Amazon Product Scraper Documentation With Streamlit

Purpose

This Python script is a **Streamlit-based Amazon Product Scraper** that allows users to fetch product details from Amazon India. It uses **Selenium** and **BeautifulSoup** to extract product information such as title, price, ratings, number of reviews, and discounts. Users can enter a product name or an Amazon URL and scrape product details across multiple pages.

Total Time Spent for This Project: - 12Hrs.

How It Works

- **Streamlit UI:** The code uses Streamlit to create a user-friendly interface where users can input a product name or Amazon URL and specify the number of pages to scrape.
- **Selenium WebDriver:** Selenium is used to automate the browser and navigate through Amazon's search results pages.
- **Web Scraping:** The code extracts product details from the HTML structure of Amazon's search results using XPath and BeautifulSoup.
- **Pagination Handling:** The code automatically navigates through multiple pages of search results (if specified by the user).
- **Data Export:** The scraped data is displayed in a table and can be downloaded as a CSV file.

How to Use

1. Run the script in a Python environment with Streamlit:

```
streamlit run script.py
```

2. Enter a **product name** (e.g., "bedsheets") or an **Amazon product URL**.
3. Select the **number of pages** to scrape.
4. Click **Start Scraping** and wait for the process to complete.
5. Download the extracted data as a CSV file.

Requirements

- Python 3.8+
- Google Chrome (latest version)
- Chrome WebDriver (matching the Chrome version)
- Required Python libraries:

```
pip install streamlit selenium beautifulsoup4 pandas
```

Required Technologies

- **Streamlit** - For the web interface.
- **Selenium** - To automate browser interactions.
- **BeautifulSoup** - For parsing HTML and extracting structured data.
- **Pandas** - For storing and processing scraped data.

Code Explanation

1. Import Necessary Libraries

```
import streamlit as st
import time
import random
import csv
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
TimeoutException, StaleElementReferenceException
from bs4 import BeautifulSoup
from urllib.parse import urlencode
```

- **Streamlit** is used to create the web-based UI.
- **Selenium** is used for web scraping by automating browser interactions.
- **BeautifulSoup** is used to parse extracted HTML content.
- **Pandas** is used to store and process the scraped data.

2. Define User Agents (Anti-Detection)

```
USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/91.0.4472.124 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15
(KHTML, like Gecko) Version/14.1.1 Safari/605.1.15",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101
Firefox/89.0",
]
```

- A list of different **user agents** is used to make requests appear as if coming from a real browser.

3. Initialize Selenium WebDriver

```
def initialize_driver():
    user_agent = random.choice(USER_AGENTS)
    options = webdriver.ChromeOptions()
    options.add_argument(f"user-agent={user_agent}")
    options.add_argument("--disable-blink-features=AutomationControlled")
    options.add_argument("--headless")
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")

    driver = webdriver.Chrome(options=options)
    driver.execute_script("Object.defineProperty(navigator, 'webdriver',
{get: () => undefined})")
    driver.implicitly_wait(5)
    return driver
```

- **Configures WebDriver** to run headlessly with random user agents.
- **Prevents bot detection** by modifying browser properties.

4. Build Amazon Search URL

```
def build_amazon_url(product_name, page=1):
    base_url = "https://www.amazon.in/s"
    params = {"k": product_name, "page": page}
    return f"{base_url}?{urlencode(params)}"
```

- Constructs a **search URL** dynamically based on the user input.

5. Scrape Amazon Product Data

```
def scrape_amazon_products(url, max_pages, category):
    driver = initialize_driver()
    driver.get(url)
```

- Scrapes product details from Amazon's search results page. It handles pagination and extracts data such as title, rating, price, etc.

Extracting Product Details:

```
product_containers = driver.find_elements(By.XPATH, "//div[@data-asin and
@data-component-type='s-search-result']")
for container in product_containers:
    try:
        title = container.find_element(By.XPATH, ".//h2/span").text.strip()
    except NoSuchElementException:
        title = "Not Available"
```

- Extracts **product title, rating, number of reviews, price, discount, and ASIN.**
- Uses **XPath selectors** to locate elements.

Handling Pagination:

```
try:
    next_button = WebDriverWait(driver, 5).until(
        EC.element_to_be_clickable((By.XPATH, "//a[contains(@class, 's-
pagination-next')]"))
    )
    if "s-pagination-disabled" in next_button.get_attribute("class"):
        break
    next_button.click()
    WebDriverWait(driver, random.uniform(2, 4)).until(
        EC.presence_of_element_located((By.XPATH, "//div[@data-asin and
@data-component-type='s-search-result']"))
    )
except (NoSuchElementException, TimeoutException,
StaleElementReferenceException):
    break
```

- Clicks **next page** button to continue scraping.
- Stops if no more pages are available.

6. Display Data in Streamlit UI

```
df = pd.DataFrame(scraped_data, columns=["Category", "Title", "Rating",
"Number of Ratings", "Current Price", "Actual Price", "Price Per Unit",
"Last Month Purchase", "Discount", "ASIN"])
st.dataframe(df)
```

- Displays the extracted product data in **tabular format**.

7. CSV Download Button

```
csv = df.to_csv(index=False).encode('utf-8')
st.download_button(label="Download CSV", data=csv,
file_name="amazon_products.csv", mime="text/csv", key="download_csv")
```

- Allows users to download the scraped data as a **CSV file**.
