Webscraping project: *eobuwie.com.pl*

1. **Names and IDs**

Author's name	**Student's book no.**
Magdalena Gruszecka	394258
Karolina Szczęsna	415989
Tomasz Starakiewicz	304905

2. **Topic and webpage description**

One of the very popular uses of Web scraping is e-commerce, including price research. The price of the products, as it is usually visible on the website, is easy to obtain. In our project we decided to scrap and analyze shoes prices.

For this purpose, we have chosen the eobuwie.com.pl website. Eobuwie.com.pl is a large online platform offering various footwear from different brands. We chose to scrape shoes under 'women shoes' as it limits the number of scraped data but it is still a large enough category to perform an interesting analysis. Category women shoes, at the time of our analysis, has around 470 pages and each page consists of about 100 pairs of shoes with product photo, name, number and price - regular and special if the product is discounted.

3. **Scraper mechanics**

   a. **Beautiful Soup**

1) Gets the number of pages (based on the bottom toolbar) or sets it to 100 if the `LIMIT_PAGES` parameter is set to True.

2) Iterates through every page and for each page scrapes data for all shoes about: f`irst_name`, `second_name`, `regular_price` and `special_price`.

3) Cleans the data for each product and appends it to the corresponding list.

4) Zips all lists and transform them into the data frame.

   b. **Selenium**

In order to initiate the selenium scraper, a certain condition needs to be fulfilled - the scraper is dedicated to Mozilla Firefox browser, as it is based on `geckodriver` and related packages. The scraping process starts from defining `gecko_path` and initiating `webdriver`. Other than

that, the limit of scraped subpages is set in a `range(0, 101)` to scroll through 100 subpages in total in a loop. To structurize the data, empty lists are created and used in the loop.

After initiating scraper in command line, the loop is activated:

1) The driver is installed and gets the predefined url for each subpage.

2) The scraper accepts cookies by clicking on a situated `By.XPATH` button "*Zgoda*" each time they appear while entering a webpage. In case the cookies do not occur, the scraper omits this step.

3) The loop is followed by extracting desired information about shoes for females, therefore first names informing about the category of a shoes and a brand, second names which are shoes' IDs as well as all prices (both regular prices if not on sale and special price combined with regular price in case of discount). Names and prices are scraped by class names, which are common for all products on the platform - `driver.find_elements(By.CLASS_NAME)`. Then, scrapped lists of classes are transformed to texts and strips and located in initially created empty lists. In addition, scraped prices are reformatted to ease further dataframe transformations and analysis.

The whole process is finished with closing the driver and transferring data about fist names, second names and prices from lists to a dataframe. Prices variable is split into two variables, therefore separate columns (`regular_price, special_price`). The final output is saved to `shoes.csv` file.

c. Scrapy

After launching the crawler, `ShoeSpider` class, inheriting from `Spider`, is initialized. Aside from standard args and keyword args, additional `pages` argument can be passed from the command line to set the number of pages crawled (500 by default), unless `LIMIT_PAGES` is hardcoded to `True`, which overrides the number to 100. List of categories to be scraped is loaded from category_links.csv (right now it's only one category - "damskie"). A list of `start_urls` is initialized as an instance variable, containing links to individual pages with shoes. Links to individual pages are generated based on categories loaded from the .csv file.

After initialization, `start_requests()` is called to generate `Request` objects from `start_urls` list. This method is not visible in the code, since the default implementation in the `Spider` class is sufficient - there is no need to override it.

Since the default `start_requests()` method is used, no callbacks are specified for the requests and the `parse` method is used by default for all urls. In this method, XPATH is used to first extract HTML with all bounding boxes with pairs of shoes. Then for each element a Selector is re-constructed from the HTML of a single box. From this selector, again using XPATH, `first_name`, `second_name`, `regular_price` and `special_price` are extracted.

The resulting variables are reformatted and cleaned of redundant whitespace. Then they are packaged into an instance of a `Shoe` class which was defined at the beginning of the code. The `Shoe` object is then yielded from the `parse` method into the .csv file, ready for data analysis.

4. Output description

The final output from all scraping methods (Beautiful Soup, Selenium and Scrapy) is a csv file with extracted four variables: `first_name`, `second_name`, `regular_price` and `special_price` of female shoes available on e-commerce platform eobuwie.com.pl. Depending on the `LIMIT_PAGES` condition as well as trends and ads of certain products on a webpage, the dataframe and csv file might be of different length and order, however for the purpose of further analysis 46 304 scraped records of shoes are used.

5. Data analysis

Notebook with the analysis code is available in the GitHub repo under [DSBA-webscraping/scrapy/analysis/analysis.ipynb]

Scraping results (as of 8.05) are presented in the table below - top 10 types in terms of shoe count (85% of all shoes) alongside the total for all types.

Type was created by extracting the first word in the first name of the items, which corresponds well to the actual list of shoe types on the website. Additional categorisation of top 10 types into winter/summer shoes was done manually after scraping. This immediately shows that summer shoes feature more prominently on the website (60% of all shoes).

Season	Category	Shoes count	as % of total	Shoes on promo	Avg. regular price	Avg. reg. price if discounted	Avg. promo price	Shoes discounted (%)	Avg. discount (%)
	total	46 304	100%	36 332	449	479	320	79%	-33%
summer	Sandały	8 561	19%	5 891	374	396	275	69%	-30%
summer	Sneakersy	7 681	17%	6 203	493	503	338	81%	-33%
winter	Botki	6 390	14%	5 863	597	621	384	92%	-38%
summer	Klapki	5 059	11%	3 494	345	379	260	69%	-31%
summer	Półbuty	2 998	7%	2 237	379	400	282	75%	-29%
winter	Buty	2 700	6%	2 260	484	483	351	84%	-27%
summer	Espadryle	2 323	5%	1 663	360	414	279	72%	-33%
summer	Japonki	1 295	3%	987	229	229	155	76%	-32%
winter	Trapery	1 210	3%	1 104	659	682	426	91%	-38%
winter	Kozaki	944	2%	864	790	821	513	92%	-38%
summer		27 917	60%	20 475	398	419	287	73%	-32%
winter		11 244	24%	10 091	595	614	392	90%	-36%

Almost 80% of all shoes are discounted, since the scraping took place during the “Mid Season Sale” communicated on the website. Percentage of shoes discounted varies across categories, though typical autumn/winter shoes are visibly more heavily promoted on average - with a higher percentage of the assortment promoted and at deeper discounts. Heavy discounting of winter shoes is intuitive, given the need to clear out stock, however it is surprising that most summer shoes are promoted as well.

Interestingly, regular prices of shoes that happen to be discounted are consistently higher than those that were not promoted at the time. This might be an indication of regular price inflation, though more data points would be needed to confirm this - e.g. by looking at the same shoes over time to analyze regular price fluctuations and % of time on promo.

6. **Performance comparison**

Scrapy is by far the fastest option, Selenium is noticeably the slowest, but at the same time it is not fully comparable since the timing depends on how long the `sleep()` calls are. Proper comparison would require reengineering the code to detect whether the full html has been loaded.

Scraping method	**Elapsed Time (seconds)**
Beautiful Soup	418.46
Selenium	1692.96
Scrapy	68.88

7. **Workload distribution**

    a. Magdalena Gruszecka - BeautifulSoup, Topic and Webpage description

    b. Karolina Szczęsna - Selenium, Output description

    c. Tomasz Starakiewicz - Scrapy, Data analysis