1. Input file generation:

For testing, we created two input files:

- test_movies.txt: Contains a small set of movies spanning multiple genres.
- test_ratings.txt: Contains ratings from different users for the movies in test_movies.txt.

These files were generated manually by our team member responsible for testing. The chatbot was not used to generate the input files, but it was used to help write the code that loads and processes these files. This ensured the test files had controlled content and could reliably test the functions.

2. Incorporating Issues in input for correctness testing:

To test correctness and robustness, we included possible problematic scenarios in the input data:

- Duplicate ratings by the same user
- Empty or missing fields
- Edge ratings to test average calculations
- User-specific and genre-specific cases

By including these scenarios, we ensured that the system correctly handled expected input issues without crashing or producing incorrect outputs.

3. Ensuring good distribution of input data:

To guarantee that all code paths were exercised:

- We included multiple genres: Action, Comedy, Drama, etc. This allowed testing of top_movies_in_genre and top_genres functions.

- Multiple users: Ratings from three different users (u1, u2, u3) ensured functions like recommend_for_user and user_preferred_genre worked in a meaningful manner.

- Varying ratings: Ratings ranged properly to test the sorting logic and average calculations.

- Sufficient quantity: Each function was guaranteed at least 3–4 data points to avoid corner cases where functions return empty lists or None.

This distribution allowed the test script to print expected outputs.