Package: cleanepi (via r-universe)

October 22, 2024

Title Clean and Standardize Epidemiological Data

Version 1.0.2.9000

Description Cleaning and standardizing tabular data package, tailored specifically for curating epidemiological data. It streamlines various data cleaning tasks that are typically expected when working with datasets in epidemiology. It returns the processed data in the same format, ensuring seamless integration into existing workflows. Additionally, it generates a comprehensive report detailing the outcomes of each cleaning task.

License MIT + file LICENSE

URL https://epiverse-trace.github.io/cleanepi/, https://github.com/epiverse-trace/cleanepi

BugReports https://github.com/epiverse-trace/cleanepi/issues

Depends R (>= 4.0.0)

Imports checkmate, cli, dplyr, janitor, linelist (>= 1.0.0), lubridate, magrittr, matchmaker, numberize, readr, rlang, tibble, utils

Suggests htmlwidgets, kableExtra, knitr, lintr, markdown, reactable, rmarkdown, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/Needs/website epiverse-trace/epiversetheme

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2.9000

Repository https://epiverse-connect.r-universe.dev

RemoteUrl https://github.com/epiverse-trace/cleanepi

RemoteRef HEAD

RemoteSha bed289331cef92b71b7e25c708f63d6062d9f1b3

Contents

`add_to_dictionary` *Add an element to the data dictionary*

Description

Add an element to the data dictionary

Usage

```
add_to_dictionary(dictionary, option, value, grp, order = NULL)
```

Arguments

`dictionary`	A data dictionary in a form of a data frame
`option`	A vector of strings with the new options that need to be added to the dictionary.
`value`	A vector with the values to be used when replacing the new options.
`grp`	A vector with the name of the column that contains the option of interest.
`order`	A numeric with the order of the new option.

Value

An object of type data frame. This is the new data dictionary with an additional line that contains the details about the new options.

Examples

```
test <- add_to_dictionary(
  dictionary = readRDS(
    system.file("extdata", "test_dict.RDS", package = "cleanepi")
  ),
  option = "ml",
  value = "male",
  grp = "gender",
  order = NULL
 )
```

`add_to_report` *Add an element to the report object*

Description

Add an element to the report object

Usage

```
add_to_report(x, key, value = NULL)
```

Arguments

`x`	A data frame or linelist
`key`	The name of the cleaning operation
`value`	The object to add to the report object

Value

The input report object with an additional element

Examples

```
# scan through the data
scan_res <- scan_data(
  data = readRDS(system.file("extdata", "test_df.RDS", package = "cleanepi"))
)

# Perform data cleaning
cleaned_data <- clean_data(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  to_numeric = list(target_columns = "sex", lang = "en"),
  dictionary = NULL
)
```

```
# add the data scanning result to the report
cleaned_data <- add_to_report(
  x = cleaned_data,
  key = "scanning_result",
  value = scan_res
)
```

---

check_date_sequence *Check whether the order of the sequence of date-events is valid.*

Description

Checks whether a date sequence in a vector of specified columns is in order or not.

Usage

```
check_date_sequence(data, target_columns)
```

Arguments

data	The input data frame or linelist
target_columns	A vector of column names for events. Users should specify at least 2 column names in the expected order. For example: `target_columns = c("date_symptoms_onset", "date_hospitalization", "date_death")`. When the input data is a `linelist` object, this parameter can be set to `linelist_tags` if you wish to use the date sequence across only the tagged columns columns only. The date values in the target columns should be in the ISO8601 format (2024-12-31). Otherwise, use the `standardize_dates()` function to standardize the target columns.

Value

The input dataset. When found, the incorrect date sequences will be stored in the report and can be accessed using `attr(data, "report")`.

Examples

```
# import the data
data <- readRDS(system.file("extdata", "test_df.RDS", package = "cleanepi"))

# standardize the date values
data <- data %>%
  standardize_dates(
    target_columns  = c("date_first_pcr_positive_test", "date.of.admission"),
    error_tolerance = 0.4,
    format = NULL,
    timeframe = NULL
  )
```

```
# check the date sequence in two columns
good_date_sequence <- check_date_sequence(
  data = data,
  target_columns = c("date_first_pcr_positive_test", "date.of.admission")
)
```

`check_subject_ids` *Check whether the subject IDs comply with the expected format. When incorrect IDs are found, the function sends a warning and the user can call the* `correct_subject_ids()` *function to correct them.*

Description

Check whether the subject IDs comply with the expected format. When incorrect IDs are found, the function sends a warning and the user can call the `correct_subject_ids()` function to correct them.

Usage

```
check_subject_ids(
  data,
  target_columns,
  prefix = NULL,
  suffix = NULL,
  range = NULL,
  nchar = NULL
)
```

Arguments

`data`	The input data frame or linelist
`target_columns`	A vector of column names with the subject ids.
`prefix`	A prefix used in the subject IDs
`suffix`	A suffix used in the subject IDs
`range`	A vector with the range of numbers in the sample IDs
`nchar`	An integer that represents the expected number of characters in the subject ids.

Value

The input dataset with a warning if incorrect subject ids were found

Examples

```
dat <- check_subject_ids(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  target_columns = "study_id",
  prefix = "PS",
  suffix = "P2",
  range = c(1, 100),
  nchar = 7
)
```

`clean_data` *Clean and standardize data*

Description

Cleans up messy data frames by performing several operations. These include among others: cleaning of column names, detecting and removing duplicates, empty records and columns, constant columns, replacing missing values by NA, converting character columns into dates when they contain a certain number of date values, detecting subject IDs with wrong formats, etc.

Usage

```
clean_data(data, ...)
```

Arguments

`data`	The input data frame or linelist
`...`	A list of cleaning operations to be applied on the input data. The acceptable arguments for `...` are:

`standardize_column_names` A list with the arguments needed to standardize the column names. The elements of this list are the input for the `standardize_column_names` function.

`replace_missing_values` A list of parameters to be used when replacing the missing values by `NA`. The elements of the list are the inputs for the `replace_missing_values` function.

`remove_duplicates` A list with the arguments that define the columns and other parameters to be considered when looking for duplicates. They are the input values for the `remove_duplicates` function.

`remove_constants` A list with the parameters that define whether to remove constant data or not. The values are the input for the `remove_constants` function.

`standardize_dates` A list of parameters that will be used to standardize the date values from the input data. They represent the input values for the `standardize_dates` function.

`standardize_subject_ids` A list of parameters that are needed to check the IDs that comply with the expect format. These arguments are the input values of the `check_subject_ids`.

`to_numeric` A list with the parameters needed to convert the specified columns into numeric. When provided, the parameters will be the input values for the `convert_to_numeric`.

`dictionary` A data frame that will be used to substitute the current values in the specified columns the those in the dictionary. It is the main argument for the `clean_using_dictionary` function.

`check_date_sequence` A list of arguments to be used when determining whether the sequence of date events is respected across all rows of the input data. The value in this list are the input for the `check_date_sequence` function.

Value

The cleaned input date according to the user-specified parameters. This is associated with a data cleaning report that can be accessed using `attr(cleaned_data, "report")`

Examples

```
# Parameters for column names standardization
standardize_column_names <- list(keep = NULL, rename = NULL)

# parameters to remove constant columns, empty rows and columns
remove_constants <- list(cutoff = 1)

# Parameters for substituting missing values with NA:
replace_missing_values <- list(target_columns = NULL, na_strings = "-99")

# Parameters for duplicates removal across all columns
remove_duplicates <- list(target_columns   = NULL)

# Parameters for dates standardization
standardize_dates <- list(
  target_columns = NULL,
  error_tolerance = 0.4,
  format = NULL,
  timeframe = as.Date(c("1973-05-29", "2023-05-29")),
  orders = list(
    world_named_months = c("Ybd", "dby"),
    world_digit_months = c("dmy", "Ymd"),
    US_formats = c("Omdy", "YOmd")
  )
)

# Parameters for subject IDs standardization
standardize_subject_ids <- list(
  target_columns = "study_id",
  prefix = "PS",
  suffix = "P2",
  range = c(1, 100),
```

```
  nchar = 7
)

# convert the 'sex' column into numeric
to_numeric <- list(target_columns = "sex", lang = "en")

# the dictionary-based cleaning will not be performed here
dictionary = NULL

# no need to check for the sequence of date events
check_date_sequence <- NULL

cleaned_data <- clean_data(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  standardize_column_names = standardize_column_names,
  remove_constants = remove_constants,
  replace_missing_values = replace_missing_values,
  remove_duplicates = remove_duplicates,
  standardize_dates = standardize_dates,
  standardize_subject_ids = standardize_subject_ids,
  to_numeric = to_numeric,
  dictionary = NULL,
  check_date_sequence = NULL
)
```

`clean_using_dictionary` *Perform dictionary-based cleaning*

Description

Perform dictionary-based cleaning

Usage

```
clean_using_dictionary(data, dictionary)
```

Arguments

`data`	The input data frame or linelist
`dictionary`	A data dictionary associated with the input data

Value

A data frame with cleaned values in the target columns specified in the data dictionary.

Examples

```
data <- readRDS(
  system.file("extdata", "messy_data.RDS", package = "cleanepi")
)
dictionary <- readRDS(
  system.file("extdata", "test_dict.RDS", package = "cleanepi")
)

data$gender[2] <- "homme"
cleaned_df <- clean_using_dictionary(
  data = data,
  dictionary = dictionary
)
```

---

common_na_strings *Common strings representing missing values*

Description

This vector contains common values of NA (missing) and is intended for use within {cleanepi} functions `replace_missing_values()`. The current list of strings used can be found by printing out `common_na_strings`. It serves as a helpful tool to explore your data for possible missing values. However, I strongly caution against using this to replace `NA` values without meticulously examining the incidence for each case. Please note that `common_na_strings` utilizes `\\` around the "?", ".", and "*" characters to prevent their wildcard

Usage

```
common_na_strings
```

Format

A vector of 35 character strings.

Source

This vector is a combination of `naniar::common_na_strings` (https://github.com/njtierney/naniar/) and other strings found in the literature.

convert_numeric_to_date *Convert numeric to date*

Description

Convert numeric to date

Usage

```
convert_numeric_to_date(data, target_columns, ref_date, forward = TRUE)
```

Arguments

data	The input data frame or linelist
target_columns	A vector of columns names to be converted from numeric to date. When the input data is a linelist object, this parameter can be set to linelist_tags if you wish to only convert the tagged columns.
ref_date	A reference date. This can also be a character string with the name of the reference column.
forward	A Boolean to indicate whether the counts started after the reference date (TRUE) or not (FALSE). The default is TRUE.

Value

A data frame where the column of interest are updated

Examples

```
data <- readRDS(system.file("extdata", "test_df1.RDS", package = "cleanepi"))
data <- convert_numeric_to_date(
  data = data,
  target_columns = "recruted_on_day",
  ref_date = as.Date("2022-10-13"),
  forward = TRUE
)
```

convert_to_numeric *Convert columns into numeric*

Description

When the function is invoked without specifying the column names to be converted, the target columns are the ones returned by the scan_data() function. Furthermore, it identifies columns where the proportion of numeric values is at least twice the percentage of character values and performs the conversion in them.

Usage

```
convert_to_numeric(data, target_columns = NULL, lang = c("en", "fr", "es"))
```

Arguments

data	The input data frame or linelist
target_columns	A vector of the target column names. When the input data is a linelist object, this parameter can be set to linelist_tags if the tagged columns are those to be converted into numeric.
lang	The text's language. Currently one of "en", "fr", "es".

Value

A data frame wherein all the specified or detected columns have been transformed into numeric format after the conversion process.

Examples

```
dat <- convert_to_numeric(
  data = readRDS(
    system.file("extdata", "messy_data.RDS", package = "cleanepi")
  ),
  target_columns = "age",
  lang = "en"
)
```

---

correct_subject_ids *Correct the wrong subject IDs based on the user-provided values.*

Description

After detecting incorrect subject IDs from the check_subject_ids() function, use this function to provide the correct IDs and perform the substitution.

Usage

```
correct_subject_ids(data, target_columns, correction_table)
```

Arguments

data	The input data frame or linelist
target_columns	A vector of column names with the subject ids.
correction_table	A data frame with the following two columns: 1. from: a column with the wrong subject IDs, 2. to: a column with the values to be used to substitute the incorrect ids.

Value

The input dataset where all subject ids comply with the expected format.

Examples

```
# detect the incorrect subject ids
dat <- check_subject_ids(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  target_columns = "study_id",
  prefix = "PS",
  suffix = "P2",
  range = c(1, 100),
  nchar = 7
)

# generate the correction table
correction_table <- data.frame(
  from = c("P0005P2", "PB500P2", "PS004P2-1"),
  to = c("PB005P2", "PB050P2", "PS004P2")
)

# perform the correction
dat <- correct_subject_ids(
  data = dat,
  target_columns = "study_id",
  correction_table = correction_table
)
```

---

find_duplicates *Identify and return duplicated rows in a data frame or linelist.*

Description

Identify and return duplicated rows in a data frame or linelist.

Usage

```
find_duplicates(data, target_columns = NULL)
```

Arguments

data	A data frame or linelist.
target_columns	A vector of columns names or indices to consider when looking for duplicates. When the input data is a `linelist` object, this parameter can be set to `linelist_tags` from which duplicates to be removed. Its default value is `NULL`, which considers duplicates across all columns.

Value

A data frame or linelist of all duplicated rows with following 2 additional columns:

1. `row_id`: the indices of the duplicated rows from the input data. Users can choose from these indices, which row they consider as redundant in each group of duplicates.
2. `group_id`: a unique identifier associated to each group of duplicates.

Examples

```
dups <- find_duplicates(
  data = readRDS(
    system.file("extdata", "test_linelist.RDS", package = "cleanepi")
  ),
  target_columns = c("dt_onset", "dt_report", "sex", "outcome")
)
```

---

print_report *Generate report from data cleaning operations*

Description

Generate report from data cleaning operations

Usage

```
print_report(
  data,
  report_title = "{cleanepi} data cleaning report",
  output_file_name = NULL,
  format = "html",
  print = TRUE
)
```

Arguments

`data`	A data frame or linelist object returned from the `clean_data()` or the main functions of each data cleaning module.
`report_title`	The title to appear on the report
`output_file_name`	A string specifying the name of the report file, excluding any file extension. If no file name is supplied, one will be automatically generated with the format `cleanepi_report_YYMMDD_HHMMSS`.
`format`	The file format of the report. Currently only `"html"` is supported.
`print`	A logical that specifies whether to print the generated HTML file or no. Default is `TRUE`.

Value

A string containing the name and path of the saved report

Examples

```
data <- readRDS(system.file("extdata", "test_df.RDS", package = "cleanepi"))
test_dictionary <- readRDS(
  system.file("extdata", "test_dictionary.RDS", package = "cleanepi")
)

# scan through the data
scan_res <- scan_data(data)

# Perform data cleaning
cleaned_data <- data %>%
 standardize_column_names(keep = NULL, rename = c("DOB" = "dateOfBirth")) %>%
 replace_missing_values(target_columns = NULL, na_strings = "-99") %>%
 remove_constants(cutoff = 1.0) %>%
 remove_duplicates(target_columns = NULL) %>%
 standardize_dates(target_columns  = NULL,
                   error_tolerance = 0.4,
                   format          = NULL,
                   timeframe   = as.Date(c("1973-05-29", "2023-05-29"))) %>%
 check_subject_ids(target_columns = "study_id",
                   prefix         = "PS",
                   suffix         = "P2",
                   range          = c(1L, 100L),
                   nchar          = 7L) %>%
 convert_to_numeric(target_columns = "sex", lang = "en") %>%
 clean_using_dictionary(dictionary = test_dictionary)

# add the data scanning result to the report
cleaned_data <- add_to_report(
  x = cleaned_data,
  key = "scanning_result",
  value = scan_res
)

# save a report in the current directory using the previously-created objects
print_report(
  data = cleaned_data,
  report_title = "{cleanepi} data cleaning report",
  output_file_name = NULL,
  format = "html",
  print = TRUE
)
```

remove_constants *Remove constant data i.e. empty rows and columns and constant columns*

Description

The function iteratively removes the constant data until there are not found anymore. It stores the details about the removed constant data in a form of a data frame within the report object.

Usage

```
remove_constants(data, cutoff = 1)
```

Arguments

data	The input data frame or linelist
cutoff	The cut-off for empty rows and columns removal. If provided, only rows and columns where the percent of missing data is greater than this cut-off will removed. Default is 1.

Value

The input dataset without the empty rows and columns and the constant columns.

Examples

```
data <- readRDS(system.file("extdata", "test_df.RDS", package = "cleanepi"))

# introduce an empty column
data$empty_column <- NA

# remove the constant columns, empty rows and columns where empty rows and
# columns are defined as those with 100% NA.
dat <- remove_constants(
  data = data,
  cutoff = 1
)

# remove the constant columns, empty rows and columns where empty rows and
# columns are defined as those with 50% NA.
dat <- remove_constants(
  data = data,
  cutoff = 0.5
)

# check the report to see what has happened
report <- attr(dat, "report")
report$constant_data
```

remove_duplicates *Remove duplicates*

Description

When removing duplicates, users can specify a set columns to consider with the `target_columns` argument.

Usage

```
remove_duplicates(data, target_columns = NULL)
```

Arguments

`data`	The input data frame or linelist.
`target_columns`	A vector of column names to use when looking for duplicates. When the input data is a `linelist` object, this parameter can be set to `linelist_tags` if you wish to look for duplicates on tagged columns only. Default is `NULL`.

Value

A data frame or linelist without the duplicated rows identified from all or the specified columns.

Examples

```
no_dups <- remove_duplicates(
  data = readRDS(
    system.file("extdata", "test_linelist.RDS", package = "cleanepi")
  ),
  target_columns = "linelist_tags"
)
```

replace_missing_values *Replace missing values with* `NA`

Description

Replace missing values with `NA`

Usage

```
replace_missing_values(
  data,
  target_columns = NULL,
  na_strings = cleanepi::common_na_strings
)
```

Arguments

data	A data frame or linelist
target_columns	A vector of column names. If provided, the substitution of missing values will only be executed in those specified columns. When the input data is a linelist object, this parameter can be set to linelist_tags if you wish to replace missing values with NA on tagged columns only.
na_strings	This is a vector of strings that represents the missing values in the columns of interest. By default, it utilizes cleanepi::common_na_strings. However, if the missing values string in the columns of interest is not included in this predefined vector, it can be used as the value for this argument.

Value

The input data where missing values are replaced by NA.

Examples

```
cleaned_data <- replace_missing_values(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  target_columns = "sex",
  na_strings = "-99"
)
```

---

scan_data *Scan through a data frame and return the proportion of* missing, numeric, Date, character, logical *values.*

Description

The function checks for the existence of character columns in the data. When found, it reports back the proportion of the data types mentioned above in those columns. See the details section to know more about how it works.

Usage

```
scan_data(data)
```

Arguments

data	A data frame or linelist

Details

How does it work? The `character` columns are identified first. When there is no character column the function returns a message. For every character column, we count:

1. the number of missing data `NA`
2. the number of numeric values. A process of detecting valid dates among the numeric values is then initiated using `lubridate::as_date()` and `date_guess()` functions. If found, a warning is triggered to alert on the presence and ambiguous (numeric values that are potentially date) values. NOTE: A date is considered valid in this case if it falls within the interval of today's date and 50 years back from today.
3. detect the Date values from the non-numeric using the `date_guess()` function. The date count is the sum of dates identified from numeric and non-numeric values. Because of the overlap between numeric and date, the sum across the rows in the scanning result might be greater than 1.
4. count the logical values. The remaining values will be those of type characters.

Value

A data frame if the input data contains columns of type character. It invisibly returns `NA` otherwise. The returned data frame will have the same number of rows as the number of character columns, and six columns representing their column names, proportion of missing, numeric, date, character, and logical values.

Examples

```
# scan through a data frame of characters
scan_result <- scan_data(
  data = readRDS(
    system.file("extdata", "messy_data.RDS", package = "cleanepi")
  )
)

# scan through a data frame with two character columns
scan_result <- scan_data(
  data = readRDS(system.file("extdata", "test_linelist.RDS",
                             package = "cleanepi"))
)

# scan through a data frame with no character columns
data(iris)
iris[["fct"]] <- as.factor(sample(c("gray", "orange"), nrow(iris),
                           replace = TRUE))
iris[["lgl"]]  <- sample(c(TRUE, FALSE), nrow(iris), replace = TRUE)
iris[["date"]] <- as.Date(seq.Date(from = as.Date("2024-01-01"),
                                   to = as.Date("2024-08-30"),
                                   length.out = nrow(iris)))
iris[["posit_ct"]] <- as.POSIXct(iris[["date"]])
scan_result        <- scan_data(data = iris)
```

standardize_column_names *Standardize column names of a data frame or linelist*

Description

All columns names will be reformatted to use the snakecase. When the conversion to snakecase does not work as expected, use the `keep` and/or `rename` arguments to reformat the column name properly.

Usage

```
standardize_column_names(data, keep = NULL, rename = NULL)
```

Arguments

`data`	The input data frame or linelist.
`keep`	A vector of column names to maintain as they are. When dealing with a linelist, this can be set to `linelist_tags`, to maintain the tagged column names. The Default is `NULL`.
`rename`	A named vector of column names to be renamed. This should be in the form of `c(new_name1 = "old_name1", new_name2 = "old_name2")` for example.

Value

A data frame or linelist with easy to work with column names.

Examples

```
# do not rename 'date.of.admission'
cleaned_data <- standardize_column_names(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  keep = "date.of.admission"
)

# do not rename 'date.of.admission', but rename 'dateOfBirth' and 'sex' to
# 'DOB' and 'gender' respectively
cleaned_data <- standardize_column_names(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  keep = "date.of.admission",
  rename = c(DOB = "dateOfBirth", gender = "sex")
)
```

`standardize_dates` *Standardize date variables*

Description

When the format of the values in a column and/or the target columns are not defined, we strongly recommend checking a few converted dates manually to make sure that the dates extracted from a `character` vector or a `factor` are correct.

Usage

```
standardize_dates(
  data,
  target_columns = NULL,
  format = NULL,
  timeframe = NULL,
  error_tolerance = 0.5,
  orders = NULL
)
```

Arguments

`data`	A data frame or linelist
`target_columns`	A vector of the target date column names. When the input data is a `linelist` object, this parameter can be set to `linelist_tags` if you wish to standardize the date columns across tagged columns only. Default is `NULL`.
`format`	A vector of the expected formats in the date values from the date columns. Default is `NULL`.
`timeframe`	A vector of 2 values of type date. If provided, date values that do not fall within this timeframe will be set to `NA`.
`error_tolerance`	A number between 0 and 1 indicating the proportion of entries which cannot be identified as dates to be tolerated; if this proportion is exceeded, the original vector is returned, and a message is issued; defaults to 0.4 (40 percent).
`orders`	A list or character vector with the date codes for fine-grained parsing of dates. This allows for parsing of mixed dates. If a list is supplied, that list will be used for successive tries in parsing. When this is not provided (`orders = NULL`), the function will use the following order defined in the guesser:

```
list(
  quarter_partial_dates = c("Y", "Ym", "Yq"),
 world_digit_months = c("Yq", "ymd", "ydm", "dmy", "mdy", "myd", "dym",
                        "Ymd", "Ydm", "dmY", "mdY", "mYd", "dYm"),
 world_named_months = c("dby", "dyb", "bdy", "byd", "ybd", "ydb",
                        "dbY", "dYb", "bdY", "bYd", "Ybd", "Ydb"),
  us_format = c("Omdy", "YOmd")
)
```

Details

Check for the presence of date values that could have multiple formats from the `$multi_format_dates` element of the `report`.

Converting ambiguous character strings to dates is difficult for many reasons:

- dates may not use the standard Ymd format
- within the same variable, dates may follow different formats
- dates may be mixed with things that are not dates
- the behavior of `as.Date` in the presence of non-date is hard to predict, sometimes returning `NA`, sometimes issuing an error.

This function tries to address all the above issues. Dates with the following format should be automatically detected, irrespective of separators (e.g. "-", " ", "/") and surrounding text:

- "19 09 2018"
- "2018 09 19"
- "19 Sep 2018"
- "2018 Sep 19"
- "Sep 19 2018"

How it works:

This function relies heavily on `lubridate::parse_date_time()`, which is an extremely flexible date parser that works well for consistent date formats, but can quickly become unwieldy and may produce spurious results. `standardize_dates()` will use a list of formats in the `orders` argument to run `parse_date_time()` with each format vector separately and take the first correctly parsed date from all the trials.

With the default orders shown above, the dates 03 Jan 2018, 07/03/1982, and 08/20/85 are correctly interpreted as 2018-01-03, 1982-03-07, and 1985-08-20. The examples section will show how you can manipulate the `orders` to be customized for your situation.

Value

The input dataset where the date columns have been standardized. The date values that are out of the specified timeframe will be reported in the report. Similarly, date values that comply with multiple formats will also be featured in the report object.

Examples

```
x <- c("03 Jan 2018", "07/03/1982", "08/20/85")
# The below will coerce values where the month is written in letters only
# into Date.
as.Date(lubridate::parse_date_time(x, orders = c("Ybd", "dby")))

# coerce values where the month is written in letters or numbers into Date.
as.Date(lubridate::parse_date_time(x, orders = c("dmy", "Ymd")))

# How to use standardize_dates()
```

```
dat <- standardize_dates(
  data = readRDS(
    system.file("extdata", "test_df.RDS", package = "cleanepi")
  ),
  target_columns = "date_first_pcr_positive_test",
  format = NULL,
  timeframe = NULL,
  error_tolerance = 0.4,
  orders = list(
    world_named_months = c("Ybd", "dby"),
    world_digit_months = c("dmy", "Ymd"),
    US_format = c("Omdy", "YOmd")
  )
)
```

`timespan` *Calculate time span between dates*

Description

Calculate time span between dates

Usage

```
timespan(
  data,
  target_column = NULL,
  end_date = Sys.Date(),
  span_unit = c("years", "months", "weeks", "days"),
  span_column_name = "span",
  span_remainder_unit = NULL
)
```

Arguments

`data`	The input data frame or linelist
`target_column`	A string used to specify the name of the date column of interest. The values in this column should be of type 'Date' in ISO8601 format ("2024-01-31").
`end_date`	An end date. It can be either a character that is the name of another column of type 'Date' from the input data or a vector of Dates or a single Date value. This should also be in the ISO8601 format ("2024-01-31"). Default is today's date `Sys.Date()`.
`span_unit`	A string that specifies the units in which the time span between the dates will be returned. The possible units are: 'years', 'months', 'weeks' or 'days'.
`span_column_name`	A string for the name of the new column to be used to store the calculated time span in the input data frame.

span_remainder_unit
: A string for the unit in which the remainder of the time span should be calculated. May be one of "months", "weeks", and "days". Remainders requested in the same unit as the age will return values of 0. Default is NULL for decimal time span.

Value

The input data frame with one or two additional columns:

1. "span" or any other name chosen by the user. This will contain the calculated time span in the desired units.
2. *"_remainder": a column with the number of the remaining days or weeks or months depending on the value of the 'span_remainder_unit' parameter. Here ''* represents the value of the 'span_column_name' argument.

Examples

```
# In the below example, this function is used to calculate patient's age from
# their dates of birth

# import the data, replace missing values with NA and convert date into ISO
# format
data <- readRDS(system.file("extdata", "test_df.RDS", package = "cleanepi"))
data <- data %>%
  replace_missing_values(target_columns = "dateOfBirth",
                         na_strings = "-99") %>%
  standardize_dates(target_columns = "dateOfBirth",
                    error_tolerance = 0.0)

# calculate the age in 'years' and return the remainder in 'months'
age <- timespan(
  data = data,
  target_column = "dateOfBirth",
  end_date = Sys.Date(),
  span_unit = "years",
  span_column_name = "age_in_years",
  span_remainder_unit = "months"
)
```

Index