joseph

February 12, 2025

Final Phase 4 Project

Student name: Mulwa Joseph
Class: DSPT-08
date: 12/2/2025
Instructor name: Daniel Ekale

0.0.1 Project Overview

We will use MovieLens dataset (see https://grouplens.org/datasets/movielens/) for finding similar users based on common movies the users have watched and how they have rated those movies. We will be using collaborative filterin to find similarity between users

0.0.2 Objectives

1. Retrieve the top 5 movie title recommended to user based on their ratings
2. Enable users explore a wide range of films similar to their preferences
3. Drive users subscriptions by suggesting appealing content

0.0.3 Libraries

[154]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from surprise.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.metrics.pairwise import cosine_similarity
from surprise import SVD, Dataset, Reader
import re
from fuzzywuzzy import process
from surprise.model_selection import GridSearchCV
from surprise import accuracy
from sklearn.preprocessing import MinMaxScaler
from collections import Counter
from surprise.model_selection import cross_validate
```

```
from surprise.prediction_algorithms import SVD, KNNWithMeans, KNNBasic,␣
 ↪KNNBaseline
```

```
c:\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\fuzzywuzzy\fuzz.py:11:
UserWarning: Using slow pure-python SequenceMatcher. Install python-Levenshtein
to remove this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-
Levenshtein to remove this warning')
```

0.0.4 Loading Dataset

[155]:
```
#Loading dataset
movies = pd.read_csv("ml-latest-small/ml-latest-small/movies.csv")
ratings =  pd.read_csv("ml-latest-small/ml-latest-small/ratings.csv")
```

[156]:
```
ratings.head()
```

[156]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

[157]:
```
#Dropping timestamp from our dataframe
ratings.drop('timestamp', axis = 1, inplace =True)
```

[158]:
```
#number of unique mivies in the dataset
len(ratings.movieId.unique())
```

[158]: 9724

[159]:
```
#Merge the datasets together
movie_ratings = ratings.merge(movies, on='movieId')
movie_ratings.head()
```

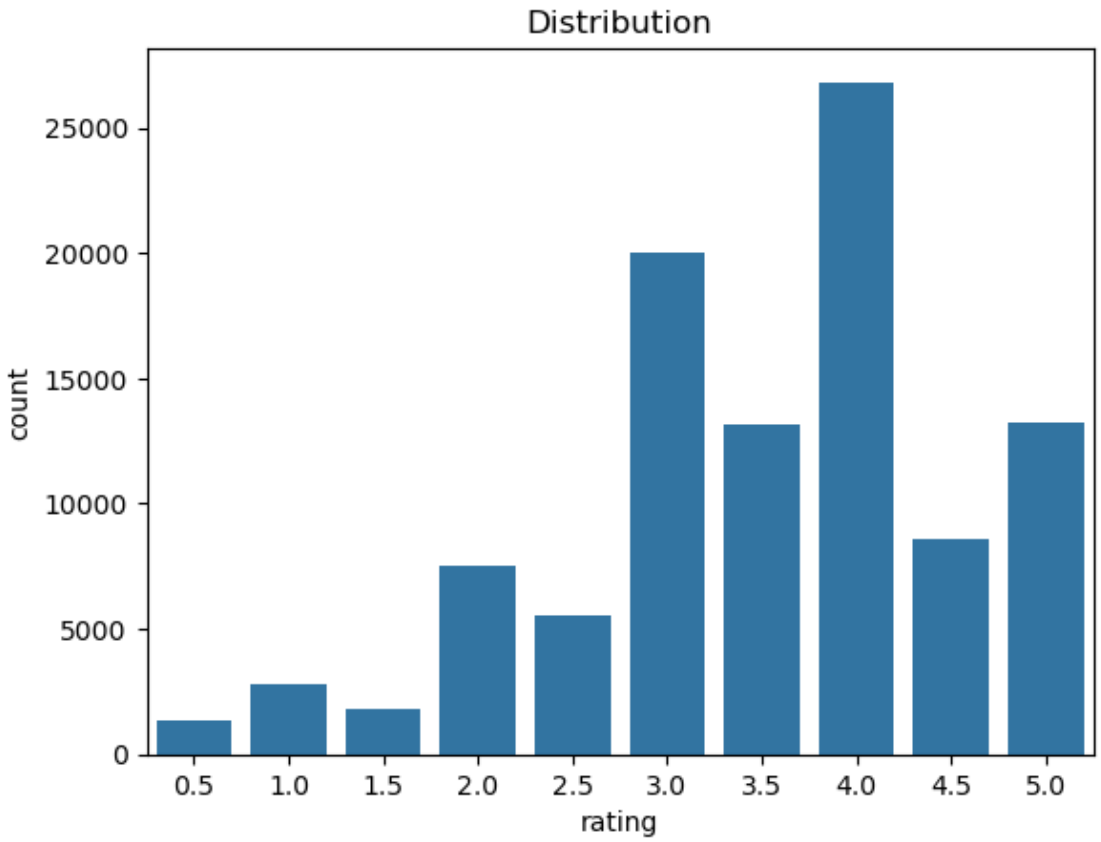
[159]:

	userId	movieId	rating	title	genres
0	1	1	4.0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	1	3	4.0	Grumpier Old Men (1995)	Comedy|Romance
2	1	6	4.0	Heat (1995)	Action|Crime|Thriller
3	1	47	5.0	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller
4	1	50	5.0	Usual Suspects, The (1995)	

4 Crime|Mystery|Thriller

0.0.5 EDA

[160]:
```
sns.countplot(x='rating',data=ratings)
plt.title('Distribution')
plt.show()
```



[161]:
```
#The top 5 rated movies
movie_ratings = ratings.merge(movies, on='movieId')
top_movies = movie_ratings['title'].value_counts().nlargest(5)
print(top_movies)
```

