Final Project Report

Link to Git Repository: https://github.com/jaslual/Final-Project

Original Goals for Open Library API: Originally, we wanted to use the Open Library API to determine the amount of books that had the same genre, and our focused genre for this was going to be 'romance'. The graph was going to show the amount of authors gathered from the API with their relation to books that had the 'romance' genre.

Original Goals for Gutendex API: Originally, the plan was to use Penguin Publishing as the second API, however after further research it was clear that the Gutendex API would be more efficient. The data from Penguin Publishing that was meant to be collected was the authors from the books. The calculation was going to be what author had written the most books and determine if that author wrote those books within the same genre. The graph was going to be a scatterplot that showed how well books of different genres did when published by a specific author.

Goals Achieved for Open Library API: We were able to achieve all of our previously listed goals. We were able to gather 100 rows of data from the API displaying the authors that had written books within the 'romance' genre.

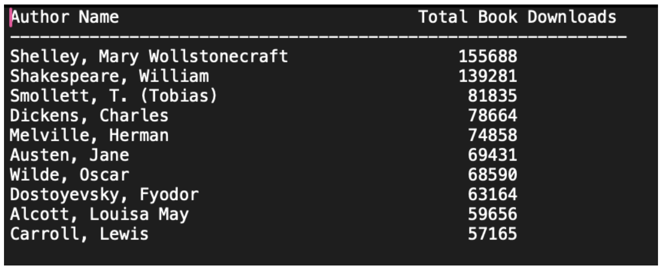
Goals Achieved for Gutendex API: After changing the API to Gutendex, the goal was still to collect the author data from the books in the API. This goal was achieved and along with also collecting the titles and download count for each book. Two tables were created in the database for Authors and Titles with the Titles table having the authorID as its foreign key in order to connect those tables. This was also important so the download count could be correctly calculated for each author, especially if they had multiple books. The top 10 authors with the most book downloads were calculated and a dot plot was created to help show this.

Problems faced for Open Library API: One specific problem faced with the API was being able to pull more data. At first, the part of the code pulling data from the API only printed out 12 books of the 'romance' genre, and it would repeat those same books over and over with each run of the code.

Problems faced for Gutendex API: The main problem that occurred originally was figuring out which API to use and which one would be able to meet our goals the best. It was also difficult figuring out the overall structure of the code and how to limit the data stored from the API to 25 or less items each time the code was run.

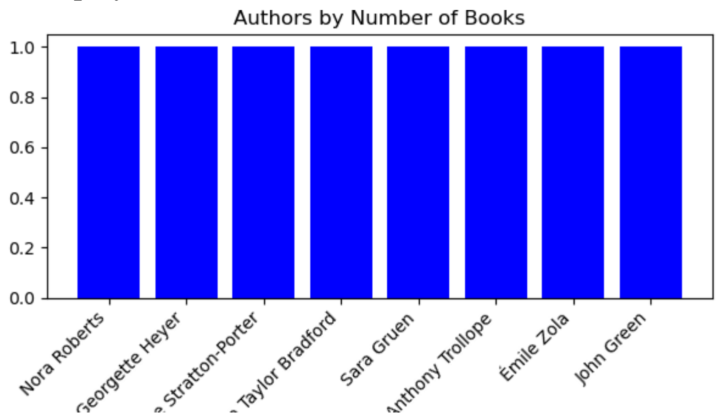
Calculations from Open Library Data: The calculation from the Open Library API was counting how many authors and books within the API's database were connected with the genre 'romance'. We were able to collect 119 rows of data in SQLite.

Calculations from Gutendex Data: The calculation from the Gutendex API was counting how many book downloads each author had overall and then displaying the top 10 from highest to lowest.

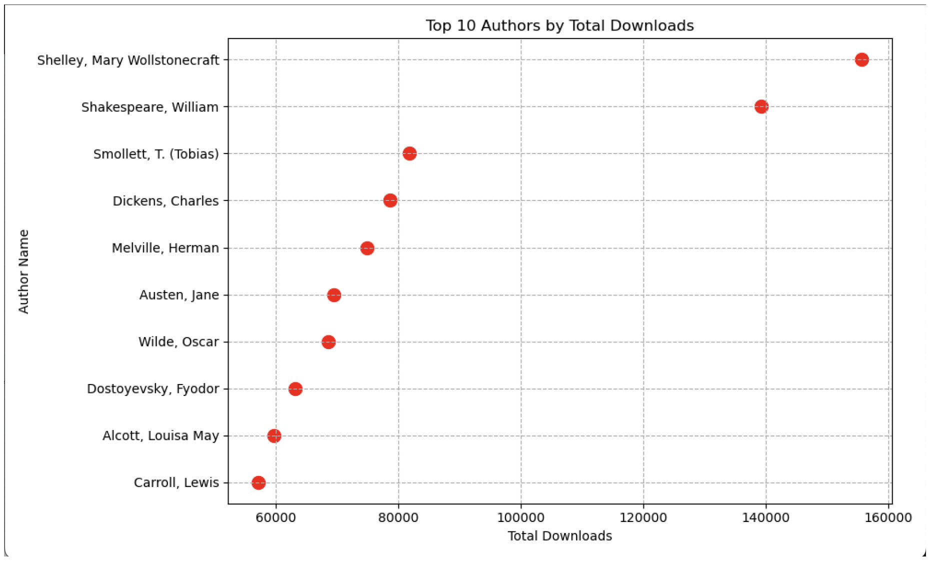


```
Author Name                                 Total Book Downloads
------------------------------------------------------------
Shelley, Mary Wollstonecraft                    155688
Shakespeare, William                            139281
Smollett, T. (Tobias)                            81835
Dickens, Charles                                 78664
Melville, Herman                                 74858
Austen, Jane                                     69431
Wilde, Oscar                                     68590
Dostoyevsky, Fyodor                              63164
Alcott, Louisa May                               59656
Carroll, Lewis                                   57165
```

Visualizations from Open Library API: The visualization for the Open Library API was meant to display the amount of books authors had within the 'romance' genre.



Visualizations from Gutendex API: The visualization was a dot plot showing the top 10 authors with the highest book downloads.



Instructions for running Open Library and Gutendex code: Press run on OpenLibrary.py and Gutendex.py to add tables and data to the database. Press run on gutendex_calculations.py then gutendex_visualization.py for data calculation and graph creation. Press run on OL_visualization.py for data calculation and graph creation. No additional instructions are needed.

Documentation for Open Library code:

For OpenLibrary.py:
The **setup_database** function is a set up to store the data we collect about the books. It connects to the final_project database and creates two tables, authors and books.
The **clear_database** function, that is commented out, was meant to clear the data from a previous run. This was a solution given to us by ChatGPT to fix the issue of repeated data, but we ended up not needing it anymore, which is why it is commented out.
The **get_last_offset** function checks if there is a file named last_offset.txt that keeps track of how far I've gotten in fetching books while the **set_last_offset(offset)** function updates last_offset.txt once I have fetched books. It also writes the new offset into the file so I can pick up from there when I run the code again.
The **fetch_and_store_books** function is the main function of the entire code, as it is responsible for fetching books of a specific genre (romance) from the API and storing it into the database. When it fetches, it grabs the title and author's name, and, when it stores, it checks if the book already exists in the book table to avoid duplicates and then puts it into the database.
The **get_books_from_database** function pulls books from our database that are linked with a specific genre and returns the results as a list of tuples. It also joins the books and authors tables and fetches both the title and author name for books that match the genre 'romance'.

The **if__name__ == "__main__"** function calls setup_database then it sets the genre to 'romance' and calls fetch_and_store_books, which fetches the books. Then, it gets the stored books from the database for the genre 'romance' and prints them in list format.

For OL_visualization.py:
The **visualize_data** function is meant mostly for error checking. If there are no books or authors to show, it prints the error message and stops the function. Then, it writes the author name and their book counts to the file authorsXgenre.txt. The file lists the author followed by the number of books they've written.
The **process_data** function fetches and processes the data from our database and creates the visualization, in the form of a bar graph using the imported matplotlib.

Documentation for Gutendex code:

For Gutendex.py:
The **create_tables** function connects to our database, creates 2 tables called Authors and Titles. The Authors table stores unique Author IDs and names. The Titles table stores unique Title IDs, title names, download counts, and makes the Author ID the foreign key so each title links to an author. It then commits those changes and closes the database connection. There are no specific inputs or outputs, it just creates the tables if they do not exist.
The **get_next_author_id** function helps determine the next available ID for a new author in the Authors table. It then checks for the current highest ID in the table and returns the next ID. If the table is empty it will return 1, if not then it will return the highest ID and add 1. There is no specific input, but once again will output the next available author ID as an integer.
The **fetch_books** function inputs the Gutendex API URL and outputs a list of books and the next page's URL in the API or None if no pages are left. If an error were to occur, it will print an error message and return an empty list for books and None for the next page.
The **store_data** function will input a list of books that have already been fetched and the starting URL for fetching the books. It will output the URL for the next page of books. It uses the fetch_books function to get books from the API, by starting with the next URL. It checks if a title or author is already in the database and if not it will add 25 authors and 25 titles to the database each time the code is run. It will print the amount of authors and titles stored after each run to ensure it was only 25.
The **main** function calls the create_tables function, starts fetching data from the API, calls the store_data function, and updates the next page URL.

For gutendex_calculations.py:
The **calculate_top_author_downloads** function does not have any specific input and outputs a list of the top 10 authors with their respective download counts. It combines data from the

Authors and Titles table and joins them based on their Author ID and Title ID. Once the information is gathered, it returns a list of tuples.
The **write_to_file** function inputs the list of tuples containing the top 10 authors and their download counts and the total_author_downloads.txt. It then outputs the text file with the formatted results. It properly writes the column names, separator line, and the data itself into the text file.
The **main** function calls the calculate_top_author_downlods function and the write_to_file function.

For gutendex_visualization.py():
The **read_data_from_file** function inputs the total_author_dowloads.txt file and outputs a list of tuples which contains the authors name and their total download count. It opens the file, skips the header information, and treats the first 40 characters as the author's name with no extra spaces with the rest being treated as the download count as in integer.
The **create_dot_plot** function inputs the list of tuples where each one contains the authors name and their total download count. It will output a dot plot showing all of the total downloads for the top authors.
The **main** function calls the read_data_from_file function and calls the create_dot_plot function.

Resource Documentation:

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/2/2024	Problems with structure and limiting data to 25 items	ChatGPT	It resolved the issue
12/5/2024	Problems with creating visualization and data selection	ChatGPT	It resolved the issue
12/11/2024	Problems with debugging and final database structure	ChatGPT	It resolved the issue