Netflix recommendation System

A Project Report Submitted

in partial fulfilment of the

Requirements for the award of the Degree of

MASTER OF SCIENCE (DATA ANALYTICS)

By

Shreya Bhattacharjee (6861)

Under the esteemed guidance of

Mr. OMKAR SHERKHANE

Designation: Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE

MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE AND SCIENCE

(AUTONOMOUS), NEW PANVEL,410206, MAHARASHTRA

(*Affiliated to University of Mumbai)*

2024-2025

MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE AND SCIENCE

(AUTONOMOUS), NEW PANVEL, 410206,
MAHARASHTRA

(*Affiliated to University of Mumbai)*

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled. **"Netflix recommendation System"**, is Bonafede work of **Shreya Bhattacharjee** bearing Roll. No: **6861** submitted in fulfilment of the requirements for the award of degree ofM.Sc. Data Analytics from University of Mumbai.

Internal guide **Coorditor**

College Seal

Date: **External Examiner**

ACKNOWLEDGEMENT

I Miss. Shreya Bhattacharjee student of PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE (AUTONOMOUS), NEW PANVEL would like to express my sincere gratitude towards our college's Computer Science Department.

I would like to thank Mr. OMKAR SHERKHANE Coordinator. (M.Sc. D.A) for granting me the opportunity to build project for the college. The project would have not been completed without the dedication, creativity and the enthusiasm which my family provided.

Yours faithfully,

Shreya Bhattacharjee

(Final Year Data Analytics)

DECLARATION

I hereby declare that the project entitled, “Netflix recommendation System” done at Pillai College of Arts, Commerce and Science (Autonomous), New Panvel, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of MASTER OF SCIENCE (DATA ANALYTICS) to be submitted as final semester project as part of our curriculum.

Shreya Bhattacharjee
Name and Signature of the Student

ABSTRACT

This project focuses on building a recommendation system for Netflix that personalizes content suggestions for users based on their viewing habits and preferences. With the vast amount of content available on streaming platforms, providing relevant recommendations becomes crucial for enhancing user satisfaction and engagement.
The system aims to analyze various factors, such as a user's watch history, ratings, genre preferences, and the behavior of similar users, to deliver tailored recommendations.

The core of this recommendation system lies in machine learning techniques such as collaborative filtering and content-based filtering. Collaborative filtering leverages data from users with similar preferences to predict and suggest content, while content-based filtering analyzes the features of movies or shows (like genres, cast, or directors) that the user has already enjoyed. By combining both approaches, the system can offer more accurate and diverse recommendations.

The project involves several stages, including data collection, preprocessing, feature extraction, model training, and performance evaluation. The effectiveness of the recommendation system is evaluated using metrics such as accuracy, precision, and recall, ensuring that the suggestions meet user expectations. This system not only aims to provide a better viewing experience but also enhances user retention by ensuring that users are consistently presented with engaging and relevant content.

Index

Chapter 1

Introduction

Recommendation systems have become indispensable tools in modern streaming platforms like Netflix, assisting users in discovering content amidst vast and ever-growing libraries. These systems provide personalized suggestions that not only improve user satisfaction but also drive platform engagement by offering relevant and appealing content. This project focuses on the development of a Netflix-like recommendation system, utilizing user data such as viewing history, preferences, and interaction patterns to predict and recommend movies or shows that are most likely to capture the user's interest.

The foundation of the recommendation system lies in two key machine learning techniques: **collaborative filtering** and **content-based filtering**. Collaborative filtering works by identifying users with similar tastes and recommending content based on shared preferences. It uses user-item interaction data (like ratings or views) to find patterns of similarity between users. On the other hand, content-based filtering analyzes the characteristics or metadata of the content (such as genres, cast, or directors) that users have previously liked and suggests similar items. Combining both approaches creates a more robust system that not only suggests content based on the behavior of similar users but also considers the unique attributes of the content itself.
This project follows a structured process involving data collection, preprocessing, feature engineering, and model building. First, user interaction data and content features are collected and cleaned to ensure high-quality input. Next, collaborative and content-based filtering models are trained and evaluated using standard performance metrics such as accuracy, precision, recall, and F1 score. Additionally, model tuning and optimization are carried out to enhance the relevance and diversity of the recommendations.

Chapter 2
Review of Literature

2.1 Background

The vast growth of digital content, particularly on streaming platforms like Netflix, has made it increasingly difficult for users to find relevant and engaging content without assistance. With thousands of movies, shows, and documentaries available, users often face the challenge of content overload. To address this issue, recommendation systems have emerged as essential tools, leveraging user interaction data to suggest personalized content. Over the years, these systems have evolved from simple algorithms to complex, machine learning-driven models that analyze user preferences, behavior, and content features to provide tailored recommendations. By personalizing content delivery, platforms like Netflix aim to enhance user satisfaction, boost engagement, and retain subscribers.

2.2 Objectives

The primary objectives of this project are:

- To develop a recommendation system that predicts and suggests relevant content based on user preferences and behavior.
- To implement machine learning techniques such as collaborative filtering and content-based filtering for generating accurate recommendations.
- To integrate user interaction data, such as viewing history and preferences, to improve the personalization of suggestions.
- To evaluate the performance of the recommendation system using key metrics like accuracy, precision, and recall.
- To address challenges such as the cold-start problem and data sparsity by employing a hybrid recommendation approach.

2.3 Purpose

The purpose of this project is to create a system that enhances the viewing experience of users on platforms like Netflix by providing them with personalized content recommendations. The system aims to reduce the time users spend searching for content by offering suggestions that align with their interests and preferences. Additionally, the project seeks to explore and

implement state-of-the-art techniques in machine learning and data analytics to improve recommendation accuracy and system efficiency. By offering a more relevant and engaging content experience, the recommendation system aims to increase user satisfaction and platform retention rates.

2.4 Scope of the Project

The scope of this project includes the design, development, and evaluation of a personalized recommendation system for streaming platforms like Netflix. The project will involve:

- Collecting and pre-processing user interaction data, such as viewing history, ratings, and content metadata.
- Implementing machine learning models, including collaborative filtering, content-based filtering, and hybrid approaches.
- Developing a recommendation engine capable of real-time suggestions based on user preferences.
- Evaluating the recommendation system using performance metrics to ensure accuracy, diversity, and relevance.
- Addressing challenges such as the cold-start problem, data sparsity, and scalability.
- Focusing on movie and TV show recommendations, although the methodology can be extended to other content types.

2.5 Applicability

This recommendation system can be applied to various industries and platforms that offer a large volume of content or products. While the focus of this project is on streaming services like Netflix, the system can be adapted for use in:

- **E-commerce platforms** (e.g., Amazon, Flipkart): Recommending products based on user purchase history and preferences.
- **Music streaming platforms** (e.g., Spotify, Apple Music): Suggesting songs and playlists based on listening history.
- **Social media platforms** (e.g., YouTube, Facebook): Recommending videos, articles, or posts based on user engagement patterns.
- **Online learning platforms** (e.g., Coursera, Udemy): Recommending courses or learning materials based on user interests and completed courses.

Chapter 3

System Planning
Survey of Technologies

3.1.1 Frontend Technologies

The frontend, responsible for the user interface, enables users to interact with the recommendation system, view content, and receive suggestions. The chosen tools for the frontend are:

- **HTML (Hypertext Mark-up Language)**: Used to structure the web pages and display content to the user.
- **CSS (Cascading Style Sheets)**: Applied to style the web pages, ensuring a visually appealing interface with responsive design elements.
- **JavaScript**: Essential for adding interactivity to the web pages, enabling dynamic features such as updating recommendations without reloading the page.

These technologies together provide a responsive, user-friendly interface, making it easier for users to interact with the recommendation system and view personalized content suggestions in real-time.

3.1.2 Backend Technologies

The backend powers the recommendation engine, handling user requests, processing data, and returning relevant suggestions. The primary tools used for backend development include:

- **Python**: As the primary programming language, Python offers robust libraries and frameworks that facilitate data processing, machine learning model integration, and system logic development.
- **Flask 3.0.3**: Flask is a lightweight web framework for building web applications in Python. It is used to manage user requests, serve frontend pages, and handle data communication between the recommendation engine and the user interface.

3.1.3 Development Environment

The development of the Netflix recommendation system requires an efficient and flexible environment for writing, testing, and deploying code. Two key platforms were utilized for this project:

PyCharm 2021.1

PyCharm **i**s an integrated development environment (IDE) specifically designed for Python development. Its robust set of features, such as intelligent code completion, real-time error detection, and seamless integration with version control systems (e.g., Git), makes it ideal for backend development and machine learning model implementation.

Key Features:

- **Virtual Environment Management**: PyCharm supports creating and managing Python virtual environments, which helps isolate project dependencies and ensures compatibility between different Python packages.
- **Debugger**: PyCharm includes an advanced debugger, allowing for real-time monitoring of variable values, breakpoints, and program flow control, which is essential for debugging complex recommendation algorithms.
- **Database Integration**: PyCharm supports built-in database tools, enabling easy access to databases used for storing user data and content information.
- **Flask Integration**: PyCharm allows seamless integration with Flask, simplifying the process of building, testing, and running the web application.

Google Colab

Google Colab (Collaboratory) is a cloud-based platform that enables the execution of Python code in a Jupyter notebook environment. Colab offers powerful GPUs and TPUs, making it highly suitable for developing and training machine learning models without the need for local computational resources.

- **Key Features**:

Cloud-Based: Google Colab allows developers to write and run Python code without local hardware constraints, making it possible to train computationally intensive models, such as deep learning-based recommendation systems, in the cloud.

Pre-installed Libraries: Colab comes with pre-installed libraries such as NumPy, pandas, Scikit-learn, TensorFlow, and PyTorch, facilitating rapid experimentation with various machine learning techniques.

Collaboration: Colab supports real-time collaboration, where multiple users can edit and run code in the same notebook simultaneously, streamlining team-based development.

GPU and TPU Support: By leveraging the cloud-based GPUs and TPUs provided by Colab, developers can accelerate the training and evaluation of large-scale recommendation models.

Data Integration: Colab integrates easily with Google Drive and other cloud storage services, allowing for efficient data management and sharing.

Together, **PyCharm** and **Google Colab** offer a flexible and powerful development environment that supports both backend system design and the computational needs of machine learning model training.

3.2 Languages Used

The development of this recommendation system relies on several programming languages, each fulfilling specific roles:

- **HTML**: For designing the structure and layout of the web interface.
- **CSS**: For enhancing the appearance and responsiveness of the user interface.
- **JavaScript**: For adding interactive features and making dynamic updates to the frontend without reloading the page.

- **Python**: The core language used for developing the backend system, handling data processing, model implementation, and API requests in the Flask framework.

3.3 System Architecture

The system architecture is based on a client-server model, where the frontend interacts with the user and communicates with the backend to retrieve personalized recommendations. The architecture follows these key steps:

1. **User Interaction**: Users access the recommendation system through a web interface. They interact with the platform by browsing content, rating items, and receiving recommendations.
2. **Frontend (Client)**: The frontend, built using HTML, CSS, and JavaScript, displays the personalized content suggestions and allows users to interact with the system.
3. **Backend (Server)**: The backend, developed using Python and Flask, processes user input, retrieves data, and communicates with the recommendation engine to generate suggestions. It uses machine learning models for collaborative and content-based filtering to make predictions.
4. **Database**: The system stores user interaction data (such as viewing history and ratings) and content metadata in a database, which is queried to generate recommendations.
5. **Recommendation Engine**: This engine employs machine learning algorithms to analyse user data and suggest personalized content. The recommendation process includes data collection, pre-processing, filtering, and prediction.
6. **API Communication**: The Flask framework facilitates the communication between the frontend and backend through API endpoints, enabling real-time content suggestions based on user input.

3.4 Fact-Finding Techniques for Netflix Recommendation System

3.3.1 Observations

User Interaction Analysis:

- **Conduct User Sessions**: Observe how users interact with the Netflix platform, focusing on how they discover and select content. Note patterns in their browsing and decision-making processes.

- **Analyze Viewing Habits**: Track viewing patterns, such as the types of content users prefer, their search behavior, and their engagement with recommendations. Identify any common trends or behaviors.
- **Identify Pain Points**: Look for areas where users may experience frustration or difficulties, such as discovering new content, navigating the interface, or finding relevant recommendations.

System Performance:

- **Monitor Current Recommendations**: Observe the effectiveness of the current recommendation system by evaluating user feedback and engagement metrics. Identify any shortcomings or areas for improvement.

- **Evaluate System Load**: Assess how the existing recommendation system handles different loads, such as peak usage times, and note any performance issues or bottlenecks.

3.3.2 Reviewing Existing Documents

System Documentation:

- **Review Technical Specifications**: Gather and review existing documentation on Netflix's current recommendation algorithms, including architectural diagrams, data flow diagrams, and system design documents.

- **Examine Performance Reports**: Analyze reports and logs related to the performance of the existing recommendation system, such as accuracy, user satisfaction, and system efficiency.

User Feedback:

- **Analyze Feedback Data**: Review user feedback, surveys, and ratings related to the recommendations provided. Identify common complaints or suggestions for improvement.
- **Study Case Studies**: Examine case studies or research papers on recommendation systems used by similar platforms to understand best practices and industry standards.

3.3.3 Feasibility Study

1. Technical Feasibility:

- **Infrastructure Requirements**: Evaluate the technical infrastructure needed to develop and maintain the recommendation system. Assess the availability of necessary hardware, software, and cloud services for processing and storing data.
- **Technology Stack**: Determine if the project can be implemented using existing technologies such as Python, machine learning libraries (e.g., TensorFlow, Scikit-learn), and cloud platforms. Identify any technical challenges or limitations that may impact development and deployment.
- **Algorithm Suitability**: Assess whether collaborative filtering, content-based filtering, or hybrid models are feasible and appropriate for the project's goals. Ensure that the chosen algorithms can be effectively integrated into the existing system.

2. Economic Feasibility:

a) **Cost Analysis**: Estimate the initial development costs, including expenses for machine learning model training, data acquisition, cloud services, and development tools.
b) **Cost-Benefit Evaluation**: Evaluate the potential cost savings and revenue generation opportunities associated with implementing the recommendation system. Consider the impact on user engagement, subscription renewals, and advertising revenue.
c) **Long-Term Costs**: Assess ongoing operational expenses such as system maintenance, updates, and support. Compare these costs with the projected benefits and returns on investment.

3.Operational Feasibility:

a) **Integration with Existing Systems**: Evaluate how the recommendation system will integrate with Netflix's current platform and workflows. Consider compatibility with existing databases, user interfaces, and content management systems.

b) **Stakeholder Impact**: Assess the impact on stakeholders, including users, content providers, and internal teams (e.g., data scientists, developers). Identify any training or support needs for effective system adoption.
c) **Organizational Readiness**: Identify any organizational or cultural barriers that may affect the adoption and acceptance of the recommendation system. Ensure that the system aligns with organizational goals and user expectations.

4. Schedule Feasibility:

a) **Project Timeline**: Develop a realistic timeline for the project, including stages such as requirements gathering, algorithm development, data preparation, testing, and deployment.
b) **Dependencies and Constraints**: Consider any external dependencies or constraints that may affect the project timeline, such as data availability, technology integration, or regulatory requirements.
c) **Resource Allocation**: Allocate resources effectively, including personnel, computing power, and budget, to ensure timely completion of project milestones.
d) **Progress Monitoring**: Regularly monitor progress and adjust the schedule as needed to address any unforeseen challenges or delays.

