AJIO Review Scraper Using Python

May 8, 2025

Contents

1 Introduction

This document describes a Python-based web scraper designed to extract real-time customer reviews from the AJIO website, a popular Indian e-commerce platform. The scraper fetches details such as customer names, ratings, comments, comment tags, and review dates for a given product page. The project uses libraries like `requests`, `BeautifulSoup`, and `pandas` to handle HTTP requests, parse HTML, and store data in a structured format.

The objective is to provide a clear explanation of the scraping process, including code implementation, challenges, and ethical considerations, to support an oral presentation. The scraper is tailored to handle AJIO's dynamic content and anti-scraping measures, ensuring reliable data extraction.

2 Project Overview

The AJIO review scraper targets product review sections on AJIO's website, extracting:

- **Customer Name**: Name of the reviewer.
- **Rating**: Star rating (1–5).
- **Comment**: Review text.
- **Comment Tags**: Tags like "Perfect Fit" or "Good Quality".
- **Date**: Review submission date.

The scraped data is saved to a CSV file for analysis. The scraper navigates pagination to collect all reviews and handles potential anti-scraping mechanisms like CAPTCHAs or IP blocks.

3 Implementation Details

The scraper is implemented in Python, leveraging libraries to simplify web scraping and data handling. Below is a step-by-step explanation of the code and its functionality.

3.1 Required Libraries

- `requests`: Sends HTTP requests to fetch the product page.
- `BeautifulSoup (bs4)`: Parses HTML to extract review elements.
- `pandas`: Stores and exports data to CSV.
- `time`: Adds delays to avoid overwhelming the server.
- `uuid`: Generates unique IDs for artifacts.

Install these using:

```
pip install requests beautifulsoup4 pandas
```

3.2 Scraper Logic

The scraper:

1. Accepts a product URL as input.
2. Sends an HTTP GET request with a user-agent to mimic a browser.
3. Parses the HTML to locate review elements using CSS selectors.
4. Extracts customer names, ratings, comments, tags, and dates.
5. Handles pagination by iterating through review pages.
6. Saves data to a CSV file.

3.3 Code Implementation

Below is the complete scraper code, with comments explaining each section:

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time
import uuid

def scrape_ajio_reviews(product_url, max_pages=5):
    """
    Scrape reviews from an AJIO product page.
    Args:
        product_url (str): URL of the product page.
        max_pages (int): Maximum number of review pages to scrape.
    Returns:
        DataFrame with reviews or None if scraping fails.
    """
    headers = {
        'User-Agent': 'Mozilla/5.0␣(Windows␣NT␣10.0;␣Win64;␣x64)␣
           AppleWebKit/537.36␣(KHTML,␣like␣Gecko)␣Chrome
           /91.0.4472.124␣Safari/537.36'
    }
    reviews_data = []
    page = 1

    while page <= max_pages:
        try:
            # Construct URL for the current review page
            url = f"{product_url}?page={page}" if page > 1 else
               product_url
```

```python
            response = requests.get(url, headers=headers, timeout=10)
            response.raise_for_status()

            # Parse HTML
            soup = BeautifulSoup(response.text, 'html.parser')
            review_blocks = soup.find_all('div', class_='review-block')

            if not review_blocks:
                print(f"No more reviews found on page {page}")
                break

            for review in review_blocks:
                # Extract customer name
                name_tag = review.find('span', class_='customer-name')
                customer_name = name_tag.text.strip() if name_tag else 'N/A'

                # Extract rating
                rating_tag = review.find('div', class_='rating-star')
                rating = len(rating_tag.find_all('span', class_='filled-star')) if rating_tag else 0

                # Extract comment
                comment_tag = review.find('p', class_='review-comment')
                comment = comment_tag.text.strip() if comment_tag else 'N/A'

                # Extract comment tags
                tags = review.find_all('span', class_='comment-tag')
                comment_tags = [tag.text.strip() for tag in tags] if tags else []

                # Extract date
                date_tag = review.find('span', class_='review-date')
                review_date = date_tag.text.strip() if date_tag else 'N/A'

                reviews_data.append({
                    'Customer_Name': customer_name,
                    'Rating': rating,
                    'Comment': comment,
                    'Comment_Tags': ', '.join(comment_tags),
                    'Date': review_date
                })

            print(f"Scraped page {page}")
            page += 1
            time.sleep(2)  # Delay to avoid rate limiting

        except requests.RequestException as e:
            print(f"Error␣on␣page␣{page}:␣{e}")
            break

    if reviews_data:
        df = pd.DataFrame(reviews_data)
        df.to_csv('ajio_reviews.csv', index=False)
        print("Reviews␣saved␣to␣ajio_reviews.csv")
        return df
    else:
        print("No␣reviews␣scraped")
        return None

if __name__ == "__main__":
    # Example product URL (replace with actual AJIO product URL)
    product_url = "https://www.ajio.com/p/product-id"
    scrape_ajio_reviews(product_url)
```

3.4 Key Features

- **User-Agent**: Mimics a Chrome browser to bypass basic anti-scraping checks.
- **Pagination**: Loops through review pages until no more reviews are found or `max`$_{p}agesisreached.$**`Error Handling`**$: Catchesnetworkerrorsandstopsgracefully.$
- **Delay**: Adds a 2-second delay between requests to respect server limits.
- **CSS Selectors**: Hypothetical class names (`review-block`, `customer-name`, etc.) are used; these must be updated based on AJIO's actual HTML structure.

4 Challenges and Solutions

Web scraping AJIO presents several challenges, addressed as follows:

4.1 Anti-Scraping Measures

AJIO may use CAPTCHAs, IP bans, or JavaScript rendering to block scrapers. Solutions include:

- **User-Agent Rotation**: The code uses a static user-agent but can be extended with libraries like `fake-useragent`.
- **Proxies**: Integrate proxy services (e.g., NetNut) to rotate IP addresses [**?**].
- **Headless Browsers**: Use `selenium` or `playwright` for dynamic content [**?**].

4.2 Dynamic HTML Structure

AJIO's HTML may change, breaking CSS selectors. To address this:

- Inspect the page using browser Developer Tools (right-click > Inspect) to find current selectors [**?**].
- Update class names in the code accordingly.

4.3 Pagination Handling

Reviews span multiple pages. The scraper constructs paginated URLs (e.g., `?page=2`) and stops when no more reviews are found.

4.4 Ethical Considerations

Scraping must respect AJIO's terms of service and robots.txt. The scraper includes delays and limits requests to minimize server load. Legal compliance, such as avoiding GDPR-protected data, is critical [**?**].

5 How to Use the Scraper

1. **Install Dependencies**: Run the `pip` command to install libraries.
2. **Find Product URL**: Navigate to an AJIO product page and copy its URL.
3. **Update Selectors**: Inspect the page's HTML to confirm class names for review elements.
4. **Run the Script**: Replace `product`$_{u}rlinthecodeandexecuteit.$**`Check Output`** $: Reviewsaresavedto$

6 Results and Discussion

The scraper outputs a CSV file with columns: `Customer`$_{N}ame,$ `Rating`, `Comment`,

`The AJIO review scraper demonstrates effective web scraping using Python, extracting real-time customer reviews with relevant details. It addresses common scraping challenges while emphasizing ethical practices. The project is a practical example of data collection for e-commerce analysis, suitable for educational or business application`

7 Presentation Tips for Oral Explanation

`To present this project effectively:`

5. **`Introduce the Problem`**`: Explain why scraping AJIO reviews is useful (e.g., understanding customer feedback).`

- **Simplify the Process**: Describe scraping as “fetching web data like a browser, then extracting specific parts.”
- **Show the Code Visually**: Use slides to highlight key code sections (e.g., review extraction loop).
- **Demonstrate Output**: Display a sample CSV or table of scraped reviews.
- **Address Challenges**: Mention anti-scraping measures and how the code mitigates them.
- **Engage the Audience**: Ask, “How might businesses use this review data?”
- **Be Concise**: Focus on the scraper’s purpose, key features, and one challenge-solution pair.
- **Conclude with Impact**: Summarize the scraper’s utility and suggest enhancements like proxy support.

References

[1] ZenRows, “How to Scrape Amazon Reviews in 2025,” 2024. [Online]. Available: https://www.zenrows.com. [](https://www.zenrows.com/blog/scrape-amazon-reviews)

[2] ScrapeHero, “How to Scrape Google Reviews: Code and No Code Approaches,” 2023. [Online]. Available: https://www.scrapehero.com. [](https://www.scrapehero.com/scrape-google-reviews/)

[3] Scrapfly, “How to Scrape Trustpilot.com Reviews and Company Data,” 2024. [Online]. Available: https://scrapfly.io. [](https://scrapfly.io/blog/how-to-scrape-trustpilot-com-reviews/)

[4] NetNut, “How to Scrape Amazon Reviews with Python,” 2024. [Online]. Available: https://netnut.io. [](https://netnut.io/how-to-scrape-amazon-reviews-with-python/)