## Source Code

# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,

# THEN FEEL FREE TO DELETE THIS CELL.

# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON

# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR

# NOTEBOOK.

import kagglehub

parasharmanas\_movie\_recommendation\_system\_path = kagglehub.dataset\_download('parasharmanas/movie-recommendation-system')

print('Data source import complete.')

import pandas as pd

import numpy as np

df1 = pd.read\_csv("/kaggle/input/movie-recommendation-system/movies.csv")

print('Jumlah Data :', len(df1.iloc[:,1]))

print('Jumlah Fitur :', len(df1.iloc[1,:]))

print(f'Terdapat {len(df1.iloc[1,:])} Kolom Fitur pada Dataset yaitu:')

print('Fitur Data :', df1.columns.tolist()[:])

pd.options.display.max\_columns = None

df1.head()

df1.info()

import re

def clean\_title(title):

    return re.sub("[^a-zA-Z0-9 ]", "", title)

# Pisahkan genre menggunakan pemisah '|'

df1['genres'] = df1['genres'].str.split('|')

# Bersihkan judul film

df1['title'] = df1['title'].apply(clean\_title)

# Perbarui movies\_data

movies\_data = df1[['movieId', 'title', 'genres']]

# Mendapatkan genre unik dari semua film

unique\_genres = pd.Series([genre for genres\_list in movies\_data['genres'] for genre in genres\_list]).unique()

# Output hasil

print(movies\_data.head())

print(f"Terdapat {movies\_data['title'].nunique()} Judul Film")

print(f"Terdapat {len(unique\_genres)} Genre Film.")

print("Genre Film:", unique\_genres)

# Periksa jumlah baris dengan '(no genres listed)'

no\_genres\_count = movies\_data[movies\_data['genres'].apply(lambda x: '(no genres listed)' in x)].shape[0]

print(f"Terdapat {no\_genres\_count} film tanpa genre.")

# Hapus baris dengan '(no genres listed)'

movies\_data = movies\_data[~movies\_data['genres'].apply(lambda x: '(no genres listed)' in x)]

# Perbarui daftar genre unik

unique\_genres = pd.Series([genre for genres\_list in movies\_data['genres'] for genre in genres\_list]).unique()

# Tampilkan hasil setelah penghapusan

print(f"Setelah penghapusan, terdapat {movies\_data['title'].nunique()} Judul Film.")

print(f"Terdapat {len(unique\_genres)} Genre Film setelah pembaruan.")

print("Genre Film:", unique\_genres)

import pandas as pd

import matplotlib.pyplot as plt

# Menghitung jumlah film per genre

genre\_counts = pd.Series([genre for genres\_list in movies\_data['genres'] for genre in genres\_list]).value\_counts()

plt.figure(figsize=(12, 6))

genre\_counts.plot(kind='bar', color='skyblue')

plt.title('Jumlah Film per Genre', fontsize=16)

plt.xlabel('Genre', fontsize=14)

plt.ylabel('Jumlah Film', fontsize=14)

plt.xticks(rotation=45, ha='right')

plt.show()

df2 = pd.read\_csv("/kaggle/input/movie-recommendation-system/ratings.csv")

print('Jumlah Data :', len(df2.iloc[:,1]))

print('Jumlah Fitur :', len(df2.iloc[1,:]))

print(f'Terdapat {len(df2.iloc[1,:])} Kolom Fitur pada Dataset yaitu:')

print('Fitur Data :', df2.columns.tolist()[:])

pd.options.display.max\_columns = None

df2.head()

# Drop timestamp column

ratings\_data = df2.drop(['timestamp'], axis=1)

print(ratings\_data.head())

# Melihat Missing Values

print("Jumlah Missing Values per Kolom:")

print(df2.isnull().sum())

print("\nJumlah Data Duplicates:")

print(df2.duplicated().sum())

ratings\_data.info()

print("Distribusi Rating:")

print(df2['rating'].value\_counts())

print("\nRating Rata-Rata per Film:")

print(df2.groupby('movieId')['rating'].mean().head())

print("\nRating Rata-Rata per Pengguna:")

print(df2.groupby('userId')['rating'].mean().head())

plt.show()import seaborn as sns

plt.figure(figsize=(8, 6))

sns.histplot(df2['rating'], bins=5, kde=False, color='skyblue')

plt.title('Distribusi Rating', fontsize=16)

plt.xlabel('Rating', fontsize=14)

plt.ylabel('Frekuensi', fontsize=14)

combined\_data = ratings\_data.merge(movies\_data, on='movieId')

print(combined\_data.head())

# Rating Rata-Rata per Film

avg\_ratings\_per\_movie = combined\_data.groupby('title')['rating'].mean().sort\_values(ascending=False)

print("Top 10 Film dengan Rating Rata-Rata Tertinggi:")

print(avg\_ratings\_per\_movie.head(10))

movie\_rating\_counts = combined\_data.groupby('title')['rating'].count().sort\_values(ascending=False)

print("Top 10 Film dengan Jumlah Rating Terbanyak:")

print(movie\_rating\_counts.head(10))

# Memisahkan Genre

exploded\_data = combined\_data.explode('genres')

# Menghitung Jumlah Pengguna yang Memberi Rating pada Tiap Genre

users\_per\_genre = exploded\_data.groupby('genres')['userId'].nunique()

