xi — Interaction expansion

### Syntax

```stata
xi [, prefix(string) noomit] term(s)
```

```stata
xi [, prefix(string) noomit] : any_stata_command varlist_with_terms ...
```

where a `term` has the form

- `i.varname` or `I.varname`
- `i.varname1*i.varname2` or `I.varname1*I.varname2`
- `i.varname1*varname3` or `I.varname1*varname3`
- `i.varname1|varname3` or `I.varname1|varname3`

`varname`, `varname1`, and `varname2` denote numeric or string categorical variables. `varname3` denotes a continuous, numeric variable.

### Menu

Data > Create or change data > Other variable-creation commands > Interaction expansion

Most commands in Stata now allow factor variables; see [U] 11.4.3 Factor variables. To determine if a command allows factor variables, see the information printed below the options table for the command. If the command allows factor variables, it will say something like “`indepvars` may contain factor variables”.

We recommend that you use factor variables instead of `xi` if a command allows factor variables.

We include [R] xi in our documentation so that readers can consult it when using a Stata command that does not allow factor variables.

### Description

`xi` expands terms containing categorical variables into indicator (also called dummy) variable sets by creating new variables and, in the second syntax (`xi: any_stata_command`), executes the specified command with the expanded terms. The dummy variables created are

- `i.varname` creates dummies for categorical variable `varname`
- `i.varname1*i.varname2` creates dummies for categorical variables `varname1` and `varname2`:
  - all interactions and main effects
- `i.varname1*varname3` creates dummies for categorical variable `varname1` and continuous variable `varname3`:
  - all interactions and main effects
- `i.varname1|varname3` creates dummies for categorical variable `varname1` and continuous variable `varname3`:
  - all interactions and main effect of `varname3`,
  - but no main effect of `varname1`
Options

prefix(string) allows you to choose a prefix other than _I for the newly created interaction variables. The prefix cannot be longer than four characters. By default, xi will create interaction variables starting with _I. When you use xi, it drops all previously created interaction variables starting with the prefix specified in the prefix(string) option or with _I by default. Therefore, if you want to keep the variables with a certain prefix, specify a different prefix in the prefix(string) option.

noomit prevents xi from omitting groups. This option provides a way to generate an indicator variable for every category having one or more variables, which is useful when combined with the noconstant option of an estimation command.

Remarks and examples

Remarks are presented under the following headings:

Background
Indicator variables for simple effects
Controlling the omitted dummy
Categorical variable interactions
Interactions with continuous variables
Using xi: Interpreting output
How xi names variables
xi as a command rather than a command prefix
Warnings

xi provides a convenient way to include dummy or indicator variables when fitting a model (say, with regress or logistic). For instance, assume that the categorical variable agegrp contains 1 for ages 20–24, 2 for ages 25–39, 3 for ages 40–44, etc. Typing

```
.xi: logistic outcome weight i.agegrp bp
```

estimates a logistic regression of outcome on weight, dummies for each agegrp category, and bp. That is, xi searches out and expands terms starting with “i.” or “I.” but ignores the other variables. xi will expand both numeric and string categorical variables, so if you had a string variable race containing “white”, “black”, and “other”, typing

```
.xi: logistic outcome i.agegrp weight i.race bp
```

would include indicator variables for the race group as well.

The i. indicator variables xi expands may appear anywhere in the varlist, so

```
.xi: logistic outcome i.agegrp weight i.race bp
```

would fit the same model.

You can also create interactions of categorical variables; typing

```
xi: logistic outcome weight bp i.agegrp*i.race
```

fits a model with indicator variables for all agegrp and race combinations, including the agegrp and race main-effect terms (that is, the terms that are created when you just type i.agegrp i.race).

You can interact dummy variables with continuous variables; typing

```
xi: logistic outcome bp i.agegrp*weight i.race
```
fits a model with indicator variables for all `agegrp` categories interacted with `weight`, plus the main-effect terms `weight` and `i.agegrp`.

You can get the interaction terms without the `agegrp` main effect (but with the `weight` main effect) by typing

```
xi: logistic outcome bp i.agegrp|weight i.race
```

You can also include multiple interactions:

```
xi: logistic outcome bp i.agegrp*weight i.agegrp*i.race
```

We will now back up and describe the construction of dummy variables in more detail.

**Background**

The terms *continuous*, *categorical*, and *indicator* or *dummy* variables are used below. Continuous variables measure something—such as height or weight—and at least conceptually can take on any real number over some range. Categorical variables, on the other hand, take on a finite number of values, each denoting membership in a subclass—for example, excellent, good, and poor, which might be coded 0, 1, 2, or 1, 2, 3, or even “Excellent”, “Good”, and “Poor”. An indicator or dummy variable—the terms are used interchangeably—is a special type of two-valued categorical variable that contains values 0, denoting false, and 1, denoting true. The information contained in any \( k \)-valued categorical variable can be equally well represented by \( k \) indicator variables. Instead of one variable recording values representing excellent, good, and poor, you can have three indicator variables, indicating the truth or falseness of “result is excellent”, “result is good”, and “result is poor”.

`xi` provides a convenient way to convert categorical variables to dummy or indicator variables when you fit a model (say, with `regress` or `logistic`).

### Example 1

For instance, assume that the categorical variable `agegrp` contains 1 for ages 20–24, 2 for ages 25–39, and 3 for ages 40–44. (There is no one over 44 in our data.) As it stands, `agegrp` would be a poor candidate for inclusion in a model even if we thought age affected the outcome. The reason is that the coding would restrict the effect of being in the second age group to be twice the effect of being in the first, and, similarly, the effect of being in the third to be three times the first. That is, if we fit the model,

\[
y = \beta_0 + \beta_1 \text{agegrp} + X \beta_2
\]

the effect of being in the first age group is \( \beta_1 \), the second \( 2\beta_1 \), and the third \( 3\beta_1 \). If the coding 1, 2, and 3 is arbitrary, we could just as well have coded the age groups 1, 4, and 9, making the effects \( \beta_1, 4\beta_1, \) and \( 9\beta_1 \).

The solution is to convert the categorical variable `agegrp` to a set of indicator variables, \( a_1, a_2, \) and \( a_3 \), where \( a_i \) is 1 if the individual is a member of the \( i \)th age group and 0 otherwise. We can then fit the model

\[
y = \beta_0 + \beta_1 a_1 + \beta_2 a_2 + \beta_3 a_3 + X \beta_2
\]

The effect of being in age group 1 is now \( \beta_1 \); 2, \( \beta_1 \); and 3, \( \beta_1 \); and these results are independent of our (arbitrary) coding. The only difficulty at this point is that the model is unidentified in the sense that there are an infinite number of \( (\beta_0, \beta_1, \beta_2, \beta_3) \) that fit the data equally well.
To see this, pretend that \((\beta_0, \beta_{11}, \beta_{12}, \beta_{13}) = (1, 1, 3, 4)\). The predicted values of \(y\) for the various age groups are

\[
y = \begin{cases} 
1 + 1 + X\beta_2 = 2 + X\beta_2 & \text{(age group 1)} \\
1 + 3 + X\beta_2 = 4 + X\beta_2 & \text{(age group 2)} \\
1 + 4 + X\beta_2 = 5 + X\beta_2 & \text{(age group 3)} 
\end{cases}
\]

Now pretend that \((\beta_0, \beta_{11}, \beta_{12}, \beta_{13}) = (2, 0, 2, 3)\). The predicted values of \(y\) are

\[
y = \begin{cases} 
2 + 0 + X\beta_2 = 2 + X\beta_2 & \text{(age group 1)} \\
2 + 2 + X\beta_2 = 4 + X\beta_2 & \text{(age group 2)} \\
2 + 3 + X\beta_2 = 5 + X\beta_2 & \text{(age group 3)} 
\end{cases}
\]

These two sets of predictions are indistinguishable: for age group 1, \(y = 2 + X\beta_2\) regardless of the coefficient vector used, and similarly for age groups 2 and 3. This arises because we have three equations and four unknowns. Any solution is as good as any other, and, for our purposes, we merely need to choose one of them. The popular selection method is to set the coefficient on the first indicator variable to 0 (as we have done in our second coefficient vector). This is equivalent to fitting the model

\[y = \beta_0 + \beta_{12}a_2 + \beta_{13}a_3 + X\beta_2\]

How we select a particular coefficient vector (identifies the model) does not matter. It does, however, affect the interpretation of the coefficients.

For instance, we could just as well choose to omit the second group. In our artificial example, this would yield \((\beta_0, \beta_{11}, \beta_{12}, \beta_{13}) = (4, -2, 0, 1)\) instead of \((2, 0, 2, 3)\). These coefficient vectors are the same in the sense that

\[
y = \begin{cases} 
2 + 0 + X\beta_2 = 2 + X\beta_2 & \text{(age group 1)} \\
2 + 2 + X\beta_2 = 4 + X\beta_2 & \text{(age group 2)} \\
2 + 3 + X\beta_2 = 5 + X\beta_2 & \text{(age group 3)} 
\end{cases}
\]

But what does it mean that \(\beta_{13}\) can just as well be 3 or 1? We obtain \(\beta_{13} = 3\) when we set \(\beta_{11} = 0\), so \(\beta_{13} = \beta_{13} - \beta_{11}\) and \(\beta_{13}\) measures the difference between age groups 3 and 1.

In the second case, we obtain \(\beta_{13} = 1\) when we set \(\beta_{12} = 0\), so \(\beta_{13} - \beta_{12} = 1\) and \(\beta_{13}\) measures the difference between age groups 3 and 2. There is no inconsistency. According to our \(\beta_{12} = 0\) model, the difference between age groups 3 and 1 is \(\beta_{13} - \beta_{11} = 1 - (-2) = 3\), the same result we got in the \(\beta_{11} = 0\) model.

Example 2

The issue of interpretation is important because it can affect the way we discuss results. Imagine that we are studying recovery after a coronary bypass operation. Assume that the age groups are children under 13 (we have two of them), young adults under 25 (we have a handful of them), adults under 46 (of which we have even more), mature adults under 56, older adults under 65, and elderly adults. We follow the prescription of omitting the first group, so all our results are reported relative to children under 13. While there is nothing statistically wrong with this, readers will be suspicious when we make statements like “compared with young children, older and elder adults . . .”. Moreover, we will probably have to end each statement with “although results are not statistically significant” because we have only two children in our comparison group. Of course, even with results reported in this way, we can do reasonable comparisons (say, with mature adults), but we will have to do extra work to perform the appropriate linear hypothesis test using Stata’s \texttt{test} command.
Here it would be better to force the omitted group to be more reasonable, such as mature adults. There is, however, a generic rule for automatic comparison group selection that, although less popular, tends to work better than the omit-the-first-group rule. That rule is to omit the most prevalent group. The most prevalent is usually a reasonable baseline.

In any case, the prescription for categorical variables is

1. Convert each \( k \)-valued categorical variable to \( k \) indicator variables.

2. Drop one of the \( k \) indicator variables; any one will do, but dropping the first is popular, dropping the most prevalent is probably better in terms of having the computer guess at a reasonable interpretation, and dropping a specified one often eases interpretation the most.

3. Fit the model on the remaining \( k - 1 \) indicator variables.

\( \text{xi} \) automates this procedure.

We will now consider each of \( \text{xi} \)'s features in detail.

**Indicator variables for simple effects**

When you type \( \text{i.varname} \), \( \text{xi} \) internally tabulates \( \text{varname} \) (which may be a string or a numeric variable) and creates indicator (dummy) variables for each observed value, omitting the indicator for the smallest value. For instance, say that \( \text{agegrp} \) takes on the values 1, 2, 3, and 4. Typing

\[ \text{xi: logistic outcome i.agegrp} \]

creates indicator variables named \( _{I\text{agegrp}_2} \), \( _{I\text{agegrp}_3} \), and \( _{I\text{agegrp}_4} \). (\( \text{xi} \) chooses the names and tries to make them readable; \( \text{xi} \) guarantees that the names are unique.) The expanded logistic model is

\[ \text{logistic outcome}_{_{I\text{agegrp}_2}} \ _{I\text{agegrp}_3} \ _{I\text{agegrp}_4} \]

Afterward, you can drop the new variables \( \text{xi} \) leaves behind by typing ‘\text{drop}_{_I^*}’ (note the capitalization).

\( \text{xi} \) provides the following features when you type \( \text{i.varname} \):

- \( \text{varname} \) may be string or numeric.
- Dummy variables are created automatically.
- By default, the dummy-variable set is identified by dropping the dummy corresponding to the smallest value of the variable (how to specify otherwise is discussed below).
- The new dummy variables are left in your dataset. By default, the names of the new dummy variables start with \( _I \); therefore, you can drop them by typing ‘\text{drop}_{_I^*}’. You do not have to do this; each time you use \( \text{xi} \), any automatically generated dummies with the same prefix as the one specified in the \( \text{prefix(string)} \) option, or \( _I \) by default, are dropped and new ones are created.
- The new dummy variables have variable labels so that you can determine what they correspond to by typing ‘\text{describe}’.
- \( \text{xi} \) may be used with any Stata command (not just \text{logistic}).
Controlling the omitted dummy

By default, `i.varname` omits the dummy corresponding to the smallest value of `varname`; for a string variable, this is interpreted as dropping the first in an alphabetical, case-sensitive sort. `xi` provides two alternatives to dropping the first: `xi` will drop the dummy corresponding to the most prevalent value of `varname`, or `xi` will let you choose the particular dummy to be dropped.

To change `xi`'s behavior to dropping the most prevalent dummy, type

```
    . char _dta[omit] prevalent
```

although whether you type “prevalent” or “yes” or anything else does not matter. Setting this characteristic affects the expansion of all categorical variables in the dataset. If you resave your dataset, the prevalent preference will be remembered. If you want to change the behavior back to the default drop-the-first rule, type

```
    . char _dta[omit]
```
to clear the characteristic.

Once you set `_dta[omit]`, `i.varname` omits the dummy corresponding to the most prevalent value of `varname`. Thus the coefficients on the dummies have the interpretation of change from the most prevalent group. For example,

```
    . char _dta[omit] prevalent
    . xi: regress y i.agegrp
```
might create `_Iagegrp_1` through `_Iagegrp_4`, resulting in `_Iagegrp_2` being omitted if `agegrp = 2` is most common (as opposed to the default dropping of `_Iagegrp_1`). The model is then

\[
y = b_0 + b_1 \_Iagegrp\_1 + b_3 \_Iagegrp\_3 + b_4 \_Iagegrp\_4 + u
\]

Then

- Predicted $y$ for `agegrp` 1 = $b_0 + b_1$
- Predicted $y$ for `agegrp` 3 = $b_0 + b_3$
- Predicted $y$ for `agegrp` 2 = $b_0$
- Predicted $y$ for `agegrp` 4 = $b_0 + b_4$

Thus the model’s reported $t$ or $Z$ statistics are for a test of whether each group is different from the most prevalent group.

Perhaps you wish to omit the dummy for `agegrp` 3 instead. You do this by setting the variable’s `omit` characteristic:

```
    . char agegrp[omit] 3
```

This overrides `_dta[omit]` if you have set it. Now when you type

```
    . xi: regress y i.agegrp
```

`_Iagegrp_3` will be omitted, and you will fit the model

\[
y = b'_0 + b'_1 \_Iagegrp\_1 + b'_2 \_Iagegrp\_2 + b'_4 \_Iagegrp\_4 + u
\]

Later if you want to return to the default omission, type

```
    . char agegrp[omit]
```
to clear the characteristic.

In summary, `i.varname` omits the first group by default, but if you define

```
    . char _dta[omit] prevalent
```
the default behavior changes to dropping the most prevalent group. Either way, if you define a characteristic of the form

```
. char varname[omit] #
```
or, if `varname` is a string,

```
. char varname[omit] string-literal
```
the specified value will be omitted.

Examples:

```
. char agegrp[omit] 1
. char race[omit] White  (for race, a string variable)
. char agegrp[omit] (to restore default for agegrp)
```

### Categorical variable interactions

`i.varname_1*i.varname_2` creates the dummy variables associated with the interaction of the categorical variables `varname_1` and `varname_2`. The identification rules—which categories are omitted—are the same as those for `i.varname`. For instance, assume that `agegrp` takes on four values and `race` takes on three values. Typing

```
. xi: regress y i.agegrp*i.race
```
results in

\[
\begin{align*}
y &= a + b_2 \_Iagegrp_2 + b_3 \_Iagegrp_3 + b_4 \_Iagegrp_4 \quad \text{(agegrp)} \\
&\quad + c_2 \_Irace_2 + c_3 \_Irace_3 \quad \text{(race)} \\
&\quad + d_{22} \_IageXrac_2_2 + d_{23} \_IageXrac_2_3 \\
&\quad + d_{32} \_IageXrac_3_2 + d_{33} \_IageXrac_3_3 \quad \text{(agegrp*race)} \\
&\quad + u
\end{align*}
\]

That is, typing

```
. xi: regress y i.agegrp*i.race
```
is the same as typing

```
. xi: regress y i.agegrp i.race i.agegrp*i.race
```

Although there are many other ways the interaction could have been parameterized, this method has the advantage that you can test the joint significance of the interactions by typing

```
. testparm _IageXrac*
```

When you perform the estimation step, whether you specify `i.agegrp*i.race` or `i.race*i.agegrp` makes no difference (other than in the names given to the interaction terms; in the first case, the names will begin with `_IageXrac`; in the second, `_RacXage`). Thus

```
. xi: regress y i.race*i.agegrp
```
fits the same model.

