Netflix Dataset Analysis & Recommendation System

Introduction

This wiki entry explores a sophisticated hybrid recommendation system for Netflix movies and TV shows. The system combines multiple advanced approaches to provide accurate and diverse content recommendations:

- **Content-Based Filtering** (Text Similarity using TF-IDF)
- **Collaborative Filtering** (User Preferences using SVD)
- **Graph-Based Learning** (Node2Vec for network-based recommendations)

In an era where vast amounts of data are generated daily, Natural Language Processing (NLP) plays a crucial role in analyzing and understanding textual information. By leveraging NLP techniques, we can extract meaningful patterns from descriptions, genres, and cast details to enhance recommendation accuracy and provide more human-like content suggestions.

The dataset used in this analysis comes from Netflix's movie and TV show listings. This dataset contains metadata about various titles available on the Netflix platform, including their type (Movie or TV Show), release year, director, cast, country of origin, duration, and genre classification. The goal of this wiki is to explore and analyze the dataset while ensuring data quality through appropriate cleaning and preprocessing steps.

About the Dataset

The dataset (`netflix_titles.csv`) consists of 12 columns that provide details about each Netflix title:

Column Name	Description
show_id	A unique identifier for each movie or TV show. It starts with an "s" followed by a number (e.g., `s1` , `s2`).

type	The category of the content, either `Movie` or `TV Show`.
title	The name of the movie or TV show.
director	The name of the director(s) of the movie or TV show. If multiple directors exist, they may be listed as a comma-separated string.
cast	The names of the main actors in the movie or TV show, separated by commas. Some entries might be missing if no cast information is available.
country	The country where the movie or TV show was produced. If multiple countries were involved, they might be listed together.
date_added	The date when the movie or TV show was added to Netflix, usually formatted as `Month Day, Year` (e.g., `September 25, 2021`).
release_year	The year when the movie or TV show was originally released.
rating	The content rating, such as `PG-13`, `TV-MA`, `R`, `G`, etc. It indicates the appropriate audience age group.
duration	The length of a movie (e.g., `90 min`) or the number of seasons for a TV show (e.g., `2 Seasons`).
listed_in	The genres or categories assigned to the movie or TV show. Multiple genres might be included, separated by commas (e.g., `Documentaries, Drama`).
description	A short summary or synopsis of the movie or TV show.

Prerequisites

Before working with this project, readers should have a basic understanding of:

1. **Python & Pandas** - Handling tabular data (`DataFrame` operations).
2. **Machine Learning Concepts** - Knowledge of **TF-IDF, cosine similarity, and SVD** for recommendations.
3. **Graph Theory** - Understanding **network-based learning** (e.g., **Node2Vec**).
4. **Basic NLP (Natural Language Processing)** - Using **textual data** for similarity analysis.

