Assignment 2: Recommender Systems

Exploratory Data Analysis

Start with EDA to better understand our dataset and to use appropriate recommender systems.

Imports & Data Loading

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

user_reviews = pd.read_csv("data/movie_reviews/user_reviews.csv")
movie_genres = pd.read_csv("data/movie_reviews/movie_genres.csv")
```

Basic Info

In [3]:

```
print("User Reviews Dataset:")
print("---------------------")
print(user_reviews.info())

print("\nMovie Genres Dataset:")
print("---------------------")
print(movie_genres.info())

print("---------------------")
print(f"\nUser Reviews Shape: {user_reviews.shape}")
print(f"Movie Genres Shape: {movie_genres.shape}")
```

```
User Reviews Dataset:
---------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Columns: 2002 entries, Unnamed: 0 to Hey Arnold! The Movie
dtypes: float64(2000), int64(1), object(1)
memory usage: 9.2+ MB
None

Movie Genres Dataset:
---------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 27 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Unnamed: 0         2000 non-null   int64 
 1   movie_title        2000 non-null   object
 2   genre_action       2000 non-null   int64 
 3   genre_adventure    2000 non-null   int64 
 4   genre_animation    2000 non-null   int64 
 5   genre_biography    2000 non-null   int64 
 6   genre_comedy       2000 non-null   int64 
 7   genre_crime        2000 non-null   int64 
 8   genre_documentary  2000 non-null   int64 
 9   genre_drama        2000 non-null   int64 
 10  genre_family       2000 non-null   int64 
 11  genre_fantasy      2000 non-null   int64 
 12  genre_film-noir    2000 non-null   int64 
 13  genre_history      2000 non-null   int64 
 14  genre_horror       2000 non-null   int64 
 15  genre_music        2000 non-null   int64 
 16  genre_musical      2000 non-null   int64 
 17  genre_mystery      2000 non-null   int64 
 18  genre_news         2000 non-null   int64 
 19  genre_reality-tv   2000 non-null   int64 
 20  genre_romance      2000 non-null   int64 
 21  genre_sci-fi       2000 non-null   int64 
 22  genre_short        2000 non-null   int64 
 23  genre_sport        2000 non-null   int64 
 24  genre_thriller     2000 non-null   int64 
 25  genre_war          2000 non-null   int64 
 26  genre_western      2000 non-null   int64 
dtypes: int64(26), object(1)
memory usage: 422.0+ KB
None
---------------------

User Reviews Shape: (600, 2002)
Movie Genres Shape: (2000, 27)
```

Remove Unecessary Data

In [4]:
```
user_reviews.drop(columns=['Unnamed: 0'], inplace=True)
movie_genres.drop(columns=['Unnamed: 0'], inplace=True)
```

In [5]:
```
user_reviews.set_index('User', inplace=True)
movie_genres.set_index('movie_title', inplace=True)
```

In [6]:
```
user_reviews.head(5)
```

Out[6]:

User	The Net	Happily N'Ever After	Tomorrowland	American Hero	Das Boot	Final Destination 3	Licence to Kill	The Hundred-Foot Journey	The Matrix	Creature	...	The Martian	Micmacs	Solomon and Sheba	In the Company of Men	Silent House	Big Fish	Get Real	Trading Places	DOA: Dead or Alive	Ar
Vincent	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Edgar	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Addilyn	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Marlee	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Javier	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 2000 columns

In [7]:
```
movie_genres.head(5)
```

Out[7]:

movie_title	genre_action	genre_adventure	genre_animation	genre_biography	genre_comedy	genre_crime	genre_documentary	genre_drama	genre_family	genre_fantasy	...	genre_m
The Net	1	0	0	0	0	1	0	1	0	0	...	
Happily N'Ever After	0	1	1	0	1	0	0	0	1	1	...	
Tomorrowland	1	1	0	0	0	0	0	0	1	0	...	
American Hero	1	0	0	0	1	0	0	1	0	0	...	
Das Boot	0	1	0	0	0	0	0	1	0	0	...	

5 rows × 25 columns

In [8]:
```
print(f"\nUser Reviews Shape: {user_reviews.shape}")
print(f"Movie Genres Shape: {movie_genres.shape}")
```

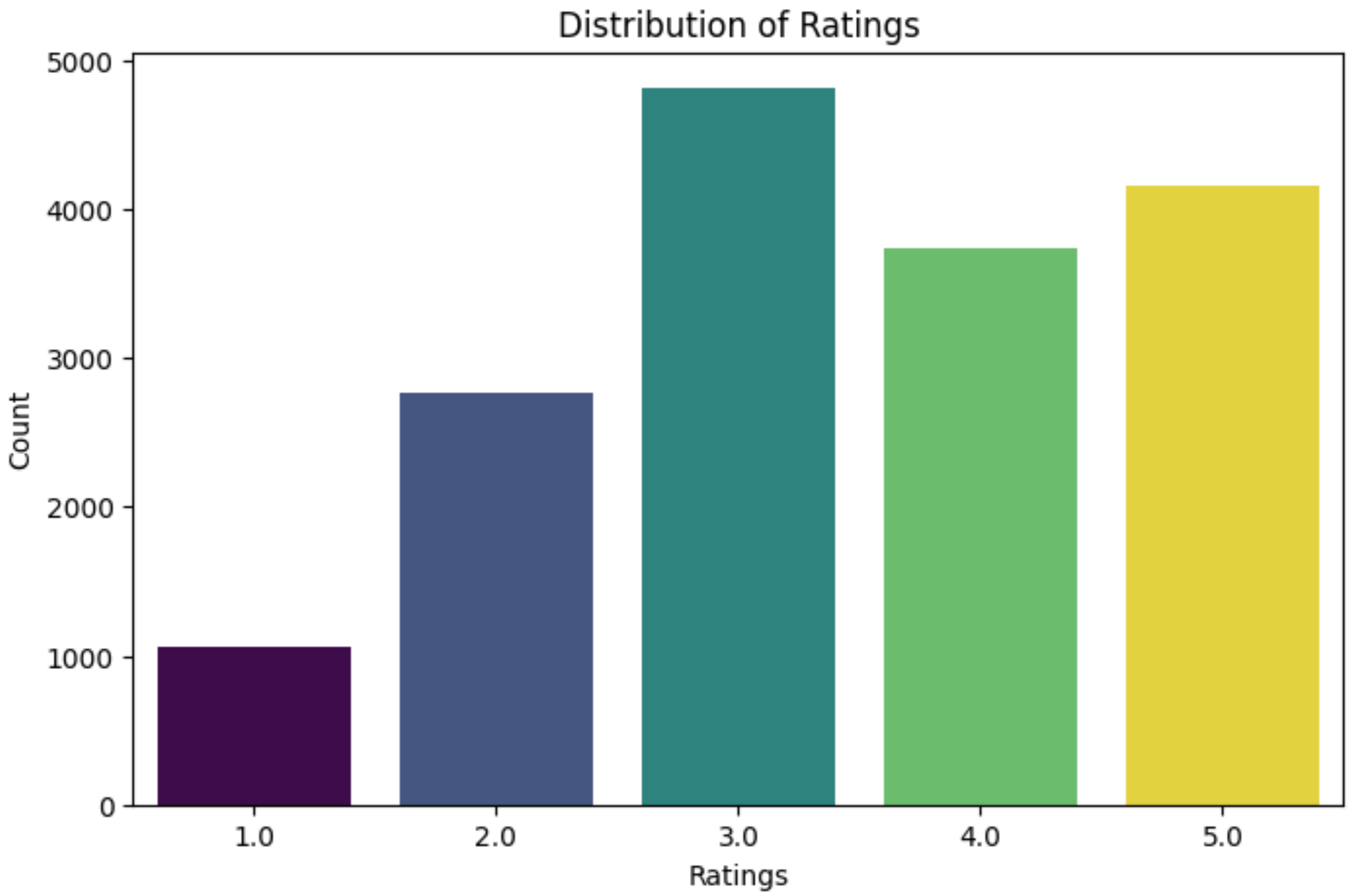
```
User Reviews Shape: (600, 2000)
Movie Genres Shape: (2000, 25)
```

Rating Distribution Analysis

In [9]:
```
num_users, num_movies = user_reviews.shape
print(f"Total Users: {num_users}, Total Movies: {num_movies}, Total Genres: {movie_genres.shape[1]}")

