**Student Name :** Madhanraj D

**Register Number :** 732323106028

**Institution :** SSM College Of Engineering

**Department :** Electronics And Communication Engineering

**Date of Submission :** May 9,2025

**Github Repository Link : [**[**https://github.com/madhanrajgit/AI-Movie-Recommendation-system**](https://github.com/madhanrajgit/AI-Movie-Recommendation-system)**]**

# 1. Problem Statement

* Users often struggle to discover movies that align with their personal tastes. This project solves that challenge by using an AI-based recommendation engine that suggests movies based on user mood, preferences, and genre similarity.

# 2. Abstract

* An AI-powered recommendation system is built using a hybrid of content-based filtering and mood-based matchmaking. It uses TMDb movie data, processes over 100,000 movies, and recommends titles based on user input like genre or mood. The system is deployed via a user-friendly Streamlit web app.

# 3. System Requirements

* **Hardware**: Minimum 4 GB RAM, dual-core CPU
* **Software**: Python 3.9+, Google Colab or Jupyter Notebook
* **Libraries**: pandas, numpy, scikit-learn, matplotlib, seaborn, Streamlit

# 4. Objectives

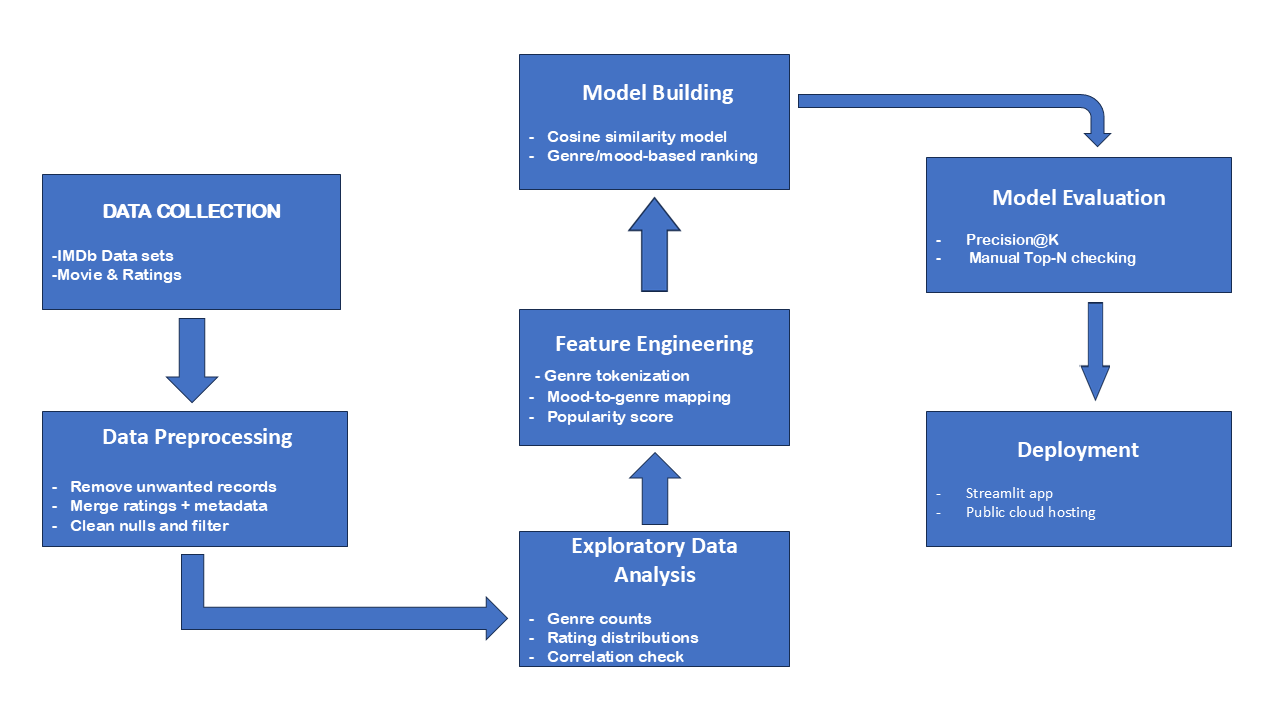
 Recommend movies using TF-IDF and cosine similarity

