

Manipulación de datos, lo básico pero en

Curso de introducción a

Pablo Cabrera-Álvarez

 |  @pablocalv

Julio 2019



El data frame `mtcars` está contenido en la lista `datasets`. ¿Cómo calcularías la media de la variable `mpg` del data frame `mtcars`?

```
str(datasets)
```

```
## List of 1
## $ : 'data.frame':  32 obs. of  11 variables:
##   ..$ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##   ..$ cyl : num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
##   ..$ disp: num [1:32] 160 160 108 258 360 ...
##   ..$ hp  : num [1:32] 110 110 93 110 175 105 245 62 95 123 ...
##   ..$ drat: num [1:32] 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##   ..$ wt  : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
##   ..$ qsec: num [1:32] 16.5 17 18.6 19.4 17 ...
##   ..$ vs  : num [1:32] 0 0 1 1 0 1 0 1 1 1 ...
##   ..$ am  : num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
##   ..$ gear: num [1:32] 4 4 4 3 3 3 3 4 4 4 ...
##   ..$ carb: num [1:32] 4 4 1 1 2 1 4 2 2 4 ...
```

Quiz

El data frame `mtcars` está contenido en la lista `datasets`. ¿Cómo calcularías la media de la variable `mpg` del data frame `mtcars`?

```
mean(datasets[[1]]$mpg)
```

```
## [1] 20.09062
```



¿Cómo calcularías la media de `peso`?

```
str(peso)
```

```
## Factor w/ 10 levels "53.0513470154256",...: 6 9 7 4 8 10 2 5 1 3
```

```
peso <- as.numeric(as.character(peso))  
mean(peso)
```

```
## [1] 68.30417
```

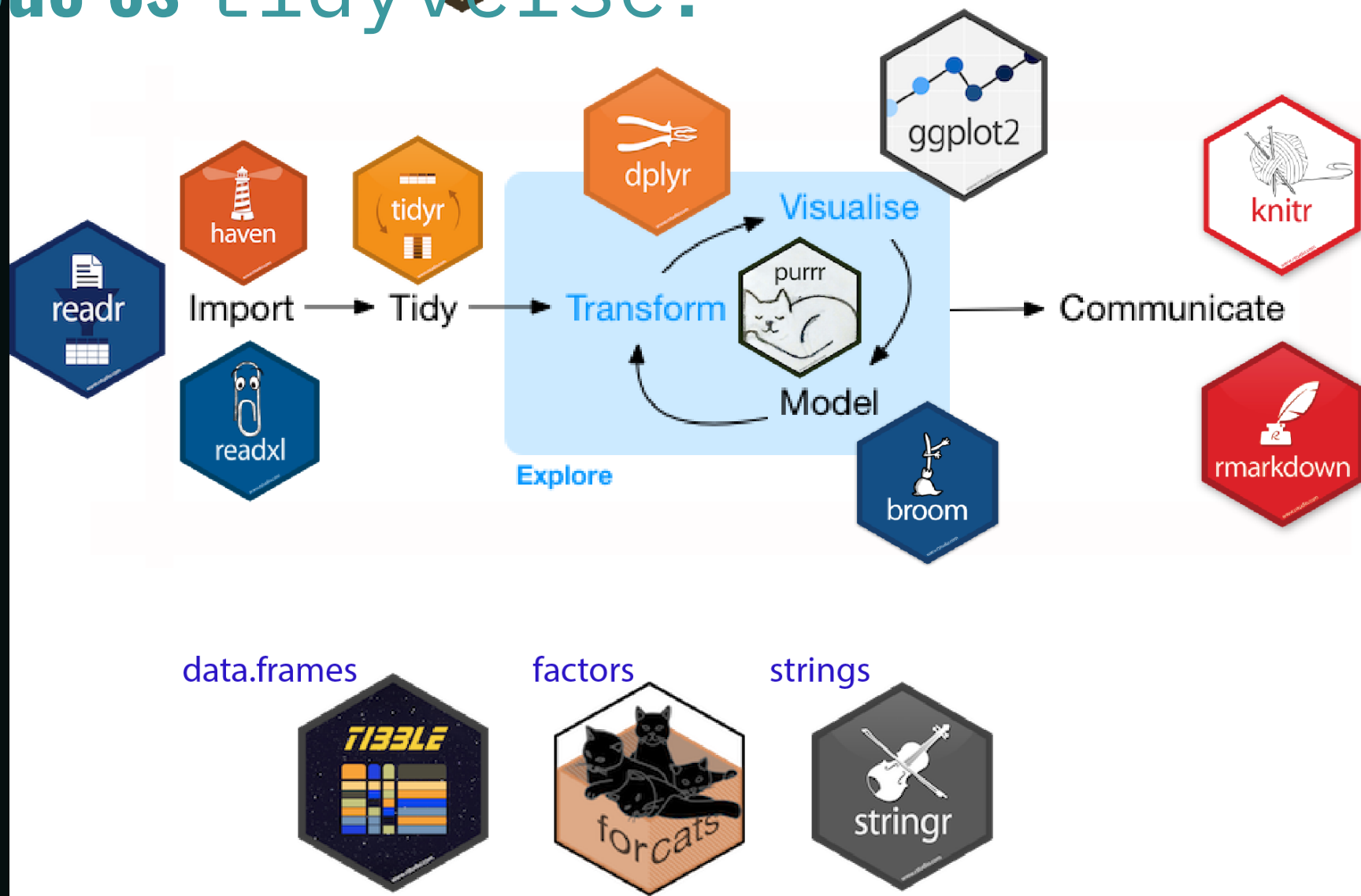
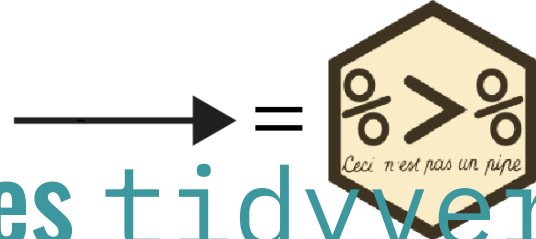
Manipulación de datos, lo básico pero en R

