OPSO79-1-UCSH2021

Corrección prueba 2, Estadística descriptiva II y transformación avazanzada de data frames

29/10/2021

Transformación avanzada de datos

Un poco más de agrupación, pivotear y combinar data frames

Introducción

La sesión subsiguiente veremos en detalle como elaborar gráficos elegantes en R.

Antes es necesario revisar algunos últimos aspectos sobre tranformación de datos.

La clave para elaborar buenos gráficos en R es tener una data frame coherente con el gráfico que queremos

Por ejemplo,

- si queremos graficar N de hogares por región, no nos servirá una base de datos de personas.
- si queremos graficar mediante barras el porcentaje de personas que reciben mas y menos del sueldo mínimo, la variable numérica salario debe ser categorizada
- Si queremos graficar 2 variables, distinguiendo la relación por una tercera, necesitamos tener una base en formato *longer* (hacia abajo), no *wider* (hacia el lado)

Introducción

A continuación veremos herramientas que nos permitirán lidiar con estos y otros problemas:

- Funciones de agrupación (`group_by()`, `summarise()`).
- Funciones para pivotear la data (`pivot_longer()`, `pivot_wider()`).
- Funciones para combinar data (`merge()`, `rbind()` y `cbind()`)

Aplicaremos estas funciones a los datos del paquete Gapminder, a datos del Banco Mundial (de donde venían los de Afganistán), entre otros.

Agrupación de datos

profundización función group_by()

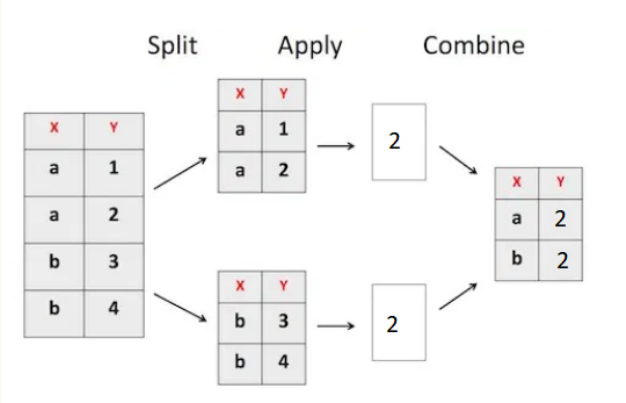
group_by() y summarise()

En conjunto nos permiten resumir información para cada grupo de una variable

Podemos obtener edad promedio por sexo, número de personas en cada región, ingresos por hogar, etcétera.

Estrategia split-apply-combine.

Esta estrategia sucede tras bambalinas (no la vemos). Solo observamos el resultados.



group_by() y summarise()

```
data <- readRDS("data/Latinobarometro_2020_Esp_Rds_v1_0.rds")
```

Conteo de frecuencias

```
data %>% group_by(sexo) %>% summarise(n=n())
```

```
## # A tibble: 2 x 2
##   sexo           n
##   <hvn_lbll> <int>
## 1 1           9667
## 2 2          10537
```

Obtención de estadísitcos para cada grupo

```
data %>% group_by(sexo) %>% summarise(edad=mean(edad))
```

```
## # A tibble: 2 x 2
##   sexo        edad
##   <hvn_lbll> <dbl>
## 1 1           41.6
```

group_by() y mutate()

Con `summarise` "perdemos" la data original. Esta es resumida a una más pequeña.

```
data2 <- data %>% group_by(sexo) %>% summarise(edad=mean(edad))
dim(data2)
```

```
## [1] 2 2
```

```
dim(data)
```

```
## [1] 20204   408
```

Pero en ocasiones queremos una medida de resumen sin perder la data, para poder generar nuevos cálculos.

