From datasets to resultssets in Stata

Roger Newson
King's College London, London, UK
roger.newson@kcl.ac.uk
http://www.kcl-phs.org.uk/rogernewson/

Distributed at the 10th UK Stata Users' Group Meeting on 29 June 2004

1 What are resultssets?

A **resultsset** is a Stata dataset created as output by a Stata command. It may be simply listed to the Stata log and/or output to a disk file and/or written to the memory, overwriting any pre-existing dataset. If you are a SAS user converting to Stata, then note that Stata datasets do the job of SAS data sets, and Stata resultssets do the job of SAS output data sets.

1.1 Why resultssets?

In general, a Stata dataset, or a dataset in any other format, should contain one observation per *thing*, and data on *attributes_of_things*. For instance, in the medical sector, a dataset might have one observation per patient, and data on the patient's baseline characteristics. Alternatively, a dataset might have one observation per visit to a health centre, and data on the state of the patient who made the visit. Usually, the variables in a dataset are either **primary key variables**, which identify the things corresponding to the observations, or **non-key variables**, which identify interesting attributes of those things. For instance, if there is one observation per patient, then one of the variables is usually a patient ID number, which identifies the observation uniquely. Or, if there is one observation per visit, and a patient may only make one visit per day, then the primary key variables are usually patient ID and visit date.

Statisticians (and other data analysts) are typically provided with (or collect for themselves) a dataset with one observation per "experimental or observational unit", where a unit may be a patient, a patient-day, a country-year, a car model, or some other thing. However, they are typically paid to produce plots for presentation, or tables for publication. To do this, they really need datasets with one observation per plotted data point, or per Y-axis label, or per X-axis label, or per table row. Axis labels and table rows do not often correspond to the original units in the original dataset.

For instance, in the `auto` data, shipped with official Stata, there is one observation per car model, and the primary key is the single variable `make`. Figure 1 gives confidence intervals from a regression model measuring differences in fuel efficiency (in miles per gallon) in cars from inside and outside the US, with various 1978 repair records, compared with a "reference car", made by a US company and with a repair record of 3. It was created using the `eclplot` package, which creates confidence interval plots, and requires, as input, a dataset with one observation per confidence interval to be plotted and data on estimates and confidence limits. Table 1 gives the same data (plus P-values) as a table. It was created using the `listtex` package, which creates tables which can be cut and pasted into a TeX, LaTeX, HTML or word processor document, and requires, as input, a dataset with one observation per table row. Both `eclplot` and `listtex` are downloadable from SSC, but neither can use directly as input the original `auto` data, with one observation per car model. Instead, resultsset-generating and resultsset-processing packages are used, taking as input the original `auto` data, and creating, as output, datasets that can be input to `eclplot` and `listtex`.

This survey explains how to do this, using example do-files distributed with this document on the conference website at *http://www.stata.com/support/meeting/10uk/*. These do-files will work under Stata 8 if the user has installed the required unofficial Stata packages, listed at the top of each do-file. These packages are downloadable from the SSC archive at *http://ideas.repec.org/s/boc/bocode.html* using the Stata `ssc` command. The full list of required packages comprises the resultsset-generating packages `descsave`, `parmest`, `xcollapse` and `xcontract`, together with the resultsset-processing packages `eclplot`, `listtex`, `dsconcat`, `sencode`, `sdecode`, `factext`, `factmerg` and `ingap`, and the estimation package `somersd`.

Figure 1: Differences in mileage in the `auto` data (compared to US car with `rep78==3`).

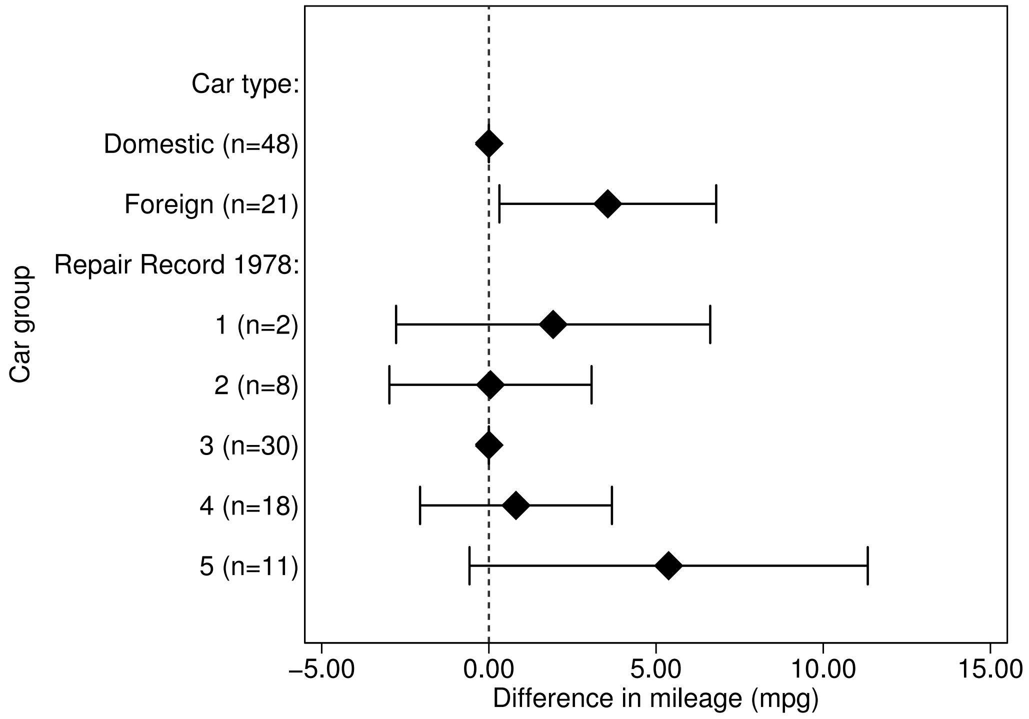


Table 1: Differences in mileage in the `auto` data (compared with US car with `rep78==3`).

Car group	*Difference (mpg)*	*(95%*	*CI)*	*P*
Car type:				
Domestic (n=48)	0.00	(ref.)		
Foreign (n=21)	3.56	(0.32,	6.80)	.032
Repair Record 1978:				
1 (n=2)	1.92	(−2.78,	6.62)	.42
2 (n=8)	0.05	(−2.98,	3.07)	.98
3 (n=30)	0.00	(ref.)		
4 (n=18)	0.81	(−2.06,	3.68)	.57
5 (n=11)	5.38	(−0.58,	11.33)	.076

2 Resultsset-generating programs

Some programs which generate resultssets are summarized in Table 2. The `descsave` package is an extended version of `describe` (see [R] **describe**), and creates a resultsset with one observation per variable, and data on variable attributes, which are the names, storage types, display formats, value labels and variable labels of the variables, and (optionally) characteristics (see [U] **15.8 Characteristics** or [P] **char**). The `parmest` package contains two programs, `parmest` and `parmby`. `parmest` creates a resultsset with one observation per parameter of the most recent set of estimation results (see [U] **29 Estimation and post-estimation commands** or [P] **ereturn**), and data on general parameter attributes, which include the names, estimates, standard errors, confidence limits and P-values of the parameters. `parmby` calls a user-specified estimation command once for each by-group (or once only, if the `by()` option is absent), and creates a resultsset with one observation per parameter (or per parameter per by-group, if the `by()` option is present), and data on the same parameter attributes as output by `parmest`. `xcollapse` and `xcontract` are extended versions of the official Stata programs `collapse` and `contract`, respectively (see [R] **collapse** and [R] **contract**). SAS programmers may note that `descsave` does the job of PROC CONTENTS in SAS, `xcontract` does the job of PROC FREQ, `xcollapse` does the job of PROC MEANS and PROC SUMMARY, and `parmest` and `parmby` do the job of the OUTEST= option of SAS estimation procedures. The other resultsset-generating programs are part of official Stata, and are documented in [R] **statsby**, [R] **bootstrap**, [R] **simulate** and

[P] **post**, respectively. `statsby` calls a Stata command once for each by-group, and creates a resultsset with one observation per by-group and data on general statistical results stored by the called Stata command in its returned results in `r()` or `e()` (see [R] **Saved results**, [P] **return** and [P] **ereturn**). The `post` package is a general low-level utility, allowing users to write their own resultsset-generating programs.

Table 2: Some resultsset-generating programs available in Stata.

	The resultsset contains:	
Program	*one observation per...*	*and data on...*
Downloadable from SSC:		
`descsave`	variable	variable attributes
`parmest`	estimated parameter	parameter attributes
`parmby`	parameter per by-group	parameter attributes
`xcollapse`	by-group	basic summary statistics
`xcontract`	combination of variable values	frequencies and percentages
Official Stata:		
`collapse`	by-group	basic summary statistics
`contract`	combination of variable values	frequencies
`statsby`	by-group	general statistics
`bootstrap`	bootstrap sample	bootstrap sample statistics
`simulate`	simulation replication	simulation results
`post`	anything	anything

The program in `example1.do` demonstrates the generation of resultssets by `descsave`, `parmby`, `xcollapse` and `xcontract`, if the user has installed the required packages (listed at the top of the do-file). It uses as input the `auto` data, shipped with Stata and accessed using the `sysuse` command. The user can open it in the Stata do-file editor, click on the icon labelled "Do current file", and return to the Stata Command window. Alternatively, the user can type the command `do example1` in the Stata Command window. Each resultsset-generating program generates a resultsset which is saved to the memory, overwriting the pre-existing data. In each case, the program lists the variables in the resultsset using `describe`, and lists the observations in the resultsset using `list`. The `more` command allows the user to look at the variables or observations before pressing the space bar, which allows the program to continue. Variables in the resultssets are discussed in the online help for each command, and in Newson (2003) in the case of `parmby` and `parmest`.

2.1 Common options for resultsset-generating programs

Different resultsset-generating programs often have options of the same names with the same functions. Some commonly-used options are listed in Table 3. Their exact syntax is given in the online help for each program, although only `parmby` has them all.

Table 3: Common options for resultsset-generating programs.

Option	*Function*
Resultsset-destination options:	
`list()`	List resultsset to log and/or Results window
`saving()`	Save resultsset to disk file
`norestore` (or `replace`)	Overwrite any existing data in memory
`fast`	Fast version of `norestore` for programmers
`flist()`	Global macro accumulating `saving()` filenames
Resultsset-modifying options:	
`rename()`	Rename variables in resultsset from default names
`format()`	Give variables in resultsset nondefault formats
`idnum()`	Value of numeric resultsset ID variable
`idstr()`	Value of string resultsset ID variable
`by()`	Specify by-groups

The first five options specify the destination of the resultsset, and are not mutually exclusive. The `list()` option specifies that the resultsset (or a subset of its variables and/or observations) is listed to the Results window and/or the Stata log, in a style that can be customized by the user using suboptions. `saving()` specifies that a resultsset is saved to a disk file. `norestore`, used in `example1.do`, specifies that the resultsset is saved to memory, overwriting any existing dataset. `fast` is a high-speed version of `norestore`, mostly for advanced programmers, which takes no action to restore the existing data if the user presses `Break`. `flist()` is used with `saving()`, and specifies the name of a global macro, to which the filename of the resultsset file will be appended, in order to accumulate a list of resultsset filenames.

Resultssets saved to disk can be used as the `using` dataset by `append`, `merge`, `joinby` and `cross`, in the same way as other disk datasets. (See [U] **25 Commands for combining data** for more on these commands.) A particularly useful package is `dsconcat`, downloadable from SSC. This takes as input a list of filenames of disk datasets, and concatenates the observations in these datasets, or a subset of these observations and/or of the variables, into the memory, overwriting any existing data. The input datasets are often resultssets, especially `parmby` resultssets. The list of input files may be stored in a global macro specified by the `flist()` option, and an example of this practice appears in the online help for `parmest`. `dsconcat` is useful if the user estimates parameters for more than one regression model, and wishes to plot or tabulate all these parameters, or an interesting subset of them, in a single plot or table. This ability is an advantage of the resultsset method over alternative and complementary methods of creating publication-ready tables of regression results, using official Stata's `estimates table` (see [R] **estimates**), or using John Gallup's `outreg` or Tony Brady's `reformat`, both downloadable from SSC. On the other hand, the complementary methods are more instant solutions, if the user only wants to tabulate results from one regression model at a time, does not require plots, and is publishing in a journal which prefers the specific format of `estimates table`, `outreg` or `reformat`.

The last five options modify the variables in the resultsset. The `rename()` and `format()` options specify nondefault names and formats, respectively, for variables in the resultsset. The `idnum()` and `idstr()` options specify values for a numeric and string resultsset ID variable, respectively, both of which have the same value for all observations in the resultsset. If we concatenate multiple input resultssets, then it is useful to know which input resultsset each observation came from, and the `idnum()` and `idstr()` options make this easy. The `by()` option specifies a list of by-variables, specifying by-groups. The resultsset will then be concatenated at birth, and contain a "resultssubset" of observations for each by-group.

2.2 Resultssets are often stored in tempfiles

Resultssets on disk are very often stored in temporary files, produced using either the `tempfile` command or the `tempfile` extended macro function. In the Stata world, temporary files are usually viewed as a technical specialist subject, mostly of interest to advanced programmers, mentioned briefly in [U] **21.7 Temporary objects**, and documented more fully in [P] **macro**. This is in contrast to the SAS world, where users typically learn about temporary SAS data sets before learning about permanent SAS data sets. Stata has the advantage over SAS that it can do much more data handling in memory. However, this advantage is limited by the fact that it can only do this with one dataset at a time. It is therefore a good idea for Stata users to know at least enough about macros to be able to handle `tempfile`s.

The program in `example2.do` demonstrates the use of `tempfile` in combination with `dsconcat`, `descsave` and `xcollapse`. After loading the `auto` data, we use the `tempfile` command to tell Stata that we intend to create temporary files with macro names `tf0` to `tf9`, and create the first one as a `descsave` resultsset, with one observation for each of 9 quantitative variables, which we might want to compare between US and non-US cars. This resultsset is sorted by variable name using the `gsort()` option, and stored on disk using the `saving()` option. We then create the other 9 temporary files as `xcollapse` resultssets, each with 1 observation per car type (US and non-US) and data on the median value of one of the 9 quantitative variables. Each `xcollapse` resultsset has a string ID variable named `idstr`, identifying the quantitative variable to which the medians in that resultsset belong. We then concatenate the resultssets into the memory using `dsconcat`, and `describe` the variables in the new concatenated dataset in memory, and `list` the observations. Finally, we rename the string ID variable `idstr` to `name`, sort the dataset by it, merge in the `descsave` resultsset, and create a new dataset in memory, with one observation per quantitative variable per car type, sorted first by car type and second by the order of the quantitative variable in the list. This dataset is listed, so that we can compare the medians of the 9 quantitative variables between US and non-US cars. This final listing should look like this:

```
. by foreign: list order name varlab median, noobs

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

```
  +-----------------------------------------------------------+
  | order          name   varlab                       median |
  |-----------------------------------------------------------|
  |     1         price   Price                        4782.5 |
  |     2           mpg   Mileage (mpg)                    19 |
  |     3      headroom   Headroom (in.)                  3.5 |
  |     4         trunk   Trunk space (cu. ft.)            16 |
  |     5        weight   Weight (lbs.)                  3360 |
  |-----------------------------------------------------------|
  |     6        length   Length (in.)                    200 |
  |     7          turn   Turn Circle (ft.)                42 |
  |     8  displacement   Displacement (cu. in.)          231 |
  |     9    gear_ratio   Gear Ratio                     2.75 |
  +-----------------------------------------------------------+

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

  +-----------------------------------------------------------+
  | order          name   varlab                       median |
  |-----------------------------------------------------------|
  |     1         price   Price                          5759 |
  |     2           mpg   Mileage (mpg)                  24.5 |
  |     3      headroom   Headroom (in.)                  2.5 |
  |     4         trunk   Trunk space (cu. ft.)            11 |
  |     5        weight   Weight (lbs.)                  2180 |
  |-----------------------------------------------------------|
  |     6        length   Length (in.)                    170 |
  |     7          turn   Turn Circle (ft.)                36 |
  |     8  displacement   Displacement (cu. in.)          101 |
  |     9    gear_ratio   Gear Ratio                     3.61 |
  +-----------------------------------------------------------+


