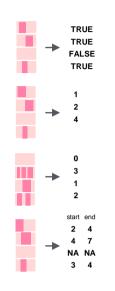
# Manipulación de cadenas con stringr : : GUÍA RÁPIDA

El paquete **stringr** proporciona un conjunto de herramientas internamente consistentes para trabajar con cadenas de caracteres, por ej. secuencias de caracteres delimitados por comillas.

# stringr

# **Detectar Coincidencias**



str\_detect(cadena, patrón) Detecta el patrón
coincidente en una cadena.
str detect(fruta, "a")

str\_which(cadena, patrón) Encuentra los índices de las cadenas que contienen un patrón coincidente. str which(fruta, "a")

str\_count(cadena, patrón) Cuenta el número de coincidencias en una cadena. str count(fruta, "a")

str\_locate(cadena, patrón) Localiza las posiciones en las que el patrón coincide con la cadena. También str\_locate\_all. str\_locate(fruta, "a")

# Subconjunto de cadenas



str\_sub(cadena, start = 1L, end = -1L) Extrae
subcadenas de un vector de caracteres.
str\_sub(fruta, 1, 3); str\_sub(fruta, -2)

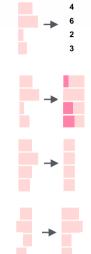
**str\_subset**(cadena, **patrón**) Devuelve sólo las cadenas que contienen un patrón coincidente. str\_subset(fruta, "b")



str\_extract(cadena, patrón) Devuelve el primer
patrón encontrado que coincide en cada cadena,
como un vector. También str\_extract\_all para
devolver cada patrón coincidente. str\_extract(fruta,
"[aeiou]")

str\_match(cadena, patrón) Devuelve el primer patrón encontrado que coincide en cada cadena, como una matriz, con una columna para cada una () agrupado por patrón. También str\_match\_all. str\_match(sentences, "(el|la) ([^]+)")

# **Gestionar Longitudes**



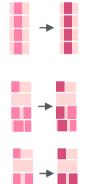
str\_length(cadena) Los largos de las cadenas (por ej. número de puntos de código, suele ser igual al número de caracteres). str\_length(fruta)

str\_pad(cadena, largo, side = c("left", "right",
"both"), pad = " ") Extiende cadenas a un largo
constante. str\_pad(fruta, 17)

str\_trunc(cadena, ancho, side = c("right", "left",
"center"), ellipsis = "...") Trunca el ancho de una
cadena, eliminando el contenido sobrante.
str\_trunc(fruta, 3)

**str\_trim**(cadena, side = c("both", "left", "right")) Elimina los espacios en blanco desde el inicio y/o al final de una cadena. *str\_trim(fruta)* 

# Transformar Cadenas



UNA

CADENA

una cadena

una cadena

UNA

**CADENA** 

una cadena

Una cadena

**str\_sub**() <- valor. Reemplaza subcadenas identificadas con str\_sub() y se asignan al resultado. str\_sub(fruta, 1, 3) <- "str"

**str\_replace**(cadena, **patrón**, reemplazo) Reemplaza el primer patrón coincidente en cada cadena. *str\_replace*(*fruta, "a", "-"*)

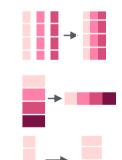
str\_replace\_all(cadena, patrón,
replacement) Remplaza todos los patrones
coincidentes en cada cadena.
str\_replace\_all(fruta, "a", "-")

**str\_to\_lower**(cadena, locale = "en")<sup>1</sup> Convierte cadenas a minúscula. str\_to\_lower(oraciones)

str\_to\_upper(string, locale = "en")¹ Convierte
cadenas a mayúsculas.
str\_to\_upper(oraciones)

**str\_to\_title**(string, locale = "en")<sup>1</sup> Convierte cadenas a título. *str\_to\_title*(oraciones)

# Juntar y Separar



{xx} {yy}

str\_c(..., sep = "", collapse = NULL) Une
múltiples cadenas en una. str\_c(letters,
 LETTERS)

str\_c(..., sep = "", collapse = NULL) Colapsa
un vector de cadenas en una sola cadena.
str\_c(letters, collapse = "")

str\_dup(cadena, veces) Repite cadenas
varias veces. str\_dup(fruta, times = 2)

str\_split\_fixed(cadena, patrón, n) Divide un vector de cadenas en una matriz de subcadenas (dividiendo en las ocurrencias de cada patrón de coincidencia). También str\_split para devolver una lista de subcadenas.
str\_split\_fixed(fruta, " ", n=2)

str\_glue(..., .sep = "", .envir = parent.frame())
Crea una cadena a partir de cadenas y
{expresiones} para evaluar. str\_glue("Pi is
{pi}")

str\_glue\_data(.x, ..., .sep = "", .envir =
parent.frame(), .na= "NA") Usa un conjunto de
datos, lista, o entorno para crear una cadena
a partir de cadenas y {expresiones} para
evaluar. str\_glue\_data(mtcars,
 "{rownames(mtcars)} has {hp} hp")

# **Ordenar Cadenas**



str\_order(x, decreasing = FALSE, na\_last =
TRUE, locale = "en", numeric = FALSE, ...)¹
Devuelve el vector de índices que ordena un
vector de caracteres. x[str\_order(x)]



str\_sort(x, decreasing = FALSE, na\_last =
TRUE, locale = "en", numeric = FALSE, ...)¹
Ordena un vector de caracteres.
str\_sort(x)

# Funciones auxiliares

**str\_conv**(cadena, encoding) Sobreescribe el tipo de codificación de una cadena. str\_conv(fruta, "ISO-8859-1")

manzana banana pera

manzana banana pera str\_view(cadena, patrón, match = NA) Vista
en HTML de la primera coincidencia de una
expresión regular en cada cadena.
str\_view(fruta, "[aeiou]")

str\_view\_all(cadena, patrón, match = NA)
Vista en HTML de todas las coincidencias de la
expresión regular. str\_view\_all(fruta, "[aeiou]")

**str\_wrap**(cadena, width = 80, indent = 0, exdent = 0) Envuelve cadenas en párrafos formateados adecuadamente. str\_wrap(sentences, 20)

<sup>1</sup> Ver bit.ly/ISO639-1 para una lista completa de locales.



# **Necesitas Saber**

Los argumentos de los patrones en stringr son interpretados como expresiones regulares después de cada carácter que ha šido segmentado

En R, se escriben expresiones regulares como cadenas, secuencias de caracteres rodeados de comillas dobles ("") o simples (").

Algunos caracteres no se pueden representar directamente como una cadena en R. Éstos son representados por caracteres especiales, secuencias de caracteres que tienen un significado específico., e.g.

Especial Character	Representa
	\
\"	"

nueva línea Ejecuta ?"" para ver una lista completa

Por esto, cuando aparece \ en una expresión regular, se tiene que escribir como \\ en la cadena que representa la expresión regular.

Usa writeLines() para ver como R ve tu cadena después de que todos los caracteres especiales se han parseado.

writeLines("\\.")

writeLines("\\is a backslash") #\is a backslash

### INTERPRETACIÓN

Los patrones en stringr son interpretados como regexs (expresiones regulares). Para cambiar este argumento predeterminado, envuelva el patrón con una de estas opciones:

regex(pattern, ignore\_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...) Modifica una regex para ignorar casos, coincide fin de líneas como también fin de cadenas, permite comentarios de R dentro de las regex para ignorar casos. de las regex , y/o tienen .coincide cualquier cosa incluyendo \n. str\_detect("I", regex("i", TRUE))

fixed() Coincide raw bits pero ignorará algunos caracteres que se pueden representar de múltiples formas (rápido). str\_detect("\u0130", fixed("i"))

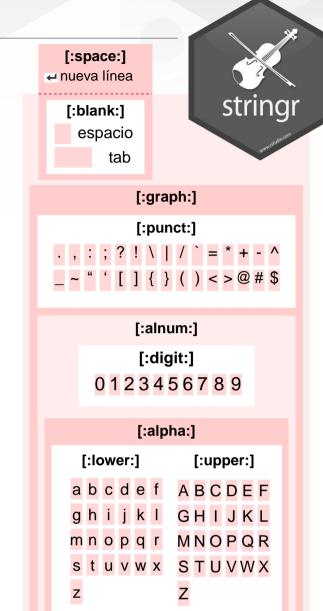
**coll**() Coincide raw bits y usará patrones específicos de los parámetros locales para reconocer caracteres que pueden ser representados en múltiples formas (lento). str\_detect("\u0130", coll("i", TRUE, locale = "tr"))

**boundary**() Coincide límites entre caracteres, separadores de líneas, sentencias o palabras. str\_split(sentencias, boundary("word"))

Expresiones Regulares Expresiones regulares, o regexps, es un lenguaje conciso para describir patrones en cadenas.

MATCH	CHARACTE	RS ver <- function(rx) str_	view_all("abc Al	BC 123\t.!?\\(){}\n", rx)
cadena (escribe esto)	regexp (para decir esto)	coincidencias (que coincide con esto)	ejemplo	
esio)	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
<b>\\</b> .	٧.		see("\\.")	abc ABC 123 .!?\(){}
\\!	<b>\!</b>	!	see("\\!")	abc ABC 123 . <mark>!</mark> ?\(){}
\\?	\?	?	see("\\?")	abc ABC 123 .! <mark>?\</mark> (){}
WV.	\\	\	see("\\\\")	abc ABC 123 .!? <mark>\</mark> (){}
\\(	\(	(	see("\\(")	abc ABC 123 .!?\ <mark>()</mark> {}
\\)	\)	)	see("\\)")	abc ABC 123 .!?\(){}
<b>\\</b> {	\{	{	see("\\{")	abc ABC 123 .!?\(){}
<b>\\</b> }	\}	}	see( "\\}")	abc ABC 123 .!?\(){}
\\n	\n	nueva línea (retorno)	see("\\n")	abc ABC 123 .!?\(){}
\\t	\t	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	\s	cualquier espacio en blanco (\S para no-blancos)	see("\\s")	abc ABC 123 .!?\(){}
\\d	\d	cualquier dígito (\D para no-dígits)	see("\\d")	abc ABC 123 .!?\(){}
\\w	\w	cualquier carácter (\W para no-caracteres	see("\\w")	abc ABC 123 .!?\(){}
\\b	\b	bordes de palabras	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:] 1	dígitos	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:] 1	letras	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:] 1	letras minúsculas	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:] 1	letras mayúsculas	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:] 1	letras y números	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:] 1	puntuación	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:] 1	letras, números, y puntuación	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:] 1	caracteres espacio (por ej. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:] 1	espacios y tab (pero no nueva línea)	see("[:blank:]")	abc ABC 123 .!?\(){}
		cada carácter excepto una nueva línea	see(".")	abc ABC 123 .!?\(){}
		1 Muchas funciones de R hase requieren que las clase	es se envuelvan en un	segundo juego de [] e a [[:dia

<sup>&</sup>lt;sup>1</sup> Muchas funciones de R base requieren que las clases se envuelvan en un segundo juego de [ ], e.g. [[:digit:]]



### **ALTERNATIVAS**

alt <- function(rx) str view all("abcde", rx)

regexp	coincidencias	ejemplo	
abld	0	alt("ab d")	abcde
[abe]	una de	alt("[abe]")	abcde
[^abe]	excepto	alt("[^abe]")	ab <mark>cd</mark> e
[a-c]	rango	alt("[a-c]")	abcde

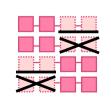
### **ANCLAS**

anchor <- function(rx) str\_view\_all("aaa", rx)

regexp	coincidencias	ejemplo	
<b>^</b> a	comienzo cadena	anchor("^a")	aaa
a\$	fin de cadena	anchor("a\$")	aaa

### MIRAR ALREDEDOR

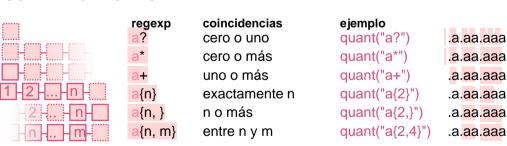
look <- function(rx) str view all("bacad", rx)



regexp	coincidencias	ejemplo	
a(?=c)	seguido por	look("a(?=c)")	b <mark>a</mark> cad
a(?!c)	no seguido por	look("a(?!c)")	bacad
(?<=b)a	precedido por	look("(?<=b)a")	b <mark>a</mark> cad
(? b)a</td <td>no precedido por</td> <td>look("(?<!--b)a")</td--><td>bac<mark>a</mark>d</td></td>	no precedido por	look("(? b)a")</td <td>bac<mark>a</mark>d</td>	bac <mark>a</mark> d

### **CUANTIFICADORES**

cuant <- function(rx) str view all(".a.aa.aaa", rx)



### **GRUPOS**

ref <- function(rx) str view all("abbaab", rx)

Usa paréntesis para fijar el precedente (orden de evaluación) y crea grupos

regexp	coincidencia	ejemplo	
(ab d)e	fija precedencia	alt("(ab d)e")	abc <mark>de</mark>

Usar un número escapado para referirse y duplicar un grupo de paréntesis que ocurre antes en un patrón. Referirse a cada grupo por su orden de aparición

Cadena	regexp	coincidencia	ejemplo
(escribe esto)	(para decir esto)	(que coincide con esto)	(el resultado es el mismo que ref("abba"))
\\1	<b>\1</b> (etc.)	first () group, etc.	ref("(a)(b)\\2\\1") abbaab

