Title: Movie Recommendation for "Moviefix" Stakeholder

Phase 4 Project submission.

Name: Stephen Ndirangu (no group)

Business Understanding

The movie industry is fast growing, and with so many options, there is need for cutting edge user interactivity. This involves tailoring their movie watching experience via intuitive recommendations that can give them interesting options based on what they prefer, while also enticing them with other content that they may like

It is for this very reason we are working on a recommendation system for the "Moviefix" firm, in order to not only capture the users, but also keep them coming back for more. To accomplish this, our model will create 5(five) recommendations that will be given to the user.

Problem Statement

The main challenge is to design and implement a movie recommendation system that employs collaborative filtering techniques to predict movie preferences for users based on their past ratings. In short, to analyse their past activities and give them recommendations based on their tastes.

To address a potential "cold start" problem (where new users or movies have limited ratings), if possible, we will attempt to explore a hybrid approach that combines collaborative filtering with content-based filtering.

The success of this project will be measured by evaluating the accuracy and relevance of the recommendations via metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) that will gauge the performance of the model.

Ultimately, the objective is to build a recommendation system that enhances user engagement, encourages exploration of diverse movies, and contributes to the overall satisfaction of users on the platform.

Data understanding

The dataset describes 5-star rating and free-text tagging activity from [MovieLens](), a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in the files `links.csv`, `movies.csv`, `ratings.csv` and `tags.csv`. The dataset was provided by MovieLens, a movie recommendation service. It includes the movies and the rating scores made for these movies contains. It contains 100,000 ratings (1–5) from 943 users on 1682 movies.

1) Movies:

Has information about each movie, such as movie ID, title, and genres.

Columns:

- **movieId:** the unique numerical identifier for each movie. This ID is used to connect the movie information with the ratings and links datasets. This identifier is crucial for linking the movie information with other datasets, especially the ratings dataset. It acts as a key to connect the movie information with user interactions (ratings) and potentially external databases (links).
- **title:** The name of the movie together with its year of release, is a string type.
- **genres:** Genres associated with the movie. Each movie belongs to one or even a combination of genres, marking its type and both distinguishing it from others as well as linking it to a certain category. Genres are a pipe-separated list, and are selected from the following:
- Action
- Adventure

- Animation
- Children's
- Comedy
- Crime
- Documentary
- Drama
- Fantasy
- Film-Noir
- Horror
- Musical
- Mystery
- Romance
- Sci-Fi
- Thriller
- War
- Western
- (no genres listed)

2) Ratings Dataset:

The ratings dataset contains user-movie interactions, including user IDs, movie IDs, and ratings. Collaborative filtering algorithms will leverage this dataset to predict movie ratings for users based on their historical ratings.

Size: The dataset contains information about user-movie interactions, where each row represents a user's rating for a specific movie.

Columns:

- **userId:** unique integer identifier for each user, to track their interactions
- **movieId:** A unique integer identifier for each movie. This identifier connects the ratings with specific movies. It links user ratings to the movies they've interacted with.
- **rating:** The value representing how much a user liked a particular movie. ranging from 1 to 5, with half-star increments.
- **timestamp:** A timestamp indicating when the rating was given. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

3)Tags: Tags are user-generated metadata about movies. Each tag is typically a single word or short phrase. The meaning, value, and purpose of a particular tag is determined by each user. Columns:

- **userId** The user's unique Identifier
- **movieId** The Movie's Unique identifier
- **tag**- the tag entered by a user to describe a movie
- **timestamp**-Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

4) Links Dataset: Identifiers that can be used to link to other sources of movie data, that is external databases Columns:

- **movieId:** A unique identifier for each movie. This identifier corresponds to the movie ID in the MovieLens dataset.
- **imdbId:** The identifier of the movie in the IMDb (Internet Movie Database) system. This identifier is used to connect the movie with its corresponding entry in the IMDb database. IMDb is a widely-used database for movie information, including details about cast, crew, plot, and more.
- **tmdbId:** The identifier of the movie in the TMDB (The Movie Database) system. This identifier links the movie to its corresponding entry in the TMDB database.

This dataset might offer additional contextual information for content-based filtering, especially for new users.

Exploratory Data Analysis

```
!pip install scikit-surprise
```

```
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.10/dist-packages (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.3.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.10.1)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from wordcloud import WordCloud
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import TruncatedSVD


from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from surprise import accuracy
```

Load all our datasets

```
movies = pd.read_csv('/content/movies.csv',encoding='latin-1')
ratings = pd.read_csv('/content/ratings.csv')
tags = pd.read_csv('/content/tags.csv')
links= pd.read_csv('/content/links.csv')
```

Movies

```
print('Shape of Movies :',movies.shape)
movies.head(10)
```

```
Shape of Movies : (9742, 3)
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

```
movies.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
```

```
print("Unique MovieIds: ", movies['movieId'].nunique())
print("Unique titles: ", movies['title'].nunique())
print("Unique genres: ", movies['genres'].nunique())
```

```
Unique MovieIds:  9742
Unique titles:  9737
Unique genres:  951
```

There are 951 unique Genres, while MovieId has 9742 unique entries and Titles have 9737 unique entries out of a possible 9742. This could mean repetitions

Lets check for duplicates

```
print("duplicates in ID: ", movies.movieId.duplicated().sum())
print("duplicates in Title: ", movies.title.duplicated().sum())
print("duplicates in Genres: ", movies.genres.duplicated().sum())
```

```
duplicates in ID:  0
duplicates in Title:  5
duplicates in Genres:  8791
```

Duplicates in Title are going to be a problem, but those in genre simply mean a movie is denoted as being in the same genres, we will clean this later.

Ratings

```
print('Shape of ratings :',ratings.shape)
ratings.head()
```

```
Shape of ratings : (100836, 4)
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

```
ratings.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
```

```
ratings.describe()
```

	userId	movieId	rating	timestamp
count	100836.000000	100836.000000	100836.000000	1.008360e+05
mean	326.127564	19435.295718	3.501557	1.205946e+09
std	182.618491	35530.987199	1.042529	2.162610e+08
min	1.000000	1.000000	0.500000	8.281246e+08
25%	177.000000	1199.000000	3.000000	1.019124e+09
50%	325.000000	2991.000000	3.500000	1.186087e+09
75%	477.000000	8122.000000	4.000000	1.435994e+09
max	610.000000	193609.000000	5.000000	1.537799e+09

Tags

```
print("tags shape: ", tags.shape)
tags.head()
```

```
tags shape:  (3683, 4)
```

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200

```
tags.value_counts().sum()
```

```
3683
```

```
tags.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
```

```
tags['userId'].nunique()
```

```
58
```

only 58 unique tags, meaning users use many similar tags to describe the movies

Links

```
links.head()
```

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

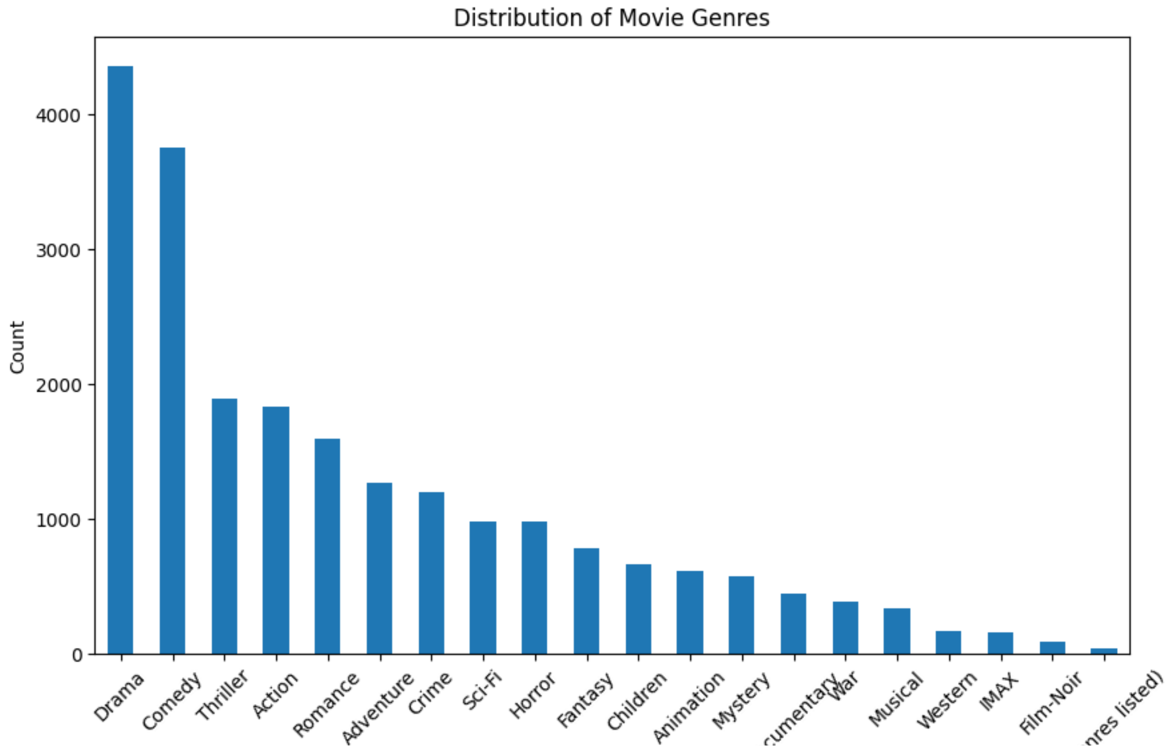
```
links.info()
```

```
<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
```

Links only helps with access to added resources from IMDB and TMDB, we shall see how useful it is

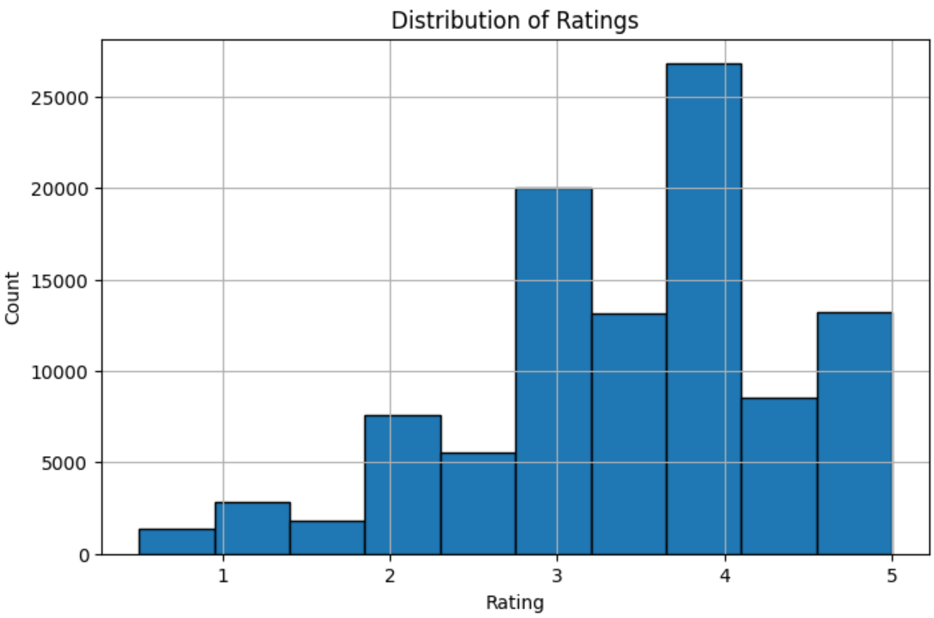
Univariate analysis

```
# Split genres into individual genres
movies['genre_list'] = movies['genres'].str.split('|')

genre_counts = movies['genre_list'].explode().value_counts()

#distribution of genres
plt.figure(figsize=(10, 6))
genre_counts.plot(kind='bar')
plt.title('Distribution of Movie Genres')
plt.xlabel('Genres')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



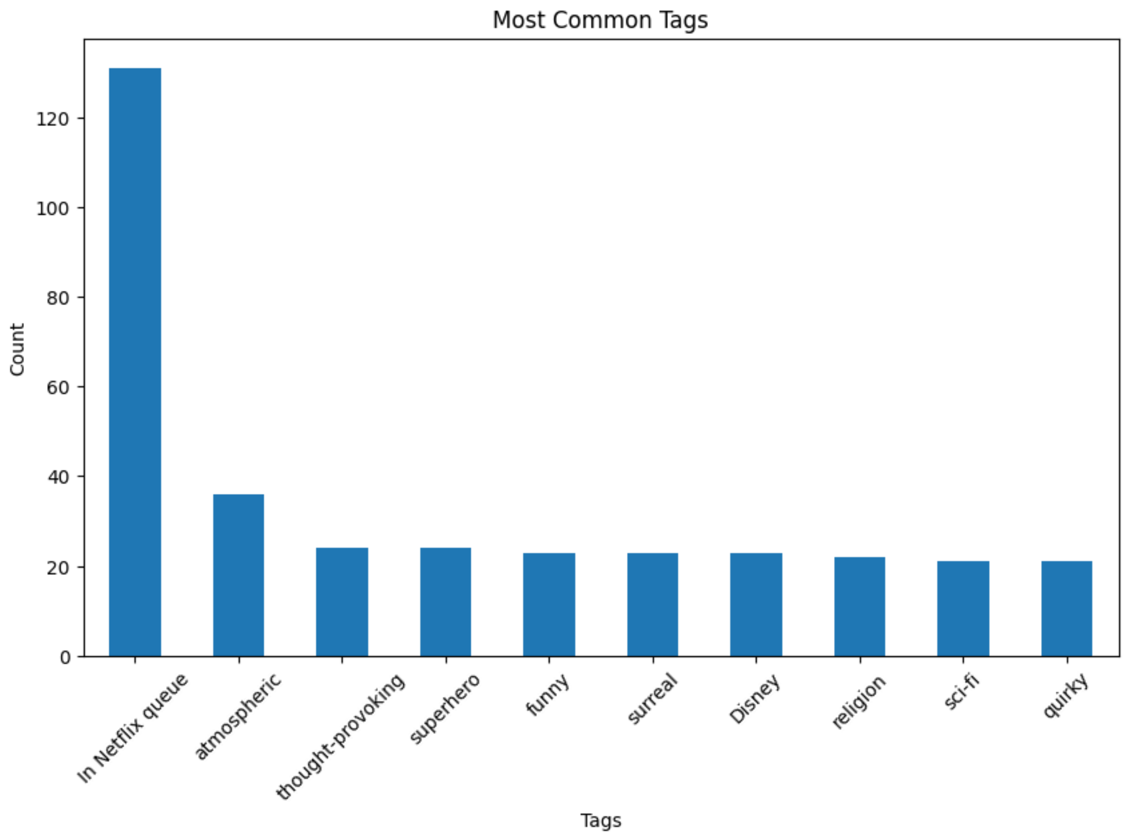
Drama has the most occurrence,followed by comedy and then thriller

