Title

stata.com

st_view( ) — Make matrix that is a view onto current Stata dataset

Description Syntax Remarks and examples Conformability
Diagnostics Reference Also see

Description

`st_view()` and `st_sview()` create a matrix that is a view onto the current Stata dataset.

Syntax

void `st_view(`*V*`,` *real matrix i*`,` *rowvector j*`)`

void `st_view(`*V*`,` *real matrix i*`,` *rowvector j*`,` *scalar selectvar*`)`

void `st_sview(`*V*`,` *real matrix i*`,` *rowvector j*`)`

void `st_sview(`*V*`,` *real matrix i*`,` *rowvector j*`,` *scalar selectvar*`)`

where

1. The type of *V* does not matter; it is replaced.
2. *i* may be specified in the same way as with `st_data()`.
3. *j* may be specified in the same way as with `st_data()`. Factor variables and time-series–operated variables may be specified.
4. *selectvar* may be specified in the same way as with `st_data()`.

See [M-5] **st_data( )**.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Overview
Advantages and disadvantages of views
When not to use views
Cautions when using views 1: Conserving memory
Cautions when using views 2: Assignment
Cautions when using views 3: View connections are ephemeral
Efficiency

Overview

st_view() serves the same purpose as st_data()—and st_sview() serves the same purpose as st_sdata()—except that, rather than returning a matrix that is a copy of the underlying values, st_view() and st_sview() create a matrix that is a view onto the Stata dataset itself.

To understand the distinction, consider

```
X = st_data(., "mpg displ weight")
```

and

```
st_view(X, ., "mpg displ weight")
```

Both commands fill in matrix X with the same data. However, were you to code

```
X[2,1] = 123
```

after the st_data() setup, you would change the value in the matrix X, but the Stata dataset would remain unchanged. After the st_view() setup, changing the value in the matrix would cause the value of mpg in the second observation to change to 123.

Advantages and disadvantages of views

Views make it easy to change the dataset, and that can be an advantage or a disadvantage, depending on your goals.

Putting that aside, views are in general better than copies because 1) they take less time to set up and 2) they consume less memory. The memory savings can be considerable. Consider a 100,000-observation dataset on 30 variables. Coding

```
X = st_data(., .)
```

creates a new matrix that is 24 MB in size. Meanwhile, the total storage requirement for

```
st_view(X, ., .)
```

is roughly 128 bytes!

There is a cost; when you use the matrix X, it takes longer to access the individual elements. You would have to do a lot of calculation with X, however, before that sum of the longer access times would equal the initial savings in setup time, and even then, the longer access time is probably worth the savings in memory.

When not to use views

Do not use views as a substitute for scalars. If you are going to loop through the data an observation at a time, and if every usage you will make of *X* is in scalar calculations, use _st_data(). There is nothing faster for that problem.

Putting aside that extreme, views become more efficient relative to copies the larger they are; that is, it is more efficient to use st_data() for small amounts of data, especially if you are going to make computationally intensive calculations with it.

Cautions when using views 1: Conserving memory

If you are using views, it is probably because you are concerned about memory, and if you are, you want to be careful to avoid making copies of views. Copies of views are not views; they are copies. For instance,

```
st_view(V, ., .)
Y = V
```

That innocuous looking `Y = V` just made a copy of the entire dataset, meaning that if the dataset had 100,000 observations on 30 variables, `Y` now consumes 24 MB. Coding `Y = V` may be desirable in certain circumstances, but in general, it is better to set up another view.

Similarly, watch out for subscripts. Consider the following code fragment

```
st_view(V, ., .)
for (i=1; i<=cols(V); i++) {
     sum = colsum(V[,i])
     ...
}
```

The problem in the above code is the `V[,i]`. That creates a new column vector containing the values from the `i`th column of `V`. Given 100,000 observations, that new column vector needs 800k of memory. Better to code would be

```
for (i=1; i<=cols(V); i++) {
     st_view(v, ., i)
     sum = colsum(v)
     ...
}
```

If you need `V` and `v`, that is okay. You can have many views of the data setup simultaneously.

Similarly, be careful using views with operators. `X'X` makes a copy of `X` in the process of creating the transpose. Use functions such as `cross()` (see [M-5] **cross( )**) that are designed to minimize the use of memory.

