mi predict — Obtain multiple-imputation predictions

Description

`mi predict using` *miestfile* is for use after `mi estimate, saving(`*miestfile*`):` ... to obtain multiple-imputation (MI) linear predictions or their standard errors.

`mi predictnl using` *miestfile* is for use after `mi estimate, saving(`*miestfile*`):` ... to obtain MI (possibly) nonlinear predictions, their standard errors, and other statistics, including statistics specific to MI.

MI predictions, their standard errors, and other statistics are obtained by applying Rubin's combination rules observationwise to the completed-data predictions, predictions computed for each imputation (White, Royston, and Wood 2011). The results are stored in the original data ($m = 0$). See [R] **predict** and [R] **predictnl** for details about the computation of the completed-data predictions.

`mi predict` and `mi predictnl` may change the sort order of the data.

Menu

Statistics > Multiple imputation

Syntax

Obtain multiple-imputation linear predictions

`mi predict` [*type*] *newvar* [*if*] `using` *miestfile* [, *predict_options options*]

Obtain multiple-imputation nonlinear predictions

`mi predictnl` [*type*] *newvar* = *pnl_exp* [*if*] `using` *miestfile* [, *pnl_options options*]

miestfile`.ster` contains estimation results previously saved by `mi estimate, saving(`*miestfile*`)`; see [MI] **mi estimate**.

pnl_exp is any valid Stata expression and may also contain calls to two special functions unique to `predictnl`: `predict()` and `xb()`; see [R] **predictnl** for details.

predict_options	Description
Predict options	
`xb`	calculate linear prediction; the default
`stdp`	calculate standard error of the prediction
`nooffset`	ignore any `offset()` or `exposure()` variable
`equation(`*eqno*`)`	specify equations after multiple-equation commands

pnl_options	Description
Predict options	
`se(`*newvar*`)`	create *newvar* containing standard errors
`variance(`*newvar*`)`	create *newvar* containing variances
`wald(`*newvar*`)`	create *newvar* containing the Wald test statistic
`p(`*newvar*`)`	create *newvar* containing the significance level (*p*-value) of the Wald test
`ci(`*newvars*`)`	create *newvars* containing lower and upper confidence intervals
`level(`#`)`	set confidence level; default is `level(95)`
`bvariance(`*newvar*`)`	create *newvar* containing between-imputation variances
`wvariance(`*newvar*`)`	create *newvar* containing within-imputation variances
`df(`*newvar*`)`	create *newvar* containing MI degrees of freedom
`nosmall`	do not apply small-sample correction to degrees of freedom
`rvi(`*newvar*`)`	create *newvar* containing relative variance increases
`fmi(`*newvar*`)`	create *newvar* containing fractions of missing information
`re(`*newvar*`)`	create *newvar* containing relative efficiencies
Advanced	
`iterate(`#`)`	maximum iterations for finding optimal step size to compute completed-data numerical derivatives of *pnl_exp*; default is 100
`force`	calculate completed-data standard errors, etc., even when possibly inappropriate

options	Description
MI options	
`nimputations(`#`)`	specify number of imputations to use in computation; default is to use all existing imputations
`imputations(`*numlist*`)`	specify which imputations to use in computation
`estimations(`*numlist*`)`	specify which estimation results to use in computation
`esample(`*varname*`)`	restrict the prediction to the estimation subsample identified by a binary variable *varname*
`storecompleted`	store completed-data predictions in the imputed data; available only in the flong and flongsep styles
Reporting	
`replay`	replay command-specific results from each individual estimation in *miestfile*`.ster`
`cmdlegend`	display the command legend
`noupdate`	do not perform `mi update`; see [MI] **noupdate option**
`noerrnotes`	suppress error notes associated with failed estimation results in *miestfile*`.ster`
`showimputations`	show imputations saved in *miestfile*`.ster`

`noupdate`, `noerrnotes`, and `showimputations` do not appear in the dialog box.

Options

Predict options

`xb`, `stdp`, `nooffset`, `equation(`*eqno*`)`; see [R] **predict**.

