Módulo 2 — Tidyverse I

mutate() y transmute()

CEPAL - Unidad de Estadísticas Sociales

2025-10-31

Introducción

Hasta ahora, hemos aprendido a seleccionar (`select`) y filtrar (`filter`) observaciones o columnas dentro de un conjunto de datos.

El siguiente paso en la cadena de manipulación es *transformar* o *crear* variables nuevas.

Aquí entran en juego dos funciones clave del paquete **dplyr**:

- `mutate()` $\rightarrow$ modifica o agrega columnas sin eliminar las existentes.
- `transmute()` $\rightarrow$ genera nuevas columnas, descartando las anteriores.

Ambas son esenciales para construir indicadores, limpiar datos o preparar insumos para modelos estadísticos.

Verbo `mutate()`

El verbo *mutate* permite encadenar transformaciones sin perder las variables originales.

- Agrega **nuevas variables** o **modifica existentes**.
- Mantiene todas las columnas originales.
- Puede usar variables recién creadas dentro del mismo `mutate()`.

Lectura de datos

```
datos <- readRDS("data/base_personas_gasto.rds")
library(dplyr)
```

Verbo `transmute()`

- Similar a `mutate()`, pero **solo conserva las variables creadas**.
- Útil para construir indicadores o reducir la salida.

```
datos %>% transmute(log_ingreso = log(ingreso)  ) %>%
  head()
```

log_ingreso
8.592584
8.592584
8.864617
8.868632
8.864617
7.791006

Devuelve un tibble únicamente con las nuevas variables.

Crear nuevas variables con `mutate()`

```
datos %>%
  mutate(
    ingreso_pc = ingreso_hh / n(),  # ingreso per cápita del hogar
    log_ingreso = log1p(ingreso),  # log(ingreso + 1)

    # gasto relativo al ingreso del hogar
    gasto_relativo = gasto / ingreso_hh
  ) %>%
  select(id_hogar, id_pers, ingreso_pc, log_ingreso, gasto_relativo) %>%
  head(10)
```

Resultados del `mutate()`

id_hogar	id_pers	ingreso_pc	log_ingreso	gasto_relativo
262	1	0.5550550	8.592769	0.5000000
262	2	0.5550550	8.592769	0.5000000
265	1	1.0943390	8.864758	0.3328869
265	2	1.0943390	8.868773	0.3342262
265	3	1.0943390	8.864758	0.3328869
277	1	0.4950869	7.791419	0.2514802
277	2	0.4950869	7.779583	0.2485198
277	3	0.4950869	7.779583	0.2485198
277	4	0.4950869	7.791419	0.2514802
288	1	0.2787142	7.226936	0.2539440

Crear un subconjunto reducido con el verbo `transmute()`

```
datos %>%
  transmute(id_hogar, id_pers,
    log_gasto = log1p(gasto_hh),
    eficiencia = gasto_hh / ingreso_hh
  ) %>% head(5)
```

id_hogar	id_pers	log_gasto	eficiencia
262	1	9.285824	1
262	2	9.285824	1
265	1	9.964617	1
265	2	9.964617	1
265	3	9.964617	1

`transmute()` **descarta** todas las columnas originales y **solo mantiene** las variables creadas o seleccionadas.

Integrar `select()`, `filter()` y `mutate()`

```r
datos %>%
  select(area, estrato, ingreso, gasto) %>%
  filter(ingreso > 0, gasto > 0) %>%
  mutate(
    log_ingreso = log(ingreso),
    log_gasto = log(gasto),
    gasto_rel = gasto / ingreso
  ) %>% head()
```

Un flujo limpio: primero seleccionas → luego filtras → luego transformas.

area	estrato	ingreso	gasto	log_ingreso	log_gasto	gasto_rel
1	11001	5391.527	5391.527	8.592584	8.592584	1
1	11001	5391.527	5391.527	8.592584	8.592584	1
1	11001	7077.083	7077.083	8.864617	8.864617	1
1	11001	7105.557	7105.557	8.868632	8.868632	1
1	11001	7077.083	7077.083	8.864617	8.864617	1
1	11001	2418.750	2418.750	7.791006	7.791006	1

`mutate()` con `across()` para aplicar funciones a varias columnas

```
datos %>%
  mutate(across(
    .cols = c(ingreso, gasto, ingreso_hh, gasto_hh),
    .fns = log1p,
    .names = "log_{.col}"
  )) %>%
  select(starts_with("log_")) %>% head()
```

`log1p(x)` aplica `log(1+x)` evitando errores cuando hay ceros. Es común en transformaciones de ingresos o gastos.

`across()` evita repetir transformaciones columna por columna.

log_ingreso	log_gasto	log_ingreso_hh	log_gasto_hh
8.592769	8.592769	9.285824	9.285824
8.592769	8.592769	9.285824	9.285824
8.864758	8.864758	9.964617	9.964617
8.868773	8.868773	9.964617	9.964617
8.864758	8.864758	9.964617	9.964617
7.791410	7.791410	9.171501	9.171501

Flujo completo `mutate()` + `transmute()`

```
datos %>%
  mutate(
    ingreso_pc = ingreso_hh / n(),
    eficiencia = gasto_hh / ingreso_hh,
    eficiencia_std = scale(eficiencia)
  ) %>%
  transmute( id_hogar, ingreso_pc,eficiencia_std
  ) %>% head()
```

`mutate()` sirve para cálculos intermedios y `transmute()` para dejar solo lo final.

id_hogar	ingreso_pc	eficiencia_std
262	0.5550550	0.09020827
262	0.5550550	0.09020827
265	1.0943390	0.09020827
265	1.0943390	0.09020827
265	1.0943390	0.09020827
277	0.4950869	0.09020827

Escalar y centrar variables económicas

```r
datos_mut1 <- datos %>%
  mutate(
    across(
      .cols = c(ingreso, gasto, ingreso_hh, gasto_hh),
      .fns = scale,
      .names = "{.col}_z"
    )
  ) %>%
  select(ends_with("_z"))
```

Se crean versiones estandarizadas (media 0, varianza 1) de las variables de ingreso y gasto, tanto individuales como del hogar.

Resultado

```
datos_mut1 %>% head(10)
```

ingreso_z	gasto_z	ingreso_hh_z	gasto_hh_z
0.3403478	0.5177630	-0.06404706	-0.07216592
0.3403478	0.5177630	-0.06404706	-0.07216592
0.6325753	0.9388263	0.43917844	0.70302210
0.6375117	0.9459392	0.43917844	0.70302210
0.6325753	0.9388263	0.43917844	0.70302210
-0.1750468	-0.2248566	-0.12000547	-0.15836640
-0.1799833	-0.2319694	-0.12000547	-0.15836640
-0.1799833	-0.2319694	-0.12000547	-0.15836640
-0.1750468	-0.2248566	-0.12000547	-0.15836640
-0.3560033	-0.4855923	-0.32191075	-0.46938909

Cuadrado y raíz cuadrada de variables continuas

```r
datos_mut3 <- datos %>%
  mutate(
    across(
      c(edad, ingreso),
      list(cuadrado = ~ .x^2, raiz = ~ sqrt(.x)),
      .names = "{.fn}_{.col}"
    )
  ) %>%  select(edad, ingreso, matches("cuadrado|raiz"))
```

Genera versiones cuadradas y raíces cuadradas de edad y anoest. Muy útil en modelos no lineales o para capturar efectos de curvatura.

Resultados

```
datos_mut3 %>% head(10)
```

edad	ingreso	cuadrado_edad	raiz_edad	cuadrado_ingreso	raiz_ingreso
51	5391.527	2601	7.141428	29068560	73.42702
46	5391.527	2116	6.782330	29068560	73.42702
26	7077.083	676	5.099019	50085109	84.12540
24	7105.557	576	4.898980	50488936	84.29446
7	7077.083	49	2.645751	50085109	84.12540
42	2418.750	1764	6.480741	5850352	49.18079
20	2390.277	400	4.472136	5713423	48.89046
17	2390.277	289	4.123106	5713423	48.89046
13	2418.750	169	3.605551	5850352	49.18079
60	1375.000	3600	7.745967	1890625	37.08099

Redondear y categorizar numéricamente sin condicionales

```r
datos_mut6 <- datos %>%
  mutate(
    across(
      c(ingreso, gasto),
      list(redondeado = ~ round(.x)),
      .names = "{.fn}_{.col}" ),
    across(
      c(ingreso, gasto),
      list(
        clasificado = ~ cut(
          .x,
          breaks = quantile(.x, probs = seq(0, 1, 0.25), na.rm = TRUE),
          include.lowest = TRUE,
          labels = c("Q1 (Bajo)", "Q2", "Q3", "Q4 (Alto)")
        )
      ), .names = "{.fn}_{.col}"
    )  )  %>% select(gasto, ingreso, matches("redondeado|clasificado"))
```

Resultados

```
datos_mut6 %>% head(10)
```

gasto	ingreso	redondeado_ingreso	redondeado_gasto	clasificado_ingreso	clasificado_gasto
5391.527	5391.527	5392	5392	Q4 (Alto)	Q4 (Alto)
5391.527	5391.527	5392	5392	Q4 (Alto)	Q4 (Alto)
7077.083	7077.083	7077	7077	Q4 (Alto)	Q4 (Alto)
7105.557	7105.557	7106	7106	Q4 (Alto)	Q4 (Alto)
7077.083	7077.083	7077	7077	Q4 (Alto)	Q4 (Alto)
2418.750	2418.750	2419	2419	Q3	Q3
2390.277	2390.277	2390	2390	Q3	Q3
2390.277	2390.277	2390	2390	Q3	Q3
2418.750	2418.750	2419	2419	Q3	Q3
1375.000	1375.000	1375	1375	Q1 (Bajo)	Q1 (Bajo)

Comparación resumida

Función	Conserva columnas originales	Crea nuevas variables	Ejemplo típico
`mutate()`	Sí	Sí	Añadir una columna calculada
`transmute()`	No	Sí	Crear indicadores derivados

Recomendaciones de uso

- Usa `mutate()` para *procesos exploratorios o pasos intermedios*.
- Usa `transmute()` para *resultados finales o salidas resumidas*.
- Combina con `across()` para aplicar transformaciones en bloque.

Conclusión

Ambas funciones son **pilares en el flujo de trabajo del `tidyverse`**. Facilitan la transformación progresiva y reproducible de datos, alineando la sintaxis con la lógica de la manipulación funcional en R.