Package: VIM (via r-universe)

October 20, 2026

Version 7.3.1

Title Visualization and Imputation of Missing Values

Depends R (>= 4.1.0),colorspace,grid

Imports grDevices, robustbase, stats, sp, vcd, nnet, e1071, methods, Rcpp, utils, graphics, laeken, ranger, MASS, Matrix, xgboost, data.table(>= 1.9.4)

Suggests mlr3, mlr3pipelines, mlr3learners, mlr3tuning (>= 1.0.0), paradox (>= 1.0.0), R6, future, dplyr, tinytest, knitr, mgcv, rmarkdown, reactable, covr, withr, pdist, enetLTS, robmixglm, stringr, glmnet, validate, ECOSolveR, cellWise, crmReg, mice, car, lgr, rpart, missRanger, tidyr

Description Provides methods for imputation and visualization of missing values. It includes graphical tools to explore the amount, structure and patterns of missing and/or imputed values, supporting exploratory data analysis and helping to investigate potential missingness mechanisms (details in Alfons, Templ and Filzmoser, <doi:10.1007/s11634-011-0102-y>). The quality of imputations can be assessed visually using a wide range of univariate, bivariate and multivariate plots. The package further provides several imputation methods, including efficient implementations of k-nearest neighbour and hot-deck imputation (Kowarik and Templ 2013, <doi:10.18637/jss.v074.i07>), iterative robust model-based multiple imputation (Templ 2011, <doi:10.1016/j.csda.2011.04.012>; Templ 2023, <doi:10.3390/math11122729>), and machine learning–based approaches such as robust GAM-based multiple imputation (Templ 2024, <doi:10.1007/s11222-024-10429-1>) as well as random forest and gradient boosting (XGBoost) imputation (Niederhametner et al., <doi:10.1177/18747655251339401>). General background and practical guidance on imputation are provided in the Springer book by Templ (2023) <doi:10.1007/978-3-031-30073-8>.

LazyData TRUE

ByteCompile TRUE

License GPL (>= 2)

URL https://github.com/statistikat/VIM

BugReports https://github.com/statistikat/VIM/issues

LinkingTo Rcpp

Encoding UTF-8

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

NeedsCompilation yes

Author Matthias Templ [aut, cre], Alexander Kowarik [aut] (ORCID: <https://orcid.org/0000-0001-8598-4130>), Andreas Alfons [aut], Johannes Gussenbauer [aut], Nina Niederhametner [aut], Eileen Vattheuer [aut], Gregor de Cillia [aut], Bernd Prantner [ctb], Wolfgang Rannetbauer [aut]

Maintainer Matthias Templ <matthias.templ@gmail.com>

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

Date/Publication 2026-09-11 09:50:59 UTC

RemoteUrl https://github.com/cran/VIM

RemoteRef HEAD

RemoteSha a0acbef38ffe81043f48261d58879de6dee3d381

Contents

aggr 4
alphablend 8
Animals_na 9
barMiss 10
bcancer 12
bgmap 13
brittleness 14
cellWeightsMCD 15
chorizonDL 16
colic 19
collisions 21
colormapMiss 22
colSequence 25
countInf 26
diabetes 27
evaluation 29
food 30
gapMiss 31

gowerD . . . 32
growdotMiss . . . 33
histMiss . . . 36
hotdeck . . . 39
impPCA . . . 41
imputeCellEM . . . 43
imputeCellIRMI . . . 46
imputeCellM . . . 49
imputeCellMCD . . . 51
imputeCellMM . . . 54
imputeCellReg . . . 55
imputeCellwise . . . 56
imputeRobust . . . 57
imputeRobustChain . . . 58
initialise . . . 60
irmi . . . 61
kNN . . . 64
kola.background . . . 67
lse_synthetic . . . 67
lse_synthetic_rules . . . 68
makeMissing . . . 69
mapMiss . . . 70
marginmatrix . . . 72
marginplot . . . 74
matchImpute . . . 77
matrixplot . . . 78
maxCat . . . 80
medianSamp . . . 81
mosaicMiss . . . 81
overimpute . . . 83
pairsVIM . . . 85
parcoordMiss . . . 87
pbox . . . 90
plot.vimmi . . . 92
prepare . . . 93
pulplignin . . . 95
rangerImpute . . . 96
register_vimpute_method . . . 97
regressionImp . . . 99
rugNA . . . 100
sampleCat . . . 101
SBS5242 . . . 102
scattJitt . . . 103
scattmatrixMiss . . . 105
scattMiss . . . 107
sleep . . . 110
spineMiss . . . 111
tableMiss . . . 114

tao 114
testdata 115
toydataMiss 116
unregister_vimpute_method 117
vim_as_mids 117
vim_complete 118
vimmi 119
vimpute 121
vimpute_methods 127
vimpute_search_space 127
vimpute_spec 128
vimpute_tune_control 130
wine 131
with.vimmi 132
xgboostImpute 133

Index **135**

`aggr` *Aggregations for missing/imputed values*

Description

Calculate or plot the amount of missing/imputed values in each variable and the amount of missing/imputed values in certain combinations of variables.

Print method for objects of class `"aggr"`.

Summary method for objects of class `"aggr"`.

Print method for objects of class `"summary.aggr"`.

Usage

```
aggr(x, delimiter = NULL, plot = TRUE, ...)

## S3 method for class 'aggr'
plot(
  x,
  col = c("skyblue", "red", "orange"),
  bars = TRUE,
  numbers = FALSE,
  prop = TRUE,
  combined = FALSE,
  varheight = FALSE,
  only.miss = FALSE,
  border = par("fg"),
  sortVars = FALSE,
  sortCombs = TRUE,
  ylabs = NULL,
```

```
  axes = TRUE,
  labels = axes,
  cex.lab = 1.2,
  cex.axis = par("cex"),
  cex.numbers = par("cex"),
  gap = 4,
  ...
)

## S3 method for class 'aggr'
print(x, ..., digits = NULL)

## S3 method for class 'aggr'
summary(object, ...)

## S3 method for class 'summary.aggr'
print(x, ...)
```

Arguments

x	an object of class "summary.aggr".
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
plot	a logical indicating whether the results should be plotted (the default is TRUE).
...	Further arguments, currently ignored.
col	a vector of length three giving the colors to be used for observed, missing and imputed data. If only one color is supplied, it is used for missing and imputed data and observed data is transparent. If only two colors are supplied, the first one is used for observed data and the second color is used for missing and imputed data.
bars	a logical indicating whether a small barplot for the frequencies of the different combinations should be drawn.
numbers	a logical indicating whether the proportion or frequencies of the different combinations should be represented by numbers.
prop	a logical indicating whether the proportion of missing/imputed values and combinations should be used rather than the total amount.
combined	a logical indicating whether the two plots should be combined. If FALSE, a separate barplot on the left hand side shows the amount of missing/imputed values in each variable. If TRUE, a small version of this barplot is drawn on top of the plot for the combinations of missing/imputed and non-missing values. See "Details" for more information.
varheight	a logical indicating whether the cell heights are given by the frequencies of occurrence of the corresponding combinations.

only.miss	a logical indicating whether the small barplot for the frequencies of the combinations should only be drawn for combinations including missing/imputed values (if bars is TRUE). This is useful if most observations are complete, in which case the corresponding bar would dominate the barplot such that the remaining bars are too compressed. The proportion or frequency of complete observations (as determined by prop) is then represented by a number instead of a bar.
border	the color to be used for the border of the bars and rectangles. Use border=NA to omit borders.
sortVars	a logical indicating whether the variables should be sorted by the number of missing/imputed values.
sortCombs	a logical indicating whether the combinations should be sorted by the frequency of occurrence.
ylabs	if combined is TRUE, a character string giving the y-axis label of the combined plot, otherwise a character vector of length two giving the y-axis labels for the two plots.
axes	a logical indicating whether axes should be drawn.
labels	either a logical indicating whether labels should be plotted on the x-axis, or a character vector giving the labels.
cex.lab	the character expansion factor to be used for the axis labels.
cex.axis	the character expansion factor to be used for the axis annotation.
cex.numbers	the character expansion factor to be used for the proportion or frequencies of the different combinations
gap	if combined is FALSE, a numeric value giving the distance between the two plots in margin lines.
digits	the minimum number of significant digits to be used (see print.default()).
object	an object of class "aggr".

Details

Often it is of interest how many missing/imputed values are contained in each variable. Even more interesting, there may be certain combinations of variables with a high number of missing/imputed values.

If combined is FALSE, two separate plots are drawn for the missing/imputed values in each variable and the combinations of missing/imputed and non-missing values. The barplot on the left hand side shows the amount of missing/imputed values in each variable. In the *aggregation plot* on the right hand side, all existing combinations of missing/imputed and non-missing values in the observations are visualized. Available, missing and imputed data are color coded as given by col. Additionally, there are two possibilities to represent the frequencies of occurrence of the different combinations. The first option is to visualize the proportions or frequencies by a small bar plot and/or numbers. The second option is to let the cell heights be given by the frequencies of the corresponding combinations. Furthermore, variables may be sorted by the number of missing/imputed values and combinations by the frequency of occurrence to give more power to finding the structure of missing/imputed values.

If combined is TRUE, a small version of the barplot showing the amount of missing/imputed values in each variable is drawn on top of the aggregation plot.

The graphical parameter oma will be set unless supplied as an argument.

Value

for aggr, a list of class "aggr" containing the following components:

- x the data used.
- combinations a character vector representing the combinations of variables.
- count the frequencies of these combinations.
- percent the percentage of these combinations.
- missings a data.frame containing the amount of missing/imputed values in each variable.
- tabcomb the indicator matrix for the combinations of variables.

a list of class "summary.aggr" containing the following components:

- missings a data.frame containing the amount of missing or imputed values in each variable.
- combinations a data.frame containing a character vector representing the combinations of variables along with their frequencies and percentages.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in VIM. For back compatibility, the arguments labs and names.arg can still be supplied to ...{} and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use ylabs and labels instead.

Author(s)

Andreas Alfons, Matthias Templ, modifications for displaying imputed values by Bernd Prantner

Matthias Templ, modifications by Andreas Alfons and Bernd Prantner

Matthias Templ, modifications by Andreas Alfons

Andreas Alfons, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

print.aggr(), summary.aggr()

aggr()

print.summary.aggr(), aggr()

summary.aggr(), aggr()

Other plotting functions: barMiss(), histMiss(), marginmatrix(), marginplot(), matrixplot(), mosaicMiss(), pairsVIM(), parcoordMiss(), pbox(), scattJitt(), scattMiss(), scattmatrixMiss(), spineMiss()

Examples

```
data(sleep, package="VIM")
## for missing values
a <- aggr(sleep)
a
summary(a)

## for imputed values
sleep_IMPUTED <- kNN(sleep)
a <- aggr(sleep_IMPUTED, delimiter="_imp")
a
summary(a)


data(sleep, package = "VIM")
a <- aggr(sleep, plot=FALSE)
a


data(sleep, package = "VIM")
summary(aggr(sleep, plot=FALSE))


data(sleep, package = "VIM")
s <- summary(aggr(sleep, plot=FALSE))
s
```

`alphablend` *Alphablending for colors*

Description

Convert colors to semitransparent colors.

Usage

```
alphablend(col, alpha = NULL, bg = NULL)
```

Arguments

`col`	a vector specifying colors.
`alpha`	a numeric vector containing the alpha values (between 0 and 1).
`bg`	the background color to be used for alphablending. This can be used as a workaround for graphics devices that do not support semitransparent colors.

Value

a vector containing the semitransparent colors.

Author(s)

Andreas Alfons

Examples

```
alphablend("red", 0.6)
```

`Animals_na` *Animals_na*

Description

Average log brain and log body weights for 28 Species

Format

A data frame with 28 observations on the following 2 variables.

lbody log body weight

lbrain log brain weight

Details

The original data can be found in package MASS. 10 values on brain weight are set to be missing.

Source

P. J. Rousseeuw and A. M. Leroy (1987) Robust Regression and Outlier Detection. Wiley, p. 57.

References

Venables, W. N. and Ripley, B. D. (1999) Modern Applied Statistics with S-PLUS. Third Edition. Springer.

Templ, M. (2022) Visualization and Imputation of Missing Values. Springer Publishing. Upcoming book.

Examples

```
data(Animals_na)
aggr(Animals_na)
```

barMiss *Barplot with information about missing/imputed values*

Description

Barplot with highlighting of missing/imputed values in other variables by splitting each bar into two parts. Additionally, information about missing/imputed values in the variable of interest is shown on the right hand side.

Usage

```
barMiss(
  x,
  delimiter = NULL,
  pos = 1,
  selection = c("any", "all"),
  col = c("skyblue", "red", "skyblue4", "red4", "orange", "orange4"),
  border = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  axes = TRUE,
  labels = axes,
  only.miss = TRUE,
  miss.labels = axes,
  interactive = TRUE,
  ...
)
```

Arguments

x	a vector, matrix or data.frame.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
pos	a numeric value giving the index of the variable of interest. Additional variables in x are used for highlighting.
selection	the selection method for highlighting missing/imputed values in multiple additional variables. Possible values are "any" (highlighting of missing/imputed values in *any* of the additional variables) and "all" (highlighting of missing/imputed values in *all* of the additional variables).

col	a vector of length six giving the colors to be used. If only one color is supplied, the bars are transparent and the supplied color is used for highlighting missing/imputed values. Else if two colors are supplied, they are recycled.
border	the color to be used for the border of the bars. Use border=NA to omit borders.
main, sub	main and sub title.
xlab, ylab	axis labels.
axes	a logical indicating whether axes should be drawn on the plot.
labels	either a logical indicating whether labels should be plotted below each bar, or a character vector giving the labels.
only.miss	logical; if TRUE, the missing/imputed values in the variable of interest are visualized by a single bar. Otherwise, a small barplot is drawn on the right hand side (see 'Details').
miss.labels	either a logical indicating whether label(s) should be plotted below the bar(s) on the right hand side, or a character string or vector giving the label(s) (see 'Details').
interactive	a logical indicating whether variables can be switched interactively (see 'Details').
...	further graphical parameters to be passed to graphics::title() and graphics::axis().

Details

If more than one variable is supplied, the bars for the variable of interest are split according to missingness/number of imputed missings in the additional variables.

If only.miss=TRUE, the missing/imputed values in the variable of interest are visualized by one bar on the right hand side. If additional variables are supplied, this bar is again split into two parts according to missingness/number of imputed missings in the additional variables.

Otherwise, a small barplot consisting of two bars is drawn on the right hand side. The first bar corresponds to observed values in the variable of interest and the second bar to missing/imputed values. Since these two bars are not on the same scale as the main barplot, a second y-axis is plotted on the right (if axes=TRUE). Each of the two bars are again split into two parts according to missingness/number of imputed missings in the additional variables. Note that this display does not make sense if only one variable is supplied, therefore only.miss is ignored in that case.

If interactive=TRUE, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session. When switching to a continuous variable, a histogram is plotted rather than a barplot.

Value

a numeric vector giving the coordinates of the midpoints of the bars.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in VIM. For back compatibility, the

arguments `axisnames`, `names.arg` and `names.miss` can still be supplied to `...{}` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `labels` and `miss.labels` instead.

Author(s)

Andreas Alfons, modifications to show imputed values by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

`spineMiss()`, `histMiss()`

Other plotting functions: `aggr()`, `histMiss()`, `marginmatrix()`, `marginplot()`, `matrixplot()`, `mosaicMiss()`, `pairsVIM()`, `parcoordMiss()`, `pbox()`, `scattJitt()`, `scattMiss()`, `scattmatrixMiss()`, `spineMiss()`

Examples

```
data(sleep, package = "VIM")
## for missing values
x <- sleep[, c("Exp", "Sleep")]
barMiss(x)
barMiss(x, only.miss = FALSE)

## for imputed values
x_IMPUTED  <- kNN(sleep[, c("Exp", "Sleep")])
barMiss(x_IMPUTED, delimiter = "_imp")
barMiss(x_IMPUTED, delimiter = "_imp", only.miss = FALSE)
```

---

bcancer *Breast cancer Wisconsin data set*

Description

Dataset containing the original Wisconsin breast cancer data.

Format

A data frame with 699 observations on the following 11 variables.

ID Sample ID

clump_thickness as integer from 1 - 10

uniformity_cellsize as integer from 1 - 10

uniformity_cellshape as integer from 1 - 10

adhesion as integer from 1 - 10

epithelial_cellsize as integer from 1 - 10

bare_nuclei as integer from 1 - 10, includes 16 missings

chromatin as integer from 1 - 10

normal_nucleoli as integer from 1 - 10

mitoses as integer from 1 - 10

class benign or malignant

References

The data downloaded and conditioned for R from the UCI machine learning repository, see https://archive.ics.uci.edu/ml/datas
This breast cancer databases was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. If you publish results when using this database, then please include this information in your acknowledgements. Also, please cite one or more of: O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18. William H. Wolberg and O.L. Mangasarian: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193-9196. O. L. Mangasarian, R. Setiono, and W.H. Wolberg: "Pattern recognition via linear programming: Theory and application to medical diagnosis", in: "Large-scale numerical optimization", Thomas F. Coleman and Yuying Li, editors, SIAM Publications, Philadelphia 1990, pp 22-30. K. P. Bennett & O. L. Mangasarian: "Robust linear programming discrimination of two linearly inseparable sets", Optimization Methods and Software 1, 1992, 23-34 (Gordon & Breach Science Publishers).

Examples

```
data(bcancer)
aggr(bcancer)
```

---

bgmap *Backgound map*

Description

Plot a background map.

Usage

```
bgmap(map, add = FALSE, ...)
```

Arguments

map	either a matrix or data.frame with two columns, a list with components x and y, or an object of any class that can be used for maps and provides its own plot method (e.g., "SpatialPolygons" from package sp). A list of the previously mentioned types can also be provided.
add	a logical indicating whether map should be added to an already existing plot (the default is FALSE).
...	further arguments and graphical parameters to be passed to plot and/or graphics::lines().

Author(s)

Andreas Alfons

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Vol. 6, pp. 29-47. DOI: 10.1007/s11634-011-0102-y.

See Also

growdotMiss(), mapMiss()

Examples

```
data(kola.background, package = "VIM")
bgmap(kola.background)
```

---

brittleness *Brittleness index data set*

Description

A plastic product is produced in three parallel reactors (TK104, TK105, or TK107). For each row in the dataset, we have the same batch of raw material that was split, and fed to the 3 reactors. These values are the brittleness index for the product produced in the reactor. A simulated data set.

Format

A data frame with 23 observations on the following 3 variables.

TK104 Brittleness for batches of raw material in reactor 104

TK105 Brittleness for batches of raw material in reactor 105

TK107 Brittleness for batches of raw material in reactor 107

Source

https://openmv.net/info/brittleness-index

Examples

```
data(brittleness)
aggr(brittleness)
```

`cellWeightsMCD` *Compute per-cell weights using MCD-based conditional residuals*

Description

For each continuous cell (i,j), computes the conditional expectation $E(x_{ij}|x_{i,-j})$ under a robust Gaussian model (estimated via MCD), standardizes the residual, and applies a weight function. This captures multivariate outlier structure that univariate standardization misses.

Usage

```
cellWeightsMCD(X, method = "tukey", alpha = NULL)
```

Arguments

`X`	a data frame or matrix of dimension n x p (continuous columns only)
`method`	weight function: `"tukey"` or `"huber"`, Default: `"tukey"`
`alpha`	tuning constant. If `NULL`, defaults to 4.685 for Tukey and 1.345 for Huber.

Value

an n x p numeric matrix of weights in $[0, 1]$

Author(s)

Matthias Templ

chorizonDL *C-horizon of the Kola data with missing values*

Description

This data set is the same as in package `mvoutlier`, except that values below the detection limit are coded as `NA`.

Format

A data frame with 606 observations on the following 110 variables.

***ID** a numeric vector

XCOO a numeric vector

YCOO a numeric vector

Ag a numeric vector

Ag_INAA a numeric vector

Al a numeric vector

Al2O3 a numeric vector

As a numeric vector

As_INAA a numeric vector

Au_INAA a numeric vector

B a numeric vector

Ba a numeric vector

Ba_INAA a numeric vector

Be a numeric vector

Bi a numeric vector

Br_IC a numeric vector

Br_INAA a numeric vector

Ca a numeric vector

Ca_INAA a numeric vector

CaO a numeric vector

Cd a numeric vector

Ce_INAA a numeric vector

Cl_IC a numeric vector

Co a numeric vector

Co_INAA a numeric vector

EC a numeric vector

Cr a numeric vector

Cr_INAA a numeric vector

Cs_INAA a numeric vector

Cu a numeric vector

Eu_INAA a numeric vector

F_IC a numeric vector

Fe a numeric vector

Fe_INAA a numeric vector

Fe2O3 a numeric vector

Hf_INAA a numeric vector

Hg a numeric vector

Hg_INAA a numeric vector

Ir_INAA a numeric vector

K a numeric vector

K2O a numeric vector

La a numeric vector

La_INAA a numeric vector

Li a numeric vector

LOI a numeric vector

Lu_INAA a numeric vector

wt_INAA a numeric vector

Mg a numeric vector

MgO a numeric vector

Mn a numeric vector

MnO a numeric vector

Mo a numeric vector

Mo_INAA a numeric vector

Na a numeric vector

Na_INAA a numeric vector

Na2O a numeric vector

Nd_INAA a numeric vector

Ni a numeric vector

Ni_INAA a numeric vector

NO3_IC a numeric vector

P a numeric vector

P2O5 a numeric vector

Pb a numeric vector

pH a numeric vector

PO4_IC a numeric vector

Rb a numeric vector

S a numeric vector

Sb a numeric vector

Sb_INAA a numeric vector

Sc a numeric vector

Sc_INAA a numeric vector

Se a numeric vector

Se_INAA a numeric vector

Si a numeric vector

SiO2 a numeric vector

Sm_INAA a numeric vector

Sn_INAA a numeric vector

SO4_IC a numeric vector

Sr a numeric vector

Sr_INAA a numeric vector

SUM_XRF a numeric vector

Ta_INAA a numeric vector

Tb_INAA a numeric vector

Te a numeric vector

Th a numeric vector

Th_INAA a numeric vector

Ti a numeric vector

TiO2 a numeric vector

U_INAA a numeric vector

V a numeric vector

W_INAA a numeric vector

Y a numeric vector

Yb_INAA a numeric vector

Zn a numeric vector

Zn_INAA a numeric vector

ELEV a numeric vector

***COUN** a numeric vector

***ASP** a numeric vector

TOPC a numeric vector

LITO a numeric vector

Al_XRF a numeric vector

Ca_XRF a numeric vector

Fe_XRF a numeric vector

K_XRF a numeric vector

Mg_XRF a numeric vector

Mn_XRF a numeric vector

Na_XRF a numeric vector

P_XRF a numeric vector

Si_XRF a numeric vector

Ti_XRF a numeric vector

Note

For a more detailed description of this data set, see the help file `chorizon` in package `mvoutlier`.

Source

Kola Project (1993-1998)

References

Reimann, C., Filzmoser, P., Garrett, R.G. and Dutter, R. (2008) *Statistical Data Analysis Explained: Applied Environmental Statistics with R*. Wiley.

Examples

```
data(chorizonDL, package = "VIM")
summary(chorizonDL)
```

---

colic *Colic horse data set*

Description

This is a modified version of the original training data set taken from the UCI repository, see reference. The modifications are only related to having appropriate levels for factor variables. This data set is about horse diseases where the task is to determine, if the lesion of the horse was surgical or not.

Format

A training data frame with 300 observations on the following 31 variables.

surgery yes or no

age 1 equals an adult horse, 2 is a horse younger than 6 months

hospitalID ID

temp_rectal rectal temperature

pulse heart rate in beats per minute

respiratory_rate a normal rate is between 8 and 10

temp_extreme temperature of extremities

pulse_peripheral factor with four categories

capillayr_refill_time a clinical judgement. The longer the refill, the poorer the circulation. Possible values are 1 = < 3 seconds and 2 = >= 3 seconds

pain a subjective judgement of the horse's pain level

peristalsis an indication of the activity in the horse's gut. As the gut becomes more distended or the horse becomes more toxic, the activity decreases

abdominal_distension An animal with abdominal distension is likely to be painful and have reduced gut motility. A horse with severe abdominal distension is likely to require surgery just tio relieve the pressure

nasogastric_tube This refers to any gas coming out of the tube. A large gas cap in the stomach is likely to give the horse discomfort

nasogastric_reflux posible values are 1 = none, 2 = > 1 liter, 3 = < 1 liter. The greater amount of reflux, the more likelihood that there is some serious obstruction to the fluid passage from the rest of the intestine

nasogastric_reflux_PH scale is from 0 to 14 with 7 being neutral. Normal values are in the 3 to 4 range

rectal_examination Rectal examination. Absent feces probably indicates an obstruction

abdomen abdomen. possible values 1 = normal, 2 = other, 3 = firm feces in the large intestine, 4 = distended small intestine, 5 = distended large intestine

cell_volume packed cell volume. normal range is 30 to 50. The level rises as the circulation becomes compromised or as the animal becomes dehydrated.

protein total protein. Normal values lie in the 6-7.5 (gms/dL) range. The higher the value the greater the dehydration

abdominocentesis_appearance Abdominocentesis appearance. A needle is put in the horse's abdomen and fluid is obtained from the abdominal cavity

abdomcentesis_protein abdomcentesis total protein. The higher the level of protein the more likely it is to have a compromised gut. Values are in gms/dL

outcome What eventually happened to the horse?

surgical_lesion retrospectively, was the problem (lesion) surgical?

lesion_type1 type of lesion

lesion_type2 type of lesion

lesion_type3 type of lesion

cp_data

temp_extreme_ordered temperature of extremities (ordered)

mucous_membranes_col mucous membranes. A subjective measurement of colour

mucous_membranes_group different recodings of mucous membrances

Source

https://archive.ics.uci.edu/ml/datasets/Horse+Colic Creators: Mary McLeish & Matt Cecile, Department of Computer Science, University of Guelph, Guelph, Ontario, Canada N1G 2W1 Donor: Will Taylor

Examples

```
data(colic)
aggr(colic)
```

---

collisions *Subset of the collision data*

Description

Subset of the collision data from December 20. to December 31. 2018 from NYCD.

Details

Each record represents a collision in NYC by city, borough, precinct and cross street.

Source

https://data.cityofnewyork.us/Public-Safety/NYPD-Motor-Vehicle-Collisions/h9gi-nx95

Examples

```
data(collisions)
aggr(collisions)
```

colormapMiss *Colored map with information about missing/imputed values*

Description

Colored map in which the proportion or amount of missing/imputed values in each region is coded according to a continuous or discrete color scheme. The sequential color palette may thereby be computed in the *HCL* or the *RGB* color space.

Usage

```
colormapMiss(
  x,
  region,
  map,
  imp_index = NULL,
  prop = TRUE,
  polysRegion = 1:length(x),
  range = NULL,
  n = NULL,
  col = c("red", "orange"),
  gamma = 2.2,
  fixup = TRUE,
  coords = NULL,
  numbers = TRUE,
  digits = 2,
  cex.numbers = 0.8,
  col.numbers = par("fg"),
  legend = TRUE,
  interactive = TRUE,
  ...
)

colormapMissLegend(
  xleft,
  ybottom,
  xright,
  ytop,
  cmap,
  n = 1000,
  horizontal = TRUE,
  digits = 2,
  cex.numbers = 0.8,
  col.numbers = par("fg"),
  ...
)
```

Arguments

x	a numeric vector.
region	a vector or factor of the same length as x giving the regions.
map	an object of any class that contains polygons and provides its own plot method (e.g., "SpatialPolygons" from package sp).
imp_index	a logical-vector indicating which values of 'x' have been imputed. If given, it is used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
prop	a logical indicating whether the proportion of missing/imputed values should be used rather than the total amount.
polysRegion	a numeric vector specifying the region that each polygon belongs to.
range	a numeric vector of length two specifying the range (minimum and maximum) of the proportion or amount of missing/imputed values to be used for the color scheme.
n	for colormapMiss, the number of equally spaced cut-off points for a discretized color scheme. If this is not a positive integer, a continuous color scheme is used (the default). In the latter case, the number of rectangles to be drawn in the legend can be specified in colormapMissLegend. A reasonably large number makes it appear continuously.
col	the color range (start end end) to be used. RGB colors may be specified as character strings or as objects of class "colorspace::RGB()". HCL colors need to be specified as objects of class "colorspace::polarLUV()". If only one color is supplied, it is used as end color, while the start color is taken to be transparent for RGB or white for HCL.
gamma	numeric; the display *gamma* value (see colorspace::hex()).
fixup	a logical indicating whether the colors should be corrected to valid RGB values (see colorspace::hex()).
coords	a matrix or data.frame with two columns giving the coordinates for the labels.
numbers	a logical indicating whether the corresponding proportions or numbers of missing/imputed values should be used as labels for the regions.
digits	the number of digits to be used in the labels (in case of proportions).
cex.numbers	the character expansion factor to be used for the labels.
col.numbers	the color to be used for the labels.
legend	a logical indicating whether a legend should be plotted.
interactive	a logical indicating whether more detailed information about missing/imputed values should be displayed interactively (see 'Details').
...	further arguments to be passed to plot.
xleft	left *x* position of the legend.
ybottom	bottom *y* position of the legend.
xright	right *x* position of the legend.
ytop	top *y* position of the legend.

cmap	a list as returned by colormapMiss that contains the required information for the legend.
horizontal	a logical indicating whether the legend should be drawn horizontally or vertically.

Details

The proportion or amount of missing/imputed values in x of each region is coded according to a continuous or discrete color scheme in the color range defined by col. In addition, the proportions or numbers can be shown as labels in the regions.

If interactive is TRUE, clicking in a region displays more detailed information about missing/imputed values on the console. Clicking outside the borders quits the interactive session.

Value

colormapMiss returns a list with the following components:

- nmiss a numeric vector containing the number of missing/imputed values in each region.
- nobs a numeric vector containing the number of observations in each region.
- pmiss a numeric vector containing the proportion of missing values in each region.
- prop a logical indicating whether the proportion of missing/imputed values have been used rather than the total amount.
- range the range of the proportion or amount of missing/imputed values corresponding to the color range.
- n either a positive integer giving the number of equally spaced cut-off points for a discretized color scheme, or NULL for a continuous color scheme.
- start the start color of the color scheme.
- end the end color of the color scheme.
- space a character string giving the color space (either "rgb" for RGB colors or "hcl" for HCL colors).
- gamma numeric; the display *gamma* value (see colorspace::hex()).
- fixup a logical indicating whether the colors have been corrected to valid RGB values (see colorspace::hex()).

Note

Some of the argument names and positions have changed with versions 1.3 and 1.4 due to extended functionality and for more consistency with other plot functions in VIM. For back compatibility, the arguments cex.text and col.text can still be supplied to ...{} and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use cex.numbers and col.numbers instead.

Author(s)

Andreas Alfons, modifications to show imputed values by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

`colSequence()`, `growdotMiss()`, `mapMiss()`

`colSequence` *HCL and RGB color sequences*

Description

Compute color sequences by linear interpolation based on a continuous color scheme between certain start and end colors. Color sequences may thereby be computed in the *HCL* or *RGB* color space.

Usage

```
colSequence(p, start, end, space = c("hcl", "rgb"), ...)

colSequenceRGB(p, start, end, fixup = TRUE, ...)

colSequenceHCL(p, start, end, fixup = TRUE, ...)
```

Arguments

`p`	a numeric vector with values between 0 and 1 giving values to be used for interpolation between the start and end color (0 corresponds to the start color, 1 to the end color).
`start`, `end`	the start and end color, respectively. For HCL colors, each can be supplied as a vector of length three (hue, chroma, luminance) or an object of class "`colorspace::polarLUV()`". For RGB colors, each can be supplied as a character string, a vector of length three (red, green, blue) or an object of class "`colorspace::RGB()`".
`space`	character string; if `start` and `end` are both numeric, this determines whether they refer to HCL or RGB values. Possible values are `"hcl"` (for the HCL space) or `"rgb"` (for the RGB space).
`...`	for `colSequence`, additional arguments to be passed to `colSequenceHCL` or `colSequenceRGB`. For `colSequenceHCL` and `colSequenceRGB`, additional arguments to be passed to `colorspace::hex()`.
`fixup`	a logical indicating whether the colors should be corrected to valid RGB values (see `colorspace::hex()`).

Value

A character vector containing hexadecimal strings of the form `"#RRGGBB"`.

Author(s)

Andreas Alfons

References

Zeileis, A., Hornik, K., Murrell, P. (2009) Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, **53 (9)**, 1259–1270.

See Also

`colorspace::hex()`, `colorspace::sequential_hcl()`

Examples

```
p <- c(0, 0.3, 0.55, 0.8, 1)

## HCL colors
colSequence(p, c(0, 0, 100), c(0, 100, 50))
colSequence(p, polarLUV(L=90, C=30, H=90), c(0, 100, 50))

## RGB colors
colSequence(p, c(1, 1, 1), c(1, 0, 0), space="rgb")
colSequence(p, RGB(1, 1, 0), "red")
```

---

countInf *Count number of infinite or missing values*

Description

Count the number of infinite or missing values in a vector.

Usage

```
countInf(x)
```

Arguments

x	a vector.

Value

`countInf` returns the number of infinite values in `x`. `countNA` returns the number of missing values in `x`.

Author(s)

Andreas Alfons

Examples

```
data(sleep, package="VIM")
countInf(log(sleep$Dream))
countNA(sleep$Dream)
```

`diabetes` *Synthetic Pima Indians Diabetes Data*

Description

A synthetic replacement for the Pima Indians Diabetes data: 768 observations on 9 variables, a binary target (`Outcome`) and medical predictors such as the number of pregnancies, plasma glucose, BMI, insulin level and age. Every value is synthetic; no row corresponds to a real person.

Format

A data frame with 768 observations on the following 9 variables.

Pregnancies Number of times pregnant (0 is a valid value)

Glucose Plasma glucose concentration at 2 hours in an oral glucose tolerance test

BloodPressure Diastolic blood pressure (mm Hg)

SkinThickness Triceps skin fold thickness (mm)

Insulin 2-Hour serum insulin (mu U/ml)

BMI Body mass index (weight in kg/(height in m)^2)

DiabetesPedigreeFunction Diabetes pedigree function

Age Age in years

Outcome Diabetes (`yes` or `no`)

Details

The original data were collected by the National Institute of Diabetes and Digestive and Kidney Diseases from female patients of Pima Indian heritage aged at least 21. The team that collected them has stated that the data should not have been distributed and has asked for redistribution to stop, while a large body of teaching material and package examples depends on them. This synthetic version exists so that such material keeps working without real patient records being circulated.

The schema follows `PimaIndiansDiabetes2` from package **mlbench**: physically impossible values (a zero glucose level, blood pressure, skin fold thickness, insulin level or BMI) are coded as `NA`, whereas 0 pregnancies is a valid value. Column names, types and the levels of `Outcome` are those VIM has used since version 6, so existing code runs unchanged; only the values differ from the real data shipped before VIM 7.3.0.

Missing values in the original are not scattered at random: `BloodPressure`, `SkinThickness` and `Insulin` are frequently missing together, and the pattern is age-related. The pattern itself was synthesised as a variable and the missing values imposed from it, so that complete-case and imputation

examples behave as they did before (placing missing values independently would produce the joint pattern about five times instead of the observed 19 to 24).

Results computed on these data resemble published analyses of the original without matching them: a logistic regression reaches an AUC of 0.868 against 0.862 on the original, and a model fitted only on the synthetic data reaches 0.854 on the real records. Individual correlations are less reliable – most come out slightly weaker than in the original, but the association between number of pregnancies and age is considerably stronger (0.74 against 0.54). Use the data for teaching, examples and method demonstrations; any result computed on them describes the synthesis, not the Pima population, and must not be read as evidence about diabetes.

Author(s)

Matthias Templ <matthias.templ@fhnw.ch> (synthetic version)

Source

Synthetic. Generated by Matthias Templ from `PimaIndiansDiabetes2` (package **mlbench**) with the **synvey** package – sequential robust synthesis with a random-forest backend and the missingness pattern treated as a structural variable (synvey 0.1.0, R 4.5.2, seed 20260804):

```
SynthDiabetes <- synvey::robsynth(PimaIndiansDiabetes2,
                                  method      = "robust_rf",
                                  missingness = "structural",
                                  coarsen     = "age",
                                  target      = "sample",
                                  seed        = 20260804)$synth
```

This configuration was selected from a comparison of five synthesis packages under two designs for the missingness mechanism. Candidates reproducing any original record verbatim were excluded; the data set contains no such record. **synvey** is not on CRAN; it is available from https://github.com/matthias-da/synvey. For VIM the columns were renamed to the historical names (`pregnant`, `glucose`, `pressure`, `triceps`, `insulin`, `mass`, `pedigree`, `age`, `diabetes` became `Pregnancies`, `Glucose`, `BloodPressure`, `SkinThickness`, `Insulin`, `BMI`, `DiabetesPedigreeFunction`, `Age`, `Outcome`), counts are stored as integers and the `Outcome` levels `neg`/`pos` as `no`/`yes`. The original data are neither distributed with VIM nor needed to use this data set.

References

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care (pp. 261–265). IEEE Computer Society Press. https://pmc.ncbi.nlm.nih.gov/articles/PMC2245318/ (The study behind the original data; the original data are not redistributed here.)

Examples

```
data(diabetes)
summary(diabetes)
```

```
# missingness is structured rather than scattered
aggr(diabetes)
```

`evaluation` *Error performance measures*

Description

Various error measures evaluating the quality of imputations

Usage

```
evaluation(x, y, m, vartypes = "guess", where = NULL)

nrmse(x, y, m)

pfc(x, y, m)

msecov(x, y)

msecor(x, y)
```

Arguments

`x`	matrix or data frame
`y`	matrix or data frame of the same size as x
`m`	the indicator matrix for missing cells (kept for backward compatibility; `where` is the documented name)
`vartypes`	a vector of length ncol(x) specifying the variable types (`"numeric"` or `"factor"`). The default `"guess"` infers the types from the columns of `x` (numeric columns become `"numeric"`, everything else `"factor"`).
`where`	the indicator matrix for missing cells under its documented name – the amputed-cell mask as returned in `makeMissing()`'s `"where"` attribute. Supply either `m` or `where`, not both.

Details

This function has been mainly written for procudures that evaluate imputation or replacement of rounded zeros. The ni parameter can thus, e.g. be used for expressing the number of rounded zeros.

Value

the error measures value

Author(s)

Matthias Templ

References

M. Templ, A. Kowarik, P. Filzmoser (2011) Iterative stepwise regression imputation using standard and robust methods. *Computational Statistics & Data Analysis*, Vol. 55, pp. 2793-2806.

Examples

```
data(iris)
iris_orig <- iris_imp <- iris
iris_imp$Sepal.Length[sample(1:nrow(iris), 10)] <- NA
iris_imp$Sepal.Width[sample(1:nrow(iris), 10)] <- NA
iris_imp$Species[sample(1:nrow(iris), 10)] <- NA
m <- is.na(iris_imp)
iris_imp <- kNN(iris_imp, imp_var = FALSE)
evaluation(iris_orig, iris_imp, m = m, vartypes = c(rep("numeric", 4), "factor"))
msecov(iris_orig[, 1:4], iris_imp[, 1:4])
```

---

food *Food consumption*

Description

The relative consumption of certain food items in European and Scandinavian countries.

Format

A data frame with 16 observations on the following 21 variables.

Details

The numbers represent the percentage of the population consuming that food type.

Source

https://openmv.net/info/food-consumption

Examples

```
data(food)
str(food)
aggr(food)
```

gapMiss *Missing value gap statistics*

Description

Computes the average missing value gap of a vector.

Usage

```
gapMiss(x, what = mean)
```

Arguments

x	a numeric vector
what	default is the arithmetic mean. One can include an own function that returns a vector of lenght 1 (e.g. median)

Details

The length of each sequence of missing values (gap) in a vector is calculated and the mean gap is reported

Value

The gap statistics

Author(s)

Matthias Templ based on a suggestion and draft from Huang Tian Yuan.

Examples

```
v <- rnorm(20)
v[3] <- NA
v[6:9] <- NA
v[13:17] <- NA
v
gapMiss(v)
gapMiss(v, what = median)
gapMiss(v, what = function(x) mean(x, trim = 0.1))
gapMiss(v, what = var)
```

gowerD *Computes the extended Gower distance of two data sets*

Description

The function gowerD is used by kNN to compute the distances for numerical, factor ordered and semi-continous variables.

Usage

```
gowerD(
  data.x,
  data.y = data.x,
  weights = rep(1, ncol(data.x)),
  numerical = colnames(data.x),
  factors = vector(),
  orders = vector(),
  mixed = vector(),
  levOrders = vector(),
  mixed.constant = rep(0, length(mixed)),
  returnIndex = FALSE,
  nMin = 1L,
  returnMin = FALSE,
  methodStand = "range"
)
```

Arguments

data.x	data frame
data.y	data frame
weights	numeric vector providing weights for the observations in x
numerical	names of numerical variables
factors	names of factor variables
orders	names of ordered variables
mixed	names of mixed variables
levOrders	vector with number of levels for each orders variable
mixed.constant	vector with length equal to the number of semi-continuous variables specifying the point of the semi-continuous distribution with non-zero probability
returnIndex	logical if TRUE return the index of the minimum distance
nMin	integer number of values with smallest distance to be returned
returnMin	logical if the computed distances for the indices should be returned
methodStand	character either "range" or "iqr", iqr is more robust for outliers

Details

Numerical and semi-continuous (mixed) variables are standardised by their range (or IQR, see `methodStand`) before the distance is computed; for mixed variables the point mass (`mixed.constant`) is excluded from the range estimate and scaled along with the data.

Missing values in a distance variable are replaced by a sentinel (the pooled column maximum plus one) after standardisation. Two records both missing the same variable therefore have distance 0 on it, and a missing-vs-observed pair contributes up to about 2 (rather than a variable's nominal maximum of 1). This sentinel convention departs from Gower's (1971) omit-and-renormalise rule and biases nearest-neighbour selection towards records that share the same missingness pattern.

The distance computation is parallelised with OpenMP where R was built with OpenMP support. The number of threads is `getOption("VIM.ncores")`; if the option is unset, at most 2 threads are used while the package is checked (`R CMD check` sets `_R_CHECK_LIMIT_CORES_`; CRAN policy allows at most two cores) and OpenMP's default – typically all cores, or `OMP_NUM_THREADS` – otherwise. Results do not depend on the number of threads.

returnIndex=FALSE: a numerical matrix n x m with the computed distances returnIndex=TRUE: a named list with "ind" containing the requested indices and "mins" the computed distances

Examples

```
data(sleep)
# all variables used as numerical
gowerD(sleep)

# split in numerical an
gowerD(sleep, numerical = c("BodyWgt", "BrainWgt", "NonD", "Dream", "Sleep", "Span", "Gest"),
  orders = c("Pred","Exp","Danger"), levOrders = c(5,5,5))

# as before but only returning the index of the closest observation
gowerD(sleep, numerical = c("BodyWgt", "BrainWgt", "NonD", "Dream", "Sleep", "Span", "Gest"),
  orders = c("Pred","Exp","Danger"), levOrders = c(5,5,5), returnIndex = TRUE)
```

---

growdotMiss *Growing dot map with information about missing/imputed values*

Description

Map with dots whose sizes correspond to the values in a certain variable. Observations with missing/imputed values in additional variables are highlighted.

Usage

```
growdotMiss(
  x,
  coords,
  map,
  pos = 1,
```

```
  delimiter = NULL,
  selection = c("any", "all"),
  log = FALSE,
  col = c("skyblue", "red", "skyblue4", "red4", "orange", "orange4"),
  border = par("bg"),
  alpha = NULL,
  scale = NULL,
  size = NULL,
  exp = c(0, 0.95, 0.05),
  col.map = grey(0.5),
  legend = TRUE,
  legtitle = "Legend",
  cex.legtitle = par("cex"),
  cex.legtext = par("cex"),
  ncircles = 6,
  ndigits = 1,
  interactive = TRUE,
  ...
)
```

Arguments

x	a vector, matrix or data.frame.
coords	a matrix or data.frame with two columns giving the spatial coordinates of the observations.
map	a background map to be passed to bgmap().
pos	a numeric value giving the index of the variable determining the dot sizes.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
selection	the selection method for highlighting missing/imputed values in multiple additional variables. Possible values are "any" (highlighting of missing/imputed values in *any* of the additional variables) and "all" (highlighting of missing/imputed values in *all* of the additional variables).
log	a logical indicating whether the variable given by pos should be log-transformed.
col	a vector of length six giving the colors to be used in the plot. If only one color is supplied, it is used for the borders of non-highlighted dots and the surface area of highlighted dots. Else if two colors are supplied, they are recycled.
border	a vector of length four giving the colors to be used for the borders of the growing dots. Use NA to omit borders.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
scale	scaling factor of the map.

size	a vector of length two giving the sizes for the smallest and largest dots.
exp	a vector of length three giving the factors that define the shape of the exponential function (see 'Details').
col.map	the color to be used for the background map.
legend	a logical indicating whether a legend should be plotted.
legtitle	the title for the legend.
cex.legtitle	the character expansion factor to be used for the title of the legend.
cex.legtext	the character expansion factor to be used in the legend.
ncircles	the number of circles displayed in the legend.
ndigits	the number of digits displayed in the legend. Note that \ this is just a suggestion (see format()).
interactive	a logical indicating whether information about certain observations can be displayed interactively (see 'Details').
...	for growdotMiss, further arguments and graphical parameters to be passed to bgmap(). For bubbleMiss, the arguments to be passed to growdotMiss.

Details

The smallest dots correspond to the 10\ the 99\ defining the shape of the exponential function. Missings/imputed missings in the variable of interest will be drawn as rectangles.

If interactive=TRUE, detailed information for an observation can be printed on the console by clicking on the corresponding point. Clicking in a region that does not contain any points quits the interactive session.

Note

The function was renamed to growdotMiss in version 1.3. bubbleMiss is a (deprecated) wrapper for growdotMiss for back compatibility with older versions. However, due to extended functionality, some of the argument positions have changed.

The code is based on (removed from CRAN) bubbleFIN from package StatDA.

Author(s)

Andreas Alfons, Matthias Templ, Peter Filzmoser, Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

bgmap(), mapMiss(), colormapMiss()

Examples

```
data(chorizonDL, package = "VIM")
data(kola.background, package = "VIM")
coo <- chorizonDL[, c("XCOO", "YCOO")]
## for missing values
x <- chorizonDL[, c("Ca","As", "Bi")]
growdotMiss(x, coo, kola.background, border = "white")

## for imputed values
x_imp <- kNN(chorizonDL[,c("Ca","As","Bi" )])
growdotMiss(x_imp, coo, kola.background, delimiter = "_imp", border = "white")
```

`histMiss` *Histogram with information about missing/imputed values*

Description

Histogram with highlighting of missing/imputed values in other variables by splitting each bin into two parts. Additionally, information about missing/imputed values in the variable of interest is shown on the right hand side.

Usage

```
histMiss(
  x,
  delimiter = NULL,
  pos = 1,
  selection = c("any", "all"),
  breaks = "Sturges",
  right = TRUE,
  col = c("skyblue", "red", "skyblue4", "red4", "orange", "orange4"),
  border = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  axes = TRUE,
  only.miss = TRUE,
  miss.labels = axes,
  interactive = TRUE,
  ...
)
```

Arguments

`x`	a vector, matrix or `data.frame`.

delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
pos	a numeric value giving the index of the variable of interest. Additional variables in x are used for highlighting.
selection	the selection method for highlighting missing/imputed values in multiple additional variables. Possible values are "any" (highlighting of missing/imputed values in *any* of the additional variables) and "all" (highlighting of missing/imputed values in *all* of the additional variables).
breaks	either a character string naming an algorithm to compute the breakpoints (see hist()), or a numeric value giving the number of cells.
right	logical; if TRUE, the histogram cells are right-closed (left-open) intervals.
col	a vector of length six giving the colors to be used. If only one color is supplied, the bars are transparent and the supplied color is used for highlighting missing/imputed values. Else if two colors are supplied, they are recycled.
border	the color to be used for the border of the cells. Use border=NA to omit borders.
main, sub	main and sub title.
xlab, ylab	axis labels.
axes	a logical indicating whether axes should be drawn on the plot.
only.miss	logical; if TRUE, the missing/imputed values in the first variable are visualized by a single bar. Otherwise, a small barplot is drawn on the right hand side (see 'Details').
miss.labels	either a logical indicating whether label(s) should be plotted below the bar(s) on the right hand side, or a character string or vector giving the label(s) (see 'Details').
interactive	a logical indicating whether the variables can be switched interactively (see 'Details').
...	further graphical parameters to be passed to graphics::title() and graphics::axis().

Details

If more than one variable is supplied, the bins for the variable of interest will be split according to missingness/number of imputed missings in the additional variables.

If only.miss=TRUE, the missing/imputed values in the variable of interest are visualized by one bar on the right hand side. If additional variables are supplied, this bar is again split into two parts according to missingness/number of imputed missings in the additional variables.

Otherwise, a small barplot consisting of two bars is drawn on the right hand side. The first bar corresponds to observed values in the variable of interest and the second bar to missing/imputed values. Since these two bars are not on the same scale as the main barplot, a second y-axis is plotted on the right (if axes=TRUE). Each of the two bars are again split into two parts according to

missingness/number of imputed missings in the additional variables. Note that this display does not make sense if only one variable is supplied, therefore `only.miss` is ignored in that case.

If `interactive=TRUE`, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session. When switching to a categorical variable, a barplot is produced rather than a histogram.

Value

a list with the following components:

- breaks the breakpoints.
- counts the number of observations in each cell.
- missings the number of highlighted observations in each cell.
- mids the cell midpoints.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `axisnames` and `names.miss` can still be supplied to `...{}` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `miss.labels` instead.

Author(s)

Andreas Alfons, Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

`spineMiss()`, `barMiss()`

Other plotting functions: `aggr()`, `barMiss()`, `marginmatrix()`, `marginplot()`, `matrixplot()`, `mosaicMiss()`, `pairsVIM()`, `parcoordMiss()`, `pbox()`, `scattJitt()`, `scattMiss()`, `scattmatrixMiss()`, `spineMiss()`

Examples

```
data(tao, package = "VIM")
## for missing values
x <- tao[, c("Air.Temp", "Humidity")]
histMiss(x)
histMiss(x, only.miss = FALSE)

## for imputed values
x_IMPUTED <- kNN(tao[, c("Air.Temp", "Humidity")])
```

```
histMiss(x_IMPUTED, delimiter = "_imp")
histMiss(x_IMPUTED, delimiter = "_imp", only.miss = FALSE)
```

---

hotdeck *Hot-Deck Imputation*

Description

Implementation of the popular Sequential, Random (within a domain) hot-deck algorithm for imputation.

Usage

```
hotdeck(
  data,
  variable = NULL,
  ord_var = NULL,
  domain_var = NULL,
  makeNA = NULL,
  NAcond = NULL,
  impNA = TRUE,
  donorcond = NULL,
  imp_var = TRUE,
  imp_suffix = "imp"
)
```

Arguments

data	data.frame or matrix
variable	variables where missing values should be imputed (not overlapping with ord_var)
ord_var	variables for sorting the data set before imputation (not overlapping with variable)
domain_var	variables for building domains and impute within these domains
makeNA	list of length equal to the number of variables, with values, that should be converted to NA for each variable
NAcond	list of length equal to the number of variables, with a condition for imputing a NA
impNA	TRUE/FALSE whether NA should be imputed
donorcond	list of length equal to the number of variables, with a donorcond condition as character string. e.g. ">5" or c(">5","<10). If the list element for a variable is NULL no condition will be applied for this variable.
imp_var	TRUE/FALSE if a TRUE/FALSE variables for each imputed variable should be created show the imputation status
imp_suffix	suffix for the TRUE/FALSE variables showing the imputation status

Value

the imputed data set.

Note

If the sequential hotdeck does not lead to a suitable, a random donor in the group will be used.

Author(s)

Alexander Kowarik

References

A. Kowarik, M. Templ (2016) Imputation with R package VIM. *Journal of Statistical Software*, 74(7), 1-16.

See Also

Other imputation methods: `impPCA()`, `imputeCellEM()`, `imputeCellIRMI()`, `imputeCellM()`, `imputeCellMCD()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `irmi()`, `kNN()`, `matchImpute()`, `medianSamp()`, `rangerImpute()`, `regressionImp()`, `sampleCat()`, `vimmi`, `vimpute()`, `xgboostImpute()`

Examples

```
data(sleep)
sleepI <- hotdeck(sleep)
sleepI2 <- hotdeck(sleep,ord_var="BodyWgt",domain_var="Pred")

# Usage of donorcond in a simple example
sleepI3 <- hotdeck(
  sleep,
  variable = c("NonD", "Dream", "Sleep", "Span", "Gest"),
  ord_var = "BodyWgt", domain_var = "Pred",
  donorcond = list(">4", "<17", ">1.5", "%between%c(8,13)", ">5")
)

set.seed(132)
nRows <- 1e3
# Generate a data set with nRows rows and several variables
x <- data.frame(
  x = rnorm(nRows), y = rnorm(nRows),
  z = sample(LETTERS, nRows, replace = TRUE),
  d1 = sample(LETTERS[1:3], nRows, replace = TRUE),
  d2 = sample(LETTERS[1:2], nRows, replace = TRUE),
  o1 = rnorm(nRows), o2 = rnorm(nRows), o3 = rnorm(100)
)
origX <- x
x[sample(1:nRows,nRows/10), 1] <- NA
x[sample(1:nRows,nRows/10), 2] <- NA
x[sample(1:nRows,nRows/10), 3] <- NA
```

```
x[sample(1:nRows,nRows/10), 4] <- NA
xImp <- hotdeck(x,ord_var = c("o1", "o2", "o3"), domain_var = "d2")
```

`impPCA` *Iterative EM PCA imputation*

Description

Greedy algorithm for EM-PCA including robust methods

Usage

```
impPCA(
  x,
  method = "classical",
  m = 1,
  eps = 0.5,
  k = ncol(x) - 1,
  maxit = 100,
  boot = FALSE,
  verbose = TRUE
)
```

Arguments

`x`	data.frame or matrix
`method`	`"classical"` or `"mcd"` (robust estimation)
`m`	number of multiple imputations (only if parameter `boot` equals `TRUE`)
`eps`	threshold for convergence
`k`	number of principal components for reconstruction of `x`
`maxit`	maximum number of iterations
`boot`	residual bootstrap (if `TRUE`)
`verbose`	TRUE/FALSE if additional information about the imputation process should be printed

Value

the imputed data set. If `boot = FALSE` this is a data.frame. If `boot = TRUE` this is a list where each list element contains a data.frame.

Author(s)

Matthias Templ

References

Serneels, Sven and Verdonck, Tim (2008). Principal component analysis for data containing outliers and missing elements. Computational Statistics and Data Analysis, Elsevier, vol. 52(3), pages 1712-1727

See Also

Other imputation methods: hotdeck(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute(), xgboostImpute()

Examples

```
data(Animals, package = "MASS")
Animals$brain[19] <- Animals$brain[19] + 0.01
Animals <- log(Animals)
colnames(Animals) <- c("log(body)", "log(brain)")
Animals_na <- Animals
probs <- abs(Animals$`log(body)`^2)
probs <- rep(0.5, nrow(Animals))
probs[c(6,16,26)] <- 0
set.seed(1234)
Animals_na[sample(1:nrow(Animals), 10, prob = probs), "log(brain)"] <- NA
w <- is.na(Animals_na$`log(brain)`)
impPCA(Animals_na)
impPCA(Animals_na, method = "mcd")
impPCA(Animals_na, boot = TRUE, m = 10)
impPCA(Animals_na, method = "mcd", boot = TRUE)[[1]]
plot(`log(brain)` ~ `log(body)`, data = Animals, type = "n", ylab = "", xlab="")
mtext(text = "impPCA robust", side = 3)
points(Animals$`log(body)`[!w], Animals$`log(brain)`[!w])
points(Animals$`log(body)`[w], Animals$`log(brain)`[w], col = "grey", pch = 17)
imputed <- impPCA(Animals_na, method = "mcd", boot = TRUE)[[1]]
colnames(imputed) <- c("log(body)", "log(brain)")
points(imputed$`log(body)`[w], imputed$`log(brain)`[w], col = "red", pch = 20, cex = 1.4)
segments(x0 = Animals$`log(body)`[w], x1 = imputed$`log(body)`[w], y0 = Animals$`log(brain)`[w],
y1 = imputed$`log(brain)`[w], lty = 2, col = "grey")
legend("topleft", legend = c("non-missings", "set to missing", "imputed values"),
pch = c(1,17,20), col = c("black","grey","red"), cex = 0.7)
mape <- round(100* 1/sum(is.na(Animals_na$`log(brain)`)) * sum(abs((Animals$`log(brain)` -
imputed$`log(brain)`) / Animals$`log(brain)`)), 2)
s2 <- var(Animals$`log(brain)`)
nrmse <- round(sqrt(1/sum(is.na(Animals_na$`log(brain)`)) * sum(abs((Animals$`log(brain)` -
imputed$`log(brain)`) / s2))), 2)
text(x = 8, y = 1.5, labels = paste("MAPE =", mape))
text(x = 8, y = 0.5, labels = paste("NRMSE =", nrmse))
```

`imputeCellEM` *Cellwise-robust EM imputation for mixed data*

Description

EM algorithm with latent contamination indicators that jointly estimates clean distribution parameters and identifies cellwise outliers. Each continuous cell has a posterior probability of being clean vs contaminated. The clean distribution is modeled as multivariate normal for continuous variables, with categorical variables handled via conditional multinomial logistic regression.

Usage

```
imputeCellEM(
  data,
  maxit_em = 100,
  eps_em = 0.005,
  gamma_init = 3,
  eps_init = 0.1,
  uncert = "conditional",
  conditioning = "weighted",
  trust_min = 0.5,
  trace = FALSE
)
```

Arguments

`data`	data.frame with missing values (mixed continuous + categorical).
`maxit_em`	maximum EM iterations (default: 100).
`eps_em`	convergence tolerance on the relative change in estimated parameters (mu, Sigma). Default: 5e-3.
`gamma_init`	initial scale inflation factor for the contamination distribution. Contaminated cells are modeled as having variance `(gamma * sigma)^2` with `gamma > 1`. Default: 3.
`eps_init`	initial contamination probability per variable (default: 0.1). Must be in $(0, 0.5)$.
`uncert`	imputation uncertainty method: `"conditional"` (default) draws from the conditional normal distribution, or `"pmm"` uses predictive mean matching.
`conditioning`	how observed cells enter the E-step conditional moments. `"weighted"` (default) multiplies each conditioning deviation $x_{ik} - \mu_k$ by its current cell weight w_{ik} – the posterior-expected clean deviation, so likely contaminated cells are shrunk towards the mean before they enter the conditional. `"trust"` conditions only on trusted cells: deviations with $w_{ik} <$ `trust_min` are set to zero (hard threshold). `"unweighted"` uses the raw deviations (all observed cells fully trusted, as in early versions).
`trust_min`	trust threshold in $(0, 1)$ used when `conditioning = "trust"` (default: 0.5). Ignored otherwise.
`trace`	logical; if `TRUE`, print progress information.

Details

The algorithm proceeds as follows:

1. **Initialization.** Missing values are filled by `initialise` (medians for continuous, modes for categorical). Initial location and scale are estimated robustly (median and MAD). Cell weights are initialized to 1.
2. **E-step** (for each continuous variable j, each observation i):
   - Compute the conditional mean and variance of x_{ij} given the other continuous variables, using the current μ and Σ.
   - For observed cells: compute the posterior probability that the cell is clean vs contaminated, yielding cell weight w_{ij}.
   - For missing cells: impute from the conditional distribution (with optional PMM).
3. **M-step**:
   - Update μ: cell-weighted mean.
   - Update Σ: pairwise cell-weighted covariance using $w_{ij} \cdot w_{ik}$ (not row-level min).
   - Update contamination rates: $\varepsilon_j = 1 - \text{mean}(w_{ij})$ over observed cells.
   - Update contamination scale γ_j from weighted variance of contaminated cells.
   - For categorical variables: fit weighted multinomial logistic with row weights derived from continuous cell weights.
4. **Convergence.** Check relative change in estimated parameters (μ, Σ); stop when below `eps_em` or after `maxit_em` iterations. The observed-data log-likelihood (sum of per-variable conditional mixture log-likelihoods) is tracked for diagnostics.

This method differs from cellGMM (Zaccaria et al., 2025) in using a single clean component rather than a mixture of clean clusters, and in supporting mixed continuous + categorical data.

The `pseudo_loglik` component is a composite (pseudo) log-likelihood: the sum of per-variable conditional mixture log-likelihoods, not the proper observed-data joint log-likelihood. It is useful for monitoring convergence but should not be compared across models or used for model selection criteria such as AIC/BIC.

This implementation is an ECM (Expectation Conditional Maximization) variant rather than a pure EM algorithm, because the conditional variance in the E-step is computed using the updated Sigma from the current M-step rather than the Sigma from the previous iteration. As a result, strict log-likelihood monotonicity is not guaranteed, but is observed empirically in practice.

Value

A list with components:

`data_imputed`	the imputed data.frame
`cellweights`	n x p matrix of posterior clean probabilities. Continuous observed cells have values in $[0, 1]$; missing cells and categorical columns have weight 1.
`mu`	estimated clean location vector (continuous variables only)
`Sigma`	estimated clean covariance matrix (continuous variables only)
`epsilon`	named numeric vector of estimated per-variable contamination rates (continuous variables only)

converged	logical indicating convergence
iterations	number of EM iterations performed
pseudo_loglik	numeric vector of composite (pseudo) log-likelihood values, one per iteration (computed after each M-step). This is a sum of per-variable conditional mixture log-likelihoods, not the proper observed-data joint log-likelihood.

Note

Model uncertainty via bootstrap (Rubin's combining rules for multiple imputation) is not yet implemented. The current version provides single imputation with stochastic uncertainty (PMM or residual draw). For valid multiple imputation, call the function repeatedly with different seeds and combine using Rubin's rules.

Author(s)

Matthias Templ

References

A.P. Dempster, N.M. Laird, D.B. Rubin (1977) Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1), 1–38.

C.F.J. Wu (1983) On the Convergence Properties of the EM Algorithm. *The Annals of Statistics*, 11(1), 95–103.

J. Raymaekers, P.J. Rousseeuw (2024) The cellwise minimum covariance determinant estimator. *Journal of the American Statistical Association*, 119(548), 2610–2621.

G. Zaccaria, L.A. Garcia-Escudero, F. Greselin, A. Mayo-Iscar (2025) Cellwise outlier detection in heterogeneous populations. *Technometrics*, 67(4), 643–654.

M. Templ, A. Kowarik, P. Filzmoser (2011) Iterative stepwise regression imputation using standard and robust methods. *Computational Statistics & Data Analysis*, Vol. 55, pp. 2793–2806.

See Also

imputeCellIRMI, imputeCellM, initialise, irmi

Other imputation methods: hotdeck(), impPCA(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute(), xgboostImpute()

Examples

```
data(sleep, package = "VIM")
result <- imputeCellEM(sleep)
head(result$data_imputed)

# Inspect estimated contamination rates
result$epsilon

# Cell weight matrix (1 = clean, low = likely contaminated)
image(result$cellweights, main = "Cell weights")
```

```
# Log-likelihood trace
plot(result$pseudo_loglik, type = "b", xlab = "Iteration",
     ylab = "Pseudo log-likelihood")

# With predictive mean matching for imputation
result2 <- imputeCellEM(sleep, uncert = "pmm", trace = TRUE)

# Mixed data example
data(testdata)
result3 <- imputeCellEM(testdata$wna)
```

`imputeCellIRMI` *Cellwise-robust iterative regression imputation for mixed data*

Description

Extends IRMI (Templ, Kowarik, and Filzmoser, 2011) with cellwise contamination handling. Each conditional regression uses a cell-weighted IRWLS engine where per-cell weights in the design matrix downweight contaminated cells without discarding entire observations.

Usage

```
imputeCellIRMI(
  data,
  method = "tukey",
  alpha = NULL,
  maxit = 100,
  maxit_irwls = 50,
  eps = 0.005,
  eps_irwls = 1e-06,
  uncert = "pmm",
  weight_update = "multivariate",
  init_weights = "ddc",
  hard_threshold = 0.5,
  trace = FALSE
)
```

Arguments

`data`	a `data.frame` with missing values (mixed continuous and categorical variables are supported).
`method`	weight function: `"tukey"` (default, Tukey bisquare) or `"huber"` (Huber).
`alpha`	tuning constant. `NULL` (default) uses 1.345 for Huber and 4.685 for Tukey, giving 95% efficiency at the normal model.

`maxit`	maximum number of outer IRMI iterations (default: 100).
`maxit_irwls`	maximum number of inner IRWLS iterations per regression (default: 50).
`eps`	convergence tolerance for the outer loop (default: 5e-3). Convergence is declared when the relative change in imputed values falls below this threshold.
`eps_irwls`	convergence tolerance for the inner IRWLS (default: 1e-6).
`uncert`	imputation uncertainty method: `"pmm"` (predictive mean matching, default), `"normalerror"` (add normal noise), or `"resid"` (bootstrap residual).
`weight_update`	strategy for updating cell weights between outer iterations: `"multivariate"` (default) uses an MCD-based multivariate update for weight coherence across variables, or `"univariate"` updates each variable independently from its residuals.
`init_weights`	method for initialising cell weights, one of `"ddc"` (default; DetectDeviatingCells, requires the **cellWise** package and falls back to univariate weights when it is unavailable), `"univariate"` (per-column median/MAD standardisation), or `"mcd"` (minimum covariance determinant on the continuous block). The default is `"ddc"` because `"mcd"` downweights high-leverage points that carry the regression signal, which can make imputation worse than unconditional median imputation.
`hard_threshold`	numeric in $[0, 1]$. After convergence, cells with weight below this value are flagged as contaminated (default: 0.5).
`trace`	logical; if `TRUE`, print progress information.

Details

The algorithm works iteratively: in each outer iteration, every variable with missing values is used as response in a conditional regression on all remaining variables. For continuous responses, the custom `cellIRWLS()` engine fits a weighted regression where each cell in the design matrix receives its own weight reflecting potential cellwise contamination. For categorical responses, a weighted multinomial model is used. After each regression, cell weights for the response variable are updated from the residuals.

The algorithm proceeds as follows:

1. Missing values are initialised using `initialise`.
2. Initial cell weights are computed with `cellWeights()` on all continuous variables in the initialised data.
3. **Outer loop** (up to `maxit` iterations):
   - For each variable j with missing values:
     - Form predictor matrix X (all other variables) and response y (variable j).
     - If j is continuous: fit `cellIRWLS(X, y, w_cell, w_response)` and impute missing values in j using the fitted model plus uncertainty.
     - If j is categorical: fit `nnet::multinom()` with row weights derived from the cell weight matrix and impute by sampling from predicted probabilities.
     - Update cell weights for j from residuals via `cellWeightsFromResiduals()`.
   - Check convergence: relative change in imputed values falls below `eps`.

Value

A list with components:

`data_imputed`	the imputed `data.frame`.
`cellweights`	$n \times p$ matrix of final cell weights (1 = clean, 0 = fully downweighted). Categorical columns always have weight 1.
`converged`	logical indicating whether the outer loop converged.
`iterations`	number of outer iterations used.

Author(s)

Matthias Templ

References

Templ, M., Kowarik, A. and Filzmoser, P. (2011). Iterative stepwise regression imputation using standard and robust methods. *Computational Statistics & Data Analysis*, **55**(10), 2793–2806.

See Also

`imputeCellM`, `imputeCellEM`, `initialise`, `irmi`

Other imputation methods: `hotdeck()`, `impPCA()`, `imputeCellEM()`, `imputeCellM()`, `imputeCellMCD()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `irmi()`, `kNN()`, `matchImpute()`, `medianSamp()`, `rangerImpute()`, `regressionImp()`, `sampleCat()`, `vimmi`, `vimpute()`, `xgboostImpute()`

Examples

```
data(sleep, package = "VIM")
result <- imputeCellIRMI(sleep)
head(result$data_imputed)
image(result$cellweights, main = "Cell weights")

# With Huber weights (less aggressive downweighting)
result2 <- imputeCellIRMI(sleep, method = "huber", trace = TRUE)

# Mixed data example
data(testdata)
result3 <- imputeCellIRMI(testdata$wna)
```

`imputeCellM` *Cellwise M-estimation imputation*

Description

Impute missing values using a cell-weighted M-estimation approach. Each cell in the predictor matrix receives its own weight reflecting potential cellwise contamination, so that contaminated predictor cells are downweighted without discarding entire observations.

Usage

```
imputeCellM(
  formula,
  data,
  method = "tukey",
  alpha = NULL,
  maxit_irwls = 50,
  eps_irwls = 1e-06,
  uncert = "pmm",
  value_back = "all",
  maxit = 10,
  eps = 0.005,
  trace = FALSE
)
```

Arguments

`formula`	a model formula (e.g., `y ~ x1 + x2`) describing a single response to impute, or a `data.frame`/matrix with missing values; in the latter case all variables with missing values are imputed by chained equations and `data` must not be supplied.
`data`	data.frame containing the data (formula interface only).
`method`	weight function: `"tukey"` (default) or `"huber"`. Tukey bisquare is recommended because the consistency proof requires redescending weights.
`alpha`	tuning constant. `NULL` (default) uses 4.685 for Tukey and 1.345 for Huber.
`maxit_irwls`	maximum IRWLS iterations (default: 50).
`eps_irwls`	convergence tolerance for IRWLS (default: 1e-6).
`uncert`	imputation uncertainty method: `"pmm"` (default), `"normalerror"`, `"resid"`, or `"none"` (deterministic predictions; categorical variables are imputed by the most probable category).
`value_back`	`"all"` (default) returns the complete dataset, or `"ymiss"` returns only the imputed values (formula interface only; ignored with a data.frame first argument).
`maxit`	maximum number of chained-equation sweeps (data.frame interface only; default: 10).

eps	convergence tolerance for the chained sweeps (data.frame interface only; default: 5e-3). Convergence is declared when the relative change in imputed values falls below this threshold.
trace	logical; if TRUE, print progress of the chained sweeps (data.frame interface only).

Details

The function has two interfaces: with a model formula, a single response variable is imputed from the specified predictors; with a data.frame (or matrix) as first argument, all variables with missing values are imputed by chained equations, i.e. each such variable is regressed on all remaining variables and the sweeps are iterated until the imputed values stabilise.

The formula interface is a lightweight single-response alternative to imputeCellIRMI. It fits one cell-weighted IRWLS regression using cellIRWLS() and imputes the missing values in the response variable. This is appropriate when only one variable needs imputation and a specific model formula is desired.

The data.frame interface runs the same per-variable machinery as a chained-equations algorithm: missing values are initialised (median/mode), then each variable with missing values in turn is used as response in a formula containing all remaining variables. Sweeps use deterministic predictions and are iterated until the relative change of the imputed values falls below eps (or maxit is reached); the requested uncert step is applied once after convergence.

For categorical response variables, a weighted multinomial model via multinom is fitted instead. Categorical predictors are not subject to the cellwise contamination model (their cell weights are always 1).

Value

If value_back = "ymiss", a named vector of imputed values (for rows that were originally missing) is returned. Otherwise, a list with components:

data_imputed the imputed data.frame (same structure as input)

cellweights n x p matrix of final cell weights (1 = clean, 0 = fully downweighted). Categorical columns always have weight 1.

converged logical; always TRUE for single-formula imputation, convergence of the sweeps for the data.frame interface

iterations integer; always 1L for single-formula imputation, the number of sweeps for the data.frame interface

Note

Model uncertainty via bootstrap (Rubin's combining rules for multiple imputation) is not yet implemented. The current version provides single imputation with stochastic uncertainty (PMM or residual draw). For valid multiple imputation, call the function repeatedly with different seeds and combine using Rubin's rules.

Author(s)

Matthias Templ

References

M. Templ, A. Kowarik, P. Filzmoser (2011) Iterative stepwise regression imputation using standard and robust methods. *Computational Statistics & Data Analysis*, Vol. 55, pp. 2793-2806.

See Also

imputeCellIRMI, imputeRobust

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute(), xgboostImpute()

Examples

```
data(sleep, package = "VIM")
# Impute Dream using BodyWgt and BrainWgt as predictors
result <- imputeCellM(Dream ~ BodyWgt + BrainWgt, data = sleep)
head(result)

# Return only imputed values
impvals <- imputeCellM(Dream ~ BodyWgt + BrainWgt, data = sleep,
                       value_back = "ymiss")

# Huber weights (less aggressive downweighting)
result2 <- imputeCellM(Dream ~ BodyWgt + BrainWgt, data = sleep,
                       method = "huber")

# Chained-equations interface: impute all variables with missings
result3 <- imputeCellM(sleep)
head(result3$data_imputed)
```

---

imputeCellMCD *Cellwise MCD-based imputation for mixed data*

Description

Extends the cellwise MCD approach (Raymaekers & Rousseeuw 2024) to mixed continuous + categorical data. Uses MCD for robust covariance estimation of the continuous block, computes cellwise weights from conditional residuals, then imputes missing values via conditional expectations (continuous) and weighted multinomial regression (categorical). Iterates until convergence.

Usage

```
imputeCellMCD(
  data,
  maxit = 50,
  eps = 0.005,
```

```
  method = "tukey",
  alpha = NULL,
  mcd_alpha = 0.75,
  hard_threshold = 0.5,
  mcd_observed = "all",
  init_method = "median",
  uncert = "conditional",
  m = 1L,
  boot = FALSE,
  trace = FALSE
)
```

Arguments

`data`	a `data.frame` with missing values (mixed continuous and categorical variables are supported).
`maxit`	maximum number of iterations (default: 50).
`eps`	convergence tolerance (default: 5e-3).
`method`	weight function for cell weights: `"tukey"` (default) or `"huber"`.
`alpha`	tuning constant. `NULL` (default) uses 4.685 for Tukey and 1.345 for Huber.
`mcd_alpha`	MCD concentration parameter (default: 0.75).
`hard_threshold`	numeric in $[0, 1]$. Before iteration, observed cells with initial MCD weight below this threshold are set to missing and re-imputed (detect-once preprocessing). Set to `NULL` or `0` to disable (default: 0.5).
`mcd_observed`	strategy for covariance estimation: `"all"` (default) runs MCD on all data including imputed values; `"weighted"` uses `cellWise::cwLocScat` with imputed cells receiving weight 0 (requires the **cellWise** package); `"pairwise"` uses pairwise robust (Gnanadesikan–Kettenring) covariances on observed cells only.
`init_method`	initialisation for missing values before iteration: `"median"` (default), `"knn"`, or `"irmi"`.
`uncert`	imputation uncertainty: `"conditional"` (default) adds noise from the conditional normal distribution.
`m`	number of multiple imputations (default: 1). If `m > 1`, a list of imputed datasets is returned.
`boot`	logical; intended to add bootstrap parameter uncertainty across the `m` imputations. **Currently ineffective**: the imputation step is a per-variable robust regression (`cellIRWLS`), so resampling the location/covariance estimate leaves the imputations essentially unchanged (measured between-imputation variance is not inflated relative to `boot = FALSE`). Proper parameter-uncertainty propagation requires bootstrapping the per-variable regression coefficients and is not yet implemented; see the note below. Default `FALSE`.
`trace`	logical; if `TRUE`, print progress information.

Value

A list with components:

`data_imputed`	the imputed `data.frame`.
`cellweights`	$n \times p$ matrix of cell weights. Continuous observed cells have values in $[0, 1]$; categorical columns always have weight 1 (cellwise detection is only applied to continuous variables).
`mu`	robust location estimate (continuous variables).
`Sigma`	robust covariance estimate (continuous variables).
`converged`	logical indicating convergence.
`iterations`	number of iterations performed.

Note

Multiple imputation with proper parameter-uncertainty propagation is not yet implemented for this method (see `boot`). `m > 1` returns repeated imputations whose between-imputation variability comes only from the stochastic uncertainty step (`uncert`) and estimator randomness, so downstream Rubin pooling understates total variance. Treat the current `m > 1` output as approximate.

Author(s)

Matthias Templ

References

Raymaekers, J. and Rousseeuw, P.J. (2024). The cellwise minimum covariance determinant estimator. *Journal of the American Statistical Association*, **119**(548), 2610–2621.

See Also

`imputeCellIRMI`, `imputeCellM`, `imputeCellEM`

Other imputation methods: `hotdeck()`, `impPCA()`, `imputeCellEM()`, `imputeCellIRMI()`, `imputeCellM()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `irmi()`, `kNN()`, `matchImpute()`, `medianSamp()`, `rangerImpute()`, `regressionImp()`, `sampleCat()`, `vimmi`, `vimpute()`, `xgboostImpute()`

Examples

```
## Not run:
data(sleep, package = "VIM")
result <- imputeCellMCD(sleep)
head(result$data_imputed)

# Inspect cell weights
image(result$cellweights, main = "Cell weights")

# With pairwise robust covariance
result2 <- imputeCellMCD(sleep, mcd_observed = "pairwise", trace = TRUE)
```

```
## End(Not run)
```

`imputeCellMM` *Cell-weighted MM imputation for mixed data (Path A)*

Description

Uses a cell-weighted MM-estimator for each variable regression: S-step on unweighted data (high breakdown), M-step with cell weights (fine-grained downweighting). This preserves the S-estimator's breakdown guarantee while incorporating cellwise information.

Usage

```
imputeCellMM(
  data,
  maxit = 50,
  eps = 0.005,
  alpha_weight = 4.685,
  init_weights = "mcd",
  uncert = "pmm",
  trace = FALSE
)
```

Arguments

`data`	data.frame with missing values
`maxit`	maximum outer IRMI iterations (default: 50)
`eps`	convergence tolerance (default: 5e-3)
`alpha_weight`	tuning constant for bisquare cell weights (default: 4.685)
`init_weights`	initial cell weight method: `"mcd"` or `"univariate"` (default: `"mcd"`)
`uncert`	uncertainty method: `"pmm"`, `"normalerror"`, or `"none"`
`trace`	logical

Value

list with data_imputed, cellweights, converged, iterations

Author(s)

Matthias Templ

`imputeCellReg` *Cellwise-robust regression imputation for mixed data*

Description

IRMI-style imputation using cellwise-robust regression as the inner engine. Three engines are available: CRM (Filzmoser et al. 2020), Shooting S (Öllerer et al. 2016), and a cellwise-weighted MM hybrid.

Usage

```
imputeCellReg(
  data,
  engine = "crm",
  maxit = 50,
  eps = 0.005,
  uncert = "pmm",
  trace = FALSE
)
```

Arguments

`data`	data.frame with missing values (mixed continuous + categorical)
`engine`	regression engine: `"crm"` (default), `"cellwise-mm"`, or `"shooting-s"`
`maxit`	maximum outer IRMI iterations (default: 50)
`eps`	convergence tolerance (default: 5e-3)
`uncert`	imputation uncertainty: `"pmm"` (default), `"normalerror"`, or `"none"`
`trace`	logical; print progress

Details

The function cycles through all variables with missing values (IRMI framework), fitting a cellwise-robust regression of each variable on all others. The `engine` argument selects the regression method:

`"crm"` CRM (Cellwise Robust M-regression) from the `crmReg` package. Uses SPADIMO for cellwise outlier detection within each regression. Requires `crmReg`.

`"cellwise-mm"` Hybrid: compute cell weights via MCD conditional residuals, then fit MM-estimation (`lmrob`) with row weights derived from cell weights. The MM-estimator provides high breakdown point.

`"shooting-s"` Shooting S-estimator (Öllerer et al. 2016). Iterates between cellwise detection and S-estimation. Implemented from the published algorithm.

Categorical variables are imputed via weighted multinomial logistic regression, with row weights derived from the continuous cell weights.

Value

A list with components:

`data_imputed`	the imputed data.frame
`cellweights`	n x p matrix of cell weights (1 = clean)
`converged`	logical
`iterations`	number of outer iterations

Author(s)

Matthias Templ

References

P. Filzmoser, S. Höppner, I. Ortner, S. Serneels, S. Van Aelst (2020) Cellwise robust M regression. *Computational Statistics and Data Analysis*, 147, 106944.

V. Öllerer, A. Alfons, C. Croux (2016) The shooting S-estimator for robust regression. *Computational Statistics*, 31(3), 829–844.

`imputeCellwise` *Unified cellwise-robust imputation dispatcher*

Description

Convenience wrapper that dispatches to one of the three cellwise-robust imputation methods: `imputeCellIRMI`, `imputeCellM`, or `imputeCellEM`.

Usage

```
imputeCellwise(data, method = c("cellIRMI", "cellM", "cellEM"), ...)
```

Arguments

`data`	data.frame with missing values (mixed continuous + categorical).
`method`	imputation method: `"cellIRMI"` (default), `"cellM"`, or `"cellEM"`.
`...`	additional arguments passed to the chosen method.

Value

The return value of the dispatched function. See the documentation of the individual methods for details.

Note

Model uncertainty via bootstrap (Rubin's combining rules for multiple imputation) is not yet implemented. The current version provides single imputation with stochastic uncertainty (PMM or residual draw). For valid multiple imputation, call the function repeatedly with different seeds and combine using Rubin's rules.

Author(s)

Matthias Templ

See Also

imputeCellIRMI, imputeCellM, imputeCellEM

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute(), xgboostImpute()

Examples

```
data(sleep, package = "VIM")
result <- imputeCellwise(sleep, method = "cellIRMI")
head(result$data_imputed)
```

---

imputeRobust *Robust imputation*

Description

Multiple imputation using classical and robust methods accounting for model and imputation uncertainty.

Usage

```
imputeRobust(
  form,
  data,
  boot = TRUE,
  robustboot = "stratified",
  method = "MM",
  takeAll = TRUE,
  alpha = 0.75,
  uncert = "pmm",
  family = "Gaussian",
  value_back = "all"
)
```

Arguments

form	Model formulas as a list.
data	Data set to impute
boot	Accounting for model uncertainty with a classical bootstrap, Default: TRUE

robustboot	Accounting for model uncertainty with robust bootstrap methods, Default: 'stratified'
method	Imputation method, Default: 'MM'
takeAll	Missing values are intialized when TRUE, Default: TRUE
alpha	Relative size of good data points. Used for the robust bootstrap methods, Default: 0.75
uncert	Imputation uncertainty method, Default: 'pmm'
family	Not supported and ignored. Foreseen for future versions, Default: 'Gaussian'
value_back	Only observations with imputed values as return object (ymiss), or the whole data set, Default: 'all'

Details

Complex formulas can be provided for each variable in your data set.

Value

Imputed data set.

See Also

initialise lmrob gam pdist

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute(), xgboostImpute()

Examples

```
## Not run:
if(interactive()){
 #EXAMPLE1
 }

## End(Not run)
```

---

imputeRobustChain *FUNCTION_TITLE*

Description

FUNCTION_DESCRIPTION

Usage

```
imputeRobustChain(
  formulas = vector("list", ncol(data)),
  data,
  boot = TRUE,
  robustboot = TRUE,
  method = "lts",
  multinom.method = "multinom",
  takeAll = TRUE,
  eps = 0.5,
  maxit = 4,
  alpha = 0.5,
  uncert = "pmm",
  familiy = "Gaussian",
  value_back = "matrix",
  trace = FALSE
)
```

Arguments

formulas	PARAM_DESCRIPTION, Default: vector("list", ncol(data))
data	PARAM_DESCRIPTION
boot	PARAM_DESCRIPTION, Default: TRUE
robustboot	PARAM_DESCRIPTION, Default: TRUE
method	PARAM_DESCRIPTION, Default: 'lts'
multinom.method	PARAM_DESCRIPTION, Default: 'multinom'
takeAll	PARAM_DESCRIPTION, Default: TRUE
eps	PARAM_DESCRIPTION, Default: 0.5
maxit	PARAM_DESCRIPTION, Default: 4
alpha	PARAM_DESCRIPTION, Default: 0.5
uncert	PARAM_DESCRIPTION, Default: 'pmm'
familiy	PARAM_DESCRIPTION, Default: 'Gaussian'
value_back	PARAM_DESCRIPTION, Default: 'matrix'
trace	PARAM_DESCRIPTION, Default: FALSE

Details

DETAILS

Value

OUTPUT_DESCRIPTION

See Also

initialise outCoDa

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute(), xgboostImpute()

Examples

```
## Not run:
if(interactive()){
 #EXAMPLE1
 }

## End(Not run)
```

---

initialise *Initialization of missing values*

Description

Rough estimation of missing values in a vector according to its type.

Usage

```
initialise(x, mixed, method = "kNN", mixed.constant = NULL)
```

Arguments

x	a vector.
mixed	a character vector containing the names of variables of type mixed (semi-continous).
method	Method used for Initialization (median or kNN)
mixed.constant	vector with length equal to the number of semi-continuous variables specifying the point of the semi-continuous distribution with non-zero probability

Details

Missing values are imputed with the mean for vectors of class "numeric", with the median for vectors of class "integer", and with the mode for vectors of class "factor". Hence, x should be prepared in the following way: assign class "numeric" to numeric vectors, assign class "integer" to ordinal vectors, and assign class "factor" to nominal or binary vectors.

Value

the initialized vector.

Note

The function is used internally by some imputation algorithms.

Author(s)

Matthias Templ, modifications by Andreas Alfons

`irmi` *Iterative robust model-based imputation (IRMI)*

Description

In each step of the iteration, one variable is used as a response variable and the remaining variables serve as the regressors.

Usage

```
irmi(
  x,
  eps = 5,
  maxit = 100,
  mixed = NULL,
  mixed.constant = NULL,
  count = NULL,
  step = FALSE,
  robust = FALSE,
  takeAll = TRUE,
  noise = TRUE,
  noise.factor = 1,
  force = FALSE,
  robMethod = "lmrob",
  force.mixed = TRUE,
  mi = 1,
  addMixedFactors = FALSE,
  trace = FALSE,
  init.method = "kNN",
  modelFormulas = NULL,
  multinom.method = "multinom",
  imp_var = TRUE,
  imp_suffix = "imp"
)
```

Arguments

`x`	data.frame or matrix
`eps`	threshold for convergency

maxit	maximum number of iterations
mixed	column index of the semi-continuous variables
mixed.constant	vector with length equal to the number of semi-continuous variables specifying the point of the semi-continuous distribution with non-zero probability
count	column index of count variables
step	a stepwise model selection is applied when the parameter is set to TRUE
robust	if TRUE, robust regression methods will be applied
takeAll	takes information of (initialised) missings in the response as well for regression imputation.
noise	irmi has the option to add a random error term to the imputed values, this creates the possibility for multiple imputation. The error term has mean 0 and variance corresponding to the variance of the regression residuals.
noise.factor	amount of noise.
force	if TRUE, the algorithm tries to find a solution in any case, possible by using different robust methods automatically.
robMethod	regression method when the response is continuous. Default is MM-regression with lmrob.
force.mixed	if TRUE, the algorithm tries to find a solution in any case, possible by using different robust methods automatically.
mi	number of multiple imputations.
addMixedFactors	if TRUE add additional factor variable for each mixed variable as X variable in the regression
trace	Additional information about the iterations when trace equals TRUE.
init.method	Method for initialization of missing values (kNN or median)
modelFormulas	a named list with the name of variables for the rhs of the formulas, which must contain a rhs formula for each variable with missing values, it should look like 'list(y1=c("x1","x2"),y2=c("x1","x3"))" if factor variables for the mixed variables should be created for the regression models
multinom.method	Method for estimating the multinomial models (current default and only available method is multinom)
imp_var	TRUE/FALSE if a TRUE/FALSE variables for each imputed variable should be created show the imputation status
imp_suffix	suffix for the TRUE/FALSE variables showing the imputation status

Details

The method works sequentially and iterative. The method can deal with a mixture of continuous, semi-continuous, ordinal and nominal variables including outliers.

A full description of the method can be found in the mentioned reference.

Value

the imputed data set.

Note

Deprecated: `irmi()` is considered legacy functionality. For new projects, please use `vimpute()` instead, which provides a more modern, flexible and robust imputation framework.

Author(s)

Matthias Templ, Alexander Kowarik

References

M. Templ, A. Kowarik, P. Filzmoser (2011) Iterative stepwise regression imputation using standard and robust methods. *Computational Statistics & Data Analysis*, Vol. 55, pp. 2793-2806.

A. Kowarik, M. Templ (2016) Imputation with R package VIM. *Journal of Statistical Software*, 74(7), 1-16.

M. Templ (2023) *Visualization and Imputation of Missing Values*. Springer Publishing. Series in Computational Statistics. Cham. Switzerland. 463 pages. DOI: 10.1007/978-3-031-30073-8

See Also

Other imputation methods: `hotdeck()`, `impPCA()`, `imputeCellEM()`, `imputeCellIRMI()`, `imputeCellM()`, `imputeCellMCD()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `kNN()`, `matchImpute()`, `medianSamp()`, `rangerImpute()`, `regressionImp()`, `sampleCat()`, `vimmi`, `vimpute()`, `xgboostImpute()`

Examples

```
data(sleep)
irmi(sleep)

data(testdata)
imp_testdata1 <- irmi(testdata$wna, mixed = testdata$mixed)

# mixed.constant != 0 (-10)
testdata$wna$m1[testdata$wna$m1 == 0] <- -10
testdata$wna$m2 <- log(testdata$wna$m2 + 0.001)
imp_testdata2 <- irmi(
  testdata$wna,
  mixed = testdata$mixed,
  mixed.constant = c(-10,log(0.001))
)
imp_testdata2$m2 <- exp(imp_testdata2$m2) - 0.001

#example with fixed formulas for the variables with missing
form = list(
  NonD  = c("BodyWgt", "BrainWgt"),
  Dream = c("BodyWgt", "BrainWgt"),
  Sleep = c("BrainWgt"           ),
```

```
  Span  = c("BodyWgt"            ),
  Gest  = c("BodyWgt", "BrainWgt")
)
irmi(sleep, modelFormulas = form, trace = TRUE)

# Example with ordered variable
td <- testdata$wna
td$c1 <- as.ordered(td$c1)
irmi(td)
```

---

kNN *k-Nearest Neighbour Imputation*

Description

k-Nearest Neighbour Imputation based on a variation of the Gower Distance for numerical, categorical, ordered and semi-continous variables.

Usage

```
kNN(
  data,
  variable = colnames(data),
  k = 5,
  dist_var = colnames(data),
  weights = NULL,
  numFun = median,
  catFun = maxCat,
  makeNA = NULL,
  NAcond = NULL,
  impNA = TRUE,
  donorcond = NULL,
  mixed = vector(),
  mixed.constant = NULL,
  trace = FALSE,
  imp_var = TRUE,
  imp_suffix = "imp",
  addRF = FALSE,
  onlyRF = FALSE,
  addRandom = FALSE,
  useImputedDist = TRUE,
  weightDist = FALSE,
  methodStand = "range",
  ordFun = medianSamp
)
```

Arguments

data	data.frame or matrix
variable	variables where missing values should be imputed
k	number of Nearest Neighbours used
dist_var	names or variables to be used for distance calculation
weights	weights for the variables for distance calculation. If weights = "auto" weights will be selected based on variable importance from random forest regression, using function ranger::ranger(). Weights are calculated for each variable seperately.
numFun	function for aggregating the k Nearest Neighbours in the case of a numerical variable
catFun	function for aggregating the k Nearest Neighbours in the case of a categorical variable
makeNA	list of length equal to the number of variables, with values, that should be converted to NA for each variable
NAcond	list of length equal to the number of variables, with a condition for imputing a NA
impNA	TRUE/FALSE whether NA should be imputed
donorcond	list of length equal to the number of variables, with a donorcond condition as character string. e.g. a list element can be ">5" or c(">5","<10). If the list element for a variable is NULL no condition will be applied for this variable.
mixed	names of mixed variables
mixed.constant	vector with length equal to the number of semi-continuous variables specifying the point of the semi-continuous distribution with non-zero probability
trace	TRUE/FALSE if additional information about the imputation process should be printed
imp_var	TRUE/FALSE if a TRUE/FALSE variables for each imputed variable should be created show the imputation status
imp_suffix	suffix for the TRUE/FALSE variables showing the imputation status
addRF	TRUE/FALSE each variable will be modelled using random forest regression (ranger::ranger()) and used as additional distance variable.
onlyRF	TRUE/FALSE if TRUE only additional distance variables created from random forest regression will be used as distance variables.
addRandom	TRUE/FALSE if an additional random variable should be added for distance calculation
useImputedDist	TRUE/FALSE if an imputed value should be used for distance calculation for imputing another variable. Be aware that this results in a dependency on the ordering of the variables.
weightDist	TRUE/FALSE if the distances of the k nearest neighbours should be used as weights in the aggregation step
methodStand	either "range" or "iqr" to be used in the standardization of numeric vaiables in the gower distance
ordFun	function for aggregating the k Nearest Neighbours in the case of a ordered factor variable

Details

Numerical and semi-continuous (mixed) distance variables are standardised by their range (or IQR, see `methodStand`) so that each variable contributes on a comparable scale; for mixed variables the point mass (`mixed.constant`) is excluded from the range estimate.

Missing values in the distance variables (`dist_var`) are handled with a sentinel: after standardisation an `NA` is replaced by the maximum of the pooled column plus one. As a consequence two records that are both missing the same distance variable have distance 0 on that variable (they are treated as identical there), while a record missing the variable and one observing it contribute up to about twice a normal variable's share. In effect kNN is drawn towards donors that share the recipient's pattern of missingness in the distance variables; under a MAR mechanism this can select unrepresentative donors. This departs from Gower's (1971) convention of omitting non-comparable variables and renormalising the remaining weights. To limit the effect, prefer `dist_var` variables with few or no missing values.

Value

the imputed data set.

Author(s)

Alexander Kowarik, Statistik Austria

References

A. Kowarik, M. Templ (2016) Imputation with R package VIM. *Journal of Statistical Software*, 74(7), 1-16.

See Also

Other imputation methods: `hotdeck()`, `impPCA()`, `imputeCellEM()`, `imputeCellIRMI()`, `imputeCellM()`, `imputeCellMCD()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `irmi()`, `matchImpute()`, `medianSamp()`, `rangerImpute()`, `regressionImp()`, `sampleCat()`, `vimmi`, `vimpute()`, `xgboostImpute()`

Examples

```
data(sleep)
kNN(sleep)
library(laeken)
kNN(sleep, numFun = weightedMean, weightDist=TRUE)
```

kola.background *Background map for the Kola project data*

Description

Coordinates of the Kola background map.

Source

Kola Project (1993-1998)

References

Reimann, C., Filzmoser, P., Garrett, R.G. and Dutter, R. (2008) *Statistical Data Analysis Explained: Applied Environmental Statistics with R*. Wiley, 2008.

Examples

```
data(kola.background, package = "VIM")
bgmap(kola.background)
```

lse_synthetic *Synthetic Austrian Structural Business Survey data*

Description

A synthetic, non-confidential enterprise-level data set inspired by the Austrian Leistungs- und Strukturstatistik from reporting years 2021 onward.

Format

A data frame with 10000 observations and 38 variables.

Details

The variables cover the main dimensions described in the Statistik Austria standard documentation: economic activity, NUTS2 region, data source, employment structure, turnover, intermediate consumption, gross value added, personnel costs, gross operating surplus, and tangible investment.

The monetary variables are expressed in thousand euros. The observations are generated synthetically and do not contain real enterprise records. Companion validation rules are available as lse_synthetic_rules.

Source

Statistik Austria, Standard-Dokumentation zur Leistungs- und Strukturstatistik ab 2021, https://www.statistik.at/fileadmin/shared/QM/Standarddokumentationen/U/std_u_ls_ab_2021.pdf

Examples

```
data(lse_synthetic)
str(lse_synthetic)
```

---

lse_synthetic_rules *Validation rules for the synthetic LSE data*

Description

Accounting, additional edit, and combined validation rules for lse_synthetic. The rules are stored as plain data frames of rule text so that the data set does not depend on the `validate` package; build the corresponding `validate::validator` objects with `validate::validator(.data = lse_synthetic_rules$edit)` before passing them to `vimpute(method = "restricted")`.

Format

A named list with three data frames: `accounting` (36 rules), `extra` (56 rules), and `edit` (92 rules, the combination of both). Each has the columns

name rule name (`V01`, `V02`, ...)

rule the rule as R code, as accepted by `validate::validator(.data = )`

Examples

```
data(lse_synthetic_rules)
names(lse_synthetic_rules)
head(lse_synthetic_rules$edit)
if (requireNamespace("validate", quietly = TRUE)) {
  edit_rules <- validate::validator(.data = lse_synthetic_rules$edit)
  length(edit_rules)
}
```

makeMissing *Generate MCAR/MAR/MNAR missingness in complete data*

Description

Amputation generator for simulation studies (the variable-wise counterpart of `mice::ampute()`): takes complete data, sets a controlled share of cells to `NA` under a chosen missingness mechanism, and returns the amputed data together with an `attr(., "where")` indicator matrix that plugs directly into `evaluation()`, `nrmse()` and `pfc()`:

Usage

```
makeMissing(
  data,
  prop = 0.1,
  mechanism = c("MCAR", "MAR", "MNAR"),
  vars = NULL,
  weights = NULL,
  seed = NULL
)
```

Arguments

`data`	A complete data.frame or data.table (no missing values).
`prop`	Proportion of cells set to missing per target variable, strictly between 0 and 1.
`mechanism`	One of `"MCAR"` (default), `"MAR"`, `"MNAR"`.
`vars`	Character vector of target variables to receive missing values. Default: all columns.
`weights`	Optional named numeric vector. For `"MAR"`: the driver columns and their weights (numeric columns, excluding the target). For `"MNAR"`: an entry named after a target scales/flips its own-value score. Ignored for `"MCAR"`.
`seed`	Optional single number: applied via `set.seed()` before drawing, for reproducible amputation.

Details

```
amp <- makeMissing(dat, prop = 0.2, mechanism = "MAR")
imp <- vimpute(amp)
evaluation(dat, imp, m = attr(amp, "where"))
```

Mechanisms (applied per target variable, each receiving exactly `round(prop * nrow(data))` missing cells):

`"MCAR"` cells are drawn uniformly at random.

`"MAR"` the probability of a cell going missing grows with a weighted score of the *other* (observed) variables: rows are drawn with probabilities `plogis(z)` where `z` is the standardized weighted sum of the driver columns. Default drivers: all numeric columns except the target, with equal weights; use `weights` to select drivers and directions (negative weights make low driver values predictive of missingness).

`"MNAR"` as `"MAR"`, but the score is the target's *own* standardized values, so high values are more often missing (a `weights` entry named after the target flips or scales the direction). Numeric targets only.

With several target variables the scores are computed on the complete input, so each target's mechanism conditions on the pre-amputation values of its drivers.

Value

The amputed data, classed like the input (data.frame in, data.frame out; data.table in, data.table out), with attributes `"where"` (logical indicator matrix of the introduced missings), `"mechanism"` and `"prop"`.

Author(s)

Matthias Templ

See Also

`evaluation()`, `vimpute()`, `kNN()`

Examples

```
data(sleep)
complete_rows <- na.omit(sleep[, c("BodyWgt", "BrainWgt", "Sleep", "Span")])
amp <- makeMissing(complete_rows, prop = 0.2, mechanism = "MAR",
                   vars = "Sleep", seed = 1)
colSums(is.na(amp))
imp <- kNN(amp, imp_var = FALSE)
evaluation(complete_rows, imp, m = attr(amp, "where"))
```

---

mapMiss *Map with information about missing/imputed values*

Description

Map of observed and missing/imputed values.

Usage

```
mapMiss(
  x,
  coords,
  map,
  delimiter = NULL,
  selection = c("any", "all"),
  col = c("skyblue", "red", "orange"),
  alpha = NULL,
  pch = c(19, 15),
  col.map = grey(0.5),
  legend = TRUE,
  interactive = TRUE,
  ...
)
```

Arguments

x	a vector, matrix or data.frame.
coords	a data.frame or matrix with two columns giving the spatial coordinates of the observations.
map	a background map to be passed to bgmap().
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
selection	the selection method for displaying missing/imputed values in the map. Possible values are "any" (display missing/imputed values in *any* variable) and "all" (display missing/imputed values in *all* variables).
col	a vector of length three giving the colors to be used for observed, missing and imputed values. If a single color is supplied, it is used for all values.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
pch	a vector of length two giving the plot characters to be used for observed and missing/imputed values. If a single plot character is supplied, it will be used for both.
col.map	the color to be used for the background map.
legend	a logical indicating whether a legend should be plotted.
interactive	a logical indicating whether information about selected observations can be displayed interactively (see 'Details').
...	further graphical parameters to be passed to bgmap() and graphics::points().

Details

If `interactive=TRUE`, detailed information for an observation can be printed on the console by clicking on the corresponding point. Clicking in a region that does not contain any points quits the interactive session.

Author(s)

Matthias Templ, Andreas Alfons, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

`bgmap()`, `bubbleMiss()`, `colormapMiss()`

Examples

```
data(chorizonDL, package = "VIM")
data(kola.background, package = "VIM")
coo <- chorizonDL[, c("XCOO", "YCOO")]
## for missing values
x <- chorizonDL[, c("As", "Bi")]
mapMiss(x, coo, kola.background)

## for imputed values
x_imp <- kNN(chorizonDL[, c("As", "Bi")])
mapMiss(x_imp, coo, kola.background, delimiter = "_imp")
```

---

marginmatrix *Marginplot Matrix*

Description

Create a scatterplot matrix with information about missing/imputed values in the plot margins of each panel.

