

Big Data System Development

Amazon Book Recommendation System

Team:

1820212029 Talantbekova Bermet

1820212032 Sikander Khan

1820212028 Kydyraliev Temirlan

1820212021 Eiman Noor Asif

24/9/2023

1. Introduction

In this project report, we explore the creation of a big data system for analyzing Amazon book reviews and building a recommendation system. We use BeautifulSoup to collect data from Amazon.com and rely on tools like Apache Spark and Hadoop to handle and process this large dataset. For analyzing and visualizing the data, we make use of PySpark MLib, Pandas, Numpy, Seaborn, and Matplotlib. The main goal of our project is to help users discover books they might enjoy based on their reading habits and preferences.

2. Data Collection - Web Crawling

Importing Libraries:

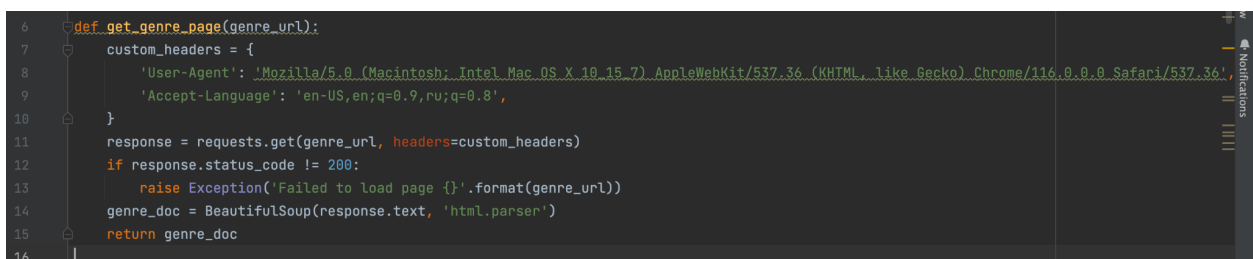
- requests: Used for making HTTP requests to the website.
- pandas: Used for data manipulation and storage.
- os: Used for managing directories and files.
- BeautifulSoup from bs4: Used for parsing HTML content.

A screenshot of a code editor with two tabs: 'main.py' and 'amazon.py'. The 'main.py' tab is active and shows the following code:

```
1 import os
2 import requests
3 from bs4 import BeautifulSoup
4 import pandas as pd
```

Functions:

- **get_genre_page(genre_url):** Retrieves and parses the HTML content of a given URL representing a genre's page on Amazon.

A screenshot of a code editor showing the implementation of the 'get_genre_page' function:

```
6 def get_genre_page(genre_url):
7     custom_headers = {
8         'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36',
9         'Accept-Language': 'en-US,en;q=0.9,ru;q=0.8',
10    }
11    response = requests.get(genre_url, headers=custom_headers)
12    if response.status_code != 200:
13        raise Exception('Failed to load page {}'.format(genre_url))
14    genre_doc = BeautifulSoup(response.text, 'html.parser')
15    return genre_doc
```

- **genre_books_info(div_tags):** Extracts information about individual books (e.g., title, author, URL, etc.) from the HTML elements.

```

49 def genre_books_info(div_tags):
50
51     Book_name_tags = div_tags.find('span', class="a-size-medium a-color-base a-text-normal")
52     Book_url_tags = 'https://www.amazon.com' + div_tags.find('a', class='a-link-normal s-underline-link-text s-link-style a-text-normal')
53     Author_name_tags = div_tags.find('a', class="a-size-base")
54     Book_publisher_tags = div_tags.find('span', class="a-badge-label-inner a-text-ellipsis")
55     Book_img_tags = div_tags.find('img')
56     Book_publish_date_tags = div_tags.find('span', class="a-size-base a-color-secondary a-text-normal")
57     Book_rating_tags = div_tags.find('span', class='a-size-base puis-normal-weight-text')
58     Book_reviews_tags = div_tags.find('span', class='a-size-base s-underline-text')
59     Book_description_tags = div_tags.find('a', class='a-size-base a-link-normal s-underline-link-text s-link-style a-text-normal')
60     Book_price_tags = div_tags.find('span', class='a-offscreen')
61     Book_category_tags = div_tags.find('a', class='a-size-base a-link-normal s-underline-link-text s-link-style a-text-normal')
62     return Book_name_tags, Book_url_tags, Author_name_tags, Book_publisher_tags, Book_img_tags, Book_publish_date_tags, Book_rating_tags, Book_reviews_tags, Book_description_tags, Book_price_tags, Book_category_tags
63

```

- **get_genre_books(genre_doc):** Extracts book information from a parsed genre page and returns it as a Pandas DataFrame.

```

64 def get_genre_books(genre_doc):
65
66     div_selection_class = 'sg-col-20-of-24 s-result-item s-asin sg-col-0-of-12 sg-col-16-of-20 sg-col s-widget-spacing-small sg-col-12-of-16'
67     div_tags = genre_doc.find_all('div', class=div_selection_class)
68
69     genre_books_dict = {
70         'Title': [],
71         'previewLink': [],
72         'authors': [],
73         'publisher': [],
74         'Book_img': [],
75         'Book_publish_date': [],
76         'Book_rating': [],
77         'Book_reviews': [],
78         'description': [],
79         'price': [],
80         'categories': []
81     }
82

```

- **scrape_genre(genre_url, path):** Scrapes book information for a specific genre, saves it as a CSV file, and avoids re-scraping if the file already exists.

```

27
28 def scrape_genre(genre_url, path):
29     if os.path.exists(path):
30         print('The file {} already exists ... Skipping ...'.format(path))
31         return
32     genre_pages = paginate_genre(genre_url)
33     genre_books = []
34     for genre_doc in genre_pages:
35         genre_books += get_genre_books(genre_doc)
36     genre_df = pd.DataFrame(genre_books)
37     genre_df.to_csv(path, index=None)
38

```

- **get_genre_info(doc):** Extracts genre titles and URLs from the Amazon Books landing page.

```

144 def get_genre_info(doc):
145     # Div with all genres
146     div_selection_class = "a-section a-spacing-none"
147     genre_div_tags = doc.find_all('div', class_=div_selection_class)
148     # List of genres inside div
149     genre_tags = genre_div_tags[2].find_all('a')
150
151     genre_titles = []
152     for i in range(0, len(genre_tags)):
153         genre_titles.append(genre_tags[i].text)
154     genre_urls = []
155     base_url = 'https://www.amazon.com'
156     for i in range(0, len(genre_tags)):
157         genre_urls.append(base_url + genre_tags[i]['href'])
158     return genre_titles, genre_urls
159

```

- **scrape_genres():** Initiates the scraping process for genre pages and returns a DataFrame containing genre titles and URLs.

```

160 def scrape_genres():
161     genre_url = 'https://www.amazon.com/Books/s?sr=17143709011&rh=n%3A283155'
162     custom_headers = {
163         'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36',
164         'Accept-Language': 'en-US,en;q=0.9,ru;q=0.8',
165     }
166     response = requests.get(genre_url, headers=custom_headers)
167     if response.status_code != 200:
168         raise Exception('Failed to load page {}'.format(genre_url))
169     doc = BeautifulSoup(response.text, 'html.parser')
170     genre_dict = {
171         'genre_titles': get_genre_info(doc)[0],
172         'genre_urls': get_genre_info(doc)[1]
173     }
174     return pd.DataFrame(genre_dict)

```

- **paginate_genre(genre_url):** Handles multiple pages of book listings within each genre. It finds the "Next" button on each page and continues scraping until there are no more pages left.

3. Data Preprocessing

In this section, we will elaborate on the data preprocessing steps carried out using Pandas. The raw data was initially gathered as a CSV file and subsequently imported into Pandas in the standard way. The following steps outline how the data was prepared for analysis:

Data Loading:

The raw data was loaded into Pandas using the standard data loading methods.

```
book_data = pd.read_csv("D:\\Uni\\Sem 5\\BigData\\Project\\Big-Data-Project\\Preprocessing\\input\\books_data.csv")
book_ratings = pd.read_csv("D:\\Uni\\Sem 5\\BigData\\Project\\Big-Data-Project\\Preprocessing\\input\\Books_Ratings.csv")
```

Data Inspection:

Upon loading the data, an initial inspection was conducted. Some attributes that were deemed unnecessary for the analysis, such as descriptions and images, were identified and subsequently excluded from the dataset.

```
: book_data = book_data[['Title', 'authors', 'publisher', 'publishedDate', 'categories', 'ratingsCount']]
: book_ratings = book_ratings[['User_id', 'Title', 'review/score']]
: book_ratings.head
```

Handling Missing Values:

Missing values in the dataset were addressed by removing records containing missing values. This approach was chosen as missing values were considered minor in quantity and not critical to the analysis.

Data Transformation:

Data transformation was performed to calculate book ratings. This involved utilizing data from two tables and using the groupby() function to aggregate and compute ratings based on specific criteria. The resulting ratings were added as a new feature in the dataset, enhancing its analytical capabilities.

```
: average_scores = book_ratings.groupby('Title')['review/score'].mean().reset_index()
: merged_data = pd.merge(book_data, average_scores, on='Title', how='left')
: merged_data.head(5)
```

Data Quality Issues:

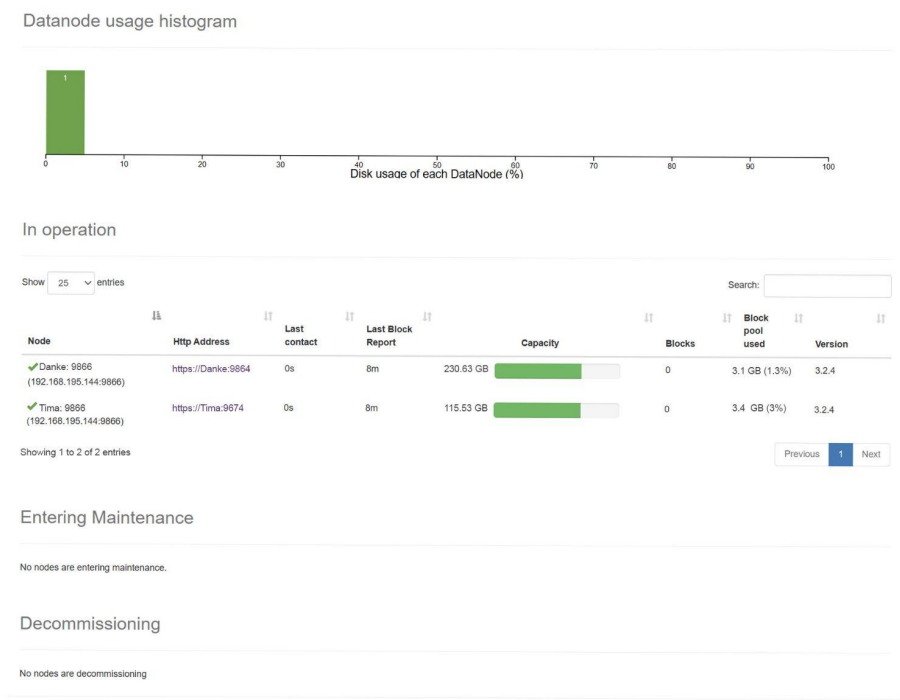
During the preprocessing phase, no major data quality issues were encountered. The data loading process was smooth, and missing values were handled appropriately through removal, which did not compromise the integrity of the dataset. The exclusion of unnecessary attributes also contributed to improved data quality and streamlined analysis.

4. Data Storage

In this section, we will explore how we stored the collected data using an HDFS (Hadoop Distributed File System) cluster. Hadoop 3.2.4 was chosen as the Hadoop distribution, and a multi-node HDFS setup was employed for robust data storage and processing capabilities.

HDFS Integration:

The HDFS cluster was set up using Hadoop 3.2.4.



Data Ingestion:

To transfer the scraped book data into the HDFS cluster, a manual command-line approach was employed. This method allowed for greater control and flexibility during the data transfer process.

File Formats:

The data was stored in CSV format within the HDFS cluster. This format was chosen for its simplicity and ease of use, making it suitable for subsequent data processing and analysis tasks.

Data Replication:

To ensure data redundancy and fault tolerance, the HDFS cluster was configured with a replication factor of 2. This means that each data block is replicated across two different nodes in the cluster, enhancing data durability.

Data Accessibility:

Access to the HDFS cluster is restricted to individuals with permissions at the NameNode and DataNode levels. This controlled access helps maintain data security and integrity.

Scalability and Redundancy:

The multi-node HDFS setup ensures scalability, allowing the cluster to handle potentially large volumes of data. Additionally, the replication factor of 2 enhances data redundancy, reducing the risk of data loss.

5. Data Processing and Analysis with Spark

In this section, we will discuss the rationale behind choosing Apache Spark for data processing and analysis, the setup of the Spark environment, the Spark operations and transformations applied to the data, and any challenges encountered during the process.

5.1 Choice of Spark for Data Processing and Analysis

Spark was selected as the preferred framework for data processing and analysis due to several key factors:

Complexity Mitigation:

The project aimed to avoid the intricacies associated with Hadoop's Java MapReduce functions. Spark's higher-level APIs offer a more intuitive and simplified approach to data processing.

Performance:

Spark's in-memory processing capabilities make it significantly faster than traditional MapReduce, enabling quicker data analysis and model building.

5.2 Setting up the Spark Environment

Configuration and Setup:

The Spark environment was set up by directly obtaining the Apache Spark distribution from the official website. The following aspects were involved in configuring the Spark environment:

Environment Variables:

Environment variables were manually configured to ensure that Spark could seamlessly interact with the Hadoop Distributed File System (HDFS) cluster, facilitating data access and manipulation.

5.3 Spark Operations and Transformations

Data Exploration and Preprocessing:

As a preliminary step, the data underwent exploration and preprocessing. This involved various tasks such as handling missing values, converting data types, and other data cleaning procedures.

Creating User-Item Matrix:

The core operation involved creating a user-item matrix, a pivotal component in building the recommendation system. The process can be summarized as follows:

```
from pyspark.sql.functions import col

# Select relevant columns for user-item matrix

user_item_matrix = df.select("User_id", "id",
                             "review_score")

# Pivot the table to create the user-item matrix

user_item_matrix =
user_item_matrix.groupBy("User_id").pivot("id").agg({"review_score": "first"}).fillna(0)
```

6. Recommendation System

In this section, we delve into the development and implementation of the recommendation system, a pivotal component of our project.

Collaborative Filtering Algorithm

Collaborative filtering, a popular recommendation technique, was employed to create personalized book recommendations for our users. Collaborative filtering operates on the principle of leveraging user behavior data to identify patterns and make recommendations based on users' similarities to others within the dataset. We specifically implemented the collaborative filtering technique for its effectiveness in providing accurate and relevant book recommendations.

Collaborative filtering in PySpark involves using techniques to make recommendations to users based on the behavior and preferences of other users. This can be achieved using the Alternating Least Squares (ALS) algorithm.

Implementation Using Spark

Choice of Spark: We opted for Apache Spark to facilitate data processing and recommendation generation due to its speed and ease of use compared to traditional Hadoop MapReduce functions. The setup involved downloading Spark directly from the Apache website, configuring environment variables, and integrating it seamlessly with our Hadoop Distributed File System (HDFS) cluster.

Data Exploration and Preprocessing:

The initial step was data exploration and preprocessing. This included handling missing values, converting data types, and other necessary data cleaning tasks.

Creating User-Item Matrix:

To build the recommendation system, we transformed the data into a user-item matrix. Each row represented a user, each column represented a book, and the values in the matrix corresponded to user ratings for the respective books. We used Spark's functionalities to pivot the table, creating the user-item matrix.

```
from pyspark.sql.functions import col

# Select relevant columns for user-item matrix

user_item_matrix = df.select("User_id", "id",
                             "review_score")

# Pivot the table to create the user-item matrix

user_item_matrix =
user_item_matrix.groupBy("User_id").pivot("id").agg({"review_score": "first"}).fillna(0)
```

Collaborative Filtering Model: With the user-item matrix in place, we applied collaborative filtering algorithms to generate recommendations. These algorithms identified users with similar preferences and suggested books that like-minded users had enjoyed.

```
# Build the ALS model
als = ALS(
    maxIter=5,
    regParam=0.01,
    userCol="User_id",
    itemCol="id_numeric",
    ratingCol="review_score",
    coldStartStrategy="drop"
)

# Split the data into training and test sets
(training, test) = df_cleaned.randomSplit([0.8, 0.2])

# Fit the ALS model on the training data
model = als.fit(training)

# Generate predictions on the test data
predictions = model.transform(test)
```

Evaluation Metrics

Performance Measurement: To assess the quality of our recommendation system, we employed the Root Mean Squared Error (RMSE) as our primary evaluation metric. RMSE provides a measure of the model's accuracy by quantifying the difference between predicted ratings and actual ratings. Lower RMSE values indicate more accurate recommendations.

```
# Evaluate the model
evaluator = RegressionEvaluator(
    metricName="rmse",
    labelCol="review_score",
    predictionCol="prediction"
)
rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE) = {rmse}")
```

✓ 23.2s

Root Mean Squared Error (RMSE) = 2.9929781215420115

Challenges Faced

Learning Curve: The journey of implementing the recommendation system was not without its challenges. Learning the fundamentals of machine learning and grappling with the intricacies of Spark was a significant hurdle. However, the team's dedication and persistence ultimately led to a successful recommendation system.

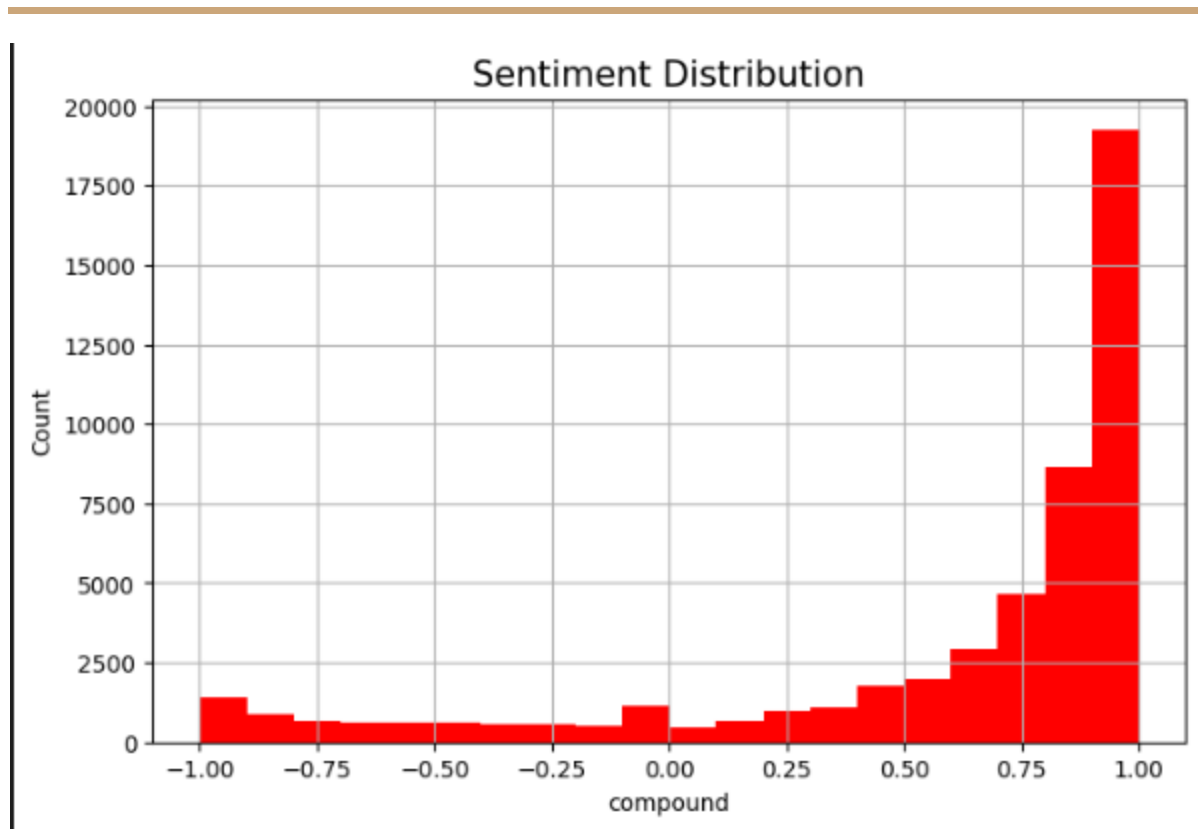
7. Results

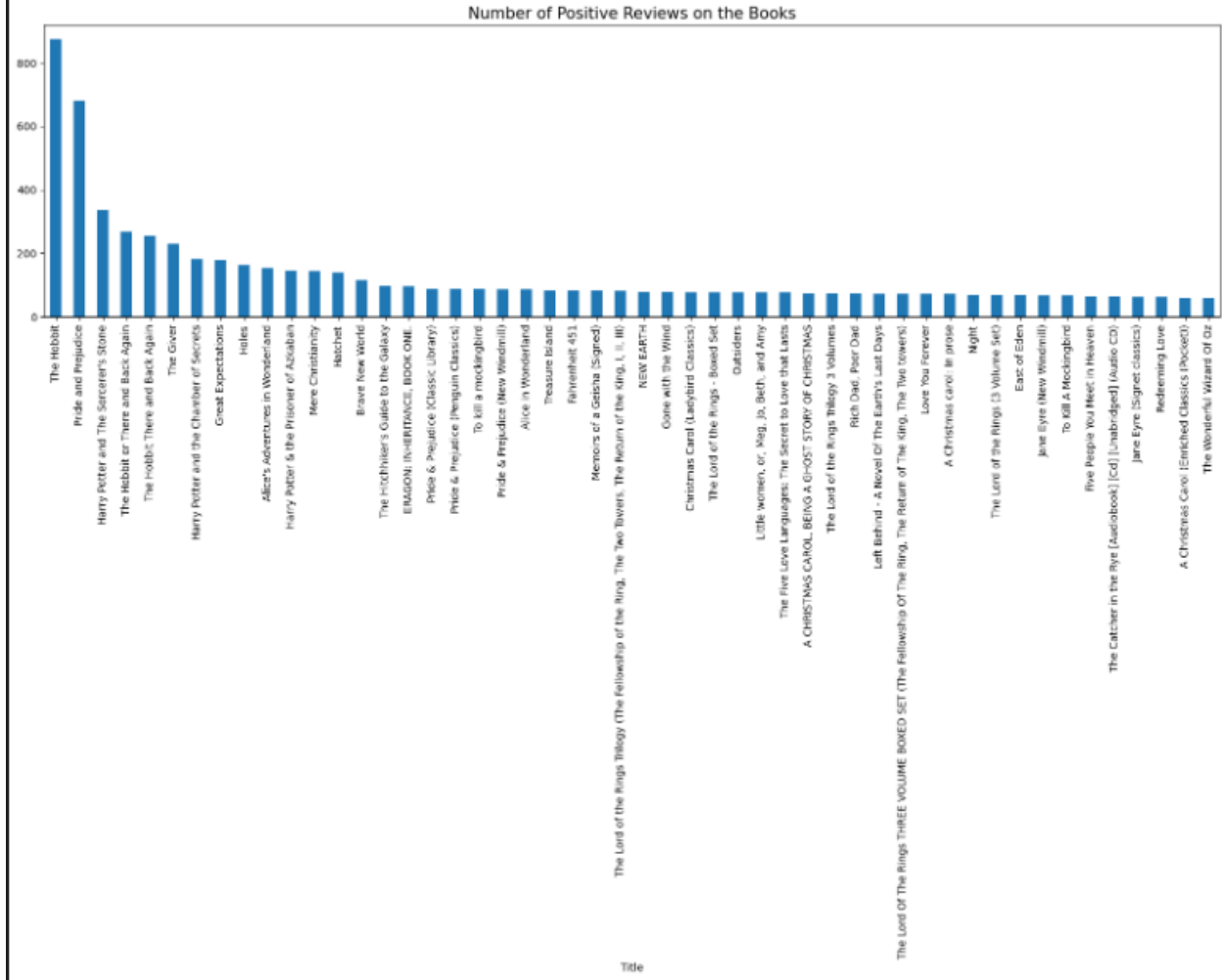
These are our results.

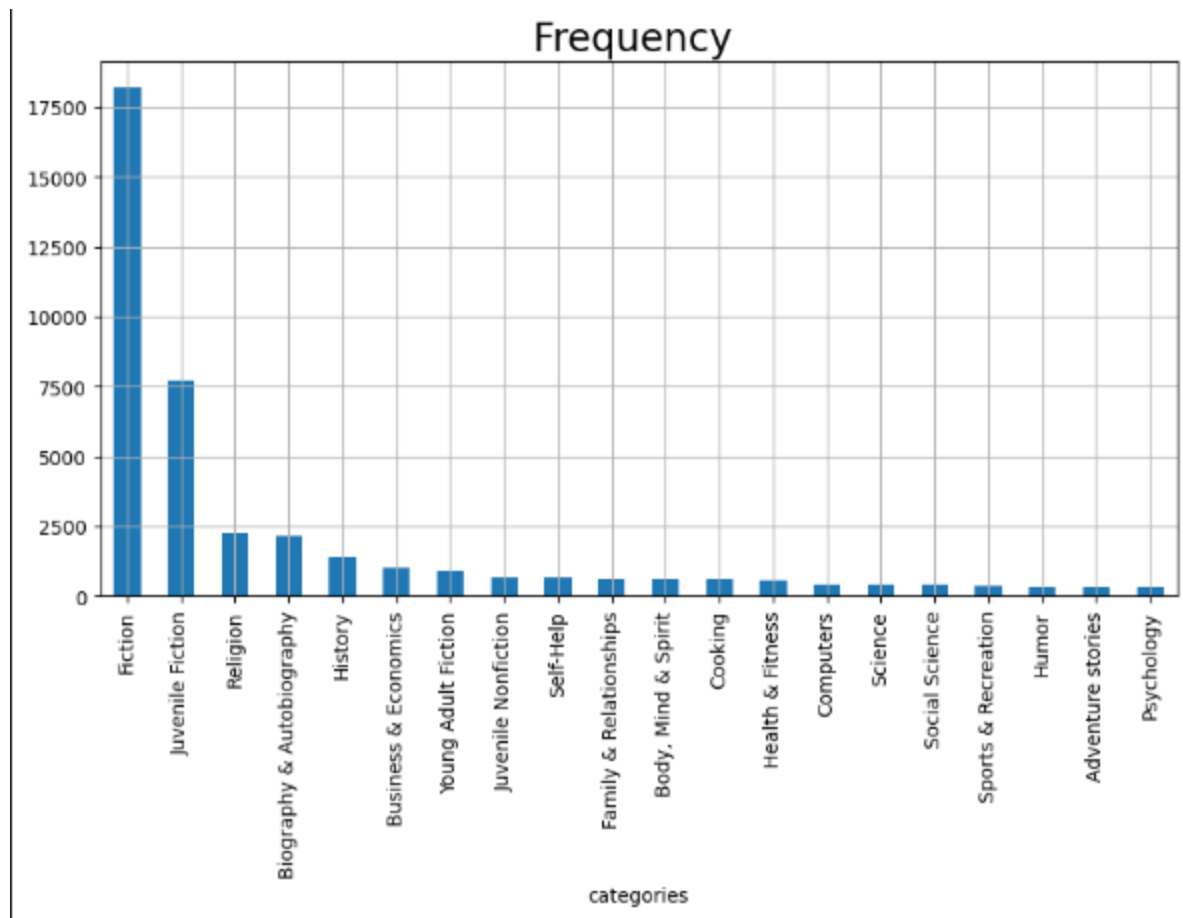
```

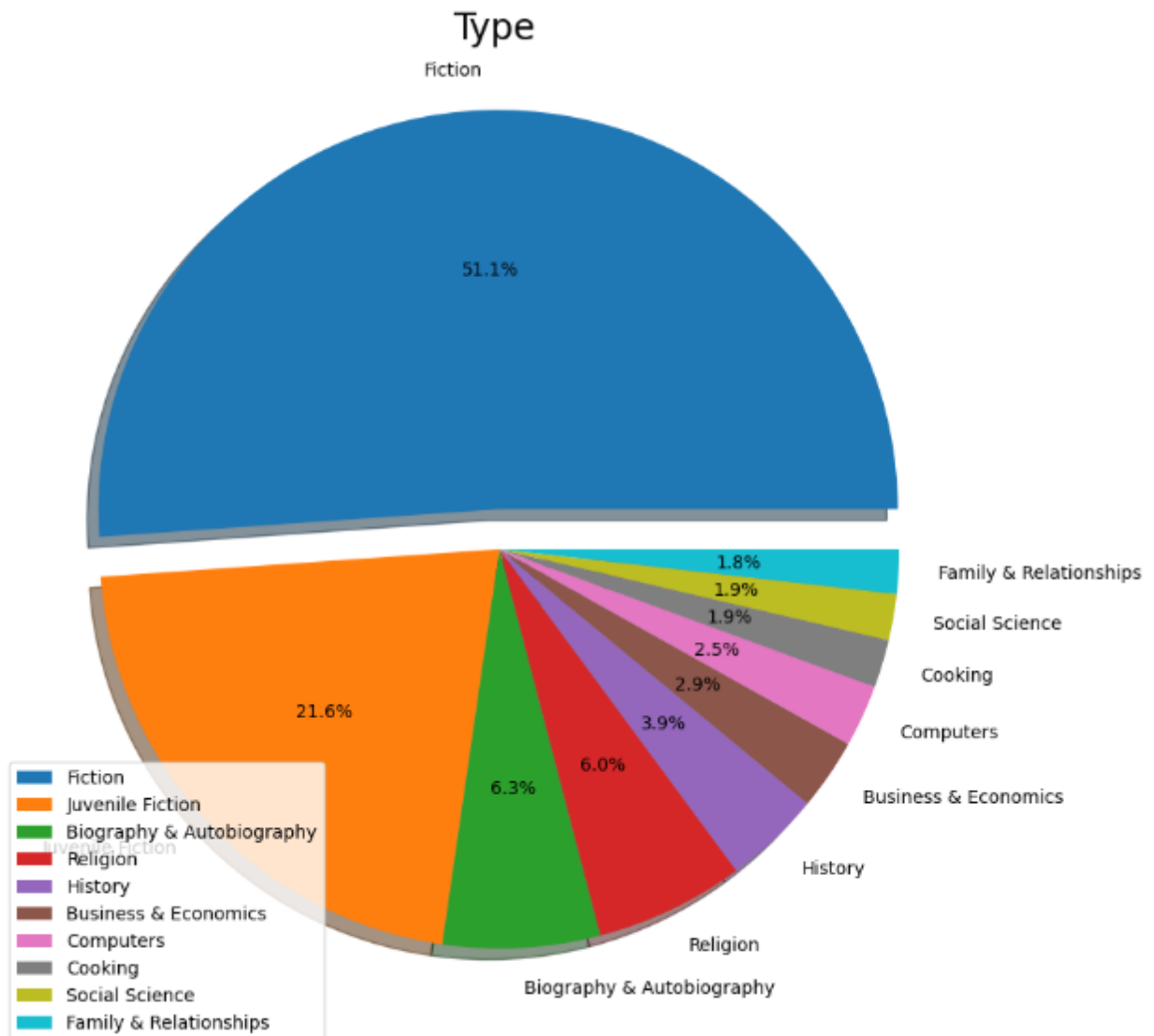
+-----+-----+-----+
|      id|      User_id|review_score|
+-----+-----+-----+
|1882931173| AVCGYZL8FQQT|         4|
|0595344550| ACO23CG8K8T7|         5|
|0595344550| A30S2QHEH495|         1|
|0595344550| A30ZDTEEF8GS|         1|
|0802841899| ANX3DDV12ZRR|         4|
|0802841899| A2H2LORTA5EZY|         4|
|B0007FIF28| A2GERYVE64DIP|         3|
|B000JINSBG| A15A5KPP3AL7|         5|
|0918973031| A1X1CW1GKKC50|         5|
|0918973031| A309DQ3THGNX|         5|
|0974289108| A1KZ0RDJZQSY4|         3|
|0974289108| A3AJA5ADM3Q8L|         5|
|B000NKGMYK| A258YNWJW2264|         3|
|B000NKGMYK| A2WY5VMJQ0MM1|         5|
|B000NKGMYK| A7IA8CTTSQ7A4|         3|
|0789480662| A2GA412HQHN8W|         5|
|0789480662| A35Z7FIHBSCHK|         4|
|0789480662| A13HDF4J03LQ8|         4|
|B0000CJHIO| A1UHTWM53B5KM|         4|
|B0000CJHIO| A22PE7W18KPDE|         5|
+-----+-----+-----+
only showing top 20 rows

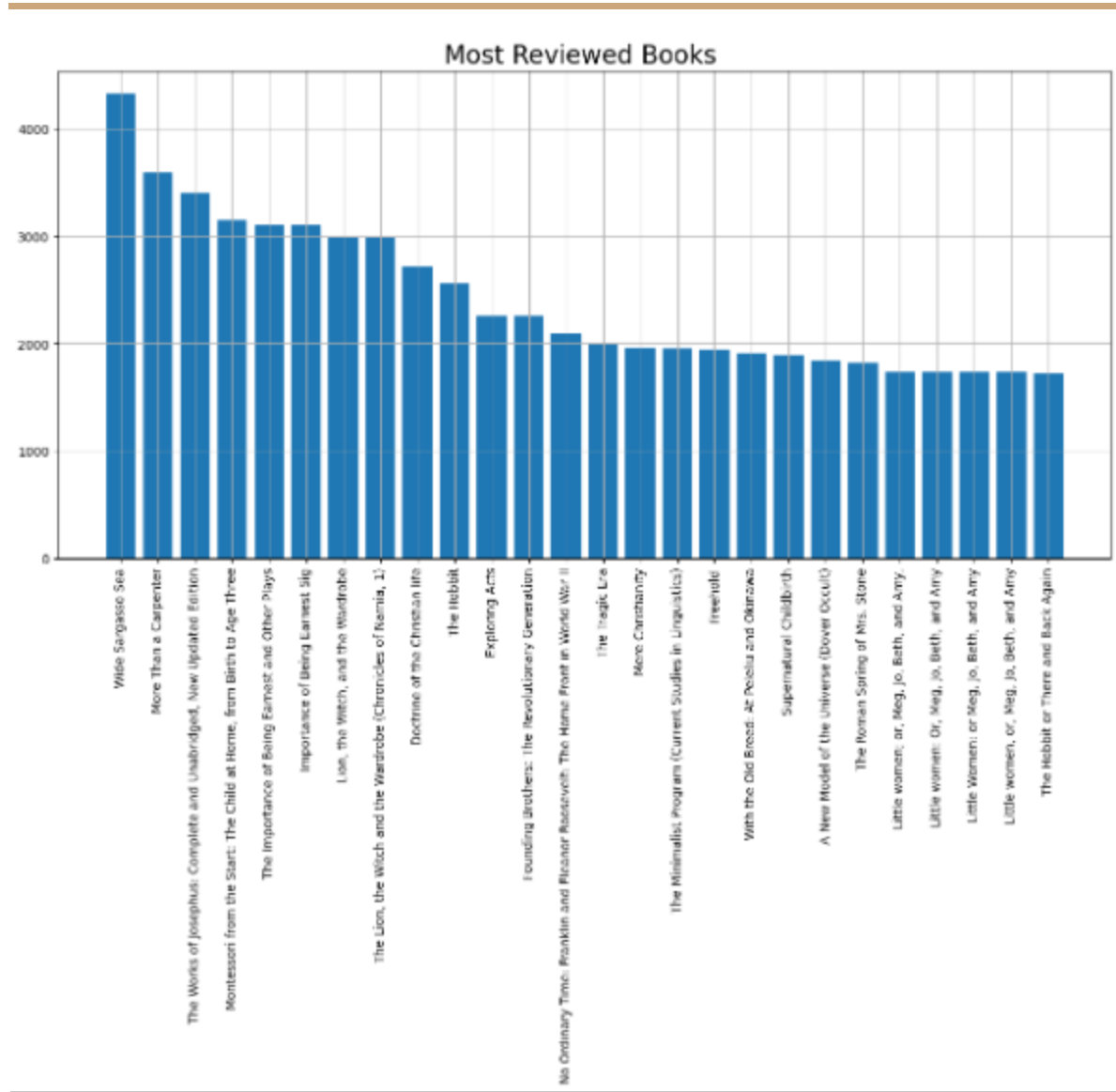
```

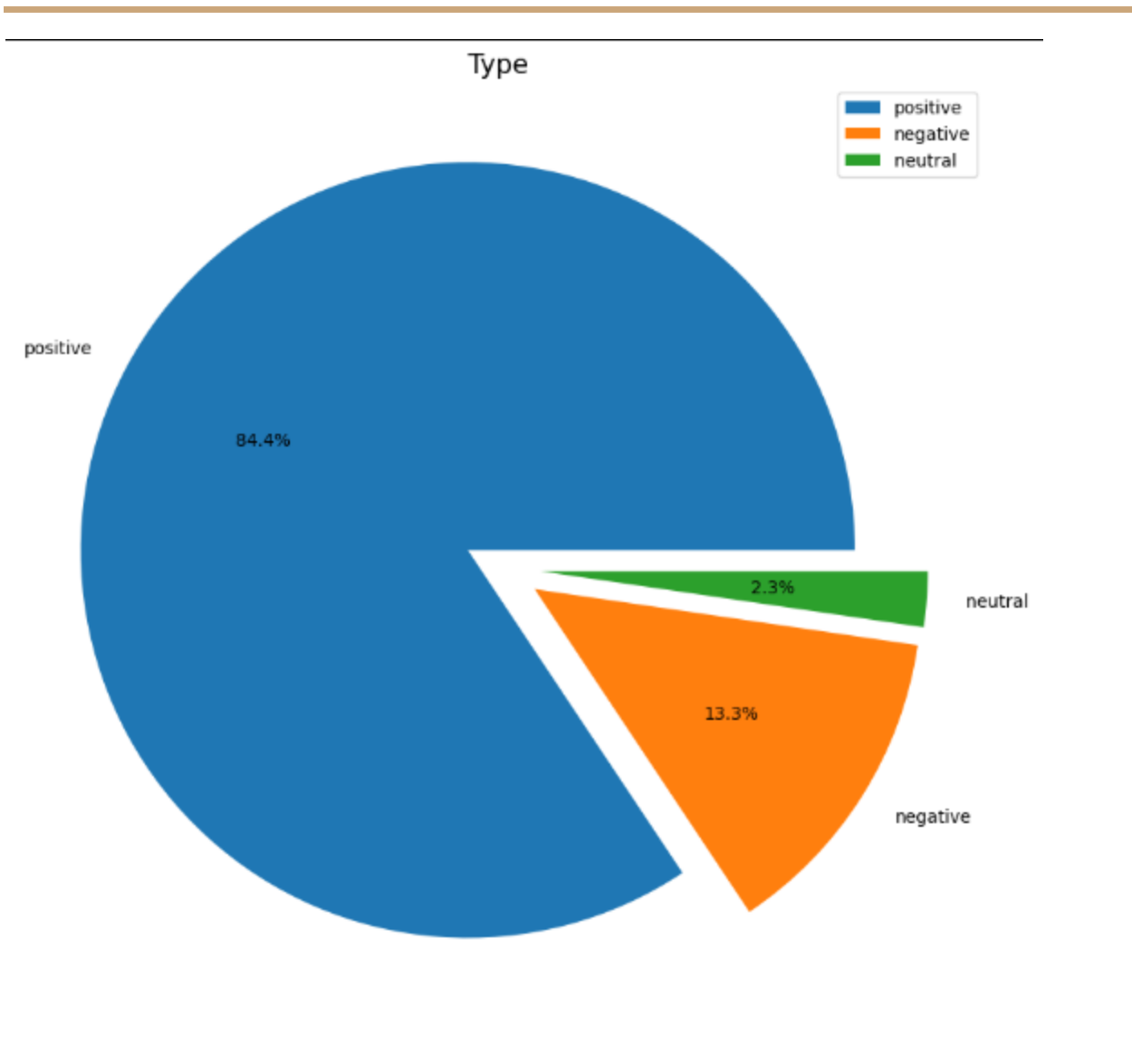


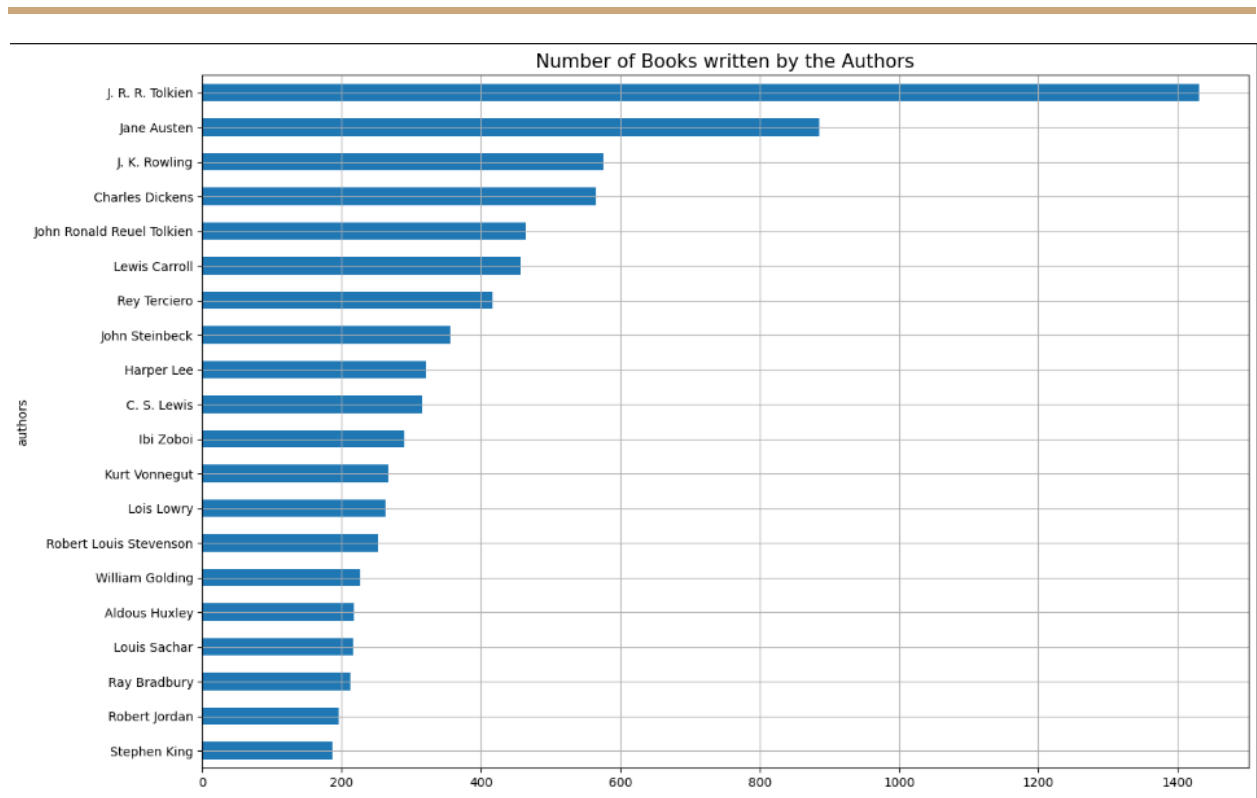


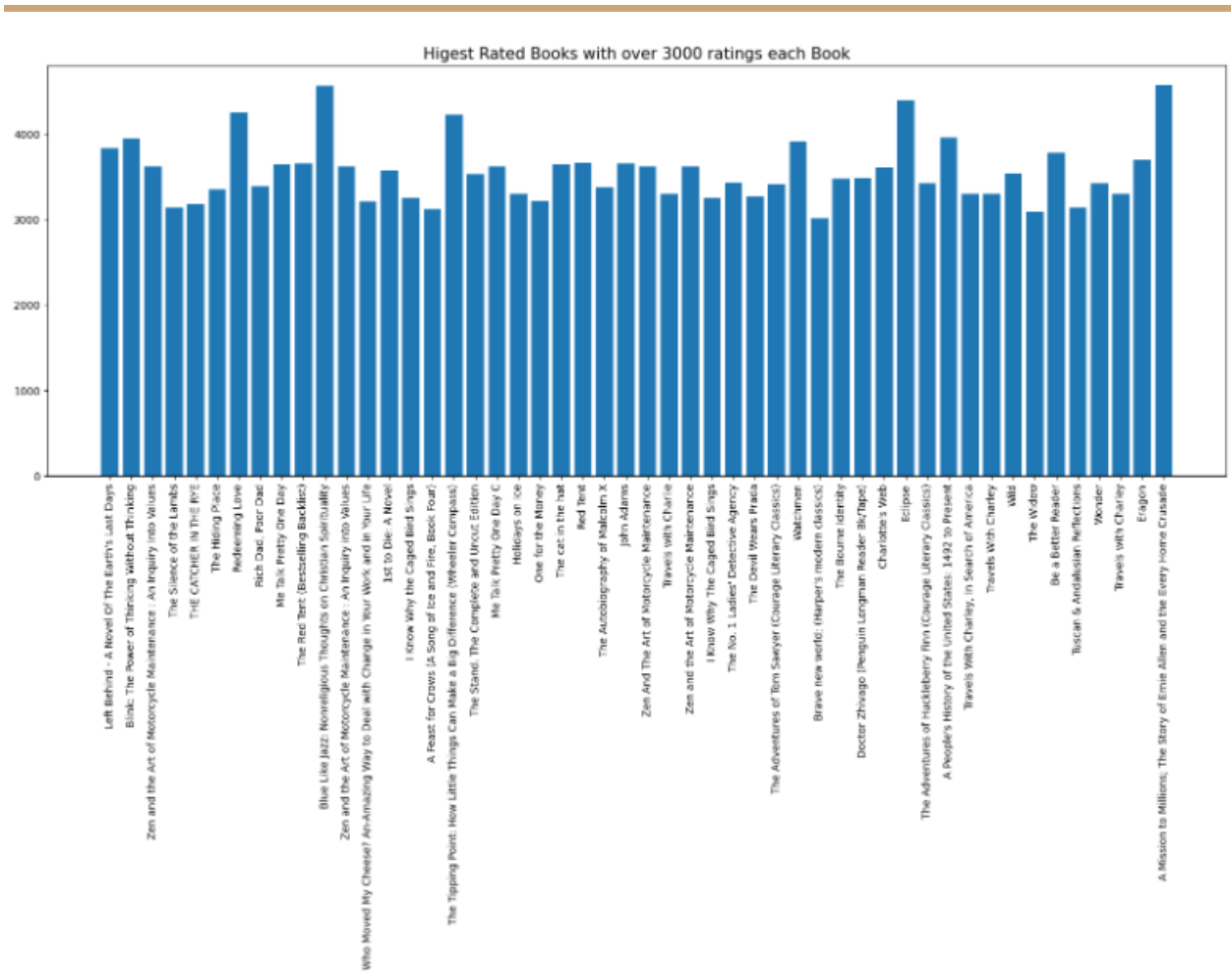


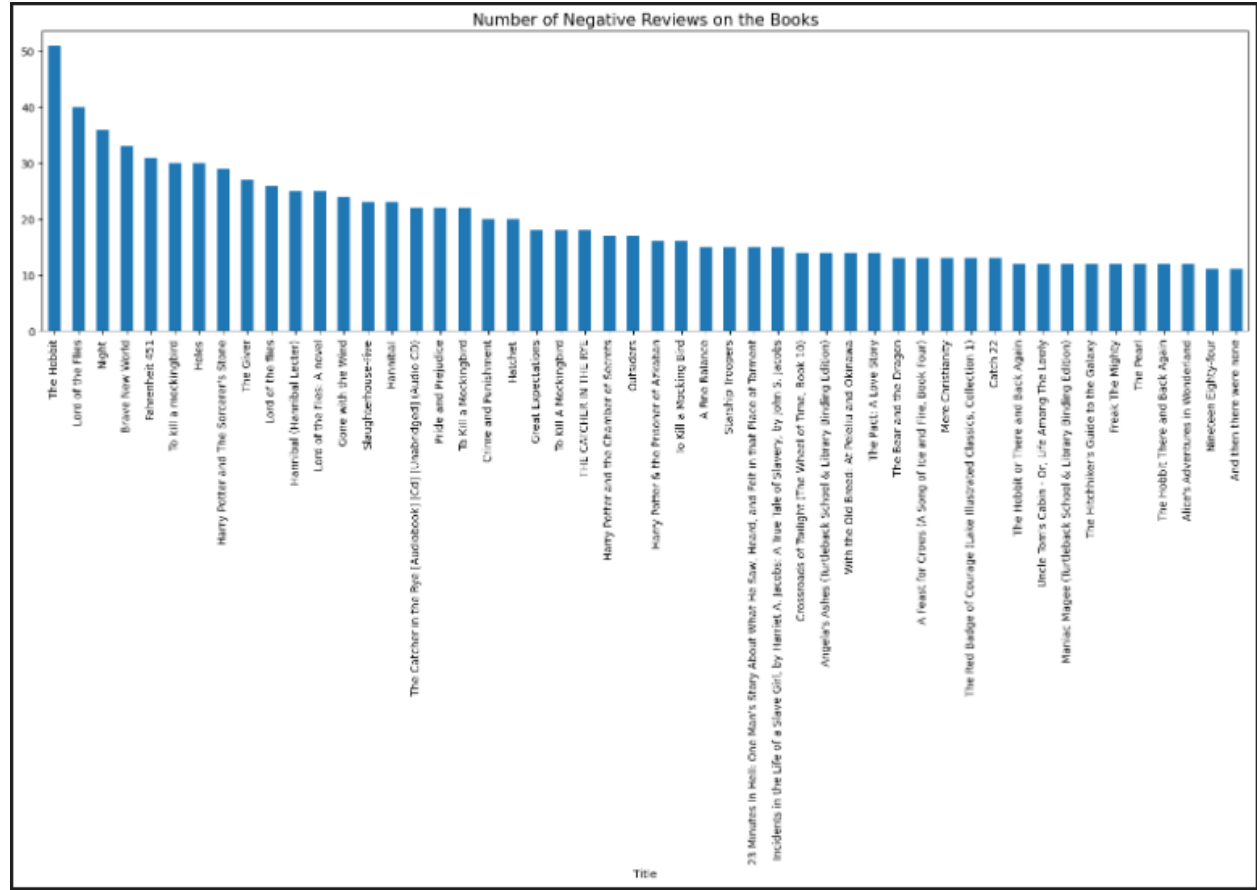


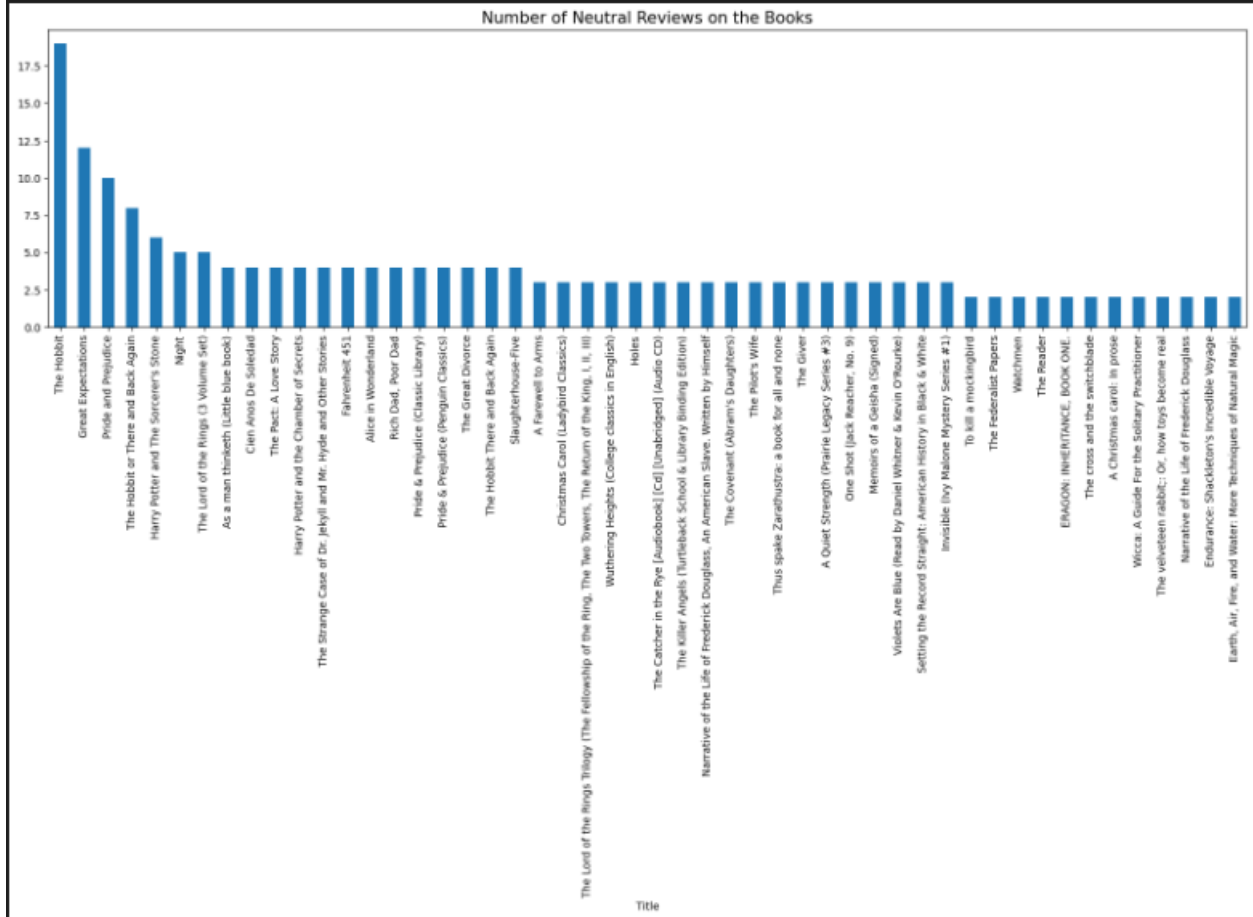


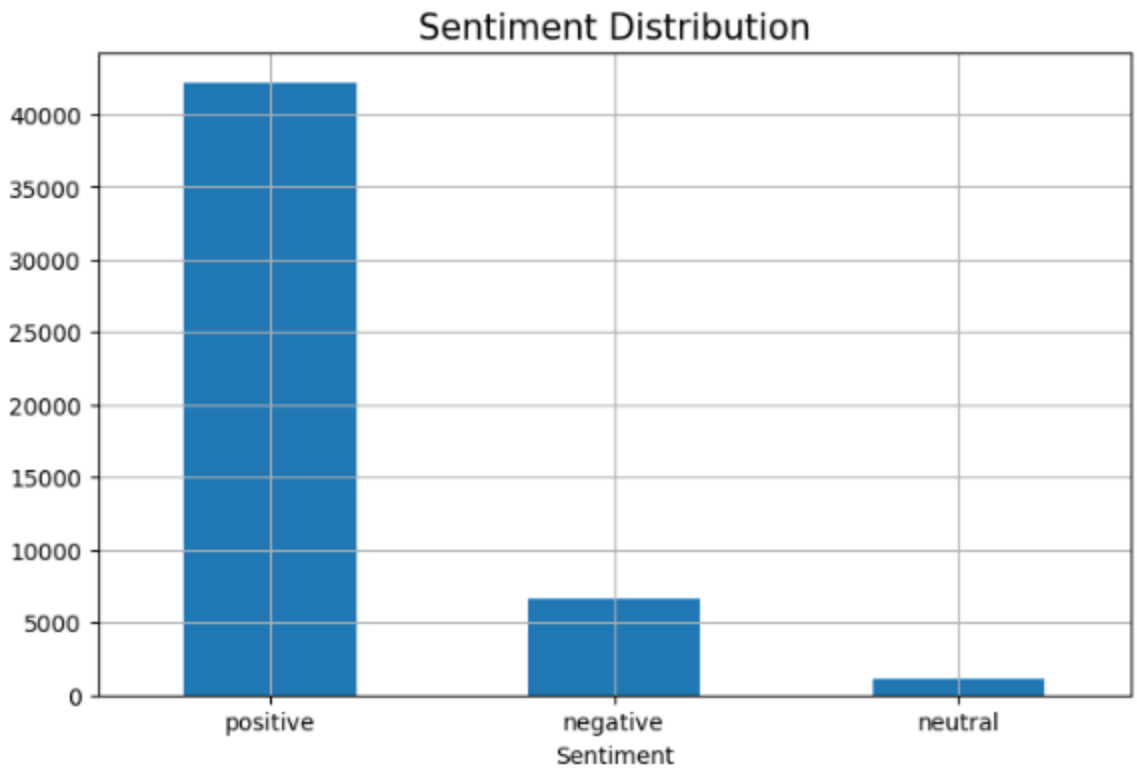












9. Conclusion

We can draw a few conclusions from this analysis.

According to our dataset, most books were fiction.

J.R.R Tolkein sold the most books, followed by Jane Austen and J.K Rowling.

Most books were rated well on Amazon.

The Hobbit was the most well reviewed book, and the one with the worst reviews, probably because it had the most reviews.

Unsurprisingly, book was the most popular word.