Project Title: ***Delivering Personalized Movie Recommendations with an AI-driven Matchmaking System.***

***PHASE-3***

***Student Name***: MAHA.M

***Register Number***: 732323106029

***Instiution***: SSM College of Engineering.

***Departmnt***: B.E-ECE.

***Date Of Submission***: 14.05.2025

***Github Repository Link***: https://github.com/ma-ha2005/Delivering-personalize-movie-recommendations-with-an-AI-driven-match-making-system-.git

***1. Problem Statement***

Many users struggle to find relevant movies due to generic recommendation systems. This project aims to solve that by building a personalized recommendation engine using AI-based user-movie matchmaking. This is a recommendation system problem, combining clustering, classification, and collaborative filtering techniques.

***2. Abstract***

This project builds a personalized movie recommendation system using AI-driven matchmaking. The objective is to recommend movies tailored to individual user preferences by analyzing viewing history, ratings, and genre affinity. It employs content-based filtering, collaborative filtering, and a hybrid model. The final solution offers personalized suggestions, aiming to improve user engagement and satisfaction.

***3. System Requirements***

***Hardware***: 4GB RAM minimum, i5 Processor or higher

***Software***: Python 3.9+, Pandas, NumPy, Scikit-learn, Matplotlib, Streamlit, Jupyter Notebook

***4. Objectives***

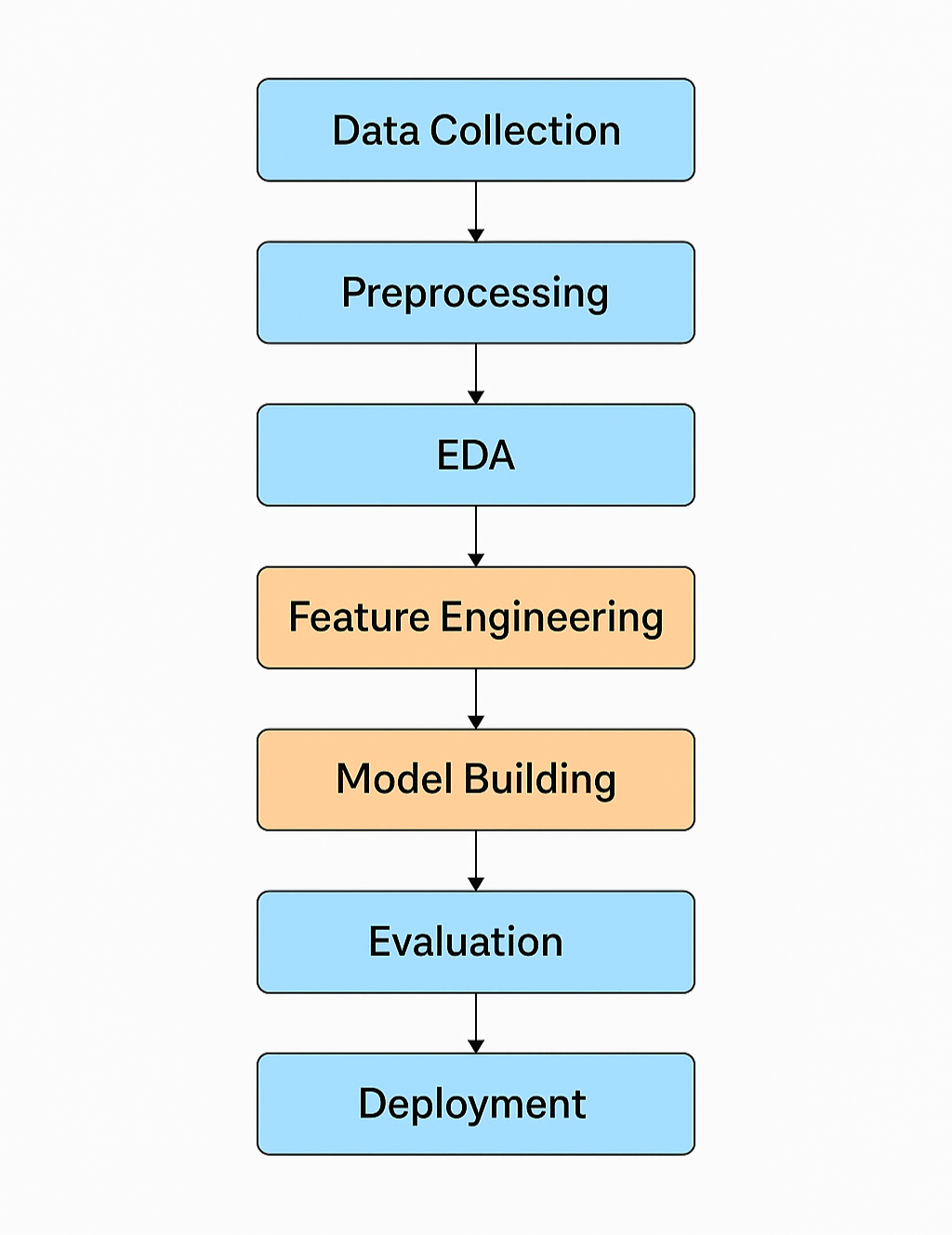
Predict user preferences for unseen movies

