1. INTRODUCTION

1.1 ORGANIZATION PROFILE

The Movie Recommendation System Analysis project is developed by a forward-thinking organization dedicated to leveraging data science and analytics for enhancing user experiences in the entertainment industry. The organization specializes in creating intelligent, user-focused solutions powered by advanced programming tools and data analysis libraries such as NumPy, Pandas, Matplotlib, and Seaborn. With a mission to make movie selection effortless and personalized, the organization integrates cutting-edge technologies to analyze large datasets, visualize trends, and generate meaningful insights. Committed to innovation and excellence, the organization aims to transform the way users discover and enjoy movies, delivering an engaging, data-driven platform tailored to diverse preferences and viewing habits.

MISSION, VALUES & OBJECTIVES

MISSION

To create a highly efficient and user-centric movie recommendation system that leverages advanced data analytics and visualization tools, enabling personalized movie suggestions while enhancing the overall entertainment experience.

VALUES

- **Innovation:** Embrace cutting-edge technologies like NumPy, Pandas, Matplotlib, and Seaborn to deliver intelligent and scalable solutions.
- **User-Focus:** Prioritize user satisfaction by offering tailored movie recommendations based on diverse preferences and behaviors.
- **Accuracy:** Ensure reliable and precise insights by analyzing and visualizing data comprehensively.
- **Accessibility:** Provide an intuitive and easy-to-use interface that caters to users with varying levels of technical expertise.
- **Data Integrity:** Maintain high standards for data privacy and integrity throughout the recommendation process.

OBJECTIVES

Personalized Recommendations

Provide movie suggestions based on user preferences, watch history, and ratings.

Efficient Filtering Mechanism

Implement content-based and collaborative filtering to suggest relevant movies.

User Engagement & Retention

Enhance user experience by keeping them engaged with tailored recommendations.

Diverse Movie Suggestions

Offer a variety of genres, languages, and categories to suit different tastes.

Real-Time Recommendation Updates

Adapt recommendations dynamically based on user interactions and new releases.

User-Friendly Interface

Design an intuitive and visually appealing UI for easy navigation and selection.

Popularity-Based Suggestions

Recommend trending and highly-rated movies based on global and local trends.

Multi-Platform Accessibility

Ensure the system works seamlessly across web, mobile, and smart TV platforms.

Social Integration

Allow users to share recommendations, reviews, and ratings with friends.

Feedback & Improvement

Continuously learn from user feedback to refine the recommendation accuracy.

1.2 SYSTEM SPECIFICATION

1.2.1 HARDWARE CONFIGURATION

MONITOR	: LED MONITOR IS COLOR
PROCESSOR	: i3 Processor 5th GENERATION
HARD DISK	: 1 TB
MOUSE	: OPTICAL MOUSE
KEYBOARD	: 11.4 MULTIMEDIA
RAM	: 4 GB DDR4

1.2.2 SOFTWARE SPECIFICATION

OPERATING SYSTEM	: WINDOWS 10
LIBRARY	: NUMPY, PANDAS, MATPLOTLIB& SEABORN
BACK-END	: PYTHON 3.12
IDES/EDITOR	: JUPYTER
SERVER	: LOCALHOST

SOFTWARE DESCRIPTION

PYTHON

Python is the core programming language used to develop the Face Detection project. Python's readability and ease of use allow for rapid development of complex applications. For this project, Python integrates with OpenCV, an open-source computer vision library, to handle video capture, face detection, and image processing tasks.

- **Python Libraries:** The project uses Python's built-in capabilities along with additional libraries like OpenCV and NumPy.
- **Real-Time Processing:** Python processes video frames from the webcam in real-time, applying face detection algorithms using the Haar Cascade Classifier available in OpenCV.
- **Interactive Output:** Python provides the logic to display the webcam feed with the detected faces highlighted by bounding boxes, making it user-friendly and intuitive.

Python provides an effective platform for integrating computer vision tasks, simplifying the development process, and ensuring flexibility for future enhancements.

INTRODUCTION TO PYTHON

Python is a high-level, versatile programming language that is widely used for various applications, including web development, data analysis, automation, artificial intelligence, and computer vision. In this project, Python serves as the foundation for implementing the Face Detection system using OpenCV. Python's simplicity and extensive support for third-party libraries, including OpenCV, make it an ideal choice for building a real-time face detection application. This project demonstrates the power and flexibility of Python in handling real-time image and video processing tasks while integrating machine learning-based algorithms for efficient face detection.

ADVANTAGES

- **Simplicity and Readability**: Python's clean and easy-to-understand syntax makes it suitable for both beginners and experienced developers. This ease of use accelerates the development process, ensuring faster implementation of face detection features.
- **Extensive Ecosystem**: Python has a vast collection of libraries for scientific computing, machine learning, and computer vision (e.g., OpenCV, TensorFlow). This makes it an excellent choice for building computer vision applications like face detection.

- **Rapid Prototyping**: Python allows quick prototyping and iteration of ideas. This project can be developed and tested in a short time, making Python ideal for proof-of-concept or real-time systems like face detection.
- **Cross-Platform**: Python is compatible with all major operating systems, including Windows, macOS, and Linux, making it versatile for deployment on different platforms.
- **Community Support**: Python has a large and active community, providing ample resources, documentation, and support for developers working on machine learning and computer vision projects.

BENEFITS

- **Ease of Development**: Python’s intuitive syntax and structure lead to faster development times. This allows for rapid debugging and testing of face detection features.
- **Open-Source**: Python and the libraries used in this project, such as OpenCV, are open-source and free to use. This makes it a cost-effective solution for developers and researchers.
- **Integration with Machine Learning**: Python’s capabilities in machine learning and AI make it a perfect choice for integrating advanced techniques like facial recognition, emotion detection, and other AI-driven tasks into the project in the future.
- **Flexible for Customization**: Python offers high flexibility for customization. The face detection system can be enhanced with additional functionalities like face recognition, emotion detection, or even integration with a larger security system.
- **Real-Time Performance**: Python, combined with OpenCV, allows real-time face detection with efficient video capture and processing, ensuring smooth user experiences in applications like security systems or user interface enhancements.

FEATURES

- **Real-Time Face Detection**: Python uses OpenCV to process live video from the webcam, detecting faces in real-time with high accuracy using the Haar Cascade Classifier algorithm.
- **Bounding Box Highlighting**: Detected faces are marked with bounding boxes, which are drawn on the webcam feed to visually highlight them for the user.
- **Webcam Integration**: Python interfaces with the computer's webcam to continuously capture and process video frames for face detection, enabling live feedback.

- **Cross-Platform Compatibility**: The Python-based system can run on various platforms (Windows, macOS, Linux) without major modifications, ensuring broader accessibility.
- **Simple User Interface**: The project provides an intuitive interface, where the user can easily see the webcam feed with detected faces and exit the application by pressing a key (e.g., 'q').
- **Easy to Extend**: The face detection system can be easily extended to include more sophisticated functionality, such as face recognition, emotion detection, or integration with external devices like security cameras.
- **Efficient Resource Usage**: Python's optimization, combined with OpenCV, ensures that the project can run efficiently, even on systems with lower computational power, making it suitable for a wide range of hardware.

NUMPY, PANDAS, MATPLOTLIB & SEABORN

- **NumPy**: A foundational library for numerical computations, offering support for multi-dimensional arrays and matrices. It provides highly optimized operations for linear algebra, random number generation, and mathematical computations.
- **Pandas**: A versatile data analysis library that simplifies handling structured data, offering tools for data cleaning, preparation, and transformation using its intuitive DataFrame and Series objects.
- **Matplotlib**: A comprehensive plotting library that supports the creation of static, animated, and interactive visualizations. It offers flexibility for creating charts like histograms, scatterplots, and bar graphs with detailed customizations.
- **Seaborn**: Built on top of Matplotlib, Seaborn specializes in creating statistical plots with ease. It provides advanced visualization capabilities, including heatmaps, pair plots, and regression plots, with an emphasis on aesthetics and simplicity.

INTRODUCTION TO NUMPY, PANDAS, MATPLOTLIB & SEABORN

NumPy, Pandas, Matplotlib, and Seaborn are core Python libraries that have revolutionized the field of data science and analytics. They provide tools for numerical computation, data manipulation, and data visualization, enabling developers and analysts to work efficiently with structured and unstructured data. Together, these libraries are integral to building robust analytical pipelines and extracting actionable insights from data.

ADVANTAGES

- **Performance:** NumPy and Pandas ensure optimized performance when processing large datasets.
- **Ease of Use:** Intuitive syntax and functions make these libraries accessible to both beginners and experts.
- **Customization:** Matplotlib and Seaborn allow precise customization of plots to meet complex requirements.
- **Interconnectivity:** Seamless integration between these libraries enables smooth workflows.
- **Cross-Platform Compatibility:** These libraries are compatible with various environments, including Jupyter Notebooks, IDEs, and production systems.

BENEFITS

- **Data Exploration:** Simplifies exploring, summarizing, and understanding datasets.
- **Actionable Insights:** Facilitates in-depth analysis and visualization, aiding in data-driven decision-making.
- **Error Reduction:** Streamlines data preprocessing and visualization, minimizing manual errors.
- **Increased Productivity:** Accelerates workflows with pre-built functions and high-level tools.
- **Wide Applicability:** Suitable for various domains, including finance, healthcare, marketing, and more.

FEATURES

NumPy:

- Efficient handling of large numerical datasets using n-dimensional arrays.
- Support for broadcasting and vectorized operations, improving computation speed.
- Mathematical functions for algebra, statistics, and probability.

Pandas:

- Powerful data manipulation tools like filtering, grouping, and merging.
- Support for time-series analysis and handling missing data.
- Compatibility with various file formats like CSV, Excel, and JSON.

2. SYSTEM STUDY

2.1 EXISTING SYSTEM

Existing movie recommendation systems utilize various techniques to suggest films based on user preferences, watch history, and ratings. These systems primarily rely on collaborative filtering, which analyzes user behavior and similarities between viewers to provide recommendations, and content-based filtering, which suggests movies with similar genres, actors, or directors. Many streaming platforms, such as Netflix and Amazon Prime, use hybrid recommendation models that combine these approaches with artificial intelligence and machine learning algorithms for more accurate predictions. Additionally, some systems incorporate popularity-based recommendations, where trending or highly-rated movies are suggested to all users. Modern movie recommendation systems also use natural language processing (NLP) to analyze reviews and social media sentiments to refine suggestions. Despite these advancements, existing systems sometimes struggle with issues like the cold start problem, where new users or new movies lack sufficient data for accurate recommendations. They may also reinforce filter bubbles, limiting users to similar types of content rather than diversifying their watch choices. These limitations highlight the need for

2.1.1 DRAWBACKS

- **Limited Personalization**: Recommendations are not tailored to user preferences or behaviors.
- **Static Analysis**: Reliance on predefined lists or ratings without considering dynamic user interactions.
- **Inefficient Data Handling**: Inability to process and analyze large datasets efficiently.
- **Lack of Visualization**: Minimal use of visual tools to analyze trends or user behavior effectively.
- **Low Accuracy**: Recommendations often miss the mark, leading to user dissatisfaction.

2.2 PROPOSED SYSTEM

The proposed movie recommendation system aims to enhance accuracy, personalization, and diversity by integrating advanced artificial intelligence (AI) and deep learning techniques. This system will use a hybrid recommendation model, combining collaborative filtering, content-based filtering, and deep learning algorithms to provide highly relevant movie suggestions. To overcome the cold start problem, it will implement context-aware recommendations, analyzing user demographics, preferences, and real-time behavior to suggest movies even for new users. Additionally, sentiment analysis using Natural Language Processing (NLP) will analyze movie reviews and social media trends to refine recommendations. The system will also focus on diverse content exposure, preventing filter bubbles by occasionally suggesting movies outside a user’s usual preferences. A user-friendly interface with multi-platform accessibility (mobile, web, and smart TVs) will ensure seamless integration and ease of use. Moreover, real-time updates will adapt recommendations based on user interactions, providing a dynamic and engaging experience. With these enhancements, the proposed system will offer a more intelligent, adaptable, and enjoyable movie discovery experience for users.

2.2.1 FEATURES

- **Dynamic Recommendations**: Uses collaborative filtering and content-based filtering for personalized suggestions.
- **Efficient Data Processing**: Employs **Pandas** and **NumPy** for handling and analyzing large datasets.
- **Advanced Visualizations**: Leverages **Matplotlib** and **Seaborn** to create insightful graphs and charts.
- **User-Centric Design**: Tailored recommendations based on historical preferences and behavior.
- **Scalability**: The system can handle large datasets and supports future expansion.
- **Trend Analysis**: Provides detailed insights into user preferences and movie trends.
- **Interactive Visuals**: Generates interactive and aesthetic plots to engage users effectively.

3. SYSTEM DESIGN AND DEVELOPMENT

The system is designed as a data-driven movie recommendation platform, leveraging NumPy, Pandas, Matplotlib, and Seaborn for data handling, analysis, and visualization. The backend processes data using Pandas and NumPy, applying filtering algorithms like collaborative and content-based filtering. Data visualization is implemented using Matplotlib and Seaborn to display user trends and recommendations effectively. The system follows a modular architecture for scalability and maintainability.

3.1 FILE DESIGN

The system organizes files in a structured format for easy management:

- **Data Files**: Includes CSV or JSON files for storing movie metadata, user ratings, and interaction logs.
- **Code Files**: Python scripts for data preprocessing, algorithm implementation, and visualization.
- **Output Files**: Visualizations and logs of recommendation results.
- **Configuration Files**: Settings for algorithms and data paths.

3.2 INPUT DESIGN

User Inputs:

- User preferences such as favorite genres or previously watched movies.
- Ratings or feedback on recommended movies.

Data Inputs:

- Movie datasets containing metadata like genres, ratings, and descriptions.
- User interaction logs for collaborative filtering.

3.3 OUTPUT DESIGN

- **Recommendations**: A personalized list of movies tailored to the user's preferences.
- **Visualizations**:
  - Graphs showing trends like most-watched genres or highest-rated movies using **Seaborn**.
  - Heatmaps or scatterplots representing correlations between user preferences and movie ratings.

3.4 SYSTEM DEVELOPMENT

The system development follows an agile approach:

- **Data Collection and Preprocessing**: Datasets are cleaned and formatted using **Pandas**.
- **Algorithm Implementation**: Recommendation algorithms (collaborative and content-based) are developed using **NumPy** for efficient computations.
- **Visualization Development**: Insights are visualized with **Matplotlib** and **Seaborn**.
- **Integration**: Combining modules into a seamless recommendation workflow.

3.4.1 DESCRIPTION OF MODULES

Data Preprocessing Module:

- Cleans and formats datasets using Pandas.
- Handles missing values and duplicates.

Recommendation Engine:

- Implements collaborative and content-based filtering.
- Uses NumPy for matrix operations and similarity calculations.

Visualization Module:

- Generates graphs and charts using Matplotlib and Seaborn.
- Displays user trends and movie patterns.

User Interaction Module:

- Captures user preferences and feedback.
- Updates the recommendation engine dynamically.

Output Generation Module:

- Produces recommendation lists and visual insights for users.

4. SYSTEM TESTING AND IMPLEMENTATION

SYSTEM TESTING

All the modules of this system were successfully implemented and testing of the project completed using test data as well as real data collected from the college. All the reports and the screens are tested for their validity and values in the data tables are checked for their correctness and consistency. After successful testing of the system, it is ready for implementation.

TYPES OF TESTING

- Unit testing
- Integration testing
- User Acceptance testing
- White box testing
- Black box testing

UNIT TESTING

In this testing, the modules in the project are tested independently with each and every forms associated with their functions and sub routines and then it gives the valid accurate data. This concentrates or focuses on individual modules independently, to locate for the errors. No errors found.

INTEGRATION TESTING

The individual modules namely staff module, calendar module, remainder module were linked together, at last the entire forms got together. Initially low volume of data is given to all the modules and the outcome is taken to consideration with the old one. Calendar module is affected when the transactions were made. The data retrieval from the table was slightly changed due to the order of parameters in the subroutine. It was observed and resolved. The candidate system has passed the test.

VALIDATION TESTING

At the culmination integration testing, software is assembling as a package, interface errors have been unconverted and corrected and a final tests- validation test begin. Validation test can be defined in many ways, but a simple definition is that validation successes when the software functions in a manner that can be reasonably expected by the customer. After validation test has been conducted, one of the possible conditions exists.

USER ACCEPTANCE TESTING

User acceptance of the system is a key factor for the success of any system. The system is tested for user acceptance by constantly keeping touch with prospective system and user at the time of developing and making changes whenever required. This is done regarding to the following points.

- Input Screen Design
- Output Screen Design
- Format of the report and other output
- User accepted the system satisfactorily

WHITE BOX TESTING

In this test, all the logical decisions involved in the project is tested by giving true and false sides of data and all the loops are executed. When anywhere in the project, the data of the procedure call went beyond the limit, it was caught and produce the alert signal to user.

BLACK BOX TESTING

It is not an alternative to white box techniques It attempts to find error in the following categories.

Incorrect or missing functions =>No error were detected.

Interface errors=>a recordseterror occurred as recordset was not closed while navigating user interface screen and it was corrected.

Error in data as structures => Validation check was placed where the system required input from the user. So that the occurrence of error in data was avoided.

SYSTEM IMPLEMENTATION

The movie recommendation system was implemented using Python's NumPy, Pandas, Matplotlib, and Seaborn libraries to handle data, generate recommendations, and visualize trends. The process began with gathering and preparing datasets, which included movie metadata and user ratings. Pandas was used to clean and preprocess the data, handling missing values, duplicates, and irrelevant information. After preprocessing, the recommendation engine was developed using NumPy to compute user similarity scores and generate recommendations based on collaborative filtering (user-based and item-based) and content-based filtering methods.

