Repaso de R - Tidyverse

Economía Laboral

Miriam Malament
Economía Laboral, UCEMA

Presentación

A lo largo de esta clase, trabajaremos principalmente con el paquete Tidyverse. El mismo agrupa una serie de paquetes que tienen una misma lógica en su diseño y por ende funcionan en armonía.
Entre ellos usaremos principalmente **dplyr** y **tidyr** para realizar transformaciones sobre nuestro set de datos, y **ggplot** para realizar gráficos (éste último se verá en la clase 3).

A continuación cargamos el paquete a nuestro ambiente. Para ello debe estar previamente instalada en nuestra pc. También utilizaremos los paquetes **readxl**, **questionr** y **scales**

```
library(tidyverse)
library(readxl)
library(scales)
library(gtable)
library(questionr)
```

Carga de Informacion

La función **read.table** nos permite levantar los archivos de extensión ".txt"
La función **read_excel** del paquete (readxl) nos permite levantar los archivos de extensión ".xlsx"

Levantamos la base individual del cuarto trimestre de 2021, y un listado que contiene los Nombres y Códigos de los Aglomerados EPH.

```
Individual_t421 <-eph::get_microdata(2021, 4)

Aglom <- read_excel("~/Desktop/LABORAL/Git Hub/Fuentes/Aglomerados EPH.xlsx")
```

Aclaraciones

Para mostrar el funcionamiento básico de tidyverse, operaremos con la base Individual de la EPH seleccionando únicamente una serie de variables de interés:

- código identificador del aglomerado (AGLOMERADO)
- identificador de aglomerados de más de 500.000 habitantes (MAS_500)
- sexo (CH04)
- edad (CH06)
- Ingreso total individual (P47T)
- Ponderador general (PONDERA),
- Ponderador para ingreso total individual (PONDII).

```
Datos  ← Individual_t421[c("AGLOMERADO","MAS_500","CH04","CH06","P47T","PONDERA","PON
```

summary

Resume la información de cada una de las variables de la base de datos, mostrando distintos momentos de la distribución para variables *numéricas*, y un conteo de casos para variables del tipo *factor* o *character*

```
summary(Datos)
```

```
##    AGLOMERADO      MAS_500               CH04           CH06
##  Min.   : 2.00   Length:50154       Min.   :1.00   Min.   : -1.00
##  1st Qu.:10.00   Class :character   1st Qu.:1.00   1st Qu.: 18.00
##  Median :22.00   Mode  :character   Median :2.00   Median : 34.00
##  Mean   :23.51                      Mean   :1.52   Mean   : 36.22
##  3rd Qu.:32.00                      3rd Qu.:2.00   3rd Qu.: 54.00
##  Max.   :93.00                      Max.   :2.00   Max.   :100.00
##
##       P47T             PONDERA            PONDII
##  Min.   :     -9   Min.   :   14.0   Min.   :    0.0
##  1st Qu.:      0   1st Qu.:  157.0   1st Qu.:  136.0
##  Median :  10000   Median :  273.0   Median :  249.0
##  Mean   :  25204   Mean   :  578.3   Mean   :  578.3
##  3rd Qu.:  40000   3rd Qu.:  532.0   3rd Qu.:  501.0
```

sample_n

Esta función está dentro del paquete tidyverse. Nos permite seleccionar de manera muy simple una muestra al azar de *n* casos de nuestra base de datos. Puede ser útil para una primera mirada de la base que trascienda a los primeros casos, o bien para procedimienos de muestreo aleatorio.

```
sample_n(tbl = Datos,size = 9)
```

```
## # A tibble: 9 × 7
##   AGLOMERADO MAS_500  CH04  CH06   P47T PONDERA PONDII
##        <int> <chr>   <int> <int>  <int>   <int>  <int>
## 1         91 N           1    69 100000     176    183
## 2          7 N           1    27 120000     213    327
## 3         91 N           2    36 115000     141    143
## 4         91 N           2    39  58000     187    183
## 5         29 S           1    78  58000     397    404
## 6          8 N           1    51  15500     187    185
## 7         22 N           1    33  50000     192    204
## 8         29 S           1    79  30000     317    322
## 9         22 N           2    18      0     280    280
```