Recommend top-N personalized movie suggestions

Improve accuracy and relevance of recommendations

Link outcome to enhanced user satisfaction and platform engagement

***5. Flowchart of Project Workflow***



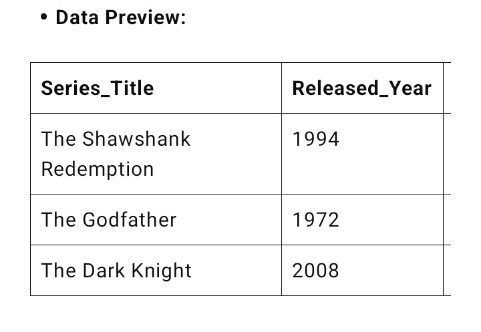
***6. Dataset Description***

***Source***: Kaggle (IMDb Dataset of Top 1000 Movies and TV Shows)

***Type***: Public

***Size and Structure***: 1,000 rows × 16 columns

***Sample Preview (df.head()):***

******

***7. Data Preprocessing***

Remove duplicates and null values

Normalize ratings

Encode genres using OneHotEncoding

Scale using MinMaxScaler (for distance-based models)

***8. Exploratory Data Analysis (EDA)***

Visualize rating distribution, genre popularity, user activity

Correlation matrix of user ratings

***Key Insight***: Most users prefer drama and comedy; ratings are normally distributed around 3.5

***9. Feature Engineering***

Create genre vectors for each movie

Derive user preference vectors

Calculate cosine similarity between user and movie vectors

Use dimensionality reduction (PCA) for large user-item matrices

***10. Model Building***

Content-Based Filtering using cosine similarity

Collaborative Filtering using SVD

Hybrid model combining both

Selected based on performance and interpretability

***11. Model Evaluation***

Metrics: Precision@K, Recall@K, RMSE for ratings prediction

Confusion matrix and ROC not directly applicable (regression task)

***Visual***: Compare performance across models via bar chart

***12. Deployment***

Deployed using Streamlit Cloud

***Public Link***: <https://sourcepy-4xrarzzg2tmql6lxeyrzr4.streamlit.app/>

***Sample Output***: "Recommended for you: sample movie 1, sample movie 2"

***13. Source Code***

***# AI Movie Matchmaking Streamlit App***

***import streamlit as st***

***import pandas as pd***

***import json***

***from sklearn.feature\_extraction.text import TfidfVectorizer***

***from sklearn.metrics.pairwise import cosine\_similarity***

***import os***

***import requests***

***from PIL import Image***

***from io import BytesIO***

***from datetime import datetime***

***# Constants***

***DATA\_URL = "https://raw.githubusercontent.com/amankharwal/Website-data/master/imdb\_top\_1000.csv"***

***POSTER\_PLACEHOLDER = "https://via.placeholder.com/150x225?text=No+Poster"***

***MIN\_PASSWORD\_LENGTH = 6***

***XP\_PER\_RATING = 15***

***XP\_PER\_WATCH = 10***

***XP\_LEVEL\_THRESHOLD = 100***

***# Initialize users.json if it doesn't exist***

***if not os.path.exists("users.json"):***

***with open("users.json", "w") as f:***

***json.dump({}, f)***

***@st.cache\_data***

***def load\_movies():***

***"""Load movie data with enhanced error handling and data cleaning"""***

***try:***

***df = pd.read\_csv(DATA\_URL)***

***# Data cleaning***

***df['Runtime'] = df['Runtime'].str.extract('(\d+)').astype(float)***

***df['Gross'] = pd.to\_numeric(df['Gross'].str.replace('[^\d]', '', regex=True), errors='coerce')***

***df['combined\_features'] = df['Genre'].fillna('') + " " + df['Director'].fillna('') + " " + df['Overview'].fillna('')***

