MUSIC RECOMMENDATION SYSTEM

EXECUTIVE SUMMARY

Objective and Dataset:

- Build a recommendation system to propose 10 songs for a user, based on the likelihood of listening to those songs.
- Building 6 different types of recommendation system. Popularity Based Recommendation System, Collaborative Recommendation Systems (User-User Similarity-Based or Item-item Similarity-Based), Model Based Collaborative Filtering –Matrix Factorization, Cluster Based Recommendation Systems and Content Based Filtering.
- We have 2 datasets. “count data”, “Song data” . The count data included features like "Unnamed: 0", "user_id", "song_id", "play_count". Song data included features like **"song_id", "title", "release", "artist_name", "year"**.

Approach:

- Used Google Colab to build the recommendation systems, loaded necessary libraries and imported the datasets.
- Introduced Surprise library to build similarity-based filtering. Used the same to predict ratings by a user for both a listened and non-listed song.
- Implemented the matrix factorization approach with SVD (Singular Value Decomposition) algorithm to predict the ratings given by the user. Created a function to recommend 10 songs to the users based predicted ratings for each song.
- Tuned all the models using GridSearchCV
- Used precision@k, recall@k , F_1 score and RSME to measure the performance of the recommendation systems.

Key Findings:

- Provided song recommendations to users using all 6 algorithms.
- Performance of all the models were somewhat similar. The clustering-based recommendation system had the lowest F_1 score as 47.2% in the baseline model and 46.5% in the optimized model.
- Further to improve model performance using hyperparameters tuning.

Overview - The Quick Pitch

This cap stone project is focused on solving a business problem, which is to build a recommendation system for Spotify to propose 10 songs for a user, based on the likelihood of listening to those songs.

As being one of the largest music service providers, with 422 million active users, 182 million subscribers and approximately 82 million songs, Spotify is highly dependent on its ability to recommend the "best" next song to its consumers. Among the countless of song choices, making strategic judgements and offering the best songs to consumers is done by utilizing recommendation systems.

In this project, six different recommendation systems have been built including population based, user-user similarity-based collaborative based, item-item similarity-based collaborative based, model based (matrix factorization), cluster based and finally content-based recommendation system. Focusing on the business objective, the performance of all of models has been evaluated, and appropriate model implementation recommended.

The Problem

Countless of businesses fail or lose competitive advantage every year because of unidentified, unrecognized problems with their operations such as not being able to keep up with their consumer preferences, trends, and potential opportunities. Many of these problems which data science address have existed for a long time however with incremental data accumulation and availability of data have made data scientist methodologies and techniques even more valuable than ever.

Spotify's music business has been a real source of strength, driving strong revenue growth and gross margin expansion.

Spotify went public in 2018, evolved dramatically over the last four years—expanding beyond music to become the leading player in audio. Spotify's goal is to make Spotify available to anyone on any device.

Spotify continues to grow, so too do the industries they play in. They believe the music streaming market alone has room to expand from $30 billion to nearly $80 billion in the next 10 years.[1]

Spotify's significant growth in the market is highly depended on its recommendation system. To maintain its market leader position, keep up with competition in the music sector, Spotify is challenged to produce innovative methods to keep its consumer base active and engaged.

Spotify runs on a Freemium model (a combination of "free" and "premium" model*), makes money from subscriptions and advertisements. 91% of the revenue is gained from subscription and the other 9% is gained from advertisements. Building new recommendation systems, providing continues improvements to its existing recommendation systems is essential for the company's revenue model.

[1] https://newsroom.spotify.com/2022-06-08/spotify-shares-our-vision-to-become-the-worlds-creator-platform/
Recommendation systems are very useful. Because they identify relevant products, personalized contents to the users, improves the users' engagements and deliver the right products to the right users.

It is paramount for Spotify to make good recommendations to the users early on, otherwise, there is a high potential risk of losing the consumers. As the competition significantly increases among the audio streaming companies (such as Amazon, Apple etc.) and new players comes in the town, the companies who design recommendation systems sensitive to user' and creators' wishes will continue to increase their competitive advantages and market positions. For the scope of the project, when designing recommendations systems, the user's benefit is considered only.

The Solution Summary

Data Preparation and Exploratory Analysis

Data Preparation:

- There are 2 datasets (count_data and song_data) provided to create music recommendation system.
- After loading the datasets, created dataframes and created copies of the dataframes.
- Checked the record counts, # of rows, # of columns, data types, missing values, memory usage in each data frame

Count_df:

- There are **2,000,000 rows** and **4 columns**.
- The **column names** are **"Unnamed: 0", "user_id", "song_id", "play_count"**.
- The "user_id" and "song_id" columns have **object** data types and "Unnamed: 0" and "play_count" columns have **integer** data types.
- The "Unnamed: 0" column s/b deleted since this column does not add any value to the models.

Song_df:

- There are **1,000,000 rows** and **5 columns**.
- The **column names** are **"song_id", "title", "release", "artist_name", "year"**.
- 4 columns have object data types, and 1 column has integer datatype.
- The "title" and "release columns" have some **missing values** (15 and 5 respectfully).

Exploratory Analysis:

- Total number of unique user id: 3155
- Total number of unique song id: 563
- Total number of unique artists: 232
- The dataset contains songs with release year from 1969 to 2010.

- The top 5 number of titles of songs based on the released year 2009, 2008, 2007, 2003 and 2006 respectfully.
- The songs released in the most recent years are liked and played out mostly by users. Therefore, we can assume that new songs will be played out mostly compared to older songs. Keeping up and timely offering of new releases to the users are essential for the business success.

- **Most interacted users:**
  Users with User Id's 61472, 15733, 37049, 9570 and 23337a are the most active users.
- **Most interacted songs:**
  Songs with Song Id's 8582, 352,2220, 1118 and 4152 are played out based by most users.

The songs with highest play count percentages are:

	song	play_count	percentage
105	Dog Days Are Over (Radio Edit)-Florence + The ...	1634	82%
392	Sehr kosmisch-Harmonia	1583	79%
511	Use Somebody-Kings Of Leon	1463	73%
391	Secrets-OneRepublic	1427	71%
140	Fireflies-Charttraxx Karaoke	1291	64%
...	...	...	...
354	Phantom Part 1.5 (Album Version)-Justice	158	7.8%
69	Camaro-Kings Of Leon	157	7.8%
90	Crazy In Love-Beyoncé feat. Jay-Z	157	7.8%
228	In Person-The Pussycat Dolls	155	7.7%
14	Alaska-Camera Obscura	146	7.3%

Comparison of Models

1. Popularity Based Recommendation System,
2. User-User Similarity-Based Recommendation System,
3. Item - Item Similarity-Based Recommendation System,
4. Model Based Recommendation System – Matrix Factorization,
5. Cluster Based Recommendation System,
6. Content Based Recommendation System.

1. Popularity Based Recommendation System

In popularity-based filtering strategy, the suggested songs are based on the popularity or trend.

Disadvantage: Not personalized to users only takes popularity in account.

Methodology:
Created a function to find the top 10 songs for a recommendation based on the average play count of song. Added a threshold for a minimum number of play counts, which is 100, for a song to be considered for recommendation.

Based on the created function, top 10 recommended songs with song ids# 7224, 6450, 9942, 5531, 5653, 8483, 2220, 657, 614 and 352

Collaborative Filtering

There are multiple ways to implement collaborative filtering but the main concept to be grasped is that in collaborative filtering multiple user's data influences the outcome of the recommendation. and doesn't depend on only one user's data for modeling.

2. User-User Similarity Based Collaborative Filtering

The basic idea here is to find users that have similar past preference patterns as the user 'A' has had and then recommending him or her items liked by those similar users which 'A' has not encountered yet. This is achieved by making a matrix of items each user has rated/viewed/liked/clicked depending upon the task at hand, and then computing the similarity score between the users and finally recommending items that the concerned user isn't aware of but users similar to him/her are and liked it.

Disadvantage:
People's taste change from time to time and as this algorithm is based on user similarity it may pick up initial similarity patterns between 2 users who after a while may have completely different preferences. There are many more users than items therefore it becomes very difficult to maintain such large matrices and therefore needs to be recomputed very regularly. This algorithm is very susceptible to shilling attacks where fake users' profiles consisting of biased preference patterns are used to manipulate key decisions.

Model Performance Metric:
In the baseline model F_1 score was 50.4%. After tunning the model with different hyperparameters using GridSearchCV, F_1 score of the optimized model improved slightly (52.5%).

Top 10 recommended songs for user Id# 6958 using user-user similarity based collaborative filtering:

	song_id	play_freq	predicted_ratings	corrected_ratings
5	7224	107	3.141147	3.044473
2	614	373	2.525000	2.473222
4	5653	108	2.514023	2.417798
0	352	748	2.425000	2.388436
7	6450	102	2.394927	2.295913
9	8324	96	2.373359	2.271297
1	952	482	2.275000	2.229451
6	8831	107	2.271924	2.175250
3	6448	109	2.270051	2.174268
8	4831	97	2.250465	2.148930

3. Item-based Collaborative Filtering

The concept in this case is to find similar songs instead of similar users and then recommending similar songs to that user 'A' has had in his/her past preferences.

Advantages over User-based Collaborative Filtering Unlike people's taste, movies don't change. There are usually a lot fewer items than people, therefore easier to maintain and compute the matrices. Shilling attacks are much harder because items cannot be faked.

Model Performance Metric:
F_1 score for baseline model is ~40%. After tunning the model with different hyperparameters using GridSearchCV, F_score for optimized model improved to 49%.

Top 10 recommended songs for user Id# 6958 using item-item similarity based collaborative filtering:

	song_id	play_freq	predicted_ratings	corrected_ratings
0	2842	232	1.650010	1.584357
2	7921	220	1.542349	1.474929
1	9386	221	1.533849	1.466582
3	733	206	1.526769	1.457096
5	7470	158	1.534020	1.454464
7	5417	135	1.538866	1.452800
8	4631	120	1.539257	1.447970
9	1767	112	1.540675	1.446184
4	8577	187	1.511054	1.437927
6	657	151	1.518838	1.437459

4. Model Based Collaborative Filtering - Matrix Factorization

Model-based Collaborative Filtering is a **personalized recommendation system**, the recommendations are based on the past behavior of the user, and it is not dependent on any additional information. We use **latent features** to find recommendations for each user.

Model Performance Metric:
Baseline F_1 score: 49.8%
Optimized model F_1 score: 50.2%

Top 10 recommended songs for user Id# 6958 using matrix factorization:

	song_id	play_freq	predicted_ratings	corrected_ratings
6	7224	107	2.601899	2.505225
5	5653	108	2.108728	2.012502
9	8324	96	2.014091	1.912029
1	614	373	1.917197	1.865419
4	9942	150	1.940115	1.858465
0	5531	618	1.895254	1.855028
7	6450	102	1.952493	1.853478
3	657	151	1.926228	1.844849
8	4831	97	1.931137	1.829602
2	4811	160	1.869229	1.790172

5. Cluster Based Recommendation System

In **clustering-based recommendation systems**, explored the **similarities and differences** in people's tastes in songs based on how they rate different songs. We cluster similar users together and recommend songs to a user based on play counts from other users in the same cluster.

Model Performance Metric:
Baseline F_1 score: 47.2%
Optimized model F_1 score: 46.5%

Top 10 recommended songs for user Id# 6958 using cluster-based recommendation system:

	song_id	play_freq	predicted_play_count	corrected_play_count
9	7224	107	3.711503	3.614829
8	5653	108	2.903883	2.807658
5	6860	169	2.691043	2.614120
6	657	151	2.606354	2.524975
7	8483	123	2.582807	2.492640
0	8138	524	2.503481	2.459796
1	5943	423	2.436141	2.387519
3	8252	267	2.448189	2.386990
2	317	411	2.423946	2.374619
4	6378	183	2.444544	2.370622

6. Content-based Filtering

This filtering strategy is based on the data provided about the items. The algorithm recommends products that are similar to the ones that a user has liked in the past. This similarity (generally cosine similarity) is computed from the data we have about the items as well as the user's past preferences. It is based on the concept "show me more of what I have liked".

Disadvantage:

Different products do not get much exposure to the user. Businesses cannot be expanded as the user does not try different types of products.

Recommendation for Implementation

- In this capstone project, the business problem needed to be solved was to propose 10 songs for a user, based on the likelihood of listening to those songs. Per the template notebook recommendation, built six different recommendation system including popularity-based recommendation system, user-user similarity-based recommendation system, item-item similarity-based recommendation system, model based collaborative recommendation system, cluster-based recommendation system and finally, content-based recommendation system.

- Excluding the popularity-based recommendation system, for all other recommendation systems, RMSE and F_1 score (the harmonic mean of Precision@kand Recall@kused) results are used to assess performance metrics. Excluding popularity-based recommendation system, the worst performing model was the clustered based recommendation system. There were not huge differences among the rest of the model performances.

- As far as implementing recommendation systems, I would suggest using different models for different user groups.

- If the users are new to platform, **popularity based** or **content-based** recommendation system. Our data set did not include any demographic information such as age, gender, region, language preference etc. for the users. If we had the demographic data, I would have suggested building default popularity-based recommendation systems for different demographic groups. I.e., Top list 10 for women who are between ages 25-40 and located in USA.

- Content based filtering is one of the popular filtering methods. Model makes recommendations that is very niche and only particular person likes. New, unique items, things can be suggested to users with content-based filtering. However, for smaller streaming services it is a good approach to use. Because content-based filtering has ability to recommend things to users that is unique to their profiles. This recommendation system will be applied to the new user groups only considering the size of Spotify users' size.

- If the users have been using the platform for at the least 3 to 4 months, for this user group I would suggest using content based or hybrid recommendation systems. Typically compared to content-based filtering, collaborative filtering offers higher efficiencies and more accuracy. Hybrid recommendation systems overcome the limitation of both content based and collaborative based filtering methods.
- Expected Cost/Benefit Analysis

  Benefits - Users will be able to find relevant content within a vast, ever-growing song library. In finding songs they like; users will spend a lot of time on the platform. Spotify will generate more revenue because they will have more users on their platform. They will see increased revenue from premium "ad-free" subscriptions. They will see more advertising revenue from users engaging on the free tier, that has advertisements.

  Costs - There are many costs associated with building a recommendation system.

  People: Hiring the right team.
  Equipment: Purchasing computers, cloud hosting, and other materials to run and host the system.
  Content: Streaming music requires making a royalty payment to the song's composer and the owner of the recorded material.
  Marketing: Spotify needs to let the world know about their service that will recommend content to its users)

  Risks: At the onset, if there are not enough users interacting with songs on the platform, there might not be enough data to make accurate predictions. Spotify will need a diverse song catalog and diverse user-base to ensure many different song preferences can be served.

  Challenges: Spotify will need an aggressive marketing campaign to ensure the baseline number of users is achieved to ensure expected model performance. Spotify will need to offer a wide range of songs, which will likely increase their cost of doing business. Spotify's team will need to strictly adhere to their cadence of validating and re-tuning the recommendation model.

APPRENDIX

A. User-User Collaborative Recommendation System Overview
B. Item-Item Similarity Based Collaborative Recommendation Overview
C. Model Based Collaborative Recommendation System - Matrix Factorization Overview
D. Cluster Based Recommendation System Overview
E. Content Based Recommendation System Overview

A. User-User Collaborative Recommendation System Overview

- Built the user-user-similarity based using the "surprise", imported necessary libraries
- Created a function to calculate the RMSE, precision@k, recall@k and F_1 score.
- To compute precision and recall, a threshold of 1.5 and k value of 30 is taken for the recommended and relevant ratings
- The function returned precision and recall at k metrics for each user.
- Loaded the dataset which is a pandas dataframe, into a different format called surprise.dataset.DatasetAutoFolds which is required by this library. To do this we will be using the classes Reader and Dataset.
- Built the first baseline similarity-based recommendation system using the cosine similarity.
- KNNBasic is an algorithm that is also associated with the surprise package. It is used to find the desired similar items among a given set of items.
- Train the algorithm on the trainset and predict play_count for the testset.
- Computed basedline precision@k, recall@k, and f_1 score with k =30.

  Based line model:
  RMSE: 1.0878
  Precision: 0.396
  Recall: 0.692
  F_1 score: 0.504

  Observations and Insights:
  We have calculated RMSE to check how far the overall predicted play counts are from the actual play counts.
  Intuition of Recall - We are getting a recall of 0.69, which means out of all the relevant songs, 69% are recommended.
  Intuition of Precision: We are getting a precision of almost 0.40, which means out of all the recommended songs 40% are relevant.
  Here F_1 score of the baseline model is ~0.50. It indicates that mostly recommended movies were relevant and relevant movies were recommended. We will try to improve this later by using GridSearchCV by tuning different hyperparameters of this algorithm.

- Predicted play_count for a sample user with a listened song.

```
user: 6958       item: 1671       r_ui = 2.00   est = 1.80   {'actual_k': 40, 'was_impossible': False}

Prediction(uid=6958, iid=1671, r_ui=2, est=1.8009387435128914, details={'actual_k': 40, 'was_impossible': False})
```

  The output of prediction showed that the actual rating is not so far from the predicted rating for this user-item pair by this user-user-similarity-based baseline model. (r_ui=2 and est=1.80)
  The output also contains "actual_k". It is the value of K in KNN that is used while training the model. The default value is 40.
  Predicted the play count for the user with user id 6958 and song_id 1671. The user has already listened the song with song_id 1671.
  The above output shows that the actual play count for this user-item pair is 2 and the predicted rating is 1.80 by the user-user-similarity based baseline model.

- Predicting play_count for a sample user with a song not-listened by the user.

```
user: 6958       item: 3232       r_ui = None   est = 1.64   {'actual_k': 40, 'was_i
mpossible': False}

Prediction(uid=6958, iid=3232, r_ui=None, est=1.6386860897998294, details={'actual_k
': 40, 'was_impossible': False}
```

- Implemented the recommendation algorithm based on optimized KNNBasic Model
- Set up parameter grid to tune hyperparameters
- Trained the best model found in a gridsearch.

Optimized model:
RMSE: 1.0521
Precision: 0.413
Recall: 0.721
F_1 score: 0.525

Observations and Insights:
We can see from above that after tuning hyperparameters, F_1 score of the tuned model has not improved much as compared to the baseline model.

- Predict the play count for the user user id 6958 and song_id 1671. The user has already listened the song with song_id 1671. output shows that the actual play count for this user-item pair is 2 and the predicted rating is 1.80 by the user-user-similarity based tuned model.

```
user: 6958       item: 1671       r_ui = 2.00   est = 1.80   {'actual_k': 40, 'was_impo
ssible': False}
                                                                                    :
Prediction(uid=6958, iid=1671, r_ui=2, est=1.8009387435128914, details={'actual_k': 40,
'was_impossible': False})
```

- Predicted the play count for a song that is not listened by the user (with user_id 6958). The user has not listened the song with song_id#3232. Predicted rating is 1.64

```
user: 6958       item: 3232       r_ui = None   est = 1.64   {'actual_k': 40, 'was_impo
ssible': False}

Prediction(uid=6958, iid=3232, r_ui=None, est=1.6386860897998294, details={'actual_k':
40, 'was_impossible': False})
```

- Along with making predictions on listened and unknown songs. We can get 5 nearest neighbors (most similar) to a certain user. *Used inner id 0.* We calculated 5 nearest neighbors (most similar) to inner id 0.
- Implemented a function. The output of the function is a **set of top_n items** recommended for the given user_id based on the given algorithm
- We predicted top 10 songs for user id 6958 using user-user similarity-based recommendation system.
- Corrected the play_counts and ranked the predicted top 10 songs.
- In the corrected rating formula, subtracted quantity 1/np.sqrt(n) to get more optimistic predictions. The reason is some songs with rating 5 and we can't have a rating more than 5 for a song.
- Applied the ranking_songs function on the final_play data.

Top 10 recommended songs for user Id# 6958 using user-user similarity based collaborative filtering:

	song_id	play_freq	predicted_ratings	corrected_ratings
5	7224	107	3.141147	3.044473
2	614	373	2.525000	2.473222
4	5653	108	2.514023	2.417798
0	352	748	2.425000	2.388436
7	6450	102	2.394927	2.295913
9	8324	96	2.373359	2.271297
1	952	482	2.275000	2.229451
6	8831	107	2.271924	2.175250
3	6448	109	2.270051	2.174268
8	4831	97	2.250465	2.148930

B. Item-Item Similarity Based Collaborative Recommendation Overview

- Applied the item-item similarity collaborative filtering model with random_state=1 and evaluated the baseline model performance.
- The KNN algorithm is used to find desired similar items
- Trained the algorithm on the trainset, and predict ratings for the test set
- Computed precision@k, recall@k, and f_1 score with k = 10
- Built the first baseline similarity-based recommendation system using the cosine similarity.
- KNNBasic is an algorithm that is also associated with the surprise package. It is used to find the desired similar items among a given set of items.
- Train the algorithm on the trainset and predict play_count for the testset.
- Computed basedline precision@k, recall@k, and f_1 score with k =30.

Based line model:

```
RMSE: 1.0394
Precision:  0.307
Recall:  0.562
F_1 score:  0.397
```

Observations and Insights:
Here the baseline model gave an F_1 score of ~0.40. We tried to improve this by using GridSearchCV by tuning different hyperparameters of this algorithm.
We have calculated RMSE to check how far the overall predicted play counts are from the actual play counts.
Intuition of Recall - We are getting a recall of 0.56, which means out of all the relevant songs, 56% are recommended.
Intuition of Precision: We are getting a precision of almost 0.30, which means out of all the recommended songs 30% are relevant.

- Predicted play_count for a sample user with a listened song.

```
user: 6958      item: 1671      r_ui = 2.00   est = 1.36   {'actual_k': 20, 'was_impossible': False}
Prediction(uid=6958, iid=1671, r_ui=2, est=1.3614157231762556, details={'actual_k': 20, 'was_impossible': False})
```

- Predicting play_count for a sample user with a song not listened by the user.

```
user: 6958      item: 3232      r_ui = None   est = 1.38   {'actual_k': 20, 'was_impossible': False}
Prediction(uid=6958, iid=3232, r_ui=None, est=1.377602711737415, details={'actual_k': 20, 'was_impossible': False})
```

Observations and Insights:

- Predicted the rating for the user with user_id 6958 and the the song with song id=1671. The user already interacted or listened to the song. The actual rating is for this user-item pair is 2, and the predicted rating is close to that.
- Then, predicted rating for the same user_id 6958 but for a song with which this user has not interacted yet (song id 3232).

- Applied grid search for enhancing model performance
- Implemented the recommendation algorithm based on optimized KNNBasic Model
- Set up parameter grid to tune hyperparameters
- Trained the best model found in a GridSearch.

Optimized model:

```
RMSE: 1.0412
Precision:  0.392
Recall:  0.654
F_1 score:  0.49
```

We can see from above that after tunning hyperparameters, F_1 score of the tuned model is much better than the baseline model. (0.50 vs 0.40)

- Predicted the play count for the user user id 6958 and song_id 1671. The user has already listened the song with song_id 1671.

```
user: 6958      item: 1671      r_ui = 2.00   est = 1.36   {'actual_k': 20, 'was_impossible': False}
Prediction(uid=6958, iid=1671, r_ui=2, est=1.3614157231762556, details={'actual_k': 20, 'was_impossible': False})
```

- Predicted the play count for a song that is not listened by the user (with user_id 6958). The user has not listened the song with song_id#3232. Predicted rating is 1.64

```
user: 6958      item: 3232      r_ui = None   est = 1.38   {'actual_k': 20, 'was_impossible': False}
Prediction(uid=6958, iid=3232, r_ui=None, est=1.377602711737415, details={'actual_k': 20, 'was_impossible': False})
```

- Calculated 10 most similar users to the user with inner id 0
- Made top 10 recommendations for user_id 6958 with item_item_similarity-based recommendation engine.
- Built the data frame for above recommendations with columns “song_id” and “predicted_play_count”
- Applied the ranking_songs function.

Top 10 recommended songs for user Id# 6958 using item-item similarity based collaborative filtering:

	song_id	play_freq	predicted_ratings	corrected_ratings
0	2842	232	1.650010	1.584357
2	7921	220	1.542349	1.474929
1	9386	221	1.533849	1.466582
3	733	206	1.526769	1.457096
5	7470	158	1.534020	1.454464
7	5417	135	1.538866	1.452800
8	4631	120	1.539257	1.447970
9	1767	112	1.540675	1.446184
4	8577	187	1.511054	1.437927
6	657	151	1.518838	1.437459

C. Model Based Collaborative Recommendation System - Matrix Factorization Overview

- Built a baseline model using SVD (SVD with matrix factorization with random state=1)
- Trained the algorithm on the training dataset
- Computed precision@k, recall@k, and f_1 score with k = 10
- Observed that the baseline model with algorithms is giving F_1 score of ~50%.
- Made prediction for user (with user_id 6958) to song (with song_id 1671), take r_ui=2
- Made prediction for user who has not listened the song (song_id 3232)
- Improved matrix factorization-based recommendation system by tuning its hyperparameters
- Built the optimized SVD model using optimal hyperparameters
- Compute precision@k, recall@k, and f_1 score with k = 10

  Observations and Insights: We can observe that after tuning hyperparameters, the model performance has not improved much.
- Using svd_algo_optimized model to recommend for userId 6958 and song_id 1671.
- Using svd_algo_optimized model to recommend for userId 6958 and song_id 3232 with unknown baseline rating.
- Observations and Insights:
- For the song that has been listened to for user id 6958, estimated prediction has not improved much after tunning.
- Getting top 5 recommendations for user_id 6958 using "svd_optimized" algorithm.
- Ranked songs based on top 5 recommendation for user 6958

Model Performance Metric:
Baseline F_1 score: 49.8%
Optimized model F_1 score: 50.2%

Top 10 recommended songs for user Id# 6958 using model based collaborative filtering (matrix factorization):

	song_id	play_freq	predicted_ratings	corrected_ratings
6	7224	107	2.601899	2.505225
5	5653	108	2.108728	2.012502
9	8324	96	2.014091	1.912029
1	614	373	1.917197	1.865419
4	9942	150	1.940115	1.858465
0	5531	618	1.895254	1.855028
7	6450	102	1.952493	1.853478
3	657	151	1.926228	1.844849
8	4831	97	1.931137	1.829602
2	4811	160	1.869229	1.790172

D. Cluster Based Recommendation System Overview

- Made baseline clustering model using CoClustering algorithm with random state=1
- Trained the algorithm on the train set
- Computed precision@k, recall@k, and F_1 score with k = 10
- Calculated RMSE to check how far the overall predicted ratings are from the actual ratings.
- Observed F_1 score of the baseline model as 0.47.
- Made prediction for user_id 6958 and song_id 1671.
- Using GridSearchCV by tunned different hyperparameters of this algorithm.
- Trained the tuned Coclustering algorithm.
- Computed precision@k, recall@k, and F_1 score with k = 10
- Using co_clustering_optimized model to recommended for userId 6958 and song_id 1671.
- Using Co_clustering based optimized model to recommend for userId 6958 and song_id 3232 with unknown baseline rating.
- Implemented the recommendation algorithm based on optimized CoClustering model
- Corrected the play count and ranking the above songs
- In the **above-corrected rating formula**, **subtracted quantity 1 / np.sqrt(n) to get more optimistic predictions**. The reason for **subtracting this quantity**, as there are some songs with ratings of 5 and **we can't have a rating more than 5 for a song**.

Model Performance Metric:
Baseline F_1 score: 47.2%
Optimized model F_1 score: 46.5.2%

Top 10 recommended songs for user Id# 6958 using cluster-based recommendation system:

Top 10 recommended songs for user Id# 6958 using cluster-based recommendation system:

	song_id	play_freq	predicted_play_count	corrected_play_count
9	7224	107	3.711503	3.614829
8	5653	108	2.903883	2.807658
5	6860	169	2.691043	2.614120
6	657	151	2.606354	2.524975
7	8483	123	2.582807	2.492640
0	8138	524	2.503481	2.459796
1	5943	423	2.436141	2.387519
3	8252	267	2.448189	2.386990
2	317	411	2.423946	2.374619
4	6378	183	2.444544	2.370622