This report aims to implement an accurate recommender system model based on a dataset that includes UserID, ItemID, and Rating by Amazon. The task is to predict the potential rating of each user for a specific item and submit the results on the Kaggle competition platform, with RMSE (Root Mean Squared Error) as the main evaluation metric.

Due to the data, I conclude that the data belongs to a Collaborating Filtering prediction task since the data doesn't include any items with features or any details, such as categories or introductions, etc., but based on the ''rating behavior on the same goods'' or 'rating trends of similar goods' among users. In other words, whether through User-based Collaborating filtering (finding people whose behaviors are similar to those of the user) or Item-based Collaborating filtering (finding similar products) methods, recommendations are made based on the interaction records between the user and items, which is in line with the essence of Collaborating filtering.

To ensure the model's performance was trustworthy, I relied on two evaluation strategies: 5-fold cross-validation RMSE and a manual RMSE calculated from an 80/20 split of the training data. Comparing these two metrics helped me monitor potential overfitting—if the manual RMSE was much worse than the CV result, that usually meant the model wasn't generalizing well. Fortunately, the two values stayed fairly close throughout most experiments, which gave me more confidence in the model's robustness. Since RMSE is just the square root of MSE, this approach aligns with the evaluation standard stated in the assignment.

Once I found the best parameters using GridSearchCV, I retrained the model on the full training set and used it to generate the final predictions. I also made sure user and item IDs were properly mapped to numeric indices, so they would be compatible with algorithms that work with latent factor matrices.

This full process—from tuning and validation, to fallback design and final training—formed the backbone of my final submission. In the meantime, the Cold-start problem also emerged in the data. For example, there were new users who had never appeared before in the test data, and the model could not be directly used for prediction. Therefore, I also designed the fallback strategy to handle this kind of situation. For more detailed model selection, parameter modification, fallback design, model comparison, etc., will be explained in the following paragraphs.

The training dataset contains 103,606 user-item rating records, with a total of 9,300 unique users and 2,970 unique items. The test set includes 25,901 records, involving 8,144 users and 2,960 items.

After checking the overlap, I found that only 4 users in the test set are completely new, which is just 0.05% of the test users. All test items, however, also appear in the training set, so no cold-start items; only cold-start users.

Although user and item coverage are high, the rating matrix is still very sparse. Most users have rated just a small number of items. There's also no extra data like item categories or user info, so this problem can only be tackled using Collaborative Filtering, based purely on rating behavior.

Because of the few new users in the test set, I also added a fallback method to handle those cases where the model can't make a direct prediction.

To handle cold-start users, I adopted a weighted fallback strategy using user average, item average, and the global mean. This fallback was chosen because no side information (like user demographics or item tags) is available in the dataset, which prevents any content-based alternatives. Since the test set contains only four cold-start users, a simple weighted smoothing fallback was sufficient and practical.

At the beginning of this project, I drew inspiration from class materials, particularly models like SVD and ALS from platforms such as latent factor models, which are designed to uncover latent user-item relationships. Among them, I was especially interested in ALS (Alternating Least Squares) for its ability to handle missing values and implicit feedback, which seemed promising for a sparse dataset like this.

Initially, I attempted to implement a hybrid model combining both SVD and ALS. However, due to compatibility issues with available packages on platforms like Google Colab and Kaggle, I eventually decided to focus on SVD alone, as it was better supported in the Python environment I was working in.

After trying a bunch of parameter combinations, I noticed that SVD always stayed around 0.890 RMSE, no matter how much I tuned it. So I started testing other models, including KNN-based recommenders, and even tried out a simple deep learning model. Specifically, I implemented a shallow neural network with embedding layers for users and items, followed by fully connected layers and dropout, trained using Optuna. In theory, deep learning models should be able to capture more complex user-item interaction patterns, and they can be useful when there's side information like user demographics or item features. However, in this case, where the data is small, sparse, and lacks additional context, the deep model didn't outperform traditional methods and was harder to fine-tune. So I decided to set it aside.

What really surprised me was the BaselineOnly model. Although I initially added BaselineOnly just as a supporting model for hybrid setups, I later realized its statistical bias adjustment actually worked better than expected. Its simplicity allowed it to handle the sparse data more gracefully, especially since it doesn't rely on learning latent factors, but instead directly estimates baseline ratings using user and item biases. This made it surprisingly effective in situations where the data lacked complexity, as was the case here. I originally intended to combine it with more complex models like SVD, but seeing that it consistently produced lower RMSE even on its own, I decided to fully evaluate it as a standalone model.

After conducting additional rounds of tuning and evaluation, I found that BaselineOnly consistently delivered strong results, both in cross-validation and manual validation. Eventually, I selected it as the final model for my submission, and it helped me achieve a top score of 0.882 RMSE on the leaderboard!