Package ‘cleaner’

April 1, 2020

Title  Fast and Easy Data Cleaning
Version  1.4.0
Date  2020-04-01
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  crayon, knitr, pillar, rlang (>= 0.3.1), vctrs
Suggests  rmarkdown, testthat (>= 1.0.2)
URL  https://github.com/msberends/cleaner
BugReports  https://github.com/msberends/cleaner/issues
License  GPL-2
Encoding  UTF-8
LazyData  true
RoxygenNote  7.1.0
NeedsCompilation  no
Author  Matthijs S. Berends [aut, cre]
        (<https://orcid.org/0000-0001-7620-1800>)
Maintainer  Matthijs S. Berends <m.s.berends@umcg.nl>
Repository  CRAN
Date/Publication  2020-04-01 10:00:14 UTC

R topics documented:

  clean ................................................................. 2
  currency ............................................................. 6
  format_datetime .................................................... 8
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 NAs. They always return the class from the function name (e.g. `clean_Date()` always returns class `Date`).

Usage

```r
# Clean data

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

# Clean logical data

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

# Clean factor data

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

# Clean numeric data

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

# Clean double data

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

# Clean integer data

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

```r
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 with 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 to indicate to 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 parameters passed on these functions

tz time zone specification to be used for the conversion, if one is required. Systemspecific (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 parameters true and false to match values using case-insensitive regular expressions (regex). Unmatched values are considered NA. At default, values are matched with regex_true and regex_false. This allows support for 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. Use parameter na to override values as NA that would else be matched with true or false. See Examples.

• clean_factor():
  Use parameter levels to set new factor levels. They can be case-insensitive regular expressions to match existing values of x. For matching, new values for levels are internally temporary sorted descending on text length. See Examples.

• clean_numeric(), clean_double(), clean_integer() and clean_character():
  Use parameter remove to match values that must be removed from the input, using regular expressions (regex). In case of clean_numeric(), comma's will be read as dots and only the last dot will be kept. Function clean_character() will keep middle spaces at 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 parameter 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 parameter will be evaluated with `format_datetime`, which means that a format like "d-mm-yy" with be translated internally to "%e-%b-%Y" for convenience. See Examples.

- `clean_POSIXct()`:
  Use parameter `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 (like 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**


**Examples**

```r
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_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_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"))

clean("12 06 2012")

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)

<table>
<thead>
<tr>
<th>currency</th>
<th>Transform to currency</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

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

```r
is.currency(x)
```

```r
## S3 method for class 'currency'
print(
    x,
```
currency

```r
decimal.mark = getOption("OutDec"),
big.mark = ifelse(decimal.mark == ",", ".", ","),
as_symbol = TRUE,
...
)

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

**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**

```r
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 = FALSE)

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"))

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 = " ",
  ...
)

## 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(}
\texttt{x},
\texttt{nmax = getOption("max.print.freq", default = 10),}
\texttt{markdown = !interactive(),}
\texttt{header = TRUE,}
\texttt{decimal.mark = getOption("OutDec"),}
\texttt{big.mark = ifelse(decimal.mark != ",", ",", ","),}
\texttt{...}
\texttt{)}

**Arguments**

\textbf{x} vector of any class or a \texttt{data.frame} or \texttt{table}

\ldots up to nine different columns of \texttt{x} when \texttt{x} is a \texttt{data.frame} or \texttt{tibble}, to calculate frequencies from - see Examples. Also supports quasiquotation.

\textbf{sort.count} sort on count, i.e. frequencies. This will be \texttt{TRUE} at default for everything except when using grouping variables.

\textbf{nmax} number of row to print. The default, 10, uses \texttt{getOption("max.print.freq")}. Use \texttt{nmax = 0, nmax = Inf, nmax = NULL or nmax = NA} to print all rows.

\textbf{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.

\textbf{row.names} a logical value indicating whether row indices should be printed as 1:nrow(\texttt{x})

\textbf{markdown} a logical value indicating whether the frequency table should be printed in markdown format. This will print all rows (except when \texttt{nmax} is defined) and is default behaviour in non-interactive R sessions (like when knitting RMarkdown files).

\textbf{digits} how many significant digits are to be used for numeric values in the header (not for the items themselves, that depends on \texttt{getOption("digits")})

\textbf{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.

\textbf{header} a logical value indicating whether an informative header should be printed

\textbf{title} text to show above frequency table, at default to tries to coerce from the variables passed to \texttt{x}

\textbf{na} a character string that should be used to show empty (NA) values (only useful when \texttt{na.rm = FALSE})

\textbf{sep} a character string to separate the terms when selecting multiple columns

\textbf{decimal.mark} used for prettying (longish) numerical and complex sequences. Passed to \texttt{prettyNum:} that help page explains the details.

\textbf{big.mark} used for prettying (longish) numerical and complex sequences. Passed to \texttt{prettyNum:} that help page explains the details.

\textbf{droplevels} a logical value indicating whether in factors empty levels should be dropped

\textbf{format} a character to define the printing format (it supports \texttt{format_datetime} to transform e.g. "d mmmm yyyy" to "%e %B %Y")

\textbf{f} a frequency table
**freq**

`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:

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

Tidyverse methods are:

```r
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 \( \frac{Q3 - Q1}{Q3 + Q1} \), 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.

```r
# @exportMethod 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)

<table>
<thead>
<tr>
<th>percentage</th>
<th>Transform to percentage</th>
</tr>
</thead>
</table>

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)

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%
**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**

```r
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**

```r
# 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**

```r
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

*Topic **datasets**
unclean, 15

*Topic **frequency**
freq, 8

*Topic **freq**
freq, 8

*Topic **summarise**
freq, 8

*Topic **summary**
freq, 8

as.currency, 4
as.currency (currency), 6
as.percentage, 4
as.percentage (percentage), 12
as.POSIXct, 5

boxplot.stats, 11

clean, 2, 15

clean_character (clean), 2
clean_currency (clean), 2
clean_Date (clean), 2
clean_double (clean), 2
clean_factor (clean), 2
clean_integer (clean), 2
clean_logical (clean), 2
clean_logical (clean), 14
clean_logical (clean), 2
clean_numeric (clean), 2
clean_percentage (clean), 2
clean_POSIXct (clean), 2
currency, 6

data.frame, 10, 15

format.currency (currency), 6
format.percentage (percentage), 12
format_datetime, 5, 8, 10
freq, 8, 15

getOption, 7, 10

header (freq), 8

is.currency (currency), 6
is.freq (freq), 8
is.percentage (percentage), 12

list, 12

mad, 11
max, 11
mean, 11
min, 11

percentage, 12

prettyNum, 10
print.currency (currency), 6
print.freq (freq), 8
print.percentage (percentage), 12

quantile, 11

rdate, 14
regex, 3–5
regex_false, 3, 4
regex_false (regex_true_false), 14
regex_true, 3, 4
regex_true (regex_true_false), 14
regex_true_false, 14

sd, 11

strptime, 3
Sys.localeconv, 3, 7

table, 10

time zones, 4
top_freq (freq), 8
trimws, 3

unclean, 15