Introducción a R y Tidyverse II

Sesión 02

Laboratorio de Innovación en Salud

2022-09-29

@healthinnovation
@innovalab_imt
innovalab.info

Contenidos

InnovaLab

- Solicitar resumen de datos (summarise)
- Considerar variables para subdividir análisis (group_by)
- Manejo de múltiples conjuntos de datos mediante funciones join.

Exploración competencial

Uso de group_by( )

Esta función permite poder indicarle a R que las operaciones que hagamos en adelante, sean divididas y aplicadas por las agrupaciones que sea haya formado. Es decir, que se puede hacer la agrupación por una o múltiples variables.

```
library(tidyverse)
storms %>%
  filter(year ⩾ 2010) %>%
  group_by(name)
```

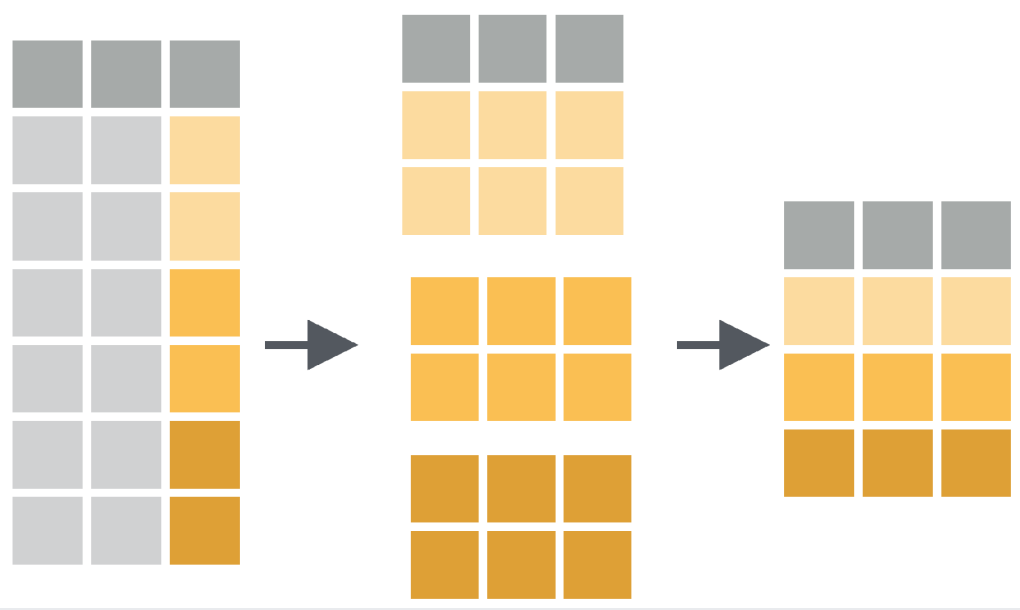
```
## # A tibble: 3,608 × 13
## # Groups:   name [112]
##    name   year month   day  hour   lat
##    <chr> <dbl> <dbl> <int> <dbl> <dbl>
##  1 Alex   2010     6    25    18  16.4
##  2 Alex   2010     6    26     0  16.6
##  3 Alex   2010     6    26     6  16.7
##  4 Alex   2010     6    26    12  16.9
```

La función `group_by( )` solo configura como será el tratamiento de la data en adelante. No observará ningún cambio percepctible en los datos con excepción de:

```
# Groups:   name [112]
```

Uso de `summarise()`

Esta función permite obtener algun resumen de los datos, como la media, desviación estándar, valores máximos, mínimos, calculos de prueba estadística, y la mayoría de funciones de esta naturaleza. El uso conjunto de `group_by()` y `summarise()` es una combinación perfecta.



Data transformation with dplyr:: CHEAT SHEET

```
mtcars %>%
  group_by(cyl) %>%
  summarise(
    avg = mean(mpg)
  )
```

Uso de summarise()

