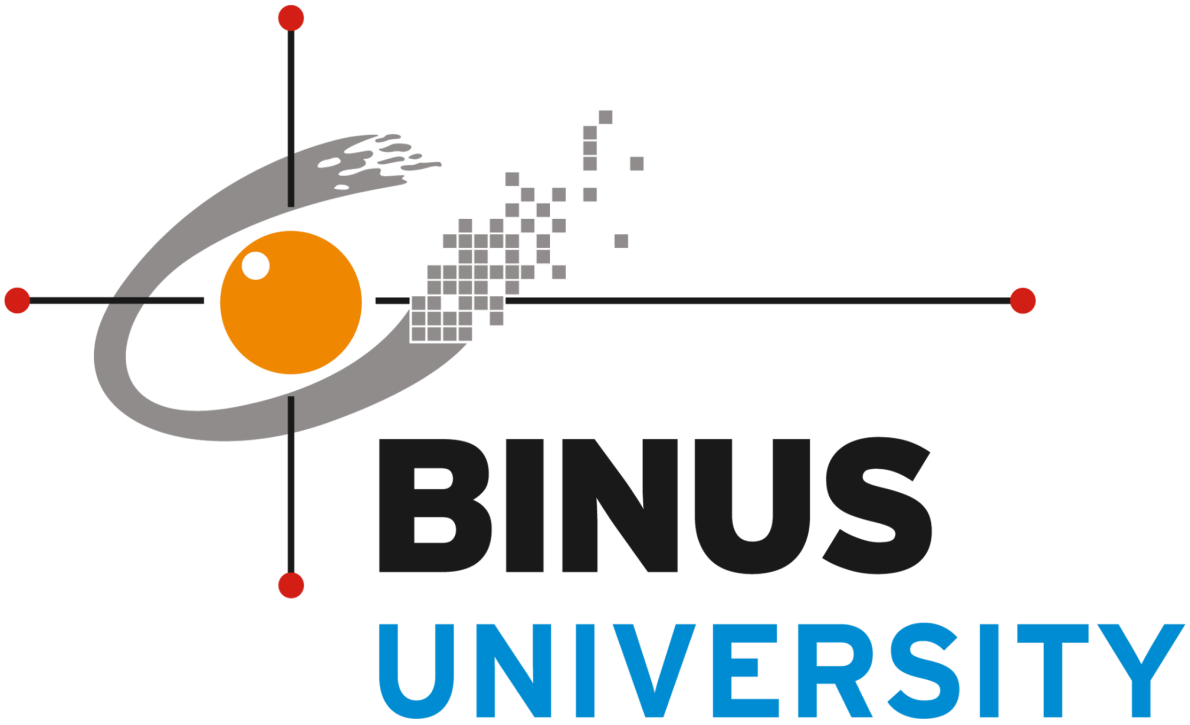
Recommendation System Using Content Based Filtering on Netflix TV Shows Data



Writers:
2702279205 - Jade Wulantrisna Aitken
2702281260 - Imerson Sanmarlow Krysthio
2702299863 - Vincent Moswen

INTRODUCTION

I. Problem Explanation

In the modern digital era, the rapid growth of online streaming platforms has led to an overwhelming abundance of content. Services such as Netflix offer thousands of movies and TV shows, making it increasingly difficult for users to manually discover content that aligns with their personal preferences. As a result, personalized recommendation systems have become an essential feature for improving user experience and engagement by suggesting relevant content based on individual interests.