. more
```

This time, the 9 quantitative variables are identified, not only by their names, but also by their more informative labels (with units), merged in from the `descsave` resultsset. Examples of processing multiple `parmby` resultssets can be found in Newson (2003), and in the online help for `parmest`.

3 String-numeric conversion in resultssets

A lot of the work in resultsset processing is converting string variables to numeric variables and/or numeric variables to string variables. To be plotted, a variable must *usually* be numeric, even if it corresponds to string axis labels, such as those on the vertical axis of Figure 1. On the other hand, if we want to perform string substitutions on a numeric variable and/or add prefixes or suffixes, then the variable must be converted to string. For instance, we might want to tabulate confidence limits with parentheses and commas (as in Table 1), or add stars to P-values, and neither of these is provided by any existing Stata display format. In practice, it is common to present the same results both as plots (for presentation in meetings) and as tables (for publication in journals). It is therefore important to be able to convert variables between string and numeric with a minimum of programming by the user.

Official Stata provides the utilities `encode` and `decode` for conversion between string variables and numeric variables with value labels, and the utilities `destring` and `tostring` for conversion between string variables and numeric variables without value labels. These utilities were discussed by Cox (2002). They all have limitations, especially `encode`, which encodes string variables to numeric in an alphanumeric order. For this reason, I have developed the programs `sencode` and `sdecode` for basic conversions between single numeric variables and single string variables, and the programs `factext` and `factmerg` for string-factor conversions, which often involve multiple string variables and/or multiple numeric variables converted in parallel. These programs are all downloadable from SSC.

3.1 Basic string-numeric conversion using sencode and sdecode

The programs `sencode` and `sdecode` are "super" versions of `encode` and `decode`, respectively, and convert string variables to labelled numeric variables and labelled numeric variables to string variables, respectively. With both programs, the output variable may either be generated as a new variable or replace the input variable, taking over its name, variable label, position in the dataset and characteristics (see [U] **15.8 Characteristics** or [P] **char**).

`sencode` takes, as input, a string variable, and generates, as output, a numeric variable, whose value label in each observation is equal to the value of the input string variable in the same observation. It has a `gsort()` option, enabling the user to define numeric codes in ascending or descending order of a user-specified list of variables, as recognised by the `gsort` command (see [R] **gsort**). `sencode` groups the observations into ordered groups according to the `gsort()` option, and then generates an encoded integer value for each combination of `gsort()` group and input string value, ordered first by `gsort()` group and second by alphanumeric order of input string values within each `gsort()` group. Usually, each `gsort()` group contains only one input string value, so the encoding is in ascending order of `gsort()` group. If the `gsort()` option is not specified, then it is set in default to be equivalent to the Stata expression `_n`, specifying a `gsort()` group for each observation, ordered by the position of that observation in the dataset. There is also a `manyto1` option, allowing the relation between output numeric values and input string values to be many-to-one. Therefore, it is possible for multiple observations to have the same output numeric value, and for multiple output numeric values to have the same input string value. In the simplest case, if `gsort()` is not specified, an observation has an output numeric value depending on its position in the dataset (if `manyto1` is specified), or on the position in the dataset of the first appearance of its input string value (if `manyto1` is not specified).

`sdecode` takes, as input, a numeric variable, and generates, as output, a string variable. In default, the value of the output string variable in each observation is equal to the value label of the input numeric value in that observation (if such a value label exists), or to the formatted string value of the input numeric value (if the input numeric value is nonmissing and has no label), or to an empty string (if the input numeric value is missing). `sdecode` therefore converts numbers to strings in a similar way to `tabulate`.

The program in `example3.do` uses the same quantitative variables as the program in `example2.do`. However, instead of comparing medians between US and non-US cars, we now compare the quantitative variables as diagnostic predictors of the "condition" of non-US origin. We do this using the `somersd` package, introduced by Newson (2000) and discussed by Newson (2002), which computes confidence intervals for the Somers' D parameter of each quantitative predictor with respect to non-US origin. For each quantitative diagnostic predictor, Somers' D is related to the area under the specificity-sensitivity or receiver-operating characteristic (ROC) curve (see [R] **roc**) by the equation $D = 2A - 1$, where D is Somers' D and A is the ROC area. Somers' D is therefore a performance indicator equivalent to the ROC area, but has the advantage that it is negative, positive or zero when the median difference between non-US and US cars is negative, positive or zero, respectively.

We start by loading the `auto` data. We then use `parmby` to call `somersd` and create in the memory a resultsset with one observation per Somers' D parameter and data on parameter attributes such as confidence intervals and P-values (with sensible display formats). We `describe` the variables in the resultsset, and `list` all observations, selecting the most interesting variables, which include `label`, a string variable containing, for each parameter, the variable label of the corresponding predictor variable. We then use `sencode` to encode this string variable to a numeric variable `predictor`, with variable labels, which are listed. The numeric codes are ordered in ascending order of the Somers' D estimates themselves by specifying the option `gsort(estimate parmseq)`, which orders codes first by Somers' D estimate, and breaks ties by the order of the predictor in the list. Next, we use `eclplot` to plot the confidence intervals for the Somers' D parameters, and save the plot in an Encapsulated PostScript file to form Figure 2. Negative predictors of non-US origin appear above the possible non-predictor `price`, which appears above positive predictors of non-US origin. Note that `eclplot` has a reverse default vertical axis scale if the user specifies a horizontal confidence interval plot.

We then produce a table. First, we use `sdecode` to convert the numeric confidence limits to string variables of the same names, use `replace` to add parentheses and commas to these string variables, and `list` the reformatted confidence intervals. Next, we use `listtex` to type the dataset to the Results window in an alien-looking style as follows:

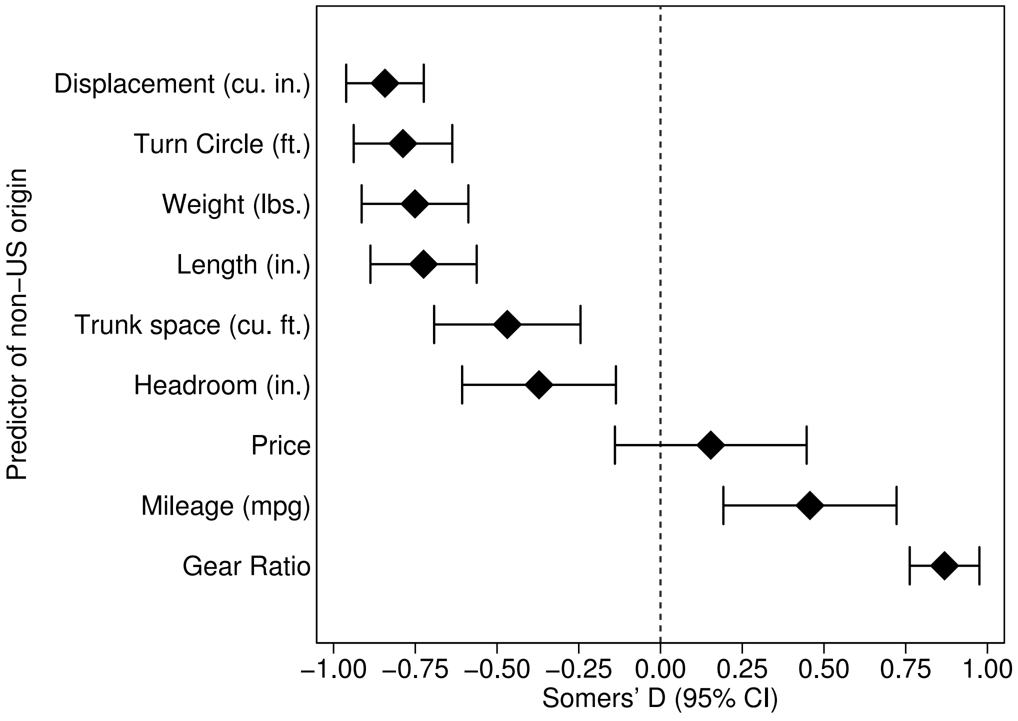
```
. listtex predictor estimate min95 max95, headline("Predictor&Somers' D&(95%&CI)") type
Predictor&Somers' D&(95%&CI)
Displacement (cu. in.)&-0.84&(-0.96,&-0.72)
Turn Circle (ft.) &-0.79&(-0.94,&-0.64)
Weight (lbs.)&-0.75&(-0.91,&-0.59)
Length (in.)&-0.72&(-0.89,&-0.56)
```

