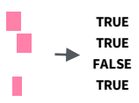
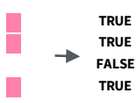
Manipulación de cadenas con stringr : : GUÍA RÁPPIDA

El paquete stringr proporciona un conjunto de herramientas internamente coherentes para trabajar con cadenas de caracteres, es decir, secuencias de caracteres entre comillas.

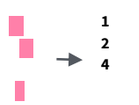
Detectar Coincidencias



str_detect(string, **pattern**, negate = FALSE**)** Detecte la presencia de una coincidencia de patrón en una cadena. Además, **str_like()**. str_detect(fruit, "a")

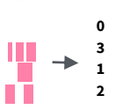


str_starts(string, **pattern**, negate = FALSE**)** Detecta la presencia de una coincidencia de patrón al principio de una cadena. Además, **str_ends()**. str_starts(fruit, "a")



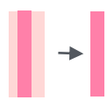
str_which(string, **pattern**, negate = FALSE**)** Buscar los índices de las cadenas que contienen una coincidencia de patrón. str_which(fruit, "a")

str_locate(string, **pattern**) Localice las posiciones de las coincidencias de patrones en una cadena. Además, **str_locate_all()**. str_locate(fruit, "a")

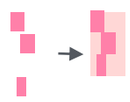


str_count(string, **pattern**) Contar el número de coincidencias en una cadena. str_count(fruit, "a")

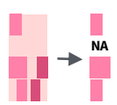
Subconjuntos de Texto



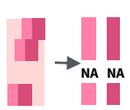
str_sub(string, start = 1L, end = -1L**)** Extraer subcadenas de un vector de caracteres. str_sub(fruit, 1, 3); str_sub(fruit, -2)



str_subset(string, **pattern**, negate = FALSE**)** Devuelve solo las cadenas que contienen una coincidencia de patrón. str_subset(fruit, "p")

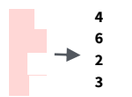


str_extract(string, **pattern**) Devuelve la primera coincidencia de patrón encontrada en cada cadena, como un vector. Además, **str_extract_all()** para devolver todas las coincidencias de patrones. str_extract(fruit, "[aeiou]")

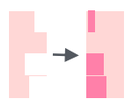


str_match(string, **pattern**) Devuelve la primera coincidencia de patrón encontrada en cada cadena, como una matriz con una columna para cada grupo ( ) en el patrón. Además **str_match_all()**. str_match(sentences, "(a|the) ([^ +])")

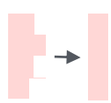
Trabajar con Longitudes



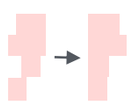
str_length(string**)** El ancho de los textos (es decir, el número de puntos de código, que generalmente es igual al número de caracteres). str_length(fruit)



str_pad(string, width, side = c("left", "right", "both"), pad = " "**)** Rellene los textos a un ancho constante. str_pad(fruit, 17)



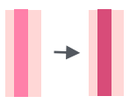
str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "..."**)** Trunque el ancho de los textos y reemplace el contenido por puntos suspensivos. str_trunc(sentences, 6)



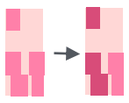
str_trim(string, side = c("both", "left", "right"**))** Recortar espacios en blanco desde el principio y/o el final de un texto. str_trim(str_pad(fruit, 17))

str_squish(string**)** Recorta los espacios en blanco de cada extremo y contrae varios espacios en espacios individuales. str_squish(str_pad(fruit, 17, "both"))

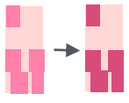
Mutar Cadenas de Texto



str_sub() <- value. Reemplace las subcadenas identificando las subcadenas con str_sub() y asignándolas a los resultados. str_sub(fruit, 1, 3) <- "str"



str_replace(string, **pattern**, replacement**)** Reemplace el primer patrón coincidente en cada cadena. Además, **str_remove()**. str_replace(fruit, "p", "-")



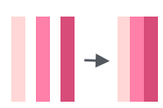
str_replace_all(string, **pattern**, replacement**)** Reemplace todos los patrones coincidentes en cada cadena. Además, **str_remove_all()**. str_replace_all(fruit, "p", "-")

str_to_lower(string, locale = "en"**)**[1] Convierte las cadenas a minúsculas. str_to_lower(sentences)

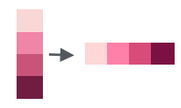
str_to_upper(string, locale = "en"**)**[1] Convierte las cadenas a mayúsculas. str_to_upper(sentences)

str_to_title(string, locale = "en"**)**[1] Convierta las cadenas a mayúsculas y minúsculas. Además **str_to_sentence()**. str_to_title(sentences)

