A Multi-Method Content-Based Recommendation System for Netflix Data

Dheer Gupta
Vyaas Subramanian
Muhammad Yahya
March, 2025

Abstract

Recommender systems have become integral to online platforms such as Netflix, Amazon Prime, and Hulu, providing tailored suggestions to users from extensive content catalogs. While collaborative filtering often headlines the research in recommender systems, content-based approaches remain vital in situations with limited user-user overlap or when item features can strongly signal user preferences. Recent developments suggest combining multiple content-based signals—such as item similarity, classification-based "like" prediction, clustering, and association rule mining—can yield a more robust personalized experience.

Hence, our research question is: *How can we develop a multi-faceted, purely content-based recommendation system for Netflix data that effectively leverages user likes/dislikes and advanced methods (kNN, Naïve Bayes, clustering, and association rules) to produce high-quality suggestions?*

1 Introduction

Netflix, like many other streaming services, provides basic recommendations based on algorithms that primarily focus on genre, viewing history, or user ratings. However, these systems tend to offer limited and often generalized suggestions that may not accurately reflect the unique tastes of each user. Users often find themselves scrolling endlessly through an overwhelming selection of content, unsure of which movies or TV shows to watch next. This issue is further compounded by the fact that most recommendation systems fail to account for subtle and diverse factors, such as specific genres, IMDB ratings, actors, or production companies, that could significantly enhance the personalization of the recommendations. The importance of an advanced recommendation system is clear, especially as competition within the streaming industry increases. A more personalized and effective recommendation engine could not only enhance user satisfaction but also improve engagement and retention for platforms like Netflix. Previous research in the field of recommendation systems has highlighted the importance of using more than just basic genre filtering to make personalized suggestions. For instance, hybrid approaches that combine collaborative filtering (which uses user behavior data) with content-based filtering (which uses item attributes like genre or director) have shown improved performance in generating relevant recommendations. Despite this, the lack of deep personalization based on granular user preferences remains a challenge. This paper seeks to address that challenge by developing a recommendation engine that leverages both user input and detailed content attributes such as genre, IMDB rating, and production company to

generate tailored suggestions. Specifically, the research question guiding this study. To answer our question, we have developed a more advanced recommendation engine using a dataset from Netflix, which includes a variety of content, such as movies, TV shows, and anime, from numerous genres. The dataset offers detailed content attributes, including genre, IMDB ratings, release year, and more, which provide the necessary information for making tailored recommendations. To gather additional user preferences, a Google form was created, allowing users to specify their likes and dislikes, genre preferences, and minimum IMDB rating. The study applies machine learning techniques, including cosine similarity, k-Nearest Neighbors (kNN), and ensemble methods, to predict the most relevant content based on both user input and content features. The findings from this research reveal that incorporating both granular user preferences and detailed content attributes allows the recommendation engine to generate more personalized and accurate suggestions compared to traditional models. Ultimately, the results show that a deeper level of personalization—considering specific tastes, ratings, and genres—leads to a significantly enhanced user experience. This paper concludes that future recommendation systems must incorporate these nuanced factors to create more engaging and user-centric platforms.

2 Literature Review

2.1 Early Recommendation Systems

Initial recommender systems leaned heavily on collaborative filtering (1; 2), deriving suggestions from user-user similarities. However, collaborative filtering struggles with the "cold start" problem (3) when new users or items have insufficient interactions. Meanwhile, content-based approaches (4; 5) overcame this by matching item attribute vectors to each user's profile of previously liked items.

2.2 Hybrid & Multi-Signal Approaches

As recommendation research evolved, hybrid methods (6; 7) combined both collaborative and content-based, or integrated multiple content-based signals. Some authors introduced classification models (Naïve Bayes, SVM) to label items as "likely liked," e.g. (8). Clustering such as K-Means or hierarchical clustering helps discover latent item groups (9). kNN item retrieval (10) can quickly find top local neighbors. Additionally, association rule mining (11) has proven valuable in discovering frequent co-attributes within item metadata.

2.3 Association Rule Mining for Content

Apriori, initially for market basket data (11), can detect frequent itemset patterns. In a recommendation context, each item's attribute set (particularly genres) can be treated as a transaction (12). Identifying co-occurring attributes ("Independent Movies" $\rightarrow$ "Dramas") can refine the recommendation process by giving these patterns a small "bonus" during ranking.

2.4 Our Position

Motivated by these approaches, we propose a purely content-based system merging multiple signals:

- Cosine-based content similarity,
- kNN retrieval for local neighbors,
- Naïve Bayes classification for synthetic likes,
- K-Means clustering for user cluster preference,
- Apriori association rules for discovering genre synergy,

all integrated in a final ensemble. While we do not incorporate user-user collaborative data, we mitigate cold-start issues and generate robust, varied recommendations by fusing these distinct content methods.

3 Data and Analysis

3.1 Dataset Collection

We used a Netflix dataset from Kaggle (13) of approximately 8,797 entries. Each row is labeled "Movie" or "TV Show" and includes:

- Basic Info: `show_id`, `title`, `type`, `date_added`, `release_year`, `rating`, `description`
- Categorical: `country`, `director`, `cast`
- Genre(s): originally in a single column `listed_in`

Missing IMDb ratings were fetched via the OMDb API (14). Table 1 provides a summary.

Characteristic	**Value**
Total Titles	∼8,797
Movies vs. TV Shows (%)	∼70% Movies, 30% TV
Mean IMDb Rating	∼6.4
Earliest Release Year	1925
Latest Release Year	2021

Table 1: *Basic Netflix Dataset Characteristics (example).*

3.2 Data Cleaning & Representation

We replaced missing `director`, `cast`, or `country` with "Unknown" and used the mode to fill missing `rating`. We dropped rows missing `date_added`. For `duration`, we extracted `duration_minutes` if the item is a Movie (e.g. "90 min"), or `duration_seasons` if the item is a TV Show (e.g. "2 Seasons"). Any remaining null IMDb ratings were replaced by the median (6.4).

