Title

stata.com

stsplit — Split and join time-span records

Syntax

Split at designated times

```
stsplit newvar [if], {at(numlist) | every(#)} [stsplitDT_options]
```

Split at failure times

```
stsplit [if], at(failures) [stsplitFT_options]
```

Join episodes

```
stjoin [, censored(numlist)]
```

stsplitDT_options	Description
Main	
* `at(`*numlist*`)`	split records at specified analysis times
* `every(#)`	split records when analysis time is a multiple of #
`after(`*spec*`)`	use time since *spec* for `at()` or `every()` rather than time since onset of risk; see *Options*
`trim`	exclude observations outside of range
`nopreserve`	do not save original data; programmer's option

*Either `at(`*numlist*`)` or `every(#)` is required with `stsplit` at designated times.

stsplitFT_options	Description
Main	
* `at(failures)`	split at observed failure times
`strata(`*varlist*`)`	restrict splitting to failures within stratum defined by *varlist*
`riskset(`*newvar*`)`	create a risk-set ID variable named *newvar*
`nopreserve`	do not save original data; programmer's option

*`at(failures)` is required with `stsplit` at failure times.

You must `stset` your dataset using the `id()` option before using `stsplit` and `stjoin`; see [ST] **stset**. `nopreserve` does not appear in the dialog box.

Menu

stsplit

Statistics > Survival analysis > Setup and utilities > Split time-span records

stjoin

Statistics > Survival analysis > Setup and utilities > Join time-span records

Description

`stsplit` with the `at(`*numlist*`)` or `every(#)` option splits episodes into two or more episodes at the implied time points since being at risk or after a time point specified by `after()`. Each resulting record contains the follow-up on one subject through one time band. Expansion on multiple time scales may be obtained by repeatedly using `stsplit`. *newvar* specifies the name of the variable to be created containing the observation's category. The new variable records the interval to which each new observation belongs and is bottom coded.

`stsplit, at(failures)` performs episode splitting at the failure times (per stratum).

`stjoin` performs the reverse operation, namely, joining episodes back together when such can be done without losing information.

Options for stsplit

Main

`at(`*numlist*`)` or `every(#)` is required in syntax one; `at(failures)` is required for syntax two. These options specify the analysis times at which the records are to be split.

`at(5(5)20)` splits records at $t = 5$, $t = 10$, $t = 15$, and $t = 20$.

If `at(`$[\ldots]$ `max)` is specified, `max` is replaced by a suitably large value. For instance, to split records every five analysis-time units from time zero to the largest follow-up time in our data, we could find out what the largest time value is by typing `summarize _t` and then explicitly typing it into the `at()` option, or we could just specify `at(0(5)max)`.

`every(#)` is a shorthand for `at(#(#)max)`; that is, episodes are split at each positive multiple of #.

`after(`*spec*`)` specifies the reference time for `at()` or `every()`. Syntax one can be thought of as corresponding to `after(`*time of onset of risk*`)`, although you cannot really type this. You could type, however, `after(time=birthdate)` or `after(time=marrydate)`.

spec has syntax

$$\{\texttt{time} \mid \texttt{t} \mid \texttt{\_t}\} = \{\textit{exp} \mid \texttt{min(}\textit{exp}\texttt{)} \mid \texttt{asis(}\textit{exp}\texttt{)}\}$$

where

`time` specifies that the expression be evaluated in the same time units as *timevar* in `stset timevar, ...`. This is the default.

`t` and `_t` specify that the expression be evaluated in units of analysis time. `t` and `_t` are synonyms; it makes no difference whether you specify one or the other.

exp specifies the reference time. For multiepisode data, *exp* should be constant within subject ID.

`min(`*exp*`)` specifies that for multiepisode data, the minimum of *exp* be taken within ID.

asis(*exp*) specifies that for multiepisode data, *exp* be allowed to vary within ID.

trim specifies that observations with values less than the minimum or greater than the maximum value listed in at() be excluded from subsequent analysis. Such observations are not dropped from the data; trim merely sets their value of variable _st to 0 so that they will not be used, yet they can still be retrieved the next time the dataset is stset.

strata(*varlist*) specifies up to five strata variables. Observations with equal values of the variables are assumed to be in the same stratum. strata() restricts episode splitting to failures that occur within the stratum, and memory requirements are reduced when strata are specified.

riskset(*newvar*) specifies the name for a new variable recording the unique risk set in which an episode occurs, and missing otherwise.

The following option is available with stsplit but is not shown in the dialog box:

nopreserve is intended for use by programmers. It speeds the transformation by not saving the original data, which can be restored should things go wrong or if you press *Break*. Programmers often specify this option when they have already preserved the original data. nopreserve does not affect the transformation.

Option for stjoin

censored(*numlist*) specifies values of the failure variable, *failvar*, from stset ..., failure(*failvar*=...) that indicate "no event" (censoring).

If you are using stjoin to rejoin records after stsplit, you do not need to specify censored(). Just do not forget to drop the variable created by stsplit before typing stjoin. See example 4 below.

Neither do you need to specify censored() if, when you stset your dataset, you specified failure(*failvar*) and not failure(*failvar*=...). Then stjoin knows that $failvar = 0$ and $failvar = .$ (missing) correspond to no event. Two records can be joined if they are contiguous and record the same data and the first record has $failvar = 0$ or $failvar = .$, meaning no event at that time.

You may need to specify censored(), and you probably do if, when you stset the dataset, you specified failure(*failvar*=...). If stjoin is to join records, it needs to know what events do not count and can be discarded. If the only such event is $failvar = .$, you do not need to specify censored().

Remarks and examples

stata.com

Remarks are presented under the following headings:

What stsplit does and why

stsplit splits records into two or more records on the basis of analysis time or on a variable that depends on analysis time, such as age. stsplit takes data like

```
id      _t0      _t     x1     x2   _d
 1        0      18     12     11    1
```

and produces

```
id     _t0     _t    x1    x2   _d     tcat
 1       0      5    12    11    0        0
 1       5     10    12    11    0        5
 1      10     18    12    11    1       10
```

or

```
id     _t0     _t    x1    x2   _d   agecat
 1       0      7    12    11    0       30
 1       7     17    12    11    0       40
 1      17     18    12    11    1       50
```

The above alternatives record the same underlying data: subject 1 had `x1` $= 12$ and `x2` $= 11$ during $0 < t \leq 18$, and at $t = 18$, the subject failed.

The difference between the two alternatives is that the first breaks out the analysis times 0–5, 5–10, and 10–20 (although subject 1 failed before $t = 20$). The second breaks out age 30–40, 40–50, and 50–60. You cannot tell from what is presented above, but at $t = 0$, subject 1 was 33 years old.

In our example, that the subject started with one record is not important. The original data on the subject might have been

```
id     _t0     _t    x1    x2   _d
 1       0     14    12    11    0
 1      14     18    12     9    1
```

and then we would have obtained

```
id     _t0     _t    x1    x2   _d     tcat
 1       0      5    12    11    0        0
 1       5     10    12    11    0        5
 1      10     14    12    11    0       10
 1      14     18    12     9    1       10
```

or

```
id     _t0     _t    x1    x2   _d   agecat
 1       0      7    12    11    0       30
 1       7     14    12    11    0       40
 1      14     17    12     9    0       40
 1      17     18    12     9    1       50
```

Also we could just as easily have produced records with analysis time or age recorded in single-year categories. That is, we could start with

```
id     _t0     _t    x1    x2   _d
 1       0     14    12    11    0
 1      14     18    12     9    1
```

and produce

```
id     _t0     _t    x1    x2   _d     tcat
 1       0      1    12    11    0        0
 1       1      2    12    11    0        1
 1       2      3    12    11    0        2
 ...
```

or

```
id     _t0     _t    x1    x2   _d   agecat
 1       0      1    12    11    0       30
 1       1      2    12    11    0       31
 1       2      3    12    11    0       32
 ...
```

Moreover, we can even do this splitting on more than one variable. Let's go back and start with

```
id      _t0     _t     x1     x2    _d
 1        0     18     12     11     1
```

Let's split it into the analysis-time intervals 0–5, 5–10, and 10–20, *and* let's split it into 10-year age intervals 30–40, 40–50, and 50–60. The result would be

```
id      _t0     _t     x1     x2    _d      tcat     agecat
 1        0      5     12     11     0         0         30
 1        5      7     12     11     0         5         30
 1        7     10     12     11     0         5         40
 1       10     17     12     11     0        10         40
 1       17     18     12     11     1        10         50
```

Why would we want to do any of this?

We might want to split on a time-dependent variable, such as age, if we want to estimate a Cox proportional hazards model and include current age among the regressors (although we could instead use `stcox`'s `tvc()` option) or if we want to make tables by age groups (see [ST] **strate**).

Using stsplit to split at designated times

`stsplit`'s syntax to split at designated times is, ignoring other options,

`stsplit` *newvar* [*if*] `, at(`*numlist*`)`
`stsplit` *newvar* [*if*] `, at(`*numlist*`) after(`*spec*`)`

`at()` specifies the analysis times at which records are to be split. Typing `at(5 10 15)` splits records at the indicated analysis times and separates records into the four intervals 0–5, 5–10, 10–15, and 15+.

In the first syntax, the splitting is done on analysis time, t. In the second syntax, the splitting is done on 5, 10, and 15 analysis-time units after the time given by `after(`*spec*`)`.

In either case, `stsplit` also creates *newvar* containing the interval to which each observation belongs. Here *newvar* would contain 0, 5, 10, and 15; it would contain 0 if the observation occurred in the interval 0–5, 5 if the observation occurred in the interval 5–10, and so on. To be precise,

Category	Precise meaning	*newvar* value
0–5	$(-\infty, 5]$	0
5–10	$(5, 10]$	5
10–15	$(10, 15]$	10
15+	$(15, \infty)$	15

If any of the `at()` numbers are negative (which would be allowed only by specifying the `after()` option and would be unusual), the first category is labeled one less than the minimum value specified by `at()`.

Consider the data

```
id      yr0     yr1  yrborn     x1  event
 1     1990    1995    1960      5     52
 2     1993    1997    1964      3     47
```

In these data, subjects became at risk in yr0. The failure event of interest is event = 47, so we stset our dataset by typing

```
. stset yr1, id(id) origin(time yr0) failure(event==47)
  (output omitted)
```

and that results in

id	_t0	_t	yr0	yr1	yrborn	x1	event	_d
1	0	5	1990	1995	1960	5	52	0
2	0	4	1993	1997	1964	3	47	1

In the jargon of st, variables _t0 and _t record the span of each record in analysis-time (t) units. Variables yr0 and yr1 also record the time span, but in time units. Variable _d records 1 for failure and 0 otherwise.

Typing stsplit cat, at(2 4 6 8) would split the records on the basis of analysis time:

```
. stsplit cat, at(2 4 6 8)
(3 observations (episodes) created)
. order id _t0 _t yr0 yr1 yrborn x1 event _d cat
. list id-cat
```

	id	_t0	_t	yr0	yr1	yrborn	x1	event	_d	cat
1.	1	0	2	1990	1992	1960	5	.	0	0
2.	1	2	4	1990	1994	1960	5	.	0	2
3.	1	4	5	1990	1995	1960	5	52	0	4
4.	2	0	2	1993	1995	1964	3	.	0	0
5.	2	2	4	1993	1997	1964	3	47	1	2

The first record, which represented the analysis-time span $(0, 5]$, was split into three records: $(0, 2]$, $(2, 4]$, and $(4, 5]$. The yrborn and x1 values from the single record were duplicated in $(0, 2]$, $(2, 4]$, and $(4, 5]$. The original event variable was changed to missing at $t = 2$ and $t = 4$ because we do not know the value of event; all we know is that event is 52 at $t = 5$. The _d variable was correspondingly set to 0 for $t = 2$ and $t = 4$ because we do know, at least, that the subject did not fail.

stsplit also keeps your original time variables up to date in case you want to streset or re-stset your dataset. yr1 was updated, too.

Now let's go back to our original dataset after we stset it but before we split it,

id	_t0	_t	yr0	yr1	yrborn	x1	event	_d
1	0	5	1990	1995	1960	5	52	0
2	0	4	1993	1997	1964	3	47	1

and consider splitting on age. In 1990, subject 1 is age 1990 − yrborn = 1990 − 1960 = 30, and subject 2 is 29. If we type

```
. stsplit acat, at(30 32 34) after(time=yrborn)
```

we will split the data according to

```
     age <= 30  (called acat=0)
30 < age <= 32  (called acat=30)
32 < age <= 34  (called acat=32)
34 < age        (called acat=34)
```

The result will be

id	_t0	_t	yr0	yr1	yrborn	x1	event	_d	acat
1	0	2	1990	1992	1960	5	.	0	30
1	2	4	1990	1994	1960	5	.	0	32
1	4	5	1990	1995	1960	5	52	0	34
2	0	1	1993	1994	1964	3	.	0	0
2	1	3	1993	1996	1964	3	.	0	30
2	3	4	1993	1997	1964	3	47	1	32

The original record on subject 1 corresponding to $(0, 5]$ was split into $(0, 2]$, $(2, 4]$, and $(4, 5]$ because those are the t values at which age becomes 32 and 34.

You can `stsplit` the data more than once. Now having these data, if we typed

```
. stsplit cat, at(2 4 6 8)
```

the result would be

id	_t0	_t	yr0	yr1	yrborn	x1	event	_d	acat	cat
1	0	2	1990	1992	1960	5	.	0	30	0
1	2	4	1990	1994	1960	5	.	0	32	2
1	4	5	1990	1995	1960	5	52	0	34	4
2	0	1	1993	1994	1964	3	.	0	0	0
2	1	2	1993	1995	1964	3	.	0	30	0
2	2	3	1993	1996	1964	3	.	0	30	2
2	3	4	1993	1997	1964	3	47	1	32	2

Whether we typed

```
. stsplit acat, at(30 32 34) after(time=yrborn)
. stsplit cat, at(2 4 6 8)
```

or

```
. stsplit cat, at(2 4 6 8)
. stsplit acat, at(30 32 34) after(time=yrborn)
```

would make no difference.

Time versus analysis time

Be careful using the `after()` option if, when you `stset` your dataset, you specified `stset`'s `scale()` option. We say be careful, but actually we mean be appreciative, because `stsplit` will do just what you would expect if you did not think too hard.

When you split a record on a time-dependent variable, `at()` is still specified in analysis-time units, meaning units of time/`scale()`.

For instance, if your original data recorded time as Stata dates, that is, number of days since 1960,

id	date0	date1	birthdate	x1	event
1	14apr1993	27mar1995	12jul1959	5	52
...					

and you previously `stset` your dataset by typing

```
. stset date1, id(id) origin(time date0) scale(365.25) ...
```

and you now wanted to split on the age implied by `birthdate`, you would specify the split points in *years* since birth:

```
. stsplit agecat, at(20(5)60) after(time=birth)
```

`at(20(5)60)` means to split the records at the ages, measured in years, of 20, 25, . . . , 60.

When you stset your dataset, you basically told st how you recorded times (you recorded them as dates) and how to map such times (dates) into analysis time. That was implied by what you typed, and all of st remembers that. Typing

```
. stsplit agecat, at(20(5)60) after(time=birth)
```

tells stsplit to split the data on 20, 25, ..., 60 analysis-time units after birthdate for each subject.

Splitting data on recorded ages

Recorded ages can sometimes be tricky. Consider the data

```
 id     yr0     yr1   age    x1  event
  1    1980    1996    30     5     52
...
```

When was age = 30 recorded—1980 or 1996? Put aside that question because things are about to get worse. Say that you stset this dataset so that yr0 is the origin(),

```
 id   _t0   _t     yr0     yr1   age    x1  event
  1     0   16    1980    1996    30     5     52
...
```

and then split on analysis time by typing stsplit cat, at(5(5)20). The result would be

```
 id   _t0   _t     yr0     yr1   age    x1  event
  1     0    5    1980    1985    30     5      .
  1     5   10    1980    1990    30     5      .
  1    10   15    1980    1995    30     5      .
  1    15   16    1980    1996    30     5     52
```

Regardless of the answer to the question on when age was measured, age is most certainly not 30 in the newly created records, although you might argue that age at baseline was 30 and that is what you wanted, anyway.

The only truly safe way to deal with ages is to convert them back to birthdates at the outset. Here we would, early on, type

```
. gen bdate = yr1 - age                    (if age was measured at yr1)
```

or

```
. gen bdate = yr0 - age                    (if age was measured at yr0)
```

In fact, stsplit tries to protect you from making age errors. Suppose that you did not do as we just recommended. Say that age was measured at yr1, and you typed, knowing that stsplit wants a date,

```
. stsplit acat, at(20(5)50) after(time= yr1-age)
```

on these already stsplit data. stsplit will issue the error message "after() should be constant within id". To use the earliest date, you need to type

```
. stsplit acat, at(20(5)50) after(time= min(yr1-age))
```

Nevertheless, be aware that when you stsplit data, if you have recorded ages in your data, and if the records were not already split to control for the range of those ages, then age values, just like all the other variables, are carried forward and no longer reflect the age of the newly created record.

Example 1: Splitting on age

Consider the data from a heart disease and diet survey. The data arose from a study described more fully in Morris, Marr, and Clayton (1977) and analyzed in Clayton and Hills (1993). (Their results differ slightly from ours because the dataset has been updated.)

```
. use http://www.stata-press.com/data/r13/diet
(Diet data with dates)

. describe

Contains data from http://www.stata-press.com/data/r13/diet.dta
  obs:           337                          Diet data with dates
 vars:            11                          1 May 2013 19:01
 size:        16,513
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
id              float   %9.0g                 Subject identity number
fail            int     %8.0g                 Outcome (CHD = 1 3 13)
job             int     %8.0g                 Occupation
month           byte    %8.0g                 month of survey
energy          float   %9.0g                 Total energy (1000kcals/day)
height          float   %9.0g                 Height (cm)
weight          float   %9.0g                 Weight (kg)
hienergy        float   %9.0g                 Indicator for high energy
doe             double  %td                   Date of entry
dox             double  %td                   Date of exit
dob             double  %td                   Date of birth
-------------------------------------------------------------------------------
Sorted by:  id
```

In this dataset, the outcome variable, `fail`, has been coded as 0, 1, 3, 5, 12, 13, 14, and 15. Codes 1, 3, and 13 indicated coronary heart disease (CHD), other nonzero values code other events such as cancer, and 0 is used to mean "no event" at the end of the study.

The variable `hienergy` is coded 1 if the total energy consumption is more than 2.75 Mcal and 0 otherwise.

We would like to expand the data, using age as the time scale with 10-year age bands. We do this by first `stset`ting the dataset, specifying the date of birth as the origin.

```
. stset dox, failure(fail) origin(time dob) enter(time doe) scale(365.25) id(id)

                id:  id
     failure event:  fail != 0 & fail < .
obs. time interval:  (dox[_n-1], dox]
 enter on or after:  time doe
 exit on or before:  failure
    t for analysis:  (time-origin)/365.25
            origin:  time dob

--------------------------------------------------------------------------------
      337  total observations
        0  exclusions
--------------------------------------------------------------------------------
      337  observations remaining, representing
      337  subjects
       80  failures in single-failure-per-subject data
 4603.669  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =  30.07529
                                          last observed exit t =  69.99863
```

The origin is set to date of birth, making time-since-birth analysis time, and the scale is set to 365.25, so that time since birth is measured in years.

Let's list a few records and verify that the analysis-time variables _t0 and _t are indeed recorded as we expect:

```
. list id dob doe dox fail _t0 _t if id==1 | id==34
```

	id	dob	doe	dox	fail	_t0	_t
1.	1	04jan1915	16aug1964	01dec1976	0	49.615332	61.908282
34.	34	12jun1899	16apr1959	31dec1966	3	59.841205	67.550992

We see that patient 1 was 49.6 years old at time of entry into our study and left at age 61.9. Patient 34 entered the study at age 59.8 and exited the study with CHD at age 67.6.

Now we can split the data by age:

```
. stsplit ageband, at(40(10)70)
(418 observations (episodes) created)
```

stsplit added 418 observations to the dataset in memory and generated a new variable, ageband, which identifies each observation's age group.

```
. list id _t0 _t ageband fail height if id==1 | id==34
```

	id	_t0	_t	ageband	fail	height
1.	1	49.615332	50	40	.	175.387
2.	1	50	60	50	.	175.387
3.	1	60	61.908282	60	0	175.387
61.	34	59.841205	60	50	.	177.8
62.	34	60	67.550992	60	3	177.8

The single record for the subject with id = 1 has expanded to three records. The first refers to the age band 40–49, coded 40, and the subject spends _t − _t0 = 0.384668 years in this band. The second refers to the age band 50–59, coded 50, and the subject spends 10 years in this band, and so on. The follow-up in each of the three bands is censored (fail = .). The single record for the subject with id = 34 is expanded to two age bands; the follow-up for the first band was censored (fail = .), and the follow-up for the second band ended in CHD (fail = 3).

The values for variables that do not change with time, such as height, are simply repeated in the new records. This can lead to much larger datasets after expansion. Dropping unneeded variables before using split may be necessary.

◁

▷ Example 2: Splitting on age and time in study

To use stsplit to expand the records on two time scales simultaneously, such as age and time in study, we can first expand on the age scale as described in example 1, and then on the time-in-study scale with the command

```
. stsplit timeband, at(0(5)25) after(time=doe)
(767 observations (episodes) created)
. list id _t0 _t ageband fail if id==1 | id==34
```

	id	_t0	_t	ageband	fail
1.	1	49.615332	50	40	.
2.	1	50	54.615332	50	.
3.	1	54.615332	59.615332	50	.
4.	1	59.615332	60	50	.
5.	1	60	61.908282	60	0
111.	34	59.841205	60	50	.
112.	34	60	64.841205	60	.
113.	34	64.841205	67.550992	60	3

By splitting the data by using two time scales, we partitioned the data into time cells corresponding to a *Lexis diagram* as described, for example, in Clayton and Hills (1993). Also see Keiding (1998) for an overview of Lexis diagrams. Each new observation created by splitting the data records the time that the individual spent in a Lexis cell. We can obtain the time spent in the cell by calculating the difference `_t` − `_t0`. For example, the subject with `id` = 1 spent 0.384668 years (50 − 49.615332) in the cell corresponding to age 40–49 and study time 0–5, and 4.615332 years (54.615332 − 50) in the cell for age 50–59 and study time 0–5.

We can also do these expansions in reverse order, that is, split first on study time and then on age.

◁

▷ Example 3: Explanatory variables that change with time

In the previous examples, time, in the form of age or time in study, is the explanatory variable to be studied or controlled for, but in some studies other explanatory variables also vary with time. The `stsplit` command can sometimes be used to expand the records so that in each new record such an explanatory variable is constant over time. For example, in the Stanford heart data (see [ST] **stset**), we would like to split the data and generate the explanatory variable `posttran`, which takes the value 0 before transplantation and 1 thereafter. The follow-up must therefore be divided into time before transplantation and time after.

