ISBN: 978-628-96613-0-9. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LEIRD2024.1.1.721

Let’s re-search. APIs and web-scraping. ResearchGate case

Mathias Haas-Mendoza, M.Sc.[1], Fabricio Zanzzi, Ph.D.[2]
[1]ESPOL Polytechnic University, Ecuador, amhaas@espol.edu.ec
[2] ESPOL Polytechnic University, Ecuador, pzanzzi@espol.edu.ec

Abstract*– This paper explores the growing need for more efficient and streamlined methods of retrieving academic data, especially when working with extensive datasets. It provides a detailed comparison of three retrieval techniques: manual searches, web scraping, and Application Programming Interfaces (APIs). While manual methods remain effective for small-scale searches, they quickly become impractical for large data volumes. In contrast, web scraping and APIs offer automation that significantly accelerates data collection. However, platforms like ResearchGate currently limit the use of these automated methods, forcing researchers to rely on manual processes. This paper advocates for ResearchGate to implement APIs, akin to those of Scopus and Web of Science, to provide controlled, secure, and efficient access to academic literature. Such advancements would be particularly beneficial to researchers from developing nations and institutions without access to subscription-based services. Furthermore, enabling automation would democratize access to scientific resources and foster a more inclusive global academic environment.

***Keywords*— web-scraping, algorithms, python, open resources**

I. INTRODUCTION

Modern research studies have found their base in a large and diversified range of data sources [1]. Consequently, traditional methods like manual iterative searches and download are not sufficient and current research methods require access to large and complex volume of information [2], specifically bibliographic data.

Academic data retrieval can be made through three different techniques: (i) manually, (ii) web scraping and (iii) APIs [3]. The first one is the traditional way, using an initial point, and iteratively search and download references according to the nature of the current research. For high volumes of data, complex networks of literature or long timespans of analysis this method can be tedious and a non-efficient process [2]. The second one is an automated program that extracts patterned data and speeds everything a normal user can do in website: do queries, search for specific keywords or data, request it, parse it and save it [4], [5] but without the time consuming manual iteration. The third one is Application Programming Interfaces (APIs, from now on). An API, defined by Mitchell [4] is a piece of software that allows the communications with other programs, even if they are written in different computer languages and structured differently. This last tool can be used to search and retrieve data [6] or to access large volumes of scientific literature in order to upgrade the analyses, find new patterns or behaviors in the scientific literature [2].

If the web-scraping or the API usage is not allowed by the repositories of scientific journals, research can take several weeks to complete. This may aggravate when articles or repositories have high costs. Subsequently, this means a disadvantage to researchers from developing countries. This disadvantage can be suppressed by the permission of APIs or web-scraping, or at least, the permission with respect to open access databases.

II. THE (NO) ACCESS TO INFORMATION

On one hand web scraping is mostly used (or preferably used) when the data that researchers are looking for is relatively small, uncommon or the source does not have the infrastructure of technical ability or the data is protected or not intended to be spread [4]. Nonetheless, any tool can be misused, and web scraping is no exception. The exploitation of data through web scraping can be made to gather data that are not meant to be gathered, for example de Cambridge Analytica scandal in 2018 and websites may have restricted access to scraping techniques [3]. Moreover, some websites could misread an automated process of scrapping in a short period of time and identify it as an informatic attack, like for example Instagram or, in this case, ResearchGate.

On the other hand, according to Harder [6], there are two kind of APIs in the academic context, the ones that require affiliation with a subscribing institution (Scopus or Web of Science) and those who don’t (Crossref, OpenAlex). The first ones mentioned have Developer Portals with user guides for researchers so they can use with programming languages to automatize the accessibility of information in these databases and get academic literature almost immediately. The subscription method of these APIs allows the platforms to delimitate under what criteria the requested data is given (for example, only giving the data for users inside the academic community, or only for research purposes, etc.). Scopus helps. ResearchGate does not yet.

III. WHY RESEARCHGATE?

Given the advantages provided by automated tools on platforms like Scopus or Wiley, one might ask: What are the benefits of using ResearchGate?

In the broader context of academic work, access to scientific information is of critical importance for authors.

Frequently, universities in developing countries lack access to or subscriptions to high-impact platforms like Scopus and Wiley. This situation not only complicates access to scientific literature for researchers of all ages and levels of expertise in these countries, but also limits the use of automation practices, which are crucial for conducting research in alignment with the objectives of the present study.

ResearchGate provides access to scientific literature irrespective of the credentials of researchers, universities, or other institutions involved in research activities [7]. From our perspective, this represents a significant added value.

Based on our experience, we have been able to find literature on ResearchGate that was unavailable on other platforms, even when using the credentials of our affiliated university.

Furthermore, another relevant aspect of ResearchGate is the proximity it fosters through its platform structure. Researchers can engage directly with those interested in their work, which creates a crucial turning point. For example, early-career researchers who are just beginning to explore the research process can establish direct contact with experts in their field of interest and even exchange messages [8]. Our experience in this regard has been overwhelmingly positive. The authors we reached out to responded kindly, and most of them were willing to share their work. For those who were unable to send their publications (due to restrictions such as copyright), they provided references to where the literature could be accessed. This interaction emphasizes the academic community's willingness to share their work, and, more importantly, it facilitates the reduction of barriers to scientific collaboration between individual researchers, institutions, and even countries. Conventional platforms, while they may offer contact information, do not support such direct communication.

For us, one of the more appealing features of ResearchGate is the simplicity of its structure, which enables the easy development of automated algorithms, particularly for searching scientific literature. In addition, the platform offers other elements that represent added value for authors, such as certain features resembling those of social media or forums but oriented towards science and the academic community. These features encourage the exchange of ideas, the suggestion of new developments in the users' fields of research, and the openness of researchers to ask and answer questions in discussion threads. ResearchGate also includes a small job board where platform users can apply for academic and research positions.

In general, for the authors of this paper, ResearchGate is not just a platform for searching academic literature or a social network. It is also a space that, in the first instance, reshapes the image of academic research (which many people might perceive as somewhat rigid and monotonous) and, in the second instance, enhances the research experience, opening the door to new ideas and facilitating networking with scientific experts from around the world across a wide array of disciplines.

IV. An alternative solution

Even with great advances in this specific area of the scientific matter, like mentioned before, some academic platforms like ResearchGate don't have any of the tools mentioned above. According to their website, ResearchGate is a free-to-join professional network for scientists and researchers intended to connect and facilitate the interaction of outputs, knowledge and expertise. In a few words, it is a social network for science people based on the academic outputs, where you can interact with the authors of a specific article, share ideas and most importantly: download/request articles directly from the authors. The nature of articles varies, there might be drafts, beta versions or open access articles.

Nonetheless, in this great platform, where you can interact directly with the authors, share some ideas with them. The only way to search and download/request for a specific article is to do it manually. The process is very simple but time consuming in cases where large volumes of articles are on the table, so we developed a python code to web-scrap the site based in a list of DOI codes (See Appendix 1). The first stage was successful but after some periods of testing, the code failed because the site asks you to demonstrate you are human through a captcha code (and a python-based code is obviously not a human) so the platform blocks any kind of "unusual" activity. Our hypothesis is that the platform might have registered large queries in an atypical timespan and blocked it thinking it was some kind of data theft.

In the ResearchGate case the blockage of the python code might be understandable in the context mentioned above; however, we also believe that the new modern methods of research demand computerized methods and ResearchGate should not fall behind. So even if ResearchGate does not allow web-scraping methods, the APIs are a good option and should be applied in the same way that Scopus, Springer or Web of Science do, they use a subscription affiliation method, so it's more secure and has more control in the usage so the data theft is out of the table.

In the case that ResearchGate decides to build an API or allow web-scraping under some kind of restricted nature, it would help thousands of registered researchers in the progress of their respective field of science, especially in two aspects: those who come from developing countries and those, in general who do research, but their affiliation institutions are not subscribed to platforms like Scopus or others.

V. Implementation: How does our code work?

The final code, written in Python, is designed to automate the search and retrieval of scientific articles from ResearchGate. The algorithm initiates by opening a browser session and navigating to the ResearchGate website, where it logs in using the credentials provided by the researcher. From there, the process involves taking DOI (Digital Object Identifier) codes stored in an .xlsx file and searching for them iteratively. Once located, the code downloads the articles or requests access from the authors, depending on the available option, and proceeds to the next DOI until the entire list is processed.

A. Preparation

The development of this script follows a structured, step-by-step approach. First, it is essential to establish an Integrated Development Environment (IDE) and code editor. For this project, Visual Studio Code version 1.94, developed by Microsoft [9], was used alongside the Python extension (version v2024.16.0) and Python language version 3.12. The next step involves installing a web driver to control web browsers programmatically. In this case, the Google Chrome WebDriver, known as chromedriver, version 127.0.6533.72, was used.

The third step is to define the packages and libraries required for the execution of the code:

OS: A module that provides operating system functionality, such as accessing directories [10].
Subprocess: Allows the execution of system commands from Python, which is particularly useful for automating the installation of necessary packages [10].
Sys: Used in conjunction with subprocess to access the Python interpreter and execute the package installation commands [10].
Openpyxl: A library that facilitates the loading and manipulation of Excel files [10].
Selenium: A powerful tool for automating web browsers [10].

Initially, the code required manual installation of each library. However, with replication and reusability in mind, we incorporated additional lines of code to automatically install the required packages. This enhancement reduces the need for manual intervention, making the code more user-friendly. The code also automates the creation of a virtual environment for code development [11], further simplifying the setup process for future users.

B. Web Browsers

Earlier versions of the algorithm allowed the user to choose which web browser to use from Microsoft Edge, Google Chrome, and Mozilla Firefox. Depending on the user's selection, the corresponding web driver was activated. However, due to the familiarity of Chrome and to streamline the testing process, we ultimately decided to only support Google Chrome in the final version. This decision was made to simplify the workflow and ensure compatibility, minimizing the potential issues associated with browser-specific behaviors.

C. The .xlsx File

Parallel to the automation of the search process, it was necessary to have a list of DOI codes in an Excel file so that the code could, via openpyxl, take each DOI and search for it on the platform. The structure of the .xlsx file is simple: the first column contains the DOIs for each article. This list serves as the input for the automated searches, and each DOI is processed in sequence to ensure comprehensive coverage of the desired papers.

D. Accessing ResearchGate

Once the virtual environment is set up, the packages installed, the web driver chosen, and the .xlsx file prepared, the next step involves using Selenium to access ResearchGate, specifically the login page. To facilitate the login process, the code launches the browser via Selenium and locates the input fields for entering the credentials. Two options are available for logging in: (i) the user can manually input their credentials, or (ii) the user can pre-enter their credentials in the code, allowing the script to automatically populate the login fields.

Once logged in, Selenium identifies [10] the search bar on the ResearchGate website and retrieves the DOI codes from the .xlsx file. For each DOI, it enters the code into the search bar and navigates directly to the article's page. If an article is not found, the code moves on to the next DOI in the list [12]. When a match is found, the code either clicks the download button or submits a request to the authors, depending on the available option, before proceeding to the next DOI. This process is entirely automated, but the code includes timed pauses to ensure that each action is completed before moving on to the next.

F. Error Handling and Continuity

As previously mentioned, the code is designed to handle errors and exceptions gracefully. If an issue arises, such as an article not being available or an error occurring during the search process, the script does not terminate [13]. Instead, it captures the error, logs it, and continues processing the remaining items in the list. This error tolerance ensures the code's robustness and reliability, even in the presence of minor issues. By allowing the script to continue despite encountering errors, the code achieves a higher level of resilience, making it suitable for large-scale operations where occasional failures are inevitable.

G. End of Process

After processing all the DOIs in the list—whether downloaded or requested—the browser session is closed automatically. In a previous version of the script, lines of code were included to store and print errors at the end of the process. However, for simplicity's sake, this feature was removed in the final version. The goal was to prioritize ease of use and avoid introducing additional complexity that might detract from the core functionality of the script.

VI. Comparative methodology:

A. *How does an API work*

Taking the Scopus API as an example, according to the Elsevier Developer Portal, the APIs provide access to a variety of valuable resources [14], including citation data, metadata, and abstracts from scholarly journals indexed by Scopus, Elsevier's citation database. Users can also access full-text journals and books published by Elsevier on the ScienceDirect platform. Additionally, research metrics are available through SciVal, Elsevier's benchmarking platform for research performance. Engineering resources can be found on Engineering Village, while curated abstracts, indices, and other metadata are indexed by Embase, Elsevier's biomedical abstract and indexing database. Furthermore, the APIs offer access to reactions, chemical structures, and chemistry information from Reaxys, Elsevier's expertly curated chemistry database. Lastly, users can retrieve safety data, efficacy data, pharmacokinetic information, and details on metabolizing enzymes and transporters from PharmaPendium, a fully searchable database containing data extracted from FDA and EMA drug approval documents, as well as FAERs, to aid in informed drug development decisions [15].

B. *APIs: A key*

To utilize the API effectively, an API Key is required. The API Key is a string of text that serves as an identifier and access key, under which platforms permit access to automation activities. Requesting the API is relatively straightforward; however, for research purposes, there are certain restrictions, such as a maximum request quota, for which an extension can be requested. Taking the Python way, packages such as Pyscopus or Pybliometrics facilitate access to Scopus through Python. Users can write a few lines of code to obtain bibliographic information easily. Nonetheless, as previously mentioned, certain restrictions apply.

C. *Our method*

In contrast, our method, as previously outlined, does not rely on APIs. Instead, it simulates the manual search for papers, mirroring the step-by-step process a researcher would follow. The added value of this approach is that researchers can run the code to automate the search process, saving them from spending time on a task that is often repetitive, tedious, and prone to error.

VII. The relevance of our method

As previously discussed in earlier sections, it is of significant importance to the authors to enhance access to scientific literature, particularly in developing countries or at universities that, for various reasons, lack access to necessary subscriptions or affiliations. Access to quality academic information is fundamental to the advancement of knowledge across various disciplines, especially in critical areas such as climate vulnerability, where informed decisions can have a significant impact on affected communities. Our proposed method utilizes a readily accessible tool that does not require researchers to possess specific credentials or affiliations, which are often costly and not universally available. Moreover, our approach allows researchers to avoid engaging in unethical practices when seeking academic literature.

In parallel, there is a growing trend in research related to bibliometrics. We confirmed this trend by executing the code provided by Volker Strobel [16] which analyzes results from Google Scholar using the keyword "Bibliometric." Figure 1 illustrates these results. A search for the same term on other platforms, such as Scopus, also indicates an increasing trend in bibliometric research.

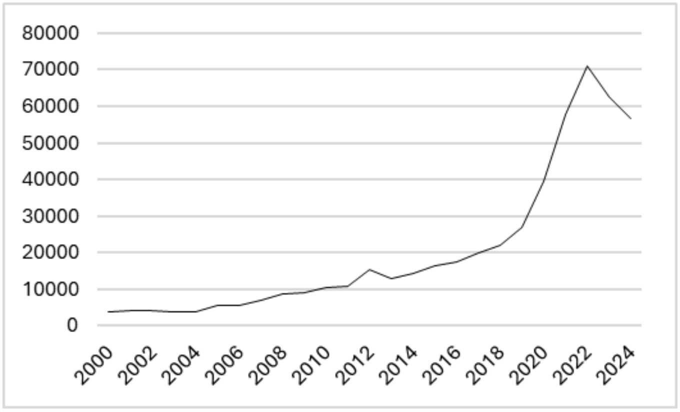

Fig. 1 Growth of Bibliometric Research between 2000 and 2024. Note: The results correspond to research found on Google Scholar. Y-axis corresponds to the number of research results.

It is crucial to reflect on how access restrictions affect not only researchers but also the scientific community as a whole. These barriers to access not only limit the ability of researchers to conduct their studies but also perpetuate inequality in access to information, ultimately impacting the equity of scientific knowledge advancement. Considering that a bibliometric analysis involves searching and reviewing a substantial amount of academic literature, and in light of the points raised in the first paragraph of this section, it is reasonable to assert that conducting bibliometric analyses becomes quite challenging when the researchers in question do not possess the appropriate credentials and/or subscriptions necessary to locate and obtain the required academic literature.

Therefore, the method proposed in this paper not only addresses the issue of managing a large number of academic literature articles without the need for credentials and subscriptions but does so in an ethical manner as well. By facilitating broader access to valuable research, we contribute to a more equitable scientific environment, fostering collaboration and knowledge sharing that can lead to more informed and impactful outcomes.

VIII. Discussion and limitations

This project has undergone a significant number of stages, encompassed various phases of tests and updates, both in terms of the code itself and the approach taken to tackle the problem at hand. Each phase has contributed to shaping the development of the final version.

The original conception of this code traces back to a larger bibliometric analysis project that required handling a vast quantity of articles for review, near two thousand bibliographic references. Most of the articles used in this project were retrieved by leveraging APIs and utilizing relatively simple Python scripts. However, it is important to note that a considerable portion of the articles could not be obtained due to several constraints, which include but are not limited to paywalls and restrictions tied to the university's subscription model with different platforms. Additionally, some articles had publication dates that were either exceedingly old or, conversely, too recent (in certain cases, editions were so newly published that they were not yet available in online repositories).

Despite these difficulties, among the set of articles that could not be downloaded or retrieved through conventional means, a significant number were, in fact, available on the platform ResearchGate. These articles were often presented as working paper versions, accepted manuscripts that had not yet been officially published, drafts, or other similar formats. These versions contained the critical information needed for the project and, in many cases, were either available for direct download or could be requested directly from the authors.

During this initial phase, the code performed with some issues that could be tackled having some creativity in the code implementation. For example, the algorithm worked well when the researcher was logged-in in ResearchGate, so we added a couple of lines of code that allowed the researcher to write the log-in credentials, or even the researcher could just save the credentials in the code, and the code runs without being interrupted to write the credentials. Other issue was that, when a researcher clicks on the button that allows to make a request of the searched article, a small window appears where the researcher could write a note or message to the author of the article. This window interrupted the code flow, but it was resolved without any problems.

Another problem was that, in some cases, the searched DOI code was not available in ResearchGate, again, with some other code lines, this issue could be resolved. After some other minor correcting iterations, the code appeared to be in one of the final forms.

After some time, this same code was implemented to carry out the search for literature pertaining to the topic of climate and socioeconomic vulnerability analysis in coastal communities, which happens to be the central theme of my master's thesis. For this purpose, the search was again automated using APIs and simple Python scripts, although, as in the previous phase, a portion of the articles had to be manually reviewed due to the aforementioned restrictions. The code was executed for about two hundred bibliographic references and worked as expected, yielding the desired results.

However, after the two previously mentioned successful iterations, the code failed to perform as expected. The first obstacle encountered was that ResearchGate had implemented a captcha verification process, designed to ensure that the search is being conducted by a human. This is a commonly employed security measure on many websites. The most straightforward solution was for the researcher to monitor the execution of the code and manually complete the captcha verification when the prompt appeared. This approach proved effective on a single occasion. Unfortunately, in subsequent attempts, even after the researcher completed the verification, the captcha prompt would continue to reappear, effectively blocking access to the website and halting the search process.

For further context, it is necessary to understand that, when automating a web scraping task in Python, the initial outcome typically involves opening a browser window in a sort of test environment. This browser window operates with limited functionality, for example, all open sessions in the browser are closed by default, extensions are disabled, and no browsing history or other stored information, such as cookies, is accessible. These limitations are standard when running automated scripts of this nature.

Another alternative solution was attempted to address this issue. The approach involved modifying the code to ensure that the browser window that opens during the search is a standard browser window, with all functions available, including the presence of active sessions, stored browsing history, cookies, and cache. This solution initially proved successful, as it allowed the search to proceed and facilitated the download or request for the desired articles. Nonetheless, as before, this solution only functioned successfully on a single occasion.

During each of these iterations, the code was meticulously modified in an effort to adapt it to the restrictions imposed by the website. This was done through a rigorous process of trial and error. However, following the iteration mentioned in the preceding paragraph, the code ceased to function. Subsequent attempts (leading up to the version of the code presented in this project) were entirely unsuccessful, and the code did not operate as intended.

Another limitation encountered during the implementation of the project, non-related to the code, pertains to the response time of authors regarding the articles that were either downloaded or requested. While the initial stages of the code (prior to the captcha and other limitations) successfully initiated the download process or submit requests for specific articles identified during the search, it is important to note that the authors to whom these requests are directed often take an extended period to respond. In certain instances, the responses have been provided several years after the initial request was made. This delayed feedback can significantly impede the progress of research, as the availability of pertinent information is crucial for timely analysis and the advancement of academic inquiry. Consequently, this aspect highlights a critical consideration in the overall efficacy of the approach adopted in this project, emphasizing the need for more immediate access to scholarly resources to facilitate ongoing research endeavors.

Additionally, one limitation of this code is that it does not extend to all user preferences regarding browser selection and error handling. The decision to focus exclusively on Chrome and omit features like error storage and logging makes the algorithm simpler but sacrifices some flexibility and functionality. For instance, users who prefer browsers other than Chrome must modify the code manually to incorporate their preferred driver, and any errors encountered during execution are not logged for later review. These trade-offs were made to prioritize the speed and simplicity of the automation process.

The algorithm works for other platforms, considering the sequence of scraping and data extraction and input, but keeping in mind the necessary changes in the code (web addresses, search boxes, etc.). Google Scholar is a good example of this, nonetheless, the accessibility of Google Scholar to articles (in the sense of, for example, the PDF files, or the contact of the researchers) is far limited compared to ResearchGate.

IX. Conclusions

The methodology we implemented in this study demonstrates a significant step forward in automating the retrieval of scientific articles, particularly from platforms like ResearchGate. Our approach, centered around Python-based automation, showcases its efficacy by reducing manual labor and time-consuming searches. However, it is essential to emphasize that this method is effective only when the platform allows for such automated processes, highlighting the need for platforms to facilitate automation through secure means, such as APIs.

ResearchGate's current limitations on web scraping, including the introduction of captchas, present obstacles that could be addressed by adopting an API-based approach. This would align ResearchGate with other leading academic platforms like Scopus and Web of Science, which offer secure, subscription-based APIs, ensuring both controlled data access and broader research capabilities. By allowing researchers to automate data retrieval ethically and securely, especially those from underfunded institutions or developing countries, ResearchGate could significantly enhance global research accessibility. This would not only democratize access to scientific knowledge but also foster collaboration and innovation within the academic community.

Acknowledgment

The authors of this article would like to express their gratitude to LACCEI and LEIRD for providing the opportunity to contribute to the advancement of science through this project, especially the reviewers of this work, for their insightful contributions. We also want to thank ESPOL and the Faculty of Social Sciences and Humanities (FCSH) for their continuous support, and to those who, with their dedication and hard work, are driving the progress of science through open resources.

References

[1] L. Waltman and V. Larivière, "Special issue on bibliographic data sources," *Quant. Sci. Stud.*, vol. 1, no. 1, pp. 360–362, Feb. 2020, doi: 10.1162/qss_e_00026.
[2] A. Velez-Estevez, I. J. Perez, P. García-Sánchez, J. A. Moral-Munoz, and M. J. Cobo, "New trends in bibliometric APIs: A comparative analysis," *Inf. Process. Manag.*, vol. 60, no. 4, p. 103385, 2023, doi: https://doi.org/10.1016/j.ipm.2023.103385.
[3] D. Trezza, "To scrape or not to scrape, this is dilemma. The post-API scenario and implications on digital research.," *Front. Sociol.*, vol. 8, p. 1145038, 2023, doi: 10.3389/fsoc.2023.1145038.
[4] R. Mitchell, *Web scraping with Python: Collecting more data from the modern web*. " O'Reilly Media, Inc.," 2018.
[5] N. Haddaway, "The Use of Web-scraping Software in Searching for Grey Literature," *Grey J.*, vol. 11, pp. 186–190, Oct. 2015.
[6] R. Harder, "Using Scopus and OpenAlex APIs to retrieve bibliographic data for evidence synthesis. A procedure based on Bash and SQL," *MethodsX*, vol. 12, p. 102601, 2024, doi: https://doi.org/10.1016/j.mex.2024.102601.
[7] ResearchGate, "About ResearchGate." [Online]. Available: https://www.researchgate.net/about.
[8] ResearchGate, "Contacting other researchers." [Online]. Available: https://help.researchgate.net/hc/en-us/articles/14292678631825-Contacting-other-researchers
[9] Microsoft, "Visual Studio Code: Getting Started." 2024.
[10] Python Software Foundation, "Python 3.8.20 Documentation." 2024. [Online]. Available: https://docs.python.org/3.8/index.html
[11] Epion, "How to create Python Virtual environment within a python

script.” [Online]. Available: https://stackoverflow.com/a/57921630

[12] Meow_Programmer, “How to skip to next url if element is not found or if timeoutexception occured in selenium wait until function.” [Online]. Available: https://stackoverflow.com/questions/63804694/

[13] Robertpsierre, “How to send ESC key to close pop up window using Python and Selenium?” [Online]. Available: https://stackoverflow.com/a/72039646

[14] S. Beatty, “Accelerate academic research using Scopus APIs.” [Online]. Available: https://blog.scopus.com/posts/accelerate-academic-research-using-scopus-apis

[15] Elsevier, “Elsevier Developer Portal.” [Online]. Available: https://dev.elsevier.com

[16] V. Strobel, “«Pold87/academic-keyword-occurrence: First release».” 2018.

APPENDIX 1: PYTHON CODE

```
import os
import subprocess
import sys
from openpyxl import *  # Imports functions to handle Excel files
from selenium import webdriver  # Imports functionalities for automatic browser handling
from selenium.webdriver.common.keys import Keys # Imports functionalities for automated text input with specific instructions
from selenium.webdriver.common.by import By  # Imports functionalities for automatic search of specific elements (Name, ID, XPATH, etc.)
from selenium.webdriver.chrome.service import Service as ChromeService
import time

# Creates the venv (Virtual environment), obtained from: https://stackoverflow.com/a/57921630
venv_dir = os.path.join(os.path.expanduser("~"), ".venv")

# Installs packages automatically so the user doesn't have to install them manually
# The install function is called to install packages using an automated pip command, obtained from: https://stackoverflow.com/a/50255019
def install(package):
        subprocess.check_call([sys.executable, "-m", "pip", "install", package])

install("selenium")  # Package for web scraping
install("openpyxl")  # Package for handling Excel files

# Configures the service for Chrome WebDriver
service = ChromeService(executable_path='.') #Replace here with the path of the ChromeDriver
driver = webdriver.Chrome(service=service)

# Accesses the ResearchGate portal, specifically the login panel
driver.get("https://www.researchgate.net/login")
user = driver.find_element(By.XPATH, '//*[@id="input-login"]')  # Finds the email input field
passw = driver.find_element(By.XPATH, '//*[@id="input-password"]')  # Finds the password input field

user.send_keys("Insert e-mail here") #Replace here with your ResearchGate e-mail
passw.send_keys("Insert password here") #Replace here with your ResearchGate password
driver.find_element(By.CLASS_NAME, "nova-legacy-c-button__label").click()
time.sleep(60) #In this line the code stops working

# Selects the Excel file. The Excel file must be in the folder. Do not change the name of the Excel file; DOIs should be entered in the first column of the Excel file.
excel = load_workbook(filename="papers.xlsx")
hojatrabajo = excel["Hoja1"]  # Selects the sheet from the selected Excel file

for col in hojatrabajo["A"]:
    try:
        # Searches for papers in each row of the selected column
            buscar = driver.find_element(By.ID, "header-search-action")
          buscar.send_keys(col.value)  # Searches for the selected paper
        buscar.send_keys(Keys.ENTER)
        driver.find_element(By.XPATH, '/html/body/div[1]/div[3]/div[1]/div/div/div[1]/div[2]/div/div/div/div[2]/div[2]/div[2]/div/div[1]').click()
        time.sleep(5)
         webdriver.ActionChains(driver).send_keys(Keys.ESCAPE).perform()  # Obtained from: https://stackoverflow.com/a/43437439
    except:
        continue  # Obtained from: https://stackoverflow.com/a/63807621
```

APPENDIX 2: README.TXT

For this piece of software, the user must have the following programs installed:
1) Python
2) A code editor. We used Visual Studio Code.
3) ChromeDriver installed.
4) ChromeDriver and the Google Chrome version must be compatible

For the Excel file, it must not be modified. The DOI codes must be on the first column of the document

APPENDIX 3: EXCEL FILE

(Included in supplementary material).

APPENDIX 4: REPOSITORY ACCESS

The code can be accessed through the following link: https://github.com/EconHaas/researchgate.git