Visualization tools such as Matplotlib and Seaborn were incorporated to display user preferences, movie rating distributions, genre popularity, and other analytical insights in graphical formats.

5. CONCLUSION

The project entitled **“MOVIE RECOMMENDATION SYSTEM”** plays a crucial role in enhancing user experience by providing personalized and relevant movie suggestions. While existing systems effectively use collaborative and content-based filtering, they face challenges such as the cold start problem and filter bubbles. The proposed system addresses these limitations by integrating AI-driven hybrid recommendation models, sentiment analysis, and real-time behavioral tracking to improve accuracy and diversity. By leveraging deep learning and NLP techniques, the system ensures dynamic, context-aware recommendations while maintaining a user-friendly and multi-platform interface. With continuous learning and adaptation, the system will significantly enhance movie discovery, keeping users engaged and helping them explore a wider range of content tailored to their preferences.

BIBLIOGRAPHY

I. BOOK REFERENCES

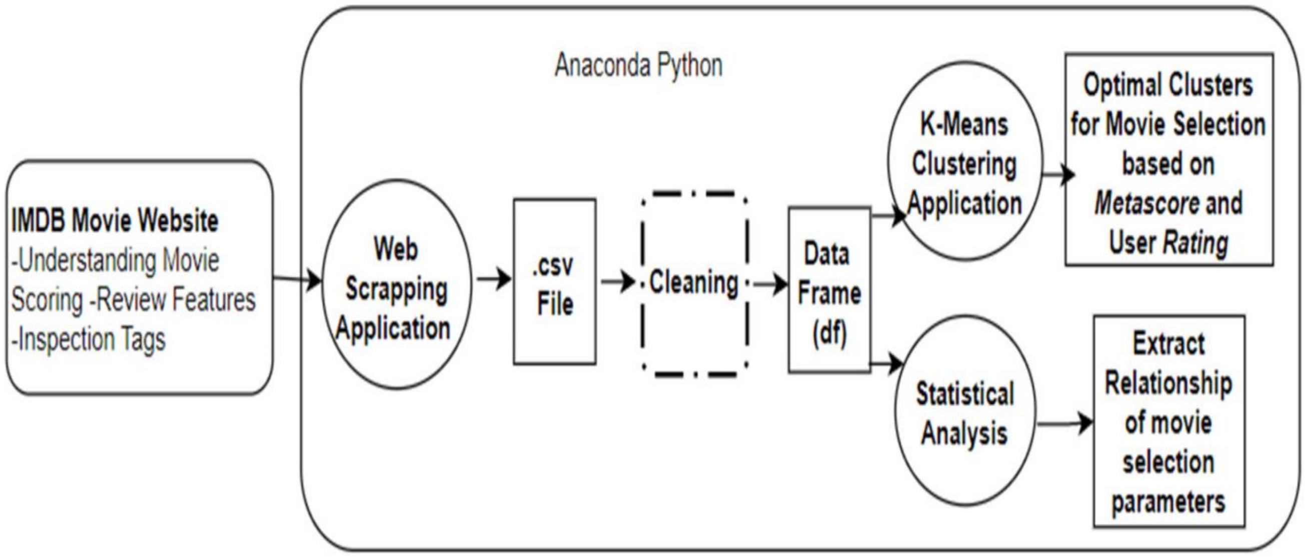
- **Python Data Science Handbook** by Jake VanderPlas
- **Data Science from Scratch:** First Principles with Python by Joel Grus
- **Python for Data Analysis** by Wes McKinney
- **Hands-On Data Analysis** with Pandas by Stefanie Molin
- **Practical Data Science with Pytho**n by Nathan Yau
- **Python Machine Learning** by Sebastian Raschka

II. URL RFERENCES

• NumPy Documentation	: https://numpy.org/doc/stable/
• Pandas Documentation	: https://pandas.pydata.org/pandas-docs/stable/
• Matplotlib Documentation:	https://matplotlib.org/stable/contents.html
• Seaborn Documentation	: https://seaborn.pydata.org/
• MovieLens Dataset	: https://grouplens.org/datasets/movielens/
• Python Programming	: https://docs.python.org/3/
• Scikit-learn Documentation	: https://scikit-learn.org/stable/
• Python for Data Science Handbook	: https://jupyter.org/

APPENDICES

A) DATA FLOW DIAGRAM



Anaconda Python
IMDB Movie Website
-Understanding Movie Scoring -Review Features -Inspection Tags
Web Scrapping Application
.csv File
Cleaning
Data Frame (df)
K-Means Clustering Application
Optimal Clusters for Movie Selection based on *Metascore* and User *Rating*
Statistical Analysis
Extract Relationship of movie selection parameters

B) SAMPLE CODING

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the Dataset

```
# Load MovieLens Dataset
movies = pd.read_csv("movies.csv")  # Replace with your dataset
ratings = pd.read_csv("ratings.csv")  # Replace with your ratings data

# Display the first few rows
print(movies.head())
print(ratings.head())
```

Data Preprocessing

```
# Merging the datasets based on movie IDs
data = pd.merge(ratings, movies, on='movieId')
# Checking for missing data
print(data.isnull().sum())
```

Creating a Movie Rating Matrix

```
# Create a pivot table with 'movieId' as rows, 'userId' as columns, and 'rating' as values
movie_matrix = data.pivot_table(index='movieId', columns='userId', values='rating')
# Filling missing values with zeros
movie_matrix.fillna(0, inplace=True)
# Display the movie rating matrix
print(movie_matrix.head())
```

Calculating Similarity

```
from sklearn.metrics.pairwise import cosine_similarity
# Calculate similarity matrix between movies using cosine similarity
movie_similarity = cosine_similarity(movie_matrix)
# Convert the similarity matrix into a DataFrame
```

```
movie_similarity_df = pd.DataFrame(movie_similarity, index=movie_matrix.index,
columns=movie_matrix.index)
# Display the similarity matrix
print(movie_similarity_df.head())
```

Movie Recommendation Function

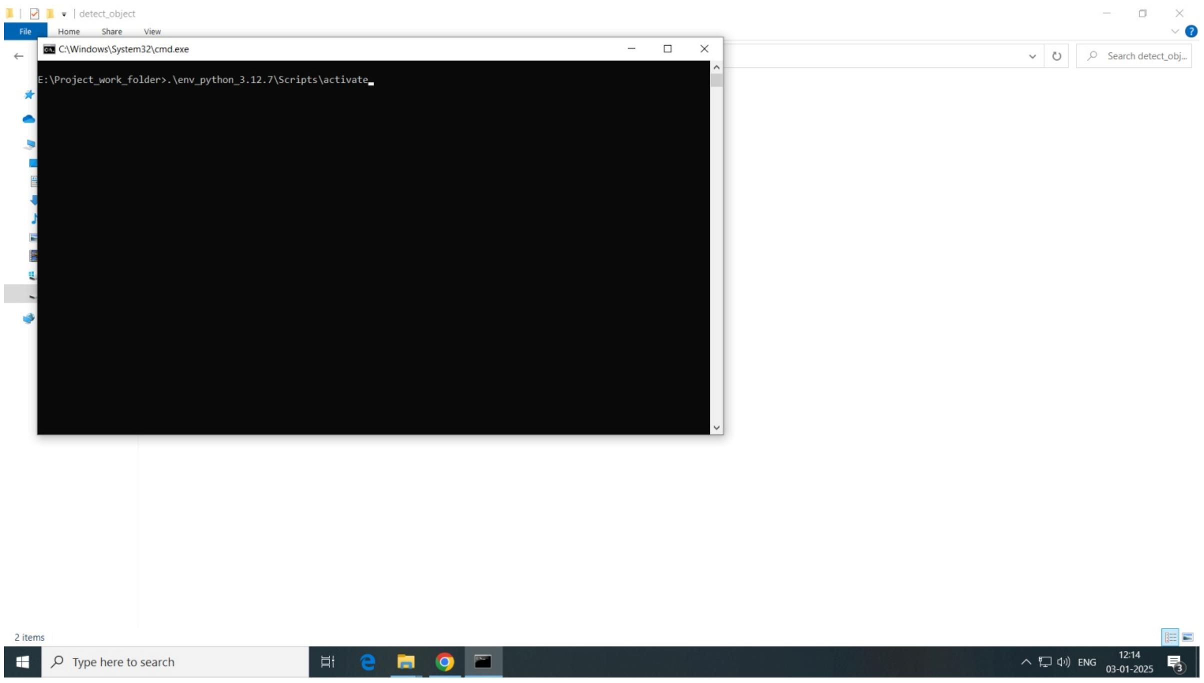
```
def recommend_movies(movie_name, similarity_df, top_n=10):
    # Get the index of the movie
    movie_idx = movies[movies['title'] == movie_name].index[0]
    # Get the similarity scores for the movie
    similar_scores = similarity_df.iloc[movie_idx]
    # Sort the movies based on similarity scores
    similar_movies = similar_scores.sort_values(ascending=False).head(top_n + 1)  # +1 to
exclude the movie itself
    # Get the movie titles
    similar_movie_titles = movies.loc[similar_movies.index, 'title']
    # Display the top N recommended movies
    return similar_movie_titles[1:]
# Example: Recommend movies based on 'Toy Story (1995)'
recommended_movies = recommend_movies('Toy Story (1995)', movie_similarity_df)
print("Recommended Movies:\n", recommended_movies)
```

Data Visualization:

```
# Visualizing the distribution of ratings
plt.figure(figsize=(10,6))
sns.histplot(data['rating'], bins=10, kde=True)
plt.title('Rating Distribution')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
# Visualizing the correlation between movies' ratings
sns.heatmap(movie_similarity_df, cmap='coolwarm', annot=False, fmt='.2f')
plt.title('Movie Similarity Matrix')
plt.show()
```

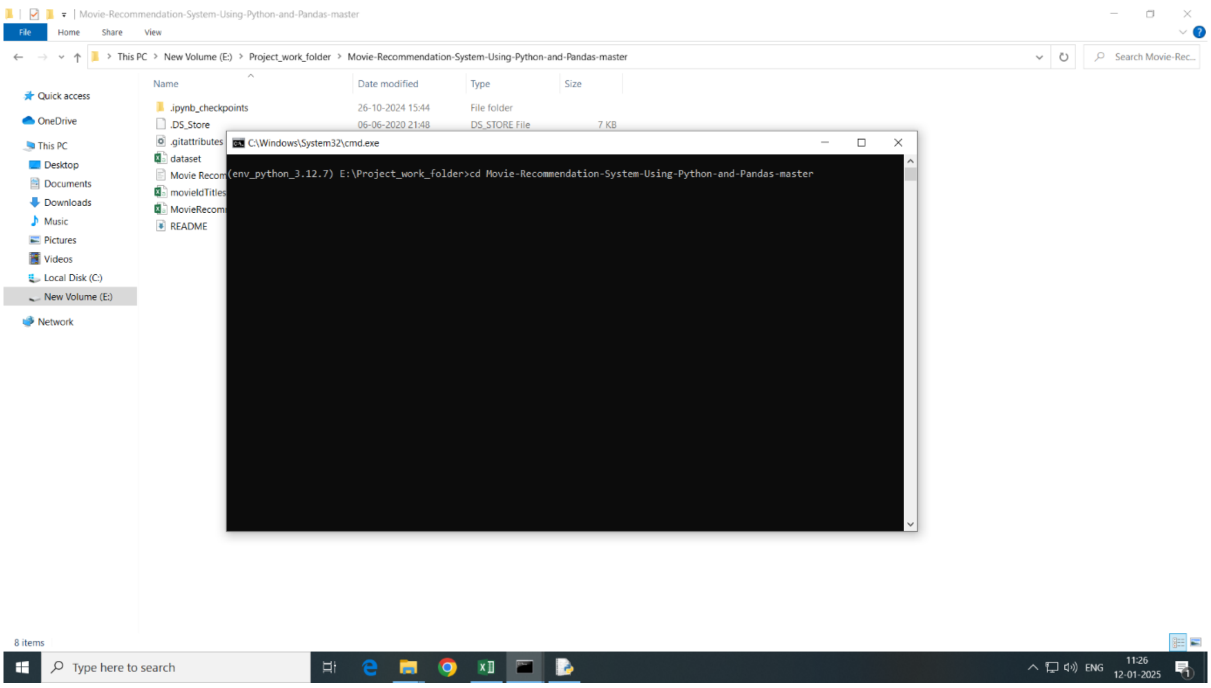
C) SAMPLE INPUT

VIRTUAL ENVIRONMENTAL SETUP



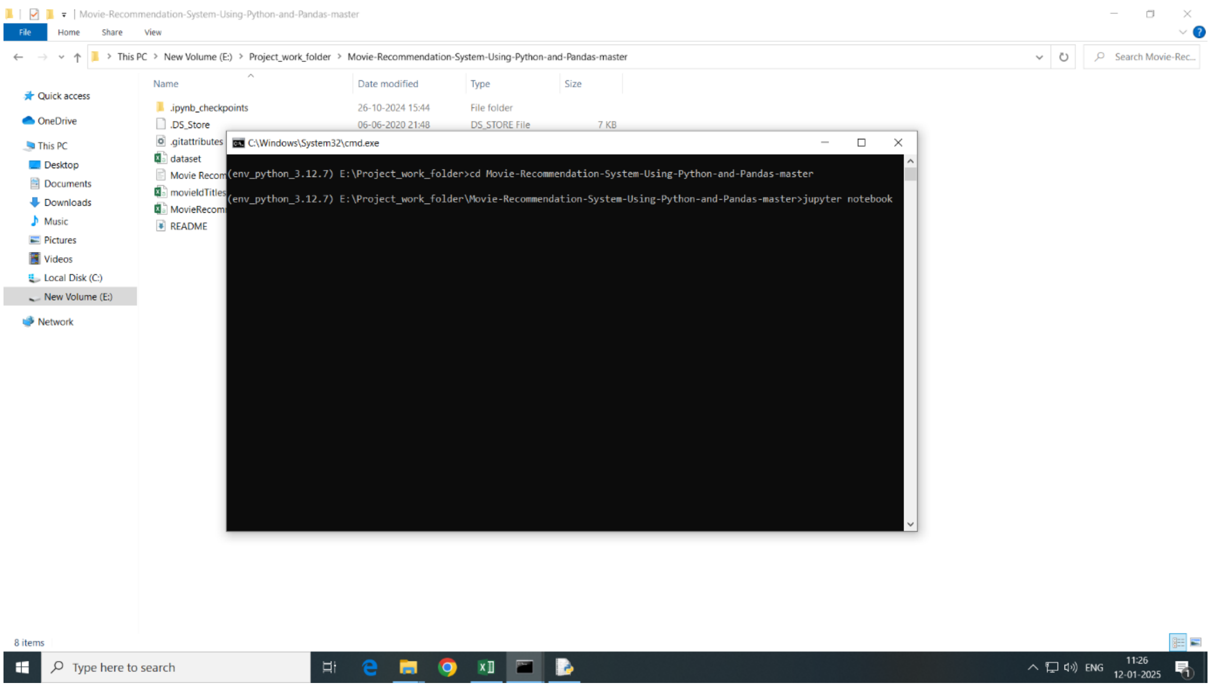
```
E:\Project_work_folder>.\env_python_3.12.7\Scripts\activate
```

PATH REDIRECTION



```
(env_python_3.12.7) E:\Project_work_folder>cd Movie-Recommendation-System-Using-Python-and-Pandas-master
```

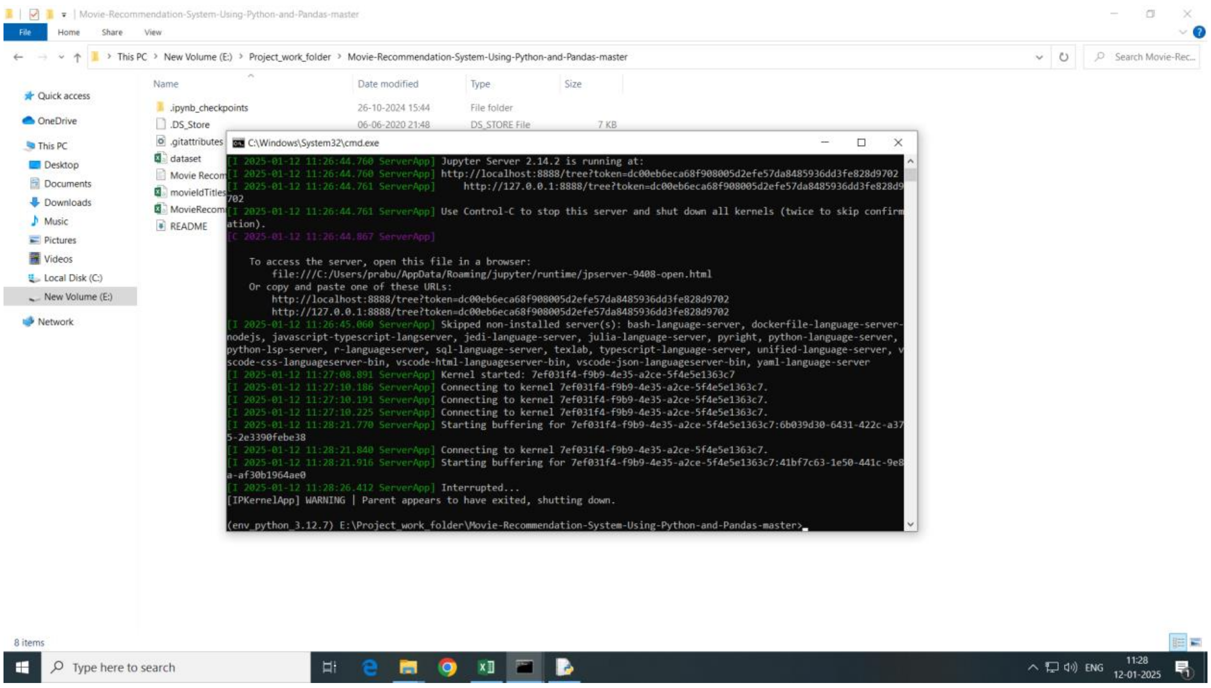
TO RUN PYTHON FILE



C:\Windows\System32\cmd.exe

```
(env_python_3.12.7) E:\Project_work_folder>cd Movie-Recommendation-System-Using-Python-and-Pandas-master

(env_python_3.12.7) E:\Project_work_folder\Movie-Recommendation-System-Using-Python-and-Pandas-master>jupyter notebook
```

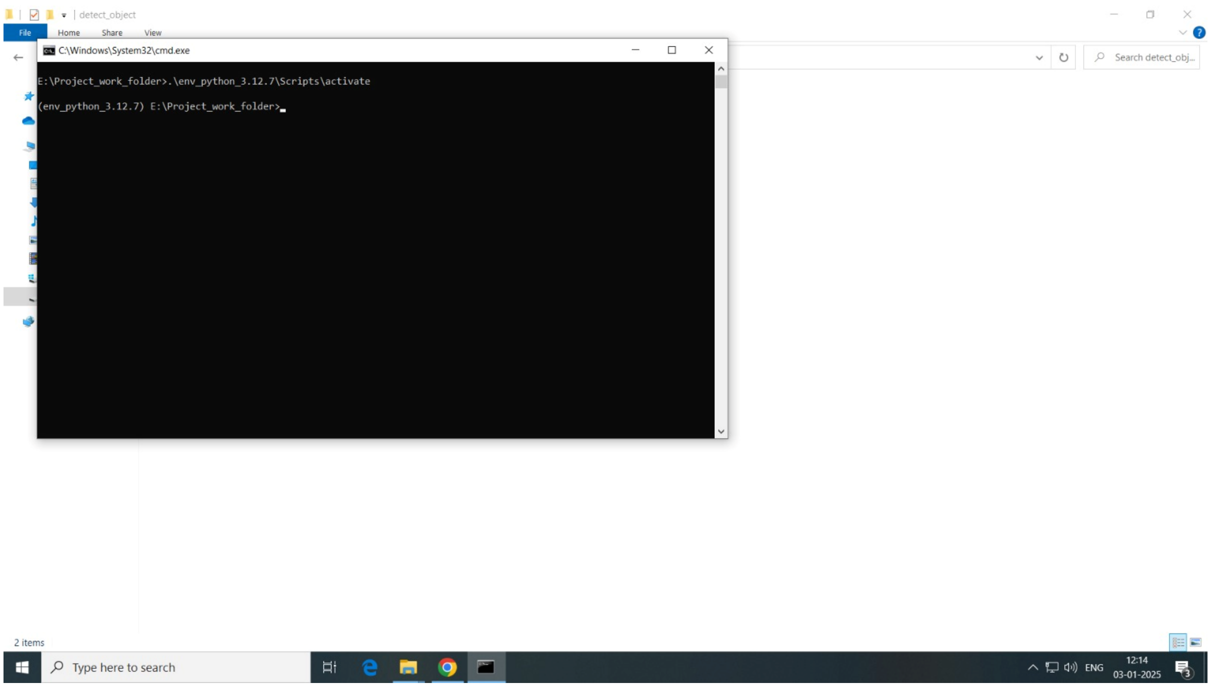
TO STOP THE PROJECT EXECUTION



```
[I 2025-01-12 11:26:44.760 ServerApp] Jupyter Server 2.14.2 is running at:
[I 2025-01-12 11:26:44.760 ServerApp] http://localhost:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9702
[I 2025-01-12 11:26:44.761 ServerApp]     http://127.0.0.1:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9
702
[I 2025-01-12 11:26:44.761 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirm
ation).
[C 2025-01-12 11:26:44.867 ServerApp]

    To access the server, open this file in a browser:
        file:///C:/Users/prabu/AppData/Roaming/jupyter/runtime/jpserver-9408-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9702
        http://127.0.0.1:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9702
[I 2025-01-12 11:26:45.060 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-
nodejs, javascript-typescript-langserver, jedi-language-server, julia-language-server, pyright, python-language-server,
python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-language-server, unified-language-server, v
scode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
[I 2025-01-12 11:27:08.891 ServerApp] Kernel started: 7ef031f4-f9b9-4e35-a2ce-5f4e5e1363c7
[I 2025-01-12 11:27:10.186 ServerApp] Connecting to kernel 7ef031f4-f9b9-4e35-a2ce-5f4e5e1363c7.
[I 2025-01-12 11:27:10.191 ServerApp] Connecting to kernel 7ef031f4-f9b9-4e35-a2ce-5f4e5e1363c7.
[I 2025-01-12 11:27:10.225 ServerApp] Connecting to kernel 7ef031f4-f9b9-4e35-a2ce-5f4e5e1363c7.
[I 2025-01-12 11:28:21.770 ServerApp] Starting buffering for 7ef031f4-f9b9-4e35-a2ce-5f4e5e1363c7:6b039d30-6431-422c-a37
5-2e3390febe38
[I 2025-01-12 11:28:21.840 ServerApp] Connecting to kernel 7ef031f4-f9b9-4e35-a2ce-5f4e5e1363c7.
[I 2025-01-12 11:28:21.916 ServerApp] Starting buffering for 7ef031f4-f9b9-4e35-a2ce-5f4e5e1363c7:41bf7c63-1e50-441c-9e8
a-af30b1964ae0
[I 2025-01-12 11:28:26.412 ServerApp] Interrupted...
[IPKernelApp] WARNING | Parent appears to have exited, shutting down.

(env_python_3.12.7) E:\Project_work_folder\Movie-Recommendation-System-Using-Python-and-Pandas-master>
```

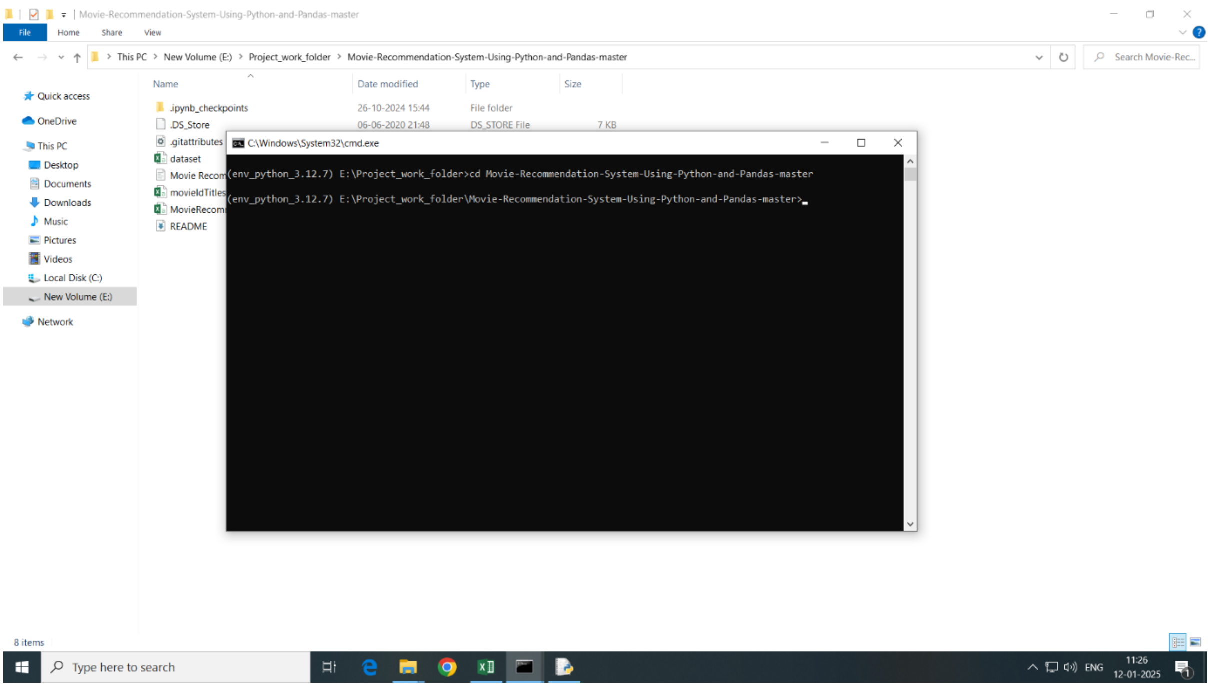
D) SAMPLE OUTPUT

TO ACTIVATE VIRTUAL ENVIRONMENT



```
E:\Project_work_folder>.\env_python_3.12.7\Scripts\activate

(env_python_3.12.7) E:\Project_work_folder>
```

