Stata Technical Bulletin

November 1994

STB-22

A publication to promote communication among Stata users

Contents of this issue

an50 Submission guidelines

Sean Becketti, Stata Technical Bulletin, FAX 914-533-2902

In recent months, I have received an increasing number of requests for written copies of our submission guidelines. In order to reduce the demand for mailed copies, this insert reproduces our current guidelines. Please regard these instructions as guidelines and not as rules or restrictions. As our masthead states, the Stata Technical Bulletin exists "to promote communication among Stata users", and my goal as editor is to make the STB as responsive as possible to the needs and desires of Stata users. If you have questions about the suitability of a potential submission, please contact me directly.

Content of submissions

The STB welcomes diversity in submissions. We regularly publish scholarly papers that promote new statistics or explore the operating characteristics of existing estimators. We also publish inserts that present simple but useful utility programs. And we encourage and publish questions and communications from users. Some communications provide new data sets for research or teaching; some use Stata to review material of pedagogical interest; some simply report personal experiences using Stata in combination with other software or hardware. The guiding principle in choosing submissions for publication is their usefulness to Stata users. If a submission will interest or benefit a substantial number of readers, we will find space for it.

There are some general guidelines that indicate the types of submissions that are most appropriate for the STB. While we publish inserts that draw on all subject areas, we are more interested in inserts that emphasize the use of Stata than in ones that emphasize the subject matter. A lengthy empirical investigation of, say, the carcinogenic properties of industrial solvents may not be interesting to a large group of STB readers unless it emphasizes how Stata was (or was *not*) useful in completing the study. Many authors find it useful to write two versions of their papers. One version, highlighting their substantive findings, is submitted to the appropriate professional journal. The second version, highlighting the role of Stata in their research, is submitted to the STB.

There is an important exception to this guideline: papers focusing on statistics or statistical computing. An interest in these two topics is the common bond among STB readers; thus, these articles are always appropriate for the STB. For example, we published a long and fruitful series of inserts debating the performance of competing tests of normality. These inserts provided Stata programs to calculate the different test statistics, but, throughout, the focus was on the performance of the tests and not on their implementation in Stata.

I have noted a concern on the part of some authors that their submission may not be "sophisticated" enough for the STB. Let me put that concern to rest once and for all. While it is gratifying to publish inserts that report new and important research, our most popular inserts remain the more modest pieces that present interesting tips and/or simple utility programs that make Stata easier to use. As a consequence, most issues of the STB have devoted and will continue to devote a significant share of the pages to these more utilitarian inserts. If you have written a Stata program that you find useful, it is likely that other Stata users will find the program useful as well.

Format of submissions

Please send two copies of your submissions directly to the Editor, one hard copy and one copy on diskette. The diskette can be either DOS or Unix format; DOS is preferred. Submissions consist of the text of the insert and any ado-files, graphs, data sets, examples, or other supporting material.

The STB is published using TEX, a formatting program for mathematical documents. In TEX, documents are stored in ordinary ASCII files, without any control or special characters. To allow us to format your submission for publication, you must send us the text of your insert in an ordinary ASCII file. We can sometimes convert files stored in the more common word processor formats, but there is no guarantee we can do so. Feel free to prepare your submission using your favorite word processor, but include a plain ASCII version on the diskette you submit. And if you use TEX, limit your use of custom macros. The STB is formatted using only a handful of custom macros. To publish your insert, our production staff must remove any custom macros you use and replace them with standard TEX control sequences. Heavy use of custom macros may permit you to mimic the appearance of an STB insert, but it will only delay the publication of your submission.

Publishing the STB presents a typographical challenge because STB inserts combine significant amounts of mathematical, tabular, and graphical material with computer listings and syntax diagrams, which require their own special typographical treatment. Authors face a similar challenge in preparing submissions. The hard copy of your insert should document any special typographical effects needed. If your word processor produces the desired effect, include it directly in the hard copy of your insert. Otherwise, add a notation, either in your word processor or simply by writing on the hard copy, to indicate the needed effect. To avoid last-minute revisions of the proofs of your insert, please be as clear as possible in communicating these typographical instructions.

Include a help file for each ado-file in your submission. Help files are plain ASCII files with the same filename as the associated ado-file and with the extension `.hlp`. All help files should adhere to the standard Stata format. The easiest way to determine this format is to examine one of the help files delivered with Stata or one of the help files on the STB distribution diskette. The caret ("^") is used in help files to turn highlighting on and off.

Examples should be included with each ado-file, along with logs of their results, so we can confirm that the copies of the ado-files we receive perform as described in the insert. Any examples in the text should be supplied as do-files, along with any data used in the examples. These data sets are typically supplied on the distribution diskette to permit readers to replicate the examples. Avoid using confidential or proprietary data in your examples or any other data that cannot be supplied freely to STB readers.

If there are Stata graphs in your insert, include a do-file that recreates the graphs. You may also supply `.gph` files, but these are typically recreated for publication. If you must supply a `.gph` file that cannot be recreated, do not use the `title()` option of `graph` to title the graph. Instead, indicate the title of the graph, and we will add the title during the publication process.

Program design

You are, of course, free to design your Stata programs as you see fit. The point of the STB is to communicate your good ideas to other Stata users, particularly when these ideas are novel. However, there are a few simple guidelines of program design that will make your program easier for others to understand and to use.

First, adopt standard Stata syntax if at all possible. Stata's stripped-down syntax is one of its greatest strengths. Command names are usually simple English verbs that describe the action to be taken (`list`, `summarize`, `tabulate`). The other components of Stata syntax cover the contingencies: variable lists specify the objects of the action, the expression details any calculations needed, the `if` and `in` clauses restrict the sample, the weight clause specifies the weight, and the options handle all other contingencies. Stata users find it easier to learn new commands if the commands follow standard syntax. And Stata's `parse` command makes it easy for your program to rely on Stata's extensive parsing and error-checking code, as long as you adopt standard syntax.

Second, allow users to type the shortest unique abbreviations of option names. The `parse` command allows you to specify the shortest acceptable abbreviation for each option. Don't make users type `display` if there are no other options that begin with the letter "d". Destructive options (`clear`, `replace`, etc.) are exceptions to this guideline. Stata style requires such options to be typed in full to avoid unintended modifications of users' data.

Third, as with program names, use ordinary English words for option names, whenever possible. And design your program to accept complete words, if the user chooses to type them. For example, don't require users to type '`disp`' without also letting them type '`display`' if they wish.

The goal throughout is to write your programs as clearly and understandably as you do your prose. Aim for clarity over cleverness.

dm21 Bringing large data sets into memory

Robert M. Farmer, Alabama Quality Assurance Foundation Inc., 205-970-1600

One characteristic of Stata that frequently frustrates new users is the way Stata allocates memory. Stata sets aside a "rectangle" of memory for data. The size of this rectangle defines the maximum numbers of observations (`maxobs`) and variables (`maxvar`) allowed. The "area" of the rectangle is the ultimate limit on the size of the data set that can be handled, but, at any given moment, data sets must also meet the `maxobs` and `maxvar` constraints. If a data set is small enough to fit, but has either too many observations or variables for the current data rectangle, the Stata `use` command will fail to load the data set. For example:

```
. describe

Contains data
  Obs:     0 (max= 20434)
 Vars:     0 (max=    99)
Width:     0 (max=   200)
Sorted by:

. describe using test
```

```
Contains data                               Example large data set
  Obs: 32200
 Vars:     3
Width:    12
  1. x            float  %9.0g
  2. y            float  %9.0g
  3. group        float  %9.0g              Grouping variable
Sorted by:

. use test
no room to add more observations
r(901);
```

If there is actually enough memory to handle the data set, you can convince Stata to `use` the data by (1) clearing any data currently in memory, (2) using the `set maxvar` or `set maxobs` command to resize the data rectangle, and (3) bringing the data set into memory. For example:

```
. drop _all

. set maxobs 32300

. use test
(Example large data set)
```

This process is annoying for the experienced user and baffling for the novice. Stata makes the process a little easier by supplying the `memsize` and `bmemsize` commands for regular and Intercooled Stata, respectively. (See [4] memory.) These commands analyze the data set and make educated guesses about the best way to resize the data rectangle. In addition, they load macros for performing the resizing into the function keys *F4*, *F5*, and *F6*. For example, instead of resetting the maximum number of observations as in the example above, we could

```
. bmemsize using test

       variables:                 3
       width:                     12
       observations:              32200
       data set size:             377 k

       data will not fit in current partition

                                   approx.   free float    free
       command                     maxobs    variables      obs
       ---------------------------------------------------------
 (F4)  set maxvar 30 width 39      104717          6      72517
 (F5)  set maxvar 116 width 125     32672         28        472
 (F6)  set maxvar 23 width 32      127625          5      95425
```

At this point, pressing *F4* would issue the command `set maxvar 30 width 39`, pressing *F5* would issue the command `set maxvar 116 width 125`, and pressing *F6* would issue the command `set maxvar 23 width 32`. The first option (*F4*) attempts to balance the number of observations and variables that can be added after the data rectangle is resized. The second option (*F5*) attempts to maximize the number of variables that can be added. The third option (*F6*) attempts to maximize the number of observations that can be added.

While `memsize` and `bmemsize` are helpful, they do not go far enough, particularly for the novice Stata user. `bringin` is my attempt to completely automate the process of loading a new data set, regardless of shape or size. The syntax is

$$\texttt{bringin}\ \textit{filename}\ \Big[\Big\{\ \texttt{vars}\ \Big|\ \texttt{obs}\ \Big\}\Big]$$

`bringin` is actually a "wrapper" for the `bmemsize` command. `bringin` silently runs `bmemsize`, executes one of the three options, then `use`s the specified data set. For instance:

```
. bringin test
(Example large data set)

. describe

Contains data from test.dta
  Obs: 32200 (max= 94750)                   Example large data set
 Vars:     3 (max=    29)
Width:    12 (max=    40)
  1. x            float  %9.0g
  2. y            float  %9.0g
  3. group        float  %9.0g              Grouping variable
Sorted by:
```

The basic command, '`bringin` *filename*', chooses the first (*F4*) `bmemsize` option, that is, it attempts to balance the number of variables and observations that can be added after the data rectangle is resized and *filename*`.dta` is loaded. The optional keyword `vars` selects the second (*F5*) option, which maximizes the number of variables that can be added. The optional keyword `obs` selects the third (*F6*) option, which maximizes the number of observations that can be added.

`bringin` may be most useful in do-files that are designed to be run by others, particularly when the others are not experienced Stata users. To make it easy to modify your existing do-files, `bringin` ignores any commas and options. As a consequence, you can safely globally replace `use` with `bringin` in your do-files. For example, changing the line

```
. use myfile, clear
```

to

```
. bringin myfile, clear
```

will cause no problems. `bringin` assumes the `clear` option, so ignoring the typed option does not change the meaning of the command.

`bringin` uses the `bmemsize` command, hence it can only be used with Intercooled Stata. If you are not running Intercooled Stata, change `bmemsize` to `memsize` in `bringin.ado`, and the program will work as described.

dm22 Sorting in descending order

David Mabb, Health Services Advisory Group Inc., FAX 602-241-0757

Stata's `sort` command supports sorting only in ascending order. Often it is beneficial to observe data in descending order. `sortd` provides a way to sort variables in descending order. The syntax of `sortd` is

`sortd` *varlist* [`in` *range*]

`sortd` offers the same features as Stata's internal `sort` command. In fact, the sorting is performed by the `sort` command, making `sortd` very fast. Aside from reversing the order of the sort, there is one difference between `sort` and `sortd`: the data set is not marked as sorted when `sortd` is used. As a consequence, the `by` prefix and similar constructions cannot be used following the `sortd` command.

