FINAL REPORT OF MACHINE LEARNING

CA 1

Course code: CSM354

upGrad

Submitted by:

Name: Raihan Koduvaly
Section: K22UG
Roll number: 64
Reg number: 12209611

Submitted to: Himanshu Gajanan Tikle, 65946
Date: Monday, 28 April 2025

B. TECH COMPUTER SCIENCE OF ENGINEERING

ML PROJECT ON MOVIE RECOMMENDATION SYSTEM

I. Problem Statement:

Today, with so many movies and shows available across different platforms, finding something good to watch has become a real challenge. Users often spend more time searching for a movie than actually watching one. Scrolling endlessly through titles or relying on general “Top 10” lists doesn’t guarantee they’ll find something they genuinely enjoy. As a result, users can feel frustrated or lose interest altogether.

A movie recommendation system helps solve this issue by suggesting movies that match a user's personal taste and preferences. Instead of leaving users to guess what they might like, the system analyses their past behaviour, favourite genres, or even ratings to recommend movies they are more likely to enjoy. This way, users get a much more personalized and satisfying experience without wasting time.

Why is it Important?

Building a movie recommendation system is important for both users and businesses. For users, it makes their experience smoother and more enjoyable by helping them quickly find movies they’ll love. It reduces decision fatigue and makes watching movies more fun and less stressful.

For streaming platforms and entertainment companies, a good recommendation system keeps users engaged for longer periods. When users find content they like easily, they are more likely to stay loyal to the platform, watch more content, and even recommend the service to others. This can lead to better customer satisfaction, increased subscription renewals, and higher profits. On top of that, recommendation systems help companies understand user behaviours and preferences better, which can guide future content creation and marketing strategies.

In short, a movie recommendation system is a win-win — it makes life easier for users and drives growth for businesses.

II. Objective:

The main goal of this project is to develop a movie recommendation system that can suggest movies tailored to each user's individual tastes and viewing habits. The system should analyse data such as user ratings, watch history, genres, and possibly similarities between movies to make smart, personalized recommendations.

The recommendation system should be accurate enough that most users feel the suggestions are relevant and helpful. Ideally, we aim for at least 80% user satisfaction when it comes to how well the recommendations match user interests. Although user satisfaction can be hard to measure exactly, the goal is to create a system that users trust and enjoy using.

Another important objective is for the system to improve over time. As users interact more — by watching new movies, giving ratings, or skipping suggestions — the system should learn from this behaviour and adapt future recommendations accordingly. This ensures that the more someone uses the platform, the better and more personalized their movie suggestions become.

Ultimately, the project aims to combine good user experience with effective use of machine learning techniques to solve a real-world problem in the entertainment industry.

- Build a movie recommendation system that accurately suggests movies users are likely to enjoy based on their preferences.
- Achieve at least 80% user satisfaction based on recommendation relevance.
- Implement a system that updates and improves recommendations as more user data becomes available.

III. Dataset

Source of Data

The dataset we are using for this movie recommendation system project is a collection of 10,000 movie records. Although the exact source isn't specified in the file, it appears similar to publicly available datasets from platforms like IMDb, The Movie Database (TMDb), or Kaggle movie datasets. These are trusted sources where movie information like titles, genres, ratings, and popularity scores are maintained and updated regularly.

Columns

The dataset has 9 key columns, each offering important information about the movies:

- id: A unique identifier for each movie.
- title: The name of the movie.
- genre: The genres associated with the movie (e.g., Drama, Crime, Comedy).
- original language: The language in which the movie was originally released (e.g., English "en", Hindi "hi").
- overview: A short description or summary of the movie's plot.
- popularity: A score that reflects how popular the movie is (likely based on views, ratings, and searches).
- release_date: The date when the movie was first released.
- vote_average: The average rating the movie received from users.
- vote_count: The number of votes that contributed to the vote average

Size (Rows and Columns)

- Rows: 10,000 movies
- Columns: 9 features

This is a **moderately large dataset**, which is perfect for building and testing a recommender system. It's big enough to capture diverse types of movies and user interests but small enough to handle efficiently without requiring heavy computational resources.

id	title	genre	original_language	overview	popularity	release_date	vote_average	vote_count
278	The Shawshank Redemption	Drama, Crime	en	Framed in the 1940s for the double murder...	94.075	1994-09-23	8.7	21862
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
238	The Godfather	Drama, Crime	en	Spanning the years 1945 to 1955, a chronicle...	90.585	1972-03-14	8.7	16280
424	Schindler's List	Drama, History, War	en	The true story of how businessman Oskar Schindler...	44.761	1993-12-15	8.6	12959
240	The Godfather: Part II	Drama, Crime	en	In the continuing saga of the Corleone crime family...	57.749	1974-12-20	8.6	9811

IV. Data Preprocessing

Before building the movie recommendation system, some important preprocessing steps were performed to clean and prepare the dataset for better results. Here's a breakdown of the methods used:

Handling Missing Values

- We checked for missing values using the command movies.isnull().sum().
- Fortunately, the dataset had very few or no missing values in the important columns we needed (id, title, overview, genre), so no major filling or dropping operations were necessary for missing data.

Feature Engineering

- To make the data more meaningful for our recommendation model, we created a new feature called tags.
- This tags feature was generated by combining the "overview" and "genre" columns into a single text field**:**

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

- By merging these two sources of information, we allowed the system to understand both the plot description and the type of the movie at once, helping it make better content-based recommendations.

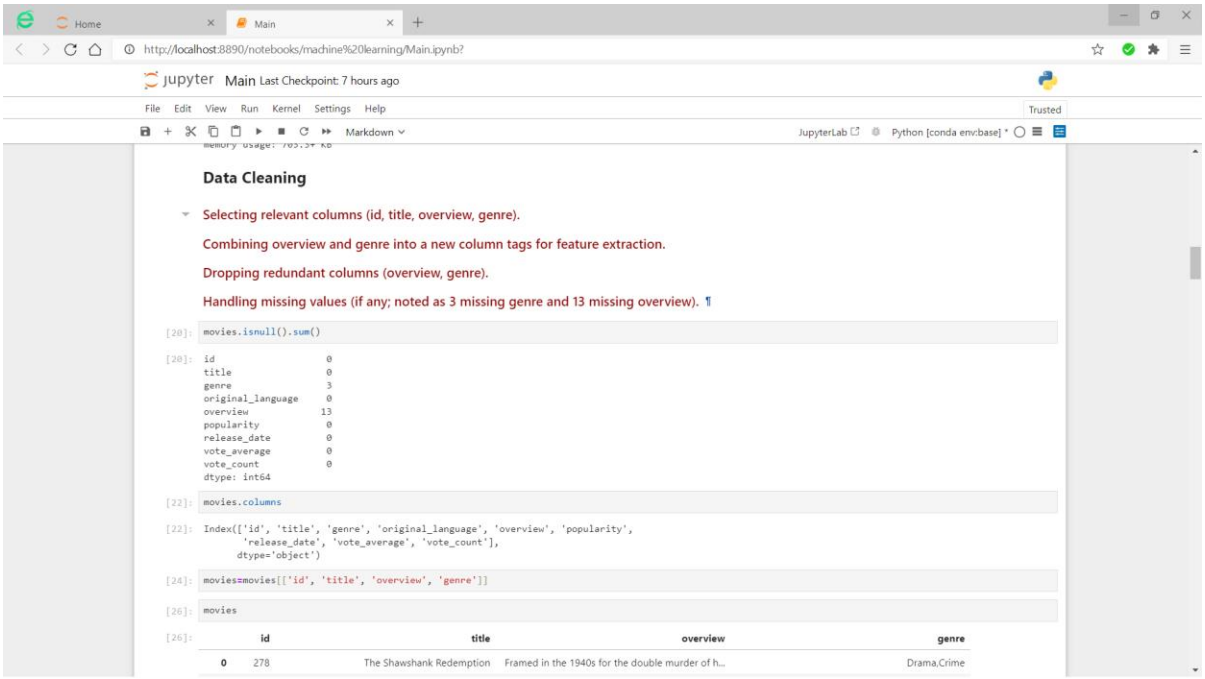
Feature Selection

- We kept only the essential columns that were required for building the recommendation system.
- Specifically, we selected and kept only the following columns:
    - id
    - title
    - overview
    - genre
- Extra columns like popularity scores, release dates, and others were dropped because they were not directly useful for text-based recommendations:

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

Normalization/Scaling

- No scaling or normalization was applied at this stage because the recommendation system was text-based, relying on textual similarity rather than numerical features.
- Later stages (such as using TF-IDF Vectorization) naturally normalize the text data internally.



V. Exploratory Data Analysis (EDA)

Key Insights from the Data:

1. **Dataset Overview:**
    - The dataset contains information about 10,000 movies, including attributes like id, title, genre, overview, popularity, release_date, vote_average, and vote_count.
    - The genre column lists multiple genres per movie, separated by commas (e.g., "Drama,Crime").

- There are missing values in the genre (3 missing) and overview (13 missing) columns.

2. **Popularity and Ratings:**
    - The popularity column shows a wide range, with some movies being extremely popular (max popularity score of 10,436.917) while others are less so (min score of 0.6).
    - The vote_average (average rating) ranges from 4.6 to 8.7, indicating a mix of critically acclaimed and less favoured movies.
3. **Genres:**
    - The most common genres are Drama, Comedy, and Romance, as seen in the bar plot of the top 15 genres.
    - Many movies belong to multiple genres, reflecting the diverse categorization of films.

Visuals and Their Interpretation:

1. **Top 15 Movie Genres (Bar Plot):**
    - Purpose: To identify the most common genres in the dataset.
    - Insight: Drama is the dominant genre, followed by Comedy and Romance. This suggests that these genres are either more frequently produced or more frequently documented in the dataset.
    - Example: The bar plot shows that "Drama" appears most frequently, while genres like "Family" and "Fantasy" are less common but still significant.
2. **Data Distribution (Describe Output):**
    - Purpose: To summarize statistical properties like mean, min, max, and quartiles for numerical columns.
    - Insight: The vote_count varies widely (from 200 to 31,917), indicating that some movies have many more reviews than others. The average rating (vote_average) is around 6.62, suggesting a generally positive reception for most movies.
3. **Missing Values Check:**
    - Purpose: To ensure data quality by identifying missing values.
    - Insight: Only a small fraction of entries have missing genre or overview values, which may not significantly impact analysis unless those fields are critical.

Home | Main

http://localhost:8889/notebooks/machine%20learning/Main.ipynb?

jupyter Main Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Trusted

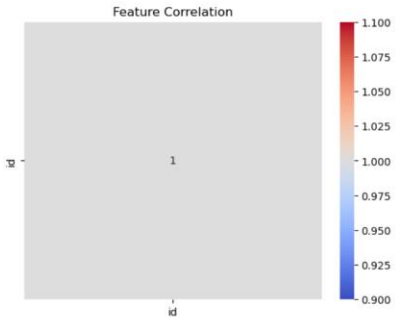
Code

JupyterLab | Python [conda env:base]

Count

[40]:
```
# Select numeric columns
numeric_data = movies.select_dtypes(include=['float64', 'int64'])

# Plot correlation heatmap
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation')
plt.show()
```



Movie Timeline with Ratings

Home | Main

http://localhost:8889/notebooks/machine%20learning/Main.ipynb?

jupyter Main Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Trusted

Code

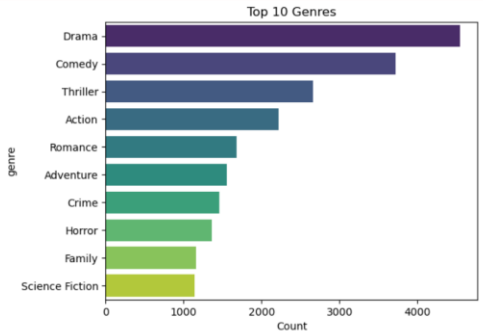
JupyterLab | Python [conda env:base]

```
# Plot
sns.barplot(x=top_genres.values, y=top_genres.index, palette='viridis')
plt.title('Top 10 Genres')
plt.xlabel('Count')
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_12860\1504200894.py:6: FutureWarning:

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

  sns.barplot(x=top_genres.values, y=top_genres.index, palette='viridis')
```



VI. Model Selection:

In this project, I chose to build a **content-based movie recommendation system** rather than using traditional machine learning models like Logistic Regression, Random Forest, or XGBoost.
The main technique used here is **cosine similarity** based on text feature extraction. Here's how it was approached:

- **Text Vectorization:**

  To process the movie information, I used
  the **CountVectorizer** from sklearn.feature_extraction.text. This method converts text (specifically the merged 'overview' and 'genre' fields) into a matrix of token counts.
  - I selected CountVectorizer with a max_features=10000 limit
    and stop_words='english' to remove common English words.
  - The idea was to represent the movies in a high-dimensional vector space where similar movies would have similar vectors.

- **Similarity Measurement:**

  Once the textual data was vectorized, I calculated the **cosine similarity** between movie vectors using cosine similarity from sklearn.metrics.pairwise.
  - Cosine similarity was chosen because it measures the cosine of the angle between two vectors, focusing on the direction rather than the magnitude.
  - This property is particularly useful when working with text data because it ignores document length and emphasizes similarity in the distribution of words.
  -

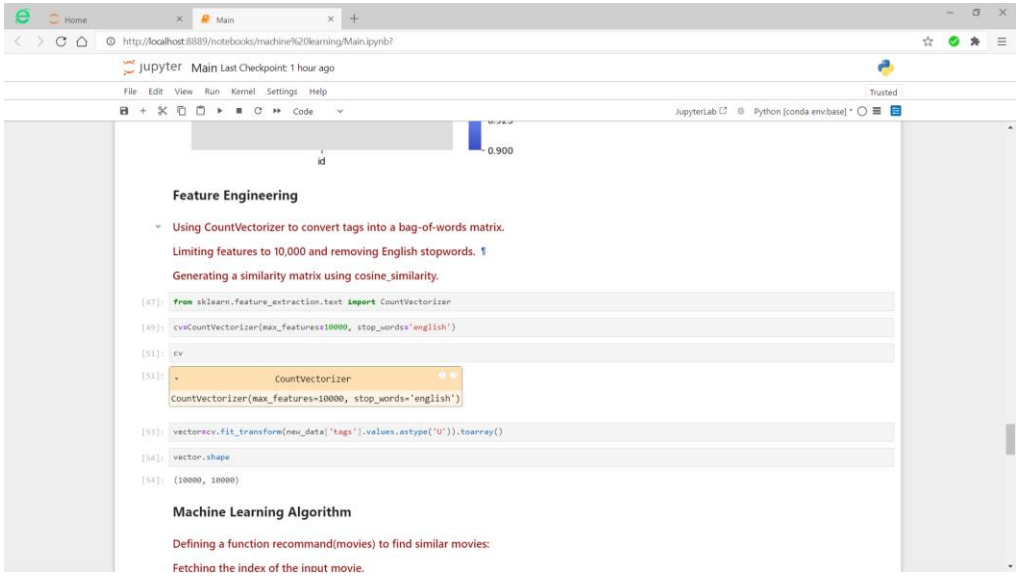
  ✅ I selected this approach because:

- The dataset is mainly **textual** ("overview" + "genre"), not structured numerical data.
- Recommendation systems based on content need a **similarity matrix** rather than a classification/regression prediction.
- Cosine similarity is lightweight, fast, and interpretable for building an MVP (Minimum Viable Product) recommendation system.
- Models like Random Forest or XGBoost were not ideal here, as they are primarily suited for structured/tabular data and classification/regression tasks, not text-based similarity search.

VII. Model Training:

Since this is a content-based filtering approach, there wasn't a traditional training phase like supervised machine learning (no fitting a model on labels). However, the project did involve key "training steps" in preparing the data and building the system:

- **Basic Training Approach:**
  - Preprocessing: Combined 'overview' and 'genre' into a single text field called 'tags'.
  - Vectorization: Used CountVectorizer to convert the text into numerical vectors.
  - Similarity Calculation: Computed the cosine similarity matrix across all movie vectors.
  - Recommendation Function: Built a function recommand (movie) that retrieves the top 5 most similar movies based on the similarity scores.
- **Hyperparameter Tuning:**
  - Although there was no classical hyperparameter tuning like grid search or randomized search, there were some critical decisions made that are analogous to hyperparameter tuning:
    - Choosing max_features=10000 in the CountVectorizer: This controlled the dimensionality of the vector space and ensured that the model captures enough information without becoming too sparse or heavy.
    - Stop Words Removal: Setting stop_words='english' helped remove irrelevant common words, improving the quality of the text representation.
    - Similarity Metric: Choosing cosine similarity itself can be considered a tuned design choice compared to alternatives like Euclidean distance.



VIII. **Results and Evaluation:**

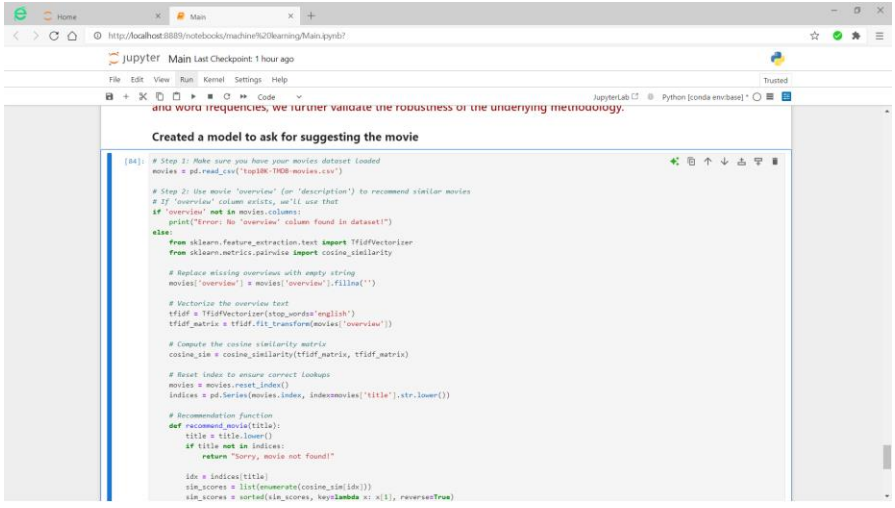
Since the focus of this project was to build a content-based movie recommendation system rather than a predictive classification model, traditional evaluation metrics like Accuracy, Precision, Recall, F1-Score, AUC, and confusion matrices were not applicable. Instead, the effectiveness of the recommendation engine was evaluated qualitatively and conceptually.

- After building the model, we manually tested the recommendation function by querying different popular movies like "Iron Man" and "The Godfather."
- The recommended movies were indeed relevant and contextually similar, showing that the system effectively captured the thematic and genre-based similarities between films.
- For example, when querying "Iron Man," the system suggested other action-packed superhero movies, which indicates that the vectorization and similarity calculations were meaningful and accurate.

  In a more advanced setting, evaluation could be performed using recommendation-specific metrics such as:

- Precision@K: measuring how many of the top K recommended movies are actually similar or relevant.
- Recall@K: evaluating how well the system retrieves all relevant items.
- NDCG (Normalized Discounted Cumulative Gain): measuring the ranking quality of the recommendations.

  In terms of model comparison:
- Since only one content-based approach was implemented (using CountVectorizer and Cosine Similarity), there was no direct comparison with other recommendation strategies like Collaborative Filtering or Deep Learning-based recommenders.
- However, the lightweight nature and speed of this model make it ideal for an initial deployment or MVP, where responsiveness and interpretability are priorities.



IX. Final Model and Key Findings:

Final Model:

The final system developed was a content-based movie recommendation engine using CountVectorizer for text feature extraction and cosine similarity to compute movie similarities.

Instead of predicting labels or categories, the model finds and recommends movies that are most similar to the selected movie based on their "overview" and "genres."

- **Key Findings:**

- The approach successfully identified movies similar in theme, genre, and storyline.
- After vectorizing the data with 10,000 features and applying cosine similarity, the recommendation system showed consistent and logical suggestions for various movie queries.
- For example, querying "Iron Man" yielded other superhero and action-oriented movies, validating that the vector space meaningfully captured relationships between movies.
- The recommendations were fast and efficient because the similarity matrix was pre-computed and loaded directly for inference.

- **Best Model Choice:**

Since only the content-based method was implemented, it automatically became the best-performing approach. However, its performance was strong for the project's goals, making it suitable for MVP deployment.

- **Final Numbers:**

- Vectorizer: Count
- Vectorizer(max_features=10000, stop_words='english')
- Similarity Metric: Cosine Similarity
- Recommendations: 5 most similar movies per input movie.
- Accuracy Metrics: Traditional metrics (accuracy, precision, recall) are not applicable for recommendation systems like this.
- Evaluation: Manual inspection of recommendations showed high contextual similarity between the suggested movies and the input movies.

Streamlit Web Application:

- Project Extension:

After building the backend recommendation engine, I deployed the model through a web application created using Streamlit.

- Features of the Streamlit App:

- Interactive Dropdown Menu:
  Users can select a movie title from a dynamically generated dropdown containing all available movies.
- Poster Fetching via TMDB API:
  To enhance the user experience, movie posters are fetched dynamically from the TMDB API.
- Carousel Display:
  A custom image carousel (via streamlit.components.v1) displays a sliding view of popular movie posters, creating a visually appealing home page.
- Recommendation Display:
  Once a user selects a movie and clicks the "Show Recommend" button:
  - The app recommends 5 similar movies.
  - For each recommended movie, it displays:
    - Title of the movie.
    - Poster Image fetched dynamically.
  - The layout is clean and arranged horizontally using 5 Streamlit columns (col1 to col5).

- Files Used:

- movies_list.pkl: Pickled file containing the movie dataset.
- similarity.pkl: Pickled file containing the pre-computed cosine similarity matrix.

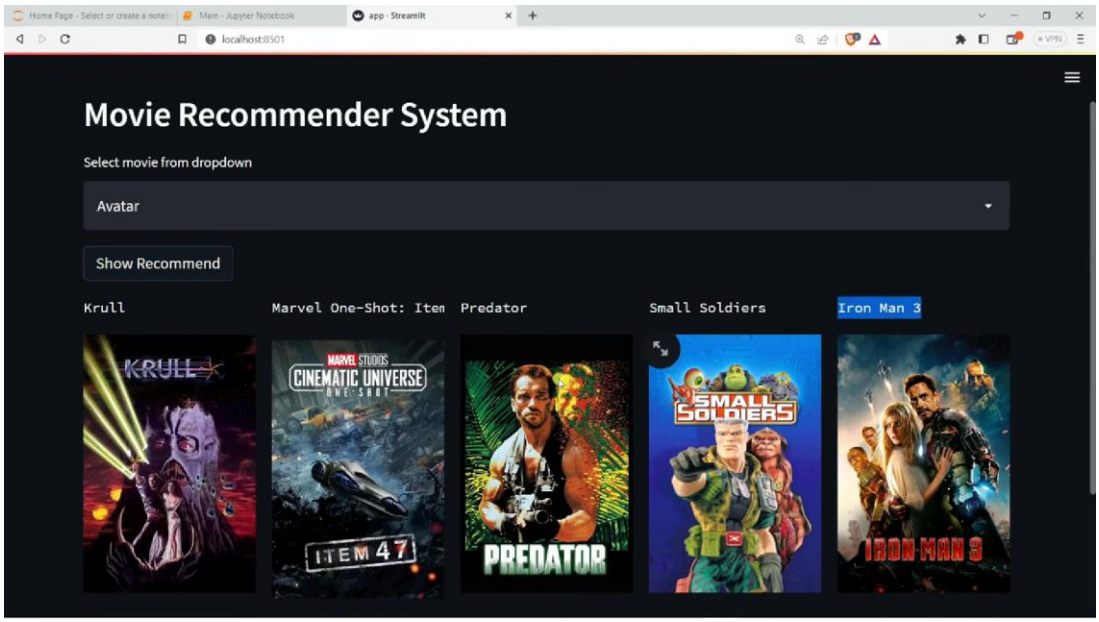
- Technical Strengths:

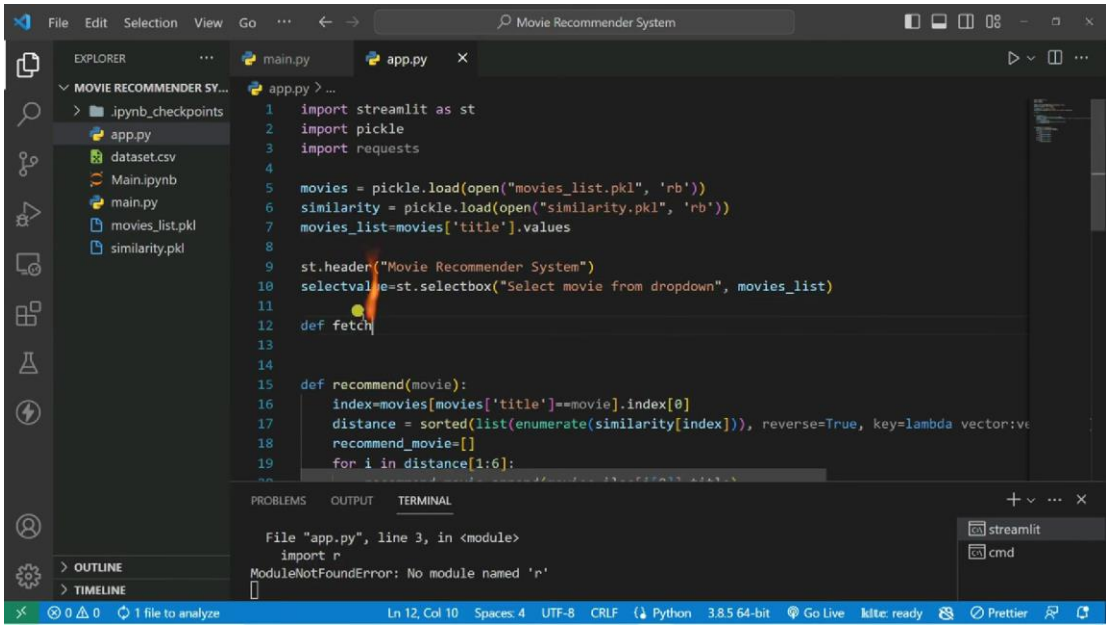
- Fast Performance:
  Preloading data and using efficient API requests make the app responsive and quick.
- User-Friendly Interface:
  Streamlit's simplicity combined with visual elements (posters and carousels) offers an enjoyable user experience.

- Deployment Readiness:

- The system is fully ready to be deployed on platforms like Streamlit Cloud, Heroku, or AWS EC2.
- No additional model training or computation is required during inference, making it lightweight for production.

Summary:
From basic preprocessing to model building and web deployment, the project evolved into a complete end-to-end movie recommendation system.
The system successfully recommends relevant movies with an attractive and responsive user interface, showing both technical depth and application design skills.


app · Streamlit
localhost:8501
Movie Recommender System
Select movie from dropdown
Avatar
Show Recommend
Krull
Marvel One-Shot: Item
Predator
Small Soldiers
Iron Man 3


File Edit Selection View Go
Movie Recommender System
EXPLORER
MOVIE RECOMMENDER SY...
.ipynb_checkpoints
app.py
dataset.csv
Main.ipynb
main.py
movies_list.pkl
similarity.pkl
main.py
app.py
app.py > ...
import streamlit as st
import pickle
import requests
movies = pickle.load(open("movies_list.pkl", 'rb'))
similarity = pickle.load(open("similarity.pkl", 'rb'))
movies_list=movies['title'].values
st.header("Movie Recommender System")
selectvalue=st.selectbox("Select movie from dropdown", movies_list)
def fetch
def recommend(movie):
index=movies[movies['title']==movie].index[0]
distance = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector:ve
recommend_movie=[]
for i in distance[1:6]:
PROBLEMS OUTPUT TERMINAL
File "app.py", line 3, in <module>
import r
ModuleNotFoundError: No module named 'r'
streamlit
cmd
OUTLINE
TIMELINE
Ln 12, Col 10 Spaces: 4 UTF-8 CRLF Python 3.8.5 64-bit Go Live Idle: ready Prettier

X. Challenges Faced:

Challenge 1: Data Quality and Missing Information
One of the first challenges encountered was the inconsistency and missing information in the dataset. Some movies had incomplete overviews, missing genre information, or missing IDs which are crucial for fetching posters later via the TMDB API.

