

SUMMARY SESION7

Victor Miguel Terrón Macias

30/1/2021

SESION 7. RStudio Cloud - Github, conexiones con BDs y lectura de datos externos

Durante el transcurso de esta sesión serás capaz de desarrollar las siguientes capacidades de R Trabajar con RStudio desde la nube y enviar (traer) código a (desde) un repositorio de github Conectarte a una BDD con R Importar datos de una BDD a R Manipular datos de una BDD en R

BASE DE DATOS

Una base de datos es una colección organizada de información estructurada, o datos, típicamente almacenados electrónicamente en un sistema de computadora. Una base de datos usualmente se controla por un sistema de gestión de base de datos (DBMS). En conjunto, los datos y el DBMS, junto con las aplicaciones que están asociados con ellos, se conocen como un sistema de base de datos.

Los datos dentro de una bases de datos se modelan generalmente en filas y columnas en una serie de tablas para su procesamiento y que la consulta de datos sea eficiente. La mayoría de las bases de datos utilizan lenguaje de consulta estructurado (SQL) para escribir y consultar datos.

SQL es un lenguaje de programación usado la mayoría de bases de datos relacionales para consultar, manipular y definir datos, y para proporcionar control de acceso. Aunque SQL todavía se usa ampliamente en la actualidad, comienzan a aparecer nuevos lenguajes de programación.

La diferencia entre las bases de datos y las hojas de cálculo (como Microsoft Excel) son dos formas convenientes de almacenar información. Las principales diferencias entre las dos son:

Cómo se almacenan y manipulan los datos Quién puede acceder a los datos Cuántos datos se pueden almacenar Las hojas de cálculo son muy buenas para un solo usuario o un pequeño número de usuarios que no necesitan manipular una gran cantidad de datos complicados. Las bases de datos, están diseñadas para contener colecciones mucho más grandes de información organizada, cantidades masivas en ocasiones, éstas permiten a múltiples usuarios al mismo tiempo acceder y consultar los datos de forma rápida y segura utilizando una lógica y un lenguaje altamente complejos.

Hay diferentes tipos de bases de datos. La mejor base de datos depende de cómo la organización pretende utilizar los datos.

- **Bases de datos relacionales.** Las bases de datos relacionales se popularizaron en los años ochenta. Los elementos de una base de datos relacional se organizan como un conjunto de tablas con columnas y filas. La tecnología de base de datos relacional proporciona la manera más eficiente y flexible de acceder a información estructurada.
- **Bases de datos orientadas a objetos.** La información en una base de datos orientada a objetos se representa en forma de objetos, como en la programación orientada a objetos.

- **Bases de datos distribuidas.** Una base de datos distribuida consta de dos o más archivos ubicados en diferentes sitios. La base de datos puede almacenarse en múltiples computadoras, ubicadas en la misma ubicación física o dispersas en diferentes redes.
- **Almacenes de datos.** Un almacén de datos es un tipo de base de datos diseñada específicamente para consultas y análisis rápidos, y funciona como un depósito central de datos.
- **Bases de datos NoSQL.** Una NoSQL, o una base de datos no relacional, permite que los datos no estructurados y semiestructurados se almacenen y manipulen, a diferencia de una base de datos relacional, que define cómo deben componerse todos los datos insertados en la base de datos. Las bases de datos NoSQL se hicieron populares a medida que las aplicaciones web se hacían más comunes y más complejas.

Algunas de las bases de datos más recientes incluyen:

- **Bases de datos de código abierto.** Un sistema de base de datos de código abierto es aquel cuyo código fuente es de código abierto; dichas bases de datos podrían ser bases de datos SQL o NoSQL.
- **Bases de datos en la nube.** Una base de datos en la nube es una colección de datos, ya sean estructurados o no estructurados, que reside en una plataforma de computación en la nube privada, pública o híbrida. Hay dos tipos de modelos de base de datos en la nube: tradicional y database as a service (DBaaS). Con DBaaS, las tareas administrativas y el mantenimiento son realizados por un proveedor de servicios.
- **Base de datos multimodelo.** Las bases de datos multimodelo combinan diferentes tipos de modelos de base de datos en un único back-end integrado. Esto significa que pueden acomodar varios tipos de datos.
- **Base de datos documental/JSON.** Diseñadas para almacenar, recuperar y administrar información orientada a documentos, las bases de datos documentales son una forma moderna de almacenar datos en formato JSON en lugar de filas y columnas.
- **Bases de datos independientes.** Las bases de datos independientes, el tipo de base de datos más nuevo e innovador (también conocidas como bases de datos autónomas), se basan en la nube y utilizan el aprendizaje autónomo para automatizar el ajuste, la seguridad, las copias de seguridad, las actualizaciones y otras tareas de administración de rutina de las bases de datos que tradicionalmente realizan los administradores de bases de datos.

LECTURA ¿QUÉ ES CLOUD?

¿Qué es cloud? Lo vemos por todos lados y no sabemos a ciencia cierta qué es. Nos metemos de lleno en este concepto que será el futuro.

Seguramente, te sonará esta palabra y la relacionarán con los servidores, con internet, aplicaciones en web, bases de datos. No vas mal desencaminado/a, así que tampoco estás tan mal ¡Eh! . Si no sabes relacionarlo con nada, no te preocupes porque aquí te ayudaremos a aprender qué es.

Podemos denominarlo como “cloud computing“, pero popularmente se le llama “cloud” al suministro de archivos o recursos a petición del usuario a través de una conexión a internet. Como casi cualquier conexión hay un solicitante (el usuario) y un receptor (el servidor), el solicitante pide un recurso a través de su aplicación y el receptor se lo proporciona.

El cloud computing se puede utilizar de diferentes formas. Como podemos ver más abajo.

SaaS SOFTWARE COMO SERVICIO

Si ponemos el ejemplo de una plataforma de streaming, se ejecuta un sistema en la nube, que están conectados a los sistemas de usuario mediante Internet y por un navegador.

