Title

stata.com

st_is — Survival analysis subroutines for programmers

Description Syntax Remarks and examples Also see

Description

These commands are provided for programmers wishing to write new st commands.

`st_is` verifies that the data in memory are survival-time (st) data. If not, it issues the error message "data not st", r(119).

st is currently "release 2", meaning that this is the second design of the system. Programs written for the previous release continue to work. (The previous release of st corresponds to Stata 5.)

Modern programs code `st_is 2 full` or `st_is 2 analysis`. `st_is 2` verifies that the dataset in memory is in release 2 format; if it is in the earlier format, it is converted to release 2 format. (Older programs simply code `st_is`. This verifies that no new features are `stset` about the data that would cause the old program to break.)

The `full` and `analysis` parts indicate whether the dataset may include past, future, or past and future data. Code `st_is 2 full` if the command is suitable for running on the analysis sample and the past and future data (many data management commands fall into this category). Code `st_is 2 analysis` if the command is suitable for use only with the analysis sample (most statistical commands fall into this category). See [ST] **stset** for the definitions of past and future.

`st_show` displays the summary of the survival-time variables or does nothing, depending on what you specify when `stset`ting the data. `noshow` requests that `st_show` display nothing.

`st_ct` is a low-level utility that provides risk-group summaries from survival-time data.

Syntax

Verify that data in memory are survival-time data

`st_is 2 {full | analysis}`

Display or do not display summary of survival-time variables

`st_show [noshow]`

Risk-group summaries

`st_ct "[`*byvars*`]" ->` *newtvar newpopvar newfailvar* `[`*newcensvar* `[`*newentvar*`]]`

You must have `stset` your data before using `st_is`, `st_show`, and `st_ct`; see [ST] **stset**.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Definitions of characteristics and st variables

From a programmer's perspective, st is a set of conventions that specify where certain pieces of information are stored and how that information should be interpreted, together with a few subroutines that make it easier to follow the conventions.

At the lowest level, st is nothing more than a set of Stata characteristics that programmers may access:

`char _dta[_dta]`	`st` (marks that the data are st)
`char _dta[st_ver]`	`2` (version number)
`char _dta[st_id]`	*varname* or nothing; `id()` variable
`char _dta[st_bt0]`	*varname* or nothing; `t0()` variable
`char _dta[st_bt]`	*varname*; *t* variable from `stset t, ...`
`char _dta[st_bd]`	*varname* or nothing; `failure()` variable
`char _dta[st_ev]`	list of numbers or nothing; *numlist* from `failure(`*varname*`[==`*numlist*`])`
`char _dta[st_enter]`	contents of `enter()` or nothing; *numlist* expanded
`char _dta[st_exit]`	contents of `exit()` or nothing; *numlist* expanded
`char _dta[st_orig]`	contents of `origin()` or nothing; *numlist* expanded
`char _dta[st_bs]`	# or `1`; `scale()` value
`char _dta[st_o]`	`_origin` or #
`char _dta[st_s]`	`_scale` or #
`char _dta[st_ifexp]`	*exp* or nothing; from `stset ... if exp ...`
`char _dta[st_if]`	*exp* or nothing; contents of `if()`
`char _dta[st_ever]`	*exp* or nothing; contents of `ever()`
`char _dta[st_never]`	*exp* or nothing; contents of `never()`
`char _dta[st_after]`	*exp* or nothing; contents of `after()`
`char _dta[st_befor]`	*exp* or nothing; contents of `before()`
`char _dta[st_wt]`	weight type or nothing; user-specified weight
`char _dta[st_wv]`	*varname* or nothing; user-specified weighting variable
`char _dta[st_w]`	`[`*weighttype*`=`*weightvar*`]` or nothing
`char _dta[st_show]`	`noshow` or nothing
`char _dta[st_t]`	`_t` (for compatibility with release 1)
`char _dta[st_t0]`	`_t0` (for compatibility with release 1)
`char _dta[st_d]`	`_d` (for compatibility with release 1)
`char _dta[st_n0]`	# or nothing; number of st notes
`char _dta[st_n1]`	text of first note or nothing
`char _dta[st_n2]`	text of second note or nothing
`char _dta[st_set]`	text or nothing. If filled in, `streset` (see [ST] **stset**) will refuse to execute and present this text as the reason

All st datasets also have the following four variables:

_t0	Time of entry (in t units) into risk pool
_t	Time of exit (in t units) from risk pool
_d	1 if failure, 0 if censoring
_st	1 if observation is to be used and 0 otherwise

Thus, in a program, you might code

```
display "the failure/censoring base time variable is _t"
display "and its mean in the uncensored subsample is"
summarize _t if _d
```

No matter how simple or complicated the data, these four variables exist and are filled in. For instance, in simple data, _t0 might contain 0 for every observation, and _d might always contain 1.

Some st datasets also contain the variables

_origin	Evaluated value of origin()
_scale	Evaluated value of scale()

The _dta[st_o] characteristic contains either the name _origin or a number, often 0. It contains a number when the origin does not vary across observations. _dta[st_s] works the same way with the scale() value. Thus the origin and scale are _dta[st_o] and _dta[st_s]. In fact, these characteristics are seldom used because variables _t and _t0 are already adjusted.

Some st datasets have an id() variable that clusters together records on the same subject. The name of the variable varies, and the name can be obtained from the _dta[st_id] characteristic. If there is no id() variable, the characteristic contains nothing.

Outline of an st command

If you are writing a new st command, place st_is near the top of your code to ensure that your command does not execute on inappropriate data. Also place st_show following the parsing of your command's syntax to display the key st variables. The minimal outline for an st command is

```
program st name
        version 18.0
        st_is 2 ...
        ... syntax command ...
        ... determined there are no syntax errors ...
        st_show
        ... guts of program ...
end
```

st_is 2 appears even before the input is parsed. This is to avoid irritating users when they type a command, get a syntax error, work hard to eliminate the error, and then learn that "data not st".

A fuller outline for an st command, particularly one that performs analysis on the data, is

```
program st name
        version 18.0
        st_is 2 ...
        syntax ... [, ... noSHow ... ]
        st_show 'show'
        marksample touse
        quietly replace 'touse' = 0 if _st==0
        ... guts of program ...
end
```

All calculations and actions are to be restricted, at the least, to observations for which `_st` $\neq 0$. Observations with `_st` $= 0$ are to be ignored.

Using the st_ct utility

`st_ct` converts the data in memory to observations containing summaries of risk groups. Consider the code

```
st_is 2 analysis
preserve
st_ct "" -> t pop die
```

Typing this would change the data in memory to contain something akin to count-time data. The transformed data would have observations containing

`t`	time
`pop`	population at risk at time `t`
`die`	number who fail at time `t`

There would be one record per time `t`, and the data would be sorted by `t`. The original data are discarded, which is why you should code `preserve`; see [P] **preserve**.

The above three lines of code could be used as the basis for calculating the Kaplan–Meier product-limit survivor-function estimate. The rest of the code is

```
keep if die
generate double hazard = die/pop
generate double km     =  1-hazard  if _n==1
replace    km     = (1-hazard)*km[_n-1] if _n>1
```

`st_ct` can be used to obtain risk groups separately for subgroups of the population. The code

```
st_is 2 analysis
preserve
st_ct "race sex" -> t pop die
```

would change the data in memory to contain

`race`	
`sex`	
`t`	time
`pop`	population at risk at time `t`
`die`	number who fail at time `t`

There would be one observation for each `race`–`sex`–`t` combination, and the data would be sorted by `race sex t`.

With this dataset, you could calculate the Kaplan–Meier product-limit survivor-function estimate for each `race`–`sex` group by coding

```
keep if die
generate double hazard = die/pop
by race sex: generate double km =  1-hazard  if _n==1
by race sex: replace         km = (1-hazard)*km[_n-1] if _n>1
```

`st_ct` is a convenient subroutine. The above code fragment works regardless of the complexity of the underlying survival-time data. It does not matter whether there is one record per subject, no censoring, and one failure per subject, or multiple records per subject, gaps, and recurring failures for the same subject. `st_ct` forms risk groups that summarize the events recorded by the data.

`st_ct` can provide the number of censored records and the number who enter the risk group. The code

```
st_ct "" -> t pop die cens ent
```

creates records containing

`t`	time
`pop`	population at risk at time `t`
`die`	number who fail at time `t`
`cens`	number who are censored at `t` (after the failures)
`ent`	number who enter at `t` (after the censorings)

As before,

```
st_ct "race sex" -> t pop die cens ent
```

would create a similar dataset with records for each `race-sex` group.

Comparison of st_ct with sttoct

`sttoct`—see [ST] **sttoct**—is related to `st_ct`, and in fact, `sttoct` is implemented in terms of `st_ct`. The differences between them are that

- `sttoct` creates ct data, meaning that the dataset is marked as being ct. `st_ct` merely creates a useful dataset; it does not `ctset` the data.
- `st_ct` creates a total population at-risk variable—which is useful in programming—but `sttoct` creates no such variable.
- `sttoct` eliminates thrashings—censorings and reentries of the same subject as covariates change—if there are no gaps, strata shifting, etc. `st_ct` does not do this. Thus, at a particular time, `sttoct` might show that there are two lost to censoring and none entered, whereas `st_ct` might show 12 censorings and 10 entries. This makes no difference in calculating the number at risk and the number who fail, which are the major ingredients in survival calculations.
- `st_ct` is faster.

Verifying data

As long as you code `st_is` at the top of your program, you need not verify the consistency of the data. That is, you need not verify that subjects do not fail before they enter, etc.

The dataset is verified when you `stset` it. If you make a substantive change to the data, you must rerun `stset` (which can be done by typing `stset` or `streset` without arguments) to reverify that all is well.

Converting data

If you write a program that converts the data from one form of st data to another, or from st data to something else, be sure to issue the appropriate `stset` command. For instance, a command we have written, `stbase`, converts the data from st to a simple cross-section in one instance. In our program, we coded `stset, clear` so that all other st commands would know that these are no longer st data and that making st calculations on them would be inappropriate.

Even if we had forgotten, other st programs would have found many of the key st variables missing and would have ended with a "[such-and-such] not found" error.

Also see

[ST] **stset** — Declare data to be survival-time data

[ST] **sttoct** — Convert survival-time data to count-time data

[ST] **Survival analysis** — Introduction to survival analysis commands