3.4 Stakeholders

Admin:

Role: Admins manage and oversee the recommendation system's data, algorithms, and overall performance. They ensure that the system is functioning as expected and handle any issues that arise.

Responsibilities: Admins control data inputs, monitor system outputs, and maintain data integrity. They also address user feedback and make necessary adjustments to the system.

Developers:

Role: Developers are responsible for building, maintaining, and enhancing the recommendation system. They work on integrating algorithms, optimizing performance, and implementing new features.

Responsibilities: Developers address technical issues, implement updates, and ensure that the recommendation system aligns with user needs and business goals.

Data Scientists:

Role: Data scientists design and train the machine learning models used in the recommendation system. They analyze data, evaluate model performance, and refine algorithms.

Responsibilities: Data scientists work on feature engineering, model selection, and evaluation metrics. They ensure that the recommendations are accurate and relevant.

Users:

Role: End-users interact with the recommendation system and provide feedback on its effectiveness. They influence the system's success through their engagement and satisfaction.

Responsibilities: Users provide input on their preferences and behaviors, which helps to fine-tune the recommendation algorithms and improve overall user experience.

Content Providers:

Role: Content providers supply the media content that is recommended by the system. Their content's visibility and engagement are affected by the recommendation algorithms.

Responsibilities: Content providers ensure their content is accurately represented and promoted through the recommendation system, contributing to its effectiveness and relevance.

Marketing and Sales Teams:

Role: Marketing and sales teams use insights from the recommendation system to develop promotional strategies and enhance user engagement.

Responsibilities: They leverage recommendation data to drive marketing campaigns, increase subscription rates, and optimize ad placements.

Chapter 4

Approach & Methodology

Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) focused on creating algorithms and models that enable computers to learn from and make predictions based on data. The goal of machine learning is to develop systems that can learn and adapt without being explicitly programmed for each specific task.

Key Concepts in Machine Learning

- **Supervised Learning**: In supervised learning, the algorithm is trained on a dataset that includes input-output pairs. The model learns to map inputs to outputs based on this labeled data. Common tasks include classification (e.g., predicting whether a user will like a movie) and regression (e.g., predicting a movie's rating).

- **Unsupervised Learning**: This approach involves training algorithms on data without labeled responses. The model tries to identify patterns or groupings in the data. Common tasks include clustering (e.g., grouping similar movies) and dimensionality reduction (e.g., reducing the number of features while retaining important information).

- **Semi-Supervised Learning**: Combining both labeled and unlabeled data, semi-supervised learning improves the model's performance when labeled data is scarce or expensive to obtain. It leverages the structure in unlabeled data to enhance learning.

- **Reinforcement Learning**: This type of learning involves an agent making decisions in an environment to maximize cumulative rewards over time. It is useful for systems that learn to make sequences of decisions, such as optimizing recommendation strategies based on user interactions.

- **Deep Learning**: A subset of machine learning focusing on neural networks with multiple layers (deep neural networks). Deep learning excels in areas such as image recognition, natural language processing, and complex recommendation systems.

Steps in the Machine Learning Process

1. **Data Collection**: Gather relevant data from various sources such as user interaction logs, content metadata, and user profiles to build a comprehensive dataset for training the recommendation system.
2. **Data Pre-processing**: Clean and transform the raw data to make it suitable for modeling. This includes handling missing values, normalizing data, and encoding categorical variables.
3. **Feature Engineering**: Develop and select features that will enhance the model's performance. This may involve creating new features based on user behavior, content characteristics, or interaction patterns.
4. **Model Selection and Training**: Choose appropriate machine learning algorithms for the recommendation task, such as collaborative filtering, content-based filtering, or hybrid methods. Train the model on the prepared dataset to learn patterns and make predictions.
5. **Evaluation and Validation**: Assess the model's performance using metrics such as precision, recall, and F1 score. Validate the model with a separate dataset to ensure it generalizes well to unseen data.
6. **Hyperparameter Tuning**: Optimize model performance by adjusting hyperparameters, such as the learning rate or the number of latent factors in collaborative filtering. Use techniques like grid search or random search to find the best configuration.
7. **Deployment and Monitoring**: Deploy the recommendation system into a production environment where it can make real-time recommendations. Monitor the system's performance and user feedback to ensure it continues to provide accurate and relevant suggestions.

Applications of Machine Learning

- **Recommendation Systems**: Personalized suggestions for users based on their preferences and behavior.
- **Predictive Analytics**: Forecasting user engagement, content popularity, and other metrics.
- **Natural Language Processing**: Understanding and processing user reviews and feedback.
- **User Profiling**: Creating detailed profiles to enhance recommendation accuracy.

Machine learning drives innovation in recommendation systems, helping platforms like Netflix deliver personalized and engaging content to users.

4.2 Methodology

1. **Requirement Gathering**: Understand stakeholder needs through surveys, interviews, and analysis of existing recommendation systems. Identify key functional requirements (e.g., recommendation accuracy, real-time processing) and non-functional requirements (e.g., system performance, scalability) to guide the development process.

2. **Data Collection and Preparation**: Collect data from user interactions, content metadata, and user profiles. Preprocess the data to handle missing values, normalize features, and encode categorical variables. This step ensures the dataset is ready for model training.

3. **Model Development**:

- **Collaborative Filtering**: Implement collaborative filtering techniques to recommend content based on user similarity or item similarity. Use matrix factorization methods such as Singular Value Decomposition (SVD) or Alternating Least Squares (ALS) for generating recommendations.
- **Content-Based Filtering**: Develop content-based models that recommend items similar to those a user has liked in the past. Utilize features such as genre, actors, or keywords.
- **Hybrid Methods**: Combine collaborative and content-based approaches to leverage the strengths of both methods and improve recommendation accuracy.

4. **Model Training and Evaluation**: Train the recommendation models using historical data and evaluate their performance using metrics such as precision, recall, and F1 score. Perform cross-validation to ensure robustness and generalizability.

5. **Integration and Testing**:

- **System Integration**: Integrate the recommendation engine with the Netflix platform, ensuring that it interfaces smoothly with the existing user interface and backend systems.
- **Testing**: Conduct extensive testing to verify the functionality, performance, and accuracy of the recommendation system. Perform A/B testing to compare different recommendation strategies and select the most effective one.

6. **Deployment**: Deploy the recommendation system to the production environment, enabling real-time recommendations for users. Ensure that the deployment process includes rollback plans in case of issues.

7. **Monitoring and Maintenance**: Continuously monitor the system's performance, user feedback, and accuracy. Update the model periodically to adapt to changing user preferences and content trends. Implement a feedback loop to incorporate user insights and improve recommendation quality over time.

By following this approach and methodology, the Netflix recommendation system aims to deliver a personalized and engaging user experience, enhancing content discovery and user satisfaction

Chapter 5
Requirement & Analysis

5.1 Problem Definition

The Netflix recommendation system aims to address the challenge of content discovery for users in a vast and diverse streaming library. With an overwhelming number of options available, users often struggle to find relevant content that matches their preferences and interests. The system's goal is to enhance user experience by providing personalized, accurate recommendations based on users' viewing history, preferences, and behavior. By leveraging advanced machine learning algorithms, the recommendation system will streamline the content discovery process, reduce decision fatigue, and ensure that users have access to engaging and relevant content.

5.2 Requirement Specification

The system is designed to meet the following requirements:

- **Personalized Recommendations**: Provide tailored content suggestions based on users' viewing history, preferences, and behavior.
- **User Profiles**: Maintain detailed user profiles that capture preferences, watch history, and ratings.
- **Content Metadata**: Utilize rich content metadata (e.g., genres, actors, directors) to enhance recommendation accuracy.
- **Real-Time Updates**: Update recommendations in real-time as users interact with the platform.
- **Scalability**: Handle large volumes of data and user interactions efficiently.
- **Performance Metrics**: Monitor the effectiveness of recommendations using metrics such as click-through rate (CTR), user engagement, and satisfaction.
- **Integration**: Seamlessly integrate with Netflix's existing infrastructure and user interface.
- **Data Privacy**: Ensure user data is handled securely and in compliance with privacy regulations.

- **Admin Functionality**: Allow administrators to manage and analyze recommendation algorithms, view system performance, and make adjustments as needed.

5.3 Modules of Proposed System

Client-Side Functionalities

- **User Authentication**: Allows users to create accounts, log in securely, and access personalized recommendations.
- **Recommendation Display**: Shows personalized content suggestions based on user profiles and viewing history.
- **Profile Management**: Enables users to update their preferences, view watch history, and rate content.
- **Search and Filter**: Allows users to search for specific content and apply filters based on genres, actors, etc.
- **Feedback Mechanism**: Provides users with options to give feedback on recommendations to improve system accuracy.
- **Logout**: Allows users to securely log out of the system to protect their accounts.

Admin-Side Functionalities

- **Algorithm Management**: Allows administrators to configure and update recommendation algorithms and models.
- **Performance Monitoring**: Provides insights into the performance of recommendation algorithms using metrics like CTR and user engagement.
- **Data Management**: Manages user data, content metadata, and feedback to improve recommendation quality.
- **System Analysis**: Offers tools to analyze system performance, identify issues, and optimize recommendations.
- **Logout**: Allows administrators to securely log out of the system to protect their accounts.

5.4 Software and Hardware Requirements

5.4.1 Developer Side

Hardware Requirements:

- ⇒ **Processor**: Minimum 2.0 GHz (higher preferred)
- ⇒ **Memory**: 8GB RAM or more
- ⇒ **Storage**: Sufficient to handle development tools and datasets

Software Requirements:

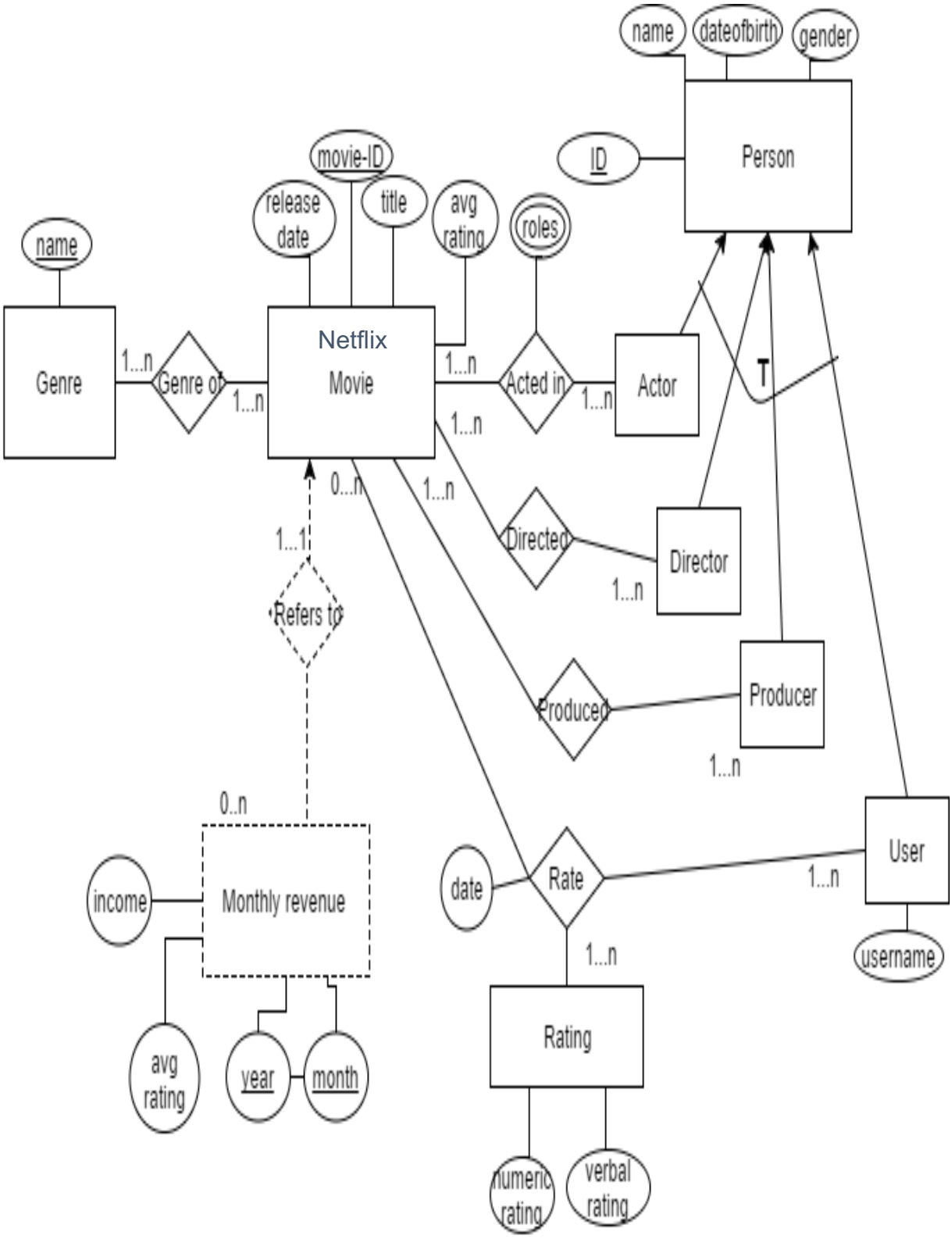
- ⇒ **Development Environment**: PyCharm, Jupyter Notebook, Google Colab
- ⇒ **Programming Languages**: Python, SQL
- ⇒ **Libraries and Frameworks**: TensorFlow, Scikit-learn, Flask
- ⇒ **Operating System**: Windows 10 or higher, macOS, Linux

5.4.2 User Side

- ⇒ **Hardware Requirements**: Laptop or desktop with a processor of 1.5 GHz or higher and 4GB RAM or more.
- ⇒ **Software Requirements**: Web browser (Google Chrome, Mozilla Firefox, or Microsoft Edge)
- ⇒ **Internet Connectivity**: Required for accessing the Netflix platform and receiving recommendations.

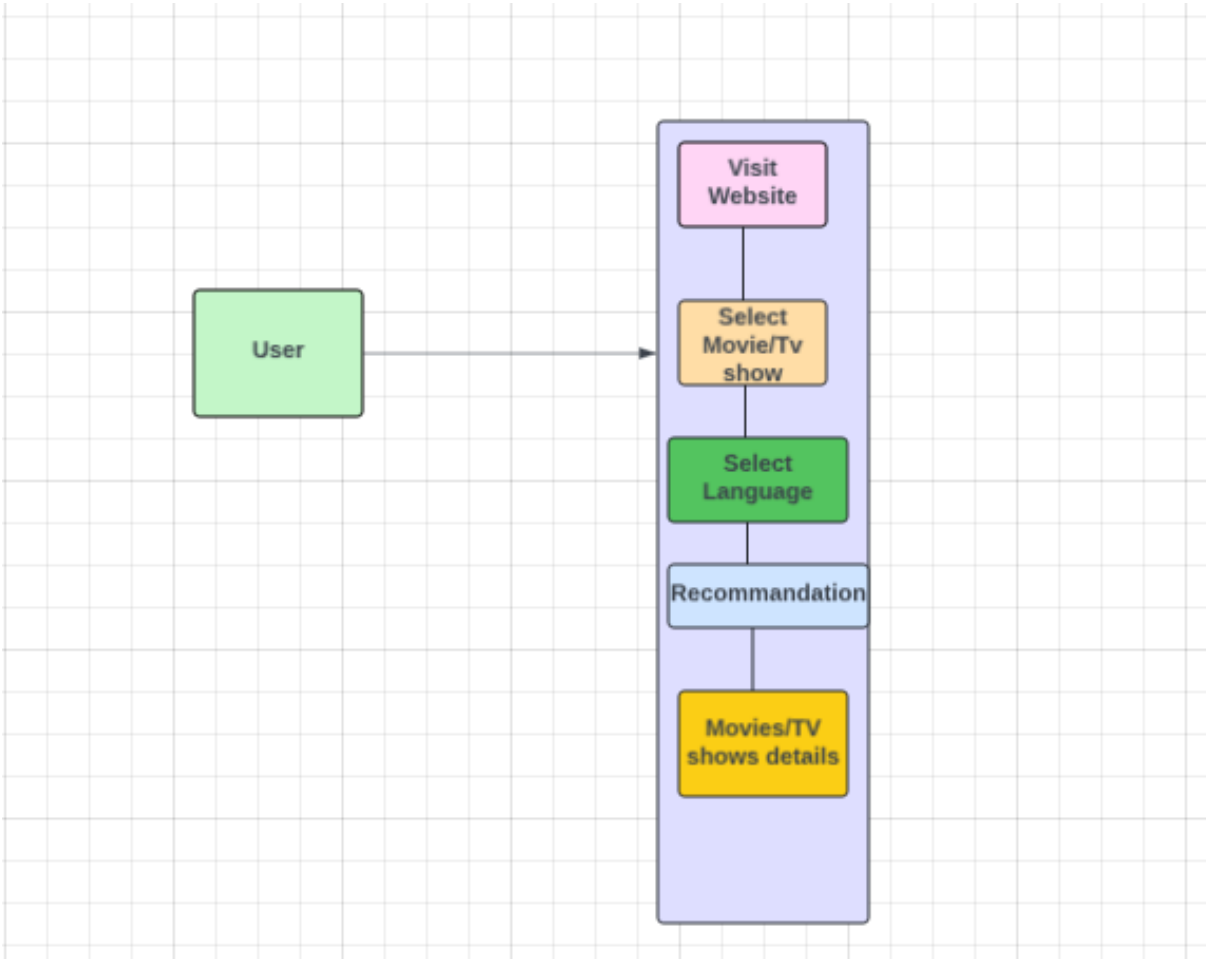
5.5 Conceptual Models

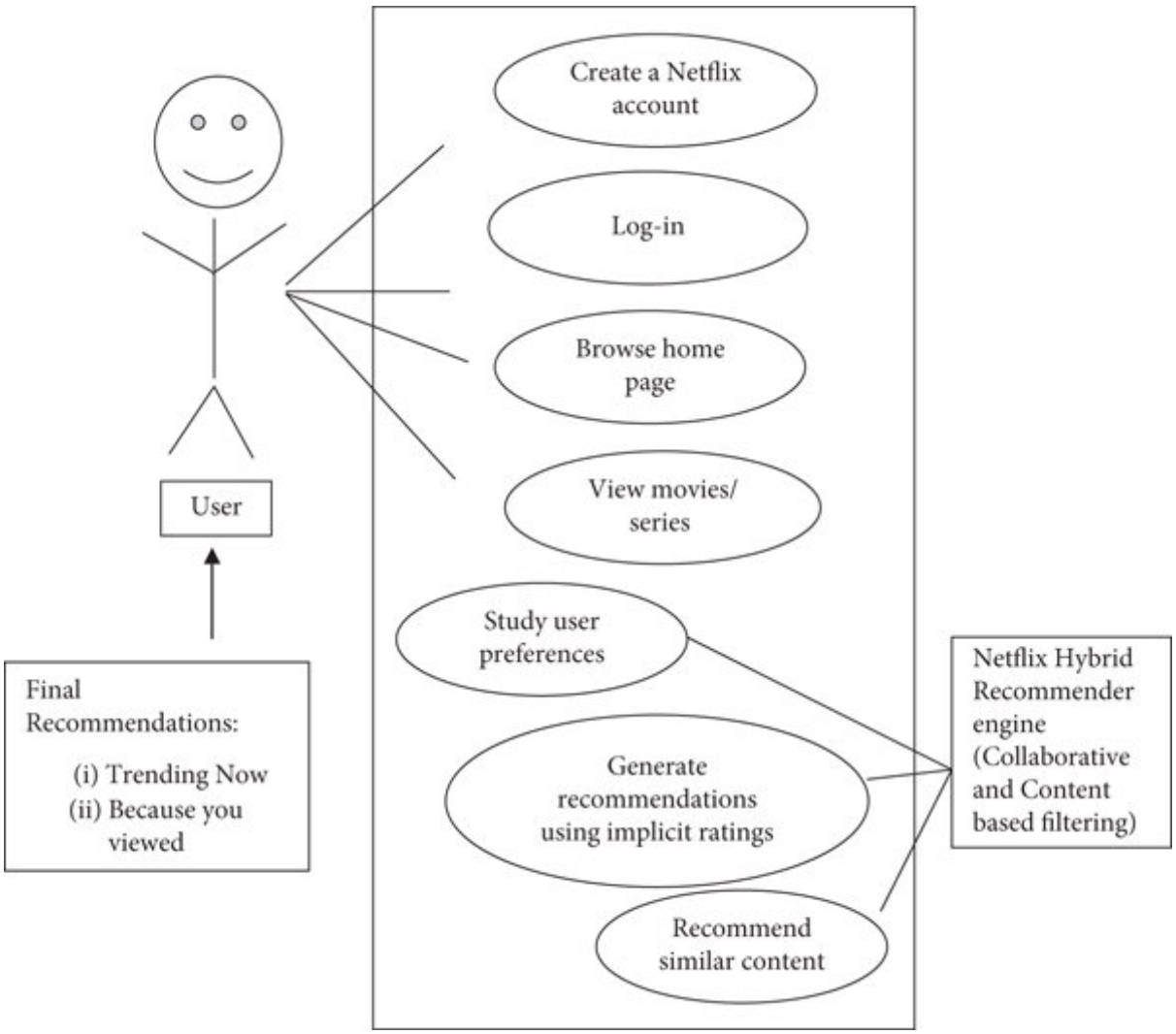
ER DIAGRAM


name
dateofbirth
gender
Person
ID
movie-ID
release date
title
avg rating
roles
name
Genre
1...n
Genre of
1...n
Netflix
Movie
1...n
Acted in
1...n
Actor
1...n
0...n
1...n
1...1
Refers to
Directed
1...n
Director
Produced
Producer
1...n
0..n
income
Monthly revenue
avg rating
year
month
date
Rate
1...n
User
1...n
username
Rating
numeric rating
verbal rating

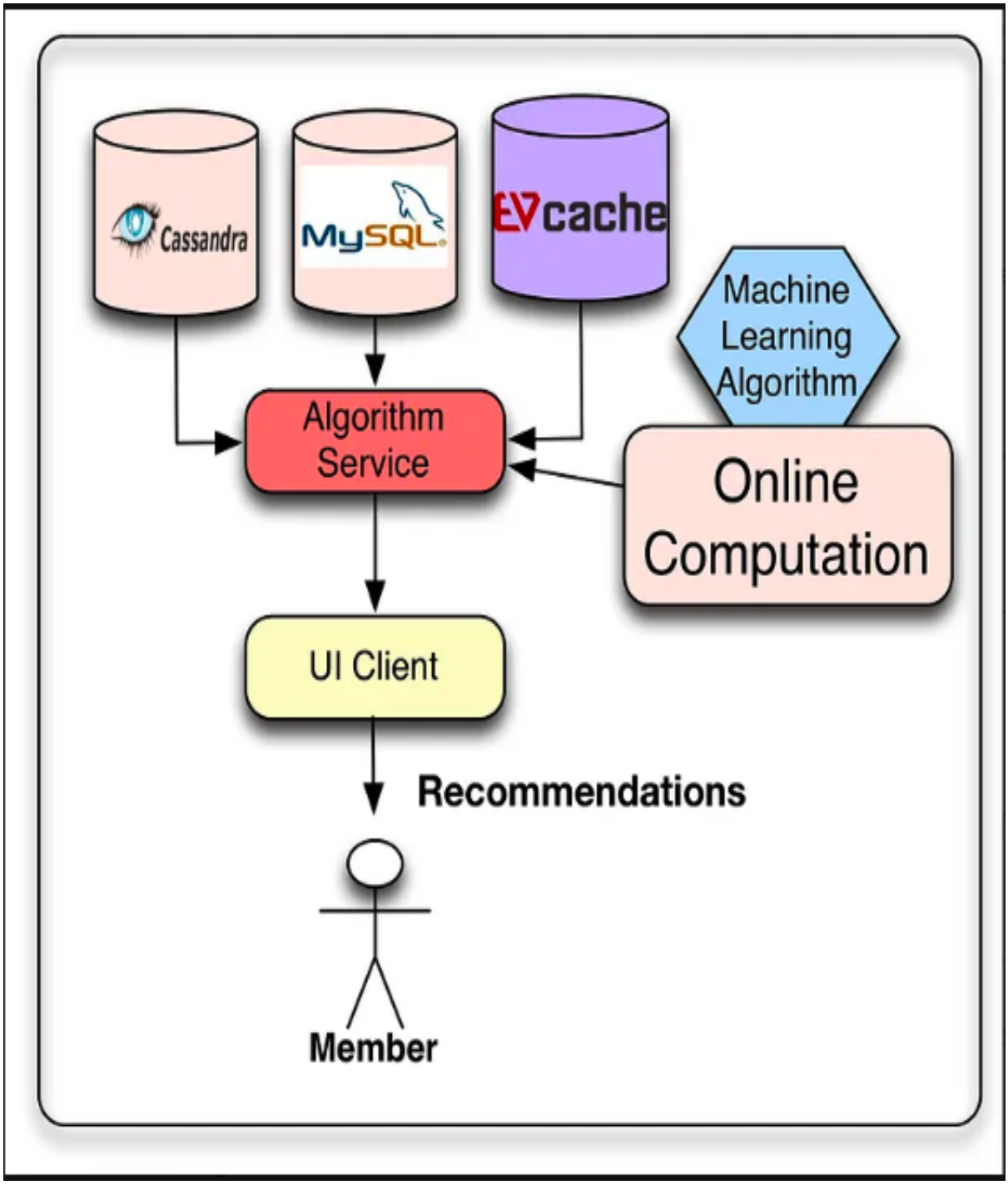
Use Case Diagram

User Use Case




Create a Netflix account
Log-in
Browse home page
View movies/ series
Study user preferences
Generate recommendations using implicit ratings
Recommend similar content
Netflix Hybrid Recommender engine (Collaborative and Content based filtering)
User
Final Recommendations:
(i) Trending Now
(ii) Because you viewed

System Architecture Diagram



Chapter 6
System Design

6.1 Data Integrity and Constraints

Component	Description	Technology
Frontend	Web interface where users can select preferences, view recommendations, and see details.	HTML, CSS, JavaScript, React/Angular/Vue.js
API Server	Handles requests from the frontend, communicates with the Recommendation and Details Services.	Flask/Django (Python), Express.js (Node.js)
Recommendation Service	Generates movie/TV show recommendations based on user preferences.	Custom algorithms, ML models
Details Service	Provides detailed information about a selected movie/TV show.	Queries Movie/TV Show Database or external APIs
User Preferences Database	Stores user preferences and interaction history.	PostgreSQL/MySQL, MongoDB
Movie/TV Show Database	Stores details of movies/TV shows, including metadata and language options.	PostgreSQL/MySQL, MongoDB
Recommendation Engine Database	Stores precomputed recommendations or models for generating recommendations.	PostgreSQL/MySQL, MongoDB
External Services	APIs for fetching mov… TV show data.	TMDb API, IMDb API

Workflow

Step	Description	Component(s) Involved
1. Preference Selection	User selects movie/TV show genre and language preferences.	Frontend
2. Submit Preferences	Frontend sends user preferences to the API server.	Frontend → API Server
3. Generate Recommendations	API Server forwards preferences to Recommendation Service.	API Server → Recommendation Service
4. Return Recommendations	Recommendation Service retrieves and sends recommendations back to the API Server.	Recommendation Service → API Server
5. Display Recommendations	API Server sends recommendations to the Frontend.	API Server → Frontend
6. Select Movie/TV Show	User selects a movie/TV show from the recommendations.	Frontend
7. Fetch Details	Frontend sends the selected movie/TV show to the API Server.	Frontend → API Server
8. Retrieve Details	API Server forwards request to Details Service.	API Server → Details Service
9. Return Details	Details Service retrieves detailed information and sends it back to the API Server.	Details Service → API Server
10. Display Details	API Server sends detailed information to the Frontend.	API Server → Frontend

Chapter 7
7. Machine Learning Code

```
!pip install plotly_express
#importing libraries
import math
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly_express as px
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
Requirement already satisfied: plotly_express in /usr/local/lib/python3.10/dist-packages (0.4.1)
Requirement already satisfied: pandas>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from plotly_express) (2.1.4)
Requirement already satisfied: plotly>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from plotly_express) (5.15.0)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from plotly_express) (0.14.2)
Requirement already satisfied: scipy>=0.18 in /usr/local/lib/python3.10/dist-packages (from plotly_express) (1.13.1)
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.10/dist-packages (from plotly_express) (0.5.6)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.10/dist-packages (from plotly_express) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->plotly_exp
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->plotly_express) (202
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->plotly_express) (2
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5->plotly_express) (1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=4.1.0->plotly_express) (9
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=4.1.0->plotly_express) (24.1)
```

