Package: rMIDAS (via r-universe)

November 11, 2026

Title Multiple Imputation with Denoising Autoencoders

Version 1.0.1

Description A tool for multiply imputing missing data using 'MIDAS', a deep learning method based on denoising autoencoder neural networks (see Lall and Robinson, 2022; <doi:10.1017/pan.2020.49>). This algorithm offers significant accuracy and efficiency advantages over other multiple imputation strategies, particularly when applied to large datasets with complex features. Alongside interfacing with 'Python' to run the core algorithm, this package contains functions for processing data before and after model training, running imputation model diagnostics, generating multiple completed datasets, and estimating regression models on these datasets. For more information see Lall and Robinson (2023) <doi:10.18637/jss.v107.i09>. This package is deprecated in favor of 'rMIDAS2'; it remains available for existing workflows but will receive only compatibility and documentation updates.

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.6.0), data.table, mltools, reticulate

Imports rappdirs, Rdpack

Suggests testthat, knitr, rmarkdown

RdMacros Rdpack

SystemRequirements Python (>= 3.6.0)

VignetteBuilder knitr

License Apache License (>= 2.0)

URL https://github.com/MIDASverse/rMIDAS, https://CRAN.R-project.org/package=rMIDAS2, https://github.com/MIDASverse/rMIDAS2

BugReports https://github.com/MIDASverse/rMIDAS/issues

NeedsCompilation no

Author Thomas Robinson [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-7097-1599>), Ranjit Lall [aut, cph] (ORCID: <https://orcid.org/0000-0003-1455-3506>), Alex Stenlake [ctb, cph], Elviss Dvinskis [ctb]

Maintainer Thomas Robinson <ts.robinson1994@gmail.com>

Config/pak/sysreqs libpng-dev python3

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

Date/Publication 2026-11-11 03:03:00 UTC

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/rMIDAS

RemoteSha 9bc44acc9e89385e85e4359378b98d7356d804c4

RemoteSubdir rMIDAS

Contents

rMIDAS-package . . . 3
add_bin_labels . . . 4
add_missingness . . . 4
coalesce_one_hot . . . 5
col_minmax . . . 6
combine . . . 6
complete . . . 7
convert . . . 9
delete_rMIDAS_env . . . 10
import_midas . . . 10
mid_py_setup . . . 11
midas_setup . . . 11
na_to_nan . . . 12
overimpute . . . 12
python_configured . . . 16
python_init . . . 16
reset_rMIDAS_env . . . 16
set_python_env . . . 17
skip_if_no_numpy . . . 17
train . . . 18
undo_minmax . . . 20

Index **22**

rMIDAS-package *rMIDAS: Multiple Imputation with Denoising Autoencoders*

Description

rMIDAS provides an R interface to the MIDAS multiple imputation algorithm.

Details

rMIDAS is deprecated in favor of rMIDAS2, which provides the actively maintained successor interface and backend. The source repository for the successor package is https://github.com/MIDASverse/rMIDAS2. rMIDAS remains available on CRAN for existing workflows but will receive only compatibility and documentation updates.

See `vignette("migrating-to-rMIDAS2", package = "rMIDAS")` for guidance on moving to rMIDAS2.

Author(s)

Maintainer: Thomas Robinson <ts.robinson1994@gmail.com> (ORCID) [copyright holder]

Authors:

- Ranjit Lall <R.Lall@lse.ac.uk> (ORCID) [copyright holder]

Other contributors:

- Alex Stenlake [contributor, copyright holder]
- Elviss Dvinskis [contributor]

See Also

Useful links:

- https://github.com/MIDASverse/rMIDAS
- https://CRAN.R-project.org/package=rMIDAS2
- https://github.com/MIDASverse/rMIDAS2
- Report bugs at https://github.com/MIDASverse/rMIDAS/issues

add_bin_labels *Reverse numeric conversion of binary vector*

Description

Helper function to re-apply binary variable labels post-imputation.