Data Cleaning & Preprocessing

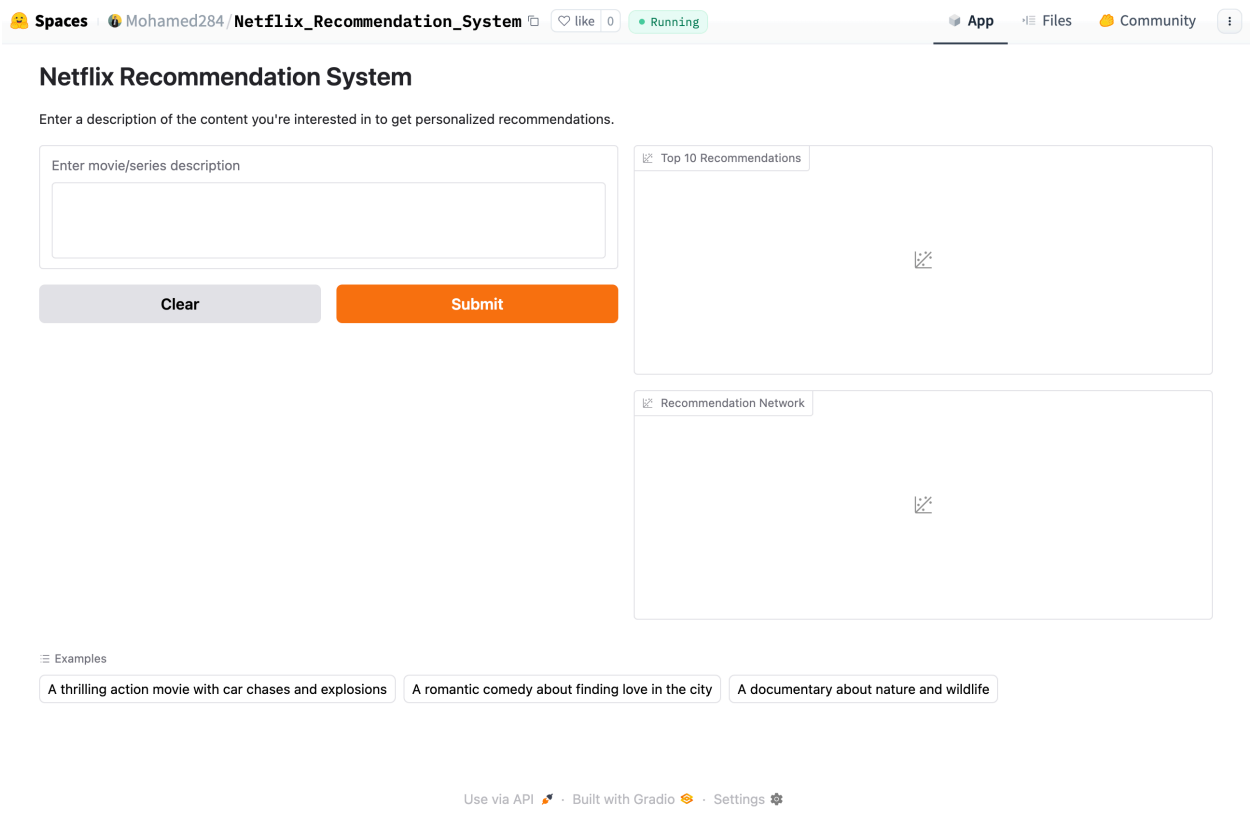
Before diving into any analysis, it is essential to clean the dataset by handling missing values, ensuring consistent formatting, and addressing potential inconsistencies. This step will enhance the reliability of our insights. The primary steps involved in the cleaning process include:

- **Handling Missing Values**: Many fields, such as director and