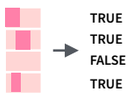
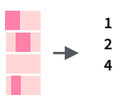
Trabajar con cadenas con stringr : : **GUÍA RÁPIDA**

El paquete **stringr** proporciona un conjunto consistente internamente de herramientas para trabajar con cadenas de caracteres i.e. secuencias de caracteres delimitados por comillas.

Detectar Coincidencias



str_detect(cadena, **patrón**) Detecta la presencia de un patrón o la coincidencia en una cadena. *str_detect(fruit, "a")*



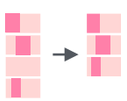
str_which(cadena, **patrón**) Encuentra los índices de las cadenas que contienen un patrón coincidente. *str_which(fruit, "a")*

str_count(cadena, **patrón**) Cuenta el número de coincidencias en una cadena. *str_count(fruit, "a")*

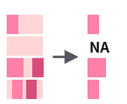
str_locate(cadena, **patrón**) Localiza las posiciones del patrón que coincide en la cadena. También **str_locate_all**. *str_locate(fruit, "a")*

Subconjunto de cadenas

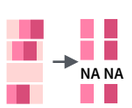
str_sub(cadena, start = 1L, end = -1L) Extrae subcadenas de un vector de caracteres. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*



str_subset(cadena, **patrón**) Devuelve solo las cadenas que contienen un patrón coincidente. *str_subset(fruit, "b")*

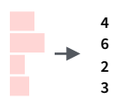


str_extract(cadena, **patrón**) Devuelve el primer patrón encontrado que coincide en cada cadena, como un vector. También **str_extract_all** para devolver cada patrón coincidente. *str_extract(fruit, "[aeiou]")*

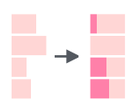


str_match(cadena, **patrón**) Devuelve el primer patrón encontrado que coincide en cada cadena, como una matriz, con una columna para cada una ( ) agrupado por patrón. También **str_match_all**. *str_match(sentences, "(a|the) ([^]+)")*

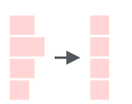
Gestionar Longitudes



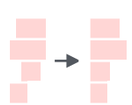
str_length(cadena) Los anchos de las cadenas (i.e. número de puntos de código, suele ser igual al número de caracteres). *str_length(fruit)*



str_pad(cadena, ancho, side = c("left", "right", "both"), pad = " ") Extiende cadenas a un ancho constante. *str_pad(fruit, 17)*

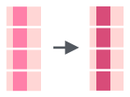


str_trunc(cadena, ancho, side = c("right", "left", "center"), ellipsis = "...") Trunca el ancho de una cadena, eliminando el contenido sobrante. *str_trunc(fruit, 3)*

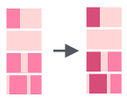


str_trim(cadena, side = c("both", "left", "right")) Elimina los espacios en blanco desde el comienzo y/o el final de una cadena. *str_trim(fruit)*

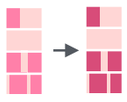
Transformar Cadenas



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



str_replace(cadena, **patrón**, reemplazo) Reemplaza el primer patrón coincidente en cada cadena. *str_replace(fruit, "a", "-")*



str_replace_all(cadena, **patrón**, replacement) Remplaza todos los patrones coincidentes en cada cadena. *str_replace_all(fruit, "a", "-")*

str_to_lower(cadena, locale = "en")[1] Convierte cadenas a minúscula. *str_to_lower(sentences)*

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

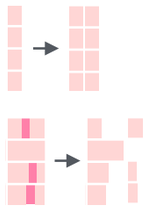
str_to_title(string, locale = "en")[1] Convierte cadenas a título. *str_to_title(sentences)*

Juntar y Separar

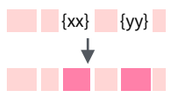
str_c(..., sep = "", collapse = NULL) Une múltiples cadenas en una. *str_c(letters, LETTERS)*

str_c(..., sep = "", **collapse = NULL**) Colapsa un vector de cadenas en una sola cadena. *str_c(letters, collapse = "")*

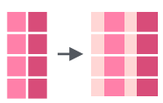
str_dup(cadena, veces) Repite cadenas varias veces. *str_dup(fruit, times = 2)*



str_split_fixed(cadena, **patrón**, n) Divide un vector de cadenas en una matriz de subcadenas (dividiendo en las ocurrencias del patrón de coincidencia). También **str_split** para devolver una lista de subcadenas. *str_split_fixed(fruit, " ", n=2)*

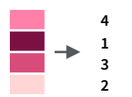


glue::**glue**(..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Crea una cadena a partir de cadenas y {expresión} para evaluar. *glue::glue("Pi is {pi}")*



glue::**glue_data**(.x, ..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Usa un data frame, lista, o entorno para crear una cadena a partir de cadenas y {expresión} para evaluar. *glue::glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

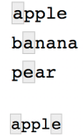
Ordenar Cadenas



str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Devuelve e 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)*

Ayudas



str_conv(cadena, encoding) Sobre escribe el tipo de codificación de una cadena. *str_conv(fruit,"ISO-8859-1")*

str_view(cadena, **patrón**, match = NA) Vista en HTML de la primera coincidencia de expresión regular en cada cadena. *str_view(fruit, "[aeiou]")*

str_view_all(cadena, **patrón**, match = NA) Vista en HTML de todas las coincidencias de la expresión regular. *str_view_all(fruit, "[aeiou]")*

str_wrap(cadena, width = 80, indent = 0, exdent = 0) Envuelve cadenas en párrafos forrajeados de forma atractiva. *str_wrap(sentences, 20)*

[1] 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 sido parseado.*

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
\\	\
\"	"
\n	nueva línea

Escribe **?"'"** 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 Para cambiar este comportamiento, envuelve 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 que los comentarios de R dentro de las regex , y/o tienen . coincide cualquier cosa incluyendo \n. *str_detect("I", regex("i", TRUE))*

fixed() Empareja bytes pero ignorará algunos caracteres que se pueden representar de múltiples formas (rápido). *str_detect("\u0130", fixed("i"))*

coll() Empareja bytes 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() Empareja 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 CHARACTERS

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

Cadena (escribe esto)	regexp (para decir esto)	Coincidencias (que coincide con esto)	Ejemplos	
	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 (retorno)	see("\\n")	abc ABC 123 .!?\(){}
\\t	**\t**	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	**\s**	espacios en blanco (**\S** *para no-blancos*)	see("\\s")	abc ABC 123 .!?\(){}
\\d	**\d**	dígitos (**\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 (i.e. \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 base de R requieren que las clases se envuelvan en un segundo juego de **[ ]**, e.g. **[[:digit:]]**

[:space:]
nueva línea

[:blank:]
espacio
tab

.

[:graph:]

[:punct:]

. , : ; ? ! \ | / ` = * + - ^
_ ~ " ' [ ] { } ( ) < > @ # $

[:alnum:]

[:digit:]

0 1 2 3 4 5 6 7 8 9

[:alpha:]

[:lower:]

a b c d e f
g h i j k l
m n o p q r
s t u v w x
z

[:upper:]

A B C D E F
G H I J K L
M N O P Q R
S T U V W X
Z

ALTERNATIVAS

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

regexp	coincidencias	ejemplo	
ab|d	o	alt("ab|d")	abcde
[abe]	una de	alt("[abe]")	abcde
[^abe]	Excepto	alt("[^abe]")	abcde
[a-c]	Rango	alt("[a-c]")	abcde

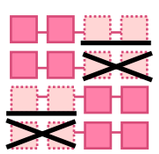
ANCLAS

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

regexp	coincidencias	ejemplo	
^a	comienzo cadena	anchor("^a")	aaa
a$	fin de cadena	anchor("a$")	aaa

MIRAR ALREDEDOR

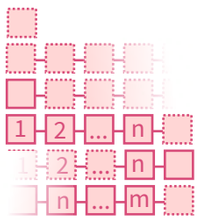
look <- function(rx) str_view_all("bacad", rx)



regexp	coincidencias	ejemplo	
a(?=c)	seguido por	look("a(?=c)")	bacad
a(?!c)	no seguido por	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	coincidencias	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_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")	abcde

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

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