Why Is More Efficient to Combine BeautifulSoup and Selenium in Scraping For Data Under Energy Crisis

Simona Vasilica Oprea
Adela Bâra
The Bucharest University of Economic Studies, Romania
simona.oprea@csie.ase.ro
bara.adela@ie.ase.ro

Abstract

The electricity prices are often sensitive data that pose challenges in terms of collection. The volatility and soaring prices make the day-ahead electricity markets appealing for scientists. Several events took place from 2020, such as COVID-19 and military conflict in Ukraine leading to higher inflation, interest rates and energy crisis. Since 2020, the electricity prices on day-ahead market (DAM) increased even up to ten times. To perform the electricity price prediction, extensive feature engineering and historical data are required. Data sources from Romania are assessed to analyze the opportunity to extract relevant data for the electricity price and traded quantities. Thus, in this, paper, we investigate opportunity to extract data from web pages. We will suggest solutions to extract historical data from web sites that do not provide APIs or csv files. Several Python libraries such as BeautifulSoup and Selenium will be showcased, and approaches will be compared.

Key words: electricity price, data scrapin, BeautifulSoup, Selenium, prediction
J.E.L. classification: Q47, C82, Q41

1. Introduction

Web "scraping" (also called "web harvesting," "web data extraction," or even "web data mining"), can be defined as "the construction of an agent to download, parse, and organize data from the web in an automated manner" (Tenorio de Farias *et al.*, 2021). Many websites nowadays offer such an API that provides means for the outside world to access their data repository in a structured way. APIs are great means to access data sources, if API exposes the required functionality. The general rule of thumb is to look for an API first and use it, before setting off to build a web scraper to gather the data. There are various reasons why web scraping might be preferable over the use of an API: the website somebody wants to extract data from does not provide an API; the API provided is not free (whereas the website is); the API provided is rate limited: meaning that the access is limited to a number of certain times per second, per day, etc.; the API does not expose all the data we wish to obtain (whereas the website does).

In this paper, we will show a use case aiming to extract data from a website with Python libraries, namely BeautifulSoup and Selenium. The webpage contains hourly electricity prices and traded quantities on DAM per day. The table is displayed in the middle of the page and above it, a selector for day, month and year is placed. Therefore, if we intend to extract data for a longer interval let's say from January 2019 to August 2022, the selector should be manually changed, and the Refresh button pressed more than 1300 times that is tedious and should be automatized.

2. Literature review

Usually, websites offer APIs or csv files with historical data, but when this is not possible, the only way is to scrape the website using dedicated tools such as BeautifulSoup and Selenium from Python (Thomas and Mathur, 2019) or curl from PHP or rvest from R. Scraping is a method to obtain data from web (vanden Broucke and Baesens, 2018; Dogucu and Çetinkaya-Rundel, 2020; Liu and Chen, 2021). Interesting use cases exist, such as: the national institute of statistics scraping the web to obtain goods (food and non-food) and services prices to calculate the consumer price indices, banks and their competitors, election, projects data to start with, price comparison, public sentiment (Subramaniyaswamy *et al.*, 2017) regarding Bitcon, psychology (Landers *et al.*, 2016), patterns of depression and suicidal thoughts (https://www.sas.com/en_ca/insights/articles/analytics/using-big-data-to-predictsuicide-risk-canada.html) and others (Han and Anderson, 2021). The purpose of extraction the data is to understand phenomena (Kusumasari and Prabowo, 2020), obtain valuable insights and improve the quality of life.

Our purpose is to extract data from OPCOM – the Romanian Market Operator website and to perform electricity price forecast that is useful for building strategies of the market participants, such as: electricity producers and suppliers. Numerous alternatives exist to scrape data, such as PHP, R and Python libraries (Wang *et al.*, 2021). Apart from BeautifulSoup and Selenium, Scrapy is another Python library. A notable drawback of Scrapy is that it is does not emulate a full browser stack. Dealing with JavaScript will hence be troublesome using this library. CatchControl is to avoid continuously hammering web servers with requests over and over again. Moreover, graphical scrapping tools are available such as Portia, Parsehub, Kapow, Fminer, Dexi. However, they showed issues with heavy loaded JavaScript. Oftentimes, these tools will fail to work once the data contained in a page is structured in a less-straightforward way.

3. Research methodology

In getting data from OPCOM website, we will use Python, namely BeautifulSoup and Selenium libraries. Also, we need to create a request to HTTP. There are several Python libraries that interacts with HTTP:

- "httplib2" - a small, fast HTTP client library. Originally developed by Googler Joe Gregorio, and now community supported;
- "urllib3" - a powerful HTTP client for Python;
- "requests" – preferred in this exemplification;
- "grequests" which extends requests to deal with asynchronous;
- "aiohttp" another library focusing on asynchronous HTTP.

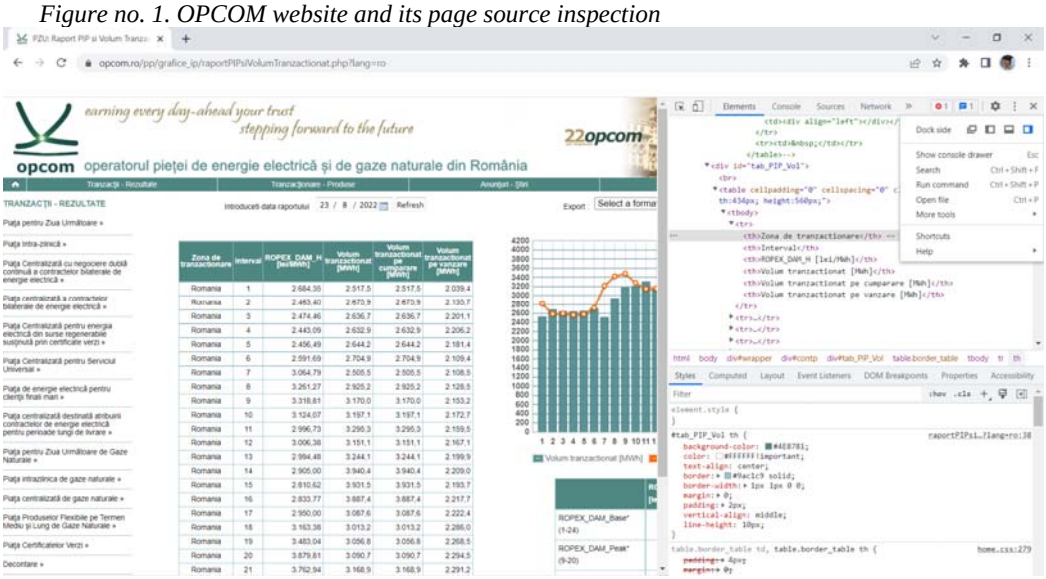
The GET method requests a representation of the specified URL. In comparison with POST, requests using GET should only retrieve data and should have no other effect, such as saving or changing user information or perform other actions. Thus, it should be safe to execute the same request multiple times. An example of http request is provided in Table 1.

Table no. 1 Example of a HTTP request

```
#pip install requests
#Import libraries
import requests
#Create an URL object
url = 'https://www.opcom.ro/pp/grafice_ip/raportPIPsiVolumTranzactionat.php?lang=ro'
#Create object page
page = requests.get(url, verify = False)
```

First, we will investigate the webpage to understand the DOM (Document Object Model) as in Figure 1.

Figure no. 1. OPCOM website and its page source inspection



Source: https://www.opcom.ro/grafice-ip-raportPIP-si-volumTranzactionat/ro

For the static components of the table (headers, rows and data from cells), BeautifulSoup (BS) is recommended. In Table 2, we will create an object page and obtain page information by creating a BS object called soup.

Table no. 2 BS library – creating a BD object

```
import requests
from bs4 import BeautifulSoup
#Create an URL object
url = 'https://www.opcom.ro/pp/grafice_ip/raportPIPsiVolumTranzactionat.php?lang=ro'
#Create object page
page = requests.get(url, verify = False)
#parser-lxml = Change html to Python friendly format
# Obtain page's information by creating a BS object (called soup)
soup = BeautifulSoup(page.text, 'lxml')
#soup = BeautifulSoup(page.content, "html.parser")
```

Beautiful Soup's main task is to take HTML content and transform it into a tree-based representation. Once a BeautifulSoup object is created, there are two methods necessary to fetch data from the page: find(name, attrs, recursive, string, **keywords); and find_all(name, attrs, recursive, string, limit, **keywords) or findAll. In Table 3, an exemplification of find_all method is provided. We will create a list with table headers and these headers will become the columns of a dataframe where the data will be temporarily stored. Moreover, find method helped us to identify the table from where we scrape the data by its class.

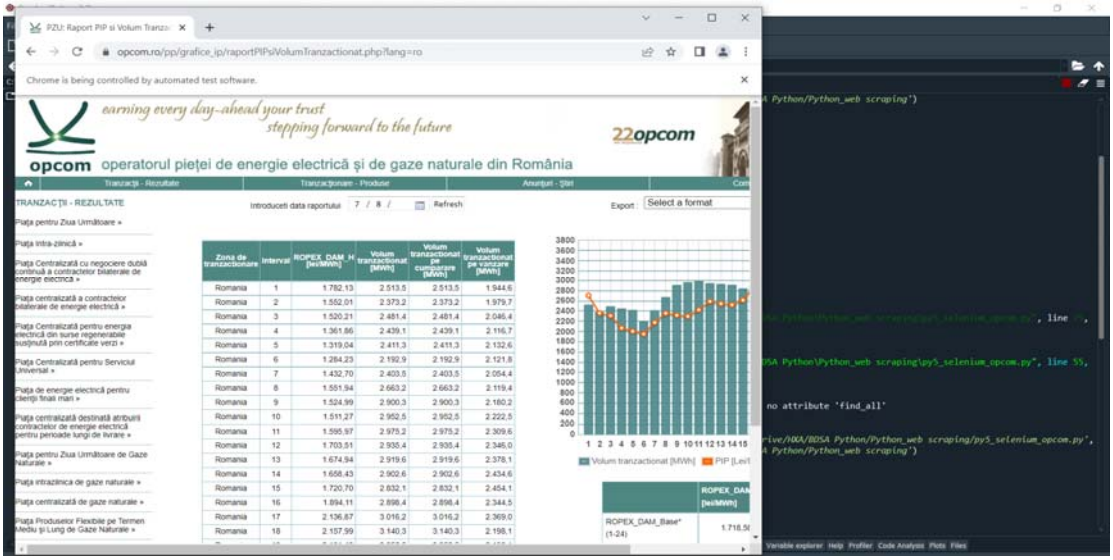
Table no. 3 BS library – find and find_all exemplifications

```
import pandas as pd
…
# Obtain information from tag <table>
table1 = soup.find('table', {"class":"border_table"})
# Obtain every title of columns with tag <th>
headers = []
for i in table1.find_all('th'):
  title = i.text
  headers.append(title)
mydata = pd.DataFrame(columns = headers)
```

However, requests and BS libraries are not enough to deal with script tags. Apart from data from table that can be extracted with BS, we have to manipulate the selector of date (day, month, year) and also to click the Refresh button every time the date is changed. Selenium is a powerful

web scraping tool that was originally developed for the purpose of automated website testing. Selenium works by automating browsers to load a website, retrieve its contents, and perform actions like a user would when using the browser (as in Figure 2). There are some options that Chrome has that can stop displaying of the webpage (headless mode). As such, it's also a powerful tool for web scraping. Selenium can be controlled from various programming languages, such as Java, C#, PHP, and of course, Python.

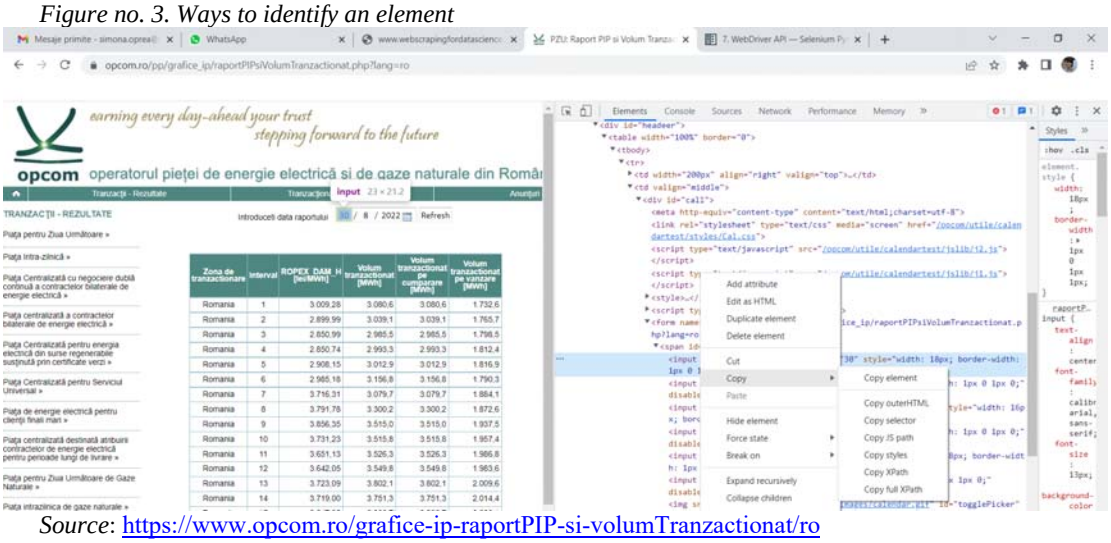
Figure no. 2. Selenium showing up the webpage it investigates



Source: https://www.opcom.ro/grafice-ip-raportPIP-si-volumTranzactionat/ro

It's important to note that Selenium itself does not come with its own web browser. Instead, it requires a piece of integration software to interact with a third party, called a WebDriver. WebDrivers exist for most modern browsers, including Chrome, Firefox, Safari, and Internet Explorer. When using these, a browser window will open up on your screen and it will perform the actions the specified in the code. chromedriver.exe has to be downloaded run and its path inserted into Advanced System Setting Environment variables PATH. The easiest way to make sure Selenium can see this executable is simply by making sure it is located in the same directory as your Python scripts. Finding elements with Selenium (Table 4) is almost similar with BeautifulSoup. An element can be identified by other characteristics if the name and ID are not available, such as CSS selector or XPath as in Figure 3.

Figure no. 3. Ways to identify an element



Source: https://www.opcom.ro/grafice-ip-raportPIP-si-volumTranzactionat/ro

Table no. 4 Methods to find elements in Selenium

Selenium version check: find_element_by_* is deprecated in the latest versions
Instead, find_element(By.*) sau find_elements(By.*)
find_element(By.ID, "id")
find_element(By.NAME, "name")
find_element(By.XPATH, "xpath")
find_element(By.LINK_TEXT, "link text")
find_element(By.PARTIAL_LINK_TEXT, "partial link text")
find_element(By.TAG_NAME, "tag name")
find_element(By.CLASS_NAME, "class name")
find_element(By.CSS_SELECTOR, "css selector")
Methods above will raise a NoSuchElementException exception in case an element could not be found. Beautiful Soup simply returns None in this case.

Source: Authors' contribution

4. Findings

In this section, we will compare two approaches in scraping data from a webpage. Selenium can handle both static and dynamic parts of the page. In other words, Selenium can extract the data from table, change the date and click automatically on Refresh button. However, the best approach is to combine Selenium and BeautifulSoup libraries. Selenium is the best to dynamically send data to input fields and press buttons, whereas BS is the best to grab data from HTML tags. Why to use this combination and bother with both libraries? Tasks can be done only with Selenium? The answer is yes, but the process is much slower! When both Selenium and BS are used the execution time to extract data for one month is 20 seconds as in Figures 4 and 5.

Figure no. 4. Using BS and Selenium to scrape data

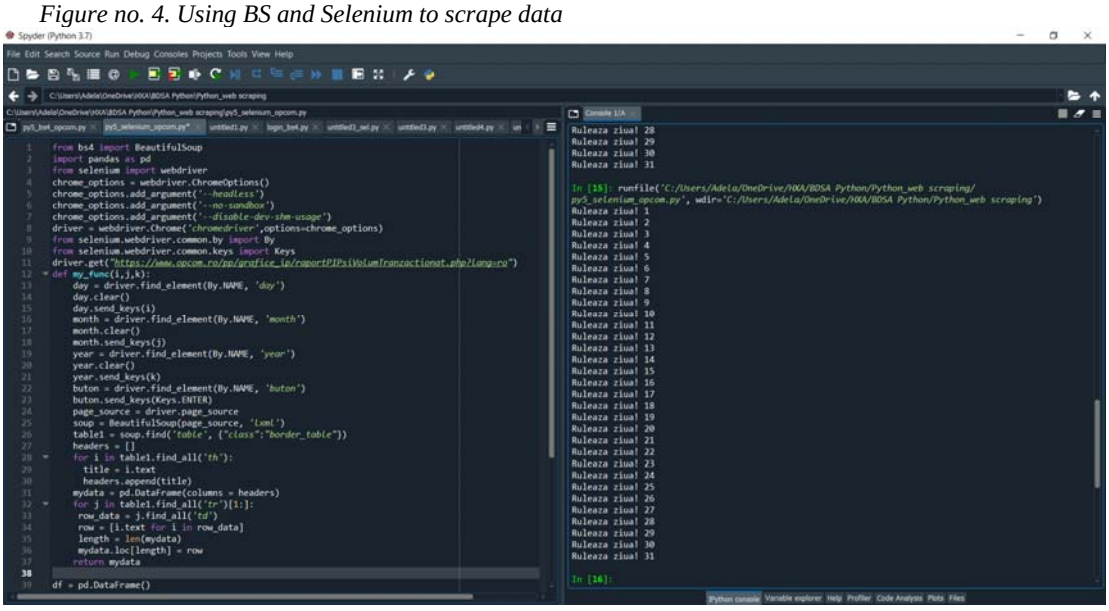
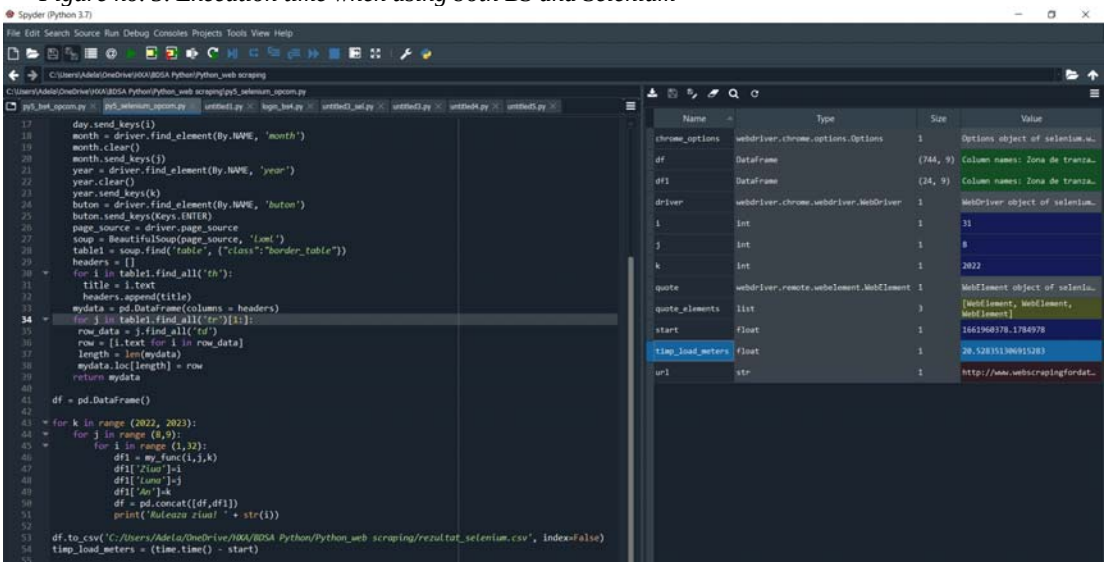
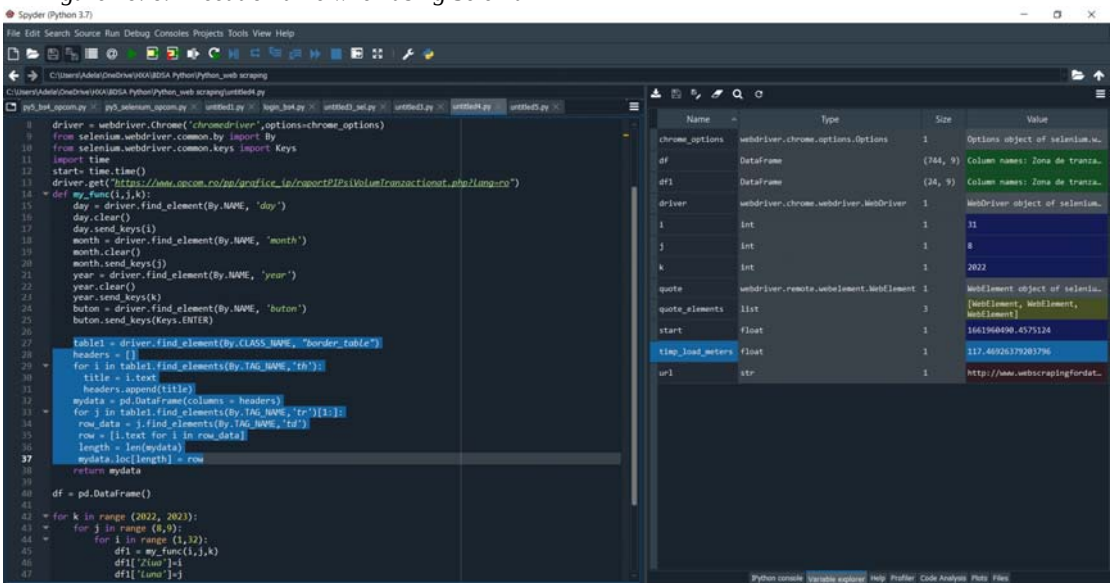


Figure no. 5. Execution time when using both BS and Selenium



In comparison, only with Selenium, the execution time is heading to 117 seconds, that is almost 6 times slower as in Figure 6.

Figure no. 6. Execution time when using Selenium



The entire code to extract data from OPCOM website is provided in GitHub.

5. Conclusions

In this paper, we propose a method to extract data from websites that do not provide APIs or csv files. We also discussed other alternatives to extract data from web and other goals for such endeavour. The website contains electricity prices and traded quantities that were extracted using both BeautifulSoup and Selenium libraries. The historical data is essential when forecast is required. However, we simulate a case where only Selenium was used and showed that the execution time increase from 20 to 117 seconds, almost six times. Therefore, we suggest to use both Selenium and BS to accomplish the task.

6. Acknowledgement

This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS- UEFISCDI, project number PN-III-P4-PCE-2021-0334, within PNCDI III.

7. References

- vanden Broucke, S. and Baesens, B., 2018. *Practical Web Scraping for Data Science*, *Practical Web Scraping for Data Science*. https://doi.org/10.1007/978-1-4842-3582-9.
- Dogucu, M. and Çetinkaya-Rundel, M., 2020. Web Scraping in the Statistics and Data Science Curriculum: Challenges and Opportunities. *Journal of Statistics Education.* https://doi.org/10.1080/10691898.2020.1787116.
- Han, S. and Anderson, C.K., 2021. Web Scraping for Hospitality Research: Overview, Opportunities, and Implications, *Cornell Hospitality Quarterly.* https://doi.org/10.1177/1938965520973587.
- Kusumasari, B. and Prabowo, N.P.A., 2020. Scraping social media data for disaster communication: how the pattern of Twitter users affects disasters in Asia and the Pacific. *Natural Hazards.* https://doi.org/10.1007/s11069-020-04136-z.
- Landers, R.N., Brusso, R.C., Cavanaugh, K.J. and Collmus, A.B., 2016. A primer on theory-driven web scraping: Automatic extraction of big data from the internet for use in psychological research. *Psychological Methods*. https://doi.org/10.1037/met0000081.
- Liu, W. and Chen, P., 2021. Justification of the behavior regulatory pattern on data scraping. *Computer Law and Security Review*. https://doi.org/10.1016/j.clsr.2021.105578.
- Subramaniyaswamy, V., Logesh, R., Abejith, M., Umasankar, S. and Umamakeswari, A., 2017. Sentiment analysis of tweets for estimating criticality and security of events. *Journal of Organizational and End User Computing.* https://doi.org/10.4018/JOEUC.2017100103.
- Tenorio de Farias, M., Angeluci, A.C.B. and Passarelli, B., 2021. WEB SCRAPING AND DATA SCIENCE IN APPLIED RESEARCH IN COMMUNICATION: a study on online reviews. *Revista Observatório*. https://doi.org/10.20873/uft.2447-4266.2021v7n3a1en.
- Thomas, D.M. and Mathur, S., 2019. Data Analysis by Web Scraping using Python. *Proceedings of the 3rd International Conference on Electronics and Communication and Aerospace Technology, ICECA 2019*. https://doi.org/10.1109/ICECA.2019.8822022.
- Wang, Y., Zhu, D., Zhang, B., Guo, Q., Wan, F. and Ma, N., 2021. Review of data scraping and data mining research. *Journal of Physics: Conference Series*. https://doi.org/10.1088/1742-6596/1982/1/012161.