user_reviews_replaced = user_reviews.replace(0, np.nan) # replace 0s with NaN just for the plot

rating_counts = user_reviews_replaced.stack().value_counts().sort_index()
print("\nRating Distribution:\n", rating_counts)
```

```python
plt.figure(figsize=(8, 5))
sns.barplot(x=rating_counts.index, y=rating_counts.values, palette="viridis", legend=False, hue=rating_counts.index)
plt.xlabel("Ratings")
plt.ylabel("Count")
plt.title("Distribution of Ratings")
plt.show()
```

```
Total Users: 600, Total Movies: 2000, Total Genres: 25

Rating Distribution:
 1.0    1058
2.0    2763
3.0    4812
4.0    3732
5.0    4160
Name: count, dtype: int64
```

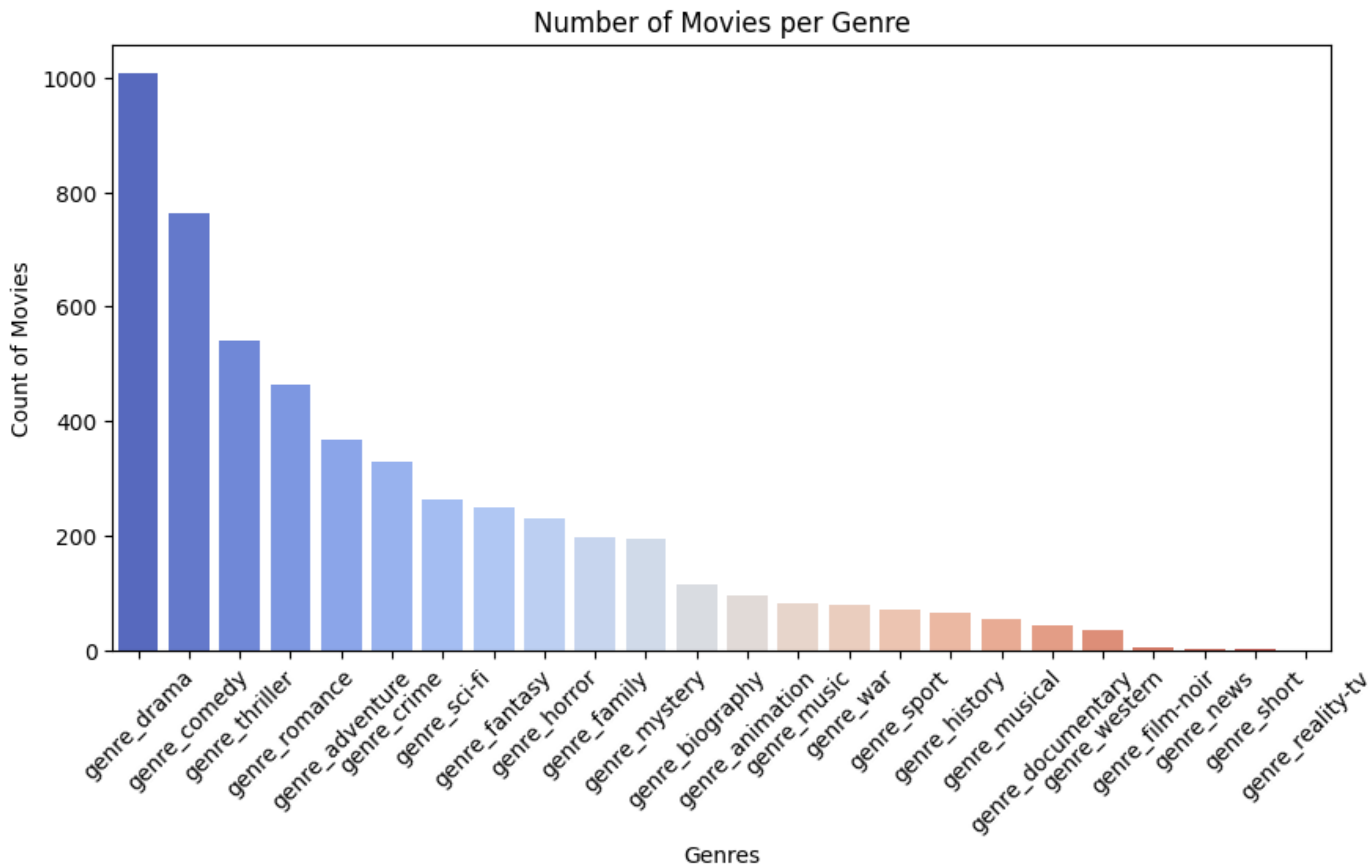


so we see here that more users are giving higher ratings with the mode being 3.0

Genre Anaylsis

In [10]:
```
genre_counts = movie_genres.iloc[:, 1:].sum().sort_values(ascending=False)  # summing binary genre indicators
print("\nMovie Genre Distribution:\n", genre_counts)