- Cargar y guardar datos en R
- Explorar datos
- Seleccionar variables
- Renombrar variables
- Ordenar casos
- Filtrar casos
- Transformar variables

¿Qué es tidyverse?

tidyverse

¿Qué es tidyverse?



Tidyverse

- Cada *acción* corresponde con un *verbo*:
 - `read_csv()`: cargar archivos
 - `select()`: seleccionar variables
 - `rename()`: renombrar variables
 - `arrange()`: ordenar casos
 - `filter()`: filtrar casos
 - `mutate()`: transformar variables
- Todas las funciones (excepto las de tipo `read`) siguen el mismo sistema:

```
select(.data = , ...)
```


Tidyverse

- Siempre **asignamos** la **transformación** a un nombre para guardarla, generalmente el mismo nombre (pensar en el trabajo con un dataset en SPSS o Stata).

```
mtcars_small <- select(mtcars, mpg, wt, cyl)
head(mtcars_small, n = 3)
```

```
##           mpg    wt  cyl
## Mazda RX4   21.0 2.620    6
## Mazda RX4 Wag 21.0 2.875    6
## Datsun 710   22.8 2.320    4
```

Cargar means read 📖

Paquetes para cargar datos

- base: `load()`
- readr: `read_rds()`, `read_csv()`, `read_csv2()`, `read_tsv()`, `read_delim()`, `read_fwf()`
- readxl: `read_xls()`, `read_xlsx()`
- haven: `read_spss()`, `read_stata()`
- googlesheets: `gs_read()`

Cargar .RData

`load(file) | save(..., file) | save.image(file)`

El formato natural de los datos en **R** es `.RData`. Para **cargar** este tipo de archivos se utiliza la función `load()`:

```
load(file = "data/my_data.RData")
```

Al utilizar `load()` los datos se cargarán con su nombre cuando fueron guardados.

Para **guardar un objeto** en formato `.RData` se pueden utilizar la función `save()`, en caso de guardar uno o varios objetos, o la función `save.image()` si se quiere guardar una copia de todos los objetos en el *environment* en ese momento.

```
save(x, y, file = "data/my_data.RData")  
save.image(file = "data/my_data.RData")
```

Cargar .RDS

```
read_rds(path) | write_rds(x, path)1
```

Otro formato propio de **R** es el **.RDS** en que se **almacena un único objeto** (p. ej. vector, lista, data frame...). Al cargar un archivo **.RDS** se debe asignar a un nombre con el fin de almacenarlo en el espacio de trabajo:

```
read_rds(path = "data/my_data.RDS") # No será almacenado en la memoria  
my_data <- read_rds(path = "data/my_data.RDS") # Será almacenado como objeto my_data
```

Para **guardar** un objeto en un archivo **.RDS** se emplea la función `write_rds()`:

```
write_rds(x = x, path = "data/my_data.RDS")
```

[1] Existen versiones de estas funciones en el paquete base `readRDS(path)` y `saveRDS(x, path)`. Se desarrollan las versiones de `readr` por consistencia con `tidyverse`.

