Package ‘dataPreparation’

July 4, 2023

Title Automated Data Preparation

Version 1.1.1

Description Do most of the painful data preparation for a data science project with a minimum amount of code; Take advantages of 'data.table' efficiency and use some algorithmic trick in order to perform data preparation in a time and RAM efficient way.

Depends R (>= 3.6.0),

License GPL-3 | file LICENSE

LazyData true

Encoding UTF-8

RoxygenNote 7.2.3

Suggests testthat (>= 2.0.0)

Imports data.table, lubridate, stringr, Matrix, progress

BugReports https://github.com/ELToulemonde/dataPreparation/issues

NeedsCompilation no

Author Emmanuel-Lin Toulemonde [aut, cre]

Maintainer Emmanuel-Lin Toulemonde <el.toulemonde@protonmail.com>

Repository CRAN

Date/Publication 2023-07-04 13:13:03 UTC

R topics documented:

adult *Adult for UCI repository*

Description

For examples and tutorials, and in order to build `messy_adult`, UCI adult data set is used.
Data Set Information:

Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

Prediction task is to determine whether a person makes over 50K a year.

Usage

```
data("adult")
```

Format

A data.frame with 32561 rows and 15 variables.

References

https://archive.ics.uci.edu/ml/datasets/adult

aggregate_by_key *Automatic data_set aggregation by key*

Description

Automatic aggregation of a data_set set according to a `key`.

Usage

```
aggregate_by_key(data_set, key, verbose = TRUE, thresh = 53, ...)
```

Arguments

`data_set`	Matrix, data.frame or data.table (with only numeric, integer, factor, logical, character columns)
`key`	Name of a column of data_set according to which the set should be aggregated (character)
`verbose`	Should the algorithm talk? (logical, default to TRUE)
`thresh`	Number of max values for frequencies count (numerical, default to 53)

`...` Optional argument: `functions`: aggregation functions for numeric columns (vector of function names (character), optional, if not set we use: c("mean", "min", "max", "sd"))

Details

Perform aggregation depending on column type:

- If column is numeric `functions` are performed on the column. So 1 numeric column give length(functions) new columns,
- If column is character or factor and have less than `thresh` different values, frequency count of values is performed,
- If column is character or factor with more than `thresh` different values, number of different values for each `key` is performed,
- If column is logical, number of TRUE is computed.

In all cases, if the set as more rows than unique `key`, a number of lines will be computed.

Be careful using functions argument, given functions should be an aggregation function, meaning that for multiple values it should only return one value.

Value

A `data.table` with one line per key elements and multiple new columns.

Examples

```
## Not run:
# Get generic dataset from R
data("adult")

# Aggregate it using aggregate_by_key, in order to extract characteristics for each country
adult_aggregated <- aggregate_by_key(adult, key = 'country')

# Example with other functions
power <- function(x) {sum(x^2)}
adult_aggregated <- aggregate_by_key(adult, key = 'country', functions = c("power", "sqrt"))

# sqrt is not an aggregation function, so it wasn't used.

## End(Not run)
# "##NOT RUN:" mean that this example hasn't been run on CRAN since its long. But you can run it!
```

as.POSIXct_fast *Faster date transformation*

Description

Based on the trick that often dates are repeated in a column, we make date transformation faster by computing date transformation only on uniques.

Usage

```
as.POSIXct_fast(x, ...)
```

Arguments

x	An object to be converted
...	other argument to pass to as.POSIXct

Details

The more

Value

as.POSIXct and as.POSIXlt return an object of the appropriate class. If tz was specified, as.POSIXlt will give an appropriate "tzone" attribute. Date-times known to be invalid will be returned as NA.

Examples

```
# Work the same as as.POSIXct
as.POSIXct_fast("2018-01-01", format="%Y-%m-%d")
```

build_bins *Compute bins*

Description

Compute bins for discretization of numeric variable (either equal_width or equal_fred).

Usage

```
build_bins(
  data_set,
  cols = "auto",
  n_bins = 10,
  type = "equal_width",
  verbose = TRUE
)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`cols`	List of numeric column(s) name(s) of data_set to transform. To transform all characters, set it to "auto". (character, default to "auto")
`n_bins`	Number of group to compute (numeric, default to 10)
`type`	Type of discretization ("equal_width" or "equal_freq")
`verbose`	Should the algorithm talk? (Logical, default to TRUE)

Details

Using equal freq first bin will start at -Inf and last bin will end at +Inf.

Value

A list where each element name is a column name of data set and each element contains bins to discretize this column.

Examples

```
# Load data
data(tiny_messy_adult)
head(tiny_messy_adult)

# Compute bins
bins <- build_bins(tiny_messy_adult, cols = "auto", n_bins = 5, type = "equal_freq")
print(bins)
```

`build_date_factor` *Date Factor*

Description

Map a vector of dates to a factor at one of these levels "yearmonth", "yearquarter", "quarter", "month"

Usage

```
build_date_factor(data_set, type = "yearmonth")
```

Arguments

`data_set`	A vector of date values
`type`	One of "year", "yearquarter", "yearmonth", "quarter", "month"

Details

The resulting vector is an ordered factor of the specified `type` (e.g. yearmonth)

Examples

```
library(data.table)
data_set <- as.Date(c("2014-01-01", "2015-01-01", "2015-06-01"))
build_date_factor(data_set, type = "yearmonth")
build_date_factor(data_set, type = "yearquarter")
build_date_factor(data_set, type = "yearquarter")
```

`build_encoding` *Compute encoding*

Description

Build a list of one hot encoding for each `cols`.

Usage

```
build_encoding(data_set, cols = "auto", verbose = TRUE, min_frequency = 0, ...)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`cols`	List of numeric column(s) name(s) of data_set to transform. To transform all characters, set it to "auto". (character, default to "auto")
`verbose`	Should the algorithm talk? (Logical, default to TRUE)
`min_frequency`	The minimal share of lines that a category should represent (numeric, between 0 and 1, default to 0)
`...`	Other arguments such as `name_separator` to separate words in new columns names (character, default to ".")

Details

To avoid creating really large sparce matrices, one can use param `min_frequency` to be sure that only most representative values will be used to create a new column (and not out-layers or mistakes in data).
Setting `min_frequency` to something greater than 0 may cause the function to be slower (especially for large data_set).

Value

A list where each element name is a column name of data set and each element new_cols and values the new columns that will be built during encoding.

Examples

```
# Get a data set
data(adult)
encoding <- build_encoding(adult, cols = "auto", verbose = TRUE)

print(encoding)

# To limit the number of generated columns, one can use min_frequency parameter:
build_encoding(adult, cols = "auto", verbose = TRUE, min_frequency = 0.1)
# Set to 0.1, it will create columns only for values that are present 10% of the time.
```

build_scales *Compute scales*

Description

Build a list of means and standard deviation for each cols.