`se(`*newvar*`)`, `variance(`*newvar*`)`, `wald(`*newvar*`)`, `p(`*newvar*`)`, `ci(`*newvars*`)`, `level(#)`; see [R] **predictnl**. These options store the specified MI statistics in variable *newvar* in the original data ($m = 0$). `level()` is relevant in combination with `ci()` only. If `storecompleted` is specified, then *newvar* contains the respective completed-data estimates in the imputed data ($m > 0$). Otherwise, *newvar* is missing in the imputed data.

`bvariance(`*newvar*`)` adds *newvar* of storage type *type*, where for each `i` in the prediction sample, *newvar*`[i]` contains the estimated between-imputation variance of *pnl_exp*`[i]`. `storecompleted` has no effect on `bvariance()`.

`wvariance(`*newvar*`)` adds *newvar* of storage type *type*, where for each `i` in the prediction sample, *newvar*`[i]` contains the estimated within-imputation variance of *pnl_exp*`[i]`. `storecompleted` has no effect on `wvariance()`.

`df(`*newvar*`)` adds *newvar* of storage type *type*, where for each `i` in the prediction sample, *newvar*`[i]` contains the estimated MI degrees of freedom of *pnl_exp*`[i]`. If `storecompleted` is specified, then *newvar* in the imputed data will contain the complete-data degrees of freedom as saved by `mi estimate`. In the absence of the complete-data degrees of freedom or if `nosmall` is used, then *newvar* is missing in the imputed data, even if `storecompleted` is specified.

`nosmall` specifies that no small-sample correction be made to the degrees of freedom. By default, the small-sample correction of Barnard and Rubin (1999) is used. This option has an effect on the results stored by `p()`, `ci()`, `df()`, `fmi()`, and `re()`.

`rvi(`*newvar*`)` adds *newvar* of storage type *type*, where for each `i` in the prediction sample, *newvar*`[i]` contains the estimated relative variance increase of *pnl_exp*`[i]`. `storecompleted` has no effect on `rvi()`.

`fmi(`*newvar*`)` adds *newvar* of storage type *type*, where for each `i` in the prediction sample, *newvar*`[i]` contains the estimated fraction of missing information of *pnl_exp*`[i]`. `storecompleted` has no effect on `fmi()`.

`re(`*newvar*`)` adds *newvar* of storage type *type*, where for each `i` in the prediction sample, *newvar*`[i]` contains the estimated relative efficiency of *pnl_exp*`[i]`. `storecompleted` has no effect on `re()`.

MI options

`nimputations(#)` specifies that the first # imputations be used; # must be $2 \leq \# \leq M$. The default is to use all imputations, M. Only one of `nimputations()`, `imputations()`, or `estimations()` may be specified.

`imputations(`*numlist*`)` specifies which imputations to use. The default is to use all of them. *numlist* must contain at least two numbers corresponding to the imputations saved in *miestfile*`.ster`. You can use the `showimputations` option to display imputations currently saved in *miestfile*`.ster`. Only one of `nimputations()`, `imputations()`, or `estimations()` may be specified.

`estimations(`*numlist*`)` does the same thing as `imputations(`*numlist*`)`, but this time the imputations are numbered differently. Say that *miestfile*`.ster` was created by `mi estimate` and `mi estimate` was told to limit itself to imputations 1, 3, 5, and 9. With `imputations()`, the imputations are still numbered 1, 3, 5, and 9. With `estimations()`, they are numbered 1, 2, 3, and 4. Usually, one does

not specify a subset of imputations when using mi estimate, and so usually, the imputations() and estimations() options are identical. The specified *numlist* must contain at least two numbers. Only one of nimputations(), imputations(), or estimations() may be specified.

esample(*varname*) restricts the prediction to the estimation sample identified by a binary variable *varname*. By default, predictions are obtained for all observations in the original data. Variable *varname* cannot be registered as imputed or passive and cannot vary across imputations.

storecompleted stores completed-data predictions in the newly created variables in each imputation. By default, the imputed data contain missing values in the newly created variables. The storecompleted option may be specified only if the data are flong or flongsep; see [MI] **mi convert** to convert to one of those styles.

Reporting

replay replays estimation results from *miestfile*.ster, previously saved by mi estimate, saving(*miestfile*).

cmdlegend requests that the command line corresponding to the estimation command used to produce the estimation results saved in *miestfile*.ster be displayed.

