

Project Title: Web Scrapping Amazon Website Using Selenium

Problem Statement:

In the current digital age, e-commerce websites like Amazon hold a vast amount of product information that can be extremely useful for market analysis, competitive research, and consumer insights. For this project, we aim to develop a robust web scraping tool to extract specific data about laptops from Amazon's website. The tool will be designed to gather essential product information, including the product name, price, rating, and the number of raters, and convert the collected data into both Excel (XLSX) and CSV formats for ease of analysis and reporting.

Objectives:

1. **Web Scrapping Setup:** Utilize Selenium to navigate and scrape data from Amazon's website.
2. **Data Extraction:** Extract specific details for each laptop listing
 - Product Name
 - Price
 - Rating
 - No of Raters
3. **Error Handling:** Implement error handling mechanisms to manage common issues such as incorrect username or password during login (if necessary).
4. **Data Storage:** Convert and store the scraped data into both Excel (XLSX) and CSV formats.

Scope and Limitations:

1. **Scope:** The scraper will focus solely on laptop listings on Amazon. It will gather data from search results and product pages.

2. Limitations:

- The project will adhere to Amazon's terms of service and robots.txt file.
- Due to the dynamic nature of web pages and potential changes to Amazon's structure, continuous maintenance and updates to the scraper may be necessary.
- The scraper will handle basic error conditions but may require additional features for comprehensive error management and captcha bypassing.

Technical Requirements:

1. Programming Language: Python

2. Libraries and Tools:

- **Selenium:** For browser automation and web scraping.
- **Pandas:** For data manipulation and conversion.
- **Openpyxl or xlswriter:** For writing data to Excel format.
- **Streamlit:** For getting the error message while error handling

3. Environment:

- Web driver (e.g., Chrome Driver for Google Chrome)
- Amazon account credentials for login (if scraping requires authentication)

Importing Libraries:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import pandas as pd
from selenium.common.exceptions import NoSuchElementException, TimeoutException
import streamlit as st
```

1. from selenium import webdriver

- Imports the 'webdriver' module from Selenium, which allows us to interact with web browsers programmatically.

2. from selenium.webdriver.chrome.service import Service

- Imports the 'Service' class, which is used to manage the ChromeDriver service required for Selenium to control the Chrome browser.

3. from selenium.webdriver.common.by import By

- Imports the 'By' class, which provides a set of locator strategies (like ID, XPATH, CSS_SELECTOR, etc.) to find elements on a web page.

**4. from selenium.webdriver.support.ui import
WebDriverWait**

- Imports WebDriverWait, which allows us to wait for certain conditions (like element visibility) before proceeding with actions.

**5. from selenium.webdriver.support import
expected_conditions as EC**

- Imports a module containing common conditions used with WebDriverWait to wait for specific states (like an element being clickable).

6. import time

- Imports the 'time' module, which provides functions to handle time-related tasks, such as adding delays with 'time.sleep ()'.

7. import pandas as pd

- Imports the pandas library, which is a powerful tool for data manipulation and analysis, especially for handling tabular data.

**8. from selenium.common.exceptions import
NoSuchElementException, TimeoutException**

- **NoSuchElementException:** Raised when an element is not found on the page.
- **TimeoutException:** Raised when an operation times out.

9. import streamlit as st

- Imports Streamlit, a framework for creating interactive web applications in Python. This allows us to build a web-based user interface for the scraper.

Login and Error Handling of the Webpage:

```
try:
    sign_in_button = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.XPATH, "/html/body/div[1]/header/div/div[1]/div[3]/div/a[2]"))
    )
    sign_in_button.click()

    # Enter username
    try:
        username_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.ID, 'ap_email'))
        )
        username_field.send_keys("*****")
        driver.find_element(By.XPATH, "//input[@id='continue']").click()

        # Enter password
        try:
            password_field = WebDriverWait(driver, 10).until(
                EC.presence_of_element_located((By.ID, 'ap_password'))
            )
            password_field.send_keys("*****") # Provide incorrect password intentionally
            driver.find_element(By.XPATH, "//input[@id='signInSubmit']").click()

            # Check for incorrect password error message
            try:
                error_message = WebDriverWait(driver, 5).until(
                    EC.visibility_of_element_located((By.XPATH, "//div[@id='auth-error-message-box']/div[@class='a-box-inner']"))
                ).text
                if "Your password is incorrect" in error_message:
                    st.error("Incorrect password provided. Please try again with the correct password.")
            except TimeoutException:
                st.error("Error: Timeout occurred while checking for incorrect password message.")

        except NoSuchElementException:
            st.error("Error: Password field not found.")

    except NoSuchElementException:
        st.error("Error: Username field not found.")

except TimeoutException:
    st.error("Error: Timeout occurred while waiting for sign-in button.")
```

The provided code snippet demonstrates a robust error-handling mechanism for automating the login process to an Amazon account using Selenium. It first attempts to click the sign-in button and waits up to 10 seconds for it to become clickable. Upon success, it enters the username, clicks the continue button, and then enters the password. The code includes nested try-except blocks to handle potential issues such as elements not being found (`NoSuchElementException`) or operations timing out (`TimeoutException`). Additionally, it checks for an incorrect password error message, providing user feedback through Streamlit if the login attempt fails due to an incorrect password. This comprehensive approach ensures that the process is resilient to common errors and provides informative feedback to the user.

Scrapping Data from the Amazon Website:

```
# Lists to store data
headings = []
prices = []
ratings = []
num_ratings = []
```

```
# Function to process product links and collect data
def process_product_links():
    # Wait for the product list to load
    product_links = WebDriverWait(driver, 10).until(
        EC.presence_of_all_elements_located((By.XPATH, "//h2/a"))
    )

    # Iterate over product links
    for product_link in product_links:
        try:
            driver.execute_script("arguments[0].scrollIntoView();", product_link) # Scroll to the product link
            driver.execute_script("arguments[0].click();", product_link)
            time.sleep(2) # Give time for the new tab to open
            original_window = driver.current_window_handle

            # Switch to the new window
            for handle in driver.window_handles:
                if handle != original_window:
                    driver.switch_to.window(handle)
                    break

            # Extract product details
            heading = WebDriverWait(driver, 5).until(
                EC.presence_of_element_located((By.ID, "productTitle"))
            ).text
```

```
price_element = None
price_xpaths = [
    "//span[@id='priceblock_ourprice']",
    "//span[@id='priceblock_dealprice']",
    "//span[contains(@class, 'priceToPay')]",
    "//span[contains(@class, 'priceBlockBuyingPriceString')]",
    "//span[contains(@class, 'a-size-medium a-color-price priceBlockBuyingPriceString')]"
]
for xpath in price_xpaths:
    elements = driver.find_elements(By.XPATH, xpath)
    if elements:
        price_element = elements[0]
        break

price = price_element.text if price_element else "N/A"

# Extract rating
try:
    rating_element = WebDriverWait(driver, 5).until(
        EC.presence_of_element_located((By.XPATH, "//span[@id='acrPopover']"))
    )
    rating = rating_element.get_attribute("title")
except:
    rating = "N/A"
```

```
# Extract number of ratings
try:
    num_raters_element = WebDriverWait(driver, 5).until(
        EC.presence_of_element_located((By.XPATH, "//span[@id='acrCustomerReviewText']"))
    )
    num_raters = num_raters_element.text
except:
    num_raters = "N/A"

# Append to lists
headings.append(heading)
prices.append(price)
ratings.append(rating)
num_ratings.append(num_raters)

# Close the new window and switch back to the original window
driver.close()
driver.switch_to.window(original_window)
time.sleep(2) # Ensure the switch is complete before continuing
except Exception as e:
    print(f"Error processing link: {e}")
    continue
```

The code snippet establishes a methodical approach to scrape specific data about laptop products from Amazon using Selenium. Initially, it defines four lists (headings, prices, ratings, and num_ratings) to store the extracted data. The main function, process_product_links, is designed to navigate through the list of product links on the Amazon search results page.

1. **Loading Product Links:** The function begins by waiting for the product links to become present on the page using `WebDriverWait` with a specified timeout of 10 seconds, ensuring that the scraper only proceeds once the elements are fully loaded.
2. **Iterating Over Links:** It iterates over each product link. For each link, the code:
 - Scrolls the product link into view and clicks on it to open the product details in a new tab.
 - Switches the Selenium driver to the new browser tab to interact with the newly opened product page.
3. **Extracting Product Details:**
 - **Product Title:** It waits for the product title element to be present and retrieves its text.
 - **Price:** The function attempts to locate the price using multiple XPaths, catering to different price display formats Amazon might use. If a price is found, it stores the text; otherwise, it marks the price as "N/A".
 - **Rating:** It tries to find the rating element and retrieve its "title" attribute. If unsuccessful, it assigns "N/A".
 - **Number of Ratings:** Similarly, it looks for the element displaying the number of ratings. If not found, it assigns "N/A".
4. **Storing Data:** Extracted data (title, price, rating, number of ratings) is appended to the respective lists.
5. **Tab Management:** After collecting data from a product page:
 - The scraper closes the new tab.
 - Switches back to the original search results tab.
 - Pauses briefly to ensure the switch is complete and avoid timing issues.
6. **Error Handling:** Throughout the process, the function includes try-except blocks to handle potential errors gracefully:
 - If an error occurs while processing a product link (such as elements not found or timeouts), it catches the exception, prints an error message, and continues with the next link,

ensuring the scraper is resilient and continues running even if individual product pages have issues.

Pagination Handling for Amazon Product Scraping:

```
page_number = 1
while True:
    process_product_links()
    try:
        if page_number == 1:
            next_button_xpath = "/html/body/div[1]/div[1]/div[1]/div[1]/div/span[1]/div[1]/div[30]/div/div/span/a[3]"
        elif page_number in range(2,5):
            next_button_xpath = f"/html/body/div[1]/div[1]/div[1]/div[1]/div/span[1]/div[1]/div[29]/div/div/span/a[{page_number + 2}]"
        else:
            next_button_xpath = "/html/body/div[1]/div[1]/div[1]/div[1]/div/span[1]/div[1]/div[29]/div/div/span/a[5]"

        next_button = WebDriverWait(driver, 15).until(
            EC.element_to_be_clickable((By.XPATH, next_button_xpath))
        )
        driver.execute_script("arguments[0].scrollIntoView();", next_button)
        driver.execute_script("arguments[0].click();", next_button)
        print(f"Clicked on next button for page {page_number}")
        time.sleep(10)
        page_number += 1
    except Exception as e:
        print(f"No more pages or error: {e}")
        break
# Close the driver
driver.quit()
```

The provided code snippet effectively manages pagination to continuously scrape product data from multiple pages on Amazon. Starting with an initial page number of 1, it employs a `while True` loop to repeatedly call the `process_product_links()` function, which extracts product details such as the title, price, rating, and number of ratings. For navigation, the script determines the appropriate XPath for the "Next" button based on the current page number: it uses different XPaths for the first page, pages 2 through 4, and pages beyond the fourth to account for changes in the button's location. After identifying the correct "Next" button, it waits until the button is clickable, scrolls it into view, and clicks it to move to the next page. If the button click is successful, it prints a confirmation message and pauses briefly to allow the next page to load. The loop increments the page number and continues this process until no more pages are found or an error occurs, at which point it breaks out of the loop and closes the Selenium WebDriver. This approach ensures comprehensive and systematic data extraction across multiple pages while gracefully handling navigation and potential errors.

Saving Scraped Data to CSV and Excel:

```
# Create DataFrame
df = pd.DataFrame({
    'Heading': headings,
    'Price': prices,
    'Rating': ratings,
    'No Of Raters': num_ratings
})

# Save DataFrame to CSV file
df.to_csv('amazon_laptops.csv', index=False)

# Save DataFrame to Excel file
df.to_excel('amazon_laptops.xlsx', index=False)

print(df)
```

The final step of the script involves consolidating the scraped laptop data into a structured format and saving it to files for further analysis. Using the pandas library, a DataFrame is created from the lists of headings, prices, ratings, and number of raters. This DataFrame is then saved to two different file formats: a CSV file named `amazon_laptops.csv` and an Excel file named `amazon_laptops.xlsx`. Both files exclude the DataFrame index to keep the output clean and focused on the relevant data. This approach ensures that the collected data is easily accessible and ready for analysis or reporting.

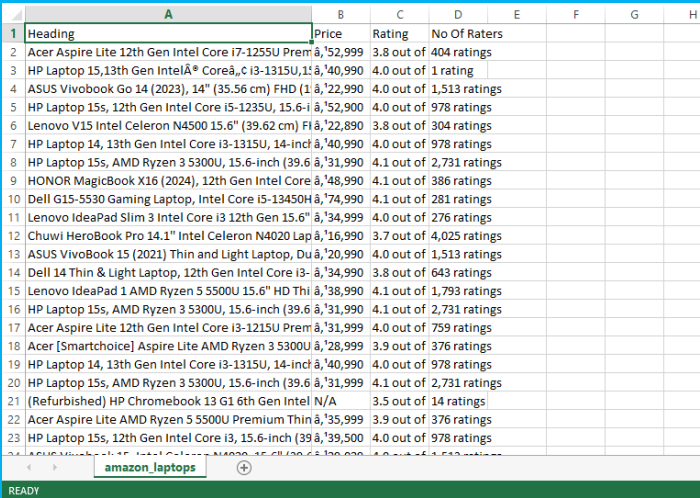
Challenges Faced During Web Scraping with Selenium:

While developing the web scraper for Amazon laptop listings, I encountered several challenges that required careful handling and adaptive coding techniques. One significant issue was the variability in the XPath paths for extracting prices, ratings, and the number of raters. Amazon uses multiple formats to display prices, necessitating the implementation of a flexible search through various XPath paths to ensure all possible price formats were captured. Similarly, the elements for ratings and the number of raters were not consistently located, requiring additional logic to identify and extract these values accurately. Another complex aspect was managing pagination, where the "Next" button's XPath changed based on the current page. Specifically, the XPath for the "Next" button on the first page differed from those on pages 2 to 4, and from page 5 onward, yet another XPath was used. Handling these dynamic XPath paths required a tailored approach to ensure the scraper could navigate through all pages seamlessly. These challenges underscored the importance of creating a resilient and

adaptable scraper capable of handling the dynamic nature of web elements on e-commerce platforms like Amazon.

Sample Data of Extracted Laptop Information Saved to CSV:

Below is a sample image showcasing the extracted laptop data that has been saved to a CSV file. This data includes the headings, prices, ratings, and the number of raters for each laptop product scraped from Amazon.



	A	B	C	D	E	F	G	H
1	Heading	Price	Rating	No Of Raters				
2	Acer Aspire Lite 12th Gen Intel Core i7-1255U Premi	\$52,999	3.8 out of	404 ratings				
3	HP Laptop 15,13th Gen Intel® Coreâ„¢ i3-1315U,15.6	\$40,990	4.0 out of	1 rating				
4	ASUS Vivobook Go 14 (2023), 14" (35.56 cm) FHD (1.9	\$22,990	4.0 out of	1,513 ratings				
5	HP Laptop 15s, 12th Gen Intel Core i5-1235U, 15.6-i	\$52,900	4.0 out of	978 ratings				
6	Lenovo V15 Intel Celeron N4500 15.6" (39.62 cm) Fi	\$22,890	3.8 out of	304 ratings				
7	HP Laptop 14, 13th Gen Intel Core i3-1315U, 14-inc	\$40,990	4.0 out of	978 ratings				
8	HP Laptop 15s, AMD Ryzen 3 5300U, 15.6-inch (39.6	\$31,990	4.1 out of	2,731 ratings				
9	HONOR MagicBook X16 (2024), 12th Gen Intel Core â	\$48,990	4.1 out of	386 ratings				
10	Dell G15-5530 Gaming Laptop, Intel Core i5-13450H	\$74,990	4.1 out of	281 ratings				
11	Lenovo IdeaPad Slim 3 Intel Core i3 12th Gen 15.6"	\$34,999	4.0 out of	276 ratings				
12	Chuwi HeroBook Pro 14.1" Intel Celeron N4020 Lap	\$16,990	3.7 out of	4,025 ratings				
13	ASUS VivoBook 15 (2021) Thin and Light Laptop, Du	\$20,990	4.0 out of	1,513 ratings				
14	Dell 14 Thin & Light Laptop, 12th Gen Intel Core i3-	\$34,990	3.8 out of	643 ratings				
15	Lenovo IdeaPad 1 AMD Ryzen 5 5500U 15.6" HD Thi	\$38,990	4.1 out of	1,793 ratings				
16	HP Laptop 15s, AMD Ryzen 3 5300U, 15.6-inch (39.6	\$31,990	4.1 out of	2,731 ratings				
17	Acer Aspire Lite 12th Gen Intel Core i3-1215U Premi	\$31,999	4.0 out of	759 ratings				
18	Acer [Smartchoice] Aspire Lite AMD Ryzen 3 5300U	\$28,999	3.9 out of	376 ratings				
19	HP Laptop 14, 13th Gen Intel Core i3-1315U, 14-inc	\$40,990	4.0 out of	978 ratings				
20	HP Laptop 15s, AMD Ryzen 3 5300U, 15.6-inch (39.6	\$31,999	4.1 out of	2,731 ratings				
21	(Refurbished) HP Chromebook 13 G1 6th Gen Intel	N/A	3.5 out of	14 ratings				
22	Acer Aspire Lite AMD Ryzen 5 5500U Premium Thin	\$35,999	3.9 out of	376 ratings				
23	HP Laptop 15s, 12th Gen Intel Core i3, 15.6-inch	\$39,500	4.0 out of	978 ratings				
24	ASUS VivoBook 15 Pro 15.6" Intel Core i5-12500H	\$519,000	4.0 out of	1,513 ratings				

Conclusion:

This project will provide a valuable tool for extracting detailed laptop product information from Amazon, facilitating better market analysis and decision-making. By converting the data into easily accessible formats (XLSX and CSV), users can efficiently utilize the information for various analytical purposes.