# Rata Rata Rating di Tiap Genre

avg\_rating\_per\_genre = exploded\_data.groupby('genres')['rating'].mean()

# Cari 3 film terbaik berdasarkan rating rata-rata di tiap genre

top\_movies\_per\_genre = (

    exploded\_data.groupby(['genres', 'title'])['rating']

    .mean()

    .reset\_index()

    .sort\_values(['genres', 'rating'], ascending=[True, False])

    .groupby('genres')

    .head(3)

)

print("Jumlah Pengguna yang Memberi Rating pada Tiap Genre:")

print(users\_per\_genre)

print("\nRata-Rata Rating per Genre:")

print(avg\_rating\_per\_genre)

print("\n3 Film Terbaik di Tiap Genre:")

print(top\_movies\_per\_genre)

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

vectorizer\_title = TfidfVectorizer(ngram\_range=(1,2))

tfidf\_title = vectorizer\_title.fit\_transform(movies\_data['title'])

def search\_by\_title(title):

    title = clean\_title(title)

    query\_vec = vectorizer\_title.transform([title])

    similarity = cosine\_similarity(query\_vec, tfidf\_title).flatten()

    indices = np.argpartition(similarity, -5)[-5:]

    results = movies\_data.iloc[indices][::-1]

    return results

movie\_results = search\_by\_title("Interstellar")

print(movie\_results)

def search\_by\_title(title):

    title = clean\_title(title)

    query\_vec = vectorizer\_title.transform([title])

    similarity = cosine\_similarity(query\_vec, tfidf\_title).flatten()

    indices = np.argpartition(similarity, -5)[-5:]

    results = movies\_data.iloc[indices][::-1]

    return results

movie\_results = search\_by\_title("Fast and Furious")

print(movie\_results)

vectorizer\_genres = TfidfVectorizer(ngram\_range=(1,2))

# Gabungkan genre list menjadi string

movies\_data['genres\_text'] = movies\_data['genres'].apply(lambda x: ' '.join(x))

tfidf\_genres = vectorizer\_genres.fit\_transform(movies\_data['genres\_text'])

def search\_similar\_genres(genres):

    query\_vec = vectorizer\_genres.transform([genres])

    similarity = cosine\_similarity(query\_vec, tfidf\_genres).flatten()

    indices = np.argpartition(similarity, -10)[-10:]

    results = movies\_data.iloc[indices][::-1]

    return results

gen = 'Adventure Action'

print(search\_similar\_genres(gen))

def scores\_calculator(movie\_id):

    # Filter data untuk pengguna serupa

    similar\_users = combined\_data.loc[

        (combined\_data['movieId'] == movie\_id) & (combined\_data['rating'] >= 4), 'userId'

    ].unique()

    # Dapatkan rekomendasi berdasarkan pengguna serupa

    similar\_user\_recs = combined\_data.loc[

        (combined\_data['userId'].isin(similar\_users)) & (combined\_data['rating'] >= 4), 'movieId'

    ].value\_counts(normalize=True)

    # Dapatkan rekomendasi berdasarkan semua pengguna

    all\_user\_recs = combined\_data.loc[

        combined\_data['movieId'].isin(similar\_user\_recs.index) & (combined\_data['rating'] >= 4)

    ]

    all\_user\_recs = all\_user\_recs['movieId'].value\_counts(normalize=True)

    # Filter genre dari film yang dipilih

    selected\_genres = combined\_data.loc[combined\_data['movieId'] == movie\_id, 'genres'].iloc[0]

    if isinstance(selected\_genres, list):

        selected\_genres = " ".join(selected\_genres)

    # Cari film dengan genre serupa

    movies\_with\_similar\_genres = search\_similar\_genres(selected\_genres)

    similar\_genre\_ids = movies\_with\_similar\_genres['movieId']

    # Kalikan skor berdasarkan genre serupa

    similar\_user\_recs.loc[similar\_user\_recs.index.isin(similar\_genre\_ids)] \*= 1.5

    all\_user\_recs.loc[all\_user\_recs.index.isin(similar\_genre\_ids)] \*= 0.9

    # Gabungkan skor dan hitung peringkat

    scores = pd.DataFrame({

        'similar': similar\_user\_recs,

        'all': all\_user\_recs

    }).fillna(0)

    # Hindari pembagian nol

    scores['score'] = np.where(scores['all'] > 0, scores['similar'] / scores['all'], 0)

    # Urutkan berdasarkan skor tertinggi

    return scores.sort\_values('score', ascending=False)

scores\_calculator(3114)

def recommendation\_results(user\_input, title=0):

    # user\_input = clean\_title(user\_input)

    title\_candidates = search\_by\_title(user\_input)

    movie\_id = title\_candidates.iloc[title]['movieId']

    scores = scores\_calculator(movie\_id)

    results = scores.head(10).merge(movies\_data, left\_index=True, right\_on='movieId')[['title', 'score', 'genres']]

    resutls = results.rename(columns={'title': 'title', 'genres': 'genres'}, inplace=True)

    return results

user\_input = "Interstellar"

print("Here a similar movies: ")

for i in range(5):

    print(i, ": ", search\_by\_title(user\_input)['title'].iloc[i])

title = 0

print("Recommendation\_results: ")

print(recommendation\_results(user\_input))