# Release Notes - Book Recommender Repository

Version 1.0

**Project Description:**

We are pleased to announce the release of our book recommendation system repository! This system allows users to receive highly-personalized book recommendations based on their reading history and ratings. To create this system, we utilized data scraping, data preparation, machine learning algorithms, and deployment techniques.

## Initial Data Gathering & Set-up

**Data Prep:**

To obtain the data we required for our recommendation system, we had to create a way to scrap the necessary data. We decided to use tools like Beautiful Soup for HTML parsing and Selenium for web navigation. With these tools we scraped data from the popular book tracking website GoodReads. Some of the data we obtained was the individual books, the details surrounding each book (rating, genre, description, author, etc.), and user ratings. The scraper was able to retrieve approximately 6000 different books, ~60000 individual users, and ~160,000 user ratings and reviews.

**Host Data on MongoDB:**

For data storage and retrieval, we decided to use a MongoDB architecture. This allowed us to deploy our system without having to locally store the large amount of data. Therefore, everytime we needed to access more data or scrap more data we would simply run our programs to automate this process.

**Collect and Aggregate Data:**

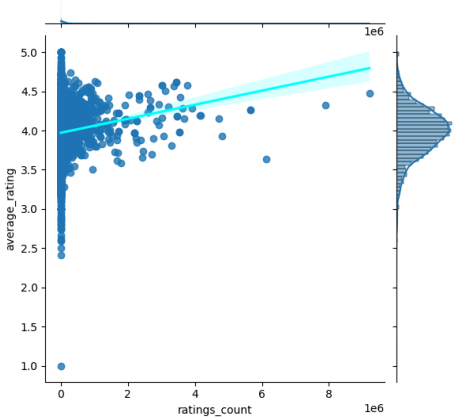
On the MongoDB platform, not only did we store the data we scraped, but also the processed data from the later models and others. In terms of the base data stored, we have three MongoDB collections to create the different book records. These included the BookList which was simply to keep track of the books already scraped and stored, BookReviews to store all the reviews for the given book, and Books to store the general information for the given book.

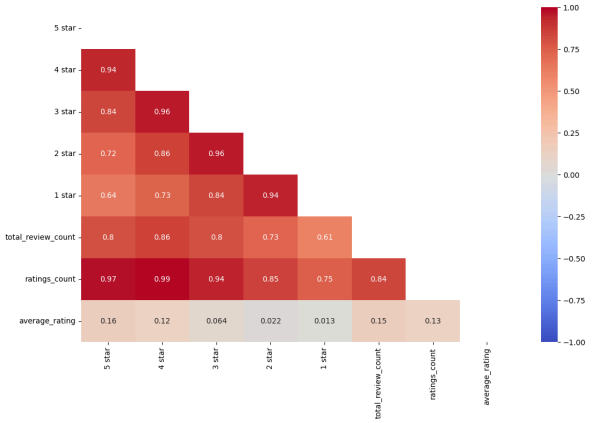
**EDA:**

For EDA we delved deep into the data collections Books and BookReviews. The main objective of the EDA was to:

- Clean & format the unstructured scraped data
- Feature engineer meaningful intricacies into a representation of data.
- Understand the right path for modelling of the recommendation system.

Find the main findings from both EDAs in the following graphs and in the files inside the "notebooks" directory. One of the findings was that in theory, it is possible that the number of ratings a book receives is related to its average rating, meaning that as a book becomes more popular, it receives better ratings. This can be found in the scatter plot above.



# Recommendation System

**Modelling:**

When first beginning, we were unsure exactly of our approach in terms of balancing our time, our access to data, and our interest in terms of what we wanted to accomplish. So we began by first exploring as many avenues as we could, we knew we wanted to construct a hybrid model and knew we would have the data to do it if given enough time.

First, we needed to come up with a solution to a cold-start. In this case we assumed that we could get some users that never read a book or couldn't recall their ratings. We developed a heuristic that took into account the count of ratings and rating distributions of books in order to recommend at the the bes rated books found. Then, we identified a potential need for a recommendation based on a generic theme and/or genre. With the use of NLP techniques like Word2Vec & LatentDirichletAllocation we were able to map generic themes to our genres to recommend the best rated books within that space based on a textual input asking for adjectives of their interest.

Regarding the actual recommendation system - ultimately, we decided on an altered implementation of LightFM. As of right now a user selects the books they have read and provides a rating, this information is then passed to a content to content based recommender which uses cosine similarity. During this stage embeddings are drawn from the title and description of the book as well as some other tricks to capture the hidden meaning in the awards. During this step we also needed to introduce some dimensionality reduction as the embeddings were quickly growing the size of our data. Therefore, we used a Truncated SVD. After the cosine similarity we pass a portion of results, N, to XGboost with the scoring of the similarity and the book’s id. XGboost was trained with its own embeddings and the

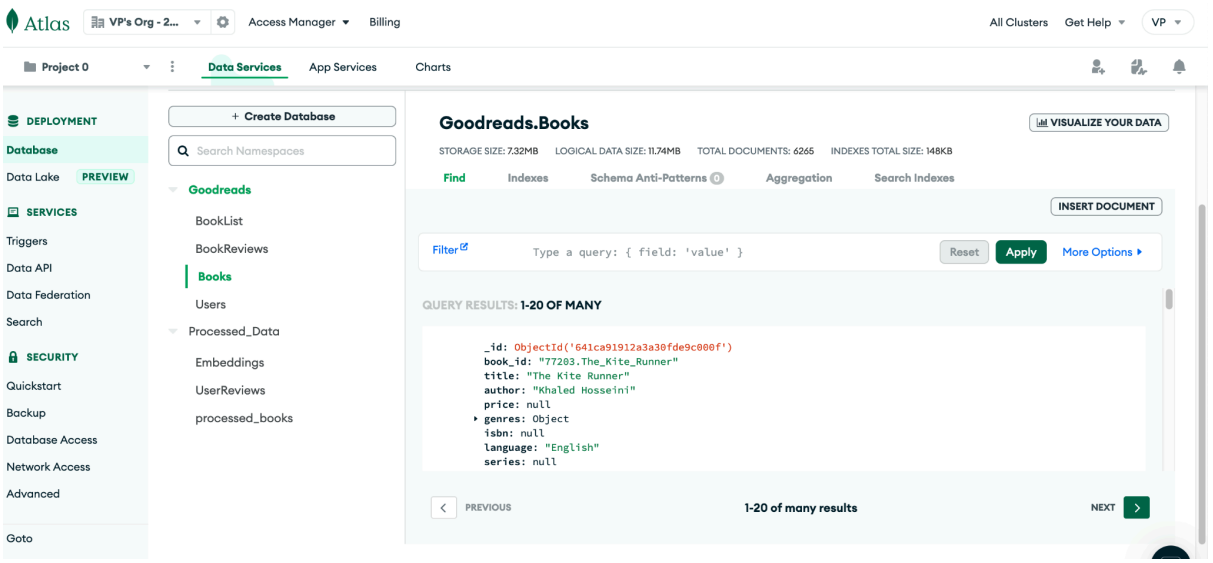
dimensionality was reduced with PCA, the goal of XGboost, in this case, was to provide some semblance of popularity, this was partially performed in cosine similarity as some popularity metrics were included, but what this meant is that if an unpopular books was selected then the recommender would output less popular choices. XGboost is meant to act as a potential solution for this and also allow for improvement moving forward as our future plans would involve the use of our users much more heavily.

One thing we had to circumvent was a sparse selection of user interaction data, we aimed to solve this by using matrix factorization, either ALS or SVD to grab a sort of User Profile for each user. Then to construct a clustering algorithm like KNN around these latent spaces, the goal being that we could pass the results of the cosine similarity and use that as a sort of User Profile then classify that profile to establish popularity. From there we have the essence of LightFM and as such we would combine our components using XGboost to yield a kind of recommendation score by predicting the associated rating of our filtered books.

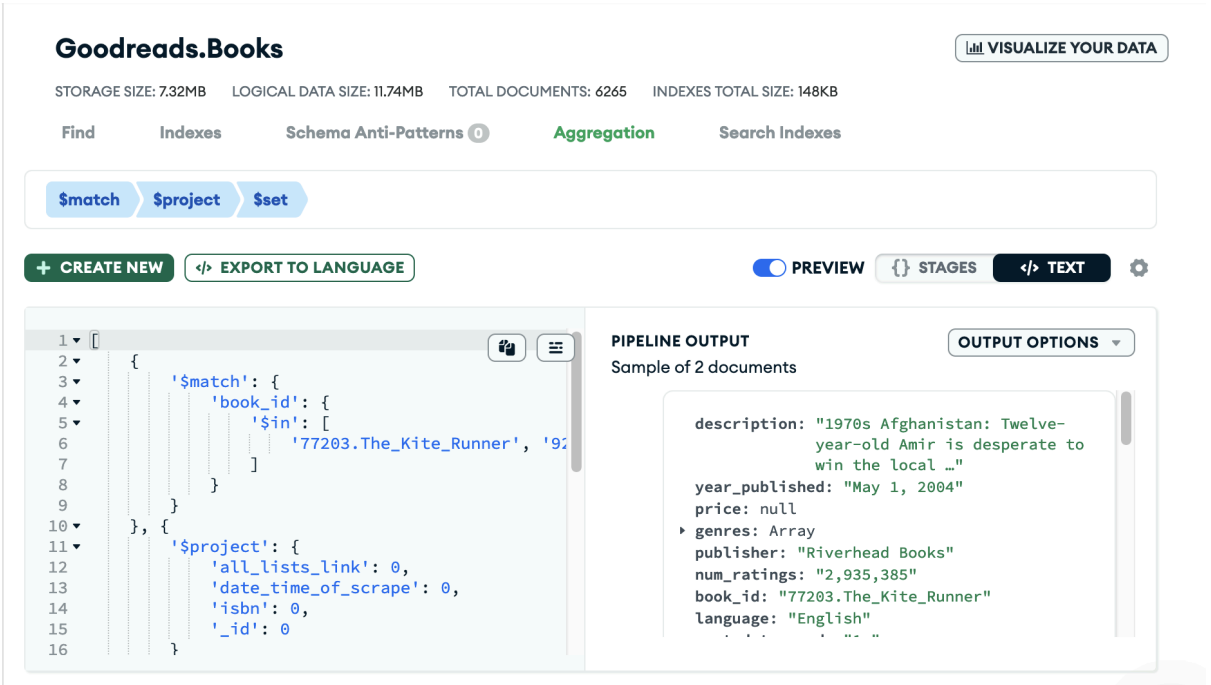
**Deployment & MLOps:**

With our database hosted on MongoDB, we were able to advance our deployment process way more efficiently. We were able to code our recommender system to use MongoDB and simply call the MongoDB handler every time data was needed. This allowed us to deploy the code without having to modify many “hard-coded” data across all the Python files. This process also made creating an application and user interface much easier. The application was coded in React with Flask for the backend to handle the requests to and from. For this, we simply needed to make an API handler that could send requests to and from the MongoDB whenever searching and storing book information as well as when we generate

recommendations. This system architecture proved to be very efficient even with a basic MongoDB implementation.




We also leverage the aggregation pipelines in mongo-db which allow for efficient data pulling from the database.



In order to deploy on Databricks, the ideal scenario would have been to have the recommender in a package. This led us to realise that we had to refactor a lot of the code in order to standardise a preprocessing pipeline, and define a reco-pipeline well. This meant dealing with a lot of technical debt which had accumulated over the past few weeks. Nonetheless it was a good learning experience.

The idea for future implementations is to streamline the process of:

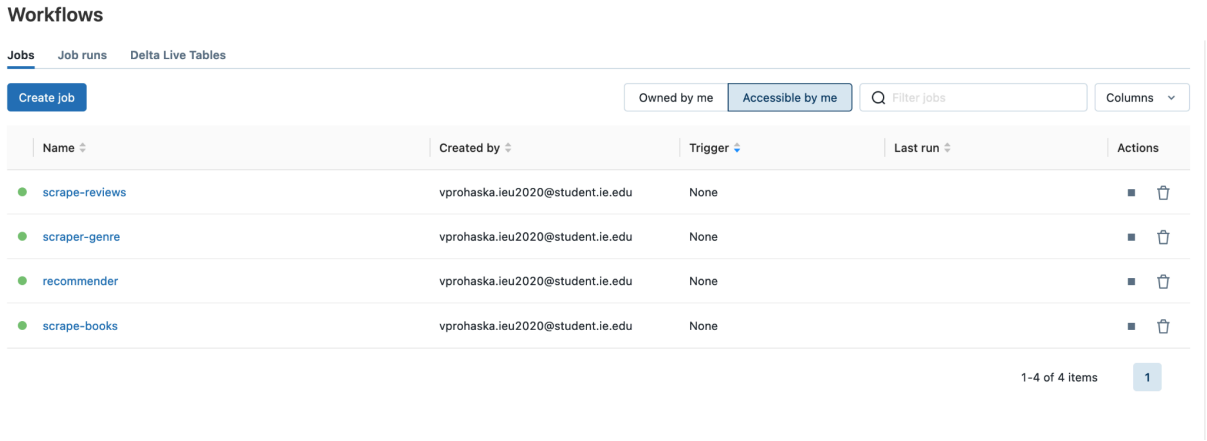
1. Re-training our models based on new data
2. Generating several data streaming pipelines in order to get recommendations in real-time
3. Implement more models/recommendation applications depending on the business need

At this very moment, the recommendation works as follows:

1. User has a list of books they like/have read
2. We get a list of books similar to that of the user based on cosine similarity
3. We use that list and predict ratings for those books
4. We generate a rec_score for each of the books which consists of a combination of a predicted rating and a

This is all packaged in a single python script get_top_k_books.py, running in a cluster in Azure. Furthermore, Databricks is currently set up to run the scrapers automatically, meaning

that we have streamlined the data-collection process.




Within our repo, we also attempt to implement tests, both for the frontend and for the recommendation engine. Furthermore

**Future Work and Improvements:**

While our system has limitations due to data availability and other drawbacks, we plan to further improve and expand the recommendation system. Some of our planned future work includes:

Incorporate More Implicit Feedback: The current system relies on explicit feedback in the form of ratings and reviews. Incorporating implicit feedback, such as user reading history and/or book want-to-read patterns, could provide additional insights into user preferences and help improve the accuracy of recommendations.

Incorporating More Natural Language Processing (NLP) Techniques: Natural Language Processing (NLP) techniques can be used to analyse book reviews, book summaries, or user-generated content, to extract meaningful features and improve the quality of recommendations. NLP techniques, such as sentiment analysis, entity recognition, or topic modelling, can provide insights into user preferences and book characteristics.

Incorporating Social Network Analysis: Social network analysis techniques can be used to analyze the relationships and interactions among users in the system. This can be easily applied as GoodReads has its own Social Network.  By considering social connections between users, we can leverage the social influence and recommendations from friends or similar users to provide more accurate and effective recommendations.

Incorporating a Knowledge Graph:

1. Define entities and relationships: The first step in building a knowledge graph for book recommendations is to define the entities and relationships that will be included. Entities could include books, authors, publishers, genres, and more. Relationships could include things like "written by," "published by," "belongs to genre," and "recommended for."
2. Populate the knowledge graph with data: Once the entities and relationships are defined, the knowledge graph can be populated with data. This could include information about books, such as title, author, genre, and publisher, as well as information about readers, such as their reading history and preferences.