In [1]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from surprise import Reader, Dataset, SVD
from surprise.model_selection import train_test_split

# Set the mode.use_inf_as_na option to True
pd.set_option('mode.use_inf_as_na', True)
```

In [2]:
```python
# Load data with low_memory option
credits = pd.read_csv('credits.csv', low_memory=False)
keywords = pd.read_csv('keywords.csv', low_memory=False)
links_small = pd.read_csv('links_small.csv', low_memory=False)
links = pd.read_csv('links.csv', low_memory=False)
movies_metadata = pd.read_csv('movies_metadata.csv', low_memory=False)
ratings_small = pd.read_csv('ratings_small.csv', low_memory=False)
ratings = pd.read_csv('ratings.csv', low_memory=False)
```

In [4]:
```python
# Handling missing values
movies_metadata = movies_metadata[movies_metadata['id'].notna()]

# Convert 'id' column to numeric, handling errors by setting invalid values
movies_metadata['id'] = pd.to_numeric(movies_metadata['id'], errors='coerce

# Drop rows where 'id' is NaN
movies_metadata = movies_metadata.dropna(subset=['id'])

# Convert 'id' to integer
movies_metadata['id'] = movies_metadata['id'].astype(int)
```

In [5]:
```python
# Handle 'release_date' column
movies_metadata['release_date'] = pd.to_datetime(movies_metadata['release_da

# Merge credits and keywords with movies_metadata on 'id'
movies_metadata = pd.merge(movies_metadata, credits, on='id', how='left')
movies_metadata = pd.merge(movies_metadata, keywords, on='id', how='left')

# Convert 'id' column in links_small and links to integer
links_small['id'] = links_small['movieId'].astype(int)
links['id'] = links['movieId'].astype(int)

# Merge links_small and links with movies_metadata on 'id'
movies_metadata = pd.merge(movies_metadata, links_small, on='id', how='left
movies_metadata = pd.merge(movies_metadata, links, on='id', how='left')

# Convert 'id' column in ratings_small to integer
ratings_small['id'] = ratings_small['movieId'].astype(int)

# Merge ratings_small with movies_metadata on 'id'
movies_metadata = pd.merge(movies_metadata, ratings_small, on='id', how='le

# Drop unnecessary columns
columns_to_drop = ['belongs_to_collection', 'homepage', 'imdb_id', 'poster_p
movies_metadata = movies_metadata.drop(columns=columns_to_drop)

# Display basic information after preprocessing
print("Movies Metadata after preprocessing:")
print(movies_metadata.info())
print("\nSample data from Movies Metadata:")
```

```
print(movies_metadata.head())

# Save the preprocessed data if necessary
# movies_metadata.to_csv('preprocessed_movies_metadata.csv', index=False)

# Convert 'budget' and 'popularity' to numeric, handling errors by setting i
movies_metadata['budget'] = pd.to_numeric(movies_metadata['budget'], errors=
movies_metadata['popularity'] = pd.to_numeric(movies_metadata['popularity'],

# Display data types after conversion
print("Data Types after Conversion:")
print(movies_metadata.dtypes)
```

```
Movies Metadata after preprocessing:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88823 entries, 0 to 88822
Data columns (total 31 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   adult                 88823 non-null  object
 1   budget                88823 non-null  object
 2   genres                88823 non-null  object
 3   id                    88823 non-null  int64
 4   original_language     88812 non-null  object
 5   original_title        88823 non-null  object
 6   overview              87712 non-null  object
 7   popularity            88819 non-null  object
 8   production_companies  88819 non-null  object
 9   production_countries  88819 non-null  object
 10  release_date          88707 non-null  datetime64[ns]
 11  revenue               88819 non-null  float64
 12  runtime               88554 non-null  float64
 13  spoken_languages      88819 non-null  object
 14  status                88737 non-null  object
 15  tagline               50537 non-null  object
 16  title                 88819 non-null  object
 17  vote_average          88819 non-null  float64
 18  vote_count            88819 non-null  float64
 19  cast                  88822 non-null  object
 20  crew                  88822 non-null  object
 21  keywords              88822 non-null  object
 22  movieId_x             45052 non-null  float64
 23  imdbId_x              45052 non-null  float64
 24  tmdbId_x              45038 non-null  float64
 25  movieId_y             49955 non-null  float64
 26  imdbId_y              49955 non-null  float64
 27  tmdbId_y              49919 non-null  float64
 28  userId                45042 non-null  float64
 29  movieId               45042 non-null  float64
 30  rating                45042 non-null  float64
dtypes: datetime64[ns](1), float64(13), int64(1), object(16)
memory usage: 21.0+ MB
None

Sample data from Movies Metadata:
   adult    budget                                             genres     i
d
0  False  30000000  [{'id': 16, 'name': 'Animation'}, {'id': 35, '...    86
2  \
1  False  65000000  [{'id': 12, 'name': 'Adventure'}, {'id': 14, '...   884
4
2  False         0  [{'id': 10749, 'name': 'Romance'}, {'id': 35, ...  1560
2
3  False  16000000  [{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...  3135
7
4  False         0                     [{'id': 35, 'name': 'Comedy'}]  1186
2

  original_language               original_title
0                en                    Toy Story  \
1                en                      Jumanji
2                en             Grumpier Old Men
3                en            Waiting to Exhale
4                en  Father of the Bride Part II

                                            overview popularity
```

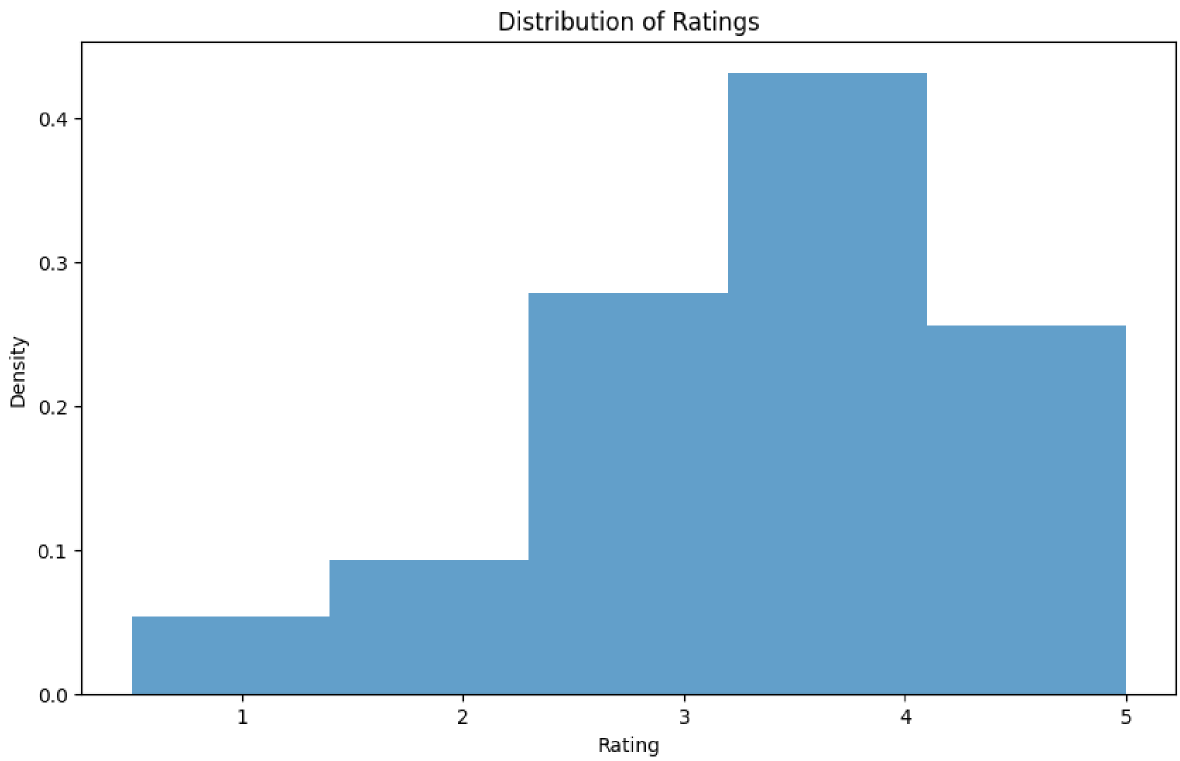
```
0  Led by Woody, Andy's toys live happily in his ...  21.946943  \
1  When siblings Judy and Peter discover an encha...  17.015539
2  A family wedding reignites the ancient feud be...    11.7129
3  Cheated on, mistreated and stepped on, the wom...   3.859495
4  Just when George Banks has recovered from his ...   8.387519

                                production_companies
0     [{'name': 'Pixar Animation Studios', 'id': 3}]  \
1  [{'name': 'TriStar Pictures', 'id': 559}, {'na...
2  [{'name': 'Warner Bros.', 'id': 6194}, {'name'...
3  [{'name': 'Twentieth Century Fox Film Corporat...
4  [{'name': 'Sandollar Productions', 'id': 5842}...

                                production_countries  ...
0  [{'iso_3166_1': 'US', 'name': 'United States o...  ...  \
1  [{'iso_3166_1': 'US', 'name': 'United States o...  ...
2  [{'iso_3166_1': 'US', 'name': 'United States o...  ...
3  [{'iso_3166_1': 'US', 'name': 'United States o...  ...
4  [{'iso_3166_1': 'US', 'name': 'United States o...  ...

                                            keywords  movieId_x  imdbId_x
0  [{'id': 931, 'name': 'jealousy'}, {'id': 4290,...        NaN       NaN
\
1  [{'id': 10090, 'name': 'board game'}, {'id': 1...        NaN       NaN
2  [{'id': 1495, 'name': 'fishing'}, {'id': 12392...        NaN       NaN
3  [{'id': 818, 'name': 'based on novel'}, {'id':...        NaN       NaN
4  [{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...        NaN       NaN

  tmdbId_x movieId_y  imdbId_y tmdbId_y  userId  movieId rating
0      NaN     862.0  116985.0  88224.0     NaN      NaN    NaN
1      NaN    8844.0   78763.0  42164.0     NaN      NaN    NaN
2      NaN       NaN       NaN      NaN     NaN      NaN    NaN
3      NaN       NaN       NaN      NaN     NaN      NaN    NaN
4      NaN       NaN       NaN      NaN     NaN      NaN    NaN

[5 rows x 31 columns]
Data Types after Conversion:
adult                           object
budget                           int64
genres                          object
id                               int64
original_language               object
original_title                  object
overview                        object
popularity                     float64
production_companies            object
production_countries            object
release_date            datetime64[ns]
revenue                        float64
runtime                        float64
spoken_languages                object
status                          object
tagline                         object
title                           object
vote_average                   float64
vote_count                     float64
cast                            object
crew                            object
keywords                        object
movieId_x                      float64
imdbId_x                       float64
tmdbId_x                       float64
movieId_y                      float64
```

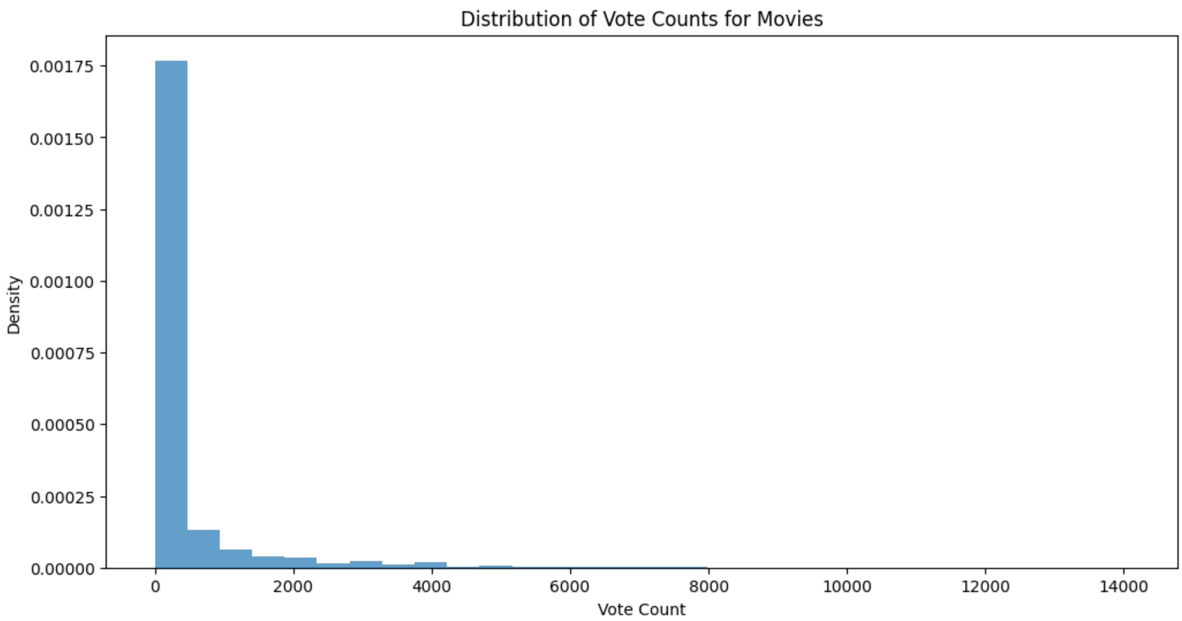
```
imdbId_y                              float64
tmdbId_y                              float64
userId                                float64
movieId                               float64
rating                                float64
dtype: object
```

In [6]:

```
# Basic cleaning: drop rows with NaN in specific columns
columns_to_drop_na = ['popularity', 'budget', 'revenue', 'runtime', 'release
movies_metadata_cleaned = movies_metadata.dropna(subset=columns_to_drop_na)

# Display basic information after basic cleaning
print("\nMovies Metadata after Basic Cleaning:")
print(movies_metadata_cleaned.info())

# Distribution of Ratings
import numpy as np
import matplotlib.pyplot as plt

# Drop NaN values from the 'rating' column
valid_ratings = ratings['rating'].dropna()

# Distribution of ratings in ratings dataset
plt.figure(figsize=(10, 6))
plt.hist(valid_ratings, bins=5, density=True, alpha=0.7)
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Density')
plt.show()

# Popularity of Movies based on vote counts
# Drop NaN values from the 'vote_count' column
valid_vote_counts = movies_metadata['vote_count'].dropna()

# Popularity of movies based on vote counts
plt.figure(figsize=(12, 6))
plt.hist(valid_vote_counts, bins=30, density=True, alpha=0.7)
plt.title('Distribution of Vote Counts for Movies')
plt.xlabel('Vote Count')
plt.ylabel('Density')
plt.show()
```

```
Movies Metadata after Basic Cleaning:
<class 'pandas.core.frame.DataFrame'>
Index: 88453 entries, 0 to 88822
Data columns (total 31 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   adult                 88453 non-null  object
 1   budget                88453 non-null  int64
 2   genres                88453 non-null  object
 3   id                    88453 non-null  int64
 4   original_language     88442 non-null  object
 5   original_title        88453 non-null  object
 6   overview              87609 non-null  object
 7   popularity            88453 non-null  float64
 8   production_companies  88453 non-null  object
 9   production_countries  88453 non-null  object
 10  release_date          88453 non-null  datetime64[ns]
 11  revenue               88453 non-null  float64
 12  runtime               88453 non-null  float64
 13  spoken_languages      88453 non-null  object
 14  status                88376 non-null  object
 15  tagline               50523 non-null  object
 16  title                 88453 non-null  object
 17  vote_average          88453 non-null  float64
 18  vote_count            88453 non-null  float64
 19  cast                  88452 non-null  object
 20  crew                  88452 non-null  object
 21  keywords              88452 non-null  object
 22  movieId_x             45019 non-null  float64
 23  imdbId_x              45019 non-null  float64
 24  tmdbId_x              45005 non-null  float64
 25  movieId_y             49900 non-null  float64
 26  imdbId_y              49900 non-null  float64
 27  tmdbId_y              49864 non-null  float64
 28  userId                45009 non-null  float64
 29  movieId               45009 non-null  float64
 30  rating                45009 non-null  float64
dtypes: datetime64[ns](1), float64(14), int64(2), object(14)
memory usage: 21.6+ MB
None
```

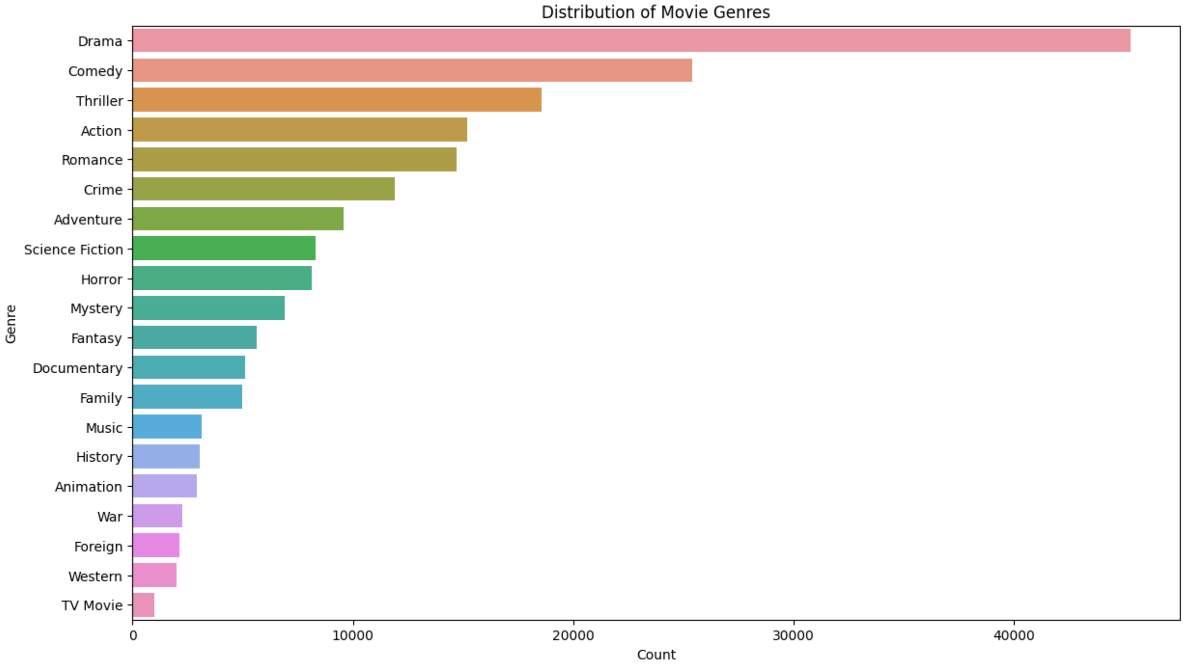




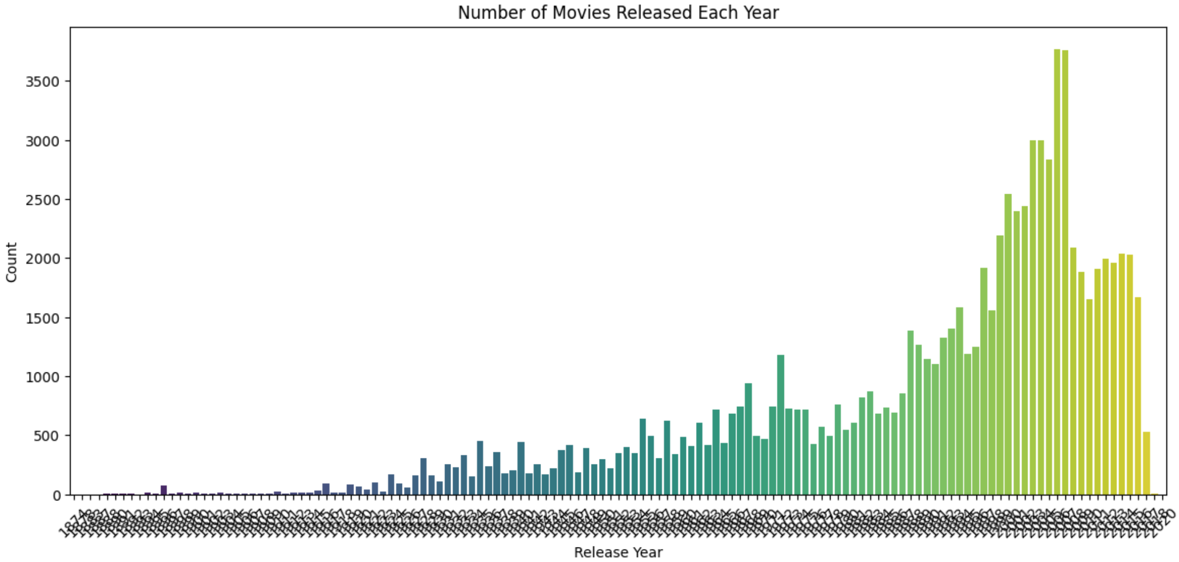
In [7]:

```
# Movie Genres Analysis
import seaborn as sns

# Extract genres from the 'genres' column
genres = []
for genre_list in movies_metadata_cleaned['genres']:
    genres.extend([genre['name'] for genre in eval(genre_list)])

# Plot the distribution of genres
plt.figure(figsize=(14, 8))
sns.countplot(y=genres, order=pd.Series(genres).value_counts().index)
plt.title('Distribution of Movie Genres')
plt.xlabel('Count')
plt.ylabel('Genre')
plt.show()

# Release Date Analysis
# Extract year from the 'release_date' column
```

```
movies_metadata_cleaned['release_year'] = movies_metadata_cleaned['release_d

# Plot the number of movies released each year
plt.figure(figsize=(14, 6))
sns.countplot(x='release_year', data=movies_metadata_cleaned, palette='virid
plt.title('Number of Movies Released Each Year')
plt.xlabel('Release Year')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```




```
/var/folders/8t/8cqkrv9d16vddxxnfykpp_rr0000gn/T/ipykernel_1562/407069115.p
y:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  movies_metadata_cleaned['release_year'] = movies_metadata_cleaned['releas
e_date'].dt.year
```



In [8]:

```
# Feature Engineering
# Handle overview length feature
movies_metadata_cleaned['overview_length'] = movies_metadata_cleaned['overvi
```

```
# Time-based features
movies_metadata_cleaned['release_month'] = movies_metadata_cleaned['release_
movies_metadata_cleaned['release_day'] = movies_metadata_cleaned['release_da
movies_metadata_cleaned['release_dayofweek'] = movies_metadata_cleaned['rele

# Drop unnecessary columns
columns_to_drop = ['id', 'original_title', 'release_date', 'movieId_x', 'mov
movies_metadata_cleaned = movies_metadata_cleaned.drop(columns=columns_to_dr

# Display basic information after feature engineering
print("Columns in movies_metadata_cleaned:")
print(movies_metadata_cleaned.columns)

# Save the cleaned and feature-engineered data if necessary
# movies_metadata_cleaned.to_csv('cleaned_featured_movies_metadata.csv', ind
```

```
Columns in movies_metadata_cleaned:
Index(['adult', 'budget', 'genres', 'original_language', 'overview',
       'popularity', 'production_companies', 'production_countries', 'reven
ue',
       'runtime', 'spoken_languages', 'status', 'tagline', 'title',
       'vote_average', 'vote_count', 'cast', 'crew', 'keywords', 'imdbId_
x',
       'tmdbId_x', 'imdbId_y', 'tmdbId_y', 'userId', 'movieId', 'rating',
       'release_year', 'overview_length', 'release_month', 'release_day',
       'release_dayofweek'],
      dtype='object')
```

```
/var/folders/8t/8cqkrv9d16vddxxnfykpp_rr0000gn/T/ipykernel_1562/983945491.p
y:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  movies_metadata_cleaned['overview_length'] = movies_metadata_cleaned['ove
rview'].apply(lambda x: len(str(x)))
/var/folders/8t/8cqkrv9d16vddxxnfykpp_rr0000gn/T/ipykernel_1562/983945491.p
y:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  movies_metadata_cleaned['release_month'] = movies_metadata_cleaned['relea
se_date'].dt.month
/var/folders/8t/8cqkrv9d16vddxxnfykpp_rr0000gn/T/ipykernel_1562/983945491.p
y:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  movies_metadata_cleaned['release_day'] = movies_metadata_cleaned['release
_date'].dt.day
/var/folders/8t/8cqkrv9d16vddxxnfykpp_rr0000gn/T/ipykernel_1562/983945491.p
y:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  movies_metadata_cleaned['release_dayofweek'] = movies_metadata_cleaned['r
elease_date'].dt.dayofweek
```

In [10]:

```python
# Convert 'budget' and 'popularity' to numeric, handling errors by setting i
movies_metadata_cleaned['budget'] = pd.to_numeric(movies_metadata_cleaned['b
movies_metadata_cleaned['popularity'] = pd.to_numeric(movies_metadata_cleane

# Display data types after conversion
print("Data Types after Conversion:")
print(movies_metadata_cleaned.dtypes)

# Basic cleaning: drop rows with NaN in specific columns
columns_to_drop_na = ['popularity', 'budget', 'revenue', 'runtime', 'vote_av
movies_metadata_cleaned = movies_metadata_cleaned.dropna(subset=columns_to_d

# Display basic information after basic cleaning
print("\nMovies Metadata after Basic Cleaning:")
print(movies_metadata_cleaned.info())
```

```
Data Types after Conversion:
adult                    object
budget                    int64
genres                   object
original_language        object
overview                 object
popularity              float64
production_companies     object
production_countries     object
revenue                 float64
runtime                 float64
spoken_languages         object
status                   object
tagline                  object
title                    object
vote_average            float64
vote_count              float64
cast                     object
crew                     object
keywords                 object
imdbId_x                float64
tmdbId_x                float64
imdbId_y                float64
tmdbId_y                float64
userId                  float64
movieId                 float64
rating                  float64
release_year              int32
overview_length           int64
release_month             int32
release_day               int32
release_dayofweek         int32
dtype: object

Movies Metadata after Basic Cleaning:
<class 'pandas.core.frame.DataFrame'>
Index: 88453 entries, 0 to 88822
Data columns (total 31 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   adult                 88453 non-null  object
 1   budget                88453 non-null  int64
 2   genres                88453 non-null  object
 3   original_language     88442 non-null  object
 4   overview              87609 non-null  object
 5   popularity            88453 non-null  float64
 6   production_companies  88453 non-null  object
 7   production_countries  88453 non-null  object
 8   revenue               88453 non-null  float64
 9   runtime               88453 non-null  float64
 10  spoken_languages      88453 non-null  object
 11  status                88376 non-null  object
 12  tagline               50523 non-null  object
 13  title                 88453 non-null  object
 14  vote_average          88453 non-null  float64
 15  vote_count            88453 non-null  float64
 16  cast                  88452 non-null  object
 17  crew                  88452 non-null  object
 18  keywords              88452 non-null  object
 19  imdbId_x              45019 non-null  float64
 20  tmdbId_x              45005 non-null  float64
 21  imdbId_y              49900 non-null  float64
 22  tmdbId_y              49864 non-null  float64
```

```
 23  userId                 45009 non-null  float64
 24  movieId                45009 non-null  float64
 25  rating                 45009 non-null  float64
 26  release_year           88453 non-null  int32
 27  overview_length        88453 non-null  int64
 28  release_month          88453 non-null  int32
 29  release_day            88453 non-null  int32
 30  release_dayofweek      88453 non-null  int32
dtypes: float64(12), int32(4), int64(2), object(13)
memory usage: 20.2+ MB
None
```

In [11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.metrics.pairwise import cosine_similarity

# Sample a smaller subset of the data for testing
movies_subset = movies_metadata_cleaned.sample(n=45000, random_state=42)

# Combine relevant features (overview and genres) into a single column
movies_subset['combined_features'] = movies_subset['overview'].fillna('') +

# Use TF-IDF Vectorizer on the subset
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(movies_subset['combined_featu

