In [ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```python
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [ ]:
```python
# packages
import numpy as np
import pandas as pd
import os
import json
import seaborn as sns
import matplotlib.pyplot as plt
import re
```

In [ ]:
```python
# data
file = "/content/drive/MyDrive/movie_dataset_public_final"
```

In [ ]:
```python
base_dir = "/content/drive/MyDrive/movie_dataset_public_final/raw/"

def load_ndjson_file(file_name, limit=None):
    data = []
    file_path = os.path.join(base_dir, file_name)

    with open(file_path, 'r', encoding='utf-8') as f:
        for i, line in enumerate(f):
            if limit and i >= limit:
                break
            try:
                data.append(json.loads(line))
            except json.JSONDecodeError as e:
                print(f"Error parsing line {i}: {line}")
                print(f"Error: {e}")
    return data

metadata = load_ndjson_file("metadata.json")
```

In [ ]:
```python
ratings = load_ndjson_file("ratings.json", limit = 5000000)
print(ratings[:3])
ratings = pd.DataFrame(ratings)
ratings.head()
```

```
[{'item_id': 5, 'user_id': 997206, 'rating': 3.0}, {'item_id': 10, 'user_id': 997206, 'rating': 4.0}, {'item_id': 13, 'user_id': 997206, 'rating': 4.0}]
```

Out[ ]:

	item_id	user_id	rating
0	5	997206	3.0
1	10	997206	4.0
2	13	997206	4.0
3	17	997206	5.0
4	21	997206	4.0

select users >= 100 and movies >= 70

In [ ]:
```python
user_counts = ratings['user_id'].value_counts()
valid_users = user_counts[user_counts > 100].index
qualified_ratings = ratings[ratings['user_id'].isin(valid_users)]
```

In [ ]:
```python
movie_counts = qualified_ratings['item_id'].value_counts()
valid_movies = movie_counts[movie_counts > 70].index
qualified_ratings = qualified_ratings[qualified_ratings['item_id'].isin(valid_movies)]
```

In [ ]:
```python
qualified_ratings
```

Out[ ]:

	item_id	user_id	rating
0	5	997206	3.0
1	10	997206	4.0
2	13	997206	4.0
3	17	997206	5.0
4	21	997206	4.0
...	...	...	...
4999944	3444	674044	3.0
4999945	3448	674044	4.0
4999946	3519	674044	4.0
4999947	4006	674044	4.0
4999948	5060	674044	4.0

3001682 rows × 3 columns

split into test and train dataset

In [ ]:

```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(qualified_ratings, test_size=0.2, random_state=10)
```

In [ ]:

```
train_file_path = '/content/drive/My Drive/movie_dataset_public_final/train_data_new.csv'
test_file_path = '/content/drive/My Drive/movie_dataset_public_final/test_data_new.csv'

train_data.to_csv(train_file_path, index=False)
test_data.to_csv(test_file_path, index=False)

print(f"Train data saved to Google Drive at: {train_file_path}")
print(f"Test data saved to Google Drive at: {test_file_path}")
```

```
Train data saved to Google Drive at: /content/drive/My Drive/movie_dataset_public_final/train_data_new.csv
Test data saved to Google Drive at: /content/drive/My Drive/movie_dataset_public_final/test_data_new.csv
```

In [ ]:

```
train_data
```

Out[ ]:

	item_id	user_id	rating
2698127	4386	288742	1.0
4729044	1466	209015	3.0
4567052	1446	29651	4.0
3360567	3307	711179	5.0
3574627	6639	89881	4.5
...	...	...	...
2806592	3730	861222	3.0
3609440	11	270526	3.0
4627188	2135	529638	3.0
3549771	898	699790	5.0
2380315	728	451757	5.0

2401345 rows × 3 columns

In [ ]:

```
user_movie_matrix = train_data.pivot_table(index='user_id', columns='item_id', values='rating')
user_movie_matrix
```

Out[ ]:

item_id	1	2	3	4	5	6	7	8	9	10	...	122882	122886	122904	134130	134853	142488	14
user_id																		
19	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	
110	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	
144	3.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
194	NaN	4.0	NaN	NaN	4.0	NaN	4.0	NaN	NaN	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	
281	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
999799	5.0	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN	
999845	NaN	NaN	NaN	NaN	NaN	3.0	1.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
999851	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
999869	NaN	NaN	NaN	NaN	NaN	3.0	2.0	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN	NaN	NaN	
999901	5.0	NaN	NaN	NaN	NaN	4.0	3.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	

12563 rows × 3804 columns

In [ ]:

```python
# Normalize Ratings by Subtracting User Means
user_means = user_movie_matrix.mean(axis=1)  # Calculate mean rating for each user
user_movie_matrix = user_movie_matrix.sub(user_means, axis=0)  # Subtract user means
user_movie_matrix.head()
```

Out[ ]:

item_id	1	2	3	4	5	6	7	8	9	10	...	122882	122886	1229
user_id														
19	1.406114	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-0.593886	...	NaN	NaN	N
110	0.872247	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-0.127753	-0.127753	...	NaN	NaN	N
144	-0.174212	NaN	NaN	NaN	NaN	1.825788	NaN	NaN	NaN	NaN	...	NaN	NaN	N
194	NaN	0.285714	NaN	NaN	0.285714	NaN	0.285714	NaN	NaN	-0.714286	...	NaN	NaN	N
281	NaN	-0.816667	NaN	NaN	NaN	NaN	NaN	NaN	0.183333	NaN	...	NaN	NaN	N

5 rows × 3804 columns

In [ ]:

```python
# Check sparsity
num_total_entries = user_movie_matrix.size  # Total number of cells
num_missing_entries = user_movie_matrix.isna().sum().sum()  # Number of missing entries
sparsity = num_missing_entries / num_total_entries  # Sparsity calculation

print(f"Data sparsity: {sparsity:.2%}")  # Output sparsity as a percentage
```

```
Data sparsity: 95.00%
```

use svd to fulfill NAN values within user-item matrix

In [ ]:

```python
user_movie_matrix.shape
```

Out[ ]:

```
(12563, 3804)
```

In [ ]:

```python
from sklearn.decomposition import TruncatedSVD
import numpy as np
from tqdm import tqdm
import time

# Fill NaN values in the user-item matrix with 0 (as required for SVD)
user_movie_matrix_filled = user_movie_matrix.fillna(0).values

def svd_with_progress(data, n_components, chunk_size=1000):
    """
    Perform SVD with progress tracking using tqdm.

    Parameters:
    - data: The user-item matrix (numpy array).
    - n_components: Number of components for dimensionality reduction.
    - chunk_size: Number of rows to process at a time.
```

```
    Returns:
    - user_features: User embedding matrix.
    - movie_features: Movie embedding matrix (transposed).
    """
    svd = TruncatedSVD(n_components=n_components, random_state=42)
    user_features_list = []
    start_time = time.time()

    # Process the matrix in chunks
    for i in tqdm(range(0, data.shape[0], chunk_size), desc="Processing SVD"):
        chunk = data[i:i + chunk_size]
        user_features_list.append(svd.fit_transform(chunk))

    # Combine processed chunks
    user_features = np.vstack(user_features_list)
    end_time = time.time()

    print(f"SVD completed in {end_time - start_time:.2f} seconds")
    return user_features, svd.components_.T

# Execute SVD decomposition
n_components = 50  # Adjust based on your dataset
user_features, movie_features = svd_with_progress(
    user_movie_matrix_filled, n_components=n_components, chunk_size=1000
)

# Predict the rating matrix by multiplying user and movie features
predicted_ratings = np.dot(user_features, movie_features.T)

# Convert predicted ratings back to a DataFrame for easier interpretation
predicted_ratings_df = pd.DataFrame(predicted_ratings, index=user_movie_matrix.index, columns=user_movie_matrix.

# Display progress
print("Predicted ratings matrix:")
print(predicted_ratings_df.head())
```

```
Processing SVD: 100%|██████████| 13/13 [00:11<00:00,  1.16it/s]
SVD completed in 11.26 seconds
Predicted ratings matrix:
item_id    1         2         3         4         5         6         7       \
user_id
19       0.439225 -0.210008 -0.067958 -0.195623 -0.249413  0.249318  0.108626
110      0.226099 -0.258291  0.022680 -0.139956 -0.178066  0.208525 -0.134432
144      3.928778  0.676412  0.465336 -0.290208 -0.085016 -0.820204 -0.559409
194      0.372853  0.064778  0.004215 -0.029979 -0.000832  0.135179 -0.008437
281      0.126331 -0.025364  0.023498 -0.024513  0.079875  0.101693  0.024951

item_id    8         9         10      ...    122882    122886    122904  \
user_id                                ...
19      -0.042914 -0.186523  0.029838  ...  0.025267 -0.035646  0.004302
110     -0.064228 -0.033422  0.080660  ... -0.053851  0.006396  0.074810
144     -0.285942 -0.237776 -0.236610  ... -0.138309  0.121350 -0.074112
194     -0.046637 -0.068643 -0.011595  ... -0.032872  0.025768 -0.013631
281      0.037433 -0.092773  0.198741  ...  0.000720  0.030816 -0.006240

item_id    134130    134853    142488    143385    148626    164179    166528
user_id
19      -0.013165  0.032932  0.047320 -0.024802  0.017723 -0.034189  0.003195
110      0.058365  0.024062  0.038030 -0.009597  0.014995  0.019629  0.027969
144      0.071783  0.132894 -0.139098 -0.141977  0.047853  0.151878 -0.072762
194     -0.005684 -0.015055 -0.019203 -0.023534 -0.004363  0.016544 -0.004241
281      0.045348 -0.003681  0.011002  0.047445 -0.012434  0.039675 -0.007875

[5 rows x 3804 columns]
```

