Manipulación de datos, más allá de lo básico

Curso de introducción a 

Pablo Cabrera-Álvarez

 |  @pablocalv

Julio 2019

Quiz

Vamos a utilizar el conjunto de datos `gss_cat`. Para poder reescribirli voy a guardarlo como `gss`:

```
gss <- gss_cat
glimpse(gss)
```

```
## Observations: 21,483
## Variables: 9
## $ year    <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, ...
## $ marital <fct> Never married, Divorced, Widowed, Never married, Divor...
## $ age     <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, 51...
## $ race    <fct> White, White, White, White, White, White, White, White...
## $ rincome <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not appl...
## $ partyid <fct> "Ind,near rep", "Not str republican", "Independent", "...
## $ relig   <fct> Protestant, Protestant, Protestant, Orthodox-christian...
## $ denom   <fct> Southern baptist, Baptist-dk which, No denomination, N...
## $ tvhours <int> 12, NA, 2, 4, 1, NA, 3, NA, 0, 3, 2, NA, 1, NA, 1, 7, ...
```

Quiz

- **Ordenar** el conjunto de datos por `age` ascendente y `year` descendente:

```
gss <- arrange(gss, age, desc(year))
glimpse(gss)
```

```
## Observations: 21,483
## Variables: 9
## $ year    <int> 2014, 2014, 2014, 2014, 2014, 2014, 2012, 2012, 2012, ...
## $ marital <fct> Never married, Never married, Never married, Never mar...
## $ age     <int> 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18...
## $ race    <fct> White, Black, White, Black, White, White, Other, Other...
## $ rincome <fct> Not applicable, Not applicable, Not applicable, Not ap...
## $ partyid <fct> "Other party", "Ind,near dem", "Ind,near rep", "Not st...
## $ relig   <fct> None, Protestant, Protestant, None, None, Protestant, ...
## $ denom   <fct> Not applicable, Baptist-dk which, Baptist-dk which, No...
## $ tvhours <int> 5, 4, 1, 3, 0, 0, 5, 3, 2, NA, NA, 1, 3, 1, 2, 4, 1, 2...
```

Quiz

- Crear una variable que sea `id` con un **identificador** único:

```
gss <- mutate(gss,
              id = 1:nrow(gss))
glimpse(gss)
```

```
## Observations: 21,483
## Variables: 10
## $ year    <int> 2014, 2014, 2014, 2014, 2014, 2014, 2012, 2012, 2012, ...
## $ marital <fct> Never married, Never married, Never married, Never mar...
## $ age     <int> 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18...
## $ race    <fct> White, Black, White, Black, White, White, Other, Other...
## $ rincome <fct> Not applicable, Not applicable, Not applicable, Not ap...
## $ partyid <fct> "Other party", "Ind,near dem", "Ind,near rep", "Not st...
## $ relig   <fct> None, Protestant, Protestant, None, None, Protestant, ...
## $ denom   <fct> Not applicable, Baptist-dk which, Baptist-dk which, No...
## $ tvhours <int> 5, 4, 1, 3, 0, 0, 5, 3, 2, NA, NA, 1, 3, 1, 2, 4, 1, 2...
## $ id      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...
```

Quiz

- **Reordenar** las variables para que `id` sea la primera del data frame:

```
gss <- select(gss,
              id, everything())
glimpse(gss)
```

```
## Observations: 21,483
## Variables: 10
## $ id      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...
## $ year    <int> 2014, 2014, 2014, 2014, 2014, 2014, 2012, 2012, 2012, ...
## $ marital <fct> Never married, Never married, Never married, Never mar...
## $ age     <int> 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18...
## $ race    <fct> White, Black, White, Black, White, White, Other, Other...
## $ rincome <fct> Not applicable, Not applicable, Not applicable, Not ap...
## $ partyid <fct> "Other party", "Ind,near dem", "Ind,near rep", "Not st...
## $ relig   <fct> None, Protestant, Protestant, None, None, Protestant, ...
## $ denom   <fct> Not applicable, Baptist-dk which, Baptist-dk which, No...
## $ tvhours <int> 5, 4, 1, 3, 0, 0, 5, 3, 2, NA, NA, 1, 3, 1, 2, 4, 1, 2...
```

Quiz

- **Filtrar** lo casos que sean mayores de 25 años:

```
gss <- filter(gss,
              age > 25)
summary(gss$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   26.00   37.00   48.00   50.15   61.00   89.00
```

