Alyssa Ma

September 3, 2020

ETL Project Report

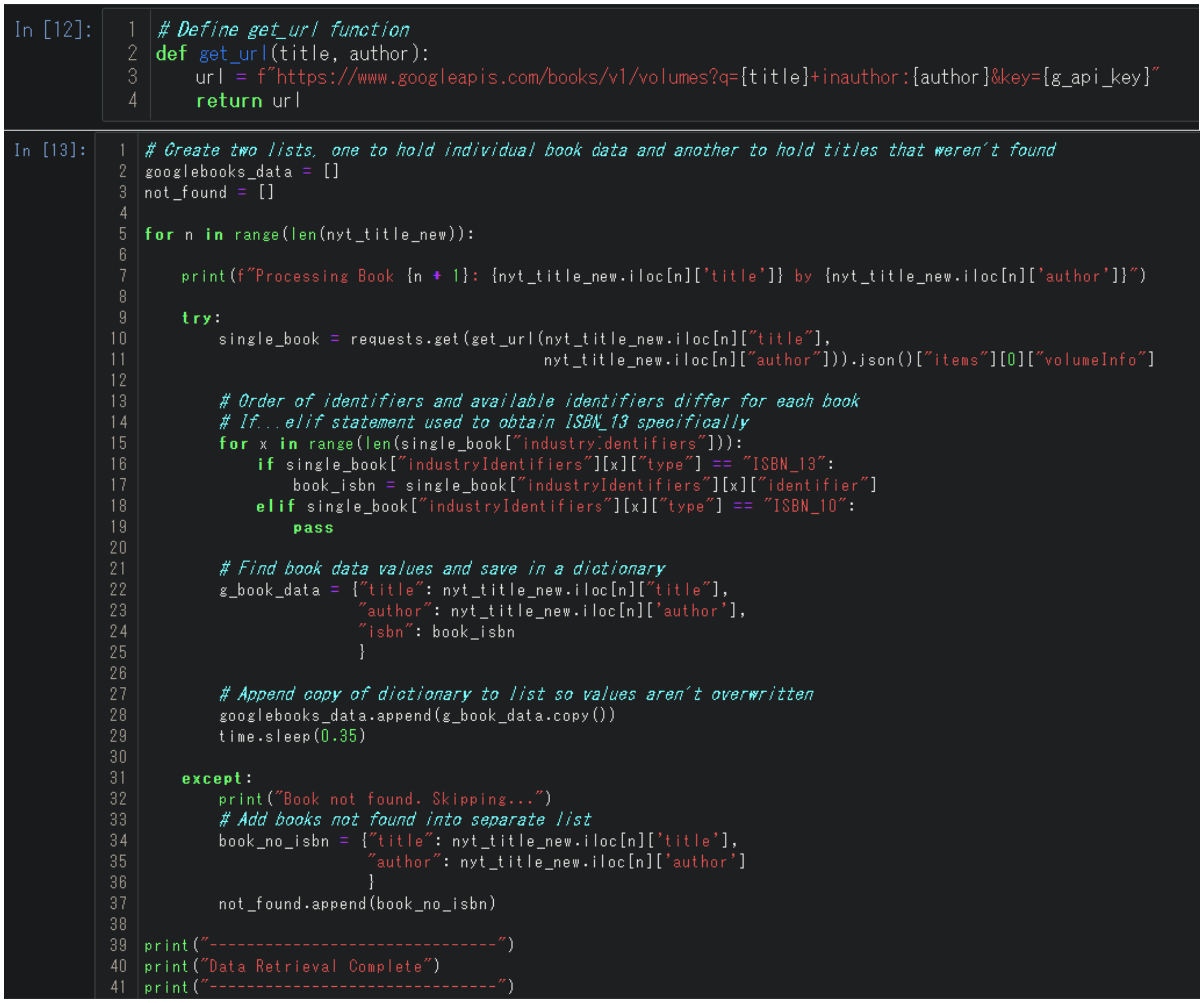
Extraction

For my project, I first looked at a CSV file from data.world regarding New York Times Bestsellers from 2011 to 2018 and the number of weeks each book was on the list (source: https://data.world/typhon/new-york-times-bestsellers-from-2011-to-2018). The original file provides 2,249 rows and 14 columns of data, the columns being "publisher," "dagger," "book_review_link," "author," "primary_isbn10," "price," "primary_isbn13," "sunday_review_link," "date," "first_chapter_link," "contributor," "title," "age_group," and "weeks_on_list." This file was directly loaded into a dataframe in Jupyter Notebook with pd.read_csv.

I also utilized three APIs in this project: Google Books, ISBNdb, and Goodreads. From the first two APIs I extracted ISBN13 information, while the Goodreads API was used to obtain the average rating, total number of ratings, and total number of text reviews on the site for each book. Each API request returned a response in JSON format, which was then examined to extract the necessary data.

I originally intended to use the ISBN13 column in the CSV file with the Goodreads API, but upon further examination, I found that the provided ISBN13 numbers only contained the first three digits with the remaining ten all zeroes. I reviewed the ISBN10 column as well but soon discovered that 631 records had "None" as an entry and several of the other provided ISBNs were not connected to book titles on Goodreads. Thus, I investigated other possible ways to obtain the ISBNs and decided that the Google Books API and ISBNdb API would be best. Unfortunately, due to API data restrictions, I had to limit the number of book titles for this project, which I will discuss in the next section.

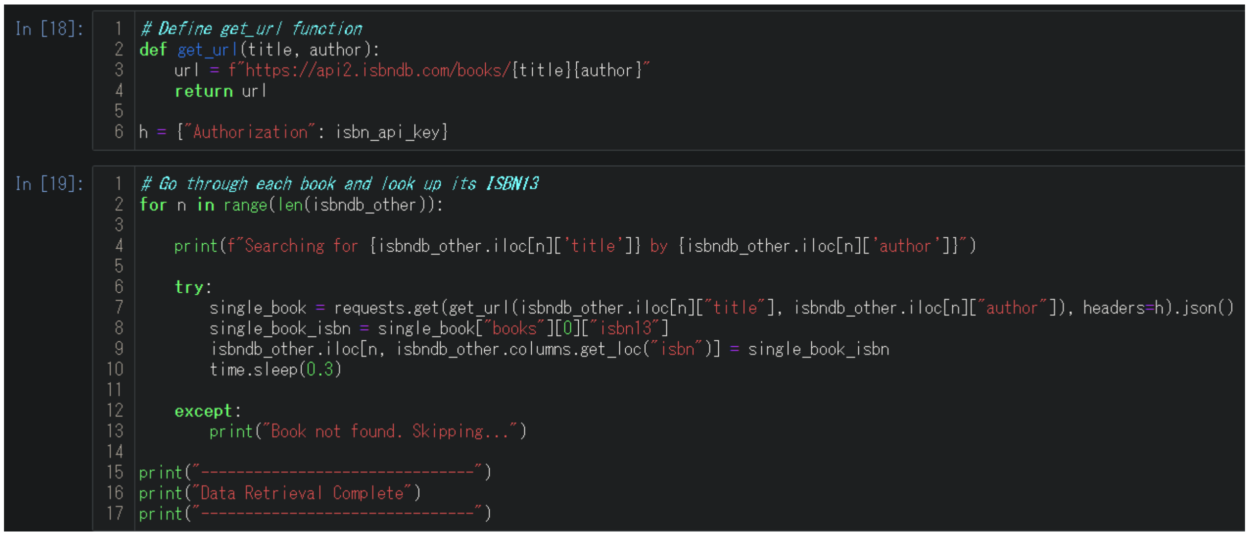
For the Google Books API, I utilized both the book title and author to create valid URLs for API requests. Two lists were made, one called "googlebooks_data" to hold individual book data and another called “not_found” to hold titles that were not found as each book was looped through. Due to variability in the order and available identifiers (ISBN_13, ISBN_10, etc.), another for loop was added to select ISBN_13 only. This number was saved in a dictionary along with the title and author, and a copy of the dictionary was then appended to the googlebooks_data list. In cases where a book was not found, the book’s title and author were added to the not_found list for further follow-up.



```
In [12]:
# Define get_url function
def get_url(title, author):
    url = f"https://www.googleapis.com/books/v1/volumes?q={title}+inauthor:{author}&key={g_api_key}"
    return url
```

```
In [13]:
# Create two lists, one to hold individual book data and another to hold titles that weren't found
googlebooks_data = []
not_found = []

for n in range(len(nyt_title_new)):

    print(f"Processing Book {n + 1}: {nyt_title_new.iloc[n]['title']} by {nyt_title_new.iloc[n]['author']}")

    try:
        single_book = requests.get(get_url(nyt_title_new.iloc[n]["title"],
                                           nyt_title_new.iloc[n]["author"])).json()["items"][0]["volumeInfo"]

        # Order of identifiers and available identifiers differ for each book
        # If...elif statement used to obtain ISBN_13 specifically
        for x in range(len(single_book["industryIdentifiers"])):
            if single_book["industryIdentifiers"][x]["type"] == "ISBN_13":
                book_isbn = single_book["industryIdentifiers"][x]["identifier"]
            elif single_book["industryIdentifiers"][x]["type"] == "ISBN_10":
                pass

        # Find book data values and save in a dictionary
        g_book_data = {"title": nyt_title_new.iloc[n]["title"],
                       "author": nyt_title_new.iloc[n]['author'],
                       "isbn": book_isbn
                       }

        # Append copy of dictionary to list so values aren't overwritten
        googlebooks_data.append(g_book_data.copy())
        time.sleep(0.35)

    except:
        print("Book not found. Skipping...")
        # Add books not found into separate list
        book_no_isbn = {"title": nyt_title_new.iloc[n]['title'],
                        "author": nyt_title_new.iloc[n]['author']
                        }
        not_found.append(book_no_isbn)

print("------------------------------")
print("Data Retrieval Complete")
print("------------------------------")
```

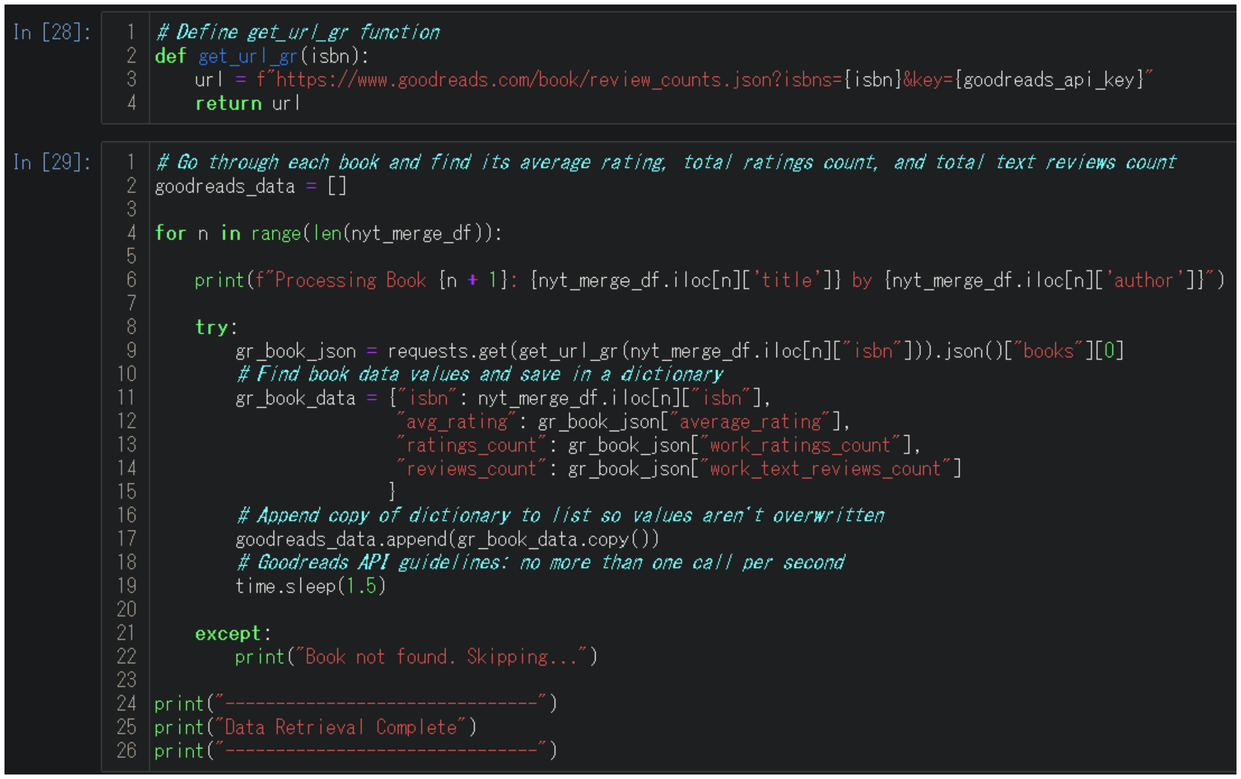
Each list was then put into a dataframe— “googlebooks_data” into the “googlebooks_df” dataframe and “not_found” into the “isbndb_other” dataframe. In this case, a total of three books were not found. These books were then run a second time through the ISBNdb API to fetch ISBN13 numbers.



```
In [18]:
# Define get_url function
def get_url(title, author):
    url = f"https://api2.isbndb.com/books/{title}{author}"
    return url

h = {"Authorization": isbn_api_key}
```

```
In [19]:
# Go through each book and look up its ISBN13
for n in range(len(isbndb_other)):

    print(f"Searching for {isbndb_other.iloc[n]['title']} by {isbndb_other.iloc[n]['author']}")

    try:
        single_book = requests.get(get_url(isbndb_other.iloc[n]["title"], isbndb_other.iloc[n]["author"]), headers=h).json()
        single_book_isbn = single_book["books"][0]["isbn13"]
        isbndb_other.iloc[n, isbndb_other.columns.get_loc("isbn")] = single_book_isbn
        time.sleep(0.3)

    except:
        print("Book not found. Skipping...")

print("------------------------------")
print("Data Retrieval Complete")
print("------------------------------")
```

The final extraction process put the gathered ISBN13 numbers through the Goodreads API to obtain average rating, total ratings count, and total text reviews count, once again saving the book data into a dictionary that was then appended to the list "goodreads_data."

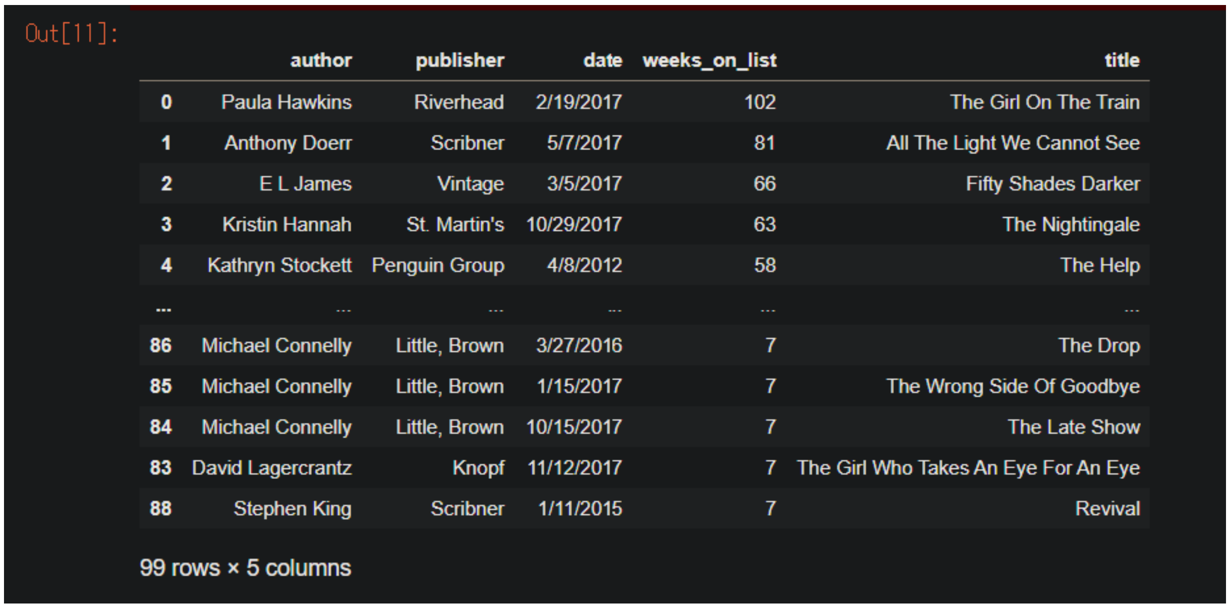


```
In [28]:
# Define get_url_gr function
def get_url_gr(isbn):
    url = f"https://www.goodreads.com/book/review_counts.json?isbns={isbn}&key={goodreads_api_key}"
    return url
```

```
In [29]:
# Go through each book and find its average rating, total ratings count, and total text reviews count
goodreads_data = []

for n in range(len(nyt_merge_df)):

    print(f"Processing Book {n + 1}: {nyt_merge_df.iloc[n]['title']} by {nyt_merge_df.iloc[n]['author']}")

    try:
        gr_book_json = requests.get(get_url_gr(nyt_merge_df.iloc[n]["isbn"])).json()["books"][0]
        # Find book data values and save in a dictionary
        gr_book_data = {"isbn": nyt_merge_df.iloc[n]["isbn"],
                        "avg_rating": gr_book_json["average_rating"],
                        "ratings_count": gr_book_json["work_ratings_count"],
                        "reviews_count": gr_book_json["work_text_reviews_count"]
                        }
        # Append copy of dictionary to list so values aren't overwritten
        goodreads_data.append(gr_book_data.copy())
        # Goodreads API guidelines: no more than one call per second
        time.sleep(1.5)

    except:
        print("Book not found. Skipping...")

print("------------------------------")
print("Data Retrieval Complete")
print("------------------------------")
```

Transform

To clean the CSV file, I began by checking for and dropping any duplicates (none were found) and selecting only five relevant columns (“title,” “author,” “publisher,” “date,” “weeks_on_list”). I also replaced the “â€™” found in a few of the titles with an apostrophe and sorted the dataframe by “weeks_on_list” in descending order. Due to certain API limits, I decided to look at New York Times bestsellers that were on the list for seven weeks or more, resulting in a total of 99 books. A new column “title_case” was added containing book titles formatted with capwords() to prevent the capitalization of the letter after an apostrophe as when using title(). The “title” column with book names in their original all-caps formatting was dropped, and the “title_case” column was renamed “title.” This “nyt_title_new” dataframe was later paired with the Google Books API to obtain ISBNs.



Out[11]:

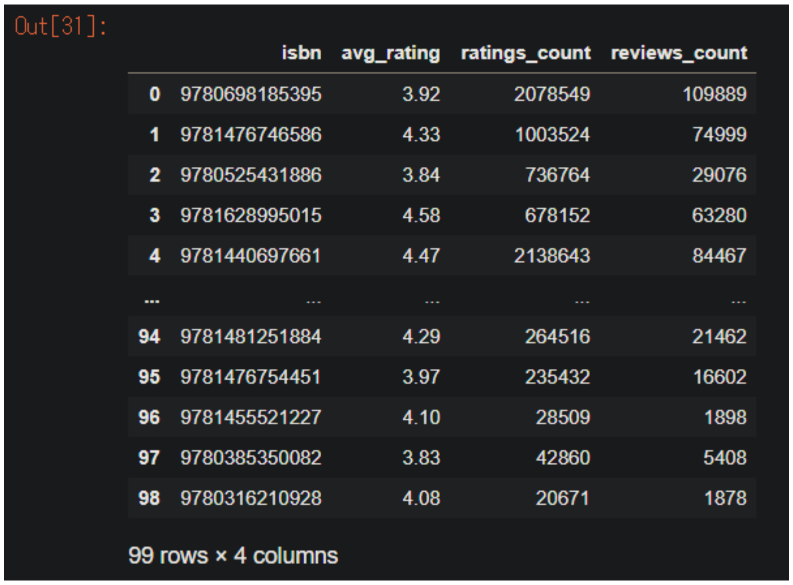
	author	publisher	date	weeks_on_list	title
0	Paula Hawkins	Riverhead	2/19/2017	102	The Girl On The Train
1	Anthony Doerr	Scribner	5/7/2017	81	All The Light We Cannot See
2	E L James	Vintage	3/5/2017	66	Fifty Shades Darker
3	Kristin Hannah	St. Martin's	10/29/2017	63	The Nightingale
4	Kathryn Stockett	Penguin Group	4/8/2012	58	The Help
...	...	...	...	...	...
86	Michael Connelly	Little, Brown	3/27/2016	7	The Drop
85	Michael Connelly	Little, Brown	1/15/2017	7	The Wrong Side Of Goodbye
84	Michael Connelly	Little, Brown	10/15/2017	7	The Late Show
83	David Lagercrantz	Knopf	11/12/2017	7	The Girl Who Takes An Eye For An Eye
88	Stephen King	Scribner	1/11/2015	7	Revival

99 rows × 5 columns

While I tried to automate the data collection process as much as possible, it is important to note that the ISBNs extracted from the Google Books and ISBNdb APIs were not foolproof. Two books —"The Liar" by Nora Roberts and "The Target" by David Baldacci—were assigned incorrect ISBNs that had to be manually corrected.

The resulting dataframe containing data from the ISBNdb API call was appended to googlebooks_df to complete it with titles, authors, and ISBNs for all 99 books. I joined this googlebooks_df dataframe with nyt_title_new on both “title” and “author” columns to create “nyt_merge_df,” which was used with the Goodreads API as previously described.

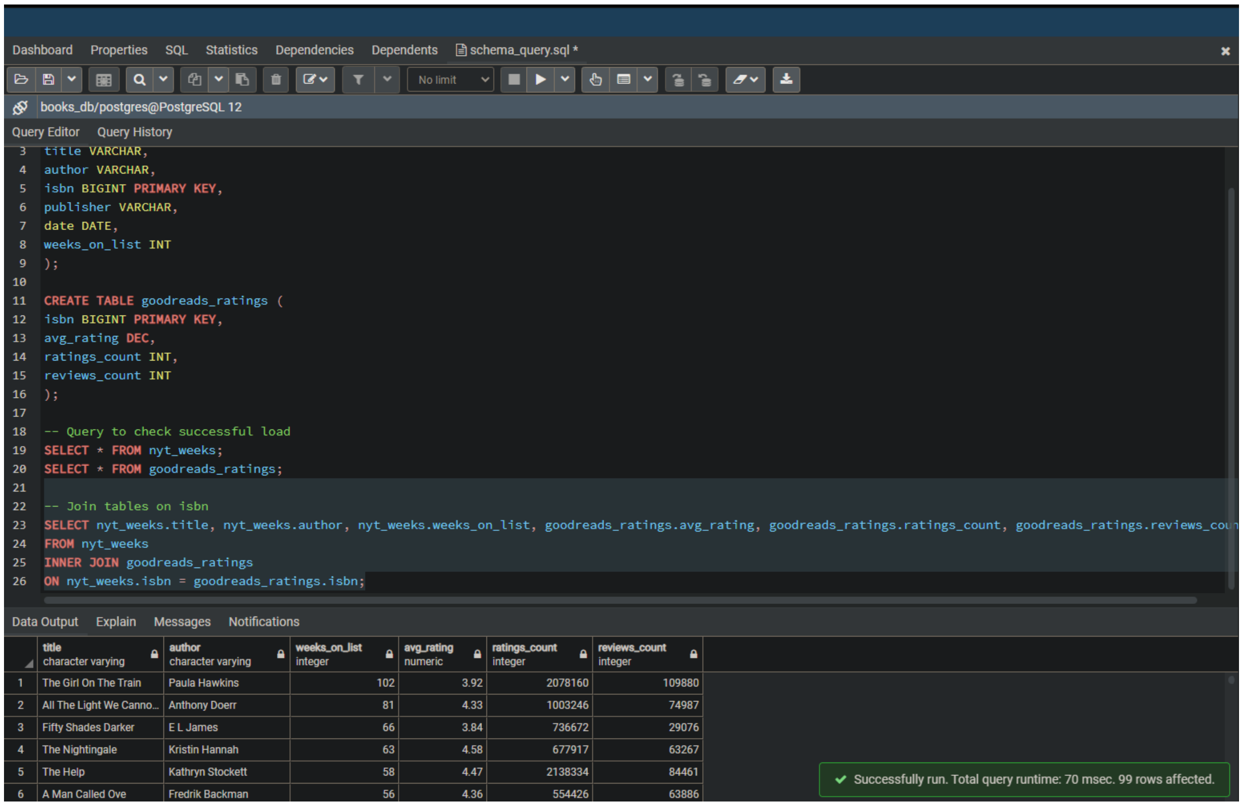
Finally, the “goodreads_df” dataframe was created with the collected Goodreads information.



```
Out[31]:
              isbn  avg_rating  ratings_count  reviews_count
 0   9780698185395        3.92        2078549         109889
 1   9781476746586        4.33        1003524          74999
 2   9780525431886        3.84         736764          29076
 3   9781628995015        4.58         678152          63280
 4   9781440697661        4.47        2138643          84467
...            ...         ...            ...            ...
94   9781481251884        4.29         264516          21462
95   9781476754451        3.97         235432          16602
96   9781455521227        4.10          28509           1898
97   9780385350082        3.83          42860           5408
98   9780316210928        4.08          20671           1878

99 rows × 4 columns
```

Load

To load the dataframes into PostgreSQL, I began by creating a database called “books_db” and the tables “nyt_weeks” and “goodreads_ratings,” setting the “isbn” column as the primary key for both tables. Then, I used to_sql to load the “nyt_merge_df” and “goodreads_df” dataframes into their respective tables after connecting to the local database with Pandas. I queried both tables with pd.read_sql_query in Jupyter Notebook as well as with the SELECT statement directly in PgAdmin to confirm that the data was successfully added. Finally, I joined both tables on the “isbn” column to easily view the final table that could be used to look at correlations between number of weeks on the New York Times best seller list and average rating, number of ratings/reviews, etc.



```sql
title VARCHAR,
author VARCHAR,
isbn BIGINT PRIMARY KEY,
publisher VARCHAR,
date DATE,
weeks_on_list INT
);

CREATE TABLE goodreads_ratings (
isbn BIGINT PRIMARY KEY,
avg_rating DEC,
ratings_count INT,
reviews_count INT
);

-- Query to check successful load
SELECT * FROM nyt_weeks;
SELECT * FROM goodreads_ratings;

-- Join tables on isbn
SELECT nyt_weeks.title, nyt_weeks.author, nyt_weeks.weeks_on_list, goodreads_ratings.avg_rating, goodreads_ratings.ratings_count, goodreads_ratings.reviews_cou
FROM nyt_weeks
INNER JOIN goodreads_ratings
ON nyt_weeks.isbn = goodreads_ratings.isbn;
```

	title character varying	author character varying	weeks_on_list integer	avg_rating numeric	ratings_count integer	reviews_count integer
1	The Girl On The Train	Paula Hawkins	102	3.92	2078160	109880
2	All The Light We Canno...	Anthony Doerr	81	4.33	1003246	74987
3	Fifty Shades Darker	E L James	66	3.84	736672	29076
4	The Nightingale	Kristin Hannah	63	4.58	677917	63267
5	The Help	Kathryn Stockett	58	4.47	2138334	84461
6	A Man Called Ove	Fredrik Backman	56	4.36	554426	63886

✔ Successfully run. Total query runtime: 70 msec. 99 rows affected.