Data and Climate

Session 4 - Web scraping R

Jean-Baptiste Guiffard

2024-01-10

Sessions

Sessions	Topics
Session 1	The Basics of R / Manipulating dataset with the DPLYR package
Session 2	Graphic representations with GGPLOT
Session 3	Making maps with R
Session 4	Web scraping with R
Session 5	Extracting and analyzing textual data using R
Session 6	Produce documents with Rmarkdown. . .

Une introduction générale au scraping

Le webscrapping ?

Définition : Le *scraping* ou *harvesting* désigne une technique qui permet d'extraire de manière automatisée des informations depuis un ou plusieurs sites web. Pour cela on passe par un script (un algorithme) qui permet l'extraction ou "pillage" de ces informations.

En tant que chercheurs, le web scraping est intéressant puisqu'il :

- permet de transformer en données organisées des informations disponibles et dispersées sur un site web, facilitant alors l'exploitation de ces informations.
- Il peut être un moyen d'obtenir gratuitement de l'information que nous n'aurions pas pu avoir autrement.

Pour cette présentation : Une étude du package le plus simple avec une structure de site web basique

Différents types de scraping (I)

1. Le webscrapping le plus simple consiste à faire une requête directement au serveur et à importer le code **HTML**[1] d'une page web et ensuite sélectionner les parties du code HTML qui nous intéresse, à l'aide de fonctions de *text mining* (traitement alphanumérique) $\rightarrow$ Package sur R : **rvest**.

[1] HTML signifie *HyperText Markup Language* et est un langage pour la production de pages web que nous pouvons lire et analyser en faisant "Ctrl + Maj + C" ou bien clic droit et "Inspecter".

Différents types de scraping (II)

2. Simuler la navigation d'un utilisateur à travers les contenus d'un serveur. Un navigateur Internet permet alors de se déplacer sur le site web. Des actions humaines peuvent être alors paramétré : cliquer sur un élément de la page web, déplacer la souris, entrer des mots dans une barre de recherche... Si on doit se connecter sur un site, cela évite de perdre la connexion entre chaque requête. → Package sur R : **RSelenium**.
3. Les robots d'indexation (*web crawler*) : collecte des ressources afin de permettre à un moteur de recherche leur indexation → Package sur R : **Rcrawler**.
4. Les API (Interfaces de programmation d'applications) : application prévue par un site web/une plateforme pour l'interaction "automatisée" avec cette dernière : collecte et envoie d'informations. → Exemple API Twitter

Des règles à respecter ?

Deux aspects importants à prendre en considération avant de commencer à scraper un site web :

- Les Conditions Générales d'Utilisation (CGU) et particulièrement les règles du site web à l'égard de la récupération automatisée de ses données. Pour avoir cette information : "nom_du_site.com"+"/robots.txt"
- La question de la collecte des données personnelles : sur une plateforme type Facebook, ou Twitter, considérer la collecte des informations personnelles et qui peuvent être "sensibles". Certaines règles sont à respecter dans le cadre de la législation sur les données personnelles (RGPD). $\rightarrow$ Question de l'anonymisation des données...

Quelques précautions avant de s'engager...

Aller au-delà des sécurités mises en place par le site web :

- Gestion du temps de scraping : Les temps d'attente entre deux requêtes ;
- Cacher/déguiser l'adresse IP : VPN, rotating proxies...
- Ne pas suivre un schéma de collecte récurrent : simuler un comportement humain (ajouter de l'aléatoire)
- Essayer de suivre les indications du site sur le scraping (robots.txt)

Les étapes du webscrapping : Un exemple simple

Les étapes du webscrapping

1. Identifier les éléments d'intérêt dans le code HTML ;
2. Télécharger la page HTML ;
3. Utiliser les fonctions de traitement alphanumériques nécessaires à l'extraction de la ou des donnée(s) souhaitée(s) ;
4. Stocker ces données dans une base.

1. Identifier les éléments d'intérêt dans le code HTML (I)

Un des principaux enjeux du scraping est de venir chercher l'information exacte qui est recherchée. Pour cela, il faut identifier des patterns qui se répètent dans une page web et qui nous assurerons de récupérer l'information toujours au bon endroit. Un code HTML est structuré en utilisant des *nodes* qui ont un nom, des attributs, un texte...

Un système de poupées russes...

1. *The root node* qui n'a pas de parent ;
2. Chaque autre node a un parent ;
3. Un node peut avoir un zero, un ou plusieurs *children* (les attributs et textes n'ont pas d'enfants).
4. Les *Siblings* sont des nodes avec les mêmes parents

1. Identifier les composants d'intérêt dans le code HTML (II)

- Les *tag names* : entre crochets ("< >") avec "p" qui désigne un paragraphe d'un text, "li" une cellule dans une liste, "a" un lien, "div" un block d'espace dans une page, "meta" de l'information à propos d'une page qui n'est pas montrée... (https://developer.mozilla.org/en-US/docs/Web/HTML/Element)
- Les attributs (associé à un élément) : "id" permet d'identifier un élément d'un document de façon unique ; "href" l'URL de la ressource liée ; class un attribut universel qui permet de définir la ou les classes auxquelles appartient un élément ; label qui définit un titre lisible par un utlisateur pour l'élément (https://developer.mozilla.org/fr/docs/Web/HTML/Attributes)

2. Télécharger la page HTML

```
install.packages('rvest')
library(rvest)
url <- "mapage."
code_html <- read_html(url, encoding="UTF-8")
```

3. Extraire l'information du code HTML (I)

Identifier formellement un élément d'un code HTML

- Avec un node :

```
title_node <- html_nodes(code_html,'title')
title <- html_text(title_node)
```

