Sesión 2. Lectura de datos II

Curso: POL304 - Estadística para el análisis político 2

Jefes de práctica: Wendy Adrianzén y Alexander Benites

Ciclo 2022-2

Existe una web para máquinas y otra para humanos (el mundo del HTML, lo que todos usamos a diario al acceder a páginas web en nuestro ordenador). Hoy vamos a aprender a bajar información de ambas fuentes de datos.

Hoy aprenderemos a descargar datos desde servicios web que proporcionan información a través de "APIs" que son consultadas directamente por ordenadores. Asimismo, aprenderemos a bajar información programáticamente de las páginas web usando las técnicas de "scraping".

1. Uso de los APIs

Video sobre qué son las APIs: https://www.youtube.com/watch?v=rq6gdwEbowU

Hay organizaciones que tienen una política de datos abiertos, por lo que ofrecen un portal ad-hoc para que se pueda acceder a sus datos, controlando el historial de algunos datos que se incrementa continuamente (diariamente, semanalmente, etc.).

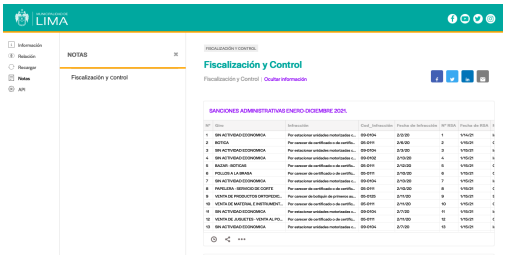
Estos portales cuentan con APIs, que significa "Aplication Programming Interface" y se puede entender como un mecanismo que nos permite interactuar con un servidor de internet, y construir pedidos de datos a través de una dirección web, de tal manera que podamos acceder a la información en tiempo real, visualizar el historial y sus actualizaciones.

Las APIs facilitan mucho la recopilación de datos al poderse acceder a ellas de forma programática ya que proveen de un proceso de acceso a ellos estandarizado: se envía una "http request" a la API y se reciben los datos en un determinado formato, generalmente JSON o XML. Vamos a ver en este curso, cómo obtener datos de una API en formato JSON.

Ejemplo

Vamos a hacer un ejemplo, extrayendo datos del portal "Datos Abiertos" de la Municipalidad Metropolitana de Lima. Entremos al siguiente link y exploremos el portal brevemente: https://aplicativos.munlima.gob.pe/extranet/datos-abiertos/

En la categoría "Fiscalización y control", podemos encontrar data sobre las sanciones administrativas realizadas por la MML en el mes de diciembre del año pasado.



Si le das click a "Obtén tu API key aquí", te dirigirá a una web que indica cómo hacer uso del API: https://datosabiertos.munlima.gob.pe/developers/

Vamos a ejecutar estos pasos en R.

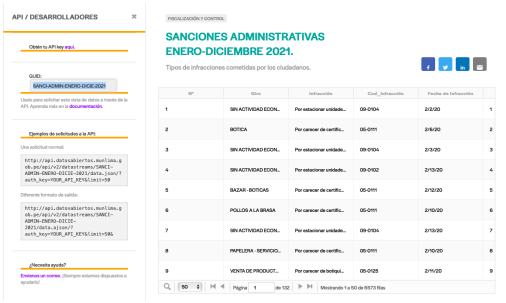
- Lo primero que te sugiere el API es que obtengas tu llave (API_KEY). Consigue dando click donde indica, y guardala como objeto.

```
miLLAVE="oKIs8cTkNuby9sDsbmLMPr27ulIqY43auxiEDnnw"
```

- Luego, te indica que construyas una solicitud (request) para colectar los datos. Según lo que nos dice la página, la estructura del request está conformada por:

El link del api de la página + el request para invocar que retornará en json + el GUID + el formato + mi clave (la que guardamos en el paso anterior).

NOTA: El link lo obtenemos de la siguiente manera:



Procedemos entonces a crear nuestro request. Vamos a crear el link en objetos en función de los componentes mencionados anteriormente. Esto es solo para fines didácticos, en el futuro ustedes pueden armar un link completo en sus notas y utilizarlo como request.

```
link="http://api.datosabiertos.munlima.gob.pe"
RQJSON="/api/v2/datastreams/"
GUID="SANCI-ADMIN-ENERO-MARZO-2022"
FORMATO="/data.json/"
KEY="?auth_key="
```

Entonces, tu solicitud se arma así:

```
request=paste0(link,RQJSON,GUID,FORMATO,KEY,miLLAVE) # La función paste0 la usamos para concatenar todo
request #mirala
```

```
## [1] "http://api.datosabiertos.munlima.gob.pe/api/v2/datastreams/SANCI-ADMIN-ENERO-MARZO-2022/data.js
```

- Finalmente, procesamos nuestra respuesta.

R necesita que instales jsonlite para poder interpretar formato JSON; luego de hacerlo, habilítala. De aquí, ya podrías colectar la data:

```
library(jsonlite)
MML = fromJSON(request)
str(MML)
```

```
## List of 20
##  $ result          :List of 6
##   ..$ fLength   : int 1768
##   ..$ fType     : chr "ARRAY"
##   ..$ fTimestamp: num 1.66e+12
##   ..$ fArray    :'data.frame':   9000 obs. of  3 variables:
##   .. ..$ fStr   : chr [1:9000] "N°" "Giro" "Infracción" "Cod_Infracción" ...
##   .. ..$ fHeader: logi [1:9000] TRUE TRUE TRUE TRUE TRUE TRUE ...
##   .. ..$ fType  : chr [1:9000] "TEXT" "TEXT" "TEXT" "TEXT" ...
##   ..$ fRows     : int 1000
##   ..$ fCols     : int 9
##  $ status          : int 3
##  $ description     : chr "Tipos de infracciones cometidas por los ciudadanos."
##  $ parameters      : list()
##  $ tags            : chr [1:14] "sanciones" "control" "fiscalizacion" "infracciones" ...
##  $ last_revision_id: int 286373
##  $ timestamp       : NULL
##  $ created_at      : chr "2022-05-11T19:24:08Z"
##  $ title           : chr "SANCIONES ADMINISTRATIVAS ENERO - MARZO 2022"
##  $ modified_at     : chr "2022-05-12T17:45:32Z"
##  $ category_id     : int 84236
##  $ methods         : NULL
##  $ sources         : list()
##  $ total_revisions : int 1
##  $ frequency       : chr "quarterly"
##  $ link            : NULL
##  $ user            : chr "alan.coello"
##  $ status_str      : NULL
##  $ guid            : chr "SANCI-ADMIN-ENERO-MARZO-2022"
##  $ category_name   : chr "Fiscalización y Control"
```

Si visualizas el objeto creado, verás que este pedido no se parece a lo que necesitas, ya que la estructura nos indica una serie de listas.

Para corregir esto, la documentación de API (https://junar.github.io/docs/es/), nos dice que podemos pedir otro formato para la vista de datos, el "PJSON":

```
FORMATO='/data.pjson/'
request2=paste0(link,RQJSON,GUID,FORMATO,KEY,miLLAVE)
```

Usemos la nueva solicitud:

```
MML = fromJSON(request2)
```

Veamos su estructura:

```
str(MML)
```

```
## List of 20
##  $ result          :'data.frame':    1000 obs. of  11 variables:
##   ..$ Cod_Infraccion     : chr [1:1000] "10-0137" "10-0101" "02-0302" "08-0205" ...
##   ..$ Fecha-de-RSA       : chr [1:1000] "1/3/22" "1/3/22" "1/3/22" "1/3/22" ...
##   ..$ Medida-Correctiva  : chr [1:1000] "Retiro" "Paralización y/o Demolición." "Sin medida correcti
##   ..$ N                  : chr [1:1000] "1" "2" "3" "4" ...
##   ..$ Infraccion         : chr [1:1000] "Por exhibir publicidad y/o propaganda electoral en un inmue
##   ..$ N-RSA              : chr [1:1000] "1" "2" "3" "4" ...
##   ..$ Giro               : chr [1:1000] "ORGANIZACION POLITICA" "SIN ACTIVIDAD ECONÓMICA" "CEVICHERI
##   ..$ Lineas             : chr [1:1000] "CENTRO HISTÓRICO DE LIMA" "CENTRO HISTÓRICO DE LIMA" "SALUD
##   ..$ Fecha-de-Infraccion: chr [1:1000] "4/7/21" "4/16/21" "4/16/21" "4/14/21" ...
##   ..$ timestamp          : num [1:1000] NA NA NA NA NA NA NA NA NA NA ...
##   ..$ length             : int [1:1000] NA NA NA NA NA NA NA NA NA NA ...
##  $ status          : int 3
##  $ description     : chr "Tipos de infracciones cometidas por los ciudadanos."
##  $ parameters      : list()
##  $ tags            : chr [1:14] "sanciones" "control" "fiscalizacion" "infracciones" ...
##  $ last_revision_id: int 286373
##  $ timestamp       : NULL
##  $ created_at      : chr "2022-05-11T19:24:08Z"
##  $ title           : chr "SANCIONES ADMINISTRATIVAS ENERO - MARZO 2022"
##  $ modified_at     : chr "2022-05-12T17:45:32Z"
##  $ category_id     : int 84236
##  $ methods         : NULL
##  $ sources         : list()
##  $ total_revisions : int 1
##  $ frequency       : chr "quarterly"
##  $ link            : NULL
##  $ user            : chr "alan.coello"
##  $ status_str      : NULL
##  $ guid            : chr "SANCI-ADMIN-ENERO-MARZO-2022"
##  $ category_name   : chr "Fiscalización y Control"
```

Para acceder a los datos en formato tabla de esa solicitud debemos acceder al elemento result y convertirlo en data frame:

```
dataMML=data.frame(MML$result)
head(dataMML)
```

```
##   Cod_Infraccion Fecha.de.RSA
## 1        10-0137       1/3/22
## 2        10-0101       1/3/22
## 3        02-0302       1/3/22
## 4        08-0205       1/3/22
## 5        03-0302       1/3/22
## 6        10-0101       1/3/22
##                                                                  Medida.Correctiva N
## 1                                                                          Retiro 1
## 2                                                 Paralización y/o Demolición. 2
## 3                                                        Sin medida correctiva 3
## 4                                                        Retiro y/o demolición 4
## 5 Clausura Definitiva y/o Paralización por treinta (30) días y/o Retención. 5
## 6                                                 Paralización y/o Demolición. 6
##
## 1
## 2
## 3
## 4                                                                    Por instalar, cercar y/o
## 5 Por no respetar la ejecución de la medida de carácter provisional o correctiva (paralización, claus
## 6
##   N.RSA                                  Giro                   Lineas
## 1     1                 ORGANIZACION POLITICA CENTRO HISTÓRICO DE LIMA
## 2     2              SIN ACTIVIDAD ECONÓMICA CENTRO HISTÓRICO DE LIMA
## 3     3           CEVICHERIAS - RESTAURANTES       SALUD Y SALUBRIDAD
## 4     4              SIN ACTIVIDAD ECONÓMICA                URBANISMO
## 5     5 VENTA MAY. MAQUINARIAS - EQUIPO Y MATER.    MORAL Y ORDEN PÚBLICO
## 6     6        TRANSPORTE DE CARGA POR CARRETERA CENTRO HISTÓRICO DE LIMA
##   Fecha.de.Infraccion timestamp length
## 1              4/7/21        NA     NA
## 2             4/16/21        NA     NA
## 3             4/16/21        NA     NA
## 4             4/14/21        NA     NA
## 5             4/12/21        NA     NA
## 6              4/7/21        NA     NA
```

Con esta ruta vamos a poder obtener los datos en tiempo real, de las posibles actualizaciones que la MML haga sobre sus acciones administrativas!

Ahora, alguien que nos apoye con otro ejemplo utilizando data de la OEFA con este link: https://datosabiertos.oefa.gob.pe/home

2. 'Scraping' tablas de datos

Video sobre qué es el scrapping: "What is Web Scraping and What is it Used For?" https://www.youtube.com/watch?v=Ct8Gxo8StBU

Ahora aprendamos a bajar información programáticamente de las páginas web usando las técnicas de "scraping".

En la vida real, no siempre nos darán las bases de datos listas para trabajar con ella,en muchos casos debemos descargarlas de páginas web y debemos limpiar y ordenar nuestra bbdd.

Librerias necesarias (si no las tiene, instale)

Opción 1

Vamos a ver como extraer tablas que están en una web, es decir no son archivos colgados, sino tablas que han sido dibujadas por el programador.

Las tablas de datos en la web pueden ser descargadas con facilidad, si se consigue identificar la ruta hacia ella. Cuando identifiques una tabla que te interesa, usa el botón derecho de tu mouse para inspeccionar el código de la página web. Usa la opción inspección hasta que resalte toda la tabla.

Vamos a extraer la tabla de los resultados del "Democracy Index" por país, de la siguiente dirección: https://en.wikipedia.org/wiki/Democracy_Index

Estando en esa página usando **GoogleChrome** ubícate en la tabla e inspecciona usando el boton derecho:

Debes inspeccionar hasta que se resalte tu tabla:

Nota que en este caso tienes varias tablas, debes ser muy preciso en tu selección. Una vez haya identificado bien tu tabla, usa nuevamente el botón derecho sobre el código en html y copia el full XPATH.

En **firefox** el procedimiento es el mismo.

Para extraer nuestra base de datos, utilizamos la función "htmltab":

```
library(htmltab)
```

```
link = "https://en.wikipedia.org/wiki/Democracy_Index"
path = "/html/body/div[3]/div[3]/div[5]/div[1]/table[5]"
dataWS = htmltab(link, path)
head(dataWS)
```

```
##           Region 2021 rank        Country      Regime type 2021 2020 2019 2018
## 2  North America        12         Canada   Full democracy 8.87 9.24 9.22 9.15
## 3  North America        26  United States Flawed democracy 7.85 7.92 7.96 7.96
## 4 Western Europe        21        Austria   Full democracy 8.07 8.16 8.29 8.29
## 5 Western Europe        36        Belgium Flawed democracy 7.51 7.51 7.64 7.78
## 6 Western Europe        37         Cyprus Flawed democracy 7.43 7.56 7.59 7.59
## 7 Western Europe         6        Denmark   Full democracy 9.09 9.15 9.22 9.22
##   2017 2016 2015 2014 2013 2012 2011 2010 2008 2006
## 2 9.15 9.15 9.08 9.08 9.08 9.08 9.08 9.08 9.07 9.07
## 3 7.98 7.98 8.05 8.11 8.11 8.11 8.11 8.18 8.22 8.22
## 4 8.42 8.41 8.54 8.54 8.48 8.62 8.49 8.49 8.49 8.69
## 5 7.78 7.77 7.93 7.93 8.05 8.05 8.05 8.05 8.16 8.15
## 6 7.59 7.65 7.53 7.40 7.29 7.29 7.29 7.29 7.70 7.60
## 7 9.22 9.20 9.11 9.11 9.38 9.52 9.52 9.52 9.52 9.52
```

Otro ejemplo:

De la página web de la CIA, vamos a extraer una tabla sobre el PBI percápita: https://www.cia.gov/the-world-factbook/field/real-gdp-per-capita/country-comparison

Procedemos:

```
linkCIA_pbi = "https://www.cia.gov/the-world-factbook/field/real-gdp-per-capita/country-comparison"
linkPath_pbi='/html/body/div/div[1]/div[2]/main/section[2]/div/div/div[2]/div/div/div/div/div/table'
dataWS2 = htmltab(linkCIA_pbi, linkPath_pbi)
```

```
## No encoding supplied: defaulting to UTF-8.
```

```
head(dataWS2)
```

```
##   Rank       Country       V1 Date of Information
## 2    1 Liechtenstein $139,100           2009 est.
## 3    2        Monaco $115,700           2015 est.
## 4    3    Luxembourg $110,300           2020 est.
## 5    4     Singapore  $93,400           2020 est.
## 6    5       Ireland  $89,700           2020 est.
## 7    6         Qatar  $85,300           2020 est.
```

Opción 2

Vamos a ver otra forma de "scrapear", utilizando el paquete "rvest" (instalenlo antes). ¿Qué pasa si los datos que quiero obtener no están en un formato de tabla? Es decir no han sido programados en la página web de esa forma.

Veamos este caso: https://www.congreso.gob.pe/?K=113

Aquí encontramos una tabla. Bueno, al menos podemos visualizarla así, sin embargo, si inspeccionamos los códigos de la web, no vamos a encontrar un "path" como en el ejemplo anterior con el cual podamos scrapear esta tabla. Esto se debe a la forma en la que está diseñada la web.

Más aún, qué pasaría si quisiéramos extraer datos de una página web como esta: https://www.congreso.gob.pe/?K=113

Para estas ocasiones, **rvest** nos sirve.

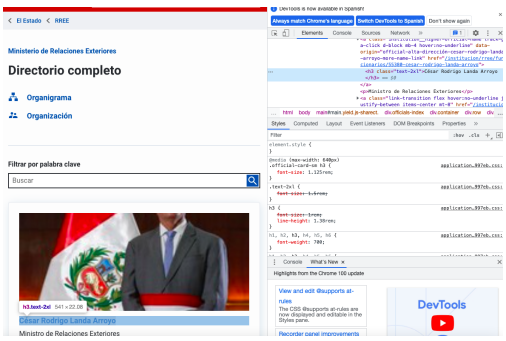
- Primero, abrimos la libreria y creamos un objeto con el link de la página web. Asimismo, obtenemos el código html de la web, con la función "read_html". Esto lo ponemos dentro de un objeto.

```
library(rvest)
url="https://www.gob.pe/institucion/rree/funcionarios"
pagina_web=read_html(url)
```

- Para scrapear vamos a requerir obtener la clase CSS de los campos que necesitamos. En términos sencillos, la clase CSS es un código que identifica a uno o varios elementos HTML. En esta página web, "Nombre" y "Cargo" han sido programados con una clase CSS para cada una.

Esta clase CSS la podemos obtener de la siguiente manera. Comencemos por Nombre

- Hacemos click derecho sobre el renglón de un país y seleccionamos inspeccionar:
- Luego, automaticamente en la parte derecha se nos abrirán los códigos resaltando el que pertenece a ese espacio, colocamos nuestro mouse por encima de esa linea resaltada del código y se nos aparecerá la clase CSS. Ese código lo escribimos copiamos.



Con ese código, continuamos de la siguiente manera:

```
css_nombre="h3.text-2xl" # contenemos la clase CSS en un objeto
nombre_html <- html_nodes(pagina_web,css_nombre) # con html_nodes y html_text, obtenemos el código html
nombre_texto <- html_text(nombre_html)
head(nombre_texto) #vemos los datos
```

```
## [1] "Miguel Ángel Rodríguez Mackay"
## [2] "Ana Cecilia Gervasi Díaz"
## [3] "Ana Rosa Valdivieso Santa Maria"
## [4] "Carlos Alberto Chocano Burga"
## [5] "Carlos Herrera Rodríguez"
## [6] "María Teresa Merino Villarán De Hart"
```

Hacemos lo mismo para la otra columna "cargo"

```
css_cargo="p"
cargo_html <- html_nodes(pagina_web,css_cargo)
cargo_texto <- html_text(cargo_html)
head(cargo_texto)
```

```
## [1] "Ministro de Relaciones Exteriores"
## [2] "Viceministra de Relaciones Exteriores"
## [3] "Secretaria General"
## [4] "Director General de América"
## [5] "Director General para Asuntos Culturales"
## [6] "Directora General de Europa"
```

Finalmente, armamos la base de datos

```
dataWS3 <- data.frame(NOMBRE = nombre_texto, CARGO = cargo_texto)
head(dataWS3)
```

```
##                                NOMBRE                                   CARGO
## 1        Miguel Ángel Rodríguez Mackay        Ministro de Relaciones Exteriores
## 2             Ana Cecilia Gervasi Díaz    Viceministra de Relaciones Exteriores
## 3      Ana Rosa Valdivieso Santa Maria                      Secretaria General
## 4         Carlos Alberto Chocano Burga             Director General de América
## 5             Carlos Herrera Rodríguez Director General para Asuntos Culturales
## 6 María Teresa Merino Villarán De Hart             Directora General de Europa
```

Hagamos otro ejemplo:

```
url="https://www.congreso.gob.pe/?K=113"
pagina_web=read_html(url)
```

```
css_nombre="a.conginfo" # contenemos la clase CSS en un objeto
nombre_html <- html_nodes(pagina_web,css_nombre) # con html_nodes y html_text, obtenemos el código html
nombre_texto <- html_text(nombre_html)
head(nombre_texto) #vemos los datos
```

```
## [1] "Acuña Peralta María Grimaneza"       "Acuña Peralta Segundo Héctor"
## [3] "Agüero Gutiérrez María Antonieta"    "Aguinaga Recuenco Alejandro Aurelio"
## [5] "Alcarraz Aguero Yorel Kira"          "Alegría García Arturo"
```

```
css_grupo="span.partidolist"
grupo_html <- html_nodes(pagina_web,css_grupo)
grupo_texto <- html_text(grupo_html)
head(grupo_texto)
```

```
## [1] "ALIANZA PARA EL PROGRESO" "INTEGRIDAD Y DESARROLLO"
## [3] "PERÚ LIBRE"               "FUERZA POPULAR"
## [5] "INTEGRIDAD Y DESARROLLO"  "FUERZA POPULAR"
```

```
dataWS4 <- data.frame(NOMBRE = nombre_texto, GRUPO = grupo_texto)
head(dataWS4)
```

```
##                               NOMBRE                   GRUPO
## 1       Acuña Peralta María Grimaneza ALIANZA PARA EL PROGRESO
## 2        Acuña Peralta Segundo Héctor  INTEGRIDAD Y DESARROLLO
## 3    Agüero Gutiérrez María Antonieta               PERÚ LIBRE
## 4 Aguinaga Recuenco Alejandro Aurelio           FUERZA POPULAR
## 5          Alcarraz Aguero Yorel Kira  INTEGRIDAD Y DESARROLLO
## 6               Alegría García Arturo           FUERZA POPULAR
```

Uno más complejo: la página de funcionarios/as del MEF

https://www.gob.pe/institucion/mef/funcionarios

```
url="https://www.gob.pe/institucion/mef/funcionarios"
pagina_web=read_html(url)


css_nombre="h3.text-2xl"
nombre_html <- html_nodes(pagina_web,css_nombre)
nombre_texto <- html_text(nombre_html)

css_cargo="p"
cargo_html <- html_nodes(pagina_web,css_cargo)
cargo_texto <- html_text(cargo_html)

dataWS5 <- data.frame(NOMBRE = nombre_texto, CARGO = cargo_texto)
head(dataWS5)
```

```
##                             NOMBRE
## 1        Kurt Johnny Burneo Farfan
## 2 José Armando Calderón Valenzuela
## 3    Alex Alonso Contreras Miranda
## 4    Kitty Elisa Trinidad Guerrero
## 5    Jaime Florencio Reyes Miranda
## 6        Hernán Martín Díaz Huamán
##                                         CARGO
## 1                Ministro de Economía y Finanzas
## 2                       Viceministro de Hacienda
## 3                       Viceministro de Economía
## 4                             Secretaria General
## 5                   Jefe de Gabinete de Asesores
## 6 Jefe del Órgano de Control Institucional - OCI
```

Pero nos faltan funcionarios/as. Solo estamos jalando la primera página de tres. ¿Qué hacemos aquí?

Para esto vamos a aprender a utilizar funciones e iteraciones, que nos permita descargar las páginas de manera sistemática.

Una función es un conjunto de instrucciones que convierten las entradas (inputs) en resultados (outputs) deseados. Las iteraciones (loops) son de gran utilidad cuando necesitamos hacer la misma tarea con multiples entradas; repetir la misma operación en diferentes columnas o en diferentes conjuntos de datos.

Creamos los objetos necesarios:

```
url="https://www.gob.pe/institucion/mef/funcionarios?sheet="
css_cargo="p"
css_name="h3.text-2xl"
final_table = list() # list es una función para crear listas
```

Construimos la función e iteración

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

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

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

```
for(i in 1:3) { # INPUT
  webpage <- read_html(paste0(url, i)) #obtenemos el código html de las 3 páginas
  cargo_texto <- webpage %>%
    html_nodes(css_cargo) %>% # obtener el código html del css del cargo
    html_text() # lo convertimos en un vector de texto
  name_texto <- webpage %>%
    html_nodes(css_name) %>% # obtener el código html del css del name
    html_text() # lo convertimos en un vector de texto

final_table[[i]] <- data.frame(NOMBRE=name_texto, CARGO=cargo_texto) # OUTPUT: Con esto estamos creando
}
```

Ahora convertimos todo en una base de datos

```
dataWS6 = bind_rows(final_table)
head(dataWS6)
```

```
##                             NOMBRE
## 1        Kurt Johnny Burneo Farfan
## 2 José Armando Calderón Valenzuela
## 3    Alex Alonso Contreras Miranda
## 4    Kitty Elisa Trinidad Guerrero
## 5    Jaime Florencio Reyes Miranda
## 6        Hernán Martín Díaz Huamán
##                                       CARGO
## 1               Ministro de Economía y Finanzas
## 2                      Viceministro de Hacienda
## 3                      Viceministro de Economía
## 4                            Secretaria General
## 5                  Jefe de Gabinete de Asesores
## 6 Jefe del Órgano de Control Institucional - OCI
```

También podemos combinar loops con *htmltab*:

```
url = "http://search.beaconforfreedom.org/search/censored_publications/result.html?author=&cauthor=&titl
path = "/html/body/div/div[2]/div[2]/table"
final_table = list()
```

Creamos la función e iteracción (solo lo haremos con las primeras 10 páginas para que no demore mucho):

```
for(i in 1:10) {
  data <- htmltab((paste0(url, i)),path)

final_table[[i]] <- data.frame(data, stringsAsFactors = F)
}
```

```
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
## Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, spe
```

Ahora convertimos todo en una base de datos

```
dataWS7 = bind_rows(final_table)
head(dataWS7)
```

```
##   V1                                                Title
## 1  1 [Seventeen of Sirjani's books was effectively banned]
## 2  2                          Ab ke as cheshme djoda shod
## 3  3                                        Adab-e-zamini
## 4  4                                                Adine
## 5  5                                                Adine
## 6  6                                                Adine
##                     Author                     Country Period.of.censorship
## 1 Sirjani, Ali-Akbar Saidi Iran, Islamic Republic of                 1994
## 2      Ghazi-noor, Ghodsi Iran, Islamic Republic of                 1978
## 3        Koushan, Mansour Iran, Islamic Republic of                 1990
## 4        Koushan, Mansour Iran, Islamic Republic of                 1998
## 5         Alinejad, Sirus Iran, Islamic Republic of                 1986
## 6          Sarkohi, Faraj Iran, Islamic Republic of                 1996
##   Type.of.censorship
## 1            Banned
## 2            Banned
## 3          Censored
## 4          Censored
## 5          Censored
## 6   Censored Closed
```

Practiquemos extrayendo data de este link: https://www.gob.pe/institucion/minsa/funcionarios

3. Exportar la base de datos

Finalmente ¿Cómo exportamos las bases de datos que hemos obtenido? En muchos casos nos va ser útil exportar a CSV,lo podemos hacer con la función export del paquete rio:

```
library(rio)
export(dataWS, "midata.csv")
```

También podemos exportar en formato R:

```
save(dataWS2,file="midata.Rda")
```

Si hay problemas con el encoding:

```
#export(dataWS1, "dataWS1.csv")
#dataWS1 = import("dataWS1.csv", encoding = "UTF-8")
#head(dataWS1)
```