Task 3:

Given a list of movies with their genres, Write a Python function that recommends movies  
based on a user’s preferred genre. Use the Cursor AI tool. Use few shot prompting.

Prompt: Write a python program that takes Given a list of movies with their genres of a function that recommends movies

import json

from typing import List, Dict, Optional

import random

class MovieRecommender:

def \_init\_(self):

# Sample movie database with genres

self.movies = [

{"title": "The Shawshank Redemption", "genres": ["Drama"], "rating": 9.3, "year": 1994},

{"title": "The Godfather", "genres": ["Crime", "Drama"], "rating": 9.2, "year": 1972},

{"title": "Pulp Fiction", "genres": ["Crime", "Drama"], "rating": 8.9, "year": 1994},

{"title": "The Dark Knight", "genres": ["Action", "Crime", "Drama"], "rating": 9.0, "year": 2008},

{"title": "Fight Club", "genres": ["Drama"], "rating": 8.8, "year": 1999},

{"title": "Inception", "genres": ["Action", "Adventure", "Sci-Fi"], "rating": 8.8, "year": 2010},

{"title": "The Matrix", "genres": ["Action", "Sci-Fi"], "rating": 8.7, "year": 1999},

{"title": "Interstellar", "genres": ["Adventure", "Drama", "Sci-Fi"], "rating": 8.6, "year": 2014},

{"title": "The Lion King", "genres": ["Animation", "Adventure", "Drama"], "rating": 8.5, "year": 1994},

{"title": "Titanic", "genres": ["Drama", "Romance"], "rating": 7.9, "year": 1997},

{"title": "La La Land", "genres": ["Comedy", "Drama", "Musical"], "rating": 8.0, "year": 2016},

{"title": "Get Out", "genres": ["Horror", "Mystery", "Thriller"], "rating": 7.7, "year": 2017},

{"title": "The Grand Budapest Hotel", "genres": ["Comedy", "Drama"], "rating": 8.1, "year": 2014},

{"title": "Mad Max: Fury Road", "genres": ["Action", "Adventure", "Sci-Fi"], "rating": 8.1, "year": 2015},

{"title": "The Martian", "genres": ["Adventure", "Drama", "Sci-Fi"], "rating": 8.0, "year": 2015}

]

# Few-shot examples for genre matching

self.few\_shot\_examples = {

"Drama": [

"User likes emotional storytelling and character development",

"Recommend: The Shawshank Redemption, The Godfather, Fight Club"

],

"Action": [

"User enjoys fast-paced sequences and thrilling scenes",

"Recommend: The Dark Knight, The Matrix, Mad Max: Fury Road"

],

"Sci-Fi": [

"User is interested in futuristic concepts and technology",

"Recommend: Inception, The Matrix, Interstellar"

],

"Comedy": [

"User prefers light-hearted and humorous content",

"Recommend: La La Land, The Grand Budapest Hotel"

],

"Horror": [

"User enjoys suspenseful and scary content",

"Recommend: Get Out"

]

}

def get\_genre\_recommendations(self, preferred\_genre: str, num\_recommendations: int = 3) -> List[Dict]:

"""

Recommend movies based on user's preferred genre using few-shot prompting approach.

Args:

preferred\_genre (str): The genre the user prefers

num\_recommendations (int): Number of movies to recommend

Returns:

List[Dict]: List of recommended movies with their details

"""

# Convert to title case for better matching

preferred\_genre = preferred\_genre.title()

# Get few-shot context for the genre

genre\_context = self.few\_shot\_examples.get(preferred\_genre, [])

# Find movies that match the preferred genre

matching\_movies = []

for movie in self.movies:

if preferred\_genre in [genre.title() for genre in movie["genres"]]:

matching\_movies.append(movie)

# Sort by rating (highest first) and then by year (newest first)

matching\_movies.sort(key=lambda x: (x["rating"], x["year"]), reverse=True)

# Apply few-shot learning logic

if genre\_context:

# Use few-shot examples to enhance recommendations

enhanced\_movies = self.\_apply\_few\_shot\_learning(matching\_movies, preferred\_genre, genre\_context)

recommendations = enhanced\_movies[:num\_recommendations]

else:

# Fallback to standard recommendations

recommendations = matching\_movies[:num\_recommendations]

return recommendations

def \_apply\_few\_shot\_learning(self, movies: List[Dict], genre: str, context: List[str]) -> List[Dict]:

"""

Apply few-shot learning to enhance movie recommendations.