cast, have missing values that have been addressed through imputation or removal.
- **Standardizing Date Formats**: The date_added column have been used to extract Month and Year to have a standard format for comparing.
- **Fixing Data Types**: The release_year column needs to be an integer.

Interactive Demo

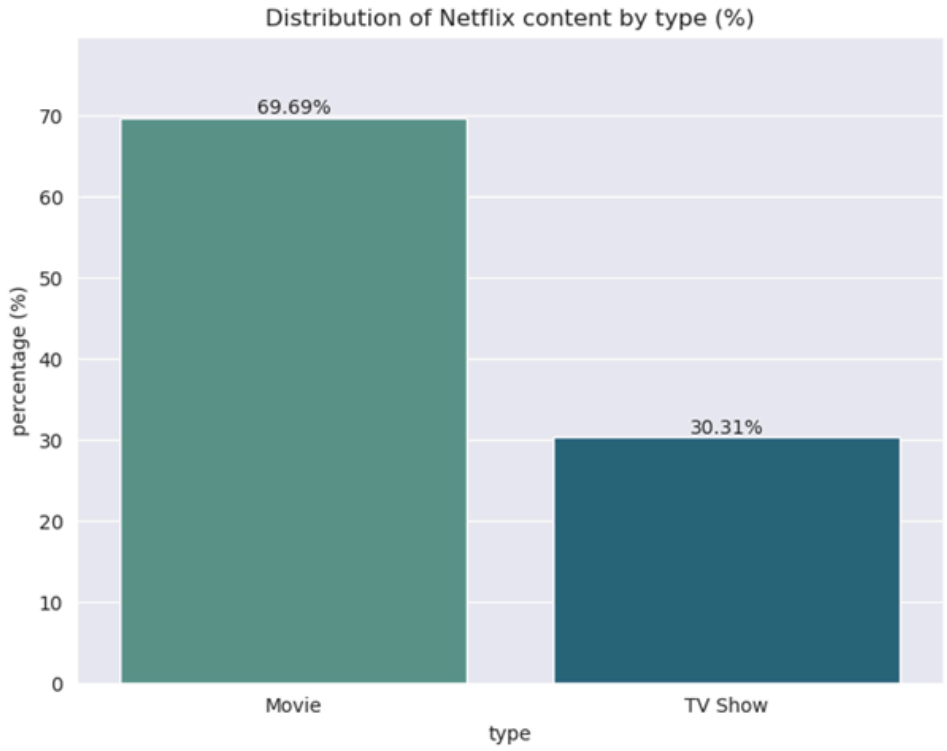
Explore the interactive demo of the Netflix Recommendation System on Gradio:

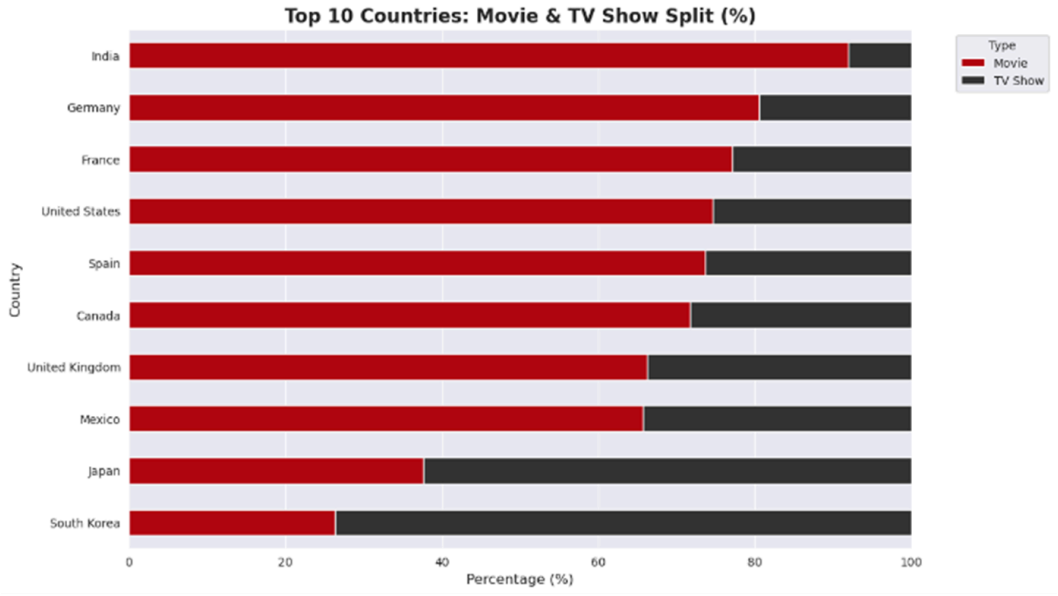
[Gradio Demo]



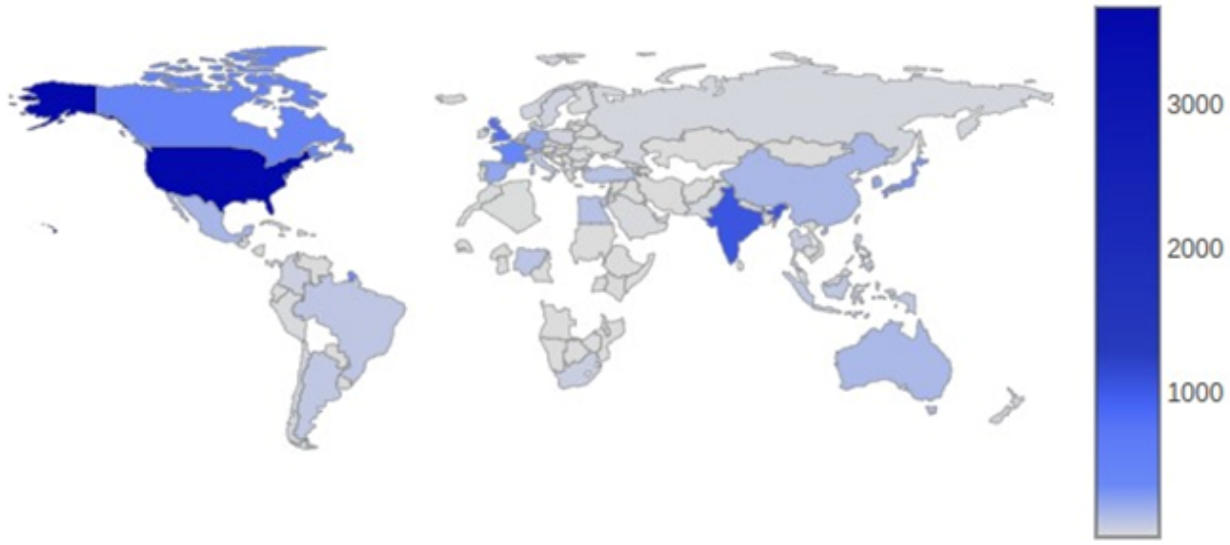
Exploratory data analysis (EDA)

By taking a first insight on the data, it can be observed that Netflix's content is composed of approximately 70% movies and 30% TV series. However, there are clear differences among different countries. For instance, the US has a balanced distribution, whereas India heavily favors movies. Other countries like Japan or South Korea have more balanced or series-focused distributions.

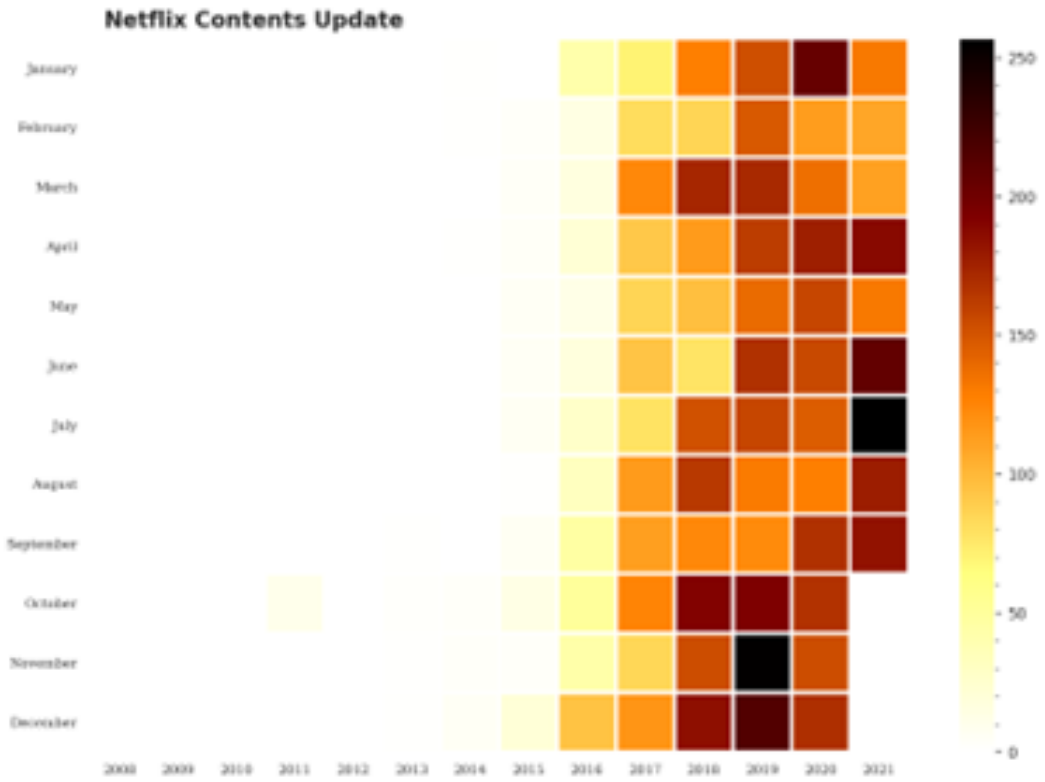




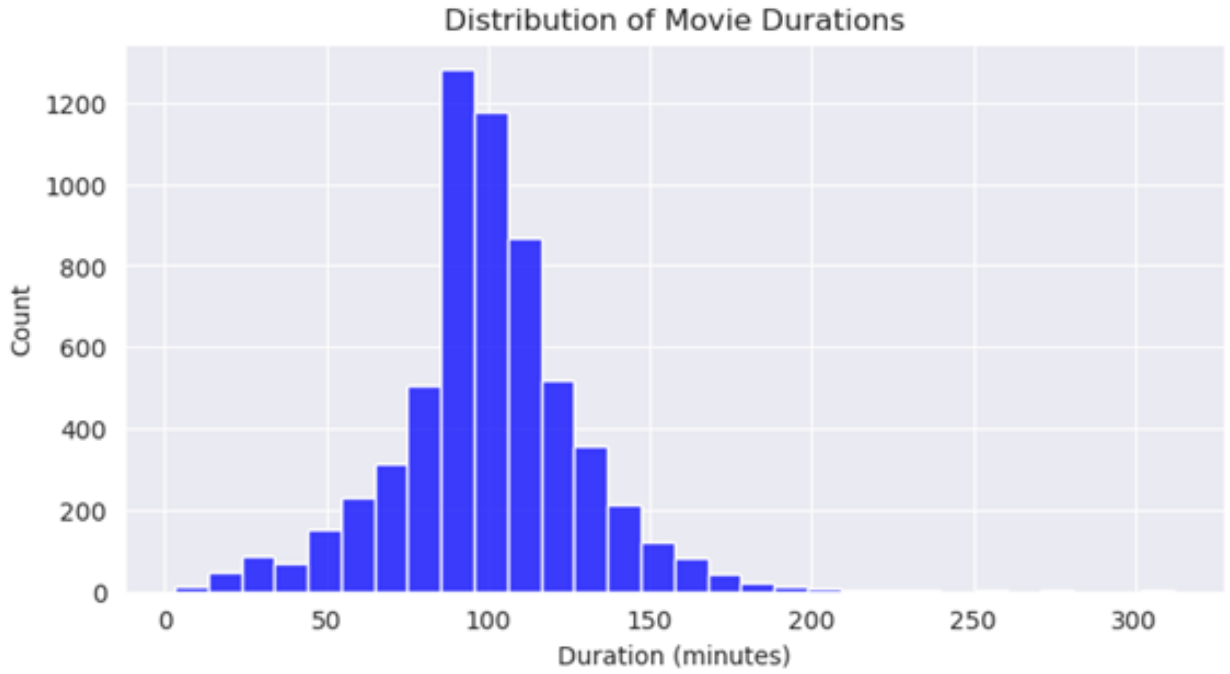
By looking at the overall number of contents by country, the United States highest amount of available titles, followed by India, the UK, and South Korea.

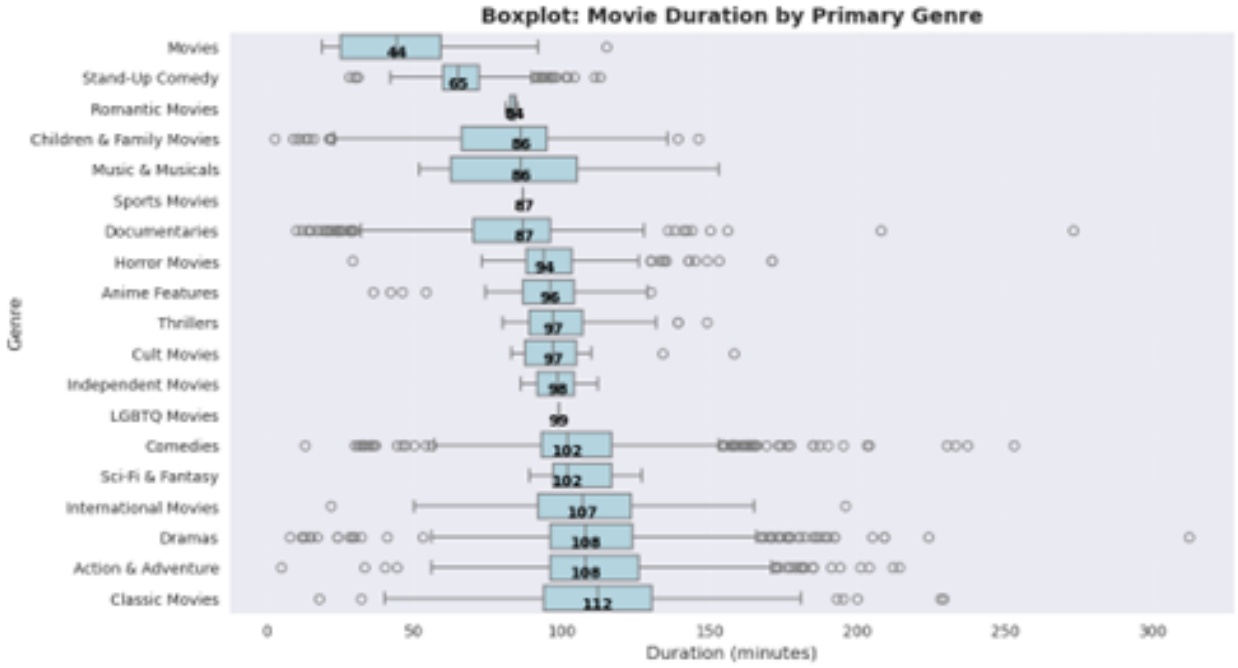


An important trend is the frequent updating of content of Netflix, which significantly expanded its library between 2015 and 2021.



The duration of Netflix movies generally clusters around standard feature lengths. These lie mostly between 90 and 110 minutes. Shorter films under 80 minutes and longer ones above 130 minutes are less frequent. Movie durations also vary significantly across genres, with dramas and action films typically being longer, while comedies and family-oriented movies are usually shorter.





Methodology

Content-Based Filtering

The content-based filtering approach utilizes TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to analyze content features:

TF-IDF Vectorization

TF-IDF is a numerical statistic that reflects the importance of a word in a document collection:

- Term Frequency (TF): Measures how frequently a term appears in a document
- Inverse Document Frequency (IDF): Downweights terms that appear in many documents

Implementation details:

```
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(df['combined_features'])
```

Cosine Similarity

Cosine similarity measures the similarity between two vectors by computing the cosine of the angle between them:

- Range: [-1, 1] where 1 means identical direction, 0 means orthogonal, -1 means opposite
- Used to compare TF-IDF vectors of different content items

```
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

Collaborative Filtering

The collaborative filtering component employs matrix factorization techniques to:

User-Item Matrix

- Creates a matrix of user ratings for items
- Handles sparsity through matrix factorization
- Simulates user behavior patterns

```
def create_user_item_matrix(df, n_users=1000):
    np.random.seed(42)
    n_items = len(df)
    user_item_matrix = np.random.randint(0, 6, size=(n_users, n_items)) * \
                       (np.random.random((n_users, n_items)) > 0.8)
    return user_item_matrix
```

Matrix Factorization

- Singular Value Decomposition (SVD) for dimensionality reduction
- Captures latent features in user-item interactions
- Predicts missing ratings

```
def matrix_factorization(ratings, n_factors=50):
    user_ratings_mean = np.mean(ratings, axis=1)
    ratings_norm = ratings - user_ratings_mean.reshape(-1, 1)
    U, sigma, Vt = svds(ratings_norm, k=n_factors)
    predicted_ratings = np.dot(np.dot(U, np.diag(sigma)), Vt) + \
                        user_ratings_mean.reshape(-1, 1)
    return predicted_ratings
```

Node Representation Learning

Implements graph-based learning using Node2Vec:

Content Graph Creation

- Builds a graph representing content relationships
- Nodes represent movies/shows and genres
- Edges represent content-genre associations

```
def create_content_graph(df):
    G = nx.Graph()
```

```
    # Pre-process genres
    genre_dict = {}
    for idx, row in df.iterrows():
        if isinstance(row['listed_in'], str):
            genres = [g.strip() for g in row['listed_in'].split(',')]
            genre_dict[idx] = genres

            # Add unique genres as nodes
            for genre in genres:
                if not G.has_node(genre):
                    G.add_node(genre, type='genre')
    return G
```

Node2Vec Algorithm

- Random walk-based approach for learning node embeddings
- Preserves network neighborhood information
- Parameters:
  - Dimensions: 32 (embedding size)
  - Walk length: 10 (steps per walk)
  - Number of walks: 50 (walks per node)

Hybrid Recommendation Function

Combines multiple recommendation approaches:

Weighted Combination

- Content similarity: 70% weight
- Node embeddings: 30% weight
- Adaptive weighting based on availability

```
def get_hybrid_recommendations(query, cosine_sim, df, n_recommendations=
10):
```

```
    """
    Get hybrid recommendations based on content similarity and node embeddi
ngs.

    Args:
        query (str): Title or description to base recommendations on
        cosine_sim (np.ndarray): Pre-computed cosine similarity matrix
        df (pd.DataFrame): DataFrame containing Netflix content
    """
```

Results Visualization

Recommendation Scores Bar Chart

Visualization of top 10 recommendations with their similarity scores:

Top 10 Recommendations: Bar chart visualization showing similarity scores for the most relevant content recommendations based on the hybrid recommendation

system.

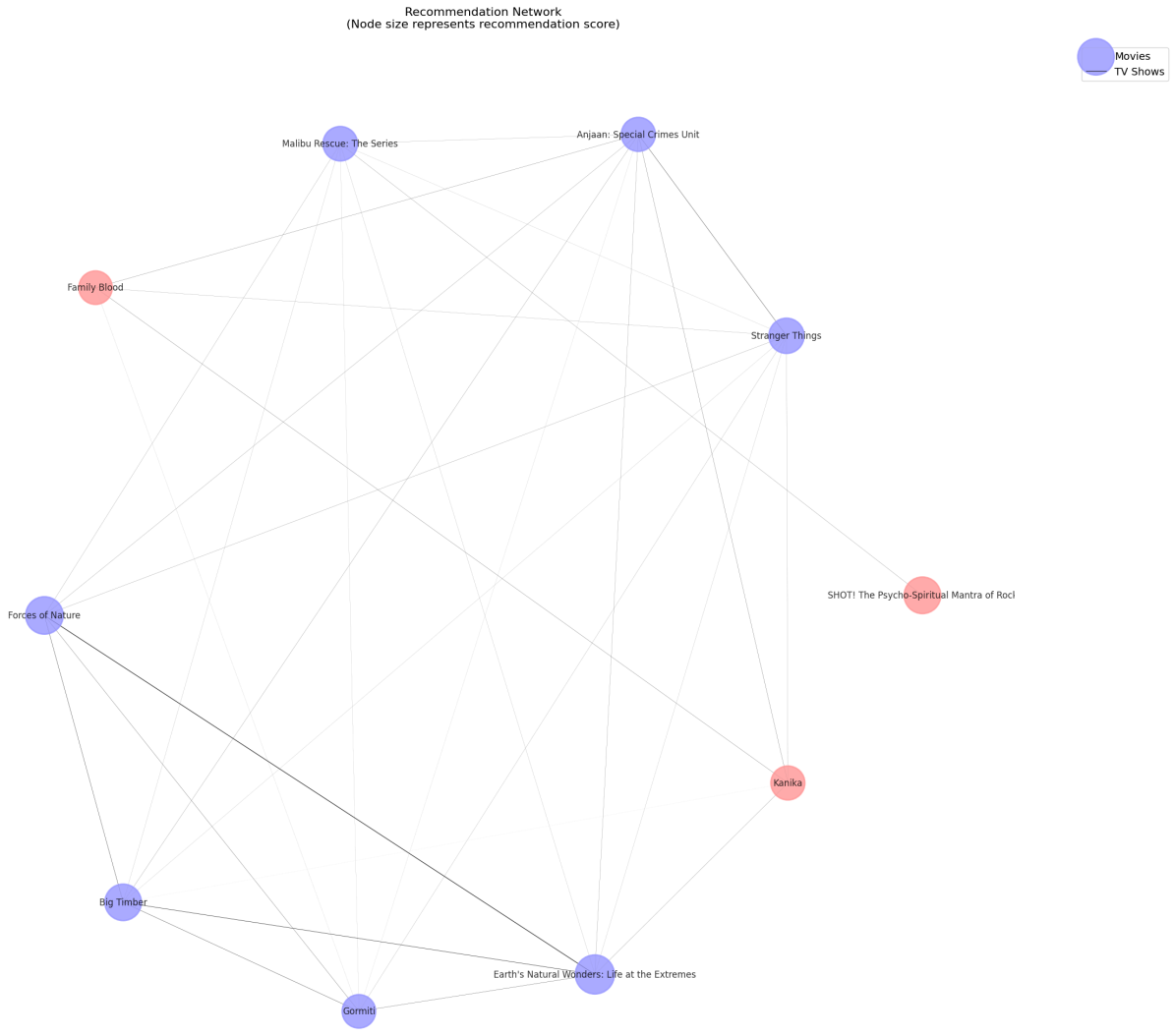
Network Analysis

The recommendation network analysis reveals:

- Number of recommended items: 10
- Number of connections: 45
- Network density: 1.000

Recommendation Network Graph

Visualization of content relationships and similarities:



Recommendation Network: Graph visualization depicting content relationships, with nodes representing recommended items and edges showing content similarities between them.

The network graph shows:

- Nodes: Recommended content items
- Node size: Recommendation score
- Edges: Content similarities
- Edge thickness: Similarity strength

Strengths and Challenges

Natural Language Processing (NLP) can have valuable capabilities. Netflix descriptions typically contain objective plot summaries and central themes. This neutrality allows NLP algorithms to categorize and also systematically analyze films efficiently based on genre, directors, or key plot elements. As a result, NLP can automate the organization of extensive libraries, which reduces the manual effort required to maintain and update catalogs. Additionally, the hybrid recommendation system benefits from a multi-modal approach. This is done by combining different aspects like content-based and collaborative filtering, as well as graph-based learning. Overall, all of these factors can provide more robust recommendations compared to single-approach systems.

More advantages are given by the fact that NLP techniques enhance recommendation systems by detecting similarities among movie descriptions. Even brief, neutral plot summaries contain sufficient signals that algorithms can interpret to identify closely related content, which allows us to deliver more accurate recommendations. NLP also supports improved search capabilities by not only relying on explicit keyword matching but also understanding context. This can enhance the access of users to movies and TV shows that match specific genres or themes. The scalability of NLP approaches further supports its practical application. Methods like TF-IDF vectorization efficiently handle large text datasets. Matrix factorization reduces dimensionality for faster processing, and pre-computed similarity matrices enable a quick recommendation generation. Furthermore, the feature-rich analysis incorporates multiple content aspects (description, cast, director, genres, title) and considers both explicit content details and implicit user behavior patterns. This shows, that network-based analysis captures complex content relationships.

