1. Chatbot Session Link:

Chatbot Session Link: https://chatgpt.com/share/68f2d20f-72b0-800f-9427-7fd18ac1d6dd
The session contains:

- Initial prompt describing the Movie Recommendation System requirements
- AI-generated Python code
- Follow-up prompts for debugging and adding features
- Clarification prompts regarding data handling and validation

2. Chatbot Used:

Chatbot Name: ChatGPT (OpenAI)
Model: GPT-5
Only one chatbot session was used throughout the development process. The chatbot was mainly used for:

- Generating the base structure of the recommendation system
- Writing functions for loading movies, ratings, and calculating averages
- Implementing features like top movies, genre popularity, and user-based recommendations
- Explaining unclear code sections and debugging logic errors
- No other chatbots (Claude or Gemini) were used in this assignment.

3. Case-Sensitivity Handling:

The code treats movie names and genres in a case-sensitive manner by default.

For example:

- Movie names like "Inception (2010)" and "inception (2010)" would be treated as two different movies.
- Genres like "Action" and "action" would also be treated as separate.

The chatbot did not automatically implement case normalization. If we wanted to make the system more robust, we could add .lower() conversions when comparing strings for movie names or genres. However, the current implementation was left case-sensitive.

4. Unclear code/chatbot explanations:

At first, I didn't fully understand how some of the functions in the generated code worked. Particularly:

- List comprehensions like [m.name for m in self.movies_by_name.values() if m.genre == genre].
- Lambda sorting inside .sort(key=lambda x: (x[1], x[2], x[0]), reverse=True).
- Dataclasses and the meaning of @dataclass(frozen=True).

I asked the chatbot to explain and it clarified that:

- List comprehensions are just a compact way to build lists with a condition.
- Lambda functions are anonymous short functions used as sorting keys.
- frozen=True in dataclasses makes the object immutable (cannot be changed once created).

5. Input data issues handling:

The generated code already handled many possible problems with input data:

- Empty lines in the files are ignored (if not line: continue).
- Empty fields (like missing movie name or rating) also raise errors.
- Non-numeric or out-of-range ratings (not between 0 and 5) are rejected.
- Duplicate ratings by the same user for the same movie are detected and cause an error.
- Ratings for unknown movies are not fatal — they're safely ignored.
- Whitespace trimming is applied to all fields before validation.
- All errors are caught and printed in the CLI so the program doesn't crash.

So overall, the chatbot-generated code took strong measures to ensure file integrity and prevent errors due to bad data.

6. Issues not handled by AI code, fixes:

The AI-generated code handled most of the basic input cases correctly, including malformed lines, missing fields, and non-numeric ratings. However, a few potential issues were not explicitly handled:

Case Sensitivity in Movie Names and Genres

- The code treats movie names and genres exactly as they appear in the input files (case-sensitive).
- The AI did not include automatic case normalization (e.g., converting input to lowercase for matching).
- No .lower() calls or transformations were added, so users must type movie names and genres exactly as they appear in the dataset. This behavior was accepted as-is because movie names include years and need exact matching.

Overall, the AI-generated code handled all other prospective input issues, and no further fixes were necessary.