Title

stata.com

reshape — Convert data from wide to long form and vice versa

Description

`reshape` converts data from *wide* to *long* form and vice versa.

`set reshape_favor` specifies whether, when performing the data conversion, `reshape` should favor conserving memory (`memory`) or running quickly (`speed`). Historically, `reshape` favored conserving memory. Switching to `speed` will make `reshape` run faster at the cost of consuming more memory. You can easily revert to the default method for reshaping the data (`default`).

Quick start

Create `v` from 2 time periods stored in `v1` and `v2` for observations identified by `idvar` and add `tvar` identifying time period

```
reshape long v, i(idvar) j(tvar)
```

Create `v` from 2 subobservations stored in `v1` and `v2` for observations identified by `idvar` and add `subobs` identifying each subobservation

```
reshape long v, i(idvar) j(subobs)
```

Same as above, but allow `subobs` to contain strings

```
reshape long v, i(idvar) j(subobs) string
```

Undo results from above

```
reshape wide
```

Create `v1` and `v2` from `v` with observations identified by `idvar` and time period identified by `tvar`

```
reshape wide v, i(idvar) j(tvar)
```

Undo results from above

```
reshape long
```

Create `var` and time identifier `tvar` from `v1ar` and `v2ar` with observation identifier `idvar`

```
reshape long v@ar, i(idvar) j(tvar)
```

Menu

Data > Create or change data > Other variable-transformation commands > Convert data between wide and long

Syntax

Overview

long

i	*j*	*stub*
1	1	4.1
1	2	4.5
2	1	3.3
2	2	3.0

reshape ⟷

wide

i	*stub*1	*stub*2
1	4.1	4.5
2	3.3	3.0

To go from long to wide:

```
                                        j existing variable
                                       /
reshape wide stub, i(i) j(j)
```

To go from wide to long:

```
reshape long stub, i(i) j(j)
                                       \
                                        j new variable
```

To go back to long after using `reshape wide`:

```
reshape long
```

To go back to wide after using `reshape long`:

```
reshape wide
```

Basic syntax

Convert data from wide form to long form

`reshape long` *stubnames* `, i(`*varlist*`)` [*options*]

Convert data from long form to wide form

`reshape wide` *stubnames* `, i(`*varlist*`)` [*options*]

Convert data back to long form after using reshape wide

`reshape long`

Convert data back to wide form after using reshape long

`reshape wide`

List problem observations when reshape fails

`reshape error`

Specify default method for reshaping the data

`set reshape_favor` { `default` | `memory` | `speed` } [`, permanently`]

options	Description
* `i(`*varlist*`)`	use *varlist* as the ID variables
`j(`*varname* [*values*]`)`	long→wide: *varname*, existing variable wide→long: *varname*, new variable optionally specify values to subset *varname*
`string`	*varname* is a string variable (default is numeric)
`favor(`*favor*`)`	specify reshape method; *favor* may be `memory` or `speed`

* `i(`*varlist*`)` is required.

`reshape` does not allow alias variables; see [D] **frunalias** for advice on how to get around this restriction.

values is	`#[ -# ] [ #[ -# ] [ ... ] ]`	if *varname* is numeric (default)
	`"`*string*`" [ "`*string*`" [ ... ] ]`	if *varname* is string

stubnames are variable names (long→wide), or stubs of variable names (wide→long), and either way, may contain `@`, denoting where *j* appears or is to appear in the name.

In the example above, when we wrote "`reshape wide` *stub*", we could have written "`reshape wide` *stub*`@`" because *j* by default ends up as a suffix. Had we written *stu*`@`*b*, then the wide variables would have been named *stu*`1`*b* and *stu*`2`*b*.

Advanced syntax

`reshape i` *varlist*

`reshape j` *varname* [*values*] [`, string`]

`reshape xij` *fvarnames* [`, atwl(`*chars*`)`]

`reshape xi` [*varlist*]

`reshape favor` { `memory` | `speed` }

`reshape` [`query`]

`reshape clear`

Options

`i(`*varlist*`)` specifies the variables whose unique values denote a logical observation. `i()` is required.

`j(`*varname* [*values*]`)` specifies the variable whose unique values denote a subobservation. *values* lists the unique values to be used from *varname*, which typically are not explicitly stated because `reshape` will determine them automatically from the data.

`string` specifies that `j()` may contain string values.

`atwl(`*chars*`)`, available only with the advanced syntax and not shown in the dialog box, specifies that plain ASCII *chars* be substituted for the `@` character when converting the data from wide to long form.

`favor(`*favor*`)` specifies the method for reshaping the data. Historically, `reshape` was coded to minimize its use of memory; this is `favor(memory)`. With `favor(speed)`, the focus is to accomplish the reshape faster at the cost of using more memory.

`permanently` specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

stata.com

Remarks are presented under the following headings:

See Mitchell (2020, chap. 9) for information and examples using `reshape`.

Description of basic syntax

Before using `reshape`, you need to determine whether the data are in long or wide form. You also must determine the logical observation (`i`) and the subobservation (`j`) by which to organize the data. Suppose that you had the following data, which could be organized in wide or long form as follows:

i		..... X_{ij}		
id	sex	inc80	inc81	inc82
1	0	5000	5500	6000
2	1	2000	2200	3300
3	0	3000	2000	1000

i	j		X_{ij}
id	year	sex	inc
1	80	0	5000
1	81	0	5500
1	82	0	6000
2	80	1	2000
2	81	1	2200
2	82	1	3300
3	80	0	3000
3	81	0	2000
3	82	0	1000

Given these data, you could use `reshape` to convert from one form to the other:

```
. reshape long inc, i(id) j(year)          /* goes from left form to right */
. reshape wide inc, i(id) j(year)          /* goes from right form to left */
```

Because we did not specify `sex` in the command, Stata assumes that it is constant within the logical observation, here `id`.

Wide and long data forms

Think of the data as a collection of observations X_{ij}, where i is the logical observation, or group identifier, and j is the subobservation, or within-group identifier.

Wide-form data are organized by logical observation, storing all the data on a particular observation in one row. Long-form data are organized by subobservation, storing the data in multiple rows.

▷ Example 1

For example, we might have data on a person's ID, gender, and annual income over the years 1980–1982. We have two X_{ij} variables with the data in wide form:

```
. use https://www.stata-press.com/data/r18/reshape1
. list
```

	id	sex	inc80	inc81	inc82	ue80	ue81	ue82
1.	1	0	5000	5500	6000	0	1	0
2.	2	1	2000	2200	3300	1	0	0
3.	3	0	3000	2000	1000	0	0	1

To convert these data to the long form, we type

```
. reshape long inc ue, i(id) j(year)
(j = 80 81 82)
Data                               Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                3   ->   9
Number of variables                   8   ->   5
j variable (3 values)                     ->   year
xij variables:
                     inc80 inc81 inc82    ->   inc
                        ue80 ue81 ue82    ->   ue
-----------------------------------------------------------------------------
```

There is no variable named `year` in our original, wide-form dataset. `year` will be a new variable in our long dataset. After this conversion, we have

```
. list, sep(3)
```

	id	year	sex	inc	ue
1.	1	80	0	5000	0
2.	1	81	0	5500	1
3.	1	82	0	6000	0
4.	2	80	1	2000	1
5.	2	81	1	2200	0
6.	2	82	1	3300	0
7.	3	80	0	3000	0
8.	3	81	0	2000	0
9.	3	82	0	1000	1

We can return to our original, wide-form dataset by using `reshape wide`.

```
. reshape wide inc ue, i(id) j(year)
(j = 80 81 82)
Data                                Long   ->   Wide
-----------------------------------------------------------------------------
Number of observations                 9   ->   3
Number of variables                    5   ->   8
j variable (3 values)               year   ->   (dropped)
xij variables:
                                     inc   ->   inc80 inc81 inc82
                                      ue   ->   ue80 ue81 ue82
-----------------------------------------------------------------------------
```

```
. list
```

	id	inc80	ue80	inc81	ue81	inc82	ue82	sex
1.	1	5000	0	5500	1	6000	0	0
2.	2	2000	1	2200	0	3300	0	1
3.	3	3000	0	2000	0	1000	1	0

Converting from wide to long creates the `j` (`year`) variable. Converting back from long to wide drops the `j` (`year`) variable.

◁

❑ Technical note

If your data are in wide form and you do not have a group identifier variable (the `i(`*varlist*`)` required option), you can create one easily by using `generate`; see [D] **generate**. For instance, in the last example, if we did not have the `id` variable in our dataset, we could have created it by typing

```
. generate id = _n
```

❑

Avoiding and correcting mistakes

`reshape` often detects when the data are not suitable for reshaping; an error is issued, and the data remain unchanged.

▷ Example 2

The following wide data contain a mistake:

```
. use https://www.stata-press.com/data/r18/reshape2, clear
. list
```

	id	sex	inc80	inc81	inc82
1.	1	0	5000	5500	6000
2.	2	1	2000	2200	3300
3.	3	0	3000	2000	1000
4.	2	0	2400	2500	2400

```
. reshape long inc, i(id) j(year)
(j = 80 81 82)
variable id does not uniquely identify the observations
    Your data are currently wide. You are performing a reshape long. You
    specified i(id) and j(year). In the current wide form, variable id should
    uniquely identify the observations. Remember this picture:
```

long

i	j	a	b
1	1	1	2
1	2	3	4
2	1	5	6
2	2	7	8

<--- reshape --->

wide

i	a1	a2	b1	b2
1	1	3	2	4
2	5	7	6	8

```
    Type reshape error for a list of the problem observations.
r(9);
```

The `i` variable must be unique when the data are in the wide form; we typed `i(id)`, yet we have 2 observations for which `id` is 2. (Is person 2 a male or female?)

◁

▷ Example 3

It is not a mistake when the `i` variable is repeated when the data are in long form, but the following data have a similar mistake:

```
. use https://www.stata-press.com/data/r18/reshapexp1
. list
```

	id	year	sex	inc
1.	1	80	0	5000
2.	1	81	0	5500
3.	1	81	0	5400
4.	1	82	0	6000

```
. reshape wide inc, i(id) j(year)
(j = 80 81 82)
values of variable year not unique within id
    Your data are currently long. You are performing a reshape wide. You
    specified i(id) and j(year). There are observations within i(id) with the
    same value of j(year). In the long data, variables i() and j() together
    must uniquely identify the observations.
```

long

i	j	a	b
1	1	1	2
1	2	3	4
2	1	5	6
2	2	7	8

<--- reshape --->

wide

i	a1	a2	b1	b2
1	1	3	2	4
2	5	7	6	8

```
    Type reshape error for a list of the problem variables.
r(9);
```

In the long form, `i(id)` does not have to be unique, but `j(year)` must be unique within `i`; otherwise, what is the value of `inc` in 1981 for which `id==1`?

reshape told us to type reshape error to view the problem observations.

```
. reshape error
(j = 80 81 82)
i (id) indicates the top-level grouping such as subject id.
j (year) indicates the subgrouping such as time.
The data are in the long form; j should be unique within i.
There are multiple observations on the same year within id.
The following 2 of 4 observations have repeated year values:
```

	id	year
2.	1	81
3.	1	81

```
(data now sorted by id year)
```

◁

▷ Example 4

Consider some long-form data that have no mistakes. We list the first 4 observations.

```
. use https://www.stata-press.com/data/r18/reshape6
. list in 1/4
```

	id	year	sex	inc	ue
1.	1	80	0	5000	0
2.	1	81	0	5500	1
3.	1	82	0	6000	0
4.	2	80	1	2000	1

Say that when converting the data to wide form, however, we forget to mention the ue variable (which varies within person).

```
. reshape wide inc, i(id) j(year)
(j = 80 81 82)
variable ue not constant within id
    Your data are currently long.  You are performing a reshape wide.  You
    typed something like
        . reshape wide a b, i(id) j(year)
    There are variables other than a, b, id, year in your data.  They must be
    constant within id because that is the only way they can fit into wide
    data without loss of information.
    The variable or variables listed above are not constant within id.
    Perhaps the values are in error.  Type reshape error for a list of the
    problem observations.
    Either that, or the values vary because they should vary, in which case
    you must either add the variables to the list of xij variables to be
    reshaped, or drop them.
r(9);
```

Here reshape observed that ue was not constant within id and so could not restructure the data so that there were single observations on id. We should have typed

```
. reshape wide inc ue, i(id) j(year)
```

◁

In summary, there are three cases in which reshape will refuse to convert the data:

1. The data are in wide form and i is not unique.
2. The data are in long form and j is not unique within i.
3. The data are in long form and an unmentioned variable is not constant within i.

▷ Example 5

With some mistakes, reshape will probably convert the data and produce a surprising result. Suppose that we forget to mention that the ue variable varies within id in the following wide data:

```
. use https://www.stata-press.com/data/r18/reshape1
. list
```

	id	sex	inc80	inc81	inc82	ue80	ue81	ue82
1.	1	0	5000	5500	6000	0	1	0
2.	2	1	2000	2200	3300	1	0	0
3.	3	0	3000	2000	1000	0	0	1

```
. reshape long inc, i(id) j(year)
(j = 80 81 82)
Data                                Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                 3   ->   9
Number of variables                    8   ->   7
j variable (3 values)                      ->   year
xij variables:
                      inc80 inc81 inc82    ->   inc
-----------------------------------------------------------------------------
. list, sep(3)
```

	id	year	sex	inc	ue80	ue81	ue82
1.	1	80	0	5000	0	1	0
2.	1	81	0	5500	0	1	0
3.	1	82	0	6000	0	1	0
4.	2	80	1	2000	1	0	0
5.	2	81	1	2200	1	0	0
6.	2	82	1	3300	1	0	0
7.	3	80	0	3000	0	0	1
8.	3	81	0	2000	0	0	1
9.	3	82	0	1000	0	0	1

We did not state that ue varied within i, so the variables ue80, ue81, and ue82 were left as is. reshape did not complain. There is no real problem here because no information has been lost. In fact, this may actually be the result we wanted. Probably, however, we simply forgot to include ue among the X_{ij} variables.

If you obtain an unexpected result, here is how to undo it:

1. If you typed reshape long ... to produce the result, type reshape wide (without arguments) to undo it.
2. If you typed reshape wide ... to produce the result, type reshape long (without arguments) to undo it.

So, we can type

```
. reshape wide
```

to get back to our original, wide-form data and then type the `reshape long` command that we intended:

```
. reshape long inc ue, i(id) j(year)
```

◁

reshape long and reshape wide without arguments

Whenever you type a `reshape long` or `reshape wide` command with arguments, `reshape` remembers it. Thus you might type

```
. reshape long inc ue, i(id) j(year)
```

and work with the data like that. You could then type

```
. reshape wide
```

to convert the data back to the wide form. Then later you could type

```
. reshape long
```

to convert them back to the long form. If you save the data, you can even continue using `reshape wide` and `reshape long` without arguments during a future Stata session.

Be careful. If you create new X_{ij} variables, you must tell `reshape` about them by typing the full `reshape` command, although no real damage will be done if you forget. If you are converting from long to wide form, `reshape` will catch your error and refuse to make the conversion. If you are converting from wide to long, `reshape` will convert the data, but the result will be surprising: remember what happened when we forgot to mention the `ue` variable and ended up with `ue80`, `ue81`, and `ue82` in our long data; see example 5. You can `reshape long` to undo the unwanted change and then try again.

Missing variables

When converting data from wide form to long form, `reshape` does not demand that all the variables exist. Missing variables are treated as variables with missing observations.

▷ Example 6

Let's drop `ue81` from the wide form of the data:

```
. use https://www.stata-press.com/data/r18/reshape1, clear
. drop ue81
. list
```

	id	sex	inc80	inc81	inc82	ue80	ue82
1.	1	0	5000	5500	6000	0	0
2.	2	1	2000	2200	3300	1	0
3.	3	0	3000	2000	1000	0	1

```
. reshape long inc ue, i(id) j(year)
(j = 80 81 82)
(variable ue81 not found)
Data                                Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                 3   ->   9
Number of variables                    7   ->   5
j variable (3 values)                      ->   year
xij variables:
                        inc80 inc81 inc82   ->   inc
                           ue80 ue81 ue82   ->   ue
-----------------------------------------------------------------------------
```

```
. list, sep(3)
```

	id	year	sex	inc	ue
1.	1	80	0	5000	0
2.	1	81	0	5500	.
3.	1	82	0	6000	0
4.	2	80	1	2000	1
5.	2	81	1	2200	.
6.	2	82	1	3300	0
7.	3	80	0	3000	0
8.	3	81	0	2000	.
9.	3	82	0	1000	1

`reshape` placed missing values where `ue81` values were unavailable. If we reshaped these data back to wide form by typing

```
. reshape wide inc ue, i(id) j(year)
```

the `ue81` variable would be created and would contain all missing values.

◁

Advanced issues with basic syntax: i()

The `i()` option can indicate one `i` variable (as our past examples have illustrated) or multiple variables. An example of multiple `i` variables would be hospital ID and patient ID within each hospital.

```
. reshape ... , i(hid pid)
```

Unique pairs of values for `hid` and `pid` in the data define the grouping variable for `reshape`.

Advanced issues with basic syntax: j()

The `j()` option takes a variable name (as our past examples have illustrated) or a variable name and a list of values. When the values are not provided, `reshape` deduces them from the data. Specifying the values with the `j()` option is rarely needed.

`reshape` never makes a mistake when the data are in long form and you type `reshape wide`. The values are easily obtained by tabulating the `j` variable.

reshape can make a mistake when the data are in wide form and you type reshape long if your variables are poorly named. Say that you have the inc80, inc81, and inc82 variables, recording income in each of the indicated years, and you have a variable named inc2, which is not income but indicates when the area was reincorporated. You type

```
. reshape long inc, i(id) j(year)
```

reshape sees the inc2, inc80, inc81, and inc82 variables and decides that there are four groups in which j = 2, 80, 81, and 82.

The easiest way to solve the problem is to rename the inc2 variable to something other than "inc" followed by a number; see [D] **rename**.

You can also keep the name and specify the j values. To perform the reshape, you can type

```
. reshape long inc, i(id) j(year 80-82)
```

or

```
. reshape long inc, i(id) j(year 80 81 82)
```

You can mix the dash notation for value ranges with individual numbers. reshape would understand 80 82-87 89 91-95 as a valid values specification.

At the other extreme, you can omit the j() option altogether with reshape long. If you do, the j variable will be named _j.

Advanced issues with basic syntax: xij

When specifying variable names, you may include @ characters to indicate where the numbers go.

▷ Example 7

Let's reshape the following data from wide to long form:

```
. use https://www.stata-press.com/data/r18/reshape3, clear
. list
```

	id	sex	inc80r	inc81r	inc82r	ue80	ue81	ue82
1.	1	0	5000	5500	6000	0	1	0
2.	2	1	2000	2200	3300	1	0	0
3.	3	0	3000	2000	1000	0	0	1

```
. reshape long inc@r ue, i(id) j(year)
(j = 80 81 82)
Data                                Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                 3   ->   9
Number of variables                    8   ->   5
j variable (3 values)                      ->   year
xij variables:
                    inc80r inc81r inc82r   ->   incr
                         ue80 ue81 ue82    ->   ue
-----------------------------------------------------------------------------
```

```
. list, sep(3)
```

	id	year	sex	incr	ue
1.	1	80	0	5000	0
2.	1	81	0	5500	1
3.	1	82	0	6000	0
4.	2	80	1	2000	1
5.	2	81	1	2200	0
6.	2	82	1	3300	0
7.	3	80	0	3000	0
8.	3	81	0	2000	0
9.	3	82	0	1000	1

At most one `@` character may appear in each name. If no `@` character appears, results are as if the `@` character appeared at the end of the name. So, the equivalent `reshape` command to the one above is

```
. reshape long inc@r ue@, i(id) j(year)
```

`inc@r` specifies variables named `inc#r` in the wide form and `incr` in the long form. The `@` notation may similarly be used for converting data from long to wide format:

```
. reshape wide inc@r ue, i(id) j(year)
```

◁

Advanced issues with basic syntax: String identifiers for j()

The `string` option allows `j` to take on string values.

▷ Example 8

Consider the following wide data on husbands and wives. In these data, `incm` is the income of the man and `incf` is the income of the woman.

```
. use https://www.stata-press.com/data/r18/reshape4, clear
. list
```

	id	kids	incm	incf
1.	1	0	5000	5500
2.	2	1	2000	2200
3.	3	2	3000	2000

These data can be reshaped into separate observations for males and females by typing

```
. reshape long inc, i(id) j(sex) string
(j = f m)
Data                                Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                 3   ->   6
Number of variables                    4   ->   4
j variable (2 values)                      ->   sex
xij variables:
                              incf incm    ->   inc
-----------------------------------------------------------------------------
```

The string option specifies that j take on nonnumeric values. The result is

```
. list, sep(2)
```

	id	sex	kids	inc
1.	1	f	0	5500
2.	1	m	0	5000
3.	2	f	1	2200
4.	2	m	1	2000
5.	3	f	2	2000
6.	3	m	2	3000

sex will be a string variable. Similarly, these data can be converted from long to wide form by typing

```
. reshape wide inc, i(id) j(sex) string
```

◁

Strings are not limited to being single characters or even having the same length. You can specify the location of the string identifier in the variable name by using the @ notation.

▷ Example 9

Suppose that our variables are named id, kids, incmale, and incfem.

```
. use https://www.stata-press.com/data/r18/reshapexp2, clear
. list
```

	id	kids	incmale	incfem
1.	1	0	5000	5500
2.	2	1	2000	2200
3.	3	2	3000	2000

```
. reshape long inc, i(id) j(sex) string
(j = fem male)
Data                               Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                3   ->   6
Number of variables                   4   ->   4
j variable (2 values)                     ->   sex
xij variables:
                          incfem incmale   ->   inc
-----------------------------------------------------------------------------
. list, sep(2)
```

	id	sex	kids	inc
1.	1	fem	0	5500
2.	1	male	0	5000
3.	2	fem	1	2200
4.	2	male	1	2000
5.	3	fem	2	2000
6.	3	male	2	3000

If the wide data had variables named minc and finc, the appropriate reshape command would have been

```
. reshape long @inc, i(id) j(sex) string
```

The resulting variable in the long form would be named inc.

We can also place strings in the middle of the variable names. If the variables were named incMome and incFome, the reshape command would be

```
. reshape long inc@ome, i(id) j(sex) string
```

Be careful with string identifiers because it is easy to be surprised by the result. Say that we have wide data having variables named incm, incf, uem, uef, agem, and agef. To make the data long, we might type

```
. reshape long inc ue age, i(id) j(sex) string
```

Along with these variables, we also have the variable agenda. reshape will decide that the sexes are m, f, and nda. This would not happen without the string option if the variables were named inc0, inc1, ue0, ue1, age0, and age1, even with the agenda variable present in the data.

◁

Advanced issues with basic syntax: Second-level nesting

Sometimes the data may have more than one possible j variable for reshaping. Suppose that your data have both a year variable and a sex variable. One logical observation in the data might be represented in any of the following four forms:

```
. list in 1/4    // The long-long form
```

	hid	sex	year	inc
1.	1	f	90	3200
2.	1	f	91	4700
3.	1	m	90	4500
4.	1	m	91	4600

```
. list in 1/2    // The long-year wide-sex form
```

	hid	year	minc	finc
1.	1	90	4500	3200
2.	1	91	4600	4700

```
. list in 1/2    // The wide-year long-sex form
```

	hid	sex	inc90	inc91
1.	1	f	3200	4700
2.	1	m	4500	4600

```
. list in 1      // The wide-wide form
```

	hid	minc90	minc91	finc90	finc91
1.	1	4500	4600	3200	4700

reshape can convert any of these forms to any other. Converting data from the long–long form to the wide–wide form (or any of the other forms) takes two reshape commands. Here is how we would do it:

From		To		
year	sex	year	sex	Command
long	long	long	wide	reshape wide @inc, i(hid year) j(sex) string
long	wide	long	long	reshape long @inc, i(hid year) j(sex) string
long	long	wide	long	reshape wide inc, i(hid sex) j(year)
wide	long	long	long	reshape long inc, i(hid sex) j(year)
long	wide	wide	wide	reshape wide minc finc, i(hid) j(year)
wide	wide	long	wide	reshape long minc finc, i(hid) j(year)
wide	long	wide	wide	reshape wide @inc90 @inc91, i(hid) j(sex) string
wide	wide	wide	long	reshape long @inc90 @inc91, i(hid) j(sex) string

Description of advanced syntax

The advanced syntax is simply a different way of specifying the reshape command, and it has one seldom-used feature that provides extra control. Rather than typing one reshape command to describe the data and perform the conversion, such as

```
. reshape long inc, i(id) j(year)
```

you type a sequence of reshape commands. The initial commands describe the data, and the last command performs the conversion:

```
. reshape i id
. reshape j year
. reshape xij inc
. reshape long
```

reshape i corresponds to i() in the basic syntax.

reshape j corresponds to j() in the basic syntax.

reshape xij corresponds to the variables specified in the basic syntax. reshape xij also accepts the atwl() option for use when @ characters are specified in the *fvarnames*. atwl stands for at-when-long. When you specify names such as inc@r or ue@, in the long form the names become incr and ue, and the @ character is ignored. atwl() allows you to change @ into whatever you specify. For example, if you specify atwl(X), the long-form names become incXr and ueX.

