AI INTERNSHIP REPORT

Task 1 – Web Scraping

Submitted by: Rija Mobin

Submitted to: Miss Samia

Submission Date: July 24, 2025

Task Requirements:

1- Scrape links using BeautifulSoup and selenium

2- Store all data in CSV file

3- Get all current and previous articles

Objective

Objectives based on my code functionality are:

- Automatically scrapes all article links from `https://www.rds.ca`
- Extracts structured data (title, date, description, URL) from each article
- Uses **Selenium + BeautifulSoup + Requests**
- Outputs everything into a structured CSV file

---

1. **Understand the Structure of Web Pages and Dynamic Content**
   Websites like `rds.ca` contain many dynamically loaded articles structured with tags like `<a>`, `<h1>`, `<time>`, and `<p>`. This project focused on understanding how such dynamic web pages are built and how to interact with them using tools like Selenium and BeautifulSoup.
2. **Automate the Extraction of All Article Data**
   Instead of manually copying information, the tool automatically:
   - Opens the website and scrolls down to load more content.
   - Collects all article links.
   - Visits each article to extract its **title**, **date**, **short description**, and **URL**.

3. **Save Extracted Data in a Structured CSV Format**
   The scraper stores the extracted data in a `.csv` file with separate columns. This makes the dataset easy to analyze in Excel, Python, or any other data analysis tool.
4. **Use and Combine Key Python Libraries**
   The tool demonstrates the combined use of:
   - `Selenium` to interact with dynamic pages and scroll content
   - `BeautifulSoup` for parsing HTML
   - `Requests` for fast article loading
   - `CSV` module for structured data storage
5. **Build an Efficient and Scalable Scraping Tool**
   The scraper is optimized to handle many articles quickly and efficiently by combining the power of Selenium (for page loading) and Requests (for speed). It can be easily extended to scrape other news websites in a similar structure.

Summary:
This project creates a Python-based tool that automatically loads, collects, and extracts article data from a real website and saves it into a structured CSV file. The tool is designed to be efficient, scalable, and reusable for similar data extraction tasks.

Would you like this version added directly to your PDF report as well?

Project Execution Steps

- Check Python Installation

python --version

- Create Virtual Environment

python -m venv venv

- Allow Script Execution in PowerShell

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

- Activate the Environment

.\venv\Scripts\Activate.ps1

- Install Required Libraries

pip install beautifulsoup4 requests

- Create and Run the Python Script

python Scraper_test.py

- Convert Script to Executable File

pip install selenium beautifulsoup4 pandas

pip install pyinstaller
pyinstaller --onefile Scraper_test.py

Code Explanation

---

from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service

These lines import Selenium tools used to control the Chrome browser. `webdriver` is used to launch the browser, `Options` lets us customize it (like headless mode), and `Service` helps manage the Chrome driver.

---

from bs4 import BeautifulSoup
import requests
import csv
import time

These libraries help us parse HTML (`BeautifulSoup`), make fast web requests (`requests`), save data in CSV format (`csv`), and pause the program for loading delays (`time`).

---

chrome_options = Options()
chrome_options.add_argument("--headless")
chrome_options.add_argument("--disable-gpu")

We create browser settings. `--headless` makes the browser run in the background without opening a window. `--disable-gpu` disables the graphics card (saves resources).

```
service = Service()
driver = webdriver.Chrome(service=service, options=chrome_options)
```

Starts the Chrome browser with the defined settings. The `driver` object will be used to interact with the website.

```
url = "https://www.rds.ca"
driver.get(url)
time.sleep(3)
```

Opens the RDS homepage and waits 3 seconds to allow it to fully load.

```
for _ in range(10):
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(2)
```

Scrolls the page down 10 times. This helps load more articles (like infinite scroll). After each scroll, the program waits 2 seconds for content to load.

```
soup = BeautifulSoup(driver.page_source, 'html.parser')
driver.quit()
```

Grabs the final HTML content after scrolling and closes the browser. `soup` holds the parsed HTML to be searched.

```
base_url = "https://www.rds.ca"
article_links = set()
```

Defines the main site link and creates an empty set to store unique article URLs.

```
for a in soup.find_all('a', href=True):
    href = a['href']
    if any(keyword in href for keyword in ['article', 'articles', 'hockey', 'boxe',
'canadiens','videos']):
        if href.startswith('/'):
            article_links.add(base_url + href)
        elif href.startswith('http'):
            article_links.add(href)
```

