geomwangi007 /
Movie-Recommendation-System-Group_12-Project

Code Issues Pull requests Actions Projects Wiki Security

Movie-Recommendation-System-Group_12-Project / Main.ipynb

DrTechBro finalized notebook e82a9a4 · 2 hours ago

5731 lines (5731 loc) · 1.6 MB

Preview Code Blame Raw

In [251…

```
import time
start_time = time.time()
```

"Intelligent Movie Recommendations with Neural Networks and NLP"

Problem Statement

In today's digital age, the sheer volume of available movies has grown exponentially, leading to an overwhelming choice paralysis for viewers seeking content that aligns with their personal tastes. Traditional recommendation systems often fall short by providing generic suggestions based on popularity or simplistic user behaviors, failing to capture the nuanced preferences of individual users. This lack of personalization results in a suboptimal viewing experience, where users spend more time searching for movies than enjoying them.

The FlickPick Engine aims to solve this problem by developing an intelligent movie recommendation system that delivers highly personalized and relevant suggestions to users. By leveraging advanced machine learning techniques—specifically neural networks for collaborative filtering and Natural Language Processing (NLP) for content analysis—the system can understand and interpret both user preferences and movie attributes on a deeper level.

By addressing the challenges of information overload and impersonal recommendations, the FlickPick Engine enhances the movie discovery process. It empowers users to effortlessly find films that resonate with their unique tastes, thereby improving user satisfaction and engagement with the platform.

Data Understanding

To develop the FlickPick Engine, we utilized the MovieLens 20M Dataset, a widely recognized dataset in the recommendation systems domain. This dataset provides a rich source of user ratings, movie metadata, and user-generated tags, enabling the creation of a robust and personalized movie recommendation system.

Data Sources

1. Ratings Data (ratings.csv): Contains 25 million ratings ranging from 0.5 to 5.0, provided by 162,541 users on 62,423 movies.
2. Movies Data (movies.csv): Includes movie IDs, titles, and genres for all movies rated in the dataset.
3. Tags Data (tags.csv): Consists of 1.1 million user-generated tags applied to movies, offering additional contextual information.
4. Links Data (links.csv): Provides identifiers that link MovieLens movie IDs with IDs

from other databases like IMDb and TMDb.

In [252…

```
# Importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from scipy.stats import ttest_rel
from collections import defaultdict
from sklearn.model_selection import train_test_split
from surprise import Prediction,accuracy,Dataset, Reader, SVD,SVDpp,KNNBasi
from sklearn.metrics.pairwise import cosine_similarity,linear_kernel
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.metrics import mean_squared_error,recall_score
from surprise.model_selection import GridSearchCV,train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
import string
import pandas as pd
import numpy as np
import gensim.downloader as api
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics.pairwise import cosine_similarity
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Embedding, Flatten, Multi
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout
from keras.optimizers import Adam
from keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
import warnings
warnings.filterwarnings('ignore')
```

Data Loading

In [253…

```
# Load the datasets
links_df = pd.read_csv('Data/ml-latest-small/links.csv')
movies_df = pd.read_csv('Data/ml-latest-small/movies.csv')
ratings_df = pd.read_csv('Data/ml-latest-small/ratings.csv')
tags_df = pd.read_csv('Data/ml-latest-small/tags.csv')
```

Inspecting the structure of the datasets

In [254…

```
links_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   imdbId   9742 non-null   int64
```

```
 2   tmdbId   9734 non-null   float64
dtypes: float64(1), int64(2)
memory usage: 228.5 KB
```

In [255…

```
movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

In [256…

```
ratings_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

In [257…

```
tags_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3683 entries, 0 to 3682
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   userId     3683 non-null   int64 
 1   movieId    3683 non-null   int64 
 2   tag        3683 non-null   object
 3   timestamp  3683 non-null   int64 
dtypes: int64(3), object(1)
memory usage: 115.2+ KB
```

Checking for missing values and duplicates

In [258…

```
# Check for missing values and duplicates

links_info = {
    "missing_values": links_df.isnull().sum(),
    "duplicates": links_df.duplicated().sum()
}

movies_info = {
    "missing_values": movies_df.isnull().sum(),
    "duplicates": movies_df.duplicated().sum()
}

ratings_info = {
    "missing_values": ratings_df.isnull().sum(),
```

```
    "duplicates": ratings_df.duplicated().sum(),
    "invalid_ratings": ratings_df[~ratings_df['rating'].between(0.5, 5)].sh
}

tags_info = {
    "missing_values": tags_df.isnull().sum(),
    "duplicates": tags_df.duplicated().sum()
}


links_info, movies_info, ratings_info, tags_info
```

Out[258…

```
({'missing_values': movieId    0
  imdbId     0
  tmdbId     8
  dtype: int64,
  'duplicates': 0},
 {'missing_values': movieId    0
  title      0
  genres     0
  dtype: int64,
  'duplicates': 0},
 {'missing_values': userId       0
  movieId      0
  rating       0
  timestamp    0
  dtype: int64,
  'duplicates': 0,
  'invalid_ratings': 0},
 {'missing_values': userId       0
  movieId      0
  tag          0
  timestamp    0
  dtype: int64,
  'duplicates': 0})
```

1. links.csv:

- Missing Values: There are 8 missing values in the tmdbId column.
- Duplicates: There are no duplicate rows.

2. movies.csv:

- Missing Values: No missing values were found.
- Duplicates: There are no duplicate rows.

3. ratings.csv:

- Missing Values: No missing values were found.
- Duplicates: There are no duplicate rows.
- Invalid Ratings: All ratings are valid, meaning they fall within the expected range (0.5 to 5).

4. tags.csv:

- Missing Values: No missing values were found.
- Duplicates: There are no duplicate rows.

In [259…

```
# Droping rows with missing tmdbId
links_df = links_df.dropna(subset=['tmdbId'])
```

In [260…

```
# Checking the current data types of each column in the datasets
links_dtypes = links_df.dtypes
movies_dtypes = movies_df.dtypes
ratings_dtypes = ratings_df.dtypes
tags_dtypes = tags_df.dtypes

# Reviewing data types
links_dtypes, movies_dtypes, ratings_dtypes, tags_dtypes
```

Out[260…

```
(movieId      int64
 imdbId       int64
 tmdbId     float64
 dtype: object,
 movieId     int64
 title      object
 genres     object
 dtype: object,
 userId         int64
 movieId        int64
 rating       float64
 timestamp      int64
 dtype: object,
 userId        int64
 movieId       int64
 tag          object
 timestamp     int64
 dtype: object)
```

1. links.csv:

- movieId: int64 (appropriate)
- imdbId: int64 (appropriate)
- tmdbId: float64 (should be int64, as tmdbId is an identifier and doesn't need decimal precision)

2. movies.csv:

- movieId: int64 (appropriate)
- title: object (appropriate for movie titles)
- genres: object (appropriate for genres, which are stored as strings)

3. ratings.csv:

- userId: int64 (appropriate)
- movieId: int64 (appropriate)
- rating: float64 (appropriate since ratings have decimal points)
- timestamp: int64 (appropriate for Unix timestamps)

4. tags.csv:

- userId: int64 (appropriate)
- movieId: int64 (appropriate)
- tag: object (appropriate for textual tags)

- timestamp: int64 (appropriate for Unix timestamps)

Adjustment:

In links.csv, the tmdbId column should be converted to int64 because it represents a unique identifier

In [261…

```
# Converting tmdbId to int64
links_df['tmdbId'] = links_df['tmdbId'].astype('int64')
```

In [262…

```
print(movies_df['genres'].dtype)
```

```
object
```

Data Visualization

First Lets look at the distribution of movie ratings

Purpose:

Understand the overall distribution of movie ratings.

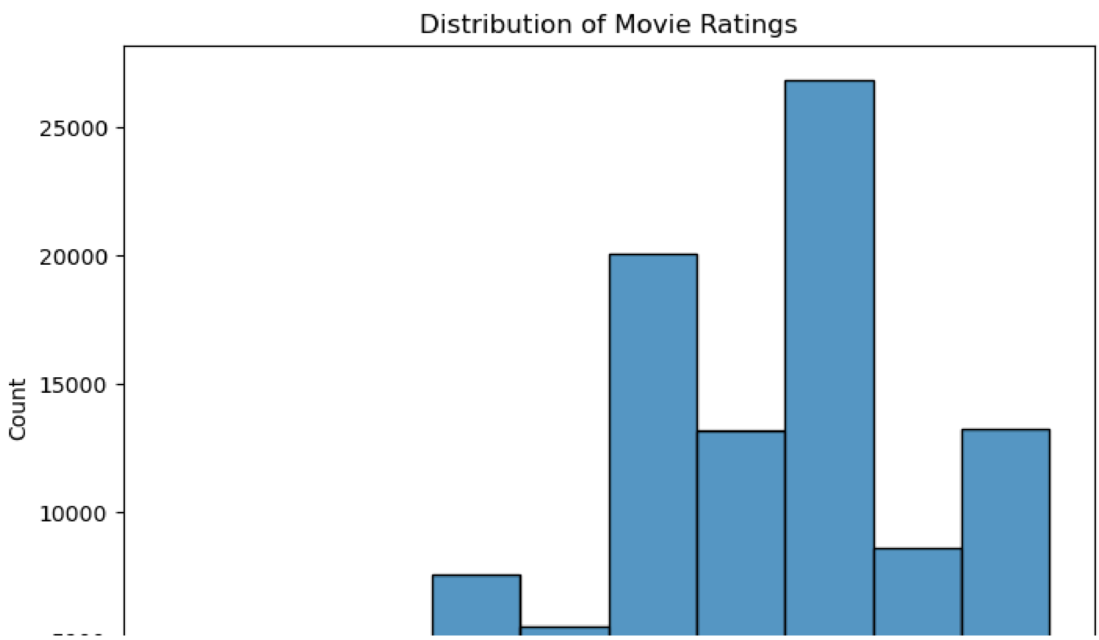
Identify any biases (e.g., users tending to give higher ratings).

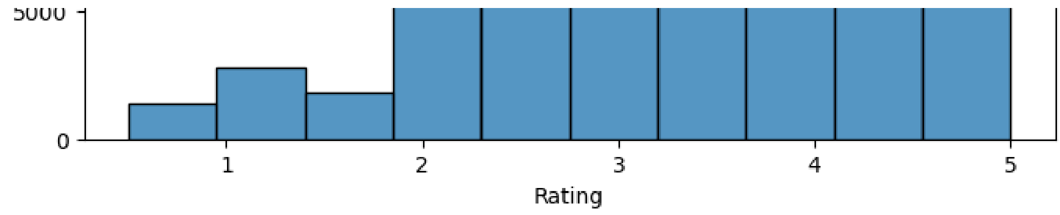
Identify the most common ratings (e.g., peaks at whole numbers like 4.0 or 5.0).

Assess whether the distribution is skewed toward higher or lower ratings.

In [263…

```
# Ploting histogram
plt.figure(figsize=(8, 6))
sns.histplot(ratings_df['rating'], bins=10)
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.savefig('Visualizations/Distribution of Movie Ratings')
plt.show()
```





- Skew Towards Higher Ratings: The distribution is right-skewed, with most ratings clustering around 3, 4, and 5. Ratings of 4 are the most frequent, indicating that users generally rate movies quite positively.
- Few Low Ratings: There are relatively few ratings of 1 and 2, which suggests that users may be less likely to give movies extremely low scores, or that most of the movies in the dataset are well-regarded.
- Peak at Rating 4: The highest count of ratings is around 4, suggesting that many users find the movies to be above average but not necessarily perfect.

This type of distribution is common in user-driven ratings, where users tend to be more inclined to rate items positively than negatively.

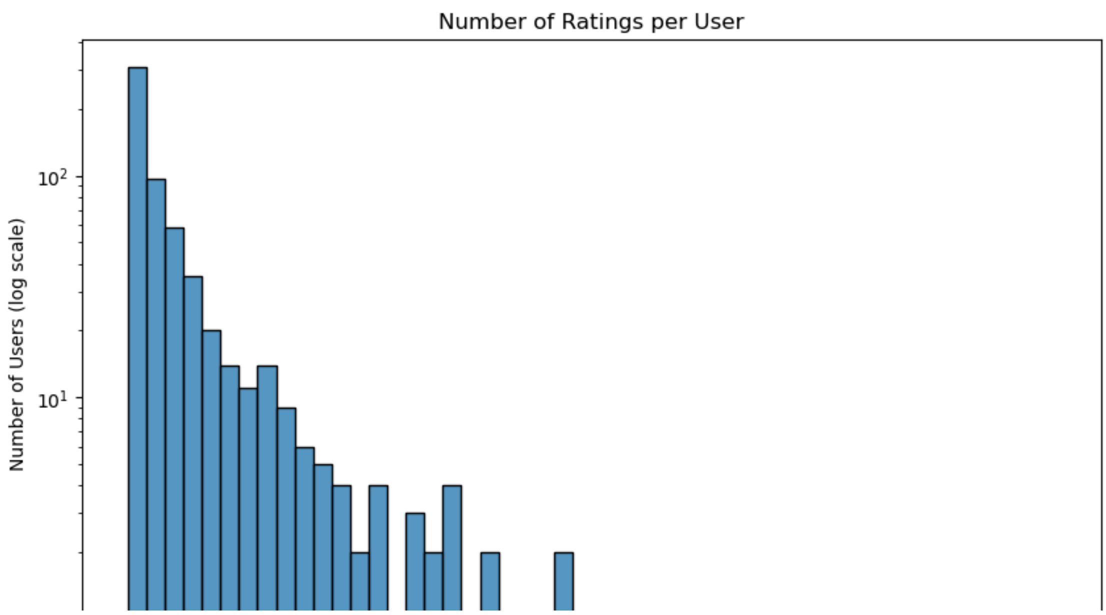
Now let's have a look at the Distribution of Ratings per User

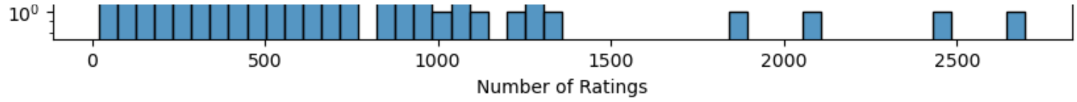
Examine how many ratings each user has provided.

Identify active users versus casual users.

In [264…

```
# Calculating number of ratings per user
user_rating_counts = ratings_df.groupby('userId')['rating'].count()

# Ploting histogram
plt.figure(figsize=(10, 6))
sns.histplot(user_rating_counts, bins=50, log_scale=(False, True))
plt.title('Number of Ratings per User')
plt.xlabel('Number of Ratings')
plt.ylabel('Number of Users (log scale)')
plt.savefig('Visualizations/Number of Ratings per User')
plt.show()
```





The distribution is often long-tailed, with a few users providing many ratings and many users providing few ratings.

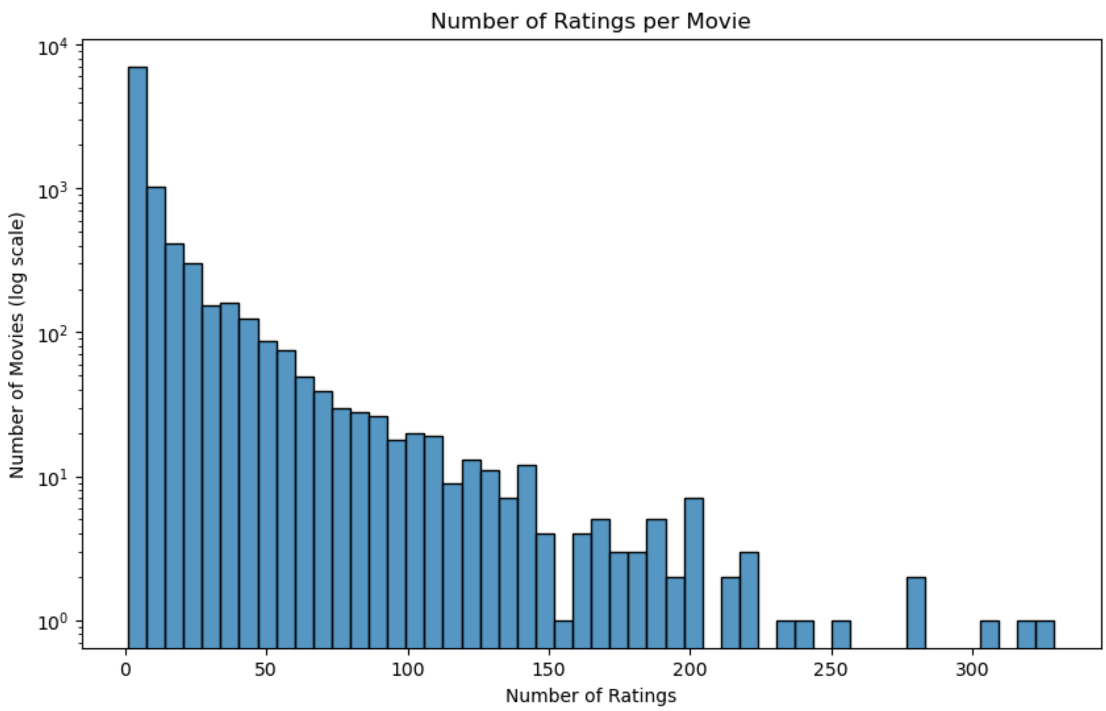
- Power-Law Distribution: The distribution exhibits a power-law trend, where most users rate very few movies, and only a few users provide a large number of ratings. This type of behavior is typical in user-generated content datasets, often referred to as the "long tail."
- Majority Have Rated Few Movies: A significant number of users have rated fewer than 100 movies, which suggests that casual users dominate the dataset.
- Heavy Users: There are a small number of users who have rated over 500 movies, with some even rating over 1000. These "heavy users" contribute disproportionately to the number of total ratings in the dataset.

Next we look at the Distribution of Ratings per Movie

To determine how many ratings each movie has received.and identify popular movies versus obscure ones.

In [265…

```
# Calculating number of ratings per movie
movie_rating_counts = ratings_df.groupby('movieId')['rating'].count()

# Ploting histogram
plt.figure(figsize=(10, 6))
sns.histplot(movie_rating_counts, bins=50, log_scale=(False, True))
plt.title('Number of Ratings per Movie')
plt.xlabel('Number of Ratings')
plt.ylabel('Number of Movies (log scale)')
plt.savefig('Visualizations/Number of Ratings per Movie')
plt.show()
```



Similar to users, movies often follow a long-tailed distribution.

Important for understanding the sparsity of the dataset.

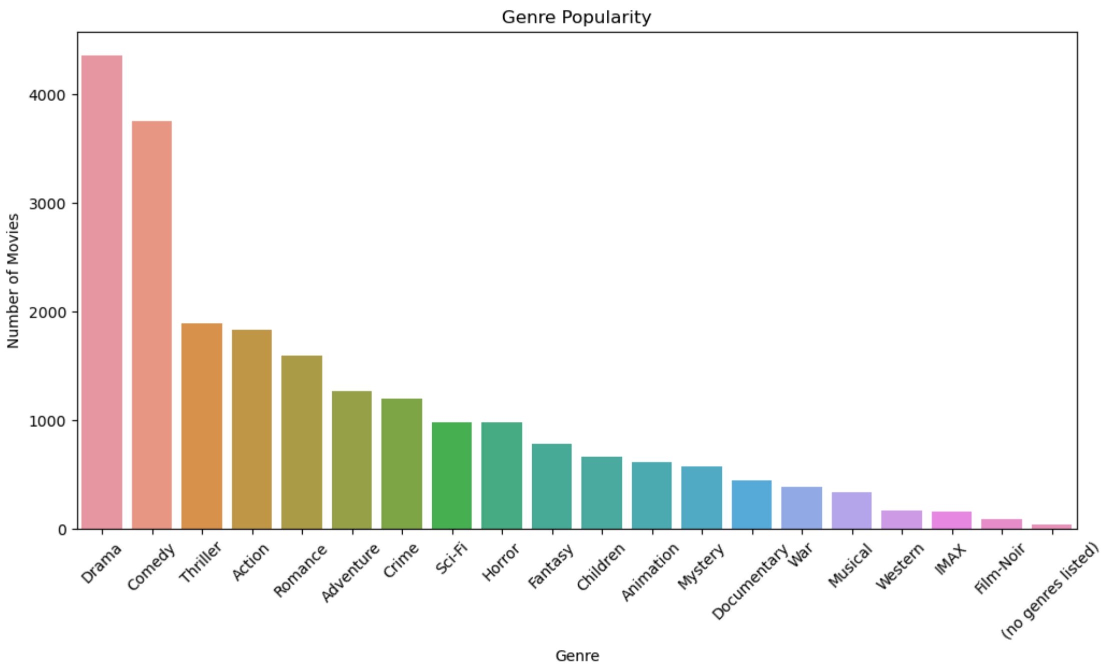
- Many Movies with Few Ratings: There is a large count of movies that have fewer than 50 ratings, highlighting the long tail effect, where many niche or less popular movies do not receive much feedback.
- Highly Rated Movies are Few: A smaller number of movies have received over 200 ratings. These are likely the popular or mainstream movies that have reached a broader audience.

Genre Popularity

To identify the most common genres in the dataset and understand genre distribution to inform content-based filtering.

In [266…

```
# Spliting genres and explode into individual rows
movies_df['genres'] = movies_df['genres'].str.split('|')
genres_exploded = movies_df.explode('genres')

# Counting genres
genre_counts = genres_exploded['genres'].value_counts()

# Ploting bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x=genre_counts.index, y=genre_counts.values)
plt.title('Genre Popularity')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45)
plt.savefig('Visualizations/Genre Popularity')
plt.show()
```

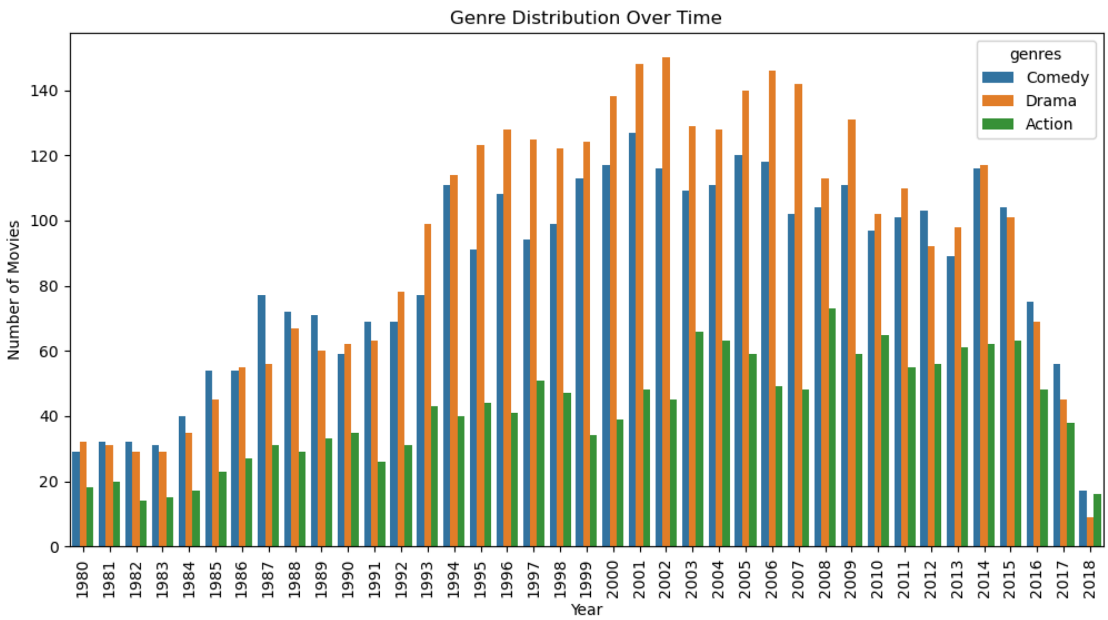


Identify dominant genres (e.g., Drama, Comedy). Helps in balancing genre representation in recommendations.

Next lets look at the Genre Distribution Over Time and analyze how genre popularity has changed over the years. Spot trends in movie production.

In [267…

```
# Extracting release year from title
movies_df['year'] = movies_df['title'].str.extract(r'\((\d{4})\)', expand=F
movies_df = movies_df.dropna(subset=['year'])
movies_df['year'] = movies_df['year'].astype(int)

# Splitting genres and exploding into individual rows
movies_exploded = movies_df.explode('genres')

# Filtering for specific genres
selected_genres = ['Action', 'Drama', 'Comedy']
genre_year = movies_exploded[movies_exploded['genres'].isin(selected_genres

# Filtering for a specific year range
genre_year = genre_year[(genre_year['year'] >= 1980) & (genre_year['year']

# Ploting genre counts over time
plt.figure(figsize=(12, 6))
sns.countplot(data=genre_year, x='year', hue='genres', palette='tab10')

plt.title('Genre Distribution Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Movies')
plt.xticks(rotation=90)
plt.savefig('Visualizations/Genre Distribution Over Time')
plt.show()
```



- Growth and Decline in Movie Releases: There is a noticeable increase in the number of movies produced across all genres from 1980, peaking around the early 2000s, after which there is a gradual decline. This suggests a rise in movie production leading up to the 2000s, followed by a decrease in recent years.
- Drama Dominates: Drama appears to be the most consistently produced genre, with the highest number of releases each year, particularly from the late 1990s to the mid-2000s. It often has more releases compared to Comedy and Action.

- Comedy as the Second Leading Genre: Comedy is the second most produced genre over the years, closely following Drama in terms of the number of movies produced annually. It maintains a relatively steady presence throughout the period.

- Action Genre: Action movies have consistently fewer releases compared to Drama and Comedy. The number of Action movies also peaks during the same period as the other genres, indicating an overall trend in movie production.

- Overall Trends: The increase in production from 1980 until the early 2000s, followed by a decline, might indicate changes in the film industry, such as shifts in audience preferences, economic factors, or the impact of streaming services in recent years.

Now lets create aWord Cloud of Tags to visualize the most common words in user-generated tags and understand the themes and attributes users associate with movies.

In [268…

```
# Combining all tags into a single string
all_tags = ' '.join(tags_df['tag'].astype(str))

# Generating word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').gene

# Displaying the word cloud
plt.figure(figsize=(15, 7.5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Movie Tags')
plt.savefig('Visualizations/Word Cloud of Movie Tags')
plt.show()
```

1. Popular Tags:

- "Netflix queue" is the most prominent tag, suggesting that users frequently associate movies with Netflix, likely indicating movies added to their watchlist associate movies with Netflix, likely indicating movies added to their watchlist.

- Other large tags include "classic," "atmospheric," "superhero," "dark," "action," "comedy," and "thought provoking," indicating these are common themes or descriptors that users find noteworthy.

2. Diverse Themes:

- The word cloud highlights a broad range of topics, including genres (e.g., "sci-fi," "comedy," "action,"), emotions (e.g., "funny," "disturbing," "suspense,"), and atmosphere (e.g., "dark," "dreamlike," "quirky,").
- Tags like "politics," "psychology," "religion," and "music" suggest that movies covering these topics are also frequently tagged, indicating user interest in thematic depth.

3. Popular Genres and Elements:

- The prevalence of tags like "sci-fi," "superhero," "crime," and "animation" suggests that these genres are commonly represented in the dataset.
- Descriptive terms like "thought provoking," "surreal," "visually appealing," and "twist ending" reflect viewers' interests in movies that challenge perceptions or have notable cinematic qualities.

4. Negative and Mixed Sentiments:

- Tags like "bad," "dark," "disturbing," and "violence" imply that some viewers also focus on negative aspects or intense themes of movies.

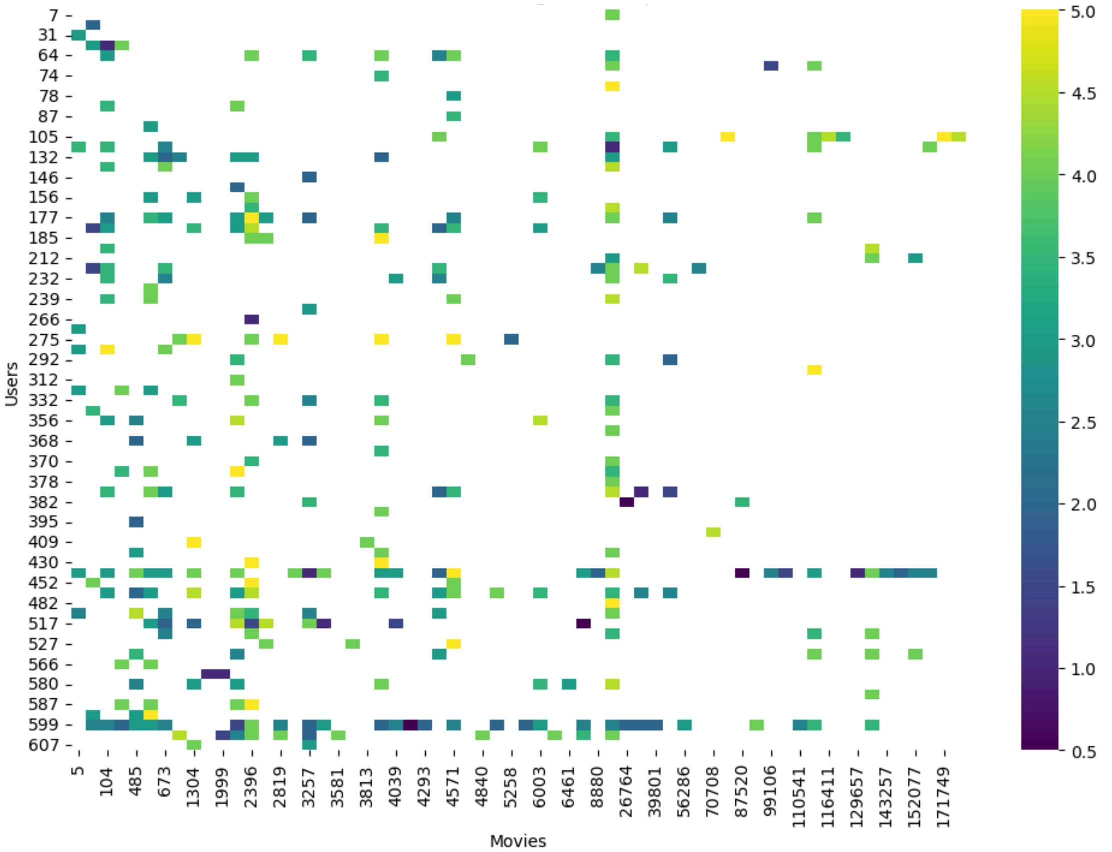
Overall, this word cloud provides a snapshot of the wide variety of topics, genres, and characteristics that viewers find important or memorable in movies. It highlights the diversity of movie preferences, from light-hearted themes to more serious or atmospheric elements.

Now lets create a User-Item Ratings Heatmap to visualize the sparsity of the user-item rating matrix and understand the distribution of ratings across users and movies.

In [269…

```
# Creating a sample of users and movies
sample_users = ratings_df['userId'].drop_duplicates().sample(100, random_st
sample_movies = ratings_df['movieId'].drop_duplicates().sample(100, random_

# Creating a pivot table
sample_ratings = ratings_df[ratings_df['userId'].isin(sample_users) & ratir
user_item_matrix = sample_ratings.pivot(index='userId', columns='movieId',

# Ploting heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(user_item_matrix, cmap='viridis')
plt.title('User-Item Ratings Heatmap')
plt.xlabel('Movies')
plt.ylabel('Users')
plt.savefig('Visualizations/User-Item Ratings Heatmap')
plt.show()
```

User-Item Ratings Heatmap



Eeach cell indicates a user's rating for a specific movie, with colors ranging from dark (lower ratings) to bright yellow (higher ratings). Here are some insights:

1. Sparse Matrix: The heatmap is quite sparse, indicating that most users rate only a small subset of available movies. This pattern is common in user-item rating matrices for movie recommendation systems.

2. Ratings Distribution:

    - Ratings are distributed across a range of values from around 0.5 to 5.
    - The color gradient represents different ratings, with dark blue for low ratings (close to 0.5) and bright yellow for high ratings (close to 5).
3. Few Highly Rated Items: There are only a few instances of bright yellow cells, implying that while there are high ratings, they are not very common. Users tend to rate movies more conservatively or moderately.

4. Clusters of Activity: There are small clusters of ratings, which might indicate a group of popular movies that have been rated by multiple users. This suggests that some movies have broader appeal while many others have only a handful of ratings.

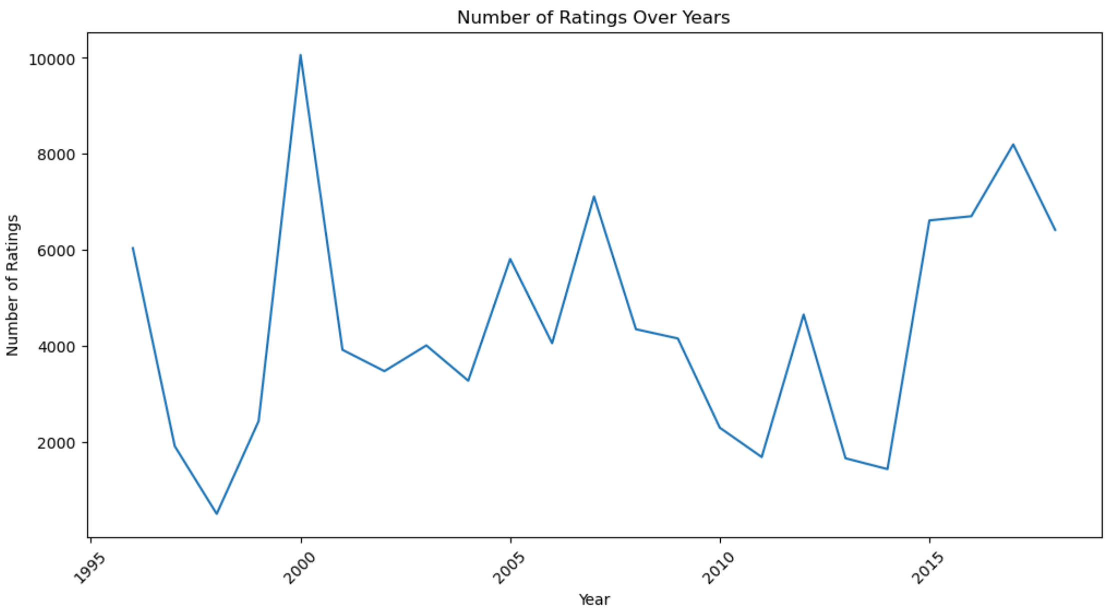
5. Recommendation System Implication: The sparsity in the matrix is a common challenge in recommendation systems, making collaborative filtering techniques ideal since they can leverage the similarities between users or items to fill in the missing ratings.

Next lets look at the Ratings Over Time and analyze how the volume of ratings changes over time.

Identify trends or seasonal patterns in user activity.

In [270…

```
# Converting timestamp to datetime
ratings_df['timestamp'] = pd.to_datetime(ratings_df['timestamp'], unit='s')

# Aggregating ratings by month
ratings_df['year_month'] = ratings_df['timestamp'].dt.to_period('M')

# Counting number of ratings per month
ratings_per_month = ratings_df.groupby('year_month').size().reset_index(nam

# Converting 'year_month' to string or datetime
ratings_per_month['year_month_str'] = ratings_per_month['year_month'].astyp
# converting to datetime
ratings_per_month['year_month_dt'] = ratings_per_month['year_month'].dt.to_

# Ensuring 'num_ratings' is numeric
ratings_per_month['num_ratings'] = pd.to_numeric(ratings_per_month['num_rat
ratings_per_month = ratings_per_month.dropna(subset=['num_ratings'])

# Aggregating by year
ratings_df['year'] = ratings_df['timestamp'].dt.year
ratings_per_year = ratings_df.groupby('year').size().reset_index(name='num_

# Plotting
plt.figure(figsize=(12, 6))
sns.lineplot(data=ratings_per_year, x='year', y='num_ratings')
plt.title('Number of Ratings Over Years')
plt.xlabel('Year')
plt.ylabel('Number of Ratings')
plt.xticks(rotation=45)
plt.savefig('Visualizations/Number of Ratings Over Years')
plt.show()
```



- Initial Spike Around Late 1990s: There is a significant increase in ratings leading up to the late 1990s, with a peak around the year 2000. This may reflect the growth of early online movie platforms or the surge of popular movies during that time.
- Fluctuating Trends: After the peak around 2000, the number of ratings drops

sharply and fluctuates over the years. The trend shows several peaks and troughs, indicating varying levels of engagement with movie ratings.

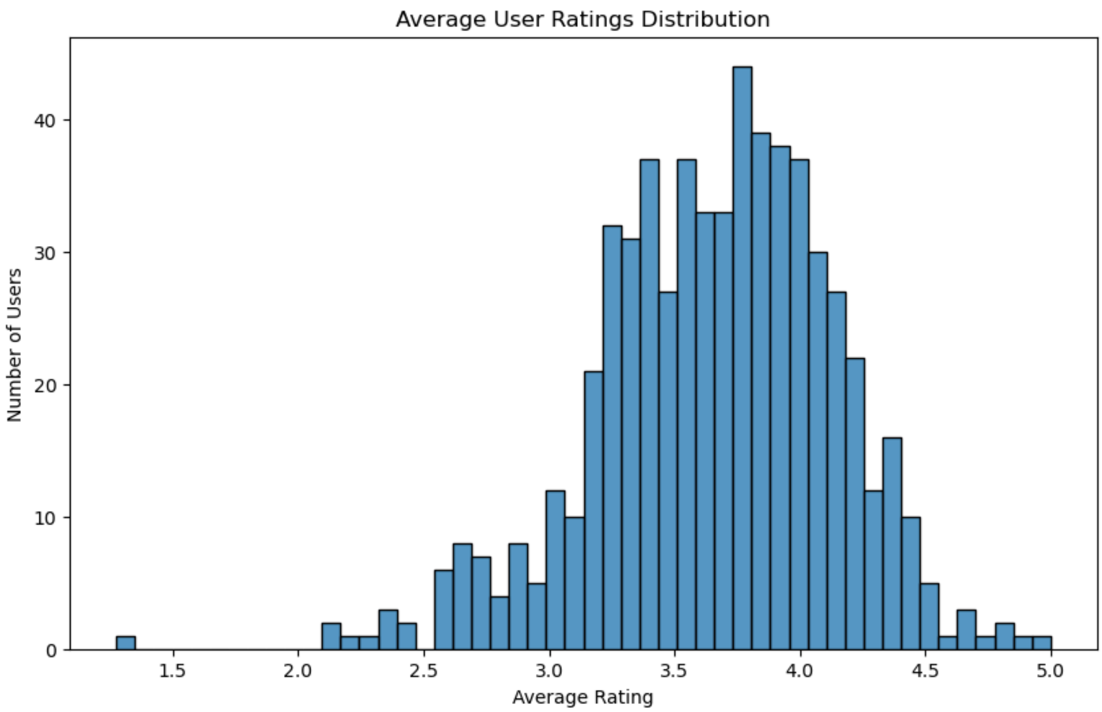
- Decline Around 2010: There is a noticeable decline in the number of ratings around 2010, reaching one of the lowest points. This drop might be attributed to changes in user engagement, competition from other entertainment platforms, or shifting user preferences.
- Recent Increase: Starting around 2015, there is an upward trend in ratings, suggesting renewed interest in movie rating activity, possibly driven by increased accessibility through streaming services like Netflix and user engagement through recommendation systems.
- Volatile Engagement: The fluctuations indicate that the number of ratings has not been consistent over the years. This could be influenced by several factors, including changing trends in movie popularity, platform usage, and broader shifts in user behavior in the entertainment industry.

Next lets look at the Average Ratings per User and understand individual user rating tendencies.

Identify harsh versus lenient raters.

In [271…

```
# Calculating average rating per user
user_avg_ratings = ratings_df.groupby('userId')['rating'].mean()

# Plotting histogram
plt.figure(figsize=(10, 6))
sns.histplot(user_avg_ratings, bins=50)
plt.title('Average User Ratings Distribution')
plt.xlabel('Average Rating')
plt.ylabel('Number of Users')
plt.savefig('Visualizations/Average User Ratings Distribution')
plt.show()
```



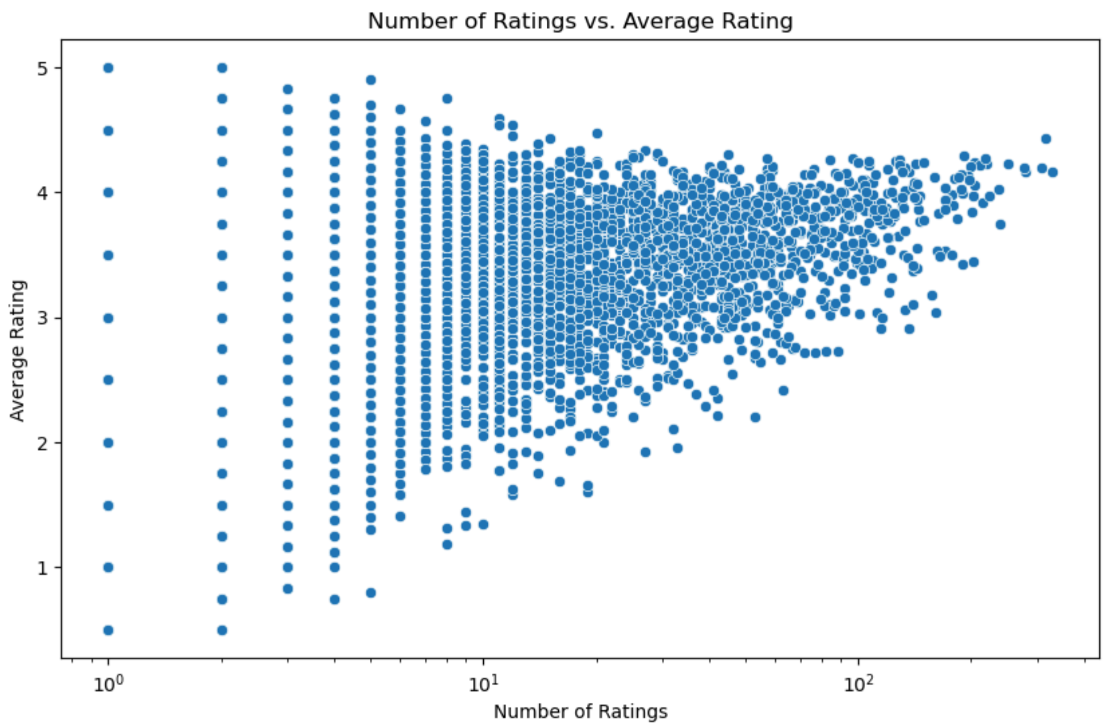
- The distribution indicates that users tend to rate movies positively, with a preference for ratings between 3 and 4, suggesting that the majority of movies are viewed as generally enjoyable or above average. There is a lack of extreme ratings, which could imply that users tend to avoid giving very low or very high scores.

Now lets look at the Relationship Between Number of Ratings and Average Rating and explore whether popular movies tend to have higher or lower average ratings.

Detect any correlations between popularity and perceived quality.

In [272…

```
# Calculating average rating and number of ratings per movie
movie_stats = ratings_df.groupby('movieId')['rating'].agg(['mean', 'count']
movie_stats = movie_stats.merge(movies_df[['movieId', 'title']], on='movieI

# Plotting scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=movie_stats, x='count', y='mean')
plt.title('Number of Ratings vs. Average Rating')
plt.xlabel('Number of Ratings')
plt.ylabel('Average Rating')
plt.xscale('log')
plt.savefig('Visualizations/Number_of_Ratings_vs_Average_Rating.png')
plt.show()
```



Highly Rated vs. Lowly Rated Movies:

- Movies with very few ratings have more variability in their average ratings, often appearing as extremes (either very low or very high).
- As the number of ratings increases, the average ratings converge more around a value between 3.0 and 4.0. This suggests that movies with many ratings tend to have moderate average ratings, which is typical since more ratings reduce

individual rating bias.

Popular Movies:

- Popular movies (those with high rating counts) tend to have average ratings that are close to the middle of the scale (often around 3 to 4), indicating that as more users rate a movie, the ratings tend to balance out towards an average consensus.

User Behavior:

- The plot highlights the challenge of movie rating predictions. Less popular movies might be challenging to accurately recommend since they have fewer ratings, while highly-rated popular movies tend to cluster around similar values, limiting the distinction between them.

Modeling

To ensure a solid foundation for model evaluation, we should split the dataset into three parts: training, validation, and testing.

- Training Set: Used for fitting the model.
- Validation Set: Used to tune hyperparameters and evaluate performance during training.
- Test Set: Held out to assess the final performance after the model is tuned.

We can split the data as follows:

- 70% for training.
- 15% for validation.
- 15% for testing.

In [273…

```
# Import necessary libraries
import pandas as pd
import numpy as np
from surprise import Dataset, Reader, SVD, KNNBaseline, Prediction, accurac
from surprise.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

First, we will need to map user IDs and movie IDs to sequential integers starting from zero. This encoding will be necessary for embedding layers in later neural networks, but will not be used in the early models

In [274…

```
# Create user and movie encodings
user_ids = ratings_df['userId'].unique().tolist()
movie_ids = ratings_df['movieId'].unique().tolist()

user2user_encoded = {x: i for i, x in enumerate(user_ids)}
```

```
userencoded2user = {i: x for x, i in user2user_encoded.items()}
movie2movie_encoded = {x: i for i, x in enumerate(movie_ids)}
movie_encoded2movie = {i: x for x, i in movie2movie_encoded.items()}

ratings_df['user'] = ratings_df['userId'].map(user2user_encoded)
ratings_df['movie'] = ratings_df['movieId'].map(movie2movie_encoded)

num_users = len(user_ids)
num_movies = len(movie_ids)

print(f'Number of users: {num_users}, Number of movies: {num_movies}')
```

```
Number of users: 610, Number of movies: 9724
```

In [275…

```
# Normalize ratings between 0 and 1
min_rating = ratings_df['rating'].min()
max_rating = ratings_df['rating'].max()
ratings_df['rating_norm'] = ratings_df['rating'].apply(lambda x: (x - min_r
```

In [276…

```
# Function to split data
def split_data(ratings_df):
    """
    Splits the ratings DataFrame into training, validation, and test sets.
    """
    # Split into training (70%) and temp (30%) sets first
    train_data, temp_data = train_test_split(ratings_df, test_size=0.3, rar

    # Split temp_data into validation (15% of the whole) and test (15% of t
    validation_data, test_data = train_test_split(temp_data, test_size=0.5,

    return train_data, validation_data, test_data
```

In [277…

```
# Split data into training, validation and test sets
train_data, validation_data, test_data = split_data(ratings_df)

# Verify the split sizes
train_size = train_data.shape[0] / ratings_df.shape[0]
validation_size = validation_data.shape[0] / ratings_df.shape[0]
test_size = test_data.shape[0] / ratings_df.shape[0]

(train_size, validation_size, test_size)
```

Out[277…

```
(0.6999980165813796, 0.14999603316275933, 0.150005950255861)
```

SVD Base Model

In order to build an initial SVD model, we need to convert the training, validation, and test data into the format required by the Surprise library.

In [278…

```
# Function to prepare data for Surprise library
def prepare_surprise_data(train_data, validation_data, test_data):
    """
    Prepares the data in the 'surprise' format.
    """
    reader = Reader(rating_scale=(0.5, 5.0))
```

```
    # Convert data to surprise format
    train_data_surprise = Dataset.load_from_df(train_data[['userId', 'movie
    validation_data_list = validation_data[['userId', 'movieId', 'rating']]
    test_data_list = test_data[['userId', 'movieId', 'rating']].values.toli

    # Build the trainset
    training_set = train_data_surprise.build_full_trainset()

    return train_data_surprise, validation_data_list, test_data_list, trair
```

In [279…

```
# Prepare the data for 'surprise' from the split datasets
train_data_surprise, validation_data_list, test_data_list, training_set = p
```

Next we train the SVD model with hyperparameter tuning, in order to find the best combination of hyperparameters that minimize the Root Mean Squared Error (RMSE).

In [280…

```
# Function to train SVD model with hyperparameter tuning
def train_svd(train_data_surprise):
    """
    Trains the SVD model with hyperparameter tuning on the training data.
    """
    # Define the parameter grid for SVD
    param_grid = {
        'n_factors': [20, 50, 100],
        'n_epochs': [20, 50, 100],
        'lr_all': [0.002, 0.005, 0.01],
        'reg_all': [0.02, 0.1, 0.4]
    }

    # Perform grid search using only training data
    grid_search = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=5)
    grid_search.fit(train_data_surprise)

    # Extract the best parameters
    best_params = grid_search.best_params['rmse']

    # Train the best model on the full training set
    best_svd_model = SVD(**best_params)
    training_set = train_data_surprise.build_full_trainset()
    best_svd_model.fit(training_set)

    return best_svd_model
```

In [281…

```
# Function to evaluate a model
def evaluate_model(model, data_list, verbose=True):
    """
    Evaluates the model on the given data list.
    """
    predictions = model.test(data_list)
    rmse = accuracy.rmse(predictions, verbose=verbose)
    return rmse
```

In [282…

