Package ‘textclean’

July 23, 2018

Title Text Cleaning Tools

Version 0.9.3

Maintainer Tyler Rinker <tyler.rinker@gmail.com>

Description Tools to clean and process text. Tools are geared at checking for substrings that are not optimal for analysis and replacing or removing them (normalizing) with more analysis friendly substrings (see Sproat, Black, Chen, Kumar, Ostendorf, & Richards (2001) <doi:10.1006/csla.2001.0169>) or extracting them into new variables. For example, emoticons are often used in text but not always easily handled by analysis algorithms. The replace_emoticon() function replaces emoticons with word equivalents.

Depends R (>= 3.4.0)

Imports data.table, english(>= 1.0-2), glue (>= 1.3.0), lexicon (>= 1.0.0), mgsub (>= 1.5.0), qdapRegex, stringi, textshape(>= 1.0.1), utils

Suggests testthat

License GPL-2

LazyData TRUE

RoxygenNote 6.0.1

URL https://github.com/trinker/textclean

BugReports https://github.com/trinker/textclean/issues

Collate 'add_comma_space.R' 'add_missing_endmark.R' 'utils.R' 'replace_html.R' 'check_text_logicals.R' 'check_text.R' 'drop_element.R' 'drop_row.R' 'fgsub.R' 'filter_element.R' 'filter_row.R' 'glue-reexports.R' 'has_endmark.R' 'make_plural.R' 'match_tokens.R' 'mgsub.R' 'replace_contraction.R' 'replace_date.R' 'replace_email.R' 'replace_emoji.R' 'replace_emoticon.R' 'replace_grade.R' 'replace_hash.R' 'replace_incomplete.R' 'replace_internet_slang.R' 'replace_kerning.R' 'replace_money.R' 'replace_names.R' 'replace_non_ascii.R' 'replace_number.R' 'replace_ordinal.R' 'replace_rating.R' 'replace_symbol.R' 'replace_tag.R' 'replace_time.R'

'replace_to.R' 'replace_tokens.R' 'replace_url.R'
'replace_white.R' 'replace_word_elongation.R' 'strip.R'
'sub_holder.R' 'swap.R' 'textclean-package.R'

NeedsCompilation no

Author Tyler Rinker [aut, cre],
ctwheels StackOverflow [ctb]

Repository CRAN

Date/Publication 2018-07-23 16:40:03 UTC

R topics documented:

`add_comma_space` *Ensure Space After Comma*

Description

Adds a space after a comma as `strip` and many other functions may consider a comma separated string as one word (i.e., `"one,two,three"` becomes `"onetwothree"` rather than `"one two three"`).

Usage

```
add_comma_space(x)
```

Arguments

`x`	The text variable.

Value

Returns a vector of strings with commas that have a space after them.

Examples

```
## Not run:
x <- c("the,  dog,went", "I,like,it", "where are you", NA, "why", ",", ",f")
add_comma_space(x)

## End(Not run)
```

add_missing_endmark *Add Missing Endmarks*

Description

Detect missing endmarks and replace with the desired symbol.

Usage

```
add_missing_endmark(x, replacement = "|", endmarks = c("?", ".", "!"), ...)
```

Arguments

x	The text variable.
replacement	Character string equal in length to pattern or of length one which are a replacement for matched pattern.
endmarks	The potential ending punctuation marks.
...	Additional arguments passed to has_endmark.

Value

Returns a vector with missing endmarks added.

Examples

```
x <- c(
    "This in a",
    "I am funny!",
    "An ending of sorts%",
    "What do you want?"
)

add_missing_endmark(x)
```

check_text *Check Text For Potential Problems*

Description

check_text - Uncleaned text may result in errors, warnings, and incorrect results in subsequent analysis. check_text checks text for potential problems and suggests possible fixes. Potential text anomalies that are detected include: factors, missing ending punctuation, empty cells, double punctuation, non-space after comma, no alphabetic characters, non-ASCII, missing value, and potentially misspelled words.

available_check - Provide a data.frame view of all the available checks in the check_text function.

Usage

```
check_text(x, file = NULL, checks = NULL, n = 10, ...)

available_checks()
```

Arguments

`x` The text variable.

`file` A connection, or a character string naming the file to print to. If `NULL` prints to the console. Note that this is assigned as an attribute and passed to `print`.

`checks` A vector of checks to include from `which_are`. If `checks = NULL`, all checks from `which_are` which be used. Note that all meta checks will be conducted (see `which_are` for details on meta checks).

`n` The number of affected elements to print out (the rest are truncated).

`...` ignored.

Value

Returns a list with the following potential text faults report:

- contraction- Text elements that contain contractions
- date- Text elements that contain dates
- digit- Text elements that contain digits/numbers
- email- Text elements that contain email addresses
- emoticon- Text elements that contain emoticons
- empty- Text elements that contain empty text cells (all white space)
- escaped- Text elements that contain escaped back spaced characters
- hash- Text elements that contain Twitter style hash tags (e.g., #rstats)
- html- Text elements that contain HTML markup
- incomplete- Text elements that contain incomplete sentences (e.g., uses ending punctuation like ...)
- kern- Text elements that contain kerning (e.g., 'The B O M B!')
- list_column- Text variable that is a list column
- missing_value- Text elements that contain missing values
- misspelled- Text elements that contain potentially misspelled words
- no_alpha- Text elements that contain elements with no alphabetic (a-z) letters
- no_endmark- Text elements that contain elements with missing ending punctuation
- no_space_after_comma- Text elements that contain commas with no space afterwards
- non_ascii- Text elements that contain non-ASCII text
- non_character- Text variable that is not a character column (likely `factor`)

- non_split_sentence- Text elements that contain unsplit sentences (more than one sentence per element)
- tag- Text elements that contain Twitter style handle tags (e.g., @trinker)
- time- Text elements that contain timestamps
- url- Text elements that contain URLs

Note

The output is a list containing meta checks and elemental checks but prints as a pretty formatted output with potential problem elements, the accompanying text, and possible suggestions to fix the text.

DATA *Fictitious Classroom Dialogue*

Description

A fictitious dataset useful for small demonstrations.

Usage

```
data(DATA)
```

Format

A data frame with 11 rows and 5 variables

Details

- person. Speaker
- sex. Gender
- adult. Dummy coded adult (0-no; 1-yes)
- state. Statement (dialogue)
- code. Dialogue coding scheme

`drop_element` *Filter Elements in a Vetor*

Description

`drop_element` - Filter to drop the matching elements of a vector.

`keep_element` - Filter to keep the matching elements of a vector.

Usage

```
drop_element(x, pattern, regex = TRUE, ...)

drop_element_regex(x, pattern, ...)

drop_element_fixed(x, ...)

keep_element(x, pattern, regex = TRUE, ...)

keep_element_fixed(x, ...)

keep_element_regex(x, pattern, ...)
```

Arguments

`x`	A character vector.
`pattern`	A regex pattern to match for exclusion.
`regex`	logical. If setting this to `TRUE` please use `drop_element_regex` or `keep_element_regex` directly as this will provide better control and optimization.
`...`	Other arguments passed to `grep` if `regex`. If `fixed`, then elements to drop/keep.