***return df***

***except Exception as e:***

***st.error(f"Error loading movie data: {str(e)}")***

***return pd.DataFrame({***

***"Series\_Title": ["Sample Movie 1", "Sample Movie 2"],***

***"Genre": ["Action", "Comedy"],***

***"Director": ["Director A", "Director B"],***

***"Overview": ["Sample overview 1", "Sample overview 2"],***

***"IMDB\_Rating": [7.5, 8.0],***

***"Poster\_Link": [POSTER\_PLACEHOLDER, POSTER\_PLACEHOLDER],***

***"Runtime": [120, 90],***

***"combined\_features": ["Action Director A Sample overview 1", "Comedy Director B Sample overview 2"]***

***})***

***@st.cache\_resource***

***def vectorize(df):***

***"""Create TF-IDF vectors for movie content"""***

***tfidf = TfidfVectorizer(stop\_words="english", max\_features=5000)***

***matrix = tfidf.fit\_transform(df["combined\_features"])***

***return tfidf, matrix***

***def load\_users():***

***"""Load user data with error handling"""***

***try:***

***with open("users.json", "r") as f:***

***return json.load(f)***

***except Exception as e:***

***st.error(f"Error loading user data: {e}")***

***return {}***

***def save\_users(data):***

***"""Save user data with error handling"""***

***try:***

***with open("users.json", "w") as f:***

***json.dump(data, f, indent=4)***

***except Exception as e:***

***st.error(f"Error saving user data: {e}")***

***def load\_poster(url):***

***"""Load movie poster with error handling"""***

***try:***

***if url == POSTER\_PLACEHOLDER:***

***return Image.new('RGB', (150, 225), color='gray')***

***response = requests.get(url, timeout=5)***

***return Image.open(BytesIO(response.content))***

***except:***

***return Image.new('RGB', (150, 225), color='gray')***

***def register\_user(username, password, users):***

***"""Register new user with validation"""***

***if len(username) < 3:***

***return "Username must be at least 3 characters"***

***if len(password) < MIN\_PASSWORD\_LENGTH:***

***return f"Password must be at least {MIN\_PASSWORD\_LENGTH} characters"***

***if username in users:***

***return "Username already exists"***

***users[username] = {***

***"password": password,***

***"friends": [],***

***"watched": [],***

***"continue\_watching": [],***

***"favorites": [],***

***"xp": 0,***

***"level": 1,***

***"ratings": {},***

***"chats": {},***

***"join\_date": datetime.now().strftime("%Y-%m-%d"),***

***"preferences": {***

***"genres": [],***

***"actors": [],***

***"directors": []***

***}***

***}***

***save\_users(users)***

***return None***

***def get\_movie\_card(movie):***

***"""Create a styled movie card"""***

***with st.container():***

***col1, col2 = st.columns([1, 3])***

***with col1:***

***st.image(load\_poster(movie['Poster\_Link']), width=150)***

***with col2:***

***st.subheader(movie['Series\_Title'])***

***st.caption(f"⭐ {movie['IMDB\_Rating']} | {movie['Runtime']} min | {movie['Genre']}")***

***st.write(movie['Overview'][:150] + "...")***

***return st***

***def recommend\_movies(user\_data, df, matrix, tfidf, n=5):***

***"""Generate personalized recommendations with error handling"""***

***try:***

***if not user\_data["watched"]:***

***return df.sample(min(n, len(df)))***

***watched\_indices = df[df["Series\_Title"].isin(user\_data["watched"])].index***

***if len(watched\_indices) == 0:***

***return df.sample(min(n, len(df)))***

***watched\_vec = matrix[watched\_indices].mean(axis=0)***

***sim\_scores = cosine\_similarity(watched\_vec, matrix).flatten()***

***top\_indices = sim\_scores.argsort()[-n-1:-1][::-1]***

***return df.iloc[top\_indices]***

***except Exception as e:***

***st.error(f"Error generating recommendations: {str(e)}")***

***return df.sample(min(n, len(df)))***

***# Initialize session state***

***if "user" not in st.session\_state:***

***st.session\_state.user = None***

***if "page" not in st.session\_state:***

***st.session\_state.page = "Login"***

***if "search\_query" not in st.session\_state:***

***st.session\_state.search\_query = ""***

***if "current\_movie" not in st.session\_state:***

***st.session\_state.current\_movie = None***

***# Load data***

***df = load\_movies()***

***tfidf, matrix = vectorize(df)***

***users = load\_users()***

***# Page functions***

***def login\_page():***

***st.title(" AI Movie Matchmaker")***

***st.write("Find your perfect movie matches and connect with friends!")