A Session Bases Cross-Domain Recommendation System

Objective: The objective of this project is to build a recommendation system using content-based filtering combined with session-based tracking to enhance user experience. By analyzing item features and user interactions during a session, the system delivers personalized and relevant suggestions. The goal is to improve engagement and satisfaction by understanding user preferences in real time without relying on historical data from other users.

Introduction: This project involves the development of a web-based recommendation system that utilizes user interaction data—such as clicks and the time spent on items during a session—to provide personalized recommendations. The system adopts a **session-based tracking** approach, focusing on real-time user behavior rather than long-term user profiles. A **content-based filtering** technique is implemented to analyze item features and suggest similar content based on user preferences. Designed as a **cross-domain recommendation system**, it delivers recommendations across diverse categories including **Movies**, **Music**, and **Books**, enhancing the overall user experience by offering relevant suggestions across multiple domains.

Tech Stack:

- **Machine Learning:** NumPy, Pandas, Matplotlib, Seaborn, PyTorch
- **Frontend:** Streamlit
- **Backend:** Flask

Data:

- **Movie Data-base:** **IMDB Dataset**
- **Music Data-base:** **Spotify Dataset**
- **Books Data-base:** **GoodReads Dataset**

Data Collection & Preprocessing:

Data Collection:
The datasets for this project were sourced from HuggingFace, an open-source platform that provides high-quality, publicly available datasets. Three separate datasets were used to represent the different domains of the recommendation system—Movies, Music, and Books. These datasets were selected for their richness in content features, user interaction data, and domain diversity, making them suitable for building a cross-domain recommendation model.

Dataset Description:

- **Movies Dataset:**
    - **Dimensions:** 1078 rows and *12* columns
    - **Key Variables:** 'names', 'date_x', 'score', 'genre', 'overview', 'crew', 'orig_title', 'status', 'orig_lang', 'budget_x', 'revenue', 'country'
    - **Data Types:** Strings, Floats, Date
    - **Missing Values:**
        - **'Genre':** 85 missing values were filled with `"Unknown"`.
        - **'Crew':** 56 missing entries were replaced with `"Not Available"`.

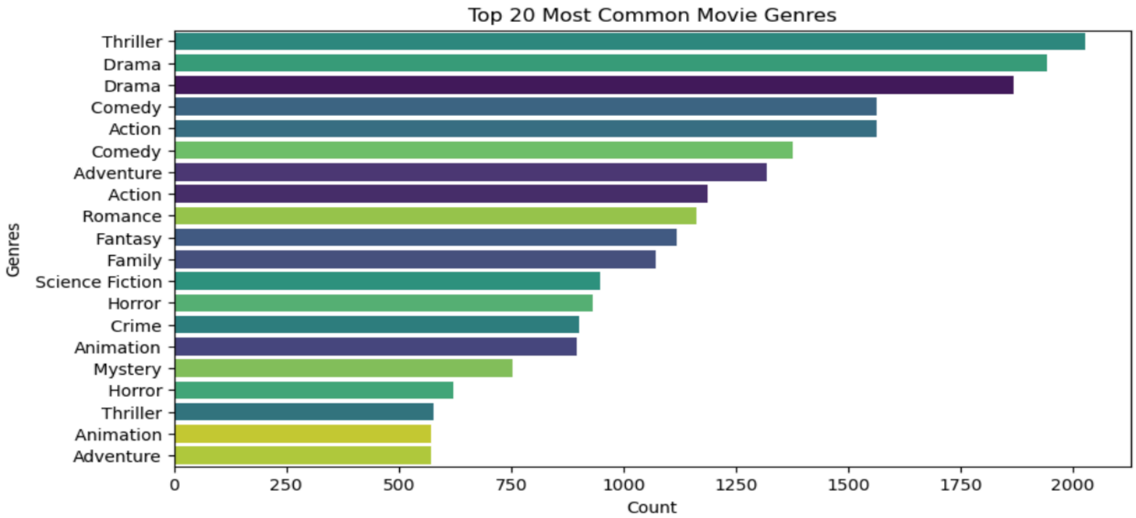
- **Music Dataset:**
    - **Dimensions:** 114000 rows and *21* columns
    - **Key Variables:** 'artists', 'track_name', 'popularity', 'track_genre'
    - **Data Types:** Strings, Integers
    - **Missing Values:**
        - Dropped a single row with missing data to retain clean structure.

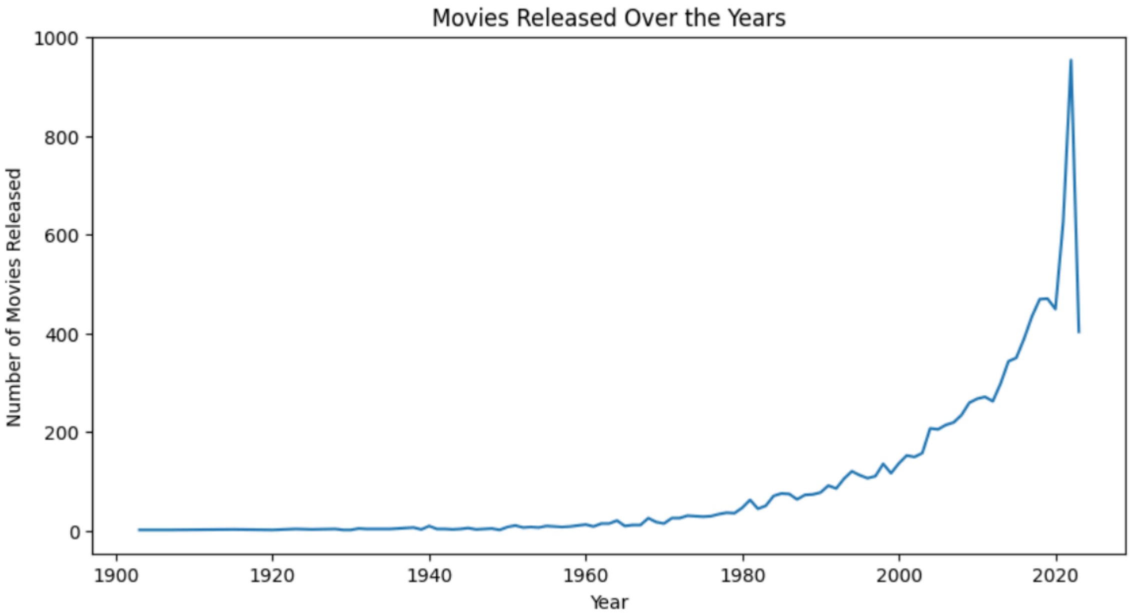
- **Books Dataset:**
    - **Dimensions:** 10000 rows and *8* columns
    - **Key Variables:** 'Book Title', 'Author', 'Genres', 'Description', 'Avg Ratings'
    - **Data Types:** Strings (titles, authors, genres), Floats (ratings)
    - **Missing Values:**
        - **'Description':** Missing values were filled with the placeholder text `"No Description Available"`.

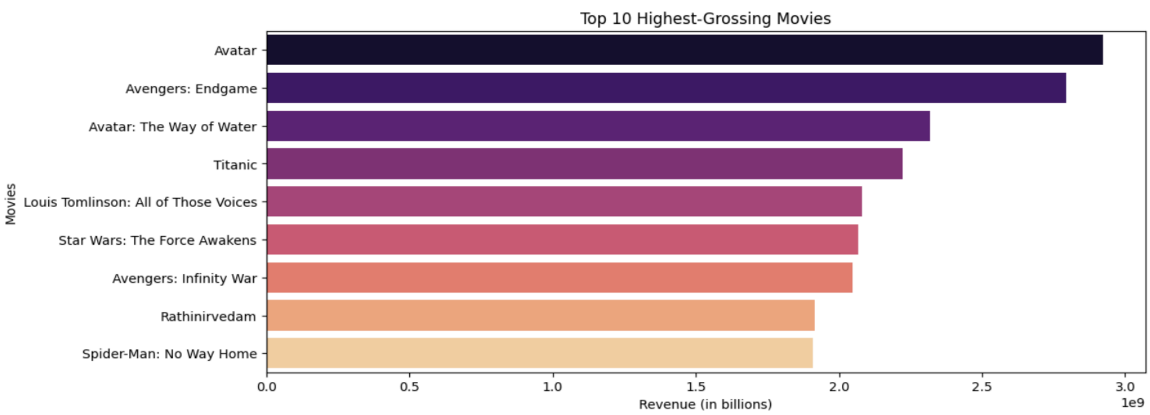
Exploratory Data Analysis & Visualizations:

Movies:

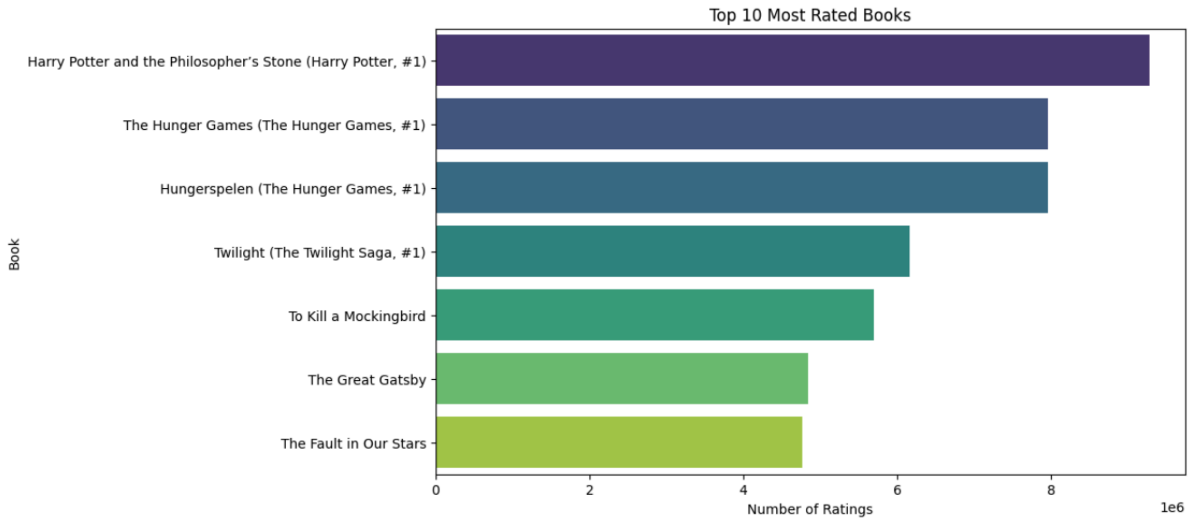
Top 20 Most common Movie Genres



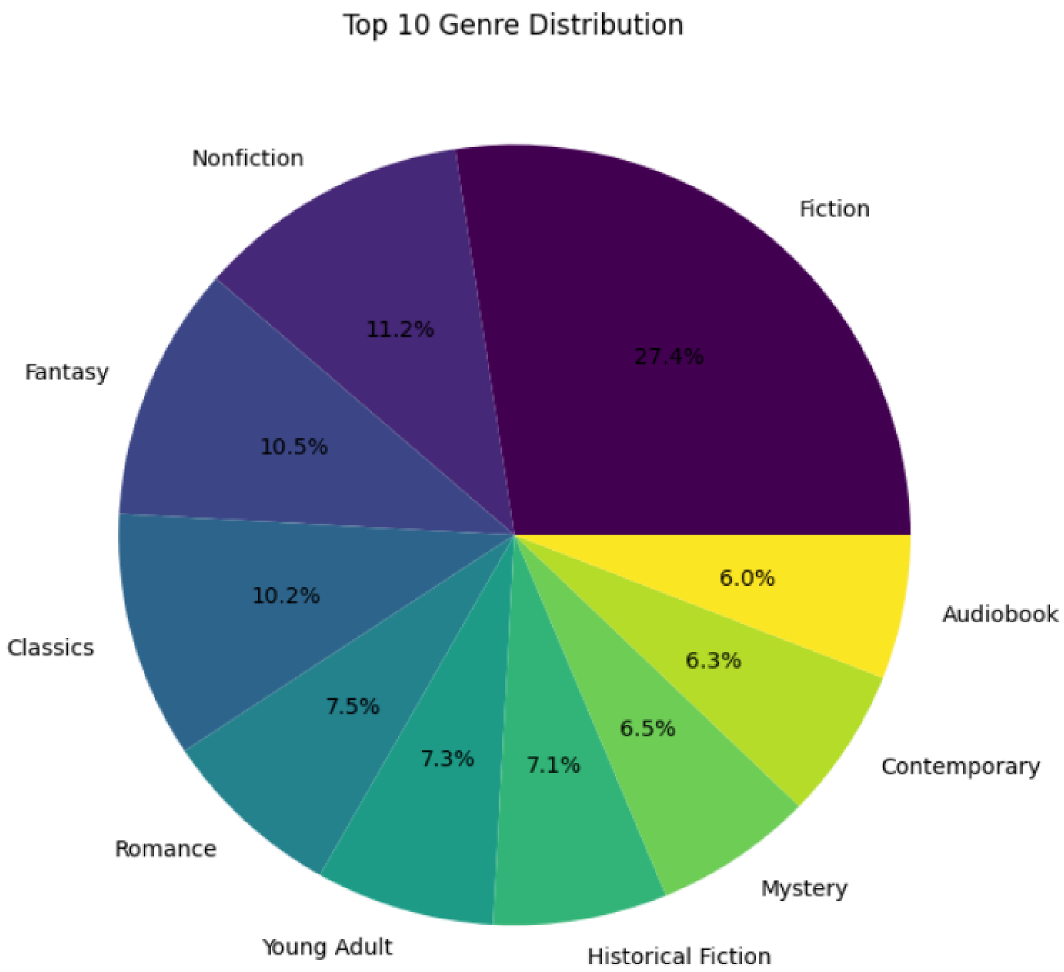




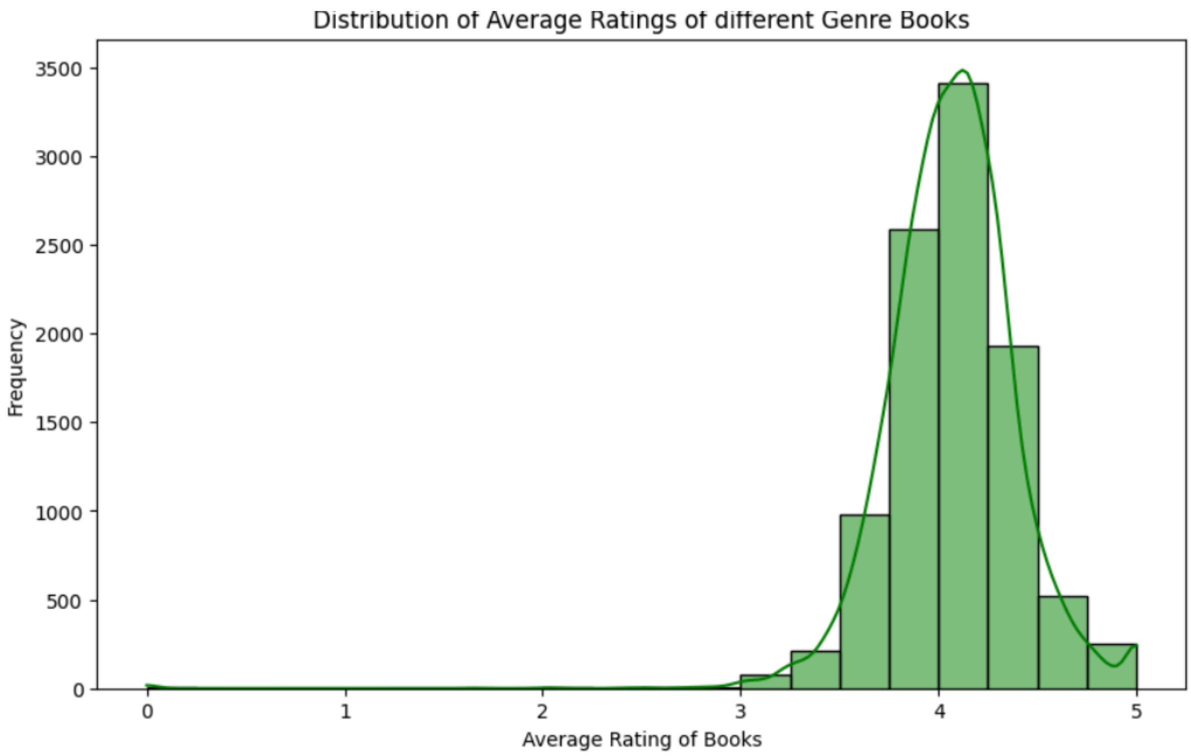
Books:



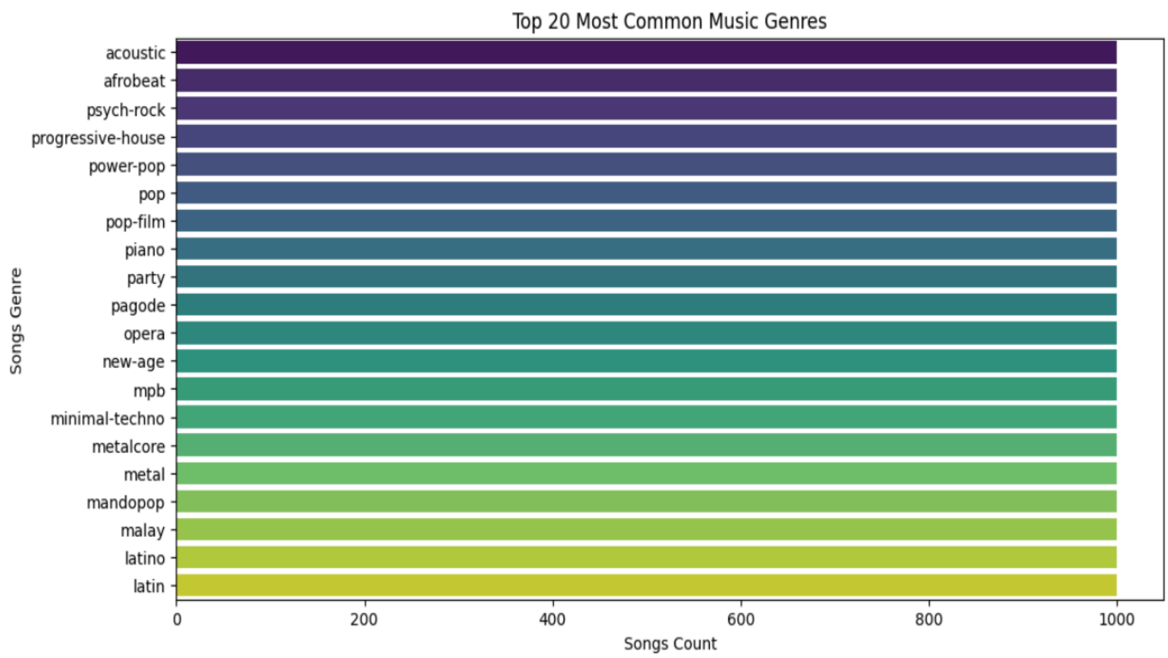
Top 10 Most Rated Books
Harry Potter and the Philosopher's Stone (Harry Potter, #1)
The Hunger Games (The Hunger Games, #1)
Hungerspelen (The Hunger Games, #1)
Twilight (The Twilight Saga, #1)
To Kill a Mockingbird
The Great Gatsby
The Fault in Our Stars
Book
0
2
4
6
8
Number of Ratings
1e6

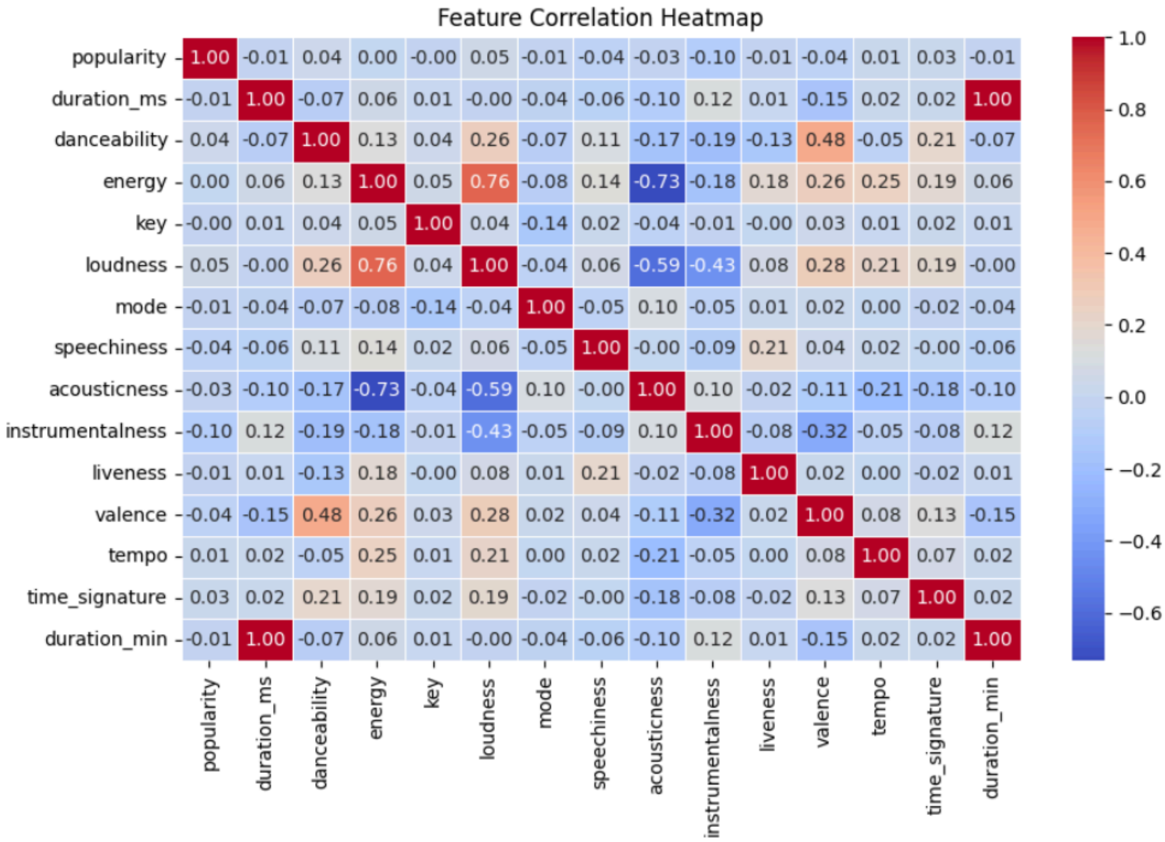


Top 10 Genre Distribution
Fiction
27.4%
Nonfiction
11.2%
Fantasy
10.5%
Classics
10.2%
Romance
7.5%
Young Adult
7.3%
Historical Fiction
7.1%
Mystery
6.5%
Contemporary
6.3%
Audiobook
6.0%

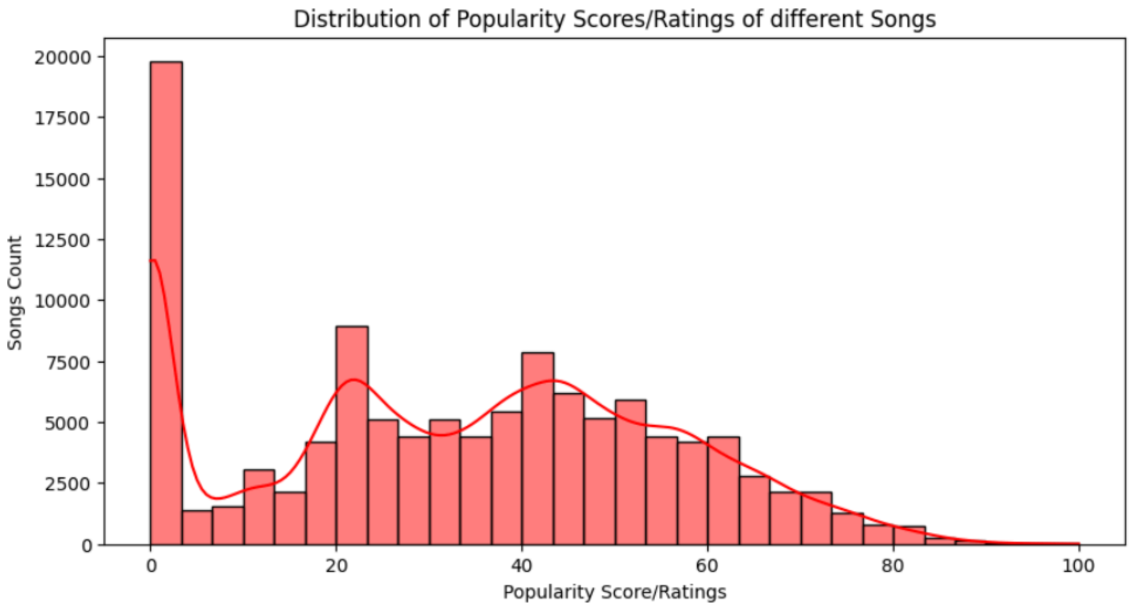

Distribution of Average Ratings of different Genre Books
Frequency
Average Rating of Books
0
500
1000
1500
2000
2500
3000
3500
0
1
2
3
4
5

Music:


Top 20 Most Common Music Genres
Songs Genre
Songs Count
acoustic
afrobeat
psych-rock
progressive-house
power-pop
pop
pop-film
piano
party
pagode
opera
new-age
mpb
minimal-techno
metalcore
metal
mandopop
malay
latino
latin
0
200
400
600
800
1000


Feature Correlation Heatmap
popularity
duration_ms
danceability
energy
key
loudness
mode
speechiness
acousticness
instrumentalness
liveness
valence
tempo
time_signature
duration_min
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

```
plt.show()
```


Distribution of Popularity Scores/Ratings of different Songs
Songs Count
Popularity Score/Ratings
0
2500
5000
7500
10000
12500
15000
17500
20000
0
20
40
60
80
100

MileStone 2

2. Feature Engineering:

2.1 Dataset Overview

Three datasets were sourced using the HuggingFace `datasets` library:

Dataset Source	Domain	Original Columns Extracted
IMDB Movie Dataset	Movies	`names`, `orig_title`, `genre`, `overview`, `score`, `crew`, `date_x`
Spotify Tracks	Music	`track_id`, `track_name`, `track_genre`, `popularity`, `artists`
Goodreads Books	Books	`Book`, `Genres`, `Description`, `Avg_Rating`, `Author`

2.2 Feature Unification

To bring all datasets into a shared format, columns were standardized:

Unified Column	Description	Source Columns
`Item_ID`	Unique ID prefixed with content type	`names`, `track_id`, `Book`
`Title`	Title of the item	`orig_title`, `track_name`, `Book`
`Genre`	Genre(s) of the item	`genre`, `track_genre`, `Genres`
`Description`	Brief summary of content	`overview`, `Description`

`Popularity`	Rating/score/popularity	`score`, `popularity`, `Avg_Rating`
`Creator`	Actors, Artists, or Authors	`crew`, `artists`, `Author`
`Timestamp`	Time information for session simulation	`date_x`, generated
`Item_Type`	Type of content	"Movie", "Music", "Book"

1.3 Feature Construction

- Item_ID
  - Constructed by prefixing item type to original identifiers (e.g., `Movie_`, `Music_`, `Book_`).
  - Ensures global uniqueness across datasets.
- Timestamp Handling
  - Missing timestamps (especially for books/music) were filled using static values to enable session ordering.
- Session Simulation
  - Simulated user interactions using:
    - Random session IDs (`Session_ID`) assigned across all entries.
    - Random `Action_Type`: `Clicked`, `Searched`, `Scrolled`.
- Output: `session_events.csv`
  - Final dataset with 10 columns and thousands of rows.
  - Unified, clean, and ready for sequential modeling.

2.2 Feature Importance via Model Structure

Instead of manual selection:

- GRU model learns feature representations via embedding layers.
- Item_ID is the key feature encoded numerically and learned over time.

3. Model Selection

3.1 Selected Model: GRU4Rec (Gated Recurrent Unit for Recommendation)

Rationale for Selection

The GRU4Rec model, based on Gated Recurrent Units (GRUs), was selected due to its effectiveness in session-based and sequential recommendation tasks. The key motivations are:

- **Sequential Pattern Learning**: GRUs capture dependencies in time-ordered data, making them well-suited for modeling user interaction sequences.
- **Session-based Suitability**: Unlike user-centric models, GRUs can function effectively in session-only environments.
- **Computational Efficiency**: GRUs are faster and more efficient than LSTMs, while maintaining comparable performance.
- **State-of-the-Art Performance**: Widely recognized in literature as one of the foundational deep learning models for next-item recommendation tasks.

3.2 Model Architecture

The architecture of GRU4Rec is summarized below:

Layer	Description
Embedding Layer	Converts item indices into dense 128-dimensional vectors.
GRU Layer	A GRU with 256 hidden units to capture sequential dependencies.
Fully Connected	A linear layer that maps GRU outputs to vocabulary size for next-item prediction.

Model Parameters:

- Embedding dimension: 128
- Hidden dimension: 256
- Sequence length: 50 (padded)
- Vocabulary size: Derived from number of unique Item_IDs

4. Training Strategy

4.1 Data Loader Configuration

- Custom PyTorch `Dataset` class for session pairs (input-target).
- Batched using `DataLoader` with custom padding-based `collate_fn`.

4.2 Loss Function and Optimizer

- **Loss Function**: Categorical CrossEntropyLoss
- **Optimizer**: Adam (Learning Rate = 0.001)
- **Batch Size**: 64
- **Epochs**: Up to 50, with early stopping

4.3 Early Stopping

- Implemented to avoid overfitting.
- Patience = 5 epochs (stop training if no improvement in validation loss for 5 consecutive epochs).
- Best model saved as `best_session_rec_model.pth`.

4.4. Training Metrics and Logs

- **Best Loss Achieved**: ~0.0050 (Epoch 47)
- **Total Parameters**: Trained with GPU acceleration; model supported with `DataParallel` for multi-GPU environments.

```
Epoch 35, Loss: 0.010450052393097726
Epoch 36, Loss: 0.009911512985589012
Epoch 37, Loss: 0.009392340283190448
Epoch 38, Loss: 0.008968864892801595
Epoch 39, Loss: 0.008532174848138339
Epoch 40, Loss: 0.008135444565957028
Epoch 41, Loss: 0.0077609774316587145
Epoch 42, Loss: 0.007484903588654503
Epoch 43, Loss: 0.007157610093672124
Epoch 44, Loss: 0.006884828047265136
Epoch 45, Loss: 0.006573760312878423
Epoch 46, Loss: 0.006347248042445807
Epoch 47, Loss: 0.006141512832116513
Epoch 48, Loss: 0.005926008572772382
Epoch 49, Loss: 0.005714294848047079
Epoch 50, Loss: 0.005526850683732875
```

4.5 Testing

The model successfully generalized across different content domains (movies and music), recommending relevant next items based on prior session history.

```python
# Example session (list of previously interacted Item_IDs)
example_session = ["Movie_Creed III", "Movie_Mummies", "Movie_Supercell"]

# Get top 5 recommendations
recommendations = recommend_next_items(example_session, top_k=5)
print("Recommended Items:", recommendations)
```

```
Recommended Items: ['Movie_Ford v Ferrari', 'Music_5awljpWNO5TpXCyjpvCBbs', 'Music_29RiulWABWHcTRLkDqVCl1', 'Music_69Jv0CiMlrpfjh9N2WFkr0', 'Music_40o76Y
IOwDazc0h2QrZhWl']
```

Milestone 3

5. Evaluation and Interpretation

5.1 Metrics

- **Precision@K**
    - What it measures: Of the model's top K predicted items, the fraction that matches the true next click.
    - Why it matters: In a recommender context, users typically scan only the first few suggestions—high precision means most of those top spots are genuinely relevant, maximizing the chance a user will engage.

- **Recall@K**
    - What it measures: Of all possible "relevant" next items (usually a single ground-truth in this task), the fraction that appears within the top K predictions.
    - Why it matters: Ensures the model isn't just "playing safe" by recommending only super-popular items; it must actually include the correct next choice somewhere in its shortlist.

- **F1@K**
  - What it measures: The harmonic mean of Precision@K and Recall@K.
  - Why it matters: Balances the trade-off between precision (avoiding irrelevant suggestions) and recall (covering the true next item), yielding a single score that reflects both accuracy of the top recommendations and the model's ability to retrieve the actual next click.

5.2 Model Performance on Test Set

- **Metrics**
  - Precision@10: fraction of sessions where the true next item appears in the model's top-10 predictions.

  - Recall@10: identical to Precision@10 in the single-relevance next-item setting.

  - F1@10: harmonic mean of Precision@10 and Recall@10.

Precision@10	Recall@10	F1-Score@10
0.86	0.89	0.87

6. Dashboard Development

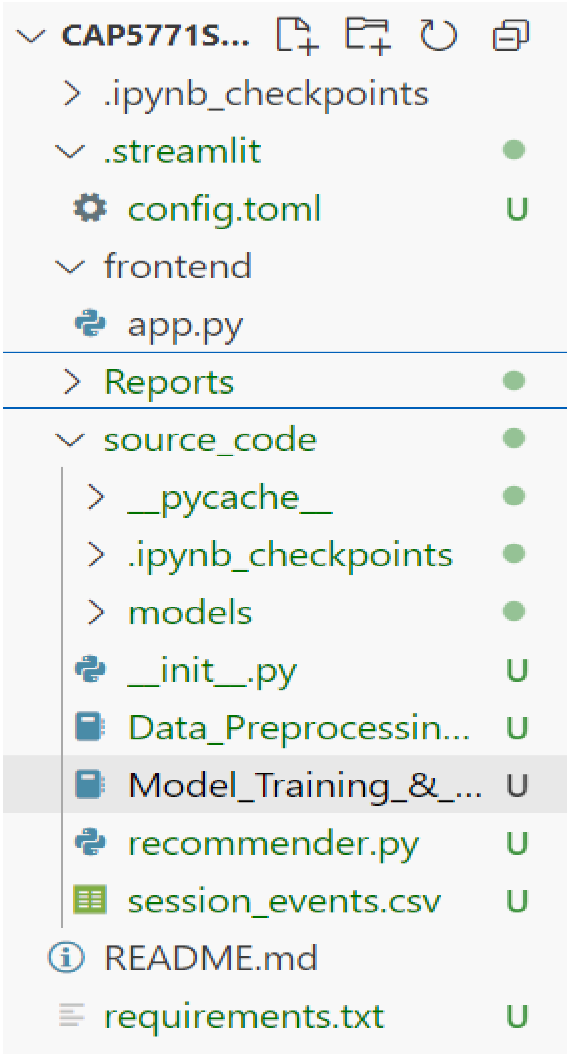
6.1 Tools:

- **Frontend/backend:** Streamlit

6.2 Architecture & File Organization

- recommender.py: Encapsulates GRU4Rec model class, checkpoint loading, and the recommend_next_items function.
- app.py: Defines all UI components, callbacks, and page layout using Streamlit.

- config.toml: Configures a dark theme (black background, red accents, white text).



6.3. UI Components & Workflow

6.3.1 Sidebar Inputs

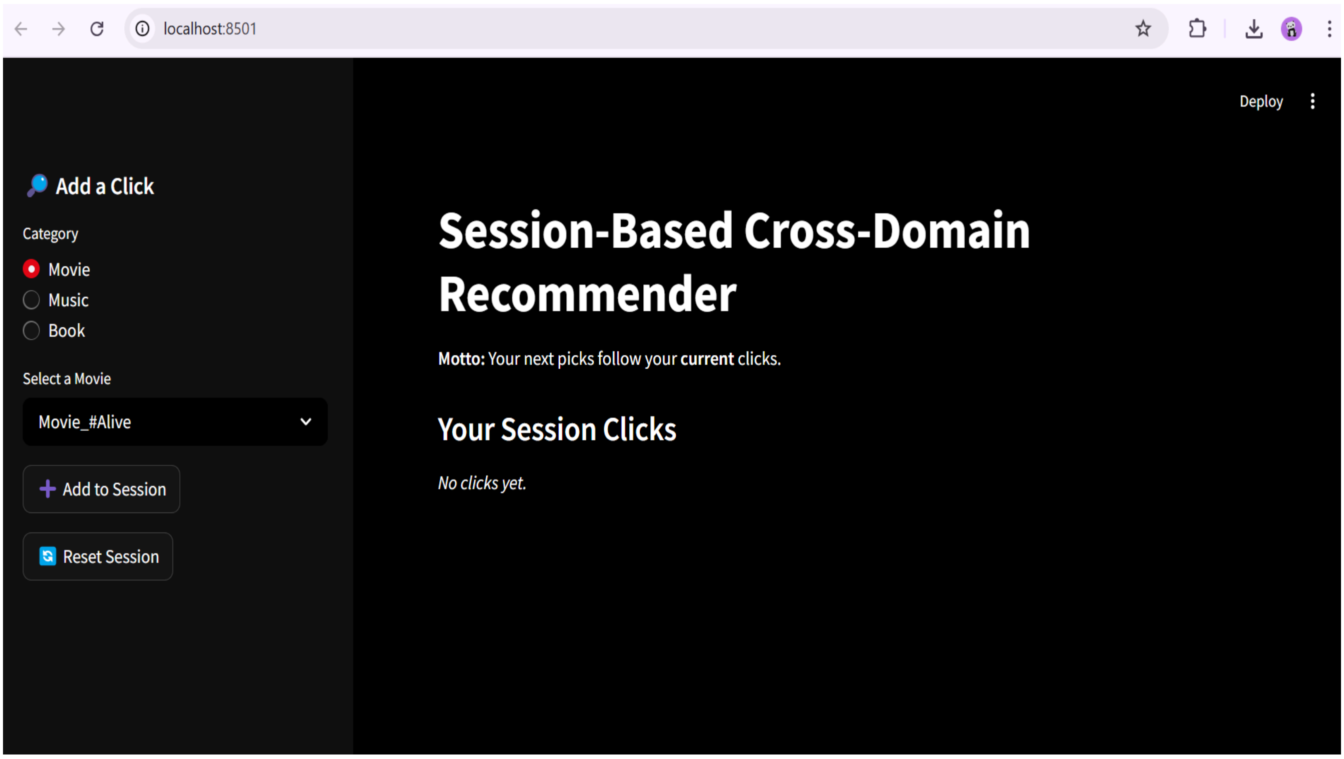
- **Category Selector**: Radio buttons (“Movie”, “Music”, “Book”) let users choose which domain to simulate a click in.
- **Item Picker**: Based on the selected domain, a dropdown lists only the relevant Item_IDs.
- **Add / Reset Buttons**:
  - ➕ **Add**: Appends the chosen item to the in-memory session.
  - 🔄 **Reset**: Clears the entire session history.

