Developer’s Manual

Book Recommendation System

Nadine Thomas, Kacie Myers, and Gracie Lovell

Project Overview

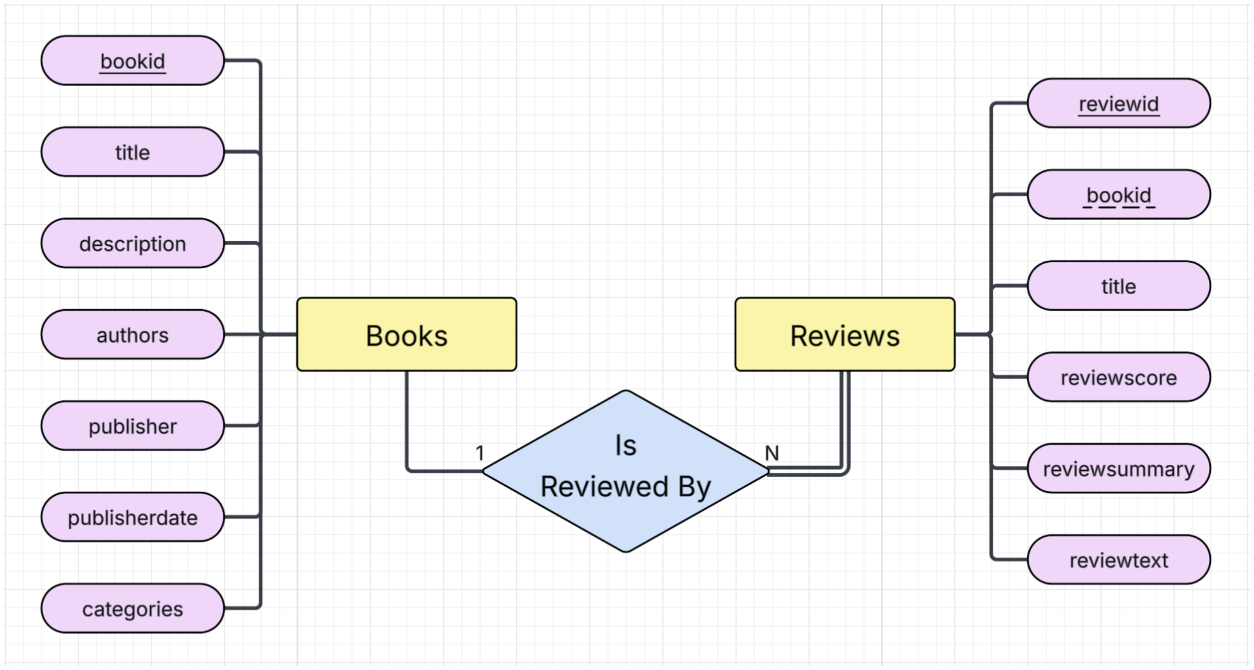
This project develops a book review analysis and recommendation system by combining database management with text analysis. We store book and review data in a MySQL database, linking tables to integrate metadata such as title, author, and rating with review text. Using Python, we preprocess and analyze the review text with techniques like sentiment analysis, keyword extraction, and Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to capture patterns in reader opinions beyond star ratings. The system calculates similarity between books based on review content, with optional weighting for positive reviews, and generates personalized recommendations while avoiding duplicate titles and same-author suggestions. By integrating database queries, text mining, and a recommendation engine, the project provides actionable insights into common themes, sentiment trends, and book-to-book similarities, ultimately offering users data-driven book recommendations.

Original Dataset

The original dataset was taken from a popular website designed to develop models, access datasets, and share code among other data scientists and machine learning practitioners called Kaggle. The data was posted by Mohamed Bekheet in a repository called Amazon Book Reviews and includes two separate CSV files. The first one is called Books_ratings.csv, and it contains information on about 3 million book reviews for around 212,000 unique books. The second file is called books_data.csv, and information about each of the 212,000 books that are listed in the first file. However, there were some fields in both files that were deemed unnecessary for our project, such as the price of the book or a link to its preview. Some rows were found to be null, so they were removed from our dataset as well.

SQL Database

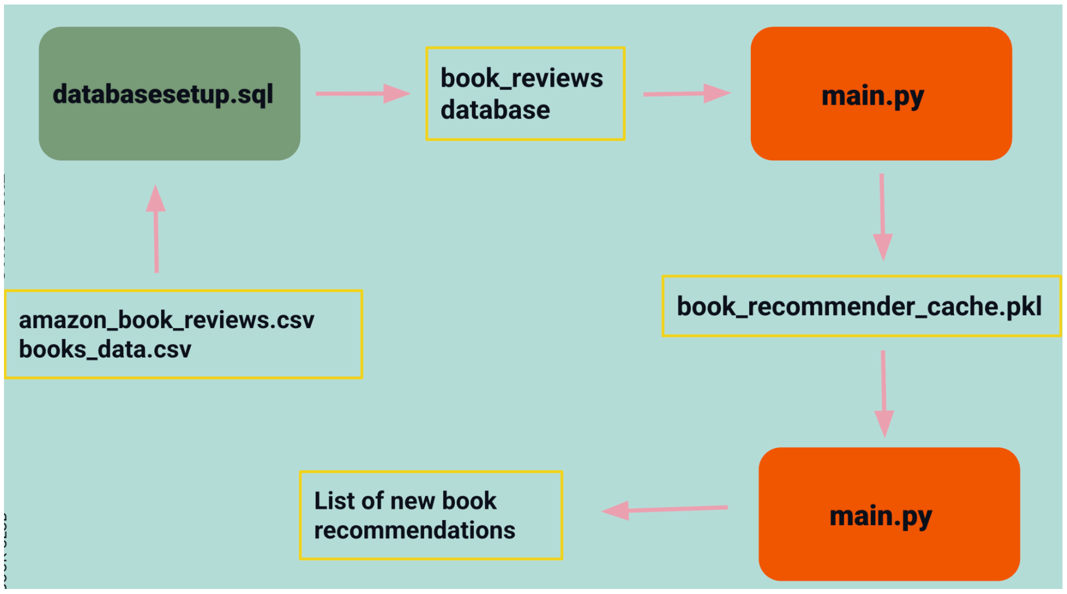
After completing our pre-analysis and cleaning of the dataset, we now have two CSV files that we can use to complete our project. The first file we called amazon_book_reviews.csv, and it contains around 1.5 million reviews. This file contains information such as title, the score given in the review, the summary of the review, and the text written in the actual review. The second file we called books_data.csv. It contains around 212,000 unique books. This file contains information such as title, the description of the book, authors, publishers, the date it was published, and the categories/genres of the book. We then used these files to create our database on MySQL. Our database contains two tables called books and reviews. An ER diagram of our database is shown below.



To make accessing the data in the database easier, we added two ID fields to our tables called bookid and reviewid. These IDs are used as primary and foreign keys to facilitate the relationship between these two tables. The relationship between books and reviews is a one-to-many relationship because one book can have many reviews; however, each review belongs to only one book.

Project Structure

The structure of our project is shown in the following picture. The source code was coded in Python with pandas, scikit-learn, collections, re, SQLAlchemy, pickle, and os libraries, and can be found on GitHub.



Project Modules

Creating Book Profiles

Each book in the system is represented by a single aggregated document known as the book profile. This profile is generated by combining all available reader reviews with the book's description and its assigned categories or genres. To emphasize meaningful themes, the system applies weighted aggregation: words that appear in highly rated reviews (ratings greater than 4) are given additional weight, allowing positive, high-quality feedback to influence the profile more strongly. This weighted book profile serves as the core text representation used for similarity comparisons and recommendation generation.

Quantifying the Text

To transform each book profile into a numerical representation, the system applies Term Frequency–Inverse Document Frequency (TF-IDF). TF-IDF assigns a score to every word based on how often it appears within a single book's profile relative to its rarity across all books in the database. Words that are both frequent in one book and uncommon in others receive higher scores, making them strong identifiers of that book's unique themes. This process highlights the most descriptive and distinguishing keywords, enabling more accurate similarity comparisons and recommendation modeling.

Finding Similarity

To identify related titles, the system compares the numerical vectors of a selected book against those of all other books using cosine similarity. This metric evaluates the angle between two vectors, where smaller angles (scores close to 1.0) indicate strong similarity due to shared core keywords, and larger angles (scores approaching 0.0) reflect substantial differences in content. By computing these vector relationships, the system produces a similarity score for every book in the database relative to the source book, forming the basis for recommendation ranking.

Refining the Recommendations

To improve recommendation accuracy, the system applies several filtering and adjustment steps before finalizing results. A quality filter removes books with fewer than 20 reviews to ensure that each book profile is supported by sufficient data. A diversity filter excludes books by the same author as the source title, preventing repetitive recommendations such as sequels or companion novels. Finally, a category boost slightly increases the similarity score for books that share the same primary genre or category, allowing the system to maintain thematic relevance while still encouraging variety in the recommendations.

Bekheet, Mohamed. "Amazon Books Reviews." *Kaggle*, www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews.