

Package: hvtiRimputation (via r-universe)

October 20, 2026

Title Missing Value Imputation for the HVTI CORR Group

Version 0.1.3

Description Fills missing values in a data frame by a stated method and returns the filled data together with a row-level record of exactly which values were changed. Ports the mean-imputation behaviour of the Cleveland Clinic CORR group's SAS library -- 'PROC STANDARD' with 'REPLACE', and the 'imputsub' macro that wraps it. The package knows nothing about the warehouse, a build, or any study: the input is a data frame and the output is a data frame plus a record.

License GPL-3

URL <https://github.com/ehrlinger/hvtiRimputation>,
<https://ehrlinger.github.io/hvtiRimputation/>

BugReports <https://github.com/ehrlinger/hvtiRimputation/issues>

Encoding UTF-8

Depends R (>= 4.4.0)

Imports mice, stats, utils, withr (>= 2.5.0)

Suggests bit64, knitr, quarto, testthat (>= 3.0.0)

VignetteBuilder quarto

SystemRequirements Quarto command line tool
(<<https://github.com/quarto-dev/quarto-cli>>)

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libx11-dev zlib1g-dev

Repository <https://ehrlinger.r-universe.dev>

Date/Publication 2026-10-14 22:54:56 UTC

RemoteUrl <https://github.com/ehrlinger/hvtiRimputation>

RemoteRef HEAD

RemoteSha 311832b1de6142f9d6187d8f84d6e0c2b3d31a93

Contents

hvti_imputation	2
hvti_imputation_multi	3
imputation-accessors	4
imputation_indicators	5
impute_mean	6
impute_multiple	7
summary.hvti_imputation	10
summary.hvti_imputation_multi	11
Index	12

hvti_imputation	<i>The imputation record</i>
-----------------	------------------------------

Description

What every function in this package returns: the completed data, and a record of exactly what was changed to complete it.

Why a matrix and not a count

The record is a logical matrix parallel to the data – one row per row of the input, one column per imputed variable, TRUE where the value was filled. A summary saying "12 values were imputed in this variable" cannot answer *was this patient's value imputed?*, which is the question an audit asks. The matrix answers it directly and can be reduced to any count; the reverse is impossible.

The matrix form is chosen over a long data frame deliberately. It lines up with the data row for row without a join, and it stays small for a wide dataset. A long form reads more easily and is much larger, and the reading is what `summary.hvti_imputation()` is for.

What it holds

- data the completed data frame, via `imputed_data()`
 - matrix the logical record, via `imputed_matrix()`
 - complete_case_pass row-level: would this row have survived listwise deletion? Read from the input, before anything was filled
 - analysis_pass row-level: is this row complete *after* imputing? Not derivable from the two above – a row can have a value filled and still be incomplete because a column outside vars is missing
 - provenance method, m, the package version that produced it, and the seed where the method is stochastic, via `imputation_provenance()`
- An imputed dataset that does not say how it was imputed cannot be reproduced. The SAS corpus this package ports is the proof: thirty years of imputed datasets whose method is now only partly recoverable, and only by scanning the code that produced them. So the provenance is not optional and travels with the artifact.

hvti_imputation_multi *The multiple-imputation record*

Description

What `impute_multiple()` returns: m completed datasets, stacked long, and the record of how they were produced.

Why long, and why not a list of m frames

`hvti_imputation`'s own record is a matrix rather than a count because the record is meant to reduce, never to require reconstruction. The same reasoning picks the storage shape here. A long frame – one `.imp` column naming the draw, one `.id` column naming the original row, then every column of the input – reduces to a single completed dataset with one `imputation == 1` filter, and is already the shape a correct downstream analysis expects: fit a model to each draw, then pool the fits. A list of m frames does not reduce to anything without picking an element out of the list first. The long form is also the shape `mice::complete(imp, "long")` already returns, and the shape SAS's PROC MI OUT= produces – unforced parity with both.

