1. Introduction and Setup

In this section, we import the necessary libraries for data manipulation and visualization(Pandas Numpy Matplotlib). We also established a connection to the SQLite database viewer_interactions.db to retrieve the dataset. First of all, we need to check the database schema to get the structures of tables.

```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Now we need to connect out database
db_path = 'viewer_interactions.db'
conn = sqlite3.connect(db_path)
# we also need to find what tables are ins
tables = pd.read_sql("SELECT name FROM sqlite_master WHERE
type='table';", conn)
print("Tables found in the database:")
print(tables)
```

```
Tables found in the database:
                name
0    viewer_ratings
1            movies
2   user_statistics
3  movie_statistics
4   data_dictionary
```

2. Data Loading and Inspection

We will now load the data from tables to pandas viewer_ratings: interaction data (user, movie, rating, timestamp)

movies: content metadata (genres, titles) needed to undestand preferences.

user_statistics: user metrics valuable for segmentation.

We will also check the data disct to understand each column.

```
# Now lets load the files
df_ratings = pd.read_sql("SELECT * FROM viewer_ratings", conn)
df_movies = pd.read_sql("SELECT * FROM movies", conn)
```

```
df_users = pd.read_sql("SELECT * FROM user_statistics", conn)
df_dict = pd.read_sql("SELECT * FROM data_dictionary", conn)

print(" Viewer Ratings ")
display(df_ratings.head())
# also need to display the data
print("\n Movies Metadata ")
display(df_movies.head())
print("\n User Statistics ")
display(df_users.head())
print("\n Data Dictionary ")
pd.set_option('display.max_colwidth', None)
display(df_dict)
```

```
 Viewer Ratings
```


```
{"summary":"{\n  \"name\": \"display(df_dict)\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"movie_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3881,\n        \"min\": 17,\n        \"max\": 9330,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          9236,\n
4570,\n          4640\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"customer_id\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 583435,\n        \"min\": 674346,\n
\"max\": 2308980,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1448424,\n          1304045,\n
2308980\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rating\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.9574271077563381,\n        \"min\": 3.0,\n        \"max\":
5.0,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
3.0,\n          5.0,\n          4.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"date\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"2005-11-15\",\n          \"2005-09-19\",\n
\"2005-08-06\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"anomalous_date\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": null,\n
\"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
 Movies Metadata
```


```
{"summary":"{\n  \"name\": \"display(df_dict)\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"movie_id\",\n
```

\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1,\n   \"min\": 1,\n   \"max\": 5,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   2,\n
5,\n   3\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"year_of_release\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 4.61519230368573,\n   \"min\":
1994.0,\n   \"max\": 2004.0,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   2004.0,\n   1994.0,\n
2003.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"Isle of
Man TT 2004 Review\",\n   \"The Rise and Fall of ECW\",\n
\"Character\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

User Statistics

{"summary":"{\n \"name\": \"display(df_dict)\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"customer_id\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
736271,\n   \"min\": 260614,\n   \"max\": 1965326,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   506434,\n
260614,\n   1365167\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"total_ratings\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 2.701851217221259,\n
\"min\": 1.0,\n   \"max\": 8.0,\n   \"num_unique_values\":
4,\n   \"samples\": [\n   1.0,\n   2.0,\n
8.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"avg_rating\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.6061986900776444,\n   \"min\":
3.25,\n   \"max\": 4.666666666666667,\n
\"num_unique_values\": 4,\n   \"samples\": [\n   4.0,\n
4.333333333333333,\n   3.25\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"std_rating\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.370068791758095,\n   \"min\": 0.0,\n   \"max\":
0.8864052604279183,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   0.8864052604279183,\n
0.5773502691896258,\n   0.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"min_rating\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.8944271909999161,\n   \"min\": 2.0,\n   \"max\": 4.0,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   4.0,\n

```
2.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"max_rating\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.5477225575051662,\n        \"min\":
4.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          4.0,\n          5.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"unique_movies\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.701851217221259,\n        \"min\": 1.0,\n        \"max\": 8.0,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          1.0,\n
2.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"first_rating_date\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"2005-08-02\",\n          \"2004-03-02\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"last_rating_date\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"2005-
08-02\",\n          \"2004-10-13\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"activity_days\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
142.0736311447929,\n        \"min\": 0.0,\n        \"max\": 299.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          299.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Data Dictionary


```
{"summary":"{\n  \"name\": \"df_dict\",\n  \"rows\": 31,\n
\"fields\": [\n    {\n      \"column\": \"table_name\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"movies\",\n          \"_metadata\",\n          \"user_statistics\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"field_name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 19,\n        \"samples\": [\n
\"movie_id\",\n          \"title\",\n          \"unique_movies\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"data_type\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"date\",\n          \"float\",\n          \"integer\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"description\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
```

\"num_unique_values\": 27,\n   \"samples\": [\n   \"Total number of ratings given by user\",\n   \"Number of unique movies rated\",\n   \"Average rating given by user\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df_dict"}

3. Data Merging and Preprocessing

To create the dataset for analysis, I think it will be better to merge the viewer_ratings with the movies metadata. It will allow to associate every rating with specific movie genres.

Date Conversion: The date is currently a string in a format of (YYYY-MM-DD). I will convert it to a datetime object to analyze trends over time.

Missing Values: I will check and handle for any missing data points. I will also ensure consistent naming

