14/1/2024

Project Report: Kindle Books Dataset

Lesson:

Business Intelligence and Big Data Analysis

Supervising Lecturer:

Damianos Hadjiantoniou

Group: 5

Team Members:

Artopoulos Georgios (8200016)

Panagiotis-Alexios Spanakis (8200158)

Content

1. Dataset description

For the dataset to be analyzed we chose the Amazon Kindle Books Dataset 2023 from the Kaggle platform, which dataset contains data for 130,000 Kindle e-books. The publicly available data was collected in October 2023 from Amazon's website https://www.amazon.com/kindle-books/ . The data was collected by asaniczka and is licensed under the ODC Attribution License (ODC-By).

More specifically, the dataset consists of a csv file called kindle_data-v2.csv, size 37.5 MB. This file contains about 130,000 book entries (specifically 133,102) and contains 16 columns for their description. First, we have the columns describing the basic properties of the book, namely the **asin** column, which is the Product ID from Amazon, **title**, **author, soldBy**, which lists the Seller(s) of the book. Next, we are given two columns which contain hyperlinks to the Amazon site, the **imgUrl** column**, which** contains the url of the book cover image and the **productURL** column**, which** contains the url of the book and the column.

Also given are the columns **stars,** which contains the average rating of the book, **reviews,** which contains the number of reviews and the **price** column. It is worth noting that for these columns, if they have a value of **0** it means that either no data were found for them, or they could not be retrieved. Furthermore, the columns **isBestSeller, isKindleUnlimited, isGoodReadsChoice** and **isEditorsPick** are given which describe the status of the book (Boolean value) as to whether it was Best Seller, whether it was available through Kindle Unlimited, whether it was GoodReadsChoice and whether it was Editor's Pick respectively. The **publishedDate** column is also included, which contains the publication date of the book. Finally, the **category_name** and **category_id** columns are given, which contains the serial identifier assigned to the category to which this book belongs.

The columns in order:
asin, title, author, soldBy, imgUrl, productURL, stars, reviews, price, isKindleUnlimited, category_id, isBestSeller, isEditorsPick, isGoodReadsChoice, publishedDate, category_name

2. ETL process

2.1 Extract

In order to create a Data Warehouse containing the information from the reported dataset, we performed a series of tasks. These activities included extracting data from the dataset, eliminating inaccurate or duplicate records, and loading the data into Microsoft SQL Server, adhering to the Star Schema structure we designed for the Data Warehouse. This entire process, including the ETL (Extract-Transform-Load) stages, was created using MSSQL Integration Services within Visual Studio 2022 and SQL Server Management Studio 2022.

The first step of the ETL process is to empty the temporary table staging_v2 to which we assign our primary data and thus this table contains all the columns of our file, with the exception of the columns **imgUrl, productURL** and **category_id** as these columns will not be useful for our analysis. To avoid unicode character formatting problems, for the columns that contain strings, we chose to convert them from the simple varchar type to **nvarchar,** which type allows sql to format unicode characters. The rest of the table data columns remained with the same type as the given file (e.g. the **stars** column remained type **float**). This step is necessary in order to refresh the Data Warehouse data and keep it up-to-date without creating technical problems.

In the second phase, we edited the given file, using the semicolon character (";") as a delimeter. This deliberate choice of delimeter was done to proactively address potential problems when reading the files. These issues mainly stemmed from incorrect assignments and the presence of unknown characters and symbols due to differences in character formatting from different languages, such as French (e.g. French accents) and Chinese, which we sought to correct before importing the data into the SQL database.

The file read has the staging_v2 table as the destination and we used SQL Server DB Destination for Connection Manager.

2.2 Transform

2.2.1 Data cleansing

Having assigned our data from the csv file to the staging_v2 table on our server, we proceed to clean it so that we can then use it more easily and efficiently in our analysis.

First, we define a transaction, which ensures that all subsequent actions are executed as a coherent unit, where this approach guarantees data integrity by allowing all changes to be reverted in case a step fails to execute correctly. The specific transaction in the workflow is called **Clean Invalid Values** which, firstly deletes rows where the title is exactly 'Not Found', removing entries with substitute titles or irrelevant titles and secondly eliminates rows where the author is null and the title contains 'USER GUIDE', as books with this combination were observed to have an invalid asin and when we tried to find them in the Amazon store our request was invalid. Finally, it updates rows with null author fields by setting them to 'Other'. This standardization replaces missing data with a consistent position holder, enhancing data uniformity.

Once the above is completed, we proceed to the process of cleaning up the duplicates. During our initial data preparation, we first create a temporary table, #main_temp, into which we copy all the data from staging_v2. We then focus on identifying duplicates by creating another temporary table, #duplicates. This table stores the rows from #main_temp that appear more than once, based on author and title. We use a window function `COUNT(*) OVER` to count these duplicates.

We then proceed to deal with special cases that have a duplicate_count greater than 2. We start by handling cases for duplicate_count equal to 4 and give each book duplicate_count=2 based on the value of the soldBy column. To handle the case with duplicate_count equal to 5 we delete from #duplicates the case with published_date equal to null and give the duplicate_count of the remaining rows a value of 2 based on soldBy. Finally, we deal with cases where there is a duplicate_count equal to 3. If all three are from the same soldBy, we add Re-Edition I and Re-Edition II to the titles of the most recent books, if there are two books in the trio with the same soldBy we add Re-Edition I, and if all three soldBy are different then we make no change. Before the iteration starts, we read from the #duplicates table all values with duplicate_count set to 3.

The copy processing loop starts after the control variable is declared. Within this loop, we handle different scenarios: when both soldBy fields are NULL, we keep the row with the most reviews and stars; when one soldBy is NULL and the other is not, we remove the row with NULL; and when neither soldBy is NULL, we compare other attributes such as publishedDate to determine if and which copy to modify by adding "Re-Edition" to its title. After processing each set of duplicates, we delete the processed lines from #duplicates and #TempResults and check for more duplicates to process.

In our last steps, we update staging_v2 by deleting all its data and then import the cleaned and updated data from #main_temp. We also update the rows in staging_v2 where soldBy is NULL to 'Other'. Finally, we clean up by deleting all the temporary tables we used in the process.

The final process in our workflow is the process called **Merge Similar Names for Sellers**, where the goal is to unify the various representations of seller names (**soldBy**) into standardized formats. Essentially, this process normalizes Amazon, Penguin Random House, HarperCollins, Simon & Schuster and Macmillan sellers, as there were many records where the sellers in those cases were common (e.g. Amazon.com and Amazon LLC).

2.2.2 Creating Dimensions

After cleaning the data, the next step is to create the dimensions to form the Star Schema. More specifically, five dimensions were created: **SoldBy, Author, Category, PDate** , **Book** and the corresponding tables in the database (soldBy_dim, author_dim, category_dim ,pdate_dim, book_dim).

More specifically, the **soldBy_dim** table has columns **id_soldBy** as the autoincremented primary key and **label_seller** containing the contents of the soldBy column of the staging_v2 table. Similarly, the **author_dim** table has columns id_author and **label_author** (content from the author column of the staging_v2 table) and the **category_dim** table has columns **id_category** and **label_category.** The **pdate_dim** table has columns **id_pdate** as the autoincremented primary key, the **label_publishedDate** column (of

type **Date**) which has as its content that of the publishedDate column of the staging_v2 table, the **label_year** column (of type **int**) which contains the time fragment from the publishedDate column of the staging_v2 table, **label_month** (of type int) containing the month part from the publishedDate column, **label_decade** (of type int) containing the decade part from the publishedDate column calculated as (YEAR(publishedDate) / 10) * 10 and the **label_monthName** column containing the name of the month (e.g. January) from the publishedDate column. Finally, the **book_dim** table has columns **id_book seller** as an autoincremented primary key, **label_asin, label_title , label_reviews** (type **int), label_isKindleUnlimited, label_isEditorsPick, label_isBestSeller** and **label_isGoodReadsChoice** (all columns containing the prefix **is** are of type **Boolean**) , where the above columns except the primary key are derived from the corresponding columns in the staging_v2 table.

2.2.3 Creating a Fact Table

Having the necessary dimensions ready, we are able to create the Fact Table. This creates the rbook_fact table in our database, which has as columns the **Book** column , which is a foreign key of the **id_book** of the book_dim table, the **Author** column , which is a foreign key of the **id_author** of the author_dim table, the **Category** column , which is a foreign key of the **id_category** of the category_dim table, the column **PDate** , which is a foreign key of the **id_pdate** of the table pdate_dim , the column **SoldBy**, which is a foreign key of the **id_soldBy** of the table soldBy_dim and, finally, the columns **Rating and Price,** which contain the values of the columns **stars** and **price** from the table staging_v2.

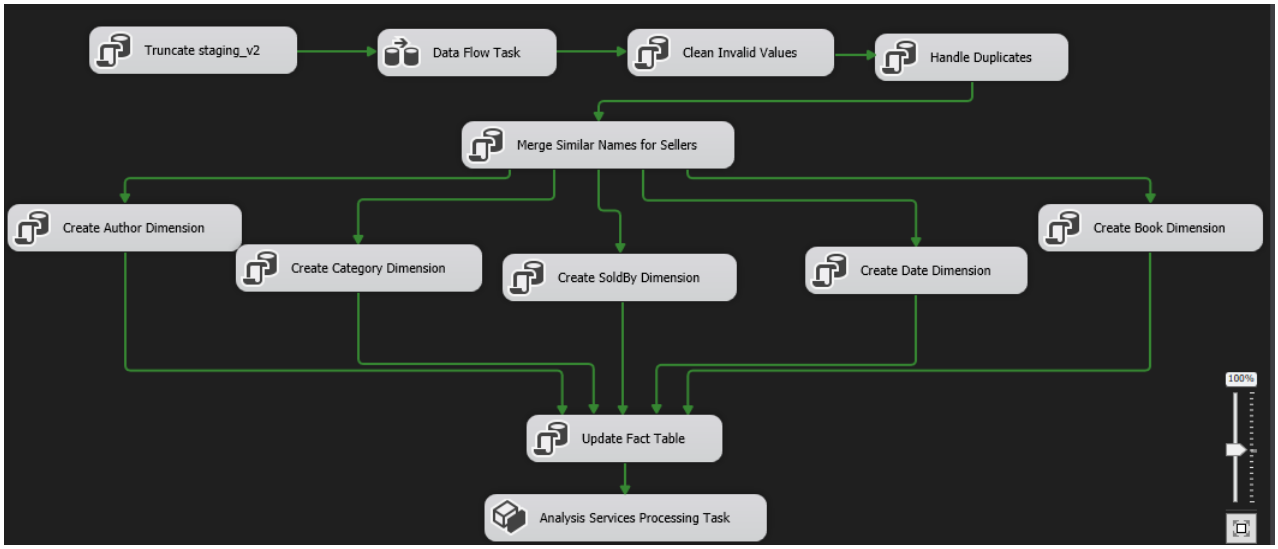
2.3 Load

With all the tables we will need to create the cube ready, we proceed to load the data we need from the staging_v2 table. Filling is accomplished by matching the values of the staging_v2 table and the primary keys from our dimensions. Thus, finally the fact table consists of 83701 records, 5 columns with foreign keys of the dimensions and two columns showing the Rating and Price metrics corresponding to this record.

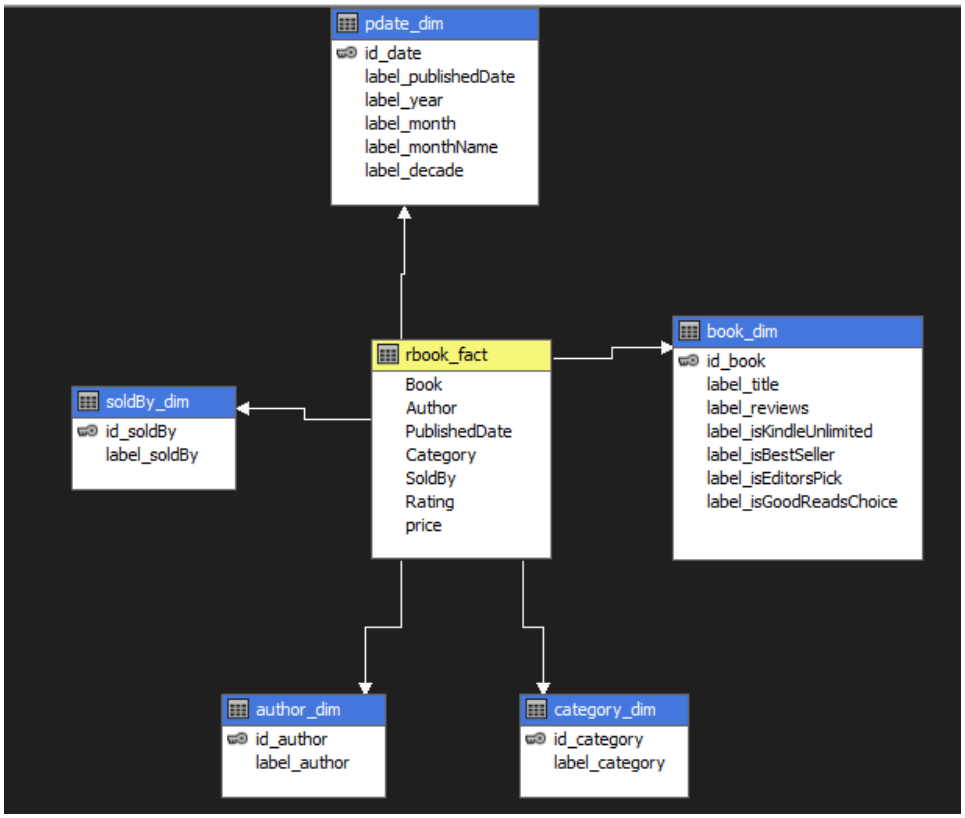
The final version of the Fact Table is illustrated as follows:

	Book	Author	PublishedDate	Category	SoldBy	Rating	price
1	6270	59326	4623	1	22	4,5	34,1699981689453
2	132192	51363	4143	1	22	4,30000019073486	6,98999977111816
3	52157	66762	4812	1	4	4,59999990463257	13,9899997711182
4	98603	65097	4587	1	22	3,29999995231628	64,5999984741211
5	67173	16341	5561	1	22	4,30000019073486	15,3900003433228

The ETL workflow that was eventually created is shown as follows (the last step is automatically processing the cube):



3. Star Scheme



The above image shows the Star Schema that was implemented, which consists of the fact table which has the Rating of each book as its basic metric. Each **rating is** accompanied by the **book** it is about, the **publisher** who wrote the book, the **date the** book was published, the **category the** book belongs to and the **seller of** the book, which are the dimensions for the rating. The rating is a metric which takes values (float type) from 0 to 5 and the price which takes real values (float type).

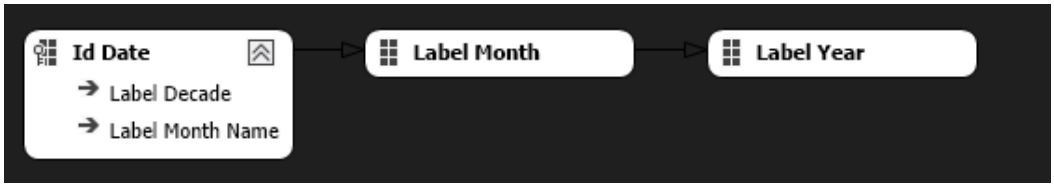
4. Cube

To create and process the cube for the Data Warehouse created, Visual Studio was used with the Multidimensional Analysis package offered by extension analysis services.

After creating a MultiDimensional Analysis Project, first, we defined the Data Source as the database used during the ETL process, where the Fact Table and its dimensions are located. Next, the Data Source View was created by loading the fact table and its associated tables, i.e. the dimension tables. Then, we proceeded to create a cube from the Data Source View by defining the Fact Table **rbook_fact** as the measure group. We kept the Rating Score and the automatically created Count metric as the metrics of the cube. As dimensions we defined the Book, Author, Seller, PDate and Category tables.

After the necessary dimensions were successfully defined, it was ensured that for each dimension, in addition to its primary key, the other features of the cube were present in order to interpret the results of the cube's calculations.

After initializing the dimensions, we decided to create a **natural** hierarchy for the published date within its dimension (**pdate_dim**). This hierarchy was intentionally created as besides optimizing querying over dates, which is sacrificed by creating an unnatural hierarchy, we are also provided with the possibility to drill-down and roll-up over published dates (e.g. for a particular year we can see data for the months of that year and vice versa). This hierarchy is depicted as follows:

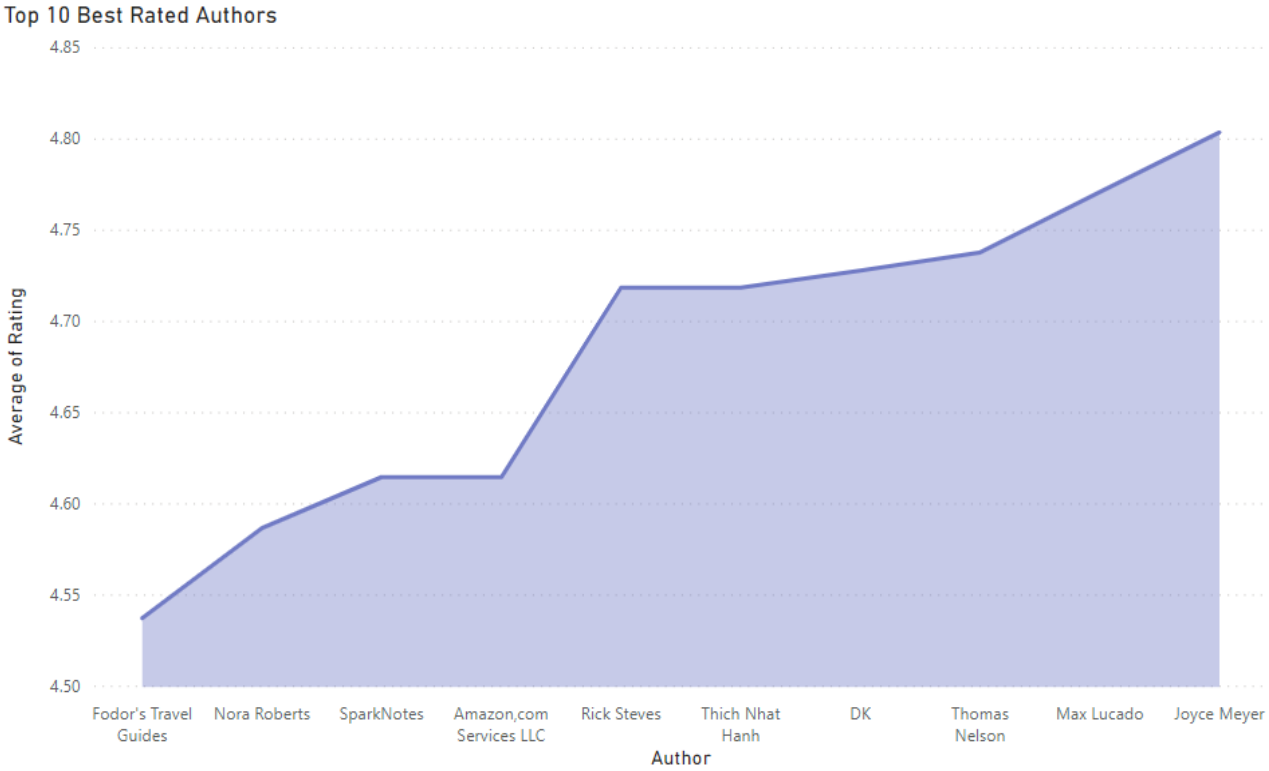


Having finished with the dimensional processing and after the successful cube process, we created a new metric. A useful measure that we want to know for our data of interest is the average book ratings, which we created via the Calculations module by doing the operation [Measures].[Rating]/[Measures].[Fact Table Ratings Count]. Furthermore, a useful measure is the average of book prices, which was also created by via the Calculations module by doing the operation [Measures].[Price]/[Measures].[Rbook Fact Count]. Then, again we made process our cube in order to use it in creating useful visualisations.

5. Visualizations via Power BI

Having extracted the data from the cube, we start the process of visualizing the information received. After loading the cube into PowerBI the following diagrams are produced.

Top 10 Best Rated Authors



To get the 10 best rated authors, we use the **label_author** from the **author_dim** dimension on the X-axis and the **Rating** with **Average** function from the fact table **rbook_fact** on the Y-axis.

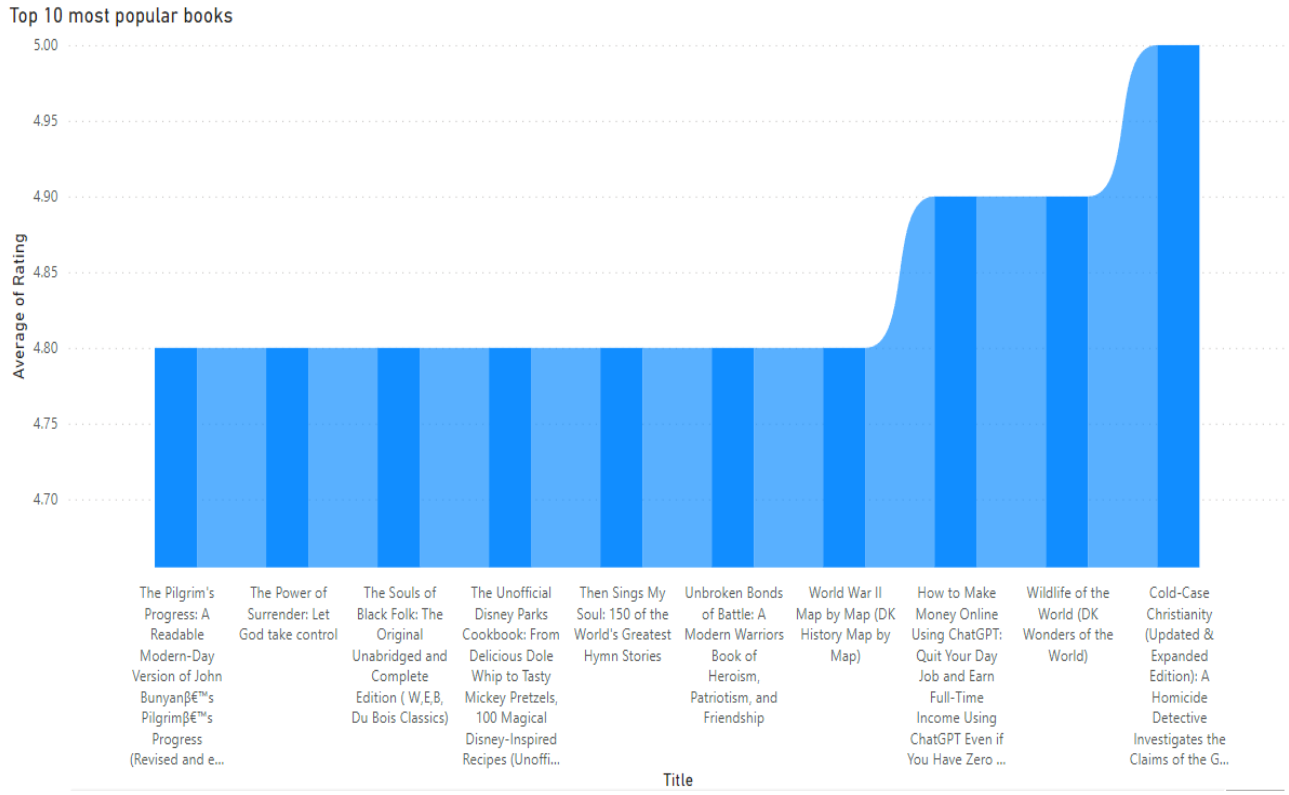
In addition, to achieve our goal we use the following filters:

Avarage Rating > 4.5: Using the **Rating** variable with the **Avarage** function we only consider authors who have an average rating above 4.5.

Count of **book > 20:** Using **Book** with the **count** function from the fact table **rbook_fact** we only consider authors who have more than 20 books.

label_reviews > 100: Using the **label_reviews** variable from the **book_dim** dimension we only consider books by each author that have more than 100 reviews.

Top 10 most popular books



To get the 10 most popular books we used the **label_title** from the **book_dim** dimension on the X-axis and the **Rating** with the **Avarage** function from the fact table **rbook_fact** on the Y-axis.

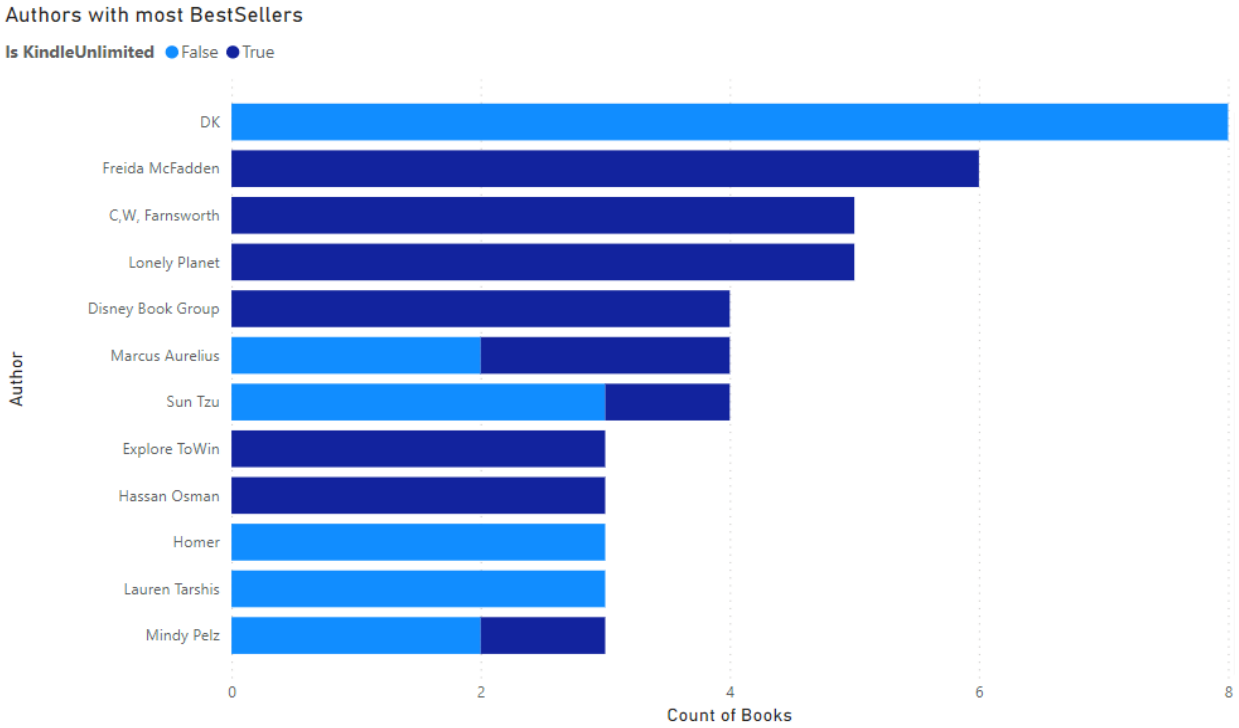
The filters used to find the ideal results are:

Avarage Rating > 4.5: Using the **Rating** column with the **Avarage** function we only take into account books that have over 4.5 average rating (as there are books with the same title that are sold by more than one Seller)

label_reviews > 450: Using the **label_reviews** variable from the **book_dim** dimension we only consider books that have more than 450 reviews.

label_isBestSeller is True: Using the **label_isBestSeller** variable from the **book_dim** dimension we only take into account books that are best sellers.

Authors with the most Best Sellers



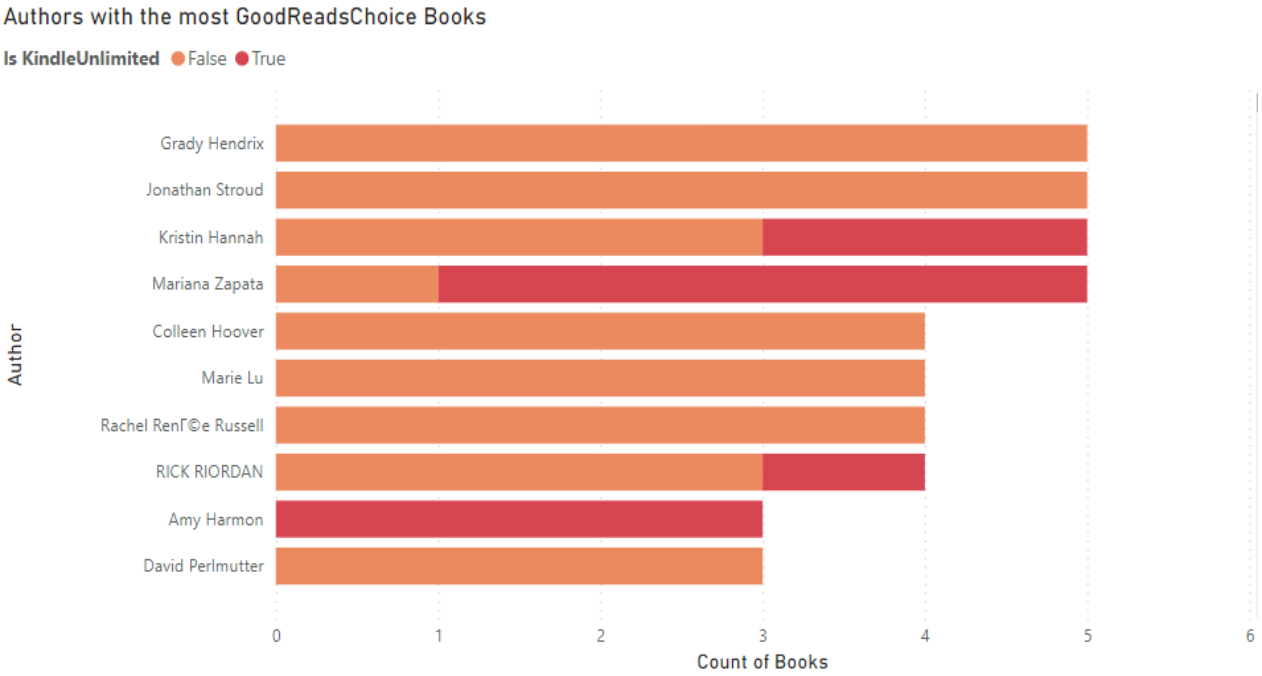
To find the authors with the most bestselling books we used **Book** with the **count** function from the fact table **rbook_fact** on the X-axis and **label_author** from the **author_dim** dimension on the Y-axis.

In addition, we used the following filters:

label_isBestSeller is True: Using the **label_isBestSeller** variable from the **book_dim** dimension we only take into account books that are best sellers.

label_isKindleUnlimited: using the variable **label_ isKindleUnlimited** from the dimension **book_dim** we show in the diagram which of the books of each author are available through the Kindle Unlimited program.

Authors with the most GoodReads Choice Books



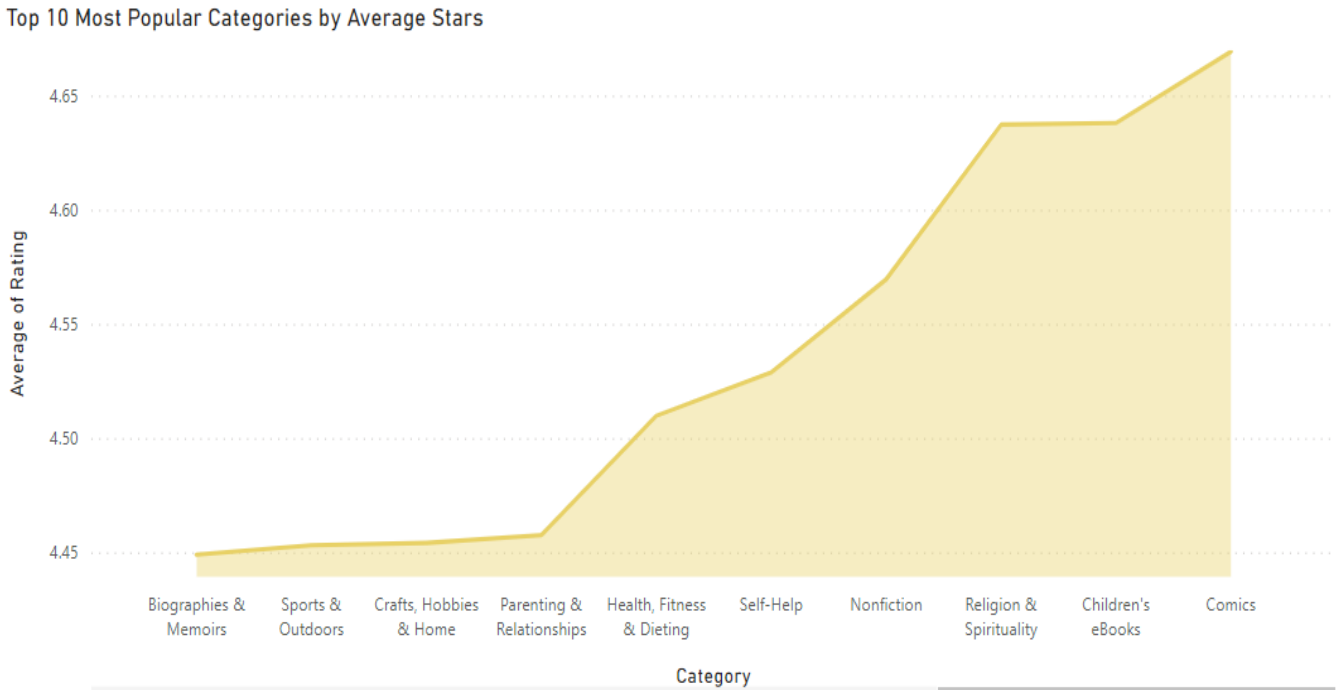
To find the authors with the most GoodReads Choice award-winning books we used **Book** with the **count** function from the fact table **rbook_fact** on the X-axis and **label_author** from the **author_dim** dimension on the Y-axis.

In addition, we also utilized the following filters:

label_isGoodReadsChoice is True: Using the variable **label_is GoodReadsChoice** from the dimension **book_dim** we only take into account the books which are awarded with GoodReads Choice

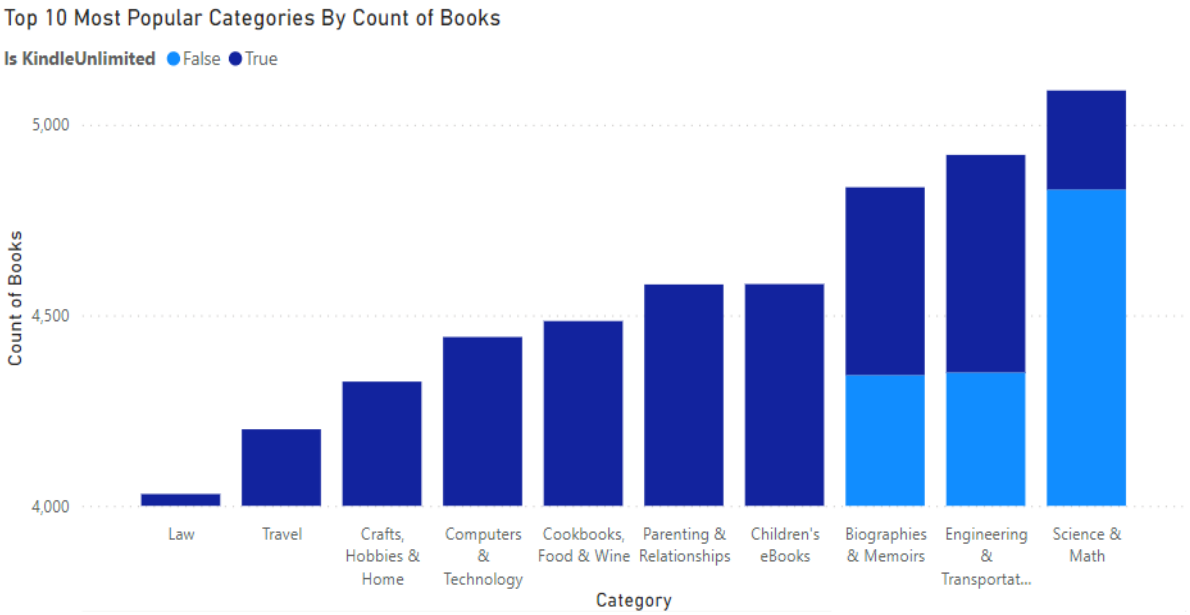
label_isKindleUnlimited: using the variable **label_ isKindleUnlimited** from the dimension **book_dim** we show in the diagram which of the books of each author are available through the Kindle Unlimited program.

Top 10 most popular categories by Average Stars



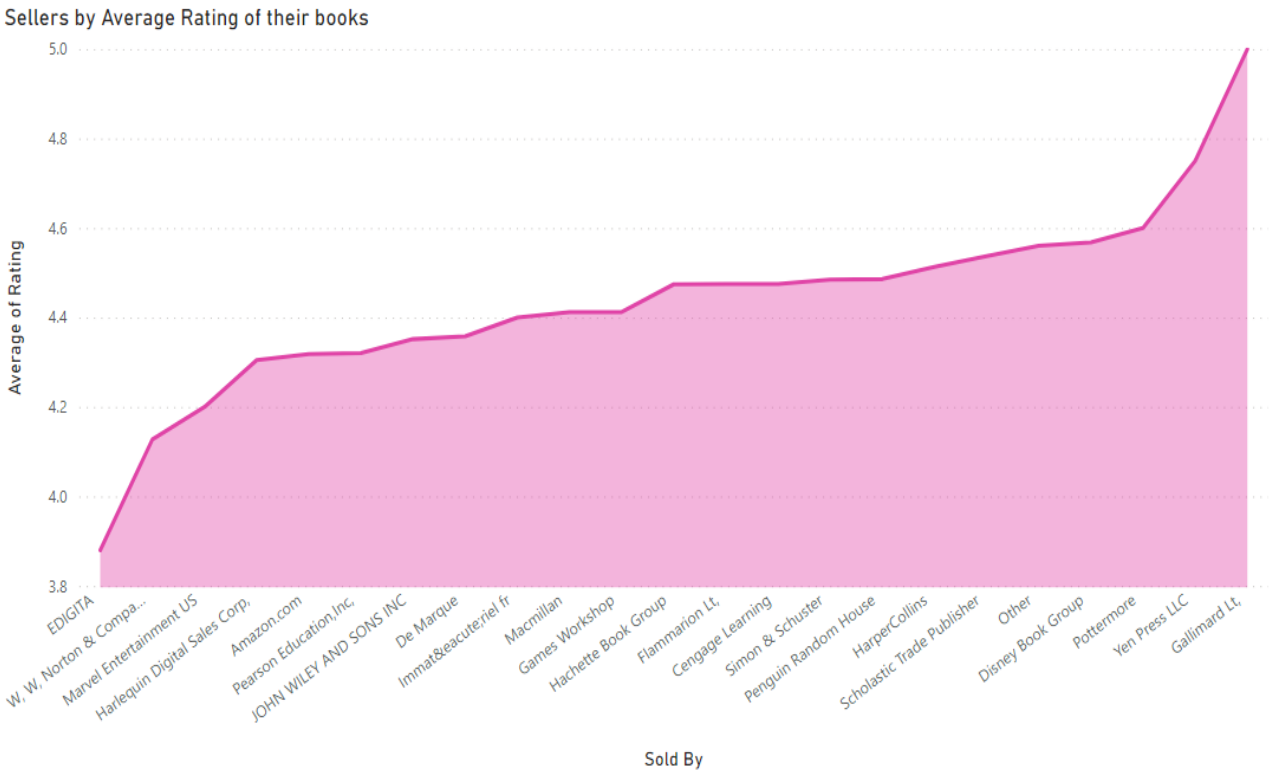
To get the top 10 best rated categories we used the **label_category** from the **category_dim** dimension on the X axis, the **Rating** with the **Avarage** function from the fact table **rbook_fact** on the Y axis and displayed the top 10 based on Average Rating.

Top 10 most popular categories by Count of Books



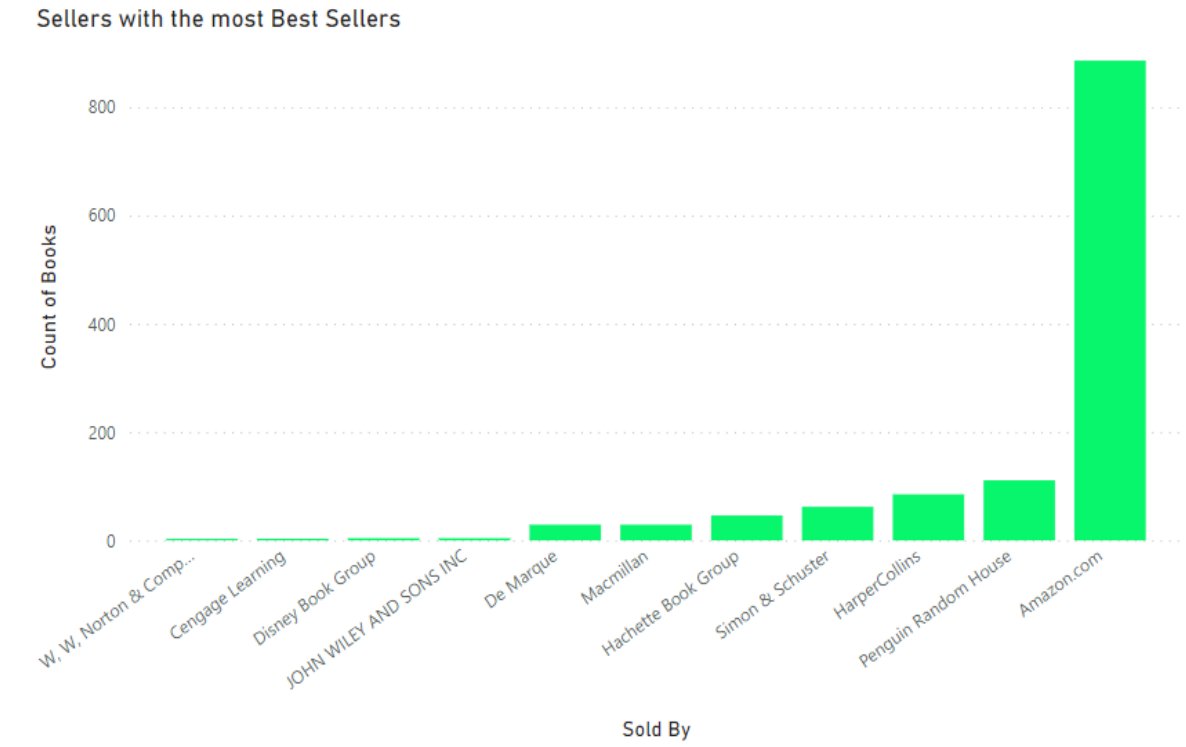
To get the top 10 best rated categories we used **label_category** from the **category_dim** dimension on the X axis, **Book** with the **count** function from the fact table **rbook_fact** on the Y axis and displayed the top 10 based on Average Rating.

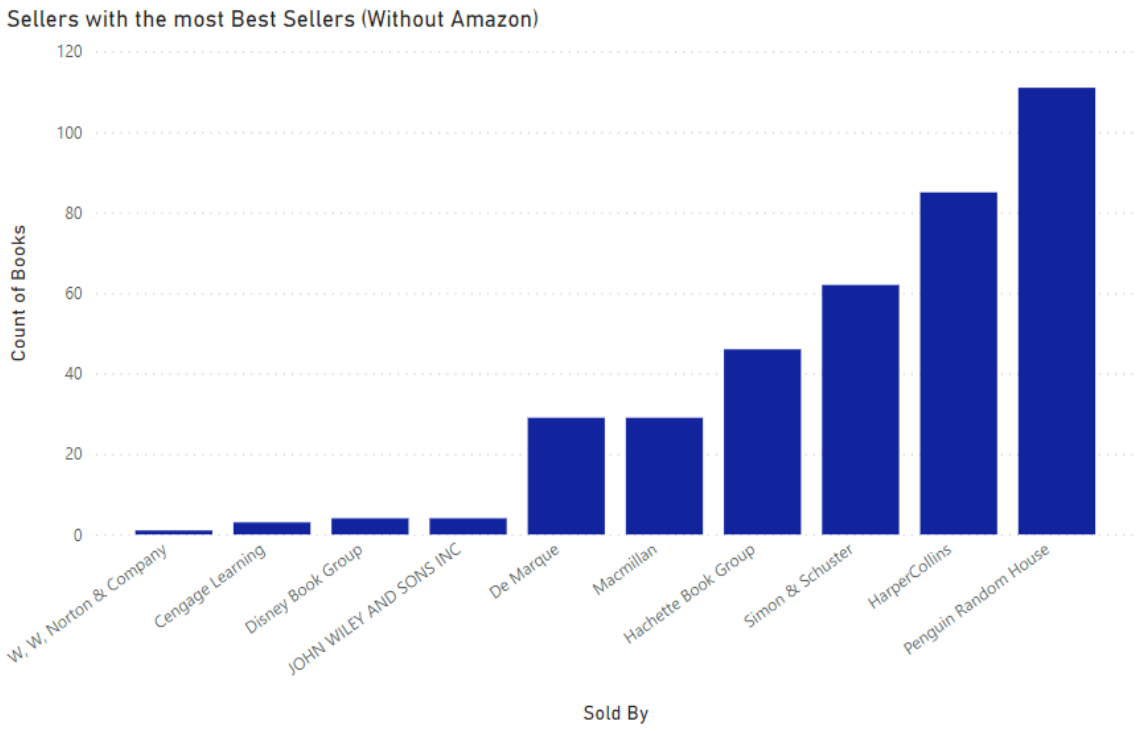
Sellers by Average Rating of their books



To get the ranking of the sellers based on their average rating, we used the **label_soldBy** from the **soldBy_dim** dimension on the X-axis, the **Rating** with the **Avarage** function from the fact table **rbook_fact** on the Y-axis and displayed the sellers by sorting from the lowest to the highest Average Ratings values.

Sellers with the most Best Sellers (With and Without Amazon)





To get the ranking of sellers based on the number of best sellers, we used the **label_soldBy** from the **soldBy_dim** dimension on the X-axis, the variable **Book** with the **count** function

from the fact table **book-fact** on the Y-axis and displayed the sellers by sorting from least to greatest the number of books of each seller.

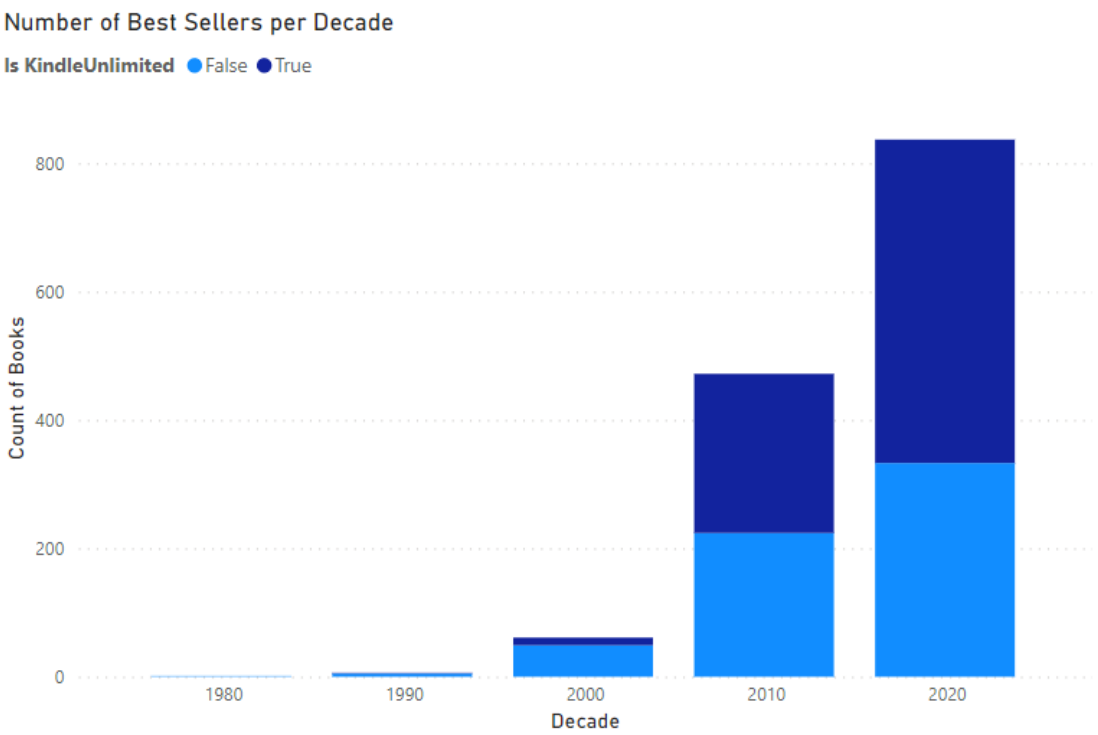
In addition, we used the following filters:

label_isBestSeller is True: Using the **label_isBestSeller** variable from the **book_dim** dimension we only take into account books that are best sellers.

Label_soldBy is not Other: by excluding the value other from the axis we get better information about the confirmed sellers.

Label_soldBy is not Amazon (In the second image only): by excluding the Amazon price from the axis we get better information about the other sellers as the kindle is owned by the company and has the largest number of books on it.

Number of Best Sellers per Decade



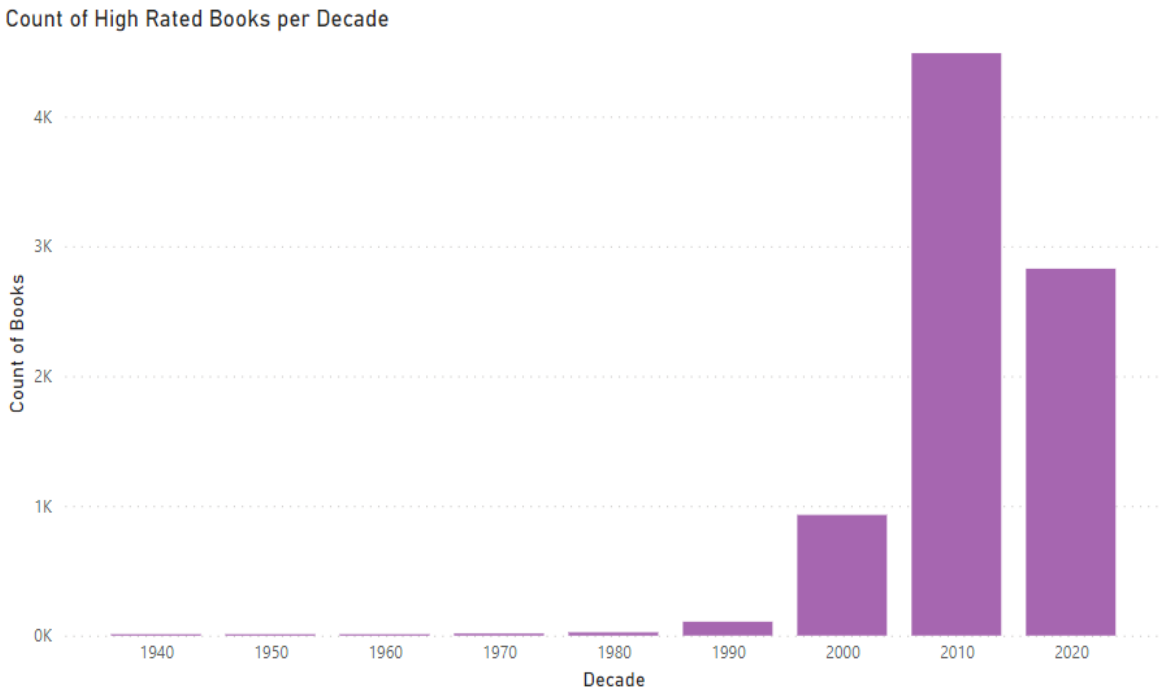
To find the number of Best Seller books per decade we used the **label_decade** column from the **pdate_dim** dimension on the X-axis, the **Book** column with the **count** function from the **book-fact** fact table on the Y-axis and displayed both axes in ascending order.

The following filters were also necessary:

label_isBestSeller is True: Using the variable **label_isBestSeller** from the **book_dim** dimension we only take into account books that are best sellers.

label_isKindleUnlimited: using the variable **label_ isKindleUnlimited** from the dimension **book_dim** we show in the diagram the percentage of books per decade that are available through the Kindle Unlimited program.

Count of High Rated Books per Decade



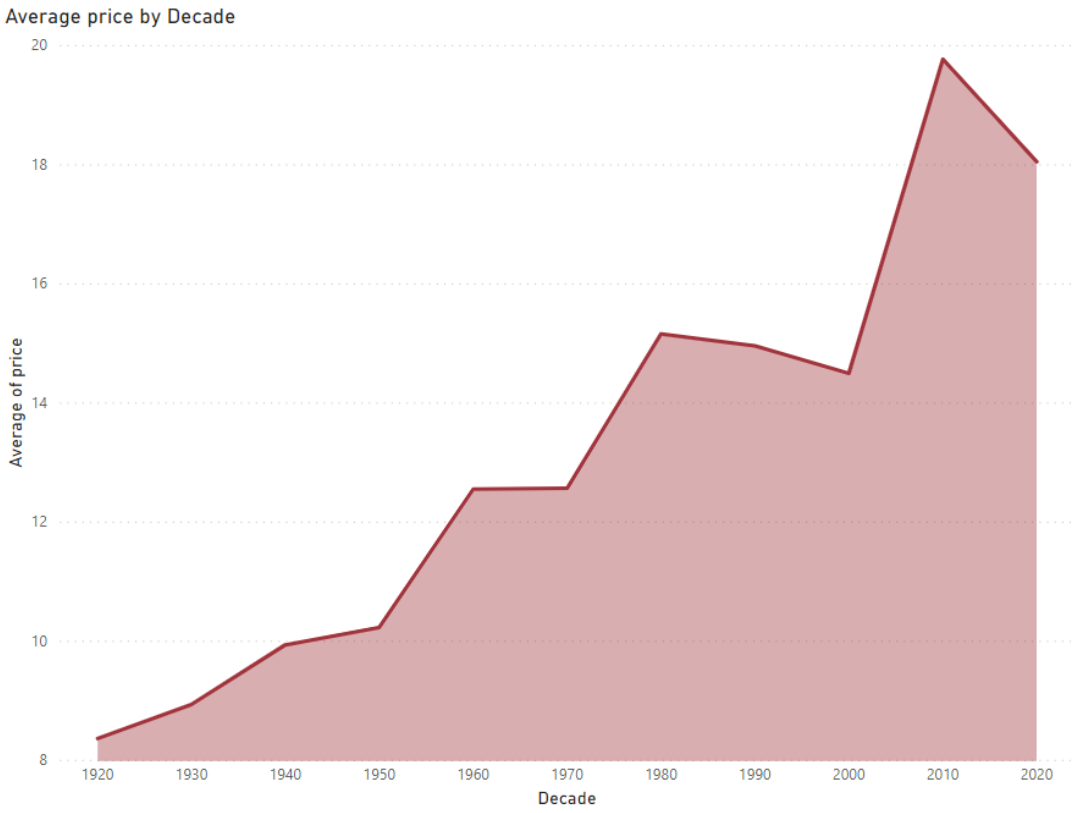
To find the number of Best Seller books per decade we used the **label_decade** column from the **pdate_dim** dimension on the X-axis, the **Rating** with **Average** function from the fact table **rbook_fact** on the Y-axis and displayed both axes in ascending order.

Finally, we put in the following filters:

Avarage Rating > 4.5: Using the **Rating** column with the **Avarage** function we only take into account books that have over 4.5 average rating (as there are books with the same title that are sold by more than one Seller)

label_reviews > 300: Using the **label_reviews** variable from the **book_dim** dimension we only consider books that have a number of reviews above 300.

Count of High Rated Books per Decade



To calculate the average book price per decade we used the **label_decade** column from the **pdate_dim** dimension on the X-axis, **Price** with the **Average** function from the fact table **rbook_fact** on the Y-axis and displayed both axes in ascending order.

label_decade is not 1800 or 1900: This filter was used as these years had very few books at high prices and the change in price would not be representative. (e.g. 1900 had 2 books with an average price of 56.64$)

6. Data Mining Models

In order to draw conclusions on our data that will provide business insights, we chose to create two models that leverage machine learning methods.

6.1 Clustering with Sentiment Analysis on Books

Data preparation and sentiment analysis:

The process begins with the preparation of the book title data, which has already undergone an extraction, transformation and loading (ETL) process. We selected the books that are Best Sellers with more than 10 reviews and from these we removed the books with a rating below 3, as they are extreme outliers that would distort the clustering results.

The data is then imported into a custom dataset class, TitlesDataset, designed for efficient handling and processing of batched data. This is a critical step in managing large datasets and ensures simplified processing.

Thus, a DataLoader object is created with this custom data set, set to a batch size of 32. For sentiment analysis, a Hugging Face pipeline is used, using a model specialized in sentiment analysis. The specified model, "lxyuan/distilbert-base-multilingual-cased-sentiments-student", provides a fine-tuned DistilBERT model capable of analyzing sentiments in multiple languages. The model is loaded on the appropriate hardware, taking advantage of GPU acceleration, if available, to speed up processing.

Sentiment analysis is then performed in batches. Each title is analyzed, and the sentiment (positive, negative or neutral) is extracted along with a confidence score. These results are compiled into a catalogue, capturing the sentiment data for each book title.

Customization of emotion tags and grouping:

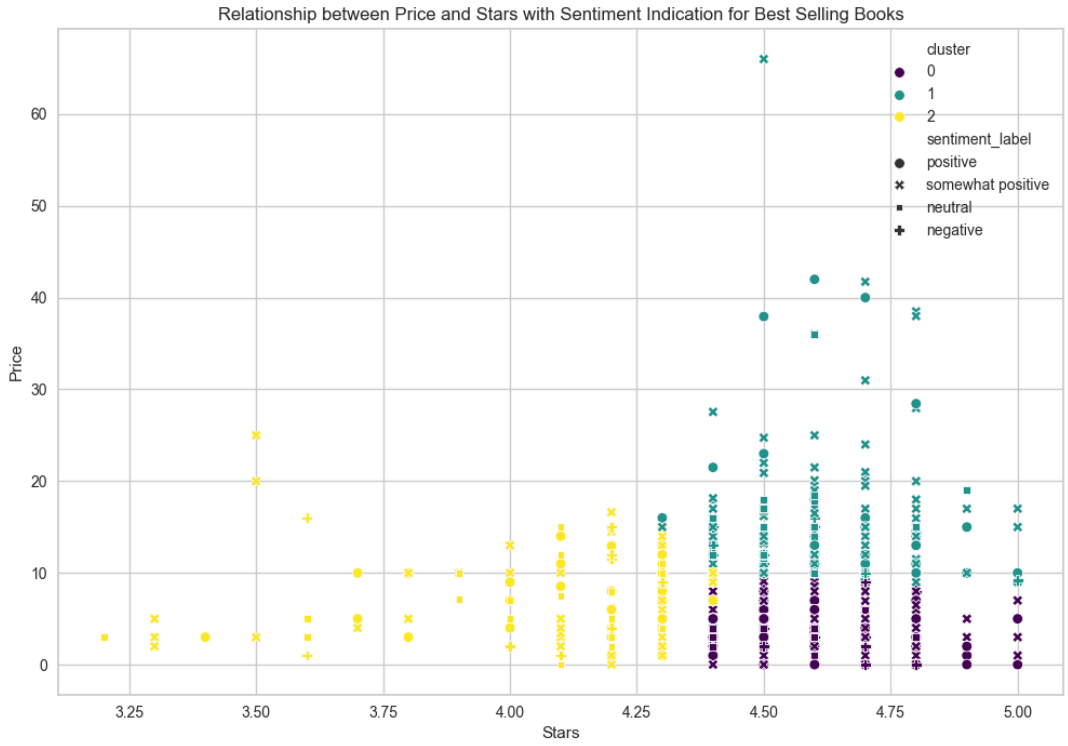
Next comes the customization of the emotion tags. Note that the sentiment analysis model shows a bias towards positive labels. To address this and achieve a balanced distribution of emotions, the labels are adjusted based on their confidence ratings. Labels with positive sentiment but lower confidence scores are relabeled as neutral or somewhat positive, based on specific score thresholds.

In the grouping phase, not only sentiment labels are taken into account, but also ratings and book prices. These additional columns are included to provide a more comprehensive understanding of the books' characteristics. To ensure uniformity and better clustering performance, these numeric columns are scaled. This standardization is crucial as it places the score and value on a similar scale to the sentiment scores, allowing for more effective clustering.

After sentiment analysis, label adjustments, and inclusion and scaling of rating and value data, we proceed with clustering. The optimal number of clusters is determined by the K-Elbow method and silhouette analysis, where we continue with the option suggested by both methods, which was for 3 clusters.

Visualization and interpretation:

The final step involves visualising these clusters. A scatter plot is used, where the x-axis represents the book ratings, and the y-axis represents the value. The clusters are differentiated by color and sentiment labels are indicated with different styles on the diagram.



From the diagram we can distinguish the three clusters as follows:

- Cluster 2 (yellow):

This cluster contains books with low ratings (below 4.5) and low prices (below $12). These books are by lesser-known authors and consist of older titles that have declined in value over time (such as Brave Companions, which is from the 2000s and has a rating of 4.4 and a price of $4.99).

- Cluster 1 (green):

This cluster consists of higher priced books (over $10), most with high ratings (greater than 4.5). These are premium titles, from well-known celebrities like **50 Cent, Kobe Bryant** and famous authors like **Markus Zusak, Jon Krakauer, Bruce Catton** with well-known titles.

- Cluster 0 (purple):

The books in this cluster are affordable, with prices under $10, but have higher scores, exceeding the 4.5 threshold by a significant amount. These books offer good value for money as they are written by mid-range authors and are sold at low prices such as 101 **Essays That Will Change The Way You Think** by Brianna West at $9.99.

After extracting the clusters we proceeded to test whether the sentiment in the titles (e.g. whether it is positive or negative) affects whether the best selling books are highly rated (with a rating >=4.5). For this procedure we performed an X squared test which evaluates whether there is a significant correlation between two categorical variables: is_high_rated (boolean variable where it is True only if the rating of the book is >=4.5) and sentiment_label. Calculates the Chi-Square statistic and p-value by comparing the observed frequencies of the data with the expected frequencies under the independence assumption.

As results we have Chi-Square: 7.567, indicating deviation from the expected if the variables were independent and p value: 0.05, which indicates that the result is at the threshold of statistical significance.

We confirm this by finding the percentage change between the observed and expected values which are shown in the table below:

sentiment_label is_hight_rated	negative	neutral	positive	somewhat positive
False	13.060076	25.285431	-1.646164	-10.975399
True	-7.282913	-14.100346	0.917978	6.120399

We can clearly see that the percentage change for the positive label is very small, however, the percentage change for the somewhat positive label and the negative and neutral emotion labels is significant, meaning that there is a relationship between the somewhat positive emotion label and the score and between the negative and neutral emotion labels and the score.

More specifically, we see that the somewhat positive label has a positive relationship with the evaluation, and the negative and neutral emotion labels have a negative relationship with the evaluation.

6.2 Association rules on books

Data preparation and initial analysis:

The process begins by reading clean data from a previous extraction, transformation, loading (ETL) process. An initial examination of the price distribution is performed, which leads to some interesting observations about book pricing. Based on this analysis, books are categorized as either cheap or expensive, with specific price thresholds defining these categories.

Feature Engineering:

To facilitate the analysis, two new columns, is_cheap and is_expensive, are created to indicate whether a book belongs to one of the two categories. In addition, the column is_highly_rated is introduced, where books with a rating higher than 4.5 are considered highly rated. This feature engineering step is critical, as it converts the data into a format more suitable for the upcoming analysis, allowing for a more nuanced understanding of the relationships between the various book features.

Encoding and data preparation with One-Hot Encoding:

Then one-hot coding of the category columns is done. One-hot encoding is a process in which categorical data is converted to a binary vector representation, which is essential for many analytical models that require boolean input. The relevant columns selected for analysis include is_highly_rated, is_cheap, is_expensive, and several label columns, such as label_isBestSeller, label_isEditorsPick, label_isGoodReadsChoice, and label_isKindleUnlimited. The inclusion of these columns indicates a focus on understanding how the various characteristics and classifications of books relate to each other.

Application of the Apriori algorithm:

The main analytical technique used is the Apriori algorithm. This algorithm is a classical method used in data mining to extract frequent data sets and extract correlation rules from a large data set. In the context of this analysis, the Apriori algorithm is applied to find frequent patterns or correlations between item sets in the dataset, focusing particularly on generated features such as book categories, ratings, and price categorizations.

Mining association rules:

After identifying frequent data sets with the Apriori algorithm, we proceed to mining association rules.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
3	(label_isKindleUnlimited)	(is_cheap)	0.155560	0.542065	0.142072	0.913294	1.684842	0.057748	5.281464	0.481352
7	(is_highly_rated, label_isKindleUnlimited)	(is_cheap)	0.062363	0.542065	0.054693	0.877011	1.617909	0.020888	3.723398	0.407320
2	(label_category_Children's eBooks)	(is_cheap)	0.054741	0.542065	0.045159	0.824967	1.521898	0.015486	2.616283	0.362785
5	(label_category_Children's eBooks)	(is_highly_rated)	0.054741	0.492354	0.043475	0.794195	1.613056	0.016523	2.466632	0.402068
0	(is_highly_rated)	(is_cheap)	0.492354	0.542065	0.257754	0.523513	0.965775	-0.009134	0.961064	-0.065254
4	(is_expensive)	(is_highly_rated)	0.244696	0.492354	0.125753	0.513915	1.043791	0.005276	1.044356	0.055545
1	(is_cheap)	(is_highly_rated)	0.542065	0.492354	0.257754	0.475503	0.965775	-0.009134	0.967872	-0.071828
6	(label_isKindleUnlimited)	(is_highly_rated)	0.155560	0.492354	0.062363	0.400891	0.814233	-0.014228	0.847335	-0.212709
8	(is_cheap, label_isKindleUnlimited)	(is_highly_rated)	0.142072	0.492354	0.054693	0.384965	0.781886	-0.015257	0.825393	-0.245371
9	(label_isKindleUnlimited)	(is_highly_rated, is_cheap)	0.155560	0.257754	0.054693	0.351586	1.364039	0.014597	1.144711	0.316048

Conclusions:

- **Strong correlation between "is_cheap" and "label_isKindleUnlimited"**: with high support (0.913294) and significant lift (1.684842), there is a high probability that if a Kindle eBook is part of the Kindle Unlimited program, it also costs cheap. This could

be a strategy by Amazon to attract more subscribers to Kindle Unlimited by offering cheaper books.

- **Highly rated and Kindle Unlimited e-books tend to be cheap**: Highly rated ebooks that are part of Kindle Unlimited also tend to be cheap (support = 0.877011, lift = 1.617909). This shows a trend where high-quality content is becoming affordable within the Kindle Unlimited program.
- **Children's ebooks are often cheap and highly rated**: Children's ebooks have a notable tendency to be cheap (support = 0.824967) and high scoring (support = 0.794195). This reflects a purchasing strategy that targets parents seeking affordable, quality educational content.
- **Expensive books and high scores**: There is a moderate correlation between expensive books and high ratings (support = 0.513915, lift = 1.043791), suggesting that higher priced books may be of higher quality or perceived as such.
- **Negative correlation in some cases**: There are also negative correlations, such as books with high scores that are not necessarily cheap (lift < 1), indicating that quality (as perceived by the ratings) does not always align with lower values.

7. Annex

SQL Script for the **Clean Invalid Values** step of the ETL Workflow:

```
BEGIN TRANSACTION; -- Start of the transaction

-- Delete rows where title is 'Not Found'
DELETE FROM staging_v2
WHERE title = 'Not Found';

-- Delete rows where author is NULL and title contains 'USER GUIDE'
DELETE FROM staging_v2
WHERE author IS NULL AND title LIKE '%USER GUIDE%';

-- Update rows where author is NULL to 'Other'
UPDATE staging_v2
SET author = 'Other'
WHERE author IS NULL;

COMMIT TRANSACTION; -- End of the transaction
```

SQL Script for the **Handle Duplicates** step of the ETL Workflow:

```
SELECT * INTO #main_temp from staging_v2

SELECT *
INTO #duplicates
FROM (
   SELECT *,
        COUNT(*) OVER (PARTITION BY author, title) as duplicate_count
   FROM #main_temp
) as subquery
WHERE duplicate_count > 1
ORDER BY  author, title;

-- Declare the variable
DECLARE @FirstCount INT;

-- Create a temporary table to store the results
-- Adjust the column definitions according to the structure of #duplicates
SELECT *
INTO #TempResults
FROM #duplicates
WHERE 1 = 0;

-- Declare a variables
DECLARE @SQLQuery NVARCHAR(MAX);
DECLARE @MaxReview INT, @MaxStar DECIMAL(3, 1), @MinAsin VARCHAR(500);
DECLARE @IDsToDelete TABLE (asin varchar(500));
DECLARE @NumberedRows TABLE (
   asin VARCHAR(MAX), -- adjust the data type as needed
   RowNum INT
);
DECLARE @FirstAsin VARCHAR(MAX); -- adjust data type as needed
DECLARE @SecondAsin VARCHAR(MAX); -- adjust data type as needed
DECLARE @FirstSoldBy VARCHAR(MAX); -- adjust data type as needed
DECLARE @SecondSoldBy VARCHAR(MAX); -- adjust data type as needed
DECLARE @FirstPublishDate DATETIME; -- adjust data type as needed
DECLARE @SecondPublishDate DATETIME;

--Handle the special cases that have more than 2 duplicates
UPDATE #duplicates
SET duplicate_count = 2
WHERE duplicate_count = 4 AND soldBy = 'De Marque'

UPDATE #duplicates
SET duplicate_count = 2
WHERE duplicate_count = 4 AND soldBy = 'Amazon.com Services LLC'

UPDATE #duplicates
SET duplicate_count = 2
WHERE duplicate_count = 5 AND soldBy = 'De Marque'

UPDATE #duplicates
SET duplicate_count = 2
WHERE duplicate_count = 5 AND soldBy = 'Amazon.com Services LLC'

DELETE FROM #duplicates
WHERE asin = 'B076PZBTZK';

UPDATE #duplicates
SET duplicate_count = 2
WHERE duplicate_count = 5 AND soldBy = 'Amazon.com Services LLC' AND title='Exploring Psychology'

UPDATE #duplicates
SET title='Exploring Psychology - Re-Editon'
WHERE asin = 'B0BSP7DJ5Q'

UPDATE #duplicates
SET title='Wuthering Heights - Re-Editon II'
WHERE asin = 'B0BY7FFRCJ';

UPDATE #duplicates
SET title='Wuthering Heights - Re-Editon'
WHERE asin = 'B077711TRG';
```

```
UPDATE #duplicates
SET title='Legal and Ethical Issues for Health Professionals - Re-Editon II'
WHERE asin = 'B0BWSFRR49';

UPDATE #duplicates
SET title='Legal and Ethical Issues for Health Professionals - Re-Editon'
WHERE asin = 'B07KJMB439'

UPDATE #duplicates
SET title='1984 - Re-Editon II'
WHERE asin = 'B0CFQD9ZQH';

UPDATE #duplicates
SET title='Animal Farm - Re-Editon II'
WHERE asin = 'B0CD2FTLFB';

UPDATE #duplicates
SET title='Animal Farm - Re-Editon'
WHERE asin = 'B0C5FLKBWK';

UPDATE #duplicates
SET title='Automotive Technology: A Systems Approach - Re-Editon II'
WHERE asin = 'B07LH2X39B';

UPDATE #duplicates
SET title='Automotive Technology: A Systems Approach - Re-Editon'
WHERE asin = 'B00H7HV7AQ';

UPDATE #duplicates
SET title='War and Peace - Re-Editon II'
WHERE asin = 'B08NXSC6Z6';

UPDATE #duplicates
SET title='War And Peace - Re-Editon'
WHERE asin = 'B0C7TMQ95X';

UPDATE #duplicates
SET title='The U.S. Supreme Court: A Very Short Introduction (VERY SHORT INTRODUCTIONS) - Re-Editon II'
WHERE asin = 'B0CD4F26L6';

UPDATE #duplicates
SET title='The U.S. Supreme Court: A Very Short Introduction (VERY SHORT INTRODUCTIONS) - Re-Editon'
WHERE asin = 'B08761Z5K2';

UPDATE #duplicates
SET title='The Master and Margarita - Re-Editon II'
WHERE asin = 'B01DV1Y7D0';

UPDATE #duplicates
SET title='The Master and Margarita - Re-Editon'
WHERE asin = 'B0BFQRZ8DW';

UPDATE #duplicates
SET title='The Art Of War - Re-Editon'
WHERE asin = 'B0BW9W18HW';

UPDATE #duplicates
SET title='Advanced Practice Nursing: Essential Knowledge for the Profession - Re-Editon II'
WHERE asin = 'B0BS77B5L1';

UPDATE #duplicates
SET title='Advanced Practice Nursing: Essential Knowledge for the Profession - Re-Editon'
WHERE asin = 'B07XQSTCYP';

DELETE FROM #duplicates WHERE duplicate_count = 3;

-- Decalre a variable to control the loop
DECLARE @Loop BIT;
SET @Loop = 1; -- Equivalent to True
```

```
-- Loop through the rows in #duplicates
WHILE @Loop = 1
   BEGIN

      SELECT TOP 1 @FirstCount = duplicate_count
      FROM #duplicates;


      -- Create a temporary table to store the results
-- Adjust the column definitions according to the structure of #duplicates

-- Construct the SQL query string
      SET @SQLQuery = 'INSERT INTO #TempResults SELECT TOP ' + CAST(@FirstCount AS NVARCHAR) + ' * FROM #duplicates';

-- Execute the SQL query
      EXECUTE sp_executesql @SQLQuery;

--IF BOTH ARE NULL IN SOLD_BY
      IF NOT EXISTS (SELECT 1 FROM #TempResults WHERE soldBy IS NOT NULL)
         BEGIN
            -- Find the row with the highest review and star values
            SELECT TOP 1 @MinAsin = asin
            FROM #TempResults
            ORDER BY reviews Asc, stars Asc;

            -- Delete other rows from the main table that do not match the max value row
            DELETE
            FROM #main_temp
            WHERE asin = @MinAsin;
         END


--IF ONLY ONE IS NULL IN SOLDBY
      IF (
         (SELECT COUNT(*) FROM #TempResults WHERE soldBy IS NULL) = 1
            AND
         (SELECT COUNT(*) FROM #TempResults WHERE soldBy IS NOT NULL) = 1
         )
         BEGIN
            -- Declare a table variable to store IDs of rows to be deleted
            -- Insert IDs of rows with NULL in 'sold_by' from #TempResults into @IDsToDelete
            INSERT INTO @IDsToDelete(asin)
            SELECT asin
            FROM #TempResults
            WHERE soldBy IS NULL;

            -- Delete rows from #main_temp where IDs match those in @IDsToDelete
            DELETE
            FROM #main_temp
            WHERE asin IN (SELECT asin FROM @IDsToDelete);

         END


      IF NOT EXISTS (SELECT 1 FROM #TempResults WHERE soldBy IS NULL)
         BEGIN
            INSERT INTO @NumberedRows (asin, RowNum)
            SELECT asin,
                  ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS RowNum
            FROM #TempResults;

            SELECT TOP 1 @FirstAsin = asin FROM @NumberedRows WHERE RowNum = 1;
            SELECT TOP 1 @SecondAsin = asin FROM @NumberedRows WHERE RowNum = 2;

            SELECT @FirstSoldBy = soldBy FROM #TempResults WHERE asin = @FirstAsin;
            SELECT @SecondSoldBy = soldBy FROM #TempResults WHERE asin = @SecondAsin;

            SELECT @FirstPublishDate = CONVERT(DATE, publishedDate) FROM #TempResults WHERE asin = @FirstAsin;
            SELECT @SecondPublishDate = CONVERT(DATE, publishedDate) FROM #TempResults WHERE asin = @SecondAsin;
```

```
            IF @FirstSoldBy = @SecondSoldBy
               BEGIN

                  -- Check which publishDate is more recent and append -Rediton to its title
                  IF @FirstPublishDate > @SecondPublishDate
                     BEGIN
                        UPDATE #main_temp
                        SET title = title + ' Re-Editon'
                        WHERE asin = @FirstAsin;
                     END
                  ELSE
                     IF @SecondPublishDate > @FirstPublishDate
                        BEGIN
                           UPDATE #main_temp
                           SET title = title + ' Re-Editon'
                           WHERE asin = @SecondAsin;
                        END
               END

         END


      DELETE FROM #duplicates WHERE asin IN (SELECT asin FROM #TempResults);

      IF NOT EXISTS (SELECT 1 FROM #duplicates)
         BEGIN
            -- If #duplicates is empty, set @Loop to 0
            SET @Loop = 0;
         END

      DELETE FROM #TempResults
      DELETE FROM @NumberedRows;

   END

DELETE FROM staging_v2

INSERT INTO staging_v2
SELECT *
FROM #main_temp
ORDER BY title,author


IF OBJECT_ID('tempdb..#duplicates') IS NOT NULL
   DROP TABLE #duplicates;

IF OBJECT_ID('tempdb..#TempResults') IS NOT NULL
DROP TABLE #TempResults;

IF OBJECT_ID('tempdb..#main_temp') IS NOT NULL
DROP TABLE #main_temp;


UPDATE staging_v2
SET soldBy = 'Other'
WHERE soldBy IS NULL;
```

SQL Script for the **Merge Similar Names for Sellers** step of the ETL Workflow:

```
UPDATE staging_v2
SET soldBy = CASE
   WHEN LOWER(soldBy) LIKE '%amazon%' THEN 'Amazon.com'
   WHEN LOWER(soldBy) LIKE '%random house%' OR LOWER(soldBy) LIKE '%penguin%' THEN 'Penguin Random House'
   WHEN LOWER(soldBy) LIKE '%prh%' THEN 'Penguin Random House'
   WHEN LOWER(soldBy) LIKE '%rh%' THEN 'Penguin Random House'
   WHEN LOWER(soldBy) LIKE '%harpercollins%' OR LOWER(soldBy) LIKE '%harper collins%' THEN 'HarperCollins'
   WHEN LOWER(soldBy) LIKE '%simon%' THEN 'Simon & Schuster'
   WHEN LOWER(soldBy) LIKE '%macmillan%' THEN 'Macmillan'
   ELSE soldBy
END
```

Code for **Data Mining models:**

1. clust_with_sent.ipynb: This notebook contains the code and implementation details of the 1ου model (along with the X-Squared check).
2. association_clean.ipynb: This notebook contains the code and implementation details of the 2ου model.

The data were extracted from the Data Warehouse as they were generated after the ETL process in csv format, where the fact table and each dimension created exists in a separate csv file.