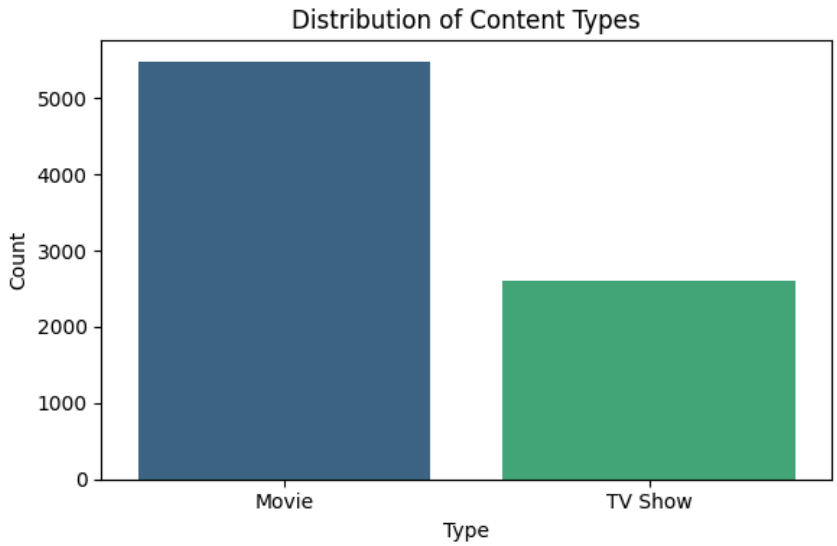
This project focuses on building a movie recommendation system using the content-based filtering approach. The recommendation system is developed based on publicly available Netflix dataset, which is obtained from Kaggle. The dataset contains various attributes related to the movies and TV shows, such as title, director, cast, genre, release year, and description, which serve as the foundation for generating recommendations.

The objective of this project is to design, implement, and deploy a content-based recommendation system that can suggest movies to users based on the content similarity of the items. The project includes several key stages, starting from data preprocessing, exploratory data analysis, model development using object-oriented programming (OOP), and using *pickle* files for storing trained models and vectorizers files. Finally, the system is deployed through a web application using Streamlit, with FastAPI serving as the backend framework.

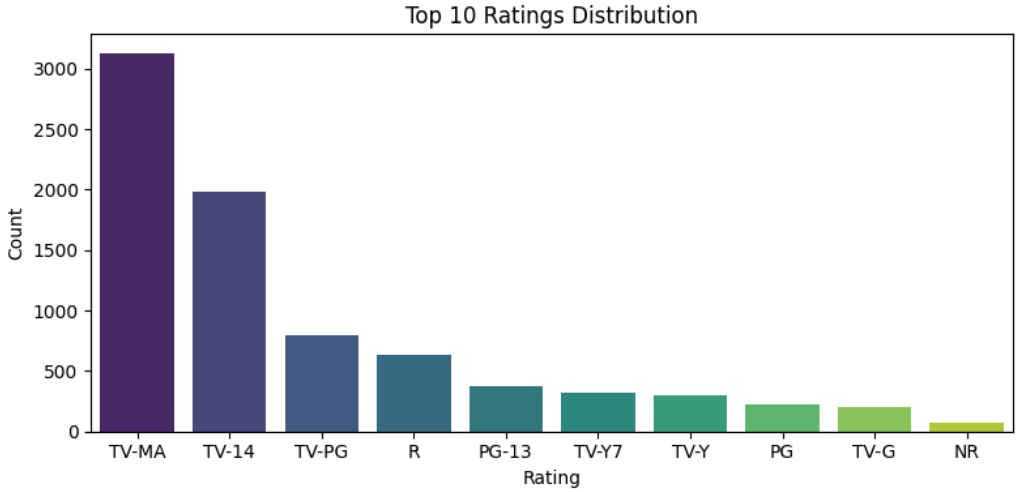
By leveraging content-based filtering, the system analyzes the attributes of movies that the user has expressed interest in, and recommends other movies that share similar characteristics. This approach enables personalized and relevant suggestions without requiring historical user ratings or external collaborative data.

II. Data Analysis Results

The dataset used in this project, titled *Netflix Movies and TV Shows*, was sourced from Kaggle (https://www.kaggle.com/datasets/anandshaw2001/netflix-movies-and-tv-shows). It contains comprehensive metadata about the movies and TV shows available on Netflix as of the time of data collection. The dataset consists of 8807 records, each representing an individual title.

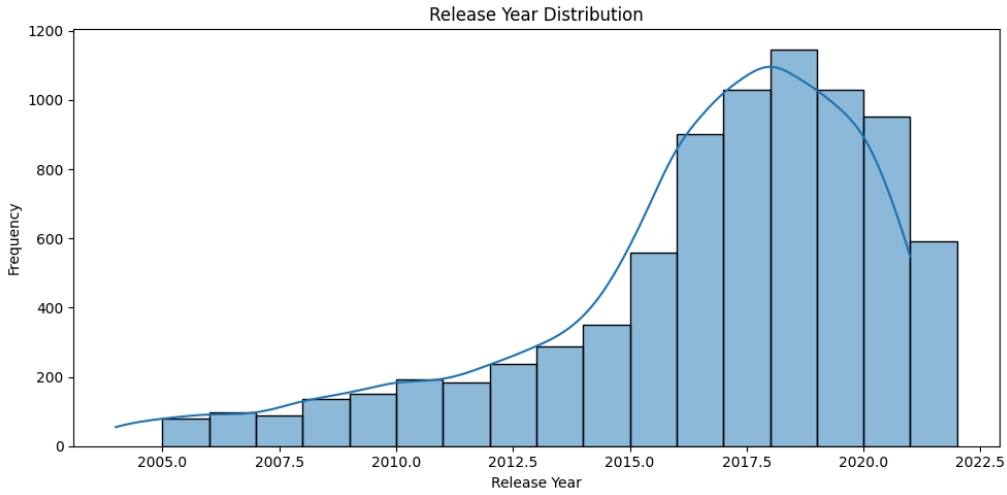


The dataset contains two main types of content, Movies and TV Shows. As shown in the figure above, the distribution is significantly skewed towards movies. This indicates that Netflix's catalog, is more heavily populated with movies than TV Shows.The imbalanced distribution of content types may influence the recommendation system's behavior.

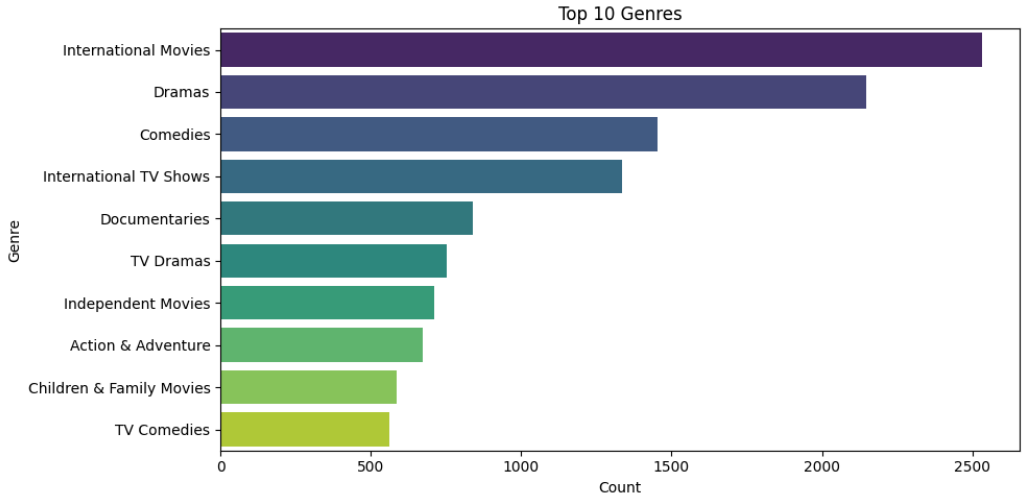


The dataset shows that Netflix’s catalog is predominantly composed of content rated TV-MA and TV-14, which together account for the largest share of titles. This indicates that Netflix primarily targets mature audiences and older teenagers. Other ratings such as TV-PG, R, and PG-13 also appear frequently, representing content suitable for general audiences with parental guidance or restricted for certain age groups. In contrast, family-oriented ratings like TV-Y, TV-G, and the unrated category (NR) are

comparatively less represented, suggesting that children's programming constitutes a smaller fraction of the overall content.



The release year distribution shows a steady increase in the number of titles added to Netflix's catalog over time, particularly from 2010 onward. The number of releases peaked around 2018 and 2019, indicating the platform's aggressive content acquisition and production during that period. After 2020, a slight decline is observed, which may be attributed to production delays or market shifts.



The dataset shows that International Movies represent the most frequent genre on Netflix, indicating the platform's strong focus on global content. This is followed by Dramas and Comedies, which remain popular across various audiences due to their broad appeal. Genres such as International TV Shows, Documentaries, and TV Dramas also hold a significant portion. In addition, genres like Independent Movies, Action & Adventure, Children & Family Movies, and TV Comedies contribute to the platform's

variety, offering content that appeals to the niche audiences that are seeking lighter entertainment.