```
#for ratings

plt.figure(figsize=(8, 5))
ratings['rating'].hist(bins=10, edgecolor='black')
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```

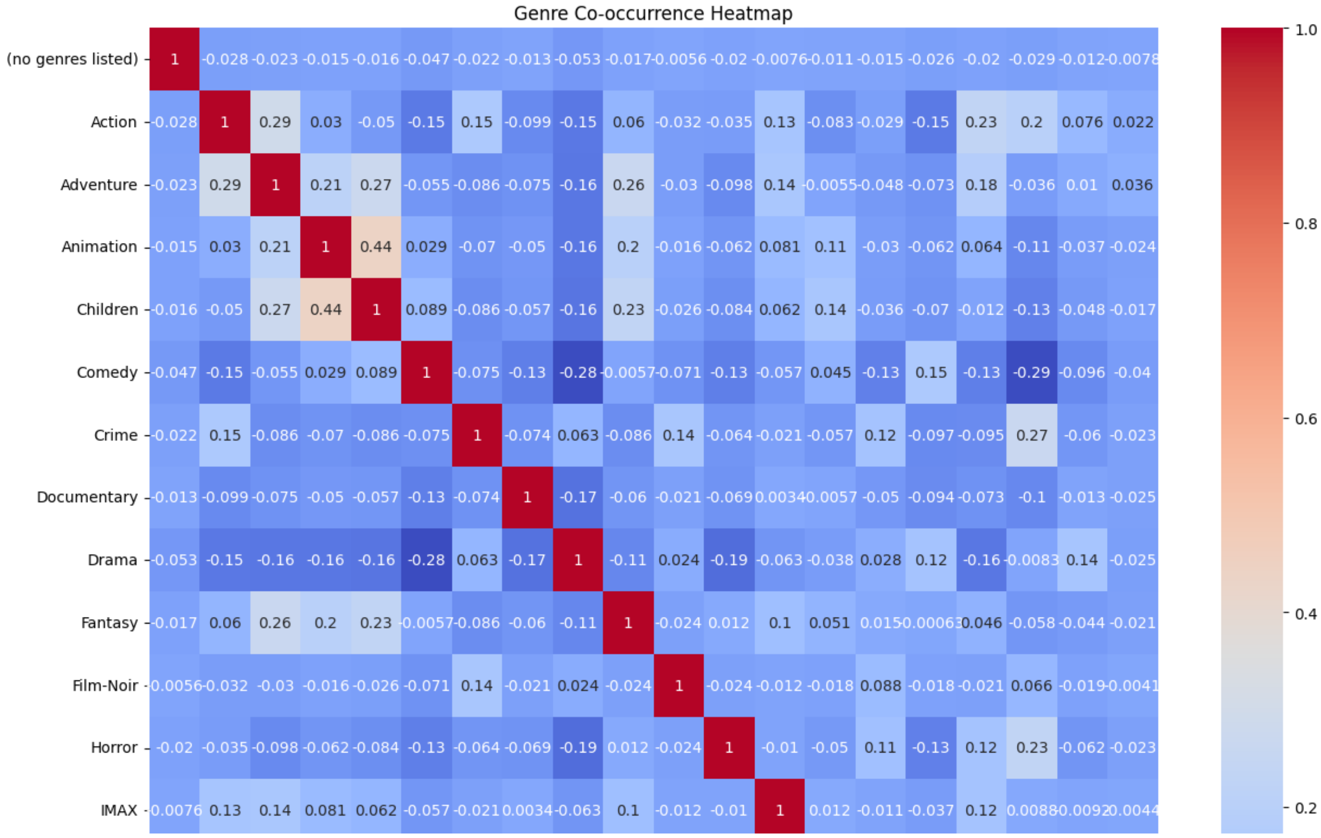



```
# For tags
tag_counts = tags['tag'].value_counts().head(10)

# Plot the most common tags
plt.figure(figsize=(10, 6))
tag_counts.plot(kind='bar')
plt.title('Most Common Tags')
plt.xlabel('Tags')
plt.ylabel('Count')
plt.xticks(rotation=45)
```

```
plt.show()
```

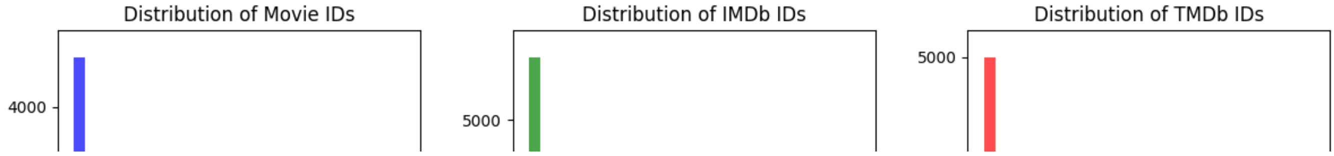



```
genre_matrix = pd.get_dummies(movies['genre_list'].apply(pd.Series).stack()).groupby(level=0).sum()
plt.figure(figsize=(15, 15))
sns.heatmap(genre_matrix.corr(), annot=True, cmap="coolwarm")
plt.title('Genre Co-occurrence Heatmap')
plt.show()
```



for correlations, the highest is animation and children's films

```
plt.figure(figsize=(12, 6))

# Distribution of Movie IDs
plt.subplot(1, 3, 1)
plt.hist(links["movieId"], bins=30, color='blue', alpha=0.7)
plt.title("Distribution of Movie IDs")

# Distribution of IMDb IDs
plt.subplot(1, 3, 2)
plt.hist(links["imdbId"], bins=30, color='green', alpha=0.7)
plt.title("Distribution of IMDb IDs")

# Distribution of TMDb IDs
plt.subplot(1, 3, 3)
plt.hist(links["tmdbId"], bins=30, color='red', alpha=0.7)
plt.title("Distribution of TMDb IDs")

plt.tight_layout()
plt.show()
```

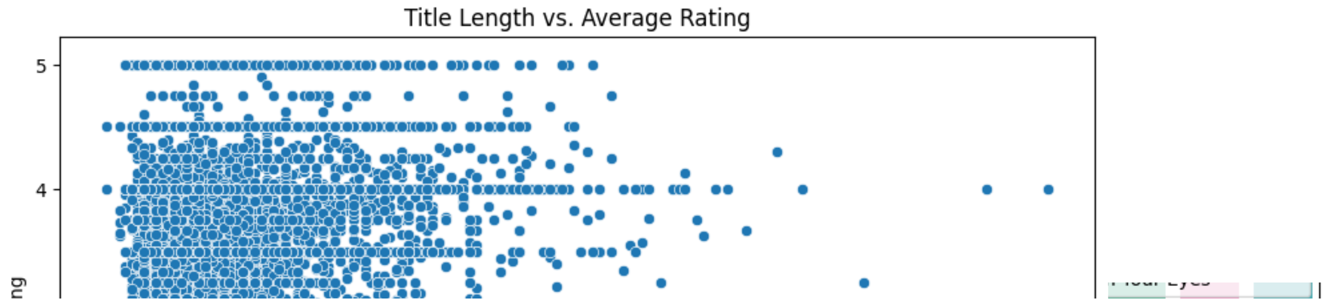



```
# Load a subset of the data (random sampling)
subset_tags = tags.sample(n=1000, random_state=42)
subset_movies = movies.sample(n=1000, random_state=42)

# Merge subset_tags with subset_movies based on movieId
subset_genre_tags = pd.merge(subset_tags, subset_movies[['movieId', 'genre_list']], on='movieId')

# Explode the genre_list column
subset_genre_tags = subset_genre_tags.explode('genre_list')

# Plot the tag frequencies by genre using a countplot
plt.figure(figsize=(12, 6))
sns.countplot(data=subset_genre_tags, x='genre_list', hue='tag', dodge=False)
plt.title('Tag Frequencies by Genre')
plt.xlabel('Genres')
plt.ylabel('Tag Count')
plt.xticks(rotation=45)
plt.legend(title='Tag')
plt.show()
```

Bivariate analysis

TITLE length vs Ratings

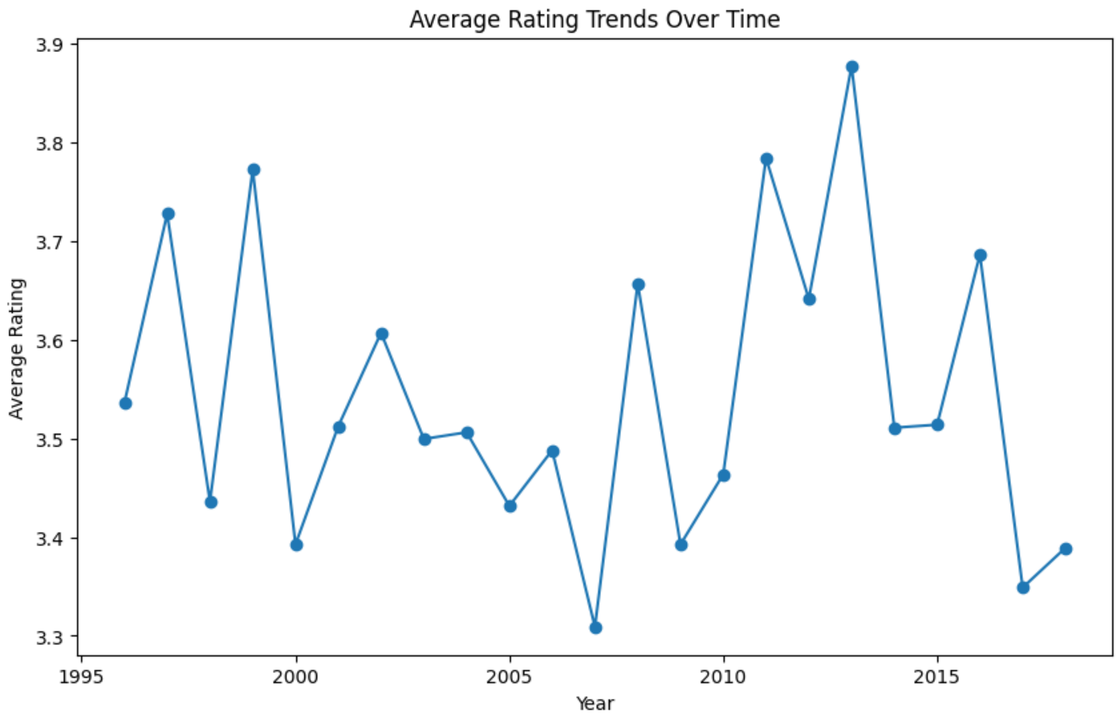
```
movies['title_length'] = movies['title'].apply(len)
merged = pd.merge(movies, ratings.groupby('movieId')['rating'].mean(), on='movieId')
plt.figure(figsize=(10, 6))
sns.scatterplot(data=merged, x='title_length', y='rating')
plt.title('Title Length vs. Average Rating')
plt.xlabel('Title Length')
plt.ylabel('Average Rating')
plt.show()
```



The sweet spot is around 20-60 minutes, a lower length is more related to the rating than higher length

Ratings over time

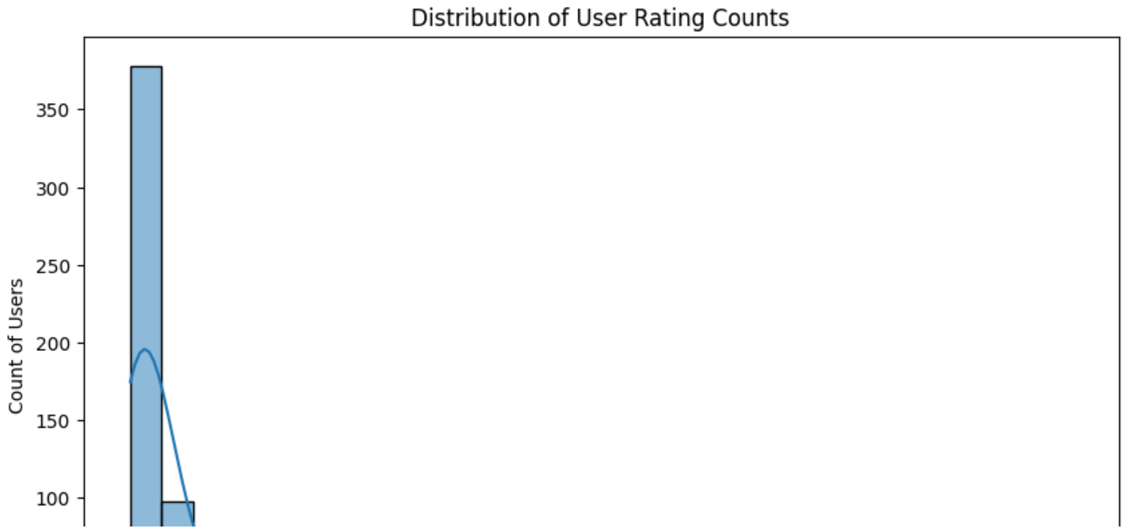
```
ratings['timestamp'] = pd.to_datetime(ratings['timestamp'], unit='s')
ratings_per_year = ratings.groupby(ratings['timestamp'].dt.year)['rating'].mean()
plt.figure(figsize=(10, 6))
ratings_per_year.plot(marker='o')
plt.title('Average Rating Trends Over Time')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.show()
```



These vary too much, probably on other factors

Rating by user

```
user_rating_counts = ratings['userId'].value_counts()
plt.figure(figsize=(10, 6))
sns.histplot(user_rating_counts, bins=30, kde=True)
plt.title('Distribution of User Rating Counts')
plt.xlabel('Number of Ratings')
plt.ylabel('Count of Users')
plt.show()
```




```
tag_text = ' '.join(tags['tag'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(tag_text)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Tag Word Cloud')
plt.axis('off')
plt.show()
```

Data Cleaning

Movies

Here we can take care of the duplicates we found

```
duplicate_titles = movies[movies.duplicated(subset="title", keep=False)]
duplicate_titles
```

	movieId	title	genres	genre_list
650	838	Emma (1996)	Comedy|Drama|Romance	[Comedy, Drama, Romance]
2141	2851	Saturn 3 (1980)	Adventure|Sci-Fi|Thriller	[Adventure, Sci-Fi, Thriller]
6932	64997	War of the Worlds (2005)	Action|Sci-Fi	[Action, Sci-Fi]
9468	168358	Saturn 3 (1980)	Sci-Fi|Thriller	[Sci-Fi, Thriller]

```
print("duplicates in ID: ", movies.movieId.duplicated().sum())
print("duplicates in Title: ", movies.title.duplicated().sum())
print("duplicates in Genres: ", movies.genres.duplicated().sum())
```

```
duplicates in ID:  0
duplicates in Title:  5
duplicates in Genres:  8791
```

```
# Remove duplicates based on the "title" column, keeping the first occurence
movies.drop_duplicates(subset="title", keep="first", inplace=True)
```

Tags

```
# Remove duplicates based on "userId", "movieId", and "tag" columns
tags.drop_duplicates(subset=["userId", "movieId", "tag"], keep="first", inplace=True)

# Check for missing values
print(tags.isnull().sum())

# Reset the index of the dataframe
tags.reset_index(drop=True, inplace=True)
```

```
userId       0
movieId      0
tag          0
timestamp    0
dtype: int64
```

```
tags.head()
```

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200

Ratings

```
# Remove duplicates based on "userId" and "movieId" columns
ratings.drop_duplicates(subset=["userId", "movieId"], keep="first", inplace=True)

# Check for missing values
print(ratings.isnull().sum())

# Reset the index of the dataframe
ratings.reset_index(drop=True, inplace=True)
```

```
userId       0
movieId      0
rating       0
timestamp    0
dtype: int64
```

```
ratings.head()
```

Links

```
# Remove duplicates based on "movieId" column
links.drop_duplicates(subset=["movieId"], keep="first", inplace=True)

# Check for missing values
print(links.isnull().sum())

# Reset the index of the dataframe
links.reset_index(drop=True, inplace=True)
```

```
movieId    0
imdbId     0
tmdbId     8
dtype: int64
```

```
# Find rows with null values in the "tmdbId" column
null_tmdb_links = links[links["tmdbId"].isnull()]

# Display rows with null values in the "tmdbId" column
print(null_tmdb_links)
```

```
      movieId  imdbId  tmdbId
624       791  113610     NaN
843      1107  102336     NaN
2141     2851   81454     NaN
3027     4051   56600     NaN
5532    26587   92337     NaN
5854    32600  377059     NaN
6059    40697  105946     NaN
7382    79299  874957     NaN
```

```
links["tmdbId"].fillna(-1, inplace=True)
```

```
links.head()
```

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

```
ratings.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

```
ratings.userId.nunique()
```

```
610
```

Merging data

```
# Merge movies and ratings dataframes based on movieId
movie_ratings = pd.merge(movies, ratings, on='movieId', how='inner')

# Merge the result with links dataframe based on movieId
movie_ratings_links = pd.merge(movie_ratings, links, on='movieId', how='inner')

# Merge the result with tags dataframe based on userId and movieId
```

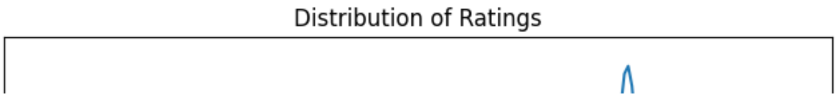
```
consolidated_data = pd.merge(movie_ratings_links, tags, on=['userId', 'movieId'], how='left')
```

Exploring the merged Data

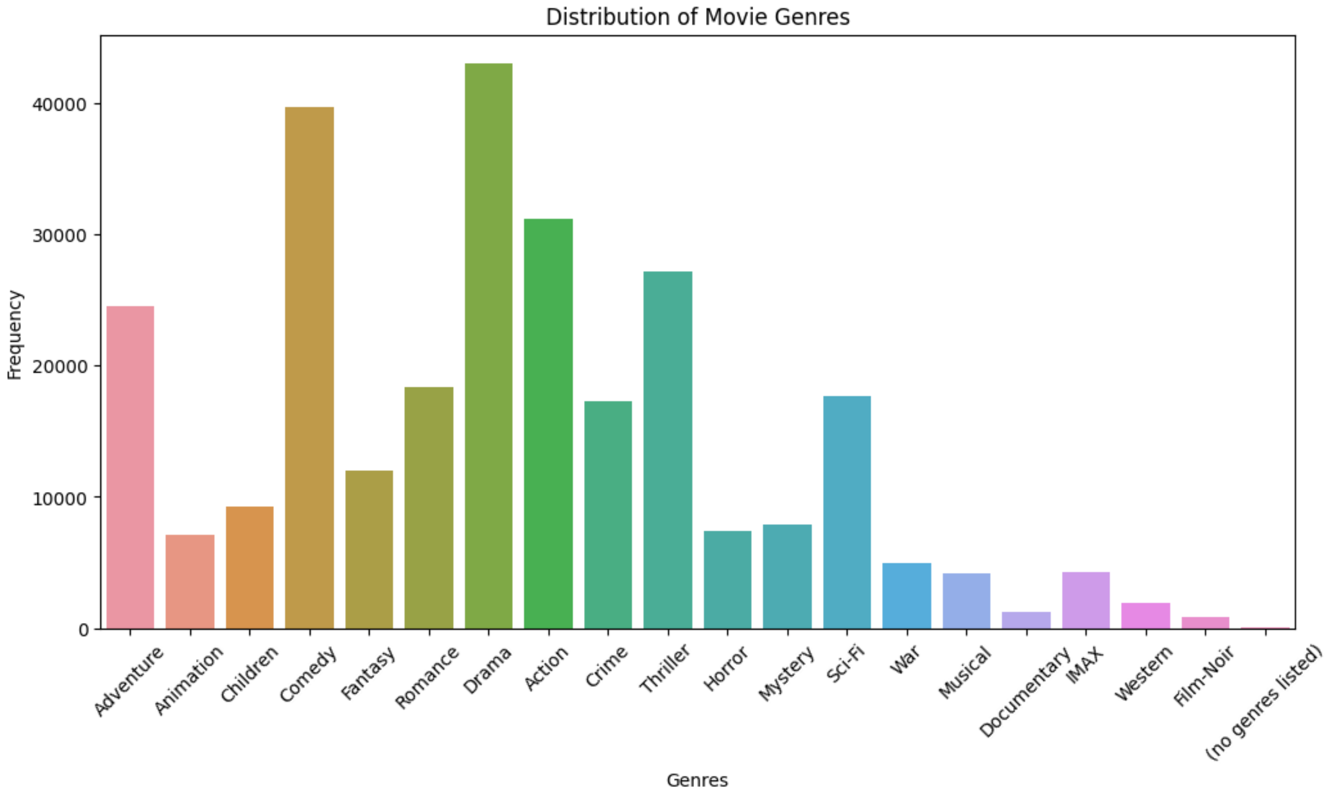
```
# 1. Data Summary
print("Data Summary:")
print(consolidated_data.info())
print(consolidated_data.describe())
```