REDIRECT PATH

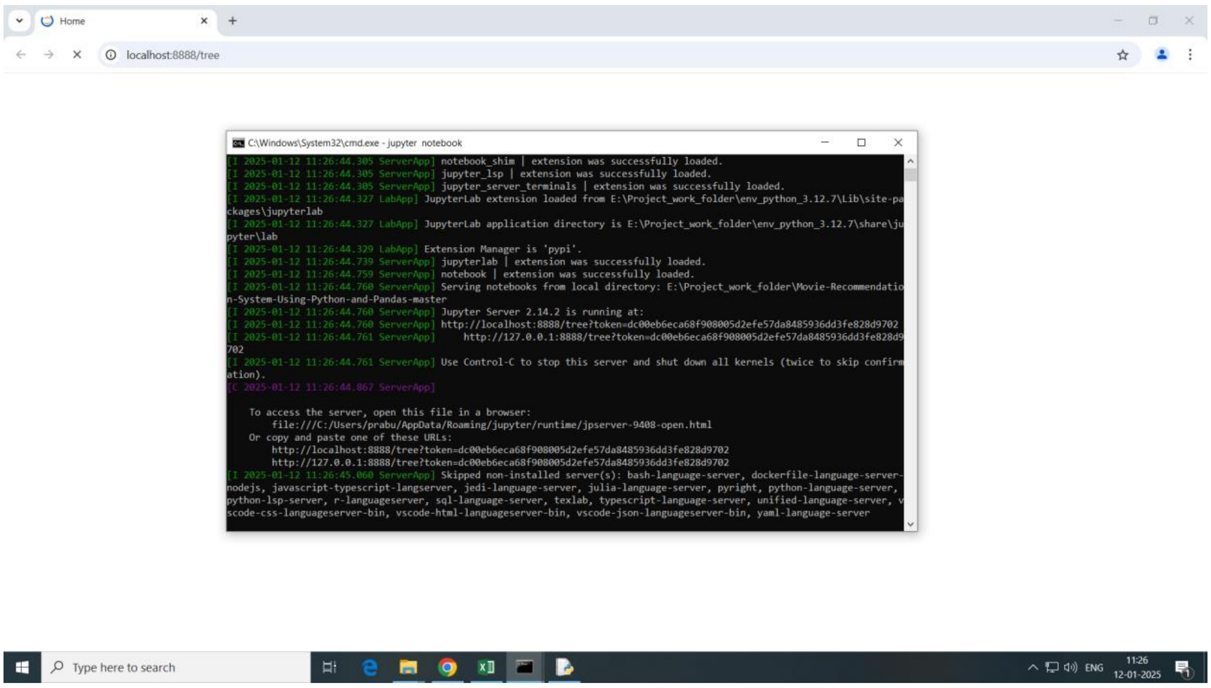


Movie-Recommendation-System-Using-Python-and-Pandas-master
This PC > New Volume (E:) > Project_work_folder > Movie-Recommendation-System-Using-Python-and-Pandas-master
C:\Windows\System32\cmd.exe
Type here to search

Name	Date modified	Type	Size
.ipynb_checkpoints	26-10-2024 15:44	File folder	
.DS_Store	06-06-2020 21:48	DS_STORE File	7 KB
.gitattributes			
dataset			
Movie Recom			
movieIdTitles			
MovieRecom			
README			

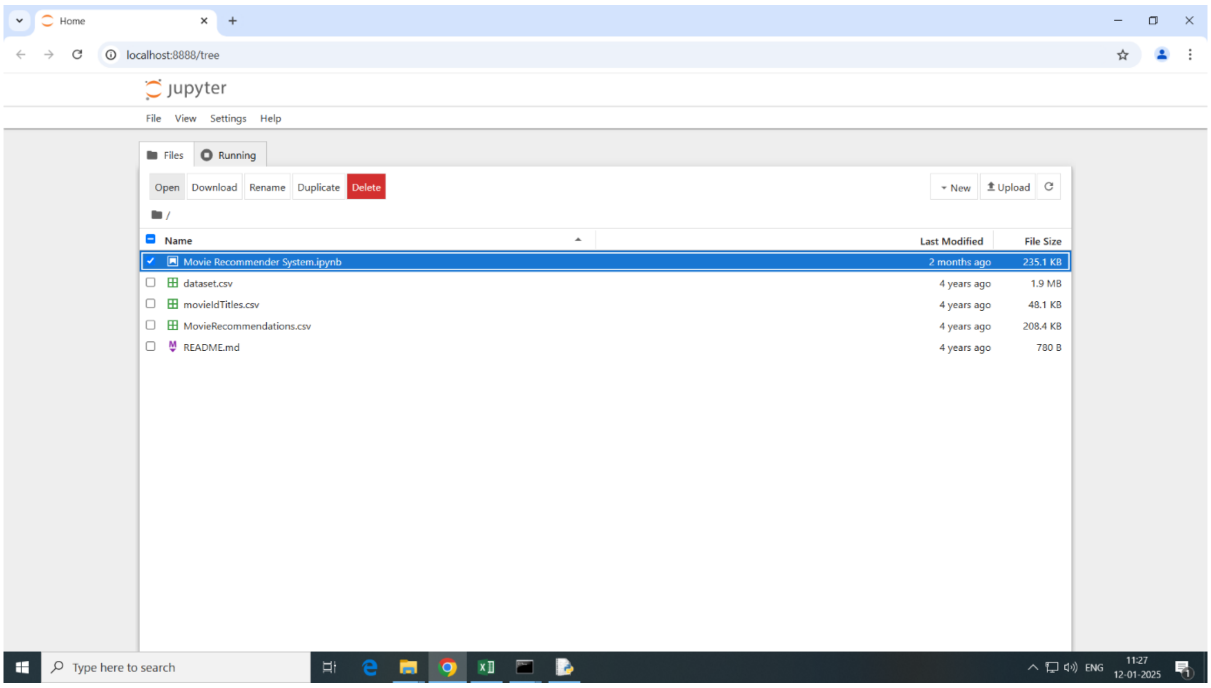
```
(env_python_3.12.7) E:\Project_work_folder>cd Movie-Recommendation-System-Using-Python-and-Pandas-master

(env_python_3.12.7) E:\Project_work_folder\Movie-Recommendation-System-Using-Python-and-Pandas-master>_
```

GET THE FILE URL



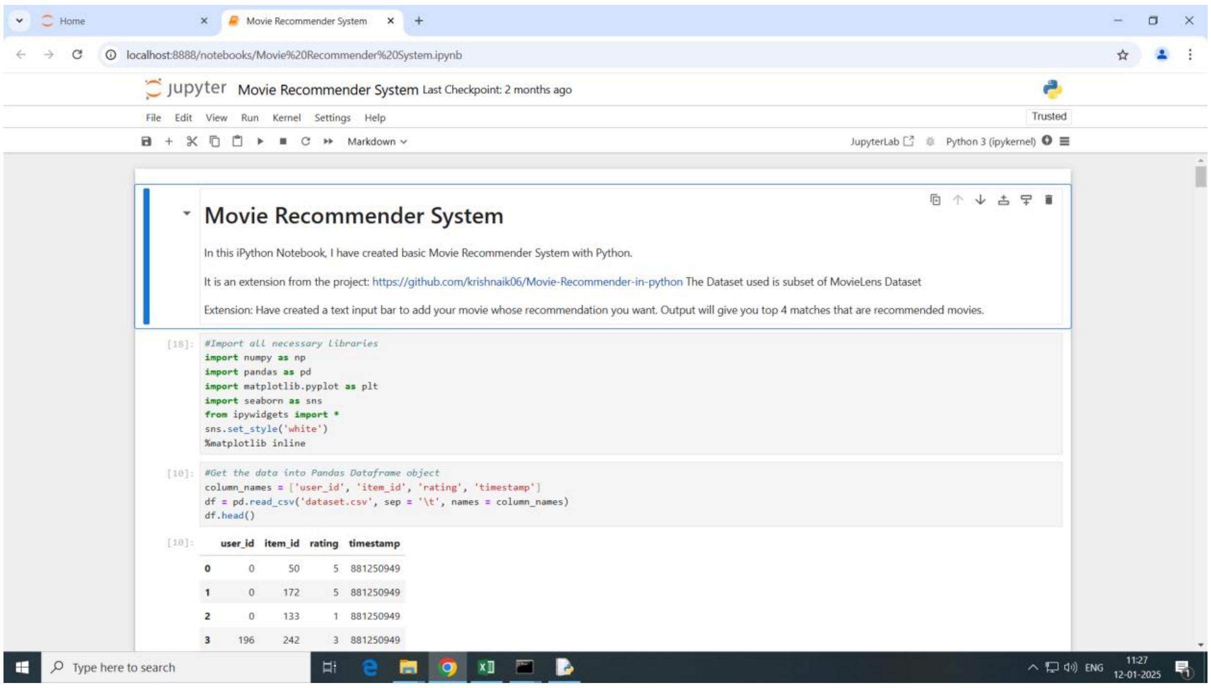
```
[I 2025-01-12 11:26:44.305 ServerApp] notebook_shim | extension was successfully loaded.
[I 2025-01-12 11:26:44.305 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2025-01-12 11:26:44.305 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2025-01-12 11:26:44.327 LabApp] JupyterLab extension loaded from E:\Project_work_folder\env_python_3.12.7\Lib\site-pa
ckages\jupyterlab
[I 2025-01-12 11:26:44.327 LabApp] JupyterLab application directory is E:\Project_work_folder\env_python_3.12.7\share\ju
pyter\lab
[I 2025-01-12 11:26:44.329 LabApp] Extension Manager is 'pypi'.
[I 2025-01-12 11:26:44.739 ServerApp] jupyterlab | extension was successfully loaded.
[I 2025-01-12 11:26:44.759 ServerApp] notebook | extension was successfully loaded.
[I 2025-01-12 11:26:44.760 ServerApp] Serving notebooks from local directory: E:\Project_work_folder\Movie-Recommendatio
n-System-Using-Python-and-Pandas-master
[I 2025-01-12 11:26:44.760 ServerApp] Jupyter Server 2.14.2 is running at:
[I 2025-01-12 11:26:44.760 ServerApp] http://localhost:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9702
[I 2025-01-12 11:26:44.761 ServerApp]     http://127.0.0.1:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9
702
[I 2025-01-12 11:26:44.761 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirm
ation).
[C 2025-01-12 11:26:44.867 ServerApp]

    To access the server, open this file in a browser:
        file:///C:/Users/prabu/AppData/Roaming/jupyter/runtime/jpserver-9408-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9702
        http://127.0.0.1:8888/tree?token=dc00eb6eca68f908005d2efe57da8485936dd3fe828d9702
[I 2025-01-12 11:26:45.060 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-
nodejs, javascript-typescript-langserver, jedi-language-server, julia-language-server, pyright, python-language-server,
python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-language-server, unified-language-server, v
scode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
```

OPEN THE JUPYTER NOTEBOOK FILE



Name	Last Modified	File Size
Movie Recommender System.ipynb	2 months ago	235.1 KB
dataset.csv	4 years ago	1.9 MB
movieIdTitles.csv	4 years ago	48.1 KB
MovieRecommendations.csv	4 years ago	208.4 KB
README.md	4 years ago	780 B

ITEM LIST BASED ON RATING



Movie Recommender System

In this iPython Notebook, I have created basic Movie Recommender System with Python.

It is an extension from the project: https://github.com/krishnaik06/Movie-Recommender-in-python The Dataset used is subset of MovieLens Dataset

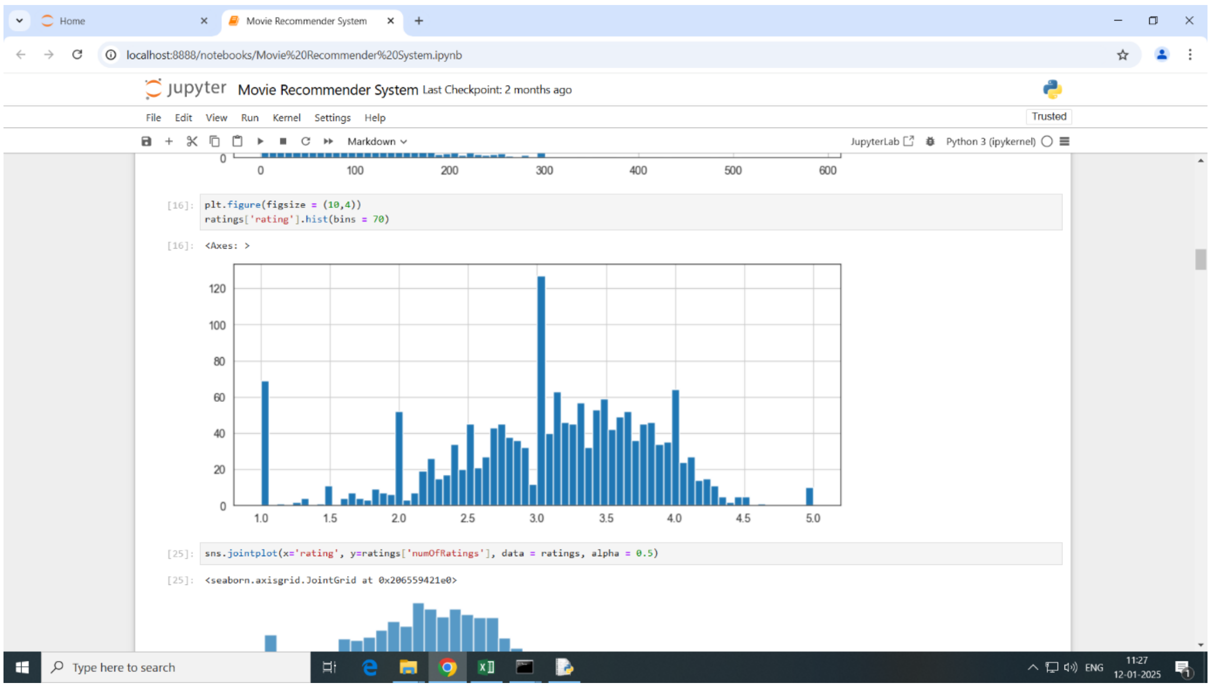
Extension: Have created a text input bar to add your movie whose recommendation you want. Output will give you top 4 matches that are recommended movies.

```
[18]: #Import all necessary Libraries
      import numpy as np
      import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      from ipywidgets import *
      sns.set_style('white')
      %matplotlib inline
```

```
[10]: #Get the data into Pandas Dataframe object
      column_names = ['user_id', 'item_id', 'rating', 'timestamp']
      df = pd.read_csv('dataset.csv', sep = '\t', names = column_names)
      df.head()
```

```
[10]:    user_id  item_id  rating  timestamp
      0        0       50       5  881250949
      1        0      172       5  881250949
      2        0      133       1  881250949
      3      196      242       3  881250949
```

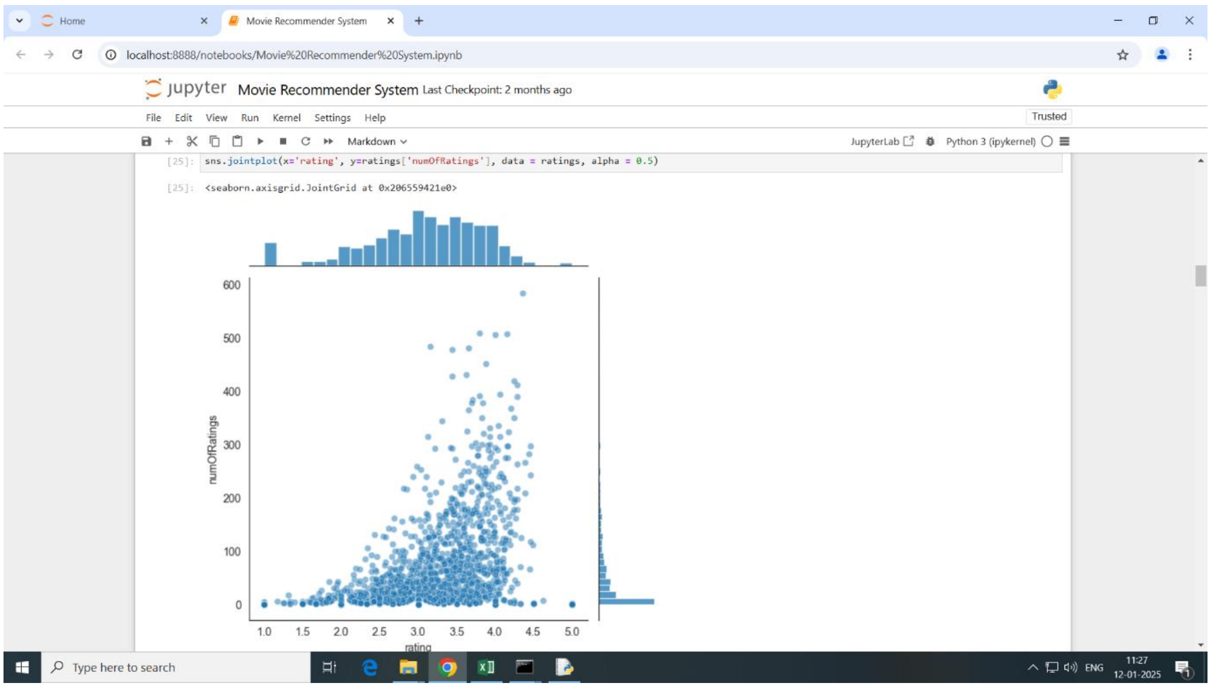
HISTOGRAM


Home
Movie Recommender System
localhost:8888/notebooks/Movie%20Recommender%20System.ipynb
jupyter Movie Recommender System Last Checkpoint: 2 months ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)
Type here to search
ENG
11:27
12-01-2025

```
[16]: plt.figure(figsize = (10,4))
      ratings['rating'].hist(bins = 70)

[16]: <Axes: >
```

```
[25]: sns.jointplot(x='rating', y=ratings['numOfRatings'], data = ratings, alpha = 0.5)

[25]: <seaborn.axisgrid.JointGrid at 0x206559421e0>
```

