Módulo 3 — Tidyverse II

Uso de las funciones más comunes del paquete `purrr`

CEPAL - Unidad de Estadísticas Sociales

2025-11-06

Introducción

Motivación

- En muchos análisis aplicamos funciones repetidas sobre columnas o listas.
- `purrr` ofrece una forma funcional, segura y legible de reemplazar bucles `for` o `apply`.
- Se integra perfectamente con el ecosistema `tidyverse`.

¿Qué es purrr?

- Es parte del `tidyverse`.
- Permite aplicar funciones sobre elementos de listas o columnas.
- Sus funciones comienzan con `map`.
- Devuelven resultados con un tipo de salida predecible.

Sintaxis general

```
map(.x, .f)
```

- `.x`: lista o vector de entrada.
- `.f`: función que se aplica a cada elemento.

Ejemplo básico:

```
library(purrr); library(tidyverse); library(tibble)
map(.x = 1:3, .f = sqrt) # lista
```

```
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051
```

Tipos de salida

```r
map_dbl(.x = 1:3,.f =  sqrt)   # numérico
```

```
[1] 1.000000 1.414214 1.732051
```

```r
map_chr(.x = 1:3,.f =  as.character)  # texto
```

```
[1] "1" "2" "3"
```

```r
map_df(.x = 1:3, ~  tibble(x = .x, y = .x^2)) # data frame
```

x	y
1	1
2	4
3	9

Aplicar funciones a columnas numéricas

Ejemplo: calcular medias de las variables monetarias del hogar.

```
datos <- readRDS("data/base_personas_gasto.rds")
datos %>%
  select(ingreso, gasto, ingreso_hh, gasto_hh) %>%
  map_dbl(mean, na.rm = TRUE)
```

```
   ingreso      gasto ingreso_hh   gasto_hh
  3428.413   3318.872  12116.451  11758.376
```

Uso con `mutate()` y `map_df()`

Calcular promedios por UPM a partir de una lista de data frames:

```
datos_list <- datos %>%
  group_split(upm)

resumen <- datos_list %>%
  map_df( ~ summarise(
    .x,
    upm = first(upm),
    media_ingreso = mean(ingreso),
    media_gasto = mean(gasto)
  ))
```

Resultados del uso con `mutate()` y `map_df()`

```
head(resumen)
```

upm	media_ingreso	media_gasto
1100100006	3848.260	3776.702
1100100008	5309.785	5309.785
1100100010	3367.852	3367.852
1100100011	5888.177	5888.177
1100100013	6647.007	6494.412
1100100016	3485.335	3485.335

Funciones anónimas con map()

```
map(1:3, ~ .x^2)
```

```
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9
```

```
map_chr(1:3, ~ paste0("UPM_", .x))
```

```
[1] "UPM_1" "UPM_2" "UPM_3"
```

Funciones personalizadas

```
media_trim <- function(x, trim = 0.1) mean(x, trim = trim)

map_dbl(list(datos$ingreso, datos$gasto), media_trim)
```

```
[1] 2560.970 2560.683
```

- El argumento **trim** indica la fracción de observaciones que se eliminarán de cada extremo de la distribución antes de calcular la media.

Combinación con `mutate()`

Crear una columna resumen por hogar que contenga listas de indicadores:

```
datos_hogar <- datos %>%
  group_by(id_hogar) %>%
  summarise(info = list(tibble::tibble(ingreso, gasto)))

temp <- datos_hogar %>%
  mutate(resumen = map(info, ~ mean(.x$ingreso))) %>%
  head()
```

Restultados de la combinación con `mutate()`

```
temp
```

id_hogar	info	resumen
262	5391.527, 5391.527, 5391.527, 5391.527	5391.527
265	7077.083, 7105.557, 7077.083, 7077.083, 7105.557, 7077.083	7086.574
277	2418.750, 2390.277, 2390.277, 2418.750, 2418.750, 2390.277, 2390.277, 2418.750	2404.513
288	1375.000, 1346.527, 1346.527, 1346.527, 1375.000, 1346.527, 1346.527, 1346.527	1353.645
289	1561.527, 1561.527, 1561.527, 1561.527, 1561.527, 1561.527, 1561.527, 1561.527, 1561.527, 1561.527	1561.527
291	1467.527, 1467.527, 1467.527, 1467.527, 1467.527, 1467.527, 1467.527, 1467.527, 1467.527, 1467.527	1467.527

map2() para dos vectores

```
map2(.x = 1:4, .y = 11:14, ~ .x + .y)
```

```
[[1]]
[1] 12

[[2]]
[1] 14

[[3]]
[1] 16

[[4]]
[1] 18
```

pmap() para múltiples columnas

```
df <- tibble(a = 1:3, b = 4:6, c = 7:9)
df
```

a	b	c
1	4	7
2	5	8
3	6	9

```
pmap_dbl(df, ~ mean(c(...)))
```

```
[1] 4 5 6
```

```
rowMeans(df)
```

```
[1] 4 5 6
```

Integración con `stringr`

Seleccionar variables que contengan la palabra "ingreso".

```
vars <- names(datos)
vars
```

```
 [1] "id_hogar"   "id_pers"    "upm"        "estrato"    "area"
 [6] "fep"        "pobreza"    "ingreso_hh" "gasto_hh"   "parentesco"
[11] "edad"       "sexo"       "etnia"      "anoest"     "niveduc_ee"
[16] "ingreso"    "gasto"
```

```
map_lgl(vars, ~ str_detect(.x, "ingreso"))
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALS
[13] FALSE FALSE FALSE  TRUE FALSE
```

Ejemplo completo (I)

1. Seleccionar columnas con “ingreso” y “gasto”.
2. Calcular su media.
3. Unir resultados con `bind_rows()`.

```
cols <- names(datos)[str_detect(names(datos), "ingreso|gasto")]

resultados <- cols %>%
  map_df(~ tibble(variable = .x,
                  media = mean(pull(datos, .x))))
```

```
resultados
```

variable	media
ingreso_hh	12116.451
gasto_hh	11758.376
ingreso	3428.413
gasto	3318.872

Ejemplo completo (II)

Divida `datos` por `area` y calcule la media de ingreso por grupo.

```
datos %>%
  split(~ area + sexo) %>%
  map_dbl(~ mean(.x$ingreso))
```

```
1.Hombre 2.Hombre  1.Mujer  2.Mujer
3783.243 2585.789 3580.095 2287.222
```

Ejemplo completo (III)

Calcule simultáneamente ingreso y gasto promedio por área.

```
datos %>%
  group_split(area) %>%
  map_df(~ summarise(.x,
                     area = first(area),
                     ing_prom = mean(ingreso),
                     gast_prom = mean(gasto)))
```

area	ing_prom	gast_prom
1	3677.658	3546.095
2	2436.735	2414.815

Beneficios del uso de `purrr`

- Reemplaza bucles con código más claro y funcional.
- Permite aplicar funciones personalizadas fácilmente.
- Se integra con `mutate` y `summarise`.
- Ideal para automatizar procesos repetitivos.

Conclusión

- `purrr` mejora la legibilidad y eficiencia del código.
- Es esencial para programación funcional en R.
- Facilita el trabajo con listas, data frames y estructuras complejas.
- Clave para análisis reproducibles y modulares.