MOVIE RECOMMENDATION SYSTEM

A PROJECT REPORT

Department of Information Technology

IT3711 – Summer Internship

VII SEMESTER

Submitted by

ARUN PRAKASH G	**731121205003**
ASHWANTHJEE A R	**731121205004**
AYYANAR S	**731121205005**
SHOBIKA T	**731121205038**
RUTCHAGAN G	**731121205307**

in partial fulfillment of the internship provided

by

TVS Training and Services Centre – 1
(Technical Training, NAPS, Staffing)
Plot no. 7/9A, 7/9B,7/9C,
MTH Road, Ambattur Industrial Estate,
Chennai – 600058

JULY -2024

TVS Training & Services

Date : 31-07-2024

BONAFIDE CERTIFICATE

This is to certify that the project "**MOVIE RECOMMENDATION SYSTEM BASED ON EMOTION**" is the bonafide work of:

No.	Name	Roll No.
1.	ARUN PRAKASH G	731121205003
2.	ASHWANTHJEE A R	731121205004
3.	AYYANAR S	731121205005
4.	RUTCHAGAN G	731121205307
5.	SHOBIKA T	731121205038

who carried out the project work under my supervision.

[illegible]

TVS Training and Services

TVS TRAINING & SERVICE LTD., CHENNAI 600 058

[illegible]

Course Instructor

MR.MOHIDEEN ABDUL KADER JAILANI

TVS Training and Services Centre – 1
(Technical Training, NAPS, Staffing)
Plot no. 7/9A, 7/9B,7/9C,
MTH Road, Ambattur Industrial Estate,
Chennai - 600058

ACKNOWLEDGEMENT

We sincerely express our whole hearted gratitude to the **TVS Training and Services** for their constant encouragement and moral support during the course of this internship.

We owe our sincere thanks to **Mrs. DHANALAKSHMI** and **Mr. AMARNATH** for finishing every essential facility for doing this Internship. We sincerely thank our course instructor **Mr. MOHIDEEN ABDUL KADER JAILANI** for guiding us through the project.

Above all we are grateful to all our Team members for their friendly cooperation and their exhilarating company.

TABLE OF CONTENTS

ABSTRACT

The objective of this project is to develop a movie recommendation system that leverages machine learning algorithms and emotion-based input to enhance user experience. The system is implemented using Python, with Flask serving as the web framework to create an intuitive and interactive interface. The model takes user emotions as input to provide personalized movie recommendations, aiming to connect users with movies that align with their current emotional state.

The project integrates two machine learning models: **Logistic Regression** and **Support Vector Machine (SVM)**. These models are trained on a dataset containing movie features, user ratings, and emotional tags, mapping movies to specific emotional states like happiness, sadness, excitement, or calmness. Based on the user's input emotion, the models predict and rank movies that could evoke a complementary or enhancing emotional experience.

Data preprocessing is a critical component, involving cleaning the dataset, handling missing values, feature scaling, and encoding categorical variables. The logistic regression model predicts the probability of a user liking a particular movie based on their current mood, while the SVM model assists in separating movies into distinct emotional categories. The combination of these models ensures accuracy and diversity in recommendations.

The system is deployed using Flask, which provides a lightweight and efficient interface for users to input their emotions. The Flask framework is integrated with the machine learning models and a backend database that stores movie information, user interaction history, and ratings. The application interface allows users to select their current emotional state through a simple and user-friendly form, after which the system returns a curated list of movie recommendations.

In addition to emotion-based filtering, the system considers user preferences, past ratings, and genres to fine-tune recommendations over time. This hybrid approach combines content-based filtering and collaborative filtering techniques, enhancing the personalization of suggestions.

The result is a movie recommendation system that dynamically adapts to both the user's emotional needs and their historical preferences. This makes the system more engaging and effective at capturing the user's current mood, providing an enjoyable and immersive movie-watching experience.

CHAPTER 1

INTRODUCTION

In today's digital age, the vast amount of available content can often overwhelm users, particularly when it comes to entertainment options like movies. Platforms like Netflix, Amazon Prime, and Disney+ offer extensive catalogs of films across multiple genres, but this abundance can lead to decision fatigue, where users struggle to choose what to watch. Traditional recommendation systems, which typically rely on past behavior and preferences, often fall short in addressing the emotional aspect of a user's decision-making process. This limitation paves the way for an evolution in recommendation technology—one that incorporates human emotions into the equation.

This project aims to develop a movie recommendation system that goes beyond conventional collaborative and content-based filtering methods. By integrating emotion-based input, the system recognizes that a user's emotional state plays a significant role in their movie selection process. Someone who is feeling sad may want to watch an uplifting movie, while someone who is excited might look for a high-energy action film. By bridging the gap between technology and psychology, this system caters more effectively to a user's current mood, enhancing their viewing experience.

Need for Evolution of the Project

Traditional movie recommendation systems rely heavily on user behavior—specifically, their past choices, ratings, and viewing habits. While this approach has proven successful to an extent, it has limitations. It does not account for a user's emotional state, which can vary day to day or even hour to hour. For instance, a user who enjoyed watching a mystery thriller a week ago might not be in the mood for something dark or intense today. Ignoring these emotional nuances often leads to dissatisfaction with the recommendations, as they may not resonate with the user's current feelings.

As artificial intelligence and machine learning continue to evolve, incorporating emotion detection into recommendation systems becomes increasingly viable. This project seeks to address this gap by introducing a machine learning-based system that takes emotion as input and outputs personalized recommendations that align with the user's emotional state. This innovation is not only a step forward in enhancing recommendation accuracy but also in improving user satisfaction and engagement.

The need for a more dynamic and responsive recommendation system has been amplified by trends in modern entertainment consumption. Streaming services, social media platforms, and even music apps have begun to embrace personalized, emotionally aware experiences. This project extends that principle to movies, recognizing that how we feel significantly impacts what we want to watch. By building a system that listens to these emotional cues, we can offer more relevant and satisfying recommendations.

Project Development

This project is built using **Python** as the primary programming language and **Flask** as the web framework to create a user-friendly interface. The development process involves several key stages:

1. **Data Collection and Preprocessing**: The first step in creating this system is gathering a dataset that contains movie information, user ratings, and associated emotional tags. This includes movie metadata (such as genre, cast, and release year) and emotional indicators (like happiness, sadness, excitement, etc.). Preprocessing involves cleaning the data, filling in missing values, and encoding categorical variables like genre and emotional tags into a machine-readable format.
2. **Model Training**: The system utilizes two machine learning models: **Logistic Regression** and **Support Vector Machine (SVM)**. Logistic regression is trained to predict the probability that a user will enjoy a certain movie given their current emotional state. SVM, on the other hand, is trained to classify movies based on emotional categories. Both models are trained on the processed dataset and tuned for performance, ensuring they can accurately predict and classify movies based on emotional input.
3. **Integration with Flask**: Flask is used to create the web interface where users can input their emotions. Flask integrates seamlessly with the machine learning models to deliver recommendations in real time. Users can interact with the system via a simple interface where they select their current emotional state, and the backend processes this input to generate a tailored list of movie recommendations.
4. **Deployment and User Feedback**: Once the system is deployed, real-time user feedback helps refine the models. As users interact with the system and rate recommended movies, this feedback loop helps improve the accuracy of future recommendations. The system evolves over time, becoming more responsive and personalized.

Real-Time Applications

1. **Streaming Services**: Platforms like Netflix, Hulu, and Amazon Prime can benefit from integrating emotion-based recommendation systems to provide users with more relevant content. Emotion-aware algorithms can reduce user frustration by aligning recommendations with their current mood, enhancing overall satisfaction and engagement with the platform.

2. **Smart TV and Media Devices**: Smart TVs and streaming devices could utilize emotion-detection cameras or wearable devices to automatically gauge user emotions and suggest appropriate movies. This creates a hands-free, seamless experience, reducing the cognitive load of selecting content.
3. **Mental Health and Wellness Apps**: Incorporating this technology into mental health and wellness applications can provide users with media that supports their emotional well-being. For instance, a mental health app could recommend calming or uplifting movies based on real-time emotional analysis, providing users with a therapeutic outlet through entertainment.
4. **Interactive Movie Theaters**: As theaters begin incorporating digital experiences, emotion-based systems could suggest movies or even organize events tailored to the collective mood of an audience, offering an engaging social experience.

By developing a movie recommendation system that takes emotion as input, this project sets the stage for a new generation of personalized media experiences. Whether integrated into streaming platforms or health and wellness applications, emotion-aware recommendations have the potential to significantly enhance user experience, making movie-watching more intuitive, satisfying, and emotionally fulfilling.

CHAPTER 2

REQUIREMENTS

To successfully develop a movie recommendation system that takes emotion as an input, certain technical, software, hardware, and dataset requirements are essential. These requirements ensure that the system operates efficiently, provides accurate recommendations, and offers a smooth user experience. Below is a detailed breakdown of the key components necessary for the development of this system:

1. Programming Language

- **Python**: Python is the primary language for this project due to its extensive libraries and frameworks for machine learning, data processing, and web development. It offers versatility and ease of use, which is crucial for developing both the recommendation engine and the web interface.

2. Frameworks and Libraries

To implement various features in the system, specific Python libraries and frameworks are required:

- **Flask**: This lightweight web framework is ideal for developing the user interface where users can input their emotions. Flask allows for easy routing, form handling, and integration with machine learning models.
- **Scikit-learn**: This library is essential for implementing machine learning algorithms such as Logistic Regression and Support Vector Machines (SVM). It provides tools for model training, validation, and tuning.
- **Pandas**: Pandas is used for data manipulation and analysis. It helps in processing large datasets, handling missing values, and performing operations on structured data.
- **NumPy**: This library is used for numerical computations, such as matrix operations, which are necessary during machine learning model training.
- **Pickle**: The pickle library in Python is used for serializing and deserializing Python objects, allowing them to be saved to a file or transmitted over a network. It converts Python objects into a byte stream and can later reconstruct the objects from the stream back into their original form.
- **Matplotlib/Seaborn**: These libraries are useful for visualizing data trends, model performance, and exploring relationships between variables like user preferences and movie ratings.
- **TensorFlow/PyTorch** (optional): If deep learning is incorporated into the project, such as using neural networks for emotion detection, TensorFlow or PyTorch would be necessary for building and training more complex models.

- **NLTK or TextBlob** (optional): These libraries can be used for natural language processing if the system aims to incorporate sentiment analysis from user reviews or comments.
- **render_template**: This function is used to render HTML templates in your Flask application. It helps you dynamically generate HTML by passing data from your backend (Python code) to your frontend (HTML files) and rendering the final output to be displayed in the browser.
- **request**: This object is used to handle incoming HTTP requests in Flask. It allows access to form data, query parameters, and other request information like HTTP methods (GET, POST). It is commonly used when you need to extract data submitted by users through forms or URL parameters.
- **predict_genre**: This is likely a custom function that is imported from a module called model. It is presumably responsible for predicting the movie genre based on certain inputs, such as user emotions or other features. This function is typically part of the machine learning logic and could be using pre-trained models to make predictions.
- **load_movies_from_csv**: This is likely a custom utility function that loads movie data from a CSV file. It could involve reading movie metadata like titles, genres, and descriptions into a usable format such as a list of dictionaries or a Pandas DataFrame.
- **filter_movies_by_genre**: This is a custom function used to filter movies by a given genre. It likely takes the list of all movies and returns a subset of movies that match the genre predicted by the machine learning model or user input.
- **csv**: The csv library in Python simplifies reading from and writing to CSV files, making it easy to work with tabular data in a structured format.
- **Textwrap:** The textwrap library helps format and wrap text to fit within a specified width, ensuring proper readability and layout in console or text-based outputs.

3. Datasets

Data plays a crucial role in training machine learning models for recommendation systems. The dataset should include detailed information about movies and user preferences. Key dataset requirements include:

- **Movie Metadata**: Information such as movie titles, genres, directors, actors, release years, and descriptions. This data will help in content-based filtering and matching movies with user preferences.
- **User Rating Data**: User ratings of movies on a scale (e.g., 1-5 stars or 1-10 ratings). This data is crucial for collaborative filtering techniques and helps the model learn which types of movies specific users prefer.
- **Emotional Tags**: Associating movies with emotional tags (e.g., happiness, sadness, excitement) is necessary for emotion-based recommendation. This can be derived from metadata, user reviews, or specific emotion datasets.

- **Emotion Datasets**: These datasets could include mappings of emotions to movie genres or individual films, as well as facial expression datasets or emotion detection datasets (e.g., FER2013 for facial emotion recognition) for advanced emotion prediction models.

4. Machine Learning Models

For this project, two core machine learning models are used:

- **Logistic Regression**: This model is used to predict the probability of a user liking a specific movie based on their current emotional input and past behavior. It is a good baseline model for binary classification tasks.
- **Support Vector Machine (SVM)**: SVM is used to categorize movies into emotional categories, helping the system match a user's current emotional state with a relevant set of movies. It is particularly useful when dealing with high-dimensional data and complex decision boundaries.

Both models require tuning and validation to ensure accuracy. Cross-validation techniques like K-Fold and hyperparameter optimization (e.g., grid search) will be used during the model development phase.

5. Emotion Detection

Emotion-based input can be gathered in multiple ways depending on the complexity of the project:

- **Manual Input**: The simplest approach is allowing users to manually select their current emotion from a list (e.g., happy, sad, excited).
- **Facial Emotion Recognition**: For a more advanced system, facial emotion detection can be integrated using computer vision. This can be done using pre-trained models in libraries like OpenCV and deep learning frameworks like TensorFlow or PyTorch.
- **Sentiment Analysis**: Sentiment analysis can be applied to textual input from users, such as short comments or feedback, to detect emotional sentiment.

6. Front-End Development

To create an intuitive and user-friendly interface, front-end development tools are essential:

- **HTML/CSS**: For structuring and styling the web pages where users interact with the system.

8. Deployment

Once the system is developed, it needs to be deployed so that users can access it remotely. Deployment tools and services include:

- **Flask**: As the system's web framework, Flask serves the application to users locally during development and in production via web hosting services.
- **Heroku/Google Cloud/AWS**: Cloud platforms like Heroku, Google Cloud, and AWS are commonly used to deploy Flask applications. These platforms offer easy scalability, database management, and integration with machine learning pipelines.
- **Docker** (optional): Containerization with Docker can help package the application and its dependencies to ensure consistency across different deployment environments.

9. Hardware Requirements

The hardware requirements for developing and running this project depend on the complexity of the models and datasets being used:

- **Development Environment**: A standard modern computer or laptop with at least 8GB of RAM, a multi-core processor, and sufficient storage (e.g., 500GB) will suffice for local development and testing.
- **Cloud-Based Infrastructure**: For large-scale model training, cloud services like Google Colab, AWS EC2, or Google Cloud AI Platform may be required. These platforms provide access to GPUs and TPUs for faster model training and can handle large datasets efficiently.

10. Version Control

- **Git/GitHub**: Version control is essential for tracking changes during development, particularly in collaborative environments. Git is the most commonly used version control tool, and GitHub provides cloud storage for code, enabling collaboration and backup.

CHAPTER 3

MACHINE LEARNING MODELS

In building an emotion-based movie recommendation system, **Logistic Regression** and **Support Vector Machine (SVM)** models are two widely used machine learning algorithms. Both play a significant role in enhancing the predictive accuracy of the recommendation engine by processing user emotions and mapping them to relevant movie genres or titles. Below is a detailed explanation of how these models work and how they are applied in this project.

Logistic Regression

Logistic Regression is a supervised machine learning algorithm primarily used for binary classification problems, but it can also be extended to multi-class classification. In this project, logistic regression helps in predicting whether a user will like a specific movie based on their emotional state and previous interactions.

How Logistic Regression Works

Logistic regression models the probability of a certain class (e.g., "like" or "dislike" for a movie) using a logistic function, also known as the sigmoid function. The sigmoid function maps predicted values into the range of probabilities between 0 and 1. These probabilities are then used to classify the input into one of the two classes (or more in the case of multi-class classification).

- **Mathematical Representation**:

  The logistic regression model predicts the probability p that a given input (e.g., user's emotional state) belongs to a particular class:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

  where β represents the model coefficients and x represents the input features (e.g., emotional features like "happiness" or "sadness").

  If the probability p exceeds a given threshold (e.g., 0.5), the input is classified as belonging to the positive class (e.g., "like"); otherwise, it is classified as belonging to the negative class (e.g., "dislike").

Application in Movie Recommendation

In the context of an emotion-based movie recommendation system, logistic regression can be used to predict the likelihood of a user enjoying a particular movie given their current emotional state and past behavior. The key steps in applying logistic regression are as follows:

1. **Feature Extraction**: Emotional states (e.g., "happy," "sad," "excited") and movie metadata (e.g., genre, actors, ratings) are extracted as features.
2. **Model Training**: The logistic regression model is trained on a dataset that contains user ratings for various movies along with associated emotional states. The training process involves estimating the model parameters that best fit the data.
3. **Prediction**: Once trained, the model can predict the probability that a user will enjoy a movie based on their current emotional input.
4. **Recommendation**: Movies with the highest predicted probabilities of being liked by the user are recommended.

By using logistic regression, the recommendation system can quickly classify whether a user will enjoy certain movies based on their emotional state, improving the personalization of recommendations.

Support Vector Machine (SVM)

Support Vector Machine (SVM) is another powerful supervised learning algorithm that excels at classification tasks, especially in high-dimensional spaces. Unlike logistic regression, SVM focuses on finding the optimal boundary (or hyperplane) that separates different classes of data.

How SVM Works

SVM works by finding the hyperplane that maximizes the margin between different classes in the dataset. The margin is defined as the distance between the hyperplane and the closest data points from each class, which are called **support vectors**. The goal of SVM is to find the hyperplane with the largest margin, ensuring better generalization on new, unseen data.

- **Mathematical Representation**:

  The decision boundary or hyperplane is represented as:

$$w \cdot x + b = 0$$

  where w is the normal vector to the hyperplane, x is the input feature vector, and b is the bias term. The model optimizes w and b to maximize the margin.

  - For linear SVM, the hyperplane is a straight line (or a plane in higher dimensions) that separates the classes.
  - For non-linear data, SVM can use kernel functions (such as radial basis functions or polynomial kernels) to map the input data into higher dimensions, where a linear separation is possible.

Application in Movie Recommendation

In the movie recommendation system, SVM can be employed to classify movies into distinct emotional categories based on features like genres, user ratings, and emotional tags. The workflow is as follows:

1. **Feature Extraction**: Movies are represented by features such as genres, emotional tags, and ratings. The system also uses emotional features derived from the user's input (e.g., "happy," "sad").
2. **Model Training**: The SVM model is trained on a labeled dataset that associates movies with emotional categories. For example, movies labeled as "happy" might include comedies, while "sad" movies could be dramas or tragedies. SVM learns to separate these categories by finding the optimal decision boundary between them.
3. **Prediction**: Given a user's current emotion, the SVM model predicts which emotional category is most appropriate for the user's mood. For instance, if the user is feeling "happy," the SVM might suggest movies classified as "happy" or "exciting."
4. **Recommendation**: After identifying the emotional category, the system filters movies based on that category and recommends those that match the user's current emotional state.

Advantages of SVM in the Project

- **High Dimensionality**: SVM is highly effective in handling high-dimensional data, which is typical in movie recommendation tasks where numerous features are involved (e.g., emotional states, user preferences, and movie metadata).
- **Clear Separation**: SVM excels at separating classes with clear margins, ensuring that movies are distinctly categorized into emotional groups, leading to more relevant recommendations.
- **Kernel Trick**: The use of kernel functions enables SVM to handle non-linear relationships in the data, which is beneficial when emotional tags do not follow a simple linear pattern.

Comparison of Logistic Regression and SVM in This Project

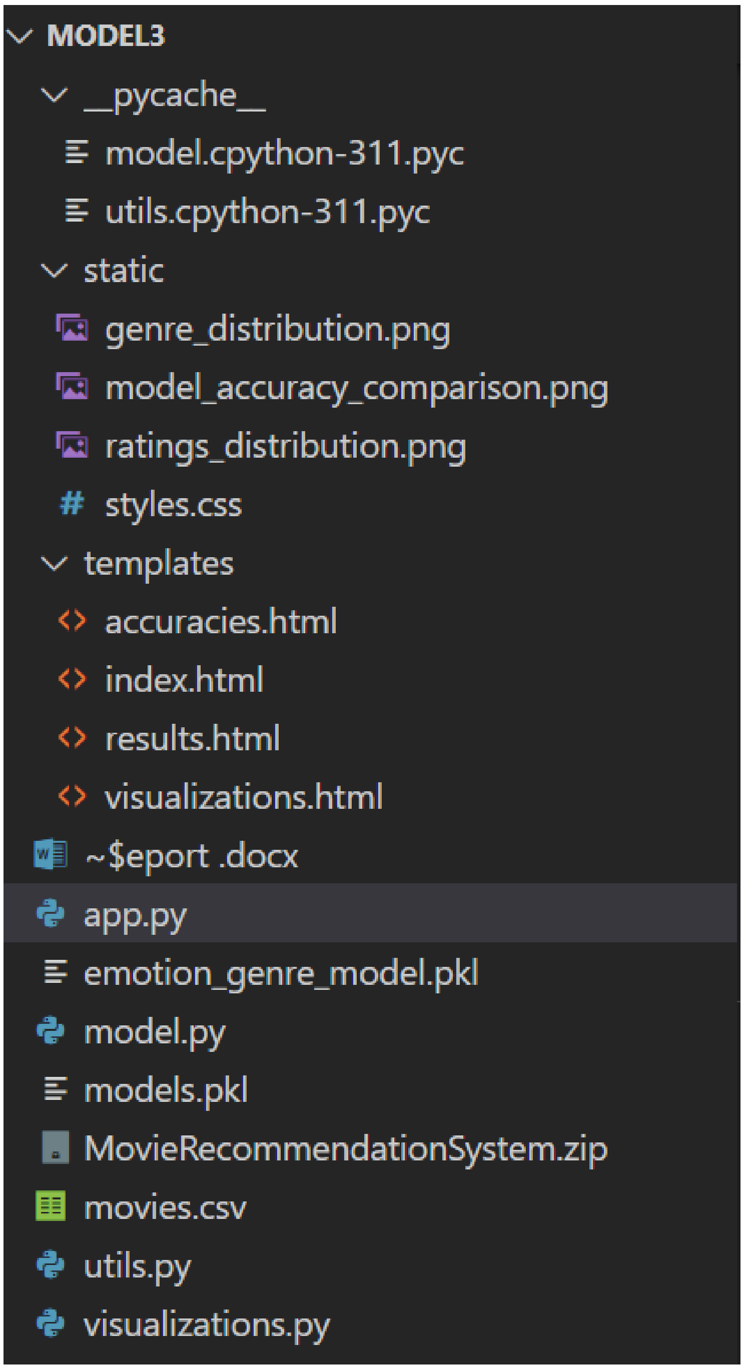
Aspect	**Logistic Regression**	**SVM**
Purpose	Predicts the probability of a user liking a movie based on emotion and past behavior.	Classifies movies into emotional categories for recommendation.
Classification Type	Primarily binary/multi-class classification based on probability.	Binary or multi-class classification with optimal margin separation.
Complexity	Relatively simple and interpretable model.	More complex, especially when using kernels for non-linear data.
Handling Non-linearity	Struggles with non-linear relationships in the data.	Can handle non-linear data via kernel functions.
Use in the Project	Used for predicting user preferences and personalizing recommendations based on emotion.	Used for classifying movies into emotional categories, improving accuracy of recommendations.

Conclusion

Both Logistic Regression and SVM serve complementary roles in the movie recommendation system. Logistic Regression helps in estimating user preferences by calculating probabilities of liking a movie based on emotional input, whereas SVM ensures accurate classification of movies into emotional categories, making the recommendations more targeted. By leveraging these two models, the system offers a balanced approach to personalization, combining predictive power with fine-tuned emotional filtering to deliver an engaging and emotionally resonant movie-watching experience.

CHAPTER 4

DIRECTORY STRUCTURE



```
MODEL3
  __pycache__
    model.cpython-311.pyc
    utils.cpython-311.pyc
  static
    genre_distribution.png
    model_accuracy_comparison.png
    ratings_distribution.png
    styles.css
  templates
    accuracies.html
    index.html
    results.html
    visualizations.html
  ~$eport .docx
  app.py
  emotion_genre_model.pkl
  model.py
  models.pkl
  MovieRecommendationSystem.zip
  movies.csv
  utils.py
  visualizations.py
```

app.py :

- Takes an emotion input from the user and predicts a movie genre based on a model (predict_genre).
- Filters a movie database (load_movies_from_csv) based on the predicted genre and sorts the results by rating (filter_movies_by_genre).
- Displays either error message (for invalid input) or filtered movie list in separate web pages (index.html and results.html).

model.py :

- Trains a Logistic Regression model to predict movie genre based on emotions using TF-IDF for feature extraction. (Creates a model that maps emotions to genres)
- Saves the trained model and supporting components (vectorizer and genre encoder) for later use. (Packages the model for easy access)
- Defines a function predict_genre that takes an emotion as input, uses the saved model to predict its genre, and returns the predicted genre. (Applies the saved model to new emotions).

visualize.py :

- Loads movie data from CSV, checks for specific columns (Genre and a text column), and prepares features (TF-IDF) and target labels (genre encoding).
- Trains Logistic Regression and Support Vector Machine models, evaluates their accuracy on a test set, saves the trained models for later use.
- Generates visualizations for genre distribution, movie rating distribution, and model accuracy comparison, saves these visualizations as images.

utils.py :

- Loads movie data from a CSV file into a list of dictionaries.
- Filters the movies based on a specified genre, returning those that match the genre.
- Wraps text to a specified width for better formatting**.**

Templates:

- index.html
- result.html
- visualization.html
- accuracies.html

Static:

After running visualizations.py, the static folder will contain two types of files:

1. **styles.css**: This file likely contains styles used to format the visualizations created by the script (e.g., fonts, colors, layout).
2. **PNG images**: These images are probably the visualizations themselves, saved in PNG format. These visualizations could be charts or graphs generated from the movie data.

movies.csv:

The dataset movies.csv consists of the following columns in which the data is already clean:

- **Name**: The title of the movie.
- **Genre**: The genre of the movie (e.g., Action, Comedy, Drama).
- **Rating**: The rating of the movie (likely on a scale from 1 to 10).
- **Director**: The name of the director of the movie.
- **Actor**: The name of the main actor in the movie.
- **Year**: The year the movie was released.
- **Hero Rating**: The rating given to the hero of the movie.
- **Movie Rating**: The rating given to the movie overall.
- **Content Rating**: The content rating of the movie (e.g., U, PG, PG-13).

This dataset appears to contain information about 330 movies, providing details like their titles, genres, ratings, directors, actors, release years, and various ratings related to the hero and the movie itself.

CHAPTER 5

PROGRAMMING MODULES

app.py:

```
from flask import Flask, render_template, request

from model import predict_genre

from utils import load_movies_from_csv, filter_movies_by_genre


app = Flask(__name__)


@app.route('/', methods=['GET'])

def index():

    return render_template('index.html')


@app.route('/results', methods=['GET','POST'])

def results():

    emotion = request.form['emotion'].strip()

    genre = predict_genre(emotion)


    if not genre:

        return render_template('results.html', error="Invalid emotion or no matching genre found.", movies=[])


    movies = load_movies_from_csv('movies.csv')

    filtered_movies = filter_movies_by_genre(movies, genre)

    filtered_movies.sort(key=lambda x: float(x['Rating']), reverse=True)


    return render_template('results.html', movies=filtered_movies[:70])
```

```
@app.route('/visualizations', methods=['GET'])

def visualizations():

    return render_template('visualizations.html')


if __name__ == '__main__':

    app.run(debug=True)
```

Code Breakdown:

- **Imports**:
    - Flask: A web framework for building web applications in Python.
    - render_template: A function to render HTML templates.
    - request: An object to handle incoming HTTP requests.
    - predict_genre: A function presumably defined in model.py to predict a movie genre based on an emotion.
    - load_movies_from_csv: A utility function from utils.py to load movie data from a CSV file.
    - filter_movies_by_genre: A utility function to filter movie data based on genre.
- **Flask App Initialization**:
    - app = Flask(__name__): Creates an instance of the Flask application.
- **Route Definitions**:
    - **/ Route**:
        - **Method**: GET
        - **Function**: index()
        - **Purpose**: Renders the index.html template, which is likely the landing page of the application.
    - **/results Route**:
        - **Methods**: GET and POST
        - **Function**: results()
        - **Purpose**: Handles form submissions and requests to display movie recommendations.
        - **Process**:

        - Retrieves the emotion parameter from the form data.
        - Uses predict_genre(emotion) to get the movie genre corresponding to the given emotion.
        - If no genre is found, it renders results.html with an error message and no movie data.
        - Loads movies from a CSV file using load_movies_from_csv('movies.csv').
        - Filters the movies by the predicted genre using filter_movies_by_genre(movies, genre).
        - Sorts the filtered movies by rating in descending order.
        - Renders results.html with the top 70 movies sorted by rating.
    - **/visualizations Route**:
        - **Method**: GET
        - **Function**: visualizations()
        - **Purpose**: Renders the visualizations.html template, which likely contains visualizations or charts related to the movie data.
- **Running the Application**:
    - if __name__ == '__main__': app.run(debug=True): Runs the Flask application in debug mode when the script is executed directly.

This code sets up a web application that provides movie recommendations based on user input and displays visualizations, integrating various utility functions and handling form submissions.

visualizations.py:

```
import pickle

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC


# Load movie data from CSV

def load_movies_from_csv(file_path):

    """Load movie data from a CSV file and return a DataFrame."""

    try:

        return pd.read_csv(file_path)

    except FileNotFoundError:

        print(f"Error: The file {file_path} was not found.")

        return pd.DataFrame()

    except Exception as e:

        print(f"Error reading file: {e}")

        return pd.DataFrame()


# Load movies data

movies_df = load_movies_from_csv('movies.csv')


# Ensure the 'Genre' and a text column are present in the DataFrame

text_column = 'Genre'  # Replace with the actual name of your text column
```

```
if 'Genre' in movies_df.columns and text_column in movies_df.columns:
    # Prepare training data
    genres = movies_df['Genre']
    text_data = movies_df[text_column]

    # Vectorize movie text data
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(text_data)

    # Encode genres
    genre_encoder = {genre: i for i, genre in enumerate(set(genres))}
    y = [genre_encoder[genre] for genre in genres]

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train Logistic Regression model
    logistic_model = LogisticRegression(max_iter=1000)
    logistic_model.fit(X_train, y_train)
    y_pred_logistic = logistic_model.predict(X_test)
    logistic_accuracy = accuracy_score(y_test, y_pred_logistic)

    # Train Support Vector Machine model
    svm_model = SVC(kernel='linear')
    svm_model.fit(X_train, y_train)
    y_pred_svm = svm_model.predict(X_test)
    svm_accuracy = accuracy_score(y_test, y_pred_svm)

    # Save models and vectorizer
```

```
with open('models.pkl', 'wb') as file:
    pickle.dump((vectorizer, logistic_model, svm_model, genre_encoder), file)

# Print accuracy
print(f"Logistic Regression Accuracy: {logistic_accuracy:.2f}")
print(f"SVM Accuracy: {svm_accuracy:.2f}")

# Visualize genre distribution in movies
plt.figure(figsize=(12, 6))
genre_counts = movies_df['Genre'].value_counts()
sns.barplot(x=genre_counts.index, y=genre_counts.values, palette='viridis')
plt.title('Distribution of Genres in Movies Dataset')
plt.xlabel('Genre')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')

# Add data labels
for index, value in enumerate(genre_counts.values):
    plt.text(index, value + 0.5, str(value), ha='center', va='bottom')

plt.tight_layout()
plt.savefig('static/genre_distribution.png')
plt.close()

# Visualize movie ratings distribution
plt.figure(figsize=(12, 6))
sns.histplot(movies_df['Rating'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')

# Add data labels
for patch in plt.gca().patches:
    height = patch.get_height()
    plt.text(patch.get_x() + patch.get_width() / 2, height + 0.5, f'{int(height)}', ha='center', va='bottom')

plt.tight_layout()
plt.savefig('static/ratings_distribution.png')
plt.close()

# Visualize model accuracy comparison
model_names = ['Logistic Regression', 'SVM']
accuracies = [logistic_accuracy, svm_accuracy]

plt.figure(figsize=(10, 5))
sns.barplot(x=model_names, y=accuracies, palette='coolwarm')
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0, 1.5)

# Add data labels
for index, value in enumerate(accuracies):
    plt.text(index, value + 0.02, f'{value:.2f}', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.savefig('static/model_accuracy_comparison.png')
plt.close()
```

```
else:
    print("The required columns are not present in the CSV file.")
```

Code Breakdown:

- **Imports**:
  - pickle, matplotlib.pyplot, seaborn, pandas: For data handling, visualization, and model serialization.
  - TfidfVectorizer, train_test_split, accuracy_score, LogisticRegression, SVC: For text vectorization, model training, and evaluation.

- **Data Loading**:

  - load_movies_from_csv(file_path) : Reads movie data from a CSV file into a DataFrame. Handles file not found and other errors.

- **Data Preparation**:

  - Ensures columns 'Genre' and a specified text column exist.
  - Vectorizes text data using TfidfVectorizer.
  - Encodes genres into numerical labels.
  - Splits data into training and testing sets.

- **Model Training**:

  - Trains a LogisticRegression model and an `SVC` model on the training data.
  - Evaluates both models on the test data and prints their accuracies.

- **Model Serialization**:

  - Saves the trained models, vectorizer, and genre encoder to a file using `pickle`.

- **Data Visualization**:

  - **Genre Distribution**: Plots the distribution of movie genres and saves it as an image.
  - **Ratings Distribution**: Creates a histogram of movie ratings and saves it as an image.
  - **Model Accuracy Comparison**: Compares the accuracies of the Logistic Regression and SVM models using a bar plot and saves it as an image.

- **Error Handling**:

  - Checks for required columns and prints an error message if they are missing.

model.py:

```
import pickle

from sklearn.linear_model import LogisticRegression

from sklearn.feature_extraction.text import TfidfVectorizer


# Sample data for training the model

emotion_genre_data = {

    "happy": "Comedy",

    "sad": "Drama",

    "excited": "Action",

    "fear": "Horror",

    "love": "Romance",

    "curious": "Mystery",

    "inspired": "Biography",

    "adventurous": "Adventure",

    "tense": "Thriller",

    "intrigued": "Crime",

    "Drama": 'Drama',

    "Action": 'Action',

    "Thriller": 'Thriller',

    "Adventure": 'Adventure',

    "Comedy": 'Comedy',

    "Biography": 'Biography',

    "Mystery": 'Mystery',

    "Romance": 'Romance',

    "Crime": 'Crime',

    "Horror": 'Horror',

}
```

```
# Prepare training data
emotions = list(emotion_genre_data.keys())
genres = list(emotion_genre_data.values())

# Vectorize emotions
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(emotions)

# Encode genres
genre_encoder = {genre: i for i, genre in enumerate(set(genres))}
y = [genre_encoder[genre] for genre in genres]

# Train logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Save model and vectorizer
with open('emotion_genre_model.pkl', 'wb') as file:
    pickle.dump((vectorizer, model, genre_encoder), file)

def predict_genre(emotion):
    """Predict the genre based on the given emotion using the trained model."""
    with open('emotion_genre_model.pkl', 'rb') as file:
        vectorizer, model, genre_encoder = pickle.load(file)

    emotion_vector = vectorizer.transform([emotion])
    predicted_genre_index = model.predict(emotion_vector)[0]

    genre_decoder = {i: genre for genre, i in genre_encoder.items()}
    return genre_decoder.get(predicted_genre_index)
```

Code Breakdown:

- **Data Preparation:** Maps emotions to genres and prepares training data.
- **Vectorization:** Uses TfidfVectorizer to convert emotions into numerical features.
- **Encoding:** Converts genres into numerical labels.
- **Training:** Trains a LogisticRegression model on the vectorized emotions and encoded genres.
- **Saving:** Serializes the model, vectorizer, and genre encoder with pickle.
- **Prediction:** Defines a predict_genre function to load the model, vectorize a given emotion, predict the genre, and decode the genre label.

utils.py:

```
import csv

import textwrap

def load_movies_from_csv(file_path):

    """Load movie data from a CSV file and return a list of dictionaries."""

    movies = []

    try:

        with open(file_path, mode='r', newline='', encoding='utf-8') as file:

            reader = csv.DictReader(file)

            for row in reader:

                movies.append(row)

    except FileNotFoundError:

        print(f"Error: The file {file_path} was not found.")

    except Exception as e:

        print(f"Error reading file: {e}")

    return movies


def filter_movies_by_genre(movies, genre):

    """Filter movies by the specified genre."""

    return [movie for movie in movies if genre in movie['Genre'].split(', ')]
```

```
def wrap_text(text, width=20):
    """Wrap text to a specified width."""
    return "\n".join(textwrap.wrap(text, width))
```

Code Breakdown:

- **load_movies_from_csv(file_path):**
  - Purpose: Reads movie data from a CSV file and returns it as a list of dictionaries.
  - Implementation:
    - Opens the CSV file with utf-8 encoding.
    - Uses csv.DictReader to parse the CSV into dictionaries where each row represents a movie.
    - Appends each row (movie) to a list.
    - Handles FileNotFoundError if the file does not exist and other general exceptions.
    - Returns the list of movie dictionaries.
- **filter_movies_by_genre(movies, genre):**
  - Purpose: Filters the list of movies to include only those matching the specified genre.
  - Implementation:
    - Iterates through the list of movie dictionaries.
    - Checks if the specified genre is in the Genre field of each movie (which is split by commas).
    - Returns a list of movies that match the genre.
- **wrap_text(text, width=20):**
  - Purpose: Formats a string into lines of specified width for better readability.
  - Implementation:
    - Uses textwrap.wrap to break the text into lines of the given width (default 20 characters).
    - Joins the lines with newline characters to create a formatted string.

These functions facilitate reading and processing movie data from a CSV file and formatting text for display or printing.

CHAPTER 6

TEMPLATES AND STYLES

Templates:

- **index.html**
  - **Document Structure**:
    - **<!DOCTYPE html>**: Declares the document type and HTML version.
    - **<html lang="en">**: Specifies English as the language of the document.
  - **Head Section**:
    - **Meta Tags**: Define the character set as UTF-8 and set viewport settings for responsive design.
    - **Title**: Sets the page title to "Movie Recommendation".
    - **Stylesheets**:
      - Links to an external CSS file (styles.css) for custom styling.
      - Includes Font Awesome for icons via a CDN.
  - **Body Section**:
    - **Container Div**: Wraps content for layout and styling.
    - **Header**: Contains a heading (<h1>) with the text "Find Your Next Movie".
    - **Main**:
      - **Form**: Posts user input to the /results endpoint.
      - **Label and Input**: Prompts users to enter an emotion, with an input field and a submit button.
  - This code sets up a simple interface for users to input emotions and request movie recommendations.

- **result.html**
  - **Head Section**:
    - **Meta Tags**: Ensure proper character encoding and responsive design.
    - **Title**: Sets the page title to "Movie Recommendations".
    - **Stylesheets**: Links to a custom CSS file (styles.css) and Font Awesome for icons.
  - **Body Section**:
    - **Container Div**: Holds the main content for layout.
    - **Header**: Displays the heading "Recommended Movies".
    - **Main Content**:
      - **Error Handling**: Displays an error message if an error variable is set.
      - **Movies Table**:
        - **Table Header**: Defines columns for serial number, title, year, director, actor, IMDB rating, genre, and public rating.
      - **Table Body**: Dynamically populates rows with movie data from a movies variable.
    - **Navigation Links**:
      - **Back Button**: Provides a link to return to the home page.
      - **View Visualizations**: Links to a page showing data visualizations.

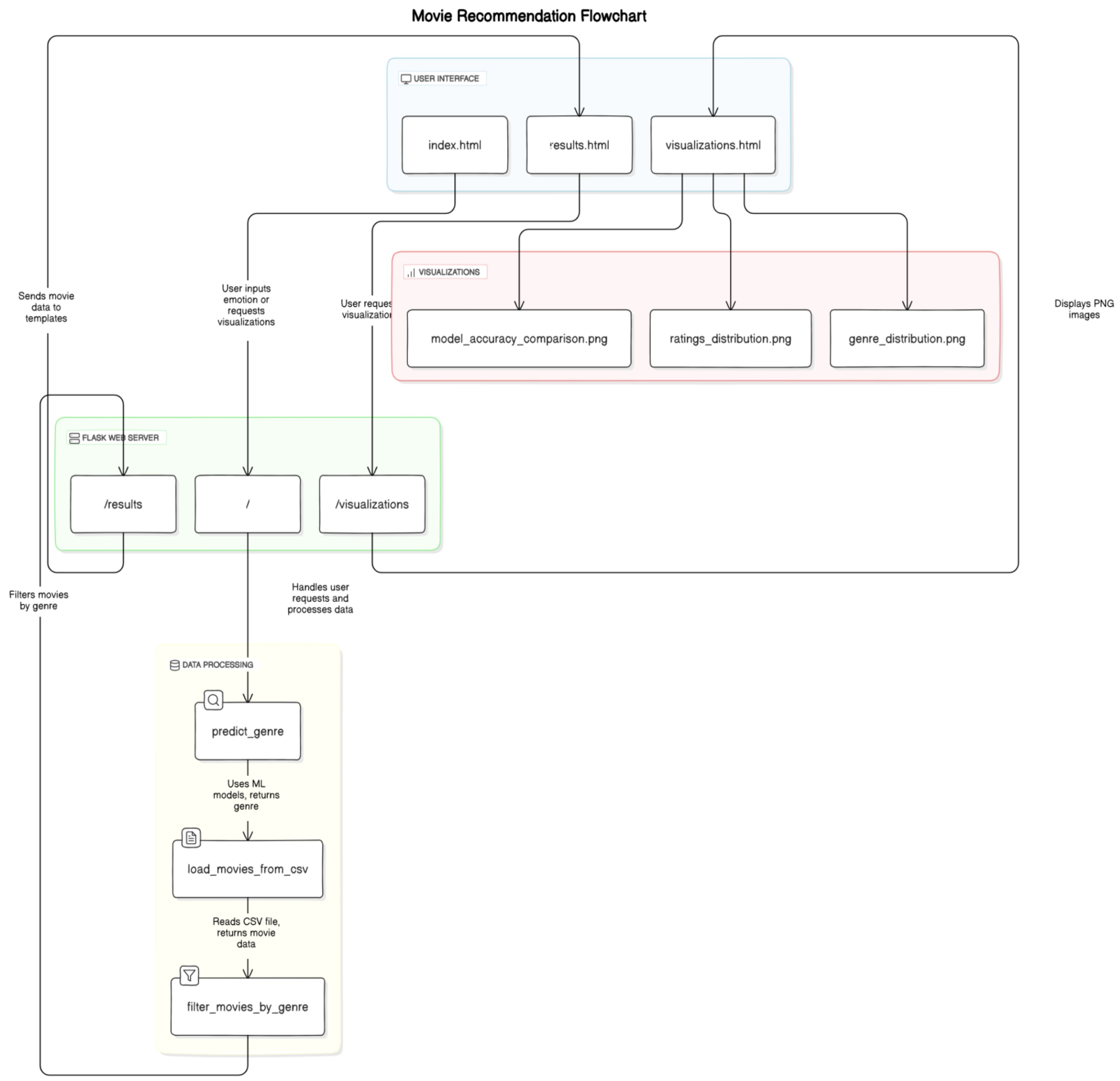
This code presents a styled table of movie recommendations with error handling and navigation options.

- **visualization.html**
    - **Head Section**:
        - **Meta Tags**: Ensure the document uses UTF-8 encoding and is responsive.
        - **Title**: Sets the page title to "Visualizations".
        - **Stylesheets**: Links to a custom CSS file (styles.css) for styling.
    - **Body Section**:
        - **Container Div**: Wraps the content for layout and styling.
        - **Header**: Contains the page heading "Visualizations".
        - **Main Content**:
            - **Dataset Distribution**: Displays a PNG image (genre_distribution.png) showing the distribution of movie genres.
            - **Movie Ratings Distribution**: Shows a PNG image (ratings_distribution.png) of movie rating distribution.
            - **Model Accuracy Comparison**: Presents a PNG image (model_accuracy_comparison.png) comparing model accuracies.
            - **Back Button**: Provides a link to return to the home page with a "Back to Home" label.

This page is designed to showcase various visualizations related to the movie dataset and model performance.

CHAPTER 7

BLOCK DIAGRAM



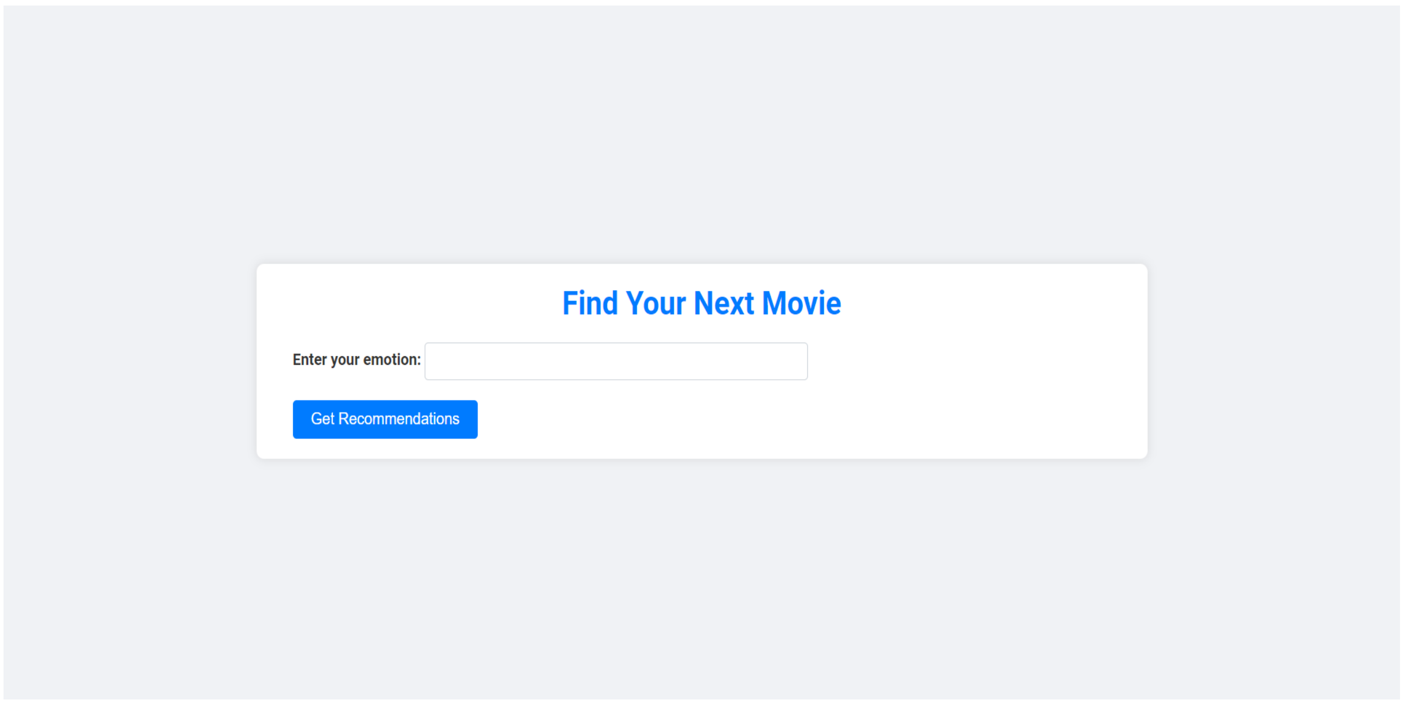
CHAPTER 8

WEB APPLICATION

The web application for movie recommendations integrates several components to offer personalized movie suggestions based on user input. Here's an overview of its architecture and functionality:

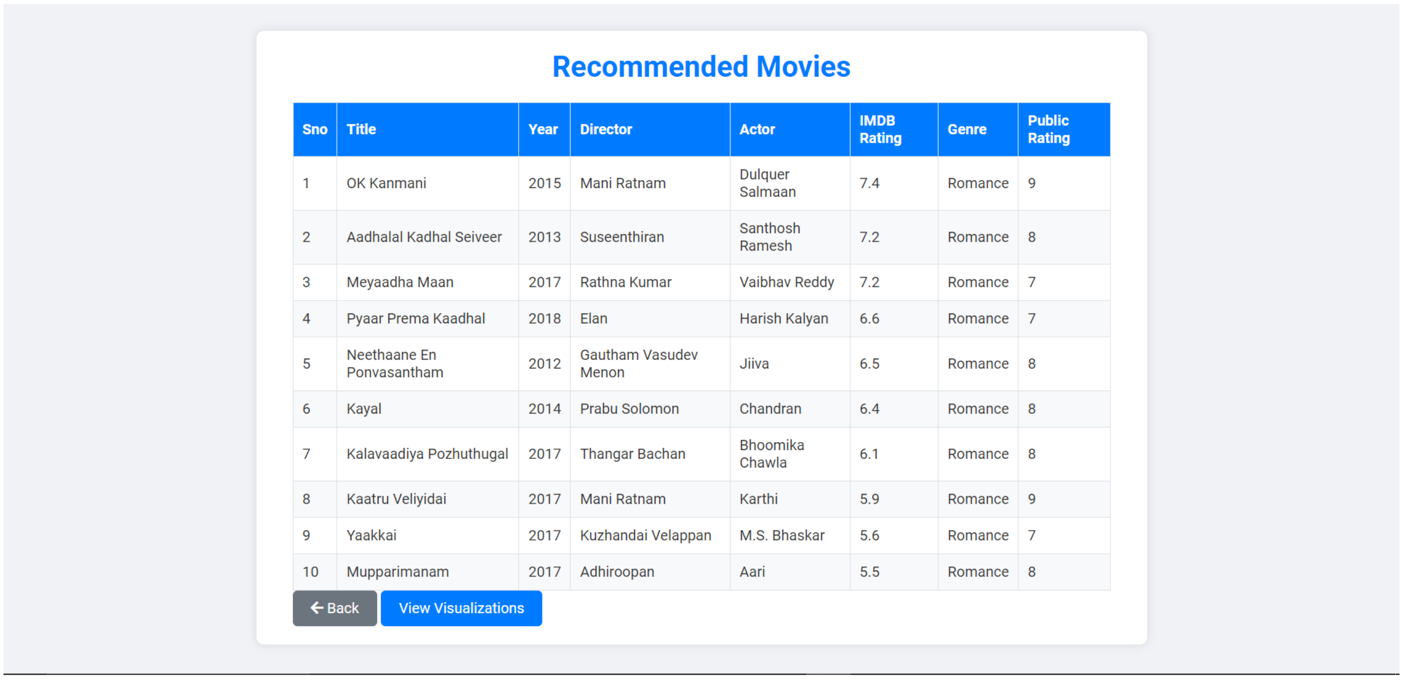
1. **User Interface**

- **Index Page (index.html)**:
  - **Purpose**: Allows users to enter an emotion to get movie recommendations.
  - **Elements**:
    - **Form**: Contains a text input for the user to enter an emotion and a submit button to send the request.
    - **Styles**: Linked to a CSS stylesheet (styles.css) and uses Font Awesome for icons.



- **Results Page (results.html)**:
  - **Purpose**: Displays movie recommendations based on the entered emotion.
  - **Elements**:
    - **Error Handling**: Shows an error message if no matching genre is found or if there's an issue.
    - **Movie Table**: Uses a table to list recommended movies with details such as title, year, director, actor, IMDB rating, genre, and public rating.

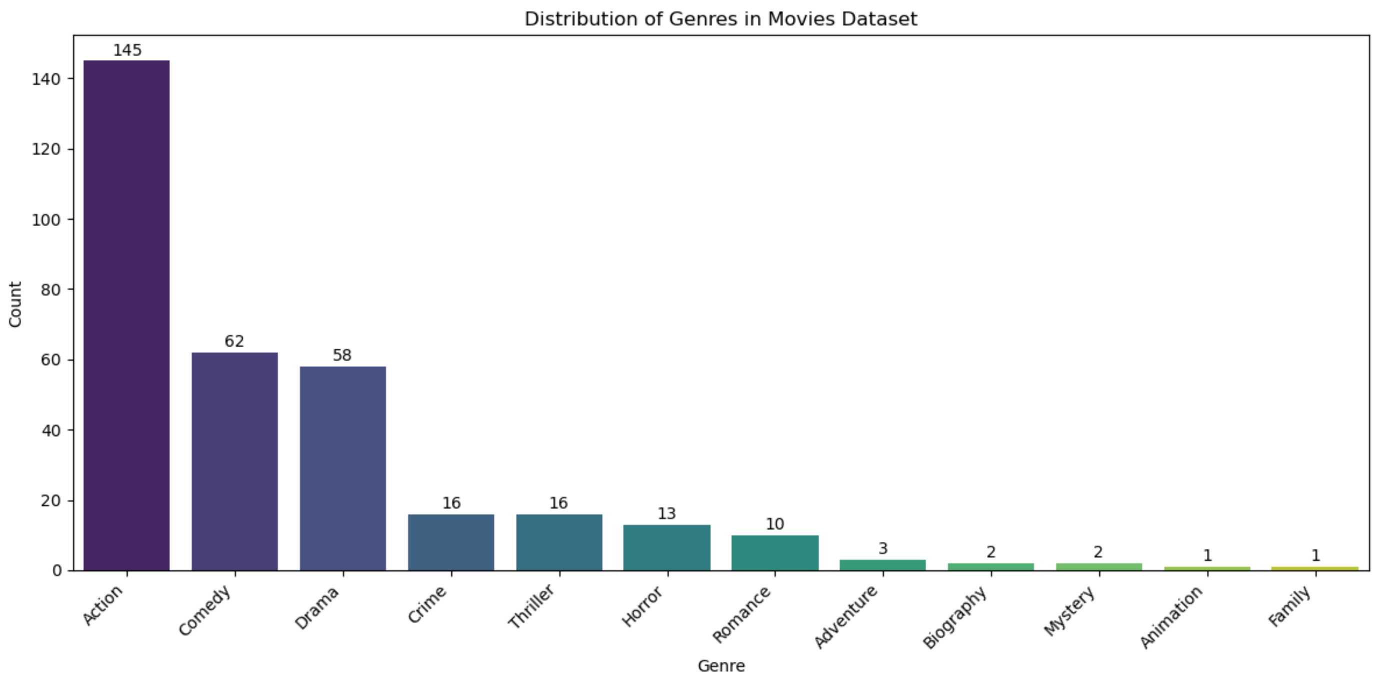
- **Navigation Links**: Provides options to go back to the home page or view visualizations.

