Taiwan Restaurants-MP2

Nhi Luong, Kyrylo Koltunov

2025-11-07

Introduction

In this project, we utilize the package 'rvest' with 'html_text' to scrape pieces of data that can appear anywhere on TripAdvisor webpage. We specifically want to look at information of restaurants in Taipei, Taiwan. The link to the page is https://www.tripadvisor.com/Restaurants-g293913-Taipei.html.

Our goal is to create a clean tibble containing each restaurant's information such as name, cuisine, ratings, number of reviews, and location. We use different functions to iterate through multiple pages and apply regular expression to extract and clean up the information.

Scraping TripAdvisor

Load in libraries

We started by downloading all the necessary libraries to get ready for content extraction.

```
library(tidyverse)
library(stringr)
library(rvest)
library(polite)
library(sf)
library(maps)
library(viridis)
library(leaflet)
library(htmltools)
```

Custom function 1: get_text_from_page()

The first step for us was creating a function called get_text_from_page.

This function takes a web page and a CSS selector as inputs. It then finds all the HTML nodes that match the selector and extracts their text content. For our convenience, we are using SelectorGadget extension for Chrome to identify the HTML nodes.

By utilizing this function, we don't have to repeat the process of writing out codes to scrape data.

```
get_text_from_page <- function(page, css_selector) {
  page |>
    html_nodes(css_selector) |>
    html_text()
}
```

Check for path and restriction

```
robotstxt::paths_allowed("https://www.tripadvisor.com/Restaurants-g293913-Taipei.html")
```

```
 www.tripadvisor.com
```

```
[1] TRUE
```

```
bow("https://www.tripadvisor.com/Restaurants-g293913-Taipei.html")
```

```
<polite session> https://www.tripadvisor.com/Restaurants-g293913-Taipei.html
    User-agent: polite R package
    robots.txt: 1606 rules are defined for 17 bots
   Crawl delay: 5 sec
  The path is scrapable for this user-agent
```

We need to check for the path to see if a bot has permission to access the TripAdvisor page, and the output is "TRUE", so we know the bot can access it. We then check if the page is scrapbable and by using the 'bow()' function, we know the path is scrapable but there is a restriction on Craw delay to be 5 seconds. Therefore, we know we can scrape the page but we have to set Sys.sleep(5).

By following these two steps, we are able to acquire data in an ethical and friendly manner. We are also making sure we are not scraping everything from the page, only essential information as we stated in the Introduction.

Custome function 2: scrape_page()

The next step for us was creating a new function called scrape_page. This function takes a link and waits for 5 seconds using Sys.sleep() before starting, so we don't overload the system. Its main goal is to collect data directly from TripAdvisor, such as the restaurant's name, rating, number of reviews, location, and cuisine type. We utilized the 'get_text_from_page()' function above to get the data from the TripAdvisor page.

Within the function, we have also included codes to combine the scraped data into a tibble and then apply regular expression to clean text data and parse numerical data.

```
scrape_page <- function(url) {
  Sys.sleep(5)
  session <- bow(url, force = TRUE)
  page <- scrape(session)
  rest_name <- get_text_from_page(page, ".mtnKn.OgHoE")
  ratings <- get_text_from_page(page, ".kzrsh span span")
  num_reviews <- get_text_from_page(page, ".kzrsh .qMyjI")
  location <- get_text_from_page(page, ".ZNjnF+ .ZNjnF")
  cuisine <- get_text_from_page(page, ".k+ div > .G")
  cuisine <- str_remove(cuisine, "\\$.*")
  tibble(rest_name, cuisine, ratings, location) |>
    mutate(ratings = as.numeric(ratings),
           rest_name = str_trim(str_extract(rest_name, "[^\\d\\.].*")),
           num_reviews = str_extract(num_reviews, "\\(.*\\)"),
           num_reviews = parse_number(str_extract(num_reviews, "\\d+(.*)\\d"))
    )
}
```

Take a look at how the function is used

```
scrape_page("https://www.tripadvisor.com/Restaurants-g293913-Taipei.html")
```

- Observation: We encountered a problem when trying to use this function to scrape the page. Based on the error, we discovered that a few locations are missing, so using this function results in an error saying the column lengths are not the same, and the code

stops executing. The first 3 columns (rest_name, ratings, and num_reviews) all have 30 observations, but 'location' only has 28 observations.

- Another problem with this is not knowing which index has missing values because the next value fills (slides-up) the gap of the missing one and so on, so it is hard to identify which restaurant's locations we are missing.

Implementing a New Method into the Function

After identifying the problem, a solution was developed by selecting a different pattern that includes restaurant information along with the location, if available. Using this longer string, the location is extracted with a regular expression. This ensures all columns have the same length and allows identifying the indices with missing values.

