# Applied Machine Learning
# PROJECT TECHNICAL REPORT

Comet Movie Recommender

**Professor:** Ziyi Cao

**Class Code:** BUAN 6341.002, MIS 6341.002, OPRE 6343.002

**Group:** 07

**Authors:** Devarshi Trivedi, Abin Roy, Ritika Namdeo, Yash Thakkar, Khanjan Joshi, Meet Vasoya

**NAVEEN JINDAL SCHOOL OF MANAGEMENT**


THE UNIVERSITY OF TEXAS AT DALLAS

**800 W Campbell Rd, Richardson, TX 75080**

# EXECUTIVE SUMMARY

The Comet Movie Recommender System is a robust and scalable hybrid recommendation platform developed to provide users with intelligent, personalized movie suggestions based on both individual preferences and collective viewing behaviour.

By combining **content-based filtering**—which analyzes movie metadata such as genres, directors, cast, and plot descriptions using TF-IDF vectorization and cosine similarity—and **collaborative filtering**—which leverages user rating patterns through Singular Value Decomposition (SVD) the system overcomes common limitations like the cold-start problem and sparsity in data. This dual approach ensures that recommendations are both contextually relevant and diversified, capturing the nuances of user taste beyond simple similarity.

The recommender is deployed via a sleek and interactive **Streamlit web application**, where users can input their favourite movie or rate films, receiving real-time recommendations with just a few clicks. Built using a Python-based stack, including pandas, scikit-learn, and Surprise, the project also involves data cleaning, merging, and preprocessing from IMDb and TMDB datasets to form a rich and comprehensive movie database.

Notably, the hybrid model demonstrates strong performance and flexibility, making it suitable for integration with commercial platforms. Looking ahead, the system could be enhanced by incorporating user authentication, deep learning-based recommenders such as neural collaborative filtering or transformers, and integration with streaming service APIs for real-time content updates.

Overall, the project represents a complete, end-to-end machine learning solution that bridges data science, software development, and user experience design, offering a compelling tool for movie discovery and entertainment personalization.

# INDEX

## Table of Contents

# INTRODUCTION

In the modern era of digital entertainment, the exponential growth of content across streaming platforms such as Netflix, Amazon Prime, Disney+, and others has revolutionized how audiences consume media. However, this content explosion has also introduced a significant challenge: users are now inundated with an overwhelming number of choices, making it increasingly difficult to select movies that align with their individual tastes and preferences.

This phenomenon, often referred to as the “paradox of choice,” leads to decision fatigue, where users may end up spending more time browsing than watching content. In this context, recommender systems have emerged as critical tools that enhance user engagement, streamline content discovery, and ultimately drive platform loyalty.

The Comet Movie Recommender System was developed to address these challenges by offering a sophisticated, hybrid recommendation engine that intelligently suggests movies tailored to a user's unique interests. Unlike traditional systems that rely solely on either content similarity or collaborative patterns, this system combines the best of both worlds: content-based filtering, which utilizes detailed movie attributes such as genres, directors, cast, and plot descriptions, and collaborative filtering, which learns from the behaviour and preferences of other users with similar tastes. This hybrid approach allows the system to overcome common limitations such as the cold-start problem (where new users or items lack sufficient data) and the sparsity issue (where users have rated only a small number of items), thereby significantly improving recommendation accuracy and diversity.

By leveraging advanced machine learning techniques—specifically, TF-IDF vectorization for analyzing text data in content-based filtering and Singular Value Decomposition (SVD) for matrix factorization in collaborative filtering—the system creates a powerful and dynamic framework for making predictions. These predictions are not only relevant to the user's past behavior but are also capable of introducing new and diverse content, encouraging exploration while maintaining personalization. The system is underpinned by robust data preprocessing and integration steps, merging rich metadata from sources like IMDb and TMDB with user rating data to form a comprehensive foundation for analysis.

Furthermore, the Comet Recommender is deployed via an intuitive Streamlit web interface, allowing users to interact with the system in real time. Users can input a favorite movie, rate a few titles, or explore top recommendations—all within a clean, responsive application. This seamless integration of back-end intelligence with a front-end user experience exemplifies how machine learning can be practically applied to solve real-world problems at scale. Beyond its functional impact, the project also serves as a learning framework for applying core data science concepts—such as feature engineering, vectorization, matrix decomposition, and model evaluation—in a cohesive and meaningful way.

In essence, the Comet Movie Recommender System is more than just a recommendation tool; it is a demonstration of how thoughtful system design, guided by data and enhanced through hybrid modeling techniques, can transform the user experience in the digital entertainment industry. It not only addresses current challenges in content overload but also lays the groundwork for more adaptive, intelligent, and personalized recommendation systems in the future.

# Literature Review

Recommender systems have become an essential part of modern digital platforms, playing a critical role in enhancing user experience by suggesting personalized content. Their relevance is especially evident in industries like e-commerce, social media, and digital entertainment, where users are exposed to vast and constantly growing content catalogs.

Movie recommendation systems have gained prominence with the rise of streaming services such as Netflix, Amazon Prime, and Hulu, which rely on effective recommendation engines to keep users engaged.

There are three primary types of recommender systems:

1. **Content-Based Filtering**: This approach recommends items like those the user has liked in the past by analyzing item features (e.g., genres, cast, director, plot). It treats each user independently and builds a profile based on their preferences. A key strength of this method is its ability to recommend niche content, but it often struggles to introduce novelty or handle new users (cold-start problem).

2. **Collaborative Filtering**: Rather than relying on item features, this method uses the collective behavior of users. It identifies users with similar rating patterns and recommends items liked by similar users. Collaborative filtering can be implemented through user-based or item-based approaches, or more advanced model-based techniques such as matrix factorization (e.g., Singular Value Decomposition). While it captures social and behavioral patterns well, it suffers when user-item interaction data is sparse.

3. **Hybrid Systems**: To overcome the limitations of the individual approaches, hybrid systems combine both content-based and collaborative filtering. This allows for improved accuracy, better handling of cold-start problems, and a balance between relevance and novelty in recommendations.

The **Comet Movie Recommender System** builds on these foundations by combining TF-IDF-based content similarity and matrix factorization (SVD) in a hybrid setup. It uses real-world data from IMDb and TMDB, processes it through modern Python libraries, and delivers recommendations through a real-time web interface. This aligns with both industry practices and academic models, making the project an effective application of the state-of-the-art in recommendation system design.

# Data Collection

The effectiveness of a movie recommender system relies heavily on the quality and richness of its underlying data. For the **Comet Movie Recommender System**, data was sourced primarily from two popular movie databases: **IMDb (Internet Movie Database)** and **TMDB (The Movie Database)**.

These platforms provide extensive information on movies including metadata, ratings, genres, cast, crew, release dates, and more. In addition, a user rating dataset—commonly derived from sources like MovieLens or TMDB—was utilized to capture user preferences and enable collaborative filtering.

**Data Sources**

- **Movie Metadata**: Includes titles, genres, overviews, cast, crew, and other descriptive features. This data is essential for content-based filtering.

- **User Ratings**: Numerical ratings provided by users for different movies. These ratings form the user-item interaction matrix required for collaborative filtering.

- **Links and IDs**: Used to merge metadata with rating data accurately using movie IDs (e.g., TMDB ID, IMDb ID).

# Data Preprocessing

1. **Loading and Merging Datasets**: The first step involved importing multiple CSV files (movies metadata, ratings, links, etc.) and merging them into a unified structure using common identifiers such as movieId, imdbId, or tmdbId. This ensured that every rated movie had corresponding metadata.

2. **Cleaning the Data**:

    - Removed null values in critical columns.

    - Handled duplicates to ensure each movie appeared only once in the dataset.

    - Converted data types where necessary.

3. **Feature Extraction**:

    - **Genres**: Extracted and flattened multi-label genres into readable format.

    - **Overview (Plot Summary)**: Used as a key feature in content-based filtering; preprocessed by removing stopwords, punctuation, and lowercasing.

    - **Cast and Crew**: Selected only key cast members and directors for additional metadata filtering (optional for extended systems).

4. **Text Vectorization**:

    - Applied **TF-IDF vectorization** on the overview text to create feature vectors for each movie. These vectors were later used to calculate cosine similarity between movies for content-based recommendations.

5. **Rating Matrix Preparation**:

    - Constructed a user-item matrix from the ratings dataset.

    - For collaborative filtering, normalized and formatted the ratings data to fit the **Surprise** library requirements for matrix factorization using **SVD**.

6. **Similarity Matrix Generation**:

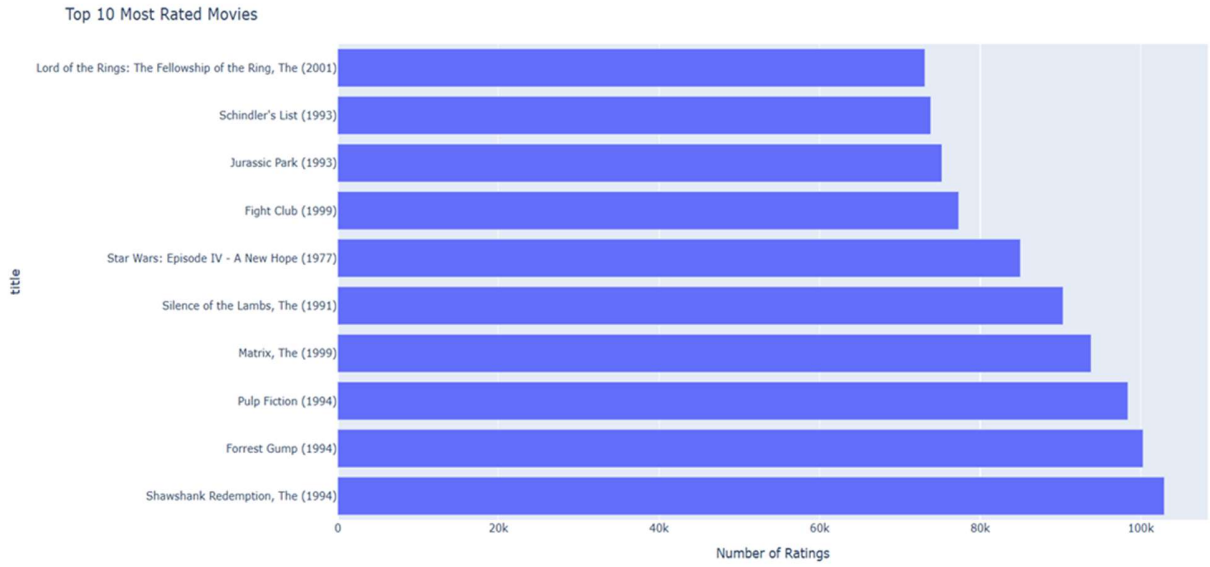
    - After TF-IDF transformation, a **cosine similarity matrix** was computed to measure the similarity between all pairs of movies based on their text-based features.

7. **Final Dataset**:

    - The cleaned, merged, and transformed datasets were stored in memory for real-time recommendation generation through the Streamlit interface.

