Módulo 3 — Tidyverse II

Expresiones regulares y manipulación de texto en R

CEPAL - Unidad de Estadísticas Sociales

2025-11-06

Introducción

Las **expresiones regulares (regex)** son patrones que permiten **buscar, identificar, reemplazar o extraer** texto dentro de cadenas de caracteres. Son esenciales para limpiar variables categóricas, estandarizar formatos o identificar patrones complejos en los datos.

¿Por qué usar expresiones regulares?

- Permiten automatizar procesos de limpieza de texto.
- Se aplican a nombres de variables, códigos geográficos, respuestas abiertas, etc.
- Aumentan la precisión en la *detección de patrones* y en la *transformación de datos textuales*.
- Son el complemento natural de `stringr` dentro del *ecosistema tidyverse*.

Funciones principales del paquete `stringr`

Función	Descripción
`str_detect()`	Detecta si un patrón está presente en una cadena
`str_replace()`	Reemplaza coincidencias con un nuevo valor
`str_extract()`	Extrae la primera coincidencia del patrón
`str_sub()`	Extrae o reemplaza una subcadena por posición
`str_to_lower()`, `str_to_upper()`	Cambia mayúsculas/minúsculas
`str_trim()`	Elimina espacios en blanco
`str_split()`	Divide una cadena según un patrón
`str_c()`	Concatena cadenas de texto
`str_pad()`	Rellena una cadena hasta una longitud específica con un carácter definido

Ejemplo 1. Detectar patrones simples

```
library(stringr)
library(tidyverse)

texto <- c("Hombre", "Mujer", "Hombre", "No responde")

# Detectar si contiene la palabra "Hombre"
str_detect(texto, "Hombre")
```

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

Ejemplo 2. Uso de comodines y cuantificadores

Los símbolos más usados son:

- `.` → cualquier carácter
- `*` → cero o más repeticiones
- `+` → una o más repeticiones
- `?` → cero o una repetición
- `\\d` → dígito
- `\\s` → espacio
- `^` → inicio de cadena
- `$` → fin de cadena

```
str_detect(c("ABC123", "XYZ", "A12"), "^[A-Z]+\\d+$")
```

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

Detecta textos que **empiezan con letras mayúsculas** y **terminan con números**.

Ejemplo 3. Extraer valores numéricos de texto

\\d+ indica *uno o más dígitos consecutivos*.

```
datos <- readRDS("data/base_personas_gasto.rds") %>%
  transmute(
    id_hogar, id_pers, area,pobreza,edad,ingreso,
    area2 = if_else(
    area == "1",
    sample(c("Área 1", "area 1", "Area 1", "área 01"), n(),
           replace = TRUE),
    sample(c("Área 2", "area 2", "Area 02", "área 2"), n(),
           replace = TRUE)
  ),
   area_num = str_extract(area2, "\\d+"))
```

Ejemplo 3. Resultados.

```
table(datos$area2, datos$area_num)
```

/	01	02	1	2
área 01	3875	0	0	0
Area 02	0	978	0	0
area 1	0	0	3909	0
Area 1	0	0	3791	0
Área 1	0	0	3950	0
area 2	0	0	0	1030
área 2	0	0	0	973
Área 2	0	0	0	921

Ejemplo 4. Reemplazo de patrones

```
datos <- datos %>%
  mutate(pobreza2 = str_replace(pobreza, "3", "No pobre"),
         pobreza2 = str_replace(pobreza2, "1", "Pobre extremo"),
         pobreza2 = str_replace(pobreza2, "2", "Pobre"))
table(datos$pobreza2, datos$pobreza)
```

/	1	2	3
No pobre	0	0	16433
Pobre	0	2597	0
Pobre extremo	397	0	0

Esto facilita la recodificación de categorías directamente desde texto.

Ejemplo 5. Limpieza de nombres o códigos

```r
datos <- datos %>%
  mutate(area3 = str_to_lower(area2))

table(datos$area3)
```

área 01	area 02	area 1	área 1	area 2	área 2
3875	978	7700	3950	1030	1894

Se homogeneizan minúsculas, ideal antes de agrupar o unir tablas.

Ejemplo 6. Separar y unir texto

```
datos <- datos %>%
  mutate(id_hogar_txt = str_c("hh_", id_hogar))
head(datos %>% select(id_hogar, id_hogar_txt))
```

id_hogar	id_hogar_txt
262	hh_262
262	hh_262
265	hh_265
265	hh_265
265	hh_265
277	hh_277

```
str_split("A_B_C", "_")
```

```
[[1]]
[1] "A" "B" "C"
```

- `str_c()` concatena texto.
- `str_split()` separa cadenas en listas de subcadenas.

Ejemplo 7. Aplicación con `mutate()` y condiciones

```
datos <- datos %>%
  mutate(categoria_edad = case_when(
    str_detect(as.character(edad), "^[0-1]?[0-9]$") ~ "Niño o joven",
    str_detect(as.character(edad), "^[2-5][0-9]$") ~ "Adulto",
    TRUE ~ "Mayor"
  ))
head(datos %>% select(edad, categoria_edad))
```

edad	categoria_edad
51	Adulto
46	Adulto
26	Adulto
24	Adulto
7	Niño o joven
42	Adulto

El uso combinado de **regex** + `mutate()` permite construir variables limpias y reproducibles.

Ejemplo 8. Patrones múltiples con |

```
datos <- datos %>%
  mutate(area_1 = str_detect(area2, "1|01"),
         area_2 = str_detect(area2, "1|01",negate = TRUE))
table(datos$area_1, datos$area2)
```

/	área 01	Area 02	area 1	Area 1	Área 1	area 2	área 2	Área 2
FALSE	0	978	0	0	0	1030	973	921
TRUE	3875	0	3909	3791	3950	0	0	0

```
table(datos$area_2, datos$area2)
```

/	área 01	Area 02	area 1	Area 1	Área 1	area 2	área 2	Área 2
FALSE	3875	0	3909	3791	3950	0	0	0
TRUE	0	978	0	0	0	1030	973	921

El operador | actúa como "o". Permite buscar **más de un patrón** simultáneamente.

Ejemplo 9. Validación de formatos

Para verificar consistencia en IDs o códigos:

```
temp <- str_detect(unique(datos$id_hogar), "^[0-9]{4}$")
table(temp)
```

FALSE	TRUE
5421	798

También se usa para validar **correos electrónicos, cédulas o fechas**.

Comparación con otras funciones base R

Base R	Equivalente en `stringr`	Diferencias
`grep()`	`str_detect()`	`stringr` devuelve lógicos directamente
`gsub()`	`str_replace_all()`	Permite reemplazos múltiples más limpios
`substr()`	`str_sub()`	Soporta vectores y es más legible
`paste()`	`str_c()`	`stringr` maneja mejor los `NA`

Conclusiones

- Las expresiones regulares son una herramienta poderosa para limpiar, transformar y validar texto.
- El paquete `stringr` ofrece una sintaxis consistente y legible dentro del tidyverse.
- Su uso conjunto con `mutate()`, `case_when()` y `join()` permite automatizar procesos de depuración en grandes bases.