Project Overview

This project focuses on implementing a full data science pipeline using data extracted from the website Books to Scrape. The website serves as a mock e-commerce platform for books, making it ideal for practicing web scraping, data processing, and analysis techniques in a controlled environment.

The workflow begins with **automated data extraction**, where book-related information is scraped from all 50 pages of the website. Each book entry includes key attributes such as the title, price, availability status, star rating, and category.

Once the raw data is collected, it undergoes a detailed **cleaning and transformation process**. This step addresses missing values, inconsistent formats, duplicated records, and text noise. Regular expressions (regex) are used to efficiently parse and standardize specific fields such as prices, stock numbers, and star ratings.

After Scrapping, the data is stored in a structured **CSV file**, which facilitates further exploration and analysis. The project then proceeds to the **analysis phase**, where patterns and trends are identified across book categories, ratings, and pricing. **Visualizations** are created to effectively present these insights using charts and graphs.

Finally, the project concludes by preparing the processed data for **storage** in a database and **deploying an interactive web app** using Streamlit for enhanced accessibility and presentation.

This end-to-end project demonstrates practical knowledge in web scraping, data wrangling, exploratory analysis, and data storytelling key skills in the field of data science.

Project Goal

The primary goal of this project is to practice and demonstrate end-to-end data science skills by performing **web scraping, data cleaning, analysis, visualization**, and **storage** on a real-world-style dataset. The project uses the website Books to Scrape, which is designed as a sandbox environment for learning and testing web scraping techniques.

Specifically, the objectives are:

- **Extract structured data** from a multi-page online catalog of books.
- **Clean and preprocess** the raw data by handling missing values, formatting issues, outliers, and inconsistencies using various techniques including **Regular Expressions (Regex)**.
- **Analyze** the dataset to generate meaningful statistics, patterns, and trends across attributes like price, rating, category, and availability.
- **Visualize** the key insights using appropriate charts and graphs to clearly communicate findings.
- **Store** the cleaned and processed dataset into a structured format (CSV and optionally into a database such as MongoDB).
- **Build an interactive dashboard** using tools like Streamlit to present the data and analysis results in an engaging and user-friendly way.

The project simulates a real-world data science workflow, from raw data collection to insight generation and final reporting. It aims to strengthen both technical and analytical skills while reinforcing data handling best practices.

Data Extraction and Web Scraping Process

1. Website Selection

The selected data source for this project is Books to Scrape, an online mock bookstore designed specifically for web scraping practice. The website contains 50 pages, each listing multiple books with details including title, price, rating, availability, and category.

---

2. Data Collection Strategy

The scraping process was carried out by simulating page-by-page browsing across all 50 pages of the website. Each page contains multiple book entries embedded in structured HTML elements. A loop was used to systematically access and parse each page's content.

For every book found on a page:

- The **title** was extracted from the anchor tag inside the `<h3>` element.
- The **price** was retrieved from a span tag with the class `price_color`.
- The **rating** was inferred by examining the CSS class assigned to the book's `<p>` tag, which includes textual descriptors like "Three", "Four", etc.
- The **availability** status was pulled from the `availability` class.
- A **link to the individual book page** was also collected in order to retrieve additional information.

---

3. Accessing Individual Book Pages

Each book's relative link was modified to generate the full URL. This allowed access to the detailed product page of each book. On this page:

- The **category** of the book was extracted from the breadcrumb navigation structure near the top of the page. Specifically, the third list item in the breadcrumb trail was used, which typically holds the book's category name.
- In cases where the category couldn't be found due to unexpected structure, a default value of "Unknown" was assigned.

---

4. Error Handling

The scraping loop included exception handling to skip over books that caused parsing errors or missing elements, ensuring that the process continued without interruption. A notification was printed each time a book was skipped due to such errors.

---

5. Data Storage

Once all 50 pages were scraped:

- The collected data was organized into a structured format with columns such as `Title`, `Price`, `Rating`, `Availability`, and `Category`.
- This data was then converted into a **CSV file (`books.csv`)**, encoded in UTF-8 for compatibility and ease of use in further analysis.

---

6. HTML Page Exploration

To better understand the website's structure and verify successful scraping:

- The entire HTML content of the homepage was explored.
- The `<title>` tag was extracted to confirm the page identity.
- All `<a>` (anchor) tags were retrieved to list every hyperlink available on the homepage.
- Additionally, the content and destination of each `<a>` tag was printed:
  - **HREF attributes** were listed to identify the actual URLs.
  - **Text content** inside the anchor tags was also printed to show what users see as clickable text.

This exploratory step helped validate the structure of the HTML and ensured the scraping logic was correctly aligned with the website layout.

---

7. Final Outcome

In total, hundreds of books were scraped, each with multiple attributes. The data was successfully saved into a CSV file, which serves as the basis for subsequent stages: data cleaning, analysis, and visualization.

Data Cleaning and Processing Report

1. Overview

After scraping data from Books to Scrape, the raw information was converted into a structured CSV file. This step ensured each book record included consistent fields such as title, price, availability, rating, and category. To prepare the dataset for meaningful analysis and visualization, we performed comprehensive cleaning and transformation steps, removing noise, correcting formats, and applying regular expressions where needed.

---

2. Cleaning Objectives

Cleaning the dataset was essential to:

- Eliminate invalid, duplicated, or irrelevant data
- Correct formatting and improve consistency
- Prepare the data for statistical operations, visualizations, and database storage

---

3. Cleaning Steps

a. Handling Missing Values

- Identified missing or empty fields in essential columns such as `Title`, `Price`, and `Rating`.
- Rows with missing critical values were removed to avoid skewed analysis.
- Less critical fields with missing values were left as `NaN` for flexible handling.

b. Dropping Duplicates

- Checked for and removed duplicate rows to prevent repeated records from affecting aggregate metrics.
- Duplicates were identified by matching identical combinations of `Title`, `Category`, and `Price`.

c. Connecting Data Types and Fixing Format Issues

- Converted all numeric-related fields (like `Price`, `Rating`, and `Stock Availability`) into proper numerical data types (**float or integer**).
- Parsed values like "£35.99" to `35.99` and "In stock (22 available)" to `22` using string manipulation and regular expressions.
- Corrected text fields that had numbers stored as text (e.g., ratings like "Three" became numeric `3`).

d. Dealing with Wrong or Inconsistent Data

- Removed out-of-context data such as unexpected symbols in text, malformed prices, or incomplete records.
- Validated `Rating` values to ensure they fall within the expected 1–5 range.
- Checked for non-standard categories or book entries with placeholder names and cleaned them.

e. Text Standardization

- Stripped leading and trailing whitespace across all textual fields.
- Applied consistent casing (e.g., Title Case for book titles and categories).
- Replaced inconsistent separators, such as double spaces or special characters, with standardized spaces or dashes.

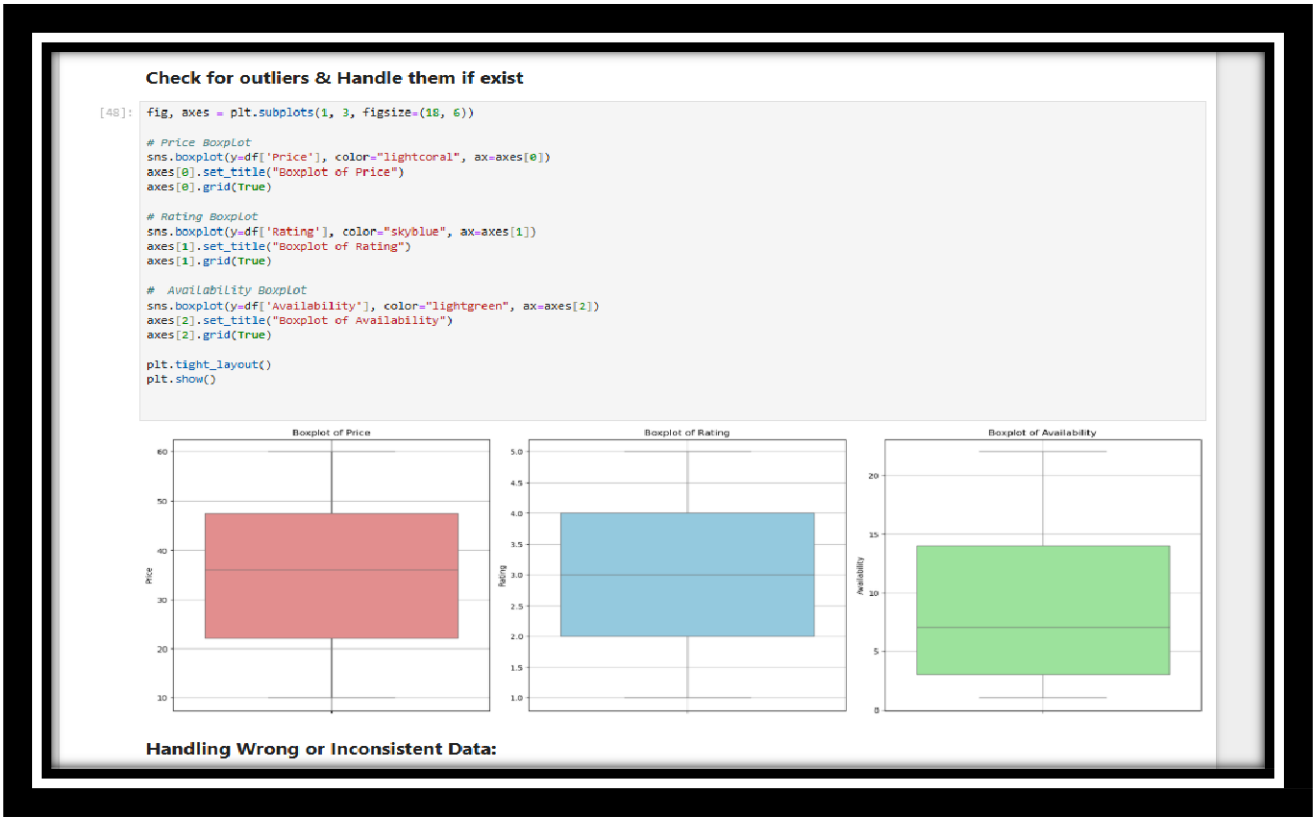
f. Encoding and Character Cleanup

- Ensured the file used UTF-8 encoding to support a wide range of characters including accented letters, symbols, and punctuation.
- Removed HTML tags, non-ASCII characters, and encoding artifacts (like `\xa0` **or** `\u2013`) that may have been scraped with the content.

g. Handling Outliers

- Detected unusually high or low price values which might indicate scraping errors or mis formatted data.
    - Example: A price like `£999.99` could be a test entry or outlier and was reviewed before analysis.
- Also reviewed books with a `0` rating or extremely high stock values to validate their authenticity.

- Created boxplots for rating , price and availability



4. Regular Expressions Applications

Regular expressions (regex) were essential for cleaning and parsing semi-structured text scraped from the website. Here are the key transformations performed using regex:

---

a. Price Extraction

Extract the numerical part of the price and remove the currency symbol.

- **Original:** `£53.74`
- **Regex Pattern:** `£(\d+\.\d+)`
- **Result:** `53.74` (as float)

---

b. Availability Count

Extract how many items are available in stock from strings like `In stock (22 available).`

- **Original:** `In stock (22 available)`

- **Regex Pattern:** `\((\d+)\savailable\)`
- **Result:** `22` (as integer)

---

c. Star Rating Conversion

Extract the rating word from HTML class name and map it to a number.

- **Original:** `class="star-rating Three"`
- **Regex Pattern:** `star-rating\s(\w+)`
- **Mapping:** `"One" → 1, "Two" → 2, ..., "Five" → 5`
- **Result:** `3` (as integer)

---

d. Extract Category from Breadcrumb (if scraped)

If categories were extracted from breadcrumbs like **`/books/mystery_3/index.html`**, extract the category name.

- **Original:** `/books/mystery_3/index.html`
- **Regex Pattern:** `/books/([a-zA-Z0-9-_]+)_\d+/`
- **Result:** `mystery`

---

e. Remove HTML Tags

Sometimes descriptions or titles may include residual HTML tags.

- **Original:** `<p>This is a <b>great</b> book!</p>`
- **Regex Pattern:** `<[^>]+>`
- **Result:** `This is a great book!`

---

f. Extract Numbers from Mixed Text

Sometimes numbers are embedded within text and need to be isolated.

- **Original:** `Only 12 left in stock!`
- **Regex Pattern:** `\d+`
- **Result:** `12`

---

g. Remove Special Characters from Titles

Clean book titles by removing unwanted symbols (e.g., currency signs, unusual punctuation).

- **Original:** `Best Book: "The £ecret #Life* of! Bees?"`
- **Regex Pattern:** `[^a-zA-Z0-9\s\.,'-]`
- **Result:** `Best Book The ecret Life of Bees`

---

h. Validate ISBN or Product Code (if included)

If product codes or ISBNs were included, regex can validate them.

- **Example ISBN (13 digits):** `978-3-16-148410-0`
- **Regex Pattern:** `\d{3}-\d{1,5}-\d{1,7}-\d{1,7}-\d{1}`

---

i. Normalize Whitespace

Reduce any sequence of multiple spaces to a single space.

- **Original:** `This book is amazing.`
- **Regex Pattern:** `\s+`
- **Result:** `This book is amazing.`

---

j. Extract Book Title Keywords

Used to isolate keywords in a book title (for search, tagging, or NLP).

- **Original:** `The Mystery of the Blue Train`
- **Regex Pattern:** `\b[A-Z][a-z]+\b`
- **Result:** `["The", "Mystery", "Blue", "Train"]` (ignores lowercase stop words)

---

k. Extract Year from Description (if dates appear)

Sometimes books mention dates like "Published in 2016".

- **Original:** `First published in 2016 by XYZ.`
- **Regex Pattern:** `\b(19|20)\d{2}\b`
- **Result:** `2016`

---

l. Detect Uppercase Acronyms in Title

Identify acronyms like "USA", "NASA", etc., used in book titles or descriptions.

- **Original:** `Traveling Through the USA on Foot`
- **Regex Pattern:** `\b[A-Z]{2,}\b`
- **Result:** `["USA"]`

---

m. Extract Decimal Numbers from Mixed Text

Used when decimal values are buried in strings.

- **Original:** `This item costs around 23.99 dollars.`
- **Regex Pattern:** `\d+\.\d+`
- **Result:** `23.99`

---

n. Remove Bracketed Text (Editorial Notes, etc.)

Clean titles or text fields that include notes like `[Hardcover]` or `(Illustrated)`.

- **Original:** `Harry Potter [Hardcover Edition]`
- **Regex Pattern:** `\[.*?\]|\(.*?\)`
- **Result:** `Harry Potter`

---

o. Extract URL Slugs (from scraped links)

If scraping links like `/book/the-great-gatsby_45/index.html`, extract slug:

- **Original:** `/book/the-great-gatsby_45/index.html`
- **Regex Pattern:** `/book/([a-z0-9-]+)_`
- **Result:** `the-great-gatsby`

---

p. Extract Currency Symbols

Useful when prices might use multiple currencies (e.g., £, $, €).

- **Original:** `$49.99` or `€45.00`
- **Regex Pattern:** `[£$€]`
- **Result:** `$` or `€`

---

q. Find Books with Ellipses or Truncated Titles

Detect titles that were truncated, possibly due to limited HTML preview.

- **Original:** `The Lord of the...`
- **Regex Pattern:** `\.\.\.$`
- **Result:** Detected as a potentially incomplete title

---

r. Strip Extra Punctuation (for NLP cleaning)

Remove repeated punctuation for clean text.

- **Original:** `"Wow!!! What a book!!!"`
- **Regex Pattern:** `[!?.]{2,}`
- **Result:** `"Wow! What a book!"`

---

s. Detect Email or Contact Info (if mistakenly included in descriptions)

Sometimes, descriptions may have leftover contact info.

- **Regex Pattern:** `[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-z]{2,}`
- **Matches:** `support@example.com`, `info.books@mail.org`

---

t. Parse Ratings as Words with Optional Tags

Handle variations like: `Rated: "Four stars"`, `4-star book`, etc.

- **Regex Pattern:** `(?:One|Two|Three|Four|Five)[\s-]*stars?`
- **Result:** Extracts phrases related to book ratings

Data Analysis Report

1. Overview

The primary objective of this data analysis is to extract valuable insights from a dataset sourced from an eBook store. This includes calculating descriptive statistics and identifying trends and correlations. By exploring both numerical and categorical features such as price and ratings we aim to uncover relationships and patterns that inform decision-making and business strategies.

---

2. Objectives of Analysis

The goals of this analysis include:

• Compute descriptive statistics for both numerical and categorical variables.
• Identify price and rating trends across different categories.
• Explore correlations and patterns using statistical techniques.

---

3. Libraries Used

To facilitate data analysis and modeling, the following libraries were employed:

• **Pandas** – For data manipulation and preprocessing.
• **NumPy** – For handling arrays and performing numerical operations.

• Scikit-learn – for clustering such as K-Means.

---

4. Descriptive Statistics

This step summarizes the characteristics of the dataset.

a. Numerical Features

Analyzed fields include `price` and `rating`

- **Computed metrics:**

  - Mean
  - Median
  - Mode
  - Standard Deviation
  - Minimum & Maximum

Example Insights:
• The average price of eBooks is **$X**, with a standard deviation of **$Y**.
• The mean rating across books is **Z**, with values ranging from 1 to 5.

After computing numerical features(price, rating, availability), we notice that:

1. The average price of eBooks is $35.06, with a standard deviation of $14.46.
2. `The mean rating across books is 2.92, with values ranging from 1 to 5.`

3. The mean Availability across books is 8.57

Summary for numerical features

	Price	Rating	Availability
count	996.000000	996.000000	996.000000
mean	35.056586	2.924699	8.571285
std	14.462543	1.436548	5.652050
min	10.000000	1.000000	1.000000
25%	22.095000	2.000000	3.000000
50%	35.940000	3.000000	7.000000
75%	47.457500	4.000000	14.000000
max	59.990000	5.000000	22.000000

b. Categorical Features

Analyzed fields of `category`.

• Frequency distributions were generated to determine common categories.
• Top and common rating categories were identified.

Example Insights:
• The most common category is **"Z"**, representing **A%** of the dataset.

After coding we realize that the most common Category is 'Default' representing 15.16% of the dataset.

---

5. Trend Identification

Trends help uncover directional patterns in key variables.

	Price	Rating
Price	1.000000	0.027128
Rating	0.027128	1.000000

We notice from the correlation matrix that price rating correlation > 0; therefore A weak positive correlation was observed between Price and Rating.

Higher-priced books tend to receive better ratings.

a. Price Trends

• Certain category such as **fiction or science** may tend to have higher average prices.

After coding we realize that Certain categories tend to have higher average prices:

- Suspense: $58.33
- Novels: $54.81
- Politics: $53.61
- Health: $51.45
- New Adult: $46.38

b. Rating Trends

• Books with higher user ratings often show stronger sales performance.

After coding we realize that Certain categories tend to have higher average ratings:

- Erotica: 5.00
- Adult Fiction: 5.00
- Novels: 5.00
- Christian Fiction: 4.17
- Health: 3.75

---

6. Pattern Recognition and Relationship Analysis

a. Correlation Analysis

Statistical correlation was computed between numerical variables such as price and rating.

Example for possible Findings:
• Price and availability showed a weak **negative correlation**.

	Price	Availability
Price	1.000000	-0.012993
Availability	-0.012993	1.000000

After coding We notice that price availability correlation < 0 ; therefore A weak negative correlation was observed between price & availability.

No linear relationship between price & availability.

b. Clustering

Using clustering techniques (e.g., K-Means), books were grouped by features like price and rating to detect natural groupings.

Example for possible Clusters:

• Cluster A – " Don't Get Caught"," Ender's Game (The Ender Quintet #1)".
• Cluster B – "The Hobbit (Middle-Earth Universe)" , "The Nightingale".

- **Number of Books in each Cluster**
- **`Cluster Centers (Price, Rating)`**

	Cluster	Book Count
0	0	307
1	1	309
2	2	380

	Price	Rating
0	47.459773	4.061688
1	20.671942	3.656958
2	36.704802	1.403694

Data Visualization Report

1. Overview

Data visualization is a powerful method for presenting complex datasets in a simplified and interpretable manner. It transforms raw data into graphical formats such as charts, graphs, and plots, making it easier to detect trends, patterns, and outliers. In this project, visualization is used to explore and communicate insights from an eBook dataset.

---

2. Objectives of Visualization

The primary goals of using data visualization are:

• Simplify and summarize complex data.
• Enhance comprehension and retention of information.
• Identify meaningful trends and anomalies.
• Support data-driven decision-making processes.

Each visualization technique was selected based on the type of data and the analytical question it addresses.

---

3. Criteria for Visualization Selection

Each visualization type is chosen based on two key factors:

• **Analytical Purpose** – What question are we trying to answer? (e.g., trends, comparison, distribution).
• **Data Type** – What is the nature of the data? (Numerical, Categorical, or Textual).

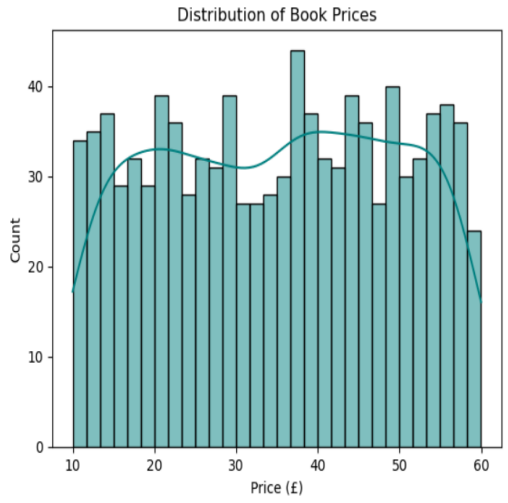
---

4. Visualization Types and Applications

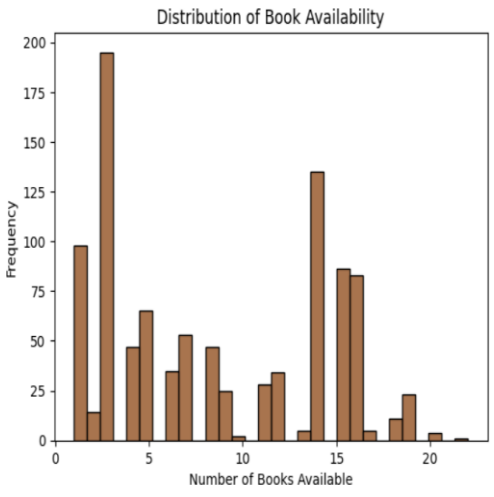
Visualization Type:	Best For:	Why will we use it in our data:
Histogram:	Distribution of numerical data.	**To show the distribution of book prices.**
Count Plot (Bar Chart):	Frequency of categorical data.	**To show how many books have each rating (1-5).**
Box Plot:	Comparing groups.	**To compare price distribution across different ratings.**
Horizontal Bar Chart:	Ranking categories by a metric.	**To rank categories by their average book price.**
Scatter Plot:	Relationship between two variables.	**To examine relationship between price & rating for top categories.**
Violin Plot:	Distribution & density within categories.	**To show where most books are priced in each genre.**
Word Cloud:	Frequent terms in text data.	**To display most frequent words in book titles.**
Heatmap:	Correlation between numerical variables.	**To check if price & rating are correlated.**
Line Plot (Cumulative Distribution):	Understanding percentiles of data.	**To show what percentage of books fall under a certain price.**

1] Histogram:

Show the distribution of book prices:

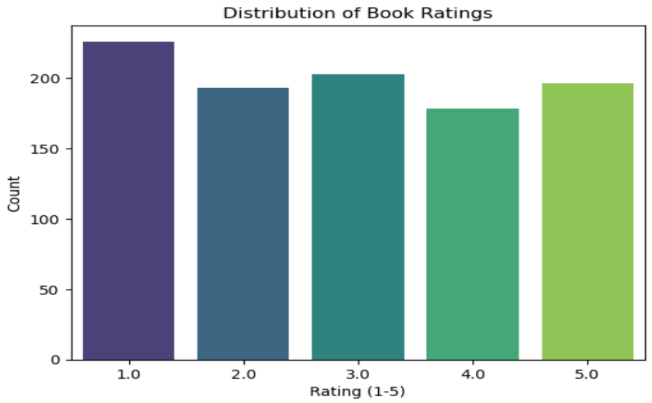


Show the distribution of book availability:



2] Count Plot (Bar Chart):

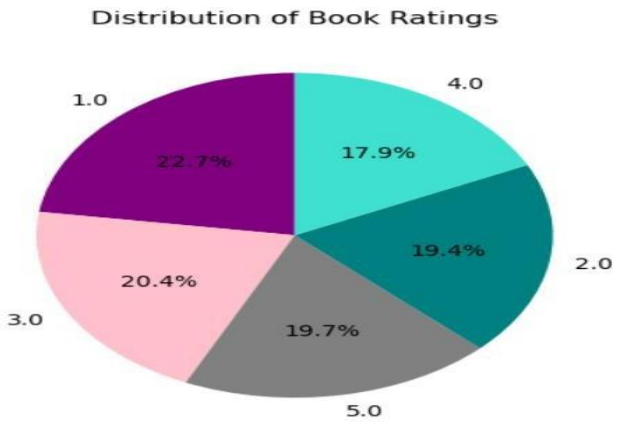
Show how many books have each rating (1-5):



Rating	
1.0	226
3.0	203
5.0	196
2.0	193
4.0	178

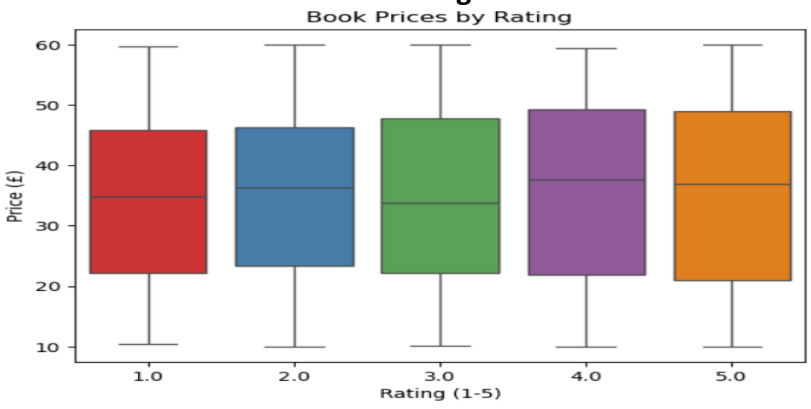
3] Pie Chart:

Show how many books have each rating (1-5):



4] Boxplot:

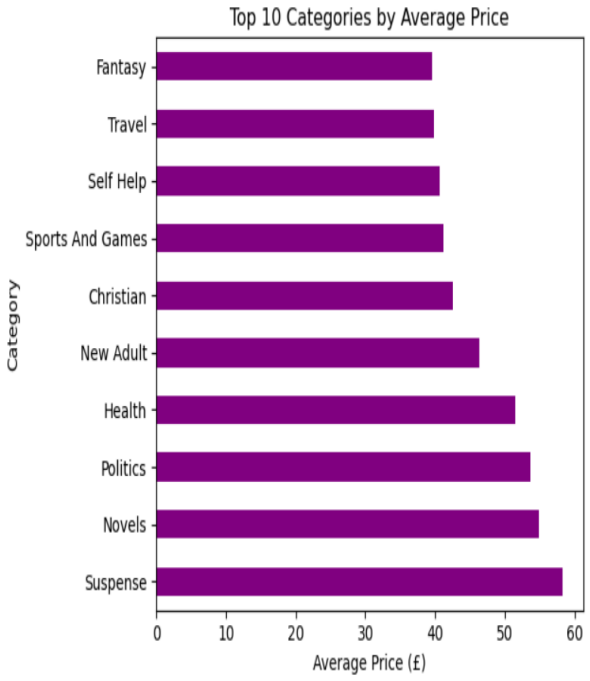
Compare price distribution across different rating:



5] Horizontal Bar Chart:

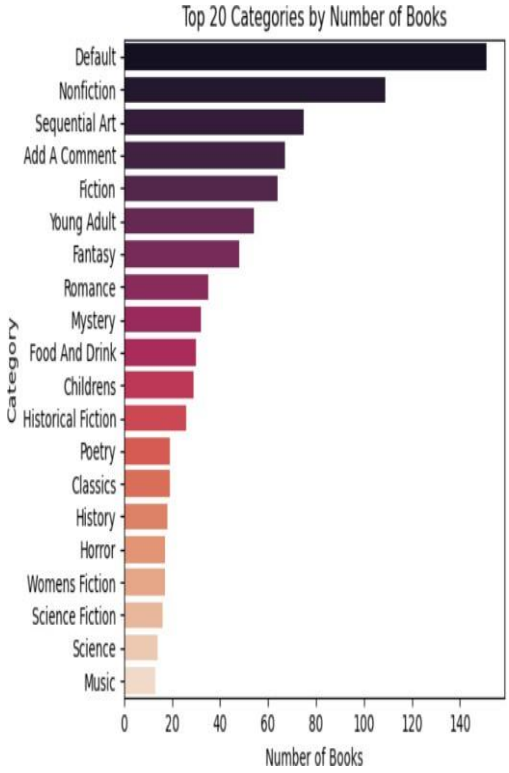
Rank top 10 categories by their average book price:

```
Category
Suspense              58.330000
Novels                54.810000
Politics              53.613333
Health                51.452500
New Adult             46.383333
Christian             42.496667
Sports And Games      41.166000
Self Help             40.620000
Travel                39.794545
Fantasy               39.593958
Name: Price, dtype: float64
```



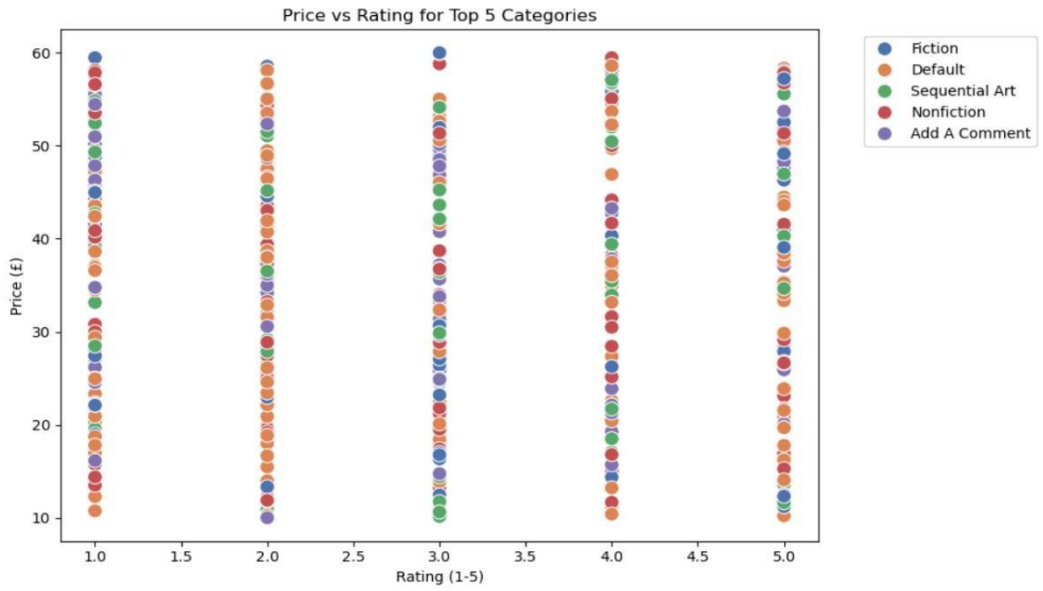
Rank top 20 categories by their book count:

```
Category
Default                151
Nonfiction             109
Sequential Art          75
Add A Comment           67
Fiction                 64
Young Adult             54
Fantasy                 48
Romance                 35
Mystery                 32
Food And Drink          30
Childrens               29
Historical Fiction      26
Poetry                  19
Classics                19
History                 18
Horror                  17
Womens Fiction          17
Science Fiction         16
Science                 14
Music                   13
Name: count, dtype: int64
```



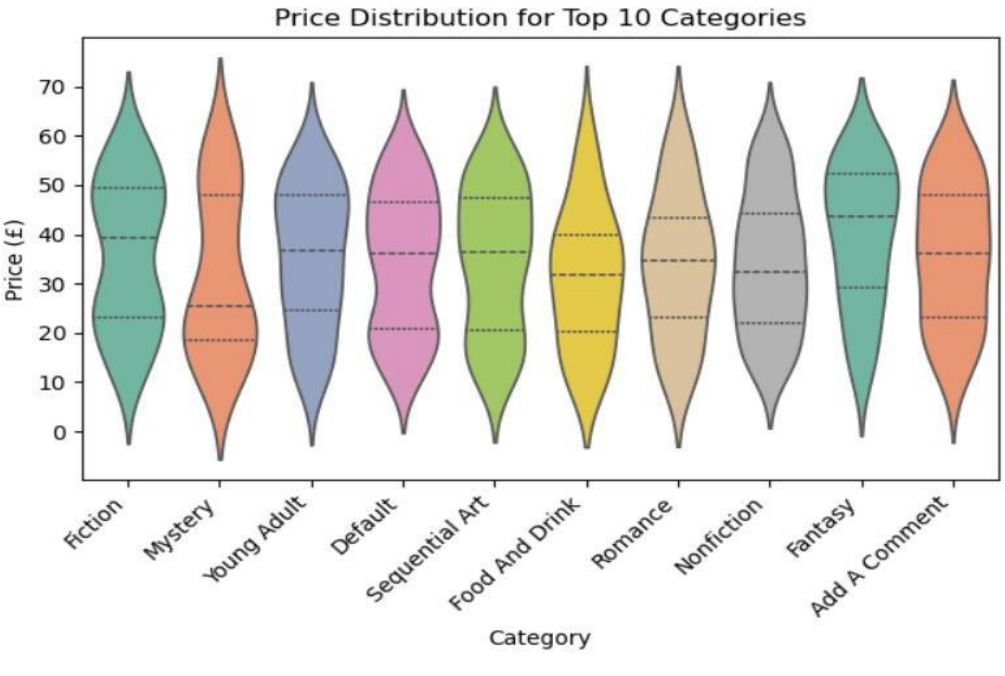
6] Scatter Plot:

Examine relationship between price & rating for top categories:



7] Violin Plot:

Show where most books are priced in each category:

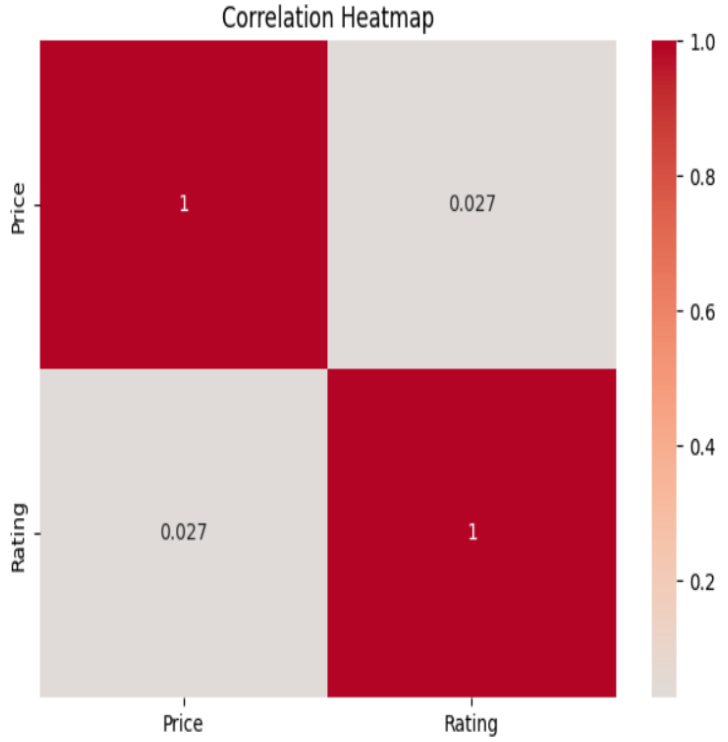


8] Word Cloud:

Display most frequent words in book titles:

9] Heatmap:

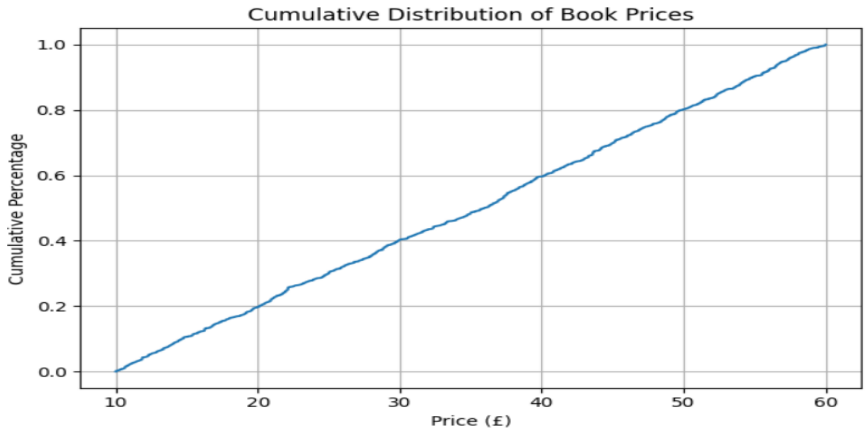
Check if price & rating are correlated:



	Price	Rating
Price	1.000000	0.027128
Rating	0.027128	1.000000

10] Line Plot (Cumulative Distribution):

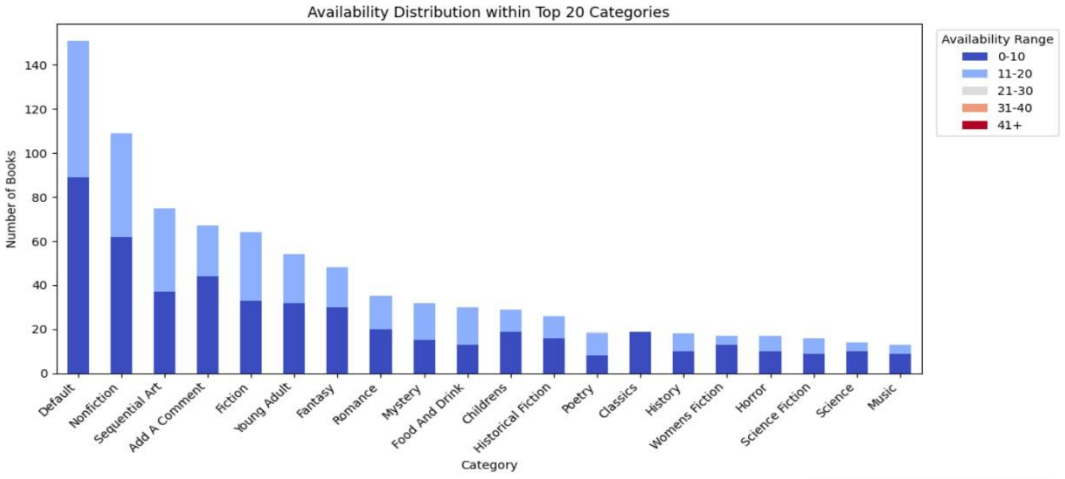
Show what percentage of books fall under a certain price:



11] Stacked Bar Chart :

Show availability distribution within categories:

Availability Range	0-10	11-20	21-30	31-40	41+
Category					
Default	89	62	0	0	0
Nonfiction	62	47	0	0	0
Sequential Art	37	38	0	0	0
Add A Comment	44	23	0	0	0
Fiction	33	31	0	0	0
Young Adult	32	22	0	0	0
Fantasy	30	18	0	0	0
Romance	20	15	0	0	0
Mystery	15	17	0	0	0
Food And Drink	13	17	0	0	0
Childrens	19	10	0	0	0
Historical Fiction	16	10	0	0	0
Poetry	8	10	1	0	0
Classics	19	0	0	0	0
History	10	8	0	0	0



Data Storage Report

---

1. Overview

Following the completion of the data cleaning and analysis processes, the final structured dataset needed to be stored in a reliable and flexible system. For this project, we selected **MongoDB**, a document-oriented NoSQL database, to store the processed data from the eBook store. This storage solution allows for seamless integration with future applications, dashboards, and querying operations.

---

2. Storage Objectives

The primary goals of the data storage phase are:

• Store the cleaned and structured eBook data in a persistent database.
• Enable flexibility for future use cases such as analysis, visualization, or API development.
• Maintain a storage solution that is both scalable and developer-friendly.

---

3. Why MongoDB?

MongoDB was chosen based on its strengths in handling semi-structured data and ease of integration with analytical tools.

a. Document-Based Model

• Stores each book as a JSON-like document, ideal for data with nested or optional fields (e.g., rating, availability).

b. Flexible Schema

• No fixed schema is required, allowing new attributes to be added without breaking the structure.

c. Developer-Friendly

• Simple setup and insertion of data via Python.
• Supports rapid prototyping and updates.

d. Ecosystem Integration

• Easily connects to data analysis libraries and web frameworks, enabling future expansion such as dashboard creation or REST API development.

---

4. Python Library Used: pymongo

To interact with MongoDB, we utilized the **pymongo** library.

Features of pymongo:

• Establishes connection to a MongoDB server.
• Allows insertion, querying, updating, and deletion of documents.
• Supports batch operations and advanced querying for future reporting or filtering.

---

5)Data Storage in MongoDB

1. Database Connection

The process begins by establishing a connection to the local MongoDB server using:

client = Mongo Client('mongodb://localhost:27017/')

This creates a MongoClient object that allows interaction with the MongoDB instance running on the default port 27017.

Next, the database is selected:

db = client['Database']

If the specified database (Database) does not exist, MongoDB will automatically create it when data is inserted.

Then, the target collection within the database is selected:

collection = db['Books']

Similarly, if the Books collection does not exist, MongoDB will create it upon data insertion.

2. Index Creation

An important step is ensuring the uniqueness of records based on a specific field:

collection.create_index('UPC', unique=True)

This command creates a unique index on the UPC field, ensuring that no two documents in the Books collection have the same UPC value. It helps maintain data integrity by preventing duplicate entries.

3. Data Preparation

Before inserting or updating the data, it needs to be prepared:

data = df.to_dict(orient="records")

This command converts the Pandas DataFrame df into a list of dictionaries. Each dictionary represents a record where the keys are the column names and the values are the corresponding data points.

4. Data Insertion or Update

The insertion or update process is handled as follows:

First, the code checks if there is any data in the `data` list.

If data exists, it iterates over each record:

```
for record in data:
    upc_value = record['UPC']
    collection.update_one(
        {'UPC': upc_value},
        {'$set': record},
        upsert=True
    )
```

Explanation:

upc_value = record['UPC']: Extracts the UPC value from the current record.

collection.update_one(...): Performs the update or insert operation:

{'UPC': upc_value}: Defines the search condition to find a document with the same UPC.

{'$set': record}: Updates the document with the fields from the current record. If the document exists, its fields are updated; otherwise, a new document is created.

upsert=True: Ensures that if no matching document is found, a new one is inserted

5. Output Messages

Depending on whether data was available, one of the following messages is printed:

If data was processed:

print("Data was successfully inserted or updated in MongoDB.")

This message confirms the successful completion of the operation.

If no data was available:

print("No data to insert.")

This indicates that there was no new data to process.

Bonus task streamlit

We used **Streamlit** to design a user-friendly web application for analyzing book data. At the beginning, we configured the page layout using **st.set_page_config()** to create a wide and clean interface.

The code starts by reading the dataset (book.csv) into a pandas DataFrame, which serves as the main data source for further analysis throughout the app.

Inside the **sidebar**, we implemented **Streamlit's option_menu** to allow users to easily navigate between different sections: **Home Page**, **View Data**, **Data Analysis**, and **Visualizations**. The sidebar menu was carefully styled to maintain a clean, modern, and easy-to-use layout.

Additionally, we initialized the **st.session_state** to ensure that the app displays the **Home Page** by default when launched. This setup ensures smooth navigation and enhances the overall user experience.

On the **Home Page**, we implemented custom HTML styling with **st.markdown()** to create a welcoming header and a brief project description.

We also provided links to the project files and the original data source, making use of Streamlit's support for HTML content.

In the **View Data** section, we used **st.dataframe()** to display a dynamic table where users can select how many random rows to view using a Streamlit slider.

Additionally, we provided an option to view descriptive statistics of the dataset with a **checkbox** component.

By combining **Streamlit's** layout tools like **st.sidebar**, interactive widgets such as **st.slider, st.checkbox**, and data display functions like **st.dataframe(),** we were able to create an engaging and interactive experience for exploring the book dataset.

In the **Data Analysis** page, we utilized **Streamlit** to create a highly interactive environment where users can explore various types of book data analyses.

First, we used **st.selectbox()** to allow users to select the type of analysis they wish to perform, offering options such as **Descriptive Statistics**, **Trend Identification**, **Patterns Recognition**, and **Clustering Analysis**.

Descriptive Statistics:

Using **st.radio(),** we allowed users to choose between numerical and categorical features.

Depending on the selection, **st.selectbox()** was used to further choose specific columns like **Price** or **Rating**.

important statistical metrics (such as mean, median, min, max, and standard deviation) were displayed using **st.metric(),** organized neatly with **st.columns()** for better layout.

Trend Identification:

We used **st.tabs()** to create separate tabs for exploring **Price Trends** and **Rating Trends** across different categories.

Key findings and correlations were dynamically shown with **st.metric(),** and short summaries were provided based on the correlation strength.

Patterns Recognition and Relationship Analysis:

In this part, we generated a correlation matrix between **Price** and **Rating** using pandas, and visualized it interactively with **st.dataframe()** styled tables.

The strongest relationships were highlighted using .**style.highlight_max()** to make patterns easier to spot.

Clustering Analysis:

We implemented a simple clustering model using KMeans from scikit-learn.

The number of clusters was adjustable through a Streamlit **st.slider().**

After fitting the model, we displayed cluster centers using **st.dataframe(),** and generated a scatter plot of the clustered data with **Matplotlib**, then embedded the plot into the app with **st.pyplot().**

Overall, **Streamlit** components such as **st.selectbox, st.radio, st.slider, st.metric,** st.dataframe, and **st.pyplot** were crucial for building an interactive and visually appealing analysis experience.

This section of the Streamlit app focuses on **visualizations**, offering interactive data exploration and graphical representations of book data. The user can interact with different filters and options to customize the data displayed in various charts and graphs.

Price Distribution:

A **histogram** is displayed showing the distribution of book prices**. Users can filter the data by category and rating** using **st.multiselect().** If no data is available for the selected filters, a warning message appears.

Ranking Categories by Rating:

A distribution of book ratings is shown using a **count plot,** and a **pie chart** displays the rating percentages for the selected category. The **user can choose specific ratings to filter the data**

Price vs Rating by Category:

Users can visualize the relationship between book price and rating for selected categories through a scatter plot. The user can choose one or more categories to compare.

Word Cloud of Book Titles:

A word cloud is generated using the titles of books from **the selected category**. **Users can filter by category, and the word cloud is displayed based on the book titles.**

Streamlit Functions Used:

- **st.subheader()**: To add a subsection title.
- **st.markdown()**: To display rich text .
- **st.multiselect()**: For filtering data based on multiple selection options.
- **st.expander()**: To provide expandable sections for optional filters.
- **st.pyplot()**: To render matplotlib plots.
- **st.selectbox()**: To allow users to choose an option from a dropdown.
- **st.info() and st.warning()**: To show informative or warning messages when no data matches the filters.

This code segment utilizes **matplotlib** for plotting, **seaborn** for enhanced visualizations, and **WordCloud** to generate a word cloud from book titles.
