A brief introduction to survival analysis using Stata

June 2012

Paul W. Dickman — Department of Medical Epidemiology and Biostatistics
Karolinska Institutet, Stockholm, Sweden
paul.dickman@ki.se
http://ki.se/research/pauldickman
http://www.pauldickman.com/

Paul C. Lambert — Centre for Biostatistics & Genetic Epidemiology
University of Leicester, UK
pl4@le.ac.uk
http://www.hs.le.ac.uk/personal/pl4/

Therese M-L. Andersson — Department of Medical Epidemiology and Biostatistics
Karolinska Institutet, Stockholm, Sweden
therese.m-l.andersson@ki.se
http://ki.se/research/thereseandersson

Sandra Eloranta — Department of Medical Epidemiology and Biostatistics
Karolinska Institutet, Stockholm, Sweden
sandra.eloranta@ki.se
http://ki.se/research/sandraeloranta

1 A brief introduction to Stata

This is a brief introduction to survival analysis using Stata.

Starting Stata

Double-click the Stata icon on the desktop (if there is one) or select Stata from the Start menu.

Closing Stata

Choose eXit from the file menu, click the Windows close box (the 'x' in the top right corner), or type `exit` at the command line. You will have to type `clear` first if you have any data in memory (or simply type `exit, clear`). Note that Stata is case sensitive. To interrupt a Stata command, click on break or press `ctrl break`.

Useful Stata links

Resources for learning Stata can be found at
http://www.stata.com/links/resources1.html

Getting help

Stata has extensive online help. Click on Help, or type `help` followed by a command name at the command line.

Types of Stata files

Data files in Stata format are given the extension `.dta`. These are created using `save filename` and read in with `use filename`. There are four other types of input file: `.raw` for raw data, `.dct` for data plus variable names, `.do` for batch files containing Stata commands, `.ado` for Stata programs, and `.log` for log files.

Syntax

`command varnames if ... in ... using ... , options`

The `if` part restricts the command to records satisfying certain logical conditions (eg sex==1), the `in` part restricts the command to certain line numbers, and the `using` part specifies any files which may be needed.

Abbreviations

Stata accepts unambiguous abbreviations for commands and variable names.

2 A 'hands-on' introduction to Stata

To introduce you to Stata we use the diet data which consists of 337 records of individuals from a pilot study evaluating the use of a weighed diet over 7 days in epidemiological studies. The primary hypothesis is the relation between dietary energy intake and incidence of coronary heart disease (CHD). The variables in the dataset are shown below.

```
 obs: 337
-------------------------------------------------------------------
          storage  display   value
variable    type   format    label    variable label
-------------------------------------------------------------------
id          int    %9.0g               Subject identity number
chd         byte   %9.0g               Failure: 1=chd, 0 otherwise
y           float  %9.0g               Time in study (years)
hieng       float  %12.0g    hieng    Indicator for high energy
energy      float  %9.0g               Total energy (kcals per day)
job         byte   %9.0g     job      Occupation
month       byte   %8.0g               Month of survey
height      float  %9.0g               Height (cm)
weight      float  %9.0g               Weight (kg)
doe         int    %dDmCY              Date of entry
dox         int    %dDmCY              Date of exit
dob         int    %dDmCY              Date of birth
-------------------------------------------------------------------
```

Type in the commands which start with the Stata prompt ('.'). Do not type the `.` prompt – this is used to indicate a Stata command. Stata distinguishes between upper and lower case letters, and accepts abbreviations for both commands and variable names. Think carefully about what is happening after each command.

The diet data contains the variables names and values for the 337 records and can be accessed over the web from within Stata. To read the data, type

```
. use use http://www.pauldickman.com/survival/diet
. describe
```

Now type the following

```
. Describe
```

Stata will return an error message (`unrecognised command: Describe`). Stata is case sensitive; `describe` is a valid Stata command, whereas `Describe` is not. A good way to start the analysis is to ask for a summary of the data by typing

```
. summarize
```

This will produce the mean, standard deviation, and range, for each variable in turn. In most datasets there will be some missing values. These are coded using the symbol `.` in place of the value which is missing. Stata can recognize other codes for missing values, but this is the one which is recommended. The `summarize` command is useful for seeing whether there are missing values (the column labelled 'Obs' gives the number of non-missing observations).

For a more detailed summary of the variable `weight` try

```
. codebook weight
```

or

```
. summarize weight, detail
```

Many Stata commands can be accessed using menus. For example, from the `Summaries` menu, select `Median/Percentiles`. You will notice that the result is identical to that obtained from the command typed previously (`summarize weight, detail`) and that Stata even shows the command which was used.

The list command is used to list the values in the data file. Try out the following and see their consequences:

```
. list in 1/5
. list weight in 1/10
. list weight
. list weight height hieng chd in 1/20
```

Stata stops after each screenfull of output. Click on more (or hit the spacebar) to get another screenfull, or press ENTER to continue line by line. The command `list` on its own would list all of the data. You can cancel this command (and any other Stata command) by clicking on Break (the icon in the toolbar which looks like a red circle with a white cross through it).

Stata also contains a spreadsheet-style editor which can be brought to the front by typing

```
. edit
```

Close this window by clicking in the close box (in the top right corner of the window).

The `browse` command will bring up a similar window, except changes cannot be made to the data. The data window can also be opened using icons on the toolbar (the two icons look like spreadsheets, with a magnifying glass over the data browser icon) or from the Data menu.

When starting to look at any new data the first step is to check that the values of the variables make sense and correspond to the codes defined in the coding schedule. For categorical variables this can be done by looking at one-way frequency tables and checking that only the specified codes occur. For metric variables we need to look at ranges.

This first look at the data will also indicate whether all values are present or whether there are some missing values on some variables. Let us begin by looking at the categorical variables. The distribution of the categorical variables `stage` and `sex` can be viewed by typing

```
. tabulate hieng
. tab job, missing
```

The missing option can be used to ensure that any missing values are displayed as a separate category. However, in this data set there are no missing values for any of the categorical variables. The missing value symbol in Stata is . and is treated as plus infinity in logical comparisons. Stata commands automatically exclude missing values when they are coded in this way.

Note that `tab` is an abbreviation for `tabulate`. The cross-tabulation of `hieng` and `job` is obtained by typing

```
. tab hieng job
```

Cross tabulations are useful when checking for consistency. The basic output from a cross tabulation reports frequencies only; to include row and/or column percentages add the options `row`, `col`, `cell`, or any combination, as in

```
. tab hieng job, col missing
```

The command `table` is used for preparing tables of summary statistics by one, two, or even more categorical variables. For example, to obtain the means and standard deviations of `weight` separately by `hieng`, type

```
. table hieng, contents(freq mean weight sd weight)
```

To make a table of the median and interquartile range for age, by sex, try

```
. table hieng, contents(freq med weight iqr weight)
```

Note that `tab` is an abbreviation for `tabulate`, NOT for `table`, which must be typed in full. You can type `whelp tabulate` and `whelp table` to understand how, if, you can abbreviate the command.

2.1 Restricting commands

Stata commands can be restricted to records $1, 2, \ldots, 10$ (for example), by adding `in 1/10` to the command. The letters `f` and `l` can be used as abbreviations for first and last, so `20/l` refers to the records from 20 onwards. Commands can also be restricted to operate only on records which satisfy given conditions. The conditions are added to the command using `if` followed by a logical expression which takes the values true or false. For example, to restrict the command `list` to records with daily energy intake less than or equal to 2000, type

```
. list if energy <= 2000
```

The record is listed only if the logical expression `energy <= 2000` is true.

A useful command when exploring data is `count` which counts the number of records which satisfy some logical expression. For example

```
. count if energy <= 2000
. count if energy <= 2000 & job==1
```

> Note the use of `&` to link two conditions both of which must be satisfied and that a double equal sign (`==`) is used for equality testing. A common error is to use `=` in a logical expression instead of `==`.

The following comparison operators and logical functions are available:

Arithmetic		Logical		Comparison	
`+`	addition	`~`	not	`>`	greater than
`-`	subtraction	`|`	or	`<`	less than
`*`	multiplication	`&`	and	`>=`	> or equal
`/`	division			`<=`	< or equal
`^`	power			`==`	equal
				`~=`	not equal

2.2 Generating and recoding variables

New variables are generated using the command `generate`, and variables can be recoded using `recode`. For example, to create a new variable `job2` which is the same as `job` but coded 0 to 2 instead of 1 to 3, try

```
. gen job2=job
. recode job2 1=0 2=1 3=2
. tab job2
```

2.3 Sorting

The records in a dataset can be sorted according to the values of one or more variables. The `diet` dataset is currently not sorted but we might have it sorted by any variable or combinations of several variables. Try the following commands and make sure you understand the sorting procedure in Stata.

```
. list job weight in 1/10
. sort job
. list job weight in 1/10
. sort job weight
. list job weight in 1/10
```

Stata commands which use the option `by()` usually require the data to be first sorted by the variable in the `by()` option. The sort is not done automatically because you should always be aware of how your data are sorted. You can check how your data is sorted using the `describe` command.

