

Un proyecto de datos tiene una gran cantidad de componentes. Sin embargo, en básicamente todos se necesita iterar sobre el ciclo que se muestra en la figura 1.

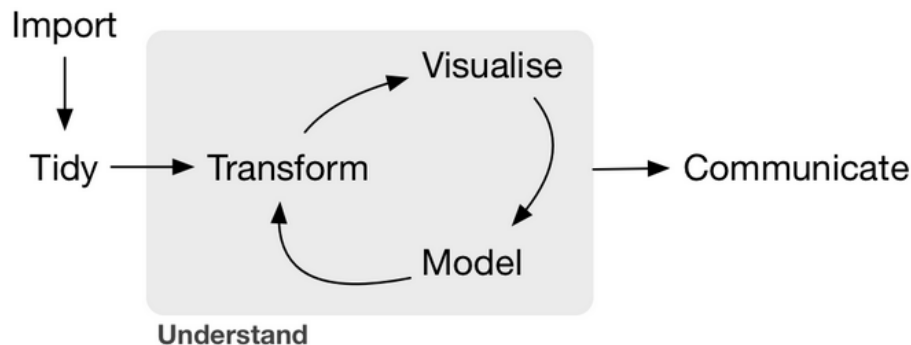


Figura 1: Modelo de las herramientas que se necesitan en un proyecto de datos según Golemund y H. Wickham (2016, Introducción).

Primero es necesario **importar** nuestros datos a R. Los datos pueden estar en una gran cantidad de formatos o lugares.

Después, normalmente es necesario **limpiar** nuestros datos, es decir, seguir criterios de datos limpios de tal forma que como guardemos los datos equivalga a la semántica de los datos que tenemos. Es muy importante primero limpiar porque esto provee de consistencia a lo largo del análisis.

Posteriormente, en casi todo proyecto, será necesario **transformar** los datos. A veces esto implica enfocarse en un subconjunto de los datos, generar nuevas variables, calcular estadísticos, arreglar los datos de cierta manera, entre muchos otros.

Solamente después de estas etapas podemos empezar a generar conocimiento a partir de los datos. Para esto tenemos dos herramientas fundamentales: la estadística descriptiva (en el diagrama reducido a **visualización**) y la generación de **modelos**. La primera es fundamental pues permite derivar preguntas pertinentes a los datos, encontrar patrones, respuestas, plantear hipótesis. Sin embargo, éstas no escalan de la misma manera que los modelos pues estos, una vez que aceptamos sus supuestos generan los resultados que esperamos o contestan la pregunta planteada.

Por último, necesitamos **comunicar** los resultados.

En este capítulo nos ocuparemos, por sección, únicamente de 4 de las etapas mencionadas: importación, limpieza, transformación y visualización.

## Importación de datos

Esta sección resume algunas de las funciones existentes para **importar** datos de distintos formatos a R. En la figura 2 podemos ver la etapa del análisis de datos correspondiente.

Para aplicar las herramientas de R a nuestro trabajo, es necesario poder importar nuestros datos a R. R tiene conectores ya implementados para casi cualquier tipo y formato de datos. Entre los más comunes están<sup>1</sup>:

Formato	Lectura	Escritura
rds	base::readRDS	base::saveRDS

<sup>1</sup>La lista no pretende ser comprehensiva, sin embargo, se presentan algunos de los formatos de datos más comunes. De igual forma, se presentan algunas funciones que sirven para conectar R con datos que están guardados en un manejador de datos externo o en la nube. En caso de presentarse más de un método es porque aunque la recomendación de uso es la función en negritas, la otra opción es más antigua y muy utilizada.

Formato	Lectura	Escritura
separado por *	<code>utils::read.table</code> ; <b><code>readr::read_delim</code></b>	<code>utils::write.table</code> ; <b><code>readr::write_delim</code></b>
csv	<code>utils::read.csv</code> ; <b><code>readr::read_csv</code></b>	<code>utils::write.csv</code> ; <b><code>readr::write_csv</code></b>
Microsoft Excel	<code>readxl::read_excel</code>	<code>xlsx::write.xlsx</code>
dbf	<code>foreign::read.dbf</code>	<code>foreign::write.dbf</code>
IBM SPSS	<code>haven::read_sav</code>	<code>haven::write_sav</code>
Stata	<code>haven::read_dta</code>	<code>haven::write_dta</code>
SAS	<code>haven::read_sas</code>	<code>haven::write_sas</code>
Google spreadsheet	<code>googlesheets::gs_read</code>	<code>googlesheets::gs_new</code>
Google bigquery		<code>bigquery::query_exec</code>
rdata	<code>base::load</code>	<code>base::save</code>

Los paquetes utilizados son (corre estos comandos en la consola):

```
library(foreign)
library(haven)
library(readr)
library(readxl)
library(xlsx)
library(googlesheets)
```

### Importancia de rutas relativas

Para leer un archivo, recordemos el comando `getwd()` para encontrar la carpeta a la cual R esta dirigido en este momento. Una buena practica es considerar el directorio de trabajo como el lugar en donde esta guardado el archivo o **script** en el que se trabaja y “moverse” desde ahi hasta el archivo que se quiere leer.

Ya sea en escritura o en lectura, R buscará a partir del directorio de trabajo (el que se despliega con `getwd()`) para buscar a partir de ahí el archivo por leer o para guardar el que se escribirá si se usan rutas relativas.

En caso de usar rutas absolutas a pesar de que esto **no** es una *buena práctica*, se hará lectura o escritura del archivo en el lugar especificado.



### Ejercicios

R tiene conexión con muchos de los formatos en los que se encuentran los datos. Veremos algunos de los mas relevantes.

El código en cada uno de los **chunks** (un chunk es el pedazo del documento en donde hay código de R) está hecho para que puedas correrlo en la consola (excepto cuando dice explícitamente *do not run* (leyenda comunmente encontrada en los ejemplos de la documentación de las funciones. Con esto entenderás mejor el concepto de rutas relativas.

## rds

La extensión **rds** es de las más comunmente utilizada en R, por ejemplo, para guardar los metadatos para un paquete. Las funciones pertenecen al **base** (R Core Team 2016a). Permiten guardar un solo objeto de R a un archivo y recuperarlo.

Para **escribirlos**

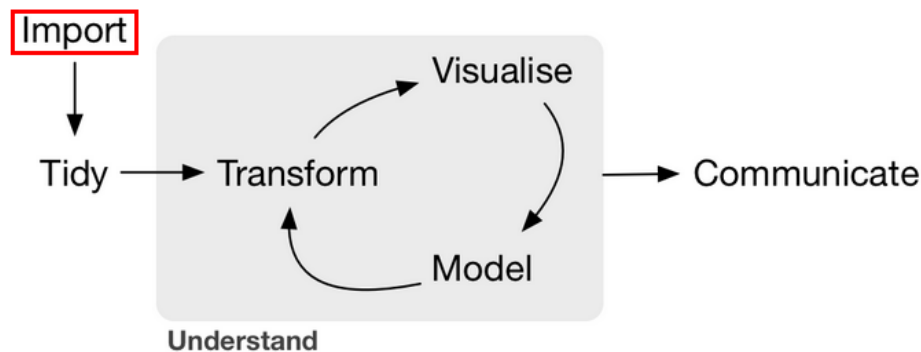


Figura 2: Importación en el análisis de datos Grolemond y H. Wickham (2016, Introducción).

```

# Creamos un dataframe llamado misdatos
misdatos <- iris
# Los guardamos en comprimido
saveRDS(misdatos, file = "misdatos.rds", ascii = FALSE, version = NULL,
        compress = TRUE, refhook = NULL)

```

Nota como si usas el comando `getwd()` y después vas a la ruta indicada por medio del explorador de archivos, verás en esa carpeta el archivo `misdatos_locales.rds`.

Para **leerlos** usamos la ruta relativa. Dado que los guardamos en el directorio de trabajo actual (Recuerda, se puede cambiar con el comando `setwd`) entonces simplemente los llamamos:

```

misdatos <- readRDS("misdatos.rds")
# Los borramos
file.remove("misdatos.rds")

```

## separado por \*

Con esto nos referimos a la colección de archivos en texto plano, es decir, `.txt`, `.tsv`, `.psv`, etcétera.

Para **escribirlos** el mas común es `write.table` del paquete `utils` (R Core Team 2016b)

```

# Do not run
write.table(misdatos, file = "~/misdatos.<extension>", append = FALSE
           , quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = "."
           , row.names = TRUE, col.names = TRUE
           , qmethod = c("escape", "double"), fileEncoding = "")

```

En el paquete `readr` se implementa también `write_delim`

```

# Do not run
write_delim(misdatos, path = "~/misdatos.<extension>"
           , delim = " ", na = "NA", append = FALSE, col_names = !append)

```

Escribamos ahora el `dataframe` `misdatos` en `psv`:

```

write_delim(misdatos, path = "misdatos.psv", delim = "|")

```

Para **leerlos** `read.table` del paquete `utils` (R Core Team 2016b) nos permite especificar casi cualquier particularidad en un archivo de texto plano.

```
# Do not run
misdatos <- read.table("~/misdatos.<extension>", header = FALSE
  , sep = ",", quote = "\"", dec = "."
  , numerals = c("allow.loss", "warn.loss", "no.loss")
  , row.names, col.names, as.is = !stringsAsFactors
  , na.strings = "NA", colClasses = NA, nrows = -1
  , skip = 0, check.names = TRUE
  , fill = !blank.lines.skip, strip.white = FALSE
  , blank.lines.skip = TRUE, comment.char = "#"
  , allowEscapes = FALSE, flush = FALSE
  , stringsAsFactors = default.stringsAsFactors()
  , fileEncoding = "", encoding = "unknown", text
  , skipNul = FALSE)
```

La función `read_delim` del paquete `readr` (Hadley Wickham, Hester y Francois 2016) lee los datos más eficientemente a un objeto de clase `tibble`.

```
# Do not run
misdatos <- read_delim(file = "~/misdatos.<extension>", delim
  , quote = "\"", escape_backslash = FALSE
  , escape_double = TRUE, col_names = TRUE
  , col_types = NULL, locale = default_locale()
  , na = c("", "NA"), quoted_na = TRUE, comment = ""
  , trim_ws = FALSE, skip = 0, n_max = Inf
  , guess_max = min(1000, n_max)
  , progress = interactive())
```

Leemos el archivo `.psv` que creamos antes:

```
misdatos <- read_delim(file = "misdatos.psv", delim = "|")
# Los borramos
file.remove("misdatos.psv")
```

## csv (archivo separado por comas)

Este es un caso particular de archivos de texto en el que se separan por comas. Como es muy utilizado, generalmente se hacen funciones donde ya se especifica el delimitador. Guardaremos el *data frame* `misdatos` en el directorio “arriba” de la ruta que se muestra usando `getwd`. Esto lo podemos hacer anteponiendo al nombre del archivo con `../`.

Para **escribirlos**

```
# utils
write.csv(misdatos, file = "../misdatos.csv", row.names = F)
# readr
write_csv(misdatos, path = "../misdatos.csv", na = "NA", append = FALSE)
```

Observa en el explorador de archivos en dónde es que se guardó el archivo `misdatos.csv`.

Para **leerlos**, seguimos usando rutas relativas.

```
# utils - como data.frame
misdatos <- read.table("../misdatos.csv", header=TRUE,
  sep=",")

misdatos <- read.csv("../misdatos.csv")
```

```
# readr - como tibble
misdatos <- read_csv("../misdatos.csv")

# Lo borro
file.remove("../misdatos.csv")
```

## Microsoft Excel

Para **escribirlos** dentro del paquete `xlsx` usamos la función `write.xlsx`

```
misdatos <- iris
write.xlsx(misdatos, "misdatos.xlsx", row.names = F)
```

Para **leerlos** dentro del paquete `readxl` se encuentra la función `read_excel` que es muy útil en este caso.

```
misdatos <- read_excel("misdatos.xlsx", sheet = 1, col_names = TRUE,
col_types = NULL, na = "", skip = 0)

# Lo borro
file.remove("misdatos.xlsx")
```

## dbf

Extensión que representa un archivo de una base de datos (*database file*).

Para **escribirlos**:

```
write.dbf(as.data.frame(misdatos), "misdatos.dbf")
```

Nota cómo tuvimos que coercionar el objeto a *data frame*. Como en el ejemplo anterior leímos un *tibble* y el paquete `foreign` es más viejo (y no conoce los *tibbles*) entonces le mandamos un objeto que sí conoce.

Veremos más adelante la ventaja de usar *tibbles* aún cuando de vez en cuando se tienen problemas de compatibilidad.

Para **leerlos**:

```
misdatos <- read.dbf("misdatos.dbf")
# Lo borro
file.remove("misdatos.dbf")
```

## IBM SPSS

SPSS guarda los datos bastante bien: si uno les pone etiquetas entonces tiene el valor y las etiquetas para factores, etc. Este tipo de cosas, si ya fueron realizadas por alguien más, es una pena perderlas al convertirlo en un csv o un excel. En R no lo pierdes.

Para **escribirlos**

```
# haven
write_sav(data = misdatos, path = "misdatos.sav")
```

Para **leerlos**

```
# haven - como tibble
misdatos <- read_sav(file = "misdatos.sav", user_na = FALSE)

# Lo borro
file.remove("misdatos.sav")
```

## Stata

### HOME DIRECTORY

El directorio (carpeta) *home* es muy utilizado. Normalmente, se le denota como  $\sim$  y es en donde un sistema operativo guarda los archivos del usuario que se encuentra en sesión. Dependiendo del sistema operativo que utilices, encontrarás este directorio en una ruta específica.

En Microsoft Windows Vista 7, 8 y 10 lo encuentras en  $\langle \text{root} \rangle \backslash \text{Users} \backslash \langle \text{username} \rangle$ .

En Linux lo encuentras en  $/\text{home}/\langle \text{username} \rangle$ .

En Mac OS X lo encuentras en  $/\text{Users}/\langle \text{username} \rangle$ .

Para **escribirlos** en Stata primero tenemos que cambiar los nombres de las variables en el *data frame* pues Stata no admite puntos en los nombres:

```
names(misdatos) <- tolower(gsub("\\\\.", "_", names(misdatos)))
# haven
write_dta(data = misdatos, path = "~/misdatos.dta", version = 14)
```

Para leerlos

```
# haven - como tibble
misdatos <- read_dta(file = "~/misdatos.dta", encoding = NULL)

# Lo borramos
file.remove("~/misdatos.dta")
```

## SAS

Para **escribirlos** también debemos asegurarnos que los nombres de variables estén compuestos por letras, números o guiones bajos

```
misdatos <- iris
names(misdatos) <- tolower(gsub("\\\\.", "_", names(misdatos)))
# haven
write_sas(data = misdatos, path = "~/misdatos.sas7bdat")
```

Para leerlos

```
# haven - como tibble
misdatos <- read_sas(data_file = "~/misdatos.sas7bdat", catalog_file = NULL, encoding = NULL)

# Lo borro
file.remove("~/misdatos.sas7bdat")
```

Para usar el paquete **foreign** ejemplificaremos la creación de un directorio de archivos en tu computadora desde R:

```
# Creamos un directorio llamado datos
dir.create("datos_sas")
```

Observa como, en el directorio que se despliega con `getwd` encuentras ahora una carpeta llamada `datos_sas`. Creamos ahí un archivo de texto plano con los datos y un `script` de `SAS` para leerlos apropiadamente:

```
# foreign - escribe los datos como texto y un programa para leerlos
write.foreign(misdatos, "datos_sas/misdatos.txt", "datos_sas/misdatos.sas", package="SAS")
```

Observa desde el explorador de archivos, los dos objetos creados. También desde R podemos borrar el directorio:

```
unlink("datos_sas", recursive = T, force = FALSE)
```

La bandera `recursive` le dice al sistema que borre todo lo contenido en esa carpeta.

## Google Spreadsheet

Para hacer este ejercicio, debes tener una cuenta de `gmail`.

Primero, deberás autenticarte. Esto lo puedes hacer en cualquier sesión interactiva utilizando alguna función del paquete `googlesheets`

```
gs_ls()
```

En la consola de R te aparece:

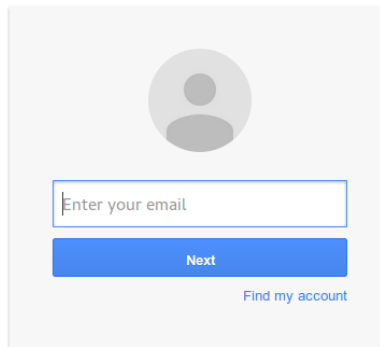
```
> library(googlesheets)
> gs_ls()
Waiting for authentication in browser...
Press Esc/Ctrl + C to abort
```

Se abrirá una ventana del explorador y deberás introducir tus credenciales de tu cuenta de `gmail`

# Google

## One account. All of Google.

Sign in with your Google Account

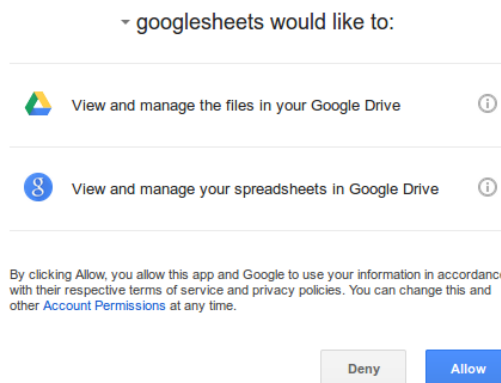
A sign-in form with a grey circular profile picture placeholder at the top. Below it is a text input field with the placeholder text "Enter your email". Under the input field is a blue button labeled "Next". To the right of the "Next" button is a link that says "Find my account".