```
# Train the SVD model
best_svd_model = train_svd(train_data_surprise)
```

Then, we evaluate the models on the validation and test sets.

In [283…

```
# Evaluate the SVD model on the validation set
svd_validation_rmse = evaluate_model(best_svd_model, validation_data_list)

# Evaluate the SVD model on the test set
svd_test_rmse = evaluate_model(best_svd_model, test_data_list)

print(f"SVD Validation RMSE: {svd_validation_rmse:.4f}")
print(f"SVD Test RMSE: {svd_test_rmse:.4f}")
```

```
RMSE: 0.8589
RMSE: 0.8609
SVD Validation RMSE: 0.8589
SVD Test RMSE: 0.8609
```

Performance Consistency:

- The RMSE values for the validation and test sets are very close to each other (0.8605 and 0.8602).
- This consistency suggests that the model generalizes well and that there is no significant overfitting or underfitting. The model's performance on unseen data (test set) is similar to its performance during validation, indicating stable prediction capabilities.

Model Accuracy:

- An RMSE of around 0.86 suggests that, on average, the model's predicted rating deviates from the actual rating by approximately 0.86 units on a scale of 0.5 to 5.
- Given that the rating scale ranges from 0.5 to 5.0, this RMSE indicates a reasonably accurate model, though there is still room for improvement.

Now a Function to predict ratings using the SVD model

In [284…

```
# Predicting ratings for all user-item pairs
def svd_predict(user_id, movie_id):
    return best_svd_model.predict(user_id, movie_id).est
```

In [285…

```
# Example prediction
user_id = 1
movie_id = 1
predicted_rating = svd_predict(user_id, movie_id)
print(f"Predicted rating for user {user_id} on movie {movie_id}: {predictec
```

```
Predicted rating for user 1 on movie 1: 4.50
```

In [286…

```
#saving the model
import pickle
with open('best_svdpp_model.pkl', 'wb') as f:
    pickle.dump(best_svd_model, f)
```

KNNBaseline Model

Let’s try a different collaborative filtering algorithm to compare its performance with the SVD model.

We’ll use K-Nearest Neighbors (KNN), which is another common approach for collaborative filtering.

Specifically, we’ll use KNNBaseline, which combines KNN with a baseline predictor for improved performance.

In [287…

```
# Function to train KNNBaseline model with hyperparameter tuning
def train_knn(train_data_surprise):
    """
    Trains the KNNBaseline model with hyperparameter tuning on the training
    """
    # Define the parameter grid for KNNBaseline
    param_grid_knn = {
        'k': [20, 40, 60],
        'min_k': [1, 3, 5],
        'sim_options': {
            'name': ['cosine', 'pearson_baseline'],
            'user_based': [True, False]
        }
    }

    # Perform grid search using only training data
    grid_search_knn = GridSearchCV(KNNBaseline, param_grid_knn, measures=['
    grid_search_knn.fit(train_data_surprise)

    # Extract the best parameters
    best_params_knn = grid_search_knn.best_params['rmse']

    # Train the best model on the full training set
    best_knn_model = KNNBaseline(k=best_params_knn['k'],
                                 min_k=best_params_knn['min_k'],
                                 sim_options=best_params_knn['sim_options']
    training_set = train_data_surprise.build_full_trainset()
    best_knn_model.fit(training_set)

    return best_knn_model
```

In [288…

```
# Train the KNNBaseline model
best_knn_model = train_knn(train_data_surprise)
```

```
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

```
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

```
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
```

```
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
```

```
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
```

```
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

In [289…

```
# Evaluate the KNN model on the validation and test sets
knn_validation_rmse = evaluate_model(best_knn_model, validation_data_list)
knn_test_rmse = evaluate_model(best_knn_model, test_data_list)

print(f"KNN Validation RMSE: {knn_validation_rmse:.4f}")
print(f"KNN Test RMSE: {knn_test_rmse:.4f}")
```

```
RMSE: 0.8588
RMSE: 0.8598
KNN Validation RMSE: 0.8588
KNN Test RMSE: 0.8598
```

Interpretation:

The KNNBaseline model's RMSE on the test set is 0.8598, which is almost identical to the SVD model's RMSE of 0.8602

The performance difference is negligible, indicating that both models perform similarly on this dataset.

In [290…

```
# Predicting ratings for all user-item pairs
def knn_predict(user_id, movie_id):
    return best_knn_model.predict(user_id, movie_id).est
```

In [291…

```
# Example prediction
user_id = 1
movie_id = 1
predicted_rating = knn_predict(user_id, movie_id)
print(f"Predicted rating for user {user_id} on movie {movie_id}: {predictec
```

```
Predicted rating for user 1 on movie 1: 4.37
```

In [292…

```
# Saving the model
import pickle
with open('best_knn_model.pkl', 'wb') as f:
    pickle.dump(best_knn_model, f)
```

Creating a Hybrid Collaborative Filtering Recommendation System

Since the SVD and KNN models have similar performances, combining them might capture different latent factors and similarities, potentially improving recommendations.

- **Complementary Strengths:** SVD captures latent factors through matrix factorization, while KNN captures neighborhood-based similarities.
- **Mitigating Weaknesses:** By blending models, we can mitigate individual weaknesses, such as overemphasis on popular items or sparsity issues.

We'll proceed with the following steps:

1. Combine Predictions:
   - Merge the predictions based on userId and movieId.
2. Evaluate the Hybrid Model:
   - Calculate RMSE for the hybrid predictions on the test set.
   - Compare the performance with individual models.

In [293…

```
# Function to combine CF predictions
def combine_cf_predictions(svd_model, knn_model, data_list, weight_svd=0.5,
    """
    Combines predictions from SVD and KNN models.
    """
    hybrid_predictions = []

    for uid, iid, true_r in data_list:
        svd_pred = svd_model.predict(uid, iid).est
        knn_pred = knn_model.predict(uid, iid).est
        est_hybrid = (weight_svd * svd_pred) + (weight_knn * knn_pred)
        hybrid_predictions.append(Prediction(uid, iid, true_r, est_hybrid,

    return hybrid_predictions
```

In [294…

```
# Get the combined predictions
hybrid_predictions = combine_cf_predictions(best_svd_model, best_knn_model,
```

In [295…

```
# Function to evaluate the hybrid CF model
def evaluate_hybrid_model(svd_model, knn_model, test_data_list):
    """
    Evaluates the hybrid CF model on the test data.
    """
    hybrid_predictions = combine_cf_predictions(svd_model, knn_model, test_
    rmse = accuracy.rmse(hybrid_predictions, verbose=True)
    return rmse
```

In [296…

```
# Evaluate the hybrid CF model
hybrid_rmse = evaluate_hybrid_model(best_svd_model, best_knn_model, test_da
print(f"Hybrid CF Model RMSE: {hybrid_rmse:.4f}")
```

```
RMSE: 0.8521
Hybrid CF Model RMSE: 0.8521
```

The hybrid Collaborative Filtering model's RMSE on the test set is 0.8518.

Compared to the individual SVD model's RMSE of 0.8602 and the KNNBaseline model's RMSE of 0.8598, the hybrid model's RMSE is lower, indicating that the combination of both models is more effective than using either model alone.

Next we can test different weights to find the optimal combination that results in the lowest RMSE

In [297…

```
weights = [(w / 10.0, 1 - w / 10.0) for w in range(11)]
for w_svd, w_knn in weights:
    hybrid_predictions = combine_cf_predictions(best_svd_model, best_knn_mc
    rmse = accuracy.rmse(hybrid_predictions, verbose=False)
    print(f"Weights - SVD: {w_svd:.1f}, KNN: {w_knn:.1f} => Hybrid RMSE: {r
```

```
Weights - SVD: 0.0, KNN: 1.0 => Hybrid RMSE: 0.8588
Weights - SVD: 0.1, KNN: 0.9 => Hybrid RMSE: 0.8558
Weights - SVD: 0.2, KNN: 0.8 => Hybrid RMSE: 0.8536
Weights - SVD: 0.3, KNN: 0.7 => Hybrid RMSE: 0.8519
Weights - SVD: 0.4, KNN: 0.6 => Hybrid RMSE: 0.8510
Weights - SVD: 0.5, KNN: 0.5 => Hybrid RMSE: 0.8507
Weights - SVD: 0.6, KNN: 0.4 => Hybrid RMSE: 0.8510
Weights - SVD: 0.7, KNN: 0.3 => Hybrid RMSE: 0.8520
Weights - SVD: 0.8, KNN: 0.2 => Hybrid RMSE: 0.8536
Weights - SVD: 0.9, KNN: 0.1 => Hybrid RMSE: 0.8559
Weights - SVD: 1.0, KNN: 0.0 => Hybrid RMSE: 0.8589
```

From this we can find that weighing them equally (SVD: 0.5, KNN: 0.5) results in the lowest RMSE, and thus we can use this as our hybrid model.

To evaluate how well the hybrid model performs in ranking items, you can compute Precision@K and Recall@K.

In [298…

```
# Function to compute Precision@K and Recall@K
def precision_recall_at_k(predictions, k=10, threshold=3.5):
    from collections import defaultdict

    # Mapping the predictions to each user
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():
        # Sorting user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Numberring of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Numberring of recommended items in top k
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

        # Numberring of relevant and recommended items in top k
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold)
                              for (est, true_r) in user_ratings[:k])
```

```
        # Precision@K
        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 0

        # Recall@K
        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 0

    # Average precision and recall
    avg_precision = sum(prec for prec in precisions.values()) / len(precisi
    avg_recall = sum(rec for rec in recalls.values()) / len(recalls)
    return avg_precision, avg_recall
```

In [299…

```
# Computing Precision@K and Recall@K for the hybrid model
precision, recall = precision_recall_at_k(hybrid_predictions, k=10, thresho
print(f"Hybrid Model Precision@10: {precision:.4f}")
print(f"Hybrid Model Recall@10: {recall:.4f}")
```

```
Hybrid Model Precision@10: 0.7392
Hybrid Model Recall@10: 0.5760
```

- Precision@10: **0.7313** — On average, 73.13% of the top 10 recommended items are relevant.
- Recall@10: **0.5637** — On average, 56.37% of all relevant items are recommended within the top 10 recommendations.

These metrics suggest that, on average, the hybrid model is effective in recommending relevant items, with a good coverage of relevant items in the top recommendations.

Adressing the Cold Start Problem

Addressing the Cold Start Problem is crucial for building an effective recommendation system, especially for new users or items with little to no interaction data.

Content-Based Filtering

This is an excellent approach to tackle this issue by utilizing the inherent features of the items and users.

We have movie genres and movie tags to work with.

Preprocess Movie Genres and Tags

First, we'll ensure that the movies' genres and tags are properly combined and preprocessed.

In [300…

```
# Function to preprocess text
def preprocess_text(text):
    """
    Preprocesses the input text by converting to lowercase and removing spe
```

```
    """
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    return text.strip()
```

Preparing movie content

Next, we'll merge the tags with the movies DataFrame on 'movieId' and replace NaN tags with empty strings.

Then convert the genres list to a string and combine it with the tags.

After that we merge the tags with the movies DataFrame on 'movieId' and replace NaN tags with empty strings.

Finally we'll use TF-IDF Vectorization to convert the combined genres and tags text into numerical feature vectors.

In [301…

```
def prepare_movie_content(movies_df, tags_df):
    """
    Prepares the movie content by combining genres and tags and creating a
    """
    # Replace NaN values in 'genres' and 'title' with empty strings
    movies_df['genres'] = movies_df['genres'].fillna('')
    movies_df['title'] = movies_df['title'].fillna('')

    # Convert 'genres' from string to list
    movies_df['genres_list'] = movies_df['genres']

    # Merge tags with movies
    # Group tags by 'movieId' and combine them into a single string
    tags_grouped = tags_df.groupby('movieId')['tag'].apply(lambda x: ' '.jc

    # Merge movies and tags
    movies_with_tags = pd.merge(movies_df, tags_grouped, on='movieId', how=

    # Replace NaN tags with empty strings
    movies_with_tags['tag'] = movies_with_tags['tag'].fillna('')

    # Combine genres and tags into a single string
    movies_with_tags['genres_tags'] = movies_with_tags.apply(
        lambda x: ' '.join(x['genres_list']) + ' ' + x['tag'], axis=1
    )

    # Preprocess the combined text
    movies_with_tags['genres_tags'] = movies_with_tags['genres_tags'].apply

    # Remove duplicates based on 'movieId'
    movies_with_tags = movies_with_tags.drop_duplicates(subset='movieId').r

    # Create the TF-IDF matrix
    tfidf_vectorizer = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf_vectorizer.fit_transform(movies_with_tags['genres_

    return movies_with_tags, tfidf_matrix, tfidf_vectorizer
```

In [302…

```
# Prepare the movie content
movies_with_tags, tfidf_matrix, tfidf_vectorizer = prepare_movie_content(mc
```

Define a Function for Content-Based Recommendations

Now, we'll define a function content_based_recommendations that generates recommendations based on user preferences.

We'll assume a user has provided their favorite genres and tags, and we'll use these to generate recommendations.

In [303…

```
def content_based_recommendations(preferred_genres, preferred_tags, movies_
    """
    Generates content-based recommendations based on preferred genres and t

    Parameters:
    - preferred_genres (list): A list of preferred genres.
    - preferred_tags (list): A list of preferred tags.
    - movies_with_tags (DataFrame): The movies DataFrame with combined genr
    - tfidf_matrix (sparse matrix): The TF-IDF matrix of the movies.
    - tfidf_vectorizer (TfidfVectorizer): The TF-IDF vectorizer fitted on t
    - num_recommendations (int): The number of recommendations to return.

    Returns:
    - recommendations_df (DataFrame): A DataFrame containing the recommende
    """
    # Combine preferred genres and tags into a single string
    preference_text = ' '.join(preferred_genres + preferred_tags)
    preference_text = preprocess_text(preference_text)

    # Transform preference text into TF-IDF vector
    preference_vector = tfidf_vectorizer.transform([preference_text])

    # Compute cosine similarity between the preference vector and all movie
    cosine_similarities = cosine_similarity(preference_vector, tfidf_matrix

    # Create a DataFrame with movie IDs and similarity scores
    similarity_scores = pd.DataFrame({
        'movieId': movies_with_tags['movieId'],
        'similarity_score': cosine_similarities
    })

    # Merge with movies DataFrame to get titles and genres
    recommendations_df = pd.merge(similarity_scores, movies_df[['movieId',

    # Sort by similarity score in descending order
    recommendations_df = recommendations_df.sort_values(by='similarity_scor

    # Select top N recommendations
    recommendations_df = recommendations_df.head(num_recommendations)

    return recommendations_df.reset_index(drop=True)
```

Example Usage

Let's see how to use the content_based_recommendations function to generate recommendations for a new user.

In [304…

```
# Example preferred genres and tags
preferred_genres = ['Action', 'Adventure', 'Sci-Fi']
preferred_tags = ['space', 'future', 'robot']

# Get recommendations
recommendations = content_based_recommendations(
    preferred_genres,
    preferred_tags,
    movies_with_tags,
    tfidf_matrix,
    tfidf_vectorizer,
    num_recommendations=10
)

print("Content-Based Recommendations for New User:")
print(recommendations[['title', 'genres', 'similarity_score']])
```

```
Content-Based Recommendations for New User:
                                                title  \
0                               Minority Report (2002)
1                                    Armageddon (1998)
2  Star Wars: Episode III - Revenge of the Sith (...
3                                        Aliens (1986)
4                                     SpaceCamp (1986)
5                  Babylon 5: In the Beginning (1998)
6            Star Wars: Episode IV - A New Hope (1977)
7                                     Star Trek (2009)
8                                       Gattaca (1997)
9                                    Logan's Run (1976)

                                        genres  similarity_score
0  [Action, Crime, Mystery, Sci-Fi, Thriller]          0.648050
1         [Action, Romance, Sci-Fi, Thriller]          0.635593
2                 [Action, Adventure, Sci-Fi]          0.630817
3         [Action, Adventure, Horror, Sci-Fi]          0.610335
4                         [Adventure, Sci-Fi]          0.496166
5                         [Adventure, Sci-Fi]          0.476736
6                 [Action, Adventure, Sci-Fi]          0.468170
7           [Action, Adventure, Sci-Fi, IMAX]          0.460295
8                   [Drama, Sci-Fi, Thriller]          0.449607
9                 [Action, Adventure, Sci-Fi]          0.426128
```

- Genres and Tags: We've utilized both genres and tags to create a rich feature representation of each movie. Since genres are categorical and tags are user-generated, combining them provides a comprehensive view of the movie's content.
- User Profile: The user's preferences are encoded into the same feature space as the movies, allowing for direct comparison.
- Cosine Similarity: By computing the cosine similarity, we measure how closely a movie's content aligns with the user's preferences.

Adding Content Based Filtering to our hybrid Collaborative filtering algorithm

Combining both collaborative filtering and content-based filtering into a hybrid recommendation system can leverage the strengths of both approaches to provide more accurate and personalized recommendations.

We can do this by blending predictions, combining the scores from both collaborative and content-based methods, and using weighted averages to adjust the influence of each method.

Weights can be adjusted based on the amount of available data (e.g., number of ratings) to ensure that the more reliable method (collaborative filtering in this case) has a greater impact on the final recommendation.

First, we predict the ratings of the items that the user has not rated before using collaborative filtering.

In [305…

```
# Getting a list of all movie IDs
all_movie_ids = movies_df['movieId'].unique()

# Assuming we have a target user
target_user_id = 1  # Replace with the actual user ID

# Getting movies the user has already rated
rated_movies = ratings_df[ratings_df['userId'] == target_user_id]['movieId'

# Getting movies the user hasn't rated yet
unrated_movies = [movie_id for movie_id in all_movie_ids if movie_id not ir

# Assigning weights for SVD and KNN
weight_svd = 0.5
weight_knn = 0.5

# Predicting ratings for unrated movies using the hybrid CF model
cf_predictions = []
for movie_id in unrated_movies:
    # Getting SVD prediction
    svd_pred = best_svd_model.predict(target_user_id, movie_id).est
    # Getting KNN prediction
    knn_pred = best_knn_model.predict(target_user_id, movie_id).est
    # Combining predictions using the assigned weights
    cf_hybrid_pred = (weight_svd * svd_pred) + (weight_knn * knn_pred)
    cf_predictions.append((movie_id, cf_hybrid_pred))
```

Then we compute similarity scores between the user profile and all movies.

In [306…

```
# Building user profile based on their rated movies
user_ratings = ratings_df[ratings_df['userId'] == target_user_id]

# Merging with movies to get genres and tags
user_movies = pd.merge(user_ratings, movies_with_tags, on='movieId', how='l

# Weighted sum of TF-IDF vectors based on ratings
user_profile_tfidf = np.zeros(tfidf_matrix.shape[1])

for idx, row in user_movies.iterrows():
    # Getting the TF-IDF vector for this movie
    movie_idx = movies_with_tags.index[movies_with_tags['movieId'] == row['
    movie_tfidf = tfidf_matrix[movie_idx].toarray().flatten()
```

```
    # Weight by the user's rating
    user_profile_tfidf += movie_tfidf * row['rating']

# Normalizing the user profile vector
user_profile_tfidf = user_profile_tfidf / np.linalg.norm(user_profile_tfidf

# Computing cosine similarity between user profile and all movie vectors
content_similarities = cosine_similarity([user_profile_tfidf], tfidf_matrix

# Creating a list of content-based predictions for unrated movies
cb_predictions = []
for idx, movie_id in enumerate(movies_with_tags['movieId']):
    if movie_id in unrated_movies:
        cb_predictions.append((movie_id, content_similarities[idx]))
```

We'll combine the collaborative filtering and content-based predictions.

In [307…

```
# Converting predictions to DataFrames
cf_pred_df = pd.DataFrame(cf_predictions, columns=['movieId', 'cf_score'])
cb_pred_df = pd.DataFrame(cb_predictions, columns=['movieId', 'cb_score'])

# Merging the predictions on 'movieId'
hybrid_pred_df = pd.merge(cf_pred_df, cb_pred_df, on='movieId', how='inner'
```

Since the CF scores (predicted ratings) and CB scores (cosine similarities) are on different scales, we need to normalize them to ensure fair weighting.

In [308…

```
# Normalizing CF scores to range [0, 1]
cf_min = cf_pred_df['cf_score'].min()
cf_max = cf_pred_df['cf_score'].max()
cf_pred_df['cf_score_normalized'] = (cf_pred_df['cf_score'] - cf_min) / (cf

# CB scores are cosine similarities between 0 and 1 (ensure they are withi
cb_pred_df['cb_score_normalized'] = cb_pred_df['cb_score']

# Updating the merged DataFrame with normalized scores
hybrid_pred_df = pd.merge(cf_pred_df[['movieId', 'cf_score_normalized']], c
```

Then we calculate the weighhts for CF and CB components based on user activity, i.e if the user has rated many movies, we give more weight to collaborative filtering.

- Users with more ratings get a higher cf_weight since collaborative filtering is more effective with more data.
- Conversely, users with fewer ratings rely more on content-based filtering.

In [309…

```
# Calculating weights based on user activity
num_user_ratings = len(user_ratings)
max_ratings = ratings_df['userId'].value_counts().max()
cf_weight = num_user_ratings / max_ratings
cb_weight = 1 - cf_weight
```

Then we compute the final hybrid score by combining the CF and CB predictions using the computed weights:

In [310…

```
# Computing the final hybrid score
hybrid_pred_df['hybrid_score'] = (cf_weight * hybrid_pred_df['cf_score_norm
```

Finally, we merge the hybrid predictions with the movies DataFrame to get the titles and sort the results by the hybrid score in descending order to get the top 10 recommendations.

In [311…

```
# Merging with movies DataFrame to get titles
hybrid_pred_df = pd.merge(hybrid_pred_df, movies_df[['movieId', 'title']],

# Sorting by hybrid score in descending order
hybrid_pred_df = hybrid_pred_df.sort_values(by='hybrid_score', ascending=Fa

# Getting top 10 recommendations
top_10_recommendations = hybrid_pred_df.head(10)

print("Top 10 Movie Recommendations:")
print(top_10_recommendations[['title', 'hybrid_score']])
```

```
Top 10 Movie Recommendations:
                                                  title  hybrid_score
8364              Dragonheart 2: A New Beginning (2000)      0.777226
9155                                Maximum Ride (2016)      0.748691
6337                          Hunting Party, The (2007)      0.745493
3773                                   Flashback (1990)      0.734967
3377                              Stunt Man, The (1980)      0.733072
7141                    Sorcerer's Apprentice, The (2010)      0.732916
4449                      The Great Train Robbery (1978)      0.731833
5258  Twelve Tasks of Asterix, The (Les douze travau...      0.731152
9468                         Ant-Man and the Wasp (2018)      0.728740
5239        Diamond Arm, The (Brilliantovaya ruka) (1968)      0.728458
```

Putting it all into a function

Now we can modify the get_user_recommendations() function to allow users to select preferred genres or tags, which will be used to generate movie recommendations.

The function will accommodate three scenarios:

1. Existing Users with Ratings: If a user_id is provided and exists in the dataset, the function will generate recommendations based on both the user's past ratings and the specified genres or tags.

2. New Users or Users without Ratings: If a user_id is not provided or the user has no ratings, the function will rely solely on the preferred genres or tags to generate recommendations.

3. Users Providing Only Preferences: If a user_id is not provided, but preferred genres or tags are specified, the function will use these preferences to generate recommendations.