Quiz

- **Renombrar** la variable `rincome` → `income_bands`:

```
gss <- rename(gss,
              income_bands = rincome)
glimpse(gss)
```

```
## Observations: 19,139
## Variables: 10
## $ id           <int> 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2...
## $ year         <int> 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2...
## $ marital      <fct> Married, Never married, Never married, Never marr...
## $ age          <int> 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 2...
## $ race         <fct> White, White, Black, Other, White, White, White, ...
## $ income_bands <fct> $25000 or more, $25000 or more, $25000 or more, $...
## $ partyid      <fct> "Not str republican", "Ind,near dem", "Not str de...
## $ relig        <fct> Catholic, None, Catholic, Catholic, Catholic, Cat...
## $ denom        <fct> Not applicable, Not applicable, Not applicable, N...
## $ tvhours      <int> NA, 3, 2, 3, NA, NA, 0, 3, NA, NA, NA, NA, 1, NA,...
```

Manipulación de datos, más allá de lo básico

- Un toque de elegancia: el uso de *pipes* %>%
- **Agrupar** datos
- **Resumir** variables
- Combinar **filas**
- Combinar **columnas**
- De archivo largo a ancho: **spread**
- De ancho a largo: **gather**
- Crear **funciones**

Operar con *pipes* %>%

Espíritu `tidyverse`

```
gss <- gss_cat
gss <- arrange(gss, age, desc(year))
gss <- mutate(gss, id = 1:nrow(gss))
gss <- select(gss, id, everything())
gss <- filter(gss, age > 25)
gss <- rename(gss, income_bands = rincome)
```

¿Cómo simplificar el código?

`gss` está repetida 11 veces

Espíritu `tidyverse`

```
gss  <- gss_cat
        arrange(age, desc(year))
        mutate(id = 1:nrow(gss))
        select(id, everything())
        filter(age > 25)
        rename(income_bands = rincome)
```

Espíritu tidyverse

```
gss  <- gss_cat %>%
        arrange(age, desc(year)) %>%
        mutate(id = 1:nrow(gss_cat)) %>%
        select(id, everything()) %>%
        filter(age > 25) %>%
        rename(income_bands = rincome)
glimpse(gss)
```

```
## Observations: 19,139
## Variables: 10
## $ id           <int> 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2...
## $ year         <int> 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2...
## $ marital      <fct> Married, Never married, Never married, Never marr...
## $ age          <int> 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 2...
## $ race         <fct> White, White, Black, Other, White, White, White, ...
## $ income_bands <fct> $25000 or more, $25000 or more, $25000 or more, $...
## $ partyid      <fct> "Not str republican", "Ind,near dem", "Not str de...
## $ relig        <fct> Catholic, None, Catholic, Catholic, Catholic, Cat...
## $ denom        <fct> Not applicable, Not applicable, Not applicable, N...
## $ tvhours      <int> NA, 3, 2, 3, NA, NA, 0, 3, NA, NA, NA, NA, 1, NA,...
```

Operar con %>%

- Las *pipes* o tuberías **transportan un objeto**, generalmente un data frame, a lo largo de una serie de **funciones** que sucesivamente van transformando los datos:

```
gss_cat %>%
  select(race, partyid) %>%
  filter(race == "White")
```

Operar con %>%

- También se pueden incluir **funciones** que generen otros **objetos** (e.g. gráficos o análisis):

```
gss_cat %>%
  select(race, partyid) %>%
  filter(race == "White") %>%
  head()
```

```
## # A tibble: 6 x 2
##   race  partyid
##   <fct> <fct>
## 1 White Ind,near rep
## 2 White Not str republican
## 3 White Independent
## 4 White Ind,near rep
## 5 White Not str democrat
## 6 White Strong democrat
```

Operar con %>%

- En ocasiones es necesario **referirse a los datos dentro de un pipe**, para ello se puede utilizar el punto (`.`).

```
gss_cat %>%
  mutate(id = 1:nrow(.)) %>%
  select(id) %>%
  head()
```

```
## # A tibble: 6 x 1
##      id
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
## 5     5
## 6     6
```

Uso de las *pipes* en el código

- Evitar *pipes* de **más de diez líneas**, en ese caso crear varios objetos
- Utiliza *pipes* siempre que haya **dos o más funciones**, pero no en caso de que corresponda usar una única función:

```
gss_cat %>% select(race, age) # mal

gss_cat %>% #bien
  select(race, age) %>%
  filter(race == "White")
```

- Utiliza el atajo `Ctrl` + `Mayús` + `M` para insertar *pipes* `%>%` en `RStudio`

Quiz

¿Se puede hacer un *pipe* que culmine en la función `table()`? ¿Cuál será el comportamiento?

```
gss_cat %>%
  select(race, marital, partyid) %>%
  table()
```

Quiz

```
## , , partyid = No answer
##
##                marital
## race            No answer Never married Separated Divorced Widowed
##   Other                 0             6         3        3       2
##   Black                 1            14         3        9       2
##   White                 4            15         3       17      12
##   Not applicable        0             0         0        0       0
##                marital
## race            Married
##   Other              11
##   Black               7
##   White              42
##   Not applicable      0
##
## , , partyid = Don't know
##
##                marital
## race            No answer Never married Separated Divorced Widowed
##   Other                 0             0         0        0       0
##   Black                 0             0         0        0       0
```

Agrupar means group 

Agrupar casos

`group_by(.data, ...)`

- La función `group_by()` permite agrupar los casos según una o varias variables:

```
gss_group <- gss_cat %>%
          group_by(race)
glimpse(gss_group)
```

```
## Observations: 21,483
## Variables: 9
## Groups: race [3]
## $ year    <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, ...
## $ marital <fct> Never married, Divorced, Widowed, Never married, Divor...
## $ age     <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, 51...
## $ race    <fct> White, White, White, White, White, White, White, White...
## $ rincome <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not appl...
## $ partyid <fct> "Ind,near rep", "Not str republican", "Independent", "...
## $ relig   <fct> Protestant, Protestant, Protestant, Orthodox-christian...
## $ denom   <fct> Southern baptist, Baptist-dk which, No denomination, N...
## $ tvhours <int> 12, NA, 2, 4, 1, NA, 3, NA, 0, 3, 2, NA, 1, NA, 1, 7, ...
```

Resumir means summarise 

Hacer estadísticos resumen

`summarise(.data, ...)`

- La función `summarise()` es útil para crear estadísticos **resumen**:

```
gss_cat %>%
summarise(mean_age = mean(age, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   mean_age
##      <dbl>
## 1     47.2
```

Hacer estadísticos resumen

- También funciona con data frames agrupados:

```
gss_group %>%
summarise(mean_age = mean(age, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   race  mean_age
##   <fct>    <dbl>
## 1 Other     39.5
## 2 Black     43.9
## 3 White     48.7
```

Funcionamiento habitual de summarise

- El **resumen** de los datos con `summarise()` se suele hacer después de una **agrupación** de los mismos con `group_by()`:

```
gss_cat %>%
  group_by(race) %>%
  summarise(mean_age = mean(age, na.rm = TRUE),
            sd_age = sd(age, na.rm = TRUE),
            first_rincome = first(rincome))
```

```
## # A tibble: 3 x 4
##   race  mean_age sd_age first_rincome
##   <fct>    <dbl>  <dbl> <fct>
## 1 Other     39.5   14.4 $20000 - 24999
## 2 Black     43.9   16.1 $25000 or more
## 3 White     48.7   17.5 $8000 to 9999
```

Funciones de apoyo a summarise

- `mean()`
- `sd()`
- `median()`
- `IQR()`
- `min()`
- `max()`
- `qualtile()`

- `first()`
- `last()`
- `nth()`
- `n()`
- `n_distinct()`
- `any()`
- `all()`

Una función extra: frecuencia más repetida

`which.max(x)`

- Una función auxiliar que no está implementada en *tidyverse* es extraer el nivel con la frecuencia más alta para cada grupo. Para ello es necesario combinar tres funciones de `rbase`: `table()`, `which.max()` y `names()`.
- Lo solucionamos en tres pasos:
    1. Con `table()` se obtienen las **frecuencias**
    2. Con `which.max()` se **selecciona la categoría** con mayor frecuencia
    3. Con `names()` se extrae el **nombre del nivel** del factor

Una función extra: frecuencia más repetida

- Empezamos con la **tabla**:

```
table(gss_cat$partyid)
```

```
##
##          No answer         Don't know        Other party
##                154                  1                393
##  Strong republican Not str republican      Ind,near rep
##               2314               3032              1791
##        Independent       Ind,near dem   Not str democrat
##               4119               2499              3690
##    Strong democrat
##               3490
```

