Title

stata.com

mi passive — Generate/replace and register passive variables

Description

`mi passive` creates and registers passive variables or replaces the contents of existing passive variables.

More precisely, `mi passive` executes the specified `generate`, `egen`, or `replace` command on each of $m = 0$, $m = 1$, ..., $m = M$. If the command is `generate` or `egen`, then `mi passive` registers the new variable as passive. If the command is `replace`, then `mi passive` verifies that the variable is already registered as passive.

Menu

Statistics > Multiple imputation

Syntax

```
mi passive: {generate|egen|replace} ...

mi passive: by varlist: {generate|egen|replace} ...
```

The full syntax is

```
mi passive[, options]: [by varlist [(varlist)]:] {generate|egen|replace} ...
```

options	Description
`noupdate`	see [MI] **noupdate option**
`nopreserve`	do not first `preserve`

Also see [D] **generate** and [D] **egen**.

Options

`noupdate` in some cases suppresses the automatic `mi update` this command might perform; see [MI] **noupdate option**.

`nopreserve` is a programmer's option. It specifies that `mi passive` is not to `preserve` the data if it ordinarily would. This is used by programmers who have already preserved the data before calling `mi passive`.

Remarks and examples

stata.com

Remarks are presented under the following headings:

mi passive basics

A passive variable is a variable that is a function of imputed variables or of other passive variables. For instance, if variable `age` were imputed and you created `lnage` from it, the `lnage` variable would be passive. The right way to create `lnage` is to type

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

Simply typing

```
. generate lnage = ln(age)
```

is not sufficient because that would create `lnage` in the $m = 0$ data, and `age`, being imputed, varies across m. There are situations where omitting the `mi passive` prefix would be almost sufficient, namely, when the data are mlong or flong style, but even then you would need to follow up by typing `mi register passive lnage`.

To create passive variables or to change the values of existing passive variables, use `mi passive`. Passive variables cannot be super-varying; see *mi passive and super-varying variables*.

mi passive works with the by prefix

You can use `mi passive` with the `by` prefix. For instance, you can type

```
. mi passive: by person: generate totaltodate = sum(amount)
```

You do not need to sort the data before issuing either of these commands, nor are you required to specify `by`'s `sort` option. `mi passive` handles sorting issues for you.

Use `by`'s parenthetical syntax to specify the order within `by`, if that is necessary. For instance,

```
. mi passive: by person (time): generate lastamount = amount[_n-1]
```

Do not omit the parenthetical `time` and instead attempt to sort the data yourself:

```
. sort person time
. mi passive: by person: generate lastamount = amount[_n-1]
```

Sorting the data yourself will work if your data happen to be wide style; it will not work in general.

mi passive works fastest with the wide style

`mi passive` works with any style, but it works fastest when the data are wide style. If you are going to issue multiple `mi passive` commands, you can usually speed execution by first converting your data to the wide style; see [MI] **mi convert**.

mi passive and super-varying variables

You should be careful not to mistakenly use `mi passive` to create super-varying variables. Super-varying variables cannot be passive variables because the values of a super-varying variable differ not only in the incomplete observations but also in the complete observations across imputations.

As noted in [MI] **mi set**, super-varying variables should never be registered. If a super-varying variable is registered as passive, it will be converted to a varying variable. All complete observations of the super-varying variable in each imputation will be replaced with their values from $m = 0$.

`mi passive` registers the created variable as passive. Even if the command you use with `mi passive` creates a super-varying variable, `mi passive` will convert it to varying, as described above.

You can use `mi passive` with any function that produces values that solely depend on values within the observation. In general, you cannot use `mi passive` with functions that produce values that depend on groups of observations.

For example, most `egen` functions result in super-varying variables. In such cases, you should use `mi xeq: egen` to create them and leave them unregistered; see [MI] **mi xeq**. You might thus conclude that you should never use `mi passive` with `egen`. That is not true, but it is nearly true. You may use `mi passive` with `egen`'s `rowmean()` function, for instance, because it produces values that depend only on one observation at a time.

Renaming passive variables

Use `mi rename` (see [MI] **mi rename**) to rename all variables, not just passive variables:

```
. mi rename oldname newname
```

`rename` (see [D] **rename**) is insufficient for renaming passive variables regardless of the style of your data.

Dropping passive variables

Use `drop` (see [D] **drop**) to drop variables (or observations), but run `mi update` (see [MI] **mi update**) afterward.

```
. drop var_or_vars
. mi update
```

This advice applies for all variables, not just passive ones.

Update passive variables when imputed values change

Passive variables are not automatically updated when the values of the underlying imputed variables change.

If imputed values change or if you add more imputations, you must update or re-create the passive variables. If you have several passive variables, we suggest you make a do-file to create them. You can run the do-file again whenever necessary. A do-file to create `lnage` and `totaltodate` might read

begin `cr_passive.do`

```
use mydata, clear
capture drop lnage
capture drop totaltodate
mi update
mi passive: generate lnage = ln(age)
mi passive: by person (time): generate totaltodate = sum(amount)
```

end `cr_passive.do`

Alternatives to mi passive

`mi passive` can run any `generate`, `replace`, or `egen` command. If that is not sufficient to create the variable you need, you will have to create the variable for yourself. Here is how you do that:

1. If your data are wide or mlong, use `mi convert` (see [MI] **mi convert**) to convert them to one of the fully long styles, flong or flongsep, and then continue with the appropriate step below.
2. If your data are flong, `mi` system variable `_mi_m` records m. Create your new variable by using standard Stata commands, but do that `by _mi_m`. After creating the variable, `mi register` it as passive; see [MI] **mi set**.
3. If your data are flongsep, create the new variable in each of the $m = 0$, $m = 1$, …, $m = M$ datasets, and then register the result. Start by working with a copy of your data:

   ```
   . mi copy newname
   ```

   The data in memory at this point correspond to $m = 0$. Create the new variable and then save the data:

   ```
   . (create new_variable)
   . save newname, replace
   ```

   Now use the $m = 1$ data and repeat the process:

   ```
   . use _1_newname
   . (create new_variable)
   . save _1_newname, replace
   ```

   Repeat for $m = 2$, $m = 3$, …, $m = M$.

   At this point, the new variable is created but not yet registered. Reload the original $m = 0$ data, register the new variable as passive, and run `mi update` (see [MI] **mi update**):

   ```
   . use newname
   . register passive new_variable
   . mi update
   ```

Finally, copy the result back to your original flongsep data,

```
. mi copy name, replace
```

or if you started with mlong, flong, or wide data, then convert the data back to your preferred style:

```
. mi convert original_style
```

Either way, erase the *newname* flongsep dataset collection:

```
. mi erase newname
```

The third procedure can be tedious and error-prone if M is large. We suggest that you make a do-file to create the variable and then run it on each of the $m = 0$, $m = 1$, ..., $m = M$ datasets:

```
. mi copy newname
. do mydofile
. save newname, replace
. forvalues m=1(1)20 {              // we assume M=20
>    use _'m'_newname
>    do mydofile
>    save _'m'_newname, replace
> }
. use newname
. register passive new_variable
. mi update
```

Also see

[MI] **Intro** — Introduction to mi

[MI] **mi reset** — Reset imputed or passive variables

[MI] **mi xeq** — Execute command(s) on individual imputations