University of Warsaw
Faculty of Economic Sciences

Jakub Bandurski Student id: 417911
Maciej Lorens Student id: 419763
Piotr Radziszewski Student id: 420895

Otomoto.pl scrapers

Final project
Webscraping 2022L

Warsaw, June 2023

Topic description

The aim of this project was to develop a web scraping tool to extract data from otomoto.pl, a well-known Polish car-selling website. By collecting data such as price, mileage, power, production year, type of fuel, colour, and number of seats, we can gain valuable insights into the online car market. The retrieved information can be used by various stakeholders, including car buyers, dealerships, and automotive analysts, to make informed decisions, identify market trends, and assess pricing strategies. To show the potential of such analysis we present a linear regression model in a further part of this report.

The characteristics chosen to be scrapped are the most common distinguishing aspects of cars of the same model. We have selected Mercedes-Benz GL as the subject of our data scraping. It is worth mentioning that the structure of otomoto websites is similar among models, hence adjusting the scrapers to the user's desired models is very easy.

Website description

All three scrapers consist of two parts. The first part where the scrapers retrieve links to offers from a webpage with a listing of current offers and from subsequent pages with the continuation of this list (https://www.otomoto.pl/osobowe/mercedes-benz/gl-klasa/). The second part is iterated access to the gathered links and scraping target data.

When it comes to the structure of the websites the webpage with listing of current offers has said offers presented in *article* tags with attribute *data-variant* equal to *regular*. This attribute distinguishes between offers concerning chosen model from offers which are advertised and can contain other models of cars. The link to the offer is within *h2* tag's child – *a* tag. Next, the offer webpages have all necessary information listed in *span* tags with the same structure for all offers. Taking advantage of that the scrapers look for tags *span* which contain text which corresponds to the target data. For instance *span* containing text *Przebieg* has the value for mileage of the car in kilometres.

Scraper description

Beautiful Soup Scraper

The program utilizes the BeautifulSoup library to parse the HTML content of each page and extract relevant information.

To begin, the program defines a list of URLs representing different pages of Mercedes-Benz GL-Class car listings on the "https://www.otomoto.pl" website. These URLs will be scraped to gather the desired data.

Next, an empty list called links_list is created to store the links to individual car offers. The program then iterates over each URL in the sites list, sending an HTTP GET request to retrieve the content of each page.

In case a cookies pop-up appears on the page, the program automatically accepts the cookies by sending another GET request to the provided link. This is achieved using the requests.get() function.

There was an issue with collecting some offers using BeautifulSoup. Sometimes program received an url link but did not find any data to a car. The code checks 3 times if an offer has the price. If it does not it means that it is something wrong with an offer and it should be skipped.

Using BeautifulSoup, the program parses the HTML content of the page and locates the links to car offers by selecting the appropriate elements using CSS selectors. The href attribute of each link is extracted, which contains the URL of the respective car offer. These URLs are then appended to the links_list.

There is an option to limit the number of collected links to 100 by setting the limit_100 variable to True. If enabled, the program stops iterating and collecting links once the count reaches 100.

After collecting the desired links, a pandas DataFrame named df is created to store the extracted data. The DataFrame is initialized with columns such as price, mileage, power, production year, number of seats, fuel type, color, and the link to the car offer.

The program continues by iterating over each link in links_list. For each link, an HTTP GET request is sent to retrieve the content of the corresponding car offer page.

Once again, using BeautifulSoup, the program selects specific elements on the page that contain information about the car, such as price, mileage, power, production year, number of seats, fuel type, color, and the link. The program extracts the relevant data from these

elements and assigns them to the corresponding columns in the df DataFrame.

To prevent overwhelming the website with requests, the program introduces a delay of 5 seconds after each request.

Upon scraping all the car offers, the program saves the DataFrame as a CSV file named "offers.csv". The resulting CSV file contains the collected data in tabular form, with each row representing a car offer and each column containing the respective attribute information.

Additionally, the program records the running time of the scraper and saves it in a text file named "running_time.txt".

In terms of data analysis, the collected data can be utilized for further exploration. The output showcases a few rows of the extracted data, including details like price, mileage, power, production year, number of seats, fuel type, color, and the link to the car offer. With this data, various insights can be derived, such as calculating average prices, examining mileage distribution, analyzing power trends, or exploring other attributes. However, for more comprehensive and meaningful analysis, additional processing and statistical methods should be applied.

Scrapy Scraper

The code for the spider is included in the *otomoto_scrapy.py* file. First, a class *Car* is created, which inherits from the *scrapy.Item* class. Objects of *Car* include *scrapy.Fields* that will be filled with data scraped from particular sale offers. Each of the variables has their own field (price, mileage, power, production year, type of fuel, colour, and number of seats) and the last field will contain the link to the sale offer.

Next class - *OtomotoSpider*, inherits from the *scrapy.Spider* class and includes the code describing behavior of the spider. Starting urls are defined (in this case first five pages with sale offers of Mercedes-Benz GL) as well as a counter and boolean parameter *limit_100*, which are used when limiting the number of scraped pages to a 100.

Method *parse* contains instructions for scraping links to offers present at each of the 5 pages. The xpath to every offer excludes any ads that may be present by specifically choosing *data-variant=regular*. Before starting to loop through the links to the offers, it is checked if the user wants to limit the number of scraped pages to 100. If yes, the counter increments by 1 at each iteration. Method *parse* yields a *scrapy.Request* to an url of an offer at current iteration and a callback to the *parse_offer* method that handles scraping the data from the

sale offer.

Method *parse_offer* starts by creating a *Car* object that will store scraped data in *scrapy.Fields*. For columns price, mileage and power, only digits are extracted (currency and units are discarded). The data in the remaining columns (except for links to the offer) is stripped (with method *strip*) of any unnecessary whitespace characters surrounding it. *parse_offer* yields the *Car* object and appends it to the output.

The remaining method *Close* describes the instructions to execute when the spider is finished scraping. *crawler.stats* attribute provides the start and finish time of the spider, which enable measuring the running time.

Some of the implemented functionalities of Scrapy exist outside of the main spider file *otomoto_scrapy.py*. In settings.py, *DOWNLOAD_DELAY* (minimum seconds to wait between 2 consecutive requests to the same domain) was set to 5 seconds in order not to overload the servers used by *otomoto.pl*. *FEED_EXPORT_FIELDS* was modified as well, for the purpose of defining the order of columns in the exported csv file.

Finally, running the spider and saving the output into a csv file is achieved in Scrapy by running *scrapy crawl otomoto -O offers.csv* in the command line.

Selenium Scraper

The whole Selenium scraper code is present in *selenium/scraper.py* file on our Github repository. The mechanics are as follows. Firstly, the timer is started to measure running time with use of *time* package. Then the parameter limiting to 100 links *limit_100* is set to True as default. Further the Selenium is set up which means pointing to the *geckodriver.exe* and setting options such as *headless=True* which disables the firefox windows from opening while the scraper is running.

A list of URLs is created with URLs representing different pages of Mercedes-Benz GL-Class car listings on the www.otomoto.pl website. These links will be scraped to gather the desired data. An empty list links_list is created to store the actual URLs to offers. Next, a for loop is created which goes through chosen pages and collects links to current offers of selected car and saves them to links_list. A button-click functionality is added to deal with cookies agreement. If the limiting parameter is set to True then the scraper breaks out of the for loop with exactly 100 links to offers.

Finally, the proper scraping process begins with the creation of empty pandas data frame

which is storing the results. Next, a for loop is run to iterate over gathered offer links and get the data. The data for each offer is appended to a list called *row* which is then appended to the output data frame. Each access to a website by URL is done by calling *driver.get(URL)* method. Then the *find_element()* and *find_elements()* methods are called to find specific tags in the HTML. The tags are search for with use of defined xpaths. The target data is stored in *span* tags with constant layout for all offers. The scraper looks for *text* inside *spans* which corresponds to the desired data. Whenever no such data is present the try-except clause returns None in place of the data point.

At the end, the data scraped from the links is saved to a CSV file *offers.csv* and the running time is saved to *running_time.txt* file. The scraper can be run from the command line with python3 interpreter.

Output description

The output file is a CSV file that contains scraped data. The columns are price, mileage, power, prod_year, seats_num, fuel, colour, and link where all of the variables are self-explanatory and the link is the link to the offer webpage. Columns have the following types: int, int, int, int, int, character, character, character. Example output can be found as *offer.csv* in our Github repository here.

Elementary Data Analysis

To demonstrate how the scraped data could be used for statistical analysis, a simple linear regression with Ordinary Least Squares was performed. The results are included in Table 1. Price was selected as the response variable and mileage, power, year of production, number of seats, fuel and colour as explanatory variables. By leaving out insignificant variables (colour) and transforming fuel into a dummy variable indicating if the car runs on diesel, R^2 of 0.685 was achieved. This result proves that it is possible to successfully use the scraped data for identifying main determinants of used car prices. It is encouraged to extend the analysis by including data from offers relating to different car models and using them as dummy variables.

Table 1: OLS results

	Dependent variable:
	price
mileage	-0.297^{***} (0.086)
power	0.368^{***} (0.091)
prod_year	0.573^{***} (0.081)
diesel	0.732^{***} (0.192)
Constant	-0.373^{***} (0.113)
Observations	100
R^2	0.685
Adjusted R^2	0.671
Residual Std. Error	0.573 (df = 95)
F Statistic	51.552^{***} (df = 4; 95)
Note:	*p<0.1; **p<0.05; ***p<0.01

Scrapers Performance Comparison

The approach to the comparison of performance was to measure the time necessary for all scrapers to complete scraping the same list of 100 links. The reader should bear in mind that due to the nature of this cooperative project, the measurements have been taken on separate computers with different internet access. The running times are listed below:

Beautiful Soup Scraper – 563.05 seconds

Scrapy Spider – 659.44 seconds

Selenium Scraper – 941.83 seconds

The conclusion overlaps with what we expected after using all three approaches during classes. Beautiful soup scraper is the fastest one and the Selenium scraper is the slowest one. This result is affected by the mechanism of Selenium, in particular, the fact that the websites have to load as if inside a browser in order for Selenium to be able to capture the data.

Work split

Beautiful Soup Scraper – Piotr Radziszewski

Scrapy Scraper – Maciej Lorens

Selenium Scraper – Jakub Bandurski