```
title
Forrest Gump (1994)                 329
Shawshank Redemption, The (1994)    317
Pulp Fiction (1994)                 307
Silence of the Lambs, The (1991)    279
Matrix, The (1999)                  278
Name: count, dtype: int64
```

[162]:
```
#Split the genres to a list
movies['genres'] =  movies['genres'].apply(lambda x: x.split ('|'))

total_genre = Counter(g for genres in movies ['genres'] for g in genres)

print(f'No of genres{len(total_genre)}')
```

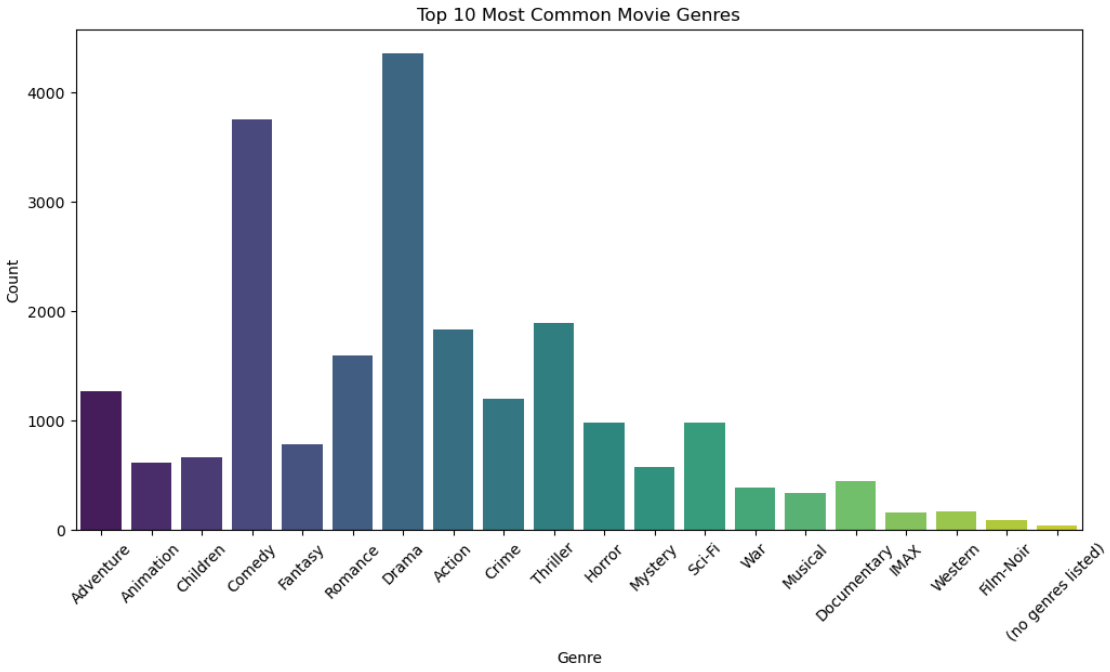
```
No of genres20
```

[ ]:
```
#Top 10 genre visualization
#Change  to dataframe
top_genre_df = pd.DataFrame([total_genre]).T.reset_index()
top_genre_df.columns = ['genre','count']
plt.figure(figsize=(12, 6))
sns.barplot(x='genre', y='count', data=top_genre_df, palette="viridis")
plt.xlabel("Genre")
plt.ylabel("Count")
plt.title("Top 10 Most Common Movie Genres")
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\Joseph\AppData\Local\Temp\ipykernel_7608\2386159201.py:6:
FutureWarning:

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

  sns.barplot(x='genre', y='count', data=top_genre_df, palette="viridis")
```



0.0.6 Data Processing

[ ]:
```
#Replacing the NAN values in our dataframe
movies_df.fillna(0, inplace = True)
movies_df.iloc[0:5, 0:10]
```

[ ]:

movieId	1	2	3	4	5	6	7	8	9	10
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[ ]:
```
# #Creating a pivot table to represent users as rows and movies as columns
movies_df = movie_ratings.pivot(index ='userId',columns='movieId',values =␣
 ↪'rating')
movies_df.index = movie_ratings.userId.unique()
```

[ ]:
```
#Representing the similarity between rows and converting them in a dataframe
#similarity closer to 1 means users are very similar and closer to 0 means␣
 ↪users are very dissimilar.
row_similarities = 1 - pairwise_distances(movies_df.values,metric ="cosine")
row_similarities_df =pd.DataFrame(row_similarities)
row_similarities_df.index = movie_ratings.userId.unique()
row_similarities_df.columns = movie_ratings.userId.unique()

row_similarities_df.iloc[0:5, 0:5]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[153], line 3
      1 #Representing the similarity between rows and converting them in a␣
 ↪dataframe
      2 #similarity closer to 1 means users are very similar and closer to 0␣
 ↪means users are very dissimilar.
----> 3 row_similarities = 1 -␣
 ↪pairwise_distances(movies_df.values,metric ="cosine")
      4 row_similarities_df =pd.DataFrame(row_similarities)
      5 row_similarities_df.index = movie_ratings.userId.unique()

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\utils\_param_val dation.
 ↪py:216, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwarg s)
    210 try:
    211     with config_context(
```

```
    212         skip_parameter_validation=(
    213             prefer_skip_nested_validation or global_skip_validation
    214         )
    215     ):
--> 216         return func(*args, **kwargs)
    217 except InvalidParameterError as e:
    218     # When the function is just a wrapper around an estimator, we allow
    219     # the function to delegate validation to the estimator, but we␣
 ↪replace
    220     # the name of the estimator by the name of the function in the erro
    221     # message to avoid confusion.
    222     msg = re.sub(
    223         r"parameter of \w+ must be",
    224         f"parameter of {func.__qualname__} must be",
    225         str(e),
    226     )

