



Introducción a R y Tidyverse

Sesión 05

Laboratorio de Innovación en Salud

2022-06-04



Contenidos

- Manejo de múltiples conjuntos de datos mediante funciones join.
- Cálculo de variables por filas (rowwise)



Exploración competencial

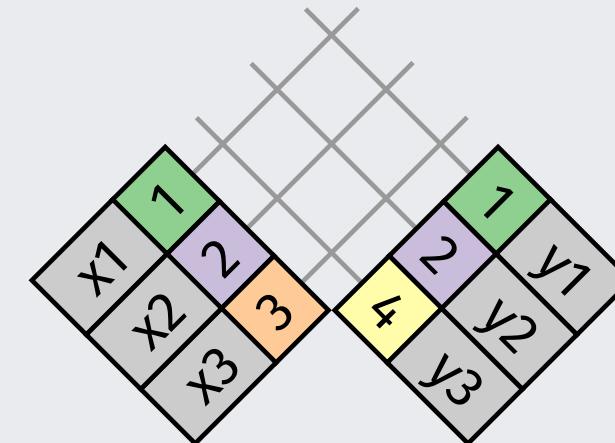


Funciones join

Las funciones `joins` permiten unir 2 o más conjuntos de datos (`dataframes` o `tibbles`) dependiendo de una o múltiples variables en común. En algunos lenguajes (ej. `SQL`) estas variables en común reciben el nombre de `keys`.

- `inner_join()`: Solo casos coincidentes en ambos df's.
- `left_join()`: Todos los casos de la primera df.
- `right_join()`: Todos los casos de la segunda df.
- `full_join()`: Casos de ambas df's.
- `anti_join()`: Solo casos en el primer df que no coincide.

	x	y
1	x1	y1
2	x2	y2
3	x3	y3



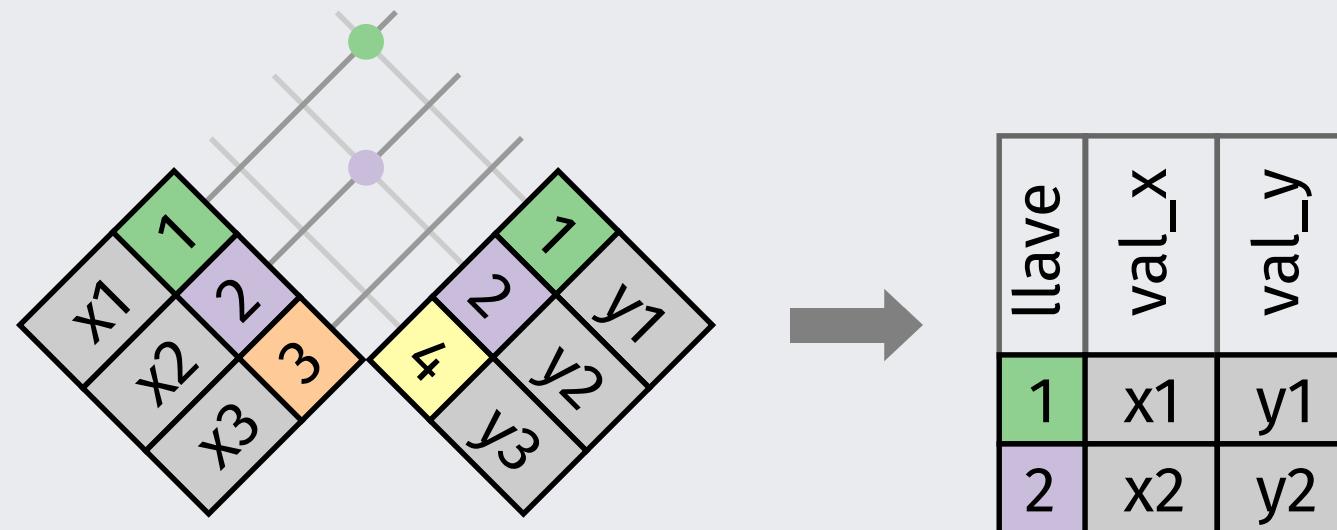
Representación de operaciones joins R para Ciencia de Datos



Función inner_join()

Esta función busca quedarse únicamente con los casos que aparezcan en ambos df analizados a partir de la variable (`key`) en común.

¡Advertencia!: Si en alguno de los df's hay un caso que tenga más de una entrada, entonces en el resultado también se observarán múltiples informaciones para un mismo caso.





Previos a inner_join()

Reconocimiento de BD

Inspección Rec1

Inspección Rec94

Preparativo a joins

Para estas actividades usaremos las bases de datos de ENDES 2020: RECH1.sav y REC94.sav.
Previamente deberíamos haber cargado a **tidyverse**: library(tidyverse).

```
rec1 ← haven::read_sav("data/RECH1.sav")
glimpse(rec1)

## # Rows: 139,653
## # Columns: 36
## $ ID1      <dbl> 2020, 2020, 2020, 2020, 202
## $ HHID     <chr> "000101101", "0
## $ HVIDX    <dbl> 1, 1, 2, 3, 1, 2, 3, 4, 1,
## $ HV101    <dbl+lbl> 1, 1, 2, 3, 1, 2, 3, 3,
## $ HV102    <dbl+lbl> 1, 1, 1, 1, 1, 1, 1, 1,
## $ HV103    <dbl+lbl> 1, 1, 1, 1, 1, 1, 1, 1,
```

```
rec94 ← haven::read_sav("data/REC94.sav")
glimpse(rec94)

## # Rows: 17,242
## # Columns: 61
## $ ID1      <dbl> 2020, 2020, 2020, 2020, 2
## $ CASEID   <chr> "000102401 2", "
## $ IDX94    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2
## $ S410B    <dbl+lbl> 9, 9, 8, 9, 9,
## $ S411B    <dbl+lbl> NA, NA, NA, NA, NA, N
## $ S411F    <dbl+lbl> NA, NA, NA, NA, NA, N
## $ S411G    <dbl+lbl> NA, NA, NA, NA, NA, N
```



Previos a inner_join()

Reconocimiento de BD

Inspección Rec1

Inspección Rec94

Preparativo a joins

Hay diversas maneras de explorar los labels de una data .sav (SPSS) o .dta (Stata), como con `labelled::generate_dictionary(rec1)` o `labelled::var_label(rec1, unlist = TRUE) %>% enframe()`. En esta ocasión usaremos al paquete `sjPlot` (no olvides instalarlo).

```
sjPlot::view_df(rec1, show.type = TRUE, show.na = TRUE)
```

```
## HV118 [21]
```

Data frame: rec1

ID	Name	Type	Label	missings	Values	Value Labels
1	ID1	numeric	Año	0 (0.00%)	range:	2020-2020
2	HHID	character	Identificación Cuestionario del Hogar	0 (0.00%)	<output omitted>	
3	HVIDX	numeric	Número de orden	0	range:	1-24



Previos a inner_join()

Reconocimiento de BD

Inspección Rec1

Inspección Rec94

Preparativo a joins

El paquete `labelled` hace algo bastante similar, sin un output *html*/que es lo que le da los colores y el estilo.

```
labelled::generate_dictionary(rec94)
```

```
## pos variable label                                col_type values
## 1   ID1     Año                                 dbl
## 2   CASEID  Identificación Cuestionario Individual chr
## 3   IDX94   Orden de historia de nacimiento      dbl
## 4   S410B   Cuantos meses de embarazo tenía en la última ~ dbl+lbl [98] No sabe
## 5   S411B   En alguno de sus controles: Le midieron la ba~ dbl+lbl [0] No
##                           [1] Si
##                           [8] No sabe
## 6   S411F   En alguno de sus controles: Escucharon los la~ dbl+lbl [0] No
##                           [1] Si
##                           [81] No sabe
```



Previos a inner_join()

Reconocimiento de BD

Inspección Rec1

Inspección Rec94

Preparativo a joins

En las base de datos ENDES hay 2 variables **keys** que se deben tener en consideración: ID de la persona (**CASEID**) y el ID del hogar (**HHID**). Sin embargo, muchas veces esto varía.

En el caso de **rec1** no existe la variable **CASEID**, debido a que ha sido nombrada como **HVIDX**. Y en el caso de **rec94** existe la variable **CASEID**, solo que esta contiene a su vez tanto el identificador de personas como el de hogares... así que necesitamos separarlos.

```
rec94 ← rec94 %>%
  mutate(
    HHID = str_sub(CASEID, 1, str_length(CASEID) - 2),
    CASEID = str_sub(CASEID, -2, -1),
    CASEID = as.numeric(CASEID),
    .after = ID1
  )
```

```
## # A tibble: 17,242 × 62
##       ID1   HHID      CASEID  IDX94  S410B
##       <dbl>  <chr>     <dbl>  <dbl>  <dbl> 
## 1 2020 "0001..." 2 1 9 N
## 2 2020 "0001..." 2 1 9 N
```



Uso de inner_join()

Introducción

Aunque ya tenemos en `rec1` y `rec94` las variables ID's de casa y persona identificadas y separadas, aún tenemos una situación pendiente: el ID de la persona tiene nombres diferentes entre las 2 base de datos.