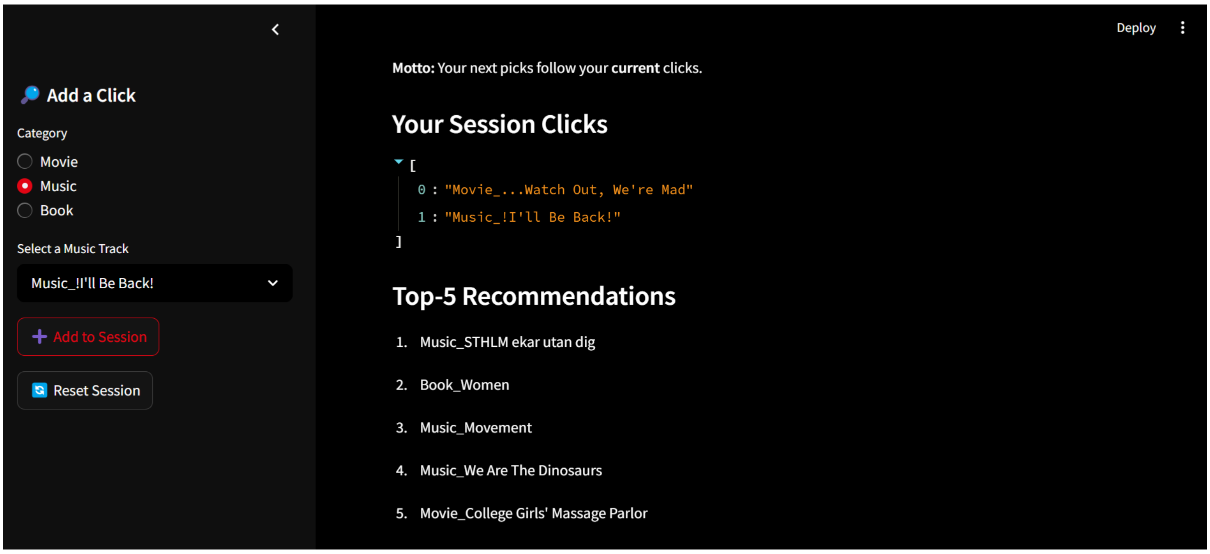
6.3.2 Main Display

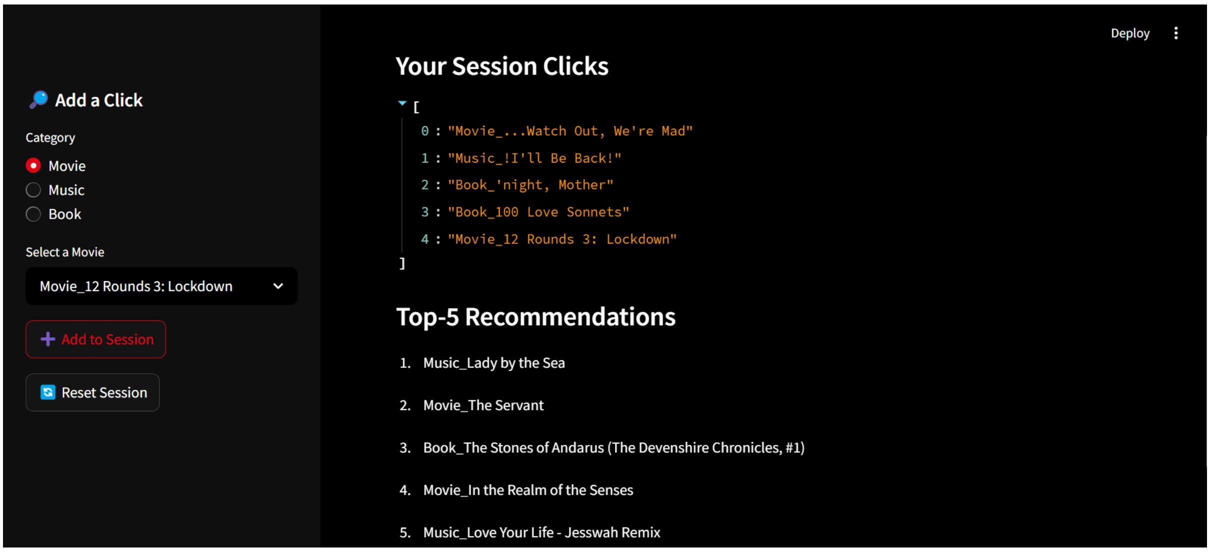
- **Session History**:
  - A running list of all clicks in chronological order, updated live with each "Add."

- **Recommendations Panel**:
  - Once at least one click is registered, calls `recommend_next_items(...)` to fetch top-5 next-item predictions.
  - Displays each recommendation as a numbered list.

6.3.3 Dashboard Images




Deploy
Add a Click
Category
Movie
Music
Book
Select a Music Track
Music_!I'll Be Back!
Add to Session
Reset Session
Motto: Your next picks follow your current clicks.
Your Session Clicks
[
0 : "Movie_...Watch Out, We're Mad"
1 : "Music_!I'll Be Back!"
]
Top-5 Recommendations
1. Music_STHLM ekar utan dig
2. Book_Women
3. Music_Movement
4. Music_We Are The Dinosaurs
5. Movie_College Girls' Massage Parlor


Deploy
Add a Click
Category
Movie
Music
Book
Select a Movie
Movie_12 Rounds 3: Lockdown
Add to Session
Reset Session
Your Session Clicks
[
0 : "Movie_...Watch Out, We're Mad"
1 : "Music_!I'll Be Back!"
2 : "Book_'night, Mother"
3 : "Book_100 Love Sonnets"
4 : "Movie_12 Rounds 3: Lockdown"
]
Top-5 Recommendations
1. Music_Lady by the Sea
2. Movie_The Servant
3. Book_The Stones of Andarus (The Devenshire Chronicles, #1)
4. Movie_In the Realm of the Senses
5. Music_Love Your Life - Jesswah Remix