3.3 Data Visualization

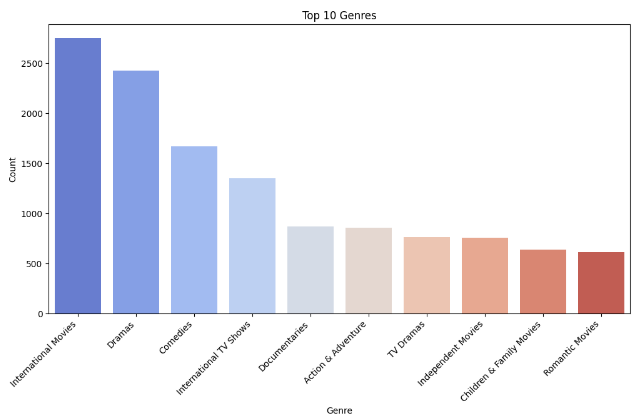

Figure 1: *Top 10 Most Popular Genres*

This bar plot clearly shows that International Movies, Dramas, and Comedies make up the bulk of Netflix's content library, with International Movies significantly leading the count. This suggests that Netflix offers a large volume of content from different countries, and genres like drama and comedy are particularly dominant in its catalog.

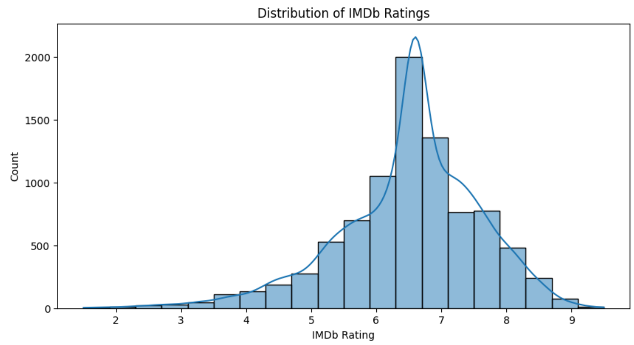

Figure 2: *Distribution of IMDb Ratings*

This histogram visualizes the distribution of IMDb ratings for the movies and TV shows in our dataset. The majority of titles have ratings between 5.5 and 7.5, with a noticeable peak around 6.5 to 7.0, indicating that most content on Netflix is rated in the mid-to-high range. The distribution follows a slightly right-skewed pattern, with fewer titles receiving very low or very high ratings. The overlaid density plot further highlights the central tendency of the dataset, helping us understand how IMDb ratings are distributed across different titles.

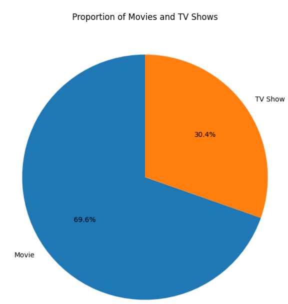

Figure 3: *Proportion of Movies and Tv Shows*

Another important feature of the dataset is the content type—whether the content is a movie or a TV show. By examining the proportion of movies and TV shows in the dataset, we can better understand the balance of content that Netflix offers to its users. As illustrated by the pie chart, 69.6 percent of the dataset consists of movies, while 30.4 percent comprises TV shows.

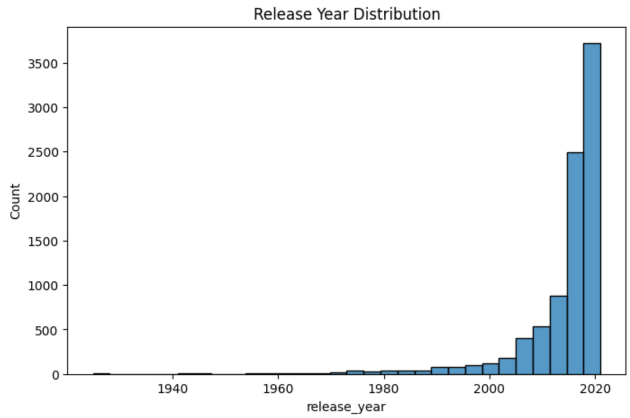

Figure 4: *Proportion of Movies and Tv Shows*

This histogram shows that the majority of Netflix's content has been released since 2010, with a dramatic rise in content production from 2020 onward. This trend is indicative of the significant increase in Netflix's library during the streaming boom of recent years, as well as the company's strategic push to produce more content in-house.

3.4 Feature Engineering

We split each item's `genre` string, then applied a `MultiLabelBinarizer` to produce a binary vector for each possible genre. We also scaled `release_year` to [0,1] using MinMaxScaler and stored it as `release_year_scaled`. We similarly scaled IMDb ratings to get `normalized_imdb`. The final "feature matrix" for each item is 47-dimensional (or so), containing `release_year_scaled`, `duration_minutes`, `normalized_imdb`, and binary genre flags.

4 Methods: Multi-Method Pipeline

4.1 Content-Based Similarity (Cosine)

We represent each user by the weighted average of vectors for the items they "like," weighting by each item's IMDb rating. Then for each candidate item, we compute:

$$\text{cosine\_similarity} = \frac{\langle user\_profile, item\_vector \rangle}{\|user\_profile\| \|item\_vector\|}.$$

To incorporate quality, we define an *adjusted_similarity*:

$$\text{adjusted\_similarity} = \alpha \times \text{cosine\_similarity} + (1 - \alpha) \times \text{normalized\_imdb}.$$

We typically set $\alpha = 0.7$. Items the user explicitly disliked or that fail the user's IMDb threshold are excluded, and the top 10 are returned.

4.2 kNN-Based Retrieval

Rather than compute similarity for *every* item, we also do a local search using `NearestNeighbors` with cosine distance.