Para solucionarlo tenemos 2 alternativas:

1. Renombrar la variable de alguna de las base de datos para que de esta manera quede de forma idéntica. Por ej: `rec1 %>% rename(CASEID = HVIDX)`.
2. Usar el argumento `by = c("HHID", "HVIDX" = "CASEID")` dentro de la función `_join()` que se vaya a utilizar



Uso de inner_join()

Solución 1

En esta primera solución usaremos la función `rename` para seguir con el enfoque *tidyverse*.

```
rec1 %>%  
  rename(CASEID = HVIDX) %>%  
  inner_join(  
    rec94,  
    by = c("HHID", "CASEID")  
)
```

```
## # A tibble: 17,242 × 96  
##   ID1.x HHID      CASEID   HV101   HV102   HV103   HV104   HV105   HV106   HV107   HV108   HV109  
##   <dbl> <chr>     <dbl> <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl> <dbl+lbl> <dbl> <dbl+lbl> <dbl> <dbl> <dbl+lbl>  
## 1 2020 "00..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 36 3 [Sup... 3 14 5 [Sup...  
## 2 2020 "00..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 32 3 [Sup... 3 14 5 [Sup...  
## 3 2020 "00..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 27 2 [Sec... 5 11 4 [Sec...  
## 4 2020 "00..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 35 3 [Sup... 5 16 5 [Sup...  
## 5 2020 "00..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 35 2 [Sec... 5 11 4 [Sec...
```



Uso de inner_join()

Solución 2

Esta solución es mucho más directa y requiere menos código.

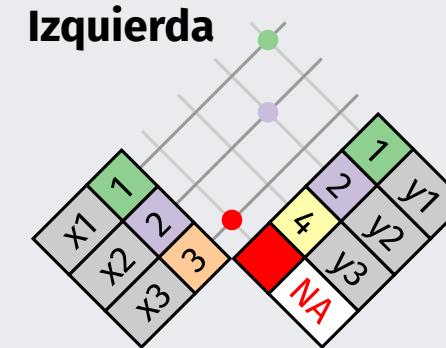
```
rec1 %>%  
  inner_join(  
    rec94,  
    by = c("HHID", "HVIDX" = "CASEID")  
)
```

```
## # A tibble: 17,242 × 96  
##   ID1.x HHID          HVIDX   HV101   HV102   HV103   HV104   HV105   HV106   HV107   HV108   HV109  
##   <dbl> <chr>        <dbl> <dbl+> <dbl+> <dbl+> <dbl+> <dbl> <dbl+> <dbl> <dbl> <dbl> <dbl+>  
## 1 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 36 3 [Sup... 3 14 5 [Sup...  
## 2 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 32 3 [Sup... 3 14 5 [Sup...  
## 3 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 27 2 [Sec... 5 11 4 [Sec...  
## 4 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 35 3 [Sup... 5 16 5 [Sup...  
## 5 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 35 2 [Sec... 5 11 4 [Sec...  
## 6 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 30 3 [Sup... 3 14 5 [Sup...
```

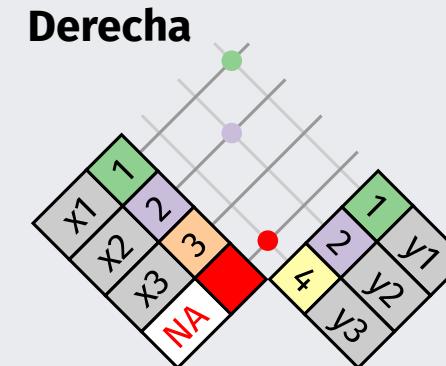
Función `left_join()` y `right_join()`

Esta función se queda con absolutamente todos los casos que estén en el lado solicitado. Los casos en los que no encuentre información, serán completados con NA.

Si se usa `left_join(df1, df2)` se quedarán todos los casos de `df1`. Mientras que si se usa `right_join(df1, df2)`, se usarán todos los casos de `df2`.



