Programación en R para el análisis de datos

Ordenar y transformar datos

Nicolás Schmidt

mail::nschmidt@cienciassociales.edu.uy
GitHub::@Nicolas-Schmidt

Departamento de Ciencia Política
Facultad de Ciencias Sociales

Ruta

1. Datos ordenados y desordenados: Problemas

2. Datos ordenados y desordenados: Soluciones

3. Datos relacionados

4. data.frame y tibble

5. Exploración de datos con dplyr

6. Pipe

Datos ordenados y desordenados: Problemas

¿Hay algún problema?

```
datos <- rio::import("elecciones_nacionales_19_montevideo.xlsx")
```

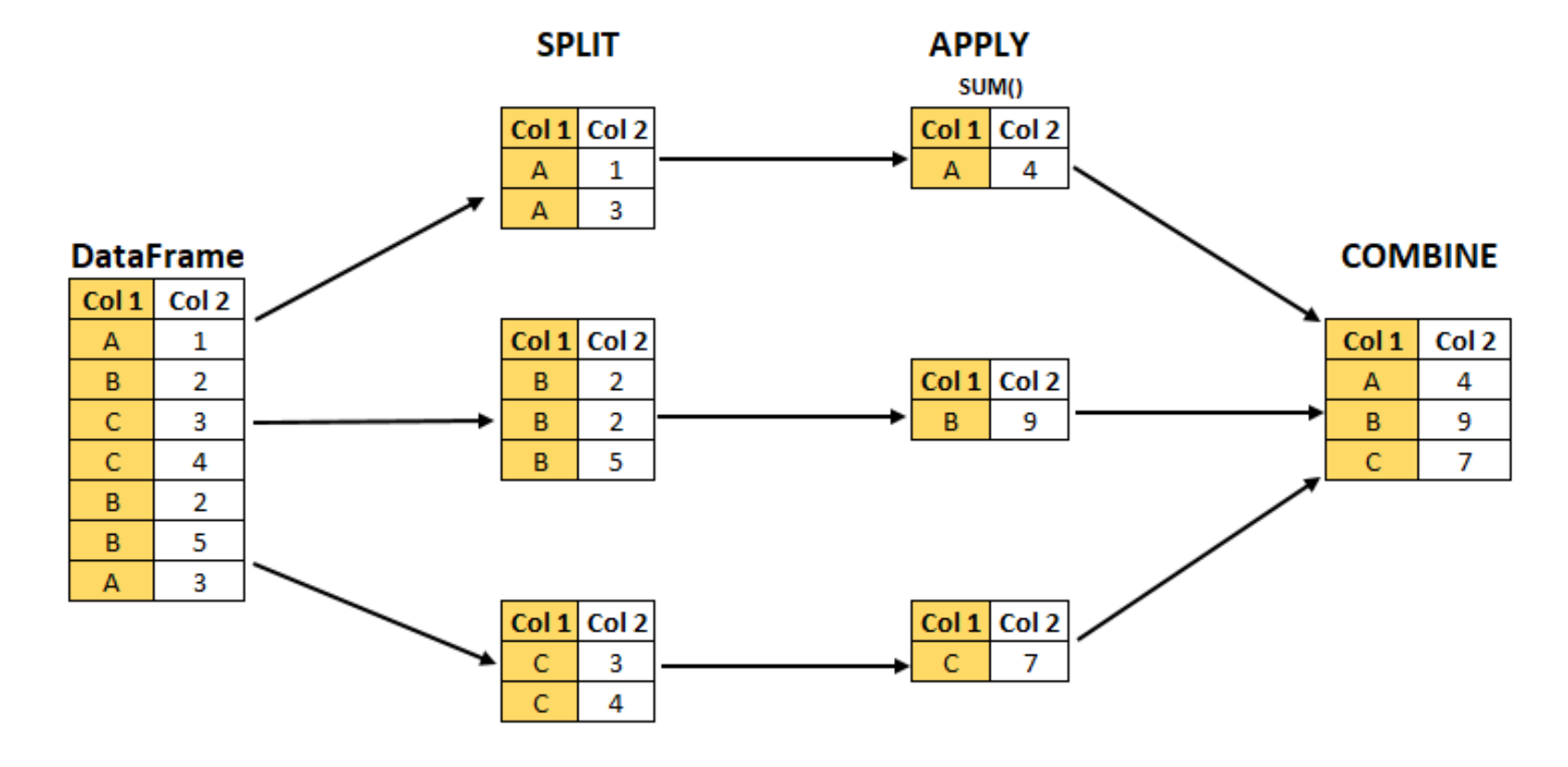
```
head(datos[, 1:7], 15)
```

```
##    CIRCUITO SERIE HABILITADO OBSERVADOS T_EMITIDOS EN_BLANCO ANULADOS
## 1         1   AAA        389          8        310         4        3
## 2         2   AAA        389          4        348         7        8
## 3         3   AAA        389          2        335         8       10
## 4         4   AAA        389          2        319         4       10
## 5         5   AAA        389          2        340         1        5
## 6         6   AAA        389          4        330         6        6
## 7         7   AAA        389          3        348         6        4
## 8         8   AAA        389          2        318         1       10
## 9         9   AAA        389          2        329         0        7
## 10       10   AAA        389          1        293         2        4
## 11       11   AAA        389          1        298         2        7
## 12       12   AAA        389          3        316         4        7
## 13       13   AAA        389          3        314         2        9
## 14       14   AAA        389          2        304         4        5
## 15       15   AAA        389          3        359         4        5
```

Split-Apply- Combine

Fuente: Filip Ciesielski

Split-Apply- Combine



Fuente: Anurag Pandey

Datos ordenados y desordenados: Soluciones

Soluciones

Hay al menos dos tipos de situaciones -que no son excluyentes- bajos las cuales vamos a tener la necesidad de cambiar la estructura de los datos.

Con cambio de estructura estamos haciendo referencia a pasar de una estructura 'wide' a 'longer'.

- → **Teórico** Muchas veces confundimos columnas con variables (falacia tabular!)
- → **Práctico** Otras veces vamos a querer pasar de una estructura a otra por cuestiones prácticas o de presentación de los datos.

`tidyr::pivot_*()`

Soluciones → tidyr::pivot_longer()

```
datos_long <- tidyr::pivot_longer(datos, 3:13, 'partido', 'votos')
```

```
## # A tibble: 36,988 x 7
##    CIRCUITO SERIE HABILITADO OBSERVADOS T_EMITIDOS partido            votos
##       <dbl> <chr>      <dbl>      <dbl>      <dbl> <chr>              <dbl>
##  1        1 AAA          389          8        310 EN_BLANCO              4
##  2        1 AAA          389          8        310 ANULADOS               3
##  3        1 AAA          389          8        310 SOLO_POR_SI            4
##  4        1 AAA          389          8        310 Partido Frente Am~   117
##  5        1 AAA          389          8        310 Partido Nacional      87
##  6        1 AAA          389          8        310 Partido Colorado      38
##  7        1 AAA          389          8        310 Partido Independi~     5
##  8        1 AAA          389          8        310 Partido Asamblea ~     2
##  9        1 AAA          389          8        310 Partido de los Tr~     0
## 10        1 AAA          389          8        310 Partido Ecologist~     5
## # ... with 36,978 more rows
```

Soluciones → tidyr::pivot_wider()

```
tidyr::pivot_wider(datos_long, names_from = partido, values_from = votos)
```

```
## # A tibble: 2,642 x 19
##    CIRCUITO SERIE HABILITADO OBSERVADOS T_EMITIDOS ANULADOS EN_BLANCO
##       <dbl> <chr>      <dbl>      <dbl>      <dbl>    <dbl>     <dbl>
##  1        1 AAA          389          8        310        3         4
##  2        2 AAA          389          4        348        8         7
##  3        3 AAA          389          2        335       10         8
##  4        4 AAA          389          2        319       10         4
##  5        5 AAA          389          2        340        5         1
##  6        6 AAA          389          4        330        6         6
##  7        7 AAA          389          3        348        4         6
##  8        8 AAA          389          2        318       10         1
##  9        9 AAA          389          2        329        7         0
## 10       10 AAA          389          1        293        4         2
## # ... with 2,632 more rows, and 12 more variables: `Partido Asamblea
## #   Popular` <dbl>, `Partido Cabildo Abierto` <dbl>, `Partido
## #   Colorado` <dbl>, `Partido de la Gente` <dbl>, `Partido de los
## #   Trabajadores` <dbl>, `Partido Digital` <dbl>, `Partido Ecologista
## #   Radical Int.` <dbl>, `Partido Frente Amplio` <dbl>, `Partido
## #   Independiente` <dbl>, `Partido Nacional` <dbl>, `Partido Verde
## #   Animalista` <dbl>, SOLO_POR_SI <dbl>
```

Volvamos a los datos!

