Data and Artificial Intelligence
Cyber Shujaa Program

Week 1 Assignment
Web Scraping and Data Handling in Python

Student Name: John Brown Ouma

Student ID: CS-DA01-25030.

Introduction

This report documents the completion of a web scraping assignment that involved extracting structured data from the "Forms and Search" page on scrapethissite.com. The goal was to gather hockey team statistics, organize them into a structured format using pandas, and export the data to a CSV file for further analysis. The entire scraping process was implemented using Python on Google Colab, utilizing key libraries including requests for making HTTP requests, BeautifulSoup for parsing HTML, and pandas for data management and export.

The objectives of the assignment were:

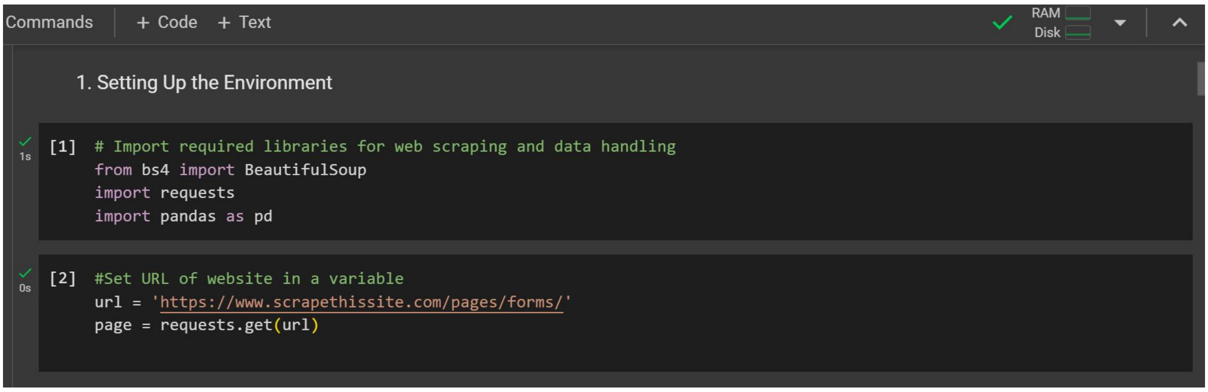
1. Practical Python coding on Jupiter Notebooks hosted on Google Colab
2. Use requests and BeautifulSoup to extract data from a web page.
3. Parse and clean the extracted data.
4. Store structured data into a Pandas DataFrame.
5. Export the final dataset to a .csv file.

Tasks Completed

1. Setting Up the Environment

Before web scraping, it's essential to import the libraries required for the task and establish a connection to the target webpage. In this step:

- **requests** fetches the webpage's HTML content.
- **BeautifulSoup** is responsible for parsing the HTML so it can be navigated and queried.
- **pandas** will later help structure the scraped data into a tabular format.



1. Setting Up the Environment

```
[1] # Import required libraries for web scraping and data handling
    from bs4 import BeautifulSoup
    import requests
    import pandas as pd
```

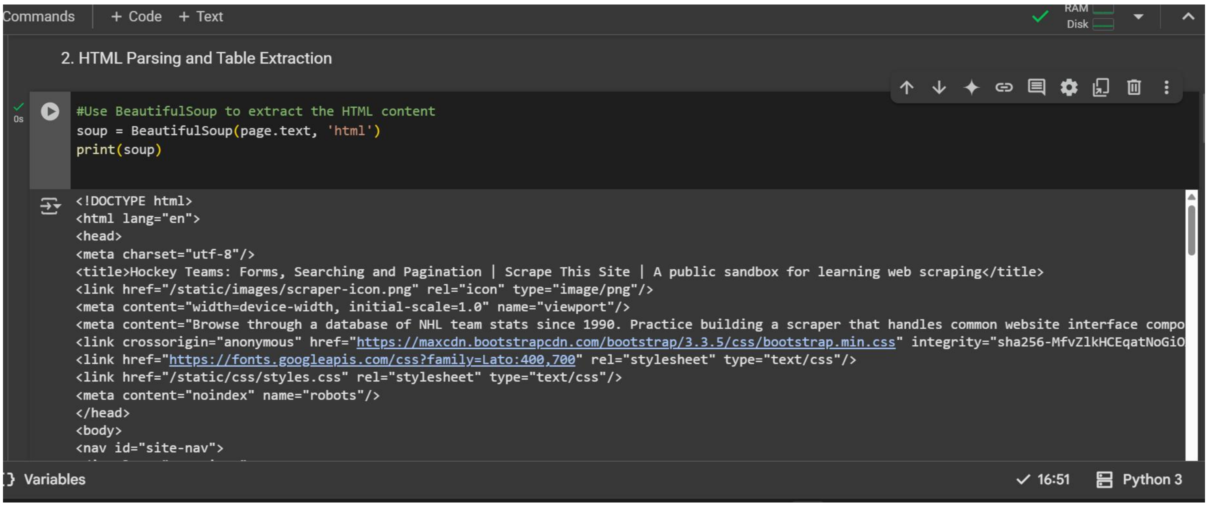
```
[2] #Set URL of website in a variable
    url = 'https://www.scrapethissite.com/pages/forms/'
    page = requests.get(url)
```

Figure 1: Importing libraries and setting up the target URL

This ensures the environment has all tools needed for making HTTP requests, parsing HTML, and handling data.

2. HTML Parsing and Table Extraction

Once the page's content is fetched, we parse it into a **BeautifulSoup object**, allowing us to interact with the page's structure and locate elements of interest — in this case, the **hockey statistics table**.

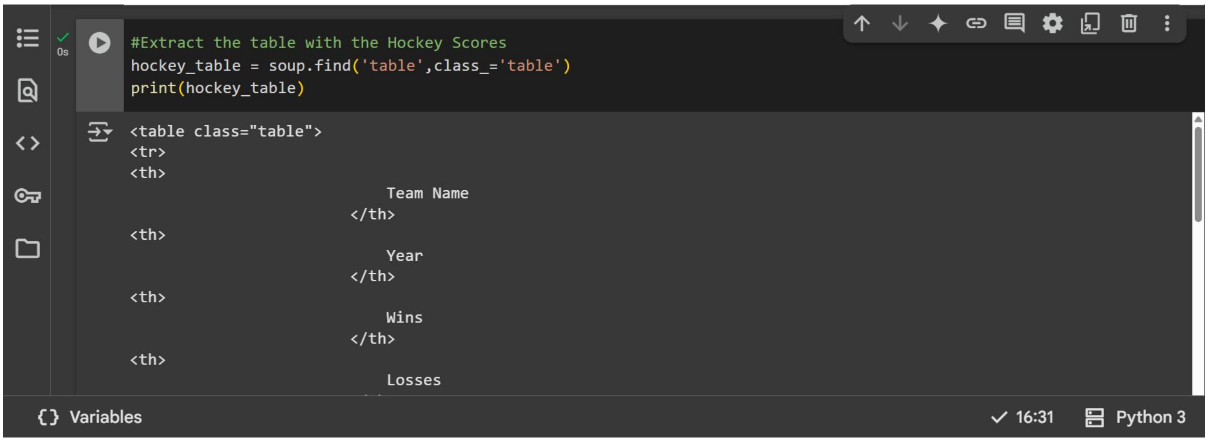


2. HTML Parsing and Table Extraction

```
#Use BeautifulSoup to extract the HTML content
soup = BeautifulSoup(page.text, 'html')
print(soup)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8"/>
<title>Hockey Teams: Forms, Searching and Pagination | Scrape This Site | A public sandbox for learning web scraping</title>
<link href="/static/images/scraper-icon.png" rel="icon" type="image/png"/>
<meta content="width=device-width, initial-scale=1.0" name="viewport"/>
<meta content="Browse through a database of NHL team stats since 1990. Practice building a scraper that handles common website interface compo
<link crossorigin="anonymous" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" integrity="sha256-MfvZlkHCEqatNoGiO
<link href="https://fonts.googleapis.com/css?family=Lato:400,700" rel="stylesheet" type="text/css"/>
<link href="/static/css/styles.css" rel="stylesheet" type="text/css"/>
<meta content="noindex" name="robots"/>
</head>
<body>
<nav id="site-nav">
```

Figure 2: Parsing HTML



```
#Extract the table with the Hockey Scores
hockey_table = soup.find('table',class_='table')
print(hockey_table)
```

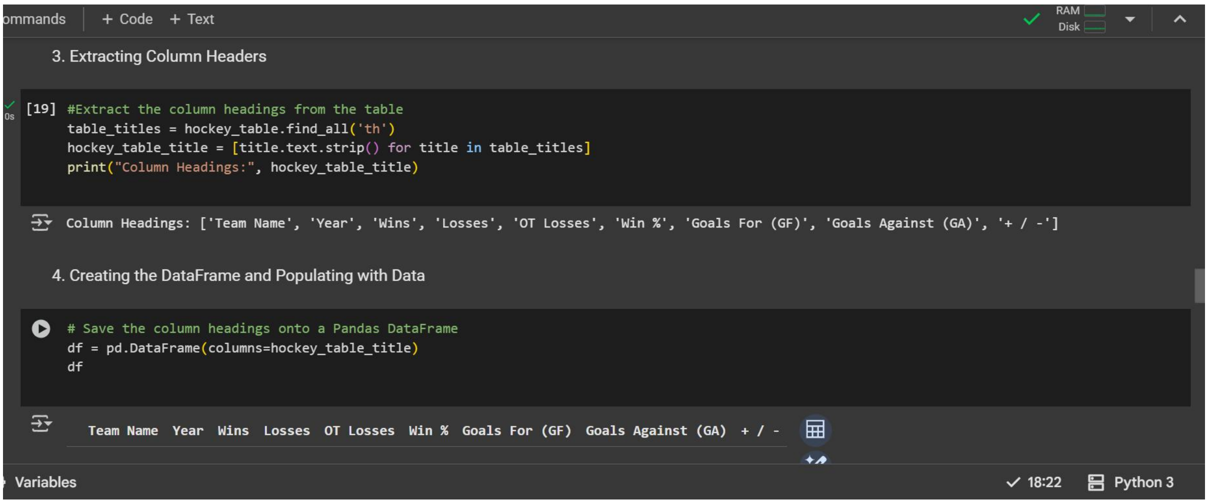
```
<table class="table">
<tr>
<th>
                            Team Name
                        </th>
<th>
                            Year
                        </th>
<th>
                            Wins
                        </th>
<th>
                            Losses
```

Figure 2.1 finding the target table

This transforms the raw HTML string into a navigable object and finds the specific table containing the data by searching for the table element with class table.

3. Extracting Column Headers

Before collecting data, it's crucial to identify the table's **column titles** to accurately structure our data. This is done by locating all <th> tags within the table.



3. Extracting Column Headers

```
[19] #Extract the column headings from the table
     table_titles = hockey_table.find_all('th')
     hockey_table_title = [title.text.strip() for title in table_titles]
     print("Column Headings:", hockey_table_title)
```

```
Column Headings: ['Team Name', 'Year', 'Wins', 'Losses', 'OT Losses', 'Win %', 'Goals For (GF)', 'Goals Against (GA)', '+ / -']
```

4. Creating the DataFrame and Populating with Data

```
# Save the column headings onto a Pandas DataFrame
df = pd.DataFrame(columns=hockey_table_title)
df
```

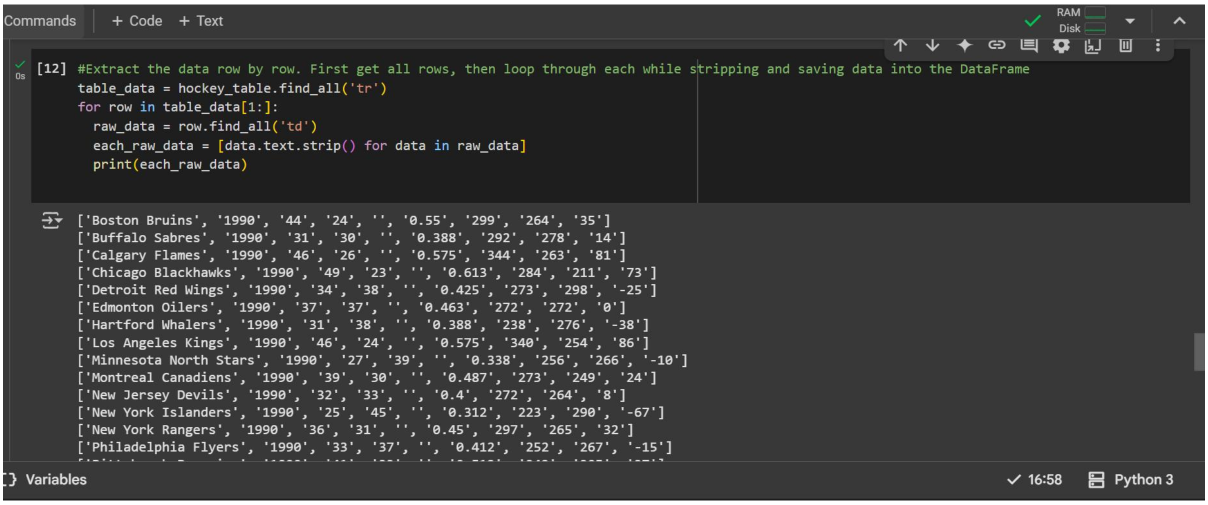
```
  Team Name  Year  Wins  Losses  OT Losses  Win %  Goals For (GF)  Goals Against (GA)  + / -
```

Figure 3: Extracting and displaying table column headers

This retrieves clean, human-readable column names which will act as the DataFrame's headers, ensuring our dataset is well-organized.

4. Creating the DataFrame and Populating with Data

With headers identified, the next step is to **initialize an empty DataFrame** using those headers and then **loop through each row in the table body** (skipping the header row) to extract
data.



```
[12] #Extract the data row by row. First get all rows, then loop through each while stripping and saving data into the DataFrame
     table_data = hockey_table.find_all('tr')
     for row in table_data[1:]:
       raw_data = row.find_all('td')
       each_raw_data = [data.text.strip() for data in raw_data]
       print(each_raw_data)
```

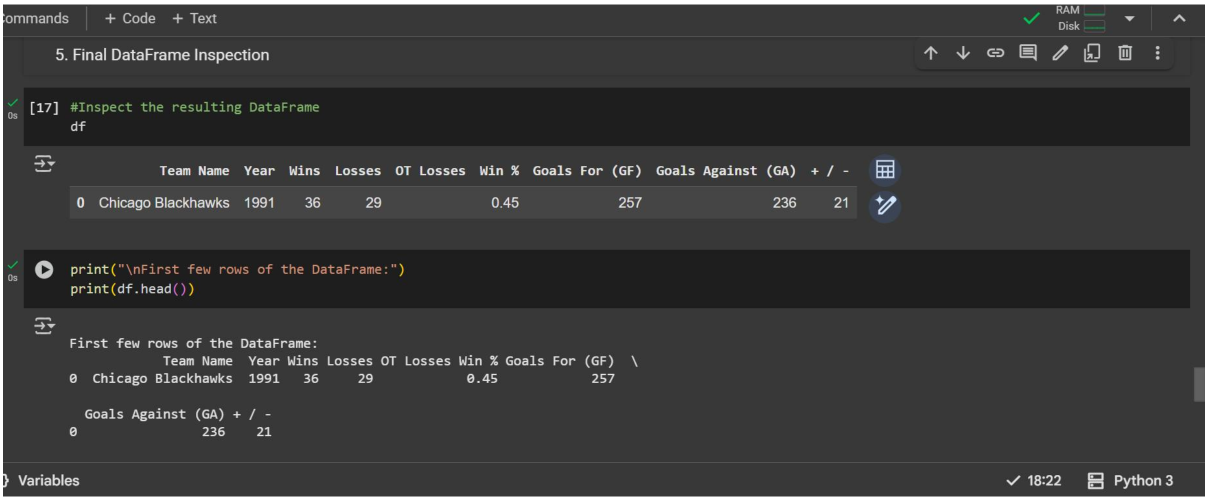
```
['Boston Bruins', '1990', '44', '24', '', '0.55', '299', '264', '35']
['Buffalo Sabres', '1990', '31', '30', '', '0.388', '292', '278', '14']
['Calgary Flames', '1990', '46', '26', '', '0.575', '344', '263', '81']
['Chicago Blackhawks', '1990', '49', '23', '', '0.613', '284', '211', '73']
['Detroit Red Wings', '1990', '34', '38', '', '0.425', '273', '298', '-25']
['Edmonton Oilers', '1990', '37', '37', '', '0.463', '272', '272', '0']
['Hartford Whalers', '1990', '31', '38', '', '0.388', '238', '276', '-38']
['Los Angeles Kings', '1990', '46', '24', '', '0.575', '340', '254', '86']
['Minnesota North Stars', '1990', '27', '39', '', '0.338', '256', '266', '-10']
['Montreal Canadiens', '1990', '39', '30', '', '0.487', '273', '249', '24']
['New Jersey Devils', '1990', '32', '33', '', '0.4', '272', '264', '8']
['New York Islanders', '1990', '25', '45', '', '0.312', '223', '290', '-67']
['New York Rangers', '1990', '36', '31', '', '0.45', '297', '265', '32']
['Philadelphia Flyers', '1990', '33', '37', '', '0.412', '252', '267', '-15']
```

Figure 4: Populating the DataFrame with scraped data

This systematically extracts each cell's content, cleans it (removes leading/trailing whitespace), and appends it to the DataFrame — one row at a time.

5. Final DataFrame Inspection

After populating the DataFrame, it's important to **review the resulting table** to ensure data integrity and verify the scraping process was successful.



5. Final DataFrame Inspection

```
[17] #Inspect the resulting DataFrame
df
```

	Team Name	Year	Wins	Losses	OT Losses	Win %	Goals For (GF)	Goals Against (GA)	+ / -
0	Chicago Blackhawks	1991	36	29		0.45	257	236	21

```
print("\nFirst few rows of the DataFrame:")
print(df.head())
```

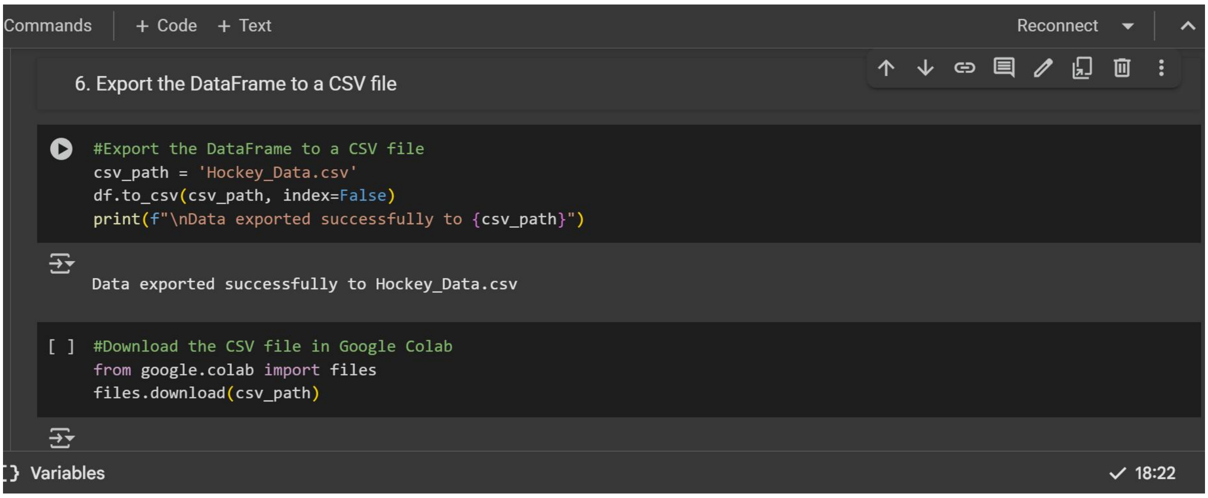
```
First few rows of the DataFrame:
          Team Name  Year Wins Losses OT Losses Win % Goals For (GF)  \
0  Chicago Blackhawks  1991   36     29              0.45             257

  Goals Against (GA) + / -
0                 236    21
```

Figure 5:DataFrame Inspection

Quickly displays the entire structured dataset within the Colab notebook, providing an opportunity to visually inspect for missing values or irregularities.

6. Exporting to CSV

Finally, we export the clean, structured dataset to a **CSV file** for future use or sharing.



6. Export the DataFrame to a CSV file

```
#Export the DataFrame to a CSV file
csv_path = 'Hockey_Data.csv'
df.to_csv(csv_path, index=False)
print(f"\nData exported successfully to {csv_path}")
```

```
Data exported successfully to Hockey_Data.csv
```

```
[ ] #Download the CSV file in Google Colab
    from google.colab import files
    files.download(csv_path)
```

Figure 6: Exporting to CSV and downloading the CSV

CSV is a versatile, widely supported file format suitable for data analysis, visualization, or integration into other tools or projects.

Link to Code:
https://colab.research.google.com/drive/1agOHXQMxoHCiqEyRxSHWb9IvQHRjpUWJ?usp=sharing

Conclusion

This assignment successfully demonstrated the process of web scraping using Python. I was able to:

1. Retrieve web page content using the Requests library
2. Parse and extract specific data using BeautifulSoup
3. Structure and clean the data using pandas
4. Export the final dataset to a CSV file

The complete Google Colab notebook can be accessed here: https://colab.research.google.com/drive/1agOHXQMxoHCiqEyRxSHWb9IvQHRjpUWJ?usp=sharing

Through this exercise, I gained practical experience in web scraping techniques and data organization, which are valuable skills for data collection and analysis tasks.