

Importar y limpiar datos en R

Programación para el análisis de datos

Departamento de Ciencias Sociales, UCU - Martín Opertti

Segunda mitad de curso

Semana	Clase
2022-10-11	Importar y limpiar datos en R
2022-10-13	Importar y limpiar datos en R
2022-10-18	Estadística descriptiva en R
2022-10-20	Estadística descriptiva en R
2022-10-25	Manipulación de datos en R
2022-10-27	Manipulación de datos en R
2022-11-01	Manipulación de datos en R II
2022-11-03	Visualización de datos en R
2022-11-08	Visualización de datos en R
2022-11-10	Estadística inferencial en R
2022-11-15	Parcial R
2022-11-17	R Markdown + Pauta trabajo final
2022-11-22	Programación avanzada
2022-11-24	Taller / Python
2022-11-29	Presentaciones
2022-12-01	Presentaciones y cierre

Directorios de trabajo y proyectos de R (.Rproj)

Directorios de trabajo

- Para abrir en R un archivo guardado en tu computadora, debes especificar en qué carpeta está guardado, para esto hay varias opciones. Primero, puedes fijar un directorio por defecto:

```
# Puedo fijar el directorio de trabajo con la función setwd()  
# Fijar la carpeta donde vamos a importar y exportar los archivos:  
setwd("micompu/micarpeta")  
getwd() # Con esta función puedo consultar el directorio
```

```
# Ahora, si quiero leer un archivo que esté en "micompu/micarpeta" simplemente  
# escribo su nombre dentro de la función, en el lugar del "path".
```

```
# Supongamos que tengo dentro de la carpeta "micarpeta" un excel con datos  
# de desempleo en Uruguay:
```

```
library(readxl)  
desempleo_uru <- read_excel("data/desempleo.xlsx")  
head(desempleo_uru, 4)
```

```
## # A tibble: 4 x 2  
##   Year  tasa  
##   <dbl> <dbl>  
## 1  1990   8.5  
## 2  1991   8.9  
## 3  1992    9  
## 4  1993   8.3
```

Directorios de trabajo

También podemos no fijar un directorio para la sesión e ir especificando los directorios completos dentro de cada función:

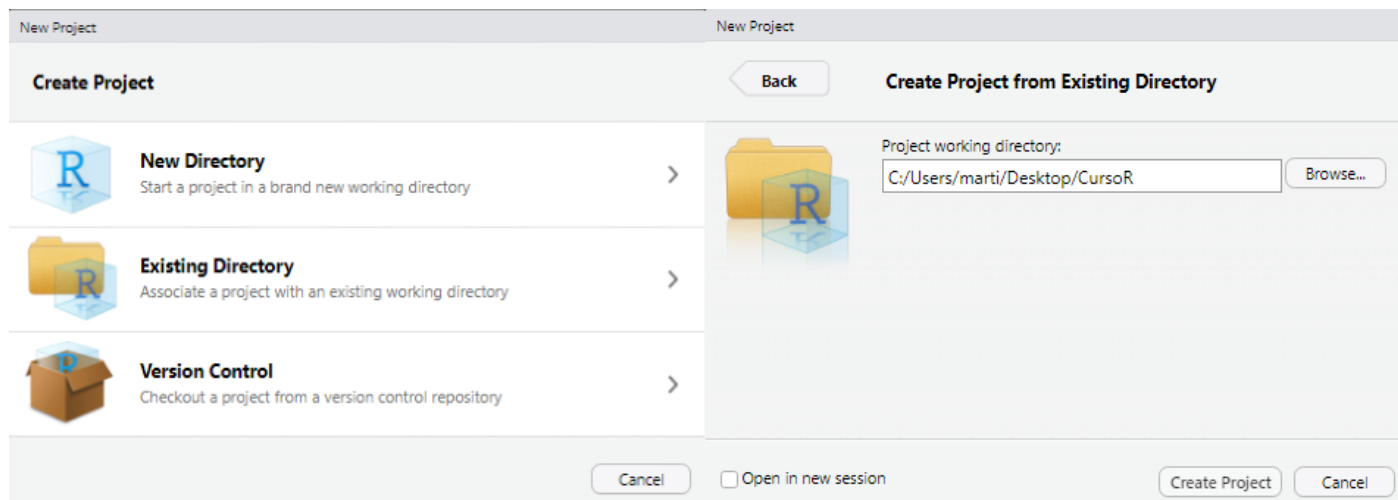
```
desempleo_uru <- read_excel("micompu/micarpeta/data/desempleo.xlsx")
```

Proyectos de R (.Rproj)

- La mejor práctica para que nuestros scripts sean portables y reproducibles, es utilizar R Projects (.Rproj).
- Para crear un .Rproj vamos a `File/New Project` y ahí nos encontramos con la opción de crear una carpeta para guardar los archivos o utilizar una carpeta ya existente.
- Al crear un proyecto de R se creará un archivo de extensión .Rproj, cuando le damos click se inicia una nueva sesión de R cuyo directorio es por defecto la carpeta en la que está guardado.
- Podemos usar directorios relativos dentro de la carpeta en la que se aloja nuestro .Rproj para importar y exportar datos a y desde R. Esto hace que uno pueda cambiar la carpeta o compartirla y el script correrá de igual manera (a diferencia de si utilizamos `setwd()`)

Proyectos de R (.Rproj)






Dentro de la carpeta del curso que ya crearon creen un proyecto de R. Para eso abran RStudio y desde ahí seleccionen `File/New Project` y seleccionen la opción "Existing Directory" y luego seleccionen la carpeta del curso.



Proyectos de R (.Rproj)

Deberían ver en su carpeta algo así:

> CursoR

<input type="checkbox"/> Name	Date modified	Type	Size
 data	15/07/2021 18:06	File folder	
 graficos	15/07/2021 18:06	File folder	
 resultados	15/07/2021 18:06	File folder	
 scripts	15/07/2021 18:06	File folder	
 CursoR	15/07/2021 18:07	R Project	1 KB

Proyectos de R (.Rproj)

- Abran el archivo `.Rproj` y desde ahí usando File/Open File abren los scripts dentro de la carpeta "scripts". Es importante que los abran desde la sesión que inicia el proyecto y no directamente haciendo click en el script.
- Ya estamos listos para empezar!

Dialectos

Ejercicio

Supongamos que tengo estos datos:

```
data
```

```
## # A tibble: 35 x 5
##   year      gdp_lcu inflation unemployment presidente
##   <dbl>      <dbl>      <dbl>          <dbl> <chr>
## 1  1985 249277574100      72.2            NA Sanguinetti
## 2  1986 271238450500      76.4            NA Sanguinetti
## 3  1987 292918912000      63.6            NA Sanguinetti
## 4  1988 297256857900      62.2            NA Sanguinetti
## 5  1989 300538279400      80.4            NA Sanguinetti
## 6  1990 301431925100     113.            NA Lacalle
## 7  1991 312099023700     102.            9.01 Lacalle
## 8  1992 336853433700      68.5            8.98 Lacalle
## 9  1993 345805469000      54.1            8.94 Lacalle
## 10 1994 370984750100      44.7             9 Lacalle
## # ... with 25 more rows
```

Ejercicio

¿Qué quiero hacer con el código debajo?

```
as.data.frame(t(sapply(X = split(
  x = data[which(data$presidente %in% c("Vázquez", "Sanguinetti")),
    which(colnames(data) %in% c("gdp_lcu", "inflation"))],
  f = data$presidente[which(data$presidente %in% c("Vázquez", "Sanguinetti"))],
  drop = TRUE),
  FUN = function(x) {apply(x, 2, mean)})))
```

Ejercicio

¿Qué quiero hacer con el código debajo?

```
data_dt <- data  
  
setDT(data_dt)  
  
data_dt[presidente %in% c("Vázquez", "Sanguinetti"),  
        c("presidente", "gdp_lcu", "inflation") ][  
        , lapply(.SD, mean), by = presidente]
```

Ejercicio

¿Qué quiero hacer con el código debajo?

```
data %>%  
  filter(presidente %in% c("Vázquez", "Sanguinetti")) %>%  
  select(presidente, gdp_lcu, inflation) %>%  
  group_by(presidente) %>%  
  summarise_all(mean)
```

R Base

```
as.data.frame(t(sapply(X = split(
  x = data[which(data$presidente %in% c("Vázquez", "Sanguinetti")),
    which(colnames(data) %in% c("gdp_lcu", "inflation"))],
  f = data$presidente[which(data$presidente %in% c("Vázquez", "Sanguinetti")),
  drop = TRUE),
  FUN = function(x) {apply(x, 2, mean)})))
```

```
##                gdp_lcu inflation
## Sanguinetti 344923753631 46.168819
## Vázquez      584455715198  7.416316
```

Data.table

```
data_dt <- data
setDT(data_dt)
data_dt[presidente %in% c("Vázquez", "Sanguinetti"),
        c("presidente", "gdp_lcu", "inflation") ][
        , lapply(.SD, mean), by = presidente]
```

```
##      presidente      gdp_lcu inflation
## 1: Sanguinetti 344923753631 46.168819
## 2:   Vázquez 584455715198  7.416316
```


Tidyverse

```
data %>%  
  filter(presidente %in% c("Vázquez", "Sanguinetti")) %>%  
  select(presidente, gdp_lcu, inflation) %>%  
  group_by(presidente) %>%  
  summarise_all(mean)
```

```
## # A tibble: 2 x 3  
##   presidente      gdp_lcu inflation  
##   <chr>          <dbl>     <dbl>  
## 1 Sanguinetti 344923753631.    46.2  
## 2 Vázquez    584455715198.     7.42
```

Dialectos

- Como vimos, en R podemos realizar una misma operación de muchas maneras distintas. Puesto de otra manera, R como lenguaje de programación tiene distintos "dialectos", esto es, paquetes (o conjuntos de paquetes) con sus propias funciones, sintaxis y comunidad de usuarios.
- Para la mayoría de las funciones requeridas para un análisis de datos estándar (importar datos, manipular, modelar y visualizar) existen -de forma muy simplificada- tres grandes dialectos: [R Base](#), [tidyverse](#) y [data.table](#).
- Tidyverse es una colección de paquetes diseñados para el análisis de datos. Este conjunto de paquetes comparte una filosofía de diseño, gramática y estructura de datos.
- Las ventajas de Tidyverse están en su gramática (fácil de leer lo que invita a compartir y replicar), consistencia, alcance y su numerosa y creciente comunidad.

Dialectos (ejemplo)

```
encuesta # Retomemos el data.frame "encuesta"
```

```
##   edad ideologia      voto
## 1   18 Izquierda Partido A
## 2   24 Izquierda Partido A
## 3   80  Derecha Partido C
```

```
# Supongamos que quiero quedarme solo con las variables de edad y voto
```

```
encuesta_base <- encuesta[ , c("edad", "voto")] # R Base
colnames(encuesta_base)
```

```
## [1] "edad" "voto"
```

```
encuesta_dt <- as.data.table(encuesta)[ , .(edad, voto)] # Datatable
colnames(encuesta_dt)
```

```
## [1] "edad" "voto"
```

```
encuesta_tidy <- select(encuesta, edad, voto) # Tidyverse
colnames(encuesta_tidy)
```

```
## [1] "edad" "voto"
```

Tidyverse

Tidyverse

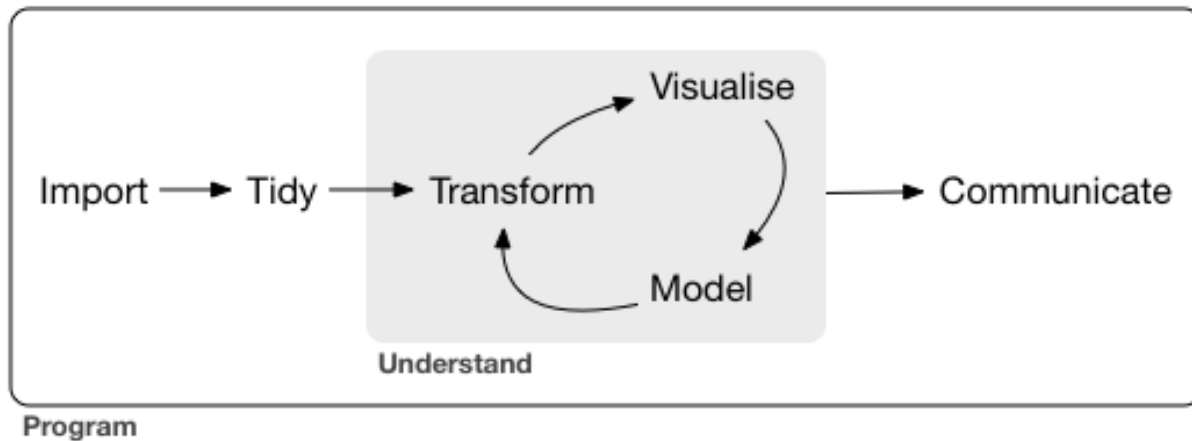
Tidyverse cuenta con varios paquetes que sirven para distintos tipos de tareas específicas. Podemos cargar todos los paquetes de forma conjunta:

```
# install.packages("tidyverse")  
library(tidyverse)  
  
# install.packages("dplyr")  
library(dplyr)
```



Tidyverse

La mejor manera de entender los principios de tidyverse es a través del libro del creador de tidyverse (Hadley Wickham) y Garrett Golemund "R for Data Science de Hadley Wickham" (2018).



Tidyverse: paquetes

Paquetes que Tidyverse carga:

- **readr**: importar y exportar datos
- **dplyr**: manipulación de datos
- **tidyr**: manipulación de datos
- **ggplot2**: visualización de datos
- **purrr**: programación avanzada
- **tibble**: estructura de datos
- **forcats**: factores
- **stringr**: variables de caracteres



Tidyverse: paquetes

Estos son algunos paquetes (para tareas más específicas) que forman parte del Tidyverse pero se tienen que cargar por separado:

- **readxl**: importar datos (excel)
- **haven**: importar (Stata, SPSS, SAS)
- **lubridate**: manipulación de fechas
- **rvest**: webscrapping
- **glue**: combinar data
- **tidymodels**: modelar datos

Importar y exportar datos

Importar datos

- Hasta ahora trabajamos principalmente con datos ingresados manualmente con las funciones `c()` y `data.frame()`
- Normalmente cuando trabajamos con datos solemos utilizar datos ya creados guardados en los formatos de otros programas (ej. Excel, Stata, SPSS)
- Existen varios paquetes que permiten importar y exportar datos desde distintos formatos. Algunos de los más utilizados son `readr`, `haven`, `readxl` y `utils`

Importar datos desde distintos formatos

Distintas funciones nos sirven para importar datos a R desde distintos formatos. Veamos algunos ejemplos:

```
# Con la función read_csv() del paquete readr importamos archivos .csv
```

```
library(tidyverse)
```

```
gapminder_csv <- read_csv("data/gapminder.csv")
```

```
# Con la función read_excel() del paquete readxl importamos archivos excel
```

```
library(readxl)
```

```
gapminder_excel <- read_excel("data/gapminder.xlsx")
```

```
# Vemos que los dataframes son iguales, tienen la mismas filas y columnas  
dim(gapminder_csv)
```

```
## [1] 1704    6
```

```
dim(gapminder_excel)
```

```
## [1] 1704    6
```

Importar datos desde paquetes

Algunos paquetes incluyen datos, por ejemplo, gapminder. En la documentación del paquete se encuentra el nombre de los datos. Con una simple asignación los podemos cargar

```
library(gapminder)
```

```
data_gapminder <- gapminder  
head(data_gapminder)
```

```
## # A tibble: 6 x 6  
##   country      continent  year lifeExp      pop gdpPercap  
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>  
## 1 Afghanistan Asia      1952   28.8  8425333    779.  
## 2 Afghanistan Asia      1957   30.3  9240934    821.  
## 3 Afghanistan Asia      1962   32.0 10267083    853.  
## 4 Afghanistan Asia      1967   34.0 11537966    836.  
## 5 Afghanistan Asia      1972   36.1 13079460    740.  
## 6 Afghanistan Asia      1977   38.4 14880372    786.
```

Importar datos en otros formatos

También es posible importar datos guardados en los formatos de otros softwares estadísticos como SPSS o Stata. Para esto usaremos el paquete `haven`.

```
library(haven)

# SPSS
gapminder_spss <- read_spss("data/gapminder.sav")

# STATA
gapminder_stata <- read_stata("data/gapminder.dta")
```

O podríamos llamar a la función y paquete dado que generalmente solo utilizamos una función de los paquetes que cargan datos (depende del caso obviamente)

```
# SPSS
gapminder_spss <- haven::read_spss("data/gapminder.sav")

# STATA
gapminder_stata <- haven::read_stata("data/gapminder.dta")
```

Importar datos en formato R

R también cuenta con sus propios formatos de almacenamiento de datos (`.rds` y `.Rdata` o `.rda`). Este enfoque es poco práctico si queremos usar los datos almacenados en otro programa, pero muy útil si solamente usaremos R dado que mantiene la información tal cual estaba en R (por ej. tipos de variables o atributos):

```
# Para esto no necesitamos cargar paquetes.  
# Guardar un objeto como .rds:  
saveRDS(object = data_gapminder,  
        file = here::here("resultados/data_gapminder.rds"))  
  
# Leemos un archivo .rds  
miobjeto_rds <- readRDS(file = here::here("resultados/data_gapminder.rds"))  
  
# Con .rda se pueden guardar varios objetos al mismo tiempo!  
# Exportamos un archivo .Rdata  
save(data_gapminder, miobjeto_rds,  
     file = here::here("resultados/dos_dataframes.Rdata"))  
  
# Importamos un archivo .Rdata  
load(here::here("resultados/dos_dataframes.Rdata"))
```

Exportar datos

- También podemos guardar archivos desde R en otros formatos.
- Con **readr** podemos exportar archivos en formato .csv
- Con **writexl** podemos exportar directamente un excel.
- Con **haven** podemos exportar archivos en formato .dta (Stata) y .sav (SPSS)

```
# Guardar .csv
library(gapminder)
data_gapminder <- gapminder
write_excel_csv(data_gapminder, here::here("resultados/gapminder.csv"))

# Guardar excel
library(writexl)
write_xlsx(data_gapminder, here::here("resultados/gapminder.xlsx"))

# Guardar .dta (Stata)
library(haven)
write_dta(data_gapminder, here::here("resultados/gapminder.dta"))

# Guardar .sav (SPSS)
write_sav(data_gapminder, here::here("resultados/gapminder.sav"))

# Guardar .sas (SAS)
write_sas(data_gapminder, here::here("resultados/gapminder.sas"))
```

Argumentos

Argumentos a tener en cuenta:

- **Nombre de columnas:** a veces debemos especificar si queremos que la primera fila de nuestros datos sean el nombre de las variables
- **Nombre de filas:** de igual manera, a veces podemos especificar si queremos que la primera columna sea el nombre de las filas (sirve para identificadores de caso por ej.)
- **Etiquetas de variables:** cuando los datos que queremos importar tienen etiquetas (pasa mucho en encuestas) podemos cargarlas como etiquetas o cargar solamente la etiqueta como cadena o factores. Ver capítulo 4 de Urdinez, F. & Labrin, A. (Eds.) (2020)
- **Append:** algunas funciones permiten agregar filas debajo de un archivo (esto es muy útil para ir actualizando bases de datos)

Ejercicio

En la carpeta data encontrarán un archivo excel llamado "urudata_sheets", deben leer la segunda hoja del archivo

Etiquetas

Etiquetas cuando importamos datos

- Cuando importamos datos que tienen etiquetas (por ejemplo de formatos como Stata o SPSS) debemos tener cuidado con cómo manejar estas etiquetas
- Por ejemplo, supongamos que queremos leer los datos de una encuesta con dos variables, guardada en formato Stata (.dta), con el paquete `haven`:

```
data <- haven::read_stata("data/ej_encuesta.dta")
head(data, 5)
```

```
## # A tibble: 5 x 2
##           P1           P14
##   <dbl+lbl> <dbl+lbl>
## 1 4 [Colonia] 1 [Muy mala]
## 2 18 [Tacuarembó] 2 [Mala]
## 3 15 [Salto] 5 [Muy buena]
## 4 1 [Artigas] 3 [Ni buena ni mala]
## 5 10 [Montevideo] 1 [Muy mala]
```

- Por defecto se leen como variables de tipo `double` (numérica) con etiquetas como atributos

Etiquetas cuando importamos datos

Si queremos quedarnos directamente con las etiquetas, podemos utilizar la función `as_factor`:

```
data <- haven::read_stata("data/ej_encuesta.dta") %>%  
  haven::as_factor()  
head(data, 5)
```

```
## # A tibble: 5 x 2  
##   P1          P14  
##   <fct>      <fct>  
## 1 Colonia   Muy mala  
## 2 Tacuarembó Mala  
## 3 Salto     Muy buena  
## 4 Artigas   Ni buena ni mala  
## 5 Montevideo Muy mala
```

Factores

Factores

- Otro tipo de variables en R son los factores (factors), utilizados para representar data categórica. Estos suelen confundirse con las variables de caracteres pero tienen algunas diferencias.
- Normalmente los factores son utilizados para las variables de caracteres con un número de valores posibles fijo y cierto orden (opcional)
- A R le gusta transformar las variables de caracteres en factores al importarlas (si usamos R Base particularmente).
- El paquete **forcats** (dentro del Tidyverse) ayuda a manejar variables de caracteres y factores:
 - `fct_relevel()` cambia manualmente el orden de los niveles
 - `fct_reorder()` cambia el orden de los niveles de acuerdo a otra variable
 - `fct_infreq()` reordena un factor por la frecuencia de sus valores
 - `fct_lump()` colapsa los valores menos frecuentes en otra categoría "other". Es muy útil para preparar datos para tablas y gráficos

Transformar factores

```
# Podemos chequear y coercionar factores  
data_gapminder <- gapminder  
is.factor(data_gapminder$continent) # Chequeo si es factor
```

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

```
levels(data_gapminder$continent) # Chequeo los niveles
```

```
## [1] "Africa" "Americas" "Asia" "Europe" "Oceania"
```

```
# Transformo a character  
data_gapminder$continent <- as.character(data_gapminder$continent)  
class(data_gapminder$continent)
```

```
## [1] "character"
```

```
# De vuelta a factor  
data_gapminder$continent <- as.factor(data_gapminder$continent)  
class(data_gapminder$continent)
```

```
## [1] "factor"
```

Crear y ordenar factores

```
# Para crear un factor usamos la función factor()  
países_mercosur <- factor(c("Argentina", "Brasil", "Paraguay", "Uruguay"))  
table(países_mercosur)
```

```
## países_mercosur  
## Argentina      Brasil  Paraguay  Uruguay  
##           1           1           1           1
```

```
# La función fct_relevel() nos permite reordenar los niveles del factor  
países_mercosur <- fct_relevel(países_mercosur, "Uruguay")  
table(países_mercosur)
```

```
## países_mercosur  
## Uruguay Argentina      Brasil  Paraguay  
##           1           1           1           1
```


Dataframes

Dataframes: tibbles

- La mayoría de los análisis de datos convencionales contienen dataframes. Cuando usamos los paquetes del tidyverse, generalmente trabajamos con "tibbles", que es muy similar a un dataframe pero con pequeños cambios.
- Una de las principales diferencias es la forma en que se imprimen los datos.
- La mayoría de las funciones del Tidyverse devuelven un tibble.

```
data_gapminder <- (gapminder)
class(data_gapminder) # Ya es un tibble
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

```
data_gapminder <- as.data.frame(data_gapminder)
class(data_gapminder) # Ahora solamente dataframe
```

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

Dataframes: imprimir dataframe

```
print(data_gapminder)
```

```
##           country continent year  lifeExp      pop  gdpPercap
##  1      Afghanistan      Asia  1952  28.80100    8425333    779.4453
##  2      Afghanistan      Asia  1957  30.33200    9240934    820.8530
##  3      Afghanistan      Asia  1962  31.99700   10267083    853.1007
##  4      Afghanistan      Asia  1967  34.02000   11537966    836.1971
##  5      Afghanistan      Asia  1972  36.08800   13079460    739.9811
##  6      Afghanistan      Asia  1977  38.43800   14880372    786.1134
##  7      Afghanistan      Asia  1982  39.85400   12881816    978.0114
##  8      Afghanistan      Asia  1987  40.82200   13867957    852.3959
##  9      Afghanistan      Asia  1992  41.67400   16317921    649.3414
## 10      Afghanistan      Asia  1997  41.76300   22227415    635.3414
## 11      Afghanistan      Asia  2002  42.12900   25268405    726.7341
## 12      Afghanistan      Asia  2007  43.82800   31889923    974.5803
## 13          Albania     Europe  1952  55.23000    1282697   1601.0561
## 14          Albania     Europe  1957  59.28000    1476505   1942.2842
## 15          Albania     Europe  1962  64.82000    1728137   2312.8890
## 16          Albania     Europe  1967  66.22000    1984060   2760.1969
## 17          Albania     Europe  1972  67.69000    2263554   3313.4222
## 18          Albania     Europe  1977  68.93000    2509048   3533.0039
## 19          Albania     Europe  1982  70.42000    2780097   3630.8807
## 20          Albania     Europe  1987  72.00000    3075321   3738.9327
## 21          Albania     Europe  1992  71.58100    3326498   2497.4379
## 22          Albania     Europe  1997  72.95000    3428038   3193.0546
## 23          Albania     Europe  2002  75.65100    3508512   4604.2117
## 24          Albania     Europe  2007  76.42300    3600523   5937.0295
## 25          Algeria     Africa  1952  43.07700    9279525   2449.0082
## 26          Algeria     Africa  1957  45.68500   10270856   3013.9760
## 27          Algeria     Africa  1962  48.30300   11000948   2550.8169
## 28          Algeria     Africa  1967  51.40700   12760499   3246.9918
## 29          Algeria     Africa  1972  54.51800   14760787   4182.6638
```

Dataframes: imprimir tibble

```
data_gapminder <- as_tibble(data_gapminder) # Pasamos nuevamente a tibble
class(data_gapminder)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

```
print(data_gapminder)
```

```
## # A tibble: 1,704 x 6
##   country      continent  year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Afghanistan Asia      1952   28.8  8425333    779.
## 2 Afghanistan Asia      1957   30.3  9240934    821.
## 3 Afghanistan Asia      1962   32.0 10267083    853.
## 4 Afghanistan Asia      1967   34.0 11537966    836.
## 5 Afghanistan Asia      1972   36.1 13079460    740.
## 6 Afghanistan Asia      1977   38.4 14880372    786.
## 7 Afghanistan Asia      1982   39.9 12881816    978.
## 8 Afghanistan Asia      1987   40.8 13867957    852.
## 9 Afghanistan Asia      1992   41.7 16317921    649.
## 10 Afghanistan Asia      1997   41.8 22227415    635.
## # ... with 1,694 more rows
```

Tidy dataset

- Hay muchas formas de estructurar un conjunto de datos. El enfoque tidy sugiere que cada variable sea una columna y cada observación sea una fila, por lo que cada valor tiene su propia celda:

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	128042583

variables

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	128042583

observations

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	128042583

values

Wickham & Grolemund (2018)

Nombres de variables

Muchas veces los usamos datos que no están documentados de manera uniforme o apropiada, por ejemplo, con nombres dispares y propensos a errores en las columnas.

Janitor es un paquete orientado al estilo Tidyverse (aunque no pertenece) que facilita algunas funciones para limpiar y explorar datos.

```
##      COLORES NombresCompletos edad_NUMERICA
## 1 Verde      María S.          32
## 2 Rojo       Juan F.          23
## 3 Azul       Pedro A.         24

##      colores nombres_completos edad_numerica
## 1 Verde      María S.          32
## 2 Rojo       Juan F.          23
## 3 Azul       Pedro A.         24
```

Explorar datos

Resumen de un dataframe

```
dim(data_gapminder) # Número de filas y columnas
```

```
## [1] 1704    6
```

```
names(data_gapminder) # Nombre de variables
```

```
## [1] "country" "continent" "year" "lifeExp" "pop" "gdpPercap"
```

```
head(data_gapminder, 3) # Imprime primeras filas (3 en este caso)
```

```
## # A tibble: 3 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>   <int>    <dbl>
## 1 Afghanistan Asia      1952    28.8  8425333    779.
## 2 Afghanistan Asia      1957    30.3  9240934    821.
## 3 Afghanistan Asia      1962    32.0 10267083    853.
```


Resumen de un dataframe

```
# Estructura del dataframe  
str(data_gapminder)
```

```
## tibble [1,704 x 6] (S3: tbl_df/tbl/data.frame)  
## $ country   : Factor w/ 142 levels "Afghanistan",...: 1 1 1 1 1 1 1 1 1 1 ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",...: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year      : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp   : num [1:1704] 28.8 30.3 32 34 36.1 ...  
## $ pop       : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881...  
## $ gdpPercap: num [1:1704] 779 821 853 836 740 ...
```

Resumen de un dataframe

```
# Pequeño resumen de las variables:  
summary(data_gapminder)
```

```
##           country          continent      year      lifeExp  
## Afghanistan: 12 Africa :624 Min. :1952 Min. :23.60  
## Albania : 12 Americas:300 1st Qu.:1966 1st Qu.:48.20  
## Algeria : 12 Asia :396 Median :1980 Median :60.71  
## Angola : 12 Europe :360 Mean :1980 Mean :59.47  
## Argentina : 12 Oceania : 24 3rd Qu.:1993 3rd Qu.:70.85  
## Australia : 12 Max. :2007 Max. :82.60  
## (Other) :1632  
##           pop          gdpPercap  
## Min. :6.001e+04 Min. : 241.2  
## 1st Qu.:2.794e+06 1st Qu.: 1202.1  
## Median :7.024e+06 Median : 3531.8  
## Mean :2.960e+07 Mean : 7215.3  
## 3rd Qu.:1.959e+07 3rd Qu.: 9325.5  
## Max. :1.319e+09 Max. :113523.1  
##
```

Resumen de un dataframe

Una de las funciones más útiles para resumir un dataframe es `glimpse()` del paquete `dplyr` o `tidyverse`. Es particularmente útil debido a que permite un vistazo al nombre, tipo y primeros valores de **todos** las variables de un dataframe.

```
# Resumen más completo:  
glimpse(gapminder)
```

```
## Rows: 1,704  
## Columns: 6  
## $ country    <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan"  
## $ continent  <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asi  
## $ year       <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 199  
## $ lifeExp    <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 4  
## $ pop        <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372,  
## $ gdpPercap  <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.113
```

Tablas

En R Base la función para obtener frecuencias es `table()` junto con `prop.table()` y `addmargins()`

```
# Para obtener una tabla de frecuencias de una variable usamos la función
# table() de R Base
tabla_1 <- table(data_gapminder$continent) # Frecuencia simple
tabla_1
```

```
##
## Africa Americas Asia Europe Oceania
## 624 300 396 360 24
```

```
prop.table(tabla_1) # Proporciones
```

```
##
## Africa Americas Asia Europe Oceania
## 0.36619718 0.17605634 0.23239437 0.21126761 0.01408451
```

```
addmargins(tabla_1) # Totales
```

```
##
## Africa Americas Asia Europe Oceania Sum
## 624 300 396 360 24 1704
```

```
addmargins(prop.table(tabla_1)) # Proporciones y totales
```

```
##
## Africa Americas Asia Europe Oceania Sum
## 0.36619718 0.17605634 0.23239437 0.21126761 0.01408451 1.00000000
```

Tablas

Para obtener tablas que crucen dos variables podemos nuevamente usar `table()` especificando dos variables.

```
tabla_2 <- table(data_gapminder$continent, data_gapminder$mercosur)
tabla_2
```

```
##
##           0      1
## Africa    624    0
## Americas  252   48
## Asia      396    0
## Europe    360    0
## Oceania    24    0
```

```
prop.table(tabla_2)
```

```
##
##           0      1
## Africa    0.36619718 0.00000000
## Americas  0.14788732 0.02816901
## Asia      0.23239437 0.00000000
## Europe    0.21126761 0.00000000
## Oceania    0.01408451 0.00000000
```

Tablas

```
# Totales por columna o fila
```

```
tabla_2 <- table(data_gapminder$continent, data_gapminder$mercosur)
```

```
addmargins(tabla_2, 1) # Total por columna
```

```
##  
##           0      1  
## Africa    624    0  
## Americas  252   48  
## Asia      396    0  
## Europe    360    0  
## Oceania   24     0  
## Sum      1656   48
```

```
addmargins(tabla_2, 2) # Total por fila
```

```
##  
##           0      1 Sum  
## Africa    624    0 624  
## Americas  252   48 300  
## Asia      396    0 396  
## Europe    360    0 360  
## Oceania   24     0  24
```

Tablas

```
# Editar nombres de columnas
```

```
tabla_2 <- table(data_gapminder$continent, data_gapminder$mercosur)
tabla_2
```

```
##
##           0    1
## Africa    624   0
## Americas  252  48
## Asia      396   0
## Europe    360   0
## Oceania    24   0
```

```
colnames(tabla_2) <- c("No mercosur", "Mercosur")
tabla_2
```

```
##
##           No mercosur Mercosur
## Africa             624         0
## Americas           252        48
## Asia               396         0
## Europe             360         0
## Oceania             24         0
```