Package 'hmi'

October 2, 2020

Type Package

Title Hierarchical Multiple Imputation

Version 1.0.0

Maintainer Matthias Speidel <matthias.speidel@googlemail.com>

Description Runs single level and multilevel imputation models as described in Speidel, Drechsler and Jolani (2020) <doi:10.18637/jss.v095.i09>. The user just has to pass the data to the main function and, optionally, his analysis model. Basically the package then translates this analysis model into commands to impute the data according to it with functions from 'mice', 'MCMCglmm' or routines build for this package.

BugReports https://github.com/matthiasspeidel/hmi/issues

Imports boot (>= 1.3-20), broom.mixed (>= 0.2.6), coda (>= 0.19-1), graphics (>= 3.0.0), linLIR (>= 1.1), lme4 (>= 1.1-15), MASS (>= 7.3-49), Matrix (>= 1.2-12), MCMCglmm (>= 2.25), mice (>= 3.5.0), msm (>= 1.6.6), mvtnorm (>= 1.0-7), nlme (>= 3.1-131.1), nnet (>= 7.3-12), ordinal (>= 2015.6-28), pbivnorm (>= 0.6.0), rlang (>= 0.3.0.1), stats (>= 3.0.0), tmvtnorm (>= 1.4-10), utils,

Suggests knitr, rmarkdown, testthat

Depends R (>= 3.4.0)

License GPL-3

LazyData TRUE

Encoding UTF-8

RoxygenNote 7.1.0

VignetteBuilder knitr

URL https://github.com/matthiasspeidel/hmi

NeedsCompilation no

Author Matthias Speidel [aut, cre] (Munich, Germany),
Joerg Drechsler [aut] (Institute for Employment Research, Nuremberg, Germany),
Shahab Jolani [aut] (Maastricht University, Maastricht, The Netherlands)

Repository CRAN

Date/Publication 2020-10-01 23:20:02 UTC

R topics documented:

`*.interval` *Multiplication function*

Description

Function to multiply single elements or vectors (of correct dimension) to the interval object

Usage

```
## S3 method for class 'interval'
interval * x
```

Arguments

`interval`	an object from class interval
`x`	an single element or vector for multiplication

Value

an interval object

+.interval *Adding function*

Description

Function to add single elements or vectors (of correct dimension) to the interval object

Usage

```
## S3 method for class 'interval'
interval + x
```

Arguments

`interval`	an object from class interval
`x`	an single element or vector to add to the interval object

Value

an interval object

-.interval *Subtraction function*

Description

Function to subtract single elements or vectors (of correct dimension) from the interval object

Usage

```
## S3 method for class 'interval'
interval - x
```

Arguments

`interval`	an object from class interval
`x`	an single element or vector to subtract to the interval object

Value

an interval object

`/.interval` *Dividing function*

Description

Function to divide single elements or vectors (of correct dimension) to the interval object

Usage

```
## S3 method for class 'interval'
interval / x
```

Arguments

`interval`	an object from class interval
`x`	an single element or vector for division

Value

an interval object

`as.data.frame.interval` *Transform interval objects into data.frames*

Description

This function transforms interval objects into data.frames. This is not only relevant on its own, it is also needed whenever a function need objects as a data.frame (e.g. `View` or `cbind`).

Usage

```
## S3 method for class 'interval'
as.data.frame(x, ...)
```

Arguments

`x`	an object of class interval.
`...`	arguments to be passed to `as.data.frame`.

Value

a data.frame containing x as a character

`as.interval` *Function to transform objects into an interval object*

Description

Function to transform numeric (or character) vectors or n times 2 matrices into an interval object

Usage

```
as.interval(x, sna = FALSE)
```

Arguments

`x`	An object to transform. Currently the function can transform numeric vectors and characters
`sna`	Boolean: if `TRUE`, `NA`s are kept as standard `NA`s. Otherwise they are turned into `"-Inf;Inf"`.

Value

A vector of class `interval`.

See Also

generate_interval

Examples

```
as.interval(c("1000;2000", "700;700", NA))
```

`ceiling.interval` *Ceiling funtion for intervals*

Description

Ceiling funtion for intervals

Usage

```
## S3 method for class 'interval'
ceiling(x, ...)
```

Arguments

`x`	numeric vector or interval object
`...`	further arguments passed to `ceiling`.

`center_interval` *Function to give the center of the interval*

Description

Function to give the center of the interval object

Usage

```
center_interval(interval, inf2NA = FALSE)
```

Arguments

`interval`	an object from class interval
`inf2NA`	logical. If `TRUE`, entries containing -Inf or Inf, will return NA.

Value

A numeric vector

`chaincheck` *Checking the chains on convergence*

Description

Formally tests the Gibbs-sampling chains on convergence. After the burn in is discarded, the remaining iterations of each chain are tested following Geweke (1992). In this test, the arithmetic means and their standard errors of the first 10% and last 50% of the chain (from now on always after discarding the burn in) are compared. In case of a stationary distribution, both means have the same expected value. The difference between both arithmetic means is divided the standard error. This is the Z-score, the test statistic. Chains not passing the test will be plotted. Each plot will flag which (fixed effect or variance) parameter was tested; and what variable was to be imputed and the cycle and imputation run. To see the next plot, the user has to hit <Return> (or "Enter").

Usage

```
chaincheck(mids, alpha = 0.01, thin = 1, plot)
```

Arguments

`mids`	A mids object generated by hmi (alternatively a list), having an element called "gibbs" with the chains of the Gibbs-sampler runs.
`alpha`	A numeric value between 0 and 1 for the desired significance level of the test on convergence.

`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000), the `geweke.diag` might fail to detect convergence. In such cases it is essential to look at a chain free from autocorrelation. When setting `thin = NULL`, the function will use internally a thinning of `max(1,round((nitt-burnin)/1000))` to get approximately 1000 iterations to be tested.
`plot`	Logical. Shall the chains be plotted in a traceplot or not. If the number of iterations and cycles is large, click through all traceplots can be interminable.

References

J Geweke (1992): Evaluating the accuracy of sampling based approaches to calculating posterior moments. In Bayesian Statistics 4 (ed. JB Bernando, JO Berger, AP Dawid and Adrian FM Smith) (pp. 169-193). Clarendon Press, Oxford, UK.

`cleanup` *cleanup data.frames*

Description

Function to exclude (factor or interval) variables that have too many levels (as they may cause numerical problems), to change binary factors to 0-1 coding (as such factors might generate linear dependent variables) and to remove multiple intercepts.

Usage

```
cleanup(X, k = Inf)
```

Arguments

`X`	A n times p data.frame with p fixed (or random) effects variables.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n times (p-r) data.frame, with r being the number of variables with too many factors.

`contributions4intervals` *get the likelihood contributions of interval data*

Description

This function calculates the likelihood contributions of interval data

Usage

```
contributions4intervals(lower_bounds, upper_bounds, mymean, mysd)
```

Arguments

`lower_bounds`	a vector with the lower bounds of an interval covariate.
`upper_bounds`	a vector with the upper bounds of an interval covariate.
`mymean`	a numeric for the expected value of the normal distribution (which is one of the parameters trying to be optimized so that the likelihood becomes maximized)
`mysd`	a numeric for the standard deviation of the normal distribution (which is one of the parameters trying to be optimized so that the likelihood becomes maximized)

Value

a vector giving the likelihood contributions of the interval data.

`decompose_interval` *decompose up intervals*

Description

This function decomposes an interval object up into precise observations (e.g. "1850.23;1850.23" into 1850.23), imprecise observations (e.g. "1800;1900") and missing observations ("-Inf;Inf" into NA)

Usage

```
decompose_interval(interval)
```

Arguments

`interval`	A vector, factor or optimally an `interval` object of length n (if it is something else, it is returned unchanged)

Value

A matrix with 5 columns. 1. A column "precise" for the precise observations (length of interval = 0, e.g. "3000;3000"). If observation i is not precise, the i-th entry in this columns will be `NA`. c("2500;2500", "3000;4000", "-Inf;0", NA) will lead to c(2500, NA, NA, NA) 2. A column "lower" for the values of the lower bounds of the imprecise observations (length of interval > 0, e.g. "3000;4000" or "-Inf;0"), precise observations will get `NA`s here. c("2500;2500", "3000;4000", "-Inf;0", NA) will lead to c(NA, 3000, -Inf, NA) 3. A column "upper" for the values of the upper bounds of the imprecise observations. c("2500;2500", "3000;4000", "-Inf;0", NA) will lead to c(NA, 4000, 0, NA) 4. A column "lower_general" with the lower bound values of all observations, without distinction between precise, imprecise or missing observations. c("2500;2500", "3000;4000", "-Inf;0", NA) will lead to c(2500, 3000, -Inf, -Inf) c("2500;2500", "3000;4000", "-Inf;0", NA) will lead to c(2500, 4000, 0, Inf)

`doubleintegral` *Function to calculate double integrals*

Description

This function is primarily build to make the function `components` neater.

Usage

```
doubleintegral(lower_inner, upper_inner, lower_outer, upper_outer, cdf, ...)
```

Arguments

`lower_inner`	The vector for the lower bound for the inner integral
`upper_inner`	The vector for the upper bound for the inner integral
`lower_outer`	The vector for the lower bound for the outer integral
`upper_outer`	The vector for the upper bound for the outer integral
`cdf`	the cumulative density function (from the class "function")
`...`	Further arguments passed to the cdf.

Value

a vector with the value of the double integral for each observation (with an observed target variable)

exp.interval *Exp function for interval objects*

Description

Exp function for interval objects

Usage

```
## S3 method for class 'interval'
exp(x, ...)
```

Arguments

x	numeric vector or interval object
...	further arguments passed to exp.

extract_varnames *Function to extract the different elements of a formula*

Description

The function searches for the target variable, fixed effects variables, if there is a cluster ID: this and the random effects variables.
The names of the fixed and random intercepts variable (if existent) are explicitly labeled In imputation models, the target variable can act as covariate for other covariates - so we treat the target variable as fix effect variable.

Usage

```
extract_varnames(
  model_formula = NULL,
  constant_variables,
  variable_names_in_data = colnames(data),
  data
)
```

Arguments

model_formula	A formula (from class formula)
constant_variables	A Boolean-vector of length equal to the number of columns in the data set specifying whether a variable is a constant variable (eg. an intercept variable) or not.
variable_names_in_data	A character-vector with the column names of the data set.
data	The data.frame the formula belongs to.

Value

A list with the names of the target variable, the intercept variable, the fixed and random effects covariates (which includes the name of the target variable), the variables with interactions and the cluster id variable.
If some of them don't exist, they get the value "".

`factors` *Function to get all factors*

Description

Function to get all factors (not limited to prime factors) of an integer.

Usage

```
factors(x)
```

Arguments

`x`	A single integer; no vector.

Value

A numeric vector with the factors

References

based on stackoverflow.com/questions/6424856 "R Function for returning ALL factors" answer by Chase

---

`fixed_intercept_check` *Function to check multilevel models on the existence of fixed intercepts*

Description

Function to check multilevel models on the existence of fixed intercepts. The specification of an intercept by calling a 1-column (e.g. "int") is not counted towards the existence of an intercept. Contradictory inputs like "~ 1 + 0 + X1 + ..." or "~ -1 + 1 + X1 + ..." will throw an error.

Usage

```
fixed_intercept_check(model_formula)
```

Arguments

`model_formula`	A formula (from class `formula`)

Value

A boolean value indicating whether there is a fixed intercept in the model or not

---

`floor.interval` *Floor function for interval objects Floor function for interval objects*

Description

Floor function for interval objects Floor function for interval objects

Usage

```
## S3 method for class 'interval'
floor(x, ...)
```

Arguments

`x`	numeric vector or interval object
`...`	further arguments passed to `floor`.

`Gcsemv` *Exam results on the GCSE*

Description

A dataset containing results on the General Certificate of Secondary Education (GCSE) exams in Great Britain for 1905 students in 73 Schools.

Usage

```
Gcsemv
```

Format

A data frame with 1905 rows and 5 variables:

school The ID of the student's school.

student The ID of the student within its school. They are not unique, different schools may have identical IDs for not identical students.

gender The gender of the student: 0 for boys and 1 for girls.

written The score in a written questionnaire.

coursework The score in a coursework.

Source

Website of the Centre for Multilevel Modelling at the University of Bristol: http://www.bristol.ac.uk/cmm/media/team/hg/msm-3rd-ed/gcsemv.xls

References

Creswell M. (1991). A Multilevel bivariate model. In R. Prosser, J. Rasbash, H. Goldstein (eds.), Data Analysis with ML3. Institute of Education, London.

Goldstein H. (2011) Multilevel Statistical Models. 4 edition. Wiley, Chichseter (UK), ISBN 978-0-470-74865-7.

`generate_interval` *Function to generate an interval object*

Description

Function to generate transform into an `interval` object from numeric (or character) vectors.

Usage

```
generate_interval(lower, upper, sna = FALSE)
```

Arguments

`lower`	a vector with the lower values of a variable
`upper`	a vector with the upper values of a variable
`sna`	Boolean: if `TRUE`, `NA`s are kept as standard `NA`s. Otherwise they are turned into `"-Inf;Inf"`.

Value

a character vector with the lower and upper bound values.

get_type *Get the type of variables.*

Description

Function checks of which type a variable is. The types are listed below (together with a rough summary of the classification mechanism).

- continuous (numeric values, or integers with more than 20 different values),
- semicontinuous (numeric values with more than 10% of them share the same value),
- rounded continuous (if more than 50% of the observations of a continuous variable are divisible by some rounding degrees)
- count data (integers).
- an intercept (the same value for all observations),
- binary (two different values - like 0s and 1s or "m" and "f"),
- categorical (the variable is a factor or has more than 3 different values)
- ordered categorical (the categorical variable is ordered.)

Usage

```
get_type(variable, spike = NULL, rounding_degrees = c(1, 10, 100, 1000))
```

Arguments

`variable`	A variable (vector) from your data set.
`spike`	A numeric value, denoting the presumed spike of semi-continuous variables.
`rounding_degrees`	A numeric vector with the presumed rounding degrees. If the rounding_degrees are set to be `NULL`, the proposed rounding degrees from the function `suggest_rounding_degrees` are used.

Value

A character denoting the type of `variable`.

Examples

```
get_type(iris$Sepal.Length); get_type(iris$Species)
```

head.interval *Head for intervals*

Description

Head function for intervals returning the first elements of an `interval` object

Usage

```
## S3 method for class 'interval'
head(x, ...)
```

Arguments

`x`	vector, matrix, table, data.frame or interval object
`...`	further arguments passed to `head`.

hmi *hmi: Hierarchical Multilevel Imputation.*

Description

The user has to pass his data to the function. Optionally he passes his analysis model formula so that `hmi` runs the imputation model in line with his analysis model formula.
And of course he can specify some parameters for the imputation routine (like the number of imputations and iterations and the number of iterations within the Gibbs sampling).

Usage

```
hmi(
  data,
  model_formula,
  family,
  additional_variables,
  list_of_types,
  m = 5,
  maxit,
  nitt = 22000,
  burnin = 2000,
  pvalue = 1,
  mn = 1,
  k,
  spike,
  heap,
  rounding_degrees,
  rounding_formula = ~.,
  pool_with_mice = TRUE
)
```

Arguments

`data`	A `data.frame` with all variables appearing in `model_formula`.
`model_formula`	A `formula` used for the analysis model. Currently the package is designed to handle formula used in `lm`, `glm`, `lmer` and `glmer`. The formula is also used for the default pooling.
`family`	To improve the default pooling, a family object supported by `glm` (resp. `glmer`) an be given. See `family` for details.
`additional_variables`	A character with names of variables (separated by "+", like "x8+x9") that should be included in the imputation model as fixed effects variables, but not in the analysis model. An alternative would be to include such variable names into the `model_formula` and run a reduced analysis model with `hmi_pool` or the functions provide by `mice`.
`list_of_types`	a list where each list element has the name of a variable in the data.frame. The elements have to contain a single character denoting the type of the variable. See `get_type` for details about the variable types. With the function `list_of_types_maker`, the user can get the framework for this object. In most scenarios this is should not be necessary. One example where it might be necessary is when only two observations of a continuous variable are left - because in this case `get_type` interpret is variable to be binary. Wrong is it in no case.
`m`	An integer defining the number of imputations that should be made.
`maxit`	An integer defining the number of times the imputation cycle (imputing $x_1\|x_{-1}$ then $x_2\|x_{-2}$, ... and finally $x_p\|x_{-p}$) shall be repeated. The task of checking convergence is left to the user, by evaluating the chainMean and chainVar!
`nitt`	An integer defining number of MCMC iterations (see `MCMCglmm`).
`burnin`	An integer for the desired number of Gibbs samples that shall be regarded as burnin.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`mn`	An integer defining the minimum number of individuals per cluster.
`k`	An integer defining the allowed maximum of levels in a factor covariate.
`spike`	A numeric value saying which value in the semi-continuous data might be the spike. Or a list with with such values and names identical to the variables with spikes (see `list_of_spikes_maker` for details.) In versions earlier to 0.9.0 it was called heap.
`heap`	Use spike instead. heap is only included due to backwards compatibility and will be removed with version 1.0.0

`rounding_degrees`
: A numeric vector with the presumed rounding degrees of rounded variables. Or a list with rounding degrees, where each list element has the name of a rounded continuous variable. Such a list can be generated using `list_of_rounding_degrees_maker(data)`. Note: it is presupposed that the rounding degrees include 1 meaning that there is a positive probability that e.g. 3500 was only rounded to the nearest integer (and not rounded to the nearest multiple of 100 or 500).

`rounding_formula`
: A formula with the model formula for the latent rounding tendency G. Or a list with model formulas for G, where each list element has the name of a rounded continuous variable. Such a list can be generated using `list_of_rounding_formulas_maker(data)`

`pool_with_mice`
: A Boolean indicating whether the user wants to pool the `m` data sets by mice using his `model_formula`. The default value is `FALSE` because this tampers the `mids` object as it adds an argument `pooling` not found in "normal" `mids` objects generated by `mice`.

Value

The function returns a `mids` object. See `mice` for further information.

References

Matthias Speidel, Joerg Drechsler and Shahab Jolani (2020): "The R Package hmi: A Convenient Tool for Hierarchical Multiple Imputation and Beyond", Journal of Statistical Software, Vol. 95, No. 9, p. 1–48, http://dx.doi.org/10.18637/jss.v095.i09

Examples

```
## Not run:
 data(Gcsemv, package = "hmi")

 model_formula <- written ~ 1 + gender + coursework + (1 + gender|school)

 set.seed(123)
 dat_imputed <- hmi(data = Gcsemv, model_formula = model_formula, m = 2, maxit = 2)
 #See ?hmi_pool for how to pool results.

## End(Not run)
```

`hmi_pool` *Averages the results of the imputation function* `hmi`.

Description

This function applies the analysis the user wants to run on every imputed dataset. The results from every dataset are pooled by simply averaging. So the user has to make sure that averaging the analysis produces results is meaningful. Currently variance estimates for the averaged results are not implemented.

Usage

```
hmi_pool(mids, analysis_function)
```

Arguments

mids	A mids (multiply imputed data set) object. Either from the hmi imputation function or mice.
analysis_function	A user generated function that gets a completed data set, runs the model and returns all model parameters he or she is interested in in a vector. See examples below.

Value

A vector with all averaged results.

Examples

```
## Not run:
 data(Gcsemv, package = "hmi")

 model_formula <- written ~ 1 + gender + coursework + (1 + gender|school)

 set.seed(123)
 dat_imputed <- hmi(data = Gcsemv, model_formula = model_formula, m = 2, maxit = 2)

 my_analysis <- function(complete_data){
  # In this list, you can write all the parameters you are interested in.
  # Those will be averaged.
  # So make sure that averaging makes sense and that you only put in single numeric values.
  parameters_of_interest <- list()

  # ---- write in the following lines, what you are interetest in to do with your complete_data
  # the following lines are an example where the analyst is interested in the fixed intercept
  # and fixed slope and the random intercepts variance,
  # the random slopes variance and their covariance
  my_model <- lme4::lmer(model_formula, data = complete_data)

  parameters_of_interest[[1]] <- lme4::fixef(my_model)
  parameters_of_interest[[2]] <- lme4::VarCorr(my_model)[[1]][,]
  ret <- unlist(parameters_of_interest)# This line is essential if the elements of interest
  #should be labeled in the following line.
  names(ret) <-
   c("beta_intercept", "beta_gender", "beta_coursework", "sigma0", "sigma01", "sigma10", "sigma1")

  return(ret)
}
hmi_pool(mids = dat_imputed, analysis_function = my_analysis)
#if you are interested in fixed effects only, consider pool from mice:
mice::pool(with(data = dat_imputed,
  expr = lme4::lmer(written ~ 1 + gender + coursework + (1 + gender|school))))
```

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

`idf2interval` *Transform interval data frames into data.frames with interval variables*

Description

This function is the path from the linLIR package (Wiencierz, 2012) to this hmi package.

Usage

```
idf2interval(idf)
```

Arguments

`idf`	an interval data frame (idf-object).

Value

A `data.frame` where the interval variables are stored as `interval` objects.

`imputationcycle` *Cycling*

Description

Function to do one imputation cycle on the given data. The function cycles through every variable sequentially imputing the values, that are NA in the original data set in that current variable. The function determines the type of the variable and calls the suitable imputation function.

Usage

```
imputationcycle(
  data_before,
  original_data,
  NA_locator,
  fe,
  interaction_names,
  list_of_types,
  nitt,
  burnin,
  thin,
  pvalue = 0.2,
  mn,
  k = Inf,
  spike = NULL,
  rounding_degrees = NULL,
  rounding_covariates
)
```

Arguments

`data_before`	The n x p data.frame with the variables to impute. It was prepared for imputation in the `wrapper` function. The preparation includes the adding of intercept variables or interactions or the joining of small clusters.
`original_data`	The original data.frame the user passed to `hmi`.
`NA_locator`	A n x p matrix localizing the missing values in the original dataset. The elements are TRUE if the original data are missing and FALSE if the are observed.
`fe`	A list with the decomposed elements of the `model_formula`.
`interaction_names`	A list with the names of the variables that have been generated as interaction variables
`list_of_types`	a list where each list element has the name of a variable in the data.frame. The elements have to contain a single character denoting the type of the variable. See `get_type` for details about the variable types. With the function `list_of_types_maker`, the user can get the framework for this object. In most scenarios this is should not be necessary. One example where it might be necessary is when only two observations of a continuous variable are left - because in this case `get_type` interpret is variable to be binary. Wrong is it in no case.
`nitt`	An integer defining number of MCMC iterations (see `MCMCglmm`).
`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000), the `geweke.diag` might fail to detect convergence. In such cases it is essential to look a chain free from autocorrelation.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`mn`	An integer defining the minimum number of individuals per cluster.
`k`	An integer defining the allowed maximum of levels in a factor covariate.
`spike`	A numeric value saying which value in the semi-continuous data might be the spike. Or a list with with such values and names identical to the variables with spikes (see `list_of_spikes_maker` for details).
`rounding_degrees`	A numeric vector with the presumed rounding degrees. Or a list with rounding degrees, where each list element has the name of a rounded continuous variable. Such a list can be generated using `list_of_rounding_degrees_maker(data)`.

`rounding_covariates`	A list for each rounded continuous variable with a character vector containing the covariate names from the original rounding formula. The transformation takes place in the wrapper function.

Value

A data.frame where the values, that have a missing value in the original dataset, are imputed.

`imp_binary_multi` *The function for hierarchical imputation of binary variables.*

Description

The function is called by the wrapper.

Usage

```
imp_binary_multi(
  y_imp,
  X_imp,
  Z_imp,
  clID,
  nitt = 22000,
  burnin = 2000,
  thin = 20,
  pvalue = 0.2,
  k = Inf
)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000),
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A list with 1. 'y_ret' the n x 1 data.frame with the original and imputed values. 2. 'Sol' the Gibbs-samples for the fixed effects parameters. 3. 'VCV' the Gibbs-samples for variance parameters.

`imp_binary_single` *The function for imputation of binary variables.*

Description

The function is called by the wrapper.

Usage

```
imp_binary_single(y_imp, X_imp, pvalue = 0.2, k = Inf)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n x 1 data.frame with the original and imputed values.

`imp_cat_multi` *The function for hierarchical imputation of categorical variables.*

Description

The function is called by the wrapper and relies on `MCMCglmm`.
While in the single level function (`imp_cat_single`) we used regression trees to impute data, here we run a multilevel multinomial model. The basic idea is that for each category of the target variable (expect the reference category) an own formula is set up, saying for example that the chances to end up in category j increase with increasing X5. So there is an own regression coefficient $beta_{5,j}$ present. In a multilevel setting, this regression coefficient $beta_{5,j}$ might be different for different clusters: for cluster 27 it would be $beta_{5,j,27} = beta_{5,j} + u_{5,27}$. This also leads to own random effect covariance matrices for each category. All those random effect variance parameters can be collected in one (quite large) covariance matrix where (for example) not only the random intercepts variance and random slopes variance and their covariance is present. Instead, there is even a covariance between the random slopes in category s and the random intercepts in category p. Beside the difficulties in interpretation, these covariances have shown to be numerically instable so they are set to be 0.

Usage

```
imp_cat_multi(
  y_imp,
  X_imp,
  Z_imp,
  clID,
  nitt = 22000,
  burnin = 2000,
  thin = 20,
  pvalue = 0.2,
  k = Inf
)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000).
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A list with 1. 'y_ret' the n x 1 data.frame with the original and imputed values. 2. 'Sol' the Gibbs-samples for the fixed effects parameters. 3. 'VCV' the Gibbs-samples for variance parameters.

`imp_cat_single` *The function to impute unordered categorical variables*

Description

The function uses regression trees for imputation implemented in `mice`. The principle is the following: For each observation it is calculated at which leave it would end. Then one (randomly selected) observation of the other observations found on this leave functions as a donor.

Usage

```
imp_cat_single(y_imp, X_imp, pvalue = 0.2, k = Inf)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n x 1 data.frame with the original and imputed values.

`imp_cont_multi` *The function for hierarchical imputation of continuous variables.*

Description

The function is called by the wrapper.

Usage

```
imp_cont_multi(
  y_imp,
  X_imp,
  Z_imp,
  clID,
  nitt = 22000,
  burnin = 2000,
  thin = 20,
  pvalue = 0.2,
  k = Inf
)
```

Arguments

`y_imp`	A vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).

`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000).
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A list with 1. 'y_ret' the n x 1 data.frame with the original and imputed values. 2. 'Sol' the Gibbs-samples for the fixed effects parameters. 3. 'VCV' the Gibbs-samples for variance parameters.

`imp_cont_single` *The function for imputation of continuous variables.*

Description

The function is called by the wrapper (hmi). It uses `mice` with the method "norm".

Usage

```
imp_cont_single(y_imp, X_imp, pvalue = 0.2, k = Inf)
```

Arguments

`y_imp`	A vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n x 1 data.frame with the original and imputed values.

`imp_count_multi` *The function for hierarchical imputation of variables with count data.*

Description

The function is called by the wrapper.

Usage

```
imp_count_multi(
  y_imp,
  X_imp,
  Z_imp,
  clID,
  nitt = 22000,
  burnin = 2000,
  thin = 20,
  pvalue = 0.2,
  k = Inf
)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000).
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A list with 1. 'y_ret' the n x 1 data.frame with the original and imputed values. 2. 'Sol' the Gibbs-samples for the fixed effects parameters. 3. 'VCV' the Gibbs-samples for variance parameters.

imp_count_single *The function for imputation of binary variables.*

Description

The function is called by the wrapper.

Usage

```
imp_count_single(
  y_imp,
  X_imp,
  nitt = 22000,
  burnin = 2000,
  thin = 20,
  pvalue = 0.2,
  k = Inf
)
```

Arguments

`y_imp`	A vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000).
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A list with 1. 'y_ret' the n x 1 data.frame with the original and imputed values. 2. 'Sol' the Gibbs-samples for the fixed effects parameters. 3. 'VCV' the Gibbs-samples for variance parameters.

imp_interval *The function to impute interval data variables*

Description

This functions imputes interval data variables. Those are variables, that consists of a lower and upper (numeric) boundary. Technically those boundaries are contained in a string, separated by a semi colon. E.g. if a person reports there income to be something between 3000 and 4000 dollars, its value in the interval covariate would be `"3000;4000"`. Left (resp. right) censored data can be denoted by `"-Inf;x"` (resp. `"x;Inf"`), with `x` being the (numeric) observed value.

Usage

```
imp_interval(y_imp, X_imp, pvalue = 0.2, k = Inf)
```

Arguments

`y_imp`	A Vector from the class `interval` with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n x 1 data.frame with the original and imputed values. Note that this function won't return `interval` data as its purpose is to "break" the interval answers into precise answers.

imp_orderedcat_multi *The function for hierarchical imputation of categorical variables.*

Description

The function is called by the wrapper.

Usage

```
imp_orderedcat_multi(
  y_imp,
  X_imp,
  Z_imp,
  clID,
  nitt = 25000,
  burnin = 5000,
  thin = 20,
  pvalue = 0.2,
  k = Inf
)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000).
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A list with 1. 'y_ret' the n x 1 data.frame with the original and imputed values. 2. 'Sol' the Gibbs-samples for the fixed effects parameters. 3. 'VCV' the Gibbs-samples for variance parameters.

`imp_orderedcat_single` *The function to impute ordered categorical variables*

Description

The function uses the proportional odds logistic regression (polr) approach, implemented in `mice`.

Usage

```
imp_orderedcat_single(y_imp, X_imp, pvalue = 0.2, k = Inf)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n x 1 data.frame with the original and imputed values as a factor.

---

imp_roundedcont *The function to impute rounded continuous variables*

Description

For example the income in surveys is often reported rounded by the respondents. See Drechsler, Kiesl and Speidel (2015) for more details.

Usage

```
imp_roundedcont(
  y_df,
  X_imp,
  PSI,
  pvalue = 0.2,
  k = Inf,
  rounding_degrees = NULL
)
```

Arguments

`y_df`	A data.frame with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables explaining y_df.
`PSI`	A data.frame with the variables explaining the latent rounding tendency G.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.
`rounding_degrees`	A numeric vector with the presumed rounding degrees for Y.

Value

A n x 1 data.frame with the original and imputed values.

References

Joerg Drechsler, Hans Kiesl, Matthias Speidel (2015): "MI Double Feature: Multiple Imputation to Address Nonresponse and Rounding Errors in Income Questions". Austrian Journal of Statistics Vol. 44, No. 2, http://dx.doi.org/10.17713/ajs.v44i2.77

`imp_semicont_multi` *The function for hierarchical imputation of semicontinuous variables.*

Description

The function is called by the wrapper. We consider data to be "semicontinuous" when more than 5% of the (non categorical) observations.
For example in surveys a certain portion of people, when asked for their income, report "0", which clearly violates the assumption of income to be (log-) normally distributed.

Usage

```
imp_semicont_multi(
  y_imp,
  X_imp,
  Z_imp,
  clID,
  spike = NULL,
  nitt = 22000,
  burnin = 2000,
  thin = 20,
  pvalue = 0.2,
  k = Inf
)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`spike`	A numeric value saying to which values Y might be spiked
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`burnin`	burnin A numeric value between 0 and 1 for the desired percentage of Gibbs samples that shall be regarded as burnin.
`thin`	An integer to set the thinning interval range. If thin = 1, every iteration of the Gibbs-sampling chain will be kept. For highly autocorrelated chains, that are only examined by few iterations (say less than 1000).
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A list with 1. 'y_ret' the n x 1 data.frame with the original and imputed values. 2. 'Sol' the Gibbs-samples for the fixed effects parameters. 3. 'VCV' the Gibbs-samples for variance parameters.

`imp_semicont_single` *The function for hierarchical imputation of semicontinuous variables.*

Description

The function is called by the wrapper. We consider data to be "semicontinuous" when more than 5% of the (non categorical) observations.
For example in surveys a certain portion of people, when asked for their income, report "0", which clearly violates the assumption of income to be (log-) normally distributed.

Usage

```
imp_semicont_single(y_imp, X_imp, spike = NULL, pvalue = 0.2, k = Inf)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`spike`	A numeric value saying to which value Y might be spiked.
`pvalue`	A numeric between 0 and 1 denoting the threshold of p-values a variable in the imputation model should not exceed. If they do, they are excluded from the imputation model.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n x 1 data.frame with the original and imputed values.

`interval2idf` *Transform interval variables to an interval data frame*

Description

This function is the path from this hmi package to the linLIR package (Wiencierz, 2012).

Usage

```
interval2idf(interval)
```

Arguments

`interval`	an `interval`

Value

an interval data frame (idf-object) with one variable (having a lower and an upper bound).

`is.na.interval` *is.na for interval objects*

Description

This functions checks whether elements from an `interval` object are `NA`

Usage

```
## S3 method for class 'interval'
is.na(interval)
```

Arguments

`interval`	An `interval` object of length n

Value

A boolean vector of length n indicating whether the entries in `interval` are `NA` or not. Cf. `is.na`.

`is_interval` *Function to check whether an object is an interval*

Description

If there are numerics separated by semicolons (;), this is considered to be an interval. intervals with 2.4e5 are not considered to be an interval.

Usage

```
is_interval(x)
```

Arguments

`x`	an object

Value

a boolean value indicting whether x is an interval or not

`list_of_rounding_degrees_maker`
Helps the user to make a list of rounding degrees

Description

In `hmi` the user can add a list of rounding degrees. This function gives him a framework with suggestions. Of course the user can make changes by herself/himself afterwards. For example, the function might wrongly classify a variable to be heaped or selects unwanted rounding degrees.

Usage

```
list_of_rounding_degrees_maker(data)
```

Arguments

`data`	the data.frame also passed to `hmi`.

Value

a list with suggested rounding degrees. Each list element has the name of a rounded continuous variable in the data.frame. The elements contain a numeric vector with the rounding degrees found for that variable.

`list_of_rounding_formulas_maker`
Helps the user to make a list of rounding formulas for the rounding degrees

---

Description

In `hmi` the user can add a list of rounding formulas for each variable suffering rounding. This function gives him/her a framework with suggestions. Of course the user can make changes by herself/himself afterwards. For example, the function might wrongly classify a variable to be heaped.

Usage

```
list_of_rounding_formulas_maker(data, default = ~.)
```

Arguments

`data`	the data.frame also passed to `hmi`.
`default`	A default formula used for every rounded variable.

Value

a list with suggested rounding degree formulas. Each list element has the name of a rounded continuous variable in the data.frame. The elements contain a very general rounding degree formula.

`list_of_spikes_maker` *Helps the user to make a list of spikes.*

Description

In `hmi` the user can add a list of spikes. This function gives her/him a framework with suggestions. Of course the user can make changes by herself/himself afterwards. For example, the function might wrongly classify a variable to have a spike.

Usage

```
list_of_spikes_maker(data)
```

Arguments

`data`	the data.frame also passed to `hmi`.

Value

a list with suggested spikes. Each list element has the name of a spiked variable in the data.frame. The elements contain a single numeric denoting the spike found for that variable.

`list_of_types_maker` *Helps the user to make a list of types.*

Description

In `hmi` the user can add a list of types. This function gives him a framework with suggestions. Of course the user can make changes by herself/himself afterwards. For example, if a continuous variable has only two observations left, then get_type interpret this as a binary variable and not a continuous.

Usage

```
list_of_types_maker(data, spike = NULL, rounding_degrees = NULL)
```

Arguments

`data`	the data.frame also passed to `hmi`.
`spike`	A numeric value or list saying which value in the semi-continuous data might be the spike.
`rounding_degrees`	A numeric vector with the presumed rounding degrees.

Value

a list with suggested types. Each list element has the name of a variable in the data.frame. The elements contain a single character denoting the type of the variable. See `get_type` for details about the variable types.

`log.interval` *Log function for interval objects*

Description

Log function for interval objects

Usage

```
## S3 method for class 'interval'
log(x, ...)
```

Arguments

`x`	numeric vector or interval object
`...`	further arguments passed to `log`

`Mode` *Get the mode*

Description

This function calculates the mode (most frequent observation) of a vector.

Usage

```
Mode(x)
```

Arguments

`x`	A vector

Value

The mode of x as a numeric value.

References

Adopted from stackoverflow.com/questions/2547402: "is there a built in function for finding the mode" from user "Ken Williams".

`negloglik` *calculate the likelihood contribution of the data*

Description

This function based on Drechsler, Kiesl & Speidel (2015) is needed in the imputation routine for rounded income. It calculates the likelihood contribution of the data (regardless whether they are observed precisely or presumably rounded).

Usage

```
negloglik(
  para,
  parnames = names(para),
  X_in_negloglik,
  PSI_in_negloglik,
  y_precise_stand,
  lower_bounds = NA,
  upper_bounds = NA,
  my_g,
  sd_of_y_precise,
  indicator_precise,
  indicator_imprecise,
  indicator_outliers,
  rounding_degrees = c(1, 10, 100, 1000)
)
```

Arguments

`para`	This is the vector Psi of parameters (see p. 62 in Drechsler, Kiesl & Speidel, 2015). With respect to them, the value returned by negloglik shall be maximized. The starting values are c(kstart, betastart, gamma1start, sigmastart) (the thresholds (or "cutting points") for the latent variable behind the rounding degree, the regression parameters explaining the logged income, the regression parameters explaining the rounding degree and the variance parameter).
`parnames`	A character vector with the names of the elements in para.
`X_in_negloglik`	The data.frame of covariates explaining Y, the observed target variable. It has to has n rows (with n being the number of precise, imprecise and missing observations).
`PSI_in_negloglik`	The data.frame of covariates explaining G, the latent rounding tendency. Without the target variable.
`y_precise_stand`	A vector of the precise (and standardized) observations from the target variable.
`lower_bounds`	The lower bounds of an interval variable.

`upper_bounds`	The upper bounds of an interval variable.
`my_g`	This vector is the indicator of the (highest possible) rounding degree for an observation. This parameter comes directly from the data.
`sd_of_y_precise`	The scalar with the value equal to the standard deviation of the target variable.
`indicator_precise`	A boolean Vector indicating whether the value in the original target variable is precise (e.g. 5123 or 5123.643634) or not.
`indicator_imprecise`	A boolean Vector indicating whether the value in the original target variable is imprecise (e.g. "5120;5130) or not.
`indicator_outliers`	A boolean Vector indicating whether the value in the precise observations of the original target are outliers (smaller than 0.5% or larger than 99.5% of the other precise observations).
`rounding_degrees`	A numeric vector with the presumed rounding degrees for Y.

Value

An integer equal to the (sum of the) negative log-likelihood contributions (of the observations)

References

Joerg Drechsler, Hans Kiesl, Matthias Speidel (2015): "MI Double Feature: Multiple Imputation to Address Nonresponse and Rounding Errors in Income Questions", Austrian Journal of Statistics, Vol. 44, No. 2, http://dx.doi.org/10.17713/ajs.v44i2.77

`negloglik2_intervalsonly` *calculate the likelihood contribution of interval data only*

Description

calculate the likelihood contribution of interval data only

Usage

```
negloglik2_intervalsonly(
  para,
  parnames = names(para),
  X,
  lower_bounds,
  upper_bounds
)
```

Arguments

`para`	This is the vector Psi of parameters defining model (see p. 62 in Drechsler, Kiesl & Speidel, 2015). With respect to them, the value returned by this function shall be maximized. The starting values are c(betastart2, sigmastart2) (the regression parameters explaining the logged income, and the variance parameter).
`parnames`	A character vector with the names of the elements in para.
`X`	the data.frame of covariates.
`lower_bounds`	the lower bound of an interval variable.
`upper_bounds`	the upper bound of an interval variable.

Value

An integer equal to the (sum of the) negative log-likelihood contributions (of the observations)

References

Joerg Drechsler, Hans Kiesl, Matthias Speidel (2015): "MI Double Feature: Multiple Imputation to Address Nonresponse and Rounding Errors in Income Questions", Austrian Journal of Statistics, Vol. 44, No. 2, http://dx.doi.org/10.17713/ajs.v44i2.77

`nhanes_imp` *National Health and Nutrition Examination Survey (2015 - 2016) - imputed*

Description

A `mids` object with the imputed income data set from the US American National Health and Nutrition Examination Survey (NHANES) collected by the Centers for Disease Control and Prevention (CDC) and the National Center for Health Statistics (NCHS) for 2015-2016. The NHANES data are included into the package for illustration. The modified data set `nhanes_mod` was imputed by running `hmi(nhanes_mod,maxit = 50)`. The Website (https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.htm) gives the following Analytic notes: "The income questions were asked as part of household interview, the interview sample weights may be used in the analysis for data in this section. However, if the data is joined with other data from the Mobile Examination Center (MEC), the MEC exam weights should be used. Please refer to the NHANES Analytic Guidelines and the on-line NHANES Tutorial for further details on the use of sample weights and other analytic issues. Both of these are available on the NHANES website."

Usage

```
nhanes_imp
```

Format

A `mids` object. Each of the 3 completed data sets has 9971 rows and 12 variables:

inq020 Income from wages/salaries? 1 = Yes, 2 = No

inq012 Income from self employment? 1 = Yes, 2 = No

inq030 Income from Social Security or Railroad Retirement? 1 = Yes, 2 = No

inq060 Income from other disability pension? 1 = Yes, 2 = No

inq080 Income from retirement/survivor pension? 1 = Yes, 2 = No

inq090 Income from Supplemental Security Income? 1 = Yes, 2 = No

inq132 Income from state/county cash assistance? 1 = Yes, 2 = No

inq140 Income from interest/dividends or rental? 1 = Yes, 2 = No

inq150 Income from other sources? 1 = Yes, 2 = No

ind235 Monthly family income?

ind310 Total savings/cash assets for the family?

inq320 How do you get to the grocery store?, 1 = In my car, 2 = In a car that belongs to someone I live with, 3 = In a car that belongs to someone who lives elsewhere, 4 = Walk, 5 = Ride bicycle, 6 = Bus, subway or other public transit, 7 = Taxi or other paid driver, 8 = Someone else delivers groceries, 9 = Other, 66 = No usual mode of traveling to store, 77 = Refused, 99 = Don't know

Source

Website of the Centers for Disease Control and Prevention: `https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.XPT`

References

Centers for Disease Control and Prevention (CDC). National Center for Health Statistics (NCHS). National Health and Nutrition Examination Survey Data. Hyattsville, MD: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention. Variables descriptions at `https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.htm`

`nhanes_mod` *National Health and Nutrition Examination Survey (2015 - 2016) - modified*

Description

The Income data set from the US American National Health and Nutrition Examination Survey (NHANES) collected by the Centers for Disease Control and Prevention (CDC) and the National Center for Health Statistics (NCHS) for 2015-2016 - modified. The NHANES data are included into the package for illustration. The following modifications compared to `nhanes_sub` were made:

ind235 Monthly family income? Was made an interval object

ind310 Total savings/cash assets for the family? Was made an interval object

The Website (https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.htm) gives the following Analytic notes: "The income questions were asked as part of household interview, the interview sample weights may be used in the analysis for data in this section. However, if the data is joined with other data from the Mobile Examination Center (MEC), the MEC exam weights should be used. Please refer to the NHANES Analytic Guidelines and the on-line NHANES Tutorial for further details on the use of sample weights and other analytic issues. Both of these are available on the NHANES website."

Usage

```
nhanes_mod
```

Format

A data frame with 9971 rows and 12 variables:

inq020 Income from wages/salaries? 1 = Yes, 2 = No

inq012 Income from self employment? 1 = Yes, 2 = No

inq030 Income from Social Security or Railroad Retirement? 1 = Yes, 2 = No

inq060 Income from other disability pension? 1 = Yes, 2 = No

inq080 Income from retirement/survivor pension? 1 = Yes, 2 = No

inq090 Income from Supplemental Security Income? 1 = Yes, 2 = No

inq132 Income from state/county cash assistance? 1 = Yes, 2 = No

inq140 Income from interest/dividends or rental? 1 = Yes, 2 = No

inq150 Income from other sources? 1 = Yes, 2 = No

ind235 Monthly family income?

ind310 Total savings/cash assets for the family?

inq320 How do you get to the grocery store?, 1 = In my car, 2 = In a car that belongs to someone I live with, 3 = In a car that belongs to someone who lives elsewhere, 4 = Walk, 5 = Ride bicycle, 6 = Bus, subway or other public transit, 7 = Taxi or other paid driver, 8 = Someone else delivers groceries, 9 = Other, 66 = No usual mode of traveling to store, 77 = Refused, 99 = Don't know

Source

Website of the Centers for Disease Control and Prevention: https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.XPT

References

Centers for Disease Control and Prevention (CDC). National Center for Health Statistics (NCHS). National Health and Nutrition Examination Survey Data. Hyattsville, MD: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention. Variables descriptions at https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.htm

nhanes_sub — *A subset of the National Health and Nutrition Examination Survey (2015 - 2016)*

Description

A subset of the Income data set from the US American National Health and Nutrition Examination Survey (NHANES) collected by the Centers for Disease Control and Prevention (CDC) and the National Center for Health Statistics (NCHS) for 2015-2016. Four variables were dropped: `seqn`, the respondent sequence number, as the information was included into the rownames; `indfmmpi` and `indfmmpc`, the Family monthly poverty level index/category as these variables are too overcomplex for the illustrative purpose of the data and `inq300`, the response to the question whether the family has more than 20000 dollars of savings - this information was merged into the variable `ind310`, the total savings of the family. This gave several new categories for `ind310`: individuals reporting savings below 20,000 USD (`inq300 == 2`), but reporting "refused" or "don't know" to the detailed question of the total savings `ind310`, get the new category 6 "0 - 20,000 USD". The individuals reported to have more than 20,000 USD savings (`inq300 == 1`), get the new category 7 "20,001 USD and over". Individuals reporting "refused" or "don't know" to `inq300` (more than 20000 USD savings?) get the new category 8 "0 USD and over". In `ind235`, the monthly family income, missing values are made a new category "0 USD and over". The variable `inq320` was transformed into a factor. "Refused" and "Don't know" responses where changed to `NA`. The Website (`https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.htm`) gives the following Analytic notes: "The income questions were asked as part of household interview, the interview sample weights may be used in the analysis for data in this section. However, if the data is joined with other data from the Mobile Examination Center (MEC), the MEC exam weights should be used. Please refer to the NHANES Analytic Guidelines and the on-line NHANES Tutorial for further details on the use of sample weights and other analytic issues. Both of these are available on the NHANES website."

Usage

```
nhanes_sub
```

Format

A data frame with 9971 rows and 12 variables:

inq020 Income from wages/salaries? 1 = Yes, 2 = No

inq012 Income from self employment? 1 = Yes, 2 = No

inq030 Income from Social Security or Railroad Retirement? 1 = Yes, 2 = No

inq060 Income from other disability pension? 1 = Yes, 2 = No

inq080 Income from retirement/survivor pension? 1 = Yes, 2 = No

inq090 Income from Supplemental Security Income? 1 = Yes, 2 = No

inq132 Income from state/county cash assistance? 1 = Yes, 2 = No

inq140 Income from interest/dividends or rental? 1 = Yes, 2 = No

inq150 Income from other sources? 1 = Yes, 2 = No

ind235 Monthly family income? 1 = 0 - 399 USD, 2 = 400 - 799 USD, 3 = 800 - 1,249 USD, 4 = 1,250 - 1,649 USD, 5 = 1,650 - 2,099 USD, 6 = 2,100 - 2,899 USD, 7 = 2,900 - 3,749USD, 8 = 3,750 - 4,599USD, 9 = 4,600 - 5,399 USD, 10 = 5,400 - 6,249 USD, 11 = 6,250 - 8,399 USD, 12 = 8,400 USD and over, 13 = 0 USD and over

ind310 Total savings/cash assets for the family? 1 = 0 - 3,000 USD, 2 = 3,001 - 5,000 USD, 3 = 5,001 - 10,000, 4 = 10,001 - 15,000 USD, 5 = 15,001 - 20,000, 6 = 0 - 20,000 USD, 7 = 20,001 USD and over, 8 = 0 USD and over

inq320 How do you get to the grocery store?, 1 = In my car, 2 = In a car that belongs to someone I live with, 3 = In a car that belongs to someone who lives elsewhere, 4 = Walk, 5 = Ride bicycle, 6 = Bus, subway or other public transit, 7 = Taxi or other paid driver, 8 = Someone else delivers groceries, 9 = Other, 66 = No usual mode of traveling to store, 77 = Refused, 99 = Don't know

Source

Website of the Centers for Disease Control and Prevention: `https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.XPT`

References

Centers for Disease Control and Prevention (CDC). National Center for Health Statistics (NCHS). National Health and Nutrition Examination Survey Data. Hyattsville, MD: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention. Variables descriptions at `https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/INQ_I.htm`

`pbivnormX` *calculate probabilities from the cumulative distribution function of a standard bivariate normal distribution*

Description

A modified version of pbivnorm() from package `pbivnorm`. It is needed in the imputation routine for rounded income.

Usage

```
pbivnormX(x, y, rho = 0)
```

Arguments

`x`	the vector (or a two columned matrix) with the values of the first random variable
`y`	the vector with the values of the second random variable
`rho`	the correlation (a scalar) between the two random variables.

Value

A vector with the values of the density distribution at the points (x, y).

`plot.interval` *Plotting interval variables*

Description

Function to plot interval variables by rectangles. The bounds of the rectangles are given by the lower and upper bounds of the interval variables. To avoid precise observations to have a line-width of 0, small values are added to the upper and lower bounds what guarantees the rectangles (or lines or points) to be easily visible in the plot.

Usage

```
## S3 method for class 'interval'
plot(
  x = NULL,
  y = NULL,
  data = NULL,
  col = "black",
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  sort = NULL,
  ...
)
```

Arguments

`x`	In its most save way, `x` is an object from class `interval` and jointly used with a second `interval` object `y`. If no `y` is given, the values of `x` are just plotted in order of appearance (cf. `plot(iris$Sepal.Length)`). `x` can also be a `formula` with two variables found in `data`.
`y`	If used jointly with `x`, it has to be a numeric vector or an `interval` object.
`data`	If `x` is a `fomula`, it has to be a data.frame or matrix with column names fitting to the two variables named in the formula.
`col`	The color of the rectangles.
`xlab`	A title for the x axis: see `title`.
`ylab`	A title for the y axis: see `title`.
`xlim`	Numeric vectors of length 2, giving the x coordinate ranges.
`ylim`	Numeric vectors of length 2, giving the y coordinate ranges.
`sort`	A character specifying how the values should be sorted if only one variable is to be plotted. By default they are sorted according to their position in the data set. `sort = "lowerbound_increasing"` sorts the data primarily by their lower bound, and secondarily (this means for equal lower bounds) by their upper

	bounds. Both in increasing order. For `sort = "lowerbound_decreasing"`, both happens in decreasing order. `sort = "mostprecise_increasing"` sorts the data by their length of the interval they represent, and within equal lengths by the lower bound. Both in increasing order. For `sort = "mostprecise_decreasing"`, both happens in decreasing order.
`...`	graphical parameters such as `main`.

`random_intercept_check` *Function to check multilevel models on the existence of random intercepts*

Description

Function to check multilevel models on the existence of random intercepts. The specification of an intercept by calling a 1-column (e.g. "int") is not counted towards the existence of an intercept. Contradictory inputs like "~ 1 + 0 + X1 + ..." or "~ -1 + 1 + X1 + ..." will throw an error.

Usage

```
random_intercept_check(model_formula)
```

Arguments

`model_formula`	A formula (from class `formula`)

Value

A boolean value indicating whether there is a fixed intercept in the model or not

---

`round.interval` *Round function for interval objects*

Description

Round function for interval objects

Usage

```
## S3 method for class 'interval'
round(x, ...)
```

Arguments

`x`	numeric vector or interval object
`...`	further arguments passed to `round`.

`sampler` *Function need to multivariate samples of a truncated multivariate normal distribution*

Description

As rtmvnorm only allows one mean vector of one multivariate normal distribution, but we need different mean vectors for different multivariate normal distributions, we implement this function. This function in combination with `apply`, allows us to sample from a truncated multivariate normal distribution with different mean vectors.

Usage

```
sampler(elements, Sigma)
```

Arguments

`elements`	Originally a matrix, but when passed to samp, it is a vector. The first length_mean elements are the mean vector of g and y, the next two elements are the lower bounds for g and y, the last two elements are the upper bounds for g and y.
`Sigma`	The covariance matrix of the multivariate normal distribution to sample from.

Value

A length_mean x 1 matrix with the samples for g and y.

`sample_imp` *Sample imputation.*

Description

Function to sample values in a variable from other (observed) values in this variable. So this imputation does not use further covariates.

Usage

```
sample_imp(variable)
```

Arguments

`variable`	A vector of size `n` with missing values.

Value

A n x 1 data.frame with the observed and imputed data

Examples

```
set.seed(123)
sample_imp(c(1, NA, 3, NA, 5))
```

---

sna_interval *Get standard NAs from interval data*

Description

This function replaces observations with "-Inf;Inf" in an interval, which basically means "no information available", with the standard NAs (therefore the name 'sna'). Observations with a finite bound (e.g.x = "0;Inf") are not replaced, since they contain information (here: "x is positive").

Usage

```
sna_interval(x)
```

Arguments

x	can by any object, but the function was designed for `interval`-objects.

Value

In case of `x` being an `interval`-object, it returns a n times 2 matrix. The first column is the lower bound, the second the upper bound. Otherwise it returns just `x`.

---

split_interval *Split up intervals*

Description

This function splits an interval object up into the lower and upper bound

Usage

```
split_interval(interval)
```

Arguments

interval	an interval object of length n (if it is something else, it is returned unchanged)

Value

a n times 2 matrix. The first column is the lower bound, the second the upper bound.

`sqrt.interval` *Sqrt function for interval objects*

Description

Sqrt function for interval objects

Usage

```
## S3 method for class 'interval'
sqrt(x, ...)
```

Arguments

`x`	numeric vector or interval object
`...`	further arguments passed to `sqrt`.

`stand` *Standardizing function*

Description

Function to standardize variables that are numeric (continuous and count variables) but no rounded continuous, semicontinuous, intercepts or categorical variables.

Usage

```
stand(X)
```

Arguments

`X`	A n times p data.frame with p fixed (or random) effects variables.

Value

A n times p data.frame with the standardized versions of the numeric variables.

`suggest_rounding_degrees` *suggesting rounding degrees*

Description

A function that suggests some rounding degrees of a continuous variable (classically formatted or as interval object). The basic idea is 1. to count which factor is observed in the data more often than expected. 2. to check whether a factor can explain at least three observed piles of observations in the data 3. to check whether a factor explains at least 20 % of observations (additional to previous factors). Factors fulfilling this premises are returned as suggested rounding degrees.

Usage

```
suggest_rounding_degrees(x)
```

Arguments

`x`	A vector or `interval` object.

`table` *Tabulating interval objects*

Description

Function to tabulate interval objects

Usage

```
table(x, ...)

## S3 method for class 'interval'
table(x, sort = "lowerbound_increasing", ...)

## Default S3 method:
table(x, ...)
```

Arguments

`x`	In its most save way, `x` is an object from class `interval`.
`...`	Other parameters passed to `table`.

`sort`	A character specifying how the values should be sorted if only one variable is to be plotted. `sort = "lowerbound_increasing"` (the default) sorts the data primarily by their lower bound, and secondarily (this means for equal lower bounds) by their upper bounds. Both in increasing order. For `sort = "lowerbound_decreasing"` both happens in decreasing order. `sort = "mostprecise_increasing"` sorts the data by their length of the interval they represent, and within equal lengths by the lower bound. Both in increasing order. `sort = "mostprecise_decreasing"` both happens in decreasing order.

Value

A table.

`tail.interval` *Tail for intervals*

Description

Tail function for intervals returning the last elements of an `interval` object

Usage

```
## S3 method for class 'interval'
tail(x, ...)
```

Arguments

`x`	vector, matrix, table, data.frame or interval object
`...`	further arguments passed to `tail`.

`[.interval` *Index for interval*

Description

Function to index elements of an interval object

Usage

```
## S3 method for class 'interval'
obj[index]
```

Arguments

`obj`	the interval object
`index`	the index of the elements to replace

`[<-.interval` *Replace for interval*

Description

Function to replace elements of an interval object

Usage

```
## S3 replacement method for class 'interval'
obj[index] <- value
```

Arguments

`obj`	the interval object
`index`	the index of the elements to replace
`value`	the value the replaced elements shall take

`%%.interval` *Modulo function*

Description

Modulo function for interval objects

Usage

```
## S3 method for class 'interval'
x %% interval
```

Arguments

`x`	an single element or vector to add to the interval object
`interval`	an object from class interval

Value

a vector with the modulo for the precise elements in `interval`. For imprecise elements, `NA` is returned.

`^.interval` *Power function*

Description

Taking the power of interval objects

Usage

```
## S3 method for class 'interval'
interval ^ x
```

Arguments

`interval`	an object from class interval
`x`	an single numeric to potentiate

Value

an interval object

Index