METHODOLOGY

I. Overview

This chapter outlines the methodology used in developing a content-based recommendation system for Netflix movies and TV shows. The development process consists of several stages, including data collection, data preprocessing, feature extraction, model development, object-oriented programming, model persistence, and model deployment.

II. Data Collection

The dataset used in this project was obtained from kaggle, titled *Netflix Movies and TV Shows*. It contains metadata for 8807 titles available on Netflix, including attributes such as show id, type, title, director, cast, country, date added, release year, rating, duration, listed in, and description.

III. Data Preprocessing

The dataset that is imported from kaggle needs to be further processed to make the data cleaned from any kind of anomalies. The steps that performed included:

- Removing Outlier :

  The release year column was analyzed using IQR method to remove any outliers.

- Removing Anomaly :

  Upon further inspection using the value_counts() method, we identified several anomalies in the rating column, where duration values were mistakenly included. To address this issue, we utilized the isin() method in combination with boolean indexing to remove these incorrect entries from the dataset.

- Removing Unused Values :

  As the scope of this study is limited to TV Shows, all records in the type column that do not correspond to “TV Show” were excluded from the dataset.

- Handling Missing Values :

  Upon further inspection, we discovered that several important columns in the dataset such as director, cast, country, and rating contained a significant number of missing values. To ensure the integrity of the dataset and to avoid introducing bias, we first assessed the extent of missing data in each column. For columns with a manageable number of missing entries, such as director, cast, and

country, we filled the missing values with ‘Unknown’. This approach preserves the structure of the data without making unjustified assumptions. For the rating column, which contains only four missing values, we filled them using the mode in the column.

- Removing Unused Columns :

  Since the dataset has been filtered to consist only of TV Shows, the type column is no longer necessary. In addition, columns such as show id and date added, which are not relevant to the recommendation process, were also removed to reduce noise and improve model efficiency.

- Features Combination :

  We created a new column called combined_feature by joining important columns like title, director, cast, listed_in, and description. This combined text is used to compare items and find similar ones for content-based filtering.

IV. Feature Extraction

To enable comparison between titles based on textual data, the following techniques were used:

- TF-IDF (Term Frequency - Inverse Document Frequency)

  TF-IDF was used to convert the combined_features text into numerical vectors. This technique assigns weights to words based on how frequently they appear in a specific TV Show (Term Frequency) and how rare they are across the entire dataset (Inverse Document Frequency). As a result, more informative and unique words are given greater importance.

- Cosine Similarity

  After we generated the TF_IDF vectors, we used cosine similarity to measure how similar each pair of TV Shows is based on their content. The output is a similarity matrix, where each value represents the degree of similarity between two TV Shows, with scores ranging from 0 (no similarity) to 1 (very similar).

V. Model Recommendation Development

The recommendation model was built using a Content-Based Filtering approach. This method recommends shows that are similar in content to a user-specified title by analyzing textual features. The steps are as follows:

1. User Input :

   The user provides the title of a TV Show for which they want to receive similar recommendations.

2. Similarity Retrieval:

   The system uses a precomputed cosine similarity matrix, based on TF-IDF vectorization of textual features, to calculate similarity scores between the input title and all other TV Shows in the dataset.