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



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

Representación de `left_join()` y `right_join()`
de R para Ciencia de Datos



Uso de left_join()

Nombraremos primero a `recl` que tiene 139653 filas, frente a las 17242 filas en `rec94`.

```
recl %>%  
  left_join(  
    rec94,  
    by = c("HHID", "HVIDX" = "CASEID")  
)
```

```
## # A tibble: 141,773 × 96  
##   ID1.x HHID      HVIDX   HV101   HV102   HV103   HV104   HV105   HV106   HV107   HV108   HV109  
##   <dbl> <chr>     <dbl> <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl> <dbl+lbl> <dbl> <dbl+lbl> <dbl> <dbl> <dbl+lbl>  
## 1 2020 "000..." 1 1 [Jef... 1 [Sí] 1 [Sí] 1 [Hom... 30 2 [Sec... 5 11 4 [Sec...  
## 2 2020 "000..." 1 1 [Jef... 1 [Sí] 1 [Sí] 1 [Hom... 26 3 [Sup... 3 14 5 [Sup...  
## 3 2020 "000..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 25 3 [Sup... 1 12 5 [Sup...  
## 4 2020 "000..." 3 3 [Hij... 1 [Sí] 1 [Sí] 1 [Hom... 5 0 [Sin... NA 0 0 [Sin...  
## 5 2020 "000..." 1 1 [Jef... 1 [Sí] 1 [Sí] 1 [Hom... 52 3 [Sup... 2 18 5 [Sup...  
## 6 2020 "000..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 36 3 [Sup... 3 14 5 [Sup...  
## 7 2020 "000..." 3 3 [Hij... 1 [Sí] 1 [Sí] 1 [Hom... 7 1 [Pri... 1 1 1 [Pri...  
## 8 2020 "000..." 4 3 [Hij... 1 [Sí] 1 [Sí] 1 [Hom... 4 0 [Sin... NA 0 0 [Sin...  
## 9 2020 "000..." 1 1 [Jef... 1 [Sí] 1 [Sí] 1 [Hom... 65 2 [Sec... 5 11 4 [Sec...  
## 10 2020 "000..." 2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 48 3 [Sup... 5 16 5 [Sup...
```



Uso de right_join()

Seguiremos el mismo orden de nombramiento, debido a que estamos usando la función `right_join`, el output de esta unión debería tener un total de 17242 filas.

```
rec1 %>%  
  right_join(  
    rec94,  
    by = c("HHID", "HVIDX" = "CASEID")  
  )
```

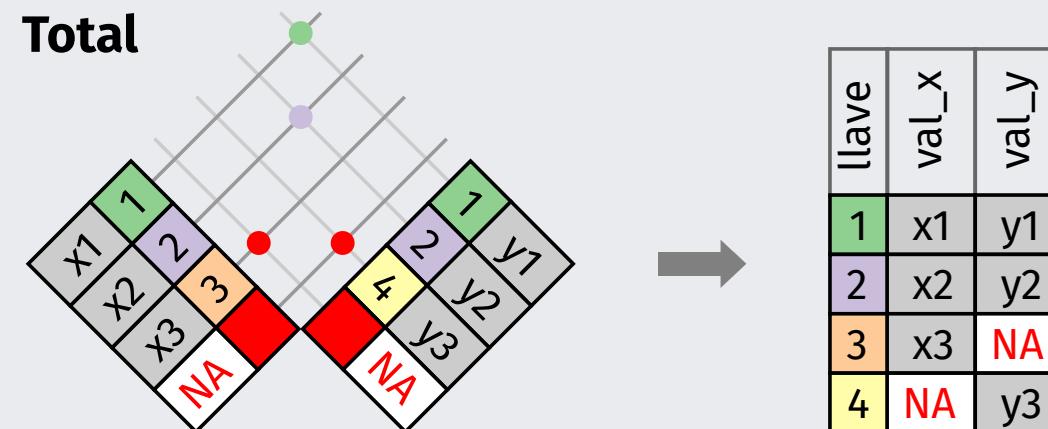
```
## # A tibble: 17,242 × 96  
##   ID1.x HHID          HVIDX   HV101   HV102   HV103   HV104   HV105   HV106   HV107   HV108   HV109  
##   <dbl> <chr>        <dbl> <dbl+lbl> <dbl+lbl> <dbl+lbl> <dbl> <dbl+lbl> <dbl> <dbl> <dbl> <dbl+lbl>  
## 1 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 36 3 [Sup... 3 14 5 [Sup...  
## 2 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 32 3 [Sup... 3 14 5 [Sup...  
## 3 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 27 2 [Sec... 5 11 4 [Sec...  
## 4 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 35 3 [Sup... 5 16 5 [Sup...  
## 5 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 35 2 [Sec... 5 11 4 [Sec...  
## 6 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 30 3 [Sup... 3 14 5 [Sup...  
## 7 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 26 3 [Sup... 2 13 5 [Sup...  
## 8 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 34 3 [Sup... 3 14 5 [Sup...  
## 9 2020 "000..."     2 2 [Esp... 1 [Sí] 1 [Sí] 2 [Muj... 34 3 [Sup... 3 14 5 [Sup...
```



Función full_join()

Esta función se queda con absolutamente todos los casos de ambas base de datos. Los que no encuentren en la primera base de datos, serán llenados con `NA` y los que no encuentren en la segunda base de datos, de igual manera.

