Movie Recommendation System Development

Made by:

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

TABLE OF CONTENTS

01 INTRODUCTION

02 WP1: ETL PIPELINE DEVELOPMENT

03 WP2: DATA PREPARATION AND TRANSFORMATION

04 WP3: RECOMMENDATION ENGINE DEVELOPMENT

05 EVALUATION METRICS OVERVIEW

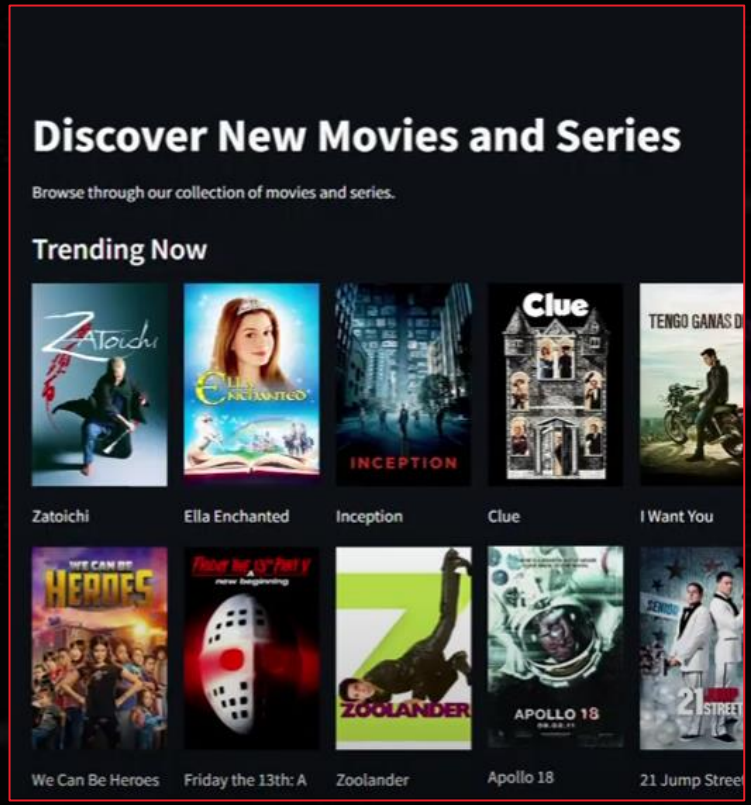
06 WP4: API DEPLOYMENT

01

INREODUCTION

PROJECT OVERVIEW:

- Built a content-based **movie recommendation system** using the **TMDB Movies Dataset**.
- Used **Cosine similarity** for content-based filtering of personalized recommendations.
- Developed an interactive user interface using **Streamlit**.
- Evaluated recommendation quality using multiple metrics including **Hit Rate**, **MAE**, **RMSE**, **Precision@K**, and more.
- Final product provides real-time personalized movie recommendations to users.



02

WP1: ETL PIPELINE DEVELOPMENT

ETL PIPELINE DEVELOPMENT

- Objective: Extract, clean, and load movie data from the TMDB dataset.
- Data Sources: Titles, genres, and overviews from TMDB.
- Tools Used: Python (pandas), Streamlit for the initial setup.
- Deliverables:
  - Functional ETL pipeline to load and preprocess movie data.

```
[ ] import pandas as pd

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

[ ] movies.head(6)

[ ] movies.describe() #describe and count numerical value

[ ] 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
```

03

WP2: DATA PREPARATION AND TRANSFORMATION

DATA PREPARATION AND TRANSFORMATION

- Objective: Clean and transform the dataset for recommendation purposes.
- Steps:
  - Data cleaning (handling missing values).
  - Combining overview and genre columns into a single tags field.
- Tools Used: Python (pandas, NumPy).
- Deliverables: Cleaned and transformed dataset.

```
[8]  movies_cleaned = movies.dropna(subset=['genre', 'overview'])
     ✓ 0.0s

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

[10] movies.isnull().sum()
     ✓ 0.0s

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

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

[ ]  new_data
```

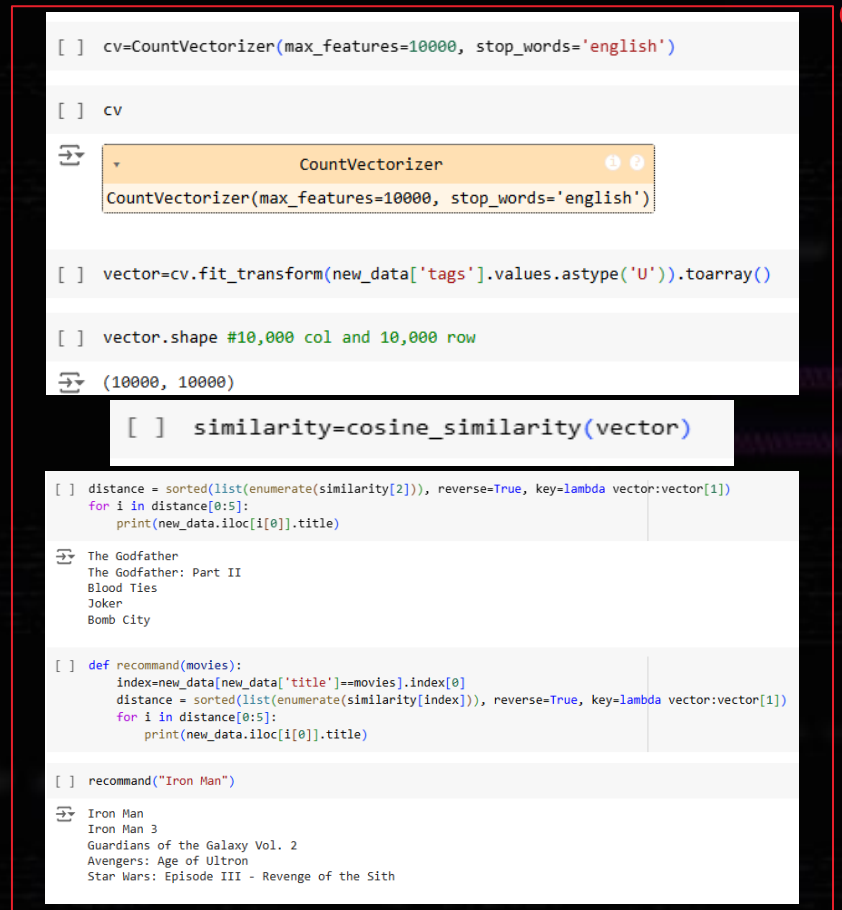
	id	title	tags
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...
3	424	Schindler's List	The true story of how businessman Oskar Schind...

04

WP3: RECOMMENDATION ENGINE DEVELOPMENT

RECOMMENDATION ENGINE DEVELOPMENT

- Objective: Build a content-based recommendation engine.
- Steps:
  - Model Selection: Cosine similarity for content-based filtering.
  - Model Building: Use of CountVectorizer to process movie descriptions and genres.
- Tools Used: Python (scikit-learn, pandas).
- Deliverables: Trained recommendation model with evaluation results.



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

[ ] cv

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)

[ ] similarity=cosine_similarity(vector)

[ ] 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
```

05

EVALUATION METRICS OVERVIEW

EVALUATION METRICS OVERVIEW

- Hit Rate: How many of the recommended movies match the genre of the input movie.
- MAE: Mean Absolute Error between predicted similarity scores and expected relevance.
- RMSE: Root Mean Squared Error to penalize larger errors.
- Precision@K: Proportion of relevant movies in the top k recommendations.
- Precision, Recall, F1-score: Measures of accuracy, completeness, and balance between precision and recall.

```
[ ] # 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
```

Hit Rate evaluation based on genre

```
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)
```

06

WP4: API DEPLOYMENT

API DEPLOYMENT

- Objective: Develop an interactive user interface for the recommendation system.
- Steps:
  - User searches for a movie and views real-time recommendations.
  - Displays movie details: Poster, Genres, Overview, Ratings.
  - Allows users to add favorites and dynamically updates the interface.
- Tools Used: Streamlit, TMDB API.
- Deliverables: Deployed Streamlit application providing movie recommendations.

```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)
```

```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 = []
```

THANKS!

Do you have any questions?