Capítulo 9: Scraping the Web

Sara Abigail Hernández Sánchez

2025-04-25

```
rmarkdown::pandoc_available()
```

```
## [1] TRUE
```

Introducción

Este capítulo se centra en la recolección de datos en la práctica. Se abordan tres aspectos principales del scraping web con R

1. Recuperación de datos desde la Web

Se estudia cómo obtener datos desde diferentes escenarios.

2. Extracción de información desde los recursos recolectados

En esta parte se estudian estrategias para extraer información de los datos obtenidos.

Ciclo general del scraping

1. Identificación de la información
2. Elección de la estrategia
3. Recuperación de los datos
4. Extracción de la información
5. Preparación de los datos
6. Validación de los datos
7. Depuración y mantenimiento
8. Generalización

9. Comportamiento ético del scraper

Se discute cómo actuar de forma respetuosa al hacer scraping

4. Mirada al futuro

El capítulo termina con una visión sobre como mejorar las interfaces de R con datos web

Escenarios de Recuperación

Para los siguientes escenarios de recuperación de datos web, se usará el siguiente conjunto de paquetes de R.

```
library(RCurl)
```

```
## Warning: package 'RCurl' was built under R version 4.4.3
```

```
library(XML)
```

```
## Warning: package 'XML' was built under R version 4.4.2
```

```
library(stringr)
```

Descargar archivos listos para su uso

En ocaciones se podrán encontrar archivos listos para usar como TXT, CSV u otros formatos texto plano.

Datos de resultados electorales en formato CSV

Para obtener los archivos deseados se siguenn tres pasos:

1. Identificar los enlaces de los archivos deseados
2. Contruir una función de Descargar
3. Ejecutar las descargas

```
url <- "https://elections.maryland.gov/elections/2012/election_data/index.html"
html <- getURL(url)
links <- getHTMLLinks(html)
filenames <- links[str_detect(links, "_General.csv")]
filenames_list <- as.list(filenames)
filenames_list[1:3]
```

```
## [[1]]
## [1] "State_Congressional_Districts_2012_General.csv"
##
## [[2]]
## [1] "State_Legislative_Districts_2012_General.csv"
##
## [[3]]
## [1] "All_By_Precinct_2012_General.csv"
```

```
downloadCSV <- function(filename, baseurl, folder) {
    dir.create(folder, showWarnings = FALSE)
    fileurl <- str_c(baseurl, filename)
    if (!file.exists(str_c(folder, "/", filename))) {
        download.file(fileurl,
            destfile = str_c(folder, "/", filename))
        Sys.sleep(1)
    }
}
```

```
library(plyr)

l_ply(filenames_list, downloadCSV,
    baseurl = "www.elections.maryland.gov/elections/2012/election_data/",
    folder = "elec12_maryland")
```

```
length(list.files("./elec12_maryland"))
```

```
## [1] 70
```

```
list.files("./elec12_maryland")[1:3]
```

```
## [1] "All_By_Precinct_2012_General.csv"
## [2] "Allegany_By_Precinct_2012_General.csv"
## [3] "Allegany_County_2012_General.csv"
```

La tarea de web scarping se ha completado exitosamente, ahora se puede continuar con el análisis.

Mapas de distritos legislativos en formato PDF

```
url <- "https://planning.maryland.gov/Redistricting/Pages/2020/legiDist.aspx"
html <- getURL(url)
links <- getHTMLLinks(html)
filenames <- links[str_detect(links, "2020Maps/Leg/2022-Legislative-District")]
filenames_list <- str_extract_all(filenames, "District.+\\.pdf")
filenames_list[sapply(filenames_list, length) > 0][1:3]
```

```
## [[1]]
## [1] "Districts-Statewide.pdf"
##
## [[2]]
## [1] "District01.pdf"
##
## [[3]]
## [1] "District13.pdf"
```

Descargamos los datos sin procesar y se almacenan en un objeto llamado pdffile. Se escriben los datos en un archivo PDF en una carpeta especificada.

```
downloadPDF <- function(filename, baseurl, folder, handle) {
    dir.create(folder, showWarnings = FALSE)
    fileurl <- str_c(baseurl, filename)
    if (!file.exists(str_c(folder, "/", filename))) {
content <- getBinaryURL(fileurl, curl = handle)
        writeBin(content, str_c(folder, "/", filename))
    Sys.sleep(1)
    }
}
```

```
handle <- getCurlHandle(useragent = str_c(R.version$platform,
R.version$version.string, sep=", "), httpheader = c(from =
"eddie@datacollection.com"))
```

```
l_ply(filenames_list, downloadPDF,
    baseurl = "planning.maryland.gov/Redistricting/Documents/2020Maps/Leg/",
    folder = "elec12_maryland_maps",
    handle = handle)
```

