Northeastern University

INFO 7390- ASSIGNMENT 1- WEB SCRAPPING

EMMANUEL CHIBUA
College of Engineering
002799484 | chibua.e@northeastern.edu

Table of Contents

Executive Summary

This report presents the process and results of a web scraping project targeting the Canadian Weather Alert website with URL- https://weather.gc.ca/index_e.html?layers=alert,#alerttable . The goal of the project was to extract **real-time** weather alert data, specifically focusing on the Province, City, Warning, and Statement information. The data was then saved in both CSV and JSON formats for further analysis. The project was successfully completed using Python, with key libraries including Selenium for browser automation, BeautifulSoup for HTML parsing, and pandas for data manipulation and storage.

Introduction

Web scraping is a method used to extract large amounts of data from websites. The data on the websites are unstructured, and web scraping enables us to convert these data into a structured form. In this project, I targeted the Canadian Weather Alert website, which provides **real-time weather alerts** across different provinces and cities in Canada, with multiple updates on daily basis.

Project Objectives

The main objectives of this project were to:

- Scrape real-time weather alert data from the Canadian Weather Alert website.
- Focus on extracting specific information: Province, City, Warning, and Statement.
- Save the scraped data in both CSV and JSON formats for easy access and further analysis.

Motivation:

The motivation here is to create a simple script in python for acquiring weather data in Canada as often as is required. This may be deployed with a task scheduler to run on hourly basis, updating a master data file which could be in CSV or JSON format. Data gathered here could be used for data analysis and studying the impact of global warming.

Methodology

The project was carried out using Python, leveraging several key libraries:

Tools and Libraries:

- Selenium: A web testing library used to automate browser activities.
- BeautifulSoup: A Python package used for parsing HTML and XML documents. It creates parse trees that are helpful to extract the data easily.
- Pandas: A library used for data manipulation and analysis. It is used to extract data and store it in the desired format.

Data Extraction:

- The Selenium library was used to open the URL of the webpage and parse the HTML content of the website.
- BeautifulSoup was then used to find all HTML elements on the webpage that match specific criteria. The find_all method returns a list of all elements that match the specified tag and attributes.

Data Cleaning and Transformation:

- The extracted data was then cleaned and transformed. Each alert was classified as either a warning or a statement based on its content. The final data included the Province, City, Warning, and Statement, along

with the current date and time for each entry to improve data quality. A detailed explanation of this is done in the Challenges and Solutions part of this report.

Challenges and Solutions

Handling Dynamic Content:

The target website being scraped is dynamic. Traditional scraping methods, which include sending a GET request to the server and parsing the returned HTML, do not work with dynamic websites. To manage this, Selenium, a tool that enables automatic interaction with a webpage in a manner similar to that of a human user was utilized. This made it able to load dynamic content before scraping.

```
# Import necessary libraries
from selenium import webdriver  # Used to interact with the webpage
from bs4 import BeautifulSoup  # Used to parse HTML and extract data
import pandas as pd  # Used to store and manipulate data
from datetime import datetime  # Used to get the current date and time
import pytz  # Used to convert the current time to Eastern Standard Time

# Set up the Chrome webdriver
# This opens a new browser window that the script can control
driver = webdriver.Chrome()

# Open the URL of the webpage you want to scrape
url = "https://weather.gc.ca/index_e.html?layers=alert,#alerttable"
driver.get(url)

# Parse the HTML content of the webpage
# This creates a BeautifulSoup object that you can search to find specific HTML elements
soup = BeautifulSoup(driver.page_source, 'html.parser')
```

Figure 1- Libraries used for handling dynamic content of the webpage.

Initializing Lists and Providing Parameters for Scrapped Data:

After processing the dynamic content of the webpage, we initialized lists to store the scraped data. We then defined parameters to filter through the scraped data, specifically focusing on the ‘Province’ and ‘Warning’ columns.

```
# Initialize lists to store the scraped data
# Each list will become a column in the final DataFrame
provinces = []
cities = []
warnings = []
statements = []
months = []
days = []
dates = []
times = []

# List of provinces to include in the scraped data
province_list = ["Northwest Territories", "Nunavut", "Ontario", "Yukon", "British Columbia", "Quebec", "Nova Scotia", "Manitoba", "Alberta"]

# List of warnings to classify alerts as warnings or statements
warning_list = ["Arctic outflow", "Blizzard", "Blowing snow", "Dust storm", "Extreme cold", "Flash freeze", "Fog", "Freezing drizzle", "Freezing rain",
                "Frost", "Heat", "Hurricane", "Rainfall", "Severe thunderstorm", "Snowfall", "Snow squall", "Storm surge", "Tornado", "Tropical storm",
                "Tsunami", "Weather", "Wind", "Winter storm"]
```

Figure 2- Initializing lists for the columns and parameters for scrapped data.

For the Warning list, please refer to Appendix 6- Weather hazards that trigger warning alerts.

Identifying the Correct HTML Elements:

Pinpointing the exact elements that contain the desired data required careful inspection of the page's HTML. This was addressed by using the browser's developer tools to inspect the HTML structure of the web pages and identify the HTML tags, classes, or IDs that contain the data.

```
# Find all HTML elements on the webpage that match specific criteria
# The find_all method returns a list of all elements that match the specified tag and attributes
province_elements = soup.find_all('b', {'data-v-b8dde33c': ''})
city_elements = soup.find_all('a', {'title': 'Click to see on map'})
alert_elements = soup.find_all('a', {'title': 'Click to see alert information', 'class': 'ga-map-table-alert-link'})
```

Figure 3- Searching for the HTML elements.

Filtering by Province and Classifying Alerts:

In the initial stages of this web scraping project, the algorithm designed to scrape 'Province' data inadvertently picked up some irrelevant information. This was due to the shared HTML tags between the desired and undesired data. To address this, I refined our approach by cross-referencing the scraped data with a predefined list of provinces, ensuring only relevant data was retained.

Similarly, the webpage's structure presented a challenge where 'Warning' and 'Statement' data were encapsulated within the same HTML tag. To differentiate, I utilized a predefined list of hazards that corresponded to 'Warnings'. Consequently, any scraped data not mentioned in this hazard list was classified as a 'Statement'. Refer to Appendices 4 and 5 for clarity.

These adjustments were crucial in the data cleaning process, transforming the raw scraped data into a structured and meaningful format, ready for further analysis and utilization."

```
# If the province and city are found, add them to the respective lists
# This loop goes through each province, city, and alert element and adds their text to the respective lists
if province_elements and city_elements and alert_elements:
    for province, city, alert in zip(province_elements, city_elements, alert_elements):
        # Check if the province is in the list of provinces
        if province.text in province_list:
            provinces.append(province.text)
            cities.append(city.text)

            # Check if the alert is a warning or a statement
            if any(warning in alert.text for warning in warning_list):
                warnings.append(alert.text)
                statements.append(None)
            else:
                warnings.append(None)
                statements.append(alert.text)
```

Figure 4- Filtering province and classifying alerts.

Adding Date and Time:

To enhance data quality, I integrated current date and time in Eastern Standard Time (EST) for each row. This was achieved using the 'pytz' library for time zone handling. Refer to Appendices 7 and 8 for a comparison of weather data before and after this addition.

```
    # Get the current date and time in Eastern Standard Time
    eastern = pytz.timezone('US/Eastern')
    now = datetime.now(eastern)
    month = now.strftime('%B')
    day = now.strftime('%A')
    date = now.strftime('%Y-%m-%d')
    time = now.strftime('%H:%M:%S')

    # Add the current date and time to the lists
    months.append(month)
    days.append(day)
    dates.append(date)
    times.append(time)
```

Figure 5- Adding date and time in Eastern Standard Time

Data Storage:

The final task was to store the scraped data in a practical format. I utilized the Pandas library to save the data in both CSV and JSON formats, ensuring flexibility for further analysis.

```
# Convert the lists to a pandas DataFrame
# This creates a DataFrame where each list is a column and each element in the list is a row
df = pd.DataFrame({
    'Province': provinces,
    'City': cities,
    'Warning': warnings,
    'Statement': statements,
    'Month': months,
    'Day': days,
    'Date': dates,
    'Time': times
})

# Close the driver
# This closes the browser window that the script was controlling
driver.quit()

# Print the DataFrame
# This displays the DataFrame in the console so you can see the scraped data
print(df)

# Save the DataFrame to a CSV file
# This creates a CSV file in the same directory as your script and writes the DataFrame to it
df.to_csv('weather_data.csv', index=False)

# Save the DataFrame to a JSON file
# This creates a JSON file in the same directory as your script and writes the DataFrame to it
df.to_json('weather_data.json', orient='records')
```

Figure 6- Storing data as CSV and JSON files.

Overcoming these challenges allowed me to successfully extract and store data from the Canadian Weather Alert website, utilizing the versatility of Python and its libraries for web scraping. This project underscores the significance of strategic planning and problem-solving in successful project execution.

Results

The web scraping was successful, with data from the Canadian Weather Alert website being extracted in real-time. The data was filtered to provinces, and each alert was classified as either a warning or a statement based on its content. The final data included the Province, City, Warning, and Statement, along with the current date and time for each entry.

Conclusion

This project demonstrated the effectiveness of web scraping in extracting large amounts of real-time data from a website. The use of Python and its libraries simplified the process of automating the browser activities, parsing the HTML, and storing the data.

Future Work

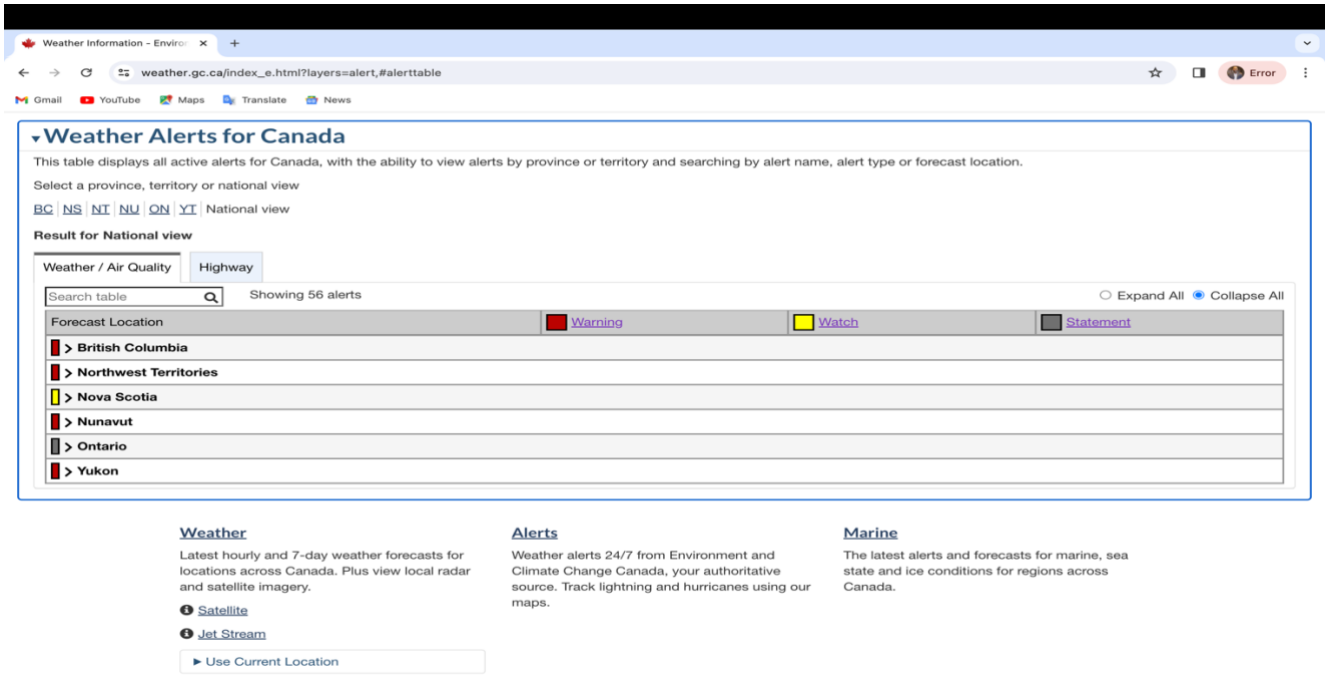
In addition to the improvements mentioned earlier, there are several other enhancements that could be made to this project in the future:

1. **Email Alerts:** One potential enhancement could be the incorporation of an email service to send weather updates to individuals who need them frequently. This could include sailors, airline companies, pilots, and others who rely heavily on weather conditions for their operations. The system could be set up to automatically send emails whenever a new weather alert is posted. This would require the integration of an email service with the web scraping script.
2. **Real-Time Updates:** Another enhancement could be to set up the script to run at regular intervals, providing real-time weather updates. This could be achieved using a task scheduler or a similar tool.
3. **Data Analysis:** The scraped data could be used for further analysis or to feed into a weather alert application. For example, historical weather patterns could be analyzed, or real-time alerts could be provided through a mobile app.
4. **Expand Geographic Scope:** The geographic scope of this project could be expanded to include weather alerts from other countries or regions, depending on the availability and accessibility of the data.

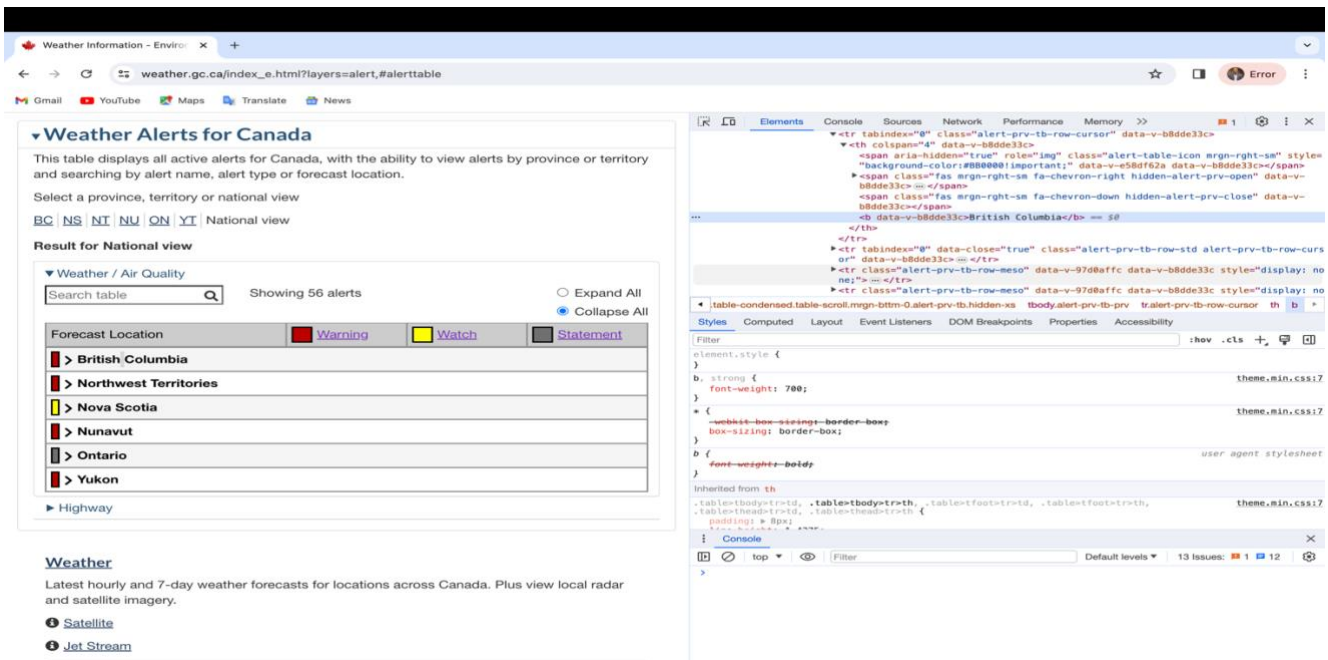
By implementing these enhancements, the project could provide even more valuable and timely information to those who need it most. As always, any future work should be carried out in accordance with web scraping best practices and the terms of service of the websites being scraped.

Appendices

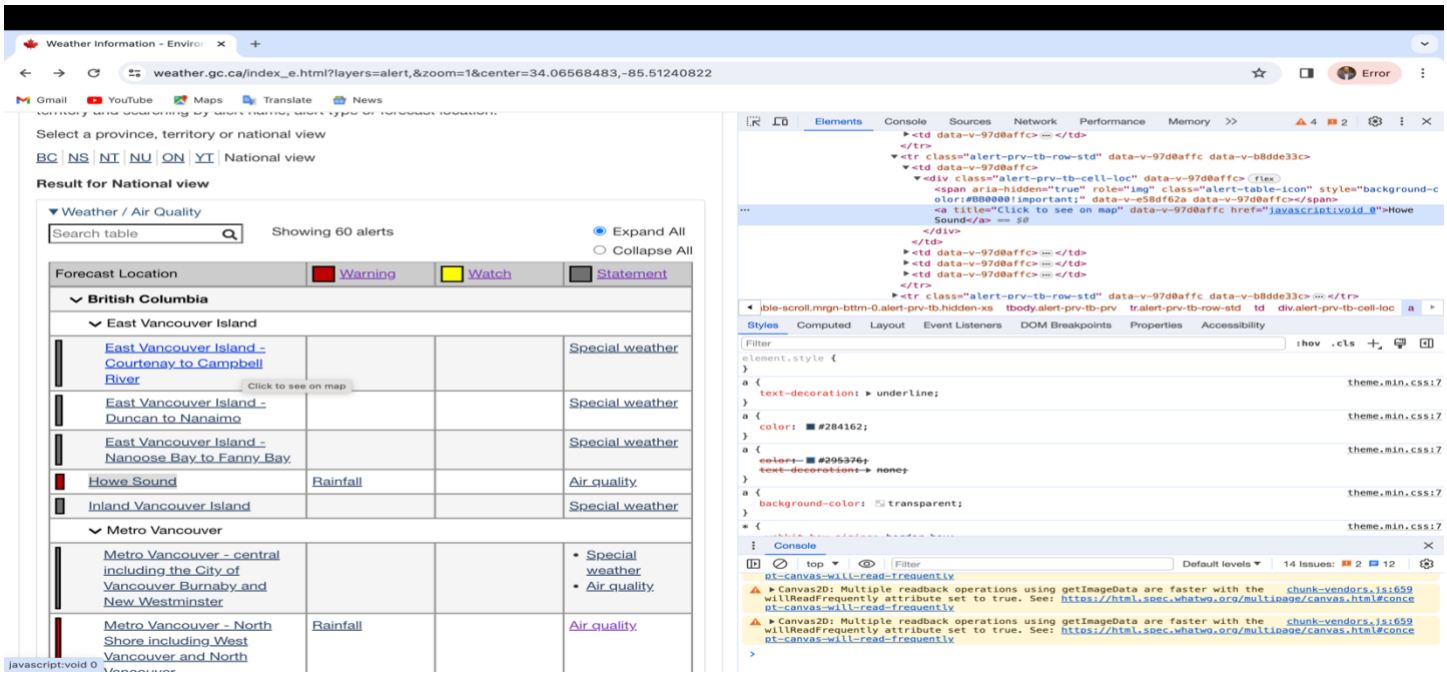
Appendix 1- Weather alert for Canada web page



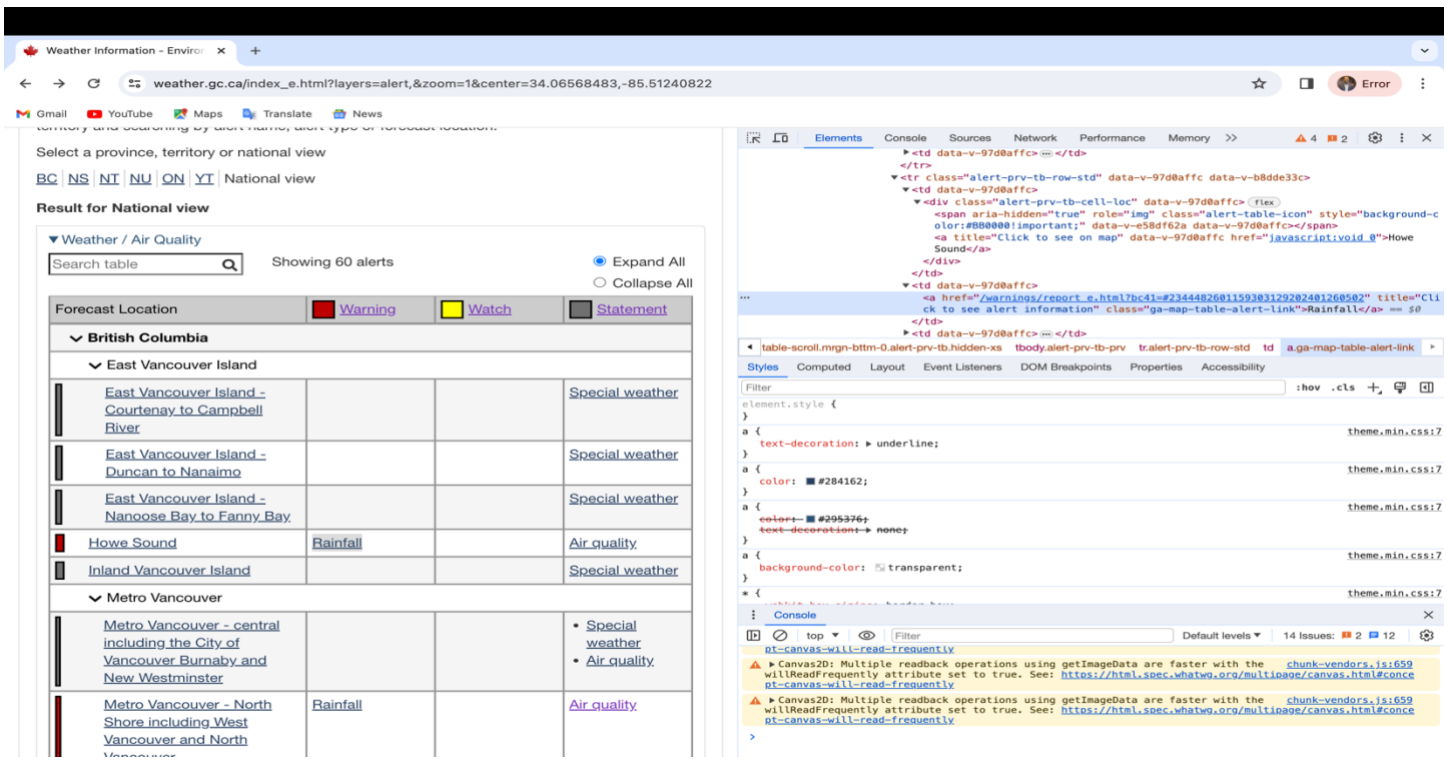
Appendix 2- Identifying the web element for Province.



