In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
sns.set(style='whitegrid')
```

In [95]:
```
# Data Preparation
# loading Dataset
ratings = pd.read_csv('ratings.csv')
movies = pd.read_csv('movies.csv')
tags = pd.read_csv('tags.csv')
links = pd.read_csv('links.csv')
```

In [96]:
```
print(f"Ratings: {ratings.shape}")
print(f"Movies: {movies.shape}")
print(f"Tags: {tags.shape}")
print(f"Links: {links.shape}")
```

```
Ratings: (100836, 4)
Movies: (9742, 3)
Tags: (3683, 4)
Links: (9742, 3)
```

In [12]:
```
print("\nRatings dataset preview:")
print(ratings.head(1))
print("\nMovies dataset preview:")
print(movies.head(1))
print("\nTags dataset preview:")
print(tags.head(1))
print("\nLinks dataset preview:")
print(links.head(1))
```

```
Ratings dataset preview:
   userId  movieId  rating  timestamp
0       1        1     4.0  964982703

Movies dataset preview:
   movieId             title                                       genres
0        1  Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy

Tags dataset preview:
   userId  movieId    tag   timestamp
0       2    60756  funny  1445714994

Links dataset preview:
   movieId  imdbId  tmdbId
0        1  114709   862.0
```

In [97]:
```
# merging datasets
movie_ratings = pd.merge(ratings, movies, on='movieId', how='left')
```

In [98]:
```
movie_tags = tags.groupby('movieId')['tag'].apply(lambda x: '|'.join(x)).reset_inde
movie_tags.columns = ['movieId', 'all_tags']
```

In [109…
```
df = pd.merge(movie_ratings, movie_tags, on='movieId', how='left')
```

In [110…
```
df = pd.merge(df, links, on='movieId', how='left')
```

In [111…
```
print(f"Final merged dataset shape: {df.shape}")
print("\nMerged dataset info:")
print(df.info())
```

```
Final merged dataset shape: (100836, 9)

Merged dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 9 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  
 4   title      100836 non-null  object 
 5   genres     100836 non-null  object 
 6   all_tags   48287 non-null   object 
 7   imdbId     100836 non-null  int64  
 8   tmdbId     100823 non-null  float64
dtypes: float64(2), int64(4), object(3)
memory usage: 6.9+ MB
None
```

In [112…
```
# Duplicates and Missing Values
print("Duplicate rows:")
print(df.duplicated().sum())
```

```
Duplicate rows:
0
```

In [20]:
```
print("\nMissing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
userId           0
movieId          0
rating           0
timestamp        0
title            0
genres           0
all_tags     52549
imdbId           0
tmdbId          13
dtype: int64
```

In [113…
```
# sorting missing values
df['all_tags'] = df['all_tags'].fillna('No Tags')
```

```
df['imdbId'] = df['imdbId'].fillna(0)
df['tmdbId'] = df['tmdbId'].fillna(0)
```

In [ ]:
```
print(f"\nAfter handling missing values:")
print(df.isnull().sum())
```

```
After handling missing values:
userId         0
movieId        0
rating         0
timestamp      0
title          0
genres         0
all_tags       0
imdbId         0
tmdbId         0
datetime       0
date           0
year_rated     0
month_rated    0
day_of_week    0
day            0
dtype: int64
```

In [114...
```
# Convet timestamp to proper datetime format
df['datetime'] = pd.to_datetime(df['timestamp'], unit='s')
df['date'] = df['datetime'].dt.date
df['day'] = df['datetime'].dt.day
df['year_rated'] = df['datetime'].dt.year
df['month_rated'] = df['datetime'].dt.month
df['day_of_week'] = df['datetime'].dt.day_name()
```

In [115...
```
print(f"\nAfter handling missing values:")
print(df.isnull().sum())