Verificamos el número de elementos descargados e imprimimos los primeros resultados

```
length(list.files("./elec12_maryland_maps"))
```

```
## [1] 48
```

```
list.files("./elec12_maryland_maps")[1:3]
```

```
## [1] "District01.pdf" "District02.pdf" "District03.pdf"
```

Descarga de múltiples archivos desde un índice FTP

Los servidores FTP (File Transfer Protocol) , Protocolo de transferencia de archivos, sólo almacenan archivos sin necesidad de eliminar diseños HTML u otra información no deseada.

```
ftp <- "ftp://cran.r-project.org/pub/R/web/views/"
ftp_files <- getURL(ftp, dirlistonly = TRUE)
```

```
ftp_files
```

Dado que la información contiene saltos de línea y retornos de carro aplicaremos lo siguiente para poder quitarlos

```
filenames <- str_split(ftp_files, "\r\n")[[1]]
filenames_html <- unlist(str_extract_all(filenames, ".+(.html)"))
filenames_html[1:3]
```

```
## [1] "ActuarialScience.html" "Agriculture.html"      "Bayesian.html"
```

Una forma equivalente de obtener sólo los archivos HTML sería lo siguiente

```
filenames_html <- getURL(ftp, customrequest = "NLST *.html")
filenames_html = str_split(filenames_html, "\\\r\\\n")[[1]]
```

Esto devuelve una lista con los nombres de los archivos en el directorio FTP que termina en .html

Finalmente, construimos una función downloadFTP() que recupera los archivos deseados desde el servidor FTP y los almacena en una carpeta especifica.

```
downloadFTP <- function(filename, folder, handle) {
    dir.create(folder, showWarnings = FALSE)
    fileurl <- str_c(ftp, filename)
    if (!file.exists(str_c(folder, "/", filename))) {
        content <- try(getURL(fileurl, curl = handle))
        write(content, str_c(folder, "/", filename))
        Sys.sleep(1)
    }
}
```

Se configura un manejador que deshabilita el modo pasivo FTP-extended y descarga los documentos HTML de la tarea de CRAN

```
handle <- getCurlHandle(ftp.use.epsv = FALSE)
l_ply(filenames_html, downloadFTP,
    folder = "cran_tasks",
    handle = handle)
```

Finalmente se hace una inspección de los archivos descargados

```
length(list.files("./cran_tasks"))
```

```
## [1] 88
```

```
list.files("./cran_tasks")[1:3]
```

```
## [1] "ActuarialScience.html" "Agriculture.html"      "Bayesian.html"
```

Es posible cargar datos en un servidor FTP. El siguiente es un ejemplo de como se puede realizar.

```
ftpUpload(what = "example.txt", to = "ftp://example.com/",
userpwd = "username:password")
```

Manipulación de URLs para acceder a múltiples páginas

Se proponen cinco pasos para crear un web scraper:

1. Identificamos el mecanismo de paginación en la sintaxis de la URL.
2. Extraemos los enlaces a las páginas de resultados.
3. Descargamos las páginas de resultados.
4. Extraemos los enlaces a los comunicados dentro de las páginas de resultados.
5. Descargamos los comunicados individuales.

Usando rvest, httr, y purrr.

Generar URLs de las páginas

```
library(rvest)
```

```
## Warning: package 'rvest' was built under R version 4.4.3
```

```
library(httr)
library(purrr)
```

```
##
## Adjuntando el paquete: 'purrr'

## The following object is masked from 'package:plyr':
##
##     compact
```

```
library(dplyr)
```

```
##
## Adjuntando el paquete: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(stringr)

getPageURLs <- function(base_url, npages = 5) {
  urls <- paste0(base_url, "?page=", 1:npages)
  return(as.list(urls))
}

base_url <- "https://www.transparency.org/en/press"
urls_list <- getPageURLs(base_url, npages = 5)
head(urls_list, 3)
```

```
## [[1]]
## [1] "https://www.transparency.org/en/press?page=1"
##
```

```
## [[2]]
## [1] "https://www.transparency.org/en/press?page=2"
##
## [[3]]
## [1] "https://www.transparency.org/en/press?page=3"
```

Descargar y guardar cada página

```
dlPages <- function(pageurl, folder) {
  dir.create(folder, showWarnings = FALSE)
  page_num <- str_extract(pageurl, "page=\\d+")
  page_name <- ifelse(is.na(page_num), "base.html", paste0(page_num, ".html"))

  filepath <- file.path(folder, page_name)
  if (!file.exists(filepath)) {
    resp <- GET(pageurl, user_agent("Mozilla/5.0"))
    writeLines(content(resp, as = "text", encoding = "UTF-8"), filepath)
    Sys.sleep(1)
  }
}

walk(urls_list, dlPages, folder = "tp_index_2025")
list.files("tp_index_2025")[1:3]
```

```
## [1] "page=1.html" "page=2.html" "page=3.html"
```

Extraer enlaces de comunicados desde los archivos locales

```
getPressURLs <- function(folder) {
  files <- list.files(folder, full.names = TRUE)
  urls <- map(files, ~ {
    page <- read_html(.x)
    page %>%
      html_elements("article h2.sub-heading a") %>%
      html_attr("href")

  }) %>%
    unlist()

  return(as.list(unique(urls)))
}

press_urls_list <- getPressURLs("tp_index_2025")
length(press_urls_list)
```

```
## [1] 15
```

Descargar cada comunicado de prensa

```
dlPress <- function(press_url, folder) {
  dir.create(folder, showWarnings = FALSE)
  press_filename <- str_extract(press_url, "[^/]+$") %>% paste0(".html")
```

```
  filepath <- file.path(folder, press_filename)

  if (!file.exists(filepath)) {
    resp <- GET(press_url, user_agent("Mozilla/5.0"))
    writeLines(content(resp, as = "text", encoding = "UTF-8"), filepath)
    Sys.sleep(1)
  }
}

walk(press_urls_list, dlPress, folder = "tp_press_2025")
length(list.files("tp_press_2025"))
```

```
## [1] 16
```

```
print(list.files("tp_press_2025"))
```

```
##  [1] "2024-corruption-perceptions-index-authoritarianism-and-weakening-democracy-undermine-action-aga
##  [2] "2024-corruption-perceptions-index-authoritarianism-chokes-climate-action-in-the-middle-east-and
##  [3] "2024-corruption-perceptions-index-climate-funds-at-risk-of-theft-as-sub-saharan-africa-faces-s
##  [4] "2024-corruption-perceptions-index-corruption-fuels-environmental-crime-across-the-americas.html
##  [5] "2024-corruption-perceptions-index-corruption-playing-devastating-role-climate-crisis.html"
##  [6] "2024-corruption-perceptions-index-western-europe-sees-declining-score-at-a-critical-time.html"
##  [7] "amendment-to-constitution-judicature-act-risk-of-state-capture-maldives.html"
##  [8] "democracy-at-risk-georgia-moves-to-silence-civil-society.html"
##  [9] "gaps-anti-money-laundering-reforms-undermine-switzerlands-efforts-corruption-fight.html"
## [10] "new-index-reveals-alarming-opacity-and-money-laundering-vulnerabilities-global-real-estate.html
## [11] "serbian-students-biking-1300-km-strasbourg-defend-rule-of-law-must-receive-support-eu-commissi
## [12] "transparency-international-partners-call-immediate-action-end-high-polluters-lobbys-climate-ta
## [13] "united-states-cta-beneficial-ownership-reinstating-corporate-secrecy-protects-money-launderers
## [14] "us-pausing-of-foreign-bribery-enforcement-will-harm.html"
## [15] "venezuela-transparency-international-forced-into-exile-amid-growing-repression-of-civil-society
## [16] "www.transparency.org.html"
```

En muchos ejercicios de web scraping, podemos aplicar manipulaciones de URL para acceder a todos los sitios que nos interesan. La desventaja de este tipo de acceso es que se requiere un conocimiento bastante detallado del sitio web y de su estructura de directorios para poder realizar estas manipulaciones.

Funciones prácticas para recopilar enlaces, listas y tablas de documentos HTML

Ilustramossu funcionalidad con un artículo de Wikipedia sobre Nicolás Maquiavelo, un "historiador, político, diplomático, filósofo, humanista y escritor italiano" (Wikipedia 2014).

Extraer enlaces, tablas y listas de documentos HTML son tareas comunes en la práctica del web scraping. Se muestra un ejemplo con rvest.

```
library(XML)
library(rvest)

mac_url <- "http://en.wikipedia.org/wiki/Machiavelli"
```

```r
pagina <- read_html(mac_url)


enlaces <- pagina %>%
  html_elements("a") %>%
  html_attr("href")
head(enlaces, 3)
```

```
## [1] "#bodyContent"             "/wiki/Main_Page"
## [3] "/wiki/Wikipedia:Contents"
```

```r
xpath_img <- "//img[contains(@src, 'Machiavelli')]/@src"
imagenes <- pagina %>%
  html_elements(xpath = xpath_img) %>%
  html_attr("src")
head(imagenes, 3)
```

```
## [1] NA NA NA
```

```r
tablas <- pagina %>%
  html_elements("table") %>%
  html_table(fill = TRUE)


if(length(tablas) > 0) {
  persondata <- tablas[[1]]
  head(persondata)
}
```

```
## # A tibble: 6 x 2
##   `Niccolò Machiavelli`                     `Niccolò Machiavelli`
##   <chr>                                     <chr>
## 1 "Portrait by Santi di Tito, c. 1550-1600" "Portrait by Santi di Tito, c. 1550~
## 2 "Born"                                    "(1469-05-03)3 May 1469Florence, Re~
## 3 "Died"                                    "21 June 1527(1527-06-21) (aged 58)~
## 4 "Spouse"                                  ".mw-parser-output .marriage-line-m~
## 5 "Children"                                "7"
## 6 ""                                        ""
```

Manejo de formularios HTML

En la práctica del web scraping: 1. reconocer los formularios involucrados, 2. determinar el método utilizado para transferir los datos, 3. determinar la dirección a la que se enviarán los datos, 4. determinar las entradas que se deben enviar, 5. construir una solicitud válida y enviarla, 6. procesar los recursos devueltos.

Especificamos un curl handle con un conjunto de opciones por defecto: cookiejar para habilitar la gestión de cookies, followlocation para seguir redirecciones

```r
info <- debugGatherer()
handle <- getCurlHandle(cookiejar = "",
                        followlocation = TRUE,
```

```
                        autoreferer = TRUE,
                        debugfunc = info$update,
                        verbose = TRUE,
                        httpheader  = list(
                             from = "eddie@r-datacollection.com",
                             'user-agent' = str_c(R.version$version.string,
                                                  ", ", R.version$platform)
                        ))
```

Definir una función que traduzca listas de atributos XML en data frames.

```
xmlAttrsToDF <- function(parsedHTML, xpath) {
    x <- xpathApply(parsedHTML, xpath, xmlAttrs)
    x <- lapply(x, function(x) as.data.frame(t(x)))
    do.call(rbind.fill, x)
}
```

La función extrae los atributos de los nodos y transforma la lista resultante en un data frame, asegurándose de que los nombres de los atributos no se pierdan y que cada valor de atributo se almacene en una columna separada.

Comprendiendo los formularios con GET

Primero, cargamos la página del formulario en R y la analizamos (parseamos).

```
url <- "https://wordnet.princeton.edu/"
html_form <- getURL(url, curl = handle)
parsed_form <- htmlParse(html_form)
```

Revisemos atributos del nodo

para conocer los detalles específicos del envío de datos al servidor

```
xmlAttrsToDF(parsed_form, "//form")
```

```
##    action method                id accept-charset
## 1 /search    get search-block-form          UTF-8
```

Usamos la función xmlAttrsToDF() para obtener el conjunto completo de entradas y sus atributos.

```
xmlAttrsToDF(parsed_form, "//form[1]/input")
```

```
## NULL
```

Dado que el método HTTP para enviar los datos al servidor es GET, usamos la función getForm().

```
html_form_res <- getForm(uri = url, curl = handle, s = "data")
parsed_form_res <- htmlParse(html_form_res)
xpathApply(parsed_form_res, "//li", xmlValue)
```

```
## [[1]]
## [1] "\n                          What is WordNet\n              "
##
## [[2]]
## [1] "\n                          People\n                        \n        \n          \n        \n
##
## [[3]]
## [1] "\n                          George A. Miller\n             "
##
## [[4]]
## [1] "\n                          News\n              "
##
## [[5]]
## [1] "\n                          Use Wordnet Online\n             "
##
## [[6]]
## [1] "\n                          Download\n                        \n        \n          \n        \n
##
## [[7]]
## [1] "\n                          Current Version\n             "
##
## [[8]]
## [1] "\n                          Old Versions\n             "
##
## [[9]]
## [1] "\n                          Standoff Files\n             "
##
## [[10]]
## [1] "\n                          Citing WordNet\n             "
##
## [[11]]
## [1] "\n                          License and Commercial Use\n             "
##
## [[12]]
## [1] "\n                          Related Projects\n             "
##
## [[13]]
## [1] "\n                          Documentation\n                        \n        \n          \n
##
## [[14]]
## [1] "\n                          binsrch(3WN)\n             "
##
## [[15]]
## [1] "\n                          cntlist(5WN)\n             "
##
## [[16]]
## [1] "\n                          grind(1WN)\n             "
##
## [[17]]
## [1] "\n                          lexnames(5WN)\n             "
##
## [[18]]
## [1] "\n                          morph(3WN)\n             "
##
```

```
## [[19]]
## [1] "\n                          morphy(7WN)\n              "
##
## [[20]]
## [1] "\n                          prologdb(5WN)\n             "
##
## [[21]]
## [1] "\n                          senseidx(5WN)\n             "
##
## [[22]]
## [1] "\n                          sensemap(5WN)\n             "
##
## [[23]]
## [1] "\n                          uniqbeg(7WN)\n              "
##
## [[24]]
## [1] "\n                          wn(1WN)\n             "
##
## [[25]]
## [1] "\n                          wnb(1WN)\n             "
##
## [[26]]
## [1] "\n                          wndb(5WN)\n              "
##
## [[27]]
## [1] "\n                          wngloss(7WN)\n              "
##
## [[28]]
## [1] "\n                          wngroups(7WN)\n             "
##
## [[29]]
## [1] "\n                          wninput(5WN)\n              "
##
## [[30]]
## [1] "\n                          wnintro(1WN)\n              "
##
## [[31]]
## [1] "\n                          wnintro(3WN)\n              "
##
## [[32]]
## [1] "\n                          wnintro(5WN)\n              "
##
## [[33]]
## [1] "\n                          wnintro(7WN)\n              "
##
## [[34]]
## [1] "\n                          wnlicens(7WN)\n             "
##
## [[35]]
## [1] "\n                          wnpkgs(7WN)\n              "
##
## [[36]]
## [1] "\n                          wnsearch(3WN)\n             "
##
```

```
## [[37]]
## [1] "\n                          wnstats(7WN)\n                          \n        \n          \n
##
## [[38]]
## [1] "\n                          2.1 wnstats(7WN)\n              "
##
## [[39]]
## [1] "\n                          2.0 wnstats(7WN)\n              "
##
## [[40]]
## [1] "\n                          wnutil(3WN)\n              "
##
## [[41]]
## [1] "\n                          Publications\n              "
##
## [[42]]
## [1] "\n                          Frequently Asked Questions\n              "
##
## [[43]]
## [1] "\n               Log in\n              "
##
## [[44]]
## [1] "\n                          What is WordNet\n              "
##
## [[45]]
## [1] "\n                          People\n              "
##
## [[46]]
## [1] "\n                          News\n              "
##
## [[47]]
## [1] "\n                          Use Wordnet Online\n              "
##
## [[48]]
## [1] "\n                          Download\n              "
##
## [[49]]
## [1] "\n                          Citing WordNet\n              "
##
## [[50]]
## [1] "\n                          License and Commercial Use\n              "
##
## [[51]]
## [1] "\n                          Related Projects\n              "
##
## [[52]]
## [1] "\n                          Documentation\n              "
##
## [[53]]
## [1] "\n                          Publications\n              "
##
## [[54]]
## [1] "\n                          Frequently Asked Questions\n              "
##
```

```
## [[55]]
## [1] "\n      Diversity & Non-Discrimination\n    "
##
## [[56]]
## [1] "\n      Accessibility Help\n    "
```

Para ver los headers de la información que enviamos,

```
cat(str_split(info$value()["headerOut"], "\r")[[1]])
```

```
## GET / HTTP/1.1
## Host: wordnet.princeton.edu
## Accept: */*
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
## GET /?s=data HTTP/1.1
## Host: wordnet.princeton.edu
## Accept: */*
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
```

```
info$reset()
```

Se podría hacer lo mismo proporcionando una URL con la cadena de consulta añadida y realizando una llamada a getURL():

```
url <- "https://wordnet.princeton.edu/"
html_form_res <- getURL(url = url, curl = handle)
```

Enviando formularios con POST

Con POST, la información se transfiere en el cuerpo de la solicitud. Existen dos estilos comunes para transportar datos en el cuerpo: url-encoded o multipart.

POST con cuerpo codificado como url-encoded

Usamos el handle previamente configurado para recuperar la página, analizarla directamente y guardarla.

```
url <- "http://read-able.com/"
form <- htmlParse(getURL(url = url, curl = handle))
```

Búsqueda de nodos

```
xmlAttrsToDF(form, "//form")
```

```
##   method                                  class          name
## 1   POST                 url_option_selected          <NA>
## 2   POST                text_option_selected          <NA>
## 3   post                                 <NA> refer_website
```

```
## 4    GET fx-footer-quote-form form-quote-proposal          <NA>
##             action
## 1             <NA>
## 2             <NA>
## 3
## 4 /free-quote-alt/
```

```
xmlAttrsToDF(form, "//form[2]//input")
```

```
## NULL
```

```
xpathApply(form, "//form[2]")
```

```
## list()
## attr(,"class")
## [1] "XMLNodeSet"
```

Usamos una cita sobre los datos para comprobar la legitibilidad

```
sentence <- "\"It is a capital mistake to theorize before one has
data. Insensibly one begins to twist facts to suit theories, instead
of theories to suit facts.\"-- Arthur Conan Doyle, Sherlock Holmes"
```

Enviamos el texto al servidor de read-able para su evaluación. Dentro de la llamada a postForm(), establecemos style = "POST" para realizar una transmisión de los datos codificada como url-encoded.

```
res <- postForm(uri = str_c(url, "check.php"),
        curl = handle,
        style = "POST",
        directInput = sentence)
```

La mayoría de los resultados se presentan en forma de tablas HTML, como se muestra a continuación.

```
readHTMLTable(res)
```

```
## named list()
```