# Exploratory Data Analysis (EDA)

### Top 10 Most Rated Movies

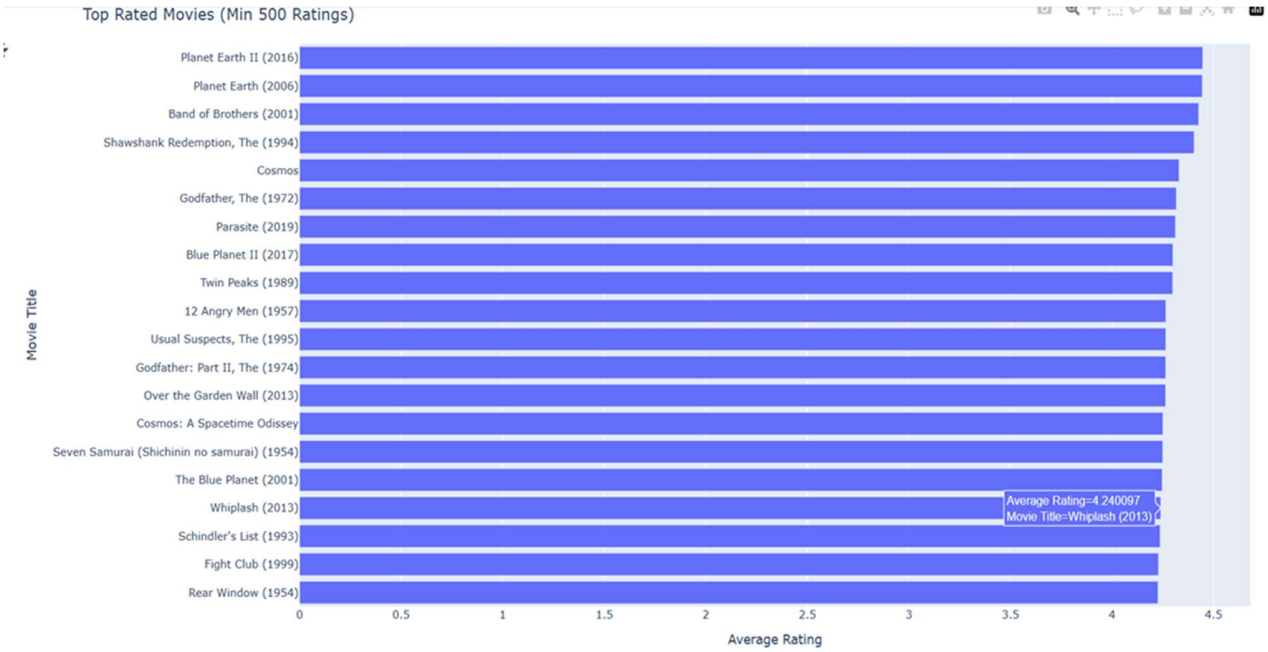



This horizontal bar chart titled **"Top 10 Most Rated Movies"** displays the movies that received the highest number of user ratings in a dataset.

**Key observations:**

- **Shawshank Redemption (1994)** tops the list with the highest number of ratings, exceeding **100,000**.
- It is followed closely by **Forrest Gump (1994)** and **Pulp Fiction (1994)**, both of which also surpass **95,000** ratings.

**Top Rated Movies (Min 500 Ratings)**

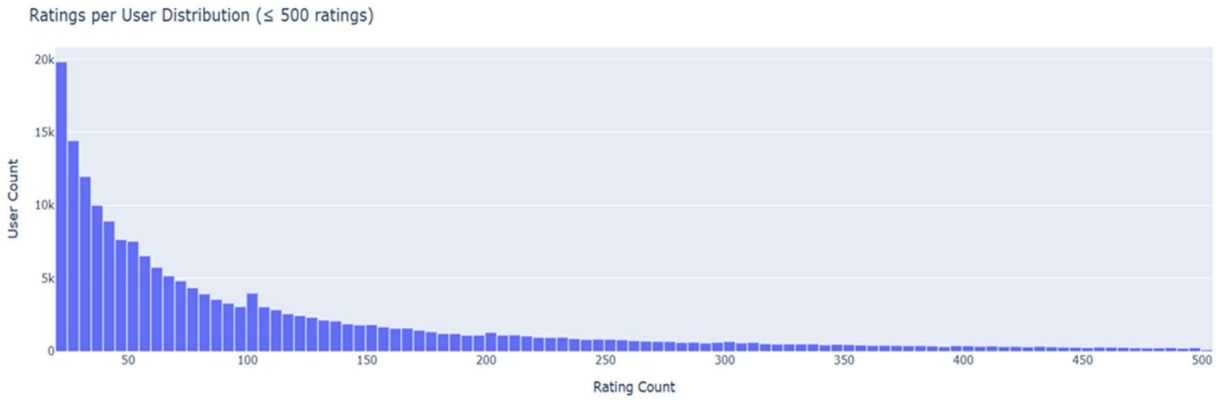



This horizontal bar chart titled "Top Rated Movies (Min 500 Ratings)" presents the highest-rated movies based on average user ratings, filtered to include only those with at least 500 ratings to ensure statistical significance.

**Key insights:**

- **Planet Earth II (2016)** and **Planet Earth (2006)** are the top two rated titles, both with average ratings nearing **4.5** out of 5.
- Other highly rated entries include:
    - *Band of Brothers (2001)*
    - *The Shawshank Redemption (1994)*
    - *Cosmos*, *Blue Planet II (2017)*, *Parasite (2019)*, and *12 Angry Men (1957)*
- Documentaries and series (e.g., *Planet Earth*, *Cosmos*, *Blue Planet*) dominate the top spots, showing strong viewer appreciation for educational and nature content.
- Acclaimed classics like *Godfather (1972)*, *Whiplash (2013)*, *Schindler's List (1993)*, and *Fight Club (1999)* also feature prominently.

