Package 'mimar'

November 5, 2026

Title Compact Multiple Imputation, Assessment, and Reporting

Version 1.4.2

Author Imad El Badisy [aut, cre]

Maintainer Imad El Badisy <elbadisyimad@gmail.com>

Description Provides compact tools for missing-data analysis, including artificial amputation, chained single and multiple imputation, statistical and machine-learning-based imputation methods, diagnostic evaluation, and post-imputation pooling.

License MIT + file LICENSE

URL https://CRAN.R-project.org/package=mimar

BugReports https://github.com/ielbadisy/mimar/issues

Encoding UTF-8

RoxygenNote 7.3.3

Imports BART, basetable (>= 1.4.2), densemlp, e1071, fastgbm, functionals, ggplot2, glmnet, missknn, missMDA, naivebayes, ranger, rpart, stats, xgboost

Suggests testthat (>= 3.0.0), knitr, rmarkdown, pkgload, survival, panglm (>= 1.1.4)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2026-11-04 23:40:19 UTC

Contents

ampute.data.frame 2
complete 3
describe 4
evaluate 5

fit 5
impute 7
imputer 9
imputer_registry 10
ipcw 10
nelsonaalen 12
plot.mimar_imputation 13
pool 14
pool_clogit 19
pool_coxph 20
pool_glm 21
pool_lm 22
pool_panglm 23
pool_survmat 24
pool_survreg 26
pool_test 27

Index **30**

`ampute.data.frame` *Artificially introduce missing data*

Description

`ampute()` creates benchmark data by adding missing values under MCAR, MAR, or MNAR mechanisms. The returned object keeps the original data, amputated data, and masks for original, added, and total missingness.

Usage

```
## S3 method for class 'data.frame'
ampute(
  x,
  prop = 0.1,
  mechanism = c("MCAR", "MAR", "MNAR"),
  target = NULL,
  by = NULL,
  direction = c("both", "left", "right"),
  seed = NULL,
  ...
)

ampute(x, ...)
```

Arguments

x	A data frame.
prop	Target marginal proportion of newly introduced missing values.
mechanism	Missingness mechanism: missing completely at random ("MCAR"), missing at random ("MAR"), or missing not at random ("MNAR").
target	Character vector of variables eligible for amputation. Defaults to all variables.
by	Character vector of fully or partially observed variables used to drive MAR probabilities. Required for mechanism = "MAR".
direction	For MNAR numeric targets, whether missingness is more likely in both tails, the left tail, or the right tail.
seed	Optional random seed.
...	Passed to methods.

Details

For each target variable X_j, observed cells are removed by drawing

$$A_{ij} \sim \text{Bernoulli}(p_i)$$

and setting X_{ij} to missing when $A_{ij} = 1$. For MCAR, $p_i = \pi$, where π is prop. For MAR, probabilities are based on observed by variables through a standardized score s_i:

$$p_i = \text{logit}^{-1}(\alpha + s_i),$$

with α calibrated so that the average probability is prop. For MNAR, s_i is derived from the target variable itself; direction selects high values, low values, or both tails for numeric targets.

Value

A mimar_amputation object.

---

complete *Extract completed imputed data*

Description

complete() extracts completed data sets from imputation objects. For mimar_imputation objects, use complete(x) or complete(x, 1) for one completed data set, complete(x, "all") for all completed data sets, and complete(x, "long") for a stacked long data frame.

Usage

```
complete(x, ...)
```

Arguments

x	An object containing completed data.
...	Passed to methods.

Value

A data frame or list of data frames.

`describe` *Describe missing data and mimar objects*

Description

`describe()` summarizes missingness for data frames and returns compact summaries for `mimar` objects.

Usage

```
describe(x, ...)
```

Arguments

x	An object to describe.
...	Passed to methods.

Details

For a data matrix $X \in R^{n \times p}$, missingness is represented by the indicator

$$R_{ij} = I(X_{ij} \text{ is missing}),$$

where i indexes rows and j indexes variables. The variable-level missingness proportion reported by `describe()` is

$$\hat{\pi}_j = n^{-1} \sum_{i=1}^{n} R_{ij}.$$

Row summaries use $p^{-1} \sum_{j=1}^{p} R_{ij}$, and missingness patterns are the unique row vectors of R.

Value

A mimar S3 object.

evaluate *Evaluate imputation quality*

Description

`evaluate()` compares imputed values with known truth when an amputation object is available. Numeric targets are summarized with errors such as

$$e_{ij} = \tilde{X}_{ij} - X_{ij},$$

while categorical targets are summarized with agreement and balanced accuracy across classes. When no truth is available, `evaluate()` reports diagnostics that can be computed from the imputed data alone. Distribution, variability, and recovery summaries are computed across all completed data sets; per-imputation recovery metrics are kept in `recovery_by_imputation`.

Usage

```
evaluate(x, ...)
```

Arguments

`x`	A `mimar_imputation` or `mimar_amputation` object.
`...`	Passed to methods.

Value

A `mimar_evaluation` object.

fit *Fit an imputer learner*

Description

`fit()` trains a `mimar_imputer` on complete observed rows for one target variable. In the imputation loop, `x` is the predictor block X_{-j}^{obs} and `y` is the observed target vector X_j^{obs}.

Usage

```
fit(object, ...)

## S3 method for class 'mimar_imputer'
fit(
  object,
  x,
  y,
  target = y,
```

```
  variable = "target",
  donors = 5,
  seed = NULL,
  ...
)
```

Arguments

`object`	Object to fit.
`...`	Passed to methods.
`x`	Predictor data frame containing observed rows for the current target variable.
`y`	Observed target vector for the current variable.
`target`	Original target vector, used to validate type support and restore imputed values to the correct storage type.
`variable`	Variable name used in diagnostics and error messages.
`donors`	Number of predictive mean matching donors for `"pmm"`.
`seed`	Optional random seed.

Details

The fitted object stores the original imputer descriptor and the model needed by `predict()`. Native imputers are implemented directly in `mimar`; wrapped imputers call their original learner packages directly:

- `"norm"` fits a linear model and draws $\tilde{y} = \hat{y} + \epsilon$, with $\epsilon \sim N(0, \hat{\sigma}^2)$.
- `"pmm"` fits the same linear model but imputes by predictive mean matching: among observed donors with fitted values closest to $\hat{y}_{mis}$, one donor value is sampled.
- `"logreg"` fits a binomial GLM and draws classes from fitted Bernoulli probabilities.
- `"polyreg"` fits one-vs-rest binomial GLMs and samples classes from normalized class probabilities.