However, applying NLP to Netflix movie descriptions introduces certain challenges. One issue is the simplification of whole movies into concise summaries. Movies, which are sometimes seen as complex artistic creations, have nuanced storytelling elements or stylistic choices that brief textual descriptions often fail to capture fully. Reducing films to a few descriptive sentences can lead NLP algorithms to overlook the complexity, which leads to misrepresenting narratives based only on superficial textual similarities.

Additionally, the concise and neutral nature of these descriptions can limit NLP's analytical depth. Short descriptions might ignore important contextual details which are essential for interpretations. This leads algorithms to draw incorrect connections or fail to distinguish between distinct narrative themes. For instance, two films described simply as "a soldier needs to overcome challenges in a warzone" could differ dramatically in tone, perspective, or narrative complexity but still appear highly similar to each other in an NLP system due to their concise textual summaries.

Moreover, NLP-based recommendation systems face some sort of “cold-start problem”. This means, that new content items initially lack user interaction data, which causes reliance on content-based features. This results in limited personalization for new users. Computational resources also pose a challenge, because large similarity matrices require significant memory. Based on this, generating network graphs is computationally intensive, which makes real-time updates difficult. Data quality dependencies also can affect system performance, as incomplete or inconsistent metadata can degrade recommendation quality and limit handling of multilingual content. Finally, balancing recommendation accuracy and diversity remains a critical trade-off, which poses further complexity for system design.

Ethical considerations

User privacy and data protection must be ensured by anonymizing user data and handling it securely to prevent privacy breaches. Recommendations should be age-appropriate by respecting content ratings and avoiding the suggestion of mature content to younger audiences. Users must provide informed consent by being made aware of how their data is collected and used for recommendations. Cultural sensitivity should be prioritized by ensuring that recommendations are diverse and inclusive, catering to different backgrounds and perspectives. Regulatory compliance must be maintained by adhering to legal frameworks such as GDPR and COPPA when handling user data