```
Data Summary:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 102671 entries, 0 to 102670
Data columns (total 11 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   movieId      102671 non-null  int64
 1   title        102671 non-null  object
 2   genres       102671 non-null  object
 3   genre_list   102671 non-null  object
 4   userId       102671 non-null  int64
 5   rating       102671 non-null  float64
 6   timestamp_x  102671 non-null  int64
 7   imdbId       102671 non-null  int64
 8   tmdbId       102671 non-null  float64
 9   tag          3476 non-null    object
 10  timestamp_y  3476 non-null    float64
dtypes: float64(3), int64(4), object(4)
memory usage: 9.4+ MB
None
             movieId         userId         rating   timestamp_x  \
count  102671.000000  102671.000000  102671.000000  1.026710e+05
mean    19737.857681     327.766010       3.514824  1.209483e+09
std     35877.574462     183.209413       1.043152  2.170104e+08
min         1.000000       1.000000       0.500000  8.281246e+08
25%      1199.000000     177.000000       3.000000  1.019138e+09
50%      3005.000000     328.000000       3.500000  1.186590e+09
75%      8366.000000     477.000000       4.000000  1.439916e+09
max    193609.000000     610.000000       5.000000  1.537799e+09

             imdbId         tmdbId   timestamp_y
count  1.026710e+05  102671.000000  3.476000e+03
mean   3.565036e+05   20474.049147  1.323525e+09
std    6.295889e+05   54096.181084  1.731554e+08
min    4.170000e+02      -1.000000  1.137179e+09
25%    9.970650e+04     709.000000  1.138032e+09
50%    1.188420e+05    6950.000000  1.279956e+09
75%    3.172480e+05   11667.000000  1.498457e+09
max    8.391976e+06  525662.000000  1.537099e+09
```

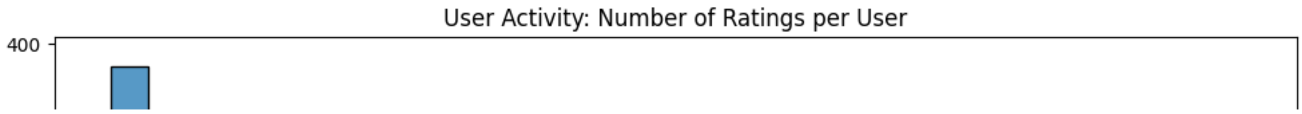
```
# 2. Distribution of Ratings
plt.figure(figsize=(8, 6))
sns.histplot(data=consolidated_data, x='rating', bins=10, kde=True)
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



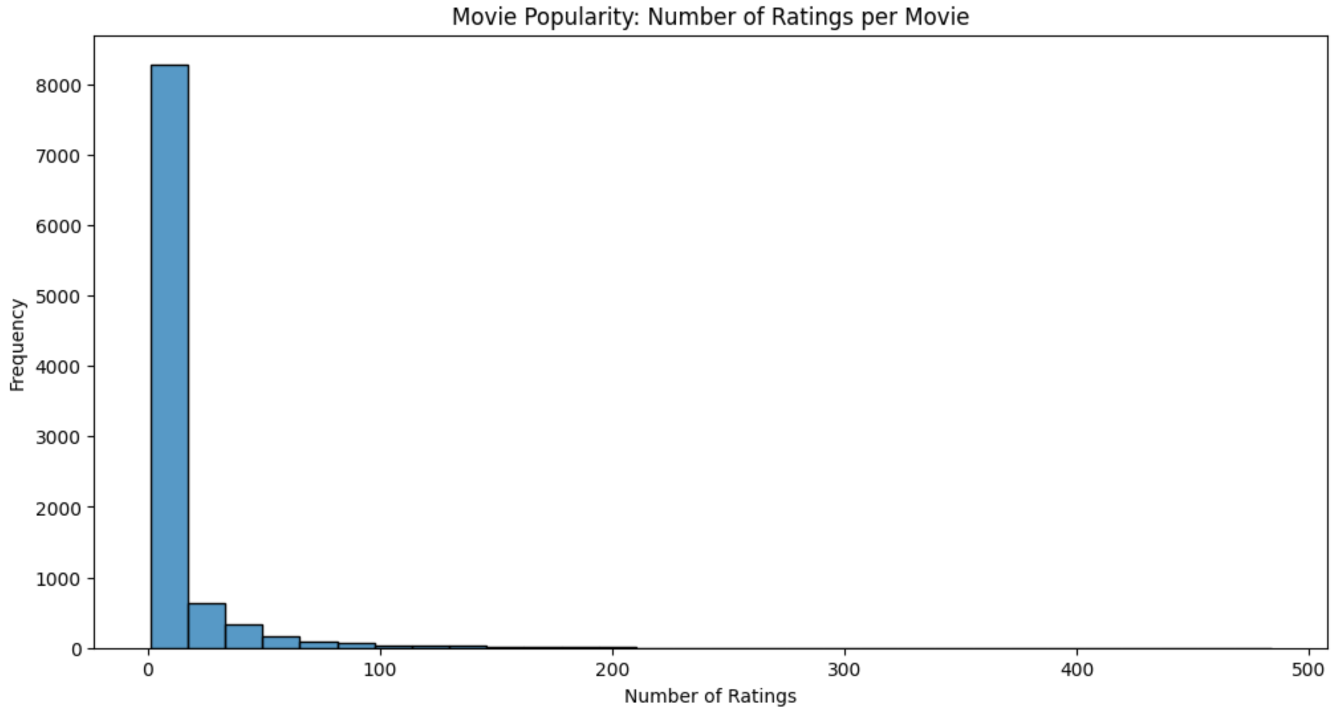

```
# 3. Genres Distribution
plt.figure(figsize=(12, 6))
sns.countplot(data=consolidated_data.explode('genre_list'), x='genre_list')
plt.title('Distribution of Movie Genres')
plt.xlabel('Genres')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```




```
# 4. User Activity
user_activity = consolidated_data['userId'].value_counts()
plt.figure(figsize=(12, 6))
sns.histplot(user_activity, bins=30)
plt.title('User Activity: Number of Ratings per User')
plt.xlabel('Number of Ratings')
plt.ylabel('Frequency')
plt.show()
```




```
# 5. Movie Popularity
movie_popularity = consolidated_data['movieId'].value_counts()
plt.figure(figsize=(12, 6))
sns.histplot(movie_popularity, bins=30)
plt.title('Movie Popularity: Number of Ratings per Movie')
plt.xlabel('Number of Ratings')
plt.ylabel('Frequency')
plt.show()
```




```
consolidated_data.head(20)
```

	movieId	title	genres	genre_list	userId	rating	timestamp_x	imdbId	tmdbId	tag	timesta
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	1	4.0	964982703	114709	862.0	NaN	
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	5	4.0	847434962	114709	862.0	NaN	
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	7	4.5	1106635946	114709	862.0	NaN	

```
merged_rating_count=consolidated_data.dropna(axis=0, subset=["title"])
rating_count = (merged_rating_count.groupby(by = ["title"])["rating"].count().reset_index().rename(columns = {"rating":"totalRatingCount'

rating_count
```

	title	totalRatingCount
0	'71 (2014)	1
1	'Hellboy': The Seeds of Creation (2004)	1
2	'Round Midnight (1986)	2
3	'Salem's Lot (2004)	1
4	'Til There Was You (1997)	2
...	...	...
9714	eXistenZ (1999)	22
9715	xXx (2002)	24
9716	xXx: State of the Union (2005)	5
9717	Â¡Three Amigos! (1986)	26
9718	Ã nous la libertÃ© (Freedom for Us) (1931)	1

9719 rows × 2 columns

Tov Animation.

```
combined_ratingCount_data = consolidated_data.merge(rating_count,left_on="title", right_on="title", how="left")
combined_ratingCount_data.head()
```

	movieId	title	genres	genre_list	userId	rating	timestamp_x	imdbId	tmdbId	tag	timestamp_
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	1	4.0	964982703	114709	862.0	NaN	Na
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	5	4.0	847434962	114709	862.0	NaN	Na
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	7	4.5	1106635946	114709	862.0	NaN	Na
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	15	2.5	1510577970	114709	862.0	NaN	Na
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	17	4.5	1305696483	114709	862.0	NaN	Na

```
pd.set_option("display.float_format", lambda x:"%.3f" % x)
print(rating_count['totalRatingCount'].describe())
```

```
count   9719.000
mean      10.564
std       23.131
min        1.000
25%        1.000
50%        3.000
75%        9.000
max      484.000
Name: totalRatingCount, dtype: float64
```

```
popularity_threshold =50
rating_popular_movie = combined_ratingCount_data.query("totalRatingCount >= " + str(popularity_threshold))
rating_popular_movie
```

	movieId	title	genres	genre_list	userId	rating	timestamp_x	imdbId	tmdbId	ta
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	1	4.000	964982703	114709	862.000	Na
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	5	4.000	847434962	114709	862.000	Na
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	7	4.500	1106635946	114709	862.000	Na
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	15	2.500	1510577970	114709	862.000	Na
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	17	4.500	1305696483	114709	862.000	Na
...	...	...	...	...	...	...	...	...	...	..
100002	122904	Deadpool (2016)	Action|Adventure|Comedy|Sci-Fi	[Action, Adventure, Comedy, Sci-Fi]	561	2.000	1491095067	1431045	293660.000	Na
100003	122904	Deadpool (2016)	Action|Adventure|Comedy|Sci-Fi	[Action, Adventure, Comedy, Sci-Fi]	586	4.000	1529899267	1431045	293660.000	Na
100004	122904	Deadpool (2016)	Action|Adventure|Comedy|Sci-Fi	[Action, Adventure, Comedy, Sci-Fi]	596	4.000	1535709074	1431045	293660.000	Na
100005	122904	Deadpool (2016)	Action|Adventure|Comedy|Sci-Fi	[Action, Adventure, Comedy, Sci-Fi]	599	3.500	1519457935	1431045	293660.000	Na
100006	122904	Deadpool (2016)	Action|Adventure|Comedy|Sci-Fi	[Action, Adventure, Comedy, Sci-Fi]	610	3.000	1493845981	1431045	293660.000	Na

42683 rows × 12 columns

```
rating_popular_movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 42683 entries, 0 to 100006
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   movieId          42683 non-null  int64
 1   title            42683 non-null  object
 2   genres           42683 non-null  object
 3   genre_list       42683 non-null  object
 4   userId           42683 non-null  int64
 5   rating           42683 non-null  float64
```

```
 6   timestamp_x       42683 non-null  int64
 7   imdbId            42683 non-null  int64
 8   tmdbId            42683 non-null  float64
 9   tag               1335 non-null   object
 10  timestamp_y       1335 non-null   float64
 11  totalRatingCount  42683 non-null  int64
dtypes: float64(3), int64(5), object(4)
memory usage: 4.2+ MB
```

Feature selection

The Tags and their timestamps are not really relevant to our approach here, neither are the links as we will not be using outside resources, but instead the existing data, especially due to their missing values, genre_list is a better representation of genres and title length is not really relevant to recommendation so we shall drop them

```
columns_to_drop = ["imdbId", "tmdbId", "tag", "timestamp_y","genres"]
movie_features1 = rating_popular_movie.drop(columns=columns_to_drop)
```

```
movie_features1= movie_features1.rename(columns={"genre_list": "genres", "timestamp_x": "timestamp"})
```

```
movie_features1
```

	movieId	title	genres	userId	rating	timestamp	totalRatingCount
0	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	1	4.000	2000-07-30 18:45:03	215
1	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	5	4.000	1996-11-08 06:36:02	215
2	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	7	4.500	2005-01-25 06:52:26	215
3	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	15	2.500	2017-11-13 12:59:30	215
4	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	17	4.500	2011-05-18 05:28:03	215
...	...	...	...	...	...	...	...
100002	122904	Deadpool (2016)	[Action, Adventure, Comedy, Sci-Fi]	561	2.000	2017-04-02 01:04:27	54
100003	122904	Deadpool (2016)	[Action, Adventure, Comedy, Sci-Fi]	586	4.000	2018-06-25 04:01:07	54

```
movie_features1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 42683 entries, 0 to 100006
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   movieId           42683 non-null  int64
 1   title             42683 non-null  object
 2   genres            42683 non-null  object
 3   userId            42683 non-null  int64
 4   rating            42683 non-null  float64
 5   timestamp         42683 non-null  datetime64[ns]
 6   totalRatingCount  42683 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(2)
memory usage: 2.6+ MB
```

```
movie_features_selected = movie_features1.pivot_table(index="title", columns="userId", values="rating").fillna(0)
movie_features_selected.head(10)
```

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608
title																			
10 Things I Hate About You (1999)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	3.000	0.000	5.000	0.000	0.000	0.000
12 Angry Men (1957)	0.000	0.000	0.000	5.000	0.000	0.000	0.000	0.000	0.000	0.000	...	5.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2001: A Space Odyssey (1968)	0.000	0.000	0.000	0.000	0.000	0.000	4.000	0.000	0.000	0.000	...	0.000	0.000	5.000	0.000	0.000	5.000	0.000	3.000
28 Days Later (2002)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.500
Old Virgin,	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Nearest Neighbors Model, utilising cosine metric

```
movie_features_selected_matrix = csr_matrix(movie_features_selected.values)
```