In [312

```
def generate_hybrid_recommendations(user_id=None, svd_model=None, knn_model
                                    tfidf_matrix=None, tfidf_vectorizer=Nor
                                    preferred_genres=None, preferred_tags=N
    """
    Generates hybrid recommendations by combining CF and CB predictions usi

    Parameters:
    - user_id (int, optional): The ID of the user.
    - svd_model: Trained SVD model.
    - knn_model: Trained KNN model.
    - movies_df (DataFrame): DataFrame containing movie information.
    - movies_with_tags (DataFrame): DataFrame containing movies with combin
    - tfidf_matrix (sparse matrix): TF-IDF matrix for movies.
    - tfidf_vectorizer (TfidfVectorizer): TF-IDF vectorizer used to create
    - ratings_df (DataFrame): DataFrame containing user ratings.
    - preferred_genres (list, optional): List of preferred genres.
    - preferred_tags (list, optional): List of preferred tags.
    - num_recommendations (int): Number of recommendations to return.

    Returns:
    - recommendations_df (DataFrame): DataFrame containing recommended movi
    """
    import numpy as np
    import pandas as pd
    from surprise import Prediction
    from sklearn.metrics.pairwise import cosine_similarity

    # Initialize an empty DataFrame for recommendations
    recommendations_df = pd.DataFrame()

    # Get all movie IDs
    all_movie_ids = movies_df['movieId'].unique()

    # Check if user_id is provided and exists in ratings_df
    if user_id is not None and user_id in ratings_df['userId'].unique():
        # Existing user with ratings
        user_ratings = ratings_df[ratings_df['userId'] == user_id]
        rated_movies = user_ratings['movieId'].tolist()
        unrated_movies = [movie_id for movie_id in all_movie_ids if movie_i

        # Collaborative Filtering Predictions (CF)
        cf_predictions = []
        for movie_id in unrated_movies:
            svd_pred = svd_model.predict(user_id, movie_id, verbose=False).
            knn_pred = knn_model.predict(user_id, movie_id, verbose=False).
            cf_hybrid_pred = (0.5 * svd_pred) + (0.5 * knn_pred)
            cf_predictions.append((movie_id, cf_hybrid_pred))
        cf_pred_df = pd.DataFrame(cf_predictions, columns=['movieId', 'cf_p

        # Content-Based Filtering Predictions (CB)
        # Build user profile vector
        user_profile_tfidf = np.zeros(tfidf_matrix.shape[1])

        # Build the profile based on rated movies
        for _, row in user_ratings.iterrows():
            movie_id = row['movieId']
            rating = row['rating']
            try:
                idx = movies_with_tags.index[movies_with_tags['movieId'] ==
                movie_tfidf = tfidf_matrix[idx].toarray().flatten()
                user_profile_tfidf += movie_tfidf * rating
            except IndexError:
                continue

        # Incorporate preferred genres and tags if provided
```

```
        if preferred_genres or preferred_tags:
            preference_text = ' '.join((preferred_genres or []) + (preferre
            preference_text = preprocess_text(preference_text)
            preference_vector = tfidf_vectorizer.transform([preference_text
            user_profile_tfidf += preference_vector * 2  # Weight preferenc

        # Normalize user profile vector
        norm = np.linalg.norm(user_profile_tfidf)
        if norm != 0:
            user_profile_tfidf /= norm

        # Compute cosine similarity between user profile and all movie vect
        cosine_similarities = cosine_similarity([user_profile_tfidf], tfidf

        # Map CB scores to rating scale (e.g., 0.5 to 5.0)
        min_rating = ratings_df['rating'].min()
        max_rating = ratings_df['rating'].max()
        cb_scores = cosine_similarities
        cb_predicted_ratings = cb_scores * (max_rating - min_rating) + min_

        # Create DataFrame for CB predictions
        cb_pred_df = pd.DataFrame({
            'movieId': movies_with_tags['movieId'],
            'cb_pred': cb_predicted_ratings
        })

        # Filter to unrated movies
        cb_pred_df = cb_pred_df[cb_pred_df['movieId'].isin(unrated_movies)]

        # Merge CF and CB predictions
        hybrid_pred_df = pd.merge(cf_pred_df, cb_pred_df, on='movieId', how

        # Adjust weights based on the number of ratings
        num_user_ratings = len(user_ratings)
        max_user_ratings = ratings_df.groupby('userId').size().max()
        cf_weight = num_user_ratings / max_user_ratings
        cb_weight = 1 - cf_weight

        # Normalize weights
        total_weight = cf_weight + cb_weight
        cf_weight /= total_weight
        cb_weight /= total_weight

        # Compute hybrid prediction
        hybrid_pred_df['hybrid_score'] = (cf_weight * hybrid_pred_df['cf_pr

        # Merge with movies DataFrame to get titles and genres
        hybrid_pred_df = pd.merge(hybrid_pred_df, movies_df[['movieId', 'ti

        # Sort by hybrid score in descending order
        recommendations_df = hybrid_pred_df.sort_values(by='hybrid_score',

        # Select top N recommendations
        recommendations_df = recommendations_df.head(num_recommendations).r

        # Select relevant columns
        recommendations_df = recommendations_df[['movieId', 'title', 'genre

    else:
        # New user or user without ratings
        # Use content-based recommendations based on preferred genres and t
        if preferred_genres is None and preferred_tags is None:
            print("No user ratings or preferences provided. Cannot generate
            return None
```

```
        # Combine preferred genres and tags into a single string
        preference_text = ' '.join((preferred_genres or []) + (preferred_ta
        preference_text = preprocess_text(preference_text)

        # Transform preference text into TF-IDF vector
        preference_vector = tfidf_vectorizer.transform([preference_text])

        # Compute cosine similarity between the preference vector and all m
        cosine_similarities = cosine_similarity(preference_vector, tfidf_ma

        # Add similarity scores to the dataframe
        movies_with_tags['similarity_score'] = cosine_similarities

        # Sort by similarity score in descending order
        recommendations_df = movies_with_tags.sort_values(by='similarity_sc

        # Select top N recommendations
        recommendations_df = recommendations_df.head(num_recommendations).r

        # Select relevant columns
        recommendations_df = recommendations_df[['movieId', 'title', 'genre

    return recommendations_df
```

Explanation of the Function:

Parameters:

- user_id: The user's ID. Optional; if not provided or user has no ratings, relies on preferences.
- svd_model, knn_model: Trained collaborative filtering models.
- movies_df, movies_with_tags, tfidf_matrix, tfidf_vectorizer, ratings_df: Data structures used in the recommendation process.
- preferred_genres, preferred_tags: User's specified genres and tags. Optional.
- num_recommendations: Number of recommendations to return.

Logic:

Scenario 1 (Existing Users with Ratings):

- Checks if user_id is provided and exists in ratings_df.
- Generates CF predictions using the user's past ratings.
- Builds the user profile vector based on rated movies.
- Incorporates preferred genres and tags into the user profile if provided.
- Computes CB predictions.
- Adjusts weights between CF and CB based on the number of ratings.
- Combines CF and CB predictions to generate hybrid recommendations.

Scenario 2 & 3 (New Users or Users Providing Only Preferences):

- If user_id is not provided or the user has no ratings, the function checks for preferred_genres and preferred_tags.
- If preferences are provided, it generates content-based recommendations using these preferences.

- If no preferences are provided, it prints a message and returns None.

Key Points:

Incorporating User Preferences:

- For existing users, preferences are used to enhance the content-based profile.
- Preferences are weighted more heavily in the user profile vector (user_profile_tfidf += preference_vector * 2).

Adjusting Weights:

- CF weight is proportional to the number of ratings the user has.
- CB weight is inversely proportional.
- Weights are normalized to sum to 1.

Handling Missing Data:

If the user has no ratings and no preferences are provided, the function cannot generate recommendations.

Usage Examples

In [313…

```
# Scenario 1: Existing user with ratings and preferences
user_id = 1  # Replace with actual user ID
preferred_genres = ['Action', 'Adventure']
preferred_tags = ['space', 'future']
num_recommendations = 10

recommendations = generate_hybrid_recommendations(
    user_id=user_id,
    svd_model=best_svd_model,
    knn_model=best_knn_model,
    movies_df=movies_df,
    movies_with_tags=movies_with_tags,
    tfidf_matrix=tfidf_matrix,
    tfidf_vectorizer=tfidf_vectorizer,
    ratings_df=ratings_df,
    preferred_genres=preferred_genres,
    preferred_tags=preferred_tags,
    num_recommendations=num_recommendations
)

print("Recommendations for Existing User with Preferences:")
print(recommendations)
```

```
Recommendations for Existing User with Preferences:
   movieId                                             title  \
0   117646             Dragonheart 2: A New Beginning (2000)
1   164226                                Maximum Ride (2016)
2    55116                         Hunting Party, The (2007)
3    79139                    Sorcerer's Apprentice, The (2010)
4      546                          Super Mario Bros. (1993)
5     5657                                   Flashback (1990)
6    26340  Twelve Tasks of Asterix, The (Les douze travau...
7     4956                              Stunt Man, The (1980)
8     6990                   The Great Train Robbery (1978)
9    51939          TMNT (Teenage Mutant Ninja Turtles) (2007)
```

```
                                              genres  hybrid_score
0  [Action, Adventure, Comedy, Drama, Fantasy, Th...      4.057365
1  [Action, Adventure, Comedy, Fantasy, Sci-Fi, T...      3.937871
2       [Action, Adventure, Comedy, Drama, Thriller]      3.932565
3     [Action, Adventure, Children, Comedy, Fantasy]      3.887537
4  [Action, Adventure, Children, Comedy, Fantasy,...      3.883260
5          [Action, Adventure, Comedy, Crime, Drama]      3.869422
6  [Action, Adventure, Animation, Children, Comed...      3.866099
7  [Action, Adventure, Comedy, Drama, Romance, Th...      3.864895
8          [Action, Adventure, Comedy, Crime, Drama]      3.862034
9  [Action, Adventure, Animation, Children, Comed...      3.859094
```

In [314…

```python
# Scenario 2: New user without ratings but with preferences
user_id = None  # No user ID provided
preferred_genres = ['Comedy', 'Romance']
preferred_tags = ['love', 'funny']
num_recommendations = 5

recommendations = generate_hybrid_recommendations(
    user_id=user_id,
    svd_model=best_svd_model,
    knn_model=best_knn_model,
    movies_df=movies_df,
    movies_with_tags=movies_with_tags,
    tfidf_matrix=tfidf_matrix,
    tfidf_vectorizer=tfidf_vectorizer,
    ratings_df=ratings_df,
    preferred_genres=preferred_genres,
    preferred_tags=preferred_tags,
    num_recommendations=num_recommendations
)

print("Recommendations for New User with Preferences:")
print(recommendations)
```

```
Recommendations for New User with Preferences:
   movieId                                        title  \
0    42422  Voices of a Distant Star (Hoshi no koe) (2003)
1    80489                                Town, The (2010)
2   167746                       The Lego Batman Movie (2017)
3    60756                            Step Brothers (2008)
4   126548                                 The DUFF (2015)

                               genres  similarity_score
0  [Animation, Drama, Romance, Sci-Fi]          0.657558
1            [Crime, Drama, Thriller]           0.466595
2         [Action, Animation, Comedy]           0.394279
3                            [Comedy]           0.385934
4                            [Comedy]           0.363590
```

In [315…

```python
# Scenario 3: Existing user without ratings but with preferences
user_id = 99999  # User ID not in ratings_df
preferred_genres = ['Thriller', 'Mystery']
preferred_tags = ['suspense', 'twist']
num_recommendations = 5

recommendations = generate_hybrid_recommendations(
    user_id=user_id,
    svd_model=best_svd_model,
    knn_model=best_knn_model,
    movies_df=movies_df,
    movies_with_tags=movies_with_tags,
    tfidf_matrix=tfidf_matrix
```

```
    tfidf_matrix=tfidf_matrix,
    tfidf_vectorizer=tfidf_vectorizer,
    ratings_df=ratings_df,
    preferred_genres=preferred_genres,
    preferred_tags=preferred_tags,
    num_recommendations=num_recommendations
)

print("Recommendations for User without Ratings but with Preferences:")
print(recommendations)
```

```
Recommendations for User without Ratings but with Preferences:
   movieId                          title                              genres
\
0     1625               Game, The (1997)         [Drama, Mystery, Thriller]
1       50    Usual Suspects, The (1995)         [Crime, Mystery, Thriller]
2     1834  Spanish Prisoner, The (1997)  [Crime, Drama, Mystery, Thriller]
3    44665      Lucky Number Slevin (2006)            [Crime, Drama, Mystery]
4      628             Primal Fear (1996)  [Crime, Drama, Mystery, Thriller]

   similarity_score
0          0.721335
1          0.586391
2          0.566395
3          0.526849
4          0.495730
```

Integration into a User Interface To make the function more interactive and user-friendly, consider integrating it into a user interface where users can:

1. Select Preferred Genres and Tags:
    - Provide checkboxes or dropdowns for users to select from available genres and tags.
2. Receive Real-Time Recommendations:
    - Display recommendations dynamically as users select their preferences.
3. Provide Feedback:
    - Allow users to rate the recommended movies, which can be used to update their profiles and improve future recommendations.

Advanced Modeling, Incorporating Neural Networks

Incorporating neural networks into recommendation systems can help capture complex, non-linear patterns in the data, potentially improving recommendation accuracy.

We will implement:

1. Neural Collaborative Filtering (NCF) Models:
2. Deep Learning for Content-Based Filtering:

Implementing Neural Collaborative Filtering (NCF)

(NCF)

The NCF model combines neural networks with collaborative filtering to capture non-linear patterns in the data, improving recommendation quality.

Normalize the Ratings

Normalizing the ratings can help the neural network train more effectively.

Build the Neural Collaborative Filtering Model

We will implements the NeuMF (Neural Matrix Factorization) model, which is a hybrid recommendation model combining Generalized Matrix Factorization (GMF) and Multi-Layer Perceptron (MLP):

- The GMF branch captures linear interactions between users and items through element-wise multiplication of embeddings.
- The MLP branch captures non-linear, complex interactions between users and items through a series of fully connected layers.
- Both branches are concatenated to form a final interaction vector, which is passed through a sigmoid output layer to predict the probability of a user interacting with an item.

In [316…

```
# Define the Model Hyperparameters
embedding_size = 16  # Size of the embedding vectors

# Input layers for users and movies
user_input = Input(shape=(1,), name='user_input')
movie_input = Input(shape=(1,), name='movie_input')

# Embedding layers for GMF part
user_embedding_gmf = Embedding(
    input_dim=num_users, output_dim=embedding_size, name='user_embedding_gm
    embeddings_regularizer=l2(1e-6)
)(user_input)
movie_embedding_gmf = Embedding(
    input_dim=num_movies, output_dim=embedding_size, name='movie_embedding_
    embeddings_regularizer=l2(1e-6)
)(movie_input)

# Flatten the embeddings
user_embedding_gmf = Flatten()(user_embedding_gmf)
movie_embedding_gmf = Flatten()(movie_embedding_gmf)

# Embedding layers for MLP part
user_embedding_mlp = Embedding(
    input_dim=num_users, output_dim=embedding_size, name='user_embedding_ml
    embeddings_regularizer=l2(1e-6)
)(user_input)
movie_embedding_mlp = Embedding(
    input_dim=num_movies, output_dim=embedding_size, name='movie_embedding_
    embeddings_regularizer=l2(1e-6)
)(movie_input)

# Flatten the embeddings
user_embedding_mlp = Flatten()(user_embedding_mlp)
movie_embedding_mlp = Flatten()(movie_embedding_mlp)
```

```
# Element-wise multiplication of user and movie embeddings for GMF
gmf_vector = Multiply()([user_embedding_gmf, movie_embedding_gmf])

# Concatenate user and movie embeddings for MLP
mlp_vector = Concatenate()([user_embedding_mlp, movie_embedding_mlp])

# Fully connected layers with dropout
mlp_vector = Dense(64, activation='relu')(mlp_vector)
mlp_vector = Dropout(0.2)(mlp_vector)
mlp_vector = Dense(32, activation='relu')(mlp_vector)
mlp_vector = Dropout(0.2)(mlp_vector)
mlp_vector = Dense(16, activation='relu')(mlp_vector)

# Concatenate GMF and MLP parts
neuMF_vector = Concatenate()([gmf_vector, mlp_vector])

# Output layer
output = Dense(1, activation='linear', name='output')(neuMF_vector)

# Define the model
model = Model(inputs=[user_input, movie_input], outputs=output)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error

model.summary()
```

Model: "functional_16"

Layer (type)	Output Shape	Param #	Connected to
user_input (InputLayer)	(None, 1)	0	-
movie_input (InputLayer)	(None, 1)	0	-
user_embedding_mlp (Embedding)	(None, 1, 16)	9,760	user_input[0]
movie_embedding_mlp (Embedding)	(None, 1, 16)	155,584	movie_input[0
flatten_70 (Flatten)	(None, 16)	0	user_embeddin
flatten_71 (Flatten)	(None, 16)	0	movie_embeddi
concatenate_32 (Concatenate)	(None, 32)	0	flatten_70[0] flatten_71[0]
dense_54 (Dense)	(None, 64)	2,112	concatenate_3
dropout_20 (Dropout)	(None, 64)	0	dense_54[0][0
user_embedding_gmf (Embedding)	(None, 1, 16)	9,760	user_input[0]

movie_embedding_gmf (Embedding)	(None, 1, 16)	155,584	movie_input[0
dense_55 (Dense)	(None, 32)	2,080	dropout_20[0]
flatten_68 (Flatten)	(None, 16)	0	user_embeddin
flatten_69 (Flatten)	(None, 16)	0	movie_embeddi
dropout_21 (Dropout)	(None, 32)	0	dense_55[0][0
multiply_16 (Multiply)	(None, 16)	0	flatten_68[0] flatten_69[0]
dense_56 (Dense)	(None, 16)	528	dropout_21[0]
concatenate_33 (Concatenate)	(None, 32)	0	multiply_16[0 dense_56[0][0
output (Dense)	(None, 1)	33	concatenate_3

Total params: 335,441 (1.28 MB)

Trainable params: 335,441 (1.28 MB)

Non-trainable params: 0 (0.00 B)

Preparing Training, Validation and Test Data

In [317…

```
# Training data
x_train = [train_data['user'].values, train_data['movie'].values]
y_train = train_data['rating_norm'].values

# Validation data
x_val = [validation_data['user'].values, validation_data['movie'].values]
y_val = validation_data['rating_norm'].values

# Test data
x_test = [test_data['user'].values, test_data['movie'].values]
y_test = test_data['rating_norm'].values
```

Training the Model

In [318…

```
early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best

history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=256,
    epochs=20,
    verbose=1,
    validation_data=(x_val, y_val),
    callbacks=[early_stopping]
)
```

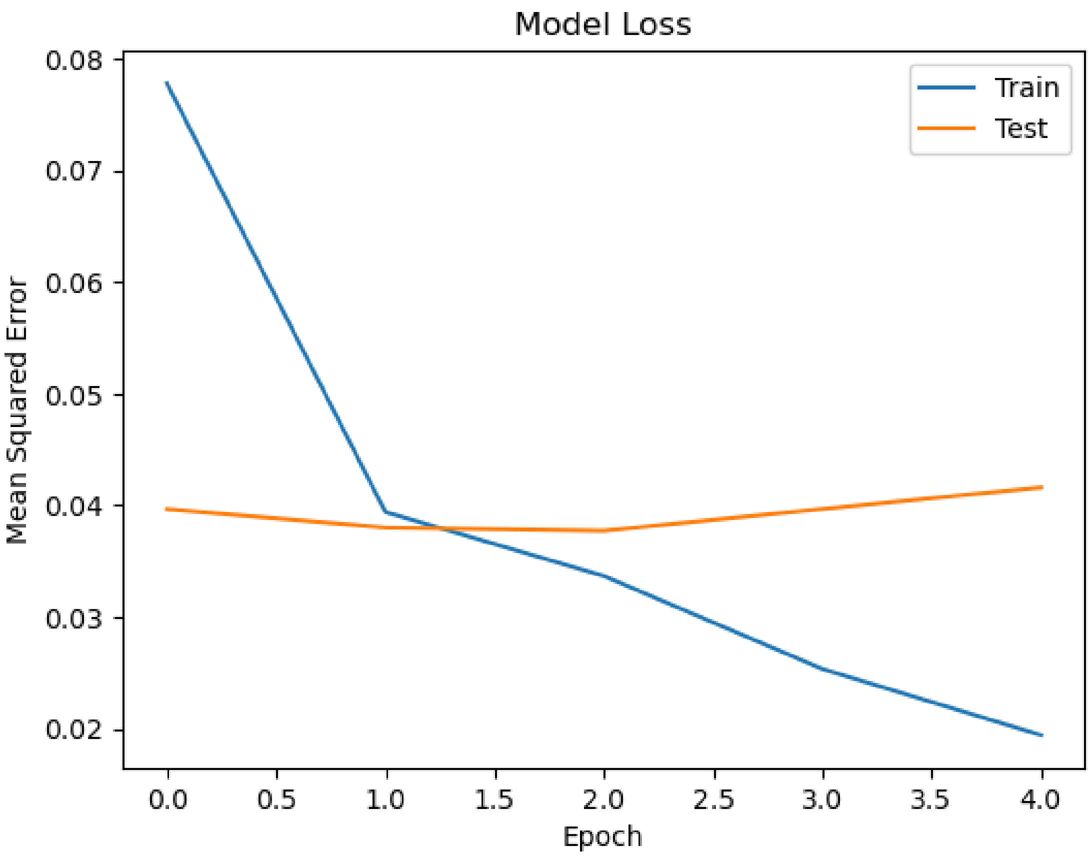
```
Epoch 1/20
276/276 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.1531 - val_loss: 0.0397
Epoch 2/20
276/276 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0394 - val_loss: 0.0380
Epoch 3/20
276/276 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0336 - val_loss: 0.0377
Epoch 4/20
276/276 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0256 - val_loss: 0.0397
Epoch 5/20
276/276 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0188 - val_loss: 0.0416
```

Evaluating the Model

Let's visualize the training history to see how the model's loss changes over epochs.

In [319…

```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Mean Squared Error')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```



Calculating RMSE and MAE on the test set

In [320…

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Predict on test data
y_pred = model.predict(x_test)

# Denormalize the predictions
```

```
y_pred_denorm = y_pred * (max_rating - min_rating) + min_rating
y_test_denorm = y_test * (max_rating - min_rating) + min_rating

# Calculate RMSE and MAE
rmse = np.sqrt(mean_squared_error(y_test_denorm, y_pred_denorm))
mae = mean_absolute_error(y_test_denorm, y_pred_denorm)

print(f'RMSE on test data: {rmse:.4f}')
print(f'MAE on test data: {mae:.4f}')
```

```
473/473 ━━━━━━━━━━━━━━━━━━━━ 0s 507us/step
RMSE on test data: 0.8743
MAE on test data: 0.6760
```

The Neural Collaborative Filtering (NCF) model has achieved an RMSE of 0.8719, higher than the previous hybrid SVD and KNN model's RMSE of 0.8518.

Our hybrid SVD and KNN model performs better than the NCF model in this case.

This could be as a result of limited data: 610 users and 9,724 movies, the dataset might be insufficient for the NCF model to learn complex patterns effectively.