Veamos la información de headers que fue enviada al servidor.

```
cat(str_split(info$value()["headerOut"], "\r")[[1]])
```

```
## GET / HTTP/1.1
## Host: wordnet.princeton.edu
## Accept: */*
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
## GET / HTTP/1.1
## Host: read-able.com
## Accept: */*
```

```
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
## GET /tools/read-able/ HTTP/1.1
## Host: www.webfx.com
## Accept: */*
## Referer: http://read-able.com/
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
## POST /check.php HTTP/1.1
## Host: read-able.com
## Accept: */*
## Referer: http://read-able.com/
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
## Content-Length: 272
## Content-Type: application/x-www-form-urlencoded
##
## GET /tools/read-able/ HTTP/1.1
## Host: www.webfx.com
## Accept: */*
## Referer: http://read-able.com/check.php
## Cookie: __cf_bm=DoWHObUfocKRYllRzpv4emJLGo3bsNi5gOycl5OYpBg-1745626790-1.0.1.1-fF54dJcO4wssEfLM.IyKV
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
```

La segunda cabecera confirma que los datos fueron enviados a través de POST, utilizando el siguiente cuerpo codificado como url-encoded

```
cat(str_split(info$value()["dataOut"], "\r")[[1]])
```

```
## directInput=%22It%20is%20a%20capital%20mistake%20to%20theorize%20before%20one%20has%20%0Adata.%20Ins
```

```
info$reset()
```

Autenticación HTTP

No todos los lugares en la Web están accesibles, HTTP ofrece técnicas de autenticación que restringen el contenido a usuarios no autorizados.

En R, podemos pasar el nombre de usuario y la contraseña al servidor con la opción userpwd de libcurl.

```
url <- "www.r-datacollection.com/materials/solutions"
cat(getURL(url, userpwd = "teacher:sesame", followlocation = TRUE))
```

```
## <p>Hallo teacher.</p><p>Sie gaben sesame als Passwort ein.</p>solutions coming soon
```

```
options(RDataCollectionLogin = "teacher:sesame")
```

```
getURL(url, userpwd = getOption("RDataCollectionLogin"),
followlocation = TRUE)
```

```
## [1] "<p>Hallo teacher.</p><p>Sie gaben sesame als Passwort ein.</p>solutions coming soon"
```

Para evitar almacenar contraseñas directamente en el código, es conveniente guardarlas en el archivo

Conexiones mediante HTTPS

Para recuperar contenido de servidores a través de HTTPS, podemos usar libcurl/RCurl, que admiten conexiones SSL.

Conectarse a sitios HTTPS:

```
url <- "https://www.icpsr.umich.edu/icpsrweb/ICPSR/ssvd/search"
getURL(url)
```

```
## [1] "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2.0//EN\">\n<html><head>\n<title>302 Found</title>\n<
```

Debemos especificar manualmente la ruta del archivo y pasársela a la función de descarga mediante el argumento cainfo.

```
signatures = system.file("CurlSSL", cainfo = "cacert.pem", package = "RCurl")
res <- getURL(url, cainfo = signatures)
```

Podemos desactivar la verificación del servidor, aunque esto es riesgoso si el servidor no es confiable.

```
res <- getURL(url, ssl.verifypeer = FALSE)
```

Otra forma:

```
url_action <- "https://www.icpsr.umich.edu/icpsrweb/ICPSR/ssvd/ variables?"
handle <- getCurlHandle(cainfo = signatures)
res <- getForm(url_action,
                variableLabel = "climate+change",
                questionText = "",
                categoryLabel = "",
                curl = handle)
str_extract(res, "Your query returned [[:digit:]]+ variables")
```

Podríamos extraer más información sobre cada pregunta y consultar otros aspectos específicos.

Uso de cookies

Las cookies se utilizan para que los servidores HTTP puedan reconocer a los clientes, ya que HTTP es un protocolo sin estado, lo que significa que trata cada intercambio de solicitud y respuesta como si fuera el primero

```
info <- debugGatherer()
handle <- getCurlHandle(cookiejar = "",
                        followlocation = TRUE,
                        autoreferer = TRUE,
                        debugfunc = info$update,
                        verbose = TRUE,
                        httpheader = list(
                            from   = "eddie@r-datacollection.com",
                            'user-agent' = str_c(R.version$version.string,
                                                 ", ", R.version$platform)
                        ))
```

Con esto se activa la gestión de cookies, incluso si no se proporciona un nombre de archivo para almacenar las cookies.

Llenar un carrito de compras en línea

Aunque el soporte para cookies suele ser necesario para acceder a páginas web que requieren inicio de sesión, el siguiente ejemplo ilustra su uso en un carrito de compras de la librería Biblio.

Definimos las URLs para la búsqueda y el carrito

```
search_url <- "www.biblio.com/search.php?keyisbn=data"
cart_url <- "www.biblio.com/cart.php"
```

Descargamos y analizamos la página de resultados de búsqueda

```
search_page <- htmlParse(getURL(url = search_url, curl = handle))
```

Agregar libros al carrito se hace mediante formularios HTML

```
xpathApply(search_page, "//div[@class='order-box'][position()<2]/ form")
```

```
## NULL
```

Extraemos los identificadores (bid) de los libros

```
xpath <- "//div[@class='order-box'][position()<4]/form/input [@name='bid']/@value"
bids <- unlist(xpathApply(search_page, xpath, as.numeric))
bids
```

```
## NULL
```

Ahora añadimos los tres primeros libros al carrito

```
for (i in seq_along(bids)) {
    res <- getForm(uri = cart_url, curl = handle, bid = bids[i],
add = 1, int = "keyword_search")
}
```

Recuperamos el contenido del carrito y verificamos los artículos almacenados

```
cart <- htmlParse(getURL(url = cart_url, curl = handle))
clean <- function(x) str_replace_all(xmlValue(x), "(\t)|(\n\n)", "")
cat(xpathSApply(cart, "//div[@class='title-block']", clean))
```

Luego revisamos los headers enviados y recibidos. La primera solicitud no contenía cookies

```
cat(str_split(info$value()["headerOut"], "\r")[[1]][1:13])
```

```
## GET /search.php?keyisbn=data HTTP/1.1
## Host: www.biblio.com
## Accept: */*
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
## GET /cart.php HTTP/1.1
## Host: www.biblio.com
## Accept: */*
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
```

```
cat(str_split(info$value()["headerIn"], "\r")[[1]][1:14])
```

```
## HTTP/1.1 403 Forbidden
## Date: Sat, 26 Apr 2025 00:19:53 GMT
## Content-Type: text/html; charset=UTF-8
## Content-Length: 5582
## Connection: close
## accept-ch: Sec-CH-UA-Bitness, Sec-CH-UA-Arch, Sec-CH-UA-Full-Version, Sec-CH-UA-Mobile, Sec-CH-UA-Mo
## cf-mitigated: challenge
## critical-ch: Sec-CH-UA-Bitness, Sec-CH-UA-Arch, Sec-CH-UA-Full-Version, Sec-CH-UA-Mobile, Sec-CH-UA-
## cross-origin-embedder-policy: require-corp
## cross-origin-opener-policy: same-origin
## cross-origin-resource-policy: same-origin
## origin-agent-cluster: ?1
## permissions-policy: accelerometer=(),autoplay=(),browsing-topics=(),camera=(),clipboard-read=(),clipb
## referrer-policy: same-origin
```

```
cat(str_split(info$value()["headerOut"], "\r")[[1]][1:13])
```

```
## GET /search.php?keyisbn=data HTTP/1.1
## Host: www.biblio.com
## Accept: */*
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
## GET /cart.php HTTP/1.1
## Host: www.biblio.com
## Accept: */*
## from: eddie@r-datacollection.com
## user-agent: R version 4.4.1 (2024-06-14 ucrt), x86_64-w64-mingw32
##
```

Si no incluimos las cookies, el carrito queda vacío. Para comprobarlo, repetimos la solicitud pero reiniciando las cookies

```
cart <- htmlParse(getURL(url = cart_url, curl = handle, cookielist = "ALL"))
clean <- function(x) str_replace_all(xmlValue(x), "(\t)|(\n\n)", "")
cat(xpathSApply(cart, "//div[@class='title-block']", clean))
```

Recuperando datos desde APIs

Las APIs son herramientas que permiten a los programadores conectar su software con "algo más", facilitando la integración con software o hardware externo sin necesidad de comprender todos los detalles técnicos.

Proceso básico para recuperar datos de APIs es simple: el proveedor de la API configura un servicio que otorga acceso a datos de la aplicación, y el usuario accede a estos datos.

```
feed_url<- "http://weather.yahooapis.com/forecastrss"
feed<- getForm(feed_url,.params= list(w= "2422673",u= "c"))
```

Como el RSS recuperado es contenido XML, podemos analizarlo con la función de XML.

```
parsed_feed<-xmlParse(feed)
```

Extraer los valores de los parámetros meteorológicos actuales que se almacenan como atributos.

```
xpath<-"//yweather:location|//yweather:wind|//yweather:condition"
conditions<-unlist(xpathSApply(parsed_feed,xpath,xmlAttrs))
data.frame(conditions)
```

Autenticación con OAuth

La autenticación se usa para identificar a los usuarios y controlar el acceso a datos. OAuth es un estándar de autorización que permite a aplicaciones de terceros acceder a información del usuario sin que este tenga que revelar su nombre de usuario o contraseña.

El proceso de autorización con OAuth usa tres tipos de credenciales - Credenciales de cliente (clave y secreto de la app), - Credenciales temporales (token de solicitud), - Token de acceso, usado para futuras solicitudes sin necesidad de contraseña.