Unir y Dividir

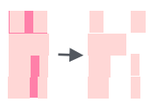


str_c(..., sep = "", collapse = NULL**)** Une varias cadenas en una sola cadena. str_c(letters, LETTERS)



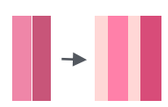
str_flatten(string, collapse = ""**)** Se combina en una sola cadena, separada por contracción. str_flatten(fruit, ", ")

str_dup(string, times**)** Repetir cadenas veces veces. Además, **str_unique()** para eliminar duplicados. str_dup(fruit, times = 2)



str_split_fixed(string, **pattern**, n**)** Divide un vector de cadenas en una matriz de subcadenas (división en las apariciones de una coincidencia de patrón). Además, **str_split()** to devuelve una lista de subcadenas y **str_split_n()** para devolver la enésima subcadena. str_split_fixed(sentences, " ", n=3)

str_glue(..., .sep = "", .envir = parent.frame()**)** Cree una cadena a partir de cadenas y {expresiones} para evaluarla. str_glue("Pi is {pi}")



str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA"**)** Utilice un marco de datos, una lista o un entorno para crear una cadena a partir de cadenas y {expresiones} para evaluarla. str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")

Ordenar Cadenas de Caracteres

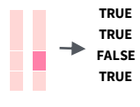
str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...**)**[1] Devuelve el vector de índices que ordena un vector de caracteres. fruit[str_order(fruit)]

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...**)**[1] Ordenar un vector de caracteres. str_sort(fruit)

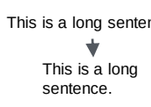
Ayudantes

str_conv(string, encoding**)** Invalide la codificación de un texto. str_conv(fruit,"ISO-8859-1")

str_view(string, **pattern**, match = NA**)** Ver la representación HTML de todas las coincidencias de expresiones regulares. str_view(sentences, "[aeiou]")



str_equal(x, y, locale = "en", ignore_case = FALSE, ...**)**[1] Determine si dos cadenas son equivalentes. str_equal(c("a", "b"), c("a", "c"))



str_wrap(string, width = 80, indent = 0, exdent = 0**)** Envuelve cadenas en párrafos con un buen formato. str_wrap(sentences, 20)

[1] Vea **bit.ly/ISO639-1** para obtener una lista completa de las configuraciones regionales.

Lo que Necesitas Saber

Los argumentos de patrón en stringr se interpretan como expresiones regulares después de que se hayan analizado los caracteres especiales.

En R, las expresiones regulares se escriben como cadenas, secuencias de caracteres entre comillas ("") o comillas simples (').

Algunos caracteres no se pueden representar directamente en una cadena de R. Estos deben representarse como caracteres especiales, secuencias de caracteres que tienen un significado específico, p. ej.

Especial Carácter	Representa
\\	\
\"	"
\n	nueva línea

Ejecuta **?"'"** para ver una lista completa

Debido a esto, cada vez que aparece un \ en una expresión regular, debe escribirlo como \\ en la cadena que representa la expresión regular.

Usa writeLines() para ver cómo ve R tu cadena después de que se hayan analizado todos los caracteres especiales.

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

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

INTERPRETACIÓN

Los patrones en stringr se interpretan como expresiones regulares. Para cambiar este valor predeterminado, envuelva el patrón en una de las siguientes opciones:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...**)** Modifica una expresión regular para ignorar mayúsculas y minúsculas, hacer coincidir el final de las líneas y el final de las cadenas, permitir comentarios de R dentro de la expresión regular y/o tener . coincidir con todo, incluido \n. str_detect("I", regex("i", TRUE))

fixed() Coincide con bytes sin procesar, pero perderá algunos caracteres que se pueden representar de varias maneras (rápido). str_detect("\u0130", fixed("i"))

coll() Coincide con bytes sin formato y usará reglas de intercalación específicas de la configuración regional para reconocer caracteres que se pueden representar de varias maneras (lentas). str_detect("\u0130", coll("i", TRUE, locale = "tr"))

boundary() Coincide con los límites entre caracteres, line_breaks, oraciones o palabras. str_split(sentences, boundary("word"))

Expresiones Regulares -

Las expresiones regulares, o regexp, son un lenguaje conciso para describir patrones en cadenas.

EMPAREJAR CARACTERES

see <- function(rx) str_view("abc ABC 123\t.!?\\(){}\n", rx)

texto (escriba esto)	regexp (significar esto)	empareja (que coincide con esto)	ejemplo	
	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
\\.	**\.**	.	see("\\.")	abc ABC 123 .!?\(){}
\\!	**\!**	!	see("\\!")	abc ABC 123 .!?\(){}
\\?	**\?**	?	see("\\?")	abc ABC 123 .!?\(){}
****	****	\	see("\\\\")	abc ABC 123 .!?\(){}
\\(	**\(**	(	see("\\(")	abc ABC 123 .!?\(){}
\\)	**\)**	)	see("\\)")	abc ABC 123 .!?\(){}
\\{	**\{**	{	see("\\{")	abc ABC 123 .!?\(){}
\\}	**\}**	}	see("\\}")	abc ABC 123 .!?\(){}
\\n	**\n**	nueva línea	see("\\n")	abc ABC 123 .!?\(){}
\\t	**\t**	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	**\s**	cualquier espacio en blanco (\S para espacios que no son espacios en blanco)	see("\\s")	abc ABC 123 .!?\(){}
\\d	**\d**	cualquier dígito (\D para no dígitos)	see("\\d")	abc ABC 123 .!?\(){}
\\w	**\w**	cualquier carácter de palabra (\W para caracteres que no son de palabra)	see("\\w")	abc ABC 123 .!?\(){}
\\b	**\b**	Límites de palabras	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:] [1]	digitos	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 signos de puntuación	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:]	[1] caracteres de espacio (p. ej.\s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:]	[1] espacio y tabulación (pero no nueva línea)	see("[:blank:]")	abc ABC 123 .!?\(){}
	.	todos los caracteres excepto una nueva línea	see(".")	abc ABC 123 .!?\(){}

[1] Muchas funciones base de R requieren que las clases se envuelvan en un segundo conjunto de [ ], p. ej. [[:d igit:]]

[:space:]
↵ nueva línea

[:blank:]
space
tab

[:graph:]

[:punct:]
. , : ; ? ! / * @ #
- _ " [ ] { } ( )

[:symbol:]
| ` = + ^
~ < > $

[:alnum:]

[:digit:]
0 1 2 3 4 5 6 7 8 9

[:alpha:]

[:lower:]	[:upper:]
a b c d e f	A B C D E F
g h i j k l	G H I J K L
m n o p q r	M N O P Q R
s t u v w x	S T U V W X
y z	Y Z

ALTERNA

alt <- function(rx) str_view("abcde", rx)

regexp	coincide	ejemplo	
ab|d	o	alt("ab|d")	abcde
[abe]	uno de	alt("[abe]")	abcde
[^abe]	cualquiera menos	alt("[^abe]")	abcde
[a-c]	rango	alt("[a-c]")	abcde

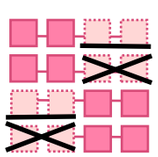
ANCLAJES

anchor <- function(rx) str_view("aaa", rx)

regexp	coincide	ejemplo	
^a	inicio de la cadena	anchor("^a")	aaa
a$	fin de la cadena	anchor("a$")	aaa

BUSCADORES

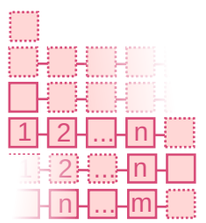
look <- function(rx) str_view("bacad", rx)



regexp	coinciden	ejemplos	
a(?=c)	seguido de	look("a(?=c)")	bacad
a(?!c)	no seguido de	look("a(?!c)")	bacad
(?<=b)a	precedido por	look("(?<=b)a")	bacad
(?<!b)a	no precedido por	look("(?<!b)a")	bacad

CUANTIFICADORES

quant <- function(rx) str_view_all(".a.aa.aaa", rx)



regexp	coincide	ejemplo	
a?	cero o uno	quant("a?")	.a.aa.aaa
a*	cero o más	quant("a*")	.a.aa.aaa
a+	uno o más	quant("a+")	.a.aa.aaa
a{n}	exactamente **n**	quant("a{2}")	.a.aa.aaa
a{n, }	**n** o más	quant("a{2,}")	.a.aa.aaa
a{n, m}	entre **n** y **m**	quant("a{2,4}")	.a.aa.aaa

GRUPOS

ref <- function(rx) str_view("abbaab", rx)

Usar paréntesis para sentar precedentes (orden de evaluación) y crear grupos

regexp	coincide	ejemplo	
(ab|d)e	establece precedente	alt("(ab|d)e")	abcde

Utilice un número de escape para hacer referencia a los grupos de paréntesis que aparecen antes en un patrón y duplicarlos. Refiérase a cada grupo por su orden de aparición

texto (escriba esto)	regexp (singnifica)	coincide (que coincide con este)	ejemplo (el resultado es el mismo que ref("abba"))	
\\1	**\1** (etc.)	first () group, etc.	ref("(a)(b)\\2\\1")	abbaab