Advanced

iterate(#), force; see [R] **predictnl**.

The following options are available with mi predict and mi predictnl but are 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.

noerrnotes suppresses notes about failed estimation results. These notes appear when *miestfile*.ster contains estimation results, previously saved by mi estimate, saving(*miestfile*), from imputations for which the estimation command used with mi estimate failed to estimate parameters.

showimputations displays imputation numbers corresponding to the estimation results saved in *miestfile*.ster.

Remarks and examples

Remarks are presented under the following headings:

Introduction

Various predictions are often of interest after estimation. Within the MI framework, one must first decide what prediction means. There is no single dataset with respect to which prediction is made. Rather, there are multiple datasets in which values of imputed predictors vary from one dataset to another.

One definition is simply to consider an observation-specific prediction to be a parameter of interest and apply Rubin's combination rules to it as to any other estimand (White, Royston, and Wood 2011). The next thing to decide is what types of predictions are appropriate for pooling. For any parameter, the applicability of combination rules is subject to a number of conditions that the parameter must satisfy. One of them is asymptotic normality of the completed-data estimates of the parameter; see, for example, *Theory underlying multiple imputation* under *Remarks and examples* of [MI] **Intro substantive** for a full set of conditions.

It is safe to apply combination rules to the linear predictor, as computed by `mi predict`. It is also safe to apply combination rules to functions, possibly nonlinear, of the linear predictor, provided the sampling distribution of that function is asymptotically normal. This can be done by using `mi predictnl`. `mi predictnl` also provides, with the `predict()` specification, a way of obtaining MI estimates for various types of predictions specific to each estimation command used with `mi estimate`. Care should be taken when using this functionality. Some predictions may require preliminary transformation to a scale that improves normality, which is more appropriate for pooling. The obtained MI estimates of predictions may then be back-transformed to obtain final predictions in the original metric. For example, one can obtain MI estimates of probabilities of a positive outcome after logistic estimation by pooling the completed-data estimates of the actual probabilities. A better approach is to pool the completed-data estimates of the linear predictor and then apply an inverse-logit transformation to obtain the probability of a positive outcome. Other available predictions, such as standard errors, may not even be applicable for pooling.

The MI predictions should be treated as a final result; they should not be used as intermediate results in computations. For example, MI estimates of the linear predictor cannot be used to compute residuals as is done in non-MI analysis. Instead, completed-data residuals should be calculated for each imputed dataset, and these can be obtained by using the `mi xeq:` command. For example,

```
. mi xeq: regress ...; predict resid, r
```

Because completed-data predictions are super varying, they should only be computed in the flong or flongsep styles.

Using mi predict and mi predictnl

`mi predict` and `mi predictnl` require that completed-data estimation results saved by `mi estimate, saving()` are supplied with the `using` specification and that the `mi` data used to obtain these results are in memory. Apart from this, the use of these commands is similar to that of their non-`mi` counterparts, `predict` and `predictnl` (see [R] **predict** and [R] **predictnl**).

By default, `mi predict` computes MI linear predictions. If the `stdp` option is specified, `mi predict` computes standard errors of the MI linear predictions. As with `predict`, the `equation()` option can be used with `mi predict` after multiple-equation commands to obtain linear predictions or their standard errors from a specific equation.

Similarly to `predictnl`, a number of statistics associated with predictions can be obtained with `mi predictnl`, such as confidence intervals and p-values. Additionally, a number of MI statistics, such as relative variance increases and fractions of missing information, are available with `mi predictnl`. As we mentioned in *Introduction*, the `predict()` function of `mi predictnl` offers a variety of predictions. However, you should carefully consider whether the requested prediction is applicable for pooling or, perhaps, needs a preliminary transformation to improve normality.

Unlike `predict`, `mi predict` always defaults to the linear prediction. It supports only the linear prediction or its standard error and does not support any other command-specific predictions. Command-specific predictions appropriate for pooling may be obtained with the `predict()` function

of mi predictnl. Also unlike predict after some multiple-equation commands, mi predict does not allow specification of multiple new variables to store predictions from all equations. For each equation *eqno*, you should use mi predict, equation(*eqno*) to obtain predictions from equation *eqno*.

To obtain estimation-sample predictions, the if e(sample) restriction is usually used with predict and predictnl. This restriction is not allowed with mi predict and mi predictnl. mi estimate does not set an estimation sample. There is no single estimation sample within the MI framework; there are M of them, and they may vary across imputed datasets. To obtain estimation-sample predictions with mi predict and mi predictnl, you must first store the estimation sample in a variable and then specify this variable in the esample() option. For example, you may use mi estimate's esample(*newvar*) option to store the estimation sample in *newvar*. To use mi estimate, esample(), you must be in flong or flongsep style; use [MI] **mi convert** to convert to one of these styles.

mi predict and mi predictnl store MI predictions and statistics associated with them in the original data ($m = 0$). If your data are flong or flongsep, you may additionally store the corresponding completed-data estimates in the imputed data ($m > 0$) by specifying the storecompleted option. This option only affects results for which completed-data counterparts are available, such as predictions, standard errors, and confidence intervals. It has no effect on statistics specific to MI, such as relative variance increases and fractions of missing information.

When you restrict predictions to a subsample, mi predict and mi predictnl verify that the prediction samples are the same across imputed datasets. If varying prediction samples are detected, the commands terminate with an error. If such a situation occurs, you may consider modifying your restriction to define a sample common to all imputations. If there are a few imputations violating the consistency of the prediction sample, you may obtain MI predictions over a selected set of imputations using, for example, the imputations() option.

Example 1: Obtain MI linear predictions and other statistics

Recall the analysis of house resale prices from *Example 2: Completed-data linear regression analysis* in [MI] **mi estimate**:

```
. use https://www.stata-press.com/data/r19/mhouses1993s30
(Albuquerque home prices Feb15-Apr30, 1993)
. mi estimate, saving(miest): regress price tax sqft age nfeatures ne custom
> corner
Multiple-imputation estimates                   Imputations       =         30
Linear regression                               Number of obs     =        117
                                                Average RVI       =     0.0648
                                                Largest FMI       =     0.2533
                                                Complete DF       =        109
DF adjustment:   Small sample                   DF:     min       =      69.12
                                                        avg       =      94.02
                                                        max       =     105.51
Model F test:       Equal FMI                   F(   7,  106.5)   =      67.18
Within VCE type:          OLS                   Prob > F          =     0.0000
```

price	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
tax	.6768015	.1241568	5.45	0.000	.4301777	.9234253
sqft	.2118129	.069177	3.06	0.003	.0745091	.3491168
age	.2471445	1.653669	0.15	0.882	-3.051732	3.546021
nfeatures	9.288033	13.30469	0.70	0.487	-17.12017	35.69623
ne	2.518996	36.99365	0.07	0.946	-70.90416	75.94215
custom	134.2193	43.29755	3.10	0.002	48.35674	220.0818
corner	-68.58686	39.9488	-1.72	0.089	-147.7934	10.61972
_cons	123.9118	71.05816	1.74	0.085	-17.19932	265.0229

We saved complete-data estimation results to `miest.ster` using `mi estimate`'s `saving()` option.

We store MI linear predictions in variable `xb_mi`:

```
. mi predict xb_mi using miest
(option xb assumed; linear prediction)
. mi xeq 0: summarize price xb_mi
m=0 data:
-> summarize price xb_mi
```

Variable	Obs	Mean	Std. dev.	Min	Max
price	117	1062.735	380.437	540	2150
xb_mi	117	1062.735	344.2862	523.0295	2042.396

MI predictions are stored in the original data ($m = 0$). The predictions of `price` seem reasonable.

We compute standard errors of MI linear predictions by using the `stdp` option:

```
. mi predict stdp_mi using miest, stdp
```

To obtain other statistics, such as confidence intervals and Wald test statistics, we can use `mi predictnl`. For example, we compute linear predictions, 95% confidence intervals, and fractions of missing information of the linear predictions as follows:

```
. mi predictnl xb1_mi = predict(xb) using miest, ci(cil_mi ciu_mi) fmi(fmi)
```

Unlike confidence intervals produced by `predictnl`, confidence intervals from `mi predictnl` are based on observation-specific degrees of freedom. Recall from [MI] **mi estimate** that the degrees of freedom used for MI inference is inversely related to relative variance increases due to missing data, which are

parameter-specific. The prediction for each observation is viewed as a separate parameter, so it has its own degrees of freedom. If desired, you may obtain observation-specific MI degrees of freedom by specifying the `df()` option with `mi predictnl`.

Example 2: Obtain MI linear predictions for the estimation sample

To obtain MI linear predictions for the estimation sample, we must first store the estimation sample in a variable. To store the estimation sample with `mi estimate`, the `mi` data must be flong or flongsep.

Continuing our house resale prices example, the data are mlong:

```
. use https://www.stata-press.com/data/r19/mhouses1993s30, clear
(Albuquerque home prices Feb15-Apr30, 1993)

. mi query
data mi set mlong, M = 30
last mi update 04feb2025 12:58:57, 11 days ago
```

We switch to the flong style by using the `mi convert` command (see [MI] **mi convert**) and store the estimation sample in variable `touse` by using `mi estimate, esample()`:

```
. mi convert flong

. mi estimate, esample(touse): regress price tax sqft age nfeatures ne custom
> corner
Multiple-imputation estimates                   Imputations       =         30
Linear regression                               Number of obs     =        117
                                                Average RVI       =     0.0648
                                                Largest FMI       =     0.2533
                                                Complete DF       =        109
DF adjustment:   Small sample                   DF:     min       =      69.12
                                                        avg       =      94.02
                                                        max       =     105.51
Model F test:       Equal FMI                   F(   7,  106.5)   =      67.18
Within VCE type:          OLS                   Prob > F          =     0.0000
```

price	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
tax	.6768015	.1241568	5.45	0.000	.4301777	.9234253
sqft	.2118129	.069177	3.06	0.003	.0745091	.3491168
age	.2471445	1.653669	0.15	0.882	-3.051732	3.546021
nfeatures	9.288033	13.30469	0.70	0.487	-17.12017	35.69623
ne	2.518996	36.99365	0.07	0.946	-70.90416	75.94215
custom	134.2193	43.29755	3.10	0.002	48.35674	220.0818
corner	-68.58686	39.9488	-1.72	0.089	-147.7934	10.61972
_cons	123.9118	71.05816	1.74	0.085	-17.19932	265.0229

Because we use the same regression model, we do not need to resave estimation results and we can use the previously saved `miest.ster` from *Example 1: Obtain MI linear predictions and other statistics* with `mi predict`.

To restrict the linear prediction to the estimation sample identified by the `touse` variable, we use `esample(touse)` with `mi predict`:

```
. mi predict xb_mi using miest, esample(touse)
(option xb assumed; linear prediction)
. mi xeq 0: summarize xb_mi
m=0 data:
-> summarize xb_mi
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       xb_mi |        117    1062.735    344.2862   523.0295   2042.396
```

The estimation sample includes all observations, so we obtain the same predictions as we did in example 1.

We could simply use an `if` restriction instead of the `esample()` option to obtain the same results:

```
. mi predict xb_mi if touse using miest
```

But if you use the `esample()` option, `mi predict` and `mi predictnl` perform additional checks to verify that the supplied variable is a proper estimation-sample variable.

By default, the MI linear prediction is only stored in the original data ($m = 0$) and the imputed data contain missing values in the corresponding variable. In the flong and flongsep styles, we can also store completed-data predictions in the imputed data ($m > 0$) by specifying the `storecompleted` option:

```
. mi predict xb_mi_all using miest, esample(touse) storecompleted
(option xb assumed; linear prediction)
. mi xeq 0 1 2: summarize xb_mi_all
m=0 data:
-> summarize xb_mi_all
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
   xb_mi_all |        117    1062.735    344.2862   523.0295   2042.396
m=1 data:
-> summarize xb_mi_all
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
   xb_mi_all |        117    1062.735    346.1095   529.5227   2042.942
m=2 data:
-> summarize xb_mi_all
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
   xb_mi_all |        117    1062.735    344.8446   515.5598   2040.374
```

Variable `xb_mi_all` contains MI linear predictions in $m = 0$; completed-data linear predictions from imputation 1 in $m = 1$; completed-data linear predictions from imputation 2 in $m = 2$; and so on.

Example 3: Obtain MI estimates of probabilities

Recall the analysis of heart attacks from *Example 1: Completed-data logistic analysis* in [MI] **mi estimate**:

```
. use https://www.stata-press.com/data/r19/mheart1s20, clear
(Fictional heart attack data; BMI missing)
. mi estimate, saving(miest, replace): logit attack smokes age bmi hsgrad female
Multiple-imputation estimates                   Imputations       =         20
Logistic regression                             Number of obs     =        154
                                                Average RVI       =     0.0312
                                                Largest FMI       =     0.1355
DF adjustment:   Large sample                   DF:     min       =   1,060.38
                                                        avg       = 223,362.56
                                                        max       = 493,335.88
Model F test:       Equal FMI                   F(   5,71379.3)   =       3.59
Within VCE type:          OIM                   Prob > F          =     0.0030
------------------------------------------------------------------------------
      attack | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      smokes |   1.198595   .3578195     3.35   0.001     .4972789    1.899911
         age |   .0360159   .0154399     2.33   0.020     .0057541    .0662776
         bmi |   .1039416   .0476136     2.18   0.029      .010514    .1973692
      hsgrad |   .1578992   .4049257     0.39   0.697    -.6357464    .9515449
      female |  -.1067433   .4164735    -0.26   0.798    -.9230191    .7095326
       _cons |  -5.478143   1.685075    -3.25   0.001    -8.782394   -2.173892
------------------------------------------------------------------------------
```

We could have used a different estimation file to store the completed-data estimation results from `logit`. Instead, we replaced the existing estimation file `miest.ster` with new results by specifying `saving()`'s `replace` option.

Following the discussion in *Introduction*, we first obtain MI estimates of the probabilities of a positive outcome by using the transformation-based approach. We obtain MI estimates of linear predictions and apply the inverse-logit transformation to obtain the probabilities:

```
. mi predict xb_mi using miest
(option xb assumed; linear prediction)
. quietly mi xeq: generate phat = invlogit(xb_mi)
```

Unlike `predict` after `logit`, `mi predict` after `mi estimate: logit` defaults to the linear prediction and not to the probability of a positive outcome. `mi predict` always assumes the linear prediction.

Alternatively, we can apply Rubin's combination rules directly to probabilities. Unlike `predict`, `mi predict` does not allow the `pr` option. You can obtain only linear predictions or standard errors using `mi predict`. We can use the `predict()` function of `mi predictnl` to obtain MI estimates of the probabilities by directly pooling completed-data probabilities:

```
. mi predictnl phat_mi = predict(pr) using miest
. mi xeq 0: summarize phat phat_mi
m=0 data:
-> summarize phat phat_mi
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
        phat |        154    .4478198    .1820425   .1410432   .8923041
     phat_mi |        154    .4480519    .1812098    .141361   .8912111
```

Although the first approach is preferable, we can see that we obtain similar estimates of the probabilities of a positive outcome with both approaches.

Example 4: Obtain other MI predictions

Consider the cancer data from *Example 3: Completed-data survival analysis* in [MI] **mi estimate**:

```
. use https://www.stata-press.com/data/r19/mdrugtrs25, clear
(Patient survival in drug trial)
. mi stset studytime, failure(died)
Survival-time data settings
         Failure event: died!=0 & died<.
Observed time interval: (0, studytime]
     Exit on or before: failure
------------------------------------------------------------------------------
         48  total observations
          0  exclusions
------------------------------------------------------------------------------
         48  observations remaining, representing
         31  failures in single-record/single-failure data
        744  total analysis time at risk and under observation
                                                At risk from t =         0
                                     Earliest observed entry t =         0
                                          Last observed exit t =        39
```

In this example, we fit a parametric Weibull regression to the survival data and as before replace the estimation results in `miest.ster` with new ones from `mi estimate: streg`:

```
. mi estimate, saving(miest, replace): streg drug age, dist(weibull)
Multiple-imputation estimates                   Imputations       =         25
Weibull PH regression                           Number of obs     =         48
                                                Average RVI       =     0.0927
                                                Largest FMI       =     0.1847
DF adjustment:   Large sample                   DF:     min       =     721.15
                                                        avg       =   6,014.48
                                                        max       =  11,383.09
Model F test:       Equal FMI                   F(   2, 2910.0)   =      14.94
Within VCE type:          OIM                   Prob > F          =     0.0000
------------------------------------------------------------------------------
          _t | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
        drug |  -2.093333   .4091925    -5.12   0.000    -2.895422   -1.291243
         age |    .126931   .0403526     3.15   0.002     .0477084    .2061536
       _cons |  -11.14588   2.584909    -4.31   0.000    -16.22013   -6.071634
-------------+----------------------------------------------------------------
      /ln_p |   .5524239   .1434973     3.85   0.000     .2711445    .8337033
-------------+----------------------------------------------------------------
           p |   1.737459   .2493207                      1.311465    2.301827
         1/p |    .575553   .0825903                      .4344374    .7625063
------------------------------------------------------------------------------
```

Suppose that we want to estimate median survival time. After `streg`, median survival time can be obtained by using `predict, median time`. `mi predict` does not support these options, but we can use the `predict(median time)` function with `mi predictnl` to obtain MI estimates of the median survival time.

To improve normality, we perform pooling in a log scale and then exponentiate results back to the original scale:

```
. mi predictnl p50_lntime_mi = ln(predict(median time)) using miest
. quietly mi xeq: generate p50_time_mi = exp(p50_lntime_mi)
```

Above, we demonstrated the use of expressions with the `predict()` function by computing median log-survival time by using `ln(predict(median time))`. Alternatively, we can compute median log-survival time directly with `predict(median lntime)`:

```
. mi predictnl p50_lntime1_mi = predict(median lntime) using miest
. quietly mi xeq: generate p50_time1_mi = exp(p50_lntime1_mi)
```

We verify that we obtain identical results:

```
. mi xeq 0: summarize p50_time_mi p50_time1_mi
m=0 data:
-> summarize p50_time_mi p50_time1_mi
```

Variable	Obs	Mean	Std. dev.	Min	Max
p50_time_mi	48	21.74607	14.60662	3.707896	53.10997
p50_time1_mi	48	21.74607	14.60662	3.707896	53.10997

Example 5: Obtain MI predictions after multiple-equation commands

For illustrative purposes, let's use `mlogit` instead of `logit` to analyze the heart-attack data from *Example 3: Obtain MI estimates of probabilities*:

```
. use https://www.stata-press.com/data/r19/mheart1s20, clear
(Fictional heart attack data; BMI missing)
. mi estimate, saving(miest, replace): mlogit attack smokes age bmi hsgrad female
Multiple-imputation estimates                   Imputations       =         20
Multinomial logistic regression                 Number of obs     =        154
                                                Average RVI       =     0.0312
                                                Largest FMI       =     0.1355
DF adjustment:   Large sample                   DF:     min       =   1,060.38
                                                        avg       = 223,362.56
                                                        max       = 493,335.88
Model F test:       Equal FMI                   F(   5,71379.3)   =       3.59
Within VCE type:          OIM                   Prob > F          =     0.0030
```

attack	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
0	(base outcome)					
1						
smokes	1.198595	.3578195	3.35	0.001	.4972789	1.899911
age	.0360159	.0154399	2.33	0.020	.0057541	.0662776
bmi	.1039416	.0476136	2.18	0.029	.010514	.1973692
hsgrad	.1578992	.4049257	0.39	0.697	-.6357464	.9515449
female	-.1067433	.4164735	-0.26	0.798	-.9230191	.7095326
_cons	-5.478143	1.685075	-3.25	0.001	-8.782394	-2.173892

We obtain the same results as with `mi estimate: logit`.

To obtain predictions after multiple-equation commands such as `mlogit`, we need to use the `equation()` option of `mi predict` or `mi predictnl` to obtain a prediction from a specific equation. By default, the first equation is assumed:

```
. mi predict xb_0_mi using miest
(option xb assumed; linear prediction)
. mi xeq 0: summarize xb_0_mi
m=0 data:
-> summarize xb_0_mi
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     xb_0_mi |        154           0           0           0          0
```

In our example, the first equation corresponds to the base category, so the linear prediction is zero for this equation.

To obtain the linear prediction from the second equation, we specify the `equation(`*eqno*`)` option. *eqno* can refer to the equation number, `#2`, or to the equation name, `1`. For example,

```
. mi predict xb_1_mi using miest, equation(#2)
(option xb assumed; linear prediction)
```

Suppose we want to compute observation-specific odds of a heart attack. Knowing that the odds of a disease is the exponentiated linear predictor, we can compute the odds simply as

```
. quietly mi xeq: generate odds_mi = exp(xb_1_mi)
```

Instead, to illustrate a more advanced syntax of `mi predictnl`, we compute the odds using their definition as the ratio of a probability of a heart attack (`attack==1`) to the probability of no heart attack (`attack==0`). Log odds are asymptotically normally distributed, so we apply combination rules to log odds and then exponentiate the result to obtain odds:

```
. mi predictnl lnodds_mi = ln(predict(pr equation(1))/predict(pr equation(0)))
> using miest
. quietly mi xeq: generate odds_mi = exp(lnodds_mi)
```

In the above, we used the names of the equations, `0` and `1`, within `equation()` to obtain probabilities of no heart attack and a heart attack, respectively.

We can see, for example, that for older subjects or subjects who smoke, the odds of having a heart attack are noticeably higher:

```
. quietly mi xeq: generate byte atrisk = smokes==1 | age>50
. mi xeq 0: by atrisk, sort: summ odds_mi
m=0 data:
-> by atrisk, sort: summ odds_mi
-------------------------------------------------------------------------------
-> atrisk = 0
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     odds_mi |         30    .3472545    .1451144   .1642029    .818259
-------------------------------------------------------------------------------
-> atrisk = 1
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     odds_mi |        124    1.327598    1.228176   .2198672   8.285403
```

Methods and formulas

Multiple-imputation predictions are obtained by considering an observation-specific prediction as an estimand and by applying Rubin's combination rules to it (White, Royston, and Wood 2011).

Let $\eta_i(\cdot)$ be a prediction of interest for subject i and $\hat{\eta}_{i,m}(\cdot)$ be a completed-data estimate of the prediction for subject i, $i = 1, \ldots, N$, from imputation m, $m = 1, \ldots, M$. In what follows, we omit the functional argument of $\eta_i(\cdot)$ for brevity.

The MI estimate of prediction η_i is

$$\overline{\eta}_{i,M} = \frac{1}{M}\sum_{m=1}^{M} \hat{\eta}_{i,m},\ i = 1, \ldots, N$$

Let $\widehat{\text{Var}}(\hat{\eta}_{i,m})$ be the completed-data variance of the completed-data prediction $\hat{\eta}_{i,m}$ for subject i from imputation m. The standard error of the MI prediction $\overline{\eta}_{i,M}$ is the square root of the total MI variance $T_{\overline{\eta}_{i,M}}$,

$$T_{\overline{\eta}_{i,M}} = \overline{U}_i + \left(1 + \frac{1}{M}\right) B_i,\ i = 1, \ldots, N$$

where $\overline{U}_i = \sum_{m=1}^{M} \widehat{\text{Var}}(\hat{\eta}_{i,m})/M$ is the within-imputation variance and

$B_i = \sum_{m=1}^{M} (\hat{\eta}_{i,m} - \overline{\eta}_{i,M})^2/(M-1)$ is the between-imputation variance.

Other statistics such as test statistics, confidence intervals, and relative variance increases are obtained by applying to η_i the same formulas as described in *Univariate case* under *Methods and formulas* of [MI] **mi estimate** for parameter Q. Also see Rubin (1987, 76–77).

As for any other parameter, the validity of applying Rubin's combination rules to η_i is subject to η_i satisfying a set of conditions as described, for example, in *Theory underlying multiple imputation* under *Remarks and examples* of [MI] **Intro substantive**. In particular, the combination rules should be applied to η_i in the metric for which the sampling distribution is closer to the normal distribution.

References

Barnard, J., and D. B. Rubin. 1999. Small-sample degrees of freedom with multiple imputation. *Biometrika* 86: 948–955. https://doi.org/10.1093/biomet/86.4.948.

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

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 postestimation** — Postestimation tools for mi estimate

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

[MI] **Intro** — Introduction to mi

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

[MI] **Glossary**