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Function to get movie recommendations
def get_recommendations(title, cosine_sim, movies):
    # Get the index of the movie that matches the title
    idx = movies.index[movies['title'] == title].tolist()

    if not idx:
        print(f"Movie '{title}' not found in the dataset.")
        return []

    idx = idx[0]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return movies['title'].iloc[movie_indices]
```

In [12]:
```
# Test the recommendation system with the chosen movie title
movie_title = 'Urban Cowboy'
recommendations = get_recommendations(movie_title, cosine_sim, movies_subse
print(f"Recommendations for '{movie_title}':")
print(recommendations)
```

```
Recommendations for 'Urban Cowboy':
86669                       Food and Shelter
62813            The Undertaker and His Pals
88032      The Gerber Syndrome: Il Contagio
52397                                 My Boy
51827                    Cooking With Stella
67812                             Rent-a-Cat
50926                                Placido
80546                             Son of Sam
78712                       Kyaa Kool Hai Hum
70614                         Video Violence
Name: title, dtype: object
```

In [13]:
```
# Test the recommendation system with the chosen movie title
movie_title = 'A Tale of Two Sisters'
recommendations = get_recommendations(movie_title, cosine_sim, movies_subse
print(f"Recommendations for '{movie_title}':")
print(recommendations)
```

```
Recommendations for 'A Tale of Two Sisters':
40659         Mother India
40661         Mother India
40657         Mother India
79439            Jolly LLB
72868             Brothers
78424          Un plus une
78425          Un plus une
56651      Paan Singh Tomar
77332        The Old Devil
59598                D-Day
Name: title, dtype: object
```

In [14]:
```
# Display the columns in movies_subset
print("Columns in movies_subset:")
print(movies_subset.columns)
```

```
Columns in movies_subset:
Index(['adult', 'budget', 'genres', 'original_language', 'overview',
       'popularity', 'production_companies', 'production_countries', 'reven
ue',
       'runtime', 'spoken_languages', 'status', 'tagline', 'title',
       'vote_average', 'vote_count', 'cast', 'crew', 'keywords', 'imdbId_
x',
       'tmdbId_x', 'imdbId_y', 'tmdbId_y', 'userId', 'movieId', 'rating',
       'release_year', 'overview_length', 'release_month', 'release_day',
       'release_dayofweek', 'combined_features'],
      dtype='object')
```

In [15]:
```
print(movies_subset['title'].unique())
```

```
['Urban Cowboy' "Mo' Better Blues" 'Crisis' ...
 'Wanda Sykes: Sick and Tired' 'Smoorverliefd' 'Love Me Deadly']
```

In [16]:
```
print(movies_subset['title'])
```

```
25680                             Urban Cowboy
16797                         Mo' Better Blues
63749                                   Crisis
81825                              Makeup Room
5769                                    Psycho
                          ...
23606                          Monsoon Wedding
27718    Terminator 3: Rise of the Machines
70350                 Mere Brother Ki Dulhan
32402                                Nostalgia
27320                              Whale Rider
Name: title, Length: 45000, dtype: object
```

In [17]:
```python
# Find the index of the chosen movie title in the subset
idx = movies_subset[movies_subset['title'] == movie_title].index

# Print the index
print(f"Index of '{movie_title}': {idx}")
```

```
Index of 'A Tale of Two Sisters': Index([32793, 32794, 32792], dtype='int6
4')
```

In [19]:
```python
def interactive_recommendation_system():
    # Ask the user to enter a movie title
    user_input = input("Enter a movie title: ")

    # Check if the entered movie title is in the dataset
    if user_input not in movies_subset['title'].values:
        print(f"Movie '{user_input}' not found in the dataset.")
        return

    # Get recommendations based on the entered movie title
    recommendations = get_recommendations(user_input, cosine_sim, movies_sul

    # Print the recommendations
    print(f"\nRecommendations for '{user_input}':")
    print(recommendations)

# Test the interactive recommendation system
interactive_recommendation_system()
```

```
Recommendations for 'Amistad':
58334    Manuel on the Island of Wonders
61891                     Survival Island
85227          How to Draw a Perfect Circle
57781                        Back to Stay
87139                         Bella Mafia
56698                               After
40725                  The White Countess
86261                            Afterlov
84444                        Candy Razors
7790                          Guantanamera
Name: title, dtype: object
```

In [20]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

# Sample a smaller subset of the data for testing
movies_subset = movies_metadata_cleaned.sample(n=26000, random_state=42)

# Check if columns exist and convert lists to strings
columns_to_combine = ['overview', 'genres', 'director', 'cast']
for column in columns_to_combine:
```

```
    if column in movies_subset.columns:
        movies_subset[column] = movies_subset[column].apply(lambda x: ' '.jo

# Combine relevant text features into a single column
movies_subset['combined_features'] = movies_subset.apply(lambda row: ' '.joi

# Use TF-IDF Vectorizer on the combined features
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(movies_subset['combined_featu

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Function to get movie recommendations with enhanced features
def get_enhanced_recommendations(title, cosine_sim, movies):
    # Get the index of the movie that matches the title
    idx = movies.index[movies['title'] == title].tolist()

    if not idx:
        print(f"Movie '{title}' not found in the dataset.")
        return []

    idx = idx[0]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return movies['title'].iloc[movie_indices]

# Test the enhanced recommendation system with the chosen movie title
movie_title = 'Urban Cowboy'
enhanced_recommendations = get_enhanced_recommendations(movie_title, cosine_

# Print the enhanced recommendations
print(f"\nEnhanced Recommendations for '{movie_title}':")
print(enhanced_recommendations)
```

```
Enhanced Recommendations for 'Urban Cowboy':
14363    Tales of Terror
8828             Titanic
8824             Titanic
8945             Titanic
8890             Titanic
8861             Titanic
8920             Titanic
8839             Titanic
8905             Titanic
8829             Titanic
Name: title, dtype: object
```

In [21]:

```
from lightfm import LightFM
from lightfm.evaluation import precision_at_k
from scipy.sparse import coo_matrix
```

```
from sklearn.model_selection import train_test_split

# Sample a smaller subset of the ratings data
ratings_subset = ratings.sample(frac=0.1, random_state=42)

# Split the subset into training and testing sets
train_data_subset, test_data_subset = train_test_split(ratings_subset, test_

# Create a user-item interaction matrix for the subset
interaction_matrix_subset = coo_matrix((train_data_subset['rating'], (train_

# Initialize the model
model = LightFM(loss='warp')

# Train the model on the subset
model.fit(interaction_matrix_subset, epochs=30, num_threads=2)

# Recommend items for a user
user_id = 1  # Replace with the user ID for whom you want to make recommenda
n_items = interaction_matrix_subset.shape[1]

# Predict scores for all items for the given user
scores = model.predict(user_id, list(range(n_items)))

# Get recommended movie indices
top_items = np.argsort(-scores)

# Print top recommended movie titles
top_movie_titles = movies_metadata_cleaned.loc[movies_metadata_cleaned['movi
print("Top Recommended Movies:")
print(top_movie_titles)
```

```
/Users/prudhvi/opt/anaconda3/lib/python3.9/site-packages/lightfm/_lightfm_f
ast.py:9: UserWarning: LightFM was compiled without OpenMP support. Only a
single thread will be used.
  warnings.warn(
Top Recommended Movies:
955                      Once Were Warriors
956                      Once Were Warriors
957                      Once Were Warriors
958                      Once Were Warriors
959                      Once Were Warriors
                                ...
28029    Terminator 3: Rise of the Machines
28030    Terminator 3: Rise of the Machines
28031    Terminator 3: Rise of the Machines
28032    Terminator 3: Rise of the Machines
28033    Terminator 3: Rise of the Machines
Name: title, Length: 1872, dtype: object
```

In [22]:
```
# Print top recommended movie titles
unique_top_items = np.unique(top_items)  # Get unique movie indices
top_movie_titles = movies_metadata_cleaned.loc[movies_metadata_cleaned['movi
print("Top Recommended Movies:")
print(top_movie_titles)
```

```
Top Recommended Movies:
['Four Rooms' 'Judgment Night' 'Ariel' 'Shadows in Paradise']
```

In [23]:
```
from surprise import Reader, Dataset, NMF
from surprise.model_selection import train_test_split

# Sample a smaller subset of the ratings data for quicker results
```

```
ratings_subset = ratings.sample(n=10000, random_state=42)

# Load the ratings data into Surprise format
reader = Reader()
data = Dataset.load_from_df(ratings_subset[['userId', 'movieId', 'rating']],

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

# Initialize the NMF model with fewer factors
nmf_model_surprise = NMF(n_factors=5, random_state=42)

# Fit the model on the training set
nmf_model_surprise.fit(trainset)

# Choose a user ID for recommendations (replace 1 with the desired user ID)
user_id = 1

# Get the list of all movie IDs
all_movie_ids = ratings_subset['movieId'].unique()

# Predict ratings for the chosen user and all movies
user_ratings = [-nmf_model_surprise.predict(user_id, movie_id).est for movie

# Get top recommended movie indices
top_movie_indices_nmf_surprise = np.argsort(user_ratings)

# Print top recommended movie titles
top_movie_titles_nmf_surprise = movies_metadata_cleaned.loc[movies_metadata_
print("Top Recommended Movies (NMF - Surprise):")
print(top_movie_titles_nmf_surprise)
```

```
Top Recommended Movies (NMF - Surprise):
29305    Holy Matrimony
29306    Holy Matrimony
29307    Holy Matrimony
29308    Holy Matrimony
29309    Holy Matrimony
29310    Holy Matrimony
29311    Holy Matrimony
29312    Holy Matrimony
29313    Holy Matrimony
29314    Holy Matrimony
29315    Holy Matrimony
Name: title, dtype: object
```

In [33]:

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

# Sample a smaller subset of your ratings data
ratings_subset = ratings.sample(frac=0.1, random_state=42)

# Assuming you have loaded and preprocessed your ratings data
reader = Reader()
data = Dataset.load_from_df(ratings_subset[['userId', 'movieId', 'rating']],
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

# Get raw ratings from the testset
test_raw_ratings = [(uid, iid, r_ui_trans) for (uid, iid, r_ui_trans) in tes

# Convert raw ratings to pandas DataFrame if needed
test_data = pd.DataFrame(test_raw_ratings, columns=['userId', 'movieId', 'ra
```

```
# Now you can use the common_movie_ids calculation
common_movie_ids = set(train_data['movieId']).intersection(set(test_data['mo
print("Number of common movie IDs:", len(common_movie_ids))
```

```
Number of common movie IDs: 14595
```

In [37]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Concatenate,

def create_ncf_model(n_users, n_movies):
    # User embedding
    user_input = Input(shape=(1,))
    user_embedding = Embedding(n_users, 50)(user_input)
    user_embedding = Flatten()(user_embedding)

    # Movie embedding
    movie_input = Input(shape=(1,))
    movie_embedding = Embedding(n_movies, 50)(movie_input)
    movie_embedding = Flatten()(movie_embedding)

    # Concatenate user and movie embeddings
    concatenated = Concatenate()([user_embedding, movie_embedding])

    # Dense layers
    dense1 = Dense(128, activation='relu')(concatenated)
    dense2 = Dense(64, activation='relu')(dense1)

    # Output layer
    output_layer = Dense(1, activation='sigmoid')(dense2)

    # Model
    model = Model(inputs=[user_input, movie_input], outputs=output_layer)

    return model

# Usage example:
n_users = 1000  # Replace with the actual number of users
n_movies_total = 5000  # Replace with the actual total number of movies
ncf_model = create_ncf_model(n_users, n_movies_total)
```

```
2023-11-28 02:32:46.763259: I tensorflow/core/platform/cpu_feature_guard.c
c:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  AVX2 AVX512F AVX512_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-11-28 02:33:05.075611: I tensorflow/core/platform/cpu_feature_guard.c
c:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  AVX2 AVX512F AVX512_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

In [38]:
```
common_user_ids = set(train_data['userId']).intersection(set(test_data['use
print("Number of common user IDs:", len(common_user_ids))
```

```
Number of common user IDs: 130936
```

In [39]:
```
# Get the number of unique users in the entire dataset
n_users_total = len(ratings['userId'].unique())

# Create the NCF model with the correct number of unique users and movies
ncf_model = create_ncf_model(n_users_total, n_movies_total)
```

In [40]:
```
print(train_data.dtypes)
```

```
userId        int64
movieId       int64
rating      float64
dtype: object
```

In [41]:
```
print([train_data['userId'], train_data['movieId']])
```

```
[0            9279
1            9279
2            9279
3            9279
4            9279
            ...
2081938     65414
2081939    240848
2081940    211106
2081941    255627
2081942    169931
Name: userId, Length: 2081943, dtype: int64, 0          2187
1          3176
2          6707
3           367
4          7757
           ...
2081938    1100
2081939    1035
2081940    3062
2081941    7153
2081942     709
Name: movieId, Length: 2081943, dtype: int64]
```

In [42]:
```
max_user_id = train_data['userId'].max()
max_movie_id = train_data['movieId'].max()

print("Max User ID:", max_user_id)
print("Max Movie ID:", max_movie_id)
```

```
Max User ID: 270896
Max Movie ID: 176219
```

In [43]:
```
def map_ids(ids, max_id):
    return ids - (max_id - n_users) if max_id >= n_users else ids
```

In [44]:
```
train_data['userId'] = map_ids(train_data['userId'], max_user_id)
train_data['movieId'] = map_ids(train_data['movieId'], max_movie_id)
```

In [45]:
```
max_user_id = train_data['userId'].max()
max_movie_id = train_data['movieId'].max()

print("Max User ID (After Adjustment):", max_user_id)
print("Max Movie ID (After Adjustment):", max_movie_id)
```

```
Max User ID (After Adjustment): 1000
Max Movie ID (After Adjustment): 1000
```

In [46]:
```
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Concatenate,
```

```
# Assuming you have loaded and preprocessed your ratings data
# Let's assume your DataFrame is named 'ratings' and contains columns 'userI

# Split the data into training and testing sets
train_data, test_data = train_test_split(ratings, test_size=0.2, random_sta

# Create a mapping of unique labels to integers for users
user_label_mapping = {label: i for i, label in enumerate(train_data['userId

# Apply the mapping to the 'userId' column in both training and test data
train_data['userId_encoded'] = train_data['userId'].map(user_label_mapping)
test_data['userId_encoded'] = test_data['userId'].map(user_label_mapping).f

