Homework Assignment 1

Movie Recommendation System - Build using AI

Worth 150 points (15% of course grade).

Posted Monday, Sep 29

Due by Friday Oct 17 at 11 PM in Autolab (autolab.cs.rutgers.edu)

(Autolab will be open for submission on Monday, Oct 6)

Late Submission: By Saturday, Oct 18, 11 PM in Autolab - 15 point penalty
(15 points will be deducted from your score)

You are required to design a prototype movie recommendation program in Python as detailed in the description below.

You are **required** to use one or more of these specific AI chatbots to build the program: OpenAI ChatGPT, Anthropic Claude, Google Gemini.
You may not use any coding tool other than these chatbots.

You will work in groups of 3.

We will ONLY grade assignments submitted to Autolab. Assignments emailed to the instructor/TAs/graders will be ignored--they will NOT be accepted for grading. Also, any assignment on your computer, or any other online space, will NOT be accepted for grading, even if it appears to be not modified after a submission date.

See **Grading Process** section at the end of this page to see how Autolab will be set up to accept your assignment. This may be different than how Autolab is setup in other courses where you may have used it. In particular, Autolab will only be used for submitting assignments, not for autograding. Grading will be done by TAs/graders outside of Autolab after the late submission deadline has passed.

Input Datasets

- **Movies file**: A text file that contains the genres of movies, with lines in the format:

  `movie_genre|movie_id|movie_name`

  Each line has a movie genre, a movie id, and the name (with year) of the movie. To keep it simple, each movie belongs to a single genre. Here's a sample movies file.

  Note: A movie name includes the year, since it's possible different movies have the same title, but were made in different years. However, no two movies will have the same name in the same year.

- **Ratings file**: A text file that contains movie ratings, with lines in the format:

  `movie_name|rating|user_id`

  Each line has the name (with year) of a movie, its rating (range 0-5 inclusive), and the id of the user who rated the movie, and these fields are separated by the pipe character '|'. A movie can have

multiple ratings from different users. A user can rate a particular movie only once. A user can however rate multiple movies. Here's a sample ratings file.

[30 pts] Coding Specification

Recommendations are to be based on the following features that your code will implement:

- [6 pts] **Movie popularity**
  Top n movies (ranked on average ratings)
- [6 pts] **Movie popularity in genre**
  Top n movies in a genre (ranked on average ratings)
- [6 pts] **Genre popularity**
  Top n genres (ranked on average of average ratings of movies in genre)
- **[6 pts] User preference for genre**
  Genre most preferred by user (average of average ratings by user of movies in genre)
- [6 pts] **Recommend movies**
  3 most popular movies from user's top genre that the user has not yet rated

Build an executable python program, `movie_recommender.py` which can be run like this:

```
> python movie_recommender.py
```

Your program should have a command line interface (CLI) with menus to load input data files and test each function so you can run the program and test for yourself

We will grade your program exactly this way, by running it, and using the menuoptions to test each feature.

Add docstring documentation to each function

Ensure that your code will work with Python 3.12

[10 pts] Testing Specification

Have the chatbot generate a python program called `test_movie_recommender.py` to do automatic testing. The tester code should be runnable like this:

```
> python test_movie_recommender.py
```

It should load input files, call each function, and print the result. This is quicker than manually running the original program and individually testing each menu option.

Compare printed results against expected values so you know the code is correct.

- [5 pts] **Coverage of features & expected output checks**
  Call all feature functions and assert/print expected vs actual results.
- [5 pts] **Edge-case & negative tests**
  Sanity tests on inputs to catch malformed rows, empty files, duplicate ratings, non-numeric ratings, and tie behavior.

We will grade your testing program by running it on our test inputs, and reviewing the outcome.

[60 pts] Documentation - Code Generation

Clearly document the following, in a file named `coding_doc.pdf`
(Note: You need to submit a PDF file, no other format will be accepted.)

- [20 pts] **Chatbot Sessions Links**
  Provide link(s) to your chatbot session(s).
  The sessions should show ALL your prompts and responses from the chatbot. This should include prompts and responses for generation of test code.