Usage

```
build_scales(data_set, cols = "auto", verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	List of numeric column(s) name(s) of data_set to transform. To transform all characters, set it to "auto". (character, default to "auto")
verbose	Should the algorithm talk? (Logical, default to TRUE)

Value

A list where each element name is a column name of data set and each element contains means and sd.

Examples

```
# Get a data set
data(adult)
scales <- build_scales(adult, cols = "auto", verbose = TRUE)

print(scales)
```

build_target_encoding *Build target encoding*

Description

Target encoding is the process of replacing a categorical value with the aggregation of the target variable. `build_target_encoding` is used to compute aggregations.

Usage

```
build_target_encoding(
  data_set,
  cols_to_encode,
  target_col,
  functions = "mean",
  verbose = TRUE
)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`cols_to_encode`	columns to aggregate according to (list)
`target_col`	column to aggregate (character)
`functions`	functions of aggregation (list or character, default to "mean"). Functions `compute_probability_ratio` and `compute_weight_of_evidence` are classically used functions
`verbose`	Should the algorithm talk? (Logical, default to TRUE)

Value

A `list` of `data.table` a data.table for each `cols_to_encode` each data.table containing a line by unique value of column and `len(functions)` + 1 columns.

Examples

```
# Build a data set
require(data.table)
data_set <- data.table(student = c("Marie", "Marie", "Pierre", "Louis", "Louis"),
                       grades = c(1, 1, 2, 3, 4))

# Perform target_encoding construction
build_target_encoding(data_set, cols_to_encode = "student", target_col = "grades",
                      functions = c("mean", "sum"))
```

compute_probability_ratio *Compute probability ratio*

Description

Probability ratio is an aggregation function that can be used for `build_target_encoding`. Probability ratio is the P(most freq element) / (1 - P(most frq element)).

Usage

```
compute_probability_ratio(x)
```

Arguments

x	A `list` of categorical elements

Details

To be more generic, the library compute P(most freq element) inplace of traditional formula P(1)/P(0)

Value

P(most freq element) / (1 - P(most frq element))

Examples

```
# Build example list
example_list <- c(1, 1, 1, 2, 2, 3)

# Compute probability ratio
compute_probability_ratio(example_list)
```

compute_weight_of_evidence *Compute weight of evidence*

Description

Weight of evidence is an aggregation function that can be used for `build_target_encoding`. Weight of evidence is the ln(P(most freq element) / (1 - P(most frq element))).

Usage

```
compute_weight_of_evidence(x)
```

Arguments

`x`	A `list` of categorical elements

Details

To be more generic, the library compute P(most freq element) inplace of traditional formula ln(P(1)/P(0))

Value

Weight of evidence

Examples

```
# Build example list
example_list <- c(1, 1, 1, 2, 2, 3)

# Compute weight of evidence
compute_weight_of_evidence(example_list)
```

`data_preparation_news` *Show the NEWS file*

Description

Show the NEWS file of the dataPreparation package.

Usage

```
data_preparation_news()
```

`date_format_unifier` *Unify dates format*

Description

Unify every column in a date format to the same date format.

Usage

```
date_format_unifier(data_set, format = "Date")
```

Arguments

`data_set`	Matrix, data.frame or data.table
`format`	Desired target format: Date, POSIXct or POSIXlt, (character, default to Date)

Details

This function only handle Date, POSIXct and POSIXlt dates.
POSIXct format is a bit slower than Date but can keep hours-min.

Value

The same data_set set but with dates column with the desired format.

Examples

```
# build a data.table
require(data.table)
data_set <- data.table( column1 = as.Date("2016-01-01"), column2 = as.POSIXct("2017-01-01") )

# Use the function
data_set = date_format_unifier(data_set, format = "Date")

# Control result
sapply(data_set, class)
# return Date for both columns
```

---

description *Describe data set*

Description

Generate extensive description of a data set.

Usage

```
description(data_set, level = 1, path_to_write = NULL, verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
level	Level of description (0: generic, 1: column by column) (numeric, default to 1)
path_to_write	Path where the report should be written (character, default to NULL)
verbose	Should the algorithm talk? (Logical, default to TRUE)

Examples

```
# Load exemple set
data(tiny_messy_adult)

# Describe it
description(tiny_messy_adult)
```

fast_discretization *Discretization*

Description

Discretization of numeric variable (either equal_width or equal_fred).

Usage

```
fast_discretization(data_set, bins = NULL, verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
bins	Result of function build_bins, (list, default to NULL). To perform the same discretization on train and test, it is recommended to compute build_bins before. If it is kept to NULL, build_bins will be called. bins could also be carefully hand written.
verbose	Should the algorithm talk? (Logical, default to TRUE)

Details

NAs will be putted in an NA category.

Value

Same dataset discretized by **reference**.
If you don't want to edit by reference please provide set data_set = copy(data_set).

Examples

```
# Load data
data(tiny_messy_adult)
head(tiny_messy_adult)

# Compute bins
bins <- build_bins(tiny_messy_adult, cols = "auto", n_bins = 5, type = "equal_freq")

# Discretize
tiny_messy_adult <- fast_discretization(tiny_messy_adult, bins = bins)

# Control
head(tiny_messy_adult)

# Example with hand written bins
data("adult")
adult <-  fast_discretization(adult, bins = list(age = c(0, 40, +Inf)))
print(table(adult$age))
```

fast_filter_variables *Filtering useless variables*

Description

Delete columns that are constant or in double in your data_set set.

Usage

```
fast_filter_variables(
  data_set,
  level = 3,
  keep_cols = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

data_set	Matrix, data.frame or data.table
level	which columns do you want to filter (1 = constant, 2 = constant and doubles, 3 = constant doubles and bijections, 4 = constant doubles bijections and included)(numeric, default to 3)
keep_cols	List of columns not to drop (list of character, default to NULL)
verbose	Should the algorithm talk (logical or 1 or 2, default to TRUE)
...	optional parameters to be passed to the function when called from another function

Details

verbose can be set to 2 have full details from which functions, otherwise they don't log. (verbose = 1 is equivalent to verbose = TRUE).

Value

The same data_set but with fewer columns. Columns that are constant, in double, or bijection of another have been deleted.

Examples

```
# First let's build a data.frame with 3 columns: a constant column, and a column in double
## Not run:
df <- data.frame(col1 = 1, col2 = rnorm(1e6), col3 = sample(c(1, 2), 1e6, replace = TRUE))
df$col4 <- df$col2
df$col5[df$col3 == 1] = "a"
df$col5[df$col3 == 2] = "b" # Same info than in col1 but with a for 1 and b for 2
head(df)
```

```
# Let's filter columns:
df <- fast_filter_variables(df)
head(df)

## End(Not run)
# Don't run for CRAN, you can run example
```

fast_handle_na *Handle NA values*

Description

Handle NAs values depending on the class of the column.

Usage

```
fast_handle_na(
  data_set,
  set_num = 0,
  set_logical = FALSE,
  set_char = "",
  verbose = TRUE
)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`set_num`	NAs replacement for numeric column, (numeric or function, default to 0)
`set_logical`	NAs replacement for logical column, (logical or function, default to FALSE)
`set_char`	NAs replacement for character column, (character or function, default to "")
`verbose`	Should the algorithm talk (logical, default to TRUE)

Details

To preserve RAM this function edits data_set by **reference**. To keep object unchanged, please use `copy`.
If you provide a function, it will be applied to the full column. So this function should handle NAs.
For factor columns, it will add NA to list of values.

Value

data_set as a `data.table` with NAs replaced.

Examples

```
# Build a useful data_set set for example
require(data.table)
data_set <- data.table(numCol = c(1, 2, 3, NA),
                   charCol = c("", "a", NA, "c"),
                   booleanCol = c(TRUE, NA, FALSE, NA))

# To set NAs to 0, FALSE and "" (respectively for numeric, logical, character)
fast_handle_na(copy(data_set))

# In a numeric column to set NAs as "missing"
fast_handle_na(copy(data_set), set_char = "missing")

# In a numeric column, to set NAs to the minimum value of the column#'
fast_handle_na(copy(data_set), set_num = min) # Won't work because min(c(1, NA)) = NA so put back NA
fast_handle_na(copy(data_set), set_num = function(x)min(x,na.rm = TRUE)) # Now we handle NAs

# In a numeric column, to set NAs to the share of NAs values
rateNA <- function(x) {
  sum(is.na(x)) / length(x)
}
fast_handle_na(copy(data_set), set_num = rateNA)
```

`fast_is_equal` *Fast checks of equality*

Description

Performs quick check if two objects are equal.

Usage

```
fast_is_equal(object1, object2)
```

Arguments

`object1`	An element, a vector, a data.frame, a data.table
`object2`	An element, a vector, a data.frame, a data.table

Details

This function uses exponential search trick, so it is fast for very large vectors, data.frame and data.table. This function is also very robust; you can compare a lot of stuff without failing.

Value

Logical (TRUE or FALSE) if the two objects are equals.

Examples

```
# Test on a character
fast_is_equal("a", "a")
fast_is_equal("a", "b")

# Test on a vector
myVector <- rep(x = "a", 10000)
fast_is_equal(myVector, myVector)

# Test on a data.table
fast_is_equal(tiny_messy_adult, messy_adult)
```

fast_round *Fast round*

Description

Fast round of numeric columns in a data.table. Will only round numeric, so don't worry about characters. Also, it computes it column by column so your RAM is safe too.

Usage

```
fast_round(data_set, cols = "auto", digits = 2, verbose = TRUE)
```

Arguments

data_set	matrix, data.frame or data.table
cols	List of numeric column(s) name(s) of data_set to transform. To transform all numerics columns, set it to "auto" (characters, default to "auto")
digits	The number of digits after comma (numeric, default to 2)
verbose	Should the algorithm talk? (logical, default to TRUE)

Details

It is performing round by **reference** on data_set, column by column, only on numercial columns. So that it avoid copying data_set in RAM.

Value

The same datasets but as a data.table and with numeric rounded.

Examples

```
# First let's build a very large data.table with random numbers
require(data.table)
M <- as.data.table(matrix(runif (3e4), ncol = 10))

M_rounded <- fast_round(M, 2)
# Lets add some character
M[, stringColumn := ”a string”]

# And use our function
M_rounded <- fast_round(M, 2)
# It still work :) and you don't have to worry about the string.
```

`fast_scale` *scale*

Description

Perform efficient scaling on a data set.

Usage

```
fast_scale(data_set, scales = NULL, way = ”scale”, verbose = TRUE)
```

Arguments

`data_set` Matrix, data.frame or data.table

`scales` Result of function `build_scales`, (list, default to NULL).
To perform the same scaling on train and test, it is recommended to compute `build_scales` before. If it is kept to NULL, build_scales will be called.

`way` should scaling or unscaling be performed? (character either "scale" or "unscale", default to "scale")

`verbose` Should the algorithm talk? (Logical, default to TRUE)

Details

Scaling numeric values is useful for some machine learning algorithm such as logistic regression or neural networks.
Unscaling numeric values can be very useful for most post-model analysis to do so set way to "unscale".
This implementation of scale will be faster that `scale` for large data sets.

Value

`data_set` with columns scaled (or unscaled) by **reference**. Scaled means that each column mean will be 0 and each column standard deviation will be 1.

Examples

```
# Load data
data(adult)

# compute scales
scales <- build_scales(adult, cols = "auto", verbose = TRUE)

# Scale data set
adult <- fast_scale(adult, scales = scales, verbose = TRUE)

# Control
print(mean(adult$age)) # Almost 0
print(sd(adult$age)) # 1

# To unscale it:
adult <- fast_scale(adult, scales = scales, way = "unscale", verbose = TRUE)

# Control
print(mean(adult$age)) # About 38.6
print(sd(adult$age)) # About 13.6
```

---

find_and_transform_dates

Identify date columns

Description

Find and transform dates that are hidden in a character column.
It use a bunch of default formats, and you can also add your own formats.

Usage

```
find_and_transform_dates(
  data_set,
  cols = "auto",
  formats = NULL,
  n_test = 30,
  ambiguities = "IGNORE",
  verbose = TRUE
)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`cols`	List of column(s) name(s) of data_set to look into. To check all all columns, set it to "auto". (characters, default to "auto")
`formats`	List of additional Date formats to check (see `strptime`)

`n_test` Number of non-null rows on which to test (numeric, default to 30)

`ambiguities` How ambiguities should be treated (see details in ambiguities section) (character, default to IGNORE)

`verbose` Should the algorithm talk? (Logical, default to TRUE)

Details

This function is using `identify_dates` to find formats. Please see it's documentation. In case `identify_dates` doesn't find wanted formats you can either provide format in param `formats` or use `set_col_as_date` to force transformation.

Value

data_set set (as a data.table) with identified dates transformed by **reference**.

Ambiguity

Ambiguities are often present in dates. For example, in date: 2017/01/01, there is no way to know if format is YYYY/MM/DD or YYYY/DD/MM.
Some times ambiguity can be solved by a human. For example 17/12/31, a human might guess that it is YY/MM/DD, but there is no sure way to know.
To be safe, find_and_transform_dates doesn't try to guess ambiguities.
To answer ambiguities problem, param `ambiguities` is now available. It can take one of the following values

- `IGNORE` function will then take the first format which match (fast, but can make some mistakes)
- `WARN` function will try all format and tell you - via prints - that there are multiple matches (and won't perform date transformation)
- `SOLVE` function will try to solve ambiguity by going through more lines, so will be slower. If it is able to solve it, it will transform the column, if not it will print the various acceptable formats.

If there are some columns that have no chance to be a match think of removing them from `cols` to save some computation time.

Examples

```
# Load exemple set
data(tiny_messy_adult)
head(tiny_messy_adult)
# using the find_and_transform_dates
find_and_transform_dates(tiny_messy_adult, n_test = 5)
head(tiny_messy_adult)

# Example with ambiguities
## Not run:
require(data.table)
data(tiny_messy_adult) # reload data
# Add an ambiguity by sorting date1
tiny_messy_adult$date1 = sort(tiny_messy_adult$date1, na.last = TRUE)
```

```
# Try all three methods:
result_1 = find_and_transform_dates(copy(tiny_messy_adult))
result_2 = find_and_transform_dates(copy(tiny_messy_adult), ambiguities = "WARN")
result_3 = find_and_transform_dates(copy(tiny_messy_adult), ambiguities = "SOLVE")

## End(Not run)
# "##NOT RUN:" mean that this example hasn't been run on CRAN since its long. But you can run it!
```

---

find_and_transform_numerics

Identify numeric columns in a data_set set

Description

Function to find and transform characters that are in fact numeric.

Usage

```
find_and_transform_numerics(
  data_set,
  cols = "auto",
  n_test = 30,
  verbose = TRUE
)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	List of column(s) name(s) of data_set to look into. To check all all columns, set it to "auto". (characters, default to "auto")
n_test	Number of non-null rows on which to test (numeric, default to 30)
verbose	Should the algorithm talk? (logical, default to TRUE)

Details

This function is looking for perfect transformation. If there are some mistakes in data_set, consider setting them to NA before.
If there are some columns that have no chance to be a match think of removing them from `cols` to save some computation time.

Value

The data_set set (as a data.table) with identified numeric transformed.

Warning

All these changes will happen **by reference**.

Examples

```
# Let's build a data_set set
data_set <- data.frame(ID = seq_len(5),
                  col1 = c("1.2", "1.3", "1.2", "1", "6"),
                  col2 = c("1,2", "1,3", "1,2", "1", "6")
                  )

# using the find_and_transform_numerics
find_and_transform_numerics(data_set, n_test = 5)
```

---

generate_date_diffs *Date difference*

Description

Perform the differences between all dates of the data_set set and optionally with a static date.

Usage

```
generate_date_diffs(
  data_set,
  cols = "auto",
  analysis_date = NULL,
  units = "years",
  drop = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`data_set` Matrix, data.frame or data.table

`cols` List of date column(s) name(s) of data_set to commute difference on. To transform all dates, set it to "auto". (character, default to "auto")

`analysis_date` Static date (Date or POSIXct, optional)

`units` Unit of difference between too dates (string, default to 'years')

`drop` Should `cols` be dropped after generation (logical, default to FALSE)

`verbose` should the function log (logical, default to TRUE)

`...` Other arguments such as `name_separator` to separate words in new columns names (character, default to ".")

Details

`units` is the same as `difftime` units, but with one more possibility: years.

Value

data_set (as a `data.table`) with more columns. A numeric column has been added for every couple of Dates. The result is in years.

Examples

```
# First build a useful data_set set
require(data.table)
data_set <- data.table(ID = seq_len(100),
                  date1 = seq(from = as.Date("2010-01-01"),
                              to = as.Date("2015-01-01"),
                              length.out = 100),
                  date2 = seq(from = as.Date("1910-01-01"),
                              to = as.Date("2000-01-01"),
                              length.out = 100)
                  )

# Now let's compute
data_set <- generate_date_diffs(data_set, cols = "auto", analysis_date = as.Date("2016-11-14"))
```

`generate_factor_from_date`

Generate factor from dates

Description

Taking Date or POSIXct colums, and building factor columns from them.

Usage

```
generate_factor_from_date(
  data_set,
  cols = "auto",
  type = "yearmonth",
  drop = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`data_set` Matrix, data.frame or data.table

`cols` List of date column(s) name(s) of data_set to transform into factor. To transform all dates, set it to "auto". (characters, default to "auto")

`type` "year", "yearquarter", "yearmonth", "quarter" or "month", way to aggregate a date, (character, default to "yearmonth")

`drop` Should `cols` be dropped after generation (logical, default to FALSE)

verbose	Should the function log (logical, default to TRUE)
...	Other arguments such as name_separator to separate words in new columns names (character, default to ".")

Value

data_set with new columns. data_set is edited by **reference**.

Examples

```
# Load set, and find dates
data(tiny_messy_adult)
tiny_messy_adult <- find_and_transform_dates(tiny_messy_adult, verbose = FALSE)

# Generate new columns
# Generate year month columns
tiny_messy_adult <- generate_factor_from_date(tiny_messy_adult, cols = c("date1", "date2", "num1"))
head(tiny_messy_adult[, .(date1.yearmonth, date2.yearmonth)])


# Generate quarter columns
tiny_messy_adult <- generate_factor_from_date(tiny_messy_adult,
                                              cols = c("date1", "date2"), type = "quarter")
head(tiny_messy_adult[, .(date1.quarter, date2.quarter)])
```

---

generate_from_character

Recode character

Description

Recode character into 3 new columns:

- was the value not NA, "NA", "",
- how often this value occurs,
- the order of the value (ex: M/F => 2/1 because F comes before M in alphabet).

Usage

```
generate_from_character(
  data_set,
  cols = "auto",
  verbose = TRUE,
  drop = FALSE,
  ...
)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`cols`	List of character column(s) name(s) of data_set to transform. To transform all characters, set it to "auto". (character, default to "auto")
`verbose`	Should the function log (logical, default to TRUE)
`drop`	Should `cols` be dropped after generation (logical, default to FALSE)
`...`	Other arguments such as `name_separator` to separate words in new columns names (character, default to ".")

Value

`data_set` with new columns. `data_set` is edited by **reference**.

Examples

```
# Load data set
data(tiny_messy_adult)
tiny_messy_adult <- un_factor(tiny_messy_adult, verbose = FALSE) # un factor ugly factors

# transform column "mail"
tiny_messy_adult <- generate_from_character(tiny_messy_adult, cols = "mail")
head(tiny_messy_adult)

# To transform all characters columns:
tiny_messy_adult <- generate_from_character(tiny_messy_adult, cols = "auto")
```

`generate_from_factor` *Recode factor*

Description

Recode factors into 3 new columns:

- was the value not NA, "NA", "",
- how often this value occurs,
- the order of the value (ex: M/F => 2/1 because F comes before M in alphabet).

Usage

```
generate_from_factor(
  data_set,
  cols = "auto",
  verbose = TRUE,
  drop = FALSE,
  ...
)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	list of character column(s) name(s) of data_set to transform. To transform all factors, set it to "auto". (character, default to "auto")
verbose	Should the function log (logical, default to TRUE)
drop	Should `cols` be dropped after generation (logical, default to FALSE)
...	Other arguments such as `name_separator` to separate words in new columns names (character, default to ".")