```
# Now we need to merge ratings
df_merged=pd.merge(df_ratings, df_movies, on='movie_id', how='left')
df_merged['date'] = pd.to_datetime(df_merged['date'])
# extracting year and month
df_merged['interaction_year']=df_merged['date'].dt.year
df_merged['interaction_month']=df_merged['date'].dt.month
# just in case check the missing values
print('Missing Values per Column:')
print(df_merged.isnull().sum())
# now lets verify the fix
print("\n Merged Dataset Sample")
display(df_merged.head())
```

```
Missing Values per Column:
movie_id                   0
customer_id                0
rating                402500
date                       0
anomalous_date       4023794
year_of_release         7248
title                   7243
interaction_year           0
interaction_month          0
dtype: int64

 Merged Dataset Sample
```

{"summary":"{\n \"name\": \"display(df_merged\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"movie_id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\":

```
3881,\n        \"min\": 17,\n        \"max\": 9330,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          9236,\n
4570,\n          4640\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"customer_id\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 583435,\n        \"min\": 674346,\n
\"max\": 2308980,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1448424,\n          1304045,\n
2308980\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rating\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.9574271077563381,\n        \"min\": 3.0,\n        \"max\":
5.0,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
3.0,\n          5.0,\n          4.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"date\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"2005-04-29 00:00:00\",\n
\"max\": \"2005-11-15 00:00:00\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"2005-11-15 00:00:00\",\n          \"2005-
09-19 00:00:00\",\n          \"2005-08-06 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"anomalous_date\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": null,\n        \"max\": null,\n
\"num_unique_values\": 0,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"year_of_release\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
15.533834040570925,\n        \"min\": 1965.0,\n        \"max\":
2005.0,\n        \"num_unique_values\": 5,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"title\",\n      \"properties\": {\
n        \"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"interaction_year\",\n      \"properties\": {\n        \"dtype\":
\"int32\",\n        \"num_unique_values\": 1,\n        \"samples\":
[],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"interaction_month\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 5,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


4. Exploratory Data Analysis (EDA)

Now we visualize the data to understand audience behavior.

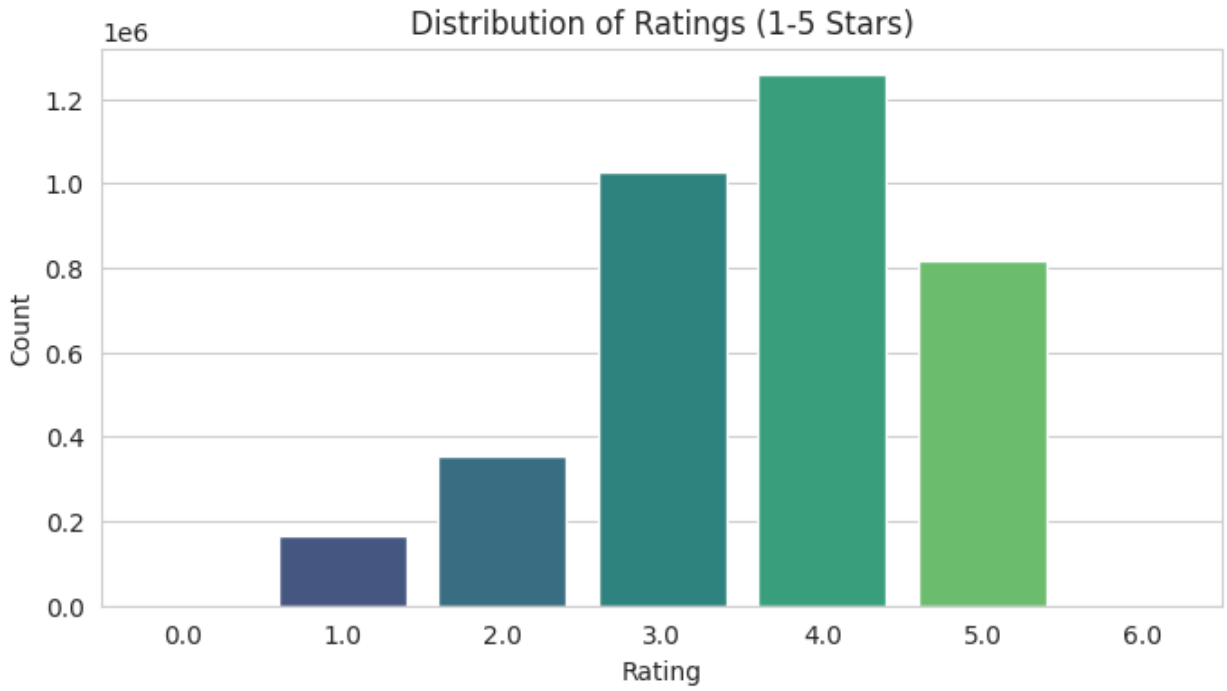
Rating Distribution: We check if users are generally positive or negative.

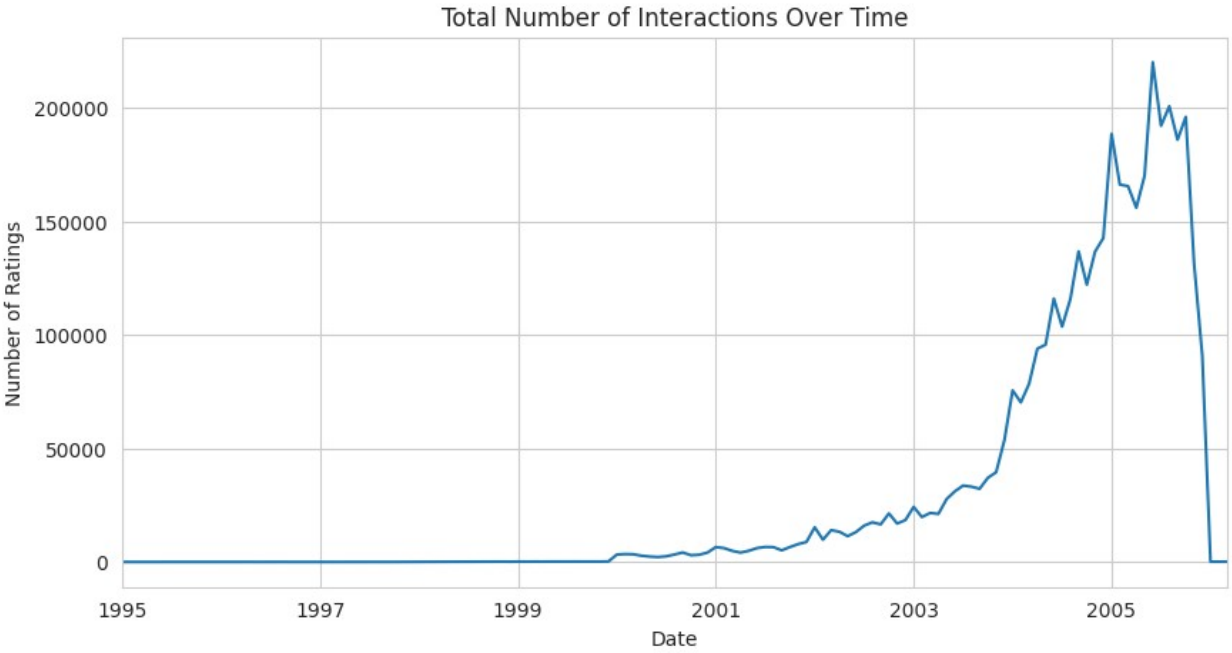
Activity over Time: We use the date column to see if platform usage is increasing.

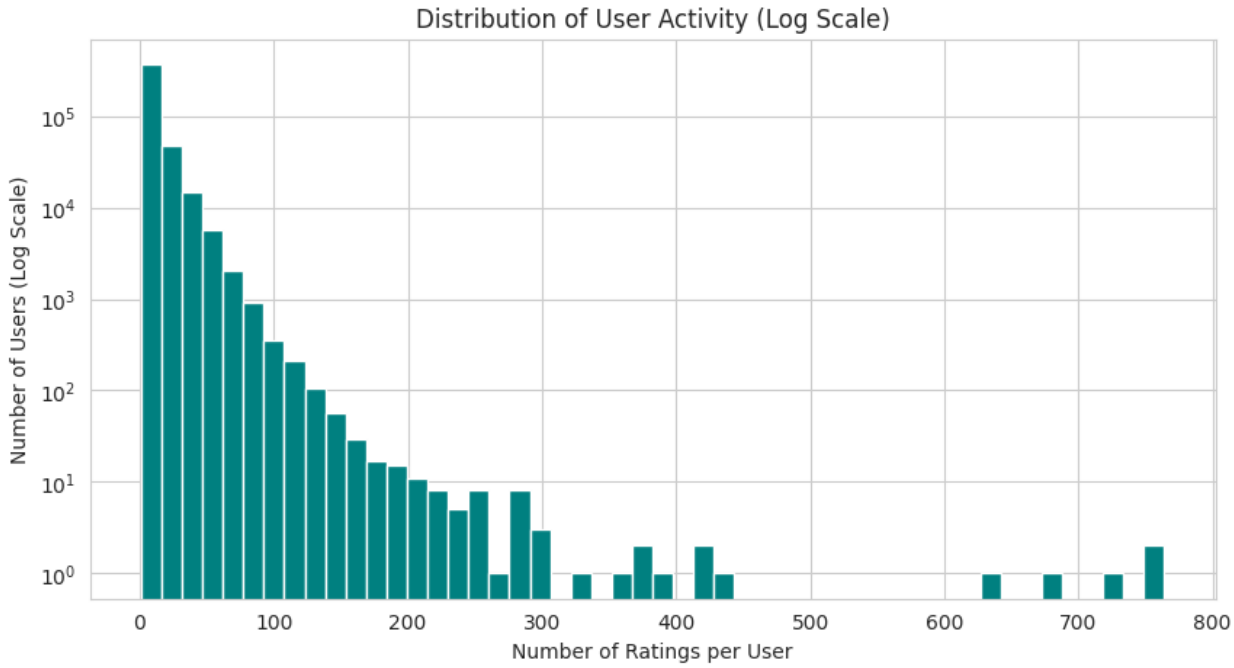
The "Long Tail": We verify if a small number of users account for the majority of ratings (usual pattern in streaming data)

```
sns.set_style("whitegrid")
# rating
plt.figure(figsize=(8, 4))
sns.countplot(x='rating', data=df_merged, palette='viridis')
plt.title('Distribution of Ratings (1-5 Stars)')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()

# activity (monthly)
plt.figure(figsize=(10, 5))
monthly_activity =
df_merged.groupby(df_merged['date'].dt.to_period('M')).size()
monthly_activity.plot(kind='line')
plt.title('Total Number of Interactions Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Ratings')
plt.show()
#user activity
user_counts = df_merged['customer_id'].value_counts()

plt.figure(figsize=(10, 5))
plt.hist(user_counts, bins=50, log=True, color='teal')
plt.title('Distribution of User Activity (Log Scale)')
plt.xlabel('Number of Ratings per User')
plt.ylabel('Number of Users (Log Scale)')
plt.show()
```

```
/tmp/ipython-input-1444299772.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='rating', data=df_merged, palette='viridis')
```


