Delivering Personalized Movie Recommendations with an AI-Driven Matchmaking System

Student Name: K . BUVANESH

Register Number: 412323243708

Institution: SRI RAMANUJAR ENGINEERNG COLLEGE

Department: ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Date of Submission:10/05/2025

Github Repository Link:

---

1. Problem Statement

In the modern era of abundant entertainment options, users often struggle to identify movies that align with their unique preferences. With the vast and ever-growing catalog of films across various genres, languages, and platforms, users face the challenge of finding relevant content that resonates with their tastes. This difficulty often leads to decision fatigue, wasted time, and user dissatisfaction with their viewing experience.

The problem being solved is delivering personalized movie recommendations to users by leveraging an AI-driven matchmaking system. This system aims to bridge the gap between users and movies by understanding individual preferences and providing tailored suggestions that meet their unique tastes and viewing habits.

Refinement Based on Dataset Understanding Based on additional understanding of the dataset:

1. **Dataset Insights:**
   - The dataset contains user behavior data (e.g., watch history, ratings, likes/dislikes).
   - It includes metadata about movies (genres, ratings, cast, directors, release year, etc.).
   - There may also be contextual data (e.g., user demographics, viewing time).
2. **Refinements:**
   - The problem focuses on leveraging both explicit feedback (e.g., user ratings) and implicit feedback (e.g., watch time, skipped content) to create a robust recommendation system.
   - The system must handle diverse user preferences and account for cold-start problems (e.g., new users or movies with limited data).

Problem Type

This is primarily a classification problem, where the system predicts whether a user will like or dislike a movie. It can also incorporate aspects of:

- Regression: Predicting a user's rating for a movie on a numerical scale.
- Clustering: Grouping users or movies into segments based on similarities, to enhance recommendations.
- Ranking: Ordering movies based on relevance to the user.

Why Solving This Problem Matters

Impact

- Enhanced User Experience: By reducing decision fatigue, users can quickly find movies they enjoy, leading to higher satisfaction.
- Increased Engagement: Personalized recommendations encourage users to spend more time on the platform, improving retention.
- Revenue Growth: For streaming services, better recommendations can lead to higher subscriptions and reduced churn.
- Content Discovery: Helps users explore lesser-known or niche films that match their preferences, broadening their horizons.

Relevance

- The solution taps into the growing demand for personalization in the entertainment industry.

- It aligns with advancements in machine learning, AI, and big data analytics.

Application Area

- The system is applicable to streaming platforms (e.g., Netflix, Amazon Prime, Disney+), movie review platforms (e.g., IMDb), and even theaters or on-demand services By solving this problem, we can create a more engaging and personalized entertainment ecosystem that caters to the diverse tastes of users.

2. Project Objectives

1. User Profiling and Preference Modeling

- Develop a comprehensive user preference model by analyzing explicit feedback (e.g., ratings, likes/dislikes) and implicit feedback (e.g., watch history, viewing duration, skipped content).
- Incorporate contextual factors like user demographics, device type, and time of interaction for enhanced personalization.

2.Movie Metadata Analysis

- Leverage metadata (e.g., genres, cast, director, release year, language, etc.) to establish meaningful relationships between movies.
- Identify key features that influence user preferences and integrate them into the recommendation algorithm.

3.Recommendation Algorithm Development

- Build and optimize a recommendation algorithm that uses collaborative filtering, content-based filtering, and hybrid approaches:
- Collaborative Filtering: Suggest movies based on the preferences of similar users.
- Content-Based Filtering: Recommend movies with attributes similar to those a user has liked.
- Hybrid Model: Combine the strengths of both approaches to address limitations like cold-start problems.

4. Cold-Start Problem Mitigation

- Design strategies to recommend movies for new users (limited interaction history) and newly added movies (limited metadata).
- Use clustering and demographic-based recommendations as fallback mechanisms.

5. Real-Time Recommendation Engine

- Ensure the system can deliver recommendations in real-time with minimal latency.
- Optimize for scalability to handle large datasets and concurrent user interactions.

6. Evaluation Metrics and Continuous Improvement

- Define and track key performance metrics such as precision, recall, F1-score, and Mean Average Precision (MAP) to evaluate recommendation quality.

- Implement A/B testing and user feedback loops for continuous improvement of the recommendation engine.