The changes made inside the function are changing to a different HTML nodes for 'location' and then uses 'str_remove()' function to only get the location of each restaurant. If there is no location, the value is "NA".

```
scrape_page_im <- function(url) {
  Sys.sleep(5)
  session <- bow(url, force = TRUE)
  page <- scrape(session)
  rest_name <- get_text_from_page(page, ".mtnKn.OgHoE")
  ratings <- get_text_from_page(page, ".kzrsh span span")
  num_reviews <- get_text_from_page(page, ".kzrsh .qMyjI")
  location <- get_text_from_page(page, ".UIwAG")
  location <- str_replace(str_extract(location, "mi(.*)"), "mi", "") # NEW LINE
  cuisine <- get_text_from_page(page, ".k+ div > .G")
  cuisine <- str_remove(cuisine, "\\$.*")
  tibble(rest_name, cuisine, ratings, location) |>
    mutate(ratings = as.numeric(ratings),
           rest_name = str_trim(str_extract(rest_name, "[^\\d\\.].*")),
           num_reviews = str_extract(num_reviews, "\\(.*\\)"),
           num_reviews = parse_number(str_extract(num_reviews, "\\d+(.*)\\d"))
    )
}
```

Test New Implemented Function

```
scrape_page_im("https://www.tripadvisor.com/Restaurants-g293913-oa")
```

```
# A tibble: 30 x 5
   rest_name                                cuisine ratings location num_reviews
   <chr>                                    <chr>     <dbl> <chr>          <dbl>
 1 Umeko Restaurant                         Chines~     4.5 Zhongsh~         635
 2  Shabu                                 Seafoo~     4.9 Xinyi D~        1374
 3 Din Tai Fung                             Taiwan~     4.5 Xinyi D~        4722
 4 A Point Steak & Bar                      Steakh~     4.9 Xinyi D~         467
 5 Shin Yeh Taiwanese Cuisine-Original Res~ Chines~     4.2 Zhongsh~         686
 6 Formosa Restaurant                       Chines~     4.9 Da'an            169
 7 Mosun Teppanyaki                         Southw~     4.9 Da'an            831
 8 Host Shabu                               Seafoo~     4.8 Xinyi D~        2441
 9 Yang Shin Vegetarian Restaurant          Canton~     4.4 Zhongsh~        2431
10 Hangzhou Xiaolong Bao                    Chines~     4.2 Da'an            806
# i 20 more rows
```

The new function works! We get a clean tibble of 30 observations since each page has 30 restaurants.

Applying the Function to Multiple Pages

After clicking through the pages, we see there is a specific part changes within the base url. Starting from page 2, the url includes "oa30", and the numerical part increases with multiples of 30, so page 3's url would contain "oa60" and so on. We concluded that except the first page, we can create a sequence starting from 30 with increment of 30 so iterate the function 'scrape_page_im()' over multiple pages.

We can see the difference in the urls between page 1 and page 2. Page 1: https://www.tripadvisor.com/Restaurants-g293913-Taipei.html Page 2: https://www.tripadvisor.com/Restaurants-g293913-oa30-Taipei.html

Create base url and sequence

We approached this problem by breaking the base url into two parts, then creating a sequence starting from 30 to 180 with increment of 30. Then, we used 'str_c()' to join multiple parts of the url together into one complete url and save it under "urls_all_pages". We saved the first page to a separate name called "url_first_page".

```
base_url1 <- "https://www.tripadvisor.com/Restaurants-g293913-oa"
base_url2 <- "-Taipei.html"
```

```
sequence <- seq(30, 150, 30)
```

```
urls_all_pages <- str_c(base_url1, sequence, base_url2)
url_first_page <- "https://www.tripadvisor.com/Restaurants-g293913-Taipei.html"
```

Using map functions in the purrr package

Here, we applied it to first_page using "url_first_page", then to all other pages using "urls_all_pages". After that, we combine all the results into one dataset called "rest_info" using 'bind_rows()' function.

```
first_page <- purrr::map(url_first_page, scrape_page_im)
pages <- purrr::map(urls_all_pages, scrape_page_im)
rest_info <- bind_rows(first_page, pages)
```

```
rest_info
```

```
# A tibble: 180 x 5
   rest_name                                cuisine ratings location num_reviews
   <chr>                                    <chr>     <dbl> <chr>          <dbl>
 1 Umeko Restaurant                         Chines~     4.5 Zhongsh~         635
 2  Shabu                                 Seafoo~     4.9 Xinyi D~        1374
 3 Din Tai Fung                             Taiwan~     4.5 Xinyi D~        4722
 4 A Point Steak & Bar                      Steakh~     4.9 Xinyi D~         467
 5 Shin Yeh Taiwanese Cuisine-Original Res~ Chines~     4.2 Zhongsh~         686
 6 Formosa Restaurant                       Chines~     4.9 Da'an            169
 7 Mosun Teppanyaki                         Southw~     4.9 Da'an            831
 8 Host Shabu                               Seafoo~     4.8 Xinyi D~        2441
 9 Yang Shin Vegetarian Restaurant          Canton~     4.4 Zhongsh~        2431
10 Hangzhou Xiaolong Bao                    Chines~     4.2 Da'an            806
# i 170 more rows
```

Finally, we saved the complete dataset as a CSV file named "rest_info.csv" using 'write_csv()' function.

```
write_csv(rest_info, "rest_info.csv")
```

Conclusion

At the end, we were able to scrape and create a clean table of each restaurant's information. The first column is restaurant name called "rest_name". The second column is the type of cuisine that the restaurant represents called "cuisine". The third is "ratings", representing the overall ratings from customers and the scale is out of 5. The fourth one is "location", which is different districts in Taipei city. The fifth column is "num_reviews", representing the total number of reviews of each restaurants.

We could use this table for further analysis such as finding out which district has the highest ratings overall or which cuisine receives highest ratings and number of reviews.