Title

stata.com

mi impute — Impute missing values

Description

`mi impute` fills in missing values (`.`) of a single variable or of multiple variables using the specified method. The available methods (by variable type and missing-data pattern) are summarized in the tables below.

Single imputation variable (univariate imputation)

Pattern	Type	Imputation method
always monotone	continuous	`regress`, `pmm`, `truncreg`, `intreg`
	binary	`logit`
	categorical	`ologit`, `mlogit`
	count	`poisson`, `nbreg`

Multiple imputation variables (multivariate imputation)

Pattern	Type	Imputation method
monotone missing	mixture	`monotone`
arbitrary missing	mixture	`chained`
arbitrary missing	continuous	`mvn`

The suggested reading order of `mi impute`'s subentries is

[MI] **mi impute regress**
[MI] **mi impute pmm**
[MI] **mi impute truncreg**
[MI] **mi impute intreg**
[MI] **mi impute logit**
[MI] **mi impute ologit**
[MI] **mi impute mlogit**
[MI] **mi impute poisson**
[MI] **mi impute nbreg**
[MI] **mi impute monotone**
[MI] **mi impute chained**
[MI] **mi impute mvn**
[MI] ***mi impute usermethod***

Menu

Statistics > Multiple imputation

Syntax

```
mi impute method ... [ , impute_options ... ]
```

method	Description
Univariate	
regress	linear regression for a continuous variable
pmm	predictive mean matching for a continuous variable
truncreg	truncated regression for a continuous variable with a restricted range
intreg	interval regression for a continuous partially observed (censored) variable
logit	logistic regression for a binary variable
ologit	ordered logistic regression for an ordinal variable
mlogit	multinomial logistic regression for a nominal variable
poisson	Poisson regression for a count variable
nbreg	negative binomial regression for an overdispersed count variable
Multivariate	
monotone	sequential imputation using a monotone-missing pattern
chained	sequential imputation using chained equations
mvn	multivariate normal regression
User-defined	
usermethod	user-defined imputation methods

impute_options	Description
Main	
* add(#)	specify number of imputations to add; required when no imputations exist
* replace	replace imputed values in existing imputations
rseed(#)	specify random-number seed
double	store imputed values in double precision; the default is to store them as float
by(*varlist* [, *byopts*])	impute separately on each group formed by *varlist* (not allowed with *usermethod*)
Reporting	
dots	display dots as imputations are performed
noisily	display intermediate output
nolegend	suppress all table legends
Advanced	
force	proceed with imputation, even when missing imputed values are encountered
noupdate	do not perform mi update (not allowed with *usermethod*); see [MI] **noupdate option**

*add(#) is required when no imputations exist; add(#) or replace is required if imputations exist.

You must mi set your data before using mi impute; see [MI] mi set.

collect is allowed; see [U] 11.1.10 Prefix commands.

The mi suite of commands does not allow alias variables; see [D] frunalias for advice on how to get around this restriction.

noupdate does not appear in the dialog box.

Options

Main

add(#) specifies the number of imputations to add to the mi data. This option is required if there are no imputations in the data. If imputations exist, then add() is optional. The total number of imputations cannot exceed 1,000.

replace specifies to replace existing imputed values with new ones. One of replace or add() must be specified when mi data already have imputations.

rseed(#) sets the random-number seed. This option can be used to reproduce results. rseed(#) is equivalent to typing set seed # prior to calling mi impute; see [R] set seed. You may also need to use the same stable ordering of the data prior to executing mi impute to reproduce results; see [D] sort.

double specifies that the imputed values be stored as doubles. By default, they are stored as floats. mi impute makes this distinction only when necessary. For example, if the logit method is used, the imputed values are stored as bytes.

by(*varlist*[, *byopts*]) specifies that imputation be performed separately for each by-group. By-groups are identified by equal values of the variables in *varlist* in the original data ($m = 0$). Missing categories in *varlist* are omitted, unless the missing suboption is specified within by(). Imputed and passive variables may not be specified within by(). This option is not allowed with user-defined imputation methods, *usermethod*.

byopts are missing, noreport, nolegend, and nostop.

missing specifies that missing categories in *varlist* are not omitted.

noreport suppresses reporting of intermediate information about each group.

nolegend suppresses the display of group legends that appear before the imputation table when long group descriptions are encountered.

nostop specifies to proceed with imputation when imputation fails in some groups. By default, mi impute terminates with error when this happens.

Reporting

dots specifies to display dots as imputations are successfully completed. An x is displayed if any of the specified imputation variables still have missing values.

noisily specifies that intermediate output from mi impute be displayed.

nolegend suppresses the display of all legends that appear before the imputation table.

Advanced

force specifies to proceed with imputation even when missing imputed values are encountered. By default, mi impute terminates with error if missing imputed values are encountered.

The following option is available with `mi impute` but is not shown in the dialog box:

`noupdate` in some cases suppresses the automatic `mi update` this command might perform; see [MI] **noupdate option**. This option is rarely used and is not allowed with user-defined imputation methods, *usermethod*.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Imputation methods

`mi impute` supports both univariate and multivariate imputation under the missing at random assumption (see *Assumptions about missing data* under *Remarks and examples* in [MI] **Intro substantive**).

Univariate imputation is used to impute a single variable. It can be used repeatedly to impute multiple variables only when the variables are independent and will be used in separate analyses. To impute a single variable, you can choose from the following methods: `regress`, `pmm`, `truncreg`, `intreg`, `logit`, `ologit`, `mlogit`, `poisson`, and `nbreg`; see [MI] **mi impute regress**, [MI] **mi impute pmm**, [MI] **mi impute truncreg**, [MI] **mi impute intreg**, [MI] **mi impute logit**, [MI] **mi impute ologit**, [MI] **mi impute mlogit**, [MI] **mi impute poisson**, and [MI] **mi impute nbreg**.

For a continuous variable, either `regress` or `pmm` can be used (for example, Rubin [1987] and Schenker and Taylor [1996]). For a continuous variable with a restricted range, a truncated variable, either `pmm` or `truncreg` (Raghunathan et al. 2001) can be used. For a continuous partially observed or censored variable, `intreg` can be used (Royston 2007). For a binary variable, `logit` can be used (Rubin 1987). For a categorical variable, `ologit` can be used to impute missing categories if they are ordered, and `mlogit` can be used to impute missing categories if they are unordered (Raghunathan et al. 2001). For a count variable, either `poisson` (Raghunathan et al. 2001) or `nbreg` (Royston 2009), in the presence of overdispersion, is often suggested. Also see van Buuren (2007) for a detailed list of univariate imputation methods.

Theory dictates that multiple variables usually must be imputed simultaneously, and that requires using a multivariate imputation method. The choice of an imputation method in this case also depends on the pattern of missing values.

If variables follow a monotone-missing pattern (see *Patterns of missing data* under *Remarks and examples* in [MI] **Intro substantive**), they can be imputed sequentially using univariate conditional

distributions, which is implemented in the `monotone` method (see [MI] **mi impute monotone**). A separate univariate imputation model can be specified for each imputation variable, which allows simultaneous imputation of variables of different types (Rubin 1987).

When a pattern of missing values is arbitrary, iterative methods are used to fill in missing values. The `mvn` method (see [MI] **mi impute mvn**) uses multivariate normal data augmentation to impute missing values of continuous imputation variables (Schafer 1997). Allison (2001), for example, also discusses how to use this method to impute binary and categorical variables.

Another multivariate imputation method that accommodates arbitrary missing-value patterns is multivariate imputation using chained equations (MICE), also known as imputation using fully conditional specifications (van Buuren, Boshuizen, and Knook 1999) and as sequential regression multivariate imputation (Raghunathan et al. 2001) in the literature. The MICE method is implemented in the `chained` method (see [MI] **mi impute chained**) and uses a Gibbs-like algorithm to impute multiple variables sequentially using univariate fully conditional specifications. Despite a lack of rigorous theoretical justification, the flexibility of MICE has made it one of the most popular choices used in practice.

For a comparison of MICE and multivariate normal imputation, see Lee and Carlin (2010).

Imputation modeling

As discussed in [MI] **Intro substantive**, imputation modeling is important to obtain proper imputations. Imputation modeling is not confined to the specification of an imputation method and an imputation model. It also requires careful consideration of how to handle complex data structures, such as survey or longitudinal data, and how to preserve existing relationships in the data during the imputation step. Rubin (1987), Meng (1994), Schafer (1997), Allison (2001), Royston (2007), Graham (2009), White, Royston, and Wood (2011), and others provide guidelines about imputation modeling. We summarize some of them below.

As with any statistical procedure, choosing an appropriate imputation approach is an art, and the choice should ultimately be determined by your data and research objectives. Regardless of which imputation approach you decide to pursue, it is good practice to check that your imputations are sensible before performing primary data analysis (see *Imputation diagnostics*) and to perform sensitivity analysis (for example, Kenward and Carpenter [2007]).

Model building

Perhaps the most important component of imputation modeling is the construction of an imputation model that preserves all the main characteristics of the observed data. This includes the following:

1. Use as many predictors as possible in the model to avoid making incorrect assumptions about the relationships between the variables. Omitting key predictors from the imputation model may lead to biased estimates for these predictors in the analysis. On the other hand, including insignificant predictors will result in less efficient yet still statistically valid results.

2. Include design variables representing the structure of the data in your imputation model. For example, sampling weights, strata and cluster identifiers of survey data, repeated-measures identifiers of longitudinal data must be included in the imputation model.

3. Specify the correct functional form of an imputation model. For example, include interactions of variables (or impute missing values separately using different subsamples; see *Imputing on subsamples*) to preserve higher-order dependencies.

The imputation model must be compatible with any model that can be used for the analysis. If variable X is to be included in the analysis model, it should also be used in the imputation model. If the analysis model estimates a correlation of X_1 and X_2, then both variables should be present in the imputation model. Accordingly, the outcome variable should always be present in the imputation model. Also, in addition to all the variables that may be used in the analysis model, you should include any auxiliary variables that may contain information about missing data. This will make the MAR assumption more plausible and will improve the quality of the imputed values. For more information about congeniality between the imputation and complete-data models, see Meng (1994).

As we mentioned above, it is important to specify the correct functional form of an imputation model to obtain proper imputations. The failure to accommodate such model features as interactions and nonlinearities during imputation may lead to severely biased results. There is no definitive recommendation for the best way to incorporate various functional forms into the imputation model. Currently, two main approaches are the joint modeling of all functional terms and modeling using passive variables (variables derived from imputation variables) also known as *passive imputation*. The joint modeling approach simply treats all functional terms as separate variables and imputes them together with the underlying imputation variables using a multivariate model, often a multivariate normal model. On the other hand, passive imputation—available within the MICE framework—fills in only the underlying imputation variables and computes the respective functional terms from the imputed variables, maintaining functional dependencies between the imputed and derived variables. The joint modeling approach imposes a rather stringent assumption of multivariate normality for possibly highly nonlinear terms and does not recognize functional dependencies between the imputed and derived variables. The naíve application of passive imputation, however, may omit certain functional relationships and thus lead to biased results. So, careful consideration for the specification of each conditional model is important. See White, Royston, and Wood (2011) for more details and some guidelines.

Outcome variables

Imputing outcome variables receive special attention in the literature because of the controversy about whether they should be imputed. As we already mentioned, it is important to include the outcome variable in the imputation model to obtain valid results. But what if the outcome variable itself has missing values? Should it be imputed? Should missing values be discarded from the analysis? There is no definitive answer to this question. The answer ultimately comes down to whether the specified imputation model describes the missing data adequately. When the percentage of missing values is low, using an incorrect imputation model may have little effect on the resulting repeated-imputation inference. With a large fraction of missing observations, a misspecified imputation model may distort the observed relationship between the outcome and predictor variables. In general, with large fractions of missing observations on any variable, the imputed values have more influence on the results, and thus more careful consideration of the imputation probability model is needed.

Transformations

Although the choice of an imputation method may not have significant impact on the results with low fractions of missing data, it may with larger fractions. A number of different imputation methods are available to model various types of imputation variables: continuous, categorical, count, and so on. However, in practice, these methods in no way cover all possible distributions that imputation variables may have. Often, the imputation variables can be transformed to the scale appropriate for an imputation method. For example, a log transformation (or, more generally, a Box–Cox transformation) can be used for highly skewed continuous variables to make them suitable for imputation using the linear regression method. If desired, the imputed values can be transformed back after the imputation.

Transformations are useful when a variable has a restricted range. For instance, a preimputation logit transformation and a postimputation inverse-logit transformation can be used to ensure that the imputed values are between 0 and 1.

It is important to remember that although the choice of a transformation is often determined based on the variable of interest alone, it is the conditional distribution of that variable given other predictors that is being modeled, and so the transformation must be suitable for it.

Categorical variables

To impute one categorical variable, you can use one of the categorical imputation methods: logistic, ordered logistic, or multinomial logistic regressions (see [MI] **mi impute logit**, [MI] **mi impute ologit**, or [MI] **mi impute mlogit**). These methods can also be used to impute multiple categorical variables with a monotone missing-data pattern using monotone imputation (see [MI] **mi impute monotone**) and with an arbitrary missing-data pattern using MICE (see [MI] **mi impute chained**). Also, for multiple categorical variables with only two categories (binary or dummy variables), a multivariate normal approach (see [MI] **mi impute mvn**) can be used to impute missing values and then, if needed, the imputed values can be rounded to 0 if the value is smaller than 0.5, or 1 otherwise. For categorical variables with more than two categories, Allison (2001) describes how to use the normal model to impute missing values.

The issue of perfect prediction during imputation of categorical data

Perfect prediction (or separation—for example, see Albert and Anderson [1984]) occurs often in the analysis of categorical data. The issue of perfect prediction is inherent to the discrete nature of categorical data and arises in the presence of covariate patterns for which outcomes of a categorical variable can be predicted almost perfectly. Perfect prediction usually leads to infinite coefficients with infinite standard errors and often causes numerical instability during estimation. This issue is often resolved by discarding the observations corresponding to offending covariate patterns as well as the independent variables perfectly predicting outcomes during estimation; see, for example, *Model identification* in [R] **logit**.

Perfect prediction is even more likely to arise during imputation because imputation models, per imputation modeling guidelines, tend to include many variables and thus may include many categorical variables. Perfect prediction may arise when variables are imputed using one of these imputation methods: `logit`, `ologit`, or `mlogit`.

Let's discuss how perfect prediction affects imputation. Recall that to obtain proper imputations (*Proper imputation methods* in [MI] **Intro substantive**), imputed values must be simulated from the posterior predictive distribution of missing data given observed data. The categorical imputation methods achieve this by first drawing a new set of regression coefficients from a normal distribution (a large-sample approximation to their posterior distribution) with mean and variance determined by the maximum likelihood estimates of the coefficients from the observed data and their variance–covariance matrix. The imputed values are then obtained using the new set of coefficients; see *Methods and formulas* in the method-specific manual entries for details.

In the presence of perfect prediction, very large estimates of coefficients and their standard errors arise during estimation. As a result, new coefficients, drawn from the corresponding asymptotic normal distribution, will either be large and positive or large and negative. As such, missing values—say, of a binary imputation variable—may all be imputed as ones in some imputations and may all be imputed as zeros in other imputations. This will clearly bias the multiple-imputation estimate of the proportion of ones (or zeros) in the sample of perfectly predicted cases.

To eliminate the issue of perfect prediction during imputation, we cannot, unfortunately, drop observations and variables when estimating model parameters as is normally done during estimation using, for example, the `logit` command. Doing so would violate one of the main requirements of imputation modeling: all variables and cases that may be used during primary, completed-data analysis must be included in the imputation model. So, what can you do?

When perfect prediction is detected, `mi impute` issues an error message:

```
. mi impute logit x1 z1 z2 ..., ...
mi impute logit: perfect predictor(s) detected
    Variables that perfectly predict an outcome were detected when logit
    executed on the observed data.  First, specify mi impute's option noisily
    to identify the problem covariates.  Then either remove perfect predictors
    from the model or specify mi impute logit's option augment to perform
    augmented regression; see The issue of perfect prediction during imputation
    of categorical data in [MI] mi impute for details.
r(498);
```

You have two alternatives at this point.

You can fit the specified imputation model to the observed data using the corresponding command (in our example, `logit`) to identify the observations and variables causing perfect prediction in your data. Depending on the research objective and specifics of the data collection process, it may be reasonable to omit the offending covariate patterns and perfect predictors from your analysis. If you do so, you must carefully document which observations and variables were removed and adjust your inferential conclusions accordingly. Once offending instances are removed, you can proceed with imputation followed by your primary data analysis. Make sure that the instances you removed from the imputation model are not used in your further analysis.

The above approach may be difficult to pursue when imputing a large number of variables, among which are many categorical variables. Another option is to handle perfect prediction directly during imputation via the `augment` option, which is available for all categorical imputation methods: `logit`, `ologit`, and `mlogit`.

`mi impute ..., augment ...` implements an augmented-regression approach, an ad hoc but computationally convenient approach suggested by White, Daniel, and Royston (2010). According to this approach, a few extra observations with small weights are added to the data during estimation of model parameters in a way that prevents perfect prediction. See White, Daniel, and Royston (2010) for simulation results and computational details.

Convergence of iterative methods

When the missing-value pattern is arbitrary, iterative Markov chain Monte Carlo (MCMC-like) imputation methods are used to simulate imputed values from the posterior predictive distribution of the missing data given the observed data; also see *Multivariate imputation*. In this case, the resulting sequences (chains) of simulated parameters or imputed values should be examined to verify the convergence of the algorithm. The modeling task may be influenced by the convergence process of the algorithm given the data. For example, a different prior distribution for the model parameters may be needed with `mi impute mvn` when some aspects of the model cannot be estimated because of the sparseness of the missing data.

Markov chain simulation is often done in one of two ways: subsampling a single chain or running multiple independent chains. Subsampling a chain involves running a single chain for a prespecified number of iterations T, discarding the first b iterations until the chain reaches stationarity (the burn-in period), and sampling the chain each kth iteration to produce a final sequence of independent draws $\{\mathbf{X}^{(b)}, \mathbf{X}^{(b+k)}, \mathbf{X}^{(b+2k)}, \ldots\}$ from the target distribution. The number of between iterations

k is chosen such that draws $\mathbf{X}^{(t)}$ and $\mathbf{X}^{(t+k)}$ are approximately independent. Alternatively, one can obtain independent draws by running multiple independent chains using different starting values $\{\mathbf{X}^{(i,t)} : t = 0, 1, \ldots\}$, $i = 1, 2, \ldots$, and discarding the first b iterations of each to obtain a final sample $\left\{\mathbf{X}^{(1,b)}, \mathbf{X}^{(2,b)}, \mathbf{X}^{(3,b)}, \ldots\right\}$ from the target distribution.

`mi impute mvn` subsamples the chain, whereas `mi impute chained` runs multiple independent chains; see [MI] **mi impute mvn** and [MI] **mi impute chained** for details on how to monitor convergence of each method.

Imputation diagnostics

After imputation, it is important to examine the sensibility of the obtained imputed values. If any abnormalities are detected, the imputation model must be revised. Diagnostics for imputations is still an ongoing research topic, but two general recommendations are to check model fit of the specified imputation model to the observed data and to compare distributions of the imputed and observed data. To check model fit of an imputation model to the observed data, you can use any standard postestimation tools usually used with that type of model. Also see, for example, [R] **mfp** to help determine an appropriate functional form of the imputation model. The differences (if any) between the distributions of the observed and of the imputed data should be plausible within the context of your study. For more information, see for example, Gelman et al. (2005), Abayomi, Gelman, and Levy (2008), Eddings and Marchenko (2012), and Marchenko and Eddings (2011) for how to perform multiple-imputation diagnostics in Stata.

Using mi impute

To use `mi impute`, you first `mi set` your data; see [MI] **mi set**. Next you register all variables whose missing values are to be imputed; see `mi register` in [MI] **mi set**.

`mi impute` has two main options: `add()` and `replace`. If you do not have imputations, use `add()` to create them. If you already have imputations, you have three choices:

1. Add new imputations to the existing ones by specifying the `add()` option.
2. Add new imputations and also replace the existing ones by specifying both the `add()` and the `replace` options.
3. Replace existing imputed values by specifying the `replace` option.

`add()` is required if no imputations exist in the `mi` data, and either `add()` or `replace` must be specified if imputations exist. See *Univariate imputation* for examples. Note that with `replace`, only imputed values of the specified imputation variables within the specified subsample will be updated.

For reproducibility, use the `rseed()` option to set the random-number seed, or equivalently, set the seed by using `set seed` immediately before calling `mi impute`. If you forget and still have `mi impute`'s stored results in memory, you can retrieve the seed from the stored result `r(rngstate)`; see *Stored results* below. If you sort your data prior to imputation, you may also need to ensure that your sorting is stable for reproducibility of your results; see [D] **sort**.

By default, `mi impute` stores the imputed values using float precision. If you need more accuracy, you can specify the `double` option. Depending on the `mi` data style, the type of the imputed variable may change in the original data, $m = 0$. For example, if your data are in the mlong (or flong) style and you are imputing a binary variable using the regression method, the type of the variable will become `float`. If you are using the logistic method, the type of the variable may become `byte` even if originally your variable was declared as `float` or `int`. `mi impute` will never demote a variable if that would result in loss of precision.

Use the by(*varlist*) option to perform imputation separately on each group formed by *varlist*. Specifying by() is equivalent to the repeated use of an if condition with mi impute to restrict the imputation sample to each of the categories formed by *varlist*. Use the missing option within by() to prevent mi impute from omitting missing categories in *varlist*. By default, mi impute terminates with error if imputation fails in any of the groups; use by()'s nostop option to proceed with imputation. You may not specify imputation and passive variables within by().

mi impute terminates with error if the imputation procedure results in missing imputed values. This may happen if you include variables containing missing values as predictors in your imputation model. If desired, you can override this behavior with the force option.

mi impute may change the sort order of the data.

Univariate imputation

Univariate imputation by itself has limited application in practice. The situations in which only one variable needs to be imputed or in which multiple incomplete variables can be imputed independently are rare in real-data applications. Univariate imputation is most useful when it is used as a building block of sequential multivariate imputation methods; see *Multivariate imputation*. It is thus beneficial to first become familiar with univariate imputation.

Consider the heart attack data in which bmi contains missing values, as described in *A brief introduction to MI using Stata* of [MI] **Intro substantive**. Here we use the already mi set version of the data with a subset of covariates of interest:

```
. use https://www.stata-press.com/data/r18/mheart1s0
(Fictional heart attack data; BMI missing)
. mi describe
Style: mlong
       last mi update 31jan2023 09:23:53, 20 days ago
Observations:
   Complete          132
   Incomplete         22  (M = 0 imputations)
   ----------------------
   Total             154
Variables:
   Imputed: 1; bmi(22)
   Passive: 0
   Regular: 5; attack smokes age female hsgrad
   System:  3; _mi_m _mi_id _mi_miss
   (there are no unregistered variables)
```

According to mi describe, the mi data style is mlong, and the dataset contains no imputations and 22 incomplete observations. The only registered imputed variable is bmi containing the 22 missing values. The other variables are registered as regular. See [MI] **mi describe** for details.

In the example in [MI] **Intro substantive**, we used `mi impute regress` to impute missing values of `bmi`. Let's concentrate on the imputation step in more detail here:

```
. mi impute regress bmi attack smokes age female hsgrad, add(20)
Univariate imputation                       Imputations =       20
Linear regression                                 added =       20
Imputed: m=1 through m=20                       updated =        0

------------------------------------------------------------------
                             |              Observations per m
                             |----------------------------------------
                    Variable |   Complete   Incomplete    Imputed  Total
-----------------------------+----------------------------------------
                         bmi |        132           22         22    154
------------------------------------------------------------------
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

The above output is common to all imputation methods of `mi impute`. In the left column, `mi impute` reports information about which imputation method was used and which imputations were created or updated. The right column contains the total number of imputations, and how many of them are new and how many are updated. The table contains the number of complete, incomplete, and imputed observations, and the total number of observations in the imputation sample, per imputation for each variable (see *Imputation and estimation samples* below). As indicated by the note, complete and incomplete observations sum to the total number of observations. The `imputed` column reports how many incomplete observations were actually imputed. This number represents the minimum across all imputations used ($m = 1$ through $m = 20$ in our example).

In the above example, we used `add(20)` to create 20 new imputations. Suppose that we decided that 20 is not enough and we want to add 30 more:

```
. mi impute regress bmi attack smokes age female hsgrad, add(30)
Univariate imputation                       Imputations =       50
Linear regression                                 added =       30
Imputed: m=21 through m=50                      updated =        0

------------------------------------------------------------------
                             |              Observations per m
                             |----------------------------------------
                    Variable |   Complete   Incomplete    Imputed  Total
-----------------------------+----------------------------------------
                         bmi |        132           22         22    154
------------------------------------------------------------------
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

The table output is unchanged, but the header reports that total number of imputations is now 50. Thirty new imputations (from $m = 21$ to $m = 50$) were added, and the existing 20 imputations were left unchanged.

Suppose that we decide we want to impute bmi using the predictive mean matching (PMM) imputation method instead of the regression method. We use mi impute pmm with five nearest neighbors and specify the replace option to update all existing imputations with new ones:

```
. mi impute pmm bmi attack smokes age female hsgrad, replace knn(5)
Univariate imputation                       Imputations =       50
Predictive mean matching                          added =        0
Imputed: m=1 through m=50                       updated =       50
                                      Nearest neighbors =        5

------------------------------------------------------------------
                             |          Observations per m
                             |----------------------------------
                    Variable |   Complete   Incomplete    Imputed |      Total
-----------------------------+----------------------------------+----------
                         bmi |        132           22         22 |        154
------------------------------------------------------------------
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

The header reports that all 50 existing imputations, from $m = 1$ to $m = 50$, are replaced with new ones.

Later we decide to use more nearest neighbors with mi impute pmm and also add 15 more imputations. We can do the latter by combining replace and add(). We specify replace to update the existing imputations with imputations from PMM with ten nearest neighbors (knn(10)) and use add(15) to add 15 more imputations.

```
. mi impute pmm bmi attack smokes age female hsgrad, add(15) replace knn(10) dots
Imputing m=1 through m=65:
  .........10.........20.........30.........40.........50.........60..... done
Univariate imputation                       Imputations =       65
Predictive mean matching                          added =       15
Imputed: m=1 through m=65                       updated =       50
                                      Nearest neighbors =       10

------------------------------------------------------------------
                             |          Observations per m
                             |----------------------------------
                    Variable |   Complete   Incomplete    Imputed |      Total
-----------------------------+----------------------------------+----------
                         bmi |        132           22         22 |        154
------------------------------------------------------------------
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

The header reports a total of 65 imputations, among which 15 are new and 50 are updated. In this example, we also used the dots option to see the imputation progress. This option is useful with larger datasets to monitor the imputation process.

See *Imputing on subsamples* for other usage of add() and replace.

Multivariate imputation

When imputing multiple variables, their missing-data pattern must first be considered. As we briefly mentioned in *Patterns of missing data* in [MI] **Intro substantive**, when a missing-data pattern is monotone distinct, multiple variables can be imputed sequentially without iteration using univariate conditional models (or monotone imputation). That is, a complicated multivariate imputation task can be replaced with a sequence of simpler univariate imputation tasks; see [MI] **mi impute monotone**.

Monotone missing-data patterns rarely arise naturally in practice. As such, it is important to be able to handle arbitrary missing-data patterns during imputation. Before we describe imputation methods accommodating arbitrary missing-data patterns, we will first discuss the difficulties arising with such patterns during imputation.

Monotone imputation is possible because variables can be ordered such that the complete observations of a variable being imputed are also complete in all prior imputed variables used to predict it. This means that the estimates of the parameters, which are obtained from complete data, do not depend on any previously imputed values (see Rubin [1987] for details). With an arbitrary pattern of missing data, such an ordering may not be possible because some variables may contain incomplete values in observations for which other variables are complete (and vice versa), resulting in estimated parameters being dependent on imputed values. The simultaneous imputation of multiple variables becomes more challenging when missingness is nonmonotone.

Consider the following example. Variable X_1 is complete in observation 1 and missing in observation 2, and variable X_2 is missing in observation 1 and complete in observation 2. We need to impute the two variables simultaneously. Suppose that we impute variable X_2 using previously imputed variable X_1. Observation 1, which contains an imputed value of X_1, is used to estimate the model parameters for X_2. As a result, the model parameters are obtained by treating the imputed value of X_1 as if it were true, thus ignoring the imputation variability in X_1. To account for the uncertainty in the imputed values during estimation, we need to iterate between the estimation step and the imputation step until the estimates of the model parameters depend only on the observed data.

Two main approaches for multivariate imputation with arbitrary missing-data patterns are joint modeling (JM) and fully conditional specification (FCS).

The JM approach assumes a genuine multivariate distribution for all imputation variables and imputes missing values as draws from the resulting posterior predictive distribution of the missing data given the observed data. The predictive distribution is often difficult to draw from directly, so the imputed values are often obtained by approximating this distribution using one of the MCMC methods. One such JM approach for continuous data is based on the multivariate normal distribution, the MVN method (Schafer 1997). The MVN method is implemented in [MI] **mi impute mvn** and uses the data augmentation MCMC method.

The FCS approach does not assume an explicit multivariate distribution for all imputation variables. Instead, it provides a set of chained equations, that is, univariate conditional distributions of each variable with fully conditional specifications of prediction equations. This approach is also known as MICE (van Buuren, Boshuizen, and Knook 1999) or sequential regression multivariate imputation (SRMI; Raghunathan et al. 2001). We will be using the terms MICE, FCS, and SRMI interchangeably throughout the documentation. MICE is similar in spirit to the Gibbs sampler, a popular MCMC method for simulating data from complicated multivariate distributions. Unlike the Gibbs sampler, however, conditional specifications within the MICE method are not guaranteed to correspond to a genuine multivariate distribution because MICE does not start from an explicit multivariate density. Regardless, MICE remains one of the popular imputation methods in practice. The MICE method is implemented in [MI] **mi impute chained**.

Currently, there is no definitive recommendation in the literature as to which approach, JM or FCS, is preferable. The JM approach ensures that imputed values are drawn from a genuine multivariate distribution, and it thus may be more attractive from a theoretical standpoint. However, except for simpler cases such as a multivariate normal model for continuous data, it may not be feasible to formulate a joint model for general data structures. In this regard, the FCS approach is more appealing because it not only can accommodate mixtures of different types of variables, but also can preserve some important characteristics often observed in real data, such as restrictions to subpopulations for certain variables and range restrictions. The tradeoff for such flexibility is a current lack of theoretical

justification. See Lee and Carlin (2010) and references therein for more discussion about the two approaches.

Consider the heart attack data in which both bmi and age contain missing values. Again we will use data that have already been mi set.

```
. use https://www.stata-press.com/data/r18/mheart5s0, clear
(Fictional heart attack data)
. mi describe
Style: mlong
       last mi update 31jan2023 09:23:53, 20 days ago
Observations:
   Complete           126
   Incomplete          28  (M = 0 imputations)
   ----------------------
   Total              154
Variables:
   Imputed: 2; bmi(28) age(12)
   Passive: 0
   Regular: 4; attack smokes female hsgrad
   System:  3; _mi_m _mi_id _mi_miss
   (there are no unregistered variables)
```

There are 28 incomplete observations in the dataset. The bmi variable contains 28 missing values and the age variable contains 12 missing values. Both bmi and age are registered as imputed. If we assume that age and BMI are independent, we can impute each of them separately by using the previously described univariate imputation methods. It is likely, however, that these variables are related, and so we use multivariate imputation.

First, we examine missing-value patterns of the data.

```
. mi misstable patterns
   Missing-value patterns
     (1 means complete)
                 |  Pattern
    Percent      |  1  2
-----------------+---------
        82%      |  1  1
                 |
        10       |  1  0
         8       |  0  0
-----------------+---------
       100%      |
  Variables are  (1) age  (2) bmi
```

From the output, 82% of observations are complete, 10% of observations contain missing values for bmi, and 8% of observations have both bmi and age missing. We can see that the dataset has a monotone-missing pattern (see [MI] **Intro substantive**), that is, missing values of age are nested within missing values of bmi. Another way to see if the pattern of missingness is monotone is to use mi misstable nested ([MI] **mi misstable**):

```
. mi misstable nested
     1.  age(12) -> bmi(28)
```

Because the missing-data pattern is monotone, we can use `mi impute monotone` to impute missing values of `bmi` and `age` simultaneously:

```
. mi impute monotone (regress) age bmi = attack smokes hsgrad female, add(10)
Conditional models:
               age: regress age attack smokes hsgrad female
               bmi: regress bmi age attack smokes hsgrad female

Multivariate imputation                     Imputations =       10
Monotone method                                    added =       10
Imputed: m=1 through m=10                        updated =        0
               age: linear regression
               bmi: linear regression
```

	Observations per m			
Variable	Complete	Incomplete	Imputed	Total
age	142	12	12	154
bmi	126	28	28	154

(Complete + Incomplete = Total; Imputed is the minimum across m of the number of filled-in observations.)

Without going into detail, `mi impute monotone` imputes missing values of multiple variables by performing a sequence of independent univariate conditional imputations. In the above example, the regression method is used to impute missing values of both variables. `age` is imputed first from the observed variables `attack`, `smokes`, `hsgrad`, and `female`. Then `bmi` is imputed using the imputed `age` variable in addition to other observed variables. The output is consistent with that of the univariate imputation methods described earlier, with some additional information. See [MI] **mi impute monotone** for details.

We can also impute missing values of `bmi` and `age` simultaneously using either `mi impute mvn`

```
. mi impute mvn age bmi = attack smokes hsgrad female, replace nolog
Multivariate imputation                     Imputations =       10
Multivariate normal regression                     added =        0
Imputed: m=1 through m=10                        updated =       10
Prior: uniform                               Iterations =     1000
                                                burn-in =      100
                                                between =      100
```

	Observations per m			
Variable	Complete	Incomplete	Imputed	Total
age	142	12	12	154
bmi	126	28	28	154

(Complete + Incomplete = Total; Imputed is the minimum across m of the number of filled-in observations.)

or `mi impute chained`

```
. mi impute chained (regress) age bmi = attack smokes hsgrad female, replace
note: missing-value pattern is monotone; no iteration performed.
Conditional models (monotone):
               age: regress age attack smokes hsgrad female
               bmi: regress bmi age attack smokes hsgrad female
Performing chained iterations ...
Multivariate imputation                     Imputations =       10
Chained equations                                 added =        0
Imputed: m=1 through m=10                       updated =       10
Initialization: monotone                     Iterations =        0
                                                burn-in =        0
               age: linear regression
               bmi: linear regression
```

	Observations per *m*			
Variable	Complete	Incomplete	Imputed	Total
age	142	12	12	154
bmi	126	28	28	154

(Complete + Incomplete = Total; Imputed is the minimum across *m* of the number of filled-in observations.)

Neither `mi impute mvn` nor `mi impute chained` requires the missing-data pattern to be monotone. `mi impute mvn` iterates to produce imputations. When the data are monotone missing, however, no iteration is required, and because `mi impute monotone` executes more quickly, it is preferred. `mi impute chained` also iterates to produce imputations, unless the missing-data pattern is monotone. However, `mi impute monotone` is still faster because it performs estimation only once on the original data, whereas `mi impute chained` performs estimation on each imputation. Use `mi impute mvn` and `mi impute chained` when there is an arbitrary missing-data pattern. See [MI] **mi impute mvn** and [MI] **mi impute chained** for details.

Imputing on subsamples

Consider the earlier example of the univariate imputation of `bmi`. Suppose that we want to perform imputation separately for females and males. Imputation on subsamples is useful when the imputation model must accommodate the interaction effects (see, for example, Allison [2001]). For example, if we want the effect of `bmi` on `attack` to vary by gender, we can perform imputation of `bmi` separately for females and males.

We first show how to do it manually using `if` and the `add()` and `replace` options:

```
. use https://www.stata-press.com/data/r18/mheart1s0, clear
(Fictional heart attack data; BMI missing)
. mi impute regress bmi attack smokes age hsgrad if female==1, add(20)
Univariate imputation                       Imputations =       20
Linear regression                                 added =       20
Imputed: m=1 through m=20                       updated =        0

------------------------------------------------------------------
                             |              Observations per m
                             |----------------------------------------
                    Variable |   Complete   Incomplete    Imputed |  Total
-----------------------------+----------------------------------------
                         bmi |         33            5          5 |     38
------------------------------------------------------------------
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)

. mi impute regress bmi attack smokes age hsgrad if female==0, replace
Univariate imputation                       Imputations =       20
Linear regression                                 added =        0
Imputed: m=1 through m=20                       updated =       20

------------------------------------------------------------------
                             |              Observations per m
                             |----------------------------------------
                    Variable |   Complete   Incomplete    Imputed |  Total
-----------------------------+----------------------------------------
                         bmi |         99           17         17 |    116
------------------------------------------------------------------
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

First, we created 20 imputations and filled in the missing values of `bmi` for females by using the corresponding subset of observations. Then we filled in the remaining missing values of `bmi` for males in the existing imputations by using the subset of male observations. We will now be able to include the interaction between `bmi` and `female` in our logistic model.

A much easier way to do the above is to use by():

```
. use https://www.stata-press.com/data/r18/mheart1s0, clear
(Fictional heart attack data; BMI missing)
. mi impute regress bmi attack smokes age hsgrad, add(20) by(female)
Performing setup for each by() group:
-> female = 0
-> female = 1

Univariate imputation                       Imputations =       20
Linear regression                                 added =       20
Imputed: m=1 through m=20                       updated =        0
```

by() Variable	Observations per *m*			
	Complete	Incomplete	Imputed	Total
female = 0				
bmi	99	17	17	116
female = 1				
bmi	33	5	5	38
Overall				
bmi	132	22	22	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

Conditional imputation

Often in practice, some variables are defined only within what we call a conditional sample, a subset of observations satisfying certain restrictions (Raghunathan et al. 2001, Royston 2009). For example, the number of cigarettes smoked is relevant to smokers only, the number of pregnancies is relevant to females only, etc. Outside the conditional sample, such variables are assumed to contain soft missing values and a nonmissing constant value, further referred to as a conditional constant, which represents a known value or an inadmissible value. We will refer to conditional imputation as imputation of such variables. So, the task of conditional imputation is to impute missing values of a variable within a conditional sample using only observations from that sample and to replace missing values outside the conditional sample with a conditional constant.

In the previous section, we learned that we can specify an `if` condition with `mi impute` to restrict imputation of variables to a subset of observations. Is this sufficient to accommodate conditional imputation? To answer this question, let's consider several examples.

We use our heart attack data as an example. Suppose that our only variable containing missing values is `hightar`, the indicator for smoking high-tar cigarettes. We want to impute missing values in `hightar` and use it among other predictors in the logistic analysis of heart attacks. Because `hightar` is relevant to smokers only, we want to impute `hightar` using the subset of observations with `smokes==1`.

Thus to impute `hightar`, we restrict our imputation sample to smokers:

```
. mi impute logit hightar attack age bmi ... if smokes==1, ...
```

Are we now ready to proceed with our primary logistic analysis of heart attacks? Not quite. Recall that we wish to use all observations of hightar in our analysis. If hightar contains missing values only in the conditional sample, smokes==1, we are finished. Otherwise, we need to replace all remaining missing values outside the conditional sample, for smokes==0, with the conditional constant, the nonmissing value of hightar in observations with smokes==0. In our example, this value is zero, so our final step is

```
. mi xeq: replace hightar = 0 if smokes==0
```

What if we have several imputation variables? Suppose now that age and bmi also contain missing values. Without making any assumptions about a missing-data pattern, we use mi impute chained to impute variables of different types: age, bmi, and hightar. We need to impute hightar for smokes==1 but use the unrestricted sample to impute age and bmi. Can we still accomplish this by specifying an if condition? The answer is yes, but we need to replace missing values of hightar for smokes==0 before imputation to ensure that age and bmi are imputed properly, using all observations, when hightar is used in their prediction equations:

```
. mi xeq: replace hightar = 0 if smokes==0
. mi impute chained (regress) bmi age (logit if smokes==1) hightar = ..., ...
```

It seems that we can get away with using if to perform conditional imputation. What is the catch? So far, we assumed that smokes does not contain any missing values. Let's see what happens if it does.

Because hightar depends on smokes, we must first impute missing values of smokes before we can impute missing values of hightar. As such, the set of observations for which smokes==1 will vary from imputation to imputation and, in the case of mi impute chained, from iteration to iteration. The replacement of missing values of hightar outside the conditional sample should be performed each time a new set of imputed values is obtained for smokes, and thus must be directly incorporated into the imputation procedure.

The answer to our earlier question about using an if condition to perform conditional imputation is no, in general. To perform conditional imputation, use the conditional() option:

```
. mi imp chained (reg) bmi age (logit) smokes (logit, conditional(if smokes==1))
> hightar ...
```

Every univariate imputation method supports option conditional(). This option is most useful within specifications of univariate methods when multiple variables are being imputed using mi impute monotone or mi impute chained, as we showed above. Although in some cases, as we saw earlier, specifying an if condition in combination with manual replacement of missing values outside the conditional sample may produce equivalent results, such use should generally be avoided and conditional() should be used instead.

When you specify option conditional(), mi impute performs checks necessary for proper conditional imputation. For example, the imputed variable is verified to be constant outside the conditional sample and an error message is issued if it is not:

```
. mi impute logit hightar age bmi ..., conditional(if smokes==1)
conditional(): imputation variable not constant outside conditional sample;
    hightar is not constant outside the subset identified by (smokes==1)
    within the imputation sample.  This may happen when missing values of
    conditioning variables are not nested within missing values of hightar.
r(459);
```

mi impute also requires that missing values of all variables involved in conditional specifications (restrictions)—that is, conditioning variables—be nested within missing values of the conditional variable being imputed. If this does not hold true, mi impute issues an error message:

```
. mi impute logit hightar age bmi ..., conditional(if smokes==1)
conditional(): conditioning variables not nested;
    conditioning variable smokes is not nested within hightar
r(459);
```

Because missing values of all conditioning variables are assumed to be nested within missing values of a conditional variable, that conditional variable is not included in the prediction equations of the corresponding conditioning variables.

As an example, let's continue with our heart attack data, in which variables hightar and smokes contain missing values, as do age and bmi:

```
. use https://www.stata-press.com/data/r18/mheart7s0, clear
(Fictional heart attack data; BMI, age, hightar, and smokes missing)
. mi describe
Style: mlong
       last mi update 31jan2023 09:23:53, 20 days ago
Observations:
   Complete          124
   Incomplete         30  (M = 0 imputations)
   ---------------------
   Total             154
Variables:
   Imputed: 4; bmi(24) age(30) hightar(8) smokes(5)
   Passive: 0
   Regular: 3; attack female hsgrad
   System:  3; _mi_m _mi_id _mi_miss
   (there are no unregistered variables)
. mi misstable nested
     1.  smokes(5) -> hightar(8) -> bmi(24) -> age(30)
```

Our data are already mi set, so we proceed with imputation. According to mi misstable nested, all imputation variables are monotone missing, so we use mi impute monotone for imputation. For the purpose of illustration, we create only two imputations:

```
. mi impute monotone (regress) bmi age
>                    (logit, conditional(if smokes==1)) hightar
>                    (logit) smokes
>                                    = attack hsgrad female, add(2)

Conditional models:
            smokes: logit smokes attack hsgrad female
           hightar: logit hightar i.smokes attack hsgrad female ,
                     conditional(if smokes==1)
               bmi: regress bmi i.hightar i.smokes attack hsgrad female
               age: regress age bmi i.hightar i.smokes attack hsgrad female
note: 1.smokes omitted because of collinearity.

Multivariate imputation                     Imputations =        2
Monotone method                                   added =        2
Imputed: m=1 through m=2                        updated =        0

Conditional imputation:
  hightar: incomplete out-of-sample obs replaced with value 0

               bmi: linear regression
               age: linear regression
           hightar: logistic regression
            smokes: logistic regression
```

	Observations per *m*			
Variable	Complete	Incomplete	Imputed	Total
bmi	130	24	24	154
age	124	30	30	154
hightar	146	8	8	154
smokes	149	5	5	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

For each variable that was imputed conditionally, `mi impute` reports the conditional value used to replace all missing observations outside the conditional sample in a legend about conditional imputation. In our example, all missing values of `hightar` outside `smokes==1` are replaced with zero. The reported numbers of complete, incomplete, and imputed observations for `hightar` correspond to the entire imputation sample (see *Imputation and estimation samples*) and not only to the conditional sample. For example, there are 146 complete and 8 incomplete observations of `hightar` in the combined sample of smokers and nonsmokers. The minimum number of imputed values across imputations is 8, so all incomplete observations of `hightar` were filled in—either imputed directly or replaced with a conditional value—in both imputations. Because `smokes` is being imputed, the numbers of incomplete and imputed observations of `hightar` for smokers and nonsmokers will vary across imputations.

You can accommodate more complicated restrictions or skip patterns, which often arise with questionnaire data, by specifying more elaborate restrictions within `conditional()` or by specifying the `conditional()` option with other variables. For example, suppose that the information about cigarette tar level (`hightar`) was collected only for heavy smokers, identified by an indicator variable `heavysmoker`. The `heavysmoker` variable contains missing values and needs to be imputed before `hightar` can be imputed. To impute `heavysmoker`, we need to restrict our sample to smokers only. Then to impute `hightar`, we need to use only heavy smokers among all smokers. We can do so as follows:

```
. mi impute chained (logit) smokes                                          ///
                    (logit, conditional(if smokes==1)) heavysmoker          ///
                    (logit, conditional(if smokes==1 & heavysmoker==1))     ///
                         hightar ...
```

Imputation and estimation samples

Rubin (1987, 160–166) describes the imputation process as three tasks: modeling, estimation, and imputation. We concentrate on the latter two tasks here. The posterior distribution of the model parameters is estimated during the estimation task. This posterior distribution is used in the imputation task to simulate the parameters of the posterior predictive distribution of the missing data from which an imputed value is drawn. Accordingly, `mi impute` distinguishes between two main samples: imputation and estimation.

The imputation sample is determined by the imputation variables used in the imputation task. It is comprised of all observations for which the imputation variables contain no hard missing values (or no extended missing values). In other words, the imputation sample consists of the complete and incomplete observations as identified by the specified imputation variables. The estimation sample is comprised of all observations used by the model fit to the observed data during the estimation task.

For example,

```
. use https://www.stata-press.com/data/r18/mheart1s0, clear
(Fictional heart attack data; BMI missing)
. mi impute regress bmi attack smokes age hsgrad female, add(1) noisily
Running regress on observed data:
```

Source	SS	df	MS	Number of obs	=	132
				F(5, 126)	=	1.24
Model	99.5998228	5	19.9199646	Prob > F	=	0.2946
Residual	2024.93667	126	16.070926	R-squared	=	0.0469
				Adj R-squared	=	0.0091
Total	2124.5365	131	16.2178358	Root MSE	=	4.0089

bmi	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
attack	1.71356	.7515229	2.28	0.024	.2263179	3.200801
smokes	-.5153181	.761685	-0.68	0.500	-2.02267	.9920341
age	-.033553	.0305745	-1.10	0.275	-.0940591	.026953
hsgrad	-.4674308	.8112327	-0.58	0.566	-2.072836	1.137975
female	-.3072767	.8074763	-0.38	0.704	-1.905249	1.290695
_cons	26.96559	1.884309	14.31	0.000	23.2366	30.69458

```
Univariate imputation                       Imputations =        1
Linear regression                                 added =        1
Imputed: m=1                                    updated =        0
```

	Observations per *m*			
Variable	Complete	Incomplete	Imputed	Total
bmi	132	22	22	154

```
(Complete + Incomplete = Total; Imputed is the minimum across m
 of the number of filled-in observations.)
```

The imputation sample contains 154 observations and the estimation sample contains 132 observations (from the regression output). The estimation task of `mi impute regress` consists of fitting a linear regression of `bmi` on other variables to the observed data. We specified the `noisily` option to see results from the estimation task. Usually, the number of complete observations in the imputation sample (132 in this example) will be equal to the number of observations used in the estimation. Sometimes, however, observations may be dropped from the estimation—for example, if independent variables contain missing values. In this case, the number of complete observations in the imputation

sample and the number of observations used in the estimation will be different, and the following note will appear following the table output:

```
Note: Right-hand-side variables (or weights) have missing values;
      model parameters estimated using listwise deletion.
```

You should evaluate such cases to verify that results are as expected.

In general, missing values in independent variables (or in a weighting variable) do not affect the imputation sample but they may lead to missing imputed values. In the above example, if `age` contained missing values in incomplete observations of `bmi`, the linear prediction for those observations would have been missing and thus the resulting imputed values would have been missing, too.

Imputing on subsamples, or in other words, using an `if` condition with `mi impute`, restricts both imputation and estimation samples to include only observations satisfying the `if` condition. Conditional imputation (the `conditional()` option), on the other hand, affects only the estimation sample. All values, within and outside of a conditional sample, except extended missing values, are included in the imputation sample. With conditional imputation, the reported number of complete observations will almost always be different from the number of observations in the estimation sample, unless the conditional sample coincides with the imputation sample. In the case of observations being dropped from a conditional sample during estimation, a note as shown above will appear following the table output.

Imputing transformations of incomplete variables

Continuing with the univariate example above, say that we discover that the distribution of `bmi` is skewed to the right, and thus we decide to impute `bmi` on the logarithmic scale instead of the original one. We can do this by creating a new variable, `lnbmi`, and imputing it instead of `bmi`.

What we will do is create `lnbmi`, register it as imputed, impute it, and then create `bmi` as a passive variable based on the formula $\texttt{bmi} = \exp(\texttt{lnbmi})$.

We need to be careful when we create `lnbmi` to get its missing values right. `mi` respects two kinds of missing values, called soft and hard missing. Soft missing values are missing values eligible for imputation. Hard missing values are missing values that are to remain missing even in the imputed data. Soft missing are recorded as ordinary missing (`.`), and hard missing are recorded as any of extended missing (`.a`–`.z`).

The issue here is that missing values could arise because of our application of the transform $\texttt{lnbmi} = \ln(\texttt{bmi})$. In the case of the $\ln()$ transform, missing values will be created whenever $\texttt{bmi} \leq 0$. (In general, transformations leading to undefined values should be avoided so that all available observed data are used during imputation.) Body mass index does not contain such values, but let's pretend it did. Here is what we would do:

1. Create $\texttt{lnbmi} = \ln(\texttt{bmi})$.
2. Replace `lnbmi` to contain `.z` in observations for which `lnbmi` contains missing but `bmi` does not.
3. Register `lnbmi` as an imputed variable and impute it.
4. Create passive variable $\texttt{newbmi} = \exp(\texttt{lnbmi})$.
5. Replace `newbmi` equal to `bmi` in observations for which `newbmi` is missing and `bmi` is not.

Alternatively, to avoid creating hard missing values in step 2, we could consider a different transformation; see, for example, [R] **lnskew0**.

As we said, for lnbmi = ln(bmi) we need not perform all the steps above because bmi > 0. In the bmi case, all we need to do is

1. Create lnbmi = ln(bmi).
2. Register lnbmi as an imputed variable and impute it.
3. Create passive variable newbmi = exp(lnbmi).

If all we wanted to do was impute lnbmi = ln(bmi) and, from that point on, just work with lnbmi, we would perform only the first two steps of the three-step procedure.

All that said, we are going to perform the five-step procedure because it will always work. We will continue from where we left off in the last example, so we will discard our previous imputation efforts by typing mi set M = 0. (Instead of typing mi set M = 0, we could just as easily begin by typing use https://www.stata-press.com/data/r18/mheart1s0.)

```
. mi set M = 0                                  // start again
. mi unregister bmi                             // we do not impute bmi
. generate lnbmi = ln(bmi)                      // create lnbmi
. replace lnbmi = .z if lnbmi==. & bmi!=.
. mi register imputed lnbmi
. mi impute regress lnbmi attack smokes age hsgrad female, add(5)
. mi passive: generate newbmi = exp(lnbmi)
. mi passive: replace newbmi = bmi if bmi!=.
```

The important thing about the above is the mechanical definition of an imputed variable. An imputed variable is a variable we actually impute, not a variable we desire to impute. In this case, we imputed lnbmi and derived bmi from it. Thus the variable we desired to impute became, mechanically, a passive variable.

Stored results

mi impute stores the following in r():

Scalars

r(M)	total number of imputations
r(M_add)	number of added imputations
r(M_update)	number of updated imputations
r(k_ivars)	number of imputed variables
r(N_g)	number of imputed groups (1 if by() is not specified)

Macros

r(method)	name of imputation method
r(ivars)	names of imputation variables
r(rngstate)	random-number state used
r(by)	names of variables specified within by()

Matrices

r(N)	number of observations in imputation sample in each group (per variable)
r(N_complete)	number of complete observations in imputation sample in each group (per variable)
r(N_incomplete)	number of incomplete observations in imputation sample in each group (per variable)
r(N_imputed)	number of imputed observations in imputation sample in each group (per variable)

Also see *Stored results* in the method-specific manual entries for additional stored results.

Methods and formulas

All imputation methods (except predictive mean matching) are based on simulating from a Bayesian (approximate) posterior predictive distribution of missing data. Univariate imputation methods and the sequential monotone method use noniterative techniques for simulating from the posterior predictive distribution of missing data. The imputation method based on multivariate normal regression uses an iterative MCMC technique to simulate from the posterior predictive distribution of missing data. The MICE method uses a Gibbs-like algorithm to obtain imputed values.

See *Methods and formulas* in the method-specific manual entries for details.

Herman Otto Hartley (1912–1980) was born in Germany as Herman Otto Hirschfeld and immigrated to England in 1934 after completing his PhD in mathematics at Berlin University. He completed a second PhD in mathematical statistics under John Wishart at Cambridge in 1940 and went on to hold positions at Harper Adams Agricultural College, Scientific Computing Services (London), University College (London), Iowa State College, Texas A&M University, and Duke University. Among other awards he received and distinguished titles he held, Professor Hartley served as the president of the American Statistical Association in 1979. Known affectionately as HOH by almost all who knew him, he founded the Institute of Statistics, later to become the Department of Statistics, at Texas A&M University. His contributions to statistical computing are particularly notable considering the available equipment at the time. Professor Hartley is best known for his two-volume *Biometrika Tables for Statisticians* (jointly written with Egon Pearson) and for his fundamental contributions to sampling theory, missing-data methodology, variance-component estimation, and computational statistics.

References

Abayomi, K. A., A. Gelman, and M. Levy. 2008. Diagnostics for multivariate imputations. *Journal of the Royal Statistical Society, Series C* 57: 273–291. https://doi.org/10.1111/j.1467-9876.2007.00613.x.

Albert, A., and J. A. Anderson. 1984. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika* 71: 1–10. https://doi.org/10.2307/2336390.

Allison, P. D. 2001. *Missing Data*. Thousand Oaks, CA: Sage.

Aloisio, K. M., N. Micali, S. A. Swanson, A. Field, and N. J. Horton. 2014. Analysis of partially observed clustered data using generalized estimating equations and multiple imputation. *Stata Journal* 14: 863–883.

Bartlett, J. W., and T. P. Morris. 2015. Multiple imputation of covariates by substantive-model compatible fully conditional specification. *Stata Journal* 15: 437–456.

Eddings, W. D., and Y. V. Marchenko. 2012. Diagnostics for multiple imputation in Stata. *Stata Journal* 12: 353–367.

Gelman, A., I. Van Mechelen, G. Verbeke, D. F. Heitjan, and M. Meulders. 2005. Multiple imputation for model checking: Completed-data plots with missing and latent data. *Biometrics* 61: 74–85. https://doi.org/10.1111/j.0006-341X.2005.031010.x.

Graham, J. W. 2009. Missing data analysis: Making it work in the real world. *Annual Review of Psychology* 60: 549–576. https://doi.org/10.1146/annurev.psych.58.110405.085530.

Halpin, B. 2016. Multiple imputation for categorical time series. *Stata Journal* 16: 590–612.

Kenward, M. G., and J. R. Carpenter. 2007. Multiple imputation: Current perspectives. *Statistical Methods in Medical Research* 16: 199–218. https://doi.org/10.1177/0962280206075304.

Lee, K. J., and J. B. Carlin. 2010. Multiple imputation for missing data: Fully conditional specification versus multivariate normal imputation. *American Journal of Epidemiology* 171: 624–632. https://doi.org/10.1093/aje/kwp425.

Marchenko, Y. V., and W. D. Eddings. 2011. A note on how to perform multiple-imputation diagnostics in Stata. https://www.stata.com/users/ymarchenko/midiagnote.pdf.

Meng, X.-L. 1994. Multiple-imputation inferences with uncongenial sources of input (with discussion). *Statistical Science* 9: 538–573. https://doi.org/10.1214/ss/1177010269.

Raghunathan, T. E., J. M. Lepkowski, J. Van Hoewyk, and P. Solenberger. 2001. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey Methodology* 27: 85–95.

Royston, P. 2007. Multiple imputation of missing values: Further update of ice, with an emphasis on interval censoring. *Stata Journal* 7: 445–464.

——. 2009. Multiple imputation of missing values: Further update of ice, with an emphasis on categorical variables. *Stata Journal* 9: 466–477.

Rubin, D. B. 1987. *Multiple Imputation for Nonresponse in Surveys*. New York: Wiley.

Schafer, J. L. 1997. *Analysis of Incomplete Multivariate Data*. Boca Raton, FL: Chapman and Hall/CRC.

Schenker, N., and J. M. G. Taylor. 1996. Partially parametric techniques for multiple imputation. *Computational Statistics and Data Analysis* 22: 425–446. https://doi.org/10.1016/0167-9473(95)00057-7.

van Buuren, S. 2007. Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research* 16: 219–242. https://doi.org/10.1177/0962280206074463.

van Buuren, S., H. C. Boshuizen, and D. L. Knook. 1999. Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine* 18: 681–694. https://doi.org/10.1002/(SICI)1097-0258(19990330)18:6⟨681::AID-SIM71⟩3.0.CO;2-R.

White, I. R., R. M. Daniel, and P. Royston. 2010. Avoiding bias due to perfect prediction in multiple imputation of incomplete categorical data. *Computational Statistics and Data Analysis* 54: 2267–2275. https://doi.org/10.1016/j.csda.2010.04.005.

White, I. R., P. Royston, and A. M. Wood. 2011. Multiple imputation using chained equations: Issues and guidance for practice. *Statistics in Medicine* 30: 377–399. https://doi.org/10.1002/sim.4067.

Also see

[MI] **mi estimate** — Estimation using multiple imputations

[MI] **Intro** — Introduction to mi

[MI] **Intro substantive** — Introduction to multiple-imputation analysis

[MI] **Glossary**

[D] **frunalias** — Change storage type of alias variables