use cosine similarity to calculate the similarity metrics

In [ ]:

```
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Compute user-to-user similarity
user_similarity = cosine_similarity(user_movie_matrix_filled)  # Input is the filled user-item matrix
user_similarity_df = pd.DataFrame(user_similarity, index=user_movie_matrix.index, columns=user_movie_matrix.inde

print("User Similarity Matrix:")
print(user_similarity_df.head())
```

```
User Similarity Matrix:
user_id    19         110        144        194        281        301        337     \
user_id
19        1.000000  0.055390  0.082475  0.052292  0.008436  0.091234  0.101988
110       0.055390  1.000000 -0.039201  0.059968  0.057699 -0.005846  0.060626
144       0.082475 -0.039201  1.000000 -0.031112 -0.025069  0.021368  0.033775
194       0.052292  0.059968 -0.031112  1.000000  0.136647  0.041423  0.055263
281       0.008436  0.057699 -0.025069  0.136647  1.000000  0.048269 -0.049340

user_id    364        372        489      ...    999506    999527    999549  \
user_id                                   ...
19        0.093516  0.127466  0.026647  ...  0.033266  0.060578  0.025270
110       0.087973  0.031983 -0.022259  ... -0.017718  0.062271  0.033002
144       0.001275  0.034263  0.041980  ...  0.067173  0.039500  0.011099
194       0.162218 -0.034590 -0.019204  ... -0.003641  0.007233 -0.032069
281       0.126720  0.079098  0.037195  ... -0.075035 -0.028103  0.011247

user_id    999590    999721    999799    999845    999851    999869    999901
user_id
19        0.027110  0.062299  0.099324  0.068719  0.054825  0.035108  0.112084
110       0.107345  0.093093  0.040699  0.002348  0.044720  0.009803  0.031383
144      -0.005779 -0.003979  0.050420  0.101318  0.017539  0.178241  0.077019
194       0.275258  0.164485  0.033521  0.005188  0.030821  0.017537  0.051423
281       0.104222  0.002516  0.054108  0.024325  0.035936 -0.005894  0.038060

[5 rows x 12563 columns]
```

In [ ]:
```
user_similarity_df
```

Out[ ]:

user_id	19	110	144	194	281	301	337	364	372	489	...	9
user_id												
19	1.000000	0.055390	0.082475	0.052292	0.008436	0.091234	0.101988	0.093516	0.127466	0.026647	...	0
110	0.055390	1.000000	-0.039201	0.059968	0.057699	-0.005846	0.060626	0.087973	0.031983	-0.022259	...	-(
144	0.082475	-0.039201	1.000000	-0.031112	-0.025069	0.021368	0.033775	0.001275	0.034263	0.041980	...	C
194	0.052292	0.059968	-0.031112	1.000000	0.136647	0.041423	0.055263	0.162218	-0.034590	-0.019204	...	-0
281	0.008436	0.057699	-0.025069	0.136647	1.000000	0.048269	-0.049340	0.126720	0.079098	0.037195	...	-0
...	...	...	...	...	...	...	...	...	...	...	...	
999799	0.099324	0.040699	0.050420	0.033521	0.054108	0.060124	0.113892	0.032156	0.008993	0.050281	...	0
999845	0.068719	0.002348	0.101318	0.005188	0.024325	0.079532	0.042707	-0.010867	0.035076	0.058930	...	0
999851	0.054825	0.044720	0.017539	0.030821	0.035936	0.126800	0.016192	0.111933	0.012182	0.000232	...	0
999869	0.035108	0.009803	0.178241	0.017537	-0.005894	0.067441	0.057670	0.012516	0.032242	0.072707	...	0
999901	0.112084	0.031383	0.077019	0.051423	0.038060	0.092930	0.018396	0.045890	0.095025	0.026879	...	C

12563 rows × 12563 columns

In [ ]:
```python
def predict_rating(target_user, target_movie, user_similarity_matrix, user_movie_matrix, K=5):
    """
    Predict the rating of a target_user for a target_movie based on similar users' ratings.
    """
    # Step 1: Get similarity scores for the target user
    similarities = user_similarity_matrix.loc[target_user]

    # Step 2: Extract ratings for the target movie from all users
    movie_ratings = user_movie_matrix[target_movie]

    # Step 3: Filter out users who haven't rated the target movie
    valid_users = movie_ratings[movie_ratings.notna()].index
    valid_similarities = similarities[valid_users]
    valid_ratings = movie_ratings[valid_users]

    # Step 4: Select top-K similar users
    top_k_users = valid_similarities.nlargest(K)

    # Step 5: Calculate the weighted average rating
    numerator = (top_k_users * valid_ratings[top_k_users.index]).sum()
    denominator = top_k_users.abs().sum()

    # Step 6: Return predicted rating or default value
    if denominator > 0:
        return numerator / denominator
```

```python
    else:
        return user_movie_matrix.loc[target_user].mean()  # Default to the user's mean rating
```

In [ ]:
```python
# Example inputs
target_user = user_movie_matrix.index[0]  # Replace with the desired user ID
target_movie = user_movie_matrix.columns[0]  # Replace with the desired movie ID

predicted_rating = predict_rating(target_user, target_movie, user_similarity_df, user_movie_matrix)
print(f"Predicted rating for user {target_user} on movie {target_movie}: {predicted_rating}")
```

```
Predicted rating for user 19 on movie 1: 1.0133675518654128
```

In [ ]:
```python
train_data[qualified_ratings['user_id']==19]
```

```
<ipython-input-52-6132d34bd993>:1: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  train_data[qualified_ratings['user_id']==19]
```

Out[ ]:

	item_id	user_id	rating
1211634	117	19	3.0
1211852	1276	19	5.0
1211689	357	19	3.0
1211724	524	19	4.0
1211716	480	19	3.0
...	...	...	...
1211705	434	19	4.0
1211713	474	19	3.0
1211621	45	19	3.0
1211722	509	19	3.0
1211718	491	19	4.0

229 rows × 3 columns

model evaluation

In [ ]:
```python
from tqdm import tqdm
import time

def safe_predict_rating(row):
    try:
        target_user = row['user_id']
        target_movie = row['item_id']

        return predict_rating(target_user, target_movie, user_similarity_df, user_movie_matrix)
    except Exception as e:
        print(f"Error predicting for user {row['user_id']} and movie {row['item_id']}: {e}")
        return default_value

start_time = time.time()

default_value = 3.0

tqdm.pandas(desc="Predicting Ratings")
train_data['predicted_rating'] = train_data.progress_apply(safe_predict_rating, axis=1)

end_time = time.time()

print(f"Predicting ratings completed in {end_time - start_time:.2f} seconds")
```

```
Predicting Ratings: 100%|██████████| 2401345/2401345 [1:55:26<00:00, 346.71it/s]
Predicting ratings completed in 6926.12 seconds
```

In [ ]:
```python
train_data
```

Out[ ]:

	item_id	user_id	rating	predicted_rating
2698127	4386	288742	1.0	-1.339137
4729044	1466	209015	3.0	-0.224275
4567052	1446	29651	4.0	0.453455
3360567	3307	711179	5.0	0.948223
3574627	6639	89881	4.5	0.640596
...	...	...	...	...
2806592	3730	861222	3.0	0.078697
3609440	11	270526	3.0	-0.421295
4627188	2135	529638	3.0	-0.449507
3549771	898	699790	5.0	0.767411
2380315	728	451757	5.0	0.890322

2401345 rows × 4 columns

In [ ]:

```
# Calculate the average rating per user
user_avg_rating = train_data.groupby('user_id')['rating'].mean()

# Map the average rating back to the DataFrame
train_data['average_rating_x'] = train_data['user_id'].map(user_avg_rating)
```

In [ ]:

```
#add average rating back
train_data.loc[:, 'restored_predicted_rating'] = (
    train_data['predicted_rating'] + train_data['average_rating_x']
)
```

In [ ]:

```
#debug: ValueError: Input contains NaN.
print(train_data[['rating', 'restored_predicted_rating']].isna().sum())
```

```
rating                       0
restored_predicted_rating    0
dtype: int64
```

In [ ]:

```
file_path = '/content/drive/My Drive/train_data_with_restored_data_new.csv'

train_data.to_csv(file_path, index=False)

print(f"File saved to Google Drive at: {file_path}")
```

```
File saved to Google Drive at: /content/drive/My Drive/train_data_with_restored_data.csv
```

In [ ]:
```
# mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [ ]:
```
import pandas as pd

file_path = '/content/drive/MyDrive/train_data_with_restored_data_new.csv'
train_data = pd.read_csv(file_path)
print("Data loaded successfully!")
```

```
Data loaded successfully!
```

In [ ]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
```

In [ ]:
```
cleaned_ratings = train_data.dropna(subset=['restored_predicted_rating'])

y_true = cleaned_ratings['rating']
y_pred = cleaned_ratings['restored_predicted_rating']

mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_true, y_pred)

print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

```
MSE: 0.1145
RMSE: 0.3384
MAE: 0.2612
```

In [ ]:
```
cleaned_ratings
```

Out[ ]:

	item_id	user_id	rating	predicted_rating	average_rating_x	restored_predicted_rating
0	4386	288742	1.0	-1.339137	3.266987	1.927850
1	1466	209015	3.0	-0.224275	3.151261	2.926985
2	1446	29651	4.0	0.453455	3.725248	4.178703
3	3307	711179	5.0	0.948223	3.921965	4.870189
4	6639	89881	4.5	0.640596	3.982219	4.622815
...	...	...	...	...	...	...
2401340	3730	861222	3.0	0.078697	3.236501	3.315198
2401341	11	270526	3.0	-0.421295	3.273973	2.852678
2401342	2135	529638	3.0	-0.449507	3.397727	2.948220
2401343	898	699790	5.0	0.767411	4.125000	4.892411
2401344	728	451757	5.0	0.890322	3.497265	4.387588

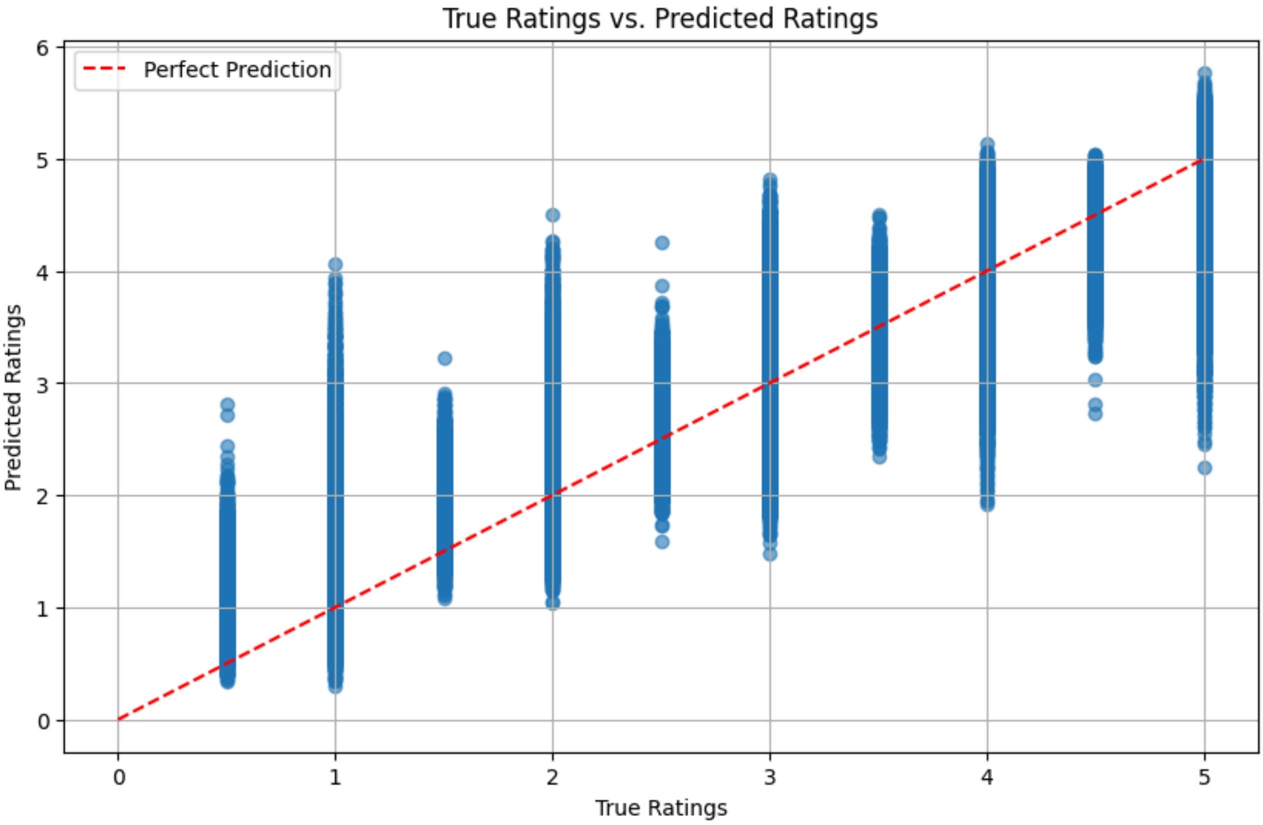
2401345 rows × 6 columns

In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

plt.scatter(y_true, y_pred, alpha=0.6)
plt.plot([0, 5], [0, 5], color='red', linestyle='--', label='Perfect Prediction')

plt.title("True Ratings vs. Predicted Ratings")
plt.xlabel("True Ratings")
plt.ylabel("Predicted Ratings")
plt.legend()
plt.grid()
plt.show()
```



In [ ]:
```
# Save the DataFrame as a .pkl file in Google Drive
file_path = '/content/drive/My Drive/cleaned_ratings_cosine_train.pkl'
cleaned_ratings.to_pickle(file_path)
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, c
all drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, cross
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardSc
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegre
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_s
import matplotlib.pyplot as plt
import seaborn as sns
import os
import json
from tqdm import tqdm
```

In [ ]:
```
base_dir = "/content/drive/MyDrive/movie_dataset_public_final/raw/"

def load_ndjson_file(file_name, limit=None):
    data = []
    file_path = os.path.join(base_dir, file_name)

    with open(file_path, 'r', encoding='utf-8') as f:
        for i, line in enumerate(f):
            if limit and i >= limit:
                break
            try:
                data.append(json.loads(line))
            except json.JSONDecodeError as e:
                print(f"Error parsing line {i}: {line}")
                print(f"Error: {e}")
    return data

metadata = load_ndjson_file("metadata.json")
```

In [ ]:
```
metadata_updated = load_ndjson_file("metadata_updated.json")
metadata = pd.DataFrame(metadata_updated)
print(metadata.head())
```

```
                             title       directedBy  \
0                 Toy Story (1995)    John Lasseter   
1                   Jumanji (1995)     Joe Johnston   
2          Grumpier Old Men (1995)    Howard Deutch   
3         Waiting to Exhale (1995)  Forest Whitaker   
4  Father of the Bride Part II (1995)    Charles Shyer   

                                            starring  avgRating   imdbId  \
0  Tim Allen, Tom Hanks, Don Rickles, Jim Varney,...    3.89146  0114709   
1  Jonathan Hyde, Bradley Pierce, Robin Williams,...    3.26605  0113497   
2  Jack Lemmon, Walter Matthau, Ann-Margret , Sop...    3.17146  0113228   
3  Angela Bassett, Loretta Devine, Whitney Housto...    2.86824  0114885   
4  Steve Martin, Martin Short, Diane Keaton, Kimb...    3.07620  0113041   

   item_id  
0        1  
1        2  
2        3  
3        4  
4        5  
```

In [ ]:
```python
metadata['release_year'] = metadata['title'].str.extract(r'\((\d{4})\)',
metadata['movie_title'] = metadata['title'].str.replace(r'\(\d{4}\)', '',

metadata = metadata.drop(columns=['title'])

print(metadata.head())
```

```
        directedBy                                           starring  \
0    John Lasseter  Tim Allen, Tom Hanks, Don Rickles, Jim Varney,...   
1     Joe Johnston  Jonathan Hyde, Bradley Pierce, Robin Williams,...   
2    Howard Deutch  Jack Lemmon, Walter Matthau, Ann-Margret , Sop...   
3  Forest Whitaker  Angela Bassett, Loretta Devine, Whitney Housto...   
4    Charles Shyer  Steve Martin, Martin Short, Diane Keaton, Kimb...   

  avgRating   imdbId  item_id release_year                  movie_title  
0   3.89146  0114709        1         1995                    Toy Story  
1   3.26605  0113497        2         1995                      Jumanji  
2   3.17146  0113228        3         1995             Grumpier Old Men  
3   2.86824  0114885        4         1995            Waiting to Exhale  
4   3.07620  0113041        5         1995  Father of the Bride Part II  
```

In [ ]:
```python
# Handle missing values
metadata['directedBy'].fillna('Unknown', inplace=True)

metadata['directedBy'] = metadata['directedBy'].str.strip()
metadata['directedBy'] = metadata['directedBy'].str.replace(r"\s+", " ",

metadata['directedBy'] = metadata['directedBy'].str.replace("&", ",")  #
metadata['directedBy'] = metadata['directedBy'].str.split(",").apply(
    lambda x: [d.strip() for d in x if d.strip()] if isinstance(x, list)
)

metadata['directedBy'] = metadata['directedBy'].apply(lambda x: ", ".join
```

```
<ipython-input-6-2ec3f22f36bd>:2: FutureWarning: A value is trying to be s
et on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never wor
k because the intermediate object on which we are setting values always be
haves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  metadata['directedBy'].fillna('Unknown', inplace=True)
```

In [ ]:

```
tag_count = load_ndjson_file("tag_count.json")
tags = load_ndjson_file("tags.json")
```

In [ ]:

```
tags = pd.DataFrame(tags)
tag_count = pd.DataFrame(tag_count)

tags_joined = pd.merge(tags, tag_count, left_on='id', right_on='tag_id',

tags_joined = tags_joined.drop(columns=['id'])

print(tags_joined.head())
```

```
       tag  item_id  tag_id  num
0  aardman      720      22    5
1  aardman      745      22   22
2  aardman     1148      22   19
3  aardman     1223      22    9
4  aardman     3429      22    3
```

In [ ]:

```
tags_grouped = tags_joined.groupby('item_id')['tag'].apply(list).reset_in
print(tags_grouped.head())
```

```
   item_id                                                tag
0        1  [nostalgic, interesting, children, witty, emot...
1        2  [children, animals, based on a book, comedy, a...
2        3  [sequel, good soundtrack, comedy, funny, funni...
3        4                         [divorce, revenge, chick flick]
4        5  [remake, touching, sequel, comedy, pregnancy, ...
```

In [ ]:

```
sentiments_df = pd.read_csv('/content/drive/MyDrive/movie_dataset_public_


merged_data = pd.merge(metadata, sentiments_df, on='item_id', how='inner'
merged_df = pd.merge(merged_data, tags_grouped, on='item_id', how='inner'
print(merged_df.head())
```

```
          directedBy                                              starring  \
0     John Lasseter  Tim Allen, Tom Hanks, Don Rickles, Jim Varney,...
1      Joe Johnston  Jonathan Hyde, Bradley Pierce, Robin Williams,...
2     Howard Deutch  Jack Lemmon, Walter Matthau, Ann-Margret , Sop...
3   Forest Whitaker  Angela Bassett, Loretta Devine, Whitney Housto...
4     Charles Shyer  Steve Martin, Martin Short, Diane Keaton, Kimb...

   avgRating   imdbId  item_id release_year                  movie_title
\
0    3.89146  0114709        1         1995                    Toy Story
1    3.26605  0113497        2         1995                      Jumanji
2    3.17146  0113228        3         1995             Grumpier Old Men
3    2.86824  0114885        4         1995            Waiting to Exhale
4    3.07620  0113041        5         1995  Father of the Bride Part II

   avg_negative  avg_neutral  avg_positive  avg_compound  \
0      0.042831     0.757843      0.199306      0.904176
1      0.064142     0.746503      0.189320      0.799850
2      0.066300     0.744933      0.188833      0.835292
3      0.080184     0.751204      0.168429      0.513924
4      0.049833     0.740121      0.210000      0.825379

                                                 tag
0  [nostalgic, interesting, children, witty, emot...
1  [children, animals, based on a book, comedy, a...
2  [sequel, good soundtrack, comedy, funny, funni...
3                        [divorce, revenge, chick flick]
4  [remake, touching, sequel, comedy, pregnancy, ...
```

In [ ]:

```python
file_path = '/content/drive/MyDrive/movie_dataset_public_final/qualified_
try:
  df = pd.read_csv(file_path)
  print("File imported successfully.")
  # Now you can work with the DataFrame 'df'
except FileNotFoundError:
  print(f"Error: File not found at {file_path}")
except Exception as e:
  print(f"An error occurred: {e}")
```

```
File imported successfully.
```

In [ ]:

```python
movies = df.item_id.unique()
```

In [ ]:

```python
filtered_merged_df = merged_df[merged_df['item_id'].isin(movies)]
print(filtered_merged_df.head())
```

```
        directedBy                                            starring  \
0    John Lasseter  Tim Allen, Tom Hanks, Don Rickles, Jim Varney,...
1     Joe Johnston  Jonathan Hyde, Bradley Pierce, Robin Williams,...
2    Howard Deutch  Jack Lemmon, Walter Matthau, Ann-Margret , Sop...
3  Forest Whitaker  Angela Bassett, Loretta Devine, Whitney Housto...
4    Charles Shyer  Steve Martin, Martin Short, Diane Keaton, Kimb...

  avgRating   imdbId  item_id release_year                  movie_title
\
0   3.89146  0114709        1         1995                    Toy Story
1   3.26605  0113497        2         1995                      Jumanji
2   3.17146  0113228        3         1995             Grumpier Old Men
3   2.86824  0114885        4         1995            Waiting to Exhale
4   3.07620  0113041        5         1995  Father of the Bride Part II

  avg_negative  avg_neutral  avg_positive  avg_compound  \
0     0.042831     0.757843      0.199306      0.904176
1     0.064142     0.746503      0.189320      0.799850
2     0.066300     0.744933      0.188833      0.835292
3     0.080184     0.751204      0.168429      0.513924
4     0.049833     0.740121      0.210000      0.825379

                                                 tag
0  [nostalgic, interesting, children, witty, emot...
1  [children, animals, based on a book, comedy, a...
2  [sequel, good soundtrack, comedy, funny, funni...
3                        [divorce, revenge, chick flick]
4  [remake, touching, sequel, comedy, pregnancy, ...
```

In [ ]:

```python
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Limit to top 50 directors
top_directors = filtered_merged_df['directedBy'].value_counts().nlargest(
filtered_merged_df['directedBy'] = filtered_merged_df['directedBy'].apply

director_encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=F
directors_encoded = director_encoder.fit_transform(filtered_merged_df[['d
director_feature_names = director_encoder.get_feature_names_out(['directe

print(f"Number of encoded features: {directors_encoded.shape[1]}")
print("Feature names:", director_feature_names[:5], "...")

directors_df = pd.DataFrame(directors_encoded, columns=director_feature_n

filtered_merged_df = pd.concat([filtered_merged_df, directors_df], axis=1
filtered_merged_df.drop('directedBy', axis=1, inplace=True)
```

```
Number of encoded features: 51
Feature names: ['directedBy_Alan Parker' 'directedBy_Alfred Hitchcock'
 'directedBy_Ang Lee' 'directedBy_Barry Levinson'
 'directedBy_Billy Wilder'] ...
```

In [ ]:

```python
filtered_merged_df.shape
```

Out[ ]: (3803, 62)

In [ ]:

```python
from sklearn.preprocessing import MultiLabelBinarizer

filtered_merged_df['starring'] = filtered_merged_df['starring'].apply(lam
```

```
# Limit to top 50 actors
all_actors = filtered_merged_df.explode('starring')['starring']
top_actors = all_actors.value_counts().nlargest(50).index
filtered_merged_df['starring'] = filtered_merged_df['starring'].apply(lam

mlb = MultiLabelBinarizer()
actors_encoded = mlb.fit_transform(filtered_merged_df['starring'])
actor_feature_names = mlb.classes_
actors_df = pd.DataFrame(actors_encoded, columns=actor_feature_names, ind

filtered_merged_df = pd.concat([filtered_merged_df, actors_df], axis=1)
filtered_merged_df.drop('starring', axis=1, inplace=True)
```

In [ ]:
```
filtered_merged_df['tag'] = filtered_merged_df['tag'].apply(lambda x: x.s

# Step 1: Limit to Top 100 Tags
all_tags = filtered_merged_df.explode('tag')['tag']
top_tags = all_tags.value_counts().nlargest(100).index
filtered_merged_df['tag'] = filtered_merged_df['tag'].apply(lambda x: [ta


mlb = MultiLabelBinarizer()
tags_encoded = mlb.fit_transform(filtered_merged_df['tag'])
tag_feature_names = mlb.classes_

tags_df = pd.DataFrame(tags_encoded, columns=tag_feature_names, index=fil

filtered_merged_df = pd.concat([filtered_merged_df, tags_df], axis=1)

filtered_merged_df.drop('tag', axis=1, inplace=True)
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=100)

title_tfidf = tfidf.fit_transform(filtered_merged_df['movie_title'])

tfidf_df = pd.DataFrame(title_tfidf.toarray(), columns=tfidf.get_feature_

filtered_merged_df = pd.concat([filtered_merged_df, tfidf_df], axis=1)
filtered_merged_df.drop('movie_title', axis=1, inplace=True)
```

In [ ]:
```
filtered_merged_df['release_year'] = pd.to_numeric(filtered_merged_df['re

filtered_merged_df['release_year'].fillna(0, inplace=True)

filtered_merged_df['release_year'] = filtered_merged_df['release_year'].a
```

```
<ipython-input-19-f1e9d019446f>:6: FutureWarning: A value is trying to be
set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never wor
k because the intermediate object on which we are setting values always be
haves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  filtered_merged_df['release_year'].fillna(0, inplace=True)
```

In [ ]:

```
filtered_merged_df['release_decade'] = (filtered_merged_df['release_year'

# One-Hot Encode release decade
decade_encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=Fal
decades_encoded = decade_encoder.fit_transform(filtered_merged_df[['relea
decade_feature_names = decade_encoder.get_feature_names_out(['release_dec
decades_df = pd.DataFrame(decades_encoded, columns=decade_feature_names,

# Concatenate and drop original columns
filtered_merged_df = pd.concat([filtered_merged_df, decades_df], axis=1)
filtered_merged_df.drop(['release_year', 'release_decade'], axis=1, inpla
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler


numerical_features = ['avg_negative', 'avg_neutral', 'avg_positive', 'avg

scaler = StandardScaler()

filtered_merged_df[numerical_features] = scaler.fit_transform(filtered_me
```

In [ ]:

```
train_data = pd.read_csv('/content/drive/MyDrive/movie_dataset_public_fin
print(train_data.head())
```

```
   item_id  user_id  rating
0     4386   288742     1.0
1     1466   209015     3.0
2     1446    29651     4.0
3     3307   711179     5.0
4     6639    89881     4.5
```

In [ ]:

```
cb_training_data = pd.merge(train_data, filtered_merged_df, on='item_id',

print(cb_training_data.head())
```

```
   item_id  user_id  rating  avgRating   imdbId  avg_negative  avg_neutral  \
0     4386   288742     1.0    2.55868  0239395      0.443117    -0.832400
1     1466   209015     3.0    3.80767  0119008     -0.816323     0.713760
2     1446    29651     4.0    4.00348  0116790     -1.177937     0.225011
3     3307   711179     5.0    4.10615  0021749     -1.202262    -0.696967
4     6639    89881     4.5    3.92525  0062467      1.024506    -0.291457

   avg_positive  avg_compound  directedBy_Alan Parker  ...  \
0      0.314369     -0.409147                     0.0  ...
1      0.102293      0.648067                     0.0  ...
2      0.819075      1.036453                     0.0  ...
3      1.609711      1.298728                     0.0  ...
4     -0.636241     -0.754227                     0.0  ...

   release_decade_1920  release_decade_1930  release_decade_1940  \
0                  0.0                  0.0                  0.0
1                  0.0                  0.0                  0.0
2                  0.0                  0.0                  0.0
3                  0.0                  1.0                  0.0
4                  0.0                  0.0                  0.0

   release_decade_1950  release_decade_1960  release_decade_1970  \
0                  0.0                  0.0                  0.0
1                  0.0                  0.0                  0.0
2                  0.0                  0.0                  0.0
3                  0.0                  0.0                  0.0
4                  0.0                  1.0                  0.0

   release_decade_1980  release_decade_1990  release_decade_2000  \
0                  0.0                  0.0                  1.0
1                  0.0                  1.0                  0.0
2                  0.0                  1.0                  0.0
3                  0.0                  0.0                  0.0
4                  0.0                  0.0                  0.0

   release_decade_2010
0                  0.0
1                  0.0
2                  0.0
3                  0.0
4                  0.0

[5 rows x 322 columns]
```

In [ ]:

```python
print(cb_training_data.columns)
```

```
Index(['item_id', 'user_id', 'rating', 'avgRating', 'imdbId', 'avg_negativ
e',
       'avg_neutral', 'avg_positive', 'avg_compound', 'directedBy_Alan Par
ker',
       ...
       'release_decade_1920', 'release_decade_1930', 'release_decade_194
0',
       'release_decade_1950', 'release_decade_1960', 'release_decade_197
0',
       'release_decade_1980', 'release_decade_1990', 'release_decade_200
0',
       'release_decade_2010'],
      dtype='object', length=322)
```

In [ ]:
```
duplicate_columns = cb_training_data.columns[cb_training_data.columns.dup
print(f"Duplicate columns: {duplicate_columns}")
```

```
Duplicate columns: Index(['Other', 'blood', 'children', 'death', 'love',
'story', 'war'], dtype='object')
```

In [ ]:
```
cb_training_data = cb_training_data.loc[:, ~cb_training_data.columns.dupl
```

In [ ]:
```
assert cb_training_data.columns.is_unique, "Duplicate columns still exist
```

In [ ]:
```
cb_training_data.columns = cb_training_data.columns.str.replace('[^A-Za-z
```

In [ ]:
```
assert cb_training_data.columns.is_unique, "Duplicate column names still
```

In [ ]:
```
import joblib
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
import xgboost as xgb
from tqdm import tqdm
import numpy as np

X = cb_training_data.drop(columns=['imdbId', 'rating', 'avgRating', 'item
y = cb_training_data['rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

xgb_model = xgb.XGBRegressor(
    n_estimators=2000,
    learning_rate=0.05,
    tree_method='hist',
    device='cuda',
    random_state=42
)

print("Training XGBoost model...")

with tqdm(total=xgb_model.n_estimators, desc="Training Progress", unit="i
    def update_progress(env):
        """Update tqdm progress bar."""
        pbar.update(1)

    xgb_model.fit(
        X_train,
        y_train,
        eval_set=[(X_test, y_test)],
        verbose=False,
    )

# Predict and evaluate
y_pred = xgb_model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"CB Model - XGBoost:")
print(f"MAE: {mae:.4f}, RMSE: {rmse:.4f}")

# Save the model
```

```
joblib.dump(xgb_model, '/content/drive/MyDrive/movie_dataset_public_final
print("XGBoost model saved as xgb_cb_model.pkl")
```

```
Training XGBoost model...
Training Progress:   0%|          | 0/2000 [02:19<?, ?iteration/s]
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[04:55:55] WARNING: /workspace/src/common/error_msg.cc:58: Falling back to
prediction using DMatrix due to mismatched devices. This might lead to hig
her memory usage and slower performance. XGBoost is running on: cuda:0, wh
ile the input data is on: cpu.
Potential solutions:
- Use a data structure that matches the device ordinal in the booster.
- Set the device for booster before call to inplace_predict.

This warning will only be shown once.

  warnings.warn(smsg, UserWarning)
CB Model - XGBoost:
MAE: 0.7443, RMSE: 0.9422
XGBoost model saved as xgb_cb_model.pkl
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [2000],
    'max_depth': [4, 6, 8],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}


xgb_model = xgb.XGBRegressor(tree_method='gpu_hist', random_state=42)

grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=5,
    verbose=1,
    n_jobs=-1
)

print("Running GridSearchCV...")
grid_search.fit(X_train, y_train)


best_model = grid_search.best_estimator_
print("Best parameters:", grid_search.best_params_)

y_pred = best_model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Tuned CB Model - XGBoost (CV): MAE: {mae:.4f}, RMSE: {rmse:.4f}")

joblib.dump(best_model, '/content/drive/MyDrive/movie_dataset_public_fina
print("Tuned XGBoost model saved as xgb_cb_model_tuned.pkl")
```

```
Running GridSearchCV...
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

In [ ]:

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

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, c
all drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
import os

content_model_path = '/content/drive/MyDrive/movie_dataset_public_final/x
cf_predictions_path = '/content/drive/MyDrive/movie_dataset_public_final/


if os.path.exists(content_model_path):
    print("Content model file found:", content_model_path)
else:
    print("Content model file not found!")

if os.path.exists(cf_predictions_path):
    print("CF predictions file found:", cf_predictions_path)
else:
    print("CF predictions file not found!")
```

```
Content model file found: /content/drive/MyDrive/movie_dataset_public_fina
l/xgb_cb_model.pkl
CF predictions file found: /content/drive/MyDrive/movie_dataset_public_fin
al/cleaned_ratings_cosine_train.pkl
```

In [ ]:
```
import joblib

content_model = joblib.load(content_model_path)


cf_predictions = pd.read_pickle(cf_predictions_path)
```

In [ ]:
```
import pandas as pd


train_data_path = '/content/drive/MyDrive/movie_dataset_public_final/filt

train_data = pd.read_csv(train_data_path)
```

In [ ]:
```
train_data
```

Out[ ]:

	item_id	user_id	rating
0	5	997206	3.0
1	10	997206	4.0
2	13	997206	4.0
3	17	997206	5.0
4	21	997206	4.0
...	...	...	...
23514990	97938	187144	3.0
23514991	98809	187144	3.0
23514992	99114	187144	3.0
23514993	102445	187144	4.0
23514994	104841	187144	3.0

23514995 rows × 3 columns

In [ ]:
```
train_data_sample = train_data.sample(n=10000, random_state=42)
train_data_sample
```

Out[ ]:

	item_id	user_id	rating
9507762	3996	306188	3.0
4777740	2827	939224	3.0
1982187	592	41489	3.0
4015247	930	347251	5.0
21325961	48394	552153	5.0
...	...	...	...
7099488	337	702670	4.0
23131247	3753	242955	3.0
16881896	7386	48864	4.5
19204730	91500	906673	3.0
13463269	2124	138135	4.0

10000 rows × 3 columns

In [ ]:
```
base_dir = "/content/drive/MyDrive/movie_dataset_public_final/raw/"

def load_ndjson_file(file_name, limit=None):
    data = []
    file_path = os.path.join(base_dir, file_name)

    with open(file_path, 'r', encoding='utf-8') as f:
        for i, line in enumerate(f):
```

```
            if limit and i >= limit:
                break
            try:
                data.append(json.loads(line))
            except json.JSONDecodeError as e:
                print(f"Error parsing line {i}: {line}")
                print(f"Error: {e}")
    return data

metadata = load_ndjson_file("metadata.json")
```

In [ ]:
```
metadata_updated = load_ndjson_file("metadata_updated.json")
metadata = pd.DataFrame(metadata_updated)
print(metadata.head())
```

```
                                title       directedBy  \
0                    Toy Story (1995)    John Lasseter
1                      Jumanji (1995)     Joe Johnston
2             Grumpier Old Men (1995)    Howard Deutch
3            Waiting to Exhale (1995)  Forest Whitaker
4  Father of the Bride Part II (1995)    Charles Shyer

                                            starring  avgRating   imdbId
\
0  Tim Allen, Tom Hanks, Don Rickles, Jim Varney,...    3.89146  0114709
1  Jonathan Hyde, Bradley Pierce, Robin Williams,...    3.26605  0113497
2  Jack Lemmon, Walter Matthau, Ann-Margret , Sop...    3.17146  0113228
3  Angela Bassett, Loretta Devine, Whitney Housto...    2.86824  0114885
4  Steve Martin, Martin Short, Diane Keaton, Kimb...    3.07620  0113041

   item_id
0        1
1        2
2        3
3        4
4        5
```

In [ ]:
```
movie_features_path = '/content/drive/MyDrive/movie_dataset_public_final/

movie_features = pd.read_csv(movie_features_path)
```

In [ ]:
```
movie_features
movie_features= movie_features.drop(columns=['avgRating','imdbId'])
```

In [ ]:
```
movie_features
```

Out[ ]:

	item_id	avg_negative	avg_neutral	avg_positive	avg_compound	directedB
0	1	-1.747673	0.447609	1.122360	1.293096	
1	2	-0.797492	-0.071923	0.741227	0.891817	
2	3	-0.701273	-0.143832	0.722673	1.028139	
3	4	-0.082240	0.143450	-0.056040	-0.207967	
4	5	-1.435475	-0.364290	1.530462	0.990010	
...	...	...	...	...	...	
3798	143385	-0.319489	1.003223	-0.562908	0.343855	
3799	148626	0.477404	0.636112	-0.937548	-0.406321	
3800	164179	-0.179858	1.577683	-1.161556	-0.234032	
3801	166528	1.903981	-1.282985	-0.560504	-1.134761	
3802	166643	-1.066447	1.285089	-0.157634	0.866084	

3803 rows × 318 columns

In [ ]:
```
trained_feature_names = content_model.feature_names_in_
```

In [ ]:
```
trained_feature_names
```

```
Out[ ]: array(['user_id', 'avg_negative', 'avg_neutral', 'avg_positive',
               'avg_compound', 'directedBy_Alfred_Hitchcock',
               'directedBy_Barry_Levinson', 'directedBy_Clint_Eastwood',
               'directedBy_Joel_Schumacher', 'directedBy_Martin_Scorsese',
               'directedBy_Mike_Nichols', 'directedBy_Norman_Jewison',
               'directedBy_Oliver_Stone', 'directedBy_Other',
               'directedBy_Richard_Donner', 'directedBy_Ridley_Scott',
               'directedBy_Rob_Reiner', 'directedBy_Robert_Altman',
               'directedBy_Robert_Stevenson', 'directedBy_Ron_Howard',
               'directedBy_Sidney_Lumet', 'directedBy_Spike_Lee',
               'directedBy_Stanley_Kubrick', 'directedBy_Steven_Soderbergh',
               'directedBy_Steven_Spielberg', 'directedBy_Woody_Allen', '',
               'Bill_Murray', 'Bruce_Willis', 'Christopher_Walken',
               'Donald_Sutherland', 'Dustin_Hoffman', 'Gene_Hackman',
               'Harvey_Keitel', 'Jack_Nicholson', 'Julianne_Moore',
               'Meryl_Streep', 'Michael_Caine', 'Morgan_Freeman', 'Other',
               'Philip_Seymour_Hoffman', 'Robert_De_Niro', 'Robert_Duvall',
               'Robin_Williams', 'Samuel_L_Jackson', 'Sean_Connery', 'Tom_Hank
        s',
               '1980s', '70mm', 'action', 'adapted_from_book', 'adultery',
               'adventure', 'atmospheric', 'based_on_a_book',
               'based_on_a_true_story', 'betrayal', 'black_and_white',
               'black_comedy', 'blood', 'boring', 'chase', 'children',
               'cinematography', 'classic', 'comedy', 'coming_of_age',
               'corruption', 'crime', 'criterion', 'cult_film', 'dancing', 'dar
        k',
               'dark_comedy', 'death', 'dialogue', 'directorial_debut', 'disne
        y',
               'dog', 'drama', 'drugs', 'england', 'family', 'fantasy',
               'father_daughter_relationship', 'father_son_relationship',
               'franchise', 'friendship', 'fun', 'funny', 'great_acting',
               'great_soundtrack', 'high_school', 'hilarious', 'history',
               'horror', 'humorous', 'imdb_top_250', 'independent_film',
               'infidelity', 'investigation', 'long', 'los_angeles', 'love',
               'marriage', 'money', 'murder', 'music', 'musical', 'mystery',
               'new_york', 'new_york_city', 'nudity', 'nudity_topless_brief_',
               'nudity_topless_', 'overrated', 'period_piece', 'police',
               'politics', 'predictable', 'prison', 'quirky', 'revenge',
               'romance', 'satire', 'sci_fi', 'sequel', 'serial_killer', 'sex',
               'sexuality', 'silly', 'slow', 'small_town', 'social_commentary',
               'story', 'stupid', 'stylized', 'suicide', 'surreal', 'suspense',
               'tense', 'thriller', 'true_story', 'violence',
               'visually_appealing', 'war', 'witty', 'about', 'adventures', 'al
        l',
               'america', 'american', 'an', 'and', 'at', 'baby', 'bad', 'big',
               'black', 'blue', 'boys', 'bride', 'by', 'cat', 'city', 'day',
               'days', 'de', 'dead', 'der', 'die', 'down', 'fire', 'first', 'fo
        r',
               'friday', 'from', 'girl', 'good', 'great', 'harry', 'heart',
               'high', 'home', 'house', 'ii', 'iii', 'in', 'is', 'it', 'kid',
               'king', 'kiss', 'la', 'last', 'le', 'legend', 'les', 'life',
               'little', 'lost', 'mad', 'man', 'me', 'men', 'movie', 'mr', 'mr
        s',
               'my', 'new', 'night', 'no', 'of', 'on', 'one', 'out', 'part',
               'planet', 'red', 'return', 'road', 'secret', 'star', 'street',
               'that', 'the', 'thing', 'three', 'time', 'to', 'trek', 'two', 'u
        p',
               'what', 'white', 'who', 'wild', 'with', 'world', 'you',
               'release_decade_1920', 'release_decade_1930',
               'release_decade_1940', 'release_decade_1950',
```

```
       'release_decade_1960', 'release_decade_1970',
       'release_decade_1980', 'release_decade_1990',
       'release_decade_2000', 'release_decade_2010'], dtype='<U28')
```

In [ ]:
```python
def get_collab_prediction(user_id, item_id, cf_data):
    """
    Retrieve the collaborative filtering prediction for a given user and
    """
    try:
        prediction = cf_data.loc[(cf_data['user_id'] == user_id) & (cf_da
    except IndexError:
        prediction = cf_data['restored_predicted_rating'].mean()
    return prediction
```

In [ ]:
```python
trained_feature_names = content_model.feature_names_in_
print("Feature names in trained model:", trained_feature_names)
print("Columns in movie_features:", movie_features.columns)

valid_trained_feature_names = [name for name in trained_feature_names if

missing_columns = [col for col in trained_feature_names if col not in mov
for col in missing_columns:
    movie_features[col] = 0


filtered_movie_features = movie_features[valid_trained_feature_names]
print("Filtered movie_features columns:", filtered_movie_features.columns
```

```
Feature names in trained model: ['user_id' 'avg_negative' 'avg_neutral' 'a
vg_positive' 'avg_compound'
 'directedBy_Alfred_Hitchcock' 'directedBy_Barry_Levinson'
 'directedBy_Clint_Eastwood' 'directedBy_Joel_Schumacher'
 'directedBy_Martin_Scorsese' 'directedBy_Mike_Nichols'
 'directedBy_Norman_Jewison' 'directedBy_Oliver_Stone' 'directedBy_Other'
 'directedBy_Richard_Donner' 'directedBy_Ridley_Scott'
 'directedBy_Rob_Reiner' 'directedBy_Robert_Altman'
 'directedBy_Robert_Stevenson' 'directedBy_Ron_Howard'
 'directedBy_Sidney_Lumet' 'directedBy_Spike_Lee'
 'directedBy_Stanley_Kubrick' 'directedBy_Steven_Soderbergh'
 'directedBy_Steven_Spielberg' 'directedBy_Woody_Allen' '' 'Bill_Murray'
 'Bruce_Willis' 'Christopher_Walken' 'Donald_Sutherland' 'Dustin_Hoffman'
 'Gene_Hackman' 'Harvey_Keitel' 'Jack_Nicholson' 'Julianne_Moore'
 'Meryl_Streep' 'Michael_Caine' 'Morgan_Freeman' 'Other'
 'Philip_Seymour_Hoffman' 'Robert_De_Niro' 'Robert_Duvall'
 'Robin_Williams' 'Samuel_L_Jackson' 'Sean_Connery' 'Tom_Hanks' '1980s'
 '70mm' 'action' 'adapted_from_book' 'adultery' 'adventure' 'atmospheric'
 'based_on_a_book' 'based_on_a_true_story' 'betrayal' 'black_and_white'
 'black_comedy' 'blood' 'boring' 'chase' 'children' 'cinematography'
 'classic' 'comedy' 'coming_of_age' 'corruption' 'crime' 'criterion'
 'cult_film' 'dancing' 'dark' 'dark_comedy' 'death' 'dialogue'
 'directorial_debut' 'disney' 'dog' 'drama' 'drugs' 'england' 'family'
 'fantasy' 'father_daughter_relationship' 'father_son_relationship'
 'franchise' 'friendship' 'fun' 'funny' 'great_acting' 'great_soundtrack'
 'high_school' 'hilarious' 'history' 'horror' 'humorous' 'imdb_top_250'
 'independent_film' 'infidelity' 'investigation' 'long' 'los_angeles'
 'love' 'marriage' 'money' 'murder' 'music' 'musical' 'mystery' 'new_york'
 'new_york_city' 'nudity' 'nudity_topless_brief_' 'nudity_topless_'
 'overrated' 'period_piece' 'police' 'politics' 'predictable' 'prison'
 'quirky' 'revenge' 'romance' 'satire' 'sci_fi' 'sequel' 'serial_killer'
 'sex' 'sexuality' 'silly' 'slow' 'small_town' 'social_commentary' 'story'
 'stupid' 'stylized' 'suicide' 'surreal' 'suspense' 'tense' 'thriller'
 'true_story' 'violence' 'visually_appealing' 'war' 'witty' 'about'
 'adventures' 'all' 'america' 'american' 'an' 'and' 'at' 'baby' 'bad'
 'big' 'black' 'blue' 'boys' 'bride' 'by' 'cat' 'city' 'day' 'days' 'de'
 'dead' 'der' 'die' 'down' 'fire' 'first' 'for' 'friday' 'from' 'girl'
 'good' 'great' 'harry' 'heart' 'high' 'home' 'house' 'ii' 'iii' 'in' 'is'
 'it' 'kid' 'king' 'kiss' 'la' 'last' 'le' 'legend' 'les' 'life' 'little'
 'lost' 'mad' 'man' 'me' 'men' 'movie' 'mr' 'mrs' 'my' 'new' 'night' 'no'
 'of' 'on' 'one' 'out' 'part' 'planet' 'red' 'return' 'road' 'secret'
 'star' 'street' 'that' 'the' 'thing' 'three' 'time' 'to' 'trek' 'two'
 'up' 'what' 'white' 'who' 'wild' 'with' 'world' 'you'
 'release_decade_1920' 'release_decade_1930' 'release_decade_1940'
 'release_decade_1950' 'release_decade_1960' 'release_decade_1970'
 'release_decade_1980' 'release_decade_1990' 'release_decade_2000'
 'release_decade_2010']
Columns in movie_features: Index(['item_id', 'avg_negative', 'avg_neutra
l', 'avg_positive',
       'avg_compound', 'directedBy_Alan Parker', 'directedBy_Alfred Hitchc
ock',
       'directedBy_Ang Lee', 'directedBy_Barry Levinson',
       'directedBy_Billy Wilder',
       ...
       'release_decade_1920', 'release_decade_1930', 'release_decade_194
0',
       'release_decade_1950', 'release_decade_1960', 'release_decade_197
0',
       'release_decade_1980', 'release_decade_1990', 'release_decade_200
0',
```

```
       'release_decade_2010'],
      dtype='object', length=318)
Filtered movie_features columns: Index(['avg_negative', 'avg_neutral', 'av
g_positive', 'avg_compound',
       'directedBy_Other', 'Other', '1980s', '70mm', 'action', 'adultery',
       ...
       'release_decade_1920', 'release_decade_1930', 'release_decade_194
0',
       'release_decade_1950', 'release_decade_1960', 'release_decade_197
0',
       'release_decade_1980', 'release_decade_1990', 'release_decade_200
0',
       'release_decade_2010'],
      dtype='object', length=181)
```

In [ ]:

```python
def get_content_prediction(item_id, movie_features, content_model):
    """
    Predict the average rating of a movie using the content-based model.
    """

    features = movie_features.loc[movie_features['item_id'] == item_id].d

    features = features[content_model.feature_names_in_]
    return content_model.predict(features)[0]
```

In [ ]:

```python
def predict_hybrid_score(user_id, item_id, cf_data, movie_features, conte
    """
    Combine predictions from content-based and collaborative filtering mo
    """
    content_pred = get_content_prediction(item_id, movie_features, conten

    collab_pred = get_collab_prediction(user_id, item_id, cf_data)

    beta = 1 - alpha
    return alpha * content_pred + beta * collab_pred
```

In [ ]:

```python
filtered_train_data = train_data_sample[train_data_sample['item_id'].isin

print(f"Original train_data size: {len(train_data_sample)}")
print(f"Filtered train_data size: {len(filtered_train_data)}")
```

```
Original train_data size: 10000
Filtered train_data size: 9975
```

In [ ]:

```python
from sklearn.metrics import mean_squared_error
import numpy as np

from sklearn.metrics import mean_squared_error
import numpy as np

def compute_mse_rmse(alpha, test_data, cf_data, movie_features, content_m
    """
    Compute MSE and RMSE for test samples, considering two cases:
    1. Samples with only CB prediction.
    2. Samples with both CB and CF predictions.
    """
    true_ratings_cb_only = []
    cb_only_predictions = []
    true_ratings_hybrid = []
    hybrid_predictions = []
```

```
    for _, row in test_data.iterrows():
        user_id = row['user_id']
        item_id = row['item_id']
        true_rating = row['rating']


        content_pred = get_content_prediction(item_id, movie_features, co

        try:
            collab_pred = get_collab_prediction(user_id, item_id, cf_data
        except KeyError:
            collab_pred = None

        if collab_pred is None:

            true_ratings_cb_only.append(true_rating)
            cb_only_predictions.append(content_pred)
        else:

            hybrid_pred = alpha * content_pred + (1 - alpha) * collab_pre
            true_ratings_hybrid.append(true_rating)
            hybrid_predictions.append(hybrid_pred)


    mse_cb_only = mean_squared_error(true_ratings_cb_only, cb_only_predic
    mse_hybrid = mean_squared_error(true_ratings_hybrid, hybrid_predictio


    total_samples = len(test_data)
    mse = (mse_cb_only * len(true_ratings_cb_only) + mse_hybrid * len(tru
    rmse = np.sqrt(mse)
    return mse, rmse
```

In [ ]:

```
from tqdm import tqdm


sample_train_data = train_data

alphas = np.linspace(0.45, 0.6, 8)
results = []

for alpha in alphas:

    mse, rmse = compute_mse_rmse(alpha, sample_train_data, cf_predictions
    results.append((alpha, mse, rmse))
    print(f"Alpha: {alpha:.2f}, MSE: {mse:.4f}, RMSE: {rmse:.4f}")

best_alpha, best_mse, best_rmse = min(results, key=lambda x: x[2])
print(f"Best alpha: {best_alpha:.2f}, Best MSE: {best_mse:.4f}, Best RMSE
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[15:44:34] WARNING: /workspace/src/common/error_msg.cc:58: Falling back to
prediction using DMatrix due to mismatched devices. This might lead to hig
her memory usage and slower performance. XGBoost is running on: cuda:0, wh
ile the input data is on: cpu.
Potential solutions:
- Use a data structure that matches the device ordinal in the booster.
- Set the device for booster before call to inplace_predict.

This warning will only be shown once.

  warnings.warn(smsg, UserWarning)
Alpha: 0.45, MSE: 0.9525, RMSE: 0.9759
```

In [ ]:

```
movie_features
```

In [29]:
```
import joblib
import json
import os

# Define paths to save files
output_dir = "/content/drive/MyDrive/movie_dataset_public_final/"
os.makedirs(output_dir, exist_ok=True)  # Create directory if it doesn't

# Save the content-based model
content_model_path = os.path.join(output_dir, "hybridwithalpha_model.pkl"
joblib.dump(content_model, content_model_path)
print(f"Content model saved at: {content_model_path}")

# Save best alpha and evaluation metrics
results_path = os.path.join(output_dir, "results.json")
results_data = {
    "best_alpha": best_alpha,
    "best_mse": best_mse,
    "best_rmse": best_rmse,
    "all_results": results
}
with open(results_path, "w") as f:
    json.dump(results_data, f, indent=4)
print(f"Results saved at: {results_path}")
```

```
Content model saved at: /content/drive/MyDrive/movie_dataset_public_final/
hybridwithalpha_model.pkl
Results saved at: /content/drive/MyDrive/movie_dataset_public_final/result
s.json
```

In [32]:
```
def add_predictions(data, alphas, cf_data, movie_features, content_model)
    data = data.copy()
    for alpha in alphas:
        col_name = f"predicted_score_alpha_{alpha:.2f}"
        data[col_name] = data.apply(
            lambda row: predict_hybrid_score(
                row['user_id'], row['item_id'], cf_data, movie_features,
            ),
```

```
            axis=1
        )
    return data

predicted_data = add_predictions(sample_train_data, [best_alpha], cf_pred
predicted_data_path = os.path.join(output_dir, "predicted_train_data.csv"
predicted_data.to_csv(predicted_data_path, index=False)
print(f"Predicted data saved at: {predicted_data_path}")
```

```
Predicted data saved at: /content/drive/MyDrive/movie_dataset_public_fina
l/predicted_train_data.csv
```

In [36]:

```
def generate_recommendations(user_id, movie_id, cf_data, movie_features,
    """
    Generate top-N movie recommendations for a user, considering a specif
    """
    recommendations = []
    user_rated_items = cf_data[cf_data['user_id'] == user_id]['item_id'].

    for item_id in movie_features['item_id'].unique():
        if item_id not in user_rated_items and item_id != movie_id:  # Ex
            # Predict the hybrid score for this user-item pair
            score = predict_hybrid_score(user_id=user_id, item_id=item_id
                                         cf_data=cf_data,
                                         movie_features=movie_features,
                                         content_model=content_model,
                                         alpha=alpha)
            recommendations.append((item_id, score))

    # Sort recommendations by score in descending order
    recommendations = sorted(recommendations, key=lambda x: x[1], reverse

    return recommendations

# Example usage
user_id = 19  # Replace with a valid user_id from your dataset
movie_id = 1276  # Replace with a valid movie_id the user is interested i
top_recommendations = generate_recommendations(user_id, movie_id, cf_pred
print("Top Recommendations for User, considering movie {}:".format(movie_
```

```
Top Recommendations for User, considering movie 1276: [(26131, 4.132750265
217117)]
```

In [38]:

```
def generate_recommendations3(user_id, movie_id, cf_data, movie_features,
    """
    Generate top-N movie recommendations for a user, considering a specif
    """
    recommendations = []
    user_rated_items = cf_data[cf_data['user_id'] == user_id]['item_id'].

    for item_id in movie_features['item_id'].unique():
        if item_id not in user_rated_items and item_id != movie_id:
            score = predict_hybrid_score(user_id=user_id, item_id=item_id
                                         cf_data=cf_data,
                                         movie_features=movie_features,
                                         content_model=content_model,
                                         alpha=alpha)
            recommendations.append((item_id, score))

    recommendations = sorted(recommendations, key=lambda x: x[1], reverse
```

```
    return recommendations

user_id = 19
movie_id = 1276
top_recommendations = generate_recommendations3(user_id, movie_id, cf_pre
print("Top Recommendations for User, considering movie {}:".format(movie_
```

```
Top Recommendations for User, considering movie 1276: [(26131, 4.132750265
217117), (1148, 4.044837548351577), (2932, 3.931541263675979)]
```

In [ ]: