Package: missMethods (via r-universe)

September 15, 2024

Title Methods for Missing Data

Version 0.4.0

Description Supply functions for the creation and handling of missing data as well as tools to evaluate missing data methods. Nearly all possibilities of generating missing data discussed by Santos et al. (2019) <doi:10.1109/ACCESS.2019.2891360> and some additional are implemented. Functions are supplied to compare parameter estimates and imputed values to true values to evaluate missing data methods. Evaluations of these types are done, for example, by Cetin-Berber et al. (2019) <doi:10.1177/0013164418805532> and Kim et al. (2005) <doi:10.1093/bioinformatics/bth499>.

License GPL-3

URL https://github.com/torockel/missMethods

BugReports https://github.com/torockel/missMethods/issues

Imports mvtnorm, stats, utils

Suggests ggplot2, knitr, lpSolve, norm, rmarkdown, testthat (>= 2.1.0), tibble

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.1

Repository https://torockel.r-universe.dev

RemoteUrl https://github.com/torockel/missmethods

RemoteRef HEAD

RemoteSha 37e73366e2109f80f19b276fd547d7e3ce3f9195

Contents

`apply_imputation` *Apply a function for imputation*

Description

Apply a function for imputation over rows, columns or combinations of both

Usage

```
apply_imputation(
  ds,
  FUN = mean,
  type = "columnwise",
  convert_tibble = TRUE,
  ...
)
```

Arguments

ds	A data frame or matrix with missing values.
FUN	The function to be applied for imputation.
type	A string specifying the values used for imputation; one of: "columnwise", "rowwise", "total", "Two-Way" or "Winer" (see details).
convert_tibble	If ds is a tibble, should it be converted (see section A note for tibble users).
...	Further arguments passed to FUN.

Details

The functionality of apply_imputation is inspired by the apply function. The function applies a function FUN to impute the missing values in ds. FUN must be a function, which takes a vector as input and returns exactly one value. The argument type is comparable to apply's MARGIN argument. It specifies the values that are used for the calculation of the imputation values. For example, type = "columnwise" and FUN = mean will impute the mean of the observed values in a column for all missing values in this column. In contrast, type = "rowwise" and FUN = mean will impute the mean of the observed values in a row for all missing values in this row.

List of all implemented types:

- "columnwise" (the default): imputes column by column; all observed values of a column are given to FUN and the returned value is used as the imputation value for all missing values of the column.
- "rowwise": imputes row by row; all observed values of a row are given to FUN and the returned value is used as the imputation value for all missing values of the row.
- "total": All observed values of ds are given to FUN and the returned value is used as the imputation value for all missing values of ds.
- "Winer": The mean value from "columnwise" and "rowwise" is used as the imputation value.
- "Two-Way": The sum of the values from "columnwise" and "rowwise" minus "total" is used as the imputation value.

If no value can be given to FUN (for example, if no value in a column is observed and type = "columnwise"), then a warning will be issued and no value will be imputed in the corresponding column or row.

Value

An object of the same class as ds with imputed missing values.

A note for tibble users

If you use tibbles and convert_tibble is TRUE the tibble is first converted to a data frame, then imputed and converted back. If convert_tibble is FALSE no conversion is done. However, depending on the tibble and the package version of tibble you use, imputation may not be possible and some errors will be thrown.

References

Beland, S., Pichette, F., & Jolani, S. (2016). Impact on Cronbach's α of simple treatment methods for missing data. *The Quantitative Methods for Psychology*, 12(1), 57-73.

See Also

A convenient interface exists for common cases like mean imputation: `impute_mean`, `impute_median`, `impute_mode`. All these functions call `apply_imputation`.

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
ds_mis <- delete_MCAR(ds, 0.2)
ds_imp_app <- apply_imputation(ds_mis, FUN = mean, type = "total")
# the same result can be achieved via impute_mean():
ds_imp_mean <- impute_mean(ds_mis, type = "total")
all.equal(ds_imp_app, ds_imp_mean)
```

---

count_NA *Count the number of* NA*s*

Description

Count the number of NA values in a vector, matrix or data frame

Usage

```
count_NA(x, type = "default")
```

Arguments

x	a vector, matrix or data frame, in which NA values are counted
type	How to count the NA values. Possible choices: • "default": If x is a matrix or a data frame, the number of missing values per column is returned. If x is something else, the total number of missing values in x is returned. • "all": The number of all missing values in x is returned. • "cols": The number of missing values per column is returned. • "rows": The number of missing values per row is returned.

Value

The number of missing values.

Examples

```
count_NA(c(1, NA, 3, NA, 5, NA))
test_df <- data.frame(X1 = rep(c(1, NA), 5), X2 = c(1:9, NA))
count_NA(test_df)
count_NA(test_df, "cols") # the default
count_NA(test_df, "rows")
count_NA(test_df, "all")
```

`delete_MAR_1_to_x` *Create MAR values using MAR1:x*

Description

Create missing at random (MAR) values using MAR1:x in a data frame or a matrix

Usage

```
delete_MAR_1_to_x(
  ds,
  p,
  cols_mis,
  cols_ctrl,
  x,
  cutoff_fun = median,
  prop = 0.5,
  use_lpSolve = TRUE,
  ordered_as_unordered = FALSE,
  n_mis_stochastic = FALSE,
  x_stochastic = FALSE,
  add_realized_x = FALSE,
  ...,
  miss_cols,
  ctrl_cols,
  stochastic
)
```

Arguments

`ds`	A data frame or matrix in which missing values will be created.
`p`	A numeric vector with length one or equal to length `cols_mis`; the probability that a value is missing.
`cols_mis`	A vector of column names or indices of columns in which missing values will be created.
`cols_ctrl`	A vector of column names or indices of columns, which controls the creation of missing values in `cols_mis`. Must be of the same length as `cols_mis`.

x	Numeric with length one (0 < x < Inf); odds are 1 to x for the probability of a value to be missing in group 1 against the probability of a value to be missing in group 2 (see details).
cutoff_fun	Function that calculates the cutoff values in the cols_ctrl.
prop	Numeric of length one; (minimum) proportion of rows in group 1 (only used for unordered factors).
use_lpSolve	Logical; should lpSolve be used for the determination of groups, if cols_ctrl[i] is an unordered factor.
ordered_as_unordered	Logical; should ordered factors be treated as unordered factors.
n_mis_stochastic	Logical, should the number of missing values be stochastic? If n_mis_stochastic = TRUE, the number of missing values for a column with missing values cols_mis[i] is a random variable with expected value nrow(ds) * p[i]. If n_mis_stochastic = FALSE, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values cols_mis[i] is round(nrow(ds) * p[i]). Possible deviations from this value, if any exists, are documented in Details.
x_stochastic	Logical; should the odds be stochastic or deterministic.
add_realized_x	Logical; if TRUE the realized odds for cols_mis will be returned (as attribute).
...	Further arguments passed to cutoff_fun.
miss_cols	Deprecated, use cols_mis instead.
ctrl_cols	Deprecated, use cols_ctrl instead.
stochastic	Deprecated, use n_mis_stochastic instead.

Details

This function creates missing at random (MAR) values in the columns specified by the argument cols_mis. The probability for missing values is controlled by p. If p is a single number, then the overall probability for a value to be missing will be p in all columns of cols_mis. (Internally p will be replicated to a vector of the same length as cols_mis. So, all p[i] in the following sections will be equal to the given single number p.) Otherwise, p must be of the same length as cols_mis. In this case, the overall probability for a value to be missing will be p[i] in the column cols_mis[i]. The position of the missing values in cols_mis[i] is controlled by cols_ctrl[i]. The following procedure is applied for each pair of cols_ctrl[i] and cols_mis[i] to determine the positions of missing values:

At first, the rows of ds are divided into two groups. Therefore, the cutoff_fun calculates a cutoff value for cols_ctrl[i] (via cutoff_fun(ds[, cols_ctrl[i]], ...)). The group 1 consists of the rows, whose values in cols_ctrl[i] are below the calculated cutoff value. If the so defined group 1 is empty, the rows that have a value equal to the cutoff value will be added to this group (otherwise, these rows will belong to group 2). The group 2 consists of the remaining rows, which are not part of group 1. Now the probabilities for the rows in the two groups are set in the way that the odds are 1:x against a missing value in cols_mis[i] for the rows in group 1 compared to the rows in group 2. That means, the probability for a value to be missing in group 1 divided by the probability for a value to be missing in group 2 equals 1 divided by x. For example, for two equal

sized groups 1 and 2, ideally the number of NAs in group 1 divided by the number of NAs in group 2 should equal 1 divided by x. But there are some restrictions, which can lead to some deviations from the odds 1:x (see below).

If `x_stochastic` and `n_mis_stochastic` are false (the default), then exactly `round(nrow(ds) * p[i])` values will be set `NA` in column `cols_mis[i]`. To achieve this, it is possible that the true odds differ from 1:x. The number of observations that are deleted in group 1 and group 2 are chosen to minimize the absolute difference between the realized odds and 1:x. Furthermore, if `round(nrow(ds) * p[i]) == 0`, then no missing value will be created in `cols_mis[i]`.

If `x_stochastic` is true, the rows from the two groups will get sampling weights proportional to 1 (group 1) and x (group 2). If `n_mis_stochastic` is false, these weights are given to `sample` via the argument `prob` and exactly `round(nrow(ds) * p[i])` values will be set `NA`. If `n_mis_stochastic` is true, the sampling weights will be scaled and compared to uniform random numbers. The scaling is done in such a way to get expected `nrow(ds) * p[i]` missing values in `cols_mis[i]`.

If `p` is high and `x` is too high or too low, it is possible that the odds 1:x and the proportion of missing values `p` cannot be realized together. For example, if `p[i] = 0.9`, then a maximum of `x = 1.25` is possible (assuming that exactly 50 % of the values are below and 50 % of the values are above the cutoff value in `cols_ctrl[i]`). If a combination of `p` and `x` that cannot be realized together is given to `delete_MAR_1_to_x`, then a warning will be generated and `x` will be adjusted in such a way that `p` can be realized as given to the function. The warning can be silenced by setting the option `missMethods.warn.too.high.p` to false.

The argument `add_realized_x` controls whether the x of the realized odds are added to the return value or not. If `add_realized_x = TRUE`, then the realized x values for all `cols_mis` will be added as an attribute to the returned object. For `x_stochastic = TRUE` these realized x will differ from the given `x` most of the time and will change if the function is rerun without setting a seed. For `x_stochastic = FALSE`, it is also possible that the realized odds differ (see above). However, the realized odds will be constant over multiple runs.

Value

An object of the same class as `ds` with missing values.

Treatment of factors

If `ds[, cols_ctrl[i]]` is an unordered factor, then the concept of a cutoff value is not meaningful and cannot be applied. Instead, a combinations of the levels of the unordered factor is searched that

- guarantees at least a proportion of `prop` rows are in group 1
- minimize the difference between `prop` and the proportion of rows in group 1.

This can be seen as a binary search problem, which is solved by the solver from the package `lpSolve`, if `use_lpSolve = TRUE`. If `use_lpSolve = FALSE`, a very simple heuristic is applied. The heuristic only guarantees that at least a proportion of `prop` rows are in group 1. The choice `use_lpSolve = FALSE` is not recommend and should only be considered, if the solver of lpSolve fails.

If `ordered_as_unordered = TRUE`, then ordered factors will be treated like unordered factors and the same binary search problem will be solved for both types of factors. If `ordered_as_unordered = FALSE` (the default), then ordered factors will be grouped via `cutoff_fun` as described in Details.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

delete_MNAR_1_to_x

Other functions to create MAR: delete_MAR_censoring(), delete_MAR_one_group(), delete_MAR_rank()

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MAR_1_to_x(ds, 0.2, "X", "Y", 3)
# beware of small datasets and x_stochastic = FALSE
attr(delete_MAR_1_to_x(ds, 0.4, "X", "Y", 3, add_realized_x = TRUE), "realized_x")
attr(delete_MAR_1_to_x(ds, 0.4, "X", "Y", 4, add_realized_x = TRUE), "realized_x")
attr(delete_MAR_1_to_x(ds, 0.4, "X", "Y", 5, add_realized_x = TRUE), "realized_x")
attr(delete_MAR_1_to_x(ds, 0.4, "X", "Y", 7, add_realized_x = TRUE), "realized_x")
# p = 0.4 and 20 values -> 8 missing values, possible combinations:
# either 6 above 2 below (x = 3) or
# 7 above and 1 below (x = 7)
# Too high combination of p and x:
tryCatch(delete_MAR_1_to_x(ds, 0.9, "X", "Y", 3), warning = function(w) w)
```

---

delete_MAR_censoring *Create MAR values using a censoring mechanism*

Description

Create missing at random (MAR) values using a censoring mechanism in a data frame or a matrix

Usage

```
delete_MAR_censoring(
  ds,
  p,
  cols_mis,
  cols_ctrl,
  n_mis_stochastic = FALSE,
  where = "lower",
  sorting = TRUE,
  miss_cols,
  ctrl_cols
)
```

Arguments

ds	A data frame or matrix in which missing values will be created.
p	A numeric vector with length one or equal to length cols_mis; the probability that a value is missing.
cols_mis	A vector of column names or indices of columns in which missing values will be created.
cols_ctrl	A vector of column names or indices of columns, which controls the creation of missing values in cols_mis. Must be of the same length as cols_mis.
n_mis_stochastic	Logical, should the number of missing values be stochastic? If n_mis_stochastic = TRUE, the number of missing values for a column with missing values cols_mis[i] is a random variable with expected value nrow(ds) * p[i]. If n_mis_stochastic = FALSE, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values cols_mis[i] is round(nrow(ds) * p[i]). Possible deviations from this value, if any exists, are documented in Details.
where	Controls where missing values are created; one of "lower", "upper" or "both" (see details).
sorting	Logical; should sorting be used or a quantile as a threshold.
miss_cols	Deprecated, use cols_mis instead.
ctrl_cols	Deprecated, use cols_ctrl instead.

Details

This function creates missing at random (MAR) values in the columns specified by the argument cols_mis. The probability for missing values is controlled by p. If p is a single number, then the overall probability for a value to be missing will be p in all columns of cols_mis. (Internally p will be replicated to a vector of the same length as cols_mis. So, all p[i] in the following sections will be equal to the given single number p.) Otherwise, p must be of the same length as cols_mis. In this case, the overall probability for a value to be missing will be p[i] in the column cols_mis[i]. The position of the missing values in cols_mis[i] is controlled by cols_ctrl[i]. The following procedure is applied for each pair of cols_ctrl[i] and cols_mis[i] to determine the positions of missing values:

The default behavior (sorting = TRUE) of this function is to first sort the column cols_ctrl[i]. Then missing values in cols_mis[i] are created in the rows with the round(nrow(ds) * p[i]) smallest values. This censors approximately the proportion of p[i] rows of smallest values in cols_ctrl[i] in cols_mis[i]. Hence, the name of the function.

If where = "upper", instead of the rows with the smallest values, the rows with the highest values will be selected. For where = "both", the one half of the round(nrow(ds) * p[i]) rows with missing values will be the rows with the smallest values and the other half will be the rows with the highest values. So the censoring rows are dived to the highest and smallest values of cols_ctrl[i]. For odd round(nrow(ds) * p[i]) one more value is set NA among the smallest values.

If n_mis_stochastic = TRUE and sorting = TRUE the procedure is lightly altered. In this case, at first the floor(nrow(ds) * p[i]) rows with the smallest values (where = "lower") are set NA. If nrow(ds) * p[i] > floor(nrow(ds) * p[i]), the row with the next greater value will be set

NA with a probability to get expected nrow(ds) * p[i] missing values. For where = "upper" this "random" missing value will be the next smallest. For where = "both" this "random" missing value will be the next greatest of the smallest values.

If sorting = FALSE, the rows of ds will not be sorted. Instead, a quantile will be calculated (using quantile). If where = "lower", the quantile(ds[, cols_ctrl[i]], p[i]) will be calculated and all rows with values in ds[, cols_ctrl[i]] below this quantile will have missing values in cols_mis[i]. For where = "upper", the quantile(ds[, cols_ctrl[i]], 1 - p[i]) will be calculated and all rows with values above this quantile will have missing values. For where = "both", the quantile(ds[, cols_ctrl[i]],p[i] / 2) and quantile(ds[, cols_ctrl[i]], 1 - p[i] / 2) will be calculated. All rows with values in cols_ctrl[i] below the first quantile or above the second quantile will have missing values in cols_mis[i].

For sorting = FALSE only n_mis_stochastic = FALSE is implemented at the moment.

The option sorting = TRUE with n_mis_stochastic = FALSE will always create exactly round(nrow(ds) * p[i]) missing values in cols_mis[i]. With n_mis_stochastic = TRUE) sorting will result in floor(nrow(ds) * p[i]) or ceiling(nrow(ds) * p[i]) missing values in cols_mis[i]. For sorting = FALSE, the number of missing values will normally be close to nrow(ds) * p[i]. But for cols_ctrl with many duplicates the choice sorting = FALSE can be problematic, because of the calculation of quantile(ds[,cols_ctrl[i]], p[i]) and setting values NA below this threshold (see examples). So, in most cases sorting = TRUE is recommended.

Value

An object of the same class as ds with missing values.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

delete_MNAR_censoring

Other functions to create MAR: delete_MAR_1_to_x(), delete_MAR_one_group(), delete_MAR_rank()

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MAR_censoring(ds, 0.2, "X", "Y")
# many dupplicated values can be problematic for sorting = FALSE:
ds_many_dup <- data.frame(X = 1:20, Y = c(rep(0, 10), rep(1, 10)))
delete_MAR_censoring(ds_many_dup, 0.2, "X", "Y") # 4 NAs as expected
quantile(ds_many_dup$Y, 0.2) # 0
# No value is BELOW 0 in ds_many_dup$Y, so no missing values will be created:
delete_MAR_censoring(ds_many_dup, 0.2, "X", "Y", sorting = FALSE) # No NA!
```

`delete_MAR_one_group` *Create MAR values by deleting values in one of two groups*

Description

Create missing at random (MAR) values by deleting values in one of two groups in a data frame or a matrix

Usage

```
delete_MAR_one_group(
  ds,
  p,
  cols_mis,
  cols_ctrl,
  cutoff_fun = median,
  prop = 0.5,
  use_lpSolve = TRUE,
  ordered_as_unordered = FALSE,
  n_mis_stochastic = FALSE,
  ...,
  miss_cols,
  ctrl_cols,
  stochastic
)
```

Arguments

`ds`	A data frame or matrix in which missing values will be created.
`p`	A numeric vector with length one or equal to length `cols_mis`; the probability that a value is missing.
`cols_mis`	A vector of column names or indices of columns in which missing values will be created.
`cols_ctrl`	A vector of column names or indices of columns, which controls the creation of missing values in `cols_mis`. Must be of the same length as `cols_mis`.
`cutoff_fun`	Function that calculates the cutoff values in the `cols_ctrl`.
`prop`	Numeric of length one; (minimum) proportion of rows in group 1 (only used for unordered factors).
`use_lpSolve`	Logical; should lpSolve be used for the determination of groups, if `cols_ctrl[i]` is an unordered factor.
`ordered_as_unordered`	Logical; should ordered factors be treated as unordered factors.
`n_mis_stochastic`	Logical, should the number of missing values be stochastic? If `n_mis_stochastic = TRUE`, the number of missing values for a column with missing values `cols_mis[i]`

	is a random variable with expected value `nrow(ds) * p[i]`. If `n_mis_stochastic = FALSE`, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values `cols_mis[i]` is `round(nrow(ds) * p[i])`. Possible deviations from this value, if any exists, are documented in Details.
`...`	Further arguments passed to `cutoff_fun`.
`miss_cols`	Deprecated, use `cols_mis` instead.
`ctrl_cols`	Deprecated, use `cols_ctrl` instead.
`stochastic`	Deprecated, use `n_mis_stochastic` instead.

Details

This function creates missing at random (MAR) values in the columns specified by the argument `cols_mis`. The probability for missing values is controlled by `p`. If `p` is a single number, then the overall probability for a value to be missing will be `p` in all columns of `cols_mis`. (Internally `p` will be replicated to a vector of the same length as `cols_mis`. So, all `p[i]` in the following sections will be equal to the given single number `p`.) Otherwise, `p` must be of the same length as `cols_mis`. In this case, the overall probability for a value to be missing will be `p[i]` in the column `cols_mis[i]`. The position of the missing values in `cols_mis[i]` is controlled by `cols_ctrl[i]`. The following procedure is applied for each pair of `cols_ctrl[i]` and `cols_mis[i]` to determine the positions of missing values:

At first, the rows of `ds` are divided into two groups. Therefore, the `cutoff_fun` calculates a cutoff value for `cols_ctrl[i]` (via `cutoff_fun(ds[, cols_ctrl[i]], ...)`. The group 1 consists of the rows, whose values in `cols_ctrl[i]` are below the calculated cutoff value. If the so defined group 1 is empty, the rows that are equal to the cutoff value will be added to this group (otherwise, these rows will belong to group 2). The group 2 consists of the remaining rows, which are not part of group 1. Now one of these two groups is chosen randomly. In the chosen group, values are deleted in `cols_mis[i]`. In the other group, no missing values will be created in `cols_mis[i]`.

If `p` is too high, it is possible that a group contains not enough objects to reach `nrow(ds) * p` missing values. In this case, `p` is reduced to the maximum possible value (given the (random) group with missing data) and a warning is given. Obviously this case will occur regularly, if `p > 0.5`. Therefore, this function should normally not be called with `p > 0.5`. However, this can occur for smaller values of `p`, too (depending on the grouping). The warning can be silenced by setting the option `missMethods.warn.too.high.p` to false.

Value

An object of the same class as `ds` with missing values.

Treatment of factors

If `ds[, cols_ctrl[i]]` is an unordered factor, then the concept of a cutoff value is not meaningful and cannot be applied. Instead, a combinations of the levels of the unordered factor is searched that

- guarantees at least a proportion of `prop` rows are in group 1
- minimize the difference between `prop` and the proportion of rows in group 1.

This can be seen as a binary search problem, which is solved by the solver from the package `lpSolve`, if `use_lpSolve = TRUE`. If `use_lpSolve = FALSE`, a very simple heuristic is applied. The heuristic only guarantees that at least a proportion of `prop` rows are in group 1. The choice `use_lpSolve = FALSE` is not recommend and should only be considered, if the solver of lpSolve fails.

If `ordered_as_unordered = TRUE`, then ordered factors will be treated like unordered factors and the same binary search problem will be solved for both types of factors. If `ordered_as_unordered = FALSE` (the default), then ordered factors will be grouped via `cutoff_fun` as described in Details.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

`delete_MNAR_one_group`

Other functions to create MAR: `delete_MAR_1_to_x()`, `delete_MAR_censoring()`, `delete_MAR_rank()`

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MAR_one_group(ds, 0.2, "X", "Y")
```

---

delete_MAR_rank *Create MAR values using a ranking mechanism*

Description

Create missing at random (MAR) values using a ranking mechanism in a data frame or a matrix

Usage

```
delete_MAR_rank(
  ds,
  p,
  cols_mis,
  cols_ctrl,
  n_mis_stochastic = FALSE,
  ties.method = "average",
  miss_cols,
  ctrl_cols
)
```

Arguments

ds	A data frame or matrix in which missing values will be created.
p	A numeric vector with length one or equal to length cols_mis; the probability that a value is missing.
cols_mis	A vector of column names or indices of columns in which missing values will be created.
cols_ctrl	A vector of column names or indices of columns, which controls the creation of missing values in cols_mis. Must be of the same length as cols_mis.
n_mis_stochastic	Logical, should the number of missing values be stochastic? If n_mis_stochastic = TRUE, the number of missing values for a column with missing values cols_mis[i] is a random variable with expected value nrow(ds) * p[i]. If n_mis_stochastic = FALSE, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values cols_mis[i] is round(nrow(ds) * p[i]). Possible deviations from this value, if any exists, are documented in Details.
ties.method	How ties are handled. Passed to rank.
miss_cols	Deprecated, use cols_mis instead.
ctrl_cols	Deprecated, use cols_ctrl instead.

Details

This function creates missing at random (MAR) values in the columns specified by the argument cols_mis. The probability for missing values is controlled by p. If p is a single number, then the overall probability for a value to be missing will be p in all columns of cols_mis. (Internally p will be replicated to a vector of the same length as cols_mis. So, all p[i] in the following sections will be equal to the given single number p.) Otherwise, p must be of the same length as cols_mis. In this case, the overall probability for a value to be missing will be p[i] in the column cols_mis[i]. The position of the missing values in cols_mis[i] is controlled by cols_ctrl[i]. The following procedure is applied for each pair of cols_ctrl[i] and cols_mis[i] to determine the positions of missing values:

At first, the probability for a value to be missing is calculated. This probability for a missing value in a row of cols_mis[i] is proportional to the rank of the value in cols_ctrl[i] in the same row. If n_mis_stochastic = FALSE these probabilities are given to the prob argument of sample. If n_mis_stochastic = TRUE, they are scaled to sum up to nrow(ds) * p[i]. Then for each probability a uniformly distributed random number is generated. If this random number is less than the probability, the value in cols_mis[i] is set NA.

The ranks are calculated via rank. The argument ties.method is directly passed to this function. Possible choices for ties.method are documented in rank.

For high values of p it is mathematically not possible to get probabilities proportional to the ranks. In this case, a warning is given. This warning can be silenced by setting the option missMethods.warn.too.high.p to false.

Value

An object of the same class as ds with missing values.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

rank, delete_MNAR_rank

Other functions to create MAR: delete_MAR_1_to_x(), delete_MAR_censoring(), delete_MAR_one_group()

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MAR_rank(ds, 0.2, "X", "Y")
```

---

delete_MCAR *Create MCAR values*

Description

Create missing completely at random (MCAR) values in a data frame or a matrix

Usage

```
delete_MCAR(
  ds,
  p,
  cols_mis = seq_len(ncol(ds)),
  n_mis_stochastic = FALSE,
  p_overall = FALSE,
  miss_cols,
  stochastic
)
```

Arguments

ds	A data frame or matrix in which missing values will be created.
p	A numeric vector with length one or equal to length cols_mis; the probability that a value is missing.
cols_mis	A vector of column names or indices of columns in which missing values will be created.
n_mis_stochastic	Logical, should the number of missing values be stochastic? If n_mis_stochastic = TRUE, the number of missing values for a column with missing values cols_mis[i] is a random variable with expected value nrow(ds) * p[i]. If n_mis_stochastic = FALSE, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values cols_mis[i] is round(nrow(ds) * p[i]). Possible deviations from this value, if any exists, are documented in Details.

p_overall	Logical; see details.
miss_cols	Deprecated, use cols_mis instead.
stochastic	Deprecated, use n_mis_stochastic instead.

Details

This function creates missing completely at random (MCAR) values in the dataset ds. The proportion of missing values is specified with p. The columns in which missing values are created can be set via cols_mis. If cols_mis is not specified, then missing values are created in all columns of ds.

The probability for missing values is controlled by p. If p is a single number, then the overall probability for a value to be missing will be p in all columns of cols_mis. (Internally p will be replicated to a vector of the same length as cols_mis. So, all p[i] in the following sections will be equal to the given single number p.) Otherwise, p must be of the same length as cols_mis. In this case, the overall probability for a value to be missing will be p[i] in the column cols_mis[i].

If n_mis_stochastic = FALSE and p_overall = FALSE (the default), then exactly round(nrow(ds) * p[i]) values will be set NA in column cols_mis[i]. If n_mis_stochastic = FALSE and p_overall = TRUE, then p must be of length one and exactly round(nrow(ds) * p * length(cols_mis)) values will be set NA (over all columns in cols_mis). This can result in a proportion of missing values in every miss_col unequal to p, but the proportion of missing values in all columns together will be close to p.

If n_mis_stochastic = TRUE, then each value in column cols_mis[i] has probability p[i] to be missing (independently of all other values). Therefore, the number of missing values in cols_mis[i] is a random variable with a binomial distribution B(nrow(ds), p[i]). This can (and will most of the time) lead to more or less missing values than round(nrow(ds) * p[i]) in column cols_mis[i]. If n_mis_stochastic = TRUE, then the argument p_overall is ignored because it is superfluous.

Value

An object of the same class as ds with missing values.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MCAR(ds, 0.2)
```