- El siguiente código da **la etiqueta y el nivel de la categoría** con mayor frecuencia:

```
which.max(table(gss_cat$partyid))
```

```
## Independent
##           7
```

Una función extra: frecuencia más repetida

- Se aplica la función `names()` para obtener la **etiqueta de la variable**:

```
names(which.max(table(gss_cat$partyid)))
```

```
## [1] "Independent"
```

Un ejemplo de la aplicación

- Para cada grupo de la variable `race` **obtener el partido más frecuente** con el que se identifican `party_id`:

```
gss_cat %>%
  group_by(race) %>%
  summarise(most_freq_partyid = names(which.max(table(partyid))))
```

```
## # A tibble: 3 x 2
##   race  most_freq_partyid
##   <fct> <chr>
## 1 Other Independent
## 2 Black Strong democrat
## 3 White Independent
```

Crear una variable resumen a nivel individual

- La función `group_by()` también funciona con `mutate()` para **crear una variable resumen** a nivel individual:

```
gss_cat %>%
  select(race, age) %>%
  group_by(race) %>%
  mutate(mean_age = mean(age, na.rm = TRUE)) %>%
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   race [1]
##   race    age mean_age
##   <fct> <int>    <dbl>
## 1 White    26     48.7
## 2 White    48     48.7
## 3 White    67     48.7
## 4 White    39     48.7
## 5 White    25     48.7
## 6 White    25     48.7
```

Combinar filas 

Combinar filas

- Cargamos dos data frames:

```
rows_a <- gss_cat[1:3, c(1, 3:4)]
glimpse(rows_a)
```

```
## Observations: 3
## Variables: 3
## $ year <int> 2000, 2000, 2000
## $ age  <int> 26, 48, 67
## $ race <fct> White, White, White
```

```
rows_b <- gss_cat[4:6, c(1:4)]
glimpse(rows_b)
```

```
## Observations: 3
## Variables: 4
## $ year    <int> 2000, 2000, 2000
## $ marital <fct> Never married, Divorced, Married
## $ age     <int> 39, 25, 25
## $ race    <fct> White, White, White
```

Combinar filas

`bind_rows(..., id)`

- La función `bind_rows()` permite **combinar dos data frames** de filas en un único objeto. La combinación se lelva acabo a partir del nombre de las columnas o variables:

```
bind_rows(rows_a, rows_b)
```

```
## # A tibble: 6 x 4
##    year   age race  marital
##   <int> <int> <fct> <fct>
## 1  2000    26 White <NA>
## 2  2000    48 White <NA>
## 3  2000    67 White <NA>
## 4  2000    39 White Never married
## 5  2000    25 White Divorced
## 6  2000    25 White Married
```

Combinar variables 

Combinar variables

- Partimos de dos data frames:

```
gss <- gss_cat %>%
  mutate(id = 1:nrow(gss_cat)) %>%
  select(id, everything())
cols_a <- gss[1:3, 1:3]
glimpse(cols_a)
```

```
## Observations: 3
## Variables: 3
## $ id      <int> 1, 2, 3
## $ year    <int> 2000, 2000, 2000
## $ marital <fct> Never married, Divorced, Widowed
```

```
cols_b <- gss[c(1:2, 4), c(1, 4:5)]
glimpse(cols_b)
```

```
## Observations: 3
## Variables: 3
## $ id   <int> 1, 2, 4
```

Combinar variables

`bind_cols(...)`

- La función `bind_cols()` realiza una **combinación de dos data frames** a partir de las columnas; la **combinación es posicional** por lo que es necesario que los casos (filas) estén ordenados apra que se realice correctamente:

```
bind_cols(cols_a, cols_b)
```

```
## # A tibble: 3 x 6
##      id  year marital         id1   age race
##   <int> <int> <fct>         <int> <int> <fct>
## 1     1  2000 Never married     1    26 White
## 2     2  2000 Divorced          2    48 White
## 3     3  2000 Widowed          4    39 White
```

Combinar variables, por la izquierda

`left_join(x, y, by)`

- La función `left_join()` sirve para **unir dos data frames** en base a una o más variables clave, tomando como referencia el data frame `x`, situado a la izquierda de la definición:

```
left_join(x = cols_a, y = cols_b, by = "id")
```

```
## # A tibble: 3 x 5
##      id  year marital         age race
##   <int> <int> <fct>         <int> <fct>
## 1     1  2000 Never married    26 White
## 2     2  2000 Divorced         48 White
## 3     3  2000 Widowed          NA <NA>
```

