Project: Scrapping pepper.pl

Participants:
- Adam Janczyszyn, 419350
- Jakub Wujec, 420463
- Jakub Żmujdzin, 420602

Description of topic:

Pepper.pl is a Polish online platform dedicated to community-driven bargain hunting. The website aggregates and showcases various deals, discounts, coupons, and promotions available online and offline, submitted by its active user base. It covers a broad range of categories including but not limited to electronics, fashion, groceries, and software.

Pepper.pl also operates as a social commerce website where users can share and discuss the best deals and promotional offerings found from a multitude of sources. The platform emphasizes user interaction and contribution. Community members can post deals they've found, comment on others' deals, vote on deal temperatures (a metric indicating how good a deal is), and engage in discussions about products or brands.

We would like to scrape the information about bargains and have some way of filtering out e.g. only categories that interest us, look at historical deals for certain products and determine whether this particular bargain is the best in the last x months.

We have omitted the static scrapping of a web page due to three issues:
- One need to accept cookies to scrape website
- Basic beautiful soup does not allow for scrapping the website as-is
- We wanted to extract bargains for a particular user

Description of scraper mechanics:

We have two scrapers:
- Selenium
- Scrapy Splash

Both of the scrappers operate do the following things:
Load page, log in using user credentials, scrape x main pages selected by user that contains bargain info, then scrape all of the bargains.
The main operations are done in a similar manner, although the logging in differs. In case of scrapy + selenium, we input the login, password and click the accept button using selenium. We

get the credentials from user input. In scrapy splash, we use a Lua script that gets user credentials from environment variables.

Selenium:

By combining Selenium with BeautifulSoup, the code of a scrapper enables efficient navigation and data extraction. The main functionality is divided into two classes: PepperLinksScrapper and PepperScrapper. The PepperLinksScrapper class initializes a Selenium WebDriver, interacts with the website's interface by clicking on the "Continue without accepting" button, and logs in to Pepper.pl. It also sets up the initial state for data scraping, such as storing the current page and creating a DataFrame to hold the scraped data. The class includes methods to extract the columns for the scraped data, derive HTML content from the WebDriver into a BeautifulSoup object, scan the current page for content, extract content from individual articles, and scan subsequent pages for more data. The PepperScrapper class works in conjunction with a WebDriver and takes a links file as input. It reads the links from the file and then proceeds to scrape additional data for each link. The class includes methods to scrape data from a single link, extracting information such as the title, category, hotness, number of comments, prices, and place of bargain price. The scraped data is stored in a DataFrame. In the main section of the code, a Chrome WebDriver is created, and the initial page is loaded. The PepperLinksScrapper is then used to extract links from the loaded page, and the PepperScrapper is employed to scrape additional data for each link. The final scraped data is saved to a CSV file, providing a comprehensive dataset for further analysis

Splash:

This script performs two key operations: URL collection and data extraction from pepper.pl. Firstly, it uses Scrapy, a Python web scraping library, to collect URLs from articles on the website, handling the first five pages. After storing these URLs, the second part of the script logs into the website using environment variables for authentication, leveraging Splash, a scriptable browser service, and a Lua script to perform the login. Successful login leads to extraction of specific data points from each URL, including title, category, "hotness" score, comments count, and various price information. This extraction process utilizes Beautiful Soup, a Python library for pulling data out of HTML and XML files, along with regular expressions. The script gracefully handles errors in data extraction, providing default values when data is missing. Each set of scraped data is then yielded as an item, ready for further processing or storage. It's important to note that the script requires valid login credentials for pepper.pl, and web scraping should comply with the target website's terms of service and applicable laws and regulations.

The main difference between scrapers is that when using scrapy splash, it came out to a few seconds (about 10 links per second). On the other hand, using selenium (and having to wait for all of the pages to load in an iterative manner) allows us to scrape 1 link per second per browser client.

Output:

We have two file outputs:

- First file (scrapped from main pages) consists of links to bargains
- Second file (scrapped from bargains) consists of information about bargain:
    - Category
    - Number of comments
    - Hotness
    - Percentage change in price
    - Website of bargain
    - Price after discount
    - Price before discount
    - Title

Data Analysis Summary

The collected data provides valuable insights into the trending bargains across different categories. Each data point serves a unique purpose in understanding consumer behavior, market trends, and the overall effectiveness of the discounts provided. Here's how each piece of data can be beneficial:

Category: This helps to understand which product categories are trending in the market. By analyzing the frequency of bargains in each category, we can determine which products are most often discounted and may infer their popularity or competitive landscape.

Number of comments: This reflects customer engagement with the bargain. Higher numbers could indicate higher consumer interest, which could lead to increased sales. It also provides a platform for sentiment analysis to gauge customer reactions to the bargain.

Hotness: This measure could be an indicator of the popularity of a bargain. Comparing the "hotness" of different bargains could help identify which types of products or discounts are most appealing to consumers.

Percentage change in price: This can show how significant the discount is. Analysis could reveal if there's a correlation between the magnitude of the discount and consumer response (comments, hotness), helping businesses determine optimal discount levels.

Website of bargain: This could be useful for comparing the performance of bargains across different platforms or retailers. It may help identify which platforms are most effective for promoting bargains.

Price after discount: This can be compared with the original price to understand the depth of the discount. It could also be compared across products to identify which are providing more value for money after discounts.

Price before discount: This helps in understanding the original price point of the product. Comparing the before and after discount prices can give insights into the discount strategies of different sellers.

In conclusion, the collected data offers rich possibilities for further analysis. The variety of data points allows for multifaceted exploration of consumer behavior, market trends, and bargain effectiveness. Future analysis can leverage these data to drive business decisions and marketing strategies, thereby enhancing the impact of bargains and promotions.

Data analysis available also at:
https://github.com/AdamJJ00/web-scrapping-pepper/blob/main/pepper_data_analysis.ipynb

Description of participant's roles:

Adam Janczyszyn:
- Upgraded scrapers to use selenium and scrapy
- Created functionality of logging in using selenium
- Created notebook with data analysis

Jakub Wujec:
- Created second scraper that uses Scrapy Splash (logging in function); scraper used xpaths and selectors from previous projects

Jakub Żmujdzin:
- Recognized from which selector/xpaths to parse information from
- Created first version of scrapper that was based on selenium and BS (further upgraded by Adam Janczyszyn)
- Added value counts in each category for notebook

We have been cooperating thoroughly throughout the whole project creation and it was relatively hard to assign certain tasks to certain people, as we have been discussing the whole project creation almost all the time over discord.