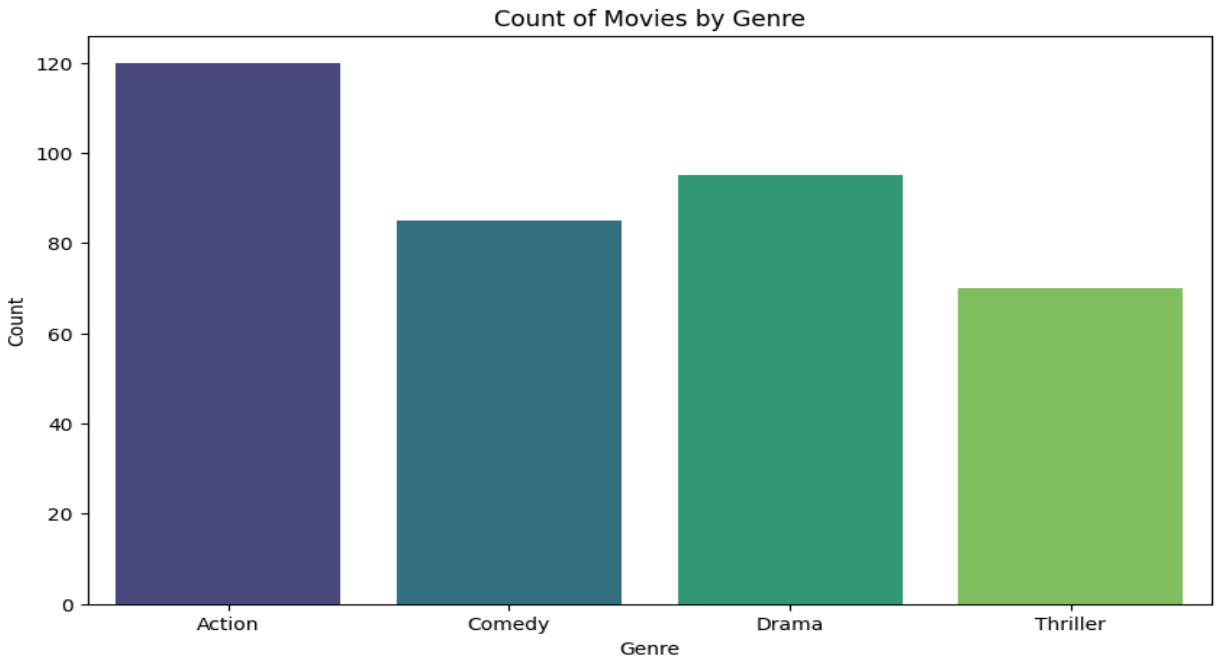
```
netflix_data = pd.read_csv('NetflixDataset.csv',encoding='latin-1',
index_col = 'Title')
netflix_data.head(2)
```

Title												
Lets Fight Ghost	Crime, Drama, Fantasy, Horror, Romance	Comedy Programmes,Romantic TV Comedies,Horror ...	Swedish, Spanish	Thailand	< 30 minutes	Tomas Alfredson	John Ajvide Lindqvist	Lina Leandersson, Kåre Hedebrant, Per Ragnar, ...	R	7.9	...	57.0 $2
HOW TO BUILD A GIRL	Comedy	Dramas,Comedies,Films Based on Books,British	English	Canada	1-2 hour	Coky Giedroyc	Caitlin Moran	Cleo, Paddy Considine, Beanie Feldstein, Dónal...	R	5.8	...	NaN

```
# Display summary statistics for numerical columns
print(netflix_data.describe())
```

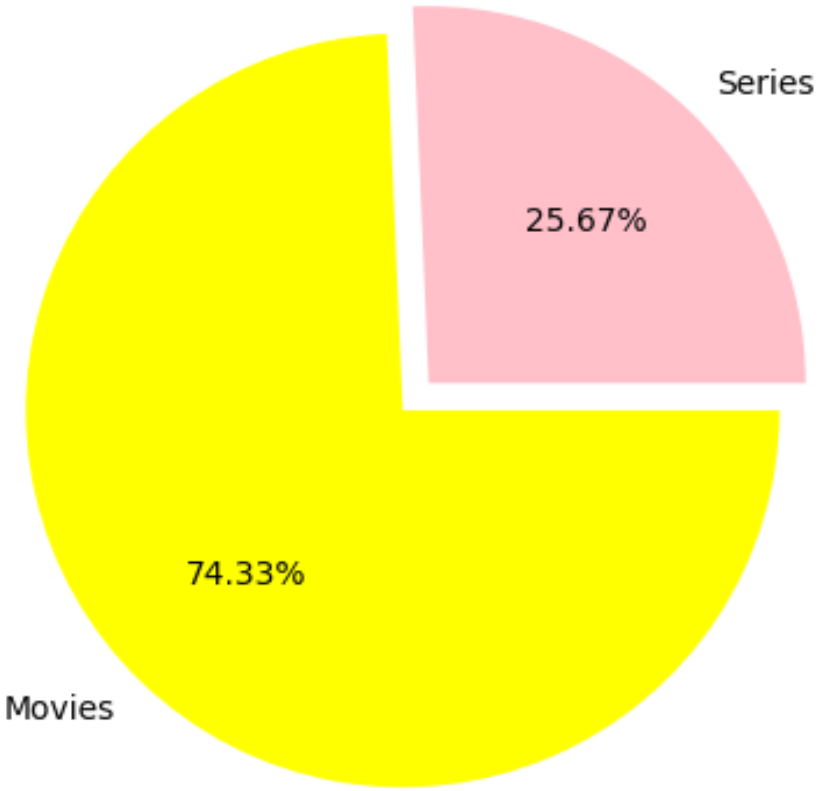
```
       IMDb Score  Awards Received  Awards Nominated For    IMDb Votes
count  9395.000000      5213.000000           6363.000000  9.393000e+03
mean      6.954902         9.725878             16.024674  5.992114e+04
std       0.899448        19.526363             32.215336  1.456879e+05
min       1.600000         1.000000              1.000000  5.000000e+00
25%       6.500000         1.000000              2.000000  9.740000e+02
50%       7.000000         4.000000              6.000000  6.602000e+03
75%       7.500000         9.000000             15.000000  5.096900e+04
max       9.700000       300.000000            386.000000  2.354197e+06
```

```
import seaborn as sns

# Sample DataFrame
data = {
    'Genre': ['Action', 'Comedy', 'Drama', 'Thriller'],
    'Count': [120, 85, 95, 70]
}
df = pd.DataFrame(data)

# Bar Plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Genre', y='Count', data=df, palette='viridis')
plt.xlabel('Genre')
plt.ylabel('Count')
plt.title('Count of Movies by Genre')
plt.show()
```




```
netflix_data.index = netflix_data.index.str.title()
color = ['pink', 'yellow']
label = ['Series', 'Movies']
sizes = [netflix_data[netflix_data['Series or Movie'] ==
'Series'].size, netflix_data[netflix_data['Series or Movie'] ==
'Movie'].size]
explode = (0.1, 0)
fig, ax = plt.subplots()
ax.pie(sizes, explode, label, color, '%2.2f%%')
ax.axis('equal')
```

```
plt.show()
```



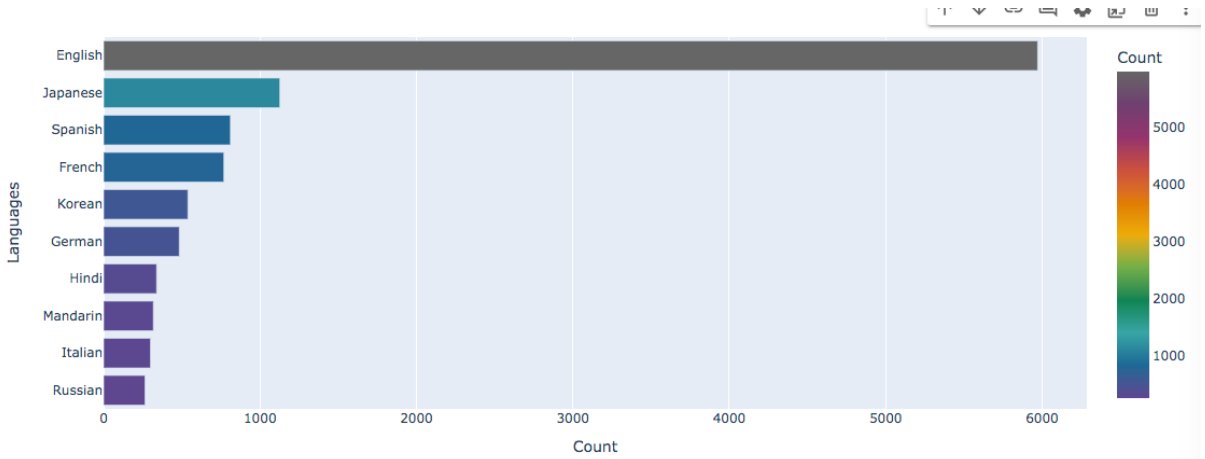

```
netflix_data.rename(columns={'View Rating':'ViewerRating'},
inplace=True)
```

```
Language = netflix_data.Languages.str.get_dummies(',')
Lang = Language.columns.str.strip().values.tolist()
Language = netflix_data['Languages']
Language_Count = dict()
for i in Lang:
    p = Language.str.count(i).sum()
    Language_Count[i] = int(p)
print(len(Language_Count))
```

184

```
Language_Count = {k: v for k, v in sorted(Language_Count.items(),
key=lambda item: item[1], reverse = True)}
top_languages = {"Languages": list(Language_Count.keys()), "Count":
list(Language_Count.values())}
```

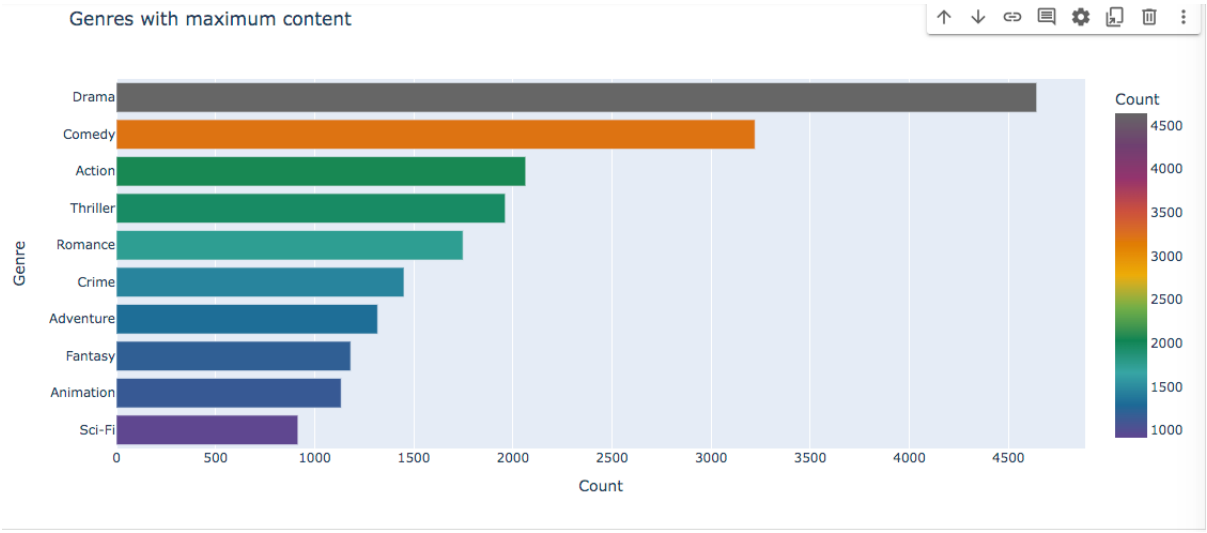
```
fig = px.bar(pd.DataFrame(top_languages)[:10], y = 'Languages', x =
'Count', orientation = 'h', title = 'Most Available Languages', color =
'Count', color_continuous_scale =
px.colors.qualitative.Prism).update_yaxes(categoryorder = 'total
ascending')
fig.show()
```



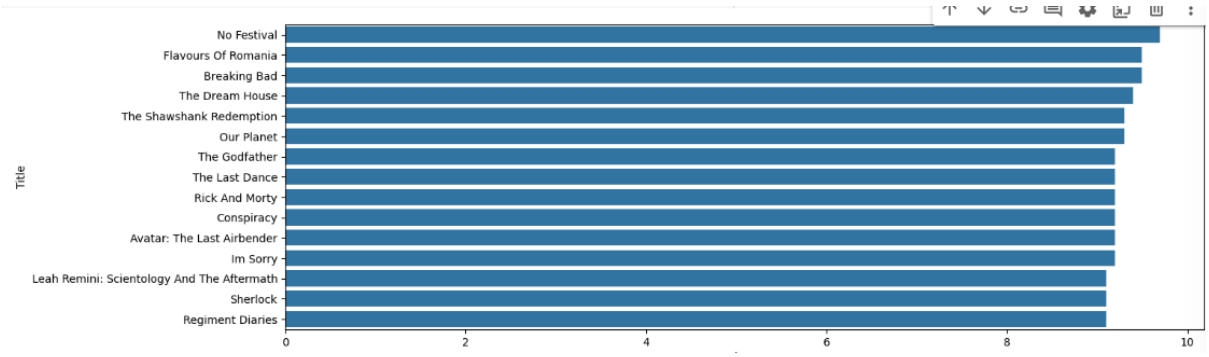

```
Genres = netflix_data.Genre.str.get_dummies(',')
Genre = Genres.columns.str.strip().values.tolist()
Genres = netflix_data['Genre']
Genre_Count = dict()
for i in Genre:
    p = Genres.str.count(i).sum()
    Genre_Count[i] = int(p)
print(len(Genre_Count))
```

29

```
Genre_Count = {k: v for k, v in sorted(Genre_Count.items(), key=lambda
item: item[1], reverse = True)}
top_genres = {"Genre": list(Genre_Count.keys()), "Count":
list(Genre_Count.values())}
fig = px.bar(pd.DataFrame(top_genres)[:10], y = 'Genre', x = 'Count',
orientation = 'h', title = 'Genres with maximum content', color =
'Count', color_continuous_scale =
px.colors.qualitative.Prism).update_yaxes(categoryorder = 'total
ascending')
fig.show()
```




```
top_15 = netflix_data.sort_values(by = ['IMDb Score'], ascending =
False).head(15)
plt.figure(figsize = (15,5))
sns.barplot(data = top_15, y = top_15.index, x = "IMDb Score")
plt.show()
```




```
netflix_data = netflix_data[~netflix_data.index.duplicated()]
```

```
netflix_data.index.duplicated().sum()
```

0

```
netflix_data['Genre'] = netflix_data['Genre'].astype('str')
print((netflix_data['Genre'] == 'nan').sum())
```

25

```
netflix_data['Tags'] = netflix_data['Tags'].astype('str')
print((netflix_data['Tags'] == 'nan').sum())
```

36

```
print(netflix_data[['IMDb Score']].describe())
```

```
netflix_data['IMDb Score'].mode()
#this feature will be used to sort the movie or series list to
represent the recommended items
```

	IMDb Score
count	9132.000000
mean	6.954862
std	0.896212
min	1.600000
25%	6.500000
50%	7.000000
75%	7.500000
max	9.700000

	IMDb Score
0	6.6

dtype: float64

```
netflix_data['IMDb Score'] = netflix_data['IMDb Score'].apply(lambda x:
6.6 if x == 0 or math.isnan(x) else x)
print(netflix_data[['IMDb Score']].describe())
#since no value has suffered for change greater than 0.0003 after
replacing the null values with mode value, so we replace the null
values with 6.6
```

	IMDb Score
count	9132.000000
mean	6.954862
std	0.896212
min	1.600000
25%	6.500000
50%	7.000000
75%	7.500000
max	9.700000

```
netflix_data['Actors'] = netflix_data['Actors'].astype('str')
netflix_data['ViewerRating'] =
netflix_data['ViewerRating'].astype('str')
```

```
def prepare_data(x):
        return str.lower(x.replace(" ", ""))
new_features = ['Genre', 'Tags', 'Actors', 'ViewerRating']
selected_data = netflix_data[new_features]

for new_feature in new_features:
    selected_data.loc[:, new_feature] = selected_data.loc[:,
new_feature].apply(prepare_data)
selected_data.index = selected_data.index.str.lower()
selected_data.index = selected_data.index.str.replace(" ",'')
selected_data.head(2)
```

Title	Genre	Tags	Actors	ViewerRating
tsfightghost	crime,drama,fantasy,horror,romance	comedyprogrammes,romantictvcomedies,horrorprog...	linaleandersson,kårehedebrant,perragnar,henrik...	r
wtobuildagirl	comedy	dramas,comedies,filmsbasedonbooks,british	cleo,paddyconsidine,beaniefeldstein,dónalfinn	r

```
def create_soup(x):
    return x['Genre'] + ' ' + x['Tags'] + ' ' +x['Actors']+' '+
x['ViewerRating']
```

```
selected_data.loc[:, 'soup'] = selected_data.apply(create_soup, axis =
1)
selected_data.head(2)
```

```
<ipython-input-64-1b58fdfef02e>:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Title	Genre	Tags	Actors	ViewerRating	
letsfightghost	crime,drama,fantasy,horror,romance	comedyprogrammes,romantictvcomedies,horrorprog...	linaleandersson,kårehedebrant,perragnar,henrik...	r	crime,drama
howtobuildagirl	comedy	dramas,comedies,filmsbasedonbooks,british	cleo,paddyconsidine,beaniefeldstein,dónalfinn	r	dramas,comedies,fi

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(selected_data['soup'])
cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

```
selected_data.reset_index(inplace = True)
selected_data.head(2)
```

	Title	Genre	Tags	Actors	ViewerRati
0	letsfightghost	crime,drama,fantasy,horror,romance	comedyprogrammes,romantictvcomedies,horrorprog...	linaleandersson,kårehedebrant,perragnar,henrik...	
1	howtobuildagirl	comedy	dramas,comedies,filmsbasedonbooks,british	cleo,paddyconsidine,beaniefeldstein,dónalfinn	

```
indices = pd.Series(selected_data.index, index=selected_data['Title'])
indices
```

Title	0
letsfightghost	0
howtobuildagirl	1
thecon-heartist	2
glebokawoda	3
onlyamother	4
snowroller	5
theinvisible	6
thesimplemindedmurderer	7
tokillachild	8
joker	9
i	10
harrysdaughters	11
qyllenetider	12

```
result = 0
def get_recommendations(title, cosine_sim):
    global result
    title=title.replace(' ','').lower()
    idx = indices[title]
    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # Get the scores of the 50 most similar movies
    sim_scores = sim_scores[1:51]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]
    # Return the top 10 most similar movies
    result =  netflix_data.iloc[movie_indices]
    result.reset_index(inplace = True)
```

```
    return result
```

```
df = pd.DataFrame()
movienames = ['Annabelle Comes Home','The Nun', 'Insidious: The Last
Key', 'Conjuring 2', 'Insidious: Chapter 3']
languages = ['English', 'Hindi']
for moviename in movienames:
    get_recommendations(moviename,cosine_sim2)
    for language in languages:
        df = pd.concat([result[result['Languages'].str.count(language)
> 0], df], ignore_index=True)
df.drop_duplicates(keep = 'first', inplace = True)
df.sort_values(by = 'IMDb Score', ascending = False, inplace = True)
```

```
print(df.shape)
print(df.head())
```

```
(118, 22)
           Title                            Genre  \
114     The Others         Horror, Mystery, Thriller
50   The Conjuring         Horror, Mystery, Thriller
169     Hereditary  Drama, Horror, Mystery, Thriller
105          Split                  Horror, Thriller
201    Conjuring 2         Horror, Mystery, Thriller

                                          Tags         Languages  \
114  20th Century Period Pieces,Thrillers,Mysteries...           English
50   Horror Films,Thrillers,Supernatural Horror Fil...    English, Latin
169  Supernatural Horror Movies,Horror Movies,Teen ...  English, Spanish
105  Psychological Thrillers,Horror Movies,Thriller...           English
201  Horror Films,Supernatural Horror Films,Films B...           English

                              Country Availability   Runtime  \
114  Italy,Sweden,Switzerland,Turkey,Iceland,India,...  1-2 hour
50   France,Belgium,Lithuania,Switzerland,United Ki...  1-2 hour
169     South Korea,United Kingdom,Canada,Japan,India   > 2 hrs
105  Belgium,Netherlands,South Korea,Poland,France,...  1-2 hour
201  South Korea,Switzerland,Brazil,Germany,Italy,A...   > 2 hrs

             Director                                          Writer  \
114  Alejandro Amenábar                              Alejandro Amenábar
50          James Wan                        Chad Hayes, Carey W. Hayes
169         Ari Aster                                       Ari Aster
105  M. Night Shyamalan                              M. Night Shyamalan
201         James Wan  Chad Hayes, David Leslie Johnson-McGoldrick, J...

                                          Actors ViewerRating  ...  \
114  Alakina Mann, Christopher Eccleston, Fionnula ...        PG-13  ...
50   Vera Farmiga, Lili Taylor, Ron Livingston, Pat...            R  ...
169  Alex Wolff, Milly Shapiro, Toni Collette, Gabr...            R  ...
105  Haley Lu Richardson, James McAvoy, Anya Taylor...        PG-13  ...
201  Frances O'Connor, Vera Farmiga, Madison Wolfe,...            R  ...

     Awards Nominated For      Boxoffice  Release Date Netflix Release Date  \
114                  55.0   $9,65,22,687     10-Aug-01           14-04-2015
50                   22.0  $13,74,00,141     19-Jul-13           14-04-2015
169                 105.0   $4,40,69,456     08-Jun-18           15-02-2019
105                  23.0  $13,82,91,365     20-Jan-17           17-09-2018
201                  13.0  $10,24,70,008     10-Jun-16           28-03-2017

                                Production House  \
114  Lucky Red, Miramax Films, Sogecine, Cruise-Wag...
50             Safran Company, Evergreen Media Group
```

	Netflix Link	\
114	https://www.netflix.com/watch/60021097	
50	https://www.netflix.com/watch/70251894	
169	https://www.netflix.com/watch/80238910	
105	https://www.netflix.com/watch/80124506	
201	https://www.netflix.com/watch/80091246	

	Summary	Series or Movie	\
114	While awaiting the return of her soldier husba...	Movie	
50	Based on true events, this spine-chiller tells...	Movie	
169	After the death of her mother, artist Annie an...	Movie	
105	A man with multiple identities abducts three t...	Movie	
201	A single mother and her four kids in London ar...	Movie	

	IMDb Votes	Image
114	338889.0	https://occ-0-3377-185.1.nflxso.net/dnm/api/v6...
50	443192.0	https://occ-0-2851-38.1.nflxso.net/dnm/api/v6/...
169	241835.0	https://occ-0-1926-41.1.nflxso.net/dnm/api/v6/...
105	428760.0	https://occ-0-768-769.1.nflxso.net/dnm/api/v6/...
201	226713.0	https://occ-0-2773-2774.1.nflxso.net/dnm/api/v...

[5 rows x 22 columns]

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Netflix Recommendation System</title>
   <link rel="stylesheet" href="../static/stylesheets/style.css" />
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></sc
ript>
   <script src="../static/js/multiselect.min.js"></script>
   <link rel="stylesheet" href="../static/stylesheets/multiselect.css">
</head>
<body>
   <img src="../static/stylesheets/logo.svg" />
   <div class="content">
   <h1>Netflix Recommendation System </h1>
      <H1>Unlimited movies,TV shows and more</H1>
      <form name="passdata" action="/about" method="post">

         <p1>Select TV Show/Movie Names that you like: </p1>
         <select name="titles" multiple id="titles">
            {% for title in titles %}
            <option value= "{{title}}">{{title}}</option>"
            {% endfor %}
         </select>
         <br>
         <p1>Select Languages that you prefer: </p1>
         <select name="languages" multiple id="languages">
            {% for language in languages %}
               <option value= "{{language}}">{{language}}</option>"
            {% endfor %}
         </select>
         <br>
         <input type="submit" value="Get Started">
      </form>
   </div>
   <script>
      $('#titles').multiselect({
         columns: 1,
         placeholder: 'Select Titles',
         search: true,
         selectAll: true
      });
      $('#languages').multiselect({
         columns: 1,
         placeholder: 'Select Languages',
         search: true,
         selectAll: true
      });
   </script>
</body>
</html>
```

Moviepage.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{details[0]}}</title>
   <link rel="stylesheet" href="../static/stylesheets/moviepage.css" />
</head>
<body>
    <h1>Here are your selected movie details....</h1>
    <div class="container">
        <div class="left-column">
        <img src="{{details[21]}}" alt="{{details[0]}}" width=300px
height=500px />
        </div>
        <div class="right-column">
            <div class="movie-description">
                <span>{{details[19]}}</span>
                <h2>{{details[0]}}</h2>
                <p>{{details[18]}}</p>
            </div>
            <div class="movie-details">
                <span><p1>Genre:</p1> {{details[1]}}</span><br>
                <span><p1>Available Languages:</p1>
{{details[3]}}</span><br>
                <span><p1>Available in Countries:</p1>
{{details[4]}}</span><br>
                <span><p1>IMDb Score:</p1> {{details[10]}}</span><br>
                <span><p1>Viewers Rating:</p1> {{details[9]}}</span><br>
            </div>
            <div class="movie-details">
                <span><p1>Director(s):</p1> {{details[6]}}</span><br><br>
                <span><p1>Writer(s):</p1> {{details[7]}}</span><br>
                <span><p1>Acting Cast:</p1> {{details[8]}}</span><br>
                <span><p1>Production House(s):</p1>
{{details[16]}}</span><br>
                <span><p1>Boxoffice Collection:</p1>
{{details[13]}}</span><br>
                <span><p1>Nominated for Awards:</p1>
{{details[12]}}</span><br>
                <span><p1>Awards Won:</p1> {{details[11]}}</span><br>
            </div>
            <div>
                <a href="{{details[17]}}">Netflix Link</a>
            </div>
        </div>
    </div>
</body>
</html>
```

Result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Recommendations</title>
   <link rel="stylesheet" href="../static/stylesheets/result.css" />
</head>
<body>
<div class=page>
  <h1>Here are your recommendations....</h1>
  {% for image in images %}
    <a href='/moviepage/{{titles[loop.index0]}}'><img src="{{image}}"
width=250px height=400px/></a>
  {% endfor %}
</div>
</body>
</html>
```

Moviepage.css

```
/* Basic Styling */
html, body {
  height: 100%;
  width: 100%;
  margin: 1.25% 0 0 2.5%;;
  font-family: 'Roboto', sans-serif;
  background-color: black;
  color: white;
}

h1{
    border-bottom: 2px solid #eee;
}

.container {
  max-width: 1200px;
  margin-left: 2.5%;
  padding: 15px;
  display: flex;
}

.left-column {
    width: 45%;
    position: relative;
}

.right-column {
    width: 55%;
}

.left-column img {
    position: absolute;
    left: 0;
    top: 0;
```

```
    transition: all 0.3s ease;
    border: 2px solid white;
}

p1{
    font-weight: bold;
    margin-right: 7px;
}

.movie-description {
    border-bottom: 1px solid #E1E8EE;
    margin-bottom: 20px;
}
.movie-description span {
    font-size: 12px;
    color: white;
    letter-spacing: 1px;
    text-transform: uppercase;
    text-decoration: none;
}
.movie-description h2 {
    font-weight: 300;
    font-size: 52px;
    color: #358ED7;
    letter-spacing: -2px;
}
.movie-description p {
    font-size: 16px;
    font-weight: 300;
    color: #2433D7;
    line-height: 24px;
}

.movie-details{
    word-wrap: break-word;
    /* overflow-wrap: normal; */
}

a {
    background-color: white;
    padding: 2px;
}

@media (max-width: 940px) {
    .container {
        flex-direction: column;
        margin-top: 60px;
    }

    .left-column, .right-column {
        width: 100%;
    }

    .left-column img {
        width: 300px;
        right: 0;
        top: -65px;
        left: initial;
    }
}
```

```
@media (max-width: 535px) {
    .left-column img {
        width: 220px;
        top: -85px;
    }
}
```

Multiselect.css

```
.multiselect-wrapper {
   width: 180px;
   display: inline-block;
   white-space: nowrap;
   font-size: 12px;
   font-family: "Segoe UI", Verdana, Helvetica, Sans-Serif;
   margin: 1% 0 2.5% 3px;
}

.multiselect-wrapper .multiselect-input {
   width: 180px;
   padding-right: 50px;
}

.multiselect-wrapper label {
   display: block;
   font-size: 12px;
   font-weight : 600;
}

.multiselect-wrapper .multiselect-list {
   z-index: 1;
   position: absolute;
   display: none;
   background-color: black;
   border: 1px solid grey;
   border-bottom-left-radius: 2px;
   border-bottom-right-radius: 2px;
   margin-top: -2px;
}

   .multiselect-wrapper .multiselect-list.active {
      display: block;
   }

   .multiselect-wrapper .multiselect-list > span {
      font-weight: bold;
   }

   .multiselect-wrapper .multiselect-list .multiselect-checkbox {
      margin-right: 2px;
   }

   .multiselect-wrapper .multiselect-list > span,
   .multiselect-wrapper .multiselect-list li {
      cursor: default;
   }

.multiselect-wrapper .multiselect-list {
```

```
   padding: 5px;
   min-width: 225px;
   max-width: 350px;
   overflow-x: auto;
}

.multiselect-wrapper ul {
   list-style: none;
   display: block;
   position: relative;
   padding: 0px;
   margin: 0px;
   max-height: 200px;
   overflow-y: auto;
   overflow-x: hidden;
}

   .multiselect-wrapper ul li {
      padding-right: 20px;
      display: block;
   }

      .multiselect-wrapper ul li.active {
         background-color: rgb(0, 102, 255);
         color: black;
      }

      .multiselect-wrapper ul li:hover {
         background-color: rgb(0, 102, 255);
         color: black;
      }

.multiselect-input-div {
   height: 34px;
}

   .multiselect-input-div input{
      border: 1px solid #ababab;
      background : black;
      margin: 5px 0 6px 0;
      padding: 5px;
      vertical-align:middle;
   }

.multiselect-count {
   position: relative;
   text-align: center;
   border-radius: 2px;
   background-color: white;
   display: inline-block !important;
   padding: 2px 7px;
   left: -45px;
}

.multiselect-wrapper.disabled .multiselect-dropdown-arrow {
   border-top: 5px solid lightgray;
}

.multiselect-wrapper.disabled .multiselect-count {
   background-color: lightgray;
}
```

```
.multiselect-dropdown-arrow {
   width: 0;
   height: 0;
   border-left: 5px solid transparent;
   border-right: 5px solid transparent;
   border-top: 5px solid black;
   position: absolute;
   line-height: 20px;
   text-align: center;
   display: inline-block !important;
   margin-top: 17px;
   margin-left: -42px;
}
```

Result.css

```
body
{
  background-image: url('../stylesheets/home-bg.jpg');
  font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;
}

h1
{
  color: #377ba8;
  margin-bottom: 25px;
  border-bottom: 2px solid #eee;
}

.page {
  margin: 3.5% 0 0 2.5%;
}

.page img{
  padding: 7px;
}
```