7. Integration and Deployment

- Integrate the recommendation engine into a user-facing platform (e.g., streaming service, movie app).
- Deploy the system on a cloud-based infrastructure to ensure high availability and scalability.

8. Ethical Considerations

- Ensure the recommendation system avoids reinforcing biases (e.g., gender, ethnicity, or language preferences).
- Implement safeguards to protect user privacy and secure sensitive data.

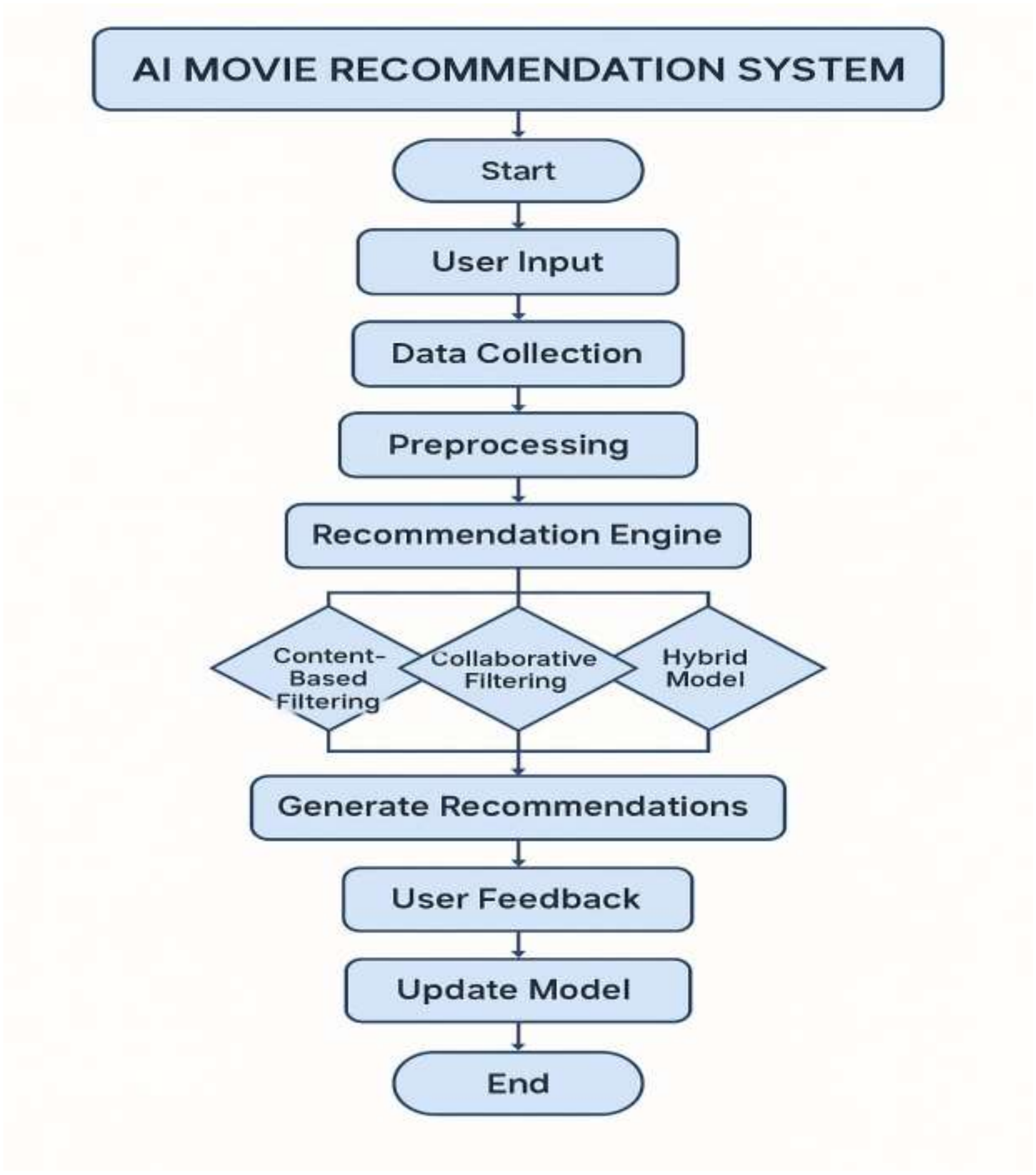
Expected Outcomes

1. A personalized movie recommendation system capable of delivering high-quality suggestions tailored to individual user preferences.