2.4 Editing commands

The 'PageUp' and 'PageDown' keys (represented as arrows on the top right of the keypad) can be used to cycle through previous commands, which can then be edited. For example, if you decide that you would also like to list the values of the variable `height` you could use the 'PageUp' key to recall the previous command and then edit it in the command line to be:

```
. list job weight height in 1/10
```

This capability is especially useful if you make a small mistake while typing a command. The command can be recalled, edited, and resubmitted. It also makes it easy to resubmit the same command with additional options.

2.5 Using Stata as a calculator

The `display` command can be used to carry out simple calculations. For example, the command

```
. display 2+2
```

will display the answer 4, while

```
. display log(10)
```

will display the answer 2.3026. Note that log means natural log in Stata. To obtain base 10 logarithms use the `log10` function. For example,

```
. display log10(1000)
```

will return the value 3.

2.6 Saving data files

The Stata data currently in memory can be saved in a file by clicking on the Save icon (the floppy disk) on the toolbar. You will need to type in a name for your file which, by default, will be saved in the default directory with the extension `.dta`.

2.7 Logging and printing results

Graphs can be printed directly by selecting 'Print graph' from the File menu, or you can copy it and past it into any of your word processor (for instance MS Word). Other output must first be written to a log file before it can be printed. A log file can be opened by clicking on the log icon on the toolbar (the fourth icon from the left. You will need to type in a name for your file which, by default, will be saved in your personal directory with the extension `.log`.

3 Survival data with Stata

The `stset` command is used to tell Stata the format of your survival data. You only have to 'tell' Stata once after which all survival analysis commands (the `st` commands) will use this information. For example, after using `stset`, a Cox proportional hazards model with job and hieng as covariates can be fitted using

```
. stcox job hieng
```

At a minimum Stata needs to know the time at risk (e.g., time from diagnosis to death or censoring) and the failure indicator (e.g., whether or not the patient died). However, the `stset` command is very flexible and powerful for setting up more complicated survival data. We will explain the use of the `stset` command through a number of examples.

3.1 Syntax of the `stset` command

`stset` *timevar* `[if] [weight] , failure(failvar[==numlist]) [`*options*`]`

For example,

```
stset survtime, failure(dead==1)
```

would be appropriate if the time at risk for each individual is in the variable `survtime` and the variable `dead` is an indicator for death.

- The *timevar* variable is compulsory. It is the survival time (or a date) of the event/censoring time.
- The `failure(`*failvar* = *numlist*`)` option is optional, but it is good practice to always use it. If this option is omitted then it is assumed that all subjects experience the event. It is a number list (`numlist` giving the values indicating a failure. In many cases this will be a single number, but the use of a number list is useful if, for example, you have different codings for different causes of death.
- The `exit` option gives the latest time at which the subject is at risk. The default is `exit(failure)`, i.e. the subject is removed from the risk set after their event. This command is useful if you want to restrict follow-up time. For example if you are using dates to define your survival times, but you want to restrict follow-up time to 31/12/2005, you can use `exit(time mdy(12,31,2005))`. If you have multiple failures then you need to specify `exit(time .)` as the default is to remove the subject from the risk set after their first failure.
- The `origin` option gives the time origin of the time-scale, that is, it is used to define when time is zero. The default is zero. For example, if we have variables representing date of diagnosis and date of exit and wish to analyse time since diagnosis then the time origin should be defined as the date of diagnosis (since the day of diagnosis is time zero for each individual). Similarly, if we wish to use attained age as the timescale then the time origin is the date of birth.
- The `enter` option gives the time at which the subject becomes at risk. You are likely to use this option if using age as the time scale. For example, if there is a date of diagnosis then you will use `enter(datediag)`. It is also useful if patients are only considered to be at risk after a certain date (e.g., in period analysis). For example, if we only want to consider time at risk after 1/1/2001 use `enter(time mdy(1,1,2001))`.

- The `scale(#)` option transforms the survival time. For example to transform the timescale from days to years use `scale(365.25)`.
- The `id(`*varname*`)` option specifies an identification number for each subject. This option is not compulsory, but it is good practice to specify it as the `stsplit` command requires an ID variable. If there are multiple failures the the `id` option must be specified.

The above are the most common options - see the manual or online help for other options. Try typing

```
. help stset
```

3.2 Variables created by the `stset` command

The `stset` command creates 4 variables. These variables contain all the necessary information for the survival data. These variables are

`_t0` - analysis time when record begins (time at which individual becomes at risk)
`_t` - analysis time when record ends (time at which individual stops being at risk)
`_d` - failure indicator: 1 if failure, 0 if censored
`_st` - 1 if the record is included in `st` analyses, 0 if excluded

All the survival analysis (`st`) commands use these variables, as all information regarding survival times is contained within these four variables.

3.2.1 'Standard' survival data

To demonstrate the use of the `stset` command we first need to create two variables that contain information about each individuals survival time. The survival times are defined as the elapsed time between the date of entry and the date of exit.

```
. gen survdays = dox - doe
. gen survyear = survdays/365.25
```

If the survival time and censoring indicator have already been created then `stset` can be used as follows

```
. stset survyear, failure(chd == 1) id(id)

                id:  id
     failure event:  chd == 1
obs. time interval:  (survyear[_n-1], survyear]
 exit on or before:  failure

--------------------------------------------------------------------------------
      337  total obs.
        0  exclusions
--------------------------------------------------------------------------------
      337  obs. remaining, representing
      337  subjects
       46  failures in single failure-per-subject data
 4603.669  total analysis time at risk, at risk from t =         0
                             earliest observed entry t =         0
                                  last observed exit t =  20.04107
```

The `id` option is not compulsory here as there should only be one row of data per subject. However, it is good practice to include it, as if splitting the data later using `stsplit` then the data must previously have been `stset` using the `id` option.

The output gives some summary information. You should check this output to see if there are any exclusions (e.g. for zero or negative survival times), that the number of events corresponds to what you expect etc. *What is the range of the survival times in this data set? How many CHD events are there in this data and what is the total number of person-years?* Ask for help if you do not know how to answer these questions.

```
. list id _t0 _t _d _st in 1/15
```

	id	_t0	_t	_d	_st
1.	127	0	16.791239	0	1
2.	200	0	19.958933	0	1
3.	198	0	19.958933	0	1
4.	222	0	15.394935	0	1
5.	305	0	1.4948665	1	1
6.	173	0	15.958932	0	1
7.	120	0	11.989049	0	1
8.	206	0	19.87406	0	1
9.	81	0	15.958932	0	1
10.	333	0	15.460644	0	1
11.	323	0	15.460644	0	1
12.	192	0	19.33744	0	1
13.	102	0	10.874743	0	1
14.	183	0	14.658453	1	1
15.	212	0	17.30048	0	1

The `stset` command has created four new variables. For this example `_t0` is 0 for all subjects; this is the default value (we have not used the enter option) and corresponds to all subjects being at risk from time 0, i.e., when they enter the study. The variable `_t` gives the survival or censoring time, i.e. when the subject stops being at risk due to experiencing the event (i.e. CHD) or censoring. The `_d` variable is the event indicator (0 if censored and 1 if an event). The `_st` variable specifies whether the observation should be included in the analysis (1 = include, 0 = exclude). `_st` will be zero if survival times are recorded as zero (or are negative) or if an `if` or `in` option was specified in the `stset` command.

3.2.2 Using the `scale` option

If survival time is measured in days and you would like the analysis time to be in years then use the `scale` option. For example

```
. stset survdays, failure(chd == 1) id(id) scale(365.25)

                id:  id
     failure event:  chd == 1
obs. time interval:  (survdays[_n-1], survdays]
 exit on or before:  failure
    t for analysis:  time/365.25

--------------------------------------------------------------------------------
      337  total obs.
        0  exclusions
--------------------------------------------------------------------------------
      337  obs. remaining, representing
      337  subjects
       46  failures in single failure-per-subject data
 4603.669  total analysis time at risk, at risk from t =         0
                             earliest observed entry t =         0
                                  last observed exit t =  20.04107
```

The survival time (in days) is divided by 365.25 to give survival time in years. This is noted in the output from the `stset` command.

The variables created by `stset` (`_t0 _t _d _st`) are exactly the same as the previous example (*Check this for the first 15 observations!*). This is to be expected as the `survyears` variable was calculated in same way as used by `stset`. It is usually safer to let `stset` to do the rescaling for you. There are other advantages, for example when using the `stsplit` command you are able to specify some options that need to remember that you have rescaled the data.

3.2.3 Using date of diagnosis and date of exit

It is common to have data that record various dates. For example, the date of diagnosis of a particular disease, the date of death or end of follow-up, the date of birth or the date patients were given particular treatments. It is of course fairly easy to use any package to calculate various times from these dates, but the `stset` command can do most of this work for you. For a general introduction to the different date functions in Stata please visit the
http://www.ats.ucla.edu/stat/stata/modules/dates.htm.

```
. stset dox, failure(chd==1) id(id) scale(365.25) origin(doe)

                  id:  id
     failure event:  chd == 1
obs. time interval:  (dox[_n-1], dox]
 exit on or before:  failure
    t for analysis:  (time-origin)/365.25
            origin:  time doe

--------------------------------------------------------------------------------
      337  total obs.
        0  exclusions
```

```
------------------------------------------------------------------------------
      337  obs. remaining, representing
      337  subjects
       46  failures in single failure-per-subject data
 4603.669  total analysis time at risk, at risk from t =         0
                             earliest observed entry t =         0
                                  last observed exit t =  20.04107
```

In the output from `stset` it is reported that `t for analysis: (time - origin)/365.25`, which is what we want. As the dates are stored in units of days and we have used the `scale()` option, the analysis time is also in units of years. Note that the variables created by `stset` (`_t0 _t _d _st`) are exactly the same as in sections 3.2.1 and 3.2.2.

3.2.4 Restricting the follow-up time

In some instances it may be necessary to define the maximum follow-up time. This may be because follow-up information after a certain date may be unreliable. Alternatively, you may only be interested in follow-up to a certain time after diagnosis. For example, if there are only a few individuals alive after ten years, you may want to restrict follow-up to 10 years. This can be achieved using the `exit` option.

```
. stset dox, failure(chd==1) id(id) scale(365.25) origin(doe) exit(time doe + 10*365.25)

                id:  id
     failure event:  chd == 1
obs. time interval:  (dox[_n-1], dox]
 exit on or before:  time doe + 10*365.25
    t for analysis:  (time-origin)/365.25
            origin:  time doe

------------------------------------------------------------------------------
      337  total obs.
        0  exclusions
------------------------------------------------------------------------------
      337  obs. remaining, representing
      337  subjects
       29  failures in single failure-per-subject data
 3096.133  total analysis time at risk, at risk from t =         0
                             earliest observed entry t =         0
                                  last observed exit t =        10
```

The option `exit(time doe + 10*365.25)` truncates the time scale ten years after entry to the study. *What is now the range of the survival times? How many CHD events are there now and what is the total number of person-years?*.

```
. list id _t0 _t _d _st in 1/15
```

	id	_t0	_t	_d	_st
1.	127	0	10	0	1
2.	200	0	10	0	1
3.	198	0	10	0	1
4.	222	0	10	0	1
5.	305	0	1.4948665	1	1
6.	173	0	10	0	1
7.	120	0	10	0	1
8.	206	0	10	0	1
9.	81	0	10	0	1
10.	333	0	10	0	1
11.	323	0	10	0	1
12.	192	0	10	0	1
13.	102	0	10	0	1
14.	183	0	10	0	1
15.	212	0	10	0	1

Note that the variables created by `stset (_t0 _t _d _st)` are no longer the same as in sections 3.2.1 and 3.2.2. For example, what happened to the event indicator for the person with id = 183 and why?

3.2.5 Left truncation

We can left truncate the time scale using the `enter` option. This will also be used when we use age as the time scale in section 3.2.6. An example of when left truncation is used is in *period analysis* where only the survival experience of subjects who are at risk in a recent time period are included in the analysis. For example, if we only want to include the survival times after 1/1/1960 we can use `enter(time mdy(1,1,1960))`.

```
. stset dox, failure(chd==1) id(id) scale(365.25) origin(doe) enter(time mdy(1,1,1960))

                id:  id
     failure event:  chd == 1
obs. time interval:  (dox[_n-1], dox]
 enter on or after:  time mdy(1,1,1960)
 exit on or before:  failure
    t for analysis:  (time-origin)/365.25
            origin:  time doe
--------------------------------------------------------------------------------
      337  total obs.
        5  obs. end on or before enter()
--------------------------------------------------------------------------------
      332  obs. remaining, representing
      332  subjects
       41  failures in single failure-per-subject data
 4361.862  total analysis time at risk, at risk from t =         0
                              earliest observed entry t =         0
                                   last observed exit t =  20.04107
```

```
. list id _t0 _t _d _st in 1/5

     +---------------------------------------+
     |  id         _t0          _t   _d   _st |
     |---------------------------------------|
  1. | 127           0   16.791239    0     1 |
  2. | 200   3.0417522   19.958932    0     1 |
  3. | 198   3.0417522   19.958932    0     1 |
  4. | 222   2.8720055   15.394935    0     1 |
  5. | 305           0   1.4948665    1     1 |
     +---------------------------------------+
```

This is the first time we have observed that `_t0` is not zero. This is because the some subjects entered the study before 1/1/1960 and we have specified that we are only interested in analyzing the survival times after this date. The variable `_t0` is still 0 for subject 127 and 305 as they were diagnosed after 1/1/1960. *How many individuals experienced the event or were censored prior to 1/1/1960?*

3.2.6 Age as the timescale

When using age as the timescale we need to make use of the `enter` and `origin` options. As we are interested in age, the time origin must be the date of birth and the entry time in the study is the date of diagnosis.

```
. stset dox, failure(chd==1) id(id) scale(365.25) origin(dob) enter(doe)

                id:  id
     failure event:  chd == 1
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 obs.
        0  exclusions
------------------------------------------------------------------------------
      337  obs. remaining, representing
      337  subjects
       46  failures in single failure-per-subject data
 4603.669  total analysis time at risk, at risk from t =         0
                             earliest observed entry t =  30.07529
                                  last observed exit t =  69.99863
```

```
. list id _t0 _t _d _st in 1/5

     +---------------------------------------+
     |  id         _t0          _t   _d   _st |
     |---------------------------------------|
  1. | 127    49.38809   66.179329    0     1 |
  2. | 200   47.496235   67.455168    0     1 |
  3. | 198   46.464066   66.422998    0     1 |
  4. | 222   54.603696   69.998631    0     1 |
  5. | 305   46.546201   48.041068    1     1 |
     +---------------------------------------+
```

In the above results the variable `_t0` denotes the age at which the subject entered the study. The variable `_t` denotes the age at which the subject experienced the event or was censored.

3.2.7 Time-splitting

Sometimes when doing a survival analysis we must split the follow-up time (for example when analysing time-varying exposures or prior to fitting a Poisson regression model). For this we can use `stsplit`.

For example, let´s say that we are interested in estimating the rate of CHD-events for different ages using the `diet.dta`. If we would simply type

```
. strate
```

we get an estimate of the average rate of CHD among all individuals during the entire follow-up, irrespective of covariate pattern. How might we get estimates of the age-specific rates in the example where attained age is used as an underlying time scale (remember that age is not a fixed covariate, but takes different values as the individual age throughout follow-up)? For example, individual number 127 entered the study at age 49 and was censored at age 66. If we would like to estimate the rate in 10-year agebands (i.e. <40, (40-50], (50-60], >60) this individual would contribute person-time to all but the first agebands. To achieve this we must use the splitting facilities in Stata (make sure that your data is still `stset` with attained age as the underlying time scale).

```
. stsplit ageband, at(30,40,50,60)
(418 observations (episodes) created)
```

Note that `ageband` is a variable that is created by `stsplit` and that the breaks for splitting the person-time are provided in the `at` option. We can now list the observations belonging to individual number 127 to ensure that the splitting has generated the agebands that we intended.

```
. list id _t0 _t _d ageband if id == 127

     +-----------------------------------------+
     |  id         _t0          _t   _d ageband |
     |-----------------------------------------|
261. | 127   49.38809          50    0      40 |
262. | 127         50          60    0      50 |
263. | 127         60   66.179329    0      60 |
     +-----------------------------------------+
```

We now see that this individual contributed with a small amount of person-time to the (40-50] ageband (which has been given the value label 40), 10 years to the subsequent ageband, and just above 6 years to the last ageband. We also see that at age 66.179329 the individual was censored. Individual 127 hence now contributes with three rows in the splitted data set. In total 418 new observations were generated after the `stsplit`.

Now, we can simply estimate the average rates of CHD within each ageband by typing

```
. strate ageband, per(1000)

         failure _d:  chd == 1
   analysis time _t:  (dox-origin)/365.25
             origin:  time dob
  enter on or after:  time doe
                 id:  id

Estimated rates (per 1000) and lower/upper bounds of 95% confidence intervals
(755 records included in the analysis)

  +-------------------------------------------------------+
  | ageband    D        Y      Rate    Lower     Upper |
  |-------------------------------------------------------|
  |      30    0   0.0963    0.0000        .         . |
  |      40    6   0.9070    6.6152   2.9719   14.7246 |
  |      50   18   2.1070    8.5428   5.3823   13.5591 |
  |      60   22   1.4933   14.7325   9.7006   22.3744 |
  +-------------------------------------------------------+
```

In the output the rates are expressed in terms of events per 1000 person-years (due to the `per(1000)` option that was used and since the data had been `stset` with attained age in years). *Why is the rate 0 for the youngest ageband?*

`stsplit` is a very powerful command which is used heavily for survival analysis in Stata. Browse the help file for a full description of its functionality and additional examples.

3.2.8 Saving and plotting estimates

In survival analysis there is a lot of focus on estimating and plotting for example the survival and hazard functions. This can easily be done using Stata´s `sts graph` command. For example, to plot the Kaplan-Meier estimates of survival by energy intake we can use:

```
. sts graph, survival by(hieng)
```

However, sometimes we want to plot estimates of the survival and hazard (or any other quantity of interest) after having fitted, say, a regression model. We can then easily store the estimates in a new variable and create any graphs of interest manually. To demonstrate the principle behind this idea we will reproduce the Kaplan-Meier graph without using the `sts graph` command.

```
. sts generate survival = s, by(hieng)

. twoway    (line survival _t if hieng == 0, sort) ///
            (line survival _t if hieng == 1, sort) ///
            ,yscale(range(0 1)) xscale(range(0 80)) ///
            ylabel(0 0.25 0.50 0.75 1) title(Survival by hieng)
```

`sts generate` creates new variables containing the estimated survivor (failure) function, the Nelson-Aalen cumulative hazard (integrated hazard) function, and related functions. Here we simply store the survival estimates in a variable called survival. We then use this new variable to plot the Kaplan-Meier estimates against follow-up time (represented by _t).

After having fitted regression models the Stata function that can be used to store model estimates is called `predict` rather than `sts generate`.

Some useful commands

A, B are categorical variables. X, Y are metric variables.

Data Management

`use`	Read in a data set already in Stata format
`infile using`	Read in data in a txt file with names
`describe` (or F3)	Describe contents of data in memory
`list`	List values of variables
`drop A`	Drops the variable called `A`
`drop if ...`	Drops all records satisfying ...
`generate A =`	Creates a new variable called `A`
`replace A =`	Replaces contents of `A`
`recode A`	Recodes the variable called `A`
`save filename`	Save data set in Stata format
`sort A`	Sort records according to the variable `A`
`count if ...`	Count number of observations satisfying ...

Statistics and Graphics

`summarize Y`	Display summary statistics for `Y`
`tabulate A`	One-way table of frequencies for A (categorical)
`tabulate A B`	Two-way table of frequencies for `A` and `B`
`table A, c(mean X)`	Table of mean X by levels of A
`graph Y, hist`	Displays histogram of `Y`
`graph Y X, scatter`	Displays scatter plot of `Y` vs `X`
`hist A`	Histogram of the categorical variable `A`
`regress Y X`	Linear regression of Y on X
`predict P`	Obtain prediction after `regress` and put in `P`

Utilities

`clear`	Clear data from memory
`display 2+2`	Display the result of 2+2
`do filename`	Execute commands from `filename.do`
`exit`	Exit Stata
`exit, clear`	Clear and exit Stata
`help`	Obtain on-line help for both data and commands
`log using filename`	Write output to `filename.log`