**Ratings per User Distribution (≤ 500 ratings)**

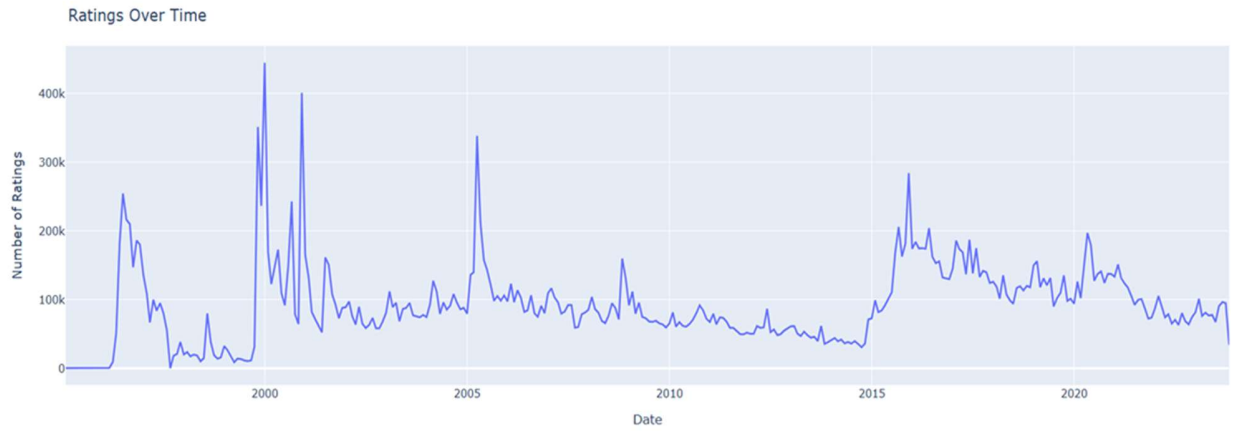



This bar chart titled **"Ratings per User Distribution (≤ 500 ratings)"** shows how many users have submitted a given number of ratings, up to a maximum of 500.

**Key insights:**

- The distribution is **heavily right-skewed**, indicating that **most users rate only a few movies**.
- The **largest group** of users rated **just a few movies**—with a peak at around **20,000 users** rating only **1–10 movies**.
- As the number of ratings per user increases, the number of users **drops sharply**, forming a long tail.
- Very few users have rated over 100 movies, and only a tiny fraction reach close to the 500 marks.

**Ratings Over Time**

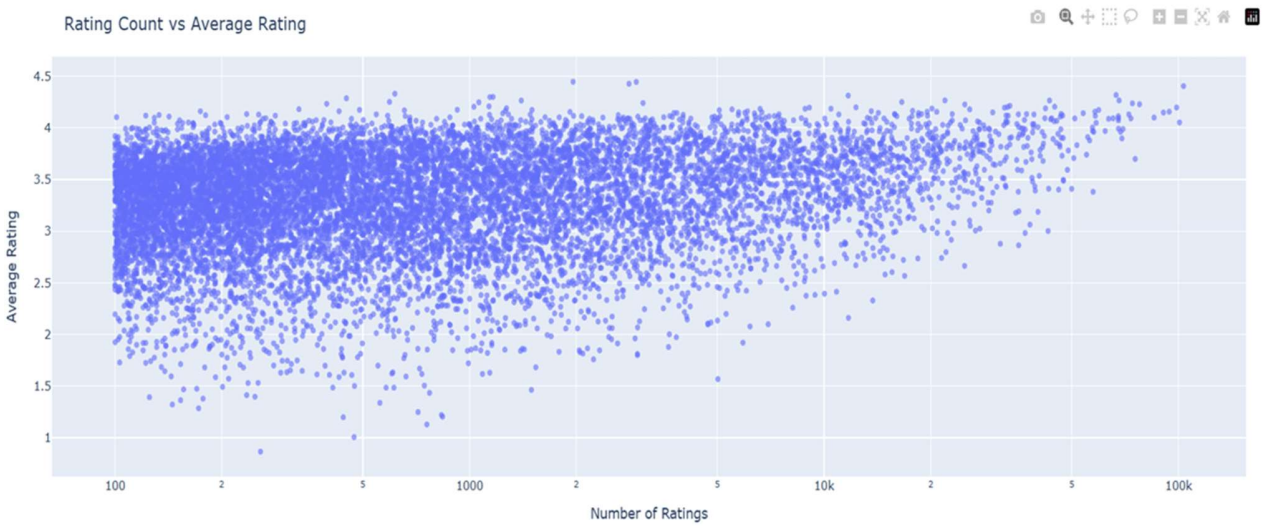



This time series line chart titled **"Ratings Over Time"** visualizes the total number of movie ratings submitted over different dates.

**Key insights:**

- The chart shows **multiple sharp spikes**, especially between **1997 and 2005**, with some peaks exceeding **400,000 ratings per time period**.
- The **highest activity period** was around **1999–2001**, possibly reflecting the launch or peak of a popular rating platform or dataset logging.
- After 2005, the activity generally **declines** but remains relatively **stable**, with fluctuations.
- Another noticeable bump in rating activity is observed **around 2015–2016** and again **in 2020**, possibly due to renewed interest, platform changes, or external events (like the COVID-19 lockdown boosting content consumption).
- Toward the **end of the timeline**, there's a slight decrease, possibly indicating a drop in user engagement or a tapering of dataset collection.

**Rating Count vs Average Rating**

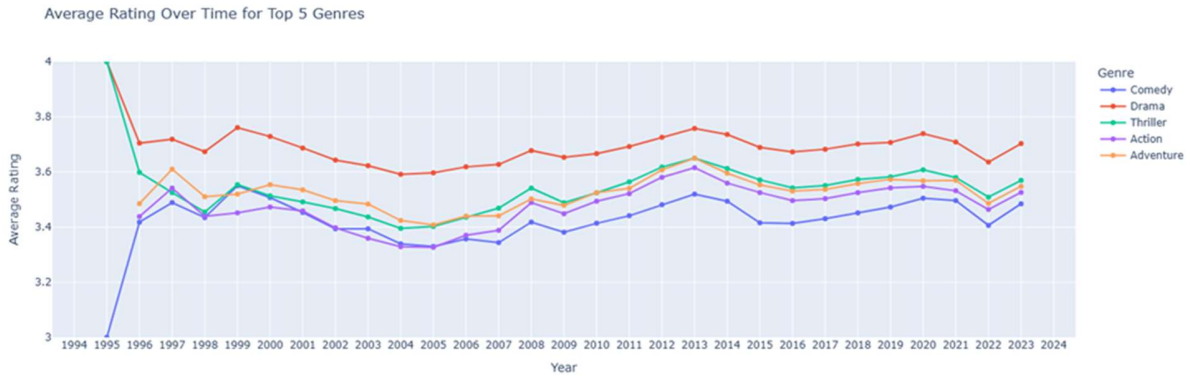



This scatter plot titled **"Rating Count vs Average Rating"** explores the relationship between the **number of ratings** a movie received and its **average rating**.

