Title

stata.com

mi xeq — Execute command(s) on individual imputations

Syntax Description Remarks and examples Stored results Also see

Syntax

`mi xeq` [*numlist*]`:` *command* [`;` *command* [`; ...`]]

Description

`mi xeq:` *XXX* executes *XXX* on $m = 0$, $m = 1$, ..., $m = M$.

`mi xeq` *numlist*`:` *XXX* executes *XXX* on $m =$ *numlist*.

XXX can be any single command or it can be multiple commands separated by a semicolon. If specifying multiple commands, the delimiter must not be set to semicolon; see [P] **#delimit**.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Using mi xeq with reporting commands
Using mi xeq with data-modification commands
Using mi xeq with data-modification commands on flongsep data

Using mi xeq with reporting commands

By reporting commands, we mean any general Stata command that reports results but leaves the data unchanged. `summarize` (see [R] **summarize**) is an example. `mi xeq` is especially useful with such commands. If you wanted to see the summary statistics for variables `outcome` and `age` among the females in your `mi` data, you could type

```
. mi xeq: summarize outcome age if sex=="female"

m=0 data:
-> summarize outcome age if sex=="female"
     (output omitted)
m=1 data:
-> summarize outcome age if sex=="female"
     (output omitted)
m=2 data:
-> summarize outcome age if sex=="female"
     (output omitted)
```

$M = 2$ in the data above.

If you wanted to see a particular regression run on the $m = 2$ data, you could type

```
. mi xeq 2: regress outcome age bp

m=2 data:
-> regress outcome age bp
     (output omitted)
```

In both cases, once the command executes, the entire `mi` dataset is brought back into memory.

Using mi xeq with data-modification commands

You can use data-modification commands with `mi xeq` but doing that is not especially useful unless you are using flongsep data.

If variable `lnage` were registered as passive and you wanted to update its values, you could type

```
. mi xeq: replace lnage = ln(age)
(output omitted)
```

That would work regardless of style, although it is just as easy to update the variable using `mi passive` (see [MI] **mi passive**):

```
. mi passive: replace lnage = ln(age)
(output omitted)
```

If what you are doing depends on the sort order of the data, include the `sort` command among the commands to be executed; do not assume that the individual datasets will be sorted the way the data in memory are sorted. For instance, if you have passive variable `totalx`, do not type

```
. sort id time
. mi xeq: by id: replace totalx = sum(x)
```

That will not work. Instead, type

```
. mi xeq: sort id time;  by id: replace totalx = sum(x)

m=0 data:
-> sort id time
-> by id: replace total x = sum(x)
(8 changes made)

m=1 data:
-> sort id time
-> by id: replace total x = sum(x)
(8 changes made)

m=2 data:
-> sort id time
-> by id: replace total x = sum(x)
(8 changes made)
```

Again we note that it would be just as easy to update this variable with `mi passive`:

```
. mi passive: by id (time): replace totalx = sum(x)
m=0:
(8 changes made)
m=1:
(8 changes made)
m=2:
(8 changes made)
```

With the wide, mlong, and flong styles, there is always another way to proceed, and often the other way is easier.

Using mi xeq with data-modification commands on flongsep data

With flongsep data, `mi xeq` is especially useful. Consider the case where you want to add new variable `lnage` = `ln(age)` to your data, and `age` is just a regular or unregistered variable. With flong, mlong, or wide data, you would just type

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

and be done with it.

With flongsep data, you have multiple datasets to update. Of course, you could `mi convert` (see [MI] **mi convert**) your data to one of the other styles, but we will assume that if you had sufficient memory to do that, you would have done that long ago and so would not be using flongsep data.

The easy way to create `lnage` with flongsep data is by typing

```
. mi xeq: gen lnage = ln(age)
(output omitted)
```

You could use the `mi xeq` approach with any of the styles, but with flong, mlong, or wide data, it is not necessary. With flongsep, it is.

Stored results

`mi xeq` stores in `r()` whatever the last command run on the last imputation or specified imputation returns. For instance,

```
. mi xeq: tabulate g ;  summarize x
```

returns the stored results for `summarize x` run on $m = M$.

```
. mi xeq 1 2: tabulate g ;  summarize x
```

returns the stored results for `summarize x` run on $m = 2$.

```
. mi xeq 0: summarize x
```

returns the stored results for `summarize x` run on $m = 0$.

Also see

[MI] **intro** — Introduction to mi

[MI] **mi passive** — Generate/replace and register passive variables