Generating Recommendations

In [321…

```
def recommend_movies(user_id, model, ratings_df, movies_df, top_k=10):
    # Check if user_id exists in the data
    if user_id not in user2user_encoded:
        print("User ID not found.")
        return None

    # Get the user's encoded ID
    user_encoded_id = user2user_encoded[user_id]

    # Get all movie IDs
    movie_df = movies_df.copy()
    movie_df['movie'] = movie_df['movieId'].map(movie2movie_encoded)
    movie_df = movie_df.dropna(subset=['movie'])
    movie_df['movie'] = movie_df['movie'].astype(int)

    # Movies that the user has already rated
    user_rated_movies = ratings_df[ratings_df['userId'] == user_id]['movieI
    user_rated_movies_encoded = [movie2movie_encoded.get(x) for x in user_r

    # Filter out movies already rated by the user
    movies_to_predict = movie_df[~movie_df['movie'].isin(user_rated_movies_

    # Prepare input data
    user_movie_array = np.hstack(
        (
            np.array([user_encoded_id] * len(movies_to_predict)).reshape(-1
            movies_to_predict['movie'].values.reshape(-1, 1)
        )
    )

    # Predict ratings
    predictions = model.predict([user_movie_array[:, 0], user_movie_array[:

    # Denormalize predictions
    predictions_denorm = predictions * (max_rating - min_rating) + min_rati

    # Add predictions to the DataFrame
    movies_to_predict['predicted_rating'] = predictions_denorm
```

```
    movies_to_predict['predicted_rating'] = predictions_denorm

    # Get the top K movies
    recommended_movies = movies_to_predict.sort_values('predicted_rating',

    # Map encoded IDs back to original IDs
    recommended_movies['movieId'] = recommended_movies['movieId']

    # Select relevant columns
    recommended_movies = recommended_movies[['movieId', 'title', 'genres',

    return recommended_movies
```

Example Usage

In [322…

```
# Example: Get recommendations for user ID 1
user_id = 1
recommended_movies = recommend_movies(user_id, model, ratings_df, movies_df

print(f"Top 10 movie recommendations for User {user_id}:\n")
print(recommended_movies)
```

```
297/297 ━━━━━━━━━━━━━━━━━━━━ 0s 485us/step
Top 10 movie recommendations for User 1:

      movieId                                             title  \
841      1104                   Streetcar Named Desire, A (1951)
1268     1683                       Wings of the Dove, The (1997)
4396     6460                     Trial, The (Procès, Le) (1962)
883      1178                            Paths of Glory (1957)
1649     2202                                  Lifeboat (1944)
687       905                       It Happened One Night (1934)
1762     2360                  Celebration, The (Festen) (1998)
924      1223  Grand Day Out with Wallace and Gromit, A (1989)
906      1204                          Lawrence of Arabia (1962)
947      1248                               Touch of Evil (1958)

                                        genres  predicted_rating
841                                    [Drama]          4.901228
1268                          [Drama, Romance]          4.886067
4396                                   [Drama]          4.863326
883                               [Drama, War]          4.837581
1649                              [Drama, War]          4.826804
687                          [Comedy, Romance]          4.820685
1762                                   [Drama]          4.815631
924   [Adventure, Animation, Children, Comedy, Sci-Fi]  4.811740
906                    [Adventure, Drama, War]          4.805047
947                   [Crime, Film-Noir, Thriller]      4.803361
```

Streamlining the Genres and Tags using NLP

Extracting Valid Genres

In [323…

```
unique_genres = set()
for genres_list in movies_df['genres']:
    unique_genres.update(genres_list)

# Convert the set to a sorted list
unique_genres = sorted(unique_genres)
```

```
unique_genres = sorted(unique_genres)

print("Available Genres:")
print(unique_genres)
```

```
Available Genres:
['(no genres listed)', 'Action', 'Adventure', 'Animation', 'Children', 'Comed
y', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'IMA
X', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']
```

In [324…

```
len(unique_genres)
```

Out[324…

```
20
```

Extracting valid tags

In [325…

```
# Filling NaN values in 'tag' column with empty strings
tags_df['tag'] = tags_df['tag'].fillna('')

# Converting tags to lowercase and strip whitespace
tags_df['tag'] = tags_df['tag'].str.lower().str.strip()

# Removing empty strings
tags_df = tags_df[tags_df['tag'] != '']

# Getting unique tags
unique_tags = tags_df['tag'].unique()

# Converting to a list and sort
unique_tags = sorted(unique_tags)

print("Available Tags:")
print(unique_tags)
```

```
Available Tags:
['"artsy"', '06 oscar nominated best movie - animation', '1900s', '1920s', '1
950s', '1960s', '1970s', '1980s', '1990s', '2001-like', '2d animation', '70m
m', "80's", 'a clever chef rat', 'a dingo ate my baby', 'aardman', 'abortio
n', 'absorbing', 'abstract', 'academy award (best supporting actress)', 'acci
dent', 'achronological', 'acting', 'action', 'action choreography', 'action p
acked', 'adam sandler', 'addiction', 'adolescence', 'adoption', 'adorable',
'adrien brody', 'adult humor', 'adultery', 'adventure', 'afghanistan', 'afric
a', 'agatha christie', 'aggressive', 'aging', 'aids', 'al pacino', 'alan rick
man', 'alcatraz', 'alcoholism', 'alfred hitchcock', 'alicia vikander', 'alien
s', 'allegorical', 'alone in the world', 'alter ego', 'alternate endings', 'a
lternate reality', 'alternate universe', 'amazing', 'amazing artwork', 'amazi
ng cinematography', 'amazing dialogues', 'american idolatry', 'american india
ns', 'american propaganda', 'amish', 'amnesia', 'amtrak', 'amy adams', 'ancie
nt rome', 'andrew lloyd weber', 'android(s)/cyborg(s)', 'androids', 'andy gar
cia', 'andy kaufman', 'andy samberg', 'angelina jolie', 'anger', 'animal movi
e', 'animation', 'anime', 'anne boleyn', 'anne hathaway', 'annoying', 'anthol
ogy', 'anthony hopkins', 'anti-intellectual', 'anti-semitism', 'anti-war', 'a
pes', 'apocalypse', 'aquarium', 'archaeology', 'ark of the covenant', 'arnold
schwarzenegger', 'art', 'art house', 'arthouse', 'arthur c. clarke', 'arthur
miller', 'artificial intelligence', 'artistic', 'artsy', 'as byatt', 'assassi
n', 'assassin-in-training (scene)', 'assassination', 'assassins', 'astaire an
d rogers', 'asylum', 'atmospheric', 'atomic bomb', 'audience intelligence und
erestimated', 'audrey tautou', 'austere', 'australia', 'autism', 'avant-garde
romantic comedy', 'aviation', 'awesome', 'awkward', 'awkward romance', 'babie
s', 'backwards. memory', 'bad', 'bad acting', 'bad ass', 'bad dialogue', 'bad
humor', 'bad jokes', 'bad language', 'bad music', 'bad plot', 'bad science',
'bad script', 'bad story', 'bad writing', 'bad-ass', 'ballet', 'ballroom danc
e', 'bank', 'baseball', 'based on a book', 'based on a play', 'based on a tru
```

```
e', 'bank', 'baseball', 'based on a book', 'based on a play', 'based on a tru
e story', 'based on a tv show', 'basketball', 'batman', 'bears', 'beat poetr
y', 'beatles', 'beautiful', 'beautiful cinematography', 'beautiful scenery',
'beautiful visuals', 'beautifully filmed', 'bechdel test:fail', 'beethoven',
'ben affleck', 'ben kingsley', 'ben stiller', 'best comedy', 'best picture',
'bette davis', 'bible', 'biblical references', 'big boys with guns', 'big bro
thers', 'big budget', 'big corporations', 'big name actors', 'big top', 'big
wave', 'biking', 'bill murray', 'biography', 'biopic', 'birds', 'bitterswee
t', 'bizarre', 'bizzare', 'black and white', 'black comedy', 'black hole', 'b
lack humor', 'black humour', 'black-and-white', 'bleak', 'blind', 'blindnes
s', 'blood', 'blood splatters', 'bloody', 'bluegrass', 'boksdrama', 'bombs',
'books', 'borg', 'boring', 'boston', 'bowling', 'boxing', 'boxing story', 'br
ad pitt', 'brainwashing', 'brilliant', 'british', 'british comedy', 'british
gangster', 'britpop', 'brittany murphy', 'broadway', 'bromance', 'bromantic',
'brooch', 'brothers', 'bruce willis', 'brutal', 'brutality', 'bubba gump shri
mp', 'bugs bunny', 'building a family', 'bus', 'business', 'busniess', 'butle
r', 'c.s. lewis', 'california', 'cambodia', 'camels', 'cameo:whoopi goldber
g', 'camp', 'canada', 'cancer', 'capone', 'capote', 'captain america', 'capta
in kirk', 'captain nemo', 'carnival', 'cartoon', 'casey affleck', 'casino',
'casual violence', 'cate blanchett', 'cattle drive', 'celebrity fetishism',
'cerebral', 'cgi', 'challenging', 'character development', 'character study',
'characters', 'charles dickens', 'charlize theron', 'charlotte bronte', 'chea
ting', 'cheeky', 'cheesy', 'chess', 'chick flick', 'child abuse', 'childish n
aivity', 'children', 'chile', 'chilly', 'china', 'chris evans', 'chris klei
n', 'christian bale', 'christina ricci', 'christmas', 'christoph waltz', 'chr
istopher lloyd', 'christopher nolan', 'chuck palahniuk', 'cia', 'cinematograp
hy', 'circus', 'city politics', 'civil war', 'claims to be true', 'class', 'c
lassic', 'classic movie', 'classic sci-fi', 'claustrophobic', 'claymation',
'clever', 'cliche characters', 'clock', 'clousseau', 'coen bros', 'coen broth
ers', 'coke', 'cold', 'cold war', 'cole porter', 'colin farrell', 'college',
'colorful', 'colourful', 'coma', 'comedy', 'comic book', 'comics', 'complicat
ed', 'computer', 'computer animation', 'computers', 'con artists', 'con men',
'conan', 'confrontational', 'confusing', 'confusing ending', 'conspiracy', 'c
onspiracy theory', 'consumerism', 'contemplative', 'controversial', 'conven
t', 'conversation', 'cool', 'cool style', 'corruption', 'costume drama', 'cou
lda been a contender', 'court', 'courtroom drama', 'crappy sequel', 'crazy',
'creative', 'creativity', 'creature feature', 'creepy', 'crime', 'crime scene
scrubbing', 'cross dressing', 'crucifixion', 'crude humor', 'cruel character
s', 'cryptic', 'cult', 'cult classic', 'cult film', 'cyberpunk', 'cynical',
'dan aykroyd', 'dance', 'dance marathon', 'dancing', 'daniel craig', 'daniel
radcliffe', 'dark', 'dark comedy', 'dark fairy tale', 'dark hero', 'dark humo
r', 'darth vader', 'dating', 'david bowie', 'david fincher', 'david thewlis',
'day and hudson', 'dc', 'dc comics', 'dead wife', 'deadpan', 'deaf', 'deafnes
s', 'death', 'death penalty', 'deep throat', 'demons', 'denzel washington',
'depressing', 'depression', 'diabetes', 'dialogue', 'dickens', 'different',
'diner', 'dinosaur', 'dinosaurs', 'directorial debut', 'disability', 'disappo
inting', 'disaster', 'disjointed timeline', 'disney', 'disney animated featur
e', 'disturbing', 'divorce', 'doc ock', 'doctors', 'documentary', 'dodie smit
h', 'dogs', 'doll', 'domestic violence', 'donkey', 'dorothy', 'double life',
'downbeat', 'dr. seuss', 'dr. strange', 'drama', 'dreamlike', 'dreams', 'drea
my', 'drug abuse', 'drug overdose', 'drugs', 'drugs & music', 'dull', 'duma
s', 'dumb', 'dumpster diving', 'dust bowl', 'dwayne johnson', 'dystopia', 'e-
mail', 'e. m. forster', 'e.m. forster', 'earnest', 'easygoing', 'eccentric',
'economics', 'ed harris', 'edith wharton', 'edward norton', 'eerie', 'einstei
n', 'elegant', 'elegiac', 'embarassing scenes', 'emilia clarke', 'emma', 'emm
a stone', 'emma thompson', 'emotional', 'empire state building', 'end of the
world', 'ending', 'england', 'engrossing adventure', 'enigmatic', 'enjoyabl
e', 'ensemble cast', 'enterprise', 'entertaining', 'entirely dialogue', 'epi
c', 'episodic', 'eric bana', 'espionage', 'ethics', "eugene o'neill", 'europ
e', 'eva green', 'everything you want is here', 'evil children', 'evolution',
'ewan mcgregor', 'ex-con', 'exciting', 'existential', 'existentialism', 'exqu
isite plotting.', 'factory', 'faerie tale', 'fairy tale', 'fairy tales', 'fal
ling', 'families', 'family', 'fantasy', 'fantasy world', 'far fetched', 'fast
paced', 'fast-paced', 'fast-paced dialogue', 'fatalistic', 'father-son relati
onship', 'fatherhood', 'favelas', 'fbi', 'feel-good', 'ferris wheel', 'fighti
```

```
ng', 'fighting the system', 'figure skating', 'film history', 'film noir', 'f
ilm-noir', 'financial crisis', 'firefly', 'first was much better', 'fish', 'f
lood', 'food', 'football', 'for katie', 'foul language', 'france', 'franchis
e', 'francis ford coppola', 'freaks', 'free speech', 'free to download', 'fre
edom', 'freedom of expression', 'french', 'friendship', 'fucked up', 'fugitiv
e', 'fun', 'fun family movie', 'funny', 'future', 'futuristic', 'gal gadot',
'gambling', 'game', 'gangs', 'gangster', 'gangsters', 'gary oldman', 'geeky',
'general motors', 'generation x', 'genius', 'genocide', 'gentle', 'george ber
nard shaw', 'george clooney', 'george lucas', 'ghost', 'ghosts', 'gintama',
'girl power', 'give me back my son!', 'gold', 'golden watch', 'golf', 'golfin
g', 'good', 'good and evil', 'good cinematography', 'good dialogue', 'good mu
sic', 'good soundtrack', 'good writing', 'goofy', 'gore', 'goretastic', 'goth
ic', 'governess', 'grace', 'graham greene', 'graphic design', 'great acting',
'great cinematography', 'great dialogue', 'great ending', 'great humor', 'gre
at movie', 'great performances', 'great screenplay', 'great soundtrack', 'gre
at villain', 'great visuals', 'grim', 'gritty', 'gruesome', 'guardians of the
galaxy', 'gulf war', 'gun fu', 'gun tactics', 'gun-fu', 'gunfight', 'guns',
'hal', 'halle berry', 'halloween', 'hallucinatory', 'hammett', 'hannibal lect
er', 'hannibal lector', 'happpiness', 'happy ending', 'harley quinn', "harley
quinn's ass", 'harper lee', 'harrison ford', 'harry potter', 'harsh', 'harvey
keitel', 'haunting', 'hawkeye', 'hayao miyazaki', 'hearst', 'heartbreaking',
'heartwarming', 'heavy metal', 'heist', 'helena bonham carter', 'hemingway',
'henry darger', 'henry james', 'hepburn and tracy', 'heroic bloodshed', 'hero
in', 'heroine in tight suit', 'high fantasy', 'high school', 'highly quotabl
e', 'highschool', 'hilarious', 'hilary swank', 'hip hop', 'hippies', 'histori
cal', 'history', 'hit men', 'hitman', 'hollywood', 'holocaust', 'holy grail',
'homeless', 'homosexuality', 'hope', 'horrible acting', 'horrible directing',
'horrid characterisation', 'horror', 'horses', 'hostage', 'hot actress', 'hot
el', 'housekeeper', 'howard hughes', 'huey long', 'hugh jackman', 'hula hoo
p', 'human rights', 'humane', 'humor', 'humorous', 'humour', 'hungary', 'i am
your father', 'i see dead people', 'ichabod crane', 'iconic', 'illusions', 'i
maginary friend', 'imagination', 'imaginative', 'imdb top 250', 'immigrants',
'immigration', 'immortality', 'in netflix queue', 'in your eyes', 'incest',
'independent', 'independent film', 'india', 'indiana jones', 'indie record la
bel', 'individualism', 'indonesia', 'infertility', 'inhumane', 'inigo montoy
a', 'innovative', 'insane', 'insanity', 'insightful', 'insomnia', 'inspiratio
nal', 'inspiring', 'insurance', 'intellectual', 'intelligent', 'intelligent s
ci-fi', 'intense', 'interesting', 'interesting characters', 'interesting scen
ario', 'interracial marriage', 'interracial romance', 'intertwining storyline
s', 'interwoven storylines', 'intimate', 'introspection', 'investor corruptio
n', 'invisibility', 'ireland', 'ironic', 'irony', 'irreverent', 'island', 'is
tanbul', 'it was melodramatic and kind of dumb', 'italy', 'jack nicholson',
'jackie chan', 'jaime pressly', 'jake gyllenhaal', 'james cameron', 'james fe
nnimore cooper', 'james franco', 'james stewart', 'jane austen', 'japan', 'ja
red leto', 'jason', 'jason biggs', 'jason segel', 'jay and silent bob', 'jaz
z', 'jean grey', 'jean reno', 'jeff bridges', 'jekyll and hyde', 'jennifer co
nnelly', 'jennifer lawrence', 'jesse eisenberg', 'jesse ventura', 'jessica al
ba', 'jim carrey', 'jim morrison', 'john cusack', 'john goodman', 'john grish
am', 'john malkovich', 'john travolta', 'johnny cash', 'johnny depp', 'joke
r', 'jon hamm', 'josh brolin', 'joss whedon', 'journalism', 'judaism', 'jude
law', 'julianne moore', 'juliette lewis', 'jungle', 'justice', 'justin timber
lake', 'katzanzakis', 'keanu reeves', 'kevin costner', 'kevin smith', 'kevin
spacey', 'kidnapping', 'kids', 'killer', 'killer-as-protagonist', 'king arthu
r', 'klingons', 'knights', 'kung fu', 'kurt russell', 'l.a.', 'lack of develo
pment', 'lack of plot', 'lack of story', 'large cast', 'las vegas', 'last man
on earth', 'lawn mower', 'lawyer', 'lawyers', 'leonardo dicaprio', 'leonardo
dicarpio', 'leopard', 'lesbian', 'lesbian subtext', 'liam neeson', 'lies', 'l
ieutenant dan', 'lion', 'lions', 'live action/animation', 'lloyd dobbler', 'l
olita theme', 'loneliness', 'lonesome polecat', 'long shots', 'long takes',
'longing', 'lord of the rings', 'loretta lynn', 'los angeles', 'lou gehrig',
'louisa may alcott', 'love', 'love story', 'lovely', 'luc besson', 'luke skyw
alker', 'lyrical', 'm. night shyamalan', 'macaulay culkin', 'macbeth', 'mach
o', 'mad scientist', 'made me cry', 'mafia', 'maggie gyllenhaal', 'magic', 'm
agic board game', 'magneto', 'male nudity', 'margot robbie', 'marijuana', 'ma
```

```
rion cotillard', 'mark ruffalo', 'mark wahlberg', 'marriage', 'martial arts',
'martin scorsese', 'marvel', 'marx brothers', 'masculinity', 'masterpiece',
'matchmaker', 'mathematics', 'matrix', 'matt damon', 'may-december romance',
'mccarthy hearings', 'mcdonalds', 'mcu', 'meaningless violence', 'mecha', 'me
diacentralism', 'medieval', 'meditative', 'mel gibson', 'melancholic', 'melan
choly', 'memory', 'memory loss', 'men in drag', 'menacing', 'mental hospita
l', 'mental illness', 'mermaid', 'meryl streep', 'metaphorical', 'mexico', 'm
ice', 'michael bay', 'michael cera', 'michael crichton', 'michigan', 'middle
east', 'mila kunis', 'military', 'milkshake', 'mind-bending', 'mind-blowing',
'mindfuck', 'mindless one liners', 'mining', 'mirrors', 'missing children',
'missionary', 'mma', 'mobster', 'mobsters', 'mockumentary', 'modern fantasy',
'modern war', 'moldy', 'money', 'monologue', 'monty python', 'moody', 'moon',
'morality', 'morgan freeman', 'morrow', 'moses', 'motherfucker', 'motherhoo
d', 'motivational', 'mount rushmore', 'mountain climbing', 'movie business',
'movies', 'movies about movies', 'moving', 'mozart', 'mrs. dewinter', 'multip
le personalities', 'multiple roles', 'multiple short stories', 'multiple stor
ies', 'multiple storylines', 'muppets', 'murder', 'music', 'music business',
'music industry', 'mystery', 'myth', 'mythology', 'nabokov', 'nanny', 'narni
a', 'narrated', 'narrative pisstake', 'nasa', 'natalie portman', 'nathan fill
ion', 'native americans', 'navy', 'nazis', 'needed more autobots', 'neil patr
ick harris', 'neo-noir', 'nerd', 'nerds', 'new composer', 'new society', 'new
york', 'new york city', 'nick and nora charles', 'nick hornby', 'nicolas cag
e', 'nightclub', 'nightmare', 'ninotchka remake', 'no dialogue', 'no dvd at n
etflix', 'nocturnal', 'noir', 'non-linear', 'non-linear timeline', 'nonlinea
r', 'nonlinear narrative', 'nonlinear storyline', 'nonlinear timeline', 'nons
ense', 'norman bates', 'nostalgia', 'not available from netflix', 'not funn
y', 'not linear', 'not seen', 'notable nudity', 'notable soundtrack', 'nuclea
r disaster', 'nuclear war', 'nudity (full frontal)', 'nudity (topless)', 'nu
n', 'nuns', 'obsession', 'ocean', 'off-beat comedy', 'offensive', 'ogres', 'o
il', 'old', 'oldie but goodie', 'olympics', 'oninous', 'opera', 'organised cr
ime', 'organized crime', 'original', 'original plot', 'orlando bloom', 'orpha
ns', 'oscar (best actress)', 'oscar (best cinematography)', 'oscar (best effe
cts - visual effects)', 'oscar (best music - original score)', 'oscar (best s
upporting actress)', 'oscar wilde', 'othello', 'out of order', 'overcomplicat
ed', 'overrated', 'pageant', 'painter', 'palahnuik', "palme d'or", 'paranoi
a', 'paranoid', 'parenthood', 'paris', 'parody', 'parrots', 'passion', 'paul
giamatti', 'paul rudd', 'peace corp', 'pearl s buck', 'pee wee herman', 'pers
onals ads', 'peta wilson', 'peter pan', 'philip k. dick', 'philip seymour hof
fman', 'philosophical', 'philosophical issues', 'philosophy', 'philosopical',
'photographer', 'photography', 'pigs', 'pixar', 'pizza beer', 'planes', 'plas
tic surgery', 'plot holes', 'plot twist', 'poetic', 'poignant', 'police', 'po
lice corruption', 'political commentary', 'political right versus left', 'pol
itics', 'pool', 'poor dialogue', 'poor plot development', 'poor story', 'poor
ly paced', 'pop culture references', 'post apocalyptic', 'post-apocalyptic',
'post-college', 'postmodern', 'pow', 'powerful ending', 'preacher', 'predicta
ble', 'predictible plot', 'pregnancy', 'prejudice', 'prequel', 'president',
'priest', 'prince', 'prison', 'procedural', 'prodigies', 'prom', 'prostitutio
n', 'psychedelic', 'psychiatrist', 'psychological', 'psychological thriller',
'psychology', 'psychopaths', 'ptsd', 'pudding', 'pulp', 'purity of essence',
'purposefulness', 'quakers', 'queen victoria', 'quentin tarantino', 'quick cu
ts', 'quirky', 'quirky romantic', 'quotable', 'r language', 'r:disturbing vio
lent content including rape', 'r:disturbing violent images', 'r:graphic sexua
lity', 'r:some violence', 'r:strong bloody violence', 'r:strong language',
'r:sustained strong stylized violence', 'r:violence', 'rabbi', 'race', 'rache
l mcadams', 'rachel weisz', 'racism', 'radio', 'ralph fiennes', 'random', 'ra
nsom', 'rap', 'rape', 'rasicm', 'ray bradbury', 'real estate', 'realistic',
'reality tv', 'really bad', 'rebellion', 'recap', 'reciprocal spectator', 're
flective', 'relaxing', 'religion', 'remade', 'remake', 'remaster', 'remix cul
ture', 'renee zellweger', 'representation of children', 'restaurant', 'retr
o', 'reunion', 'revenge', 'revolution', 'revolutionary', 'rich guy - poor gir
l', 'ridiculous', 'rita hayworth can dance!', 'river', 'road trip', 'roald da
hl', 'rob zombie', 'robbery', 'robert de niro', 'robert downey jr.', 'robert
ludlum', 'robert penn warren', 'robin williams', 'robots', 'robots and androi
ds', 'roger avary', 'rogers and hammerstein', 'rogue', 'rolling stone', 'roma
nce', 'romans', 'romantic', 'romantic comedy', 'rome', 'rosebud', 'royal with
```

```
cheese', 'royalty', 'rug', 'russell crowe', 'russia', 'ryan reynolds', 's.e.
hinton', 'sad', 'saint', 'saints', 'salieri', 'salute to douglas sirk', 'samu
el l. jackson', 'samurai', 'sarcasm', 'satire', 'satirical', 'saturday night
live', 'savannah', 'scandal', 'scary', 'scenic', 'schizophrenia', 'school',
'sci-fi', 'science fiction', 'scifi', 'scifi cult', 'scifi masterpiece', 'sco
tland', 'scott turow', 'screwball', 'sean connery', 'seann william scott', 's
ecret society', 'secrets', 'seen at the cinema', 'seen more than once', 'self
discovery', 'sentimental', 'sequel', 'serial killer', 'seth macfarlane', 'set
h rogen', 'setting:space/space ship', 'sex', 'sexual humor', 'sexuality', 'se
xy', 'sexy female scientist', 'shakespeare', 'shakespeare sort of', 'shangri-
la', 'shark', 'shenanigans', 'shia labeouf', 'ships', 'shipwreck', 'short fil
ms', 'short stories', 'show business', 'shrimp', 'siam', 'silly', 'simon and
garfunkel', 'simon pegg', 'simple', 'sinbad', 'singers', 'singletons', 'siste
rhood', 'sisters', 'six-fingered man', 'skiing', 'slasher', 'slavery', 'slic
k', 'slim pickens', 'slow', 'slow action', 'slow paced', 'small time criminal
s', 'small towns', 'smart', 'smart writing', 'sniper', 'snl', 'soccer', 'soci
al commentary', 'societal criticism', 'sofia coppola', 'solitude', 'somber',
'something for everyone in this one... saw it without and plan on seeing it w
ith kids!', 'sophisticated', 'soundtrack', 'south africa', 'south america',
'south park', 'southern us', 'space', 'space action', 'space adventure', 'spa
ce craft', 'space epic', 'space opera', 'space station', 'space travel', 'spa
cecraft', 'spaghetti western', 'special effects', 'spelling bee', 'spiders',
'spies', 'splatter', 'spoof', 'sports', 'spying', 'stage', 'stand up', 'stand
-up comedy', 'stanley kubrick', 'stapler', 'star trek', 'star wars', 'start o
f a beautiful friendship', 'statue', 'stephen crane', 'stephen king', 'steve
buscemi', 'steve carell', 'steven spielberg', 'stiller', 'stock market', 'sto
ne age', 'stoner movie', 'stones of summer', 'stop looking at me swan', 'stop
using useless characters for filler', 'storytelling', 'stranded', 'strange',
'strangers on a train', 'studio ghibli', 'stupid', 'stupid but funny', 'stupi
d ending', 'stupid is as stupid does', 'stylish', 'stylized', 'submarine', 's
uburbia', 'subway', 'suicide', 'sundance award winner', 'superb soundtrack',
'superficial plot', 'superhero', 'superhero team', 'superman', 'surfing', 'su
rprise ending', 'surreal', 'surrealism', 'survival', 'suspense', 'suspensefu
l', 'sustainability', 'swashbuckler', 'sweet', 'sword fight', 'symbolic', 'sy
mbolism', 'system holism', 'tarantino', 'teacher', 'teachers', 'tear jerker',
'tearjerking', 'technology', 'tedious', 'teen', 'teen movie', 'teenage pregna
ncy', 'teenagers', 'televangelist', 'television', 'tennessee williams', 'tens
e', 'tension', 'tension building', 'terminal illness', 'terrorism', 'test ta
g', 'thanksgiving', 'thanos', 'the catholic church is the most corrupt organi
zation in history', 'the entertainer', 'the force', 'theater', 'they might be
giants', 'thor', 'thought provoking', 'thought-provoking', 'threesome', 'thri
ller', 'thrilling', 'tilda swinton', 'tim burton', 'time travel', 'time-trave
l', 'titanic', 'tobacco', 'toga', 'tokyo', 'tolkein', 'tolkien', 'tolstoy',
'tom clancy', 'tom hanks', 'tom hardy', 'too long', 'too many characters', 't
oo much love interest', 'toto', 'touching', 'tradition!', 'tragedy', 'tragi
c', 'train', 'trains', 'transplants', 'transvestite', 'travolta', 'treasure h
unt', 'trey parker', 'tricky', 'trippy', 'truckers', 'true story', 'truman ca
pote', 'truth', 'turkey', 'tv', 'twins', 'twist', 'twist ending', 'twisted',
'twists & turns', 'uma thurman', 'ummarti2006', 'uncomfortable', 'unconventio
nal', 'undercover cop', 'understated', 'unexplained', 'unintelligent', 'unio
n', 'unique', 'unlikely hero', 'unnecessary sequel', 'unoriginal', 'unpredict
able', 'unsettling', 'unusual', 'up series', 'updated classics', 'uplifting',
'vampire', 'vampires', 'van gogh', 'venice', 'vertriloquism', 'very funny',
'veterinarian', 'video', 'video game adaptation', 'video games', 'vietnam',
'viggo mortensen', 'villain nonexistent or not needed for good story', 'viole
nce', 'violence in america', 'violent', 'virginity', 'virtual reality', 'visu
al', 'visually appealing', 'visually striking', 'visually stunning', 'von bul
ow', 'voyeurism', 'vulgar', 'wall street', 'wapendrama', 'war', 'watergate',
'way too long', 'weak plot', 'weather forecaster', 'wedding', 'weddings', 'we
ird', 'well done', 'werewolf', 'wesley snipes', 'western', 'whales', 'whimsic
al', 'white guilt', 'widows/widowers', 'will ferrell', 'will smith', 'wine',
'winona ryder', 'wistful', 'witty', 'wizards', 'wolverine', 'women', 'wonderw
oman', 'woody harrelson', 'workplace', 'world war i', 'world war ii', 'writin
g', 'wrongful imprisonment', 'wry', 'younger men', 'zither', 'zoe kazan', 'zo
```

```
mbies', 'zooey deschanel']
```

In [326…

```
len(unique_tags)
```

Out[326…

```
1475
```

The available tags are numerous and sometimes overlapping or redundant, making it difficult for users to select from them.

We can use NLP to analyze the tags and group them into meaningful categories or clusters, resulting in a condensed list of tags that users can easily select from.

We will:

1. Preprocess the Tags: Clean and normalize the tags for consistent processing.
2. Represent Tags Using Word Embeddings: Convert tags into numerical vectors that capture semantic meaning.
3. Cluster Similar Tags: Group tags based on their semantic similarity.
4. Generate Condensed Tag List: Select representative tags or create labels for each cluster.
5. Present the Condensed List: Provide the condensed list to the user.

Preprocessing the Tags

- Convert to Lowercase: Ensure all tags are in lowercase.
- Remove Punctuation: Clean tags of any punctuation.
- Tokenization: Split multi-word tags into individual words.
- Lemmatization: Reduce words to their base form.
- Remove Stopwords: Eliminate common words that don't add meaning.

In [327…

```
# Downloading necessary NLTK data files
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')


# Initializing Lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

tags_list = unique_tags

# Preprocess tags
def preprocess_tag(tag):

    tag = tag.translate(str.maketrans('', '', string.punctuation))
    tag = tag.lower()
    tokens = word_tokenize(tag)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in
    preprocessed_tag = ' '.join(tokens)
    return preprocessed_tag

preprocessed_tags = [preprocess_tag(tag) for tag in tags_list]
```

```
[nltk_data] Downloading package punkt to /Users/user/nltk_data...
```

```
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/user/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/user/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Represent Tags Using Word Embeddings

- We will use pre-trained word embeddings from Gensim's word2vec-google-news-300 model to represent each tag as a vector.
- Handle Multi-word Tags: For tags with multiple words, we'll compute the average of the word vectors.

In [328…

```
# Loading pre-trained word2vec model (this may take a few minutes and requi
model = api.load('word2vec-google-news-300')  # 300-dimensional vectors

# Function to get vector for a tag
def get_tag_vector(tag):
    tokens = tag.split()
    vectors = []
    for token in tokens:
        if token in model:
            vectors.append(model[token])
    if vectors:
        # Compute average vector
        tag_vector = np.mean(vectors, axis=0)
        return tag_vector
    else:
        # If none of the words are in the model, return a zero vector
        return np.zeros(model.vector_size)

# Getting vectors for all tags
tag_vectors = []
valid_tags = []
for tag in preprocessed_tags:
    vector = get_tag_vector(tag)
    if np.any(vector):
        tag_vectors.append(vector)
        valid_tags.append(tag)
    else:
        # Skip tags that cannot be vectorized
        pass

# Converting list to numpy array
tag_vectors = np.array(tag_vectors)
```

In [329…

```
tag_vectors1 = pd.DataFrame(tag_vectors)
```

Clustering Similar Tags

- We'll use Agglomerative Hierarchical Clustering to cluster the tags based on their semantic similarity.

In [330…

```
# Clustering Similar Tags
from sklearn.cluster import AgglomerativeClustering
```

```
num_clusters = 20

# Performing clustering
clustering_model = AgglomerativeClustering(
    n_clusters=num_clusters,
    metric='euclidean',
    linkage='ward'
)
clustering_model.fit(tag_vectors)

# Getting cluster labels
labels = clustering_model.labels_

# Create a DataFrame to hold tags and their cluster labels
tag_cluster_df = pd.DataFrame({'tag': valid_tags, 'cluster': labels})

print(f"Number of clusters formed: {len(set(labels))}")
```

```
Number of clusters formed: 20
```

Generating Condensed Tag List

- We'll select a representative tag for each cluster by finding the tag closest to the cluster centroid.

In [331…

```
from sklearn.metrics.pairwise import cosine_distances

# Initializing list to store representative tags
condensed_tags = []

for cluster_num in range(num_clusters):
    cluster_tags = tag_cluster_df[tag_cluster_df['cluster'] == cluster_num]
    indices = cluster_tags.index
    cluster_vectors = tag_vectors[indices]

    # Computing the centroid of the cluster
    centroid = np.mean(cluster_vectors, axis=0)

    # Calculating distances from each tag vector to the centroid
    distances = cosine_distances(cluster_vectors, [centroid]).flatten()

    closest_index = distances.argmin()
    representative_tag = cluster_tags.iloc[closest_index]['tag']

    condensed_tags.append(representative_tag)
```

Presenting the Condensed List

- Displaying the Condensed Tag List

In [332…

```
# Sorting the condensed tags alphabetically
condensed_tags_sorted = sorted(condensed_tags)

print("Condensed Tag List:")
for tag in condensed_tags_sorted:
    print(tag)
```

```
Condensed Tag List:
badass
classic movie
```

```
cliche character
computer animation
dance
dinosaur
europe
heroin
interracial romance
melancholic
melodramatic kind dumb
mental illness
murder
neil patrick harris
offbeat comedy
philip seymour hoffman
priest
rstrong bloody violence
something everyone one saw without plan seeing kid
submarine
```

In [333…

```
len(condensed_tags_sorted)
```

Out[333…

```
20
```

In [334…

```
print("\nClusters and Their Tags:")
for cluster_num in range(num_clusters):
    cluster_tags = tag_cluster_df[tag_cluster_df['cluster'] == cluster_num]
    representative_tag = condensed_tags[cluster_num]
    print(f"Cluster {cluster_num}: Representative Tag - '{representative_ta
    print(f"Tags: {', '.join(cluster_tags)}\n")
```

```
Clusters and Their Tags:
Cluster 0: Representative Tag - 'cliche character'
Tags: abstract, alien, allegorical, amazing dialogue, atmospheric, bad dialog
ue, bad plot, bad story, based true story, biblical reference, brainwashing,
celebrity fetishism, cerebral, character development, character study, charac
ter, city politics, claim true, classic, cliche character, conspiracy, conspi
racy theory, consumerism, conversation, cruel character, cryptic, dark fairy
tale, demon, dialogue, disjointed timeline, dream, enigmatic, entirely dialog
ue, epic, episodic, ethic, evil child, existential, exquisite plotting, faeri
e tale, fairy tale, fairy tale, fantasy, fantasy world, fastpaced dialogue, f
atalistic, freedom, freedom expression, genius, ghost, ghost, good evil, good
dialogue, grace, great dialogue, guardian galaxy, high fantasy, historical, h
istory, iconic, illusion, immortality, individualism, insanity, intellectual,
interesting character, intertwining storyline, interwoven storyline, introspe
ction, invisibility, lack development, lack plot, lack story, loneliness, lon
ging, love, love story, macho, magic, magneto, marvel, masculinity, masterpie
ce, matrix, metaphorical, modern fantasy, morality, multiple short story, mul
tiple story, multiple storyline, mystery, myth, mythology, narrated, narrativ
e pisstake, new composer, nightmare, dialogue, nonlinear, nonlinear timeline,
nonlinear, nonlinear narrative, nonlinear storyline, nonlinear timeline, nost
algia, linear, obsession, ogre, original, original plot, paranoia, paranoid,
passion, philosophical, philosophical issue, philosophy, political commentar
y, politics, poor dialogue, poor plot development, poor story, post apocalypt
ic, predictible plot, prodigy, purity essence, purposefulness, ray bradbury,
recap, religion, revenge, secret, short story, social commentary, solitude, s
tart beautiful friendship, storytelling, stylized, superficial plot, symboli
c, symbolism, titanic, many character, much love interest, tradition, true st
ory, truth, unexplained, updated classic, weak plot, wizard

Cluster 1: Representative Tag - 'melodramatic kind dumb'
Tags: absorbing, acting, adorable, adventure, aggressive, amazing, amazing ar
```

```
twork, annoying, austere, awesome, awkward, awkward romance, bad, bad acting,
bad as, bad joke, bad language, bad science, bad script, bad writing, beautif
ul, beautiful scenery, beautiful visuals, beautifully filmed, bittersweet, bi
zarre, bizzare, bleak, boring, brilliant, challenging, cheeky, cheesy, childi
sh naivity, claustrophobic, clever, colorful, complicated, confrontational, c
onfusing, controversial, cool, cool style, crazy, creepy, cynical, depressin
g, different, disappointing, disturbing, downbeat, dull, dumb, easygoing, ecc
entric, elegant, engrossing adventure, enjoyable, entertaining, exciting, fre
ak, fun, futuristic, gentle, good, good writing, goofy, graphic design, great
acting, great performance, great visuals, grim, gritty, harsh, heartbreaking,
heartwarming, horrible acting, horrible directing, horrid characterisation, i
maginative, eye, innovative, insane, inspirational, inspiring, intelligent, i
ntelligent scifi, intense, interesting, interesting scenario, intimate, melod
ramatic kind dumb, lovely, menacing, mindblowing, mirror, motivational, nonse
nse, overcomplicated, overrated, poignant, predictable, quirky, quirky romant
ic, realistic, really bad, reflective, relaxing, retro, ridiculous, sad, scar
y, sentimental, silly, simple, slick, smart, smart writing, somber, sophistic
ated, strange, stupid, stupid funny, stupid stupid, stylish, surreal, sweet,
tedious, thrilling, tricky, twisted, uncomfortable, unconventional, understat
ed, unintelligent, unique, unoriginal, unpredictable, unsettling, unusual, up
lifting, visual, visually appealing, visually striking, visually stunning, vu
lgar, weird, whimsical, writing

Cluster 2: Representative Tag - 'dance'
Tags: bad music, ballet, ballroom dance, baseball, basketball, biking, bluegr
ass, bowling, boxing, boxing story, carnival, casino, chess, christoph waltz,
circus, dance, dance marathon, dancing, diner, football, gambling, golf, golf
ing, good music, hotel, hula hoop, indie record label, jazz, mma, music, musi
c business, music industry, nightclub, opera, pageant, prom, rap, remix cultu
re, restaurant, reunion, rita hayworth dance, singer, skiing, soccer, space o
pera, sport, surfing, entertainer, theater, wedding, wedding, zither

Cluster 3: Representative Tag - 'classic movie'
Tags: artsy, 06 oscar nominated best movie animation, animal movie, apocalyps
e, arthouse, artsy, bromantic, cgi, classic movie, classic scifi, crappy sequ
el, creature feature, cult, cult classic, cult film, cyberpunk, directorial d
ebut, disney animated feature, dystopia, existentialism, feelgood, film histo
ry, film noir, fun family movie, good soundtrack, gothic, great movie, great
screenplay, great soundtrack, horror, lolita theme, mindbending, mindfuck, mo
vie business, movie, movie movie, ninotchka remake, noir, notable soundtrack,
oscar best actress, oscar best cinematography, oscar best music original scor
e, oscar best supporting actress, oscar wilde, pixar, postapocalyptic, postmo
dern, prequel, remade, remake, remaster, scifi, scifi, scifi cult, scifi mast
erpiece, screwball, sequel, short film, soundtrack, stoner movie, studio ghib
li, suburbia, superb soundtrack, surrealism, tearjerking, unnecessary sequel

Cluster 4: Representative Tag - 'murder'
Tags: assassin, assassination, assassin, asylum, corruption, court, crime, cr
ime scene scrubbing, death, death penalty, espionage, fugitive, gang, gangste
r, gangster, heist, hitman, hostage, humane, immigrant, immigration, inhuman
e, investor corruption, justice, kidnapping, killer, mafia, mobster, mobster,
murder, organised crime, organized crime, police, police corruption, prison,
procedural, psychopath, ransom, rape, robbery, serial killer, spy, spying, su
icide, terrorism, undercover cop, wrongful imprisonment

Cluster 5: Representative Tag - 'priest'
Tags: dingo ate baby, baby, brother, butler, child abuse, child, coen brothe
r, convent, doctor, family, family, governess, hannibal lector, homeless, hou
sekeeper, father, kid, lawyer, lawyer, missing child, missionary, nanny, nun,
nun, orphan, preacher, priest, psychiatrist, representation child, saint, sai
nt, sister, sixfingered man, stranded, teen, teen movie, teenager, televangel
ist, trucker, veterinarian, woman

Cluster 6: Representative Tag - 'philip seymour hoffman'
Tags: agatha christie, al pacino, alfred hitchcock, alicia vikander, american
```

```
idolatry, american indian, american propaganda, ancient rome, ark covenant, a
rnold schwarzenegger, beatles, ben affleck, ben kingsley, bible, bill murray,
borg, british, british comedy, british gangster, broadway, brooch, cameowhoop
i goldberg, cate blanchett, charlotte bronte, china, christian bale, christma
s, cia, crucifixion, dan aykroyd, day hudson, dc, disney, doc ock, dorothy, e
mail, e forster, einstein, eugene oneill, eva green, fbi, hal, halle berry, h
alloween, harley quinn, harley quinns as, hawkeye, hollywood, netflix queue,
jaime pressly, jay silent bob, jean grey, jean reno, jeff bridge, jekyll hyd
e, jessica alba, johnny cash, johnny depp, king arthur, knight, kung fu, la,
la vega, leonardo dicaprio, leonardo dicarpio, lieutenant dan, los angeles, l
uc besson, matt damon, mcu, medieval, mila kunis, morrow, moses, nasa, natali
e portman, native american, new york, new york city, nick nora charles, dvd n
etflix, norman bates, available netflix, olympics, orlando bloom, palme dor,
paris, pee wee herman, philip k dick, philip seymour hoffman, prince, queen v
ictoria, rabbi, renee zellweger, roald dahl, robin williams, roman, rome, ros
ebud, royal cheese, rug, se hinton, salute douglas sirk, scotland, seth macfa
rlane, seth rogen, shia labeouf, snl, sofia coppola, stephen king, thanksgivi
ng, thor, toto, uma thurman, venice, wesley snipe, winona ryder, zoe kazan

Cluster 7: Representative Tag - 'rstrong bloody violence'
Tags: anger, antisemitism, bloody, brutal, brutality, casual violence, civil
war, domestic violence, genocide, gruesome, gulf war, heroic bloodshed, holoc
aust, meaningless violence, modern war, nuclear disaster, nuclear war, prejud
ice, rdisturbing violent content including rape, rdisturbing violent image, r
some violence, rstrong bloody violence, rsustained strong stylized violence,
racism, rebellion, revolution, revolutionary, slavery, star war, tense, tensi
on, tension building, violence, violent, war, world war, world war ii

Cluster 8: Representative Tag - 'submarine'
Tags: atomic bomb, aviation, bomb, bus, favelas, gun fu, gun tactic, gunfigh
t, gun, island, jungle, keanu reef, military, navy, ocean, plane, river, sava
nnah, scenic, settingspacespace ship, ship, shipwreck, sniper, stapler, stran
ger train, submarine, subway, train, train, van gogh

Cluster 9: Representative Tag - 'offbeat comedy'
Tags: adult humor, anthology, bad humor, best comedy, biography, biopic, blac
k comedy, black humor, book, comedy, comic book, comic, costume drama, courtr
oom drama, crude humor, dark comedy, dark humor, dc comic, deadpan, documenta
ry, drama, funny, great humor, highly quotable, hilarious, humor, humorous, i
nsightful, ironic, irony, irreverent, mockumentary, monologue, funny, offbeat
comedy, parody, quotable, sarcasm, satire, satirical, sexual humor, spoof, st
andup comedy, suspense, suspenseful, thriller, funny, witty, wry

Cluster 10: Representative Tag - 'mental illness'
Tags: accident, addiction, alcoholism, amnesia, assassinintraining scene, aut
ism, backwards memory, blind, blindness, cancer, coma, deaf, deafness, depres
sion, diabetes, disaster, embarassing scene, emotional, flood, insomnia, memo
ry, memory loss, mental hospital, mental illness, psychological, psychologica
l thriller, schizophrenia, terminal illness, tragedy, tragic, transplant

Cluster 11: Representative Tag - 'interracial romance'
Tags: abortion, adolescence, adultery, avantgarde romantic comedy, bromance,
cheating, chick flick, coen bros, dating, divorce, fatherson relationship, fa
therhood, friendship, gal gadot, homosexuality, hot actress, incest, infertil
ity, interracial marriage, interracial romance, lesbian, lesbian subtext, mal
e nudity, marriage, matchmaker, maydecember romance, motherhood, notable nudi
ty, nudity full frontal, nudity topless, oldie goodie, parenthood, pregnancy,
prostitution, rgraphic sexuality, romance, romantic, romantic comedy, scanda
l, sex, sexuality, sexy, shenanigan, singleton, sisterhood, teenage pregnanc
y, threesome, transvestite, virginity, voyeurism

Cluster 12: Representative Tag - 'heroin'
Tags: bubba gump shrimp, chile, coke, drug abuse, drug overdose, drug, drug m
usic, fish, food, heroin, marijuana, milkshake, pig, pizza beer, pudding, pul
```

```
p, shrimp, tobacco, turkey, wine

Cluster 13: Representative Tag - 'melancholic'
Tags: contemplative, dreamlike, dreamy, eerie, elegiac, hallucinatory, haunti
ng, lyrical, meditative, melancholic, melancholy, moody, poetic, psychedelic,
trippy, wistful

Cluster 14: Representative Tag - 'europe'
Tags: afghanistan, africa, australia, boston, california, canada, denzel wash
ington, england, europe, france, french, india, ireland, italy, japan, mexic
o, michigan, russia, vietnam

Cluster 15: Representative Tag - 'dinosaur'
Tags: ape, aquarium, bird, camel, cartoon, dinosaur, dinosaur, dog, doll, don
key, firefly, horse, ichabod crane, leopard, lion, lion, mermaid, nocturnal,
parrot, shark, spider, statue, stephen crane, whale, wolverine

Cluster 16: Representative Tag - 'neil patrick harris'
Tags: adam sandler, alan rickman, amy adam, andrew lloyd weber, andy garcia,
andy kaufman, andy samberg, angelina jolie, anne boleyn, anne hathaway, antho
ny hopkins, arthur c clarke, arthur miller, astaire rogers, bette davis, brad
pitt, brittany murphy, bruce willis, c lewis, casey affleck, chris evans, chr
is klein, christina ricci, christopher lloyd, christopher nolan, cole porter,
colin farrell, conan, daniel craig, daniel radcliffe, david bowie, david finc
her, david thewlis, dodie smith, dr seuss, dr strange, dwayne johnson, ed har
ris, edward norton, emilia clarke, emma, emma thompson, eric bana, katie, fra
ncis ford coppola, gary oldman, george bernard shaw, george clooney, george l
ucas, graham greene, harper lee, harrison ford, harry potter, harvey keitel,
helena bonham carter, henry darger, henry james, hepburn tracy, howard hughe
s, indiana jones, indonesia, jackie chan, jake gyllenhaal, james cameron, jam
es fennimore cooper, james franco, james stewart, jane austen, jared leto, ja
son, jason biggs, jason segel, jennifer connelly, jennifer lawrence, jesse ei
senberg, jesse ventura, jim carrey, jim morrison, john cusack, john goodman,
john grisham, john malkovich, john travolta, jon hamm, josh brolin, julianne
moore, juliette lewis, justin timberlake, kevin costner, kevin smith, kevin s
pacey, kurt russell, liam neeson, lloyd dobbler, loretta lynn, lou gehrig, lu
ke skywalker, maggie gyllenhaal, margot robbie, marion cotillard, martin scor
sese, mcdonalds, mel gibson, michael bay, michael cera, michael crichton, mor
gan freeman, mr dewinter, nathan fillion, neil patrick harris, paul giamatti,
paul rudd, peta wilson, rachel mcadams, rachel weisz, ralph fiennes, robert d
e niro, robert downey jr, robert ludlum, robert penn warren, roger avary, rog
ers hammerstein, russell crowe, ryan reynolds, samuel l jackson, scott turow,
sean connery, seann william scott, simon garfunkel, simon pegg, stanley kubri
ck, steve buscemi, steve carell, steven spielberg, tennessee williams, tim bu
rton, tokyo, tom clancy, tom hank, tom hardy, smith, woody harrelson

Cluster 17: Representative Tag - 'computer animation'
Tags: 2d animation, amazing cinematography, animation, art, art house, artist
ic, beat poetry, beautiful cinematography, cinematography, claymation, colleg
e, computer, computer animation, computer, creative, creativity, economics, f
oul language, good cinematography, great cinematography, high school, highsch
ool, imagination, journalism, martial art, mathematics, mouse, painter, photo
grapher, photography, psychology, r language, rstrong language, radio, realit
y tv, school, science fiction, spelling bee, teacher, teacher, television, tv

Cluster 18: Representative Tag - 'badass'
Tags: android, anime, antiwar, badass, batman, blood, blood splatter, charles
dickens, dickens, em forster, fucked, geeky, gore, hippy, joker, mecha, mothe
rfucker, muppets, nazi, nerd, nerd, pow, rob zombie, robot, robot android, sa
murai, slasher, splatter, superhero, superhero team, superman, swashbuckler,
toga, vampire, vampire, werewolf, zombie

Cluster 19: Representative Tag - 'something everyone one saw without plan see
ing kid'
Tags: clever chef rat, academy award best supporting actress, action, action
```

```
choreography, action packed, adoption, aging, aid, alone world, alter ego, al
ternate ending, alternate reality, alternate universe, amish, amtrak, artific
ial intelligence, audience intelligence underestimated, bank, based book, bas
ed play, based tv show, bear, ben stiller, best picture, big boy gun, big bro
ther, big budget, big corporation, big name actor, big top, big wave, black w
hite, black hole, black humour, bug bunny, building family, business, busnies
s, camp, captain america, captain kirk, captain nemo, cattle drive, chilly, c
huck palahniuk, class, clock, cold, cold war, con artist, con men, confusing
ending, coulda contender, cross dressing, dark, dark hero, darth vader, dead
wife, deep throat, disability, double life, duma, dumpster diving, dust bowl,
earnest, emma stone, empire state building, end world, ending, ensemble cast,
enterprise, everything want, evolution, ewan mcgregor, factory, falling, far
fetched, fast paced, fastpaced, ferris wheel, fighting, fighting system, figu
re skating, financial crisis, first much better, franchise, free speech, free
download, future, game, general motor, generation x, girl power, give back so
n, gold, golden watch, great ending, great villain, happy ending, heavy meta
l, heroine tight suit, hilary swank, hip hop, hit men, holy grail, hope, huey
long, hugh jackman, human right, hungary, see dead people, imaginary friend,
imdb top 250, independent, independent film, insurance, istanbul, jack nichol
son, joss whedon, judaism, jude law, large cast, last man earth, lawn mower,
lie, live actionanimation, lonesome polecat, long shot, long take, lord ring,
louisa may alcott, night shyamalan, mad scientist, made cry, magic board gam
e, mark ruffalo, mark wahlberg, marx brother, mccarthy hearing, men drag, mid
dle east, mindless one liner, mining, moldy, money, monty python, moon, mount
rushmore, mountain climbing, moving, multiple personality, multiple role, nee
ded autobots, new society, nick hornby, nicolas cage, seen, offensive, oil, o
ld, oscar best effect visual effect, order, peace corp, pearl buck, personal
ad, peter pan, plastic surgery, plot hole, plot twist, political right versus
left, pool, poorly paced, pop culture reference, postcollege, powerful endin
g, president, quaker, quick cut, race, random, real estate, reciprocal specta
tor, rich guy poor girl, road trip, rogue, rolling stone, royalty, saturday n
ight live, secret society, seen cinema, seen, self discovery, sexy female sci
entist, shakespeare, shakespeare sort, show business, siam, slim pickens, slo
w, slow action, slow paced, small time criminal, small town, societal critici
sm, something everyone one saw without plan seeing kid, south africa, south a
merica, south park, southern u, space, space action, space adventure, space c
raft, space epic, space station, space travel, spacecraft, spaghetti western,
special effect, stage, stand, star trek, stiller, stock market, stone age, st
one summer, stop looking swan, stop using useless character filler, stupid en
ding, sundance award winner, surprise ending, survival, sustainability, sword
fight, system holism, tear jerker, technology, test tag, catholic church corr
upt organization history, force, might giant, thought provoking, time travel,
long, touching, travolta, treasure hunt, trey parker, twin, twist, twist endi
ng, twist turn, union, unlikely hero, series, video, video game adaptation, v
ideo game, villain nonexistent needed good story, violence america, virtual r
eality, von bulow, wall street, watergate, way long, weather forecaster, well
done, western, white guilt, workplace, younger men
```

Using contextual embeddings like BERT can capture the semantic meaning of tags more effectively than traditional word embeddings. This can lead to more accurate clustering and a condensed tag list that makes sense.

Preprocess the Tags

We'll start by preprocessing the tags to ensure consistency.

In [335…

```python
from nltk.util import ngrams
```

In [336…

```python
# Initializing Lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
```

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

tags_list = unique_tags

# Preprocessing tags
def preprocess_tag(tag):

    tag = tag.translate(str.maketrans('', '', string.punctuation))
    tag = tag.lower()
    tokens = word_tokenize(tag)
    bigrams = ['_'.join(gram) for gram in ngrams(tokens, 2)]
    tokens.extend(bigrams)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in
    preprocessed_tag = ' '.join(tokens)
    return preprocessed_tag

preprocessed_tags = [preprocess_tag(tag) for tag in tags_list]
```

Generate BERT Embeddings for Tags

We'll use the pre-trained BERT model from Hugging Face's transformers library to generate embeddings.

Load BERT Model

We'll use the sentence-transformers library, which simplifies obtaining embeddings for sentences or phrases using BERT.

In [337…

```
from sentence_transformers import SentenceTransformer

# Loading the pre-trained BERT model (this may take some time)
model = SentenceTransformer('all-MiniLM-L6-v2')
```

Generate Embeddings

In [338…

```
# Generating embeddings for all preprocessed tags
tag_embeddings = model.encode(preprocessed_tags)
```

Cluster Similar Tags

- We'll use Agglomerative Hierarchical Clustering to cluster the tag embeddings.

In [339…

```
# Deciding on the number of clusters (adjust as needed)
num_clusters = 50

# Performing clustering
clustering_model = AgglomerativeClustering(
    n_clusters=num_clusters,
    metric='euclidean',
    linkage='ward'
)
clustering_model.fit(tag_embeddings)

# Getting cluster labels
labels = clustering_model.labels_
```

```
tag_cluster_df = pd.DataFrame({'tag': preprocessed_tags, 'cluster': labels}

print(f"Number of clusters formed: {len(set(labels))}")
```

```
Number of clusters formed: 50
```

Generate Condensed Tag List

- We'll select a representative tag for each cluster.

In [340…

```
# Initializing list to store representative tags
condensed_tags = []

for cluster_num in range(num_clusters):

    cluster_tags = tag_cluster_df[tag_cluster_df['cluster'] == cluster_num]
    indices = cluster_tags.index
    cluster_vectors = tag_embeddings[indices]

    centroid = np.mean(cluster_vectors, axis=0)

    distances = cosine_distances(cluster_vectors, [centroid]).flatten()

    closest_index = distances.argmin()
    representative_tag = cluster_tags.iloc[closest_index]['tag']

    condensed_tags.append(representative_tag)
```

Display the Condensed Tag List

In [341…

```
# Sorting the condensed tags alphabetically
condensed_tags_sorted = sorted(condensed_tags)

print("Condensed Tag List:")
for tag in condensed_tags_sorted:
    print(tag)
```

```
Condensed Tag List:
1970s
amazing
andy kaufman andy_kaufman
artistic
bad acting bad_acting
bible
big top big_top
bill murray bill_murray
crime
dark comedy dark_comedy
doctor
drug
emma thompson emma_thompson
entertaining
family
fast paced fast_paced
geeky
good dialogue good_dialogue
great cinematography great_cinematography
gunfight
history
```

```
horror
humorous
lack plot lack_of of_plot
murder
music
nonlinear timeline nonlinear_timeline
nuclear war nuclear_war
nudity topless nudity_topless
original
oscar best actress oscar_best best_actress
parenthood
philosophical
psychological
restaurant
robert downey jr robert_downey downey_jr
rolling stone rolling_stone
romance
russia
school
scifi cult scifi_cult
sentimental
space adventure space_adventure
spacecraft
storytelling
superhero
thriller
twist ending twist_ending
violence
whale
```