# Repeat the same process for movies
movie_label_mapping = {label: i for i, label in enumerate(train_data['movieI
train_data['movieId_encoded'] = train_data['movieId'].map(movie_label_mappin
test_data['movieId_encoded'] = test_data['movieId'].map(movie_label_mapping)

# Get the number of unique users and movies in the training data
n_users = len(train_data['userId_encoded'].unique())
n_movies = len(train_data['movieId_encoded'].unique())

# Specify the embedding dimension
embedding_dim = 10

# Create the NCF model
def create_ncf_model(n_users, n_movies, embedding_dim=10):
    user_input = Input(shape=(1,), name='user_input')
    movie_input = Input(shape=(1,), name='movie_input')

    user_embedding = Embedding(input_dim=n_users, output_dim=embedding_dim,
    movie_embedding = Embedding(input_dim=n_movies, output_dim=embedding_dim

    user_flatten = Flatten()(user_embedding)
    movie_flatten = Flatten()(movie_embedding)

    # Concatenate user and movie embeddings
    concat = Concatenate()([user_flatten, movie_flatten])

    # Neural Collaborative Filtering layers
    dense1 = Dense(64, activation='relu')(concat)
    dense2 = Dense(32, activation='relu')(dense1)
    output = Dense(1, activation='linear')(dense2)

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

    return model

# Recreate the model with the specified embedding dimension
ncf_model = create_ncf_model(n_users, n_movies, embedding_dim)

# Compile the model
ncf_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae

# Train the model
history = ncf_model.fit(
    [train_data['userId_encoded'], train_data['movieId_encoded']],
    train_data['rating'],
    epochs=5,
    batch_size=64,
    validation_split=0.1
)
```

```
# Handle unknown labels in test data
test_data['userId_encoded'] = test_data['userId'].apply(lambda x: user_label
test_data['movieId_encoded'] = test_data['movieId'].apply(lambda x: movie_la

# Remove rows with unknown labels
test_data = test_data[(test_data['userId_encoded'] != -1) & (test_data['movi

# Evaluate the model
test_loss = ncf_model.evaluate([test_data['userId_encoded'], test_data['movi
print(f"Test Loss: {test_loss}")

# Make predictions
predictions = ncf_model.predict([test_data['userId_encoded'], test_data['mov

# Example: Print the predicted rating for the first test sample
print(f"Actual Rating: {test_data['rating'].iloc[0]}, Predicted Rating: {pre
```

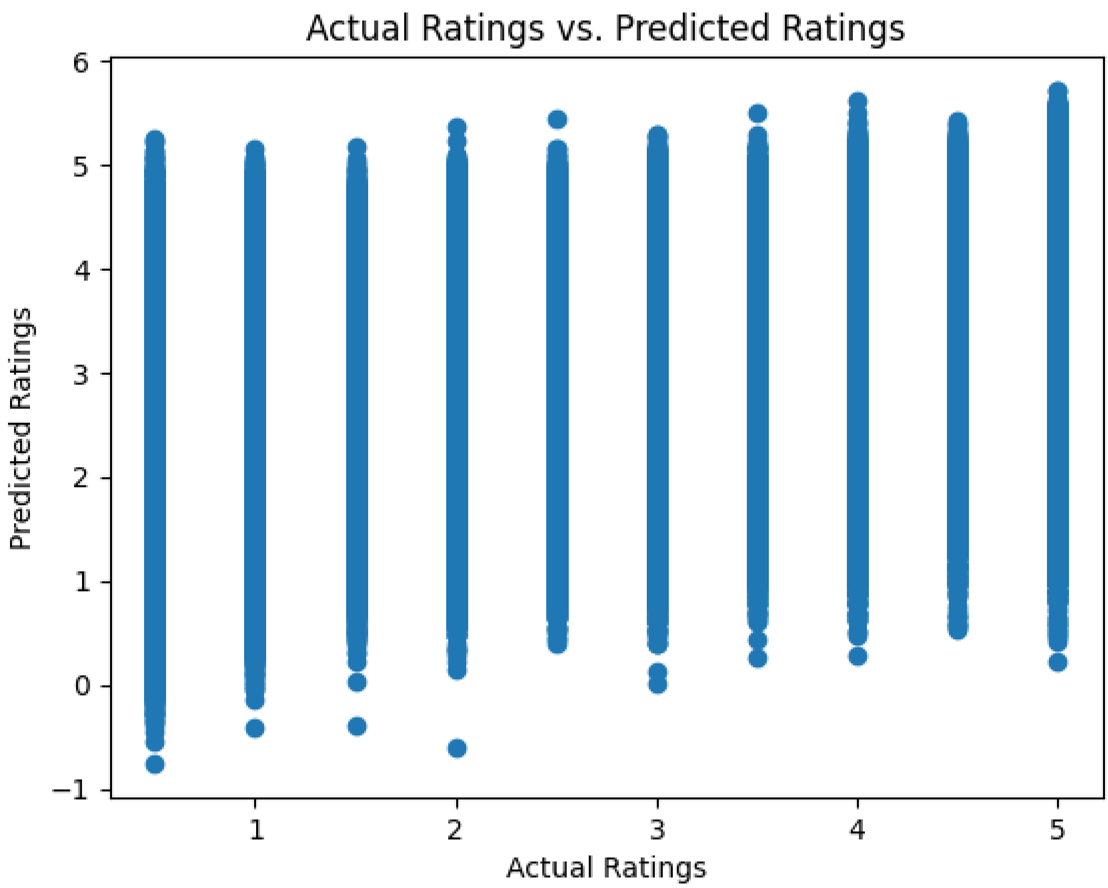
```
Epoch 1/5
292774/292774 [==============================] - 7846s 27ms/step - loss: 0.
7661 - mae: 0.6669 - val_loss: 0.7235 - val_mae: 0.6462
Epoch 2/5
292774/292774 [==============================] - 8446s 29ms/step - loss: 0.
6980 - mae: 0.6324 - val_loss: 0.7019 - val_mae: 0.6376
Epoch 3/5
292774/292774 [==============================] - 8846s 30ms/step - loss: 0.
6684 - mae: 0.6165 - val_loss: 0.6908 - val_mae: 0.6251
Epoch 4/5
292774/292774 [==============================] - 9118s 31ms/step - loss: 0.
6483 - mae: 0.6057 - val_loss: 0.6825 - val_mae: 0.6215
Epoch 5/5
292774/292774 [==============================] - 9122s 31ms/step - loss: 0.
6355 - mae: 0.5985 - val_loss: 0.6780 - val_mae: 0.6204
162543/162543 [==============================] - 282s 2ms/step - loss: 0.67
81 - mae: 0.6208
Test Loss: [0.6780929565429688, 0.620798647403717]
162543/162543 [==============================] - 259s 2ms/step
Actual Rating: 4.0, Predicted Rating: 4.293751239776611
```

In [47]:

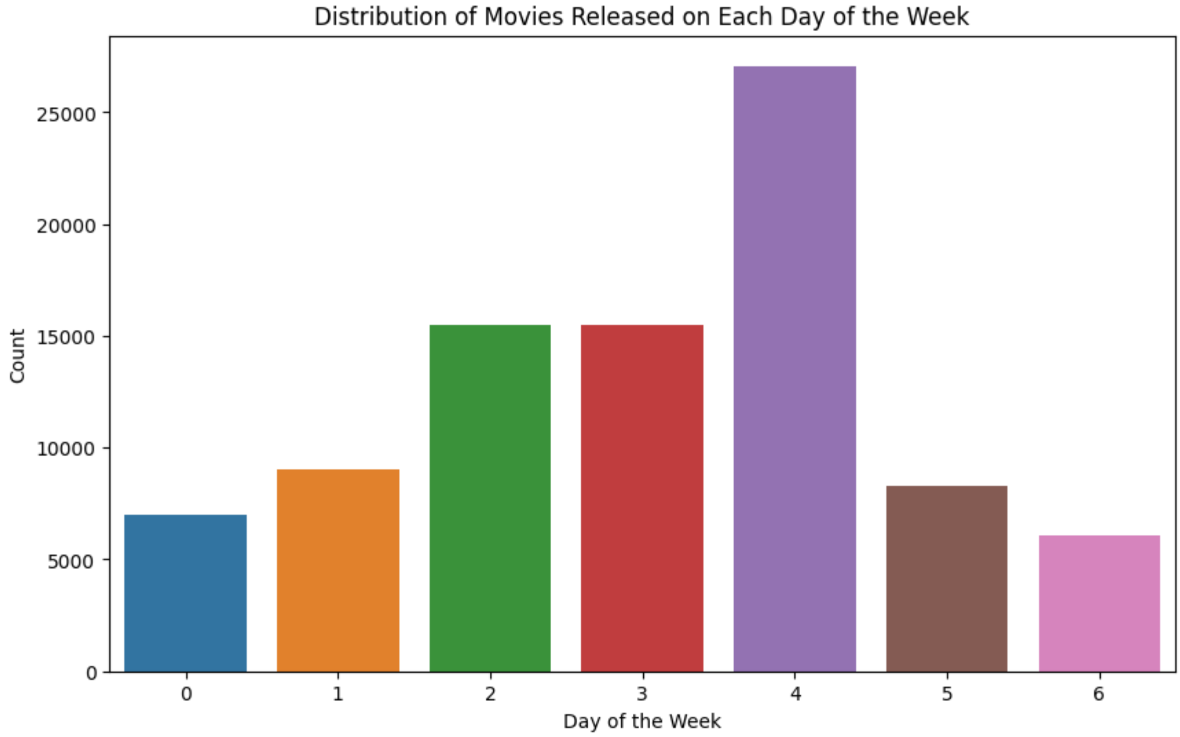
```
# Evaluate the model
test_loss = ncf_model.evaluate([test_data['userId_encoded'], test_data['movi
print(f"Test Loss: {test_loss}")

