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.

<table>
<thead>
<tr>
<th>predict_options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb</td>
<td>calculate linear prediction; the default</td>
</tr>
<tr>
<td>stdp</td>
<td>calculate standard error of the prediction</td>
</tr>
<tr>
<td>nooffset</td>
<td>ignore any offset() or exposure() variable</td>
</tr>
<tr>
<td>equation(eqno)</td>
<td>specify equations after multiple-equation commands</td>
</tr>
</tbody>
</table>
### mi predict — Obtain multiple-imputation predictions

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

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

*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] predict.

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 ≤ # ≤ 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.

eexample(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 store-completed 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
Using mi predict and mi predictnl
Example 1: Obtain MI linear predictions and other statistics
Example 2: Obtain MI linear predictions for the estimation sample
Example 3: Obtain MI estimates of probabilities
Example 4: Obtain other MI predictions
Example 5: Obtain MI predictions after multiple-equation commands

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 \textit{eqno}, you should use \texttt{mi predict, equation(eqno)} to obtain predictions from equation \textit{eqno}.

To obtain estimation-sample predictions, the \texttt{if e(sample)} restriction is usually used with \texttt{predict} and \texttt{predictnl}. This restriction is not allowed with \texttt{mi predict} and \texttt{mi predictnl}. \texttt{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 \texttt{mi predict} and \texttt{mi predictnl}, you must first store the estimation sample in a variable and then specify this variable in the \texttt{esample()} option. For example, you may use \texttt{mi estimate’s esample(newvar)} option to store the estimation sample in \texttt{newvar}. To use \texttt{mi estimate, esample()}, you must be in flong or flongsep style; use \texttt{[MI] mi convert} to convert to one of these styles.

\texttt{mi predict} and \texttt{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 \texttt{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, \texttt{mi predict} and \texttt{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 \texttt{imputations()} option.

\textbf{Example 1: Obtain MI linear predictions and other statistics}

Recall the analysis of house resale prices from \textit{Example 2: Completed-data linear regression analysis} in [MI] \texttt{mi estimate}:
We saved complete-data estimation results to `miest.ster` using `mi estimate`'s `saving()` option.

We store MI linear predictions in variable `xb_mi`:

```stata
. mi predict xb_mi using miest
(option xb assumed; linear prediction)
```

```stata
. mi xeq 0: summarize price xb_mi
```

```
Variable | Obs Mean Std. Dev. Min Max
---------|------|----------------|------|-------|
price | 117 1062.735 380.437 540 2150
xb_mi | 117 1062.735 344.286 2 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:

```stata
. 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:

```stata
. 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`. 

We computed the following:

```
   price       Coef.  Std. Err.     t   P>|t|   [95% Conf. Interval]
-------------|--------|-------------|------|--------|-----------------------------|
   tax         .6768015   .1241568  5.45  0.000    .4301777    .9234253
   sqft        .2118129   .0691770  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   39.94880  1.72  0.089   -147.7934   10.61972
   _cons       123.9118   71.05816  1.74  0.085   -17.19932   265.0229
```

We saved multiple-imputation predictions. To use those predictions, we need to estimate the models first:

```stata
. use https://www.stata-press.com/data/r16/mhouses1993s30
(Albuquerque Home Prices Feb15-Apr30, 1993)
. mi estimate, saving(miest): regress price tax sqft age nfeatures ne custom corner
```

We use MI linear regression:

```
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
```
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/r16/mhouses1993s30, clear
(Albuquerque Home Prices Feb15-Apr30, 1993)
. mi query
data mi set mlong, M = 30
last mi update 19apr2019 14:00:11, 8 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
```

```
| Variable | Coef. | Std. Err. | t     | P>|t|   | [95% Conf. Interval] |
|----------|-------|-----------|-------|-------|----------------------|
| price    | .6768015 | .1241568 | 5.45  | 0.000 | .4301777 - .9234253  |
| tax      | .2118129 | .069177  | 3.06  | 0.003 | .0745091 - .3491168 |
| sqft     | .2471445 | 1.653669 | 0.15  | 0.882 | -3.051732 - 3.546021 |
| age      | 9.288033  | 13.30469 | 0.70  | 0.487 | -17.12017 - 35.69623 |
| nfeatures| 2.518996  | 36.99365 | 0.07  | 0.946 | -70.90416 - 75.94215 |
| ne       | 134.2193  | 43.29755 | 3.10  | 0.002 | 48.35674 - 220.0818  |
| custom   | 68.58686  | 39.9488  | -1.72 | 0.089 | -147.7934 - 10.61972 |
| corner   | 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
```

```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>xb_mi</td>
<td>117</td>
<td>1062.735</td>
<td>344.2862</td>
<td>523.0295</td>
<td>2042.396</td>
</tr>
</tbody>
</table>
```