Usage

```
add_bin_labels(x, one, zero, fast = TRUE)
```

Arguments

x	A numeric vector or column, scaled between 0 and 1
one	A character string, the label associated with binary value 1
zero	A character string, the label associated with binary value 0
fast	Boolean indicating whether to return binary value 1 if predicted probability >= 0.5 (TRUE), or take random draw using predicted probability as weighting.

Value

Vector of character strings corresponding to binary values

Examples

```
ex_bin <- c(1,0,0,1,1,0,0,1,0)
cat <- "cat"
dog <- "dog"

add_bin_labels(x = ex_bin, one = cat, zero = dog)
```

add_missingness *Apply MAR missingness to data*

Description

Helper function to add missing values to data.

Usage

```
add_missingness(X, prop, cols = NULL)
```

Arguments

X	A data.frame or similar
prop	Numeric value between 0 and 1; the proportion of observations set to missing
cols	A vector of column names to be corrupted; if NULL, all columns are used

Value

Data with missing values

Examples

```
whole_data <- data.frame(a = rnorm(1000),
                         b = rnorm(1000))

missing_data <- add_missingness(whole_data, 0.1)
```

---

coalesce_one_hot *Coalesce one-hot encoding back to a single variable*

Description

Helper function to reverse one-hot encoding post-imputation.

Usage

```
coalesce_one_hot(X, var_name, fast = TRUE)
```

Arguments

X	A data.frame, data.table or matrix, for a single variable
var_name	A character string, with the original variable label
fast	Boolean, indicating whether to choose category with highest predicted probability (TRUE), or use predicted probabilities as weights in draw from random distribution

Value

A vector of length equal to `nrow(X)`, containing categorical labels corresponding to the columns of `X`

col_minmax *Scale numeric vector between 0 and 1*

Description

Helper function to scale numeric variables. Aids convergence of Midas model.

Usage

```
col_minmax(x)
```

Arguments

x	A numeric vector or column.

Value

Vector scaled between 0 and 1

Examples

```
ex_num <- runif(100,1,10)
scaled <- col_minmax(ex_num)
```

combine *Estimate and combine regression models from multiply-imputed data*

Description

`combine()` calculates *m* individual regression models, then applies "Rubin's Rules" to produce a single, combined estimate of the regression parameters and uncertainty.

Usage

```
combine(formula, df_list, dof_adjust = TRUE, ...)
```

Arguments

formula	A formula, or character string coercible to a formula
df_list	A list, containing data.frames or objects coercible to data.frames
dof_adjust	Boolean, indicating whether or not to apply the Rubin and Barnard (1999) degrees of freedom adjustment for small-samples
...	Further arguments passed onto `glm()`

Value

Data.frame of combined model results.

Examples

```
set.seed(89)
test_dfs <- lapply(1:5, function (x) data.frame(a = rnorm(1000),
                                                b = runif(1000),
                                                c = 2*rnorm(1000)))

midas_res <- combine("a ~ b + c", df_list = test_dfs)
```

`complete` *Impute missing values using imputation model*

Description

Having trained an imputation model, complete() produces `m` completed datasets, saved as a list.

Usage

```
complete(
  mid_obj,
  m = 10L,
  unscale = TRUE,
  bin_label = TRUE,
  cat_coalesce = TRUE,
  fast = FALSE,
  file = NULL,
  file_root = NULL
)
```

Arguments

`mid_obj`	Object of class `midas`, the result of running `rMIDAS::train()`
`m`	An integer, the number of completed datasets required
`unscale`	Boolean, indicating whether to unscale any columns that were previously min-max scaled between 0 and 1
`bin_label`	Boolean, indicating whether to add back labels for binary columns
`cat_coalesce`	Boolean, indicating whether to decode the one-hot encoded categorical variables
`fast`	Boolean, indicating whether to impute category with highest predicted probability (TRUE), or to use predicted probabilities to make weighted sample of category levels (FALSE)
`file`	Path to save completed datasets. If `NULL`, completed datasets are only loaded into memory.
`file_root`	A character string, used as the root for all filenames when saving completed datasets if a `filepath` is supplied. If no file_root is provided, completed datasets will be saved as "file/midas_impute_yymmdd_hhmmss_m.csv"

Details

For more information, see Lall and Robinson (2023): doi:10.18637/jss.v107.i09.

Value

List of length m, each element of which is a completed data.frame (i.e. no missing values)

References

Lall R, Robinson T (2023). "Efficient Multiple Imputation for Diverse Data in Python and R: MIDASpy and rMIDAS." *Journal of Statistical Software*, **107**(9), 1–38. doi:10.18637/jss.v107.i09.

Examples

```
# Generate raw data, with numeric, binary, and categorical variables
## Not run:
# Run where Python available and configured correctly
if (python_configured()) {
set.seed(89)
n_obs <- 10000
raw_data <- data.table(a = sample(c("red","yellow","blue",NA),n_obs, replace = TRUE),
                       b = 1:n_obs,
                       c = sample(c("YES","NO",NA),n_obs,replace=TRUE),
                       d = runif(n_obs,1,10),
                       e = sample(c("YES","NO"), n_obs, replace = TRUE),
                  f = sample(c("male","female","trans","other",NA), n_obs, replace = TRUE))

# Names of bin./cat. variables
test_bin <- c("c","e")
test_cat <- c("a","f")

# Pre-process data
test_data <- convert(raw_data,
                     bin_cols = test_bin,
                     cat_cols = test_cat,
                     minmax_scale = TRUE)

# Run imputations
test_imp <- train(test_data)

# Generate datasets
complete_datasets <- complete(test_imp, m = 5, fast = FALSE)

# Use Rubin's rules to combine m regression models
midas_pool <- combine(formula = d~a+c+e+f,
                      complete_datasets)
}

## End(Not run)
```

convert *Pre-process data for Midas imputation*

Description

`convert` pre-processes datasets to enable user-friendly interface with the main `train()` function.

Usage

```
convert(data, bin_cols = NULL, cat_cols = NULL, minmax_scale = FALSE)
```

Arguments

`data`	Either an object of class `data.frame`, `data.table`, or a path to a regular, delimited file
`bin_cols, cat_cols`	A vector, column names corresponding to binary and categorical variables respectively
`minmax_scale`	Boolean, indicating whether to scale all numeric columns between 0 and 1, to improve model convergence

Details

The function has two advantages over manual pre-processing:

1. Utilises data.table for fast read-in and processing of large datasets
2. Outputs an object that can be passed directly to `train()` without re-specifying column names etc.

For more information, see Lall and Robinson (2023): doi:10.18637/jss.v107.i09.

Value

Returns custom S3 object of class 'midas_preproc' containing:

- `data` – processed version of input data,
- `bin_list` – vector of binary variable names
- `cat_lists` – embedded list of one-hot encoded categorical variable names
- `minmax_params` – list of min. and max. values for each numeric object scaled

List containing converted data, categorical and binary labels to be imported into the imputation model, and scaling parameters for post-imputation transformations.

References

Lall R, Robinson T (2023). "Efficient Multiple Imputation for Diverse Data in Python and R: MIDASpy and rMIDAS." *Journal of Statistical Software*, **107**(9), 1–38. doi:10.18637/jss.v107.i09.

Examples

```
data = data.frame(a = sample(c("red","yellow","blue",NA),100, replace = TRUE),
                  b = 1:100,
                  c = sample(c("YES","NO",NA),100,replace = TRUE),
                  d = runif(100),
                  e = sample(c("YES","NO"), 100, replace = TRUE),
                 f = sample(c("male","female","trans","other",NA), 100, replace = TRUE),
                  stringsAsFactors = FALSE)

bin <- c("c","e")
cat <- c("a","f")

convert(data, bin_cols = bin, cat_cols = cat)
```

---

delete_rMIDAS_env *Delete the rMIDAS Environment and Configuration*

Description

Deletes both the virtual environment and the configuration file for the rMIDAS package. After deletion, it is necessary to restart the R session and then load the rMIDAS package once more. This will trigger the setup process again.

Usage

```
delete_rMIDAS_env()
```

Value

A message indicating the completion of the deletion process.

---

import_midas *Instantiate Midas class*

Description

Import Midas class into R environment, and instantiates passed parameters.

Usage

```
import_midas(...)
```

Arguments

...	Arguments passed to the MIDAS class for instantiating network

Value

Object of class 'midas'

mid_py_setup *Configure python for MIDAS imputation*

Description

This helper function checks if the required Python dependencies are installed, and if not, checks with user before installing them. Users should not call this function directly. Users can set a custom python install using `set_python_env()` so long as this is done prior to the first call to `train()` or `complete()`.

Usage

```
mid_py_setup()
```

midas_setup *Manually set up Python connection*

Description

This function allows users to initialise a custom Python configuration to run MIDAS, having manually set a Python version using `reticulate::use_python`, `reticulate::use_virtualenv`, `reticulate::use_condaenv`, or `reticulate::use_miniconda`.

Usage

```
midas_setup()
```

Note

This function is primarily for users who wish to have complete control over configuring Python versions and environments.

This function call is **not** required if users either use the `rMIDAS::set_python_env()` function or leave settings at their default.

If users set a custom binary/environment, this must be completed prior to the first call to either `train()` or `complete()`.

na_to_nan *Replace NA missing values with NaN*

Description

Helper function to convert NA values in a data.frame to NaN. This ensures the correct conversion of missing values when reticulate converts R objects to their Python equivalent. See the reticulate package documentation on type conversions for more information.

Usage

```
na_to_nan(df)
```

Arguments

df	Data frame, or object coercible to one.

Value

Data frame with NA values replaced with NaN values.

Examples

```
na_to_nan(data.frame(a = c(1,NA,0,0,NA,NA)))
```

overimpute *Perform overimputation diagnostic test*

Description

overimpute() spikes additional missingness into the input data and reports imputation accuracy at training intervals specified by the user. overimpute() works like train() – users must specify input data, binary and categorical columns (if data is not generated via convert(), model parameters for the neural network, and then overimputation parameters (see below for full details).

Usage

```
overimpute(
  data,
  binary_columns = NULL,
  softmax_columns = NULL,
  spikein = 0.3,
  training_epochs,
  report_ival = 35,
  plot_vars = FALSE,
  skip_plot = FALSE,
```

```
  spike_seed = NULL,
  save_path = "",
  layer_structure = c(256, 256, 256),
  learn_rate = 4e-04,
  input_drop = 0.8,
  seed = 123L,
  train_batch = 16L,
  latent_space_size = 4,
  cont_adj = 1,
  binary_adj = 1,
  softmax_adj = 1,
  dropout_level = 0.5,
  vae_layer = FALSE,
  vae_alpha = 1,
  vae_sample_var = 1
)
```

Arguments

`data`	A data.frame (or coercible) object, or an object of class `midas_pre` created from rMIDAS::convert()
`binary_columns`	A vector of column names, containing binary variables. NOTE: if `data` is a `midas_pre` object, this argument will be overwritten.
`softmax_columns`	A list of lists, each internal list corresponding to a single categorical variable and containing names of the one-hot encoded variable names. NOTE: if `data` is a `midas_pre` object, this argument will be overwritten.
`spikein`	A numeric between 0 and 1; the proportion of observed values in the input dataset to be randomly removed.
`training_epochs`	An integer, specifying the number of overimputation training epochs.
`report_ival`	An integer, specifying the number of overimputation training epochs between calculations of loss. Shorter intervals provide a more granular view of model performance but slow down the overimputation process.
`plot_vars`	Boolean, specifies whether to plot the distribution of original versus overimputed values. This takes the form of a density plot for continuous variables and a barplot for categorical variables (showing proportions of each class).
`skip_plot`	Boolean, specifies whether to suppress the main graphical output. This may be desirable when users are conducting a series of overimputation exercises and are primarily interested in the console output. **Note**, when `skip_plot = FALSE`, users must manually close the resulting pyplot window before the code will terminate.
`spike_seed, seed`	An integer, to initialize the pseudo-random number generators. Separate seeds can be provided for the spiked-in missingness and imputation, otherwise `spike_seed` is set to `seed` (default = 123L).