2. Improved user engagement, satisfaction, and retention on the platform.

3. Insights into user behavior and trends for content creators and platform managers.

3. Flowchart of the Project Workflow



4. Data Description

Dataset Name and Origin

The dataset for delivering personalized movie recommendations with an AI-driven matchmaking system can be sourced from publicly available platforms and proprietary data sources, such as:

MovieLens: A popular open dataset for movie recommendations (available on Kaggle and GroupLens).
IMDB Open Dataset: Metadata about movies, including ratings, genres, and cast.
The Movie Database (TMDb) API: Provides extensive movie metadata and user ratings.
Streaming Service Logs (if available): User interaction data from streaming platforms (proprietary).

Type of Data

1. **Structured Data:**
- User behavior data: Watch history, ratings, skip events, etc.
- Movie metadata: Genres, release year, cast, directors, runtime, etc.
2. **Unstructured Data:**
   - User reviews or comments (text data, optional).
3. **Temporal Data:**
   - Viewing timestamps to capture trends over time.

Dataset Characteristics

1. Number of Records:

- User data: Hundreds of thousands to millions of user profiles.
- Movie data: Thousands to tens of thousands of movies.

2. Features:

- User Features: User ID, demographics (age, gender, location), preferences (e.g., liked genres).
- Movie Features: Movie ID, title, genres, cast, directors, release year, rating, popularity score.

- Interaction Features: User-movie interactions such as ratings, watch duration, skip history, and timestamp.

3. Static or Dynamic:

- The dataset is dynamic as it evolves with new users, movies, and interactions.

Target Variable (Supervised Learning)
For Classification:

- Binary target: Whether a user likes or dislikes a movie.

For Regression:

- Numerical target: Predicting the user's rating for a movie (e.g., 1-5 scale).

For Clustering:

- No specific target; grouping users or movies based on similarity.

This dataset enables the creation of a robust, dynamic, and scalable recommendation system that adapts to user preferences and trends over time.

5. Data Preprocessing

1. Handle Missing Values

Missing values in the dataset can lead to inaccurate predictions and errors in the recommendation system. The following strategies are applied:

User Data:

- Missing demographic details (e.g., age, gender) are imputed using the median (for numerical fields) or the most frequent value (for categorical fields).
- If critical user interaction data is missing (e.g., no watch history or ratings), the user is flagged as a new user to handle through cold-start strategies.

Movie Data:

- Missing metadata (e.g., missing genre, director) is filled with "Unknown" or imputed based on similar movies.

Interaction Data:

- Missing ratings are left as null but handled during modeling as implicit feedback (e.g., watch duration).

2. Remove Duplicate Records

Duplicate records can skew the recommendation system by overrepresenting certain users, movies, or interactions. The following steps are taken:

User Data:

- Remove duplicate user profiles based on unique user IDs.

Movie Data:

- Deduplicate movies based on unique movie IDs or titles.

Interaction Data:

- Remove duplicate user-movie interactions (e.g., multiple ratings for the same movie by the same user). Retain the most recent interaction or calculate an average rating.

3. Normalize Features

To ensure all features have comparable scales:

Numerical Features:

- Normalize numerical fields (e.g., popularity score, runtime) using Min-Max scaling or Z-score normalization.

Categorical Features:

- Encode categorical fields (e.g., genres, directors) using one-hot encoding or label encoding.

4. Transform Temporal Data

- Temporal data (e.g., viewing timestamps, release years) is processed to extract meaningful features:

Viewing Timestamps:

- Extract features like time of day, day of the week, or season to identify viewing patterns.

Release Years:

- Convert to "movie age" relative to the current year to capture recency trends.

5. Feature Engineering

Additional features are engineered to enhance the recommendation model:

User Preferences:

- Aggregate user watch history to identify preferred genres, actors, or directors.

Movie Popularity:

- Calculate popularity scores using metrics like average rating, number of ratings, or total views.

6. Filter Outliers

Outliers in the dataset can negatively impact model performance:

User Data:

- Remove users with extremely low interaction data (e.g., users who rated less than 5 movies).

Movie Data:

- Exclude movies with very few interactions (e.g., less than 10 ratings) to avoid noise.

7. Split Dataset for Modeling

The dataset is split into training, validation, and testing sets:

- Training Set: Used to train the recommendation model.
- Validation Set: Used for hyperparameter tuning.
- Testing Set: Used to evaluate the final model performance.

Summary of Preprocessing Steps

1. Handled missing values through imputation or marking as unknown/new.

2. Removed duplicate records for users, movies, and interactions.

3. Normalized numerical features and encoded categorical features.

4. Extracted meaningful features from temporal data.

5. Engineered features to capture user preferences and movie popularity.

6. Filtered outliers to maintain data quality.

7. Prepared the dataset for modeling by splitting it into training, validation, and testing sets.

These steps ensure the dataset is clean, consistent, and ready for building an effective recommendation system.

6. Exploratory Data Analysis (EDA)

1. Univariate Analysis

Objectives:

- Understand the distribution of individual features (e.g., user ratings, movie genres, release year).
- Identify outliers, missing values, and trends.

Suggested Visualizations:

1. Histograms:

- Distribution of user ratings (e.g., 1-5 scale).
- Popularity scores of movies.
- Release years of movies (e.g., clustering around specific decades).

2. Boxplots:

- Analyze the variability in movie ratings.
- Identify outliers in watch durations or popularity scores.

3. Countplots:

- Count of movies per genre.
- Count of users by demographic breakdown (e.g., age, gender).

2. Bivariate/Multivariate Analysis

Objectives:

- Explore relationships between multiple features.
- Understand how user behavior interacts with movie metadata.

Suggested Visualizations:

1. Correlation Matrix:

- Identify correlations between numerical variables (e.g., user ratings, popularity, runtime).

2. Scatterplots:

- Relationship between popularity score and average user rating.
- Watch duration vs. user ratings.

3. Grouped Bar Plots:

- Average ratings grouped by genres.
- Watch counts by demographic groups (e.g., age, region).

4. Pairplots:

- Pairwise relationships between rating, popularity, and runtime.

5. Heatmaps:

- Heatmap of user-movie interactions (e.g., watch counts or ratings).

7. Feature Engineering

1. User Interaction Features

- Watch Duration Percentage: Calculate the percentage of a movie watched by the user.
- Justification: Helps capture user engagement with a movie.
- Skip Count: Number of times a user skips content.
- Justification: Indicates disinterest or mismatch with user preferences.
- Rating Variance: Variance in a user's ratings across different genres or movies.
- Justification: Helps identify users with diverse or niche preferences.

2. Temporal Features

- Day of the Week: Extract the day of the week from the timestamp.
- Justification: User viewing habits may vary by day (e.g., weekends vs. weekdays).
- Time of Day: Categorize timestamps into morning, afternoon, evening, or night.
- Justification: Captures context for viewing preferences based on time.

3. Movie Metadata Features

- Genre Vectorization: Convert movie genres into one-hot encoded vectors or multi-label binary encoding.
- Justification: Allows the model to understand and utilize genre information effectively.

- Cast and Director Popularity: Aggregate popularity scores or ratings of the cast and director.
- Justification: Popular directors and actors can influence user preferences.
- Age of Movie: Calculate the difference between the movie's release year and the current year.
- Justification: Users may prefer newer or older movies based on personal tastes.

4. Contextual Features

- Device Type: Include the device type used for viewing (e.g., mobile, desktop, smart TV).
- Justification: Different devices may influence content preferences (e.g., short videos on mobile).
- Location-Based Features: Include user location data (e.g., country or region).
- Justification: Regional preferences might affect movie choices (e.g., language, culture).

8. Model Building

1. Collaborative Filtering with Matrix Factorization (Model 1)

Why Selected*:*

- Collaborative filtering is widely used for recommendation systems as it leverages user-item interaction data effectively.
- Matrix factorization techniques (e.g., Singular Value Decomposition - SVD) are powerful for identifying latent factors that represent user and item preferences.
- Suitable for structured interaction data (ratings, watch history).

Implementation:

- Use SVD (or its variants) to decompose the user-item interaction matrix into latent factors.
- Predict missing values (ratings) in the matrix to generate recommendations.

2. Content-Based Recommendation using K-Nearest Neighbors (KNN) (Model 2)

- Why Selected:

- Content-based filtering focuses on matching items (movies) based on their attributes (e.g., genres, cast, directors).
- KNN is intuitive and effective for similarity-based models.
- Suitable for scenarios where user-item interaction data is sparse (cold-start problem).

Implementation:

- Compute similarity scores between movies using cosine similarity or Euclidean distance on feature vectors.
- Recommend movies with the highest similarity scores for each user.

Data Preparation

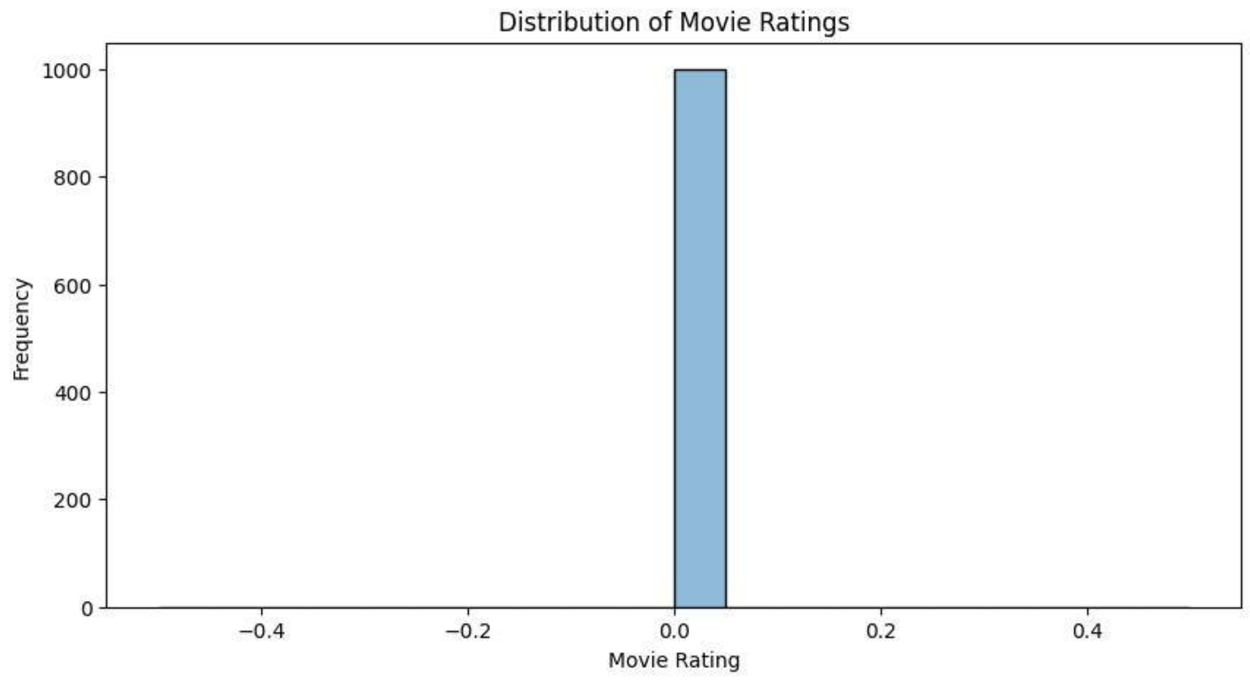
1. Dataset Splitting:

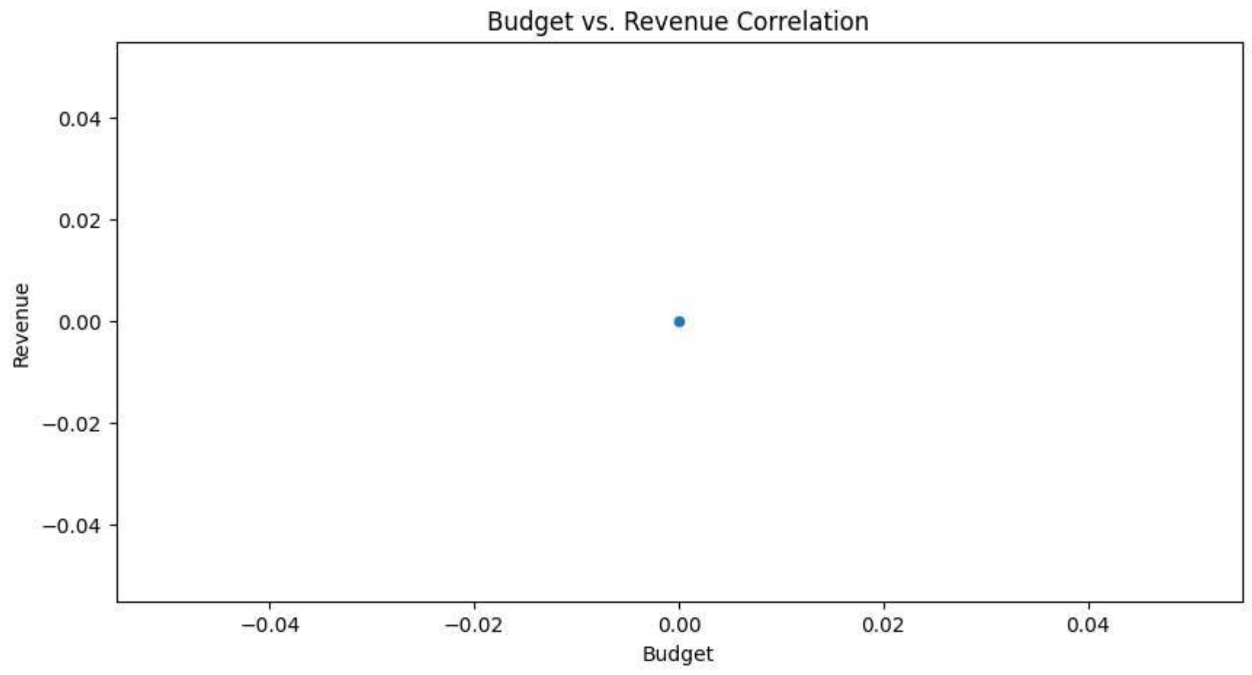
- Split the dataset into training (80%) and testing (20%) sets.
- Use stratified sampling to ensure balanced distribution of user interaction data across the splits.

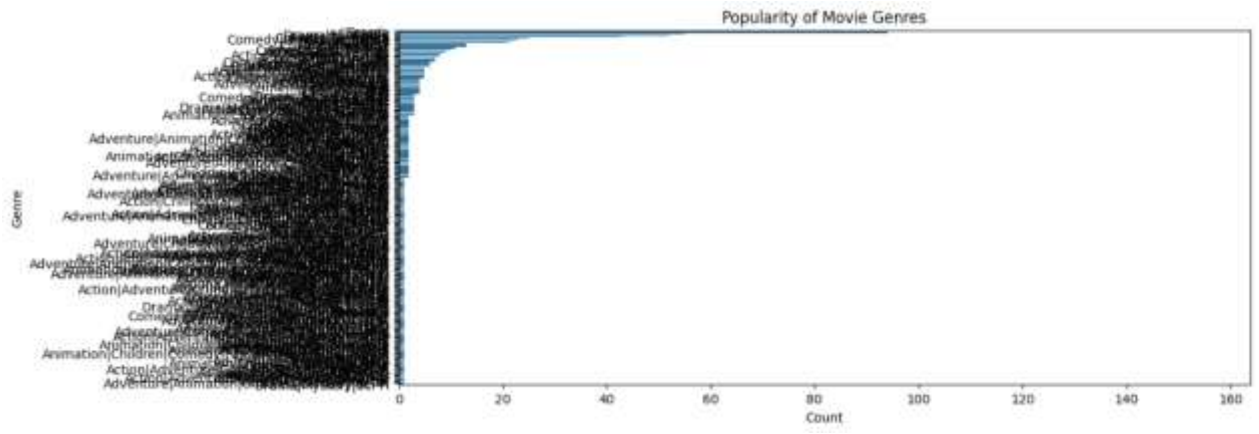
2. Preprocessing Steps:

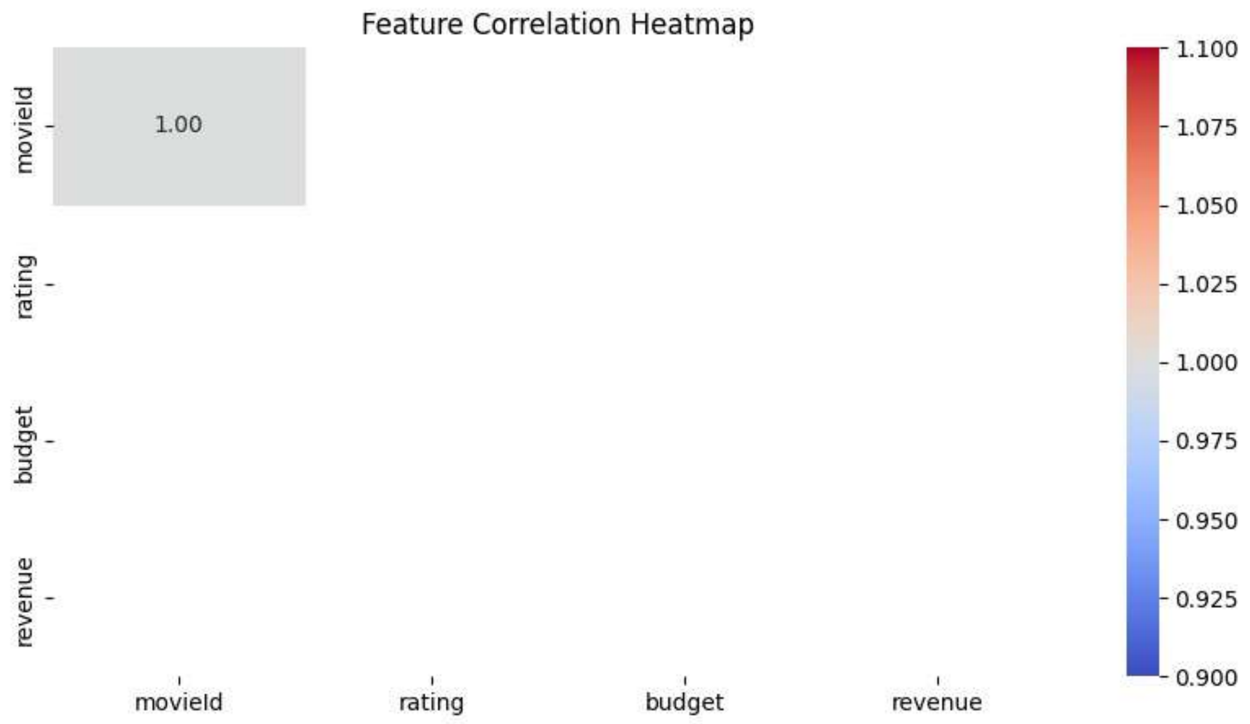
- Normalize numerical features (e.g., ratings, watch duration).
- One-hot encode categorical features (e.g., genres, languages).
- Create user-item interaction matrices for collaborative filtering.

9. Visualization of Results & Model Insights









10. Tools and Technologies Used

1. Programming Language

- Python: Chosen for its versatility, extensive libraries, and ease of use for data analysis and machine learning tasks.

2. IDE/Notebook

- Google Colab: Used for collaborative coding and access to free GPU resources for model training.
- VS Code: Used for writing and maintaining production-ready code.

3. Libraries

Data Manipulation and Analysis

- pandas: For handling and processing structured data.
- numpy: For numerical computations and array manipulations.

Data Visualization

- matplotlib: For creating basic plots and visualizations.
- seaborn: For advanced statistical data visualization.
- Plotly: For creating interactive and dynamic visualizations.

Machine Learning

- scikit-learn: For implementing machine learning models, preprocessing, and evaluation metrics.
- XGBoost: For gradient boosting algorithms to improve recommendation accuracy.

Recommender System-Specific Libraries

- Surprise: A Python library specifically designed for building and analyzing recommender systems.
- LightFM: For hybrid recommendation systems combining collaborative and content-based filtering.

4. Visualization Tools

- Plotly: Primarily used for interactive visualizations during EDA and model insights.
- Tableau: Employed for creating polished dashboards and sharing insights with stakeholders.
- Power BI: Used for integrating results into business intelligence workflows.

5. Cloud and Deployment Resources

- Google Cloud Platform (GCP): For deploying the recommendation engine and ensuring scalability.
- AWS S3: Used for storing large datasets.
- Docker: For containerization of the recommendation engine to ensure consistency across environments.

6. Other Tools

- GitHub: For version control and collaboration on code
- TMDb API: For accessing movie metadata.
- IMDB Open Dataset: As an additional source of movie-related data.

11. Team Members and Contributions

1. **PRADEEP. V** (Team Lead) He has done his role in building the skeleton code and executing the code

2**. BHAGAVATHI .K** He has done is role by performing data collection

3. **BHUVANESH . K** He has done is role in the documentation.