Combinar variables, por la derecha

`right_join(x, y, by)`

- La función `right_join()` sirve para **unir dos data frames** en base a una o más variables clave, tomando como referencia el data frame `y`, situado a la derecha de la definición:

```
right_join(x = cols_a, y = cols_b, by = "id")
```

```
## # A tibble: 3 x 5
##      id  year marital         age race
##   <int> <int> <fct>         <int> <fct>
## 1     1  2000 Never married    26 White
## 2     2  2000 Divorced         48 White
## 3     4    NA <NA>             39 White
```

Combinar variables, todas

`full_join(x, y, by)`

- La función `full_join()` sirve para **unir dos data frames** en base a una o más variables clave:

```
full_join(x = cols_a, y = cols_b, by = "id")
```

```
## # A tibble: 4 x 5
##      id  year marital         age race
##   <int> <int> <fct>         <int> <fct>
## 1     1  2000 Never married    26 White
## 2     2  2000 Divorced         48 White
## 3     3  2000 Widowed          NA <NA>
## 4     4    NA <NA>             39 White
```

Variables sin combinar

`anti_join()`

- En ocasiones es necesario tener un **listado de aquellos casos que no se han combinado** al no estar en ambos data frames. Para ello se puede utilizar la función `anti_join()`:

```
anti_join(x = cols_a, y = cols_b, by = "id")
```

```
## # A tibble: 1 x 3
##      id  year marital
##   <int> <int> <fct>
## 1     3  2000 Widowed
```

Quiz

¿Cuál es la **diferencia** entre las siguientes líneas de código?

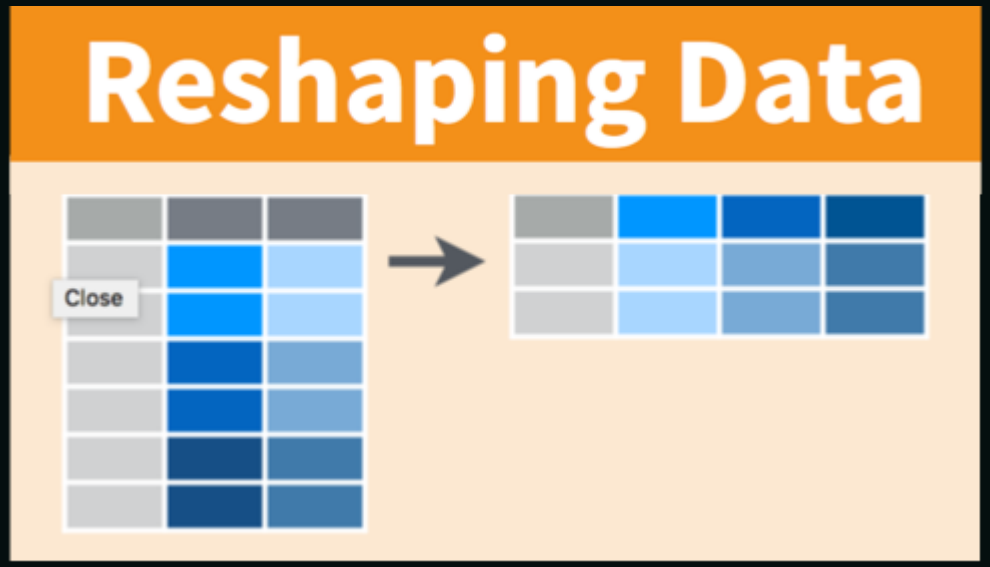
```
left_join(x, y, by = c("region", "id"))
```

```
right_join(y, x, by = c("region", "id"))
```

A formato ancho 

Cambiar la forma de los datos

- En ocasiones los datos deben ser **cambiados de forma** par apoder ser analizados:



Datos en formato largo

- Datos en formato largo:

```
muni
```

```
## # A tibble: 4 x 3
##   muni      item    valor_mill
##   <chr>     <chr>        <dbl>
## 1 Madrid    Poblac.        5.7
## 2 Madrid    Votos          2.4
## 3 Barcelona Poblac.        2.5
## 4 Barcelona Votos          1.6
```

De largo a ancho

`spread(data, key, value)`

- Para pasar los datos de formato **largo a ancho** se usa lafunción `spread()`:

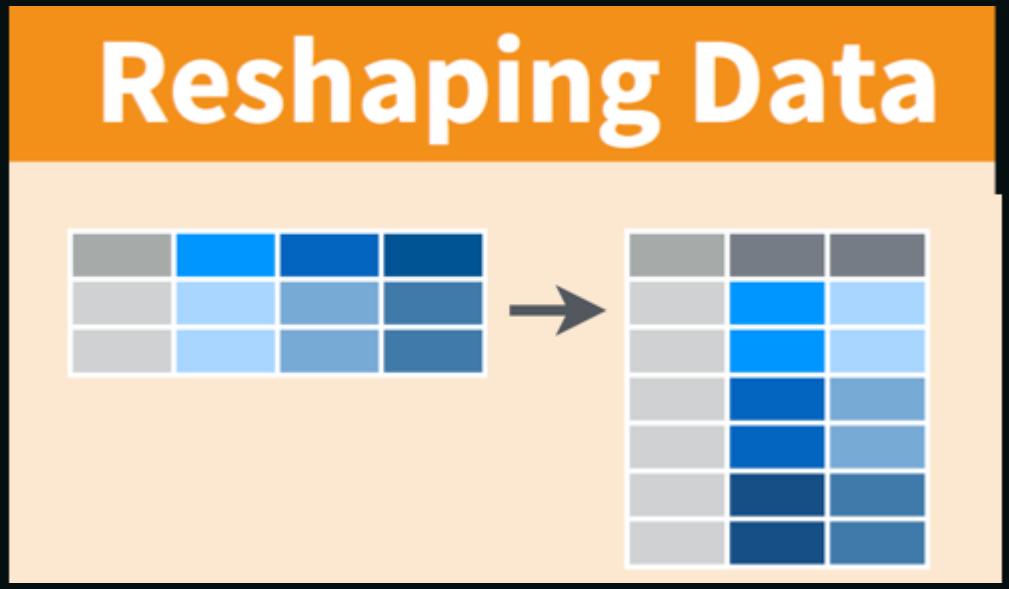
```
spread(data = muni, key = item, value = valor_mill)
```

```
## # A tibble: 2 x 3
##   muni      Poblac. Votos
##   <chr>       <dbl> <dbl>
## 1 Barcelona     2.5   1.6
## 2 Madrid        5.7   2.4
```

A formato largo ⤵

Cambiar la forma de los datos

- Otra manera de formatear los datos es pasarlos **de ancho a largo**:



De ancho a largo

- Datos en formato **ancho**:

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
head(iris, n = 5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

De ancho a largo

`gather(data, key, value, ...)`

- Con la función `gather()` los datos pasan de **formato ancho a largo**. Los argumentos son los mismos que en la función `spread()`, pero los valores esperados son distintos. Se espera unas nuevas versiones de estas funciones próximamente: `pivot_wide()` y `pivot_long()`.

```
gather(data = iris, key = "var", value = "value", Sepal.Length:Petal.Width)
```

```
##         Species          var value
## 1        setosa Sepal.Length   5.1
## 2        setosa Sepal.Length   4.9
## 3        setosa Sepal.Length   4.7
## 4        setosa Sepal.Length   4.6
## 5        setosa Sepal.Length   5.0
## 6        setosa Sepal.Length   5.4
## 7        setosa Sepal.Length   4.6
## 8        setosa Sepal.Length   5.0
## 9        setosa Sepal.Length   4.4
## 10       setosa Sepal.Length   4.9
## 11       setosa Sepal.Length   5.4
```

Intro a creación de funciones 

La estructura de una función

`function(args)`

```
mifuncion <- function(argumento1, argumento2, ...) {
  cuerpo
  resultado
}
```

- El nombre de la función es `mifuncion`
- Para crear una función se utiliza la función `function(){}`
- Una función está compuesta por argumentos: `argumento1`
- En el **cuerpo** de la función están las acciones
- El **resultado** es el output de la función

Argumentos de una función

- Los **argumentos** de una función pueden tener valores predeterminados:

```
mifuncion <- function(argumento1 =  TRUE, argumento2 = 10) {
  cuerpo
  resultado
}
```

Retornar un objeto

`return(x)`

- Para retornar un objeto se utiliza la función `return()`:

```
mifuncion <- function(datos) {
  x <- datos*2
  return(x)
}
```

Ejemplo:

- Crear una función que **multiplique un vector por sí mismo**:

```
mult_vector <- function(vector){

  vector_2 <- vector*vector

  return(vector_2)
}

nums <- c(2, 3, 5)

mult_vector(nums)
```

```
## [1]  4  9 25
```