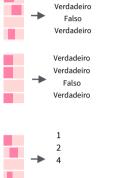
Manipulação de string com stringr:: Folha de Referência

O pacote stringr fornece um conjunto de ferramentas consistentes para trabalhar com cadeia de caracteres (strings), ou seja, sequência de caracteres entre aspas.

Detectar Encontros



3 4

Verdadeiro

str_detect(string, pattern, negate = FALSE)
Detecta a presença de um padrão em uma
string. Ver também str_like(). str_detect(fruit,
"a")

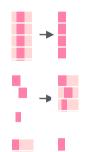
str_starts(string, pattern, negate = FALSE) Detecta a presença de um padrão no início da string. Ver também str_ends(). str_starts(fruit, "a")

str_which(string, pattern, negate = FALSE) Find the indexes of strings that contain a pattern match. str_which(fruit, "a")

str_locate(string, pattern) Localiza a posição que o padrão foi encontrado em uma string. Ver também str_locate_all(). str_locate(fruit, "a")

str_count(string, pattern) Conta quantas vezes o padrão foi encontrado na string. str_count(fruit, "a")

Partes da String



→ NA NA

str_sub(string, start = 1L, end = -1L) Extrair
partes de uma string. str_sub(fruit, 1, 3);
str_sub(fruit, -2)

str_subset(string, pattern, negate = FALSE) Retorna apenas as strings que contém um padrão encontrado. str_subset(fruit, "p")

str_extract(string, pattern) Retorna apenas o primeiro padrão encontrado em cada string, como um vetor. Ver também str_extract_all() para retornar todos os padrões encontrados. str_extract(fruit, "[aeiou]")

str_match(string, pattern) Retorna o primeiro padrão encontrado em cada string como uma matriz com uma coluna para cada grupo do padrão. Ver também str_match_all(). str_match(sentences, "(a|the) ([^ +])")

Gerenciar Comprimento



str_length(string) Retorna o comprimento da string (ou seja, número de pontos de código que em geral é igual ao número de caracteres). str_length(fruit)



str_pad(string, width, side = c("left", "right", "both"), pad = " ") Ajusta a string à um comprimento constante. str_pad(fruit, 17)



str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Trunca a string à um comprimento, colocando o conteúdo restante como elipse (...). str_trunc(sentences, 6)

str_trim(string, side = c("both", "left", "right"))
Remove espaços em branco do início/fim da
string. str_trim(str_pad(fruit, 17))

str_squish(string) Corta espaços em branco das extremidades e remove os espaços em branco duplicados no meio. str_squish(str_pad(fruit, 17, "both"))

Modificar Strings



--|-

UMA STRING

Uma string

Uma string

UMA STRING

str_sub() <- value. Substitui partes da string pelo padrão identificado com str_sub(), com o valor atribuído.

str_sub(fruit, 1, 3) <- "str"

str_replace(string, pattern, replacement) Substitui o primeiro padrão encontrado em cada string. Ver também.

str_replace(fruit, "p", "-")

str_replace_all(string, pattern, replacement) Substitui todos os padrões encontrados em cada string. Ver também str_remove_all(). str_replace_all(fruit, "p", "-")

str_to_lower(string, locale = "en")¹ Converte strings para minúsculo.

str to lower(sentences)

str_to_upper(string, locale = "en")¹
Converte strings para maiúsculo.
str to upper(sentences)

sti_to_abbei(selitelices)

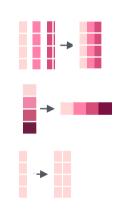
uma string str_tc

Uma String string

str_tc

str_to_title(string, locale = "en")¹ Converte strings para títulos. Ver também str_to_sentence(). str_to_title(sentences)

Juntar e Dividir



{xx} {yy}

str_c(..., sep = "", collapse = NULL) Junta várias
strings em uma única. str_c(letters, LETTERS)

str_flatten(string, collapse = "") Combina em
uma única separada pelo collapse.
str_flatten(fruit, ", ")

str_dup(string, times) Repete a string n vezes. Ver também str_unique() para remover duplicadas. str_dup(fruit, times = 2)

str_split_fixed(string, pattern, n) Divide um vetor de strings em uma matriz de partes da string (dividindo cada ocorrência do padrão encontrado). Ver também str_split() para retornar uma lista de partes da string e str_split_i() para dividir em i partes. str_split_fixed(sentences, " ", n=3)

str_glue(..., .sep = "", .envir = parent.frame())
Cria uma string juntando strings e {expressões}
{expressions}. str_glue("Pi is {pi}")

str_glue_data(.x, ..., .sep = "", .envir =
parent.frame(), .na = "NA") Use um data frame,
lista, ou ambiente para criar uma string de
strings ou {expressões}. str_glue_data(mtcars,
"{rownames(mtcars)} has {hp} hp")

Ordenar Strings



str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹
Retorna um vetor de índices com a ordem de cada vetor de caracteres. fruit[str_order(fruit)]



str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹
Ordena um vetor de caracteres. str sort(fruit)

Auxiliares



str_conv(string, encoding) Sobrescreve a
codificação de uma string. str_conv(fruit,"ISO8859-1")



str_view_all(string, pattern, match = NA) Mostra uma renderização HTML dos padrões regulares (regex) encontrados. Ver str_view() para mostrar apenas o primeiro padrão encontrado. str_view_all(sentences, "[aeiou]")



Esta é uma

sentença longa.

str_equal(x, y, locale = "en", ignore_case = FALSE, ...)¹ Determina se duas strings são equivalentes. str_equal(c("a", "b"), c("a", "c"))

str_wrap(string, width = 80, indent = 0, exdent = 0) Encapsula strings em uma formatação agradável de parágrafos. str_wrap(sentences, 20)

¹ Ver <u>bit.ly/ISO639-1</u> para a lista completa de localizações.



Importante Saber

O argumento de padrões (pattern) no stringr são interpretados como uma expressão regular (regex) depois que qualquer caractere especial seja processado.

No R, você escreve expressões regulares como strings, ou seja, como sequência de caracteres entre aspas duplas ("") ou simples ('').

Alguns caracteres não podem ser representados diretamente como um string no R. Estes devem ser representados como um caractere especial, ou seja, uma sequência de caracteres com significado especial., e.x.

Caractere Especial Representa // nova linha \n

Execute ?"" para ver a lista completa

Devido a isto, sempre que ver \ em uma expressão regular, você deve escrevê-la como \\ na string que representa a expressão.

Use writeLines() para ver como o R vê sua string depois que os caracteres especiais são processados.

writeLines("||.")

writeLines("|| é uma barra invertida") #|éuma barra invertida

INTERPRETADORES

Padrões no stringr são interpretados como regex. Para mudar isto, encapsule o padrão em umas das funções:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...) Modifica o regex para ignorar maiúsculas/minúsculas, encontrar fim de linha como fim da string, permitir comentários do R dentro do regex. Encontrar tudo incluindo \n. str_detect("I", regex("i", TRUE))

fixed() Encontra bytes básicos mas irá perder alguns caracteres que podem estar representados de outras formas (rápido). str detect("\u0130", fixed("i"))

coll() Encontra bytes básicos e usa a localização para reconhecer os caracteres que podem ser representados de várias formas (lento). str_detect("\u0130", coll("i", TRUE, locale = "tr"))

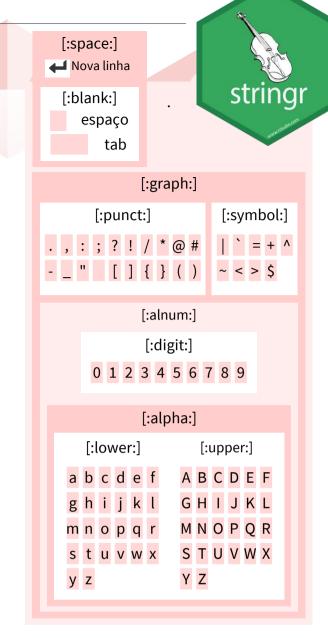
boundary() Encontra fronteiras entre caracteres, quebra de linhas, sentenças ou palavras. str_split(sentences, boundary("word"))

Expressões Regulares - Expressões regulares, ou regexps, é uma linguagem concisa para descrever padões em strings.

ENCONTRAR CARACTERES

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

string (digite)	regexp (para dizer)	encontra (que encontra isto)	exemplo	
	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 .!?\(<mark>)</mark> {}
\\ {	\{	{	see("\\{")	abc ABC 123 .!?\() <mark>{</mark> }
\\}	\}	}	see("\\}")	abc ABC 123 .!?\(){ <mark>}</mark>
\\n	\n	nova linha (return)	see("\\n")	abc ABC 123 .!?\(){}
\\t	\t	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	\s	espaço em branco (\S para não-brancos)	see("\\s")	abc ABC 123 .!?\(){}
\/d	\d	qualquer digito (\D para não-digitos)	see("\\d")	abc ABC 123 .!?\(){}
\\w	\w	qualquer letra (\W fpara não-letras)	see("\\w")	abc ABC 123 .!?\(){}
\\b	\b	limite das palavras	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:]	digitos	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:]	letras	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:]	letras minúsculas	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:]	letras maiúsculas	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:]	letras e números	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:]	pontuação	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:]	letras, números e pontuações	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:]	caractere de espaço (ou seja, \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:]	espaços e tabs (mas não nova linha)	see("[:blank:]")	abc ABC 123 .!?\(){}
		qualquer caractere exceto nova linha	see(".")	abc ABC 123 .!?\(){}



¹ Muitas funções do R básico exigem que as classes sejam colocadas com um segundo colchete, ou seja, [[:digit:]]

ALTERNADORES

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

regexp	encontra	exemplo	
ab d	ou	alt("ab d")	abcde
[abe]	um dos	alt("[abe]")	<mark>ab</mark> cde
[^abe]	tudo menos	alt("[^abe]")	ab <mark>cd</mark> e
[a-c]	range	alt("[a-c]")	abcde

ANCORAGEM

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

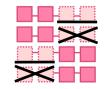
exemplo

^a início da s	
a\$	fim da string

anchor("^a")

aaa anchor("a\$") aaa

look <- function(rx) str_view_all("bacad", rx)</pre>



OLHAR AO REDOR

regexp encontra exemplo a(?=c) seguido por look("a(?=c)") bacad a(?!c) não seguido por look("a(?!c)") bacad (?<=b)aprecedido por look("(?<=b)a") bacad (?<!b)a Não precedido por look("(?<!b)a") bacad

OUANTIFICADORES

regexp a? a+ a{n} **a**{n, } $a\{n, m\}$

matches zero ou um zero or mais um ou mais exatamente n n ou mais

exemplo quant("a?" .a.aa.aaa quant("a*" .a.aa.aaa quant("a+") .a.aa.aaa quant("a{2}") .a.aa.aaa quant("a{2,}") .a.aa.aaa

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

GRUPOS

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

quant("a{2,4}")

Use parênteses para definir precedência (ordem de avaliação) e criar grupos

entre n e m



matches sets precedence

exemplo alt("(ab|d)e")

abcde

.a.aa.aaa

Use um número com dupla barra invertida pafra referenciar ou duplicar grupos identificados anteriormente no padrão. Refencia cada grupo, pela sua ordem de aparição

string	
(digite)	
\\1	

regexp (significa) \1 (etc.)

encontra (que encontra isto) first () group, etc.

exemplo (o resultado é o mesmo que ref("abba")) $ref("(a)(b)\2\1")$ abbaab