There is also one more specification, which has no counterpart in the basic syntax:

```
. reshape xi varlist
```

In the basic syntax, Stata assumes that all unspecified variables are constant within i. The advanced syntax works the same way, unless you specify the reshape xi command, which names the constant-within-i variables. If you specify reshape xi, any variables that you do not explicitly specify are dropped from the data during the conversion.

As a practical matter, you should explicitly drop the unwanted variables before conversion. For instance, suppose that the data have variables inc80, inc81, inc82, sex, age, and age2 and that you no longer want the age2 variable. You could specify

```
. reshape xi sex age
```

or

```
. drop age2
```

and leave reshape xi unspecified.

reshape xi does have one minor advantage. It saves reshape the work of determining which variables are unspecified. This saves a relatively small amount of computer time.

Another advanced-syntax feature is reshape query, which is equivalent to typing reshape by itself. reshape query reports which reshape parameters have been defined. reshape i, reshape j, reshape xij, and reshape xi specifications may be given in any order and may be repeated to change or correct what has been specified.

Finally, reshape clear clears the definitions. reshape definitions are stored with the dataset when you save it. reshape clear allows you to erase these definitions.

The basic syntax of reshape is implemented in terms of the advanced syntax, so you can mix basic and advanced syntaxes.

Why favor memory over speed?

The original code for reshape was written in a time when computer memory was not as abundantly available as it is today and Stata could not handle multiple datasets in memory at the same time. This code uses the commands preserve, save, use, append, and merge to reshape the data between forms. Incrementally reshaping the data this way accommodated the memory limitations of the time at the cost of being slow for bigger datasets. This is the method used with favor(memory).

With favor(speed), reshape preallocates a data frame with the target form and fills it with the data from the current frame. This method of data conversion is typically much faster but requires double the memory used for the original data.

If you have enough memory, favor(speed) is preferable. For example, if your datasets require much less than half the physical memory on your machine, then you probably want to use option favor(speed) or put

```
. set reshape_favor speed
```

at the top of your do-files.

In most cases, reshape settings should be considered for specific data. If you type

```
. set reshape_favor speed, permanently
```

and forget about it, you may experience Mata run time failures with reshape for datasets that are larger than half the available memory on your computer.

You can always type

```
. set reshape_favor default
```

to restore the default setting.

Video examples

How to reshape data from long format to wide format

How to reshape data from wide format to long format

Stored results

`reshape` stores the following characteristics with the data (see [P] **char**):

`_dta[ReS_i]`	`i` variable names
`_dta[ReS_j]`	`j` variable name
`_dta[ReS_jv]`	`j` values, if specified
`_dta[ReS_Xij]`	X_{ij} variable names
`_dta[ReS_Xij_n]`	number of X_{ij} variables
`_dta[ReS_Xij_long#]`	name of #th X_{ij} variable in long form
`_dta[ReS_Xij_wide#]`	name of #th X_{ij} variable in wide form
`_dta[ReS_Xi]`	X_i variable names, if specified
`_dta[ReS_atwl]`	`atwl()` value, if specified
`_dta[ReS_str]`	`1` if option `string` specified, `0` otherwise
`_dta[ReS_favor]`	`favor()` value, if specified

Acknowledgment

This version of `reshape` was based in part on the work of Jeroen Weesie (1997) of the Department of Sociology at Utrecht University, The Netherlands.

References

Baum, C. F., and N. J. Cox. 2007. Stata tip 45: Getting those data into shape. *Stata Journal* 7: 268–271.

Huber, C. 2014. How to simulate multilevel/longitudinal data. *The Stata Blog: Not Elsewhere Classified.* http://blog.stata.com/2014/07/18/how-to-simulate-multilevellongitudinal-data/.

Jeanty, P. W. 2010. Using the World Development Indicators database for statistical analysis in Stata. *Stata Journal* 10: 30–45.

Mitchell, M. N. 2020. *Data Management Using Stata: A Practical Handbook*. 2nd ed. College Station, TX: Stata Press.

Simons, K. L. 2016. A sparser, speedier reshape. *Stata Journal* 16: 632–649.

Weesie, J. 1997. dm48: An enhancement of reshape. *Stata Technical Bulletin* 38: 2–4. Reprinted in *Stata Technical Bulletin Reprints*, vol. 7, pp. 40–43. College Station, TX: Stata Press.

Also see

[D] **frunalias** — Change storage type of alias variables

[D] **save** — Save Stata dataset

[D] **stack** — Stack data

[D] **xpose** — Interchange observations and variables

[P] **char** — Characteristics