workflow — Suggested workflow

Description Remarks and examples Also see

Description

Provided below are suggested workflows for working with original data and for working with data that already have imputations.

Remarks and examples

Remarks are presented under the following headings:

Suggested workflow for original data

By original data, we mean data with missing values for which you do not already have imputations. Your task is to identify the missing values, impute values for them, and perform estimation.

`mi` does not have a fixed order in which you must perform tasks except that you must `mi set` the data first.

1. `mi set` your data; see [MI] **mi set**.

   Set the data to be wide, mlong, flong, or flongsep. Choose flongsep only if your data are bumping up against the constraints of memory. Choose flong or flongsep if you will need super-varying variables.

   Memory is not usually a problem, and super-varying variables are seldom necessary, so we generally start with the data as wide:

   ```
   . use originaldata
   . mi set wide
   ```

   If you need to use flongsep, you also need to specify a name for the flongsep dataset collection. Choose a name different from the current name of the dataset:

   ```
   . use originaldata
   . mi set flongsep newname
   ```

   If the original dataset is `chd.dta`, you might choose `chdm` for *newname*. *newname* does not include the `.dta` suffix. If you choose `chdm`, the data will then be stored in `chdm.dta`, `_1_chdm.dta`, and so on. It is important that you choose a name different from *originaldata* because you do not want your `mi` data to overwrite the original. Stata users are used to working with a copy of the data in memory, meaning that the changes made to the data are not reflected in the `.dta` dataset until the user saves them. With flongsep data, however, changes are made to the `mi .dta` dataset collection as you work. See *Advice for using flongsep* in [MI] **styles**.

2. Use `mi describe` often; see [MI] **mi describe**.

   `mi describe` will not tell you anything useful yet, but as you set more about the data, `mi describe` will be more informative.

   ```
           . mi describe
   ```

3. Use `mi misstable` to identify missing values; see [MI] **mi misstable**.

   `mi misstable` is the standard `misstable` (see [R] **misstable**) but tailored for `mi` data. Several Stata commands have `mi` variants—become familiar with them. If there is no `mi` variant, then it is generally safe to use the standard command directly, although it may not be appropriate. For instance, typing `misstable` rather than `mi misstable` would produce appropriate results right now, but it would not produce appropriate results later. If `mi` datasets $m = 0$, $m = 1$, ..., $m = M$ exist and you run `misstable`, you might end up running the command on a strange combination of the m's. We recommend the wide style because general Stata commands will do what you expect. The same is true for the flongsep style. It is your responsibility to get this right.

   So what is the difference between `mi misstable` and `misstable`? `mi misstable` amounts to `mi xeq 0: misstable, exok`, which is to say it runs on $m = 0$ and specifies the `exok` option so that extended missing values are treated as hard missings.

   In general, you need to become familiar with all the `mi` commands, use the `mi` variant of regular Stata commands whenever one exists, and think twice before using a command without an `mi` prefix. Doing the right thing will become automatic once you gain familiarity with the styles; see [MI] **styles**.

   To learn about the missing values in your data, type

   ```
           . mi misstable summarize
   ```

4. Use `mi register imputed` to register the variables you wish to impute; see [MI] **mi set**.

   The only variables that `mi` will impute are those registered as `imputed`. You can register variables one at a time or all at once. If you register a variable mistakenly, use `mi unregister` to unregister it.

   ```
           . mi register imputed varname [varname ...]
   ```

5. Use `mi impute` to impute (fill in) the missing values; see [MI] **mi impute**.

   There is a lot to be said here. For instance, in a dataset where variables `age` and `bmi` contain missing, you might type

   ```
           . mi register imputed age bmi
           . mi impute mvn age bmi = attack smokes hsgrad, add(10)
   ```

   `mi impute`'s `add(#)` option specifies the number of imputations to be added. We currently have 0 imputations, so after imputation, we will have 10. We usually start with a small number of imputations and add more later.

6. Use `mi describe` to verify that all missing values are filled in; see [MI] **mi describe**.

   ```
           . mi describe
   ```

   You might also want to use `mi xeq` (see [MI] **mi xeq**) to look at summary statistics in each of the imputation datasets:

   ```
           . mi xeq: summarize
   ```

7. Generate passive variables; see [MI] **mi passive**.

   Passive variables are variables that are functions of imputed variables, such as `lnage` when some values of `age` are imputed. The values of passive variables differ across m just as the values of imputed variables do. The official way to generate imputed values is by using `mi passive`:

   ```
   . mi passive: generate lnage = ln(age)
   ```

   Rather than use the official way, however, we often switch our data to mlong and just generate the passive variables directly:

   ```
   . mi convert mlong
   . generate lnage = ln(age)
   . mi register passive lnage
   ```

   If you work as we do, remember to register any passive variables you create. When you are done, you may `mi convert` your data back to wide, but there is no reason to do that.

8. Use `mi estimate` (see [MI] **mi estimate**) to fit models:

   ```
   . mi estimate: logistic attack smokes age bmi hsgrad
   ```

   You fit your model just as you would ordinarily except that you add `mi estimate:` in front of the command.

To see an example of the advice applied to a simple dataset, see *Example* below.

In theory, you should get your data cleaning and data management out of the way before `mi setting` your data. In practice that will not happen, so you will want to become familiar with the other `mi` commands. Among the data management commands available are `mi append` (see [MI] **mi append**), `mi merge` (see [MI] **mi merge**), `mi expand` (see [MI] **mi expand**), and `mi reshape` (see [MI] **mi reshape**). If you are working with survival-time data, also see [MI] **mi stsplit**. To `stset` your data, or `svyset`, or `xtset`, see [MI] **mi set** and [MI] **mi XXXset**.

Suggested workflow for data that already have imputations

Data sometimes come with imputations included. The data might be made by another researcher for you or the data might come from an official source. Either way, we will assume that the data are not in Stata format, because if they were, you would just use the data and would type `mi describe`.

`mi` can import officially produced datasets created by the National Health and Nutrition Examination Survey (NHANES) with the `mi import nhanes1` command, and `mi` can import more informally created datasets that are wide-, flong-, or flongsep-like with `mi import wide`, `mi import flong`, or `mi import flongsep`; see [MI] **mi import**.

The required workflow is hardly different from *Suggested workflow for original data*, presented above. The differences are that you will use `mi import` rather than `mi set` and you will skip using `mi impute` to generate the imputations. In this sense, your job is easier.

On the other hand, you need to verify that you have imported your data correctly, and we have a lot to say about that. Basically, after importing, you need to be careful about which `mi` commands you use until you have verified that you have the variables registered correctly. That is discussed in [MI] **mi import**.

Example

We are going to repeat *A simple example* from [MI] **intro**, but this time we are going to follow the advice given above in *Suggested workflow for original data*.

We have fictional data on 154 patients and want to examine the relationship between binary outcome `attack`, recording heart attacks, and variables `smokes`, `age`, `bmi`, `hsgrad`, and `female`. We will use logistic regression. Below we load our original data and show you a little about it using the standard commands `describe` and `summarize`. We emphasize that `mheart5.dta` is just a standard Stata dataset; it has not been `mi set`.

```
. use http://www.stata-press.com/data/r13/mheart5
(Fictional heart attack data)

. describe

Contains data from http://www.stata-press.com/data/r13/mheart5.dta
  obs:           154                          Fictional heart attack data
 vars:             6                          19 Jun 2012 10:50
 size:         1,848
───────────────────────────────────────────────────────────────────────────────
              storage   display    value
variable name   type    format     label      variable label
───────────────────────────────────────────────────────────────────────────────
attack          byte    %9.0g                 Outcome (heart attack)
smokes          byte    %9.0g                 Current smoker
age             float   %9.0g                 Age, in years
bmi             float   %9.0g                 Body Mass Index, kg/m^2
female          byte    %9.0g                 Gender
hsgrad          byte    %9.0g                 High school graduate
───────────────────────────────────────────────────────────────────────────────
Sorted by:

. summarize

    Variable │       Obs        Mean    Std. Dev.       Min        Max
─────────────┼────────────────────────────────────────────────────────
      attack │       154    .4480519    .4989166          0          1
      smokes │       154    .4155844    .4944304          0          1
         age │       142    56.43324    11.59131   20.73613   83.78423
         bmi │       126    25.23523    4.029325   17.22643   38.24214
      female │       154    .2467532    .4325285          0          1
─────────────┼────────────────────────────────────────────────────────
      hsgrad │       154    .7532468    .4325285          0          1
```

The first guideline is

1. `mi set` your data; see [MI] **mi set**.

We will set the data to be flong even though in *A simple example* we set the data to be mlong. `mi` provides four styles—flong, mlong, wide, and flongsep—and at this point it does not matter which we choose. `mi` commands work the same way regardless of style. Four styles are provided because, should we decide to step outside of `mi` and attack the data with standard Stata commands, we will find different styles more convenient depending on what we want to do. It is easy to switch styles.

Below we type `mi set flong` and then, to show you what that command did to the data, we show you the output from a standard `describe`:

```
. mi set flong
. describe
Contains data from http://www.stata-press.com/data/r13/mheart5.dta
  obs:           154                          Fictional heart attack data
 vars:             9                          19 Jun 2012 10:50
 size:         2,618
--------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
attack          byte    %9.0g                 Outcome (heart attack)
smokes          byte    %9.0g                 Current smoker
age             float   %9.0g                 Age, in years
bmi             float   %9.0g                 Body Mass Index, kg/m^2
female          byte    %9.0g                 Gender
hsgrad          byte    %9.0g                 High school graduate
_mi_miss        byte    %8.0g
_mi_m           int     %8.0g
_mi_id          int     %12.0g
--------------------------------------------------------------------------------
Sorted by:
```

Typing `mi set flong` added three variables to our data: `_mi_miss`, `_mi_m`, and `_mi_id`. Those variables belong to `mi`. If you are curious about them, see [MI] **styles**. Advanced users can even use them. No matter how advanced you are, however, you must never change their contents.

Except for the three added variables, the data are unchanged, and we would see that if we typed `summarize`. The three added variables are due to the style we chose. When you `mi set` your data, different styles will change the data differently, but the changes will be just around the edges.

The second guideline is

2. Use `mi describe` often; see [MI] **mi describe**.

The guideline is to use `mi describe`, not `describe` as we just did. Here is the result:

```
. mi describe
  Style:  flong
          last mi update 07feb2013 13:13:56, 0 seconds ago
  Obs.:   complete          154
          incomplete          0  (M = 0 imputations)
          ---------------------
          total             154
  Vars.:  imputed:  0
          passive:  0
          regular:  0
          system:   3; _mi_m _mi_id _mi_miss
         (there are 6 unregistered variables)
```

As the guideline warned us, "`mi describe` will not tell you anything useful yet."

The third guideline is

3. Use mi misstable to identify missing values; see [MI] **mi misstable**.

Below we type mi misstable summarize and mi misstable nested:

```
. mi misstable summarize
                                                               Obs<.
                                                 +------------------------------
                |                                | Unique
       Variable |     Obs=.     Obs>.     Obs<.  | values          Min         Max
  --------------+--------------------------------+------------------------------
            age |        12                 142  |    142     20.73613    83.78423
            bmi |        28                 126  |    126     17.22643    38.24214
  ------------------------------------------------------------------------------

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

mi misstable summarize reports the variables containing missing values. Those variables in our data are age and bmi. Notice that mi misstable summarize draws a distinction between, as it puts it, "Obs=." and "Obs>.", which is to say between standard missing (.) and extended missing (.a, .b, ..., .z). That is because mi has a concept of soft and hard missing, and it associates soft missing with system missing and hard missing with extended missing. Hard missing values—extended missings—are taken to mean missing values that are not to be imputed. Our data have no missing values like that.

After typing mi misstable summarize, we typed mi misstable nested because we were curious whether the missing values were nested or, to use the jargon, monotone. We discovered that they were. That is, age has 12 missing values in the data, and in every observation in which age is missing, so is bmi, although bmi has another 16 missing values scattered around the data. That means we can use a monotone imputation method, and that is good news because monotone methods are more flexible and faster. We will discuss the implications of that shortly. There is a mechanical detail we must handle first.

The fourth guideline is

4. Use mi register imputed to register the variables you wish to impute; see [MI] **mi set**.

We know that age and bmi have missing values, and before we can impute replacements for those missing values, we must register the variables as to-be-imputed, which we do by typing

```
. mi register imputed age bmi
(28 m=0 obs. now marked as incomplete)
```

Guideline 2 suggested that we type mi describe often. Perhaps now would be a good time:

```
. mi describe
  Style:  flong
          last mi update 07feb2013 13:13:56, 0 seconds ago
  Obs.:   complete          126
          incomplete         28  (M = 0 imputations)
          ---------------------
          total             154
  Vars.:  imputed:  2; age(12) bmi(28)
          passive:  0
          regular:  0
          system:   3; _mi_m _mi_id _mi_miss
         (there are 4 unregistered variables; attack smokes female hsgrad)
```

The output has indeed changed. mi knows just as it did before that we have 154 observations, and it now knows that 126 of them are complete and 28 of them are incomplete. It also knows that age and bmi are to be imputed. The numbers in parentheses are the number of missing values.

The fifth guideline is

5. Use mi impute to impute (fill in) the missing values; see [MI] **mi impute**.

In *A simple example* from [MI] **intro**, we imputed values for age and bmi by typing