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\metrics\pairwise
 ↪py:2480, in pairwise_distances(X, Y, metric, n_jobs, force_all_finite,␣
 ↪ensure_all_finite, **kwds)
  2477         return distance.squareform(distance.pdist(X, metric=metric,␣
 ↪**kwds))
  2478     func = partial(distance.cdist, metric=metric, **kwds)
-> 2480 return _parallel_pairwise(X, Y, func, n_jobs, **kwds)

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\metrics\pairwise
 ↪py:1973, in _parallel_pairwise(X, Y, func, n_jobs, **kwds)
  1970 X, Y, dtype = _return_float_dtype(X, Y)
  1972 if effective_n_jobs(n_jobs) == 1:
-> 1973     return func(X, Y, **kwds)
  1975 # enforce a threading backend to prevent data communication overhead
  1976 fd = delayed(_dist_wrapper)

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\utils\_param_val
 ↪py:189, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwarg
   187 global_skip_validation = get_config()["skip_parameter_validation"]
   188 if global_skip_validation:
--> 189     return func(*args, **kwargs)
   191 func_sig = signature(func)
   193 # Map *args/**kwargs to the function signature

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\metrics\pairwise
 ↪py:1171, in cosine_distances(X, Y)
  1168 xp, _ = get_namespace(X, Y)
  1170 # 1.0 - cosine_similarity(X, Y) without copy
```

```
-> 1171 S = cosine_similarity(X, Y)
   1172 S *= -1
   1173 S += 1

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\utils\_param_val dation.
 ↪py:189, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwarg s)
    187 global_skip_validation = get_config()["skip_parameter_validation"]
    188 if global_skip_validation:
--> 189     return func(*args, **kwargs)
    191 func_sig = signature(func)
    193 # Map *args/**kwargs to the function signature

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\metrics\pairwise
 ↪py:1741, in cosine_similarity(X, Y, dense_output)
   1695 """Compute cosine similarity between samples in X and Y.
   1696
   1697 Cosine similarity, or the cosine kernel, computes similarity as the
   (…)
   1737         [0.57…, 0.81…]])
   1738 """
   1739 # to avoid recursive import
-> 1741 X, Y = check_pairwise_arrays(X, Y)
   1743 X_normalized = normalize(X, copy=True)
   1744 if X is Y:

File c:
 ↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\metrics\pairwise
 ↪py:190, in check_pairwise_arrays(X, Y, precomputed, dtype, accept_sparse,␣
 ↪force_all_finite, ensure_all_finite, ensure_2d, copy)
    187     dtype = dtype_float
    189 if Y is X or Y is None:
--> 190     X = Y = check_array(
    191         X,
    192         accept_sparse=accept_sparse,
    193         dtype=dtype,
    194         copy=copy,
    195         ensure_all_finite=ensure_all_finite,
    196         estimator=estimator,
    197         ensure_2d=ensure_2d,
    198     )
    199 else:
    200     X = check_array(
    201         X,
    202         accept_sparse=accept_sparse,
   (…)
    207         ensure_2d=ensure_2d,
    208     )
```

```
File c:
↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\utils\validation
↪py:1107, in check_array(array, accept_sparse, accept_large_sparse, dtype,␣
↪order, copy, force_writeable, force_all_finite, ensure_all_finite,␣
↪ensure_non_negative, ensure_2d, allow_nd, ensure_min_samples,␣
↪ensure_min_features, estimator, input_name)
   1101     raise ValueError(
   1102         "Found array with dim %d. %s expected <= 2."
   1103         % (array.ndim, estimator_name)
   1104     )
   1106 if ensure_all_finite:
-> 1107     _assert_all_finite(
   1108         array,
   1109         input_name=input_name,
   1110         estimator_name=estimator_name,
   1111         allow_nan=ensure_all_finite == "allow-nan",
   1112     )
   1114 if copy:
   1115     if _is_numpy_namespace(xp):
   1116         # only make a copy if `array` and `array_orig` may share memory