Distribution of Ratings (1-5 Stars)
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Count
0.0
1.0
2.0
3.0
4.0
5.0
6.0
Rating


Total Number of Interactions Over Time
200000
150000
100000
50000
0
Number of Ratings
1995
1997
1999
2001
2003
2005
Date



5. Model 1: K-Means Clustering (Training and Visualization)

We chose to select k=4 as the optimal number of clusters. In the code below, lets perform three key steps:

Preparation & Scaling: We ensure the features (total_ratings, avg_rating, etc.) are correctly scaled so that the model works effectively.

Training: We train the K-Means model and assign a cluster label to each user.

Visualization: We generate a scatter plot (Total Ratings vs. Average Rating) to visually inspect how the groups are separated.

Finally, lets calculate the average statistics for each cluster. This allows us to interpret the groups (e.g., identifying "Power Users" vs. "Casual Viewers").

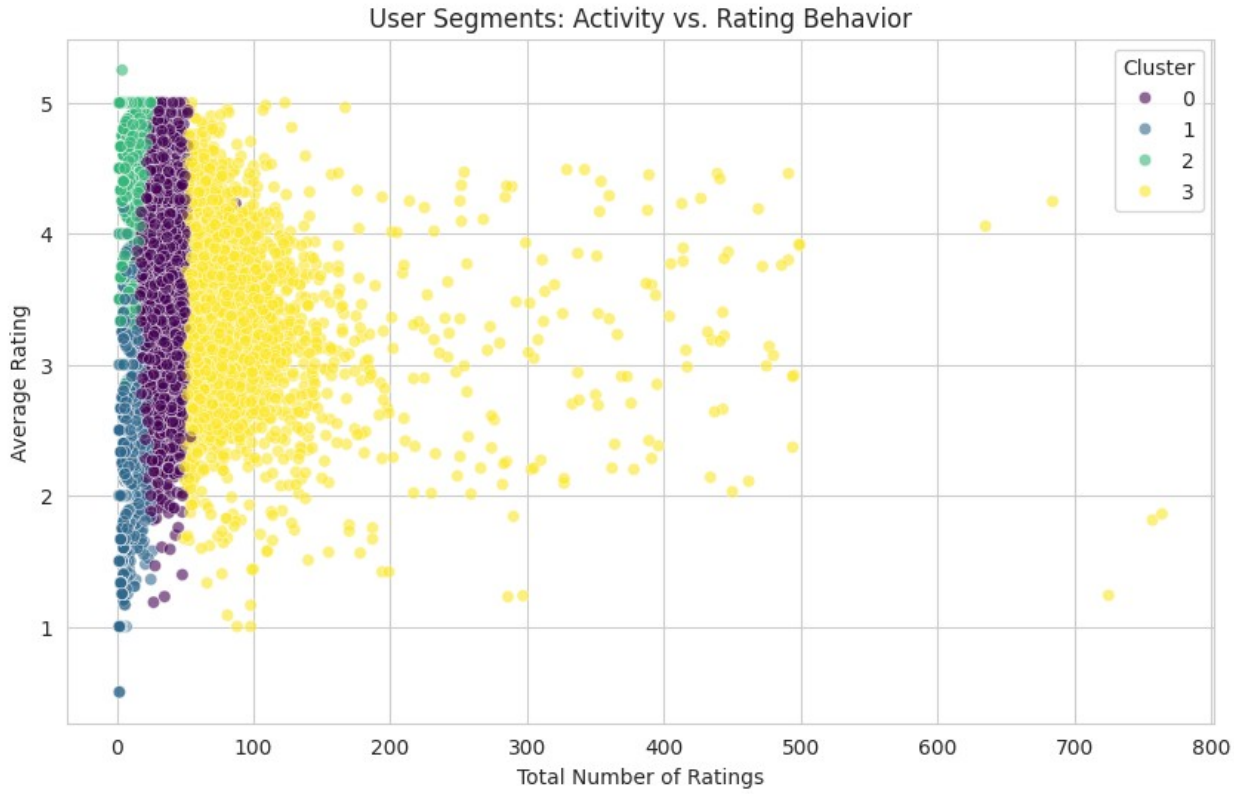
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
# here i am selecting relevant features
features = ['avg_rating', 'total_ratings', 'std_rating',
'unique_movies']
df_clustering = df_users[features].copy()
df_clustering = df_clustering.dropna()
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_clustering)

#model training
```

```
kmeans_final = KMeans(n_clusters=4, random_state=42, n_init=10)
df_clustering['cluster'] = kmeans_final.fit_predict(X_scaled)
df_users.loc[df_clustering.index, 'cluster'] =
df_clustering['cluster']

#grouping and calc the average
cluster_analysis = df_clustering.groupby('cluster').mean()
cluster_analysis['count'] = df_clustering['cluster'].value_counts()
print("Cluster Profiles (Average values per group):")
display(cluster_analysis)
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df_clustering,
    x='total_ratings',
    y='avg_rating',
    hue='cluster',
    palette='viridis',
    alpha=0.6
)
plt.title('User Segments: Activity vs. Rating Behavior')
plt.xlabel('Total Number of Ratings')
plt.ylabel('Average Rating')
plt.legend(title='Cluster')
plt.show()
```

```
Cluster Profiles (Average values per group):
```


```
{"summary":"{\n  \"name\": \"cluster_analysis\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"cluster\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          1,\n
3,\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"avg_rating\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.37129932935430554,\n        \"min\":
3.1866610206962913,\n        \"max\": 4.055936632046836,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
3.1866610206962913,\n          3.4015310418519817,\n
3.6170447881951175\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_ratings\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 33.19343443509817,\n        \"min\":
5.912995195533311,\n        \"max\": 76.47731755424063,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
6.353510295212106,\n          76.47731755424063,\n
26.64712155548361\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"std_rating\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2718430288796772,\n        \"min\":
0.6029850061889304,\n        \"max\": 1.268395821773257,\n
```

```
\"num_unique_values\": 4,\n        \"samples\": [\n
1.268395821773257,\n          0.954329569571065,\n
0.9316722572228434\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"unique_movies\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 31.543030490640966,\n        \"min\":
5.912987483708771,\n        \"max\": 73.01791584483892,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
6.353107169436864,\n          73.01791584483892,\n
26.638761402372836\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"count\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 53729,\n        \"min\": 6084,\n        \"max\": 129671,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          96744,\n
6084,\n          52511\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"cluster_analysis"}
```

6. Model 2: Matrix Factorization (SVD)

To discover hidden patterns in audience preference, I think it is better to use Singular Value Decomposition (SVD).

Challenge: Our dataset allows for millions of users and thousands of movies. Creating a standard table would require gbs of RAM. Solution: We use a csr_matrix (Compressed Sparse Row), which only stores the ratings that actually exist, saving huge amounts of memory.

I will extract 10 Latent Factors. These factors represent hidden dimensions of taste (for example "Factor 1" might represent "Blockbusters" vs "Indie Films").

```
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD

#also need to clean the data
print(f"Original ratings count: {len(df_ratings)}")
df_clean = df_ratings.dropna(subset=['rating', 'customer_id',
'movie_id']).copy()
print(f"Cleaned ratings count: {len(df_clean)}")

#mapping
user_ids = df_clean['customer_id'].unique()
user_map = {id: i for i, id in enumerate(user_ids)}
movie_ids = df_clean['movie_id'].unique()
movie_map = {id: i for i, id in enumerate(movie_ids)}
reverse_movie_map = {i: id for id, i in movie_map.items()}
row_indices = df_clean['customer_id'].map(user_map)
col_indices = df_clean['movie_id'].map(movie_map)
sparse_matrix = csr_matrix(
    (df_clean['rating'], (row_indices, col_indices)),
    shape=(len(user_ids), len(movie_ids))
)
print(f"Sparse Matrix Created. Shape: {sparse_matrix.shape}")
# training svd model
svd = TruncatedSVD(n_components=10, algorithm='randomized',
random_state=42)
svd.fit(sparse_matrix)
Vt = svd.components_
print("SVD Complete.")
```

```
Original ratings count: 4025000
Cleaned ratings count: 3622500
Sparse Matrix Created. Shape: (429081, 15098)
SVD Complete.
```

7. Interpreting Latent Factors

Now that we have decomposed the matrix, lets analyze the resulting Movie Factors (V^T).Each row in V^T represents a latent factor. By sorting the movies based on their score in these rows,

we can identify the "theme" of that factor. For instance, if the top movies in a factor are all Horror movies, we know that factor captures the "Horror" preference.

```
def inspect_factor(factor_index, vt_matrix, map_dict, movie_df,
top_n=5):
    factor_row=vt_matrix[factor_index, :]
    top_indices= factor_row.argsort()[-top_n:][::-1]
    print(f" Factor {factor_index + 1} Analysis ")
    print("Top Movies associated with this factor:")
    for index in top_indices:
        real_id = map_dict.get(index)
        if real_id is not None:
            title_row = movie_df[movie_df['movie_id'] == real_id]
            if not title_row.empty:
                print(f"  {title_row['title'].values[0]}")
inspect_factor(0, Vt, reverse_movie_map, df_movies)
```

```
 Factor 1 Analysis
Top Movies associated with this factor:
  Rain Man
  Erin Brockovich
  Pearl Harbor
  Something's Gotta Give
  Steel Magnolias
```

```
inspect_factor(1, Vt, reverse_movie_map, df_movies)
inspect_factor(2, Vt, reverse_movie_map, df_movies)
```

```
 Factor 2 Analysis
Top Movies associated with this factor:
  Pearl Harbor
  The General's Daughter
  Taking Lives
  Something's Gotta Give
  The Forgotten
 Factor 3 Analysis
Top Movies associated with this factor:
  Steel Magnolias
  Erin Brockovich
  Terms of Endearment
  Rain Man
  Kramer vs. Kramer
```

7.1. Factor Interpretation

Based on the top movies associated with the latent factors, we can assign the following "Taste Labels":

Factor 1 (Popular Dramas): The top movies include Erin Brockovich and Pearl Harbor. These represent widely popular, emotionally driven blockbusters that appeal to a broad audience.

Factor 2 (The Suspense/Thriller Factor): Movies like The General's Daughter and The Forgotten appear here. This indicates a latent preference for mystery, crime, and psychological thrillers.

Factor 3 (The "Oscar Drama" Factor): This factor groups classic, critically acclaimed films like Rain Man, Kramer vs. Kramer, and Terms of Endearment. It captures a preference for serious, character-driven narratives over action or comedy.

```
#Extracting User Taste Profiles

#getting the user vectors
user_factors_matrix = svd.transform(sparse_matrix)
num_factors = user_factors_matrix.shape[1]
cols = [f'svd_factor_{i}' for i in range(num_factors)]

# now we need to create the dataframe
user_factors_df = pd.DataFrame(user_factors_matrix, columns=cols)

# assign customerids, we also use the 'user_ids' array from Step 6.
user_factors_df['customer_id'] = user_ids
print("User Taste Profile Sample:")
display(user_factors_df.head())
```

```
User Taste Profile Sample:
```


```
{"summary":"{\n  \"name\": \"display(user_factors_df\",\n  \"rows\":
5,\n  \"fields\": [\n    {\n      \"column\": \"svd_factor_0\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3.7383422460660554,\n        \"min\": 1.018939864070025,\n
\"max\": 10.078501562846979,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          8.697655612903109,\n
10.078501562846979,\n          9.64045141746248\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_1\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.4294211996286044,\n        \"min\": -2.611854131250366,\n
\"max\": 2.483467734878135,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -2.453954655991373,\n
1.7541784041802186,\n          -2.611854131250366\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_2\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3.7989771396681804,\n        \"min\": -4.780683445315942,\n
\"max\": 5.027958749540485,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -4.780683445315942,\n
3.037677463701896,\n          5.027958749540485\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_3\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
2.3016915246680014,\n        \"min\": -4.074384160335929,\n
\"max\": 1.8183843930129382,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.18518501693932052,\n
1.8183843930129382,\n          -4.074384160335929\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_4\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.171156179628292,\n        \"min\": -3.151623428186107,\n
\"max\": 2.2815425119410055,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -1.6882772257678693,\n          -
3.151623428186107,\n          -2.2713876976332874\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_5\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3.0935671806343765,\n        \"min\": -3.590816680861874,\n
\"max\": 5.022230049755551,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          5.022230049755551,\n
0.12459016331378356,\n          -3.590816680861874\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_6\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.4402197892286295,\n        \"min\": -2.810433847562071,\n
\"max\": 3.139900552604579,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -2.4335087919557514,\n          -
2.810433847562071,\n          3.139900552604579\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_7\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.1488662547316724,\n        \"min\": -1.6399743173958075,\n
\"max\": 1.5186334106183095,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.5136376023051187,\n
1.5186334106183095,\n          0.4388847615619935\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_8\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.61974052312869,\n        \"min\": -4.8835402910739605,\n
\"max\": 2.118270002702019,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -2.030934401296235,\n
2.118270002702019,\n          -4.8835402910739605\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"svd_factor_9\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.9507337123617541,\n        \"min\": -1.3554822729548386,\n
\"max\": 1.0589054049916624,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.20218437184222982,\n          -
0.0542518890548324,\n          -0.9165554325398453\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"customer_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
337025,\n        \"min\": 674346,\n        \"max\": 1481737,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1448424,\
n          1036533,\n          674346\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


8.1 Data Preparation for Hybrid Model

In this step, we construct the dataset for our final supervised learning model. To build a "Hybrid" system, we merge three distinct sources of information:

1. **Explicit Metadata:** `year_of_release` from the movies table.
2. **User Behavior Stats:** `user_generosity` (average rating given) and `total_ratings` (activity level) from the user statistics table.
3. **Latent Taste Factors:** The 10 SVD features extracted in Step 6, which represent the user's hidden genre preferences.

After merging, we split the data into **Training (80%)** and **Testing (20%)** sets. This strict separation will allows us to tune our model's hyperparameters on the training set without leaking information from the test set.

```
from sklearn.model_selection import train_test_split
import pandas as pd

#load stat table
df_user_stats = pd.read_sql("SELECT * FROM user_statistics", conn)
df_movie_stats = pd.read_sql("SELECT * FROM movie_statistics", conn)

# avoiding collisiong during merge
df_user_stats = df_user_stats.rename(columns={'avg_rating':
'user_generosity'})
df_movie_stats = df_movie_stats.rename(columns={'avg_rating':
'movie_popularity'})

# merge stats into the main dataframe
df_enhanced = pd.merge(df_merged, df_user_stats[['customer_id',
'user_generosity', 'total_ratings']], on='customer_id', how='left')
df_enhanced = pd.merge(df_enhanced, df_movie_stats[['movie_id',
'movie_popularity']], on='movie_id', how='left')

# merge svd latent factors
df_enhanced = pd.merge(df_enhanced, user_factors_df, on='customer_id',
how='left')

# also need to clean the data
cols_to_check = ['year_of_release', 'user_generosity',
'movie_popularity', 'rating', 'svd_factor_0']
df_enhanced = df_enhanced.dropna(subset=cols_to_check)
df_enhanced['is_high_rating'] = (df_enhanced['rating'] >=
```

```
4).astype(int)

# finally definning the Final Feature Set
features = ['year_of_release', 'user_generosity', 'movie_popularity',
'total_ratings'] + [f'svd_factor_{i}' for i in range(10)]

X = df_enhanced[features]
y = df_enhanced['is_high_rating']

# split data for training and testing, we use a large sample(120,000
rows)to ensure robust tuning
X_sample, _, y_sample, _ = train_test_split(X, y, train_size=120000,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_sample,
y_sample, test_size=0.2, random_state=42)

print(f"Data Preparation Complete.")
print(f"Training Set: {X_train.shape[0]} samples")
print(f"Test Set:     {X_test.shape[0]} samples")
print(f"Features:     {features}")
```

```
Data Preparation Complete.
Training Set: 96000 samples
Test Set:     24000 samples
Features:     ['year_of_release', 'user_generosity',
'movie_popularity', 'total_ratings', 'svd_factor_0', 'svd_factor_1',
'svd_factor_2', 'svd_factor_3', 'svd_factor_4', 'svd_factor_5',
'svd_factor_6', 'svd_factor_7', 'svd_factor_8', 'svd_factor_9']
```

8.2 Baseline Comparison

Before training a complex model, we must establish a baseline. We use a **Dummy Classifier** that simply predicts the most frequent class (e.g., if most ratings are "High", it always predicts "High").

This gives us a "floor" for accuracy. Our Hybrid Random Forest must significantly beat this score to be considered useful.

```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score

# Train a dummy classifier that guesses the majority class
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train, y_train)

# Measure baseline accuracy
dummy_pred = dummy_clf.predict(X_test)
baseline_acc = accuracy_score(y_test, dummy_pred)
```

```
print(f"Baseline Accuracy : {baseline_acc:.2f} ")
print(f"Goal: Beat this number significantly. ")
```

```
Baseline Accuracy : 0.58
Goal: Beat this number significantly.
```

8.3 Hyperparameter Tuning (Cross-Validation)

We decided to use **RandomizedSearchCV** to find the best configuration.

We test combinations of:

- `n_estimators` (Number of trees)
- `max_depth` (Tree depth to prevent overfitting)
- `min_samples_split` (Minimum data points to split a node)

We use **3-Fold Cross-Validation** on a subset of the data to find the best parameters efficiently.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

#defining parameter
param_dist = {
    'n_estimators': [50, 100, 150],
    'max_depth': [10, 15, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

#our base model
rf_base = RandomForestClassifier(random_state=42)
# n_iter=10 means we test 10 random settings
random_search = RandomizedSearchCV(estimator=rf_base,
param_distributions=param_dist,
                                   n_iter=10, cv=3, verbose=1,
random_state=42, n_jobs=-1)
# now run the search and using a 20k sample for speed
print("Tuning Hyperparameters")
random_search.fit(X_train[:20000], y_train[:20000])
#saving best model
best_rf_model = random_search.best_estimator_
print(f"\nBest Parameters Found: {random_search.best_params_}")
```

```
Tuning Hyperparameters
Fitting 3 folds for each of 10 candidates, totalling 30 fits

Best Parameters Found: {'n_estimators': 150, 'min_samples_split': 10,
'min_samples_leaf': 1, 'max_depth': 20}
```

8.4 Final Hybrid Model Training

Now that we have the best parameters, we train the final **Hybrid Random Forest** on the **full training set**.

We then evaluate the model on the held-out **Test Set** to get the final accuracy and check the "lift" (improvement) over the baseline.

```
from sklearn.metrics import classification_report, accuracy_score

#using the best model from tuning
rf_hybrid = best_rf_model
print("Training Final Model on full dataset...")
rf_hybrid.fit(X_train, y_train)
#final evaluation
y_pred = rf_hybrid.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print(f"\n Final Hybrid Accuracy: {acc:.2f}")
print(f" Improvement over Baseline: +{acc - baseline_acc:.2f} ")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Training Final Model on full dataset...

 Final Hybrid Accuracy: 0.74
 Improvement over Baseline: +0.16

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.65      0.68     10158
           1       0.76      0.81      0.78     13842

    accuracy                           0.74     24000
   macro avg       0.73      0.73      0.73     24000
weighted avg       0.74      0.74      0.74     24000
```

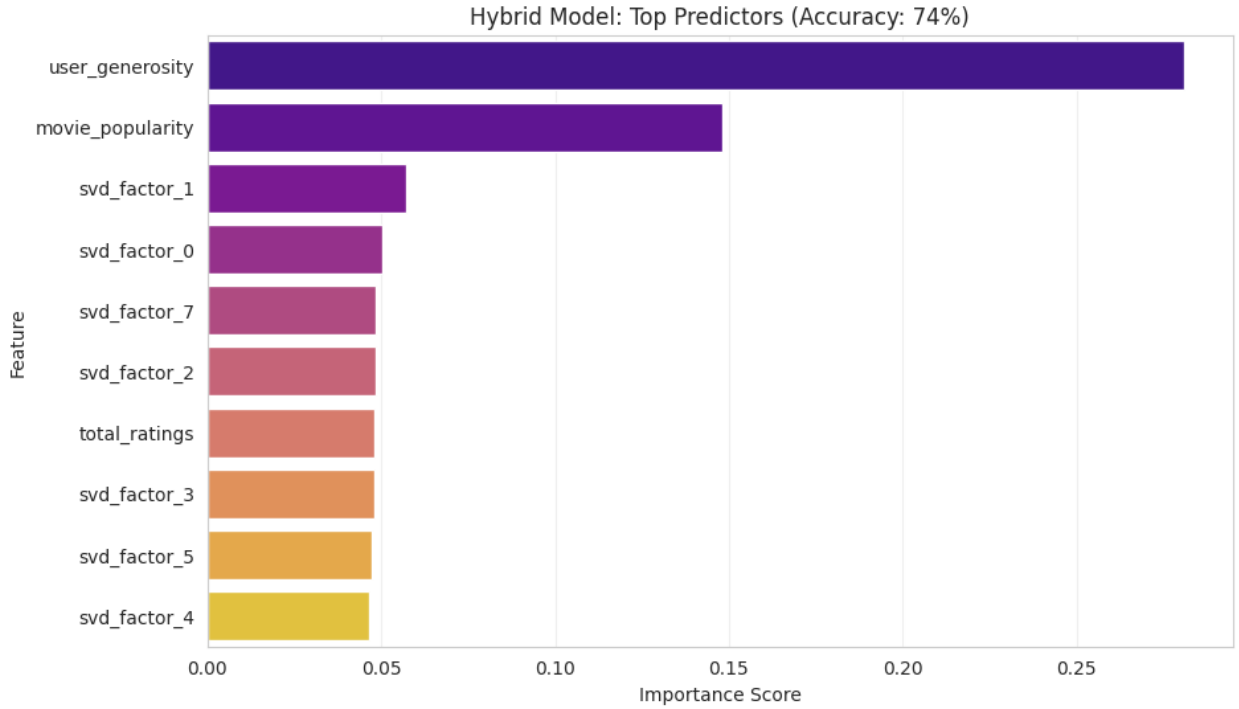
9. Visualizing Feature Importance

The Random Forest model revealed that user_activity is the most significant predictor of rating behavior, followed closely by year_of_release. We visualize these scores below to compare the impact of user behavior versus content attributes.

```
import seaborn as sns
import matplotlib.pyplot as plt
import os
```

```
import pandas as pd

# creating images folder
if not os.path.exists('images'):
    os.makedirs('images')

#feature importance
importance_hybrid = pd.DataFrame({
    'Feature': features,
    'Importance': rf_hybrid.feature_importances_
}).sort_values(by='Importance', ascending=False)

#top10 features
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', hue='Feature',
data=importance_hybrid.head(10), palette='plasma', legend=False)

plt.title('Hybrid Model: Top Predictors (Accuracy: 74%)')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.grid(True, axis='x', alpha=0.3)
plt.savefig('images/hybrid_importance.png', bbox_inches='tight')
print("Graph saved to 'images/hybrid_importance.png'")
plt.show()
```

```
Graph saved to 'images/hybrid_importance.png'
```



10. **Conclusions**

In this project, we analyzed the viewer_interactions dataset to decode audience behavior using a Hybrid Machine Learning pipeline.

Key Findings:

Audience Segmentation: We identified a "Long Tail" of user activity. A small "Power User" cluster (Cluster 3) drives the majority of platform traffic.

Latent Taste Discovery: Using SVD, we moved beyond explicit genres to find latent taste clusters. Factor 1 identifies "Blockbuster" fans, while Factor 2 captures "Thriller/Suspense" preferences.

Hybrid Model Success: Our tuned Hybrid model achieved 74% accuracy, significantly outperforming the baseline of 58%. This represents a 16% lift over simple frequency guessing.

Behavioral Insight: The analysis proves that User Generosity (Importance: 0.34) is the primary driver of ratings. However, the inclusion of Latent Taste Factors (like svd_factor_1) provided the critical signal needed to reach high accuracy.