Documentation CNN webscraper

This document will explain how we have built the CNN scraper, what choices we have made in the design process and what exactly the code does.

Libraries

For the scraper we have used several libraries to make the process easier and even doable. For scraping we use the selenium webdriver in combination with BeautifulSoup. Since CNN is a JavaScript heavy website, scraping it would not be possible with BeautifulSoup alone. Further, we have used pandas to combine the scraped search terms, and to do some checks. Since the website formatted the article dates like 'Oct 07, 2023', we have also used datetime to convert this string into datetime to make sure that we could filter dates properly.

The scraper function

We have created a function called 'scrape_ccn_articles', which takes the argument stop_date, search_term, and start_page. The stop date is the date of the last article the scraper has to extract, and then close itself. It is given in the same format as the cnn website, namely 'Oct 07, 2023' for example, as can be seen, this input is converted into datetime. The second argument is 'search_term', which is the keyword that is looked up on the CNN website, for example 'Palestine'. The start_page argument is the page number the scraper starts scraping. For anything related to Israel and Palestine, this would be page 1 since the conflict has not been going on for more than 3 months. For the Russo-Ukrainian conflict, we have manually looked up a page that is close to the first of March 2022, since that is approximately 2 months after the war started.

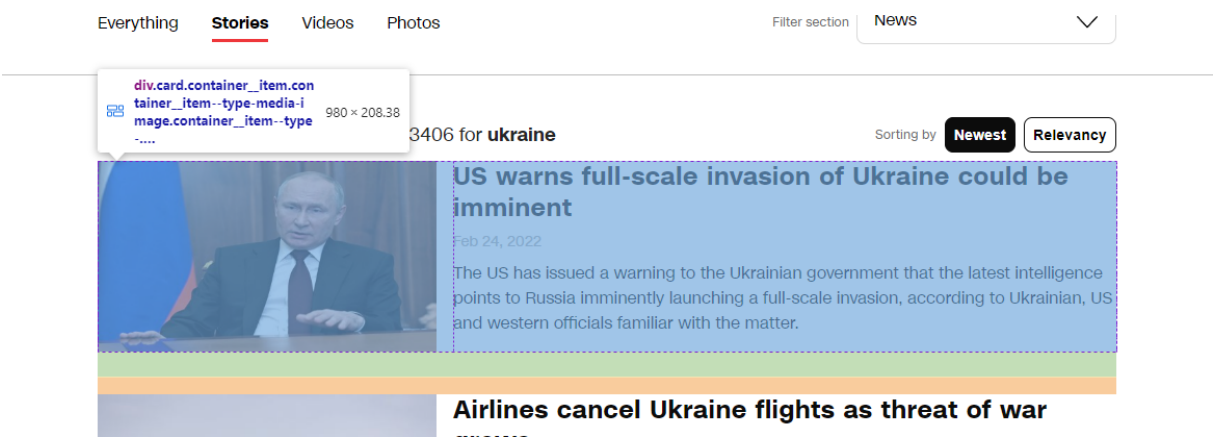
```
def scrape_cnn_articles(stop_date, search_term, start_page):
    options = Options()
    options.headless = False
    driver = webdriver.Chrome(options=options)
    results = []

    stop_date_object = datetime.strptime(stop_date, '%b %d, %Y')
```

The scraper start by going to link that is provided. Going to this link, it takes the arguments 'search_term' and 'start_page'

```
try:
    driver.get(f'https://edition.cnn.com/search?q={search_term}&from=0&size=10&page={start_page}&sort=newest&types
```

If the link works, and it usually does, the website looks for ‘articles’. BeautifulSoup looks for all elements ‘div’, that has card as a class. All articles on the CNN website are structured like this, see the screenshot below.

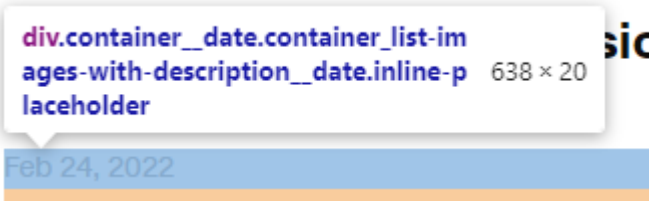
```
while True:
    WebDriverWait(driver, 10).until(
        EC.presence_of_all_elements_located((By.CLASS_NAME, 'card'))
    )

    soup = BeautifulSoup(driver.page_source, 'html.parser')
    articles = soup.find_all('div', class_='card')
```



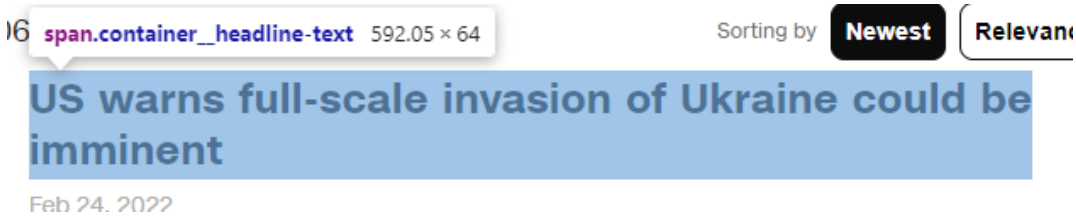
After finding the articles on the webpage, the scraper start extracting information.

```
for article in articles:
    date_element = article.find('div', class_='container__date')
    date_string = date_element.get_text(strip=True) if date_element else 'Date not available'

    date_object = datetime.strptime(date_string, '%b %d, %Y')

    headline_element = article.find('span', class_='container__headline-text')
    headline = headline_element.get_text(strip=True) if headline_element else 'Headline not available'

    link_element = article.find('a')
    link = link_element['href'] if link_element else 'Link not available'

    results.append({
        'Headline': headline,
        'Link': link,
        'Date': date_object
    })
```

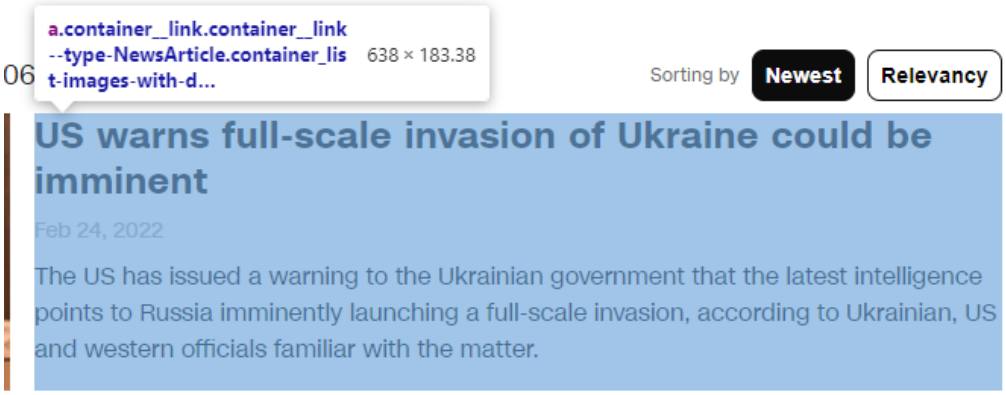
For every article, it looks for the date_element, on the CNN website, the dates of every article is a string with the ‘container__date’ class.



This string then gets converted into datetime, this is important for later, when we have to sort for dates of stop scraping at a certain date. Then the scraper extracts the headline text, namely any element in the article that has the class ‘container__headline-text’.



Finally it looks for the element with the link to the article and the href inside



All this information gets added to the results list, for later.

The articles get added to the lists, until the date of the article is lower than the stop date that we put in the argument of the function. When it hits a date lower than the stop date, the results list gets put into a CSV.

```
if date_object < stop_date_object:
    print(f'Stopping scraping as an article with date {date_string} has been reached.')

    with open(f'{search_term}_scraped_data.csv', 'w', newline='', encoding='utf-8') as csv_file:
        fieldnames = ['Headline', 'Link', 'Date']
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
        for result in results:
            writer.writerow(result)
    print(f'Scraped data has been saved to {search_term}_scraped_data.csv')
    return results
```

Meanwhile, every time the scraper hits a page and the content is extracted, selenium looks for the next page button. It takes 7 seconds before clicking, as to not overload the website. I read somewhere that 7 seconds is good coding etiquette. Sometimes a JavaScript website has a 'Stale element', and the button wont click, if this happens, it will try again. One problem is that it then scrapes the webpage twice, but I solved this later by removing duplicates from the CSV.

```
try:
    next_page_button = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CLASS_NAME, 'pagination-arrow-right'))
    )

    driver.execute_script("arguments[0].scrollIntoView(true);", next_page_button)

    time.sleep(7)

    next_page_button.click()

except StaleElementReferenceException:
    print('Stale element reference. Retrying to find the next page button.')
    continue
```

Finally, after everything is completed and the final date is found, the driver closes.

```
finally:
    driver.quit()
```

Combining the different CSV's

This scraper function is called like below for every term that we agreed on. A few terms for the Israel-Palestine war, and a few terms for the Russian-Ukrainian war.

```
#IDF
scraped_articles = scrape_cnn_articles('Oct 07, 2023', "IDF", "1")
```

For every term there is a CSV created, these CSV get combined using pandas, and since a lot of search terms have the same articles as results, the duplicates get removed. This also solves the problem of double scraped articles because of the Stale Element error.

```
csv_files = ['Gaza_scraped_data.csv', 'Hamas_scraped_data.csv', 'Israel_scraped_data.csv',

dfs = []

for file in csv_files:
    df = pd.read_csv(file)
    dfs.append(df)

combined_df = pd.concat(dfs, ignore_index=True)
combined_df.drop_duplicates(inplace=True)
combined_df.to_csv('israel_palestine_conflict.csv', index=False)
```

Finally, because we started scraping from the latest article for Israel, and for approximately March 2022 for Ukraine, this script filters the dates in the CSV from the start of the conflict to exactly two months after the conflict started. It then returns the results to the same CSV.

```
df = pd.read_csv('israel_palestine_conflict.csv')

df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

start_date = '2023-10-07'
end_date = '2023-12-07'
filtered_df = df.loc[start_date:end_date]

filtered_df.reset_index(inplace=True)

filtered_df.to_csv('israel_palestine_conflict.csv', index=False)
```

This process is the same for the Russia-Ukraine csv's, just different dates.