- [5 pts] **Chatbots used**
  Of the suggested chatbots ChatGPT, Claude, and Gemini, which ones did you use? If you used more than one, briefly describe which aspects of coding and testing were handled by which chatbot.

- [7 pts] **Case-sensitivity handling**
  What, if any, parts of the data are to be processed in a case-sensitive manner, and which ones are not? Why? Did the chatbot automatically handle this in its code? If not, which aspects of your requirement weren't handled and how did you fix it?

- [8 pts] **Unclear code/chatbot explanations**
  What parts of the generated code did you not understand? How did you get your chatbot to explain the code to help you understand? Of these, which parts were specifically language features that were not covered in class?

- [10 pts] **Input data issues handling**
  What specific measures did the generated code take to circumvent possible issues with the input data, such as formatting errors, etc? List all these measures.

- [10 pts] **Issues not handled by AI code, fixes**
  Were there issues with the input data you could think of that were not taken into account by the generated code? If so, what were these prospective issues, and how did you get your chatbot to generate code for them?

[40 pts] Documentation - Testing

Clearly document the following, in a file named `testing_doc.pdf` (again, only PDF accepted)

- [5 pts] **Input file generation**
  Did you generate input files? Did you have the chatbot generate input files? Detail which files were generated by you, and which ones by the chatbot.

- [20 pts] **Incorporating isses in input for correctness testing**
  How did you incoporate possible issues in the input files so you can test the orrectness of code that handles these issues?

- [15 pts] **Ensuring good distribution of input data**
  How did you ensure a good distribution of data in the input files so that you could be confident that your tests exercised various paths through the code?

[10 pts] Group Zoom Call

It is important that all partners contribute meaningfully to the assignment.

Submit a document named `contributions_doc.pdf` that details the contributions made by each group member. Under each of your names, document the specifics of exactly what you contributed in each of the aspects of prompting, generating code, and documenting.

Your grader will schedule a 10-minute Zoom call with your group to go over individual contributions, plus each member’s experience using AI, both benefits and limitations. Credit for this part will be individually assigned.

What to submit

Zip the files `movie_recommender.py`, `test_movie_recommender.py`, `coding_doc.pdf`, `testing_doc.pdf`, and `contributions_doc.pdf` into a file named `hw1.zip`

Do NOT include any data files

IMPORTANT NOTES ON CREATING THE ZIP FILE:

- When you zip these files, make sure it does not create an extra folder.

  You can test this by unzipping the file, which should extract the files you zipped in the same folder where you are unzipping the file, without creating a containing folder for the extracted files.

- Do NOT zip inside an IDE (VSCode, etc.), this might introduce peculiarities that will cause issues while unzipping.

  Instead, do the zipping outside the IDE, at the OS level, either in file explorer, or on the command line (Mac/Linux, zip command). Then unzip using the same approach with file explorer, or command line (unzip command) to make sure that no extra folder was created while zipping.

If you have any questions on this, ask your section TA or group grader.

Submit `hw1.zip`

Autolab will be set up with group info, so all submissions will be treated as group submissions. This means any time you submit, that submission will be attributed to your partners as well, and they will see the same submission against their name. See more details below in the **Grading Process** section.

Grading Process

- ONLY submissions to Autolab will be graded.
- We will ignore any code that is emailed to us, no exceptions.
- We will not inspect or accept code that is on your computer, or on any other online space (such as GitHub), no exceptions.
- Any submission after the regular deadline of Oct 17, 11 pm will be treated as a late submission, and will accrue a deduction of 15 points on the score.
- **You can submit a max of 3 times, but only the LAST submission will be graded.**
  (We are allowing 3 submissions in case you forget to upload one or more of the required files, or want to make changes after you have uploaded.)
- As soon as you submit your program, the **autochecker** (NOT grader) in Autolab will immediately run, NOT to grade your program, but to make sure you have submitted all the required files. Any issues with the submission will be pointed out in the grade report so you can fix them and resubmit.
- After the late submission deadline has passed, your group's grader will download submissions from Autolab, and grade them outside of Autolab.
- Once they finish grading your submission, they will schedule a 10-minute group Zoom call.