File c:
↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\utils\validation
↪py:120, in _assert_all_finite(X, allow_nan, msg_dtype, estimator_name,␣
↪input_name)
    117 if first_pass_isfinite:
    118     return
--> 120 _assert_all_finite_element_wise(
    121     X,
    122     xp=xp,
    123     allow_nan=allow_nan,
    124     msg_dtype=msg_dtype,
    125     estimator_name=estimator_name,
    126     input_name=input_name,
    127 )

File c:
↪\Users\Joseph\anaconda3\envs\myenv\lib\site-packages\sklearn\utils\validation
↪py:169, in _assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype,␣
↪estimator_name, input_name)
    152 if estimator_name and input_name == "X" and has_nan_error:
    153     # Improve the error message on how to handle missing values in
    154     # scikit-learn.
    155     msg_err += (
    156         f"\n{estimator_name} does not accept missing values"
    157         " encoded as NaN natively. For supervised learning, you might␣
↪want"
   (…)
    167         "#estimators-that-handle-nan-values"
```

```
    168     )
--> 169 raise ValueError(msg_err)

ValueError: Input contains NaN.
```

[15]:
```
#To find silmilar users
np.fill_diagonal(row_similarities, -1)
similar_users = row_similarities_df.idxmax(axis=1)
print(similar_users[0:5])
```

```
1    266
2    366
3    313
4    391
5    470
dtype: int64
```

[16]:
```
#To get movies rated greater than 4
high_rated_movies = movie_ratings[movie_ratings['rating'] >4]
high_rated_movies
```

[16]:

	userId	movieId	rating	title	genres
3	1	47	5.0	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller
4	1	50	5.0	Usual Suspects, The (1995)	Crime|Mystery|Thriller
6	1	101	5.0	Bottle Rocket (1996)	Adventure|Comedy|Crime|Romance
8	1	151	5.0	Rob Roy (1995)	Action|Drama|Romance|War
9	1	157	5.0	Canadian Bacon (1995)	Comedy|War
…	…	…	…	…	…
100821	610	160527	4.5	Sympathy for the Underdog (1971)	Action|Crime|Drama
100829	610	164179	5.0	Arrival (2016)	Sci-Fi
100832	610	168248	5.0	John Wick: Chapter Two (2017)	Action|Crime|Thriller
100833	610	168250	5.0	Get Out (2017)	Horror
100834	610	168252	5.0	Logan (2017)	Action|Sci-Fi

```
[21762 rows x 5 columns]
```

#Creating the Cosine similarity between movies

```
[ ]: user_item_matrix = movie_ratings.pivot(index='userId', columns='movieId',␣
      ↪values='rating')
     user_item_matrix = user_item_matrix.fillna(0)
     item_item_matrix = user_item_matrix.T

     movies_sim = cosine_similarity(item_item_matrix)
     movies_sim_df = pd.DataFrame(movies_sim, index=item_item_matrix.index,␣
      ↪columns=item_item_matrix.index)
     def get_similar_movies(movie_id, top_n=5):
         if movie_id not in movies_sim_df.index:
             return f"Movie ID {movie_id} not found in the dataset."
         sim_score = movies_sim_df.loc[movie_id]
         sim_movies = sim_score.sort_values(ascending=False)
         similar_movie_ids = sim_movies.drop(movie_id).head(top_n).index

         similar_movies_details = movie_ratings[movie_ratings['movieId'].
      ↪isin(similar_movie_ids)][['movieId', 'title', 'genres']].drop_duplicates()

         return similar_movies_details

     movie_id = 344
     similar_movies = get_similar_movies(movie_id, top_n=3)


     print(f"Top 3 similar movies to movieId {movie_id}:")
     print(similar_movies)
```

Using Suprise

```
[ ]: reader = Reader(rating_scale=(1,5))
     data = Dataset.
      ↪load_from_df(movie_ratings[['userId','movieId','rating']],reader=reader)
```

```
[ ]: #KNN Model
     item_based_sim = {'name':'pearson',
                       'user_based':True}
     knn = KNNBasic(k=20,
                 min_k = 5,
                 sim_options = item_based_sim)
```

```
[ ]: #5-fold Validation to measure RMSE
     cv = cross_validate(knn,
                         data,
                         measures = ['RMSE'],
```

```
                    cv =5,
                    verbose =False)
```

[ ]:
```
knn_rmse = cv['test_rmse'].mean()
knn_rmse
```

[ ]:
```
#  the SVD model,with cross validation
svd = SVD()
svd_cv = cross_validate(svd, data, measures=['RMSE'], cv=5, verbose=False)
svd_rmse = svd_cv['test_rmse'].mean()
print(f"SVD Model RMSE: {svd_rmse:.4f}")
```

[ ]:
```
#comparing KNN and SVD to determine whch has better performance
if knn_rmse < svd_rmse:
    print("KNN performs better than SVD.")
else:
    print("SVD performs better than KNN.")
```

0.1 Recommender System

[ ]:
```
#splitting dataset
train_set,test_set = train_test_split(data, test_size =0.2, random_state=42)
```

[ ]:
```
#Grid search on SVD for better performance
#Hyperparameter Tuning for SVD
param_grid = {
    'n_factors': [50, 100, 200],
    'lr_all': [0.002, 0.005, 0.01],
    'reg_all': [0.02, 0.04, 0.06]
}

grid_search = GridSearchCV(SVD, param_grid, measures=['rmse', 'MAE'], cv=5)
grid_search.fit(data)
print(grid_search.best_score['rmse'])
print(grid_search.best_params['rmse'])
```

[ ]:
```
#Using SVD for collaborative filtering for model trainin
best_svd = grid_search.best_estimator['rmse']
trainset = data.build_full_trainset()
svd_model = SVD(n_factors=50, random_state=42)
svd_model.fit(trainset)
```

[ ]:
```
predictions = svd_model.test(test_set)
svd_rmse = accuracy.rmse(predictions)
print(f"SVD RMSE: {svd_rmse}")
```

[ ]:
```
# #Allowing partial matches
# def get_similar_movies_fuzzy(movie_title, top_n=5):
#     movie_title = movie_title.strip().lower()
#     best_match, score = process.extractOne(movie_title, movies_sim_df.index,␣
 ↪score_cutoff=80)
#     if best_match:
#         print(f"Best match: '{best_match}' (confidence: {score})")
#         sim_scores = movies_sim_df.loc[best_match]
#         sorted_scores = sim_scores.sort_values(ascending=False)[1:top_n+1]
#         return sorted_scores
#     else:
#         print(f"No close match found for '{movie_title}' in the dataset.")
#         return None
```

[ ]:
```
# Create a mapping between movie titles and movie IDs
movie_title_to_id = dict(zip(movie_ratings['title'], movie_ratings['movieId']))
movie_id_to_title = dict(zip(movie_ratings['movieId'], movie_ratings['title']))

# Function to handle user input and get similar movies
def recommend_movies(user_input, top_n=5):
    if user_input.isdigit():
        movie_id = int(user_input)
        if movie_id not in movie_id_to_title:
            return f"Movie ID {movie_id} not found in the dataset."
    else:
        if user_input not in movie_title_to_id:
            return f"Movie title '{user_input}' not found in the dataset."
        movie_id = movie_title_to_id[user_input]

    similar_movies = get_similar_movies(movie_id, top_n)
    return similar_movies

# Example usage
user_input = input("Enter a movie title or movie ID: ")
similar_movies = recommend_movies(user_input, top_n=5)

print(f"Top 5 similar movies to '{movie_id_to_title[int(user_input)] if␣
 ↪user_input.isdigit() else user_input}':")
print(similar_movies)
```

[ ]:
```
print(movies_sim_df.index)
```

[ ]:
```
movies.head()
```

[ ]:
```

```