song interaction data is collected.

Improve the logics by capturing some basic information at time of new user sign up.

Advacned Model Performance Metrics

MAE & RMSE (KNNBasic, SVD)

Mean Absolute Error and Root Mean Squared Error to evaluate prediction accuracy.

Silhouette Score & Davies-Bouldin Index (KMeans)

Used for Clustering models to assess cluster quality.

Precision, Recall, F1, MAP, NDCG, Novelty (Recommendation System)

Used for evaluating recommendation quality at top 30 items.
Threshold=1.5, Confidence =2.5

User-User Similarity-based (KNNBasic)

The model recommends the top N new songs based on predicted play counts inferred from frequently played songs from similar users. This model does not perform well with sparse data. Sample user 22570 is predicted to play 2.5 times for song 1286 vs an actual play count of 6.

Default Model

Parameters: K=30, Pearson_baseline. MAE: 0.47, Precision: 0.151, Recall is 0.151. Underperforms in recommendation quality despite reasonable prediction accuracy.

On average, 15% of the ideal ranking quality has been achieved.

Optimized Model

Adjusted Parameters: K=10, Min_k=3. MAE 0.46, Precision 0.115, Recall is 0.115. Improved accuracy but reduced recommendation quality.

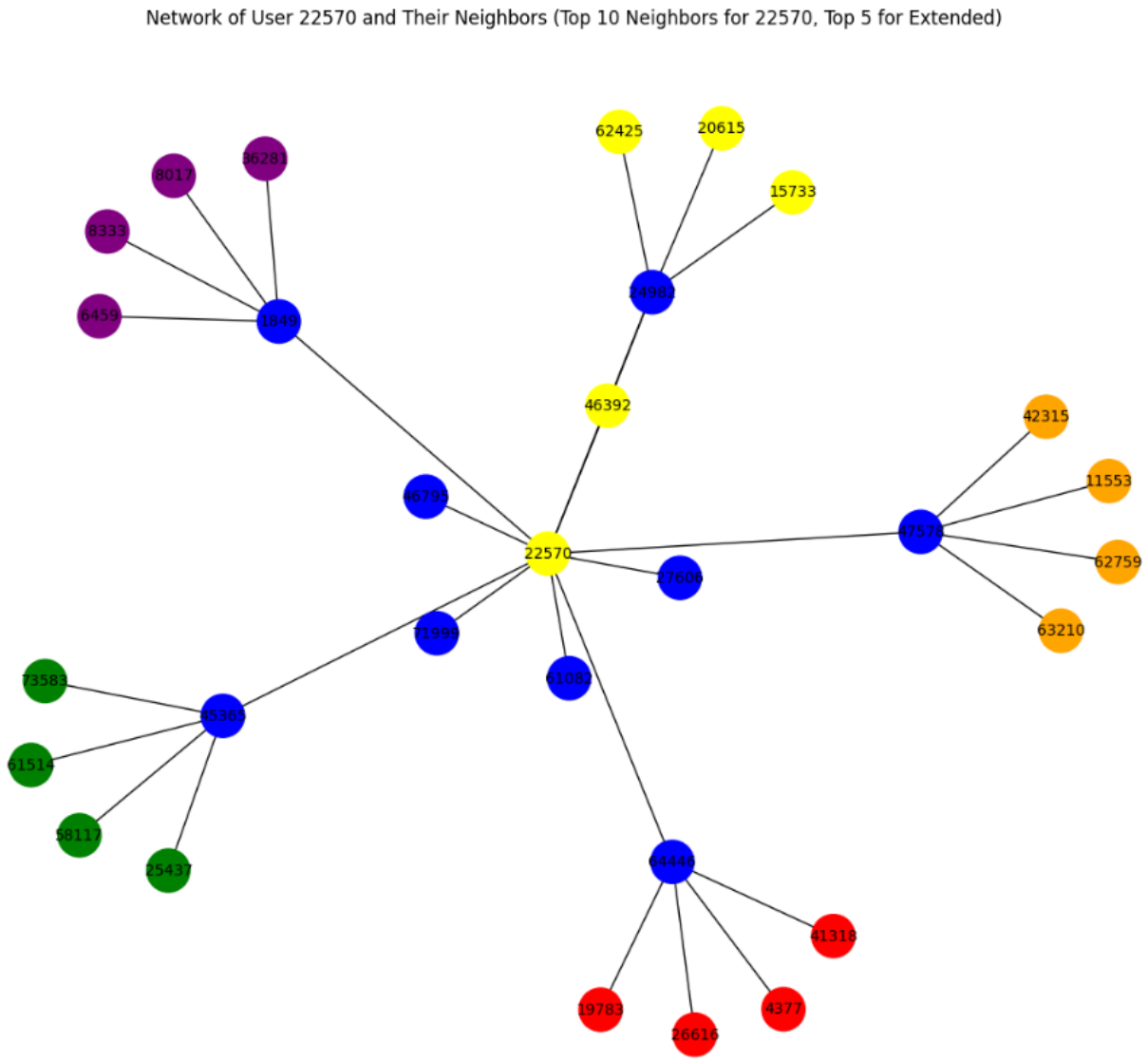
On average, 12% of the ideal ranking quality has been achieved.