1. Fit kNN on the entire feature matrix (8,797 items).
2. Query the user profile vector to find, say, the top-20 neighbors.
3. Recompute the same adjusted similarity formula for these neighbors, filter out disliked/low-IMDb items, then pick the top 10.

This can yield a slightly different set, sometimes more variety.

4.3 Naïve Bayes Classification

We define a "synthetic label" `like` $= 1$ if `imdb_rating` ≥ 7.5 else 0. We exclude user-labeled items from training (to avoid data leakage), then do an 80/20 split on the remaining items. Using *GaussianNB*, we get a probability *nb_score* $\in [0, 1]$ for each item. We reapply the trained model on the *full* dataset to produce an *nb_score* for all items.

4.4 K-Means Clustering

We tried $k = 3, 4, \ldots, 9$ with silhouette scoring. Suppose $k = 4$ was selected. Each item is assigned a cluster. We compute:

$$\text{cluster\_score} = \frac{\text{\# of liked items in that cluster}}{\text{\# of liked items total}}.$$

Hence, items in the same cluster as the user's liked items might get up to 1.0 if that cluster is entirely user-liked. This cluster_score helps promote items from the user's "favorite clusters."

4.5 Association Rule Mining (Apriori)

We treat each item's genres as a transaction. Using `min_support=0.05`, `min_confidence=0.6`, we find rules like ($\{\text{Independent Movies}\} \to \{\text{Dramas}\}$) with confidence ~ 0.78, lift ~ 2.82. If an item's genre set triggers a discovered rule, we may assign a small `association_bonus`=0.05.

4.6 Ensemble Score

To unify all signals, we define:

$$\text{ensemble\_score} = w_{adj}\,\text{adjusted\_similarity} + w_{nb}\,nb\_score + w_{cl}\,\text{cluster\_score} + \text{association\_bonus}.$$

We exclude disliked items, sort by ensemble_score, and show the top 10. The weights $(w_{adj}, w_{nb}, w_{cl})$ can be chosen (e.g., sum to 1.0) to balance quality vs. diversity.

5 Results and Discussion

5.1 Google Form Preferences Results

Category	**Selections**
Liked Movies	Don, Dragon Quest Your Story, Prom Night, Miss Virginia
Disliked Movies	Twogether, Brother
Preferred Genres	International Movies, Romantic Movies, Comedies
IMDb Minimum Rating	7.2
Content Preference	Movies Only

Table 2: *User Preferences for Movie Recommendations.*

Here are the results that were extracted from the Google Form that the user filled out. These results were with the help of Google API's: Form and Sheets. The app's script was used to extract random movies and tv shows from the data set so when a user fills out a form they will see a set of different titles each time.

5.2 Content-Based Retrival (Cosine Similarity)

Rank	Title	Type	IMDb Rating	Adjusted Similarity
1	Dick Johnson Is Dead	Movie	7.4	0.920910
2	Jaws	Movie	8.1	0.947406
3	Training Day	Movie	7.8	0.936165
4	InuYasha the Movie 2: The Castle Beyond the Looking Glass	Movie	7.3	0.917307
5	InuYasha the Movie 3: Swords of an Honorable Ruler	Movie	7.6	0.928512
6	InuYasha the Movie 4: Fire on the Mystic Island	Movie	7.2	0.913403
7	InuYasha the Movie: Affections Touching Across Time	Movie	7.2	0.913581
8	Schumacher	Movie	7.4	0.921120
9	Omo Ghetto: the Saga	Movie	7.6	0.928715
10	If I Leave Here Tomorrow: A Film About Lynyrd Skynyrd	Movie	7.8	0.935915

Table 3: *Top 10 Recommendations Based on Cosine Similarity.*

5.3 kNN-Based Retrieval

Rank	Title	Type	IMDb Rating	Adjusted Similarity
1	Django Unchained	Movie	8.5	0.962482
2	Inglourious Basterds	Movie	8.4	0.958737
3	Casino Royale	Movie	8.0	0.943736
4	GoldenEye	Movie	7.2	0.913734

Table 4: *Top kNN-Based Recommendations.*

5.4 Naïve Bayes Performance

Excluding user-labeled items from training, we got $\sim$ 84% accuracy, $\sim$ 0.54 precision, $\sim$ 0.59 recall, and 0.57 F1. Despite moderate performance on the synthetic "IMDb7.5" label, *nb_score* still offers a useful dimension for boosting items likely to be user "likes."

Metric	Score
Accuracy	0.84
Precision	0.54
Recall	0.59
F1 Score	0.57

Table 5: *Naïve Bayes Classifier Performance Metrics.*

5.5 Clustering Outcomes

5.5.1 Elbow Method for Optimal k

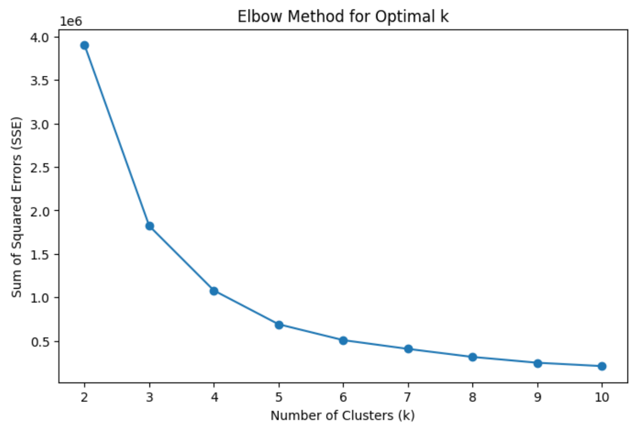

Figure 5: *Elbow Method for Optimal K Selection in K-Means Clustering*

5.5.2 Cluster Distribution and Silhouette Score

Number of Clusters (k)	Cluster Distribution	Silhouette Score
3	[2915, 4155, 1727]	0.665
4	[2807, 3650, 1326, 1014]	0.659
5	[2796, 1787, 571, 810, 2833]	0.649
6	[2779, 2045, 339, 667, 1778, 1189]	0.625
7	[2766, 2253, 656, 550, 1082, 1327, 163]	0.613
8	[2689, 1671, 710, 632, 201, 1137, 254, 1503]	0.610
9	[2689, 1573, 722, 632, 281, 1103, 254, 1503, 40]	0.610

Table 6: *K-Means Clustering Results for Different k Values.*

We found $k = 4$ gave a silhouette of around 0.659. Table 7 shows the cluster distribution. Each cluster had a distinct set of common genres and a different average IMDb rating.