***

***tab1, tab2 = st.tabs(["Login", "Register"])***

***with tab1:***

***username = st.text\_input("Username", key="login\_user")***

***password = st.text\_input("Password", type="password", key="login\_pass")***

***if st.button("Login"):***

***if username in users and users[username]["password"] == password:***

***st.session\_state.user = username***

***st.session\_state.page = "Home"***

***st.rerun()***

***else:***

***st.error("Invalid credentials. Please try again.")***

***with tab2:***

***new\_user = st.text\_input("Choose a username", key="reg\_user")***

***new\_pass = st.text\_input("Choose a password", type="password", key="reg\_pass")***

***confirm\_pass = st.text\_input("Confirm password", type="password", key="confirm\_pass")***

***if st.button("Create Account"):***

***if new\_pass != confirm\_pass:***

***st.error("Passwords don't match!")***

***else:***

***error = register\_user(new\_user, new\_pass, users)***

***if error:***

***st.error(error)***

***else:***

***st.success("Account created! Please login.")***

***def home\_page():***

***st.sidebar.title(f"Welcome, {st.session\_state.user}")***

***user\_data = users[st.session\_state.user]***

***# Navigation***

***pages = {***

***" Home": "Home",***

***" Discover": "Discover",***

***" Friends": "Friends",***

***" Profile": "Profile"***

***}***

***selection = st.sidebar.radio("Menu", list(pages.keys()))***

***st.session\_state.page = pages[selection]***

***if st.sidebar.button("Logout"):***

***st.session\_state.user = None***

***st.session\_state.page = "Login"***

***st.rerun()***

***# Main content***

***st.title(" Your Movie Dashboard")***

***# Continue Watching***

***if user\_data["continue\_watching"]:***

***st.subheader("Continue Watching")***

***cols = st.columns(min(4, len(user\_data["continue\_watching"])))***

***for idx, title in enumerate(user\_data["continue\_watching"][:4]):***

***movie = df[df["Series\_Title"] == title].iloc[0]***

***with cols[idx % 4]:***

***get\_movie\_card(movie)***

***if st.button(f"Continue {title[:15]}...", key=f"cont\_{idx}"):***

***st.session\_state.current\_movie = movie.to\_dict()***

***st.session\_state.page = "Watch"***

***st.rerun()***

***# Recommendations***

***st.subheader("Recommended For You")***

***recs = recommend\_movies(user\_data, df, matrix, tfidf, 4)***

***if len(recs) > 0:***

***cols = st.columns(min(4, len(recs)))***

***for idx, (\_, movie) in enumerate(recs.iterrows()):***

***with cols[idx % 4]:***

***get\_movie\_card(movie)***

***if st.button(f"Watch {movie['Series\_Title'][:15]}...", key=f"rec\_{idx}"):***

***users[st.session\_state.user]["continue\_watching"].append(movie["Series\_Title"])***

***save\_users(users)***

***st.session\_state.current\_movie = movie.to\_dict()***

***st.session\_state.page = "Watch"***

***st.rerun()***

***else:***

***st.warning("No recommendations available. Watch some movies first!")***

***def discover\_page():***

***st.title(" Discover Movies")***

***# Search and filters***

***col1, col2 = st.columns(2)***

***with col1:***

***st.session\_state.search\_query = st.text\_input("Search movies", st.session\_state.search\_query)***

***with col2:***

***genre\_filter = st.selectbox("Filter by genre", ["All"] + list(df['Genre'].str.split(',').explode().str.strip().unique()))***

***# Display results***

***filtered = df.copy()***

***if st.session\_state.search\_query:***

***mask = (***

***filtered['Series\_Title'].str.contains(st.session\_state.search\_query, case=False) |***

***filtered['Director'].str.contains(st.session\_state.search\_query, case=False) |***

***filtered['Overview'].str.contains(st.session\_state.search\_query, case=False)***

***)***

***filtered = filtered[mask]***

***if genre\_filter != "All":***

***filtered = filtered[filtered['Genre'].str.contains(genre\_filter, case=False)]***

***st.subheader(f"Found {len(filtered)} movies")***

***for \_, movie in filtered.iterrows():***

***get\_movie\_card(movie)***

***if st.button(f"Watch {movie['Series\_Title']}", key=f"dis\_{movie['Series\_Title']}"):***

***users[st.session\_state.user]["continue\_watching"].append(movie["Series\_Title"])***

***save\_users(users)***

***st.session\_state.current\_movie = movie.to\_dict()***

***st.session\_state.page = "Watch"***

***st.rerun()***

***def watch\_page():***

***if st.session\_state.current\_movie is None:***

***st.warning("No movie selected")***

***st.session\_state.page = "Home"***

***st.rerun()***

***movie = st.session\_state.current\_movie***

***st.title(f" Watching: {movie['Series\_Title']}")***

***col1, col2 = st.columns([1, 2])***

***with col1:***

***st.image(load\_poster(movie['Poster\_Link']), width=300)***

***with col2:***

***st.subheader(movie['Series\_Title'])***

***st.write(f"\*\*Director:\*\* {movie['Director']}")***

***st.write(f"\*\*Genre:\*\* {movie['Genre']}")***

***st.write(f"\*\*Rating:\*\* ⭐ {movie['IMDB\_Rating']}")***

***st.write(f"\*\*Runtime:\*\* {movie['Runtime']} minutes")***

***st.write("\*\*Overview:\*\*")***

***st.write(movie['Overview'])***

***# Video player placeholder***

***st.video("https://sample-videos.com/video123/mp4/720/big\_buck\_bunny\_720p\_1mb.mp4")***

***# Rating and actions***

***with st.expander("Rate this movie"):***

***rating = st.slider("Your rating", 1, 10, 5)***

***if st.button("Submit Rating"):***

***user = st.session\_state.user***

***users[user]["ratings"][movie['Series\_Title']] = rating***

***if movie['Series\_Title'] not in users[user]["watched"]:***

***users[user]["watched"].append(movie['Series\_Title'])***

***users[user]["xp"] += XP\_PER\_WATCH***

***users[user]["xp"] += XP\_PER\_RATING***

***save\_users(users)***

***st.success("Rating submitted!")***

***if st.button("Back to Home"):***

***st.session\_state.page = "Home"***

***st.rerun()***

***def friends\_page():***

***st.title(" Friends")***

***username = st.session\_state.user***

***user\_data = users[username]***

***tab1, tab2 = st.tabs(["Your Friends", "Find Friends"])***

***with tab1:***

***st.subheader("Your Friends")***

***if not user\_data["friends"]:***

***st.info("You don't have any friends yet. Add some below!")***

***for friend in user\_data["friends"]:***

***with st.expander(friend):***

***st.write(f"Member since: {users[friend].get('join\_date', 'Unknown')}")***

***st.write(f"Level: {users[friend].get('level', 1)}")***

***# Display friend's recently watched***

***st.write("Recently watched:")***

***for movie in users[friend]["watched"][-3:]:***

***st.write(f"- {movie}")***

***# Chat button***

***if st.button(f"Chat with {friend}"):***

***st.session\_state.chat\_friend = friend***

***with tab2:***

***st.subheader("Find New Friends")***

***search\_term = st.text\_input("Search by username")***

***if search\_term:***

***results = [u for u in users if search\_term.lower() in u.lower() and u != username]***

***if not results:***

***st.info("No users found")***

***for user in results:***

***with st.container():***

***st.write(f"\*\*{user}\*\* (Level {users[user].get('level', 1)})")***

***if user not in user\_data["friends"]:***

***if st.button(f"Add {user}", key=f"add\_{user}"):***

***user\_data["friends"].append(user)***

***users[user]["friends"].append(username)***

***save\_users(users)***

***st.success(f"Added {user} as a friend!")