3. Top-N Recommendation:

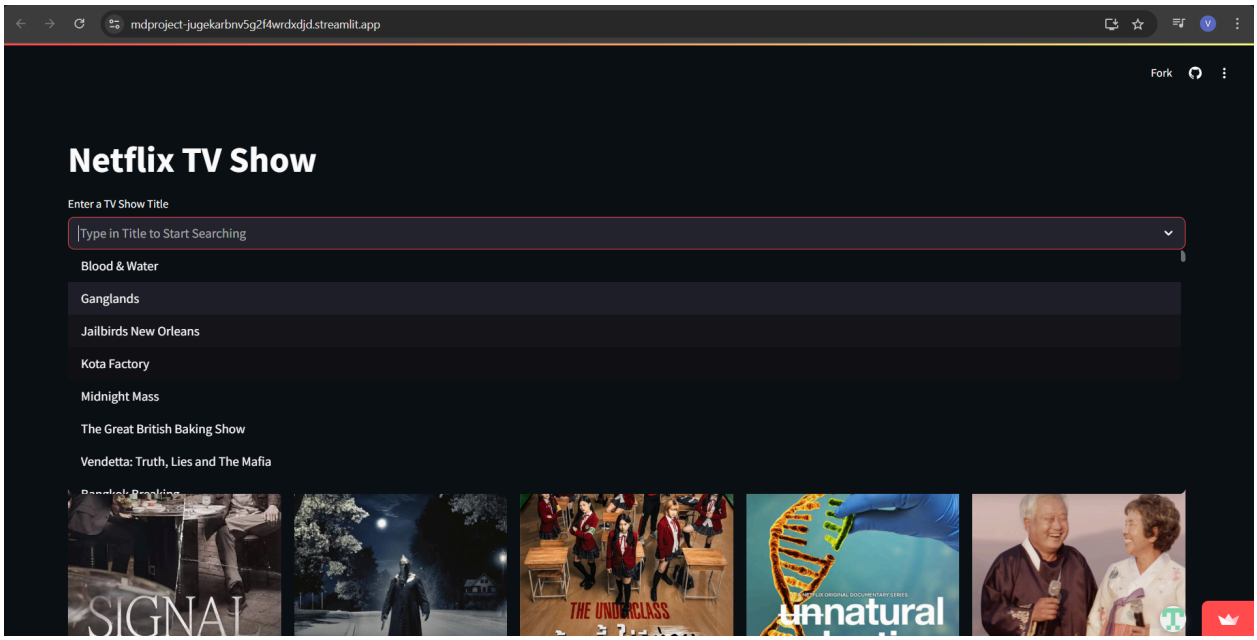
   The system selects and returns the top N TV Shows with the highest similarity scores excluding the input titles. These results serve as personalized content suggestions.

4. Function Implementation:

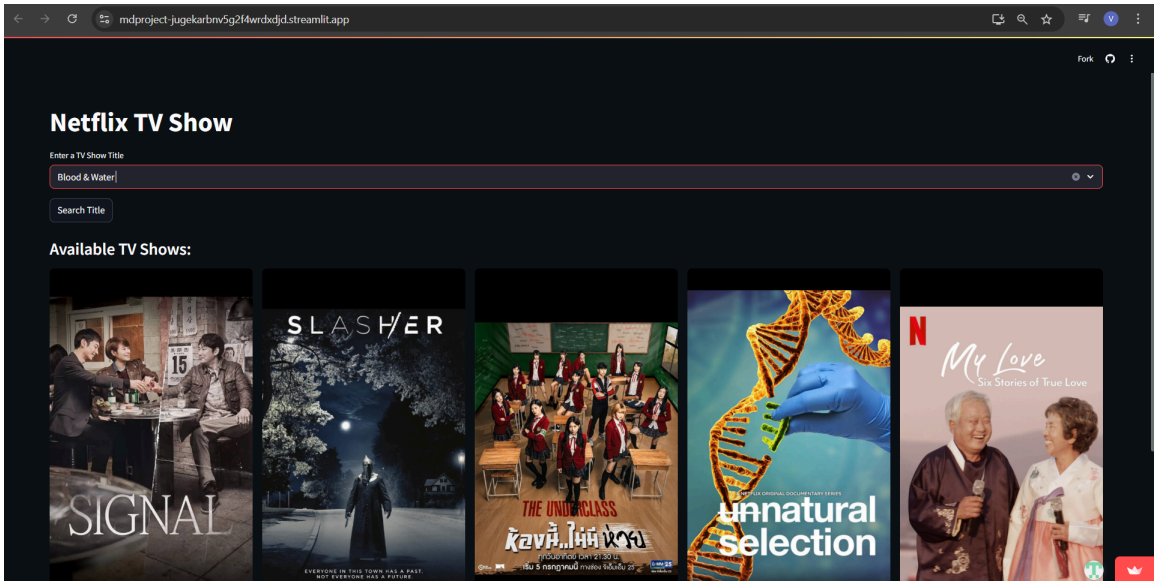
   This process is handled by a custom Python function named recommend(title, df, sim_matrix, top_n=5), which retrieves and returns five most relevant recommendations based on the given title.

RESULTS AND ANALYSIS

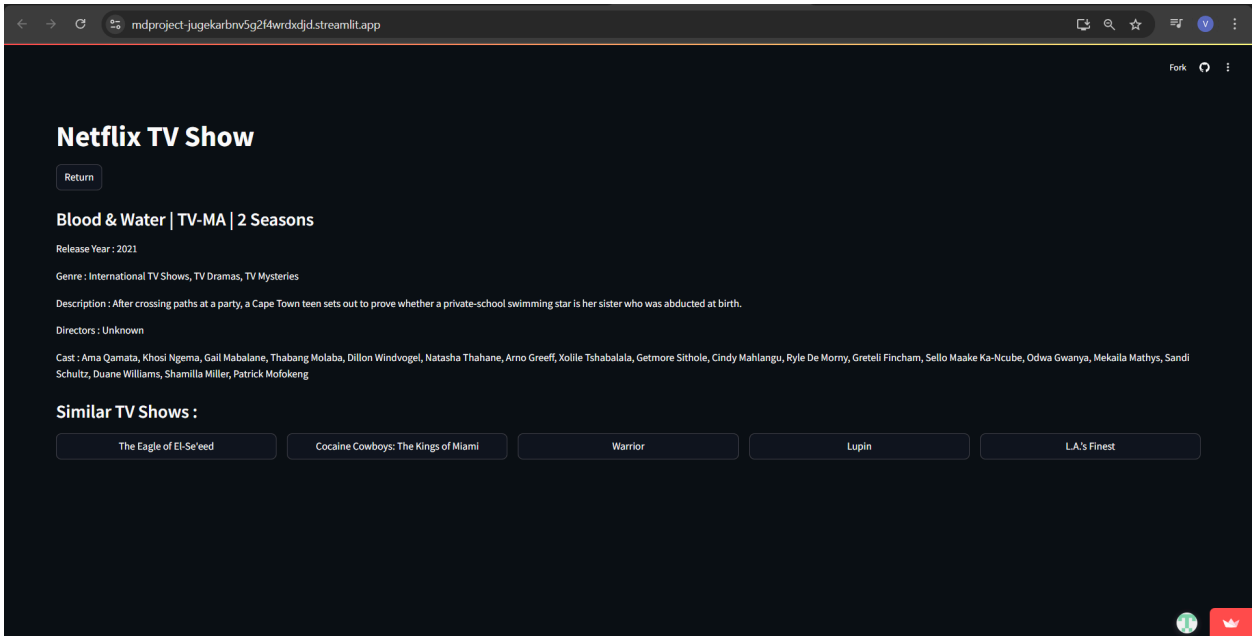
I. Results



In the Streamlit interface, users are provided with a scrollable list of TV show titles from which they can select one as input for the recommendation system and at the bottom we have the featured TV Shows.



For this case I chose one of the TV Shows titles named ‘ Blood & Water’ and clicked the search title.



In this section, we can see that the deployment was successful as it successfully suggests five titles with similar themes, genres, or content characteristics.

II. Analysis

The content-based filtering model shows solid performance in recommending TV Shows that share strong descriptive similarities with the user selected title. The following key points summarize the strengths and limitations of the model :

- Strengths :
    1. Accuracy of Similarity :

        The recommended titles such as *Cocaine Cowboys: The Kings of Miami* demonstrate strong alignment in terms of genre and description with the input titles. This indicated that the TF-IDF vectorization method is effective in capturing meaningful textual patterns and content-based similarities between TV Shows.

    2. Explainability :

        Because the recommendations are generated based on textual metadata (such as title, genre, description, cast, and director). Users can clearly understand why a particular TV show is recommended.

- Limitations :
    1. Lack of Personalization :

        The model has no consideration for user preferences or history, so every user receives the same TV Shows recommendation for the same input.

    2. Limited by Available Metadata :

        If the dataset has poor-quality descriptions, recommendations will be less accurate or even never be recommended.

DEPLOYMENT PROCESS

After the modelling and the development of the recommendation model has been done, the next step was to deploy the system, by making it accessible and usable by end-users. Where this deployment process is to transform the model prototype from ".ipynb" into a functional and interactive application. For this project, streamlit was the main framework for deployment, where by using Streamlit its ease to use when building applications.

I. Deployment Environment Setup

Because streamlit requires a specific environment, including necessary Python libraries and model files. We need to establish a suitable deployment environment, this has been done by installing essential Python libraries that serve as project dependencies, where we use **streamlit**, **pandas**, **numpy**, **scikit-learn**, and **joblib**. These were then listed in the **requirements.txt** file to ensure replication can be done. For the next part we want to have all the necessary model files that were generated during the modeling phase. In this case, **df.pkl** (containing the processed Netflix TV Show DataFrame) and **tfidf_matrix.pkl** (containing the TF-IDF matrix used for similarity calculations) that we want to load into the application. These files are stored within a different directory to maintain a clean structure for the project.

II. Streamlit Application Structure

The structure of the streamlit application is to ensure that users could see and have better interaction with the application not only on the recommendation part, but including the interface when first seeing the application. Here is the structure of the streamlit :

- Application Initialization :
  When first initializing the application, basic config was done using "st.set_page_config" where defining the browser title and the page layout. And for the main application title is displayed so users know what the application is about "st.title("Netflix TV Show").

- Model and Data Loading (**load_data()**) :
  The "load_data()" function is responsible for loading the assets for the application and the required recommendation system. This function loads **df.pkl**, which contains the TV Shows data, and **tfidf_matrix.pkl**, the numerical data of text features.

Initially, our team encountered a challenge when first attempting to directly save the **cosine_similarity** matrix into a pickle file and due to its

exceptionally large size, which could lead to performance issues and the uploading limits on the github platform. To overcome this, we decided to only store the **tfidf_matrix**. The **cosine_similarity** is then computed using **cosine_similarity(tfidf_matrix, tfidf_matrix)** after the **tfidf_matrix** is loaded within the Streamlit application. And this approach ensures efficient initialization without changing the recommendation accuracy.

- Recommendation Function (**recommend()**) :
  For the core function we have the **recommend(title, top_n=5)**, where it takes a TV Show title as input, and would retrieve its index in the dataset, and after that similarity score was calculated with all other TV Shows based on its **cosine_sim**. The output then turns into a list of top 5 (as per **top_n**) most similar TV Shows. And to ensure the function runs smoothly error handling was included for cases where the title is not found in the dataset, or when there is no recommendation for a TV Show title.

- Interactive User Interface (UI) :
  To help users when using the interface, we have designed the Streamlit application to be as easy to use as possible where users can search for TV Shows via an **st.selectbox**, with autofill options for the available title list. Then the "Search title" button where it would trigger the search process. The applications then would feature a view state.

  Search view. That would display the search box and the list of TV Shows as initial recommendations. Or available TV Shows so users can try the first TV Shows that they see.

  Detail view. Once a user has selected a title, then the application would display the details of that TV Show (rating, duration, release year, genre, description, director, and cast). And below the details would be the Recommended TV Shows based on the user's input. Using **st.columns** it helps to lay-out the recommendations and it can help users read and add bonus posters for the TV Shows.

III. Deployment Steps to Streamlit Platform

After Streamlit application development has been done locally, we want to deploy it to the Streamlit online platform to make it publicly accessible. The steps is :

- GitHub Repository : All the code from modelling to Streamlit application, requirements.txt file, and assets (pickle and assets) were uploaded to the GitHub repository.

- Streamlit Platform : The GitHub repository then connected with Streamlit Platform, where it would automatically detect the Streamlit application and initiate the build process.
- Application Initialization : After that Streamlit platform would read the **requirements.txt** to install all the necessary dependencies then execute the main application script. The application can be accessed via the following URL : Streamlit application.

CONCLUSION

With the success of developing and deploying a content-based filtering recommendation system for Netflix TV Shows. The system has addressed the challenge of content overload on streaming platforms by suggesting relevant TV Shows based on content features for example genres, description, casts, and directors. The other significant challenge encountered during deployment was the large file size of the "**cosine_similarity**" matrix, which has limit streamlit and github movement, this has been effectively overcome by only storing the "**tfidf_matrix**" and dynamically computing"**cosine_similarity**" at runtime within the Streamlit application, to ensure efficient initialization without lowering the recommendation accuracy. This highlighted the part that as a data scientist we need to learn how to optimize resource management in deployment. And with that future enhancements could explore integrating collaborative filtering techniques for a more personalized user experience, and optimizing the system for larger datasets.