Value

`data_set` with new columns. `data_set` is edited by **reference**.

Examples

```
# Load data set
data(tiny_messy_adult)

# transform column "type_employer"
tiny_messy_adult <- generate_from_factor(tiny_messy_adult, cols = "type_employer")
head(tiny_messy_adult)

# To transform all factor columns:
tiny_messy_adult <- generate_from_factor(tiny_messy_adult, cols = "auto")
```

---

get_most_frequent_element *Get most frequent element*

Description

Provide most frequent element in a list, a `data.frame` or `data.table` column

Usage

```
get_most_frequent_element(x)
```

Arguments

x	A list, `data.frame` or `data.table` column

Value

The most frequent element

Examples

```
# Build example list
example_list <- c(1, 1, 2, 3, 1, 4, 1)

# Compute most frequent element
get_most_frequent_element(example_list)
```

---

identify_dates *Identify date columns*

Description

Function to identify dates columns and give there format. It use a bunch of default formats. But you can also add your own formats.

Usage

```
identify_dates(
  data_set,
  cols = "auto",
  formats = NULL,
  n_test = 30,
  ambiguities = "IGNORE",
  verbose = TRUE
)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	List of column(s) name(s) of data_set to look into. To check all all columns, set it to "auto". (characters, default to "auto")
formats	List of additional Date formats to check (see strptime)
n_test	Number of non-null rows on which to test (numeric, default to 30)
ambiguities	How ambiguities should be treated (see details in ambiguities section) (character, default to IGNORE)
verbose	Should the algorithm talk? (Logical, default to TRUE)

Details

This function is looking for perfect transformation. If there are some mistakes in data_set, consider setting them to NA before.
In the unlikely case where you have numeric higher than `as.numeric(as.POSIXct("1990-01-01"))` they will be considered as timestamps and you might have some issues. On the other side, if you have timestamps before 1990-01-01, they won't be found, but you can use `set_col_as_date` to force transformation.

Value

A named list with names being col names of `data_set` and values being formats.

Ambiguity

Ambiguities are often present in dates. For example, in date: 2017/01/01, there is no way to know if format is YYYY/MM/DD or YYYY/DD/MM.
Some times ambiguity can be solved by a human. For example 17/12/31, a human might guess that it is YY/MM/DD, but there is no sure way to know.
To be safe, find_and_transform_dates doesn't try to guess ambiguities.
To answer ambiguities problem, param `ambiguities` is now available. It can take one of the following values

- `IGNORE` function will then take the first format which match (fast, but can make some mistakes)
- `WARN` function will try all format and tell you - via prints - that there are multiple matches (and won't perform date transformation)
- `SOLVE` function will try to solve ambiguity by going through more lines, so will be slower. If it is able to solve it, it will transform the column, if not it will print the various acceptable formats.

Examples

```
# Load exemple set
data(tiny_messy_adult)
head(tiny_messy_adult)
# using the find_and_transform_dates
identify_dates(tiny_messy_adult, n_test = 5)
```

`messy_adult` *Adult with some ugly columns added*

Description

For examples and tutorials, messy_adult has been built using UCI `adult`.

Usage

```
data(tiny_messy_adult)
```

Format

A data.table with 32561 rows and 24 variables.

Details

We added 9 really ugly columns to the data set:

- 4 dates with various formats and time stamp, containing NAs
- 1 constant column
- 3 numeric with different decimal separator
- 1 email address

`one_hot_encoder` *One hot encoder*

Description

Transform factor column into 0/1 columns with one column per values of the column.

Usage

```
one_hot_encoder(
  data_set,
  encoding = NULL,
  type = "integer",
  verbose = TRUE,
  drop = FALSE
)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`encoding`	Result of function `build_encoding`, (list, default to NULL). To perform the same encoding on train and test, it is recommended to compute `build_encoding` before. If it is kept to NULL, build_encoding will be called.
`type`	What class of columns is expected? "integer" (0L/1L), "numeric" (0/1), or "logical" (TRUE/FALSE), (character, default to "integer")
`verbose`	Should the function log (logical, default to TRUE)
`drop`	Should `cols` be dropped after generation (logical, default to FALSE)

Details

If you don't want to edit your data set consider sending `copy(data_set)` as an input.
Please **be careful** using this function, it will generate as many columns as there different values in your column and might use a lot of RAM. To be safe, you can use parameter `min_frequency` in `build_encoding`.

Value

`data_set` edited by **reference** with new columns.

Examples

```
data(tiny_messy_adult)

# Compute encoding
encoding <- build_encoding(tiny_messy_adult, cols = c("marital", "occupation"), verbose = TRUE)

# Apply it
tiny_messy_adult <- one_hot_encoder(tiny_messy_adult, encoding = encoding, drop = TRUE)

# Apply same encoding to adult
data(adult)
adult <- one_hot_encoder(adult, encoding = encoding, drop = TRUE)

# To have encoding as logical (TRUE/FALSE), pass it in type argument
data(adult)
adult <- one_hot_encoder(adult, encoding = encoding, type = "logical", drop = TRUE)
```

---

prepare_set *Preparation pipeline*

Description

Full pipeline for preparing your data_set set.

Usage

```
prepare_set(data_set, final_form = "data.table", verbose = TRUE, ...)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`final_form`	"data.table" or "numerical_matrix" (default to data.table)
`verbose`	Should the algorithm talk? (logical, default to TRUE)
`...`	Additional parameters to tune pipeline (see details)

Details

Additional arguments are available to tune pipeline:

- `key` Name of a column of data_set according to which data_set should be aggregated (character)
- `analysis_date` A date at which the data_set should be aggregated (differences between every date and analysis_date will be computed) (Date)
- `n_unfactor` Number of max value in a factor, set it to -1 to disable `un_factor` function. (numeric, default to 53)
- `digits` The number of digits after comma (optional, numeric, if set will perform `fast_round`)

- dateFormats List of format of Dates in data_set (list of characters)
- name_separator character to separate parts of new column names (character, default to ".")
- functions Aggregation functions for numeric columns, see aggregate_by_key (list of functions names (character))
- factor_date_type Aggregation level to factorize date (see generate_factor_from_date) (character, default to "yearmonth")
- target_col A target column to perform target encoding, see target_encode (character)
- target_encoding_functions Functions to perform target encoding, see build_target_encoding, if target_col is not given will not do anything, (list, default to "mean")

Value

A data.table or a numerical matrix (according to final_form).
It will perform the following steps:

- Correct set: unfactor factor with many values, id dates and numeric that are hiden in character
- Transform set: compute differences between every date, transform dates into factors, generate features from character..., if key is provided, will perform aggregate according to this key
- Filter set: filter constant, in double or bijection variables. If ‘digits‘ is provided, will round numeric
- Handle NA: will perform fast_handle_na)
- Shape set: will put the result in asked shape (final_form) with acceptable columns format.

Examples

```
# Load ugly set
## Not run:
data(tiny_messy_adult)

# Have a look to set
head(tiny_messy_adult)

# Compute full pipeline
clean_adult <- prepare_set(tiny_messy_adult)

# With a reference date
adult_agg <- prepare_set(tiny_messy_adult, analysis_date = as.Date("2017-01-01"))

# Add aggregation by country
adult_agg <- prepare_set(tiny_messy_adult, analysis_date = as.Date("2017-01-01"), key = "country")

# With some new aggregation functions
power <- function(x) {sum(x^2)}
adult_agg <- prepare_set(tiny_messy_adult, analysis_date = as.Date("2017-01-01"), key = "country",
                         functions = c("min", "max", "mean", "power"))

## End(Not run)
# "##NOT RUN:" mean that this example hasn't been run on CRAN since its long. But you can run it!
```

remove_percentile_outlier
Percentile outlier filtering

Description

Remove outliers based on percentiles.
Only values within nth and `100 - nth` percentiles are kept.

Usage

```
remove_percentile_outlier(
  data_set,
  cols = "auto",
  percentile = 1,
  verbose = TRUE
)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`cols`	List of numeric column(s) name(s) of data_set to transform. To transform all numeric columns, set it to "auto". (character, default to "auto")
`percentile`	percentiles to filter (numeric, default to 1)
`verbose`	Should the algorithm talk? (logical, default to TRUE)

Details

Filtering is made column by column, meaning that extreme values from first element of `cols` are removed, then extreme values from second element of `cols` are removed, ...
So if filtering is performed on too many column, there ia high risk that a lot of rows will be dropped.

Value

Same dataset with less rows, edited by **reference**.
If you don't want to edit by reference please provide set `data_set = copy(data_set)`.

Examples

```
# Given
library(data.table)
data_set <- data.table(num_col = seq_len(100))

# When
data_set <- remove_percentile_outlier(data_set, cols = "auto", percentile = 1, verbose = TRUE)

# Then extreme value is no longer in set
```

```
1 %in% data_set[["num_col"]] # Is false
2 %in% data_set[["num_col"]] # Is true
```

remove_rare_categorical

Filter rare categories

Description

Filter rows that have a rare occurrences

Usage

```
remove_rare_categorical(
  data_set,
  cols = "auto",
  threshold = 0.01,
  verbose = TRUE
)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	List of column(s) name(s) of data_set to transform. To transform all columns, set it to "auto". (character, default to "auto")
threshold	share of occurrences under which row should be removed (numeric, default to 0.01)
verbose	Should the algorithm talk? (logical, default to TRUE)

Details

Filtering is made column by column, meaning that extreme values from first element of cols are removed, then extreme values from second element of cols are removed, ...
So if filtering is performed on too many column, there ia high risk that a lot of rows will be dropped.

Value

Same dataset with less rows, edited by **reference**.
If you don't want to edit by reference please provide set data_set = copy(data_set).

Examples

```
# Given a set with rare "C"
library(data.table)
data_set <- data.table(cat_col = c(sample(c("A", "B"), 1000, replace=TRUE), "C"))

# When calling function
```

```
data_set <- remove_rare_categorical(data_set, cols = "cat_col",
                                   threshold = 0.01, verbose = TRUE)

# Then there are no "C"
unique(data_set[["cat_col"]])
```

remove_sd_outlier *Standard deviation outlier filtering*

Description

Remove outliers based on standard deviation thresholds.
Only values within mean - sd * n_sigmas and mean + sd * n_sigmas are kept.

Usage

```
remove_sd_outlier(data_set, cols = "auto", n_sigmas = 3, verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	List of numeric column(s) name(s) of data_set to transform. To transform all numeric columns, set it to "auto". (character, default to "auto")
n_sigmas	number of times standard deviation is accepted (integer, default to 3)
verbose	Should the algorithm talk? (logical, default to TRUE)

Details

Filtering is made column by column, meaning that extreme values from first element of cols are removed, then extreme values from second element of cols are removed, ...
So if filtering is performed on too many column, there ia high risk that a lot of rows will be dropped.

Value

Same dataset with less rows, edited by **reference**.
If you don't want to edit by reference please provide set data_set = copy(data_set).

Examples

```
# Given
library(data.table)
col_vals <- runif(1000)
col_mean <- mean(col_vals)
col_sd <- sd(col_vals)
extreme_val <- col_mean + 6 * col_sd
data_set <- data.table(num_col = c(col_vals, extreme_val))

# When
```

```
data_set <- remove_sd_outlier(data_set, cols = "auto", n_sigmas = 3, verbose = TRUE)

# Then extreme value is no longer in set
extreme_val %in% data_set[["num_col"]] # Is false
```

same_shape *Give same shape*

Description

Transform `data_set` into the same shape as `reference_set`. Especially this function will be useful to make your test set have the same shape as your train set.

Usage

```
same_shape(data_set, reference_set, verbose = TRUE)
```

Arguments

`data_set`	Matrix, data.frame or data.table to transform
`reference_set`	Matrix, data.frame or data.table
`verbose`	Should the algorithm talk? (logical, default to TRUE)

Details

This function will make sure that `data_set` and `reference_set`

- have the same class
- have exactly the same columns
- have columns with exactly the same class
- have factor factor with exactly the same levels

You should always use this function before applying your model on a new data set to make sure that everything will go smoothly. But if this function change a lot of stuff you should have a look to your preparation process, there might be something wrong.

Value

Return `data_set` transformed in order to make it have the same shape as `reference_set`

Examples

```
## Not run:
# Build a train and a test
data(tiny_messy_adult)
data(adult)
train <- messy_adult
test <- adult # So test will have missing columns

# Prepare them
train <- prepare_set(train, verbose = FALSE, key = "country")
test <- prepare_set(test, verbose = FALSE, key = "country")

# Give them the same shape
test <- same_shape(test, train)
# As one can see in log, a lot of small change had to be done.
# This is an extreme case but you get the idea.

## End(Not run)
# "##NOT RUN:" mean that this example hasn't been run on CRAN since its long. But you can run it!
```

`set_as_numeric_matrix` *Numeric matrix preparation for Machine Learning.*

Description

Prepare a numeric matrix from a data.table. This matrix is suitable for machine learning purposes, since factors are binary. It may be sparse, include an intercept, and drop a reference column for each factor if required (when using `lm()`, for instance)

Usage

```
set_as_numeric_matrix(
  data_set,
  intercept = FALSE,
  all_cols = FALSE,
  sparse = FALSE
)
```

Arguments

`data_set` data.table

`intercept` Should a constant column be added? (logical, default to FALSE)

`all_cols` For each factor, should we create all possible dummies, or should we drop a reference dummy? (logical, default to FALSE)

`sparse` Should the resulting matrix be of a (sparse) Matrix class? (logical, default to FALSE)

`set_col_as_character` *Set columns as character*

Description

Set as character a column (or a list of columns) from a data.table.

Usage

```
set_col_as_character(data_set, cols = "auto", verbose = TRUE)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`cols`	List of column(s) name(s) of data_set to transform into characters. To transform all columns, set it to "auto". (characters, default to "auto")
`verbose`	Should the function log (logical, default to TRUE)

Value

data_set (as a `data.table`), with specified columns set as character.

Examples

```
# Build a fake data.frame
data_set <- data.frame(numCol = c(1, 2, 3), factorCol = as.factor(c("a", "b", "c")))

# Set numCol and factorCol as character
data_set <- set_col_as_character(data_set, cols = c("numCol", "factorCol"))
```

`set_col_as_date` *Set columns as POSIXct*

Description

Set as POSIXct a character column (or a list of columns) from a data.table.

Usage

```
set_col_as_date(data_set, cols = NULL, format = NULL, verbose = TRUE)
```

Arguments

`data_set` Matrix, data.frame or data.table

`cols` List of column(s) name(s) of data_set to transform into dates

`format` Date's format (function will be faster if the format is provided) (character or list of character, default to NULL).
For timestamps, format need to be provided ("s" or "ms" or second or millisecond timestamps)

`verbose` Should the function log (logical, default to TRUE)

Details

set_col_as_date is way faster when format is provided. If you want to identify dates and format automatically, have a look to `identify_dates`.
If input column is a factor, it will be returned as a POSIXct column.
If `cols` is kept to default (NULL) set_col_as_date won't do anything.

Value

`data_set` (as a `data.table`), with specified columns set as Date. If the transformation generated only NA, the column is set back to its original value.

Examples

```
# Lets build a data_set set
data_set <- data.frame(ID = seq_len(5),
                 date1 = c("2015-01-01", "2016-01-01", "2015-09-01", "2015-03-01", "2015-01-31"),
                 date2 = c("2015_01_01", "2016_01_01", "2015_09_01", "2015_03_01", "2015_01_31")
                          )

# Using set_col_as_date for date2
data_transformed <- set_col_as_date(data_set, cols = "date2", format = "%Y_%m_%d")

# Control the results
lapply(data_transformed, class)

# With multiple formats:
data_transformed <- set_col_as_date(data_set, format = list(date1 = "%Y-%m-%d", date2 = "%Y_%m_%d"))
lapply(data_transformed, class)

# It also works with timestamps
data_set <- data.frame(time_stamp = c(1483225200, 1485990000, 1488495600))
set_col_as_date(data_set, cols = "time_stamp", format = "s")
```

set_col_as_factor *Set columns as factor*

Description

Set columns as factor and control number of unique element, to avoid having too large factors.

Usage

```
set_col_as_factor(data_set, cols = "auto", n_levels = 53, verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	List of column(s) name(s) of data_set to transform into factor. To transform all columns set it to "auto", (characters, default to auto).
n_levels	Max number of levels for factor (integer, default to 53) set it to -1 to disable control.
verbose	Should the function log (logical, default to TRUE)

Details

Control number of levels will help you to distinguish true categorical columns from just characters that should be handled in another way.

Value

data_set(as a data.table), with specified columns set as factor or logical.

Examples

```
# Load messy_adult
data(tiny_messy_adult)

# we wil change education
tiny_messy_adult <- set_col_as_factor(tiny_messy_adult, cols = "education")

sapply(tiny_messy_adult[, .(education)], class)
# education is now a factor
```

set_col_as_numeric *Set columns as numeric*

Description

Set as numeric a character column (or a list of columns) from a data.table.

Usage

```
set_col_as_numeric(data_set, cols, strip_string = FALSE, verbose = TRUE)
```

Arguments

`data_set` Matrix, data.frame or data.table

`cols` List of column(s) name(s) of data_set to transform into numerics

`strip_string` should I change "," to "." in the string? (logical, default to FALSE) If set to TRUE, computation will be a bit longer

`verbose` Should the function log (logical, default to TRUE)

Value

data_set (as a `data.table`), with specified columns set as numeric.

Examples

```
# Build a fake data.table
data_set <- data.frame(charCol1 = c("1", "2", "3"),
 charCol2 = c("4", "5", "6"))

# Set charCol1 and charCol2 as numeric
data_set <- set_col_as_numeric(data_set, cols = c("charCol1", "charCol2"))

# Using strip string when spaces or wrong decimal separator is used
data_set <- data.frame(charCol1 = c("1", "2", "3"),
                      charCol2 = c("4, 1", "5, 2", "6, 3"))

# Set charCol1 and charCol2 as numeric
set_col_as_numeric(data_set, cols = c("charCol1", "charCol2"))
# generate mistakes
set_col_as_numeric(data_set, cols = c("charCol1", "charCol2"), strip_string = TRUE)
# Doesn't generate any mistake (but is a bit slower)
```

shape_set *Final preparation before ML algorithm*

Description

Prepare a data.table by:

- transforming numeric variables into factors whenever they take less than `thresh` unique variables
- transforming characters using `generate_from_character`
- transforming logical into binary integers
- dropping constant columns
- Sending the data.table to `set_as_numeric_matrix` (when `final_form` == `"numerical_matrix"`) will then allow you to get a numerical matrix usable by most Machine Learning Algorithms.

Usage

```
shape_set(data_set, final_form = "data.table", thresh = 10, verbose = TRUE)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`final_form`	"data.table" or "numerical_matrix" (default to data.table)
`thresh`	Threshold such that a numerical column is transformed into a factor whenever its number of unique modalities is smaller or equal to `thresh` (numeric, default to 10)
`verbose`	Should the algorithm talk? (logical, default to TRUE)

Warning

All these changes will happen **by reference**.

target_encode *Target encode*

Description

Target encoding is the process of replacing a categorical value with the aggregation of the target variable. the target variable. `target_encode` is used to apply this transformations on a data set. Function `build_target_encoding` must be used first to compute aggregations.

Usage

```
target_encode(data_set, target_encoding, drop = FALSE, verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
target_encoding	result of function build_target_encoding (list)
drop	Should col_to_encode be dropped after generation (logical, default to FALSE)
verbose	Should the algorithm talk? (Logical, default to TRUE)

Value

data_set with new cols of target_encoding merged to data_set using target_encoding names as merging key. data_set is edited by **reference**.

Examples

```
# Build a data set
require(data.table)
data_set <- data.table(student = c("Marie", "Marie", "Pierre", "Louis", "Louis"),
                       grades = c(1, 1, 2, 3, 4))

# Construct encoding
target_encoding <- build_target_encoding(data_set, cols_to_encode = "student",
                                         target_col = "grades", functions = c("mean", "sum"))

# Apply them
target_encode(data_set, target_encoding = target_encoding)
```

tiny_messy_adult *First 500 rows of* messy_adult

Description

First 500 rows of messy_adult

Usage

```
data(tiny_messy_adult)
```

Format

A data.table with 500 rows and 24 variables.

un_factor *Unfactor factor with too many values*

Description

To un-factorize all columns that have more than a given amount of various values. This function will be usefully after using some reading functions that put every string as factor.

Usage

```
un_factor(data_set, cols = "auto", n_unfactor = 53, verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
cols	List of column(s) name(s) of data_set to look into. To check all all columns, set it to "auto". (characters, default to "auto")
n_unfactor	Number of max element in a factor (numeric, default to 53)
verbose	Should the algorithm talk? (logical, default to TRUE)

Details

If a factor has (strictly) more than `n_unfactor` values it is un-factored.
It is recommended to use `find_and_transform_numerics` and `find_and_transform_dates` after this function.
If `n_unfactor` is set to -1, nothing will be performed.
If there are a lot of column that have been transformed, you might want to look at the documentation of your data reader in order to stop transforming everything into a factor.

Value

Same data_set (as a data.table) with less factor columns.

Examples

```
# Let's build a data_set
data_set <- data.frame(true_factor = factor(rep(c(1,2), 13)),
                       false_factor = factor(LETTERS))

# Let's un factorize all factor that have more than 5 different values
data_set <- un_factor(data_set, n_unfactor = 5)
sapply(data_set, class)
# Let's un factorize all factor that have more than 5 different values
data_set <- un_factor(data_set, n_unfactor = 0)
sapply(data_set, class)
```

which_are_bijection *Identify bijections*

Description

Find all the columns that are bijections of another column.

Usage

```
which_are_bijection(data_set, keep_cols = NULL, verbose = TRUE)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`keep_cols`	List of columns not to drop (list of character, default to NULL)
`verbose`	Should the algorithm talk (logical, default to TRUE)

Details

Bijection, meaning that there is another column containing the exact same information (but maybe coded differently) for example col1: Men/Women, col2 M/W.
This function is performing search by looking to every couple of columns. It computes numbers of unique elements in each column, and number of unique tuples of values.
Computation is made by exponential search, so that the function is faster.
If `verbose` is TRUE, the column logged will be the one returned.
Ex: if column i and column j (with j > i) are bijections it will return j, expect if j is a character then it return i.

Value

A list of index of columns that have an exact bijection in the data_set set.

Examples

```
# First let's get a data set
data("adult")

# Now let's check which columns are equals
which_are_in_double(adult)
# It doesn't give any result.

# Let's look of bijections
which_are_bijection(adult)
# Return education_num index because education_num and education which
# contain the same info
```

`which_are_constant` *Identify constant columns*

Description

Find all the columns that are constant.

Usage

```
which_are_constant(data_set, keep_cols = NULL, verbose = TRUE)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`keep_cols`	List of columns not to drop (list of character, default to NULL)
`verbose`	Should the algorithm talk (logical, default to TRUE)

Details

Algorithm is performing exponential search: it check constancy on row 1 to 10, if it's not constant it stops, if it's constant then on 11 to 100 ...
If you have a lot of columns than aren't constant, this function is way faster than a simple `length(unique())`!
The larger the data_set set is, the more interesting it is to use this function.

Value

List of column's indexes that are constant in the data_set set.

Examples

```
# Let's load our data_set
data(tiny_messy_adult)

# Let's try our function
which_are_constant(tiny_messy_adult)
# Indeed it return constant the name of the constant column.
```

which_are_included *Identify columns that are included in others*

Description

Find all the columns that don't contain more information than another column. For example if you have a column with an amount and another with the same amount but rounded, the second column is included in the first.

Usage

```
which_are_included(data_set, keep_cols = NULL, verbose = TRUE)
```

Arguments

`data_set`	Matrix, data.frame or data.table
`keep_cols`	List of columns not to drop (list of character, default to NULL)
`verbose`	Should the algorithm talk (logical, default to TRUE)

Details

This function is performing exponential search and is looking to every couple of columns.
Be very careful while using this function:
- if there is an id column, it will say everything is included in the id column;
- the order of columns will influence the result.

For example if you have a column with an amount and another with the same amount but rounded, the second column is included in the first.

And last but not least, with some machine learning algorithm it's not always smart to drop columns even if they don't give more info: the extreme example is the id example.

Value

A list of index of columns that have an exact duplicate in the `data_set`.

Examples

```
# Load toy data set
require(data.table)
data(tiny_messy_adult)

# Check for included columns
which_are_included(tiny_messy_adult)

# Return columns that are also constant, double and bijection
# Let's add a truly just included column
tiny_messy_adult$are50OrMore <- tiny_messy_adult$age > 50
```

```
which_are_included(tiny_messy_adult[, .(age, are50OrMore)])

# As one can, see this column that doesn't have additional info than age is spotted.

# But you should be careful, if there is a column id, every column will be dropped:
tiny_messy_adult$id = seq_len(nrow(tiny_messy_adult)) # build id
which_are_included(tiny_messy_adult)
```

which_are_in_double *Identify double columns*

Description

Find all the columns that are in double.

Usage

```
which_are_in_double(data_set, keep_cols = NULL, verbose = TRUE)
```

Arguments

data_set	Matrix, data.frame or data.table
keep_cols	List of columns not to drop (list of character, default to NULL)
verbose	Should the algorithm talk (logical, default to TRUE)

Details

This function is performing search by looking to every couple of columns. First it compares the first 10 lines of both columns. If they are not equal then the columns aren't identical, else it compares lines 11 to 100; then 101 to 1000... So this function is fast with data_set set with a large number of lines and a lot of columns that aren't equals.
If verbose is TRUE, the column logged will be the one returned.

Value

A list of index of columns that have an exact duplicate in the data_set set. Ex: if column i and column j (with j > i) are equal it will return j.

Examples

```
# First let's build a matrix with 3 columns and a lot of lines, with 1's everywhere
M <- matrix(1, nrow = 1e6, ncol = 3)

# Now let's check which columns are equals
which_are_in_double(M)
# It return 2 and 3: you should only keep column 1.

# Let's change the column 2, line 1 to 0. And check again
```

```
M[1, 2] <- 0
which_are_in_double(M)
# It only returns 3

# What about NA? NA vs not NA => not equal
M[1, 2] <- NA
which_are_in_double(M)
# It only returns 3

# What about NA?  Na vs NA => yep it's the same
M[1, 1] <- NA
which_are_in_double(M)
# It only returns 2
```

Index