Usage

```
marginmatrix(
  x,
  delimiter = NULL,
  col = c("skyblue", "red", "red4", "orange", "orange4"),
  alpha = NULL,
  ...
)
```

Arguments

x	a matrix or data.frame.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
col	a vector of length five giving the colors to be used in the marginplots in the off-diagonal panels. The first color is used for the scatterplot and the boxplots for the available data, the second/fourth color for the univariate scatterplots and boxplots for the missing/imputed values in one variable, and the third/fifth color for the frequency of missing/imputed values in both variables (see 'Details'). If only one color is supplied, it is used for the bivariate and univariate scatterplots and the boxplots for missing/imputed values in one variable, whereas the boxplots for the available data are transparent. Else if two colors are supplied, the second one is recycled.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
...	further arguments and graphical parameters to be passed to pairsVIM() and marginplot(). par("oma") will be set appropriately unless supplied (see graphics::par()).

Details

marginmatrix uses pairsVIM() with a panel function based on marginplot().

The graphical parameter oma will be set unless supplied as an argument.

Author(s)

Andreas Alfons, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

marginplot(), pairsVIM(), scattmatrixMiss()

Other plotting functions: aggr(), barMiss(), histMiss(), marginplot(), matrixplot(), mosaicMiss(), pairsVIM(), parcoordMiss(), pbox(), scattJitt(), scattMiss(), scattmatrixMiss(), spineMiss()

Examples

```
data(sleep, package = "VIM")
## for missing values
x <- sleep[, 1:5]
```

```
x[,c(1,2,4)] <- log10(x[,c(1,2,4)])
marginmatrix(x)

## for imputed values
x_imp <- kNN(sleep[, 1:5])
x_imp[,c(1,2,4)] <- log10(x_imp[,c(1,2,4)])
marginmatrix(x_imp, delimiter = "_imp")
```

`marginplot` *Scatterplot with additional information in the margins*

Description

In addition to a standard scatterplot, information about missing/imputed values is shown in the plot margins. Furthermore, imputed values are highlighted in the scatterplot.

Usage

```
marginplot(
  x,
  delimiter = NULL,
  col = c("skyblue", "red", "red4", "orange", "orange4"),
  alpha = NULL,
  pch = c(1, 16),
  cex = par("cex"),
  numbers = TRUE,
  cex.numbers = par("cex"),
  zeros = FALSE,
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  ann = par("ann"),
  axes = TRUE,
  frame.plot = axes,
  ...
)
```

Arguments

`x`	a `matrix` or `data.frame` with two columns.
`delimiter`	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, `x` needs to have `colnames()`). If given, it is used to determine the corresponding imputation-index for any imputed variable (a

logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see `col`).

`col` a vector of length five giving the colors to be used in the plot. The first color is used for the scatterplot and the boxplots for the available data. In case of missing values, the second color is taken for the univariate scatterplots and boxplots for missing values in one variable and the third for the frequency of missing/imputed values in both variables (see 'Details'). Otherwise, in case of imputed values, the fourth color is used for the highlighting, the frequency, the univariate scatterplot and the boxplots of mputed values in the first variable and the fifth color for the same applied to the second variable. A black color is used for the highlighting and the frequency of imputed values in both variables instead. If only one color is supplied, it is used for the bivariate and univariate scatterplots and the boxplots for missing/imputed values in one variable, whereas the boxplots for the available data are transparent. Else if two colors are supplied, the second one is recycled.

`alpha` a numeric value between 0 and 1 giving the level of transparency of the colors, or `NULL`. This can be used to prevent overplotting.

`pch` a vector of length two giving the plot symbols to be used for the scatterplot and the univariate scatterplots. If a single plot character is supplied, it is used for the scatterplot and the default value will be used for the univariate scatterplots (see 'Details').

`cex` the character expansion factor to be used for the bivariate and univariate scatterplots.

`numbers` a logical indicating whether the frequencies of missing/imputed values should be displayed in the lower left of the plot (see 'Details').

`cex.numbers` the character expansion factor to be used for the frequencies of the missing/imputed values.

`zeros` a logical vector of length two indicating whether the variables are semi-continuous, i.e., contain a considerable amount of zeros. If `TRUE`, only the non-zero observations are used for drawing the respective boxplot. If a single logical is supplied, it is recycled.

`xlim`, `ylim` axis limits.

`main`, `sub` main and sub title.

`xlab`, `ylab` axis labels.

`ann` a logical indicating whether plot annotation (`main`, `sub`, `xlab`, `ylab`) should be displayed.

`axes` a logical indicating whether both axes should be drawn on the plot. Use graphical parameter `"xaxt"` or `"yaxt"` to suppress only one of the axes.

`frame.plot` a logical indicating whether a box should be drawn around the plot.

`...` further graphical parameters to be passed down (see `graphics::par()`).

Details

Boxplots for available and missing/imputed data, as well as univariate scatterplots for missing/imputed values in one variable are shown in the plot margins.

Imputed values in either of the variables are highlighted in the scatterplot.

Furthermore, the frequencies of the missing/imputed values can be displayed by a number (lower left of the plot). The number in the lower left corner is the number of observations that are missing/imputed in both variables.

Note

Some of the argument names and positions have changed with versions 1.3 and 1.4 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the argument `cex.text` can still be supplied to `...{}` and is handled correctly. Nevertheless, it is deprecated and no longer documented. Use `cex.numbers` instead.

Author(s)

Andreas Alfons, Matthias Templ, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

`scattMiss()`

Other plotting functions: `aggr()`, `barMiss()`, `histMiss()`, `marginmatrix()`, `matrixplot()`, `mosaicMiss()`, `pairsVIM()`, `parcoordMiss()`, `pbox()`, `scattJitt()`, `scattMiss()`, `scattmatrixMiss()`, `spineMiss()`

Examples

```
data(tao, package = "VIM")
data(chorizonDL, package = "VIM")
## for missing values
marginplot(tao[,c("Air.Temp", "Humidity")])
marginplot(log10(chorizonDL[,c("CaO", "Bi")]))

## for imputed values
marginplot(kNN(tao[,c("Air.Temp", "Humidity")]), delimiter = "_imp")
marginplot(kNN(log10(chorizonDL[,c("CaO", "Bi")])), delimiter = "_imp")
```

`matchImpute` *Fast matching/imputation based on categorical variable*

Description

Suitable donors are searched based on matching of the categorical variables. The variables are dropped in reversed order, so that the last element of 'match_var' is dropped first and the first element of the vector is dropped last.

Usage

```
matchImpute(
  data,
  variable = colnames(data)[!colnames(data) %in% match_var],
  match_var,
  imp_var = TRUE,
  imp_suffix = "imp"
)
```

Arguments

`data`	data.frame, data.table or matrix
`variable`	variables to be imputed
`match_var`	variables used for matching
`imp_var`	TRUE/FALSE if a TRUE/FALSE variables for each imputed variable should be created show the imputation status
`imp_suffix`	suffix for the TRUE/FALSE variables showing the imputation status

Details

The method works by sampling values from the suitable donors.

Value

the imputed data set.

Author(s)

Johannes Gussenbauer, Alexander Kowarik

See Also

`hotdeck()`

Other imputation methods: `hotdeck()`, `impPCA()`, `imputeCellEM()`, `imputeCellIRMI()`, `imputeCellM()`, `imputeCellMCD()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `irmi()`, `kNN()`, `medianSamp()`, `rangerImpute()`, `regressionImp()`, `sampleCat()`, `vimmi`, `vimpute()`, `xgboostImpute()`

Examples

```
data(sleep,package="VIM")
imp_data <- matchImpute(sleep,variable=c("NonD","Dream","Sleep","Span","Gest"),
  match_var=c("Exp","Danger"))

data(testdata,package="VIM")
imp_testdata1 <- matchImpute(testdata$wna,match_var=c("c1","c2","b1","b2"))

dt <- data.table::data.table(testdata$wna)
imp_testdata2 <- matchImpute(dt,match_var=c("c1","c2","b1","b2"))
```

---

matrixplot *Matrix plot*

Description

Create a matrix plot, in which all cells of a data matrix are visualized by rectangles. Available data is coded according to a continuous color scheme, while missing/imputed data is visualized by a clearly distinguishable color.

Usage

```
matrixplot(
  x,
  delimiter = NULL,
  sortby = NULL,
  col = c("red", "orange"),
  fixup = TRUE,
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  axes = TRUE,
  labels = axes,
  xpd = NULL,
  interactive = TRUE,
  ...
)
```

Arguments

x	a matrix or data.frame.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a

	logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see `col`).
`sortby`	a numeric or character value specifying the variable to sort the data matrix by, or `NULL` to plot without sorting.
`col`	the colors to be used in the plot. RGB colors may be specified as character strings or as objects of class "`colorspace::RGB()`". HCL colors need to be specified as objects of class "`colorspace::polarLUV()`". If only one color is supplied, it is used for missing and imputed data and a greyscale is used for available data. If two colors are supplied, the first is used for missing and the second for imputed data and a greyscale for available data. If three colors are supplied, the first is used as end color for the available data, while the start color is taken to be transparent for RGB or white for HCL. Missing/imputed data is visualized by the second/third color in this case. If four colors are supplied, the first is used as start color and the second as end color for the available data, while the third/fourth color is used for missing/imputed data.
`fixup`	a logical indicating whether the colors should be corrected to valid RGB values (see `colorspace::hex()`).
`xlim`, `ylim`	axis limits.
`main`, `sub`	main and sub title.
`xlab`, `ylab`	axis labels.
`axes`	a logical indicating whether axes should be drawn on the plot.
`labels`	either a logical indicating whether labels should be plotted below each column, or a character vector giving the labels.
`xpd`	a logical indicating whether the rectangles should be allowed to go outside the plot region. If `NULL`, it defaults to `TRUE` unless axis limits are specified.
`interactive`	a logical indicating whether a variable to be used for sorting can be selected interactively (see 'Details').
`...`	for `matrixplot` and `iimagMiss`, further graphical parameters to be passed to `graphics::plot.window()`, `graphics::title()` and `graphics::axis()`. For `TKRmatrixplot`, further arguments to be passed to `matrixplot`.

Details

In a *matrix plot*, all cells of a data matrix are visualized by rectangles. Available data is coded according to a continuous color scheme. To compute the colors via interpolation, the variables are first scaled to the interval between 0 and 1. Missing/imputed values can then be visualized by a clearly distinguishable color. It is thereby possible to use colors in the *HCL* or *RGB* color space. A simple way of visualizing the magnitude of the available data is to apply a greyscale, which has the advantage that missing/imputed values can easily be distinguished by using a color such as red/orange. Note that `-Inf` and `Inf` are always assigned the begin and end color, respectively, of the continuous color scheme.

Additionally, the observations can be sorted by the magnitude of a selected variable. If `interactive` is `TRUE`, clicking in a column redraws the plot with observations sorted by the corresponding variable. Clicking anywhere outside the plot region quits the interactive session.

Note

This is a much more powerful extension to the function `imagmiss` in the former CRAN package `dprep`.

`iimagMiss` is deprecated and may be omitted in future versions of `VIM`. Use `matrixplot` instead.

Author(s)

Andreas Alfons, Matthias Templ, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

Other plotting functions: `aggr()`, `barMiss()`, `histMiss()`, `marginmatrix()`, `marginplot()`, `mosaicMiss()`, `pairsVIM()`, `parcoordMiss()`, `pbox()`, `scattJitt()`, `scattMiss()`, `scattmatrixMiss()`, `spineMiss()`

Examples

```
data(sleep, package = "VIM")
## for missing values
x <- sleep[, -(8:10)]
x[,c(1,2,4,6,7)] <- log10(x[,c(1,2,4,6,7)])
matrixplot(x, sortby = "BrainWgt")

## for imputed values
x_imp <- kNN(sleep[, -(8:10)])
x_imp[,c(1,2,4,6,7)] <- log10(x_imp[,c(1,2,4,6,7)])
matrixplot(x_imp, delimiter = "_imp", sortby = "BrainWgt")
```

---

maxCat *Aggregation function for a factor variable*

Description

The function maxCat chooses the level with the most occurrences and random if the maximum is not unique.

Usage

```
maxCat(x, weights = NULL)
```

Arguments

x	factor vector
weights	numeric vector providing weights for the observations in x

---

medianSamp *Aggregation function for a ordinal variable*

Description

The function medianSamp chooses the level as the median or randomly between two levels.

Usage

```
medianSamp(x, weights = NULL)
```

Arguments

x	ordered factor vector
weights	numeric vector providing weights for the observations in x

See Also

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute(), xgboostImpute()

---

mosaicMiss *Mosaic plot with information about missing/imputed values*

Description

Create a mosaic plot with information about missing/imputed values.

Usage

```
mosaicMiss(
  x,
  delimiter = NULL,
  highlight = NULL,
  selection = c("any", "all"),
  plotvars = NULL,
  col = c("skyblue", "red", "orange"),
  labels = NULL,
  miss.labels = TRUE,
  ...
)
```

Arguments

x	a matrix or data.frame.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
highlight	a vector giving the variables to be used for highlighting. If NULL (the default), all variables are used for highlighting.
selection	the selection method for highlighting missing/imputed values in multiple highlight variables. Possible values are "any" (highlighting of missing/imputed values in *any* of the highlight variables) and "all" (highlighting of missing/imputed values in *all* of the highlight variables).
plotvars	a vector giving the categorical variables to be plotted. If NULL (the default), all variables are plotted.
col	a vector of length three giving the colors to be used for observed, missing and imputed data. If only one color is supplied, the tiles corresponding to observed data are transparent and the supplied color is used for highlighting.
labels	a list of arguments for the labeling function vcd::labeling_border().
miss.labels	either a logical indicating whether labels should be plotted for observed and missing/imputed (highlighted) data, or a character vector giving the labels.
...	additional arguments to be passed to vcd::mosaic().

Details

Mosaic plots are graphical representations of multi-way contingency tables. The frequencies of the different cells are visualized by area-proportional rectangles (tiles). Additional tiles are be used to display the frequencies of missing/imputed values. Furthermore, missing/imputed values in a certain variable or combination of variables can be highlighted in order to explore their structure.

Value

An object of class "structable" is returned invisibly.

Note

This function uses the highly flexible strucplot framework of package vcd.

Author(s)

Andreas Alfons, modifications by Bernd Prantner

References

Meyer, D., Zeileis, A. and Hornik, K. (2006) The `strucplot` framework: Visualizing multi-way contingency tables with **vcd**. *Journal of Statistical Software*, **17 (3)**, 1–48.

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

`spineMiss()`, `vcd::mosaic()`

Other plotting functions: `aggr()`, `barMiss()`, `histMiss()`, `marginmatrix()`, `marginplot()`, `matrixplot()`, `pairsVIM()`, `parcoordMiss()`, `pbox()`, `scattJitt()`, `scattMiss()`, `scattmatrixMiss()`, `spineMiss()`

Examples

```
data(sleep, package = "VIM")
## for missing values
mosaicMiss(sleep, highlight = 4,
    plotvars = 8:10, miss.labels = FALSE)

## for imputed values
mosaicMiss(kNN(sleep), highlight = 4,
    plotvars = 8:10, delimiter = "_imp", miss.labels = FALSE)
```

`overimpute` *Overimputation: calibration diagnostic for an imputation model*

Description

Answers "is my imputation model well calibrated?" without ground truth (the analogue of `Amelia::overimpute()`, model-agnostic): the *observed* cells of one variable are treated as missing – fold by fold, so every observed cell is overimputed exactly once – and imputed with `draws` multiple imputations via `vimpute()`. For each cell the observed value is compared with the mean and a `level` interval of its draws; a well-calibrated model covers roughly `level` of the observed values. Any genuinely missing cells stay missing and are imputed as part of the model, exactly as they would be in a real run.

Usage

```
overimpute(data, variable, ..., draws = 5, folds = 5, level = 0.9, seed = NULL)

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

## S3 method for class 'vimpute_overimpute'
plot(x, ...)
```

Arguments

data	Dataset with (possibly) missing values.
variable	Single column name: the numeric variable whose observed cells are overimputed.
...	Passed on to vimpute() (e.g. method, spec, uncert, sequential). m is controlled via draws and cannot be supplied.
draws	Number of multiple-imputation draws per cell (>= 2).
folds	Number of folds the observed cells are split into (each fold is set missing and overimputed in one vimpute() run; capped at the number of observed cells).
level	Coverage level of the reported interval (default 0.9).
seed	Optional single number: applied via set.seed() before the fold split and the imputation runs.
x	A vimpute_overimpute object

Details

The ... arguments are passed on to vimpute(), so the diagnostic runs for any method, spec, or grammar configuration – e.g. overimpute(dat, "y", method = "robust") or overimpute(dat, "y", spec = list(y = vs_ranger(num.trees = 300))). Note the cost: folds x draws imputation runs; lower folds and draws (the defaults are modest) or use a fast method for large data. For a stochastic spread of the draws keep an uncertainty source switched on (the default uncert = "pmm", or boot/uncert); with purely deterministic settings all draws coincide and the intervals collapse.

Value

A data.frame of class vimpute_overimpute with one row per observed cell: row (row index in data), observed, mean, lower, upper (quantiles of the draws at level), and covered. Attributes: variable, level, draws, folds. print() reports the empirical coverage; plot() draws the observed-vs-imputed calibration plot with intervals and the 45-degree line.

Author(s)

Matthias Templ

See Also

vimpute(), makeMissing(), evaluation()

Examples

```
## Not run:
data(sleep)
ov <- overimpute(sleep, "Sleep", method = "robust", sequential = FALSE,
                 draws = 5, folds = 5, seed = 1)
print(ov)   # empirical coverage of the 90% intervals
plot(ov)
```

```
## End(Not run)
```

---

pairsVIM *Scatterplot Matrices*

Description

Create a scatterplot matrix.

Usage

```
pairsVIM(
  x,
  ...,
  delimiter = NULL,
  main = NULL,
  sub = NULL,
  panel = points,
  lower = panel,
  upper = panel,
  diagonal = NULL,
  labels = TRUE,
  pos.labels = NULL,
  cex.labels = NULL,
  font.labels = par("font"),
  layout = c("matrix", "graph"),
  gap = 1
)
```

Arguments

x	a matrix or data.frame.
...	further arguments and graphical parameters to be passed down. par("oma") will be set appropriately unless supplied (see graphics::par()).
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
main, sub	main and sub title.
panel	a function(x, y, ...{}), which is used to plot the contents of each off-diagonal panel of the display.
lower, upper	separate panel functions to be used below and above the diagonal, respectively.

diagonal	optional function(x, ...{}) to be applied on the diagonal panels.
labels	either a logical indicating whether labels should be plotted in the diagonal panels, or a character vector giving the labels.
pos.labels	the vertical position of the labels in the diagonal panels.
cex.labels	the character expansion factor to be used for the labels.
font.labels	the font to be used for the labels.
layout	a character string giving the layout of the scatterplot matrix. Possible values are "matrix" (a matrix-like layout with the first row on top) and "graph" (a graph-like layout with the first row at the bottom).
gap	a numeric value giving the distance between the panels in margin lines.

Details

This function is the workhorse for marginmatrix() and scattmatrixMiss().

The graphical parameter oma will be set unless supplied as an argument.

A panel function should not attempt to start a new plot, since the coordinate system for each panel is set up by pairsVIM.

Note

The code is based on graphics::pairs(). Starting with version 1.4, infinite values are no longer removed before passing the x and y vectors to the panel functions.

Author(s)

Andreas Alfons, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

marginmatrix(), scattmatrixMiss()

Other plotting functions: aggr(), barMiss(), histMiss(), marginmatrix(), marginplot(), matrixplot(), mosaicMiss(), parcoordMiss(), pbox(), scattJitt(), scattMiss(), scattmatrixMiss(), spineMiss()

Examples

```
data(sleep, package = "VIM")
x <- sleep[, -(8:10)]
x[,c(1,2,4,6,7)] <- log10(x[,c(1,2,4,6,7)])
pairsVIM(x)
```

parcoordMiss *Parallel coordinate plot with information about missing/imputed values*

Description

Parallel coordinate plot with adjustments for missing/imputed values. Missing values in the plotted variables may be represented by a point above the corresponding coordinate axis to prevent disconnected lines. In addition, observations with missing/imputed values in selected variables may be highlighted.

Usage

```
parcoordMiss(
  x,
  delimiter = NULL,
  highlight = NULL,
  selection = c("any", "all"),
  plotvars = NULL,
  plotNA = TRUE,
  col = c("skyblue", "red", "skyblue4", "red4", "orange", "orange4"),
  alpha = NULL,
  lty = par("lty"),
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  labels = TRUE,
  xpd = NULL,
  interactive = TRUE,
  ...
)
```

Arguments

x	a matrix or data.frame.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
highlight	a vector giving the variables to be used for highlighting. If NULL (the default), all variables are used for highlighting.

selection	the selection method for highlighting missing/imputed values in multiple highlight variables. Possible values are "any" (highlighting of missing/imputed values in *any* of the highlight variables) and "all" (highlighting of missing/imputed values in *all* of the highlight variables).
plotvars	a vector giving the variables to be plotted. If NULL (the default), all variables are plotted.
plotNA	a logical indicating whether missing values in the plot variables should be represented by a point above the corresponding coordinate axis to prevent disconnected lines.
col	if plotNA is TRUE, a vector of length six giving the colors to be used for observations with different combinations of observed and missing/imputed values in the plot variables and highlight variables (vectors of length one or two are recycled). Otherwise, a vector of length two giving the colors for non-highlighted and highlighted observations (if a single color is supplied, it is used for both).
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
lty	if plotNA is TRUE, a vector of length four giving the line types to be used for observations with different combinations of observed and missing/imputed values in the plot variables and highlight variables (vectors of length one or two are recycled). Otherwise, a vector of length two giving the line types for non-highlighted and highlighted observations (if a single line type is supplied, it is used for both).
xlim, ylim	axis limits.
main, sub	main and sub title.
xlab, ylab	axis labels.
labels	either a logical indicating whether labels should be plotted below each coordinate axis, or a character vector giving the labels.
xpd	a logical indicating whether the lines should be allowed to go outside the plot region. If NULL, it defaults to TRUE unless axis limits are specified.
interactive	a logical indicating whether interactive features should be enabled (see 'Details').
...	for parcoordMiss, further graphical parameters to be passed down (see graphics::par()). For TKRparcoordMiss, further arguments to be passed to parcoordMiss.

Details

In parallel coordinate plots, the variables are represented by parallel axes. Each observation of the scaled data is shown as a line. Observations with missing/imputed values in selected variables may thereby be highlighted. However, plotting variables with missing values results in disconnected lines, making it impossible to trace the respective observations across the graph. As a remedy, missing values may be represented by a point above the corresponding coordinate axis, which is separated from the main plot by a small gap and a horizontal line, as determined by plotNA. Connected lines can then be drawn for all observations. Nevertheless, a caveat of this display is that it may draw attention away from the main relationships between the variables.

If `interactive` is `TRUE`, it is possible switch between this display and the standard display without the separate level for missing values by clicking in the top margin of the plot. In addition, the variables to be used for highlighting can be selected interactively. Observations with missing/imputed values in any or in all of the selected variables are highlighted (as determined by `selection`). A variable can be added to the selection by clicking on a coordinate axis. If a variable is already selected, clicking on its coordinate axis removes it from the selection. Clicking anywhere outside the plot region (except the top margin, if missing/imputed values exist) quits the interactive session.

Note

Some of the argument names and positions have changed with versions 1.3 and 1.4 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `colcomb` and `xaxlabels` can still be supplied to `...{}` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `highlight` and `labels` instead.

Author(s)

Andreas Alfons, Matthias Templ, modifications by Bernd Prantner

References

Wegman, E. J. (1990) Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association* **85 (411)**, 664–675.

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

`pbox()`

Other plotting functions: `aggr()`, `barMiss()`, `histMiss()`, `marginmatrix()`, `marginplot()`, `matrixplot()`, `mosaicMiss()`, `pairsVIM()`, `pbox()`, `scattJitt()`, `scattMiss()`, `scattmatrixMiss()`, `spineMiss()`

Examples

```
data(chorizonDL, package = "VIM")
## for missing values
parcoordMiss(chorizonDL[,c(15,101:110)],
    plotvars=2:11, interactive = FALSE)
legend("top", col = c("skyblue", "red"), lwd = c(1,1),
    legend = c("observed in Bi", "missing in Bi"))

## for imputed values
parcoordMiss(kNN(chorizonDL[,c(15,101:110)]), delimiter = "_imp" ,
    plotvars=2:11, interactive = FALSE)
legend("top", col = c("skyblue", "orange"), lwd = c(1,1),
    legend = c("observed in Bi", "imputed in Bi"))
```

pbox *Parallel boxplots with information about missing/imputed values*

Description

Boxplot of one variable of interest plus information about missing/imputed values in other variables.

Usage

```
pbox(
  x,
  delimiter = NULL,
  pos = 1,
  selection = c("none", "any", "all"),
  col = c("skyblue", "red", "red4", "orange", "orange4"),
  numbers = TRUE,
  cex.numbers = par("cex"),
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  axes = TRUE,
  frame.plot = axes,
  labels = axes,
  interactive = TRUE,
  ...
)
```

Arguments

x	a vector, matrix or data.frame.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
pos	a numeric value giving the index of the variable of interest. Additional variables in x are used for grouping according to missingness/number of imputed missings.
selection	the selection method for grouping according to missingness/number of imputed missings in multiple additional variables. Possible values are "none" (grouping according to missingness/number of imputed missings in every other variable

that contains missing/imputed values), `"any"` (grouping according to missingness/number of imputed missings in *any* of the additional variables) and `"all"` (grouping according to missingness/number of imputed missings in *all* of the additional variables).

`col` a vector of length five giving the colors to be used in the plot. The first color is used for the boxplots of the available data, the second/fourth are used for missing/imputed data, respectively, and the third/fifth color for the frequencies of missing/imputed values in both variables (see 'Details'). If only one color is supplied, it is used for the boxplots for missing/imputed data, whereas the boxplots for the available data are transparent. Else if two colors are supplied, the second one is recycled.

`numbers` a logical indicating whether the frequencies of missing/imputed values should be displayed (see 'Details').

`cex.numbers` the character expansion factor to be used for the frequencies of the missing/imputed values.

`xlim`, `ylim` axis limits.

`main`, `sub` main and sub title.

`xlab`, `ylab` axis labels.

`axes` a logical indicating whether axes should be drawn on the plot.

`frame.plot` a logical indicating whether a box should be drawn around the plot.

`labels` either a logical indicating whether labels should be plotted below each box, or a character vector giving the labels.

`interactive` a logical indicating whether variables can be switched interactively (see 'Details').

`...` for `pbox`, further arguments and graphical parameters to be passed to `graphics::boxplot()` and other functions. For `TKRpbox`, further arguments to be passed to `pbox`.

Details

This plot consists of several boxplots. First, a standard boxplot of the variable of interest is produced. Second, boxplots grouped by observed and missing/imputed values according to `selection` are produced for the variable of interest.

Additionally, the frequencies of the missing/imputed values can be represented by numbers. If so, the first line corresponds to the observed values of the variable of interest and their distribution in the different groups, the second line to the missing/imputed values.

If `interactive=TRUE`, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session.

Value

a list as returned by `graphics::boxplot()`.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in VIM. For back compatibility, the arguments names and cex.text can still be supplied to ...{} and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use labels and cex.numbers instead.

Author(s)

Andreas Alfons, Matthias Templ, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

parcoordMiss()

Other plotting functions: aggr(), barMiss(), histMiss(), marginmatrix(), marginplot(), matrixplot(), mosaicMiss(), pairsVIM(), parcoordMiss(), scattJitt(), scattMiss(), scattmatrixMiss(), spineMiss()

Examples

```
data(chorizonDL, package = "VIM")
## for missing values
pbox(log(chorizonDL[, c(4,5,8,10,11,16:17,19,25,29,37,38,40)]))

## for imputed values
pbox(kNN(log(chorizonDL[, c(4,8,10,11,17,19,25,29,37,38,40)])),
     delimiter = "_imp")
```

---

plot.vimmi *Diagnostic plots for a vimmi object*

Description

Draws mice-style multiple-imputation diagnostics for a vimmi object, for every numeric variable with imputed cells:

"chains" convergence trace plots of the sequential (FCS) imputation: the mean and the standard deviation of the imputed values against the iteration number, one line per imputation. Chains that mix well fluctuate around a common level without trends; systematic drift suggests increasing nseq. Requires chain statistics (stored by vimpute() since VIM 7.3.0).

"density" the density of the observed values (blue, bold) overlaid with the density of each imputation's imputed values (red, thin) – the analogue of mice::densityplot(). Imputed densities that deviate wildly from the observed one can flag model misfit (or genuine MAR shifts).

`"strip"` every value as a point: column 0 holds the observed values (blue), columns 1..m the imputed values of each imputation (red) – the analogue of `mice::stripplot()`.

Usage

```
## S3 method for class 'vimmi'
plot(x, y = c("chains", "density", "strip"), ...)
```

Arguments

`x`	A `vimmi` object created by `vimpute()` with `m > 1`
`y`	Type of diagnostic: `"chains"` (default), `"density"`, or `"strip"`
`...`	Passed on to the underlying base-graphics calls

Value

`x`, invisibly

See Also

`vimmi`, `vimpute`

Examples

```
## Not run:
result <- vimpute(sleep, method = "ranger", m = 5, seed = 1)
plot(result)             # convergence chains
plot(result, "density")  # observed vs imputed densities
plot(result, "strip")    # observed vs imputed values

## End(Not run)
```

---

prepare *Transformation and standardization*

Description

This function is used by the `VIM` GUI for transformation and standardization of the data.

Usage

```
prepare(
  x,
  scaling = c("none", "classical", "MCD", "robust", "onestep"),
 transformation = c("none", "minus", "reciprocal", "logarithm", "exponential", "boxcox",
    "clr", "ilr", "alr"),
  alpha = NULL,
  powers = NULL,
```

```
  start = 0,
  alrVar
)
```

Arguments

x	a vector, matrix or data.frame.
scaling	the scaling to be applied to the data. Possible values are "none", "classical", MCD, "robust" and "onestep".
transformation	the transformation of the data. Possible values are "none", "minus", "reciprocal", "logarithm", "exponential", "boxcox", "clr", "ilr" and "alr".
alpha	a numeric parameter controlling the size of the subset for the *MCD* (if scaling="MCD"). See robustbase::covMcd().
powers	a numeric vector giving the powers to be used in the Box-Cox transformation (if transformation="boxcox"). If NULL, the powers are calculated with function car::powerTransform().
start	a constant to be added prior to Box-Cox transformation (if transformation="boxcox").
alrVar	variable to be used as denominator in the additive logratio transformation (if transformation="alr").

Details

Transformation:

"none": no transformation is used.

"logarithm": compute the the logarithm (to the base 10).

"boxcox": apply a Box-Cox transformation. Powers may be specified or calculated with the function car::powerTransform().

Standardization:

"none": no standardization is used.

"classical": apply a z-Transformation on each variable by using function scale().

"robust": apply a robustified z-Transformation by using median and MAD.

Value

Transformed and standardized data.

Author(s)

Matthias Templ, modifications by Andreas Alfons

See Also

scale(), car::powerTransform()

Examples

```
data(sleep, package = "VIM")
x <- sleep[, c("BodyWgt", "BrainWgt")]
prepare(x, scaling = "robust", transformation = "logarithm")
```

`pulplignin` *Pulp lignin content*

Description

Pulp quality by lignin content remaining

Format

A data frame with 301 observations on the following 23 variables.

Details

Pulp quality is measured by the lignin content remaining in the pulp: the Kappa number. This data set is used to understand which variables in the process influence the Kappa number, and if it can be predicted accurately enough for an inferential sensor application. Variables with a number at the end have been lagged by that number of hours to line up the data.

Source

https://openmv.net/info/kamyr-digester

References

K. Walkush and R.R. Gustafson. Application of feedforward neural networks and partial least squares regression for modelling Kappa number in a continuous Kamyr digester", Pulp and Paper Canada, 95, 1994, p T7-T13.

Examples

```
data(pulplignin)
str(pulplignin)
aggr(pulplignin)
```

rangerImpute *Random Forest Imputation*

Description

Impute missing values based on random-forest models via `vimpute()`.

Usage

```
rangerImpute(
  formula,
  data,
  imp_var = TRUE,
  imp_suffix = "imp",
  ...,
  verbose = FALSE,
  median = FALSE
)
```

Arguments

`formula`	model formula for the imputation
`data`	A `data.frame` containing the data
`imp_var`	`TRUE`/`FALSE` if a `TRUE`/`FALSE` variables for each imputed variable should be created show the imputation status
`imp_suffix`	suffix used for TF imputation variables
`...`	Additional ranger hyperparameters (e.g. `num.trees`, `mtry`, `min.node.size`), forwarded to the ranger learner via `vimpute()`.
`verbose`	Show the number of observations used for training and evaluating the RF-Model.
`median`	`TRUE`/`FALSE`. If `TRUE`, ranger regression predictions are aggregated tree-wise using the median (via `vimpute()`).

Value

the imputed data set.

See Also

Other imputation methods: `hotdeck()`, `impPCA()`, `imputeCellEM()`, `imputeCellIRMI()`, `imputeCellM()`, `imputeCellMCD()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `irmi()`, `kNN()`, `matchImpute()`, `medianSamp()`, `regressionImp()`, `sampleCat()`, `vimmi`, `vimpute()`, `xgboostImpute()`

Examples

```
data(sleep)
# rangerImpute() runs on vimpute(), which VIM backs with the suggested
# mlr3 stack -- skip the example where that stack is not installed.
if (requireNamespace("mlr3", quietly = TRUE) &&
    requireNamespace("mlr3learners", quietly = TRUE) &&
    requireNamespace("mlr3pipelines", quietly = TRUE)) {
  rangerImpute(Dream+NonD~BodyWgt+BrainWgt,data=sleep)
}
```

---

register_vimpute_method

Register an imputation method for `vimpute()`

Description

`vimpute()` resolves its `method` argument through a package-level method registry. The built-in methods (`"ranger"`, `"xgboost"`, `"regularized"`, `"robust"`, `"gam"`, `"robgam"`, `"restricted"`) are pre-registered; this function adds (or, with `overwrite = TRUE`, replaces) a user-defined method backed by any pair of mlr3 learners – e.g. `regr.rpart`/`classif.rpart` from mlr3 itself, or learners from `mlr3extralearners` such as lightgbm – without modifying VIM. After registration the new name can be used anywhere the built-in method names work: as a global `method`, in a per-variable method list, and in method-keyed `learner_params`.

Usage

```
register_vimpute_method(
  name,
  learner,
  packages = character(),
  setup = NULL,
  defaults = NULL,
  search_space = NULL,
  supports_formula = FALSE,
  fallback = "robust",
  validate = NULL,
  overwrite = FALSE
)
```

Arguments

`name`	Single character string: the method name to be used in `vimpute(method = )`. Must not collide with a registered method unless `overwrite = TRUE`. Built-in methods cannot be replaced or removed.
`learner`	Named list with elements `regr` and/or `classif`, each a character vector of mlr3 learner ids (candidates in preference order; the first is the default, multiple candidates are compared by cross-validation like the built-in `"regularized"`

method). Methods registered with only a `regr` (or only a `classif`) learner fall back to `fallback` for target variables of the other type, with a warning.

`packages` Character vector of packages that must be installed when the method is used (checked with `requireNamespace()` at `vimpute()` call time, not at registration).

`setup` `NULL` or a function with no arguments, called once per `vimpute()` call before the method's learners are constructed. Use it to register custom mlr3 learners or load learner collections (e.g. `function() library(mlr3extralearners)`).

`defaults` `NULL`, a named list of learner parameter values, or a function `function(task_type, nthread)` returning such a list (`task_type` is `"regr"` or `"classif"`, `nthread` the thread count vimpute chose for the data size). User-supplied `learner_params` override these defaults.

`search_space` `NULL` or a function `function(learner_id, task)` returning `list(space = paradox::ps(...), n_eva` consulted when `tune = TRUE`. Without it, tuning is skipped for the method with a warning (as for unknown learners).

`supports_formula`
Logical: can the method be used with the `formula` argument of `vimpute()`? Formula-based imputation requires a learner that models from a design matrix; the built-ins with formula support are `"robust"`, `"regularized"`, `"gam"`, `"robgam"`, and `"restricted"`.

`fallback` Single method name used when `validate` rejects a variable or a target type has no learner. Defaults to `"robust"`.

`validate` `NULL` or a function `function(y_obs, data, variable)` called during prechecking for every variable the method is assigned to (`y_obs`: the observed values of the target; `data`: the full dataset; `variable`: the target's name). Return `NULL` to accept, a character string (the warning message) to reject towards `fallback`, or `list(reason = , fallback = )` to reject towards a specific method. Fallbacks are validated in turn until a method accepts.

`overwrite` Logical: replace an existing registration of the same name? Built-in methods can never be replaced.

Details

Uncertainty handling for registered methods: PMM (`uncert = "pmm"`, the default, and `pmm = TRUE`) and `uncert = "midastouch"` work with any method because they only use the method's predictions. `uncert = "normalerror"`/`"resid"` and `boot = TRUE` derive residuals from training predictions when the model object does not expose them.

Value

Invisibly, the registered method name.

See Also

`vimpute_methods()`, `unregister_vimpute_method()`, `vimpute()`

Other vimpute method registry: `unregister_vimpute_method()`, `vimpute_methods()`

Examples

```
# a CART method backed by mlr3's rpart learners -- one call, no VIM patching
register_vimpute_method("cart",
  learner  = list(regr = "regr.rpart", classif = "classif.rpart"),
  packages = "rpart")
"cart" %in% vimpute_methods()

data(sleep)
res <- vimpute(sleep[, c("Sleep", "Dream", "Span")], method = "cart",
               sequential = FALSE)

unregister_vimpute_method("cart")
```

---

regressionImp *Regression Imputation (via vimpute)*

Description

Impute missing values based on a regression / classification model.

Usage

```
regressionImp(
  formula,
  data,
  family = "AUTO",
  robust = FALSE,
  imp_var = TRUE,
  imp_suffix = "imp",
  mod_cat = FALSE
)
```

Arguments

formula	model formula to impute one variable
data	A data.frame containing the data
family	ignored in this wrapper version (only "AUTO"-like behaviour via vimpute)
robust	logical; if TRUE use method = "robust", otherwise "regularized"
imp_var	logical; if TRUE create TRUE/FALSE variables indicating imputation status
imp_suffix	suffix used for TF imputation variables
mod_cat	ignored in this wrapper version (classification handled by vimpute)

Details

By default (lm() for numeric responses and glm() for binary responses, with any number of predictors) the imputation is deterministic and matches a plain regression fit. When the design is rank-deficient (collinear predictors or $p \geq n$) or the response is a multi-level factor, it falls back to the regularized (glmnet) path via vimpute() with method = "regularized". If robust = TRUE, vimpute() with method = "robust" (lmrob() / glmrob()) is used.

Value

the imputed data set (same class as data)

See Also

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), sampleCat(), vimmi, vimpute(), xgboostImpute()

---

rugNA *Rug representation of missing/imputed values*

Description

Add a rug representation of missing/imputed values in only one of the variables to scatterplots.

Usage

```
rugNA(
  x,
  y,
  ticksize = NULL,
  side = 1,
  col = "red",
  alpha = NULL,
  miss = NULL,
  lwd = 0.5,
  ...
)
```

Arguments

x, y	numeric vectors.
ticksize	the length of the ticks. Positive lengths give inward ticks.
side	an integer giving the side of the plot to draw the rug representation.
col	the color to be used for the ticks.
alpha	the alpha value (between 0 and 1).

miss	a data.frame or matrix with two columns and logical values. If NULL, x and y are searched for missing values, otherwise, the first column of miss is used to determine the imputed values in x and the second one for the imputed values in y.
lwd	the line width to be used for the ticks.
...	further arguments to be passed to graphics::Axis().

Details

If side is 1 or 3, the rug representation consists of values available in x but missing/imputed in y. Else if side is 2 or 4, it consists of values available in y but missing/imputed in x.

Author(s)

Andreas Alfons, modifications by Bernd Prantner

Examples

```
data(tao, package = "VIM")
## for missing values
x <- tao[, "Air.Temp"]
y <- tao[, "Humidity"]
plot(x, y)
rugNA(x, y, side = 1)
rugNA(x, y, side = 2)

## for imputed values
x_imp <- kNN(tao[, c("Air.Temp","Humidity")])
x <- x_imp[, "Air.Temp"]
y <- x_imp[, "Humidity"]
miss <- x_imp[, c("Air.Temp_imp","Humidity_imp")]
plot(x, y)
rugNA(x, y, side = 1, col = "orange", miss = miss)
rugNA(x, y, side = 2, col = "orange", miss = miss)
```

---

sampleCat *Random aggregation function for a factor variable*

Description

The function sampleCat samples with probabilites corresponding to the occurrence of the level in the NNs.

Usage

```
sampleCat(x, weights = NULL)
```

Arguments

x	factor vector
weights	numeric vector providing weights for the observations in x

See Also

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), vimmi, vimpute(), xgboostImpute()

SBS5242 *Synthetic subset of the Austrian structural business statistics data*

Description

Synthetic subset of the Austrian structural business statistics (SBS) data, namely NACE code 52.42 (retail sale of clothing).

Details

The Austrian SBS data set consists of more than 320.000 enterprises. Available raw (unedited) data set: 21669 observations in 90 variables, structured according NACE revision 1.1 with 3891 missing values.

We investigate 9 variables of NACE 52.42 (retail sale of clothing).

From these confidential raw data set a non-confidential, close-to-reality, synthetic data set was generated.

Source

http://www.statistik.at

Examples

```
data(SBS5242)
aggr(SBS5242)
```

scattJitt *Bivariate jitter plot*

Description

Create a bivariate jitter plot.

Usage

```
scattJitt(
  x,
  delimiter = NULL,
  col = c("skyblue", "red", "red4", "orange", "orange4"),
  alpha = NULL,
  cex = par("cex"),
  col.line = "lightgrey",
  lty = "dashed",
  lwd = par("lwd"),
  numbers = TRUE,
  cex.numbers = par("cex"),
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  axes = TRUE,
  frame.plot = axes,
  labels = c("observed", "missing", "imputed"),
  ...
)
```

Arguments

x	a data.frame or matrix with two columns.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
col	a vector of length five giving the colors to be used in the plot. The first color will be used for complete observations, the second/fourth color for missing/imputed values in only one variable, and the third/fifth color for missing/imputed values in both variables. If only one color is supplied, it is used for all. Else if two colors are supplied, the second one is recycled.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.

cex	the character expansion factor for the plot characters.
col.line	the color for the lines dividing the plot region.
lty	the line type for the lines dividing the plot region (see graphics::par()).
lwd	the line width for the lines dividing the plot region.
numbers	a logical indicating whether the frequencies of observed and missing/imputed values should be displayed (see 'Details').
cex.numbers	the character expansion factor to be used for the frequencies of the observed and missing/imputed values.
main, sub	main and sub title.
xlab, ylab	axis labels.
axes	a logical indicating whether both axes should be drawn on the plot. Use graphical parameter "xaxt" or "yaxt" to suppress just one of the axes.
frame.plot	a logical indicating whether a box should be drawn around the plot.
labels	a vector of length three giving the axis labels for the regions for observed, missing and imputed values (see 'Details').
...	further graphical parameters to be passed down (see graphics::par()).

Details

The amount of observed and missing/imputed values is visualized by jittered points. Thereby the plot region is divided into up to four regions according to the existence of missing/imputed values in one or both variables. In addition, the amount of observed and missing/imputed values can be represented by a number.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in VIM. For back compatibility, the argument cex.text can still be supplied to ...{} and is handled correctly. Nevertheless, it is deprecated and no longer documented. Use cex.numbers instead.

Author(s)

Matthias Templ, modifications by Andreas Alfons and Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

Other plotting functions: aggr(), barMiss(), histMiss(), marginmatrix(), marginplot(), matrixplot(), mosaicMiss(), pairsVIM(), parcoordMiss(), pbox(), scattMiss(), scattmatrixMiss(), spineMiss()

Examples

```
data(tao, package = "VIM")
## for missing values
scattJitt(tao[, c("Air.Temp", "Humidity")])

## for imputed values
scattJitt(kNN(tao[, c("Air.Temp", "Humidity")]), delimiter = "_imp")
```

`scattmatrixMiss` *Scatterplot matrix with information about missing/imputed values*

Description

Scatterplot matrix in which observations with missing/imputed values in certain variables are highlighted.

Usage

```
scattmatrixMiss(
  x,
  delimiter = NULL,
  highlight = NULL,
  selection = c("any", "all"),
  plotvars = NULL,
  col = c("skyblue", "red", "orange"),
  alpha = NULL,
  pch = c(1, 3),
  lty = par("lty"),
  diagonal = c("density", "none"),
  interactive = TRUE,
  ...
)
```

Arguments

`x`	a matrix or `data.frame`.
`delimiter`	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, `x` needs to have `colnames()`). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see `col`).
`highlight`	a vector giving the variables to be used for highlighting. If `NULL` (the default), all variables are used for highlighting.

selection	the selection method for highlighting missing/imputed values in multiple highlight variables. Possible values are "any" (highlighting of missing/imputed values in *any* of the highlight variables) and "all" (highlighting of missing/imputed values in *all* of the highlight variables).
plotvars	a vector giving the variables to be plotted. If NULL (the default), all variables are plotted.
col	a vector of length three giving the colors to be used in the plot. The second/third color will be used for highlighting missing/imputed values.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
pch	a vector of length two giving the plot characters. The second plot character will be used for the highlighted observations.
lty	a vector of length two giving the line types for the density plots in the diagonal panels (if diagonal="density"). The second line type is used for the highlighted observations. If a single value is supplied, it is used for both non-highlighted and highlighted observations.
diagonal	a character string specifying the plot to be drawn in the diagonal panels. Possible values are "density" (density plots for non-highlighted and highlighted observations) and "none".
interactive	a logical indicating whether the variables to be used for highlighting can be selected interactively (see 'Details').
...	for scattmatrixMiss, further arguments and graphical parameters to be passed to pairsVIM(). par("oma") will be set appropriately unless supplied (see graphics::par()). For TKRscattmatrixMiss, further arguments to be passed to scattmatrixMiss.

Details

scattmatrixMiss uses pairsVIM() with a panel function that allows highlighting of missing/imputed values.

If interactive=TRUE, the variables to be used for highlighting can be selected interactively. Observations with missing/imputed values in any or in all of the selected variables are highlighted (as determined by selection). A variable can be added to the selection by clicking in a diagonal panel. If a variable is already selected, clicking on the corresponding diagonal panel removes it from the selection. Clicking anywhere else quits the interactive session.

The graphical parameter oma will be set unless supplied as an argument.

TKRscattmatrixMiss behaves like scattmatrixMiss, but uses tkrplot to embed the plot in a *Tcl/Tk* window. This is useful if the number of variables is large, because scrollbars allow to move from one part of the plot to another.

Note

Some of the argument names and positions have changed with version 1.3 due to a re-implementation and for more consistency with other plot functions in VIM. For back compatibility, the argument colcomb can still be supplied to ...{} and is handled correctly. Nevertheless, it is deprecated and no longer documented. Use highlight instead. The arguments smooth, reg.line and legend.plot are no longer used and ignored if supplied.

Author(s)

Andreas Alfons, Matthias Templ, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

pairsVIM(), marginmatrix()

Other plotting functions: aggr(), barMiss(), histMiss(), marginmatrix(), marginplot(), matrixplot(), mosaicMiss(), pairsVIM(), parcoordMiss(), pbox(), scattJitt(), scattMiss(), spineMiss()

Examples

```
data(sleep, package = "VIM")
## for missing values
x <- sleep[, 1:5]
x[,c(1,2,4)] <- log10(x[,c(1,2,4)])
scattmatrixMiss(x, highlight = "Dream")

## for imputed values
x_imp <- kNN(sleep[, 1:5])
x_imp[,c(1,2,4)] <- log10(x_imp[,c(1,2,4)])
scattmatrixMiss(x_imp, delimiter = "_imp", highlight = "Dream")
```

---

scattMiss *Scatterplot with information about missing/imputed values*

Description

In addition to a standard scatterplot, lines are plotted for the missing values in one variable. If there are imputed values, they will be highlighted.

Usage

```
scattMiss(
  x,
  delimiter = NULL,
  side = 1,
  col = c("skyblue", "red", "orange", "lightgrey"),
  alpha = NULL,
  lty = c("dashed", "dotted"),
  lwd = par("lwd"),
  quantiles = c(0.5, 0.975),
```

```
  inEllipse = FALSE,
  zeros = FALSE,
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  interactive = TRUE,
  ...
)
```

Arguments

x	a matrix or data.frame with two columns.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
side	if side=1, a rug representation and vertical lines are plotted for the missing/imputed values in the second variable; if side=2, a rug representation and horizontal lines for the missing/imputed values in the first variable.
col	a vector of length four giving the colors to be used in the plot. The first color is used for the scatterplot, the second/third color for the rug representation for missing/imputed values. The second color is also used for the lines for missing values. Imputed values will be highlighted with the third color, and the fourth color is used for the ellipses (see 'Details'). If only one color is supplied, it is used for the scatterplot, the rug representation and the lines, whereas the default color is used for the ellipses. Else if a vector of length two is supplied, the default color is used for the ellipses as well.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
lty	a vector of length two giving the line types for the lines and ellipses. If a single value is supplied, it will be used for both.
lwd	a vector of length two giving the line widths for the lines and ellipses. If a single value is supplied, it will be used for both.
quantiles	a vector giving the quantiles of the chi-square distribution to be used for the tolerance ellipses, or NULL to suppress plotting ellipses (see 'Details').
inEllipse	plot lines only inside the largest ellipse. Ignored if quantiles is NULL or if there are imputed values.
zeros	a logical vector of length two indicating whether the variables are semi-continuous, i.e., contain a considerable amount of zeros. If TRUE, only the non-zero observations are used for computing the tolerance ellipses. If a single logical is supplied, it is recycled. Ignored if quantiles is NULL.

xlim, ylim	axis limits.
main, sub	main and sub title.
xlab, ylab	axis labels.
interactive	a logical indicating whether the side argument can be changed interactively (see 'Details').
...	further graphical parameters to be passed down (see graphics::par()).

Details

Information about missing values in one variable is included as vertical or horizontal lines, as determined by the side argument. The lines are thereby drawn at the observed x- or y-value. In case of imputed values, they will additionally be highlighted in the scatterplot. Supplementary, percentage coverage ellipses can be drawn to give a clue about the shape of the bivariate data distribution.

If interactiveis TRUE, clicking in the bottom margin redraws the plot with information about missing/imputed values in the first variable and clicking in the left margin redraws the plot with information about missing/imputed values in the second variable. Clicking anywhere else in the plot quits the interactive session.

Note

The argument zeros has been introduced in version 1.4. As a result, some of the argument positions have changed.

Author(s)

Andreas Alfons, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

marginplot()

Other plotting functions: aggr(), barMiss(), histMiss(), marginmatrix(), marginplot(), matrixplot(), mosaicMiss(), pairsVIM(), parcoordMiss(), pbox(), scattJitt(), scattmatrixMiss(), spineMiss()

Examples

```
data(tao, package = "VIM")
## for missing values
scattMiss(tao[,c("Air.Temp", "Humidity")])

## for imputed values
scattMiss(kNN(tao[,c("Air.Temp", "Humidity")]), delimiter = "_imp")
```

`sleep` *Mammal sleep data*

Description

Sleep data with missing values.

Format

A data frame with 62 observations on the following 10 variables.

BodyWgt a numeric vector

BrainWgt a numeric vector

NonD a numeric vector

Dream a numeric vector

Sleep a numeric vector

Span a numeric vector

Gest a numeric vector

Pred a numeric vector

Exp a numeric vector

Danger a numeric vector

Source

Allison, T. and Chichetti, D. (1976) Sleep in mammals: ecological and constitutional correlates. *Science* **194 (4266)**, 732–734.

The data set was imported from `GGobi`.

Examples

```
data(sleep, package = "VIM")
summary(sleep)
aggr(sleep)
```

spineMiss *Spineplot with information about missing/imputed values*

Description

Spineplot or spinogram with highlighting of missing/imputed values in other variables by splitting each cell into two parts. Additionally, information about missing/imputed values in the variable of interest is shown on the right hand side.

Usage

```
spineMiss(
  x,
  delimiter = NULL,
  pos = 1,
  selection = c("any", "all"),
  breaks = "Sturges",
  right = TRUE,
  col = c("skyblue", "red", "skyblue4", "red4", "orange", "orange4"),
  border = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  axes = TRUE,
  labels = axes,
  only.miss = TRUE,
  miss.labels = axes,
  interactive = TRUE,
  ...
)
```

Arguments

x	a vector, matrix or data.frame.
delimiter	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, x needs to have colnames()). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see col).
pos	a numeric value giving the index of the variable of interest. Additional variables in x are used for highlighting.
selection	the selection method for highlighting missing/imputed values in multiple additional variables. Possible values are "any" (highlighting of missing/imputed values in *any* of the additional variables) and "all" (highlighting of missing/imputed values in *all* of the additional variables).

breaks	if the variable of interest is numeric, breaks controls the breakpoints (see graphics::hist() for possible values).
right	logical; if TRUE and the variable of interest is numeric, the spinogram cells are right-closed (left-open) intervals.
col	a vector of length six giving the colors to be used. If only one color is supplied, the bars are transparent and the supplied color is used for highlighting missing/imputed values. Else if two colors are supplied, they are recycled.
border	the color to be used for the border of the cells. Use border=NA to omit borders.
main, sub	main and sub title.
xlab, ylab	axis labels.
axes	a logical indicating whether axes should be drawn on the plot.
labels	if the variable of interest is categorical, either a logical indicating whether labels should be plotted below each cell, or a character vector giving the labels. This is ignored if the variable of interest is numeric.
only.miss	logical; if TRUE, the missing/imputed values in the variable of interest are also visualized by a cell in the spineplot or spinogram. Otherwise, a small spineplot is drawn on the right hand side (see 'Details').
miss.labels	either a logical indicating whether label(s) should be plotted below the cell(s) on the right hand side, or a character string or vector giving the label(s) (see 'Details').
interactive	a logical indicating whether the variables can be switched interactively (see 'Details').
...	further graphical parameters to be passed to graphics::title() and graphics::axis().

Details

A spineplot is created if the variable of interest is categorial and a spinogram if it is numerical. The horizontal axis is scaled according to relative frequencies of the categories/classes. If more than one variable is supplied, the cells are split according to missingness/number of imputed values in the additional variables. Thus the proportion of highlighted observations in each category/class is displayed on the vertical axis. Since the height of each cell corresponds to the proportion of highlighted observations, it is now possible to compare the proportions of missing/imputed values among the different categories/classes.

If only.miss=TRUE, the missing/imputed values in the variable of interest are also visualized by a cell in the spine plot or spinogram. If additional variables are supplied, this cell is again split into two parts according to missingness/number if imputed values in the additional variables.

Otherwise, a small spineplot that visualizes missing/imputed values in the variable of interest is drawn on the right hand side. The first cell corresponds to observed values and the second cell to missing/imputed values. Each of the two cells is again split into two parts according to missingness/number of imputed values in the additional variables. Note that this display does not make sense if only one variable is supplied, therefore only.miss is ignored in that case.

If interactive=TRUE, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session.

Value

a table containing the frequencies corresponding to the cells.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in VIM. For back compatibility, the arguments xaxlabels and missaxlabels can still be supplied to ...{} and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use labels and miss.labels instead.

The code is based on the function graphics::spineplot() by Achim Zeileis.

Author(s)

Andreas Alfons, Matthias Templ, modifications by Bernd Prantner

References

M. Templ, A. Alfons, P. Filzmoser (2012) Exploring incomplete data using visualization tools. *Advances in Data Analysis and Classification*, Online first. DOI: 10.1007/s11634-011-0102-y.

See Also

histMiss(), barMiss(), mosaicMiss()

Other plotting functions: aggr(), barMiss(), histMiss(), marginmatrix(), marginplot(), matrixplot(), mosaicMiss(), pairsVIM(), parcoordMiss(), pbox(), scattJitt(), scattMiss(), scattmatrixMiss()

Examples

```
data(tao, package = "VIM")
data(sleep, package = "VIM")
## for missing values
spineMiss(tao[, c("Air.Temp", "Humidity")])
spineMiss(sleep[, c("Exp", "Sleep")])

## for imputed values
spineMiss(kNN(tao[, c("Air.Temp", "Humidity")]), delimiter = "_imp")
spineMiss(kNN(sleep[, c("Exp", "Sleep")]), delimiter = "_imp")
```

tableMiss *create table with highlighted missings/imputations*

Description

Create a `reactable` table that highlights missing values and imputed values with the same colors as `histMiss()`

Usage

```
tableMiss(x, delimiter = "_imp")
```

Arguments

`x`	a vector, matrix or `data.frame`.
`delimiter`	a character-vector to distinguish between variables and imputation-indices for imputed variables (therefore, `x` needs to have `colnames()`). If given, it is used to determine the corresponding imputation-index for any imputed variable (a logical-vector indicating which values of the variable have been imputed). If such imputation-indices are found, they are used for highlighting and the colors are adjusted according to the given colors for imputed variables (see `col`).

Examples

```
data(tao)
x_IMPUTED <- kNN(tao[, c("Air.Temp", "Humidity")])
tableMiss(x_IMPUTED[105:114, ])
x_IMPUTED[106, 2] <- NA
x_IMPUTED[105, 1] <- NA
x_IMPUTED[107, "Humidity_imp"] <- TRUE
tableMiss(x_IMPUTED[105:114, ])
```

tao *Tropical Atmosphere Ocean (TAO) project data*

Description

A small subsample of the Tropical Atmosphere Ocean (TAO) project data, derived from the `GGOBI` project.

Format

A data frame with 736 observations on the following 8 variables.

Year a numeric vector

Latitude a numeric vector

Longitude a numeric vector

Sea.Surface.Temp a numeric vector

Air.Temp a numeric vector

Humidity a numeric vector

UWind zonal wind, i.e. latitude-parallel wind

VWind meridional wind, i.e. longitude-parallel wind

Details

All cases recorded for five locations and two time periods.

Source

http://www.pmel.noaa.gov/tao/

Examples

```
data(tao, package = "VIM")
summary(tao)
aggr(tao)
```

`testdata` *Simulated data set for testing purpose*

Description

2 numeric, 2 binary, 2 nominal and 2 mixed (semi-continous) variables

Format

The format is: List of 4

- `$wna` : a `data.frame` with 500 obs. of 8 variables:
  - `x1`: numeric 10.87 9.53 7.83 8.53 8.67 ...
  - `x2`: numeric 10.9 9.32 7.68 8.2 8.41 ... ..
  - `c1`: Factor w/ 4 levels "a","b","c","d": 3 2 2 1 2 2 1 3 3 2 ...
  - `c2`: Factor w/ 4 levels "a","b","c","d": 2 3 2 2 2 2 2 4 2 2 ...
  - `b1`: Factor w/ 2 levels "0","1": 2 2 1 2 1 2 1 2 1 1 ...
  - `b2`: Factor w/ 2 levels "0","1": 2 2 1 1 1 1 1 2 2 2 ...

  - m1: numeric 0 8.29 9.08 0 0 ...
  - m2: numeric 10.66 9.39 7.8 8.11 7.33 ...
- $wona : a ‘data.frame“ with 500 obs. of 8 variables:
  - x1: numeric 10.87 9.53 7.83 8.53 8.67 ...
  - x2: numeric 10.9 9.32 7.68 8.2 8.41 ...
  - c1: Factor w/ 4 levels "a","b","c","d": 3 2 2 1 2 2 1 3 3 2 ...
  - c2: Factor w/ 4 levels "a","b","c","d": 2 3 2 2 2 2 2 4 2 2 ...
  - b1: Factor w/ 2 levels "0","1": 2 2 1 2 1 2 1 2 1 1 ...
  - b2: Factor w/ 2 levels "0","1": 2 2 1 1 1 1 1 2 2 2 ...
  - m1: numeric 0 8.29 9.08 0 0 ...
  - m2: numeric 10.66 9.39 7.8 8.11 7.33 ...
- $mixed: c("m1", "m2")
- $outlierInd: ‘NULL“

Examples

```
data(testdata)
```

---

toydataMiss *Simulated toy data set for examples*

Description

A 2-dimensional data set with additional information.

Format

data frame with 100 observations and 12 variables. The first two variables represent the fully observed data.

Examples

```
data(toydataMiss)
```

unregister_vimpute_method *Remove a user-registered* vimpute() *method*

Description

Remove a user-registered vimpute() method

Usage

```
unregister_vimpute_method(name)
```

Arguments

name	Single character string: the method name to remove. Built-in methods cannot be removed.

Value

Invisibly TRUE on success.

See Also

register_vimpute_method(), vimpute_methods()

Other vimpute method registry: register_vimpute_method(), vimpute_methods()

Examples

```
register_vimpute_method("cart",
  learner = list(regr = "regr.rpart", classif = "classif.rpart"))
unregister_vimpute_method("cart")
```

vim_as_mids *Convert a vimmi object to a mice mids object*

Description

Converts a vimmi object to long format and uses mice::as.mids() to create a proper mids object. This enables use of mice::pool(), mice::with.mids(), and other mice infrastructure.

Usage

```
vim_as_mids(x, ...)

as.mids.vimmi(x, ...)
```

Arguments

x	A vimmi object
...	Currently unused

Details

vim_as_mids() is the documented name. as.mids.vimmi() is the same function under its historical name (kept for backward compatibility): despite the dotted suffix it is a plain function, not an S3 method – mice::as.mids() is not a generic, so it never dispatches on vimmi objects.

Value

A mids object (from the mice package)

Examples

```
## Not run:
result <- vimpute(sleep, method = "ranger", m = 5,
                  boot = TRUE, uncert = "normalerror")
mids_obj <- vim_as_mids(result)
# Now use mice infrastructure:
# fits <- with(mids_obj, lm(Sleep ~ Dream + Span))
# mice::pool(fits)

## End(Not run)
```

---

vim_complete *Extract completed datasets from a vimmi object*

Description

Reconstructs one or more completed datasets by filling in imputed values from the specified imputation(s).

Usage

```
vim_complete(data, action = 1, ...)

## S3 method for class 'vimmi'
complete(data, action = 1, ...)
```

Arguments

data	A vimmi object (produced by vimpute with m > 1)
action	Specifies which completed dataset(s) to return: • Integer (1..m): return a single completed data.frame for that imputation • "all": return a named list of all m completed data.frames

- `"long"`: return a single data.frame in long format with `.imp` (imputation number) and `.id` (row number) columns

`...` Currently unused

Details

`vim_complete()` is the documented name and always works. The same function is additionally registered as an S3 method on `mice::complete()` and `tidyr::complete()`, so if either package is attached the familiar `complete(result, 1)` dispatches to it.

VIM deliberately does *not* export a `complete()` generic of its own. Both **mice** and **tidyr** export a generic of that name, so an exported VIM generic would mask them (and be masked by them), and would make any package that imports VIM and one of those packages wholesale emit "replacing previous import" at load time. Registering the method on the foreign generics gives the same user-facing call without the clash – the same approach VIM takes for `vim_as_mids`.

Value

A data.frame, list of data.frames, or long-format data.frame

Examples

```
## Not run:
result <- vimpute(sleep, method = "ranger", m = 5, boot = TRUE, uncert = "normalerror")
d1 <- vim_complete(result, 1)        # first completed dataset
all_d <- vim_complete(result, "all")  # list of 5 datasets
long_d <- vim_complete(result, "long") # long format with .imp column

# With mice or tidyr attached, the generic dispatches to the same function:
# library(mice); d1 <- complete(result, 1)

## End(Not run)
```

`vimmi` *VIM Multiple Imputations (vimmi)*

Description

S3 class for storing multiple imputations from `vimpute`. Stores the original data once and only the imputed values per variable per imputation, for memory efficiency.

Usage

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

## S3 method for class 'vimmi'
summary(object, ...)
```

Arguments

x	A vimmi object
...	Currently unused
object	A vimmi object

Details

A vimmi object is returned by vimpute when m > 1. It contains:

data The original data.frame with NAs intact (stored once).

imp Named list: for each variable with missings, a data.frame with nmis rows and m columns of imputed values.

where Logical matrix indicating which cells were imputed.

m Integer: number of imputations.

nmis Named integer vector of missing counts per variable.

method Named list of imputation methods used per variable.

boot Logical: was bootstrap resampling used?

uncert Character: uncertainty method used.

call The original function call.

tuning_log Tuning report of the (single) tuning run, or NULL.

chain Per-iteration chain statistics (mean/var arrays [variable, iteration, imputation]) behind plot.vimmi trace plots.

seed The seed applied at entry, or NULL.

Use vim_complete to extract completed datasets, with.vimmi to fit models across imputations (returns a mice-compatible mira), vim_as_mids to convert to a mice mids object for pooling with mice::pool(), and plot.vimmi for convergence trace plots.

Author(s)

Matthias Templ

See Also

vimpute, vim_complete, with.vimmi, vim_as_mids, plot.vimmi

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimpute(), xgboostImpute()

Examples

```
## Not run:
# Multiple imputation with bootstrap and normal error uncertainty
result <- vimpute(sleep, method = "ranger", m = 5,
                  boot = TRUE, uncert = "normalerror")
print(result)
summary(result)

# Extract completed datasets
d1 <- vim_complete(result, 1)
all_d <- vim_complete(result, "all")
# complete(result, 1) does the same when mice or tidyr is attached

# Fit models and pool
fits <- with(result, lm(Sleep ~ Dream + Span))
# mice::pool(fits)  # requires mice

## End(Not run)
```

`vimpute` *Impute missing values with prefered model, sequentially, with hyperparametertuning and with PMM (if wanted)*

Description

Impute missing values with prefered model, sequentially, with hyperparametertuning and with PMM (if wanted)

Usage

```
vimpute(
  data,
  ...,
  considered_variables = names(data),
  method = setNames(as.list(rep("ranger", length(considered_variables))),
    considered_variables),
  pmm = FALSE,
  pmm_k = NULL,
  pmm_k_method = "mean",
  learner_params = NULL,
  formula = FALSE,
  makeNA = NULL,
  donorcond = NULL,
  sequential = TRUE,
  nseq = 10,
  eps = 0.005,
  imp_var = TRUE,
  keep_all_columns = TRUE,
```

```
  pred_history = FALSE,
  tune = FALSE,
  verbose = FALSE,
  boot = NULL,
  robustboot = "stratified",
  uncert = "pmm",
  m = 1L,
  seed = NULL,
  tuned_params = NULL,
  tune_control = NULL,
  predictors = NULL,
  visit_sequence = "asis",
  spec = NULL
)
```

Arguments

`data`	Dataset with missing values. Provide as a data.table.
`...`	Optional bare **grammar formulas**, one per variable: `target ~ predictors | method(...)` – e.g. `Sleep ~ Dream + Span | ranger(tune = TRUE)` or `NonD ~ . | robust(donorcond = ">= 0")` – plus an optional `.default =` spec for unlisted variables. A plain-column right-hand side restricts the `predictors` (works for every method); a right-hand side with transformations (`s(x)`, `log(x)`, `I(x^2)`, interactions) becomes a model `formula` (formula-capable methods only); `.` means all other variables. Grammar formulas compile to `spec` objects and cannot be combined with `spec =` or the flat per-variable arguments.
`considered_variables`	A character vector of variable names to be either imputed or used as predictors, excluding irrelevant columns from the imputation process. Excluded columns are still returned unchanged by default (see `keep_all_columns`).
`method`	Specifies the imputation method for each variable. Can be provided either: • as a **single global method** (e.g. "ranger"), applied to all variables, or • as a **named list** (e.g. list(var1 = "xgboost", var2 = "robust")), assigning a method to each variable individually. Built-in methods: • ranger (Random Forest) • xgboost (Gradient Boosting) • regularized (glmnet regression/classification) • robust (robustbase::lmrob / glmrob) • gam (Generalized Additive Model via mgcv::gam) • robgam (Robust GAM with outlier downweighting, simple or iterative reweighting) • restricted (ECOSolveR least-squares regression with validate rules) Additional methods backed by any mlr3 learner pair can be added with `register_vimpute_method()`; `vimpute_methods()` lists everything currently registered.
`pmm`	Predictive Mean Matching (PMM) settings. Can be provided: • as a **single TRUE/FALSE** (global), or

- as a **named list**, assigning PMM per (numeric) variable.

`pmm_k` Number of nearest neighbors used in PMM. Accepted forms:

- single global integer (applies to all variables), or
- named list assigning values per variable, or
- NULL (default), meaning:
  - k = 1 automatically for variables using PMM,
  - k = NULL for variables without PMM

`pmm_k_method` Aggregation method used when `pmm_k > 1` in PMM. Default is `"mean"`. Accepted forms:

- single global string (`"mean"`, `"median"`, `"random"`), or
- single global function (called with the k nearest observed values), or
- named list assigning methods per variable, or
- `NULL` values inside such lists, which fall back to `"mean"` Semantics:
- `"mean"`: mean of the k nearest neighbors
- `"median"`: median of the k nearest neighbors
- `"random"`: random draw of one among the k nearest neighbors
- function: custom aggregator returning one numeric value

`learner_params` Hyperparameters for the chosen methods. Can be provided in **three ways**:

- **Per variable** (e.g. list(mpg = list(num.trees = 500)))
- **Per method** (e.g. list(ranger = list(num.trees = 600)))
- **Global**, applied to all variables using the same method For `restricted`, set `save_optimization_problem = TRUE` to attach the exact ECOS problem arguments to the returned object under the `restricted_optimization_problems` attribute. Set `robust = TRUE` to replace least squares with Huber loss; `huber_k` controls its tuning constant and defaults to `1.345`.

`formula` Optional modeling formula to restrict or transform predictor variables. Only supported for methods whose registry entry declares formula support: among the built-ins **regularized** (glmnet), **robust** (lmrob/glmrob), **gam** (mgcv::gam), **robgam** (robust GAM), and **restricted** (ECOSolveR least-squares with validate rules) Provide as a named list, e.g.:

- list(mpg = mpg ~ hp + drat)
- list(hp = log(hp) ~ wt + cyl) For X: follows the rules of model.matrix For Y: transformations supported are log(), exp(), sqrt(), I(1/..). Only applicable for numeric variables.

`makeNA` Optional named list that defines additional values to be treated as imputable missing values per variable, similar to `kNN()`. For variables listed in `makeNA`, only the specified values are imputed; existing `NA` values are left untouched. Variables not listed in `makeNA` continue to impute regular `NA` values.

`donorcond` Optional named list of donor conditions per variable, similar to `kNN()`. Rows whose observed target values do not satisfy the condition are excluded from the donor pool for model fitting for that variable.

`sequential` If TRUE, all variables with missing data are imputed sequentially across iterations.

`nseq` Maximum number of iterations (if sequential is TRUE).

`eps` Convergence threshold on the per-variable *relative* change between iterations: for numeric variables the mean squared change of the imputed values divided by the variance of the observed values, for factors the share of imputed cells whose category changed. The sequential process stops early once the largest per-variable change stays below `eps` for two consecutive iterations. The full iterations-by-variables change matrix is returned as `attr(result, "convergence")`.

`imp_var` If TRUE, additional columns indicating imputed values (VAR_imp) are added.

`keep_all_columns`
If TRUE (default), the full input is returned: columns excluded via `considered_variables` are passed through unchanged (original column order, with any `VAR_imp` indicators appended), matching `kNN()`, `hotdeck()` and `irmi()`. Set FALSE to return only the considered columns (plus their indicators), dropping the rest.

`pred_history` If TRUE, all predicted values across all iterations are stored.

`tune` Hyperparameter tuning flag. Can be:

- TRUE/FALSE globally
- or a list specifying tuning per variable, e.g. list(var1 = TRUE) Tuning runs once per variable, early in the iteration sequence. With `m > 1`, tuning runs once in the first imputation and the chosen parameters are shared by all `m` imputations (as in **mice**); the resulting `vimmi` object carries the tuning report in its `tuning_log` element.

`verbose` If TRUE additional debugging output is provided

`boot` If TRUE, bootstrap resampling is applied before model fitting to account for model uncertainty. Defaults to `TRUE` when `m > 1` (each imputation then refits on a bootstrap sample, giving approximately proper multiple-imputation draws) and to `FALSE` for single imputation (`m = 1`); set explicitly to override. The bootstrap strategy is controlled by `robustboot`. Most effective with method = "robust". Default: FALSE

`robustboot` Bootstrap strategy when `boot = TRUE`. Options: `"standard"` (classical bootstrap), `"stratified"` (good/bad residual split, default), `"residual"` (inverse residual weighting).

`uncert` Imputation uncertainty method applied to numeric predictions: `"pmm"` (default since 7.3.0: predictive mean matching, Little 1988 – a random draw among the 5 donors whose *predicted* values are nearest the missing cell's prediction, so imputed values are observed values; for a no-bootstrap ranger fit the donors are scored by their out-of-bag predictions), `"none"` (deterministic point prediction; the pre-7.3.0 default), `"normalerror"` (add N(0, sigma_hat)), `"resid"` (add sampled residual), `"midastouch"` (covariate-distance-weighted PMM, Siddique & Belin 2008). Factor targets draw from the predicted class probabilities whenever `uncert != "none"` (and are imputed by the most probable class otherwise). If `pmm = TRUE` is set, it takes precedence over `uncert`. Variables imputed by `method = "restricted"` always use `uncert = "none"`: a value-level draw on top of the constrained solution would break the validation rules the solver just enforced. The default is overridden silently, an explicitly requested draw mechanism warns.

`m` Number of multiple imputations. Default: 1 (single imputation). When `m > 1`, returns a `vimmi` object storing the original data and imputed values efficiently. Use `vim_complete` to extract completed datasets.

`seed` Optional single number for reproducibility. Applied once via `set.seed` at the start of the call (as in **mice**), so the whole run – including all `m` imputations – is reproducible while the `m` imputations still differ from each other. Default `NULL` leaves the random-number stream untouched.

`tuned_params` Optional named list mapping variable names to learner parameter lists (e.g. `list(Sleep = list(num.trees = 300L))`). The parameters are applied to the variable's learner without running the tuner – use this to reuse tuning results across calls (each entry of a previous run's `tuning_log` carries its chosen parameters in `$params`). Used internally by `m > 1` to share the first imputation's tuned parameters across all imputations.

`tune_control` `NULL` (default) or a `vimpute_tune_control` object controlling the tuning of `tune = TRUE`: evaluation budget, CV folds of the tuning resampling, tuner and batch size. `NULL` keeps the built-in data-size heuristics. With `m > 1` it applies to the single tuning run whose parameters all imputations share. The `tuning_log` records the budget and folds used per variable.

`predictors` Optional per-variable predictor control, the equivalent of **mice**'s `predictorMatrix` – and unlike `formula` it works for EVERY method, including `ranger` and `xgboost`. Either a named list mapping a target variable to the character vector of its predictors (e.g. `list(Sleep = c("Dream", "Span"))`), or a 0/1 (or logical) matrix with targets in rows and predictors in columns (compatible with `mice::make.predictorMatrix`). Variables without an entry use all other considered variables. A `formula` supplied for a variable takes precedence over its `predictors` entry (as in **mice**).

`visit_sequence` Order in which the variables with missings are imputed: `"asis"` (default; column order), `"increasing.na"` (fewest missings first), `"decreasing.na"`, or a character vector giving an explicit permutation of the NA-variables.

`spec` `NULL` or a named list of `vimpute_spec` objects – one per variable, e.g. `spec = list(Sleep = vs_ranger(num.trees = 300, tune = TRUE), NonD = vs_robust(donorcond = ">= 0"), .default = vs_ranger())`. Each spec bundles the variable's method, learner parameters (validated eagerly), `formula`/`predictors`, `tune`, PMM settings, `makeNA` and `donorcond`; the reserved name `".default"` covers unlisted variables. Compiles to the flat per-variable arguments, which therefore cannot be given in the same call.

Details

Missingness assumptions. Like all conditional (fully conditional specification) imputation, `vimpute()` assumes the data are **MAR** (missing at random: the probability of missingness may depend on *observed* values) – which includes **MCAR** (missing completely at random) as a special case. Under **MNAR** (missingness depending on the unobserved values themselves) imputations and downstream estimates can be biased, and no imputation method can fix this from the observed data alone; sensitivity analyses are advisable. `makeMissing` generates MCAR/MAR/MNAR missingness in complete data for exactly such simulation-based checks, and `overimpute` diagnoses the calibration of the imputation model on the observed cells.

Value

For `m = 1`: the imputed dataset, classed like the input (data.frame in, data.frame out; data.table in, data.table out). When `tune = TRUE` the tuning report is attached as `attr(result, "tuning_log")`; when `pred_history = TRUE` the prediction history is attached as `attr(result, "pred_history")`; sequential runs attach the per-variable convergence matrix as `attr(result, "convergence")` and the chain statistics as `attr(result, "chain")`; the per-variable model quality (NRMSE/PFC, out-of-bag for ranger, in-sample otherwise) is attached as `attr(result, "model_error")` – the return is always the data itself, never a wrapper list. For `m > 1`: a `vimmi` object.

Author(s)

Eileen Vattheuer, Matthias Templ, Alexander Kowarik

See Also

Other imputation methods: `hotdeck()`, `impPCA()`, `imputeCellEM()`, `imputeCellIRMI()`, `imputeCellM()`, `imputeCellMCD()`, `imputeCellwise()`, `imputeRobust()`, `imputeRobustChain()`, `irmi()`, `kNN()`, `matchImpute()`, `medianSamp()`, `rangerImpute()`, `regressionImp()`, `sampleCat()`, `vimmi`, `xgboostImpute()`

Examples

```
## Not run:
# Single imputation (default)
x <- vimpute(data = sleep, sequential = FALSE)

# Sequential imputation with 3 iterations
y <- vimpute(data = sleep, sequential = TRUE, nseq = 3)

# Impute only selected variables
z <- vimpute(data = sleep, considered_variables =
       c("Sleep", "Dream", "Span", "BodyWgt"), sequential = FALSE)

# Multiple imputation (m = 5) with bootstrap and residual uncertainty
# Returns a vimmi object
result <- vimpute(data = sleep, method = "ranger", sequential = FALSE,
                  imp_var = FALSE, m = 5, boot = TRUE, uncert = "resid")
print(result)

# Extract completed datasets
d1 <- vim_complete(result, 1)         # first imputed dataset
all_d <- vim_complete(result, "all")  # list of 5 datasets
long_d <- vim_complete(result, "long") # long format with .imp column

# Fit a model on each imputed dataset
fits <- with(result, lm(Sleep ~ Dream + Span))

# Multiple imputation with robust method and residual uncertainty
result2 <- vimpute(data = sleep, method = "robust", m = 5,
                   boot = TRUE, robustboot = "stratified",
                   uncert = "normalerror")
```

```
## End(Not run)
```

`vimpute_methods` *List the imputation methods registered for* `vimpute()`

Description

List the imputation methods registered for `vimpute()`

Usage

```
vimpute_methods()
```

Value

Character vector of registered method names (built-ins plus any methods added via `register_vimpute_method()`).

See Also

`register_vimpute_method()`, `unregister_vimpute_method()`

Other vimpute method registry: `register_vimpute_method()`, `unregister_vimpute_method()`

Examples

```
vimpute_methods()
```

`vimpute_search_space` *The built-in tuning search space of a learner*

Description

Returns the search space and default evaluation budget that `vimpute(tune = TRUE)` uses for one of its built-in learners. The bounds and budgets scale with the task size. Useful as the starting point for a `search_space` hook in `register_vimpute_method()` and for inspecting what the tuner explores.

Usage

```
vimpute_search_space(learner_id, task)
```

Arguments

`learner_id`	Single string: an mlr3 learner id used by vimpute's built-in methods, e.g. `"regr.ranger"`, `"classif.xgboost"`, `"regr.cv_glmnet"`, `"regr.lm_rob"`, `"regr.gam_imp"`.
`task`	An mlr3 task (its number of rows scales the bounds and the budget).

Value

A list with elements space (a paradox::ParamSet, or NULL if the learner has no built-in space) and n_evals (the default evaluation budget).

See Also

vimpute_tune_control(), register_vimpute_method()

Examples

```
## Not run:
task <- mlr3::TaskRegr$new("t", backend = data.frame(y = 1:20, x = 1:20),
                           target = "y")
vimpute_search_space("regr.ranger", task)

## End(Not run)
```

---

vimpute_spec *Per-variable imputation specification for* vimpute()

Description

Bundles everything vimpute() can configure for a single variable – method, learner parameters, formula or predictors, tune, PMM settings, makeNA values and a donorcond donor condition – into one object, instead of coordinating up to nine parallel per-variable arguments. Pass a named list of specs as vimpute(spec =); the reserved name ".default" supplies the spec for variables not listed (its method, learner parameters and tune/PMM knobs; formula/predictors/makeNA/donorcond are variable-specific and not allowed in ".default").

Usage

```
vimpute_spec(
  method,
  ...,
  formula = NULL,
  predictors = NULL,
  tune = FALSE,
  pmm = FALSE,
  pmm_k = NULL,
  pmm_k_method = NULL,
  makeNA = NULL,
  donorcond = NULL,
  uncert = NULL
)

vs_ranger(...)
```

```
vs_xgboost(...)

vs_regularized(...)

vs_robust(...)

vs_gam(...)

vs_robgam(...)

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

Arguments

method	Single string: a registered imputation method (see vimpute_methods()).
...	Named learner parameters for the method's learners (e.g. num.trees = 500 for "ranger"), validated eagerly.
formula	One-sided (~ x1 + s(x2)) or two-sided formula for the variable's model; only for methods with formula support. A one-sided formula is completed with the variable as its left-hand side.
predictors	Character vector of predictor columns (the predictors equivalent, works for every method). A spec takes either formula or predictors, not both.
tune	Logical: tune this variable's learner (see vimpute_tune_control()).
pmm	Logical: predictive mean matching for this (numeric) variable.
pmm_k	NULL or a positive integer: PMM donor count.
pmm_k_method	NULL, "mean", "median", "random", or a function: PMM aggregation when pmm_k > 1.
makeNA	NULL or a vector of values to be treated as missing for this variable (as in vimpute(makeNA =)).
donorcond	NULL or a donor condition string such as ">= 0" (as in vimpute(donorcond =)).
uncert	Not a spec knob – set uncert on the vimpute() call; supplying it here errors (it guards against silently treating it as a learner parameter).
x	A vimpute_spec object

Details

Learner parameters in ... are validated **eagerly** against the method's learner parameter sets, so a typo fails at the constructor call, not in iteration 7 of the imputation. vs_ranger(), vs_xgboost(), vs_regularized(), vs_robust(), vs_gam() and vs_robgam() are shorthands for the built-in methods; methods added via register_vimpute_method() use vimpute_spec("<name>", ...).

A variable with a spec uses that spec's settings exactly (no field-level merging with ".default"). uncert remains a call-level argument of vimpute() and is not a spec knob.

Value

An object of class vimpute_spec.

See Also

vimpute(), register_vimpute_method()

Examples

```
## Not run:
data(sleep)
res <- vimpute(sleep,
  spec = list(
    Sleep    = vs_ranger(num.trees = 300, tune = TRUE),
    NonD     = vs_robust(donorcond = ">= 0"),
    .default = vs_ranger()
  ),
  seed = 1)

# the same, as formula grammar:
res2 <- vimpute(sleep,
  Sleep ~ . | ranger(num.trees = 300, tune = TRUE),
  NonD  ~ . | robust(donorcond = ">= 0"),
  .default = vs_ranger(),
  seed = 1)

## End(Not run)
```

---

vimpute_tune_control *Control the hyperparameter tuning of* vimpute()

Description

Bundles the tuning controls that vimpute(tune = TRUE) previously hard-coded: the evaluation budget, the cross-validation folds of the tuning resampling, the mlr3tuning tuner and its batch size. Pass the result to vimpute(tune_control =). With m > 1 the control applies to the single tuning run (run 1), whose parameters all m imputations share.

Usage

```
vimpute_tune_control(
  budget = NULL,
  folds = NULL,
  tuner = "random_search",
  batch_size = 1L
)
```

Arguments

budget	NULL or a single positive integer: the number of configurations the tuner evaluates. NULL (default) keeps the per-learner, data-size-dependent budgets of the built-in search spaces (see vimpute_search_space()).
folds	NULL or a single integer >= 2: the cross-validation folds used during tuning (capped by what the data supports). NULL (default) keeps the heuristic of 5 folds up to 3000 rows, 3 above.
tuner	Single string: an mlr3tuning tuner id, e.g. "random_search" (default) or "grid_search". Tuners beyond the defaults may require additional packages.
batch_size	Single positive integer: configurations evaluated per tuner batch. The default 1 keeps the random-search RNG consumption machine-independent, so seed = reproduces tuning results everywhere; larger values evaluate in batches (faster with a parallel future::plan, still reproducible for a fixed value).

Details

Nested resampling is intentionally not offered: the goal of tuning inside an imputation loop is good imputations, not an unbiased estimate of the learner's generalisation error, and the tuned-vs-default comparison that vimpute() runs on a fresh resampling already guards against tuning that overfits the folds.

Value

An object of class vimpute_tune_control.

See Also

vimpute(), vimpute_search_space()

Examples

```
## Not run:
data(sleep)
res <- vimpute(sleep[, c("Sleep", "Dream", "Span")], method = "ranger",
               tune = TRUE, sequential = FALSE,
               tune_control = vimpute_tune_control(budget = 10, folds = 3))
attr(res, "tuning_log")

## End(Not run)
```

---

wine *Wine tasting and price*

Description

Wine reviews from France, Switzerland, Austria and Germany.

Format

A data frame with 9627 observations on the following 9 variables.

country country of origin

points the number of points WineEnthusiast rated the wine on a scale of 1-100 (though they say they only post reviews for wines that score >=80)

price the cost for a bottle of the wine

province the province or state that the wine is from

taster_name name of the person who tasted and reviewed the wine

taster_twitter_handle Twitter handle for the person who tasted ane reviewed the wine

variety the type of grapes used to make the wine (ie pinot noir)

winery the winery that made the wine

variety_main broader category as variety

Details

The data was scraped from WineEnthusiast during the week of Nov 22th, 2017. The code for the scraper can be found at https://github.com/zackthoutt/wine-deep-learning This data set is slightly modified, i.e. only four countries are selected and broader categories on the variety have been added.

Source

https://www.kaggle.com/zynicide/wine-reviews

Examples

```
data(wine)
str(wine)
aggr(wine)
```

---

with.vimmi *Evaluate an expression across all imputations*

Description

Applies an expression (typically a model fit) to each completed dataset in a `vimmi` object. The return is a mice-compatible `mira` object (elements `call`, `call1`, `nmis`, `analyses`), so the standard mice pipeline runs unchanged: `mice::pool()`, `summary(mice::pool(fits))`, `mice::getfit(fits)`. The mice package is not needed to create the object, only to pool it; for `mitools::MIcombine()` pass the fit list `fits$analyses`.

Usage

```
## S3 method for class 'vimmi'
with(data, expr, ...)
```

Arguments

data	A vimmi object
expr	An expression to evaluate, e.g. lm(y ~ x)
...	Currently unused

Value

An object of class mira: the m results are in $analyses (extract with mice::getfit())

Examples

```
## Not run:
result <- vimpute(sleep, method = "ranger", m = 5, boot = TRUE, uncert = "normalerror")
fits <- with(result, lm(Sleep ~ Dream + Span))
# Pool with mice:
# mice::pool(fits)

## End(Not run)
```

---

xgboostImpute *Xgboost Imputation*

Description

Impute missing values based on a random forest model using xgboost::xgboost()

Usage

```
xgboostImpute(
  formula,
  data,
  imp_var = TRUE,
  imp_suffix = "imp",
  verbose = FALSE,
  nrounds = 100,
  objective = NULL,
  ...
)
```

Arguments

formula	model formula for the imputation
data	A data.frame containing the data
imp_var	TRUE/FALSE if a TRUE/FALSE variables for each imputed variable should be created show the imputation status
imp_suffix	suffix used for TF imputation variables
verbose	Show the number of observations used for training and evaluating the RF-Model. This parameter is also passed down to xgboost::xgboost() to show computation status.
nrounds	max number of boosting iterations, argument passed to xgboost::xgboost()
objective	objective for xgboost, argument passed to xgboost::xgboost()
...	Arguments passed to xgboost::xgboost()

Value

the imputed data set.

See Also

Other imputation methods: hotdeck(), impPCA(), imputeCellEM(), imputeCellIRMI(), imputeCellM(), imputeCellMCD(), imputeCellwise(), imputeRobust(), imputeRobustChain(), irmi(), kNN(), matchImpute(), medianSamp(), rangerImpute(), regressionImp(), sampleCat(), vimmi, vimpute()

Examples

```
data(sleep)
sleepx <- sleep
sleepx$Pred <- as.factor(LETTERS[sleepx$Pred])
sleepx$Pred[1] <- NA
# a numeric and a factor target imputed at once; nrounds = 30 keeps the
# example fast (the default is 100 boosting rounds)
# xgboostImpute() runs on vimpute(), which VIM backs with the suggested
# mlr3 stack -- skip the example where that stack is not installed.
if (requireNamespace("mlr3", quietly = TRUE) &&
    requireNamespace("mlr3learners", quietly = TRUE) &&
    requireNamespace("mlr3pipelines", quietly = TRUE)) {
  imp <- xgboostImpute(Dream + Pred ~ BodyWgt + BrainWgt, data = sleepx, nrounds = 30)
  head(imp)
}
```

Index

∗ **color**
alphablend, 8
colSequence, 25
rugNA, 100
∗ **datasets**
Animals_na, 9
bcancer, 12
brittleness, 14
chorizonDL, 16
colic, 19
collisions, 21
diabetes, 27
food, 30
kola.background, 67
lse_synthetic, 67
lse_synthetic_rules, 68
pulplignin, 95
SBS5242, 102
sleep, 110
tao, 114
testdata, 115
toydataMiss, 116
wine, 131
∗ **hplot**
aggr, 4
barMiss, 10
bgmap, 13
colormapMiss, 22
growdotMiss, 33
histMiss, 36
mapMiss, 70
marginmatrix, 72
marginplot, 74
matrixplot, 78
mosaicMiss, 81
pairsVIM, 85
parcoordMiss, 87
pbox, 90
scattJitt, 103
scattmatrixMiss, 105
scattMiss, 107
spineMiss, 111
∗ **imputation methods**
hotdeck, 39
impPCA, 41
imputeCellEM, 43
imputeCellIRMI, 46
imputeCellM, 49
imputeCellMCD, 51
imputeCellwise, 56
imputeRobust, 57
imputeRobustChain, 58
irmi, 61
kNN, 64
matchImpute, 77
medianSamp, 81
rangerImpute, 96
regressionImp, 99
sampleCat, 101
vimmi, 119
vimpute, 121
xgboostImpute, 133
∗ **manip**
evaluation, 29
gapMiss, 31
hotdeck, 39
impPCA, 41
initialise, 60
irmi, 61
kNN, 64
matchImpute, 77
prepare, 93
∗ **plotting functions**
aggr, 4
barMiss, 10
histMiss, 36
marginmatrix, 72
marginplot, 74

matrixplot, 78
mosaicMiss, 81
pairsVIM, 85
parcoordMiss, 87
pbox, 90
scattJitt, 103
scattmatrixMiss, 105
scattMiss, 107
spineMiss, 111
∗ **print**
aggr, 4
∗ **utilities**
countInf, 26
∗ **vimpute method registry**
register_vimpute_method, 97
unregister_vimpute_method, 117
vimpute_methods, 127
)‘ *(*kNN*)*, 64
= *(*kNN*)*, 64
‘R *(*kNN*)*, 64

2 *(*kNN*)*, 64

aggr, 4, *12*, *38*, *73*, *76*, *80*, *83*, *86*, *89*, *92*, *104*, *107*, *109*, *113*
aggr(), *7*
alphablend, 8
Animals_na, 9
are *(*kNN*)*, 64
as.mids.vimmi *(*vim_as_mids*)*, 117

barMiss, *7*, 10, *38*, *73*, *76*, *80*, *83*, *86*, *89*, *92*, *104*, *107*, *109*, *113*
barMiss(), *38*, *113*
bcancer, 12
bgmap, 13
bgmap(), *34*, *35*, *71*, *72*
brittleness, 14
bubbleMiss *(*growdotMiss*)*, 33
bubbleMiss(), *72*
by *(*kNN*)*, 64

car::powerTransform(), *94*
cellWeightsMCD, 15
check‘); *(*kNN*)*, 64
chorizonDL, 16
CMD *(*kNN*)*, 64
colic, 19
collisions, 21
colnames(), *5*, *10*, *34*, *37*, *71*, *73*, *74*, *78*, *82*, *85*, *87*, *90*, *103*, *105*, *108*, *111*, *114*
colormapMiss, 22
colormapMiss(), *35*, *72*
colormapMissLegend *(*colormapMiss*)*, 22
colorspace::hex(), *23–26*, *79*
colorspace::polarLUV(), *23*, *25*, *79*
colorspace::RGB(), *23*, *25*, *79*
colorspace::sequential_hcl(), *26*
colSequence, 25
colSequence(), *25*
colSequenceHCL *(*colSequence*)*, 25
colSequenceRGB *(*colSequence*)*, 25
complete.vimmi *(*vim_complete*)*, 118
computed *(*kNN*)*, 64
countInf, 26
countNA *(*countInf*)*, 26

diabetes, 27
Distances *(*kNN*)*, 64
during *(*kNN*)*, 64

evaluation, 29, *69*
evaluation(), *70*, *84*

food, 30
format(), *35*

gam, *58*
gapMiss, 31
glm(), *100*
glmrob(), *100*
governed *(*kNN*)*, 64
gowerD, 32
graphics::Axis(), *101*
graphics::axis(), *11*, *37*, *79*, *112*
graphics::boxplot(), *91*
graphics::hist(), *112*
graphics::lines(), *14*
graphics::pairs(), *86*
graphics::par(), *73*, *75*, *85*, *88*, *104*, *106*, *109*
graphics::plot.window(), *79*
graphics::points(), *71*
graphics::spineplot(), *113*
graphics::title(), *11*, *37*, *79*, *112*
growdotMiss, 33
growdotMiss(), *14*, *25*

hist(), *37*

histMiss, *7*, *12*, 36, *73*, *76*, *80*, *83*, *86*, *89*, *92*, *104*, *107*, *109*, *113*
histMiss(), *12*, *113*, *114*
hotdeck, 39, *42*, *45*, *48*, *51*, *53*, *57*, *58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
hotdeck(), *77*

iimagMiss *(*matrixplot*)*, 78
impPCA, *40*, 41, *45*, *48*, *51*, *53*, *57*, *58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
imputeCellEM, *40*, *42*, 43, *48*, *51*, *53*, *56–58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
imputeCellIRMI, *40*, *42*, *45*, 46, *50*, *51*, *53*, *56–58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
imputeCellM, *40*, *42*, *45*, *48*, 49, *53*, *56–58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
imputeCellMCD, *40*, *42*, *45*, *48*, 51, *51*, *57*, *58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
imputeCellMM, 54
imputeCellReg, 55
imputeCellwise, *40*, *42*, *45*, *48*, *51*, *53*, 56, *58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
imputeRobust, *40*, *42*, *45*, *48*, *51*, *53*, 57, *57*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
imputeRobustChain, *40*, *42*, *45*, *48*, *51*, *53*, *57*, 58, *58*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
initialise, *44*, *45*, *47*, *48*, *58*, 60, *60*
irmi, *40*, *42*, *45*, *48*, *51*, *53*, *57*, *58*, *60*, 61, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*

kNN, *40*, *42*, *45*, *48*, *51*, *53*, *57*, *58*, *60*, *63*, 64, *77*, *81*, *96*, *100*, *102*, *120*, *126*, *134*
kNN(), *70*
kola.background, 67

lm(), *100*
lmrob, *58*
lmrob(), *100*
lse_synthetic, 67, *68*
lse_synthetic_rules, *67*, 68

makeMissing, 69, *125*
makeMissing(), *84*
mapMiss, 70
mapMiss(), *14*, *25*, *35*
marginmatrix, *7*, *12*, *38*, 72, *76*, *80*, *83*, *86*, *89*, *92*, *104*, *107*, *109*, *113*
marginmatrix(), *86*, *107*
marginplot, *7*, *12*, *38*, *73*, 74, *80*, *83*, *86*, *89*, *92*, *104*, *107*, *109*, *113*
marginplot(), *73*, *109*
matchImpute, *40*, *42*, *45*, *48*, *51*, *53*, *57*, *58*, *60*, *63*, *66*, 77, *81*, *96*, *100*, *102*, *120*, *126*, *134*
matrixplot, *7*, *12*, *38*, *73*, *76*, 78, *83*, *86*, *89*, *92*, *104*, *107*, *109*, *113*
maxCat, 80
medianSamp, *40*, *42*, *45*, *48*, *51*, *53*, *57*, *58*, *60*, *63*, *66*, *77*, 81, *96*, *100*, *102*, *120*, *126*, *134*
mosaicMiss, *7*, *12*, *38*, *73*, *76*, *80*, 81, *86*, *89*, *92*, *104*, *107*, *109*, *113*
mosaicMiss(), *113*
most *(*kNN*)*, 64
msecor *(*evaluation*)*, 29
msecov *(*evaluation*)*, 29
multinom, *50*

nrmse, *69*
nrmse *(*evaluation*)*, 29

OpenMP *(*kNN*)*, 64
outCoDa, *60*
overimpute, 83, *125*

pairsVIM, *7*, *12*, *38*, *73*, *76*, *80*, *83*, 85, *89*, *92*, *104*, *107*, *109*, *113*
pairsVIM(), *73*, *106*, *107*
parcoordMiss, *7*, *12*, *38*, *73*, *76*, *80*, *83*, *86*, 87, *92*, *104*, *107*, *109*, *113*
parcoordMiss(), *92*
pbox, *7*, *12*, *38*, *73*, *76*, *80*, *83*, *86*, *89*, 90, *104*, *107*, *109*, *113*
pbox(), *89*
pdist, *58*
pfc, *69*
pfc *(*evaluation*)*, 29
plot.aggr *(*aggr*)*, 4
plot.vimmi, 92, *120*
plot.vimpute_overimpute *(*overimpute*)*, 83

prepare, 93
print.aggr *(aggr)*, 4
print.aggr(), *7*
print.default(), *6*
print.summary.aggr *(aggr)*, 4
print.summary.aggr(), *7*
print.vimmi *(vimmi)*, 119
print.vimpute_overimpute *(overimpute)*, 83
print.vimpute_spec *(vimpute_spec)*, 128
pulplignin, 95

ranger::ranger(), *65*
rangerImpute, *40*, *42*, *45*, *48*, *51*, *53*, *57, 58*, *60*, *63*, *66*, *77*, *81*, 96, *100*, *102*, *120*, *126*, *134*
register_vimpute_method, 97, *117*, *122*, *127*
register_vimpute_method(), *117*, *127–130*
regressionImp, *40*, *42*, *45*, *48*, *51*, *53*, *57, 58*, *60*, *63*, *66*, *77*, *81*, *96*, 99, *102*, *120*, *126*, *134*
robustbase::covMcd(), *94*
rugNA, 100

sampleCat, *40*, *42*, *45*, *48*, *51*, *53*, *57, 58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, 101, *120*, *126*, *134*
SBS5242, 102
scale(), *94*
scattJitt, *7*, *12*, *38*, *73*, *76*, *80*, *83*, *86*, *89*, *92*, 103, *107*, *109*, *113*
scattmatrixMiss, *7*, *12*, *38*, *73*, *76*, *80*, *83*, *86*, *89*, *92*, *104*, 105, *109*, *113*
scattmatrixMiss(), *73*, *86*
scattMiss, *7*, *12*, *38*, *73*, *76*, *80*, *83*, *86*, *89*, *92*, *104*, 107, *107*, *113*
scattMiss(), *76*
see *(kNN)*, 64
set.seed, *125*
sleep, 110
spineMiss, *7*, *12*, *38*, *73*, *76*, *80*, *83*, *86*, *89*, *92*, *104*, *107*, *109*, 111
spineMiss(), *12*, *38*, *83*
summary.aggr *(aggr)*, 4
summary.aggr(), *7*
summary.vimmi *(vimmi)*, 119

tableMiss, 114
tao, 114
testdata, 115
there. *(kNN)*, 64
threads *(kNN)*, 64
TKRmatrixplot *(matrixplot)*, 78
toydataMiss, 116

unregister_vimpute_method, *98*, 117, *127*
unregister_vimpute_method(), *98*, *127*

vcd::labeling_border(), *82*
vcd::mosaic(), *82, 83*
vim_as_mids, 117, *119, 120*
vim_complete, 118, *120*, *125*
vimmi, *40*, *42*, *45*, *48*, *51*, *53*, *57, 58*, *60*, *63*, *66*, *77*, *81*, *93*, *96*, *100*, *102*, 119, *125, 126*, *134*
vimpute, *40*, *42*, *45*, *48*, *51*, *53*, *57, 58*, *60*, *63*, *66*, *77*, *81*, *93*, *96*, *100*, *102*, *118–120*, 121, *134*
vimpute(), *63*, *70*, *83, 84*, *96*, *98*, *100*, *129–131*
vimpute_methods, *98*, *117*, *122*, 127
vimpute_methods(), *98*, *117*, *129*
vimpute_search_space, 127
vimpute_search_space(), *131*
vimpute_spec, *125*, 128
vimpute_tune_control, *125*, 130
vimpute_tune_control(), *128, 129*
vs_gam *(vimpute_spec)*, 128
vs_ranger *(vimpute_spec)*, 128
vs_regularized *(vimpute_spec)*, 128
vs_robgam *(vimpute_spec)*, 128
vs_robust *(vimpute_spec)*, 128
vs_xgboost *(vimpute_spec)*, 128

whose *(kNN)*, 64
wine, 131
with.vimmi, *120*, 132

xgboost::xgboost(), *133, 134*
xgboostImpute, *40*, *42*, *45*, *48*, *51*, *53*, *57, 58*, *60*, *63*, *66*, *77*, *81*, *96*, *100*, *102*, *120*, *126*, 133