# Make predictions
predictions = ncf_model.predict([test_data['userId_encoded'], test_data['mov

# Calculate additional evaluation metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(test_data['rating'], predictions)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(test_data['rating'], predictions)
print(f"Mean Absolute Error (MAE): {mae}")

# Calculate R-squared
r2 = r2_score(test_data['rating'], predictions)
print(f"R-squared: {r2}")
```

```
162543/162543 [==============================] - 283s 2ms/step - loss: 0.67
81 - mae: 0.6208
Test Loss: [0.6780929565429688, 0.620798647403717]
162543/162543 [==============================] - 256s 2ms/step
Mean Squared Error (MSE): 0.6780859808353811
Mean Absolute Error (MAE): 0.620801268477189
R-squared: 0.4025479033376498
```

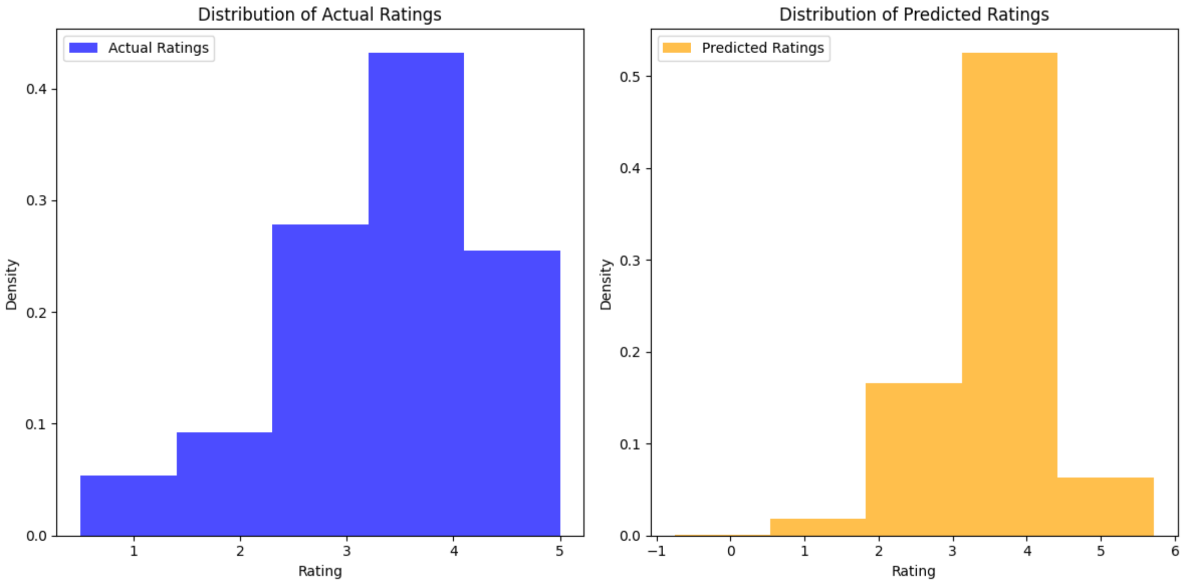
In [48]:
```
# Visualize the predictions vs. actual ratings
import matplotlib.pyplot as plt

plt.scatter(test_data['rating'], predictions)
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.title('Actual Ratings vs. Predicted Ratings')
plt.show()
```



In [49]:
```
#Release Day Analysis:
plt.figure(figsize=(10, 6))
sns.countplot(x='release_dayofweek', data=movies_metadata_cleaned)
plt.title('Distribution of Movies Released on Each Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Count')
plt.show()
```



In [50]:
```
plt.figure(figsize=(12, 6))

# Plot actual ratings distribution
plt.subplot(1, 2, 1)
plt.hist(test_data['rating'], bins=5, density=True, alpha=0.7, color='blue',
plt.title('Distribution of Actual Ratings')
plt.xlabel('Rating')
plt.ylabel('Density')
plt.legend()

# Plot predicted ratings distribution
plt.subplot(1, 2, 2)
plt.hist(predictions, bins=5, density=True, alpha=0.7, color='orange', labe
plt.title('Distribution of Predicted Ratings')
plt.xlabel('Rating')
plt.ylabel('Density')
plt.legend()

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


Distribution of Actual Ratings
Actual Ratings
Density
Rating
Distribution of Predicted Ratings
Predicted Ratings
Density
Rating

In [ ]: