stringr

Entornos de Análisis de Datos: R

Alberto Torres Barrán

2021-01-07

Expresiones regulares

- Lenguaje que describe patrones en cadenas de caracteres
- La mayoría de lenguajes de programación implementan expresiones regulares
- La sintaxis difiere ligeramente
- Expresiones regulares en R
- *Vignette* expresiones regulares

Ejemplos

```
x <- c("moto", "coche", "autobus")
str_view(x, "co")
```

moto

coche

autobus

```
str_view(x, ".o")
```

moto

coche

autobus

```
str_view_all(x, ".o")
```

```
moto
coche
autobus
```

Caracteres especiales

- El `.` es un carácter especial de las expresiones regulares que concuerda con todos
- ¿Como podemos concordar con el carácter `"."`?
- Hay que "escapar" el `.` en la expresión regular con la barra invertida `"\"`
- Las expresiones regulares se representan con cadenas de caracteres:
  - la barra invertida también es un carácter especial de las cadenas de caracteres
  - tenemos que escapar la barra invertida con otra barra invertida

```
writeLines("\\.")
## \.
```

```
str_view(c("hola.", "adios."), "a\\.")
```

hola.

adios.

Anclas

`^` representa el inicio de la cadena y `$` representa el final

```
str_view(c("tapar", "destapar"), "^tapar")
```

```
tapar
```

```
destapar
```

Clases de caracteres

- Listas de caracteres. Se concuerda con cualquier carácter de la lista:
  - `[abc]`: a, b, o c
  - `[^abc]`: cualquier cosa excepto a, b, o c
  - `ab|cd`: "ab" o "cd", pero no "abd" ni "acd"
- Listas de caracteres predefinidas:
  - `[:alpha:]`, caracteres del alfabeto, `[A-z]`
  - `[:alnum:]`, caracteres alfanuméricos
  - `\\d` o `[:digit:]`: cualquier dígito
  - `\\s` o `[:space:]`: espacios, tabulación, saltos de línea y retorno de carro

Repetición

- `?`: 0 o 1 vez
- `+`: 1 o más
- `*`: 0 o más
- `{n}`: exactamente n veces
- `{n,}`: n veces o más
- `{,m}`: como mucho m veces
- `{n,m}`: entre n y m veces

```
n <- c("4.5", "6", ".5", "5.", "a", "4.a")
str_view(n, "^\\d+\\.?\\d*$")
```

4.5

6

.5

5.

a

4.a

Ejemplos

```
str_view(c("test@test.com", "test@test12.com", "test@test",
           "test@test.es", "@test.com", "te st@test.com", "test@test.com hola"),
         ".+@[^\\d\\s]+\\.(com|es)")
```

`test@test.com`

test@test12.com

test@test

`test@test.es`

@test.com

`te st@test.com`

`test@test.com` hola

```
str_view(c("test@test.com", "test@test12.com", "test@test",
           "test@test.es", "@test.com", "te st@test.com", "test@test.com hola"),
         ".+@[^\\d\\s]+\\.(com|es)$")
```

test@test.com

test@test12.com

test@test

test@test.es

@test.com

te st@test.com

test@test.com hola

```
str_view(c("981945678", "981 945678", "+34 981945678"),
         "(\\+34\\s)?\\d{9}")
```

```
981945678
```

```
981 945678
```

```
+34 981945678
```

```
str_view(c("981945678", "981 945678", "+34 981945678", "981 94 56 78"),
         "(\\+34\\s)?\\d{3}\\s?\\d{6}")
```

```
981945678
```

```
981 945678
```

```
+34 981945678
```

```
981 94 56 78
```

Cadenas de caracteres

- Las cadenas de caracteres se crean con comillas dobles `""` o simples `''`
- La `\` se usa para escapar ciertos caracteres especiales: `"\""`, `"\\"`, `"\n"`, etc.

Librería stringr

- Implementa muchas operaciones con cadenas de caracteres
- Todas las funciones comienzan con el prefijo común `str_`
- Muchas de ellas tienen como entrada las expresiones regulares que vimos anteriormente

Operaciones básicas con cadenas

- Longitud

```
str_length(c("hola", "alberto", NA))
## [1]  4  7 NA
```

- Concatenar cadenas

```
str_c("a", "b", "c")
## [1] "abc"
```

```
str_c("a", "b", "c", sep = ", ")
## [1] "a, b, c"
```

```
str_c("pre-", c("a", "b", "c"), "-suf")
## [1] "pre-a-suf" "pre-b-suf" "pre-c-suf"
```

Indexando cadenas

- Se puede obtener una subcadena a partir de las posiciones

```
str_sub("hola", 2, 3)
## [1] "ol"
```

- También se puede modificar si le asignamos un nuevo valor

```
x <- c("hola", "que", "tal")
str_sub(x, 2, 4) <- str_to_upper(str_sub(x, 2, 4))
x
## [1] "hOLA" "qUE"  "tAL"
```

Operaciones con expresiones regulares

- `str_detect()` devuelve un vector lógico indicando si la expresión regular concuerda con la cadena o no

```
str_detect(c("aba", "ebf", "atp"), "^a")
## [1]  TRUE FALSE  TRUE
```

- `str_count()` devuelve **cuántas** concordancias hay en cada cadena

```
str_count(c("aba", "ebf", "atp"), "a")
## [1] 2 0 1
```

Extraer concordancias

- `str_extract()` : extrae la parte de la cadena que concuerda con la expresión regular (únicamente la primera concordancia)

```
str_extract(c("ab (cd)", "ef (gh)", "ij (kl)"), "\\(.*\\)")
## [1] "(cd)" "(gh)" "(kl)"
```

- `str_extract_all()` : devuelve todas las concordancias

```
str_extract_all(c("a b c", "a f g"), "[abc]")
## [[1]]
## [1] "a" "b" "c"
##
## [[2]]
## [1] "a"
```

Reemplazar concordancias

- `str_replace()` : reemplaza la primera concordancia por otra cadena de texto

```
x <- c("coche", "moto", "autobus")
str_replace(x, "[aeiou]", "-")
## [1] "c-che"   "m-to"    "-utobus"
```

- `str_replace_all()` : reemplaza **todas** las concordancias

```
str_replace_all(x, "[aeiou]", "-")
## [1] "c-ch-"   "m-t-"    "--t-b-s"
```

Dividir una cadena

- `str_split()` divide una cadena de acuerdo con una expresión regular

```
str_split(c("a b c", "a f g"), "\\s")
## [[1]]
## [1] "a" "b" "c"
##
## [[2]]
## [1] "a" "f" "g"
```

- `str_split_fixed()` es una variante donde se especifica exactamente cuantas partes se devuelven

```
str_split_fixed(c("a b c", "a f g"), "\\s", n = 3)
##      [,1] [,2] [,3]
## [1,] "a"  "b"  "c"
## [2,] "a"  "f"  "g"
```

- `tidyr::separate()` realiza esta misma operación sobre las columnas de un data frame, acepta expresiones regulares