Finds all `<a>` links. Keeps only those that look like article links. If the link is relative (starts with `/`), we add the base site. Full URLs are added directly.

```
csv_filename = "rds_all_articles.csv"
with open(csv_filename, mode='w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerow(['Title', 'Date', 'Description', 'URL'])
```

Creates and opens a CSV file to save the extracted article data. The first row is the column headers.

```
print(f"\n⏳ Scraping {len(article_links)} articles (no limit)...\n")
```

Shows how many articles will be scraped.

```
for i, link in enumerate(sorted(article_links), start=1):
    try:
        response = requests.get(link, timeout=10)
        article_soup = BeautifulSoup(response.text, 'html.parser')
```

Loops through each article link. Uses `requests` to fetch the page quickly, then parses it with BeautifulSoup.

```
        title_tag = article_soup.find(['h1', 'h2'])
        title = title_tag.get_text(strip=True) if title_tag else "No Title"
```

Finds the title of the article in either an `<h1>` or `<h2>` tag. If not found, sets "No Title".

```
date_tag = article_soup.find('time')
date = date_tag.get_text(strip=True) if date_tag else "No Date"
```

Looks for the date tag (`<time>`). If found, extracts the text; otherwise, sets "No Date".

```
desc_tag = article_soup.find('p')
description = desc_tag.get_text(strip=True) if desc_tag else "No Description"
```

Gets the first paragraph of the article as a short description.

```
writer.writerow([title, date, description, link])
print(f"☑ {i}. {title[:60]}...")
```

Saves the extracted data to the CSV and shows a short success message for each article.

```
except Exception as e:
print(f"✗ Error scraping {link}: {str(e)}")
```

If something goes wrong (timeout, page error), the program prints the error and continues.

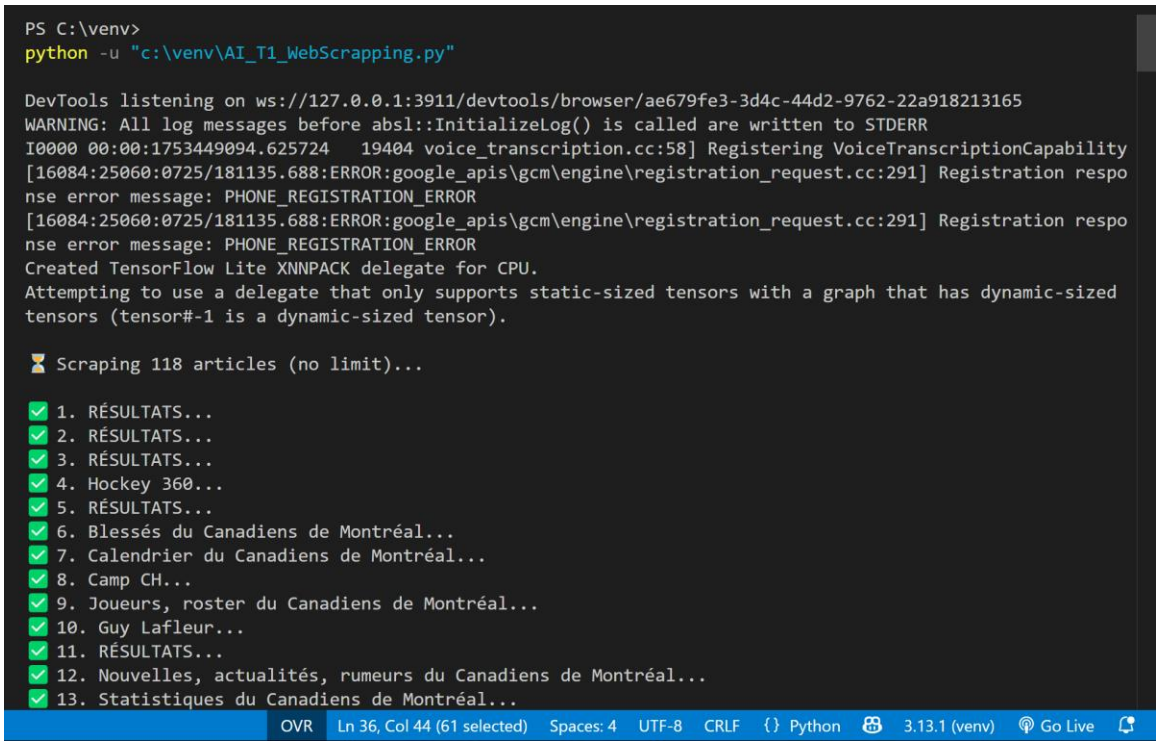
```
print(f"\n☑ All article data saved to '{csv_filename}'")
```

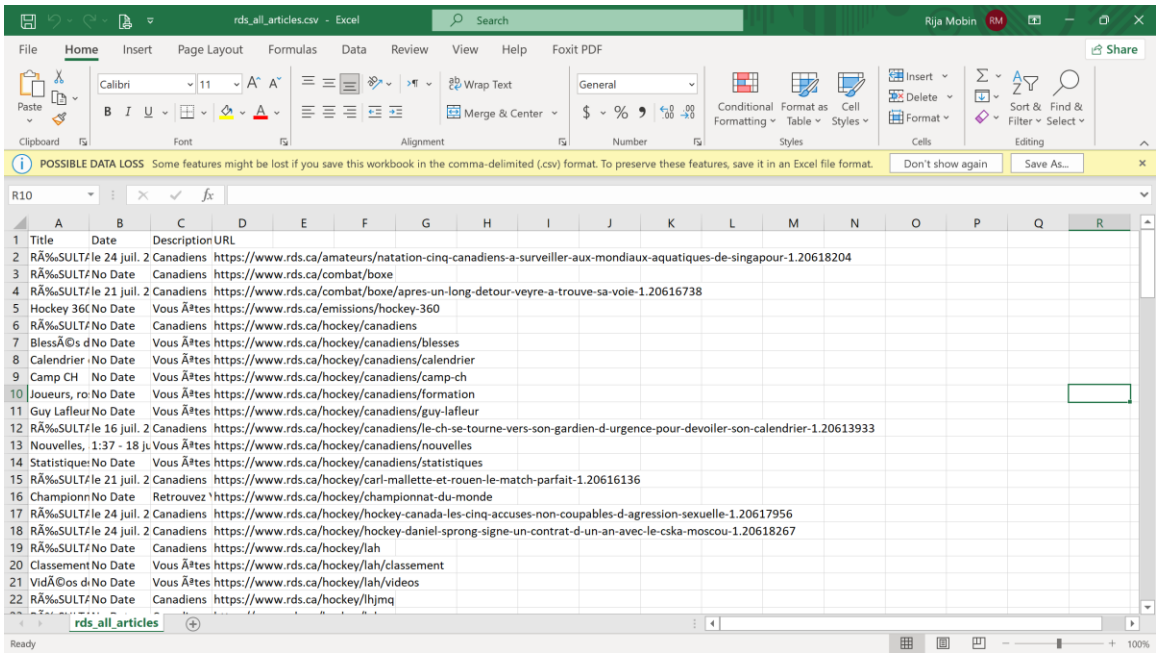
After all articles are processed, a final message confirms the data was saved successfully.

Details About the CSV File

Name: `rds_all_articles.csv`
Contents: Four columns – **Title**, **Date**, **Description**, and **URL**
Use: This file contains all the scraped articles from the RDS website. Each row represents one article. It includes the article's title, publication date, a short description (first paragraph), and the direct link to the article.
It can be opened in Excel or any spreadsheet software for further reading, sorting, or data analysis.

Output:





Learning Outcomes

- Learnt how websites use dynamic loading and how to handle it using **Selenium**
- Practiced using **BeautifulSoup** to parse and extract data from complex HTML structures
- Understood how to use the **Requests** library for fast article scraping
- Learned how to automate the collection of multiple articles from a live website
- Gained experience in writing clean, structured data to a **CSV** file for analysis
- Applied Python scripting to a real-world use case involving live news article extraction
- Improved script performance by combining the speed of Requests with Selenium's flexibility

Conclusion

This project helped me build a powerful and efficient web scraping tool that collects article data from a real news website (`rds.ca`). It extracts all available article links from the homepage, then automatically scrapes the **title**, **publication date**, **short description**, and **URL** from each article. The data is saved into a CSV file in a structured format, making it easy to view or analyze later.

The project gave me hands-on experience with real-world tools like **Selenium**, **Requests**, and **BeautifulSoup**, and taught me how to optimize web scrapers for both **speed** and **scalability**. These are essential skills for data science, automation, and AI development tasks involving web data collection.