Lo más interesante del SaaS es que podemos iniciar sesión y utilizar las aplicaciones. Podemos acceder a los datos desde cualquier lado, mientras tengamos internet. Si el sistema falla, no perderemos los datos y el servicio es escalable.

PaaS PLATAFORMA COMO SERVICIO

Es ideal para grupos de trabajo e intercambiar datos o recursos. Mientras uno puede subir y descargar, los otros sólo pueden acceder a esos datos.

Estas plataformas son muy usadas por las empresas de gran tamaño.

IaaS INFRAESTRUCTURA COMO SERVICIO

Este sistema cloud permite dotar de una infraestructura a las empresas para sus recursos, servidores, redes, almacenamiento de datos, etc. Es un servicio muy usado para las empresas que quieren tener una especie de intranet en la que subir aplicaciones o datos, como descargarla.

Los beneficios de este cloud es que no hace falta invertir en hardware, el cloud es escalable y los servicios se adaptan a las empresas.

Rstudio Cloud y Github se encuentra dentro de las categorías SaaS

JSON JAVASCRIPT OBJECT NOTATION

Es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que resulta mucho más sencillo escribir un analizador sintáctico (parser) para él.

XML

Siglas en inglés de eXtensible Markup Language, traducido como “Lenguaje de Marcado Extensible” o “Lenguaje de Marcas Extensible”, es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

EJEMPLO 1. Conexión entre RStudio y Github

OBJETIVO

- Hacer uso de ecurs nube de RStudio y Github
- Utilizar como repositorio y control de cambios a GitHub
- Hacer commit, push y pull

DESARROLLO

Dentro de esta sesión podrás realizar una interacción entre RStudio y Github, haciendo posible trabajar con scripts en una interface de RStudio desde la nube, es decir, que se cargará en la ventada de tu explorador, con esto podrás trabajar de manera remota con scripts y poder alojarlos en algún repositorio.

- Debes entrar a RStudio cloud y linkear el github con el RStudio con las contraseñas de tu Git y se puede hacer gcommit y pull dependiendo de los cambios hechos.

EJEMPLO 2 CONEXION DE UNA BASE DE DATOS CON R

OBJETIVO

- Conectarse a una BD utilizando R
- Lectura de una BD en R

DESARROLLO

CONFIGURACIONES DE CONEXIÓN: Hay 5 configuraciones necesarias para hacer una conexión:

1. Driver : consulta la sección previa de controladores para obtener información sobre la configuración, se utilizarán los drivers de MySQL
2. Server : una ruta de red al servidor de la base de datos
3. UID : nombre de usuario utilizado para acceder al servidor MySQL
4. PWD : la contraseña correspondiente al UID proporcionado
5. Port : debe establecerse en 3306 generalmente

Comenzaremos instalando las librerías necesarias para realizar la conexión y lectura de la base de datos en RStudio, si previamente los tenías instalados omite la instalación, recuerda que solo necesitas realizarla una vez.

```
install.packages("DBI") install.packages("RMySQL")
```

library(DBI) library(RMySQL) Una vez que se tengan las librerías necesarias se procede a la lectura (podría ser que necesites otras, si te las solicita instalalas y cargalas), de la base de datos de Shiny la cual es un demo y nos permite interactuar con este tipo de objetos. El comando dbConnect es el indicado para realizar la lectura, los demás parámetros son los que nos dan acceso a la BDD.

```
MyDataBase <- dbConnect( drv = RMySQL::MySQL(), dbname = "shinydemo", host = "shiny-  
demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com", username = "guest", password = "guest")
```

Si no se arrojaron errores por parte de R, vamos a explorar la BDD

```
dbListTables(MyDataBase)
```

```
> MyDataBase <- dbConnect(  
+   drv = RMySQL::MySQL(),  
+   dbname = "shinydemo",  
+   host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",  
+   username = "guest",  
+   password = "guest")  
> dbListTables(MyDataBase)  
[1] "city"          "country"       "countryLanguage"
```

Figure 1: TABLAS

Ahora si se quieren desplegar los campos o variables que contiene la tabla City se hará lo siguiente

```
dbListFields(MyDataBase, 'City')
```

Para realizar una consulta tipo MySQL sobre la tabla seleccionada haremos lo siguiente

DataDB <- dbGetQuery(MyDataBase, "select * from City") Observemos que el objeto DataDB es un data frame, por lo tanto ya es un objeto de R y podemos aplicar los comandos usuales

```
class(DataDB) dim(DataDB) head(DataDB)
```

```
> DataDB <- dbGetQuery(MyDataBase, "select * from City")
> class(DataDB)
[1] "data.frame"
> dim(DataDB)
[1] 3427    5
> head(DataDB)
```

	ID	Name	CountryCode	District	Population
1	1	Kabul	AFG	Kabul	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200
6	6	Rotterdam	NLD	Zuid-Holland	593321

Figure 2: TABLAS 2

```
pop.mean <- mean(DataDB$Population) # Media a la variable de población pop.mean
```

```
pop.3 <- pop.mean *3 # Operaciones aritméticas pop.3 Incluso podemos hacer uso de otros comandos de búsqueda aplicando la librería dplyr
```

```
library(dplyr) pop50.mex <- DataDB %>% filter(CountryCode == "MEX" , Population > 50000) # Ciudades del país de México con más de 50,000 habitantes
```

```
head(pop50.mex)
```

```
unique(DataDB$CountryCode) # Países que contiene la BDD
```

Ejemplo 2. Conexión a una BDD con R

Comenzaremos instalando las librerías necesarias para realizar la conexión y lectura de la base de datos en RStudio, si previamente los tenías instalados # omite la instalación, recuerda que solo necesitas realizarla una vez.

```
# install.packages("DBI")
# install.packages("RMySQL")
```

```
library(DBI)
library(RMySQL)
```

Una vez que se tengan las librerías necesarias se procede a la lectura (podría ser que necesites otras, si te las solicita instalalas y cargalas), de la base de datos de Shiny la cual es un demo y nos permite interactuar con este tipo de objetos. El comando dbConnect es el indicado para realizar la lectura, los demás parametros son los que nos dan acceso a la BDD.

```
MyDataBase <- dbConnect(
  drv = RMySQL::MySQL(),
```

```
dbname = "shinydemo",
host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
username = "guest",
password = "guest")
```

Si no se arrojaron errores por parte de R, vamos a explorar la BDD

```
dbListTables(MyDataBase)
```

```
[1] "City"          "Country"       "CountryLanguage"
```

*# Ahora si se quieren desplegar los campos o variables que contiene la tabla
City se hará lo siguiente*

```
dbListFields(MyDataBase, 'City')
```

```
[1] "ID"            "Name"          "CountryCode"  "District"     "Population"
```

*# Para realizar una consulta tipo MySQL sobre la tabla seleccionada haremos lo
siguiente*

```
DataDB <- dbGetQuery(MyDataBase, "select * from City")
```

*# Observemos que el objeto DataDB es un data frame, por lo tanto ya es un objeto
de R y podemos aplicar los comandos usuales*

```
class(DataDB)
```

```
[1] "data.frame"
```

```
head(DataDB)
```

	ID	Name	CountryCode	District	Population
1	1	Kabul	AFG	Kabul	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200
6	6	Rotterdam	NLD	Zuid-Holland	593321

```
pop.mean <- mean(DataDB$Population) # Media a la variable de población
pop.mean
```

```
[1] 359985.8
```

```
pop.3 <- pop.mean *3 # Operaciones aritméticas
pop.3
```

```
[1] 1079957
```

```
# Incluso podemos hacer unos de otros comandos de busqueda aplicando la
# libreria dplyr
```

```
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

```
pop50.mex <- DataDB %>% filter(CountryCode == "MEX" , Population > 50000) # Ciudades del país de Mé.
```

```
head(pop50.mex)
```

	ID	Name	CountryCode	District	Population
1	2516	Guadalajara	MEX	Jalisco	1647720
2	2518	Puebla	MEX	Puebla	1346176
3	2521	Tijuana	MEX	Baja California	1212232
4	2524	Zapopan	MEX	Jalisco	1002239
5	2526	Mexicali	MEX	Baja California	764902
6	2531	Chihuahua	MEX	Chihuahua	670208

```
unique(DataDB$CountryCode) # Países que contiene la BDD
```

```
[1] "AFG" "NLD" "ALB" "DZA" "ASM" "AND" "AGO" "ARE" "ARG" "ARM" "AUS" "AZE"
[13] "BHS" "BHR" "BGD" "BRB" "BEL" "BLZ" "BEN" "BMU" "BTN" "BOL" "BIH" "BWA"
[25] "BRA" "GBR" "VGB" "BRN" "BGR" "BFA" "BDI" "CYM" "CHL" "COK" "DJI" "DMA"
[37] "DOM" "ECU" "EGY" "SLV" "ERI" "ESP" "ZAF" "ETH" "FLK" "FJI" "PHL" "GAB"
[49] "GMB" "GEO" "GHA" "GRL" "GLP" "GTM" "GIN" "GNB" "GUY" "HTI" "HND" "HKG"
[61] "IDN" "IND" "IRQ" "IRN" "IRL" "ISR" "ITA" "TMP" "AUT" "JAM" "JPN" "YEM"
[73] "JOR" "YUG" "KHM" "CMR" "CAN" "KAZ" "KEN" "CAF" "CHN" "KGZ" "KIR" "COL"
[85] "COM" "COG" "COD" "CCK" "PRK" "KOR" "GRC" "HRV" "CUB" "KWT" "CYP" "LAO"
[97] "LVA" "LSO" "LBN" "LBR" "LBY" "LIE" "LTU" "MAC" "MDG" "MKD" "MWI" "MDV"
[109] "MYS" "MLI" "MLT" "MAR" "MHL" "MTQ" "MRT" "MUS" "MYT" "MEX" "FSM" "MDA"
[121] "MNG" "MSR" "MOZ" "MMR" "NAM" "NPL" "NIC" "NER" "NGA" "NOR" "CIV" "OMN"
[133] "PAK" "PLW" "PAN" "PNG" "PRY" "PER" "MNP" "PRT" "PRI" "POL" "GNQ" "QAT"
[145] "FRA" "GUF" "PYF" "REU" "ROM" "RWA" "SWE" "SHN" "KNA" "LCA" "VCT" "SPM"
[157] "DEU" "SLB" "ZMB" "WSM" "SMR" "SAU" "SEN" "SLE" "SVK" "SVN" "SOM" "LKA"
[169] "SDN" "FIN" "SUR" "SWZ" "CHE" "SYR" "TJK" "TWN" "TZA" "DNK" "THA" "TKL"
[181] "TTO" "TCD" "TUN" "TUR" "TKM" "TCA" "TUV" "UGA" "UKR" "HUN" "URY" "NZL"
[193] "UZB" "BLR" "WLF" "VUT" "VEN" "RUS" "VNM" "EST" "USA" "VIR" "ZWE" "PSE"
```

RETO 1 RStudio Cloud -> GitHub

OBJETIVO

Practicar como transferir un archivo desde RStudio Cloud hacia Github y viceversa

DESARROLLO

Ahora vas a practicar los conocimientos adquiridos en esta sesión principalmente con RStudio y Github

- Crea un repositorio en Github llamado Reto_Sesion_7
- Crea un Project llamado Reto_Sesion_07 dentro de RStudio Cloud utilizando tu cuenta de RStudio, que esté ligado al repositorio recién creado
- Ahora en RStudio crea un script llamado queries.Ren donde se conecte a la BDD shinydemo
- Una vez hecha la conexión a la BDD, generar una búsqueda con dplyr que devuelva el porcentaje de personas que hablan español en todos los países
- Realizar una gráfica con ggplot que represente este porcentaje de tal modo que en el eje de las Y aparezca el país y en X el porcentaje, y que diferencie entre aquellos que es su lengua oficial y los que no con diferente color (puedes utilizar la geom_bin2d() y coord_flip())
- Una vez hecho esto hacer el commit y push para mandar tu archivo al repositorio de Github Reto_Sesion_7

SOLUCION

CREAR EL REPOSITORIO EN GitHub

VINCULARLO A RSTUDIO CLOUD

REALIZAR EL COMMIT Y EL PUSH



COMPROBACION DEL PUSH EN GITHUB

CODIGO DEL RETO

```
# install.packages("DBI")
# install.packages("RMySQL")
# install.packages("dplyr")
# install.packages("ggplot2")
library(dplyr)
library(DBI)
library(RMySQL)
library(ggplot2)


MyDataBase <- dbConnect(
  drv = RMySQL::MySQL(),
  dbname = "shinydemo",
  host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
```



Owner * Repository name *

 Usuario / Reto_Sesion_7 

Great repository names are short and memorable. Need inspiration? How about [fictional-octo-guide?](#)

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.


☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

Create repository

Figure 3: SOL1

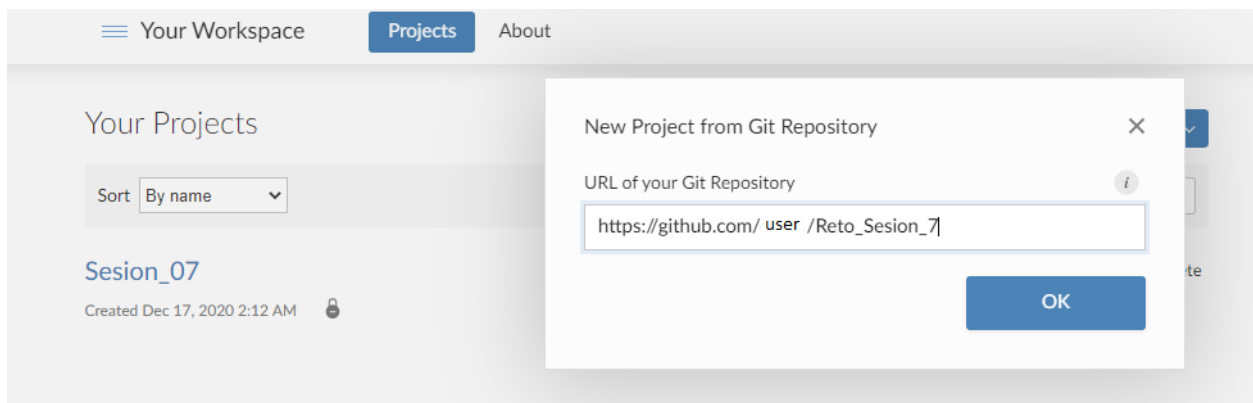


Figure 4: SOL2

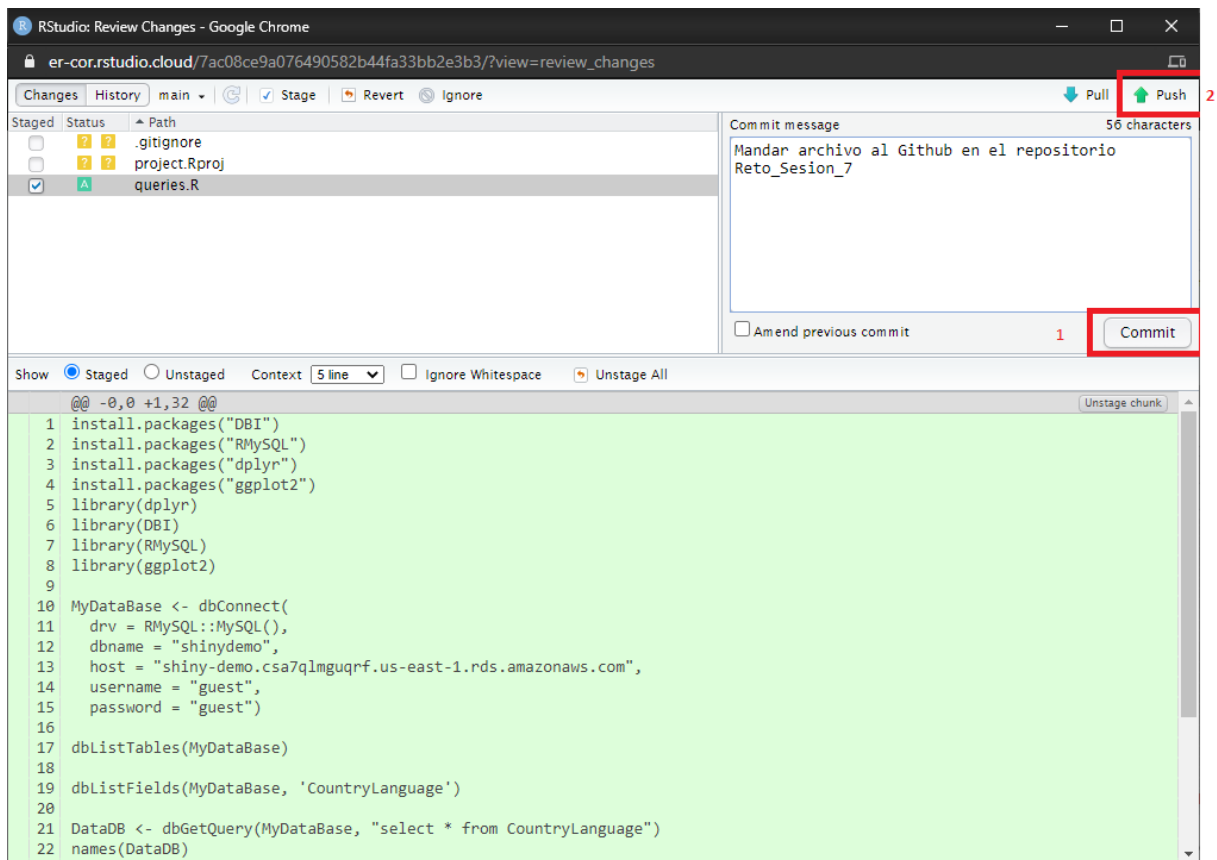


Figure 5: SOLU3

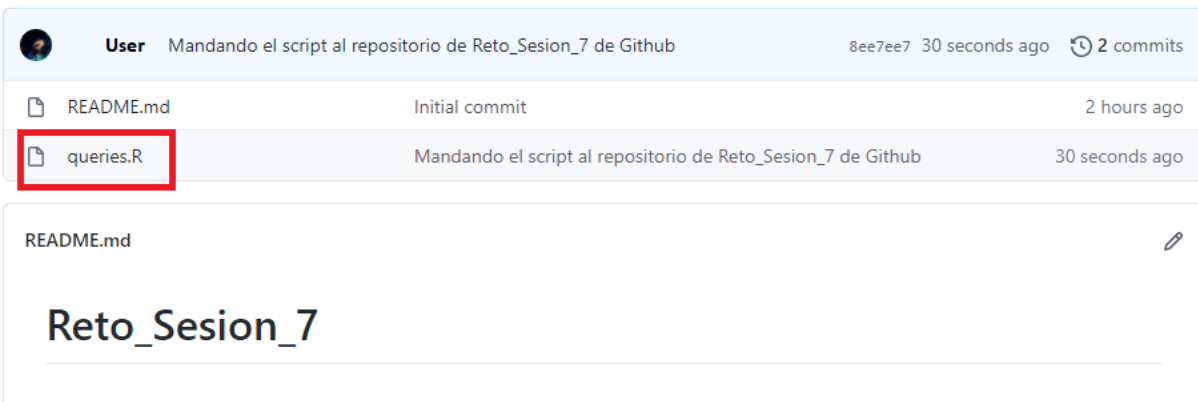


Figure 6: SOLU4

```
username = "guest",
password = "guest")

dbListTables(MyDataBase)

[1] "City"          "Country"       "CountryLanguage"

dbListFields(MyDataBase, 'CountryLanguage')

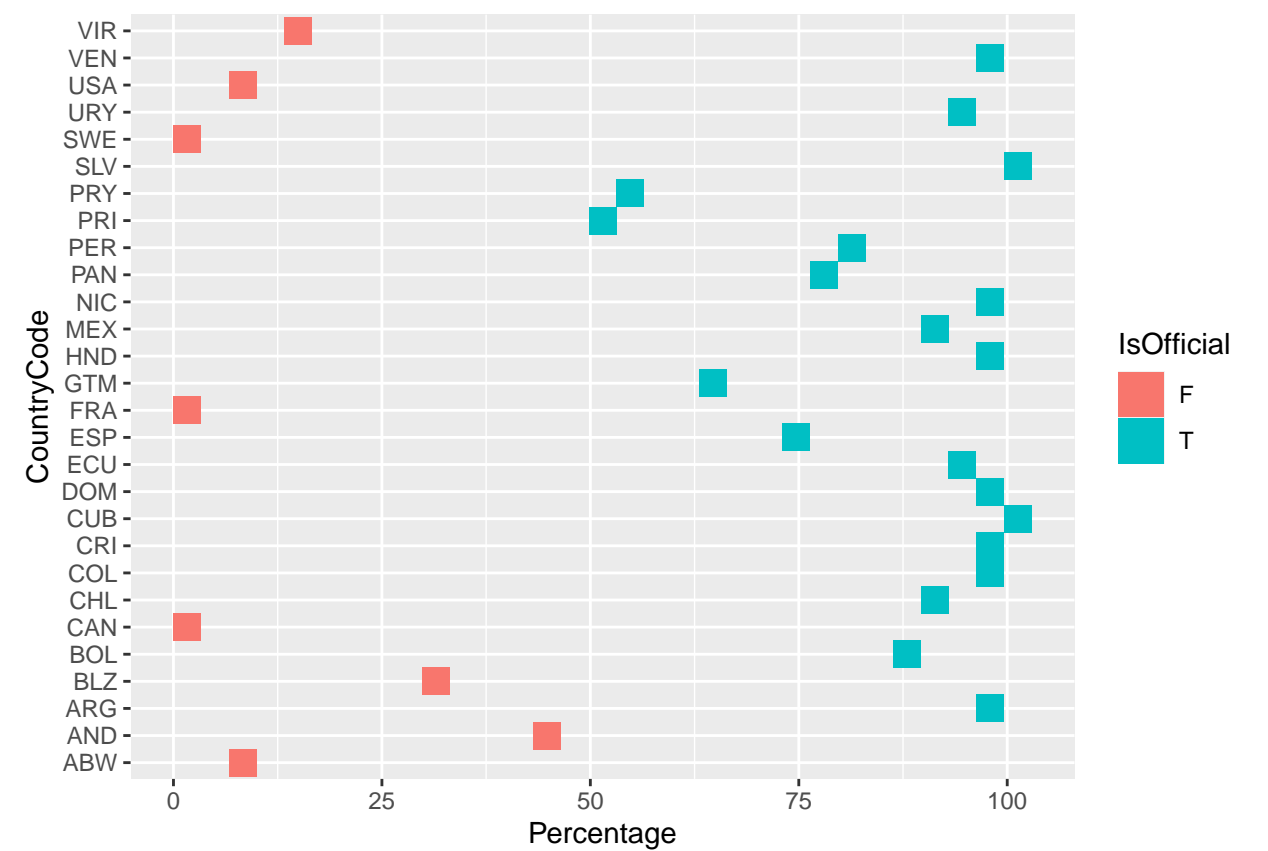
[1] "CountryCode" "Language"      "IsOfficial"  "Percentage"

DataDB <- dbGetQuery(MyDataBase, "select * from CountryLanguage")
names(DataDB)

[1] "CountryCode" "Language"      "IsOfficial"  "Percentage"

SP <- DataDB %>% filter(Language == "Spanish")
SP.df <- as.data.frame(SP)

SP.df %>% ggplot(aes( x = CountryCode, y=Percentage, fill = IsOfficial )) +
  geom_bin2d() +
  coord_flip()
```



EJEMPLO 3. Variantes en la lectura de BDD con R

OBJETIVO

Utilizar la librería *dplyr* y *pool* para hacer queries a *MySQL*

DESARROLLO

Ahora utilizaremos otra opción para realizar queries a una BDD con la ayuda de *dplyr* que sustituye a *SELECT* en *MySQL* y el operador *%>%*, hay que recordar que con este comando también podemos realizar búsquedas de forma local.

Comenzamos instalando las paqueterías necesarias y cargándolas a R

```
install.packages("pool") install.packages("dbplyr")
```

`library(dbplyr)` `library(pool)` Se realiza la lectura de la BDD con el comando `dbPool`, los demás parámetros se siguen utilizando igual que el ejemplo anterior

```
my_db <- dbPool( RMySQL::MySQL(), dbname = "shinydemo", host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com", username = "guest", password = "guest" )
```

 Para ver el contenido de la BDD y realizar una búsqueda se procede de la siguiente manera

```
dbListTables(my_db)
```

```
my_db %>% tbl("Country") %>% head(5) # library(dplyr)
```

```
my_db %>% tbl("CountryLanguage") %>% head(5)
```

 Otra forma de generar una búsqueda será con la librería *DBI*, utilizando el comando `dbSendQuery`

```
library(DBI) conn <- dbConnect( drv = RMySQL::MySQL(), dbname = "shinydemo", host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com", username = "guest", password = "guest" )
```

```
rs <- dbSendQuery(conn, "SELECT * FROM City LIMIT 5;")
```

```
dbFetch(rs)
```

```
> rs <- dbSendQuery(conn, "SELECT * FROM City LIMIT 5;")
> dbFetch(rs)
```

	ID	Name	CountryCode	District	Population
1	1	Kabul	AFG	Kabul	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200

Figure 7: QUERIE

Para finalizar nos desconectamos de la BDD

```
dbClearResult(rs) dbDisconnect(conn)}
```

Ejemplo 3. Variantes en la lectura de BDD con R

*# Ahora utilizaremos otra opción para realizar queries a una BDD con la ayuda
de dplyr que sustituye a SELECT en MySQL y el operador %>%, hay que recordar*

```
# que con este comando también podemos realizar búsquedas de forma local.

# Comenzamos instalando las paqueterías necesarias y cargándolas a R

# install.packages("pool")
# install.packages("dbplyr")

library(dbplyr)
```

Attaching package: 'dbplyr'

The following objects are masked from 'package:dplyr':

```
ident, sql
```

```
library(pool)

# Se realiza la lectura de la BDD con el comando dbPool, los demás parámetros
# se siguen utilizando igual que el ejemplo anterior

my_db <- dbPool(
  RMySQL::MySQL(),
  dbname = "shinydemo",
  host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
  username = "guest",
  password = "guest"
)

# Para ver el contenido de la BDD y realizar una búsqueda se procede de la
# siguiente manera

dbListTables(my_db)
```

```
[1] "City"          "Country"       "CountryLanguage"
```

```
# Obtener los primeros 5 registros de Country
```

```
my_db %>% tbl("Country") %>% head(5) # library(dplyr)
```

```
# Source:   lazy query [?? x 15]
# Database: mysql 10.1.34-MariaDB
#   [guest@shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com:/shinydemo]
#   Code Name Continent Region SurfaceArea IndepYear Population LifeExpectancy
#   <chr> <chr> <chr>      <chr>          <dbl>      <int>      <int>      <dbl>
1 ABW  Aruba North Am~  Carib~          193         NA        103000      78.4
2 AFG  Afgh~ Asia      South~        652090      1919     22720000     45.9
3 AGO  Ango~ Africa  Centr~       1246700      1975     12878000     38.3
4 AIA  Angu~ North Am~  Carib~          96         NA         8000      76.1
5 ALB  Alba~ Europe  South~       28748      1912     3401200     71.6
# ... with 7 more variables: GNP <dbl>, GNPOld <dbl>, LocalName <chr>,
#   GovernmentForm <chr>, HeadOfState <chr>, Capital <int>, Code2 <chr>
```

```
# Obtener los primeros 5 registros de CountryLanguage
```

```
my_db %>% tbl("CountryLanguage") %>% head(5)
```

```
# Source:   lazy query [?? x 4]
# Database: mysql 10.1.34-MariaDB
# [guest@shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com:/shinydemo]
  CountryCode Language  IsOfficial Percentage
  <chr>        <chr>    <chr>          <dbl>
1 ABW         Dutch     T              5.3
2 ABW         English   F              9.5
3 ABW         Papiament F         76.7
4 ABW         Spanish   F              7.4
5 AFG         Balochi    F              0.9
```

```
# Otra forma de generar una búsqueda será con la librería DBI, utilizando el
# comando dbSendQuery
```

```
library(DBI)
conn <- dbConnect(
  drv = RMySQL::MySQL(),
  dbname = "shinydemo",
  host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
  username = "guest",
  password = "guest")

rs <- dbSendQuery(conn, "SELECT * FROM City LIMIT 5;")

dbFetch(rs)
```

	ID	Name	CountryCode	District	Population
1	1	Kabul	AFG	Kabul	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200

```
# Para finalizar nos desconectamos de la BDD
```

```
dbClearResult(rs)
```

```
[1] TRUE
```

```
dbDisconnect(conn)
```

```
[1] TRUE
```

EJEMPLO 4. LECTURA DE ARCHIVOS JSON, XML Y TABLAS EN HTML

OBJETIVO

- Realizar lectura de archivos JSON y XML para poder aplicar las funciones que se requieran de R y poder extraer información convirtiéndola en un data frame

DESARROLLO

Comenzaremos instalando los paquetes necesarios para después cargarlos a R

```
install.packages("rjson") #Siempre usar comillas en el nombre del paquete ->
```

```
library(rjson) # Quitar comillas del nombre
```

```
# EJEMPLO 4 SESION 7
# Ejemplo 4. Lectura de archivos JSON, XML y tablas en HTML

# Comenzaremos instalando los paquetes necesarios para después cargarlos a R

# install.packages("rjson") #Siempre usar comillas en el nombre del paquete

library(rjson) # Quitar comillas del nombre

# Json
# Vamos a leer un archivo Json de prueba alojado aquí

URL <- "https://tools.learningcontainer.com/sample-json-file.json" # Asignando el link a una variable

JsonData <- fromJSON(file= URL) # Se guarda el JSON en un objeto de R

class(JsonData) # Vemos que tipo de objeto es JsonData
```

```
[1] "list"
```

```
str(JsonData) # Vemos la naturaleza de sus variables
```

```
List of 5
 $ Name      : chr "Test"
 $ Mobile    : num 12345678
 $ Boolean: logi TRUE
 $ Pets      : chr [1:2] "Dog" "cat"
 $ Address:List of 2
  ..$ Permanent address: chr "USA"
  ..$ current Address   : chr "AU"
```

```
# Finalmente ya que pudimos acceder al contenido del Json, también podemos
# realizar la manipulación de los datos dentro del Json, por ejemplo:
```

```
sqrt(JsonData$Mobile)
```

```
[1] 3513.642
```

```
# Para entrar a las demás variables recuerda que puedes usar el operador de $,
# es decir, JsonData$

# XML
# Ahora vamos a leer datos XML en R, utilizando un archivo XML alojado aquí

# Lo primero es instalar y cargar el paquete XML y alojar el link en una variable
# link, para su lectura

# install.packages("XML")
library(XML)
link <- "http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/xml/cd_catalog.xml"

# Analizando el XML desde la web
xmlfile <- xmlTreeParse(link)
# Ahora ya podemos ver las propiedades del objeto xmlfile

summary(xmlfile)
```

```
      Length Class          Mode
doc  3      XMLDocumentContent list
dtd  2      DTDList             list
```

```
head(xmlfile)
```

```
$doc
```

```
$file
```

```
[1] "http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/xml/cd_catalog.xml"
```

```
$version
```

```
[1] "1.0"
```

```
$children
```

```
$children$CATALOG
```

```
<CATALOG>
```

```
<CD>
```

```
<TITLE>Empire Burlesque</TITLE>
```

```
<ARTIST>Bob Dylan</ARTIST>
```

```
<COUNTRY>USA</COUNTRY>
```

```
<COMPANY>Columbia</COMPANY>
```

```
<PRICE>10.90</PRICE>
```

```
<YEAR>1985</YEAR>
```

```
</CD>
```

```
<CD>
```

```
<TITLE>Hide your heart</TITLE>
```

```
<ARTIST>Bonnie Tyler</ARTIST>
```

```
<COUNTRY>UK</COUNTRY>
```

```
<COMPANY>CBS Records</COMPANY>
```

```
<PRICE>9.90</PRICE>
```

```
<YEAR>1988</YEAR>
```

```
</CD>
```



```

<CD>
  <TITLE>Greatest Hits</TITLE>
  <ARTIST>Dolly Parton</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>RCA</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1982</YEAR>
</CD>
<CD>
  <TITLE>Still got the blues</TITLE>
  <ARTIST>Gary Moore</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Virgin records</COMPANY>
  <PRICE>10.20</PRICE>
  <YEAR>1990</YEAR>
</CD>
<CD>
  <TITLE>Eros</TITLE>
  <ARTIST>Eros Ramazzotti</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>BMG</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1997</YEAR>
</CD>
<CD>
  <TITLE>One night only</TITLE>
  <ARTIST>Bee Gees</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Polydor</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1998</YEAR>
</CD>
<CD>
  <TITLE>Sylvias Mother</TITLE>
  <ARTIST>Dr.Hook</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>CBS</COMPANY>
  <PRICE>8.10</PRICE>
  <YEAR>1973</YEAR>
</CD>
<CD>
  <TITLE>Maggie May</TITLE>
  <ARTIST>Rod Stewart</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Pickwick</COMPANY>
  <PRICE>8.50</PRICE>
  <YEAR>1990</YEAR>
</CD>
<CD>
  <TITLE>Romanza</TITLE>
  <ARTIST>Andrea Bocelli</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>Polydor</COMPANY>
  <PRICE>10.80</PRICE>

```

```

    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>When a man loves a woman</TITLE>
    <ARTIST>Percy Sledge</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Black angel</TITLE>
    <ARTIST>Savage Rose</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Mega</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
  </CD>
  <CD>
    <TITLE>For the good times</TITLE>
    <ARTIST>Kenny Rogers</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Mucik Master</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
  <CD>
    <TITLE>Tupelo Honey</TITLE>
    <ARTIST>Van Morrison</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1971</YEAR>
  </CD>
  <CD>
    <TITLE>Soulsville</TITLE>
    <ARTIST>Jorn Hoel</ARTIST>
    <COUNTRY>Norway</COUNTRY>

```

```

<COMPANY>WEA</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1996</YEAR>
</CD>
<CD>
<TITLE>The very best of</TITLE>
<ARTIST>Cat Stevens</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Island</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1990</YEAR>
</CD>
<CD>
<TITLE>Stop</TITLE>
<ARTIST>Sam Brown</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>A and M</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Bridge of Spies</TITLE>
<ARTIST>T&apos;Pau</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Siren</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Private Dancer</TITLE>
<ARTIST>Tina Turner</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Capitol</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1983</YEAR>
</CD>
<CD>
<TITLE>Midt om natten</TITLE>
<ARTIST>Kim Larsen</ARTIST>
<COUNTRY>EU</COUNTRY>
<COMPANY>Medley</COMPANY>
<PRICE>7.80</PRICE>
<YEAR>1983</YEAR>
</CD>
<CD>
<TITLE>Pavarotti Gala Concert</TITLE>
<ARTIST>Luciano Pavarotti</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>DECCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1991</YEAR>
</CD>
<CD>
<TITLE>The dock of the bay</TITLE>

```

```

<ARTIST>Otis Redding</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Atlantic</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Picture book</TITLE>
<ARTIST>Simply Red</ARTIST>
<COUNTRY>EU</COUNTRY>
<COMPANY>Elektra</COMPANY>
<PRICE>7.20</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Red</TITLE>
<ARTIST>The Communards</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>London</COMPANY>
<PRICE>7.80</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Unchain my heart</TITLE>
<ARTIST>Joe Cocker</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>EMI</COMPANY>
<PRICE>8.20</PRICE>
<YEAR>1987</YEAR>
</CD>
</CATALOG>

```

```

attr("class")
[1] "XMLDocumentContent"

```

```

$dtd
$external
NULL

```

```

$internal
NULL

```

```

attr("class")
[1] "DTDList"

```

También gracias al xmlTreeParse podemos extraer los datos contenidos en el archivo

#Extraer los valores xml

```
topxml <- xmlSApply(xmlfile, function(x) xmlSApply(x, xmlValue))
```

Colocandolos en un Data Frame

```
xml_df <- data.frame(t(topxml), row.names= NULL)
```

```
str(xml_df) # Observar la naturaleza de las variables del DF
```

```
'data.frame': 26 obs. of 6 variables:
 $ TITLE : chr "Empire Burlesque" "Hide your heart" "Greatest Hits" "Still got the blues" ...
 $ ARTIST : chr "Bob Dylan" "Bonnie Tyler" "Dolly Parton" "Gary Moore" ...
 $ COUNTRY: chr "USA" "UK" "USA" "UK" ...
 $ COMPANY: chr "Columbia" "CBS Records" "RCA" "Virgin records" ...
 $ PRICE : chr "10.90" "9.90" "9.90" "10.20" ...
 $ YEAR : chr "1985" "1988" "1982" "1990" ...
```

```
# Convertiremos incluso las variables de PRICE y YEAR en datos numéricos para
# poder realizar operaciones con este dato
```

```
xml_df$PRICE <- as.numeric(xml_df$PRICE)
xml_df$YEAR <- as.numeric(xml_df$YEAR)

mean(xml_df$PRICE)
```

```
[1] 9.115385
```

```
mean(xml_df$YEAR)
```

```
[1] 1988.731
```

```
# Todo esto se puede realizar en un solo paso utilizando el siguiente comando
```

```
data_df <- xmlToDataFrame(link)
head(data_df)
```

	TITLE	ARTIST	COUNTRY	COMPANY	PRICE	YEAR
1	Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
2	Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988
3	Greatest Hits	Dolly Parton	USA	RCA	9.90	1982
4	Still got the blues	Gary Moore	UK	Virgin records	10.20	1990
5	Eros	Eros Ramazzotti	EU	BMG	9.90	1997
6	One night only	Bee Gees	UK	Polydor	10.90	1998

```
# Tablas en HTML
```

```
# Comenzamos instalando el paquete rvest el cual nos permitirá realizar la
# lectura de la tabla en el HTML
```

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

```
Loading required package: xml2
```

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

The following object is masked from 'package:XML':

```
xml
```

```
# Introducimos una dirección URL donde se encuentre una tabla

theurl <- "https://solarviews.com/span/data2.htm"
file <- read_html(theurl) # Leemos el html
# Selecciona pedazos dentro del HTML para identificar la tabla

tables <- html_nodes(file, "table")
# Hay que analizar 'tables' para determinar cual es la posición en la lista
# que contiene la tabla, en este caso es la no. 4

# Extraemos la tabla de acuerdo a la posición en la lista

table1 <- html_table(tables[4], fill = TRUE)

table <- na.omit(as.data.frame(table1)) # Quitamos NA's que meten filas extras
# y convertimos la lista en un data frame para su manipulación con R

str(table) # Vemos la naturaleza de las variables
```

```
'data.frame': 71 obs. of 8 variables:
 $ Nombre      : chr  "Sol" "Mercurio" "Venus" "Tierra" ...
 $ Vo          : chr  "-26.8" "-1.9" "-4.4" "-" ...
 $ Distancia.103.km.: chr  "0" "57,910" "108,200" "149,600" ...
 $ Radio.km.    : chr  "695000" "2439.7" "6051.8" "6378.14" ...
 $ Masa.kg.     : chr  "1.989e+30" "3.303e+23" "4.869e+24" "5.976e+24" ...
 $ Densidad.g.cm3. : chr  "1.410" "5.42" "5.25" "5.515" ...
 $ V..Escape.km.s. : chr  "618.02" "4.2507" "10.362" "11.182" ...
 $ Albedo       : chr  "-" "0.10" "0.65" "0.37" ...
 - attr(*, "na.action")= 'omit' Named int [1:20] 1 2 4 5 7 8 10 11 14 15 ...
 .. attr(*, "names")= chr [1:20] "1" "2" "4" "5" ...
```

```
# Por último realizamos una conversión de una columna tipo chr a num, se pueden
# hacer las conversiones que se requieran
```

```
table$Albedo <- as.numeric(table$Albedo)
```

Warning: NAs introducidos por coerción

```
str(table)
```

```
'data.frame': 71 obs. of 8 variables:
 $ Nombre      : chr  "Sol" "Mercurio" "Venus" "Tierra" ...
 $ Vo          : chr  "-26.8" "-1.9" "-4.4" "-" ...
 $ Distancia.103.km.: chr  "0" "57,910" "108,200" "149,600" ...
 $ Radio.km.    : chr  "695000" "2439.7" "6051.8" "6378.14" ...
 $ Masa.kg.     : chr  "1.989e+30" "3.303e+23" "4.869e+24" "5.976e+24" ...
 $ Densidad.g.cm3. : chr  "1.410" "5.42" "5.25" "5.515" ...
 $ V..Escape.km.s. : chr  "618.02" "4.2507" "10.362" "11.182" ...
```

```

$ Albedo          : num  NA 0.1 0.65 0.37 0.12 0.15 0.06 0.07 0.52 0.05 ...
- attr(*, "na.action")= 'omit' Named int [1:20] 1 2 4 5 7 8 10 11 14 15 ...
..- attr(*, "names")= chr [1:20] "1" "2" "4" "5" ...

```