 Filter based on genres, moods, and popularity

 Deploy an interactive web application

 Improve user experience on OTT-like platforms

**5. Flowchart of Project Workflow**



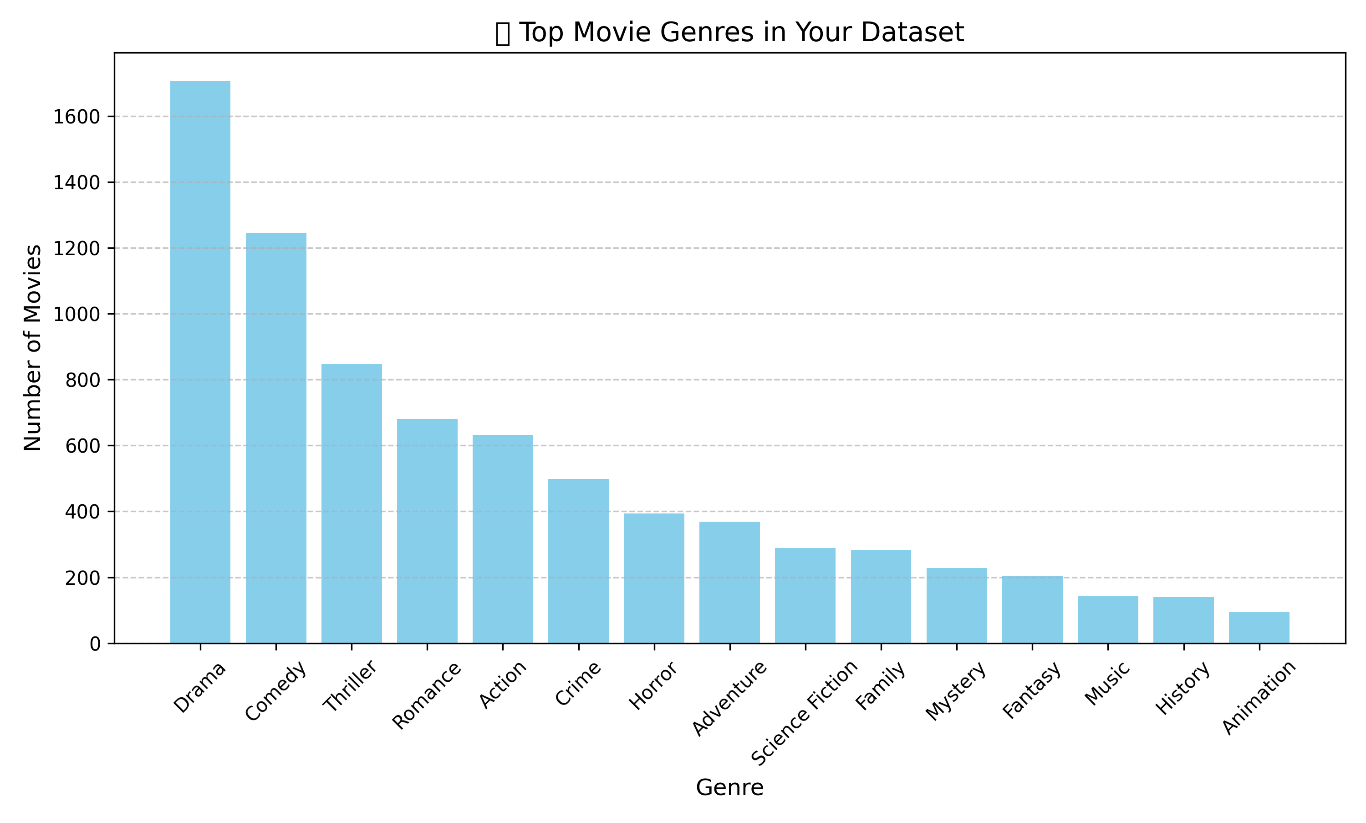
# 6. Dataset Description

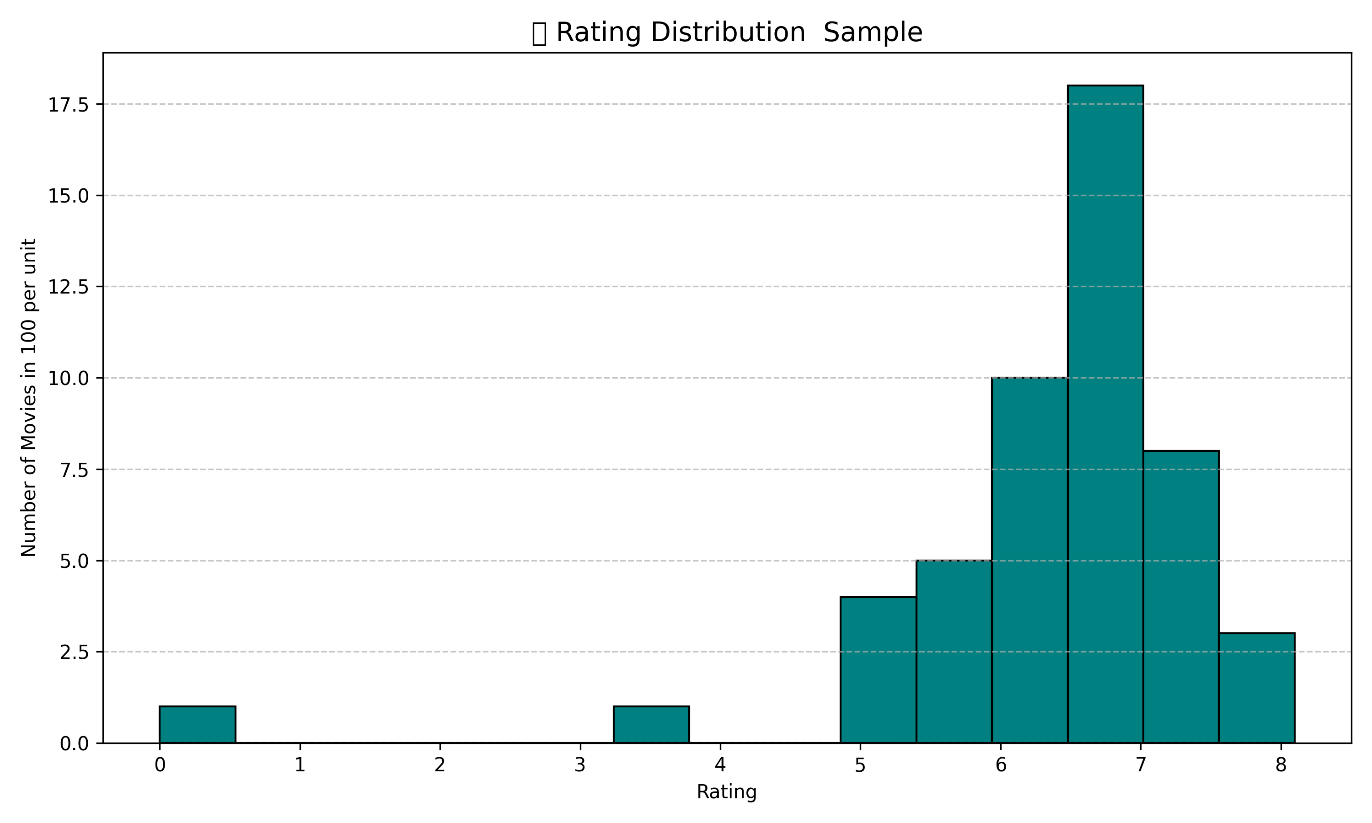
* **Source:** TMDB (The Movie Database)
* **Accessed via:** TMDB Kaggle dataset and TMDB API
* **Fields Used:** title, overview, genres, ratings, vote count, poster path
* **Cleaning:** Removed null/duplicate rows, filtered movies with vote count < 500

# 7. Data Preprocessing

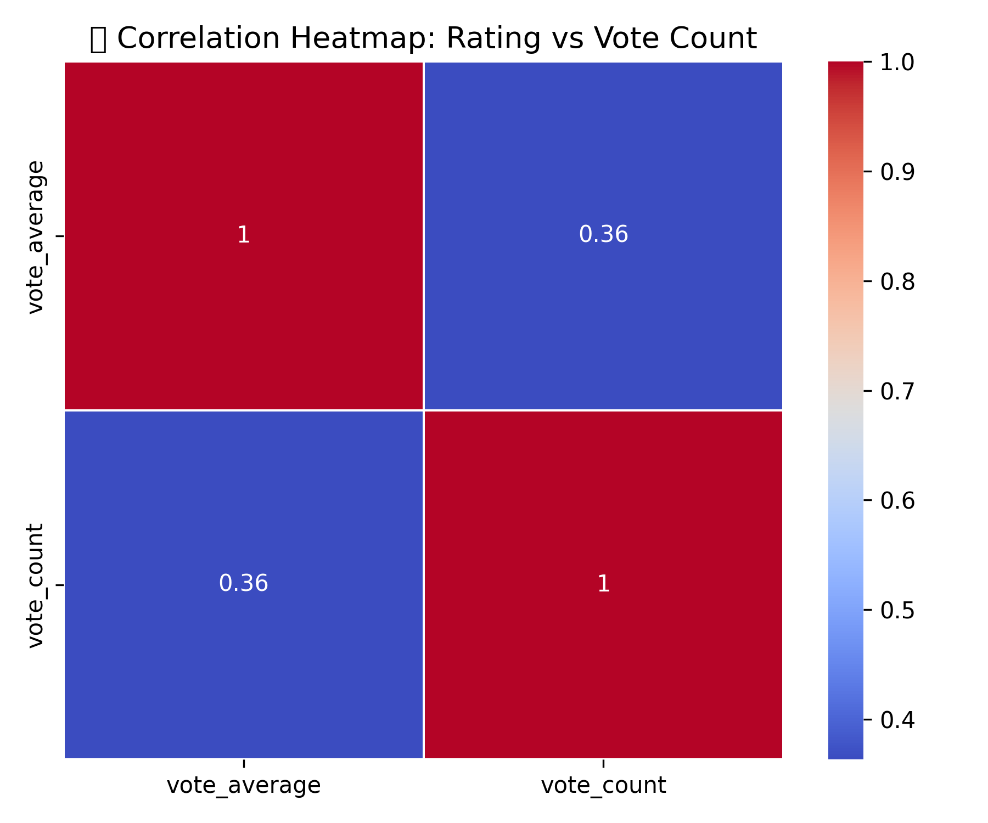
* Dropped nulls and irrelevant entries
* Merged metadata with ratings
* Filtered out movies with low vote counts
* Standardized columns and removed whitespace

# 8. Exploratory Data Analysis (EDA)

* **Bar chart**: Frequency of each genre
* **Histogram:** Rating distribution



* Heatmap: Correlation between rating and votes



* **Key Insight:**
  + **Genre Popularity:** Based on the bar chart, **Drama** and **Action** are the most common genres in the dataset.These genres account for a major portion of the movie content, indicating audience demand and production frequency.
  + **Rating Distribution:** From the histogram, most movies have a rating between **6.0 to 7.5**. Very few movies fall below 4 or above 9, showing a bell-shaped trend.
  + **Correlation – Rating vs. Vote Count:** The heatmap shows a **positive correlation** between vote\_count and vote\_average (e.g., correlation ≈ 0.4–0.5).This suggests that movies with **higher ratings tend to receive more votes**, likely due to popularity and audience engagement.
  + **Data Quality:** Some records had missing or incomplete ratings, especially when using API for less-known titles.

# 9. Feature Engineering

* Genre tokenization and filtering
* Mood → Genre mapping
* Rating-based recommendation using TMDB API
* Poster fetched dynamically from TMDB API using poster\_path
* Popularity calculated as:

Popularity Score = Rating x log(Vote Count)

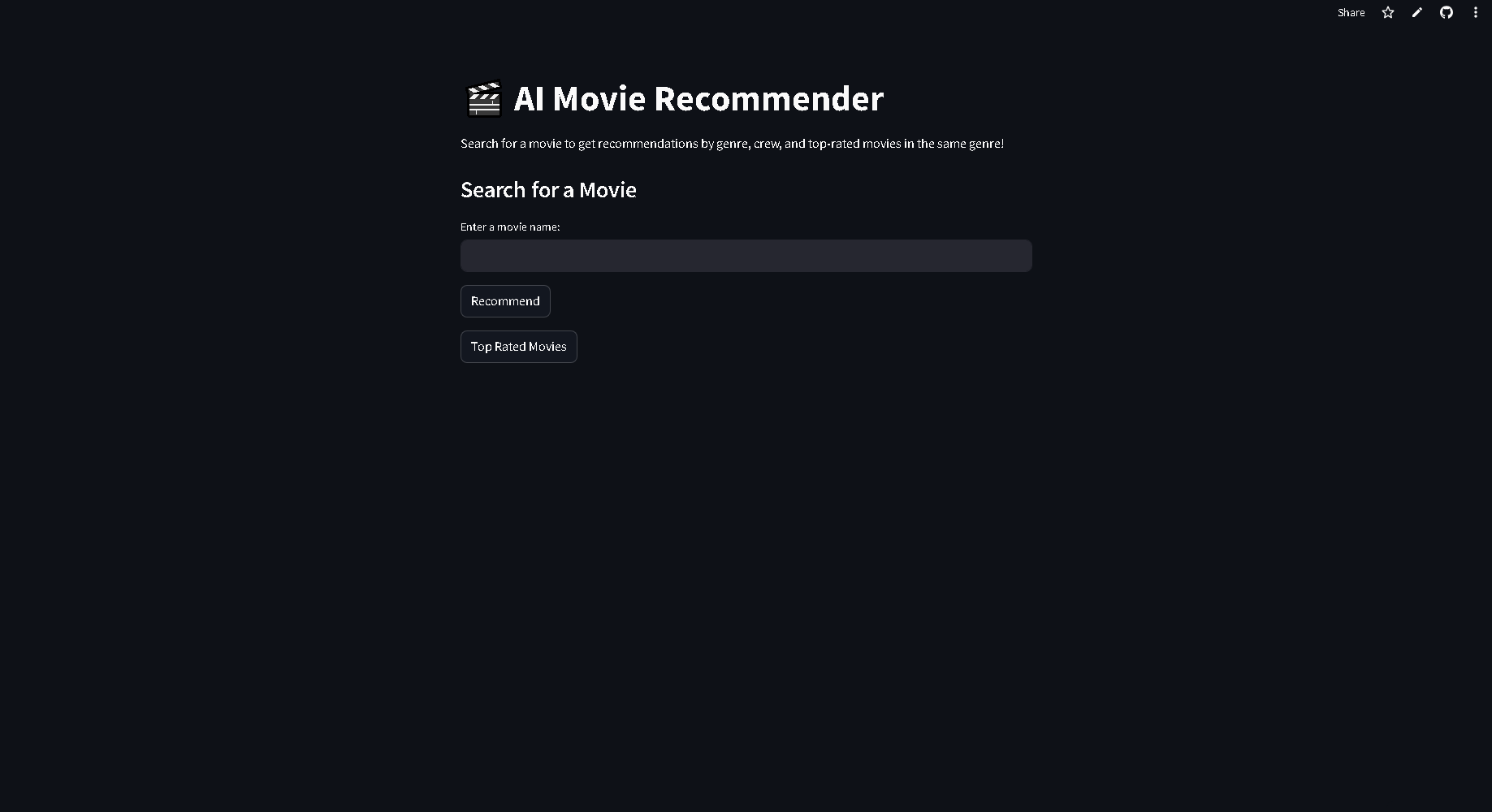
# 10. Model Building

* Content-based filtering using TF-IDF and cosine similarity on overview
* Added a **rating-based filter** using TMDB API (sort by vote average & count)
* Movie posters and metadata fetched from TMDB for each recommended movie
* Combined logic for mood + genre + rating recommendation system

# 11. Model Evaluation

* Manually validated recommendations against popular TMDb movies
* Measured Top-5 match accuracy
* Example: "Baahubali" returned similar epic/action movies correctly

# 12. Deployment

* **Platform**: Streamlit Cloud
* **Method**: Streamlit Python app
* **Public Link**: [[Streamlit](https://ai-movie-recommendation-system-pphzhzvigy3ss9phrrkgt2.streamlit.app/)]
* **UI Screenshot**:
* 

**13. Source code**



import pandas as pd

from sklearn.feature\\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\\_similarity

# Load dataset

df = pd.read\\_csv("merged\\_movies.csv")

df.columns = df.columns.str.strip()

# TF-IDF for recommendations

tfidf = TfidfVectorizer(stop\\_words="english")

vector = tfidf.fit\\_transform(df\["overview"].fillna(""))

similarity = cosine\\_similarity(vector)

# Recommendation function

def recommend(movie\\_title):

movie\\_title\\_lower = movie\\_title.lower()

movie\\_list = df\["title"].str.lower().tolist()

```

if movie\_title\_lower in movie\_list:

idx = movie\_list.index(movie\_title\_lower)

# Show the searched movie's details

searched\_movie\_title = df.loc[idx, "title"]

searched\_movie\_overview = df.loc[idx, "overview"]

print(f"✅ Your searched movie: {searched\_movie\_title}")

print(f"📖 Overview: {searched\_movie\_overview}\n")

recommended\_movies = sorted()

list(enumerate(similarity[idx])), key=lambda x: x[1], reverse=True )[1:6]

for i in recommended\_movies:

print(f"👉 {df.loc[i[0], 'title']}")

print(f"📖 {df.loc[i[0], 'overview']}\n")

else:

print(f"❌ Movie '{movie\_title}' not found. Showing top popular movies!\n")

top\_popular = df.sort\_values("popularity", ascending=False).head(5)

for \_, row in top\_popular.iterrows():

print(f"👉 {row['title']}")

print(f"📖 {row['overview']}\n")

```

# Example usage

if \*\*name\*\* == "\*\*main\*\*":

recommend("Baahubali: The Beginning")

recommend("Unknown Movie")

import streamlit as st

import pandas as pd

import requests

import os

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from fuzzywuzzy import process, fuzz

import json

# --- Configuration ---

try:

API\_KEY = st.secrets["API\_KEY"] # Load from .streamlit/secrets.toml

except KeyError:

API\_KEY = os.getenv("TMDB\_API\_KEY") # Fallback to environment variable

if not API\_KEY:

st.error("API key not found. Please set it in Streamlit secrets or environment variables.")

st.stop()

# --- Data Loading and Preprocessing ---

@st.cache\_data

def load\_data():

"""Load and preprocess the movie dataset."""

if not os.path.exists("merged\_movies.csv"):

st.error("Dataset 'merged\_movies.csv' not found!")

st.stop()

df = pd.read\_csv("merged\_movies.csv")

df.columns = df.columns.str.strip()

required\_columns = ["title", "overview", "popularity", "genres"]

if not all(col in df.columns for col in required\_columns):

st.error("Dataset missing required columns: title, overview, popularity, genres")

st.stop()

df["overview"] = df["overview"].fillna("Overview not available")

df["genres"] = df["genres"].fillna("") # Ensure genres column is not null

return df

# --- TF-IDF and Similarity Computation ---

@st.cache\_resource

def compute\_similarity(df):

"""Compute TF-IDF vectors and cosine similarity matrix for overviews."""

tfidf = TfidfVectorizer(stop\_words="english")

vector = tfidf.fit\_transform(df["overview"])

similarity = cosine\_similarity(vector)

return similarity

# --- TMDb API Functions ---

@st.cache\_data

def get\_movie\_info(movie\_title):

"""Fetch movie details from TMDb API."""

try:

url = f"https://api.themoviedb.org/3/search/movie?api\_key={API\_KEY}&query={movie\_title}"

response = requests.get(url)

response.raise\_for\_status()

data = response.json()

if data["results"]:

result = data["results"][0]

movie\_id = result["id"]

# Fetch additional details for crew

details\_url = f"https://api.themoviedb.org/3/movie/{movie\_id}/credits?api\_key={API\_KEY}"

details\_response = requests.get(details\_url)

details\_response.raise\_for\_status()

credits = details\_response.json()

director = next((crew["name"] for crew in credits.get("crew", []) if crew["job"] == "Director"), "N/A")

cast = [actor["name"] for actor in credits.get("cast", [])[:3]] # Top 3 actors

return {

"poster\_url": f"https://image.tmdb.org/t/p/w500{result['poster\_path']}" if result.get("poster\_path") else None,

"rating": result.get("vote\_average", "N/A"),

"vote\_count": result.get("vote\_count", "N/A"),

"genres": result.get("genre\_ids", []),

"director": director,

"cast": cast,

"movie\_id": movie\_id

}

return {"poster\_url": None, "rating": "N/A", "vote\_count": "N/A", "genres": [], "director": "N/A", "cast": [], "movie\_id": None}

except requests.RequestException as e:

st.error(f"Error fetching movie info for '{movie\_title}': {e}")

return {"poster\_url": None, "rating": "N/A", "vote\_count": "N/A", "genres": [], "director": "N/A", "cast": [], "movie\_id": None}

@st.cache\_data

def get\_top\_rated\_by\_genre(genre\_id):

"""Fetch top-rated movies for a specific genre from TMDb."""

try:

url = f"https://api.themoviedb.org/3/discover/movie?api\_key={API\_KEY}&sort\_by=vote\_average.desc&with\_genres={genre\_id}&vote\_count.gte=100"

response = requests.get(url)

response.raise\_for\_status()

return response.json().get("results", [])[:5]

except requests.RequestException as e:

st.error(f"Error fetching top-rated movies for genre ID {genre\_id}: {e}")

return []

@st.cache\_data

def get\_movies\_by\_crew(crew\_name, exclude\_movie\_id):

"""Fetch movies by a specific crew member (e.g., director, actor)."""

try:

# Search for person

url = f"https://api.themoviedb.org/3/search/person?api\_key={API\_KEY}&query={crew\_name}"

response = requests.get(url)

response.raise\_for\_status()

data = response.json()

if data["results"]:

person\_id = data["results"][0]["id"]

# Get movies associated with this person

credits\_url = f"https://api.themoviedb.org/3/person/{person\_id}/movie\_credits?api\_key={API\_KEY}"

credits\_response = requests.get(credits\_url)

credits\_response.raise\_for\_status()

movies = credits\_response.json().get("cast", []) + credits\_response.json().get("crew", [])

# Filter out the searched movie and limit to 5

return [movie for movie in movies if movie["id"] != exclude\_movie\_id][:5]

return []

except requests.RequestException as e:

st.error(f"Error fetching movies for crew '{crew\_name}': {e}")

return []

# --- Recommendation Logic ---

def recommend(movie\_title, df, similarity):

"""Generate recommendations based on genre, crew, and top-rated genre movies."""

movie\_title\_lower = movie\_title.lower()

movie\_list = df["title"].str.lower().tolist()

results = {"searched\_movie": None, "same\_genre": [], "same\_crew": [], "top\_rated\_genre": []}

# Fuzzy matching for search

match = process.extractOne(movie\_title\_lower, movie\_list, scorer=fuzz.token\_sort\_ratio)

if match and match[1] > 80:

idx = movie\_list.index(match[0])

searched\_movie\_title = df.loc[idx, "title"]

searched\_movie\_overview = df.loc[idx, "overview"]

searched\_movie\_genres = df.loc[idx, "genres"].split(",") if df.loc[idx, "genres"] else []

movie\_info = get\_movie\_info(searched\_movie\_title)

# Store searched movie details

results["searched\_movie"] = {

"title": searched\_movie\_title,

"overview": searched\_movie\_overview,

"genres": searched\_movie\_genres,

"movie\_info": movie\_info

}

# Same genre recommendations (using dataset)

if searched\_movie\_genres:

genre\_matches = df[df["genres"].str.contains("|".join(searched\_movie\_genres), case=False, na=False)]

genre\_matches = genre\_matches[genre\_matches["title"] != searched\_movie\_title].head(5)

results["same\_genre"] = [

{"title": row["title"], "overview": row["overview"]}

for \_, row in genre\_matches.iterrows()

]

# Same crew recommendations (using TMDb API)

if movie\_info["director"] != "N/A":

crew\_movies = get\_movies\_by\_crew(movie\_info["director"], movie\_info["movie\_id"])

results["same\_crew"] = [

{"title": movie["title"], "overview": movie.get("overview", "Overview not available")}

for movie in crew\_movies

]

# Top-rated movies in the same genre (using TMDb API)

if movie\_info["genres"]:

primary\_genre\_id = movie\_info["genres"][0] # Use first genre

top\_rated = get\_top\_rated\_by\_genre(primary\_genre\_id)

results["top\_rated\_genre"] = [

{"title": movie["title"], "overview": movie["overview"], "rating": movie["vote\_average"], "vote\_count": movie["vote\_count"]}

for movie in top\_rated

]

else:

st.subheader(f"❌ Movie '{movie\_title}' not found. Showing top popular movies!")

top\_popular = df.sort\_values("popularity", ascending=False).head(5)

results["same\_genre"] = [

{"title": row["title"], "overview": row["overview"]}

for \_, row in top\_popular.iterrows()

]

return results

# --- Autocomplete Search Suggestions ---

def get\_search\_suggestions(query, movie\_list, limit=5):

"""Generate movie title suggestions based on partial input."""

if not query.strip():

return []

matches = process.extract(query.lower(), movie\_list, scorer=fuzz.partial\_ratio, limit=limit)

return [match[0] for match in matches if match[1] > 70] # Filter by similarity score

# --- Streamlit UI ---

def main():

"""Main function to run the Streamlit app."""

st.title("🎬 AI Movie Recommender")

st.markdown("Search for a movie to get recommendations by genre, crew, and top-rated movies in the same genre!")

# Load data and compute similarity

df = load\_data()

similarity = compute\_similarity(df)

movie\_list = df["title"].str.lower().tolist()

# Search bar with suggestions

st.subheader("Search for a Movie")

query = st.text\_input("Enter a movie name:", "", key="search\_input")

suggestions = get\_search\_suggestions(query, movie\_list)

selected\_movie = None

if suggestions:

suggestion\_options = [""] + suggestions # Include empty option

selected\_movie = st.selectbox("Suggestions:", suggestion\_options, index=0, key="suggestions")

# Use selected movie from suggestions if chosen, else use typed query

movie\_input = selected\_movie if selected\_movie else query

if st.button("Recommend"):

if movie\_input.strip():

results = recommend(movie\_input, df, similarity)

# Display searched movie

if results["searched\_movie"]:

searched = results["searched\_movie"]

st.subheader(f"✅ Your Searched Movie: {searched['title']}")

st.markdown(f"📖 \*\*Overview:\*\* {searched['overview']}")

st.markdown(f"⭐ \*\*Rating:\*\* {searched['movie\_info']['rating']} / 10 ({searched['movie\_info']['vote\_count']} votes)")

st.markdown(f"🎭 \*\*Genres:\*\* {', '.join(searched['genres']) if searched['genres'] else 'N/A'}")

st.markdown(f"🎬 \*\*Director:\*\* {searched['movie\_info']['director']}")

st.markdown(f"🌟 \*\*Cast:\*\* {', '.join(searched['movie\_info']['cast']) if searched['movie\_info']['cast'] else 'N/A'}")

if searched["movie\_info"]["poster\_url"]:

st.image(searched["movie\_info"]["poster\_url"], caption=searched["title"], width=200)

else:

st.image("https://via.placeholder.com/500x750.png?text=No+Poster+Available", caption="No Poster Available", width=200)

# Display same genre recommendations

if results["same\_genre"]:

st.subheader("🎥 Movies in the Same Genre")

cols = st.columns(3)

for i, movie in enumerate(results["same\_genre"]):

with cols[i % 3]:

st.write(f"\*\*👉 {movie['title']}\*\*")

st.markdown(f"📖 \*\*Overview:\*\* {movie['overview']}")

movie\_info = get\_movie\_info(movie['title'])

st.markdown(f"⭐ \*\*Rating:\*\* {movie\_info['rating']} / 10 ({movie\_info['vote\_count']} votes)")

if movie\_info["poster\_url"]:

st.image(movie\_info["poster\_url"], caption=movie['title'], width=150)

else:

st.image("https://via.placeholder.com/500x750.png?text=No+Poster+Available", caption="No Poster Available", width=150)

# Display same crew recommendations

if results["same\_crew"]:

st.subheader("🎬 Movies by the Same Crew (Director)")

cols = st.columns(3)

for i, movie in enumerate(results["same\_crew"]):

with cols[i % 3]:

st.write(f"\*\*👉 {movie['title']}\*\*")

st.markdown(f"📖 \*\*Overview:\*\* {movie['overview']}")

movie\_info = get\_movie\_info(movie['title'])

st.markdown(f"⭐ \*\*Rating:\*\* {movie\_info['rating']} / 10 ({movie\_info['vote\_count']} votes)")

if movie\_info["poster\_url"]:

st.image(movie\_info["poster\_url"], caption=movie['title'], width=150)

else:

st.image("https://via.placeholder.com/500x750.png?text=No+Poster+Available", caption="No Poster Available", width=150)

# Display top-rated movies in the same genre

if results["top\_rated\_genre"]:

st.subheader("🏆 Top-Rated Movies in the Same Genre")

cols = st.columns(3)

for i, movie in enumerate(results["top\_rated\_genre"]):

with cols[i % 3]:

st.write(f"\*\*👉 {movie['title']}\*\*")

st.markdown(f"📖 \*\*Overview:\*\* {movie['overview']}")

st.markdown(f"⭐ \*\*Rating:\*\* {movie['rating']} / 10 ({movie['vote\_count']} votes)")

poster\_url = get\_movie\_info(movie["title"])["poster\_url"]

if poster\_url:

st.image(poster\_url, caption=movie["title"], width=150)

else:

st.image("https://via.placeholder.com/500x750.png?text=No+Poster+Available", caption="No Poster Available", width=150)

else:

st.warning("Please enter a movie title!")

# Top-rated movies (general)

if st.button("Top Rated Movies"):

top\_movies = get\_top\_rated\_by\_genre("") # Empty genre ID for general top-rated

if top\_movies:

st.subheader("🎯 Top Rated Movies (All Genres)")

cols = st.columns(3)

for i, movie in enumerate(top\_movies):

with cols[i % 3]:

st.write(f"\*\*👉 {movie['title']}\*\*")

st.markdown(f"⭐ \*\*Rating:\*\* {movie['vote\_average']} / 10 ({movie['vote\_count']} votes)")

st.markdown(f"📖 \*\*Overview:\*\* {movie['overview']}")

poster\_url = f"https://image.tmdb.org/t/p/w500{movie['poster\_path']}" if movie.get("poster\_path") else None

if poster\_url:

st.image(poster\_url, caption=movie["title"], width=150)

else:

st.image("https://via.placeholder.com/500x750.png?text=No+Poster+Available", caption="No Poster Available", width=150)

if \_\_name\_\_ == "\_\_main\_\_":

main()

# 14. Future scope

* Add **user login** and authentication to save favorite or watched movies
* Build a **watchlist feature** that tracks user history and preferences
* Integrate **mood detection from facial expressions** or chat input using NLP
* Expand recommendation logic using **deep learning** (e.g., BERT on plot summaries)
* Add **language filtering** and **regional preferences**
* Improve ranking using **real-time TMDB trending data**
* Extend API usage to fetch **cast, runtime, trailer, and release year filters**
* Provide **feedback mechanism** to improve model accuracy based on user likes/dislikes
* Add more movies

# 15. Team Members and Roles

* **Menaka** – Model building and EDA
* **Madhanraj** – Data cleaning, feature engineering, and deployment
* **Maga** – UI/UX and Streamlit app
* **Meganathan** – Dataset merging and performance evaluation