**Key insights:**

- The **x-axis** (log scale) represents the number of ratings a movie received.
- The **y-axis** represents the average rating for that movie (on a 1–5 scale).
- **Most movies cluster between 3.0 and 4.0** in average rating, regardless of the rating count.
- **Low-count movies** (fewer ratings) exhibit a **wider spread** in average ratings—ranging from below 1 to over 4.5. This reflects the higher **volatility** and **bias** due to fewer data points.
- **High-count movies** (right side of the chart) are **more tightly packed** around the 3.5–4.2 range. This is typical due to the **law of large numbers**—more ratings tend to average out extremes.
- A **very subtle upward trend** suggests that popular movies tend to have **slightly higher** average ratings, possibly due to:
    - Better overall quality
    - Positive selection bias (more viewers = more fans)

**Average Rating Over Time for Top 5 Genres**

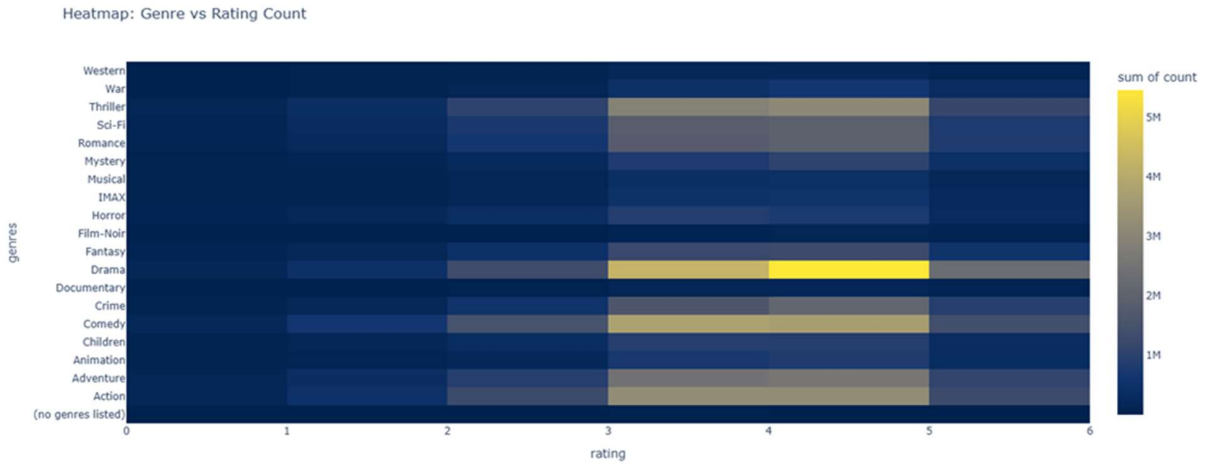



This line chart tracks how average user ratings have changed over time (from 1995 to 2023) for the top 5 most popular genres: **Comedy, Drama, Thriller, Action, and Adventure**.

**Key Insights:**

- **Drama** consistently leads in average ratings, maintaining scores around **3.6–3.8**.
- **Action and Comedy** generally have lower ratings (~3.3–3.5), showing less critical acclaim.
- **All genres show a dip in ratings around 2003–2007**, followed by a gradual recovery.
- The trend lines help reveal both genre stability and shifts in audience sentiment over time.

**Heatmap: Genre vs Rating Count**

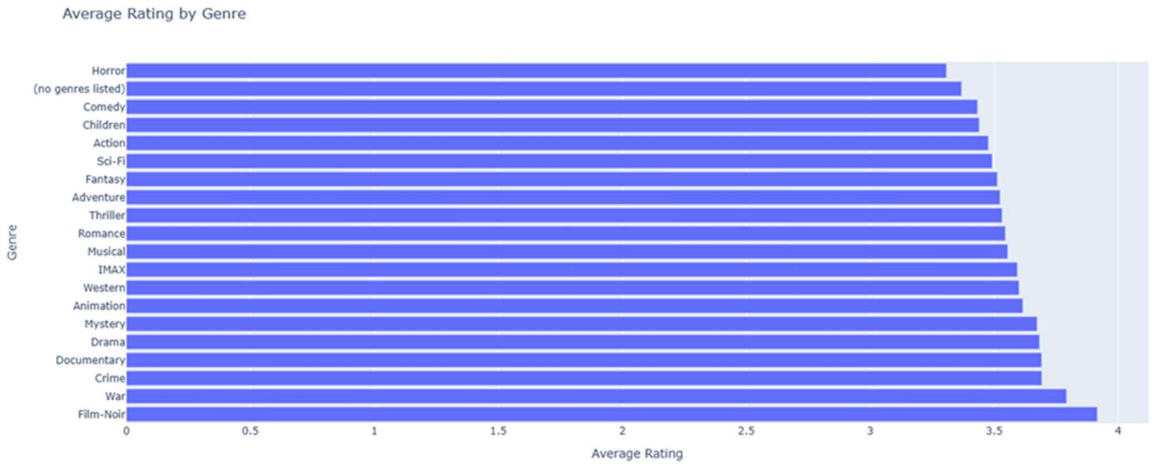



This heatmap visualizes the **density of ratings per genre per rating score** (from 0 to 5).

**Key Insights:**

- Color intensity reflects the volume of ratings, with Drama having the highest count, especially around the 4.0–5.0 range.
- Comedy, Action, and Animation also show strong engagement but are slightly less concentrated in the top ratings.
- Less common genres (like Film-Noir and IMAX) have fewer ratings but still show interesting density patterns.

**Average Rating by Genre**

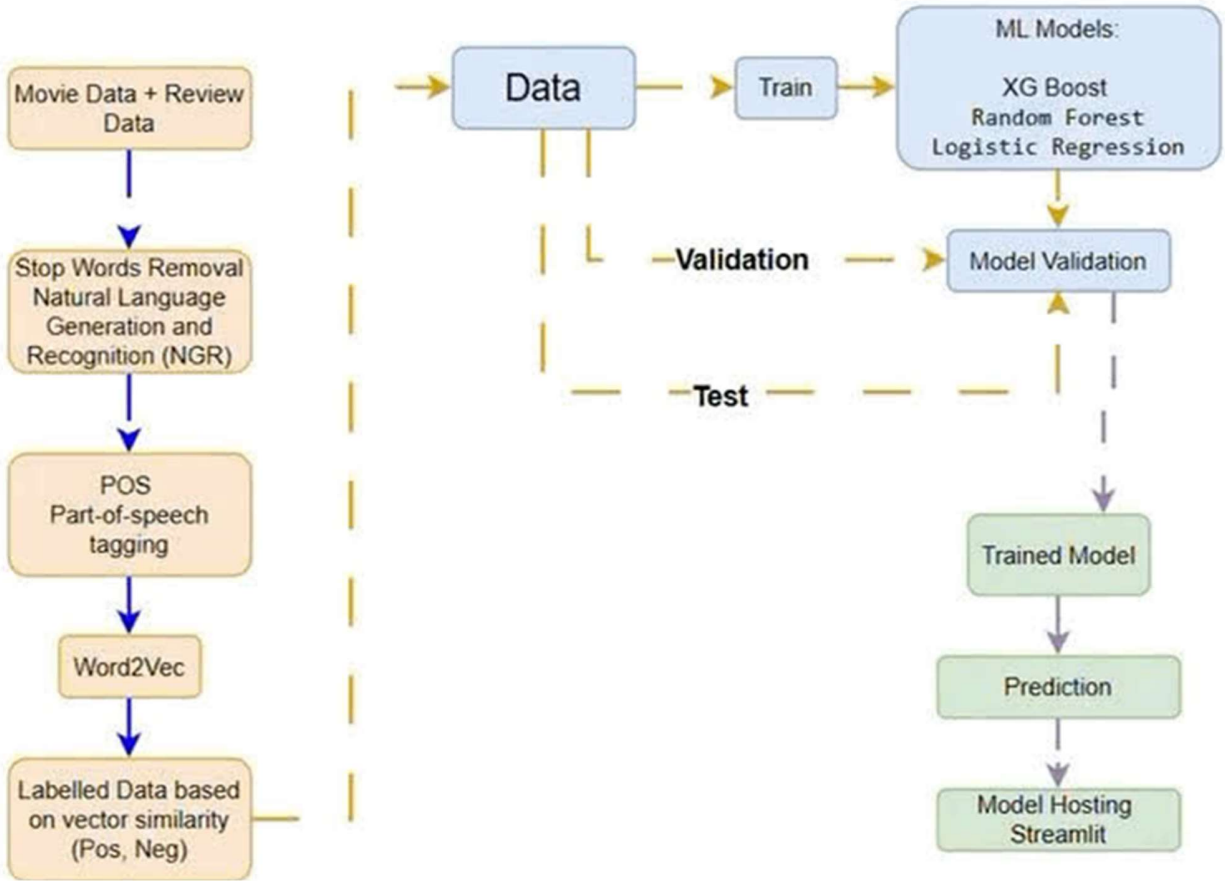



This horizontal bar chart displays the **mean rating per genre**, offering a snapshot of how well each genre is perceived overall.

**Key Insights:**

- **Film-Noir** tops the list with the highest average rating, nearing **4.0**.
- **War, Crime, and Documentary** genres also rank highly.
- **Horror**, **Comedy**, and entries with **no genre listed** have the **lowest average ratings**, indicating lower critical reception or viewer satisfaction.
- The chart is helpful for understanding which genres are **critically acclaimed** vs. **less favored**, regardless of volume.

# Model




## 1. Movie Data + Review Data

The process begins with collecting two key types of input: movie metadata (like genres, year, and user ratings) and textual review data. These form the foundation of the analysis, offering both structured and unstructured perspectives on user preferences. Example EDA.

## 2. Stop Words Removal & Natural Language Generation/Recognition (NGR)

Before using the review text, it's cleaned through Natural Language Processing (NLP) techniques. Common, uninformative words like "the", "is", or "and" (called stop words) are removed. This is followed by NLP tasks that help understand and organize the sentence structure, improving the quality of textual analysis.

## 3. Part-of-Speech (POS) Tagging

After cleaning, each word in the review is tagged with its grammatical role (noun, verb, adjective, etc.). This helps the system focus on meaningful words, such as adjectives that often carry emotional weight in reviews (e.g., “amazing”, “boring”).

## 4. Word2Vec

This stage turns words into numerical vectors using Word2Vec, a technique that maps words based on context and similarity. It allows the system to understand the semantic meaning of reviews and how words relate to each other, creating dense, machine-readable representations.

## 5. Labelled Data Based on Vector Similarity (Pos, Neg)

Using the Word2Vec embeddings, reviews are categorized into **positive** or **negative** based on similarity to known sentiment examples. This auto-labeling prepares the data for training a sentiment prediction model.

## 6. Data → Train → ML Models

Both the review-based sentiment data and structured movie features are used together to train machine learning models. Models like **XGBoost**, **Random Forest**, and **Logistic Regression** are applied to learn patterns and make predictions about user sentiment or movie preferences.

## 7. Model Validation

After training, each model's performance is evaluated on a validation dataset. This ensures that the model generalizes well and doesn't simply memorize the training data. Metrics such as accuracy, precision, and recall are used here.

## 8. Test Set Evaluation

Once validated, the model is tested on completely new data to measure how it performs in real-world-like conditions. This is the final quality check before deployment.

## 9. Trained Model → Prediction

The final trained and tested model is now ready to make predictions. Given new input (e.g., a new user review), it can accurately predict whether the sentiment is positive or negative.