Volvamos a trabajar con datos electorales

```
library(magrittr)
datos %>%
    tidyr::pivot_longer(cols = 6:18, names_to = 'partido', values_to = 'votos') %>%
    dplyr::select(partido, votos) %>%
    dplyr::group_by(partido) %>%
    dplyr::summarise_all(list(min = min,
                              media = mean,
                              median = median,
                              maximo = max,
                              suma = sum))
```

```
## # A tibble: 14 x 6
##    partido                          min   media median maximo   suma
##    <chr>                          <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
##  1 ANULADOS                           0   6.95      7       19  18358
##  2 EN_BLANCO                          0   3.61      3       12   9527
##  3 Partido Asamblea Popular           0   4.12      4       20  10888
##  4 Partido Cabildo Abierto            0  28.9      26      117  76410
##  5 Partido Colorado                   0  37.3      30      149  98559
##  6 Partido de la Gente                0   4.43      4       18  11715
##  7 Partido de los Trabajadores        0   0.273     0        4    720
##  8 Partido Digital                    0   1.17      1        9   3079
##  9 Partido Ecologista Radical Int.    0   6.05      6       19  15978
## 10 Partido Frente Amplio              0 166.      170      269 438839
## 11 Partido Independiente              0   4.70      4       18  12415
## 12 Partido Nacional                   0  81.3      79      170 214675
## 13 Partido Verde Animalista           0   3.82      3.5     15  10081
## 14 SOLO_POR_SI                        0   1.67      1       10   4417
```

Datos relacionados

merge and join

```
df1 <- data.frame(id      = 1:5,
                  letras1 = letters[1:5],
                  letras2 = letters[6:10],
                  stringsAsFactors = FALSE)

df2 <- data.frame(id      = 6:10,
                  LETRAS1 = LETTERS[1:5],
                  LETRAS2 = LETTERS[6:10],
                  stringsAsFactors = FALSE)
df1
```

```
##   id letras1 letras2
## 1  1       a       f
## 2  2       b       g
## 3  3       c       h
## 4  4       d       i
## 5  5       e       j
```

```
df2
```

```
##   id LETRAS1 LETRAS2
## 1  6       A       F
## 2  7       B       G
## 3  8       C       H
## 4  9       D       I
## 5 10       E       J
```

merge and join

```r
merge(df1, df2, by = 'id', all = TRUE)
```

```
##    id letras1 letras2 LETRAS1 LETRAS2
## 1   1       a       f    <NA>    <NA>
## 2   2       b       g    <NA>    <NA>
## 3   3       c       h    <NA>    <NA>
## 4   4       d       i    <NA>    <NA>
## 5   5       e       j    <NA>    <NA>
## 6   6    <NA>    <NA>       A       F
## 7   7    <NA>    <NA>       B       G
## 8   8    <NA>    <NA>       C       H
## 9   9    <NA>    <NA>       D       I
## 10 10    <NA>    <NA>       E       J
```

```r
df1$union <- rep('A', 5)
df2$union <- rep('B', 5)
merge(df1, df2, by = 'id', all = TRUE) # PROBLEMA!
```

```
##    id letras1 letras2 union.x LETRAS1 LETRAS2 union.y
## 1   1       a       f       A    <NA>    <NA>    <NA>
## 2   2       b       g       A    <NA>    <NA>    <NA>
## 3   3       c       h       A    <NA>    <NA>    <NA>
## 4   4       d       i       A    <NA>    <NA>    <NA>
## 5   5       e       j       A    <NA>    <NA>    <NA>
## 6   6    <NA>    <NA>    <NA>       A       F       B
## 7   7    <NA>    <NA>    <NA>       B       G       B
## 8   8    <NA>    <NA>    <NA>       C       H       B
## 9   9    <NA>    <NA>    <NA>       D       I       B
## 10 10    <NA>    <NA>    <NA>       E       J       B
```

merge and join

```r
merge(df1, df2, all = TRUE)
```

```
##    id union letras1 letras2 LETRAS1 LETRAS2
## 1   1     A       a       f    <NA>    <NA>
## 2   2     A       b       g    <NA>    <NA>
## 3   3     A       c       h    <NA>    <NA>
## 4   4     A       d       i    <NA>    <NA>
## 5   5     A       e       j    <NA>    <NA>
## 6   6     B    <NA>    <NA>       A       F
## 7   7     B    <NA>    <NA>       B       G
## 8   8     B    <NA>    <NA>       C       H
## 9   9     B    <NA>    <NA>       D       I
## 10 10     B    <NA>    <NA>       E       J
```

```r
merge(df1, df2, by = intersect(names(df1), names(df2)), all = TRUE)
```

```
##    id union letras1 letras2 LETRAS1 LETRAS2
## 1   1     A       a       f    <NA>    <NA>
## 2   2     A       b       g    <NA>    <NA>
## 3   3     A       c       h    <NA>    <NA>
## 4   4     A       d       i    <NA>    <NA>
## 5   5     A       e       j    <NA>    <NA>
## 6   6     B    <NA>    <NA>       A       F
## 7   7     B    <NA>    <NA>       B       G
## 8   8     B    <NA>    <NA>       C       H
## 9   9     B    <NA>    <NA>       D       I
## 10 10     B    <NA>    <NA>       E       J
```

merge and join

Dupliquemos un ’id’ y dupliquemos el nombre de una variable

```
df2[1, "id"] <- 5
df2[1, "id"]  == df1[5, "id"]
```

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

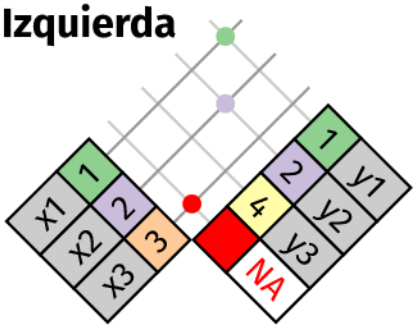
```
names(df1)[2] <- "LETRAS1"
merge(df1, df2, all = TRUE)
```

```
##    id LETRAS1 union letras2 LETRAS2
## 1   1       a     A       f    <NA>
## 2   2       b     A       g    <NA>
## 3   3       c     A       h    <NA>
## 4   4       d     A       i    <NA>
## 5   5       A     B    <NA>       F
## 6   5       e     A       j    <NA>
## 7   7       B     B    <NA>       G
## 8   8       C     B    <NA>       H
## 9   9       D     B    <NA>       I
## 10 10       E     B    <NA>       J
```

`dplyr::*_join()`

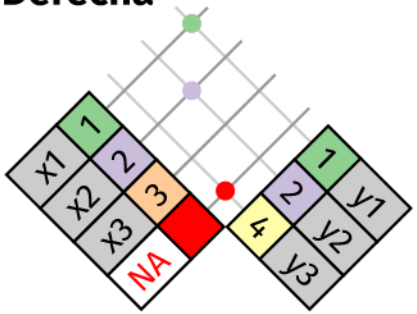
merge and join

Izquierda



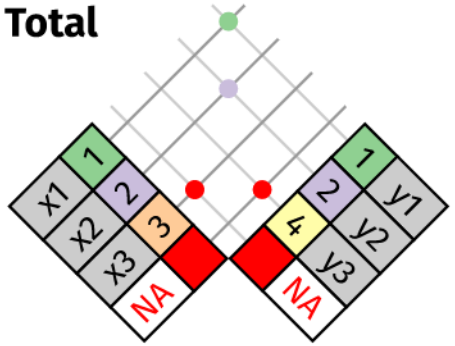
llave	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Derecha



