Web Scraping Reviews From The Uber Eats Page on Trustpilot

Anju Sambasivan

Introduction

In this project, we will learn how to collect data from websites using web scraping. Specifically, we will scrape reviews from the Uber Eats page on Trustpilot. We will extract important information such as the review text, reviewer name, date, star rating, and review title. Using the **rvest** package in R, we will first scrape data from one page, and then automate the process to collect data from multiple pages. This project will help us understand web scraping and how to work with website data.

```
library(rvest)
# Loads tools to scrape data from websites.

library(lubridate)
# Provides functions to easily handle and format dates.

library(dplyr)
# Helps in cleaning, sorting, and manipulating data efficiently.
```

I use three key libraries to make web scraping and data processing easier. The rvest package helps extract data from websites, such as review text and ratings. The lubridate package simplifies working with dates, converting them into a standard format. The dplyr package makes it easy to clean, sort, and organize the data. These libraries work together to collect and prepare the data for analysis.

```
extract_dates <- function(review_date) {

# Removes(Substitute) the text "Date of experience: " from the input string.
  dates <- sub("Date of experience: ", "", review_date)
```

```
# Converts the remaining date string into a date format (month-day-year).
  dates <- mdy(dates)

# Returns the cleaned date in the correct format.
  return(dates)
}

# sub() searches for a specific pattern in a string
# and replaces the first occurrence of that pattern with a new value.
```

The `extract_dates` function is used to clean and format the dates extracted from reviews. It removes the unnecessary text "Date of experience:" using the `sub` function, leaving only the actual date. The cleaned date is then converted into a standard Month-Day-Year format using the `mdy` function from the `lubridate` package. This ensures the dates are consistent and easy to use for further analysis.

```
extract_rating <- function(rating_text) {
# Finds the rating in the format "Rated X out of 5 stars" and
# extracts the number X.

  rating <- sub("Rated (\\d) out of 5 stars", "\\1", rating_text)
# The part (\\d) captures the digit X
# \\1 tells sub() to replace the entire match with the content
# of the first captured group.
# i.e., the first part of the pattern inside parentheses.

# Converts the extracted rating (which is a string) into an integer.
  return(as.integer(rating))
}
```

The `extract_rating` function is used to extract star ratings from review text. It looks for text in the format "Rated X out of 5 stars" and extracts the number `X` using the `sub` function. The extracted number, which is initially a string, is then converted into an integer for easy use in analysis. This function ensures that the star ratings are clean and in a consistent format for further processing.

```
scrape_page <- function(page_num) {
# Function to get data from a specific page.
  url <- paste0("https://www.trustpilot.com/review/ubereats.com?page=", page_num)
# Combine base URL with the page number.
  page <- read_html(url)
```

```r
# Read the webpage's content.



  review_texts <- page %>%
# Select the HTML page content.
    html_nodes(".typography_body-l__KUYFJ.typography_appearance-default__AAY17.typography_co
# Find elements with the specified class names (likely review text).
    html_text(trim = TRUE)
# Extract the text from those elements and remove extra spaces.



  reviewers <- page %>%
# Select <span> elements with specified classes (reviewer names).
    html_nodes("span.typography_heading-xxs__QKBS8.typography_appearance-default__AAY17") %>%
# Extract and trim text content.
    html_text(trim = TRUE)



  raw_text <- page %>%
# Use the HTML content from the 'page' object.
    html_nodes("p.typography_body-m__xgxZ_.typography_appearance-default__AAY17") %>%
# Find all <p> elements with the given class names.
    html_text(trim = TRUE)
# Extract the text content and remove extra spaces.



  date_entries <- raw_text[grep("^Date of experience:", raw_text)]
# Find all elements in 'raw_text' that start with "Date of experience:".
# Use 'grep' to locate these elements by matching the pattern.
# Extract and store them in 'date_entries'.


  cleaned_dates <- extract_dates(date_entries)
# Apply the 'extract_dates' function to 'date_entries'.
# This processes the text to extract and clean date information.
# Then stored in 'cleaned_dates'.
```

```
  star_ratings <- page %>%
# Find the star rating elements (inside divs with a specific class).
    html_nodes("div.star-rating_starRating__4rrcf img") %>%
# Get the 'alt' attribute from the images
#(this contains the star rating, like "5 stars").
    html_attr("alt") %>%
# Use the 'extract_rating' function to convert the text
# (e.g., "5 stars") into a numeric rating.
    extract_rating()
# The alt attribute in HTML is typically used to
# provide alternative text for an image (e.g., "5 stars" in a star rating image).



# Remove NA values from the star ratings
  clean_star_ratings <- na.omit(star_ratings)
# When scraping data from web pages,
# some entries might not have valid star ratings, resulting in NA values.
# These NA values can cause issues in further analysis,
# such as calculating averages or performing other numerical operations.



  review_titles <- page %>%
# Find all <h2> elements with the specified class (likely review titles).
    html_nodes("h2.typography_heading-s__f7029.typography_appearance-default__AAY17") %>%
# Extract the text content from those elements and trim extra spaces.
    html_text(trim = TRUE)



  min_length <- min(
    length(review_titles),
    length(review_texts),
    length(reviewers),
    length(cleaned_dates),
    length(clean_star_ratings)
  )
# Find the smallest number of items among all lists.
# This ensures all lists can match in size for consistent analysis.
```

```
  data.frame(
    Title = review_titles[1:min_length],
    Text = review_texts[1:min_length],
    Reviewer = reviewers[1:min_length],
    Date = cleaned_dates[1:min_length],
    Rating = clean_star_ratings[1:min_length],
    stringsAsFactors = FALSE
  )
# Create a table with review data, ensuring all columns have the same number of rows.
}
```

The `scrape_page` function collects review data from a specific page on Trustpilot. It builds the page's URL, retrieves its content, and extracts important details such as review titles, texts, reviewer names, dates, and star ratings. The function ensures all the extracted lists are of the same length, cleans the data (e.g., formatting dates and handling missing ratings), and combines it into a table. This table organizes the review data into a consistent format, making it ready for further analysis.

```
page_numbers <- 1:5
# Create a sequence of page numbers from 1 to 5.

all_pages_data <- lapply(page_numbers, scrape_page)
```

```
Warning in extract_rating(.): NAs introduced by coercion
Warning in extract_rating(.): NAs introduced by coercion
Warning in extract_rating(.): NAs introduced by coercion
Warning in extract_rating(.): NAs introduced by coercion
Warning in extract_rating(.): NAs introduced by coercion
```

```
# Apply the 'scrape_page' function to each page number.
# Collect the scraped data for each page into a list of data frames.

combined_df <- do.call(rbind, all_pages_data)
# Combine all the data frames from the list into a single data frame.
# Rows from each page are stacked together into 'combined_df'.

# do.call(): It splits a list (all_pages_data) into separate pieces
# and applies a function (like rbind) to them.
# In our case, it merges multiple data frames from all_pages_data into one using rbind.
```

This part of the code collects reviews from multiple pages by applying the `scrape_page` function to pages 1 to 5. It combines the data from all pages into a single table, where each row represents a review. This ensures that all the scraped data is organized and ready for analysis.

```
output_file <- "scraped_reviews.csv"
# Specify the name of the file to save the data as "scraped_reviews.csv".

write.csv(combined_df, file = output_file, row.names = FALSE)
# Save the combined data frame (combined_df) to a CSV file.
# Exclude row numbers from the saved file by setting row.names = FALSE.

cat("Data saved into", output_file, "\n")
```

```
Data saved into scraped_reviews.csv
```

```
# Confirmation message
```

This part of the code saves the combined review data into a CSV file named `scraped_reviews.csv` using the `write.csv` function. It excludes row numbers from the file for better readability. After saving, it prints a confirmation message to let us know that the data has been successfully saved.

Conclusion

Based on the results of this project, we successfully scraped reviews from the Uber Eats Trustpilot page and organized them into a clean and structured dataset. The dataset includes key details such as review titles, review text, reviewer names, dates, and star ratings for analysis.

This project demonstrates how web scraping techniques can effectively collect real-world data for deeper insights. The extracted data highlights both positive and negative customer experiences with Uber Eats, providing valuable information for further exploration, such as identifying common complaints, trends in ratings, or customer satisfaction over time. Overall, the project provides a solid foundation for data analysis and decision-making.