Cargar datos .csv

```
read_csv(file, col_names = TRUE, quote = "\"") | write_csv(x, path)2
```

El formato con el que tradicionalmente se han cargado datos en **R** es el **fichero de texto** separado por comas **.csv**. El argumento **colnames** hace referencia a la **primera fila del archivo .csv**, en caso de que contenga los nombres de las variables (**TRUE**).

```
my_data <- read_csv(file = "data/my_data.csv")
```

Para exportar un archivo en formato **.csv**:

```
write_csv(x = x, path = "data/my_data.csv")
```

[2] Existen versiones de estas funciones en el paquete base **read.csv(path, header = TRUE)** y **write.csv(x, path)**.

Los archivos csv con ;

```
read_csv2(file, col_names = TRUE, quote = "\"") | write_csv2(x, path)3
```

! Un problema para los no GB US es el **uso de la coma y los puntos** en los números. El archivo `.csv` guardado en sistemas de influencia europea está separado por `;` en vez de por `,`.

✓ Esto hay que tenerlo en cuenta a la hora de cargar los ficheros. Así, los ficheros separados por `;` se cargan con el comando `read_csv2()`:

```
my_data <- read_csv2(file = "data/my_data.csv")
```

Para guardar los datos en un `.csv` separado por `;`:

```
write_csv2(x = x, path = "data/my_data.csv")
```

[3] Existen versiones de estas funciones en el paquete base `read.csv(path, header = TRUE)` y `write.csv(x, path)`.

Cargar .txt

```
read_tsv(file, col_names = TRUE, quote = "\"") | read_delim(file, delim, col_names = TRUE, quote = "\"") | read_fwf(file, col_positions)
```

- En otras ocasiones los datos están en archivos de texto `.txt`. Si el archivo está separado por tabuladores, entonces usar `read_tsv()`.
- En caso de que el archivo utilice cualquier otro delimitador, utilizar `read_delim()`, especificando el delimitador en el argumento `delim`, por ejemplo: `delim = "\$"`.
- En algunos casos los archivos vienen con ancho fijo de columna y la especificación de la estructura del fichero (p. ej. INE). En esos casos hay que especificar la estructura en el argumento `col_positions` de la función `read_fwf()`.

```
my_data <- read_fwf(file = "data/my_data.txt",  
                    fwf_widths(c(20, 10, 12),  
                                c("nombre", "apellidos", "edad")))
```


Encoding

! Un problema al usar los archivos en `es` es la inclusión de caracteres especiales como por ejemplo ñ o los **acentos**. Leer archivos de este tipo puede dar problemas en las variables de cadenas de texto o en los niveles de los factores.

✓ Para solucionarlo hay que indicar en la función el tipo de **encoding** que tienen los datos en origen. Esto se hace a partir del argumento `locale` de las funciones del paquete `readr`. En el caso del español, suele funcionar el `"Latin2"` o `"Latin1"`. Por ejemplo:

```
my_data <- read_csv2(file = "data/my_data.csv",  
                     locale = locale(encoding = "Latin2"))
```

Cargar .xlsx y .xls

```
read_xlsx(path, sheet, col_names = TRUE) | read_xls(path, sheet, col_names = TRUE)
```

Los archivos originarios de **MS Excel** son leídos con las funciones `read_xlsx()` y `read_xls()`:

```
my_data <- read_xlsx(path = "data/my_data.xlsx", sheet = "Datos agosto")
```

Cargar .sav y .dta

```
read_spss(file, user_na) | read_stata(file)
```

Con el paquete `haven` también se pueden leer archivos `.sav` de `SPSS` o `.dta` de `Stata`. El paquete `haven` crea atributos adicionales de los vectores para evitar la pérdida de información como las etiquetas de los valores o variables.

```
my_data <- read_spss(file = "data/my_data.sav", user_na = TRUE)
```

```
my_data <- read_stata(file = "data/my_data.dta")
```

Trabajar con labelled

```
as_factor(df) | zap_labels(df)
```

! Cargar los ficheros con [haven](#) tiene la ventaja de que se pierde el mínimo de información, sin embargo la estructura de datos resultante no es completamente compatible con [R](#)⁴. Por ejemplo:

```
str(my_data)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2487 obs. of  1 variable:
## $ P1: 'haven_labelled' num  3 4 4 4 3 3 3 5 3 4 ...
## ..- attr(*, "label")= chr "Valoración de la situación económica general de España"
## ..- attr(*, "format.spss")= chr "F1.0"
## ..- attr(*, "display_width")= int 4
## ..- attr(*, "labels")= Named num  1 2 3 4 5 8 9
## .. ..- attr(*, "names")= chr  "Muy buena" "Buena" "Regular" "Mala" ...
```

[4] La explicación de los creadores de [haven](#) aquí. Una forma de importar de forma directa los datos con factores es con el paquete [foreign](#) y la función [read.spss\(\)](#).

