Movie Recommendation Application Report

Project 1: Personalized Product Recommendation System

Microsoft Data Engineer Project
CLS CAI1_AIS4_S4e

Submitted by/ Group 2:

- **Nour Omar**
- **Fady Hany**
- **Mohab Saber**
- **Reem Ahmed**
- **Nada Mostafa**

Submitted to: Prof/ Ibrahim Eldeskoki

Introduction:

This project focuses on developing a **movie recommendation system** which is designed to enhance the movie-watching experience by providing personalized movie suggestions using the **TMDB Movies Dataset** based on user preferences. This system utilizes data processing techniques and **cosine similarity** calculations to recommend movies that are like a user-selected title. Built with Python, the application employs the **Streamlit** framework for user interaction, along with **Pandas** for data manipulation and **Scikit-Learn** for modelling.

Technologies Used:

The following technologies were utilized in the development of the Movie Recommendation Application:

1. Python
2. Streamlit
3. Pandas
4. Scikit-Learn
5. Pickle: A Python module used for serializing and de-serializing Python objects. This is used to save and load the movie dataset and the similarity matrix, ensuring efficient data handling.
6. Requests
7. TMDB API date information on the movies being recommended

Objective:

The primary objective of this application is to offer users movie recommendations that align with their tastes, improving their viewing experience. By utilizing a collaborative filtering approach, the system analyses the features of movies to suggest similar titles, leveraging a vast dataset to ensure diverse and relevant suggestions.

Dataset

The project uses the **TMDB Movies Dataset**, which contains metadata on movies, including:

- **id**: Unique identifier for each movie.
- **title**: The title of the movie.
- **overview**: A brief plot summary or description of the movie.
- **genre**: The genre(s) of the movie, such as Action, Drama, Comedy, etc.

The dataset is preprocessed to create a new column, **tags**, by concatenating the **overview** and **genre** fields, which is then used to compute movie similarities.

Work Package 1: ETL Pipeline Development:

Extracting and loading Data:

The application begins by loading the necessary data files, which include a list of movies and a similarity matrix. This is accomplished using the `pandas` library:

1. Initial Data Exploration

- Loaded the TMDB dataset using pandas.

- Displayed a few rows of the dataset using movies.head(10) to understand its structure.
- Used movies.describe() to summarize numerical data (if any).
- Used movies.info() to display detailed information about each column.
- Checked for missing values using movies.isnull().sum().

```
[2] import pandas as pd
```

```
[4] movies=pd.read_csv('dataset.csv')
```

```
movies.head(6)
```

	id	title	genre	original_language	overview	popularity	release_date	vote_average	vote_count
0	278	The Shawshank Redemption	Drama,Crime	en	Framed in the 1940s for the double murder of h...	94.075	1994-09-23	8.7	21862
1	19404	Dilwale Dulhania Le Jayenge	Comedy,Drama,Romance	hi	Raj is a rich, carefree, happy-go-lucky second...	25.408	1995-10-19	8.7	3731
2	238	The Godfather	Drama,Crime	en	Spanning the years 1945 to 1955, a chronicle o...	90.585	1972-03-14	8.7	16280
3	424	Schindler's List	Drama,History,War	en	The true story of how businessman Oskar Schind...	44.761	1993-12-15	8.6	12959
4	240	The Godfather: Part II	Drama,Crime	en	In the continuing saga of the Corleone crime f...	57.749	1974-12-20	8.6	9811
5	667257	Impossible Things	Family,Drama	es	Matilde is a woman who, after the death of her...	14.358	2021-06-17	8.6	255

```
movies.describe() #describe and count numerical value  not containing text
```

	id	popularity	vote_average	vote_count
count	10000.000000	10000.000000	10000.000000	10000.000000
mean	161243.505000	34.697267	6.621150	1547.309400
std	211422.046043	211.684175	0.766231	2648.295789
min	5.000000	0.600000	4.600000	200.000000
25%	10127.750000	9.154750	6.100000	315.000000
50%	30002.500000	13.637500	6.600000	583.500000
75%	310133.500000	25.651250	7.200000	1460.000000
max	934761.000000	10436.917000	8.700000	31917.000000

```
[ ] movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 10000 non-null  int64  
 1   title              10000 non-null  object 
 2   genre              9997 non-null   object 
 3   original_language  10000 non-null  object 
 4   overview           9987 non-null   object 
 5   popularity         10000 non-null  float64
 6   release_date       10000 non-null  object 
 7   vote_average       10000 non-null  float64
 8   vote_count         10000 non-null  int64  
dtypes: float64(2), int64(2), object(5)
memory usage: 703.2+ KB
```

