Web Scraping Midterm Project Report

Name: Giorgi Sakhelashvili

Website Chosen: *books.toscrape.com*

The project scrapes books.toscrape.com, a free, public, and intentionally simple website designed for practicing web scraping. It was chosen because:

- Easy to understand explicitly allows scraping and mimics real e-commerce layouts.
- Structured data: clear HTML with consistent classes/IDs, ideal for learning BeautifulSoup.
- No legal/ethical concerns: No login walls, CAPTCHAs, or complex JavaScript rendering.

Implementation Challenges & Solutions

The web scraper follows a modular, object-oriented design to extract book data from books.toscrape.com. The system is divided into four key components, each handling a specific responsibility:

1. Scraper Engine (scraper.py)

The core scraping logic uses BeautifulSoup and requests to navigate pages:

- **Initialization:**
  The BookScraper class sets up a requests Session with headers (user-agent) and a configurable delay (default: 1.5s) to avoid overwhelming the server.

- **Page Fetching:**
  The _get_soup() method handles HTTP requests, applies rate limiting via time.sleep(), and returns parsed HTML. It includes error handling for timeouts/HTTP errors.

- **Data Extraction:**
  - Categories: Scraped from the sidebar menu using find() to navigate nested <ul> tags.
  - Books: Extracted via CSS selectors (e.g., article.product_pod), with prices cleaned using regex in _parse_price().
  - Pagination: Detects "Next" buttons with select_one('li.next > a') and joins URLs dynamically.

- **Detail Scraping:**
  The _scrape_book_details() method visits individual book pages to get descriptions/availability, using sibling navigation (e.g., find_next_sibling('p')).

2. Data Modeling (models.py)

Two classes structure the scraped data:

- **Book:**
  Encapsulates attributes (title, price, and rating) with validation. The RATING_MAPPING converts text ratings (e.g., "Five") to integers. Includes to_dict() for JSON serialization.

- **Category:**
  Manages a collection of books, with methods like get_books_by_rating() to filter results. Uses properties (e.g., book_count) for calculated fields.

3. Storage & Analysis (storage.py)

Handles data persistence and metrics:

- **File Operations:**
  - save_to_csv()/save_to_json() use Python's built-in libraries with error handling for file permissions.
  - Paths are created dynamically with os.makedirs().

- **Data Processing:**
  The process_data() method calculates:
  - Average prices per category
  - Rating distributions (counts of 1–5 stars)

  Results are structured for easy visualization (e.g., charts from analysis.json).

4. Execution Flow (main.py)

Orchestrates the workflow:

1. Initializes BookScraper and DataStorage.
2. Scrapes categories, then iterates through each to extract books.
3. Converts raw data into Book/Category objects.
4. Saves results to CSV/JSON and runs analysis.

Key Design Choices

- Separation of Concerns: Each class/file handles one responsibility (e.g., scraping vs. storage).

- Error Resilience: Try-except blocks guard against missing HTML elements or failed requests.
- Ethical Scraping: Respects robots.txt and mimics human browsing patterns with delays/headers.

The code prioritizes clarity and maintainability—methods are short, typed, and documented. While optimized for accuracy over speed, its modular design allows easy upgrades (e.g., adding concurrency).

Challenges

- **Price Parsing**

Challenge: Extracting prices with the £ symbol (e.g., "£45.17") caused ValueError during float conversion.

Solution: Added a helper function (_parse_price()) using regex to remove non-numeric characters before conversion.

- **Pagination Handling**

Challenge: Categories span multiple pages with dynamic URLs.

Solution: Implemented a while loop to follow "Next" buttons until none remain, joining URLs with urljoin().

- **Missing Data**

Challenge: Some books lacked descriptions or had varying availability text.

Solution: Used try-except blocks and default values (e.g., empty strings) to maintain data consistency.

- **Rate Limiting**

Challenge: Avoiding IP bans while minimizing scrape time.

Solution: Added a configurable delay parameter (1.5s) between requests.

Analysis of Collected Data

The script extracts:

- Basic metrics: all the books of all the categories if given no limits.
- Price analysis: Average price per category (e.g., Travel vs. Fiction).
- Rating distribution: Count of 1–5-star ratings


```
{
    "total_books": 69,
    "categories": [
        {
            "name": "Travel",
            "book_count": 11,
            "average_price": 39.79,
            "rating_distribution": {
                "1": 2,
                "2": 3,
                "3": 3,
                "4": 2,
                "5": 1
            }
        },
        {
            "name": "Mystery",
            "book_count": 32,
            "average_price": 31.72,
            "rating_distribution": {
                "1": 7,
                "2": 5,
                "3": 8,
                "4": 7,
                "5": 5
            }
        },
        {
            "name": "Historical Fiction",
            "book_count": 26,
            "average_price": 33.64,
            "rating_distribution": {
                "1": 5,
                "2": 4,
                "3": 5,
                "4": 4,
                "5": 8
```


Key findings:

- Most books are priced under £40.
- The "Travel" category has the highest average price.
- There are much more books in some categories than others (Mystery > Travel)
- Rating distribution (1-3 or 3-5) really depends on the category

Potential Improvements

1. **Speed Optimization**

   Current: The scraper runs slowly (~1.5s delay/request) but achieves 100% precision (no parsing errors).

   Improvement: Use threading (with fixed worker limits) to scrape multiple pages concurrently while respecting robots.txt or change the DOM or select methods for better optimization

2. **Enhanced Data Processing**
   - Add more functions for better data manipulation and analytics.
   - Scrape additional attributes (e.g., ISBN, publication date).

3. **Error Recovery**

   Resume interrupted scrapes by caching progress (e.g., saving state after each page).

Conclusion

This project demonstrates core web scraping techniques while adhering to ethical guidelines. The trade-off between speed and accuracy ensures reliable data collection, and future optimizations could expand its utility for large-scale analysis.