```
model_knn = NearestNeighbors(metric ="cosine", algorithm="brute")
model_knn.fit(movie_features_selected_matrix)
```

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine')
```

```
movie_features_selected.shape
```

```
(459, 606)
```

```
query_index = np.random.choice(movie_features_selected.shape[0])
print(query_index)
distances, indices =model_knn.kneighbors(movie_features_selected.iloc[query_index,:].values.reshape(1,-1), n_neighbors = 6)
```

```
326
```

```
for i in range(0, len(distances.flatten())):
  if i==0:
    print("recommendations for {0}:\n".format(movie_features_selected.index[query_index]))
  else:
    print("{0}:{1}, with distance of {2}:".format(i, movie_features_selected.index[indices.flatten()[i]], distances.flatten()[i]))
```

```
recommendations for Quiz Show (1994):

1:Crimson Tide (1995), with distance of 0.47089748328125036:
2:Dances with Wolves (1990), with distance of 0.498502023262272:
3:Get Shorty (1995), with distance of 0.5054374745389245:
4:Fugitive, The (1993), with distance of 0.5102065722438254:
5:Firm, The (1993), with distance of 0.511727793299094:
```

```
def get_movie_recommendations(movie_title, n_neighbors=6):
    try:
        # Find the index of the given movie title
        movie_index = movie_features_selected.index.get_loc(movie_title)

        # Query the k-NN model for nearest neighbors
        distances, indices = model_knn.kneighbors(movie_features_selected.iloc[movie_index, :].values.reshape(1, -1), n_neighbors=n_neigh

        # Print recommendations
        print("Recommendations for {0}:\n".format(movie_title))
        for i in range(0, len(distances.flatten())):
            if i == 0:
                continue
            print("{0}: {1}, with distance of {2}".format(i, movie_features_selected.index[indices.flatten()[i]], distances.flatten()[i])

    except KeyError:
        print("Movie title not found in the dataset.")

# Example usage
get_movie_recommendations("Toy Story (1995)")
```

```
Recommendations for Toy Story (1995):

1: Toy Story 2 (1999), with distance of 0.4273987396802844
```

```
2: Jurassic Park (1993), with distance of 0.4343631959138433
3: Independence Day (a.k.a. ID4) (1996), with distance of 0.43573830647233425
4: Star Wars: Episode IV - A New Hope (1977), with distance of 0.4426118294200634
5: Forrest Gump (1994), with distance of 0.4529040920598262
```

```
def get_movie_recommendations_with_threshold(movie_title, n_neighbors=6, min_ratings_threshold=10):
    try:
        # Find the index of the given movie title
        movie_index = movie_features_selected.index.get_loc(movie_title)

        # Check if the movie meets the minimum ratings threshold
        if movie_features_selected.iloc[movie_index, :].sum() < min_ratings_threshold:
            print("This movie has too few ratings to provide reliable recommendations.")
            return

        # Query the k-NN model and print recommendations
        get_movie_recommendations(movie_title, n_neighbors)

    except KeyError:
        print("Movie title not found in the dataset.")

# Example usage
get_movie_recommendations_with_threshold("Toy Story (1995)")
```

```
Recommendations for Toy Story (1995):

1: Toy Story 2 (1999), with distance of 0.4273987396802844
2: Jurassic Park (1993), with distance of 0.4343631959138433
3: Independence Day (a.k.a. ID4) (1996), with distance of 0.43573830647233425
4: Star Wars: Episode IV - A New Hope (1977), with distance of 0.4426118294200634
5: Forrest Gump (1994), with distance of 0.4529040920598262
```

```
# Load the data into a Surprise Dataset
reader = Reader(rating_scale=(0.5, 5.0))
data = Dataset.load_from_df(movie_features1[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

```
# train_data_sparse = csr_matrix(train_data)
# test_data_sparse = csr_matrix(test_data)

# # Apply TruncatedSVD to reduce dimensionality
# n_components = 50  # You can adjust the number of components
# svd = TruncatedSVD(n_components=n_components)
# train_data_svd = svd.fit_transform(train_data_sparse)
# test_data_svd = svd.transform(test_data_sparse)

# # Calculate cosine similarity between the transformed test data and the training data
# cosine_sim = cosine_similarity(test_data_svd, train_data_svd)

# # Calculate predicted ratings using cosine similarity
# predicted_ratings = cosine_sim.dot(train_data)

# # Calculate RMSE
# rmse = mean_squared_error(test_data, predicted_ratings, squared=False)
# print("RMSE: {0:.2f}".format(rmse))
```

```
from surprise import SVD
from surprise.model_selection import train_test_split
from surprise import accuracy

# Create Surprise Dataset
reader = Reader(rating_scale=(0.5, 5.0))
data = Dataset.load_from_df(movie_features1[['userId', 'movieId', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

# Train an SVD model on the trainset
svd_model = SVD(n_factors=50, random_state=42)
svd_model.fit(trainset)

# Get the user-item predicted ratings
test_predictions = svd_model.test(testset)
```

```
# Calculate RMSE using the Surprise accuracy module
rmse = accuracy.rmse(test_predictions)
print("RMSE: {:.2f}".format(rmse))
```

```
RMSE: 0.8333
RMSE: 0.83
```

This RMSE is good and means our model is working effeciently

```
# # Attempted to handle cold start, Colab crashes repeatedly


# cold_start_threshold = 5

# genre_features = movie_features1['genres'].str.get_dummies('|')
# content_similarity = cosine_similarity(genre_features)
# # Convert your DataFrame into a Surprise Dataset
# reader = Reader(rating_scale=(0.5, 5.0))
# data = Dataset.load_from_df(movie_features1[['userId', 'movieId', 'rating']], reader)

# # Split the data into train and test sets
# trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

# # Train an SVD model on the trainset
# svd_model = SVD(n_factors=50, random_state=42)
# svd_model.fit(trainset)


# def hybrid_recommend(user_id, movie_id):
#     # Check if user or movie is a cold start case
#     user_history = movie_features1[movie_features1['userId'] == user_id]
#     if len(user_history) <= cold_start_threshold:
#         # Use content-based recommendations
#         content_similarities = content_similarity[movie_id]
#         content_based_scores = content_similarities
#         return content_based_scores

#     # Use collaborative filtering (SVD) predictions
#     cf_prediction = svd_model.predict(user_id, movie_id).est
#     return cf_prediction

# # Example user and movie for recommendation
# user_id = 1
# movie_id = 2

# # Get hybrid recommendation
# hybrid_score = hybrid_recommend(user_id, movie_id)
# print("Hybrid Score:", hybrid_score)
```

Conclusion and Recommendations

In this project, we set out to build a movie recommendation system using collaborative filtering and content-based approaches. We explored a diverse dataset containing movie ratings, genres, and user interactions to create a personalized movie recommendation system. Through data cleaning, exploratory data analysis (EDA), and the implementation of recommendation algorithms, we've gained insights into the movie preferences of users and successfully generated movie recommendations.

Key Findings and Achievements

Data Cleaning and EDA: We started by preprocessing the dataset, removing duplicates, and handling missing values. Exploratory data analysis provided us with valuable insights into the distribution of movie genres, user ratings, and user interactions.

Collaborative Filtering (SVD): Using the Surprise library, we built a collaborative filtering model based on matrix factorization, specifically the Singular Value Decomposition (SVD) algorithm. The model effectively captured user preferences and generated accurate movie recommendations. The calculated Root Mean Square Error (RMSE) of 0.83 indicated the model's reasonable predictive performance.

Recommendations for Improvement

Hybrid Recommendations: While not implemented in this project, combining collaborative filtering and content-based recommendations can enhance the accuracy and coverage of recommendations. Hybrid models address the limitations of each approach, providing more robust