Trabajar con labelled

- ✓ Para trabajar con datos totalmente compatibles con R existen dos posibilidades, covertir los datos etiquetados en factores con `as_factor()` o en variables *numeric/character* con `zap_labels()`.

```
my_data_factor <- as_factor(my_data)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2487 obs. of  1 variable:
## $ P1: Factor w/ 7 levels "Muy buena","Buena",...: 3 4 4 4 3 3 3 5 3 4 ...
## ..- attr(*, "label")= chr "Valoración de la situación económica general de España"
```

```
my_data_unlabel <- zap_labels(my_data)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2487 obs. of  1 variable:
## $ P1: num  3 4 4 4 3 3 3 5 3 4 ...
## ..- attr(*, "label")= chr "Valoración de la situación económica general de España"
## ..- attr(*, "format.spss")= chr "F1.0"
## ..- attr(*, "display_width")= int 4
```

Cargar .googlesheets

```
gs_auth() | gs_title(title) | gs_read(ss)
```

1. **Autenticarse** con la cuenta de google (opcionalmente se puede guardar el token⁵):

```
gs_auth()
```

1. **Localizar** la hoja de .googlesheets por el título:

```
my_gsheet <- gs_title(title = "my_data")
```

1. **Cargar** el fichero en R:

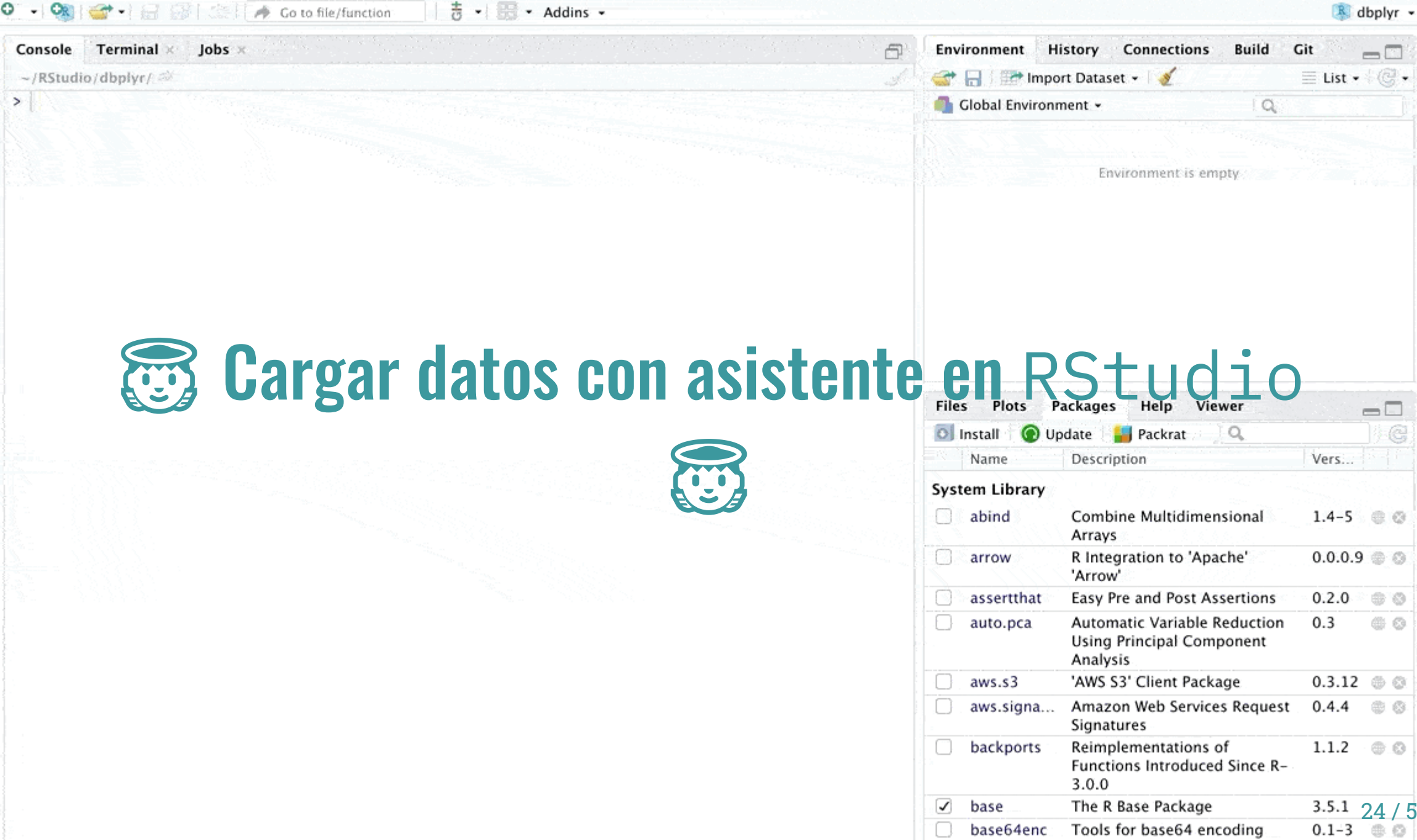
```
my_data <- gs_read(ss = my_gsheet)
```

