Web Scraping US White House Press Briefings

By Anthony Lighterness

Table of Contents

Introduction

The White House news briefings are a source of information pertaining to official US government actions. Since it is a US presidential election year, voters can benefit from accessing summarised, analysed, and critiqued insights on the actions taken during the first term of the current US president, Donald J. Trump. Analysis of this content could empower undecided voters to make a more informed decision in time for the 2020 US presidential election. Furthermore, the white house website is not copyright protected as the news briefings contain facts and information exclusively, making it an ideal source for web-scraping (The White House, 2020).

Finally, the layout complexity of the the news webpage HTML and each news briefing therein is appropriate. To the best of our knowledge, only one report to date has published a text-based analysis of the US press briefings, contrasting the overall sentiments between the press journalists and administration of several US presidents (Gutbrod-Meyer & Woolley, 2020). As such, the current study implemented a web-scraper to crawl through each White House news briefing to extract key data for future text analysis.

Methods

The Selenium Webdriver was used to navigate and scrape key data items from each news briefing accessible at https://www.whitehouse.gov/news/. To date, over 6,000 news briefings are available. The webscraping protocol implemented can be summarised into three key steps: retrieve and store news article URLs, define a user function to specify the elements of interest, and then apply the user function in a webdriver to extract the specified data items and stored these into a dataframe. This workflow is described in more detail as follows.

Firstly, the Selenium library was used to import the following packages: “webdriver”, “By”, “WebDriverWait”, “expected_conditions”, and “TimeoutException”. Pandas was also used to handle the dataframes. Therefore, the Selenium Chrome webdriver was used to firstly navigate through all 835 news pages, retrieving and storing the URLs of each news document. To do this, a simple for-loop was defined to iterate through each of 835 news pages. To retrieve the URLs of each news article, the webdriver was specified to identify the elements of each news title, and extract the URL attribute, defined by “href”. This can be seen in Appendix 1. Then, these were stored, ensuring the WebDriverWait function was activated. This allowed all news articles’ URLs to be retrieved before the webdriver scrolled down and clicked on the next page. Following URL extraction, the webdriver was terminated.

Next, a user defined function was constructed to identify the elements of each article, which was applied in the final Selenium Chrome web-scraper. Five key data items were specified to be identified and stored. These include date, category, style, title, and transcript. It should be noted that the category and style of the articles are different. While the former describes a possible theme or area of interest of that briefing, e.g. science & technology or healthcare, the style specifies the format of briefing, e.g. remarks, letter, etc. Using the elements' Xpath, class name, and CSS selector, the data and category, style and title, and transcript were identified and stored, respectively. A try-except statement was used to embed the data retrieval, because some news articles lacked the category or style variables. In these instances, an "NaN" value was appended.

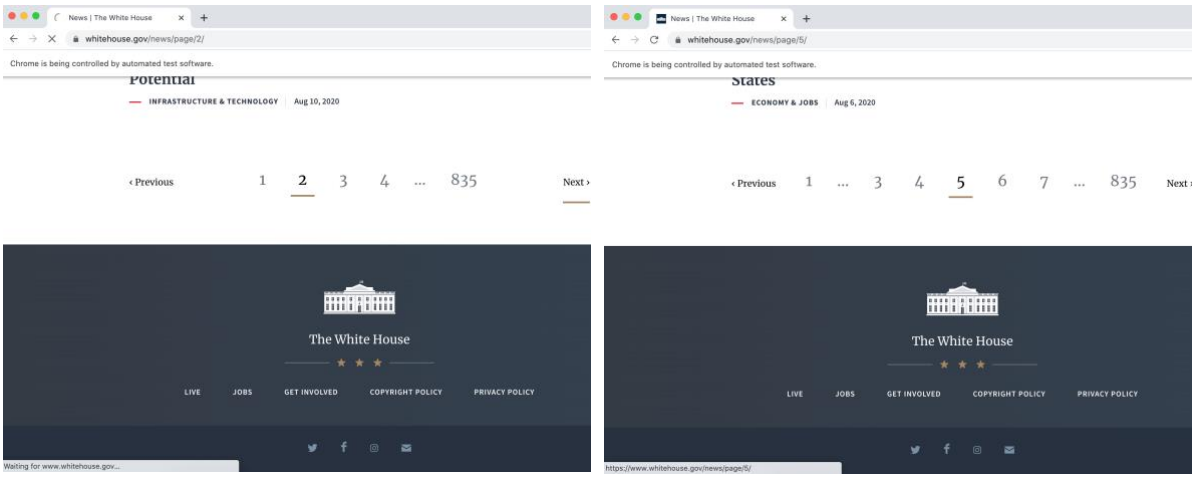
Finally, the Selenium Chrome webdriver applied the user function the retrieve and store the specified data items by accessing the news URLs. Using a for-loop, the Chrome webdriver iterated through each news article URL and was instructed to apply the user function to extract the specified data items. The final pandas dataframe was then exported as an excel document for future text analysis.

Results and Discussion

Overall, the Selenium Chrome webdriver successfully scraped all specified data items from the news briefings available to date. The final webscrape protocol required 22,297 seconds, i.e. ~6.2 hours, to complete. Figure 1 below shows the webcrawler in action as it extracted and stored the URLs of each news article located on all of 835 pages. Figure 2 below shows that when the output dataset was a CSV file, it contained artefacts that makes it unideal for text analysis. Therefore, it was decided to output the dataset as an xlsx file instead. This produced a clean, useable text dataset.

While the current study found that the designed webcrawler and webscraper were successfully implemented, the need for two key concessions in place suggest future work can be done to optimise it. Firstly, the waiting period needed tuning to ensure that the webscraper performs its iterations swiftly but allows enough time to pass before the page is loaded. As such, this may need adjustment depending on the user's internet speed and can in some instances produce a timeout error. Secondly, the webscraper produced in some instances an 'element not found'. This was surprising for elements such as date and title, because each news article clearly contains both variables in question. However, this was overcome by including the element extraction with try-except statements.

A B



C

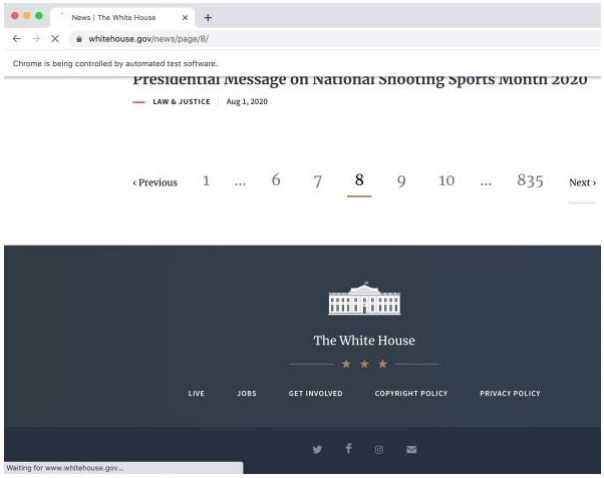


Figure 1 Screenshots of 'web-crawler in action', extracting URLs of each news briefing of each webpage.

CSV Output XLSX Output

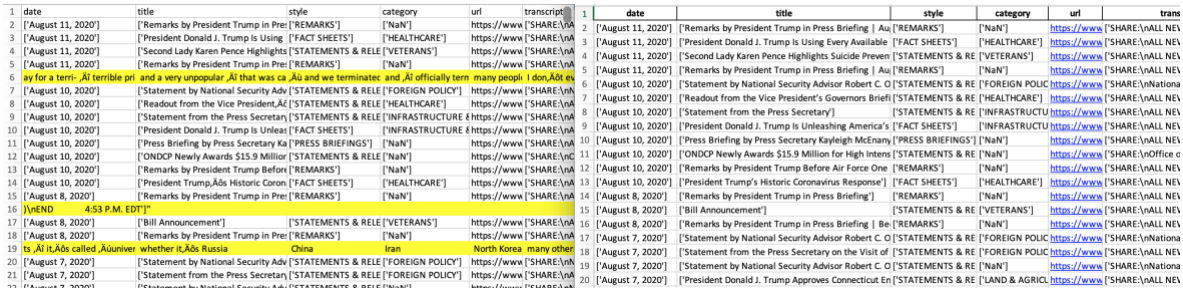


Figure 2: Text corpus outputs as CSV and XLSX files, highlighting artefacts on the left.

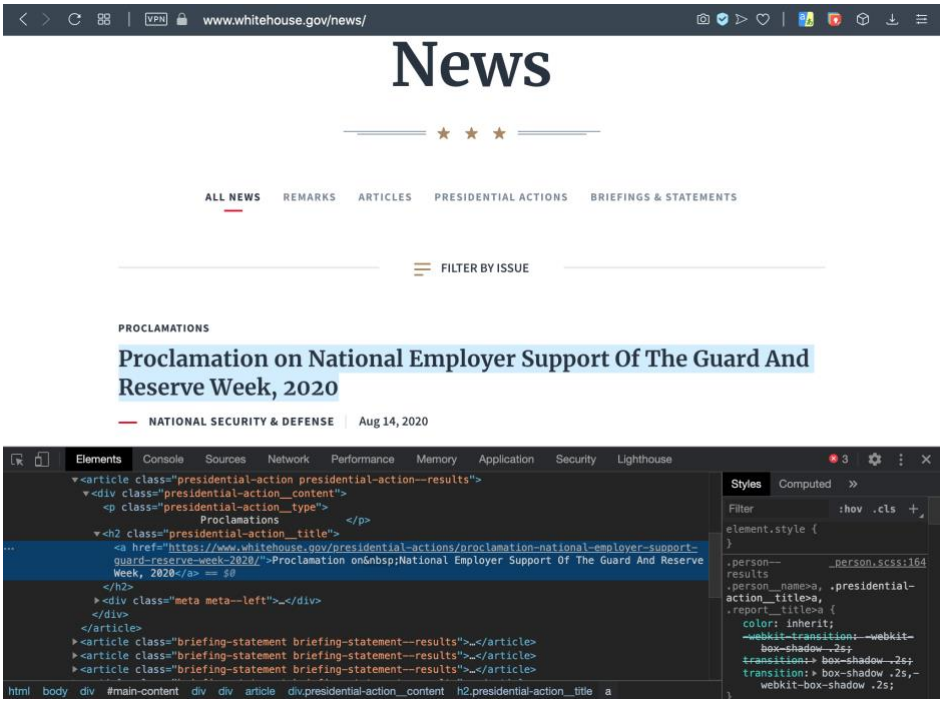
Conclusion

In summary, a Selenium Chrome webdriver was successfully implemented to extract text data from the official White House news briefings. Since this is a US presidential election year, the output corpus of text data herein can be used in future NLP projects, such as sentiment analysis or text summarisation. While the current study reports a successful implementation of the Selenium Chrome webdriver for web-crawling and -scraping, future work may need to further investigate optimising the waiting time to avoid potential timeout errors. Another recommendation is to explore the use of the BeautifulSoup or prettify packages to remove any unwanted elements in the output text corpus. However, this can be easily achieved in a future body of work conducting natural language processing on it.

References

Gutbrod-Meyer, J., and Woolley, J. (2020). New Conflicts in the Briefing Room: Using Sentiment Analysis to Evaluate Administration-press Relations from Clinton through Trump. *Political Communication*. Doi: 10.1080/10584609.2020.1763527

Appendix 1




```
# Initialise ChromeDriver
driver_path = '/usr/local/bin/chromedriver'
driver = webdriver.Chrome(driver_path)
driver.get('https://www.whitehouse.gov/news/')
driver.implicitly_wait(1) # short waiting time

# Initialise empty URL list for Each News Briefing
briefing_list = []

# Please note, when subsequent press briefings are stored, the total page
# number may change, requiring adjustment for the range(n). At the time of
# development, 835 pages were available. A new reader can also reduce the range(n)
# to a smaller integer like 2 or 3 to see the web crawler in action.

for i in range(835):
    # Define the URL for each briefing on every page
    briefings = driver.find_elements_by_class_name('briefing-statement__title')

    # For each briefing URL, append to the initialised empty list.
    for i in briefings:
        briefing_list.append(i.find_element_by_css_selector('a').get_attribute('href'))

    # Define the next page button to be clicked to access all pages
    element = WebDriverWait(driver,5).until(EC.element_to_be_clickable((By.CLASS_NAME,'pagination__next')))
    driver.execute_script("return arguments[0].scrollIntoView();", element)
    # Click next page
    element.click()

driver.quit()
```

Appendix 1: Webcrawler identifying the URLs of each White House news briefing to be stored in a list.