```
# 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 LIBRARY

```
import pandas as pd
import numpy as np
```

EXPLORATORY DATASET

Movies Dataset

```
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()
```

```
Jumlah Data : 62423
Jumlah Fitur : 3
Terdapat 3 Kolom Fitur pada Dataset yaitu:
Fitur Data : ['movieId', 'title', 'genres']
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62423 entries, 0 to 62422
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  62423 non-null  int64 
 1   title    62423 non-null  object
 2   genres   62423 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.4+ MB
```

```
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())
```

```
   movieId                             title  \
0        1                   Toy Story 1995
1        2                     Jumanji 1995
2        3            Grumpier Old Men 1995
3        4           Waiting to Exhale 1995
4        5  Father of the Bride Part II 1995

                                        genres
0  [Adventure, Animation, Children, Comedy, Fantasy]
1                    [Adventure, Children, Fantasy]
2                                [Comedy, Romance]
3                         [Comedy, Drama, Romance]
4                                         [Comedy]
```

```
print(f"Terdapat {movies_data['title'].nunique()} Judul Film")
print(f"Terdapat {len(unique_genres)} Genre Film.")
print("Genre Film:", unique_genres)
```

```
Terdapat 62298 Judul Film
Terdapat 20 Genre Film.
Genre Film: ['Adventure' 'Animation' 'Children' 'Comedy' 'Fantasy' 'Romance' 'Drama'
 'Action' 'Crime' 'Thriller' 'Horror' 'Mystery' 'Sci-Fi' 'IMAX'
 'Documentary' 'War' 'Musical' 'Western' 'Film-Noir' '(no genres listed)']
```

```
# 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)
```

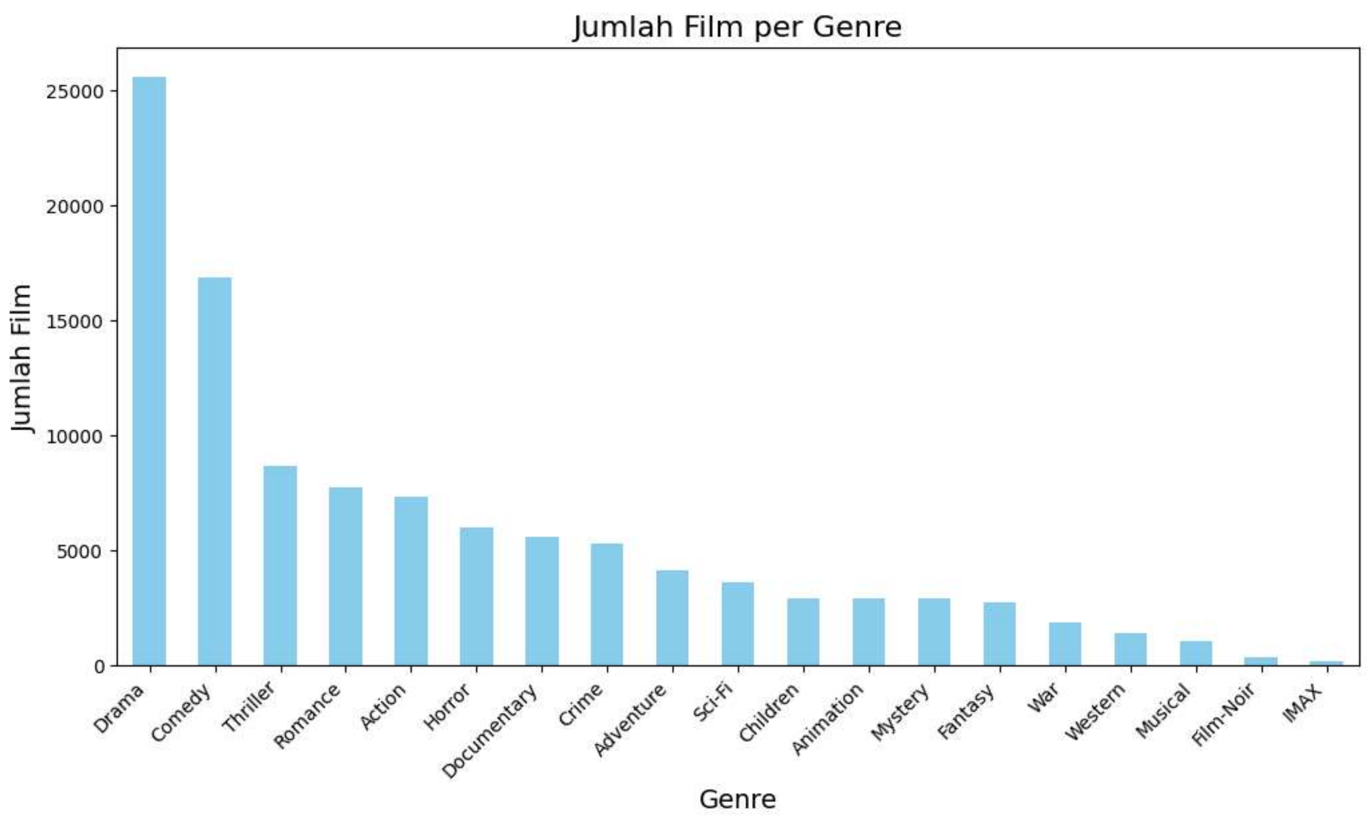
```
Terdapat 5062 film tanpa genre.
Setelah penghapusan, terdapat 57264 Judul Film.
Terdapat 19 Genre Film setelah pembaruan.
Genre Film: ['Adventure' 'Animation' 'Children' 'Comedy' 'Fantasy' 'Romance' 'Drama'
 'Action' 'Crime' 'Thriller' 'Horror' 'Mystery' 'Sci-Fi' 'IMAX'
 'Documentary' 'War' 'Musical' 'Western' 'Film-Noir']
```

Visualisasi Dataset

```
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()
```



Ratings Dataset

```
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())
```

```
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)
plt.show()
```

MERGE 2 DATAFRAMES

```
combined_data = ratings_data.merge(movies_data, on='movieId')
print(combined_data.head())
```

Eksplorasi Dataset

```
# 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)
```

PEMBUATAN MODEL TF-IDF

```
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)
```

Menggunakan TF-IDF untuk mencari 5 movies dengan judul yang mirip

```
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)
```

Mencari movies dengan genre yang mirip

```
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))
```

CALCULATE SCORE

```
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)
```

RECOMMENDATION RESULT

```
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))
```