The Stata Journal (2001) **1**, Number 1, pp. 1–24

Confidence intervals and p-values for delivery to the end user

Roger Newson
King's College London, UK
roger.newson@kcl.ac.uk

Abstract. Statisticians make their living producing confidence intervals and *p*-values. However, those in the Stata log are not ready for delivery to the end user, who usually wants to see statistical output either as a plot or as a table. This article describes a suite of programs used to convert Stata results to one or other of these forms. The `eclplot` package creates plots of estimates with confidence intervals, and the `listtex` package outputs a Stata data set in the form of table rows that can be inserted into a plain TEX, LATEX, HTML or word processor table. To create a Stata data set that can be output in these ways, we can use the `parmest`, `dsconcat` and `lincomest` packages to create data sets with one observation per estimated parameter, the `sencode`, `tostring`, `ingap` and `reshape` packages to process these data sets into a form ready to be output, and the `descsave` and `factext` packages to reconstruct, in the output data set, categorical predictor variables represented by dummy variables in regression models.

Keywords: st0001, confidence interval, *p*-value, plot, table, estimation results, TEX, LATEX, HTML, word processor, presentation, `eclplot`, `listtex`, `parmest`, `dsconcat`, `lincomest`, `sencode`, `tostring`, `ingap`, `reshape`, `descsave`, `factext`

1 Introduction

Statisticians (and other statistically-minded scientists) make much of their living producing confidence intervals and *p*-values. However, the confidence intervals and *p*-values output to Stata logs are not in any form which can be delivered to end users, such as the audience at a presentation or the readers of a journal. These end users, at the very least, want the results to be formatted to a consistent number of decimal places, or possibly significant figures. To be understood by an audience not composed of statistical programmers, statistical results are usually presented in one of two ways. These are tables, which are typically preferred by medical periodicals, particularly those in the *British Medical Journal* (*BMJ*) group, and plots, which have much more immediate impact in a seminar or a conference presentation. To this day, many Stata users spend a lot of time manually rounding the estimates, confidence limits and *p*-values in Stata logs, writing them down in tables (often large), and plotting them (often using graphics software other than Stata). This does not constitute "the human use of human beings" (see Wiener (1988)), although there are good reasons why users of Stata versions before Version 8 traditionally did a lot of their graphics otherwise.

This article outlines how the whole process of producing statistical results for delivery to readers and audiences can be executed within Stata, transferring the plots and tables

to word processors, presentation packages and typesetting packages only at the end. This is done using a suite of packages downloadable from SSC, together with some of the facilities of official Stata. Each of these packages can be downloaded alone, but the different packages can be used together in ways not easily appreciated by examining the on-line help of each package in isolation, although the on-line help files of these packages frequently refer to each other. The remaining Sections describe the various subsets into which these packages naturally fall. First, we introduce the `eclplot` and `listtex` packages, which are used to produce the plots and tables, respectively, taking, as input, Stata data sets with one observation per estimated parameter, or one observation per table row. Second, we introduce the packages `parmest`, `dsconcat` and `lincomest`, which create Stata data sets with one observation per estimated parameter and data on parameter estimates, confidence limits, p-values and possibly other parameter attributes requested by the user. Third, we describe some tools for processing such data sets into a form where the observations correspond to rows of a table or of a horizontal confidence interval plot. These tools are the `sencode` and `ingap` packages, available on SSC, and the official Stata utilities `tostring` and `reshape`. Finally, we introduce the `descsave` and `factext` packages, which are used for reconstructing, in a `parmest` output data set, categorical variables present as predictors in the regression models used to produce the output. These categorical variables can also be plotted and tabulated.

2 Creating plots and tables using eclplot and listtex

Statistical results presented in presentations at meetings, journal articles and Web pages are presented in two ways, namely plots and tables. We start by introducing two packages used to produce these two respective forms of output for insertion into a presentation document, a word processor or typesetting document, or a HTML document. These packages are `eclplot`, which produces Stata 8 graphics plotting estimates and confidence limits against another variable, and `listtex`, which outputs Stata data sets as table rows that can be inserted into tables in a range of document types. This is intended to motivate the user to read further and find out how to create the Stata data sets that `eclplot` and `listtex` take as input. Such input data sets are usually themselves created as output by the other Stata packages described in this article.

2.1 Creating confidence interval plots using eclplot

As a statistician, I find that most of the graphics that I present are horizontal plots of confidence intervals. Before I had access to Stata 8, I presented these at meetings using Nicholas J. Cox's package `hplot`, downloadable from SSC. This package represented the state of the art in Stata 6 and 7 graphics, and the plots produced are at least of presentation quality, in that they communicate confidence intervals with a lot more immediate impact than could be done with a table. On upgrading to Stata 8, I naturally wanted to do the same, only with better graphics.

For the special case of confidence intervals for means, Nicholas J. Cox has produced

a Stata 8 package `ciplot`, available on SSC. However, for the more general confidence interval plot, I developed the `eclplot` package. `eclplot` takes, as input, a Stata data set with four variables, containing, respectively, the estimates, the lower confidence limits, the upper confidence limits, and a fourth variable, to be plotted on the other axis of the confidence interval plot. It creates, as output, horizontal or vertical confidence interval plots, in which the estimates and upper and lower confidence limits are plotted on one axis and the fourth variable on the other axis.

For instance, suppose that we have created a data set with one observation per confidence interval for a regression model comparing each of the 6 countries of origin of the companies making the cars in the `auto` data, shipped with Stata (US, Germany, Japan, France, Italy, Sweden). Suppose, also, that this data set has variables `country`, encoding the country, and `estimate`, `min95` and `max95`, representing the estimates and lower and upper 95% confidence limits, respectively, of the parameters, most of which are differences in fuel use (gallons per 100 miles) between cars from a given country and cars from the US. (We will find out later how we might construct such a data set.) The countries and confidence intervals might be listed and plotted as follows:

```
. list parm label country estimate min95 max95,clean
             parm        label   country   estimate   min95   max95
  1.                                   US       0.00    0.00    0.00
  2.   _Icountry_2   country==2   Germany      -1.17   -2.13   -0.21
  3.   _Icountry_3   country==3     Japan      -1.30   -2.09   -0.50
  4.   _Icountry_4   country==4    France       0.18   -1.54    1.90
  5.   _Icountry_5   country==5     Italy      -0.56   -2.97    1.85
  6.   _Icountry_6   country==6    Sweden       0.56   -1.85    2.97
  7.         _cons     Constant         .       5.32    4.99    5.65
. eclplot estimate min95 max95 country if parm != "_cons", hori ///
>  estopts(msymbol(circle) msize(vlarge)) ciopts(msize(huge)) ///
>  yscale(range(0 7)) ylabel(1(1)6) ///
>  xlabel(-4(1)4,format(%8.0f)) xline(0, lpattern(dot)) ///
>  xtitle("Difference in fuel use from US (gallons/100 miles)") ///
>  xsize(4) ysize(3)
```

The resulting graph is given as Figure 1. Note that US-made cars are the reference category. The variables `parm` and `label`, and the last observation (excluded from the plot), will be explained in later Sections.

