Amazon-IN E-Commerce Scraper Documentation

# 1. Overview of the Scraper

This document outlines the development and functionality of a web scraper designed to extract laptop data from Amazon India (Amazon-IN). The scraper targets specific pin codes (Bangalore - 560001, Delhi - 110001) and extracts detailed product information from multiple pages.

# 2. Scraping Logic and Methodology

The scraper employs a sophisticated strategy to navigate Amazon-IN's web pages, extract relevant laptop data, and handle variations in page structure.

* **amazonscraper.py**

Structure and Purpose: This file contains the core functionality for scraping Amazon's website. It probably defines a class or functions to handle the scraping process.

Methods Used:

1. HTTP Requests: Utilizes Python's requests library to fetch web pages from Amazon.
2. HTML Parsing: Employs selectorLib for parsing HTML content and extracting information.
3. Data Extraction: Methods within this file are expected to target specific HTML elements, extract relevant data like product names, prices, and descriptions.

* **pincodeamazonscraper.py**

Extended Functionality: This file extends or modifies the functionality in amazonscraper.py to handle pin code-specific and multiple pages scraping tasks.

Specialized Methods:

1. Pincodes Handling: Adds functionality to process delivery information based on different pincodes.
2. Multi page Handling: Laptops from multiple pages can be scrapped.
3. Enhanced Data Extraction: Could include additional methods for extracting data that varies by geographical location, such as delivery times and shipping costs.

* **search\_results.yml**

Configuration and Selectors: This YAML file contains configuration settings or selectors specifically for parsing search results.

Usage:

1. Selector Mapping: Maps CSS selectors to various elements in the search results page (like product titles, links, and prices).
2. Dynamic Adaptation: Allows the scraper to adapt to changes in the website’s structure without modifying the Python codebase.

* **selectors.py**

Generic Selectors Configuration: Similar to search\_results.yml, this file likely contains a broader range of CSS selectors for different parts of the Amazon website.

Functionality:

1. Wide Range of Elements: Selectors for a variety of webpage elements, not limited to search results.
2. Flexibility in Scraping: Provides a flexible framework for scraping different types of data from Amazon.

* **General Methodology**

1. Modular Design: The separation of Python code and YAML selectors shows a modular design. The Python scripts handle the logic and mechanics of scraping, while YAML files provide the necessary details for what to scrape.
2. Scalability and Maintenance: This design makes the scraper more scalable and easier to maintain. Changes in the website’s layout can be addressed by updating the YAML files without altering the core scraping logic in the Python scripts.
3. Error Handling and Compliance: Although not directly observable from the content, proper scrapers should include error handling (for issues like network errors or changes in website structure) and ensure compliance with legal and ethical standards, including respecting robots.txt and rate limits. Also, if some data is not present for scrapping then it prints for that respective element.

# 3. Programming Practices

The scraper is developed following Object-Oriented Programming principles, ensuring a robust and scalable structure.

* **Object-Oriented Programming (OOP)**

Class-Based Design: Utilizing classes to encapsulate scraping logic, which allows for easy maintenance and scalability. For instance, AmazonScraper and PincodeAmazonScraper classes defined to handle different aspects of the scraping process.

Method Organization: Methods within these classes are organized to perform specific tasks, such as sending requests, parsing HTML, and extracting data.

* **Modularity and Reusability**

Separation of Concerns: By separating the scraping logic (Python scripts) and the configuration/selectors (YAML files), the codebase becomes more modular. This separation allows for reusing and modifying parts of the code independently.

Reusable Components: Functions and classes are designed to be reusable across different parts of the application, enhancing code efficiency and reducing redundancy.

* **Error Handling and Robustness**

Exception Management: Implementing try-except blocks to handle potential errors that may occur during HTTP requests, parsing, or data extraction.

Reliable Data Extraction: Ensuring the scraper can handle variations in webpage structure and recover gracefully from unexpected webpage changes.

# 4. Data Extraction and Output

This scraper extracts various data fields from Amazon-IN, including SKU id, Product Name, Title, Description, Category, MRP, Selling Price, Discount, Weight, Brand Name, Image URL, Laptop Specifications, Delivery Fee and Delivery Time. The scraper is designed to handle different data structures encountered across various product pages. It also extracts the delivery fee and estimated delivery time for each pincode/location. Maps the extracted data with the categories and subcategories.

Extract all laptops and links are stored in laptop\_urls.txt, the final output which is product details is stored in output\_amazon.ndjson.gz.

Note:

1. SKU/ASIN Identification: The SKU (Stock Keeping Unit) has been identified as the ASIN (Amazon Standard Identification Number).
2. Product Name Field: There was no specific field for the product name, so it has been labeled as "Generic Name".
3. Category Field: In the absence of a distinct category field, this too has been labeled as "Generic Name".
4. Image URL Accessibility: While the image URLs were successfully extracted, accessing these images directly via the links is not possible due to restrictions. The images can only be accessed by visiting the product’s page on Amazon, based on my research.
5. Weight Field Issue: The weight field contains the character '\u200e', which is likely due to '‎' (Left-to-right mark) in the source code causing issues in proper formatting.
6. MRP Extraction Challenges: The MRP (Maximum Retail Price) wasn't being extracted correctly. To resolve this, it became necessary to include 'M.R.P.:' in the extraction process. This adjustment was made because the price was being imported twice, once as the actual price and once as a struck-through price (indicating a discount).
7. Refurbished Products Issue: The scraper does not work effectively for refurbished products because the page structure for these items differs significantly from that of new products.

# 5. Quality Control and Data Stats

Quality control measures are implemented to ensure data accuracy and integrity. Stats of data cover the total count as per pincode/location, as well as not null and null stats.

The quality check was done manually by me by randomly picking any link and comparing its data from the output file. Most of the times the data were correct and errors were handled.

# 6. Challenges

The development of this scraper involved overcoming several challenges, particularly related to dynamic web content and anti-scraping measures.

1. When I ran the scraper on a large scale, Amazon blocked my access to the pages. I tried using multiple VPNs, but I still encountered problems.
2. I encountered difficulties in finding the element IDs and CSS selectors for elements while configuring the YAML file.
3. I attempted to use Selenium, but the screen suddenly froze. Upon my subsequent attempt, I encountered a pop-up message stating, 'Our anti-bot system has detected the presence of other scrapers. Please adhere to the terms and conditions, or your account will be suspended.
4. I also encountered challenges in pincode-based data extraction, and there were very few viable solutions available.

# 7. Improvements

The scraper's development and deployment comply with Amazon-IN's terms of service. But there are few improvements which can be done:

1. It is possible to further improve pincode-based data extraction.
2. There can be a more proper way to implement the extraction of refurbished items.

**General Instructions:**

* amazonscraper.py can be used to run the program to collect small sets of data.
* pincodeamazonscraper.py can be used to run the program to collect large amount of data from multiple pages based on pincode.
* requirements.txt is given in the zip file.
* The output which I got is present in the output folder
  + laptop\_urls.txt contains links of all the laptops
  + output\_amazon.ndjson.gz contains the extracted data
  + output.json is a file which consists the data in a more readable format

Thank you,

SREYANSH BARANWAL