[5] Más información sobre cómo guardar y reutilizar el token aquí.

Si esto es solo para cargar los datos...



Cargar datos con asistente en RStudio



The screenshot shows the RStudio interface. The top bar includes 'Go to file/function' and 'Addins'. The left pane has tabs for 'Console', 'Terminal', and 'Jobs'. The main editor area shows a file path '~ /RStudio/dbplyr/'. The right pane has tabs for 'Environment', 'History', 'Connections', 'Build', and 'Git'. The 'Environment' tab shows 'Global Environment' and 'Environment is empty'. The 'Packages' tab shows a list of installed and available packages.

Files	Plots	Packages	Help	Viewer
Install	Update	Packrat		
Name	Description	Vers...		
System Library				
<input type="checkbox"/> abind	Combine Multidimensional Arrays	1.4-5		
<input type="checkbox"/> arrow	R Integration to 'Apache' 'Arrow'	0.0.0.9		
<input type="checkbox"/> assertthat	Easy Pre and Post Assertions	0.2.0		
<input type="checkbox"/> auto.pca	Automatic Variable Reduction Using Principal Component Analysis	0.3		
<input type="checkbox"/> aws.s3	'AWS S3' Client Package	0.3.12		
<input type="checkbox"/> aws.signa...	Amazon Web Services Request Signatures	0.4.4		
<input type="checkbox"/> backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.2		
<input checked="" type="checkbox"/> base	The R Base Package	3.5.1		
<input type="checkbox"/> base64enc	Tools for base64 encoding	0.1-3		

**Explorar los datos means view (and glimpse,
head, summary...) 🙄 👊**

Vista de datos

`View(x)`

- Vamos a utilizar los datos `gss_cat`, una submuestra de los datos de la Encuesta General Social de EE.UU. Los puedes encontrar tecleando `gss_cat` una vez que esté cargado el paquete `tidyverse`.
- Para **ver la matriz de datos** una opción es imprimir el data frame, otra posibilidad es utilizar la función `View()`.

```
View(gss_cat)
```

Explorar los casos

```
head(x, n= 6) | tail(x, n = 6)
```

- Para observar los **n primeros** casos del conjunto de tados se utiliza `head()`:

```
head(x = gss_cat, n = 1)
```

```
## # A tibble: 1 x 9
##   year marital    age race  rincome    partyid  relig  denom    tvhours
##   <int> <fct>    <int> <fct> <fct>    <fct>    <fct> <fct>    <int>
## 1  2000 Never mar...   26 White $8000 to ... Ind,nea... Prote... Southern...    12
```

- La función `tail()` se emplea para imprimir los **n últimos** casos:

```
tail(x = gss_cat, n = 1)
```

```
## # A tibble: 1 x 9
##   year marital    age race  rincome    partyid  relig  denom    tvhours
##   <int> <fct>    <int> <fct> <fct>    <fct>    <fct> <fct>    <int>
## 1  2014 Widowed    71 White $20000 - 24... Ind,near r... Protest... Other        2
```

Estructura de los datos

`glimpse(x)`

- Tanto `str()` como `glimpse()` son formas alternativas de **explorar la estructura** del data frame:

```
str(object = gss_cat[,1:3])
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    21483 obs. of  3 variables:
## $ year      : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
## $ marital: Factor w/ 6 levels "No answer","Never married",...: 2 4 5 2 4 6 2 4 6 6 ...
## $ age       : int  26 48 67 39 25 25 36 44 44 47 ...
```

```
glimpse(x = gss_cat[,1:3])
```

```
## Observations: 21,483
## Variables: 3
## $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 20...
## $ marital <fct> Never married, Divorced, Widowed, Never married, Divorce...
## $ age      <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, 51, ...
```

