In [317…
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

collaborative filtering-knn

In [318…
```python
movies_1 = pd.read_csv('movies.csv')
ratings_1 = pd.read_csv('ratings.csv')
```

In [319…
```python
movies = movies_1.copy()
ratings = ratings_1.copy()
```

In [320…
```python
movies.shape
```

Out[320]:
```
(62423, 3)
```

In [321…
```python
ratings.shape
```

Out[321]:
```
(25000095, 4)
```

In [322…
```python
# Step 1: Ensure unique movieid
# If your data has duplicate movie IDs, remove them
movies = movies.drop_duplicates(subset='movieId')

# Step 2: Randomly sample 10000 movies based on unique movieid
movies = movies.sample(n=10000, random_state=42)

# Step 3: Proceed with further processing on the sampled data
print(movies.shape)  # Should print (10000, 5)
print(movies.head())
```

```
(10000, 3)
       movieId                             title  \
4884      4990  Jimmy Neutron: Boy Genius (2001)
22971   116698              Dead Men Tell (1941)
26257   125517                   The D.I. (1957)
57524   196541         Makar - Pathfinder (1984)
39134   156511              Feudin' Fools (1952)

                                 genres
4884   Adventure|Animation|Children|Comedy
22971  Comedy|Crime|Drama|Mystery|Thriller
26257                                Drama
57524                   Adventure|Children
39134                               Comedy
```

In [323…
```python
#merge ratings with movies to select ratings
ratings = ratings.merge(movies).drop(['title', 'genres'],axis=1)
ratings.shape
```

Out[323]:
```
(3623792, 4)
```

In [324…
```python
#selecting only 100,000
ratings = ratings.sample(n=100000, random_state=49)
```

In [325…
```python
movies.head(2)
```

Out[325]:

	movieId	title	genres
4884	4990	Jimmy Neutron: Boy Genius (2001)	Adventure|Animation|Children|Comedy
22971	116698	Dead Men Tell (1941)	Comedy|Crime|Drama|Mystery|Thriller

In [326…
```python
ratings.head(2)
```

Out[326]:

	userId	movieId	rating	timestamp
2689454	23964	7618	4.0	1446980049
3380193	129009	8369	3.5	1480277119

In [327…
```python
#pivot the data so that columns will be userid, index will be movieID and values in the dataframe will be ratin
data = pd.pivot(index = 'movieId',columns = 'userId', data = ratings,values ='rating')
data.head()
```

Out[327]:

userId	3	4	5	8	10	12	14	18	20	21	...	162507	162508	162515	162516	162521	162524	162529	162533	1
movieId																				
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
35	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
36	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 54614 columns

In [328…

```python
#number of users votes for movie
numberOf_user_voted_for_movie = pd.DataFrame(ratings.groupby('movieId')['rating'].agg('count'))
numberOf_user_voted_for_movie.reset_index(level = 0,inplace = True)
numberOf_user_voted_for_movie.head()
```

Out[328]:

	movieId	rating
0	5	302
1	7	346
2	8	41
3	35	44
4	36	546

In [329…

```python
#number of movies voted by user
numberOf_movies_voted_by_user = pd.DataFrame(ratings.groupby('userId')['rating'].agg('count'))
numberOf_movies_voted_by_user.reset_index(level = 0,inplace = True)
numberOf_movies_voted_by_user.head()
```

Out[329]:

	userId	rating
0	3	4
1	4	2
2	5	1
3	8	2
4	10	1

In [330…

```python
#cleaning--> fill na with 0
data.fillna(0, inplace=True)
data.head()
```

Out[330]:

userId	3	4	5	8	10	12	14	18	20	21	...	162507	162508	162515	162516	162521	162524	162529	162533	162534	1625
movieId																					
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
35	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
36	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5 rows × 54614 columns

In [331…

```python
#summary statistics
numberOf_user_voted_for_movie.describe().T
```

Out[331]:

	count	mean	std	min	25%	50%	75%	max
movieId	3209.0	71404.825802	65437.362639	5.0	5824.0	59621.0	128914.0	208603.0
rating	3209.0	31.162356	115.476321	1.0	1.0	3.0	15.0	2308.0

In [332…

```python
#summary statistics
numberOf_movies_voted_by_user.describe().T
```

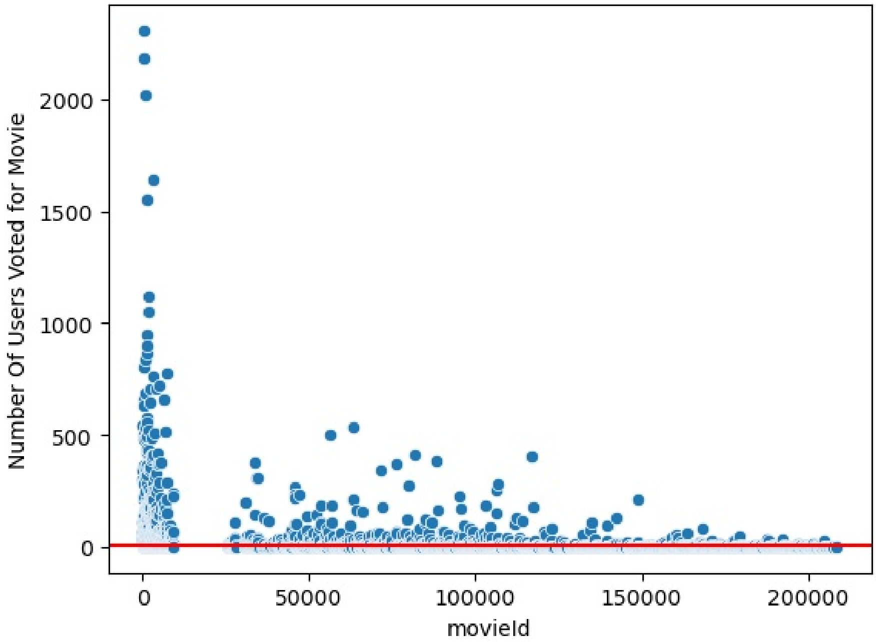
Out[332]:

	count	mean	std	min	25%	50%	75%	max
userId	54614.0	81132.586901	46883.625043	3.0	40575.75	80882.5	121647.25	162536.0
rating	54614.0	1.831032	1.786095	1.0	1.00	1.0	2.00	127.0

In [333…

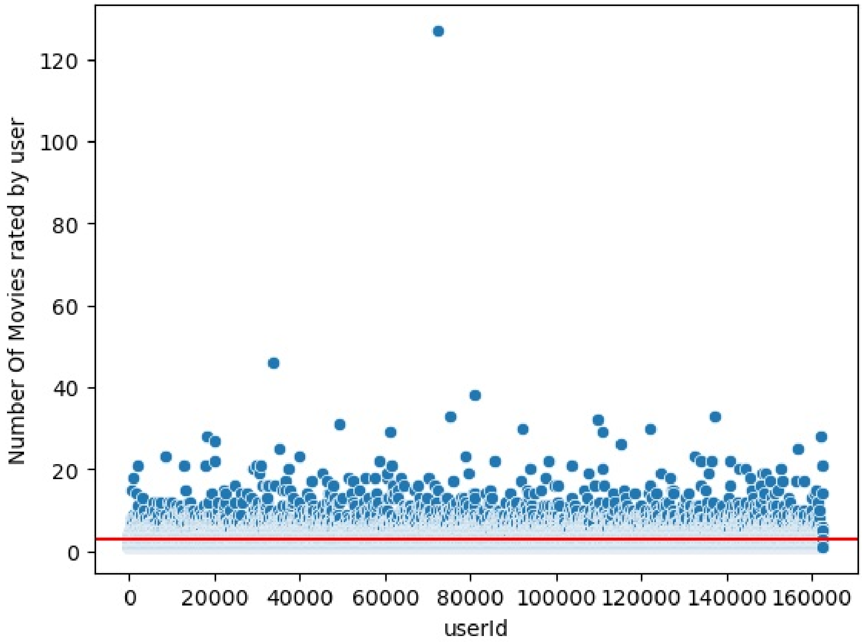
```python
#to select threshold for users voting
```

```
sns.scatterplot(y = 'rating', x = 'movieId', data = numberOf_user_voted_for_movie)
plt.axhline(y=5,color='r')
plt.ylabel('Number Of Users Voted for Movie')
plt.show()
```



In [334…

```
#threshold for movies rated by user
sns.scatterplot(y = 'rating', x = 'userId', data = numberOf_movies_voted_by_user)
plt.axhline(y=3,color='r')
plt.ylabel('Number Of Movies rated by user')
plt.show()
```



In [335…

```
#To qualify a user, a minimum of 5 movies should have voted by the user.
data_final = data.loc[numberOf_user_voted_for_movie[numberOf_user_voted_for_movie['rating'] > 5]['movieId'],:]

#To qualify a movie, a minimum of 5 users should have voted a movie.
data_final = data_final.loc[:,numberOf_movies_voted_by_user[numberOf_movies_voted_by_user['rating'] > 3]['userI
data_final.shape
```

Out[335]:

```
(1252, 5158)
```

In [336…

```
data_final.head()
```

Out[336]:

userId	3	80	84	107	187	256	313	321	406	426	...	162245	162271	162297	162334	162349	162387	162445	162495	162508	1
movieId																					
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
35	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
36	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 5158 columns

In [337…

```
#create matrix
from scipy.sparse import csr_matrix
csr_data = csr_matrix(data_final.values)
data_final.reset_index(inplace=True)
```

In [338…

```
data_final.head()
```

Out[338]:

userId	movieId	3	80	84	107	187	256	313	321	406	...	162245	162271	162297	162334	162349	162387	162445	162495	162508
0	5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	35	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	36	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5159 columns

In [339…

```
#apply NearestNeighbors with algorithm='brute', metric='cosine', n_neighbors=10

from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(csr_data)
```

Out[339]:

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

In [340…

```
def get_movie_recommendation(movie_name):
    n= 30
    movie_list = movies[movies['title'].str.contains(movie_name)]
    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId'] #movieId
        movie_idx = data_final[data_final['movieId'] == movie_idx].index[0] #userId acc to movieId
        distances , indices = knn.kneighbors(csr_data[movie_idx],n_neighbors=n+1)
        rec_movie_indices = sorted(list(zip(indices.squeeze(),distances.squeeze())),key=lambda x: x[1])[1::1]
        recommend = []
        recommend2 = []
        for val in rec_movie_indices:
            movie_idx = data_final.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            recommend.append(movies.iloc[idx]['title'].values[0])
            recommend2.append(val[1])
        df1 = pd.DataFrame(recommend)
        df2 = pd.DataFrame(recommend2)
        df = pd.concat([df1,df2],axis = 'columns')
        df.columns = ['Title','Distance']
        df.set_index('Distance',inplace = True)
        return df
    else:
        return "No movies found. Please check your input"
```

In [341…

```
#using the function to recommend movies
recommended_movies = get_movie_recommendation('Sabrina')
recommended_movies = recommended_movies['Title'].values
```

In [342…

```
recommended_movies
```

```
Out[342]: array(['Burning Rage (1984)', 'Countdown to Zero (2010)',
       'Nappily Ever After (2018)', 'Diana and I (2017)',
       'The Triangle (2016)', 'Terror of Frankenstein (1977)',
       'Fender Bender (2016)', 'Northern Soul (2014)',
       'Pat Garrett and Billy the Kid (1973)', 'God Loves Caviar (2012)',
       'Claire Dolan (1998)', 'Petersburg: Only for Love (2016)',
       'Trench of Hope (2003)', "Bad Man's River (1971)",
       'Polisse (2011)', 'Racket, The (1951)',
       'Angels in the Outfield (1994)', 'Christmas Belle (2013)',
       'Sex and Lucia (Lucía y el sexo) (2001)',
       'Daddy and the Muscle Academy (1991)', 'Fog Over Frisco (1934)',
       'Death Proof (2007)', 'Little Hiawatha (1937)', 'Ping Pong (2002)',
       'Underground (1976)', 'The Ever After (2015)',
       'Wyatt Cenac: Brooklyn (2014)', 'Salome Where She Danced (1945)',
       'Far from Home (1989)', 'Chicks (2010)'], dtype=object)
```

other

```
In [343…: data_final = ratings.merge(movies)
```

```
In [344…: # Example usage: #128027,137293
# user_id = 94937 (example from the dataset in the image)
#print(get_user_recommendation(user_id=122011, n=10))
```

final for knn

```
In [345…: import pandas as pd
from sklearn.neighbors import NearestNeighbors
from sklearn.model_selection import train_test_split
from scipy.sparse import csr_matrix
```

```
In [346…: # Load your data
# data_final = ... (your dataset)
# movies = ... (your movie dataset)



# Step 1: Split the dataset
train_data = data_final.sample(frac=0.8, random_state=42)  # 80% for training
test_data = data_final.drop(train_data.index)  # Remaining 20% for testing

# Step 2: Create the user-item matrix for training data
user_movie_matrix_train = train_data.pivot(index='userId', columns='movieId', values='rating').fillna(0)

# Convert to a sparse matrix format for training
csr_data_train = csr_matrix(user_movie_matrix_train.values)

# Step 3: Fit the Nearest Neighbors model on the training data
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(csr_data_train)

# Step 4: Create the user-item matrix for testing data (keep the same structure)
user_movie_matrix_test = test_data.pivot(index='userId', columns='movieId', values='rating').fillna(0)

# Convert to a sparse matrix format for testing
csr_data_test = csr_matrix(user_movie_matrix_test.values)

# Ensure the model is using the correct input shape
def get_user_recommendation(user_id, n=10):
    if user_id not in user_movie_matrix_train.index:
        return "User not found. Please check the user ID."

    user_idx = user_movie_matrix_train.index.get_loc(user_id)

    # Find the nearest neighbors (similar users)
    distances, indices = knn.kneighbors(csr_data_train[user_idx], n_neighbors=n+1)

    # Get the user ids of similar users
    similar_users = user_movie_matrix_train.index[indices.squeeze()].tolist()

    # Remove the first user (which is the user itself)
    similar_users = similar_users[1:]

    # Collect movies from all similar users
    recommend_movies = {}

    for similar_user in similar_users:
        similar_user_ratings = test_data[test_data['userId'] == similar_user]
        target_user_ratings = test_data[test_data['userId'] == user_id]
        unrated_movies = similar_user_ratings[~similar_user_ratings['movieId'].isin(target_user_ratings['movieI

        for _, row in unrated_movies.iterrows():
            movie_id = row['movieId']
```

```python
            rating = row['rating']
            if movie_id not in recommend_movies:
                recommend_movies[movie_id] = rating
            else:
                recommend_movies[movie_id] += rating

    sorted_recommendations = sorted(recommend_movies.items(), key=lambda x: x[1], reverse=True)
    top_movie_ids = [movie_id for movie_id, _ in sorted_recommendations][:n]
    recommend_titles = movies[movies['movieId'].isin(top_movie_ids)]['title'].tolist()

    return recommend_titles if recommend_titles else "No new recommendations found."
```

In [347…

```python
#80974, 49403,92046, 122011, 85757, 136310, 132651, 39896
get_user_recommendation(user_id=74794, n=10)
```

Out[347]:

```
['Day the Earth Stood Still, The (1951)',
 'Identity (2003)',
 'National Treasure (2004)',
 'Spy Kids 2: The Island of Lost Dreams (2002)',
 'Star Wars: Episode V - The Empire Strikes Back (1980)',
 'City Lights (1931)',
 'Cliffhanger (1993)',
 'Spy (2015)']
```

In [348…

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

# Step 1: Filter users who have at least 2 ratings
user_ratings_count = data_final['userId'].value_counts()
filtered_users = user_ratings_count[user_ratings_count >= 20].index

# Keep only the filtered users in your dataset
filtered_data = data_final[data_final['userId'].isin(filtered_users)]

# Step 2: Perform the train/test split on the filtered data
train_data, test_data = train_test_split(filtered_data, test_size=0.2, random_state=4)
```

In [349…

```python
# Step 2: Create user-item matrix from the train set (rows = userId, columns = movieId)
user_movie_matrix_train = train_data.pivot(index='userId', columns='movieId', values='rating').fillna(0)

# Convert to a sparse matrix
csr_data_train = csr_matrix(user_movie_matrix_train.values)

# Train Nearest Neighbors model on the train set
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(csr_data_train)
```

Out[349]:

```
NearestNeighbors

NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

In [350…

```python
def get_user_recommendation(user_id, n=20):
    if user_id not in user_movie_matrix_train.index:
        return []

    # Get the index of the user in the matrix
    user_idx = user_movie_matrix_train.index.get_loc(user_id)

    # Find the nearest neighbors (similar users)
    distances, indices = knn.kneighbors(csr_data_train[user_idx], n_neighbors=n+1)

    # Get the user ids of similar users
    similar_users = user_movie_matrix_train.index[indices.squeeze()].tolist()
    similar_users = similar_users[1:]  # Remove the user itself from the list

    # Collect movies from all similar users
    recommend_movies = {}
    target_user_ratings = train_data[train_data['userId'] == user_id]

    for similar_user in similar_users:
        similar_user_ratings = train_data[train_data['userId'] == similar_user]
        unrated_movies = similar_user_ratings[~similar_user_ratings['movieId'].isin(target_user_ratings['movieI

        for _, row in unrated_movies.iterrows():
            movie_id = row['movieId']
            rating = row['rating']
            if movie_id not in recommend_movies:
                recommend_movies[movie_id] = rating
            else:
                recommend_movies[movie_id] += rating

    # Sort recommendations by rating
    sorted_recommendations = sorted(recommend_movies.items(), key=lambda x: x[1], reverse=True)
    top_movie_ids = [movie_id for movie_id, _ in sorted_recommendations][:n]

    # Get movie titles
    recommend_titles = movies[movies['movieId'].isin(top_movie_ids)]['title'].tolist()
```

```python
    return recommend_titles
```

In [351…:
```python
get_user_recommendation(user_id=72315, n=20)
```

Out[351]:
```
['Keeping the Faith (2000)',
 'Abominable Dr. Phibes, The (1971)',
 'Superman (1978)',
 'Killer Joe (2011)',
 'Mindhunters (2004)',
 'Sliding Doors (1998)',
 'Chaplin (1992)',
 'Drums Along the Mohawk (1939)',
 'Mad Max (1979)',
 'Star Wars: Episode V - The Empire Strikes Back (1980)',
 'Thirty-Two Short Films About Glenn Gould (1993)',
 'What Lies Beneath (2000)',
 'Titanic (1997)',
 "Let's Go to Prison (2006)",
 'Another Thin Man (1939)',
 'Shawshank Redemption, The (1994)',
 'Annie Get Your Gun (1950)',
 'Painted Veil, The (2006)',
 'Licence to Kill (1989)',
 'Guns of Navarone, The (1961)']
```

In [352…:
```python
from sklearn.metrics import precision_score, recall_score

# Step 4: Evaluate the recommender system
def evaluate_recommender(test_data, n_recommendations=20):
    all_precisions = []
    all_recalls = []

    for user_id in test_data['userId'].unique():
        # Get the list of relevant items (movies the user rated in the test set)
        relevant_items = test_data[test_data['userId'] == user_id]['movieId'].tolist()

        # Get the recommended items from the model (based on train data)
        recommended_items = get_user_recommendation(user_id, n=n_recommendations)

        if not relevant_items or not recommended_items:
            continue

        # Convert movie titles to movie IDs
        recommended_movie_ids = movies[movies['title'].isin(recommended_items)]['movieId'].tolist()

        # Compute precision and recall
        true_positive = len(set(recommended_movie_ids) & set(relevant_items))
        precision = true_positive / len(recommended_movie_ids) if recommended_movie_ids else 0
        recall = true_positive / len(relevant_items) if relevant_items else 0

        all_precisions.append(precision)
        all_recalls.append(recall)

    # Calculate average precision and recall
    avg_precision = sum(all_precisions) / len(all_precisions) if all_precisions else 0
    avg_recall = sum(all_recalls) / len(all_recalls) if all_recalls else 0
    f1_score = 2 * (avg_precision * avg_recall) / (avg_precision + avg_recall) if (avg_precision + avg_recall) 

    return avg_precision, avg_recall, f1_score

# Run the evaluation
precision, recall, f1 = evaluate_recommender(test_data)
print(f"Precision: {precision}, Recall: {recall}, F1 Score: {f1}")
```

```
Precision: 0.002325581395348837, Recall: 0.0079734219269103, F1 Score: 0.0036009002250562638
```

SURPRISE

In [37]:
```python
ratings.shape
```

Out[37]:
```
(100000, 4)
```

In [356…:
```python
df_ratings = ratings.copy()
```

In [39]:
```python
#algorithms
from surprise import SVD

from surprise.prediction_algorithms.knns import KNNBasic
from surprise import Dataset, Reader
from surprise.model_selection import cross_validate
```

In [40]:
```python
#to read
reader = Reader(line_format = 'user item rating', rating_scale=(0.5,5))
```

In [41]:
```python
#creating data suitable for algorithm
```

```
data_1 = Dataset.load_from_df(df_ratings[['userId', 'movieId', 'rating']], reader)
```

In [42]:
```
#training with cross validation of 5 for RMSE, MAE
knn = KNNBasic()

# Run 5-fold cross-validation and then print results
cross_validate(knn, data_1, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.1159  1.1201  1.1138  1.1078  1.1063  1.1128  0.0051
MAE (testset)     0.8786  0.8796  0.8761  0.8703  0.8686  0.8746  0.0044
Fit time          70.94   89.03   83.94   84.91   89.54   83.67   6.73
Test time         2.97    3.36    2.62    2.59    2.82    2.87    0.28
```

Out[42]:
```
{'test_rmse': array([1.11591602, 1.12008455, 1.11380001, 1.10777701, 1.1063239 ]),
 'test_mae': array([0.8786377 , 0.8795898 , 0.87608819, 0.87027357, 0.86855494]),
 'fit_time': (70.94488406181335,
  89.02788233757019,
  83.93716931343079,
  84.90560221672058,
  89.53563380241394),
 'test_time': (2.9672834873199463,
  3.357412815093994,
  2.6189143657684326,
  2.589510917663574,
  2.817070245742798)}
```

In [43]:
```
# Train on the entire dataset
trainset = data_1.build_full_trainset()
knn.fit(trainset)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
```

Out[43]:
```
<surprise.prediction_algorithms.knns.KNNBasic at 0x1e16d48ac50>
```

In [44]:
```
# Predict the rating for a specific user and movie
user_id = 3
movie_id = 1252
predicted_rating = knn.predict(user_id, movie_id)

print(f"Predicted rating for user {user_id} on movie {movie_id} is: {predicted_rating.est}")
```

```
Predicted rating for user 3 on movie 1252 is: 5
```

In [45]:
```
#actual rating for the above prediction
df_ratings[df_ratings['userId']==3]
```

Out[45]:

	userId	movieId	rating	timestamp
886466	3	5959	4.0	1439474161
707841	3	1252	5.0	1484753938
1015669	3	27728	5.0	1484754362
801376	3	3798	3.0	1439473012

In [358…
```
#just dropping timestamp because it not needed
df_ratings_1 = df_ratings.merge(movies).drop('timestamp',axis=1).copy()
```

In [47]:
```
# Function to get top N movie recommendations
def top_n_movies(userid, model, n=5):
    predict_ratings = {}

    # Predict ratings for all movies for a specific user
    for movieid in df_ratings_1['movieId']:
        if not trainset.knows_item(movieid):  # If the movie wasn't rated by this user
            pred = model.predict(userid, movieid)
            predict_ratings[movieid] = pred.est

    # Sort movies based on predicted ratings (descending order)
    top_movies = sorted(predict_ratings.items(), key=lambda x: x[1], reverse=True)[:n]

    # Print the top N movie titles
    for movieid, rating in top_movies:
        movie_title = df_ratings_1.loc[df_ratings_1['movieId'] == movieid, 'title'].values[0]
        print(f"Movie: {movie_title}, Predicted rating: {rating}")
```

In [48]:
```python
# Example usage
top_n_movies(3, knn, n=5)
```

```
Movie: Paperman (2012), Predicted rating: 5
Movie: Triplets of Belleville, The (Les triplettes de Belleville) (2003), Predicted rating: 5
Movie: Maltese Falcon, The (a.k.a. Dangerous Female) (1931), Predicted rating: 5
Movie: Her (2013), Predicted rating: 5
Movie: Louis C.K.: Chewed Up (2008), Predicted rating: 5
```

cosine_similarity

In [353…
```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix
```

In [359…
```python
# Step 1: the dataset
data = df_ratings_1.copy()
data.head()
```

Out[359]:

	userId	movieId	rating	title	genres
0	23964	7618	4.0	Chaplin (1992)	Drama
1	107521	7618	3.0	Chaplin (1992)	Drama
2	142844	7618	0.5	Chaplin (1992)	Drama
3	88597	7618	5.0	Chaplin (1992)	Drama
4	129375	7618	4.0	Chaplin (1992)	Drama

In [360…
```python
# Step 1: Remove duplicates based on movieid or title
# Keep only one instance of each unique movie (either by movieid or title)
data = data.drop_duplicates(subset='movieId')  # You can also use 'movieid' if that's more appropriate
data.head()
```

Out[360]:

	userId	movieId	rating	title	genres
0	23964	7618	4.0	Chaplin (1992)	Drama
30	129009	8369	3.5	Mindhunters (2004)	Action|Crime|Horror|Mystery|Thriller
47	161242	45732	2.5	My Super Ex-Girlfriend (2006)	Comedy|Fantasy|Romance
93	128027	7438	4.5	Kill Bill: Vol. 2 (2004)	Action|Drama|Thriller
866	77206	474	3.0	In the Line of Fire (1993)	Action|Thriller

In [361…
```python
# Step 2: Combine titles and genres into a single text feature with space between genres
data['combined'] = data.apply(lambda row: f"{row['title']} {' '.join(row['genres'].split('|'))}", axis=1)
data.head(3)
```

Out[361]:

	userId	movieId	rating	title	genres	combined
0	23964	7618	4.0	Chaplin (1992)	Drama	Chaplin (1992) Drama
30	129009	8369	3.5	Mindhunters (2004)	Action|Crime|Horror|Mystery|Thriller	Mindhunters (2004) Action Crime Horror Mystery...
47	161242	45732	2.5	My Super Ex-Girlfriend (2006)	Comedy|Fantasy|Romance	My Super Ex-Girlfriend (2006) Comedy Fantasy R...

In [362…
```python
# Step 3: Preprocess the combined text (lowercase)
data['combined'] = data['combined'].str.lower()
data.head(3)
```

Out[362]:

	userId	movieId	rating	title	genres	combined
0	23964	7618	4.0	Chaplin (1992)	Drama	chaplin (1992) drama
30	129009	8369	3.5	Mindhunters (2004)	Action|Crime|Horror|Mystery|Thriller	mindhunters (2004) action crime horror mystery...
47	161242	45732	2.5	My Super Ex-Girlfriend (2006)	Comedy|Fantasy|Romance	my super ex-girlfriend (2006) comedy fantasy r...

In [363…
```python
# Step 4: Apply TfidfVectorizer to the combined text
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(data['combined'])
```

In [364…
```python
# Convert the TF-IDF matrix to a sparse matrix format
tfidf_matrix_sparse = csr_matrix(tfidf_matrix)
```

In [365…
```python
# Compute cosine similarity using the sparse matrix
cosine_sim_sparse = cosine_similarity(tfidf_matrix_sparse, dense_output=False) # Keep the output sparse
```

In [366…
```python
# Step 5: Compute cosine similarity
#cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
cosine_sim_sparse
```

Out[366]:
```
<3209x3209 sparse matrix of type '<class 'numpy.float64'>'
	with 4820139 stored elements in Compressed Sparse Row format>
```

In [513…
```
# Step 7: Function to recommend movies
def recommend_movies_sparse(movie_title, cosine_sim=cosine_sim_sparse):
    # Get the index of the movie that matches the title
    idx = data[data['title'] == movie_title].index[0]

    # Get the pairwise similarity scores for that movie with all others (row in the sparse matrix)
    sim_scores = list(enumerate(cosine_sim[idx].toarray().flatten()))  # Convert to array to handle indexing

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

    # Exclude the input movie itself
    sim_scores = sim_scores[1:]

    # Get the indices of the top 5 most similar movies
    movie_indices = [i[0] for i in sim_scores[:1000]]

    # Return the top 5 most similar movies
    return data['title'].iloc[movie_indices]
```

In [528…
```
# Example usage
recommended = recommend_movies_sparse('Kill Bill: Vol. 2 (2004)')
print("Recommended Movies:")
print(recommended)
```

```
Recommended Movies:
99136                   Welcome (2007)
98578                  Arranged (2007)
99890               Frenchmen 2 (2007)
95754                      Guru (2007)
98753                 After Sex (2007)
                       ...
98298    Anna and the Apocalypse (2017)
71206       Hudsucker Proxy, The (1994)
98168                      Kids (2008)
98902            Scare Campaign (2016)
99805          In Family I Trust (2019)
Name: title, Length: 1000, dtype: object
```

In [515…
```
#exp_relevant = data[(data['genres'].str.contains('Th')) | (data['genres'].str.contains('Ac'))]['title'].values
#exp_relevant = pd.DataFrame(exp_relevant).sample(800).values
```

In [516…
```
exp_relevant = data[(data['genres'].str.contains('Dra'))]['title'].values
exp_relevant = pd.DataFrame(exp_relevant).sample(1000).values
```

In [517…
```
exp_relevant_2 = data[(data['genres'].str.contains('Come'))]['title'].values
exp_relevant_2 = pd.DataFrame(exp_relevant_2).sample(1000).values
```

In [518…
```
exp_relevant_3 = data[(data['genres'].str.contains('Adv')) | (data['genres'].str.contains('Act'))|
                    (data['genres'].str.contains('Dra'))]['title'].values
exp_relevant_3 = pd.DataFrame(exp_relevant_3).sample(1000).values
```

In [519…
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Single test case (title, expected relevant movies)
test_case = {
    'movie_title': 'In the Line of Fire (1993)',
    'expected_relevant': exp_relevant,
}
```

best - one use case

In [520…
```
# Initialize lists to store actual results and predicted results
y_true = []  # Actual relevant movies
y_pred = []  # Recommended movies

# Evaluate the model for the single test case
movie_title = test_case['movie_title']
expected_relevant = test_case['expected_relevant']

# Get recommended movies
recommended_movies = recommend_movies_sparse(movie_title)

# Convert recommendations to a list
recommended_list = recommended_movies.tolist()

# Add the expected relevant movies to y_true
y_true += [1] * len(expected_relevant)  # Mark all expected relevant movies as 1
y_true += [0] * (len(recommended_list) - len(expected_relevant))  # Add 0s for non-relevant

# Mark the predicted relevant movies
for movie in recommended_list:
    if movie in expected_relevant:
```

```python
        y_pred.append(1)  # Relevant movie predicted
    else:
        y_pred.append(0)  # Non-relevant movie predicted

# Calculate precision, recall, and F1 score
precision = precision_score(y_true, y_pred, average='binary', zero_division=0)
recall = recall_score(y_true, y_pred, average='binary', zero_division=0)
f1 = f1_score(y_true, y_pred, average='binary', zero_division=0)

# Print the evaluation metrics
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
```

```
Precision: 1.0
Recall: 0.652
F1 Score: 0.7893462469733656
```

In [ ]: