Final Project:
Book recommendation systems

COMPUTATIONAL TOOLS FOR DATA SCIENCE

Authors:
Manfrin Pool Antialon Barrera (s222437)
Aniol Bisquert Parés (s231813)
Barbara Anna Koska (s233500)
Eduard Puga Creus (s231919)
Javier Alonso Fernández (232509)

Group 22

November 28, 2023

Contents

1 Motivation and Introduction **1**

2 Dataset **1**
2.1 Data preprocessing 1
2.2 Description of the dataset 1
2.3 Data visualization 2

3 Association Rules **3**
3.1 Methodology 3
3.1.1 Reccommendation and Evaluation 4
3.2 Pros and Cons of Association Rules Recommendation Systems 4

4 Content based book recommendation system **5**
4.1 Methodology 5
4.1.1 Recommendation and Evaluation 6
4.2 Pros and Cons of Content Based Book Recommendation System . . . 7

5 Collaborative filtering System **7**
5.1 Methodology 7
5.1.1 Recommendation and Evaluation 7
5.2 Pros and Cons of Collaborative filtering Systems 8

6 Discussion **8**

List of Figures

1 Histogram and Box Plot for the Book-Rating attribute 2
2 Box Plot for 'publishedDate' and 'pageCount' 2
3 Distribution of Ratings for Top 10 Authors and Categories with Highest Mean Ratings 3
4 Association rules: results 4
5 Dendogram from Hierarchical Clustering 5
6 Finding optimal number of clusters 6
7 Fantasy Topics 6
8 Detective Topics 6
9 War Topics 6

1 Motivation and Introduction

The aim of this project is to discover links between readers and stories, crafting a personalized route through the expansive realm of human creativity. This academic endeavor demands a dedicated commitment to meaningfully contribute to the book recommendation system, ensuring users can effortlessly access books aligned with their preferences, fostering a joyful reading experience.

The research in question delves into the exploration of recommendation systems for a diverse collection of books. The dataset, a subset derived from a comprehensive compilation available on Kaggle [1], has been judiciously selected to align with the scope of our resources.

The main goal is to compare techniques and algorithms, emphasizing efficiency and robustness in the recommendation system. The analysis focuses on three approaches: rules filtering, content-based recommendation systems, and collaborative filtering. It aims to discuss the merits and drawbacks of each technique, providing insights into their strengths and limitations. This effort contributes to advancing understanding of recommendation systems for books, guiding the development of future systems in this domain.

Analyzing the recommendation system algorithms encompasses a scrutiny of association rule techniques, specifically A-priori and FP-Growth. In the context of clustering, examination is conducted on hierarchical clustering and KMeans clustering methods. Additionally, the SVD method is explored as an supplementary approach.

2 Dataset

2.1 Data preprocessing

To enhance the depth and quality of the dataset, a targeted retrieval of additional bibliographic details was performed for a randomly selected subset of books. This extra information was extracted through the Google Books API [2], enabling the supplementation of records with a wealth of metadata. The added details include titles, subtitles, authors, publication dates, descriptions, page counts, categories, and language specifications.

Additionally, in the rating dataset, all 'Book-Rating' values that are 0 were excluded. This decision was made to improve the accuracy of the analysis, as a rating of 0 may not necessarily indicate a negative evaluation but could be prone to misunderstanding or misinterpretation.

All the codes and datasets are available in Github [3].

2.2 Description of the dataset

The datasets comprises two main components: BOOKFINAL2 and RATINGSFINAL.

RATINGSFINAL incorporates a total of 66082 ratings, organized into three key columns—'User-ID,' 'ISBN,' and 'Book-Rating.' The 'User-ID' signifies user identifiers, 'ISBN' serves as book identification codes, and 'Book-Rating' records the ratings assigned to the books.

On the other hand, **BOOKFINAL2** consists of 6272 records, featuring a diverse array of information about each book. Columns include 'ISBN', 'title', 'subtitle', 'authors', 'publisher', 'publishedDate', 'description', 'pageCount', 'categories' and 'language'. All the books in the dataset are written in English.

2.3 Data visualization

In the **RATINGSFINAL** dataset, the provided statistics reveal a distribution of ratings that is notably skewed towards higher values (see Figure 1), with a mean rating of approximately 7.78. This distribution suggests a bias towards positive ratings, a phenomenon common in recommendation systems and user-generated content platforms where users tend to rate items they enjoyed more positively.

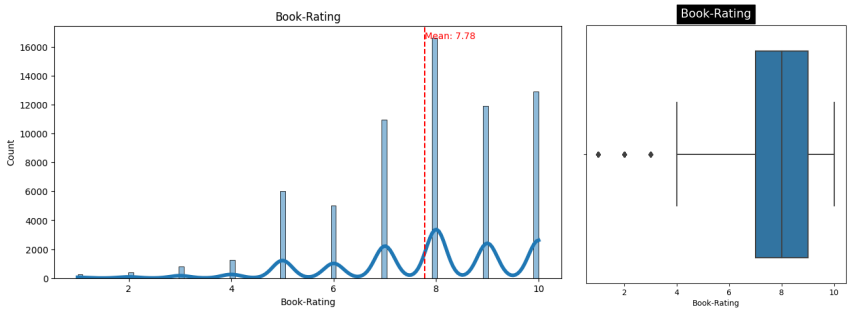


Figure 1: Histogram and Box Plot for the Book-Rating attribute

In the **BOOKFINAL2** dataset, books with 0 pages, indicating potential misinformation, are excluded from the mean calculation. The resulting average page count for the remaining entries is 320 pages(see Figure 2). The dataset predominantly features books published between 1980 and 2004, with "Fiction" standing out as the most prevalent category. Notably, Stephen King emerges as the most prolific author, having published the highest number of books(36), while Penguin stands out as the leading publisher with 168 books published.

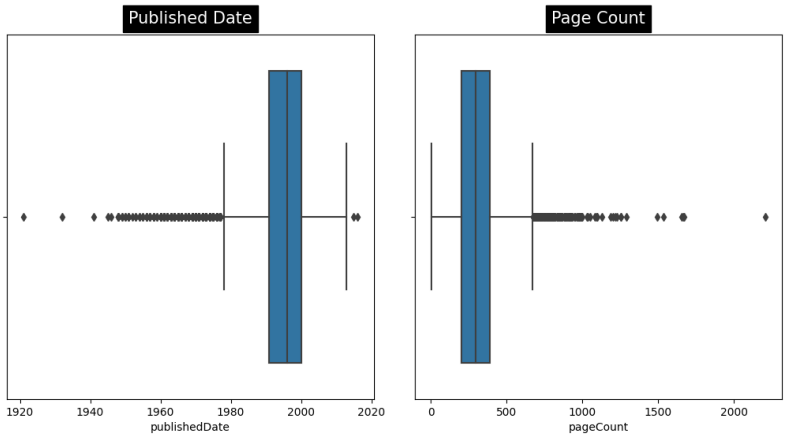


Figure 2: Box Plot for 'publishedDate' and 'pageCount'

In merging datasets on 'ISBN' to explore correlations between book ratings and information, 'The Lovely Bones' stands out as the most rated book with 707 ratings. The mean ratings count for all books is 17. 'The Two Towers' holds the highest average rating at 9.72. For robust analysis, authors with only one rated book are

selectively disregarded, recognizing potential representational limitations. Consequently, 'Helene Hanff' emerges as the most highly rated author with an average rating of 9.46 (see Figure 3). Additionally, the category 'Baggins, Frodo (Fictitious character)' stands out with the highest mean rating at 9.29. This analysis provides valuable insights into the nuanced landscape of book ratings, identifying trends and ensuring a focused exploration of representative data points.

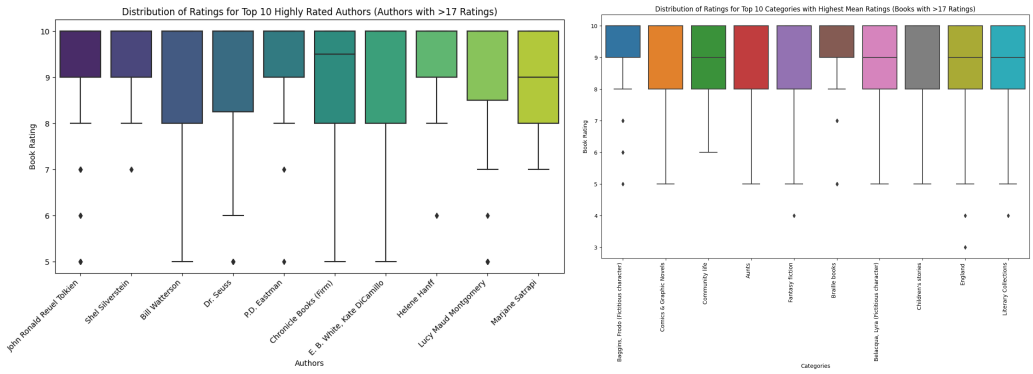


Figure 3: Distribution of Ratings for Top 10 Authors and Categories with Highest Mean Ratings

3 Association Rules

3.1 Methodology

In unveiling inherent patterns among co-occurring items, association rules played a pivotal role. In the context of book recommendation system, metrics as support, confidence and lift where thoroughly investigated. **Lift** indicates the strength of association between items, **confidence** measures the likelihood that the recommended item will be chosen, and **support** measures the frequency of occurrence of the itemset. In the initial phase of exploration, user ratings spanning from 1 to 10 were filtered to indicate only the books that user liked (7+ score). The filtering process resulted in 52354 ratings, that grouped by the user, created 21010 individual transactions.

For investigating association rules, **A-priori** algorithm was used to get frequent items due to its simplicity and efficiency. A-priori follows a two-step routine: extracting frequent itemsets and generating association rules. The efficiency boost comes from the Apriori principle, stating that if an itemset isn't frequent, neither are its subsets. This principle significantly speeds up execution compared to older comprehensive methods. [4] During the process of obtaining more reliable recommendation system, different support threshold was tested. When lowering threshold to 0.0005, A-priori algorithm failed due to size of a data and **FP-Growth** Algorithm was used, that mines frequent itemsets using tree-based approach, reducing the need for generating candidate itemsets, making it more scalable and efficient [5].

All support thresholds and size of frequent items and given association rules can be seen in Table 1, where each step was analysed and respective action to obtain more effective recommendation system was created. All rules datasets had confidence threshold set to 0.5 to investigate the associations between different items.

Min Support	Support * transactions size	Frequent items size	Association rules size	Comment
0.003	63 items	159 rows	9 rows	Too high support produces little rules, which cannot support recommendation system.
0.001	21 items	731 rows	121 rows	Even though there are more much more frequent items, there are still only 121 rules, meaning that not all books are included in the association rules, creating big limitations for the recommendation system.
0.0005	10.5 items	1628 rows	319 rows	Lowering support generates more association rules, however it also produces less reliable results, as there are less pattern repeated in the dataset. Still, majority of books are not included in the rules.

Table 1: Association rules

3.1.1 Reccommendation and Evaluation

To understand how the recommendation could be used based on this rules set, a look can be taken in Figure 4 where, for the person that liked books of ISBN '439136350' is recommended book '439064864'. The support value means that items occured 77 times in the dataset together, which could create a reliable pattern, however confidence value of 0.59 suggests that the likelihood of consequent item being liked given antecedent liking is 59%, whereas lift value of 104.57 suggests that the occurrence of items together is much more likely than if the items were independent, which provides a positive correlation between those items.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(439136350.0)	(439064864.0)	0.006188	0.005664	0.003665	0.592308	104.574661	0.003630	2.438937

Figure 4: Association rules: results

3.2 Pros and Cons of Association Rules Recommendation Systems

While there are rules with high confidence and high lift value, the challenge with association rules analysis is its focus on common items, overlooking unique book attributes and individual user preferences. This narrow perspective leads to the exclusion of many items that are less common (not many users read and liked the book) or lack strong associations, limiting the system's effectiveness for a wide range of users. Additionally, neglecting the specifics of books and users hinders the system's ability to offer personalized recommendations. While the method excels at identifying patterns in frequently occurring items, its restricted approach highlights the necessity for a more comprehensive recommendation system.

4 Content based book recommendation system

In the realm of recommendation systems, the content-based approach, particularly in book recommendations, emphasizes intrinsic item attributes for personalized suggestions [6]. This approach aims to enhance content-based recommendations using advanced clustering techniques, like Hierarchical and K-Means, to categorize books based on textual information. The goal is to provide a nuanced and diverse user experience by capturing the multifaceted nature of literature and accommodating dynamic reading preferences.

4.1 Methodology

Data Cleaning and Feature Extraction The process began with extracting further additional textual information, such as 'Title,' 'Subtitle,' 'Authors', 'Description' and 'Category' from ISBN codes using the Google Books API. 'Category' variable contained 1532 distinct values among all the 6272 books, so it was not possobile to categorize them using it. All these features were combined to create a book corpus. Subsequently, the text data underwent cleaning by removing stop words, numbers, punctuation, and applying stemming. The **TfidfVectorizer** bag-of-words method [7] was then utilized to obtain a vector array representation for each book corpus. The feature extraction method was configured to ignore terms appearing in over 50% of the documents and terms not present in at least 5 documents, resulting in a final matrix size of (6273 books, 6762 terms) with 0.6% non-zero entries (sparse matrix).

For clustering, two common methods were explored: Hierarchical Clustering and K-means.

Clustering **Hierarchical clustering** employed the cosine similarity distance, which tends to be very effective with text data and provides high interpretability, to generate a distance matrix for the clustering algorithm. The agglomerative approach with the 'ward' cluster linkage method produced a dendrogram (Figure 5), but poor separation between clusters due to high dimensionality was observed.

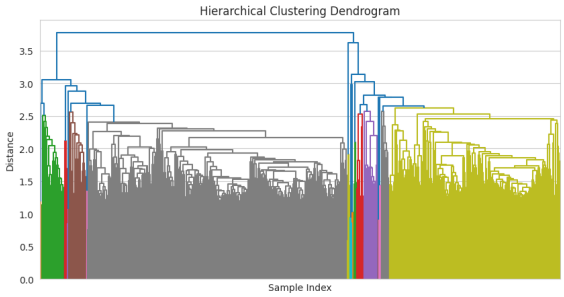
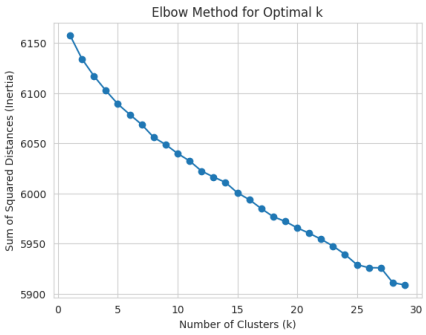


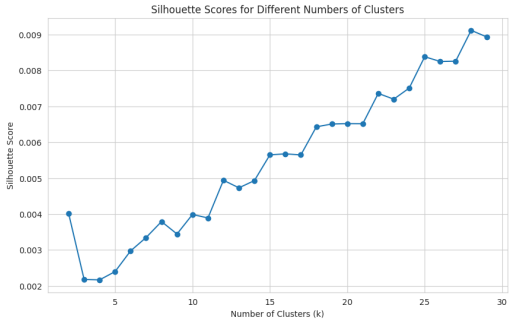
Figure 5: Dendogram from Hierarchical Clustering

KMeans was then applied, due to its simplicity and scalability, using the silhouette coefficient as a metric for clustering quality. Dealing with sparse, high-dimensional data, dimensionality reduction via SVD (Latent Semantic Analysis [8]) was employed, retaining 72.3% explained variance with 1500 components. To address imbalanced clusters from random initialization, 10 runs were performed. The elbow method in Figure either the silhouette in Figure 6 for optimal k were inconclusive due to the curse of dimensionality, so 20 classes were chosen based on domain

knowledge and specific goals of analysis. K-Means clustering yielded a silhouette score of 0.02. Despite the lower score, meaningful groupings were observed in word clouds (Figures 7-9), highlighting clusters focused on detective fiction, vampire lore, and war. Consequently, the book categorization obtained from this approach was used to enhance the content based book recommendation system.



(a) Elbow method



(b) Silhouette Score

Figure 6: Finding optimal number of clusters

Figure 7: Fantasy Topics

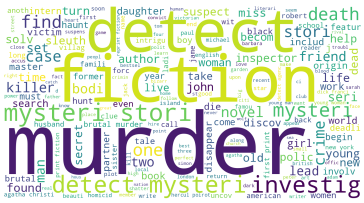


Figure 8: Detective Topics

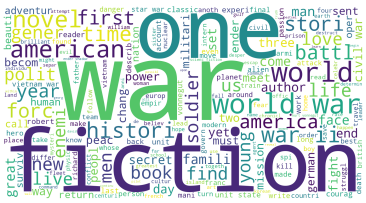


Figure 9: War Topics

4.1.1 Recommendation and Evaluation

The recommendation algorithm initiates by establishing a user profile, by taking their highest-rated book in this case. Subsequently, it identifies the cluster to which this book belongs and computes the similarity by measuring distances within that specific cluster. The algorithm then selects the top 5 books within the same cluster that exhibit the highest similarity to the user's preferred book.

Table 2 shows an example of a recommendation for user 107784 given its preference Jurassic Park:

Book title	**Author**	**Genre**
Mourning Glory	Warren Adler, Charlotte Smith	Divorced women
Let the Wind Speak	Juan Carlos Onetti	Men
Left Behind	Tim LaHaye	FICTION
The Best Baby Name Book in the Whole Wide World	Bruce Lansky	English Language
A Journal of the Seasons on an Ozark Farm	Leonard Hall	History

Table 2: Book recommendation

4.2 Pros and Cons of Content Based Book Recommendation System

Content-based recommendation systems offer advantages over other types by not requiring data from other users, addressing the cold start problem [9]. They provide personalized recommendations based on item characteristics and user preferences, particularly useful for domain-specific recommendations such as this case. However, their effectiveness depends on the quality of item feature representation, and they may lack diversity, relying heavily on clustering and similarities with past user interactions.

5 Collaborative filtering System

5.1 Methodology

Collaborative filtering is a method used in recommendation systems to suggest items, in this case books to users. It analyzes the preferences and behaviors of various users, then makes recommendations based on similarities among them. There are two main types: User-Based and Item-based collaborative Filtering. Collaborative filtering overcomes the limitations of content-based systems, which only suggest items similar to what the user has already liked and cannot capture diverse preferences or provide cross-genre recommendations. It also adds a personal 'taste' to recommendations, unlike content-based systems that offer the same suggestions to everyone.

In the dataset, using the SVD method, due to its robustness and ability to generalize and recognize local patterns, the Root Mean Square Error resulted in approximately 1.65 points. Ideally, this value should be less than 1. When the dataset is limited in user ratings or if only a few numbers of users rate a book, SVD, like other collaborative filtering techniques often bases its predictions on the mean, which can differ from the actual value. However, an RMSE of 1.65, although not ideal, is not overly high.

5.1.1 Recommendation and Evaluation

Beyond evaluating the RMSE, a deeper interpretation of the model was performed to generate a top 10 book recommendations for user 11676. Additionally, a random sample of user ratings was removed from the training model to allow for a comparison with the actual rating scores. An examination of the user's preferences revealed a strong interest in fiction genre and the book recommendation system accurately suggested many fiction books, as shown in the table 3. This demonstrates the strength of Collaborative filtering methods in capturing users' 'taste'.

ISBN	TITLE	CATEGORY
0553270931	Still Life with Woodpecker	Fiction
006017143X	The Night Listener	Fiction
0515134481	Circus of the Damned	Fiction
0345339711	The Two Towers	Fiction
0345339681	The Hobbit	Fiction
...	...	...
1558744630	Chicken Soup for the Teenage Soul	Christian life

Table 3: Top recommendations for User 11676

In predicting the score ratings of the removed ratings, the model had similar approximations in multiple instances (e.g: actual 8, predicted 7.9). However, there were also notable discrepancies (e.g: actual: 5, predicted: 8.62), as shown in the table 4. While the model can reasonably predict user preferences, it tends to overestimate higher values. This skew in prediction accuracy is likely due to the imbalance of the dataset, where more than 75% of total rating falls within 7 and 10.

User-ID	ISBN	Book-Rating	Predicted Book-Rating
6073	0440226430	6	8.17
6073	0440214041	5	8.62
...	...	...	...
11676	0971880107	6	5.27
11676	0316666343	5	9.47
11676	0385504209	9	9.99
...	...	...	...
23768	0060934417	8	7.90
23768	0316666343	7	7.54

Table 4: Comparison of Actual and Predicted Book Ratings

5.2 Pros and Cons of Collaborative filtering Systems

The main advantage of collaborative filtering is its ability to provide personalized book recommendations to each user. In analysis of the top 10 recommendations, the SVD model effectively captures the 'taste' of User '11676' and recommend books particularly in the fiction category. This is significant, as the SVD model relies solely on book rating scores. In the sample analysis, the SVD model generally predicts rating scores close to the actual values, which suggests that users will likely be satisfied with the recommendations. However, a major disadvantage of collaborative filtering methods is their dependency on a large amount of data, requiring multiple ratings per book. Additionally, the dataset is imbalanced, which could lead to over-fitting in the model and resulting in mean-based predictions for new recommendations, due to the lack of diverse reviews.

6 Discussion

The challenges in recommendation systems vary across methodologies. Association rules face limitations in catering to diverse user tastes due to a focus on common elements. Content-based approaches, while effective against the cold start problem, struggle with capturing the full range of user preferences due to high dimensionality. Collaborative filtering excels in personalized recommendations but faces challenges with sparse data and imbalanced ratings. To address these issues, a potential enhancement involves combining content-based and collaborative techniques to create a hybrid system, leveraging strengths and mitigating individual limitations for a more robust solution. Investigating these recommendation system approaches is crucial for understanding their distinct characteristics and limitations, providing valuable insights for the development and improvement of future systems. This exploration not only enhances comprehension of user preferences but also guides the pursuit of a harmonious balance between common patterns and individual tastes, contributing to the creation of more robust and adaptable recommendation systems.

Appendix

Student's Contribution

According to the DTU regulations, a student's contribution to the report must be clearly specified. Therefore, for each section, specify who was responsible for it.

Section	Manfrin	Aniol	Barbara	Eduard	Javier
Introduction	15%	15%	40%	15%	15%
Dataset	15%	15%	15%	15%	40%
Association Rules	15%	15%	40%	15%	15%
Content Based Recommendation System	10%	35%	10%	35%	10%
Collaborative Filtering Recommendation System	40%	15%	15%	15%	15%
Discussion	15%	15%	30%	15%	25%

References

[1] MÖBIUS. *Book Recommendation Dataset*. [Online; accessed 10-November-2023]. URL: {https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset}.

[2] Google. *Google Books API*. [Online; accessed 12-November-2023]. URL: {https://developers.google.com/books/docs/v1/using}.

[3] anbipa. *Github Codes*. [Online; accessed 27-November-2023]. 2023. URL: {https://github.com/anbipa/BookRecommenderCT}.

[4] Mohammad ElBes Shadi AlZu'bi1 Bilal Hawashin and Mahmoud Al-Ayyoub. "A Novel Recommender System based on Apriori Algorithm for Requirements Engineering". In: (2018).

[5] Java Point. *Data Mining Techniques*. [Online; accessed 15-November-2023]. URL: {https://www.javatpoint.com/fp-growth-algorithm-in-data-mining}.

[6] Michael J. Pazzani and Daniel Billsus. "Content-Based Recommendation Systems". In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_10. URL: https://doi.org/10.1007/978-3-540-72079-9_10.

[7] scikitlearn. $sklearn.feature_extraction.text.TfidfVectorizer$. [Online; accessed 09-November-2023]. URL: {https://scikitlearn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html}.

[8] Wikipedia. *Latent Semantic Analysis*. [Online; accessed 12-November-2023]. URL: {https://en.wikipedia.org/wiki/Latent_semantic_analysis}.

[9] Wikipedia. *Cold Start (Recommender Systems)*. [Online; accessed 10-November-2023]. URL: {https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)}.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.frequent_patterns import fpgrowth, association_rules
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/pexpect/popen_spawn.py:60: DeprecationWarnin
g: setDaemon() is deprecated, set the daemon attribute instead
  self._read_thread.setDaemon(True)
Mounted at /content/drive
```

Data Preprocessing

Data Preprocessing has already been executed for you and the preprocessed data is ready for consumption in the Github repository.

Limiting the dataset to 6185 samples

In [ ]:
```
'''
df_users = pd.read_csv('/content/drive/MyDrive/DTU/CT/Data Science project/BOOK RECOMM
df_books = pd.read_csv('/content/drive/MyDrive/DTU/CT/Data Science project/BOOK RECOMM

df_books=df_books.iloc[:,:-4]
df_books.head()
'''
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
<ipython-input-4-9d2739b6d64a>:2: DtypeWarning: Columns (3) have mixed types. Specify
dtype option on import or set low_memory=False.
  df_books = pd.read_csv('/content/drive/MyDrive/DTU/CT/Data Science project/BOOK REC
OMMENDATION/Books.csv')
```

Out[ ]:

	ISBN	Book-Title	Book-Author	Year-Of-Publication
0	0195153448	Classical Mythology	Mark P. O. Morford	2002
1	0002005018	Clara Callan	Richard Bruce Wright	2001
2	0060973129	Decision in Normandy	Carlo D'Este	1991
3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999
4	0393045218	The Mummies of Urumchi	E. J. W. Barber	1999

In [ ]:
```
'''
unique_ISBN=list(df_books['ISBN'].unique())
len(unique_ISBN)
'''
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
271360
```

In [ ]:
```
'''
df_books = df_books[df_books['ISBN'].isin(unique_ISBN)]
df_books = df_books.sample(frac=1).reset_index(drop=True)
len(df_books)
'''
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
271360
```

To enhance the depth and quality of our dataset, we performed a targeted retrieval of additional bibliographic details for a randomly selected subset of books. This extra information was extracted through the Google Books API, which allowed us to supplement our records with a wealth of metadata, including titles, subtitles, authors, publication dates, descriptions, page counts, categories, and language specifications.

In [ ]:
```
'''
import requests
import pandas as pd
import os

def get_book_details(isbn):
    """ Fetch book details from Google Books API using ISBN. """
    url = f'https://www.googleapis.com/books/v1/volumes?q=isbn:{isbn}'
    response = requests.get(url)
    if response.status_code == 200:
```

```
        results = response.json()
        if results.get('totalItems', 0) > 0:
            book = results['items'][0]['volumeInfo']
            return {
                'isbn': isbn,
                'title': book.get('title', ''),
                'subtitle': book.get('subtitle', ''),
                'authors': ', '.join(book.get('authors', [])),
                'publisher': book.get('publisher', ''),
                'publishedDate': book.get('publishedDate', ''),
                'description': book.get('description', ''),
                'pageCount': book.get('pageCount', 0),
                'categories': ', '.join(book.get('categories', [])),
                'language': book.get('language', '')
            }
    return {}

# Lists to store extracted information
details_list = []
count = 0
excel_path = "/content/drive/MyDrive/DTU/CT/Data Science project/BOOK RECOMMENDATION/B

for isbn in df_books['ISBN']:
    book_details = get_book_details(isbn)
    if book_details and book_details['description'] and book_details['language'] == 'e
        details_list.append(book_details)
        count += 1
        if count % 100 == 0:
            # Check if the file exists
            file_existsh = os.path.isfile(excel_path)
            temp_df = pd.DataFrame(details_list)
            # Write to Excel file
            temp_df.to_csv(excel_path, mode='a', header=not file_exists, index=False)
            print(f"Appended {count} books to the Excel file.")
            # Clear the details list for the next batch
            details_list = []

    # Finish the dowload after 685 books
    if len(details_list) >= 6185:
      break

df_books = pd.DataFrame(details_list)
'''
```

In [ ]:
```
#df_books.to_excel("/content/drive/MyDrive/DTU/CT/Data Science project/BOOK RECOMMENDA
```

In [ ]:
```
#df = pd.read_csv('/content/drive/MyDrive/DTU/CT/Data Science project/BOOK RECOMMENDAT
```

In [ ]:
```
#df = df[df['Book-Rating'] != 0]
#df = df.drop_duplicates(['User-ID', 'ISBN'])
```

In [ ]:
```
#df = df.sort_values(by='User-ID')
```

In [ ]:
```
#unique_ISBN=list(df_books['isbn'].unique())
#len(unique_ISBN)
```

In [ ]:
```
#df = df[df['ISBN'].isin(unique_ISBN)]
```

In [ ]:
```
#len(df)
```

In [ ]:
```
#df.to_csv("/content/drive/MyDrive/DTU/CT/Data Science project/BOOK RECOMMENDATION/RAT
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Data Visualization

In [ ]:
```
df = pd.read_csv("https://raw.githubusercontent.com/anbipa/BookRecommenderCT/main/RATI
df_books = pd.read_csv("https://raw.githubusercontent.com/anbipa/BookRecommenderCT/mai
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Ratings

In [ ]:
```
df_ratings=df
df_ratings
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	Unnamed: 0	User-ID	ISBN	Book-Rating
0	9591	16	0345402871	9
1	9594	17	0425099148	7
2	9608	26	0449005615	9
3	9607	26	0446310786	10
4	9616	39	0553582909	8
...	...	...	...	...
66077	9463	278843	014028009X	8
66078	9521	278844	0679781587	7
66079	9548	278851	1558531025	8
66080	9536	278851	0440486599	5
66081	9560	278854	0553579606	8

66082 rows × 4 columns

In [ ]:
```
df_ratings
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	Unnamed: 0	User-ID	ISBN	Book-Rating
0	9591	16	0345402871	9
1	9594	17	0425099148	7
2	9608	26	0449005615	9
3	9607	26	0446310786	10
4	9616	39	0553582909	8
...	...	...	...	...
66077	9463	278843	014028009X	8
66078	9521	278844	0679781587	7
66079	9548	278851	1558531025	8
66080	9536	278851	0440486599	5
66081	9560	278854	0553579606	8

66082 rows × 4 columns

In [ ]:
```
df_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66082 entries, 0 to 66081
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Unnamed: 0   66082 non-null  int64 
 1   User-ID      66082 non-null  int64 
 2   ISBN         66082 non-null  object
 3   Book-Rating  66082 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 2.0+ MB
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

The dataset has a total of 66082 ratings. The information is organized into three columns: 'User-ID,' representing quantitative attributes in the form of user identifiers; 'ISBN,' a qualitative column containing identification codes for books; and 'Book-Rating,' another quantitative column recording ratings given to the books.

In [ ]:
```
df_ratings.drop(columns='Unnamed: 0', inplace=True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
df_ratings.describe().T
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
User-ID	66082.0	137380.074423	80573.603712	16.0	67840.0	135149.0	208509.25	278854.0
Book-Rating	66082.0	7.781030	1.792506	1.0	7.0	8.0	9.00	10.0

The provided statistics reveal a distribution of ratings that is notably skewed towards higher values, with a mean rating of approximately 7.78, close to the upper limit of the 1 to 10 scale. The low standard deviation of 1.79 indicates limited variability in ratings, suggesting a concentration around the mean. The fact that 25% of ratings fall at 7 or lower (25th percentile), and the median is 8.0, reflecting that half of the ratings are 8 or lower, signifies a scarcity of low ratings. This distribution suggests a bias towards positive ratings, a phenomenon common in recommendation systems and user-generated content platforms where users tend to rate items they enjoyed more positively. Recognizing this rating pattern is essential for interpreting user sentiment and satisfaction levels within the dataset.

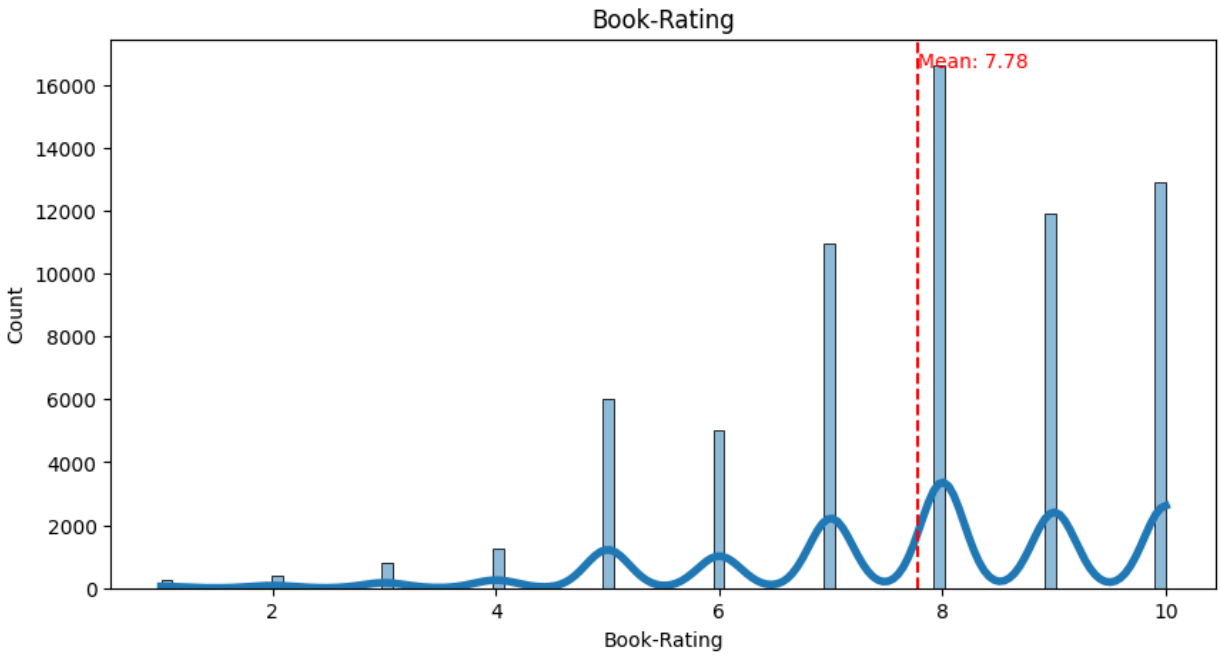
In [ ]:
```
unique_user_ids = df_ratings['User-ID'].nunique()

print("Number of Unique Users:", unique_user_ids)
```

```
Number of Unique Users: 24933
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Distribution of numerical variables

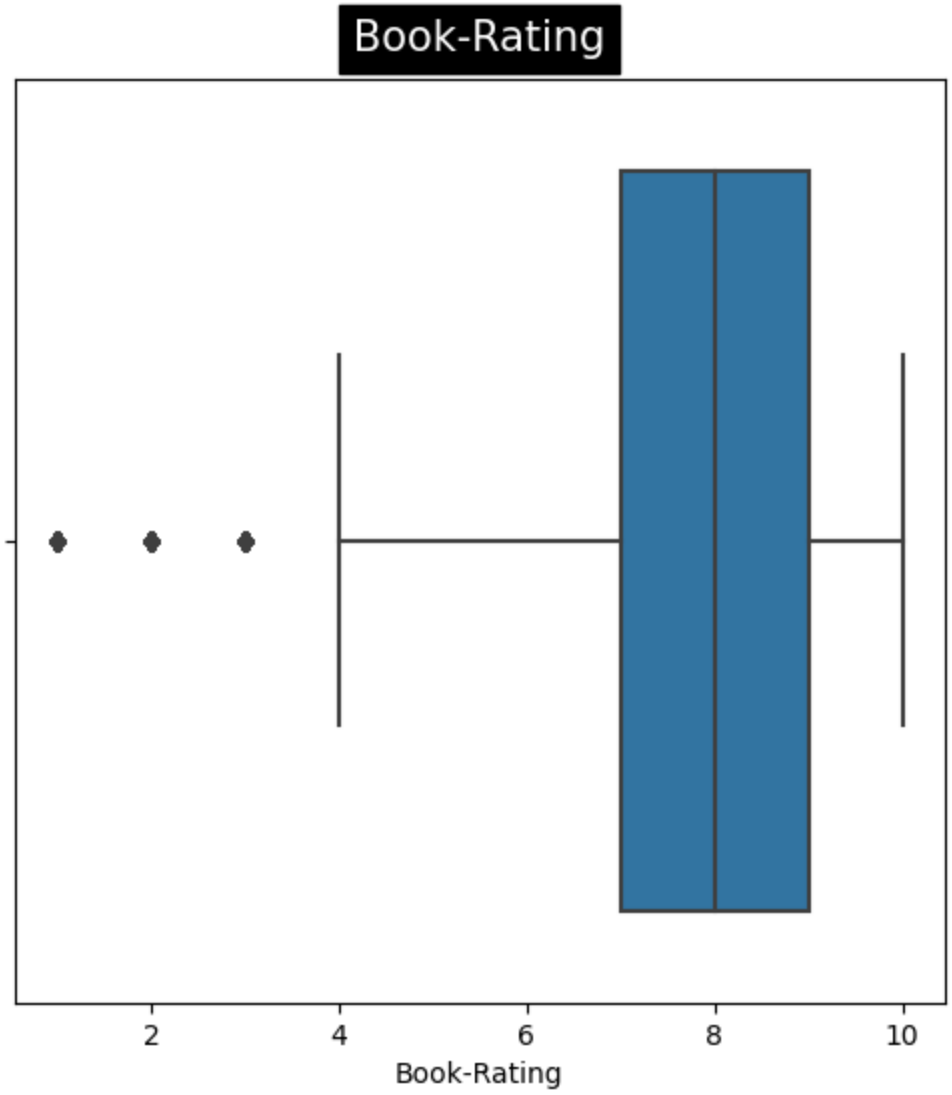
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the column for 'rating'
column_to_plot = 'Book-Rating'

# Convert 'rating' to numeric, coerce non-numeric values to NaN
df_ratings[column_to_plot] = pd.to_numeric(df_ratings[column_to_plot], errors='coerce'

# Drop rows with NaN values in 'rating'
df_ratings = df_ratings.dropna(subset=[column_to_plot])

# Create a histogram for the specified column
plt.figure(figsize=(10, 5))
sns.histplot(df_ratings[column_to_plot], kde=True, line_kws={'linewidth': 4, 'color':
plt.title(column_to_plot)

# Add mean line and annotation
mean_val = df_ratings[column_to_plot].mean()
plt.axvline(mean_val, color='red', linestyle='--')
plt.text(mean_val, 0.95 * plt.ylim()[1], f'Mean: {mean_val:.2f}', color='red')

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



In [ ]:
```
fig, ax = plt.subplots(figsize=(6, 6))

# Create a boxplot for the 'Rating' column
sns.boxplot(data=df_ratings, x='Book-Rating', ax=ax)
ax.set_title('Book-Rating', backgroundcolor='black', color='white', fontsize=15, pad=1

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



Books

In [ ]:
```
df_books
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCo
0	0884115631	The Shepherd	NaN	Frederick Forsyth	NaN	1960-06-01	A pilot and his plane are saved by a mysteriou...	
1	0060175648	Identity	A Novel	Milan Kundera	Harper	1998-04-21	Milan Kundera's Identity translated from the F...	
2	0394535383	Pillar of the Sky	A Novel	Cecelia Holland	Alfred a Knopf Incorporated	1985	Two thousand years ago, at the site of Stonehe...	
3	0843114401	Jingle Bear	NaN	Stephen Cosgrove	NaN	1985	When the other bears get ready to hibernate fo...	
4	0451207408	The Adventurer	NaN	Jaclyn Reding	Signet Book	2002	While attempting to return a mysterious, legen...	
...	...	...	...	...	...	...	...	
6268	0425092917	The Accidental Tourist	NaN	Anne Tyler	Berkley	1986	Meet Macon Leary--a travel writer who hates bo...	
6269	0743439740	Every Breath You Take	A True Story of Obsession, Revenge, and Murder	Ann Rule	Pocket Books	2002-12-01	America’s #1 true-crime writer fulfills a murd...	
6270	0449213943	All Quiet on the Western Front	A Novel	Erich Maria Remarque	Ballantine Books	1982	The masterpiece of the German experience durin...	
6271	0446611921	The Millionaires	NaN	Brad Meltzer	Warner Books (NY)	2002	Charlie and Oliver, brothers and bankers at a ...	

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCo
6272	0399148639	Mortal Prey	NaN	John Sandford	Putnam Adult	2002	Nearly killed in a hit attempt, retired top hi...	

6273 rows × 10 columns

In [ ]:
```
# Assuming 'isbn' is the current column name
df_books.rename(columns={'isbn': 'ISBN'}, inplace=True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
unique_user_ids = df_books['ISBN'].nunique()

print("Number of Unique Books:", unique_user_ids)
```

```
Number of Unique Books: 6273
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
#keep only the year
df_books['publishedDate'] = df_books['publishedDate'].apply(lambda x: str(x)[:4] if pd
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
df_books.info()
```

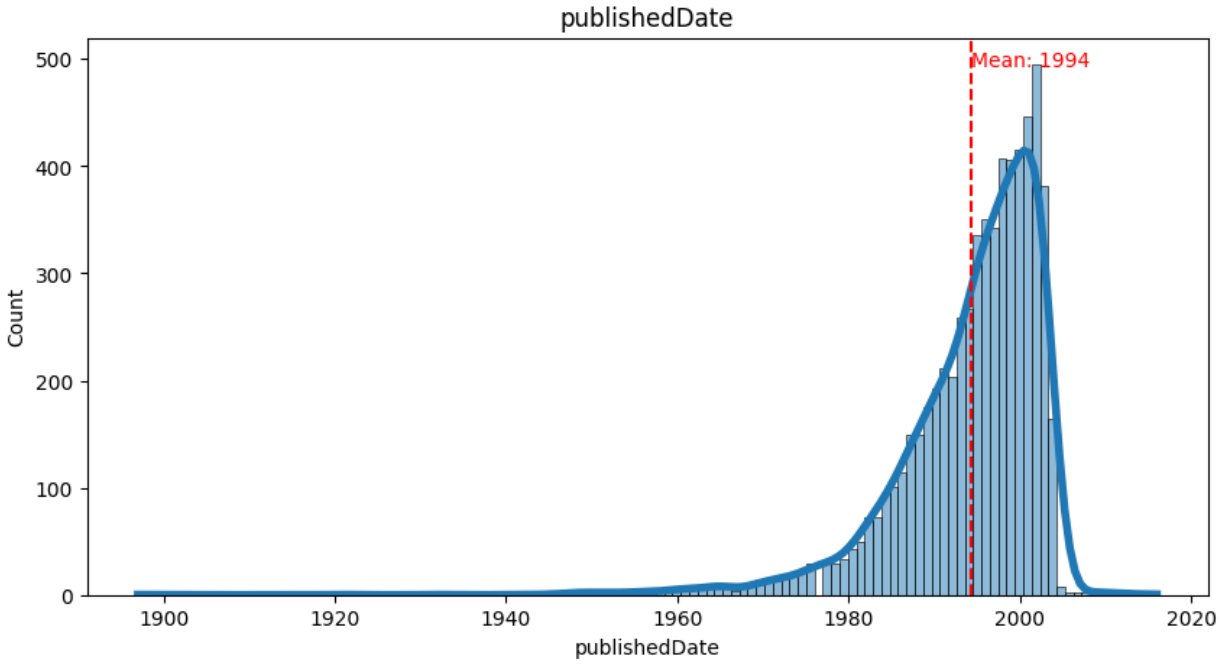
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6273 entries, 0 to 6272
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   ISBN           6273 non-null   object
 1   title          6273 non-null   object
 2   subtitle       2005 non-null   object
 3   authors        6216 non-null   object
 4   publisher      5348 non-null   object
 5   publishedDate  6268 non-null   object
 6   description    6273 non-null   object
 7   pageCount      6273 non-null   int64 
 8   categories     6157 non-null   object
 9   language       6273 non-null   object
dtypes: int64(1), object(9)
memory usage: 490.2+ KB
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Distribution of numerical variables

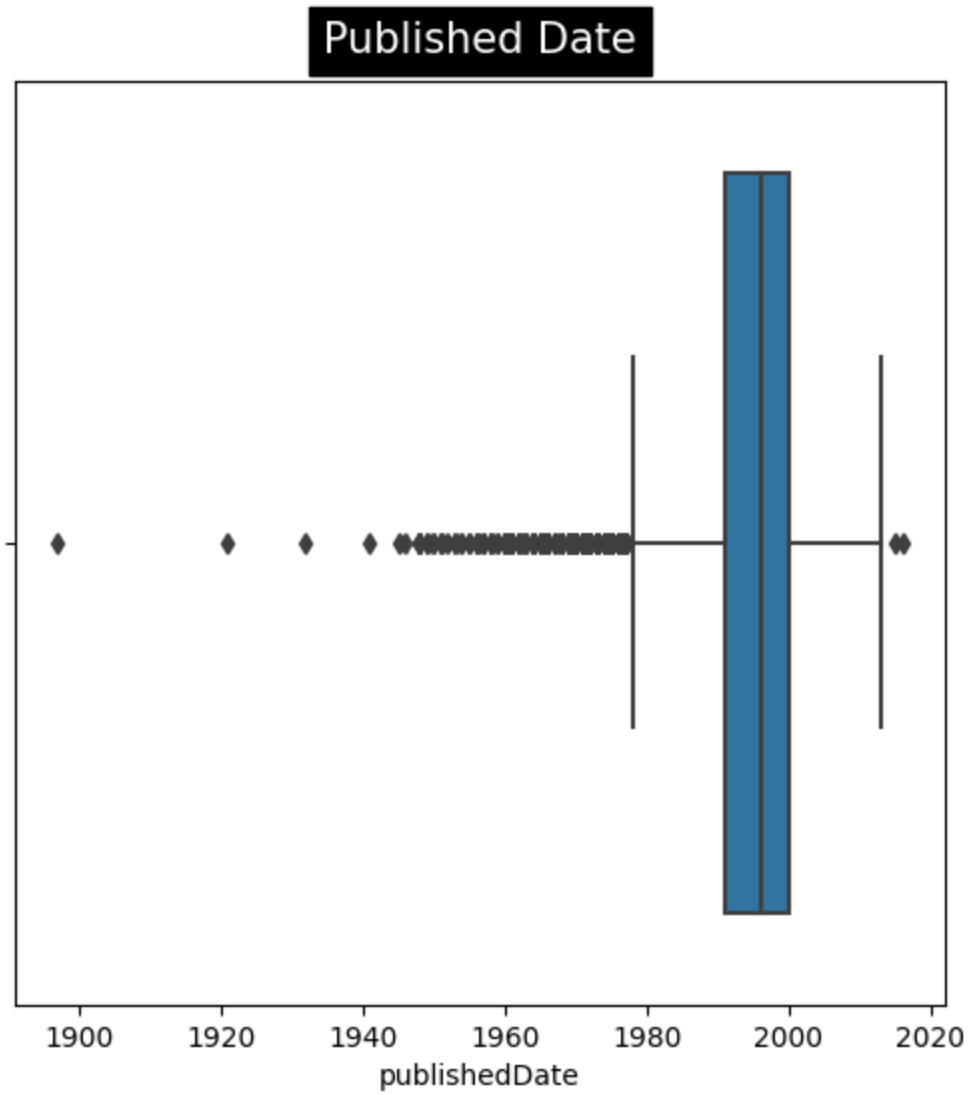
In [ ]:
```
#published Date
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the column for 'Year-Of-Publication'
column_to_plot = 'publishedDate'

# Convert 'Year-Of-Publication' to numeric, coerce non-numeric values to NaN
df_books[column_to_plot] = pd.to_numeric(df_books[column_to_plot], errors='coerce')

# Drop rows with NaN values in 'Year-Of-Publication'
df_books = df_books.dropna(subset=[column_to_plot])

# Create a histogram for the specified column
plt.figure(figsize=(10, 5))
sns.histplot(df_books[column_to_plot], kde=True, line_kws={'linewidth': 4, 'color': 'r
plt.title(column_to_plot)

# Add mean line and annotation
mean_val = df_books[column_to_plot].mean()
plt.axvline(mean_val, color='red', linestyle='--')
plt.text(mean_val, 0.95 * plt.ylim()[1], f'Mean: {mean_val:.0f}', color='red')

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



In [ ]:
```
fig, ax = plt.subplots(figsize=(6, 6))

# Create a boxplot for the 'Rating' column
sns.boxplot(data=df_books, x='publishedDate', ax=ax)
ax.set_title('Published Date', backgroundcolor='black', color='white', fontsize=15, pa

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

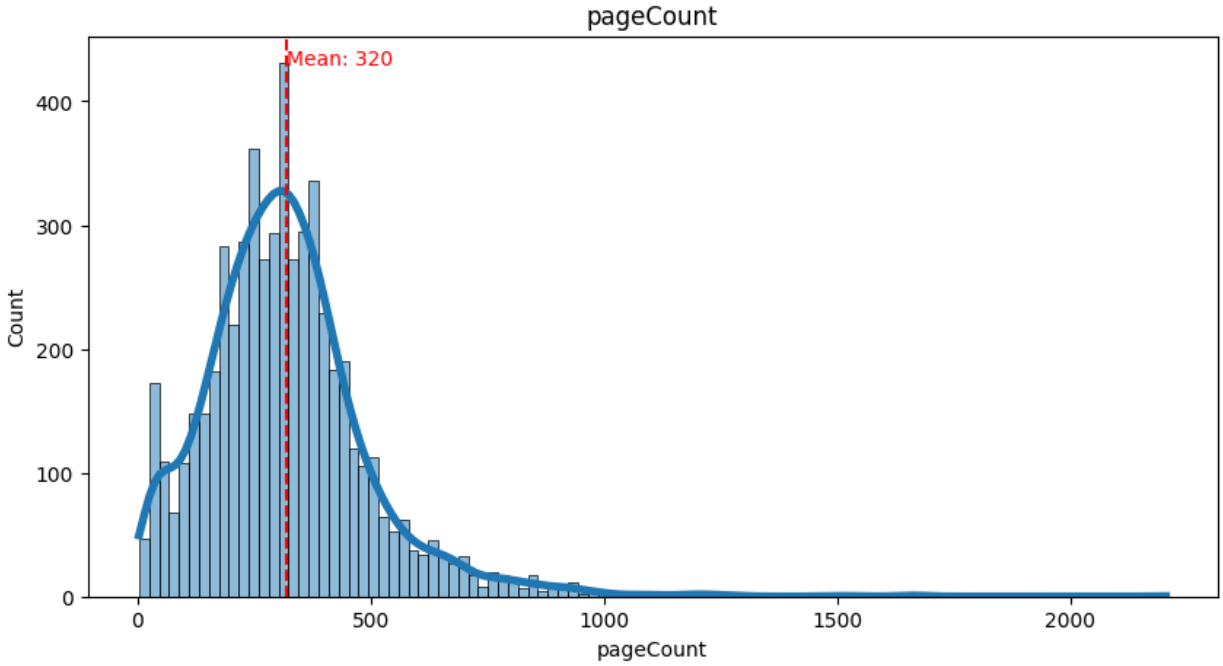


The majority of books in the dataset were published between the years 1980 and 2004.

In [ ]:
```
#pageCount
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the column for 'pageCount'
column_to_plot = 'pageCount'

# Convert 'pageCount' to numeric, coerce non-numeric values to NaN
df_books[column_to_plot] = pd.to_numeric(df_books[column_to_plot], errors='coerce')

# Drop rows with NaN or 0 values in 'pageCount'
df_books = df_books[(df_books[column_to_plot].notna()) & (df_books[column_to_plot] !=

# Create a histogram for the specified column
plt.figure(figsize=(10, 5))
sns.histplot(df_books[column_to_plot], kde=True, line_kws={'linewidth': 4, 'color': 'r
plt.title(column_to_plot)

# Add mean line and annotation
mean_val = df_books[column_to_plot].mean()
plt.axvline(mean_val, color='red', linestyle='--')
plt.text(mean_val, 0.95 * plt.ylim()[1], f'Mean: {mean_val:.0f}', color='red')
```

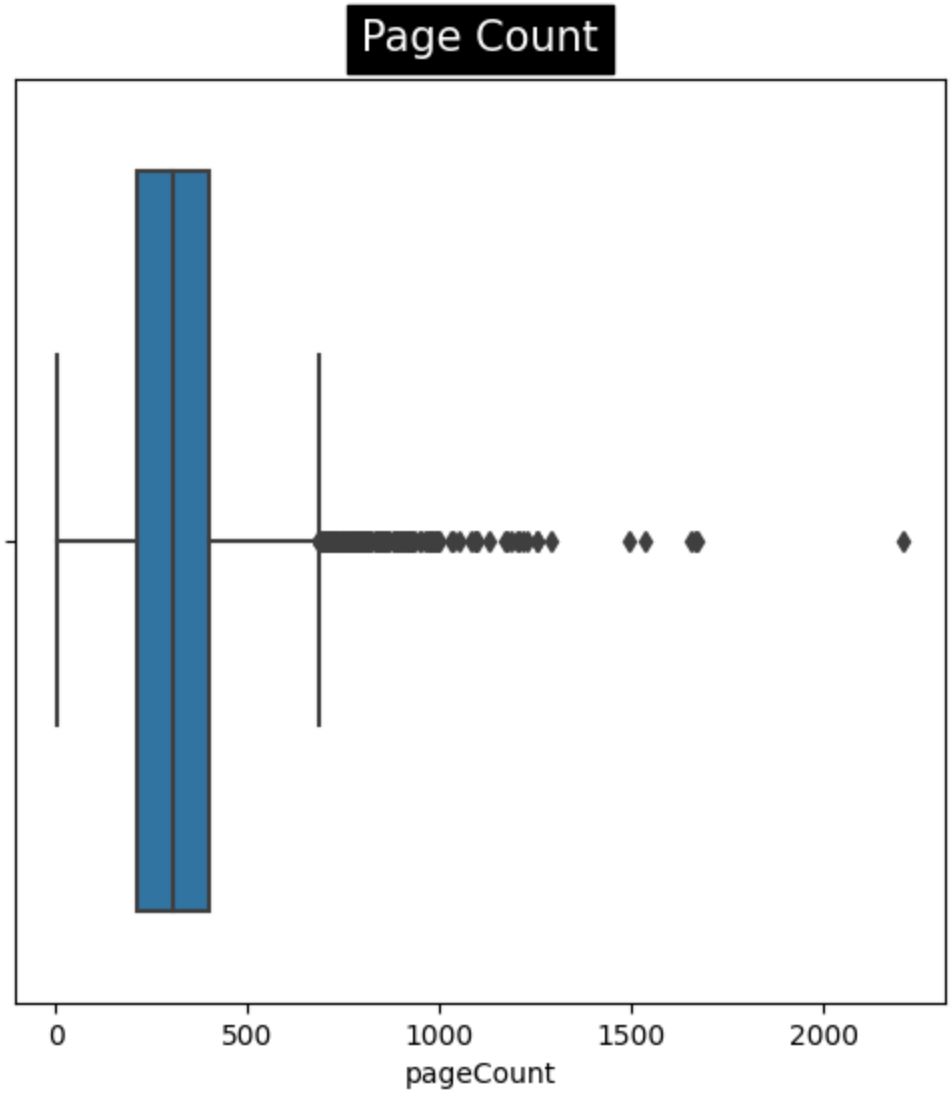
```
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
<ipython-input-19-9b67cfad52ff>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_books[column_to_plot] = pd.to_numeric(df_books[column_to_plot], errors='coerce')
```



In [ ]:
```
# Filter out rows with 0 pages
df_books_filtered = df_books[df_books['pageCount'] != 0]

# Create a boxplot for the 'pageCount' column
fig, ax = plt.subplots(figsize=(6, 6))
sns.boxplot(data=df_books_filtered, x='pageCount', ax=ax)
ax.set_title('Page Count', backgroundcolor='black', color='white', fontsize=15, pad=10

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



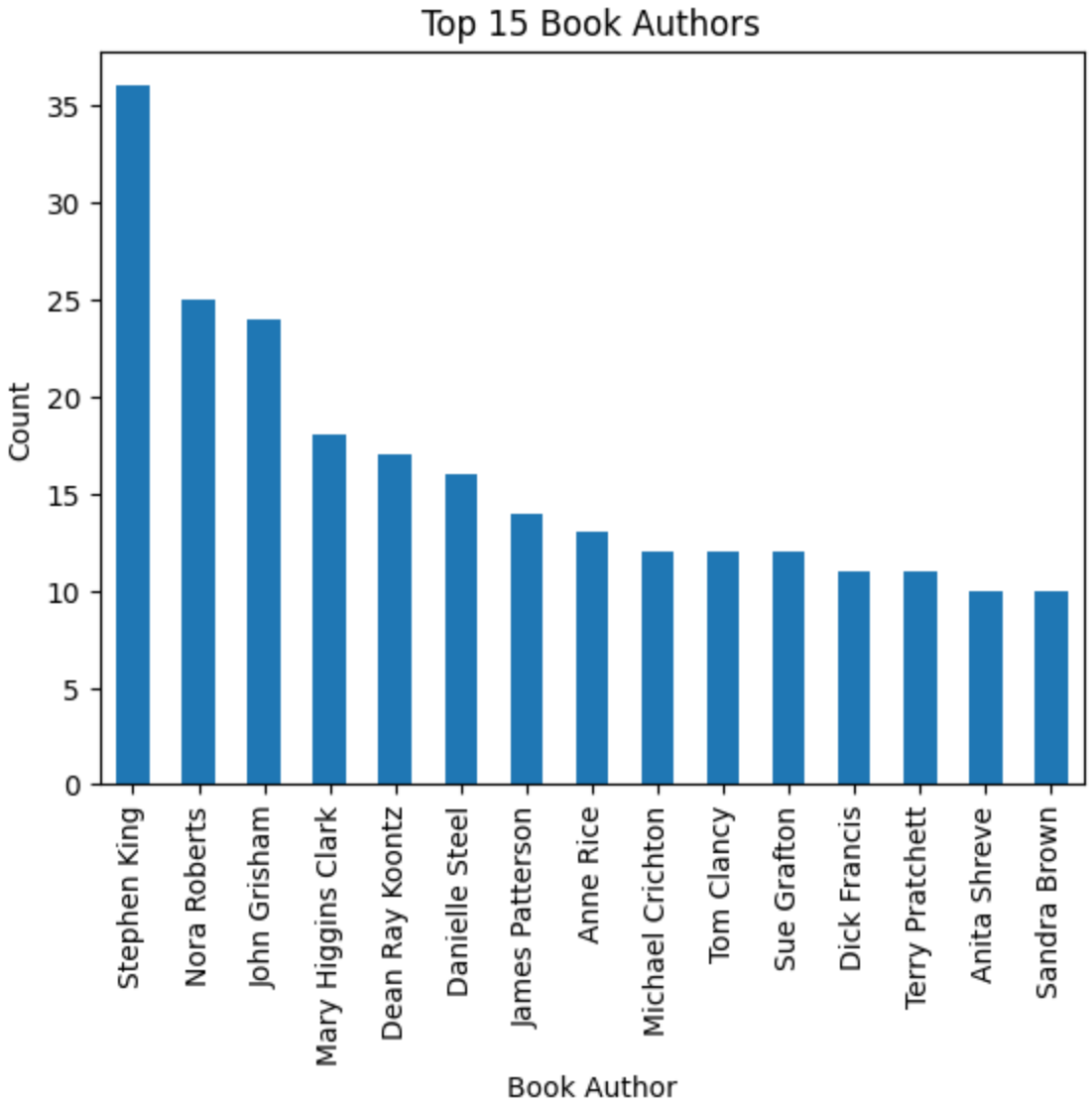
We disregard books with 0 pages as it indicates missing information, and therefore, they are not considered in our analysis. The average page count for the remaining books is 320 pages.

Distribution of categorical variables

In [ ]:
```
import matplotlib.pyplot as plt

# Get the value counts for 'authors'
author_counts = df_books['authors'].value_counts()

# Find the author with the most books
most_books_author = author_counts.idxmax()
num_books = author_counts.max()

# Print the result
print(f"The author with the most books is {most_books_author} with {num_books} books."

# Select the top 15 authors
top_authors = author_counts.head(15)

# Plot the bar chart with X as 'Book Author' and Y as the count
top_authors.plot(kind='bar')
```

```
# Labeling the axes
plt.xlabel('Book Author')
plt.ylabel('Count')
plt.title('Top 15 Book Authors')

# Display the plot
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
The author with the most books is Stephen King with 36 books.
```

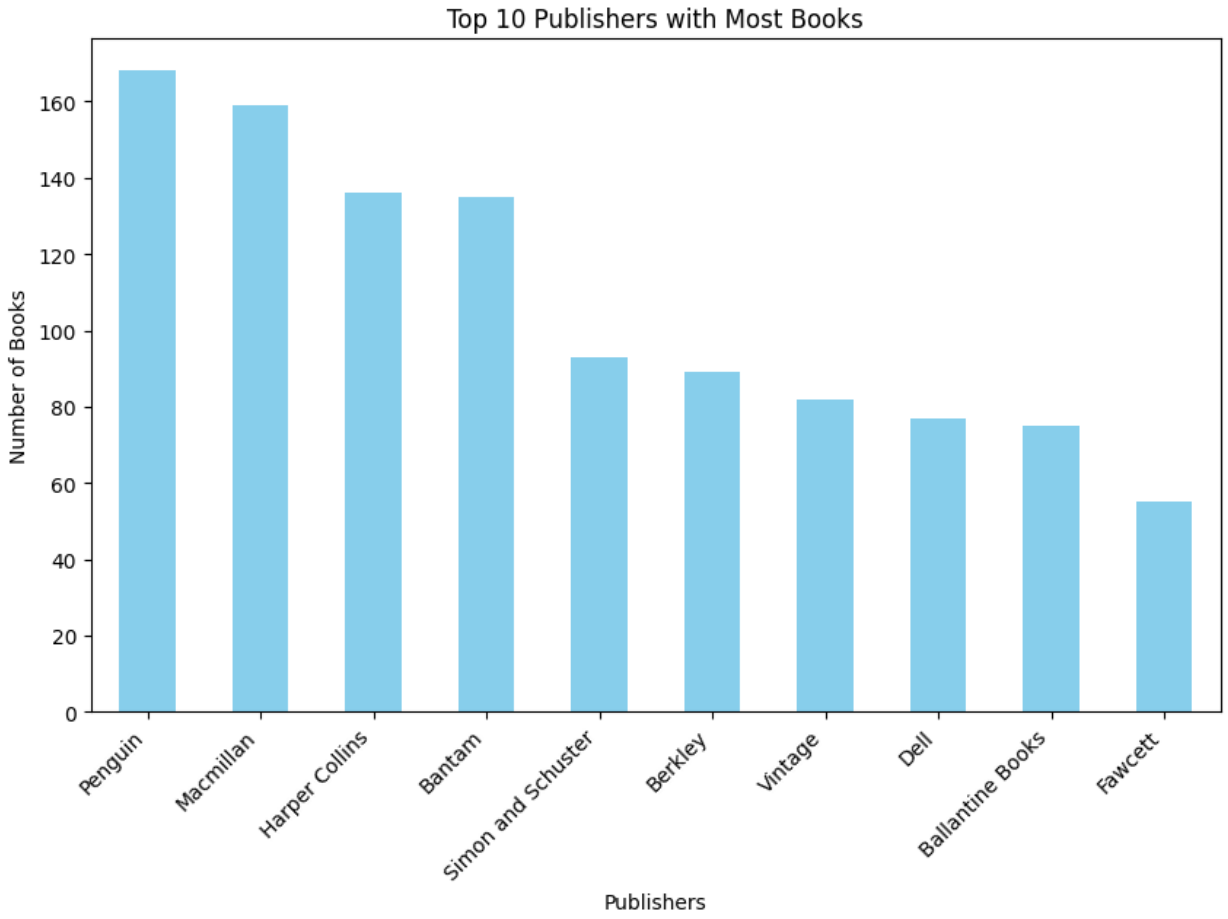


In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming df_books is your DataFrame
# Replace 'df_books' with the actual name of your DataFrame if it's different

# Calculate the count of books for each publisher
books_count = df_books['publisher'].value_counts()
```

```
# Find the publisher with the most books
most_books_publisher = books_count.idxmax()
num_books = books_count.max()

# Print the result
print(f"The publisher with the most books is {most_books_publisher} with {num_books} b

# Sort the publishers based on the number of books in descending order and select the
top_publishers = books_count.sort_values(ascending=False).head(10)

# Plot the data
plt.figure(figsize=(10, 6))
top_publishers.plot(kind='bar', color='skyblue')
plt.title('Top 10 Publishers with Most Books')
plt.xlabel('Publishers')
plt.ylabel('Number of Books')
plt.xticks(rotation=45, ha='right')
plt.show()
```

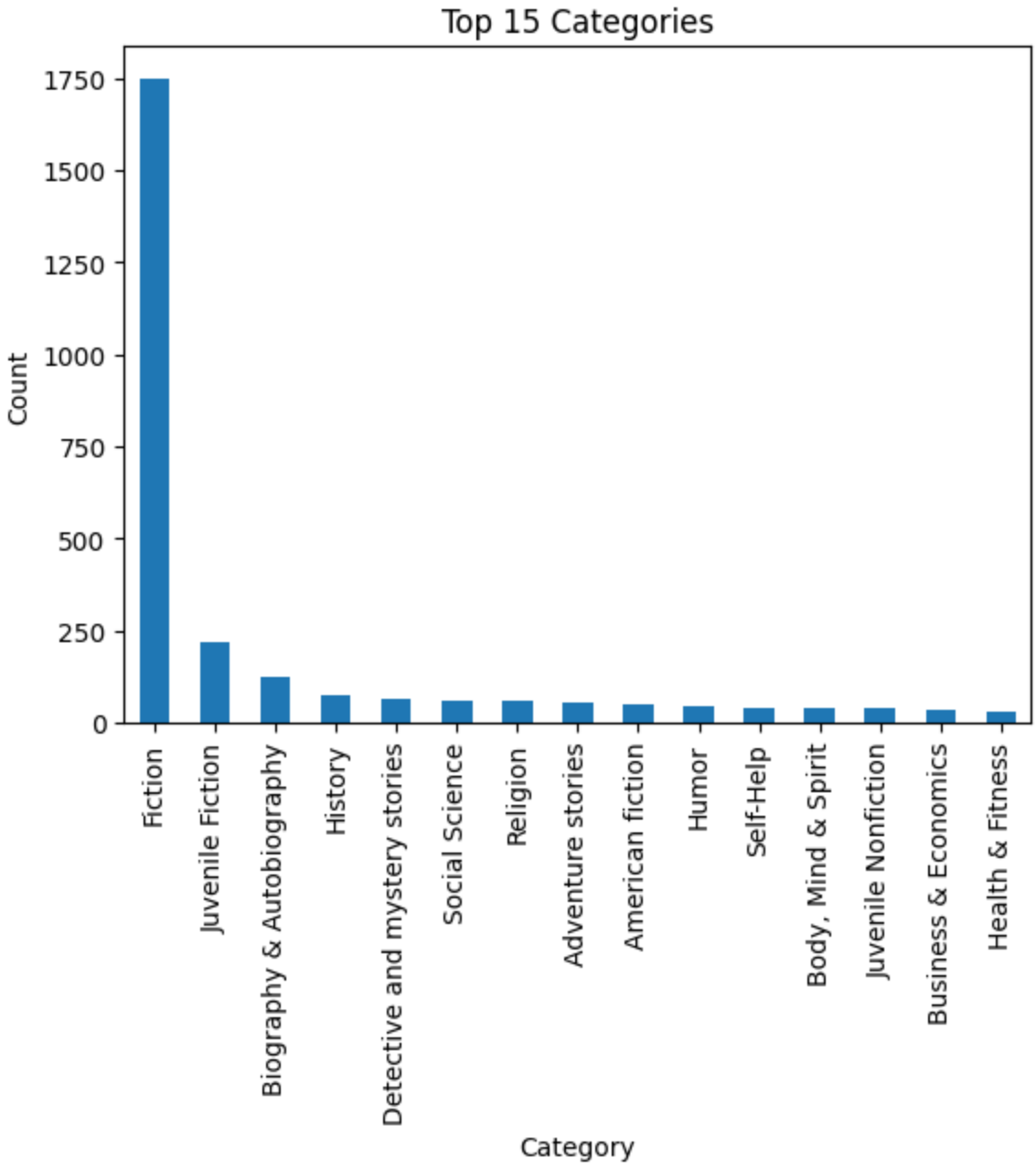
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
The publisher with the most books is Penguin with 168 books.
```



In [ ]:
```
import matplotlib.pyplot as plt

# Get the value counts for 'Category' and select the top 15
category_counts = df_books['categories'].value_counts().head(15)
```

```
# Plot the bar chart with X as 'Category' and Y as the count
category_counts.plot(kind='bar')

# Labeling the axes
plt.xlabel('Category')
plt.ylabel('Count')
plt.title('Top 15 Categories')

# Display the plot
plt.show()
```

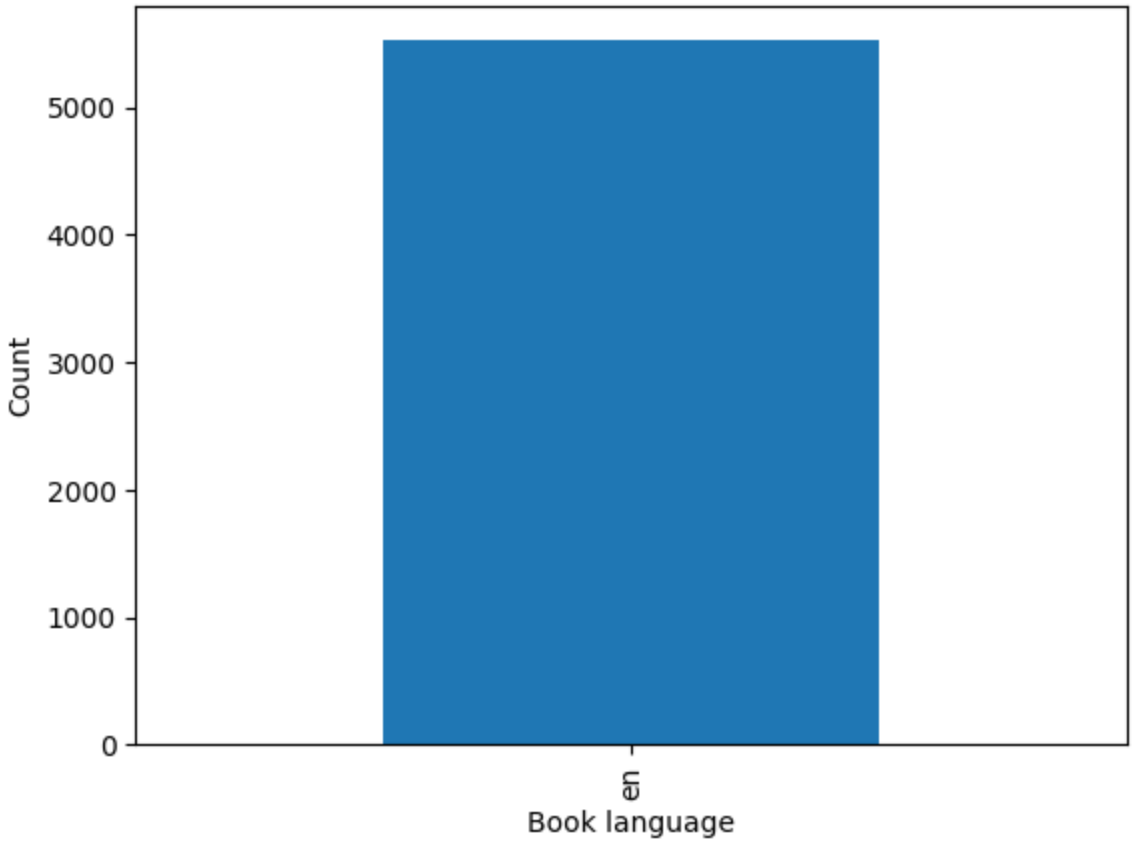
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



"Fiction" emerges as the most prevalent category in the dataset.

In [ ]:
```
import matplotlib.pyplot as plt

# Get the value counts for 'Category' and select the top 15
category_counts = df_books['language'].value_counts().head(15)

# Plot the bar chart with X as 'Category' and Y as the count
category_counts.plot(kind='bar')

# Labeling the axes
plt.xlabel('Book language')
plt.ylabel('Count')
# Display the plot
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



All the books are written in english

Merging datasets

In [ ]:
```
#Merging ratings and books dataframes on ISBN
complete_df=df_ratings.merge(df_books, on="ISBN")
print("Merging the 2 dataframes: ratings and books on ISBN")
display(complete_df)
```

```
Merging the 2 dataframes: ratings and books on ISBN
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

	User-ID	ISBN	Book-Rating	title	subtitle	authors	publisher	publishedDate
0	16	0345402871	9	Airframe	NaN	Michael Crichton	Random House Digital, Inc.	1996.0
1	10215	0345402871	3	Airframe	NaN	Michael Crichton	Random House Digital, Inc.	1996.0
2	14801	0345402871	7	Airframe	NaN	Michael Crichton	Random House Digital, Inc.	1996.0
3	19011	0345402871	9	Airframe	NaN	Michael Crichton	Random House Digital, Inc.	1996.0
4	22463	0345402871	6	Airframe	NaN	Michael Crichton	Random House Digital, Inc.	1996.0
...	...	...	...	...	...	...	...	...
61852	276626	0306809044	5	Skywalking	The Life And Films Of George Lucas, Updated Ed...	Dale Pollock	Da Capo Press	1999.0
61853	276664	0140136908	7	A History of Economic Thought	NaN	William J. Barber	Harmondsworth : Penguin	1967.0
61854	277203	0590442899	9	Clifford Goes to Hollywood	NaN	NaN	Cartwheel Books	1980.0
61855	277527	0805048782	7	Loving You Was My Undoing	A Novel	Javier Gonzalez-Rubio	Henry Holt and Company	1999.0

	User-ID	ISBN	Book-Rating	title	subtitle	authors	publisher	publishedDate
61856	278094	0761128093	8	Rambam's Ladder	A Meditation on Generosity and why it is Neces...	Julie Salamon	Workman Publishing	2003.0

61857 rows × 12 columns

In [ ]:
```
complete_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 61857 entries, 0 to 61856
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   User-ID        61857 non-null  int64  
 1   ISBN           61857 non-null  object 
 2   Book-Rating    61857 non-null  int64  
 3   title          61857 non-null  object 
 4   subtitle       21036 non-null  object 
 5   authors        61697 non-null  object 
 6   publisher      57737 non-null  object 
 7   publishedDate  61857 non-null  float64
 8   description    61857 non-null  object 
 9   pageCount      61857 non-null  int64  
 10  categories     61815 non-null  object 
 11  language       61857 non-null  object 
dtypes: float64(1), int64(3), object(8)
memory usage: 6.1+ MB
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

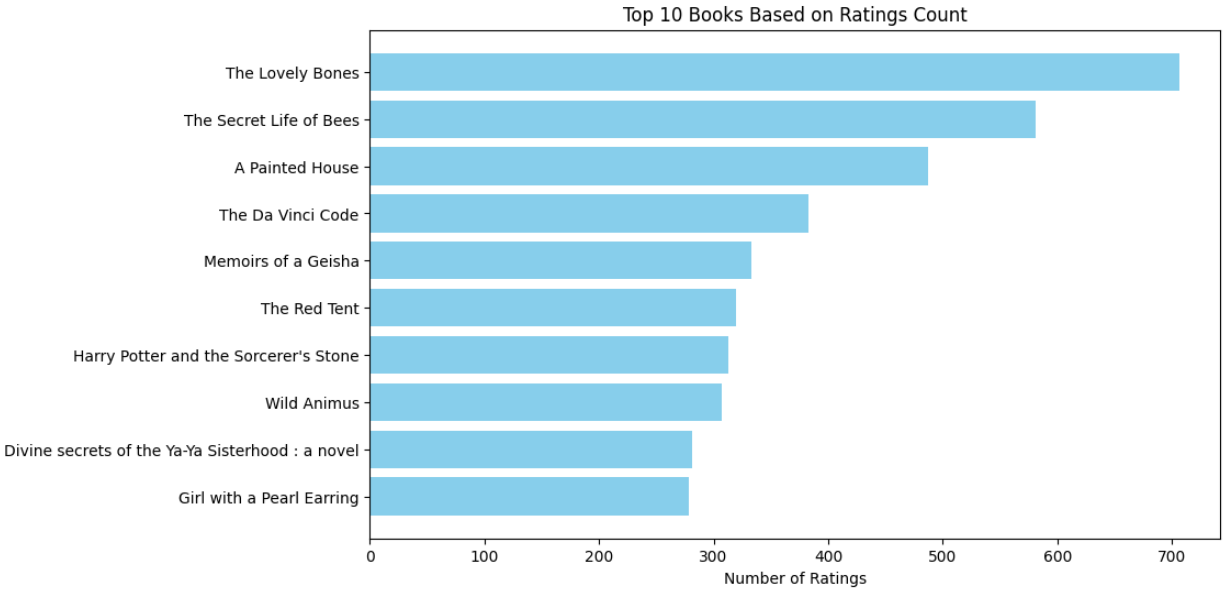
Correlations

Most readed books

In [ ]:
```
import matplotlib.pyplot as plt

# Assuming complete_df is your DataFrame
# Replace 'complete_df' with the actual name of your DataFrame if it's different

# Count the ratings for each book (ISBN)
book_ratings_count = complete_df['ISBN'].value_counts().head(10)

# Extracting data for plotting
top_10_books_count = book_ratings_count.tolist()
top_10_isbns = book_ratings_count.index
top_10_book_titles = list(set(complete_df[complete_df["ISBN"].isin(top_10_isbns)]["tit
top_10_books = dict(zip(top_10_book_titles, top_10_books_count))
```

```
# Identify the most read book
most_read_book_title = max(top_10_books, key=top_10_books.get)

# Plotting
plt.figure(figsize=(10, 6))
plt.barh(list(top_10_books.keys()), list(top_10_books.values()), color='skyblue')
plt.xlabel('Number of Ratings')
plt.title('Top 10 Books Based on Ratings Count')
plt.gca().invert_yaxis()  # To display the highest count at the top
plt.show()

print(f"The most rated book in the dataset is '{most_read_book_title}' with {top_10_bc
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```




```
The most rated book in the dataset is 'The Lovely Bones' with 707 ratings.
```

In [ ]:
```
mean_ratings_count = complete_df['ISBN'].value_counts().mean()

# Display the mean count of ratings
print("Mean Ratings Count for All Books:", mean_ratings_count)
```

```
Mean Ratings Count for All Books: 16.682038834951456
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

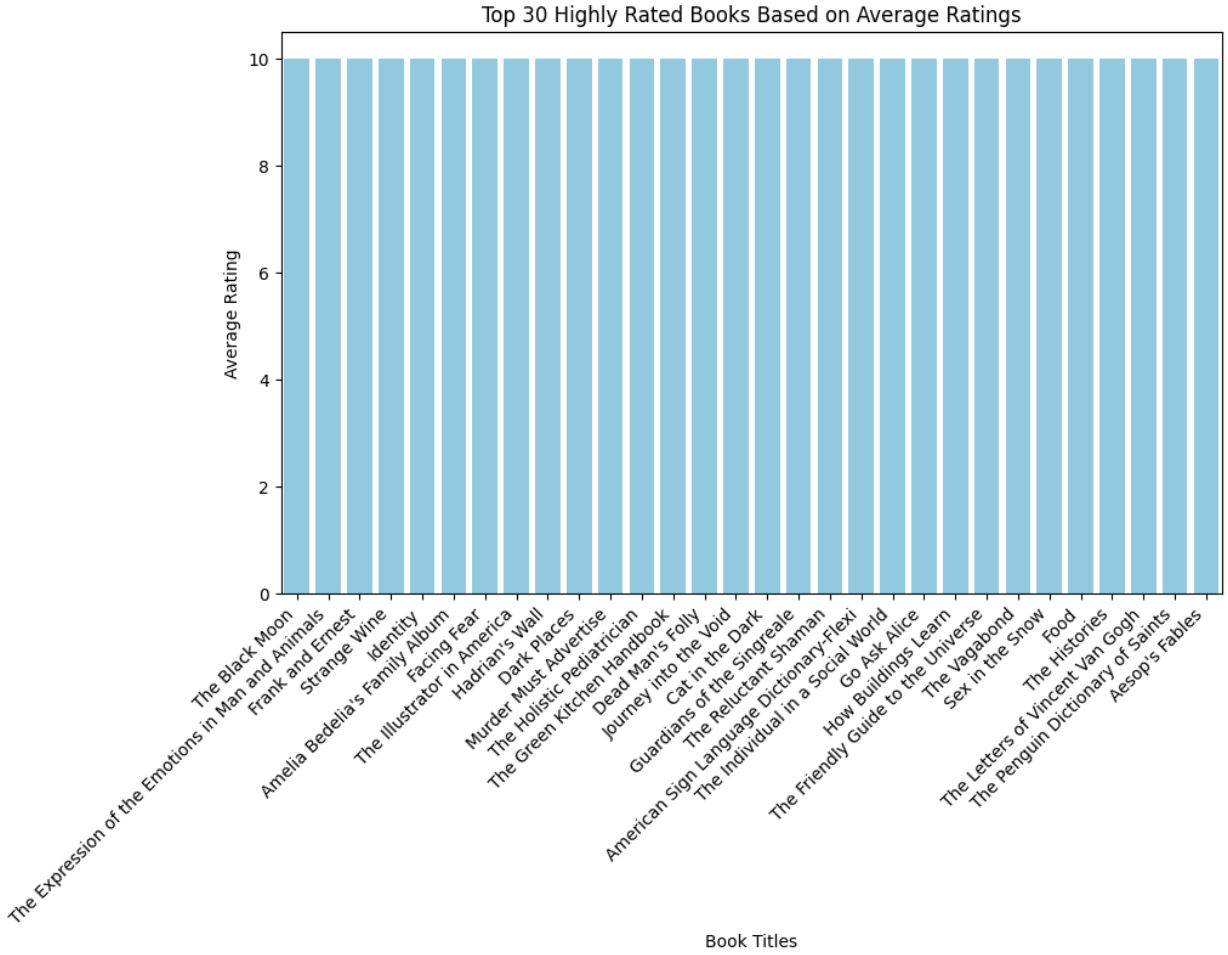
Correlation between Book-Rating and the significant attributes

Highly rated books by title

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the average rating for each book
```

```
average_ratings = complete_df.groupby('ISBN')['Book-Rating'].mean()

# Create a DataFrame with ISBN, title, and average rating
books_ratings_df = pd.DataFrame({
    'ISBN': average_ratings.index,
    'title': complete_df.groupby('ISBN')['title'].first(),
    'Average Rating': average_ratings.values
})

# Select the top 10 highly rated books
top_10_highly_rated_books = books_ratings_df.nlargest(30, 'Average Rating')

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(data=top_10_highly_rated_books, x='title', y='Average Rating', color='skyb
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better visibility
plt.xlabel('Book Titles')
plt.ylabel('Average Rating')
plt.title('Top 30 Highly Rated Books Based on Average Ratings')
plt.show()
```

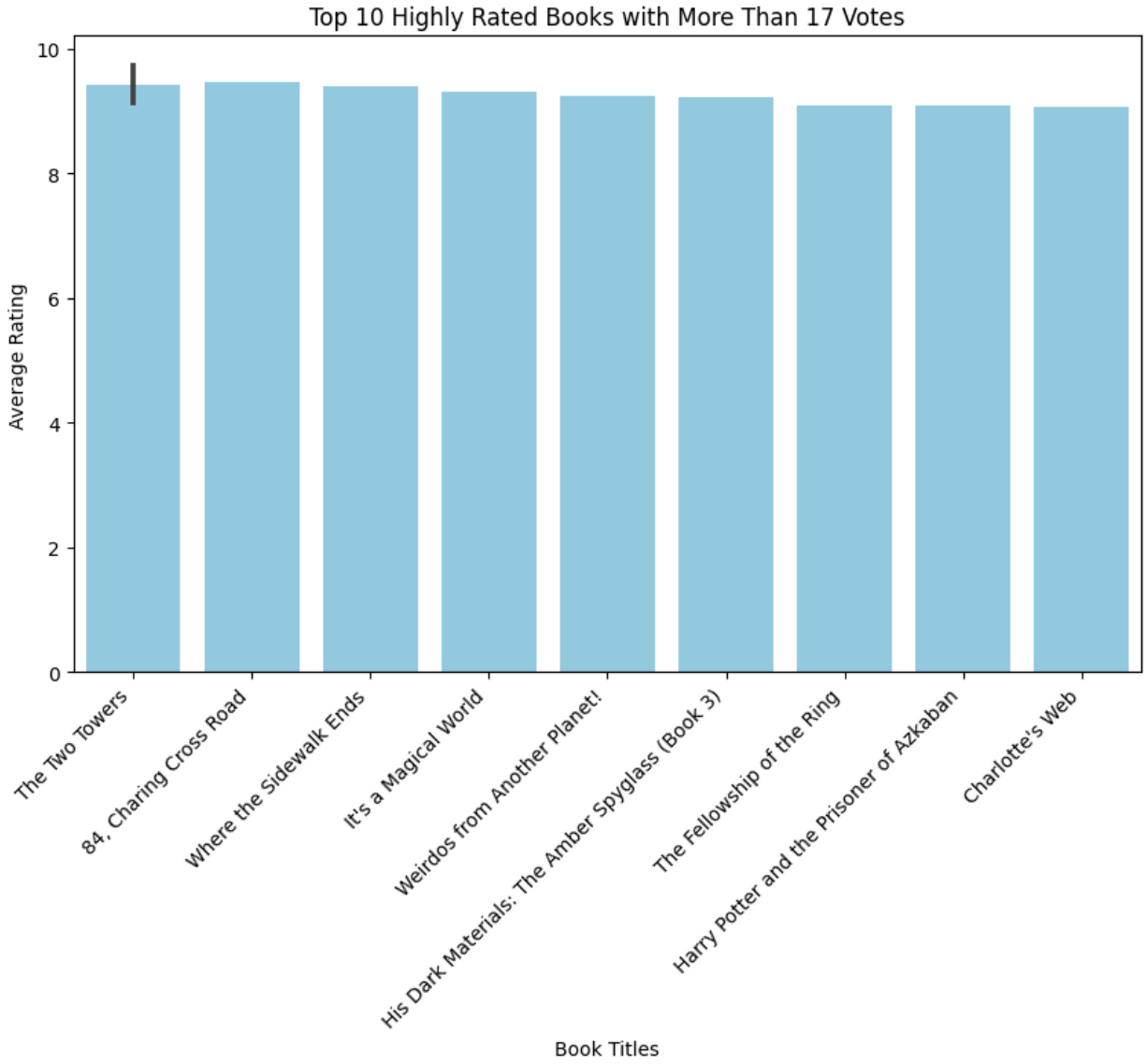
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



There are a lot of books with only one rating and a 10, those will not be considered. Books with more than 17 votes(the mean)

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the count of votes for each book
votes_count = complete_df['ISBN'].value_counts()

# Filter books that have more than 12 votes
popular_books_isbn = votes_count[votes_count > 17].index
popular_books_df = complete_df[complete_df['ISBN'].isin(popular_books_isbn)]

# Calculate the average rating for each popular book
average_ratings = popular_books_df.groupby('ISBN')['Book-Rating'].mean()

# Create a DataFrame with ISBN, title, and average rating
popular_books_ratings_df = pd.DataFrame({
    'ISBN': average_ratings.index,
    'title': popular_books_df.groupby('ISBN')['title'].first(),
    'Average Rating': average_ratings.values
})

# Identify the most highly rated book
most_highly_rated_book = popular_books_ratings_df.loc[popular_books_ratings_df['Averag

# Select the top 10 highly rated books
top_10_highly_rated_books = popular_books_ratings_df.nlargest(10, 'Average Rating')

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(data=top_10_highly_rated_books, x='title', y='Average Rating', color='skyb
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better visibility
plt.xlabel('Book Titles')
plt.ylabel('Average Rating')
plt.title('Top 10 Highly Rated Books with More Than 17 Votes')
plt.show()

print(f"The most highly rated book in the dataset is '{most_highly_rated_book['title']
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



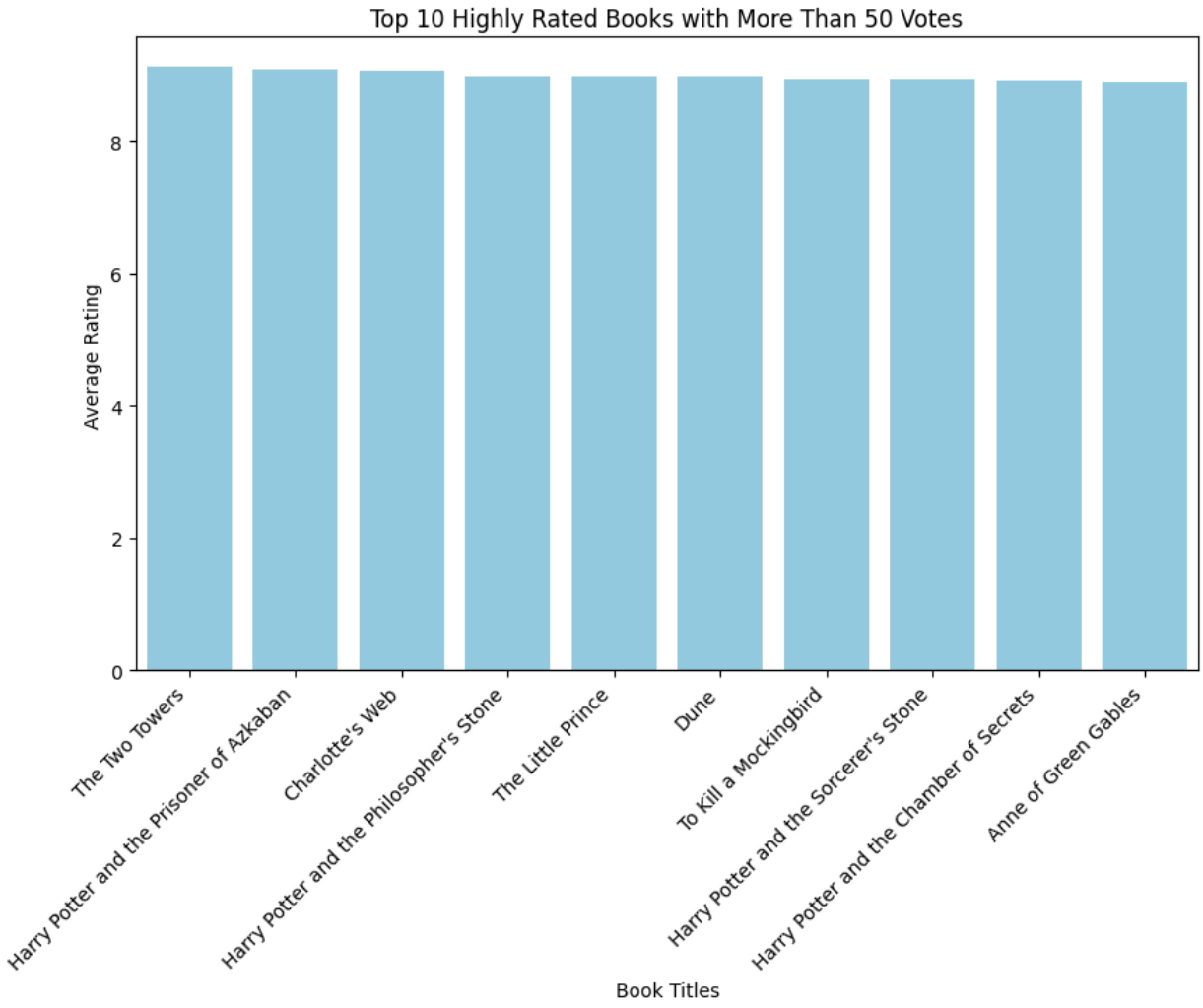
The most highly rated book in the dataset is 'The Two Towers' with an average rating of 9.72.

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the count of votes for each book
votes_count = complete_df['ISBN'].value_counts()

# Filter books that have more than 10 votes
popular_books_isbn = votes_count[votes_count > 50].index
popular_books_df = complete_df[complete_df['ISBN'].isin(popular_books_isbn)]

# Calculate the average rating for each popular book
average_ratings = popular_books_df.groupby('ISBN')['Book-Rating'].mean()

# Create a DataFrame with ISBN, title, and average rating
popular_books_ratings_df = pd.DataFrame({
    'ISBN': average_ratings.index,
    'title': popular_books_df.groupby('ISBN')['title'].first(),
    'Average Rating': average_ratings.values
})

# Select the top 10 highly rated books
```

```
top_10_highly_rated_books = popular_books_ratings_df.nlargest(10, 'Average Rating')

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(data=top_10_highly_rated_books, x='title', y='Average Rating', color='skyb
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better visibility
plt.xlabel('Book Titles')
plt.ylabel('Average Rating')
plt.title('Top 10 Highly Rated Books with More Than 50 Votes')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



Highly rated authors

In [ ]:

```
authors_ratings_df = complete_df.groupby('authors')['Book-Rating'].agg(['mean', 'count

# Find the three most highly rated authors
top_10_highly_rated_authors = authors_ratings_df.nlargest(10, 'mean')

# Print the three most highly rated authors
print("Top 10 Most Highly Rated Authors:")
print(top_10_highly_rated_authors[['authors', 'mean', 'count']])
```

```
Top 10 Most Highly Rated Authors:
                    authors  mean  count
4            A. E. van Vogt  10.0      1
6             A. K. Dewdney  10.0      1
7          A. N. Roquelaure  10.0      1
15        Adrian J. Desmond  10.0      1
17  Aesop, Jack David Zipes  10.0      1
20           Ahmed Souaiaia  10.0      1
22          Aileen Humphrey  10.0      1
25     Alan Alexander Milne  10.0      2
48            Alexandra Dirk  10.0      1
64       Alister E. McGrath  10.0      1
```

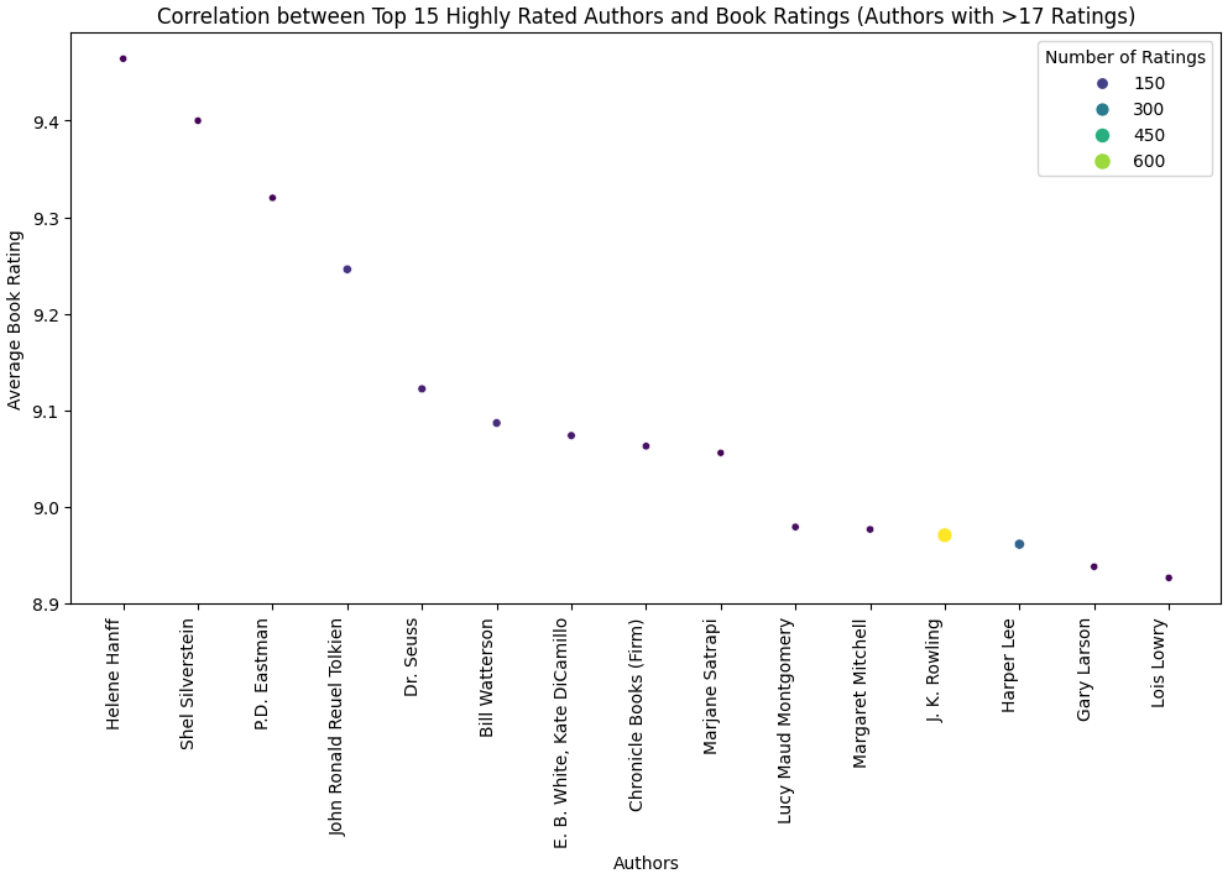
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

We will disregard authors who have only one rated book, even if it has the highest rating, as such cases may not be representative.

limiting to authors that have more than 17 ratings(the mean)

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the count of ratings for each author
authors_ratings_count = complete_df['authors'].value_counts()

# Filter authors with more than 17 ratings
popular_authors = authors_ratings_count[authors_ratings_count > 17].index
popular_authors_df = complete_df[complete_df['authors'].isin(popular_authors)]

# Create a DataFrame with 'authors', 'Book-Rating', and count of ratings for popular a
popular_authors_ratings_df = popular_authors_df.groupby('authors')['Book-Rating'].agg(

# Identify the top 10 most highly rated authors
top_10_authors = popular_authors_ratings_df.nlargest(15, 'mean')

# Plotting
plt.figure(figsize=(12, 6))

# Scatter plot for the correlation between 'authors' and 'Book-Rating' for top 10 auth
sns.scatterplot(data=top_10_authors, x='authors', y='mean', size='count', hue='count',
plt.xticks(rotation=90, ha='right')  # Rotate x-axis labels for better visibility
plt.xlabel('Authors')
plt.ylabel('Average Book Rating')
plt.title('Correlation between Top 15 Highly Rated Authors and Book Ratings (Authors w
plt.legend(title='Number of Ratings', loc='upper right')
plt.show()

print(f"The most highly rated author in the top 15 is '{top_10_authors.iloc[0]['author
```

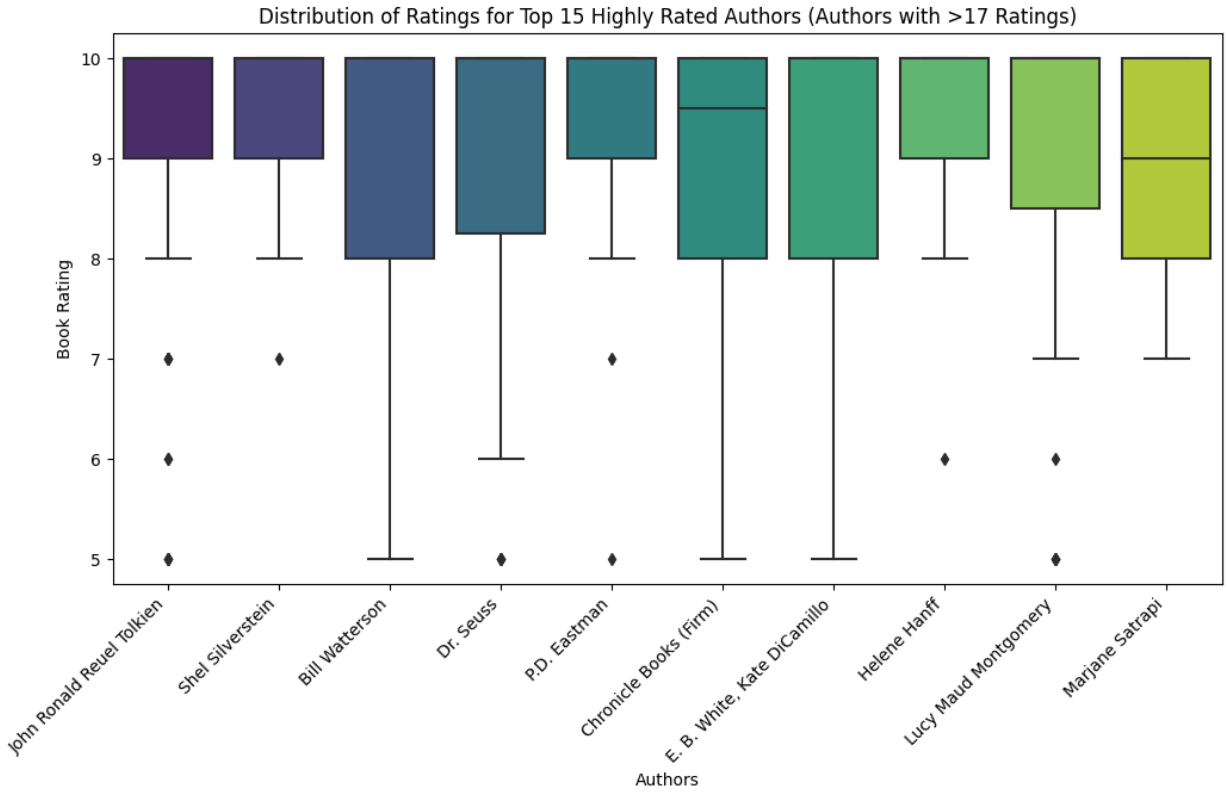
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```




```
The most highly rated author in the top 15 is 'Helene Hanff' with an average rating o
f 9.46.
```

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the count of ratings for each author
authors_ratings_count = complete_df['authors'].value_counts()

# Filter authors with more than 17 ratings
popular_authors = authors_ratings_count[authors_ratings_count > 17].index
popular_authors_df = complete_df[complete_df['authors'].isin(popular_authors)]

# Create a DataFrame with 'authors', 'Book-Rating', and count of ratings for popular a
popular_authors_ratings_df = popular_authors_df.groupby('authors')['Book-Rating'].agg(

# Identify the top 10 most highly rated authors
top_10_authors = popular_authors_ratings_df.nlargest(10, 'mean')

# Plotting
plt.figure(figsize=(12, 6))

# Boxplot for the distribution of ratings for top 10 authors
sns.boxplot(data=popular_authors_df[popular_authors_df['authors'].isin(top_10_authors[
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better visibility
plt.xlabel('Authors')
plt.ylabel('Book Rating')
plt.title('Distribution of Ratings for Top 15 Highly Rated Authors (Authors with >17 R
plt.show()

print(f"The most highly rated author in the top 10 is '{top_10_authors.iloc[0]['author
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```




```
The most highly rated author in the top 10 is 'Helene Hanff' with an average rating o
f 9.46.
```

In [ ]:
```
top_5_highly_rated_authors = popular_authors_ratings_df.nlargest(5, 'mean')

# Print the three most highly rated authors
print("Top 5 Most Highly Rated Authors:")
print(top_5_highly_rated_authors[['authors', 'mean', 'count']])
```

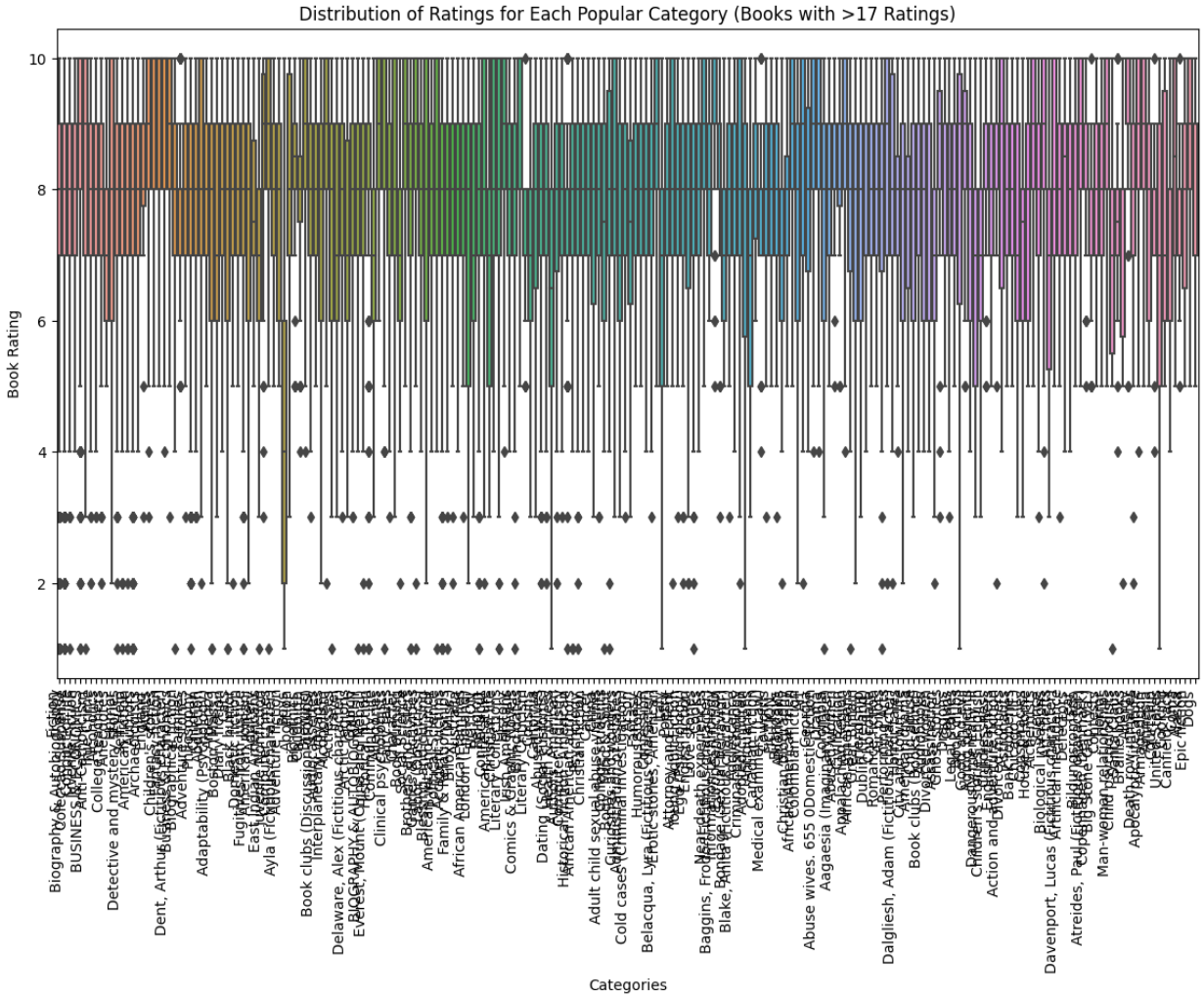
```
Top 5 Most Highly Rated Authors:
                      authors      mean  count
173               Helene Hanff  9.464286     28
425           Shel Silverstein  9.400000     25
355               P.D. Eastman  9.320000     25
238  John Ronald Reuel Tolkien  9.245902    122
122                  Dr. Seuss  9.121951     82
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Distribution of ratings for each popular category

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming complete_df is your DataFrame
```

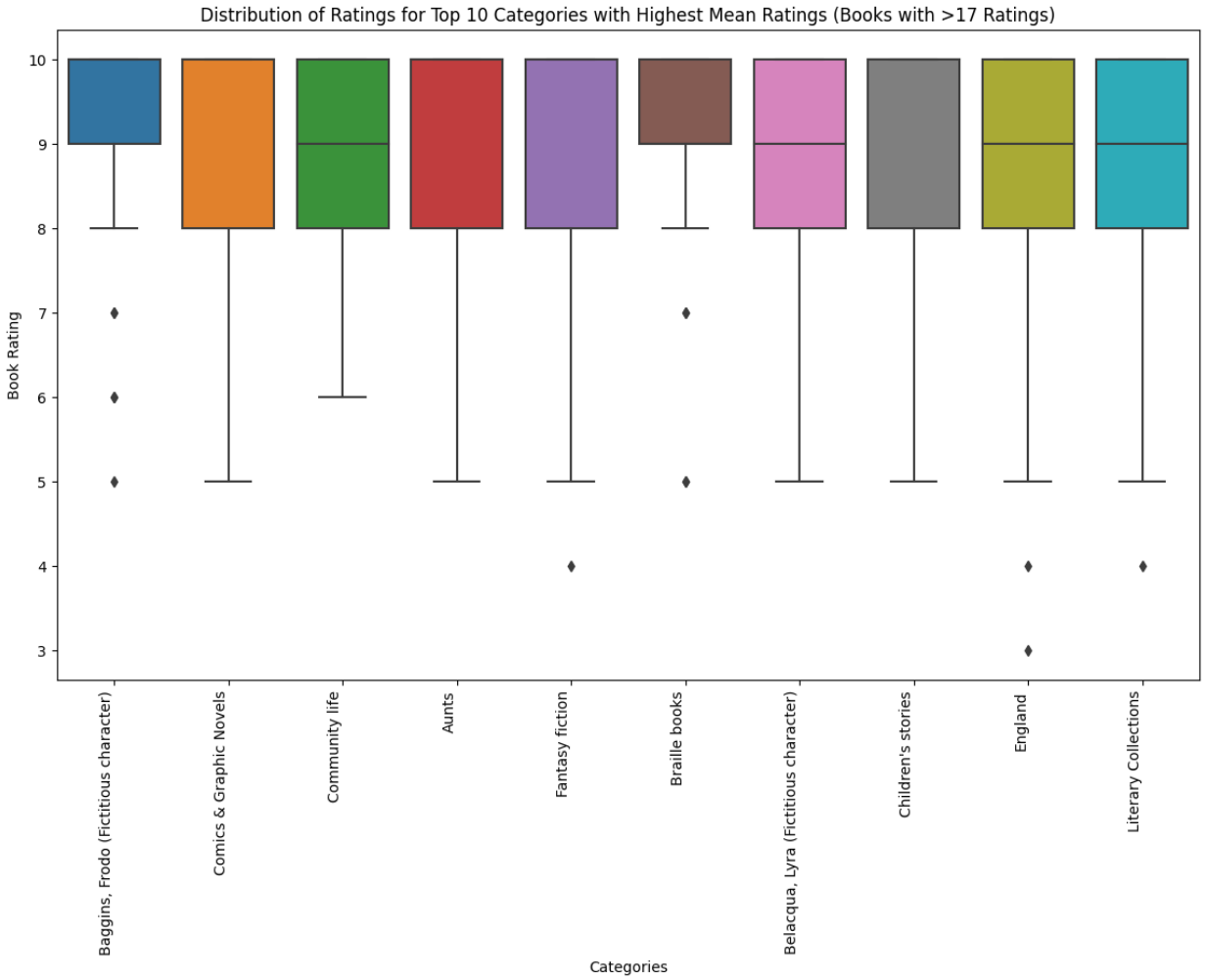
```
# Replace 'complete_df' with the actual name of your DataFrame if it's different

# Filter categories with more than 17 ratings
categories_count = complete_df['categories'].value_counts()
popular_categories = categories_count[categories_count > 17].index
filtered_df = complete_df[complete_df['categories'].isin(popular_categories)]

# Calculate mean ratings for each category
mean_ratings_by_category = filtered_df.groupby('categories')['Book-Rating'].mean()

# Identify the category with the highest mean rating
highest_mean_category = mean_ratings_by_category.idxmax()
highest_mean_value = mean_ratings_by_category.max()

print(f"The category with the highest mean rating is '{highest_mean_category}' with a

# Box plot to visualize the distribution of ratings for each category
plt.figure(figsize=(14, 8))
sns.boxplot(x='categories', y='Book-Rating', data=filtered_df)
plt.title('Distribution of Ratings for Each Popular Category (Books with >17 Ratings)'
plt.xlabel('Categories')
plt.ylabel('Book Rating')
plt.xticks(rotation=90, ha='right')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
The category with the highest mean rating is 'Baggins, Frodo (Fictitious character)'
with a mean rating of 9.29
```



In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming complete_df is your DataFrame
# Replace 'complete_df' with the actual name of your DataFrame if it's different

# Filter categories with more than 17 ratings
categories_count = complete_df['categories'].value_counts()
popular_categories = categories_count[categories_count > 17].index
filtered_df = complete_df[complete_df['categories'].isin(popular_categories)]

# Calculate mean ratings for each category
mean_ratings_by_category = filtered_df.groupby('categories')['Book-Rating'].mean()

# Select the top 10 categories with the highest mean rating
top_10_categories = mean_ratings_by_category.nlargest(10)

# Box plot to visualize the distribution of ratings for the top 10 categories
plt.figure(figsize=(14, 8))
sns.boxplot(x='categories', y='Book-Rating', data=filtered_df, order=top_10_categories
plt.title('Distribution of Ratings for Top 10 Categories with Highest Mean Ratings (Bo
plt.xlabel('Categories')
plt.ylabel('Book Rating')
plt.xticks(rotation=90, ha='right')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



Association Rules

Frequent Itemset by Hand with support threshold 0.002. Iteratively finding frequent itemsets, which are sets of items that frequently occur together in the dataset. The algorithm uses a support threshold. It works with principle, which states that if an itemset is frequent, then all of its subsets must also be frequent. This property is essential for efficiently pruning the search space of candidate itemsets.

In [ ]:
```
# Filter and remove duplicates to get liked items (ratings > 6) for each user

df_liked = df[df['Book-Rating'] > 6]
df_liked = df_liked.drop_duplicates(['User-ID', 'ISBN'])
df_liked['User-ID'] = df_liked['User-ID'].astype(str)
df_liked['ISBN'] = df_liked['ISBN'].astype(str)
# Count the number of liked items per user and create baskets of items for each user
baskets = df_liked.groupby('User-ID')['ISBN'].apply(list).tolist()
items = set(list(np.concatenate(baskets).flat))
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```python
# Create a DataFrame to hash items to integers
df_item_hash = pd.DataFrame(range(len(items)), index=list(items), columns=['hashcode']

# Count the occurrences of each item and store in an array
item_count_arr = np.zeros((len(items), 1))

for basket in baskets:
    for item in basket:
        idx = df_item_hash.loc[item, 'hashcode']
        item_count_arr[idx] += 1
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```python
# Find frequent items with support > 0.002 (s1 = 0.002)
freq_items = [df_item_hash[df_item_hash['hashcode'] == x].index[0] for x in np.where(i


df_freq_item_hash = pd.DataFrame(range(1, len(freq_items) + 1), index=freq_items, colu

# Count pairs using only frequent items and store in a matrix (if the itemset is infre
pair_mat_hashed = np.zeros((len(freq_items) + 1, len(freq_items) + 1))

for basket in baskets:
    cand_list = [item for item in basket if item in freq_items]
    if len(cand_list) < 2:
        continue
    for idx, item1 in enumerate(cand_list):
        for item2 in cand_list[idx + 1:]:
            i = df_freq_item_hash.loc[item1, 'hashcode']
            j = df_freq_item_hash.loc[item2, 'hashcode']
            pair_mat_hashed[max(i, j), min(i, j)] += 1


freq_pairs = [
    [df_freq_item_hash[df_freq_item_hash['hashcode'] == x].index[0],
     df_freq_item_hash[df_freq_item_hash['hashcode'] == y].index[0]]
    for x, y in zip(*np.where(pair_mat_hashed > 0.002 * len(baskets)))]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```python
print(freq_pairs)
```

```
[['0590353403', '0439064864'], ['0439136350', '0439064864'], ['0439136350', '05903534
03'], ['0439139597', '0439064864'], ['0439139597', '0590353403'], ['0439139597', '043
9136350'], ['0385504209', '0671027360'], ['059035342X', '0439136369'], ['043935806X',
'0439064864'], ['043935806X', '0439136350'], ['043935806X', '0439139597']]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Association Rules Algorithms

1. A-priori Support threshold set to 0.003 and using predifined libraries to obtain frequent items together list, investigating its size, how many appears of items were based on transactions length and support values and investigating the association rules (later on checked in excel for better interpretation)

In [ ]:
```
# SUPPORT 0.003
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
# Convert data to a transactions

df_subset = df_liked
transactions = df_subset.groupby('User-ID')['ISBN'].apply(list).tolist()

# Use TransactionEncoder to convert the data into a binary matrix
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
binary_df = pd.DataFrame(te_ary, columns=te.columns_)
# frequent itemsets
frequent_itemsets = apriori(binary_df, min_support=0.003, use_colnames=True)
print("Frequent Itemsets (with more than one element):")
print(frequent_itemsets)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
Frequent Itemsets (with more than one element):
      support                  itemsets
0    0.003189              (0060175400)
1    0.003094              (0060391626)
2    0.005950              (0060392452)
3    0.006283              (0060502258)
4    0.012327              (0060928336)
..        ...                       ...
154  0.003665  (0439064864, 0439136350)
155  0.003141  (0439064864, 0439139597)
156  0.003332  (0439064864, 0590353403)
157  0.003427  (0439139597, 0439136350)
158  0.003046  (0439136350, 0590353403)

[159 rows x 2 columns]
```

In [ ]:
```
#min support * transactions
len(transactions)*0.003
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
63.03
```

In [ ]:
```
# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

# Display frequent itemsets and association rules
#print("Frequent Itemsets:")
#print(frequent_itemsets)
print("\nAssociation Rules:")
len(rules)
```

```
Association Rules:
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
9
```

In [ ]:
```
rules=pd.DataFrame(rules)
rules.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	c
0	(0439064864)	(0439136350)	0.005664	0.006188	0.003665	0.647059	104.574661	0.003630	
1	(0439136350)	(0439064864)	0.006188	0.005664	0.003665	0.592308	104.574661	0.003630	
2	(0439064864)	(0439139597)	0.005664	0.006283	0.003141	0.554622	88.277311	0.003106	
3	(0439139597)	(0439064864)	0.006283	0.005664	0.003141	0.500000	88.277311	0.003106	
4	(0439064864)	(0590353403)	0.005664	0.005236	0.003332	0.588235	112.352941	0.003302	

1. A-priori support threshold set to 0.001

In [ ]:
```
# SUPPORT 0.002
# Convert data to transactions

df_subset = df_liked
transactions = df_subset.groupby('User-ID')['ISBN'].apply(list).tolist()

# Use TransactionEncoder to convert the data into a binary matrix
te = TransactionEncoder()
```

```
te_ary = te.fit(transactions).transform(transactions)
binary_df = pd.DataFrame(te_ary, columns=te.columns_)
# Use Apriori to find frequent itemsets
frequent_itemsets = apriori(binary_df, min_support=0.002, use_colnames=True)
print("Frequent Itemsets (with more than one element):")
print(frequent_itemsets)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
Frequent Itemsets (with more than one element):
      support                                          itemsets
0    0.002856                                     (002542730X)
1    0.003189                                     (0060175400)
2    0.003094                                     (0060391626)
3    0.005950                                     (0060392452)
4    0.006283                                     (0060502258)
..        ...                                              ...
294  0.002713             (0439064864, 0439139597, 0439136350)
295  0.002808             (0439064864, 0439136350, 0590353403)
296  0.002237             (0439064864, 0439139597, 0590353403)
297  0.002189             (0439139597, 0439136350, 0590353403)
298  0.002094  (0439064864, 0439139597, 0439136350, 0590353403)

[299 rows x 2 columns]
```

In [ ]:
```
#min support * transactions
len(transactions)*0.002
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
42.02
```

In [ ]:
```
# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
print("\nAssociation Rules:")
len(rules)
```

```
Association Rules:
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
32
```

1. FP-Growth Algorithm with support threshold set to 0.0005 due to scalability issues of A-prori algorithm. FP-Growth uses tree based approach, which makes it more efficient for mining frequent itemsets

In [ ]:
```
# Convert data to a list of lists (transactions)
df_subset = df_liked
transactions = df_subset.groupby('User-ID')['ISBN'].apply(list).tolist()


# Use TransactionEncoder to convert the data into a binary matrix
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
binary_df = pd.DataFrame(te_ary, columns=te.columns_)

# Use FP-growth to find frequent itemsets
frequent_itemsets = fpgrowth(binary_df, min_support=0.0005, use_colnames=True)
print("Frequent Itemsets:")
print(frequent_itemsets)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
Frequent Itemsets:
       support                 itemsets
0     0.001475              (0345397819)
1     0.014089              (059035342X)
2     0.005569              (0345339703)
3     0.020419              (0385504209)
4     0.005950              (0060392452)
...        ...                       ...
1623  0.000524  (0553289411, 0553280945)
1624  0.001142  (0515136530, 0515136379)
1625  0.000762  (0552128481, 0552124753)
1626  0.000571  (0552128481, 0552131067)
1627  0.000571  (0552128481, 0552131059)

[1628 rows x 2 columns]
```

In [ ]:
```
# Generate association rules
rules_0005 = association_rules(frequent_itemsets, metric="confidence", min_threshold=0

# Display frequent itemsets and association rules
#print("Frequent Itemsets:")
#print(frequent_itemsets)
print("\nAssociation Rules:")
print(len(rules_0005))
```

```
Association Rules:
319
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
rules_0005=pd.DataFrame(rules_0005)
rules_0005.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	c
0	(0439139597)	(0439136350)	0.006283	0.006188	0.003427	0.545455	88.153846	0.003388	
1	(0439136350)	(0439139597)	0.006188	0.006283	0.003427	0.553846	88.153846	0.003388	
2	(0439139597, 043935806X)	(0439136350)	0.002475	0.006188	0.001856	0.750000	121.211538	0.001841	
3	(043935806X, 0439136350)	(0439139597)	0.002427	0.006283	0.001856	0.764706	121.715686	0.001841	
4	(0439139597, 0439136350)	(043935806X)	0.003427	0.009329	0.001856	0.541667	58.063350	0.001824	

Overall findings:

- Not enough rules found using the Apriori
- Minimizing support provides less reliable recommendation system, where still not all items will be included
- Confidence threshold set to 0.5 in all association rules. Confidence indicates support of union of antecedent and consequent/antecedent support. The higher the confidence, the stronger association between the items

SUMMARY: Association rules seem not to be a very effective method for recommendation system. They cut many items. Additionaly, it seems like not many people read and liked similar books, which create problematic with associatio rules fidings

Content Based Recommendation System

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from scipy.spatial.distance import pdist

import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline, make_union
import string
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import cosine_similarity

import spacy
```

```
import re
import nltk
from nltk.stem import PorterStemmer
from wordcloud import WordCloud
import seaborn as sns
sns.set_style('whitegrid')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[ ]: True

Download the csv previously created

In [ ]:
```
# Loading book information
df_books = pd.read_csv('https://raw.githubusercontent.com/anbipa/BookRecommenderCT/mai

# Loading reviews information
df_reviews = pd.read_csv('https://raw.githubusercontent.com/anbipa/BookRecommenderCT/m
```

In [ ]:
```
df_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6273 entries, 0 to 6272
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   isbn           6273 non-null   object
 1   title          6273 non-null   object
 2   subtitle       2005 non-null   object
 3   authors        6216 non-null   object
 4   publisher      5348 non-null   object
 5   publishedDate  6268 non-null   object
 6   description    6273 non-null   object
 7   pageCount      6273 non-null   int64 
 8   categories     6157 non-null   object
 9   language       6273 non-null   object
dtypes: int64(1), object(9)
memory usage: 490.2+ KB
```

In [ ]:
```
df_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66082 entries, 0 to 66081
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Unnamed: 0   66082 non-null  int64 
 1   User-ID      66082 non-null  int64 
 2   ISBN         66082 non-null  object
 3   Book-Rating  66082 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 2.0+ MB
```

In this section, we define several functions for text cleaning and preprocessing, that will help us remove unnecessary elements from our text data, such as HTML tags, stopwords, digits, punctuation, and they will also perform stemming. After defining these functions, we combine

relevant text fields from the `df_books` dataset into a new `CombinedText` column and apply our text processing functions to this column.

In [ ]:
```
def strip_html(text):
    clean = re.compile('<.*?>')
    return re.sub(clean, '', text)

def remove_stopwords(text):
    text = text.split()
    text = [word for word in text if word not in stopwords]
    return ' '.join(text)

def remove_digits(text):
    text = re.sub(r'[0-9]', '', text)
    return text

def remove_punctuation(text):
    text = ''.join([word for word in text if word not in punctuation])
    return text

def stem_text(text):
    stemmer = PorterStemmer()
    text = ' '.join([stemmer.stem(word) for word in text.split()])
    return text

def parse_text(text):
    text = text.lower()
    text = strip_html(text)
    text = remove_stopwords(text)
    text = remove_digits(text)
    text = remove_punctuation(text)
    text = stem_text(text)
    return text
```

In [ ]:
```
df = df_books.copy()
```

In [ ]:
```
df['CombinedText'] = (
    df['title'].astype(str).fillna('') + ' ' +
    df['subtitle'].astype(str).fillna('') + ' ' +
    df['authors'].astype(str).fillna('') + ' ' +
    df['description'].astype(str).fillna('') + ' ' +
    df['categories'].astype(str).fillna('')
)

stopwords = set(stopwords.words('english'))
punctuation = string.punctuation + '—'

df['CombinedText'] = df['CombinedText'].apply(parse_text)
df.head()
```

Out[ ]:

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCount	c
0	0884115631	The Shepherd	NaN	Frederick Forsyth	NaN	1960-06-01	A pilot and his plane are saved by a mysteriou...	0	
1	0060175648	Identity	A Novel	Milan Kundera	Harper	1998-04-21	Milan Kundera's Identity translated from the F...	176	
2	0394535383	Pillar of the Sky	A Novel	Cecelia Holland	Alfred a Knopf Incorporated	1985	Two thousand years ago, at the site of Stonehe...	534	
3	0843114401	Jingle Bear	NaN	Stephen Cosgrove	NaN	1985	When the other bears get ready to hibernate fo...	36	
4	0451207408	The Adventurer	NaN	Jaclyn Reding	Signet Book	2002	While attempting to return a mysterious, legen...	314	

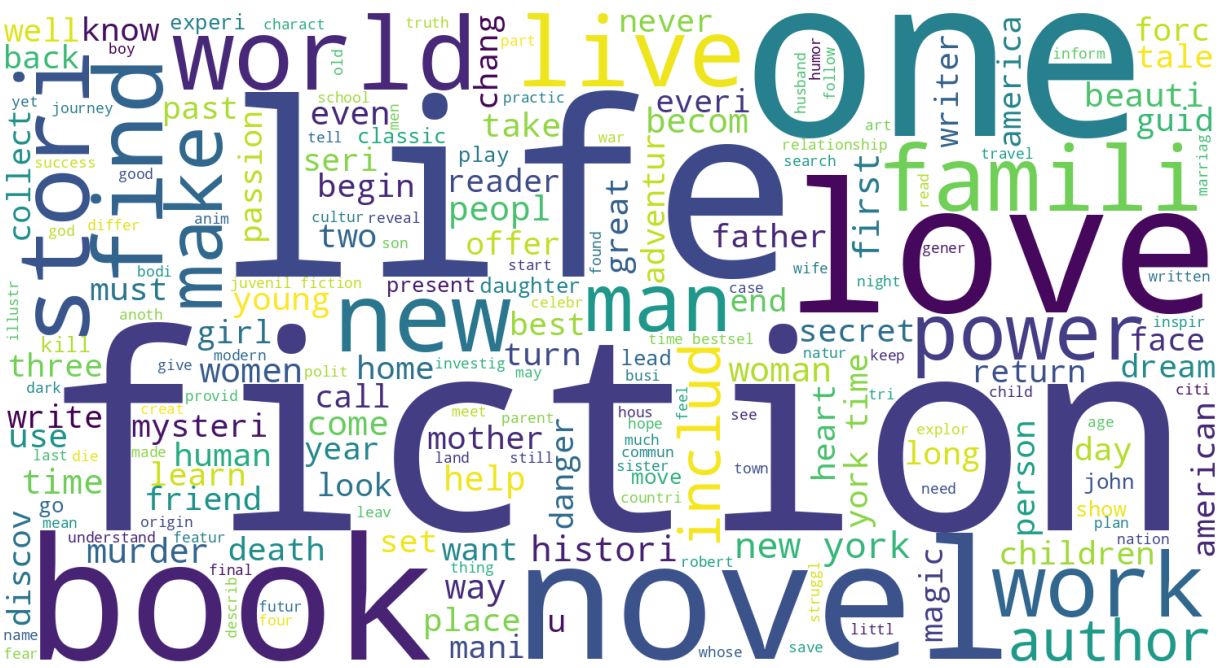
We remove the "nan" words from the combined text

In [ ]:
```
df['CombinedText'] = df['CombinedText'].replace('nan', '', regex=True)
```

To visually explore the most frequent words in our combined book data, we generate a word cloud, this visualization will help us understand the predominant themes or topics in our dataset.

In [ ]:
```
def create_wordcloud(data):
    wordcloud = WordCloud(
        width=1500,
        height=800,
        min_font_size=12,
        background_color='white'
    ).generate(data)
    plt.figure(figsize=(15,8))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.tight_layout(pad=0)
    plt.show()
```

In [ ]:
```
all_corpus = ' '.join(df['CombinedText'])
create_wordcloud(all_corpus)
```



Vectorize text

We transform our cleaned and combined text data into a numerical format using TF-IDF vectorization, to feed the data to the clustering and recommendation algorithms as it converts text into a form that can be processed by machine learning algorithms.

Ignore terms that appear in more than 50% of the documents (as set by max_df=0.5) and terms that are not present in at least 5 documents (set by min_df=5)

In [ ]:
```
vectorizer = TfidfVectorizer(max_df=0.5,min_df=5)
tfidf = vectorizer.fit_transform(df['CombinedText'])
tfidf
```

Out[ ]:
```
<6273x6762 sparse matrix of type '<class 'numpy.float64'>'
        with 235708 stored elements in Compressed Sparse Row format>
```

Compute spasity of the matrix as the fraction of non-zero entries divided by the total number of elements

In [ ]:
```
print(f"{tfidf.nnz / np.prod(tfidf.shape):.3f}")
```

```
0.006
```

Hierarchical Clustering

first approach. We use pdist to compute the cosine distance between the documents.

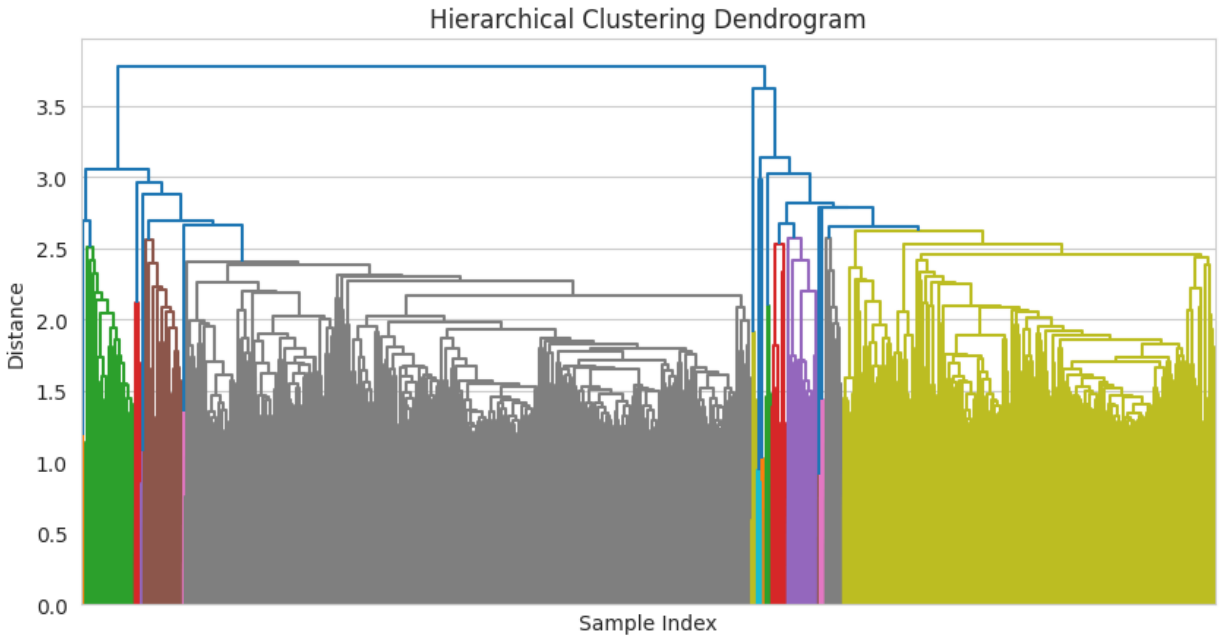
In [ ]:
```
X = tfidf.todense()
dist_array = pdist(X, 'cosine')
```

In [ ]:
```
from scipy.spatial.distance import squareform

len(dist_array)
distance_matrix = squareform(dist_array)
```

Perform hierarchical clustering

First we perform the dendrogram in order to identify the different clusters inside our data

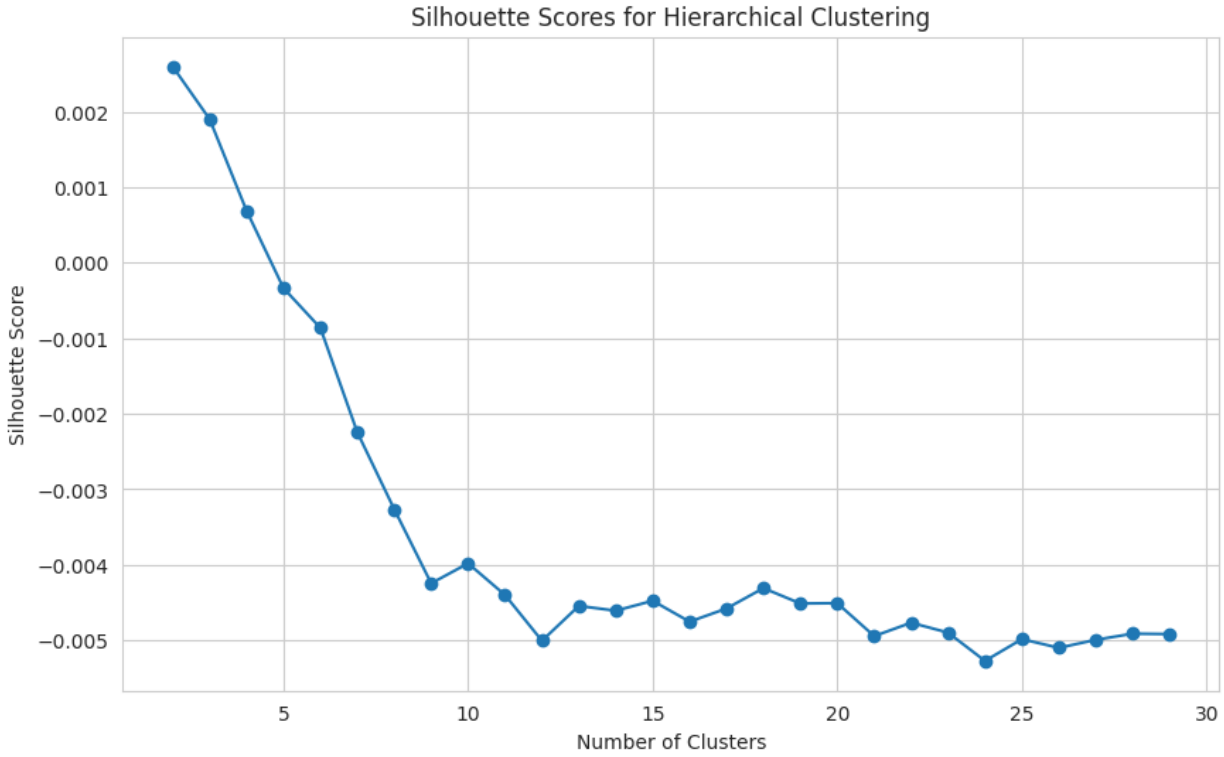
In [ ]:
```
from scipy.cluster.hierarchy import linkage, dendrogram

# Perform hierarchical clustering
linkage_matrix = linkage(dist_array, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix, no_labels=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



We also performed the silhoeutte score from a range from 2 to 30 in order to identify teh quality of the clsuters.

In [ ]:
```
from sklearn.metrics import silhouette_score

cluster_numbers = range(2, 30)
silhouette_scores_hierarchical = []

for k in cluster_numbers:
    # Extract cluster labels
    cluster_labels = fcluster(linkage_matrix, k, criterion='maxclust')
```

```
    # Calculate silhouette score using the X_lsa
    silhouette_avg = silhouette_score(distance_matrix, cluster_labels, metric='precomp
    silhouette_scores_hierarchical.append(silhouette_avg)

# Plot the silhouette scores
plt.figure(figsize=(10, 6))
plt.plot(cluster_numbers, silhouette_scores_hierarchical, marker='o')
plt.title('Silhouette Scores for Hierarchical Clustering')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



We select k based on the dendogram visualization and the silhoeutte score

In [ ]:
```
clusters = fcluster(linkage_matrix, t=12, criterion='maxclust')

# Append cluster labels to the dataframe
df['cluster_h'] = clusters
```

In [ ]:
```
# Print the distribution of clusters
cluster_distribution = df['cluster_h'].value_counts().sort_index()
print(cluster_distribution)
```

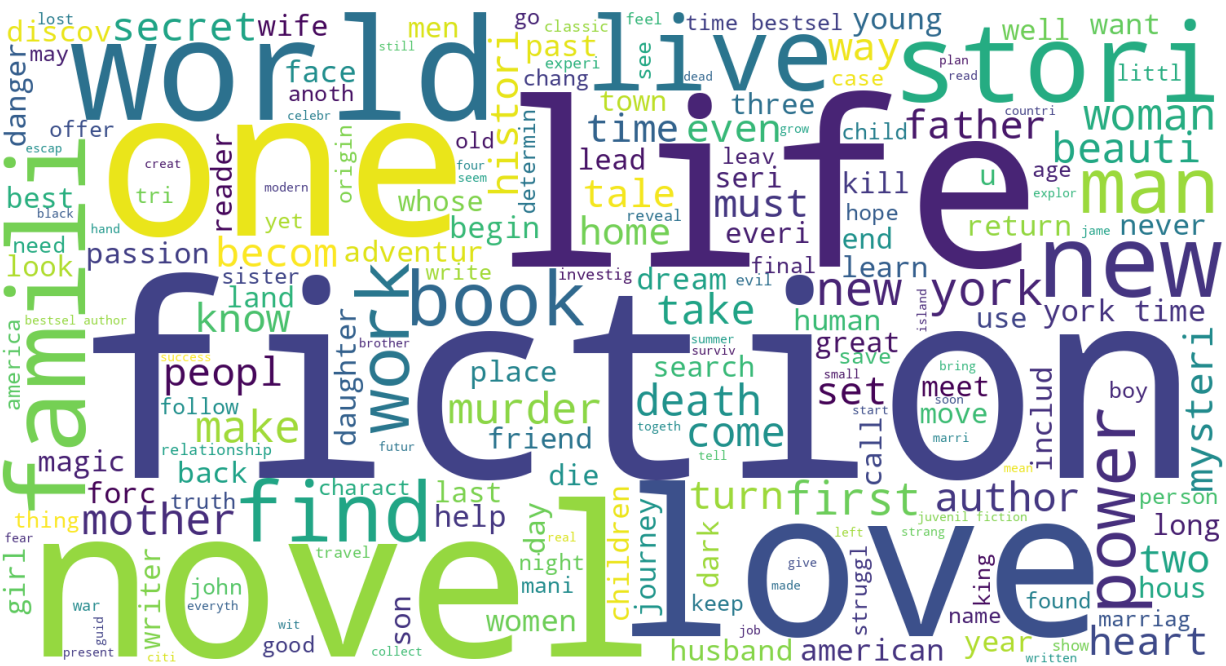
```
1       293
2        39
3        14
4      3362
5        32
6        17
7        29
8        32
9       258
10       11
11       28
12     2158
Name: cluster_h, dtype: int64
```

Print wordclouds

Upon examining the word clouds generated for each cluster, it becomes evident that certain clusters distinctly encapsulate specific genres or topics, this highlight the effectiveness of the clustering algorithm in discerning and grouping books based on meaningful thematic elements.

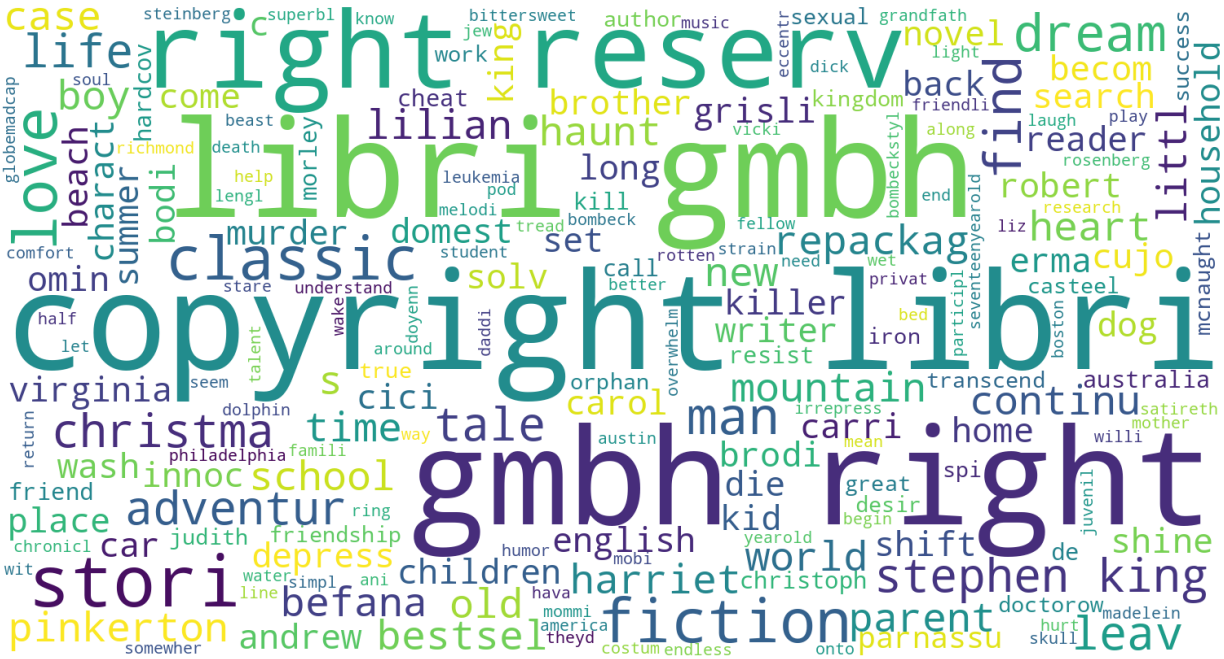
In [ ]:
```
unique_clusters = df['cluster_h'].unique()

# Create a dictionary to store DataFrames for each cluster
cluster_dataframes = {}

# Split the DataFrame by cluster label
for cluster_label in unique_clusters:
    cluster_dataframes[cluster_label] = df[df['cluster_h'] == cluster_label]
    joined_genre = ' '.join(cluster_dataframes[cluster_label]['CombinedText'])
    print("CLUSTER NUMBER: ", cluster_label)
    create_wordcloud(joined_genre)
```

```
CLUSTER NUMBER:  4
```

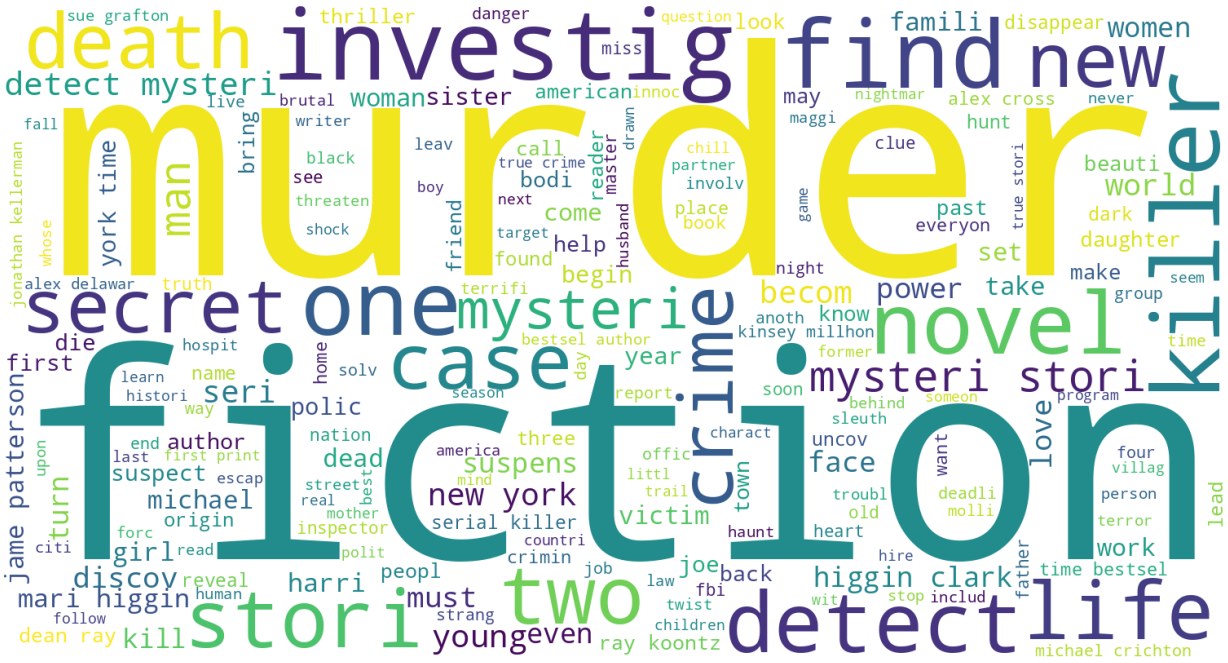



```
CLUSTER NUMBER:  12
```

CLUSTER NUMBER: 7

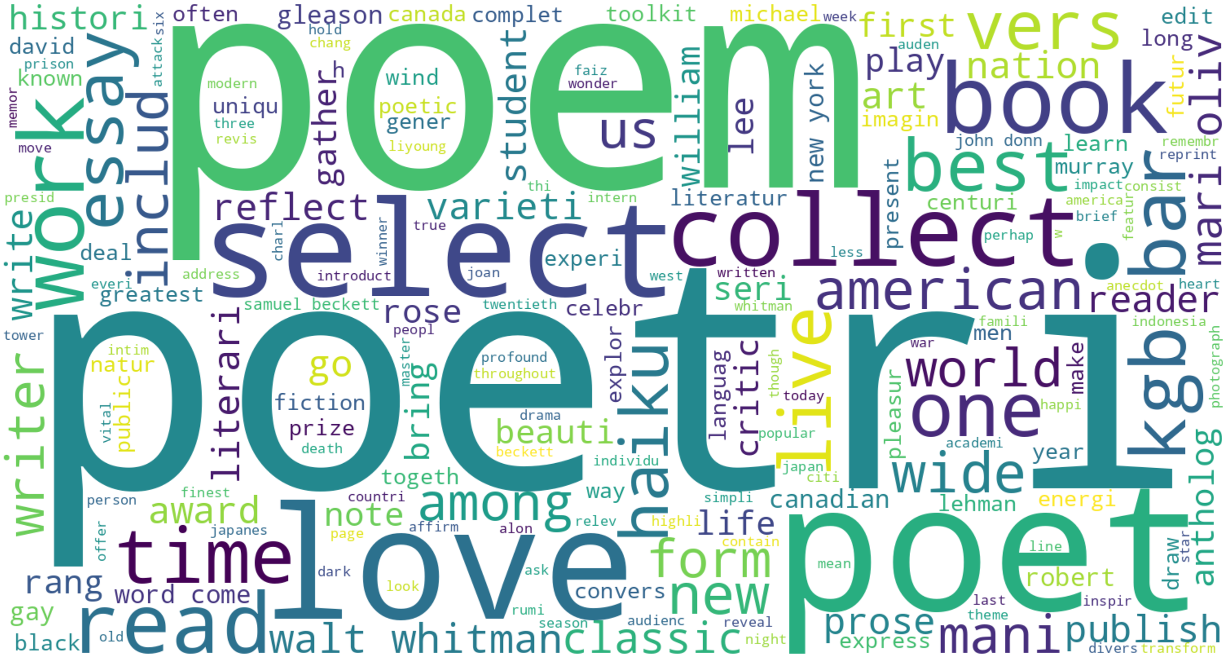


CLUSTER NUMBER: 1



CLUSTER NUMBER: 9

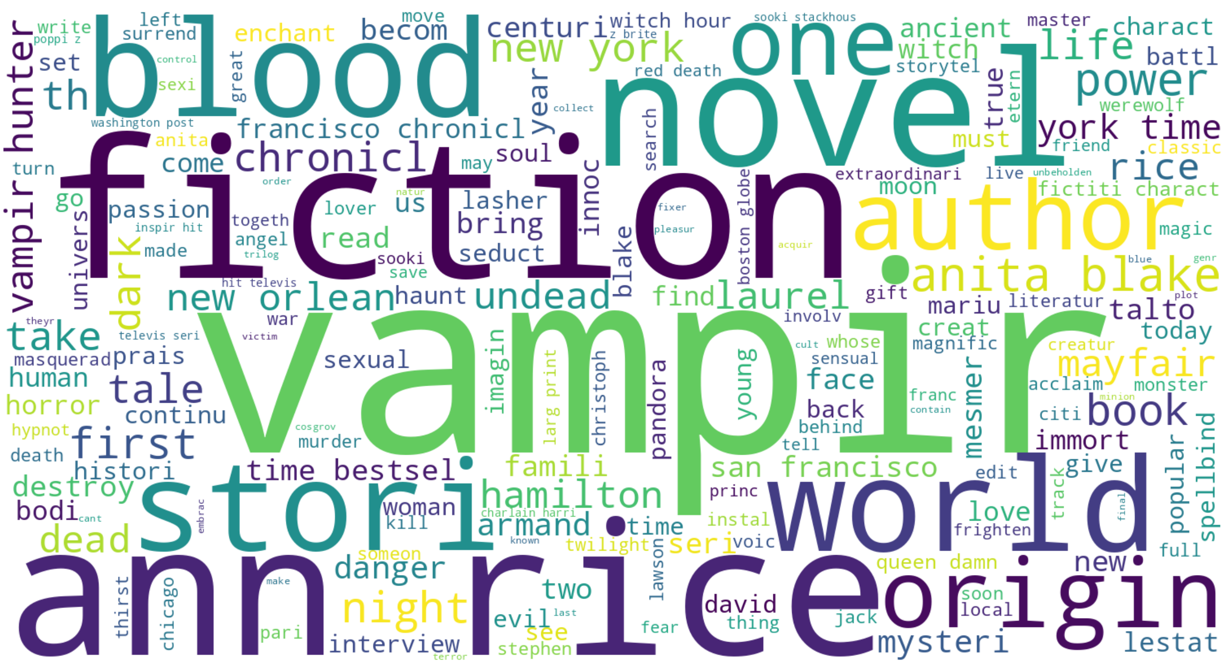
CLUSTER NUMBER: 11



CLUSTER NUMBER: 8

CLUSTER NUMBER: 5

CLUSTER NUMBER: 2



CLUSTER NUMBER: 6

CLUSTER NUMBER: 3

CLUSTER NUMBER: 10

Validation

While the silhouette scores suggest less than optimal clustering results due to their negative values, this does not fully capture the effectiveness of our clustering. Contrary to the silhouette scores, the word clouds generated for individual clusters reveal a different story, the representations show that some clusters do indeed encapsulate distinct and meaningful themes.

In [ ]:
```
from scipy.spatial.distance import squareform
from sklearn.metrics import silhouette_score


distance_matrix = squareform(dist_array)

silhouette_avg = silhouette_score(distance_matrix, clusters, metric='precomputed')
print(f"Silhouette Score: {silhouette_avg}")
```

```
Silhouette Score: -0.005005541642158681
```

K-Means

In [ ]:
```
from sklearn import metrics
def evaluate(km, X):
  print('silhouette: ', metrics.silhouette_score(X, km.labels_, sample_size=2000))
```

In [ ]:
```
from sklearn.cluster import KMeans


kmeans = KMeans(
        n_clusters=20,
        max_iter=100,
        n_init=1,
        random_state=42,
    ).fit(tfidf)
```

```
evaluate(kmeans, tfidf)
```

```
silhouette:  0.0032707083539478326
```

Clustering sparse data with k-means

Problem: Kmeans random initialization can be very sensible using high dimensional sparse data.

Solutions:

- Performing dimensionality reduction using SVD
- Increase the number of runs with independent random initiations n_init

Dimensionality reduction using SVD

In [ ]:
```
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer

lsa = make_pipeline(TruncatedSVD(n_components=1500), Normalizer(copy=False))

X_lsa = lsa.fit_transform(tfidf)
explained_variance = lsa[0].explained_variance_ratio_.sum()

print(f"Explained variance of the SVD step: {explained_variance * 100:.1f}%")
```

```
Explained variance of the SVD step: 72.3%
```

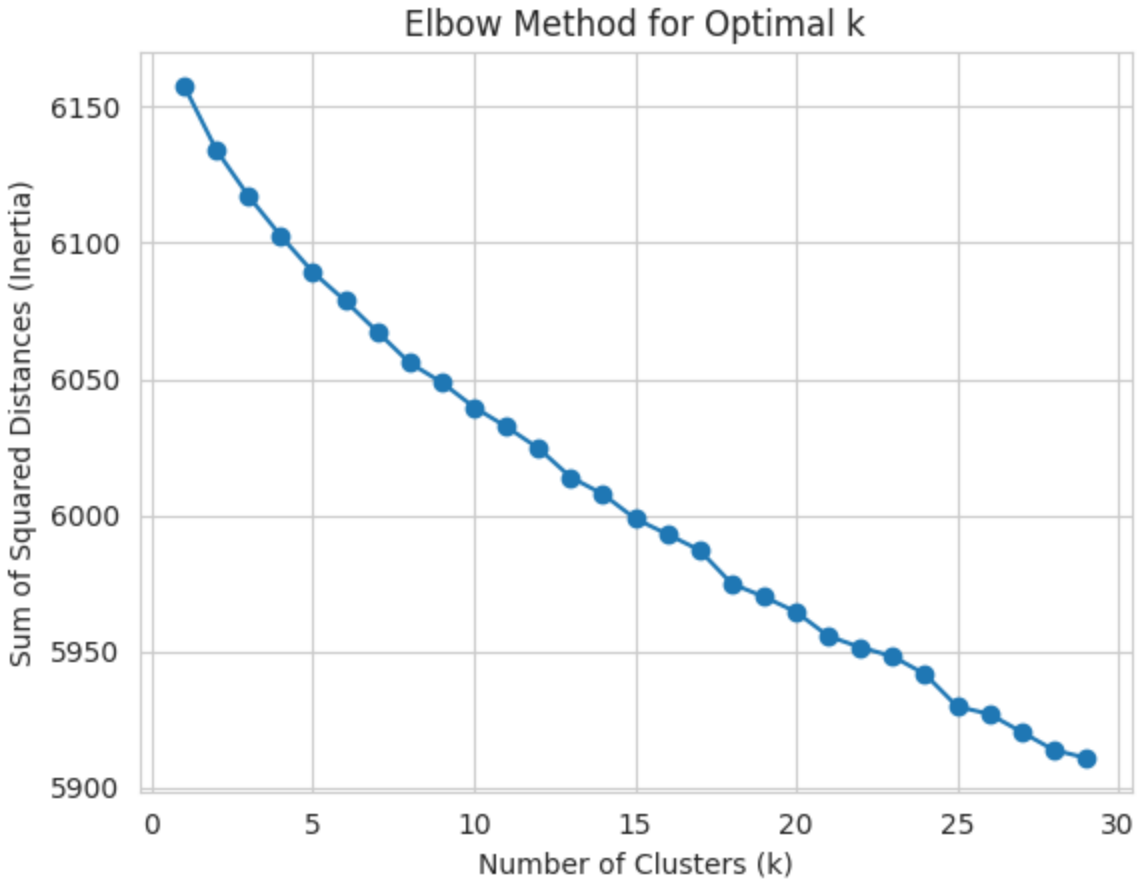
The dimensions of the data has been reduced while keeping the 72.3% of the total variance

Finding optimal k

The silhouette scores are close to zero, indicating overlapping or indistinct clusters, and the elbow method does not present a clear point at which the inertia's rate of decrease significantly changes. This is a bit disappointing as it implies that the dataset may not have a clear or meaningful cluster structure that can be easily detected with these methods.

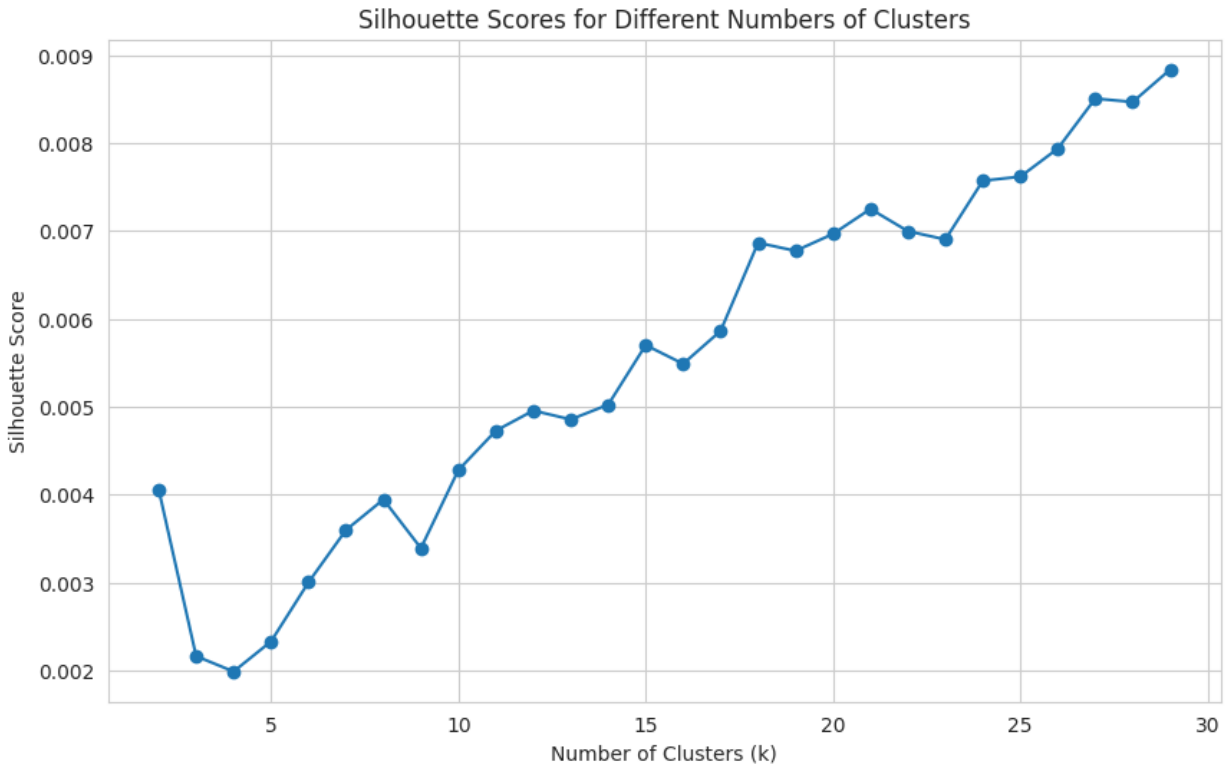
In [ ]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs


k_values = range(1, 30)

inertia_values = []

# Run k-means for each k and calculate inertia
for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X_lsa)
    inertia_values.append(kmeans.inertia_)
```

```
# Plot the elbow method
plt.plot(k_values, inertia_values, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Distances (Inertia)')
plt.show()
```



In [ ]:

```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

silhouette_scores = []

# Calculate silhouette score for each k
for k in range(2, 30):
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X_lsa)
    labels = kmeans.labels_
    score = silhouette_score(X_lsa, labels)
    silhouette_scores.append(score)

# Plotting the silhouette scores
plt.figure(figsize=(10, 6))
plt.plot(range(2, 30), silhouette_scores, marker='o')
plt.title('Silhouette Scores for Different Numbers of Clusters')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.show()
```



Select k = 20 based on domain knowledge

In [ ]:
```
kmeans_LDA = KMeans(
    n_clusters=20,
    max_iter=100,
    n_init=10,
)

labels = kmeans_LDA.fit_predict(X_lsa)
df['cluster_k'] = labels
```

In [ ]:
```
df.head()
```

Out[ ]:

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCount	c
0	0884115631	The Shepherd	NaN	Frederick Forsyth	NaN	1960-06-01	A pilot and his plane are saved by a mysteriou...	0	
1	0060175648	Identity	A Novel	Milan Kundera	Harper	1998-04-21	Milan Kundera's Identity translated from the F...	176	
2	0394535383	Pillar of the Sky	A Novel	Cecelia Holland	Alfred a Knopf Incorporated	1985	Two thousand years ago, at the site of Stonehe...	534	
3	0843114401	Jingle Bear	NaN	Stephen Cosgrove	NaN	1985	When the other bears get ready to hibernate fo...	36	
4	0451207408	The Adventurer	NaN	Jaclyn Reding	Signet Book	2002	While attempting to return a mysterious, legen...	314	

After a careful examination of the word clouds generated from the K-means algorithm clusters, it is evident that, despite the unimpressive results indicated by the elbow method and the silhouette scores, the clusters contain discernible topics and genres. Besides, some of these topics align with those identified by the hierarchical clustering method, suggesting that there are indeed meaningful patterns within the data that both clustering approaches are able to capture, regardless of the quantitative metrics implying otherwise.

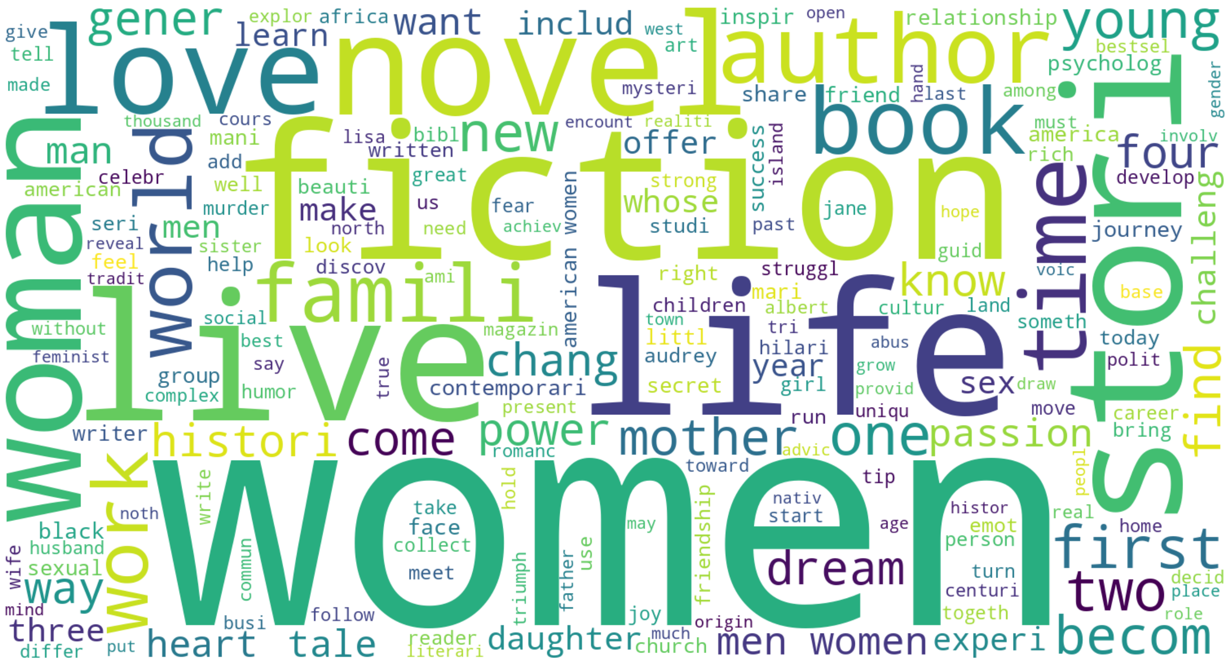
In [ ]:

```
# Get unique cluster labels
unique_clusters = df['cluster_k'].unique()

# Create a dictionary to store DataFrames for each cluster
cluster_dataframes = {}

# Split the DataFrame by cluster label
for cluster_label in unique_clusters:
    cluster_dataframes[cluster_label] = df[df['cluster_k'] == cluster_label]
    joined_genre = ' '.join(cluster_dataframes[cluster_label]['CombinedText'])
    print("CLUSTER NUMBER: ", cluster_label)
    create_wordcloud(joined_genre)
```

```
CLUSTER NUMBER:  2
```

CLUSTER NUMBER: 3

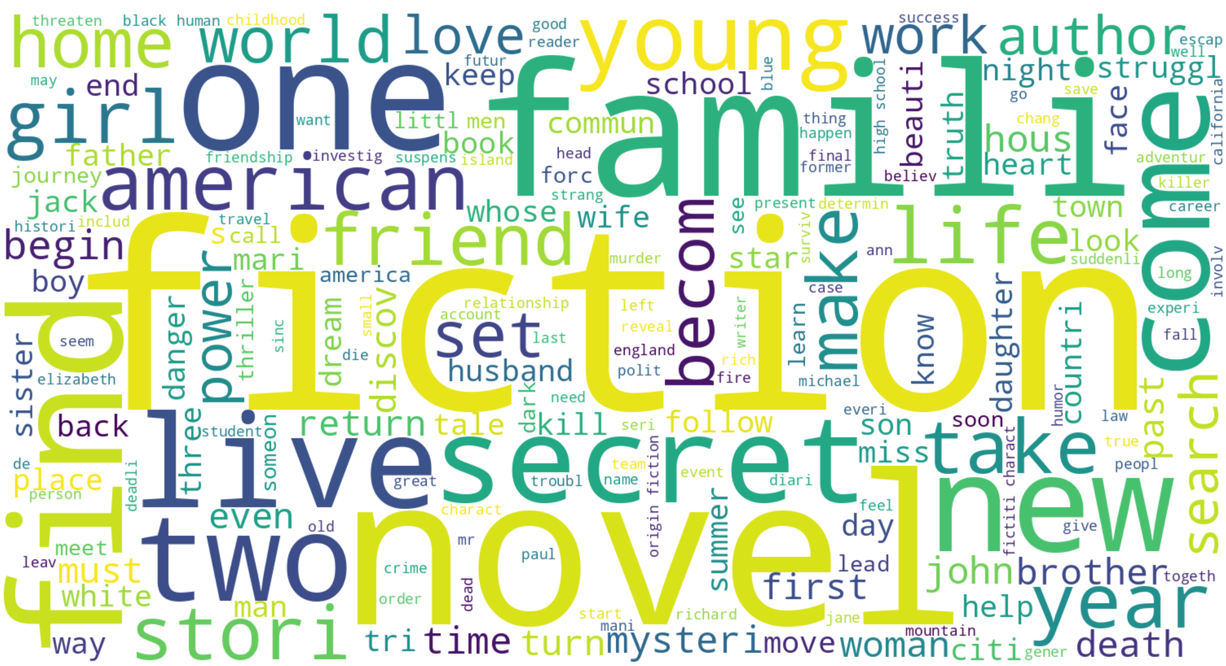
CLUSTER NUMBER: 18



CLUSTER NUMBER: 7

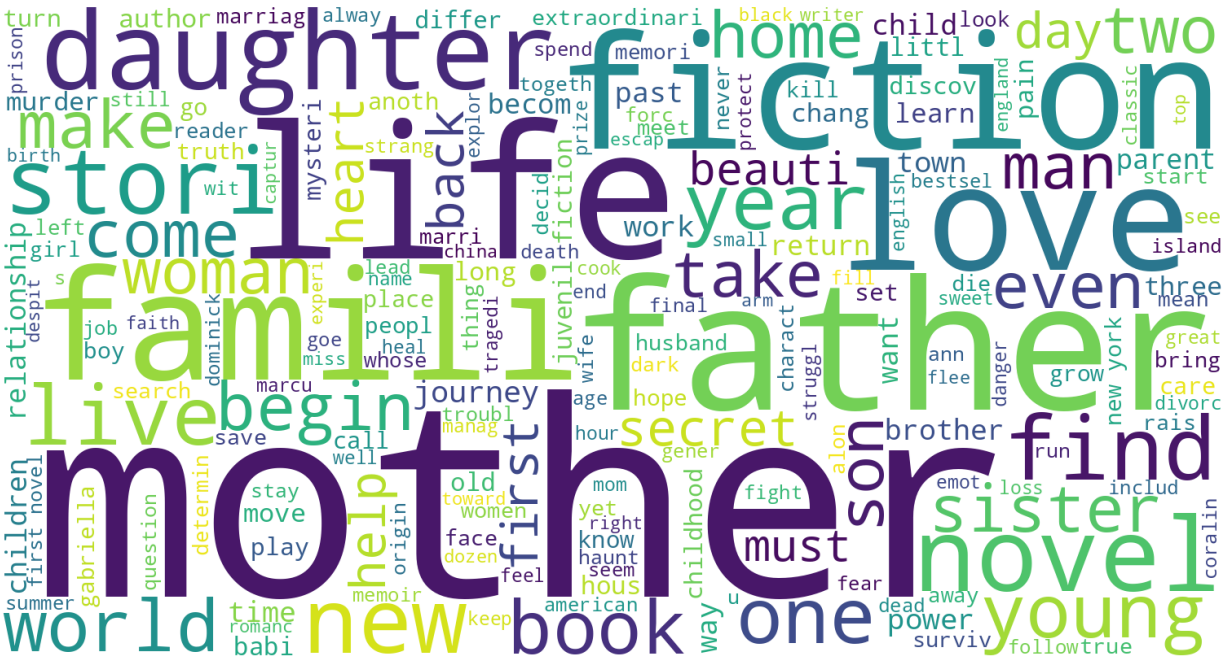
CLUSTER NUMBER: 10

CLUSTER NUMBER: 14

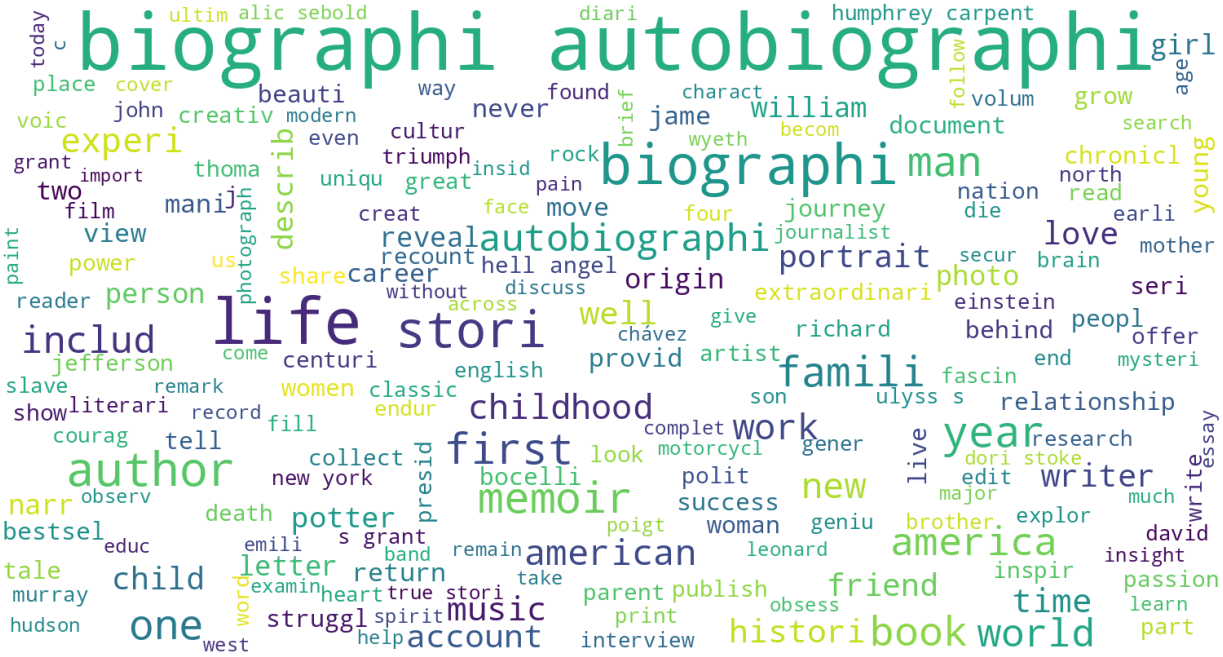


CLUSTER NUMBER: 12

CLUSTER NUMBER: 13



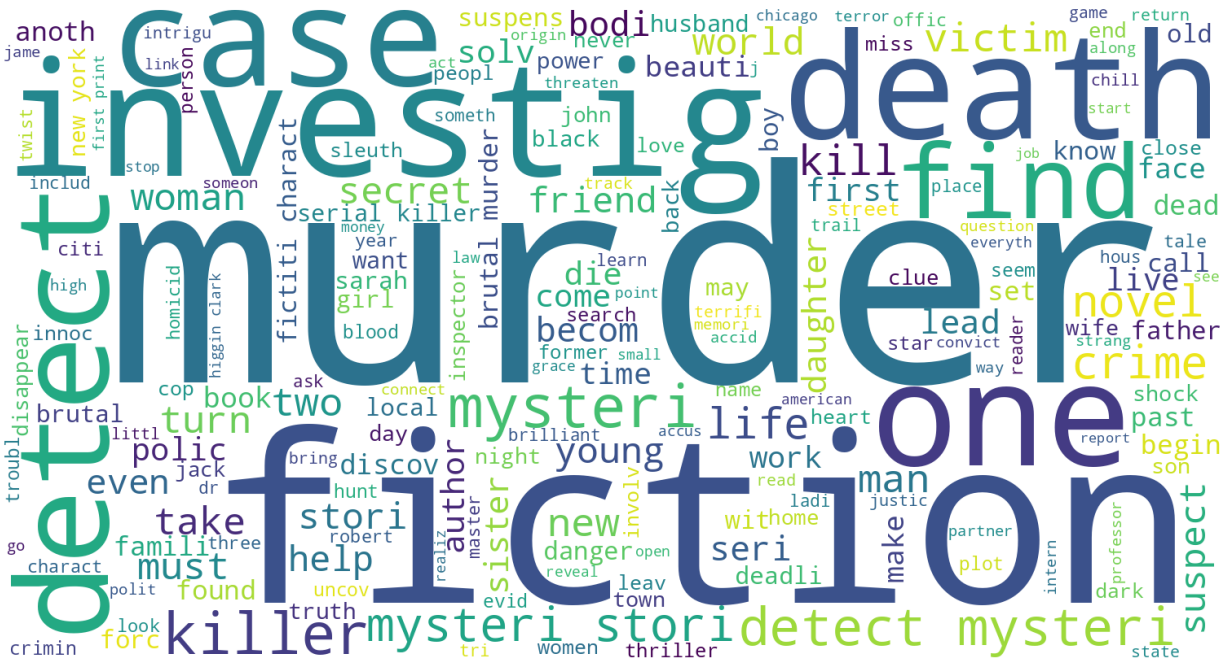
CLUSTER NUMBER: 17



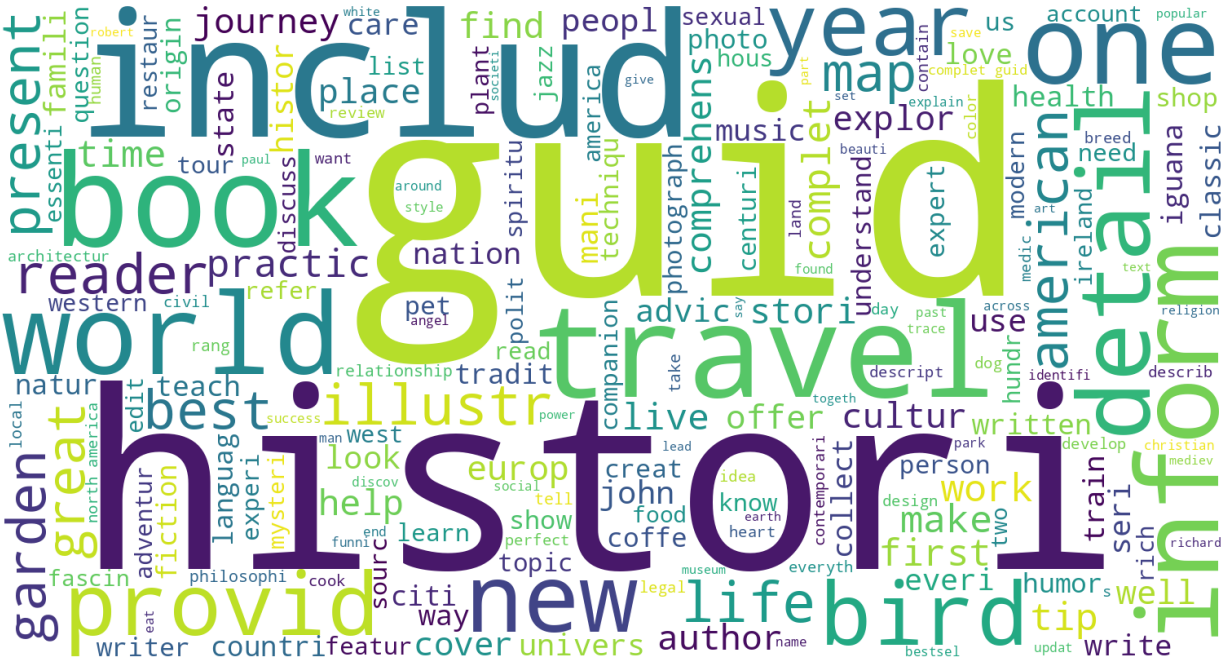
CLUSTER NUMBER: 1

CLUSTER NUMBER: 6

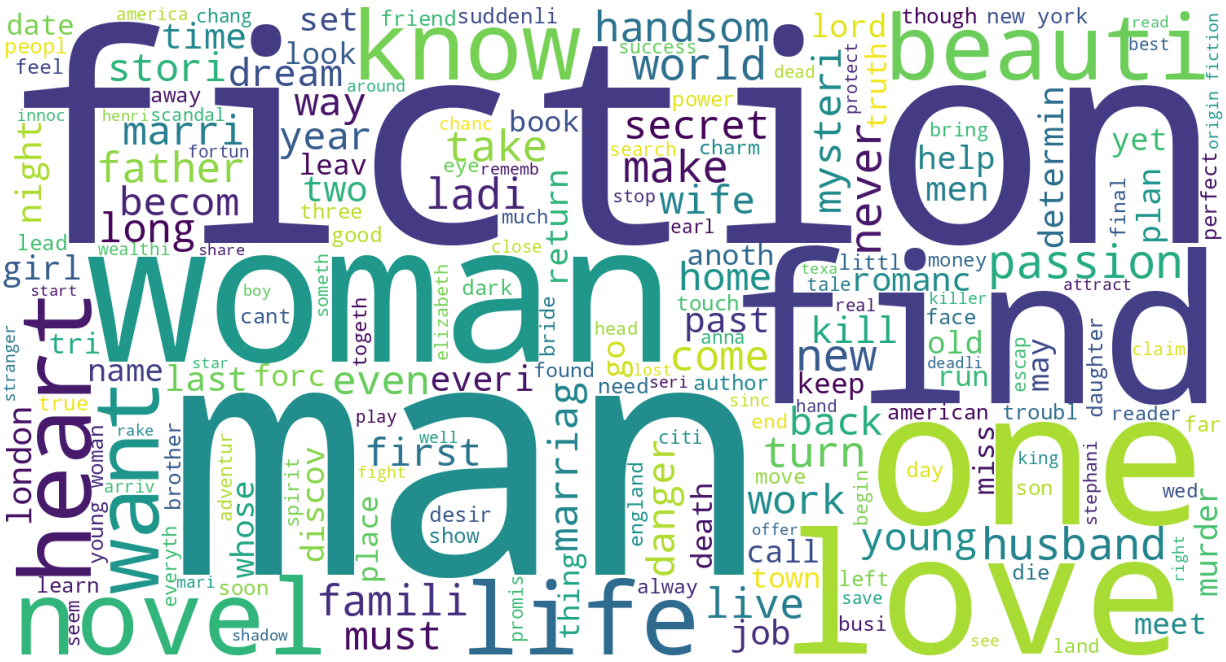
CLUSTER NUMBER: 8



CLUSTER NUMBER: 5



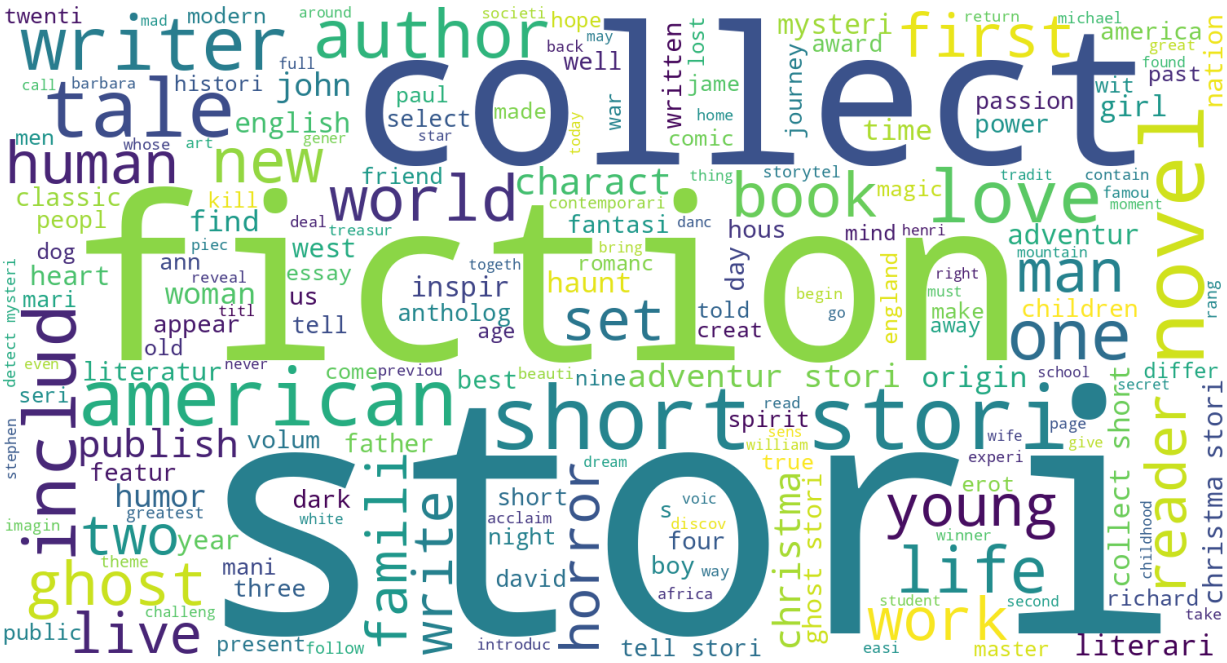
CLUSTER NUMBER: 9



CLUSTER NUMBER: 19

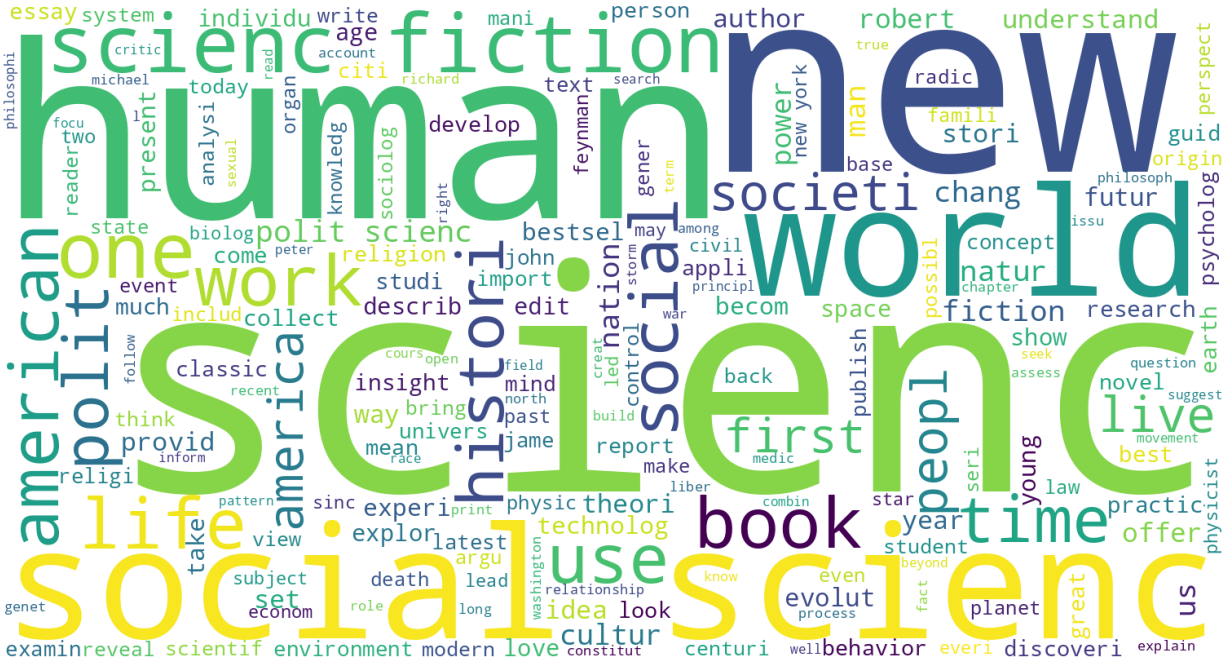
CLUSTER NUMBER: 11

CLUSTER NUMBER: 4



CLUSTER NUMBER: 16

CLUSTER NUMBER: 0



CLUSTER NUMBER: 15

In [ ]:
```
cluster_distribution = df['cluster_k'].value_counts().sort_index()
print(cluster_distribution)
```

```
0      154
1      777
2      200
3      493
4      284
5      299
6      176
7       82
8      468
9      401
10     486
11     400
12     225
13     195
14    1099
15      98
16      96
17     125
18     147
19      68
Name: cluster_k, dtype: int64
```

Recommender system

Finally, we use our clustering results to build a book recommender system, this system is designed to personalize recommendations for each user, focusing particularly on their highest-rated book, it uses the cosine similarity to find books that are similar to the user's highest-rated book within the same cluster. This way improves the relevance and accuracy of the recommendations, aligning them closely with the user's demonstrated preferences.

In [ ]:
```
df_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66082 entries, 0 to 66081
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Unnamed: 0   66082 non-null  int64 
 1   User-ID      66082 non-null  int64 
 2   ISBN         66082 non-null  object
 3   Book-Rating  66082 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 2.0+ MB
```

In [ ]:
```
from sklearn.metrics.pairwise import cosine_similarity

def recommend_books(user_id, df_reviews, df, tfidf_matrix):
    # Select all reviews for the specified user
    user_reviews = df_reviews[df_reviews['User-ID'] == user_id]

    # If the user has no reviews, return an empty list
    if user_reviews.empty:
        print(f"No reviews found for User ID {user_id}.")
        return []

    # Find the ISBN of the highest-rated book for the user
    highest_rated_isbn = user_reviews.loc[user_reviews['Book-Rating'].idxmax(), 'ISBN'
```

```
    print(df[df['isbn'] == highest_rated_isbn]['title'])

    # Find the index of the highest-rated ISBN in the book DataFrame
    index_of_highest_rated = df[df['isbn'] == highest_rated_isbn].index[0]

    # Assuming tfidf_matrix is your TF-IDF matrix
    tfidf_row_highest_rated = tfidf_matrix[index_of_highest_rated, :]

    # Calculate cosine similarity with all other books in the cluster
    similarities = cosine_similarity(tfidf_row_highest_rated.reshape(1, -1), tfidf_mat

    # Get the cluster of the highest-rated book
    cluster_of_highest_rated = df.loc[df["isbn"] == highest_rated_isbn, 'cluster_k'].v

    # Mask out books not in the same cluster
    cluster_similarities = similarities[0, df['cluster_k'] == cluster_of_highest_rated

    # Find the top 5 most similar books within the cluster
    top_similar_indexes = cluster_similarities.argsort()[-6:-1][::-1]

    # Filter out books that the user has already rated
    user_rated_books = set(user_reviews['ISBN'])
    recommended_books = []

    # Display the top 5 most similar and unrated books' titles
    print(f"Top 5 recommended books for User ID {user_id} (unrated):")
    for similar_index in top_similar_indexes:
        similar_isbn = df.loc[similar_index, 'isbn']
        if similar_isbn not in user_rated_books:
            similar_title = df.loc[similar_index, 'title']
            similar_author = df.loc[similar_index, 'authors']
            similar_genre = df.loc[similar_index, 'categories']
            recommended_books.append(similar_title)
            print(f'{similar_title}, {similar_author}, {similar_genre}')

    return recommended_books
```

In [ ]:

```
# Assuming your DataFrame is named df
random_user_id = df_reviews['User-ID'].sample(1, random_state=523).values[0]
print(random_user_id)
recommendations = recommend_books(random_user_id, df_reviews, df, tfidf)
```

```
145459
5978    Jurassic Park : [novel]
Name: title, dtype: object
Top 5 recommended books for User ID 145459 (unrated):
Fried by Jury, Claudia Bishop, Fiction
Lucky Jim, Kingsley Amis, College teachers
The Sacred Balance, David Suzuki, Amanda McConnell, Conservation of natural resources
Temporary Mistress, Susan Johnson, Fiction
The Power of a Praying Parent, Stormie Omartian, Devotional calendars
```

Collaborative filtering

Collaborative filtering is a method used in recommendation systems to suggest items (like books) to users. It works by analyzing the preferences and behaviors of various users and then

making recommendations based on similarities among them. There are two main types:

User-Based Filtering: This type makes recommendations based on the preferences of similar users. For instance, if two users have rated several books similarly, the system will recommend books liked by one user to the other. The similarity between users is calculated using methods like Pearson correlation or cosine similarity.

Item-Based Collaborative Filtering: Instead of focusing on user similarity, this method recommends items based on their similarity with items that the target user has already rated. It calculates the similarity between items (like books), so if a user likes a particular book, the system will recommend other books that are similar to it.

Collaborative filtering addresses the limitations of content-based systems, which only suggest items similar to what the user has already liked, and can't capture diverse tastes or provide cross-genre recommendations. It also adds a personal 'taste' to recommendations, unlike content-based systems that offer the same suggestions to everyone.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
df_sim = df.copy()
df_sim = df_sim[df_sim['Book-Rating']!=0]
df_similarity=df_sim.copy()
df_similarity = df_similarity.astype(str)
df_similarity.drop('Unnamed: 0', axis=1, inplace=True)
df_similarity['ISBN'] = df_similarity['ISBN'].str.replace('[^a-zA-Z0-9]', '', regex=Tr
#df_similarity[df_similarity['User-ID']=='11676']
df_similarity.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	User-ID	ISBN	Book-Rating
0	16	0345402871	9
1	17	0425099148	7
2	26	0449005615	9
3	26	0446310786	10
4	39	0553582909	8

Selected users analysis

In addition to the overall accuracy of SVD model in the entire dataset and recommending the top 10 best books for a user, we plan to take a sample of users, predict their scores, and then compare these predictions with the actual scores.

Before we remove these selected users from the main dataset, we must ensure that the collaborative filtering model has sufficient information about the books, based on ratings from other users, which the selected users have also rated.

In [ ]:
```
#Users and their number of rating score
def user_books(users):
    user_books_dict = {}
    selected_users = df_similarity[df_similarity['User-ID'].isin(users)]
    for user in users:
        user_books_dict[user] = selected_users[selected_users['User-ID'] == user]['ISE
    return user_books_dict
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
users = ['15497','23768','11676','6242','6073']
user_books_dict = user_books(users)
value_counts = {}
value_counts = {key: len(values) for key, values in user_books_dict.items()}
value_counts
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
{'15497': 1, '23768': 26, '11676': 752, '6242': 33, '6073': 15}
```

In [ ]:
```
# table without the selected users
df_no_selected_users = df_similarity[~df_similarity['User-ID'].isin(user_books_dict.ke
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
len(df_similarity) - len(df_no_selected_users)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
827
```

In [ ]:
```
# Counts of selected user's ISBN
item_frequency = {}

for key in user_books_dict:
    for item in user_books_dict[key]:
        if item in item_frequency:
            item_frequency[item] += 1
        else:
            item_frequency[item] = 1

print(item_frequency)
```

{'0140293248': 3, '0805063897': 1, '1400031346': 2, '0684857820': 2, '0743418174': 2,
'0060934417': 2, '0060931418': 1, '0060929871': 1, '0060198125': 1, '1580173128': 1,
'0375413634': 1, '0380718340': 2, '038548951X': 2, '0316666343': 2, '0156505150': 1,
'0312265859': 2, '0394800184': 2, '0671880187': 1, '0671740504': 1, '0670869902': 1,
'0553109588': 2, '0394800133': 2, '0516098020': 1, '0440226198': 1, '039480029X': 1,
'0394800206': 1, '0440204194': 1, '0805054073': 1, '0807553670': 1, '0812502124': 1,
'0812550706': 1, '0786861347': 1, '0786866462': 1, '0786867175': 1, '0786885688': 1,
'0786881852': 1, '0786868716': 1, '0786867906': 1, '0786889667': 1, '0804105820': 1,
'080410753X': 1, '0804119708': 1, '0804115761': 1, '0804114986': 1, '0804111359': 1,
'080411109X': 1, '0804108749': 1, '0802130208': 2, '067697175X': 1, '0671888587': 1,
'0679425241': 1, '0679432477': 1, '0679407596': 1, '0671750402': 1, '0671732277': 1,
'0671867091': 1, '0671873199': 1, '0671873202': 1, '0671789422': 1, '0671793489': 1,
'0671793551': 1, '0679439323': 1, '0679776591': 1, '0679776818': 2, '0679777547': 2,
'0684184710': 1, '0684832178': 1, '0679775439': 1, '067945960X': 1, '0679721886': 1,
'0679441220': 1, '0679442790': 1, '0679751602': 1, '067976402X': 2, '0679731148': 1,
'0679735909': 2, '0684835959': 1, '0671003364': 1, '0671003755': 1, '0670894699': 1,
'0670851086': 1, '0670880728': 1, '0670891576': 1, '0670892963': 1, '0671020293': 1,
'0670844527': 1, '0671007696': 1, '0671011367': 1, '0671011375': 1, '0671014188': 1,
'0671014196': 1, '067101420X': 1, '0671014218': 1, '0609603620': 1, '060980619X': 1,
'0618002227': 1, '0618129030': 1, '0671568175': 1, '0671644475': 1, '0671705091': 1,
'0671722840': 1, '0671670646': 1, '0671683993': 1, '0671693816': 1, '067103619X': 1,
'0671036505': 1, '0671041789': 1, '067102423X': 1, '0671027573': 1, '0671042262': 1,
'0671477099': 1, '0671510053': 1, '0671454048': 1, '0671042572': 1, '0749395990': 1,
'0743439740': 1, '0743437802': 1, '0743418220': 1, '0743454146': 1, '0743469801': 1,
'0743460529': 1, '0743467523': 1, '0767902890': 2, '0767903579': 1, '0767905385': 1,
'0767908473': 1, '0767912233': 1, '0767905318': 1, '0767902513': 1, '076791404X': 1,
'0767914767': 1, '0756400856': 1, '0684841185': 1, '0689846355': 1, '0688177859': 1,
'0698113608': 1, '0688171923': 1, '0688167888': 1, '0684856093': 1, '0684865459': 1,
'068484477X': 2, '0684848783': 1, '0684855755': 1, '0684872153': 1, '0684874350': 1,
'074322535X': 1, '0740734121': 1, '0743202759': 1, '0743206029': 1, '0743225406': 1,
'0743225724': 1, '0743411250': 1, '0743411439': 1, '0743262174': 1, '0743229878': 1,
'0743237188': 1, '0451628039': 1, '0451526341': 1, '0452268060': 1, '0452282195': 1,
'0452282829': 1, '0452283205': 1, '0452283442': 1, '0452281903': 1, '0452279178': 1,
'0452281032': 1, '0452281202': 1, '0451205626': 1, '0451207521': 1, '0451207645': 1,
'0451204301': 1, '0451410904': 1, '0451454243': 1, '0451209907': 1, '0451210638': 1,
'0451403703': 1, '0515130923': 1, '0515130966': 1, '0515132020': 1, '0515132187': 1,
'051513239X': 1, '051513287X': 1, '0515133302': 1, '0515128600': 1, '0515122734': 1,
'051512608X': 1, '0515127833': 1, '0515128015': 1, '0515128554': 1, '0515133973': 1,
'051751401X': 1, '0515135062': 1, '0515135739': 1, '0515136379': 1, '0515136530': 1,
'0486270629': 1, '048627263X': 1, '0515110132': 1, '0515118249': 1, '051511992X': 1,
'0515107867': 1, '0515088757': 1, '0449005615': 1, '0449006522': 1, '0449207552': 1,
'0449208281': 1, '0449149900': 1, '0449203794': 1, '0449205878': 1, '0449212602': 1,
'0446610038': 1, '0446611476': 1, '0446611778': 1, '0446609838': 1, '0446608122': 1,
'0446608386': 1, '0446608815': 1, '0446608955': 1, '0446609145': 1, '0446609323': 1,
'0446611808': 1, '0446611921': 1, '044661193X': 1, '0446675059': 1, '0446676098': 1,
'0446679364': 1, '0446612510': 1, '0446612545': 1, '0446612626': 1, '0446671029': 1,
'0451197747': 1, '0451163494': 1, '0451169522': 1, '0451169530': 1, '0451172817': 1,
'0451156609': 1, '0451190521': 1, '0451191137': 1, '0451188454': 1, '0451177037': 1,
'0451179188': 1, '0451180410': 1, '0449223612': 1, '0449227421': 1, '0449221490': 1,
'0449213943': 1, '0449216861': 1, '0449217493': 1, '0449219364': 1, '0449219569': 1,
'0451092163': 1, '0449910571': 1, '0553582526': 1, '0553582747': 1, '0553582763': 1,
'0553582895': 1, '0553582143': 1, '0553583573': 1, '0553583913': 1, '0553582127': 1,
'0553579754': 1, '0553580906': 1, '0553587188': 1, '0571197639': 1, '0553802496': 1,
'0553562738': 1, '055356160X': 1, '055356451X': 1, '0553382179': 1, '0553382411': 1,
'0553560247': 1, '0553569031': 1, '0553573403': 1, '0553573861': 1, '0553575422': 1,
'0553576801': 1, '0553572512': 1, '055357230X': 1, '0553569155': 1, '0553569619': 1,
'0553569910': 1, '0553571583': 1, '055357227X': 1, '0553380095': 2, '059045367X': 1,
'0590436031': 1, '0590629794': 1, '0590494465': 1, '059035342X': 1, '0590353403': 1,
'0552998834': 1, '0552546933': 1, '0553053272': 1, '0553151657': 1, '0553076256': 1,
'0553096834': 1, '0525945210': 1, '052594527X': 1, '0517703963': 1, '0552141275': 1,

'0552142379': 1, '0552143774': 1, '0525946829': 1, '0553279378': 1, '0553279556': 1,
'0553280147': 1, '0553280511': 1, '0553281798': 1, '0553273639': 1, '0553277472': 1,
'0553284789': 1, '0553297260': 1, '0553348973': 1, '0553348981': 1, '0553289411': 1,
'0553289691': 1, '0553272616': 1, '0553227599': 1, '0553213148': 1, '0553210092': 1,
'0553212451': 1, '0553213105': 2, '055321313X': 1, '0553267868': 1, '0553258001': 1,
'0974433926': 1, '0971880107': 1, '1400030927': 1, '0932653251': 1, '1558531025': 1,
'155874262X': 1, '1558744150': 1, '1558745157': 1, '1558745718': 1, '1559029838': 1,
'1551665301': 1, '1550415980': 1, '1551668726': 1, '1551667517': 1, '1551667894': 1,
'1551668246': 1, '0842329129': 1, '0842329218': 1, '0842329250': 1, '0842342702': 1,
'0849932017': 1, '0821749668': 1, '0836218353': 1, '0836218817': 1, '0890818304': 1,
'0891073906': 1, '0891455094': 1, '0887307876': 1, '0892967811': 1, '0899193625': 1,
'0894803204': 1, '0894808249': 1, '0875163831': 1, '0871136791': 2, '087113795X': 1,
'0877017883': 1, '1893017508': 1, '1885211716': 1, '1931561648': 1, '1903039568': 1,
'1576733165': 1, '1580082319': 1, '1573229571': 1, '1573227331': 2, '1565302370': 1,
'1573220221': 1, '1573220485': 1, '1573222267': 1, '1857978013': 1, '1844262553': 1,
'184195280X': 1, '0842329242': 1, '0312282575': 1, '0312278586': 1, '0312274920': 1,
'0312263120': 1, '0312437285': 1, '031242227X': 1, '0312310617': 1, '0312305060': 1,
'0312291639': 2, '0312243022': 2, '0312147015': 1, '031220499X': 1, '0312195516': 2,
'0312199430': 1, '0312976860': 1, '0312977026': 1, '0312966970': 1, '0312983298': 1,
'0312982518': 1, '0312982402': 1, '0312978332': 1, '0312979479': 1, '0312890044': 1,
'0312924801': 1, '0312866097': 1, '0312925271': 1, '0312955006': 1, '0152007822': 1,
'0156007754': 1, '0156007479': 1, '014100018X': 1, '0142004278': 1, '0142004235': 1,
'0142002267': 1, '0142000345': 1, '014131088X': 1, '0142000205': 1, '0307262073': 1,
'0201196778': 1, '0345313860': 1, '0345311485': 1, '0345305183': 1, '034529873X': 1,
'0345340906': 1, '0345335465': 1, '0345346491': 1, '0345260791': 1, '0345404793': 1,
'034541005X': 1, '0345402308': 1, '0345402022': 1, '034540047X': 1, '0345404475': 2,
'0345412214': 1, '0345423615': 1, '0345421426': 1, '0345417623': 1, '034541389X': 1,
'0345413873': 1, '0345391055': 1, '0345368754': 1, '0345361792': 1, '0345389964': 1,
'0345370775': 1, '034538475X': 1, '0345378490': 1, '0345378482': 1, '0316693286': 1,
'0316693235': 1, '0316734837': 1, '0316693200': 1, '0316603570': 1, '0316776963': 1,
'0316969443': 1, '0316898163': 1, '0316781142': 1, '0316780375': 1, '0316107352': 1,
'0316096199': 1, '0312995423': 1, '0312985207': 1, '0312983867': 1, '0316168688': 1,
'0340401486': 1, '0330376276': 1, '0330375253': 1, '0330367358': 1, '006093395X': 1,
'0060929790': 1, '0060938455': 1, '0060972459': 1, '0060970790': 1, '0060915153': 1,
'0060958022': 2, '0060504080': 1, '0060502258': 1, '0060512822': 1, '0061093343': 1,
'0061092886': 1, '0061092193': 1, '0060988649': 1, '0060987561': 1, '0060987103': 2,
'0060977337': 1, '0061000175': 1, '0061015725': 1, '0060391626': 1, '002542730X': 1,
'006019491X': 1, '0060198133': 1, '0060256672': 1, '0060008032': 1, '0060002492': 1,
'0060175400': 1, '0060172916': 1, '0060093579': 1, '0345425707': 1, '014023828X': 1,
'0140185216': 1, '014025448X': 1, '0140167773': 1, '0140067477': 1, '0140143505': 1,
'014038572X': 1, '0140620850': 1, '0140620303': 1, '0140042210': 1, '0140280464': 1,
'014028009X': 1, '0140279288': 1, '014027927X': 1, '0140278826': 1, '0140298479': 1,
'0064471969': 1, '0064407675': 1, '0064407055': 1, '0066212855': 1, '0070212570': 1,
'0061099708': 1, '0061099694': 1, '0061098035': 1, '0061097861': 1, '0061097667': 1,
'0061099015': 1, '0064400557': 1, '0064400085': 1, '006251279X': 1, '0064400034': 1,
'0099747200': 1, '0099604302': 1, '0099464810': 1, '0099521016': 1, '0099263815': 1,
'0099245027': 1, '0099244926': 1, '0099410869': 1, '0345337662': 1, '0425177173': 1,
'0425176932': 1, '0425170349': 1, '0425182878': 1, '0425169863': 1, '042518286X': 1,
'0425179559': 1, '0425155404': 1, '0425144062': 1, '042518630X': 1, '0425167313': 1,
'0425163407': 1, '0425157539': 1, '0425161242': 1, '0439139597': 1, '0439136369': 1,
'0439404371': 1, '0440127793': 1, '043912042X': 1, '0425192725': 1, '0425192032': 1,
'0425189864': 1, '042518904X': 1, '0425187144': 1, '0425189414': 1, '0439083699': 1,
'0439064864': 1, '0425195449': 1, '0399135782': 1, '0399144714': 1, '039914840X': 1,
'0399148248': 1, '0399148027': 1, '0399144463': 1, '0399146431': 1, '0399146008': 1,
'0399146601': 1, '0394900901': 1, '039592720X': 2, '0399148639': 1, '0425107469': 1,
'0425099148': 1, '0425098605': 1, '042513525X': 1, '0425132048': 1, '0425122123': 1,
'0425121259': 1, '0425120279': 1, '042511984X': 1, '0399150897': 1, '0425092178': 1,
'0440146577': 1, '0446353205': 1, '0446343455': 1, '0446322180': 1, '0446356611': 1,
'0446356832': 1, '0446364193': 1, '0446356859': 1, '0446360589': 1, '0441004520': 1,
'0441003257': 1, '044100069X': 1, '0441865038': 1, '0441790348': 1, '0441009239': 1,

'0441172695': 1, '0441569595': 1, '0446364754': 1, '0446365386': 1, '0446602450': 1,
'0446601640': 1, '0446600474': 1, '0446600253': 1, '0446532231': 1, '0446530891': 1,
'0446606251': 1, '0446606189': 1, '0446605409': 1, '0446605239': 1, '0446603309': 1,
'0446604844': 1, '0446604666': 1, '0446603716': 1, '0446604801': 1, '0446527785': 1,
'044651652X': 1, '0446391301': 1, '0446370398': 1, '044651862X': 1, '0446527165': 1,
'0446527017': 1, '0446525537': 1, '0446519138': 1, '0446525502': 1, '0446519790': 1,
'0440998050': 1, '0440216818': 1, '0440216745': 1, '0440215900': 1, '0440213525': 1,
'0440212359': 1, '0440215625': 1, '044021145X': 1, '0440224713': 1, '0440224675': 1,
'0440222656': 1, '044022103X': 1, '0440218535': 1, '0440221471': 1, '0440211263': 1,
'0440193613': 1, '0440201926': 1, '0440180295': 1, '0440176484': 1, '0440173701': 1,
'0440209412': 1, '0440207770': 1, '0440207622': 1, '044020352X': 1, '0440204887': 1,
'0440225108': 1, '0440486599': 1, '0440940001': 1, '0440507626': 1, '0440234743': 1,
'0440226430': 2, '0440226104': 1, '0440225701': 1, '0440225698': 1, '0440225299': 1,
'0440295858': 1, '044024126X': 1, '0440241073': 1, '0440237262': 1, '044023722X': 1,
'0440236673': 1, '0440236061': 1, '0440235804': 1, '0375400117': 1, '0374270325': 1,
'0374199698': 1, '0375406530': 1, '0375422307': 1, '037541309X': 1, '0375412824': 1,
'0375412530': 1, '0375411070': 1, '0375502238': 1, '037582233X': 1, '0375760911': 1,
'0375758453': 1, '0375756981': 1, '0380700247': 1, '0380542625': 1, '0380018179': 1,
'0380005239': 1, '0380619458': 1, '0375703055': 1, '0375702709': 1, '0375504397': 1,
'0375700269': 1, '0375727345': 1, '0375726403': 2, '0375713751': 1, '0375707972': 2,
'0375706771': 1, '0345460707': 1, '0345441133': 1, '0345438825': 1, '0345435249': 1,
'0345435168': 1, '0345433491': 1, '034543076X': 1, '0345430573': 1, '0345459202': 1,
'0345453743': 1, '0345452003': 1, '0345447840': 1, '0345445848': 1, '0345449347': 1,
'0373250479': 1, '0373218036': 1, '038549081X': 1, '0385490445': 1, '0385479565': 1,
'0385484518': 1, '0385424736': 1, '0385512104': 1, '0385511612': 1, '038550926X': 1,
'0385508042': 1, '0385504470': 1, '0385504209': 1, '0385503822': 1, '0385319568': 1,
'0385319207': 1, '038533303X': 1, '0385337639': 1, '038533656X': 1, '0385336179': 1,
'0385335830': 1, '0385335482': 1, '0394587618': 1, '0394551907': 1, '0394805720': 1,
'0385720106': 1, '0393046974': 1, '0385722206': 1, '0385722192': 1, '0394281802': 2,
'0393317552': 1, '038531258X': 1, '0380775840': 1, '0380774933': 1, '0380792893': 1,
'0380789019': 1, '038072152X': 1, '0380720132': 1, '0380710218': 1, '0380730847': 1,
'0380727501': 1, '0446607193': 1, '0380977303': 1, '0380973650': 1, '0385301499': 1,
'0385247923': 1, '0385199570': 1, '0380807343': 1, '0380805995': 1, '0380804697': 1,
'0380818973': 1, '0380820293': 1, '0380813815': 1, '068482499X': 1, '0684803356': 1,
'0749397543': 1, '0679746048': 1, '1573228214': 1, '0771055706': 1, '0771060548': 1,
'0864922299': 1, '0312099436': 1, '0312144075': 1, '0375705856': 1, '0156006391': 1,
'0060975547': 1, '0099394316': 1, '0385259956': 1, '0446678457': 1, '0375724834': 1,
'0671742515': 1, '0679723161': 1, '067976397X': 1, '0345427637': 1, '0060981180': 1,
'0440214041': 1, '0440211727': 1}

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
value_counts = df_no_selected_users['ISBN'].value_counts()
counts_users_books = {}
for user, books in user_books_dict.items():
    user_book_counts = value_counts[value_counts.index.isin(books)]
    counts_users_books[user] = user_book_counts.to_dict()
    #print(f"Items for {key}: {items}")
counts_users_books
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

```
{'15497': {'0140293248': 162},
 '23768': {'0316666343': 705,
  '0743418174': 234,
  '0060934417': 157,
  '0805063897': 133,
  '0380718340': 78,
  '0060929871': 68,
  '0312265859': 50,
  '1400031346': 46,
  '0671880187': 43,
  '0375413634': 40,
  '0684857820': 36,
  '038548951X': 33,
  '0060931418': 32,
  '0440204194': 28,
  '0670869902': 24,
  '039480029X': 18,
  '0394800133': 17,
  '0553109588': 14,
  '0394800206': 12,
  '0440226198': 11,
  '0060198125': 11,
  '0394800184': 10,
  '0671740504': 4,
  '0156505150': 2},
 '11676': {'0316666343': 705,
  '0971880107': 580,
  '0385504209': 486,
  '0312195516': 381,
  '059035342X': 312,
  '044023722X': 280,
  '067976402X': 254,
  '0786868716': 241,
  '0743418174': 234,
  '0345337662': 229,
  '0375727345': 228,
  '0312278586': 225,
  '0440226430': 212,
  '044021145X': 207,
  '0671003755': 206,
  '0446605239': 205,
  '0345370775': 199,
  '0385484518': 199,
  '0440241073': 199,
  '0345417623': 194,
  '0345361792': 180,
  '0446610038': 176,
  '0440234743': 168,
  '0060502258': 165,
  '0312291639': 165,
  '0385335482': 162,
  '0140293248': 162,
  '0316776963': 158,
  '0060987103': 154,
  '0842329129': 154,
  '0060938455': 153,
  '0375706771': 152,
  '0440221471': 151,
  '044651652X': 151,
  '0440236673': 149,
```

```
'0671510053': 149,
'0440213525': 144,
'0375726403': 143,
'0375707972': 143,
'0345313860': 142,
'0440222656': 139,
'0452282829': 138,
'0439139597': 136,
'0684872153': 135,
'0316096199': 132,
'0440225701': 132,
'0439136369': 132,
'0684874350': 131,
'014028009X': 130,
'0312305060': 128,
'0142000205': 128,
'0385722206': 128,
'0743237188': 126,
'0380789019': 125,
'0439064864': 125,
'068484477X': 123,
'0446608955': 123,
'0449005615': 119,
'0590353403': 118,
'0812550706': 116,
'0446364193': 115,
'0140298479': 112,
'0804114986': 111,
'0385720106': 110,
'0449212602': 110,
'0385503822': 106,
'0446612545': 103,
'0312966970': 102,
'0440224675': 100,
'0140067477': 100,
'0786881852': 99,
'080410753X': 97,
'0345378490': 97,
'0440226104': 97,
'155874262X': 94,
'0671041789': 93,
'0060987561': 93,
'0451526341': 91,
'0312995423': 90,
'034538475X': 89,
'0399144463': 89,
'067102423X': 89,
'0446532231': 88,
'051513287X': 88,
'0385508042': 88,
'0441003257': 86,
'0070212570': 85,
'0842329218': 85,
'080411109X': 83,
'014025448X': 82,
'0440180295': 82,
'0375702709': 82,
'0440211263': 79,
'0380718340': 78,
'0316969443': 78,
```

```
'0345391055': 77,
'0060175400': 77,
'0871136791': 76,
'0446606189': 76,
'002542730X': 76,
'0061015725': 76,
'0440998050': 76,
'0684848783': 76,
'0441569595': 76,
'0515132020': 75,
'031242227X': 75,
'0060391626': 74,
'0451156609': 74,
'0553277472': 74,
'0385511612': 73,
'039592720X': 72,
'0446604666': 72,
'0375703055': 72,
'0380018179': 72,
'044651862X': 71,
'0441790348': 71,
'0312243022': 71,
'0553582747': 71,
'0446608815': 71,
'0425163407': 71,
'0385479565': 71,
'0425167313': 70,
'0312983867': 69,
'0451169530': 69,
'0786885688': 69,
'1558745157': 69,
'0345378482': 69,
'0440215625': 69,
'0451172817': 69,
'0142004235': 68,
'0767905385': 68,
'0425169863': 68,
'0743225406': 68,
'0452283205': 67,
'0802130208': 67,
'0064400557': 67,
'0385336179': 67,
'0446604801': 67,
'0553279378': 67,
'0679731148': 66,
'038549081X': 66,
'0804111359': 66,
'0060958022': 65,
'1558744150': 65,
'0515128554': 64,
'0671014196': 64,
'0670892963': 62,
'0345435168': 62,
'0553579754': 61,
'0671042262': 61,
'0316780375': 61,
'0515135062': 60,
'038550926X': 60,
'0380813815': 60,
'0316781142': 59,
```

```
'0380710218': 59,
'0449219364': 59,
'034541389X': 59,
'0515132187': 59,
'0425182878': 58,
'0842329242': 58,
'0316693200': 58,
'0399150897': 57,
'0671793489': 56,
'1573229571': 55,
'0671693816': 55,
'055356451X': 55,
'0375502238': 55,
'0671011367': 55,
'0375412824': 54,
'0385512104': 53,
'0553279556': 53,
'0425192725': 53,
'1559029838': 53,
'055321313X': 52,
'1573227331': 52,
'0670880728': 52,
'0316693235': 52,
'0553582127': 51,
'014100018X': 51,
'0553284789': 51,
'0515133973': 51,
'0743467523': 50,
'0743411250': 50,
'0312265859': 50,
'0449006522': 50,
'0385335830': 50,
'0515136530': 50,
'0446611808': 49,
'0446675059': 49,
'0515136379': 49,
'1844262553': 49,
'0375760911': 49,
'0380727501': 49,
'0385424736': 49,
'044022103X': 49,
'0553582526': 49,
'0452282195': 49,
'0451169522': 48,
'0375504397': 48,
'014038572X': 48,
'0345445848': 48,
'0446519138': 48,
'0425177173': 48,
'0449223612': 48,
'0425107469': 47,
'0064407675': 47,
'0842342702': 47,
'0399146431': 47,
'0345413873': 47,
'0877017883': 47,
'0452268060': 47,
'0156007754': 46,
'1558745718': 46,
'0553289411': 46,
```

```
'0449221490': 46,
'042513525X': 46,
'0316168688': 46,
'0330367358': 46,
'0451207521': 46,
'0842329250': 46,
'1400031346': 46,
'0312263120': 45,
'0688177859': 45,
'0679735909': 45,
'0375400117': 45,
'0553213148': 45,
'0446605409': 45,
'0375412530': 44,
'0440204887': 44,
'0804105820': 44,
'0425170349': 44,
'0446604844': 44,
'0446527785': 44,
'037541309X': 43,
'0553573403': 43,
'0743206029': 43,
'0345447840': 43,
'0679442790': 43,
'0743460529': 42,
'0618002227': 42,
'0553576801': 42,
'0451188454': 41,
'0446365386': 41,
'0373218036': 41,
'0440940001': 41,
'1558531025': 41,
'0425122123': 41,
'0446391301': 41,
'0671042572': 41,
'0553569910': 41,
'0345368754': 41,
'0553213105': 40,
'0330375253': 40,
'0060929790': 40,
'0446607193': 40,
'0312274920': 40,
'0679432477': 39,
'0446525537': 39,
'0374199698': 39,
'0743469801': 39,
'0552143774': 39,
'006019491X': 39,
'0553569155': 39,
'0671568175': 39,
'0451209907': 39,
'0887307876': 39,
'014131088X': 39,
'0452283442': 39,
'0515133302': 39,
'0671011375': 39,
'0446525502': 39,
'0894808249': 39,
'042518286X': 38,
'0553289691': 38,
```

```
'0553582763': 38,
'0312199430': 38,
'0439404371': 38,
'0385490445': 38,
'0553210092': 38,
'0452281903': 38,
'0375756981': 38,
'0525946829': 38,
'0385199570': 38,
'0316603570': 38,
'0380730847': 38,
'0446611778': 38,
'044661193X': 38,
'0446609323': 37,
'0553258001': 37,
'0425155404': 37,
'0312983298': 37,
'0515135739': 37,
'074322535X': 37,
'0446603716': 37,
'0679751602': 37,
'055356160X': 36,
'0894803204': 36,
'0312982518': 36,
'0684857820': 36,
'067697175X': 36,
'0142002267': 36,
'1931561648': 36,
'067101420X': 36,
'0743437802': 36,
'0399148027': 35,
'0393317552': 35,
'0380820293': 35,
'0671888587': 35,
'0671036505': 35,
'0553582143': 35,
'0553802496': 35,
'0553560247': 35,
'0553380095': 35,
'0671867091': 35,
'0804108749': 35,
'0515127833': 35,
'0061000175': 35,
'0099245027': 34,
'0060008032': 34,
'0804115761': 34,
'037582233X': 34,
'0446360589': 34,
'0452279178': 34,
'0425121259': 34,
'0345459202': 34,
'051511992X': 34,
'0440127793': 33,
'051512608X': 33,
'0345449347': 33,
'038548951X': 33,
'0670844527': 33,
'0060512822': 33,
'0312924801': 33,
'0345435249': 33,
```

```
 '0316693286': 33,
 '042518630X': 32,
 '0425132048': 32,
 '0449203794': 32,
 '0446527165': 32,
 '0451190521': 32,
 '0671683993': 32,
 '0140185216': 32,
 '0749395990': 32,
 '052594527X': 32,
 '0515130966': 32,
 '0449213943': 31,
 '0451197747': 31,
 '0446611921': 31,
 '0743439740': 31,
 '0446679364': 31,
 '0671027573': 31,
 '0767902890': 31,
 '0061097861': 31,
 '038531258X': 31,
 '0385722192': 31,
 '0399148639': 31,
 '0440218535': 30,
 '0380807343': 30,
 '0425161242': 30,
 '0515122734': 30,
 '0553348981': 30,
 '0375406530': 30,
 '0446343455': 29,
 '0425189864': 29,
 '0446676098': 29,
 '0345441133': 28,
 '0060198133': 28,
 '0345433491': 28,
 '0449227421': 28,
 '0767902513': 28,
 '0441004520': 28,
 '0140143505': 27,
 '0553562738': 27,
 '0671789422': 27,
 '0446530891': 27,
 '0156007479': 27,
 '0553212451': 27,
 '0099747200': 27,
 '0312147015': 26,
 '055357230X': 26,
 '1551665301': 26,
 '0345389964': 26,
 '0312955006': 26,
 '0399146008': 26,
 '0385337639': 26,
 '0767903579': 26,
 '0345335465': 26,
 '0440236061': 25,
 '060980619X': 25,
 '0380973650': 25,
 '0425120279': 25,
 '0440212359': 25,
 '0446322180': 25,
 '0553569031': 25,
```

```
'0684835959': 25,
'0679775439': 25,
'055357227X': 25,
'0394281802': 24,
'0374270325': 24,
'0440207770': 24,
'0316734837': 24,
'0385504470': 24,
'0451191137': 24,
'0743454146': 24,
'0446600253': 24,
'0060256672': 24,
'0446527017': 23,
'0451180410': 23,
'0441172695': 23,
'0553348973': 23,
'0743262174': 23,
'087113795X': 23,
'0449910571': 23,
'0441009239': 23,
'0440225299': 23,
'0064407055': 23,
'0345425707': 23,
'0671705091': 23,
'0679721886': 22,
'038533656X': 22,
'0061092886': 22,
'0767914767': 22,
'039914840X': 22,
'0440237262': 22,
'0440216745': 22,
'048627263X': 22,
'0786861347': 22,
'014023828X': 22,
'1573222267': 21,
'0399148248': 21,
'0451403703': 21,
'0345404793': 21,
'067103619X': 21,
'0891073906': 21,
'0345423615': 21,
'0312979479': 21,
'0385319207': 21,
'0440176484': 21,
'0446600474': 21,
'0099263815': 21,
'0060988649': 21,
'0452281032': 21,
'0440201926': 20,
'0451454243': 20,
'0140280464': 20,
'0451205626': 20,
'0064400034': 20,
'0380818973': 20,
'0345404475': 20,
'0440207622': 20,
'0684841185': 20,
'0446603309': 20,
'0060977337': 20,
'0671003364': 20,
```

```
'0380005239': 20,
'042511984X': 20,
'0767912233': 20,
'0553573861': 20,
'0380804697': 20,
'0142004278': 19,
'0679776818': 19,
'0525945210': 19,
'0380805995': 19,
'0425157539': 19,
'034529873X': 19,
'0553382179': 19,
'0836218353': 19,
'0552142379': 19,
'0670851086': 19,
'0345346491': 19,
'0425098605': 19,
'1400030927': 18,
'0312977026': 18,
'1573220221': 18,
'0670891576': 18,
'0375758453': 18,
'0345438825': 18,
'0684856093': 18,
'0446611476': 18,
'0671020293': 18,
'0836218817': 17,
'0440193613': 17,
'0394800133': 17,
'076791404X': 17,
'0446353205': 17,
'0375422307': 17,
'1551668246': 17,
'0446609145': 17,
'0440235804': 17,
'0425192032': 17,
'0446606251': 17,
'0449217493': 17,
'0099521016': 17,
'044100069X': 17,
'0446601640': 17,
'0553572512': 16,
'0099464810': 16,
'0375713751': 16,
'044024126X': 16,
'0061099708': 16,
'1551668726': 16,
'0517703963': 16,
'0440209412': 16,
'0399144714': 16,
'0451210638': 16,
'042518904X': 16,
'0552546933': 16,
'038533303X': 16,
'051513239X': 16,
'0515107867': 16,
'0399135782': 16,
'0099604302': 16,
'0679777547': 16,
'0671873202': 16,
```

```
'0553583573': 16,
'0446608122': 15,
'0553280147': 15,
'0440225108': 15,
'0553096834': 15,
'043912042X': 15,
'067945960X': 15,
'0553076256': 15,
'0671873199': 15,
'0515130923': 15,
'0446608386': 15,
'034540047X': 15,
'0345453743': 15,
'0312978332': 15,
'0618129030': 14,
'0099410869': 14,
'0060915153': 14,
'0312985207': 14,
'0684832178': 14,
'0451204301': 14,
'0066212855': 14,
'0345430573': 14,
'0743225724': 14,
'0380619458': 14,
'0451207645': 14,
'0446612626': 14,
'0440225698': 14,
'0449205878': 14,
'0553109588': 14,
'0446612510': 14,
'0452281202': 14,
'0553580906': 14,
'0515128015': 14,
'0345305183': 14,
'0060970790': 14,
'0446519790': 14,
'014027927X': 14,
'0515128600': 13,
'0553382411': 13,
'0316107352': 13,
'0140278826': 13,
'0671644475': 13,
'0679425241': 13,
'0553297260': 13,
'0552998834': 13,
'0679776591': 13,
'0380977303': 13,
'0571197639': 13,
'0440295858': 13,
'0385301499': 13,
'0743229878': 13,
'0375411070': 13,
'0425176932': 13,
'0671793551': 13,
'0449216861': 13,
'0345412214': 13,
'0345311485': 13,
'0440173701': 13,
'0688167888': 12,
'0446356611': 12,
```

```
'0671014188': 12,
'0061097667': 12,
'1551667517': 12,
'0440215900': 12,
'0099244926': 12,
'0140042210': 12,
'0061098035': 12,
'0553575422': 12,
'0449219569': 12,
'0451092163': 12,
'0399146601': 12,
'0440224713': 12,
'0061099694': 12,
'0425092178': 12,
'0449208281': 12,
'0345402308': 12,
'0060504080': 12,
'0312890044': 12,
'0743411439': 12,
'0804119708': 12,
'0140167773': 12,
'0061092193': 12,
'0849932017': 12,
'0060972459': 12,
'0553582895': 12,
'0671014218': 12,
'0425189414': 12,
'0393046974': 12,
'059045367X': 12,
'0446602450': 12,
'0671477099': 12,
'0425179559': 12,
'0061099015': 12,
'0380775840': 12,
'0064471969': 11,
'0451163494': 11,
'044020352X': 11,
'0553272616': 11,
'0060172916': 11,
'0064400085': 11,
'0380774933': 11,
'0553280511': 11,
'0786866462': 11,
'0385319568': 11,
'0373250479': 11,
'0425195449': 11,
'0786867175': 11,
'0394551907': 11,
'0440146577': 11,
'0671732277': 11,
'0451179188': 11,
'0786889667': 11,
'0671007696': 11,
'0060002492': 11,
'0553281798': 11,
'034543076X': 11,
'0446356832': 11,
'0805054073': 11,
'0553571583': 11,
'0440486599': 11,
```

```
'0899193625': 11,
'0440216818': 11,
'0786867906': 11,
'0425144062': 10,
'0767908473': 10,
'0767905318': 10,
'0553569619': 10,
'0061093343': 10,
'0590629794': 10,
'1857978013': 10,
'1573220485': 10,
'034541005X': 10,
'0140279288': 10,
'0375700269': 10,
'0441865038': 10,
'0553583913': 10,
'0590494465': 10,
'1580082319': 10,
'0743418220': 10,
'0446364754': 10,
'0553587188': 10,
'0892967811': 10,
'0449149900': 10,
'0142000345': 10,
'0140620303': 10,
'0394800184': 10,
'038072152X': 10,
'0394805720': 10,
'0451628039': 10,
'0486270629': 10,
'0425187144': 10,
'1551667894': 10,
'0671670646': 10,
'0812502124': 9,
'0670894699': 9,
'0345260791': 8,
'0380542625': 8,
'0515118249': 8,
'006251279X': 8,
'0316898163': 7,
'0449207552': 7,
'0451177037': 7,
'0552141275': 6,
'0684865459': 6,
'0140620850': 6,
'0425099148': 5,
'0380720132': 5,
'0679441220': 5,
'0446356859': 5,
'0345452003': 5,
'0340401486': 5,
'0345340906': 5,
'0312925271': 5,
'0440507626': 4,
'006093395X': 4,
'0380700247': 4,
'0671722840': 4,
'0446609838': 4,
'0451410904': 4,
'0553273639': 4,
```

```
 '0515110132': 4,
 '0345460707': 4,
 '184195280X': 4,
 '0446671029': 4,
 '0312982402': 3,
 '0756400856': 3,
 '0553151657': 3,
 '0671454048': 3,
 '0152007822': 2,
 '0671750402': 2,
 '0380792893': 2,
 '0609603620': 2,
 '0515088757': 2,
 '0553227599': 2,
 '0307262073': 2,
 '0821749668': 2,
 '0590436031': 2,
 '0891455094': 1,
 '0312437285': 1,
 '0345421426': 1,
 '0875163831': 1,
 '0740734121': 1,
 '0974433926': 1,
 '0394900901': 1,
 '0312976860': 1,
 '0698113608': 1,
 '1576733165': 1,
 '0330376276': 1,
 '0385247923': 1,
 '0932653251': 1,
 '0743202759': 1,
 '0446370398': 1,
 '0312310617': 1},
'6242': {'0312195516': 381,
 '067976402X': 254,
 '0312291639': 165,
 '0140293248': 162,
 '0060934417': 157,
 '0060987103': 154,
 '0375726403': 143,
 '0375707972': 143,
 '068484477X': 123,
 '0679746048': 117,
 '0871136791': 76,
 '039592720X': 72,
 '0749397543': 71,
 '0312243022': 71,
 '0375705856': 66,
 '0060958022': 65,
 '1573227331': 52,
 '0312144075': 39,
 '1573228214': 38,
 '0771060548': 36,
 '0446678457': 31,
 '0394281802': 24,
 '0312099436': 21,
 '0684803356': 17,
 '0679777547': 16,
 '0771055706': 15,
 '0385259956': 14,
```

```
  '068482499X': 14,
  '0375724834': 13,
  '0156006391': 13,
  '0060975547': 13,
  '0099394316': 10},
 '6073': {'0440226430': 212,
  '0440211727': 193,
  '0440214041': 173,
  '0802130208': 67,
  '0679723161': 49,
  '0345427637': 47,
  '0679735909': 45,
  '0553213105': 40,
  '0553380095': 35,
  '067976397X': 33,
  '0767902890': 31,
  '0671742515': 25,
  '0345404475': 20,
  '0679776818': 19,
  '0060981180': 12}}
```

A large number of users (without including the selected users) have rated the same ISBN (book codes) as those chosen by the the selected users, across all the dataset. This ensures that the model has ratings for the user's ISBN from other user's ratings, thereby we reduce the risk of a "cold start" or uncertainty.

In [ ]:
```
test = {}
for key in counts_users_books:
    # Keep only the first three items for each key
    test[key] = dict(list(counts_users_books[key].items())[:5])
test
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
{'15497': {'0140293248': 162},
 '23768': {'0316666343': 705,
  '0743418174': 234,
  '0060934417': 157,
  '0805063897': 133,
  '0380718340': 78},
 '11676': {'0316666343': 705,
  '0971880107': 580,
  '0385504209': 486,
  '0312195516': 381,
  '059035342X': 312},
 '6242': {'0312195516': 381,
  '067976402X': 254,
  '0312291639': 165,
  '0140293248': 162,
  '0060934417': 157},
 '6073': {'0440226430': 212,
  '0440211727': 193,
  '0440214041': 173,
  '0802130208': 67,
  '0679723161': 49}}
```

In [ ]:
```
# We will remove the ratings of these users' books from our dataset.
# This allows us to later predict their ratings and compare them with the actual score
to_remove = {
    '15497': ['0140293248'],
    '23768': ['0316666343', '0743418174', '0060934417'],
    '11676': ['0316666343', '0971880107', '0385504209'],
    '6242': ['0312195516', '067976402X', '0312291639'],
    '6073': ['0440226430', '0440211727', '0440214041']}
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
#shows the rating of the users before remove its rows in df_similarity
def match_row(row):
    user_id = row['User-ID']
    isbn = row['ISBN']
    return user_id in to_remove and isbn in to_remove[user_id]

df_to_compare = df_similarity[df_similarity.apply(match_row, axis=1)]
df_to_compare
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	User-ID	ISBN	Book-Rating
1028	6073	0440226430	6
1029	6073	0440214041	5
1030	6073	0440211727	6
1065	6242	067976402X	8
1077	6242	0312195516	7
1079	6242	0312291639	5
2719	11676	0971880107	6
2843	11676	0316666343	5
3125	11676	0385504209	9
3910	15497	0140293248	6
5833	23768	0743418174	7
5834	23768	0060934417	8
5842	23768	0316666343	7

Remove some ISBN's of the selected user from the main dataset, to later use the trained SVD model to predict their rating score and compare to the actual ones.

In [ ]:
```
for user_id, isbns in to_remove.items():
    for isbn in isbns:
        mask = ~((df_similarity['User-ID'] == user_id) & (df_similarity['ISBN'] == isb
        df_similarity = df_similarity[mask]

# Reset index if needed
#df_similarity.reset_index(drop=True, inplace=True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
len(df_similarity)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
66069
```

In [ ]:
```
print(df['Book-Rating'].describe())
```

```
count    66082.000000
mean         7.781030
std          1.792506
min          1.000000
25%          7.000000
50%          8.000000
75%          9.000000
max         10.000000
Name: Book-Rating, dtype: float64
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

COLLABORATIVE FILTERING

Collaborative filtering faces challenges like scalability (the system's performance as the number of users and items grows) and sparsity (the issue of having too many items and not enough ratings). To tackle these, we are going to use Single Value Decomposition (SVD), which reduces the dimensionality of the problem and help in understanding the relationships between users and items by mapping them into a 'latent space'. This approach turns recommendation into an optimization problem, focusing on predicting ratings as accurately as possible.

In [ ]:
```
from surprise import Dataset, Reader, SVD, accuracy
from surprise.model_selection import cross_validate

# Assuming your dataset is in a DataFrame df with columns 'userid', 'isbn', 'rating'
reader = Reader(rating_scale=(1, 10))
```

```
data = Dataset.load_from_df(df_similarity[['User-ID', 'ISBN', 'Book-Rating']], reader)
#data.split(n_folds=5)
algo = SVD()
cross_validate(algo, data, measures=['RMSE', 'MAE'],cv=5,verbose=True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.6518  1.6396  1.6613  1.6342  1.6398  1.6454  0.0098
MAE (testset)     1.2841  1.2833  1.2865  1.2727  1.2702  1.2794  0.0066
Fit time          3.37    3.45    1.24    1.33    1.21    2.12    1.05
Test time         0.17    0.14    0.10    0.09    0.08    0.11    0.03
```

Out[ ]:
```
{'test_rmse': array([1.65183358, 1.63958403, 1.66132671, 1.63423684, 1.63980307]),
 'test_mae': array([1.28413436, 1.28331042, 1.28654852, 1.27268703, 1.27022058]),
 'fit_time': (3.366396903991699,
  3.4462506771087646,
  1.242121696472168,
  1.3259177207946777,
  1.206843376159668),
 'test_time': (0.1653432846069336,
  0.14249110221862793,
  0.09584164619445801,
  0.0865316390991211,
  0.08172607421875)}
```

Using SVD method in our dataset, the RMSE is around 1.65 points. Ideally this should be less than 1. However, the full dataset was reduced to 6600 samples due to computational limitations, ff the dataset has limited user rating score or if only a few users rate a book, SVD, like other collaborative filtering methods, struggles to predict the scores accurately. Therefore, it often bases its predictions around the mean, which can be somewhat distant from the actual values, as is the case here. However, an RMSE error prediction of 1.65, while not ideal, is not overly high.

The SVD model deviates from the predicted to the actual value by around 1.65 points. Ideally, the RMSE (Root Mean Square Error) value should be less than 1. Due to computational limitations, the full dataset of books and ratings scores was reduced for the other algorithms. Collaborative filtering models, including SVD, heavily rely on available data to make accurate recommendations to users. If the dataset has limited user rating score or if only a few users rate a book,SVD, like other collaborative filtering methods, struggles to predict the scores accurately. Therefore, it often bases its predictions around the mean, which can be somewhat distant from the actual values, as is the case here. However, an RMSE error prediction of 1.65, while not ideal, is not overly high. At the end of the notebook, we will compare the actual and predicted scores of some users using the SVD."

In [ ]:
```
#Take all data as trainset
dataset = data.build_full_trainset()
algo = SVD()
algo.fit(dataset)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7aebaf46b760>
```

In [ ]:
```python
# give user id, trained dataset, train dataset, df all user rating
def get_top_recommendations(user_id):
    # Convert raw user ID to internal user ID
    try:
        internal_user_id = dataset.to_inner_uid(str(user_id))
    except ValueError:
        print(f"User ID {user_id} not found in the dataset.")
        return []

    # Get the set of items (books) that the user has already rated
    internal_user_ratings = set([j for (j, _) in dataset.ur[internal_user_id]])
    user_ratings_isbn = set([dataset.to_raw_iid(iid) for iid in internal_user_ratings]

    # Get the set of all books the user hasn't rated
    all_books = set(df_similarity['ISBN'])
    unrated_books = all_books - user_ratings_isbn

    # Predict ratings for all unrated books
    predicted_ratings = []
    for book_isbn in unrated_books:
        # Predict rating for each unrated book
        prediction = algo.predict(str(user_id), book_isbn)
        predicted_ratings.append((book_isbn, prediction.est))

    # Sort the predictions by estimated rating in descending order
    predicted_ratings.sort(key=lambda x: x[1], reverse=True)

    top_10_recommendations = predicted_ratings[:10]
    return top_10_recommendations
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```python
user_id = 11676
#  '15497', '23768', '6242', 6073'
top_10_books = get_top_recommendations(user_id)

for isbn, rating in top_10_books:
    print(f"ISBN: {isbn}, Predicted Rating: {rating}")

#If the user in question has very few ratings, or if the items to be predicted have be
# the algorithm might struggle to find relevant neighbors, leading to average-based pr
```

```
ISBN: 0345339681, Predicted Rating: 10
ISBN: 0385504209, Predicted Rating: 9.990533014021535
ISBN: 006017143X, Predicted Rating: 9.874532193518874
ISBN: 0345339711, Predicted Rating: 9.591714460334407
ISBN: 0836218051, Predicted Rating: 9.580984765926663
ISBN: 0439136350, Predicted Rating: 9.53964606411394
ISBN: 1558744630, Predicted Rating: 9.533651543961536
ISBN: 0553270931, Predicted Rating: 9.50898651097769
ISBN: 0515134481, Predicted Rating: 9.507747849979072
ISBN: 0446301582, Predicted Rating: 9.507121930876439
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [ ]:
```
df_bookss = df_books
df_bookss = df_bookss.astype(str)
df_bookss.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCount	c
0	0884115631	The Shepherd	nan	Frederick Forsyth	nan	1960-06-01	A pilot and his plane are saved by a mysteriou...	0	
1	0060175648	Identity	A Novel	Milan Kundera	Harper	1998-04-21	Milan Kundera's Identity translated from the F...	176	
2	0394535383	Pillar of the Sky	A Novel	Cecelia Holland	Alfred a Knopf Incorporated	1985	Two thousand years ago, at the site of Stonehe...	534	
3	0843114401	Jingle Bear	nan	Stephen Cosgrove	nan	1985	When the other bears get ready to hibernate fo...	36	
4	0451207408	The Adventurer	nan	Jaclyn Reding	Signet Book	2002	While attempting to return a mysterious, legen...	314	

In [ ]:
```
#Actual rated book of user: 11676
a_user = df_similarity[df_similarity['User-ID'] == '11676']
a_user_books = df_bookss[df_bookss['isbn'].isin(a_user['ISBN'])]
a_user_books.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCour
40	0440176484	Secrets	A Novel	Danielle Steel	Dell	1986-10-01	Danielle Steel, America's number-one best-sell...	44
45	038548951X	Sister of My Heart	A Novel	Chitra Banerjee Divakaruni	Anchor	2000-01-18	From the award-winning author of Mistress of S...	34
70	0394900901	The Bears' Christmas	nan	Stan Berenstain, Jan Berenstain	Random House Books for Young Readers	1970	Father Bear tries to show his son how to use t...	5
71	0439083699	Franklin's Neighborhood	nan	Sharon Jennings, Paulette Bourgeois	Scholastic	1999	For a school project, Franklin is asked to dra...	3
129	0842342702	Left Behind	A Novel of the Earth's Last Days	Tim LaHaye, Jerry B. Jenkins	nan	1998	After millions of people around the world vani...	36

In [ ]:

```
# Extract ISBNs from the recommendations
recommended_isbns = [isbn for isbn, _ in top_10_books]

# Filter the books DataFrame to only include recommended books
recommended_books_df = df_books[df_books['isbn'].isin(recommended_isbns)]

recommended_books_df
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCo
196	0553270931	Still Life with Woodpecker	NaN	Tom Robbins	Bantam	1980	A sort of love story revealing the purpose of ...	
5054	006017143X	The Night Listener	A Novel	Armistead Maupin	Harper	2000-09-19	Here is the much-anticipated new work from one...	
5189	0515134481	Circus of the Damned	An Anita Blake, Vampire Hunter Novel	Laurell K. Hamilton	Penguin	2002-09-24	When a powerful centuries-old vampire hits Ani...	
5257	0345339711	The Two Towers	The Lord of the Rings: Part Two	J.R.R. Tolkien	Del Rey	1986-08-12	The middle novel in The Lord of the Rings—the ...	
5598	0446301582	Word of Honor	NaN	Nelson DeMille	Grand Central Publishing	1987-03-01	Read the gripping story of a Vietnam vet whose...	
5784	0345339681	The Hobbit	The Enchanting Prelude to The Lord of the Rings	J.R.R. Tolkien	Del Rey	1986-07-12	The stirring adventure that begins The Lord of...	
5832	0836218051	The Essential Calvin And Hobbes	NaN	Bill Watterson	Andrews McMeel Publishing	1988	The Essential Calvin and Hobbes is an over-siz...	
5961	0385504209	The Da Vinci Code	NaN	Dan Brown	Doubleday	2003	Harvard symbologist Robert Langdon and French ...	
6008	0439136350	Harry Potter	The Complete Series	J. K. Rowling	NaN	NaN	During his third year at Hogwarts School for w...	
6120	1558744630	Chicken Soup for the Teenage Soul	101 Stories of Life, Love and Learning	Jack Canfield, Mark Victor	Chicken Soup for the Soul	1997	101 stories of life, love, and learning.	

	isbn	title	subtitle	authors	publisher	publishedDate	description	pageCc
				Hansen, Kimberly Ki...				

User 11676 rated multiple books, recommendated many fiction books, the model also recommends many multiple books

Selected users: Predict rating score

In [ ]:
```
predicted_ratings = []

# Iterate over each row in the DataFrame
for index, row in df_to_compare.iterrows():
    user_id = row['User-ID']
    isbn = row['ISBN']

    # Predict the rating
    prediction = algo.predict(str(user_id), str(isbn))
    predicted_ratings.append(prediction.est)

df_to_compare['predicted bookRating'] = predicted_ratings

df_to_compare
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarnin
g: `should_run_async` will not call `transform_cell` automatically in the future. Ple
ase pass the result to `transformed_cell` argument and any exception that happen duri
ng thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[ ]:

	User-ID	ISBN	Book-Rating	predicted bookRating
1028	6073	0440226430	6	8.173271
1029	6073	0440214041	5	8.620667
1030	6073	0440211727	6	8.889222
1065	6242	067976402X	8	6.028799
1077	6242	0312195516	7	5.582681
1079	6242	0312291639	5	5.886632
2719	11676	0971880107	6	5.268219
2843	11676	0316666343	5	9.479094
3125	11676	0385504209	9	9.990533
3910	15497	0140293248	6	6.956833
5833	23768	0743418174	7	7.710979
5834	23768	0060934417	8	7.898089
5842	23768	0316666343	7	7.542955

Apart from the RMSE value, we analysed the rating score on a sample users. The selected user were estracted from the dataset to predict their rating scores. We found differences between actual and predicted book valuations. Some predictions were quite close, as in the case of User 23768 (actual 8, predicted 7.90), but others were not, as in the case of User 6073 (actual 5, predicted 8.62). This shows that our model can guess user preferences to some extent, but does not always get it right.

In [ ]: