Package 'cleaner'

May 8, 2026

Title Fast and Easy Data Cleaning

Version 1.5.5

Date 2024-11-17

Description Data cleaning functions for classes logical, factor, numeric, character, currency and Date to make data cleaning fast and easy. Relying on very few dependencies, it provides smart guessing, but with user options to override anything if needed.

Depends R (>= 3.0.0)

Imports backports, crayon, knitr, pillar, rlang (>= 0.3.1), vctrs

Suggests ggplot2, rmarkdown, testthat (>= 1.0.2)

URL https://msberends.github.io/cleaner/, https://github.com/msberends/cleaner

BugReports https://github.com/msberends/cleaner/issues

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author Matthijs S. Berends [aut, cre] (ORCID: <https://orcid.org/0000-0001-7620-1800>)

Maintainer Matthijs S. Berends <m.s.berends@umcg.nl>

Repository CRAN

Date/Publication 2024-11-19 10:50:04 UTC

Contents

`clean` *Clean column data to a class*

Description

Use any of these functions to quickly clean columns in your data set. Use `clean()` to pick the functions that return the least relative number of `NA`s. They **always** return the class from the function name (e.g. `clean_Date()` always returns class `Date`).

Usage

```
clean(x)

## S3 method for class 'data.frame'
clean(x)

clean_logical(
  x,
  true = regex_true(),
  false = regex_false(),
  na = NULL,
  fixed = FALSE,
  ignore.case = TRUE
)

clean_factor(
  x,
  levels = unique(x),
  ordered = FALSE,
  droplevels = FALSE,
  fixed = FALSE,
  ignore.case = TRUE
)

clean_numeric(x, remove = "[^0-9.,-]", fixed = FALSE)
```

```
clean_double(x, remove = "[^0-9.,-]", fixed = FALSE)

clean_integer(x, remove = "[^0-9.,-]", fixed = FALSE)

clean_character(
  x,
  remove = "[^a-z \t\r\n]",
  fixed = FALSE,
  ignore.case = TRUE,
  trim = TRUE
)

clean_currency(x, currency_symbol = NULL, remove = "[^0-9.,-]", fixed = FALSE)

clean_percentage(x, remove = "[^0-9.,-]", fixed = FALSE)

clean_Date(x, format = NULL, guess_each = FALSE, max_date = Sys.Date(), ...)

clean_POSIXct(
  x,
  tz = "",
  remove = "[^.0-9 :/-]",
  fixed = FALSE,
  max_date = Sys.Date(),
  ...
)
```

Arguments

x	data to clean
true	regex to interpret values as TRUE (which defaults to regex_true()), see Details
false	regex to interpret values as FALSE (which defaults to regex_false()), see Details
na	regex to force interpret values as NA, i.e. not as TRUE or FALSE
fixed	logical to indicate whether regular expressions should be turned off
ignore.case	logical to indicate whether matching should be case-insensitive
levels	new factor levels, may be named regular expressions to match existing values, see Details
ordered	logical to indicate whether the factor levels should be ordered
droplevels	logical to indicate whether non-existing factor levels should be dropped
remove	regex to define the character(s) that should be removed, see Details
trim	logical to indicate whether the result should be trimmed with trimws(..., which = "both")
currency_symbol	the currency symbol to use, which will be guessed based on the input and otherwise defaults to the current system locale setting (see Sys.localeconv())

format	character string giving a date-time format as used by strptime(). For clean_Date(..., guess_each = TRUE), this can be a vector of values to be used for guessing, see Examples.
guess_each	logical to indicate whether all items of x should be guessed one by one, see Examples
max_date	date (coercible with as.Date()) to indicate the maximum allowed of x, which defaults to today. This is to prevent that clean_Date("23-03-47") will return 23 March 2047 and instead returns 23 March 1947 with a warning.
...	for clean_Date and clean_POSIXct: other arguments passed on these functions
tz	a character string. The time zone specification to be used for the conversion, *if one is required*. System-specific (see time zones), but "" is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

Using clean() on a vector will guess a cleaning function based on the potential number of NAs it returns. Using clean() on a data frame to apply this guessed cleaning over all columns.

Info about the different functions:

- clean_logical(): Use arguments true and false to match values using case-insensitive regular expressions (regex). Unmatched values are considered NA. By default, values are matched with regex_true() and regex_false(). This allows support for values "Yes" and "No" in various languages. Use argument na to override values as NA that would otherwise be matched with true or false. See Examples.
- clean_factor(): Use argument levels to set new factor levels. They can be named case-insensitive regular expressions to match existing values of x. For matching, new values for levels are internally temporarily sorted descending on text length. See Examples.
- clean_numeric(), clean_double(), clean_integer() **and** clean_character(): Use argument remove to match values that must be removed from the input, using regular expressions (regex). In the case of clean_numeric(), commas will be read as dots and only the last dot will be kept. Function clean_character() will keep middle spaces by default. See Examples.
- clean_percentage(): This new class works like clean_numeric(), but transforms it with as.percentage(), which will retain the original values but will print them as percentages. See Examples.
- clean_currency(): This new class works like clean_numeric(), but transforms it with as.currency(). The currency symbol is guessed based on the most traded currencies by value (see Source): the United States dollar, Euro, Japanese yen, Pound sterling, Swiss franc, Renminbi, Swedish krona, Mexican peso, South Korean won, Turkish lira, Russian ruble, Indian rupee, and the South African rand. See Examples.
- clean_Date(): Use argument format to define a date format or leave it empty to have the format guessed. Use "Excel" to read values as Microsoft Excel dates. The format argument will be evaluated with format_datetime(), meaning that a format like "d-mmm-yy" will be translated internally to "%e-%b-%y" for convenience. See Examples.

- `clean_POSIXct()`: Use argument `remove` to match values that must be removed from the input, using regular expressions (regex). The resulting string will be coerced to a date/time element with class `POSIXct`, using `as.POSIXct()`. See Examples.

The use of invalid regular expressions in any of the above functions will not return an error (as in base R) but will instead interpret the expression as a fixed value and will throw a warning.

Value

The `clean_*` functions **always** return the class from the function name:

- `clean_logical()`: class `logical`
- `clean_factor()`: class `factor`
- `clean_numeric()` and `clean_double()`: class `numeric`
- `clean_integer()`: class `integer`
- `clean_character()`: class `character`
- `clean_percentage()`: class `percentage`
- `clean_currency()`: class `currency`
- `clean_Date()`: class `Date`
- `clean_POSIXct()`: classes `POSIXct/POSIXt`

Source

Triennial Central Bank Survey Foreign exchange turnover in April 2016 (PDF). Bank for International Settlements. 11 December 2016. p. 10.

Examples

```
clean_logical(c("Yes", "No"))   # English
clean_logical(c("Oui", "Non"))  # French
clean_logical(c("ya", "tidak")) # Indonesian
clean_logical(x = c("Positive", "Negative", "Unknown", "Some value"),
              true = "pos", false = "neg")

gender_age <- c("male 0-50", "male 50+", "female 0-50", "female 50+")
clean_factor(gender_age, c("M", "F"))
clean_factor(gender_age, c("Male", "Female"))
clean_factor(gender_age, c("0-50", "50+"), ordered = TRUE)
clean_factor(gender_age, levels = c("Group A" = "female", "Group B" = "male 50+", Other = ".*"))

clean_Date("13jul18", "ddmmmyy")
clean_Date("12 August 2010")
clean_Date("12 06 2012")
clean_Date("October 1st 2012")
clean_Date("43658")
clean_Date("14526", "Excel")
clean_Date(c("1 Oct 13", "October 1st 2012")) # could not be fitted in 1 format
clean_Date(c("1 Oct 13", "October 1st 2012"), guess_each = TRUE)
clean_Date(c("12-14-13", "1 Oct 2012"),
```

```
            guess_each = TRUE,
            format = c("d mmm yyyy", "mm-yy-dd")) # only these formats will be tried

clean_POSIXct("Created log on 2020/02/11 11:23 by user Joe")
clean_POSIXct("Created log on 2020.02.11 11:23 by user Joe", tz = "UTC")

clean_numeric("qwerty123456")
clean_numeric("Positive (0.143)")
clean_numeric("0,143")
clean_numeric("minus 12 degrees")

clean_percentage("PCT: 0.143")
clean_percentage(c("Total of -12.3%", "Total of +4.5%"))

clean_character("qwerty123456")
clean_character("Positive (0.143)")

clean_currency(c("Received 25", "Received 31.40"))
clean_currency(c("Jack sent £ 25", "Bill sent £ 31.40"))

df <- data.frame(A = c("2 Apr 2016", "5 Feb 2020"),
                 B = c("yes", "no"),
                 C = c("Total of -12.3%", "Total of +4.5%"),
                 D = c("Marker: 0.4513 mmol/l", "Marker: 0.2732 mmol/l"))
df
clean(df)
```

---

currency *Transform to currency*

Description

Transform input to a currency. The actual values are numeric, but will be printed as formatted currency values.

Usage

```
as.currency(x, currency_symbol = Sys.localeconv()["int_curr_symbol"], ...)

is.currency(x)

## S3 method for class 'currency'
print(
  x,
  decimal.mark = getOption("OutDec"),
  big.mark = ifelse(decimal.mark == ",", ".", ","),
  as_symbol = FALSE,
  ...
)
```

```
## S3 method for class 'currency'
format(
  x,
  currency_symbol = attributes(x)$currency_symbol,
  decimal.mark = getOption("OutDec"),
  big.mark = ifelse(decimal.mark == ",", ".", ","),
  as_symbol = FALSE,
  ...
)
```

Arguments

x	input
currency_symbol	the currency symbol to use, which defaults to the current system locale setting (see Sys.localeconv)
...	other parameters passed on to methods
decimal.mark	symbol to use as a decimal separator, defaults to getOption("OutDec")
big.mark	symbol to use as a thousands separator, defaults to a dot if decimal.mark is a comma, and a comma otherwise
as_symbol	try to format and print using currency symbols instead of text

Details

Printing currency will always have a currency symbol followed by a space, 2 decimal places and is never written in scientific format (like 2.5e+04).

Examples

```
money <- as.currency(c(0.25, 2.5, 25, 25000))
money
sum(money)
max(money)
mean(money)

format(money, currency_symbol = "USD")
format(money, currency_symbol = "EUR", decimal.mark = ",")
format(money, currency_symbol = "EUR", as_symbol = TRUE)

as.currency(2.5e+04)
```

format_datetime *Readable date format to POSIX*

Description

Use this function to transform generic date/time info writing (dd-mm-yyyy) into POSIX standardised format (%d-%m-%Y), see Examples.

Usage

```
format_datetime(format)
```

Arguments

format	the format that needs to be transformed

Value

A character string (a POSIX standardised format)

Examples

```
format_datetime("yyyy-mm-dd")

# Very hard to remember all these characters:
format(Sys.time(), "%a %b %d %Y %X")

# Easy to remember and write the same as above:
format(Sys.time(), format_datetime("ddd mmm dd yyyy HH:MM:ss"))

# seconds since the Epoch, 1970-01-01 00:00:00
format(Sys.time(), format_datetime("epoch"))
```

format_names *Format names and values*

Description

This function can be used on any data.frame, list or character vector to format their names or values. It supports snake case and camel case.

Usage

```
format_names(
  x,
  ...,
  snake_case = FALSE,
  camelCase = FALSE,
  tolower = FALSE,
  toupper = FALSE
)
```

Arguments

x	a data.frame, list or character vector
...	when x is a data.frame: new column names to set, which can be named (in the form old = "new"). The original column names do not need to be quoted, see Examples.
snake_case	logical to indicate whether the column names must be in snake case. This will have no effect on manually set column names.
camelCase	logical to indicate whether the column names must be in camel case. This will have no effect on manually set column names.
tolower, toupper	logical to indicate whether the column names must be lower/upper case. This will have no effect on manually set column names.

Examples

```
df <- data.frame(Name.341ABC = "value",
                 name_123def = "value",
                 This.is.a.column = "value")

format_names(df, snake_case = TRUE)

format_names(df, camelCase = TRUE)

format_names(df, letters[1:3])

format_names(df, This.is.a.column = "a_new_colname")

rownames(mtcars) <- format_names(rownames(mtcars), snake_case = TRUE)
mtcars[, 1:5]

format_names(list(a = 1, b = 2), c("new_1", "new_2"))

## Not run:
library(dplyr)
starwars %>%
  format_names(camelCase = TRUE) %>%        # new column names
  mutate(name = name %>%
           format_names(name,
```

```
                        snake_case = TRUE)) # new values in column

## End(Not run)
```

`format_p_value` *Format p values (APA guideline)*

Description

This function will round p values according to the APA guideline. It will try to round to two decimals where possible, and will try to avoid printing the value of `alpha`, see Examples.

Usage

```
format_p_value(p, alpha = 0.05, prepend_p = FALSE)
```

Arguments

`p`	p value(s) to transform
`alpha`	the value of alpha, defaults to 0.05
`prepend_p`	a logical to indicate whether "p =" should be prepended to the result

Value

A character

Examples

```
format_p_value(0.345678)
format_p_value(0.05125)

# this must not be "0.05", but is not "0.049" either,
# so it will add as many decimals as needed:
format_p_value(0.04993)

format_p_value(c(0.123, 0.00000001))
format_p_value(c(0.123, 0.00000001), prepend_p = TRUE)
```

`freq` *Frequency table*

Description

Create a frequency table of a `vector` or a `data.frame`. It supports tidyverse's quasiquotation and RMarkdown for reports. Easiest practice is: `data %>% freq(var)` using the tidyverse.

`top_freq` can be used to get the top/bottom *n* items of a frequency table, with counts as names. It respects ties.

Usage

```
freq(x, ...)

## Default S3 method:
freq(
  x,
  sort.count = TRUE,
  nmax = getOption("max.print.freq"),
  na.rm = TRUE,
  row.names = TRUE,
  markdown = !interactive(),
  digits = 2,
  quote = NULL,
  header = TRUE,
  title = NULL,
  na = "<NA>",
  sep = " ",
  decimal.mark = getOption("OutDec"),
  big.mark = "",
  wt = NULL,
  ...
)

## S3 method for class 'factor'
freq(x, ..., droplevels = FALSE)

## S3 method for class 'matrix'
freq(x, ..., quote = FALSE)

## S3 method for class 'table'
freq(x, ..., sep = " ")

## S3 method for class 'numeric'
freq(x, ..., digits = 2)
```

```
## S3 method for class 'Date'
freq(x, ..., format = "yyyy-mm-dd")

## S3 method for class 'hms'
freq(x, ..., format = "HH:MM:SS")

is.freq(f)

top_freq(f, n)

header(f, property = NULL)

## S3 method for class 'freq'
print(
  x,
  nmax = getOption("max.print.freq", default = 10),
  markdown = !interactive(),
  header = TRUE,
  decimal.mark = getOption("OutDec"),
  big.mark = ifelse(decimal.mark != ",", ",", "."),
  ...
)
```

Arguments

x	vector of any class or a data.frame or table
...	up to nine different columns of x when x is a data.frame or tibble, to calculate frequencies from - see Examples. Also supports quasiquotion.
sort.count	sort on count, i.e. frequencies. This will be TRUE at default for everything except when using grouping variables.
nmax	number of row to print. The default, 10, uses getOption("max.print.freq"). Use nmax = 0, nmax = Inf, nmax = NULL or nmax = NA to print all rows.
na.rm	a logical value indicating whether NA values should be removed from the frequency table. The header (if set) will always print the amount of NAs.
row.names	a logical value indicating whether row indices should be printed as 1:nrow(x)
markdown	a logical value indicating whether the frequency table should be printed in markdown format. This will print all rows (except when nmax is defined) and is default behaviour in non-interactive R sessions (like when knitting RMarkdown files).
digits	how many significant digits are to be used for numeric values in the header (not for the items themselves, that depends on getOption("digits"))
quote	a logical value indicating whether or not strings should be printed with surrounding quotes. Default is to print them only around characters that are actually numeric values.
header	a logical value indicating whether an informative header should be printed

title	text to show above frequency table, at default to tries to coerce from the variables passed to x
na	a character string that should be used to show empty (NA) values (only useful when na.rm = FALSE)
sep	a character string to separate the terms when selecting multiple columns
decimal.mark	the character to be used to indicate the numeric decimal point
big.mark	character; if not empty used as mark between every 'big.interval' decimals *before* (hence big) the decimal point
wt	frequency weights. If a variable, computes sum(wt) instead of counting the rows.
droplevels	a logical value indicating whether in factors empty levels should be dropped
format	a character to define the printing format (it supports format_datetime to transform e.g. "d mmmm yyyy" to "%e %B %Y")
f	a frequency table
n	number of top *n* items to return, use -n for the bottom *n* items. It will include more than n rows if there are ties.
property	property in header to return this value directly

Details

Frequency tables (or frequency distributions) are summaries of the distribution of values in a sample. With the 'freq' function, you can create univariate frequency tables. Multiple variables will be pasted into one variable, so it forces a univariate distribution.

Input can be done in many different ways. Base R methods are:

```
freq(df$variable)
freq(df[, "variable"])
```

Tidyverse methods are:

```
df$variable %>% freq()
df[, "variable"] %>% freq()
df %>% freq("variable")
df %>% freq(variable)
```

For numeric values of any class, these additional values will all be calculated with na.rm = TRUE and shown into the header:

- Mean, using mean
- Standard Deviation, using sd
- Coefficient of Variation (CV), the standard deviation divided by the mean
- Mean Absolute Deviation (MAD), using mad
- Tukey Five-Number Summaries (minimum, Q1, median, Q3, maximum), see *NOTE* below
- Interquartile Range (IQR) calculated as Q3 - Q1, see *NOTE* below

- Coefficient of Quartile Variation (CQV, sometimes called coefficient of dispersion) calculated as `(Q3 - Q1) / (Q3 + Q1)`, see *NOTE* below
- Outliers (total count and percentage), using `boxplot.stats`

NOTE: These values are calculated using the same algorithm as used by Minitab and SPSS: *p[k] = E[F(x[k])]*. See Type 6 on the `quantile` page.

For dates and times of any class, these additional values will be calculated with `na.rm = TRUE` and shown into the header:

- Oldest, using `min`
- Newest, using `max`, with difference between newest and oldest

In factors, all factor levels that are not existing in the input data will be dropped at default.

The function `top_freq` will include more than `n` rows if there are ties. Use a negative number for *n* (like `n = -3`) to select the bottom *n* values.

Value

A `data.frame` (with an additional class `"freq"`) with five columns: `item`, `count`, `percent`, `cum_count` and `cum_percent`.

Extending the `freq()` function

Interested in extending the `freq()` function with your own class? Add a method like below to your package, and optionally define some header info by passing a `list` to the `.add_header` parameter, like below example for class `difftime`. This example assumes that you use the `roxygen2` package for package development.

```
#' @method freq difftime
#' @importFrom cleaner freq.default
#' @export
#' @noRd
freq.difftime <- function(x, ...) {
  freq.default(x = x, ...,
               .add_header = list(units = attributes(x)$units))
}
```

Be sure to call `freq.default` in your function and not just `freq`. Also, add `cleaner` to the `Imports:` field of your `DESCRIPTION` file, to make sure that it will be installed with your package, e.g.:

```
Imports: cleaner
```

Examples

```
freq(unclean$gender, markdown = FALSE)

freq(x = clean_factor(unclean$gender,
                      levels = c("^m" = "Male",
                                 "^f" = "Female")),
```

```
        markdown = TRUE,
        title = "Frequencies of a cleaned version for a markdown report!",
        header = FALSE,
        quote = TRUE)
```

`na_replace` *Replace NA values*

Description

This is a generic function to replace NA values in data. It takes most data types as input and is extendible by other packages.

Usage

```
na_replace(x, ...)

## Default S3 method:
na_replace(x, replacement = "", ...)

## S3 method for class 'data.frame'
na_replace(x, ..., replacement = NULL)

## S3 method for class 'matrix'
na_replace(x, replacement = 0, ...)

## S3 method for class 'list'
na_replace(x, replacement = NULL, ...)

## S3 method for class 'numeric'
na_replace(x, replacement = 0, ...)

## S3 method for class 'Date'
na_replace(x, replacement = Sys.Date(), ...)

## S3 method for class 'logical'
na_replace(x, replacement = FALSE, ...)
```

Arguments

`x`	any vector, data.frame, matrix or list with values of which `NA` must be replaced
`...`	When `x` is a `data.frame`: columns of `x` to affect. This supports tidy evaluation without the need to quote the columns, see Examples.
`replacement`	value to replace `NA` with. This is at default: `0` for numeric values and class matrix, `FALSE` for class logical, today for class `Date`, and `""` otherwise. Can also be a vector with the length of the number of NAs of `x` (`sum(is.na(x))`). When `x` is a `data.frame`, this can be a vector with the length of the number of columns to be affected, see Examples.

Details

All functions preserve attributes. Within a `list` or `data.frame`, all attributes per index/item/column are also preserved.

Examples

```
mtrx <- matrix(c(1, 2, NA, 3), nrow = 2)
mtrx
na_replace(mtrx)

na_replace(c(1, 2, NA, NA))
na_replace(c(1, 2, NA, NA), replacement = -1)
na_replace(c(1, 2, NA, NA), replacement = c(0, -1))

na_replace(c(Sys.Date(), NA)) # replacement defaults to 'today'

na_replace(c(TRUE, FALSE, NA))
na_replace(c(TRUE, FALSE, NA), replacement = TRUE)

# we're flexible, the class only remains the same if
# the replacement value allows it
na_replace(c(1, 2, 3, NA), replacement = "-")

# data.frame is a special case
mtcars[1:6, c("mpg", "hp")] <- NA
head(mtcars)
head(na_replace(mtcars, mpg, hp)) # no need to quote columns (but you can)
head(na_replace(mtcars, mpg, hp, replacement = c(999, 123)))

## Not run:
# practical way using tidyverse
library(dplyr)
starwars %>%
  na_replace()

# even maintains groups
starwars %>%
  group_by(hair_color) %>%
  na_replace(hair_color, replacement = "TEST!") %>%
  summarise(n = n())

## End(Not run)
```

---

percentage *Transform to percentage*

Description

Transform input to a percentage. The actual values are numeric, but will be printed as formatted percentages.

Usage

```
as.percentage(x, ...)

is.percentage(x)

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

## S3 method for class 'percentage'
format(x, digits = NULL, ...)

percentage(x, digits = NULL, ...)
```

Arguments

x	input
...	other parameters passed on to methods
digits	how many digits should be printed. It defaults to printing all decimals available in the data after transforming to a percentage, with a minimum of 0 and a maximum of 3.

Details

Printing percentages will always have a percentage symbol and is never written in scientific format (like 2.5e+04%).

The function `percentage` is a wrapper around `format(as.percentage(...))` with automatic determination of the number of digits, varying between 0 and 1. It also, unlike R, rounds according to basic math rules: `percentage(0.4455)` returns `"44.6%"` and not `"44.5%"`. This function always returns a character, and can also be used in plotting, see Examples.

Examples

```
proportion <- as.percentage(c(0.25, 2.5, 0.0025))
proportion
sum(proportion)
max(proportion)
mean(proportion)

as.percentage(2.5e-14)

as.percentage(pi)
format(as.percentage(pi))
format(as.percentage(pi), digits = 6)

round(0.4455 * 100, 1) # mind the rounding
percentage(0.4455) # does not round to 44.5%

if (require("ggplot2")) {
  ggplot(iris) +
```

```
    geom_col(aes(Species, Sepal.Length / sum(Sepal.Length)),
             position = "stack") +
    # add percentage as function to the labels:
    scale_y_continuous(labels = percentage)
}
```

---

rdate *Generate random dates*

Description

This function provides random date generation with a specified range, that defaults to the beginning and end of the current year.

Usage

```
rdate(
  n,
  min = paste0(format(Sys.Date(), "%Y"), "-01-01"),
  max = paste0(format(Sys.Date(), "%Y"), "-12-31"),
  ...
)
```

Arguments

n	number of observations. If length(n) > 1, the length is taken to be the number required.
min, max	lower and upper limits of the distribution. Must be (coercible to) valid dates.
...	parameters given to as.Date() for coercing the values of min and max

Examples

```
# generate a million random dates and check the distribution
hist(rdate(1000000), breaks = "months")
```

---

regex_true_false *Regular expressions for* TRUE *and* FALSE

Description

These functions just return a regular expression to define values TRUE and FALSE in the most spoken languages in the world. They are the default input for the function clean_logical.

Usage

```
regex_true()

regex_false()
```

Details

Both functions support values "Yes" and "No" in the following languages: Arabic, Bengali, Chinese (Mandarin), Dutch, English, French, German, Hindi, Indonesian, Japanese, Malay, Portuguese, Russian, Spanish, Telugu, Turkish and Urdu.

Note: all these translations are in Latin characters only (e.g. "da" for Russian, "haan" for Hindi and "hai" for Japanese).

Source

Wolfram Alpha, query: https://www.wolframalpha.com/input/?i=20+most+spoken+languages

---

unclean *Example data that is not clean*

Description

This typical data example can be used for checking and cleaning.

Usage

```
unclean
```

Format

A data.frame with 500 observations and the following variables:

`date` Dates imported from Excel, they are integers ranging from ~30,000 to ~43,000.

`gender` Characters with mixed values observed in original data about patients gender.

See Also

freq to check values and clean to clean them.

Index