R: Importación de datos

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.

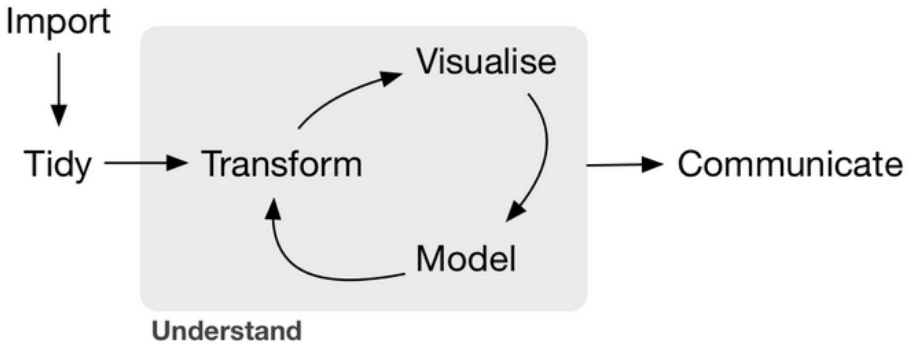

Figura 1: Modelo de las herramientas que se necesitan en un proyecto de datos según **grolemund2016r**

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[1]:

[1]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.

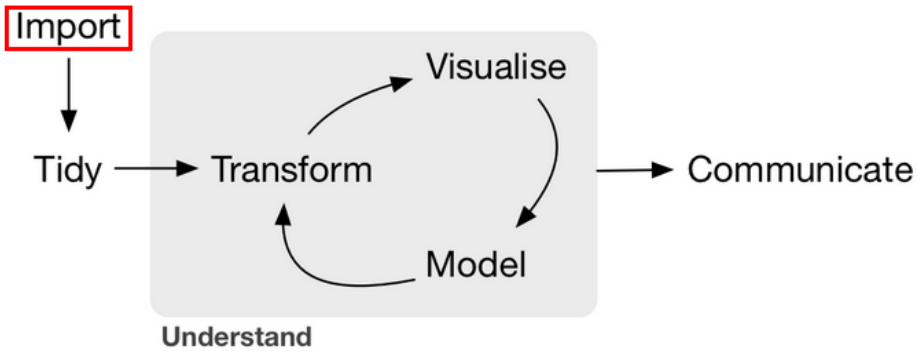

Figura 2: Importación en el análisis de datos **grolemund2016r**

Formato	Lectura	Escritura
rds	base::readRDS	base::saveRDS
separado por *	utils::read.table; **readr::read_delim**	utils::write.table; **readr::write_delim**
csv		
Microsoft Excel		
IBM SPSS		
Stata		
SAS		
Google spreadsheet		
Google bigquery		
Heroku Postgres		

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.

`R` tiene conexion con basicamente todos los tipos de archivo. Veremos algunos de los mas relevantes.

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` (**rbase**). Permiten guardar un solo objeto de `R` a un archivo y recuperarlo.

Para **leerlos**

```
misdatos <- readRDS("~/misdatos_locales.rds")
```

Para **escribirlos**

```
saveRDS(misdatos, file = "~/misdatos_locales.rds", ascii = FALSE, version = NULL,
        compress = TRUE, refhook = NULL)
```


rdata

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`.

separado por *

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

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

```
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` (**readr**) lee los datos más eficientemente a un objeto de clase `tibble`.

```
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())
```

Para **escribirlos** el mas comun es `write.table` del paquete `utils` (**utils**)

```
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 un

csv (archivo separado por comas)

Para **leerlos**

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

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

Para **escribirlos**

```
# utils
write.csv(misdatos, file = "~/misdatos.csv")
# readr
write_csv(misdatos, path = "~/misdatos.csv", na = "NA", append = FALSE, col_names = !append)
```

Microsoft Excel

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

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

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

```
write.xlsx(misdatos, "~/misdatos.xlsx")
```

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 realizados por alguien mas, es una pena perderlos al convertirlo en un csv o un excel.

Para **leerlos**

```
# foreign - como data.frame
misdatos <- read.spss("~/misdatos.sav", use.value.labels = TRUE,
          max.value.labels = Inf, trim.factor.names = FALSE,
          trim_values = TRUE, reencode = NA)
# haven - como tibble
```

Para **escribirlos**

```
# foreign - escribe los datos como texto y un programa para leerlos
write.foreign(misdatos, "~/misdatos.txt", "c:/misdatos.sps",   package="SPSS")
```

Stata

Para **leerlos**

```
# foreign - como data.frame
misdatos <- read.dta("~/misdatos.dta", convert.dates = TRUE, convert.factors = TRUE,
         missing.type = FALSE,
         convert.underscore = FALSE, warn.missing.labels = TRUE)
# haven - como tibble
```

Para **escribirlos**

```
# foreign
write.dta(misdatos, "~/misdatos.dta")
```

SAS

Para **leerlos**

```
# foreign - como data.frame

# haven - como tibble
```

Para **escribirlos**

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

Google Spreadsheet

Google bigquery

Heroku Postgres