Guardaremos la data pivoteada en un objeto llamado covid19_tidy, para no extender demasiado el código. Además, veamos lo sencillo que es poder introducir un filter a los resultados obtenidos.

```
covid19_tidy %>%
  mutate(
    fecha = lubridate::mdy(fecha)
  ) %>%
  filter(fecha > "2022-01-01") %>%
  group_by(`Country/Region`) %>%
  summarise(
    M = mean(fallecidos, na.rm = TRUE
    DE = sd(fallecidos, na.rm = TRUE)
    Max = max(fallecidos, na.rm = TRU
    Min = min(fallecidos, na.rm = TRU
  ) %>%
  arrange(desc(M))
```

```
## # A tibble: 201 × 5
##    `Country/Region`       M     DE     Max    Min
##    <chr>              <dbl>  <dbl>   <dbl>  <dbl>
##  1 US                986126. 58749. 1058506 826411
##  2 Brazil            661839. 19116.  685927 619401
##  3 India             518425. 12565.  528611 481893
##  4 Russia            360156. 20591.  379227 304284
##  5 Mexico            321724.  8447.  330046 299544
##  6 Peru              212012.  3584.  216526 202782
##  7 Italy             162831. 10626.  177024 137646
##  8 Indonesia         153762.  4918.  158076 144097
##  9 Iran              139749.  3807.  144416 131680
## 10 Colombia          138987.  2979.  141769 130026
## # … with 191 more rows
```

¡Hazlo tú mismo!

InnovaLab

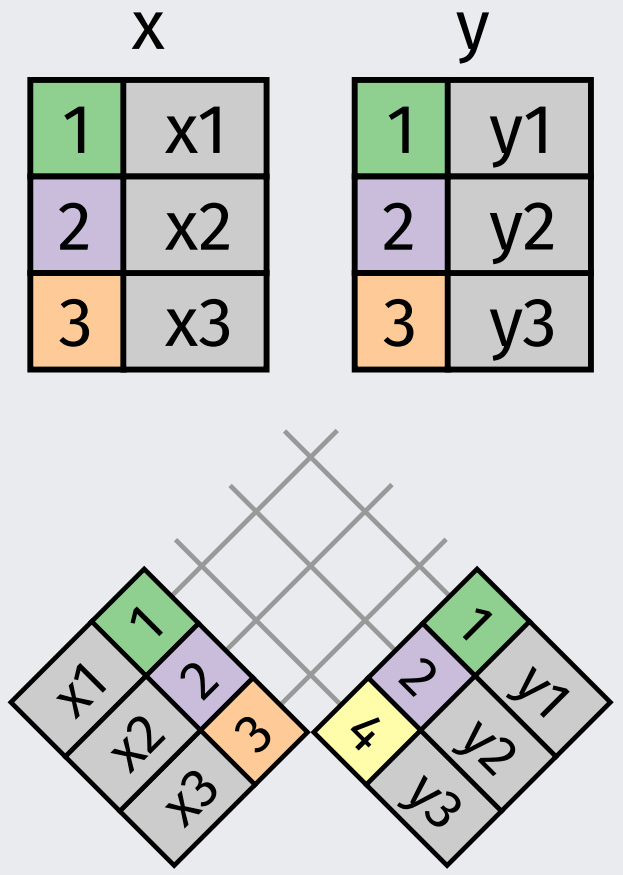
1. Replica los procedimientos observados en las diapositivas
2. Genera nuevas solicitudes en summarise que consideres pertinente.
3. Usa nuevas reglas de filtrado

08:00

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 coincida.

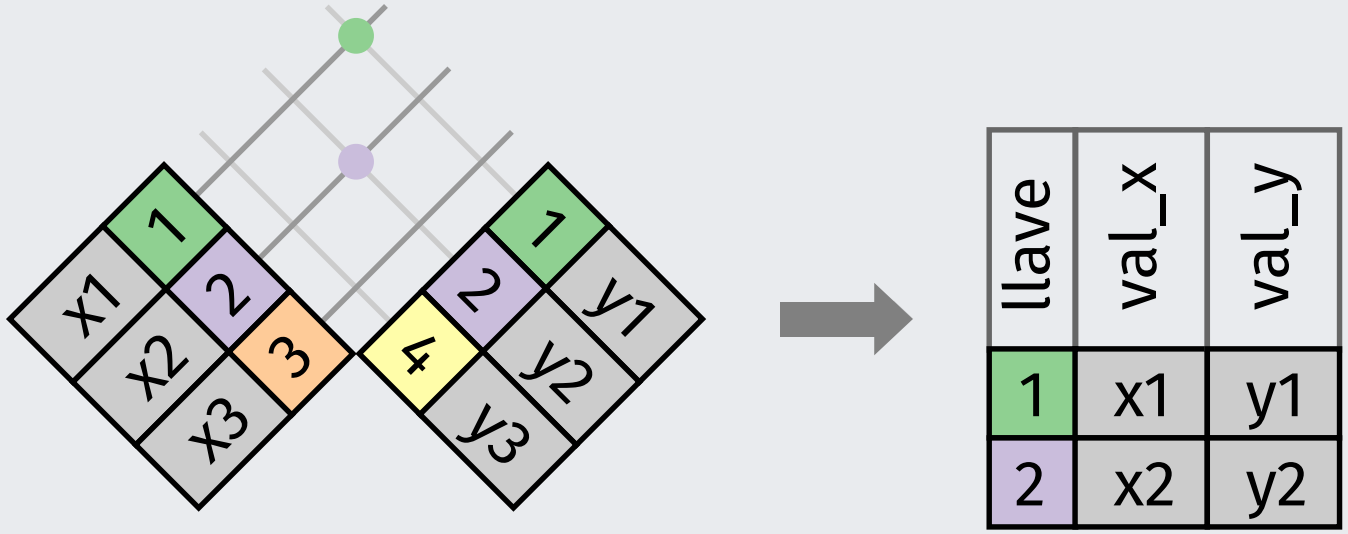


Representación de operaciones joins R para Ciencia de Datos

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.



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

Previos a `inner_join()`

InnovaLab

Reconocimiento de BD | Inspección Recl | 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, 2020.
## $ HHID   <chr> "      000101101", "      00.
## $ HVIDX  <dbl> 1, 1, 2, 3, 1, 2, 3, 4, 1, 2.
## $ 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, .
## $ HV104  <dbl+lbl> 1, 1, 2, 1, 1, 2, 1, 1, .
```

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

```
## Rows: 17,242
## Columns: 61
## $ ID1      <dbl> 2020, 2020, 2020, 2020, 20.
## $ CASEID   <chr> "      000102401  2", "    .
## $ IDX94    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2,.
## $ S410B    <dbl+lbl>  9,  9,  8,  9,  9,  9.
## $ S411B    <dbl+lbl> NA, NA, NA, NA, NA, NA.
## $ S411F    <dbl+lbl> NA, NA, NA, NA, NA, NA.
## $ S411G    <dbl+lbl> NA, NA, NA, NA, NA, NA.
```

Uso de `inner_join()`

InnovaLab

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+l> <dbl+> <dbl+> <dbl+l> <dbl> <dbl+l> <dbl> <dbl> <dbl+l>
##  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.

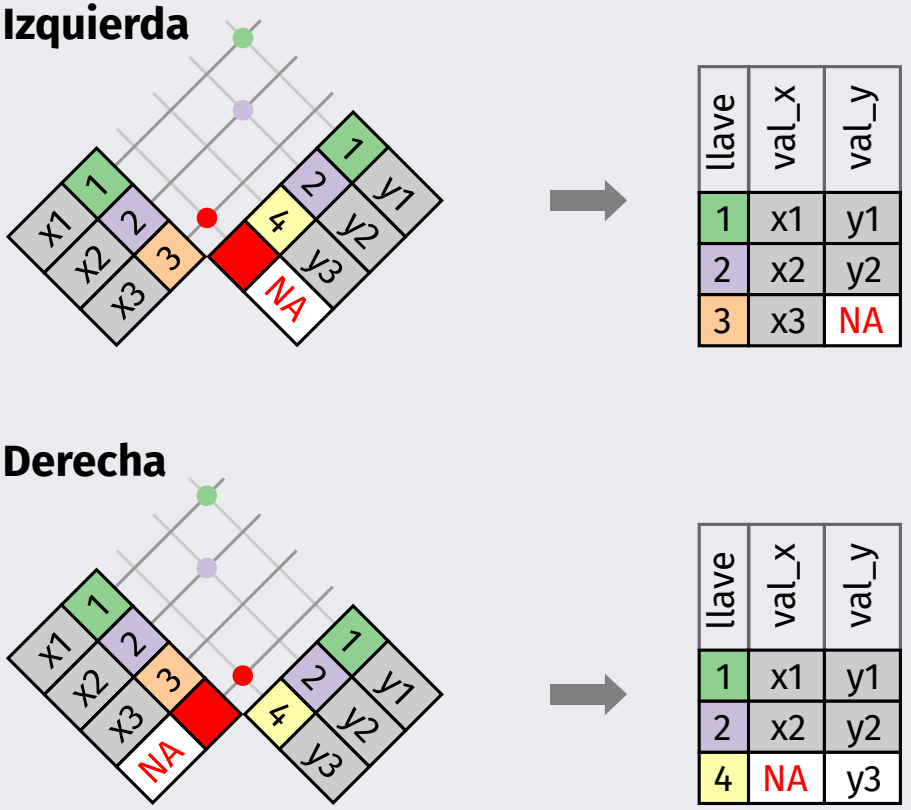
```
rec1 %>%
  inner_join(
    rec94,
    by = c("HHID", "HVIDX" = "CASEID")
  )
```

```
## # A tibble: 17,242 × 96
##    ID1.x HHID        HVIDX HV101   HV102  HV103  HV104   HV105 HV106   HV107 HV108 HV109
##    <dbl> <chr>       <dbl> <dbl+l> <dbl+> <dbl+> <dbl+l> <dbl> <dbl+l> <dbl> <dbl> <dbl+l>
##  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`.



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`.

```
rec1 %>%
  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+l> <dbl+> <dbl+> <dbl+l> <dbl> <dbl+l> <dbl> <dbl> <dbl+l>
##  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.

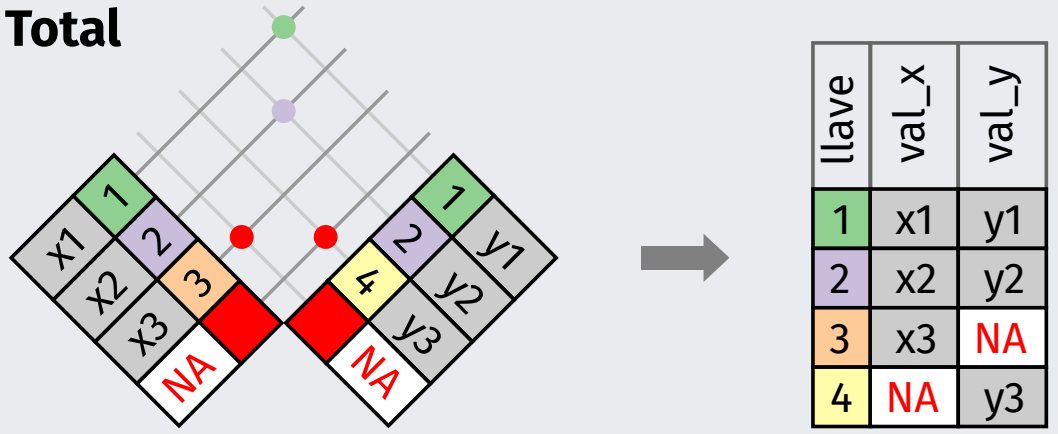
```
rec1 %>%
  right_join(
    rec94,
    by = c("HHID", "HVIDX" = "CASEID")
  )
```

```
## # A tibble: 17,242 × 96
##    ID1.x HHID        HVIDX HV101   HV102  HV103  HV104   HV105 HV106   HV107 HV108 HV109
##    <dbl> <chr>       <dbl> <dbl+l> <dbl+> <dbl+> <dbl+l> <dbl> <dbl+l> <dbl> <dbl> <dbl+l>
##  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> <dbl+l> <dbl+l> <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]  1 [Si]
##  9  2020 "    …      2     2 NA    NA      NA      NA      NA      NA      NA      NA
```

¡Hazlo tú mismo!

InnovaLab

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

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 =
  ) %>%
  ungroup() %>%
  select(ID1:CASEID, complicaciones_t
```

```
## # A tibble: 1,000 × 4
##       ID1 HHID                CASEID complicaciones_tot
##     <dbl> <chr>                <dbl>                <dbl>
##  1  2020 "      000102401"       2                    0
##  2  2020 "      000117001"       2                    0
##  3  2020 "      000119801"       2                    1
##  4  2020 "      000123601"       2                    0
##  5  2020 "      000123901"       2                    0
##  6  2020 "      000206801"       2                    1
##  7  2020 "      000208401"       2                    0
##  8  2020 "      000303001"       2                    0
##  9  2020 "      000312401"       2                    2
## 10  2020 "      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] "2022-09-29"
```

```
now()
```

```
## [1] "2022-09-29 09:52:44 -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/E
```

```
## [1] "2011-06-04 12:00:00 -05"
```

¡Hazlo tú mismo!

InnovaLab

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

@innovalab_imt

Estas diapositivas fueron creadas mediante el paquete xaringan y xaringanthemer.