table

Esta función puede utilizarse para hacer tabulados univariados o bivariados en variables discretas.

```
table(Datos$MAS_500,Datos$CH04)
```

```
##
##          1     2
##   N 13199 14101
##   S 10859 11995
```

wtd.table

En este caso el resultado es un conteo *muestral* de los casos de varones (CH04 == 1) y mujeres (CH04 == 2) según sean de aglomerados de más de 500.000 habitantes (MAS_500 == "S") o menos de 500.000 habitantes (MAS_500 == "N").

```
questionr::wtd.table(x = Datos$MAS_500,y = Datos$CH04,weights = Datos$PONDERA)
```

```
##          1        2
## N  2548522  2690137
## S 11558593 12209116
```

Más adelante en el curso veremos una implementación del *paquete eph* que extiende las potencialidades de estas funciones, permitiendo expresar directamente los tabulados en distribuciones porcentuales, agregar totales por filas y columnas, etiquetas, entre otras.

Dplyr

El caracter principal para utilizar este paquete es %>% , *pipe* (de tubería).

Los %>% toman el set de datos a su izquierda, y los transforman mediante los comandos a su derecha, en los cuales los elementos de la izquierda están implícitos. Es decír, que una vez específicado el DataFrame con el cual se trabaja, no será necesario nombrarlo nuevamente para referirse a una determinada variable/columna del mismo.

Veamos las principales funciones que pueden utilizarse con la lógica de este paquete:

Tidyverse

filter

Permite filtrar la tabla acorde al cumplimiento de condiciones lógicas

```
pepito ← Datos %>%
  filter(CH04==1 , CH06 ⩾ 50)
```

Nótese que en este caso al separar con una , las condiciones se exige el cumplimiento de ambas. En caso de desear que se cumpla al menos una condición debe utilizarse el caracter |

```
Datos %>%
    filter(CH04==1| CH06 ⩾ 50)
```

```
## # A tibble: 32,391 × 7
##    AGLOMERADO MAS_500  CH04  CH06   P47T PONDERA PONDII
##         <int> <chr>   <int> <int>  <int>   <int>  <int>
##  1         17 N           1    13      0     309    309
##  2         33 S           1    79  35000    1573   1954
##  3         33 S           2    79  29000    1573   1960
##  4         33 S           1    48     -9    2635      0
##  5          6 N           2    65  26000      87    106
```

rename

Permite renombrar una columna de la tabla. Funciona de la siguiente manera: Data %>% rename(nuevo_nombre = viejo_nombre)

```
Datos ← Datos %>%
  rename(EDAD = CH06)
Datos
```

```
## # A tibble: 50,154 × 7
##    AGLOMERADO MAS_500  CH04  EDAD   P47T PONDERA PONDII
##         <int> <chr>   <int> <int>  <int>   <int>  <int>
##  1         17 N           1    13      0     309    309
##  2         33 S           1    79  35000    1573   1954
##  3         33 S           2    79  29000    1573   1960
##  4         33 S           1    48     -9    2635      0
##  5          6 N           2    65  26000      87    106
##  6         23 S           2    76  80000     314    321
##  7         23 S           1    45 120000     314    324
##  8         17 N           2    61  45000     353    394
##  9         17 N           2    29  75000     353    414
## 10         17 N           1     8      0     353    353
```

mutate

Permite agregar una variable a la tabla (especificando el nombre que tomará esta), que puede ser el resultado de operaciones sobre otras variables de la misma tabla.

Dentro del **mutate** cada "," permitirá crear otra variable.

