Package 'matchmaker'

July 22, 2025

Title Flexible Dictionary-Based Cleaning

Version 0.1.1

Description Provides flexible dictionary-based
cleaning that allows users to specify implicit and explicit missing data,
regular expressions for both data and columns, and global matches, while
respecting ordering of factors. This package is part of the 'RECON'
(<https://www.repidemicsconsortium.org/>) toolkit for outbreak analysis.

URL https://www.repidemicsconsortium.org/matchmaker,
https://github.com/reconhub/matchmaker

License GPL-3

Suggests testthat (>= 2.1.0), covr, knitr, rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Imports rlang, forcats, cli

VignetteBuilder knitr

NeedsCompilation no

Author Zhian N. Kamvar [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1458-7108>),
Thibaut Jombart [ctb],
Patrick Barks [ctb]

Maintainer Zhian N. Kamvar <zkamvar@gmail.com>

Repository CRAN

Date/Publication 2020-02-21 19:00:02 UTC

Contents

matchmaker_example *show the path to a matchmaker example file*

Description

show the path to a matchmaker example file

Usage

```
matchmaker_example(name = NULL)
```

Arguments

name	the name of a matchmaker example file

Value

a path to a matchmaker example file

Author(s)

Zhian N. Kamvar

Examples

```
matchmaker_example() # list all of the example files

# read in example spelling dictionary
sd <- matchmaker_example("spelling-dictionary.csv")
read.csv(sd, stringsAsFactors = FALSE)

# read in example coded data
coded_data <- matchmaker_example("coded-data.csv")
coded_data <- read.csv(coded_data, stringsAsFactors = FALSE)
str(coded_data)
coded_data$date <- as.Date(coded_data$date)
```

match_df *Check and clean spelling or codes of multiple variables in a data frame*

Description

This function allows you to clean your data according to pre-defined rules encapsulated in either a data frame or list of data frames. It has application for addressing mis-spellings and recoding variables (e.g. from electronic survey data).

Usage

```
match_df(
  x = data.frame(),
  dictionary = list(),
  from = 1,
  to = 2,
  by = 3,
  order = NULL,
  warn = FALSE
)
```

Arguments

x	a character or factor vector
dictionary	a data frame or named list of data frames with at least two columns defining the word list to be used. If this is a data frame, a third column must be present to split the dictionary by column in x (see by).
from	a column name or position defining words or keys to be replaced
to	a column name or position defining replacement values
by	character or integer. If dictionary is a data frame, then this column in defines the columns in x corresponding to each section of the dictionary data frame. This defaults to 3, indicating the third column is to be used.
order	a character the column to be used for sorting the values in each data frame. If the incoming variables are factors, this determines how the resulting factors will be sorted.
warn	if TRUE, warnings and errors from match_vec() will be shown as a single warning. Defaults to FALSE, which shows nothing.

Details

By default, this applies the function match_vec() to all columns specified by the column names listed in by, or, if a global dictionary is used, this includes all character and factor columns as well.

by column:

Spelling variables within dictionary represent keys that you want to match to column names in x (the data set). These are expected to match exactly with the exception of two reserved keywords that starts with a full stop:

- .regex [pattern]: any column whose name is matched by [pattern]. The [pattern] should be an unquoted, valid, PERL-flavored regular expression.
- .global: any column (see Section *Global dictionary*)

Global dictionary:

A global dictionary is a set of definitions applied to all valid columns of x indiscriminantly.

- **.global keyword in** by: If you want to apply a set of definitions to all valid columns in addition to specified columns, then you can include a `.global` group in the by column of your `dictionary` data frame. This is useful for setting up a dictionary of common spelling errors. *NOTE: specific variable definitions will override global defintions.* For example: if you have a column for cardinal directions and a definiton for `N = North`, then the global variable `N = no` will not override that. See Example.
- `by = NULL`: If you want your data frame to be applied to all character/factor columns indiscriminantly, then setting `by = NULL` will use that dictionary globally.

Value

a data frame with re-defined data based on the dictionary

Author(s)

Zhian N. Kamvar

Patrick Barks

See Also

`match_vec()`, which this function wraps.

Examples

```
# Read in dictionary and coded date examples --------------------

dict <- read.csv(matchmaker_example("spelling-dictionary.csv"),
  stringsAsFactors = FALSE)
dat <- read.csv(matchmaker_example("coded-data.csv"),
  stringsAsFactors = FALSE)
dat$date <- as.Date(dat$date)

# Clean spelling based on dictionary ----------------------------

dict # show the dict
head(dat) # show the data

res1 <- match_df(dat,
  dictionary = dict,
  from = "options",
  to = "values",
  by = "grp")
head(res1)

# Show warnings/errors from each column -------------------------
# Internally, the `match_vec()` function can be quite noisy with warnings for
# various reasons. Thus, by default, the `match_df()` function will keep
# these quiet, but you can have them printed to your console if you use the
# warn = TRUE option:

res1 <- match_df(dat,
```

```
  dictionary = dict,
  from = "options",
  to = "values",
  by = "grp",
  warn = TRUE)
head(res1)


# You can ensure the order of the factors are correct by specifying
# a column that defines order.

dat[] <- lapply(dat, as.factor)
as.list(head(dat))
res2 <- match_df(dat,
  dictionary = dict,
  from = "options",
  to = "values",
  by = "grp",
  order = "orders")
head(res2)
as.list(head(res2))
```

`match_vec` *Rename values in a vector based on a dictionary*

Description

This function provides an interface for `forcats::fct_recode()`, `forcats::fct_explicit_na()`, and `forcats::fct_relevel()` in such a way that a data dictionary can be imported from a data frame.

Usage

```
match_vec(
  x = character(),
  dictionary = data.frame(),
  from = 1,
  to = 2,
  quiet = FALSE,
  warn_default = TRUE,
  anchor_regex = TRUE
)
```

Arguments

`x`	a character or factor vector
`dictionary`	a matrix or data frame defining mis-spelled words or keys in one column (`from`) and replacement values (`to`) in another column. There are keywords that can be appended to the `from` column for addressing default values and missing data.

from	a column name or position defining words or keys to be replaced
to	a column name or position defining replacement values
quiet	a logical indicating if warnings should be issued if no replacement is made; if FALSE, these warnings will be disabled
warn_default	a logical. When a .default keyword is set and warn_default = TRUE, a warning will be issued listing the variables that were changed to the default value. This can be used to update your dictionary.
anchor_regex	a logical. When TRUE (default), any regex within the keywork

Details

Keys (from column):

The from column of the dictionary will contain the keys that you want to match in your current data set. These are expected to match exactly with the exception of three reserved keywords that start with a full stop:

- .regex [pattern]: will replace anything matching [pattern]. **This is executed before any other replacements are made**. The [pattern] should be an unquoted, valid, PERL-flavored regular expression. Any whitespace padding the regular expression is discarded.
- .missing: replaces any missing values (see NOTE)
- .default: replaces **ALL** values that are not defined in the dictionary and are not missing.

Values (to column):

The values will replace their respective keys exactly as they are presented.

There is currently one recognised keyword that can be placed in the to column of your dictionary:

- .na: Replace keys with missing data. When used in combination with the .missing keyword (in column 1), it can allow you to differentiate between explicit and implicit missing data.

Value

a vector of the same type as x with mis-spelled labels cleaned. Note that factors will be arranged by the order presented in the data dictionary; other levels will appear afterwards.

Note

If there are any missing values in the from column (keys), then they are automatically converted to the character "NA" with a warning. If you want to target missing data with your dictionary, use the .missing keyword. The .regex keyword uses gsub() with the perl = TRUE option for replacement.

Author(s)

Zhian N. Kamvar

See Also

match_df() for an implementation that acts across multiple variables in a data frame.

Examples

```
corrections <- data.frame(
  bad = c("foubar", "foobr", "fubar", "unknown", ".missing"),
  good = c("foobar", "foobar", "foobar", ".na", "missing"),
  stringsAsFactors = FALSE
)
corrections

# create some fake data
my_data <- c(letters[1:5], sample(corrections$bad[-5], 10, replace = TRUE))
my_data[sample(6:15, 2)] <- NA  # with missing elements

match_vec(my_data, corrections)

# You can use regular expressions to simplify your list
corrections <- data.frame(
  bad =  c(".regex f[ou][^m].+?r$", "unknown", ".missing"),
  good = c("foobar",                ".na",     "missing"),
  stringsAsFactors = FALSE
)

# You can also set a default value
corrections_with_default <- rbind(corrections, c(bad = ".default", good = "unknown"))
corrections_with_default

# a warning will be issued about the data that were converted
match_vec(my_data, corrections_with_default)

# use the warn_default = FALSE, if you are absolutely sure you don't want it.
match_vec(my_data, corrections_with_default, warn_default = FALSE)

# The function will give you a warning if the dictionary does not
# match the data
match_vec(letters, corrections)

# The can be used for translating survey output

words <- data.frame(
  option_code = c(".regex ^[yY][eE]?[sS]?",
    ".regex ^[nN][oO]?",
    ".regex ^[uU][nN]?[kK]?",
    ".missing"),
  option_name = c("Yes", "No", ".na", "Missing"),
  stringsAsFactors = FALSE
)
match_vec(c("Y", "Y", NA, "No", "U", "UNK", "N"), words)
```

Index