Book Data Scraping Report

Introduction:

This project focuses on scraping book details from Books to Scrape (http://books.toscrape.com/) using Python, to analyze trends in pricing, ratings and stock availability.

Objective:

- Extract **Title, Pricing, Availability and Product URL** for ~1000 books.
- Store data in structured **CSV file** (books_data.csv).
- Handle errors and missing data.
- Validate extracted data using a **testing framework.**

Technologies Used:

- **Programming language** - Python
- **Web Scraping** – Requests, BeautifulSoup
- **Data Handling** – Pandas
- **Logging** – Python's Logging module
- **Storage** – CSV

Business Flow:

1. **Initiate Scraping** – Start from page 1.
2. **Send HTTP Request** – Validate status code.
3. **Extract Data** – Scrape attributes (Title, Pricing, Rating, Availability, URL).
4. **Handle Pagination** – Move to the next page until no books remain.
5. **Log Errors** – Handle missing fields and HTTP issues.
6. **Store Data** – Save structured output into books_data.csv
7. **Performing Testing** – Verify data integrity, format and presence of errors.

Web Scraping Process:

- The scraper fetches webpage content and extracts book details from structured sections of the website.
- A **rating conversion system** transforms text ratings (One, Two, etc.) into numeric values for consistency.
- The scraper moves through multiple pages dynamically, ensuring **all available books** are collected.

- Extracted data is **formatted, cleaned, and stored** in CSV File.

Implementation and Code Structure

Key Functions and their role:

1. fetch_page(url)
    - Sends an HTTP request to the page.
    - Checks for **response success** (200) or logs errors.
    - Implements **timeouts and Exception handling.**

2. process_books(book_list, books)
    - Extracts book details and structure them into a dictionary.
    - Handles **missing fields and rating conversions**.
    - Appends valid data to main **books** list.

3. scrape()
    - **Loops through all the pages**, extracting book details.
    - Manages **pagination.**
    - Calls helper functions like fetch_page(), process_books().
    - Stops if **no more books or http request fails**.

4. save_to_csv(books)
    - Converts data into a structured **DataFrame** using **Pandas.**
    - Saves cleaned data into books_data.csv.

Error Handling Method:

1. **Missing Data** – Logs error and skips invalid books.
2. **HTTP Errors** – Retries request and logs failure in **scraping_errors.log**.
3. **Unexpected Rating Format** – Defaults to **None** and Records issue.
4. **Empty Response** – **Stops** pagination process.

Logging Example:

```
logging.error(f"Skipping book due to missing data: {e}")
```

Data Storage Structure

The extracted data in stored in CSV File (books_data.csv) as:

Title	Price	Rating	Availability	Product URL
A Light in the Attic	51.77	Three	In stock	a-light-in-the-attic_1000/index.html

Testing and validation

Test Cases covered:

- **Test Case 1:** Verify CSV File Exists (os.path.isfile("books_data.csv")).
- **Test Case 2:** Check File Format (.csv).
- **Test Case 3:** Validate Column Structure (Title, Price, Rating, Availability, URL).
- **Test Case 4:** Ensure Correct Data Types (Price as float).
- **Test Case 5:** Handle Missing or Invalid Data (df.isnull().sum()).

Challenges and Solutions

1. **Handling Timeouts** – Added a timeout requests (response = requests.get(timeout=10)
2. **Rating conversion errors** – Used dictionary mapping (One->1, Two->2, etc)
3. **Avoiding any crashes** – Wrapped functions in try-except blocks

Conclusion

This project successfully implements **structured book data scraping**, ensuring **error resilience and accurate data storage**. By incorporating **robust validation, logging, and testing mechanisms**, the extracted dataset is **reliable and ready for analysis.**