Assignment 3

Document 2: WebCrawler

MA5851 Data Science Master Class 1

James Cook University

Sukhchain Singh Bhathal

13863320

Domains:

- **Website URLs to be crawled:**
  For this project mainly I have selected two websites with 3+ URLs:
  https://www.bbc.com/news
  https://www.bbc.com/news/world/asia
  https://www.abc.net.au/news/
- **Coverage of the chosen domains on the issue:**
  As the topic is related to news sensationalism so the BBC news and ABC news are the widely famous websites with global audiences as well as a large variety of content related to almost all the topics.
  There are many more websites also available put I have discarded them due to the copyright issue or due to the DOM (Document Object Models) complexity due to the project duration and scope.
- **The Natural Language data, meta-data, or other data on each domain and how these data align to the issue:**
  On BBC news website has rich content as well as other useful metadata such as date, time, author name, etc which can play a crucial role in the development of NLP systems. As on the ABC News website, the content was huge so I have decided to extract only a small portion of the data in order to save time and memory. From the BBC news, I have scraped almost 500 stories whereas from the ABC news I scraped almost around 400 news.
- **Copyright of the chosen domains:**
  BBC news has copyright restrictions on its data use. However, I believe I have not breached their copyright as I have fetched only a limited amount of data and secondly, I will not use this data for my project to deliver the end results to any client.
  https://www.bbc.co.uk/copyrightaware/copyright-permissions

  ABC News also has the same policy related to the use of their content. However, with the help of a web crawler, I was able to scrap the data from their website but the content which I had fetched is copyright, so I cannot use this content for my personal as well as for any client use.
  https://abcnews.go.com/Technology/RSS/story?id=32076

WebCrawler workflow:

- **Technology components used for the web crawler:**

  For this project's web crawler I have used the selenium package from python. Selenium was originally was developed to perform web testing but it has many components which are very suitable for web scrapping. **Selenium has the plus point as it can send web requests as well as it can also parse HTML** and can also load javascript without additional requests both are the advantages over beautiful soap and scrapy respectively.

- **The complexity of the domains and where the targeted data resides:**
- **Sequencing of the crawler(s), using the complexity, data structures, and website access restrictions to optimize the crawler:**

  Firstly, I have imported the required packages for the project and for the web scrapping.

```
import selenium
from selenium import webdriver
from selenium.webdriver.firefox.options import Options
from webdriver_manager.chrome import ChromeDriverManager
import pandas as pd
import time
from selenium.webdriver.common.by import By
```

Code Block 1: Importing packages

Code block 1 imports the selenium, pandas as well as other required packages for the project. Once I have all the required packages I then move further to perform the actual coding where firstly, I have defined the URL's to scrap after that I called the `scrap_urls` The function will scrap the web pages as well as their inner pages content (pagination) with the help of a selenium scraper.

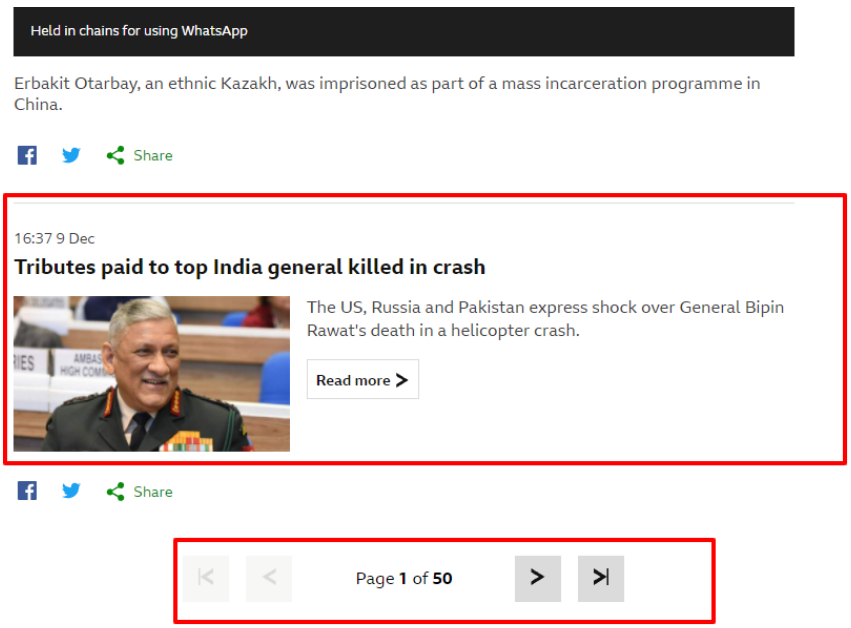

Image 1: BBC News scraper section

Image 1 shows the section of the BBC news website which my scraper will scrap and at the bottom, there is pagination which my code will read and get the content for each of the pages. After scraping the title, summary, author, category from the website all will be concatenated into a data frame for further processing.

- **Data storage:**

  In order to save the scraped data, I have used the two files. One file named "**scraped_data.csv**" contains all the data directly from the scraper and then the second file named "**prod_data.csv**" contains the dataset in the clean form for the other team members to use.

Demonstrates:

- **Python code extracts use appropriate PEP8 and PEP 256 code formats:**

  I have used the proper format for PEP8 and PEP256 coding practice like I have used perfect function names: `scrap_urls`

```
page_name = [driver.title for story in stories]
headline = [story.find_element(By.CLASS_NAME,'lx-stream-post__header').text for story in
stories]
list_pn.extend(page_name)
list_h.extend(headline)
```

Code Block 2: Showing PEP 8 AND PEP 256 formats

As, It can be seen from **Code Block 2** that, I have used the proper names for page_name as well as the proper variable name for the headline code line too,

- **Robust, fault-tolerant coding practices:**

  For the fault-tolerant practice, I have used python's **try, except, and finally** functions which are very crucial for any project in order to control the flow of the project in any issue while fetching the data or processing it.

```
try:
    section = driver.find_elements(By.XPATH,'a.metalink.sdd5v1')[0].text
except:
    try:
        section = driver.find_element(By.XPATH,'a[href = "/sport/"]').text
    except:
        section = ''
finally:
    seedata.append(section)
```

Code Block 3: Showing the try, except and Finally in action

From code block 3, we can see that fault-tolerant code practice using the try-except blocks which I have used in all the parts of the code as well as in the NLP tasks coding section too.

- **Illustration of the WebCrawler(s):**

  Web Crawler is the main base for this project in order to scrap the data from BBC News and ABC news websites. So, In order to scrap the data from a website I have to follow the following steps:

  1. Select the websites to scrape the data from.
  2. Check their copy-right restrictions in order to be on the safe side.
  3. Decide on the programming language (we have python in this project).
  4. In python, there were many packages that I can use for the data scraping purpose. However, above all, I have selected the selenium package due to its number of benefits on the other packages.
  5. Once I have the programming language and package then, I check the HTML structure of the target websites, from **Figure 1.1** we can see the HTML structure of BBC news for the title of the news.

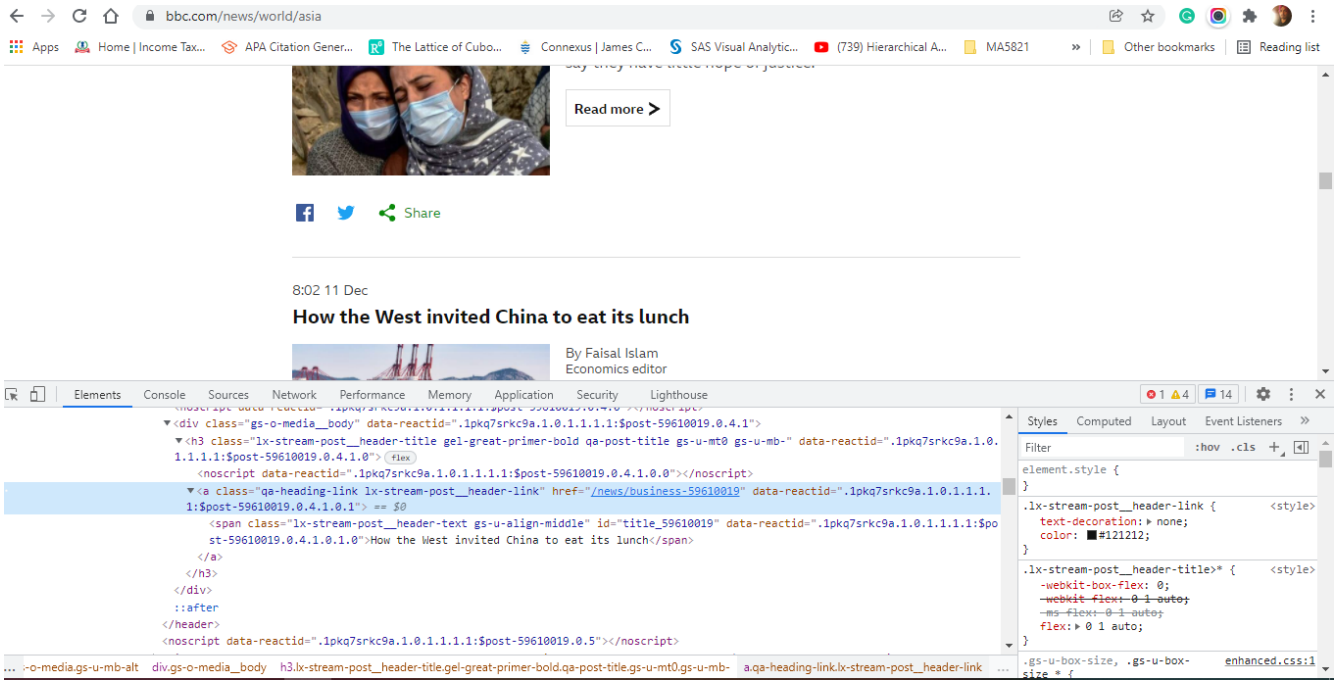

Figure 1.1: BBC News Website HTML Structure.

Figure 1.2 shows the HTML structure of the ABC news website. Here, one thing to notice is that the HTML structure of the ABC news is much easier than the BBC news.

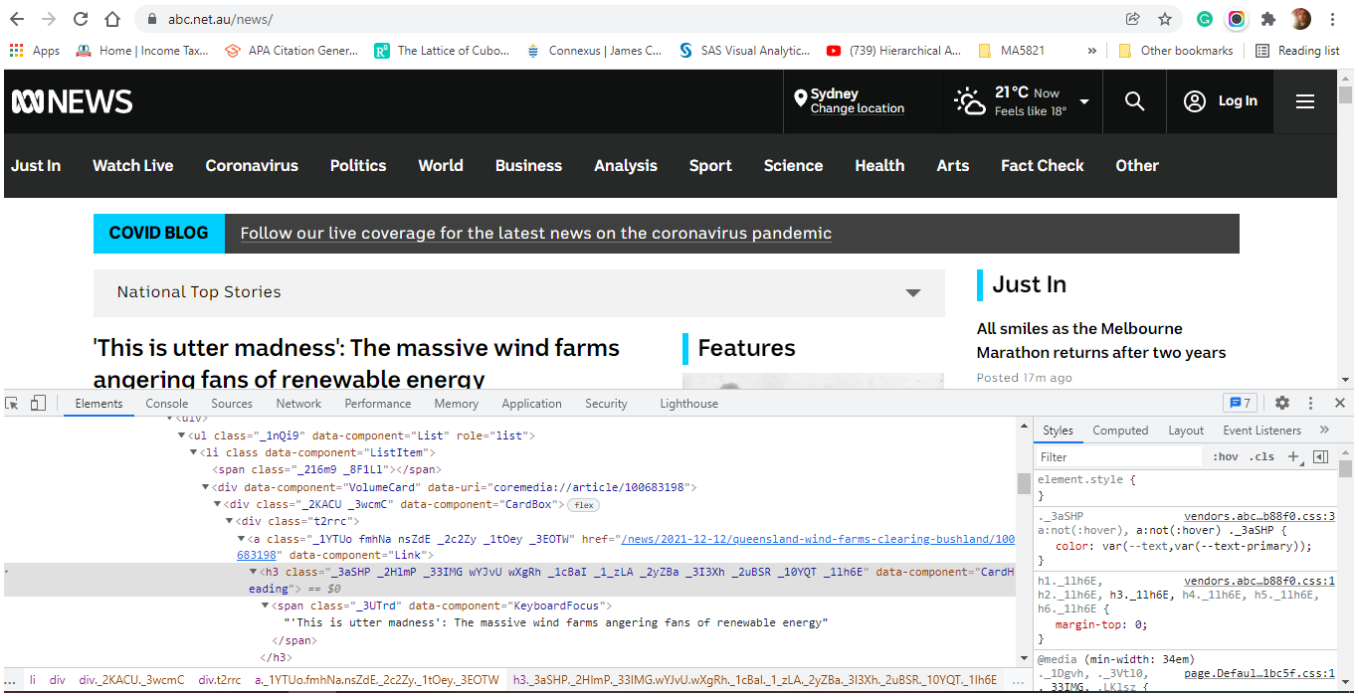

Figure 1.2: Showing the HTML Structure of the ABC News website.

However, the complicated part of this data scraping was the pagination as pagination on the BBC news was a little tricky and which took my most time to code that part.

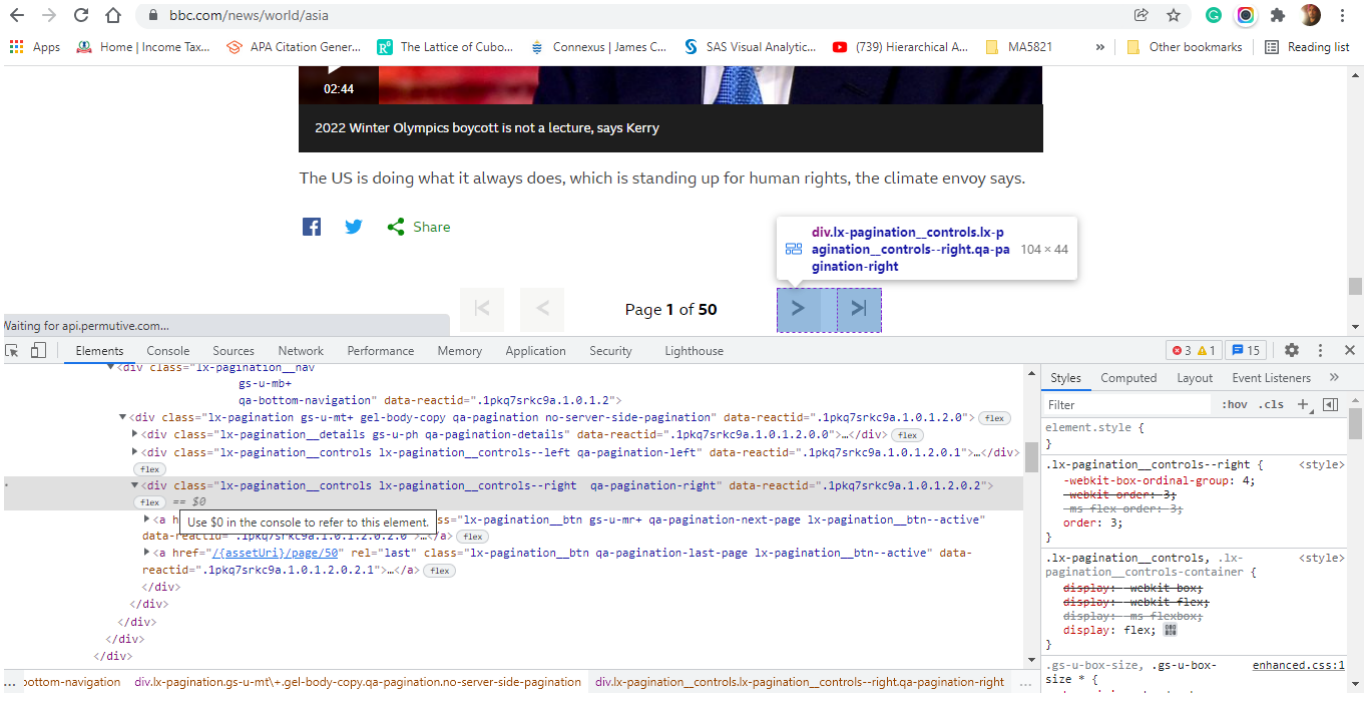

Figure 1.3 Showing the pagination structure of the BBC news website.

Harvested Corpus EDA:

- **Data wrangling of the raw corpus:**

The scraped data is in the raw form so in order to make most of this raw corpus I have used exploratory data analysis in order to clean the data for the team to work on (via the GIT repository). For data wrangling I have followed the following steps:

1. Firstly, I have removed the special characters from the data in order to read the data without any issue.
2. Secondly, I have removed the null values from each of the columns such as headlines, summary, time_stamp, author, etc.
3. Then, I did the enriching of the data which means optimizing the dataset to a more readable form by updating column heading names as well as by optimizing the codebase.

Field	Pre-Cleaning	Post-Cleaning
headline	5,000	3,000
summary	4,868	2,796
categories	5,596	4,783

While scraping the data I did care on the crawler so that at the end I have to do less cleaning work. So while fetching the data **I have fetched only those stores which are from 2020** and I have ignored the rest of the stores while scraping the data. I have also updated all the timestamps to AEST timestamps from the UTC in order to save time on converting them to AEST during the cleaning process.

- **Summary of the generated corpus:**

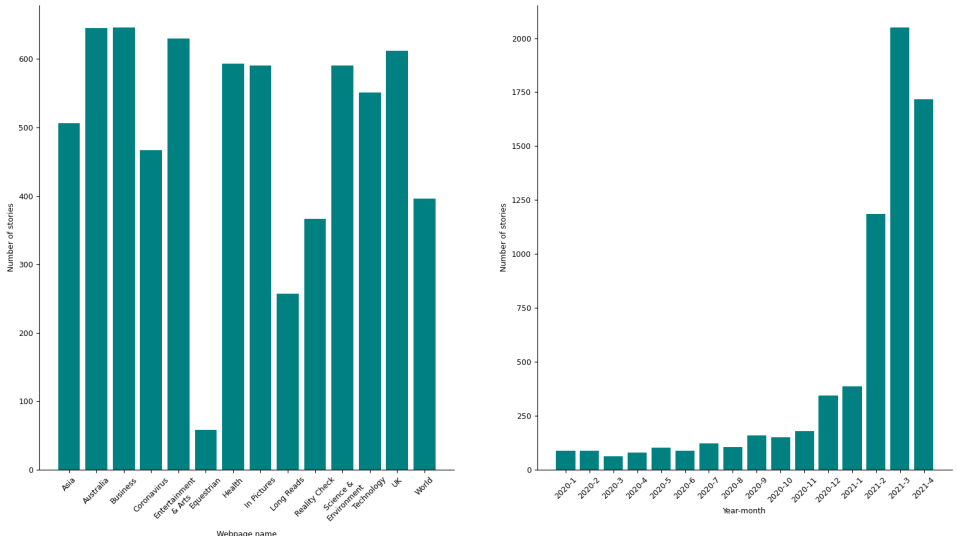

Figure 1.4: Showing the corpus summary

Figure 1.4 shows the data scraping from the web pages and their corresponding number of stories fetched from each of the web pages. So basically the left side graph shows the number of stories fetched based on the web page whereas the right side graph shows the stories fetched based on the year and month.

- **Visualization of the corpus:**

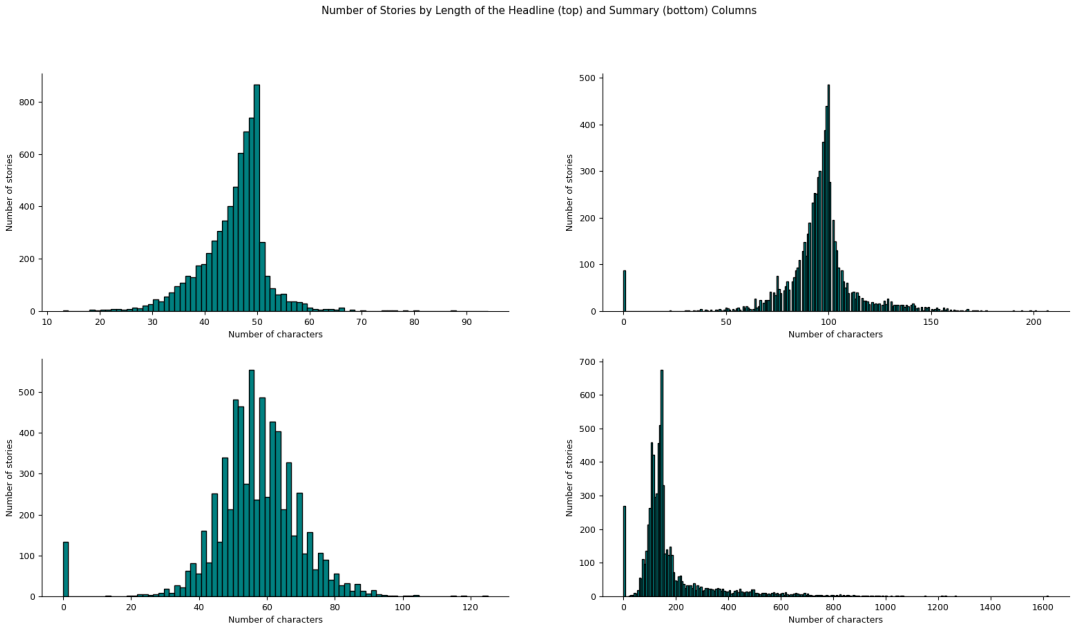

Figure 1.5: Showing the corpus distribution by characters from each column

Figure 1.5 shows the length of the news story headlines in the top columns whereas the bottom columns display the length of each story summary in the scraped data.

- **Descriptive statistics of the corpus:**

Figure 1.5: Showing the word cloud of the corpus

Field	Pre-Cleaning	Post-Cleaning
headline	5,000	3,000
summary	4,868	2,796
categories	5,596	4,783

Table 1: Showing the basic corpus statics.

Table 1 shows the basic statics of the corpus where headlines of news before data cleaning were 5000 however after cleaning the data and removing na values it decreased to 3000. Similarly, for a summary of the news decreased to 2796 and same for categories decreased to 4783.

Word Count: 1591

Python Code:

```
import selenium
from selenium import webdriver
from selenium.webdriver.firefox.options import Options
from webdriver_manager.chrome import ChromeDriverManager
import pandas as pd
import time
```

```
from selenium.webdriver.common.by import By

# settingup the chromedrivers
DRIVER_PATH = 'chromedriver.exe'
options = Options()
options.headless = True
options.add_argument("--window-size=1920,1200")
driver = webdriver.Chrome(ChromeDriverManager().install())
# list of urls for the scraper
url_list = ['https://www.bbc.com/news/world/asia',
            'https://www.bbc.com/news/world/australia',
            'https://www.bbc.com/news/technology',
            'https://www.bbc.com/news/science_and_environment',
            'https://www.abc.net.au/news/']
def scrape_content(url_list, num_pages=2):
    # initialise lists
    list_pn = []
    list_h = []
    list_s = []
    list_hl = []

    for url in url_list:
        # navigate to each url in turn and halt execution for two seconds
        driver.get(url)
        time.sleep(2)

        # locate and extract the webpage name and current page number
        page_name = driver.title
        page_current =
driver.find_element(By.CLASS_NAME,'qa-pagination-current-page-number').text
        # pages_total = driver.find_element
        # ('span.lx-pagination__page-number.qa-pagination-total-page-number').text

        while int(page_current) < num_pages:
            # halt execution for two seconds and then locate the current page number
```

```
        time.sleep(2)
        page_current = driver.find_element(By.CLASS_NAME,
            'qa-pagination-current-page-number').text

        print('Preliminary content scrape in progress on page {} of {} of {}'
              .format(page_current, str(num_pages), page_name[0]))

        # locate all stories on the current webpage
        stories = driver.find_elements(By.CLASS_NAME,'lx-stream__post-container')

        # locate and extract page name, headline, summary/description information and the
hyperlink for each story
        page_name = [driver.title for story in stories]
        headline = [story.find_element(By.CLASS_NAME,'lx-stream-post__header').text for story in
stories]
        list_pn.extend(page_name)
        list_h.extend(headline)

        for story in stories:
            try:
                summary = story.find_element(By.XPATH,'p.lx-stream-related-story--summary').text
            except:
                try:
                    summary = story.find_element(By.XPATH,'p.lx-media-asset-summary').text
                except:
                    summary = ''
            finally:
                list_s.append(summary)

        for story in stories:
            try:
                hyperlink = story.find_element(By.XPATH,'a.qa-heading-link').get_attribute("href")
            except:
                hyperlink = ''
            finally:
```

```
                list_hl.append(hyperlink)

            # locate the pagination ribbon and then the button to the next page, click it and halt
execution for two
            # seconds
            # pagination = driver.find_element(By.CLASS_NAME,'lx-pagination__nav')
            # next_button = pagination.find_element(By.CSS_SELECTOR,'a[rel="next"]')
            #next_button.click()
            time.sleep(2)

        else:
            # let the user know that the while loop is finished (i.e. a progress update)
            print('Done! Proceeding to next webpage.')

    # convert the lists into a set of series and then aggregate into a dataframe
    series_hl = pd.Series(list_hl)
    series_pn = pd.Series(list_pn)
    series_h = pd.Series(list_h)
    series_s = pd.Series(list_s)

    data_dict = {'hyperlink': series_hl,
                 'page_name': series_pn,
                 'headline': series_h,
                 'summary': series_s}

    prelim_df = pd.DataFrame(data_dict)

    # sort the data and drop duplicate headlines
    prelim_deduped = prelim_df.sort_values(by='headline')
    prelim_deduped = prelim_deduped.drop_duplicates(subset='headline', keep='first')

    # split the data by the presence of a hyperlink element
    prelim_hyperlinks = prelim_deduped[prelim_deduped.hyperlink != '']
    prelim_no_hyperlinks = prelim_deduped[prelim_deduped.hyperlink == '']
```

```
    return prelim_hyperlinks, prelim_no_hyperlinks
    for hyperlink in hyperlink_list:
        try:
            driver.get(hyperlink)
            time.sleep(3)

            status = "Success"
            print("Full content scrape in progress for link {} of {}"
                  .format(hyperlink_list.index(hyperlink), len(hyperlink_list)))

            try:
                headline2 = driver.find_element(By.ID,'main-heading').text
            except:
                try:
                    headline2 = driver.find_element(By.ID,'lx-event-title').text
                except:
                    try:
                        headline2 =
driver.find_element(By.XPATH,'h1.gel-trafalgar-bold.qa-story-headline').text
                    except:
                        headline2 = ''
            finally:
                list_h2.append(headline2)

            try:
                summary2 =
driver.find_element(By.XPATH,'div.ssrcss-3z08n3-RichTextContainer.e5tfeyi2').text
            except:
                try:
                    summary2 = driver.find_element(By.XPATH,'b.ssrcss-14iz86j-BoldText.e5tfeyi0').text
                except:
                    try:
                        summary2 =
driver.find_element(By.XPATH,'ol.lx-c-summary-points.gel-long-primer').text
                    except:
```

```
                try:
                    summary2 = driver.find_element(By.XPATH,'p.qa-introduction').text
                except:
                    summary2 = ''
            finally:
                list_s2.append(summary2)

            try:
                timestamp = driver.find_element(By.XPATH,'time[data-testid="timestamp"]') \
                    .get_attribute('datetime')
            except:
                try:
                    timestamp = driver.find_element(By.XPATH,'time.gs-o-bullet__text.qa-status-date') \
                        .get_attribute('datetime')
                except:
                    timestamp = ''
            finally:
                list_ts.append(timestamp)

            try:
                contributor_name =
driver.find_element(By.XPATH,'p.ssrcss-1pjc44v-Contributor.e5xb54n0') \
                    .find_element_by_tag_name('strong').text
            except:
                try:
                    contributor_name = driver.find_element(By.XPATH,'span.qa-contributor-name').text
                except:
                    contributor_name = ''
            finally:
                list_cn.append(contributor_name)

            try:
                contributor_team =
driver.find_element(By.XPATH,'p.ssrcss-1pjc44v-Contributor.e5xb54n0').text
            except:
```

```
            try:
                contributor_team = driver.find_element(By.XPATH,'span.qa-contributor-title').text
            except:
                contributor_team = ''
        finally:
            list_ct.append(contributor_team)

        try:
            category = driver.find_element(By.XPATH,'a.ssrcss-1yno9a1-StyledLink.ed0g1kj1').text
        except:
            category = ''
        finally:
            list_c.append(category)

        try:
            section =
driver.find_elements(By.XPATH,'a.ssrcss-zf4gw3-MetadataLink.ecn1o5v1')[0].text
        except:
            try:
                section = driver.find_element(By.XPATH,'a[href = "/sport/"]').text
            except:
                section = ''
        finally:
            list_se.append(section)

        try:
            subsection =
driver.find_elements(By.XPATH,'a.ssrcss-zf4gw3-MetadataLink.ecn1o5v1')[1].text
        except:
            subsection = ''
        finally:
            list_ss.append(subsection)
startTime_all = time.time()
startTime_prelim = time.time()
prelim_hyperlink_data, prelim_no_hyperlink_data, hyperlink_list = concatenate_content(url_list)
```

```
executionTime_prelim = (time.time() - startTime_prelim)
print('Time taken to scrape preliminary content: {} minutes.'.format(str(round(executionTime_prelim, 2) / 60)))

startTime_det = time.time()
bbc_data = concatenate_content(hyperlink_list, prelim_hyperlink_data, prelim_no_hyperlink_data)
executionTime_det = (time.time() - startTime_det)
print('Time taken to scrape detailed content: {} minutes.'.format(str(round(executionTime_det, 2) / 60)))

executionTime_all = (time.time() - startTime_all)
print('Time taken for the entire webscrape: {} minutes.'.format(str(round(executionTime_all, 2) / 60)))
```

References:

1. Real Python. (2021, November 10). *Beautiful Soup: Build a Web Scraper With Python*. Retrieved December 12, 2021, from https://realpython.com/beautiful-soup-web-scraper-python/
2. Rungta, K. (2021, October 7). *Selenium Webdriver with Python: Tutorial with Example*. Guru99. Retrieved December 12, 2021, from https://www.guru99.com/selenium-python.html