WEB SCRAPPING AMAZONE PRODUCTS

Revolutionizing Data Mining: Unveiling the Best Gaming Laptops on Amazon!

In this cutting-edge report, we delve into the world of web scraping, using Python's powerful libraries like Selenium and BeautifulSoup. Journey with us as we unveil the secrets of automating web browsers and navigating through Amazon's vast gaming laptop inventory. Our expertly crafted code scrapes data from multiple pages, unveiling a treasure trove of gaming laptops' vital details, including titles, prices, product URLs, image URLs, and ratings. We even uncover brand names cleverly hidden in the product titles! Dive into the world of web scraping and witness the magic of data extraction as we showcase the fascinating results in a neatly organized JSON format. Uncover valuable insights from a representative sample of Amazon's 3000 gaming laptop offerings. Get ready to unlock the potential of web scraping in your data analysis arsenal. Embark on a data-driven adventure and harness the power of information with our captivating code and results.

Contents

Step 1: Importing Required Libraries

```python
from selenium import webdriver
from bs4 import BeautifulSoup
import json
import os
```

In this step, we import the necessary libraries for web scraping and data manipulation:

- ***selenium***: Used for automating web browsers, allowing us to interact with and scrape web pages.
- ***BeautifulSoup***: Used for parsing and navigating HTML pages, making it easy to extract data from the web page source.
- ***Json***: Used for working with JSON data, allowing us to save the scraped data in JSON format.
- ***os***: Used for interacting with the operating system, helping us get the current directory path.

Step 2: Setting up Web-driver (Chrome Browser Automation)

```python
# Get the path to the current directory and chromedriver.exe
current_directory = os.path.dirname(os.path.abspath(__file__))
chromedriver_path = os.path.join(current_directory, 'chromedriver.exe')

# Launch the Chrome browser using webdriver and specify the chromedriver pa
driver = webdriver.Chrome(executable_path=chromedriver_path)
```

We set up the web-driver to automate the Chrome browser for web scraping:

- We get the path to the current directory where the script is located using *os.path.dirname* and *os.path.abspath*.
- We combine the current directory path with the filename 'chromedriver.exe' to get the full path to the *chromedriver* executable.
- We launch the Chrome browser using *webdriver.Chrome()* and specify the path to the *chromedriver* using the *executable_path* argument.

Step 3: Defining the Amazon URL and Data Variables

```
amazon_url = "https://www.amazon.com/s?k=gaming+laptop"

# Set the maximum number of pages to scrape (adjust as needed)
max_pages = 5

# Set the number of results per page to scrape (adjust as needed)
results_per_page = 10

product_data = []
```

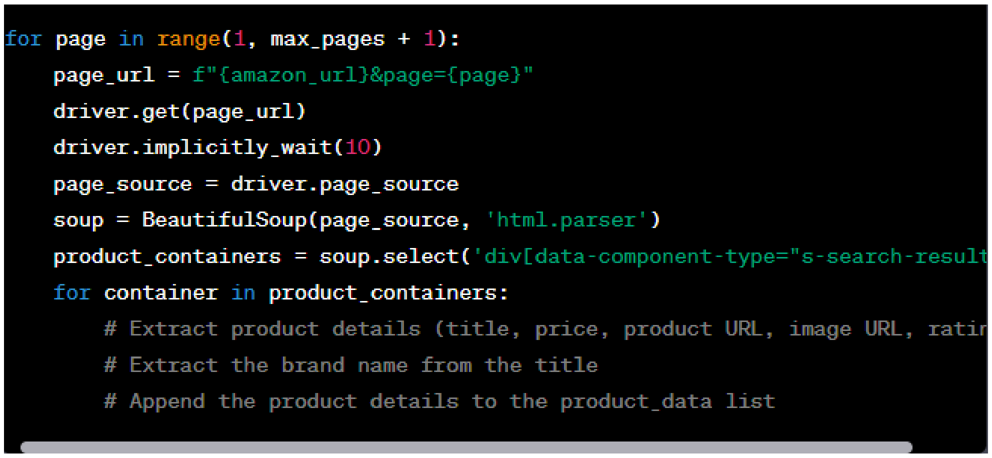
We define the URL of the Amazon search page for gaming laptops.

We set the *max_pages* variable to 5, which means we want to scrape data from a maximum of 5 pages of search results.

The *results_per_page* variable is set to 10, indicating that we want to extract 10 results per page.

We create an empty list called *product_data* to store the scraped product information.

Step 4: Scraping Data from Multiple Pages



```python
for page in range(1, max_pages + 1):
    page_url = f"{amazon_url}&page={page}"
    driver.get(page_url)
    driver.implicitly_wait(10)
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    product_containers = soup.select('div[data-component-type="s-search-result
    for container in product_containers:
        # Extract product details (title, price, product URL, image URL, ratir
        # Extract the brand name from the title
        # Append the product details to the product_data list
```

We use a for loop to iterate through page numbers from 1 to *max_pages*. This allows us to navigate through multiple pages of search results.

For each page, we construct the URL with the page number and load the page using *driver.get(page_url).* This allows us to load and view different pages.

We use *driver.implicitly_wait(10)* to wait for the page to load and dynamic content to populate. This ensures that the web page is fully loaded before we start extracting data.

We use *driver.page_source* to get the HTML source of the page after dynamic content has loaded.

We use BeautifulSoup to parse the page source and create a BeautifulSoup object soup that allows us to navigate and extract data from the HTML.

We use *soup.select('div[data-component-type="s-search-result"]')* to find all the product containers on the page. These containers hold the information about each product listed in the search results.

For each product container, we extract the product details such as title, price, product URL, image URL, and rating. We also attempt to extract the brand name from the title by splitting the title and taking the first word as the brand name.

We append the extracted product details to the product_data list for each product container.

Step 5: Closing the Web-driver

```
driver.quit()
```

After scraping all the required data, we close the browser to free up system resources using *driver.quit()*.

Step 6: Saving Data to JSON File

```
with open('amazon-search.json', 'w', encoding='utf-8') as file:
    json.dump(product_data, file, ensure_ascii=False, indent=2)
```

Finally, we save the scraped data stored in the product_data list to a JSON file named *'amazon-search.json'*.

We use *json.dump()* to write the data to the file with indentation for better readability.

The *ensure_ascii=False* parameter ensures that non-ASCII characters are encoded properly in the JSON file.