```
Trunk space (cu. ft.)&-0.47&(-0.69,&-0.24)
Headroom (in.)&-0.37&(-0.61,&-0.14)
Price&0.15&(-0.14,&0.45)
Mileage (mpg)&0.46&(0.19,&0.72)
Gear Ratio&0.87&(0.76,&0.98)

. more
```

The typed output (which appears bright yellow in the Results window) can then be copied from the Results window, pasted into a Microsoft Word document, selected inside Microsoft Word, and converted to a Microsoft Word table using the menu sequence `Table->Convert->Text to Table`. Alternatively, it can be pasted into a Microsoft Excel document and converted to Microsoft Excel columns using the menu sequence `Data->Text to Columns`. Note that `listtex` can also produce tables that can be cut and pasted into TEX, LATEX or HTML documents, usually with even less work, because Microsoft Word requires users to do their own table justification, autoformatting and italicizing.

Figure 2: Somers' D for quantitative diagnostic predictors of non-US origin in the `auto` data.



3.2 String-factor conversion using factext and factmerg

Often, the predictor variables in an estimation are dummy variables, indicating membership of a group defined by a value of a categorical variable. Such categorical variables are known as factors in the classical Rothamsted terminology, as used by the statistical packages GLIM and Genstat. The dummy variables are usually produced by `xi` (see [R] **xi**), but may also be produced by `tabulate` (see [R] **tabulate**), or by the `desmat` command of John Hendrickx (introduced in Hendrickx (1999), and updated in Hendrickx (2000, 2001a and 2001b)). Their variable labels are usually of the form

$$\text{"}\textit{factor\_name}\text{==}\textit{value}\text{"} \qquad (1)$$

(where *factor_name* is a valid Stata variable name and *value* is a value of the variable), and will be saved in `parmest` and `parmby` resultssets in the variable `label`, if the `label` option is specified. To produce plots like Figure 1, it would be useful to regenerate these categorical variables in the resultssets.

The `factext` program allows us to do this. It takes, as input, a list of string variables, which defaults to a single variable named `label`, as found in a `parmby` output. It generates, as output, a set of factors, with names extracted from the left-hand side of input string values of the form (1), and values extracted from the right-hand side of the same input string values. These output factors are usually numeric.

String values of the form (1) do not contain enough information to reconstruct a numeric factor complete with its storage type, display format, value labels, variable labels and characteristics (see [U] **15.8 characteristics** or [P] **char**). However, `factext` can be helped by `descsave`, which is not only a resultsset generator, but also a program generator. `descsave` has a `dofile()` option, which allows it to generate a do-file, which, if run, can reconstruct the variable attributes of the variables described, assuming that variables of the same name and mode (numeric or string) exist in the current dataset. `factext` also has a `dofile()` option, which allows it to run this do-file after extracting the factor values. The factors in the resultsset will then have the same storage types, display formats, value labels and variable labels as the factors of the same names in the input dataset, and also (optionally) the same values of characteristics specified by the user (such as the `omit` characteristic mentioned in [R] **xi**).

The `factext` program may cause a resultsset to be "wide" and "sparse" by generating a long list of factors, each of which may have missing values in most observations, because most of the parameters do not belong to dummies belonging to that factor. In Table 1 and Figure 1, by contrast, there is a single column of row or axis labels, containing information on both factors (`foreign` and `rep78`). The `factmerg` program converts in the opposite direction to `factext`, and can be useful in generating such row label variables. It takes, as input, a list of factors (usually numeric), and generates, as output, up to three string variables, containing, in each observation, the name, variable label and string value, respectively, of the first factor in the list with a nonmissing value. These string variables can be used in further processing to create row label variables for multi-factor plots and tables, such as Table 1 and Figure 1.

The program in `example4.do` demonstrates the use of `descsave`, `factext` and `factmerg`. After loading the `auto` data, we assign reference levels to the factors `foreign` and `rep78` using the `omit` characteristic, and use `descsave` to create a temporary do-file to reconstruct these factors. We then use `parmby` to replace the dataset in memory with a resultsset, with one observation per parameter of a regression model and data including a string variable `label`. Next, we use `factext` to extract the factors `foreign` and `rep78` from this variable, list them, and create a single-factor confidence interval plot for the differences between cars with non-reference and reference repair records, which is saved as Figure 3. After that, we use `factmerg` to merge the two factors and generate string variables `faclab` and `facval`, containing the variable label and value, respectively, of the factor corresponding to each parameter. We can now use `generate` and `sencode` to create a labelled numeric row variable `cargp`, identifying car groups with non-reference car origin type or repair record. Finally, we produce a multi-factor confidence interval plot for the differences in mileage associated with these non-reference car groups as Figure 4.

Figure 3: Differences in miles per gallon by repair record (compared to reference category 3).

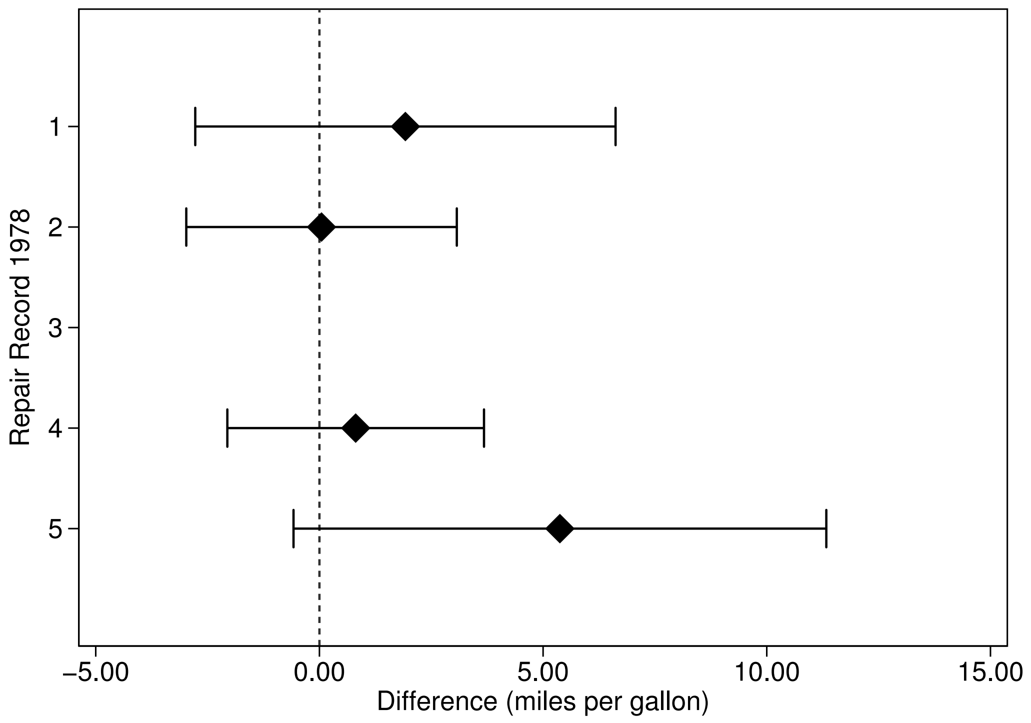
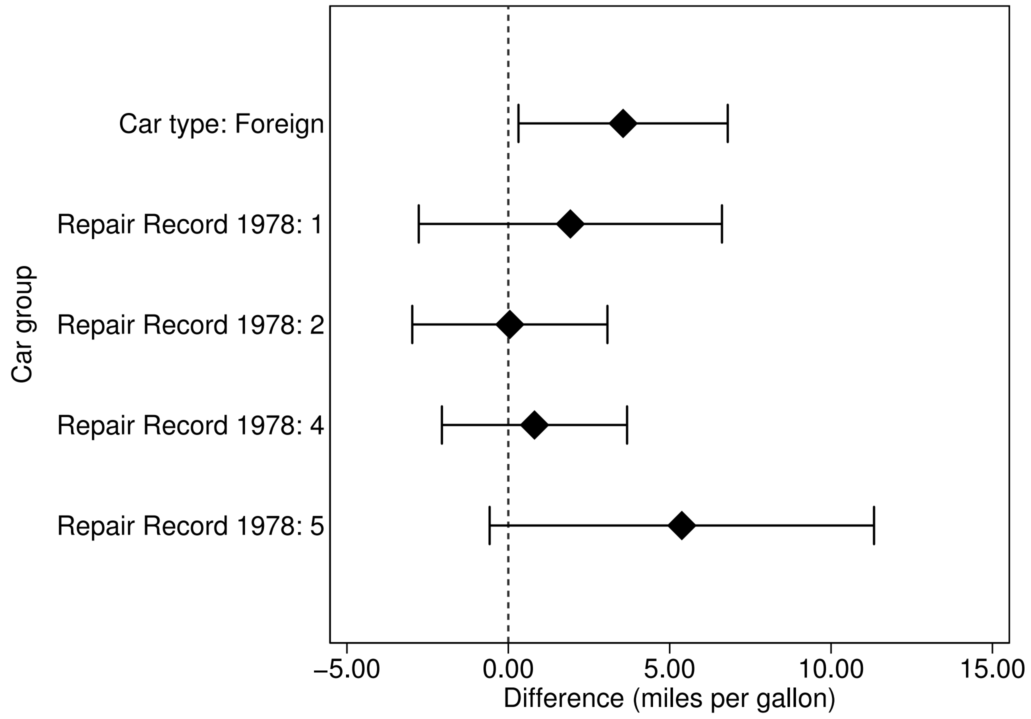


Figure 4: Differences in mileage in the `auto` data (compared to US car with `rep78==3`).



4 Adding reference and gap observations to resultssets

Figure 4 is not Figure 1. The reference categories are absent. The axis labels contain no group numbers. Also, there is little space for group numbers, because the labels are very repetitive, repeating factor labels instead of putting them in gaps on the axis. The first two problems can be solved by merging `xcontract` resultssets into the `parmby` resultsset. The third problem can be solved by adding gap observations to the resultsset, using the `ingap` package, downloadable from SSC.

The `ingap` package is discussed in Newson (2003). It adds gap observations to the dataset in memory, at a user-specified list of positions within the dataset, or within each by-group if by-groups are specified. Existing variables in the gap observations (apart from by-variables) are set to missing, unless the user specifies otherwise. Usually, one string variable is specified as a row or axis label variable by the `rowlabel()` option, and its values in the gap observations are specified by the `growlabel()` option or by the `grexpression()` option.

The program in `example5.do` is a more advanced version of `example4.do`, and uses `xcontract` and `ingap` to produce Figure 1 and Table 1. After creating the do-file using `descsave`, and before creating the `parmby` resultsset, we create two `xcontract` resultssets, containing the frequencies of `foreign` and `rep78` in cars with nonmissing repair record (included in the regression). After extracting the factors from the string variable `label`, we merge in these `xcontract` resultssets, and then set the confidence interval variables `estimate`, `min95` and `max95` to zero in observations with reference values for the two factors. After using `factmerge` to create `faclab` and `facval`, we generate the row variable `cargp`, adding gap observations using `ingap` in the process. In these gap observations, the value of `cargp` is set by the `grexpression()` option of `ingap`. We then plot the confidence intervals against `cargp` as before, this time creating a better-looking graph (Figure 1). Finally, we use `sdecode` and `replace` to convert the confidence limits to strings with parentheses and commas in the right places, and use `listtex` to list to the Results window a LaTeX `tabular` environment as follows:

```
. listtex cargp estimate min95 max95 p if !missing(factor), rstyle(tabular) type ///
>   headline("\begin{tabular}{rrrrl}" "\hline" ///
>     "\textit{Car group}&\textit{Difference (mpg)}&\textit{(95\%}&\textit{CI)}&\textit{P}\\\\" ///
>     "\hline") ///
>   footline("\hline" "\end{tabular}")
\begin{tabular}{rrrrl}
\hline
```

```
\textit{Car group}&\textit{Difference (mpg)}&\textit{(95\%}&\textit{CI)}&\textit{P}\\
\hline
Car type:&&&&\\
Domestic (N=48)&0.00&(ref.)&&\\
Foreign (N=21)&3.56&(0.32,&6.80)&.032    \\
Repair Record 1978:&&&&\\
1 (N=2)&1.92&($-$2.78,&6.62)&.42     \\
2 (N=8)&0.05&($-$2.98,&3.07)&.98     \\
3 (N=30)&0.00&(ref.)&&\\
4 (N=18)&0.81&($-$2.06,&3.68)&.57     \\
5 (N=11)&5.38&($-$0.58,&11.33)&.076    \\
\hline
\end{tabular}

. more
```

This typed output was cut and pasted into Table 1 of the LaTeX version of this document. The large program in `example5.do` may seem to be a very complicated way of creating a small plot and a small table. However, there are often economies of scale if the user needs to create large plots and tables, or a lot of plots and tables, or multiple versions of the same plot or table.

An alternative approach to drawing plots with gap labels is to use the `egen` function `axis()`, which is part of the `egenmore` package, written by Nicholas J. Cox and downloadable from SSC (see [R] **egen**). This approach has the feature that it does not add gap observations to the resultsset. This feature may or may not be an advantage, depending mainly on whether or not the user wishes to use the same resultsset as input both for `eclplot` and for `listtex`. It is often a good idea to use the `preserve` and `restore` commands (see [P] **preserve**) before and after a sequence of resultsset-modifying commands such as those in `example5.do`.

5 Acknowledgements

I would like to thank Nicholas J. Cox of Durham University, UK, for coining the term "resultsset" and for many helpful comments on this document; Christopher F. Baum of Boston College, MA, USA for distributing on SSC the resultsset-generating and resultsset-processing programs demonstrated here (and many more); and all the members of the Stata community who have given me their constructive advice on developing these programs.

6 References

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

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.

Hendrickx, J. 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.

Hendrickx, J. 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.

Hendrickx, J. 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. 2000. `somersd` – Confidence intervals for nonparametric statistics and their differences. *Stata Technical Bulletin* 55: 47–55. In *Stata Technical Bulletin Reprints*, vol. 10, 312–322. College Station, TX: Stata Press. Post-publication update downloadable from *http://www.kcl-phs.org.uk/rogernewson/* as of 20 April 2004.

Newson, R. 2002. Parameters behind "nonparametric" statistics: Kendall's tau, Somers' D and median differences. *The Stata Journal* 2(1): 45–64. Pre-publication draft downloadable from *http://www.kcl-phs.org.uk/rogernewson/* as of 20 April 2004.

Newson, R. 2003. Confidence intervals and p-values for delivery to the end user. *The Stata Journal* 3(3): 245–269. Pre-publication draft downloadable from *http://www.kcl-phs.org.uk/rogernewson/* as of 20 April 2004.