[Create account](#)

One Google Account for everything Google



Después de poner tus credenciales, te aparecerá un mensaje pidiendo acceso a tus datos en **drive**:

A permissions screen titled "googlesheets would like to:". It contains two permission items, each with an icon, a description, and an information icon. The first item has a Google Drive icon and the text "View and manage the files in your Google Drive". The second item has a Google Sheets icon and the text "View and manage your spreadsheets in Google Drive". At the bottom, there is a paragraph of legal text and two buttons: "Deny" and "Allow".

Al aceptar darle acceso, recibirás un mensaje parecido a *Authentication complete. Please close this page and return to R.*

Ahora verás en la consola de R un listado de las **google spreadsheets** en tu cuenta de gmail.

Ahora, vamos a **escribir** una nueva hoja en tu cuenta.

```
gs_new("misdatos", ws_title = "mihoja", input = head(iris), trim = TRUE, verbose = FALSE)
```

Si vas a tu google drive, deberás ver que se creó un nuevo elemento que se ve así:



	A	B	C	D	E
1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
2		5.1	3.5	1.4	0.2
3		4.9	3.0	1.4	0.2
4		4.7	3.2	1.3	0.2
5		4.6	3.1	1.5	0.2
6	5.0		3.6	1.4	0.2
7		5.4	3.9	1.7	0.4

Agrega 1000 más filas al final.

De igual forma, puedes ahora **leer** los datos de cualquier **google spreadsheet** que tengas en tu cuenta.

```
misdatos <- gs_read(gs_title("misdatos"), ws = "mihoja")
# La borro
gs_delete(gs_title("misdatos"))
```

## Google bigquery

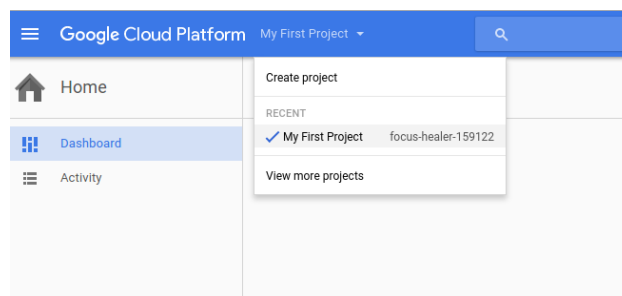
Google **bigquery** es un *data warehouse* que permite guardar grandes bases de datos. Al contratar el servicio, **google** se encarga del *hardware* y la infraestructura necesaria para que su procesamiento sea rápido [whatisbigquery].

Para guardar tus datos en **bigquery** debes crear un proyecto en la consola de desarrolladores.

Existen varias bases de dato públicas disponibles. Para poder utilizarlas, necesitas tener una cuenta. Puedes empezar una prueba gratis en la página de google cloud platform. Verás una pantalla como esta:



Sigue las instrucciones y eventualmente llegarás a una pantalla como esta



Copia el identificador de tu proyecto para que puedas realizar **queries** (llamadas a las bases de datos).

Leemos la base de datos pública de natalidad en Estados Unidos.

```
project <- "focus-healer-159122" # pon tu projectID aquí

sql <- 'SELECT year, count(*) as babies, avg(mother_age) as mother_age_avg
FROM[publicdata:samples.natality]
WHERE year > 1980 and year < 2006
group by year;'

data <- query_exec(query = sql, project = project)
```

Nota como la tabla cuenta con aprox. 140 millones de registros y se obtiene el detalle en segundos.

## rdata

También es posible guardar objetos específicos del ambiente dentro de un formato especial con extensión **rdata** o **RData**. Esto es muy útil cuando no han acabado o quieren seguir trabajando con algo.

Para **escribirlos**

```
save(...,  
      file = "~/misdatos.rdata",  
      ascii = FALSE, version = NULL, envir = parent.frame(),  
      compress = isTRUE(!ascii), compression_level,  
      eval.promises = TRUE, precheck = TRUE)
```

Nota como ... pueden ser uno o más objetos de R.

Para leerlos

```
load("~/misdatos.rdata")
```

Los objetos se cargarán al ambiente con los nombres con los que fueron guardados.