**Amazon Product Scraper – Documentation**  
 **Submitted by:**  
 Shalini Devi R

**Project Description**

The Amazon Product Scraper is a Python-based web scraping tool designed to extract product data from Amazon.in.  
It collects product details including:

1. Product Name
2. Price (₹)
3. Rating
4. Image URL
5. Product URL
6. Date/Time Scraped

This tool is useful for:

* Price comparison and monitoring
* Market research and trend analysis
* Tracking product ratings and availability over time

**Note:** Web scraping Amazon requires caution and adherence to their legal policies to avoid violating their terms of service.

**Technologies Used**

1. **Programming Language**
   * Python 3.10+
2. **Libraries**
   * **Selenium:** Browser automation for dynamic content scraping.
   * **BeautifulSoup:** Parsing HTML content.
   * **Pandas:** Data manipulation and storage (CSV, JSON, Excel).
   * **Datetime:** Recording date and time of scraping.
   * **Random & Time:** Adding human-like delays to avoid detection.
3. **Data Storage**
   * CSV, JSON, and Excel formats.
4. **Browser Automation**
   * Selenium with headless ChromeDriver for dynamic page content.

**Features**

1. **Product Search and Extraction**
   * Scrapes product name, price, rating, image URL, and product URL from Amazon search results.
2. **Filtering**
   * Optional filters for minimum price and minimum rating to extract relevant products.
3. **Save Results**
   * Export scraped data to CSV, JSON, or Excel format for analysis.
4. **Pagination Handling**
   * Automatically scrapes multiple pages of search results.
5. **Scraped On Timestamp**
   * Each product entry includes the date and time when it was scraped.
6. **Human-like Browsing Simulation**
   * Random delays prevent detection and reduce the risk of being blocked.

**Potential Use Cases**

1. **E-commerce Analysts & Retailers**
   * Track competitor product pricing and ratings.
2. **Consumers**
   * Monitor product prices for best deals.
3. **Data Analysts & Researchers**
   * Use structured datasets for visualizations and trend analysis.
4. **Market Researchers**
   * Identify popular products and price fluctuations over time.

**Improvements Over Previous Versions**

1. Added **Scraped On timestamp** for each product.
2. Optional **filters for price and rating**.
3. Handles missing or malformed data gracefully.
4. Random delays to simulate human browsing.

**Source Code (Sample)**

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from bs4 import BeautifulSoup

import pandas as pd

import time

import random

from datetime import datetime  # Added for timestamp

# ---------- Scraper Function ----------

def scrape\_amazon(product, pages, file\_format, filename, min\_price=None, min\_rating=None):

# Setup Chrome options

    options = Options()

    options.add\_argument("--headless")  # Run browser in background

    options.add\_argument("--disable-gpu")

    options.add\_argument("--no-sandbox")

    options.add\_argument("--window-size=1920,1080")

    options.add\_argument("--disable-blink-features=AutomationControlled")  # Avoid detection

    driver = webdriver.Chrome(options=options)

    all\_products = []

    for page in range(1, pages + 1):

        url = f"https://www.amazon.in/s?k={product}&page={page}"

        print(f"\nScraping page {page}: {url}")

        driver.get(url)

# Wait until product containers load

        try:

            WebDriverWait(driver, 10).until(

                EC.presence\_of\_all\_elements\_located(

                    (By.CSS\_SELECTOR, "div[data-component-type='s-search-result']")

                )

            )

        except:

            print(f"No results found on page {page}")

            continue

        soup = BeautifulSoup(driver.page\_source, "html.parser")

        for item in soup.select("div[data-component-type='s-search-result']"):

            # Title

            title = item.h2.get\_text(strip=True) if item.h2 else None

            # Price

            price\_elem = item.select\_one("span.a-price span.a-offscreen")

            if price\_elem:

                try:

                    price = int(price\_elem.get\_text(strip=True).replace("₹", "").replace(",", ""))

                except ValueError:

                    price = None

            else:

                price = None

# Rating

            rating\_elem = item.select\_one("span.a-icon-alt")

            if rating\_elem:

                try:

                    rating = float(rating\_elem.get\_text(strip=True).split()[0])

                except ValueError:

                    rating = None

            else:

                rating = None

# Image URL

            image = item.select\_one("img.s-image")

            image\_url = image["src"] if image else None

            # Product URL

            link\_elem = item.select\_one("h2 a")

            if link\_elem and link\_elem.get("href"):

                product\_url = "https://www.amazon.in" + link\_elem.get("href")

            else:

  # fallback: try any link inside the item

                alt\_link = item.select\_one("a.a-link-normal")

                product\_url = "https://www.amazon.in" + alt\_link.get("href") if alt\_link else None

            if title:

                product\_data = {

                    "Title": title,

                    "Price (₹)": price,

                    "Rating": rating,

                    "Image URL": image\_url,

                    "Product URL": product\_url,

                    "Scraped On": datetime.now().strftime("%Y-%m-%d %H:%M:%S")  # Added timestamp

                }

                # Apply filters if provided

                if min\_price and (price is None or price < min\_price):

                    continue

                if min\_rating and (rating is None or rating < min\_rating):

                    continue

                all\_products.append(product\_data)

        # Random delay to mimic human browsing

        time.sleep(random.uniform(2, 5))

    driver.quit()

# ---------- Save Data ----------

    df = pd.DataFrame(all\_products)

    if file\_format == "csv":

        df.to\_csv(filename + ".csv", index=False, encoding="utf-8-sig")

    elif file\_format == "json":

        df.to\_json(filename + ".json", orient="records", indent=4)

    elif file\_format == "excel":

        df.to\_excel(filename + ".xlsx", index=False)

    else:

        print("Invalid format. Saving as CSV by default.")

        df.to\_csv(filename + ".csv", index=False, encoding="utf-8-sig")

    print(f"\nScraping finished! {len(df)} products saved as {filename}.{file\_format}")

# ---------- Main ----------

if \_\_name\_\_ == "\_\_main\_\_":

    print("\n===== Amazon Scraper =====")

    product = input("Enter product name to search: ").strip().replace(" ", "+")

    pages = int(input("Enter number of pages to scrape: "))

    file\_format = input("Choose file format (csv/json/excel): ").lower().strip()

    filename = input("Enter output file name (without extension): ").strip()

    # Filtering options

    apply\_filter = input("Do you want to filter products? (yes/no): ").lower().strip()

    min\_price = None

    min\_rating = None

    if apply\_filter == "yes":

        price\_input = input("Enter minimum price (or press Enter to skip): ").strip()

        rating\_input = input("Enter minimum rating (1-5, or press Enter to skip): ").strip()

        min\_price = int(price\_input) if price\_input else None

        min\_rating = float(rating\_input) if rating\_input else None

    scrape\_amazon(product, pages, file\_format, filename, min\_price, min\_rating)

**Output Example**

A screenshot of a computer program

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Conclusion**

The Amazon Product Scraper is a robust tool for collecting structured e-commerce product data. It enables:

1. Efficient price and rating monitoring
2. Trend tracking for products over time
3. Easy export for analysis and dashboards

**Future Improvements:**

1. Include product reviews and availability status.
2. Automate periodic scraping for real-time price monitoring.
3. Integrate database storage for long-term collection (MySQL/MongoDB).
4. Use proxies or CAPTCHA handling for robust, large-scale scraping.