We first generate for each observation an entry time and an exit time that preserve the correct follow-up time in such a way that the time of transplants is the same for all individuals. By summarizing `wait`, the time to transplant, we obtain its maximum value of 310. By selecting a value greater than this maximum, say, 320, we now generate two new variables:

```
. use http://www.stata-press.com/data/r13/stanford, clear
(Heart transplant data)
. generate enter = 320 - wait
. generate exit = 320 + stime
```

We have created a new artificial time scale where all transplants are coded as being performed at time 320. By defining `enter` and `exit` in this manner, we maintain the correct total follow-up time for each patient. We now `stset` and `stsplit` the data:

```
. stset exit, enter(time enter) failure(died) id(id)
                id:  id
     failure event:  died != 0 & died < .
obs. time interval:  (exit[_n-1], exit]
 enter on or after:  time enter
 exit on or before:  failure
------------------------------------------------------------------------------
      103  total observations
        0  exclusions
------------------------------------------------------------------------------
      103  observations remaining, representing
      103  subjects
       75  failures in single-failure-per-subject data
  34589.1  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =        10
                                          last observed exit t =      2119

. stsplit posttran, at(0,320)
(69 observations (episodes) created)

. replace posttran=0 if transplant==0
(34 real changes made)

. replace posttran=1 if posttran==320
(69 real changes made)
```

We replaced `posttran` in the last command so that it is now a 0/1 indicator variable. We can now `generate` our follow-up time, `t1`, as the difference between our analysis-time variables, `list` the data, and `stset` the dataset.

```
. generate t1 =_t - _t0

. list id enter exit _t0 _t posttran if id==16 | id==44
```

	id	enter	exit	_t0	_t	posttran
41.	44	320	360	320	360	0
110.	16	292	320	292	320	0
111.	16	292	628	320	628	1

```
. stset t1, failure(died) id(id)
                id:  id
     failure event:  died != 0 & died < .
obs. time interval:  (t1[_n-1], t1]
 exit on or before:  failure
------------------------------------------------------------------------------
      172  total observations
        0  exclusions
------------------------------------------------------------------------------
      172  observations remaining, representing
      103  subjects
       75  failures in single-failure-per-subject data
  31938.1  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =      1799
```

Using stsplit to split at failure times

To split data at failure times, you would use `stsplit` with the following syntax, ignoring other options:

`stsplit` [*if*] `, at(failures)`

This form of episode splitting is useful for Cox regression with time-varying covariates. Splitting at the failure times is useful because of a property of the maximum partial-likelihood estimator for a Cox regression model: the likelihood is evaluated only at the times at which failures occur in the data, and the computation depends only on the risk pools at those failure times. Changes in covariates between failure times do not affect estimates for a Cox regression model. Thus, to fit a model with time-varying covariates, all you have to do is define the values of these time-varying covariates at all failure times at which a subject was at risk (Collett 2003, chap. 8). After splitting at failure times, you define time-varying covariates by referring to the system variable `_t` (analysis time) or the *timevar* variable used to `stset` the data.

After splitting at failure times, all st commands still work fine and produce the same results as before splitting. To fit parametric models with time-varying covariates, it does not suffice to specify covariates at failure times. Stata can fit "piecewise constant" models by manipulating data using `stsplit,` {`at()` | `every()`}.

◁

▷ Example 4: Splitting on failure times to test the proportional-hazards assumption

Collett (2003, 187–190) presents data on 26 ovarian cancer patients who underwent two different chemotherapy protocols after a surgical intervention. Here are a few of the observations:

```
. use http://www.stata-press.com/data/r13/ocancer, clear
. list patient time cens treat age rdisea in 1/6, separator(0)
```

	patient	time	cens	treat	age	rdisea
1.	1	156	1	1	66	2
2.	2	1040	0	1	38	2
3.	3	59	1	1	72	2
4.	4	421	0	2	53	2
5.	5	329	1	1	43	2
6.	6	769	0	2	59	2

The `treat` variable indicates the chemotherapy protocol administered, `age` records the age of the patient at the beginning of the treatment, and `rdisea` records each patient's residual disease after surgery. After `stset`ting this dataset, we fit a Cox proportional-hazards regression model on `age` and `treat` to ascertain the effect of treatment, controlling for age.

```
. stset time, failure(cens) id(patient)
                id:  patient
     failure event:  cens != 0 & cens < .
obs. time interval:  (time[_n-1], time]
 exit on or before:  failure
------------------------------------------------------------------------------
       26  total observations
        0  exclusions
------------------------------------------------------------------------------
       26  observations remaining, representing
       26  subjects
       12  failures in single-failure-per-subject data
    15588  total analysis time at risk and under observation
                                        at risk from t =         0
                              earliest observed entry t =         0
                                   last observed exit t =      1227

. stcox age treat, nolog nohr
         failure _d:  cens
   analysis time _t:  time
                 id:  patient

Cox regression -- no ties

No. of subjects =           26                     Number of obs   =        26
No. of failures =           12
Time at risk    =        15588
                                                   LR chi2(2)      =     15.82
Log likelihood  =   -27.073767                     Prob > chi2     =    0.0004

------------------------------------------------------------------------------
          _t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |    .1465698   .0458537     3.20   0.001     .0566982    .2364415
       treat |   -.7959324   .6329411    -1.26   0.209    -2.036474    .4446094
------------------------------------------------------------------------------
```

One way to test the proportional-hazards assumption is to include in the model a term for the interaction between age and time at risk, which is a continuously varying covariate. This can be easily done by first splitting the data at the failure times and then generating the interaction term.

```
. stsplit, at(failures)
(12 failure times)
(218 observations (episodes) created)

. generate tage = age * _t

. stcox age treat tage, nolog nohr
         failure _d:  cens
   analysis time _t:  time
                 id:  patient

Cox regression -- no ties

No. of subjects =           26                     Number of obs   =       244
No. of failures =           12
Time at risk    =        15588
                                                   LR chi2(3)      =     16.36
Log likelihood  =   -26.806607                     Prob > chi2     =    0.0010

------------------------------------------------------------------------------
          _t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |    .2156499   .1126093     1.92   0.055    -.0050602      .43636
       treat |   -.6635945   .6695492    -0.99   0.322    -1.975887    .6486978
        tage |   -.0002031   .0002832    -0.72   0.473    -.0007582     .000352
------------------------------------------------------------------------------
```

Other time-varying interactions of age and time at risk could be generated. For instance,

```
. generate lntage = age * ln(_t)
. generate dage = age * (_t >= 500)
```

Although in most analyses in which we include interactions we also include main effects, if we include in a Cox regression a multiplicative interaction between analysis time (or any transformation) and some covariate, we should not include the analysis time as a covariate in `stcox`. The analysis time is constant within each risk set, and hence, its effect is not identified.

❑ Technical note

If our interest really were just in performing this test of the proportional-hazards assumption, we would not have had to use `stsplit` at all. We could have just typed

```
. stcox age treat, tvc(age)
```

to have fit a model including t`*age`, and if we wanted instead to include $\ln(t)$`*age` or `age*`$t \geq 500$, we could have typed

```
. stcox age treat, tvc(age) texp(ln(_t))
. cstoc age treat, tvc(age) texp(_t >= 500)
```

Still, it is worth understanding how `stsplit` could be used to obtain the same results for instances when `stcox`'s `tvc()` and `texp()` options are not rich enough to handle the desired specification.

❑

Assume that we want to control for `rdisea` as a stratification variable. If the data are already split at all failure times, we can proceed with

```
. stcox age treat tage, strata(rdisea)
```

If the data are not yet split, and memory is scarce, then we could just split the data at the failure times within the respective stratum. That is, with the original data in memory, we could type

```
. stset time, failure(cens) id(patient)
. stsplit, at(failures) strata(rdisea)
. generate tage = age * _t
. stcox treat age tage, strata(rdisea)
```

This would save memory by reducing the size of the split dataset.

◁

❑ Technical note

Of course, the above model could also be obtained by typing

```
. stcox age treat, tvc(age) strata(rdisea)
```

without splitting the data.

❑

Example 5: Cox regression versus conditional logistic regression

Cox regression with the "exact partial" method of handling ties is tightly related to conditional logistic regression. In fact, we can perform Cox regression via `clogit`, as illustrated in the following example using Stata's cancer data. First, let's fit the Cox model.

```
. use http://www.stata-press.com/data/r13/cancer, clear
(Patient Survival in Drug Trial)
. generate id =_n
. stset studytim, failure(died) id(id)
                id:  id
     failure event:  died != 0 & died < .
obs. time interval:  (studytime[_n-1], studytime]
 exit on or before:  failure
------------------------------------------------------------------------------
       48  total observations
        0  exclusions
------------------------------------------------------------------------------
       48  observations remaining, representing
       48  subjects
       31  failures in single-failure-per-subject data
      744  total analysis time at risk and under observation
                                        at risk from t =         0
                              earliest observed entry t =         0
                                   last observed exit t =        39
. stcox age drug, nolog nohr exactp
         failure _d:  died
   analysis time _t:  studytime
                 id:  id
Cox regression -- exact partial likelihood
No. of subjects =           48                     Number of obs   =        48
No. of failures =           31
Time at risk    =          744
                                                   LR chi2(2)      =     38.13
Log likelihood  =    -73.10556                     Prob > chi2     =    0.0000
------------------------------------------------------------------------------
          _t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .1169906   .0374955     3.12   0.002     .0435008    .1904805
        drug |  -1.664873   .3437487    -4.84   0.000    -2.338608   -.9911376
------------------------------------------------------------------------------
```

We will now perform the same analysis by using `clogit`. To do this, we first split the data at failure times, specifying the `riskset()` option so that a risk set identifier is added to each observation. We then fit the conditional logistic regression, using `_d` as the outcome variable and the risk set identifier as the grouping variable.

```
. stsplit, at(failures) riskset(RS)
(21 failure times)
(534 observations (episodes) created)
. clogit _d age drug, group(RS) nolog
note: multiple positive outcomes within groups encountered.
Conditional (fixed-effects) logistic regression   Number of obs    =        573
                                                  LR chi2(2)       =      38.13
                                                  Prob > chi2      =     0.0000
Log likelihood =  -73.10556                       Pseudo R2        =     0.2069
```

_d	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
age	.1169906	.0374955	3.12	0.002	.0435008	.1904805
drug	-1.664873	.3437487	-4.84	0.000	-2.338608	-.9911376

◁

▷ Example 6: Joining data that have been split with stsplit

Let's return to the first example. We split the diet data into age bands, using the following commands:

```
. use http://www.stata-press.com/data/r13/diet, clear
(Diet data with dates)
. stset dox, failure(fail) origin(time dob) enter(time doe) scale(365.25) id(id)
  (output omitted)
. stsplit ageband, at(40(10)70)
(418 observations (episodes) created)
```

We can rejoin the data by typing `stjoin`:

```
. stjoin
(option censored(0) assumed)
(0 obs. eliminated)
```

Nothing happened! `stjoin` will combine records that are contiguous and record the same data. Here, when we split the data, `stsplit` created the new variable `ageband`, and that variable takes on different values across the split observations. Remember to drop the variable that `stsplit` creates:

```
. drop ageband
. stjoin
(option censored(0) assumed)
(418 obs. eliminated)
```

◁

Wilhelm Lexis (1837–1914) was born near Aachen in Germany. He studied law, mathematics, and science at the University of Bonn and developed interests in the social sciences during a period in Paris. Lexis held posts at universities in Strassburg (now Strasbourg, in France), Dorpat (now Tartu, in Estonia), Freiburg, Breslau (now Wroclaw, in Poland), and Göttingen. During this peripatetic career, he carried out original work in statistics on the analysis of dispersion, foreshadowing the later development of chi-squared and analysis of variance.

Acknowledgments

`stsplit` and `stjoin` are extensions of `lexis` by David Clayton of the Cambridge Institute for Medical Research and Michael Hills (retired) of the London School of Hygiene and Tropical Medicine (Clayton and Hills 1995). The original `stsplit` and `stjoin` commands were written by Jeroen Weesie of the Department of Sociology at Utrecht University, The Netherlands (Weesie 1998a, 1998b), as was the revised `stsplit` command.

References

Clayton, D. G., and M. Hills. 1993. *Statistical Models in Epidemiology*. Oxford: Oxford University Press.

——. 1995. ssa7: Analysis of follow-up studies. *Stata Technical Bulletin* 27: 19–26. Reprinted in *Stata Technical Bulletin Reprints*, vol. 5, pp. 219–227. College Station, TX: Stata Press.

Cleves, M. A., W. W. Gould, R. G. Gutierrez, and Y. V. Marchenko. 2010. *An Introduction to Survival Analysis Using Stata*. 3rd ed. College Station, TX: Stata Press.

Collett, D. 2003. *Modelling Survival Data in Medical Research*. 2nd ed. London: Chapman & Hall/CRC.

Hertz, S. 2001. Wilhelm Lexis. In *Statisticians of the Centuries*, ed. C. C. Heyde and E. Seneta, 204–207. New York: Springer.

Keiding, N. 1998. Lexis diagrams. In *Encyclopedia of Biostatistics*, ed. P. Armitage and T. Colton, 2844–2850. New York: Wiley.

Lexis, W. H. 1875. *Einleitung in die Theorie der Bevölkerungsstatistik*. Strassburg: Trübner.

Mander, A. P. 1998. gr31: Graphical representation of follow-up by time bands. *Stata Technical Bulletin* 45: 14–17. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 50–53. College Station, TX: Stata Press.

Morris, J. N., J. W. Marr, and D. G. Clayton. 1977. Diet and heart: A postscript. *British Medical Journal* 19: 1307–1314.

Weesie, J. 1998a. ssa11: Survival analysis with time-varying covariates. *Stata Technical Bulletin* 41: 25–43. Reprinted in *Stata Technical Bulletin Reprints*, vol. 7, pp. 268–292. College Station, TX: Stata Press.

——. 1998b. dm62: Joining episodes in multi-record survival time data. *Stata Technical Bulletin* 45: 5–6. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 27–28. College Station, TX: Stata Press.

Also see

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