**Web Scraping in R**

Web scraping needs no introduction among Data enthusiasts. It’s one of the most viable and most essential ways of collecting Data when the data itself isn’t available.

Knowing web scraping comes very handy when you are in shortage of data or in need of Macroeconomics indicators or simply no data available for a particular project like a Word2vec / Language with a custom text dataset.

rvest a beautiful (like BeautifulSoup in Python) package in R for web scraping. It also goes very well with the universe of tidyverse and the super-handy %>% pipe operator.

**Disclaimer: This tutorial is for pure educational purpose, Please check any website’s ToS before scraping them**

**Sample Use-case**

Text Analysis of how customers feel about Etsy.com. For this, we are going to extract reviews data from [trustpilot.com](http://trustpilot.com/).

Below is the R code for scraping reviews from the first page of Trustpilot’s Etsy page. [URL: https://www.trustpilot.com/review/www.etsy.com?page=1](https://www.trustpilot.com/review/www.etsy.com?page=1)

library(tidyverse) #for data manipulation - here for pipe

library(rvest) - for web scraping #single-page

scrapingurl <- "<https://www.trustpilot.com/review/www.etsy.com?page=1>"

scrapingurl %>%

read\_html() %>%

html\_nodes(".review-content\_\_text") %>%

html\_text() -> reviews

This is fairly a straightforward code where we pass on the URL to read the html content. Once the content is read, we use html\_nodes function to get the reviews text based on its css selector property and finally just taking the text out of it html\_text() and assigning it to the R object reviews .

Below is the sample output of reviews:



Well and Good. We’ve successfully scraped the reviews we wanted for our Analysis.



But the catch is the amount of reviews we’ve got is just 20 reviews — in that as we can see in the screenshot we’ve already got a non-English review that we might have to exclude in the data cleaning process.

This all puts us in a situation to collect more data to compensate the above mentioned data loss and make the analysis more effective.

**Need for Scale**

With the above code, we had scraped only from the first page (which is the most recent). So, Due to the need for more data, we have to expand our search to further pages, let’s say 10 other pages which will give us 200 raw reviews to work with before data processing.

**Conventional Way**

The very conventional way of doing this is to use a loop — typically forloop to iterate the URL from 1 to 20 to create 20 different URLs (String Concatenation at work) based on a base url. As we all know that’s more computationally intensive and the code wouldn’t be compact either.

**The Functional Programming way**

This is where we are going to use R’s functional programming support from the package purrr to perform the same iteration but quite in R’s tidy way within the same data pipeline as the above code. We’re going to use two functions from purrr ,

1. map() is the typical map from the functional programming paradigm, that takes a function and maps onto a series of values.
2. map2\_chr() is the evolution of map that takes additional arguments for the function and formats the output as a character.

**Below is our Functional Programming Code**

library(tidyverse)

library(rvest)

library(purrr) #multi-page

url <- "<https://www.trustpilot.com/review/www.etsy.com?page=>" # base URL without the page number

url %>%

map2\_chr(1:10,paste0) %>% #for building 20 URLs

map(. %>%

read\_html() %>%

html\_nodes(".review-content\_\_text") %>%

html\_text()

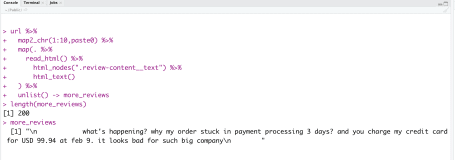
) %>%

unlist() -> more\_reviews

As you can see, this code is very similar to the above single-page code and hence it makes it easier for anyone who understand the previous code to read this through with minimal prior knowledge.

The additional operations in this code is that we build 20 new URLs (by changing the query value of the URL) and pass on those 20 URLs one-by-one for web scraping and finally as we’d get a list in return, we use unlist to save all the reviews whose count must be 200 (20 reviews per page x 10 pages).

Let’s check how the output looks:



Yes, 200 reviews it is. That fulfills our goal of collecting (fairly) sufficient data for performing the text analysis use-case we mentioned above.

But the point of this article is to introduce you to the world of functional programming in R and to show how easily it fits in with the existing data pipeline / workflow and how compact it is and with a pinch of doubt, how efficient it is (than a typical for-loop). Hope, the article served its purpose.

* **If you are more interested, Check out this** [**Datacamp course on Functional Programming with purrr**](https://bit.ly/2ktbOzd)