plt.figure(figsize=(10, 5))
sns.barplot(x=genre_counts.index, y=genre_counts.values, palette="coolwarm", legend=False, hue=genre_counts.index)
plt.xticks(rotation=45) # better readability
plt.xlabel("Genres")
plt.ylabel("Count of Movies")
plt.title("Number of Movies per Genre")
plt.show()
```

```
Movie Genre Distribution:
 genre_drama          1007
genre_comedy          763
genre_thriller        541
genre_romance         464
genre_adventure       368
genre_crime           330
genre_sci-fi          263
genre_fantasy         251
genre_horror          230
genre_family          198
genre_mystery         195
genre_biography       114
genre_animation        95
genre_music            81
genre_war              80
genre_sport            70
genre_history          67
genre_musical          55
genre_documentary      45
genre_western          36
genre_film-noir         4
genre_news              2
genre_short             2
genre_reality-tv        1
dtype: int64
```



we see here that most movies in the list are tagged drama, but note that a single movie may have multiple genre tags

Sparsity Check for SVD in Collab Filtering

In [11]:

```
num_missing = user_reviews.isin([0]).sum().sum()
total_cells = num_users * num_movies
sparsity = (num_missing / total_cells) * 100
print(f"\nDataset Sparsity: {sparsity}%")
```

```
Dataset Sparsity: 98.62291666666667%
```

we see here that the dataset is quite sparse, with about 98.6% of the dataset being empty (unrated), which leads us to believe that collaborative filtering methods like matrix factorisation may be needed here.

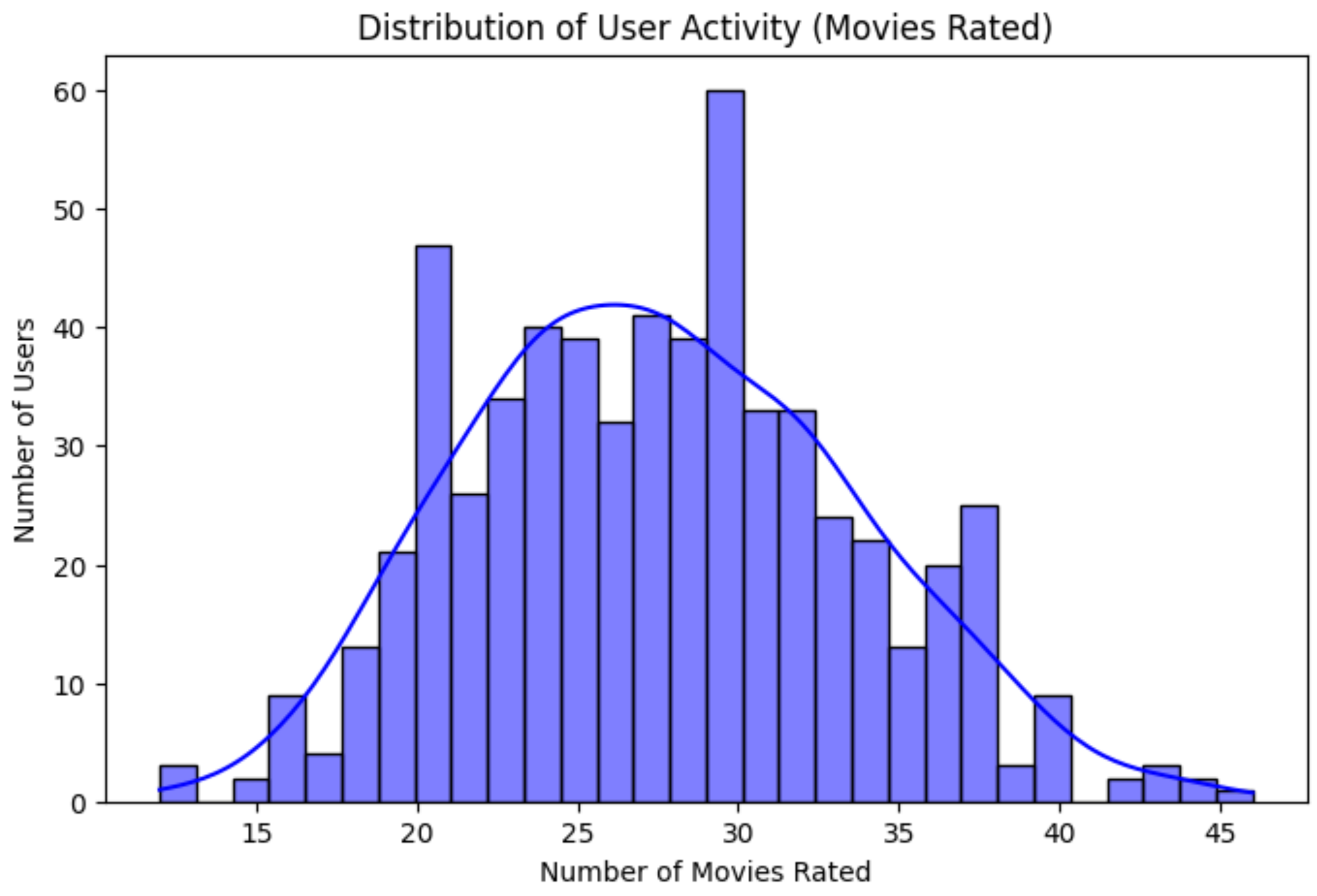
User Activity

In [55]:

```
user_activity = (user_reviews != 0).sum(axis=1)

print("Top 10 most active users (most ratings given):")
print("---------------------")
print(user_activity.sort_values(ascending=False).head(10))
```

```python
print("---------------------")

print("Top 10 least active users (least ratings given):")
print("---------------------")
print(user_activity.sort_values(ascending=True).head(10))

plt.figure(figsize=(8, 5))
sns.histplot(user_activity, bins=30, kde=True, color="blue")
plt.xlabel("Number of Movies Rated")
plt.ylabel("Number of Users")
plt.title("Distribution of User Activity (Movies Rated)")
plt.show()
```

```
Top 10 most active users (most ratings given):
---------------------
User
Jane        46
Evelyn      44
Abraham     44
David       43
Zachary     43
Sergio      43
Dante       42
Erin        42
Amina       40
Phillip     40
dtype: int64
---------------------
Top 10 least active users (least ratings given):
---------------------
User
Ace         12
Amira       13
Eden        13
Luka        15
Brady       15
Gael        16
Bethany     16
Aaron       16
Liana       16
Nathan      16
dtype: int64
```



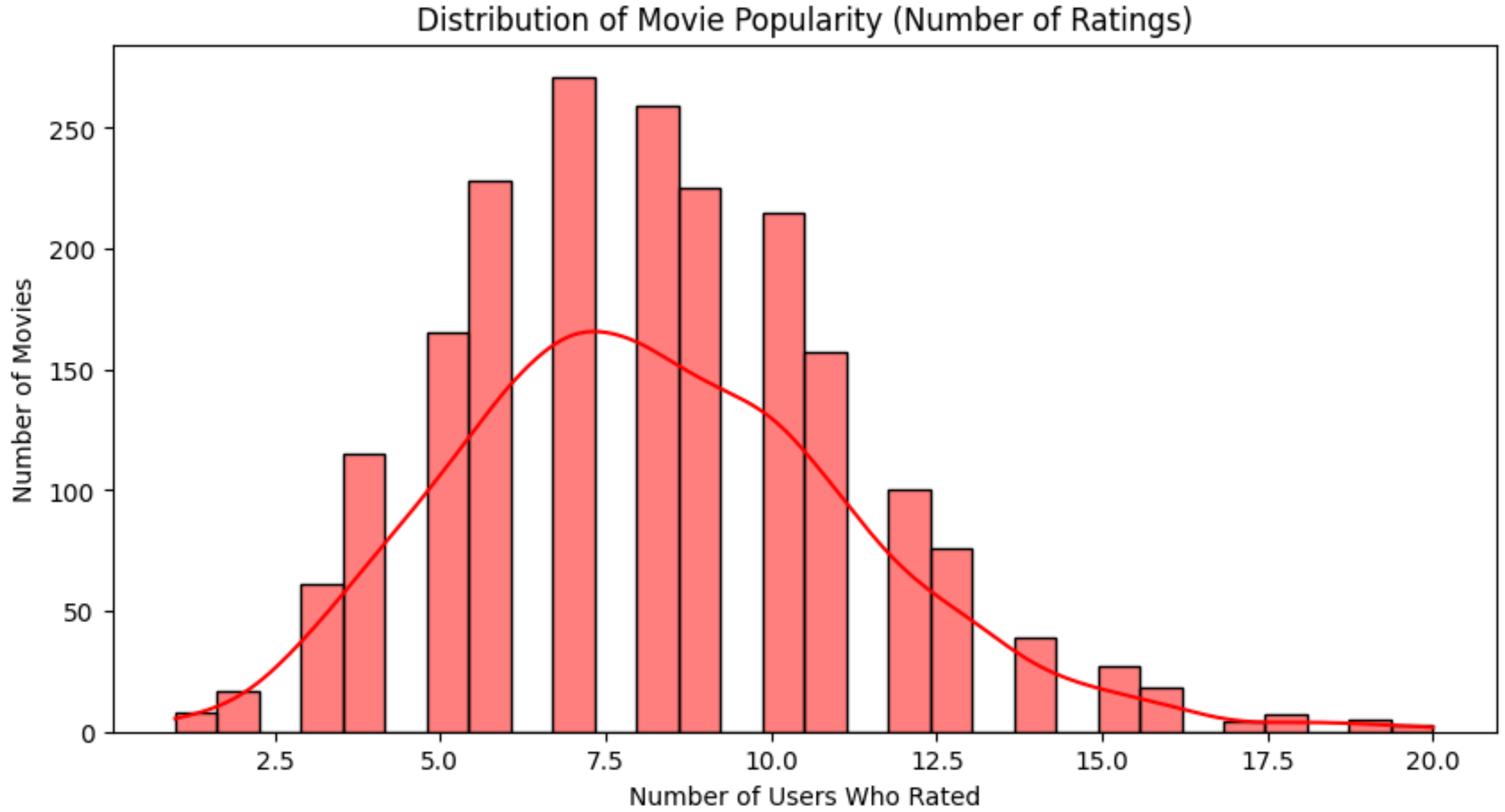
out of 2000 movies, the most a user has rated is only 46, which is very little.

In [13]:
```
## Movie Popularity
```

in terms of number of votes per movie, shows which are widely known and rated

In [14]:
```
movie_popularity = (user_reviews != 0).sum(axis=0)

print("Top 10 Most Rated Movies:")
print("---------------------")
print(movie_popularity.sort_values(ascending=False).head(10))

print("\nTop 10 Least Rated Movies:")
print("---------------------")
print(movie_popularity.sort_values(ascending=True).head(10))

plt.figure(figsize=(10, 5))
sns.histplot(movie_popularity, bins=30, kde=True, color="red")
plt.xlabel("Number of Users Who Rated")
plt.ylabel("Number of Movies")
plt.title("Distribution of Movie Popularity (Number of Ratings)")
plt.show()
```

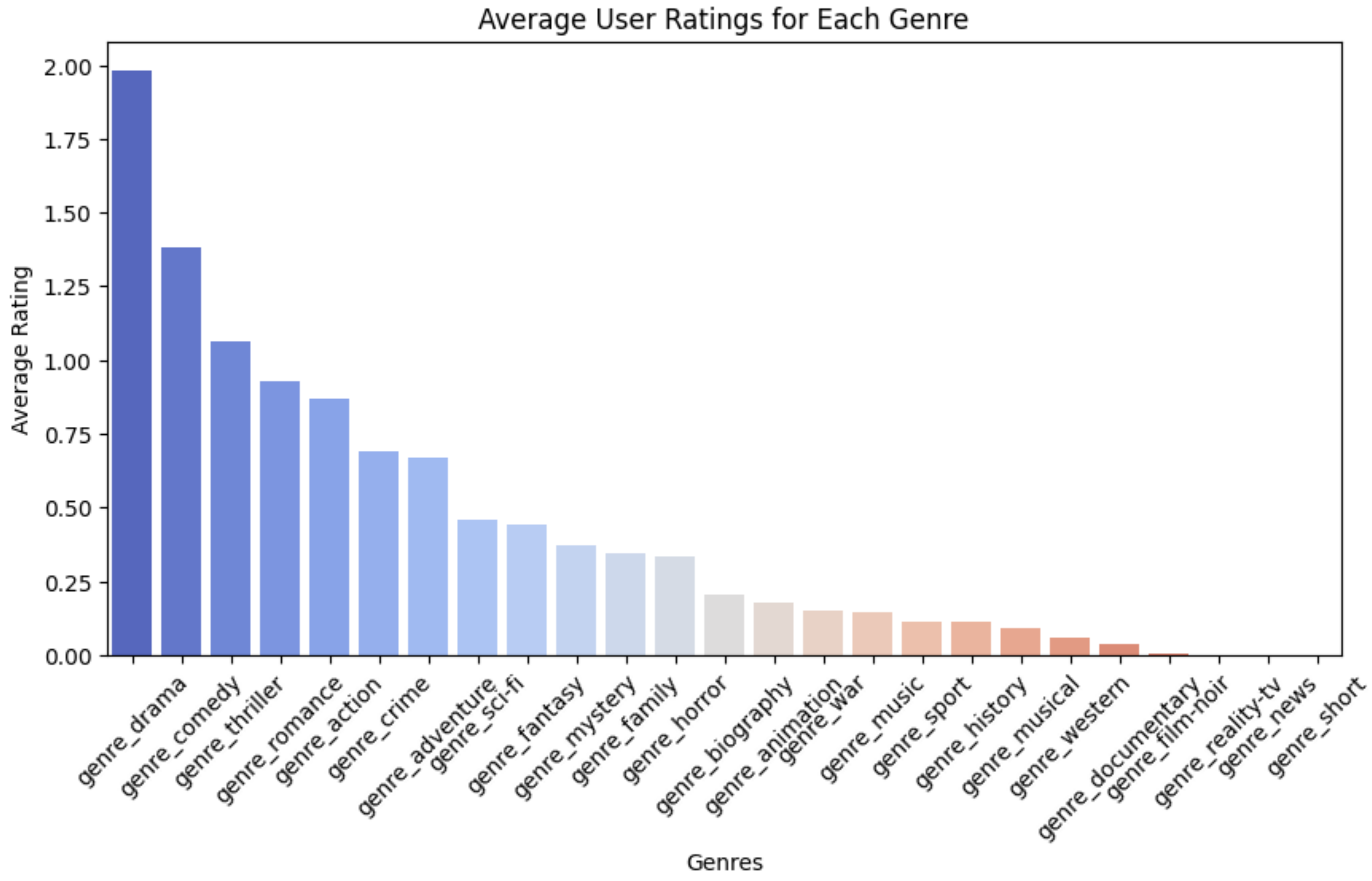
```
Top 10 Most Rated Movies:
---------------------
ATL                            20
Rang De Basanti                20
Observe and Report             20
Creepshow 2                    19
Perrier's Bounty               19
Furious 7                      19
Dysfunctional Friends          19
The Other End of the Line      19
Now You See Me 2               18
Killer Joe                     18
dtype: int64

Top 10 Least Rated Movies:
---------------------
Goal! The Dream Begins                        1
The Wolf of Wall Street                       1
Tarnation                                     1
The Men Who Stare at Goats                    1
United 93                                     1
12 Rounds                                     1
Ted 2                                         1
The Ballad of Gregorio Cortez                 1
The Living Wake                               2
Star Wars: Episode VI - Return of the Jedi    2
dtype: int64
```



Genre-Based User Preferences

In [15]:
```python
movies = user_reviews.columns
movie_genres["Movie"] = movies  # combine movie titles to movie_genres

ratings_long = user_reviews.melt(id_vars=[], var_name="Movie", value_name="Rating") # melt to long form
ratings_long = ratings_long[ratings_long["Rating"] > 0]  # remove missing ratings

ratings_with_genres = ratings_long.merge(movie_genres, on="Movie", how="left")

genre_ratings = ratings_with_genres.iloc[:, 2:].multiply(ratings_with_genres["Rating"], axis=0)
avg_genre_ratings = genre_ratings.mean().sort_values(ascending=False)

print("\nAverage User Ratings per Genre:")
print(avg_genre_ratings)

plt.figure(figsize=(10, 5))
sns.barplot(x=avg_genre_ratings.index, y=avg_genre_ratings.values, palette="coolwarm", legend=False, hue=avg_genre_ratings.index)
plt.xticks(rotation=45)
plt.xlabel("Genres")
plt.ylabel("Average Rating")
plt.title("Average User Ratings for Each Genre")
plt.show()
```

```
Average User Ratings per Genre:
genre_drama          1.980454
genre_comedy         1.380635
genre_thriller       1.061120
genre_romance        0.929259
genre_action         0.869228
genre_crime          0.689380
genre_adventure      0.666868
genre_sci-fi         0.459667
genre_fantasy        0.445083
genre_mystery        0.371800
genre_family         0.344932
genre_horror         0.336460
genre_biography      0.203691
genre_animation      0.176339
genre_war            0.152980
genre_music          0.144085
genre_sport          0.115340
genre_history        0.112980
genre_musical        0.093918
genre_western        0.057126
genre_documentary    0.038790
genre_film-noir      0.006293
genre_reality-tv     0.001573
genre_news           0.001513
genre_short          0.001392
dtype: float64
```



Hybrid Recommender System

Data Transformation

In [36]:
```
from surprise import Dataset, Reader, SVD

# melt into long format
ratings = user_reviews_replaced.stack().reset_index() # stack drops all the '0' rated movies, however it does not affect our training set anyway as we only want observed ratin
ratings.columns = ['user', 'movie', 'rating']

ratings
```

Out[36]:

	user	movie	rating
0	Vincent	About Last Night	2.0
1	Vincent	Shattered	3.0
2	Vincent	Passchendaele	3.0
3	Vincent	Broken Arrow	3.0
4	Vincent	Songcatcher	4.0
...	...	...	...
16520	Sarai	The Bank Job	4.0
16521	Sarai	The Stepfather	4.0
16522	Sarai	Good Kill	2.0
16523	Sarai	Sugar Hill	2.0
16524	Sarai	Waking Ned Devine	3.0

16525 rows × 3 columns

In [37]:

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings[['user', 'movie', 'rating']], reader)
```

Training SVD Model

In [ ]:

```
recco = SVD(n_factors=20, n_epochs=20, lr_all=0.005, reg_all=0.02) # general numbers for now, not optimised
trainset = data.build_full_trainset()
recco.fit(trainset)
```

Out[ ]:

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7fee1bf55610>
```

Generate CF Predictions

In [ ]:

```
# for each user, predict ratings for all unrated movies
cf_predictions = {}
for user in user_reviews_replaced.index[:5]:
    unrated_movies = user_reviews_replaced.columns[user_reviews_replaced.loc[user].isna()]
    # print(unrated_movies)
    predictions = [recco.predict(user, movie).est for movie in unrated_movies]
    cf_predictions[user] = pd.Series(predictions, index=unrated_movies)

cf_predictions
```

```
Index(['The Net', 'Happily N'Ever After', 'Tomorrowland', 'American Hero',
       'Das Boot', 'Final Destination 3', 'Licence to Kill',
       'The Hundred-Foot Journey', 'The Matrix', 'Creature',
       ...
       'The Martian', 'Micmacs', 'Solomon and Sheba', 'In the Company of Men',
       'Silent House', 'Big Fish', 'Get Real', 'Trading Places',
       'DOA: Dead or Alive', 'Hey Arnold! The Movie'],
      dtype='object', length=1961)
Index(['The Net', 'Happily N'Ever After', 'Tomorrowland', 'American Hero',
       'Das Boot', 'Final Destination 3', 'Licence to Kill',
       'The Hundred-Foot Journey', 'The Matrix', 'Creature',
       ...
       'The Martian', 'Micmacs', 'Solomon and Sheba', 'In the Company of Men',
       'Silent House', 'Big Fish', 'Get Real', 'Trading Places',
       'DOA: Dead or Alive', 'Hey Arnold! The Movie'],
      dtype='object', length=1970)
Index(['The Net', 'Happily N'Ever After', 'Tomorrowland', 'American Hero',
       'Das Boot', 'Final Destination 3', 'Licence to Kill',
       'The Hundred-Foot Journey', 'The Matrix', 'Creature',
       ...
       'The Martian', 'Micmacs', 'Solomon and Sheba', 'In the Company of Men',
       'Silent House', 'Big Fish', 'Get Real', 'Trading Places',
       'DOA: Dead or Alive', 'Hey Arnold! The Movie'],
      dtype='object', length=1964)
Index(['The Net', 'Happily N'Ever After', 'Tomorrowland', 'American Hero',
       'Das Boot', 'Final Destination 3', 'Licence to Kill',
       'The Hundred-Foot Journey', 'The Matrix', 'Creature',
       ...
       'The Martian', 'Micmacs', 'Solomon and Sheba', 'In the Company of Men',
       'Silent House', 'Big Fish', 'Get Real', 'Trading Places',
       'DOA: Dead or Alive', 'Hey Arnold! The Movie'],
      dtype='object', length=1968)
Index(['The Net', 'Happily N'Ever After', 'Tomorrowland', 'American Hero',
       'Das Boot', 'Final Destination 3', 'Licence to Kill',
       'The Hundred-Foot Journey', 'The Matrix', 'Creature',
       ...
       'The Martian', 'Micmacs', 'Solomon and Sheba', 'In the Company of Men',
       'Silent House', 'Big Fish', 'Get Real', 'Trading Places',
       'DOA: Dead or Alive', 'Hey Arnold! The Movie'],
      dtype='object', length=1972)
```

```
Out[ ]: {'Vincent': The Net                  4.285382
         Happily N'Ever After     3.791504
         Tomorrowland             4.081595
         American Hero            3.979292
         Das Boot                 4.111503
                                    ...
         Big Fish                 3.816723
         Get Real                 4.159638
         Trading Places           3.777461
         DOA: Dead or Alive       3.719609
         Hey Arnold! The Movie    3.747211
         Length: 1961, dtype: float64,
         'Edgar': The Net                  4.389378
         Happily N'Ever After     3.789844
         Tomorrowland             4.046551
         American Hero            4.013114
         Das Boot                 4.139079
                                    ...
         Big Fish                 3.906581
         Get Real                 4.109923
         Trading Places           3.765250
         DOA: Dead or Alive       3.559065
         Hey Arnold! The Movie    3.823449
         Length: 1970, dtype: float64,
         'Addilyn': The Net                  4.203173
         Happily N'Ever After     3.766691
         Tomorrowland             3.935846
         American Hero            3.863375
         Das Boot                 4.047522
                                    ...
         Big Fish                 3.689539
         Get Real                 3.913873
         Trading Places           3.655150
         DOA: Dead or Alive       3.529697
         Hey Arnold! The Movie    3.648536
         Length: 1964, dtype: float64,
         'Marlee': The Net                  4.071781
         Happily N'Ever After     3.582016
         Tomorrowland             3.620207
         American Hero            3.611717
         Das Boot                 3.783442
                                    ...
         Big Fish                 3.565291
         Get Real                 3.776320
         Trading Places           3.538117
         DOA: Dead or Alive       3.170607
         Hey Arnold! The Movie    3.534202
         Length: 1968, dtype: float64,
         'Javier': The Net                  3.721904
         Happily N'Ever After     3.176534
         Tomorrowland             3.333121
         American Hero            3.327716
         Das Boot                 3.540964
                                    ...
         Big Fish                 3.246796
         Get Real                 3.521667
         Trading Places           3.138313
         DOA: Dead or Alive       2.973268
         Hey Arnold! The Movie    3.228894
         Length: 1972, dtype: float64}
```

Using CBF Methods

we now use the movie genres to obtain user preferences and recommend similar movies with similar genres

In [46]:
```
movie_genres.drop(columns=['Movie'], inplace=True) # remove unnecessary column name from plot just now
```

In [48]:
```
user_genre_profiles = {}
for user in user_reviews_replaced.index[:5]:
    # movies the user has rated
    rated_movies = user_reviews_replaced.loc[user].dropna().index
    if len(rated_movies) == 0: # if cold start we use the average value
        user_genre_profiles[user] = movie_genres.mean()
    else:
        # weight genres by user ratings (higher-rated movies contribute more)
        user_ratings = user_reviews_replaced.loc[user, rated_movies]
        user_genre_profiles[user] = np.average(
            movie_genres.loc[rated_movies],
            axis=0,
            weights=user_ratings
        )

user_genre_profiles
```

Out[48]:
```
{'Vincent': array([0.34228188, 0.17449664, 0.02013423, 0.05369128, 0.23489933,
        0.11409396, 0.        , 0.80536913, 0.02013423, 0.19463087,
        0.        , 0.02013423, 0.03355705, 0.06040268, 0.03355705,
        0.09395973, 0.        , 0.        , 0.38926174, 0.12080537,
        0.        , 0.02013423, 0.30201342, 0.02013423, 0.        ]),
 'Edgar': array([0.10344828, 0.09482759, 0.07758621, 0.02586207, 0.18965517,
        0.15517241, 0.04310345, 0.75      , 0.03448276, 0.12068966,
        0.        , 0.04310345, 0.0862069 , 0.06034483, 0.12931034,
        0.06896552, 0.        , 0.        , 0.29310345, 0.13793103,
        0.        , 0.04310345, 0.26724138, 0.0862069 , 0.04310345]),
 'Addilyn': array([0.17164179, 0.17910448, 0.07462687, 0.04477612, 0.69402985,
        0.19402985, 0.        , 0.49253731, 0.13432836, 0.17164179,
        0.        , 0.        , 0.17164179, 0.1119403 , 0.07462687,
        0.01492537, 0.        , 0.        , 0.21641791, 0.1119403 ,
        0.        , 0.05223881, 0.23134328, 0.        , 0.        ]),
 'Marlee': array([0.16216216, 0.12612613, 0.        , 0.03603604, 0.3963964 ,
        0.24324324, 0.        , 0.61261261, 0.07207207, 0.09009009,
        0.        , 0.03603604, 0.18918919, 0.03603604, 0.        ,
        0.16216216, 0.        , 0.        , 0.13513514, 0.12612613,
        0.        , 0.04504505, 0.40540541, 0.03603604, 0.03603604]),
 'Javier': array([0.13636364, 0.22727273, 0.27272727, 0.11363636, 0.56818182,
        0.05681818, 0.        , 0.5       , 0.40909091, 0.22727273,
        0.        , 0.        , 0.14772727, 0.04545455, 0.02272727,
        0.05681818, 0.        , 0.        , 0.11363636, 0.04545455,
        0.        , 0.04545455, 0.125     , 0.04545455, 0.        ])}
```

Genre Similarity Score

using cosine similarity to get similar genres

In [50]:
```
from sklearn.metrics.pairwise import cosine_similarity

content_predictions = {}
```

```
for user in user_reviews_replaced.index[:5]:
    user_profile = user_genre_profiles[user].reshape(1, -1)
    unrated_movies = user_reviews_replaced.columns[user_reviews_replaced.loc[user].isna()]
    movie_genre_vectors = movie_genres.loc[unrated_movies]
    similarity_scores = cosine_similarity(user_profile, movie_genre_vectors)
    content_predictions[user] = pd.Series(similarity_scores[0], index=unrated_movies)

content_predictions
```

```
Out[50]: {'Vincent': The Net                  0.682683
          Happily N'Ever After     0.265334
          Tomorrowland             0.309556
          American Hero            0.692189
          Das Boot                 0.599485
                                     ...
          Big Fish                 0.624431
          Get Real                 0.760021
          Trading Places           0.216309
          DOA: Dead or Alive       0.336498
          Hey Arnold! The Movie    0.207039
          Length: 1961, dtype: float64,
          'Edgar': The Net                  0.638065
          Happily N'Ever After     0.245410
          Tomorrowland             0.208598
          American Hero            0.626492
          Das Boot                 0.635638
                                     ...
          Big Fish                 0.591402
          Get Real                 0.755093
          Trading Places           0.201209
          DOA: Dead or Alive       0.148744
          Hey Arnold! The Movie    0.210355
          Length: 1970, dtype: float64,
          'Addilyn': The Net                  0.483895
          Happily N'Ever After     0.549287
          Tomorrowland             0.268104
          American Hero            0.720130
          Das Boot                 0.442313
                                     ...
          Big Fish                 0.476972
          Get Real                 0.793546
          Trading Places           0.679920
          DOA: Dead or Alive       0.242973
          Hey Arnold! The Movie    0.530045
          Length: 1964, dtype: float64,
          'Marlee': The Net                  0.740798
          Happily N'Ever After     0.319890
          Tomorrowland             0.303054
          American Hero            0.677649
          Das Boot                 0.616472
                                     ...
          Big Fish                 0.499919
          Get Real                 0.690105
          Trading Places           0.414119
          DOA: Dead or Alive       0.212965
          Hey Arnold! The Movie    0.310589
          Length: 1968, dtype: float64,
          'Javier': The Net                  0.388414
          Happily N'Ever After     0.756650
          Tomorrowland             0.388414
          American Hero            0.620371
          Das Boot                 0.445539
                                     ...
          Big Fish                 0.547025
          Get Real                 0.677269
          Trading Places           0.563974
          DOA: Dead or Alive       0.255225
          Hey Arnold! The Movie    0.733166
          Length: 1972, dtype: float64}
```

Hybrid Recommender

In [51]:
```
hybrid_predictions = {}
alpha = 0.7  # arbitrary weight for CF, requires tuning later probably

for user in user_reviews_replaced.index[:5]:
    # normalize CF and CB scores
    cf_scores = cf_predictions[user]
    cf_scores = (cf_scores - cf_scores.min()) / (cf_scores.max() - cf_scores.min())

    cb_scores = content_predictions[user]
    cb_scores = (cb_scores - cb_scores.min()) / (cb_scores.max() - cb_scores.min())

    # use simple weighted average to combine scores
    hybrid_scores = alpha * cf_scores + (1 - alpha) * cb_scores
    hybrid_predictions[user] = hybrid_scores.sort_values(ascending=False)

hybrid_predictions
```

```
Out[51]: {'Vincent': Perrier's Bounty                            0.991436
          The Edge                                    0.898930
          The Good Thief                              0.896487
          Chill Factor                                0.880845
          Seeking a Friend for the End of the World   0.875372
                                                        ...
          Samsara                                     0.072205
          She Wore a Yellow Ribbon                    0.039841
          Peace, Propaganda & the Promised Land       0.039765
          One Missed Call                             0.029420
          Doc Holliday's Revenge                      0.000379
          Length: 1961, dtype: float64,
          'Edgar': Perrier's Bounty                            0.984061
          Chill Factor                                0.883221
          Seeking a Friend for the End of the World   0.875175
          The Good Thief                              0.870716
          BrainDead                                   0.839471
                                                        ...
          The Innkeepers                              0.085992
          Samsara                                     0.071417
          Pale Rider                                  0.048692
          One Missed Call                             0.032744
          Doc Holliday's Revenge                      0.028871
          Length: 1970, dtype: float64,
          'Addilyn': Perrier's Bounty                            0.991943
          Chill Factor                                0.910336
          Seeking a Friend for the End of the World   0.910308
          The Hunting Party                           0.874899
          BrainDead                                   0.852687
                                                        ...
          Samsara                                     0.078969
          She Wore a Yellow Ribbon                    0.050316
          One Missed Call                             0.047170
          Pale Rider                                  0.025195
          Doc Holliday's Revenge                      0.011914
          Length: 1964, dtype: float64,
          'Marlee': Perrier's Bounty            0.988880
          Chill Factor                0.907257
          The Hunting Party           0.896598
          BrainDead                   0.893709
          The Good Thief              0.864962
                                        ...
          One Missed Call             0.090879
          She Wore a Yellow Ribbon    0.069275
          Samsara                     0.038086
          Pale Rider                  0.014458
          Doc Holliday's Revenge      0.013043
          Length: 1968, dtype: float64,
          'Javier': Perrier's Bounty                            0.907757
          Sinbad: Legend of the Seven Seas            0.875037
          The Magic Sword: Quest for Camelot          0.868900
          Seeking a Friend for the End of the World   0.832632
          Chill Factor                                0.828354
                                                        ...
          Peace, Propaganda & the Promised Land       0.099594
          Evil Dead                                   0.091477
          She Wore a Yellow Ribbon                    0.059709
          Pale Rider                                  0.037375
          Doc Holliday's Revenge                      0.000000
          Length: 1972, dtype: float64}
```

Final Recommendations

In [52]:

```
recommendations = {}
for user in user_reviews_replaced.index[:5]:
    top_5 = hybrid_predictions[user].head(5).index.tolist()
    recommendations[user] = top_5

recommendations

for user, movies in recommendations.items():
    print(f"{user}: {movies}")
```

```
Vincent: ["Perrier's Bounty", 'The Edge', 'The Good Thief', 'Chill Factor', 'Seeking a Friend for the End of the World']
Edgar: ["Perrier's Bounty", 'Chill Factor', 'Seeking a Friend for the End of the World', 'The Good Thief', 'BrainDead']
Addilyn: ["Perrier's Bounty", 'Chill Factor', 'Seeking a Friend for the End of the World', 'The Hunting Party', 'BrainDead']
Marlee: ["Perrier's Bounty", 'Chill Factor', 'The Hunting Party', 'BrainDead', 'The Good Thief']
Javier: ["Perrier's Bounty", 'Sinbad: Legend of the Seven Seas', 'The Magic Sword: Quest for Camelot', 'Seeking a Friend for the End of the World', 'Chill Factor']
```