String manipulation with stringr: CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks

Subset Strings





Detect the presence of a pattern match in a string. Also str_like(). str_detect(fruit, "a") str_detect(string, pattern, negate = FALSE)



the beginning of a string. Also str_ends().
str_starts(fruit, "a") Detect the presence of a pattern match at str_starts(string, pattern, negate = FALSE)



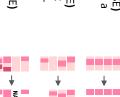
Find the indexes of strings that contain str_which(string, pattern, negate = FALSE) pattern match. str_which(fruit,



₩

of matches in a string. str_count(fruit, "a" str_count(string, pattern) Count the number positions of pattern matches in a string.
Also str_locate_all(). str_locate(fruit, "a")

str_locate(string, pattern) Locate the



Return only the strings that contain a patterr match. str_subset(fruit, "p") str_subset(string, pattern, negate = FALSE)

substrings from a character vector.

str_sub(string, start = 1L, end = -1L) Extract

str_sub(fruit, 1, 3); str_sub(fruit, -2)



str_extract(string, pattern) Return the first
pattern match found in each string, as a vector Also str_extract_all() to return every pattern match. str_extract(fruit, "[aeiou]"



a matrix with a column for each () group in first pattern match found in each string, as str_match(sentences, "(a|the) ([^ +])") pattern. Also **str_match_all(**) str_match(string, pattern) Return the



Manage Lengths



str_pad(string, width, side = c("left", "right",
"both"), pad = "") Pad strings to constant the number of characters). str_length(truit)

number of code points, which generally equals str_length(string) The width of strings (i.e.



width. str_pad(fruit, 17)



of strings, replacing content with ellipsis. str_trunc(sentences, 6) str_trunc(string, width, side = c("right", "left"
"center"), ellipsis = "...") Truncate the width



a string. str_trim(str_pad(fruit, 17)) str_trim(string, side = c("both", "left", "right");
Trim whitespace from the start and/or end of

spaces. str_squish(str_pad(fruit, 17, "both") end and collapse multiple spaces into single **str_squish(**string) Trim whitespace from each

Mutate Strings

Join and Split

Order Strings



identifying the substrings with str_sub() and str_sub() <- value. Replace substrings by str_sub(fruit, 1, 3) <- "str assigning into the results.

str_c(letters, LETTERS)

str_c(..., sep = "", collapse = NULL) Join
multiple strings into a single string.

+

¥

into a single string, separated by collapse. str_flatten(string, collapse = "") Combines

str_flatten(fruit,



str_replace(fruit, "p", string. Also str_remove(). Replace the first matched pattern in each str_replace(string, pattern, replacement)



Also str_remove_all(). str_replace_all(fruit, "p", "-") Replace all matched patterns in each string. str_replace_all(string, pattern, replacement)

₩

times. Also str_unique() to remove duplicates. str_dup(string, times) Repeat strings times

str_dup(fruit, times = 2



str_to_lower(string, locale = "en")¹ Convert strings to lower case. str_to_lower(sentences)



str_to_upper(sentences, Convert strings to upper case str_to_upper(string, locale = "en")1



a string

A String

str_to_title(string, locale = "en")¹ Convert
strings to title case. Also str_to_sentence().
str_to_title(sentences)



and **str_split_n()** to return the nth substring Also **str_split()** to return a list of substrings vector of strings into a matrix of substrings str_split_fixed(string, pattern, n) Split a (splitting at occurrences of a pattern match)



to evaluate. str_glue("Pi is {pi}") Create a string from strings and {expressions **str_glue(...,** .sep = "", ", .envir = parent.frame()



str_glue_data(.x, ..., .sep = "", .envir =
parent.frame(), .na = "NA") Use a data frame,
list, or environment to create a string from

.sep = "", .envir =

strings and {expressions} to evaluate.

"{rownames(mtcars)} has

₩ ₩

> str_order(x, decreasing = FALSE, na_last =
> TRUE, locale = "en", numeric = FALSE, ...)¹ character vector. truit[str_order(truit)] Return the vector of indexes that sorts a



Sort a character vector. str_sort(truit) str_sort(x, decreasing = FALSE, na_last =
TRUE, locale = "en", numeric = FALSE, ...)¹

Helpers

str_conv(string, encoding) Override the encoding of a string. str_conv(fruit,"ISO-8859-1")



Also **str_view()** to see only the first match. View HTML rendering of all regex matches str_view_all(string, pattern, match = NA) aerou



equivalent. str_equal(c("a", "b"), c(FALSE, ...)¹ Determine if two strings are str_equal(x, y, locale = "en", ignore_case =

str_wrap(string, width = 80, indent = 0, paragraphs. str_wrap(sentences, 20) exdent = 0) Wrap strings into nicely formatted

See bit.ly/ISO639-1 for a complete list of locales



Need to Know

have been parsed. regular expressions after any special characters Pattern arguments in stringr are interpreted as

string

MATCH CHARACTERS

sequences of characters surrounded by quotes ("") or single quotes("). In R, you write regular expressions as strings,

have a specific meaning., e.g. special characters, sequences of characters that in an R string . These must be represented as Some characters cannot be represented directly

```
Run?"" to see a complete list
                                                                                                      Special Character
                                                                                                    Represents
                                new line
```

Because of this, whenever a \appears in a regular expression, you must write it as \\in the string that represents the regular expression.

after all special characters have been parsed. Use writeLines() to see how R views your string

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

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

INTERPRETATION

Patterns in stringr are interpreted as regexs. To change this default, wrap the pattern in one of:

AL_I

str_detect("I", regex("i", TRUE)) including \n. within regex's , and/or to have . match everything Modifies a regex to ignore cases, match end of lines as well of end of strings, allow R comments regex(pattern, ignore_case = FALSE, multiline =
FALSE, comments = FALSE, dotall = FALSE, ...)

characters that can be represented in multiple ways (fast). str_detect("\u0130", fixed("i")) fixed() Matches raw bytes but will miss some

AN

specific collation rules to recognize characters that can be represented in multiple ways (slow). str_detect("\u0130",col\("",TRUE, locale = "tr")) coll() Matches raw bytes and will use locale

boundary() Matches boundaries between characters, line_breaks, sentences, or words. str_split(sentences, boundary("word"))

LOOK AROUNDS

a\$

end of string

a(?!c) a(?=c)

not followed by

followed by

regexp

look <- function(rx) str_view

(?<=b)

not preceded by preceded by

Regular Expressions ī

Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

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

										\\b	//w	/\d	l\s	 	//n	=		\leq	=		?	=	=		(type this)
٠	[:blank:]	[:space:]	[:graph:]	[:punct:]	[:alnum:]	[:upper:]	[:lower:]	[:alpha:]	[:digit:]	Ь	\w	ď	15	+	'n	4	*	5	<	=	12	!	<u>, </u>	a (etc.)	(to mean this)
every character except a new line	space and tab (but not new line)	space characters (i.e. \s)	letters, numbers, and punctuation	punctuation	letters and numbers	uppercase letters	lowercase letters	letters	digits	word boundaries	any word character (\ W for non-word chars)	any digit \D for non-digits)	any whitespace (\ S for non-whitespaces)	tab	new line (return)	})			?		•	a (etc.)	(which matches this)
see(":")	see("[:blank:]")	see("[:space:]")	see("[:graph:]")	see("[:punct:]")	see("[:alnum:]")	see("[:upper:]")	see("[:lower:]")	see("[:alpha:]")	see("[:digit:]")	see("\\b")	see("\\w")	see("\\d")	see("\\s")	see("\\t")	see("\\n")	see("\\}")	see("\\{")	see("\\)")	see("\\(")	see("\\\\")	see("\\?")	see("\\!")	see("\\.")	see("a")	-
abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}		abc ABC 123 .!?\(){}	abc ABC 123 .!?\()\{}	abc ABC 123 .!?\(){}	abc ABC 123 .!?\(){}								

[:space:] <u>J</u>

		space	[:blank:]	1 new line
[:gr	tab	асе	<u></u>	ine
[:graph:]				L
			Strii	محرير .

											- 1	•	
٧	<	S	\exists	œ	а						1	•	
1	7	+	_		Ь						=	• •	
		⊏	0			õ		0			-	• •	∓
		<				ĕ		\vdash			_	.~	Ĕ
		5		$\overline{}$		[:lower:]		2				•	[:punct:]
		_		_			ä	ω	Ξ	ä	~	_	☳
							ਰ	4	Ë	Ξ			
-	<	S	<	G	\triangleright		[:alpha:]	5	[:digit:]	[:alnum:]	_	®	
				エ			<u></u>	6 7	_	≕	_	#	
						[:upper:]		8					_
				ے		Ď		9			₹	_	is
				$\overline{}$		<u>::</u>					۸ ۷		[:symbol:]
				_							Ş		귳
											- 0,	>	<u>.</u>

¹ Many	,
base R	
functio	
ns requ	
Many base R functions require classes to be wrapped in a second set of [], e.g. [[:digit:]]	
es to be v	
wrapped	
d in a sec	
cond set	
t of [], e	
;; ===================================	

QUANTIFIERS

regexp

zero or more zero or one matches

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

n or more exactly **n** one or more

quant("a{2,}") quant("a{2}") quant("a+") quant("a*") quant("a?")

.a.aa.aaa .a.aa.aaa .a.aa.aaa .a.aa.aaa .a.aa.aaa

TERNATES		alt <- function	alt <- function(rx) str_view_all("abcde", rx)	ibcde", rx)
	regexp	matches	example	
	ab d	or	alt("ab d")	abcde
	abe	one of	alt("[abe]")	abcde
	[^abe]	anything but	alt("[^abe]")	abcde
	a-c	range	alt("[a-c]")	abcde
ICHORS		anchor <- functi	<pre>anchor <- function(rx) str_view_all("aaa", rx)</pre>	("aaa", rx)
	regexp	matches	example	
	^ a	start of string	anchor("^a")	aaa

look("(? b)a") bacac</th <th>look("a(?!c)") bacad look("(?<=b)a") bacad</th> <th>example look("a(?=c)") bacad</th> <th>rx) str_view_all("bacad", rx)</th> <th>anchor("^a") aaa anchor("a\$") aaa</th> <th>n(rx) str_view_all("aaa", rx) example</th>	look("a(?!c)") bacad look("(?<=b)a") bacad	example look("a(?=c)") bacad	rx) str_view_all("bacad", rx)	anchor("^a") aaa anchor("a\$") aaa	n(rx) str_view_all("aaa", rx) example
			×:		
	string regexp (type this) (to mean this)	Use an escaped number to refer to and duplicate parentheses groearlier in a pattern. Refer to each group by its order of appearance	regexp (ab d)e	JPS arentheses to	nma{n, m}
?		mber to refer Refer to eac	matches sets pro	set precede	a{ n, m }
	matches (which matches this)	to and dupli h group by it:	matches sets precedence	ref <- nt (order of e	between n and m
5111.	example (the resul	cate parent s order of ap	example alt("(ab d)e")	function(rx valuation) a	
rof("(a)(b)\\13\\1")	example (the result is the same as ref("abba"))	Use an escaped number to refer to and duplicate parentheses groups that occur earlier in a pattern. Refer to each group by its order of appearance	(d)e")	ref < function(rx) str_view_all("ab use parentheses to set precedent (order of evaluation) and create groups	quant("a{2,4}
ahhaah	ref("abba"))	s that occur	abc <mark>de</mark>	<pre>ref <- function(rx) str_view_all("abbaab", rx) er of evaluation) and create groups</pre>	quant("a{2,4}") .a.aa.aaa