Key Considerations

Log transformation is used to scale Play Count taken to address skewed data issues.

80/20 test split to ensure enough data is used to train the model.

Optimizing MAE during grid search for low error across the board (predicting songs generally will be preferred by users) vs penalizing a few edge cases.



An illustration of sample user 22570 with top 10 user neighbors and their top 5 neighbors.

Item-Item Similarity-based (KNNBasic)

The model recommends the top N new songs based on predicted play counts inferred from similar songs of frequently played songs by a user. This model does not perform well with sparse data. Sample user 22570 is predicted to play 2.5 times for song 1286 vs an actual play count of 6.

Default Model

Parameters: K=30, Pearson_baseline. MAE 0.46, Precision 0.153, Recall 0.153. Better novelty but still underperforms in recommendation relevance.

On average, 15% of the ideal ranking quality has been achieved.

Optimized Model

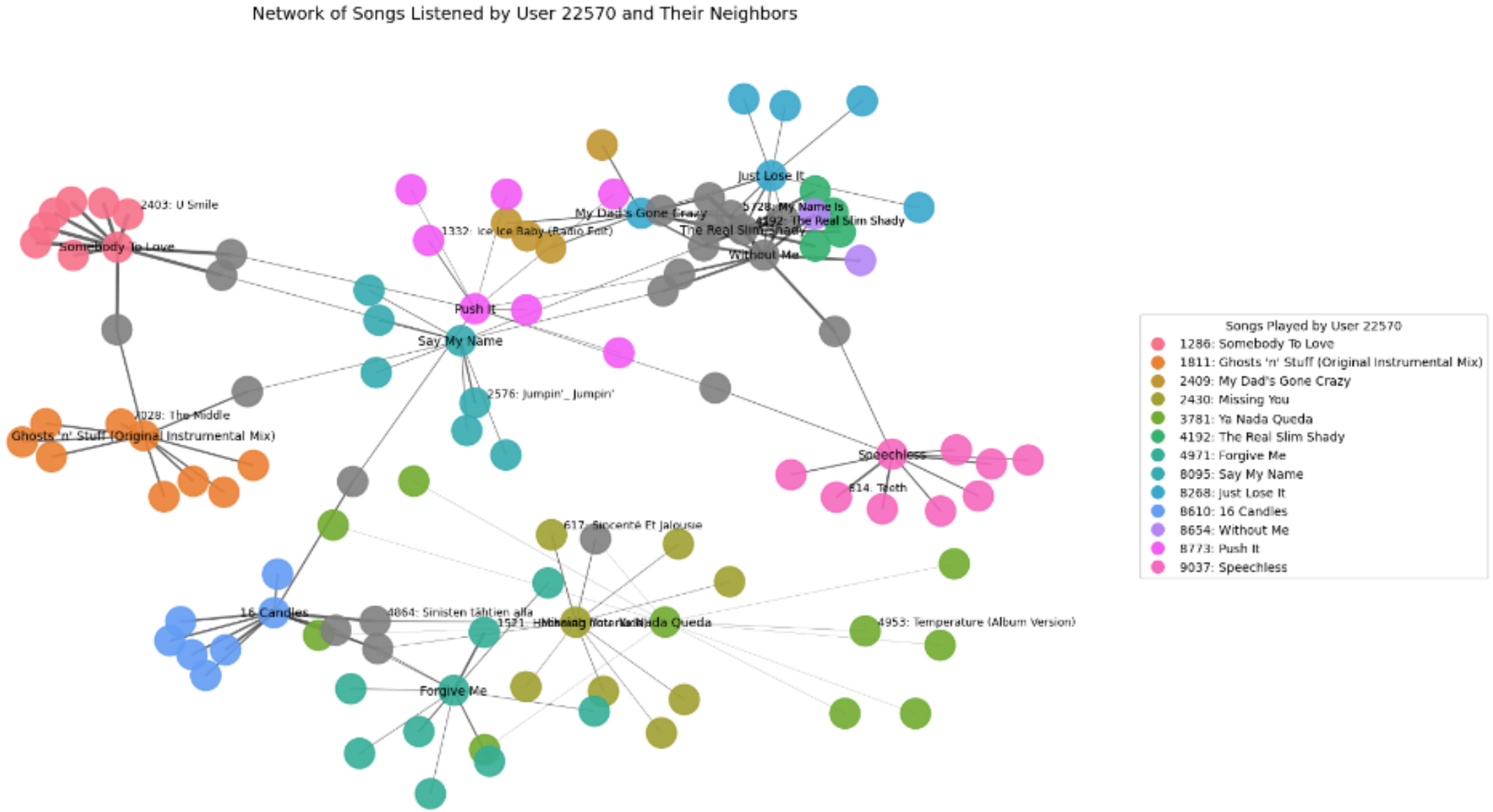
Increased Min_k=3. Similar accuracy (MAE 0.46) and reduced recommendation quality (Precision 0.116, recall 0.116).

On average, 12% of the ideal ranking quality has been achieved.

Key Considerations

Both user-user and item-item KNNBasic optimized models show identical performance metrics and prediction results. This can be caused by:

- Users or items with very similar interaction patterns
- The set of users or items that are used for making predictions may overlap significantly



An illustration of song neighbors for sample users. Songs that are shared neighbors have a gray node color

Matrix Factorization-based (SVD)

The model reduces the dimensionality of the data and captures the most important patterns while ignoring noise in the data. This model shows worse prediction accuracy and recommendation quality than the User-User/Item-Item similarity models. Sample user 22570 is predicted to play 2.5 times for song 1286 vs an actual play count of 6.

Default Model

Parameters: n_epochs=20, lr_all=0.005. MAE: 0.492, Precision: 0.025, Recall: 0.026.

On average, 2.5% of the ideal ranking quality has been achieved.

Optimized Model

Adjusted n_epochs=30, reg_all=0.2. Similar MAE (0.492), Precision: 0.024, Recall: 0.024.

On average, 2.4% of the ideal ranking quality has been achieved.

Key Considerations

This model is struggling to balance predictive accuracy with meaningful recommendations.

Data sparsity with many less popular items could negatively affect the quality of recommendations.

I can try to introduce more content features (e.g., song metadata, user profiles) to assist the model in making better recommendations. Or combing SVD with other models (e.g. KNN, content-based).

Clustering Model (User-Item Hybrid KMeans)

The model clusters both users and songs and predicts play counts based on other users and songs in the same cluster. It then averages the two for the final predicted play count. The model doesn't handle cold-start users well and needs sufficient data to make accurate predictions. Model performance is evaluated based on how well similar users or songs are clustered together. Sample user 22570 is predicted to play 5.3 times for song 1286 vs an actual play count of 6 (optimized model).

Default Model

Parameters: n_clusters=8, N_init=1.

Silhouette Score User : 0.341.
Silhouette Score Item : 0.362.
Davies-Bouldin Index User : 0.826

MAE: 0.617
Precision: 0.87
Recall: 0.87

On average, 87% of the ideal ranking quality has been achieved.

Predicted 8.5 times for sample user 22570 and song 1286.

Optimized Model

Adjusted: n_clusters_user=7, n_clusters_item=5, N_init=10.

Silhouette Score User : 0.356.
Silhouette Score Item : 0.365.
Davies-Bouldin Index User : 0.806

MAE: 0.527
Precision: 0.68
Recall: 0.64

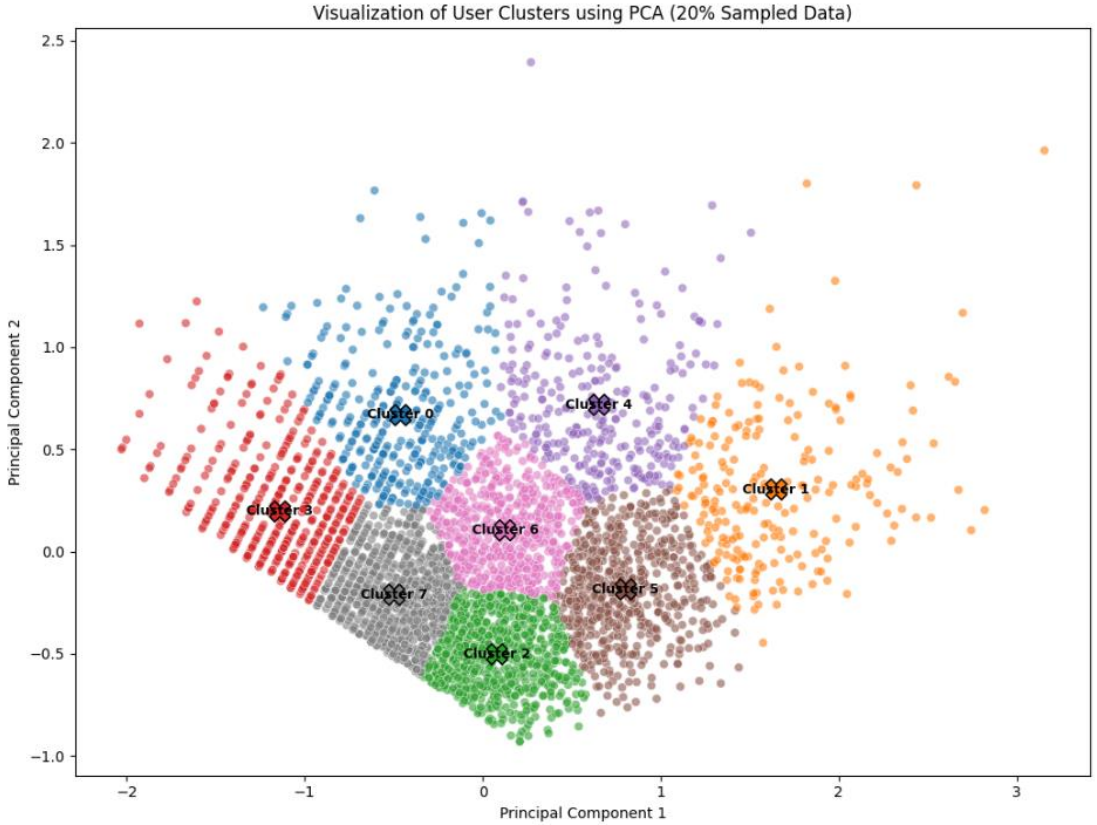
On average, 68% of the ideal ranking quality has been achieved. Better clustering and prediction accuracy but worse recommendation quality.

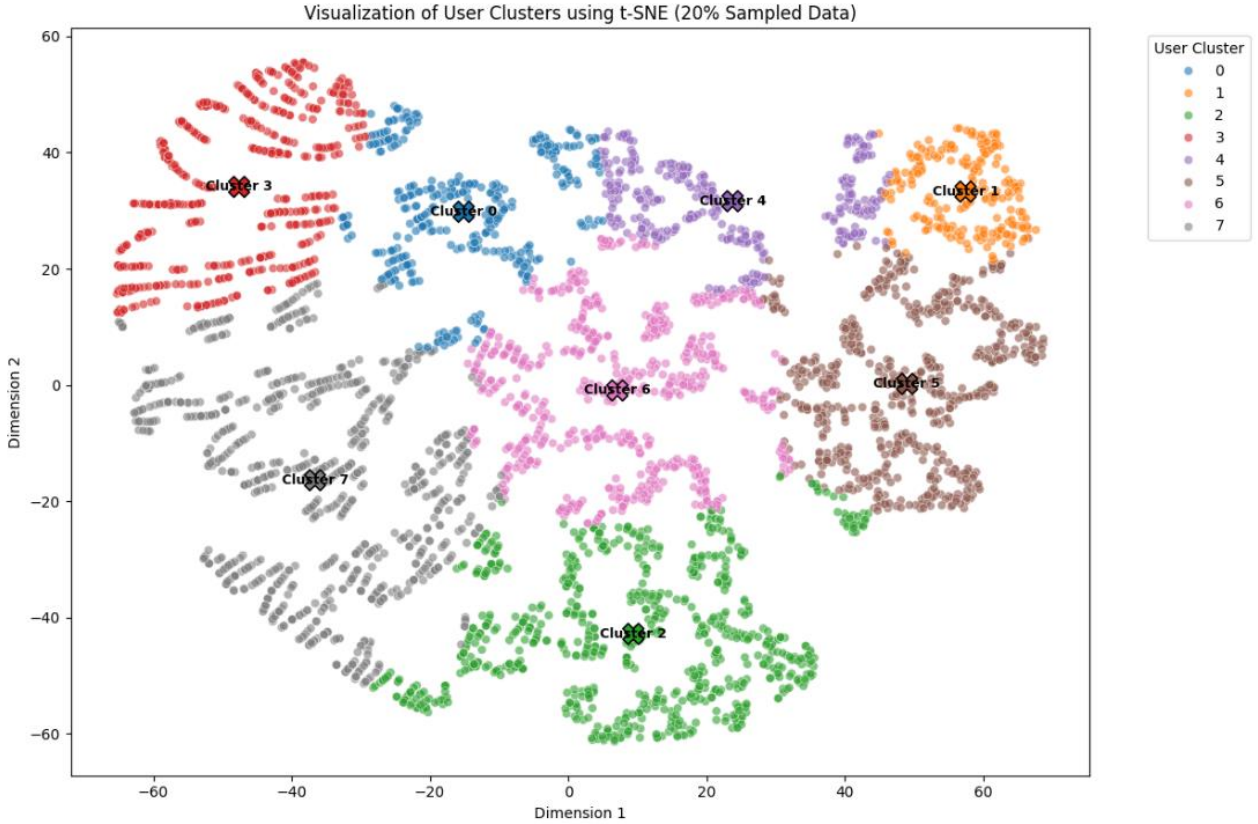
Key Considerations

Significantly out-performed previous models in both prediction accuracy and ranking quality.

Compute intensive, can leverage GPU or TPU accelerator.

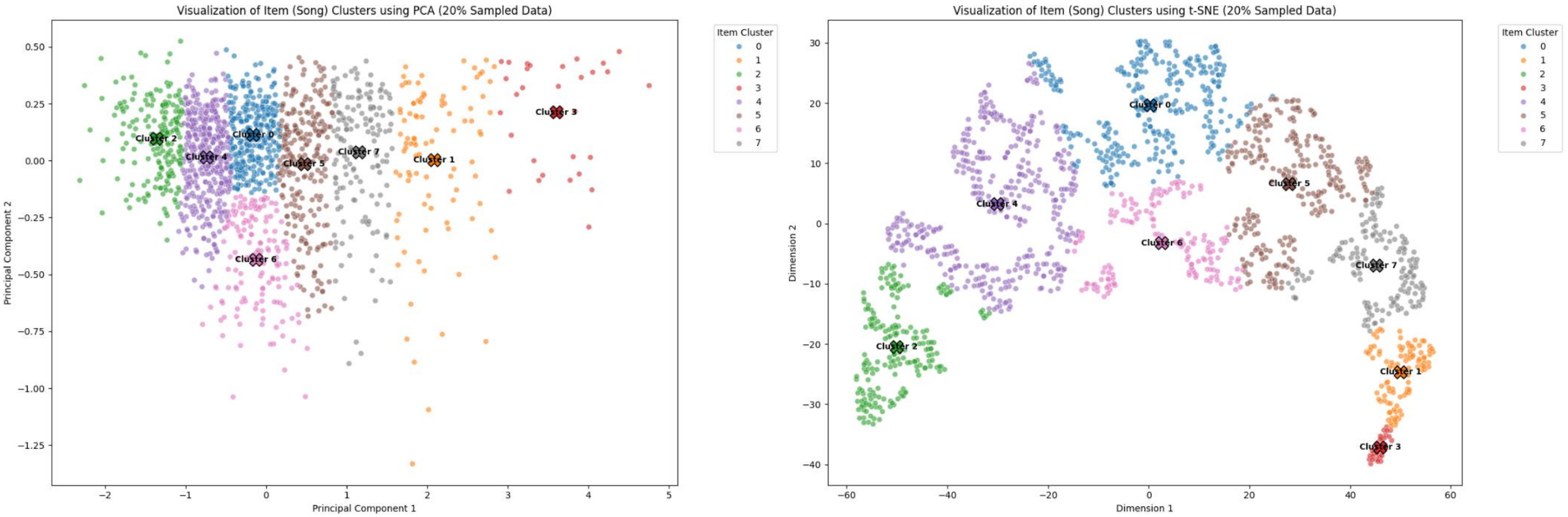
Can also try using KMeans with a collaborative filtering approach to recommend songs (vs predicting play count).





An illustration of user clusters (default KMeans model using 20% sample data)

Cluster # - user count: 0 - 1655, 1 - 1150, 2 - 3888, 3 - 2478, 4 -1433, 5 - 2758, 6 - 2969, 7 - 4314



An illustration of song clusters (default KMeans model with 20% sample data)
Cluster # - song count: 0 - 675, 1 - 857, 2 - 1839, 3 - 169, 4 -1221, 5 - 457, 6 - 1430, 7 – 1063

Content-based Model (TF-IDF)

A simple Natural Language Processing technique to numerically represent text data by considering the importance of words in a document relative to a corpus of documents. Text from the song's combined Title, Release, and Artist Name are tokenized. The similarity between songs is measured using the cosine similarity metric. The system takes a song input and recommends a list of songs based on the highest similarity scores. For song 1286 "Somebody to love" by Justin Beaver, all top 5 recommended songs were from Justin Bieber (Love Me, That Should Be Me…etc).

Text Representation	Similarity Measures	Key Considerations
Used combined song metadata (titles, artist). Simple implementation but lacks deep semantic understanding.	Cosine similarity for recommending new songs based on user song history.	Match exact words vs considering the contextual meanings. The user's metadata is not being considered. Suitable for cold-start or low-engagement users. Multiple options to take an input song explicitly or implicitly (e.g. user searching a song, user preference questionnaire).

Content-based Model (DistilBERT)

A transformer model that can weigh the importance of different words in a sequence when processing information and produces contextualized embeddings for each word. This is a lighter version of BERT (6 vs 12 layers) with a high accuracy close to BERT. For song 1286, “Somebody to Love” by Justin Beaver, “I Gotta Feeling” by The Black Eyed Peas were recommended. However, “Orgelblut,” an instrumental music, was also recommended.

Text Representation	Similarity Measures	Key Considerations
Pre-trained DistilBERT for embeddings. Captures contextual relationships.	Top similarity scores for personalized but novel recommendations.	It is more robust than TF-IDF, where the context of words is considered. It would perform better when more song meta (e.g. genre, rhythm type). Suitable for all types of users. Can combine with other models to improve diversity in recommendations.

Graph Neural Network Model with DistilBERT

The model uses Graph Convolutional Network Convolution layers to gather feature vectors of each node's immediate neighbors and aggregate these features to make predictions. Users and songs are nodes; play count is the edge; 2 layers of GCNConv. Song meta embedding from the DistrillBERT is passed along with Song Popularity Score to the neural network as an input feature. The model predicted the play count for a user-song pair input. Sample user 22570 is predicted to play 5.1 times for song 1286 vs an actual play count of 6 (optimized model).

Model Architecture:

In_channels: 769 of input features per node (embedding size 768 + song popularity score 1).

Hidden layer size: 64 number of neurons in the hidden layer.

Out_channels: 1 output features (play count prediction).

Number of nodes: 20,642 unique users plus 7,711 unique songs

Training Parameters:

Learning rate: 0.0001.

Optimizer: Adam.

Epochs: 30.

Performance:

MAE: 9629, Precision/Recall: 1.0

Key Considerations:

GNN can learn complex representations but require more computational resources and are more prone to overfitting.

Best model with high potential on improvements so far.

Next step to improve the model:

- Investigate high MAE score.
- Add user profile data, add more song meta.
- Hyperparameter tuning using on hidden layer size, learning rate, and number of epochs.
- Change activation function from ReLu to Leaky ReLU to accommodate skewed and sparse data.
- Leverage GPU or TPU accelerator on training.

Production deployment challenges:

- Implementation at scale (high memory consumption, long training time, frequent re-training need)
- Complex data pipeline (data ingestion & processing)
- Difficult debugging & monitoring graph changes

Conclusion and Recommendations

Given the complexity of our goal, I propose to start experimenting with a combination of models in production while continuing experimenting to find better-performed models:

Cold Start. For new customers, we should collect some basic information on their preferences and then use the rule-based recommendation system to recommend top popularity songs per their preference and also use the hybrid clustering model to recommend similar songs from the same cluster of those in their preference.

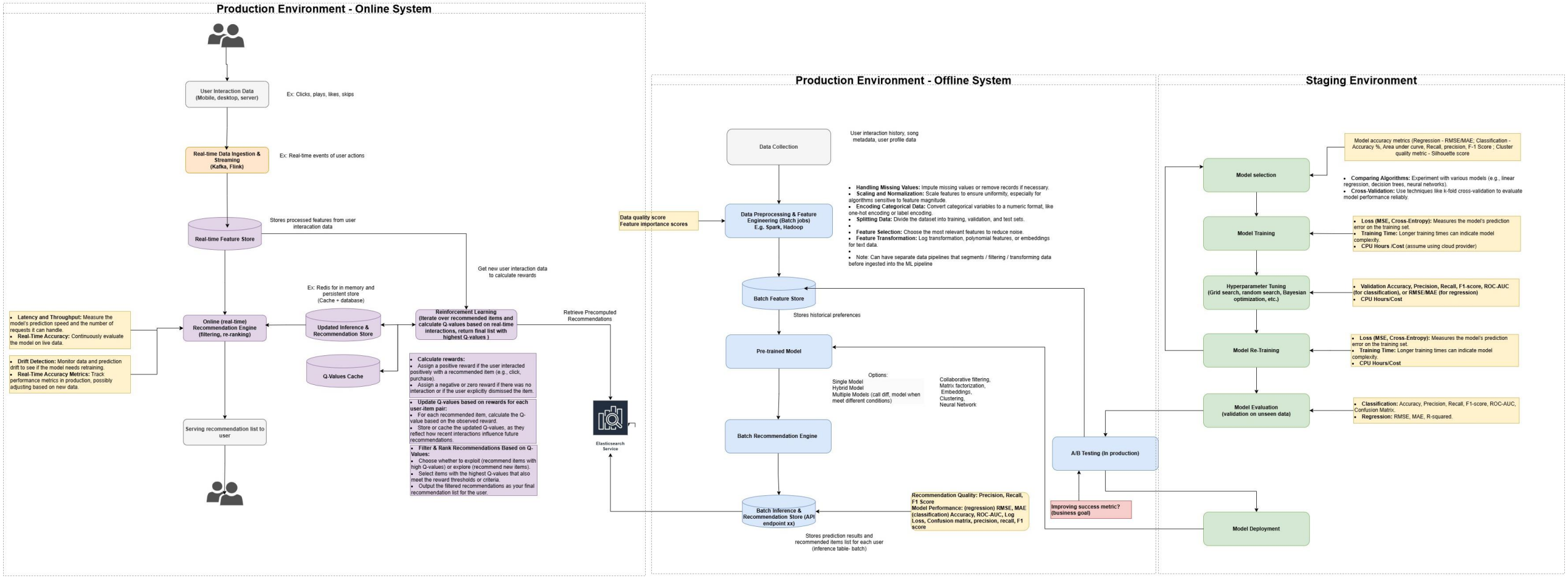
Existing customers with low engagement. We should create a mix of songs from the hybrid clustering model and the DistilBERT model, as well as a few highly popular songs from the rule-based ranking. Popular songs will attract users to the platform, and songs from other models will promote discoveries. With more data we collect, we can improve recommendation quality and drive more engagement.

Existing customers with low discovery. For those users, we can focus on recommending songs using the hybrid clustering and DistilBERT models to drive discovery.

In addition, to promote new songs created or newly onboarded to the catalog, we should cluster those songs and also **explicitly create a new song playlist to recommend to users**. By combining different models for different segments of users, we will be able to drive better engagement and experience.

Future steps include experimenting with the Hybird GNN model with NLP, improving data quality, feature engineering, model tuning, exploring alternative recommendation algorithms, and balancing accuracy with recommendation quality. Continuous monitoring and user feedback integration are essential for improvements. We should also design the system with an online system for reinforcement learning and an offline model re-training for scalability in the future.

Recommendation System Sample Architecture Diagram



Appendix

Top 30 recommendations	Range	User-based KNN Basic Default	User-based KNN Basic – MAE Optimized	Item-based KNN Basic Default	Item-based KNN Basic – MAE Optimized	Matrix Factorization-SVD Default	Matrix Factorization-SVD Optimized	Hybrid User/Item Clustering-KMeans	Hybrid User/Item Clustering-KMeans Optimized
Model Parameters		K=30, Pearson_baseline, User-based, Min_k=1	K=10, Pearson_baseline, User-based, Min_k=3	K=30, Pearson_baseline, Item-based, Min_k=1	K=30, Pearson_baseline, Item-based, Min_k=3	n_epochs=20, lr_all=0.005, reg_all=0.02	n_epochs=30, lr_all=0.01, reg_all=0.2	n_cltr_i = 8 n_cltr_u = 8 , N_init = 1	n_cltr_i = 5 n_cltr_u = 7 N_init = 10
MAE	0(no error)	0.4659	0.464	0.4647	0.4632	0.4920	0.4920	0.617	0.527
RSME	0(no error)	0.6614	0.6484	0.6586	0.6482	0.6488	0.6488	0.747	0.673
Precision	1(100% relevancy)	0.151	0.115	0.153	0.116	0.025	0.024	0.87	0.678
Recall		0.151	0.115	0.153	0.116	0.026	0.024	0.866	0.603
F1 Score	0(perfect predictions)	0.151	0.115	0.153	0.116	0.025	0.024	0.868	0.638
MAP		0.151	0.115	0.153	0.116	0.026	0.024	0.87	0.678
NDCG		0.151	0.115	0.153	0.116	0.025	0.024	0.87	0.678
Novelty	0(no new items)	0.325	0.331	0.32	0.333	0.348	0.347	0.216	0.264