Example

```
. use \stata\census, clear
(1980 Census data by state)

. sort pop

. list state pop in f/5

             state        pop
  1.        Alaska     401851
  2.       Wyoming     469557
  3.       Vermont     511456
  4.      Delaware     594338
  5.     N. Dakota     652717

. list state pop in -5/l

             state        pop
 46.      Illinois   11426518
 47.  Pennsylvania   11863895
 48.         Texas   14229191
 49.      New York   17558072
 50.    California   23667902

. sortd pop

. list state pop in f/5

             state        pop
  1.    California   23667902
  2.      New York   17558072
  3.         Texas   14229191
  4.  Pennsylvania   11863895
  5.      Illinois   11426518
```

```
. list state pop in -5/l
             state        pop
 46.      N. Dakota     652717
 47.       Delaware     594338
 48.        Vermont     511456
 49.        Wyoming     469557
 50.         Alaska     401851
```

dm23 Saving a subset of the current data set

David Mabb, Health Services Advisory Group, FAX 602-241-0757

savin is a utility that extends the save command by allowing you to save a subset of the current data set. The syntax of savin is

savin [*varlist*] [if *exp*] [in *range*] using *filename* [, nolabel replace]

Like Stata's save command, savin leaves the current data set undisturbed. The nolabel and replace options also work just as in Stata's save command. Unlike the save command, savin requires you to type the using keyword before the filename.

Discussion

I frequently need to make different data subsets from a primary Stata data file. The process usually involves using the primary file, keeping just the information I want, saving the subset with a new name, and then retrieving the primary file again. Typically, this process is performed in do-files as follows

```
. use main
(Primary data set)
. keep mpn age sex
. keep if sex==1
(27 observations deleted)
. save male
file male.dta saved
. use main
(Primary data set)
. keep mpn age sex
. keep if sex==2
(23 observations deleted)
. save female
file female.dta saved
. use main
(Primary data set)
```

With savin, this process is simplified to

```
. use main
(Primary data set)
. savin mpn age sex if sex==1 using male
file male.dta saved
. savin mpn age sex if sex==2 using female
file female.dta saved
```

At the end of this sequence of commands, the primary data set (main.dta) is still the current data set.

dt2 Reading public use microdata samples into Stata

Charles L. Sigmund, M.S., Oregon Employment Department, EMAIL sigmundc@ucs.orst.edu
D. H. Judson, Ph.D., University of Nevada, Reno, EMAIL djudson@scs.unv.edu

The Public Use Microdata Sample-A (PUMS-A, hereafter simply PUMS) of the U.S. Census is a data set containing a 5 percent sample of responses to the long-form census questionnaires for each state. These data are available for each state on CD-ROM from the U.S. Census Bureau at minimal cost. (Call Customer Services, 301-763-4100, for more information on obtaining these CDs.)

The purpose of the PUMS file is to provide researchers with direct access to household-by-household and person-by-person data. Individual household and person data are not available in Summary Tape Files, the other major Census product. Individual Census responses are confidential; thus, these data have been statistically modified to protect the confidentiality of individuals. They are designed, however, to provide unbiased estimates and to maintain the covariance structure among variables to the extent possible.

The data are divided into two file types: household records and person records. Data in the person records include demographic, socio-economic, family, education, and employment characteristics. Household records include such things as mortgage or rent payment, size and type of dwelling, number and age of all residents in the household, location of the household, and relationships among household members. The combined file for each state contains over 500 variables. For the state of Oregon—to choose an example with which we are familiar—more than 140,000 people are represented.

The ability to read PUMS data into Stata generates an unlimited number of potential uses. The data contain information useful in almost any field, from advertising to demography. By selecting only the variables that are of interest, the researcher can optimize memory use and eliminate superfluous information. However, due to the size of the file, it is still recommended that Intercooled Stata be used whenever possible.

Included on the distribution diskette are two versions of a Stata dictionary we created to read 1990 PUMS data into Stata. One of the dictionaries, `pumsh1.dct`, is used to read household data into Stata. The other dictionary, `pumsp1.dct`, is used to read person data. In `pumsh1.dct`, person-level variables are commented out in the dictionary header. The reverse is true in `pumsp1.dct`. On our system, the PUMS data file is stored in the subdirectory `d:\pums`. You will need to modify the top line of each dictionary to provide the file location of your PUMS data file. PUMS data files have names of the form `pumsa`*xx*`.txt`, where $xx =$ the state initials. For example, the file for Oregon is `pumsaxor.txt`.

Because the file is divided into two record types and because each record is set up with a hierarchical structure in which each person record is subordinate to the associated household record, it is necessary to read the household data in separately from the person data. The file structure is

Record Type	**Serial Number**	**Data**
H	HH serial number	Household characteristics
P	HH serial number	Person 1's characteristics
P	HH serial number	Person 2's characteristics

and so on for each household.

We reproduce a portion of the dictionary here:

```
dictionary using d:\pums\pumsaxor.txt
*
* HOUSEHOLD RECORDS
*
* _column(1) str1 rectype %1s
* _column(2) long SerialNo %7f
* _column(9) byte Sample %1f
* _column(10) byte division %1f
* _column(11) byte state %2f
* _column(13) long puma %5f
* _column(18) byte areatype %2f
* _column(20) int msapmsa %4f
* _column(24) int psa %3f
* _column(27) int subsmpl %2f
* _column(29) int houswgt %4f
* _column(33) byte persons %2f
* _column(35) byte gqtype %1f
* _column(39) byte units1 %2f
* _column(41) byte husflag %1f
* _column(42) byte pdsflag %1f
* _column(43) byte rooms %1f
* _column(44) byte tenure %1f
* _column(45) byte acreage %1f
* _column(46) byte commuse %1f
...
Household variables continue
...
* _column(203) byte amoblhme %1f
*
*  PERSON RECORDS
*
* _column(9) byte relat1 %2f
* _column(11) byte sex %1f
* _column(12) int race %3f
* _column(15) byte age %2f
* _column(17) byte marital %1f
* _column(18) int pwgt1 %4f
* _column(26) int remplpar %3f
* _column(29) byte rpob %2f
* _column(31) byte rspouse %1f
* _column(32) byte rownchld %1f
* _column(33) byte ragechld %1f
* _column(34) byte rrlchld2 %1f
* _column(35) byte relat2 %1f
* _column(36) byte subfam2 %1f
* _column(37) byte subfam1 %1f
* _column(38) int hispanic %3f
...
Person variables continue
...
* _column(230) byte aincome7 %1f
* _column(231) byte aincome8 %1f
end
```

As we noted above, to read in the variables for households, first comment out all of the person variables (the second group) and any unwanted household variables. When reading in the data, use '`if rectype=="H"`' to read only household variables and '`if rectype=="P"`' to read only person variables. To allow all of the variables to be read in, use the `set maxvar` command to expand the memory space allocated to variables. The following commands sketch the steps for reading the household data.

```
. clear
. * Reduce maxvar to make space for more observations.
. set maxvar 80
. * Bring the household data into Stata using the infile command.
. infile using pums.dct if rectype=="H"
. * Sort the file by the variable SerialNo.
. sort SerialNo
. * After this has been completed, save the household data.
. save hh.dta
```

Repeat this procedure for the person data, commenting out all of the household variables and unwanted person variables. Again, you must also read in `rectype` in order to distinguish between the record types.

```
. * Bring the person data into Stata using the infile command.
. infile using pums.dct if rectype=="P"
. * Sort the file by the variable SerialNo.
. sort SerialNo
. * Save the person data.
. save p.dta
```

To merge the two files, you must read in the variable `SerialNo`. This is a unique identifier that links all of the person records with their associated household record. If you do not wish to merge the person and household records, you do not need to read `SerialNo`.

Finally to link the two files, use the `merge` command.

```
. use hh.dta
. merge SerialNo using p.dta
```

This sequence of commands will create one complete file which includes all of the variables you have specified. Tabulate `_merge` to ensure that all records have been properly merged and the data are ready to be used. A `_merge` value of 1 or 2 does not necessarily mean an incorrect merge; for example, in the Oregon file, approximately 6,000 household records have no corresponding person records (and contain virtually all missing values). These records probably correspond to individuals or households who refused to respond to the census taker. In these cases, the census taker is instructed to obtain as much information about the household as possible from neighbors or by observation.

In order to make your analyses applicable at the state level, you will need to use the weighting variable provided in the PUMS file. For analyses based on persons, use the weighting variable `pwgt`. For analyses based on households, use the weighting variable `houswgt`.

