Validación de datos en R

Cátedra de Bioestadística, FCEyE, UNR

AUTOR/A
Gino Bartolelli

FECHA DE PUBLICACIÓN
3 de abril de 2025

Introducción

En este material de carácter introductorio vamos a abordar un procedimiento imprescindible en cualquier estudio que implique la recolección sistemática de información: la validación de los datos. Para ello adoptaremos un enfoque didáctico planteando distintos ejemplos.

Arranquemos suponiendo que tenemos la siguiente base de individuos:

```
library(tidyverse)

individuos = tribble(
  ~id, ~edad, ~mayor,
  1,   15,    "no",
  2,   NA,    "no",
  3,   23,    "si",
  4,   40,    "no",
  5,  120,    "ni"
  )
```

¿De qué manera probaríamos la validez de estos registros? Tendríamos que reconocer alguna lógica entre la información recopilada que derive en una o varias condiciones que deben cumplirse para consignar al registro como válido. Por ejemplo, ya que esta es una base de individuos, podríamos ver si entre las edades reportadas hay algún valor no admisible:

```
edad_invalida = individuos$edad < 0 | individuos$edad > 100
edad_invalida
```

```
[1] FALSE    NA FALSE FALSE  TRUE
```

Al parecer hay edades fuera de rango. Sabiendo esto, sería útil conocer cuántos registros presentan esta inconsistencia:

```
sum(edad_invalida, na.rm = T)
```

```
[1] 1
```

Además, tendríamos que identificar quiénes son estos individuos para corroborar su edad:

```
individuos$id[edad_invalida]
```

```
[1] NA  5
```

Con la misma lógica podríamos definir nuevas condiciones y seguir validando la base, pero este camino tiene varias desventajas:

- No queda registro de las condiciones que estoy evaluando, tengo que leer a través del código y saber identificarlas
- Las condiciones se evalúan por separado y los resultados van a parar a lugares distintos
- Hay una repetición innecesaria de código que lo vuelve extenso y poco eficiente

A continuación veremos una manera de evitar estos inconvenientes.

Conjunto de validación

Generalmente, todas las condiciones o `reglas` que evaluamos se guardan en una tabla:

```
reglas = tribble(
  ~id,  ~desc,                                    ~cond,
  "r1", "(edad) es faltante",                     "is.na(edad)",
  "r2", "(edad) fuera de rango",                  "edad < 0 | edad > 100",
  "r3", "(mayor) es faltante",                    "is.na(mayor)",
  "r4", "(mayor) debe ser si/no",                 "!mayor %in% c('no','si')",
  "r5", "Si (edad) >= 18, (mayor) debe ser 'si'", "edad >= 18 & mayor == 'no'",
  "r6", "Si (edad) <  18, (mayor) debe ser 'no'", "edad <  18 & mayor == 'si'"
  )
```

Definir el conjunto de validación es muy importante porque queda documentado explícitamente qué condiciones vamos a evaluar para validar la base y qué se prueba en cada una de ellas. Además, podemos agregar otra información relevante para usar después. Por ejemplo, podemos listar las variables involucradas en cada regla para luego analizar qué campos están más comprometidos.

Si analizamos las seis reglas que creamos, las validaciones se pueden clasificar en tres tipos:

- **Existencia**: el valor falta (o sobra)
- **Rango**: el valor cae fuera de un rango preestablecido
- **Consistencia**: el valor no es congruente respecto del valor de otro campo

Para lograr una validación exhaustiva de la base, es recomendable ir campo por campo definiendo reglas de los tres tipos.

Evaluar el conjunto de validación

El problema con estas reglas es que, para R, las condiciones lógicas son meramente caracteres (por más que la sintáxis sea correcta). Esto significa que como primer paso tenemos que *parsear* las condiciones, es decir, coercionarlas en una expresión que R pueda evaluar. Esto se hace con la función `parse()`:

```
expresion = parse(text = "is.na(edad)")
expresion
```

```
expression(is.na(edad))
```

En el ejemplo, `expresion` es un objeto de tipo *expression*, el cual puede evaluarse en un contexto apropiado con la función `eval()`:

```
eval(expresion, individuos)
```

```
[1] FALSE  TRUE FALSE FALSE FALSE
```

Incorporando ambos conceptos, podemos definir una función que se encargue de preparar el conjunto de validación para ser evaluado, y otra para confrontarlo con la base de datos:

```
# Funcion validador()
# argumentos:
#   - datos: conjunto de validacion
#   - id   : nombre de la columna en (datos) con el identificador
#   - cond : nombre de la columna en (datos) con la condicion
# salida: vector nombrado
validador = function(datos, id, cond){
  reglas = datos[[cond]]
  names(reglas) = datos[[id]]
  reglas
}

# Funcion validar()
# argumentos:
#   - datos    : conjunto de datos a validar
#   - id       : nombre de la columna en (datos) con el identificador
#   - validador: salida de validador()
# salida: tibble con el resultado de la validación
validar = function(datos, id, validador){
  sapply(
    validador,
    function(x) eval(parse(text = x), datos)
    ) |>
    dplyr::as_tibble() |>
    dplyr::mutate(registro = datos[[id]], .before = 0)
}
```

`validador()` simplemente crea un vector nombrado a partir del conjunto de validación, y `validar()` evalúa sistemáticamente las reglas contra los datos y devuelve el resultado en formato de tabla. Pongamos las funciones a prueba:

```
validacion = validar(individuos, "id", validador(reglas, "id", "cond"))
validacion
```

```
# A tibble: 5 × 7
  registro r1    r2    r3    r4    r5    r6
     <dbl> <lgl> <lgl> <lgl> <lgl> <lgl> <lgl>
1        1 FALSE FALSE FALSE FALSE FALSE FALSE
2        2 TRUE  NA    FALSE FALSE NA    FALSE
3        3 FALSE FALSE FALSE FALSE FALSE FALSE
4        4 FALSE FALSE FALSE FALSE TRUE  FALSE
5        5 FALSE TRUE  FALSE TRUE  FALSE FALSE
```

`validacion` guarda los resultados del proceso de validación. Los valores lógicos marcan la presencia (`TRUE`) o ausencia (`FALSE`) de inconsistencias. Si el resultado de la evaluación es incierto por la existencia de valores faltantes, el output es `NA`.

Las funciones de validación preservan los identificadores de registros y reglas. Esto es importante porque nos va a permitir unir los resultados con las bases originales y de esta manera incorporar información adicional.

Análisis de los resultados

En general, siempre es conveniente pasar los datos a formato largo:

```
validacion_largo = pivot_longer(validacion, -registro, names_to = "regla", values_to = "error")
head(validacion_largo)
```

```
# A tibble: 6 × 3
  registro regla error
     <dbl> <chr> <lgl>
1        1 r1    FALSE
2        1 r2    FALSE
3        1 r3    FALSE
4        1 r4    FALSE
5        1 r5    FALSE
6        1 r6    FALSE
```

Con esto podemos calcular fácilmente cualquier estadística de interés. Empecemos por lo más básico: ¿cuántos individuos presentan inconsistencias?

```
validacion_largo |>
  filter(error) |>
  distinct(registro) |>
  count()
```

```
# A tibble: 1 × 1
      n
  <int>
1     3
```

La pregunta siguiente sería quiénes son esos individuos y cuántos errores tienen:

```
validacion_largo |>
  filter(error) |>
  count(registro)
```

```
# A tibble: 3 × 2
  registro     n
     <dbl> <int>
1        2     1
2        4     1
3        5     2
```

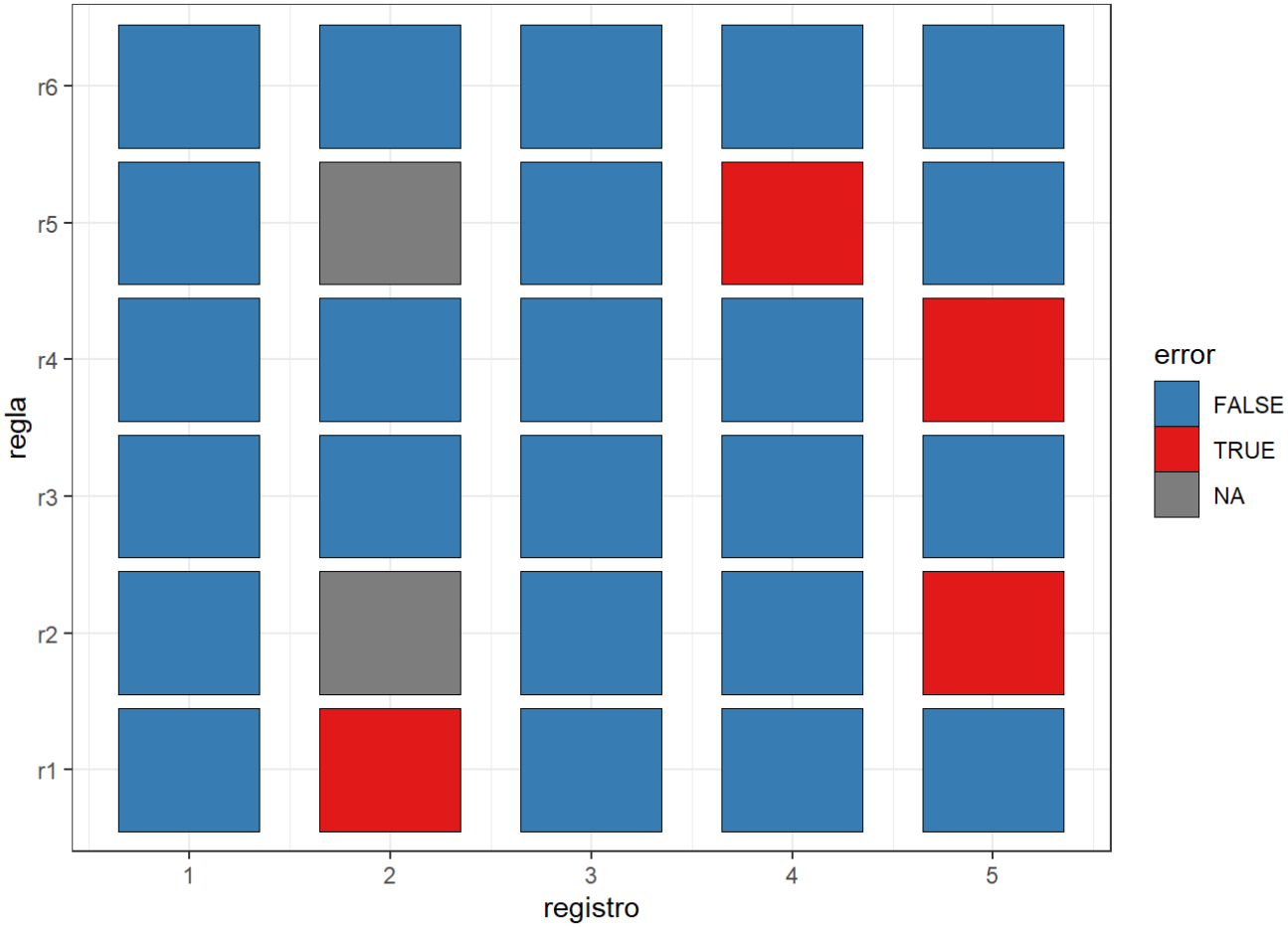
También podríamos listar qué reglas están comprometidas:

```
validacion_largo |>
  filter(error) |>
  group_by(registro) |>
  summarise(inconsistencias = str_flatten(regla, collapse = ", "))
```

```
# A tibble: 3 × 2
  registro inconsistencias
     <dbl> <chr>
1        2 r1
2        4 r5
3        5 r2, r4
```

Un gráfico puede ser efectivo dependiendo del volúmen de datos:

```
ggplot(validacion_largo)+
  aes(x = registro, y = regla, fill = error)+
  geom_tile(width = 0.7, height = .9, color = "black")+
  scale_fill_manual(values = c("#377EB8","#E41A1C","#999999"))+
  theme_bw()
```



Por último, si quisiéramos asegurarnos de que las reglas funcionan correctamente, podemos unir las bases de la siguiente manera:

```
validacion_largo |>
  filter(error) |>
```

```r
  left_join(reglas, by = c("regla" = "id")) |>
  left_join(individuos, by = c("registro" = "id")) |>
  select(-cond)
```

```
# A tibble: 4 × 6
  registro regla error desc                                    edad mayor
     <dbl> <chr> <lgl> <chr>                                  <dbl> <chr>
1        2 r1    TRUE  (edad) es faltante                       NA no
2        4 r5    TRUE  Si (edad) >= 18, (mayor) debe ser 'si'    40 no
3        5 r2    TRUE  (edad) fuera de rango                   120 ni
4        5 r4    TRUE  (mayor) debe ser si/no                  120 ni
```