Spotify Music Recommendation Systems

DSCI 641 Recommender Systems

Group 18 : Nadimpalli Rohit Varma(rn493), Manish Reddy Pannala(mp3787)

Introduction

Music has become one of the most consumed forms of entertainment in the world, and streaming platforms like Spotify host millions of songs across different genres, artists, and eras. With such a massive library, listeners often rely on recommender systems to help them discover music that matches their taste. In this project, we set out to design and implement a music recommendation system using a Spotify dataset. The system uses both **content-based filtering**, which compares audio features such as tempo, energy, and danceability, and **popularity-based filtering**, which suggests songs that are widely played and trending. We also extended these ideas into a **hybrid system** that balances similarity and popularity, and added **genre- and year-based recommenders** for more focused exploration. To make the system transparent, we implemented an explanation function that shows why a song was recommended, such as having a similar tempo or mood. While we faced challenges with scalability and could only work with a smaller portion of the dataset due to system capacity, the project gave us valuable hands-on experience in building, explaining, and reflecting on different types of music recommender systems.

1. Dataset Description

The dataset used in this project is a **Spotify music dataset** sourced from Kaggle. It contains metadata and audio features for over **170,000 tracks** spanning from 1921 to 2020. Each row in the dataset represents a song, with attributes grouped into three broad categories:

- **Basic Track Information:**
  - ***Track ID*:** Unique identifier for each song, ***Name***: Title of the track, ***Artists***: Artist(s) who performed the song, ***Popularity***: A score from 0 to 100 reflecting how frequently and how recently the song has been played on Spotify, ***Year***: The year the track was released.
- **Musical Composition Details:**
  - ***Tempo***: Speed of the track in beats per minute (BPM), ***Time Signature***: Number of beats per measure, ***Mode***: Whether the song is in a major (happy/bright) or minor (sad/dark) scale, ***Key***: The primary pitch of the track.
- **Audio Features (numeric descriptors of sound):**
  - ***Acousticness***: Degree to which the track is acoustic, ***Danceability***: Suitability of the track for dancing based on tempo, rhythm stability, and beat strength, ***Energy***: Intensity and activity level of the track, ***Instrumentalness***: Likelihood the track contains no vocals, ***Liveness***: Probability the track was recorded in front of a live audience, ***Loudness***: Overall volume of the track in decibels, ***Speechiness***: Presence of spoken words or rap elements, ***Valence***: Musical positivity or emotional cheerfulness of the track, This dataset is valuable for building recommendation systems because it combines:
- **Objective audio features** that capture the musical profile of each song. **Popularity scores** that reflect social listening trends. **Metadata (artist, year, genre)** that enables grouping, filtering, and evaluation.

Together, these attributes allow us to explore both **content-based** and **popularity-based** recommendation strategies, as well as hybrid approaches that combine the two.

2. Type of Recommender System We Built

In this project, we did not just build one recommender system — we explored **several different types**, each with its own style of giving suggestions:

1. **Popularity-Based Recommender**
   - This system recommends songs that are the most popular overall.
   - It is simple and works well if the goal is to suggest music that "everyone is listening to."
2. **Content-Based Recommender**
   - This system looks at the audio features of a song (like danceability, energy, tempo, valence) and finds other songs with a similar sound profile.
   - If you like one song, it tries to suggest other songs that "sound" alike.
3. **Hybrid Recommender**
   - This combines the two ideas above: it balances **how similar the songs are in sound** with **how popular they are**.
   - This way, you don't only get niche songs (content-based) or just the top hits (popularity-based), but a mix of both.
4. **Genre-Constrained Recommender**
   - This focuses only on songs from the **same genre** as the input song, and then ranks them by similarity.
   - For example, if you like a pop song, it only shows other pop songs that are close in style.
5. **Year-Constrained Recommender**
   - This filters songs based on their release year.
   - For instance, if you like a modern track but want to find something similar from the 1990s, this recommender makes that possible.

In short, our system is a **multi-paradigm recommender**, meaning it can recommend music in different ways depending on what the user is looking for — the most popular, the most similar, or constrained by genre or year.

3. Methods and Algorithms Used

To build our music recommenders, we used a mix of **data preprocessing, similarity measures, and ranking methods**. Below are the main techniques:

1. **Data Preprocessing**
   - We standardized all audio features (like tempo, energy, danceability) so they are on the same scale. Missing values were replaced with median values so the system could still use the songs without errors.
2. **Cosine Similarity**
   - For the content-based recommender, we used **cosine similarity**. This is a mathematical way of measuring how close two songs are based on their audio features. Example: If two songs have similar tempo, energy, and valence, their cosine similarity will be high.
3. **Popularity Ranking**
   - For the popularity-based recommender, we simply ranked songs by their **Spotify popularity score**. The higher the score, the more frequently and recently a song is played on Spotify.
4. **Hybrid Scoring**
   - In the hybrid recommender, we combined both similarity and popularity. We used a formula like:
     $$\text{Final Score} = \alpha \times \text{Similarity} + (1-\alpha) \times \text{Popularity}.$$
   - The parameter **α (alpha)** controls the balance between popularity and similarity. If $\alpha = 0.7$, then similarity is weighted more heavily, but popularity still plays a role.
5. **Filtering by Genre and Year**
   - For genre- and year-based recommenders, we first **filtered** the dataset to only include songs from the same genre or year, and then ranked them using similarity.
6. **Recommendation Explanations**

- We added a simple explanation module that checks which features (like tempo, energy, or danceability) were most similar between the query song and the recommended song. This makes the system more transparent — users can see *why* a certain song was recommended.

So in summary, our system is built mainly on **cosine similarity for song features** and **ranking methods for popularity**, and then combines them in different ways (pure, hybrid, filtered).

4. Interpretation and Explain the Recommended Results

The recommendations from our system can be understood based on the type of recommender used:

1. **Popularity-Based Recommendations**
   - These simply show the songs with the highest popularity scores. If a song appears here, it is because it is widely played and trending on Spotify.
2. **Content-Based Recommendations**
   - These are based on the similarity of audio features such as tempo, energy, and danceability.
   - For example, if you choose *"Shape of You"*, the system may recommend *"Cheap Thrills"* because both are upbeat, danceable tracks with similar tempo and energy.
   - **Explanation Module:** In our notebook, we implemented explanations **only for the content-based recommender**. The system highlights which features (like tempo, energy, or valence) were most similar between the input song and the recommended song. This helps users understand *why* a recommendation was made.
3. **Hybrid, Genre-, and Year-Based Recommendations**
   - These recommend songs using combinations of similarity, popularity, or filters like genre and year. However, in our current implementation we did not add explanation outputs for these recommenders.

5. How We Evaluated the System's Performance

In this project notebook, we mainly focused on providing **explanations** for recommendations rather than running formal evaluation metrics like precision@K or recall@K.

Recommendation Explanation Function

The explain_recommendation function was used to interpret and validate recommendations. It works as follows:

1. It compares the audio features of the **input song** and the **recommended song**.
2. It calculates the **cosine similarity** between the two songs' feature vectors.
3. It identifies the **top 2–3 features** (e.g., tempo, energy, danceability) that are most similar between the songs.
4. It produces a human-readable explanation such as:
   - *"We recommended this because it has similar tempo, energy, and danceability to your chosen track."*

Why This is Evaluation

While this is not a traditional metric-based evaluation (like accuracy or precision@K), it still helps evaluate the system by:

- **Checking correctness**: Are the recommended songs actually similar in the expected features?
- **Providing transparency**: Users can see why a song was recommended, which makes the system more trustworthy.
- **Validating logic**: If explanations don't make sense (e.g., recommending a slow ballad after a fast dance track), we know the system needs improvement.

6. Limitations and Future Improvements

Limitations of the Current System

1. **Memory Usage**
    - The notebook precomputes a full cosine similarity matrix for all songs in the dataset. With ~170,000 songs, this requires massive memory (over 200 GB), which makes it impossible to run on normal machines. Due to system memory and processing capacity, we were only able to test the recommender on about 2.5k rows of the dataset. This means our results are based on a smaller sample and may not fully represent the performance on the complete dataset.
    -
2. **Evaluation**
    - The only evaluation implemented is the explain_recommendation function. This provides useful insights into why recommendations are made, but it does not give a **quantitative measure** (such as precision, recall, or NDCG).
3. **Cold Start Problem**
    - For new or less popular songs, the system can only rely on audio features. Without user interaction data, recommendations for new tracks may not be as accurate.
4. **Lack of Personalization**
    - The system currently recommends songs based on similarity and popularity, not based on individual user listening history. This means all users get the same type of recommendations, rather than personalized suggestions.
5. **Limited Feature Use**
    - The model only uses Spotify's audio features and popularity. It ignores other important data like **lyrics, moods, playlists, or cultural context**.
    -

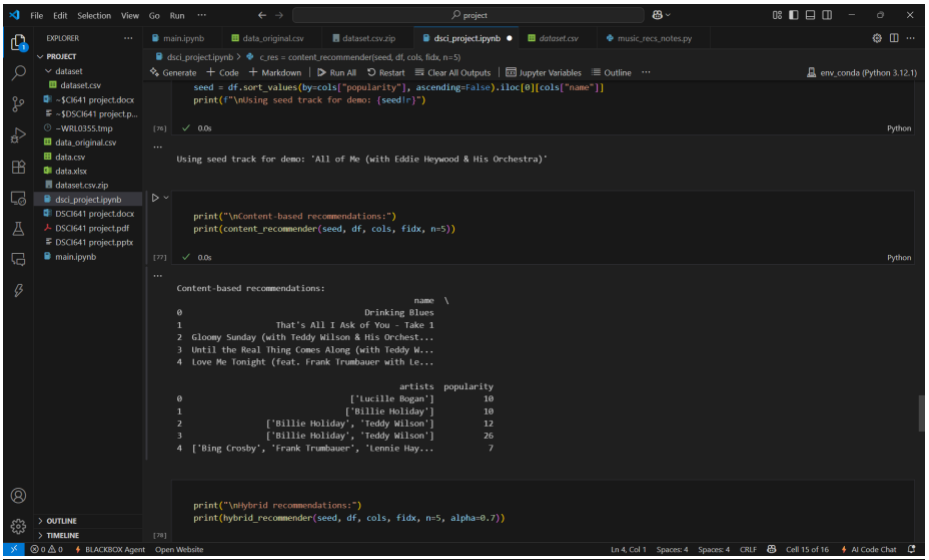
Future Improvements

1. **Efficient Similarity Search**
    - Replace the full similarity matrix with **Approximate Nearest Neighbors (ANN)** methods like FAISS or Annoy. This would make the system scalable to millions of songs.
2. **Hybrid with Collaborative Filtering**
    - Incorporate collaborative filtering (based on user listening data) alongside content-based similarity. This would personalize recommendations.
3. **Use of Lyrics and Metadata**
    - Apply NLP techniques on song lyrics to capture thematic or emotional similarity. Include metadata like album, playlist tags, or mood annotations for richer recommendations.
4. **Interactive User Feedback**
    - Build a simple web app where users can give thumbs-up or thumbs-down on recommendations. Use this feedback to fine-tune and adapt the system in real-time.

7. What we have learnt from the project?

From this project, we learned how to process a large dataset, scale audio features, and use cosine similarity to recommend songs based on sound profiles. We also explored combining similarity with popularity in a hybrid approach and added explanations to make results more transparent. We realized that popularity-based systems highlight trending songs, content-based systems capture musical similarity, and hybrids provide a balance. At the same time, we saw challenges such as memory limits when handling large datasets and the lack of personalization without user data. Overall, the project gave us practical experience and taught us that a good recommender must balance accuracy, efficiency, diversity, and user trust.

Result Screenshots:



```
    seed = df.sort_values(by=cols["popularity"], ascending=False).iloc[0][cols["name"]]
    print(f"\nUsing seed track for demo: {seed!r}")
```

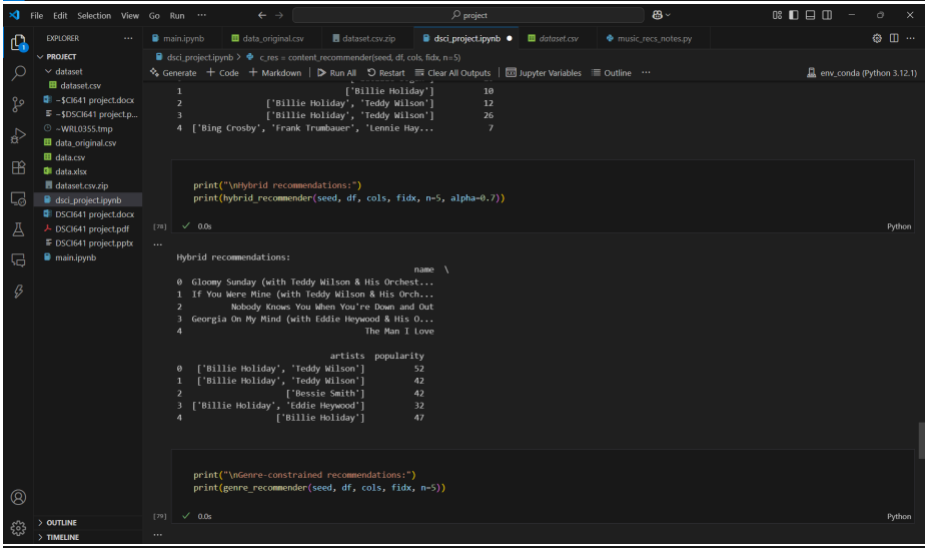
```
Using seed track for demo: 'All of Me (with Eddie Heywood & His Orchestra)'
```

```
print("\nContent-based recommendations:")
print(content_recommender(seed, df, cols, fidx, n=5))
```

```
Content-based recommendations:
                                        name  \
0                                Drinking Blues
1                  That's All I Ask of You - Take 1
2  Gloomy Sunday (with Teddy Wilson & His Orchest...
3  Until the Real Thing Comes Along (with Teddy W...
4  Love Me Tonight (feat. Frank Trumbauer with Le...

                                  artists  popularity
0                          ['Lucille Bogan']          10
1                         ['Billie Holiday']          10
2            ['Billie Holiday', 'Teddy Wilson']          12
3            ['Billie Holiday', 'Teddy Wilson']          26
4  ['Bing Crosby', 'Frank Trumbauer', 'Lennie Hay...          7
```

```
print("\nHybrid recommendations:")
print(hybrid_recommender(seed, df, cols, fidx, n=5, alpha=0.7))
```



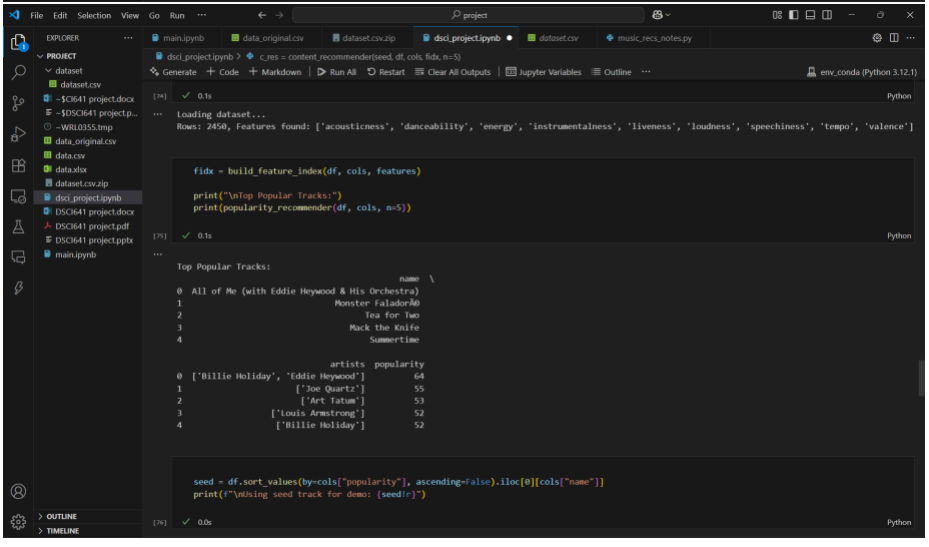
```
1                         ['Billie Holiday']          10
2            ['Billie Holiday', 'Teddy Wilson']          12
3            ['Billie Holiday', 'Teddy Wilson']          26
4  ['Bing Crosby', 'Frank Trumbauer', 'Lennie Hay...          7
```

```
print("\nHybrid recommendations:")
print(hybrid_recommender(seed, df, cols, fidx, n=5, alpha=0.7))
```

```
Hybrid recommendations:
                                        name  \
0  Gloomy Sunday (with Teddy Wilson & His Orchest...
1  If You Were Mine (with Teddy Wilson & His Orch...
2          Nobody Knows You When You're Down and Out
3  Georgia On My Mind (with Eddie Heywood & His O...
4                                The Man I Love

                                  artists  popularity
0  ['Billie Holiday', 'Teddy Wilson']          52
1  ['Billie Holiday', 'Teddy Wilson']          42
2                     ['Bessie Smith']          42
3  ['Billie Holiday', 'Eddie Heywood']          32
4                ['Billie Holiday']          47
```

```
print("\nGenre-constrained recommendations:")
print(genre_recommender(seed, df, cols, fidx, n=5))
```



```
Loading dataset...
Rows: 2450, Features found: ['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', 'valence']
```

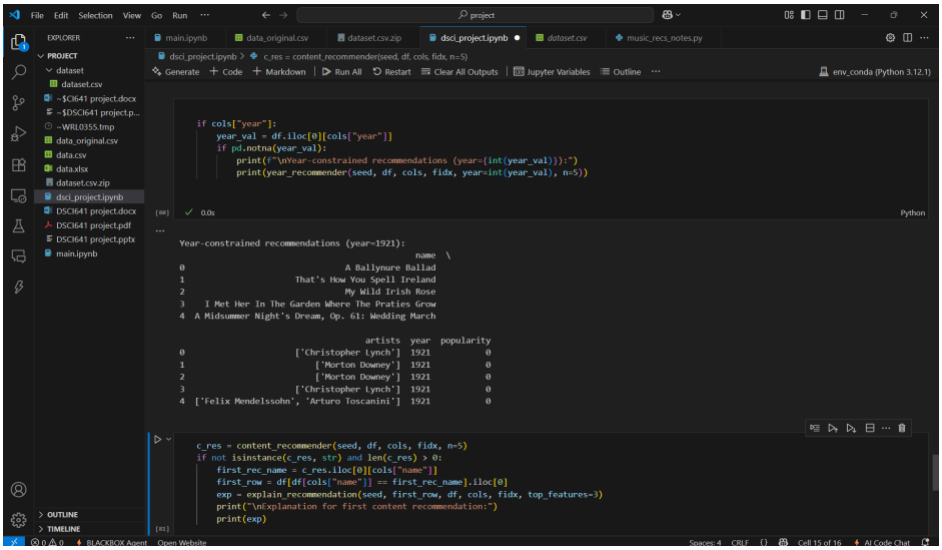
```
fidx = build_feature_index(df, cols, features)

print("\nTop Popular Tracks:")
print(popularity_recommender(df, cols, n=5))
```

```
Top Popular Tracks:
                                        name  \
0  All of Me (with Eddie Heywood & His Orchestra)
1                              Monster FaladorÃ©
2                                   Tea for Two
3                                Mack the Knife
4                                    Summertime

                                  artists  popularity
0  ['Billie Holiday', 'Eddie Heywood']          64
1                       ['Joe Quartz']          55
2                        ['Art Tatum']          53
3                  ['Louis Armstrong']          52
4                   ['Billie Holiday']          52
```

```
seed = df.sort_values(by=cols["popularity"], ascending=False).iloc[0][cols["name"]]
print(f"\nUsing seed track for demo: {seed!r}")
```



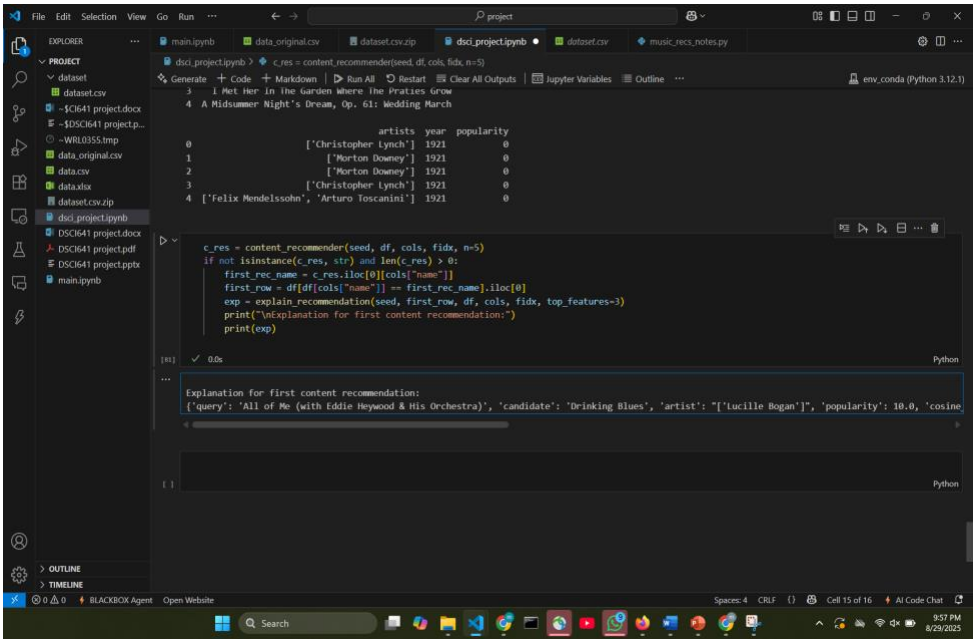
```
if cols["year"]:
    year_val = df.iloc[0][cols["year"]]
    if pd.notna(year_val):
        print(f"\nYear-constrained recommendations (year={int(year_val)}):")
        print(year_recommender(seed, df, cols, fidx, year=int(year_val), n=5))
```

```
Year-constrained recommendations (year=1921):
                                                name  \
0                                  A Ballynure Ballad
1                          That's How You Spell Ireland
2                                   My Wild Irish Rose
3    I Met Her In The Garden Where The Praties Grow
4  A Midsummer Night's Dream, Op. 61: Wedding March

                                artists  year  popularity
0                    ['Christopher Lynch']  1921           0
1                       ['Morton Downey']  1921           0
2                       ['Morton Downey']  1921           0
3                    ['Christopher Lynch']  1921           0
4  ['Felix Mendelssohn', 'Arturo Toscanini']  1921           0
```

```
c_res = content_recommender(seed, df, cols, fidx, n=5)
if not isinstance(c_res, str) and len(c_res) > 0:
    first_rec_name = c_res.iloc[0][cols["name"]]
    first_row = df[df[cols["name"]] == first_rec_name].iloc[0]
    exp = explain_recommendation(seed, first_row, df, cols, fidx, top_features=3)
    print("\nExplanation for first content recommendation:")
    print(exp)
```

Explain_recommendation:



```
Explanation for first content recommendation:
{'query': 'All of Me (with Eddie Heywood & His Orchestra)', 'candidate': 'Drinking Blues', 'artist': "['Lucille Bogan']", 'popularity': 10.0, 'cosine
```

Explanation for first content recommendation: {'query': 'All of Me (with Eddie Heywood & His Orchestra)', 'candidate': 'Drinking Blues', 'artist': "['Lucille Bogan']", 'popularity': 10.0, 'cosine_similarity': 0.9442, 'key_features': ['instrumentalness', 'speechiness', 'energy'], 'why': "Because it has similar instrumentalness, similar speechiness, similar energy and matches your track's audio profile."}

Thankyou

Group 18 : Nadimpalli Rohit Varma(rn493), Manish Reddy Pannala(mp3787)