***

***st.rerun()***

***# Chat interface***

***if "chat\_friend" in st.session\_state:***

***friend = st.session\_state.chat\_friend***

***st.subheader(f" Chat with {friend}")***

***# Get chat history***

***chat\_key = "-".join(sorted([username, friend]))***

***messages = user\_data["chats"].get(chat\_key, [])***

***# Display messages***

***for msg in messages[-10:]: # Show last 10 messages***

***align = "right" if msg["sender"] == username else "left"***

***st.chat\_message(align).write(f"{msg['sender']}: {msg['text']}")***

***# Send new message***

***new\_msg = st.chat\_input("Type your message")***

***if new\_msg:***

***new\_entry = {"sender": username, "text": new\_msg, "time": datetime.now().strftime("%H:%M")}***

***messages.append(new\_entry)***

***user\_data["chats"][chat\_key] = messages***

***users[friend]["chats"][chat\_key] = messages***

***save\_users(users)***

***st.rerun()***

***def profile\_page():***

***st.title(" Your Profile")***

***username = st.session\_state.user***

***user\_data = users[username]***

***col1, col2 = st.columns([1, 3])***

***with col1:***

***st.metric("Level", user\_data["level"])***

***st.progress(min(user\_data["xp"] / XP\_LEVEL\_THRESHOLD, 1.0))***

***st.caption(f"{user\_data['xp']}/{XP\_LEVEL\_THRESHOLD} XP to next level")***

***st.write(f"Member since: {user\_data.get('join\_date', 'Unknown')}")***

***with col2:***

***st.subheader("Your Stats")***

***st.write(f" Movies watched: {len(user\_data['watched'])}")***

***st.write(f"⭐ Movies rated: {len(user\_data['ratings'])}")***

***st.write(f"❤️ Favorites: {len(user\_data['favorites'])}")***

***st.write(f" Friends: {len(user\_data['friends'])}")***

***tab1, tab2, tab3 = st.tabs(["Your Movies", "Preferences", "Account"])***

***with tab1:***

***st.subheader("Your Watched Movies")***

***for movie in user\_data["watched"][-10:]: # Show last 10 watched***

***rating = user\_data["ratings"].get(movie, "Not rated")***

***st.write(f"- {movie} ({'⭐' \* int(rating) if isinstance(rating, int) else rating})")***

***st.subheader("Your Favorites")***

***for movie in user\_data["favorites"]:***

***st.write(f"- {movie}")***

***with tab2:***

***st.subheader("Update Preferences")***

***# Genre preferences***

***selected\_genres = st.multiselect(***

***"Favorite genres",***

***options=df['Genre'].str.split(',').explode().str.strip().unique(),***

***default=user\_data["preferences"]["genres"]***

***)***

***# Director preferences***

***selected\_directors = st.multiselect(***

***"Favorite directors",***

***options=df['Director'].unique(),***

***default=user\_data["preferences"]["directors"]***

***)***

***if st.button("Save Preferences"):***

***user\_data["preferences"]["genres"] = selected\_genres***

***user\_data["preferences"]["directors"] = selected\_directors***

***save\_users(users)***

***st.success("Preferences updated!")***

***with tab3:***

***st.subheader("Account Settings")***

***st.warning("Coming soon: Password change and account deletion")***

***# Main app router***

***def main():***

***if st.session\_state.user is None:***

***login\_page()***

***else:***

***if st.session\_state.page == "Home":***

***home\_page()***

***elif st.session\_state.page == "Discover":***

***discover\_page()***

***elif st.session\_state.page == "Watch":***

***watch\_page()***

***elif st.session\_state.page == "Friends":***

***friends\_page()***

***elif st.session\_state.page == "Profile":***

***profile\_page()***

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

***main()***

***14. Future Scope***

Integrate user reviews using NLP sentiment analysis

Add deep learning model (e.g., Neural Collaborative Filtering)

Use real-time user behavior for dynamic recommendations

***15. Team Members and Roles***

***Data Preprocessing and Interface Design and Deployment:*** SURYANARAYANA.S

***Exploratory Data Analysis (EDA) and Visualization***: SUSENTHIRAN.S

***Model Development and Documentation and Reporting:*** RAMESH.M

***Feature Engineering***: PRAVIN.R