Multiselect.min.js

```
if (!m_helper)
{
    var m_helper = {
        removeNode: function (id)
        {
            var el = document.getElementById(id);
            if (el) { el.parentNode.removeChild(el) }
        },
        insertAfter: function (item, target)
        {
```

```
            var parent = target.parentNode;
            if (target.nextElementSibling) { parent.insertBefore(item,
target.nextElementSibling) }
            else { parent.appendChild(item) }
        },
        hide: function (element)
        {
            element.style.display = 'none'
        },
        hideAll: function (array)
        {
            for (var i = 0; i < array.length; i++)
            { this.hide(array[i]) }
        },
        show: function (element)
        {
            element.style.display = 'block'
        },
        showAll: function (array)
        {
            for (var i = 0; i < array.length; i++)
            { this.show(array[i]) }
        },
        parent: function (element, id)
        {
            var parent = element.parentElement;
            while (parent && parent.tagName != 'BODY') {
                if (parent.id == id) { return parent }
                parent = parent.parentElement
            }
            return null
        },
        create: function (data) {
            var result = document.createElement(data.tag);
            if (data.id) { result.id = data.id }
            if (data.class) { result.className = data.class }
            if (data.attributes)
            {
                for (var prop in data.attributes) {
result.setAttribute(prop, data.attributes[prop]) }
            }
            if (data.data) { for (var prop in data.data) {
result.dataset[prop] = data.data[prop] } }
            return result
        },
        div: function (data)
        {
            if (!data) { data = new Object() }
            data.tag = 'div';
            return this.create(data)
        },
        label: function (data) {
            if (!data) { data = new Object() }
            data.tag = 'label'; return this.create(data)
        },
        textField: function (data) {
            if (!data) { data = new Object() }
            data.tag = 'input';
            if (!data.attributes)
                data.attributes = new Object(); data.attributes.type =
'text'; return this.create(data)
```

```
        },
        checkbox: function (data) {
            if (!data) { data = new Object() }
            data.tag = 'input';
            if (!data.attributes)
                data.attributes = new Object();
            data.attributes.type = 'checkbox';
            return this.create(data)
        },
        each: function (array, handler)
        {
            for (var i = 0; i < array.length; i++){ handler(array[i]) }
        },
        setActive: function (element)
        {
            element.classList.add('active')
        },
        setUnactive: function (element)
        {
            element.classList.remove('active')
        },
        select: function (element)
        {
            element.selected = !0;
            element.setAttribute('selected', 'selected')
        },
        deselect: function (element)
        {
            element.selected = !1;
            element.removeAttribute('selected')
        },
        check: function (element)
        {
            element.checked = !0
        },
        uncheck: function (element)
        {
            element.checked = !1
        },
        click: function (element)
        {
            if (element.fireEvent) { el.fireEvent('onclick') }
            else {
                var evObj = document.createEvent('Events');
                evObj.initEvent('click', !0, !1);
element.dispatchEvent(evObj)
            }
        },
        setDisabled: function (element, value)
        {
            element.disabled = value
        },
    }
}
function Multiselect(item, opts)
{
    if ((typeof ($) != 'undefined' && !$(item).is('select')) || (typeof ($)
== 'undefined' && item.tagName != 'SELECT'))
    {
        throw "Multiselect: passed object must be a select"
    }
```

```
    if ((typeof ($) != 'undefined' && !$(item).attr('multiple')) || (typeof
($) == 'undefined' && !item.hasAttribute('multiple')))
    {
        throw "Multiselect: passed object should contain 'multiple'
attribute"
    }
    this._item = item;
    this._createUI();
    this._appendEvents();
    this._initSelectedFields();
    this._initIsEnabled()
}
Multiselect.prototype =
{
    _createUI: function () {
        m_helper.removeNode(this._getIdentifier());
        var wrapper = this._createWrapper(); m_helper.insertAfter(wrapper,
this._item); wrapper.appendChild(this._createInputField());
wrapper.appendChild(this._createItemList()); m_helper.hide(this._item)
    },
    _createWrapper: function () {
        var result = document.createElement('div');
        result.className = 'multiselect-wrapper';
        result.id = this._getIdentifier();
        return result
    },
    _createInputField: function () {
        var input = m_helper.textField({
            id: this._getInputFieldIdentifier(),
            class: 'multiselect-input', attributes: { autocomplete: 'off' }
        }),
            label = m_helper.label({
                id: this._getInputBadgeIdentifier(),
                class: 'multiselect-count', attributes: { for:
this._getInputFieldIdentifier() }
            }),
            dropDownArrow = m_helper.label({
                class: 'multiselect-dropdown-arrow',
                attributes: { for: this._getInputFieldIdentifier() }
            }),
            result = m_helper.div({ class: 'multiselect-input-div' });
        label.style.visibility = 'hidden';
        label.innerHTML = 0;
        result.appendChild(input);
        result.appendChild(label);
        result.appendChild(dropDownArrow);
        return result
    },
    _createItemList: function ()
    {
        var list = m_helper.create({ tag: 'ul' });
        var self = this;
        m_helper.each(this._getItems(this._item),
            function (e) {
                var insertItem = self._createItem('li', e.id, e.text,
e.selected);
                list.appendChild(insertItem);
                var checkBox =
insertItem.querySelector('input[type=checkbox]');
                e.multiselectElement = checkBox;
                checkBox.dataset.multiselectElement = JSON.stringify(e)
```

```
            });
        var selectAll = this._createItem('span', -1, 'Select all');
        var result = m_helper.div({ id: this._getItemListIdentifier(),
class: 'multiselect-list' });
        result.appendChild(selectAll);
        result.appendChild(m_helper.create({ tag: 'hr' }));
        result.appendChild(list);
        return result
    },
    _createItem: function (wrapper, value, text, selected)
    {
        var checkBox = m_helper.checkbox({ class: 'multiselect-checkbox',
data: { val: value } }),
            textBox = m_helper.create({ tag: 'span', class: 'multiselect-
text' }),
            result = m_helper.create({ tag: wrapper }),
            label = m_helper.label();
        textBox.className = 'multiselect-text';
        textBox.innerHTML = text;
        label.appendChild(checkBox);
        label.appendChild(textBox);
        result.appendChild(label);
        return result
    },
    _initSelectedFields: function () {
        var itemResult = this._getItems().filter(function (obj) { return
obj.selected });
        if (itemResult.length != 0)
        {
            var self = this;
            m_helper.each(itemResult, function (e) { self.select(e.id) })
        }
        this._hideList(this)
    },
    _initIsEnabled: function ()
    { this.setIsEnabled(!this._item.disabled) },
        destroy() {
            m_helper.removeNode(this._getIdentifier());
            m_helper.show(this._item);
            var index = window.multiselects._items.indexOf(this._item);
            if (index > -1) {
                window.multiselects._items.splice(index, 1);
                window.multiselects.splice(index, 1)
            }
    },
    select: function (val) { this._toggle(val, !0) },
    deselect: function (val) { this._toggle(val, !1) },
    setIsEnabled(isEnabled) {
        if (this._isEnabled === isEnabled) return;
        var wrapperItem = document.getElementById(this._getIdentifier());
        if (isEnabled) { wrapperItem.classList.remove('disabled') }
        else {
            wrapperItem.classList.add('disabled')
        }
        m_helper.setDisabled(this._item, !isEnabled);

m_helper.setDisabled(document.getElementById(this._getInputFieldIdentifier(
)), !isEnabled);
        this._isEnabled = isEnabled
    },
    _toggle: function (val, setCheck)
```

```
    {
        var self = this;
        if (val) {

m_helper.each(document.getElementById(this._getIdentifier()).querySelectorA
ll('.multiselect-checkbox'), function (e)
            {
                if (e.dataset.val == val)
                {
                    if (setCheck && !e.checked)
                    { m_helper.check(e); self._onCheckBoxChange(e, self) }
                    else if (!setCheck && e.checked)
                    {
                        m_helper.uncheck(e);
                        self._onCheckBoxChange(e, self)
                    }
                }
            });
            self._updateText(self)
        }
    },
    selectAll: function (val)
    {
        var selectAllChkBox = document.querySelector('#' +
this._getIdentifier() + ' .multiselect-checkbox');
        m_helper.check(selectAllChkBox);
        this._onCheckBoxChange(selectAllChkBox, this);
        this._updateText(this)
    },
    deselectAll: function ()
    {
        var selectAllChkBox = document.querySelector('#' +
this._getIdentifier() + ' .multiselect-checkbox');
        m_helper.uncheck(selectAllChkBox);
        this._onCheckBoxChange(selectAllChkBox, this);
        this._updateText(this)
    },
    _checkboxClickEvents: {},
    setCheckBoxClick(id, handler) {
        if (typeof handler === "function")
        { this._checkboxClickEvents[id] = handler }
        else { console.error("Checkbox click handler for checkbox value=" +
id + " is not a function") }
        return this
    },
    _appendEvents: function () {
        var self = this;

document.getElementById(self._getInputFieldIdentifier()).addEventListener('
focus', function (event)
        {
            self._showList(self);
            document.getElementById(self._getInputFieldIdentifier()).value
= '';
            m_helper.each(window.multiselects, function (e)
            {
                if
(document.getElementById(e._getItemListIdentifier()).offsetParent &&
m_helper.parent(event.target, e._getIdentifier()))
                { e._hideList(self) }
            })
```

```
        });

document.getElementById(self._getInputFieldIdentifier()).addEventListener('
click', function ()
        { self._showList(self);
document.getElementById(self._getInputFieldIdentifier()).value = '' });

document.getElementById(self._getIdentifier()).addEventListener('click',
function (event)
        {
            event = event || window.event;
            var target = event.target || event.srcElement;
            if (m_helper.parent(target, self._getIdentifier())) {
event.stopPropagation() }
        });

document.getElementById(self._getItemListIdentifier()).addEventListener('mo
useover', function ()
        {
            self._showList(self)
        });

m_helper.each(document.getElementById(self._getIdentifier()).querySelectorA
ll('.multiselect-checkbox'), function (e)
        {
            e.addEventListener('change', function (event)
            { self._onCheckBoxChange(e, self, event) })
        });
        var onInput = function ()
        {
            var text = this.value.toLowerCase();
            if (!text || text == '') {
                m_helper.show(document.querySelector('#' +
self._getItemListIdentifier() + ' > span'));
                m_helper.show(document.querySelector('#' +
self._getItemListIdentifier() + ' > hr'));
                m_helper.showAll(document.querySelectorAll('#' +
self._getItemListIdentifier() + ' li'))
            } else {
                m_helper.hide(document.querySelector('#' +
self._getItemListIdentifier() + ' > span'));
                m_helper.hide(document.querySelector('#' +
self._getItemListIdentifier() + ' > hr'));
                var array =
Array.prototype.filter.call(document.querySelectorAll('#' +
self._getItemListIdentifier() + ' li span'), function (obj)
                { return obj.innerHTML.toLowerCase().indexOf(text) > -1 });
                m_helper.hideAll(document.querySelectorAll('#' +
self._getItemListIdentifier() + ' li'));
                m_helper.each(array, function (e) {
m_helper.show(e.parentElement.parentElement) })
            }
        }

document.getElementById(self._getInputFieldIdentifier()).addEventListener('
propertychange', onInput);

document.getElementById(self._getInputFieldIdentifier()).addEventListener('
input', onInput)
    },
    _onCheckBoxChange: function (checkbox, self, event) {
```

```
        if (!checkbox.dataset.multiselectElement) {
            var checkedState = self._performSelectAll(checkbox, self);
            if (typeof self._checkboxClickEvents.checkboxAll ===
"function")
            { self._checkboxClickEvents.checkboxAll(checkbox, { checked:
checkedState }) }
        }
        else {
            var checkedState = self._performSelectItem(checkbox, self);
            if (typeof self._checkboxClickEvents[checkedState.id] ===
"function") { self._checkboxClickEvents[checkedState.id](checkbox,
checkedState) }
            self._updateSelectAll(self)
        }
        self._forceUpdate()
    }, _performSelectItem: function (checkbox, self) {
        var item = JSON.parse(checkbox.dataset.multiselectElement);
        if (checkbox.checked) {
            self._itemCounter++;
m_helper.select(this._item.options[item.index]);
            m_helper.setActive(checkbox.parentElement.parentElement);
            return { id: item.id, checked: !0 }
        }
        self._itemCounter--;
m_helper.deselect(this._item.options[item.index]);
        m_helper.setUnactive(checkbox.parentElement.parentElement);
        return { id: item.id, checked: !1 }
    },
    _performSelectAll: function (checkbox, self) {
        var items = self._getItems();
        if (checkbox.checked) {
            self._itemCounter = items.length;
            m_helper.each(items, function (e) {

m_helper.setActive(e.multiselectElement.parentElement.parentElement);
                m_helper.select(self._item.options[e.index]);
                m_helper.check(e.multiselectElement)
            });
            return !0
        }
        self._itemCounter = 0; m_helper.each(items, function (e) {

e.multiselectElement.parentElement.parentElement.classList.remove('active')
;
            m_helper.deselect(self._item.options[e.index]);
            m_helper.uncheck(e.multiselectElement)
        });
        return !1
    },
    _updateSelectAll: function (self) {
        var allChkBox =
document.getElementById(self._getItemListIdentifier()).querySelector('input
[type=checkbox]');
        if (self._itemCounter == self._getItems().length) {
allChkBox.checked = !0 }
        else if (allChkBox.checked) { allChkBox.checked = !1 }
    }, _hideList: function (context, event) {

m_helper.setUnactive(document.getElementById(context._getItemListIdentifier
()));
```

```
m_helper.show(document.getElementById(context._getItemListIdentifier()).que
rySelector('span'));

m_helper.show(document.getElementById(context._getItemListIdentifier()).que
rySelector('hr'));

m_helper.showAll(document.getElementById(context._getItemListIdentifier()).
querySelectorAll('li'));
        context._updateText(context);
        if (event)
            event.stopPropagation()
    },
    _updateText: function (context) {
        var activeItems =
document.getElementById(context._getItemListIdentifier()).querySelectorAll(
'ul .active');
        if (activeItems.length > 0) {
            var val = '';
            for (var i = 0; i < (activeItems.length < 5 ?
activeItems.length : 5); i++)
            { val += activeItems[i].innerText + ", " }
            val = val.substr(0, val.length - 2);
            if (val.length > 20) { val = val.substr(0, 17) + '...' }
        }
        if (activeItems.length ==
document.getElementById(context._getItemListIdentifier()).querySelectorAll(
'ul li').length) { val = 'All selected' }
        document.getElementById(context._getInputFieldIdentifier()).value =
val ? val : ''
    },
    _showList: function (context) {
m_helper.setActive(document.getElementById(context._getItemListIdentifier()
)) }, _forceUpdate: function () {
        var badge =
document.getElementById(this._getInputBadgeIdentifier());
        badge.style.visibility = 'hidden';
        if (this._itemCounter != 0) {
            badge.innerHTML = this._itemCounter;
            badge.style.visibility = 'visible';
            var ddArrow = badge.nextElementSibling;
            if (this._itemCounter < 10) {
                badge.style.left = '-45px';
                ddArrow.style.marginLeft = '-42px'
            }
            else if (this._itemCounter < 100) {
                badge.style.left = '-50px';
                ddArrow.style.marginLeft = '-47px'
            }
            else if (this._itemCounter < 1000) {
                badge.style.left = '-55px';
                ddArrow.style.marginLeft = '-52px'
            }
            else if (this._itemCounter < 10000) {
                badge.style.left = '-60px';
                ddArrow.style.marginLeft = '-57px'
            }
        }
    },
    _items: undefined, _itemCounter: 0, _isEnabled: !0, _getItems: function
() {
        if (this._items == undefined) {
```

```
            var result = [];
            var opts = this._item.options;
            for (var i = 0; i < opts.length; i++){
                var insertItem = { id: opts[i].value, index: i, text:
opts[i].innerHTML, selected: !!opts[i].selected, selectElement: opts[i] };
                result.push(insertItem)
            }
            this._items = result
        }
        return this._items
    },
    _getItemUniqueIdentifier: function () {
        var id = this._item.getAttribute('id'), name =
this._item.getAttribute('name');
        if (!(id || name)) { throw "Multiselect: object does not contain
any identifier (id or name)" }
        return id ? id : name
    },
    _getIdentifier: function () { return this._getItemUniqueIdentifier() +
'_multiSelect' }, _getInputFieldIdentifier:
        function () { return this._getItemUniqueIdentifier() + '_input' },
_getItemListIdentifier:
        function () { return this._getItemUniqueIdentifier() + '_itemList'
}, _getInputBadgeIdentifier:
        function () { return this._getItemUniqueIdentifier() +
'_inputCount' }
}
window.multiselects = [];
if (typeof ($) != 'undefined') {
    $.fn.multiselect = function () {
        var res = [];
        if (!window.multiselects._items) { window.multiselects._items = []
}
        if (this.length != 0) {
            $(this).each(function (i, e)
            {
                var index = window.multiselects._items.indexOf(e);
                if (index == -1)
                {
                    var inputItem = new Multiselect(e);
                    window.multiselects.push(inputItem);
                    window.multiselects._items.push(e);
                    res.push(inputItem)
                } else { res.push(window.multiselects[index]) }
            })
        }
        return res.length == 1 ? res[0] : $(res)
    };
    $(document).click(function (event) { hideMultiselects(event) })
} else {
    document.multiselect = function (selector) {
        var res = []; if (!window.multiselects._items)
        {
            window.multiselects._items = []
        }
        m_helper.each(document.querySelectorAll(selector), function (e) {
            var index = window.multiselects._items.indexOf(e);
            if (index == -1) {
                var inputItem = new Multiselect(e);
                window.multiselects.push(inputItem);
                window.multiselects._items.push(e);
```

```
                res.push(inputItem)
            } else { res.push(window.multiselects[index]) }
        }); return res.length == 1 ? res[0] : res
    }
    window.onclick = function (event) { hideMultiselects(event) }
}
function hideMultiselects(event) {
    m_helper.each(window.multiselects,
        function (e) {
            if
(document.getElementById(e._getItemListIdentifier()).offsetParent &&
!m_helper.parent(event.target, e._getIdentifier()))
            { e._hideList(e, event) }
        })
}
```

app.py

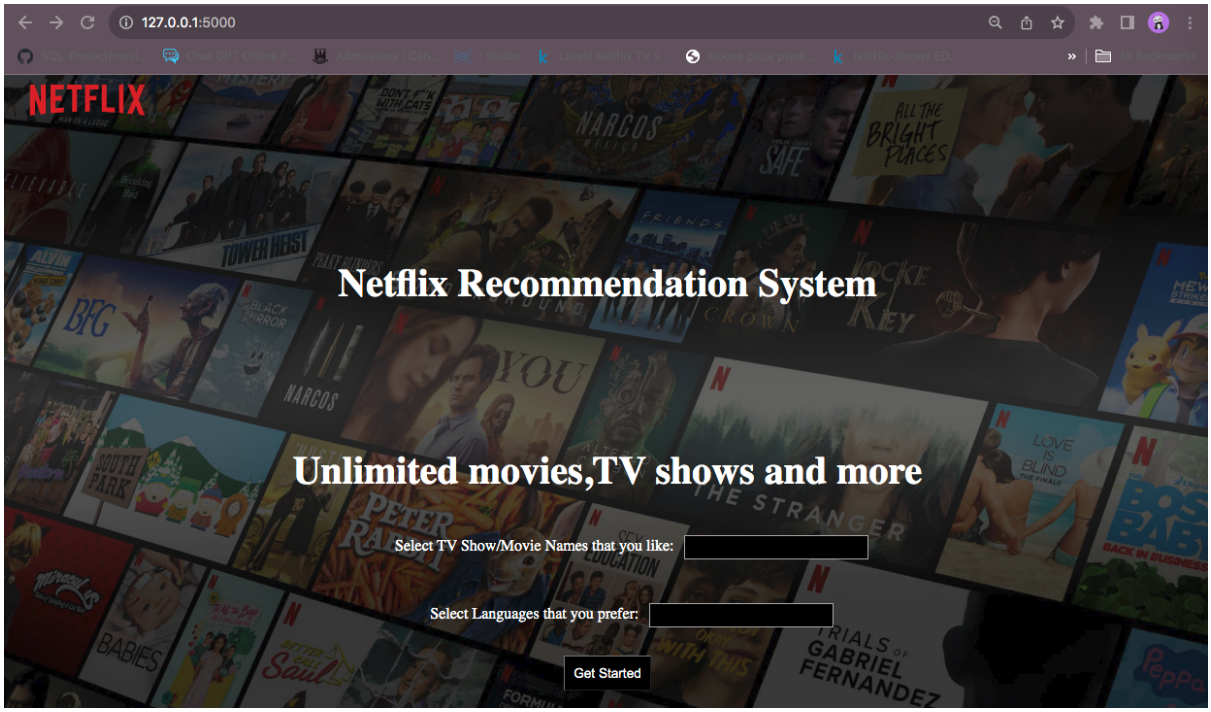
```
import math
from flask import Flask, render_template, request
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def prepare_data(x):
    return str.lower(x.replace(" ", ""))

def create_soup(x):
    return f"{x['Genre']} {x['Tags']} {x['Actors']} {x['ViewerRating']}"

def get_recommendations(title, cosine_sim):
    global result
    title = title.replace(' ', '').lower()
    if title not in indices:
        return pd.DataFrame()  # Return empty DataFrame if title not found

    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:51]
    movie_indices = [i[0] for i in sim_scores]
    result = netflix_data.iloc[movie_indices]
    result.reset_index(inplace=True)
    return result

# Load and preprocess data
netflix_data = pd.read_csv('NetflixDataset.csv', encoding='latin-1',
index_col='Title')
netflix_data.index = netflix_data.index.str.title()
netflix_data = netflix_data[~netflix_data.index.duplicated()]
netflix_data.rename(columns={'View Rating': 'ViewerRating'}, inplace=True)

# Create language and title sets
Language = netflix_data.Languages.str.get_dummies(',')
```

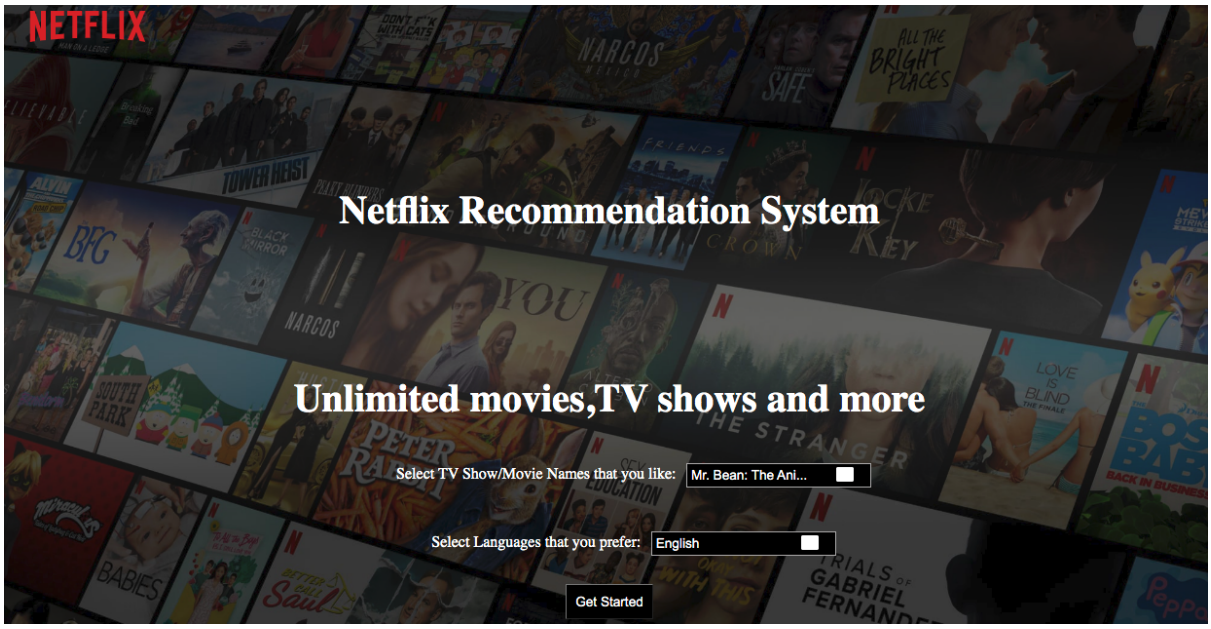
```
Lang = Language.columns.str.strip().values.tolist()
Lang = set(Lang)
Titles = set(netflix_data.index.to_list())

# Fill missing values and prepare features
netflix_data['Genre'] = netflix_data['Genre'].astype('str')
netflix_data['Tags'] = netflix_data['Tags'].astype('str')
netflix_data['IMDb Score'] = netflix_data['IMDb Score'].apply(lambda x: 6.6
if math.isnan(x) else x)
netflix_data['Actors'] = netflix_data['Actors'].astype('str')
netflix_data['ViewerRating'] = netflix_data['ViewerRating'].astype('str')
new_features = ['Genre', 'Tags', 'Actors', 'ViewerRating']

# Prepare selected data
selected_data = netflix_data[new_features].copy()
for new_feature in new_features:
    selected_data.loc[:, new_feature] =
selected_data[new_feature].apply(prepare_data)
selected_data.index = selected_data.index.str.lower()
selected_data.index = selected_data.index.str.replace(" ", '')

# Create soup for feature extraction
selected_data['soup'] = selected_data.apply(create_soup, axis=1)

# Vectorize and calculate cosine similarity
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(selected_data['soup'])
cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
selected_data.reset_index(inplace=True)
indices = pd.Series(selected_data.index, index=selected_data['Title'])
result = pd.DataFrame()
df = pd.DataFrame()

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html', languages=Lang, titles=Titles)

@app.route('/about', methods=['POST'])
def getvalue():
    global df
    movienames = request.form.getlist('titles')
    languages = request.form.getlist('languages')
    df = pd.DataFrame()  # Clear df before processing new recommendations
    for moviename in movienames:
        recommended_movies = get_recommendations(moviename, cosine_sim2)
        for language in languages:
            df =
pd.concat([recommended_movies[recommended_movies['Languages'].str.count(lan
guage) > 0], df], ignore_index=True)
    df.drop_duplicates(keep='first', inplace=True)
    df.sort_values(by='IMDb Score', ascending=False, inplace=True)
    images = df['Image'].tolist()
    titles = df['Title'].tolist()
    return render_template('result.html', titles=titles, images=images)

@app.route('/moviepage/<name>')
def movie_details(name):
    global df
    details_list = df[df['Title'] == name].to_numpy().tolist()
```

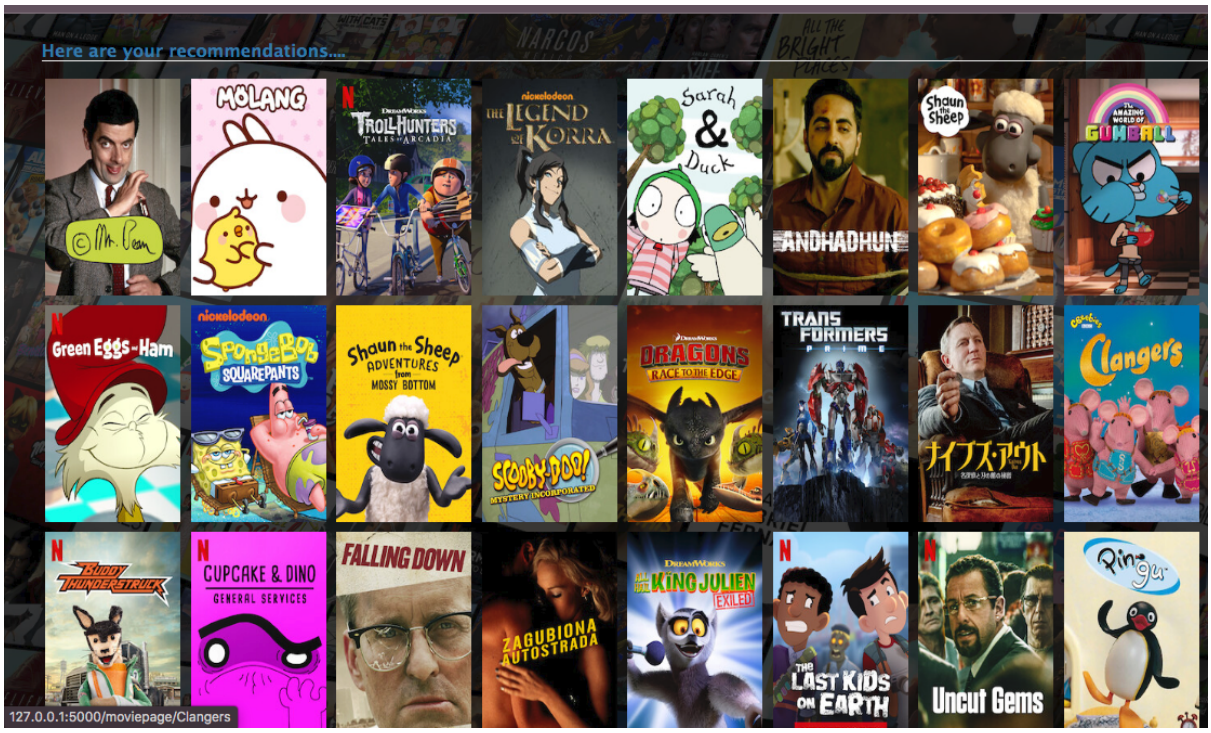
```
    if not details_list:
        return "Movie not found", 404
    return render_template('moviepage.html', details=details_list[0])

if __name__ == '__main__':
    app.run(debug=False,port=5000)
```