`delete_MNAR_1_to_x` *Create MNAR values using MNAR1:x*

Description

Create missing not at random (MNAR) values using MNAR1:x in a data frame or a matrix

Usage

```
delete_MNAR_1_to_x(
  ds,
  p,
  cols_mis,
  x,
  cutoff_fun = median,
  prop = 0.5,
  use_lpSolve = TRUE,
  ordered_as_unordered = FALSE,
  n_mis_stochastic = FALSE,
  x_stochastic = FALSE,
  add_realized_x = FALSE,
  ...,
  miss_cols,
  stochastic
)
```

Arguments

`ds`	A data frame or matrix in which missing values will be created.
`p`	A numeric vector with length one or equal to length `cols_mis`; the probability that a value is missing.
`cols_mis`	A vector of column names or indices of columns in which missing values will be created.
`x`	Numeric with length one ($0 < x <$ `Inf`); odds are 1 to x for the probability of a value to be missing in group 1 against the probability of a value to be missing in group 2 (see details).
`cutoff_fun`	Function that calculates the cutoff values in the `cols_ctrl`.
`prop`	Numeric of length one; (minimum) proportion of rows in group 1 (only used for unordered factors).
`use_lpSolve`	Logical; should lpSolve be used for the determination of groups, if `cols_ctrl[i]` is an unordered factor.
`ordered_as_unordered`	Logical; should ordered factors be treated as unordered factors.

`n_mis_stochastic`
: Logical, should the number of missing values be stochastic? If `n_mis_stochastic = TRUE`, the number of missing values for a column with missing values `cols_mis[i]` is a random variable with expected value `nrow(ds) * p[i]`. If `n_mis_stochastic = FALSE`, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values `cols_mis[i]` is `round(nrow(ds) * p[i])`. Possible deviations from this value, if any exists, are documented in Details.

`x_stochastic`
: Logical; should the odds be stochastic or deterministic.

`add_realized_x`
: Logical; if TRUE the realized odds for cols_mis will be returned (as attribute).

`...`
: Further arguments passed to `cutoff_fun`.

`miss_cols`
: Deprecated, use `cols_mis` instead.

`stochastic`
: Deprecated, use `n_mis_stochastic` instead.

Details

The functions `delete_MNAR_1_to_x` and `delete_MAR_1_to_x` are sisters. The only difference between these two functions is the column that controls the generation of missing values. In `delete_MAR_1_to_x` a separate column `cols_ctrl[i]` controls the generation of missing values in `cols_mis[i]`. In contrast, in `delete_MNAR_1_to_x` the generation of missing values in `cols_mis[i]` is controlled by `cols_mis[i]` itself. All other aspects are identical for both functions. Therefore, further details can be found in `delete_MAR_1_to_x`.

Value

An object of the same class as `ds` with missing values.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

`delete_MAR_1_to_x`

Other functions to create MNAR: `delete_MNAR_censoring()`, `delete_MNAR_one_group()`, `delete_MNAR_rank()`

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MNAR_1_to_x(ds, 0.2, "X", x = 3)
```

`delete_MNAR_censoring` *Create MNAR values using a censoring mechanism*

Description

Create missing not at random (MNAR) values using a censoring mechanism in a data frame or a matrix

Usage

```
delete_MNAR_censoring(
  ds,
  p,
  cols_mis,
  n_mis_stochastic = FALSE,
  where = "lower",
  sorting = TRUE,
  miss_cols
)
```

Arguments

`ds`	A data frame or matrix in which missing values will be created.
`p`	A numeric vector with length one or equal to length `cols_mis`; the probability that a value is missing.
`cols_mis`	A vector of column names or indices of columns in which missing values will be created.
`n_mis_stochastic`	Logical, should the number of missing values be stochastic? If `n_mis_stochastic = TRUE`, the number of missing values for a column with missing values `cols_mis[i]` is a random variable with expected value `nrow(ds) * p[i]`. If `n_mis_stochastic = FALSE`, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values `cols_mis[i]` is `round(nrow(ds) * p[i])`. Possible deviations from this value, if any exists, are documented in Details.
`where`	Controls where missing values are created; one of "lower", "upper" or "both" (see details).
`sorting`	Logical; should sorting be used or a quantile as a threshold.
`miss_cols`	Deprecated, use `cols_mis` instead.

Details

The functions `delete_MNAR_censoring` and `delete_MAR_censoring` are sisters. The only difference between these two functions is the column that controls the generation of missing values. In `delete_MAR_censoring` a separate column `cols_ctrl[i]` controls the generation of missing values in `cols_mis[i]`. In contrast, in `delete_MNAR_censoring` the generation of missing values in

`cols_mis[i]` is controlled by `cols_mis[i]` itself. All other aspects are identical for both functions. Therefore, further details can be found in `delete_MAR_censoring`.

Value

An object of the same class as `ds` with missing values.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

`delete_MAR_censoring`

Other functions to create MNAR: `delete_MNAR_1_to_x()`, `delete_MNAR_one_group()`, `delete_MNAR_rank()`

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MNAR_censoring(ds, 0.2, "X")
```

---

delete_MNAR_one_group *Create MNAR values by deleting values in one of two groups*

Description

Create missing not at random (MNAR) values by deleting values in one of two groups in a data frame or a matrix

Usage

```
delete_MNAR_one_group(
  ds,
  p,
  cols_mis,
  cutoff_fun = median,
  prop = 0.5,
  use_lpSolve = TRUE,
  ordered_as_unordered = FALSE,
  n_mis_stochastic = FALSE,
  ...,
  miss_cols,
  stochastic
)
```

Arguments

ds	A data frame or matrix in which missing values will be created.
p	A numeric vector with length one or equal to length cols_mis; the probability that a value is missing.
cols_mis	A vector of column names or indices of columns in which missing values will be created.
cutoff_fun	Function that calculates the cutoff values in the cols_ctrl.
prop	Numeric of length one; (minimum) proportion of rows in group 1 (only used for unordered factors).
use_lpSolve	Logical; should lpSolve be used for the determination of groups, if cols_ctrl[i] is an unordered factor.
ordered_as_unordered	Logical; should ordered factors be treated as unordered factors.
n_mis_stochastic	Logical, should the number of missing values be stochastic? If n_mis_stochastic = TRUE, the number of missing values for a column with missing values cols_mis[i] is a random variable with expected value nrow(ds) * p[i]. If n_mis_stochastic = FALSE, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values cols_mis[i] is round(nrow(ds) * p[i]). Possible deviations from this value, if any exists, are documented in Details.
...	Further arguments passed to cutoff_fun.
miss_cols	Deprecated, use cols_mis instead.
stochastic	Deprecated, use n_mis_stochastic instead.

Details

The functions delete_MNAR_one_group and delete_MAR_one_group are sisters. The only difference between these two functions is the column that controls the generation of missing values. In delete_MAR_one_group a separate column cols_ctrl[i] controls the generation of missing values in cols_mis[i]. In contrast, in delete_MNAR_one_group the generation of missing values in cols_mis[i] is controlled by cols_mis[i] itself. All other aspects are identical for both functions. Therefore, further details can be found in delete_MAR_one_group.

Value

An object of the same class as ds with missing values.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

delete_MAR_one_group

Other functions to create MNAR: delete_MNAR_1_to_x(), delete_MNAR_censoring(), delete_MNAR_rank()

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MNAR_one_group(ds, 0.2, "X")
```

`delete_MNAR_rank` *Create MNAR values using a ranking mechanism*

Description

Create missing not at random (MNAR) values using a ranking mechanism in a data frame or a matrix

Usage

```
delete_MNAR_rank(
  ds,
  p,
  cols_mis,
  n_mis_stochastic = FALSE,
  ties.method = "average",
  miss_cols
)
```

Arguments

`ds`	A data frame or matrix in which missing values will be created.
`p`	A numeric vector with length one or equal to length `cols_mis`; the probability that a value is missing.
`cols_mis`	A vector of column names or indices of columns in which missing values will be created.
`n_mis_stochastic`	Logical, should the number of missing values be stochastic? If `n_mis_stochastic = TRUE`, the number of missing values for a column with missing values `cols_mis[i]` is a random variable with expected value `nrow(ds) * p[i]`. If `n_mis_stochastic = FALSE`, the number of missing values will be deterministic. Normally, the number of missing values for a column with missing values `cols_mis[i]` is `round(nrow(ds) * p[i])`. Possible deviations from this value, if any exists, are documented in Details.
`ties.method`	How ties are handled. Passed to `rank`.
`miss_cols`	Deprecated, use `cols_mis` instead.

Details

The functions delete_MNAR_rank and delete_MAR_rank are sisters. The only difference between these two functions is the column that controls the generation of missing values. In delete_MAR_rank a separate column cols_ctrl[i] controls the generation of missing values in cols_mis[i]. In contrast, in delete_MNAR_rank the generation of missing values in cols_mis[i] is controlled by cols_mis[i] itself. All other aspects are identical for both functions. Therefore, further details can be found in delete_MAR_rank.

Value

An object of the same class as ds with missing values.

References

Santos, M. S., Pereira, R. C., Costa, A. F., Soares, J. P., Santos, J., & Abreu, P. H. (2019). Generating Synthetic Missing Data: A Review by Missing Mechanism. *IEEE Access*, 7, 11651-11667

See Also

delete_MAR_rank

Other functions to create MNAR: delete_MNAR_1_to_x(), delete_MNAR_censoring(), delete_MNAR_one_group()

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
delete_MNAR_rank(ds, 0.2, "X")
```

---

evaluate_imputation_parameters

Evaluate estimated parameters after imputation

Description

Compares estimated parameters after imputation to true parameters or estimates based on the original dataset

Usage

```
evaluate_imputation_parameters(
  ds_imp,
  ds_orig = NULL,
  pars_true = NULL,
  parameter = "mean",
  criterion = "RMSE",
  cols_which = seq_len(ncol(ds_imp)),
  tolerance = sqrt(.Machine$double.eps),
  ...,
```

```
  imp_ds,
  true_pars,
  which_cols
)
```

Arguments

ds_imp	A data frame or matrix with imputed values.
ds_orig	A data frame or matrix with original (true) values.
pars_true	True parameters, normally a vector or a matrix.
parameter	A string specifying the estimated parameters for comparison.
criterion	A string specifying the used criterion for comparing the imputed and original values.
cols_which	Indices or names of columns used for evaluation.
tolerance	Numeric, only used for criterion = "precision": numeric differences smaller than tolerance are treated as zero/equal.
...	Further arguments passed to the function for parameter estimation.
imp_ds	Deprecated, renamed to ds_imp.
true_pars	Deprecated, renamed to pars_true.
which_cols	Deprecated, renamed to cols_which.

Details

Either ds_orig or pars_true must be supplied and the other one must be NULL (default: both are NULL, just supply one, see examples). The following parameters are implemented: "mean", "median", "var", "sd", "quantile", "cov", "cov_only", cor", "cor_only". Some details follow:

- "var", "cov" and "cov_only": For "var" only the variances of the columns (the diagonal elements of the covariance matrix) are compared. For "cov" the whole covariance matrix is compared. For "cov_only" only the upper triangle (excluding the diagonal) of the covariance matrix is compared.
- "cor", "cor_only": For "cor" the whole correlation matrix is compared. For "cor_only" only the upper triangle (excluding the diagonal) of the correlation matrix is compared.
- "quantile": the quantiles can be set via the additional argument probs (see examples). Otherwise, the default quantiles from quantile will be used.

The argument cols_which allows the selection of columns for comparison (see examples). If pars_true is used, it is assumed that only relevant parameters are supplied (see examples).

Possible choices for the argument criterion are documented in evaluate_imputed_values

Value

A numeric vector of length one.

References

Cetin-Berber, D. D., Sari, H. I., & Huggins-Manley, A. C. (2019). Imputation Methods to Deal With Missing Responses in Computerized Adaptive Multistage Testing. *Educational and psychological measurement*, 79(3), 495-511.

See Also

Other evaluation functions: `evaluate_imputed_values()`, `evaluate_parameters()`

Examples

```
# only ds_orig known
ds_orig <- data.frame(X = 1:10, Y = 101:101)
ds_imp <- impute_mean(delete_MCAR(ds_orig, 0.4))
evaluate_imputation_parameters(ds_imp, ds_orig = ds_orig)

# true parameters known
ds_orig <- data.frame(X = rnorm(100), Y = rnorm(100, mean = 10))
ds_imp <- impute_mean(delete_MCAR(ds_orig, 0.3))
evaluate_imputation_parameters(ds_imp, pars_true = c(0, 10), parameter = "mean")
evaluate_imputation_parameters(ds_imp, pars_true = c(1, 1), parameter = "var")

# set quantiles
evaluate_imputation_parameters(ds_imp,
  pars_true = c(qnorm(0.3), qnorm(0.3, mean = 10)),
  parameter = "quantile", probs = 0.3
)

# compare only column Y
evaluate_imputation_parameters(ds_imp,
  pars_true = c(Y = 10), parameter = "mean",
  cols_which = "Y"
)
```

---

evaluate_imputed_values

Evaluate imputed values

Description

Compare imputed to true values

Usage

```
evaluate_imputed_values(
  ds_imp,
  ds_orig,
  criterion = "RMSE",
  M = NULL,
```

```
  cols_which = seq_len(ncol(ds_imp)),
  tolerance = sqrt(.Machine$double.eps),
  imp_ds,
  orig_ds,
  which_cols
)
```

Arguments

ds_imp	A data frame or matrix with imputed values.
ds_orig	A data frame or matrix with original (true) values.
criterion	A string specifying the used criterion for comparing the imputed and original values.
M	NULL (the default) or a missing data indicator matrix. The missing data indicator matrix is normally created via `is.na(ds_mis)`, where `ds_mis` is the dataset after deleting values from `ds_orig`.
cols_which	Indices or names of columns used for evaluation.
tolerance	Numeric, only used for `criterion = "precision"`: numeric differences smaller than tolerance are treated as zero/equal.
imp_ds	Deprecated, renamed to `ds_imp`.
orig_ds	Deprecated, renamed to `ds_orig`.
which_cols	Deprecated, renamed to `cols_which`.

Details

The following `criterion`s are implemented to compare the imputed values to the true values:

- "RMSE" (the default): The Root Mean Squared Error between the imputed and true values
- "bias": The mean difference between the imputed and the true values
- "cor": The correlation between the imputed and true values
- "MAE": The Mean Absolute Error between the imputed and true values
- "MSE": The Mean Squared Error between the imputed and true values
- "NRMSE_col_mean": For every column the RMSE divided by the mean of the true values is calculated. Then these columnwise values are squared and averaged. Finally, the square root of this average is returned.
- "NRMSE_col_mean_sq": For every column the RMSE divided by the square root of the mean of the squared true values is calculated. Then these columnwise values are squared and averaged. Finally, the square root of this average is returned.
- "NRMSE_col_sd": For every column the RMSE divided by the standard deviation of all true values is calculated. Then these columnwise values are squared and averaged. Finally, the square root of this average is returned.
- "NRMSE_tot_mean": RMSE divided by the mean of all true values
- "NRMSE_tot_mean_sq": RMSE divided by the square root of the mean of all squared true values

- "NRMSE_tot_sd": RMSE divided by the standard deviation of all true values
- "nr_equal": number of imputed values that are equal to the true values
- "nr_NA": number of values in ds_imp that are NA (not imputed)
- "precision": proportion of imputed values that are equal to the true values

Additionally there are relative versions of bias and MAE implemented. In the relative versions, the differences are divided by the absolute values of the true values. These relative versions can be selected via "bias_rel" and "MAE_rel". The "NRMSE_tot_" and "NRMSE_col_" are equal, if the columnwise normalization values are equal to the total normalization value (see examples).

The argument cols_which allows the selection of columns for comparison (see examples).

If M = NULL (the default), then all values of ds_imp and ds_orig will be used for the calculation of the evaluation criterion. If a missing data indicator matrix is given via M, only the truly imputed values (values that are marked as missing via M) will be used for the calculation. If you want to provide M, M must be a logical matrix of the same dimensions as ds_orig and missing values must be coded as TRUE. This is the standard behavior, if you use is.na on a dataset with missing values to generate M (see examples). It is possible to combine M and cols_which.

Value

A numeric vector of length one.

References

Kim, H., Golub, G. H., & Park, H. (2005). Missing value estimation for DNA microarray gene expression data: local least squares imputation. *Bioinformatics*, 21(2), 187-198.

See Also

Other evaluation functions: evaluate_imputation_parameters(), evaluate_parameters()

Examples

```
ds_orig <- data.frame(X = 1:10, Y = 101:110)
ds_mis <- delete_MCAR(ds_orig, 0.3)
ds_imp <- impute_mean(ds_mis)
# compare all values from ds_orig and ds_imp
evaluate_imputed_values(ds_imp, ds_orig)
# compare only the imputed values
M <- is.na(ds_mis)
evaluate_imputed_values(ds_imp, ds_orig, M = M)
# compare only the imputed values in column X
evaluate_imputed_values(ds_imp, ds_orig, M = M, cols_which = "X")

# NRMSE_tot_mean and NRMSE_col_mean are equal, if columnwise means are equal
ds_orig <- data.frame(X = 1:10, Y = 10:1)
ds_mis <- delete_MCAR(ds_orig, 0.3)
ds_imp <- impute_mean(ds_mis)
evaluate_imputed_values(ds_imp, ds_orig, "NRMSE_tot_mean")
evaluate_imputed_values(ds_imp, ds_orig, "NRMSE_col_mean")
```

`evaluate_parameters` *Evaluate estimated parameters*

Description

Compare estimated parameters to true parameters

Usage

```
evaluate_parameters(
  pars_est,
  pars_true,
  criterion = "RMSE",
  tolerance = sqrt(.Machine$double.eps),
  est_pars,
  true_pars
)
```

Arguments

`pars_est`	A vector or matrix of estimated parameters.
`pars_true`	True parameters, normally a vector or a matrix.
`criterion`	A string specifying the used criterion for comparing the imputed and original values.
`tolerance`	Numeric, only used for `criterion = "precision"`: numeric differences smaller than tolerance are treated as zero/equal.
`est_pars`	Deprecated, renamed to `pars_est`.
`true_pars`	Deprecated, renamed to `pars_true`.

Details

The same `criterions` are implemented for `evaluate_parameters` and `evaluate_imputed_values`. The possible choices are documented in `evaluate_imputed_values`.

Value

A numeric vector of length one.

References

Cetin-Berber, D. D., Sari, H. I., & Huggins-Manley, A. C. (2019). Imputation Methods to Deal With Missing Responses in Computerized Adaptive Multistage Testing. *Educational and psychological measurement*, 79(3), 495-511.

See Also

Other evaluation functions: `evaluate_imputation_parameters()`, `evaluate_imputed_values()`

Examples

```
evaluate_parameters(1:4, 2:5, "RMSE")
```

`impute_EM` *EM imputation*

Description

Impute missing values in a data frame or a matrix using parameters estimated via EM

Usage

```
impute_EM(
  ds,
  stochastic = TRUE,
  maxits = 1000,
  criterion = 1e-04,
  verbose = FALSE
)
```

Arguments

`ds`	A data frame or matrix with missing values.
`stochastic`	Logical; see details.
`maxits`	Maximum number of iterations for the EM, passed to `norm::em.norm()`.
`criterion`	If maximum relative difference in parameter estimates is below this threshold, the EM algorithm stops. Argument is directly passed to `norm::em.norm()`.
`verbose`	Should messages be given for special cases (see details)?

Details

At first parameters are estimated via `norm::em.norm()`. Then these parameters are used in regression like models to impute the missing values. If `stochachstic = FALSE`, the expected values (given the observed values and the estimated parameters via EM) are imputed for the missing values of an object. If `stochastic = TRUE`, residuals from a multivariate normal distribution are added to these expected values.

If all values in a row are `NA` or the required part of the covariance matrix for the calculation of the expected values is not invertible, parts of the estimated mean vector will be imputed. If `stochastic = TRUE`, residuals will be added to these values. If `verbose = TRUE`, a message will be given for these rows.

Value

An object of the same class as `ds` with imputed missing values.

The number of EM iterations are added as an attribute (`iterations`).

See Also

- norm::em.norm(), which estimates the parameters
- impute_expected_values(), which calculates the imputation values

Examples

```
ds_orig <- mvtnorm::rmvnorm(100, rep(0, 7))
ds_mis <- delete_MCAR(ds_orig, p = 0.2)
ds_imp <- impute_EM(ds_mis, stochastic = FALSE)
```

---

impute_expected_values

Impute expected values

Description

Impute the missing values with expected values given the observed values and estimated parameters assuming a multivariate normal distribution

Usage

```
impute_expected_values(
  ds,
  mu,
  S,
  stochastic = FALSE,
  M = is.na(ds),
  verbose = FALSE
)
```

Arguments

ds	A data frame or matrix with missing values.
mu	Vector of means for the variables.
S	Covariance matrix of the variables.
stochastic	Logical, should residuals be added to the expected values.
M	Missing data indicator matrix.
verbose	Should messages be given for special cases (see details)?

Details

Normally, this function is called by other imputation function and should not be called directly. The function imputes the missing values assuming a multivariate normal distribution. This is equivalent to imputing the least squares estimate of the missing values in some kind of way.

If no values is observed in a row or a relevant submatrix of the covariance matrix (`S_22`) is not invertible, the missing values are imputed with (parts of) `mu` (plus a residuum, if `stochastich = TRUE`). If `verbose = TRUE`, these cases will be listed in a message. Otherwise, they will be imputed silently.

Value

An object of the same class as `ds` with imputed missing values.

Examples

```
ds_orig <- mvtnorm::rmvnorm(100, rep(0, 2))
ds_mis <- delete_MCAR(ds_orig, p = 0.2)
# impute using true parameters:
ds_imp <- impute_expected_values(ds_mis, mu = c(0, 0), diag(1, 2))
```

---

impute_hot_deck_in_classes

Hot deck imputation in imputation classes

Description

Impute missing values in a data frame or a matrix using a hot deck within imputation classes

Usage

```
impute_hot_deck_in_classes(
  ds,
  cols_class,
  type = "cols_seq",
  breaks = Inf,
  use_quantiles = FALSE,
  min_objs_in_class = 1,
  min_obs_comp = 0,
  min_obs_per_col = 1,
  donor_limit = Inf,
  add_imputation_classes = FALSE
)
```

Arguments

ds	A data frame or matrix with missing values.
cols_class	Columns that are used for constructing the imputation classes.
type	The type of hot deck (for details, see impute_sRHD()).
breaks	Number of intervals / levels a column is broken into (see cut(), which is used internally for cutting numeric columns). If breaks = Inf (the default), every unique value of a column can be in a separate class (if no other restrictions apply).
use_quantiles	Should quantiles be used for cutting numeric vectors? Normally, cut() divides the range of an vector into equal spaced intervals. If use_quantiles = TRUE, the classes will be of roughly equal content.
min_objs_in_class	Minimum number of objects (rows) in an imputation class.
min_obs_comp	Minimum number of completely observed objects (rows) in an imputation class.
min_obs_per_col	Minimum number of observed values in every column of an imputation class.
donor_limit	Minimum odds between incomplete and complete values in a column, if type = cols_seq. If type = sim_comp, minimum odds between incomplete and complete rows. For type = sim_part the donor limit option is not implemented and donor_limit should be Inf.
add_imputation_classes	Should imputation classes be added as attributes to the imputed dataset?

Details

This function is a combination of impute_in_classes() and impute_sRHD(). It applies impute_sRHD() inside of imputation classes (adjustment cells), which are constructed via impute_in_classes(). More details can be found in these two functions.

Value

An object of the same class as ds with imputed missing values.

References

Andridge, R.R. and Little, R.J.A. (2010), A Review of Hot Deck Imputation for Survey Non-response. International Statistical Review, 78: 40-64. doi:10.1111/j.1751-5823.2010.00103.x

See Also

impute_in_classes(), which is used for the construction of the imputation classes.

impute_sRHD(), which is used for the imputation.

Examples

```
impute_hot_deck_in_classes(data.frame(
  X = c(rep("A", 10), rep("B", 10)),
  Y = c(rep(NA, 5), 106:120)
),
"X",
donor_limit = 1
)
```

`impute_in_classes` *Impute in classes*

Description

Apply an imputation function inside imputation classes

Usage

```
impute_in_classes(
  ds,
  cols_class,
  FUN,
  breaks = Inf,
  use_quantiles = FALSE,
  min_objs_in_class = 1,
  min_obs_comp = 0,
  min_obs_per_col = 1,
  donor_limit = Inf,
  dl_type = "cols_seq",
  add_imputation_classes = FALSE,
  ...
)
```

Arguments

`ds`	A data frame or matrix with missing values.
`cols_class`	Columns that are used for constructing the imputation classes.
`FUN`	An imputation function that is applied to impute the missing values.
`breaks`	Number of intervals / levels a column is broken into (see `cut()`, which is used internally for cutting numeric columns). If `breaks = Inf` (the default), every unique value of a column can be in a separate class (if no other restrictions apply).
`use_quantiles`	Should quantiles be used for cutting numeric vectors? Normally, `cut()` divides the range of an vector into equal spaced intervals. If `use_quantiles = TRUE`, the classes will be of roughly equal content.

`min_objs_in_class`
: Minimum number of objects (rows) in an imputation class.

`min_obs_comp`
: Minimum number of completely observed objects (rows) in an imputation class.

`min_obs_per_col`
: Minimum number of observed values in every column of an imputation class.

`donor_limit`
: Minimum odds between incomplete and complete values in a column, if `dl_type = cols_seq`. If `dl_type = sim_comp`, minimum odds between incomplete and complete rows.

`dl_type`
: See `donor_limit`.

`add_imputation_classes`
: Should imputation classes be added as attributes to the imputed dataset?

`...`
: Arguments passed to `FUN`.

Details

Imputation classes (sometimes also called adjustment cells) are build using cross-validation of all `cols_class`. The classes are collapsed, if they do not satisfy all of the criteria defined by `min_objs_in_class`, `min_obs_comp`, `min_obs_per_col` and `donor_limit`. Collapsing starts from the last value of `cols_class`. Internally, a mixture of collapsing and early stopping is used for the construction of the classes.

Value

An object of the same class as `ds` with imputed missing values.

References

Andridge, R.R. and Little, R.J.A. (2010), A Review of Hot Deck Imputation for Survey Non-response. International Statistical Review, 78: 40-64. doi:10.1111/j.1751-5823.2010.00103.x

Examples

```
# Mean imputation in classes
impute_in_classes(data.frame(X = 1:5, Y = c(NA, 12:15)), "X",
  impute_mean,
  min_obs_per_col = 2
)
```

`impute_LS_adaptive` *LSimpute_adaptive*

Description

Perform LSimpute_adaptive as described by Bo et al. (2004)

Usage

```
impute_LS_adaptive(
  ds,
  k = 10,
  eps = 1e-06,
  min_common_obs = 5,
  r_max_min = 100,
  p_mis_sim = 0.05,
  warn_r_max = FALSE,
  verbose_gene = FALSE,
  verbose_array = FALSE,
  verbose_gene_p = FALSE,
  verbose_array_p = FALSE
)
```

Arguments

ds	A data frame or matrix with missing values.
k	Directly passed to impute_LS_gene().
eps	Directly passed to impute_LS_gene().
min_common_obs	Directly passed to impute_LS_gene().
r_max_min	Minimum number of nearest genes used for imputation. The default value (100) corresponds to the choice of Bo et al. (2004).
p_mis_sim	Percentage of observed values that are set NA to estimate the mixing coefficient p. The default value (0.05) corresponds to the choice of Bo et al. (2004).
warn_r_max	Should a warning be given, if r_max_min is set too high?
verbose_gene	Should impute_LS_gene() be verbose?
verbose_array	Should impute_LS_array() be verbose?
verbose_gene_p	Should impute_LS_gene() be verbose while estimating p?
verbose_array_p	Should impute_LS_array() be verbose while estimating p?

