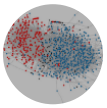
Automating the Google search for the web presence of 8000+ organizations



CuriosityBits Data Lab

Dec 8, 2016 · 4 min read

I recently worked on a spreadsheet containing a list of over 8500 nonprofits. For each nonprofit, I wanted to get the URLs to its website, Facebook page and Twitter account. The task is arduous, if done without some sort of automation. In the below, I will show you how to use a few lines of simple R scripts to get the task done.

1. **Install R, RStudio and libraries.**

R is an open-source data analytics engine, the lingua franca in data mining, along with Python. RStudio is an integrated development environment (IDE) for running R scripts and tracking outputs. You can install it from here.

After you create a project in RStudio, you can use the following scripts to install R libraries needed for the task. For example, rvest is a library for simple web scrapping.

```
install.packages(“rvest”)
install.packages(“urltools”)
```

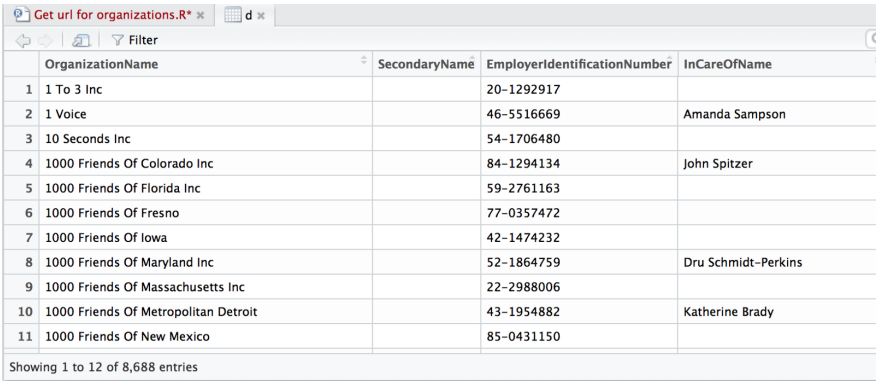
```
require("rvest")
require("urltools")
```

2. Load spreadsheet into R

My spreadsheet is called *data.csv*, and is located in the same folder where the R project for this task is saved. I can use the following line to import the spreadsheet. In R the imported spreadsheet is named *d*.

```
d <- read.csv("data.csv")
View(d)
```

View(d) will open the spreadsheet and you will see how the data are structured.



Get url for organizations.R* | d

Filter

	OrganizationName	SecondaryName	EmployerIdentificationNumber	InCareOfName
1	1 To 3 Inc		20-1292917	
2	1 Voice		46-5516669	Amanda Sampson
3	10 Seconds Inc		54-1706480	
4	1000 Friends Of Colorado Inc		84-1294134	John Spitzer
5	1000 Friends Of Florida Inc		59-2761163	
6	1000 Friends Of Fresno		77-0357472	
7	1000 Friends Of Iowa		42-1474232	
8	1000 Friends Of Maryland Inc		52-1864759	Dru Schmidt-Perkins
9	1000 Friends Of Massachusetts Inc		22-2988006	
10	1000 Friends Of Metropolitan Detroit		43-1954882	Katherine Brady
11	1000 Friends Of New Mexico		85-0431150	

Showing 1 to 12 of 8,688 entries

To access a particular column, for example the first column which contains the names of the organizations, simply run

```
d$OrganizationName
```

3. Start web scrapping—the simple version

We first create three new columns, named *Website*, *Twitter* and *Facebook* respectively. We fill the columns with *NA* as the placeholder for the output from web scrapping.

```
d$Website <- "NA"
d$Twitter <- "NA"
d$Facebook <- "NA"
```

We then use the library *rvest* to make Google Search requests, with each organization name as the search term. Here is a simple version of the web scrapping function.

```
for (name in d$OrganizationName[1:2000]){

 print(paste0("finding the url for:",name))


 Sys.sleep(3)


 url1 = URLencode(paste0("https://www.google.com/search?
q=",name))
 page1 <- read_html(url1)
 results1 <- page1 %>% html_nodes("cite") %>% html_text()
 result1 <- as.character(results1[1])


 d[d$OrganizationName==name,]$Website <- result1


}
```

In line 1, you will notice that we let R to parse from the **1st through the 2000th cell in the column called *OrganizationName*** (you can of course change the numbers to suit your need). We also let R to **print** (a.k.a, display in the R console) each organization name.

Sys.sleep(3) is put in so that R will rest for **3 seconds** after each scrapping request.

url1 is the unique Google search url based on a given search term. It is https://www.google.com/search?q=, followed by a search term. So, to search *red cross*, the url is: https://www.google.com/search?q=red+cross (*note:* the space between two words is replaced by + or %20).

page1 <- read_html(url1) is using the **read_html** function in *rvest* to download the html code using a search url. **results1** contains the html output. And **result1** is the first search result.

4. Web scrapping—full features

The simple web scrapping function we've created from the above step only returns website URLs. What about Facebook pages and Twitter accounts? We can add a few lines to the function to do more advanced scrapping. The idea is simple: we add two more Google searches, each is limited to a specific domain. For example, by adding

"**site:twitter.com**" to the search url, we can get search results strictly from twitter.com.

The complete web scrapping function will be like this:

```
for (name in d$OrganizationName[1531:2000]){
 print(paste0("finding the url for:",name))
 Sys.sleep(3)

 url1 = URLencode(paste0("https://www.google.com/search?
q=",name))
 page1 <- read_html(url1)
 results1 <- page1 %>% html_nodes("cite") %>% html_text()
 result1 <- as.character(results1[1])
 d[d$OrganizationName==name,]$Website <- result1

 print(paste0("finding the Twitter url for:",name))
 url2 = URLencode(paste0("https://www.google.com/search?
q=",gsub(" ","+",name),"+site:twitter.com"))
 page2 <- read_html(url2)
 results2 <- page2 %>% html_nodes("cite") %>% html_text()
 result2 <- as.character(results2[1])
 d[d$OrganizationName==name,]$Twitter <- result2

 print(paste0("finding the Facebook url for:",gsub("
","+",name)))
 url3 = URLencode(paste0("https://www.google.com/search?
q=",gsub(" ","+",name),"+site:facebook.com"))
 page3 <- read_html(url3)
 results3 <- page3 %>% html_nodes("cite") %>% html_text()
 result3 <- as.character(results3[1])
 d[d$OrganizationName==name,]$Facebook <- result3
}
```

Caveats

You may not always get accurate search result—the first website may not be related to your target organization at all. That's when you need data cleaning—part of it can be automated as well!

You can create a new column to store the second website in a returned search result. For example, you can add the following two lines to the function.

```
result4 <- as.character(results1[2])
d[d$OrganizationName==name,]$Alternative_result <- result4
```

Since all organizations on my list are nonprofits. Their website URLs should end with .org. Using the following, I can create a new data frame (i.e., spreadsheet) to include only organizations whose URLs contain ".org".

```
c<-d[grep(".org", d$Website), ]
```

In the end, we probably cannot get away without having human eyes check the data. But now the task is far less daunting.