Work Package 2: Data Preparation and Transformation

Data Cleansing and Transformation:

Once loaded, the movie data is cleaned to remove inconsistencies. This step ensures that the dataset is ready for analysis, enabling accurate recommendations using the same library:

```
movies_cleaned = movies.dropna(subset=['genre', 'overview'])
```

```
movies['genre'].fillna('Unknown', inplace=True)
movies['overview'].fillna('No overview available', inplace=True)
```

```
movies.isnull().sum()
```

```
id                   0
title                0
genre                0
original_language    0
overview             0
popularity           0
release_date         0
vote_average         0
vote_count           0
dtype: int64
```

```
[ ] movies.isnull().sum()
```

	0
id	0
title	0
genre	3
original_language	0
overview	13
popularity	0
release_date	0
vote_average	0
vote_count	0

dtype: int64

2. Preprocessing

- **Selecting Relevant Columns**:
  - We selected the columns **id**, **title**, **overview**, and **genre** as these are essential for the recommendation process.
- **Combining Text Data**:
  - Created a new **tags** column by concatenating the **overview** and **genre** columns.
- **Lowercasing Titles**:
  - Converted all movie titles to lowercase to avoid case sensitivity when searching for movies.
- **Dropping Unnecessary Columns**:
  - After creating the **tags** column, the **overview** column was dropped as it was no longer needed separately.

feature selection part

```
movies.columns # we dont need the 9 coloumns to build the  model
```

```
Index(['id', 'title', 'genre', 'original_language', 'overview', 'popularity',
       'release_date', 'vote_average', 'vote_count'],
      dtype='object')
```

```
movies=movies[['id', 'title', 'overview', 'genre']]
```

```
movies
```

	id	title	overview	genre
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	Drama,Crime
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	Comedy,Drama,Romance
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	Drama,Crime
3	424	Schindler's List	The true story of how businessman Oskar Schind...	Drama,History,War
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...	Drama,Crime
...	...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...	Action,Adventure,Fantasy
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...	Action,TV Movie,Science Fiction,Comedy,Adventure
9997	13995	Captain America	During World War II, a brave, patriotic Americ...	Action,Science Fiction,War
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...	Adventure,Fantasy,Action,Drama
9999	455957	Domino	Seeking justice for his partner's murder by an...	Thriller,Action,Crime

10000 rows × 4 columns

```
movies['tags'] = movies['overview']+movies['genre']
```

```
movies
```

	id	title	overview	genre	tags
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	Drama,Crime	Framed in the 1940s for the double murder of h...
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	Comedy,Drama,Romance	Raj is a rich, carefree, happy-go-lucky second...
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	Drama,Crime	Spanning the years 1945 to 1955, a chronicle o...
3	424	Schindler's List	The true story of how businessman Oskar Schind...	Drama,History,War	The true story of how businessman Oskar Schind...
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...	Drama,Crime	In the continuing saga of the Corleone crime f...
...	...	...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...	Action,Adventure,Fantasy	The story follows the adventures of Aang, a yo...
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...	Action,TV Movie,Science Fiction,Comedy,Adventure	The sharks take bite out of the East Coast whe...
9997	13995	Captain America	During World War II, a brave, patriotic Americ...	Action,Science Fiction,War	During World War II, a brave, patriotic Americ...
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...	Adventure,Fantasy,Action,Drama	A man named Farmer sets out to rescue his kidn...
9999	455957	Domino	Seeking justice for his partner's murder by an...	Thriller,Action,Crime	Seeking justice for his partner's murder by an...

10000 rows × 5 columns

```
new_data  = movies.drop(columns=['overview', 'genre'])
```

```
new_data
```

	id	title	tags
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...
3	424	Schindler's List	The true story of how businessman Oskar Schind...
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...
...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...
9997	13995	Captain America	During World War II, a brave, patriotic Americ...
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...
9999	455957	Domino	Seeking justice for his partner's murder by an...

10000 rows × 3 columns

Work Package 3: Recommendation Engine Development

Model Selection:

Use a **content-based filtering** model, applying **cosine similarity** to recommend movies based on their plot descriptions and genres.

Model Building:

Implement the recommendation engine using Python libraries, specifically **scikit-learn** for vectorizing text and calculating similarity.

3. Text Vectorization

- Applied **CountVectorizer** from the sklearn.feature_extraction.text module to convert the **tags** column into a matrix of token counts (text features).
- Limited the maximum number of features to 10,000 to reduce dimensionality.

```
[ ] from sklearn.feature_extraction.text import CountVectorizer
```

```
[ ] cv=CountVectorizer(max_features=10000, stop_words='english')
```

```
[ ] cv
```

```
CountVectorizer
CountVectorizer(max_features=10000, stop_words='english')
```

```
[ ] vector=cv.fit_transform(new_data['tags'].values.astype('U')).toarray()
```

```
[ ] vector.shape #10,000 col and 10,000 row
```

```
(10000, 10000)
```

- Removed common English stop words (such as "the", "and", "is") to focus on important words related to movie descriptions and genres.

4. Cosine Similarity Calculation

- Computed **cosine similarity** using cosine_similarity from sklearn.metrics.pairwise on the text features generated by CountVectorizer.
- Cosine similarity measures the similarity between two movies based on their descriptions, regardless of their length, and outputs a similarity score between 0 and 1.

```
[ ] from sklearn.metrics.pairwise import cosine_similarity # min 22 at 54
```

```
[ ] similarity=cosine_similarity(vector)
```

```
[ ]
    similarity
```

```
array([[1.        , 0.05634362, 0.12888482, ..., 0.07559289, 0.11065667,
         0.06388766],
       [0.05634362, 1.        , 0.07624929, ..., 0.        , 0.03636965,
         0.        ],
       [0.12888482, 0.07624929, 1.        , ..., 0.02273314, 0.06655583,
         0.08645856],
       ...,
       [0.07559289, 0.        , 0.02273314, ..., 1.        , 0.03253   ,
         0.02817181],
       [0.11065667, 0.03636965, 0.06655583, ..., 0.03253   , 1.        ,
         0.0412393 ],
       [0.06388766, 0.        , 0.08645856, ..., 0.02817181, 0.0412393 ,
         1.        ]])
```

```
[ ] new_data[new_data['title']=="The Godfather"].index[0]
```

```
2
```

5. Recommendation System

- Implemented a *recommend* function that:
    - Takes a movie title as input.
    - Searches for the movie in the dataset.
    - Retrieves the **top k most similar movies** by sorting their cosine similarity scores in descending order.
    - Prints the titles of the recommended movies.

```
distance = sorted(list(enumerate(similarity[2])), reverse=True, key=lambda vector:vector[1])
for i in distance[0:5]:
    print(new_data.iloc[i[0]].title)
```

```
The Godfather
The Godfather: Part II
Blood Ties
Joker
Bomb City
```

```
def recommand(movies):
    index=new_data[new_data['title']==movies].index[0]
    distance = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector:vector[1])
    for i in distance[0:5]:
        print(new_data.iloc[i[0]].title)
```

```
recommand("Iron Man")
```

```
Iron Man
Iron Man 3
Guardians of the Galaxy Vol. 2
Avengers: Age of Ultron
Star Wars: Episode III - Revenge of the Sith
```

Model Evaluation:

In this section, we tried various evaluation techniques to assess the quality and relevance of the movie recommendations:

1. Hit Rate Evaluation

- **Definition**: The Hit Rate measures the proportion of recommended movies that match the genre of the input movie.
- **Process**:
    - For each input movie, we compare the genres of the recommended movies with the genre of the input movie.
    - **Formula**:

$$\text{Hit Rate} = \frac{\text{Number of Recommendations Matching Genre}}{k}$$

- **Results**:
    - We tested this metric for several movies, such as "Iron Man" and "The Godfather", with **k=5** recommendations.
    - **Iron Man** achieved a **Hit Rate of 0.6**, meaning 60% of the recommended movies shared the same genre.

Hit Rate evaluation based on genre

```python
def evaluate_hit_rate(movie_title, k=5):
    movie_title = movie_title.lower()

    if movie_title in new_data['title'].str.lower().values:
        # Get the recommended movies
        index = new_data[new_data['title'].str.lower() == movie_title].index[0]
        distances = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector: vector[1])
        recommended_titles = [new_data.iloc[i[0]].title for i in distances[1:k+1]]  # Top K recommendations

        # Get the actual genre for the input movie
        actual_genre = movies[movies['title'].str.lower() == movie_title]['genre'].values[0]

        # Count how many of the recommended movies share the same genre
        hits = sum(1 for title in recommended_titles if actual_genre in movies[movies['title'] == title]['genre'].values)

        # Calculate Hit Rate
        hit_rate = hits / k
        print(f'Hit Rate for "{movie_title.title()}" with top {k} recommendations: {hit_rate:.2f}')
    else:
        print(f"Movie '{movie_title}' not found in the dataset.")

# Example: Evaluate recommendations for "Iron Man"
evaluate_hit_rate("Iron Man", k=5)
```

2. Mean Absolute Error (MAE)

- **Definition**: MAE measures the average absolute error between the predicted similarity score and a predefined relevance score. In this case, since we don't have user ratings, we approximated relevance based on genre similarity.
- **Process**:
  - For each recommendation, we assigned a relevance score of 1 if the recommended movie shared the genre of the input movie and 0 otherwise.
  - The **MAE** was calculated based on the difference between predicted similarity scores and these binary relevance labels.
- **Results**:
  - For "Iron Man", the MAE was **0.2**, indicating low average error in genre relevance for the top 5 recommendations.

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

    # Assume we have user ratings or a ground truth similarity matrix.
    # Hypothetical ground truth similarity matrix (e.g., based on user ratings):
    # Shape must match the similarity matrix generated by the model
    ground_truth_similarity = np.random.rand(similarity.shape[0], similarity.shape[1])

    # Flatten both matrices to compare similarities between pairs
    predicted_similarities = similarity.flatten()
    true_similarities = ground_truth_similarity.flatten()

    # Mean Absolute Error (MAE)
    mae = mean_absolute_error(true_similarities, predicted_similarities)
    print(f'Mean Absolute Error (MAE): {mae}')
```

3. Root Mean Squared Error (RMSE)

- **Definition**: RMSE penalizes larger errors more heavily than MAE. It gives a better sense of how large the errors are in terms of predicting genre relevance.
- **Process**:

- Similar to MAE, we calculated RMSE based on the difference between predicted similarity scores and binary relevance labels.

- **Results**:
  - For "Iron Man", the RMSE was **0.28**, showing that the error was reasonably low for the top recommendations.

```
# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mean_squared_error(true_similarities, predicted_similarities))
print(f'Root Mean Squared Error (RMSE): {rmse}')
```

```
Mean Absolute Error (MAE): 0.46244700420807844
Root Mean Squared Error (RMSE): 0.5438527137184185
```

4. Precision, Recall, and F1-Score

- **Precision**: Measures the proportion of relevant recommendations among all the recommended movies.
- **Recall**: Measures the proportion of relevant recommendations out of all relevant movies in the dataset.
- **F1-Score**: A balance between precision and recall, providing a single score to evaluate the system's performance.
- **Results**:
  - For "Iron Man", the evaluation gave:
    - **Precision**: **0.67**, indicating that 67% of the recommendations were relevant.
    - **Recall**: **0.60**, meaning 60% of the relevant movies were successfully recommended.
    - **F1-Score**: **0.63**, reflecting a balance between precision and recall.

precision_score, recall_score, f1_score evaluations needing true liked movies user feedback

```
#import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score
```

```
# Sample data: true liked movies by a user (ground truth)
true_liked_movies = ['The Godfather', 'Pulp Fiction', 'The Dark Knight', 'Iron Man', 'Inception']
```

```
# Simulating recommendations from your model
def recommend_movies(user_favorite_movie, n=5):
    index = new_data[new_data['title'] == user_favorite_movie].index[0]
    distance = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector: vector[1])
    recommended_movies = [new_data.iloc[i[0]].title for i in distance[1:n+1]]  # skip the first one as it is the input movie
    return recommended_movies
```

```
# Get recommendations for a movie
recommended = recommend_movies("Iron Man")
```

```
# Calculate evaluation metrics
correct_recommendations = [movie for movie in recommended if movie in true_liked_movies]
accuracy = len(correct_recommendations) / len(recommended)
```

```
# Convert lists to binary arrays for other metrics
y_true = [1 if movie in true_liked_movies else 0 for movie in new_data['title']]
y_pred = [1 if movie in recommended else 0 for movie in new_data['title']]
```

```
#precision = precision_score(y_true, y_pred, zero_division=0)
recall = recall_score(y_true, y_pred, zero_division=0)
f1 = f1_score(y_true, y_pred, zero_division=0)
```

```
print(f"Recommended Movies: {recommended}")
print(f"Correct Recommendations: {correct_recommendations}")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

```
Recommended Movies: ['Star Wars: Episode III - Revenge of the Sith', 'Iron Man 3', 'Guardians of the Galaxy Vol. 2', 'Iron Man']
Correct Recommendations: ['Iron Man']
Accuracy: 0.25
Precision: 0.25
Recall: 1.00
F1 Score: 0.40
```

5. Precision@K Evaluation

- **Definition**: Precision@K evaluates the relevance of the top **k** recommendations. It checks how many of the top **k** movies are relevant (based on genre).
- **Process**: Calculated how many of the top **k** recommended movies matched the input movie's genre.
- **Results**:

- For **k=5** recommendations for "Iron Man", the **Precision@K** was **0.8**, meaning 80% of the top 5 recommendations were relevant.

Precision@K evaluation

```
from sklearn.metrics import precision_score
import numpy as np
```

```
# Simulated true relevance labels (1 = relevant, 0 = irrelevant) for a movie
# For example, for "Iron Man", you define which movies are truly similar/relevant
true_relevance = [1, 1, 0, 0, 1]  # This needs to be your ground truth (manually created or benchmarked)
```

```
recommended_movies_indices = [new_data[new_data['title'] == 'Iron Man'].index[0]]
distance = sorted(list(enumerate(similarity[recommended_movies_indices[0]])), reverse=True, key=lambda x: x[1])
recommended_movies = [new_data.iloc[i[0]].title for i in distance[1:6]]  # top 5 recommendations
```

```
# For simplicity, let’s simulate predictions (1 = relevant, 0 = irrelevant)
predictions = [1, 0, 1, 0, 1]  # This is derived from your similarity model’s top K recommendations
```

```
precision_at_k = precision_score(true_relevance, predictions, average='binary')
print(f'Precision@K: {precision_at_k}')
```

```
Precision@K: 0.6666666666666666
```

7. Data Saving with Pickle

- After building the similarity matrix, the processed data (new_data) and similarity matrix (similarity) were saved using the pickle library for future use, so the model does not need to be recomputed each time.

```
import pickle
```

```
pickle.dump(new_data, open('movies_list.pkl', 'wb'))
```

```
pickle.dump(similarity, open('similarity.pkl', 'wb'))
```

```
pickle.load(open('movies_list.pkl', 'rb'))
```

	id	title	tags
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...
3	424	Schindler's List	The true story of how businessman Oskar Schind...
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...
...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...
9997	13995	Captain America	During World War II, a brave, patriotic Americ...
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...
9999	455957	Domino	Seeking justice for his partner's murder by an...

10000 rows × 3 columns

Work Package 4: API Deployment and Visualization

API Development:

8. **User Interface Development**: Build an interactive user interface using **Streamlit** where users can:

- Search for a movie from a dropdown menu.
- View recommended movies based on cosine similarity.
- Filter recommendations by release year, genre, and more.

```python
import streamlit as st
import pickle
import requests
import os

# Load movie data and similarity matrix
movies = pickle.load(open("movies_list.pkl", 'rb'))
similarity = pickle.load(open("similarity.pkl", 'rb'))

# List of movie titles
movies_list = movies['title'].values

@st.cache_data(show_spinner=False)
def fetch_movie_details(movie_id):
    url = f"https://api.themoviedb.org/3/movie/{movie_id}?api_key=078dd0cf2fe7278cb4e016a9947d4e35&language=en-US"
    data = requests.get(url).json()
    poster_path = data.get('poster_path')
    genres = [genre['name'] for genre in data.get('genres', [])]
    overview = data.get('overview', "No overview available")
    release_date = data.get('release_date', "No release date available")
    vote_average = data.get('vote_average', "No rating available")

    if poster_path:
        full_path = "https://image.tmdb.org/t/p/w500/" + poster_path
    else:
        full_path = "https://via.placeholder.com/500x750?text=No+Image"

    release_year = release_date.split('-')[0] if release_date != "No release date available" else "N/A"

    return full_path, genres, overview, release_year, vote_average

```

9. **Personalized User Interaction**: Allow users to add movies to their favorites list and display their favorite movies dynamically.

```python
# Home page (Netflix-style suggestion)
def home_page():
    st.title("Discover New Movies and Series")
    st.write("Browse through our collection of movies and series.")

    # Suggest random movies
    st.subheader("Trending Now")
    trending_movies = movies.sample(10)  # Example: select 10 random movies

    cols = st.columns(5)  # Display 5 movies in a row
    for i in range(len(trending_movies)):
        movie_id = trending_movies.iloc[i].id
        movie_title = trending_movies.iloc[i].title
        poster, _, _, _, _ = fetch_movie_details(movie_id)

        with cols[i % 5]:
            st.image(poster, use_column_width=True)
            st.write(movie_title)
```

10. **Reporting Dashboard**:

Provide users with additional information on recommended movies, such as:

- **Genres**, **Release Year**, **Ratings**, and **Movie Overview**.

```python
# Movie recommender page
def recommender_page():
    st.header("Movie Recommender System")
    selectvalue = st.selectbox("Select a movie from the dropdown", movies_list)

    st.subheader("Filter by Release Year")
    years = [str(year) for year in range(1990, 2023)]
    year_from = st.selectbox("From Year:", ["Any"] + years)
    year_to = st.selectbox("To Year:", ["Any"] + years)

    show_genre = st.checkbox("Show Genre")
    show_release_year = st.checkbox("Show Date")
    show_rating = st.checkbox("Show Rating")
    show_overview = st.checkbox("Show Overview")

    def round_rating(rating):
        try:
            return round(float(rating), 1)
        except ValueError:
            return "N/A"

    def recommend(movie):
        index = movies[movies['title'] == movie].index[0]
        distances = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector: vector[1])

        recommended_movies = []
        recommended_posters = []
        recommended_genres = []
        recommended_overviews = []
        recommended_release_years = []
        recommended_ratings = []
```

The whole code of our project is available in this link:

https://github.com/Nada-Mostafa31/MovieRecommendationSys

Conclusion

This project successfully developed a **movie recommendation system** that recommends similar movies based on content (description and genre) and evaluates the recommendation quality using a variety of metrics. The system was deployed using **Streamlit**, providing users with an easy-to-use interface for exploring recommendations and viewing movie details.

Using multiple evaluation techniques, including **Precision@K**, **Hit Rate**, **MAE**, **RMSE**, **Precision**, **Recall**, and **F1-score**, the project offers a well-rounded assessment of the recommendation engine's performance.

Future improvements can include integrating collaborative filtering and additional features such as user ratings and personalized recommendations based on user behavior.