“WEB SCRAPING USING R”

STAT 260 : Final Project

Samaira (301592227)

Introduction

Web scraping is the process of extracting data from websites and converting it into a format suitable for analysis or storage. In today's data-driven world, much of the information we need is available on websites. However, it is not always available in a downloadable or structured format such as CSV or Excel files. That's where web scraping becomes essential. This is a tutorial on how to do web scraping in R.

What is Web Scraping?

Web scraping is an automated technique used to extract data from web pages. Websites are built using HTML (HyperText Markup Language), and within this structure, various types of information are displayed, such as text, images, and links. Web scraping allows you to programmatically access that content and save it in a more usable format.

For instance, imagine you want to collect the prices and names of books from an online bookstore for comparison. While doing this manually might take hours, a simple web scraper can collect all the data in seconds. This makes web scraping an incredibly powerful tool for researchers, analysts, marketers, and developers alike.

Common uses of web scraping include:

- Price comparison between products across e-commerce sites
- Real-time news aggregation
- Market research and trend tracking
- Data collection for academic research
- Monitoring social media or job postings

It is important to note that ethical considerations must be taken into account. Always check a website's terms of service and `robots.txt` file to ensure your scraping is compliant.

Why Use R for Web Scraping?

R is an excellent choice for web scraping for several reasons:

- **Integrated Tools**: R has built-in libraries like `rvest`, `xml2`, and `httr` that make it easy to scrape and manipulate web content.
- **Data Handling**: R is a statistical programming language, so once you've scraped the data, you can immediately begin cleaning, visualizing, and analyzing it using familiar packages like `dplyr`, `ggplot2`, or `tidyr`.

- **Learning Curve**: R is beginner-friendly, and many tutorials and resources exist online to help you learn how to scrape data.

What Do These Packages Do?

- `rvest`: A package designed for web scraping. It helps you easily read and extract parts of HTML using CSS selectors.
- `tidyverse`: A collection of R packages like `dplyr`, `ggplot2`, and `readr` used for data manipulation, visualization, and importing/exporting data. It also includes the `%>%` pipe operator which passes the result from one function into the next.

Getting Started with Web Scraping in R

```
# install.packages("rvest")
# install.packages("tidyverse")
library(rvest)      # rvest is used for web scraping
library(tidyverse)  # tidyverse is a collection of packages for data manipulation and visualization
```

Understanding HTML Structure

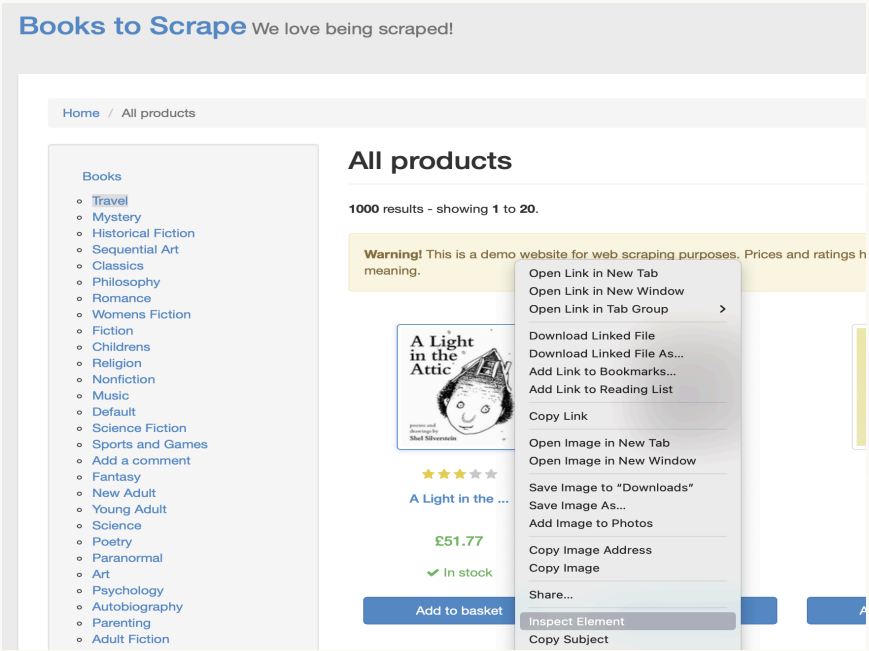
To scrape information from a webpage, you must first understand how that information is structured in the HTML code. HTML consists of elements enclosed in tags such as `<div>`, `<p>`, `<h1>`, and `<a>`. These tags may include attributes like `class` or `id` which can help you target specific elements.

To inspect a webpage:

1. Right-click on the element of interest.

2. Choose **Inspect** (or "Inspect Element" depending on your browser).

3. A developer panel will open showing the HTML structure of the page.



When scraping a webpage, it's crucial to understand how the information is structured in the HTML. This helps us figure out exactly what to target in our code using CSS selectors.

Here's a simplified example of the HTML structure from the **Books to Scrape** website:

```
<article class="product_pod">
  <h3>
    <a href="booklink.html" title="A Light in the Attic">A Light in the Attic</a>
  </h3>
  <div class="product_price">
    <p class="price_color">£51.77</p>
    <p class="instock availability">
      In stock
    </p>
  </div>
</article>
```

Knowing this structure lets us write accurate and efficient scraping code.By inspecting the HTML and understanding the tag and class structure, we know exactly what to extract and how. This is a key skill in web scraping!

Choosing a Website to Scrape

For this project, we'll use a beginner-friendly website specifically built for scraping practice:**https://books.toscrape.com** . It displays book titles and prices in a consistent format, which makes it ideal for our tutorial.

Step-by-Step Guide

Step 1: Read the Web Page

```
url <- "https://books.toscrape.com/"
page <- read_html(url)
```

Explanation:

- `url` is a character string storing the web address of the page to scrape.

- `read_html(url)` fetches and parses the HTML content of the page.
- The result is stored in `page`, which holds the HTML structure we'll work with.

Step 2: Identify HTML Elements

Use your browser's **Inspect** tool to locate the HTML tags that contain the book titles and prices. For example:

- Titles are inside `<h3><a>` tags.
- Prices are in `<p class='price_color'>`.

Step 3: Extract Book Titles and Prices

```
titles <- page %>% html_nodes("h3 a") %>% html_attr("title")
prices <- page %>% html_nodes("p.price_color") %>% html_text()
books_df <- data.frame(Titles = titles, Prices = prices)
head(books_df)
```

Explanation:

- `%>%`: The pipe passes the result from one function to the next.
- `html_nodes("h3 a")`: Selects all anchor tags inside h3 headers.
- `html_attr("title")`: Extracts the value of the `title` attribute (book title).
- `html_nodes("p.price_color")`: Selects price tags.
- `html_text()`: Gets the visible price text.
- `data.frame()`: Creates a table of titles and prices.

Step 4: Clean the Price Data

```
books_df$Prices <- as.numeric(gsub("[^0-9.]", "", books_df$Prices))
```

Explanation:

- `gsub()` removes currency symbols and unwanted characters.
- `as.numeric()` converts text to numeric values.

Step 5: Check for Missing Data

```
sum(is.na(books_df))
books_df <- na.omit(books_df)
```

Explanation:

- `is.na()` identifies missing values.
- `sum()` counts how many.
- `na.omit()` removes rows with missing data.

Step 6: Save to CSV

```
write.csv(books_df, "books_data.csv", row.names =FALSE)
```

Explanation:

- This writes the cleaned data to a file you can open in Excel or Google Sheets.

Scraping Multiple Pages

```
all_books <- data.frame()

for (i in 1:5) {
  url <- paste0("https://books.toscrape.com/catalogue/page-", i, ".html")
  page <- read_html(url)

  titles <- page %>% html_nodes("h3 a") %>% html_attr("title")
  prices <- page %>% html_nodes("p.price_color") %>% html_text()

  temp_df <- data.frame(Titles = titles, Prices = prices)
  all_books <- rbind(all_books, temp_df)
}
head(all_books)
```

Explanation:

- The loop changes the page number in the URL (1 to 5).
- Each page is scraped and added to `all_books`.

Extracting Data to a CSV File

Once we've scraped all the necessary information — such as book titles, and prices — the next step is to export the data into a structured file format for analysis or sharing. One of the most common formats used is the CSV (Comma-Separated Values) file.

Saving the data frame:

```
write.csv(books_df, "books_data.csv", row.names = FALSE)
```

Explanation:

- `write.csv(...)`: This function writes your data frame to a `.csv` file.
- `"books_data.csv"`: This is the name of the file that will be created in your R working directory.
- `row.names = FALSE`: Prevents row numbers from being added as an extra column in the CSV file.

This command will generate a file named `books_data.csv` in your project folder, containing the dataset you created. You can open this file in Excel or any spreadsheet tool to verify the structure and contents.

Conclusion

In this report, we explored the fundamentals of web scraping in R using the `rvest` package. We began by understanding the structure of HTML documents and how to use developer tools to identify the specific elements that contain the data we want to extract. Using the "Books to Scrape" website as an example, we demonstrated how to extract book titles and prices by targeting HTML elements with specific CSS classes.

Overall, this tutorial serves as a foundational guide for beginners who want to learn web scraping in R. The techniques introduced here can be extended to other websites and datasets, making web scraping a valuable tool in the data science workflow.

References

1. Wickham, H. (2022). *rvest: Easily Harvest (Scrape) Web Pages*. Retrieved from https://rvest.tidyverse.org
2. Grolemund, G., & Wickham, H. (2017). *R for Data Science*. O'Reilly Media. Available at https://r4ds.had.co.nz
3. W3Schools. (n.d.). *HTML Tutorial*. Retrieved from https://www.w3schools.com/html
4. DataCamp. (n.d.). *Web Scraping in R with rvest*. Retrieved from https://www.datacamp.com