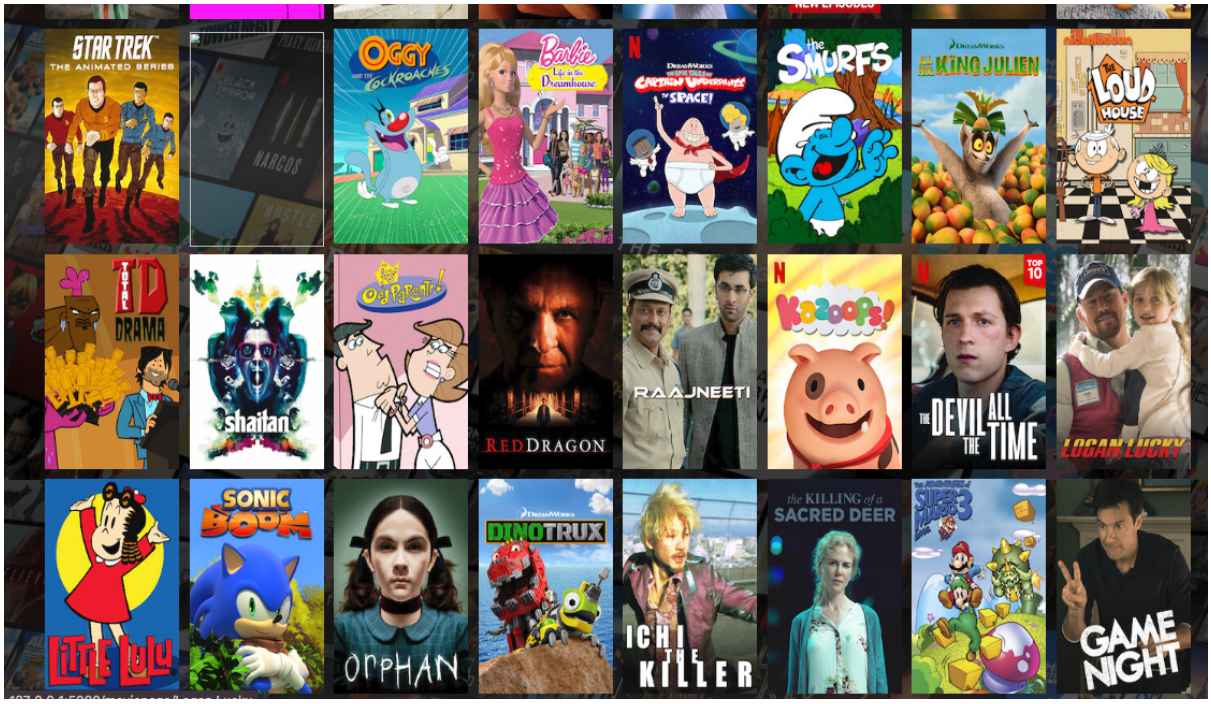
Output

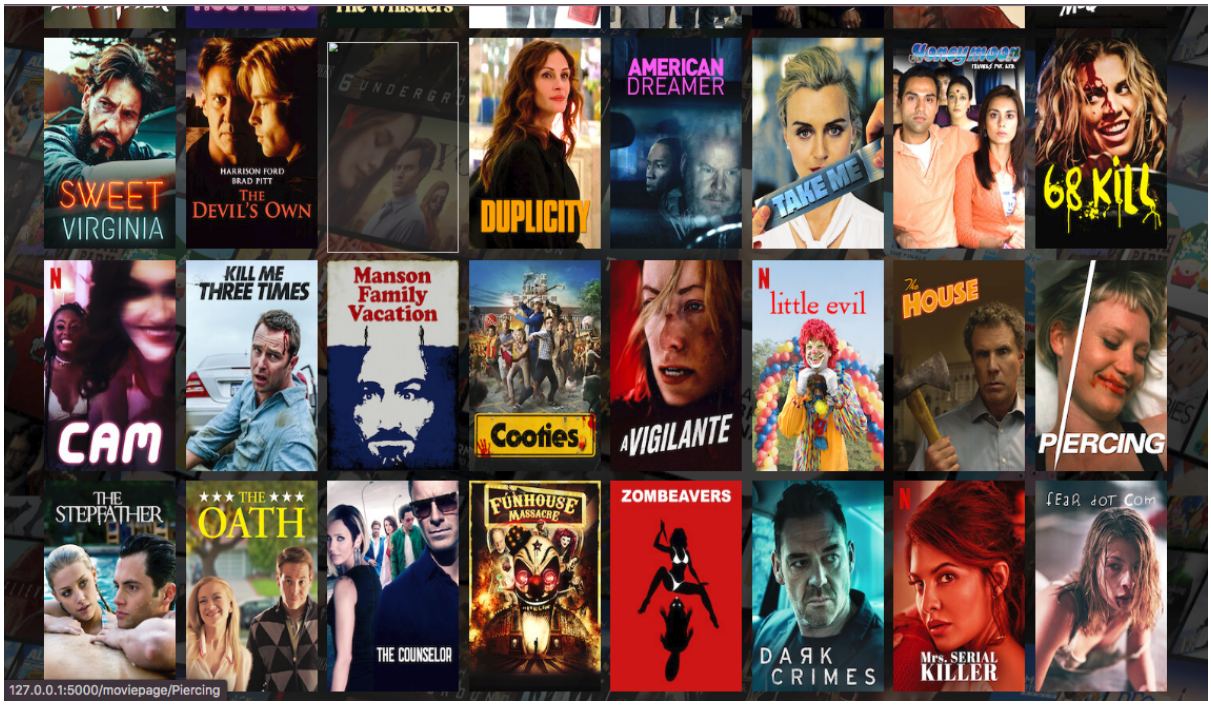
```
/Users/shreyabhattacharjee/PycharmProjects/Netflix-Recommendation-System2/venv/bin/python /Users/shreyabhattacharjee/PycharmProjects/Netfli
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [13/Sep/2024 11:43:14] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [13/Sep/2024 11:43:14] "GET /static/stylesheets/style.css HTTP/1.1" 200 -
127.0.0.1 - - [13/Sep/2024 11:43:14] "GET /static/stylesheets/multiselect.css HTTP/1.1" 200 -
```




NETFLIX
Netflix Recommendation System
Unlimited movies,TV shows and more
Select TV Show/Movie Names that you like:
Mr. Bean: The Ani...
Select Languages that you prefer:
English
Get Started


Here are your recommendations....
127.0.0.1:5000/moviepage/Clangers


STAR TREK
THE ANIMATED SERIES
OGGY AND THE COCKROACHES
Barbie Life in the Dreamhouse
CAPTAIN UNDERPANTS IN SPACE!
the SMURFS
KING JULIEN
THE LOUD HOUSE
TOTAL DRAMA
shaitan
RED DRAGON
RAAJNEETI
KAZOOPS!
TOP 10
THE DEVIL ALL THE TIME
LOGAN LUCKY
LITTLE LULU
SONIC BOOM
ORPHAN
DINOTRUX
ICHI THE KILLER
the KILLING of a SACRED DEER
SUPER MARIO 3
GAME NIGHT


SWEET VIRGINIA
HARRISON FORD
BRAD PITT
THE DEVIL'S OWN
DUPLICITY
AMERICAN DREAMER
TAKE ME
68 KILL
CAM
KILL ME THREE TIMES
Manson Family Vacation
Cooties
VIGILANTE
little evil
The HOUSE
PIERCING
THE STEPFATHER
THE OATH
THE COUNSELOR
FUNHOUSE MASSACRE
ZOMBEAVERS
DARK CRIMES
Mrs. SERIAL KILLER
FEAR DOT COM
127.0.0.1:5000/moviepage/Piercing

Here are your selected movie details....

SERIES

Mr. Bean: The Whole Bean

Funnyman Rowan Atkinson stars as the incorrigible Mr. Bean in this anthology that includes all 14 hilarious episodes of the sketch television series.

Genre: Comedy, Family
Available Languages: English
Available in Countries: United Kingdom
IMDb Score: 8.5
Viewers Rating: TV-G
Director(s): nan

Writer(s): Rowan Atkinson, Richard Curtis
Acting Cast: Rowan Atkinson
Production House(s): nan
Boxoffice Collection: nan
Nominated for Awards: 8.0
Awards Won: 4.0
Netflix Link

Here are your selected movie details....

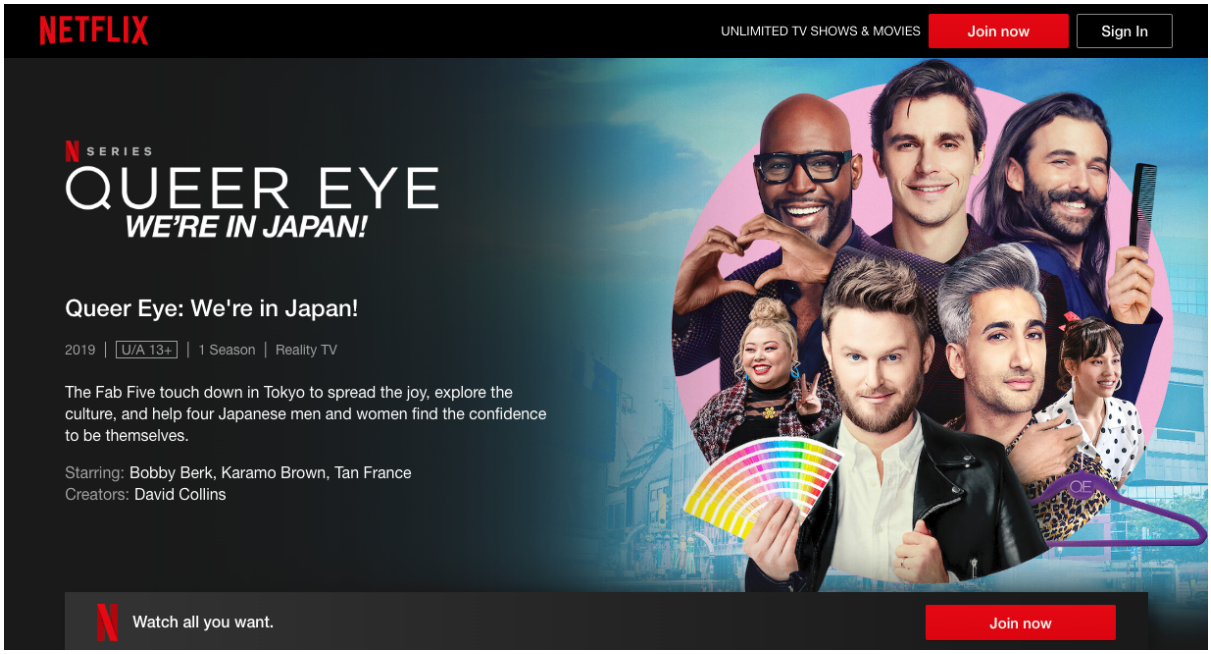
MOVIE

Andhadhun

A visually impaired pianist s world careens into a series of shocking twists after he unintentionally lands at the murder scene of a former film star.

Genre: Crime, Drama, Music, Mystery, Thriller
Available Languages: Hindi, English
Available in Countries:
Australia,Russia,Singapore,India,Hungary,Slovakia,Germany,Lithuania,Romania,Switzerland, United Kingdom,Canada,Hong Kong,Argentina,Mexico,South Africa,Thailand,Turkey,Czech Republic,Belgium,United States,Greece,Malaysia,Brazil,Netherlands,Italy,Iceland,Israel,Colombia
IMDb Score: 8.3
Viewers Rating: NotRated
Director(s): Sriram Raghavan

Writer(s): Hemanth M. Rao, Pooja Ladha Surti, Sriram Raghavan, Yogesh Chandekar, Olivier Treiner, Arijit Biswas
Acting Cast: Radhika Apte, Anil Dhawan, Tabu, Ayushmann Khurrana
Production House(s): Matchbox Pictures Inc., Viacom 18 Motion Pictures
Boxoffice Collection: $11,93,046
Nominated for Awards: 25.0
Awards Won: 27.0
Netflix Link


NETFLIX
UNLIMITED TV SHOWS & MOVIES
Join now
Sign In
SERIES
QUEER EYE
WE'RE IN JAPAN!
Queer Eye: We're in Japan!
2019 | U/A 13+ | 1 Season | Reality TV
The Fab Five touch down in Tokyo to spread the joy, explore the culture, and help four Japanese men and women find the confidence to be themselves.
Starring: Bobby Berk, Karamo Brown, Tan France
Creators: David Collins
Watch all you want.
Join now

Chapter 8

Conclusion & Future Scope

8.1 Conclusion

In conclusion, the Netflix recommendation system has effectively met its goals by delivering a functional and user-friendly platform that provides personalized movie and TV show recommendations based on user preferences. The successful integration of various components and thorough testing have ensured reliable performance and a satisfying user experience. Moving forward, there are ample opportunities for enhancement, including the adoption of advanced machine learning models, improved user experience features, expanded data integration, and enhanced scalability. By addressing these areas, the system can continue to evolve, offering even more accurate, engaging, and seamless recommendations for users.

8.2 Future Scope

Looking to the future, there are numerous opportunities to enhance and expand the Netflix recommendation system. Incorporating advanced machine learning models, such as collaborative filtering, content-based filtering, and hybrid approaches, could significantly improve the accuracy and relevance of recommendations.

1. **Advanced Recommendation Algorithms:**
   - **Machine Learning Models:** Implement collaborative filtering, content-based filtering, and hybrid approaches to enhance recommendation accuracy and relevance.
   - **User Behavior Analysis:** Utilize detailed user behavior and interaction data to refine recommendation algorithms and offer more personalized suggestions.

2. **Enhanced User Experience:**

- **User Profiles:** Develop detailed user profiles to deliver more targeted and engaging recommendations.
- **Feedback Mechanism:** Introduce a feedback system for users to rate recommendations, allowing for continuous improvement and personalization.

3. **Expanded Data Integration:**

- **Third-Party APIs:** Integrate additional APIs to provide comprehensive movie/TV show data, including reviews, ratings, and trending content.
- **Cross-Platform Integration:** Extend integration to other streaming platforms and services, offering a unified recommendation experience.

4. **Scalability and Performance:**

- **Load Balancing:** Implement load balancing techniques to handle increased traffic and ensure system stability.
- **Optimization:** Optimize database queries and recommendation algorithms to improve performance and response times.

5. **Mobile and Multi-Platform Support:**

- **Mobile Applications:** Develop mobile apps for iOS and Android to reach a wider audience and enhance cross-device accessibility.
- **Responsive Design:** Ensure the frontend interface is fully responsive and optimized for various screen sizes and devices.

6. **Security and Privacy:**

- **Data Security:** Enhance measures to protect user information and preferences.
- **Privacy Policies:** Ensure compliance with data protection regulations and implement clear privacy policies to build user trust.

7.Future Trends:

- **AI Integration:** Explore the use of artificial intelligence technologies, such as natural language processing and sentiment analysis, to further enhance recommendation accuracy.
- **Voice Assistants:** Integrate with voice assistants to provide voice-activated recommendations and improve user convenience.

8.3 References

1. Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer. https://www.springer.com/gp/book/9783319296579

2. Koren, Y., Bell, R., & Volinsky, C. (2009). *Matrix Factorization Techniques for Recommender Systems*. IEEE Computer, 42(8), 30-37. https://ieeexplore.ieee.org/document/5197422

3. Netflix Research. (n.d.). *Netflix Prize and Recommendation System Research*. https://research.netflix.com/

4. Karypis, G. (2001). *Evaluation of Item-Based Top-N Recommendation Algorithms*. Proceedings of the Tenth International Conference on Information and Knowledge Management. https://dl.acm.org/doi/10.1145/502585.502627

5. Mnih, A., & Salakhutdinov, R. (2008). *Probabilistic Matrix Factorization*. In *Advances in Neural Information Processing Systems* (pp. 1257-1264). https://proceedings.neurips.cc/paper/2007/file/d7322ed717dedf1eb4e6e52a37ea7a1f-Paper.pdf