Args:

movies (List[Dict]): List of movies matching the genre

genre (str): The preferred genre

context (List[str]): Few-shot examples for the genre

Returns:

List[Dict]: Enhanced list of movies

"""

enhanced\_movies = movies.copy()

# Apply genre-specific logic based on few-shot examples

if genre == "Drama":

# Drama movies: prioritize emotional depth and character development

enhanced\_movies.sort(key=lambda x: (x["rating"], len(x["genres"]) == 1, x["year"]), reverse=True)

elif genre == "Action":

# Action movies: prioritize high energy and visual appeal

enhanced\_movies.sort(key=lambda x: (x["rating"], "Adventure" in x["genres"], x["year"]), reverse=True)

elif genre == "Sci-Fi":

# Sci-Fi movies: prioritize innovative concepts and modern effects

enhanced\_movies.sort(key=lambda x: (x["rating"], x["year"], "Action" in x["genres"]), reverse=True)

elif genre == "Comedy":

# Comedy movies: prioritize humor and entertainment value

enhanced\_movies.sort(key=lambda x: (x["rating"], "Musical" in x["genres"], x["year"]), reverse=True)

elif genre == "Horror":

# Horror movies: prioritize suspense and psychological elements

enhanced\_movies.sort(key=lambda x: (x["rating"], "Mystery" in x["genres"], x["year"]), reverse=True)

return enhanced\_movies

def get\_multi\_genre\_recommendations(self, preferred\_genres: List[str], num\_recommendations: int = 5) -> List[Dict]:

"""

Recommend movies based on multiple preferred genres.

Args:

preferred\_genres (List[str]): List of genres the user prefers

num\_recommendations (int): Number of movies to recommend

Returns:

List[Dict]: List of recommended movies

"""

# Find movies that match any of the preferred genres

matching\_movies = []

for movie in self.movies:

movie\_genres = [genre.title() for genre in movie["genres"]]

if any(genre.title() in movie\_genres for genre in preferred\_genres):

# Calculate genre match score

match\_score = sum(1 for genre in preferred\_genres if genre.title() in movie\_genres)

movie["genre\_match\_score"] = match\_score

matching\_movies.append(movie)

# Sort by genre match score, then by rating, then by year

matching\_movies.sort(key=lambda x: (x["genre\_match\_score"], x["rating"], x["year"]), reverse=True)

# Remove the temporary score field

for movie in matching\_movies:

movie.pop("genre\_match\_score", None)

return matching\_movies[:num\_recommendations]

def get\_personalized\_recommendations(self, user\_preferences: Dict[str, float], num\_recommendations: int = 5) -> List[Dict]:

"""

Get personalized recommendations based on user genre preferences with weights.

Args:

user\_preferences (Dict[str, float]): Dictionary mapping genres to preference weights (0.0 to 1.0)

num\_recommendations (int): Number of movies to recommend

Returns:

List[Dict]: List of recommended movies

"""

scored\_movies = []

for movie in self.movies:

total\_score = 0.0

for genre, weight in user\_preferences.items():

if genre.title() in [g.title() for g in movie["genres"]]:

total\_score += weight

if total\_score > 0:

# Normalize score by number of genres and add rating bonus

normalized\_score = (total\_score / len(movie["genres"])) + (movie["rating"] / 10.0)

movie\_copy = movie.copy()

movie\_copy["personalized\_score"] = normalized\_score

scored\_movies.append(movie\_copy)

# Sort by personalized score

scored\_movies.sort(key=lambda x: x["personalized\_score"], reverse=True)

# Remove the temporary score field

for movie in scored\_movies:

movie.pop("personalized\_score", None)

return scored\_movies[:num\_recommendations]

def display\_recommendations(self, recommendations: List[Dict], genre: str = None):

"""

Display movie recommendations in a formatted way.

Args:

recommendations (List[Dict]): List of recommended movies

genre (str): The genre used for recommendations

"""

if not recommendations:

print(f"No movies found for genre: {genre}")

return

print(f"\n🎬 Movie Recommendations for {genre or 'your preferences'}:")

print("=" \* 60)

for i, movie in enumerate(recommendations, 1):

genres\_str = ", ".join(movie["genres"])

print(f"{i}. {movie['title']} ({movie['year']})")

print(f" Genres: {genres\_str}")

print(f" Rating: ⭐ {movie['rating']}/10")

print()

def main():

"""Main function to demonstrate the movie recommendation system."""

recommender = MovieRecommender()

print("🎭 Movie Recommendation System with Few-Shot Learning")

print("=" \* 60)

# Example 1: Single genre recommendation

print("\n1️⃣ Single Genre Recommendation:")

drama\_movies = recommender.get\_genre\_recommendations("Drama", 3)

recommender.display\_recommendations(drama\_movies, "Drama")

# Example 2: Multi-genre recommendation

print("\n2️⃣ Multi-Genre Recommendation:")

action\_sci\_fi = recommender.get\_multi\_genre\_recommendations(["Action", "Sci-Fi"], 4)

recommender.display\_recommendations(action\_sci\_fi, "Action & Sci-Fi")

# Example 3: Personalized recommendation

print("\n3️⃣ Personalized Recommendation:")

user\_prefs = {"Drama": 0.8, "Comedy": 0.6, "Romance": 0.4}

personalized = recommender.get\_personalized\_recommendations(user\_prefs, 4)

recommender.display\_recommendations(personalized, "Personalized")

# Interactive demo

print("\n🎯 Interactive Demo:")

print("Enter a genre to get recommendations (or 'quit' to exit):")

while True:

user\_input = input("\nEnter genre: ").strip()

if user\_input.lower() in ['quit', 'exit', 'q']:

print("Thanks for using the Movie Recommendation System! 🎬")

break

if user\_input:

recommendations = recommender.get\_genre\_recommendations(user\_input, 5)

recommender.display\_recommendations(recommendations, user\_input)

else:

print("Please enter a valid genre.")

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

main()

#Output:

