كليـة الهـنـدسـة
Faculty of Engineering

جامعة حلوان
Helwan University
1975

2024

Machine Learning Project

SUPERVISOR: *Dr. Noor El-Deen Magdy*

Movie Collaborative Recommendation System :

Faculty of Engineering,

1st Year Computer and Systems Department

Table of Contents

1-Participants.. **3**

2-Overview.. **2**

2.1-Main Concept.. 2

3-Installation Steps.. **3**

4- Usage 4-5

5- Project Structure and Functionality 8-13

6- System Workflow Overview 14

7- Technologies Used 14

Participants

1-احمد صلاح فاروق مراد

2-احمد ممدوح صادق محمد ابراهيم

3-سيف الدين خالد نظمى لطفى محمد

4-يوسف محمد محرز محمد عبدالفتاح

5-محمد عاصم احمد محمد ابراهيم

Overview

- This project implements a movie recommendation system that leverages collaborative filtering to suggest movies based on a user's previous ratings and preferences. It predicts movie recommendations by analyzing patterns from a large dataset of user interactions with various films.
- Main Concept:

The project is divided into two key components:

1. Data Preprocessing and Model Training: In this phase, the raw movie and user data is cleaned, preprocessed, and used to train a machine learning model. The collaborative filtering model is built to learn user preferences and identify patterns in movie ratings.
2. User Interface (UI): The UI allows users to interact with the system by entering the name of a movie they like. The system then outputs a list of 10 recommended movies tailored to the user's taste based on the trained model.

This combination of backend machine learning and frontend interaction ensures a seamless experience for users seeking personalized movie suggestions.

Installation Steps

1. Prerequisites:
   - Python 3.7 or higher
   - Libraries listed in requirements.txt

2. Clone the repository:

```
git clone https://github.com/AhmedElnajjar/Movie-recommendation-system-main.git
cd Movie-recommendation-system-main
```

