Import pandas as pd

From sklearn.feature\_extraction.text import TfidfVectorizer

From sklearn.metrics.pairwise import cosine\_similarity

# Load movie data (replace with your actual data loading)

Movies\_df = pd.read\_csv(‘movies.csv’)

# Preprocess movie data (handle missing values, select relevant features)

# Assuming ‘title’, ‘genres’, and ‘description’ are relevant

Movies\_df[‘description’] = movies\_df[‘description’].fillna(‘’)

Movies\_df[‘combined\_features’] = movies\_df[‘title’] + ‘ ‘ + movies\_df[‘genres’] + ‘ ‘ + movies\_df[‘description’]

# Feature extraction using TF-IDF

Tfidf\_vectorizer = TfidfVectorizer(stop\_words=’english’)

Tfidf\_matrix = tfidf\_vectorizer.fit\_transform(movies\_df[‘combined\_features’])

# Calculate cosine similarity matrix

Cosine\_sim = cosine\_similarity(tfidf\_matrix)

# Create a mapping of movie title to index

Indices = pd.Series(movies\_df.index, index=movies\_df[‘title’]).drop\_duplicates()

Def get\_movie\_recommendations(title, cosine\_sim=cosine\_sim, movies\_df=movies\_df, indices=indices):

“””

Generates movie recommendations based on cosine similarity.

Args:

Title (str): The title of the movie to find recommendations for.

Cosine\_sim (numpy.ndarray): The cosine similarity matrix.

Movies\_df (pd.DataFrame): The DataFrame containing movie information.

Indices (pd.Series): Mapping of movie title to index.

Returns:

Pandas.Series: Top 10 recommended movie titles.

“””

Try:

# Get the index of the movie

Idx = indices[title]

# Get the pairwise similarity scores of all movies with that movie

Sim\_scores = list(enumerate(cosine\_sim[idx]))

# Sort the movies based on the similarity scores

Sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)

# Get the scores of the 10 most similar movies

Sim\_scores = sim\_scores[1:11]

# Get the movie indices

Movie\_indices = [i[0] for i in sim\_scores]

# Return the top 10 most similar movies

Return movies\_df[‘title’].iloc[movie\_indices]

Except KeyError:

Return “Movie not found in the dataset.”

# --- Integration with Matchmaking System (Conceptual) ---

# Assume you have a user profile with their movie preferences

Def get\_user\_movie\_preferences(user\_id):

“””

Retrieves a list of movies liked by a specific user.

(This would involve querying your user data)

“””

# Replace this with your actual data retrieval logic

User\_preferences = {

‘user\_1’: [‘Inception’, ‘The Dark Knight’, ‘Interstellar’],

‘user\_2’: [‘Pride & Prejudice’, ‘Sense and Sensibility’],

‘user\_3’: [‘Spirited Away’, ‘Princess Mononoke’]

}

Return user\_preferences.get(user\_id, [])

Def recommend\_movies\_for\_user(user\_id):

“””

Recommends movies for a user based on their past preferences.

“””

Liked\_movies = get\_user\_movie\_preferences(user\_id)

If not liked\_movies:

Return “No past movie preferences found for this user.”

Recommendations = pd.Series()

For movie\_title in liked\_movies:

Recommended = get\_movie\_recommendations(movie\_title)

If isinstance(recommended, pd.Series):

Recommendations = pd.concat([recommendations, recommended])

# Remove duplicates and potentially rank/sort recommendations

Return recommendations.drop\_duplicates().head(10)

# Example usage:

If \_\_name\_\_ == “\_\_main\_\_”:

Movie\_to\_find = “Avatar”

Recommendations = get\_movie\_recommendations(movie\_to\_find)

Print(f”Recommendations for ‘{movie\_to\_find}’:”)

Print(recommendations)

User\_id = ‘user\_1’

User\_recommendations = recommend\_movies\_for\_user(user\_id)

Print(f”\nMovie recommendations for user ‘{user\_id}’:”)

Print(user\_recommendations)