Details

This function performs LSimpute_adaptive as described by Bo et al. (2004).The function assumes that the genes are the rows of ds.

LSimpute_adaptive combines imputation values from impute_LS_gene() and impute_LS_array() using a local (adaptive) approach for the mixing coefficient p.

If the dataset is too small or has too many missing values, there are some fallback systems implemented. First, if ncol(ds) <= min_common_obs (normally, this should not the case!), values are imputed through impute_LS_array(). Second, r_max_min is automatically adjusted, if it is too high. In this case, a warning will be given, if warn_r_max = TRUE. Third, if there are not enough observed values in a row (less than min_common_obs), the calculation of the mixing coefficient is not possible and missing values of these rows are imputed with the values from impute_LS_array().

The amount of feedback given from `impute_LS_gene()` and `impute_LS_array()` is controlled via `verbose_gene`, `verbose_array`, `verbose_gene_p` and `verbose_array_p`. The last two control the amount of feedback while estimating p and the first two the amount of feedback during the estimation of the values that are mixed with p. Internally, the imputed dataset from `impute_LS_gene()` is passed on to `impute_LS_array()`. Therefore, all messages from `impute_LS_gene()` are truly from `impute_LS_gene()` and not a part of `impute_LS_array()`, which never calls `impute_LS_gene()` in this case. Furthermore, all messages from `impute_expected_values()` belong to `impute_LS_array()`.

Value

An object of the same class as `ds` with imputed missing values.

References

Bo, T. H., Dysvik, B., & Jonassen, I. (2004). LSimpute: accurate estimation of missing values in microarray data with least squares methods. Nucleic acids research, 32(3), e34

See Also

Other LSimpute functions: `impute_LS_array()`, `impute_LS_combined()`, `impute_LS_gene()`

Examples

```
set.seed(123)
ds_mis <- delete_MCAR(mvtnorm::rmvnorm(100, rep(0, 10)), 0.1)
ds_imp <- impute_LS_adaptive(ds_mis)
```

`impute_LS_array` *LSimpute_array*

Description

Perform LSimpute_array as described by Bo et al. (2004)

Usage

```
impute_LS_array(
  ds,
  k = 10,
  eps = 1e-06,
  min_common_obs = 5,
  ds_impute_LS_gene = NULL,
  verbose_gene = FALSE,
  verbose_expected_values = FALSE
)
```

Arguments

ds	A data frame or matrix with missing values.
k	Directly passed to impute_LS_gene().
eps	Directly passed to impute_LS_gene().
min_common_obs	Directly passed to impute_LS_gene().
ds_impute_LS_gene	Result of imputing ds with ds_impute_LS_gene(), if this already exists (see details).
verbose_gene	Should impute_LS_gene() be verbose?
verbose_expected_values	Should impute_expected_values() be verbose?

Details

This function performs LSimpute_array as described by Bo et al. (2004).The function assumes that the genes are the rows of ds.

The mean vector and covariance matrix for the imputation in LSimpute_array is based on a imputed dataset from LSimpute_gene. This dataset can be supplied directly via ds_impute_LS_gene or will automatically be created with impute_LS_gene() (if ds_impute_LS_gene is NULL). The imputation values are the expected values given the estimated parameters and the observed values. They are calculated via impute_expected_values(). The amount of feedback from these two functions is controlled via verbose_gene and verbose_expected_values. The values of these two arguments are passed on to the argument verbose from impute_LS_gene() and impute_expected_values().

Value

An object of the same class as ds with imputed missing values.

References

Bo, T. H., Dysvik, B., & Jonassen, I. (2004). LSimpute: accurate estimation of missing values in microarray data with least squares methods. Nucleic acids research, 32(3), e34

See Also

Other LSimpute functions: impute_LS_adaptive(), impute_LS_combined(), impute_LS_gene()

Examples

```
set.seed(123)
ds_mis <- delete_MCAR(mvtnorm::rmvnorm(100, rep(0, 10)), 0.1)
ds_imp <- impute_LS_array(ds_mis)
```

impute_LS_combined *LSimpute_combined*

Description

Perform LSimpute_combined as described by Bo et al. (2004)

Usage

```
impute_LS_combined(
  ds,
  k = 10,
  eps = 1e-06,
  min_common_obs = 5,
  p_mis_sim = 0.05,
  verbose_gene = FALSE,
  verbose_array = FALSE,
  verbose_gene_p = FALSE,
  verbose_array_p = FALSE
)
```

Arguments

ds	A data frame or matrix with missing values.
k	Directly passed to impute_LS_gene().
eps	Directly passed to impute_LS_gene().
min_common_obs	Directly passed to impute_LS_gene().
p_mis_sim	Percentage of observed values that are set NA to estimate the mixing coefficient p. The default value (0.05) corresponds to the choice of Bo et al. (2004).
verbose_gene	Should impute_LS_gene() be verbose?
verbose_array	Should impute_LS_array() be verbose?
verbose_gene_p	Should impute_LS_gene() be verbose while estimating p?
verbose_array_p	Should impute_LS_array() be verbose while estimating p?

Details

This function performs LSimpute_combined as described by Bo et al. (2004).The function assumes that the genes are the rows of ds.

LSimpute_combined combines imputation values from impute_LS_gene() and impute_LS_array() using a global approach for the mixing coefficient p. The amount of feedback given from these underlying functions is controlled via verbose_gene, verbose_array, verbose_gene_p, verbose_array_p. The last two control the amount of feedback while estimating p and the first two the amount of feedback during the estimation of the values that are mixed with p. Internally, the imputed

dataset from `impute_LS_gene()` is passed on to `impute_LS_array()`. Therefore, all messages from `impute_LS_gene()` are truly from `impute_LS_gene()` and not a part of `impute_LS_array()`, which never calls `impute_LS_gene()` in this case. Furthermore, all messages from `impute_expected_values()` belong to `impute_LS_array()`.

Value

An object of the same class as `ds` with imputed missing values.

References

Bo, T. H., Dysvik, B., & Jonassen, I. (2004). LSimpute: accurate estimation of missing values in microarray data with least squares methods. Nucleic acids research, 32(3), e34

See Also

Other LSimpute functions: `impute_LS_adaptive()`, `impute_LS_array()`, `impute_LS_gene()`

Examples

```
set.seed(123)
ds_mis <- delete_MCAR(mvtnorm::rmvnorm(100, rep(0, 10)), 0.1)
ds_imp <- impute_LS_combined(ds_mis)
```

`impute_LS_gene` *LSimpute_gene*

Description

Perform LSimpute_gene as described by Bo et al. (2004)

Usage

```
impute_LS_gene(
  ds,
  k = 10,
  eps = 1e-06,
  min_common_obs = 5,
  return_r_max = FALSE,
  verbose = FALSE
)
```

Arguments

ds	A data frame or matrix with missing values.
k	Number of most correlated genes used for the imputation of a gene.
eps	Used in the calculation of the weights (Bo et al. (2004) used eps = 1e-6).
min_common_obs	A row can only take part in the imputation of another row, if both rows share at least min_common_obs columns with no missing values.
return_r_max	Logical; normally, this should be FALSE. TRUE is used inside of impute_LS_adaptive() to speed up some computations.
verbose	Should messages be given for special cases (see details)?

Details

This function performs LSimpute_gene as described by Bo et al. (2004).The function assumes that the genes are the rows of ds.

Bo et al. (2004) seem to have chosen min_common_obs = 5. However, they did not document this behavior. This value emerged from inspecting imputation results from the original jar-file, which is provided by Bo et al. (2004).

If there are less than min_common_obs observed values in a row and at least one observed value, the mean of the observed row values is imputed. If no value is observed in a row, the observed column means are imputed for the missing row values. This is the only known difference between this function and the original one from Bo et al. (2004). The original function would not impute such a row and return a dataset with missing values in this row. There is one more case that needs a special treatment: If no suitable row can be found to impute a row, the mean of the observed values is imputed, too. If verbose = TRUE, a message will be given for the encountered instances of the described special cases. If verbose = FALSE, the function will deal with theses cases silently.

Value

An object of the same class as ds with imputed missing values.

If return_r_max = TRUE, a list with the imputed dataset and r_max.

References

Bo, T. H., Dysvik, B., & Jonassen, I. (2004). LSimpute: accurate estimation of missing values in microarray data with least squares methods. Nucleic acids research, 32(3), e34

See Also

Other LSimpute functions: impute_LS_adaptive(), impute_LS_array(), impute_LS_combined()

Examples

```
set.seed(123)
ds_mis <- delete_MCAR(mvtnorm::rmvnorm(100, rep(0, 10)), 0.1)
ds_imp <- impute_LS_gene(ds_mis)
```

`impute_mean` *Mean imputation*

Description

Impute an observed mean for the missing values

Usage

```
impute_mean(ds, type = "columnwise", convert_tibble = TRUE)
```

Arguments

`ds`	A data frame or matrix with missing values.
`type`	A string specifying the values used for imputation; one of: "columnwise", "rowwise", "total", "Two-Way" or "Winer" (see details).
`convert_tibble`	If `ds` is a tibble, should it be converted (see section A note for tibble users).

Details

For every missing value the mean of some observed values is imputed. The observed values to be used are specified via `type`. For example, `type = "columnwise"` (the default) imputes the mean of the observed values in a column for all missing values in the column. This is normally meant, if someone speaks of "imputing the mean" or "mean imputation".