2. Or Simply download the zipped folder then navigate to it in the terminal:

3. Install dependencies:

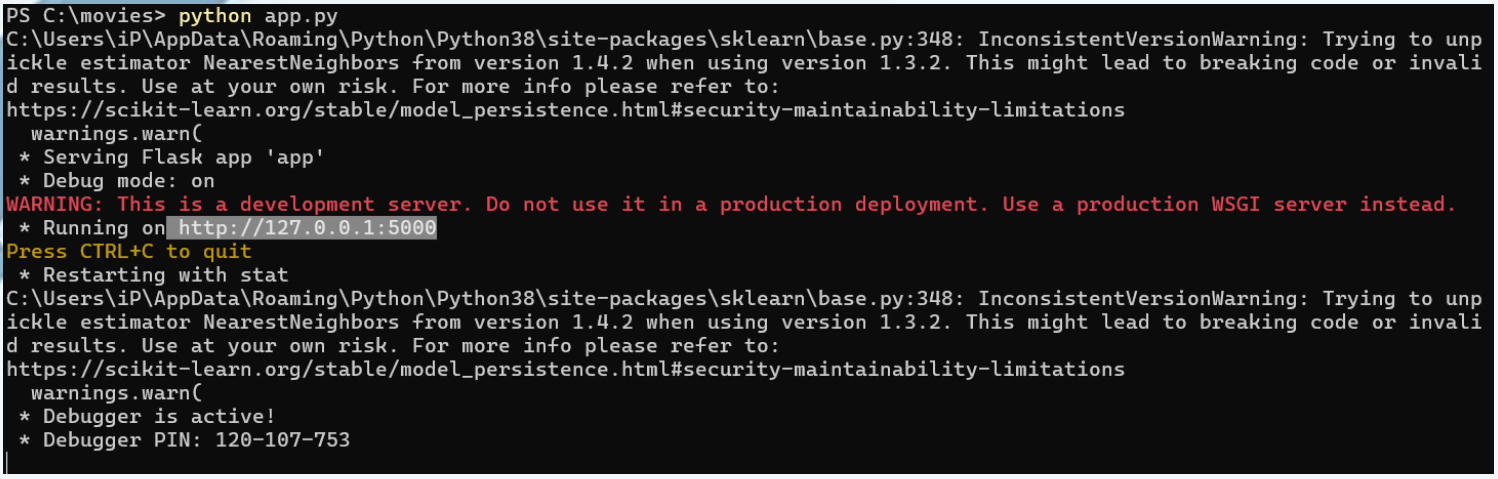
```
pip install -r requirements.txt
```

Usage

1. Run the app.py:

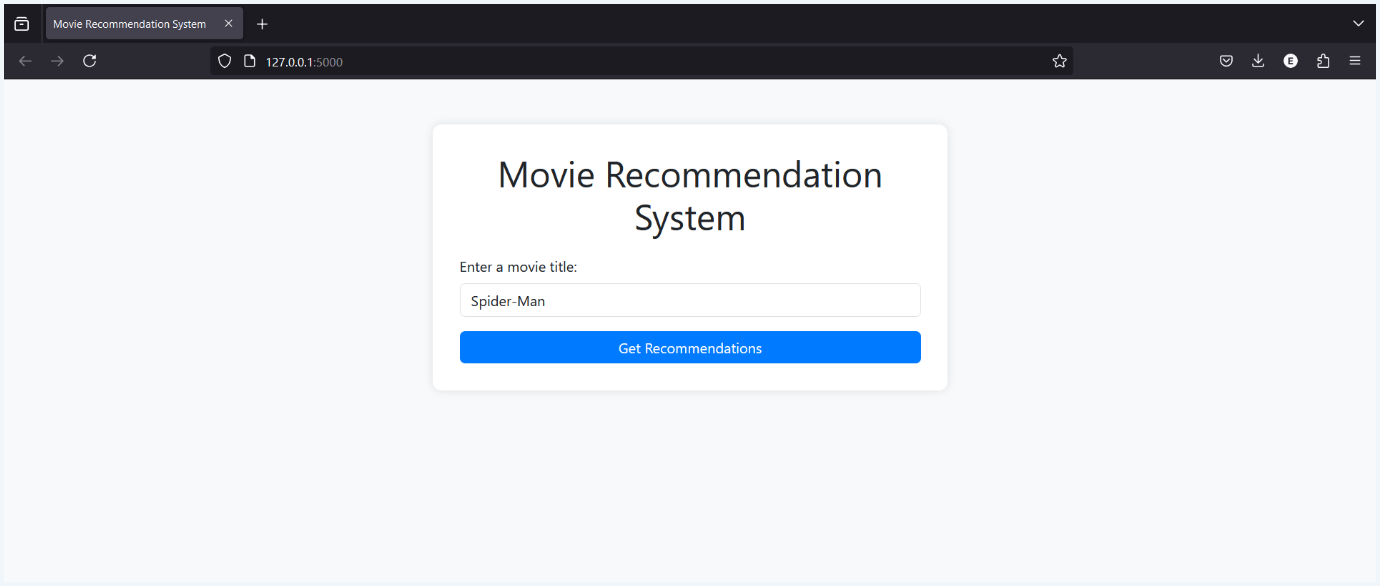
```
python app.py
```

2. Copy the highlighted local host URL:



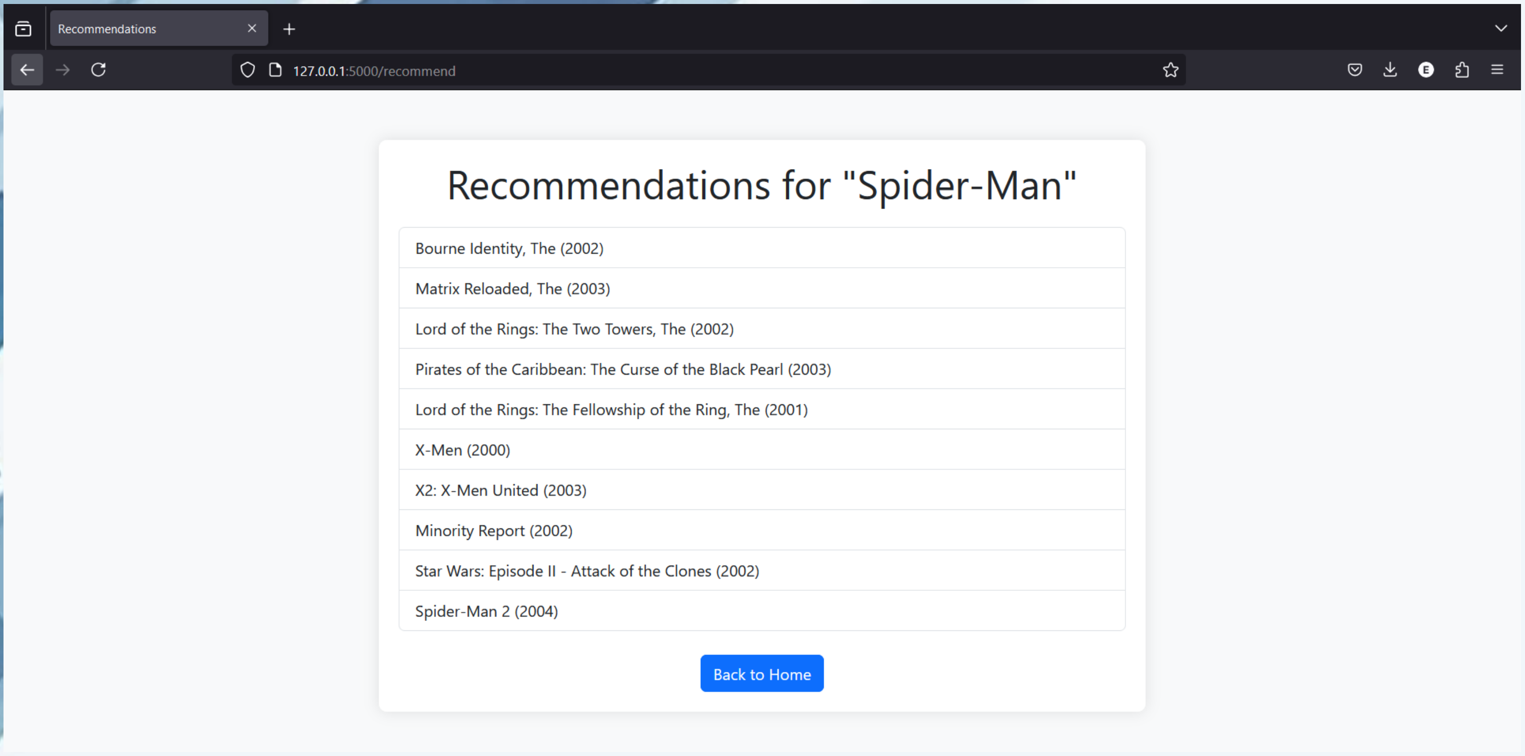
```
PS C:\movies> python app.py
C:\Users\iP\AppData\Roaming\Python\Python38\site-packages\sklearn\base.py:348: InconsistentVersionWarning: Trying to unp
ickle estimator NearestNeighbors from version 1.4.2 when using version 1.3.2. This might lead to breaking code or invali
d results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
C:\Users\iP\AppData\Roaming\Python\Python38\site-packages\sklearn\base.py:348: InconsistentVersionWarning: Trying to unp
ickle estimator NearestNeighbors from version 1.4.2 when using version 1.3.2. This might lead to breaking code or invali
d results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Debugger is active!
 * Debugger PIN: 120-107-753
```

3. Paste it to your browser to use the recommendation app:



4. Write the name of the movie you want to get recommendation to then press Get Recommendations

5. Here are your Recommendations



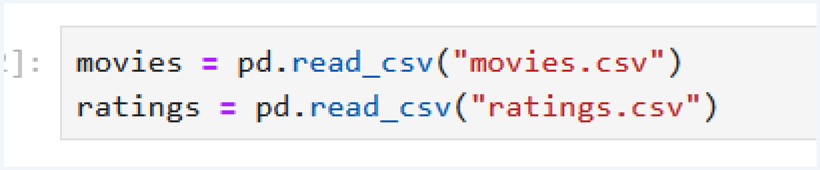
Recommendations for "Spider-Man"

- Bourne Identity, The (2002)
- Matrix Reloaded, The (2003)
- Lord of the Rings: The Two Towers, The (2002)
- Pirates of the Caribbean: The Curse of the Black Pearl (2003)
- Lord of the Rings: The Fellowship of the Ring, The (2001)
- X-Men (2000)
- X2: X-Men United (2003)
- Minority Report (2002)
- Star Wars: Episode II - Attack of the Clones (2002)
- Spider-Man 2 (2004)

Project Structure and Functionality

1- **Movie Recommendation System.ipynb:** Jupyter notebook file for system development and preprocessing data and model training

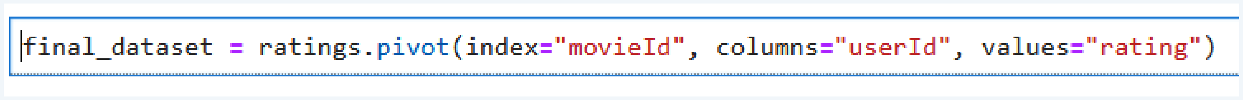
- Data Loading:
  a) movies.csv: Contains the movie details such as movie ID and title.
  b) ratings.csv: Contains user ratings of movies, where each rating is associated with a user ID and a movie ID.



```
movies = pd.read_csv("movies.csv")
ratings = pd.read_csv("ratings.csv")
```

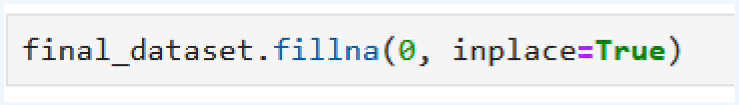
- Data Preparation:

a) Creating a Pivot Table: where the rows represent movies, the columns represent users, and the values are the ratings.



```
final_dataset = ratings.pivot(index="movieId", columns="userId", values="rating")
```

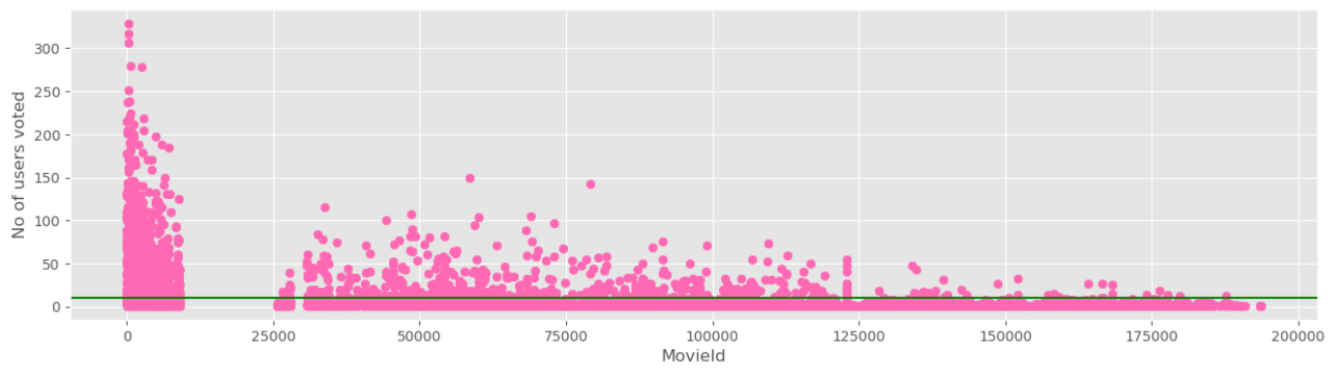
b) Handling Missing Values: The missing ratings (NaN values) are filled with zeros.



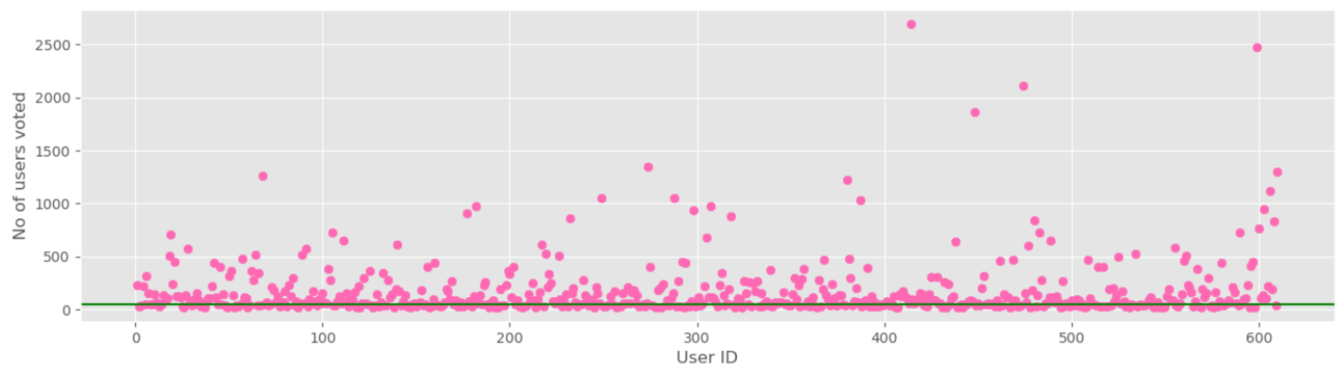
```
final_dataset.fillna(0, inplace=True)
```

c) Visualizations : scatter plots are generated to visualize the number of ratings per movie and per user. This helps in identifying how many users are active and how many movies have sufficient ratings

```python
import matplotlib.pyplot as plt
plt.style.use("ggplot")
fig,axes = plt.subplots(1,1, figsize=(16,4))
plt.scatter(no_user_voted.index, no_user_voted, color="hotpink")
plt.axhline(y=10, color='green')
plt.xlabel("MovieId")
plt.ylabel("No of users voted")
plt.show()
```



```python
import matplotlib.pyplot as plt
plt.style.use("ggplot")
fig,axes = plt.subplots(1,1, figsize=(16,4))
plt.scatter(no_movies_voted.index, no_movies_voted, color="hotpink")
plt.axhline(y=50, color='green')
plt.xlabel("User ID")
plt.ylabel("No of users voted")
plt.show()
```



d) Filtering Low-Activity Data: removing movies that have less than 10 ratings and users who have rated fewer than 50 movies.

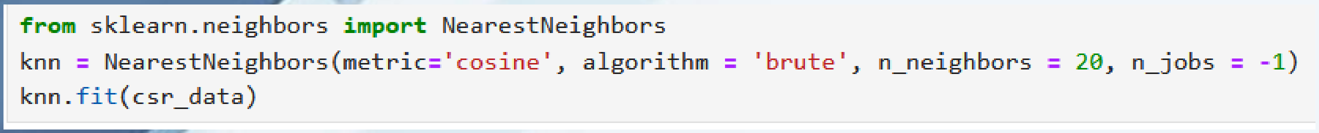
```python
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index, :]
```

```python
final_dataset = final_dataset.loc[: , no_movies_voted[no_movies_voted>50].index]
```

e) Converting to Sparse Matrix : To optimize memory usage, the dense matrix of movie ratings is converted into a sparse matrix.

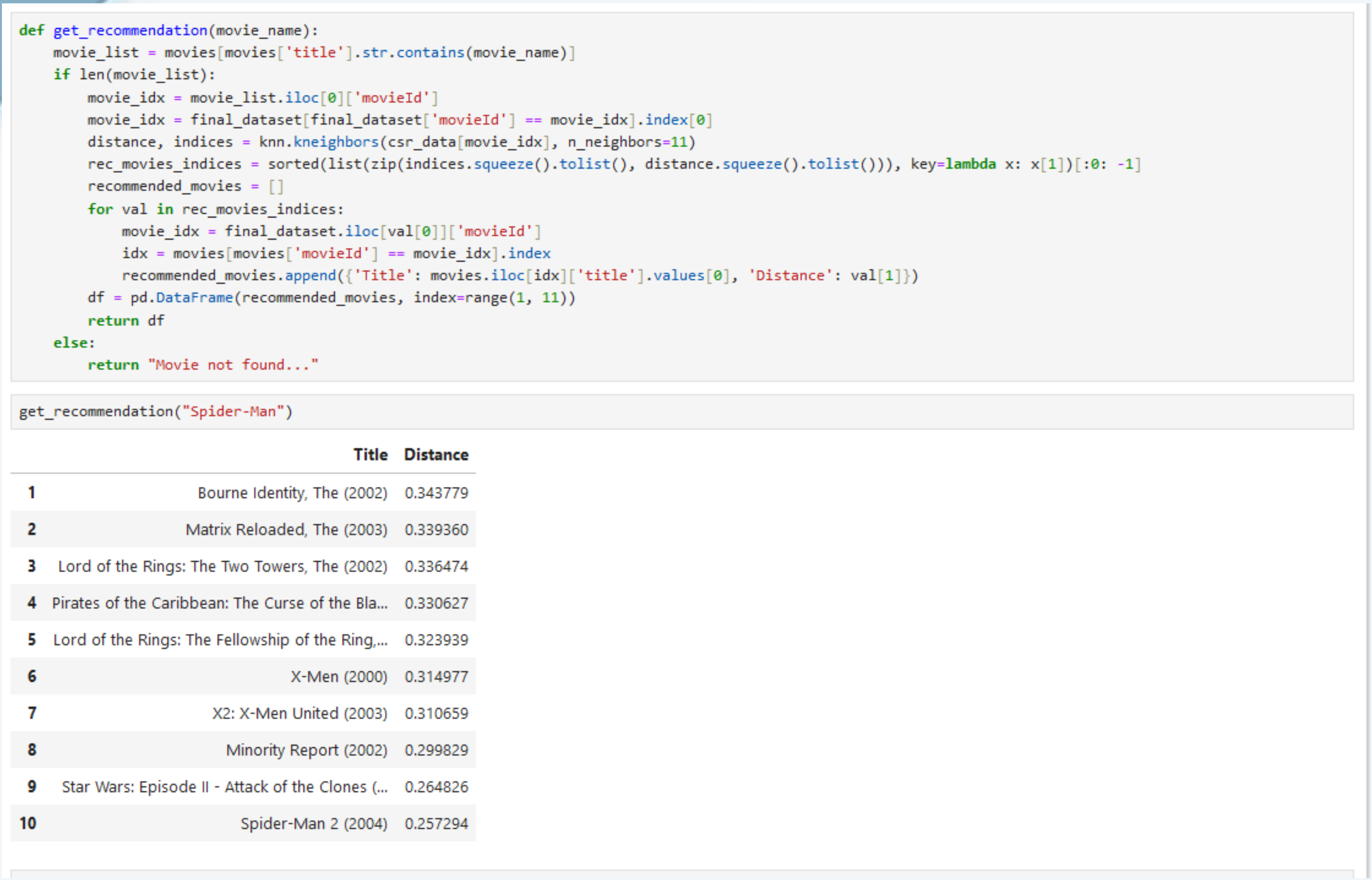
```python
from scipy.sparse import csr_matrix
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
```

F) K-Nearest Neighbors (KNN) Algorithm : The KNN model is built using the NearestNeighbors class from sklearn, where the cosine similarity metric is used to find similar movies.



```
from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(metric='cosine', algorithm = 'brute', n_neighbors = 20, n_jobs = -1)
knn.fit(csr_data)
```

g) Movie Recommendation Function: function takes a movie name as input, finds its ID, and returns a list of 10 recommended movies based on the nearest neighbors.

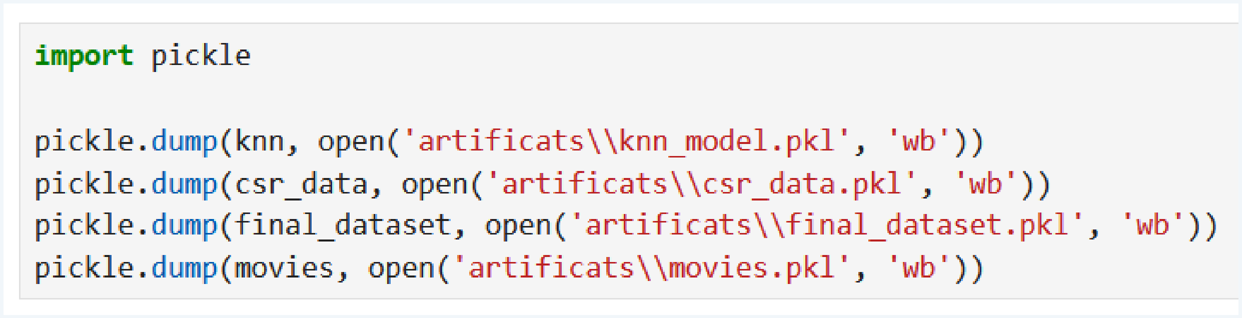


```
def get_recommendation(movie_name):
    movie_list = movies[movies['title'].str.contains(movie_name)]
    if len(movie_list):
        movie_idx = movie_list.iloc[0]['movieId']
        movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]
        distance, indices = knn.kneighbors(csr_data[movie_idx], n_neighbors=11)
        rec_movies_indices = sorted(list(zip(indices.squeeze().tolist(), distance.squeeze().tolist())), key=lambda x: x[1])[:0: -1]
        recommended_movies = []
        for val in rec_movies_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            recommended_movies.append({'Title': movies.iloc[idx]['title'].values[0], 'Distance': val[1]})
        df = pd.DataFrame(recommended_movies, index=range(1, 11))
        return df
    else:
        return "Movie not found..."
```

```
get_recommendation("Spider-Man")
```

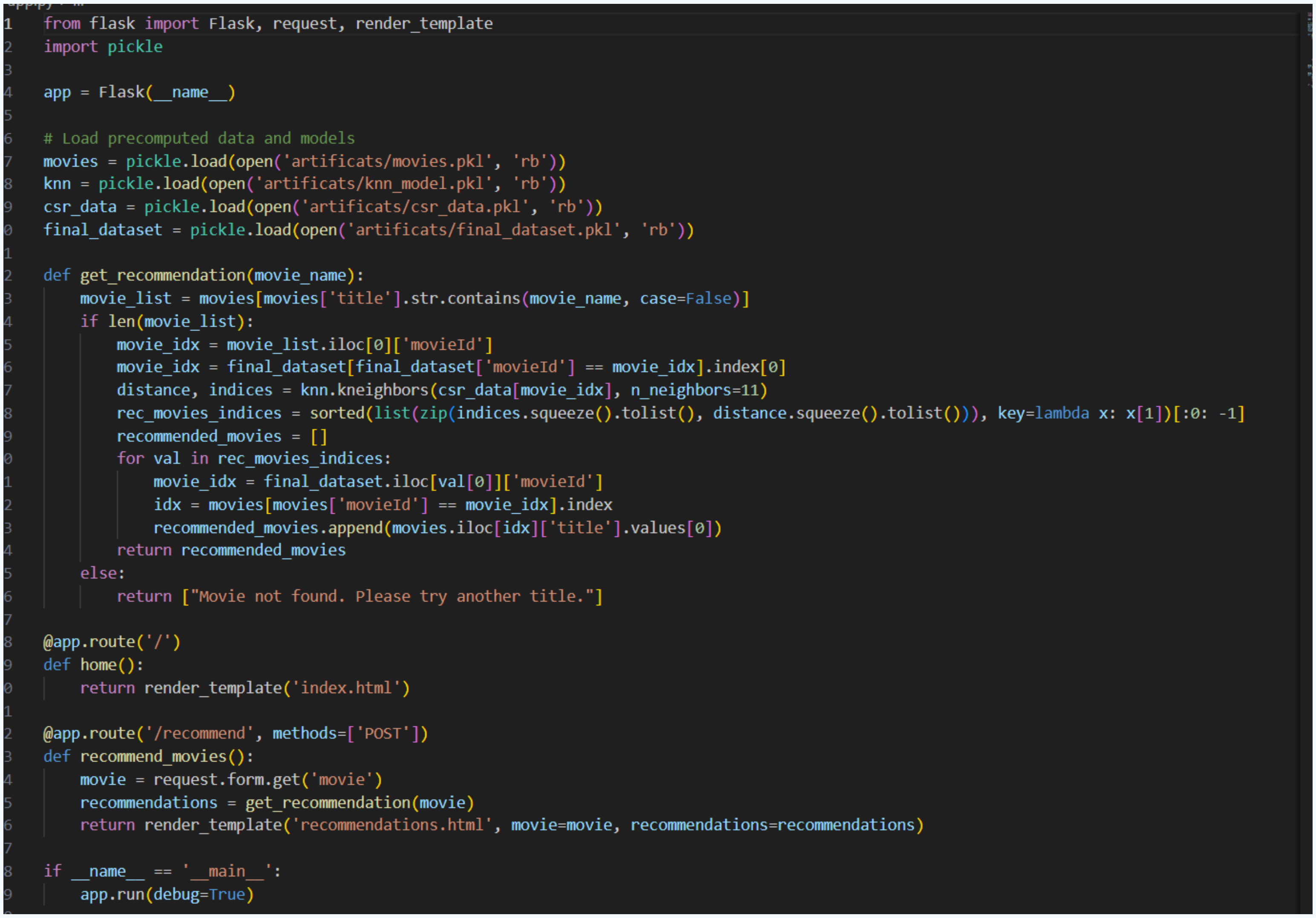
	Title	Distance
1	Bourne Identity, The (2002)	0.343779
2	Matrix Reloaded, The (2003)	0.339360
3	Lord of the Rings: The Two Towers, The (2002)	0.336474
4	Pirates of the Caribbean: The Curse of the Bla...	0.330627
5	Lord of the Rings: The Fellowship of the Ring,...	0.323939
6	X-Men (2000)	0.314977
7	X2: X-Men United (2003)	0.310659
8	Minority Report (2002)	0.299829
9	Star Wars: Episode II - Attack of the Clones (...	0.264826
10	Spider-Man 2 (2004)	0.257294

h) Model Saving : The KNN model, as well as the processed datasets, are saved using pickle for future use



```
import pickle

pickle.dump(knn, open('artificats\\knn_model.pkl', 'wb'))
pickle.dump(csr_data, open('artificats\\csr_data.pkl', 'wb'))
pickle.dump(final_dataset, open('artificats\\final_dataset.pkl', 'wb'))
pickle.dump(movies, open('artificats\\movies.pkl', 'wb'))
```

2- **app.py:** This is the main application file where the Flask web



```python
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

# Load precomputed data and models
movies = pickle.load(open('artificats/movies.pkl', 'rb'))
knn = pickle.load(open('artificats/knn_model.pkl', 'rb'))
csr_data = pickle.load(open('artificats/csr_data.pkl', 'rb'))
final_dataset = pickle.load(open('artificats/final_dataset.pkl', 'rb'))

def get_recommendation(movie_name):
    movie_list = movies[movies['title'].str.contains(movie_name, case=False)]
    if len(movie_list):
        movie_idx = movie_list.iloc[0]['movieId']
        movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]
        distance, indices = knn.kneighbors(csr_data[movie_idx], n_neighbors=11)
        rec_movies_indices = sorted(list(zip(indices.squeeze().tolist(), distance.squeeze().tolist())), key=lambda x: x[1])[:0: -1]
        recommended_movies = []
        for val in rec_movies_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            recommended_movies.append(movies.iloc[idx]['title'].values[0])
        return recommended_movies
    else:
        return ["Movie not found. Please try another title."]

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/recommend', methods=['POST'])
def recommend_movies():
    movie = request.form.get('movie')
    recommendations = get_recommendation(movie)
    return render_template('recommendations.html', movie=movie, recommendations=recommendations)

if __name__ == '__main__':
    app.run(debug=True)
```

framework handles routes and requests.

- home(): Renders the homepage where users can input a movie title.
- recommend_movies(): Handles the movie recommendation logic, retrieves the movie title from the form, and passes it to the recommendation function.
- get_recommendation(movie_name): This function processes the user input, finds the corresponding movie in the dataset, and uses KNN to return a list of recommended movies.

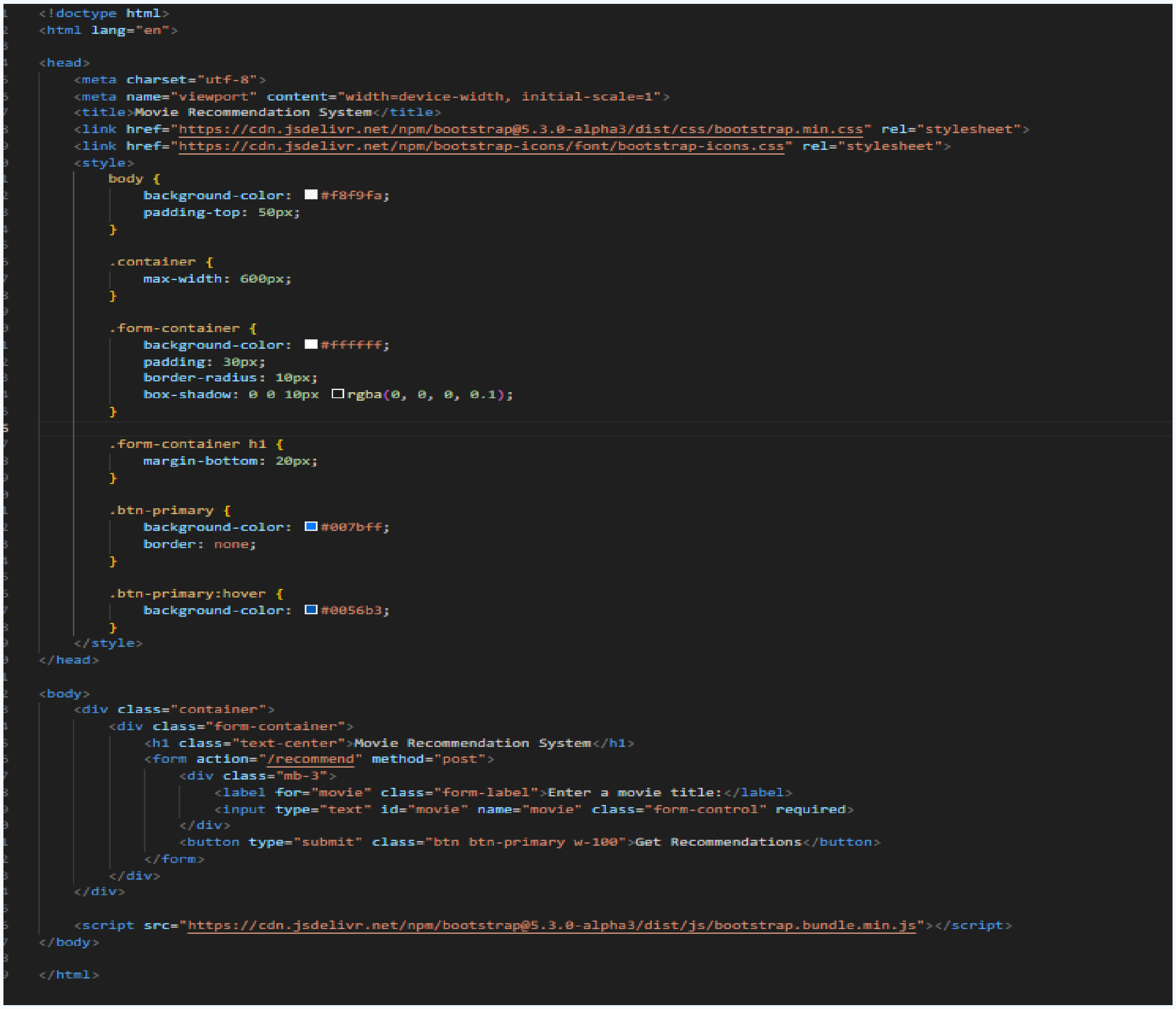
3- **artificats/:**

- movies.pkl: Serialized dataset of movies with features like movie titles and IDs.
- knn_model.pkl: Pre-trained KNN model for finding similar movies.

- csr_data.pkl: Sparse matrix used for similarity calculations between movies.
- final_dataset.pkl: The final processed dataset used for recommendation.

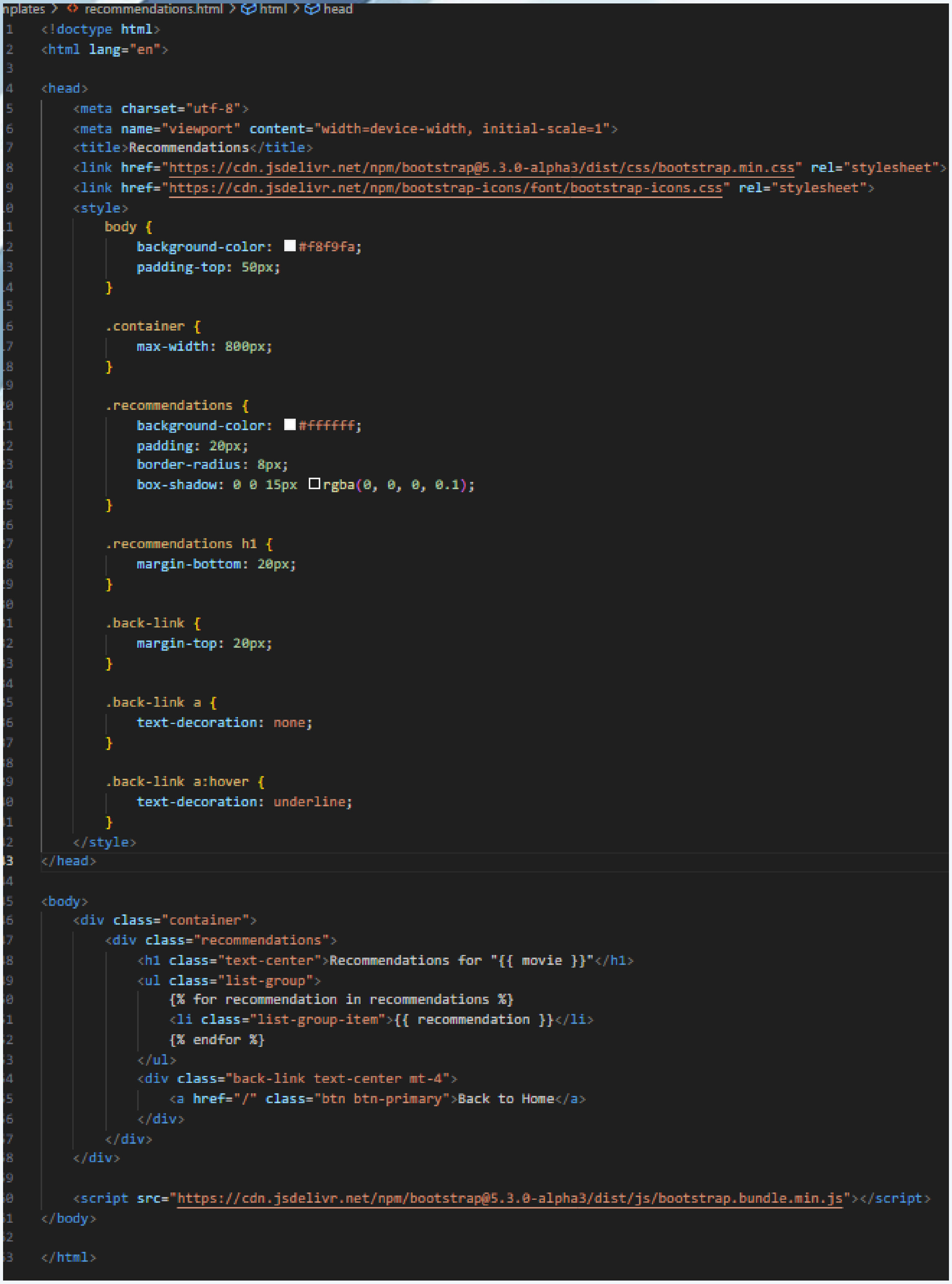
4-**templates/:**

index.html: Basic HTML form that allows users to input a movie title.



```html
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Movie Recommendation System</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
            padding-top: 50px;
        }

        .container {
            max-width: 600px;
        }

        .form-container {
            background-color: #ffffff;
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }

        .form-container h1 {
            margin-bottom: 20px;
        }

        .btn-primary {
            background-color: #007bff;
            border: none;
        }

        .btn-primary:hover {
            background-color: #0056b3;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="form-container">
            <h1 class="text-center">Movie Recommendation System</h1>
            <form action="/recommend" method="post">
                <div class="mb-3">
                    <label for="movie" class="form-label">Enter a movie title:</label>
                    <input type="text" id="movie" name="movie" class="form-control" required>
                </div>
                <button type="submit" class="btn btn-primary w-100">Get Recommendations</button>
            </form>
        </div>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"></script>
</body>

</html>
```

- recommendations.html: Displays the recommended movies based on the user input.



nplates > recommendations.html > html > head

```html
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Recommendations</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
            padding-top: 50px;
        }

        .container {
            max-width: 800px;
        }

        .recommendations {
            background-color: #ffffff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
        }

        .recommendations h1 {
            margin-bottom: 20px;
        }

        .back-link {
            margin-top: 20px;
        }

        .back-link a {
            text-decoration: none;
        }

        .back-link a:hover {
            text-decoration: underline;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="recommendations">
            <h1 class="text-center">Recommendations for "{{ movie }}"</h1>
            <ul class="list-group">
                {% for recommendation in recommendations %}
                <li class="list-group-item">{{ recommendation }}</li>
                {% endfor %}
            </ul>
            <div class="back-link text-center mt-4">
                <a href="/" class="btn btn-primary">Back to Home</a>
            </div>
        </div>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"></script>
</body>

</html>
```

System Workflow Overview

- Data Loading: When the application starts, it loads the necessary movie data and pre-trained KNN model from the artificats folder using Python's pickle library.

- User Input: Users provide a movie title on the home page.
- Recommendation Process:

  1. The system looks for the movie in the dataset.
  2. If found, it retrieves similar movies using KNN, based on the cosine similarity between the movies' features.
  3. If no similar movies are found or if the movie does not exist in the dataset, an error message is shown.

- Result Display: The system returns a list of recommended movies on the recommendations.html page.

Technologies and Libararies Used

- Flask: A lightweight web framework for Python.
- HTML: For creating the web interface.
- Pandas: For reading and manipulating the movie and ratings data.
- Numpy: For handling numerical operations.
- Matplotlib: For visualizing user activity (e.g., how many users have rated a movie).
- Scipy.sparse: For creating sparse matrices.
- Sklearn.neighbors: To implement KNN for recommendation purposes.
- Pickle: For serializing and deserializing data.

My Contribution to the Project:

During the Movie Collaborative Recommendation System project, I primarily focused on the data preprocessing and model preparation tasks. Here's what I did:

1. Data Cleaning and Preparation: I took care of cleaning the raw data, making sure it was ready for use. This involved handling missing data and creating a pivot table that linked users to the movies they rated.
2. Building a Sparse Matrix: To make the recommendation system faster and more efficient, I converted the ratings data into a sparse matrix. This was an important step in optimizing how the system processes data.