Why the matrix and the two row-level columns are not lists of m

Which cells were originally missing does not change across the m draws – only what they were filled *with* does – so `imputed_matrix()` is identical for every draw and is stored once. `complete_case_pass()` is read from the input before any imputation runs, so it is the same for every draw by construction. `analysis_pass()` is less obvious but still invariant: `mice` guarantees every column that entered the imputation is complete in every draw, so completeness over `cc_vars` evaluates identically regardless of which draw's values are substituted in. Storing three lists of m identical vectors would work, but it would misstate the record – it would let a caller ask a question ("does row 12's analysis-pass status differ by draw?") that cannot have a "yes" for any input this function accepts.

What it holds

`data_long` every column of the input, stacked m times with `.imp (1:m)` and `.id` (the original row number) columns added, via `imputed_data()`

`matrix` the logical record, via `imputed_matrix()`

`complete_case_pass` row-level, read from the input

`analysis_pass` row-level, after imputing

`provenance` method (one per imputed variable), m , `maxit`, and the rest, via `imputation_provenance()`

imputation-accessors *Read an imputation record*

Description

Accessors for the object `impute_mean()` returns, and (with one exception) for the object `impute_multiple()` returns. `imputed_data()` is the exception: `impute_multiple()`'s result holds `m` completed datasets, so `imputed_data(x, imputation)` requires a second argument there, with no default – see `impute_multiple()`.

Usage

```
imputed_data(x, ...)

imputed_matrix(x)

imputed_any(x)

complete_case_pass(x)

analysis_pass(x)

kept_by_imputation(x)

imputation_provenance(x)

## S3 method for class 'hvti_imputation_multi'
imputed_data(x, imputation, ...)
```

Arguments

<code>x</code>	An object of class <code>hvti_imputation</code> or <code>hvti_imputation_multi</code> .
<code>...</code>	Passed on to methods. <code>imputed_data()</code> 's <code>hvti_imputation_multi</code> method takes a required <code>imputation</code> argument this way; see <code>impute_multiple()</code> .
<code>imputation</code>	For the <code>hvti_imputation_multi</code> method: which completed dataset to return. A single integer in <code>1:m</code> , or "long" for all <code>m</code> stacked together. Required, with no default – there is no zero-argument form that hands back something a caller could average cell by cell across the <code>m</code> draws. See <code>dev/specs/2026-09-23-impute-multiple-design.</code> for why that specific silent default would reproduce the exact bug this design was written against.

Details

`imputed_any()` and `complete_case_pass()` are the two row-level columns the attrition record consumes. They enter a CONSORT tracker as an **annotation** – a stage that records something about the rows without removing any – and not as an exclusion. Rows kept only because a covariate was filled in are not excluded, and they are not fully observed either, and a record with no vocabulary for that third state will misreport one or the other.

Use `kept_by_imputation()` for the counterfactual, not `imputed_any(x) & !complete_case_pass(x)`. That expression overcounts. It is TRUE for a row that had a value filled but is *still* incomplete because a column outside vars is missing – a row that never enters the analysis at all, and so was not kept by anything.

The two agree only when imputation completes every row, which is why the error is easy to miss: in the study the design was written against, the analysis set was every row, so post-imputation completeness was uniformly TRUE and the wrong formula gave the right answer.

Value

`imputed_data()` a data frame; `imputed_matrix()` a logical matrix; `imputed_any()`, `complete_case_pass()`, `analysis_pass()` and `kept_by_imputation()` logical vectors, one element per row; `imputation_indicators()` a data frame of logical columns; `imputation_provenance()` a list.

Examples

```
dat <- data.frame(age = c(50, 60, NA, 70), bmi = c(NA, 25, 27, 29))
out <- impute_mean(dat, vars = c("age", "bmi"))

imputed_any(out)
complete_case_pass(out)
analysis_pass(out)

# Rows in the analysis only because a covariate was filled:
sum(kept_by_imputation(out))
```

`imputation_indicators` *Missingness indicators, generated from the record*

Description

Returns one logical column per imputed variable, TRUE where that variable's value was filled. A caller reproducing a SAS analysis that carried `ms_*` indicators as model terms renames in one step: `imputation_indicators(x, prefix = "ms_")`.