- Avec la position hiérarchique

```
info_node <- code_html %>% html_nodes('title') %>% html_nodes('p')
info <- html_att(info_node, "href")
```

3. Extraire l'information du code HTML (II)

- Avec CSS selectors (CSS ou X-Path) : essayer d'identifier un élément unique

```
info_node <- code_html %>% html_nodes(".cast_list .character")
info <- html_att(info_node, "href")

#xpath="./text()[normalize-space()]")
```

pour la sélection
https://data-lessons.github.io/library-webscraping-DEPRECATED/02-csssel/

https://rvest.tidyverse.org/articles/harvesting-the-web.html#extracting-html-elements-with-xpath-1

Exemple : Scrappons un tableau de données sur Wikipédia

```
url <- "https://fr.wikipedia.org/wiki/Liste_de_batailles_du_XIXe_si%C3%A8cle"
code_html <- read_html(url, encoding="UTF-8")

tables <- code_html %>% html_nodes("main") %>% html_nodes("table")
batailles_afr <- as.data.frame(tables[1] %>% html_table(fill=TRUE))
batailles_afr$guerre <- ifelse(batailles_afr$Bataille==batailles_afr$Date.s., b
for (x in 1:length(batailles_afr$guerre)){
  if(is.na(batailles_afr[x,'guerre'])){
    batailles_afr[x,'guerre']=value
  }else{
    value=batailles_afr[x,'guerre']
  }
}
batailles_afr <- subset(batailles_afr, Résultat!=guerre)
```

Exemple : Réalisation d'une carte avec les éléments scrappés

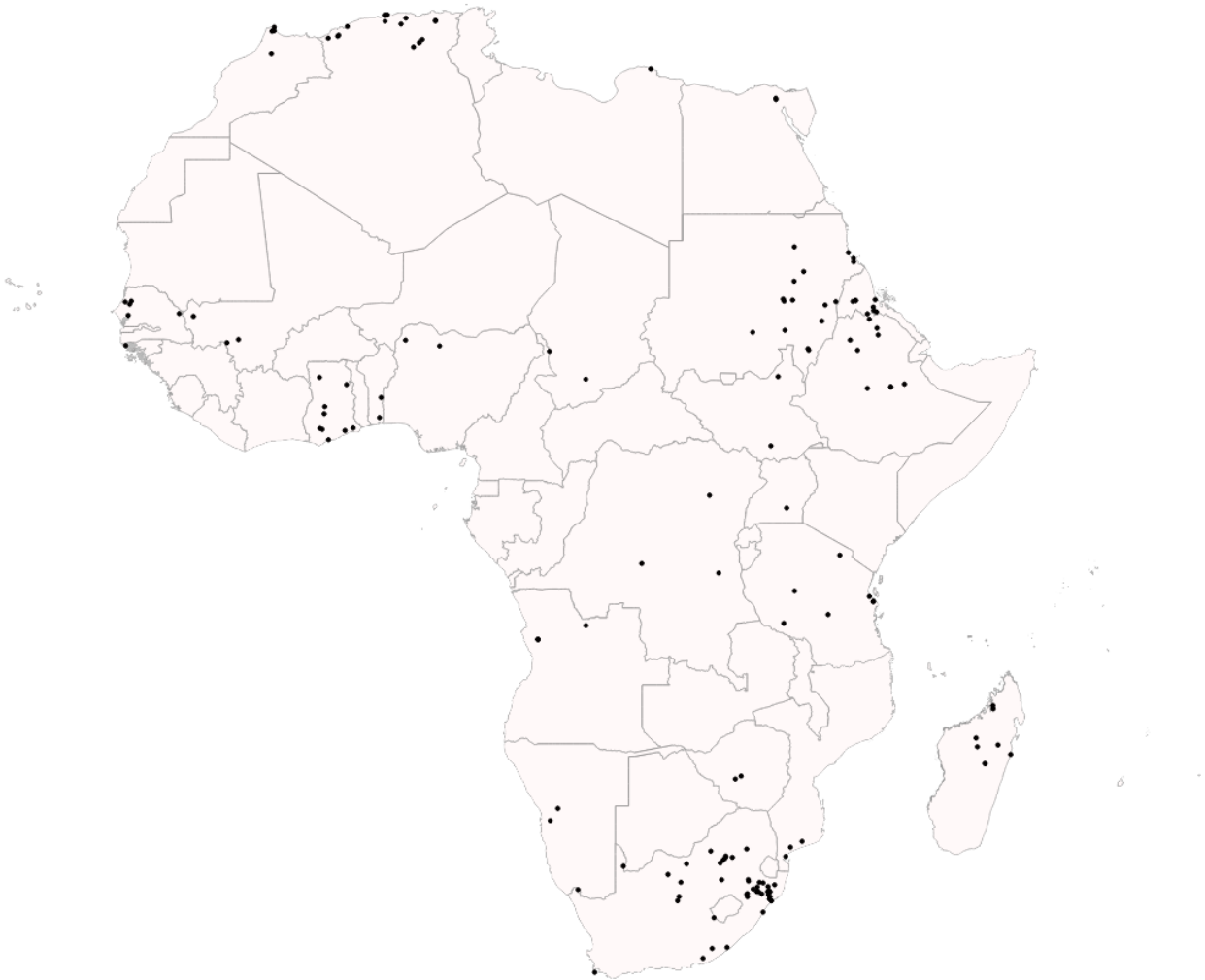

Figure 1: Carte Batailles XIXème siècle en Afrique

Exemple : Réaliser une carte des pays du monde par leur empreinte écologique brute (Global Footprint Network)