Tip: En caso de especificar el nombre de una columna existente, el resultado de la operación realizada "sobrescribirá" la información de la columna con dicho nombre

mutate

```
Datos ← Datos %>%
  mutate(Edad_cuadrado=EDAD^2,
         Edad_cubo =EDAD^3)
Datos
```

```
## # A tibble: 50,154 × 9
##    AGLOMERADO MAS_500  CH04  EDAD   P47T PONDERA PONDII Edad_cuadrado Edad_cubo
##         <int> <chr>   <int> <int>  <int>   <int>  <int>         <dbl>     <dbl>
##  1         17 N           1    13      0     309    309           169      2197
##  2         33 S           1    79  35000    1573   1954          6241    493039
##  3         33 S           2    79  29000    1573   1960          6241    493039
##  4         33 S           1    48     -9    2635      0          2304    110592
##  5          6 N           2    65  26000      87    106          4225    274625
##  6         23 S           2    76  80000     314    321          5776    438976
##  7         23 S           1    45 120000     314    324          2025     91125
##  8         17 N           2    61  45000     353    394          3721    226981
##  9         17 N           2    29  75000     353    414           841     24389
## 10         17 N           1     8      0     353    353            64       512
## # … with 50,144 more rows
## # ℹ Use `print(n = ...)` to see more rows
```

case_when

Permite definir una variable, la cual toma un valor particular para cada condición establecida. En caso de no cumplir ninguna de las condiciones establecidas la variable tomara valor **NA**.
Su funcionamiento es el siguiente:
case_when(condicion1 ~ "Valor1",condicion2 ~ "Valor2",condicion3 ~ "Valor3")

case_when

```
Datos ← Datos %>%
  mutate(Grupos_Etarios = case_when(EDAD  < 18   ~ "Menores",
                                    EDAD  %in%  18:65   ~ "Adultos",
                                    EDAD  > 65 ~ "Adultos Mayores"))
Datos
```

```
## # A tibble: 50,154 × 10
##    AGLOMERADO MAS_500  CH04  EDAD   P47T PONDERA PONDII Edad_c…¹ Edad_…² Grupo…³
##         <int> <chr>   <int> <int>  <int>   <int>  <int>    <dbl>   <dbl> <chr>
##  1         17 N           1    13      0     309    309      169    2197 Menores
##  2         33 S           1    79  35000    1573   1954     6241  493039 Adulto…
##  3         33 S           2    79  29000    1573   1960     6241  493039 Adulto…
##  4         33 S           1    48     -9    2635      0     2304  110592 Adultos
##  5          6 N           2    65  26000      87    106     4225  274625 Adultos
##  6         23 S           2    76  80000     314    321     5776  438976 Adulto…
##  7         23 S           1    45 120000     314    324     2025   91125 Adultos
##  8         17 N           2    61  45000     353    394     3721  226981 Adultos
##  9         17 N           2    29  75000     353    414      841   24389 Adultos
## 10         17 N           1     8      0     353    353       64     512 Menores
## # … with 50,144 more rows, and abbreviated variable names ¹Edad_cuadrado,
## #   ²Edad_cubo, ³Grupos_Etarios
```

select

Permite especificar la serie de columnas que se desea conservar de un DataFrame. También pueden especificarse las columnas que se desean descartar (agregándoles un -). Muy útil para agilizar el trabajo en bases de datos de gran tamaño.

```
#Conservo solo 2 variables
Datos %>%
  select(CH04,PONDERA)
```

```
## # A tibble: 50,154 × 2
##     CH04 PONDERA
##    <int>   <int>
##  1     1     309
##  2     1    1573
##  3     2    1573
##  4     1    2635
##  5     2      87
##  6     2     314
##  7     1     314
##  8     2     353
##  9     2     353
```