2.2 Outputting table rows using listtex

Tables have less immediate impact than plots, but have the advantages of consuming less space, and of being easier for readers to copy, if they want to re-tabulate or re-plot the data in their own ways. They therefore tend to be the preferred medium for many periodicals, with the result that scientists are often required to deliver them. A comprehensive labour-saving tool for producing tables is the `listtex` package, whose most recent version (currently in Stata 8) can be downloaded from SSC, although a Stata 7 version can be downloaded from my website at *http://www.kcl-phs.org.uk/rogernewson*. This package uses, as input, a Stata data set (or a subset of a Stata data set), and cre-

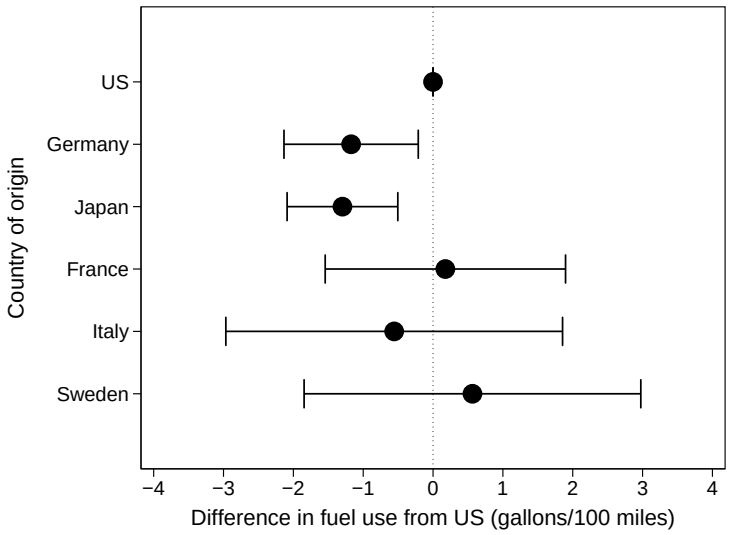

Figure 1: Differences in fuel use between cars from non-US countries and US-made cars.

ates, as output, a list of table rows, which may be inserted into a table in a TEX, LATEX, word processor, presentation or HTML document. These table rows may be echoed to the Stata log to be cut and pasted, written to a file to be linked or embedded, or both.

For instance, in the `auto` data, we might want to create tables of attributes of the 4 Volkswagen models in Microsoft Word and/or LATEX and/or HTML. The `listtex` package can create all 3 types of table rows, as follows:

```
. * Demonstrate listtex *
. listtex make mpg weight price if index(make,"VW ") == 1, type
VW Dasher&23&2,160&7,140
VW Diesel&41&2,040&5,397
VW Rabbit&25&1,930&4,697
VW Scirocco&25&1,990&6,850

. listtex make mpg weight price if index(make,"VW ") == 1, type rstyle(html)
<tr><td>VW Dasher</td><td>23</td><td>2,160</td><td>7,140</td></tr>
<tr><td>VW Diesel</td><td>41</td><td>2,040</td><td>5,397</td></tr>
<tr><td>VW Rabbit</td><td>25</td><td>1,930</td><td>4,697</td></tr>
<tr><td>VW Scirocco</td><td>25</td><td>1,990</td><td>6,850</td></tr>

. listtex make mpg weight price if index(make,"VW ") == 1, type rstyle(tabular)
VW Dasher&23&2,160&7,140\\
VW Diesel&41&2,040&5,397\\
VW Rabbit&25&1,930&4,697\\
VW Scirocco&25&1,990&6,850\\
```

The `listtex` command is executed 3 times, with the same `type` option (indicating output to the Stata log and the Results window) and different `rstyle()` options (indicating row style). In each case, `listtex` produces 4 lines of alien-looking output,

which appear in yellow in the Results window, whereas the commands appear in green. The first output has the default row style, and can be cut and pasted from the Results window into a Microsoft Word document, where it can be converted to the rows of a table using the menu sequence `Table->Convert->Text to Table` and specifying the ampersand as the text separation option. The second output can be cut and pasted into a HTML table, where it will be interpreted as table rows. The third output can be cut and pasted into an empty LaTeX `tabular` environment, which might be as follows:

```
\begin{tabular}{rrrr}
\hline
\textit{Make}&\textit{Mileage (mpg)}&\textit{Weight (lbs)}&\textit{Price (dollars)}\\
\hline
\hline
\end{tabular}
```

If the third `listtex` output is inserted between the fourth and fifth lines of this empty environment, then the resulting table might look like Table 1.

Table 1: Volkswagen cars in the `auto` data

Make	*Mileage (mpg)*	*Weight (lbs)*	*Price (dollars)*
VW Dasher	23	2,160	7,140
VW Diesel	41	2,040	5,397
VW Rabbit	25	1,930	4,697
VW Scirocco	25	1,990	6,850

In its current Stata 8 version, `listtex` has `headlines()` and `footlines()` options, which are an alternative way of generating the parts of TeX, LaTeX or HTML tables above and below the table rows. Many other row styles are possible, because the `rstyle()` option simply specifies a combination of 4 other options, `begin()`, `delimiter()`, `end()` and `missnum()`, specifying, respectively, a string beginning each row, a string delimiting columns, a string ending each row, and a string for missing numeric values. All of these options except `delimiter()` may be empty strings. `listtex` can therefore write table rows for many table styles, possibly including styles for TeX dialects not yet invented. In particular, `listtex` can specify TeX row styles with horizontal rules between rows and/or columns. (See the on-line help for `listtex` for an example.)

`listtex` is not the only Stata package for outputting Stata results as a table. For outputting a variable list to a TeX table, James Hardin's `textab` command creates a whole table (not only the rows), and provides a greater range of user-specifiable options than `listtex`, see Hardin (1995). Antoine Terracol's `outtex`, Marc–Andreas Muendler's `est2tex`, Antoine Terracol's `sutex` and Christopher F. Baum's `outtable` also output estimation results, estimation results (again), summary results and matrices, respectively, to TeX, and can be located for downloading using the Stata `findit` command. These packages are complementary to `listtex`, which has the advantage (and disadvantage) that the user has the freedom (and responsibility) to specify the row style and general

table format in an arbitrary way. For instance, many tables of confidence intervals, such as Table 5, have multiple header rows, some of which contain multi-column sets of merged cells, typically spanning the columns containing the estimates and the lower and upper confidence limits.

3 Creating data sets with one observation per parameter

To create a plot such as Figure 1 using `eclplot`, the user must first create a data set with one observation per confidence interval to be plotted, and data on the parameter estimates, confidence limits and other parameter attributes. Such data sets may be created initially by using the `parmest` package, downloadable from SSC, which creates these data sets from Stata estimation results, and is described in the first Subsection of this Section. The remaining Subsections describe two packages, also downloadable from SSC, which are often used together with `parmest`. These are `dsconcat`, which concatenates subsets of multiple Stata data sets, and `lincomest`, a version of `lincom` which saves the results as Stata estimation results which can then be used by `parmest`.

3.1 Saving estimation results as a data set using parmest

The `parmest` package was first published in Newson (1999) and updated in Newson (2000), The latest version on SSC, currently written in Stata 8, has evolved greatly since then, although Stata 7, 6 and 5 versions can still be downloaded from my website at *http://www.kcl-phs.org.uk/rogernewson* using either a browser or the `net` command. The package was written because it did not seem easy in Stata, at the time, to save estimation results to be plotted, or even to be listed to a consistent format, both of which were easily done using other software such as Genstat or SAS. The program `parmest` was my first response to this problem, and takes, as input, the currently available Stata estimation results, creating, as output, a data set with one observation per parameter. I now usually use the program `parmby`, also in the `parmest` package, which is a "quasi-byable" front end to `parmest`, and calls a user-specified estimation command to create an output data set with one observation per parameter, or one observation per parameter per by-group if the `by()` option is specified. With both programs, the output data sets may be listed to the Stata log and/or saved to disk and/or written to memory, overwriting any previously existing data set. More about Stata estimation commands and results can be found in the Stata manuals under [R] **estimates**, [P] **_estimates**, [P] **ereturn** and [U] **23 Estimation and post-estimation commands**.

The possible variables in the `parmest` and `parmby` output data sets are listed in Table 2. The subset of these variables that is actually created depends on various options mentioned in the online help for `parmest` and `parmby`. The variables `idnum` and `idstr` are only present if the user specifies the options `idnum()` and `idstr()`, respectively, and have the same value for all observations in the output data set. The variable `eq` is only present if the estimation results are from a command which creates estimation matrices with equation names, such as a command which fits a multiple equation model.

Table 2: Variables in the `parmest` and `parmby` output data sets

Name	*Numeric or String*	*Description*
`parmest` and `parmby`:		
`idnum`	Numeric	Numeric ID
`idstr`	String	String ID
`eq`	String	Equation name
`parm`	String	Parameter name
`label`	String	Parameter X-variable label
`ylabel`	String	Y-variable label
`estimate`	Numeric	Parameter estimate
`stderr`	Numeric	Standard error of parameter estimate
`dof`	Numeric	Degrees of freedom
`t`	Numeric	t–test statistic
`z`	Numeric	z–test statistic
`p`	Numeric	p–value
`stars`	String	Stars for p–value
`min`yy	Numeric	Lower xx% confidence limit
`max`yy	Numeric	Upper xx% confidence limit
`em_`y	String	yth macro estimation result requested
`es_`y	Numeric	yth scalar estimation result requested
`ev_`y	Numeric	yth vector estimation result requested
`parmby` only:		
By-variables	Either	Variables specified in the `by()` option
`parmseq`	Numeric	Parameter sequence number
`command`	String	Estimation command

The variable `label` contains the variable label of the X-variable corresponding to the parameter, and is only present if the `label` option is specified. The variable `ylabel` contains the variable label of the Y-variable, and is only present if the `ylabel` option is specified. Either the variables `dof` and `t` are present or the variable `z` is present, depending on whether the parameter estimates are assumed to have a t-distribution or a Normal distribution. The variable `p` contains p-values to test the hypothesis that the corresponding parameter is zero, or one if the `eform` option is specified. The variable `stars` is only generated if the user specifies a set of p-value thresholds in the `stars()` option. Pairs of xx% confidence limits `min`yy and `max`yy are calculated using a list of one

or more confidence levels xx, which may be specified in the `level()` option, and which defaults to the currently set confidence level. The number yy used to number the xx percent confidence limits `min`yy and `max`yy is equal to the confidence level xx unless the user specifies the option `clnumber(rank)`, in which case the lower and upper confidence limits are numbered `min1`, `min2`, ..., and `max1`, `max2`, ..., respectively, in ascending order of confidence level. The variables `em_`y, `es_`y and `ev_`y contain extra macro, scalar and vector estimation results, respectively, as and when specified by the user in the `emac()`, `escal()` and `evec()` options, respectively. The user can therefore save additional parameter attributes, such as the number of observations in the regression analysis as stored in `e(N)`, or the name of the Y-variable in the analysis as specified in `e(depvar)`. To find out what estimation results are available for a given estimation command, the user can type `ereturn list` in Stata 8, or `estimates list` in Stata 6 or 7, after the estimation command has been called. The by-variables are only present if the `by()` option of `parmby` is specified. The variable `command` is only present if the `command` option of `parmby` is specified. The variable `parmseq` contains the sequential order of the parameter in the model specification. All of these variables except the by-variables can be renamed using the `rename()` option, and this option can therefore be used to prevent name clashes involving the by-variables.

The `parmest` package can be used at different levels of sophistication. On the simplest level, we can use `parmest` with its `list()` option as a more powerful version of the `estimates table` command of Stata 8, and simply list the estimation results in a user-friendly way. For instance, in the `auto` data, we might compare the weights of non-American cars with those of American cars as follows:

```
. regress weight foreign, robust
Regression with robust standard errors                Number of obs =      74
                                                      F(  1,    72) =   56.52
                                                      Prob > F      =  0.0000
                                                      R-squared     =  0.3514
                                                      Root MSE      =  630.23
------------------------------------------------------------------------------
             |               Robust
      weight |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     foreign |  -1001.206   133.1696    -7.52   0.000    -1266.675   -735.7376
       _cons |   3317.115   96.81517    34.26   0.000     3124.118    3510.113
------------------------------------------------------------------------------
. parmest, label list(parm label estimate min* max* p) ///
>   format(estimate min* max* %8.2f p %8.2e)
```

parm	label	estimate	min95	max95	p
foreign	Car type	-1001.21	-1266.67	-735.74	1.2e-10
_cons	Constant	3317.12	3124.12	3510.11	2.6e-46

In this case, `parmest` lists the estimation results of the user's choice (using the `list()` option) to the user's chosen numbers of decimal places, or significant figures for the p-values (using the `format()` option), and labels the parameters (using the `label`

option). If we do this, then the `parmest` output data set is simply listed, and not saved, and the old data set remains in the memory.

On a less simple level, we might want to keep the `parmest` output data set in memory, overwriting the existing data set, and produce more advanced listings (or even plots). In the `auto` data, we might generate variables `gp100m=100/mpg` and `tons=weight/2240`, containing a car's fuel use in gallons/100 miles and weight in UK tons, respectively, and estimate the numbers of gallons consumed per 100 miles per incremental ton in US and non-US cars, using the following program:

```
parmby "regress gp100m tons, robust", by(foreign) escal(N) rename(es_1 N) ///
  format(estimate min* max* %8.2f p %8.2e) label norestore
bysort foreign N (parmseq): list parm label estimate min* max* p, clean noobs
```

We use `parmby` to fit a separate regression model for each car type, using the `escal(N)` option to create a variable `es_1` containing the extra scalar estimation result `e(N)` containing the number of observations for each by-group, and use the `rename()` option to rename `es_1` to the more understandable `N`. We set the formats as before, and use the `norestore` option to save the output data set in memory, overwriting the original data. This new data set is then listed, sorted by `foreign N (parmseq)`, or primarily by car type and secondarily by the order of specification of the parameter in the model:

```
. parmby "regress gp100m tons, robust", by(foreign) escal(N) rename(es_1 N) ///
>   format(estimate min* max* %8.2f p %8.2e) label norestore
Command: regress gp100m tons, robust
By variables: foreign

-------------------------------------------------------------------------------
-> foreign = Domestic

Regression with robust standard errors                Number of obs =      52
                                                      F(  1,    50) =  127.84
                                                      Prob > F      =  0.0000
                                                      R-squared     =  0.7651
                                                      Root MSE      =  .59936

------------------------------------------------------------------------------
             |               Robust
      gp100m |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        tons |    3.44974    .305102    11.31   0.000     2.836924    4.062555
       _cons |    .209591   .4055204     0.52   0.608    -.6049207    1.024103
------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-> foreign = Foreign

Regression with robust standard errors                Number of obs =      22
                                                      F(  1,    20) =   62.05
                                                      Prob > F      =  0.0000
                                                      R-squared     =  0.6679
                                                      Root MSE      =   .6758

------------------------------------------------------------------------------
             |               Robust
      gp100m |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
```

```
       tons |   4.838136   .6141827     7.88   0.000     3.556974    6.119299
      _cons |  -.6892427   .6774664    -1.02   0.321    -2.102413    .7239274
------------------------------------------------------------------------------

. bysort foreign N (parmseq): list parm label estimate min* max* p, clean noobs

-------------------------------------------------------------------------------
-> foreign = Domestic, N = 52
  parm           label   estimate   min95   max95        p
  tons   Weight (tons)       3.45    2.84    4.06   2.2e-15
 _cons        Constant       0.21   -0.60    1.02   6.1e-01

-------------------------------------------------------------------------------
-> foreign = Foreign, N = 22
  parm           label   estimate   min95   max95        p
  tons   Weight (tons)       4.84    3.56    6.12   1.5e-07
 _cons        Constant      -0.69   -2.10    0.72   3.2e-01
```

We note that the 52 US cars consume 3.45 gallons (95% CI, 2.84 to 4.06 gallons) per 100 incremental ton-miles, whereas the 22 non-US cars consume 4.84 gallons (95% CI, 3.56 to 6.12 gallons) per 100 incremental ton-miles.

We might be even more advanced, and store the `parmby` output to a disk file, leaving the existing data set unharmed:

```
parmby "regress gp100m tons, robust", by(foreign) escal(N) rename(es_1 N) ///
  format(estimate min* max* %8.2f p %8.2e) label saving(myparms,replace)
```

This time, we save the same output data set as before in a file `myparms.dta`, and have the option of returning to it later for further processing.

The `parmest` package is not the only way of creating a data set with one observation per parameter and data on confidence intervals and/or p-values. Not all confidence intervals are calculated using standard errors, as are those from Stata estimation results. The official Stata packages `centile` and `epitab` create conservative confidence intervals, using formulas based on exact distributions. Such confidence intervals can be saved to Stata data sets using the `postfile` package, or possibly using `statsby`.

3.2 Concatenating subsets of parameters using dsconcat

A `parmest` or `parmby` output contains parameters from only one set of estimation results, even though, in the case of `parmby`, there may be multiple by-groups. By contrast, most of the plots and tables I present in my work contain a subset of the parameters from multiple sets of estimation results. For instance, a medical scientist often presents a plot of confidence intervals for unadjusted odds ratios for a disease associated with a list of exposures, and also a similar plot of confidence intervals for the corresponding confounder-adjusted odds ratios for the same disease associated with the same list of exposures. These typically arise from a list of logistic regression models (one per exposure), with the disease as outcome and the appropriate exposure as the predictor, and a second list of logistic regression models (one per exposure), with the disease as outcome and the exposure and a list of confounders as predictors. The audience might have neither the time nor the inclination to view the confounder-associated odds ratios,

and probably will not even understand the baseline odds.

The `dsconcat` package concatenates subsets of the observations and/or the variables from a list of Stata data sets, and writes the resulting concatenated data set into the memory, overwriting any existing data set. This is especially useful if the user has created a list of `parmest` outputs, one for each of a list of regression models, and wants to collect, from each one, the parameters that the audience or readership want to see. The latest version is downloadable from SSC, although an earlier Stata 6 version can still be downloaded from *http://www.kcl-phs.org.uk/rogernewson*.

If all the interesting parameters are concatenated in one data set, then it is useful to know which parameters are from which regression model. Fortunately, `parmest` allows the user to do this, because some of the variables mentioned in Table 2 are constant within any one `parmest` output data set, and can be different in different `parmest` output data sets. The variables `idnum` and `idstr` contain numeric and string data set identifiers, respectively. The variable `ylabel` contains the variable label of the outcome variable. The variable `command` contains the whole estimation command called by `parmby`, complete with variable list, subsetting clauses and options, truncated if necessary to the maximum string variable length in the Stata version used. Finally, the extra macro-result variables `em_`y and the extra scalar-result variables `es_`y, generated by the `emac()` and `escal()` options respectively, are also the same for all observations in the same output data set, and have values copied from the estimation results.

For instance, in the `auto` data, after generating `gp100m` and `tons`, we might fit unadjusted and adjusted regression models using `tons` and `foreign` to predict `gp100m`, using the following program:

```
tempfile tf1 tf2 tf3
parmby "regr gp100m tons", lab saving(‘tf1’,replace) idn(1) ids(Unadjusted)
parmby "regr gp100m foreign", lab saving(‘tf2’,replace) idn(2) ids(Unadjusted)
parmby "regr gp100m tons foreign", lab saving(‘tf3’,replace) idn(3) ids(Adjusted)
dsconcat ‘tf1’ ‘tf2’ ‘tf3’
format estimate min* max* %8.2f
format p %8.2e
bysort idnum idstr (parmseq): list parm label estimate min* max* p, ///
  nocomp clean noobs
```

We use `parmby` 3 times, each time saving the output in a temporary file and using the `idnum` and `idstr` options to create numeric and string data set identifier variables. (Note that `parmby` does not always have a `by()` option.) We then use `dsconcat` to concatenate the 3 temporary output files into the memory (overwriting the existing data), add some formats to the confidence intervals and p-values, and list the concatenated data set as follows:

```
. bysort idnum idstr (parmseq): list parm label estimate min* max* p, ///
>   nocomp clean noobs

-------------------------------------------------------------------------------
-> idnum = 1, idstr = Unadjusted
         parm           label   estimate      min95      max95          p
         tons   Weight (tons)       3.15       2.70       3.60    3.7e-22
```

```
       _cons        Constant       0.77       0.14       1.40    1.7e-02

------------------------------------------------------------------------------
-> idnum = 2, idstr = Unadjusted
         parm           label   estimate      min95      max95          p
      foreign        Car type      -1.01      -1.61      -0.40    1.6e-03
       _cons        Constant       5.32       4.99       5.65    3.1e-44

------------------------------------------------------------------------------
-> idnum = 3, idstr = Adjusted
         parm           label   estimate      min95      max95          p
        tons   Weight (tons)       3.64       3.11       4.17    1.1e-21
      foreign        Car type       0.62       0.22       1.02    2.7e-03
       _cons        Constant      -0.07      -0.88       0.73    8.6e-01
```

Non-US cars consume fewer gallons per 100 miles than US cars, but more than US cars when adjusted for weight. Note that the numeric ID `idnum`, used mainly for sorting, is different in the 3 models, but the string ID `idstr` is used to distinguish the 2 unadjusted models from the adjusted model. This new concatenated data set can, of course, be saved in turn for future use.

In this case, `dsconcat` concatenated all observations from the output files, creating a data set in memory that included all the parameters. However, `dsconcat` has a `subset()` option, allowing it to include only a subset of observations and/or variables. The user could have typed

```
dsconcat 'tf1' 'tf2' 'tf3', subset(if parm != "_cons")
```

and excluded the uninteresting constant parameters, which represent the fuel consumption over 100 miles of hypothetical US-made cars with zero weight. This option is useful if the user has fitted a large number of regression models, each of which has many uninteresting confounder-related parameters.

3.3 Linear combinations using lincomest

We might want to include linear combinations in our data set of parameter estimates, especially as these are often the most interesting parameters estimated. Unfortunately, the `lincom` command cannot be used with `parmby` or `parmest`, as it is not an estimation command and saves no estimation results in `e()`. Fortunately, the `lincomest` program is a version of `lincom` which is an estimation command, and can be used with `parmest`.

In the `auto` data with the added variables `gp100m`, `tons` and `us = !foreign`, we might use the `somersd` package to measure association of US origin with the two added variables, and compare the two associations. The `somersd` package is described in Newson (2002), where it is used to measure and compare these two associations in the `auto` data. Using `parmby` and `lincomest`, we can carry out the same analysis using the following program:

```
. tempfile tf1 tf2
. parmby "somersd us tons gp100m", label saving('tf1',replace)
Command: somersd us tons gp100m
```

```
Somers' D with variable: us
Transformation: Untransformed
Valid observations: 74
Symmetric 95% CI
```

us	Coef.	Jackknife Std. Err.	z	P>|z|	[95% Conf.	Interval]
tons	.7508741	.0832485	9.02	0.000	.58771	.9140383
gp100m	.4571678	.135146	3.38	0.001	.1922866	.7220491

```
(note: file C:\WINDOWS\TEMP\ST_01000079.tmp not found)
file C:\WINDOWS\TEMP\ST_01000079.tmp saved
. parmby "lincomest (tons-gp100m)/2", label saving(‘tf2’,replace)
Command: lincomest (tons-gp100m)/2
Confidence interval for formula:
(tons-gp100m)/2
```

us	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
(1)	.1468531	.0442198	3.32	0.001	.0601838	.2335224

```
(note: file C:\WINDOWS\TEMP\ST_0100007a.tmp not found)
file C:\WINDOWS\TEMP\ST_0100007a.tmp saved
. dsconcat ‘tf1’ ‘tf2’
. format estimate min* max* %8.2f
. format p %8.2e
. list parm label estimate min* max* p, clean noobs
     parm                    label   estimate   min95   max95         p
     tons             Weight (tons)       0.75    0.59    0.91   1.9e-19
   gp100m   Fuel use (gallons/100m)       0.46    0.19    0.72   7.2e-04
      (1)           (tons-gp100m)/2       0.15    0.06    0.23   9.0e-04
```

We use `parmby` to call `somersd` and `lincomest`, each time saving the results to a temporary output file, and then concatenate the two output files into the memory to be formatted and listed. The parameters are two Somers' D values measuring the association between US origin and the 2 predictors (weight and fuel use), and half the difference between the two Somers' D values. The message is that weight is better than fuel consumption as a predictor of US origin, because, if one car uses less gas to move more mass and another uses more gas to move less mass, then the *first* car is more likely to be US-made. Note that, for a linear combination parameter, `parm` is set to `"(1)"`, and `label` is set to the linear combination formula.

4 Reformatting data with sencode, tostring, ingap and reshape

Output data sets created by the methods of the previous Section have the disadvantage that some variables are string and others are numeric. The end user, on the other hand, wants either plots or tables. If the user wants plots, then all variables plotted should be numeric. If the user wants tables, on the other hand, then all variables tabulated should

ideally be string, because confidence limits are usually tabulated with parentheses and commas, and p-values are sometimes tabulated with stars, and neither of these are provided by any Stata format at present. Also, we might like to add gaps between rows and column titles, or create tables with two or more columns of confidence intervals. The remaining Subsections of this Section show how these things can be done in Stata.

4.1 Encoding string variables to numeric with sencode

The `sencode` package is a "sequential" version of `encode`. It takes, as input, a string variable, and creates, as output, a numeric variable, with value labels equal, in each observation, to the input string variable. However, unlike `encode`, `sencode` encodes the string values to numbers in a non-alphanumeric order, which defaults to the order of appearance of the string value in the data set. The output numeric variable can be useful in creating plots, either as a by-variable or as an axis labelling variable.

Given the data set listed on Page 11, we might use `sencode` to create a plot as follows:

```
drop if parm == "_cons"
sencode idstr, gene(modtype)
sencode label, gene(predictor)
lab var modtype "Model type"
lab var predictor "Predictor"
sort modtype predictor
list modtype predictor estimate min* max* p, clean noobs
eclplot estimate min95 max95 predictor, hori ///
  estopts(msymbol(circle) msize(vlarge)) ciopts(msize(huge)) ///
  by(modtype,legend(off) compact) ///
  yscale(range(0 3)) ylabel(1 2, nogrid) ///
  xline(0,lpattern(shortdash)) xlabel(, format(%8.0f)) ///
  xtitle("Effect (gallons/100 miles)") ///
  xsize(4) ysize(3)
```

We first drop the uninteresting intercepts, and then encode sequentially the string variables `idstr` and `label` to give the labelled numeric variables `modtype` and `predictor`, respectively. We then sort the data set and list it, and it looks like this:

```
. list modtype predictor estimate min* max* p, clean noobs
      modtype       predictor   estimate   min95   max95         p
  Unadjusted   Weight (tons)       3.15    2.70    3.60   3.7e-22
  Unadjusted        Car type      -1.01   -1.61   -0.40   1.6e-03
    Adjusted   Weight (tons)       3.64    3.11    4.17   1.1e-21
    Adjusted        Car type       0.62    0.22    1.02   2.7e-03
```

We then create the confidence interval plot in Figure 2. Note that the labels of both `modtype` and `predictor` are coded non-alphabetically, in order of appearance in the data set.

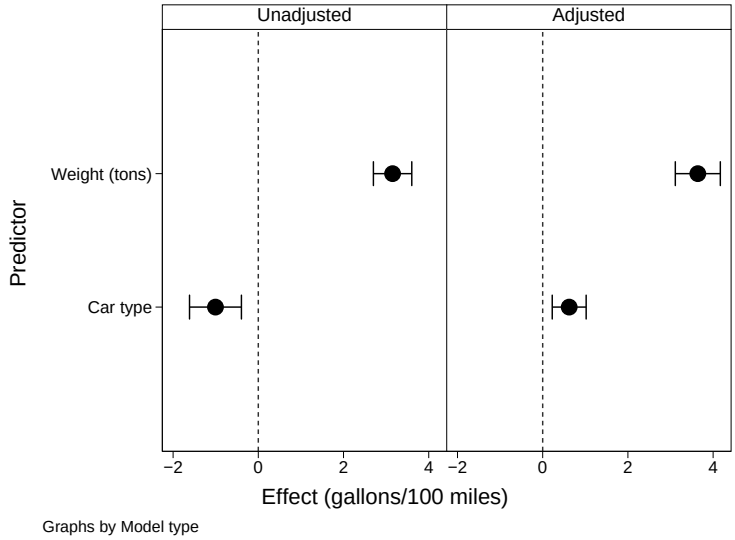

Figure 2: Effects of weight and car type on fuel use in the `auto` data.

4.2 Converting numeric variables to string using tostring

The `tostring` package was written by Nicholas J. Cox and Jeremy B. Wernow, who introduced it in Cox and Wernow (2000a) and updated it in Cox and Wernow (2000b). It is discussed in Cox (2002). The latest version is part of official Stata 8 as from 13 August 2003, although a Stata 6 version can still be downloaded by typing, in Stata,

```
net from http://www.stata.com/stb/stb57
```

and selecting the package `dm80_1`. Had `tostring` not existed, then I would have had to invent something similar. It replaces numeric variables with string variables of the same names, containing the formatted string values of the numbers. This is very useful when making tables of confidence intervals and p-values using `listtex`. For instance, using the output data set listed on Page 11, we might create a table as follows:

```
. drop if parm == "_cons"
(3 observations deleted)
. tostring estimate min* max* p, use replace force
estimate was double now str5
estimate was forced to string; some loss of information
min95 was double now str5
min95 was forced to string; some loss of information
max95 was double now str5
max95 was forced to string; some loss of information
p was double now str7
p was forced to string; some loss of information
. replace min95 = "(" + min95 + ","
```

```
min95 was str5 now str7
(4 real changes made)
. replace max95 = max95 + ")"
max95 was str5 now str6
(4 real changes made)
. replace p="$" + subinstr(p,"e"," \times 10^{",1) + "}$"
p was str7 now str21
(4 real changes made)
. listtex idstr label estimate min95 max95 p, type rstyle(tabular)
Unadjusted&Weight (tons)&3.15&(2.70,&3.60)&$3.7 \times 10^{-22}$\\
Unadjusted&Car type&-1.01&(-1.61,&-0.40)&$1.6 \times 10^{-03}$\\
Adjusted&Weight (tons)&3.64&(3.11,&4.17)&$1.1 \times 10^{-21}$\\
Adjusted&Car type&0.62&(0.22,&1.02)&$2.7 \times 10^{-03}$\\
```

We first drop the intercepts, and then use `tostring` to convert the numeric variables `estimate`, `min95`, `max95` and `p` to string. We then add parentheses and commas to the confidence limits, convert the p-values from the Stata `%8.2e` format to a TEX exponentiated form, and finally output the data set to LATEX `tabular` table rows using `listtex`. These table rows can be cut and pasted from the Stata log into a LATEX table to form Table 3.

Table 3: Effects of weight and car type on fuel use in the `auto` data

Model type	*Predictor*	*Estimate*	*(95%*	*CI)*	*p*
Unadjusted	Weight (tons)	3.15	(2.70,	3.60)	3.7×10^{-22}
Unadjusted	Car type	-1.01	(-1.61,	-0.40)	1.6×10^{-03}
Adjusted	Weight (tons)	3.64	(3.11,	4.17)	1.1×10^{-21}
Adjusted	Car type	0.62	(0.22,	1.02)	2.7×10^{-03}

4.3 Inserting gap rows using ingap

Table 3 might look better if, instead of having 2 repetitive "row label" columns on the left, it had a single, less repetitive row label column, as in Table 4. Similarly, Figure 2 might look better if the 4 confidence intervals were arrayed in a single column, as in Figure 3. In Stata 6 and 7, plots like Figure 3 are produced using the `hplot` package mentioned previously, with the `gaps()` and `glegend()` options.

To do the same in Stata 8, both for plots and for tables, I wrote the `ingap` package, which inserts gap observations in Stata data sets next to (before or after) a list of existing observations. `ingap` assigns values to the variables in the new gap observations according to a rule under which there are up to 3 classes of variables, namely the by-variables, a string row label variable (specified by the `rowlabel()` option), and the other variables. In a gap observation, the by-variables are assigned the same values as in the existing observation next to which the gap observation is inserted. The row label variable is assigned the appropriate value in a list of string values specified by

the `growlabel()` option. The other variables are assigned missing values, unless the `rstring()` option is specified, in which case the string variables are assigned values equal to their names or labels. (The `rstring()` option was added to allow the user to make title rows.)

The program to create the rows of Table 4 is as follows:

```
drop if parm=="_cons"
tostring estimate min* max* p,use replace force
replace min95 = "(" + min95 + ","
replace max95 = max95 + ")"
replace p = "$" + subinstr(p,"e"," \times 10^{",1) + "}$"
ingap 1 3, rowlabel(label) ///
  growlabel("Unadjusted effects:" "Adjusted effects:")
list label estimate min95 max95 p, clean noobs
listtex label estimate min95 max95 p, type rstyle(tabular)
```

We drop the intercepts and convert the estimates, confidence limits and p-values to string as before. We then use `ingap` to insert gap observations before existing observations 1 and 3, using `label` as the row label variable and specifying the values of `label` in the 2 gap observations to be `"Unadjusted effects:"` and `"Adjusted effects:"` respectively. Finally, we use `list` and `listtex` to list the data in human-readable and LaTeX-readable forms, respectively. The human-readable version is as follows:

```
. list label estimate min95 max95 p, clean noobs
                  label   estimate     min95    max95                       p
    Unadjusted effects:
          Weight (tons)       3.15    (2.70,    3.60)   $3.7 \times 10^{-22}$
               Car type      -1.01   (-1.61,   -0.40)   $1.6 \times 10^{-03}$
      Adjusted effects:
          Weight (tons)       3.64    (3.11,    4.17)   $1.1 \times 10^{-21}$
               Car type       0.62    (0.22,    1.02)   $2.7 \times 10^{-03}$
```

Note that, in the gap observations, the variable `label` has the values assigned by the `growlabel()` option, and the other variables have missing values. (There are no by-variables in this example.)

The program to create Figure 3 is as follows:

```
drop if parm == "_cons"
ingap 1 3, rowlabel(label) ///
  growlabel("Unadjusted effects:" "Adjusted effects:")
sencode label, gene(predictor) manyto1
lab var predictor "Predictor"
list predictor estimate min* max* p, clean noobs
eclplot estimate min95 max95 predictor, hori ///
  estopts(msymbol(circle) msize(vlarge)) ciopts(msize(huge)) ///
  yscale(range(0 7)) ylabel(1(1)6, nogrid) ///
  xline(0, lpattern(shortdash)) xlabel(, format(%8.0f)) ///
  xtitle("Effect (gallons/100 miles)") ///
  xsize(4) ysize(3)
```

This time, we drop the intercepts and then insert the gap observations as before, with `label` as the row label variable. We then sequentially encode `label` to create the

numeric variable `predictor`, using the `manyto1` option of `sencode`, because we want the mapping from numbers to labels to be many-to-one. (The numbers 2 and 5 both have the label `"Weight (tons)"`, and the numbers 3 and 6 both have the label `"Car type"`.) We then list and plot the data set. The listing is as follows:

```
. list predictor estimate min* max* p, clean noobs
             predictor   estimate   min95   max95         p
   Unadjusted effects:          .       .       .         .
         Weight (tons)       3.15    2.70    3.60   3.7e-22
              Car type      -1.01   -1.61   -0.40   1.6e-03
     Adjusted effects:          .       .       .         .
         Weight (tons)       3.64    3.11    4.17   1.1e-21
              Car type       0.62    0.22    1.02   2.7e-03
```

Note that, this time, the variables listed are all numeric, so they can be plotted, and the estimates, confidence limits and p-values have numeric missing values in the gap observations.

It is often a good idea to include a `preserve` and a `restore` before and after programs like these, which modify the data set in memory, so that the original data set is restored afterwards. This is especially true if the data set is to be modified in one way (using `sencode`) to produce plots and in another way (using `tostring`) to produce tables.

Table 4: Effects of weight and car type on fuel use in the `auto` data (using `ingap`)

Predictor	*Estimate*	*(95%*	*CI)*	*p*
Unadjusted effects:				
Weight (tons)	3.15	(2.70,	3.60)	3.7×10^{-22}
Car type	-1.01	(-1.61,	-0.40)	1.6×10^{-03}
Adjusted effects:				
Weight (tons)	3.64	(3.11,	4.17)	1.1×10^{-21}
Car type	0.62	(0.22,	1.02)	2.7×10^{-03}

4.4 Multi-column tables of confidence intervals using reshape

Scientists often create tables with more than one column of confidence intervals. An example is Table 5, which is a rearrangement of the confidence intervals of Table 4, with unadjusted and adjusted effects side by side. (The p-values have been dropped to save space, as often happens in medical journals.) In Stata, such tables can be produced using the `reshape` package, based on work in Weesie (1997), but now part of official Stata (see [R] **reshape**). The program to produce the rows of Table 5 is as follows:

```
drop if parm == "_cons"
sencode idstr, gene(modtype)
sencode label, gene(predictor)
```

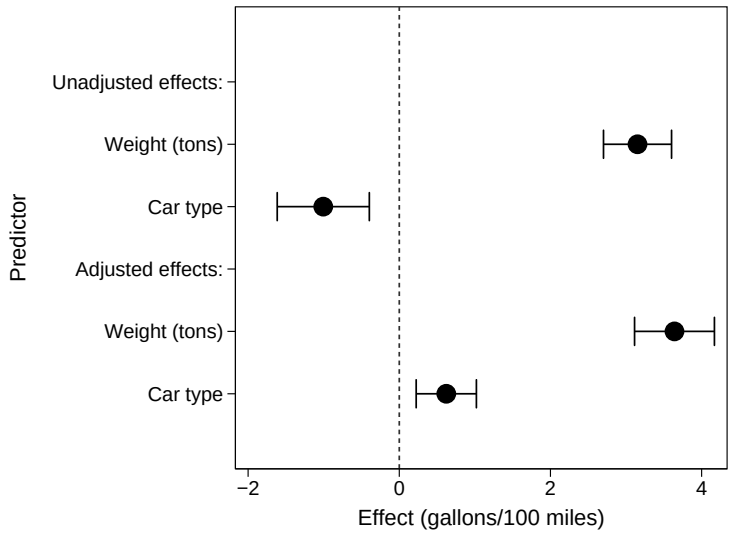

Figure 3: Effects of weight and car type on fuel use in the `auto` data (using `ingap`).

```
lab var modtype "Model type"
lab var predictor "Predictor"
keep modtype predictor estimate min* max*
tostring estimate min* max*, use replace force
replace min95 = "(" + min95 + ","
replace max95 = max95 + ")"
reshape wide estimate min95 max95, i(predictor) j(modtype)
list predictor estimate1 min951 max951 estimate2 min952 max952, clean noobs
listtex predictor estimate1 min951 max951 estimate2 min952 max952, ///
  type rstyle(tabular)
```

This time, we drop the intercepts, sequentially encode the two key variables specifying model type and predictor, drop all variables not wanted in the Table, convert the estimates and confidence limits to string, and then use `reshape wide` to reformat the data set to have 1 observation per value of `predictor` and two versions of each of the estimate and confidence limit variables. The resulting data set is listed in human-readable form as follows, before listing it in LaTeX-readable form:

```
. list predictor estimate1 min951 max951 estimate2 min952 max952, clean noobs
       predictor   estima~1    min951   max951   estima~2   min952   max952
   Weight (tons)       3.15    (2.70,    3.60)       3.64   (3.11,    4.17)
        Car type      -1.01   (-1.61,   -0.40)       0.62   (0.22,    1.02)
```

Table 5: Effects of weight and car type on fuel use in the `auto` data (using `reshape`)

	Unadjusted effects:			*Adjusted effects:*		
Predictor	*Est.*	*(95%*	*CI)*	*Est.*	*(95%*	*CI)*
Weight (tons)	3.15	(2.70,	3.60)	3.64	(3.11,	4.17)
Car type	-1.01	(-1.61,	-0.40)	0.62	(0.22,	1.02)

5 Reconstructing factors using descsave and factext

In Figure 1, the confidence intervals are plotted, not against a numeric variable created by using `sencode` on the variable `label` in a `parmest` output, but against a categorical variable `country`, which was added to the original `auto` data. Categorical variables are usually included in regression models using the `xi` package of official Stata (see [R] **xi**) to create dummy indicator variables. However, such dummy indicator variables may also be created using the official Stata command `tabulate` with the `generate()` option (see [R] **tabulate**), or by John Hendrickx's `desmat` package, introduced in Hendrickx (1999), updated in Hendrickx (2000), Hendrickx (2001a) and Hendrickx (2001b), and downloadable in its latest version from SSC. Whichever package is used to create dummy variables from a categorical variable, the original categorical variable can be reconstructed in a `parmest` output data set using the `descsave` and `factext` packages, downloadable from SSC.

The `descsave` package is an extended version of `describe` (see [R] **describe**), and produces up to 2 output files. One of these is a Stata data set with 1 observation per variable and data on the variable's name, type, format, variable label, value label, and (optionally) characteristics specified by the user (see [P] **char**). The other is a Stata program file, which can be run to reconstruct all these variable attributes in a data set in which a variable exists with the same name and mode (numeric or string). Such a second data set might be created using the official Stata command `insheet` (see [R] **insheet**), but may, alternatively, be a `parmest` output.

The `factext` package contains the programs `factext`, `factref` and `factmerg`, all of which are intended for use in an output data set created using the `label` option of `parmest` or `parmby` to generate the variable `label` mentioned in Table 2. If the regression model fitted included dummy variables created by `xi`, `tabulate` or `desmat`, then the labels of these dummy variables will typically be of the form

"*categorical_variable_name*==*value*"

and will be stored in the variable `label`, as will the labels of any other X-variables. The `factext` program reads the labels from `label` and re-generates the original categorical variables, optionally using a do-file created by `descsave` to reconstruct the names, types, formats, variable labels, value labels and characteristics of these categorical variables. The `factref` program inserts extra observations with the reference values of

categorical variables, which must be specified by the `omit` characteristic (see [R] **xi** and [R] **char**). The `factmerg` program may be useful for creating string row label variables. It "merges" a list of input categorical variables to create string variables containing, in each observation, the name, label and string-mode value, respectively, of the first variable in the input list to have a non-missing value in that observation. The reason for the `fact` prefix in these program names is that the author used to work with the packages GLIM and Genstat, originating from Rothamsted Experimental Station, UK, and, in these packages, a categorical variable is known as a factor.