Value

Returns a vector with matching elements removed.

Examples

```
x <- c('dog', 'cat', 'bat', 'dingo', 'dragon', 'dino')
drop_element(x, '^d.+?g')
keep_element(x, '^d.+?g')
drop_element(x, 'at$')
drop_element(x, '^d')
drop_element(x, '\\b(dog|cat)\\b')

drop_element_fixed(x, 'bat', 'cat')
drops <- c('bat', 'cat')
drop_element_fixed(x, drops)
```

drop_row *Filter Rows That Contain Markers*

Description

drop_row - Remove rows from a data set that contain a given marker/term.

keep_row - Keep rows from a data set that contain a given marker/term.

drop_empty_row - Removes the empty rows of a data set that are common in reading in data.

drop_NA - Removes the NA rows of a data set.

Usage

```
drop_row(dataframe, column, terms, ...)

keep_row(dataframe, column, terms, ...)

drop_empty_row(dataframe)

drop_NA(dataframe, column = TRUE, ...)
```

Arguments

`dataframe` A dataframe object.

`column` Column name to search for markers/terms.

`terms` The regex terms/markers of the rows that are to be removed from the dataframe.

`...` Other arguments passed to `grepl`.

Value

drop_row - returns a dataframe with the termed/markered rows removed.

drop_empty_row - returns a dataframe with empty rows removed.

drop_NA - returns a dataframe with NA rows removed.

Examples

```
## Not run:
## drop_row EXAMPLE:
drop_row(DATA, "person", c("sam", "greg"))
keep_row(DATA, "person", c("sam", "greg"))
drop_row(DATA, 1, c("sam", "greg"))
drop_row(DATA, "state", c("Comp"))
drop_row(DATA, "state", c("I "))
drop_row(DATA, "state", c("you"), ignore.case=TRUE)

## drop_empty_row EXAMPLE:
(dat <- rbind.data.frame(DATA[, c(1, 4)], matrix(rep(" ", 4),
   ncol =2, dimnames=list(12:13, colnames(DATA)[c(1, 4)]))))
drop_empty_row(dat)

## drop_NA EXAMPLE:
DATA[1:3, ”state”] <- NA
drop_NA(DATA)

## End(Not run)
```

fgsub *Replace a Regex with an Functional Operation on the Regex Match*

Description

This is a stripped down version of `gsubfn` from the **gsubfn** package. It finds a regex match, and then uses a function to operate on these matches and uses them to replace the original matches. Note that the **stringi** packages is used for matching and extracting the regex matches. For more powerful or flexible needs please see the **gsubfn** package.

Usage

```
fgsub(x, pattern, fun, ...)
```

Arguments

`x`	A character vector.
`pattern`	Character string to be matched in the given character vector.
`fun`	A function to operate on the extracted matches.
`...`	ignored.

Value

Returns a vector with the pattern replaced.

See Also

`gsubfn`

Examples

```
## In this example the regex looks for words that contain a lower case letter
## followed by the same letter at least 2 more times.  It then extracts these
## words, splits them appart into letters, reverses the string, pastes them
## back together, wraps them with double angle braces, and then puts them back
## at the original locations.
fgsub(
    x = c(NA, 'df dft sdf', 'sd fdggg sd dfhhh d', 'ddd'),
    pattern = ”\\b\\w*([a-z])(\\1{2,})\\w*\\b”,
    fun = function(x) {
        paste0('<<', paste(rev(strsplit(x, '')[[1]]), collapse =''), '>>')
    }
)

## In this example we extract numbers, strip out non-digits, coerce them to
## numeric, cut them in half, round up to the closest integer, add the commas
## back, and replace back into the original locations.
fgsub(
    x = c(NA, 'I want 32 grapes', 'he wants 4 ice creams',
        'they want 1,234,567 dollars'
    ),
    pattern = "[\\d,]+",
    fun = function(x) {
        prettyNum(
            ceiling(as.numeric(gsub('[^0-9]', '', x))/2),
            big.mark = ','
        )
    }
)

## In this example we extract leading zeros, convert to an equal number of
## spaces.
fgsub(
    x = c(NA, "00:04", "00:08", "00:01", "06:14", "00:02", "00:04"),
    pattern = '^0+',
    fun = function(x) {gsub('0', ' ', x)}
)
```

filter_element *Remove Elements in a Vetor*

Description

Removes the matching elements of a vector.

Usage

```
filter_element(x, pattern, ...)
```

Arguments

x	A character vector.
pattern	A regex pattern to match for exclusion.
...	Other arguments passed to grep.

Value

Returns a vector with matching elements removed.

Examples

```
x <- c('dog', 'cat', 'bat', 'dingo', 'dragon', 'dino')
filter_element(x, '^d.+?g')
filter_element(x, 'at$')
filter_element(x, '^d')
filter_element(x, '\\b(dog|cat)\\b')
```

`filter_row` *Remove Rows That Contain Markers*

Description

`filter_row` - Remove rows from a data set that contain a given marker/term.

`filter_empty_row` - Removes the empty rows of a data set that are common in reading in data.

`filter_NA` - Removes the `NA` rows of a data set.

Usage

```
filter_row(dataframe, column, terms, ...)

filter_empty_row(dataframe)

filter_NA(dataframe, column = TRUE, ...)
```

Arguments

`dataframe`	A dataframe object.
`column`	Column name to search for markers/terms.
`terms`	The regex terms/markers of the rows that are to be removed from the dataframe.
`...`	Other arguments passed to `grepl`.

Value

`filter_row` - returns a dataframe with the termed/markered rows removed.

`filter_empty_row` - returns a dataframe with empty rows removed.

`filter_NA` - returns a dataframe with `NA` rows removed.

Examples

```
## Not run:
## filter_row EXAMPLE:
filter_row(DATA, "person", c("sam", "greg"))
filter_row(DATA, 1, c("sam", "greg"))
filter_row(DATA, "state", c("Comp"))
filter_row(DATA, "state", c("I "))
filter_row(DATA, "state", c("you"), ignore.case=TRUE)
```

```
## filter_empty_row EXAMPLE:
(dat <- rbind.data.frame(DATA[, c(1, 4)], matrix(rep(" ", 4),
   ncol =2, dimnames=list(12:13, colnames(DATA)[c(1, 4)]))))
filter_empty_row(dat)

## filter_NA EXAMPLE:
DATA[1:3, "state"] <- NA
filter_NA(DATA)

## End(Not run)
```

---

has_endmark *Test for Incomplete Sentences*

Description

A logical test of missing sentence ending punctuation.

Usage

```
has_endmark(x, endmarks = c("?", ".", "!"), ...)
```

Arguments

x	A character vector.
endmarks	The potential ending punctuation marks,
...	ignored.

Value

Returns a logical vector.

Examples

```
x <- c(
    "I like it.",
    "Et tu?",
    "Not so much",
    "Oh, I understand.",
    "At 3 p.m., we go",
    NA
)
has_endmark(x)
```

make_plural *Make Plural (or Verb to Singular) Versions of Words*

Description

Add -s, -es, or -ies to words.

Usage

```
make_plural(x, keep.original = FALSE,
  irregular = lexicon::pos_df_irregular_nouns)
```

Arguments

x	A vector of words to make plural.
keep.original	logical. If TRUE the original words are kept in the return vector.
irregular	A data.frame of singular and plural conversions for irregular nouns. The first column should be singular and the second plural form of the irregular noun.

Value

Returns a vector of plural words.

Examples

```
x <- c('fox', 'sky', 'dog', 'church', 'fish', 'miss', 'match', 'deer', 'block')
make_plural(x)
```

match_tokens *Find Tokens that Match a Regex*

Description

Given a text, find all the tokens that match a regex(es). This function is particularly useful with replace_tokens.

Usage

```
match_tokens(x, pattern, ignore.case = TRUE, ...)
```

Arguments

x	A character vector.
pattern	Character string(s) to be matched in the given character vector.
ignore.case	logical. If TRUE the case of the tokens/patterns will be ignored.
...	ignored.

Value

Returns a vector of tokens from a text matching a specific regex pattern.

See Also

replace_tokens

Examples

```
with(DATA, match_tokens(state, c('^li', 'ou')))

with(DATA, match_tokens(state, c('^Th', '^I'), ignore.case = TRUE))
with(DATA, match_tokens(state, c('^Th', '^I'), ignore.case = FALSE))
```

mgsub *Multiple* gsub

Description

mgsub - A wrapper for gsub that takes a vector of search terms and a vector or single value of replacements.

mgsub_fixed - An alias for mgsub.

mgsub_regex - An wrapper for mgsub with fixed = FALSE.

mgsub_regex_safe - An wrapper for mgsub.

Usage

```
mgsub(x, pattern, replacement, leadspace = FALSE, trailspace = FALSE,
  fixed = TRUE, trim = FALSE, order.pattern = fixed, safe = FALSE, ...)

mgsub_fixed(x, pattern, replacement, leadspace = FALSE, trailspace = FALSE,
  fixed = TRUE, trim = FALSE, order.pattern = fixed, safe = FALSE, ...)

mgsub_regex(x, pattern, replacement, leadspace = FALSE, trailspace = FALSE,
  fixed = FALSE, trim = FALSE, order.pattern = fixed, ...)

mgsub_regex_safe(x, pattern, replacement, ...)
```

Arguments

x	A character vector.
pattern	Character string to be matched in the given character vector.
replacement	Character string equal in length to pattern or of length one which are a replacement for matched pattern.
leadspace	logical. If TRUE inserts a leading space in the replacements.

`trailspace` logical. If `TRUE` inserts a trailing space in the replacements.

`fixed` logical. If `TRUE`, pattern is a string to be matched as is. Overrides all conflicting arguments.

`trim` logical. If `TRUE` leading and trailing white spaces are removed and multiple white spaces are reduced to a single white space.

`order.pattern` logical. If `TRUE` and `fixed = TRUE`, the `pattern` string is sorted by number of characters to prevent substrings replacing meta strings (e.g., `pattern = c("the", "then")` resorts to search for "then" first).

`safe` logical. If `TRUE` then the **mgsub** package is used as the backend and performs safe substitutions. The trade-off is that this mode will slow the replacements down considerably.

`...` Additional arguments passed to `gsub`. In `mgsub_regex_safe` this is other arguments passed to `mgsub`.

Value

`mgsub` - Returns a vector with the pattern replaced.

See Also

`replace_tokens` `gsub`

Examples

```
mgsub(DATA$state, c("it's", "I'm"), c("it is", "I am"))
mgsub(DATA$state, "[[:punct:]]", "PUNC", fixed = FALSE)
## Not run:
library(textclean)
hunthou <- replace_number(seq_len(1e5))

textclean::mgsub(
    "'twenty thousand three hundred five' into 20305",
    hunthou,
    seq_len(1e5)
)
## "'20305' into 20305"

## Larger example from: https://stackoverflow.com/q/18332463/1000343
## A slower approach
fivehunthou <- replace_number(seq_len(5e5))

testvect <- c("fifty seven", "four hundred fifty seven",
    "six thousand four hundred fifty seven",
    "forty six thousand four hundred fifty seven",
    "forty six thousand four hundred fifty seven",
    "three hundred forty six thousand four hundred fifty seven"
)

textclean::mgsub(testvect, fivehunthou, seq_len(5e5))
```

```
## Safe substitution: Uses the mgsub package as the backend
dubious_string <- "Dopazamine is a fake chemical"
pattern <- c("dopazamin","do.*ne")
replacement <- c("freakout","metazamine")

mgsub(dubious_string, pattern, replacement, ignore.case = TRUE, fixed = FALSE)
mgsub(dubious_string, pattern, replacement, safe = TRUE, fixed = FALSE)

## End(Not run)
```

print.check_text *Prints a check_text Object*

Description

Prints a check_text object.

Usage

```
## S3 method for class 'check_text'
print(x, include.text = TRUE, file = NULL, n = NULL,
  ...)
```

Arguments

x	The check_text object.
include.text	logical. If TRUE the offending text is printed as well.
file	A connection, or a character string naming the file to print to. If NULL prints to the console.
n	The number of affected elements to print out (the rest are truncated)
...	ignored

print.sub_holder *Prints a sub_holder object*

Description

Prints a sub_holder object

Usage

```
## S3 method for class 'sub_holder'
print(x, ...)
```

Arguments

x	The sub_holder object
...	ignored

`print.which_are_locs` *Prints a which_are_locs Object*

Description

Prints a which_are_locs object

Usage

```
## S3 method for class 'which_are_locs'
print(x, n = 100, file = NULL, ...)
```

Arguments

`x`	A which_are_locs object
`n`	The number of affected elements to print out (the rest are truncated)
`file`	Path to an external file to print to
`...`	ignored.

`replace_contraction` *Replace Contractions*

Description

This function replaces contractions with long form.

Usage

```
replace_contraction(x, contraction.key = lexicon::key_contractions,
  ignore.case = TRUE, ...)
```

Arguments

`x`	The text variable.
`contraction.key`	A two column hash of contractions (column 1) and expanded form replacements (column 2). Default is to use `key_contractions` data set.
`ignore.case`	logical. Should case be ignored?
`...`	ignored.

Value

Returns a vector with contractions replaced.

Examples

```
## Not run:
x <- c("Mr. Jones isn't going.",
    "Check it out what's going on.",
    "He's here but didn't go.",
    "the robot at t.s. wasn't nice",
    "he'd like it if i'd go away")

replace_contraction(x)

## End(Not run)
```

---

replace_date *Replace Dates With Words*

Description

Replaces dates with word equivalents.

Usage

```
replace_date(x, pattern = NULL, replacement = NULL, ...)
```

Arguments

x	The text variable.
pattern	Character date regex string to be matched in the given character vector.
replacement	A function to operate on the extracted matches or a character string which is a replacement for the matched pattern.
...	ignored.

Value

Returns a vector with the pattern replaced.

Examples

```
x <- c(
    NA, '11-16-1980 and 11/16/1980',
    "and 2017-02-08 but then there's 2/8/2017 too"
)

replace_date(x)
replace_date(x, replacement = '<<DATE>>')
```

`replace_email` *Replace Email Addresses*

Description

Replaces email addresses.

Usage

```
replace_email(x, pattern = qdapRegex::grab("rm_email"), replacement = "",
  ...)
```

Arguments

`x` The text variable.

`pattern` Character time regex string to be matched in the given character vector.

`replacement` A function to operate on the extracted matches or a character string which is a replacement for the matched pattern.

`...` ignored.

Value

Returns a vector with email addresses replaced.

Examples

```
x <- c(
    "fred is fred@foo.com and joe is joe@example.com - but @this is a",
    "twitter handle for twit@here.com or foo+bar@google.com/fred@foo.fnord",
    "hello world",
    NA
)

replace_email(x)
replace_email(x, replacement = '<<EMAIL>>')
replace_email(x, replacement = '<a href="mailto:$1" target="_blank">$1</a>')

## Replacement with a function
replace_email(x,
    replacement = function(x){
        sprintf('<a href="mailto:%s" target="_blank">%s</a>', x, x)
    }
)


replace_email(x,
    replacement = function(x){
        gsub('@.+$', ' {{at domain}}', x)
    }
)
```

`replace_emoji` *Replace Emojis With Words/Identifier*

Description

Replaces emojis with word equivalents or a token identifier for use in the **sentimentr** package. Not that this function will coerce the text to ASCII using `Encoding(x) <- "latin1"; iconv(x, "latin1", "ASCII", "byte"` The function `replace_emoji` replaces emojis with text representations while `replace_emoji_identifier` replaces with a unique identifier that corresponds to `lexicon::hash_sentiment_emoji` for use in the **sentimentr** package.

Usage

```
replace_emoji(x, emoji_dt = lexicon::hash_emojis, ...)

replace_emoji_identifier(x, emoji_dt = lexicon::hash_emojis_identifier, ...)
```

Arguments

`x` The text variable.

`emoji_dt` A **data.table** of emojis (ASCII byte representations) and corresponding word/identifier meanings.

`...` Other arguments passed to `.mgsub` (see `textclean:::.mgsub` for details).

Value

Returns a vector of strings with emojis replaced with word equivalents.

Examples

```
fls <- system.file("docs/emoji_sample.txt", package = "textclean")
x <- readLines(fls)[1]
replace_emoji(x)
replace_emoji_identifier(x)
```

replace_emoticon *Replace Emoticons With Words*

Description

Replaces emoticons with word equivalents.

Usage

```
replace_emoticon(x, emoticon_dt = lexicon::hash_emoticons, ...)
```

Arguments

`x`	The text variable.
`emoticon_dt`	A **data.table** of emoticons (graphical representations) and corresponding word meanings.
`...`	Other arguments passed to `.mgsub` (see `textclean:::.mgsub` for details).

Value

Returns a vector of strings with emoticons replaced with word equivalents.

Examples

```
x <- c(
    paste(
        "text from:",
        "http://www.webopedia.com/quick_ref/textmessageabbreviations_02.asp"
    ),
    "... understanding what different characters used in smiley faces mean:",
    "The close bracket represents a sideways smile  )",
    "Add in the colon and you have sideways eyes   :",
    "Put them together to make a smiley face  :)",
    "Use the dash -  to add a nose   :-)",
    paste(
        "Change the colon to a semi-colon ;",
        "and you have a winking face ;)  with a nose  ;-)"
    ),
    paste(
        "Put a zero 0 (halo) on top and now you have a winking,",
        "smiling angel 0;) with a nose 0;-)"
    ),
    "Use the letter 8 in place of the colon for sunglasses 8-)",
    "Use the open bracket ( to turn the smile into a frown  :-("
)

replace_emoticon(x)
```

replace_grade *Replace Grades With Words*

Description

Replaces grades with word equivalents.

Usage

```
replace_grade(x, grade_dt = lexicon::key_grade, ...)
```

Arguments

`x`	The text variable.
`grade_dt`	A **data.table** of grades and corresponding word meanings.
`...`	ignored.

Value

Returns a vector of strings with grades replaced with word equivalents.

Examples

```
(text <- replace_grade(c(
    "I give an A+",
    "He deserves an F",
    "It's C+ work",
    "A poor example deserves a C!"
)))
```

replace_hash *Replace Hashes*

Description

Replaces Twitter style hash tags (e.g., '#rstats').

Usage

```
replace_hash(x, pattern = qdapRegex::grab("rm_hash"), replacement = "", ...)
```

Arguments

`x` The text variable.

`pattern` Character time regex string to be matched in the given character vector.

`replacement` A function to operate on the extracted matches or a character string which is a replacement for the matched pattern.

`...` ignored.

Value

Returns a vector with hashes replaced.

Examples

```
x <- c("@hadley I like #rstats for #ggplot2 work.",
   "Difference between #magrittr and #pipeR, both implement pipeline operators for #rstats:
      http://renkun.me/r/2014/07/26/difference-between-magrittr-and-pipeR.html @timelyportfolio",
   "Slides from great talk: @ramnath_vaidya: Interactive slides from Interactive Visualization
        presentation #user2014. http://ramnathv.github.io/user2014-rcharts/#1"
)

replace_hash(x)
replace_hash(x, replacement = '<<HASH>>')
replace_hash(x, replacement = '$3')

## Replacement with a function
replace_hash(x,
    replacement = function(x){
        paste0('{{', gsub('^#', 'TOPIC: ', x), '}}')
    }
)
```

`replace_html` *Replace HTML Markup*

Description

Replaces HTML markup. The angle braces are removed and the HTML symbol markup is replaced with equivalent symbols.

Usage

```
replace_html(x, symbol = TRUE, ...)
```

Arguments

`x` The text variable.

`symbol` logical. If codeTRUE the symbols are retained with appropriate replacements. If `FALSE` they are removed.

`...` Ignored.

Details

Replacements for symbols are as follows:

html	symbol
&copy;	(c)
&reg;	(r)
&trade;	tm
&ldquo;	"
&rdquo;	"
&lsquo;	’
&rsquo;	’
&bull;	-
&middot;	-
&sdot;	[]
&ndash;	-
&mdash;	-
&cent;	cents
&pound;	pounds
&euro;	euro
&ne;	!=
&frac12;	half
&frac14;	quarter
&frac34;	three fourths
&deg;	degrees
&larr;	<-
&rarr;	->
&hellip;	...
&nbsp;	
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	’
&yen;	yen

Value

Returns a vector with HTML markup replaced.

Examples

```
x <- c(
    "<bold>Random</bold> text with symbols: &nbsp; &lt; &gt; &amp; &quot; &apos;",
    "<p>More text</p> &cent; &pound; &yen; &euro; &copy; &reg;"
)

replace_html(x)
replace_html(x, FALSE)
```

```
replace_white(replace_html(x, FALSE))
```

replace_incomplete *Denote Incomplete End Marks With "|"*

Description

Replaces incomplete sentence end marks (.., ..., .?, ..?, en & em dash etc.) with "|".

Usage

```
replace_incomplete(x, replacement = "|", ...)
```

Arguments

x	The text variable.
replacement	A string to replace incomplete punctuation marks with.
...	ignored.

Value

Returns a text variable (character sting) with incomplete sentence marks (.., ..., .?, ..?, en & em dash etc.) replaced with "|".

Examples

```
x <- c("the...",  "I.?", "you.", "threw..", "we?")
replace_incomplete(x)
replace_incomplete(x, '...')
```

replace_internet_slang *Replace Internet Slang*

Description

Replaces Internet slang.

Usage

```
replace_internet_slang(x, slang = paste0("\\b",
  lexicon::hash_internet_slang[[1]], "\\b"),
  replacement = lexicon::hash_internet_slang[[2]], ignore.case = TRUE, ...)
```

Arguments

`x` The text variable.

`slang` A vector of slang strings to replace.

`replacement` A vector of string to replace slang with.

`ignore.case` logical. If `TRUE` the case of `slang` will be ignored (replacement regardless of case).

`...` Other arguments passed to `replace_tokens`.

Value

Returns a vector with names replaced.

Examples

```
x <- c(
    "Marc the n00b needs to RTFM otherwise ymmv.",
    "TGIF and a big w00t!  The weekend is GR8!",
    "Will will do it",
    'w8...this PITA needs me to say LMGTFY...lmao.',
    NA
)

replace_internet_slang(x)
replace_internet_slang(x, ignore.case = FALSE)
replace_internet_slang(x, replacement = '<<SLANG>>')
replace_internet_slang(
    x,
    replacement = paste0('{{ ', lexicon::hash_internet_slang[[2]], ' }}')
)
```

replace_kern *Replace Kerned (Spaced) with No Space Version*

Description

In typography kerning is the adjustment of spacing. Often, in informal writing, adding manual spaces (a form of kerning) coupled with all capital letters is used for emphasis. This tool looks for 3 or more consecutive capital letters with spaces in between and removes the spaces. Essentially, the capitalized, kerned version is replaced with the word equivalent.

Usage

```
replace_kern(x, ...)
```

Arguments

`x` The text variable.

`...` ignored.

Value

Returns a vector with kern spaces removed.

Author(s)

StackOverflow user @ctwheels

References

https://stackoverflow.com/a/47438305/1000343

Examples

```
x <- c(
    "Welcome to A I: the best W O R L D!",
    "Hi I R is the B O M B for sure: we A G R E E indeed.",
    "A sort C A T indeed!",
    NA
)

replace_kern(x)
```

replace_money *Replace Money With Words*

Description

Replaces money with word equivalents.

Usage

```
replace_money(x, pattern = "(-?)([$])([0-9,]+)(\\.\\d{2})?",
  replacement = NULL, ...)
```

Arguments

`x` The text variable.

`pattern` Character money regex string to be matched in the given character vector.

`replacement` A function to operate on the extracted matches or a character string which is a replacement for the matched pattern.

`...` ignored.

Value

Returns a vector with the pattern replaced.

Examples

```
x <- c(
    NA,
    '$3.16 into "three dollars, sixteen cents"',
    "-$20,333.18 too", 'fff'
)

replace_money(x)
replace_money(x, replacement = '<<MONEY>>')
```

---

replace_names *Replace First/Last Names*

Description

Replaces first/last names.

Usage

```
replace_names(x, names = textclean::drop_element(gsub("(^.)(.*)",
  "\\U\\1\\L\\2", c(lexicon::freq_last_names[[1]],
  lexicon::common_names), perl = TRUE), "^([AIU]n|[TSD]o|H[ea]Pa|Oh)$"),
  replacement = "", ...)
```

Arguments

`x`	The text variable.
`names`	A vector of names to replace. This may be made more custom through a vector provided from a named entity extractor.
`replacement`	A string to replace names with.
`...`	Other arguments passed to `replace_tokens`.

Value

Returns a vector with names replaced.

Examples

```
x <- c(
    "Mary Smith is not here",
    "Karen is not a nice person",
    "Will will do it",
    NA
)

replace_names(x)
replace_names(x, replacement = '<<NAME>>')
```

replace_non_ascii *Replace Common Non-ASCII Characters*

Description

`replace_non_ascii` - Replaces common non-ASCII characters.

`replace_curly_quote` - Replaces curly single and double quotes. This provides a subset of functionality found in `replace_non_ascii` specific to quotes.

Usage

```
replace_non_ascii(x, replacement = "", remove.nonconverted = TRUE, ...)

replace_curly_quote(x, ...)
```

Arguments

`x` The text variable.

`replacement` Character string equal in length to pattern or of length one which are a replacement for matched pattern.

`remove.nonconverted` logical. If `TRUE` unmapped encodings are deleted from the string.

`...` ignored.

Value

Returns a text variable (character sting) with non-ASCII characters replaced.

Examples

```
x <- c(
    "Hello World", "6 Ekstr\xf8m", "J\xf6reskog", "bi\xdfchen Z\xfcrcher",
    'This is a \xA9 but not a \xAE', '6 \xF7 2 = 3',
    'fractions \xBC, \xBD, \xBE', 'cows go \xB5', '30\xA2'
)
Encoding(x) <- "latin1"
x

replace_non_ascii(x)
replace_non_ascii(x, remove.nonconverted = FALSE)

z <- '\x95He said, \x93Gross, I am going to!\x94'
Encoding(z) <- "latin1"
z

replace_curly_quote(z)
replace_non_ascii(z)
```

replace_number *Replace Numbers With Text Representation*

Description

Replaces numeric represented numbers with words (e.g., 1001 becomes one thousand one).

Usage

```
replace_number(x, num.paste = FALSE, remove = FALSE, ...)
```

Arguments

x	The text variable.
num.paste	logical. If FALSE the elements of larger numbers are separated with spaces. If TRUE the elements will be joined without spaces.
remove	logical. If TRUE numbers are removed from the text.
...	Other arguments passed to as.english

Value

Returns a vector with numbers replaced.

Note

The user may want to use replace_ordinal first to remove ordinal number notation. For example replace_number would turn "21st" into "twenty onest", whereas replace_ordinal would generate "twenty first".

References

Fox, J. (2005). Programmer's niche: How do you spell that number? R News. Vol. 5(1), pp. 51-55.

Examples

```
x <- c(
    NA,
    'then .456 good',
    'none',
    "I like 346,457 ice cream cones.",
    "I like 123456789 cashes.",
    "They are 99 percent good and 45678.2345667"
)
replace_number(x)
replace_number(x, num.paste = TRUE)
replace_number(x, remove=TRUE)
## Not run:
library(textclean)
```

```
hunthou <- replace_number(seq_len(1e5))

textclean::mgsub(
    "'twenty thousand three hundred five' into 20305",
    hunthou,
    seq_len(1e5)
)
## "'20305' into 20305"

## Larger example from: https://stackoverflow.com/q/18332463/1000343
## A slower approach
fivehunthou <- replace_number(seq_len(5e5))

testvect <- c("fifty seven", "four hundred fifty seven",
    "six thousand four hundred fifty seven",
    "forty six thousand four hundred fifty seven",
    "forty six thousand four hundred fifty seven",
    "three hundred forty six thousand four hundred fifty seven"
)

textclean::mgsub(testvect, fivehunthou, seq_len(5e5))

## End(Not run)
```

replace_ordinal *Replace Mixed Ordinal Numbers With Text Representation*

Description

Replaces mixed text/numeric represented ordinal numbers with words (e.g., "1st" becomes "first").

Usage

```
replace_ordinal(x, num.paste = FALSE, remove = FALSE, ...)
```

Arguments

x	The text variable.
num.paste	logical. If TRUE a the elements of larger numbers are separated with spaces. If FALSE the elements will be joined without spaces.
remove	logical. If TRUE ordinal numbers are removed from the text.
...	ignored.

Note

Currently only implemented for ordinal values 1 through 100

Examples

```
x <- c(
    "I like the 1st one not the 22nd one.",
    "For the 100th time stop!"
)
replace_ordinal(x)
replace_ordinal(x, TRUE)
replace_ordinal(x, remove = TRUE)
replace_number(replace_ordinal("I like the 1st 1 not the 22nd 1."))
```

replace_rating *Replace Ratings With Words*

Description

Replaces ratings with word equivalents.

Usage

```
replace_rating(x, rating_dt = lexicon::key_rating, ...)
```

Arguments

x	The text variable.
rating_dt	A **data.table** of ratings and corresponding word meanings.
...	ignored.

Value

Returns a vector of strings with ratings replaced with word equivalents.

Examples

```
x <- c("This place receives 5 stars for their APPETIZERS!!!",
     "Four stars for the food & the guy in the blue shirt for his great vibe!",
     "10 out of 10 for both the movie and trilogy.",
     "* Both the Hot & Sour & the Egg Flower Soups were absolutely 5 Stars!",
     "For service, I give them no stars.", "This place deserves no stars.",
     "10 out of 10 stars.",
     "My rating: just 3 out of 10.",
     "If there were zero stars I would give it zero stars.",
     "Rating: 1 out of 10.",
     "I gave it 5 stars because of the sound quality.",
     "If it were possible to give them 0/10, they'd have it."
)

replace_rating(x)
```

replace_symbol *Replace Symbols With Word Equivalents*

Description

This function replaces symbols with word equivalents (e.g., @ becomes "at".

Usage

```
replace_symbol(x, dollar = TRUE, percent = TRUE, pound = TRUE,
  at = TRUE, and = TRUE, with = TRUE, ...)
```

Arguments

x	A character vector.
dollar	logical. If TRUE replaces dollar sign ($) with "dollar".
percent	logical. If TRUE replaces percent sign (%) with "percent".
pound	logical. If TRUE replaces pound sign (#) with "number".
at	logical. If TRUE replaces at sign (@) with "at".
and	logical. If TRUE replaces and sign (&) with "and".
with	logical. If TRUE replaces with sign (w/) with "with".
...	ignored.

Value

Returns a character vector with symbols replaced..

Examples

```
x <- c("I am @ Jon's & Jim's w/ Marry",
    "I owe $41 for food",
    "two is 10% of a #"
)
replace_symbol(x)
```

replace_tag *Replace Handle Tags*

Description

Replaces Twitter style handle tags (e.g., '@trinker').

Usage

```
replace_tag(x, pattern = qdapRegex::grab("rm_tag"), replacement = "", ...)
```

Arguments

x	The text variable.
pattern	Character time regex string to be matched in the given character vector.
replacement	A function to operate on the extracted matches or a character string which is a replacement for the matched pattern.
...	ignored.

Value

Returns a vector with tags replaced.

Examples

```
x <- c("@hadley I like #rstats for #ggplot2 work.",
   "Difference between #magrittr and #pipeR, both implement pipeline operators for #rstats:
      http://renkun.me/r/2014/07/26/difference-between-magrittr-and-pipeR.html @timelyportfolio",
   "Slides from great talk: @ramnath_vaidya: Interactive slides from Interactive Visualization
        presentation #user2014. http://ramnathv.github.io/user2014-rcharts/#1"
)

replace_tag(x)
replace_tag(x, replacement = '<<TAG>>')
replace_tag(x, replacement = '$3')

## Replacement with a function
replace_tag(x,
    replacement = function(x){
        gsub('@', ' <<TO>> ', x)
    }
)
```

replace_time *Replace Time Stamps With Words*

Description

Replaces time stamps with word equivalents.

Usage

```
replace_time(x,
  pattern = "(2[0-3]|[01]?[0-9]):([0-5][0-9])[.:]?([0-5]?[0-9])?",
  replacement = NULL, ...)
```

Arguments

x	The text variable.
pattern	Character time regex string to be matched in the given character vector.
replacement	A function to operate on the extracted matches or a character string which is a replacement for the matched pattern.
...	ignored.

Value

Returns a vector with the pattern replaced.

Examples

```
x <- c(
    NA, '12:47 to "twelve forty-seven" and also 8:35:02',
    'what about 14:24.5', 'And then 99:99:99?'
)

## Textual: Word version
replace_time(x)

## Normalization: <<TIME>>
replace_time(x, replacement = '<<TIME>>')

## Normalization: hh:mm:ss or hh:mm
replace_time(x, replacement = function(y){
        z <- unlist(strsplit(y, '[:.]'))
        z[1] <- 'hh'
        z[2] <- 'mm'
        if(!is.na(z[3])) z[3] <- 'ss'
        glue_collapse(z, ':')
    }
)
```

```
## Textual: Word version (forced seconds)
replace_time(x, replacement = function(y){
        z <- replace_number(unlist(strsplit(y, '[:.]')))
        z[3] <- paste0('and ', ifelse(is.na(z[3]), '0', z[3]), ' seconds')
        paste(z, collapse = ' ')
    }
)

## Normalization: hh:mm:ss
replace_time(x, replacement = function(y){
        z <- unlist(strsplit(y, '[:.]'))
        z[1] <- 'hh'
        z[2] <- 'mm'
        z[3] <- 'ss'
        glue_collapse(z, ':')
    }
)
```

replace_to *Grab Begin/End of String to/from Character*

Description

`replace_to` - Grab from beginning of string to a character(s).

`replace_from` - Grab from character(s) to end of string.

Usage

```
replace_to(x, char = " ", n = 1, include = FALSE, ...)

replace_from(x, char = " ", n = 1, include = FALSE, ...)
```

Arguments

`x` A character string

`char` The character from which to grab until/from.

`n` Number of times the character appears before the grab.

`include` logical. If `TRUE` includes the character in the grab.

`...` ignored.

Value

returns a vector of text with begin/end of string to/from character removed.

Author(s)

Josh O'Brien and Tyler Rinker <tyler.rinker@gmail.com>.

References

http://stackoverflow.com/q/15909626/1000343

Examples

```
## Not run:
x <- c("a_b_c_d", "1_2_3_4", "<_?_._:")
replace_to(x, "_")
replace_to(x, "_", 2)
replace_to(x, "_", 3)
replace_to(x, "_", 4)
replace_to(x, "_", 3, include=TRUE)

replace_from(x, "_")
replace_from(x, "_", 2)
replace_from(x, "_", 3)
replace_from(x, "_", 4)
replace_from(x, "_", 3, include=TRUE)

x2 <- gsub("_", " ", x)
replace_from(x2, " ", 2)
replace_to(x2, " ", 2)

x3 <- gsub("_", "\^", x)
replace_from(x3, "^", 2)
replace_to(x3, "^", 2)

x4 <- c("_a_b", "a__b")
replace_from(x4, "_", 1)
replace_to(x4, "_", 1)

## End(Not run)
```

replace_tokens *Replace Tokens*

Description

Replace tokens with a single substring. This is much faster than mgsub if one wants to replace fixed tokens with a single value or remove them all together. This can be useful for quickly replacing tokens like names in string with a single value in order to reduce noise.

Usage

```
replace_tokens(x, tokens, replacement = NULL, ignore.case = FALSE, ...)
```

Arguments

x	A character vector.
tokens	A vector of token to be replaced.
replacement	A single character string to replace the tokens with. The default, NULL, replaces the tokens with nothing.
ignore.case	logical. If TRUE the case of the tokens will be ignored.
...	ignored.

Value

Returns a vector of strings with tokens replaced.

Note

The function splits the string apart into tokens for speed optimization. After the replacement occurs the strings are pasted back together. The strings are not guaranteed to retain exact spacing of the original.

See Also

mgsub, match_tokens

Examples

```
replace_tokens(DATA$state, c('No', 'what', "it's"))
replace_tokens(DATA$state, c('No', 'what', "it's"), "<<TOKEN>>")
replace_tokens(
    DATA$state,
    c('No', 'what', "it's"),
    "<<TOKEN>>",
    ignore.case = TRUE
)

## Not run:
## Now let's see the speed
## Set up data
library(textshape)
data(hamlet)
set.seed(11)
tokens <- sample(unique(unlist(split_token(hamlet$dialogue))), 2000)

tic <- Sys.time()
head(replace_tokens(hamlet$dialogue, tokens))
(toc <- Sys.time() - tic)


tic <- Sys.time()
head(mgsub(hamlet$dialogue, tokens, ""))
(toc <- Sys.time() - tic)
```

```
## Amp it up 20x more data
tic <- Sys.time()
head(replace_tokens(rep(hamlet$dialogue, 20), tokens))
(toc <- Sys.time() - tic)

## Replace names example

library(lexicon)
library(textshape)
nms <- gsub("(^.)(.*)", "\\U\\1\\L\\2", common_names, perl = TRUE)
x <- split_portion(
    sample(c(sample(grady_augmented, 5000), sample(nms, 10000, TRUE))),
    n.words = 12
)
x$text.var <- paste0(
    x$text.var,
    sample(c('.', '!', '?'), length(x$text.var), TRUE)
 )
replace_tokens(x$text.var, nms, 'NAME')

## End(Not run)
```

replace_url *Replace URLs*

Description

Replaces URLs.

Usage

```
replace_url(x, pattern = qdapRegex::grab("rm_url"), replacement = "", ...)
```

Arguments

x	The text variable.
pattern	Character time regex string to be matched in the given character vector.
replacement	A function to operate on the extracted matches or a character string which is a replacement for the matched pattern.
...	ignored.

Value

Returns a vector with URLs replaced.

Examples

```
x <- c("@hadley I like #rstats for #ggplot2 work. ftp://cran.r-project.org/incoming/",
   "Difference between #magrittr and #pipeR, both implement pipeline operators for #rstats:
      http://renkun.me/r/2014/07/26/difference-between-magrittr-and-pipeR.html @timelyportfolio",
   "Slides from great talk: @ramnath_vaidya: Interactive slides from Interactive Visualization
        presentation #user2014. https://ramnathv.github.io/user2014-rcharts/#1",
    NA
)

replace_url(x)
replace_url(x, replacement = '<<URL>>')

## Not run:
## Replacement with a function
library(urltools)
replace_url(x,
    replacement = function(x){
        sprintf('{{%s}}', urltools::url_parse(x)$domain)
    }
)

## End(Not run)
```

replace_white *Remove Escaped Characters*

Description

Pre-process data to replace one or more white space character with a single space.

Usage

```
replace_white(x, ...)
```

Arguments

`x`	The character vector.
`...`	ignored.

Value

Returns a vector of character strings with escaped characters removed.

Examples

```
x <- "I go \r
    to   the \tnext line"
x
replace_white(x)
```

replace_word_elongation *Replace Word Elongations*

Description

In informal writing people may use a form of text embellishment to emphasize or alter word meanings called elongation (a.k.a. "word lengthening"). For example, the use of "Whyyyyy" conveys frustration. Other times the usage may be to be more sexy (e.g., "Heyyyy there"). Other times it may be used for emphasis (e.g., "This is so gooood"). This function uses an augmented form of Armstrong & Fogarty's (2007) algorithm. The algorithm first attempts to replace the elongation with known semantic replacements (optional; default is `FALSE`). After this the algorithm locates all places were the same letter (case insensitive) appears 3 times consecutively. These elements are then further processed. The matches are replaced via `fgsub` by first taking the elongation to it's canonical form (drop all > 1 consecutive letters to a single letter) and then replacing with the most common word used in 2008 in Google's ngram data set that takes the canonical form. If the canonical form is not found in the Google data set then the canonical form is used as the replacement.

Usage

```
replace_word_elongation(x, impart.meaning = FALSE, ...)
```

Arguments

`x`	The text variable.
`impart.meaning`	logical. If `TRUE`, known elongation semantics are used as replacements (see `textclean::meaning_elongations` for known elongation semantics and replacements).
`...`	ignored.

Value

Returns a vector with word elongations replaced.

References

Armstrong, D. B., Fogarty, G. J., & Dingsdag, D. (2007). Scales measuring characteristics of small business information systems. Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (pp. 562-570). Edinburgh, Scotland. Retrieved from http://www.aclweb.org/anthology/D11-1052

http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

https://www.theatlantic.com/magazine/archive/2013/03/dragging-it-out/309220

https://english.stackexchange.com/questions/189517/is-there-a-name-term-for-multiplied-vowels

Examples

```
x <- c('look', 'noooooo!', 'real coooool!', "it's sooo goooood", 'fsdfds',
    'fdddf', 'as', "aaaahahahahaha", "aabbccxccbbaa", 'I said heyyy!',
    "I'm liiiike whyyyyy me?", "Wwwhhatttt!", NA)

replace_word_elongation(x)
replace_word_elongation(x, impart.meaning = TRUE)
```

strip *Strip Text*

Description

Strip text of unwanted characters.

`strip.character` - factor method for `strip`.

`strip.factor` - factor method for `strip`.

`strip.default` - factor method for `strip`.

`strip.list` - factor method for `strip`.

Usage

```
strip(x, char.keep = "~~", digit.remove = TRUE, apostrophe.remove = FALSE,
  lower.case = TRUE)

## S3 method for class 'character'
strip(x, char.keep = "~~", digit.remove = TRUE,
  apostrophe.remove = FALSE, lower.case = TRUE)

## S3 method for class 'factor'
strip(x, char.keep = "~~", digit.remove = TRUE,
  apostrophe.remove = TRUE, lower.case = TRUE)

## Default S3 method:
strip(x, char.keep = "~~", digit.remove = TRUE,
  apostrophe.remove = TRUE, lower.case = TRUE)

## S3 method for class 'list'
strip(x, char.keep = "~~", digit.remove = TRUE,
  apostrophe.remove = TRUE, lower.case = TRUE)
```

Arguments

`x` The text variable.

`char.keep` A character vector of symbols (i.e., punctuation) that strip should keep. The default is to strip every symbol except apostrophes and a double tilde "~~". The double tilde "~~" is included for a convenient means of keeping word groups together in functions that split text apart based on spaces. To remove double tildes "~~" set char.keep to NULL.

`digit.remove` logical. If TRUE strips digits from the text.

`apostrophe.remove` logical. If TRUE removes apostrophes from the output.

`lower.case` logical. If TRUE forces all alpha characters to lower case.

Value

Returns a vector of text that has been stripped of unwanted characters.

Examples

```
## Not run:
DATA$state #no strip applied
strip(DATA$state)
strip(DATA$state, apostrophe.remove=TRUE)
strip(DATA$state, char.keep = c("?", "."))

## End(Not run)
```

sub_holder *Hold the Place of Characters Prior to Subbing*

Description

This function holds the place for particular character values, allowing the user to manipulate the vector and then revert the place holders back to the original values.

Usage

```
sub_holder(x, pattern, alpha.type = TRUE, holder.prefix = "zzzplaceholder",
  holder.suffix = "zzz", ...)
```

Arguments

`x` A character vector.

`pattern` Character string to be matched in the given character vector.

`alpha.type` logical. If TRUE alpha (lower case letters) are used for the key. If FALSE numbers are used as the key.

`holder.prefix` The prefix to use before the alpha key in the palce holder when alpha.type = TRUE; this ensures uniqueness.

`holder.suffix` The suffix to use after the alpha key in the palce holder when alpha.type = TRUE; this ensures uniqueness.

`...` Additional arguments passed to gsub.

Value

Returns a list with the following:

`output` keyed place holder character vector

`unhold` A function used to revert back to the original values

Note

The `unhold` function for `sub_holder` will only work on keys that have not been disturbed by subsequent alterations. The key follows the pattern of holder.prefix (‘zzzplaceholder‘) followed by lower case letter keys followed by holder.suffix (‘zzz‘) when `alpha.type = TRUE`, otherwise the holder is numeric.

Examples

```
## `alpha.type` as TRUE
library(lexicon); library(textshape)
(fake_dat <- paste(hash_emoticons[1:11, 1, with=FALSE][[1]], DATA$state))
(m <- sub_holder(fake_dat, hash_emoticons[[1]]))
m$unhold(strip(m$output))

## `alpha.type` as FALSE (numeric keys)
vowels <- LETTERS[c(1, 5, 9, 15, 21)]
(m2 <- sub_holder(toupper(DATA$state), vowels, alpha.type = FALSE))
m2$unhold(gsub("[^0-9]", "", m2$output))
mtabulate(strsplit(m2$unhold(gsub("[^0-9]", "", m2$output)), ""))
```

`swap` *Swap Two Patterns Simultaneously*

Description

Swap pattern x for pattern y and pattern y for pattern x in one fell swoop.

Usage

```
swap(x, pattern1, pattern2, ...)
```

Arguments

`x` A text variable.

`pattern1` Character string to be matched in the given character vector. This will be replaced by `pattern2`.

`pattern2` Character string to be matched in the given character vector. This will be replaced by `pattern1`.

`...` ignored.

Value

Returns a vector with patterns 1 & 2 swapped.

Examples

```
x <- c("hash_abbreviation", "hash_contractions", "hash_grade", "key_emoticons",
    "key_power", "key_sentiment", "key_sentiment_nrc", "key_strength",
    "key_syllable", "key_valence_shifters")

x
swap(x, 'hash_', 'key_')
```

`textclean` *Text Cleaning Tools*

Description

Tools to clean and process text.

`which_are` *Detect/Locate Potential Non-Normalized Text*

Description

Detect/Locate potential issues with text data. This family of functions generates a list of detections/location functions that can be accessed via the dollar sign or square bracket operators. Accessible functions include:

Usage

```
which_are()

is_it()
```

Details

contraction Contains contractions

date Contains dates

digit Contains digits

email Contains email addresses

emoticon Contains emoticons

empty Contains just white space

escaped Contains escaped backslash character

hash Contains Twitter style hash tags

html Contains html mark-up

incomplete Contains incomplete sentences (e.g., ends with ...)

kern Contains kerning (e.g. "The B O M B!")

list_column Is a list of atomic vectors (Not provided by `which_are`))

misspelled Contains potentially misspelled words

no_endmark Contains a sentence with no ending punctuation

no_space_after_comma Contains commas with no space after them

non_ascii Contains non-ASCII characters

non_character Is a non-character vector (Not provided by `which_are`))

non_split_sentence Contains non split sentences

tag Contains a Twitter style handle used to tag others (use of the at symbol)

time Contains a time stamp

url Contains a URL

The functions above that have a description starting with 'is' rather than 'contains' are meta functions that describe the attribute of the column/vector being passed rather than attributes about the individual elements of the column/vector. The meta functions will return a logical of length one and are not available under `which_are`.

Value

`which_are` returns an environment of functions that can be used to locate and return the integer locations of the particular non-normalized text named by the function.

`is_it` returns an environment of functions that can be used to detect and return a logical atomic vector of equal length to the input vector (except for meta functions) of the particular non-normalized text named by the function.

Examples

```
wa <- which_are()
it <- is_it()
wa$digit(c('The dog',  "I like 2", NA))
it$digit(c('The dog',  "I like 2", NA))

is_it()$list_column(c('the dog', 'ate the chicken'))
```

Index