Tidyverse

arrange

Permite ordenar la tabla por los valores de determinada/s variable/s. Es útil cuando luego deben hacerse otras operaciones que requieran del ordenamiento de la tabla

```
Datos ← Datos %>%
  arrange(CH04,EDAD)
Datos
```

```
## # A tibble: 50,154 × 10
##    AGLOMERADO MAS_500  CH04  EDAD  P47T PONDERA PONDII Edad_cu…¹ Edad_…² Grupo…³
##         <int> <chr>   <int> <int> <int>   <int>  <int>     <dbl>   <dbl> <chr>
##  1         13 S           1    -1     0     246    246         1      -1 Menores
##  2         17 N           1    -1     0     439    439         1      -1 Menores
##  3         23 S           1    -1     0     327    327         1      -1 Menores
##  4         30 N           1    -1     0     214    214         1      -1 Menores
##  5         91 N           1    -1     0     133    133         1      -1 Menores
##  6          2 S           1    -1     0     730    730         1      -1 Menores
##  7         29 S           1    -1     0     423    423         1      -1 Menores
##  8         38 N           1    -1     0     244    244         1      -1 Menores
##  9         33 S           1    -1     0    3563   3563         1      -1 Menores
## 10         33 S           1    -1     0    2577   2577         1      -1 Menores
```

summarise

Crea una nueva tabla que resume la información original. Para ello, definimos las variables de resumen y las formas de agregación. Calculemos por ejemplo la edad promedio de la población de nuestra base.

```
#Recuerden que los menores de un año están clasificados con el valor -1
Datos ← Datos %>%
  mutate(edad.corregida=ifelse(EDAD == -1,yes = 0,no = EDAD))
#R BASE#
mean(Datos$edad.corregida,na.rm = T) #sin ponderar
```

```
## [1] 36.23213
```

```
weighted.mean(Datos$edad.corregida,Datos$PONDERA) #ponderado
```

```
## [1] 34.65037
```

Tidyverse

summarise

```
#Tidyverse
Datos %>%
 summarise(Edad_prom = mean(edad.corregida),  #sin ponderar
           Edad_prom_pond = weighted.mean(x = edad.corregida,w = PONDERA)) #ponderado
```

```
## # A tibble: 1 × 2
##   Edad_prom Edad_prom_pond
##       <dbl>          <dbl>
## 1      36.2           34.7
```

group_by

Esta función permite realizar operaciones de forma agrupada. Lo que hace la función es "separar" a la tabla según los valores de la variable indicada y realizar las operaciones que se especifican a continuación, de manera independiente para cada una de las "subtablas". En nuestro ejemplo, sería útil para calcular el promedio de edad según *sexo*

```
Datos %>%
  group_by(CH04) %>%
  summarise(Edad_Prom = weighted.mean(EDAD,PONDERA))
```

```
## # A tibble: 2 × 2
##    CH04 Edad_Prom
##   <int>     <dbl>
## 1     1      33.2
## 2     2      36.0
```

Tidyverse

group_by

Notese que los %>% pueden usarse encadenados para realizar numerosos procedimientos sobre un dataframe original. Veamos un ejemplo con multiples encadenamietnos

```
Encadenado ← Datos %>%
  filter(Grupos_Etarios == "Adultos") %>%
  mutate(Sexo = case_when(CH04 == 1 ~ "Varon",
                          CH04 == 2 ~ "Mujer")) %>%
  select(-Edad_cuadrado)

Encadenado
```

```
## # A tibble: 31,337 × 11
##    AGLOMERADO MAS_500  CH04  EDAD  P47T PONDERA PONDII Edad_cubo Grupo…¹ edad.…²
##         <int> <chr>   <int> <int> <int>   <int>  <int>      <dbl> <chr>     <dbl>
##  1          2 S           1    18     0     623    623       5832 Adultos      18
##  2         13 S           1    18     0     753    753       5832 Adultos      18
##  3         15 N           1    18 20000     159    173       5832 Adultos      18
##  4         93 N           1    18     0      79     79       5832 Adultos      18
##  5         23 S           1    18 10000     299    297       5832 Adultos      18
##  6         23 S           1    18  3000     201    200       5832 Adultos      18
##  7         13 S           1    18     0     648    648       5832 Adultos      18
```

Joins

left_join

Veamos un ejemplo de la función **left_join** (una de las más utilizadas en la práctica). Para ello utilizaremos el Dataframe *Aglom* con los códigos y los nombres de los aglomerados EPH

```
Aglom
```

```
## # A tibble: 32 × 2
##    AGLOMERADO Nom_Aglo
##         <dbl> <chr>
##  1         32 Ciudad de Bs As
##  2         33 Partidos del GBA
##  3         10 Gran Mendoza
##  4         27 Gran San Juan
##  5         26 San Luis - El Chorrillo
##  6         12 Corrientes
##  7         15 Formosa
##  8          8 Gran Resistencia
```

Tidyr

El paquete tidyr esta pensado para facilitar el emprolijamiento de los datos. Veremos las funciones *pivot_longer* y *pivot_wider* que nos permitirán pivotear la base según como queramos ordenar los datos para trabajarlos posteriormente o bien para presentarlos como tabla de resultados.

pivot_longer

Nos permite pivotear los datos en columnas hacia una única variable. El *"longer"* refiere a que ahora nuestro dataframe va a tener más filas (y menos columnas)

```
pob.aglo.long ← Poblacion_Aglomerados %>%
  pivot_longer(cols = 2:4,names_to = "Grupo_Etario",values_to = "Poblacion")
pob.aglo.long
```

```
## # A tibble: 96 × 3
##    Nom_Aglo                  Grupo_Etario    Poblacion
##    <chr>                     <chr>                <int>
##  1 Bahía Blanca - Cerri      Menores              74628
##  2 Bahía Blanca - Cerri      Adultos             198412
##  3 Bahía Blanca - Cerri      Adultos_Mayores      43825
##  4 Cdro. Rivadavia - R.Tilly Menores              72936
##  5 Cdro. Rivadavia - R.Tilly Adultos             151425
##  6 Cdro. Rivadavia - R.Tilly Adultos_Mayores      18386
##  7 Ciudad de Bs As           Menores             677425
##  8 Ciudad de Bs As           Adultos            1852853
##  9 Ciudad de Bs As           Adultos_Mayores     472457
## 10 Concordia                 Menores              50153
```

pivot_wider

Es el opuesto de **pivot_longer**. Nos permite pivotear los datos de una variable para obtener múltiples columnas en función de alguna variable categórica que clasifique a la anterior. El "wider" refiere a que ahora nuestro dataframe va a tener menos filas (y más columnas, en función de la cantidad de categorías de la variable que use para pivotear)

```
pob.aglo.long %>%
  pivot_wider(names_from = "Grupo_Etario",values_from = "Poblacion")
```

```
## # A tibble: 32 × 4
##    Nom_Aglo                  Menores Adultos Adultos_Mayores
##    <chr>                       <int>   <int>           <int>
##  1 Bahía Blanca - Cerri        74628  198412           43825
##  2 Cdro. Rivadavia - R.Tilly   72936  151425           18386
##  3 Ciudad de Bs As            677425 1852853          472457
##  4 Concordia                   50153   97979           14287
##  5 Corrientes                 101317  248876           33829
##  6 Formosa                     74951  158792           22649
##  7 Gran Catamarca              65097  141578           19634
##  8 Gran Córdoba               414967  997254          164549
##  9 Gran La Plata              240191  562440          106112
```

Tasas del Mercado de Trabajo

Creamos una tabla con los niveles de:

- Población
- Ocupados

```
Poblacion_ocupados ← Individual_t421 %>%
  summarise(Poblacion         = sum(PONDERA),
            Ocupados          = sum(PONDERA[ESTADO == 1]))
Poblacion_ocupados
```

```
## # A tibble: 1 × 2
##   Poblacion Ocupados
##       <int>    <int>
## 1  29006368 12643582
```

Tasas del Mercado de Trabajo

- Población: Si contaramos cuantos registros tiene la base, simplemente tendríamos el numero de individuos muestral de la EPH, por ende debemos **sumar los valores de la variable PONDERA**, para contemplar a cuantas personas representa cada individuo encuestado.
- Ocupados: En este caso, debemos agregar un **filtro** al procedimiento anterior, ya que unicamente queremos sumar los ponderadores de aquellas personas que se encuentran ocupadas. (La lógica seria: "Suma los valores de la columna PONDERA, solo para aquellos registros donde el ESTADO == 1")