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
```

```
. mi xeq 0 1 2: summarize xb_mi_all
```

```

\(m=0\) data:
```

```
Variable        |   Obs  |      Mean   |        Std. Dev. |        Min   |        Max
xb_mi_all        |     117 | 1062.735   | 344.2862        | 523.0295    | 2042.396
```

```
\(m=1\) data:
```

```
Variable        |   Obs  |      Mean   |        Std. Dev. |        Min   |        Max
xb_mi_all        |     117 | 1062.735   | 346.1095        | 529.5227    | 2042.942
```

```
\(m=2\) data:
```

```
Variable        |   Obs  |      Mean   |        Std. Dev. |        Min   |        Max
xb_mi_all        |     117 | 1062.735   | 344.8446        | 515.5598    | 2040.374
```

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

Following the discussion in \textit{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:

\begin{verbatim}
    . mi predict xb_mi using miest
    (option \texttt{xb} assumed; linear prediction)
    . quietly mi xeq: generate phat = invlogit(xb_mi)
\end{verbatim}

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

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

\begin{verbatim}
    . mi predictnl phat_mi = predict(pr) using miest
    . mi xeq 0: summarize phat phat_mi
\end{verbatim}

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/r16/mdrugtrs25, clear
(Patient Survival in Drug Trial)
. mi stset studytime, failure(died)
   failure event:  died ! = 0 & died < .
obs. 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)
```

Weibull PH regression
Imputations = 25
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

|    | Coef.  | 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  | 3.85 | 0.000 | 1.311465 2.301827  |
| 1/p  | .575553  | .0825903  | 6.70 | 0.000 | .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)
```
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 21.746 14.60662   3.70796 53.10997
p50_time1_mi |       48 21.746 14.60662   3.70796 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/r16/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

          | Coef.  Std. Err.      t    P>|t|   [95% Conf. Interval]
-------------+--------------------------------------------------
     attack   |                                                 
       0       |  (base outcome)                                 
       1       |                                                 
   smokes     |  1.198595   .3578195   3.35    0.001     0.4972789    1.899911
   age        |  .0360159   .0154399   2.33    0.020     0.0057541    .0662776
   bmi        |  .1039416   .0476136   2.18    0.029     0.010514     .1973692
   hsgrad     |  .1578992   .4049257   0.39    0.697    -0.6357464    .9515449
   female     | -0.1067433  .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
     ------- |-------:------:------:------:------:
              |    30   0.3472545  0.1451144  0.1642029  0.818259
     -> atrisk = 1
       Variable |     Obs   1.327598   1.228176 0.2198672  8.285403
     ------- |-------:------:------:------:------:
              |   124
```

**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 $\eta_i$ is

$$\bar{\eta}_{i,M} = \frac{1}{M} \sum_{m=1}^{M} \hat{\eta}_{i,m}, \quad i = 1, \ldots, N$$
Let $\hat{\text{Var}}(\hat{\eta}_{i,m})$ 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 $\bar{\eta}_{i,M}$ is the square root of the total MI variance $T_{\bar{\eta}_{i,M}}$,

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

where $\bar{U}_i = \sum_{m=1}^{M} \hat{\text{Var}}(\hat{\eta}_{i,m})/M$ is the within-imputation variance and $B_i = \sum_{m=1}^{M} (\hat{\eta}_{i,m} - \bar{\eta}_{i,M})^2/(M - 1)$ is the between-imputation variance.

Other statistics such as test statistics, confidence intervals, and relative variance increases are obtained by applying to $\eta_i$ 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 $\eta_i$ is subject to $\eta_i$ 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 $\eta_i$ in the metric for which the sampling distribution is closer to the normal distribution.

References


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