Usage

```
imputation_indicators(x, prefix = "imputed_")
```

Arguments

<code>x</code>	An object of class <code>hvti_imputation</code> or <code>hvti_imputation_multi</code> .
<code>prefix</code>	Prepended to each imputed variable's name. Defaults to <code>"imputed_"</code> .

Details

The indicators are generated rather than stored, and the SAS `ms_` naming is not the default. A convention adopted for parity becomes permanent – it would be ours to maintain forever the moment a study depended on it – so the record stays the single source of truth and the name stays the caller’s choice.

The principle underneath is not negotiable and predates the naming: **a value that means "observed" and a value that means "filled in" must not be the same value**. The indicators are how that survives into a model.

Value

A data frame of logical columns, one per imputed variable, with `nrow()` matching the imputed data.

Examples

```
dat <- data.frame(age = c(50, 60, NA, 70), bmi = c(NA, 25, 27, 29))
out <- impute_mean(dat, vars = c("age", "bmi"))
imputation_indicators(out)
imputation_indicators(out, prefix = "ms_")
```

`impute_mean`

Fill missing values with the variable mean

Description

Replaces every missing value in `vars` with that variable’s mean over its non-missing values, and returns the filled data together with a record of exactly which values were changed.

Usage

```
impute_mean(data, vars, cc_vars = names(data))
```

Arguments

<code>data</code>	A data frame.
<code>vars</code>	Character vector of columns to impute. Required, with no default – see Divergences.
<code>cc_vars</code>	Character vector of columns over which <code>complete_case_pass</code> is evaluated, defaulting to every column of data. This is the listwise deletion the imputation is standing in for, so it is usually the whole analysis frame rather than just <code>vars</code> .

Details

This is the R form of `PROC STANDARD ... REPLACE with no MEAN=STD=`, which is what the `imputsub` macro wraps. It is **single** imputation: one completed dataset, and no between-imputation variance. Multiple imputation is not yet implemented; see the README. It will be a separate function, because a function that quietly does one method when the caller expected the other is a worse failure than no function at all.

Value

An object of class `hvti_imputation`. `imputed_data()` gives the completed data frame, `imputed_matrix()` the row-by-variable record of what was filled, and `imputation_provenance()` how it was produced. See `hvti_imputation` for the full shape.

Divergences from SAS

vars is required. PROC STANDARD with no VAR statement processes every numeric variable in the dataset. That default is not inherited here, for the same reason `impute_multiple()` inherits no `m`: an imputation that silently chose its own variable list is the failure the record in this package exists to catch. A port that mean-imputes the *wrong* variable list still reaches the right row count and still looks correct; it does not reach the right `sum(imputed_any())`.

PROC STANDARD MEAN=0 STD=1 REPLACE **is not this function**. That form standardises *and* fills – in standardised units the fill value 0 is the mean, so it is imputation too, but it also rescales every value including the observed ones. It is not implemented here. Do not reach for `impute_mean()` to reproduce a job that used it.

BY-group means are not implemented. How much of the corpus imputes within BY groups has not been measured, so this is a gap rather than a decision.

See Also

`imputed_any()` and `complete_case_pass()` for the two row-level columns the attrition record consumes. Multiple imputation (`impute_multiple()`) is designed but not yet implemented; see the README.

Examples

```
dat <- data.frame(age = c(50, 60, NA, 70), bmi = c(NA, 25, 27, 29))
out <- impute_mean(dat, vars = c("age", "bmi"))
imputed_data(out)
imputed_matrix(out)
```

impute_multiple

Fill missing values with multiple imputation

Description

Draws `m` completed datasets by chained-equations multiple imputation (mice), and returns them together with a record of exactly what was filled. This is the R form of PROC MI, which the `mult_imput` macro wraps.

Usage

```
impute_multiple(
  data,
  vars,
  m,
  cc_vars = names(data),
  method = NULL,
  maxit = 5L,
  seed = getOption("hvtiRimputation.seed")
)
```

Arguments

<code>data</code>	A data frame.
<code>vars</code>	Character vector of columns to impute. Required, with no default – see Divergences.
<code>m</code>	Number of completed datasets to draw. Required, with no default – for the same reason <code>vars</code> has none in <code>impute_mean()</code> : the SAS corpus's NIMPUTE defaults disagree across macro copies, so there is no single "the default" to inherit.
<code>cc_vars</code>	Character vector of columns over which <code>complete_case_pass</code> is evaluated, defaulting to every column of data. This is the listwise deletion the imputation is standing in for, so it is usually the whole analysis frame rather than just <code>vars</code> .
<code>method</code>	<code>NULL</code> (the default) uses <code>mice</code> 's own per-column method selection for every <code>vars</code> column. A named character vector overrides specific columns by name; every column not named keeps its default. There is no single unnamed string form.
<code>maxit</code>	Number of chained-equations iterations. Passed to <code>mice::mice()</code> .
<code>seed</code>	A single whole number. Required , either here or once per session through <code>options(hvtiRimputation.seed =)</code> , which is where the default reads it from. With neither, the call fails rather than drawing from an unseeded RNG. Passed to <code>mice::mice()</code> and recorded in <code>imputation_provenance()</code> .

Details

Unlike `impute_mean()`, a categorical variable is a normal imputation target here: `method` defaults to `mice::make.method(data[vars])`, `mice`'s own per-column dispatch (`pmm` for numeric, `logreg` for a 2-level factor, `polyreg` for an unordered factor with more than two levels, `polr` for an ordered factor). **There is no single method string applied to every column.** A caller overriding specific columns passes a named character vector naming only those columns; every other column keeps its own default. Forcing one method across a mix of column types is a silent, package-wide default of exactly the kind this package's design avoids for `vars` and `m`, and it is not what `impute_mean()`'s single completed dataset needs multiplied by `m` – it needs `mice` making its own per-column decision, `m` times.

`impute_multiple()` trusts the class of each `vars` column as the sole signal for what kind of variable it is, and does not re-derive that from its distinct-value count. Deciding a column's type is `hvtiRutilities::r_data_types()`'s job (or the caller's own `factor()/as.logical()`), done once, before imputation – not a second, independent guess made here that could silently disagree with the first one.

Reproducibility. The same data, arguments and seed give the same imputed values whatever state R's random number generator was in beforehand. The call runs under `withr::with_seed()` with the RNG kind pinned to R's defaults ("Mersenne-Twister", "Inversion", "Rejection"), so a caller who changed `RNGkind()` still gets the same draws, and the caller's own RNG state is restored afterwards: calling this function does not move their random stream. Before 0.1.2 the default was `seed = NA`, which left the draws to whatever the global RNG held; a call that already passed a seed gets the same values as before when the RNG kind is R's default.

Choosing m . For a prediction model, $m = 1$, an outcome-free single draw, is defensible. $m > 1$ costs more but lets a downstream variable screen show how much its selection depends on which draw it saw.

Value

An object of class `hvti_imputation_multi`. `imputed_data()` with a required imputation argument gives one completed data frame or all m stacked long, `imputed_matrix()` the row-by-variable record of what was filled (identical across every draw), and `imputation_provenance()` how it was produced. See `hvti_imputation_multi` for the full shape.

Divergences from SAS

vars and m are both required. PROC MI with no VAR statement processes every numeric variable, and its NIMPUTE default varies by macro copy in the corpus this package ports. Neither default is inherited here, for the reason recorded in `impute_mean()`'s own Divergences section.

vars needs at least two columns. `mice::mice()` fits chained equations, so a single column has no other column to condition on – passing one fails inside `mice` with no context. This function checks for it and refuses with an explanation instead.

data cannot use `.imp` or `.id` as column names. Both are reserved for the long-format draw index and row identifier this function adds.

No pooling. This function returns m valid completed datasets and their record. Combining them into a single inferential answer – fitting a model to each draw and pooling by Rubin's rules – is the caller's job. Multiple imputation is only multiple imputation once the results are pooled, and that is separate work with its own verification; see the package README.

See Also

`impute_mean()` for single imputation – there is deliberately no one function that picks between them. `imputed_any()` and `complete_case_pass()` for the two row-level columns the attrition record consumes.

Examples

```
set.seed(42)
n <- 30
dat <- data.frame(
  age = round(rnorm(n, 60, 8)),
  grp = factor(sample(c("a", "b", "c"), n, replace = TRUE)),
  flag = sample(c(TRUE, FALSE), n, replace = TRUE)
)
dat$age[sample(n, 4)] <- NA
```

```

dat$grp[sample(n, 4)] <- NA
dat$flag[sample(n, 4)] <- NA

out <- impute_multiple(dat, vars = c("age", "grp", "flag"), m = 2,
                      maxit = 2, seed = 1)
imputed_data(out, imputation = 1)
imputed_data(out, imputation = "long")
imputed_matrix(out)

```

```
summary.hvti_imputation
```

Summarise what was imputed

Description

Reduces the record to a per-variable table. It is a view of `imputed_matrix()`, never a replacement for it: a count cannot say whether a particular row was filled.

Usage

```

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

```

Arguments

<code>object</code>	An object of class <code>hvti_imputation</code> .
<code>...</code>	Ignored.

Value

A data frame with one row per imputed variable: the variable name, the number of values filled, the proportion of rows that is, and the value used to fill them.

Examples

```

dat <- data.frame(age = c(50, 60, NA, 70), bmi = c(NA, 25, 27, 29))
summary(impute_mean(dat, vars = c("age", "bmi")))

```

summary.hvti_imputation_multi

Summarise what was imputed, across the m completed datasets

Description

Reduces the record to a per-variable table: which method each variable was imputed with, and how many of its values were filled. Unlike `summary.hvti_imputation()`, there is no single `fill_value` column – each of the `m` draws can fill a given cell with a different value, which is the entire point of multiple imputation, so no single value speaks for all of them.

Usage

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

Arguments

<code>object</code>	An object of class <code>hvti_imputation_multi</code> .
<code>...</code>	Ignored.

Value

A data frame with one row per imputed variable: the variable name, the method used, the number of values filled, and the proportion of rows that is.

Examples

```
set.seed(42)
n <- 30
dat <- data.frame(
  age = round(rnorm(n, 60, 8)),
  grp = factor(sample(c("a", "b", "c"), n, replace = TRUE))
)
dat$age[sample(n, 4)] <- NA
dat$grp[sample(n, 4)] <- NA

out <- impute_multiple(dat, vars = c("age", "grp"), m = 2, maxit = 2,
  seed = 1)
summary(out)
```

Index

`analysis_pass(imputation-accessors)`, 4
`analysis_pass()`, 3

`complete_case_pass`
 `(imputation-accessors)`, 4
`complete_case_pass()`, 3, 7, 9

`hvti_imputation`, 2, 3–5, 7, 10
`hvti_imputation_multi`, 3, 9

`imputation-accessors`, 4
`imputation_indicators`, 5
`imputation_provenance`
 `(imputation-accessors)`, 4
`imputation_provenance()`, 2, 3, 7–9
`impute_mean`, 6
`impute_mean()`, 4, 8, 9
`impute_multiple`, 7
`impute_multiple()`, 3, 4
`imputed_any(imputation-accessors)`, 4
`imputed_any()`, 7, 9
`imputed_data(imputation-accessors)`, 4
`imputed_data()`, 2, 3, 7, 9
`imputed_matrix(imputation-accessors)`, 4
`imputed_matrix()`, 2, 3, 7, 9, 10

`kept_by_imputation`
 `(imputation-accessors)`, 4
`kept_by_imputation()`, 5

`RNGkind()`, 9

`summary.hvti_imputation`, 10
`summary.hvti_imputation()`, 2, 11
`summary.hvti_imputation_multi`, 11