Tasas del Mercado de Trabajo

La función summarise() nos permite crear multiples variables de resumen al mismo tiempo, simplemente separando con una "," cada uno de ellas. A su vez, se pueden crear variables, a partir de las variables creadas por la propia función. De esta forma, podemos, directamente calcular la **tasa de empleo** a partir del total poblacional y de ocupados.

```
Empleo ← Individual_t421 %>%
  summarise(Poblacion         = sum(PONDERA),
            Ocupados          = sum(PONDERA[ESTADO == 1]),
            Tasa_Empleo    = Ocupados/Poblacion)
Empleo
```

```
## # A tibble: 1 × 3
##   Poblacion Ocupados Tasa_Empleo
##       <int>    <int>       <dbl>
## 1  29006368 12643582       0.436
```

Tasas del Mercado de Trabajo

En caso de querer expresar los resultados como porcentajes, podemos utilizar la función **percent** del paquete *scales*. Para ello debemos utilizar **mutate** para transformar la variable Tasa_Empleo

Nótese que en este caso, para poder añadir el %, la función transforma a la variable en un Character, por ende debe tenerse en cuenta que se pierde la información del numero completo. Esto es simplemente una herramienta para visualizar la información final en terminos de %, no utilizarla si se va a seguir operando con dicho valor.

```
Empleo %>%
  mutate(Tasa_Empleo_Porc = scales::percent(Tasa_Empleo))
```

```
## # A tibble: 1 × 4
##   Poblacion Ocupados Tasa_Empleo Tasa_Empleo_Porc
##       <int>    <int>       <dbl> <chr>
## 1  29006368 12643582       0.436 44%
```

Gtable

- También trabajaremos con **gtable**.
- De la misma manera que "ggplot" se refiere a "grammar of plots", "gtable" se refiere a "grammar of tables".

A modo de ejemplo:

```
library(gt)

Empleo %>%
  mutate(Tasa_Empleo_Porc = scales::percent(Tasa_Empleo)) %>% gt()
```

Poblacion	Ocupados	Tasa_Empleo	Tasa_Empleo_Porc
29006368	12643582	0.4358899	44%

Gtable

Lo podemos mejorar un poco:

```
Empleo %>%
  mutate(Tasa_Empleo_Porc = scales::percent(Tasa_Empleo)) %>% gt() %>%
  tab_header(
    title = md("**Tasa de Empleo**"),
    subtitle = md("Poblacion, Ocupados y Tasa de Empleo")
  ) %>%
  cols_align(
    align = c("center"),
    columns = everything()) %>%
  cols_label("Tasa_Empleo"=md("Tasa de Empleo"),
             "Tasa_Empleo_Porc" = md("Tasa de Empleo (Porcentaje)")) %>%
  tab_source_note(md("Datos correspondientes al cuatro trimestre del 2021"))
```

Tasa de Empleo

Poblacion, Ocupados y Tasa de Empleo

Poblacion	Ocupados	Tasa de Empleo	Tasa de Empleo (Porcentaje)
29006368	12643582	0.4358899	44%

Datos correspondientes al cuatro trimestre del 2021

Exportar resultados a Excel

La función **write.xlsx** de la libreria openxlsx nos permite exportar un dataframe a un archivo de Excel. Hay una infinidad de formas para hacer la exportación, agregando títulos a los cuadros, múltiples dataframes en una misma solapa del excel, o creando múltiples solapas en el excel. Todas estas variantes, con ejemplos, se pueden consultar acá.

La forma más simple, sólo requiere indicar el dataframe a exportar, y la ruta completa, con extensión del archivo incluido

```
dir.create("Resultados")
openxlsx::write.xlsx(x = Empleo,file =  "Resultados/miexportacion.xlsx")
```