Do not be overly concerned about this issue. Making a copy of a column of a view amounts to the same thing as introducing a temporary variable in a Stata program—something that is done all the time.

Cautions when using views 2: Assignment

The ability to assign to a view and so change the underlying data can be either convenient or dangerous, depending on your goals. When making such assignments, there are two things you need be aware of.

The first is more of a Stata issue than it is a Mata issue. Assignment does not cause promotion. Coding

```
V[1,2] = 4059.125
```

might store 4059.125 in the first observation of the second variable of the view. Or, if that second variable is an `int`, what will be stored is 4059, or if it is a `byte`, what will be stored is missing.

The second caution is a Mata issue. To reassign all the values of the view, code

```
V[.,.] = matrix_expression
```

Do not code

```
V = matrix_expression
```

The second expression does not assign to the underlying dataset, it redefines `V` to be a regular matrix.

Mata does not allow the use of views as the destination of assignment when the view contains factor variables or time-series–operated variables such as `i.rep78` or `l.gnp`.

Cautions when using views 3: View connections are ephemeral

For faster data access, an `st_view()` connection accesses data using variable indices, not variable names. However, variable indices can change when variables are created or removed. If a variable is created or removed while your code is using a view connection, there is a chance the view will switch to another variable.

Efficiency

Whenever possible, specify argument *i* of `st_view(`*V*`,` *i*`,` *j*`)` and `st_sview(`*V*`,` *i*`,` *j*`)` as `.` (missing value) or as a row vector range (for example, (i_1,i_2)) rather than as a column vector list.

Specify argument *j* as a real row vector rather than as a string whenever `st_view()` and `st_sview()` are used inside loops with the same variables (and the view does not contain factor variables nor time-series–operated variables). This prevents Mata from having to look up the same names over and over again.

Conformability

`st_view(`*V*`,` *i*`,` *j*`)`, `st_sview(`*V*`,` *i*`,` *j*`)`:

input:

i:	$n \times 1$	or	$n_2 \times 2$
j:	$1 \times k$	or	1×2 containing k elements when expanded

output:

V:	$n \times k$

`st_view(`*V*`,` *i*`,` *j*`,` *selectvar*`)`, `st_sview(`*V*`,` *i*`,` *j*`,` *selectvar*`)`:

input:

i:	$n \times 1$	or	$n_2 \times 2$
j:	$1 \times k$	or	1×2 containing k elements when expanded
selectvar:	1×1		

output:

V:	$(n - e) \times k$, where e is number of observations excluded by *selectvar*

Diagnostics

`st_view(`*i*`, `*j*`[, `*selectvar*`])` and `st_sview(`*i*`, `*j*`[, `*selectvar*`])` abort with error if any element of *i* is outside the range of observations or if a variable name or index recorded in *j* is not found. Variable-name abbreviations are allowed. If you do not want this and no factor variables nor time-series–operated variables are specified, use `st_varindex()` (see [M-5] **st_varindex( )**) to translate variable names into variable indices.

`st_view()` and `st_sview()` abort with error if any element of *i* is out of range as described under the heading *Details of observation subscripting using st_data( ) and st_sdata( )* in [M-5] **st_data( )**.

Some functions do not allow views as arguments. If `example(`*X*`)` does not allow views, you can still use it by coding

```
... example(X=V) ...
```

because that will make a copy of view `V` in `X`. Most functions that do not allow views mention that in their *Diagnostics* section, but some do not because it was unexpected that anyone would want to use a view in that case. If a function does not allow a view, you will see in the traceback log:

```
: myfunction(...)
       example():  3103  view found where array required
         mysub():     -  function returned error
    myfunction():     -  function returned error
          <istmt>:    -  function returned error
r(3103);
```

The above means that function `example()` does not allow views.

Reference

Gould, W. W. 2005. Mata Matters: Using views onto the data. *Stata Journal* 5: 567–573.

Also see

[M-5] **select( )** — Select rows, columns, or indices

[M-5] **st_data( )** — Load copy of current Stata dataset

[M-5] **st_subview( )** — Make view from view

[M-5] **st_viewvars( )** — Variables and observations of view

[M-4] **Stata** — Stata interface functions

[D] **putmata** — Put Stata variables into Mata and vice versa