Top 30 recommendations	Range	User-based KNN Basic Default	User-based KNN Basic – MAE Optimized	Item-based KNN Basic Default	Item-based KNN Basic – MAE Optimized	Matrix Factorization-SVD Default	Matrix Factorization-SVD Optimized	Hybrid User/Item Clustering-KMeans	Hybrid User/Item Clustering-KMeans Optimized
Model Parameters		K=30, Pearson_baseline, User-based, Min_k=1	K=10, Pearson_baseline, User-based, Min_k=3	K=30, Pearson_baseline, Item-based, Min_k=1	K=30, Pearson_baseline, Item-based, Min_k=3	n_epochs=20, lr_all=0.005, reg_all=0.02	n_epochs=30, lr_all=0.01, reg_all=0.2	n_cltr_i = 8 n_cltr_u = 8 , N_init = 1	n_cltr_i = 5 n_cltr_u = 7 N_init = 10
Silhouette Score (cluster classification metric)	1(well defined)							User:0.341 Item: 0.362	User:0.356 Item:0.365
Davies-Bouldin Index (cluster distance metric)	>0(more overlaps)							User: 0.826 Item: 0.836	User:0.806 Item:0.831
Predicted play count on song 1283 for user 22570 (actual = 6)		2.53	2.54	2.54	2.54	2.54	2.54	8.52	5.26
User not played the song:(User ID: 63855 ; Song ID: 1286)		2.53	2.54	2.54	2.54	2.54	2.54	10.96	5.31

Approaches	Key Characteristic	Pro	Con
Ruled-based ranking	Leverages universal user population's preference	Suitable for cold-start problems (user hasn't interacted with the system). Simple, interpretable, quick to deploy.	Lack of personalization and flexibility, everyone gets the same recommended products
User-based collaborative filtering – optimized KNNBasic	Leverages the other similar user's rating behavior on an item	Personalized, interpretable	Require good amount data to find similarities among users. (not enough overlap in the items that the target user and similar users have rated) Computation cost increases as number
Item-based collaborative filtering - optimized KNNBasic	Leverages the same user's rating behavior on similar items	More robust to sparsity (items tend to have more overlap in ratings from different users.) Stable over time, adding new users is more efficient in compute than user-based KNN.	Require good amount of historical product ratings to perform well. Computation cost increases as the amount of products increases. Cold start for new items with no ratings.
Matrix Factorization-optimized SVD	Leverages underlying patterns and relationships between users and items.	Highly personalized, scalable on large datasets after factorization is complete. Works well to find missing ratings. Better cold start handling than KNN.	More complex to implement and less interpretable. Still struggles with users who have no history or very limited interactions. Compute heavy for very large datasets.

Approaches	Key Characteristic	Pro	Con
Clustering KMeans Optimized User-Item Hybrid	Clusters users and/or items (songs) based on similar behavior (e.g., play counts) to make recommendations.	Captures user-item interactions by clustering similar users and items. Can be tailored for personalized recommendations. Simple and scalable for large datasets.	Doesn't capture deep semantic relationships in metadata (e.g., titles, genres). Requires careful tuning of hyperparameters (number of clusters). Less effective on sparse datasets with limited play history.
Content-Based (TF-IDF)	Text-based method that focuses on word frequency in song metadata (titles, lyrics, genres) to recommend similar items.	Simple, fast, and computationally efficient. Works well when metadata (e.g., titles, lyrics) is the primary data source of data. No need for user interaction data.	Lacks contextual or semantic understanding (exact word matches only). Doesn't incorporate user preferences or behavior. Limited in personalization and deeper recommendations.
Content-Based (DistilBERT)	Transformer-based NLP model that generates contextual embeddings for song metadata, understanding meaning and relationships between words.	Captures semantic relationships and context (e.g., synonyms). Excellent for content-based recommendations with text-heavy data. Adaptable to various NLP tasks beyond recommendations.	Computationally expensive, requires more resources (GPUs). Slower inference compared to simpler models like TF-IDF. Requires fine-tuning for best results on domain-specific data.

Thank you!