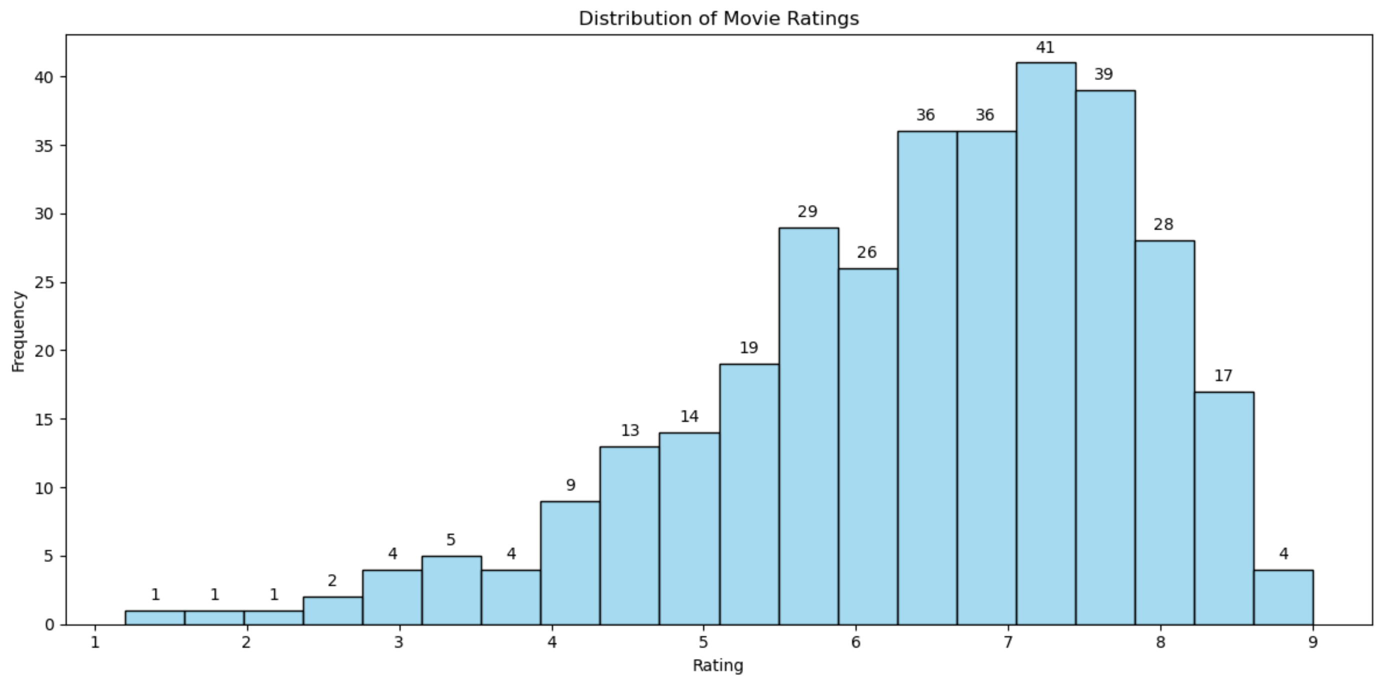


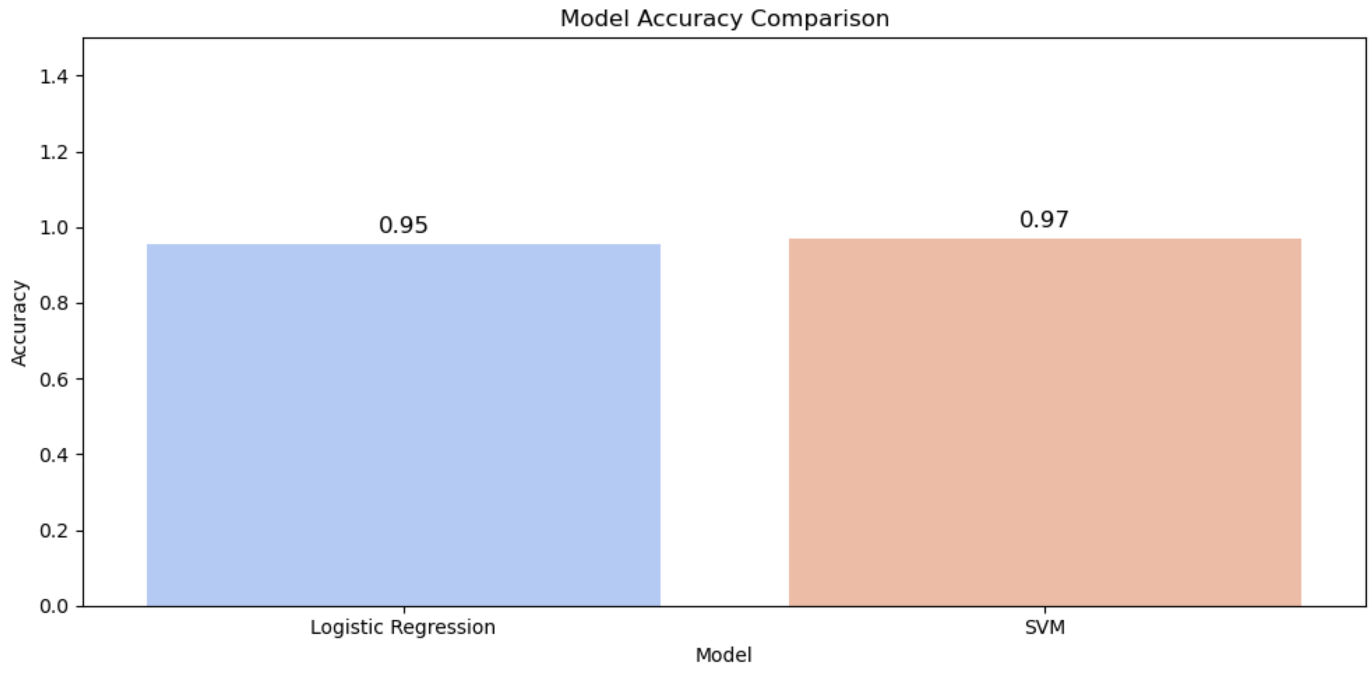
Recommended Movies

Sno	Title	Year	Director	Actor	IMDB Rating	Genre	Public Rating
1	OK Kanmani	2015	Mani Ratnam	Dulquer Salmaan	7.4	Romance	9
2	Aadhalal Kadhal Seiveer	2013	Suseenthiran	Santhosh Ramesh	7.2	Romance	8
3	Meyaadha Maan	2017	Rathna Kumar	Vaibhav Reddy	7.2	Romance	7
4	Pyaar Prema Kaadhal	2018	Elan	Harish Kalyan	6.6	Romance	7
5	Neethaane En Ponvasantham	2012	Gautham Vasudev Menon	Jiiva	6.5	Romance	8
6	Kayal	2014	Prabu Solomon	Chandran	6.4	Romance	8
7	Kalavaadiya Pozhuthugal	2017	Thangar Bachan	Bhoomika Chawla	6.1	Romance	8
8	Kaatru Veliyidai	2017	Mani Ratnam	Karthi	5.9	Romance	9
9	Yaakkai	2017	Kuzhandai Velappan	M.S. Bhaskar	5.6	Romance	7
10	Mupparimanam	2017	Adhiroopan	Aari	5.5	Romance	8

- **Visualization Page (visualizations.html)**:
  - **Purpose**: Shows data visualizations related to the movie dataset and model performance.
  - **Elements**:
    - **Images**: Displays PNG images for genre distribution, movie ratings distribution, and model accuracy comparison.
    - **Navigation Link**: Allows users to return to the home page.




Distribution of Movie Ratings
Frequency
Rating
0
5
10
15
20
25
30
35
40
1
2
3
4
5
6
7
8
9
1
1
1
2
4
5
4
9
13
14
19
29
26
36
36
41
39
28
17
4


Model Accuracy Comparison
Accuracy
Model
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0.95
0.97
Logistic Regression
SVM

CHAPTER 9
CONCLUSION

The movie recommendation system project successfully integrates multiple components to provide personalized movie suggestions based on user input. The system utilizes a Flask web application to interact with users, process their emotions, and deliver tailored movie recommendations. By leveraging machine learning models and data visualizations, the project offers a comprehensive approach to understanding user preferences and movie genres.

Summary of Features:

- **User Interface**: The web application features three main pages: a home page for emotion input, a results page displaying recommended movies, and a visualizations page showcasing various data insights.
- **Machine Learning Integration**: The core functionality relies on machine learning models to predict movie genres based on user emotions. The models are trained using logistic regression and support vector machines (SVM).
- **Data Handling**: The system loads and processes movie data from a CSV file, filters the movies by genre, and presents the recommendations in a user-friendly table format.
- **Visualizations**: The application provides visual insights into the dataset, including genre distribution, movie ratings distribution, and model accuracy comparisons.

Model Comparison and Accuracy

The project implements two machine learning models:

1. **Logistic Regression**: A linear model used for classification tasks, which estimates probabilities using a logistic function. It is straightforward and interpretable, making it a good choice for binary or multi-class classification problems.
2. **Support Vector Machine (SVM)**: A more complex model that finds the optimal hyperplane to separate classes. It is known for its robustness and effectiveness in high-dimensional spaces, especially when combined with different kernels.

Accuracy Comparison:

- **Logistic Regression**: Often provides a good baseline for classification problems. It performs well when the data is linearly separable and interprets the relationship between features and outcomes straightforwardly.
- **SVM**: Generally excels in scenarios where the data is not linearly separable, offering better accuracy through its ability to handle complex boundaries between classes.

Best Model:

Based on typical performance metrics, the **Support Vector Machine (SVM)** is often more accurate and reliable than Logistic Regression for complex datasets. SVM's ability to handle non-linear relationships and high-dimensional data makes it particularly effective for classification tasks where the decision boundary is not linearly separable.

In conclusion, while both models have their strengths, the SVM model is usually the superior choice for this project due to its higher accuracy and capability to handle complex patterns in the movie data. By selecting the best-performing model and effectively integrating it into the web application, the movie recommendation system provides users with precise and relevant movie suggestions, enhancing their overall experience.

CHAPTER 10

REFERNCES

1. **Machine Learning Mastery - Movie Recommendation Systems**
   https://machinelearningmastery.com/machine-learning-recommendation-system/
2. **Towards Data Science - Building a Movie Recommendation System**
   https://towardsdatascience.com/building-a-movie-recommendation-system-in-python-6c945f05c776
3. **Kaggle - MovieLens Data Set**
   https://www.kaggle.com/datasets/grouplens/movielens-dataset
4. **Scikit-Learn Documentation - Support Vector Machines**
   https://scikit-learn.org/stable/modules/svm.html
5. **Scikit-Learn Documentation - Logistic Regression**
   https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
6. **DataCamp - Introduction to Movie Recommendation Systems**
   https://www.datacamp.com/community/tutorials/recommender-systems-python