R avancé

Web Scraping

Nicolas Klutchnikoff

February 2023

Introduction

Beaucoup d'informations disponibles

Les sites web sont une source importante d'information :

- Site interne d'entrprise ;
- Wikipedia ;
- IMDb ;
- etc ;

Parfois, l'information est assez structurée pour qu'on puisse la récupérer de façon semi-automatisée.

Objectifs

Nous allons chercher avec quel acteur Louis de Funès a le plus joué dans les années 1950.

On peut commencer par visiter la page dédiée à la filmographie de Louis de Funès sur WIkipédia.

On constate :

- Les films sont classés par décénie ;
- Chaque titre de film possède un lien vers une page dédiée ;
- Cette dernière page contient une liste d'acteurs.

Le package `rvest`

C'est un package difficile à installer car il requiert la présence de certains outils sur le système d'exploitation : `curl` par exemple.

Tout est expliqué dans les messages d'aide :

```
install.packages("rvest")
```

On peut ensuite l'utiliser

```
library(rvest)
```

```
##
## Attaching package: 'rvest'

## The following object is masked from 'package:readr':
##
##     guess_encoding
```

Le DOM

HTML, CSS

Rappelons que notre navigateur web se contente de transformer des fichiers textes en pages lisibles et bien présentées. Pour simplifier un peu :

- les informations structurées sont contenues dans un fichier HTML ;
- la présentation est codée dans des feuilles de style css.

C'est donc dans le fichier source HTML que nous allons trouver les renseignements que nous voulons.

Des arbres en plein HTML

En inspectant le fichier HTML, nous constatons que la partie du code qui contient les informations qui nous intéressent est à peu près structurée de la façon suivante :

```
<h3>Longs métrages</h3>
  <h4>Années 40<h4>
  <ul>
    <li>...</li>
  </ul>
```

Ces données peuvent être représentées de façon **arborescente**.

L'interface de programmation DOM permet d'examiner et de mo- difier cet arbre et donc le contenu d'une page web. C'est de cette façon que des pages web dynamiques sont créées.

C’est parti !

Le package **rvest** est capable de parcourir cet arbre et donc de récupérer des infor- mations.

```
url_wikipedia <- "https://fr.wikipedia.org/"
url_filmographie <- "wiki/Filmographie_de_Louis_de_Funès"
url <- paste0(url_wikipedia, url_filmographie)
```

```
data_html <- read_html(url)
data_html
```

```
## {html_document}
## <html class="client-nojs vector-feature-language-in-header-enabled vector-
feature-language-in-main-page-header-disabled vector-feature-language-
alert-in-sidebar-enabled vector-feature-sticky-header-disabled vector-
feature-page-tools-disabled vector-feature-page-tools-pinned-disabled vector-
feature-main-menu-pinned-disabled vector-feature-limited-width-enabled vector-
feature-limited-width-content-enabled" lang="fr" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-
8 ...
## [2] <body class="skin-vector skin-vector-search-vue vector-toc-pinned mediawi ..
```

Les titres de section

On veut tous les titres des sections de niveau `<h4>` en utilisant la fonction `html_nodes()` qui permet de retourner une liste de nœuds :

```
data_html |> html_nodes("h4") |> head(1)
```

```
## {xml_nodeset (1)}
## [1] <h4>\n<span id="Ann.C3.A9es_1930"></span><span class="mw-headline" id="An ..
```

Ce n'est pas encore très satisfaisant.

```
data_html |> html_nodes("h4") |> head(1) |> html_text()
```

```
## [1] "Années 1930[modifier | modifier le code]"
```

Le noeud du problème

Sélectionner les nœuds

Il existe au moins deux méthodes pour le faire :

- utiliser des **sélecteurs css** ;
- utiliser XPath.

Ce dernier est plus complexe mais plus puissant.

- Utilisation occasionnelle => sélecteurs css
- Utilisation fréquente => XPath

Exemple, sélecteurs css (1)

```
data_html |>
  html_nodes('#mw-content-text > div > ul:nth-child(10) > li:nth-child(1) > i > a') |>
  html_attrs()
```

```
## [[1]]
##                                href                                title
## "/wiki/La_Tentation_de_Barbizon"       "La Tentation de Barbizon"
```

Pas mal ! mais :

- D'où vient ce sélecteur étrange !
- Comment s'en sortir de façon plus automatique ?

Exemple, sélecteurs css (2)

```
data_html |>
  html_nodes('#mw-content-text > div > ul:nth-of-type(3) > li > i > a') |>
  html_attrs()
```

```
## [[1]]
##                              href                                 title
## "/wiki/Au_revoir_monsieur_Grock"       "Au revoir monsieur Grock"
##
## [[2]]
##                                    href
## "/wiki/Pas_de_week-end_pour_notre_amour"
##                                   title
##       "Pas de week-end pour notre amour"
##
## [[3]]
##                      href                    title
## "/wiki/Mon_ami_Sainfoin"       "Mon ami Sainfoin"
##
## [[4]]
##                                href                                   title
## "/wiki/Rendez-vous_avec_la_chance"       "Rendez-vous avec la chance"
##
```

Exemple, XPath

```
data_html |>
  html_nodes(xpath ='//*[@id="mw-content-text"]
    /div/ul[
    preceding::h4[span/@id="Années_1950"]
    and
    following::h4[span/@id="Années_1960"]]
    /li/i/a') |>
  head(5) |>
  html_text()
```

```
## [1] "Au revoir monsieur Grock"          "Pas de week-end pour notre amour"
## [3] "Mon ami Sainfoin"                  "Rendez-vous avec la chance"
## [5] "Adémaï au poteau-frontière"
```

XPath permet de se déplacer librement dans l'arbre tandis que les sélecteurs css ne permettent pas de remonter. C'est bien plus puissant et utile pour toutes les données de type XML.

Quai de Grenelle

URL

```
url_film <- "wiki/Quai_de_Grenelle"
url <- paste0(url_wikipedia,url_film)
data_html <- read_html(url)
```

Un bon découpage entre `url_wikipedia` et `url_film` est important pour automatiser la lecture des pages.

Mauvaise nouvelle : il est indispensable de lire le code source HTML pour comprendre !

Un peu de XPath

```
data_html |>
  html_nodes(xpath = '(
    //*[@id="mw-content-text"]
    //ul[preceding::h2[span/@id="Distribution"]]
    )[1]/li/a[1]') |>
  html_text()
```

Explications :

- `//*[@id="mw-content-text"]` permet de sélectionner tous les nœuds descendants de la balise ayant cet identifiant ;
- `//ul[preceding::h2[span/@id="Distribution"]]` sélectionne tous les `<ul>` dans ce sous-arbre qui suivent une balise `<h2>` contenant une balise `<span>` identifiée par le mot-clé `Distribution` ;
- on sélectionne la première liste avec `[1]`
- puis, dans cette liste, toutes les premières balises `<a>` qui suivent une balise `<li>` avec le code `/li/a[1]`

Tout ensemble (1)

On commence par obtenir la liste des films :

```
url_wikipedia <- "https://fr.wikipedia.org"
url_de_funes <- "/wiki/Filmographie_de_Louis_de_Funès"
url <- paste0(url_wikipedia, url_de_funes)
data_html <- read_html(url)
films <- data_html |>
  html_nodes('#mw-content-text > div > ul:nth-of-type(3) > li > i > a') |>
  html_attrs()
```

```
films
```

```
## [[1]]
##                              href                               title
## "/wiki/Au_revoir_monsieur_Grock"       "Au revoir monsieur Grock"
##
## [[2]]
##                                      href
## "/wiki/Pas_de_week-end_pour_notre_amour"
##                                     title
##       "Pas de week-end pour notre amour"
##
## [[3]]
```

Tout ensemble (2)

Puis la liste des acteurs

```
liste_acteurs <- tibble()
for(i in seq_along(films)){
  titre <- films[[i]][2]
  url_film <- films[[i]][1]
  url <- paste0(url_wikipedia, url_film)
  data_html <- read_html(url)
  acteurs <- data_html |>
    html_nodes(xpath = '(
        //*[@id="mw-content-text"]
        //ul[preceding::h2[span/@id="Distribution"]]
        )[1]/li/a[1]') |>
    html_text()
liste_acteurs <- rbind(liste_acteurs,tibble(nom = acteurs,titre = titre))
}
```

Tout ensemble (3)

```
liste_acteurs |>
  group_by(nom) |>
  summarise(n = n()) |>
  arrange(desc(n)) |>
  head(4)
```

```
## # A tibble: 4 x 2
##   nom                n
##   <chr>          <int>
## 1 Louis de Funès    82
## 2 Paul Demange      16
## 3 Charles Bayard    14
## 4 Noël Roquevert    14
```

On constate qu'il s'agit de Paul Demange !

Et avec les tables HTML ?

Une fonction dédiée

```
read_html("https://fr.wikipedia.org/wiki/Roger_Federer") |>
  html_table(fill = TRUE) |>
  pluck(2)
```

```
## # A tibble: 16 x 92
##    X1    X2    X3    X4    X5    X6    X7    X8       X9   X10   X11 X12     X13
##    <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <int> <int> <int> <chr> <int>
##  1 "Tit~ Caté~ Dur   Terre Gazon Synt~ Total Gran~    11     1     8 -        20
##  2 "Cat~ Dur   Terre Gazon Synt~ Total <NA>  <NA>     NA    NA    NA <NA>     NA
##  3 "Gra~ 11    1     8     -     20    <NA>  <NA>     NA    NA    NA <NA>     NA
##  4 "Mas~ 6     -     -     -     6     <NA>  <NA>     NA    NA    NA <NA>     NA
##  5 "Mas~ 22    6     -     -     28    <NA>  <NA>     NA    NA    NA <NA>     NA
##  6 "ATP~ 21    0     3     0     24    <NA>  <NA>     NA    NA    NA <NA>     NA
##  7 "ATP~ 11    4     8     2     25    <NA>  <NA>     NA    NA    NA <NA>     NA
##  8 "Tot~ 71    11    19    2     103   <NA>  <NA>     NA    NA    NA <NA>     NA
##  9 "Cat~ Dur   Terre Gazon Synt~ Total <NA>  <NA>     NA    NA    NA <NA>     NA
## 10 "Gra~ 3     4     4     -     11    <NA>  <NA>     NA    NA    NA <NA>     NA
## 11 "Mas~ 3     -     -     1     4     <NA>  <NA>     NA    NA    NA <NA>     NA
## 12 "Mas~ 12    10    -     -     22    <NA>  <NA>     NA    NA    NA <NA>     NA
## 13 "ATP~ 6     -     1     -     7     <NA>  <NA>     NA    NA    NA <NA>     NA
## 14 "ATP~ 3     1     2     3     9     <NA>  <NA>     NA    NA    NA <NA>     NA
## 15 "Jeu~ 0     -     1     -     1     <NA>  <NA>     NA    NA    NA <NA>     NA
```