llave	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

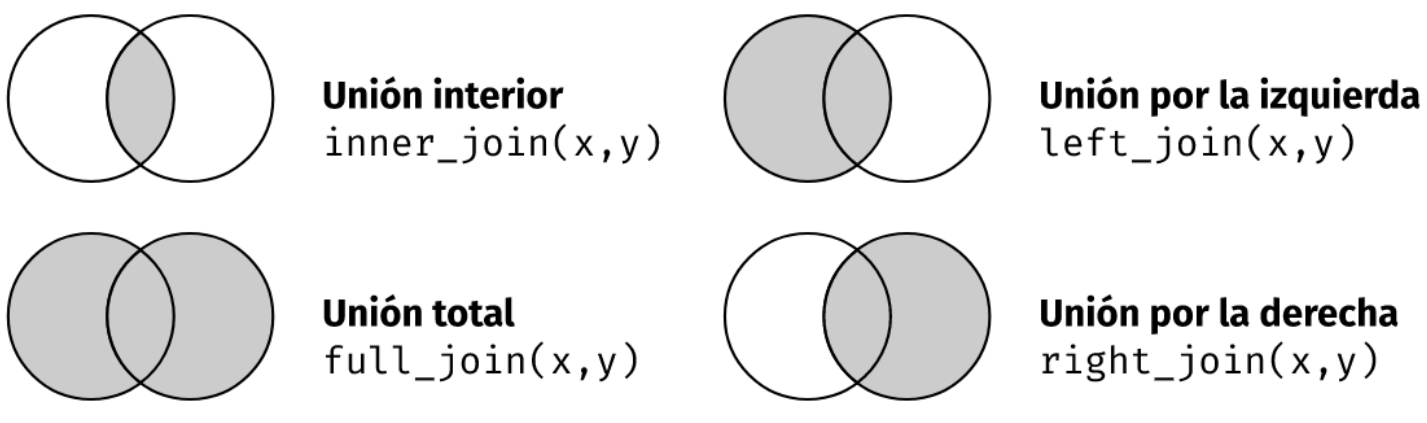
Total



llave	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Fuente: es.r4ds

merge and join



Fuente: es.r4ds

`merge()` *vs.* `*_join()`

merge and join

¿argumentos o funciones?

dplyr::*_join()		base::merge()
`inner_join(x, y)`	$\leftrightarrow$	`merge(x, y)`
`left_join(x, y)`	$\leftrightarrow$	`merge(x, y, all.x = TRUE)`
`right_join(x, y)`	$\leftrightarrow$	`merge(x, y, all.y = TRUE)`
`full_join(x, y)`	$\leftrightarrow$	`merge(x, y, all= TRUE)`

data.frame y tibble

tibbls

Los tibbls son `data.frame` pero con algunas características adicionales y con otras distintas.

Un `tibble` se crea con la función `tibble` del paquete `tibble`.
Tambien hay una funcion para convertir: `tibble::as_tibble()`

Creando tibbles

```r
data.frame(variable_1 = 1:5,
           variable_2 = 101:105,
           variable_3 = sample(letters, 5))
```

```
##   variable_1 variable_2 variable_3
## 1          1        101          s
## 2          2        102          l
## 3          3        103          q
## 4          4        104          i
## 5          5        105          a
```

```r
tibble::tibble(variable_1 = 1:5,
               variable_2 = 101:105,
               variable_3 = sample(letters, 5))
```

```
## # A tibble: 5 x 3
##   variable_1 variable_2 variable_3
##        <int>      <int> <chr>
## 1          1        101 r
## 2          2        102 g
## 3          3        103 w
## 4          4        104 l
## 5          5        105 x
```

Creando tibbles

Una de las características que se dice que tienen los tibbles es que acepta nombres no sintácticos a diferencia de los `data.frame`. Esto no es estrictamente cierto.

```
data.frame('1' = 1:3, a = 1:3)
```

```
##   X1 a
## 1  1 1
## 2  2 2
## 3  3 3
```

```
tibble::tibble('1' = 1:3, a = 1:3)
```

```
## # A tibble: 3 x 2
##     `1`     a
##   <int> <int>
## 1     1     1
## 2     2     2
## 3     3     3
```

```
data.frame('1' = 1:3, a = 1:3, check.names = FALSE)
```

```
##   1 a
## 1 1 1
## 2 2 2
## 3 3 3
```

Innovaciones de los tibbles

→ Impresión en pantalla
  - Muestra las primeras 10 filas y las columnas que entran en el ancho de la pantalla,
  - Muestra debajo del nombre de cada variable el tipo de dato que contiene la variable.

→ Indexación: todas las maneras de indexar devuelven un tibble.

→ No-factor: un tibble nunca coerciona los datos de tipo character a factor.

→ Los nombres de filas se eliminan en las conversiones. Hay una función que permite pasar estos nombres a una variable (`rownames_to_column()`)

→ Referencias inmediatas en la creación de variables

Ejemplos: stringsAsFactors

```
str(data.frame(a = 1:12, b = month.name))
```

```
## 'data.frame':^^I12 obs. of  2 variables:
##  $ a: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ b: Factor w/ 12 levels "April","August",..: 5 4 8 1 9 7 6 2 12 11 ...
```

```
str(tibble::tibble(a = 1:12, b = month.name))
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':^^I12 obs. of  2 variables:
##  $ a: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ b: chr  "January" "February" "March" "April" ...
```

```
str(data.frame(a = 1:12, b = month.name, stringsAsFactors = FALSE))
```

```
## 'data.frame':^^I12 obs. of  2 variables:
##  $ a: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ b: chr  "January" "February" "March" "April" ...
```

Ejemplos: print()

```
tibble::as_tibble(iris)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            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           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

Ejemplos: print()

```
iris
```

```
##     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
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 25           4.8         3.4          1.9         0.2     setosa
## 26           5.0         3.0          1.6         0.2     setosa
## 27           5.0         3.4          1.6         0.4     setosa
## 28           5.2         3.5          1.5         0.2     setosa
## 29           5.2         3.4          1.4         0.2     setosa
## 30           4.7         3.2          1.6         0.2     setosa
## 31           4.8         3.1          1.6         0.2     setosa
```

Ejemplos: print()

```
print(tibble::as_tibble(iris), n = 20)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            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           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## 11          5.4         3.7          1.5         0.2 setosa
## 12          4.8         3.4          1.6         0.2 setosa
## 13          4.8         3            1.4         0.1 setosa
## 14          4.3         3            1.1         0.1 setosa
## 15          5.8         4            1.2         0.2 setosa
## 16          5.7         4.4          1.5         0.4 setosa
## 17          5.4         3.9          1.3         0.4 setosa
## 18          5.1         3.5          1.4         0.3 setosa
## 19          5.7         3.8          1.7         0.3 setosa
## 20          5.1         3.8          1.5         0.3 setosa
## # ... with 130 more rows
```

Ejemplos: nombres de filas

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
tibble::as_tibble(tibble::rownames_to_column(mtcars))
```

```
## # A tibble: 32 x 12
##    rowname   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear
##    <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Mazda ~  21       6  160    110  3.9   2.62  16.5     0     1     4
##  2 Mazda ~  21       6  160    110  3.9   2.88  17.0     0     1     4
##  3 Datsun~  22.8     4  108     93  3.85  2.32  18.6     1     1     4
##  4 Hornet~  21.4     6  258    110  3.08  3.22  19.4     1     0     3
##  5 Hornet~  18.7     8  360    175  3.15  3.44  17.0     0     0     3
##  6 Valiant  18.1     6  225    105  2.76  3.46  20.2     1     0     3
##  7 Duster~  14.3     8  360    245  3.21  3.57  15.8     0     0     3
##  8 Merc 2~  24.4     4  147.    62  3.69  3.19  20       1     0     4
##  9 Merc 2~  22.8     4  141.    95  3.92  3.15  22.9     1     0     4
## 10 Merc 2~  19.2     6  168.   123  3.92  3.44  18.3     1     0     4
## # ... with 22 more rows, and 1 more variable: carb <dbl>
```

Ejemplos: referencias internas

```
data.frame(
            x = 1:5,
            y = x * 2
          )
```

```
## Error in data.frame(x = 1:5, y = x * 2): object 'x' not found
```

```
tibble::tibble(
               x = 1:5,
               y = x * 2
              )
```

```
## # A tibble: 5 x 2
##       x     y
##   <int> <dbl>
## 1     1     2
## 2     2     4
## 3     3     6
## 4     4     8
## 5     5    10
```

tribbles

```
tibble::tribble(
        ~var1, ~var2, ~var3,
        1,2,3,
        4,5,6,
        7,8,9,
        10, 11, 12,
        13, 14, 15
    )
```

```
## # A tibble: 5 x 3
##    var1  var2  var3
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
## 3     7     8     9
## 4    10    11    12
## 5    13    14    15
```

Exploración de datos con dplyr

dplyr

El paquete 'dplyr' es muy usado por la comunidad R. Ofrece un conjunto de funciones que en su mayoría ya existen en el paquete base pero tienen modificaciones que hacen que el flujo de trabajo pueda ser mejor. Es un paquete que tiene más de 70 funciones!, si, 70!!.