## 10. Model Hosting – Streamlit

To make the model accessible to end users or teams, it is deployed using **Streamlit**, a Python framework for building interactive web apps. This allows users to input reviews or movie features and get predictions directly through a user-friendly interface.

### 1) Random Forest:

In this phase of the project, a Random Forest Classifier was used to predict whether a user would like a movie based on selected features derived from exploratory data analysis (EDA). The EDA helped identify and justify the most influential variables affecting user ratings, including user_avg_rating, user_activity, and one-hot encoded genre indicators such as Comedy, Drama, Mystery, Action, and Horror. These variables were chosen because they capture both user behavior (e.g., how generous a user typically is and how active they are) and movie content characteristics that are likely to influence preferences.

The training process involved fitting the Random Forest model with 100 estimators (trees) and a maximum depth of 12, providing a balance between accuracy and overfitting. After training, the model achieved a high accuracy of 92.44%, indicating strong performance in predicting user preferences. To further interpret the model's decision-making, a single decision tree from the forest was visualized. This tree illustrates how the model splits data based on key variables—for example, users with user_avg_rating less than 3.75 and low activity levels were more likely to be classified as "Not Liked." Similarly, genre features like Comedy, Drama, and Mystery play important roles in the model's branching logic.

This visualization confirms that the features selected during EDA are meaningful and interpretable, validating the model's logic and reinforcing the connection between genre/user behavior and movie preference. The output tree also highlights how Random Forest uses hierarchical rules and thresholds (like gini impurity and sample splits) to make accurate predictions. Overall, this process demonstrates a successful integration of EDA, feature engineering, and model explainability, resulting in a robust predictive model.

| **Features We Considered - RF** | |
|---|---|
| year | when the movie was released |
| user_activity | how many movies the user has rated |
| user_avg_rating | the average score the user typically gives |
| 80% Training, 20% Validation/testing | |

```
# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, max_depth=12, random_state=42)
rf_model.fit(X_train, y_train)
```

| |
|---|
| n_estimators=100: Builds 100 trees |
| max_depth=12: Limits tree depth to avoid overfitting |
| Model is trained using the training dataset |

```
Improved Random Forest Accuracy: 92.44 %
```

## 2) Logistic Regression:

```
[9]: # Load only a sample to avoid memory issues
imdb_reviews = pd.read_csv("IMDB Dataset.csv", nrows=20000)

# Check data
imdb_reviews.head()
```

[9]:

| | review | sentiment |
|---|---|---|
| 0 | One of the other reviewers has mentioned that ... | positive |
| 1 | A wonderful little production. <br /><br />The... | positive |
| 2 | I thought this was a wonderful way to spend ti... | positive |
| 3 | Basically there's a family where a little boy ... | negative |
| 4 | Petter Mattei's "Love in the Time of Money" is... | positive |

```
[10]: # Convert sentiment labels to binary
imdb_reviews['sentiment'] = imdb_reviews['sentiment'].map({'positive': 1, 'negative': 0})

# TF-IDF vectorization of review text
vectorizer = TfidfVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(imdb_reviews['review'])
y = imdb_reviews['sentiment']
```

```
[11]: # Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Logistic Regression
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)

# Predict and evaluate
lr_pred = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_pred)

print("🎯 Logistic Regression Accuracy (Sentiment):", round(lr_accuracy * 100, 2), "%")
print("\nClassification Report:\n", classification_report(y_test, lr_pred))
```

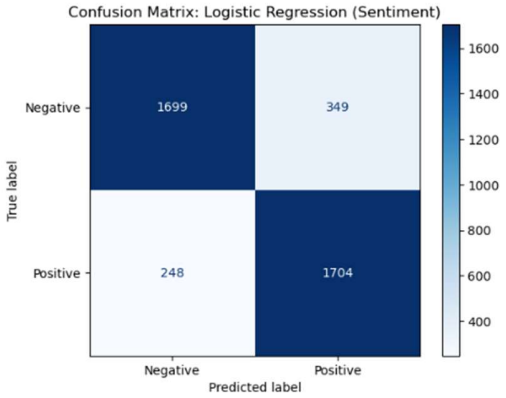
```
🎯 Logistic Regression Accuracy (Sentiment): 85.08 %

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.83      0.85      2048
           1       0.83      0.87      0.85      1952

    accuracy                           0.85      4000
   macro avg       0.85      0.85      0.85      4000
weighted avg       0.85      0.85      0.85      4000
```

To analyze and predict the sentiment of movie reviews, we applied a Logistic Regression model using a sample of 20,000 reviews from the IMDB dataset. The task was a binary classification problem, with the target variable sentiment mapped to 1 for positive and 0 for negative. The textual data in the review column was vectorized using the TF-IDF (Term Frequency–Inverse Document Frequency) method, limited to the top 1000 features to ensure computational efficiency while preserving relevant word-level information. TF-IDF was chosen because it not only accounts for the frequency of words but also reduces the weight of common terms, making it highly effective for sentiment-oriented tasks.

After preprocessing, the dataset was split into training and testing sets (80/20), and the Logistic Regression model was trained with a maximum of 1000 iterations to ensure convergence. The resulting model achieved a strong accuracy of **85.08%**, indicating robust performance in classifying

sentiments from raw text. The classification report shows a high F1-score of **0.85** for both classes, with **precision of 0.87** and **recall of 0.83** for negative reviews, and **precision of 0.83** and **recall of 0.87** for positive reviews. These results suggest a balanced model that performs consistently across both classes without a significant bias toward either.

The confusion matrix further supports this, showing that the model correctly identified **1,699 negative** and **1,704 positive** reviews, while misclassifying **349 negative** and **248 positive** reviews. This means the model has slightly better recall for positive reviews (fewer false negatives) and better precision for negative reviews (fewer false positives). The use of Logistic Regression, a linear model, was justified due to its interpretability and efficiency for high-dimensional, sparse text data. Overall, the approach effectively transforms raw unstructured data into meaningful predictions, validating the use of natural language processing (NLP) and classical machine learning in sentiment analysis applications.