La alternativa es agrupar sin resumir, sino que mutando la data.

```
data2 <- data %>% group_by(sexo) %>% mutate(edad_promedio=mean(edad))
dim(data2)
```

```
## [1] 20204   409
```

group_by() y mutate()

Veamos un pedazo de la nueva data

```
data2 %>% select(idenpa,sexo,edad,edad_promedio) %>% head()
```

```
## # A tibble: 6 x 4
## # Groups:   sexo [2]
##   idenpa     sexo       edad       edad_promedio
##   <hvn_lbll> <hvn_lbll> <hvn_lbll>         <dbl>
## 1 32         2          63                  40.4
## 2 32         1          24                  41.6
## 3 32         1          20                  41.6
## 4 32         2          54                  40.4
## 5 32         1          38                  41.6
## 6 32         2          62                  40.4
```

Edad promedio aparece en cada observación.

Es el promedio de la edad del grupo (sexo) al que pertenece la observación.

En este caso solo hay valores 41,6 (para los hombres) y 40,4 (para las mujeres)

group_by() y mutate()

¿Cuál es la utilidad?

Sirve para el procesamiento de datos más que para el análisis.

Por ejemplo, identificar casos extraños dentro de un conjunto para luego editarlos.

Países que pertenecen a continentes pobres pero que son **MUY** ricos:

```
library(gapminder)

paises_1972 <- gapminder %>%
  filter(year==1972 ) %>%
  group_by(continent) %>%
  mutate(gdpPercap_continente=quantile(gdpPercap,0.90)) %>%
  ungroup()
```

group_by() y mutate()

```
paises_1972  %>%
  filter(continent %in% c("Africa","Americas") &
         gdpPercap>gdpPercap_continente) %>%
  arrange(-gdpPercap) %>% select(-year,-continent)
```

country	lifeExp	pop	gdpPercap	gdpPercap_continente
United States	71.340	209896000	21806.036	10080.371
Libya	52.773	2183877	21011.497	4139.005
Canada	72.880	22284500	18970.571	10080.371
Gabon	48.690	537977	11401.948	4139.005
Venezuela	65.712	11515649	10505.260	10080.371
South Africa	53.696	23935810	7765.963	4139.005
Angola	37.928	5894858	5473.288	4139.005
Reunion	64.274	461633	5047.659	4139.005
Algeria	54.518	14760787	4182.664	4139.005

Pivotear los datos

`funciones pivot_wider() y pivot_longer()`

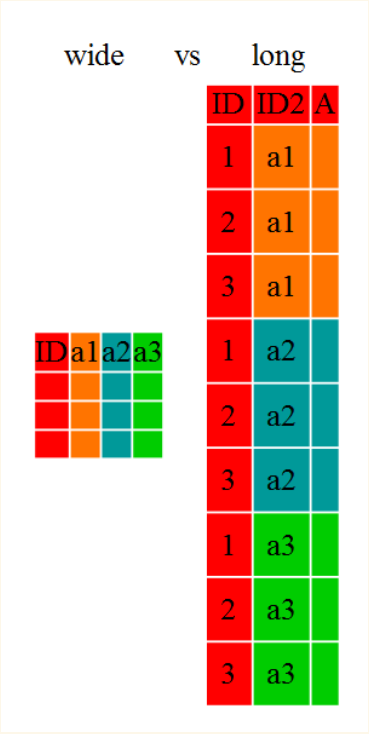
Pivotear los datos

Alargamiento o ensanchamiento de una data frame.

Alargamiento: incremento en el número de filas y decrecimiento del número de columnas

Ensanchamiento: incremento en el número de columnas y decrecimiento del número de filas

Para esto utilizaremos las funciones `pivot_wider()` y `pivot_longer()` del paquete `tidyr`



Función pivot_wider()

Esta función se utiliza para ordenar un dataframe de forma tal de mostrar categorías de una variable como columnas de un dataframe.

Incrementa el número de las columnas y disminuye el número de las filas.

Es útil para la presentación de cuadros de resumen con doble entrada.

sexo	posicion_politica	n
1	centro	3740
1	der	2161
1	izq	2245
1	ninguna	979
2	centro	3582
2	der	2187
2	izq	2327
2	ninguna	1415

Función pivot_wider()

Ahora vemos las categorías de sexo hacia la derecha

posicion_politica	1	2
centro	3740	3582
der	2161	2187
izq	2245	2327
ninguna	979	1415

Pasamos de un formato largo a uno ancho

```
library(tidyr)
```

```
data %>%
  filter(!is.na(posicion_politica)) %>%
  group_by(sexo,posicion_politica) %>%
  summarise(n=n()) %>%
  pivot_wider(names_from = sexo,
              values_from = n)
```

Función pivot_wider()

Básicamente dos argumentos:

- *names_from:* categorías que se quiere convertir en columnas
- *values_from:* columna desde la cual extraer los valores

Además, podemos usar el argumento `names_prefix` cuando tenemos números

```
data %>% filter(!is.na(posicion_politica)) %>%
  group_by(sexo,posicion_politica) %>%
  summarise(n=n()) %>%
  pivot_wider(names_from = sexo,
              values_from = n,
              names_prefix = "sexo_")
```

```
## # A tibble: 4 x 3
##   posicion_politica sexo_1 sexo_2
##   <chr>              <int>  <int>
## 1 centro              3740   3582
## 2 der                 2161   2187
## 3 izq                 2245   2327
## 4 ninguna              979   1415
```

Función pivot_longer()

Esta función se puede considerar como la opuesta a pivot_wider().

Esta función incrementa el número de filas y disminuye el número de columnas.

Los dataframes obtenidos por esta función son más fáciles de manipular, pero no de visualizar

```
df1 <- data.frame(region = c(1, 2),
                  hombres = c(100, 200),
                  mujeres = c(50, 300))
df1
```

```
##   region hombres mujeres
## 1      1     100      50
## 2      2     200     300
```

Función pivot_longer()

```
df1 %>%
  pivot_longer(cols = c(hombres,mujeres) )
```

```
## # A tibble: 4 x 3
##   region name    value
##    <dbl> <chr>   <dbl>
## 1      1 hombres   100
## 2      1 mujeres    50
## 3      2 hombres   200
## 4      2 mujeres   300
```

El argumento principal es cols:

- *cols:* columnas a las que se le aplicará la operación (que se convertirán en categorías de una nueva variable)

Función pivot_longer()

Además, se pueden especificar los nombres de las columnas "name" y "value"

- *names_to:* indica el nombre de la variable que será creada para "guardar" los nombres de las categorías.
- *values_to:* indica el nombre de la variable que será creada para "guardar" los valores asociados a las categorías.

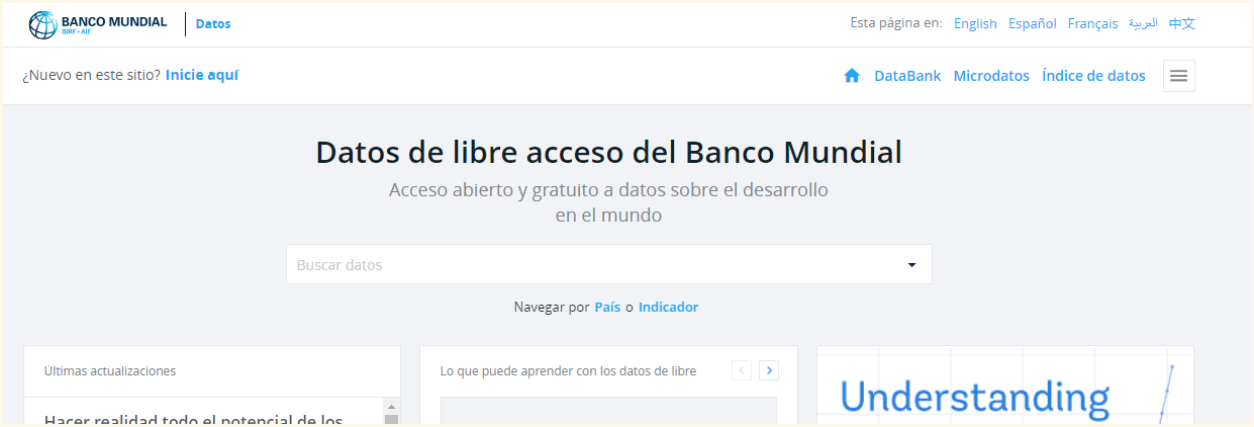
```
df1 %>%
  pivot_longer(cols = c(hombres,mujeres) ,
               names_to = "sexo", values_to = "total_sexo")
```

```
## # A tibble: 4 x 3
##   region sexo    total_sexo
##    <dbl> <chr>        <dbl>
## 1      1 hombres        100
## 2      1 mujeres         50
## 3      2 hombres        200
## 4      2 mujeres        300
```

Pivotear los datos

Relevante para visualizar (ggplot2) y para trabajar datos importados

Por ejemplo, descarguemos los datos de Afganistán que usamos clases atrás. Esta vez sin trampa.



Pivotear los datos

¿Cómo vienen los datos?

```
afganistan <- readxl::read_excel("data/API_AFG_DS2_es_excel_v2_3162018.x
                                 skip = 3)
```

Country Code	2005	2012
AFG	NA	NA
AFG	3.910455	0.5579121
AFG	210.115520	771.2416744
AFG	NA	1.9269141
AFG	NA	0.0000000

¡Las variables vienen como filas!

Pivotear los datos

La data no es un dato ordenado (tidy data)

¿Como graficamos el PIB de Afganistan si no es una variable? Solo podemos tabular años, lo que no tiene sentido:

```
table(afganistan$`1962`)
```

Pivotear los datos

La solución es pivotear los datos. Hacer que las filas pasen a ser variables.

```
# Alargar la data
afganistan <- afganistan %>% pivot_longer(5:ncol(afganistan)) %>%
  select(-`Country Name`,`Country Code`,`Indicator Code`)
```

```
# Quitar filas repetidas para evitar errores
afganistan <- afganistan %>%
  distinct(`Indicator Name`,value,name)
```

```
# Ensanchar la data
afganistan <- afganistan %>%
  tidyr::pivot_wider(names_from = `Indicator Name`,
                                  values_from = value,
                                  values_fn = {sum})
```

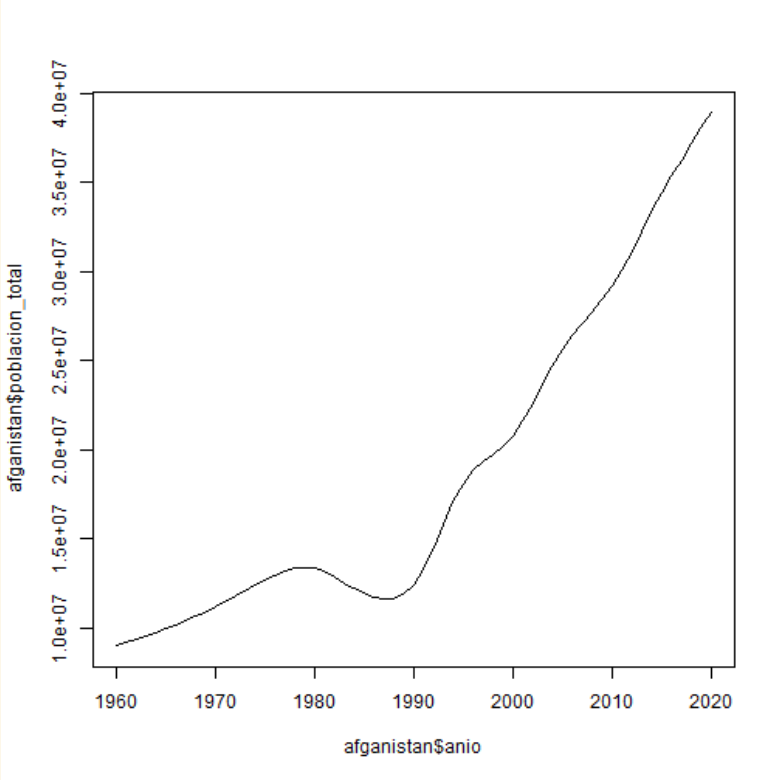
```
# Limpiar los nombres
afganistan <- afganistan %>%
  janitor::clean_names() %>% rename(anio=name)
```