We have created an example file from Oregon's PUMS file called `testor.dta`. This example file is included on the distribution diskette. This file contains 1474 records of persons with their associated household characteristics. The following log file documents how we created `testor.dta` and displays some basic tabulations.

```
. * This section describes how to read in the data, and
. * how to combine the household and person records.
. *
. * Reduce maxvar to make room for all the observations.
. set maxvar 80
. * first read in the household data
.  quietly infile using pumsh1.dct if rectype=="H"
. * sort the data so it can be merged later
. sort SerialNo
. save hh1
file d:\pums\hh1.dta saved
. drop _all
. * now read in the person data
. quietly infile using pumsp1.dct if rectype=="P"
. * again, sort it so it can be merged
. sort SerialNo
. save pers1
file d:\pums\pers1.dta saved
. * merge the data using the unique identifier "SerialNo"
. merge SerialNo using hh1
. * check the merge to verify success
. tabulate  _merge
      _merge|      Freq.     Percent        Cum.
------------+-----------------------------------
          2 |       5904        4.02        4.02
          3 |     140984       95.98      100.00
------------+-----------------------------------
      Total |     146888      100.00
. drop if uniform()>.01
(145414 observations deleted)
. * create a test file which contains 1% of the cases
. save testor.dta
file d:\pums\testor.dta saved
. describe
Contains data from d:\pums\test.dta
  Obs:  1474 (max=184695)
 Vars:    42 (max=    80)
Width:    91 (max=   162)
  1. rectype      str1   %9s
  2. SerialNo     long   %10.0g
  3. relat1       byte   %8.0g
  4. sex          byte   %8.0g
  5. race         int    %8.0g
  6. age          byte   %8.0g
  7. marital      byte   %8.0g
  8. pwgt1        int    %8.0g
  9. hispanic     int    %8.0g
 10. poverty      int    %8.0g
 11. rlabor       byte   %8.0g
 12. industry     int    %8.0g
 13. occup        int    %8.0g
 14. class        byte   %8.0g
 15. work89       byte   %8.0g
 16. week89       byte   %8.0g
 17. hour89       byte   %8.0g
 18. rearning     long   %10.0g
 19. rpincome     long   %10.0g
 20. income1      long   %10.0g
 21. income2      long   %10.0g
 22. income3      long   %10.0g
 23. income4      long   %10.0g
 24. income5      long   %10.0g
 25. income6      long   %10.0g
 26. income7      long   %10.0g
 27. income8      long   %10.0g
 28. Sample       byte   %8.0g
 29. division     byte   %8.0g
 30. state        byte   %8.0g
 31. puma         long   %10.0g
 32. areatype     byte   %8.0g
 33. houswgt      int    %8.0g
 34. persons      byte   %8.0g
 35. rooms        byte   %8.0g
 36. acreage      byte   %8.0g
 37. rfarm        byte   %8.0g
 38. rfaminc      long   %10.0g
 39. rhhinc       long   %10.0g
 40. rwrkr89      byte   %8.0g
 41. fampers      byte   %8.0g
 42. _merge       byte   %8.0g
Sorted by:
Note:  Data has changed since last save



. * now some basic tabulations
. tabulate sex
        sex|      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        730       51.66       51.66
          1 |        683       48.34      100.00
------------+-----------------------------------
      Total |       1413      100.00

. tabulate sex [weight=pwgt]
(frequency weights assumed)
        sex|      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |      14515       51.28       51.28
          1 |      13793       48.72      100.00
------------+-----------------------------------
      Total |      28308      100.00

. * where:
. * 0=male
. * 1=female
. *
. tabulate marital
    marital|      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        687       48.62       48.62
          1 |         54        3.82       52.44
          2 |        114        8.07       60.51
          3 |         23        1.63       62.14
          4 |        535       37.86      100.00
------------+-----------------------------------
      Total |       1413      100.00

. tabulate marital [weight=pwgt]
(frequency weights assumed)
    marital|      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |      13376       47.25       47.25
          1 |       1110        3.92       51.17
          2 |       2370        8.37       59.55
          3 |        429        1.52       61.06
          4 |      11023       38.94      100.00
------------+-----------------------------------
      Total |      28308      100.00

. * where:
. * 0 = Now married, except separated
. * 1 = Widowed
. * 2 = Divorced
. * 3 = Separated
. * 4 = Never married, or under 15 years old
. *
. generate inpov=(poverty<100)

. * inpov = 1 if person is living at less than 100% of US poverty level
. tabulate inpov marital [weight=pwgt], col
(frequency weights assumed)
```

```
           | marital
      inpov|         0          1          2          3          4 |     Total
-----------+-------------------------------------------------------+----------
         0 |     12630        875       1925        322       8960 |     24712
           |     94.42      78.83      81.22      75.06      81.28 |     87.30
-----------+-------------------------------------------------------+----------
         1 |       746        235        445        107       2063 |      3596
           |      5.58      21.17      18.78      24.94      18.72 |     12.70
-----------+-------------------------------------------------------+----------
      Total|     13376       1110       2370        429      11023 |     28308
           |    100.00     100.00     100.00     100.00     100.00 |    100.00

. tabulate race [weight=pwgt]
(frequency weights assumed)
       race|      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |      26059       92.06       92.06
          2 |        530        1.87       93.93
          4 |         19        0.07       93.99
          6 |        209        0.74       94.73
          8 |         77        0.27       95.00
          9 |         93        0.33       95.33
         10 |         28        0.10       95.43
         11 |        226        0.80       96.23
         12 |         54        0.19       96.42
         14 |         19        0.07       96.49
         15 |         16        0.06       96.55
         16 |         51        0.18       96.73
         23 |         38        0.13       96.86
         37 |        516        1.82       98.68
        302 |         10        0.04       98.72
        304 |         13        0.05       98.76
        307 |          6        0.02       98.78
        308 |          6        0.02       98.81
        312 |         16        0.06       98.86
        315 |          3        0.01       98.87
        316 |         29        0.10       98.98
        317 |         22        0.08       99.05
        320 |          7        0.02       99.08
        322 |         12        0.04       99.12
        323 |         57        0.20       99.32
        326 |        131        0.46       99.78
        327 |         61        0.22      100.00
------------+-----------------------------------
      Total |      28308      100.00
. * where:
. * 001 White    002 Black    004 Eskimo    005 Aleut
. * ...
. * others are as defined in the  PUMS documentation.
```

As you can see, the PUMS data provide a rich source of socioeconomic data that can readily be analyzed using Stata. We encourage you to explore these data.

sg26.1 Fractional polynomials: Correction

Patrick Royston, Royal Postgraduate Medical School, London, FAX (011)-44-81-740-3119
Douglas G. Altman, Imperial Cancer Research Fund, London, FAX (011)-44-71-269-3429

Three errors in `fp.ado` have come to light since its publication in STB-21 (Royston and Altman 1994). These are corrected in the present release. The errors are as follows.

1. The `comparison` and `estimates` options were inoperative when used at the model-fitting stage (they did work in 'replay' mode).
2. Replay of models containing `basevars` did not display regression coefficients for the `basevars` as it should.
3. With the `estimates` option, the deviance of the best-fitting fractional polynomial (FP) model should be displayed at the foot of the regression output. In fact, the deviance of the base model (that is, without FP terms) was given instead.

Reference

Royston, P. and D. G. Altman. 1994. sg26: Using fractional polynomials to model curved regression relationships. *Stata Technical Bulletin* 21: 11–23.

sg27 The overlapping coefficient and an "improved" rank-sum statistic

Richard Goldstein, Qualitas, Inc., EMAIL richgold@netcom.com

We all know there is a difference between statistical and substantive significance; the rejection of a null hypothesis does not necessarily imply that something meaningful has been found. Yet statistical packages, including Stata, offer many tests of statistical significance and few, if any, indicators of substantive significance. Here we offer two simple measures, the overlapping coefficient and a transformation of the rank-sum statistic, that can help users assess the importance of statistically significant findings.

The overlapping coefficient is a measure of the agreement between two distributions (or "the area under two probability (density) functions simultaneously" (Bradley 1985, 546)). It is equal to one minus the dissimilarity coefficient, a measure widely used by social scientists in the study of segregation.

Consider the simplest case of two normally distributed variables, $x \sim N(\mu_x, \sigma^2)$ and $y \sim N(\mu_y, \sigma^2)$. When the variances of the variables being compared are equal, as they are in this case, the overlapping coefficient, o, is

$$o = 2 * \Phi(-|\mu_x - \mu_y|/2\sigma)$$

where $\Phi(\cdot)$ is the standard normal distribution function. If $\mu_x = \mu_y$, the distributions of x and y agree completely and

$$o = 2 * \Phi(0) = 1$$

The overlapping coefficient is smaller the further apart are μ_x and μ_y, and

$$o \to 0 \quad \text{as} \quad |\mu_x - \mu_y| \to \infty$$

The overlapping coefficient can be generalized to the case where x and y have different variances (Inman and Bradley 1989). Asymptotic arguments justify the use of the overlapping coefficient with a wide variety of non-normally distributed variables.

From the formula above, it is clear that the overlapping coefficient is a measure of the closeness of the location of two distributions. The correlation of x and y has no influence on the overlapping coefficient. Uncorrelated variables with identical means overlap perfectly. On the other hand, if $y \equiv x + \delta$, then o goes to zero for sufficiently large values of δ.

The overlapping coefficient can be used, for instance, to help determine whether a statistically significant t statistic is important in practical terms. It is also closely related to "the misclassification probability in the two population classification problem" (Inman and Bradley 1989, 3868). Because I use the statistic only for these purposes, I do not provide measures of the variance of the statistic.

Formulas used in calculating the overlapping coefficient o come from Inman and Bradley (1989). The calculations are different depending on whether the variances are equal. If you provide just the names of the variables, both measures are presented. The "OVL is invariant when a suitable common transformation is made to both variables" (Inman and Bradley 1989, 3852).

As pointed out in Gastwirth (1975), there is a potential problem with the overlapping coefficient: if changes take place only on one side of the point(s) of intersection of the two distributions, the overlapping coefficient will not reflect these. However, for the purpose of helping decide whether a t statistic is meaningful this is of little relevance (though it is very relevant for other purposes).

`overlap` displays the overlapping coefficient. There are two syntaxes:

```
overlap var1 var2 [ if exp ] [ in range ]

overlap var1 [ if exp ] [ in range ] , by(var2)
```

In the first syntax, *var1* and *var2* are continuous variables. The second syntax calculates the overlapping coefficient for the continuous variable *var1* across the two groups defined by *var2*.

`overlapi` is an immediate version of `overlap`. For the case of equal variances, the syntax is

`overlapi` *mean1 mean2 sd*

where *sd* is the common standard deviation of the two variables. For the case of unequal variances, the syntax is

`overlapi` *mean1 mean2 sd1 sd2* $(sd1 \neq sd2)$

For the general case, the syntax is

`overlapi` *mean1 mean2 sd1 sd2 n1 n2*

where *n1* and *n2* are the numbers of observations used to estimate the means and standard deviations of the two variables. This syntax displays both the equal and unequal variance versions of the overlapping coefficient.

Example

As an example, we compare the distributions of fuel efficiency of foreign and domestic cars in the automobile data set supplied with Stata. We begin by calculating the t test of the null hypothesis that the average miles per gallon is the same for both sets of cars.

```
. use \stata\auto, clear
(1978 Automobile Data)

. ttest mpg, by(foreign)
  Variable |      Obs        Mean   Std. Dev.
  ---------+---------------------------------
         0 |       52    19.82692    4.743297
         1 |       22    24.77273    6.611187
  ---------+---------------------------------
  combined |       74     21.2973    5.785503
           Ho:  mean(x) = mean(y)  (assuming equal variances)
                      t = -3.63 with 72 d.f.
               Pr > |t| = 0.0005
```

The t test reveals a highly significant difference between the average mileage for foreign cars (24.8 MPG) and the average mileage for domestic cars (19.8 MPG). However, the variance of MPG seems a bit higher for the foreign cars. Since the default t test assumes equal variances across groups, we investigate further to see whether this assumption is justified.

```
. sdtest mpg, by(foreign)
  Variable |      Obs        Mean   Std. Dev.
  ---------+---------------------------------
         0 |       52    19.82692    4.743297
         1 |       22    24.77273    6.611187
  ---------+---------------------------------
  combined |       74           .     5.35582
           Ho:      sd(x) = sd(y)    (two-tailed test)
                 F(21,51) = 1.94
               2*(Pr > F) = 0.0549
```

The evidence is mixed: the p value of the test statistic is 5.5 percent. Since we are not sure whether the variances are equal, we calculate the unequal variance version of the t test, to see if the apparent significance of the mileage difference is sensitive to the assumption of equal variances.

```
. ttest mpg, by(foreign) unequal
  Variable |      Obs        Mean   Std. Dev.
  ---------+---------------------------------
         0 |       52    19.82692    4.743297
         1 |       22    24.77273    6.611187
  ---------+---------------------------------
  combined |       74     21.2973
           Ho:  mean(x) = mean(y)  (assuming unequal variances)
                      t = -3.18 with 31 d.f.
               Pr > |t| = 0.0033
```

The 5 MPG difference is still highly significant. At this point, we might be tempted to halt the analysis and to rely on the t test to support the hypothesis that foreign cars get better mileage than domestic cars. Instead, let's calculate the overlapping coefficient to see whether the statistically significant difference detected by the t test corresponds to a meaningful difference in the distributions of MPG between groups.

```
. overlap mpg, by(foreign)
MLE of overlap:
  Variances equal:   0.9313     Dissimilarity Index=1-OVL: 0.0687
  Variances UNequal: 0.6421     Dissimilarity Index=1-OVL: 0.3579
```

When the variance is assumed to be the same in both groups, the overlapping coefficient is .93, indicating substantial overlap of the two distributions and raising questions about the interpretation of the t test for these data. Note, that for these data, the overlapping coefficient is sensitive to the assumption of equal variances. When the variances are allowed to differ, the overlapping coefficient drops to .64, indicating substantially less overlap than the initial calculation.

The immediate version of the program, `overlapi`, produces the same results as `overlap`. The immediate version makes it easy to calculate the overlapping coefficient when the data are not readily available, for instance, when only summary statistics are reported in a book or research paper.

```
. overlapi 19.82692 24.77273 4.743297 6.61187 52 22
MLE of overlap:
  Variances equal:   0.9313     Dissimilarity Index=1-OVL: 0.0687
  Variances UNequal: 0.6421     Dissimilarity Index=1-OVL: 0.3579
```

An "improved" rank-sum statistic

The Mann–Whitney–Wilcoxon two-sample rank-sum test provides a nonparametric alternative to the t test (signrank [5s]; Moses, Emerson, and Hosseini 1992; Fleiss 1981). There are a variety of equivalent ways of stating the test, but the intuition behind the test is straightforward. A characteristic is measured in two independent samples; in our example above, we've measured fuel efficiency in a sample of domestic cars and a sample of foreign cars. The null hypothesis states there is no systematic difference in the characteristic between the two samples. The rank-sum test merges the two samples and sorts them in order of the characteristic. Under the null hypothesis, the expected sum of the ranks of the observations from the first sample is equal to the expected sum of the ranks of the observations from the second sample, corrected for any imbalance in sample sizes. The Mann–Whitney U statistic is a function of the sum of the ranks with a known distribution under the null hypothesis. A U statistic can be calculated for either sample, and it can be show that

$$U_1 + U_2 = mn$$

where U_i is the U statistic for the ith sample and m and n are the numbers of observations in the two samples. By convention, the smaller of U_1 and U_2 is the test statistic.

Stata's `ranksum` command reports the p value of the rank-sum test. As with the t test, `ranksum` provides a measure of the statistical significance but not the substantive importance of the difference in the locations of the distributions of the characteristic between the two samples.

`ranksum2` augments Stata's `ranksum` command by also reporting U/mn. From the formula above, it is clear the $U/mn = 1/2$ under the null hypothesis. By construction, $0 \leq U/mn \leq 1/2$. Thus, this measure provides an intuitive measure of how much the data deviate from the null hypothesis.

We use the automobile data again to demonstrate `ranksum2`:

```
. use \stata\auto, clear
(1978 Automobile Data)

. ranksum2 mpg, by(foreign)

Test: Equality of medians (Two-Sample Wilcoxon Rank-Sum)

 Sum of Ranks: 1086.5 (foreign == 1)
 Expected Sum: 825

 z-statistic  3.09
 Prob > |z|   0.0020

 U/mn         .27141608
```

Every line but the last is identical to that produced by Stata's `ranksum` command. The last line reports the ratio of the U statistic to the product of the numbers of observations in each of the two samples. `ranksum2` also augments the stored results supplied by `ranksum`: `S_5` contains the Mann–Whitney U statistic and `S_6` contains the ratio U/mn.

Consider "inverting" the model implied by the tests in the example above. In other words, instead of explaining the mean fuel efficiency by noting whether a car is domestic or foreign, consider predicting whether a car is an import by measuring its fuel efficiency. Since the dependent variable is qualitative (domestic/import), a logistic model is a natural framework for this prediction exercise. It turns out that $U/mn = 1 - \text{ROC}$ where ROC is the area under the ROC curve for the logistic model. Thus:

```
. logistic foreign mpg

Logit Estimates                                       Number of obs =     74
                                                      chi2(1)       =  11.49
                                                      Prob > chi2   = 0.0007
Log Likelihood =  -39.28864                           Pseudo R2     = 0.1276

------------------------------------------------------------------------------
 foreign | Odds Ratio   Std. Err.       z     P>|z|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
     mpg |   1.173232   .0616972      3.038   0.002       1.058331    1.300608
------------------------------------------------------------------------------

. lroc, nograph

Logistic estimates for foreign
Area under ROC curve = 0.7286
```

References

Bradley, E. L. 1985. Overlapping coefficient. In *Encyclopedia of Statistical Sciences*, ed. S. Kotz and N. L. Johnson, vol. 6, 546–547. New York: Wiley.

Fleiss, J. L. 1981. *Statistical Methods for Rates and Proportions*. 2d ed. New York: Wiley.

Gastwirth, J. L. 1975. Statistical measures of earnings differentials. *The American Statistician* 29: 32–35.

Inman, H. F. and E. L. Bradley, Jr. 1989. The overlapping coefficient as a measure of agreement between two probability distributions and point estimation of the overlap of two normal densities. *Communications in Statistics—Theory and Methodology* 18: 3851–3874.

Moses, L. E., J. D. Emerson, and H. Hosseini. 1992. Analyzing data from ordered categories. In *Medical Uses of Statistics*, 2d ed., ed. J. C. Bailar III and F. Mosteller, 259–279. Boston: NEJM Books.

sg28 | Multiple comparisons of categories after regression-like methods

William H. Rogers, Stata Corporation, FAX 409-696-4601

In a typical experiment or survey setting, we compare the responses of two or more groups. If we estimate a parametric model, the covariance matrix of the parameters supplies us with standard error estimates for any individual parameter or contrast. The theory of hypothesis testing provides ways of using these estimated standard errors to calculate tests of hypotheses about the responses of different groups. For example, we might test whether crop yield is affected by the application of various fertilizers. These tests are well known and are provided by virtually every statistical package. Ambiguities arise, however, when we make multiple comparisons; that is, when we test more than one hypothesis about a model.

We can illustrate this problem using the automobile data set provided with Stata. This data set contains a variable, `rep78`, that records the repair record in 1978 of each car. `rep78` is coded as '`1`' for cars with poor repair records, as '`2`' for cars with fair repair records, and so on up to '`5`' for cars with excellent repair records. For the sake of the example, we treat `rep78` as a categorical variable rather than as an ordinal variable.

An interesting question is whether the price of a car depends on its repair record. One way to answer this question is to estimate a regression for price where the repair record is an explanatory variable. Since the repair record is a categorical variable, we cannot enter it directly as a regressor. Instead, we use the `tabulate` command to create indicator or dummy variables, one for each level of `rep78`. All but one of these dummies are entered in the price regression. (The set of all five dummies is collinear with the constant term in the regression; either the constant or one of the dummies must be dropped.) In this parameterization, the coefficient on each dummy variable estimates the difference in the average price between the indicated level of `rep78` and the level corresponding to the omitted dummy variable.

```
. use \stata\auto
(1978 Automobile Data)

. tabulate rep78, generate(r)
```

```
      Repair|
 Record 1978|      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        2.90        2.90
          2 |          8       11.59       14.49
          3 |         30       43.48       57.97
          4 |         18       26.09       84.06
          5 |         11       15.94      100.00
------------+-----------------------------------
      Total |         69      100.00
. regress price r2 r3 r4 r5 length displ weight mpg
  Source |       SS       df       MS                  Number of obs =      69
---------+------------------------------               F(  8,    60) =    6.47
   Model |  267197833     8  33399729.2               Prob > F      =  0.0000
Residual |  309599125    60  5159985.42               R-square      =  0.4632
---------+------------------------------               Adj R-square  =  0.3917
   Total |  576796959    68  8482308.22               Root MSE      =  2271.6
------------------------------------------------------------------------------
   price |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
      r2 |   907.3499   1817.764      0.499   0.619      -2728.719    4543.419
      r3 |   1105.359   1668.122      0.663   0.510      -2231.381    4442.099
      r4 |   2147.658   1702.115      1.262   0.212       -1257.08    5552.395
      r5 |   3816.672    1787.51      2.135   0.037       241.1194    7392.226
  length |  -117.3064   40.65207     -2.886   0.005      -198.6226   -35.99012
   displ |   8.447532   8.423298      1.003   0.320      -8.401571    25.29664
  weight |   4.089227   1.597143      2.560   0.013       .8944658    7.283989
     mpg |  -129.2005   84.52707     -1.529   0.132      -298.2799    39.87876
   _cons |   15158.53   6179.409      2.453   0.017       2797.871    27519.19
------------------------------------------------------------------------------
```

According to these estimates, cars with fair repair records cost an average of $907 more than cars with poor repair records. The gap increases with each improvement in repair record. Cars with excellent repair records cost an average of $3,817 more than cars with poor repair records.

The question may now arise: Which pairs of groups (categories of `rep78`) can we legitimately claim are different from each other; which of these differences are unlikely to have arisen by chance? The answer hinges on what we view as "legitimate".

The aggressive investigator might argue that groups 1 and 5 are different on the strength of the t statistic for the coefficient on `r5` ($t = 2.135$, with a p value of .037). The cautious investigator (or journal editor), however, would counter that many comparisons of the different groups could have been made. Perhaps this test was selected for focus solely because it happens to show a "significant" difference. And when multiple comparisons are made, the probability under the null of finding, say, a t statistic as large as 2.135 is greater than .037. But how much greater is it—that is, what is the correct p value for this t statistic when multiple comparisons are made?

There are many philosophical views on this problem. I examine the mechanics of one view—traditional adjustment for multiple comparisons—in the context of regression-like models. (See [5s] oneway for a discussion of this approach in an ANOVA context.) This view provides methods for making each test more conservative when there are multiple comparisons, so the overall probability of making a Type I error for any pairwise comparison (declaring a difference significant when it is merely due to chance) remains less than a predetermined value, such as 5 percent. We discuss three widely used approaches: the Bonferroni, Šidák, and Scheffé tests.

The Bonferroni test is the simplest to implement. In this method, the cautious investigator would note that 10 pairs of groups could have been compared and treat a reported p value of 0.037 as if it were $10 \times 0.037 = 0.37$. It would take a t value of 2.9146 to be "significant" at the 5 percent level according to this logic. Using the Bonferroni rule, the contrast `r5` vs. `r1` just misses attaining significance.

The Šidák test is almost identical to the Bonferroni, unless the number of comparison groups is quite large. In our example, the relevant critical value is about the same; a t statistic must be at least 2.9063 to be significant. The Scheffé test is even more conservative, requiring a t statistic of 3.178. The Scheffé procedure is designed to hold for any linear combination of the categories, not just for contrasts (comparisons of any two categories).

There is another consideration in a regression model that doesn't arise in the one-way ANOVA context. In the ANOVA, the means are guaranteed to be independent, since they come from independent samples. In the regression, a common adjustment introduces correlation between the category means. The Scheffé method is a conservative answer that applies equally well in the regression and ANOVA models. The Bonferroni and Šidák methods can become non-conservative in a regression context.

The remainder of this insert presents a series of Stata commands I have written to produce p values adjusted for multiple comparisons. The first command, `mcompp`, calculates and saves the p values but produces no output. The second command, `mcompr1`, reports contrasts that are not significantly different from each other. The third command, `mcompr2`, displays a report of all pairwise significant differences.

mcompp: Calculate p values

The syntax of `mcompp` is

`mcompp` *varlist* [`, nocons bonferroni(`*varname*`) scheffe(`*varname*`) sidak(`*varname*`)`]

varlist is a list of dummy variables that (in combination with the constant) defines a set of categories. This is a list of variables as might have been produced with a `tabulate, generate()` command, and these variables must also appear in the list of explanatory variables in the most recent estimation command. (All of Stata's estimation commands store the parameter estimates and the covariance matrix of estimates. This feature of Stata underlies the design of `mcompp`.) In the current version of `mcompp`, there must be one dummy variable for each category except the default, or omitted, category. The dummy variables must be coded so that '`1`' means the category defined is present and '`0`' means it is absent. This is the standard convention for Boolean algebra, but it is not the way data are usually received from a survey.

The options named for the three methods (`bonferroni`, `sidak`, and `scheffe`) specify variables that are to contain the p values for the Bonferroni, Šidák, and Scheffé methods, respectively.

If `nocons` is specified, then the variable list is assumed to contain an exhaustive list of categories to be compared. Otherwise, the list is assumed to omit one dummy variable corresponding to the default category.

The `nocons` option can be used even if it was not used in the original regression. This feature allows you to perform multiple comparisons on a subset of the categories represented by the dummy variables in the regression. For example, I recently needed to analyze differences between 34 health plans, but I only wanted to show contrasts within market areas. For all regions except the region that contained the default plan, I used the `nocons` option and the dummy variables standing for all the plans in that region.

We can illustrate `mcompp` by applying it to our auto price regression:

```
. mcompp r2 r3 r4 r5, scheffe(schp1) bonferr(bonp1) sidak(sidp1)
. list schp1 bonp1 sidp1 in 1/10
          schp1      bonp1      sidp1
  1.  .9926566          1   .9999364
  2.  .9786549          1   .9992037
  3.  .9997282          1          1
  4.  .8092963          1   .9075922
  5.  .8137907          1   .9117512
  6.  .7040055          1   .7921936
  7.  .3465602   .3683796   .3129417
  8.  .1976694   .1536581   .1434571
  9.   .092699    .052915   .0516726
 10.  .5256634   .7744335   .5533879
```

The contrasts are listed in the following order: `r2` vs. default, `r3` vs. default, `r3` vs. `r2`, `r4` vs. default, `r4` vs. `r3`, etc.

As you can see, the output from `mcompp` is difficult to decipher. The next two programs, `mcompr1` and `mcompr2`, display this output in more readable forms.

mcompr1: Report similar groups

One question we might want to answer after estimating the auto price regression is which groups of repair categories have similar prices. In this context, "similar" means there are no statistically significant differences between any of the categories within a group. `mcompr1` creates a report of similar groups, using the output from `mcompp` as its criterion for similarity.

The syntax of `mcompr1` is

`mcompr1` *varname* [`, cutoff(`*#*`) default(`*name*`) id(`*varname*`) generate(`*varname*`) label`] mcompr1 displays lists of categories whose mean responses are insignificantly different from each other. The *varname* following the command name contains the p values, calculated by mcompp, that determine whether differences are significant.

The cutoff() option specifies the p value to be used in grouping categories. The default cutoff is 5 percent.

The default() option supplies a name for the default or omitted category. Since the dummy variable for this category is omitted from the regression, mcompr1 has no way of finding its name. If you omit this option, mcompr1 uses the string "default" to identify this category.

The id() and generate() options specify variables in which to store the lists produced by mcompr1. These options are provided for users who wish to write more elaborate report writing programs than mcompr1. There is no reason to specify these options when using mcompr1 interactively.

The label option indicates that variable labels, rather than variable names, are to be used in the lists of categories. The default is to display the names of the dummy variables when listing groups of similar categories.

Continuing with our original example, we can use mcompr1 to display the groups of repair record categories that have similar auto prices. We use the Scheffé p values as the criterion, and we label the dummy variables to make the output more readable. For the sake of the example, we set the cutoff to 50 percent. In practice, you would generally leave the cutoff at its default level of 5 percent. We also add a label ("Poor") for the omitted category.

```
. label var r2 Fair

. label var r3 Average

. label var r4 Good

. label var r5 Excellent

. mcompr1 schp1, label default(Poor) cutoff(0.5)
Group A
 Poor
 Fair
 Average
 Good
Group B
 Good
 Excellent
```

According to this report, cars in the lowest four repair record categories have similar prices, as do cars in the highest two repair record categories.

mcompr2: Report significant pairwise differences

mcompr2 is the converse of mcompr1: instead of reporting groups of similar categories, mcompr2 reports of pairs of categories with significantly different mean responses. The syntax of mcompr2 is similar to that of mcompr1:

mcompr2 *varname* [, cutoff(*#*) default(*name*) effects label]

As before, the *varname* following the command name contains the p values calculated by mcompp. The default() and label options have the same meaning in mcompr2 as in mcompr1. The cutoff() option works slightly differently, though. By default, all pairwise contrasts are printed along with their p values. If the cutoff() option is specified, only contrasts with p values less than the cutoff are displayed. If the effects option is specified, the effects (contrasts) are also printed.

The following listing continues with the example of the auto price model and illustrates the behavior of mcompr2:

```
. mcompr2 schp1
   Group   Group  P-value
 ------------------------
      r5      r3  0.0927
      r5      r2  0.1977
      r5 default  0.3466
      r5      r4  0.5257
      r4      r3  0.7040
      r4 default  0.8093
      r4      r2  0.8138
      r3 default  0.9787
      r2 default  0.9927
      r3      r2  0.9997
```

```
. mcompr2 schp1, default(r1) cutoff(0.8) effects
 Group   Vs.     Diff.     P-value
 ---------------------------------
    r5    r3  2711.313     0.0927
    r5    r2  2909.323     0.1977
    r5    r1  3816.672     0.3466
    r5    r4  1669.015     0.5257
    r4    r3  1042.298     0.7040
. label var r2 "Fair"
. label var r3 "Average"
. label var r4 "Good"
. label var r5 "Excellent"
. mcompr2 schp1, default(Poor) cutoff(0.5) effects label
     Group       Vs.     Diff.     P-value
 -----------------------------------------
 Excellent   Average  2711.313     0.0927
 Excellent      Fair  2909.323     0.1977
 Excellent      Poor  3816.672     0.3466
```

An additional program for displaying pairwise differences is included on the distribution diskette. This program, `mcompr3`, was developed for the special application of a client, but it may be of interest to other Stata users as well. The syntax of `mcompr3` is

`mcompr3` *varname* `,` `greater(`*macro list*`)` `less(`*macro list*`)` [`cutoff(`*#*`)` `default(`*name*`)` `label`]

There must be as many macro names in each list as there are categories that are compared with `mcompp`. These macros receive the names or descriptions of the variables that this category is definitely greater than or less than. All other options work the same as in `mcompr2`.

`mcompr3` can be a little tricky, but the following example should give you the general idea of its operation and of its flexibility.

```
. mcompr3 schp1, label great(G1 G2 G3 G4 G5) less(L1 L2 L3 L4 L5) cutoff(.3) default(Poor)
. capture program drop doit
. program define doit
  1. di "Category              Clearly Greater Than        Clearly Less Than"
  2. di "Poor" _col(26) "$G1" _col(54) "$L1"
  3. di "Fair" _col(26) "$G2" _col(54) "$L2"
  4. di "Average" _col(26) "$G3" _col(54) "$L3"
  5. di "Good" _col(26) "$G4" _col(54) "$L4"
  6. di "Excellent" _col(26) "$G5" _col(54) "$L5"
  7. end
. doit
Category              Clearly Greater Than        Clearly Less Than
Poor
Fair                                                 Excellent
Average                                              Excellent
Good
Excellent                Fair,Average
```

There is yet another program, `ehcvsrc`, on the distribution diskette. The adventurous reader can examine `ehcvsrc` for a further elaboration of the use of `mcompr3`.

Formulas

The formulas are virtually identical to those described in [5s] oneway and, hence, are not repeated here.

snp7 | Natural cubic splines

Peter Sasieni, Imperial Cancer Research Fund, London, FAX (011)-44-171-269-3429

This entry consists of three related programs for smoothing by regression onto the truncated power base for a natural cubic spline: `spline`, `sp_adj` and `spbase`.

`spline` may be regarded as an alternative to `ksm`. It smooths a y-variable against an x-variable and displays a graph of the original data with the smooth superimposed. The smooth is calculated by regression onto a cubic spline basis. The user may specify the type of regression used to fit the smooth, e.g., `logistic`, `poisson`,. . . . By default the program uses `regress` (least squares).

`sp_adj` is similar to `spline` except that it permits the user to specify other covariates that are to be included in the regression model. The smooth is then that part of the "linear predictor" corresponding to the spline basis, and it is this that is plotted against the x-variable. The smooth thus represents the transformation of the covariate. If the regression is carried out with a non-linear link, the smooth will not be in the same scale as the original y-variable, For this reason the original data is not included on the graph.

`spbase` simply generates the truncated power basis for a natural cubic spline. The basis may then be used to adjust for the x-variable in a variety of regression models.

A natural cubic spline is a piecewise cubic polynomial that is everywhere twice continuously differentiable. In addition it is linear beyond the extreme knots; these are taken to be the minimum and maximum of the x-variable. The number or position of the interior knots may be specified. By default the program selects approximately $N^{-1/4}$ knots and places them at equally spaced percentiles (N is the number of observations).

Remarks

1. There is much overlap between the three programs, and a reader who is good at Stata programming may wish to write a "hidden" ado-file, so that these three programs can be rewritten with each one calling the hidden program.
2. The truncated power basis is easy to define, but computationally unstable. Ideally these programs would be converted to produce a B-spline basis.

Syntax

`spline` *yvar xvar* [`if` *exp*] [`in` *range*] [, `nknots(`*#*`)` `knots(`*#...#*`)` `regress(`*command*`)` `gen(`*smooth_fit*`)` `logit` `nograph` *graph_options*]

`sp_adj` *yvar xvar* [`if` *exp*] [`in` *range*] [, `nknots(`*#*`)` `knots(`*#...#*`)` `regress(`*command*`)` `adjust(`*varlist*`)` `gen(`*x_transform*`)` `nograph` *graph_options*]

`spbase` *xvar* [`if` *exp*] [`in` *range*] , `gen(`*basis*`)` [`nknots(`*#*`)` `knots(`*#...#*`)`]

Options

`gen(`*smooth_fit*`)` creates a new variable *smooth_fit* containing the smoothed fitted values from `spline`. Note that if the `logit` option is specified the values in *smooth_fit* will be in the logit scale.

`gen(`*x_transform*`)` creates a new variable *x_transform* containing the component of the linear predictor corresponding to the spline in *xvar* produced by `sp_adj`.

`gen(`*basis*`)` is not optional; it creates k new variables *basis*1,..., *basisk*, where k is the number of internal knots. *xvar* together with the k new variables form the spline basis. The names of all the variables in the basis are contained in a macro with the name of `basis`.

`knots(`*#...#*`)` specifies the exact location of the interior knots. The numbers may be separated by spaces and/or commas.

`nknots(`*#*`)` specifies the number of interior knots. `nknots` is ignored if the locations are specified using `knots`.

`regress(`*command*`)` selects the estimation command used to fit the model. By default `spline` and `sp_adj` both use `regress`. The program has been tested with `blogit`, `bprobit`, `clogit`, `cox`, `glogit`, `gprobit`, `logistic`, `logit`, `poisson`, and `probit`. If `clogit` is used, the stratification variable may be specified in the usual way. Similarly for the censoring variable with `cox`. The programs may require modification for use with other estimation commands such as `mlogit`. The stratification variable in `clogit` and the censoring variable in `cox` may be regarded as "covariates" and for this reason the options `dead` and `strata` may only be used with `sp_adj`. To use `blogit`, `bprobit`, `glogit`, or `gprobit`, you must specify the *pos_var* as the *xvar* and the *pop_var* as the *first* variable in the *varlist* of `adjust`.

`adjust(`*varlist*`)` adds the variables in *varlist* to the regression, so that the transform of *xvar* is adjusted. This option is not required; for instance, one may use `sp_adj` with `reg(clogit)` without `adjust`. To use `blogit`, `bprobit`, `glogit`, or `gprobit`, the *first* variable in the *varlist* of `adjust` must be the *pop_var*.

`logit` transforms the fitted values and plots the y-axis on a logit scale. The observations are automatically jittered vertically and plotted just outside the range of the smoothed curve. This option should only be used with binary data.

`nograph` suppresses production of the graph.

graph_options are any of the options allowed with `graph twoway`. `spline` plots *yvar* followed by its smooth against *xvar*; the default options are `s(oi)` and `c(.l)`. If there are more than 1000 observations, we advise using `s(.i)` instead. `sp_adj` only plots the smooth against *xvar*.

Example 1: spline

In this example `spline` is used to approximate a sinusoidal signal in the presence of Gaussian noise.

```
. clear
. set obs 1000
obs was 0, now 1000
. generate x = (_n-500)/100
. generate sinx = sin(x) + 0.2*invnorm(uniform())
. spline sinx x, xlabel ylabel(-1,-.5,0,.5,1) yline(0)
```

(graph appears, see Figure 1)

Example 2: sp_adj

In this example, we first generate Poisson data and then use `sp_adj` to fit the model. The program used to generate Poisson random variables may be of independent interest. It works using the property that the number of renewals by time 1 in a renewal process of independent exponential random variables with rate μ has a Poisson distribution with mean μ.

```
. clear
. set obs 1000
obs was 0, now 1000
. generate age = 40+15*invnorm(uniform())
. replace age = 30+15*uniform() if (age<15) | (age>75)
(43 real changes made)
. generate sex = uniform()<.4
. generate case = mod(_n,2)
. generate mu = age - 3*(log(age)^2) + 10*sin(age/15) + 50/age + 5*sex if case==0
(500 missing values generated)
. replace mu = age -3*(log(age)^2) +10*sin(age/25) +90/age -exp((age-65)/10) + 5*sex if case ==1
(500 real changes made)
. program define mk_pois
  1. generate tx = -(1/mu)*log(uniform())    /* tx is an exponential random variable */
  2. quietly generate int nx = .
  3. local i = 0
  4. while `i´ < 50 {
  5. quietly replace nx = `i´ if tx> 1 & nx==.
  6. quietly replace tx = tx - log(uniform())/mu
  7. local i = `i´ + 1
  8. }
  9. end
. mk_pois
. summarize nx
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+-----------------------------------------------------
      nx |    1000      10.104   5.975524          0         32
. sp_adj nx age if case==1, kn(20,35,50,65) reg(poisson) adj(sex) gen(sp1) nog
. sp_adj nx age if case==0, kn(20,35,50,65) reg(poisson) adj(sex) gen(sp0) nog
```

The smooths obtained from `sp_adj` with `poisson` regression must be transformed back to obtain the estimated mean function.

```
. generate sn1 = exp(sp1)
(500 missing values generated)
. generate sn0 = exp(sp0)
(500 missing values generated)
. graph sn1 sn0 mu age, sort c(ll.) s(ii.) l1("Count") xlabel ylabel
```

(graph appears, see Figure 2)

The dotted curves are the means which differ for males and females as well as cases and controls. The fitted curves are drawn only for those with `sex==0`. The raw Poisson data is not shown.

Bugs

Note that `sp_adj` will fail to produce an error message if the same variable is listed twice in `adjust`. The result however will be incorrect.

Figures

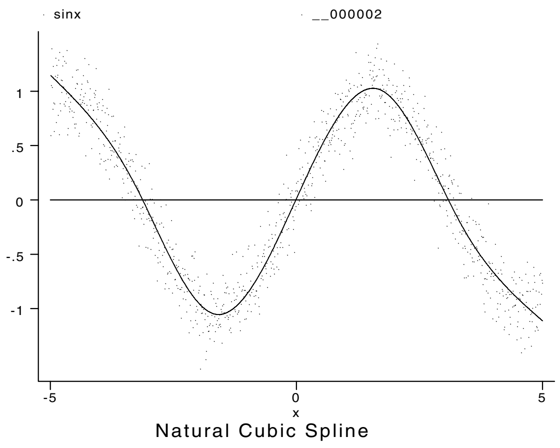

Figure 1: Recovered sine wave

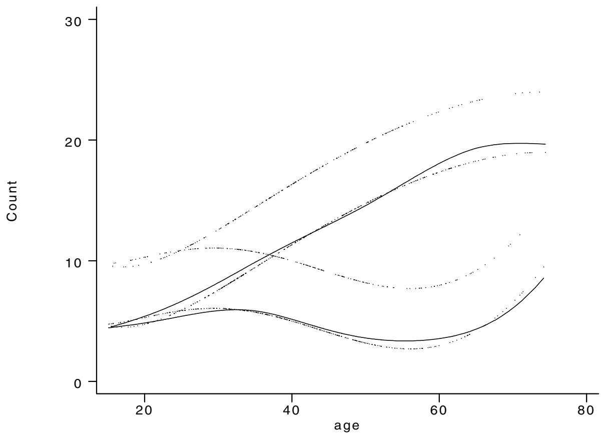

Figure 2: Poisson model

ssa6	Utilities for survival analysis with time-varying regressors

Dr. Philippe Bocquier, CERPOD, EMAIL bocquier@orstom.orstom.fr

Only one observation (record, line) per individual is needed for most survival analyses. However, when using time-varying covariates, one needs several observations per individual. The data set typically contains separate observations for each state the individual experiences from birth (or from the start of the period at risk) until the time of the event or of censorship. The time of interest for calculations is the time at the end of each state—that is, the time at the end of the period represented by each observation in the data set.

This insert presents four utilities that assist in preparing data for survival analysis when there are time-varying regressors. `censor` generates a censoring variable; `firstocc` identifies the first occurrence of an event; `slice` creates additional observations to trace fixed-time transitions; and `tmerge` match merges two files of ordered information on individuals. Each of these utilities is explained and demonstrated in the sections below.

Creating a censoring variable

One difficulty in file preparation is creating a censoring variable that corresponds to the particular event one wants to study. `censor` is a utility to create two variables, the waiting time until the event (or censorship), and a dummy that tells whether the event is a failure or a censorship. Its syntax is

`censor` *ident_var time_var* [*order_var*] `=` *exp* [`if` *exp*] [`in` *range*]
`,` `generate(`*new_var*`)` [`before(`*exp*`)`]

`censor` creates two new variables, specified by the `generate()` option. The first new variable is the censoring variable, a 0/1 variable that is set to one in the period at the end of which the individual changed state. The change in state is defined by the '`=` *exp*'. The censoring variable, *new_var*, is set to missing in periods after the individual changed state. The second new variable, `T`*new_var*, is the time at censoring. This variable is also set to missing after the event occurred.

Consider the following example data set:

```
. use example

. describe

Contains data from example.dta
  Obs:    20 (max=  1107)
 Vars:     7 (max=    99)
Width:    14 (max=   200)
  1. id           int    %8.0g                Identification number
  2. sex          int    %8.0g    sex         Sex
  3. activity     int    %8.0g    activity    Activity
  4. age          int    %8.0g                Age
  5. lost         int    %8.0g    lost        Still in survey?
  6. birth        int    %8.0g                Birth year
  7. year         int    %8.0g                Year
Sorted by:

. list id sex activity age

           id       sex  activity       age
  1.        1      Male  nonactiv        22
  2.        1      Male   selfemp        27
  3.        1      Male   selfemp        39
  4.        2    Female  nonactiv        19
  5.        3    Female  nonactiv        19
  6.        3    Female   selfemp        22
  7.        3    Female     waged        28
  8.        4      Male  nonactiv         7
  9.        4      Male  nonactiv        20
 10.        4      Male   selfemp        23
 11.        5      Male  nonactiv         4
 12.        5      Male     waged        11
 13.        5      Male  nonactiv        22
 14.        5      Male     waged        25
 15.        6    Female  nonactiv        12
 16.        6    Female  nonactiv        24
 17.        6    Female     waged        27
 18.        7    Female  nonactiv        24
 19.        8    Female  nonactiv        16
 20.        8    Female  nonactiv        39
```

This data set contains observations on eight individuals, identified by the variable `id`. The sex, work status (`activity`), and age of each individual is recorded in the data set along with a variable (`lost`) that indicates whether the individual left the sample for some reason. This data set also contains other variables that are used later in this insert. The data set is supplied on the STB diskette.

The work status variable is coded as a '`0`' when the individual is not-working (`nonactiv`), as a '`1`' when the individual is employed (`waged`), and as a '`2`' when the individual is self-employed (`selfemp`). We can use `censor` to create a censoring variable for the individual's first entry into work:

```
. censor id age = activity==1 | activity==2, generate(job1)

. list id activity age job1 Tjob1

           id  activity       age      job1     Tjob1
  1.        1  nonactiv        22         1        22
  2.        1   selfemp        27         .         .
  3.        1   selfemp        39         .         .
  4.        2  nonactiv        19         0        19
  5.        3  nonactiv        19         1        19
  6.        3   selfemp        22         .         .
  7.        3     waged        28         .         .
  8.        4  nonactiv         7         0         7
  9.        4  nonactiv        20         1        20
 10.        4   selfemp        23         .         .
 11.        5  nonactiv         4         1         4
 12.        5     waged        11         .         .
 13.        5  nonactiv        22         .         .
 14.        5     waged        25         .         .
 15.        6  nonactiv        12         0        12
 16.        6  nonactiv        24         1        24
 17.        6     waged        27         .         .
 18.        7  nonactiv        24         0        24
 19.        8  nonactiv        16         0        16
 20.        8  nonactiv        39         0        39
```

In this example, censor generates two variables, job1 and Tjob1. job1 is the 0/1 variable that records the individual's change in status. Tjob1 records the time—age in this example—at which the change occurs. For individual '1', the transition took place at the end of the first observation, when he passed from the not working state to the self-employed state. This individual was 22 years old at the end of the first observation.

Censoring can occur for reasons unrelated to the phenomenon under study. In this example data set, some individuals are lost from the survey. The variable lost is coded as '0' when an individual is still in the survey and as '1' when the individual is "lost".

The before() option of censor provides a convenient way to handle the lost individuals:

```
. drop job1 Tjob1
. censor id age = activity==1 | activity==2, generate(job1) before(lost==1)
. list id activity age lost job1 Tjob1
          id  activity       age      lost      job1     Tjob1
  1.       1  nonactiv        22        in         1        22
  2.       1   selfemp        27        in         .         .
  3.       1   selfemp        39        in         .         .
  4.       2  nonactiv        19        in         0        19
  5.       3  nonactiv        19        in         1        19
  6.       3   selfemp        22        in         .         .
  7.       3     waged        28        in         .         .
  8.       4  nonactiv         7        in         0         7
  9.       4  nonactiv        20       out         .         .
 10.       4   selfemp        23       out         .         .
 11.       5  nonactiv         4        in         1         4
 12.       5     waged        11        in         .         .
 13.       5  nonactiv        22        in         .         .
 14.       5     waged        25        in         .         .
 15.       6  nonactiv        12        in         0        12
 16.       6  nonactiv        24       out         .         .
 17.       6     waged        27       out         .         .
 18.       7  nonactiv        24        in         0        24
 19.       8  nonactiv        16        in         0        16
 20.       8  nonactiv        39       out         .         .
```

Suppose we need to create two censoring variables for two competing risks; for example, wage-work and self-employment. censor can handle this case as well:

```
. censor id age = activity==1, generate(waged) before(lost==1 | activity==2)
. censor id age = activity==2, generate(self) before(lost==1 | activity==1)
. list id activity age waged Twaged self Tself, nodisplay
          id  activity       age     waged    Twaged      self     Tself
  1.       1  nonactiv        22         0        22         1        22
  2.       1   selfemp        27         .         .         .         .
  3.       1   selfemp        39         .         .         .         .
  4.       2  nonactiv        19         0        19         0        19
  5.       3  nonactiv        19         0        19         1        19
  6.       3   selfemp        22         .         .         .         .
  7.       3     waged        28         .         .         .         .
  8.       4  nonactiv         7         0         7         0         7
  9.       4  nonactiv        20         .         .         .         .
 10.       4   selfemp        23         .         .         .         .
 11.       5  nonactiv         4         1         4         0         4
 12.       5     waged        11         .         .         .         .
 13.       5  nonactiv        22         .         .         .         .
 14.       5     waged        25         .         .         .         .
 15.       6  nonactiv        12         0        12         0        12
 16.       6  nonactiv        24         .         .         .         .
 17.       6     waged        27         .         .         .         .
 18.       7  nonactiv        24         0        24         0        24
 19.       8  nonactiv        16         0        16         0        16
 20.       8  nonactiv        39         .         .         .         .
```

Locating the first occurrence of an event

`firstocc` is a utility to identify the first occurrence of an event for each individual in an ordered sequence of observations and (optionally) to store the corresponding observations in a new file. The syntax of `firstocc` is

`firstocc` *ident_var rank_var* `=` *exp* [`if` *exp*] [`in` *range*] , { `generate(`*new_var*`)` | `saving(`*new_file*`)` }

Either the `generate()` or the `saving()` option must be specified. If the `generate()` option is chosen, `firstocc` generates a 0/1 variable, *new_var*, that is equal to '`1`' when the `=` *exp* is true for the first time in a sequence of ordered observations. When the `saving()` option is chosen, `firstocc` creates a new file that contains only the observations where *new_var* would be equal to '`1`'.

Using the same example data set, we can use `firstocc` to retain only the observations on the first period of labor market activity. The ranking (ordering) variable in this example is the individual's age:

```
. use example, clear

. firstocc id age = activity==1 | activity==2, saving(job1)
file job1.dta saved

. list id sex activity age lost
           id       sex  activity       age      lost
  1.        1      Male   selfemp        27        in
  2.        3    Female   selfemp        22        in
  3.        4      Male   selfemp        23       out
  4.        5      Male     waged        11        in
  5.        6    Female     waged        27       out
```

Make sure your original file is saved before running `firstocc` with `saving()` option, because `firstocc` will delete observations from your original data set without asking you for any confirmation.

Creating multiple observations for fixed-time transitions

`slice` is used in survival analysis to add observations for times or ages that may not have been directly observed. In the example data set used above, an observation is added only when an individual changes job status. If an individual is recorded as '`nonactiv`' at age 20 and '`selfemp`' at age 30, the individual's states at ages 21–29 can be inferred, but they are not recorded in the data set. `slice` fills in observations for any desired age or time interval, making it easy to analyze the distribution of states for any arbitrary interval. This explanation of `slice` may sound a bit convoluted, but the operation of `slice` is easily understood from an example or two.

The syntax of `slice` is

`slice` *timevar* [`if` *exp*] [`in` *range*] , `generate(`*new_var*`)` `interval(`*interval*`)`
`saving(`*file_name*`)` `tvid(`*varname*`)` [`nolabel` `start(`*varname*`)`]

timevar specifies the time at the end of the period represented by each observation. Observations with missing *timevar* are ignored and are stored after the others for each individual. Note that the number of observations added to the original data can be substantial and is dependent on the number of periods that individuals pass through. You may need to repartition memory with the `memsize` command before using `slice`.

To illustrate `slice`, suppose we are interested in studying the job status of individuals at different ages, using the example data set introduced above. For example, we can examine the `slice` of the data corresponding to age 26 by typing

```
. use example, clear

. slice age, tvid(id) interval(25,26) saving(new26, replace) generate(grage)
   -25    + 4
[25-26    + 4
[26-     --------
           8 records added to 20
```

This command added eight observations to the original twenty and stored a copy of the result in `new26.dta`. `slice` also added a new variable, `grage`, to the data. `grage` is coded as '`0`' when the individual's age is less than or equal to 25, as '`1`' when the age is exactly 26, and as '`2`' when the age is greater than 26. `grage` has an attached value label in the following list:

```
. list id activity age birth year grage
```

```
             id  activity       age     birth       year     grage
  1.          1  nonactiv        22        54         76       -25
  2.          1   selfemp        25        54         81       -25
  3.          1   selfemp        26        54         81    [25-26
  4.          1   selfemp        27        54         81      [26-
  5.          1   selfemp        39        54         93      [26-
  6.          2  nonactiv        19        74         93       -25
  7.          3  nonactiv        19        65         84       -25
  8.          3   selfemp        22        65         87       -25
  9.          3     waged        25        65         93       -25
 10.          3     waged        26        65         93    [25-26
 11.          3     waged        28        65         93      [26-
 12.          4  nonactiv         7        70         77       -25
 13.          4  nonactiv        20        70         90       -25
 14.          4   selfemp        23        70         93       -25
 15.          5  nonactiv         4        68         72       -25
 16.          5     waged        11        68         79       -25
 17.          5  nonactiv        22        68         90       -25
 18.          5     waged        25        68         93       -25
 19.          6  nonactiv        12        66         78       -25
 20.          6  nonactiv        24        66         90       -25
 21.          6     waged        25        66         93       -25
 22.          6     waged        26        66         93    [25-26
 23.          6     waged        27        66         93      [26-
 24.          7  nonactiv        24        69         93       -25
 25.          8  nonactiv        16        54         70       -25
 26.          8  nonactiv        25        54         93       -25
 27.          8  nonactiv        26        54         93    [25-26
 28.          8  nonactiv        39        54         93      [26-
```

Now it is straightforward to analyze the job status of the four individuals in this sample who reached the age of 26 during the survey period.

```
. tabulate activity sex if grage==1

           | Sex
  Activity|      Male     Female |     Total
-----------+----------------------+----------
 nonactiv |         0          1 |         1
    waged |         0          2 |         2
  selfemp |         1          0 |         1
-----------+----------------------+----------
     Total|         1          3 |         4
```

Suppose that we are interested in studying the employment situation at a particular date, say 1991:

```
. use example, clear

. list id activity age birth year

             id  activity       age     birth       year
  1.          1  nonactiv        22        54         76
  2.          1   selfemp        27        54         81
  3.          1   selfemp        39        54         93
  4.          2  nonactiv        19        74         93
  5.          3  nonactiv        19        65         84
  6.          3   selfemp        22        65         87
  7.          3     waged        28        65         93
  8.          4  nonactiv         7        70         77
  9.          4  nonactiv        20        70         90
 10.          4   selfemp        23        70         93
 11.          5  nonactiv         4        68         72
 12.          5     waged        11        68         79
 13.          5  nonactiv        22        68         90
 14.          5     waged        25        68         93
 15.          6  nonactiv        12        66         78
 16.          6  nonactiv        24        66         90
 17.          6     waged        27        66         93
 18.          7  nonactiv        24        69         93
 19.          8  nonactiv        16        54         70
 20.          8  nonactiv        39        54         93

. slice year, tvid(id) interval(90,91) saving(new91, replace) generate(period)
  -90    + 5
[90-91   + 8
[91-     --------
           13 records added to 20
```

```
. list id activity age birth year period
            id  activity       age     birth      year    period
  1.         1  nonactiv        22        54        76       -90
  2.         1   selfemp        27        54        81       -90
  3.         1   selfemp        39        54        90       -90
  4.         1   selfemp        39        54        91    [90-91
  5.         1   selfemp        39        54        93      [91-
  6.         2  nonactiv        19        74        90       -90
  7.         2  nonactiv        19        74        91    [90-91
  8.         2  nonactiv        19        74        93      [91-
  9.         3  nonactiv        19        65        84       -90
 10.         3   selfemp        22        65        87       -90
 11.         3     waged        28        65        90       -90
 12.         3     waged        28        65        91    [90-91
 13.         3     waged        28        65        93      [91-
 14.         4  nonactiv         7        70        77       -90
 15.         4  nonactiv        20        70        90       -90
 16.         4   selfemp        23        70        91    [90-91
 17.         4   selfemp        23        70        93      [91-
 18.         5  nonactiv         4        68        72       -90
 19.         5     waged        11        68        79       -90
 20.         5  nonactiv        22        68        90       -90
 21.         5     waged        25        68        91    [90-91
 22.         5     waged        25        68        93      [91-
 23.         6  nonactiv        12        66        78       -90
 24.         6  nonactiv        24        66        90       -90
 25.         6     waged        27        66        91    [90-91
 26.         6     waged        27        66        93      [91-
 27.         7  nonactiv        24        69        90       -90
 28.         7  nonactiv        24        69        91    [90-91
 29.         7  nonactiv        24        69        93      [91-
 30.         8  nonactiv        16        54        70       -90
 31.         8  nonactiv        39        54        90       -90
 32.         8  nonactiv        39        54        91    [90-91
 33.         8  nonactiv        39        54        93      [91-
```

Since every individual is observed as late as 1993 and none is observed in both 1990 and 1991, slicing the data on the 1990–91 interval affects the observations for each individual. For instance, `slice` adds two observations for individual '`1`', one for 1990, and one for 1991. The values of all the variables except `year` are copied down from the observation where `year==93`, the first observation following the 1990–91 interval. As a consequence, the age is incorrect in the added observations. This problem is easily corrected.

```
. replace age = year - birth
(13 real changes made)
```

Now we can examine activity in 1991 (`period==1`):

```
. tabulate activity sex if period==1
           | Sex
  Activity|      Male     Female |     Total
-----------+----------------------+----------
 nonactiv |        0          3 |         3
    waged |        1          2 |         3
  selfemp |        2          0 |         2
-----------+----------------------+----------
     Total|        3          5 |         8
```

Apart from descriptive analysis, the `slice` command is most useful in Cox regression, when we use time-varying regressors to test the effect of passing through different calendar periods or age groups. For example, we can slice the time into three periods: from the earliest year through 1980, from 1981 through 1990, and from 1991 until the time at survey (1993):

```
. use example, clear
. slice year, tvid(id) interval(80,90) saving(new) generate(period)
   -80    + 8
[80-90    + 5
[90-     --------
           13 records added to 20
```

Note that `age` has to be corrected again, before running the regression:

```
. replace age = year - birth
(13 real changes made)
```

Now, after creating dummies for the time period . . .

```
. tabulate period, generate(per)
 Time period|
      period|      Freq.     Percent        Cum.
------------+-----------------------------------
        -80 |         14       42.42       42.42
     [80-90 |         11       33.33       75.76
       [90- |          8       24.24      100.00
------------+-----------------------------------
      Total |         33      100.00
```

and a censoring variable . . .

```
. censor id age = activity==1 | activity==2, generate(job1)
```

we can estimate a Cox regression model:

```
. cox Tjob1 per2 per3, tvid(id) dead(job1)
```

We suppress the output of `cox` because our example data are too artificial to make the results meaningful. We have, however, illustrated the process of preparing data for Cox regression; you can apply it to authentic data of your own.

Combining files by time

`tmerge` is a utility to combine two files each containing ordered observations on one or more individuals. The syntax of `tmerge` is

`tmerge` *ident_var filenameA*(*time_varA*) *filenameB*(*time_varB*) *new_filename*(*new_time_var*)

`tmerge` combines *filenameA* and *filenameB* and stores the result in *new_filename* (and in the current data set). Observations are matched according to the values of *time_varA* and *time_varB*, which must be expressed in the same scale.