Por ejemplo, para acceder a la API de Facebook

```
facebook <- oauth_endpoint(
    authorize = "https://www.facebook.com/dialog/oauth",
    access = "https://graph.facebook.com/oauth/access_token")
```

```
fb_app <- oauth_app("facebook", "485980054864321")
```

```
permissions <- "user_birthday, user_hometown, user_location,
user_status, user_checkins, friends_birthday, friends_hometown,
friends_location, friends_relationships, friends_status, friends_ checkins,
publish_actions, read_stream, export_stream"
```

```
fb_token <- oauth2.0_token(facebook, fb_app, scope = permissions,
type = "application/x-www-form-urlencoded")

fb_sig <- sign_oauth2.0(fb_token)

getUsers("hadleywickham", fb_sig, private_info = FALSE)

friends <- getFriends(fb_sig, simplify = TRUE)
nrow(friends)
table(friends_info$gender)
```

Estrategias de extracción

Existen tres estrategias principales para recolectar datos de la web:

1. Scraping con expresiones regulares Consiste en identificar patrones generales en el contenido HTML o XML, descargar los sitios web, importar el contenido como texto, desarrollar expresiones regulares para extraer la información y depurar el código para corregir errores como falsos positivos o negativos.

- Ventajas: útil para datos con patrones repetitivos, eficiente, resistente a HTML mal formado.
- Desventajas: poco contexto, difícil de mantener, frágil ante cambios en la estructura.
- Ideal para: tareas de limpieza o como complemento a otras técnicas.

2. Consultas XPath XPath es un lenguaje que permite navegar por la estructura jerárquica de documentos XML y HTML. Esta técnica es más robusta y precisa que las expresiones regulares cuando se trabaja con páginas estructuradas correctamente.

- Ventajas: accede al contenido de manera estructurada (DOM), permite consultas contextuales y complejas, es más robusto ante pequeños cambios.
- Desventajas: requiere HTML bien formado, curva de aprendizaje más alta, puede ser más lento.
- Herramientas en R: xml2, rvest, xml_find_all().

3. Uso de APIs Las APIs son la forma más recomendada y profesional de obtener datos desde servicios web,están diseñadas específicamente para el acceso automatizado a datos, lo que las hace más robustas, eficientes y éticas.

- Ventajas: acceso directo y legal a datos estructurados (JSON/XML), más rápido y estable, no depende del diseño visual del sitio.
- Desventajas: requiere claves de acceso, tiene límites de uso, acceso restringido a los datos disponibles en la API.
- Herramientas en R: httr::GET(), httr::POST(), jsonlite::fromJSON().

Se recomienda priorizar estas técnicas en este orden, aunque en la práctica a menudo se combinan según las necesidades del proyecto.

Web scraping: buenas prácticas

¿Es legal el web scraping?

El scraping implica copiar contenido, lo que puede acarrear problemas legales por infracción de derechos de autor. Aunque se recomienda actuar con transparencia y documentar siempre las fuentes, no existe una línea clara entre lo que es legal y lo que no.

- Ambigüedad legal No hay leyes universales claras; depende del país, el propósito y el impacto del scraping.
- Casos clave
  - *eBay vs. Bidder's Edge*: acceso masivo, fallo a favor de eBay por sobrecarga del servidor.
  - *AP vs. Meltwater*: fallo en contra por uso comercial sin licencia.
  - *Facebook vs. Pete Warden*: incluso respetando `robots.txt`, se exigió eliminar los datos.
  - *Estados Unidos vs. Aaron Swartz*: escargó masivamente artículos científicos de JSTOR.

Evitar scraping para fines comerciales sin permiso. Ser transparente, respetar las normas del sitio, y minimizar el impacto en los servidores.

robots.txt

El archivo robots.txt es parte del Robots Exclusion Protocol, un estándar no oficial creado en 1994 por Martijn Koster. Se encuentra en la raíz de un sitio web y establece reglas para los bots sobre qué partes del sitio pueden o no ser rastreadas. No es legalmente vinculante, pero es una buena práctica respetarlo.

- Ejemplo de estructura:

```
aintext
User-agent: *
Disallow: /private/
```

- Alternativa en HTML: `<meta name="robots" content="noindex,nofollow">`
- En R: se puede usar `robotsCheck()` para verificar restricciones.

Buenas prácticas para scraping

- Usar APIs oficiales cuando estén disponibles.
- Respetar `robots.txt` y términos de uso.
- Identificarse en las solicitudes (User-Agent, correo en `From`).
- Minimizar tráfico: usar compresión, `If-Modified-Since`, y almacenar copias locales.

Fuentes de inspiración y recursos útiles

Guía para quienes desean comenzar con proyectos de scraping y buscar inspiración en lo que otros ya han hecho

- CRAN Task View sobre tecnologías web: ofrece una visión general de paquetes en R para scraping y APIs.

- GitHub: proyectos y paquetes útiles; se puede instalar con `devtools::install_github()`.
- rOpenSci: facilita el acceso a APIs científicas (paquetes como `rgbif`, `RMendeley`, `rfishbase`).
- Omega Project: herramientas como `RCurl` y `XML` para comunicación web.
- Libro recomendado: *Nolan y Temple Lang (2014)* sobre tecnologías web con R.