Display Clusters and Their Tags

In [342…

```python
print("\nClusters and Their Tags:")
for cluster_num in range(num_clusters):
    cluster_tags_list = tag_cluster_df[tag_cluster_df['cluster'] == cluster
    representative_tag = condensed_tags[cluster_num]
    print(f"Cluster {cluster_num}: Representative Tag - '{representative_ta
    print(f"Tags: {', '.join(cluster_tags_list)}\n")
```

```
Clusters and Their Tags:
Cluster 0: Representative Tag - 'philosophical'
Tags: absorbing, abstract, achronological, allegorical, american propaganda a
merican_propaganda, audience intelligence underestimated audience_intelligenc
e intelligence_underestimated, brainwashing, claymation, consumerism, contemp
lative, crucifixion, einstein, ethic, evil child evil_children, existential,
existentialism, fighting system fighting_the the_system, first much better fi
rst_was was_much much_better, free speech free_speech, freedom, freedom expre
ssion freedom_of of_expression, good evil good_and and_evil, hallucinatory, h
uman right human_right, individualism, irreverent, journalism, mathematics, m
ediacentralism, metaphorical, mindbending, mindblowing, mindfuck, mindless on
e liner mindless_one one_liners, mirror, morality, new society new_society, p
hilosophical, philosophical issue philosophical_issues, philosophy, philosopi
cal, political commentary political_commentary, political right versus left p
olitical_right right_versus versus_left, procedural, reciprocal spectator rec
iprocal_spectator, reflective, representation child representation_of of_chil
dren, social commentary social_commentary, societal criticism societal_critic
ism, sustainability, symbolic, symbolism, system holism system_holism, though
t provoking thought_provoking, thoughtprovoking, transvestite, vertriloquism

Cluster 1: Representative Tag - 'school'
Tags: adolescence, ballet, ballroom dance ballroom_dance, baseball, basketbal
l, biking, bowling, broadway, camp, carnival, casino, circus, college, dance,
```

```
dance marathon dance_marathon, dancing, football, gambling, golf, golfing, hi
gh school high_school, highschool, hollywood, hotel, nightclub, prom, school,
soccer, sport, stage, teen, teen movie teen_movie, teenager, theater

Cluster 2: Representative Tag - 'storytelling'
Tags: anthology, bad story bad_story, based true story based_on on_a a_true t
rue_story, character development character_development, character study chara
cter_study, character, claim true claims_to to_be be_true, cliche character c
liche_characters, costume drama costume_drama, courtroom drama courtroom_dram
a, cruel character cruel_characters, dark fairy tale dark_fairy fairy_tale, d
rama, faerie tale faerie_tale, fairy tale fairy_tale, fairy tale fairy_tale,
interesting character interesting_characters, intertwining storyline intertwi
ning_storylines, interwoven storyline interwoven_storylines, lack development
lack_of of_development, lack story lack_of of_story, love story love_story, m
acbeth, monologue, multiple short story multiple_short short_story, multiple
story multiple_stories, multiple storyline multiple_storylines, narrated, nar
rative pisstake narrative_pisstake, poor story poor_story, shakespeare, shake
speare sort shakespeare_sort sort_of, short story short_story, storytelling,
many character too_many many_characters, true story true_story, villain nonex
istent needed good story villain_nonexistent nonexistent_or or_not not_needed
needed_for for_good good_story

Cluster 3: Representative Tag - 'bill murray bill_murray'
Tags: adam sandler adam_sandler, adrien brody adrien_brody, alfred hitchcock
alfred_hitchcock, amy adam amy_adams, anthony hopkins anthony_hopkins, arnold
schwarzenegger arnold_schwarzenegger, ben affleck ben_affleck, ben kingsley b
en_kingsley, bill murray bill_murray, bruce willis bruce_willis, captain amer
ica captain_america, captain kirk captain_kirk, captain nemo captain_nemo, ca
sey affleck casey_affleck, chris evans chris_evans, chris klein chris_klein,
christoph waltz christoph_waltz, christopher nolan christopher_nolan, colin f
arrell colin_farrell, daniel craig daniel_craig, daniel radcliffe daniel_radc
liffe, darth vader darth_vader, denzel washington denzel_washington, ed harri
s ed_harris, ewan mcgregor ewan_mcgregor, george bernard shaw george_bernard
bernard_shaw, george clooney george_clooney, george lucas george_lucas, guard
ian galaxy guardians_of of_the the_galaxy, hannibal lecter hannibal_lecter, h
annibal lector hannibal_lector, harrison ford harrison_ford, harry potter har
ry_potter, helena bonham carter helena_bonham bonham_carter, hemingway, hugh
jackman hugh_jackman, jack nicholson jack_nicholson, jaime pressly jaime_pres
sly, james cameron james_cameron, james fennimore cooper james_fennimore fenn
imore_cooper, james franco james_franco, james stewart james_stewart, jared l
eto jared_leto, jason, jason biggs jason_biggs, jason segel jason_segel, jess
e eisenberg jesse_eisenberg, jesse ventura jesse_ventura, jim carrey jim_carr
ey, john cusack john_cusack, john malkovich john_malkovich, john travolta joh
n_travolta, johnny cash johnny_cash, johnny depp johnny_depp, justin timberla
ke justin_timberlake, keanu reef keanu_reeves, kevin costner kevin_costner, k
evin smith kevin_smith, kevin spacey kevin_spacey, kurt russell kurt_russell,
leonardo dicaprio leonardo_dicaprio, leonardo dicarpio leonardo_dicarpio, lia
m neeson liam_neeson, luke skywalker luke_skywalker, mark ruffalo mark_ruffal
o, mark wahlberg mark_wahlberg, martin scorsese martin_scorsese, matt damon m
att_damon, mel gibson mel_gibson, morgan freeman morgan_freeman, nabokov, nei
l patrick harris neil_patrick patrick_harris, nicolas cage nicolas_cage, norm
an bates norman_bates, oscar wilde oscar_wilde, peter pan peter_pan, quentin
tarantino quentin_tarantino, ralph fiennes ralph_fiennes, ray bradbury ray_br
adbury, roald dahl roald_dahl, robin williams robin_williams, russell crowe r
ussell_crowe, ryan reynolds ryan_reynolds, sean connery sean_connery, seann w
illiam scott seann_william william_scott, stanley kubrick stanley_kubrick, st
ar trek star_trek, star war star_wars, stephen crane stephen_crane, stephen k
ing stephen_king, steven spielberg steven_spielberg, sundance award winner su
ndance_award award_winner, tarantino, tim burton tim_burton, tolstoy, tom han
k tom_hanks, tom hardy tom_hardy, travolta, trey parker trey_parker

Cluster 4: Representative Tag - 'twist ending twist_ending'
Tags: alternate ending alternate_endings, confusing ending confusing_ending,
end world end_of of_the the_world, ending, great ending great_ending, happy e
nding happy_ending, last man earth last_man man_on on_earth, plot twist plot
```

```
twist, powerful ending powerful_ending, stupid ending stupid_ending, surprise
ending surprise_ending, twist, twist ending twist_ending, twisted, twist turn
twists_turns

Cluster 5: Representative Tag - 'robert downey jr robert_downey downey_jr'
Tags: aardman, agatha christie agatha_christie, al pacino al_pacino, alan ric
kman alan_rickman, alicia vikander alicia_vikander, andrew lloyd weber andrew
_lloyd lloyd_weber, anne boleyn anne_boleyn, anne hathaway anne_hathaway, art
hur c clarke arthur_c c_clarke, arthur miller arthur_miller, bette davis bett
e_davis, butler, c lewis cs_lewis, cate blanchett cate_blanchett, charles dic
kens charles_dickens, christian bale christian_bale, christina ricci christin
a_ricci, christopher lloyd christopher_lloyd, cole porter cole_porter, dan ay
kroyd dan_aykroyd, david bowie david_bowie, david fincher david_fincher, davi
d thewlis david_thewlis, dickens, dodie smith dodie_smith, dr seuss dr_seuss,
dr strange dr_strange, dust bowl dust_bowl, dwayne johnson dwayne_johnson, e
forster e_m m_forster, em forster em_forster, earnest, edith wharton edith_wh
arton, edward norton edward_norton, eric bana eric_bana, eugene oneill eugene
_oneill, francis ford coppola francis_ford ford_coppola, gary oldman gary_old
man, graham greene graham_greene, harper lee harper_lee, harvey keitel harvey
_keitel, henry darger henry_darger, henry james henry_james, howard hughes ho
ward_hughes, inigo montoya inigo_montoya, jane austen jane_austen, jean grey
jean_grey, jean reno jean_reno, jeff bridge jeff_bridges, jim morrison jim_mo
rrison, john goodman john_goodman, john grisham john_grisham, jon hamm jon_ha
mm, josh brolin josh_brolin, juliette lewis juliette_lewis, king arthur king_
arthur, lieutenant dan lieutenant_dan, lloyd dobbler lloyd_dobbler, loretta l
ynn loretta_lynn, lou gehrig lou_gehrig, louisa may alcott louisa_may may_alc
ott, luc besson luc_besson, marion cotillard marion_cotillard, maydecember ro
mance maydecember_romance, michael bay michael_bay, michael cera michael_cer
a, michael crichton michael_crichton, mila kunis mila_kunis, mr dewinter mrs_
dewinter, nathan fillion nathan_fillion, oldie goodie oldie_but but_goodie, p
aul giamatti paul_giamatti, paul rudd paul_rudd, peta wilson peta_wilson, phi
lip k dick philip_k k_dick, philip seymour hoffman philip_seymour seymour_hof
fman, robert de niro robert_de de_niro, robert downey jr robert_downey downey
_jr, robert ludlum robert_ludlum, robert penn warren robert_penn penn_warren,
roger avary roger_avary, scott turow scott_turow, seth macfarlane seth_macfar
lane, seth rogen seth_rogen, shia labeouf shia_labeouf, sixfingered man sixfi
ngered_man, sofia coppola sofia_coppola, steve buscemi steve_buscemi, steve c
arell steve_carell, tennessee williams tennessee_williams, uma thurman uma_th
urman, ummarti2006, van gogh van_gogh, viggo mortensen viggo_mortensen, von b
ulow von_bulow, wesley snipe wesley_snipes, smith will_smith, woody harrelson
woody_harrelson

Cluster 6: Representative Tag - 'psychological'
Tags: amnesia, autism, backwards memory backwards_memory, blind, blindness, c
erebral, claustrophobic, coma, deaf, deafness, disability, happpiness, insomn
ia, invisibility, meditative, memory, memory loss memory_loss, mental hospita
l mental_hospital, mental illness mental_illness, nocturnal, paranoia, parano
id, postapocalyptic, psychedelic, psychological, psychology, psychopath, rela
xing, schizophrenia, special effect special_effect, terminal illness terminal
_illness, uncomfortable, unsettling

Cluster 7: Representative Tag - 'rolling stone rolling_stone'
Tags: artificial intelligence artificial_intelligence, byatt as_byatt, bank,
bechdel testfail bechdel_testfail, black white black_and and_white, black hol
e black_hole, blackandwhite, bug bunny bugs_bunny, cattle drive cattle_drive,
charlize theron charlize_theron, cross dressing cross_dressing, dark, dark he
ro dark_hero, dumpster diving dumpster_diving, empire state building empire_s
tate state_building, ensemble cast ensemble_cast, everything want everything_
you you_want want_is is_here, falling, ferris wheel ferris_wheel, financial c
risis financial_crisis, free download free_to to_download, general motor gene
ral_motors, golden watch golden_watch, heavy metal heavy_metal, huey long hue
y_long, hula hoop hula_hoop, ichabod crane ichabod_crane, lawn mower lawn_mow
er, night shyamalan m_night night_shyamalan, macaulay culkin macaulay_culkin,
made cry made_me me_cry, matrix, mount rushmore mount_rushmore, mountain clim
```

```
bing mountain_climbing, multiple personality multiple_personality, multiple r
ole multiple_roles, needed autobots needed_more more_autobots, linear not_lin
ear, order out_of of_order, peace corp peace_corp, pearl buck pearl_s s_buck,
personal ad personals_ads, pizza beer pizza_beer, purity essence purity_of of
_essence, rasicm, real estate real_estate, rolling stone rolling_stone, salut
e douglas sirk salute_to to_douglas douglas_sirk, self discovery self_discove
ry, south park south_park, spelling bee spelling_bee, stand stand_up, stock m
arket stock_market, stone age stone_age, stoner movie stoner_movie, stone sum
mer stones_of of_summer, stop using useless character filler stop_using using
_useless useless_characters characters_for for_filler, tear jerker tear_jerke
r, tearjerking, tension, tension building tension_building, test tag test_ta
g, force the_force, treasure hunt treasure_hunt, series up_series, wall stree
t wall_street, weather forecaster weather_forecaster, white guilt white_guil
t, zooey deschanel zooey_deschanel

Cluster 8: Representative Tag - 'family'
Tags: american indian american_indian, big brother big_brother, brother, buil
ding family building_a a_family, coen bros coen_bros, coen brother coen_broth
ers, family, family, fun family movie fun_family family_movie, immigrant, imm
igration, incest, interracial marriage interracial_marriage, interracial roma
nce interracial_romance, marx brother marx_brothers, native american native_a
merican, prejudice, race, racism, sisterhood, sister

Cluster 9: Representative Tag - 'russia'
Tags: afghanistan, africa, australia, boston, british, california, cambodia,
canada, chile, china, city politics city_politics, england, europe, france, f
rench, hungary, india, indonesia, ireland, istanbul, italy, japan, jungle, l
a, la vega las_vegas, los angeles los_angeles, mexico, michigan, middle east
middle_east, new york new_york, new york city new_york york_city, paris, russ
ia, savannah, scotland, shangrila, siam, south africa south_africa, south ame
rica south_america, southern u southern_us, suburbia, tokyo, turkey, venice,
vietnam

Cluster 10: Representative Tag - 'doctor'
Tags: accident, aging, aid, asylum, blood, bloody, borg, cancer, cheating, ch
illy, christmas, class, clock, cold, conan, court, day hudson day_and and_hud
son, dc, dead wife dead_wife, deadpan, diabetes, doctor, dog, doll, dorothy,
email, excon, factory, figure skating figure_skating, franchise, generation x
generation_x, grace, hal, halloween, hammett, heist, heroine tight suit heroi
ne_in in_tight tight_suit, homeless, housekeeper, immortality, insurance, jaz
z, joss whedon joss_whedon, justice, lawyer, lawyer, matchmaker, mcu, mouse,
mockumentary, morrow, moving, nanny, olympics, pageant, postcollege, psychiat
rist, recap, reunion, revenge, rosebud, rug, screwball, shenanigan, skiing, s
tapler, statue, stranded, survival, teacher, teacher, thanksgiving, traditio
n, transplant, union, veterinarian, widowswidowers, woman, wonderwoman, workp
lace

Cluster 11: Representative Tag - 'spacecraft'
Tags: amtrak, atmospheric, aviation, military, nasa, navy, plane, settingspac
espace ship settingspacespace_ship, ship, shipwreck, space, space craft space
_craft, space epic space_epic, space station space_station, space travel spac
e_travel, spacecraft, stranger train strangers_on on_a a_train, submarine, su
bway, titanic, train, train

Cluster 12: Representative Tag - 'thriller'
Tags: alien, alone world alone_in in_the the_world, demon, film noir film_noi
r, firefly, independent, independent film independent_film, loneliness, mobst
er, mobster, moon, neonoir, noir, prequel, priest, prodigy, psychological thr
iller psychological_thriller, rob zombie rob_zombie, rogue, singleton, solitu
de, suspense, suspenseful, thriller, vampire, vampire, werewolf, zombie

Cluster 13: Representative Tag - 'space adventure space_adventure'
Tags: 2d animation 2d_animation, adventure, alternate reality alternate_reali
ty, alternate universe alternate_universe, androidscyborgs, android, animatio
n, cgi, chess, computer animation computer_animation, creature feature creatu
```

```
re_feature, disney, disney animated feature disney_animated animated_feature,
engrossing adventure engrossing_adventure, fantasy, fantasy world fantasy_wor
ld, game, high fantasy high_fantasy, indiana jones indiana_jones, live action
animation live_actionanimation, magic board game magic_board board_game, mode
rn fantasy modern_fantasy, narnia, pixar, reality tv reality_tv, road trip ro
ad_trip, robot, robot android robots_and and_androids, space adventure space_
adventure, time travel time_travel, timetravel, video game adaptation video_g
ame game_adaptation, video game video_game, virtual reality virtual_reality

Cluster 14: Representative Tag - 'artistic'
Tags: artsy, amazing artwork amazing_artwork, art, art house art_house, artho
use, artistic, artsy, beautiful scenery beautiful_scenery, beautiful visuals
beautiful_visuals, beautifully filmed beautifully_filmed, best picture best_p
icture, colorful, colourful, cool style cool_style, futuristic, gothic, graph
ic design graphic_design, great performance great_performances, great visuals
great_visuals, painter, photographer, photography, postmodern, rsustained str
ong stylized violence rsustained_strong strong_stylized stylized_violence, re
tro, scenic, stylish, stylized, visual, visually appealing visually_appealin
g, visually striking visually_striking, visually stunning visually_stunning,
well done well_done

Cluster 15: Representative Tag - 'entertaining'
Tags: boring, challenging, clever, complicated, confusing, creative, creativi
ty, dull, elegant, enigmatic, enjoyable, entertaining, episodic, exciting, fu
n, innovative, insightful, intellectual, intelligent, interesting scenario in
teresting_scenario, introspection, mystery, predictable, random, simple, smar
t, sophisticated, tedious, tricky, unconventional, unexplained, unintelligen
t, unoriginal, unpredictable, unusual, witty

Cluster 16: Representative Tag - 'sentimental'
Tags: anger, beat poetry beat_poetry, bittersweet, bleak, controversial, cryp
tic, depressing, depression, downbeat, emotional, epic, feelgood, grim, heart
breaking, heartwarming, highly quotable highly_quotable, iconic, inspirationa
l, inspiring, intense, longing, love, lyrical, masterpiece, melancholy, mood
y, motivational, nostalgia, obsession, overcomplicated, overrated, passion, p
oetic, purposefulness, quotable, realistic, sentimental, somber, suicide, muc
h love interest too_much much_love love_interest, understated, uplifting, whi
msical, wistful, wry

Cluster 17: Representative Tag - 'original'
Tags: 2001like, 70mm, 80, alter ego alter_ego, amish, austere, beatles, ben s
tiller ben_stiller, bizzare, blood splatter blood_splatters, britpop, classi
c, classic movie classic_movie, clousseau, cyberpunk, different, doc ock doc_
ock, favelas, klingons, lolita theme lolita_theme, macho, magneto, muppets, n
inotchka remake ninotchka_remake, ogre, old, oninous, original, othello, palm
e dor palme_dor, plastic surgery plastic_surgery, remade, remake, remaster, s
e hinton se_hinton, slasher, slim pickens slim_pickens, splatter, spoof, stil
ler, tense, thanos, toga, tolkein, toto, trucker, unique, updated classic upd
ated_classics, zither

Cluster 18: Representative Tag - 'andy kaufman andy_kaufman'
Tags: american idolatry american_idolatry, andy garcia andy_garcia, andy kauf
man andy_kaufman, andy samberg andy_samberg, astaire rogers astaire_and and_r
ogers, beethoven, harley quinn harley_quinn, harley quinns as harley_quinns q
uinns_ass, hepburn tracy hepburn_and and_tracy, jay silent bob jay_and and_si
lent silent_bob, lonesome polecat lonesome_polecat, margot robbie margot_robb
ie, mozart, new composer new_composer, nick nora charles nick_and and_nora no
ra_charles, nick hornby nick_hornby, opera, pee wee herman pee_wee wee_herma
n, rita hayworth dance rita_hayworth hayworth_can can_dance, rogers hammerste
in rogers_and and_hammerstein, salieri, saturday night live saturday_night ni
ght_live, simon garfunkel simon_and and_garfunkel, simon pegg simon_pegg, sin
ger, snl, space opera space_opera, entertainer the_entertainer, ferrell will_
ferrell
```

```
Cluster 19: Representative Tag - 'gunfight'
Tags: anime, boksdrama, boxing, boxing story boxing_story, bromance, bromanti
c, brooch, capone, capote, chuck palahniuk chuck_palahniuk, con artist con_ar
tist, con men con_man, duma, gintama, gun fu gun_fu, gun tactic gun_tactics,
gunfu, gunfight, gun, hayao miyazaki hayao_miyazaki, hearst, hit men hit_man,
hitman, jackie chan jackie_chan, kung fu kung_fu, martial art martial_art, ma
sculinity, men drag men_in in_drag, mma, motherfucker, palahnuik, samurai, si
nbad, sniper, studio ghibli studio_ghibli, sword fight sword_fight, truman ca
pote truman_capote, wapendrama, younger men younger_men

Cluster 20: Representative Tag - 'amazing'
Tags: adorable, amazing, annoying, awesome, beautiful, bizarre, brilliant, br
utal, cool, disappointing, genius, good, gruesome, harsh, hope, interesting,
lovely, mecha, nonsense, poignant, ridiculous, sad, sexy, silly, strange, swe
et, tragic, weird

Cluster 21: Representative Tag - 'whale'
Tags: ape, aquarium, bear, bird, camel, dinosaur, dinosaur, donkey, evolutio
n, fish, flood, horse, island, leopard, lion, lion, mermaid, ocean, parrot, p
ig, pool, river, shark, spider, surfing, thor, whale

Cluster 22: Representative Tag - 'restaurant'
Tags: clever chef rat a_clever clever_chef chef_rat, bluegrass, bubba gump sh
rimp bubba_gump gump_shrimp, bus, business, busniess, deep throat deep_throa
t, diner, enterprise, food, gritty, melodramatic kind dumb it_was was_melodra
matic melodramatic_and and_kind kind_of of_dumb, jekyll hyde jekyll_and and_h
yde, katzanzakis, mcdonalds, melancholic, milkshake, moldy, prince, pudding,
pulp, quaker, restaurant, royal cheese royal_with with_cheese, royalty, show
business show_business, shrimp, spaghetti western spaghetti_western, western,
wine

Cluster 23: Representative Tag - 'dark comedy dark_comedy'
Tags: adult humor adult_humor, avantgarde romantic comedy avantgarde_romantic
romantic_comedy, bad humor bad_humor, bad joke bad_jokes, best comedy best_co
medy, black comedy black_comedy, black humor black_humor, black humour black_
humour, british comedy british_comedy, comedy, crude humor crude_humor, dark
comedy dark_comedy, dark humor dark_humor, great humor great_humor, offbeat c
omedy offbeat_comedy, romantic comedy romantic_comedy, sexual humor sexual_hu
mor, standup comedy standup_comedy

Cluster 24: Representative Tag - 'big top big_top'
Tags: big boy gun big_boys boys_with with_guns, big budget big_budget, big co
rporation big_corporations, big name actor big_name name_actors, big top big_
top, big wave big_wave, coulda contender coulda_been been_a a_contender, larg
e cast large_cast, rich guy poor girl rich_guy guy_poor poor_girl, might gian
t they_might might_be be_giants

Cluster 25: Representative Tag - 'fast paced fast_paced'
Tags: far fetched far_fetched, fast paced fast_paced, fastpaced, fastpaced di
alogue fastpaced_dialogue, long shot long_shot, long take long_takes, poorly
paced poorly_paced, quick cut quick_cuts, slow, slow action slow_action, slow
paced slow_paced, long too_long, way long way_too too_long

Cluster 26: Representative Tag - 'horror'
Tags: crazy, creepy, disturbing, dreamlike, dream, dreamy, eerie, freak, ghos
t, ghost, haunting, horror, illusion, imagination, imaginative, insane, insan
ity, magic, nightmare, scary, surreal, surrealism, wizard

Cluster 27: Representative Tag - 'bad acting bad_acting'
Tags: acting, action, action choreography action_choreography, action packed
action_packed, bad, bad acting bad_acting, bad as bad_ass, bad language bad_l
anguage, bad script bad_script, bad writing bad_writing, badass, foul languag
e foul_language, good writing good_writing, great acting great_acting, horrib
le acting horrible_acting, horrible directing horrible_directing, r language
r language, rstrong language rstrong language, really bad really bad, smart w
```

```
riting smart_writing, space action space_action, writing

Cluster 28: Representative Tag - 'good dialogue good_dialogue'
Tags: amazing dialogue amazing_dialogues, bad dialogue bad_dialogue, conversa
tion, dialogue, entirely dialogue entirely_dialogue, good dialogue good_dialo
gue, great dialogue great_dialogue, dialogue no_dialogue, poor dialogue poor_
dialogue

Cluster 29: Representative Tag - 'scifi cult scifi_cult'
Tags: bad science bad_science, classic scifi classic_scifi, cult, cult classi
c cult_classic, cult film cult_film, intelligent scifi intelligent_scifi, mad
scientist mad_scientist, scifi, science fiction science_fiction, scifi, scifi
cult scifi_cult, scifi masterpiece scifi_masterpiece, sexy female scientist s
exy_female female_scientist

Cluster 30: Representative Tag - 'humorous'
Tags: dumb, funny, hilarious, humor, humorous, humour, ironic, irony, lie, fu
nny not_funny, parody, sarcasm, satire, satirical, stupid, stupid funny stupi
d_but but_funny, stupid stupid stupid_is is_as as_stupid stupid_does, truth,
funny very_funny

Cluster 31: Representative Tag - 'romance'
Tags: adultery, awkward, awkward romance awkward_romance, dating, divorce, do
uble life double_life, friendship, fucked fucked_up, gentle, homosexuality, i
maginary friend imaginary_friend, intimate, lesbian, lesbian subtext lesbian_
subtext, marriage, prostitution, quirky romantic quirky_romantic, rgraphic se
xuality rgraphic_sexuality, rape, romance, romantic, sex, sexuality, start be
autiful friendship start_of of_a a_beautiful beautiful_friendship, threesome,
thrilling, touching, twin, virginity, wedding, wedding

Cluster 32: Representative Tag - 'crime'
Tags: alcatraz, british gangster british_gangster, cia, conspiracy, conspirac
y theory conspiracy_theory, corruption, crime, espionage, fbi, fugitive, gan
g, gangster, gangster, hostage, investor corruption investor_corruption, kidn
apping, mafia, organised crime organised_crime, organized crime organized_cri
me, police, police corruption police_corruption, prison, ransom, robbery, sca
ndal, secret society secret_society, secret, small time criminal small_time t
ime_criminals, small town small_town, spy, spying, catholic church corrupt or
ganization history the_catholic catholic_church church_is is_the the_most mos
t_corrupt corrupt_organization organization_in in_history, undercover cop und
ercover_cop

Cluster 33: Representative Tag - 'bible'
Tags: ark covenant ark_of of_the the_covenant, bible, biblical reference bibl
ical_references, convent, holy grail holy_grail, judaism, jude law jude_law,
lord ring lord_of of_the the_rings, missionary, monty python monty_python, mo
ses, nun, nun, preacher, rabbi, religion, saint, saint, samuel l jackson samu
el_l l_jackson, televangelist, tolkien

Cluster 34: Representative Tag - 'nuclear war nuclear_war'
Tags: apocalypse, atomic bomb atomic_bomb, bomb, civil war civil_war, cold wa
r cold_war, disaster, dystopia, gulf war gulf_war, heroic bloodshed heroic_bl
oodshed, mccarthy hearing mccarthy_hearings, modern war modern_war, myth, myt
hology, nuclear disaster nuclear_disaster, nuclear war nuclear_war, post apoc
alyptic post_apocalyptic, ptsd, terrorism, tom clancy tom_clancy, tragedy, un
likely hero unlikely_hero, world war world_war war_i, world war ii world_war
war_ii

Cluster 35: Representative Tag - 'nudity topless nudity_topless'
Tags: based book based_on on_a a_book, based play based_on on_a a_play, based
tv show based_on on_a a_tv tv_show, celebrity fetishism celebrity_fetishism,
crime scene scrubbing crime_scene scene_scrubbing, embarassing scene embarass
ing_scenes, see dead people i_see see_dead dead_people, netflix queue in_netf
lix netflix_queue, eye in_your your_eyes, male nudity male_nudity, dvd netfli
```

```
x no_dvd dvd_at at_netflix, available netflix not_available available_from fr
om_netflix, seen not_seen, notable nudity notable_nudity, nudity full frontal
nudity_full full_frontal, nudity topless nudity_topless, seen cinema seen_at
at_the the_cinema, seen seen_more more_than than_once, something everyone one
saw without plan seeing kid something_for for_everyone everyone_in in_this th
is_one one_saw saw_it it_without without_and and_plan plan_on on_seeing seein
g_it it_with with_kids, stop looking swan stop_looking looking_at at_me me_sw
an, voyeurism

Cluster 36: Representative Tag - '1970s'
Tags: 1900s, 1920s, 1950s, 1960s, 1970s, 1980s, 1990s

Cluster 37: Representative Tag - 'superhero'
Tags: batman, cartoon, comic book comic_book, comic, dc comic dc_comics, hawk
eye, joker, knight, marvel, superhero, superhero team superhero_team, superma
n, wolverine

Cluster 38: Representative Tag - 'emma thompson emma_thompson'
Tags: angelina jolie angelina_jolie, audrey tautou audrey_tautou, brad pitt b
rad_pitt, brittany murphy brittany_murphy, cameowhoopi goldberg cameowhoopi_g
oldberg, charlotte bronte charlotte_bronte, emilia clarke emilia_clarke, emm
a, emma stone emma_stone, emma thompson emma_thompson, eva green eva_green, k
atie for_katie, gal gadot gal_gadot, girl power girl_power, halle berry halle
_berry, hilary swank hilary_swank, hot actress hot_actress, jake gyllenhaal j
ake_gyllenhaal, jennifer connelly jennifer_connelly, jennifer lawrence jennif
er_lawrence, jessica alba jessica_alba, julianne moore julianne_moore, maggie
gyllenhaal maggie_gyllenhaal, meryl streep meryl_streep, natalie portman nata
lie_portman, orlando bloom orlando_bloom, queen victoria queen_victoria, rach
el mcadams rachel_mcadams, rachel weisz rachel_weisz, renee zellweger renee_z
ellweger, tilda swinton tilda_swinton, winona ryder winona_ryder, zoe kazan z
oe_kazan

Cluster 39: Representative Tag - 'nonlinear timeline nonlinear_timeline'
Tags: disjointed timeline disjointed_timeline, nonlinear, nonlinear timeline
nonlinear_timeline, nonlinear, nonlinear narrative nonlinear_narrative, nonli
near storyline nonlinear_storyline, nonlinear timeline nonlinear_timeline

Cluster 40: Representative Tag - 'music'
Tags: bad music bad_music, good music good_music, good soundtrack good_soundt
rack, great soundtrack great_soundtrack, hip hop hip_hop, indie record label
indie_record record_label, music, music business music_business, music indust
ry music_industry, notable soundtrack notable_soundtrack, pop culture referen
ce pop_culture culture_references, radio, rap, remix culture remix_culture, s
oundtrack, superb soundtrack superb_soundtrack

Cluster 41: Representative Tag - 'violence'
Tags: brutality, casual violence casual_violence, child abuse child_abuse, do
mestic violence domestic_violence, meaningless violence meaningless_violence,
rdisturbing violent content including rape rdisturbing_violent violent_conten
t content_including including_rape, rdisturbing violent image rdisturbing_vio
lent violent_images, rsome violence rsome_violence, rstrong bloody violence r
strong_bloody bloody_violence, violence, violence america violence_in in_amer
ica, violent

Cluster 42: Representative Tag - 'parenthood'
Tags: dingo ate baby a_dingo dingo_ate ate_my my_baby, abortion, adoption, ba
by, child, fatherson relationship fatherson_relationship, fatherhood, give ba
ck son give_me me_back back_my my_son, father i_am am_your your_father, infer
tility, kid, missing child missing_children, motherhood, orphan, parenthood,
pregnancy, teenage pregnancy teenage_pregnancy

Cluster 43: Representative Tag - 'drug'
Tags: addiction, alcoholism, coke, drug abuse drug_abuse, drug overdose drug_
overdose, drug, drug music drugs_music, heroin, marijuana, tobacco
```

```
Cluster 44: Representative Tag - 'murder'
Tags: antiintellectual, antisemitism, antiwar, assassin, assassinintraining s
cene assassinintraining_scene, assassination, assassin, death, death penalty
death_penalty, fatalistic, genocide, holocaust, humane, inhumane, killer, kil
lerasprotagonist, murder, nazi, serial killer serial_killer, wrongful impriso
nment wrongful_imprisonment

Cluster 45: Representative Tag - 'geeky'
Tags: aggressive, cheeky, cheesy, childish naivity childish_naivity, confront
ational, cynical, easygoing, eccentric, elegiac, geeky, goofy, gore, goretast
ic, hippy, horrid characterisation horrid_characterisation, menacing, nerd, n
erd, offensive, quirky, rviolence, slick, swashbuckler, trippy, vulgar

Cluster 46: Representative Tag - 'great cinematography great_cinematography'
Tags: amazing cinematography amazing_cinematography, animal movie animal_movi
e, beautiful cinematography beautiful_cinematography, chick flick chick_flic
k, cinematography, crappy sequel crappy_sequel, directorial debut directorial
_debut, film history film_history, filmnoir, good cinematography good_cinemat
ography, great cinematography great_cinematography, great movie great_movie,
great screenplay great_screenplay, great villain great_villain, imdb top 250
imdb_top top_250, movie business movie_business, movie movie movies_about abo
ut_movies, sequel, short film short_films, unnecessary sequel unnecessary_seq
uel

Cluster 47: Representative Tag - 'history'
Tags: ancient rome ancient_rome, archaeology, biography, biopic, book, comput
er, computer, documentary, economics, fighting, future, gold, governess, hist
orical, history, medieval, mining, money, movie, oil, politics, pow, presiden
t, rebellion, revolution, revolutionary, roman, rome, slavery, technology, te
levision, tv, video, war, watergate

Cluster 48: Representative Tag - 'oscar best actress oscar_best best_actress'
Tags: 06 oscar nominated best movie animation 06_oscar oscar_nominated nomina
ted_best best_movie movie_animation, academy award best supporting actress ac
ademy_award award_best best_supporting supporting_actress, oscar best actress
oscar_best best_actress, oscar best cinematography oscar_best best_cinematogr
aphy, oscar best effect visual effect oscar_best best_effects effects_visual
visual_effects, oscar best music original score oscar_best best_music music_o
riginal original_score, oscar best supporting actress oscar_best best_support
ing supporting_actress

Cluster 49: Representative Tag - 'lack plot lack_of of_plot'
Tags: bad plot bad_plot, exquisite plotting exquisite_plotting, lack plot lac
k_of of_plot, original plot original_plot, plot hole plot_holes, poor plot de
velopment poor_plot plot_development, predictible plot predictible_plot, supe
rficial plot superficial_plot, weak plot weak_plot
```

Final Thoughts

By using BERT embeddings, we've improved the semantic representation of tags, leading to more meaningful clustering and a condensed tag list that makes sense. This list can enhance the user experience by simplifying the selection of tags for personalized recommendations.

Final Recommendation Algorithm

Now, we'll create a final function that integrates both Collaborative Filtering (CF) using a hybrid of SVD and KNN models, and Content-Based (CB) recommendations using the condensed tags and genres.

Overview

- User Input: The user can input their preferred genres and tags from the available lists.
- Collaborative Filtering: Uses a hybrid model combining SVD and KNN to predict ratings for unrated movies.
- Content-Based Filtering: Generates recommendations based on the user's preferred genres and tags using TF-IDF and cosine similarity.
- Hybrid Recommendation: Combines CF and CB predictions to generate a final list of recommended movies.

In [343…

```
def generate_hybrid_recommendations(user_id=None, svd_model=None, knn_model
                                    tfidf_matrix=None, tfidf_vectorizer=Non
                                    preferred_genres=None, preferred_tags=N
    """
    Generates hybrid recommendations by combining CF and CB predictions usi

    Parameters:
    - user_id (int, optional): The ID of the user.
    - svd_model: Trained SVD model.
    - knn_model: Trained KNN model.
    - movies_df (DataFrame): DataFrame containing movie information.
    - movies_with_tags (DataFrame): DataFrame containing movies with combin
    - tfidf_matrix (sparse matrix): TF-IDF matrix for movies.
    - tfidf_vectorizer (TfidfVectorizer): TF-IDF vectorizer used to create
    - ratings_df (DataFrame): DataFrame containing user ratings.
    - preferred_genres (list, optional): List of preferred genres.
    - preferred_tags (list, optional): List of preferred tags.
    - num_recommendations (int): Number of recommendations to return.

    Returns:
    - recommendations_df (DataFrame): DataFrame containing recommended movi
    """
    import numpy as np
    import pandas as pd
    from surprise import Prediction
    from sklearn.metrics.pairwise import cosine_similarity

    # Initialize an empty DataFrame for recommendations
    recommendations_df = pd.DataFrame()

    # Get all movie IDs
    all_movie_ids = movies_df['movieId'].unique()

    # Check if user_id is provided and exists in ratings_df
    if user_id is not None and user_id in ratings_df['userId'].unique():
        # Existing user with ratings
        user_ratings = ratings_df[ratings_df['userId'] == user_id]
        rated_movies = user_ratings['movieId'].tolist()
        unrated_movies = [movie_id for movie_id in all_movie_ids if movie_i

        # Collaborative Filtering Predictions (CF)
        cf_predictions = []
        for movie_id in unrated_movies:
            svd_pred = svd_model.predict(user_id, movie_id, verbose=False).
            knn_pred = knn_model.predict(user_id, movie_id, verbose=False).
            cf_hybrid_pred = (0.5 * svd_pred) + (0.5 * knn_pred)
```

```
        cf_predictions.append((movie_id, cf_hybrid_pred))
    cf_pred_df = pd.DataFrame(cf_predictions, columns=['movieId', 'cf_p

    # Content-Based Filtering Predictions (CB)
    # Build user profile vector
    user_profile_tfidf = np.zeros(tfidf_matrix.shape[1])

    # Build the profile based on rated movies
    for _, row in user_ratings.iterrows():
        movie_id = row['movieId']
        rating = row['rating']
        try:
            idx = movies_with_tags.index[movies_with_tags['movieId'] ==
            movie_tfidf = tfidf_matrix[idx].toarray().flatten()
            user_profile_tfidf += movie_tfidf * rating
        except IndexError:
            continue

    # Incorporate preferred genres and tags if provided
    if preferred_genres or preferred_tags:
        preference_text = ' '.join((preferred_genres or []) + (preferre
        preference_text = preprocess_text(preference_text)
        preference_vector = tfidf_vectorizer.transform([preference_text
        user_profile_tfidf += preference_vector * 2  # Weight preferenc

    # Normalize user profile vector
    norm = np.linalg.norm(user_profile_tfidf)
    if norm != 0:
        user_profile_tfidf /= norm

    # Compute cosine similarity between user profile and all movie vect
    cosine_similarities = cosine_similarity([user_profile_tfidf], tfidf

    # Map CB scores to rating scale (e.g., 0.5 to 5.0)
    min_rating = ratings_df['rating'].min()
    max_rating = ratings_df['rating'].max()
    cb_scores = cosine_similarities
    cb_predicted_ratings = cb_scores * (max_rating - min_rating) + min_

    # Create DataFrame for CB predictions
    cb_pred_df = pd.DataFrame({
        'movieId': movies_with_tags['movieId'],
        'cb_pred': cb_predicted_ratings
    })

    # Filter to unrated movies
    cb_pred_df = cb_pred_df[cb_pred_df['movieId'].isin(unrated_movies)]

    # Merge CF and CB predictions
    hybrid_pred_df = pd.merge(cf_pred_df, cb_pred_df, on='movieId', how

    # Adjust weights based on the number of ratings
    num_user_ratings = len(user_ratings)
    max_user_ratings = ratings_df.groupby('userId').size().max()
    cf_weight = num_user_ratings / max_user_ratings
    cb_weight = 1 - cf_weight

    # Normalize weights
    total_weight = cf_weight + cb_weight
    cf_weight /= total_weight
    cb_weight /= total_weight

    # Compute hybrid prediction
    hybrid_pred_df['hybrid_score'] = (cf_weight * hybrid_pred_df['cf_pr
```

```
            # Merge with movies DataFrame to get titles and genres
            hybrid_pred_df = pd.merge(hybrid_pred_df, movies_df[['movieId', 'ti

            # Sort by hybrid score in descending order
            recommendations_df = hybrid_pred_df.sort_values(by='hybrid_score',

            # Select top N recommendations
            recommendations_df = recommendations_df.head(num_recommendations).r

            # Select relevant columns
            recommendations_df = recommendations_df[['movieId', 'title', 'genre

        else:
            # New user or user without ratings
            # Use content-based recommendations based on preferred genres and t
            if preferred_genres is None and preferred_tags is None:
                print("No user ratings or preferences provided. Cannot generate
                return None

            # Combine preferred genres and tags into a single string
            preference_text = ' '.join((preferred_genres or []) + (preferred_ta
            preference_text = preprocess_text(preference_text)

            # Transform preference text into TF-IDF vector
            preference_vector = tfidf_vectorizer.transform([preference_text])

            # Compute cosine similarity between the preference vector and all m
            cosine_similarities = cosine_similarity(preference_vector, tfidf_ma

            # Add similarity scores to the dataframe
            movies_with_tags['similarity_score'] = cosine_similarities

            # Sort by similarity score in descending order
```