Pumpkin Meter: A Collaborative Filtering Movie Recommendation

# **Introduction**

We live in a world where entertainment becomes a big part of our life. Digital technology has already made its impact into the entertainment world and has totally disrupted the marketspace. As the means of watching movies diversifies, the demand for credible platforms that provide a recommendation for viewers becomes higher as well. The most interesting fact is that we all have unique choices to watch movies. Taking this into account, Ripe Pumpkins has built a movie recommendation engine called Pumpkin Meter. The Pumpkin Meter suggests movies by collaborative filtering with Pumpkin Meter Score. A recommendation system provides suggestions to the users through a filtering process that is based on user preferences and browsing history. Ripe Pumpkins will use data generated by GroupLens research to train the system and develop recommendations for its customers. This information reflects the prior usage of the product as well as the assigned ratings. Pumpkin Meter is a platform that provides its users with various contents based on their preferences and likings.

# **Dataset**

Ripe Pumpkins and the Pumpkin Meter are built upon a dataset provided by the GroupLens research lab based out of the Department of Computer Science and Engineering at University of Minnesota, Twin Cities. The mission of the lab is to “advance the theory and practice of social computing by building and understanding systems used by real people” and specializes in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

The data is accessible via the GroupLens website (<http://grouplens.org/datasets/>), and downloadable as a zip file containing several csv (comma-separated values) files. The file contains 6 csv documents: ratings.csv, tags.csv, movies.csv, links.csv, genome-scores.csv, and genome-tags.csv. GroupLens provides 2 data sets: Small (ml-latest-small) and Large (ml-latest). The small dataset was used as a preliminary benchmark, but the full dataset was used for the final product. Both data sets contained the same information; both containing the same csv-type files, but with different numbers of records.

Please note, this project only required the use of the ratings.csv and movies.csv files. Please see dataset specific information below:

### **Small Data Set**

* + The latest small dataset (“ml-latest-small”) was created by 610 random users from March 29, 1996 and September 24, 2018 and published on September 26, 2018.
  + Each review and movie is sourced from MovieLens.org (a movie recommendation service), where each film has at least 1 rating.
  + All non-rated films were omitted from this dataset.
  + In total, the full dataset utilized in this project contains 10,0836 movie ratings and 3,683 tag applications across 9,742 movies.

### **Full Data Set**

* + The latest full dataset (“ml-latest”) was created by 283,228 random users from January 09, 1995 to September 24, 2018 and published on September 26, 2018.
  + Each review and movie is sourced from MovieLens.org (a movie recommendation service), where each film has at least 1 rating.
  + All non-rated films were omitted from this dataset.
  + In total, the full dataset utilized in this project contains 27,753,444 movie ratings and 1,108,997 tag applications across 58,098 movies.

## **Data Limitations**

The ml-latest dataset is a development dataset. It was collected with the purpose of algorithm training and evaluating a classifier. This dataset is designed to be updated and changed, and is not immutable, meaning that it will not suffice for full academic scrutiny or reproduction. In addition to the changing nature of the data, the data scope is limited to ratings and tags. There is no demographic or personal information listed and each user is noted by a distinct ID.

In addition, the datasets are encoded as UTF-8 (UCS Transformation Format 8), which means that the data has to specifically be declared as UTF-8. If it is not, accents or punctuations may not display accurately.

## **Data Summary & Structure**

There are 2 identifier fields that are joinable between tables: User IDs (userId) and Movie IDs (movieId). User IDs are anonymized and joinable across the ratings and tags files, while Movie IDs are joinable across the ratings, tags, movies, and links csv files. Movie IDs also correspond directly to the movielens.org url (eg: <https://movielens.org/movies/1> is the url for the movie with ID 1). In addition, timestamps in all csv files are standardized to document the seconds since midnight of January 1, 1970 (UTC).

### **Ratings Data File (ratings.csv)**

Document contains all ratings (scaled 0.5 to 5 stars) where each row is representative of a single rating from a single user of a singular film. The file is ordered first by userId, then by the associated movieIds. Timestamps mark the seconds since midnight, January 1, 1970 (UTC). The format is as follows: + userId, movieId, rating, timestamp.

### **Movies Data File (movies.csv)**

Document contains all Movie information, where each row holds a unique MovieId/Movie and is imported from <https://www.themoviedb.org/> (MovieId corresponds directly to the website url). The format is as follows: + movieId, title, genres.

Each title field includes the release year (in parenthesis), while the genre field can hold several standardized categories pipe-separated. Genre categories are as follows:

* Action
* Adventure
* Animation
* Children's
* Comedy
* Crime
* Documentary
* Drama
* Fantasy
* Film-Noir
* Horror
* Musical
* Mystery
* Romance
* Sci-Fi
* Thriller
* War
* Western
* (no genres listed)

## Data Source URLs & Resources

<https://movielens.org/>

<https://www.themoviedb.org/>

<https://grouplens.org/datasets/movielens/latest/>

<https://files.grouplens.org/datasets/movielens/ml-latest-README.html>

# **Technical Details**

## **General Practices**

In order to provide the most accurate and efficient movie recommender, our team utilized Spark and Python through Jupyter Notebooks. We relied heavily on the PySpark MLlib library (Spark’s machine learning package) to complete the analysis. With Jupyter Notebooks, our team was able to document each individual step of our process with markdowns and headings. Readability and sustainability were top priority to ensure everyone on the team was able to contribute and update as needed.

The world is ever changing, and as more data is updated and adapted, it is important that we code with the future in mind. To do this we prepared the dataset and model so they can both be saved to storage to ease future work. This will save time in the future as only the new data needs to be processed and joined with the existing processed dataset. Then either a new model can be trained, or in lieu of new training data, the previous model can be loaded to perform new predictions. All this code is housed in a notebook which allows the code to be easily reused on demand.

**Tools**

EC2/AWS: Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. It also provides a virtual computing environment (instances). API: Application Programming interface (API) is a software intermediary that allows two software applications to talk to each other.

## **Project Methodology and Process**

In this project, our mission is to build a movie recommendation system using Spark. First, we get the data from the movie, rating file and then parse it into Spark RDDs. Then in order to build our system, we use the alternating least squares (ALS) algorithm to implement collaborative filtering from the PySpark MLlib library. ALS is a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for large-scale collaborative filtering problems. MLib library directly works with the base API of RDD which is the reason we use the textFile function from SparkContext which reads in the file as RDD of strings. RDD is the base fundamental block of Spark data unit on top of which other APIs like dataframe and dataset have been created. RDD usually works the best when we don't want to have structure to input data and want to modify input data by functional transformations. To select specific data from RDD, we use basic RDD functions like filter, take, map along with necessary lambda transformations. As you can see in the Jupyter notebook, we first filter the header and then map each line to tokens of user\_id, movie\_id, and ratings by splitting it with "," delimiter.

By using collaborative filtering, we can predict what movies users might like based on users who have similar tastes in movies from the preference information of other users. In order to get the best ALS parameter, we did hyperparameter tuning with data split into train, validation, and test for 60%, 20%, 20%, and then we got the best\_rank for the model. Then we split the data into train, test for 70%, 30% along with the best ALS parameter, and then get the final trained model. When we get the new user rating, we have to include the new user rating in the full dataset and train the model with the best ALS parameter again. That's because we have to compare new user ratings with other users in the dataset. And after training is done, we can predict the movies which might match the taste of the given user that users haven't rated. Since we have the model and recommendations ready, we can just filter out the top 15 recommended movies with either more than 25/100 ratings based on needs.

**Debugging Details**

Since the difference in size between the full dataset and the small dataset was more than 200x there were difficulties. The instance ran into size constraints when processing the full dataset. That instance would produce an out of storage error (see appendix A2.1) when training the model. There was one change made to help alleviate this error. The Elastic Block Store (EBS) was modified from 30 to 100 gigs. This increase in storage space allowed Spark to process the larger datasets (See Appendix Section A2). Please see the reference below for the article detailing this issue and steps to take in resolving it. After the EBS was increased the training did not throw the “no space left on device” error. No other major issues were found in the process.

# **Results**

Ripe Pumpkins' goal was to determine the likelihood of customer retention through Pumpkin Meter's collaborative recommendations model. To explore the effectiveness of the Pumpkin Meter to provide recommendations, we tested two users, with each rating a total of 10 movies. We generated 15 movie recommendations per user in two scenarios. First, filtering out movies with less than 25 ratings, and second, filtering out movies with less than 100 ratings for a total of 4 cases. We then compared the results of Pumpkin Meter's insight on customer preferences to the initial rating and from user feedback to determine the accuracy and, therefore, how the model may increase the chances of customers staying with Ripe Pumpkins.

User one (see Appendix figure A3.1) completed their ratings using a one-five system, with five being the highest or most liked and one being the least. Both users created ratings with an even distribution of liked and not liked movies using a variety of genres. The following results show encouraging insights on customer preferences and Pumpkin Meter.

First, user one results filtering out movies with less than 25 ratings (see Appendix figure A4.2) shows both accurate and surprising results. For example, only one children's movie, Shrek (2001), was rated highly in the initial rating. However, there was an even mixed result of children's films recommended compared to movies more similar to the other highly rated films. Although there appeared to be a disproportionate number of children's films recommended, the remainder of the recommendations were accurate in their ratings as per the user's feedback.

Next, user one results filtering out movies with less than 100 ratings (see Appendix figure A4.2) shows a more promising depiction of the user's choices. Based on user one's feedback, the vast majority of the films recommended were either movies the user had already seen and enjoyed or were movies they were interested in seeing.

User 2 (see Appendix figures A5.1, A5.2) results from both scenarios have also shown promising insights into user preferences. Interestingly, most of the movie recommendations were old, and the scenario filtering out movies with less than 25 reviews provided recommendations with higher ratings. However, as reported by the user, results from filtering out movies with less than 100 and 25 ratings presented equally relevant and accurate movie options. The user would be interested in watching most of the films recommended from both scenarios.

# **Insights**

The results discussed above reveal a promising insight into users' preferences while using Pumpkin Meter. All four cases provided accurate and reliable movie recommendations, and both users were interested in exploring the suggestions Pumpkin Meter had to offer. However, there were limitations to our testing. Pumpkin Meter tested only two users' inputs for recommendations; therefore, it may be beneficial to consider analyzing more users' results before making a conclusive decision on Pumpkin Meter’s accuracy. Regardless, based on the two users tested, we feel confident that Ripe Pumpkins may benefit from using Pumpkin Meter's collaborative recommendations model.

Based on the results, we have actionable insights that may improve the model's accuracy. For example, the movie recommendation engine forecasted slightly more reliable movies while filtering out anything with less than 100 ratings. Having more ratings per movie may improve the accuracy of each movie's ratings and, therefore, the model's ability to make recommendations based on these ratings. Therefore, ensuring the final recommendation engine follows this same format may be beneficial.

Second, customer tastes change with time due to the nature of trends, a changing environment, or the season. Therefore, we recommend updating the user's preferences as they adjust to ensure the most accurate representation of the user and, consequently, better prediction for movies they may enjoy. To do this, Ripe Pumpkins should make sure Pumpkin Meter is reusing the trained model for each new user and any change in a current user's interests to update preferences.

**Business Implications**

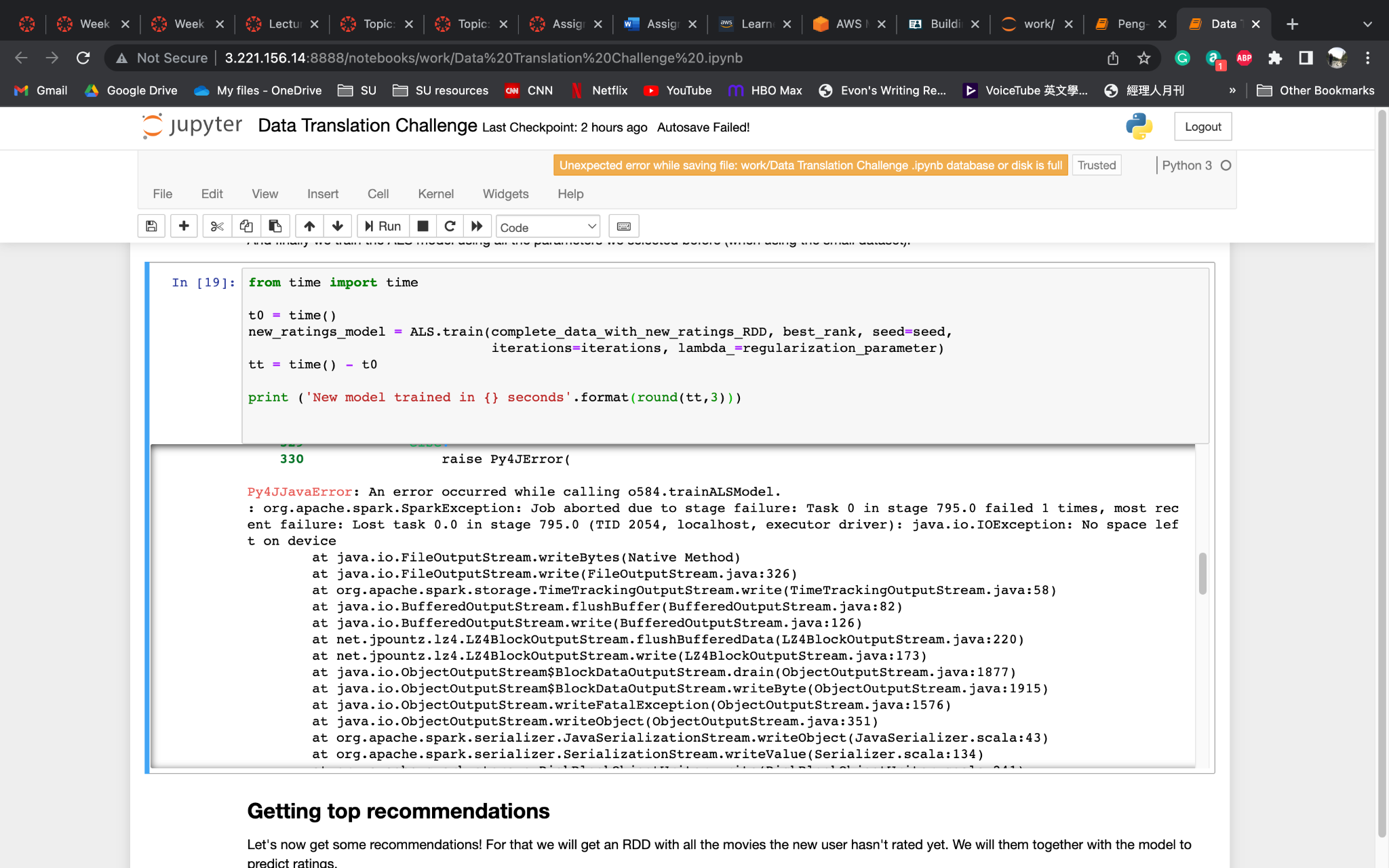
The business implication from a well-performing recommendation engine may increase loyalty and engagement for Ripe Pumpkins' service. In addition, Ripe Pumpkins' ability to predict customers' needs and construct up-to-date recommendations to fulfill those wishes may put them at an advantage over their competition. Therefore, by accurately predicting customers' needs, Pumpkin Meter's model may provide the potential to incentivize future and current customers to stay with the service. Thus, increasing profits for Ripe Pumpkins.

# **Appendix A**

## **A1.** [**Presentation Deck**](https://docs.google.com/presentation/d/1gIndCGoIaP3vgLfHjkXCR8S71TdoMVhjT9ayJs79f1A/edit?usp=sharing)

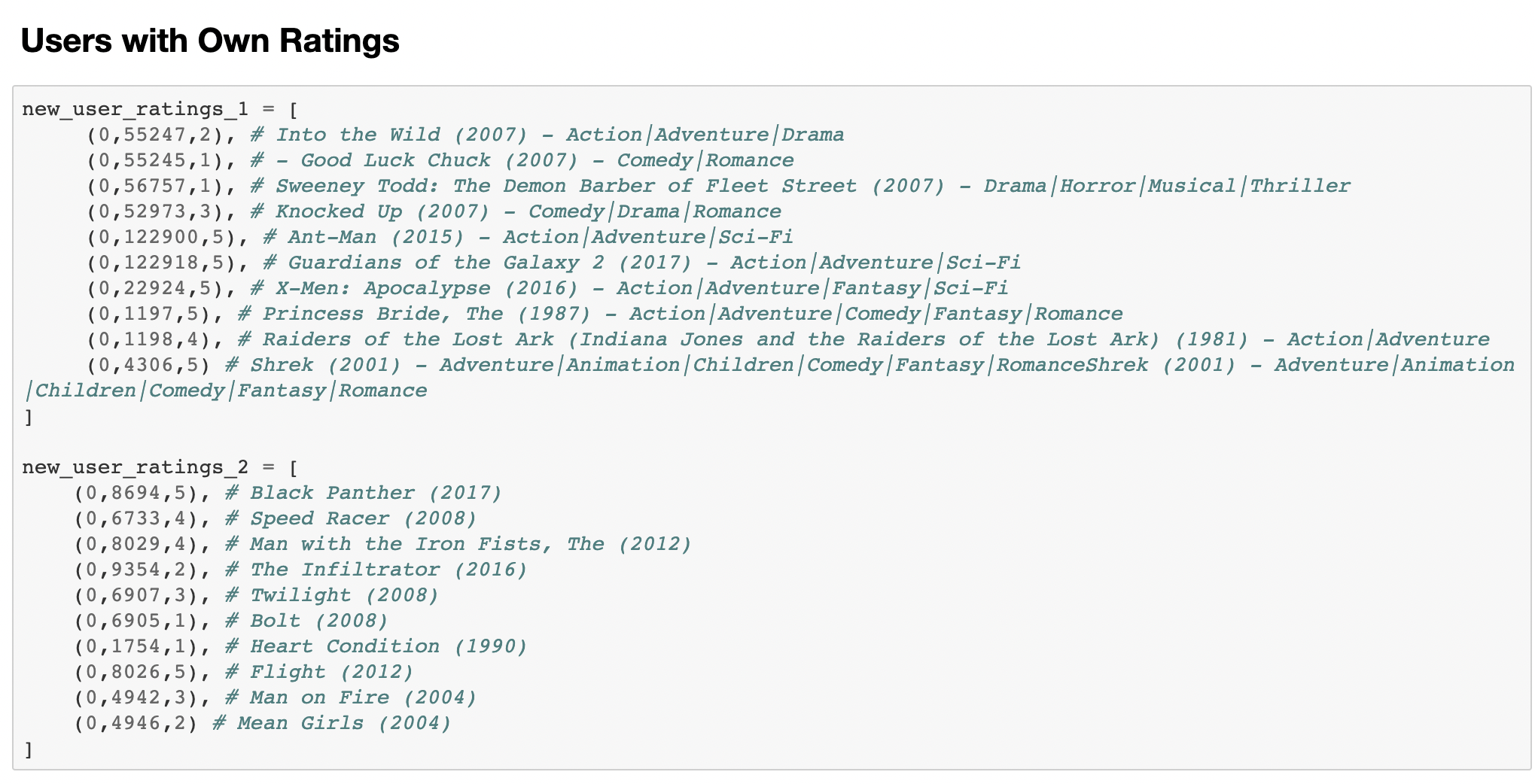
## **A2. Debugging Errors**

Screenshot: While replacing the small dataset to full dataset without adjusting any specification from Amazon EC2 instance, due to the size of dataset changed, the system showed the error message. Below is the screenshot of ‘Py4JavaError: An error occurred while calling….No space left on device’.



A2.1. EC2 Error Screenshot

## **A3 User Ratings**



A3.1. User Rating Inputs from user 1 and user 2.

## **A4. User 1 Results/Returns**

## 

Figure A4.1. Top recommended movies for User 1, with more than 25 reviews.

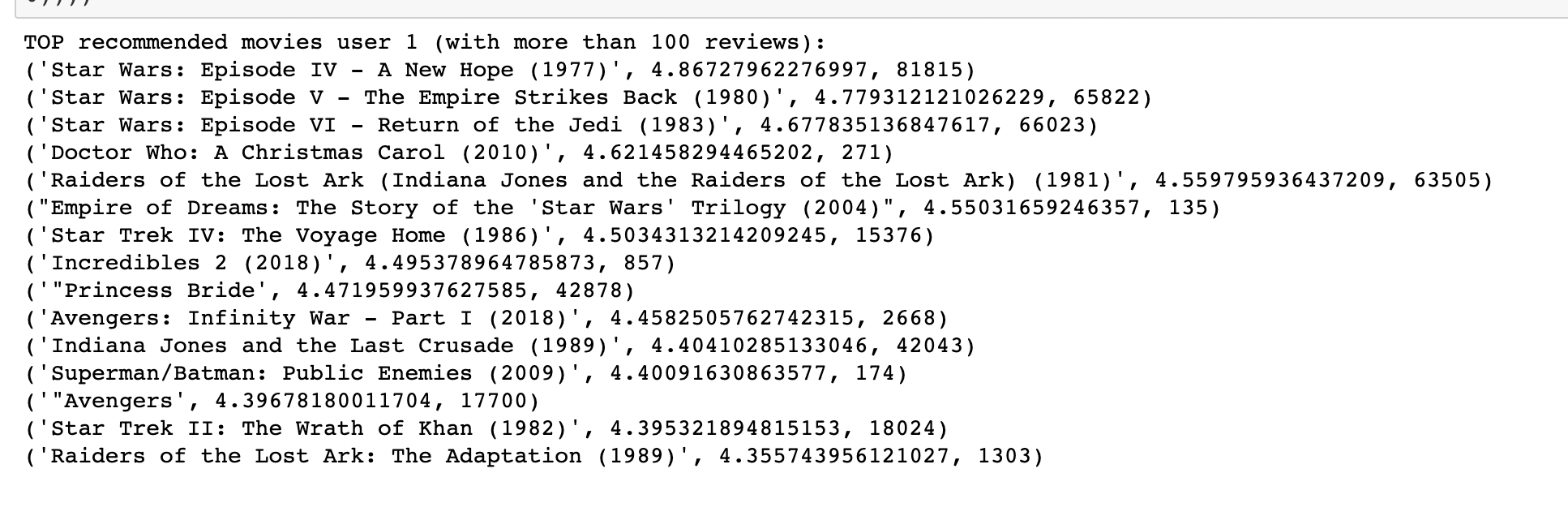


Figure A4.2. Top recommended movies for User 1, with more than 100 reviews.

## **A5. User 2 Results/Returns**

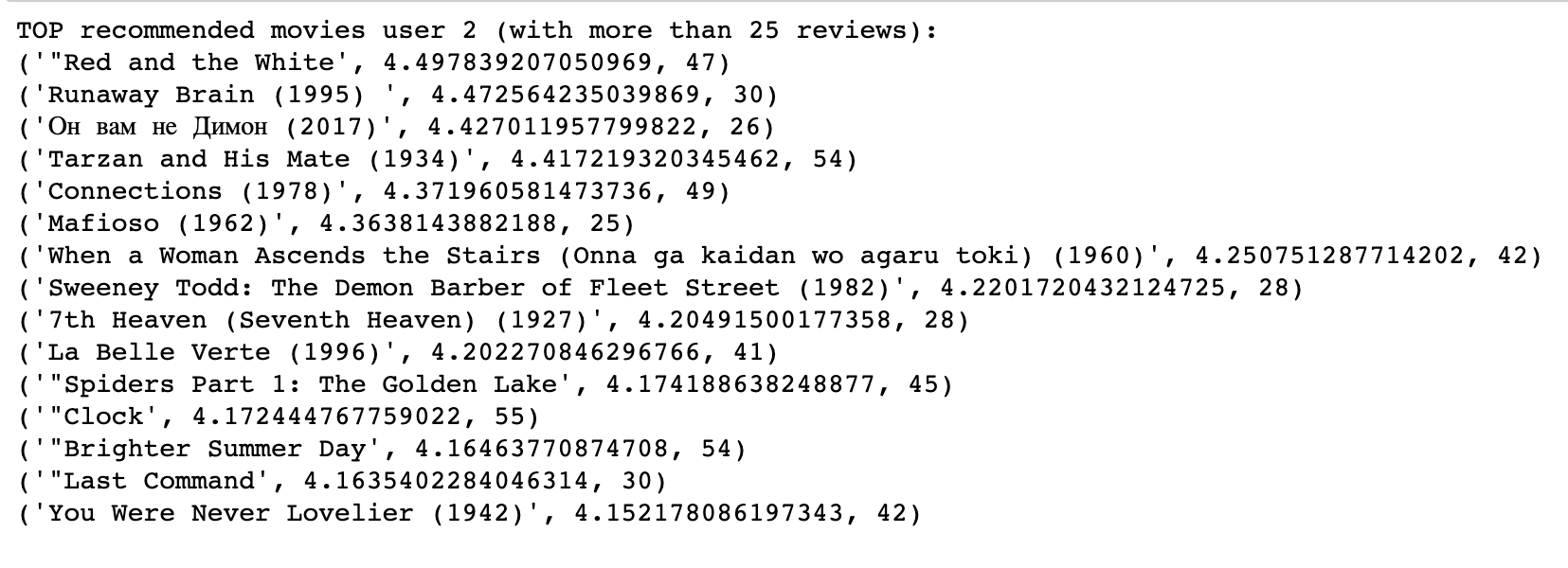


Figure A5.1. Top recommended movies for User 2, with more than 25 reviews.

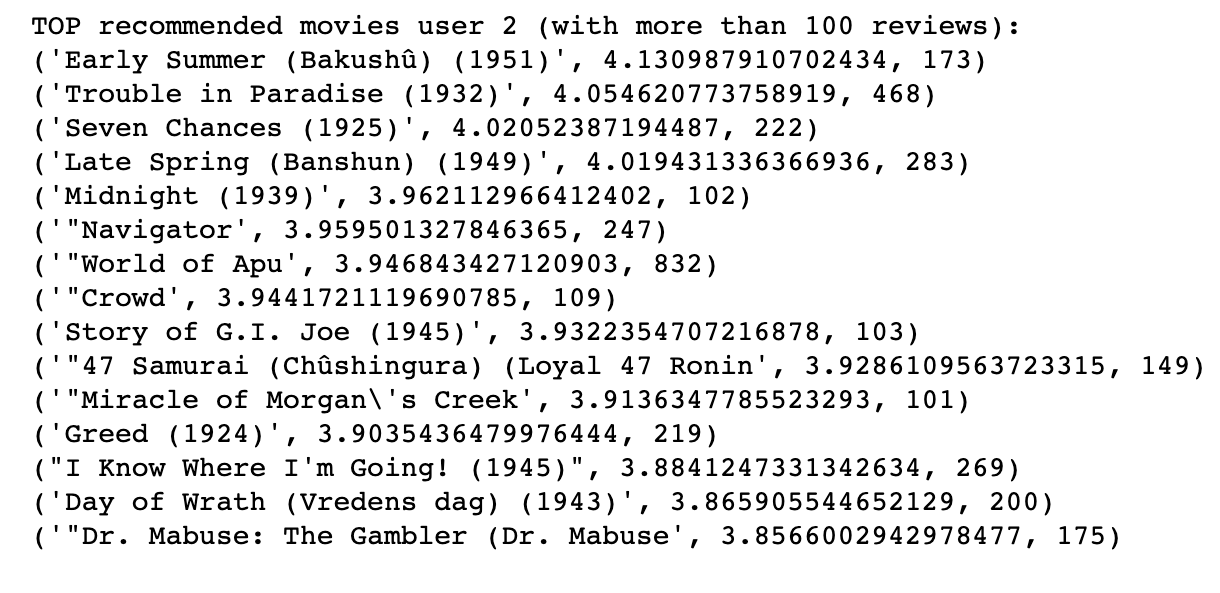


Figure A5.2. Top recommended movies for User 2, with more than 100 reviews.

# References

Amazon. (2022, January 31). *Resolve the "no space left on device" error in spark on ...* AWS. Retrieved March 12, 2022, from https://aws.amazon.com/premiumsupport/knowledge-center/no-space-left-on-device-emr-spark/

Ajitsaria, Abhinav. “Build a Recommendation Engine with Collaborative Filtering.” RealPython, Real Python, 5 June 2021, https://realpython.com/build-recommendation-engine-collaborative-filtering/

Dianes, Jose A. “Building a Movie Recommendation Service with Apache Spark & Flask - Part 1https://www.codementor.io/@jadianes/building-a-recommender%EF%BF%BEwith-apache-spark-python-example-app-part1-du1083qbw

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19.<https://doi.org/10.1145/2827872>