# Check again after changing timestamp to correct datatype to see if there are miss
```

```
After handling missing values:
userId         0
movieId        0
rating         0
timestamp      0
title          0
genres         0
all_tags       0
imdbId         0
tmdbId         0
datetime       0
date           0
day            0
year_rated     0
month_rated    0
day_of_week    0
dtype: int64
```

In [116…
```
print("Timestamp conversion completed:")
print(df[['timestamp', 'datetime', 'date', 'day',
          'day_of_week', 'month_rated', 'year_rated']].head(5))
```

```
Timestamp conversion completed:
   timestamp            datetime        date  day day_of_week  month_rated  \
0  964982703 2000-07-30 18:45:03  2000-07-30   30      Sunday            7
1  964981247 2000-07-30 18:20:47  2000-07-30   30      Sunday            7
2  964982224 2000-07-30 18:37:04  2000-07-30   30      Sunday            7
3  964983815 2000-07-30 19:03:35  2000-07-30   30      Sunday            7
4  964982931 2000-07-30 18:48:51  2000-07-30   30      Sunday            7

   year_rated
0        2000
1        2000
2        2000
3        2000
4        2000
```

In [117…
```
# Feature Engineering(creating new variables)
# feature_1: release_year from title
df['release_year'] = df['title'].str.extract(r'\((\d{4})\)').astype(float)
df['clean_title'] = df['title'].str.replace(r'\(\d{4}\)', '', regex=True).str.strip


print("\nRelease year extracted from title:")
print(df[['title', 'release_year', 'clean_title']].head(5))
```

```
Release year extracted from title:
                         title  release_year            clean_title
0             Toy Story (1995)        1995.0              Toy Story
1      Grumpier Old Men (1995)        1995.0       Grumpier Old Men
2                  Heat (1995)        1995.0                   Heat
3 Seven (a.k.a. Se7en) (1995)        1995.0   Seven (a.k.a. Se7en)
4  Usual Suspects, The (1995)        1995.0    Usual Suspects, The
```

In [165…
```
# feature_2: number of genres per movie
df['genre_count'] = df['genres'].apply(lambda x: len(x.split('|')) if x != '(no gen

print("\nNumber of genres per movie:")
print(df[['genres', 'genre_count']].head(5))
```

```
Number of genres per movie:
                                        genres  genre_count
0  Adventure|Animation|Children|Comedy|Fantasy            5
1                               Comedy|Romance            2
2                        Action|Crime|Thriller            3
3                             Mystery|Thriller            2
4                        Crime|Mystery|Thriller            3
```

In [119…
```
# # feature_3: primary genre of movies
df['primary_genre'] = df['genres'].apply(lambda x: x.split('|')[0] if x != '(no gen

print("\nPrimary genre extracted:")
print(df[['genres', 'primary_genre']].head(5))
```

```
Primary genre extracted:
                                        genres primary_genre
0  Adventure|Animation|Children|Comedy|Fantasy     Adventure
1                               Comedy|Romance        Comedy
2                        Action|Crime|Thriller        Action
3                             Mystery|Thriller       Mystery
4                       Crime|Mystery|Thriller         Crime
```

In [120…

```
# feature_4: no of tags per movie
df['tag_count'] = df['all_tags'].apply(lambda x: len(x.split('|')) if x != 'No Tags

print("\nNumber of tags per movie:")
print(df[['all_tags', 'tag_count']].head(5))
```

```
Number of tags per movie:
                                          all_tags  tag_count
0                                  pixar|pixar|fun          3
1                                        moldy|old          2
2                                          No Tags          0
3                mystery|twist ending|serial killer          3
4  mindfuck|suspense|thriller|tricky|twist ending...          6
```

In [ ]:

```
# feature_5: movie number of years when rated
df['movie_age_when_rated'] = df['year_rated'] - df['release_year']

print("\nMovie age when rated:")
print(df[['release_year', 'year_rated', 'movie_age_when_rated']].head(5))
```

```
Movie age when rated:
   release_year  year_rated  movie_age_when_rated
0        1995.0        2000                   5.0
1        1995.0        2000                   5.0
2        1995.0        2000                   5.0
3        1995.0        2000                   5.0
4        1995.0        2000                   5.0
```

In [ ]:

```
# feature_6: movie rating counts
movie_rating_counts = df['movieId'].value_counts().reset_index()
movie_rating_counts.columns = ['movieId', 'movie_ratings']
df = pd.merge(df, movie_rating_counts, on='movieId', how='left')

print("\nMovie rating counts added:")
print(df[['movieId', 'movie_ratings']].drop_duplicates().head(5))
```

```
Movie rating counts added:
   movieId  movie_ratings
0        1            215
1        3             52
2        6            102
3       47            203
4       50            204
```

In [133…

```
# feature 7: each user rating counts
user_rating_counts = df['userId'].value_counts().reset_index()
user_rating_counts.columns = ['userId', 'user_rating_frequency']
df = pd.merge(df, user_rating_counts, on='userId', how='left')
```

```
print("\nUser rating frequency added:")
print(df[['userId', 'user_rating_frequency']].drop_duplicates().head(5))
```

```
User rating frequency added:
     userId  user_rating_frequency
0         1                    232
232       2                     29
261       3                     39
300       4                    216
516       5                     44
```

In [138…

```
# feature 8: each user average rating
user_average_ratings = df.groupby('userId')['rating'].mean().reset_index()
user_average_ratings.columns = ['userId', 'user_average_rating']
df = pd.merge(df, user_average_ratings, on='userId', how='left')

print("\nUser average rating added:")
print(df[['userId', 'user_average_rating']].drop_duplicates().head(5))
```

```
User average rating added:
     userId  user_average_rating
0         1             4.366379
232       2             3.948276
261       3             2.435897
300       4             3.555556
516       5             3.636364
```

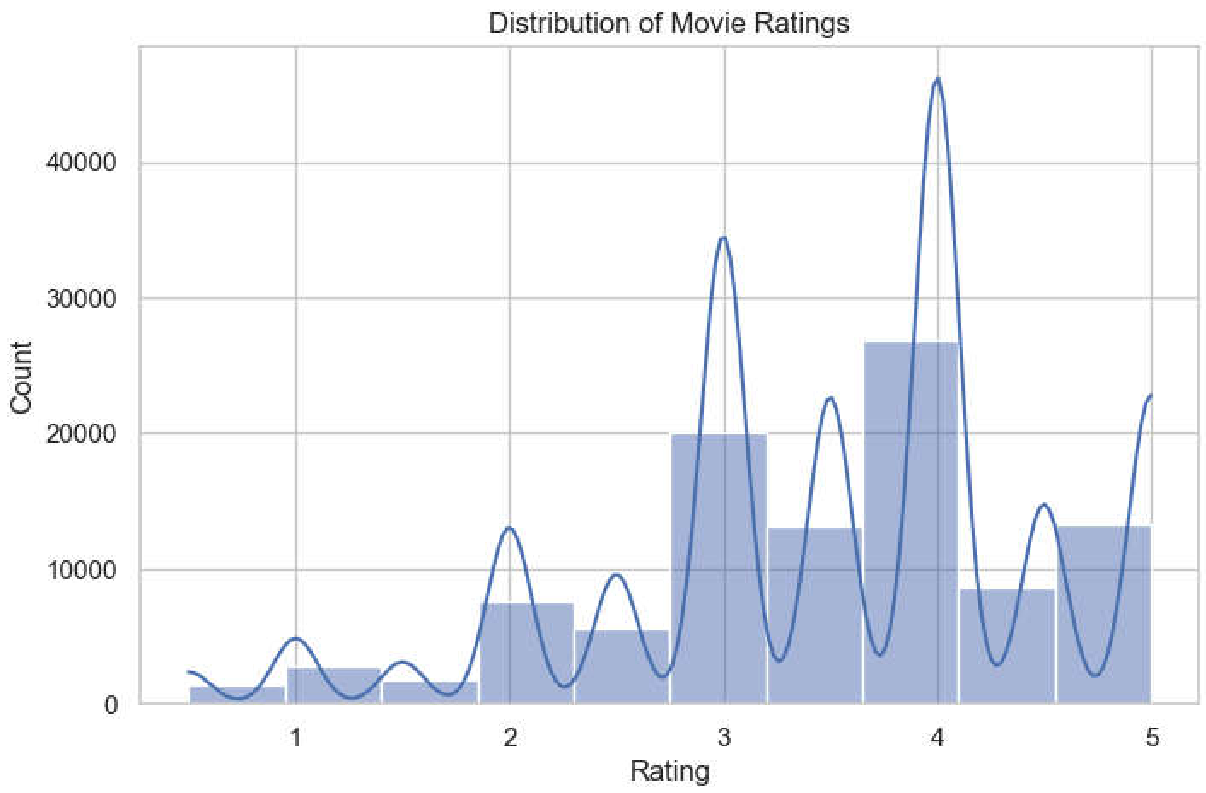
In [158…

```
# feature_8: user engagement per movie rating
def categorize_user_engagement(freq):
    if freq >= 100:
        return 'Most Active User'
    elif freq >= 50:
        return 'Active User'
    elif freq >= 20:
        return 'Less Active User'
    else:
        return 'Non-Active User'

df['user_engagement'] = df['user_rating_frequency'].apply(categorize_user_engagemen

print("\nuser engagement per movie rating:")
print(df[['release_year', 'genre_count', 'user_rating_frequency', 'movie_popularity
          'user_engagement']].head(10))
```

```
user engagement per movie rating:
   release_year  genre_count  user_rating_frequency  movie_popularity  \
0        1995.0            5                    232               215
1        1995.0            2                    232                52
2        1995.0            3                    232               102
3        1995.0            2                    232               203
4        1995.0            3                    232               204
5        1996.0            4                    232                55
6        1996.0            4                    232                23
7        1995.0            3                    232               237
8        1995.0            4                    232                44
9        1995.0            2                    232                11

    user_engagement
0  Most Active User
1  Most Active User
2  Most Active User
3  Most Active User
4  Most Active User
5  Most Active User
6  Most Active User
7  Most Active User
8  Most Active User
9  Most Active User
```

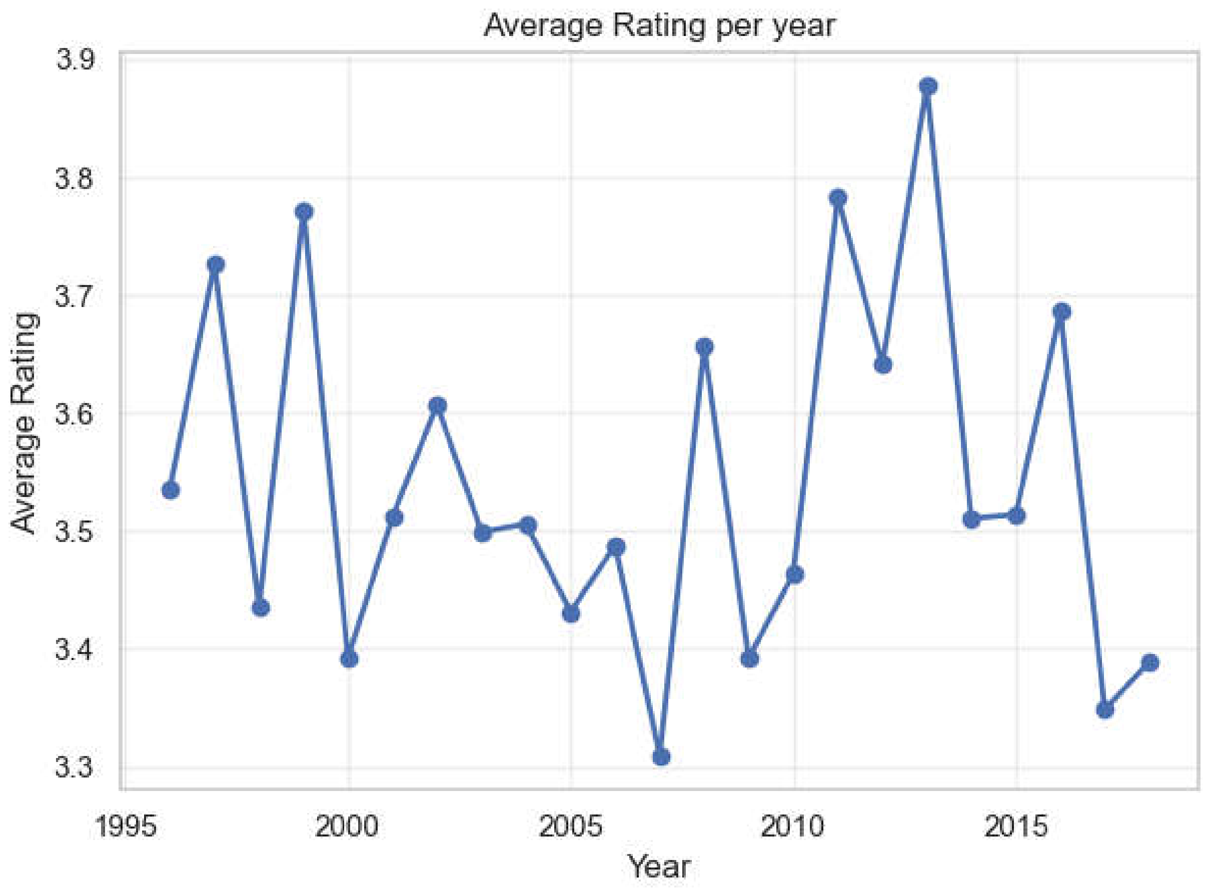
In [ ]:

```
# step 3: Exploratory Data Analysis (EDA)
# Insights 1: Distribution of movie ratings
plt.figure(figsize=(8,5))
sns.histplot(df['rating'], bins=10, kde=True)
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```

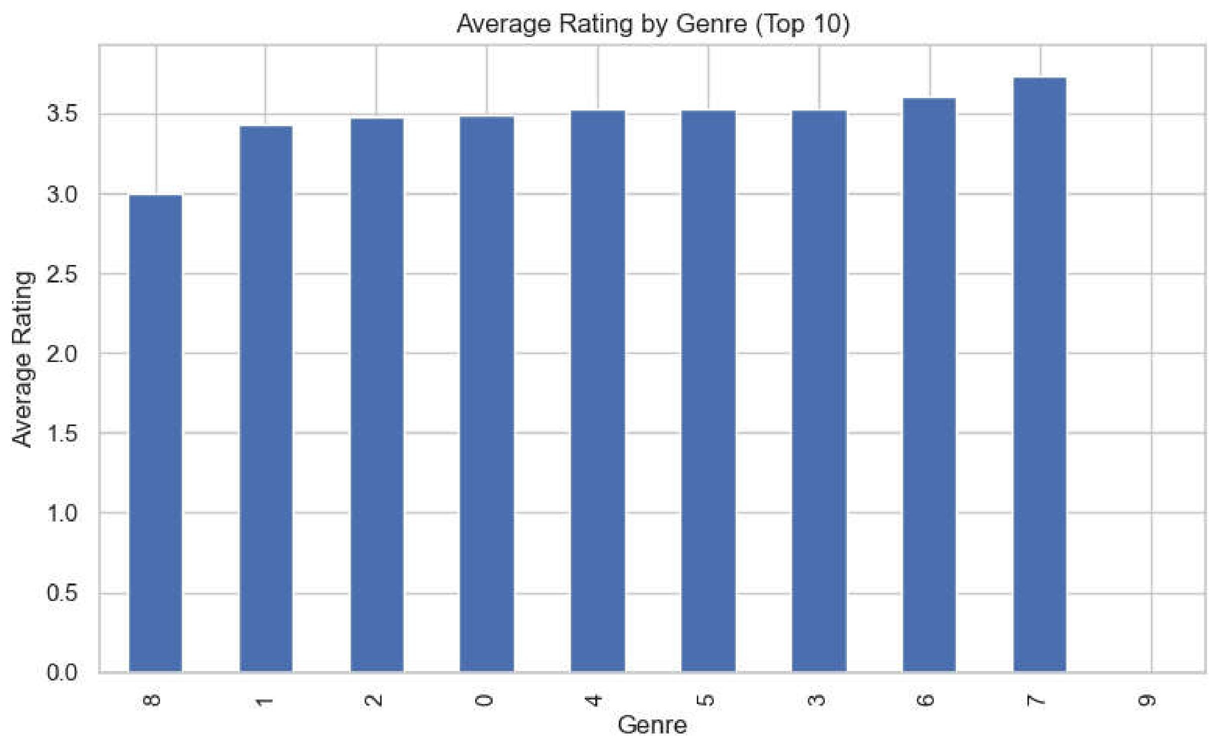


In [210…

```
# Insight 2: Average rating over year
yearly_avg_rating = df.groupby('year_rated')['rating'].mean()
plt.plot(yearly_avg_rating.index, yearly_avg_rating.values, marker='o', linewidth=2
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.title('Average Rating per year')
plt.grid(True, alpha=0.3)

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



In [ ]:
```
# Insights 3:Average rating per genre
genre_ratings = {}
for genre in (df[['genre_count']]).index[:10]:  # Top 15 genres
    genre_movies = df[df['genre_count']==genre]
    genre_ratings[genre] = genre_movies['rating'].mean()

plt.figure(figsize=(8, 5))
pd.Series(genre_ratings).sort_values().plot(kind='bar')
plt.xlabel('Genre')
plt.ylabel('Average Rating')
plt.title('Average Rating by Genre (Top 10)')
plt.tight_layout()
plt.show()
```

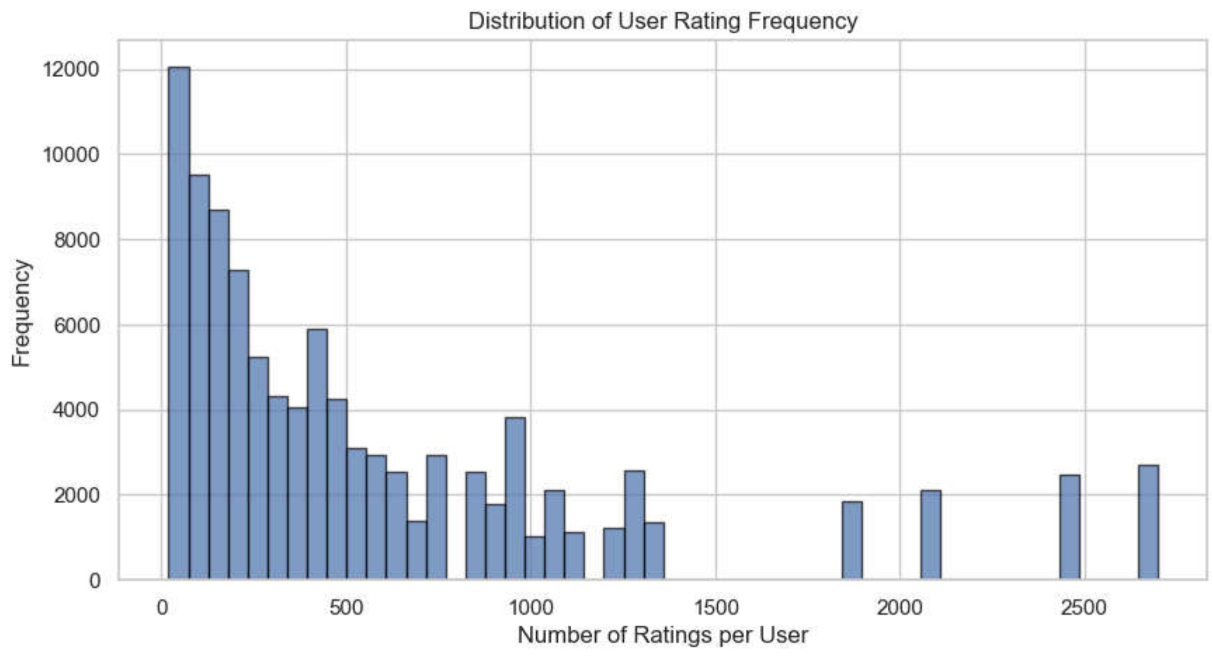


In [216…

```
# Insights 4: User rating frequency distribution
plt.figure(figsize=(10, 5))

user_freq = df['user_rating_frequency'].value_counts().sort_index()
plt.hist(df['user_rating_frequency'], bins=50, alpha=0.7, edgecolor='black')
plt.xlabel('Number of Ratings per User')
plt.ylabel('Frequency')
plt.title('Distribution of User Rating Frequency')
```

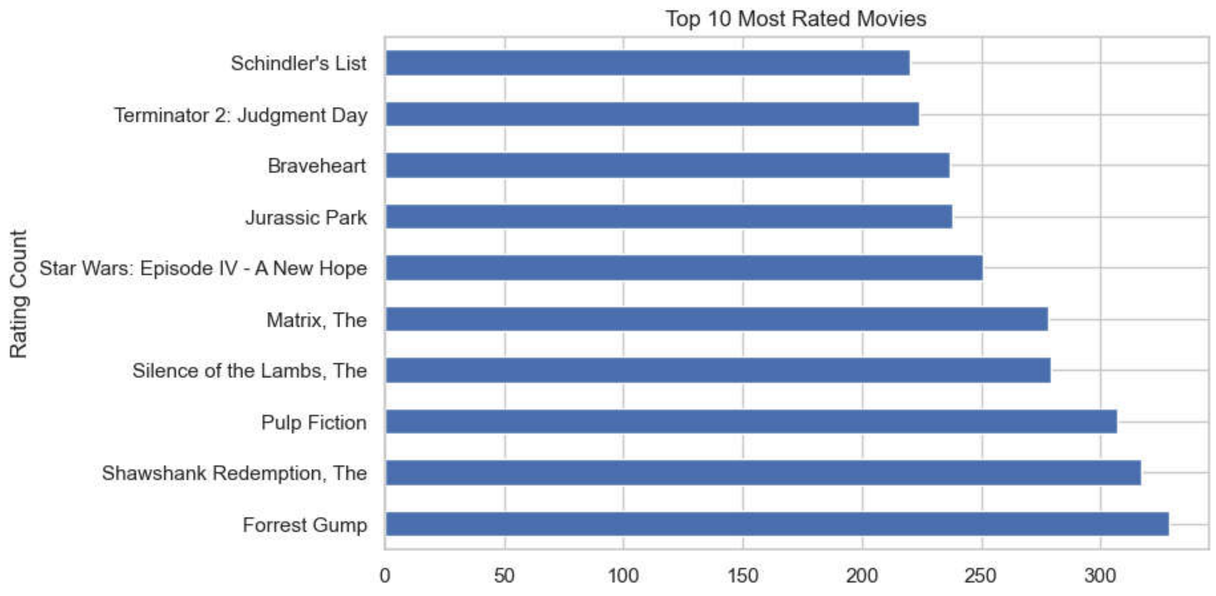
Out[216…

```
Text(0.5, 1.0, 'Distribution of User Rating Frequency')
```



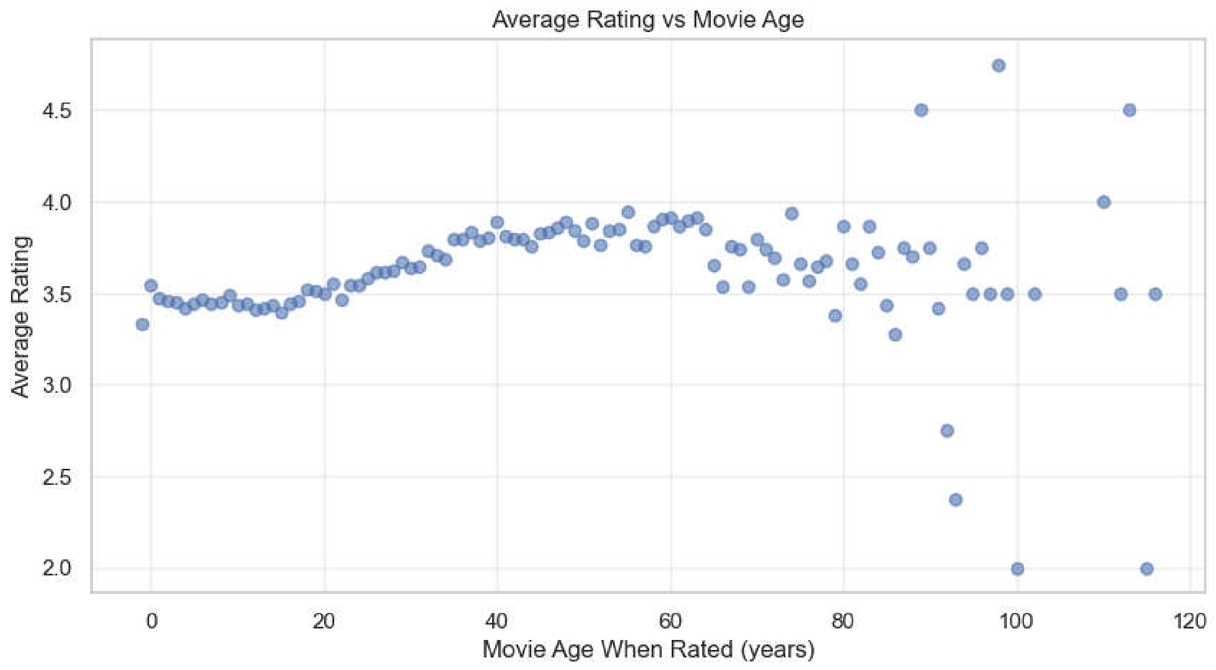
In [208…

```
# Insights 5: Most Rated Movies
popular_movies = df.groupby('clean_title')['rating_count'].max().sort_values(ascend
popular_movies.plot(kind='barh', figsize=(8,5))
plt.title('Top 10 Most Rated Movies')
plt.ylabel('Rating Count')
plt.show()
```



In [215…

```
# Insight 6: Movies Ratings vs movie Age
plt.figure(figsize=(10, 5))
movie_age_ratings = df.groupby('movie_age_when_rated')['rating'].mean()
plt.scatter(movie_age_ratings.index, movie_age_ratings.values, alpha=0.6)
plt.xlabel('Movie Age When Rated (years)')
plt.ylabel('Average Rating')
plt.title('Average Rating vs Movie Age')
plt.grid(True, alpha=0.3)
```



In [219…

```
# export movie_ratings
df.to_csv('cleaned_merged_movie_dataset.csv', index=False)
```

In [ ]:

```
```