5.1 An example in the auto data

The program to create Figure 1 can now be revealed. Its log is as follows:

```
. * Plot graph of CIs for gp100m by country *
. preserve
. char country[omit] 1
. tempfile tf0
. descsave country, char(omit) do('tf0')
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
country         byte   %9.0g       country    Country of origin
. parmby "xi:regress gp100m i.country", float label ///
>  format(estimate min* max* %8.2f) norestore
Command: xi:regress gp100m i.country
i.country         _Icountry_1-6       (naturally coded; _Icountry_1 omitted)
      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  5,    68) =    3.10
       Model |  22.2142794     5  4.44285589           Prob > F      =  0.0139
    Residual |  97.3620073    68  1.43179423           R-squared     =  0.1858
-------------+------------------------------           Adj R-squared =  0.1259
       Total |  119.576287    73  1.63803133           Root MSE      =  1.1966

------------------------------------------------------------------------------
     gp100m |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
 _Icountry_2 |  -1.172866   .4817432    -2.43   0.018    -2.134169    -.211562
 _Icountry_3 |  -1.297032   .3971116    -3.27   0.002    -2.089455   -.5046077
 _Icountry_4 |   .1763502   .8622248     0.20   0.839    -1.544193    1.896893
 _Icountry_5 |  -.5562508   1.208027    -0.46   0.647    -2.966831    1.854329
 _Icountry_6 |   .5641973   1.208027     0.47   0.642    -1.846383    2.974777
       _cons |   5.318156   .1659352    32.05   0.000     4.987037    5.649274
------------------------------------------------------------------------------
. factext country, do('tf0')
. factref country, rzero(estimate min* max*)
. desc
Contains data from C:\WINDOWS\TEMP\ST_01000060.tmp
  obs:             7
 vars:            11                          25 Jul 2003 14:37
 size:           364 (99.9% of memory free)
------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
```

```
-------------------------------------------------------------------------------
parmseq         byte   %12.0g                 Parameter sequence number
parm            str11  %11s                   Parameter name
label           str10  %10s                   Parameter label
estimate        float  %8.2f                  Parameter estimate
stderr          float  %10.0g                 SE of parameter estimate
dof             byte   %10.0g                 Degrees of freedom
t               float  %10.0g                 t-test statistic
p               float  %10.0g                 P-value
min95           float  %8.2f                  Lower 95% confidence limit
max95           float  %8.2f                  Upper 95% confidence limit
country         byte   %9.0g       country    Country of origin
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. list parm label country estimate min95 max95,clean

             parm        label   country   estimate   min95   max95
  1.                                   US       0.00    0.00    0.00
  2.   _Icountry_2   country==2   Germany      -1.17   -2.13   -0.21
  3.   _Icountry_3   country==3     Japan      -1.30   -2.09   -0.50
  4.   _Icountry_4   country==4    France       0.18   -1.54    1.90
  5.   _Icountry_5   country==5     Italy      -0.56   -2.97    1.85
  6.   _Icountry_6   country==6    Sweden       0.56   -1.85    2.97
  7.         _cons     Constant         .       5.32    4.99    5.65

. eclplot estimate min95 max95 country if parm != "_cons", hori ///
>  estopts(msymbol(circle) msize(vlarge)) ciopts(msize(huge)) ///
>  yscale(range(0 7)) ylabel(1(1)6) ///
>  xlabel(-4(1)4,format(%8.0f)) xline(0, lpattern(dot)) ///
>  xtitle("Difference in fuel use from US (gallons/100 miles)") ///
>  xsize(4) ysize(3)

. graph export figseq1.eps, replace
(file figseq1.eps written in .eps format)

. more

. restore
```

We start in the `auto` data with the added variable `country`. We first set the `omit` characteristic of `country` to 1 (US cars). We then use `descsave` with the `do()` option to create a temporary do-file containing commands to reconstruct the variable attributes of the variable `country`, including the `omit` characteristic (see [R] **char** and [R] **xi**). Then we use `parmby` with the `norestore` and `label` options to run a regression model, using `xi` to create dummy variables for `country`. This creates a `parmest` output data set in memory, overwriting the pre-existing `auto` data. Using `factext`, we add to this data set an additional variable `country`, using the temporary do-file created previously to reconstruct the variable attributes of the variable of the same name in the pre-existing data set. We then use `factref` to insert an extra observation in the data set with the reference category value for `country` (as stored in the `omit` characteristic), and with zero values for the variables listed in the `rzero()` option, and with missing values for all other variables. The data set is then displayed, using `describe` and `list` to display the variables and observations, respectively. Note that the variable `label` contains `"Constant"` for the intercept, `xi`-format variable labels for parameters corresponding to dummy variables, and a missing value for the reference-value observation for US-made cars inserted by `factref`. Finally, we use `eclplot` and `graph export` (see [G] **graph**

export) to create Figure 1 before restoring the pre-existing `auto` data to the memory.

6 Acknowledgements

I would like to thank Nicholas J. Cox for writing the `hplot` package, from which I discovered a lot of the ideas behind the packages presented here, and which was my usual package for graphing the confidence intervals from which I make my living, before I had access to Stata 8. I would also like to thank Nicholas J. Cox and Jeremy Wernow for writing `tostring`, Nicholas J. Cox and Patrick Joly for their helpful suggestions on improvements for `sencode`, Jeff Pitblado for his helpful programming advice on improving `listtex`, and Dan Blanchette, Nicholas J. Cox, William Gould, Ken Higbee, Jeff Pitblado, Philip Ryan, Vince Wiggins, Nicholas Winter and Fred Wolfe for their helpful suggestions and advice on improving and certifying the `parmest` package.

7 References

Cox, N. J. 2002. Speaking Stata: On numbers and strings. *The Stata Journal* 2(3): 314–329.

Cox, N. J. and J. Wernow. 2000a. dm80: Changing numeric variables to string. *Stata Technical Bulletin* 56: 8–12. In *Stata Technical Bulletin Reprints*, vol. 10, 24–28. College Station, TX: Stata Press.

—. 2000b. dm80.1: Update to changing numeric variables to string. *Stata Technical Bulletin* 57: 2. In *Stata Technical Bulletin Reprints*, vol. 10, 28–29. College Station, TX: Stata Press.

Hardin, J. 1995. dm29: Create TeX tables from data. *Stata Technical Bulletin* 25: 3–7. In *Stata Technical Bulletin Reprints*, vol. 5, 20–25. College Station, TX: Stata Press.

Hendrickx, J. 1999. dm73: Using categorical variables in Stata. *Stata Technical Bulletin* 52: 2–8. In *Stata Technical Bulletin Reprints*, vol. 9, 51–59. College Station, TX: Stata Press.

—. 2000. dm73.1: Contrasts for categorical variables: update. *Stata Technical Bulletin* 54: 7. In *Stata Technical Bulletin Reprints*, vol. 9, 60–61. College Station, TX: Stata Press.

—. 2001a. dm73.2: Contrasts for categorical variables: update. *Stata Technical Bulletin* 59: 2–5. In *Stata Technical Bulletin Reprints*, vol. 10, 9–14. College Station, TX: Stata Press.

—. 2001b. dm73.3: Contrasts for categorical variables: update. *Stata Technical Bulletin* 61: 5. In *Stata Technical Bulletin Reprints*, vol. 10, 14–15. College Station, TX: Stata Press.

Newson, R. 1999. dm65: A program for saving a model fit as a dataset. *Stata Technical Bulletin* 49: 2–5. In *Stata Technical Bulletin Reprints*, vol. 9, 19–23. College Station, TX: Stata Press.

—. 2000. dm65.1: Update to a program for saving a model fit as a dataset. *Stata Technical Bulletin* 58: 2. In *Stata Technical Bulletin Reprints*, vol. 10, 7. College Station, TX: Stata Press.

—. 2002. Parameters behind "nonparametric" statistics: Kendall's tau, Somers' D and median differences. *The Stata Journal* 2(1): 45–64.

Weesie, J. 1997. dm48: An enhancement of `reshape`. *Stata Technical Bulletin* 38: 2–4. In *Stata Technical Bulletin Reprints*, vol. 7, 40–43. College Station, TX: Stata Press.

Wiener, N. 1988. *The Human Use of Human Beings: Cybernetics and Society*. 2d ed. Cambridge, MA: DaCapo Press.

About the Author

Roger Newson is a Lecturer in Medical Statistics at King's College London, UK, working primarily in asthma research. He wrote the unofficial Stata packages `eclplot`, `listtex`, `parmest`, `dsconcat`, `lincomest`, `sencode`, `ingap`, `descsave` and `factext`, presented here.