Value

A fitted object.

Methods (by class)

- `fit(mimar_imputer)`: Fit a `mimar_imputer`.

`impute` *Impute missing data*

Description

`impute()` performs single or multiple imputation through a chained update procedure owned by `mimar`. The default `imputer = "pmm"` uses predictive mean matching. A named imputer such as `"naive"`, `"rf"`, `"xgboost"`, `"knn"`, or `"glmnet"` can also be supplied to use that learner for every incomplete variable it supports. The returned object keeps completed datasets first; use `complete()` to extract one completed dataset, all completed datasets, or a stacked long data frame.

Usage

```
impute(
  x,
  m = 5,
  imputer = "pmm",
  maxit = 5,
  seed = NULL,
  donors = 5,
  ncore = 1,
  verbose = FALSE,
  ...
)

## S3 method for class 'data.frame'
impute(
  x,
  m = 5,
  imputer = "pmm",
  maxit = 5,
  seed = NULL,
  donors = 5,
  ncore = 1,
  verbose = FALSE,
  progress = interactive() && !verbose,
  ...
)

## S3 method for class 'mimar_amputation'
impute(
  x,
  m = 5,
  imputer = "pmm",
  maxit = 5,
  seed = NULL,
  donors = 5,
```

```
  ncore = 1,
  verbose = FALSE,
  progress = interactive() && !verbose,
  ...
)
```

Arguments

x	A data frame or mimar_amputation object.
m	Number of completed data sets to generate.
imputer	Imputer name or a mimar_imputer specification. Any name returned by imputer_registry() can be supplied to use that imputer for all incomplete variables it supports.
maxit	Number of chained-equation iterations.
seed	Optional random seed.
donors	Number of donor candidates used by donor-based imputers.
ncore	Number of CPU cores used to run completed datasets in parallel. The default, 1, runs sequentially. Values greater than one are passed to functionals::fmap() as ncores.
verbose	Logical; if TRUE, print an informative progress log for the chained-imputation workflow. The default, FALSE, runs silently.
...	Passed to methods.
progress	Logical; if TRUE, display a progress bar (elapsed/ETA) over completed datasets via functionals::fmap(). The default runs silently unless the session is interactive and verbose = FALSE. Has no effect when imputer = "missknn".

Details

Hyperparameters for learner-backed imputers can be supplied through the imputer() specification or directly through ... when calling impute(). For donor-based imputers, donors controls the donor pool used by "pmm", "spmm", "knn", and "hotdeck".

Let X_j be an incomplete variable and X_{-j} all remaining variables. At each iteration, mimar fits an imputer learner on observed rows

$$\hat{f}_j : X_{-j}^{obs} \rightarrow X_j^{obs}$$

and predicts missing cells X_j^{mis} from X_{-j}^{mis}. This is repeated across incomplete variables for maxit iterations and across m independent completed data sets. The algorithm is intentionally learner-agnostic: each imputer is constructed with imputer(), trained with fit(), and used through predict().

Each requested imputer is applied to all incomplete variables it supports. Use imputer_registry() to inspect target-type compatibility. Learner-backed methods are supervised stochastic update rules inside the chained workflow; inspect diagnostics and downstream sensitivity rather than treating any single learner as a guarantee of proper uncertainty quantification.

Value

A mimar_imputation object.

Methods (by class)

- `impute(data.frame)`: Impute a data frame.
- `impute(mimar_amputation)`: Impute a `mimar_amputation` object and retain truth masks for later evaluation.

`imputer` *Build an imputer learner*

Description

`imputer()` constructs a standalone learner descriptor used by the chained imputation engine. All imputers expose the same standard lifecycle:

Usage

```
imputer(method, ...)

## Default S3 method:
imputer(method, spec = NULL, ...)
```

Arguments

`method`	Imputer method name.
`...`	Hyperparameters retained for later use by `impute()` or `fit()`.
`spec`	Optional learner specification retained for future extensions.

Details

1. construct with `imputer(method)`;
2. fit on observed rows with `fit(object, x, y)`;
3. impute new rows with `predict(fitted, newdata)`.

Native learners implemented in `mimar` include `"mean"`, `"median"`, `"mode"`, `"naive"`, `"norm"`, `"pmm"`, `"spmm"`, `"logreg"`, `"polyreg"`, `"knn"`, and `"hotdeck"`. Learner-backed imputers such as `"rf"`, `"ranger"`, `"xgboost"`, `"svm"`, `"bart"`, `"nbayes"`, `"rpart"`, `"glmnet"`, `"fastgbm"`, `"densemlp"`, `"famd"`, and `"missknn"` are called directly through their original packages installed with `mimar`. Additional arguments supplied to `imputer()` are retained as hyperparameters and used by `impute()` and `fit()`. The `"superlearner"` imputer, also available as `"sl"`, cross-validates a candidate imputer library on observed cells and combines candidates using non-negative loss-based weights.

Compatibility with target types is explicit. If an imputer does not support a numeric, binary, or multiclass target, `mimar` stops with an error rather than silently falling back to another method.

Value

A `mimar_imputer` object.

Methods (by class)

- `imputer(default)`: Construct a `mimar_imputer`.

`imputer_registry` *List available mimar imputers*

Description

`imputer_registry()` returns the imputer names accepted by `impute()` and metadata describing target-type support and backend packages. Native `mimar` methods display `package = "internal"`. The result is returned as a basetable.

Usage

```
imputer_registry()
```

Value

A basetable.

`ipcw` *Inverse probability of censoring weights*

Description

`ipcw()` computes subject-level inverse-probability-of-censoring weights $w_i = \Delta_i / \hat{G}(T_i -)$, where Δ_i is the event indicator and $\hat{G}$ is the Kaplan-Meier estimate of the censoring survivor function (Kaplan-Meier applied to the reversed indicator $1 - \texttt{status}$). Weights of this form remove the bias that right-censoring introduces in a complete-case analysis and can be carried into weighted post-imputation analyses or into `evaluate()`.