Aquí no importa el orden en que pongas las bases de datos, salvo el orden en que quieras que aparezcan las columnas.



Representación de `full_join()` de R para Ciencia de Datos



Uso de full_join()

Invertiremos el orden de nombramiento, simplemente para apreciar otro enfoque. Todos los casos se mantienen en ambas base de datos.

```
rec94 %>%  
  full_join(  
    rec1,  
    by = c("HHID", "CASEID" = "HVIDX")  
  )
```

```
## # A tibble: 141,773 × 96  
##   ID1.x HHID CASEID IDX94 S410B   S411B   S411F   S411G   S411H   S411I   S411J   S411K  
##   <dbl> <chr>  <dbl> <dbl> <dbl>+<l> <dbl>+<l> <dbl>+<l> <dbl>+<l> <dbl>+<l>  
## 1 2020   "     ...    2     1     9 NA      NA      NA      NA      NA      NA      NA  
## 2 2020   "     ...    2     1     9 NA      NA      NA      NA      NA      NA      NA  
## 3 2020   "     ...    2     1     8 NA      NA      NA      NA      NA      NA      NA  
## 4 2020   "     ...    2     1     9 NA      NA      NA      NA      NA      NA      NA  
## 5 2020   "     ...    2     1     9 NA      NA      NA      NA      NA      NA      NA  
## 6 2020   "     ...    2     1     9 NA      NA      NA      NA      NA      NA      NA  
## 7 2020   "     ...    2     1     8 NA      NA      NA      NA      NA      NA      NA  
## 8 2020   "     ...    2     1     7 1 [Si]  1 [Si]  1 [Si]  1 [Si]  1 [Si]  1 [Si]  
## 9 2020   "     ...    2     2     NA NA      NA      NA      NA      NA      NA      NA
```



¡Hazlo tú mismo!

1. Replica los procedimientos observados en las diapositivas
2. Cambia el orden de las base de datos
3. Selecciona solo las primeras variables de ambas bases de datos para que puedas apreciar realmente la unión entre ellas.

08 : 00



Función `rowwise()` y `c_across()`

- Permite realizar operaciones por filas.
- Especialmente útil cuando no existe una función vectorizada para dicha operación: `rowSums()` por ejemplo.
- Requiere indicar la(s) variable(s) sobre las cuales se ejecutará una operación en filas: `c_across()`
- La práctica más común es realizar una sumatoria de variables por filas (`raw scores` por ejemplo), pero también puede usarse con cualquier otra operación.

```
rec94 %>%  
  select(S427DA:S427DG) %>%  
  sjPlot::view_df(rec94, show.type = TRUE, s
```

Data frame: .

<i>ID</i>	<i>Name</i>	<i>Type</i>	<i>Label</i>	<i>missi</i>
1	S427DA	numeric	Complicaciones después del parto: sangrado intenso	2120 (12.3)
2	S427DB	numeric	Complicaciones después del parto: pérdida	2120 (12.3)



Uso de `rowwise()`

Si quisieramos contabilizar directamente cuantas complicaciones en el embarazo suman por cada mujer entrevistada:



Uso de rowwise()

Si quisieramos contabilizar directamente cuantas complicaciones en el embarazo suman por cada mujer entrevistada:

```
rec94 %>%  
  drop_na(S427DA:S427DG) %>%  
  slice(1:1000)
```

A tibble: 1,000 × 62

... with 990 more rows, and 50 more variables: S411L <



Uso de rowwise()

