

Tutoría 4

Repaso de tidyrr y algunas claves para la **programación funcional**

Sofía Madariaga

Pontificia Universidad Católica de Chile
Diplomado Ciencia de Datos para Políticas Públicas
Taller de Análisis de Datos I

6 de septiembre de 2023



Materiales

En el portal del diplomado [Materiales del Curso](#) > [Tutorías](#).

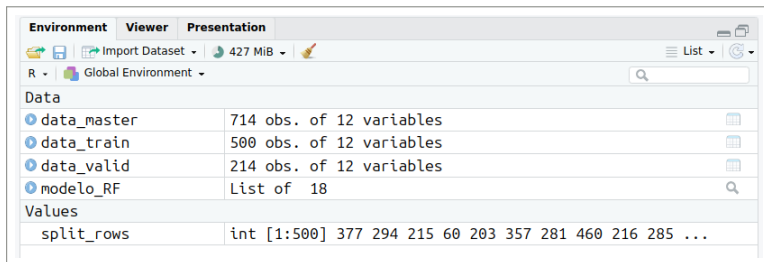
- input: carpeta con materiales relativo a datos e insumos.
- tutorial-4.r ← [aquí estaremos trabajando](#)
- tutorial-4_presentacion.r
- tutorial-4.Rproj
- environment.RData

Objetivos de la Tutoría

- 1 Terminar de pasar los contenidos de tidyrr.
- 2 Presentar algunas claves y aplicación básica sobre programación funcional.

Repaso tutoría anterior

- Vimos los archivos RData:
 - Los archivos RData, son archivos muy livianos que nos permiten importar el **enviroment** de Rstudio.



The screenshot shows the RStudio Environment pane. At the top, there are tabs for 'Environment', 'Viewer', and 'Presentation'. Below the tabs, there are icons for 'Import Dataset', '427 MiB', and a 'List' button. The 'Global Environment' is selected. The 'Data' section lists four variables: 'data_master' (714 obs. of 12 variables), 'data_train' (500 obs. of 12 variables), 'data_valid' (214 obs. of 12 variables), and 'modelo_RF' (List of 18). The 'Values' section shows the 'split_rows' variable as an integer vector of length 500, with values 377, 294, 215, 60, 203, 357, 281, 460, 216, 285, and so on.

Data	
data_master	714 obs. of 12 variables
data_train	500 obs. of 12 variables
data_valid	214 obs. of 12 variables
modelo_RF	List of 18
Values	
split_rows	int [1:500] 377 294 215 60 203 357 281 460 216 285 ...

Figura: *Enviroment* en Rstudio

Repaso tutoría anterior

- Vimos los archivos RData:
 - Los archivos RData, son archivos muy livianos que nos permiten importar el **enviroment** de Rstudio.

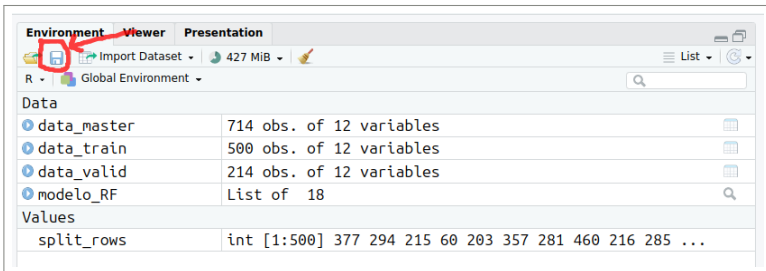


Figura: *Enviroment* en Rstudio

Vamos a R para ver su funcionamiento →

R Base vs. dplyr

Podemos realizar las mismas operaciones usando los métodos de **Rbase** y **dplyr**. **dplyr** son métodos más intuitivos para operar bases de datos.

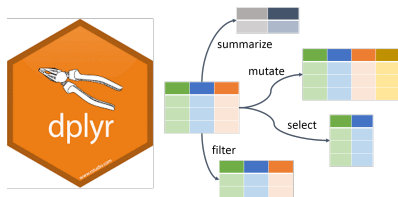
Rbase

```
data[c(" variable_1" ),]  
data[data$variable_1 > n,]
```

dplyr

```
data%>% select(variable_1)  
data%>% filter(variable_1 > 1)
```

dplyr



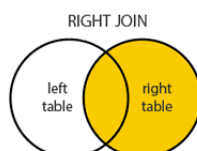
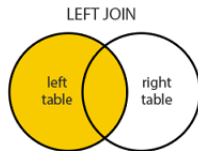
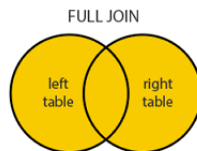
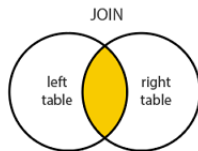
- Vimos los métodos de **mutate**:
 - *mutate_all*, *mutate_at* y *mutate_if*.
- Vimos los métodos de **summarise**:
 - *summarise_all*, *summarise_at* y *summarise_if*.

Algunas aplicaciones comunes

- `data %>% mutate_all(as.numeric)`
- `data %>% summarise_all_all(~ sum(is.na(.)))`

Familia de funciones join

Además, dplyr nos permite unir utilizando la familia de funciones join: `inner_join`, `full_join`, `left_join`, `right_join` y `anti_join`.



tidyr



- Es una colección de funciones que nos permiten ordenar los datos y reestructurarlos, con métodos más avanzados.
- Contempla, principalmente, las siguientes transformaciones:
 - **Tablas "largas" .anchas**: con las funciones `pivot_longer` y `pivot_wider`.
 - **Separar y unir**: con las funciones `unite` y `separate`.

tidyr

Reestructuración: formato *wide* y Formato *long*

- **pivot_longer:** nos permite transformar datos "anchos" (wide), a datos "largos".
- **pivot_wider:** nos permite transformar datos "largos" a datos "anchos".

wide vs long

				ID	ID2	A
1	a	1	a	1	a1	
				2	a1	
				3	a1	
2	a	2	a	1	a2	
				2	a2	
				3	a2	
3	a	3	a	1	a3	
				2	a3	
				3	a3	

tidyr

Pivot longer

Codigo

```
1 data_wide %>%  
2   pivot_longer(cols = [columnas a transformar],  
3                 names_to = [variable etiquetado],  
4                 values_to = [columna con valores a lo  
                             largo])
```

tidyr

Pivot wider

Codigo

```
1 data_long %>%  
2   pivot_wider(names_from = [etiquetados],  
3               values_from = [variable con valores])
```

Ejercicios

Ejercicios

- 1. Trabaje con la base de datos de `elsoc_long` y `elsoc_wide`.**
Pase de long a wide, y de wide a long.
- 2. Summarise.**
- 3. Para unir dos bases de datos, una tiene que estar en wide.**

Introducción a Programación Funcional

- La **programación funcional**, como indica su nombre, es un paradigma que consiste en la configuración de programas por medio de la **aplicación** y **composición** de funciones (procesos).
- En R, el paradigma predominante de programación es la **programación funcional**.
- No solo implica la aplicación de funciones, sino que implica la creación de funciones. Para crear funciones o construir nuestras propias mecánicas necesitamos algunos procesos básicos: **iterativos** y/o **vectorizados**.

Crear una función

§ Redundancia

```
1 df$a[df$a == -99] <- NA
2 df$b[df$b == -99] <- NA
3 df$c[df$c == -98] <- NA
4 df$d[df$d == -99] <- NA
5 df$e[df$e == -99] <- NA
6 df$f[df$g == -99] <- NA
```

(Wickham, 2023)

Funciones

Un recurso valioso

También, puede guardar funciones, que son mecanismos que se aplican a los objetos, pero mantiene el objeto como variable.

```
suma <- function(x, y){  
  resultado <- x + y  
  return(resultado) # return, opcional.  
}
```


Funciones

Crear una función

```
descriptivos <- function(data, rounded = 2, remove_na = TRUE){  
  data <- summarise_all(  
    list(  
      promedio = ~round(mean(., na.rm = remove_na), rounded),  
      desv = ~round(mean(., na.rm = remove_na), rounded),  
      minimo = ~round(mean(., na.rm = remove_na), rounded),  
      maximo = ~round(mean(., na.rm = remove_na), rounded)  
    )  
  )  
}
```

Funciones

Ejercicios

Ejercicios funciones

- 1 Arme una función del promedio.
- 2 Arme una función que cuente los NA.

Estructuras condicionales

if, else, ifelse

Las estructuras condicionales ejecutan una instrucción en base a una condición lógica.

- **if:** ejecuta un bloque de código si se ha cumplido la condición.
- **if + else:** al agregar **else**, esta instrucción permite ejecutar código solo si no se cumple la primera condición de **if**. **if** y **else** suelen ir juntos.
- **ifelse:** permite expresar un valor si cada uno de los valores del vector cumple la condición. Para el resto aplica otros valores.

No se pueden aplicar funciones.

Loops (bucles)

for y while

Puedo repetir un bloque de código de manera controlada, ya sea:

- **Bucle for:** recorriendo un número definido de elementos.
- **Bucle while:** iterando hasta que se cumpla una condición.

Otros procesos iterativos

Familia Apply

También es posible encontrar las funciones de la **familia apply**: *apply*, *tapply*, *sapply* y *lapply*. Las más relevantes para bases de datos son las últimas dos.

- **sapply**: nos permite aplicar una misma función a todas las variables de una base de datos, y devuelve una matriz.
- **lapply**: nos permite aplicar una misma función a todas las variables de una base de datos, y nos devuelve una lista.

La desventaja de estos métodos es que devuelve una matriz, aunque siempre es posible transformar esta matriz en un `data.frame`.

Otros procesos iterativos

Purrr

Con una aplicación similar, se encuentran los métodos de **purrr**, los cuales son similares a `lapply` y `sapply`, pero permiten recorrer diferentes elementos y tienen como resultado vectores o matrices de datos, en lugar de matrices.

```
numeros <- c(0, 5, 8, 3, 2, 1)
```

```
multiplicar_dos <- function(x) (2*x)
```

```
map(numeros, multiplicar_dos )
```

```
map_dbl(numeros, multiplicar_dos )
```

```
map_df(data.frame(numeros), multiplicar_dos)
```

Ejercicios

Ejerccios

- 1 Arme una función que muestre la distribución de lanzar n veces un dado.
- 2 Arme una tabla de promedios con intervalos de confianza utilizando `summarise_all`, `sapply` y `map` ¿Cuál es la diferencia?
- 3 Genere una estructura de control que parta en dos los datos (`split`). Si los datos son pares, haga un `split` entre la mitad de los datos. Si es impar, haga un `split` que se aproxime a esta decisión.

Referencias I



Wickham, H. (2023). Functional Programming. *En Advanced R.*
([enlace](#)).