Usage

```
ipcw(
  data,
  timevar,
  statusvar,
  aux = NULL,
  method = c("km", "cox"),
  type = c("event", "all"),
  stabilized = FALSE,
  truncate = NULL
)
```

Arguments

`data`	A data frame.
`timevar`	Follow-up time. A column name given as a string or as an unquoted symbol.
`statusvar`	Event indicator, `1`/`TRUE` for an event and `0`/`FALSE` for censoring. A column name given as a string or as an unquoted symbol.
`aux`	Optional character vector of auxiliary column names used to make the estimate conditional (strata for categorical variables, Cox covariates otherwise).
`method`	How `aux` enters the censoring model: `"km"` (stratified Kaplan-Meier, the default) or `"cox"` (Cox model).
`type`	`"event"` (the default) sets the weight to `0` for censored subjects; `"all"` returns `1 / G(T-)` for every subject.
`stabilized`	Multiply the weights by the marginal censoring survivor function, giving stabilized weights with mean near 1.
`truncate`	Optional upper quantile in `(0, 1)` at which to winsorise the weights; `NULL` (the default) applies no truncation.

Details

`type = "all"` instead returns $1/\hat{G}(T_i-)$ for every subject (censored subjects keep a positive weight), which is the more useful form when the weights are an auxiliary quantity rather than analysis weights. `stabilized = TRUE` multiplies the weights by the marginal censoring survivor function $\hat{G}_0(T_i-)$, giving stabilized weights with mean near 1. `truncate` winsorises the weights at the given upper quantile (e.g. `0.99`) to limit the influence of a few very large weights.

With `aux` and `method = "cox"` the censoring survivor function is subject-specific, from a Cox model of the censoring process on `aux` (requires **survival**); with `method = "km"` and categorical `aux`, $\hat{G}$ is estimated within strata.

Value

A numeric vector of weights, `nrow(data)` long. Rows with a missing time or status (or missing `aux`, when `aux` is supplied) yield `NA`.

References

Robins, J. M. and Finkelstein, D. M. (2000). Correcting for noncompliance and dependent censoring in an AIDS clinical trial with inverse probability of censoring weighted (IPCW) log-rank tests. *Biometrics*, 56(3), 779-788.

See Also

`nelsonaalen()`

Examples

```
set.seed(1)
n <- 120
sex <- factor(sample(c("F", "M"), n, TRUE))
event_time <- rexp(n, rate = 1 / 8)
cens_time <- rexp(n, rate = 1 / 12)
df <- data.frame(
  sex = sex,
  time = pmin(event_time, cens_time),
  status = as.integer(event_time <= cens_time)
)

w <- ipcw(df, "time", "status")
summary(w[w > 0])
summary(ipcw(df, time, status, type = "all"))
summary(ipcw(df, time, status, aux = "sex", type = "all", truncate = 0.95))
```

`nelsonaalen` *Nelson-Aalen cumulative hazard as an imputation predictor*

Description

`nelsonaalen()` returns the Nelson-Aalen estimate of the cumulative hazard $\hat{H}(t)$ evaluated at each subject's event or censoring time. Following White and Royston (2009), the cumulative hazard together with the event indicator is the recommended way to carry time-to-event information into the imputation model for other, partially observed variables: the raw survival time must never be used as a predictor, as it biases the imputed values.

Usage

```
nelsonaalen(
  data,
  timevar,
  statusvar,
  aux = NULL,
  method = c("auto", "strata", "breslow")
)
```

Arguments

`data`	A data frame.
`timevar`	Follow-up time. A column name given as a string or as an unquoted symbol.
`statusvar`	Event indicator, `1`/`TRUE` for an event and `0`/`FALSE` for censoring. A column name given as a string or as an unquoted symbol.
`aux`	Optional character vector of auxiliary column names used to make the estimate conditional (strata for categorical variables, Cox covariates otherwise).
`method`	How `aux` is used: `"auto"` (the default), `"strata"` for stratified estimation, or `"breslow"` for a Cox model with a Nelson-Aalen (Breslow) baseline.

Details

With no aux, the marginal Nelson-Aalen estimator is used, matching `mice::nelsonaalen()`. When aux names one or more auxiliary variables the estimate is made conditional on them: `method = "strata"` estimates a separate cumulative hazard within each combination of the (categorical) aux variables, while `method = "breslow"` fits a Cox model on aux and returns the per-subject cumulative hazard $\hat{H}_0(t)\exp(x_i^\top \hat{\beta})$ (this path requires the **survival** package). The default, `method = "auto"`, picks `"strata"` when every aux variable is categorical and `"breslow"` otherwise.

Value

A numeric vector, `nrow(data)` long, giving the cumulative hazard at each subject's time. Rows with a missing time or status (or missing aux, when aux is supplied) yield `NA`.

References

White, I. R. and Royston, P. (2009). Imputing missing covariate values for the Cox model. *Statistics in Medicine*, 28(15), 1982-1998.

See Also

`ipcw()`

Examples

```
set.seed(1)
n <- 120
df <- data.frame(
  sex = factor(sample(c("F", "M"), n, TRUE)),
  bmi = rnorm(n, 26, 4)
)
event_time <- rexp(n, rate = 1 / 8)
cens_time <- rexp(n, rate = 1 / 12)
df$time <- pmin(event_time, cens_time)
df$status <- as.integer(event_time <= cens_time)

## add H0 as a predictor to use in place of the raw survival time
df$H0 <- nelsonaalen(df, "time", "status")
df$H0_bysex <- nelsonaalen(df, time, status, aux = "sex")

df$bmi[sample(n, 25)] <- NA
impute(df[c("bmi", "sex", "status", "H0")], m = 5, imputer = "pmm", seed = 1)
```

`plot.mimar_imputation` *Diagnostic plots for mimar imputations*

Description

`plot.mimar_imputation()` draws imputation diagnostics. By default it shows imputed cell counts. Other plot types show a cell-status map, observed versus imputed distributions, boxplots across imputations, bivariate diagnostics, categorical proportions, convergence traces, variable-level imputation methods, or between-imputation variability.

Usage

```
## S3 method for class 'mimar_imputation'
plot(
  x,
 type = c("imputed", "missing", "density", "strip", "boxplot", "xy", "proportion",
    "trace", "methods", "variability"),
  variable = NULL,
  formula = NULL,
  statistic = c("mean", "sd"),
  ...
)
```

Arguments

`x`	A `mimar_imputation` object.
`type`	Plot type: `"imputed"`, `"missing"`, `"density"`, `"strip"`, `"boxplot"`, `"xy"`, `"proportion"`, `"trace"`, `"methods"`, or `"variability"`.
`variable`	Optional variable name or names used by distribution plots.
`formula`	Optional formula for bivariate and stratified diagnostics. Use forms such as `height ~ weight`, `height ~ weight | gender`, or `group ~ sex`.
`statistic`	Trace statistic, either `"mean"` or `"sd"`.
`...`	Unused.

Value

A `ggplot` object.

`pool` *Pool post-fit quantities across imputations*

Description

`pool()` combines post-fit quantities estimated separately on each completed data set. The object being pooled is a quantity: a scalar, vector, matrix, array, model coefficient, survival probability, metric, or other estimate of interest. A data frame is only a convenient tabular adapter for tidy scalar estimates; it is not the statistical target being pooled.

Usage

```
pool(x, ...)

## S3 method for class 'numeric'
pool(
  x,
  variance = NULL,
  std.error = NULL,
  covariance = NULL,
  rule = NULL,
  transform = NULL,
  inverse = NULL,
  conf.level = 0.95,
  name = "quantity",
  dfcom = NULL,
  ...
)

## S3 method for class 'list'
pool(
  x,
  variance = NULL,
  std.error = NULL,
  covariance = NULL,
  rule = NULL,
  transform = NULL,
  inverse = NULL,
  conf.level = 0.95,
  dfcom = NULL,
  ...
)

## S3 method for class 'matrix'
pool(
  x,
  variance = NULL,
  std.error = NULL,
  covariance = NULL,
  rule = NULL,
  transform = NULL,
  inverse = NULL,
  conf.level = 0.95,
  dfcom = NULL,
  ...
)

## S3 method for class 'data.frame'
pool(
```

```
  x,
  variance = NULL,
  std.error = NULL,
  covariance = NULL,
  rule = NULL,
  transform = NULL,
  inverse = NULL,
  conf.level = 0.95,
  dfcom = NULL,
  ...
)
```

Arguments

x	Quantity estimates across imputations. Use a numeric vector for one scalar quantity, a list for scalar/vector/matrix/array quantities, a matrix with imputations in rows and quantities in columns, or a data frame as a tabular adapter for tidy scalar estimates.
...	Passed to methods.
variance	Complete-data variance for the quantity in each imputation. For vector quantities this may also be a list of elementwise variance vectors or matrices matching x.
std.error	Complete-data standard error for the quantity in each imputation. Ignored when variance is supplied.
covariance	For a list of numeric vectors, optional list of covariance matrices. When supplied, vector pooling uses Rubin's multivariate matrix form and returns the pooled covariance matrix.
rule	Pooling rule. "rubin" applies Rubin's rules and requires variance, std.error, or covariance. "robust" reports median, IQR, and range across imputations. "mean" reports the mean and the between-imputation standard error $\sqrt{B/m}$, with a normal-theory interval at conf.level. It ignores within-imputation variance, so it describes Monte Carlo spread across imputations rather than total uncertainty; use "rubin" whenever complete-data variances are available. Defaults to "rubin" when variance is available and "robust" otherwise.
transform	Scale on which to pool. Either a named scale, one of "identity", "log" (positive quantities: hazard ratios, odds ratios, standard deviations, times), "logit" (probabilities, AUC, C-index), "cloglog" (survival probabilities), or "fisherz" (correlations); or a function applied to each estimate before pooling. See Details.
inverse	Inverse of a function transform, applied to pooled estimates and interval limits. Set automatically for a named scale.
conf.level	Confidence level for interval estimates.
name	Name of a scalar quantity.
dfcom	Optional complete-data degrees of freedom, i.e. the degrees of freedom the analysis would have had without missing data (for example n - p for a regression coefficient, or n - 1 for a mean). When supplied, Rubin pooling uses the Barnard and Rubin (1999) small-sample degrees of freedom, which never exceed dfcom.

When NULL (the default), the classic Rubin (1987) degrees of freedom are used; they assume a normal complete-data reference distribution and can be far too large in small samples.

Details

For a scalar quantity with estimates $Q_1, \ldots, Q_m$ and complete-data variances $U_1, \ldots, U_m$, Rubin rules are

$$\bar{Q} = m^{-1} \sum_{k=1}^{m} Q_k,$$

$$\bar{U} = m^{-1} \sum_{k=1}^{m} U_k,$$

$$B = (m-1)^{-1} \sum_{k=1}^{m} (Q_k - \bar{Q})^2,$$

and total variance is

$$T = \bar{U} + (1 + m^{-1})B.$$

The reported standard error is $\sqrt{T}$. Confidence intervals use a t reference distribution with

$$\nu = (m-1)\left(1 + r^{-1}\right)^2, \quad r = \frac{(1 + m^{-1})B}{\bar{U}}.$$

For a vector quantity, pass x as a list of numeric vectors and covariance as a list of complete-data covariance matrices. Rubin's matrix form is then used: $\bar{Q}$ is a vector and $T = \bar{U} + (1 + m^{-1})B$ is the pooled covariance matrix. For matrices or arrays, pass a list of same-shaped quantities. Unless a joint covariance structure is supplied through a vector input, these are pooled element by element, which is appropriate for grids of scalar estimands such as survival probabilities at several times and covariate profiles. For survival-probability matrices, pool_survmat() applies the complementary log-log transform and back-transform automatically.

Some metrics do not have reliable complete-data variance estimates or do not satisfy approximate normality. Following Marshall et al. (2009), pool() reports robust summaries by default when no variance is supplied: median, interquartile range, and range across imputations. Use rule = "mean" to request a mean and between-imputation standard error for such metrics.

Rubin's rules assume that each complete-data estimate is approximately normal with the supplied variance, and inference refers the pooled estimate to a t distribution. For bounded or skewed quantities (probabilities, AUC, C-index, correlations, ratios) that assumption holds far better on a transformed scale, so pool there and back-transform (Marshall et al., 2009). With a named transform, variance/std.error are given on the original scale and carried to the pooling scale with the delta method; the reported estimate, std.error, and interval are back on the original scale, while statistic, p.value, and the variance components (within_variance, between_variance, total_variance) refer to the pooling scale (for "log" the test is of a ratio equal to 1, for "logit" of a probability equal to 0.5). With a function transform, variances must already be on the transformed scale and std.error is reported there. Interval limits are ordered correctly for decreasing inverses such as the complementary log-log.

A list is the preferred input for post-fit quantities. Use a list of length m, one element per imputation. Each element can be a scalar, vector, matrix, or array. When covariance is supplied for a list of

vectors, Rubin's multivariate matrix rule is used. Otherwise list elements are pooled element by element.

Data frames are accepted as a convenience adapter, but the pooled object is not the data frame itself. Rows must encode post-fit scalar quantities: `term`, `estimate`, `std.error`, and `imputation` for Rubin pooling, or `metric`, `value`, and `imputation` for metric summaries. Metric rows that also carry a `std.error` column are pooled with Rubin's rules (for example `transform = "logit"` for a C-index or AUC); without it they get the robust summary.

Value

A `mimar_pool` object.

Methods (by class)

- `pool(numeric)`: Pool a scalar quantity observed across imputations.
- `pool(list)`: Pool a list of scalar, vector, matrix, or array quantities.
- `pool(matrix)`: Pool a matrix whose rows are imputations and columns are scalar quantities.
- `pool(data.frame)`: Tabular adapter for tidy scalar estimates or metrics.

References

Marshall A, Altman DG, Holder RL, Royston P. Combining estimates of interest in prognostic modelling studies after multiple imputation: current practice and guidelines. BMC Medical Research Methodology. 2009;9:57.

Examples

```
pool(c(0.10, 0.11, 0.09), std.error = c(0.04, 0.05, 0.04), name = "age")

betas <- list(c(age = 0.10, bmi = 0.30),
              c(age = 0.11, bmi = 0.32),
              c(age = 0.09, bmi = 0.29))
covs <- list(diag(c(0.04, 0.08)^2),
             diag(c(0.05, 0.09)^2),
             diag(c(0.04, 0.08)^2))
pool(betas, covariance = covs)

# C-index with standard errors: pool on the logit scale
pool(c(0.71, 0.74, 0.69), std.error = c(0.020, 0.021, 0.019),
     transform = "logit", name = "cindex")
```

pool_clogit *Pool conditional logistic regression models across imputations*

Description

`pool_clogit()` combines a list of `clogit` models (`survival::clogit()`), one fitted on each completed data set, using Rubin's rules. `clogit()` fits a stratified Cox model internally and returns an object of class `c("clogit", "coxph")`, so `pool_clogit()` pools and formats output identically to `pool_coxph()` (a `coef`/`exp(coef)`/`se(coef)` table plus an odds-ratio confidence-interval table), including using `nevent - p` as the complete-data df for the Barnard-Rubin correction.

Usage

```
pool_clogit(fits, conf.level = 0.95, ...)
```

Arguments

`fits`	A list of `clogit` model fits, one per completed data set, fitted with the same formula (identical coefficient names and order).
`conf.level`	Confidence level for the odds-ratio interval.
`...`	Currently unused.

Details

For term j, let $\hat{\beta}_{jk}$ and $U_{jk} = \widehat{\mathrm{Var}}(\hat{\beta}_{jk})$ be the coefficient and its model-based variance from the fit on completed data set $k = 1, \ldots, m$. Rubin's rules give the pooled coefficient $\bar{\beta}_j = m^{-1} \sum_k \hat{\beta}_{jk}$, within-imputation variance $\bar{U}_j = m^{-1} \sum_k U_{jk}$, between-imputation variance $B_j = (m-1)^{-1} \sum_k (\hat{\beta}_{jk} - \bar{\beta}_j)^2$, and total variance $T_j = \bar{U}_j + (1 + m^{-1}) B_j$. Inference uses a t reference distribution with Barnard-Rubin degrees of freedom. Hazard ratios are obtained by exponentiating the pooled coefficient and its confidence limits.

Value

A `mimar_pool_coxph`-family object (see `pool_coxph()`).

See Also

`pool_coxph()`, `pool_glm()`, `pool()`

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  library(survival)
  d <- data.frame(
    status = rep(c(1, 0), 50),
    age = rnorm(100),
    id = rep(1:50, each = 2)
```

```
  )
  d$age[sample(100, 10)] <- NA
  imp <- impute(d, m = 3, imputer = "pmm", seed = 1)
  fits <- lapply(complete(imp, "all"), function(dd) {
    clogit(status ~ age + strata(id), data = dd)
  })
  pool_clogit(fits)
}
```

`pool_coxph` *Pool Cox proportional hazards models across imputations*

Description

`pool_coxph()` combines a list of `coxph` models, one fitted on each completed data set (see `complete(x, "all")`), using Rubin's rules on the log-hazard scale. The printed output mirrors `summary.coxph()`: a coefficient table with `coef`, `exp(coef)`, `se(coef)`, and a hazard-ratio confidence interval table, with an added `df` column for the pooled (Barnard-Rubin) degrees of freedom.

Usage

```
pool_coxph(fits, conf.level = 0.95, ...)
```

Arguments

`fits`	A list of `coxph` model fits, one per completed data set, fitted with the same formula (identical coefficient names and order).
`conf.level`	Confidence level for the hazard-ratio interval.
`...`	Currently unused.

Details

For term j, let $\hat{\beta}_{jk}$ and $U_{jk} = \widehat{\mathrm{Var}}(\hat{\beta}_{jk})$ be the coefficient and its model-based variance from the fit on completed data set $k = 1, \ldots, m$. Rubin's rules give the pooled coefficient $\bar{\beta}_j = m^{-1} \sum_k \hat{\beta}_{jk}$, within-imputation variance $\bar{U}_j = m^{-1} \sum_k U_{jk}$, between-imputation variance $B_j = (m-1)^{-1} \sum_k (\hat{\beta}_{jk} - \bar{\beta}_j)^2$, and total variance $T_j = \bar{U}_j + (1+m^{-1}) B_j$. Inference uses a t reference distribution with Barnard-Rubin degrees of freedom. Hazard ratios are obtained by exponentiating the pooled coefficient and its confidence limits.

Value

A `mimar_pool_coxph` object with `coefficients` (pooled log-hazard table) and `conf_int` (pooled hazard-ratio table).

See Also

`pool_glm()`, `pool()`

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  library(survival)
  imp <- impute(survival::lung, m = 3, imputer = "pmm", seed = 1)
  fits <- lapply(complete(imp, "all"), function(d) {
    coxph(Surv(time, status) ~ age + sex + ph.ecog, data = d)
  })
  pool_coxph(fits)
}
```

`pool_glm` *Pool generalized linear models across imputations*

Description

`pool_glm()` combines a list of `glm` models, one fitted on each completed data set (see `complete(x, "all")`), using Rubin's rules on the linear predictor scale. The printed output mirrors `summary.glm()`'s coefficient table (`Estimate`, `Std. Error`, test statistic, p-value), with an added `df` column for the pooled (Barnard-Rubin) degrees of freedom. As with `pool_coxph()`, the reference distribution is Student's t throughout; the printed statistic is labelled `t value` for families with an estimated dispersion (gaussian, Gamma, inverse.gaussian) and `z value` otherwise, to match the labelling convention of `summary.glm()`, though both derive from the same pooled t reference distribution.

Usage

```
pool_glm(fits, conf.level = 0.95, ...)
```

Arguments

`fits`	A list of `glm` model fits, one per completed data set, fitted with the same formula (identical coefficient names and order) and the same family.
`conf.level`	Confidence level; currently only used for internal pooled-quantity computation, not printed by default.
`...`	Currently unused.

Details

For term j, let $\hat{\beta}_{jk}$ and $U_{jk} = \widehat{\mathrm{Var}}(\hat{\beta}_{jk})$ be the coefficient and its model-based variance from the fit on completed data set $k = 1, \ldots, m$. Rubin's rules give the pooled coefficient $\bar{\beta}_j = m^{-1} \sum_k \hat{\beta}_{jk}$, within-imputation variance $\bar{U}_j = m^{-1} \sum_k U_{jk}$, between-imputation variance $B_j = (m-1)^{-1} \sum_k (\hat{\beta}_{jk} - \bar{\beta}_j)^2$, and total variance $T_j = \bar{U}_j + (1 + m^{-1}) B_j$. Inference uses a t reference distribution with Barnard-Rubin degrees of freedom. Hazard ratios are obtained by exponentiating the pooled coefficient and its confidence limits.

Value

A `mimar_pool_glm` object with a pooled `coefficients` table.

See Also

pool_coxph(), pool()

Examples

```
imp <- impute(mtcars, m = 3, imputer = "pmm", seed = 1)
fits <- lapply(complete(imp, "all"), function(d) {
  glm(vs ~ mpg + wt, data = d, family = binomial())
})
pool_glm(fits)
```

---

pool_lm *Pool linear models across imputations*

Description

pool_lm() combines a list of lm models, one fitted on each completed data set (see complete(x, "all")), using Rubin's rules. The printed output mirrors summary.lm()'s coefficient table (Estimate, Std. Error, t value, Pr(>|t|)), with an added df column for the pooled (Barnard-Rubin) degrees of freedom, using df.residual() of a single fit as the complete-data df.

Usage

```
pool_lm(fits, conf.level = 0.95, ...)
```

Arguments

fits	A list of lm model fits, one per completed data set, fitted with the same formula (identical coefficient names and order).
conf.level	Confidence level; currently only used for internal pooled-quantity computation, not printed by default.
...	Currently unused.

Details

For term j, let $\hat{\beta}_{jk}$ and $U_{jk} = \widehat{\mathrm{Var}}(\hat{\beta}_{jk})$ be the coefficient and its model-based variance from the fit on completed data set $k = 1, \ldots, m$. Rubin's rules give the pooled coefficient $\bar{\beta}_j = m^{-1} \sum_k \hat{\beta}_{jk}$, within-imputation variance $\bar{U}_j = m^{-1} \sum_k U_{jk}$, between-imputation variance $B_j = (m-1)^{-1} \sum_k (\hat{\beta}_{jk} - \bar{\beta}_j)^2$, and total variance $T_j = \bar{U}_j + (1+m^{-1}) B_j$. Inference uses a t reference distribution with Barnard-Rubin degrees of freedom. Hazard ratios are obtained by exponentiating the pooled coefficient and its confidence limits.

Value

A mimar_pool_lm object with a pooled coefficients table.

See Also

pool_glm(), pool_coxph(), pool()

Examples

```
imp <- impute(mtcars, m = 3, imputer = "pmm", seed = 1)
fits <- lapply(complete(imp, "all"), function(d) lm(mpg ~ wt + hp, data = d))
pool_lm(fits)
```

---

pool_panglm *Pool panel (panglm) models across imputations*

Description

pool_panglm() combines a list of panglm panel-data model fits (see the panglm package), one fitted on each completed data set (see complete(x, "all")), using Rubin's rules. The printed output mirrors pool_glm()'s coefficient table (Estimate, Std. Error, test statistic, p-value), with an added df column for the pooled (Barnard-Rubin) degrees of freedom. As with pool_glm(), the reference distribution is Student's t throughout; the printed statistic is labelled t value for the gaussian family and z value otherwise.

Usage

```
pool_panglm(fits, conf.level = 0.95, dfcom = NULL, ...)
```

Arguments

fits	A list of panglm model fits, one per completed data set, fitted with the same formula and the same model/effect/family/ vcov settings (identical coefficient names and order).
conf.level	Confidence level; currently only used for internal pooled-quantity computation, not printed by default.
dfcom	Optional complete-data degrees of freedom. NULL (default) uses $G - 1$ for cluster-robust fits and df.residual() otherwise.
...	Currently unused.

Details

The within-imputation variance is the covariance each fit carries (vcov(fit), set by the vcov argument of panglm::panglm()). The complete-data df must match that covariance. By default it is $G-1$ (G panel individuals) for a cluster-robust covariance, the usual reference for cluster-robust inference, and df.residual() of the first fit otherwise (for a within fit, $NT-N-K$). Supply dfcom to override it. When no complete-data df is available (random effects fits have no df.residual()), Rubin's (1987) large-sample df is used.

All fits must share the same formula and the same panglm settings (model, effect, family, vcov_type), since these determine what the pooled coefficients represent; pool_panglm() warns if they differ.

Rubin's rules are valid only if the imputation model is at least as rich as the analysis model. For `model = "within"` or `"random"` analyses, imputing a time-varying variable with a single-level ("flat") imputation model that ignores the panel structure biases the within coefficient toward the between association and makes the fixed- versus random-effects contrast spurious; impute with a model that includes the panel structure (for example individual indicators or a random intercept).

Value

A `mimar_pool_panglm` object with a pooled `coefficients` table.

See Also

`pool_glm()`, `pool_lm()`, `pool_coxph()`, `pool()`

Examples

```
if (requireNamespace("panglm", quietly = TRUE)) {
  dat <- panglm::copd
  dat$crp[sample(nrow(dat), 20)] <- NA
  imp <- impute(dat, m = 3, imputer = "pmm", seed = 1)
  fits <- lapply(complete(imp, "all"), function(d) {
    panglm::panglm(fev1 ~ treatment + age + crp, data = d,
                   index = c("id", "visit"), model = "pooling",
                   family = "gaussian")
  })
  pool_panglm(fits)
}
```

---

pool_survmat *Pool survival-probability matrices across imputations*

Description

`pool_survmat()` pools a list of same-shaped survival-probability matrices or arrays by applying Rubin-style pooling on the complementary log-log scale and back-transforming the result. The helper is designed for predicted survival probabilities at a grid of times, subjects, or covariate profiles.

Usage

```
pool_survmat(
  x,
  variance = NULL,
  std.error = NULL,
  rule = NULL,
  conf.level = 0.95,
  clip = 1e-12,
  dfcom = NULL,
```

```
  ...
)
```

Arguments

`x`	A non-empty list of numeric matrices or arrays containing survival probabilities. All elements must have the same dimensions.
`variance`	Optional list of within-imputation variances with the same dimensions as `x`. When supplied, the variance is pooled on the transformed scale.
`std.error`	Optional list of within-imputation standard errors with the same dimensions as `x`. Ignored when `variance` is supplied.
`rule`	Pooling rule. Defaults to Rubin pooling when within-imputation variance is supplied and to the robust median/IQR/range summary otherwise.
`conf.level`	Confidence level for interval estimates.
`clip`	Small positive value used to keep probabilities away from 0 and 1 before applying the cloglog transform.
`dfcom`	Optional complete-data degrees of freedom. See `pool()`.
`...`	Passed to lower-level pooling helpers.

Details

Let S_{ijk} be the survival probability for imputation j, row index i, and column index k, with $j = 1, \ldots, m$. Define the complementary log-log transform

$$Z_{ijk} = g(S_{ijk}) = \log\{-\log(S_{ijk})\}.$$

Rubin pooling is then applied elementwise on Z_{ijk}:

$$\bar{Z}_{ik} = m^{-1} \sum_{j=1}^{m} Z_{ijk},$$

$$\bar{U}_{ik} = m^{-1} \sum_{j=1}^{m} U_{ijk},$$

$$B_{ik} = (m-1)^{-1} \sum_{j=1}^{m} (Z_{ijk} - \bar{Z}_{ik})^2,$$

$$T_{ik} = \bar{U}_{ik} + (1 + m^{-1}) B_{ik}.$$

The pooled survival probability is

$$\hat{S}_{ik} = g^{-1}(\bar{Z}_{ik}) = \exp\{-\exp(\bar{Z}_{ik})\}.$$

A delta-method standard error on the original scale is

$$\mathrm{SE}(\hat{S}_{ik}) = \left| \frac{d}{dz} g^{-1}(z) \right|_{z=\bar{Z}_{ik}} \sqrt{T_{ik}} = \hat{S}_{ik}\{-\log(\hat{S}_{ik})\}\sqrt{T_{ik}}.$$

Confidence intervals are obtained on the transformed scale and then back-transformed:

$$[\ g^{-1}(\bar{Z}_{ik} + t_{\nu,1-\alpha/2}\sqrt{T_{ik}}),\ g^{-1}(\bar{Z}_{ik} - t_{\nu,1-\alpha/2}\sqrt{T_{ik}})\ ],$$

where ν is the Rubin (1987) degrees of freedom, or the Barnard-Rubin (1999) degrees of freedom when `dfcom` is supplied. Because g^{-1} is decreasing, the lower survival bound comes from the upper transformed bound.

Probabilities are clipped to `[clip, 1 - clip]` before transformation to avoid `log(0)` at the boundaries.

Value

A `mimar_pool` object with pooled survival probabilities.

Examples

```
surv <- list(
  matrix(c(0.90, 0.80, 0.70, 0.60), 2, 2),
  matrix(c(0.91, 0.79, 0.72, 0.61), 2, 2),
  matrix(c(0.89, 0.81, 0.71, 0.59), 2, 2)
)
pool_survmat(surv)
```

---

pool_survreg *Pool parametric survival regression models across imputations*

Description

`pool_survreg()` combines a list of `survreg` accelerated failure time models (`survival::survreg()`), one fitted on each completed data set, using Rubin's rules. The printed output mirrors the coefficient table of `summary.survreg()` (`Value`, `Std. Error`, `z`, `p`), with an added `df` column for the pooled (Barnard-Rubin) degrees of freedom, using `df.residual()` of a single fit as the complete-data df. Only the regression coefficients are pooled; the distribution's scale (or shape) parameter is treated as a nuisance parameter and is not pooled, matching how these models are typically reported after multiple imputation.

Usage

```
pool_survreg(fits, conf.level = 0.95, ...)
```

Arguments

`fits`	A list of `survreg` model fits, one per completed data set, fitted with the same formula and distribution (identical coefficient names and order).
`conf.level`	Confidence level; currently only used for internal pooled-quantity computation, not printed by default.
`...`	Currently unused.

Details

For term j, let $\hat{\beta}_{jk}$ and $U_{jk} = \widehat{\mathrm{Var}}(\hat{\beta}_{jk})$ be the coefficient and its model-based variance from the fit on completed data set $k = 1, \ldots, m$. Rubin's rules give the pooled coefficient $\bar{\beta}_j = m^{-1} \sum_k \hat{\beta}_{jk}$, within-imputation variance $\bar{U}_j = m^{-1} \sum_k U_{jk}$, between-imputation variance $B_j = (m-1)^{-1} \sum_k (\hat{\beta}_{jk} - \bar{\beta}_j)^2$, and total variance $T_j = \bar{U}_j + (1+m^{-1}) B_j$. Inference uses a t reference distribution with Barnard-Rubin degrees of freedom. Hazard ratios are obtained by exponentiating the pooled coefficient and its confidence limits.

Value

A `mimar_pool_survreg` object with a pooled `coefficients` table.

See Also

`pool_coxph()`, `pool_glm()`, `pool()`

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  library(survival)
  imp <- impute(survival::lung, m = 3, imputer = "pmm", seed = 1)
  fits <- lapply(complete(imp, "all"), function(d) {
    survreg(Surv(time, status) ~ age + sex, data = d, dist = "weibull")
  })
  pool_survreg(fits)
}
```

---

pool_test *Pooled multi-parameter tests across imputations*

Description

`pool_test()` tests several coefficients at once after multiple imputation, for example all dummy variables of a categorical predictor or a full model against a nested one. Testing each pooled coefficient with its own t statistic does not answer that question, and the per-imputation Wald or likelihood-ratio statistics cannot simply be averaged, because that ignores between-imputation variability. The three standard combining rules each refer a pooled statistic to an $F(k, \nu)$ distribution, where k is the number of tested parameters.

Usage

```
pool_test(
  fits,
  fits0 = NULL,
  terms = NULL,
  method = c("D1", "D2", "D3"),
```

```
  use = c("wald", "likelihood"),
  dfcom = NULL
)
```

Arguments

`fits`	A list of fitted models, one per completed data set.
`fits0`	Optional list of nested (null) models fitted on the same completed data sets. Required for D3 and for D2 with `use = "likelihood"`.
`terms`	Character vector of coefficient names to test against zero. Used by D1 and Wald-based D2 when `fits0` is `NULL`.
`method`	One of `"D1"`, `"D2"`, or `"D3"`.
`use`	For D2, whether to combine Wald (`"wald"`) or likelihood-ratio (`"likelihood"`) statistics.
`dfcom`	Optional complete-data degrees of freedom, used by D1 only. For `panglm` fits, `NULL` uses the same default as `pool_panglm()`: $G - 1$ for a cluster-robust covariance, `df.residual()` otherwise.

Details

- `"D1"` (Li, Raghunathan and Rubin, 1991): multivariate Wald test built from the pooled coefficients and the within- and between-imputation covariance matrices. It needs `coef()` and `vcov()` for every fit, and assumes the fraction of missing information is similar across the tested parameters. With `dfcom`, the denominator df uses the Reiter (2007) small-sample correction.
- `"D2"` (Li, Meng, Raghunathan and Rubin, 1991): combines the m per-imputation χ^2 statistics, either Wald statistics (`use = "wald"`) or likelihood-ratio statistics from `logLik()` (`use = "likelihood"`, needs `fits0`). It works whenever those statistics exist but is known to be less powerful and can be anti-conservative, so prefer D1 or D3 when available.
- `"D3"` (Meng and Rubin, 1992): likelihood-ratio test that also evaluates each imputation's likelihood at the pooled estimates. It needs nested `fits0` and is implemented for `lm` fits and for `glm` fits with a gaussian, binomial, or poisson family.

The terms tested are either `terms`, or, when `fits0` is supplied, the coefficients present in `fits` but absent from `fits0`. The estimated relative increase in variance due to nonresponse (`riv`) is reported untruncated; a negative value, possible for D2 and D3 with small `m`, signals that more imputations are needed. Results match `mitml::testModels()`. For `glm` fits, D3 evaluates each likelihood exactly at the pooled coefficients; `mice::D3()` instead refits an intercept with the pooled linear predictor as offset, so the two can differ slightly.

Value

A `mimar_pool_test` object: a one-row table with the method, `statistic` (the F value), `df1`, `df2`, `p.value`, relative increase in variance `riv`, and `m`, plus the tested `terms`.

References

Li, K. H., Raghunathan, T. E., and Rubin, D. B. (1991). Large-sample significance levels from multiply imputed data using moment-based statistics and an F reference distribution. *Journal of the American Statistical Association*, 86, 1065-1073.

Li, K. H., Meng, X.-L., Raghunathan, T. E., and Rubin, D. B. (1991). Significance levels from repeated p-values with multiply-imputed data. *Statistica Sinica*, 1, 65-92.

Meng, X.-L., and Rubin, D. B. (1992). Performing likelihood ratio tests with multiply-imputed data sets. *Biometrika*, 79, 103-111.

Reiter, J. P. (2007). Small-sample degrees of freedom for multi-component significance tests with multiple imputation for missing data. *Biometrika*, 94, 502-508.

See Also

pool_lm(), pool_glm(), pool()

Examples

```
d <- mtcars
d$cyl <- factor(d$cyl)
set.seed(1)
d$hp[sample(32, 6)] <- NA
imp <- impute(d, m = 5, imputer = "pmm", seed = 1)
comp <- complete(imp, "all")
fits <- lapply(comp, function(x) lm(mpg ~ cyl + hp + wt, data = x))
fits0 <- lapply(comp, function(x) lm(mpg ~ hp + wt, data = x))
# Does cyl (two dummy coefficients) matter?
pool_test(fits, fits0, method = "D1")
pool_test(fits, fits0, method = "D3")
pool_test(fits, terms = c("cyl6", "cyl8"), method = "D2")
```

Index

ampute *(*ampute.data.frame*)*, 2
ampute.data.frame, 2

complete, 3

describe, 4

evaluate, 5
evaluate(), *10*

fit, 5

impute, 7
imputer, 9
imputer_registry, 10
ipcw, 10
ipcw(), *13*

nelsonaalen, 12
nelsonaalen(), *11*

plot.mimar_imputation, 13
pool, 14
pool(), *19*, *20*, *22–25*, *27*, *29*
pool_clogit, 19
pool_coxph, 20
pool_coxph(), *19*, *21–24*, *27*
pool_glm, 21
pool_glm(), *19*, *20*, *23*, *24*, *27*, *29*
pool_lm, 22
pool_lm(), *24*, *29*
pool_panglm, 23
pool_panglm(), *28*
pool_survmat, 24
pool_survreg, 26
pool_test, 27