Pivotear los datos

anio	ingreso_nacional_bruto_ing_us	poblacion_total
2012	20033093818	31161378
2013	20632806188	32269592
2014	20482514566	33370804
2015	20087077459	34413603
2016	18197299091	35383028
2017	19118263186	36296111
2018	18544615040	37171922
2019	19598008726	38041757
2020	19996141020	38928341

Pivotear los datos

```
plot(afganistan$anio,afganistan$poblacion_total,type = "l")
```



Combinación de data frames

`funciones cbind(), rbind() y merge()`

Combinación de data frames

Descarguemos la siguiente base de datos con datos de algunos países de América Latina desde 1960 a 2020: DATA BANCO MUNDIAL

El archivo excel tiene 5 hojas, una para cada país (Argentina, Bolivia, Chile, Haití y México)

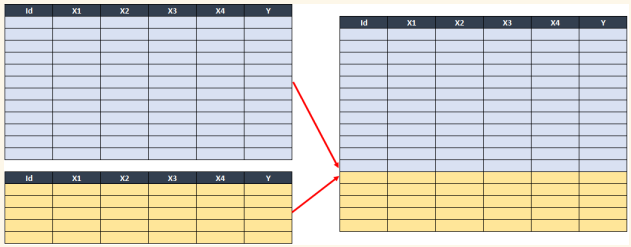
```
library(readxl)
arg <- read_excel("data/datos_bm/paises_banco_mundial.xlsx",sheet = 1)
bol <- read_excel("data/datos_bm/paises_banco_mundial.xlsx",sheet = 2)
chl <- read_excel("data/datos_bm/paises_banco_mundial.xlsx",sheet = 3)
hti <- read_excel("data/datos_bm/paises_banco_mundial.xlsx",sheet = 4)
mex <- read_excel("data/datos_bm/paises_banco_mundial.xlsx",sheet = 5)
```

```
head(arg,3)
```

```
## # A tibble: 3 x 5
##   year       pop lifeExp gdpPercap country
##   <chr>    <dbl>   <dbl>     <dbl> <chr>
## 1 1960  20481781    65.1       NA  Argentina
## 2 1961  20817270    65.2       NA  Argentina
## 3 1962  21153042    65.3     1156. Argentina
```

rbind()

Cuando tenemos las mismas variables, lo más sencillo es combinar pegando los datos uno bajo el otro.




```
data <- rbind(arg,chl,bol,hti,mex)
dim(data)
```

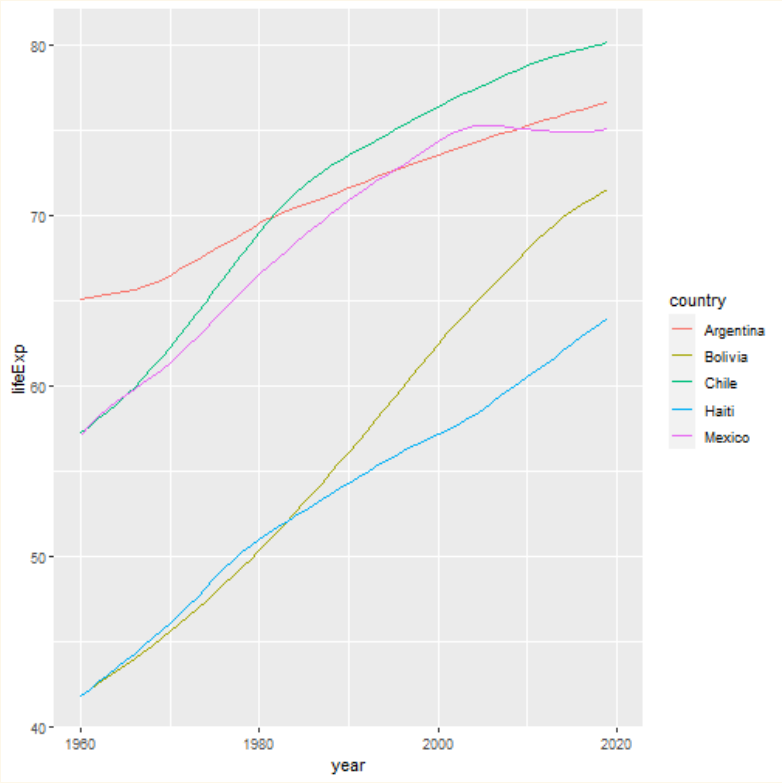
```
## [1] 305   5
```

```
table(data$country)
```

```
##
## Argentina   Bolivia     Chile     Haiti    Mexico
##        61        61        61        61        61
```

rbind()

Con más de un país en la data podemos distinguir distintas series temporales



cbind()

Tenemos los datos para el 2018 de la data recién cargada

```
data %>% filter(year==2018 & country!="Haiti") %>% arrange(country)
```

year	pop	lifeExp	gdpPercap	country
2018	44494502	76.520	11633.498	Argentina
2018	11353140	71.239	3548.591	Bolivia
2018	18729166	80.042	15888.144	Chile
2018	126190782	74.992	9686.514	Mexico

Y de latinobarómetro

idenpa	evangelicos
32	3.750000
68	15.666667
152	9.583333
484	2.500000

cbind()

```
cbind(bm2018,evangelicos)
```

```
##   year       pop lifeExp gdpPercap   country idenpa evangelicos
## 1 2018  44494502  76.520 11633.498 Argentina     32    3.750000
## 2 2018  11353140  71.239  3548.591   Bolivia     68   15.666667
## 3 2018  18729166  80.042 15888.144     Chile    152    9.583333
## 4 2018 126190782  74.992  9686.514    Mexico    484    2.500000
```

¿Cuál es el gran problema?

Los países estaban desordenados.

Ordenar países

```
evangelicos<- evangelicos %>%
                mutate(idenpa=case_when(idenpa==32 ~ "Argentina",
                                        idenpa==68 ~ "Bolivia",
                                        idenpa==152 ~ "Chile",
                                        idenpa==484 ~ "Mexico"))
```

cbind()

Ahora sí combinar:

```
cbind(bm2018,evangelicos) %>% select(-idenpa)
```

```
##   year       pop lifeExp gdpPercap   country evangelicos
## 1 2018  44494502  76.520 11633.498 Argentina    3.750000
## 2 2018  11353140  71.239  3548.591   Bolivia   15.666667
## 3 2018  18729166  80.042 15888.144     Chile    9.583333
## 4 2018 126190782  74.992  9686.514    Mexico    2.500000
```

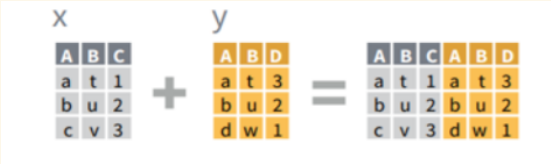
`cbind()` y `bind()` tienen limitaciones, solo combinan cuando existe el mismo número de filas y las variables se llaman igual.

`bind_rows()` y `col_rows()`, las versiones `dplyr` de las funciones vistas, son un poco más flexibles.

De todas formas `merge()` nos permitirá hacer más cosas, siendo fundamental el uso de variables llaves

merge()

La lógica de bind_rows()



La lógica de merge()

Para especificar el tipo de merge()

```
all.x=TRUE

all.y=TRUE

all=FALSE

all=TRUE
```

merge()

La función tiene 4 argumentos fundamentales

```
merge(x, y, by="key", all.x=FALSE)
```

- `x` es la data 1
- `y` es la data 2
- "key" es la variable llave usada para combinar
- con `all` indicamos si queremos mantener los valores de x, de y, de todas o de ninguna

merge()

Crear dos bases de datos

```
songs <- tibble(song = c("Come Together", "Dream On",
                         "Hello,Goodbye", "It's Not Unusual"),
                album  = c("Abbey Road", "Aerosmith",
                           "Magical Mystery Tour", "Along Came Jones"),
                first = c("John", "Steven", "Paul", "Tom"),
                last = c("Lennon", "Tyler", "McCartney", "Jones"))

albums <- tibble( album = c("A Hard Day's Night",
                            "Magical Mystery Tour", "Beggar's Banquet",
                           "Abbey Road", "Led Zeppelin IV",
                           "The Dark Side of the Moon",
                           "Aerosmith", "Rumours", "Hotel California"),
                  band = c("The Beatles", "The Beatles",
                           "The Rolling Stones",
                           "The Beatles", "Led Zeppelin",
                           "Pink Floyd", "Aerosmith",
                           "Fleetwood Mac", "Eagles"),
                year = c(1964,1967,1968,1969,1971,1973,1973,1977,1982))
```

merge()

¿Que variable tienen en común songs y albums?

album, por lo que será la llave.

```
merge(albums,songs, by="album", all = TRUE)
```

album	band	year	song	first	last
A Hard Day's Night	The Beatles	1964	NA	NA	NA
Abbey Road	The Beatles	1969	Come Together	John	Lennon
Aerosmith	Aerosmith	1973	Dream On	Steven	Tyler
Along Came Jones	NA	NA	It's Not Unusual	Tom	Jones
Beggar's Banquet	The Rolling Stones	1968	NA	NA	NA
Hotel California	Eagles	1982	NA	NA	NA

merge()

Probemos con `all=FALSE`

```
merge(albums,songs, by="album", all = FALSE)
```

album	band	year	song	first	last
Abbey Road	The Beatles	1969	Come Together	John	Lennon
Aerosmith	Aerosmith	1973	Dream On	Steven	Tyler
Magical Mystery Tour	The Beatles	1967	Hello,Goodbye	Paul	McCartney

¿Que sucedió?

Solo se mantienen las observaciones que simultáneamente están en X y en Y

merge()

Quedarse con todos los valores de x (`all.x=TRUE`)

```
merge(albums,songs, by="album", all.x = TRUE)
```

album	band	year	song	first	last
A Hard Day's Night	The Beatles	1964	NA	NA	NA
Abbey Road	The Beatles	1969	Come Together	John	Lennon
Aerosmith	Aerosmith	1973	Dream On	Steven	Tyler
Beggar's Banquet	The Rolling Stones	1968	NA	NA	NA
Hotel California	Eagles	1982	NA	NA	NA
Led Zeppelin IV	Led Zeppelin	1971	NA	NA	NA
Magical Mystery Tour	The Beatles	1967	Hello,Goodbye	Paul	McCartney
Rumours	Fleetwood Mac	1977	NA	NA	NA

merge()

Quedarse con todos los valores de y (`all.y=TRUE`)

```
merge(albums,songs, by="album", all.y = TRUE)
```

album	band	year	song	first	last
Abbey Road	The Beatles	1969	Come Together	John	Lennon
Aerosmith	Aerosmith	1973	Dream On	Steven	Tyler
Along Came Jones	NA	NA	It's Not Unusual	Tom	Jones
Magical Mystery Tour	The Beatles	1967	Hello,Goodbye	Paul	McCartney

Ejercicio breve

Agregar afganistán a la base de países de AL (CHL, ARG, BOL, MEX, HTI)

Luego, pegar la variable continente de gapminder

```
head(data)
```

```
## # A tibble: 6 x 5
##    year      pop lifeExp gdpPercap country
##   <dbl>    <dbl>   <dbl>     <dbl> <chr>
## 1  1960 20481781    65.1       NA  Argentina
## 2  1961 20817270    65.2       NA  Argentina
## 3  1962 21153042    65.3     1156. Argentina
## 4  1963 21488916    65.3      850. Argentina
## 5  1964 21824427    65.4     1173. Argentina
## 6  1965 22159644    65.5     1279. Argentina
```

Ejercicio breve

Adecuar Afganistán al formato de la data

```
afganistan <- afganistan %>% select(anio,
                      poblacion_total,
                      esperanza_de_vida_al_nacer_total_anos,
                      pib_per_capita_us_a_precios_actuales) %>%
  rename(year=anio,
         pop=poblacion_total,
         gdpPercap=pib_per_capita_us_a_precios_actuales,
         lifeExp=esperanza_de_vida_al_nacer_total_anos) %>%
  mutate(country="Afghanistan")
```

Combinar las dos datas

```
data <- rbind(afganistan,data)
table(data$country)
```

```
##
## Afghanistan   Argentina     Bolivia       Chile       Haiti      Mexico
##          61          61          61          61          61          61
```

Ejercicio breve

Extrae continente de gapminder

```
continente <- gapminder %>%
  filter(year==1952) %>%
  select(country,continent)

table(continente$continent)
```

```
##
##   Africa Americas     Asia   Europe  Oceania
##       52       25       33       30        2
```

Pegar el continente

```
dataconcontinente <- merge(data,continente,by=c("country"),all.x = TRUE)
```

Ejercicio breve

```
head(dataconcontinente)
```

```
##       country year     pop lifeExp gdpPercap continent
## 1 Afghanistan 1960 8996967  32.446  59.77323      Asia
## 2 Afghanistan 1961 9169406  32.962  59.86090      Asia
## 3 Afghanistan 1962 9351442  33.471  58.45801      Asia
## 4 Afghanistan 1963 9543200  33.971  78.70643      Asia
## 5 Afghanistan 1964 9744772  34.463  82.09531      Asia
## 6 Afghanistan 1965 9956318  34.948 101.10833      Asia
```

```
tail(dataconcontinente)
```

```
##     country year       pop lifeExp gdpPercap continent
## 361  Mexico 2015 121858251  74.904  9616.646  Americas
## 362  Mexico 2016 123333379  74.917  8744.516  Americas
## 363  Mexico 2017 124777326  74.947  9287.850  Americas
## 364  Mexico 2018 126190782  74.992  9686.514  Americas
## 365  Mexico 2019 127575529  75.054  9946.034  Americas
## 366  Mexico 2020 128932753      NA  8346.702  Americas
```

Ejercicio breve

¿Que pasaría con `all.y=TRUE`?

```
merge(data,continente,by=c("country"),all.y = TRUE)
```

¿Con cuantas observaciones nos quedaríamos? (desafío para la casa)

merge()

Para cerrar

¿Podemos usar más de una llave?

Sí, con `by=c("var1","var2")`

¿Se puede combinar más de una data frame al mismo tiempo?

Sí, teóricamente infinitas hasta que colapse la memoria del pc:

```
Reduce(function(x, y) merge(x, y), list(x, y, z, etc))
```

Deben escribirse dentro de `list()`

Con comandos más avanzados se pueden leer y combinar todas las bases de datos de una carpeta del computador o del ambiente.

Recursos web utilizados

Xaringan: Presentation Ninja, de Yihui Xie. Para generar esta presentación.

Ilustraciones de Allison Horst

Para reforzar y seguir aprendiendo

Funciones merge() en R

Video "el juego de las estadísticas" (utiliza gapminder)