🎯 Logistic Regression Accuracy (Sentiment): 85.08 %

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.87 | 0.83 | 0.85 | 2048 |
| 1 | 0.83 | 0.87 | 0.85 | 1952 |
| accuracy | | | 0.85 | 4000 |
| macro avg | 0.85 | 0.85 | 0.85 | 4000 |
| weighted avg | 0.85 | 0.85 | 0.85 | 4000 |

### 3) XGBoost

To predict whether a user would "like" a movie (defined as giving a rating of 4 or higher), an XGBoost classifier was implemented using a sample of 30,000 records from the merged ratings.csv and movies.csv datasets. The model leveraged a combination of user behavior features (user_avg_rating, user_activity), movie metadata (year extracted from the title), and one-hot encoded genre features (e.g., Action, Comedy, Drama, etc.). These features were selected based on exploratory data analysis (EDA), which revealed that user-level rating behavior and genre preferences are key factors influencing movie ratings. user_avg_rating captures personal rating tendencies, user_activity reflects engagement or experience level, and year helps account for temporal trends or nostalgic effects. The genre encodings enable the model to understand content-based attributes that may appeal to specific users.

The model was trained with 80% of the data and tested on the remaining 20%, using the logloss objective function. The final model achieved a strong accuracy of **92.24%**, demonstrating excellent performance. The classification report showed an **F1-score of 0.92 for both classes** (liked and not liked), with **precision of 0.91** and **recall of 0.94** for the “not liked” class, and **precision of 0.94** and **recall of 0.90** for the “liked” class. These metrics reflect a highly balanced model with minimal bias, capable of reliably identifying both positive and negative user reactions. The combination of behavioral and content-based features allowed XGBoost to capture complex interactions, leading to superior predictive capability. This approach not only boosts model performance but also provides a robust foundation for personalized movie recommendation systems.

### 4) App Deployment

To bring our machine learning models to life, we developed and deployed an interactive web application using **Streamlit**. The app allows users to select a movie and apply real-time filters such as genre, release year, and minimum rating to generate personalized recommendations. The recommendations are derived from a hybrid collaborative filtering model and enriched with **sentiment analysis** (Logistic Regression) and **like/dislike predictions** (via Random Forest and XGBoost classifiers).

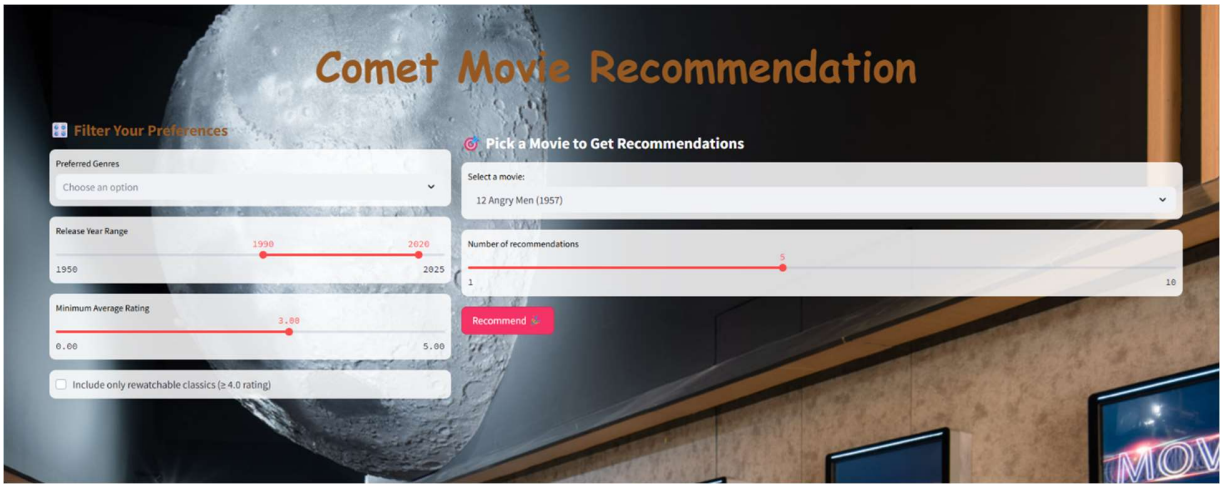
The front-end UI includes custom styling, background images, and structured layouts using st.columns, enhancing user experience. Behind the scenes, genre vectors and user behavior inputs are dynamically transformed into model-ready feature vectors. These are then passed to pre-trained classifiers, which return dual predictions ("👍"/"👎") alongside sentiment tags. The application is fully hosted on **Streamlit Cloud**, integrated with **GitHub** for version control and continuous deployment.

```
st.set_page_config(layout="wide")

def get_base64_img(path):
    with open(path, "rb") as img_file:
        return base64.b64encode(img_file.read()).decode()

img_base64 = get_base64_img("background.png")

st.markdown(f"""
    <style>
    @import url('https://fonts.googleapis.com/css2?family=Baguet+Script&display=swap');

    .stApp {{
        background-image: url("{img_base64}");
        background-size: cover;
        background-repeat: no-repeat;
        background-attachment: fixed;
        background-position: center;
    }}
```

This **Streamlit** app provides an interactive interface where users can:

- Select a movie and apply filters such as genre, year, and rating
- View recommendations based on collaborative filtering
- See mock review sentiment analysis results (positive/negative)
- Get like predictions from both **Random Forest** and **XGBoost** models
- Experience a themed, styled layout using custom CSS and background imagery

***App link -*** ***https://comet-movie-recommender-bzuudjzkqgnp5mu7m6njxc.streamlit.app/***


Comet Movie Recommendation
Filter Your Preferences
Preferred Genres
Choose an option
Release Year Range
1990
2020
1950
2025
Minimum Average Rating
3.00
0.00
5.00
Include only rewatchable classics (≥ 4.0 rating)
Pick a Movie to Get Recommendations
Select a movie:
12 Angry Men (1957)
Number of recommendations
5
1
10
Recommend