Movie Recommendation System

ICS 104-232 Lab Project – Section F72-F81

The aim of the Movie Recommendation System is to provide information/recommendations to interested users for a set of movies. You are required to write a Python program that stores, processes, and retrieves information as specified below.

Specifications:

The program should display the following menu:

Movies Recommendation System

1. Add new Movie information
2. Update Movie information
3. Display information for all Movie
4. Display specific information
5. Save specific genre in new files
6. Exit

1. **Adding new movies:** This option allows the user to add movie information to the system. The fields for each movie are described in Table 1. Data should be stored in a file called "movie.txt".

Table 1 Data specification

Field	Specification	Type of data	Possible values
MovieID	Movie identification	String of 5 digits	00001
MovieName	Name of the Movie	String	60 Rules for Life
MovieAuthor	The writer of the Movie	String	First_name Last_name
UserRating	user rating	Float value from 1 to 5	3.5
NumOfReviews	Number of reviews	Integer	1730
Revenue	The revenue of the movie in millions	float	100.0M, 1.5M
Year	The evaluation year	Integer	2022
Genre	The domain of the Movie It might be 0: sport, 1: art, 2: economic, 3: comedy 4: fiction and 5: politics	String	Sport

Table 2 presents examples of valid data.

Table 2 Examples of Valid data

MovieID	MovieName	MovieWriter	UserRating	NumOfReviews	Revenue	Year	Genre
12018	A Wrinkle in Time (Time Quintet)	Madeleine L'Engle	4.5	5153	20M	2018	fiction

12019	A Wrinkle in Time (Time Quintet)	Madeleine L'Engle	4.4	5050	20M	2019	fiction
22011	UP	Disney	5.0	10000	80M	2011	Fiction, art

The program should accept only valid names and values. Each Movie should have one record per year. The system should display an appropriate error message if the MovieID already has been reviewed for the same Year. For example, the data presented in Table 3 is not valid. This is because each movie should be reviewed once per a year.
You should ask the user to enter the number corresponding to the movie Genre, then write it in letters. For instance, if the user entered 0, this means the genre is sport and this is how it should be saved in the file. A movie can have up to 5 Genres, so you should ask the user how many Genres he will enter, if he entered a value larger than 5, then you should print an error message, then ask again.

When adding the new movie the MovieID should be automatically assigned. You should check the Movie Name, if the movie has the same name, then you should check the movie year, if the movie name and year are the same you should display an error and not add the movie, get the user back to the main menu. If the movie has the same name but not the same year, then the new id is the old id + the date. For instance, movie 1 is created in 2012 and the ID is 12012. The second review for the same movie is in 2013, then the ID should be 12013. The first number '1' is just a sequential number. The second number is the date of Review. So, to create the movieID you should check if the movie already exist, if it does use the same sequential number (ex; 1), else add one to the last existing sequential number and use that (ex; 2) . (check Table 2)

Note that when the user Enter the MovieName, they may use upper or lower case letters, your code should be able to recognize if the same name is entered but with different letters (check table 3)

Table 3 Examples of NOT valid data

MovieID	**MovieName**	**MovieWriter**	**UserRating**	**NumOfReviews**	Revenue	**ReviewYear**	**Genre**
~~12018~~	~~A Wrinkle in Time (Time Quintet)~~	~~Madeleine L'Engle~~	~~4.5~~	~~5153~~	~~20M~~	~~2018~~	~~fiction~~
~~12018~~	~~A Wrinkle in Time (Time Quintet)~~	~~Madeleine L'Engle~~	~~4.4~~	~~5050~~	~~20M~~	~~2018~~	~~fiction~~
~~12018~~	~~a wrinkle in time (time quintet)~~	~~Madeleine L'Engle~~	~~4.4~~	~~5050~~	~~20M~~	~~2018~~	~~fiction~~

2. **Updating Movie information**: this option allows the user to update UserRating, NumOfReviews, and Revenue. Once this option is selected, the program should ask for MovieName and Year, then retrieve Movie information based on MovieName and Year. For each field, the program should ask whether it needs to be updated or not. If it needs to be updated, the program asks for the new information. Once all information is entered, the program updates them and displays an appropriate message. The updated movie should be changed in the file as well.

3. **Displaying information about all Movies**: this option allows the user to display Movies' information in a nicely formatted table. It has the following sub-options (select only 3 of the following sub-options)
    - Sort by MovieID
    - Sort by MovieName
    - Sort by Genre
    - Sort by Revenue
    - Sorted by Year

4. **Displaying specific information:** this feature allows the user to display Movies' information based on some specified criteria. It has the following sub-options**:** (select only 4 of the following sub-options)
    - View the top 5 selling movies in the last 10 years (sorted based on Year and UserRating).
    - Print the Total revenue for all movies per Year.
    - Print the Average revenue of all movies per Year.
    - Print the Total number of all (unique) movies per Author.
    - Print the Average revenue of all (unique) movies per Author.
    - View a sorted list of movies (by movieName) for a given Year (In alphabetical order).
    - View a sorted list of movies (by movieID) for a given year for a given Genre (in alphabetical order) (ask the user for Genre first).

5. **Saving movie information** for a certain Genre in a new file. The program should ask the user for the Genre, then it should write the retrieved information for that Genre in the new file. For instance, create a new file containing all the fiction movies.

6. **Terminating the program**: Remember, your program should be always running until the user closes it with this option. For example, press 'Q' to exit.

7. **Otherwise:** the program should display an appropriate error message and inform the user to enter a valid option. After each menu item is terminated the code should return the user to the main menu.

8. **Testing**: You should create a text file with 10-20 movie entries, so that your code can be tested for part 4. You are recommended to create test cases and check that all the parts are working correctly before submission.

Please consider the following:

- Comments are important. (worth 5%).
- The code must use meaningful variable names and modular programming (using functions). (worth 10%)
- Global variables are not allowed (pass parameters to functions and receive results).
- Each team must submit a working program. Non-working parts can be submitted separately. If a team submits a non-working program, it loses 20% of the grade.
- User input must be validated by the program i.e. valid range and valid type

Submission

Deliverable:

Each team has to submit:

- The code as a Jupyter notebook
- The report as part of the Jupyter notebook or as a separate word file. The report will describe how you solved the problem. In addition, you need to describe the different functions with your task and screenshots of your running code. (worth 5%)

Lab demo/presentation:

- **We will announce a date for the demos.**
- A slot of 15 minutes will be allocated to each team for their presentation and questions.
- Students who do not appear for the lab demo/presentation will get a 0.
- Each team should consist of 2 students.

20% of the grade is highlighted above. The remaining 80% will be on the code itself and presentation.

```
# This work done by group $$:
# Name of First Student, ID, Percentage of his work contribution

# Name of Second Student, ID, Percentage of his work contribution
```