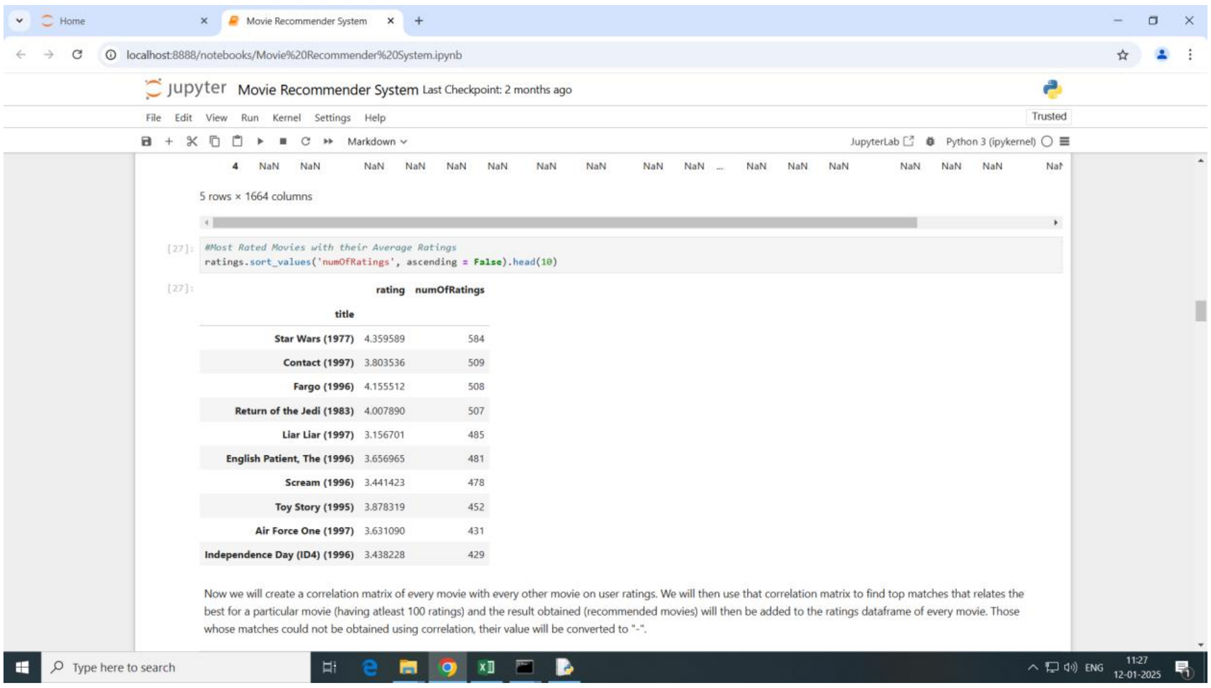
PLOT GRAPH BASED ON NUMBER OF RATINGS



```
[25]: sns.jointplot(x='rating', y=ratings['numOfRatings'], data = ratings, alpha = 0.5)
```

```
[25]: <seaborn.axisgrid.JointGrid at 0x206559421e0>
```

MOST RATED MOVIES LIST



localhost:8888/notebooks/Movie%20Recommender%20System.ipynb

Movie Recommender System Last Checkpoint: 2 months ago

```
4  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN  NaN  NaN  NaN  Nal
```

5 rows × 1664 columns

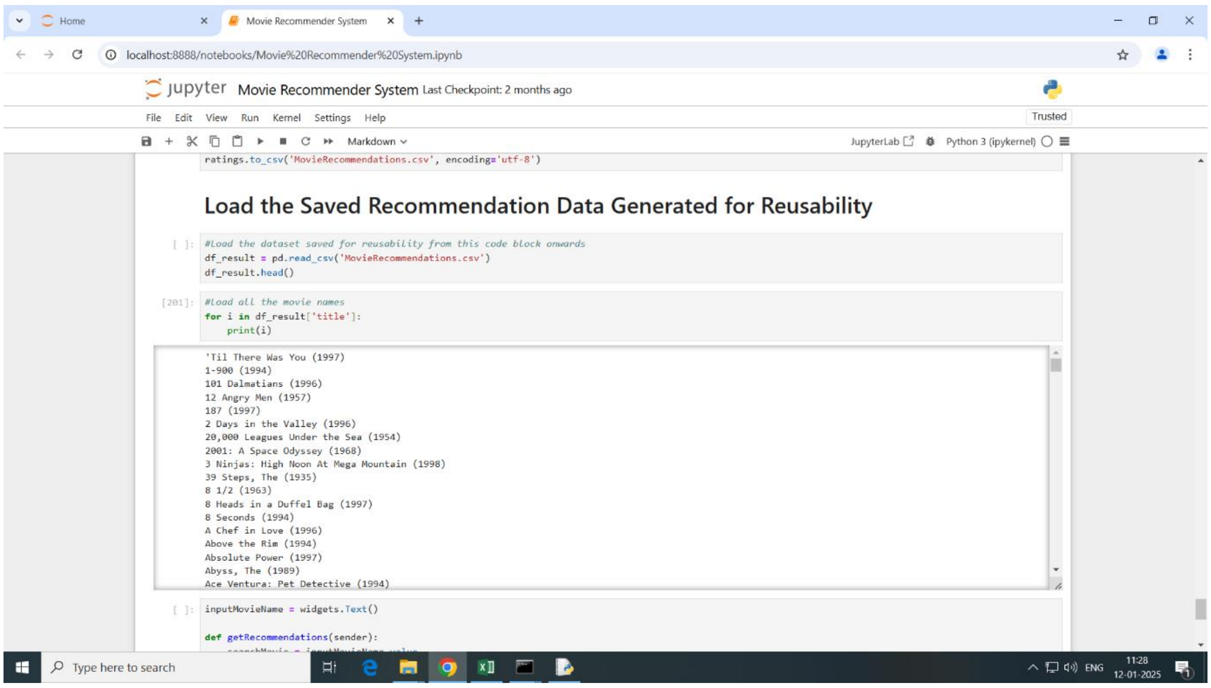
```
[27]: #Most Rated Movies with their Average Ratings
      ratings.sort_values('numOfRatings', ascending = False).head(10)
```

[27]:

title	rating	numOfRatings
Star Wars (1977)	4.359589	584
Contact (1997)	3.803536	509
Fargo (1996)	4.155512	508
Return of the Jedi (1983)	4.007890	507
Liar Liar (1997)	3.156701	485
English Patient, The (1996)	3.656965	481
Scream (1996)	3.441423	478
Toy Story (1995)	3.878319	452
Air Force One (1997)	3.631090	431
Independence Day (ID4) (1996)	3.438228	429

Now we will create a correlation matrix of every movie with every other movie on user ratings. We will then use that correlation matrix to find top matches that relates the best for a particular movie (having atleast 100 ratings) and the result obtained (recommended movies) will then be added to the ratings dataframe of every movie. Those whose matches could not be obtained using correlation, their value will be converted to "-".

SAVED RECOMMENDATION DATA


Home
Movie Recommender System
localhost:8888/notebooks/Movie%20Recommender%20System.ipynb
jupyter Movie Recommender System Last Checkpoint: 2 months ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)
Type here to search
ENG
11:28
12-01-2025

```
ratings.to_csv('MovieRecommendations.csv', encoding='utf-8')
```

Load the Saved Recommendation Data Generated for Reusability

```
[ ]: #Load the dataset saved for reusability from this code block onwards
     df_result = pd.read_csv('MovieRecommendations.csv')
     df_result.head()
```

```
[201]: #Load all the movie names
       for i in df_result['title']:
           print(i)
```

```
'Til There Was You (1997)
1-900 (1994)
101 Dalmatians (1996)
12 Angry Men (1957)
187 (1997)
2 Days in the Valley (1996)
20,000 Leagues Under the Sea (1954)
2001: A Space Odyssey (1968)
3 Ninjas: High Noon At Mega Mountain (1998)
39 Steps, The (1935)
8 1/2 (1963)
8 Heads in a Duffel Bag (1997)
8 Seconds (1994)
A Chef in Love (1996)
Above the Rim (1994)
Absolute Power (1997)
Abyss, The (1989)
Ace Ventura: Pet Detective (1994)
```

```
[ ]: inputMovieName = widgets.Text()

     def getRecommendations(sender):
```