You may also include multiple interactions simultaneously:

```
. xi: regress y i.agegrp*i.race i.agegrp*i.sex
```
The model fit is

\[
\begin{align*}
\text{model:} & \quad y = a + b_2 \_\text{Iagegrp}_2 + b_3 \_\text{Iagegrp}_3 + b_4 \_\text{Iagegrp}_4 \\
& \quad + c_2 \_\text{Irace}_2 + c_3 \_\text{Irace}_3 \\
& \quad + d_{22} \_\text{IageXrac}_2 \_2 + d_{23} \_\text{IageXrac}_2 \_3 \\
& \quad + d_{32} \_\text{IageXrac}_3 \_2 + d_{33} \_\text{IageXrac}_3 \_3 \\
& \quad + d_{42} \_\text{IageXrac}_4 \_2 + d_{43} \_\text{IageXrac}_4 \_3 \\
& \quad + e_2 \_\text{Isex}_2 \\
& \quad + f_{22} \_\text{IageXsex}_2 \_2 + f_{23} \_\text{IageXsex}_2 \_3 + f_{24} \_\text{IageXsex}_2 \_4 \quad (\text{agegrp*sex}) \\
& \quad + u
\end{align*}
\]

The **agegrp** dummies are (correctly) included only once.

### Interactions with continuous variables

\( i.\text{varname}_1*\text{varname}_2 \) (as distinguished from \( i.\text{varname}_1*i.\text{varname}_2 \)—note the second \( i.\)) specifies an interaction of a categorical variable with a continuous variable. For instance,

\[ .\text{xi: regress y i.agegrp\_wgt} \]

results in the model

\[
\begin{align*}
y = a + b_2 \_\text{Iagegrp}_2 + b_3 \_\text{Iagegrp}_3 + b_4 \_\text{Iagegrp}_4 & \quad (\text{agegrp dummies}) \\
& \quad + c \_\text{wgt} \\
& \quad + d_2 \_\text{IageXwgt}_2 + d_3 \_\text{IageXwgt}_3 + d_4 \_\text{IageXwgt}_4 & \quad (\text{agegrp*wgt interactions}) \\
& \quad + u
\end{align*}
\]

A variation on this notation, using \( | \) rather than \( * \), omits the **agegrp** dummies. Typing

\[ .\text{xi: regress y i.agegrp\_\_wgt} \]

fits the model

\[
\begin{align*}
y = a' + c' \_\text{wgt} & \quad (\text{continuous wgt effect}) \\
& \quad + d'_2 \_\text{IageXwgt}_2 + d'_3 \_\text{IageXwgt}_3 + d'_4 \_\text{IageXwgt}_4 & \quad (\text{agegrp*wgt interactions}) \\
& \quad + u'
\end{align*}
\]

The predicted values of \( y \) are

<table>
<thead>
<tr>
<th>agegrp__wgt model</th>
<th>agegrp__wgt model</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = a + c _\text{wgt} )</td>
<td>( a' + c' _\text{wgt} )</td>
</tr>
<tr>
<td>( a + c _\text{wgt} + b_2 + d_2 _\text{wgt} )</td>
<td>( a' + c' _\text{wgt} + d'_2 _\text{wgt} )</td>
</tr>
<tr>
<td>( a + c _\text{wgt} + b_3 + d_3 _\text{wgt} )</td>
<td>( a' + c' _\text{wgt} + d'_3 _\text{wgt} )</td>
</tr>
<tr>
<td>( a + c _\text{wgt} + b_4 + d_4 _\text{wgt} )</td>
<td>( a' + c' _\text{wgt} + d'_4 _\text{wgt} )</td>
</tr>
</tbody>
</table>

That is, typing

\[ .\text{xi: regress y i.agegrp\_wgt} \]
is equivalent to typing

    . xi: regress y i.agegrp i.agegrp|wgt

In either case, you do not need to specify separately the continuous variable \texttt{wgt}; it is included automatically.

\textbf{Using xi: Interpreting output}

    . xi: regress mpg i.rep78
    i.rep78 _Irep78_1-5 (naturally coded; _Irep78_1 omitted)
    (output from regress appears)

Interpretation: \texttt{i.rep78} expanded to the dummies \texttt{_Irep78_1}, \texttt{_Irep78_2}, ..., \texttt{_Irep78_5}. The numbers on the end are “natural” in the sense that \texttt{_Irep78_1} corresponds to \texttt{rep78 = 1}, \texttt{_Irep78_2} to \texttt{rep78 = 2}, and so on. Finally, the dummy for \texttt{rep78 = 1} was omitted.

    . xi: regress mpg i.make
    i.make _Imake_1-74 (_Imake_1 for make==AMC Concord omitted)
    (output from regress appears)

Interpretation: \texttt{i.make} expanded to \texttt{_Imake_1}, \texttt{_Imake_2}, ..., \texttt{_Imake_74}. The coding is not natural because \texttt{make} is a string variable. \texttt{_Imake_1} corresponds to one make, \texttt{_Imake_2} to another, and so on. You can find out the coding by typing \texttt{describe}. \texttt{_Imake_1} for the AMC Concord was omitted.

\textbf{How xi names variables}

By default, \texttt{xi} assigns to the dummy variables it creates names having the form

\[ _{\text{Istub}} \_\text{groupid} \]

You may subsequently refer to the entire set of variables by typing \texttt{`_{\text{Istub}}*`}. For example,

<table>
<thead>
<tr>
<th>name</th>
<th>_I</th>
<th>stub</th>
<th>_+</th>
<th>groupid</th>
<th>Entire set</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Iagegrp_1</td>
<td>_I</td>
<td>agegrp</td>
<td>_</td>
<td>1</td>
<td>_Iagegrp*</td>
</tr>
<tr>
<td>_Iagegrp_2</td>
<td>_I</td>
<td>agegrp</td>
<td>_</td>
<td>2</td>
<td>_Iagegrp*</td>
</tr>
<tr>
<td>_IageXwgt_1</td>
<td>_I</td>
<td>ageXwgt</td>
<td>_</td>
<td>1</td>
<td>_IageXwgt*</td>
</tr>
<tr>
<td>_IageXrac_1_2</td>
<td>_I</td>
<td>ageXrac</td>
<td>_</td>
<td>1_2</td>
<td>_IageXrac*</td>
</tr>
<tr>
<td>_IageXrac_2_1</td>
<td>_I</td>
<td>ageXrac</td>
<td>_</td>
<td>2_1</td>
<td>_IageXrac*</td>
</tr>
</tbody>
</table>

If you specify a prefix in the \texttt{prefix(string)} option, say, \texttt{_{S}}, then \texttt{xi} will name the variables starting with the prefix

\[ _{\text{Sstub}} \_\text{groupid} \]
xi as a command rather than a command prefix

`xi` can be used as a command prefix or as a command by itself. In the latter form, `xi` merely creates the indicator and interaction variables. Typing

```
    . xi: regress y i.agegrp*wgt
    i.agegrp     _Iagegrp_1-4 (naturally coded; _Iagegrp_1 omitted)
    i.agegrp*wgt _IageXwgt_1-4 (coded as above)
    (output from regress appears)
```

is equivalent to typing

```
    . xi i.agegrp*wgt
    i.agegrp     _Iagegrp_1-4 (naturally coded; _Iagegrp_1 omitted)
    i.agegrp*wgt _IageXwgt_1-4 (coded as above)
    . regress y _Iagegrp* _IageXwgt*
    (output from regress appears)
```

Warnings

1. `xi` creates new variables in your dataset; most are byte, but interactions with continuous variables will have the storage type of the underlying continuous variable. You may get the message “insufficient memory”. If so, you will need to increase the amount of memory allocated to Stata’s data areas; see [U] 6 Managing memory.

2. When using `xi` with an estimation command, you may get the message “matsize too small”. If so, see [R] matsize.

Stored results

`xi` stores the following characteristics:

```
_dta[*_xi__Vars__Prefix__]*  prefix names
_dta[*_xi__Vars__To__Drop__]  variables created
```

References


Also see

[U] 11.1.10 Prefix commands

[U] 20 Estimation and postestimation commands