`save_path` String, indicating path to directory to save overimputation figures. Users should include a trailing "/" at the end of the path i.e. save_path = "path/to/figures/".

`layer_structure` A vector of integers, The number of nodes in each layer of the network (default = `c(256, 256, 256)`, denoting a three-layer network with 256 nodes per layer). Larger networks can learn more complex data structures but require longer training and are more prone to overfitting.

`learn_rate` A number, the learning rate γ (default = 0.0001), which controls the size of the weight adjustment in each training epoch. In general, higher values reduce training time at the expense of less accurate results.

`input_drop` A number between 0 and 1. The probability of corruption for input columns in training mini-batches (default = 0.8). Higher values increase training time but reduce the risk of overfitting. In our experience, values between 0.7 and 0.95 deliver the best performance.

`train_batch` An integer, the number of observations in training mini-batches (default = 16).

`latent_space_size` An integer, the number of normal dimensions used to parameterize the latent space.

`cont_adj` A number, weights the importance of continuous variables in the loss function

`binary_adj` A number, weights the importance of binary variables in the loss function

`softmax_adj` A number, weights the importance of categorical variables in the loss function

`dropout_level` A number between 0 and 1, determines the number of nodes dropped to "thin" the network

`vae_layer` Boolean, specifies whether to include a variational autoencoder layer in the network

`vae_alpha` A number, the strength of the prior imposed on the Kullback-Leibler divergence term in the variational autoencoder loss functions.

`vae_sample_var` A number, the sampling variance of the normal distributions used to parameterize the latent space.

Details

Accuracy is measured as the RMSE of imputed values versus actual values for continuous variables and classification error for categorical variables (i.e., the fraction of correctly predicted classes subtracted from 1). Both metrics are reported in two forms:

1. their summed value over all Monte Carlo samples from the estimated missing-data posterior – "Aggregated RMSE" and "Aggregated softmax error";
2. their aggregated value divided by the number of such samples – "Individual RMSE" and "Individual softmax error".

In the final model, we recommend selecting the number of training epochs that minimizes the average value of these metrics — weighted by the proportion (or substantive importance) of continuous and categorical variables — in the overimputation exercise. This “early stopping” rule reduces the risk of overtraining and thus, in effect, serves as an extra layer of regularization in the network.

For more information, see Lall and Robinson (2023): doi:10.18637/jss.v107.i09.

Value

Object of class `midas`, and outputs both overimputation loss values to the console and generates overimputation graphs.

References

Lall R, Robinson T (2023). "Efficient Multiple Imputation for Diverse Data in Python and R: MIDASpy and rMIDAS." *Journal of Statistical Software*, **107**(9), 1–38. doi:10.18637/jss.v107.i09.

See Also

`train` for the main imputation function.

Examples

```
## Not run:
# Run where Python initialised and configured correctly
if (python_configured()) {

raw_data <- data.table(a = sample(c("red","yellow","blue",NA),1000, replace = TRUE),
                       b = 1:1000,
                       c = sample(c("YES","NO",NA),1000,replace=TRUE),
                       d = runif(1000,1,10),
                       e = sample(c("YES","NO"), 1000, replace = TRUE),
                  f = sample(c("male","female","trans","other",NA), 1000, replace = TRUE))

# Names of bin./cat. variables
test_bin <- c("c","e")
test_cat <- c("a","f")

# Pre-process data
test_data <- convert(raw_data,
                     bin_cols = test_bin,
                     cat_cols = test_cat,
                     minmax_scale = TRUE)

# Overimpute - without plots
test_imp <- overimpute(test_data,
                     spikein = 0.3,
                     plot_vars = FALSE,
                     skip_plot = TRUE,
                     training_epochs = 10,
                     report_ival = 5)
}

## End(Not run)
```

python_configured *Check whether Python is capable of executing example code*

Description

Checks if each Python dependency is available. This function is called within some examples to ensure code executes properly.

Usage

```
python_configured()
```

Value

NULL

python_init *Initialise connection to Python*

Description

Internal function. Checks if Python has already been initialised, and if not, completes the required setup to run the MIDAS algorithm. This function is called automatically, and users should not call it directly. To configure which Python install/environment/conda is used, see documentation for `set_python_env()`.

Usage

```
python_init()
```

reset_rMIDAS_env *Reset the rMIDAS Environment Configuration*

Description

Resets the configuration for the rMIDAS package by deleting the configuration file. Once the configuration is reset, it is necessary to restart the R session and then load the rMIDAS package once more.

Usage

```
reset_rMIDAS_env()
```

Value

A message indicating the completion of the reset process.

set_python_env *Manually select python binary*

Description

This function allows users to set a custom python binary, virtualenv or conda environment, from which the MIDAS algorithm is run. Users comfortable with reticulate can configure Python manually using `reticulate::use_`. Note: If users wish to set a custom binary/environment, this must be completed prior to the first call to either `train()` or `complete()`. The same is true if users use the reticulate package directly. If users wish to switch to a different Python binaries, R must be restarted prior to calling this function.

Usage

```
set_python_env(x, type = "auto", ...)
```

Arguments

`x`	Character string, path to python binary, or directory of virtualenv, or name of conda environment
`type`	Character string, specifies whether to set a python binary ("auto"), "virtualenv", or "conda"
`...`	Further arguments passed to `reticulate::use_condaenv()`

Value

Boolean indicating whether the custom python environment was activated.

skip_if_no_numpy *Skip test where 'numpy' not available.*

Description

Check if Python's numpy is available, and skip test if not. This function is called within some tests to ensure server tests involving `reticulate` calls execute properly.

Usage

```
skip_if_no_numpy()
```

Value

`NULL`

`train` *Train an imputation model using Midas*

Description

Build and run a MIDAS neural network on the supplied missing data.

Usage

```
train(
  data,
  binary_columns = NULL,
  softmax_columns = NULL,
  training_epochs = 10L,
  layer_structure = c(256, 256, 256),
  learn_rate = 4e-04,
  input_drop = 0.8,
  seed = 123L,
  train_batch = 16L,
  latent_space_size = 4,
  cont_adj = 1,
  binary_adj = 1,
  softmax_adj = 1,
  dropout_level = 0.5,
  vae_layer = FALSE,
  vae_alpha = 1,
  vae_sample_var = 1
)
```

Arguments

`data`	A data.frame (or coercible) object, or an object of class `midas_pre` created from rMIDAS::convert()
`binary_columns`	A vector of column names, containing binary variables. NOTE: if `data` is a `midas_pre` object, this argument will be overwritten.
`softmax_columns`	A list of lists, each internal list corresponding to a single categorical variable and containing names of the one-hot encoded variable names. NOTE: if `data` is a `midas_pre` object, this argument will be overwritten.
`training_epochs`	An integer, indicating the number of forward passes to conduct when running the model.
`layer_structure`	A vector of integers, The number of nodes in each layer of the network (default = `c(256, 256, 256)`, denoting a three-layer network with 256 nodes per layer). Larger networks can learn more complex data structures but require longer training and are more prone to overfitting.

`learn_rate` A number, the learning rate γ (default = 0.0001), which controls the size of the weight adjustment in each training epoch. In general, higher values reduce training time at the expense of less accurate results.

`input_drop` A number between 0 and 1. The probability of corruption for input columns in training mini-batches (default = 0.8). Higher values increase training time but reduce the risk of overfitting. In our experience, values between 0.7 and 0.95 deliver the best performance.

`seed` An integer, the value to which Python's pseudo-random number generator is initialized. This enables users to ensure that data shuffling, weight and bias initialization, and missingness indicator vectors are reproducible.

`train_batch` An integer, the number of observations in training mini-batches (default = 16).

`latent_space_size` An integer, the number of normal dimensions used to parameterize the latent space.

`cont_adj` A number, weights the importance of continuous variables in the loss function

`binary_adj` A number, weights the importance of binary variables in the loss function

`softmax_adj` A number, weights the importance of categorical variables in the loss function

`dropout_level` A number between 0 and 1, determines the number of nodes dropped to "thin" the network

`vae_layer` Boolean, specifies whether to include a variational autoencoder layer in the network

`vae_alpha` A number, the strength of the prior imposed on the Kullback-Leibler divergence term in the variational autoencoder loss functions.

`vae_sample_var` A number, the sampling variance of the normal distributions used to parameterize the latent space.

Details

For more information, see Lall and Robinson (2023): doi:10.18637/jss.v107.i09.

Value

Object of class `midas` from which completed datasets can be drawn, using `rMIDAS::complete()`

References

Lall R, Robinson T (2023). "Efficient Multiple Imputation for Diverse Data in Python and R: MIDASpy and rMIDAS." *Journal of Statistical Software*, **107**(9), 1–38. doi:10.18637/jss.v107.i09.

Examples

```
# Generate raw data, with numeric, binary, and categorical variables
## Not run:
# Run where Python available and configured correctly
if (python_configured()) {
set.seed(89)
n_obs <- 10000
```

```
raw_data <- data.table(a = sample(c("red","yellow","blue",NA),n_obs, replace = TRUE),
                       b = 1:n_obs,
                       c = sample(c("YES","NO",NA),n_obs,replace=TRUE),
                       d = runif(n_obs,1,10),
                       e = sample(c("YES","NO"), n_obs, replace = TRUE),
                  f = sample(c("male","female","trans","other",NA), n_obs, replace = TRUE))

# Names of bin./cat. variables
test_bin <- c("c","e")
test_cat <- c("a","f")

# Pre-process data
test_data <- convert(raw_data,
                     bin_cols = test_bin,
                     cat_cols = test_cat,
                     minmax_scale = TRUE)

# Run imputations
test_imp <- train(test_data)

# Generate datasets
complete_datasets <- complete(test_imp, m = 5, fast = FALSE)

# Use Rubin's rules to combine m regression models
midas_pool <- combine(formula = d~a+c+e+f,
                      complete_datasets)
}

## End(Not run)
```

---

undo_minmax *Reverse minmax scaling of numeric vector*

Description

Helper function to reverse minmax scaling applied in the pre-processing step.

Usage

```
undo_minmax(s, s_min, s_max)
```

Arguments

s	A numeric vector or column, scaled between 0 and 1.
s_min	A numeric value, the minimum of the unscaled vector
s_max	A numeric value, the maximum of the unscaled vector

Value

Vector re-scaled using original parameters `s_min` and `s_max`

Examples

```
ex_num <- runif(100,1,10)
scaled <- col_minmax(ex_num)
undo_scale <- undo_minmax(scaled, s_min = min(ex_num), s_max = max(ex_num))

# Prove two are identical
all.equal(ex_num, undo_scale)
```

Index

∗ **diagnostics**
  overimpute, 12
∗ **import**
  combine, 6
∗ **imputation**
  complete, 7
  import_midas, 10
  train, 18
∗ **package**
  rMIDAS-package, 3
∗ **postprocessing**
  add_bin_labels, 4
  coalesce_one_hot, 5
  undo_minmax, 20
∗ **preprocessing**
  add_missingness, 4
  col_minmax, 6
  convert, 9
  na_to_nan, 12
∗ **setup**
  mid_py_setup, 11
  midas_setup, 11
  python_configured, 16
  python_init, 16
  set_python_env, 17
  skip_if_no_numpy, 17

add_bin_labels, 4
add_missingness, 4

coalesce_one_hot, 5
col_minmax, 6
combine, 6
complete, 7
convert, 9

delete_rMIDAS_env, 10

import_midas, 10

mid_py_setup, 11
midas_setup, 11

na_to_nan, 12

overimpute, 12

python_configured, 16
python_init, 16

reset_rMIDAS_env, 16
rMIDAS *(*rMIDAS-package*)*, 3
rMIDAS-package, 3

set_python_env, 17
skip_if_no_numpy, 17

train, *15*, 18

undo_minmax, 20