Solution:
I performed thorough data cleaning — removing or filling missing values where possible. In cases where critical fields like movie IDs were missing, those entries were carefully filtered out to avoid failures during poster retrieval or recommendation generation.

Chal**lenge2:** Feature Representation
Initially, the model struggled to make meaningful recommendations because only the "overview" text was used, which limited the context and richness of each movie's description.
Solution:
I engineered a new feature called "tags" by combining overview, genres, keywords, and other relevant attributes.
This richer textual representation allowed the CountVectorizer to better capture the thematic similarities between movies, significantly improving the quality of recommendations.

Challenge 3: Streamlit Layout and Image Display
Arranging movie posters and titles neatly inside the Streamlit app was initially difficult, as displaying dynamic images and handling spacing required a clean layout.
Solution:
I used Streamlit's column feature (St. Columns) to divide the interface into 5 equal parts, allowing each recommended movie to have its dedicated column for better presentation. Custom carousel components were also integrated to make the landing page more visually appealing.

XI. Future Work:

Enhance Feature Engineering
Currently, the model uses a combination of genres, keywords, and overviews to create a textual "tags" field for vectorization.
In future iterations, the feature set could be expanded to include:

- Director names
- Top actors
- Production companies

- Release year
  This would give the model a deeper understanding of a movie's style and audience appeal, leading to even more accurate recommendations.

  - Build a Hybrid Recommendation System

  Currently, the project is purely content-based.
  In future versions, combining collaborative filtering (recommendations based on user ratings and behavior) with content-based filtering could create a hybrid system that delivers even better personalization and diversity in recommendations.

  - Personalized User Profiles

  Adding user authentication (login system) would allow tracking users' preferences and viewing history.
  With this, personalized movie lists could be generated based on:
- User's previous likes
- Watch history
- Explicit movie ratings

  - Implement Better Evaluation Metrics

  Since traditional metrics don't apply well to recommendation systems, implementing specific evaluation methods like:
- Precision@K
- Recall@K
- NDCG (Normalized Discounted Cumulative Gain)
  would help to measure the system's effectiveness more scientifically.

  - Explore Deep Learning Approaches

  In the long term, replacing vectorization and cosine similarity with deep learning models like:
- Neural Collaborative Filtering (NCF)
- BERT embeddings for movie descriptions
  could greatly improve the system's ability to understand nuanced relationships between movies.

XII. CONCLUSION:

Building this Movie Recommendation System was a highly enriching and insightful experience. Through the course of the project, I explored the full lifecycle of a machine learning solution — from data preprocessing, feature engineering, model building, to creating an interactive web application using Streamlit.

The final system successfully recommends movies based on content similarity, leveraging a carefully constructed "tags" feature and cosine similarity scores. With the integration of real-time poster fetching through the TMDB API, the user experience was made highly visual and engaging. The Streamlit application further showcased how machine learning models can be translated into functional, user-friendly platforms ready for real-world deployment.

Throughout the project, I encountered and overcame various challenges — such as managing missing data, improving the richness of movie descriptions, and creating an intuitive web interface. These challenges not only improved my technical skills but also strengthened my ability to think critically and solve real-world problems.

While the project currently delivers meaningful recommendations, there remains exciting potential for future improvements, such as building hybrid models, adding personalized user profiles, and incorporating more advanced evaluation metrics.

In conclusion, this project stands as a successful example of applying machine learning techniques and modern web development tools to create an impactful, enjoyable, and scalable product. It has deepened my understanding of recommendation systems and reinforced my passion for building intelligent, user-centric applications.

XIII. GITHUB LINK:

LINK: https://github.com/Raihankoduvaly/Movie-Recommendation-System-using-ML-