Resumen de los datos

`summary(object)`

- La función `summary()` hace un resumen de un objeto. En el caso de los **data frames** da **información básica sobre las variables** o vectores:

```
summary(object = gss_cat[,1:3])
```

```
##      year      marital      age
## Min.   :2000   No answer   :   17   Min.   :18.00
## 1st Qu.:2002   Never married: 5416   1st Qu.:33.00
## Median :2006   Separated   :   743   Median :46.00
## Mean   :2007   Divorced    : 3383   Mean   :47.18
## 3rd Qu.:2010   Widowed     : 1807   3rd Qu.:59.00
## Max.   :2014   Married     :10117   Max.   :89.00
##                                     NA's    :76
```

Lista de variables

`colnames(x)`

- Para obtener un listado de variables se utiliza la función `colnames()`:

```
colnames(x = gss_cat)
```

```
## [1] "year"      "marital" "age"      "race"      "rincome" "partyid" "relig"  
## [8] "denom"     "tvhours"
```

Seleccionar (variables) means select   

Seleccionar variables

```
select(.data, ...)
```

- Para **seleccionar variables** se emplea la función `select()` de la siguiente forma:

```
gss_cat_red <- select(.data = gss_cat, age, marital)
colnames(gss_cat_red)
```

```
## [1] "age"      "marital"
```

- Para **seleccionar variables que son contiguas** en el data frame se puede utilizar los **dos puntos (:)**:

```
gss_cat_red <- select(.data = gss_cat, age:partyid)
colnames(gss_cat_red)
```

```
## [1] "age"      "race"     "rincome"  "partyid"
```


Descartar columnas

- Se puede **descartar columnas** empleando el **signo menos (-)** antes del nombre de la variable:

```
gss_cat_red <- select(.data = gss_cat, -age, -marital)
colnames(gss_cat_red)
```

```
## [1] "year"      "race"      "rincome"   "partyid"   "relig"     "denom"     "tvhours"
```

- También se puede aplicar el signo menos (-) a las series de **variables contiguas**, pero las variables deben estar entre paréntesis ():

```
gss_cat_red <- select(.data = gss_cat, -(age:partyid))
colnames(gss_cat_red)
```

```
## [1] "year"      "marital"   "relig"     "denom"     "tvhours"
```

Funciones de apoyo a la selección

`starts_with(match) | ends_with(match) | contains(match)`

- Para **seleccionar** todas las variables que **empiezan de la misma forma** se utiliza `starts_with()`:

```
gss_cat_red <- select(.data = gss_cat, starts_with("r"))  
colnames(gss_cat_red)
```

```
## [1] "race"      "rincome" "relig"
```

- Las tres funciones también se pueden utilizar para **descartar variables** si van precedidas del signo menos (-):

```
gss_cat_red <- select(.data = gss_cat, -ends_with("e"))  
colnames(gss_cat_red)
```

```
## [1] "year"      "marital" "partyid" "relig"    "denom"    "tvhours"
```

Funciones de apoyo a la selección

`one_of(...)` | `everything()`

- La función `one_of()` es útil para **seleccionar a partir de un vector** character con los nombres de las columnas:

```
vars <- c("age", "partyid")
gss_cat_red <- select(.data = gss_cat, one_of(vars))
colnames(gss_cat_red)
```

```
## [1] "age"      "partyid"
```

- La función `everything()` se refiere a todas las variables y puede ser útil para **reordenar el data frame**:

```
gss_cat_red <- select(.data = gss_cat, tvhours, age, everything())
colnames(gss_cat_red)
```

```
## [1] "tvhours" "age"      "year"      "marital" "race"      "rincome" "partyid"
## [8] "relig"    "denom"
```

Renombrar means rename



Renombrar variables

`rename(.data, ...)`

- Con la función `rename()` se pueden **renombrar** las columnas de un data frame. En la función se especificará primero el nombre de la **variable nueva** y después el nombre de la **variable antigua**, separados por un signo igual (=). Cambiamos el nombre de la columna `rincome` a `income_bands`.

```
gss_cat_rename <- rename(gss_cat, income_bands = rincome)
colnames(gss_cat_rename)
```

```
## [1] "year"      "marital"   "age"       "race"
## [5] "income_bands" "partyid"   "relig"     "denom"
## [9] "tvhours"
```

Ordenar means arrange 

Ordenar casos

```
arrange(.data, ...) | desc()
```

- La función `arrange()` sirve para ordenar los datos según el criterio de una o varias variables de forma ascendente
- Para ordenar por una o más variables de forma descendente, el nombre de la variable debe ser pasado por la función `desc()`.

```
arrange(.data = gss_cat, age, desc(tvhours))
```

```
## # A tibble: 21,483 x 9
##   year marital    age race  rincome  partyid  relig  denom  tvhours
##   <int> <fct>    <int> <fct> <fct>    <fct>    <fct> <fct>    <int>
## 1  2010 Never ma...   18 Black Refused  Independe... Moslem... Not ap...    13
## 2  2006 Never ma...   18 Black Lt $1000 Ind,near ... Protes... Baptis...     7
## 3  2010 Never ma...   18 Black Not appl... Not str r... Protes... Other      5
## 4  2010 Never ma...   18 White Don't kn... Not str d... Cathol... Not ap...     5
## 5  2012 Never ma...   18 Other Not appl... Independe... Cathol... Not ap...     5
## 6  2014 Never ma...   18 White Not appl... Other par... None      Not ap...     5
## 7  2000 Never ma...   18 White Don't kn... Independe... Cathol... Not ap...     4
```

Filtrar (casos) means filter 🇪🇸

Filtrar casos

`filter(.data, ...)`

- Con la función `filter()` se selecciona a un grupo de casos en base a una condición, por ejemplo, los que tengan 18 años:

```
summary(gss_cat$age)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's  
##      18.00   33.00   46.00   47.18   59.00   89.00    76
```

```
gss_cat_filter <- filter(.data = gss_cat, age == 18)  
summary(gss_cat_filter$age)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.  
##       18       18       18       18       18       18
```

- Los operadores lógicos como `&`, `|` y `!` pueden ser utilizados para hacer selecciones complejas. La coma `,` equivale a `&`.

Filtrar a partir de factores

- Para usar `filter()` con factores se debe utilizar los `levels` de la variable:

```
summary(gss_cat[, c("marital", "age")])
```

```
##           marital           age
## No answer      :    17   Min.   :18.00
## Never married: 5416   1st Qu.:33.00
## Separated      :   743   Median :46.00
## Divorced        : 3383   Mean    :47.18
## Widowed         : 1807   3rd Qu.:59.00
## Married         :10117   Max.    :89.00
##                NA's     :76
```

Filtrar a partir de factores

- Para usar `filter()` con factores se debe utilizar los `levels` de la variable:

```
gss_cat_filter <- filter(.data = gss_cat, marital == "Married", age == 18)
summary(gss_cat_filter[, c("marital", "age")])
```

```
##           marital      age
## No answer      :0   Min.   :18
## Never married:0   1st Qu.:18
## Separated      :0   Median :18
## Divorced        :0   Mean    :18
## Widowed         :0   3rd Qu.:18
## Married         :2   Max.    :18
```

Funciones de apoyo al filtrado

`between(x, left, right)`

- La función `between()` es útil para filtrar los casos que en la variable se encuentran **entre dos valores**. Ahora seleccionamos a aquellos que están entre 18 y 21 años:

```
summary(gss_cat$age)
```

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
##	18.00	33.00	46.00	47.18	59.00	89.00	76

```
gss_cat_filter <- filter(.data = gss_cat, between(age, 18, 21))  
summary(gss_cat_filter$age)
```

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	18.00	19.00	20.00	19.82	21.00	21.00

Segmentar un data frame

```
group_split(.tbl)
```

- Para **segmentar un data frame** por uno o varios factores se puede utilizar la función `group_split()`, que generará un objeto de tipo `list` con tantos data frames en su interior como categorías tenga el factor.

```
gss_cat_split <- group_split(gss_cat, race)
```

Quiz

¿Cómo filtrar los casos de `gss_cat` que **no tienen un valor perdido NA** en la variable `tvhours`?

```
summary(gss_cat$tvhours)
```

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
##	0.000	1.000	2.000	2.981	4.000	24.000	10146

```
gss_cat_filter <- filter(gss_cat, !is.na(tvhours))  
summary(gss_cat_filter$tvhours)
```

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.000	1.000	2.000	2.981	4.000	24.000

Transformer means mutate 

Transformar y crear variables

`mutate(.data, ...)`

- Para **transformar o crear** variables se utiliza la función `mutate()`. Una vez especificados los datos, dentro de `mutate` se especifican las transformaciones a realizar separadas por comas.
- Creamos dos variables nuevas en `gss_cat`:
 1. **age2**: edad multiplicada por 2
 2. **age4**: edad multiplicada por 4

```
gss_cat_mutate <- mutate(.data = gss_cat,  
                        age2 = age*2,  
                        age4 = age*4)  
head(x = select(gss_cat_mutate, starts_with("age")), n = 2)
```

```
## # A tibble: 2 x 3  
##   age  age2  age4  
##   <int> <dbl> <dbl>  
## 1    26    52   104  
## 2    48    96   192
```


Funciones de apoyo a la transformación

`ifelse(test, yes, no)`

- La función `ifelse()` es útil para crear una variable a partir de una condición. Por ejemplo, si el caso es `marital = "Married"` las horas de televisión `tvhours` deben dividirse entre 2, generando una nueva variable, `tvhours_new`:

```
gss_cat_mutate <- mutate(.data = gss_cat,  
                          tvhours_new = ifelse(marital == "Married", tvhours/2, tvhours))
```

```
## # A tibble: 6 x 3  
##   marital      mean_tvhours mean_tvhours_new  
##   <fct>          <dbl>          <dbl>  
## 1 No answer      2.56            2.56  
## 2 Never married  3.11            3.11  
## 3 Separated     3.55            3.55  
## 4 Divorced      3.09            3.09  
## 5 Widowed       3.91            3.91  
## 6 Married       2.65            1.33
```

Funciones de apoyo a la transformación

`case_when(...)`

- La función `case_when()` se utiliza dentro de `mutate()` para hacer **transformaciones de variables condicionales** en una o más de una variable. Vamos a crear una variable que identifique a los demócratas (`partyid = "Strong democrat"`) según su estado civil (`marital`).

```
democrat_levels <- c("Not str democrat", "Strong democrat" )
gss_cat_mutate <- mutate(.data = gss_cat,
  partyid_married = case_when(
    partyid %in% democrat_levels & marital == "Married" ~ "Married democrat",
    partyid %in% democrat_levels & marital == "Divorced" ~ "Divorced democrat",
    partyid %in% democrat_levels & marital == "Widowed" ~ "Widowed democrat",
    TRUE ~ "Others"
  ))
```

Funciones de apoyo a la transformación

`recode(...)`

- La función `recode()` puede utilizarse dentro de `mutate()` con el fin de agrupar todas las transformaciones de variables. Para ejemplificarlo vamos a recodificar `partyid` en `democrat`, `republican` y `other` creando una nueva variable `partyid_recode`.

```
democrat_levels <- c("Not str democrat", "Strong democrat" )
gss_cat_mutate <- mutate(.data = gss_cat,
                        partyid_recode = recode(partyid,
                                                "Strong republican" = "republican",
                                                "Not str republican" = "republican",
                                                "Not str democrat" = "democrat",
                                                "Strong democrat" = "democrat",
                                                .default = "other"))
```

Funciones de apoyo a la transformación

```
table(gss_cat_mutate$partyid, gss_cat_mutate$partyid_recode)
```

```
##  
##          other republican democrat  
## No answer      154           0           0  
## Don't know       1           0           0  
## Other party     393           0           0  
## Strong republican  0        2314           0  
## Not str republican  0        3032           0  
## Ind,near rep    1791           0           0  
## Independent     4119           0           0  
## Ind,near dem    2499           0           0  
## Not str democrat  0           0        3690  
## Strong democrat  0           0        3490
```

Otras funciones de apoyo a la transformación

Existen más **funciones auxiliares**⁶ para apoyar la transformación de datos:

- `lead()`
- `lag()`
- `dense_rank()`
- `min_rank()`
- `percent_rank()`
- `row_number()`
- `cume_dist()`
- `ntile()`
- `na_if()`
- `coalesce()`
- `cumsum()`
- `cummean()`
- `cummin()`
- `cummax()`
- `cumany()`
- `cumall()`

[6] Más información sobre las funciones auxiliares a mutate aquí.

Proyecto

Proyecto

Proyecto y workflow

Objetivo: Crear una base de datos con las **notas de la EVaU** de los centros educativos de Castilla-La Mancha y las características de los centros.

1. Notas por centro educativo de la EVaU disponible en la UCLM en PDF.
2. Notas de los centros educativos disponibles en una base de datos de consulta individual de la Consejería de Educación.
3. Combinar las extracciones de las dos fuentes de información.

Hoy..

- Conocer la información y **planificar** las tareas
- Buscar **funciones**