Other options for `type` are: "rowwise", "total", "Winer" and "Two-way". The option "rowwise" imputes all missing values in a row with the mean of the observed values in the same row. "total" will impute every missing value with the mean of all observed values in `ds`. "Winer" imputes the mean of the rowwise and columnwise mean. Beland et al. (2016) called this method "Winer" and they attributed the method to Winer (1971). "Two-way" imputes the sum of rowwise and columnwise mean minus the total mean. This method was suggested by D.B Rubin to Bernaards & Sijtsma, K. (2000).

Value

An object of the same class as `ds` with imputed missing values.

A note for tibble users

If you use tibbles and `convert_tibble` is `TRUE` the tibble is first converted to a data frame, then imputed and converted back. If `convert_tibble` is `FALSE` no conversion is done. However, depending on the tibble and the package version of tibble you use, imputation may not be possible and some errors will be thrown.

References

Beland, S., Pichette, F., & Jolani, S. (2016). Impact on Cronbach's α of simple treatment methods for missing data. *The Quantitative Methods for Psychology*, 12(1), 57-73.

Bernaards, C. A., & Sijtsma, K. (2000). Influence of imputation and EM methods on factor analysis when item nonresponse in questionnaire data is nonignorable. *Multivariate Behavioral Research*, 35(3), 321-364.

Winer, B. J. (1971). *Statistical principles in experimental design (2ed ed.)* New York: McGraw-Hill

See Also

`apply_imputation` the workhorse for this function.

Other location parameter imputation functions: `impute_median()`, `impute_mode()`

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
ds_mis <- delete_MCAR(ds, 0.2)
ds_imp <- impute_mean(ds_mis)
# completely observed columns can be of any type:
ds_mis_char <- cbind(ds_mis, letters[1:20])
ds_imp_char <- impute_mean(ds_mis_char)
```

---

impute_median *Median imputation*

Description

Impute an observed median value for every missing value

Usage

```
impute_median(
  ds,
  type = "columnwise",
  ordered_low = FALSE,
  convert_tibble = TRUE
)
```

Arguments

ds	A data frame or matrix with missing values.
type	A string specifying the values used for imputation; one of: "columnwise", "row-wise", "total", "Two-Way" or "Winer" (see details).
ordered_low	Logical; used for the calculation of the median from ordered factors (for details see: median.factor).
convert_tibble	If ds is a tibble, should it be converted (see section A note for tibble users).

Details

This function behaves exactly like impute_mean. The only difference is that it imputes a median instead of a mean. All types from impute_mean are also implemented for impute_median. They are documented in impute_mean and apply_imputation. The function median is used for the calculation of the median values for imputation.

Value

An object of the same class as ds with imputed missing values.

A note for tibble users

If you use tibbles and convert_tibble is TRUE the tibble is first converted to a data frame, then imputed and converted back. If convert_tibble is FALSE no conversion is done. However, depending on the tibble and the package version of tibble you use, imputation may not be possible and some errors will be thrown.

References

Beland, S., Pichette, F., & Jolani, S. (2016). Impact on Cronbach's α of simple treatment methods for missing data. *The Quantitative Methods for Psychology*, 12(1), 57-73.

See Also

apply_imputation the workhorse for this function.

median, median.factor

Other location parameter imputation functions: impute_mean(), impute_mode()

Examples

```
ds <- data.frame(X = 1:20, Y = ordered(LETTERS[1:20]))
ds_mis <- delete_MCAR(ds, 0.2)
ds_imp <- impute_median(ds_mis)
# completely observed columns can be of any type:
ds_mis_char <- cbind(ds_mis, letters[1:20])
ds_imp_char <- impute_median(ds_mis_char)
```

---

impute_mode *Mode imputation*

Description

Impute an observed mode value for every missing value

Usage

```
impute_mode(ds, type = "columnwise", convert_tibble = TRUE)
```

Arguments

ds	A data frame or matrix with missing values.
type	A string specifying the values used for imputation; one of: "columnwise", "row-wise", "total", "Two-Way" or "Winer" (see details).
convert_tibble	If ds is a tibble, should it be converted (see section A note for tibble users).

Details

This function behaves exactly like impute_mean. The only difference is that it imputes a mode instead of a mean. All types from impute_mean are also implemented for impute_mode. They are documented in impute_mean and apply_imputation.

A mode value of a vector x is a most frequent value of x. If this value is not unique, the first occurring mode value in x will be used as imputation value.

Value

An object of the same class as ds with imputed missing values.

References

Beland, S., Pichette, F., & Jolani, S. (2016). Impact on Cronbach's α of simple treatment methods for missing data. *The Quantitative Methods for Psychology*, 12(1), 57-73.

See Also

apply_imputation the workhorse for this function.

Other location parameter imputation functions: impute_mean(), impute_median()

Examples

```
ds <- data.frame(X = c(1:12, rep(8, 8)), Y = 101:120)
ds_mis <- delete_MCAR(ds, 0.2)
ds_imp <- impute_mode(ds_mis)
```

---

impute_sRHD *Simple random hot deck imputation*

Description

Impute missing values in a data frame or a matrix using a simple random hot deck

Usage

```
impute_sRHD(ds, type = "cols_seq", donor_limit = Inf)
```

Arguments

ds	A data frame or matrix with missing values.
type	The type of hot deck; the default ("cols_seq") is a random hot deck that imputes each column separately. Other choices are "sim_comp" and "sim_part". Both impute all missing values in an object (row) simultaneously using a single donor object. The difference between the two types is the choice of objects that can act as donors. "sim_comp:" only completely observed objects can be donors. "sim_part": all objects that have no missing values in the missing parts of a recipient can be donors.
donor_limit	Numeric of length one or "min"; how many times an object can be a donor. Default is Inf (no restriction).

Details

There are three types of simple random hot decks implemented. They can be selected via type:

- "cols_seq" (the default): Each variable (column) is handled separately. If an object (row) has a missing value in a variable (column), then one of the observed values in the same variable is chosen randomly and the missing value is replaced with this chosen value. This is done for all missing values.
- "sim_comp": All missing variables (columns) of an object are imputed together ("simultaneous"). For every object with missing values (such an object is called a recipient in hot deck terms), one complete object is chosen randomly and all missing values of the recipient are imputed with the values from the complete object. A complete object used for imputation is called a donor.
- "sim_part": All missing variables (columns) of an object are imputed together ("simultaneous"). For every object with missing values (recipient) one donor is chosen. The donor must have observed values in all the variables that are missing in the recipient. The donor is allowed to have unobserved values in the non-missing parts of the recipient. So, in contrast to "sim_comp", the donor can be partly incomplete.

The parameter donor_limit controls how often an object can be a donor. This parameter is only implemented for types "cols_seq" and "sim_comp". If type = "sim_part" and donor_limit is not Inf, then an error will be thrown. For "sim_comp" the default value (Inf) allows every object to be a donor for an infinite number of times (there is no restriction on the times an object can be a donor). If a numeric value less than Inf is chosen, then every object can be a donor at most donor_limit times. For example donor_limit = 1 ensures that every object donates at most one time. If there are only few complete objects and donor_limit is set too low, then an imputation might not be possible with the chosen donor_limit. In this case, the donor_limit will be adjusted (see examples). Setting donor_limit = "min" chooses automatically the minimum value for donor_limit that allows imputation of all missing values. For type = "cols_seq" the donor limit is applied for every column separately.

Value

An object of the same class as ds with imputed missing values.

References

Andridge, R. R., & Little, R. J. (2010). A review of hot deck imputation for survey non-response. *International statistical review*, 78(1), 40-64.

Examples

```
ds <- data.frame(X = 1:20, Y = 101:120)
ds_mis <- delete_MCAR(ds, 0.2)
ds_imp <- impute_sRHD(ds_mis)

# Warning: donor limit to low
ds_mis_one_donor <- ds
ds_mis_one_donor[1:19, "X"] <- NA
impute_sRHD(ds_mis_one_donor, donor_limit = 3)
```

`median.factor` *Median for ordered factors*

Description

Compute the median of an ordered factor

Usage

```
## S3 method for class 'factor'
median(x, na.rm = FALSE, ordered_low = FALSE, ...)
```

Arguments

`x`	An ordered factor (for unordered factors an error will be thrown).
`na.rm`	Logical; should `NA` be removed before computation?
`ordered_low`	Logical; only used if the length of `x` is even and the two middle values are unequal (see details).
`...`	Not used in this function.

Details

Currently, the median for an ordered factor is not implemented in base R. This function is a remedy for this. It allows the computation of "a median" for ordered factors (see below) and overwrites the error message for unordered factors from `median.default` (hence, the function name is median.factor and not median.ordered).

If the length of `x` is odd, then the median will be the middle value of the sorted list of elements from `x`. If the length of `x` is even and the two middle values of the sorted list of elements from `x` are equal, then the median is one of these (equal) middle values. The only problematic case is an even length `x` with unequal middle values. In this case, the median of a numeric vector is normally defined as the

mean of the two middle values. However, for ordered factors the mean is not defined. The argument `ordered_low` cures this problem. If `ordered_low = FALSE` (the default), then the larger of the two middle values is returned (this value is called 'hi-median' in `mad`). If `ordered_low = TRUE`, then the smaller of the two middle values is returned (this value is called 'low-median' in `mad`).

Value

a length-one factor

Examples

```
ord_factor_odd <- ordered(letters[1:5])
median(ord_factor_odd) # calls median.factor, if package is loaded

# If only base R is loaded, median.default will be called
# and will throw an error:
tryCatch(median.default(ord_factor_odd), error = function(e) e)

ord_factor_even <- ordered(letters[1:4])
median(ord_factor_even, ordered_low = FALSE)
median(ord_factor_even, ordered_low = TRUE)
```

Index