As with the commands described above, `tmerge` is best understood by looking at an example. Suppose you have two files containing different event histories for the same individual. One file, `job.dta`, contains the individual's employment history:

```
. use job, clear

. list
             id     birth      Tjob      prof
  1.          2        63        80         3
  2.          2        63        84         1
  3.          2        63        85         2
  4.          2        63        89         1
```

The other file, `mar.dta`, contains the individual's marriage history:

```
. use mar, clear

. list
             id     birth      car1      Tmar      car2
  1.          2        63         1        78         4
  2.          2        63         2        89         3
```

Time variables—`Tjob` in `job.dta`, `Tmar` in `mar.dta`—must be expressed in the same scale (here, in years). In each file the censoring time is 89. `tmerge` will be interrupted and an error message will appear if the times at censoring are not the same in both files. To combine the files, type

```
. drop _all

. tmerge id job(Tjob) mar(Tmar) job_mar(time)
set maxvar 19 width 29

Please wait...
```

```
The variable ´_File´ indicates in which file the time change is originated,
either job , mar , or both.
 Record from|
     file...|      Freq.     Percent        Cum.
------------+-----------------------------------
        job |          3       50.00       50.00
        mar |          2       33.33       83.33
       both |          1       16.67      100.00
------------+-----------------------------------
      Total |          6      100.00
file job_mar.dta saved
```

The result is

```
. list, nodisplay noobs
      id     birth      Tjob      prof      time      car1      Tmar      car2     _File
       2        63        80         3        78         1        78         4       mar
       2        63        80         3        80         2        89         3       job
       2        63        84         1        84         2        89         3       job
       2        63        85         2        85         2        89         3       job
       2        63        89         1        89         2        89         3      both
```

Note that the original dates, `Tjob` and `Tmar`, are not modified. Only the new time variable, `time`, should be used in subsequent analysis. `tmerge` creates a new variable, `_File`, which records the file from which the transition originated.

sss1.1 Updated U.S. marginal income tax rate function

Timothy J. Schmidt, Federal Reserve Bank of Kansas City, 816-881-2307

STB-15 included an addition to the `egen` command ([5d] egen)—the `mtr()` function (Schmidt 1993). The `mtr()` function finds the marginal income tax rate corresponding to any given level of taxable income for a married couple filing jointly in the United States. While the original version covered the years 1930 to 1990, the updated version of `mtr()` extends through the 1994 tax year.

The syntax is unchanged from the original version:

egen [*type*] *varname* = **mtr(***year*,*income***)** [**if** *exp*] [**in** *range*]

The year and income variables can be replaced with constants. For example,

```
. egen taxrate = mtr(1993,myincome)
```

Reference

Schmidt, T. 1993. sss1: Calculating U.S. marginal income tax rates. *Stata Technical Bulletin* 15: 17–19.

sts7.4 A library of time series programs for Stata (Update)

Sean Becketti, Stata Technical Bulletin, FAX 914-533-2902

In *sts7*, a library of time series programs for Stata was introduced (Becketti 1994). That insert described an approach to time series analysis that builds on Stata's core commands and on its extensibility. The insert also cataloged the programs in the time series library.

As *sts7* promised, the time series library is updated in each issue of the STB. New programs and revisions are posted on the STB distribution diskette. If you use the time series library, you should copy each new version over your existing version. Type `help tsnew` to see a history of the changes in the library. Type `help ts` to see a catalog of all the programs in the library.

New features

This edition of the time series library marks the introduction of commands to handle panel data sets; that is, data sets that include time series observations on multiple cross-sectional units. This extension is not fully implemented yet: not all commands in the time series library handle panel data. Moreover, this extension introduces fundamental changes in the internal operations of several important commands, most notably the `lag` command. These changes should not affect the operation of the commands if you do not use the panel data extensions. However, the potential for unintended side effects exists.

As a consequence, we strongly recommend that you retain the previous version of the time series library. In the event that a time series command behaves in an unexpected manner, simply revert to the previous version. In subsequent releases, the panel data features will be extended to more and more of the programs in the time series library. When this extension becomes fully integrated into the library, the older version can be erased.

Command to define cross-sectional units: The `csunits` command specifies the variables that identify cross-sectional units. The syntax is

`csunits` *varlist* [, `clear`]

`csunits` is the cross-sectional analog of the `datevars` command. The `clear` option is a convenience feature; it “erases” the existing definition. `csunits` is illustrated in the example that follows the discussion of the `lag` command.

Generalization of the `lag` command: `lag` has been extended to handle panel data correctly. When a variable is lagged (lead), missing values are created at the beginning (end) of the time series. In panel data, the time series restart for each cross-sectional unit. If the cross-sectional units have been defined by the `csunits` command, `lag` will operate on each cross-sectional unit independently, similar to the way the `by` *varlist*`:` prefix operates on each by-group independently.

`lag` is the most heavily-used command in the time series library. Almost every other program in the library calls `lag`. Thus, this extension to `lag` may affect other programs in the library in unexpected ways. For example, `tsreg` and `tsfit` should now handle panel data appropriately. Both these routines also call `findsmpl` to report sample coverage. But `findsmpl` does *not* handle panel data, thus the information on sample coverage should be suppressed or ignored. More importantly, the time series features of `regdiag`, such as the Durbin–Watson statistic, do not handle panel data correctly yet. None of these side effects are relevant unless you use panel data *and* you identify the cross-sectional units with the `csunits` command.

Here is a simple, artificial example that illustrates `csunits` and the new behavior of `lag`. We have observations on three cross-sectional units defined by the variable `id`. We observe unit 100 from period 1 through period 5, unit 101 from period 3 through period 7, and unit 105 from period 2 through period 4. If no time series or cross section information is specified, `lag` operates as before:

```
. lag x

. list
            id       time          x        L.x
  1.       100          1       3.21          .
  2.       100          2      67.10       3.21
  3.       100          3      98.30       67.1
  4.       100          4      62.96       98.3
  5.       100          5      24.59      62.96
  6.       101          3      89.84      24.59
  7.       101          4      33.59      89.84
  8.       101          5       4.07      33.59
  9.       101          6      31.31       4.07
 10.       101          7      78.12      31.31
 11.       105          2      94.58      78.12
 12.       105          3       8.63      94.58
 13.       105          4      89.53       8.63
```

`lag` assumes the data are in the appropriate order when it creates `L.x`. Note that `lag` does not respect the boundaries of the cross-sectional units. For example, `L.x` is 24.59 in observation 6, the first observation on unit 101, but 24.59 is the value of `x` in the last observation on unit 100.

If time series information is recorded by `period` and `datevars`, the new version of `lag` will sort the data before generating leads and lags:

```
. period 1
1 (annual)

. datevars time

. lag x
(note:  L.x replaced)

. list
            id       time          x        L.x
  1.       100          1       3.21          .
  2.       105          2      94.58       3.21
  3.       100          2      67.10      94.58
  4.       100          3      98.30       67.1
  5.       105          3       8.63       98.3
  6.       101          3    89.84       8.63
  7.       105          4    89.53      89.84
  8.       100          4    62.96      89.53
  9.       101          4    33.59      62.96
 10.       101          5     4.07      33.59
 11.       100          5    24.59       4.07
 12.       101          6    31.31      24.59
 13.       101          7    78.12      31.31
```

This is still not the desired result. We use `csunits` to specify the cross-sectional units, then we recreate `L.x`:

```
. csunits id

. lag x
(note:  L.x replaced)

. list
            id       time          x        L.x
  1.       100          1     3.21          .
  2.       105          2    94.58          .
  3.       100          2    67.10       3.21
  4.       100          3    98.30       67.1
  5.       105          3     8.63      94.58
  6.       101          3    89.84          .
  7.       101          4    33.59      89.84
  8.       100          4    62.96       98.3
  9.       105          4    89.53       8.63
 10.       101          5     4.07      33.59
 11.       100          5    24.59      62.96
 12.       101          6    31.31       4.07
 13.       101          7    78.12      31.31
```

`lag` now takes a `nosort` option that restores its original behavior.

Reference

Becketti, S. 1994. sts7: A library of time series programs for Stata. *Stata Technical Bulletin* 17: 28–32.

zz3.6 Computerized index for the STB (Update)

William Gould, Stata Corporation, FAX 409-696-4601

The STBinformer is a computerized index to every article and program published in the STB. The command (and entire syntax) to run the STBinformer is `stb`. Once the program is running, you can get complete instructions for searching the index by typing `?` for help or `??` for more detailed help.

The STBinformer appeared for the first time on the STB-16 distribution diskette and included indices for the first fifteen issues of the STB. The STB-22 distribution diskette contains an updated version of the STBinformer that includes indices for the first *twenty-one* issues of the STB. As the original insert stated, I intend to include an updated copy of this computerized index on every STB diskette. I encourage you to contact me with suggestions for changes and improvements in the program.

Reference

Gould W. 1993. Computerized index for the STB. *Stata Technical Bulletin* 16: 27–32.

STB categories and insert codes

Inserts in the STB are presently categorized as follows:

General Categories:			
an	announcements	*ip*	instruction on programming
cc	communications & letters	*os*	operating system, hardware, & interprogram communication
dm	data management		
dt	data sets	*qs*	questions and suggestions
gr	graphics	*tt*	teaching
in	instruction	*zz*	not elsewhere classified
Statistical Categories:			
sbe	biostatistics & epidemiology	*srd*	robust methods & statistical diagnostics
sed	exploratory data analysis	*ssa*	survival analysis
sg	general statistics	*ssi*	simulation & random numbers
smv	multivariate analysis	*sss*	social science & psychometrics
snp	nonparametric methods	*sts*	time-series, econometrics
sqc	quality control	*sxd*	experimental design
sqv	analysis of qualitative variables	*szz*	not elsewhere classified

In addition, we have granted one other prefix, *crc*, to the manufacturers of Stata for their exclusive use.

International Stata Distributors

International Stata users may also order subscriptions to the *Stata Technical Bulletin* from our International Stata Distributors.

Company:	Dittrich & Partner Consulting	Company:	Oasis Systems BV
Address:	Prinzenstrasse 2 D-42697 Solingen Germany	Address:	Lekstraat 4 3433 ZB Nieuwegein The Netherlands
Phone:	+49 212-3390 99	Phone:	+31 3402 66336
Fax:	+49 212-3390 90	Fax:	+31 3402 65844
Countries served:	Austria, Germany	Countries served:	The Netherlands
Company:	Howching	Company:	Ritme Informatique
Address:	11th Fl. 356 Fu-Shin N. Road Taipei, Taiwan, R.O.C.	Address:	34 boulevard Haussmann 75009 Paris, France
Phone:	+886-2-505-0525	Phone:	+33 1 42 46 00 42
Fax:	+886-2-503-1680	Fax:	+33 1 42 46 00 33
Countries served:	Taiwan	Countries served:	Belgium, France, Luxembourg, Switzerland
Company:	Metrika Consulting	Company:	Timberlake Consultants
Address:	Ruddammsvagen 21 11421 Stockholm Sweden	Address:	47 Hartfield Crescent West Wickham Kent BR4 9DW, U.K
Phone:	+46-8-163128	Phone:	+44 181 462 0495
Fax:	+46-8-6122383	Fax:	+44 181 462 0493
Countries served:	Baltic States, Denmark, Finland, Iceland, Norway, Sweden	Countries served:	Eire, Portugal, U.K.