Si quisieramos contabilizar directamente cuantas complicaciones en el embarazo suman por cada mujer entrevistada:

```
rec94 %>%  
  drop_na(S427DA:S427DG) %>%  
  slice(1:1000) %>%  
  rowwise()  
  
## # A tibble: 1,000 × 62  
## # Rowwise:  
##   ID1 HHID CASEID IDX94 S410B S411B S411F S  
##   <dbl> <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <db  
## 1 2020 "     ...    2     1     9 NA     NA     NA  
## 2 2020 "     ...    2     1     9 NA     NA     NA  
## 3 2020 "     ...    2     1     8 NA     NA     NA  
## 4 2020 "     ...    2     1     9 NA     NA     NA  
## 5 2020 "     ...    2     1     9 NA     NA     NA  
## 6 2020 "     ...    2     1     9 NA     NA     NA  
## 7 2020 "     ...    2     1     8 NA     NA     NA  
## 8 2020 "     ...    2     1     7 1 [Si]  1 [Si]  1  
## 9 2020 "     ...    2     1     9 1 [Si]  1 [Si]  1  
## 10 2020 "    ...    2     1     8 1 [Si]  1 [Si]  1
```



Uso de rowwise()

Si quisieramos contabilizar directamente cuantas complicaciones en el embarazo suman por cada mujer entrevistada:

```
rec94 %>%  
  drop_na(S427DA:S427DG) %>%  
  slice(1:1000) %>%  
  rowwise() %>%  
  mutate(  
    complicaciones_tot = sum(c_across(.  
      na.rm = TRUE), .by_group = TRUE)  
  )
```

```
## # A tibble: 1,000 × 63  
## # Rowwise:  
##       ID1 HHID   CASEID IDX94 S410B S411B S411F S  
##     <dbl> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <db  
## 1 2020 " " 2 1 9 NA NA NA  
## 2 2020 " " 2 1 9 NA NA NA  
## 3 2020 " " 2 1 8 NA NA NA  
## 4 2020 " " 2 1 9 NA NA NA  
## 5 2020 " " 2 1 9 NA NA NA  
## 6 2020 " " 2 1 9 NA NA NA  
## 7 2020 " " 2 1 8 NA NA NA  
## 8 2020 " " 2 1 7 1 [Si] 1 [Si] 1  
## 9 2020 " " 2 1 9 1 [Si] 1 [Si] 1  
## 10 2020 " " 2 1 8 1 [Si] 1 [Si] 1
```



Uso de rowwise()

Si quisieramos contabilizar directamente cuantas complicaciones en el embarazo suman por cada mujer entrevistada:



Uso de rowwise()

Si quisieramos contabilizar directamente cuantas complicaciones en el embarazo suman por cada mujer entrevistada:

```
rec94 %>%
  drop_na(S427DA:S427DG) %>%
  slice(1:1000) %>%
  rowwise() %>%
  mutate(
    complicaciones_tot = sum(c_across(
      na.rm = TRUE),
    .by_group = TRUE)
  ) %>%
  ungroup() %>%
  select(ID1:CASEID, complicaciones_tot)
```

ID1	HHID	CASEID	complicaciones_tot
1	"000102401"	2	0
2	"000117001"	2	0
3	"000119801"	2	1
4	"000123601"	2	0
5	"000123901"	2	0
6	"000206801"	2	1
7	"000208401"	2	0
8	"000303001"	2	0
9	"000312401"	2	2
10	"000322101"	2	0
## # ... with 990 more rows			



Paquete lubridate

- Lubridate es una libreria que facilita el trabajar con fechas y horas.
- Contiene tres tipos de objetos
 - <date>
 - <time>
 - <dttm>

```
library(lubridate)
today()
```

```
## [1] "2022-06-04"
```

```
now()
```

```
## [1] "2022-06-04 05:38:24 -05"
```

Permite procesar diferentes formatos de fechas

```
ymd("20110604")
```

```
## [1] "2011-06-04"
```

```
mdy("06-04-2011")
```

```
## [1] "2011-06-04"
```

```
dmy("04/06/2011")
```

```
## [1] "2011-06-04"
```

Y permite procesar formatos de fechas y horas

```
ymd_hms("2011-06-04 12:00:00", tz = "America/Bogota")
```



¡Hazlo tú mismo!

1. Replica los procedimientos observados en `rowwise()` y `c_across()`
2. Crea un vector con la fecha de tu cumpleaños
3. Usando la función `today()`, ¡calcula la cantidad de días que llevas vivo! (solo resta)
4. Explora la función `time_length(x, unit = "years")` para la resta anterior

08:00



Retroalimentación



¡Gracias!

✉ imt.innovlab@oficinas-upch.pe

🗣 [@healthinnovation](https://twitter.com/@healthinnovation)

🐦 [@innovalab_imt](https://twitter.com/@innovalab_imt)

Estas diapositivas fueron creadas mediante el paquete `xaringan` y `xaringanthemer`.