Principales funciones

`filter()`	→	filtrar filas (`subset(), ']'`)
`select()`	→	seleccionar columnas (`subset(), ']'`)
`arrange()`	→	reordenar (`order()`)
`mutate()`	→	transformar, mutar (`transform()`)
`summarise()`	→	Resumir datos
`group_by()`	→	agrupar (`split()`)

Ejemplo

```r
names(datos)
```

```
##  [1] "CIRCUITO"                      "SERIE"
##  [3] "HABILITADO"                    "OBSERVADOS"
##  [5] "T_EMITIDOS"                    "EN_BLANCO"
##  [7] "ANULADOS"                      "SOLO_POR_SI"
##  [9] "Partido_Frente_Amplio"         "Partido_Nacional"
## [11] "Partido_Colorado"              "Partido_Independiente"
## [13] "Partido_Asamblea_Popular"      "Partido_de_los_Trabajadores"
## [15] "Partido_Ecologista_Radical_Int." "Partido_de_la_Gente"
## [17] "Partido_Verde_Animalista"      "Partido_Digital"
## [19] "Partido_Cabildo_Abierto"
```

```r
select(filter(datos, Partido_Frente_Amplio < 25), 'CIRCUITO')
```

```
##   CIRCUITO
## 1      961
## 2      962
## 3      966
## 4      967
## 5     2615
```

```r
count(as_tibble(select(filter(datos, Partido_Frente_Amplio < 25), 'CIRCUITO')))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     5
```

Ejemplo

```
d <- select(datos, contains("Partido"))
names(d)
```

```
##  [1] "Partido_Frente_Amplio"           "Partido_Nacional"
##  [3] "Partido_Colorado"                "Partido_Independiente"
##  [5] "Partido_Asamblea_Popular"        "Partido_de_los_Trabajadores"
##  [7] "Partido_Ecologista_Radical_Int." "Partido_de_la_Gente"
##  [9] "Partido_Verde_Animalista"        "Partido_Digital"
## [11] "Partido_Cabildo_Abierto"
```

- starts_with()
- ends_with()
- matches()
- everything()

La función select tiene como argumento '...', que permite seleccionar la cantidad deseada de variables. Se pueden excluir -variable, -c(variable1, variable2) o establecer secuencias, variable1:variable4 (de la uno a la 4)

Pipe

Ejemplo

El operador pipe (%>%) que pertenece al paquete 'magrittr' tiene como objetivo principal mejorar la legibilidad del código.

Funcionalmente, lo que hace es pasar la salida de una función y pasarla a la siguiente función.

Ejemplo

Objetivo: identificar cual es la letra que se repite mas en el objeto llamado 'acrhivo'.

```
archivo = "ascasdaaaaasdasdadadshgywyefgqoyefqeuyr2t3642876294872"

names(which.max(table(unlist(strsplit(stringr::str_remove_all(archivo, pattern = "[^[A-z]]"), "")))))
```

```
## [1] "a"
```

```
library(magrittr)

archivo %>%
    stringr::str_remove_all(pattern = "[^[A-z]]") %>%
    strsplit(., "") %>%
    unlist() %>%
    table() %>%
    which.max() %>%
    names()
```

```
## [1] "a"
```

Ejemplo

Objetivo: saber la cantidad de circuitos en los que el partido FA obtuvo menos de 25 votos.

```
count(select(filter(datos, Partido_Frente_Amplio < 25), 'CIRCUITO'))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     5
```

```
datos_fa <- filter(datos, Partido_Frente_Amplio < 25) # solo con esto ya se responde!
datos_fa <- select(datos_fa, 'CIRCUITO')
count(datos_fa)
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     5
```

```
datos %>%
    filter(Partido_Frente_Amplio < 25) %>%
    select(CIRCUITO) %>%
    count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     5
```