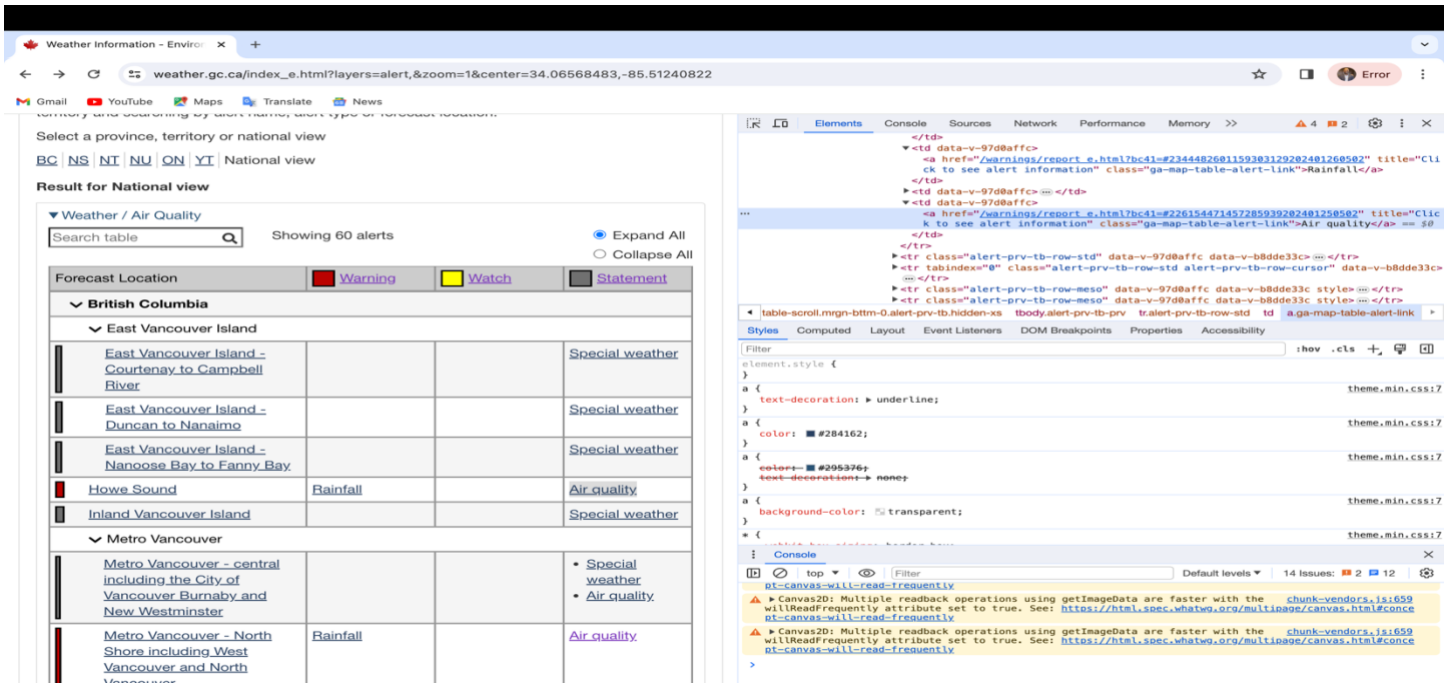
Appendix 3- Identifying the web element for City.



Appendix 4- Identifying the web element for Warning.



Appendix 5- Identifying the web element for Statement.



Appendix 6- Weather hazards that triggers Warning.

Appendix 7- Weather data (CSV) before adding Month, Day, Date, and Time to enhance data quality.

weather_data

Province	City	Warning	Statement
British Columbia	East Vancouver Island - Nanoose Bay to Fanny Bay		Special weather
Northwest Territories	Howe Sound	Rainfall	
Nova Scotia	Inland Vancouver Island		Air quality
Nunavut	Metro Vancouver - central including the City of Vancouver Burnaby and New Westminster		Special weather
Ontario	Metro Vancouver - North Shore including West Vancouver and North Vancouver		Special weather
Yukon	Metro Vancouver - northeast including Coquitlam and Maple Ridge		Air quality
British Columbia	North Vancouver Island	Rainfall	
Northwest Territories	Sunshine Coast - Gibsons to Earls Cove		Air quality
Nova Scotia	Sunshine Coast - Saltery Bay to Powell River	Rainfall	
Nunavut	West Vancouver Island		Air quality
Ontario	Aklavik Region		Special weather
Yukon	South Delta Region including Ft. McPherson - Tsiigehtchic		Special weather
British Columbia	Annapolis County		Special weather
Yukon	Antigonish County		Special weather
British Columbia	Colchester County - Truro and south	Extreme cold	
Yukon	Digby County	Extreme cold	

Appendix 8- Weather data (CSV) after adding Month, Day, Date, and Time to enhance data quality.

weather_data

Province	City	Warning	Statement	Month	Day	Date	Time
British Columbia	East Vancouver Island - Nanoose Bay to Fanny Bay		Special weather	January	Saturday	2024-01-27	19:52:44
Northwest Territories	Howe Sound	Rainfall		January	Saturday	2024-01-27	19:52:44
Nova Scotia	Inland Vancouver Island		Air quality	January	Saturday	2024-01-27	19:52:44
Nunavut	Metro Vancouver - central including the City of Vancouver Burnaby and New Westminster		Special weather	January	Saturday	2024-01-27	19:52:44
Ontario	Metro Vancouver - North Shore including West Vancouver and North Vancouver		Special weather	January	Saturday	2024-01-27	19:52:44
Yukon	Metro Vancouver - northeast including Coquitlam and Maple Ridge		Air quality	January	Saturday	2024-01-27	19:52:44
British Columbia	North Vancouver Island	Rainfall		January	Saturday	2024-01-27	19:52:44
Northwest Territories	Sunshine Coast - Gibsons to Earls Cove		Air quality	January	Saturday	2024-01-27	19:52:44
Nova Scotia	Sunshine Coast - Saltery Bay to Powell River	Rainfall		January	Saturday	2024-01-27	19:52:44
Nunavut	West Vancouver Island		Air quality	January	Saturday	2024-01-27	19:52:44
Ontario	Aklavik Region		Special weather	January	Saturday	2024-01-27	19:52:44
Yukon	South Delta Region including Ft. McPherson - Tsiigehtchic		Special weather	January	Saturday	2024-01-27	19:52:44
British Columbia	Annapolis County		Special weather	January	Saturday	2024-01-27	19:52:44
Yukon	Antigonish County		Special weather	January	Saturday	2024-01-27	19:52:44
British Columbia	Colchester County - Truro and south	Extreme cold		January	Saturday	2024-01-27	19:52:44
Yukon	Digby County	Extreme cold		January	Saturday	2024-01-27	19:52:44

Appendix 9- Printed data frame of the scrapped data

	Province	City	\
0	British Columbia	East Vancouver Island – Nanoose Bay to Fanny Bay	
1	Northwest Territories	Howe Sound	
2	Nova Scotia	Inland Vancouver Island	
3	Nunavut	Metro Vancouver – central including the City o...	
4	Ontario	Metro Vancouver – North Shore including West V...	
5	Yukon	Metro Vancouver – northeast including Coquitla...	
6	British Columbia	North Vancouver Island	
7	Northwest Territories	Sunshine Coast – Gibsons to Earls Cove	
8	Nova Scotia	Sunshine Coast – Saltery Bay to Powell River	
9	Nunavut	West Vancouver Island	
10	Ontario	Aklavik Region	
11	Yukon	South Delta Region including Ft. McPherson – T...	
12	British Columbia	Annapolis County	
13	Yukon	Antigonish County	
14	British Columbia	Colchester County – Truro and south	
15	Yukon	Digby County	

	Warning	Statement	Month	Day	Date	Time
0	None	Special weather	January	Saturday	2024-01-27	20:05:43
1	Rainfall	None	January	Saturday	2024-01-27	20:05:43
2	None	Air quality	January	Saturday	2024-01-27	20:05:43
3	None	Special weather	January	Saturday	2024-01-27	20:05:43
4	None	Special weather	January	Saturday	2024-01-27	20:05:43
5	None	Air quality	January	Saturday	2024-01-27	20:05:43
...						
12	None	Special weather	January	Saturday	2024-01-27	20:05:43
13	None	Special weather	January	Saturday	2024-01-27	20:05:43
14	Extreme cold	None	January	Saturday	2024-01-27	20:05:43
15	Extreme cold	None	January	Saturday	2024-01-27	20:05:43