stringr

Entornos de Análisis de Datos: R

Alberto Torres Barrán

2020-01-14

stringr

Cadenas de caracteres

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

Operaciones con cadenas

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

```
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))
```

Expresiones regulares

- Lenguaje que describe patrones en cadenas de caracteres
- La mayoría de lenguajes implementan expresiones regulares
- La sintaxis difiere ligeramente

Ejemplos

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

moto

coche

autobus

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

moto

coche

autobus

Caracteres especiales

- El `.` es un carácter especial de las regexp que hace `match` con cualquiera
- Como hacer match con el carácter `"."`?
- Se escapa el `.` en la regexp con el carácter `"\"`
- Como las regexp se representan como cadenas de caracteres, a su vez hay que escapar el `\`

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

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

hola.

adios.

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

```
str_view("carpeta\\fichero", "\\\\")
```

carpeta\fichero

Anclas

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

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

```
tapar
```

```
destapar
```

Clases

Otros patrones especiales:

- `\d` : cualquier dígito
- `\s` : espacios, tabulación y saltos de línea (e.g. space, tab, newline)
- `[abc]` : a, b, o c
- `[^abc]` : cualquier cosa excepto a, b, o c
- `ab|cd` : "ab" o "cd", pero no "abd" ni "acd"

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

Ejemplos

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

`test@test.com`

test@test12.com

test@test

`test@test.es`

@test.com

`te st@test.com`

```
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
```

Concordancia con un patrón

- `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 regula (solo se devuelve la primera)

```
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 las concordancias por una cadena (solo la primera)

```
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"
```

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