```
. mi impute mvn age bmi = attack smokes hsgrad female, add(10)
```

This time, we will impute values by typing

```
. mi impute monotone (regress) age bmi = attack smokes hsgrad female, add(20)
```

We changed add(10) to add(20) for no other reason than to show that we could, although we admit to a preference for more imputations whenever possible. add() specifies the number of imputations to be added to the data. For every missing value, we will impute 20 nonmissing replacements.

We switched from mi impute mvn to mi impute monotone because our data are monotone. Here mi impute monotone will be faster than mi impute mvn but will offer no statistical advantage. In other cases, there might be statistical advantages. All of which is to say that when you get to the imputation step, you have important decisions to make and you need to become knowledgeable about the subject. You can start by reading [MI] **mi impute**.

```
. set seed 20039

. mi impute monotone (regress) age bmi = attack smokes hsgrad female,
> add(20)

Conditional models:
               age: regress age attack smokes hsgrad female
               bmi: regress bmi age attack smokes hsgrad female

Multivariate imputation                     Imputations =        20
Monotone method                                   added =        20
Imputed: m=1 through m=20                       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.)
```

Note that we typed set seed 20039 before issuing the mi impute command. Doing that made our results reproducible. We could have specified mi impute's rseed(20039) option instead. Or we could have skipped setting the random-number seed altogether, and then we would not be able to reproduce our results.

The sixth guideline is

6. Use mi describe to verify that all missing values are filled in; see [MI] **mi describe**.

```
. mi describe, detail
  Style:  flong
          last mi update 07feb2013 13:13:57, 0 seconds ago
  Obs.:   complete           126
          incomplete          28  (M = 20 imputations)
          ---------------------
          total              154
  Vars.:  imputed:  2; age(12; 20*0) bmi(28; 20*0)
          passive:  0
          regular:  0
          system:   3; _mi_m _mi_id _mi_miss
         (there are 4 unregistered variables; attack smokes female hsgrad)
```

This time, we specified `mi describe`'s `detail` option, although you have to look closely at the output to see the effect. When you do not specify `detail`, `mi describe` reports results for the original, unimputed data only, what we call $m = 0$ throughout this documentation. When you specify `detail`, `mi describe` also includes information about the imputation data, what we call $m > 0$ and is $m = 1$, $m = 2$, …, $m = 20$ here. Previously, `mi describe` reported "age(12)", meaning that `age` in $m = 0$ has 12 missing values. This time, it reports "age(12; 20*0)", meaning that `age` still has 12 missing values in $m = 0$, and it has 0 missing values in the 20 imputations. `bmi` also has 0 missing values in the imputations. Success!

Let's take a detour to see how our data really look. Let's type Stata's standard `describe` command. The last time we looked, our data had three extra variables.

```
. describe
Contains data from http://www.stata-press.com/data/r13/mheart5.dta
  obs:         3,234                          Fictional heart attack data
 vars:             9                          7 Feb 2013 13:13
 size:        54,978
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
attack          byte    %9.0g                 Outcome (heart attack)
smokes          byte    %9.0g                 Current smoker
age             float   %9.0g                 Age, in years
bmi             float   %9.0g                 Body Mass Index, kg/m^2
female          byte    %9.0g                 Gender
hsgrad          byte    %9.0g                 High school graduate
_mi_id          int     %12.0g
_mi_miss        byte    %8.0g
_mi_m           int     %8.0g
-------------------------------------------------------------------------------
Sorted by:  _mi_m  _mi_id
```

Nothing has changed as far as variables are concerned, but notice the number of observations. Previously, we had 154 observations. Now we have 3,234! That works out to 21*154. Stored is our original data plus 20 imputations. The flong style makes extra copies of the data.

We chose style flong only because it is so easy to explain. In *A simple example* from [MI] **intro** using this same data, we choose style mlong. It is not too late:

```
. mi convert mlong
```

All that is required to change styles is typing `mi convert`. The style of the data changes, but not the contents. Let's see what `describe` has to report:

```
. describe

Contains data from http://www.stata-press.com/data/r13/mheart5.dta
  obs:           714                          Fictional heart attack data
 vars:             9                          7 Feb 2013 13:13
 size:        12,138
──────────────────────────────────────────────────────────────────────────────
              storage   display    value
variable name   type    format     label      variable label
──────────────────────────────────────────────────────────────────────────────
attack          byte    %9.0g                 Outcome (heart attack)
smokes          byte    %9.0g                 Current smoker
age             float   %9.0g                 Age, in years
bmi             float   %9.0g                 Body Mass Index, kg/m^2
female          byte    %9.0g                 Gender
hsgrad          byte    %9.0g                 High school graduate
_mi_id          int     %12.0g
_mi_miss        byte    %8.0g
_mi_m           int     %8.0g
──────────────────────────────────────────────────────────────────────────────
Sorted by:  _mi_m  _mi_id
```

The data look much like they did when they were flong, except that the number of observations has fallen from 3,234 to 714! Style mlong is an efficient style in that rather than storing the full data for every imputation, it stores only the changes. Back when the data were flong, `mi describe` reported that we had 28 incomplete observations. We get 714 from the 154 original observations plus 20×28 replacement observations for the incomplete observations.

We recommend style mlong. Style wide is also recommended. Below we type `mi convert` to convert our mlong data to wide, and then we run the standard `describe` command:

```
. mi convert wide, clear

. describe

Contains data from http://www.stata-press.com/data/r13/mheart5.dta
  obs:           154                          Fictional heart attack data
 vars:            47                          7 Feb 2013 13:13
 size:        26,642
──────────────────────────────────────────────────────────────────────────────
              storage   display    value
variable name   type    format     label      variable label
──────────────────────────────────────────────────────────────────────────────
attack          byte    %9.0g                 Outcome (heart attack)
smokes          byte    %9.0g                 Current smoker
age             float   %9.0g                 Age, in years
bmi             float   %9.0g                 Body Mass Index, kg/m^2
female          byte    %9.0g                 Gender
hsgrad          byte    %9.0g                 High school graduate
_mi_miss        byte    %8.0g
_1_age          float   %9.0g                 Age, in years
_1_bmi          float   %9.0g                 Body Mass Index, kg/m^2
_2_age          float   %9.0g                 Age, in years
_2_bmi          float   %9.0g                 Body Mass Index, kg/m^2
  (output omitted)
_20_age         float   %9.0g                 Age, in years
_20_bmi         float   %9.0g                 Body Mass Index, kg/m^2
──────────────────────────────────────────────────────────────────────────────
Sorted by:
```

In the wide style, our data are back to having 154 observations, but now we have 47 variables! Variable `_1_age` contains `age` for $m = 1$, `_1_bmi` contains `bmi` for $m = 1$, `_2_age` contains `age` for $m = 2$, and so on.

Guideline 7 is

7. Generate passive variables.

Passive variables are variables derived from imputed variables. For instance, if we needed `lnage` = ln(`age`), variable `lnage` would be passive. Passive variables are easy to create; see [MI] **mi passive**. We are not going to need any passive variables in this example.

Guideline 8 is

8. Use `mi estimate` to fit models; see [MI] **mi estimate**.

Our data are wide right now, but that does not matter. We fit our model:

```
. mi estimate: logistic attack smokes age bmi hsgrad female

Multiple-imputation estimates                    Imputations     =         20
Logistic regression                              Number of obs   =        154
                                                 Average RVI     =     0.0547
                                                 Largest FMI     =     0.1377
DF adjustment:   Large sample                    DF:     min     =    1027.48
                                                         avg     =   55394.62
                                                         max     =  168501.59
Model F test:       Equal FMI                    F(   5,25165.6) =       3.35
Within VCE type:          OIM                    Prob > F        =     0.0050

------------------------------------------------------------------------------
      attack |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      smokes |   1.186791    .359663     3.30   0.001     .4818394    1.891743
         age |   .0297742   .0164346     1.81   0.070    -.0024699    .0620184
         bmi |   .1033297   .0468362     2.21   0.028     .0114494    .1952101
      hsgrad |   .1529883   .4033788     0.38   0.704    -.6376254     .943602
      female |   -.079329   .4145832    -0.19   0.848    -.8919049    .7332468
       _cons |  -5.100976   1.685697    -3.03   0.003    -8.408779   -1.793173
------------------------------------------------------------------------------
```

Those familiar with the `logistic` command will be surprised that `mi estimate: logistic` reported coefficients rather than odds ratios. That is because the estimation command is not `logistic` using `mi estimate`, it is `mi estimate` using `logistic`. If we wanted to see odds ratios at estimation time, we could have typed

```
. mi estimate, or:  logistic ...
```

By the same token, if we wanted to replay results, we would not type `logistic`, we would type `mi estimate`:

```
. mi estimate
  (output omitted)
```

If we wanted to replay results with odds ratios, we would type

```
. mi estimate, or
```

And that concludes the guidelines.

Also see

[MI] **intro** — Introduction to mi

[MI] **Glossary**