Scraping Webpages

Rodrigo Esteves de Lima-Lopes
State University of Campinas
rll307@unicamp.br

Contents

1 Introduction **1**

2 What we are going to need **1**
2.1 R packages 1
2.2 External software 2
2.3 Data souce 2

3 Preparing Data scraping **2**
3.1 Setp 1: Getting the codes 2
3.2 Step 2: Getting the titles and addresses 2
3.3 Setp 3: Downloading function 2

4 Scraping the data **3**

1 Introduction

In this tutorial we are going to try to scrape some data from websites. However, we should consider that

- A common and basic knowledge about CSS and HTML is necessary. To make things easier I would suggest SelectorGadget a Google Chrome extension which might help us to get the job done.
- Not all recipes work the same in all websites. It is mostly due to security matters and, sometimes, changes in the way the site's is coded.
- I am not a full trained programmer, **R** master or software engineer, so probably there will be more elegant and functional scripts around.
  - This is why you will notice I am unable to get all the addresses from a website recursively. On the contrary, I need to save them in a data frame as we will see and then get the texts.
- Scraping websites is a nasty endeavour, mostly because one is not like the other, sometimes a change in the code will make us change our entirely.
- I will not address some other issues, such as text cleaning or data consolidation due to the scope of this tutorial.

Please keep in mind that this script aims to be used for ***research purposes only***. It is imperative you visit the website and get information about their policy, because sometimes there might be some restrictions.

2 What we are going to need

2.1 R packages

We are going to need some R packages:

1. rvest

2. purrr
3. xml2
4. readr
5. dplyr

These packages are for reading external files, reading the website code and manipulating the final data frame.

2.2 External software

2.2.1 Spreadsheet

We will need to use a spreadsheet for collecting the addresses and titles. I would suggest to use LibreOffice, since other suites might ruin the text encoding.

2.2.2 Google Chrome extention

Since I do not know much about CSS and HTML5 coding, I cheat my code using SelectorGadgeton Google Chrome. Such extension opens Pandora's Box of CSS code.

2.3 Data souce

In this tutorial we will be using Lula's interviews at Roda Viva interviews, scraped from the official FAPESP and Roda Viva website.

3 Preparing Data scraping

3.1 Setp 1: Getting the codes

We are going to check the website's consistency in terms of the CSS codes for data scraping. Please:

1. Access any of Lula's interview from the website.
2. Run `SelectorGadget` and click on any paragraph.
3. Check which code is given to us.

I all my tries, the `SelectorGadget` showed me that `p` is the code for the texts paragraph. I will use such information in a short while.

3.2 Step 2: Getting the titles and addresses

Like I said earlier, I do not have the expertise to scrape a website recursively. So I prepare a CSV document using a LibreOffice spreadsheet. In this spreadsheet it write four columns:

1. Article: The word *A* plus a number, simply for identification
2. Title: The article's title, for more identification
3. Address: The article's `URL`, for scraping in **R**
4. Text: Empty, that will have the article text we are going to scrap

This file is available under the name *base.csv* at data directory

3.3 Setp 3: Downloading function

Now we are going to write a basic function to download the articles and save it to the original data frame. You don't need to rewrite it, but I will take my time to explain how it was written. The name of my function is `extract_articles` and its scope will be a single argument `x`, which is the URL saved as as vector (see later)

```
extract_articles <- function(x){
```

The `read_html(x)` will extract the html code from the URL and save it temporally in a variable called `webpage`

```
webpage <- read_html(x)
```

The next step is to extract only the node `p` to another variable. It is importat to keep in mind that this variable might change depending on the webpage.

```
  text <- html_nodes(webpage,'p') #this "p"should change depending on file
```

Now we are going to extract the text regarding this node `p`, since we are only interested in it. We are also going to collapse the text in a single line.

```
  text_data <- html_text(text)
  Article <- paste(text_data,collapse="\n")
```

Our last step is to return the variable and close the function.

```
  return(Article)
}
```

4 Scraping the data

First loading the packages.

```
library(rvest)
library(purrr)
library(xml2)
library(readr)
```

Now, let us load the data we prepared.

```
base <- read.csv("./data/base.csv", row.names = 1, header= TRUE)
```

It is time to save our addresses for the scraping.

```
# Pulling out the address of the article
article_general <- as.vector(base[,2])
```

Then load the scraping function we wrote just above

```
extract_articles <- function(x){
  webpage <- read_html(x)
  text <- html_nodes(webpage,'p')
  text_data <- html_text(text)
  Article <- paste(text_data,collapse="\n")
  return(Article)
}
```

Finally let us scrap the data. Please remember that we have to change the number of the element in our vector and the number of the line in our data frame, so we have the right text in the right place:

```
## Extracting --------------------------------------------
base[6,3] <-extract_articles(article_general[6])
```