Cluster	Size	Avg IMDb	Top Genres
0	2,807	6.92	{International TV Shows, TV Dramas, TV Comedies}
1	3,650	6.16	{International Movies, Dramas, Comedies}
2	1,326	6.66	{International Movies, Dramas}
3	1,014	6.42	{Documentaries, Stand-up Comedy}

Table 7: *Cluster distribution and top genres at $k = 4$.*

5.6 Final Recommendations

After computing ensemble scores, the top items typically balanced:

- `adjusted_similarity` (cosine with user profile, plus IMDb weighting),
- `nb_score` (Naïve Bayes probability),
- `cluster_score` (user's liked cluster fraction),
- `association_bonus` (0.05 if the item matches strong genre rules).

Table 8 illustrates sample final recommendations for a user with certain likes/dislikes.

Rank	Title	Type	IMDb Rating	Ensemble Score
1	Monty Python and the Holy Grail	Movie	8.2	0.904121
2	Monty Python's Life of Brian	Movie	8.0	0.901261
3	Cat on a Hot Tin Roof	Movie	7.9	0.899925
4	Ferris Bueller's Day Off	Movie	7.8	0.898450
5	Willy Wonka and the Chocolate Factory	Movie	7.8	0.898439
6	Bonnie and Clyde	Movie	7.7	0.897060
7	Rebel Without a Cause	Movie	7.6	0.895620
8	Enter the Dragon	Movie	7.6	0.895600
9	Dark Waters	Movie	7.6	0.895591
10	Forbidden Planet	Movie	7.5	0.894122

Table 8: *Final Top 10 Movie Recommendations Based on Ensemble Scoring.*

The user-labeled disliked items are excluded, and we see comedic, older classics that still align with user interest.

5.7 Limitations

We rely solely on item content (no user-user collaborative data). The Naïve Bayes "like=IMDb7.5" is synthetic, potentially missing real user preferences. Some lift-laden association rules might not reflect actual user taste. But this system does illustrate how combining multiple content-based signals yields more robust coverage than a single approach.

6 Conclusion and Future Work

We presented a multi-method content-based recommender system for Netflix data, integrating:

- Cosine-based user profile matching,
- kNN local retrieval,
- Naïve Bayes synthetic classification,
- K-Means cluster preference,
- Apriori association rules.

Our pipeline overcame many typical single-method limitations, fostering more diverse, user-aligned recommendations. While successful in bridging item-based signals, we have not yet incorporated collaborative filtering or real user rating data. We also used synthetic labels for the Naïve Bayes step.

Future directions include collecting actual user ratings, exploring dynamic weighting for association rules, or a hybrid with partial user-user data. Overall, this project demonstrates that a synergy of multiple content-based techniques can handle large catalogs like Netflix and yield personalized results with minimal cold start overhead.

References

[1] D. Goldberg, D. Nichols, B. M. Oki, & D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[2] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, & J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186, 1994.

[3] A. I. Schein, A. Popescul, L. H. Ungar, & D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 253–260, 2002.

[4] R. J. Mooney & L. Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pp. 195–204, 2000.

[5] M. Pazzani & D. Billsus. Content-based recommendation systems. In *The Adaptive Web*, pages 325–341. Springer, 2007.

[6] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[7] G. Adomavicius & A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[8] C. A. Gómez-Uribe & N. Hunt. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(4):1–19, 2015.

[9] E. Alpaydın. *Introduction to Machine Learning* (4th ed.). MIT Press, 2020.

[10] A. Das, M. Datar, A. Garg, & S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, pp. 271–280, 2007.

[11] R. Agrawal & R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases (VLDB)*, pages 487–499, 1994.

[12] B. Mobasher, H. Dai, T. Luo, & M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proceedings of the 3rd international workshop on Web information and data management*, pp. 9–15, 2001.

[13] Chirag9073. Netflix Data Analysis. In *Kaggle.com*, Kaggle, June 3, 2020. Available at: `https://www.kaggle.com/code/chirag9073/netflix-data-analysis/notebook`.

[14] OMDb API. The Open Movie Database. In *Omdbapi.com*, 2000. Available at: `https://www.omdbapi.com/`.

7 Appendix

```

# Import required libraries and mount Google Drive

import os, time, math, io, requests
from concurrent.futures import ThreadPoolExecutor

# Data handling and numerical computation
import numpy as np
import pandas as pd
import scipy.stats as ss

# Machine learning and preprocessing
from sklearn.preprocessing import MultiLabelBinarizer, MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
from sklearn.naive_bayes import GaussianNB
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import silhouette_score
from mlxtend.frequent_patterns import apriori, association_rules

# Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Google API libraries
!pip install --upgrade gspread google-auth google-auth-oauthlib google-auth-httplib2 google-api-python-client
import gspread
from google.oauth2.service_account import Credentials

# Progress bar utility
from tqdm import tqdm

# Mount Google Drive (for Colab)
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

```

```python
# working directory project folder in Google Drive
pathname = '/content/drive/MyDrive/DataMiningTeam/'
os.chdir(pathname)


# Load the raw Netflix dataset and perform data cleaning

filename = "netflix_database.xlsx"
dt = pd.read_excel(filename)

print("Initial dataset preview:")
display(dt.head())

# Fill missing values for director, cast, and country
dt[['director', 'cast', 'country']] = dt[['director', 'cast', 'country']].fillna("Unknown")

# Fill missing ratings with the most common rating (mode)
mode_rating = dt['rating'].mode()[0]
dt['rating'] = dt['rating'].fillna(mode_rating)

# Rename 'listed_in' to 'genre'
dt.rename(columns={'listed_in': 'genre'}, inplace=True)

# Drop rows with missing 'date_added' and extract day, month, year
dt = dt.dropna(subset=['date_added']).copy()
dt['day_added'] = dt['date_added'].dt.day.astype(int)
dt['month_added'] = dt['date_added'].dt.month.astype(int)
dt['year_added'] = dt['date_added'].dt.year.astype(int)

# Fill missing duration separately for TV Shows and Movies using mode
tv_mode = dt.loc[dt['type'] == 'TV Show', 'duration'].mode()[0]
mv_mode = dt.loc[dt['type'] == 'Movie', 'duration'].mode()[0]
dt.loc[dt['type'] == 'TV Show', 'duration'] = dt.loc[dt['type'] == 'TV Show', 'duration'].fillna(tv_mode)
dt.loc[dt['type'] == 'Movie', 'duration'] = dt.loc[dt['type'] == 'Movie', 'duration'].fillna(mv_mode)

# Extract numerical duration:
```

```python
# For movies: extract minutes; for TV shows: extract number of seasons.
dt['duration_minutes'] = dt['duration'].apply(lambda x: float(x.split()[0]) if 'min' in str(x) else np.nan)
dt['duration_seasons'] = dt['duration'].apply(lambda x: float(x.split()[0]) if 'Season' in str(x) else np.nan)
dt.drop(columns=['duration'], inplace=True)

print("Missing values after cleaning:")
print(dt.isnull().sum())
print("\nDataset preview after cleaning:")
display(dt.head())


    #  Fetch missing IMDb ratings using the OMDb API with multithreading

    OMDB_API_KEY = "17bd2674"
    OMDB_URL = "http://www.omdbapi.com/"

    def get_imdb_rating(title):
        params = {"t": title, "apikey": OMDB_API_KEY}
        try:
            response = requests.get(OMDB_URL, params=params, timeout=5)
            data = response.json()
            if response.status_code == 200 and data.get("Response") == "True":
                return data.get("imdbRating", "N/A")
        except Exception as e:
            print(f"Error fetching rating for {title}: {e}")
        return None

    # Ensure 'imdb_rating' exists
    if "imdb_rating" not in dt.columns:
        dt["imdb_rating"] = None

    titles_to_fetch = dt[(dt["imdb_rating"].isna()) | (dt["imdb_rating"] == "N/A")]["title"].tolist()

    with ThreadPoolExecutor(max_workers=20) as executor:
        results = list(tqdm(executor.map(get_imdb_rating, titles_to_fetch), total=len(titles_to_fetch)))

    for title, imdb in zip(titles_to_fetch, results):
```

```python
      dt.loc[dt["title"] == title, "imdb_rating"] = imdb

  # Convert to numeric and fill any missing values with the median
  dt["imdb_rating"] = pd.to_numeric(dt["imdb_rating"], errors='coerce')
  median_rating = dt["imdb_rating"].median()
  dt["imdb_rating"].fillna(median_rating, inplace=True)

  file_path = "/content/drive/My Drive/DataMiningTeam/netflix_database_cleaned.xlsx"
  dt.to_excel(file_path, index=False)
  print(f"IMDb Ratings Fetching Complete! Data saved to: {file_path}")


# Cell 4: Visualize dataset characteristics

def plot_visualizations(dt):
    plt.figure(figsize=(6, 4))
    sns.countplot(x='type', data=dt)
    plt.title('Count of Movies vs TV Shows')
    plt.show()

    plt.figure(figsize=(10, 5))
    sns.histplot(dt["imdb_rating"].dropna(), bins=20, kde=True)
    plt.title("Distribution of IMDb Ratings")
    plt.xlabel("IMDb Rating")
    plt.ylabel("Count")
    plt.show()

    plt.figure(figsize=(10, 5))
    sns.countplot(x='rating', data=dt, order=dt['rating'].value_counts().index)
    plt.title('Content Rating Distribution')
    plt.xticks(rotation=45)
    plt.show()

    plt.figure(figsize=(8, 5))
    sns.histplot(x='release_year', data=dt, bins=30)
    plt.title('Release Year Distribution')
    plt.show()

    plt.figure(figsize=(8, 5))
```

```python
    sns.histplot(x='duration_minutes', data=dt[dt['type'] == 'Movie'])
    plt.title('Movie Duration (Minutes)')
    plt.show()

    plt.figure(figsize=(8, 5))
    sns.histplot(x='duration_seasons', data=dt[dt['type'] == 'TV Show'], bins=range(1,15))
    plt.title('TV Show Seasons Distribution')
    plt.show()

    dt["genre_list"] = dt["genre"].fillna("").apply(lambda x: x.split(", "))
    all_genres = [genre for sublist in dt["genre_list"] for genre in sublist if genre]
    top_genres = pd.Series(all_genres).value_counts().head(10)
    plt.figure(figsize=(12, 6))
    sns.barplot(x=top_genres.index, y=top_genres.values, palette="coolwarm")
    plt.xticks(rotation=45, ha="right")
    plt.title("Top 10 Genres")
    plt.xlabel("Genre")
    plt.ylabel("Count")
    plt.show()

plot_visualizations(dt)


# engineered features

# Created a genre list from the 'genre' column
dt['genre_list'] = dt['genre'].fillna("").apply(lambda x: x.split(", "))

# One-hot encode genres
mlb = MultiLabelBinarizer()
genre_encoded = mlb.fit_transform(dt['genre_list'])
genre_df = pd.DataFrame(genre_encoded, columns=mlb.classes_, index=dt.index)

# Scale release_year
scaler = MinMaxScaler()
dt['release_year_scaled'] = scaler.fit_transform(dt[['release_year']].fillna(dt['release_year'].min()
```

```python

# final feature-engineered DataFrame
dt_final = pd.concat([
    dt[['show_id', 'title', 'type', 'release_year_scaled', 'duration_minutes', 'duration_seasons']],
    genre_df
], axis=1).copy()

# Add IMDb rating
dt_final["imdb_rating"] = dt["imdb_rating"]

# Save engineered dataset
file_path = "/content/drive/My Drive/DataMiningTeam/netflix_feature_engineered.xlsx"
dt_final.to_excel(file_path, index=False)
print(f"Feature Engineered Dataset Saved: {file_path}")

print("Shape of dt_final:", dt_final.shape)
display(dt_final.head())


# Construct the numerical feature matrix for similarity calculations
# and normalize IMDb ratings in one cell.

# First, update dt_final with normalized IMDb ratings.
dt_final["imdb_rating"] = pd.to_numeric(dt_final["imdb_rating"], errors='coerce')
median_imdb = dt_final["imdb_rating"].median()
dt_final["imdb_rating"] = dt_final["imdb_rating"].fillna(median_imdb)

scaler = MinMaxScaler()
dt_final["normalized_imdb"] = scaler.fit_transform(dt_final[["imdb_rating"]])

# Now, drop non-numeric columns and fill any remaining missing values.
feature_columns = dt_final.drop(columns=['show_id', 'title', 'type'], errors='ignore').fillna(0)
feature_matrix = feature_columns.to_numpy()

print("Feature Matrix Created!")
print("Shape:", feature_matrix.shape)
print("Sample rows:\n", feature_matrix[:1])
```

```python


#  Load user preferences via Google Sheets

# Load user preferences via Google Sheets and parse them into usable Python objects.

SERVICE_ACCOUNT_FILE = "/content/drive/My Drive/DataMiningTeam/service_account.json"

creds = Credentials.from_service_account_file(
    SERVICE_ACCOUNT_FILE,
    scopes=["https://spreadsheets.google.com/feeds", "https://www.googleapis.com/auth/drive"]
)
gc = gspread.authorize(creds)
print("Google Sheets Authentication Successful!")

SPREADSHEET_URL = "https://docs.google.com/spreadsheets/d/11COs6jCFzElSMi78x39_VZtGqhVOpi1p
spreadsheet = gc.open_by_url(SPREADSHEET_URL)
worksheet = spreadsheet.sheet1
data = worksheet.get_all_values()
latest_response = pd.DataFrame([data[-1]], columns=data[0])
print("User History Loaded!")

# Parse user responses
latest_response.columns = latest_response.columns.str.strip()

liked_movies = latest_response["Liked Movies/TV Shows"].values[0] if "Liked Movies/TV Shows" in latest_response else ""
disliked_movies = latest_response["Disliked Movies/TV Shows"].values[0] if "Disliked Movies/TV Shows" in latest_response else ""
preferred_genres = latest_response["Preferred Genres"].values[0] if "Preferred Genres" in latest_response else ""
imdb_min_rating = latest_response["IMDb Rating Preference"].values[0] if "IMDb Rating Preference" in latest_response else "No preference"
content_preference = latest_response["Do you prefer Movies or TV Shows?"].values[0] if "Do you prefer Movies or TV Shows?" in latest_response else "No preference"
```

```python

liked_movies = liked_movies.split(", ") if liked_movies else []
disliked_movies = disliked_movies.split(", ") if disliked_movies else []
preferred_genres = preferred_genres.split(", ") if preferred_genres else []

if imdb_min_rating not in ["No preference", ""]:
    imdb_min_rating = float(imdb_min_rating.rstrip("+"))
else:
    imdb_min_rating = 0

print(f"Liked Movies: {liked_movies}")
print(f"Disliked Movies: {disliked_movies}")
print(f"Preferred Genres: {preferred_genres}")
print(f"IMDb Minimum Rating: {imdb_min_rating}")
print(f"Content Preference: {content_preference}")




# Compute a weighted user profile based on liked movies and calculate cosine similarity.

# Find indices of movies the user likes.
liked_indices = dt_final[dt_final['title'].isin(liked_movies)].index.to_numpy()

# If no liked movies but preferred genres exist, use a fallback: select top 3 movies in those genres.
if len(liked_indices) == 0 and preferred_genres:
    genre_filtered = dt_final[dt_final[preferred_genres].sum(axis=1) > 0]
    liked_movies = genre_filtered['title'].head(3).tolist()
    liked_indices = dt_final[dt_final['title'].isin(liked_movies)].index.to_numpy()

# Compute user profile: weighted average of feature vectors for liked movies.
if len(liked_indices) > 0:
    weights = dt_final.iloc[liked_indices]['imdb_rating'] / dt_final.iloc[liked_indices]['imdb_rating'].sum()
```

```python
    user_profile = np.average(feature_matrix[liked_indices], axis=0, weights=weights).reshape(1, -1)
else:
    # Fallback to using the average feature vector across all items if no liked movies.
    user_profile = np.mean(feature_matrix, axis=0).reshape(1, -1)

# Compute cosine similarity between the user profile and all items.
similarities = cosine_similarity(user_profile, feature_matrix)[0]
dt_final['similarity_score'] = similarities

# Compute adjusted similarity using a weighted sum (alpha for similarity, 1-alpha for normalized IMDb rating).
alpha = 0.7
dt_final["adjusted_similarity"] = alpha * dt_final["similarity_score"] + (1 - alpha) * dt_final["normalized_imdb"]

print("User Profile & Cosine Similarity Computed!")




# Filter recommendations based on user preferences and display final content-based recommendations.

# Exclude items the user already liked or disliked.
recommended = dt_final[~dt_final['title'].isin(liked_movies)]
recommended = recommended[~recommended['title'].isin(disliked_movies)]
# Apply the IMDb rating filter.
recommended = recommended[recommended["imdb_rating"] >= imdb_min_rating]

# Fallback: If fewer than 10 recommendations are found, expand search using preferred genres.
if len(recommended) < 10:
    print("Less than 10 strong matches found. Expanding search with fallback recommendations...")
    if preferred_genres:
```

```python
        fallback = dt_final[dt_final[preferred_genres].sum(axis=1) > 0]
        fallback = fallback.sort_values(by=["imdb_rating", "similarity_score"], ascending=[False, False])
        recommended = pd.concat([recommended, fallback]).drop_duplicates().head(10)
    else:
        recommended = recommended.head(10)

# If the user prefers a specific type, split recommendations accordingly.
recommended_movies = recommended[recommended["type"] == "Movie"]
recommended_shows = recommended[recommended["type"] == "TV Show"]

if content_preference == "Both":
    top_movies = recommended_movies.head(5)
    top_shows = recommended_shows.head(5)
    final_recommendations = pd.concat([top_movies, top_shows]).sort_values(by="adjusted_similarity", ascending=False)

elif content_preference == "Movies Only":
    final_recommendations = recommended_movies.head(10)

elif content_preference == "TV Shows Only":
    final_recommendations = recommended_shows.head(10)

else:  # "No preference" or any other input
    final_recommendations = recommended.head(10)

print("Top Recommendations:Based on Cosine Similarity")
display(final_recommendations[['title', 'type', 'imdb_rating', 'adjusted_similarity']])



# alternative recommendations using kNN.

# Initialize and fit the kNN model on the full feature matrix
knn = NearestNeighbors(n_neighbors=20, metric='cosine')
knn.fit(feature_matrix)
```

```python

# Find the indices of the 20 nearest neighbors to the user profile
_, indices = knn.kneighbors(user_profile)
knn_recommendations = dt_final.iloc[indices[0]].copy()

# Compute cosine similarity for these kNN recommendations using their numeric features
# Note: Reconstruct the feature matrix for just these neighbors
neighbors_features = feature_columns.iloc[indices[0]].to_numpy()
knn_similarities = cosine_similarity(user_profile, neighbors_features)[0]
knn_recommendations['similarity_score'] = knn_similarities

# Compute adjusted similarity as a weighted sum of cosine similarity and normalized IMDb rating
alpha = 0.7  # weight for similarity; 30% weight for normalized IMDb rating
knn_recommendations["adjusted_similarity"] = alpha * knn_recommendations["similarity_score"] + (1 - alpha) * knn_recommendations["normalized_imdb"]

# Exclude items the user disliked
knn_recommendations = knn_recommendations[~knn_recommendations['title'].isin(disliked_movies)]

# Apply the minimum IMDb rating filter from user preference
knn_recommendations = knn_recommendations[knn_recommendations["imdb_rating"] >= imdb_min_rating]

# Filter recommendations based on content preference:
if content_preference == "Movies Only":
    knn_recommendations = knn_recommendations[knn_recommendations["type"] == "Movie"]
elif content_preference == "TV Shows Only":
    knn_recommendations = knn_recommendations[knn_recommendations["type"] == "TV Show"]


```

```python
# Sort by adjusted similarity in descending order for final ranking
knn_recommendations = knn_recommendations.sort_values(by="adjusted_similarity", ascending=False)

print("Top kNN-Based Recommendations:")
display(knn_recommendations[['title', 'type', 'imdb_rating', 'adjusted_similarity']].head(10))


# Association Rule Mining on Genre Data using Apriori

# Convert genre_df to boolean type.
genre_df = genre_df.astype(bool)

frequent_itemsets = apriori(genre_df, min_support=0.05, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)

print("Association Rules based on genres (sample):")
display(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())


# Naïve Bayes classifier

# 1. Define 'like' using your synthetic rule
dt_final['like'] = (dt_final['imdb_rating'] >= 7.5).astype(int)

# 2. Combine user-labeled items
user_labeled_titles = set(liked_movies + disliked_movies)

# 3. Exclude those items from dt_final before building the train set
excluded_indices = dt_final[dt_final['title'].isin(user_labeled_titles)].index
train_dt = dt_final.drop(index=excluded_indices)

# 4. List columns we DO NOT want for model training
exclude_cols = [
    'show_id', 'title', 'type',             # not features
```

```
    'like', 'cluster', 'similarity_score', # or any added columns
    'adjusted_similarity', 'ensemble_score',
    'association_bonus', 'nb_score', 'cluster_score'
]

# 5. Prepare the training feature matrix
train_feature_cols = train_dt.drop(columns=exclude_cols, errors='ignore').fillna(0)
train_X = train_feature_cols.to_numpy()
train_y = train_dt['like'].values

# 6. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    train_X,
    train_y,
    test_size=0.2,
    random_state=42
)

clf = GaussianNB()
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)

# Evaluate the classifier
acc = accuracy_score(y_test, predictions)
prec = precision_score(y_test, predictions)
rec = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print("Naïve Bayes classifier trained (excluding user-labeled items).")
print(f"Accuracy: {acc:.2f}, Precision: {prec:.2f}, Recall: {rec:.2f}, F1 Score: {f1:.2f}")


# Re-applying  the model to the FULL dt_final so we can get nb_score for all items
full_feature_cols = dt_final.drop(columns=exclude_cols, errors='ignore').fillna(0)
full_X = full_feature_cols.to_numpy()

nb_proba_all = clf.predict_proba(full_X)[:, 1]
dt_final['nb_score'] = nb_proba_all
```

```
print("Example nb_score:")
display(dt_final[['title', 'nb_score']].head(10))


# Elbow

sse = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(feature_matrix)
    sse.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), sse, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Sum of Squared Errors (SSE)")
plt.show()


# Apply K-Means clustering to group items



print("Trying different k values for K-Means:")
for k in range(3, 10):
    kmeans_temp = KMeans(n_clusters=k, random_state=42)
    clusters_temp = kmeans_temp.fit_predict(feature_matrix)
    sil_score = silhouette_score(feature_matrix, clusters_temp)
    print(f"k={k}: Cluster Distribution: {np.bincount(clusters_temp)} | Silhouette Score: {sil_score:.3f}")

# Set optimal k
k_optimal = 4
kmeans = KMeans(n_clusters=k_optimal, random_state=42)
clusters = kmeans.fit_predict(feature_matrix)
dt_final['cluster'] = clusters

print("K-Means clustering complete. Cluster distribution:")
print(dt_final['cluster'].value_counts())
```

```python
#  Silhouette Bar Chart

# We'll reuse the range(3, 10) and store silhouette scores in a list for plotting
k_values = list(range(3, 10))
silhouette_scores = []

print("\nSilhouette Bar Chart for K-Means:")
for k in k_values:
    kmeans_temp = KMeans(n_clusters=k, random_state=42)
    clusters_temp = kmeans_temp.fit_predict(feature_matrix)
    sil_score = silhouette_score(feature_matrix, clusters_temp)
    silhouette_scores.append(sil_score)

plt.figure(figsize=(7, 5))
plt.bar(k_values, silhouette_scores, color='skyblue')
plt.title("Silhouette Scores for Different k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Silhouette Score")
for i, score in enumerate(silhouette_scores):
    plt.text(k_values[i], score, f"{score:.3f}", ha='center', va='bottom', fontsize=9)
plt.ylim(0, max(silhouette_scores) + 0.05)
plt.show()


#  Cluster Profiling

# Suppose you want to see the average IMDb rating, release_year_scaled,
# and the top 5 genres in each cluster.

genre_cols = genre_df.columns  # from your one-hot-encoded df
unique_clusters = sorted(dt_final['cluster'].unique())

print("\n=== Cluster Profiling ===")
for c in unique_clusters:
    cluster_data = dt_final[dt_final['cluster'] == c]
    print(f"\n--- Cluster {c} ---")
    print(f"Number of items: {len(cluster_data)}")
    print(f"Average IMDb rating: {cluster_data['imdb_rating'].mean():.2f}")
```

```python
    print(f"Average release_year_scaled: {cluster_data['release_year_scaled'].mean():.2f}")

    # Summation over genre columns to see which genres are most common
    cluster_genre_sum = cluster_data[genre_cols].sum().sort_values(ascending=False).head(5)
    print("Top 5 genres:")
    display(cluster_genre_sum)


# Compute an ensemble recommendation score combining multiple methods

# Ensure 'normalized_imdb' exists; recompute if missing
if "normalized_imdb" not in dt_final.columns:
    print(" 'normalized_imdb' column missing. Recomputing...")
    dt_final["imdb_rating"] = pd.to_numeric(dt_final["imdb_rating"], errors='coerce')
    median_imdb = dt_final["imdb_rating"].median()
    dt_final["imdb_rating"] = dt_final["imdb_rating"].fillna(median_imdb)
    scaler = MinMaxScaler()
    dt_final["normalized_imdb"] = scaler.fit_transform(dt_final[["imdb_rating"]])

# Ensure 'adjusted_similarity' exists; recompute if missing
if "adjusted_similarity" not in dt_final.columns:
    print(" 'adjusted_similarity' missing. Recomputing...")
    similarities = cosine_similarity(user_profile, feature_matrix)[0]
    dt_final['similarity_score'] = similarities
    alpha = 0.7  # weight for similarity
    dt_final["adjusted_similarity"] = alpha * dt_final["similarity_score"] + (1 - alpha) * dt_final["normalized_imdb"]
    dt_final["adjusted_similarity"] = dt_final["adjusted_similarity"].fillna(0)


# Get Naïve Bayes classifier probability for the positive (like) class
nb_proba = clf.predict_proba(feature_matrix)[:, 1]
```

```python
dt_final['nb_score'] = nb_proba

# Compute fraction of liked movies in each cluster as a measure of cluster preference
liked_clusters = dt_final.loc[dt_final['title'].isin(liked_movies), 'cluster']
if len(liked_clusters) > 0:
    cluster_pref = liked_clusters.value_counts(normalize=True).to_dict()
else:
    cluster_pref = {}
dt_final['cluster_score'] = dt_final['cluster'].apply(lambda x: cluster_pref.get(x, 0))

# Adjust ensemble score weights: Here, we use 0.4 for adjusted similarity, 0.4 for NB score, and 0.2 for cluster score.
# (tweak these weights to get a better balance of quality vs. diversity.)
total = 0.4 + 0.4 + 0.5

w_adj = 0.5/total
w_nb_score = 0.4/total
w_cluster_score = 0.4/total



dt_final['ensemble_score'] = (w_adj * dt_final['adjusted_similarity'] +
                              w_nb_score * dt_final['nb_score'] +
                              w_cluster_score * dt_final['cluster_score'])

#  association rule
associated_titles = []

dt_final['association_bonus'] = dt_final['title'].apply(lambda t: 0.05 if t in associated_titles else 0)
dt_final['ensemble_score'] += dt_final['association_bonus']

```

```python
# Filter out liked and disliked items from ensemble recommendations
ensemble_recs = dt_final[~dt_final['title'].isin(liked_movies)]
ensemble_recs = ensemble_recs[~ensemble_recs['title'].isin(disliked_movies)]
ensemble_recs = ensemble_recs.sort_values(by='ensemble_score', ascending=False)

print(" Final Recommendations:")
display(ensemble_recs[['title', 'type', 'imdb_rating', 'ensemble_score']].head(10))
```